Monte–Carlo–Simulationen zum Phasen– und Keimbildungsverhalten von Polymerlösungen

DISSERTATION

ZUR ERLANGUNG DES GRADES
„DOKTOR DER NATURWISSENSCHAFTEN“
AM FACHBEREICH PHYSIK DER
JOHANNES GUTENBERG–UNIVERSITÄT
MAINZ

vorgelegt von

Peter Martin Virnau
geboren in Bad Dürkheim

Mainz, im Mai 2003

Datum der mündlichen Prüfung: 24. Juli 2003

Meinen Eltern

Peter Martin Virnau

Monte–Carlo–Simulationen zum Phasen– und Keimbildungsverhalten von Polymerlösungen

Koexistierende Phasen sind in der Regel durch eine große Barriere in der freien Energie getrennt. In dieser Arbeit werden zwei Ansätze verfolgt, die entscheidende Vorteile gegenüber etablierten Techniken aufweisen und eine Bearbeitung der physikalischen Fragestellungen überhaupt erst ermöglichen. Die erste Methode überträgt einen von *Wang* und *Landau* für Spin–Gittermodelle entwickelten Algorithmus auf Polymermischungen im Kontinuum und passt ihn an die speziellen Anforderungen des (μ, V, T)–Ensembles an. Bei der zweiten Methode unterteilt man die Simulation in mehrere Abschnitte, die nacheinander abgearbeitet werden. Dies ermöglicht eine Extrapolation der Wahrscheinlichkeitsverteilung in den darauf folgenden Abschnitt und somit die Erzeugung einer Gewichtsfunktion. Beide Methoden erlauben eine direkte Bestimmung der freien Energie an einer beliebigen Stelle des Phasendiagramms. Der zweite Ansatz ermöglicht darüber hinaus eine exakte Berechnung des Fehlers. Der Gesamtfehler ist dabei unabhängig von der Art der Unterteilung. Beide Simulationsverfahren können mit geringem Aufwand auf andere Systeme und Problemstellungen übertragen werden.

Das Phasenverhalten von Polymer–Lösungsmittel–Mischsystemen wird anhand eines vergröberten Modells für Hexadekan–CO_2 untersucht: CO_2 wird auf eine einzelne Lennard–Jones–Kugel abgebildet und Hexadekan auf eine Kette von fünf Teilchen. Zur Bestimmung der Parameter im Potential setzt man die kritischen Temperaturen von Simulation und Experiment gleich. Wechselwirkungen zwischen beiden Komponenten werden durch eine modifizierte Lorentz–Berthelot–Regel modelliert. Die Übereinstimmung des Phasenverhaltens und der Grenzflächeneigenschaften ist sehr gut. Insbesondere kann das Phasendiagramm des Mischsystems inklusive kritischer Linien mit nur einem freien Parameter reproduziert werden. Ein Vergleich mit numerischen Störungsrechnungen liefert eine qualitative Übereinstimmung und Hinweise zur Verbesserung der verwendeten Zustandsgleichung. Ferner wird ein Verfahren zur Bestimmung der Spinodalen mittels Simulationen getestet.

Computersimulationen von Keimbildungsphänomenen sind durch endliche Boxgrößen limitiert. Für das Lennard–Jones–System wird der Übergang vom homogenen Gas zu einem einzelnen Tropfen im endlichen Volumen direkt nachgewiesen und dessen Ordnung und Systemgrößenabhängigkeit bestimmt. Aufbauend auf diesen Betrachtungen wird die freie Energie von kleinen Clustern mit einem einfachen Modell untersucht und nach oben abgeschätzt. Untersuchungen zur Dynamik der Entmischung zeigen Benetzungseffekte an den Grenzflächen. Dies führt zu einer Absenkung der Nukleationsbarriere.

Inhaltsverzeichnis

Einleitung

Das Aufschäumen von Polymeren mit umweltfreundlichen Treibgasen wie Kohlenstoffdioxid ist von beträchtlichem wirtschaftlichen Interesse. Alle Herstellungsverfahren beruhen dabei auf dem gleichen Grundprinzip [1]. CO_2 wird zunächst unter großer Hitze und großem Druck in der Polymerschmelze gelöst und nach erfolgter Sättigung durch einen Drucksprung entspannt. Hierdurch wird eine Phasenseparation eingeleitet – ähnlich dem Öffnen einer Sprudelflasche. Das Material schäumt auf und verhärtet beim Unterschreiten der Glasübergangstemperatur, die durch die Anwesenheit von CO_2 zusätzlich beeinflusst wird [2]. Im Labor können so Zellgrößen im μm–Bereich realisiert werden [1]. Ein gutes Beispiel für die industrielle Nutzung dieses Verfahrens ist Styrodur® (Abb. 1). Der von der BASF AG in großer Menge produzierte Schaumstoff auf Polystyrolbasis wird seit über 30 Jahren erfolgreich z.B. bei der Wärmedämmung von Hauswänden eingesetzt.

Abbildung 1: (a) Zellstruktur von Styrodur®, (b) Einsatz von Styrodur® als Dämmstoff [3].

Zur Verbesserung der Materialeigenschaften ist eine genaue Kenntnis des Phasenverhaltens von entscheidender Bedeutung. Neben dem anwendungsbezogenen Interesse ist das Studium von Phasengleichgewichten seit den Arbeiten von *van der Waals* [4] eine der zentralen Aufgabestellungen der statistischen Physik. Ebenso wichtig für ein besseres Verständnis des Aufschäumverhaltens ist eine intensive Beschäftigung mit Keimbildungsprozessen, die auch in vielen anderen Bereichen der Natur auftreten. So ist das Ausperlen von Stickstoffmolekülen beim Auftauchen die Ursache der so genannten Taucherkrankheit [5]. Metallschäume dienen als Träger von Katalysatoren [6] und selbst in der Astrophysik sind Keimbildungskonzepte von Nutzen [7]. Bedauerlicherweise ist die Übereinstimmung von Theorie und Experiment immer noch von eher qualitativer Natur.

Computersimulationen eignen sich im Zusammenspiel mit Experimenten auf besondere Weise als Werkzeug zur Untersuchung dieser Art von Phänomenen. Das Phasenverhalten kann ohne aufwendige Apparaturen auch in Regionen bestimmt werden, die dem Experiment nur schwer oder gar nicht zugänglich sind. Zudem können durch einfache Modifikationen wie der Veränderung der Wechselwirkungseigenschaften qualitative Trends aufgezeigt werden. Zur gleichen Zeit benötigt man jedoch auch experimentelle Eingaben, um Wechselwirkungsparameter zu bestimmen und die Wirklichkeitsnähe des verwendeten Modells zu überprüfen. Die zur Simulation benötigten Techniken beruhen auf fortgeschrittenen Konzepten, die zum Teil in dieser Arbeit weiterentwickelt wurden. Bei der Keimbildung findet man eine vergleichbare Situation vor. Heutige Experimente ermöglichen, im Gegensatz zu Computersimulationen, meist keinen direkten Zugriff auf mikroskopische Details. Auch in diesem Bereich ergeben sich interessante technische Fragestellungen wie z.B. der Einfluss endlicher Simulationsvolumina auf das Nukleationsverhalten.

Die Doktorarbeit entstand im Rahmen eines Projektes mit der BASF AG, Ludwigshafen, an der sich auch die Arbeitsgruppen von Professor Strey (experimentelle physikalische Chemie in Köln) und von Professor Heermann (atomistische Molekulardynamik–Simulationen) beteiligten. Ein reales Polystyrol–CO_2–System lässt sich nur schwer mit Hilfe von Simulationen beschreiben. Zum Einen lassen sich die großen Ketten in der Schmelze mit heutigen Rechenkapazitäten nicht darstellen. Zum Anderen sind industriell gefertigte Polymere in der Regel polydispers. Die Ketten weisen also ein breites Spektrum unterschiedlicher Längen auf, was die Simulation zusätzlich erschweren würde. Im Rahmen des Projektes sollte deswegen das einfache Referenzsystem Hexadekan–CO_2 untersucht werden, um auf diese Weise einen besseren Einblick in das Phasen– und Keimbildungsverhalten von Polymer–Lösungsmittel–Mischungen zu erhalten.

Zunächst wurde ein vergröbertes Modell der beiden Einzelkomponenten erstellt. Dieser Ansatz basiert auf der Annahme, dass die statischen Gleichgewichtseigenschaften in erster Linie durch die grobe Struktur des Moleküls und nicht durch intramole-

kulare Wechselwirkungen bestimmt werden. Kapitel 1 stellt zudem die grundlegenden Monte–Carlo– und Analysetechniken vor. Kapitel 2 beschäftigt sich mit fortgeschrittenen Konzepten zur Bestimmung von Phasendiagrammen, die zum Teil im Rahmen der Dissertation entwickelt wurden. Ein besonderer Schwerpunkt liegt hierbei auf einem neu entwickelten Simulationsverfahren, dass die Überwindung von Barrieren in der freien Energie ohne vorherige Kenntnis von deren Höhe und Beschaffenheit ermöglicht. In Kapitel 3 werden Phasendiagramme und Grenzflächeneigenschaften der beiden Einzelkomponenten und des Mischsystems mit Experimenten verglichen. Dieser Test ist von essentieller Bedeutung für alle weiteren Untersuchungen, da nur so die Qualität des verwendeten Modells überprüft werden kann. Kapitel 4 vertieft die Betrachtungen zum Phasenverhalten durch einen Vergleich mit numerischen Störungsrechnungen, die im Rahmen einer Zusammenarbeit mit *L.G. MacDowell* entstanden. Kapitel 5 beginnt mit allgemeinen Betrachtungen zur Keimbildung in endlichen Systemen. Hierauf aufbauend wird die freie Energie von Clustern kleiner und mittlerer Größe bestimmt und mit einer einfachen Theorie verglichen. Die Arbeit schließt mit einem kurzen Ausblick auf die Kinetik von Entmischungsvorgängen.

Kapitel 1

Modell, Monte–Carlo–Techniken und Datenanalyse

Für klassische Simulationen verwendet man in der Regel eine kubische Simulationsbox mit Kantenlänge $L \approx 2.5 - 8.5\ nm$ (vgl. Abb. 1.1). In eine solche Box passen bis

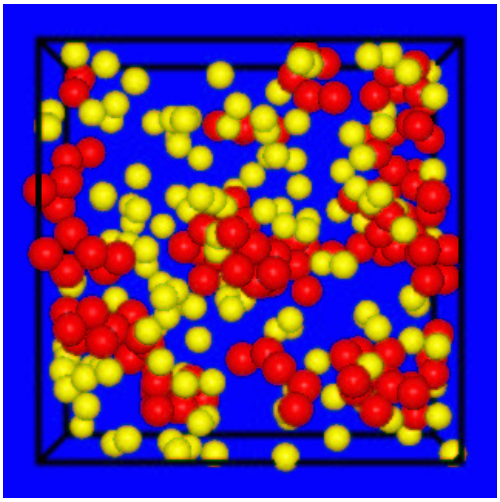

Abbildung 1.1: Schnappschuss einer typischen Konfiguration in der Gasphase. Polymere sind rot und Lösungsmittelteilchen gelb.

zu 1000 Polymere bzw. Lösungsmittelteilchen, die sich gemäß eines Monte–Carlo–Schemas (MC) bewegen. Verlässt ein Teilchen die Box durch eine Seite, so erscheint es wieder an der gegenüberliegenden (periodische Randbedingungen). Das statistische Verhalten entspricht daher im Wesentlichen dem von Teilchen eines sehr viel größeren makroskopischen System.

1.1 Modell

Hexadekan und CO_2 werden durch ein eigens für diese Arbeit entwickeltes, vergröbertes Modell dargestellt. Die Vergröberung führt zu einer signifikanten Reduzierung des Rechenaufwands und ermöglicht Simulationen in Regionen des Phasendiagramms, die sonst nicht zugänglich wären. In einem der denkbar einfachsten Ansätze wird CO_2 auf eine Kugel abgebildet, die über ein Lennard–Jones–Potential mit anderen Teilchen wechselwirkt. Da das Molekulargewicht von CO_2 (44 g/mol) in etwa um den Faktor Fünf kleiner ist als das von $C_{16}H_{34}$ (226 g/mol), liegt es nahe, Hexadekan auf eine Kette von fünf effektiven Monomeren abzubilden. Diese Teilchen sind frei verschiebbar und entsprechen jeweils etwa 3 CH_2–Gruppen. In einem realistischeren United–Atom–Modell (vgl. z.B. [8]) werden einzelne C und O Atome in CO_2 bzw. CH_2–Gruppen auf effektive Teilchen abgebildet, die in einem mehr oder weniger festen Bindungswinkel zueinander stehen und sich mittels eines Torsionspotentials gegeneinander verdrehen können. Wie Abbildung 1.2 zeigt, bewirkt dies einen großen Überlapp bei den Wechselwirkungsradien. Für die Wechselwirkungen zwischen verschiedenen Molekülen spielen diese intramolekularen Potentiale jedoch nur eine untergeordnete Rolle. Entscheidend ist vielmehr die ungefähre Struktur (Einhüllende) des Moleküls, die durch das vergröberte Modell gut nachgestellt wird.

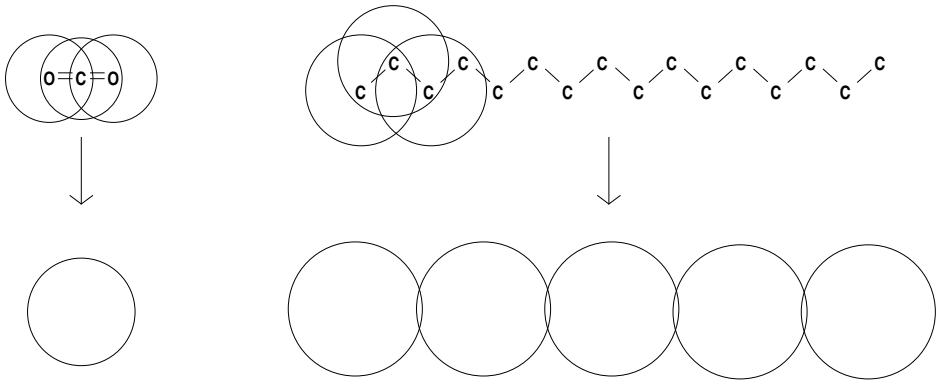

Abbildung 1.2: Modellierung: $CO_2 \rightarrow$ Monomer, $C_{16}H_{34} \rightarrow$ Fünfmer. Die Durchmesser der Kugeln entsprechen jeweils σ (dem Nullpunkt des Potentials). Die Werte für das United–Atom–Modell stammen aus [9], die Werte des vergröberten Modells sind Kapitel 3.1 entnommen. Aus Gründen der Übersichtlichkeit wird Hexadekan gestreckt dargestellt.

Zwischen zwei effektiven Kugeln wirkt ein angehobenes und abgeschnittenes Lennard–Jones–Potential der Form:

$$V_{\mathrm{LJ}} = \begin{cases} 4\varepsilon \cdot \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + \frac{127}{16384} \right] & \text{, falls} \quad r < 2 \cdot \sqrt[6]{2}\sigma \\ 0 & \text{, sonst.} \end{cases} \tag{1.1}$$

Der r^{-6} Term beschreibt attraktive Van–der–Waals–Wechselwirkungen. Bei einzelnen Atomen bilden Kern und Hülle einen fluktuierenden Dipol, der mit seinem Nachbarn wechselwirkt. Grob gesprochen ist das elektrische Feld proportional zu r^{-3} und die hieraus abgeleitete Wechselwirkungsenergie proportional zum Quadrat des Feldes, also zu r^{-6}. Eine quantenmechanische Ableitung (Störungsrechnung zweiter Ordnung) wurde zum ersten Mal von *London* durchgeführt [10]. Geraten zwei Teilchen zu nahe aneinander, stoßen sie sich ab. Im Lennard–Jones–Potential wird dies ebenfalls über ein Potenzgesetz realisiert. Der exakte Exponent ist jedoch theoretisch weniger gut fundiert. Für $r = \sigma$ ist $V_{\mathrm{LJ}} = 0$. Das Minimum des Potentials liegt bei $2^{1/6}\,\sigma \approx 1.12\,\sigma$. An dieser Stelle ist $V_{\mathrm{LJ}} \approx -\varepsilon(1-0.031)$. Der konstante Anteil ist in dieser Arbeit so gewählt, dass das Potential bei der zweifachen Länge des Minimums Null wird. Dies hat zwar eine Verschiebung des kritischen Punktes zur Folge [11], erspart jedoch Rechenzeit und Korrekturen für das Abschneiden des langreichweitigen Potentialschwanzes. σ und ε normieren Längen– und Energieskalen und ermöglichen so eine Abbildung auf reale Substanzen (vgl. Kapitel 3.1). σ ist ein Maß für die „Breite“ des Potentials bzw. der Bindungslänge, während ε die Potentialtiefe definiert (vgl. Abb. 1.3). Da sich die Wechselwirkungen zwischen zwei CO_2–Kugeln von Wechselwirkungen zweier Hexadekan–Kugeln unterscheiden, ergeben sich aus dem Vergleich mit dem Experiment jeweils unterschiedliche Werte.

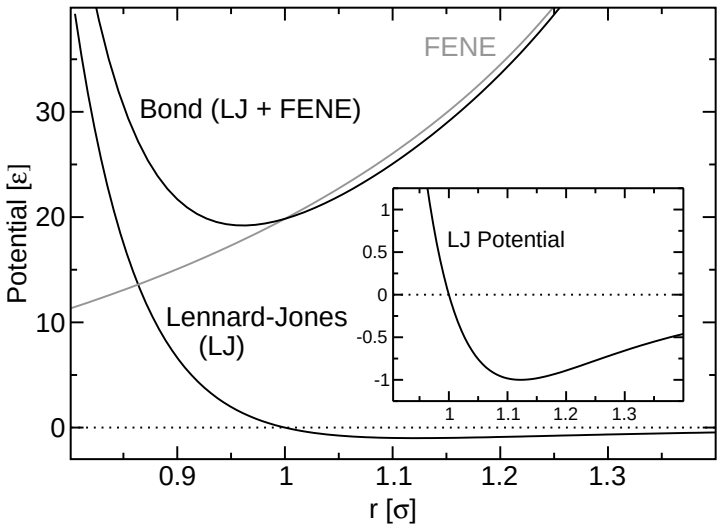

Abbildung 1.3: LJ– und FENE–Potentiale. Benachbarte Kugeln auf einer Kette bevorzugen einen Abstand von 0.96 σ, unverknüpfte Kugeln einen Abstand von 1.12 σ. Die Einführung der zweiten Längenskala unterdrückt bei den Polymeren die Kristallisation bei tiefen Temperaturen.

Benachbarte Teilchen auf einer Polymerkette erhalten einen zusätzlichen FENE–

Anteil („**f**inite **e**xtentible **n**onlinear **e**lastic“) zum Potential (vgl. Abb. 1.3):

$$V_{\mathrm{FENE}} = -33.75\varepsilon \cdot \ln\left[1 - \left(\frac{r}{1.5\sigma}\right)^2\right] \tag{1.2}$$

Die Vorfaktoren gewährleisten eine gewisse Steifigkeit und verhindern somit ein Durchtrennen von Ketten in der Simulation. Der bevorzugte Abstand des aus Lennard–Jones– und FENE–Wechselwirkung zusammengesetzten Potentials (0.96 σ) ist etwas geringer als für ungebundene Teilchen (1.12 σ). Die Einführung einer zweiten Längenskala unterdrückt bei den Polymeren die Kristallisation bei tiefen Temperaturen. [12]. Die Ursprünge dieser Art von Kugel–Feder–Modellen gehen auf *Kremer* und *Grest* zurück [13].

Lennard–Jones–Wechselwirkungen zwischen einer CO_2– und einer Hexadekan–Kugel in der Mischung können z.B. mit Hilfe einer modifizierten Lorentz–Berthelot–Regel bestimmt werden [14], [15], [16]:

$$\sigma_{1-5} = \frac{\sigma_1 + \sigma_5}{2} \tag{1.3}$$

$$\varepsilon_{1-5} = \xi \cdot \sqrt{\varepsilon_1 \varepsilon_5} \tag{1.4}$$

Betrachtet man z.B. eine Mischung von harten Kugeln unterschiedlicher Größe, so wird der „Wechselwirkungsradius“ gerade durch das arithmetische Mittel der beiden einzelnen Radien definiert (1.3). Der energetische Wechselwirkungsparameter ε sollte bei einer Mischung zweier Lennard–Jones–Kugeln A und B im Grenzfall A=B gerade wieder den ursprünglichen Wert ergeben. Zudem ist $\varepsilon_{\mathrm{A,B}}$ proportional zum Quadrat der jeweiligen Polarisierbarkeiten [16]. Die Mischregeln sollten lediglich als Anhaltspunkt verstanden werden. Gerade bei Polymer–Lösungsmittel–Mischungen hat es sich eingebürgert, einen Fitparameter $\xi < 1$ in Gleichung (1.4) einzuführen , der Abweichungen von der Regel ermöglicht [16]. So sind sich Polymer– und (ungeladene) Lösungsmittelteilchen im hier vorgestellten Modell ähnlicher als in Wirklichkeit, da CO_2 Partialladungen und hieraus resultierende Multipolmomente aufweist. Die modifizierte Lorentz–Berthelot–Regel ist in der physikalischen Chemie aufgrund ihrer Einfachheit weit verbreitet. Es gibt jedoch durchaus auch komplexere Ansätze [16], [17]. Ein Vergleich diverser Mischregeln findet sich z.B. in [9].

Das hier vorgeschlagene Modell wird im weiteren Verlauf der Arbeit sowohl für reines CO_2 und Hexadekan (Kapitel 3.2) als auch für das Mischsystem (Kapitel 3.3) ausführlich getestet und mit Experimenten verglichen. Die Übereinstimmung ist, soviel sei vorweggenommen, sehr gut.

1.2 Grundlagen einer Monte–Carlo–Simulation

Eine zentrale Fragestellung in der statistischen Physik ist die Bestimmung von Erwartungswerten einer Observablen O in einem wohldefinierten Ensemble. Da die Anzahl aller möglichen Konfigurationen mit steigender Teilchenzahl schnell sehr groß wird, führt eine einfache Mittelung über alle Zustände nicht zum Erfolg. Würde man z.B. eine Simulationsbox entlang jeder Achse in lediglich zehn Abschnitte unterteilen, erhält man bei zehn Teilchen in $d = 3$ Dimensionen etwa 10^{30} Möglichkeiten. Selbst wenn alle Teilchen gleichartig sind und man lediglich unterschiedliche Konfigurationen berücksichtigt, liegt deren Gesamtzahl immer noch bei $2.6 \cdot 10^{23}$ und somit jenseits jeglicher heute zugänglicher Rechenleistung. Die Bestimmung eines Erwartungswertes muss folglich durch eine Mittelung über eine endliche Auswahl von Zuständen erfolgen. Da der mit Abstand größte Teil der Konfigurationen lediglich einen vernachlässigbaren Beitrag zum Mittelwert liefert, führt eine zufällige Auswahl („random sampling“) nicht zum Erfolg. Vielmehr sollte das statistische Gewicht eines Zustand S_i berücksichtigt werden, welches in einem klassischen System durch die Boltzmannverteilung gegeben ist. Im kanonischen Ensemble ist:

$$P(S_i) = \frac{e^{-\beta U(S_i)}}{Z}. \tag{1.5}$$

U entspricht der Energie der Konfiguration und Z der kanonischen Zustandssumme. Der kinetische Anteil spielt bei den folgenden Betrachtungen keine Rolle und kann z.B. in Z hineingezogen werden. Die Monte–Carlo–Simulation versucht nun Konfigurationen zu erzeugen, die einer vorgegebenen Wahrscheinlichkeitsverteilung (z.B. (1.5)) genügen. Der im folgenden Abschnitt vorgestellte Algorithmus („importance sampling“) wurde 1953 von *Metropolis et al.* veröffentlicht [18]. Die hierauf aufbauenden Monte–Carlo–Techniken [19], [20] bilden zusammen mit der Molekulardynamik–Methode (MD) die Grundlagen klassischer molekularer Simulation. Die Darstellung des Algorithmus folgt im Wesentlichen [19].

Zustände, die einer gegebenen Verteilung genügen, können mit Hilfe einer Markov–Kette realisiert werden. Hierzu definiere man einen stochastischen Prozess mit diskreten Zeitschritten $t_1, t_2, \ldots$ für ein System mit einer endlichen Anzahl von Zuständen $S_1, S_2, \ldots$. X_t sei der Zustand eines Systems zum Zeitpunkt t. Ferner betrachte man:

$$P(X_{t_n} = S_{i_n} | X_{t_{n-1}} = S_{i_{n-1}}, X_{t_{n-2}} = S_{i_{n-2}}, \ldots, X_{t_1} = S_{i_1}), \tag{1.6}$$

also die bedingte Wahrscheinlichkeit, dass sich das momentane System ($X_{t_n = S_{in}}$) zum Zeitpunkt t_{n-1} im Zustand S_{n-1}, zum Zeitpunkt t_{n-2} im Zustand S_{n-2} usw. befand. Eine Abfolge von Zuständen $\{X_t\}$ bezeichnet man als Markov–Kette, falls die bedingte Wahrscheinlichkeit das System zum Zeitpunkt t_n im Zustand S_i anzutreffen lediglich vom direkten Vorgängerzustand $X_{t_{n-1}}$ abhängt:

$$P = P(X_{t_n} = S_{i_n} | X_{t_{n-1}} = S_{i_{n-1}}) =: W_{i_{n-1} i_n} = W(S_{i_{n-1}} \to S_{i_n}). \tag{1.7}$$

Die „Vergangenheit“ vor t_{n-1} hat also keinen Einfluss auf die Eintrittswahrscheinlichkeit von X_{t_n} und die weitere Entwicklung der Kette. Wie in Gleichung (1.7) angedeutet, lässt sich P auch als Übergangswahrscheinlichkeit W_{ij} vom Zustand i nach j interpretieren. Ist diese unabhängig von t, bezeichnet man den Prozess als stationär. Als Sprungwahrscheinlichkeit ist W_{ij} immer größer Null. Die Wahrscheinlichkeit von i in einen beliebigen Zustand j zu springen und somit $\sum_j W_{ij}$ ist gleich Eins.

Die so genannte „master equation“ betrachtet die Änderung der bedingten Wahrscheinlichkeit als Funktion der Zeit. Der Fluss von einem Zustand i in einen Zustand j ist gegeben durch das Produkt aus der Wahrscheinlichkeit $P_i := P(X_{t_n} = S_i)$, dass sich das System zum Zeitpunkt t im Zustand i befindet, mit der Sprungwahrscheinlichkeit W_{ij} von i nach j. Summiert man über alle Zustände i, erhält man den gesamten Zufluss nach j. Der Abfluss aus j ist entsprechend gegeben durch die Summe über alle Produkte $P_j W_{ji}$. Insgesamt erhält man also

$$\frac{dP_j(t)}{dt} = \sum_i P_i(t) W_{ij} - \sum_j P_j(t) W_{ji}. \tag{1.8}$$

Gleichung 1.8 kann als „Kontinuitätsgleichung“ aufgefasst werden. Die Gesamtwahrscheinlichkeit ($\sum_j P_j$) bleibt zu jedem Zeitpunkt erhalten. Im Gleichgewichtszustand ist der Zufluss in einen Zustand gleich dem Abfluss aus dem Zustand, d.h. $dP_j/dt = 0$. Diese Bedingung ist auf jeden Fall erfüllt, falls gilt:

$$P_i(t) W_{ij} = P_j(t) W_{ji}. \tag{1.9}$$

Die strengere Forderung, dass zwei beliebige Zustände zueinander im Gleichgewicht stehen, bezeichnet man als „detailed balance“.

Die folgende Diskussion bezieht sich auf das kanonische Ensemble. Obwohl die Wahrscheinlichkeit in Gleichung (1.5) wegen der Zustandsumme im Nenner in der Regel nicht genau bestimmbar ist, kann man dennoch das Verhältnis zweier Wahrscheinlichkeiten angeben. Z kürzt sich dann in Gleichung (1.9) weg und es wird lediglich die Differenz in der Energie der beiden Zustände $\Delta U = U_j - U_i$ benötigt. Jede Übergangsrate, die die Detailed–Balance–Bedingung erfüllt, ist prinzipiell akzeptabel. Das erste solche Kriterium, welches in der statistischen Physik bei der Simulation harter Scheiben Anwendung fand, stammt von *Metropolis et al.* [18]:

$$W_{ij} = \begin{cases} \exp(-\beta \Delta U) & \text{, falls} \quad \Delta U > 0 \\ 1 & \text{, falls} \quad \Delta U < 0. \end{cases} \tag{1.10}$$

oder kürzer:

$$W_{ij} = \min(1, \exp(-\beta \Delta U)) =: \text{metrop}\left(\exp(-\beta \Delta U)\right). \tag{1.11}$$

Ein Sprung auf ein energetisch tieferes Niveau wird demnach immer akzeptiert. Ein Sprung auf ein höheres Niveau jedoch nur mit einer exponentiellen Wahrscheinlichkeit. Diese wird umso geringer, je größer die Energiezunahme ausfällt. Die Überprüfung der Detailed–Balance–Bedingung erfolgt durch das Einsetzen von Bedingung (1.10) in (1.9). Der Metropolis–Algorithmus kann auf einfache Weise implementiert werden. Das System befinde sich hierzu im Ausgangszustand i:

|: 1. Wähle zufällig einen neuen Zustand j aus.
2. Berechne die Energiedifferenz ΔU zwischen j und dem alten Zustand i.
3. Ziehe eine Zufallszahl z aus [0,1].
4. Falls $z < \exp(-\beta\Delta U)$, akzeptiere den Schritt und führe das System in den neuen Zustand j über. :|

In einer typischen Simulation wird die obere Abfolge sehr oft wiederholt. Nach einer kurzen Relaxationsphase in der sich das System ins Gleichgewicht bewegt, können Messungen einer Observablen vorgenommen werden. Da die erzeugten Zustände bereits der Boltzmannverteilung genügen, berechnet sich der Erwartungswert $\langle O \rangle = \sum_i P_i O_i$, einfach aus dem arithmetischen Mittel.

Im Vergleich mit einer Molekulardynamik–Simulation hat Monte Carlo den großen Vorteil, dass Sprünge von einem Zustand in den nächsten beliebig „unphysikalisch“ ausfallen können. Statistisch unabhängige Konfigurationen können so auf Kosten einer unrealistischen Dynamik sehr viel schneller erzeugt werden. Im Folgenden sollen die in dieser Arbeit verwendeten Übergänge kurz erläutert werden.

1.2.1 Lokale Verrückungen

Eine der einfachsten und ältesten Übergänge ist die lokale Verrückung eines Polymer– oder Lösungsmittelteilchens [18]. Hierbei wählt man zufällig ein Teilchen im System aus und verschiebt es um eine bestimmt Länge und Orientierung (Abb. 1.4). Im Anschluss wird die durch die Verrückung verursachte Energiedifferenz berechnet und der

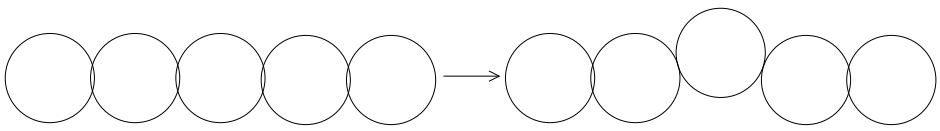

Abbildung 1.4: Schematische Darstellung eines lokalen Monte–Carlo–Schrittes.

Zug gemäß des Metropolis–Kriteriums akzeptiert oder abgelehnt. Der lokale Übergang ist zur Äquilibrierung einer kanonischen Konfiguration (konstante Teilchenzahl in der Box) geringer oder mittlerer Dichte nur bedingt geeignet. In einer dichten Phase ist er hingegen einer der wenigen Schritte, die überhaupt akzeptiert werden.

1.2.2 Reptation

Beim Reptationsalgorithmus [21], [22], der manchmal auch als Slithering–Snake–Algorithmus bezeichnet wird, schneidet man ein Endmonomer einer zufällig ausgewählten Kette ab und fügt es an der gegenüberliegenden Seite wieder an (Abb. 1.5). Dies bewirkt eine „schlangenartige“ Kriechbewegung des Polymers. Die Bindungs-

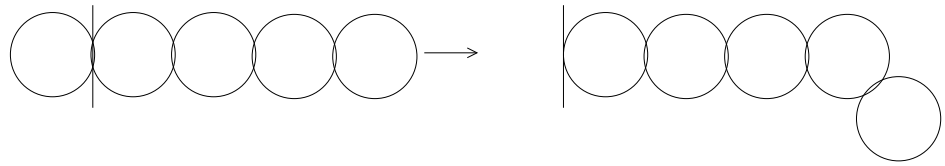

Abbildung 1.5: Schematische Darstellung eines Reptationsschrittes.

länge bleibt dabei in der einfachsten Ausführung erhalten. Die Implementierung des Algorithmus erfolgt wie bei der lokalen Verrückung über das Metropolis–Kriterium. Der Slithering–Snake–Algorithmus ist ein gutes Beispiel für einen unphysikalischen Monte–Carlo–Schritt. Er ermöglicht jedoch bei geringen und mittleren Dichten eine wesentlich effizientere Äquilibrierung eines Kettenmoleküls als die lokale Verrückung.

1.2.3 Großkanonisches „configurational–bias“

Bei der lokalen Verrückung und der Reptation handelt es sich um kanonische Monte–Carlo–Schritte, die die Anzahl der Polymere in der Box unverändert lassen. Der mit Abstand größte Teil der Simulationen in dieser Arbeit findet jedoch im großkanonischen (μ, V, T)–Ensemble statt, d.h. das chemische Potential μ, das Volumen V und die Temperatur T bleiben konstant während die Dichte fluktuiert. Nun ist es nahe liegend einfach eine zufällig gewachsene Kette in die Konfiguration einzusetzen bzw. zu entnehmen. Selbst in Systemen geringer Dichte führt jedoch ein einziger größerer Überlapp eines (zufällig) gewachsenen Monomers mit seinem Vorgänger aufgrund der großen Differenz in der potentiellen Energie zu einer Ablehnung des gesamten Schrittes. Bei höheren Dichten ist die Akzeptanzrate für alle praktischen Zwecke gleich Null. Großkanonisches „configurational–bias“ [23] löst dieses Problem auf elegante Weise

indem es die Umgebung eines Monomers nach geeigneten Lücken absucht und das Polymer in eine energetisch günstige Richtung weiterwachsen lässt (vgl. Abb. 1.6). Diese Ausrichtung muss jedoch in der Monte–Carlo–Abfrage berücksichtigt werden.

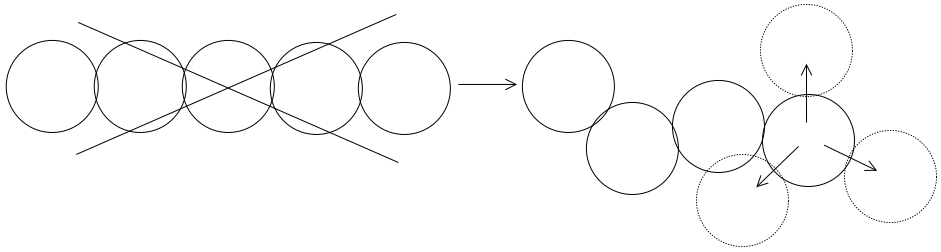

Abbildung 1.6: Schematische Darstellung eines großkanonischen Configurational–Bias–Schrittes.

Zunächst soll die Vorgehensweise beim Einsetzen eines Polymers der Länge l in eine Box mit n Polymeren beschrieben werden:

1. Setze das erste Monomer zufällig ein.
   Berechne das Rosenbluthgewicht [24] des ersten Monomers:
   $w_1 = \exp(-\beta U_1)$.
   U_1 beschreibt die Lennard–Jones–Wechselwirkungen (oder allgemein alle nicht durch Bindungen verursachten Wechselwirkungen) des ersten Monomers.

2. |: Generiere t Positionen $\{\mathbf{b_1}, ..., \mathbf{b_t}\}$ um das Monomer im gleichen boltzmannverteilten Abstand:
   $P(r = r_0) \propto \int P(r_x, r_y, r_z)\delta(r = r_0)\mathrm{d}r_x\mathrm{d}r_y\mathrm{d}r_z \propto r_0^2 \exp(-\beta U_{\mathrm{FENE}}(r_0))$.
   Z_{FENE} normiert die P(r)–Verteilung (Fläche unter P(r)).

   Wähle eine der Positionen $\mathbf{b}^*$ gemäß ihres Boltzmanngewichtes aus:
   $P(\mathbf{b}^*) = \frac{\exp(-\beta U_i(\mathbf{b}^*))}{w_i}$ mit $w_i = \sum_{j=1}^{t} \exp(-\beta U_i(\mathbf{b_j}))$.
   $\mathbf{b}^*$ wird das i–te Segment der Kette ($i > 1$) . :|

3. Berechne den Rosenbluthfaktor der gesamten Kette: $R_{\mathrm{neu}} = \prod_{i=1}^{l} w_i$
   und akzeptiere die Kette mit Wahrscheinlichkeit:
   $W_{n,n+1} = \mathrm{metrop}\left(\frac{\mathrm{R_{neu}}}{n+1} \exp(\beta\mu')\right)$.

Die Vorschlagswahrscheinlichkeit für eine derartige Kette beträgt:

$$P_{\text{vor}}^{+} = \frac{1}{2V}\prod_{i=1}^{l-1}\left(\frac{e^{-\beta U_{\text{FENE,i}}}}{Z_{\text{FENE}}}\,\frac{e^{(-\beta U_i(\mathbf{b}^*))}}{w_i}\right) = \frac{1}{2V}\frac{1}{Z_{\text{FENE}}^{l-1}}\frac{e^{-\beta U_{\text{Kette}}}}{R_{\text{neu}}}. \tag{1.12}$$

Einsetzen und Entfernen von Ketten wird mit jeweils 50 Prozent vorgeschlagen, daher der Faktor 1/2. 1/V ist proportional zur Wahrscheinlichkeit das erste Monomer einzusetzen. Der Ausdruck in der Klammer entspricht der Wahrscheinlichkeit, ein darauf folgendes Monomer in einem bestimmten Abstand zum Vorgänger aus den t Versuchpositionen zu erzeugen. U_{Kette} beschreibt die Energiezunahme durch das Einsetzen. Der Abbau einer Kette aus einem Zustand mit $n+1$ Polymeren verläuft analog:

1. Wähle zufällig eine Kette aus.

2. |: Generiere $t-1$ Positionen um das Endmonomer der Kette. Der Abstand entspricht der Entfernung des Monomers zu seinem Vorgänger. Zusammen mit dem Vorgänger am Ort $\mathbf{b}^*$, erhält man so t Positionen $\{\mathbf{b_1},...,\mathbf{b_t}\}$.
Berechne den Rosenbluthfaktor: $R_i = \sum_{j=1}^{t}\exp(-\beta U_i(\mathbf{b_j}))$.
und entferne das Endmonomer. :|

3. Der Rosenbluthfaktor des letzten Monomers beträgt: $R_i = \exp(-\beta U_i)$.
Berechne den Rosenbluthfaktor der gesamten Kette: $R_{\text{alt}} = \prod_{i=1}^{l} w_i$
Die Akzeptanzrate zum Entfernen der Kette wird definiert als:
$W_{n+1,n} = \text{metrop}\left(\frac{n+1}{R_{\text{alt}}}\exp(-\beta\mu')\right)$.

Die Vorschlagswahrscheinlichkeit eine bestimmte Kette aus einer Konfiguration von $n+1$ Ketten auszuwählen ist gegeben durch:

$$P_{\text{vor}}^{-} = \frac{1}{2}\,\frac{1}{n+1}. \tag{1.13}$$

Zum Schluss muss noch gezeigt werden, dass der Algorithmus die Detailed–Balance–Bedingung (1.9) erfüllt. Hierzu betrachtet man zwei Konfiguration A,B mit n und $n+1$ Teilchen. B unterscheide sich von A durch das zusätzliche Polymer. Beide stehen im Gleichgewicht zueinander, falls gilt:

$$\frac{P(n+1)}{P(n)} = e^{-\beta U_{\text{Kette}}+\beta\mu}. \tag{1.14}$$

A unterscheide sich nun von B' durch das zusätzliche Polymer und den dazugehörigen Satz von Versuchpositionen. Fordert man, dass auch diese beiden Zustände im

Gleichgewicht zueinander stehen („super–detailed balance“), so ist „detailed balance“ in jedem Fall erfüllt:

$$P(A)W_{A,B} = \sum_{B'} P(A)P^{+}_{\mathrm{vor}}W_{A,B'} = \sum_{B'} P(B)P^{-}_{\mathrm{vor}}W_{B',A} = P(B)W_{B,A}. \tag{1.15}$$

Es gelte also „super-detailed balance“, d.h.

$$\frac{P(n+1)}{P(n)} = \frac{P^{+}_{\mathrm{vor}}\, W_{\mathrm{n,n+1}}}{P^{-}_{\mathrm{vor}}\, W_{\mathrm{n+1,n}}}. \tag{1.16}$$

Mit $\frac{\mathrm{metrop(x)}}{\mathrm{metrop}(1/\mathrm{x})} = x$ und $R_{\mathrm{alt}} = R_{\mathrm{neu}}$ folgt:

$$\frac{P(n+1)}{P(n)} = e^{-\beta U_{\mathrm{Kette}}+\beta\mu'}\frac{1}{VZ^{l-1}}. \tag{1.17}$$

Damit (1.17) Gleichung (1.14) genügt, muss das chemische Potential wie folgt modifiziert werden:

$$\mu' = \mu + kT\ln V + kT(l-1)\ln Z_{\mathrm{FENE},1}. \tag{1.18}$$

Beim Mischsystem simuliert man im (μ_1, μ_2, V, T)–Ensemble und benötigt ein zweites chemisches Potential:

$$\mu_2' = \mu_2 + kT\ln V + kT(l_2-1)\ln Z_{\mathrm{FENE},2}. \tag{1.19}$$

1.3 Regel der gleichen Flächen und Histogramm–Extrapolation

Während einer großkanonischen Simulation schreibt man die momentane Teilchenzahl in der Box, die Energie des Systems und nach Bedarf weitere Werte in regelmäßigen Abständen in eine Datei. Nach der Simulation kann hieraus ein Histogramm erstellt werden (Abb. 1.7). Liegt das chemische Potential der Simulation in der Nähe des chemischen Potentials bei Koexistenz, erhält man zwei Maxima. Der Peak bei geringer Teilchenzahl entspricht der Gasphase, der Peak bei hoher Teilchenzahl der flüssigen Phase. Beide Phasen koexistieren, wenn die Flächen $A_{\mathrm{I,II}}$ unter den Maxima gleich groß werden. Die Box ist dann die Hälfte der Zeit mit Gas und die andere Hälfte der Zeit mit Flüssigkeit gefüllt. Diese so genannte „Regel der gleichen Flächen“ [25], [26] kann auf einfache Weise begründet werden:

$$A_{\mathrm{I}} = \sum_{n\le\langle n\rangle} P(n) = \frac{Z_{\mathrm{I}}}{Z_{\mathrm{I}}+Z_{\mathrm{II}}} \ \text{bzw.}\ A_{\mathrm{II}} = \frac{Z_{\mathrm{II}}}{Z_{\mathrm{I}}+Z_{\mathrm{II}}}, \tag{1.20}$$

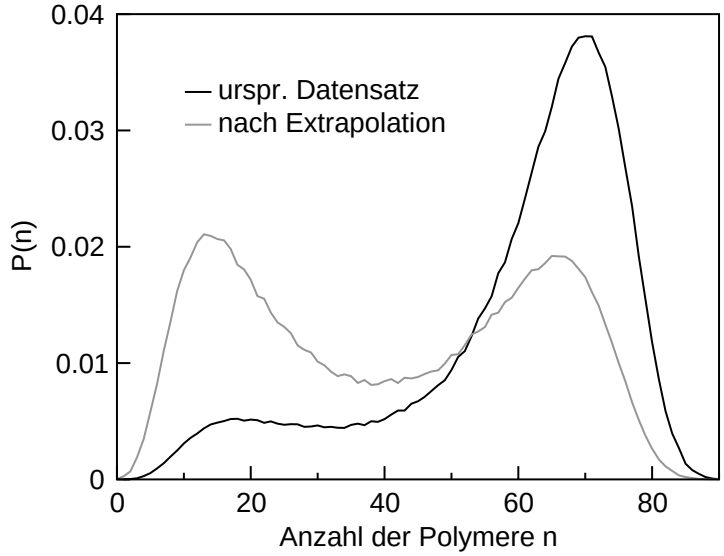

Abbildung 1.7: Typische Wahrscheinlichkeitsverteilung einer großkanonischen Simulation am kritischen Punkt des Fünfmer–Systems vor und nach der Extrapolation ($T = 1.725\ \frac{\varepsilon_5}{k}$, $L = 9\ \sigma_5$). Vergleiche mit Kapitel 1.4.

$Z_{\mathrm{I,II}}$ steht hierbei für die großkanonische Zustandssumme, die auf den Bereich $n \leq \langle n \rangle$ bei Z_{I} bzw. $n > \langle n \rangle$ bei Z_{II} eingeschränkt ist. Im Gleichgewicht sind die Drücke in beiden Phasen gleich, d.h. $p_{\mathrm{I}} = p_{\mathrm{II}}$. Mit $\Omega_{\mathrm{I,II}} = -p_{\mathrm{I,II}}V$ und $\Omega_{\mathrm{I,II}} = -kT \ln Z_{\mathrm{I,II}}$ folgt $Z_{\mathrm{I}} = Z_{\mathrm{II}}$ und somit Flächengleichheit nach (1.20).

Abbildung 1.7 zeigt ein typisches Simulationsergebnis. Zwar wurden Zustände im Gas– und Flüssigkeitsbereich erzeugt, das chemische Potential der Simulation entspricht jedoch nicht ganz dem chemischen Potential bei Koexistenz. Im Folgenden soll gezeigt werden wie eine Wahrscheinlichkeitsverteilung $P(n)$, die durch eine großkanonische Simulation bei chemischem Potential μ und Temperatur T erzeugt wurde, zu μ' und T' extrapoliert werden kann [27]. Die Wahrscheinlichkeit eine bestimmte Konfiguration S bei μ' und T' zu erhalten ist gegeben durch:

$$P_{\mu',T'}(S) \propto \frac{1}{Z'}\, e^{-\beta' U + \mu'\beta' n} = \frac{Z}{Z'} P_{\mu,T}(S)\, e^{-(\beta'-\beta)U + (\mu'\beta' - \mu\beta)n}. \tag{1.21}$$

$P_{\mu',T'}(n_0)$ erhält man durch Aufsummieren aller Konfigurationen mit Teilchenzahl n_0. Dabei ist $\sum_{\{S_i\}|n=n_0} P_{\mu,T}(S_i) \propto \sum_{\{S_i\}} \delta(n - n_0)$ und kann aus dem ursprünglichen Datensatz bestimmt werden. Nach einer geeigneten Normierung entfallen die Proportionalitätskonstanten und man erhält:

$$P_{\mu',T'}(n_0) = \frac{\sum_{\{S_i\}} \delta(n - n_0)\, e^{-(\beta'-\beta)U_i + (\mu'\beta' - \mu\beta)n_i}}{\sum_{\{S_i\}}\ e^{-(\beta'-\beta)U_i + (\mu'\beta' - \mu\beta)n_i}}. \tag{1.22}$$

Eine weitere Simulation kann so u.U. vermieden werden. Die Bestimmung von Koexistenz kann mit Hilfe des Newton'schen Nährungsverfahrens automatisiert werden. Hierbei wird μ iterativ angepasst bis die beiden Flächen $A_{\mathrm{I,II}}$ gleich groß sind. Die Reichweite der Extrapolation ist jedoch begrenzt. So müssen die Datenpunkte (n,E) der ursprünglichen Verteilung und der Verteilung bei μ', T' genügend überlappen. Die Überschneidung wird mit wachsender Systemgröße geringer, da die Breite der Peaks mit $\sqrt{L^3}$ abnimmt.

Verteilungen wie in Abbildung 1.7 finden sich lediglich in der Nähe eines kritischen Punktes. Als Solchen bezeichnet man die Stelle im Phasendiagramm, oberhalb der die Phasen eines makroskopisches System nicht mehr separieren. Unterhalb des kritischen Punktes können Phasen koexistieren. Senkt man bei einem einkomponentigen System beispielsweise die Temperatur, so sinken die Übergangswahrscheinlichkeiten (Positionen in der Mitte von Abb. 1.7) exponentiell. Eine einfache Monte–Carlo–Simulation verharrt dann entweder im Gas– oder Flüssigkeitspeak, je nachdem von welcher Konfiguration aus gestartet wurde. Zur Überwindung der Barriere benötigt man fortgeschrittene Konzepte, die zum Teil im Rahmen dieser Arbeit entwickelt wurden (vgl. mit Kap. 2).

Bei Mischsystemen erhält man statt $P(n)$ zweidimensionale Verteilungen $P(n_1,n_2)$. Abbildung 1.8 a zeigt ein solches Profil. Die Farben verdeutlichen dabei die Höhe der

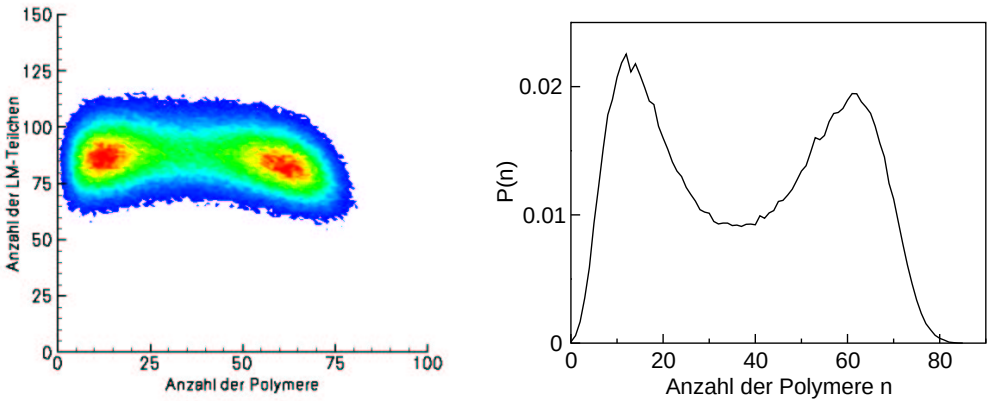

Abbildung 1.8: (a) Wahrscheinlichkeitsprofil des Polymer–Lösungsmittel–Mischsystems an einem kritischen Punkt ($T = 1.55\frac{\varepsilon_5}{k}$, $\xi = 1.0$, $L = 9\,\sigma_5$), (b) Projektion auf die Polymerachse.

Maxima. Rot steht für den größten Wert und Blau für den niedrigsten. Projiziert man die Verteilung in diesem Fall auf die Polymerachse (Abb. 1.8 b), erhält man eine Verteilung, die Abbildung 1.7 ähnelt. Für die meisten Mischsystem–Simulationen liefert die Projektion auf eine der beiden Achsen eine gute Grundlage zur Bestimmung der Phasenkoexistenz. Die Extrapolation eines Datensatzes verläuft analog Gleichung (1.21)

und (1.22) unter der Einbeziehung des zweiten chemischen Potentials.

1.4 Zur Bestimmung kritischer Punkte

Phasenübergänge können mit Hilfe eines Ordnungsparameters M charakterisiert werden. Hierbei handelt es sich um eine Größe, die in der ungeordneten Phase Null wird und in der geordneten ungleich Null. Ein klassisches Beispiel hierfür ist die spontane Magnetisierung in einem Ising–System. Bei einem Gas–flüssig–Übergang betrachtet man den Unterschied zwischen Flüssigkeits– und Gasdichte:

$$M := \rho_l - \rho_g. \tag{1.23}$$

Kritische Punkte (bei $T = T_c$) markieren den Wechsel vom Ein– zum Zwei–Phasen–Gebiet. Sie lassen sich durch einen Phasenübergang zweiter Ordnung beschreiben, d.h. die zweiten Ableitungen der freien Energie werden singulär. Unterhalb von T_c können die thermodynamischen Eigenschaften eines (makroskopischen) Systems mit Hilfe einfacher Potenzgesetze beschrieben werden [28], [29]:

$$M = M_0\, \varepsilon^{\beta}, \tag{1.24}$$

$$\chi = \chi_0\, \varepsilon^{-\gamma}, \tag{1.25}$$

$$C = C_0\, \varepsilon^{-\alpha}, \tag{1.26}$$

$$\xi = \xi_0\, \varepsilon^{-\nu}. \tag{1.27}$$

Der Parameter ε kennzeichnet an dieser Stelle den Abstand vom kritischen Punkt. Im einfachsten Fall ist $\varepsilon = (1 - T/T_c)$. χ steht für die Suszeptibilität, C für die spezifische Wärme und ξ für die Korrelationslänge (und an dieser Stelle nicht für die Abweichung von der Lorentz–Berthelot–Regel). Gleichungen (1.25), (1.26), (1.27) gelten im Prinzip auch oberhalb des kritischen Punktes, die Amplituden unterscheiden sich jedoch. Systeme, die den gleichen Satz von kritischen Exponenten aufweisen, gehören der gleichen Universalitätsklasse an [29]. Das kritische Verhalten von Flüssigkeiten entspricht dem von Isingmagneten gleicher räumlicher Dimension [30], [31]. Die entsprechenden Exponenten können mit Hilfe von Computersimulationen bestimmt werden [32], [33]: $\beta = 0.3258(44)$, $\gamma = 1.2390(71)$, $\alpha = 0.110(5)$ und $\nu = 0.6294(2)$.

In endlichen Systemen ist die Wahrscheinlichkeitsverteilung des Ordnungsparameters abhängig von der Systemgröße:

$$P(M') = L^{\frac{\beta}{\nu}} P(\tilde{M}, x) = L^{\frac{\beta}{\nu}} f\left(L^{\frac{\beta}{\nu}} M, L^{\frac{1}{\nu}} \varepsilon\right) \quad \text{mit} \quad M' = \rho - \langle \rho \rangle. \tag{1.28}$$

Die Wahl der Skalenvariablen x wird durch die Beobachtung motiviert, dass die Korrelationslänge durch die Systemgröße L begrenzt wird (L „skaliert“ wie ξ. Statt $L/\xi \propto \varepsilon^{\nu} L$, kann man auch $x = \varepsilon L^{\frac{1}{\nu}}$ als Argument in der oberen Funktion benutzen). Setzt man $M' = a\, g(\varepsilon L^{\frac{1}{\nu}}) \propto \varepsilon^{\beta}$, folgt $a = L^{-\frac{\beta}{\nu}}$. Hierdurch lässt sich die Einführung von $\tilde{M} = L^{\frac{\beta}{\nu}} M'$ motivieren. Der Vorfaktor $L^{\frac{\beta}{\nu}}$ in (1.28) ergibt sich durch die Normierung der Wahrscheinlichkeitsverteilung auf eine Größe, die lediglich von x abhängt: $\int P(M')\,\mathrm{dM}' = \int L^{\frac{\beta}{\nu}} P(\tilde{M}) L^{-\frac{\beta}{\nu}}\,\mathrm{d}\tilde{\mathrm{M}}$. Wegen (1.28) ist die Wahrscheinlichkeitsverteilung am kritischen Punkt bimodal (vgl. Abb.1.7 und 1.8). Für $L \to \infty$ geht der Abstand beider Maxima gegen Null.

Als Nächstes betrachtet man die durchschnittliche Schwankung von M':

$$\langle M'^2 \rangle = \int M'^2 P(M')\,\mathrm{dM}' = \int \tilde{M}^2 L^{-\frac{2\beta}{\nu}} L^{\frac{\beta}{\nu}} f(\tilde{M}, x) L^{-\frac{\beta}{\nu}} \mathrm{d}\tilde{\mathrm{M}} = L^{-\frac{2\beta}{\nu}}\, h_{\mathrm{M}'^2}\left(L^{\frac{1}{\nu}}\varepsilon\right). \tag{1.29}$$

Es gilt:

$$\langle M'^4 \rangle = \int M'^4 P(M')\,\mathrm{dM}' = L^{-\frac{4\beta}{\nu}}\, h_{\mathrm{M}'^4}\left(L^{\frac{1}{\nu}}\varepsilon\right). \tag{1.30}$$

Bildet man das Verhältnis

$$U_4 = \frac{\langle M'^4 \rangle}{\langle M'^2 \rangle^2}, \tag{1.31}$$

so kürzen sich die L–Abhängigkeiten gerade weg. Am kritischen Punkt wird $U_4 = h_{\mathrm{M}'^4}(0)/h_{\mathrm{M}'^2}(0)$ und somit unabhängig von der Systemgröße [34]. Trägt man U_4 für verschiedene Systemgrößen gegen die Temperatur auf, schneiden sich diese am kritischen Punkt (Abb. 1.9 a). Die gleiche Aussage gilt prinzipiell auch für $U_2 = \frac{\langle M'^2 \rangle}{\langle |M'| \rangle^2}$ (Abb. 1.9 b) oder andere auf vergleichbare Weise konstruierte Größen. Im Allgemeinen

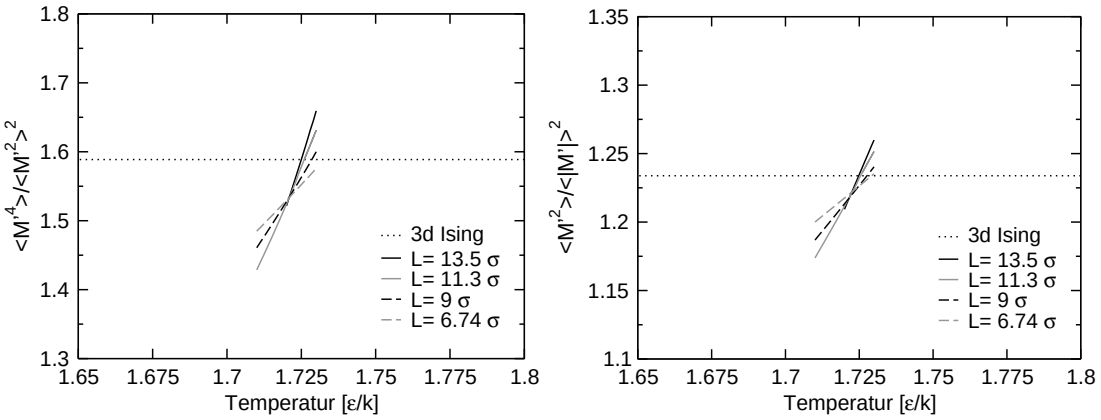

Abbildung 1.9: Kumulanten des Fünfmer–Systems in der Umgebung des kritischen Punktes ($T_c = 1.725\ \frac{\varepsilon}{k}$).

genügt es, für jede Systemgröße einen einzelnen Datensatz zu generieren und die Kumulanten als Funktion von T durch Extrapolation zu erzeugen (vgl. Kap. 1.3) Der kritische Punkt kann auch als Schnittpunkt mit dem Kumulantenwert eines großen dreidimensionalen Ising–Systems bei T_c bestimmt werden. Die Abweichungen im Vergleich zu den Kumulantenschnittpunkten sind kleiner als 1% und lassen sich auf Effekte höherer Ordnung zurückführen. Bei dieser Größe liegt in etwa auch die Genauigkeit der Simulationen.

Ein äquivalentes Verfahren ist die Abbildung der Wahrscheinlichkeitsverteilung auf die „3d–Ising–Masterkurve“. Hierbei trägt man $\sqrt{\langle M'^2\rangle}\, P(M')$ gegen $M'/\sqrt{\langle M'^2\rangle}$ auf und vergleicht die Verteilung mit der eines entsprechend normierten Ising–Systems. Wegen $M' \propto L^{-\frac{\beta}{\nu}}$, $P(M') \propto L^{\frac{\beta}{\nu}}$ und $\sqrt{\langle M'^2\rangle} \propto L^{-\frac{\beta}{\nu}}$ (1.29), kürzen sich die L–Abhängigkeiten weg und man erhält am kritischen Punkt eine universelle Verteilung (Abb. 1.10). Da man mit dieser Methode lediglich einen einzigen Lauf zur Bestimmung eines kritischen Punktes benötigt, wurde sie zur Bestimmung aller kritischen Punkte eingesetzt.

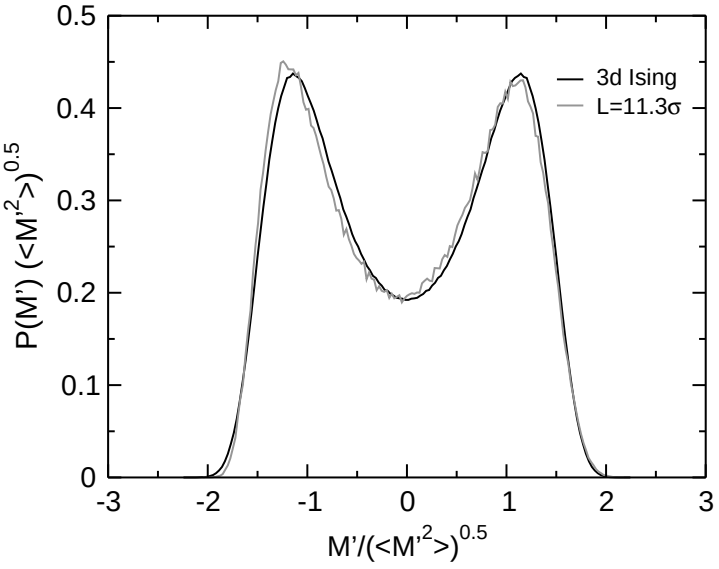

Abbildung 1.10: Normierte Wahrscheinlichkeitsverteilung am kritischen Punkt im Vergleich mit der „3d–Ising–Masterkurve“ [35] ($L = 13.5\ \sigma$).

Der Vollständigkeit halber sollte erwähnt werden, dass Definition (1.23) keine vollständige Äquivalenz mit dem kritischen Verhalten eines Ising–Magneten gewährleistet. Eine exakte Abbildung wird durch eine Linearkombination von Dichte und Energie erreicht [31]. Die leichte Asymmetrie der normierten Wahrscheinlichkeitsverteilung (Abb. 1.10) lässt sich auf die Vernachlässigung dieses Effektes zurückführen. Da der

Einfluss insgesamt gesehen jedoch eher gering ausfällt und mit zunehmender Systemgröße kleiner wird, wurde in dieser Arbeit die einfachere Definition angewendet.

1.5 Zur Bestimmung von Phasendiagrammen

Aus der Wahrscheinlichkeitsverteilung $P_{\mu,\mathrm{V,T}}(n)$ kann die Gas– und Flüssigkeits–Koexistenzdichte (Maxima von $P(n)$) bei Temperatur T abgelesen werden. Trägt man die Daten für unterschiedliche Temperaturen zusammen, erhält man die Koexistenzkurve (Abb. 1.11). Diese auch als Binodale bezeichnete Linie endet im kritischen Punkt, der gemäß Kapitel 1.4 bestimmt wurde. In dessen Nähe zeigen sich die im vorangegangenen Abschnitt beschriebenen Finite–Size–Effekte, d.h. die durch Simulation bestimmten Koexistenzdichten weichen vom wahren Verlauf der Koexistenzkurve ab (vgl. Abb. 1.11). Zur Korrektur bestimmt man in der Nähe von T_c Koexistenzdichten für verschiedene Systemgrößen. Fallen diese aufeinander, sind die Abweichungen vernachlässigbar. Für den verbleibenden Bereich „fittet“ man gemäß (1.24) die Umkehrfunktion von $\rho_{\mathrm{l,g}} = \langle\rho\rangle \pm \frac{1}{2} M_0\, \varepsilon^\beta$ an die Kurve. $\langle\rho(T)\rangle = \rho_\mathrm{c} + a\varepsilon^{1-\alpha} + b\varepsilon$ wird zuvor durch eine lineare Funktion angenähert.

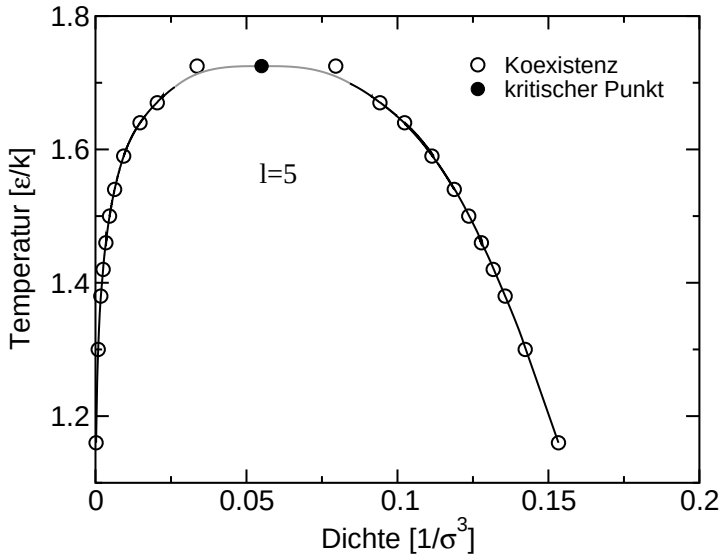

Abbildung 1.11: Phasendiagramme des Fünfmer–Systems. Der Verlauf der Binodalen in der Nähe des kritischen Punktes (grau) wurde durch „finite size scaling“ bestimmt.

Bei einer Isotherme des Mischsystems wird gewöhnlich der Druck p als Funktion der molaren Zusammensetzung x aufgetragen. In den in dieser Arbeit untersuchten

Fällen kann der Dichteunterschied der Polymerphasen als Ordnungsparameter herangezogen werden. Es gilt also: $\rho_{5\,\mathrm{l,g}} = \langle\rho_5\rangle \pm \frac{1}{2} M_0\, \varepsilon^{\beta}$, mit $\varepsilon = (1 - p/p_\mathrm{c})$. Die Dichte der Lösungsmittelteilchen ρ_1 wird durch eine lineare Funktion genähert. Die Umkehrfunktion von $x = \rho_{1\,\mathrm{l,g}}/(\rho_{1\,\mathrm{l,g}} + \rho_{5\,\mathrm{l,g}})$ liefert dann den gewünschten Verlauf in der Umgebung von p_c.

1.6 Zur Bestimmung von Grenzflächenspannungen

Im großkanonischen Ensemble ist die Wahrscheinlichkeit n Teilchen in der Box vorzufinden gegeben durch das Verhältnis der kanonischen mit der großkanonischen Zustandsumme: $P(n) = Z_\mathrm{k}/Z_\mathrm{gk}$. Auf diese Weise lässt sich die freie Energie $F(n)$ mit $P(n)$ verknüpfen [36]:

$$F(n) = -kT \ln Z_\mathrm{k} = -kT \ln P(n) + \text{konst.} \tag{1.32}$$

Die beiden Minima in F (Maxima in P) kennzeichnen die homogene Gas– und Flüssigkeitsphase (Abb. 1.12 a) Bei mittleren Dichten findet man hingegen scheibenförmige Konfigurationen vor, die je zur Hälfte mit Gas bzw. mit Flüssigkeit gefüllt sind (Abb. 1.12 b). Da beide Phasen Koexistenzdichte aufweisen, entspricht die zusätzliche freie Energie gerade dem Beitrag der Grenzfläche. Hieraus kann die Grenzflächenspannung γ bestimmt werden:

$$\gamma = \frac{\Delta F}{2L^2} \qquad \text{mit} \qquad \Delta F = F\left(\frac{n_g + n_f}{2}\right) - \frac{1}{2}\left(F(n_g) + F(n_f)\right). \tag{1.33}$$

Die Konfiguration in Abbildung 1.12 b weist zwei Grenzflächen der Größe L^2 auf, daher der Faktor Zwei im Nenner des Ausdrucks (1.33). Zur Bestimmung von γ genügt es, dass $F(n)$ in der Mitte flach wird. Eine Vergrößerung der jeweiligen Phase führt dann nicht zu einer Änderung in F, da die freie Energie der Grenzfläche konstant bleibt. In jedem Fall ist es für die Bestimmung von γ günstig, lang gestreckte Simulationsboxen (z.B. $V = L \times L \times 2L$) zu benutzen. Dies unterbindet eine Wechselwirkung beider Grenzflächen über die periodischen Randbedingungen und somit ungewollte Beiträge in der freien Energie.

Abbildungen 1.13 a und b zeigen die Grenzflächenspannung des reinen Lennard–Jones–Systems sowie des Fünfmer–Systems als Funktion der Temperatur. In der Nähe des kritischen Punktes wurde gemäß (1.27) eine Funktion der Form $\gamma = \gamma_0\, \varepsilon^{2\nu}$ an den Graphen angepasst. Ein Vergleich mit experimentellen Daten findet sich in Kapitel 3.

Der Übergang vom homogenen Gas zur Flüssigkeit vollzieht sich stufenweise. Ab einer gewissen Dichte formt sich ein einzelner Tropfen, der dann in eine Konfiguration mit linearer Grenzfläche einmündet. Mit weiter anwachsender Dichte bildet sich

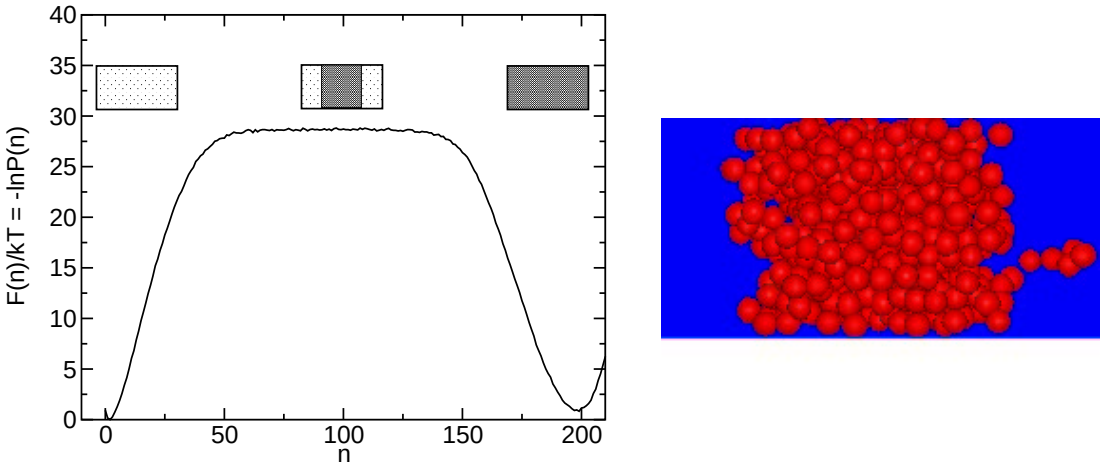

Abbildung 1.12: (a) Freie Energie als Funktion der Teilchenzahl ($T = 1.38\ \frac{\varepsilon}{k}, V = 9 \times 9 \times 18\ \sigma^3$, Kettenlänge $l = 5$), (b) typische Konfiguration bei $n = 100$ Polymeren.

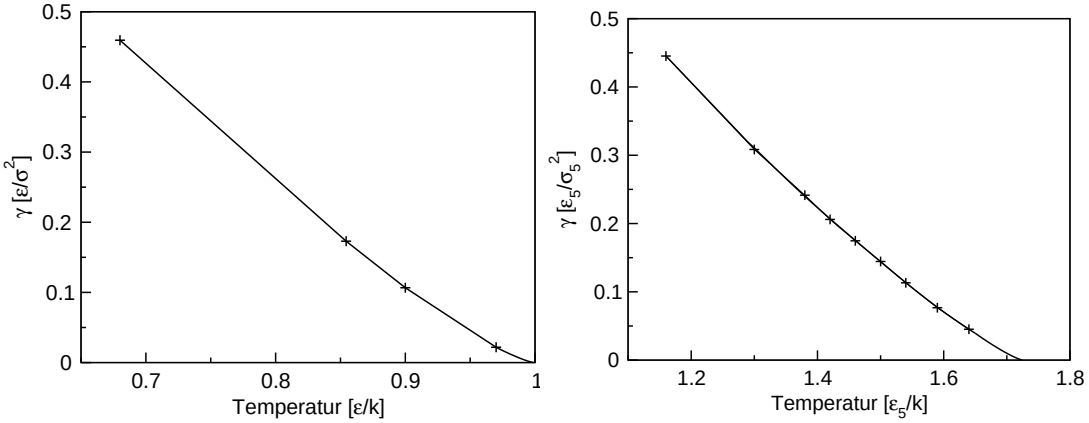

Abbildung 1.13: Grenzflächenspannung als Funktion der Temperatur: (a) reines Lennard–Jones–System, (b) Fünfmer–System.

eine von Flüssigkeit umgebene Blase, die schließlich kollabiert. Diese Phasenübergänge erster Ordnung schmieren bei kleinen Boxen und hohen Temperaturen aus [37] und werden nochmals ausführlich in Kapitel 5 diskutiert. Wie in Abbildung 1.12 a angedeutet, werden die Übergangswahrscheinlichkeiten von der Gas– zur Flüssigkeitsseite mit wachsendem Abstand vom kritischen Punkt sehr klein. Eine ungewichtete Simulation würde in einer der beiden Phasen verharren, je nachdem welche Startkonfiguration gewählt wurde. Zur Überwindung der Barriere in der freien Energie werden fortgeschrittene Konzepte benötigt, die im folgenden Kapitel vorgestellt werden.

Kapitel 2

Großkanonische Simulationsverfahren

Im vorangegangenen Kapitel wurde erläutert, wie sich aus der Wahrscheinlichkeitsverteilung des Ordnungsparameters Phasenkoexistenz, kritische Punkte und Grenzflächeneigenschaften ableiten lassen. In Ergänzung hierzu beschäftigt sich Kapitel 2 mit der Erzeugung der Wahrscheinlichkeitsverteilung. Die hier vorgestellten Methoden lassen sich leicht auf eine Vielzahl von Gitter– (Ising–, Bond–Fluktuations–) und Kontinuumsmodellen übertragen.

Die zu Grunde liegenden Schwierigkeiten einer großkanonischen Simulation (bei konstantem chemischen Potential μ, Volumen V und Temperatur T) werden nochmals in Fig. 2.1 verdeutlicht. In der Nähe des kritischen Punktes kann ein endlich großes

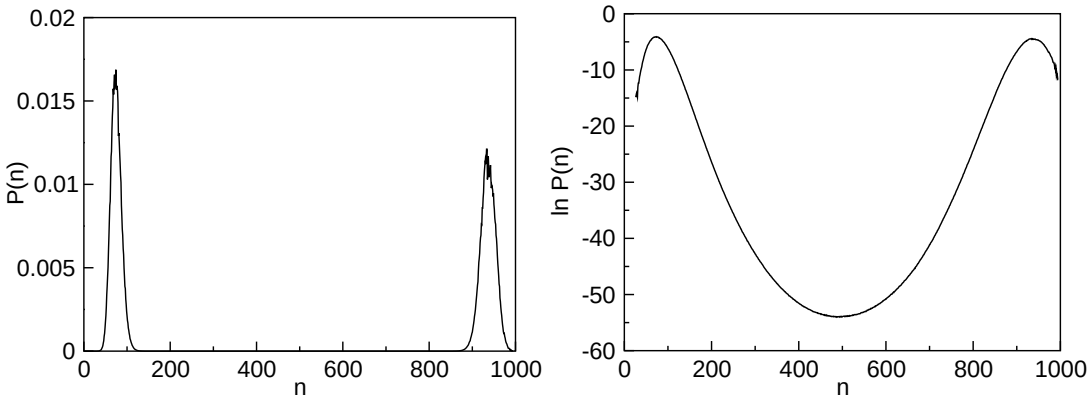

Abbildung 2.1: Wahrscheinlichkeitsverteilung eines Lennard–Jones–Systems der Größe $L^3 = (11.3\,\sigma)^3$ bei $T = 0.85437\,\frac{\varepsilon}{k}$ und Koexistenz.

System bei Koexistenz zwischen der Gas– und der Flüssigkeitsphase hin und her wechseln. Bei einer nur geringfügig tieferen Temperatur (hier $T = 0.85437\,\frac{\varepsilon}{k} = 0.85\,T_c$) bildet sich zwischen beiden Phasen eine große Barriere in der freien Energie heraus.

Die Wahrscheinlichkeit einen Zwischenzustand zu generieren sinkt exponentiell. So ist im oberen Beispiel die Häufigkeit mit der sich 75 Monomere in der Simulationsbox aufhalten e^{50} mal höher als für 500 Monomere. Das System würde demnach bei einer nicht–gewichteten Simulation und endlicher Simulationszeit entweder im Gas– oder Flüssigkeitspeak verharren – je nachdem welcher Anfangszustand gewählt wurde. Zur Bestimmung von Koexistenz fernab des kritischen Punktes wurden Methoden entwickelt, die im Folgenden vorgestellt werden.

2.1 Multikanonische Simulationen

In einer multikanonischen Simulation modifiziert man den Hamiltonian in der Monte–Carlo–Abfrage mit dem Ziel, jede Dichte mit der gleichen Häufigkeit auftreten zu lassen. Die wahre Wahrscheinlichkeitsverteilung kann im Anschluss aus den Simulationsdaten und der Kenntnis der Modifikation rekonstruiert werden. Hierdurch werden die eingangs geschilderten Probleme umgangen. Die Bestimmung des Modifikationsfaktors ergibt sich in der Regel aus der Extrapolation vorangegangener Simulationsläufe (vgl. Kapitel 1.3). Die Methode wurde 1991 von *Berg* und *Neuhaus* [38], [39] für das zweidimensionale Ising–Modell entwickelt und gilt heute als Standard. Eine Übertragung auf Lennard–Jones–Systeme findet sich unter Anderem in [40].

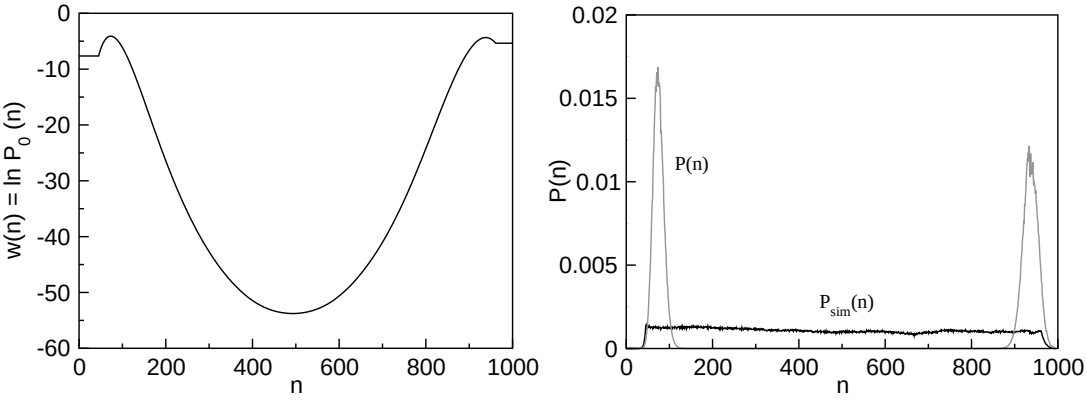

Abbildung 2.2: (a) Gewichtsfunktion für ein Lennard–Jones–Systems der Größe $L^3 = (11.3\ \sigma)^3$ bei $T = 0.85437\ \frac{\varepsilon}{k}$. An den beiden Rändern wird die Simulation nicht gewichtet, (b) $P_{\mathrm{sim}}(n)$ und die im Analyseschritt konstruierte wahre Verteilung $P(n)$ (beide normiert).

In unserem Modell sei $U = U_{\mathrm{kan}} + \mu n$ der Hamiltonian vor und U_{sim} der Hamiltonian nach der Modifikation. Sei $P(n)$ die gesuchte Wahrscheinlichkeitsverteilung, $P_0(n)$ ein Schätzwert für $P(n)$, $P_{\mathrm{sim}}(n)$ die durch die Modifikation erzeugte, simulierte multikanonische Verteilung, w die Gewichtsfunktion und Z_{gk} die großkanonische

Zustandssumme. Wählt man

$$U_{\text{eff}} = U + kT \cdot w(n) \quad \text{mit} \quad w(n) = \ln(P_0(n)), \tag{2.1}$$

so ist

$$P_{\text{sim}}(n) = \frac{\sum_{\{S_n\}} e^{-\beta U_{\text{eff}}}}{Z_{\text{gk}}} = \frac{1}{P_0(n)} \frac{\sum_{\{S_n\}} e^{-\beta U}}{Z_{\text{gk}}} = \frac{P(n)}{P_0(n)}, \tag{2.2}$$

wobei über alle Zustände S_n mit n Teilchen summiert wird. Ist der Vorschlag für die Wahrscheinlichkeitsverteilung gut, d.h. $P_0(n) \approx P(n)$ für alle n, so ist $P_{\text{sim}}(n)$ ungefähr konstant (vgl. Fig. 2.2 b). Die wahre Verteilung ist dann nach (2.2)

$$P(n) = P_{\text{sim}}(n) P_0(n) = P_{\text{sim}}(n) \cdot e^w. \tag{2.3}$$

Es bleibt anzumerken, dass man für Zustände mit schlechter Statistik (z.B. an den Rändern) entgegen (2.1) ein konstantes w wählen sollte. Da in die Monte–Carlo–Abfrage lediglich Energiedifferenzen eingehen, entspricht dies einer nicht-gewichteten Simulation des Bereichs (vgl. Fig. 2.2 a). Die Parallelisierung des Algorithmus gestaltet sich denkbar einfach. Zur Verbesserung der Statistik lässt man das gleiche System einfach auf mehreren Prozessoren laufen. Hierbei sollte jedoch beachtet werden, dass das System auf einem einzelnen Prozessor oft genug zwischen Gas– und Flüssigkeitspeak hin und her wechselt.

Möchte man Polymere statt Lennard–Jones–Teilchen simulieren, beschreibt n die Anzahl der Ketten im System. Bei der Simulation von Polymer–Mischungen im (μ_A, μ_B, V, T)–Ensemble liegt es nahe, das multikanonische Verfahren durch die Einführung einer zweidimensionalen Gewichtsfunktion zu erweitern. Da die Statistik eines einzelnen Punktes (n_A, n_B) schlechter ist als für eine reine Schmelze, empfiehlt es sich die Simulation lediglich entlang eines Parameters zu gewichten:

$$w(n_A, n_B) = w(n_A) = \ln(P_0(n_A)) = \text{konstant für alle } n_B. \tag{2.4}$$

Erwartet man beispielsweise im Wesentlichen eine Koexistenz zweier Polymerphasen, würde man die Simulation entlang der Lösungsmittel–Teilchen–Achse nicht gewichten, also $w(n_{\text{Poly}}, n_{\text{LM}}) = w(n_{\text{Poly}})$ konstant wählen. In der Nähe des kritischen Punktes der Lösungsmittelteilchen wählt man entsprechend $w(n_{\text{Poly}}, n_{\text{LM}}) = w(n_{\text{LM}}) =$ konstant.

Die eigentliche Schwierigkeit der multikanonischen Methode besteht in der Erzeugung einer brauchbaren Gewichtsfunktion, da diese die Kenntnis der Wahrscheinlichkeitsverteilung bereits voraussetzt. Um einen Zustand fernab eines kritischen Punktes zu generieren, beginnt man typischer Weise mit einer nicht–gewichteten Simulation in dessen Nähe. Diese Daten werden dann zu neuen Parametern hin extrapoliert (vgl. Kapitel 1.3). Der Schätzwert für $P(n)$ wird dann als Eingabe für eine neue Simulation benutzt. Auf diese Weise kann man sich Schritt für Schritt dem gewünschten

Zustand annähern. Hierin liegt zugleich der größte Nachteil der Methode begründet. Möchte man unterschiedliche Systemgrößen bei tiefen Temperaturen betrachten, muss jede Gewichtsfunktion einzeln durch die oben beschriebene Serie von Simulationen erzeugt werden. Eine Übertragung von einer Systemgröße auf eine andere ist für praktische Zwecke zu ungenau. Die nun folgenden Abschnitte werden diese Problematik nochmals aufgreifen und neue Lösungsansätze vorstellen.

2.2 Zur Erzeugung von Gewichtsfunktionen

Im folgenden Kapitel soll eine neue Methode zur Ad–hoc–Erzeugung von Gewichtsfunktionen beschrieben werden. Sie basiert auf einem von *Wang* und *Landau* für Ising–Systeme entwickelten Verfahren zur Bestimmung der Zustandsdichte $g(E)$ [41]. Die grundlegende Idee ist einfach. Man nähert während eines Laufs die Gewichtsfunktion dem Logarithmus der Wahrscheinlichkeitsverteilung schrittweise an (vgl. (2.1) und Abbildung 2.3). Als Kontrolle dient ein Histogramm $H(n)$. Wird dieses „flach“, ist die Approximation gut. Im Einzelnen funktioniert der Algorithmus wie folgt:

Start:	Man startet mit $w(n) = 0$ und $H(n) = 0$ für alle n. (Würde man $w(n)$ jetzt nicht mehr verändern, erhielte man eine nicht–gewichtete Simulation.)
|: Iterationsschritt:	|: Wird ein MC–Schritt (z.B. von $n_1 \rightarrow n_2$) akzeptiert, wird w(n_2) um ein Inkrement Δw und $H(n_2)$ um Eins erhöht. Wird der Schritt abgelehnt, erhöht man $w(n_1)$ und $H(n_1)$. (Wegen $w = \ln(P(n))$ entspricht dies jeweils einer Multiplikation von $e^{\Delta w}$ an $P(n)$.)
	Die Prozedur wird so lange wiederholt bis $H(n)$ „flach“ wird, d.h. $\lvert H(n) - \langle H(n)\rangle\rvert < (1-\varepsilon)\cdot\langle H(n)\rangle$. :|
Neustart:	Das Inkrement Δw wird erniedrigt und der Iterationsschritt wiederholt. $w(n)$ wird beibehalten aber $H(n)$ wieder auf Null gesetzt :|
Abbruchkriterium:	Die Simulation endet, wenn ein vorher festgelegtes Δw unterschritten wurde. (An dieser Stelle sollte $w(n) \approx \ln(P(n))$ sein.)

$w(n)$ wird hierdurch bevorzugt an Stellen erhöht, die stark von der wahren Verteilung abweichen. Startet man z.B. bei Koexistenz und einer leeren Box, werden zunächst die Gewichte um den Gaspeak erhöht und anschließend die der benachbarten Zustände. Nach einer gewissen Zeit ist das System dann in der Lage, die Barriere in der freien Energie zu überwinden und den Flüssigkeitsbereich zu erreichen. Dieser wird wiederum sukzessive um das Maximum herum aufgebaut. Anschließend wechselt das System mehr oder weniger gleichmäßig zwischen den verschiedenen Zuständen hin und her. Auf diese Weise bildet $\ln(w(n))$ die Wahrscheinlichkeitsverteilung Schritt für Schritt nach. Im Grunde genügt es, direkt mit dem Endwert für Δw zu beginnen und die Simulation im Anschluss nicht mehr neu zu starten. Sind die Barrieren in der freien Energie jedoch groß, ist es häufig vorteilhaft zunächst die grobe Struktur der Verteilung mit einem großen Inkrement nachzubilden und diese dann durch kleinere Δws sukzessive zu verfeinern (vgl. Abb. 2.3 a).

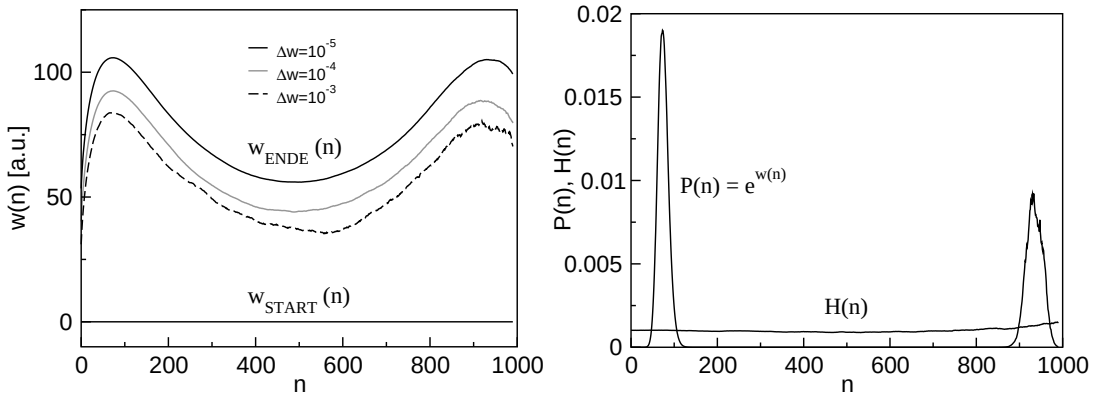

Abbildung 2.3: (a) Gewichtsfunktion am Ende der jeweiligen Iterationsschritte (Lennard–Jones–System der Größe $L^3 = (11.3\ \sigma)^3$ bei $T = 0.85437\ \frac{\varepsilon}{k}$), (b) $H(n)$ und $P(n)$ am Ende der Simulation (beide normiert).

Der Algorithmus verletzt wegen der ständigen Anpassung des Hamiltonians die Detailed–Balance–Bedingung in der Monte–Carlo–Abfrage (vgl. Kapitel 1.2). Die Abweichung fällt dabei um so geringer aus je kleiner Δw wird und verschwindet für den Fall einer konstanten Gewichtsfunktion ($\Delta w = 0$). Da wir jedoch lediglich an einem Schätzwert für $P(n)$ interessiert sind, stört dies nicht weiter. Die eigentliche Schwierigkeit besteht im Festlegen eines geeigneten Start– und Abbruchwertes für das Inkrement. Wird Δw zu groß (z.B. 10), bewirkt jeder Schritt eine unverhältnismäßige Erhöhung von $w(n)$ und die Simulation wird u.U. aus einem Gleichgewichtszustand herausdrückt. Zu einem gewissen Grad ist dieses zusätzliche Moment jedoch durchaus gewollt, da so der gesamte Dichtebereich recht schnell abgedeckt wird. Durch diverse Testläufe hat sich für unser System ein Startwert von $\Delta w = 10^{-3}$ und ein Abbruch nach der Iteration mit $\Delta w = 10^{-5}$ als geeignet erwiesen. Δw wurde bei jedem Neustart um den Faktor 10 erniedrigt, ε betrug 0.5. Diese Werte sollten jedoch lediglich als Richt-

linie verstanden werden. In Bereichen großer Dichte und geringer großkanonischer Akzeptanzraten muss Δw u.U. kleiner gewählt werden.

Eine Fehlerabschätzung erweist sich wegen des grundsätzlichen Problems der Verletzung von „detailed balance“ als schwierig. Es ist daher ratsam, die Methode lediglich zur Generierung einer Gewichtsfunktion zu benutzten. Diese liefert nach einer Extrapolation einen Schätzwert für das chemische Potential und die Verteilung bei Koexistenz. Im Anschluss sollte immer ein multikanonischer Produktionslauf (mit $\Delta w = 0$) durchgeführt werden. Ist die erzeugte Gewichtsfunktion von ausreichender Güte, kann das System zwischen allen Zuständen hin und her wechseln. Die Erweiterung auf Polymerlösungen erfolgt analog (2.4). Die Gewichtsfunktion wird lediglich in eine Richtung (z.B. für Polymere) geändert und ist konstant für die zweite Teilchensorte (z.B. das Lösungsmittel). In ihrer ursprünglichen Veröffentlichung [41] schlagen *Wang* und *Landau* vor, die Simulation entsprechend eines Umbrella–Sampling–Schemas [42] in mehrere überlappende Simulationsfenster aufzuteilen. Diese werden am Ende wieder zur Wahrscheinlichkeitsverteilung zusammengesetzt. Der Algorithmus kann so einfach parallelisiert werden. Berechnet man $w(n)$ auf einem schnellen Prozessor, erspart man sich jedoch das Zusammensetzen der Fenster, vermeidet Probleme mit der Fehlerfortpflanzung und der Bestimmung des Abbruchwertes von Δw. Des Weiteren umgeht man unterschiedliche Laufzeiten in den Dichteintervallen. Es ist zudem zu hinterfragen, ob diese Art von Parallelisierung wie vielfach behauptet [43], [20] wirklich Geschwindigkeitsvorteile bringt (siehe hierzu Diskussion in Kapitel 2.3.3). Wir werden die Idee der Simulationsfenster jedoch keineswegs fallen lassen sondern im nächsten Abschnitt aufgreifen und ausbauen.

2.3 Diskussion eines neuen Simulationsverfahren

Das folgende Kapitel beschäftigt sich mit einer Weiterentwicklung des Umbrella–Sampling–Schemas [44]. Der Grundgedanke ist hierbei benachbarte, um je einen Zustand überlappende Fenster nacheinander zu simulieren. Diese können dann bereits während der Simulation verknüpft werden. Startet man o.B.d.A. mit einer leeren Box und $P(0) = 1$, lässt sich die (unnormierte) Wahrscheinlichkeitsverteilung als Produkt der relativen Häufigkeiten ausdrücken:

$$\frac{P(n)}{1} = \frac{P(1)}{1} \cdot \frac{P(2)}{P(1)} \cdot \ldots \cdot \frac{P(n)}{P(n-1)}. \tag{2.5}$$

Nach einer vorher festgelegten Anzahl von Monte–Carlo–Schritten berechnet man $P(n)$ für alle n des simulierten Fensters:

$$P(n) = P(n-1)\frac{P(n)}{P(n-1)}. \tag{2.6}$$

Anschließend erhöht man die Teilchenzahl, simuliert das nächste Fenster und wiederholt den gesamten Vorgang bis alle relevanten Abschnitte der Verteilung erzeugt wurden.

Hierbei ist zu beachten, dass $P(n)$ z.B. nicht von $P(n-2)$ abhängt und es letztlich nur auf die oben dargestellten Wahrscheinlichkeitsverhältnisse ankommt. Man könnte also auch die gesamte Verteilung durch eine Aneinanderreihung von Fenstern der Größe $\omega = 1$ (Fenster, die lediglich aus zwei Zuständen bestehen) erzeugen. Voraussetzung ist jedoch, dass die verwendeten Monte–Carlo–Schritte innerhalb des vorgegebenen Intervalls alle möglichen Zustände effizient abdecken (vgl. hierzu Kapitel 2.3.3 und letzter Abschnitt in Kapitel 5.2). Es ist weiterhin zu beachten, dass die Zählweise an den Grenzen der Fenster richtig implementiert wird [45]. Das Histogramm $H(n)$ sollte nach jedem Monte–Carlo–Schritt erhöht werden, egal ob dieser akzeptiert oder abgelehnt wurde. Wird also ein Zug, der das Simulationsfenster verlassen würde, vorgeschlagen und abgelehnt, muss $H(n)$ des Randzustandes erhöht werden. Dies ist unmittelbar einsichtig. Wäre das Intervall größer, würde die Simulation nach dem Verlassen des ursprünglichen Fensters nach einer gewissen Zeit wieder zum ursprünglichen Grenzzustand zurückkehren und $H(n)$ müsste entsprechend erhöht werden. Unterbleibt diese Maßnahme treten unerwünschte Fehler auf, die sich u.A. bei [46] und [47] finden.

Der hier vorgestellte Ansatz lässt sich leicht mit dem Konzept der Gewichtsfunktion verbinden. Das erste Fenster ist in der Regel ungewichtet (d.h. $w(n) = 0\ \forall n$ in diesem Abschnitt). Da benachbarte Bereiche nacheinander simuliert werden, kann für alle folgenden Fenster eine Gewichtsfunktion durch Extrapolation erzeugt werden. Aus (2.3) und (2.5) ermittelt man zunächst das Wahrscheinlichkeitsverhältnis:

$$\frac{P(n)}{P(n-1)} = \frac{P_{\text{sim}}(n)}{P_{\text{sim}}(n-1)} \exp\left(w(n) - w(n-1)\right), \tag{2.7}$$

Nachdem $P(n)$ gemäß (2.6) bestimmt wurde, wird $w(n) = \ln(P(n))$ aktualisiert und z.B. quadratisch in das nächste Fenster extrapoliert. Für kleine Fenstergrößen liefert dieses Verfahren in der Regel eine brauchbare Gewichtsfunktion. Manchmal reicht die Genauigkeit jedoch nicht aus. Weichen die gewichteten Wahrscheinlichkeitsverhältnisse (z.B. im ersten Fenster) stark von Eins ab, wird $w(n)$ wie oben beschrieben aktualisiert und die Simulation im gleichen Abschnitt wiederholt. Werden einige Zustände in der vorgegebenen Simulationszeit nicht erreicht, kann $w(n)$ auch um eine feste Anzahl von kT in jedem Iterationsschritt erhöht oder erniedrigt werden. Ein Ansatz wie in Kapitel 2.2 ist ebenfalls denkbar.

Steht lediglich ein einzelner Prozessor zur Verfügung, sollte $w(n)$ durch einen ersten Lauf bestimmt und festgelegt werden. Für alle weiteren Läufe empfiehlt es sich, die Gewichtsfunktion nicht mehr zu ändern, da kleine Unterschiede im Prinzip zu einer leicht unterschiedlichen Dynamik führen können. Dies würde eine exakte Bestimmung des Fehlers in Frage stellen. Die resultierenden Abweichungen sind jedoch in der Regel

klein. In dieser Form liefert das Verfahren im Prinzip eine Methode zur Bestimmung der Gewichtsfunktion, die dann in darauffolgenden multikanonischen Produktionsläufen genutzt werden kann. Auf einem Parallelrechner können alle Prozessoren gleichzeitig das gleiche Fenster bearbeiten. Die Simulation kann so direkt und in einem einzigen Lauf durchgeführt werden. Da $P(n)$ mit steigender Prozessorzahl genauer wird, verbessert sich zudem die Qualität der Extrapolation. Die hier vorgestellte Methode funktioniert prinzipiell an jeder Stelle des Phasendiagramms bzw. überall dort, wo die Dichte ein Einsetzen von Teilchen erlaubt. Im Gegensatz zu [41] erfüllt die Simulation zu jedem Zeitpunkt „detailed balance“. Dies ermöglicht eine exakte Bestimmung des Fehlers.

Alle weiterführenden Beispiele in den sich anschließenden Unterkapiteln beziehen sich auf das unten abgebildete Lennard–Jones–Referenzsystem (Abbildung 2.4). Es wurde bei Koexistenz und einer Temperatur von $T = 0.85437\ \frac{\varepsilon}{k}$ simuliert. Die Kantenlänge der kubischen Box betrug $L = 6.74\ \sigma$.

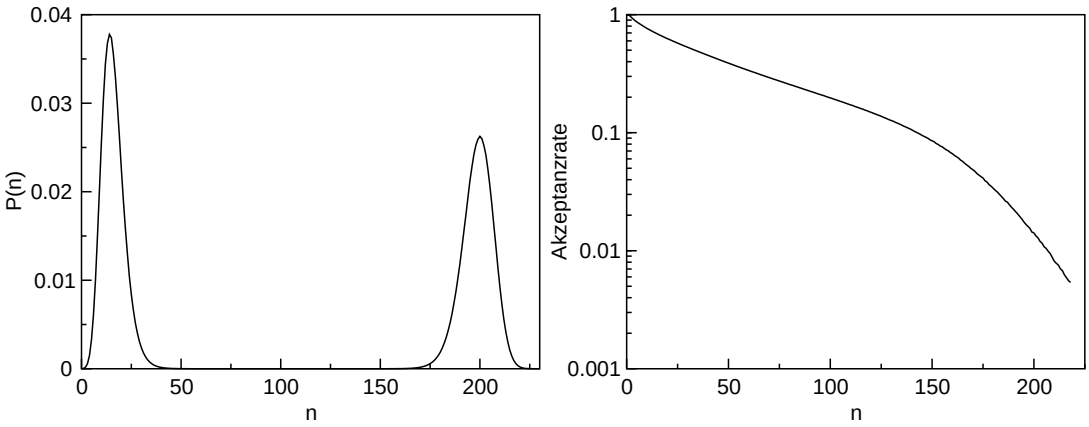

Abbildung 2.4: (a) Wahrscheinlichkeitsverteilung eines Lennard–Jones–Systems der Größe $L^3 = (6.74\ \sigma)^3$ bei $T = 0.85437\ \frac{\varepsilon}{k}$, (b) Akzeptanzrate der großkanonischen Schritte als Funktion der Teilchenzahl für das gleiche System.

2.3.1 Fehleranalyse

Die folgenden Überlegungen sind keineswegs von dem oben erläuterten Extrapolationsschema abhängig. Sie lassen sich allgemein auf alle Verfahren anwenden, die eine Simulation in mehrere Fenster aufteilen, also im Prinzip auf alle Umbrella–Sampling–Schemata. Da bisher jedoch keine ausführliche Fehleranalyse dieser Verfahren vorliegt, soll dies hier nachgeholt werden. Der relative statistische Fehler lässt sich direkt

aus den Schwankungen der Wahrscheinlichkeitsverteilung bestimmen. So ist

$$\Delta_{\text{stat,rel}}\,(P(n)) = \frac{1}{P(n)}\sqrt{\frac{\langle P(n)^2\rangle - \langle P(n)\rangle^2}{a}}, \tag{2.8}$$

a steht für die Anzahl der Läufe bzw. die Anzahl der Prozessoren in der parallelisierten Version. Da die Gewichtsfunktion während eines Laufs konstant bleibt, ist der obere Ausdruck exakt.

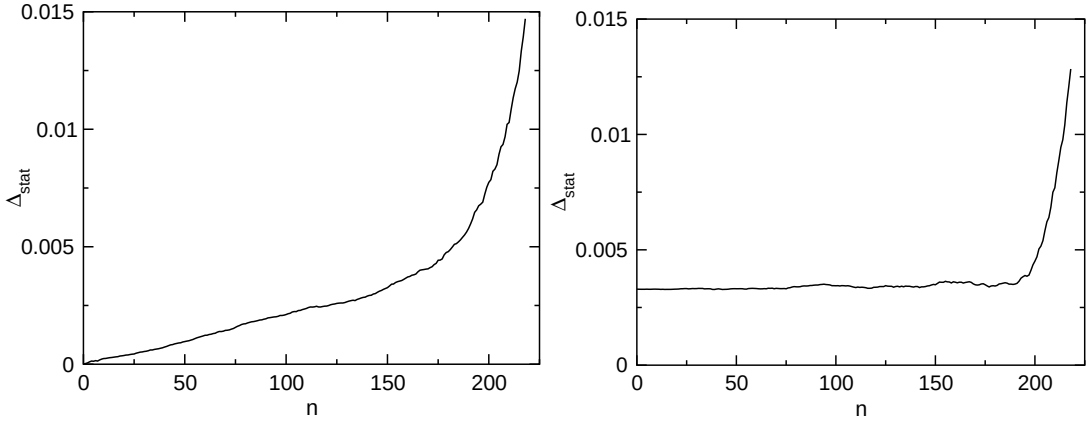

Abbildung 2.5: Fehler der unnormierten (a) und normierten (b) Wahrscheinlichkeitsverteilung. Die Mittelung erfolgte über $a = 400$ Ein–Prozessor–Läufe mit je 2 Millionen Monte–Carlo–Schritten pro Zustand n. Die Fenstergröße ω betrug Eins, d.h. in jedem Simulationsabschnitt wurden lediglich zwei Zustände n und $(n-1)$ zugelassen.

Abbildung 2.5 a zeigt den Fehler der unnormierten Verteilung. Da sich der Fehler von Abschnitt zu Abschnitt „weitervererbt“, ist er am linken Rand Null und am rechten Rand am größten. Der überproportionale Anstieg bei hohen Dichten erklärt sich durch das Absinken der Akzeptanzrate in diesem Bereich (vgl. mit Abbildung 2.4 b). Normiert man jedoch die Verteilung der verschiedenen Läufe bzw. Prozessoren vor Anwendung von Gleichung (2.8) auf Eins, verschieben sich die Absolutwerte und die Fehler nivellieren sich in Bereichen niedriger und mittlerer Dichten (Abb. 2.5 b). Der absolute Fehler bei höheren Dichten ist immer groß im Vergleich zur Mitte und zum linken Rand. Da $P(n)$ am rechten Rand klein ist, hat dieser Bereich jedoch wenig Einfluss auf die Norm und der relative Fehler wird groß. Normierte und unnormierte Darstellung sind prinzipiell gleichwertig. Der Fehler der unnormierten Verteilung lässt sich jedoch einfach aus den Einzelfehlern ableiten und ermöglicht im Folgenden eine Untersuchung von Korrelationen.

Innerhalb eines Intervalls kürzen sich alle Zwischenwerte in Gleichung (2.5) exakt weg und Fehler pflanzen sich nicht fort. Beim Übergang zum nächsten Abschnitt ist der letzte Zustand des alten Intervalls jedoch zugleich der erste Zustand des neuen.

Der Fehler dieses Wertes wird also an das neue Intervall weitergegeben. Nach dem Gauß'schen Fehlerfortpflanzungsgesetz ist der Fehler der unnormierten Verteilung gegeben durch:

$$\Delta_{\text{stat,rel}}\left(P_j(j\omega)\right) = \sqrt{\sum_{i=1}^{j} \Delta^2_{\text{stat,rel}} \left(\frac{P_i(i\omega)}{P_i((i-1)\omega)} \right)}. \tag{2.9}$$

$P_j(n)$ ist $P(n)$ im j–ten Intervall mit $n \in [(j-1)\omega, jw]$. Im Prinzip sollten gemessener (2.8) und durch Akkumulation der Einzelfehler berechneter Fehler (2.9) übereinstimmen. Zur Überprüfung wurden beide für zwei unterschiedlich lange Laufzeiten ermittelt und verglichen (Abbildung 2.6). Die Fenstergröße betrug $\omega = 1$, die Anzahl der Ein–Prozessor–Läufe $a = 400$. Da man am Fehler des einzelnen Laufs interessiert ist, wurden die Ergebnisse mit $\sqrt{a}$ multipliziert. Für die lange Laufzeit ist die Überein-

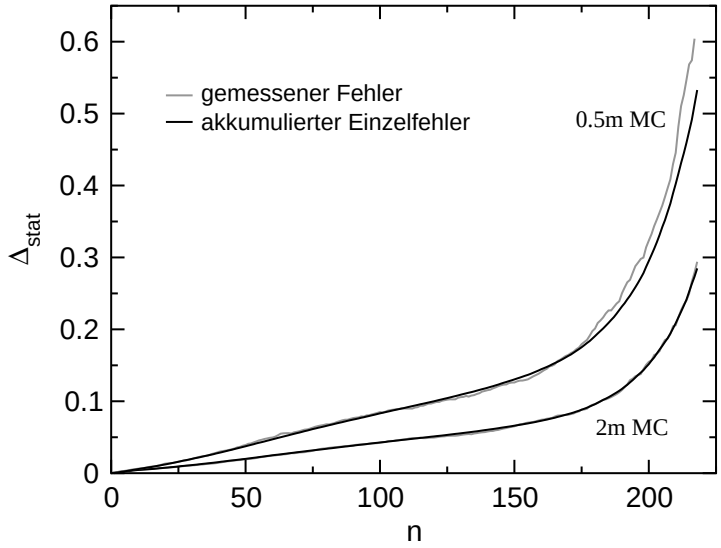

Abbildung 2.6: Gemessener und berechneter Fehler der Wahrscheinlichkeitsverteilung für ein System mit Fenstergröße $\omega = 1$ bei zwei unterschiedlichen Laufzeiten: 0.5 und 2 Millionen MC–Schritte pro Zustand. Schritte, die das Intervall verlassen würden, sind dabei nicht enthalten. Der Fehler des Einzellaufes wurde durch $a = 400$ Läufe bestimmt. Für die längere Laufzeit können gemessener und berechneter Fehler nicht unterschieden werden.

stimmung beider Fehler sehr gut. Bei der kürzeren Laufzeit gilt dies nur für kleine und mittlere Dichten. Bei hohen Dichten liegt der gemessene Fehler oberhalb des berechneten, was auf Korrelationen zwischen den einzelnen Schritten weist. Dies ist jedoch nicht überraschend, da die Startkonfigurationen des neuen Intervalls aus den Endkonfigurationen des alten erzeugt werden und die hieraus resultierenden Korrelationen bei hohen Dichten nur sehr langsam zerfallen. Da der Fehler jedoch nach Gleichung (2.8)

richtig bestimmt wird, stören die Abweichungen nicht weiter. Der oben durchgeführte Vergleich hat dennoch durchaus praktische Relevanz. Er erlaubt zu testen, ob eine Konfiguration äquilibriert ist. Dies ist genau dann der Fall, wenn keinerlei Korrelationen zwischen den Abschnitten mehr auftreten und beide Graphen aufeinanderfallen. Die zu Grunde liegenden Verhältnisse sind dann statistisch unabhängig und es können keine Abweichungen zum Gauß'schen Fehlerfortpflanzungsgesetz festgestellt werden. Erreicht wird dies z.B. durch eine Erhöhung der Anzahl der großkanonischen Schritte (siehe Abb. 2.6), durch die Beimischung lokaler Schritte oder aber einer Erhöhung der Intervallgröße.

Neben dem statistischen Fehler kann bei der Berechnung eines Verhältnisses auch ein systematischer auftreten. Es wird sich im weiteren Verlauf zeigen, dass dieser unter normalen Simulationsbedingungen vernachlässigt werden kann. Zunächst soll jedoch sein Zustandekommen formal begründet werden. Die folgenden Betrachtungen beziehen sich auf den Spezialfall $\omega = 1$. Sei $H_1 = H(n_1)$ die Häufigkeit mit der Zustand n_1 besucht wird und $\langle H_1 \rangle$ der wahre Mittelwert von H_1. Für $\omega = 1$ ist $H = H_1 + H_2$ konstant und endlich. Eine Taylor–Entwicklung von $\frac{H_1}{H_2}$ um $\langle H_1 \rangle$ liefert:

$$\frac{H_1}{H_2} = \frac{\langle H_1 \rangle}{(H - \langle H_1 \rangle)} + \frac{H(H_1 - \langle H_1 \rangle)}{(H - \langle H_1 \rangle)^2} + \frac{H(H_1 - \langle H_1 \rangle)^2}{(H - \langle H_1 \rangle)^3} + \ldots \tag{2.10}$$

Mittelt man den oberen Ausdruck, wird der zweite Term auf der rechten Seite identisch Null. Es gilt:

$$\left\langle \frac{H_1}{H_2} \right\rangle - \frac{\langle H_1 \rangle}{\langle H_2 \rangle} = \frac{H(\langle H_1^2 \rangle - \langle H_1 \rangle^2)}{(H - \langle H_1 \rangle)^3} + \ldots \tag{2.11}$$

Hieraus folgt:

$$\left\langle \frac{H_1}{H_2} \right\rangle > \frac{P(n_1)}{P(n_2)} = \frac{\langle H_1 \rangle}{\langle H_2 \rangle}, \quad \text{aber} \tag{2.12}$$

$$\lim_{H \to \infty} \left\langle \frac{H_1}{H_2} \right\rangle \to \frac{P(n_1)}{P(n_2)}. \tag{2.13}$$

Eine (endliche) Messung eines Verhältnisses liefert nach (2.12) also immer einen zu großen Wert. Dies lässt sich auch an einem Beispiel nachvollziehen. Beim Wurf einer Münze erwartet man im Durchschnitt ein Verhältnis von $\frac{P(\text{Kopf})}{P(\text{Zahl})} = \frac{\langle H(\text{Kopf}) \rangle}{\langle H(\text{Zahl}) \rangle} = 1$. Bestimmt man den Wert jedoch durch die Mittelung der Verhältnisse aus jeweils $H = 10$ Würfen, erhält man einen Wert größer Eins. So ist 0.5 ($\frac{4}{6} + \frac{6}{4}$)=$\frac{13}{12} > 1$ – die Wertepaare (3,7) (2,8) und (1,9) liefern ebenfalls zu große Beiträge. Nach (2.13) werden Abweichungen vom wahren Verhältnis mit größer werdenen Messumfang H geringer (2.13). Das Wahrscheinlichkeitsverhältnis sollte deswegen immer als Verhältnis der Häufigkeitssummen über alle Läufe, bzw. Prozessoren und nicht als Summe über die verschiedenen Verhältnisse ermittelt werden. Gemäß (2.12) definiert man den syste-

matischen Fehler als:

$$\Delta_{\text{sys,rel}}\left(\frac{P(n_1)}{P(n_2)}\right) := \frac{\langle H_2\rangle}{\langle H_1\rangle}\left(\left\langle\frac{H_1}{H_2}\right\rangle - \frac{\langle H_1\rangle}{\langle H_2\rangle}\right). \tag{2.14}$$

Im Prinzip kann der systematische Fehler direkt nach Gleichung (2.14) bestimmt werden. In der Praxis ist dies jedoch nicht umsetzbar, da eine genaue Bestimmung der beiden Terme einen nicht zu verantwortenden Simulationsaufwand mit sich brächte, der besser in die Reduzierung der Fehler investiert wird. Man benötigt deswegen einen mathematischen Ausdruck, der sich direkt aus Gleichung (2.11) ergibt:

$$\Delta_{\text{sys,rel}}\left(\frac{P(n_1)}{P(n_2)}\right) = \frac{(H-\langle H_1\rangle)}{\langle H_1\rangle}\left(\frac{H(\langle H_1^2\rangle - \langle H_1\rangle^2)}{(H-\langle H_1\rangle)^3} + \ldots\right). \tag{2.15}$$

Insgesamt fällt der führende Term in Gleichung (2.15) mit $\frac{1}{H}$ ab. Die Annahme $H = H_1 + H_2 =$ konstant gilt nur für $\omega = 1$. Bei größeren Fenstern und einer festgelegten Gesamtzahl von Monte–Carlo–Schritten pro Fenster benötigt man eine Taylor–Entwicklung mit zwei Veränderlichen H_1 und H_2. Eine analoge Kalkulation wie oben liefert dann:

$$\Delta_{\text{sys,rel}} = \frac{\langle H_2\rangle}{\langle H_1\rangle}\left(\frac{\langle H_1\rangle}{\langle H_2\rangle^3}(\langle H_2^2\rangle - \langle H_2^2\rangle) - \frac{1}{\langle H_2\rangle^2}(\langle H_1H_2\rangle - \langle H_1\rangle\langle H_2\rangle) + \ldots\right). \tag{2.16}$$

Setzt man hier $H_2 = H - \langle H_1\rangle$ mit $H =$ konstant, erhält man wieder (2.15). Wie schon in (2.15) ist der Abfall proportional zum Kehrwert der Monte–Carlo–Schritte im Intervall. Da der statistische Fehler wie der Kehrwert der Wurzel der Gesamtschritte abfällt, gilt für den Einzelfehler:

$$\Delta_{\text{sys,rel}} \propto \Delta^2_{\text{stat,rel}}. \tag{2.17}$$

Die Überlegungen zur Fortpflanzung des systematischen Fehlers ähneln denen des statistischen. Innerhalb eines Intervalls kürzen sich alle Zwischenwerte in Gleichung (2.5) exakt weg und Fehler werden lediglich beim Übergang zum nächsten Simulationsabschnitt weitergegeben. Im Gegensatz zum statistischen Fehler sind systematische Fehler jedoch positiv korreliert. Unter Vernachlässigung von Mischtermen höherer Ordnung erhält man für die unnormierte Verteilung:

$$\Delta_{\text{sys,rel}}\left(P_j(j\omega)\right) = \sum_{i=1}^{j}\Delta_{\text{sys,rel}}\left(\frac{P_i(i\omega)}{P_i((i-1)\omega)}\right). \tag{2.18}$$

Unter Berücksichtigung von (2.9) folgt direkt, dass (2.17) im Prinzip auch für den Gesamtfehler gilt und der systematische Fehler in allen sinnvollen Simulationssituationen vernachlässigt werden kann.

Für $\omega = 1$ soll dies quantifiziert werden. Zur Überprüfung der Abschätzung (2.15), wurde zunächst eine exakte Gewichtsfunktion für das Wahrscheinlichkeitsverhältnis

$\frac{P(n=14)}{P(n=13)}$ durch einen langen Lauf bestimmt. Die angegebenen Teichenanzahlen entsprechen einer Simulation im Gaspeak des Referenzsystems (vgl. Abbildung 2.4 a). Anschliessend wurde für $H = 30, 50, 80, 200, 800$ und 2400 der systematische Fehler direkt (2.14) und nach (2.15) bestimmt. Bei Letzterem wurde nur der führende Term berücksichtigt. Die Läufe wurden so lange wiederholt, bis der systematische Fehler auf 1% genau bestimmt war. Abbildung 2.7 zeigt den systematischen Fehler des Ver-

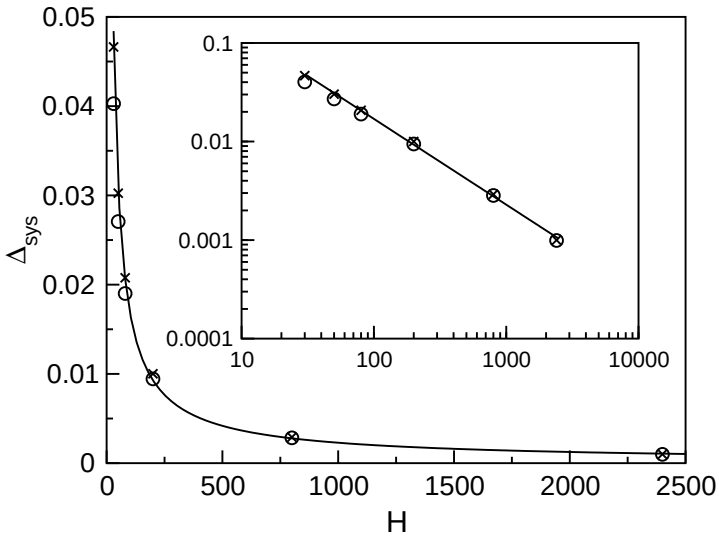

Abbildung 2.7: Systematischer Fehler von $\frac{P(14)}{P(13)}$ als Funktion der Anzahl von MC–Schritten H, die zur Bestimmung des Verhältnisses aufgewendet wurden. x entspricht dem wahren Fehler nach (2.14), ∘ der Näherung nach (2.15), wobei lediglich der führende Term berücksichtigt wurde. Die durchgezogene Linie ist ein „guide to the eye“. Inset: gleicher Graph in doppelt–logarithmischer Auftragung.

hältnisses als von Funktion von H in normaler und doppelt–logarithmischer Auftragung. Man erkennt deutlich, dass (2.15) selbst für extrem kleine H noch einen guten Schätzwert liefert. Da für normale Simulationen H in der Größenordnung von Millionen Monte–Carlo–Schritten liegt, ist die Abschätzung (2.15) für alle praktischen Belange völlig ausreichend, selbst wenn die Akzeptanzraten kleiner werden. In der doppelt–logarithmischen Darstellung deutet sich ein Abfall des Fehler mit etwa $\frac{1}{H}$ an. Zur Abschätzung der Größenordnung vergleicht man nun den systematischen Fehler eines Verhältnisses mit dem statistischen. Für ein festes H ist:

$$\Delta_{\text{stat,rel}}\left(\frac{P(n_1)}{P(n_2)}\right) = \frac{H\sqrt{\langle H_1^2\rangle - \langle H_1\rangle^2}}{\langle H_1\rangle(H - \langle H_1\rangle)}, \tag{2.19}$$

also von Ordnung $\frac{1}{\sqrt{H}}$. Berücksichtigt man in (2.8) lediglich den führenden Term, so

kann der Proportionalitätsfaktor in (2.17) direkt bestimmt werden. Es gilt dann:

$$\Delta_{\mathrm{sys,rel}} \approx \Delta^2_{\mathrm{stat,rel}} \frac{\langle H_1 \rangle}{H}. \tag{2.20}$$

Dieser Zusammenhang wird in Abbildung 2.8 verdeutlicht. Die durchgezogene Li-

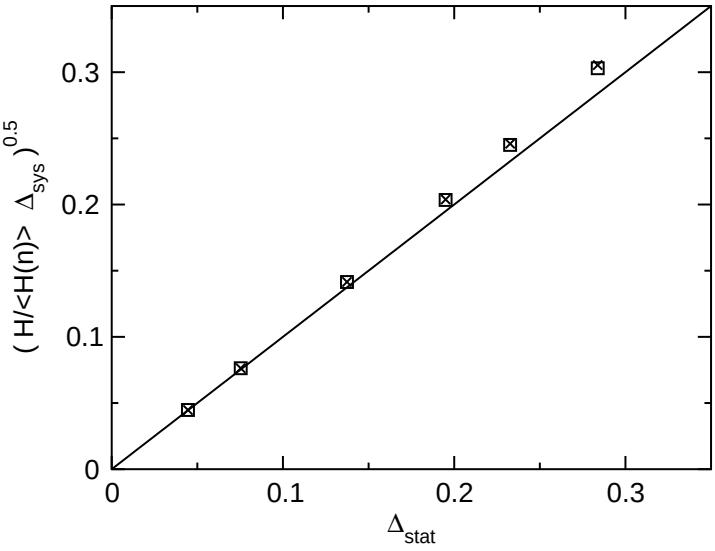

Abbildung 2.8: Abhängigkeit des relativen systematischen Fehlers vom statistischen Fehler. Die durchgezogene Linie entspricht Relation (2.20), d.h. bei der Bestimmung des systematischen Fehlers wurde lediglich der führende Term in (2.15) berücksichtigt. Bei x wurde der nach (2.14) gemessene Fehler in Relation zum statistischen gesetzt, während bei □ der systematische Fehler nach (2.15) unter Berücksichtigung von Termen höherer Ordnung (3,4 und 5) in $(H - \langle H(n) \rangle)$ ermittelt wurde.

nie entspricht (2.20), d.h. es wurde lediglich der führende Term von Gleichung (2.15) berücksichtigt. Der nach (2.14) bestimmte wahre systematische Fehler (x) liegt für extrem kleine Werte von H etwas oberhalb dieser Linie. Für $H > 200$ ist der Unterschied jedoch kaum noch wahrzunehmen. Da H in der Regel in der Größenordnung von Millionen Monte–Carlo–Schritten liegt, liefert (2.20) für alle sinnvollen Simulationssituationen mit $\omega = 1$ eine hervorragende Abschätzung. Setzt man Gleichung (2.20) in (2.18) ein und nimmt ferner an, dass $\frac{\langle H(n) \rangle}{H} \approx 0.5$ ist (gute Gewichtsfunktion), so gilt (2.20) auch für den Gesamtfehler. Ist z.B. $\Delta_{\mathrm{stat}} = 10\%$, dann erwartet man einen systematischen Gesamtfehler von $\Delta_{\mathrm{sys}} = 0.5\%$. Es sollte an dieser Stelle darauf hingewiesen werden, dass in Regionen mit extrem kleinen Akzeptanzraten (z.B. in der Nähe eines Glasübergangs) die Anzahl der unkorrelierten Messschritte trotz großem Hs sehr klein ausfallen kann. Der statistische Gesamtfehler sollte in jedem Fall kleiner als Eins sein.

2.3.2 Ein alternativer Ansatz

Es ist bei der Multiprozessorversion auch möglich, den Gesamtfehler im Voraus einzustellen. In dieser Variante berechnet das Programm nach einer festgelegten Anzahl von Monte–Carlo–Schritten den Einzelfehler und geht zum nächsten Fenster über, falls der Schätzwert des Einzelfehlers in der Simulation erstmals einen bestimmten Schwellenwert unterschreitet. Ist der Einzelfehler größer als der Schwellenwert, wird die Simulation im gleichen Abschnitt fortgesetzt. Im Folgenden geht man davon aus, dass der Einzelfehler in jedem Intervall gleich groß sein soll. Dann errechnet sich der Schwellenwert aus dem Gesamtfehlers der (unnormierten) Wahrscheinlichkeitsverteilung und Gleichung (2.9):

$$\Delta_{\text{stat,rel,einzel}} = \frac{\Delta_{\text{stat,rel,total}}}{\sqrt{m}.} \tag{2.21}$$

m steht hierbei für die Anzahl der Fenster, in die die Simulation unterteilt wurde (d.h. $m\omega = N$). Zur Überprüfung vergleicht man den nach (2.8) gemessenen mit dem nach (2.21) vorhergesagten Fehler.Da man hierbei zusätzliche Korrelationen auf Grund hoher Dichten vermeiden möchte (vgl. Abbildung 2.6), beschränkt man die Simulation auf einen Bereich niedriger Dichte. Wie erwartet, folgen die Fehlerkurven in etwa dem

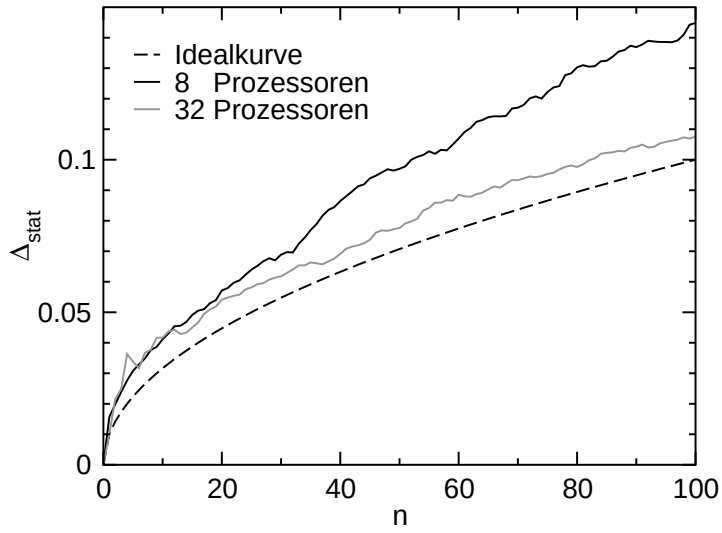

Abbildung 2.9: Statistischer Fehler als Funktion der Teilchenzahl. Die Simulationen wurden auf einem Parallelrechner mit 8 bzw. 32 Prozessoren bei Fenstergröße $\omega = 1$ durchgeführt. Zur Verbesserung der Statistik wurden die Läufe jeweils 250 mal wiederholt. Der Gesamtfehler eines einzelnen Laufs wurde auf 10% eingestellt. Die schwarze durchgezogene Linie beschreibt den erwarteten Verlauf nach (2.9).

Verlauf einer Wurzelfunktion. Abbildung 2.9 verdeutlicht aber auch das grundlegende Problem der Methode. Verwendet man zur Fehlerbestimmung lediglich eine kleine

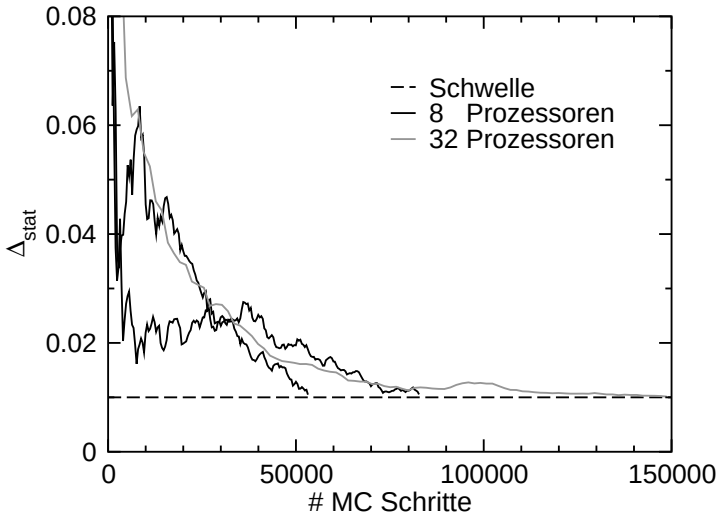

Abbildung 2.10: Statistischer Fehler als Funktion der Gesamtzahl von Monte–Carlo–Schritten. Das untersuchte Intervall war auf 13 und 14 Lennard–Jones–Teilchen im System beschränkt. Die Simulation lief auf 8 bzw. 32 Prozessoren. Der Schwellwert, bei dessen Unterschreiten die Simulation mit dem nächsten Intervall fortfährt, wurde auf 0.01 festgesetzt. Bei 32 Prozessoren sind die Schwankungen des Fehlers geringer. Als Folge benötigt man mehr Schritte zum ersten Unterschreiten der Schwelle.

Anzahl von Prozessoren, wird der Fehler durch (2.9) systematisch unterschätzt. Die Abweichung verringert sich jedoch mit steigender Anzahl von Prozessoren. Abbildung 2.10 liefert hierfür eine mögliche Begründung. Bei einer kleinen Anzahl von Prozessoren fluktuiert der Fehler sehr stark. Das Programm fährt jedoch mit dem nächsten Intervall fort, wenn der Fehlerschwellwert zum ersten Mal unterschritten wurde. Erhöht man die Anzahl der Prozessoren, nehmen die Fluktuationen ab und das System braucht im Mittel mehr Monte–Carlo–Schritte zur Unterschreitung der Schwelle.

Obwohl der gemessene Fehler nicht mit der Vorhersage aus (2.9) übereinstimmt, wird er nach (2.8) dennoch richtig bestimmt. Es empfiehlt sich jedoch, eine eher große Anzahl von Prozessoren (≥ 32) einzusetzen. Der Benutzer erhält durch die Methode prinzipiell die Möglichkeit, den ungefähren Fehler im Voraus einzustellen. Simuliert man mit einer festen Anzahl von Monte–Carlo–Schritten, ist man jedoch in jedem Fall auf der sicheren Seite.

2.3.3 Laufzeitvergleich

Neben der exakten Bestimmung des Fehlers, ist auch ein Vergleich der Laufzeiten der miteinander konkurrierenden Methoden interessant und wichtig. Alle bisher existie-

renden Techniken dienen bei genauerem Hinsehen lediglich der Erzeugung von Gewichtsfunktionen. „Histogram–reweighting“ (vgl. Kapitel 2.1) schneidet hierbei am Schlechtesten ab, da eine Gewichtsfunktion fernab des kritischen Punktes nur durch eine mühsame und rechenzeitintensive Abfolge von Simulationen und Extrapolationen generiert werden kann. Mit dem Wang–Landau–Algorithmus lässt sich hingegen prinzipiell an jeder Stelle des Phasendiagramms eine Gewichtsfunktion erzeugen (vgl. Kapitel 2.2). Da jedoch $w(n)$ in jedem Schritt angepasst wird, lässt sich der Fehler nur schwer abschätzen. Deshalb sollte sich immer eine reguläre multikanonische Simulation mit konstanten Gewichten anschließen. Zudem ist bei dieser Methode nicht a priori klar, wann ein günstiger Abbruchwert erreicht wird. Eine weitere Technik zur Bestimmung von Gewichtsfunktionen ist die so genannte Multikanonische Rekursion [48]. Hierbei wird, vereinfacht ausgedrückt, $w(n)$ nach einem kurzen Lauf gemäß Gleichung (2.1) anpasst und so nach und nach das gesamte Histogramm erschlossen. Eine direkte Bestimmung des Fehler ist jedoch aufgrund der unterschiedlichen Dynamiken in den verschiedenen Simulationsabschnitten ebenfalls problematisch. Zusammenfassend setzt sich die benötigte Rechenzeit all dieser Verfahren aus der Zeit zur Erzeugung einer Gewichtsfunktion plus der Zeit eines regulären multikanonischen Produktionslaufs zusammen. Im Gegensatz hierzu geht bei der neu vorgestellten Methode jeder Lauf in Bestimmung des Fehler mit ein. Der Simulationsaufwand zur Erzeugung einer Gewichtsfunktion entfällt folglich. Es bleibt jedoch anzumerken, dass dieser in der Regel klein gegen den Aufwand eines Produktionslaufs ist, da für eine multikanonische Simulation im Grunde ein grober Schätzwert der Verteilung genügt. Der hieraus resultierende Vorteil ist somit gering.

Als Nächstes muss geklärt werden, wie die Rechenzeit mit der Fenstergröße skaliert. In der Standardliteratur [43], [20] wird wie folgt argumentiert. Sei m die Anzahl der Intervalle in die die Simulation unterteilt wird und ω die Fenstergröße. Sei ferner τ^1_{cpu} die Zeit, die man zur Erzeugung der Wahrscheinlichkeitsverteilung in einem Intervall mit vorher festgelegtem Fehler benötigt. Vernachlässigt man Dichteeffekte (vgl. Abbildung 2.5) und unterstellt der gewichteten Simulation die Dynamik eines „random walks“ im Dichteraum, so folgt für das einzelne Fenster: $\tau^1_{\text{cpu}} \propto \omega^2$ und insgesamt $\tau_{\text{cpu}} \propto m\omega^2$. Besteht die Simulation lediglich aus einem Abschnitt, d.h. $m' = 1$, dann folgt $\tau'_{\text{cpu}}=(m\omega)^2=m\tau_{\text{cpu}}$. Der Simulationsaufwand wäre demnach um Ordnung m größer.

Da entsprechende Simulationen fehlen, wurde zunächst eine gute Gewichtsfunktion erzeugt und anschließend Umbrella–Sampling–Simulationen mit unterschiedlichen Fenstergrößen ($\omega = 1, 20$ und 218) durchgeführt. Die Anzahl der Monte–Carlo–Schritte pro Zustand betrug 2 Millionen. Rein rechnerisch benötigt man für Fenstergröße $\omega = 1$ fast die doppelte Gesamtzahl von Monte–Carlo–Schritten als für $\omega = 218$. Diese zusätzlichen Schritte beanspruchen jedoch keine zusätzliche Laufzeit, da für $\omega = 1$ die Hälfte der Sprünge das jeweilige Simulationsfenster verlassen würden und ohne weiteren Rechenaufwand abgelehnt werden. Es wurden für jede Fenstergröße

$a = 400$ Läufe durchgeführt. Alle Fehler wurden abermals mit $\sqrt{a}$ multipliziert, da man letztlich am Fehler eines einzelnen Laufs interessiert ist und dieser durch die 400 Läufe lediglich mit guter statistischer Genauigkeit bestimmt werden soll. Alle Wahrscheinlichkeitsverteilungen wurden vor der Berechnung der Fehler auf Eins normiert. Abbildung 2.11 zeigt deutlich, dass der Fehler im Widerspruch zu der Argumentation in [43] und [20] für alle untersuchten Fenstergrößen gleich ist. Er ist somit unabhängig

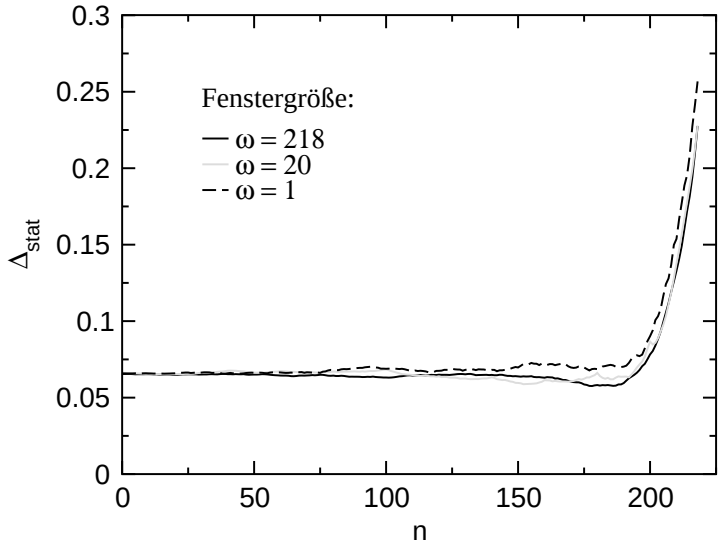

Abbildung 2.11: Vergleich des Fehlers der (normierten) Wahrscheinlichkeitsverteilung für Umbrella–Sampling–Simulationen mit unterschiedlichen Fenstergrößen ($\omega = 1, 20$ und 218) und 2 Millionen Monte–Carlo–Schritten pro Zustand. Der Fehler eines einzelnen Laufs wurde jeweils durch Mittelung über $a = 400$ Läufe bestimmt. Die Gewichtsfunktion für $\omega = 1$ ist 15 mal schlechter als für $\omega = 20$ und 218. Alle Simulationen liefern in etwa den gleichen Fehler.

von der Anzahl der Abschnitte, in die die Simulation unterteilt wurde. Ferner erkennt man, dass die Qualität von $w(n)$ nur eine untergeordnete Rolle spielt. Obwohl der Fehler in der Gewichtsfunktion für $\omega = 1$ deutlich schlechter als für $\omega = 20$ und 218 ist (maximale Abweichung 30% statt 2%), liegt die Fehlerkurve nur knapp oberhalb der Kurven mit der besseren Gewichtsfunktion.

Die Diskrepanz zwischen Simulation und theoretischen Überlegungen in [43] und [20] lässt sich auf einfache Weise beseitigen. In der oberen Argumentation ist der Fehler in jedem der m Intervalle gleich dem Gesamtfehler für $m' = 1$. Dies ist jedoch nicht richtig, da sich der statistische Fehler von Fenster zu Fenster nach (2.9) fortpflanzt. Bei m Fenstern muss der statistische Einzelfehler in jedem Intervall folglich um $O(\sqrt{m})$ kleiner sein als für $m' = 1$, um auf denselben Gesamtfehler zu kommen. Die Rechenzeit muss demnach in jedem Fenster um den Faktor m vergrößert werden. Der Gesamtaufwand beträgt also auch bei m Fenstern $m(m\omega^2) = N^2$ und ist somit un-

abhängig von der Art der Unterteilung der Simulation. Voraussetzung ist jedoch das Vorliegen einer akzeptablen Gewichtsfunkion.

Wiederholt man die gleichen Läufe für sehr große Boxen, geraten die Simulationen ins Stocken. Neuere Überlegungen [37], [49] zeigen, dass das hier besprochene System bei kleinen Dichten einen Phasenübergang erster Ordnung durchführt und bei konstanter Teilchenzahl n zwischen homogener Gasphase und einer Phase mit einem Tropfen wechseln möchte. Beide Phasen sind jedoch von einer Barriere getrennt, die mit wachsender Boxgröße immer größer wird. Großkanonische Schritte können das System daher nicht mehr effizient von einer Phase in die andere überführen. Für kleine Systeme überlappen sich beide Peaks vollständig. Der Übergang von übersättiger Gasphase zur Tröpfchenphase ist fließend und bremst die Simulation nicht aus. Der Phasenübergang ist im Prinzip systemspezifisch. Er tritt jedoch auch bei verwandten Systemen wie dem Ising–Modell auf [37] und wird nochmals ausführlich in Kapitel 5 diskutiert. Möchte man den hier beschriebenen Übergang effizient großkanonisch simulieren, muss die Energie in die Gewichtsfunktion integriert werden. Statt $w(n)$ benötigt man dann eine zweidimensionale Gewichtsfunktion $w(n,E)$. Eine ähnliche Aufgabe stellt sich auch bei der Simulation von Polymermischungen.

2.3.4 Erweiterung auf Polymermischungen

Möchte man Polymere statt Lennard–Jones–Teilchen simulieren, bleibt die prinzipielle Vorgehensweise gleich. Die Dichte ist weiterhin Ordnungsparameter und n beschreibt die Anzahl der Ketten im System. Auch die Erweiterung auf kompressible Polymermischungen ähnelt den in Kapitel 2.1 und 2.2 vorgeschlagenen Vorgehensweisen. Erwartet man in erster Linie eine Koexistenz zweier Polymerphasen, wählt man die Fenster entlang der Polymerachse und lässt die Lösungsmittelteichen ungewichtet mitlaufen. Erwartet man im Wesentlichen eine Koexistenz zweier Phasen von Lösungsmittelteilchen, extrapoliert man entlang dieser Achse und lässt die Polymere ungewichtet mitlaufen. Zusätzlich besteht auch noch die Möglichkeit, eine komplette Isotherme des Mischsystems auf einmal zu generieren. Hierzu simuliert man z.B. zunächst das reine Polymersystem (von 0 bis N_{pol} Ketten) ohne Lösungsmittel. Im Anschluss führt man $N_{\mathrm{pol}}+1$ Simulationen parallel zur Lösungsmittelachse und jeweils festgehaltener Kettenzahl $(0-N_{\mathrm{pol}})$ durch und verknüpft diese entlang der Polymerachse. Simulationsfenster, die sowohl eine bestimmte Anzahl von Polymeren als auch Lösungsmittelteichen zulassen, sind ebenfalls denkbar. Die Verknüpfung erfolgt dann entlang des Randes. Die chemischen Potentialpaare für Koexistenz können im Anschluss durch Extrapolation aus dem Datensatz gewonnen werden.

Der in diesem Kapitel vorgestellte Algorithmus bietet substantielle Vorteile. Die Simulation kann direkt und ohne vorherige Erzeugung einer Gewichtsfunktion an jeder Stelle des Phasendiagramms gestartet werden, die großkanonische Simulationen

zulässt. Zudem wurde der mühsame Prozess des Zusammensetzens verschiedener Simulationsfenster automatisiert und in die Simulation integriert. Zum ersten Mal wurde eine ausführliche Fehleranalyse der Umbrella–Sampling–Methode durchgeführt, die auch auf verwandte Schemata angewendet werden kann. Insbesondere wurde gezeigt, dass der Fehler der neuen Methode vollständig verstanden ist. In diesem Zusammenhang stellte sich heraus, dass entgegen der herrschenden Meinung der Fehler unabhängig von der gewählten Fenstergröße ω ist. Die Methode kann also direkt für $\omega = 1$ implementiert werden. Sie ist ferner einfach zu parallelisieren und leicht erweiterbar. Der Algorithmus kann auf eine Vielzahl von Systemen, beginnend mit einfachen Gittermodellen bis hin zu Kontinuumsmodellen beliebiger Komplexität, angewendet werden. Mögliche Fragestellung, die sich mit Hilfe der neuen Methode bearbeiten lassen, umfassen das Studium von Phasenübergängen, der Nukleation, der Bewegung von Partikeln durch Nanoröhren und vieles mehr.

Kapitel 3

Phasendiagramme und Grenzflächenspannungen

Die in den vorangegangen Abschnitten vorgestellten Methoden erlauben nun eine Überprüfung des in Kapitel 1.1 vorgeschlagenen Modells durch einen Vergleich mit experimentellen Daten. Hierbei kommt der Bestimmung der Phasendiagramme entscheidende Bedeutung zu. Können die experimentellen Vorgaben nicht in befriedigender Weise reproduziert werden, ist das Modell für eine Beschreibung des Referenzsystems ungeeignet. Die Bestimmung der Phasendiagramme bildet darüber hinaus auch die notwendige Voraussetzung für alle weiteren Betrachtungen in dieser Arbeit: den Test einer analytischen Störungstheorie (TPT1) in Kapitel 4 [50] und den in Kapitel 5 folgenden Betrachtungen zur Keimbildung. Auch die Analyse der Phasendiagramme an sich, insbesondere der des Mischsystems, ist überaus interessant. Zum Einen lässt sich über die Anpassung eines einzelnen Parameters im verwendeten Modell der Übergang von verschieden Phasendiagrammtypen nachweisen. Dies hat profunden Einfluss auf das Auftreten von physikalisch interessanten Effekten (z.B. einer Flüssig–flüssig–Entmischung). Zum Anderen ermöglicht die starke Vergröberung erstmals die Bestimmung eines (fast) vollständigen Phasendiagramms inklusive kritischer Linien im Rahmen von Simulationen. Die Ermittlung der Grenzflächenspannung liefert darüber hinaus Anhaltspunkte für die weitere Untersuchung des Nukleationsverhaltens. Zunächst müssen jedoch die Wechselwirkungsparameter der Potentiale ermittelt werden.

3.1 Bestimmung der Wechselwirkungsparameter

In Kapitel 1.1 wurde ein vergröbertes Modell vorgeschlagen, welches Hexadekan auf eine kompressible Kette von fünf Lennard–Jones–Teilchen und CO_2 auf ein einzelnes Lennard–Jones–Teilchen abbildet. Ein Vergleich mit experimentellen Daten wird hierbei durch eine geeignete Wahl von ε und σ in Gleichung (1.1) und (1.2) ermöglicht. ε_1, σ_1 beschreiben in diesem Zusammenhang CO_2–CO_2–, ε_5, σ_5 Polymer–Polymer– und ε_{1-5}, σ_{1-5} CO_2–Polymer–Wechselwirkungen. Bei der Simulation von reinem Hexadekan bzw. CO_2 setzt man zunächst ε und σ gleich Eins. Dadurch erhält man Temperatur, Monomer–Teilchenzahldichte, Druck und Grenzflächenspannung in Lennard–Jones–Einheiten. Zwischen den Zahlenwerten der Simulation und den entsprechenden experimentellen Einheiten herrscht folgender Zusammenhang:

$$k_B \cdot T_{exp} = T_{LJ} \cdot \varepsilon \tag{3.1}$$

$$\rho_{exp} \cdot \frac{N_A}{\text{Molgewicht}} = \frac{\rho_{LJ}}{\text{Anzahl der Monomere / Kette}} \cdot \frac{1}{\sigma^3} \tag{3.2}$$

$$p_{exp} = p_{LJ} \cdot \frac{\varepsilon}{\sigma^3} \tag{3.3}$$

$$\gamma_{exp} = \gamma_{LJ} \cdot \frac{\varepsilon}{\sigma^2} \tag{3.4}$$

Die Anpassung der Simulation an das Experiment erfolgt für reines Hexadekan und CO_2 jeweils am kritischen Punkt (Tabelle 3.1, die experimentellen Werte wurden Referenzen [51], [52], [53], [54] und [55] entnommen). Der Vergleich der kritischen

CO_2	T_c	ρ_c	p_c
SIM	$0.999\ \frac{\varepsilon}{k}$	$0.32\ \frac{1}{\sigma^3}$	$0.088\ \frac{\varepsilon}{\sigma^3}$
EXP	$304\ K$	$0.464\ \frac{g}{cm^3}$	$73.87\ bar$

$C_{16}H_{34}$	T_c	ρ_c	p_c
SIM	$1.725\ \frac{\varepsilon}{k}$	$0.27\ \frac{1}{\sigma^3}$	$0.022\ \frac{\varepsilon}{\sigma^3}$
EXP	$723\ K$	$0.219\ \frac{g}{cm^3}$	$13.98\ bar$

Tabelle 3.1: Kritische Parameter für CO_2 und $C_{16}H_{34}$

Temperaturen von Simulation (Kapitel 1.4) und Experiment liefert dann nach (3.1) ε_1 bzw. ε_5, σ_1 und σ_5 bestimmt man analog durch Vergleich der kritischen Dichten nach Gleichung (3.2) (Tabelle 3.2). Nach (3.1)-(3.4) können nun auch jenseits des kritischen

ε_1	ε_5	$\frac{\varepsilon_1}{\varepsilon_5}$	σ_1	σ_5	$\frac{\sigma_1}{\sigma_5}$
$4.201 \cdot 10^{-21}$J	$5.787 \cdot 10^{-21}$J	0.726	$3.693 \cdot 10^{-10}$m	$4.523 \cdot 10^{-10}$m	0.816

Tabelle 3.2: Wechselwirkungsparameter

Punktes Simulationsdaten mit experimentellen Daten verglichen werden (Tabelle 3.3). Die Übereinstimmung wird dabei tendenziell umso schlechter ausfallen, je weiter der Messpunkt vom kritischen Punkt entfernt ist (vgl. auch Abbildung 3.2). Nach Tabelle

	$T_{LJ}[\frac{\varepsilon}{k}] \rightarrow T_{exp}[K]$	$\rho_{LJ}[\frac{1}{\sigma^3}] \rightarrow \rho_{exp}[\frac{g}{cm^3}]$
CO_2	304.32	1.45
$C_{16}H_{34}$	419.15	0.81
	$p_{LJ}[\frac{\varepsilon}{\sigma^3}] \rightarrow p_{exp}[bar]$	$\gamma_{LJ}[\frac{\varepsilon}{\sigma^2}] \rightarrow \gamma_{exp}[\frac{mN}{m}]$
CO_2	833.92	30.80
$C_{16}H_{34}$	625.45	28.29

Tabelle 3.3: Umrechnungsfaktoren: LJ–Einheiten → experimentelle Einheiten

3.1 und 3.3 ist die Abweichung vom experimentell bestimmten kritischen Druck für CO_2 und Hexadekan zu den Simulationsdaten jeweils kleiner als 2%. Dieser erste unabhängige Test kann als Indiz für die Gültigkeit des Modells gewertet werden. Für die Erweiterung auf Polymer–CO_2–Mischsysteme bestimmt man ε_{1-5} und σ_{1-5} nach der modifizierten Lorentz–Berthelot–Regel (1.3), (1.4). Die Mischsystem–Simulationen zur Bestimmung von ξ wurden in Einheiten des Fünfmer–Systems durchgeführt, d.h. $\varepsilon_5 = \sigma_5 = 1$, $\varepsilon_1 = 0.726$, $\sigma_5 = 0.816$ (vgl. Tabelle 3.2), $\varepsilon_{1-5} = \xi \cdot \sqrt{\varepsilon_1 \varepsilon_5} = \xi \cdot 0.852$, $\sigma_{1-5} = 0.5 \cdot (\sigma_1 + \sigma_5) = 0.908$.

3.2 Hexadekan und CO_2

Die Betrachtung der einzelnen Komponenten ist für die Diskussion der Mischsysteme in vielerlei Hinsicht wichtig. Zum Einem benötigt man die kritischen Parameter wie oben gezeigt zur Modellierung der Wechselwirkungen im Mischsystem. Zum Anderen sind die Phasendiagramme und kritischen Parameter bereits Teil des Zwei–Komponenten–Phasendiagramms.

Nach der Gibbs'schen Phasenregel benötigt man zur vollständigen Beschreibung eines Ein–Komponenten–Systems im Ein–Phasen–Gebiet zwei (1+2-1) intensive Variablen, z.B. Temperatur und Dichte oder Temperatur und Druck. Koexistieren zwei Phasen, reduziert sich die Zahl der Freiheitsgrade auf Eins (1+2-2), d.h. die Temperatur ist z.B. eine Funktion der Dichte. Diese Koexistenzlinie bezeichnet man auch als Binodale (vgl. Abbildung 3.1). Die Binodale unterteilt das Phasendiagramm grob in zwei Bereiche. Oberhalb der Binodalen kann lediglich eine einzelne Phase existieren. An der Binodalen herrscht Koexistenz zwischen Gas– und Flüssigphase und unterhalb befindet sich die Mischungslücke, welche man aus thermodynamischen Erwägungen ebenfalls in zwei Bereiche aufteilt. Ein Sprung vom Ein–Phasen–Gebiet

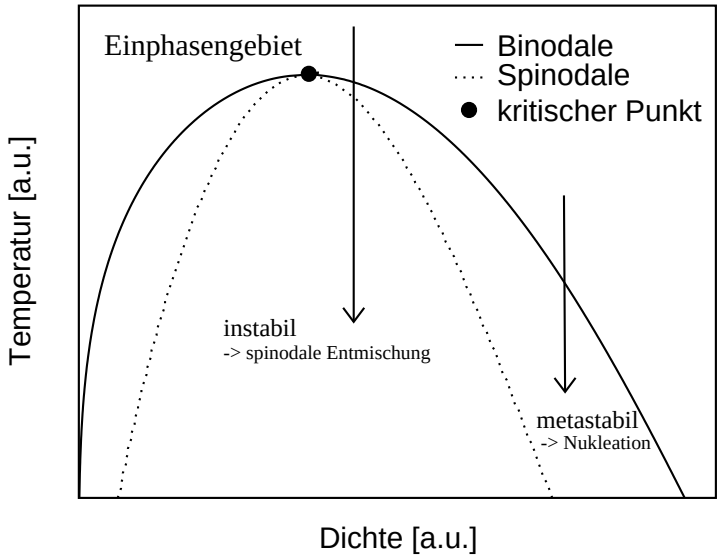

Abbildung 3.1: Phasendiagramm (T, ρ) eines Ein–Stoff–Systems

in den metastabilen Bereich bewirkt eine Phasentrennung durch Nukleation, d.h. das System muss zunächst eine endliche Barriere in der freien Energie überwinden bevor Phasenseparation auftritt. An der Binodalen divergiert die Barriere und an der Spinodalen verschwindet sie. Innerhalb des instabilen Gebiets verläuft die Entmischung deswegen spontan. Den dazugehörigen Mechanismus bezeichnet man als „spinodale Entmischung“. Sowohl Binodale als auch Spinodale treffen sich am kritischen Punkt. Nukleation und am Rande auch spinodale Entmischung werden nochmals in Kapitel 5 behandelt.

Abbildung 3.2 zeigt die in der Simulation bestimmte Binodale von CO_2 und $C_{16}H_{34}$ im Vergleich mit dem Experiment [56], [57]. Für CO_2 wurde zusätzlich die Lage der Spinodalen bestimmt. Da diese experimentell nicht zugänglich ist, erfolgt eine Diskussion im Rahmen von Kapitel 4 im Vergleich mit entsprechenden analytischen Berechnungen. Die Übereinstimmung der Binodalen mit dem Experiment fällt für Hexadekan erwartungsgemäß besser aus als für CO_2. Die Diskrepanzen in der Flüssigphase von CO_2 lassen sich durch die starke Vergröberung im verwendeten Modell erklären. So werden weder die aus der Ladungsverteilung resultierenden Multipolmomente noch die gestreckte Form des Moleküls im Modell adäquat berücksichtigt. Man beachte, dass in Abbildung 3.2 durch die Anpassung des kritischen Punktes lediglich Temperatur– und Dichteskalen aufeinander normiert wurden. Dies garantiert jedoch keineswegs, dass auch die kritische Amplitude des Ordnungsparameters wie hier in etwa mit dem Experiment übereinstimmt (vgl. mit Kap. 1.5).

Der Druck wurde in der Simulation mittels des Virialsausdrucks bestimmt [58]. Der

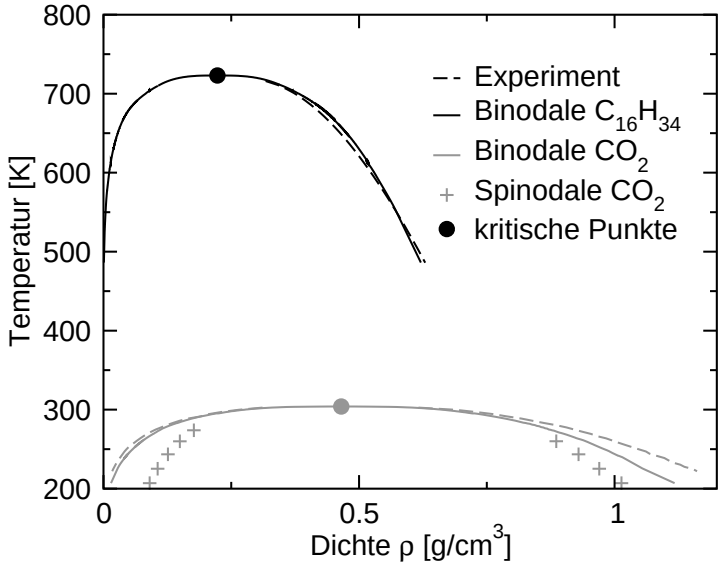

Abbildung 3.2: Binodale von reinem CO_2 und $C_{16}H_{34}$ im Vergleich mit experimentellen Messungen. Für CO_2 wurde ebenfalls die Spinodale bestimmt. Die zur Erstellung des Graphen benötigten Simulationsdaten finden sich in Tabellen A.1 und A.2 im Anhang A.

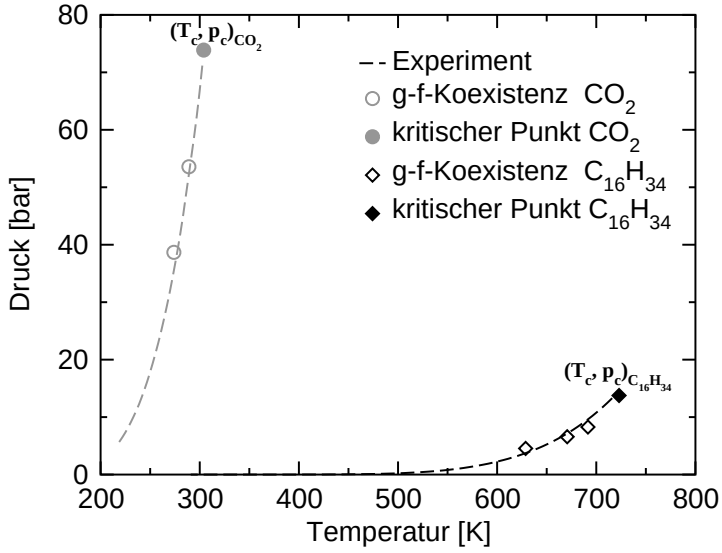

Abbildung 3.3: pT–Phasendiagramm für reines CO_2 bzw. $C_{16}H_{34}$.

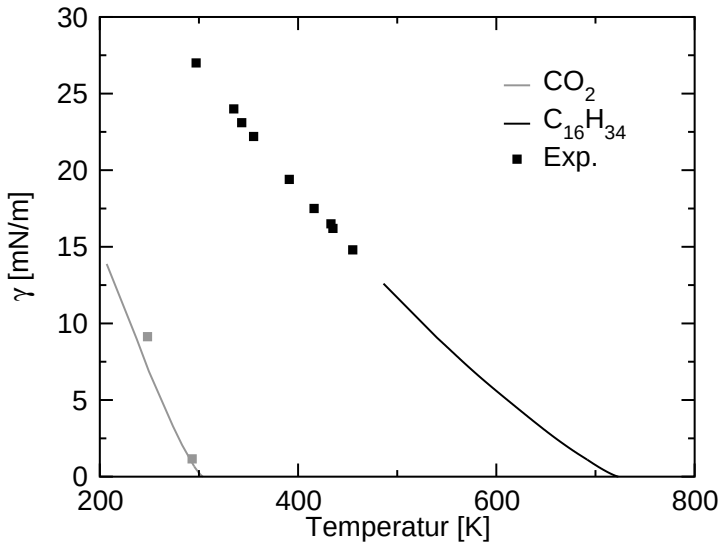

Abbildung 3.4: Grenzflächenspannung von reinem CO_2 und $C_{16}H_{34}$ (vgl. mit Abb. 1.13).

Vergleich mit experimentellen Daten entlang der Gas–flüssig–Koexistenzlinie (Abbildung 3.3) liefert eine gute Übereinstimmung obwohl keine zusätzlichen Parameter mehr eingingen. Diese Darstellung ist bereits Teil des Mischsystem–Phasendiagramms in Abbildung 3.7. Als Vorbereitung zur Analyse von Phasenseparationen wurde zusätzlich die Grenzflächenspannung des Gas–flüssig–Übergangs als Funktion der Temperatur bestimmt (Abbildung 3.4). Hierbei ergibt sich ein ähnliches Bild wie bei den Binodalen. Obwohl sich bei den Polymeren der Messbereich von Experiment [59], [60], [51] und Simulation nicht überschneidet, würde eine Extrapolation der Simulationsdaten eine sehr gute Übereinstimmung liefern. Für CO_2 ist das Ergebnis abermals schlechter. Wie schon bei den Koexistenzlinien tritt im kritischen Gebiet eine nicht–triviale kritische Amplitude auf, die vom Modell annähernd richtig vorausgesagt wird (vgl. mit Kap. 1.6).

Insgesamt sind die Betrachtungen zu den statischen Gleichgewichtseigenschaften überaus zufriedenstellend. Für Hexadekan wird in allen Fällen eine sehr gute Übereinstimmung mit experimentellen Daten erzielt, die an die Qualität von atomistischen Modellen heran reicht (vgl. z.B. mit [61]). Die Güte der Ergebnisse überrascht, da im vergröberten Modell auf alle atomistischen Details verzichtet wurde. Es bestätigt sich demnach die Grundannahme, dass Gleichgewichtseigenschaften von Alkanen in erster Linie durch deren grobe Struktur bestimmt werden.

3.3 Hexadekan–CO_2–Mischsysteme

Zwei–Komponenten–Systeme sind in ihrem Phasenverhalten erheblich komplexer als Ein–Stoff–Systeme. Bei Letzteren genügen zur vollständigen Beschreibung die Angabe von z.B. Temperatur und Dichte. Bei einem Zwei–Komponenten–System wird nach der Gibbs'schen Phasenregel eine weitere intensive Größe (2+2-1) benötigt. In dem dieser Arbeit zu Grunde liegenden Schema von *Scott* und *van Konynenburg* (vgl. mit Ref. [62], [63]) ist dies der Druck. Zudem verwendet man statt der Dichte einer Komponente eine Kombination beider Dichten, den so genannten Molbruch:

$$x(A) = \frac{n_A}{n_A + n_B}, \tag{3.5}$$

n_A und n_B geben die Anzahl der im System vorhandenen Teilchen von Komponente 1 bzw. 2 an. Ein typisches Mischsystem–Phasendiagramm als Funktion von p, T und x findet sich auf der linken Seite von Abbildung 3.5. Es enthält die Koexistenzlinien der

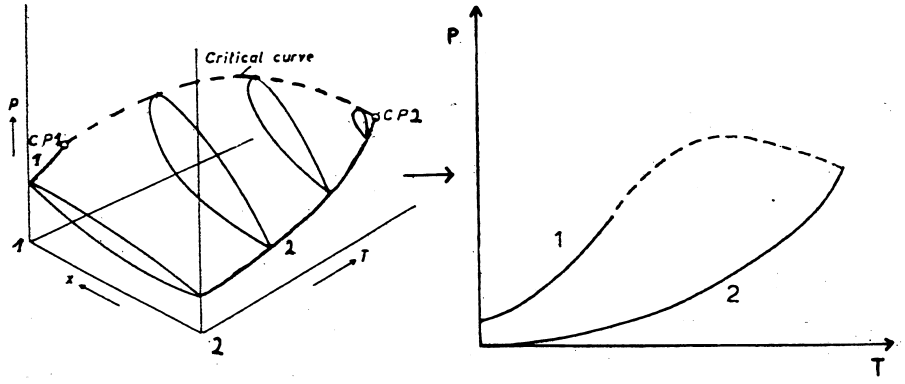

Abbildung 3.5: Projektion eines dreidimensionalen Mischsystem–Phasendiagramms auf die pT–Ebene (Typ I). Durchgezogene Linien mit Beschriftung 1 und 2 kennzeichnen die Koexistenzlinien von Komponente 1 und 2. Beide Linien enden im jeweiligen kritischen Punkt und liegen in der (p,T,x)–Darstellung in der $x = 1$ bzw. $x = 0$ Ebene. Man beachte, dass bei isothermen Schnitten der Gas–flüssig–Übergang im Mischsystem durch eine Tau– und eine Siedelinie begrenzt wird (geschlossene Kurven bei konstanter Temperatur in der (p,x)–Ebene der linken Abbildung). Ist die Temperatur größer als die kritische Temperatur T_c von Stoff 1, laufen Tau– und Siedelinien (bei Typ I) in einem Maximum, dem kritischen Punkt $(p_c(T), x_c(T))$ mit $x_c < 1$ zusammen. Die Linie dieser kritischen Punkte ist gestrichelt gezeichnet. Die linke Darstellung stammt aus [64], die rechte aus [65].

reinen Komponenten in der Darstellung von Abbildung 3.3. In der $x = 1$ Ebene, d.h. im System befinden sich ausschließlich Teilchen der Sorte 1, liegt die Koexistenzlinie von Komponente 1. Analog findet man für $x = 0$ die Binodale von Komponente 2.

Unterschiede im Phasenverhalten ergeben sich durch Unterschiede im Bereich von $0 < x < 1$. Im einfachsten Fall werden beide kritischen Punkte der Reinsysteme durch eine Linie von kritischen Punkten des Mischsystems verbunden.

Im Folgenden soll das unterschiedliche Phasenverhalten klassifiziert werden. Insgesamt unterscheidet man sechs verschiedene Typen von Phasendiagrammen. Im Rahmen dieser Arbeit werden jedoch lediglich die für das System Alkan–CO_2 relevanten Typen I–IV kurz vorgestellt. Eine ausführlichere Darstellung findet sich u.A. in Ref. [65] und [64]. Aus Gründen der Übersichtlichkeit betrachtet man in der Regel Projektionen der dreidimensionalen Diagramme auf die pT–Ebene. Diese enthalten, wie in Abbildung 3.5 dargestellt, bereits alle wichtigen Merkmale. So erkennt man sowohl die beiden Koexistenzlinien (1 und 2) als auch die kritische Linie, die die beiden kritischen Punkte der Reinsysteme verbindet.

Ein solches Phasendiagramm ist vom Typ I. Die Phasenzusammensetzung knapp unterhalb der kritischen Linie ändert sich hierbei kontinuierlich von einer Gas–flüssig–Koexistenz von Komponente 1 (in der Nähe des kritischen Punktes von Komponente 1) zu einer Gas–flüssig–Koexistenz von Komponente 2 (in der Nähe des kritischen Punktes von Komponente 2). Dies wird sich im späteren Verlauf als wichtiges Unterscheidungskriterium erweisen. Zusätzlich sind flüssige Phasen grundsätzlich mischbar. Phasendiagramme vom Typ I findet man in der Regel bei Mischungen von chemisch ähnlichen Substanzen mit nahe beieinander liegenden kritischen Punkten wie z.B. Argon+Krypton [66], Methan+Ethan [67], [68], CO_2+O_2 [69], [70]. Bei Alkan–CO_2–Mischungen erhält man Typ–I–Diagramme bis n–Hexan [71].

Phasendiagramme vom Typ II (vgl. Abbildung 3.6 b) weisen als zusätzliches Merkmal eine Flüssig–flüssig–Unmischbarkeit (Komponente 1 – Komponente 2) bei Temperaturen unterhalb der kritischen Temperatur der leichter flüchtigen Komponente auf. Diese zusätzliche kritische Linie mündet in einem oberen kritischen Endpunkt (UCEP), an dem sich eine weitere Gasphase anschließt. Von dort aus erstreckt sich eine Drei–Phasen–Koexistenzlinie hin zu niedrigeren Drücken und Temperaturen. Beispiele dieses Verhaltens findet man bei Mischungen von n–Heptan bis n–Undekan [71], [72] mit CO_2.

Je schlechter die Mischbarkeit der beiden Komponenten wird, desto weiter verschiebt sich die kritische Flüssig–flüssig–Linie zu höheren Temperaturen bis sie schließlich in der kritischen Gas–flüssig–Linie einmündet. Bei Phasendiagrammen des Typs III (Abb. 3.6 c) ändert sich im Gegensatz zu Typ I oder II die Zusammensetzung entlang der kritischen Linie – mit dem kritischen Punkt des schwerer flüchtigen Systems beginnend von einer Gas–flüssig–Koexistenz dieser Komponente zu einer Koexistenz der beiden flüssigen Phasen (von Komponente 1 und 2). Die Temperatur des UCEP ist nun höher als die kritische Temperatur der leichter flüchtigen Komponente. Beide Punkte werden durch eine in der Regel kurze, kritische Linie verbunden. Das von uns betrachtete Modellsystem Hexadekan–CO_2 lässt sich Typ III zuordnen [72], [73].

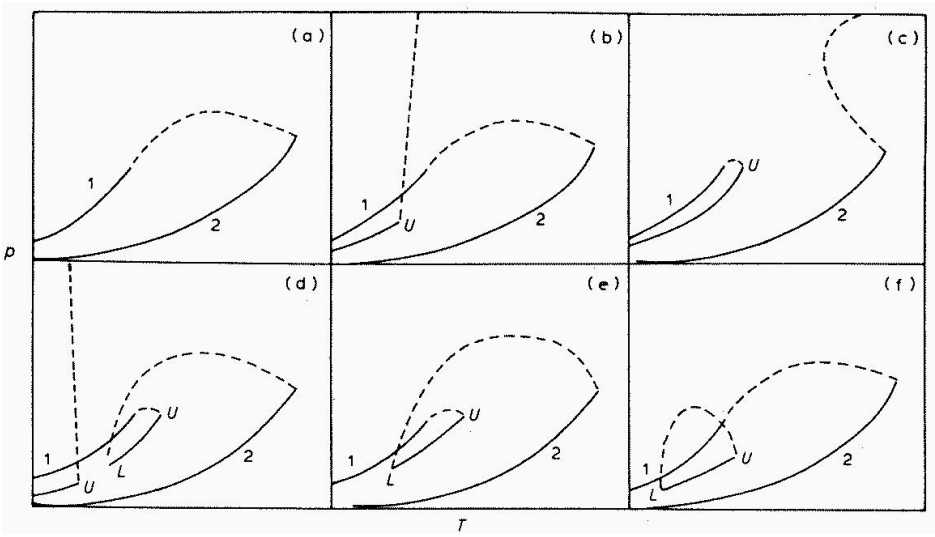

Abbildung 3.6: Die sechs Phasendiagrammtypen für Zwei–Komponenten–Systeme. Zu Unterscheidungszwecken verwendet man statt der dreidimensionalen (p,T,x)–Darstellungen, die Projektionen auf die pT–Ebene. Die Koexistenzlinien von Komponente 1 und 2, die im kritischen Punkt der jeweiligen Komponente enden, sind als durchgezogene Linien (mit Beschriftung 1 und 2) gekennzeichnet. Bei allen Phasendiagrammtypen mit Ausnahme von Typ I (Abb. a) gibt es zudem eine (ebenfalls durchgezogene) Drei–Phasen–Koexistenzlinie (f1–f2–g1). Diese wird immer durch einen oberen (U) und manchmal (Typ IV, V, VI; Abb. d–f) auch durch einen unteren kritischen Endpunkt (L) begrenzt. Linien von kritischen Punkten sind gestrichelt. Die Darstellung entstammt [65].

Experimente [74], [75] und neuere theoretische Arbeiten [73] zeigen, dass Tridekan–CO_2 Typ–IV–Phasenverhalten (Abb. 3.6 d) aufweist. Hierbei handelt es sich jedoch lediglich um einen Übergangstyp. Ab n–Tetradekan [76], [73] erhält wieder man Typ III. Der exakte Übergang von Typ II nach Typ III über Typ IV wird u.A. in [77] erläutert. Grob gesprochen bildet sich ein trikritischer Punkt auf der kritischen Linie, der in den LCEP und den UCEP separariert und somit die auftretende Drei–Phasen–Koexistenzlinie begrenzt (Typ II $\rightarrow$ Typ IV). Der LCEP verschiebt sich zu niedrigeren Temperaturen, während sich der Ausgangspunkt der kritischen Flüssig–flüssig–Linie zu höheren Temperaturen verschiebt. Treffen beide zusammen, verschmelzen die kritischen Linien und es entsteht ein Übergang von Typ IV zu Typ III.

Der folgende Abschnitt demonstriert, wie man im Rahmen des vergröberten Modells durch die Variation eines einzelnen Parameters ξ fast quantitativ das richtige Phasenverhalten des Referenzsystems nachbilden kann. Durch eine sukzessive Absenkung von ξ (und damit der Wechselwirkungen zwischen Hexadekan– und CO_2–Kugeln) wird die Mischbarkeit der beiden Komponenten verschlechtert und das System von Typ

I in Typ III übergeführt. Diese Anpassung ist notwendig, da die Asymmetrie zwischen einem ungeladenen Alkan und einem kleinen Molekül mit Partialladungen (wie CO_2) größer ist als bei einem Alkan und dem hier verwendeten ungeladenen Lösungsmittelteilchen. Zur Überprüfung des Phasendiagrammtyps untersucht man die Projektionen auf die pT–Ebene (Abbildung 3.7), die „Zusammensetzung“ der kritischen Linie (Abbildung 3.8) und Wahrscheinlichkeitsprofile an ausgewählten kritischen Punkten (Abbildung 3.9 und 3.10). Da Simulationsboxen notwendigerweise eine endliche Größe aufweisen, ist, wie in Kapitel 1.4 dargelegt, die Wahrscheinlichkeitsverteilung des Ordnungsparameters (Dichte) am kritischen Punkt bimodal. Dies gilt auch, wenn man sich an Stelle einer Projektion der Verteilung auf die Polymer– oder CO_2–Ebene das gesamte Wahrscheinlichkeitsprofil in einem dreidimensionalen Konturplot betrachtet (vgl. mit Kap. 1.3). Eine solche Auftragung liefert Hinweise auf die Zusammensetzung der beiden koexistierenden Phasen knapp unterhalb des kritischen Punktes und somit auf die „Zusammensetzung“ der kritischen Linie.

Für $\xi = 1$ ist die kritische Linie ($\times$) in Abbildung 3.7 geschlossen. Abbildung 3.8 zeigt einen kontinuierlichen Übergang der kritischen Zusammensetzung von reinem Polymer ($x = 0$) zu reinem CO_2 ($x = 1$). Man erwartet folglich ein Phasendiagramm des Typs I oder II. Diese Vermutung wird weiter untermauert, wenn man sich die Wahrscheinlichkeitsverteilungen entlang der kritischen Linie betrachtet (Abbildung 3.9). Knapp oberhalb des kritischen Punktes von reinem CO_2 (Abb. 3.9 a) erhält man eine Koexistenz zwischen einer Gas– und einer Flüssigkeitsphase von CO_2, die jeweils nur einen geringen Anteil von Hexadekan aufweisen. Mit steigender Temperatur verändert sich die „kritische Zusammensetzung“ sukzessive bis man schließlich in der Nähe des kritischen Punktes von reinem Hexadekan (Abb. 3.9 f) eine Gas–flüssig–Koexistenz zweier Polymerphasen erhält. Die Simulationen mit $\xi = 1$ lassen sich folglich einem Phasendiagramm des Typs I oder II zuordnen.

Für $\xi = 0.9$ zeigt Abbildung 3.8 einen nicht unerheblichen Restanteil von Hexadekan nahe des kritischen Punktes von reinem CO_2. Dies ist bereits ein Merkmal von Typ III. Für $\xi = 0.9$ gelang es zusätzlich, eine Drei–Phasen–Koexistenz bei $T_c(CO_2)$ zu simulieren (Abbildung 3.11). In der pT–Projektion entspricht dies einem Punkt auf der Drei–Phasen–Linie (vgl. mit Abb. 3.6). Somit kann schon für $\xi = 0.9$ Typ I und II ausgeschlossen werden, da dort der Endpunkt der Drei–Phasen–Koexistenzlinie unterhalb der kritischen Temperatur der ersten Komponente liegt und deswegen am kritischen Punkt keine drei Phasen koexistieren können.

Für $\xi = 0.886$ verstärken sich diese Tendenzen weiter. Die kritische Linie in Abbildung 3.7 schließt nun eindeutig nicht mit dem kritischen Punkt von CO_2 ab und auch der Molbruch bei $T_c(CO_2)$ unterscheidet sich deutlich von Eins (Abb. 3.8). Die Zusammensetzung für $\xi = 0.886$ knapp oberhalb des kritischen Punktes von reinem CO_2 (Abb. 3.10 a) zeigt deutlich eine Koexistenz einer flüssigen CO_2– mit einer flüssigen, hexadekanreichen Phase an. Die Zusammensetzung nahe des kritischen Punk-

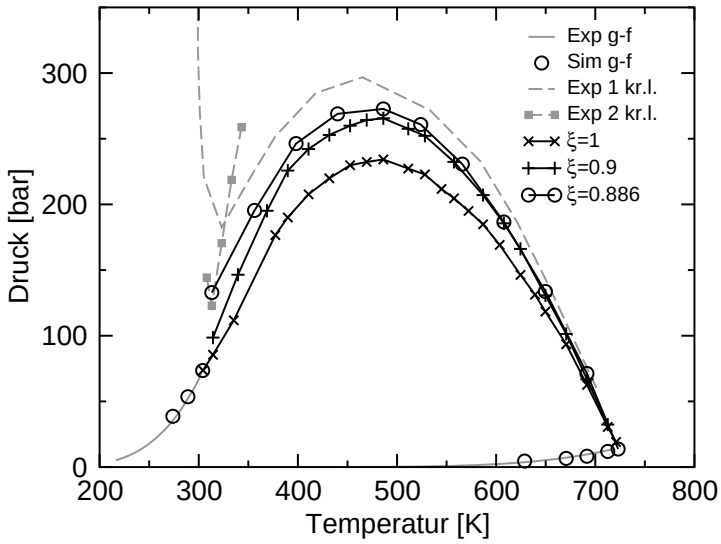

Abbildung 3.7: Mischsystem–Phasendiagramm in der pT–Projektion für $\xi = 1$, $\xi = 0.9$ und $\xi = 0.886$. Simulationsdaten: entlang der Gas–flüssig–Koexistenzlinie von reinem CO_2 und $C_{16}H_{34}$ ($\circ$), entlang der kritischen Linie des Mischsystems ($\times$: $\xi = 1.0$, $+$: $\xi = 0.9$, $\circ$: $\xi = 0.886$). Die zur Erstellung dieses Graphen benötigten Simulationsdaten finden sich in den Tabellen A.3, A.4 und A.5 im Anhang A.

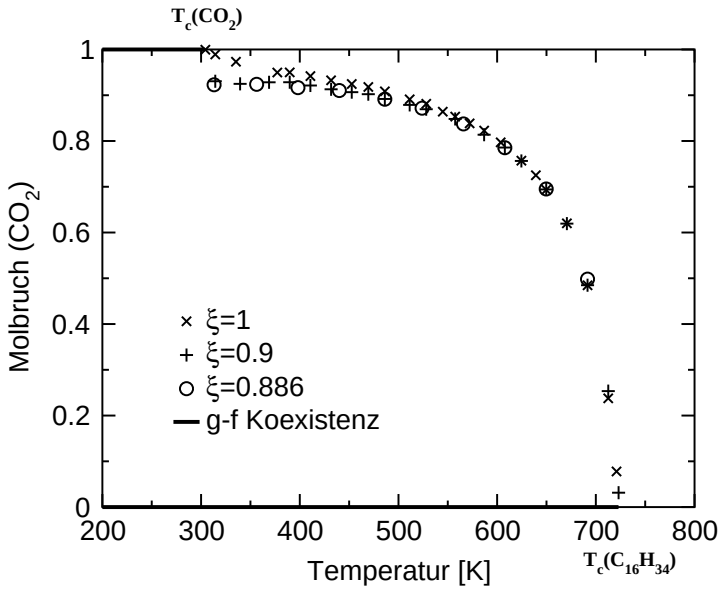

Abbildung 3.8: Mischsystem–Phasendiagramm in der Tx–Projektion für $\xi = 1$, $\xi = 0.9$ und $\xi = 0.886$. Simulationsdaten: entlang der kritischen Linie des Mischsystems ($\times$: $\xi = 1.0$, $+$: $\xi = 0.9$, $\circ$: $\xi = 0.886$).

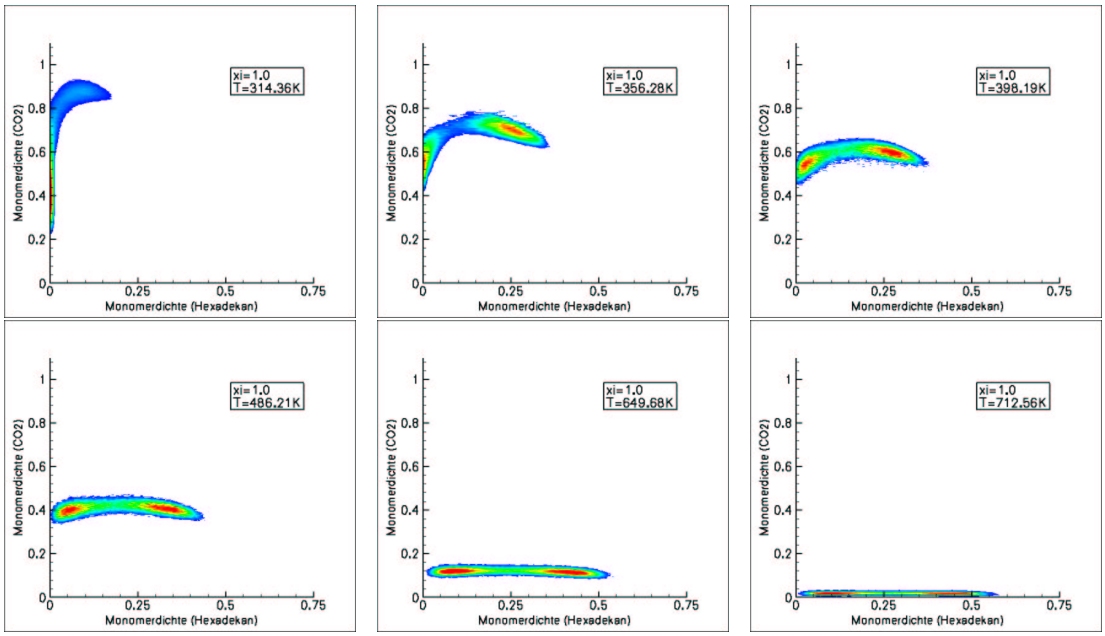

Abbildung 3.9: Wahrscheinlichkeitsprofile entlang der kritischen Linie ($\xi = 1$). Boxgröße: (a), (b) $L = 6.74\ \sigma_5$, Rest: $L = 9\ \sigma_5$.

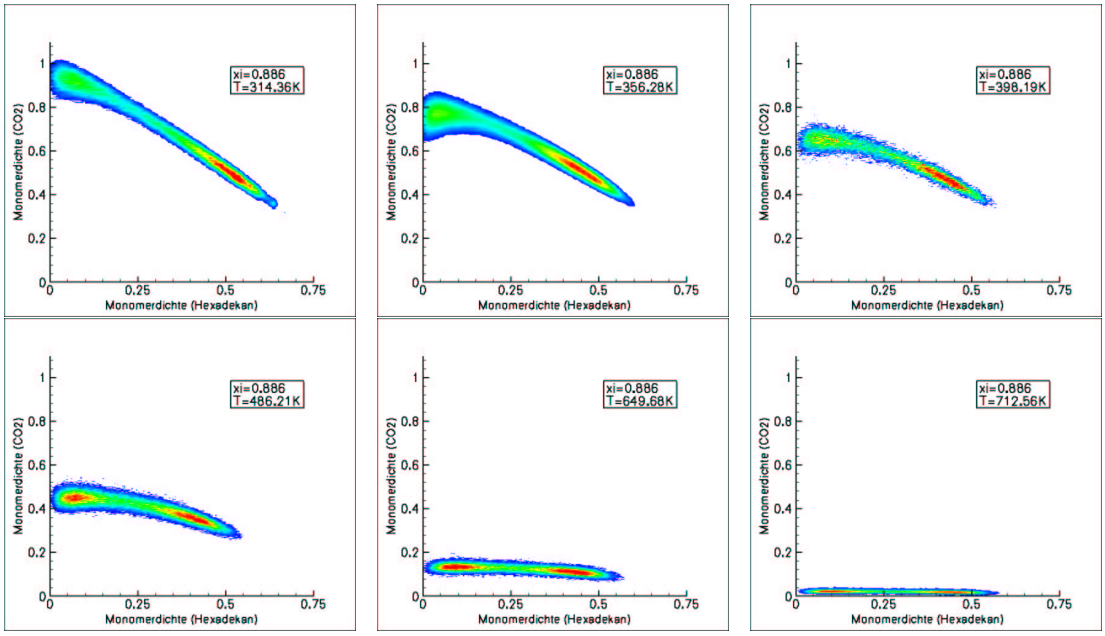

Abbildung 3.10: Wahrscheinlichkeitsprofile entlang der kritischen Linie ($\xi = 0.886$). Boxgröße: (a), (b) $L = 6.74\ \sigma_5$, Rest: $L = 9\ \sigma_5$. Für $\xi = 0.9$ erhält man qualitativ vergleichbare „kritische Profile“.

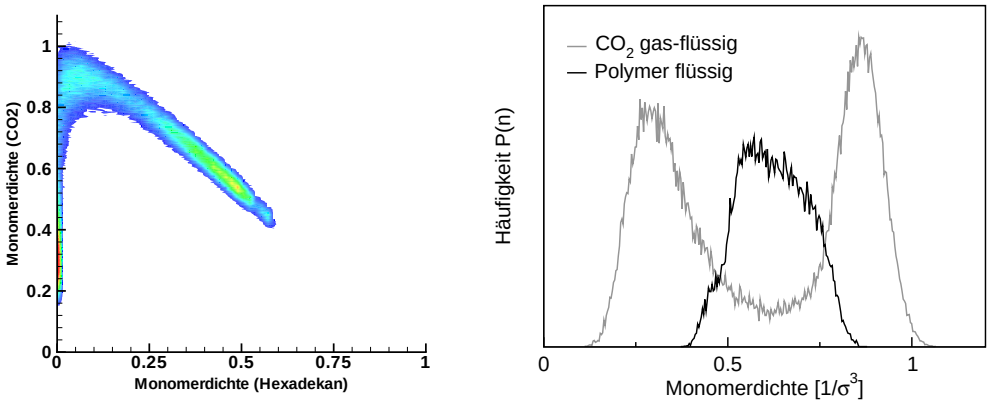

Abbildung 3.11: Wahrscheinlichkeitsverteilungen bei Drei–Phasen–Koexistenz $T_c(CO_2)$ ($L = 6.74\ \sigma_5$, $\xi = 0.9$, $T = 304\ K$, $\mu_5 = 43.9075$, $\mu_1 = -2.06643$): (a) wahre Verteilung, (b) Projektion auf Monomerachse.

tes von reinem Hexadekan (Abb. 3.10 f) entspricht der für $\xi = 1$. Entlang der kritischen Linie ändert sich die „kritische Zusammensetzung“ also von Gas(Polymer)–Flüssigkeit(Polymer) zu Flüssigkeit(CO_2)–Flüssigkeit(Polymer). Dieses Verhalten entspricht einem Phasendiagramm des Typs III. Obwohl für $\xi = 0.886$ Typ IV durch die Simulation nicht gänzlich ausgeschlossen werden kann, unterstützen auch Störungsrechnungen in Kapitel 4 die Zuweisung zu Typ III. Ein quantitativer Vergleich mit Experimenten gestaltet sich schwierig, da verlässliche Daten für das Mischsystem fehlen. In Abbildung 3.7 wurden zwei experimentell bestimmte kritische Linien aufgetragen [72], [78]. Auch neuere Messungen von *B.Rathke* [79], welche im Rahmen des BASF–Projektes durchgeführt wurden, weichen etwas von beiden Linien ab ($p_c = 165\ bar$ bei $T = 313.15\ K$). Da sich die experimentelle Situation derart uneinheitlich darstellt und für $\xi = 0.886$ bereits alle qualitativen Eigenschaften des Phasendiagramms erfasst wurden, soll auf eine weitere Anpassung des Parameters verzichtet werden. Würde man ξ noch etwas weiter absenken, könnte die experimentelle Kurve von *Schneider et al.* [72] (Exp.1 in Abb. 3.7) auch quantitativ reproduziert werden.

An Stelle von Projektionen können auch Schnitte durch das Phasendiagramm bei konstanter Temperatur durchgeführt werden (Abbildung 3.12, vgl. mit den geschlossenen Linien im (p,T,x)–Diagramm in Abbildung 3.5 a). Hierbei handelt es sich um die bevorzugte Darstellung von koexistierenden Mischsystemphasen in vielen experimentellen Arbeiten [78], [79] und Computersimulationen [9]. Die Darstellung ähnelt in ihrer Struktur Abbildung 3.1. Die Binodale grenzt das Ein–Phasen–Gebiet von der Mischungslücke ab. Bei einem Sprung in den metastabilen Bereich erwartet man Phasenseparation durch Nukleation, im instabilen Bereich durch spinodale Entmischung. Binodale und Spinodale enden im kritischen Punkt. Bei Phasendiagrammen vom Typ

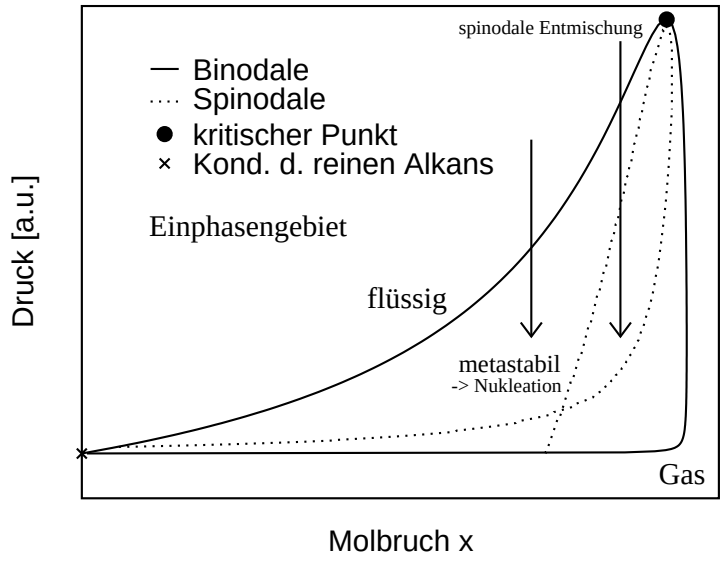

Abbildung 3.12: Isotherme im Zwei–Komponenten–Mischsystem (p,x)

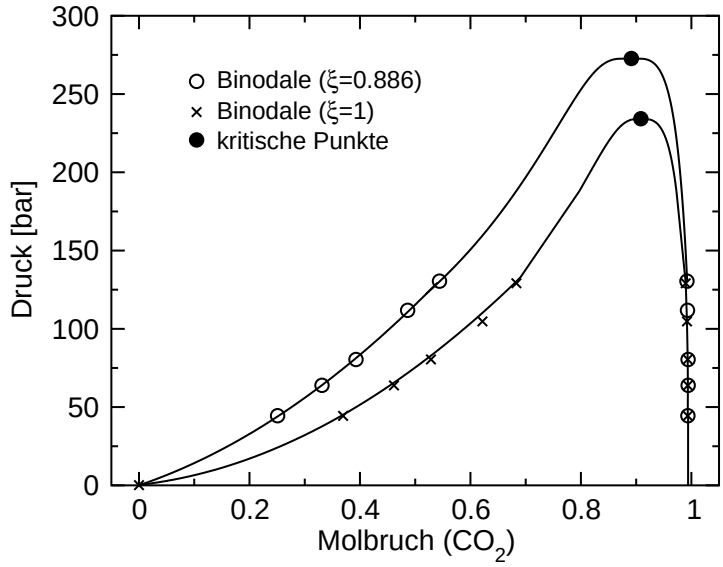

Abbildung 3.13: Isotherme für $\xi = 0.886$ und $\xi = 1$ bei $T = 486.2\ K$. Kreise und Kreuze sind Simulationsergebnisse, die Kurve ein „guide to the eye“. Die Kurvenverlauf in der Nähe des kritischen Punktes wurde durch „finite size scaling“ bestimmt. Die zur Erstellung dieses Graphen benötigten Simulationsdaten finden sich in den Tabellen A.6 und A.7 im Anhang A.

III können in der Nähe des kritischen Punktes der leichter flüchtigen Komponente ein zweiter kritischer Punkt sowie eine Drei–Phasen–Koexistenzlinie erscheinen. Da die tiefen Temperaturen die Erstellung eines solchen Diagramms verhindern, sei auf eine analytische Betrachtung eines ähnlichen Modells verwiesen [80].

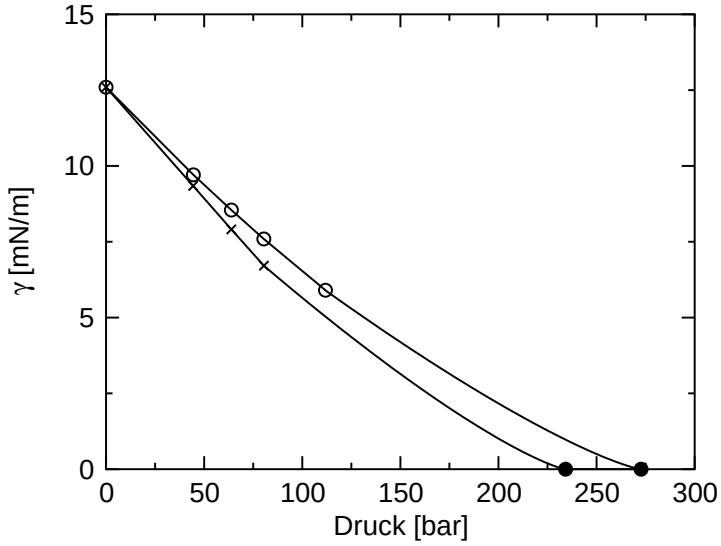

Abbildung 3.14: Grenzflächenspannung der Isotherme als Funktion des Drucks. Die Kreuze ($\xi = 1$) und Kreise ($\xi = 0.886$) sind Simulationsergebnisse an die (im Bereich von 80 bzw. 110 $bar - p_c$) eine Kurve der Form $\gamma = \gamma_0(1 - \frac{p}{p_c})^{2\nu}$ mit dem Ising–Exponenten $\nu = 0.63$ „gefittet“ wurde.

Abbildung 3.13 zeigt die Binodale zweier Isothermen bei einer Temperatur von 486.2 K – einmal für $\xi = 1$ und einmal für $\xi = 0.886$ (vgl. auch mit Abb. 3.10 d). Erwartungsgemäß (vgl. mit Abbildung 3.7) liegt der kritische Punkt für Letztere bei einem höheren Druck. Ein Vergleich mit experimentellen Daten ist nicht möglich, da für diese Temperatur keine Messwerte existieren. Tendenziell wird der kritische Druck, wie oben erläutert, jedoch etwas höher liegen. Abbildung 3.14 zeigt die Grenzflächenspannung entlang der Isotherme. Für $x \to 0$ strebt γ dabei gegen die Grenzflächenspannung von reinem Hexadekan. Dieser Wert würde gemäß Abbildung 3.4 voraussichtlich sehr gut mit dem Experiment übereinstimmen. Abweichungen erwartet man abermals in der Größe des kritischen Drucks ($\gamma = 0$). Abbildung 3.15 zeigt die Binodale und die Grenzflächenspannung für $T = 649.7\ K$ und $\xi = 0.9$. Im Unterschied zur niedrigeren Temperatur ist der Anteil von Hexadekan in beiden Phasen deutlich angestiegen (vgl. Absolutwerte der Molbrüche). Die maximale Grenzflächenspannung hat sich hingegen erwartungsgemäß erniedrigt.

Insgesamt gesehen sind Ergebnisse der Mischsystem–Simulationen äußerst zufriedenstellend. Es konnte zum erstem Mal ein (fast) vollständiges Phasendiagramm in-

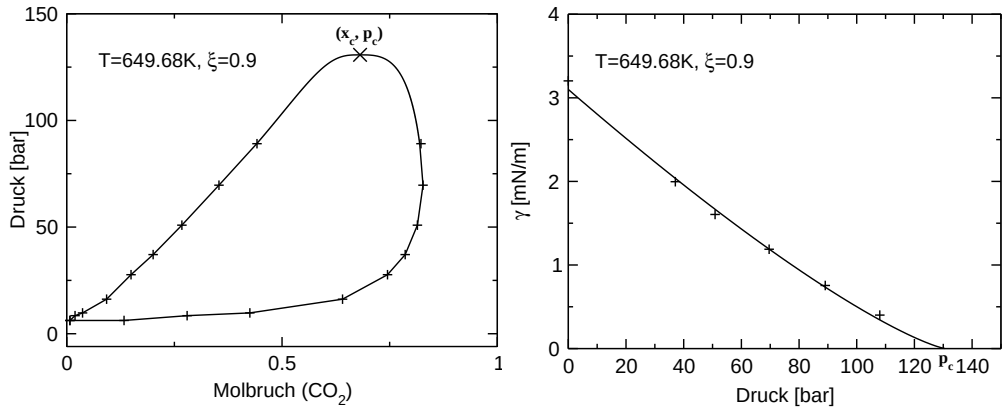

Abbildung 3.15: Wie Abbildungen 3.13 und 3.14, aber für $T = 649.68\ K$ und $\xi = 0.9$. Für $p > 100\ bar$ kommen bei der Grenzflächenspannung Abweichungen durch Finite–Size–Effekte hinzu. Die zur Erstellung des Graphen benötigten Simulationsdaten finden sich in Tabelle A.8 im Anhang A.

klusive kritischer Linien simuliert werden. Zudem gelang es trotz eines überaus einfachen Modells und nur einer frei einstellbaren Größe (ξ), den Typ des Referenzsystems und experimentelle Vorgaben fast quantitativ nachzuvollziehen. Im folgenden Kapitel soll die Diskussion der Phasendiagramme im Vergleich mit einer analytischen Theorie (TPT1) nochmals vertieft werden.

Kapitel 4

Vergleich der Simulationsergebnisse mit TPT1–Störungsrechnung

In diesem Kapitel werden die umfangreichen Simulationsdaten für Mischsysteme mit numerischen Störungsrechnungen von *L.G. MacDowell* [50] verglichen. Die verschiedenen Variationen der auf molekularer Basis entwickelten Zustandsgleichung fasst man gewöhnlich unter dem Begriff TPT1 („**t**hermodynamic **p**ertubation **t**heory“) oder SAFT („**s**tatistical **a**ssociating **f**luid **t**heory“) zusammen. Der ursprünglich von *Wertheim* vorgeschlagene Ansatz [81] wurde im Laufe der Zeit vielfach verbessert und u.A. auch auf Mischungen von Ketten [82] übertragen. Während die Theorie bereits ausführlich für einkomponentige Systeme getestet wurde [83], [84], sind Überprüfungen für Alkan–Lösungsmittel–Systeme eher selten [84], [85]. Dies liegt vor allem an einem Mangel an passenden Simulationsdaten, die durch diese Arbeit nun vorliegen [50] .

Numerische Rechnungen benötigen weitaus weniger Computerzeit als entsprechende Simulationen und können deswegen das Phasenverhalten eines Systems über einen größeren Bereich ausloten. In diesem Zusammenhang werden die in Kapitel 3.3 bestimmten Phasendiagramme nochmals genauer klassifiziert und der Einfluss der Kettenlängen der Alkane auf das Phasenverhalten geklärt. Letzteres lässt sich mit Simulationen nur unter enormen Aufwand realisieren. Der direkte Vergleich mit einer „mean field“–artigen Theorie erlaubt des Weiteren die Überprüfung einer Methode zur Bestimmung von Spinodalen bei Simulationen. Zudem lässt sich durch einen Vergleich die Güte der in der Theorie verwendeten Näherungen stufenweise überprüfen.

4.1 Grundlagen

Da sämtliche Berechnungen mit der TPT1–Störungstheorie von *L.G. MacDowell* durchgeführt wurden, soll an dieser Stelle lediglich ein kurzer Einblick in die Theorie gewährt werden. Die folgenden Darstellungen entsprechen mit kleinen Abweichungen einer Ausarbeitung von Abschnitt III A und D in Referenz [50].

Im Allgemeinen wird die Zustandsgleichung von TPT1 über die Helmholtz'sche freie Energie F formuliert. Diese setzt sich aus der Summe über die einzelnen mikroskopischen Beiträge zusammen:

$$\frac{F}{NkT} = \frac{F^{\mathrm{IDEAL}}}{NkT} + \frac{F^{\mathrm{MONO}}}{NkT} + \frac{F^{\mathrm{KETTE}}}{NkT}. \tag{4.1}$$

F^{IDEAL} steht für den Beitrag des idealen Gases, F^{MONO} für die Wechselwirkungen der Monomere, F^{KETTE} für die Verknüpfungsterme und N für die Gesamtzahl der Teilchen im System. Die hier eingesetzte Variante von TPT1 soll das Referenzsystem der Simulation möglichst detailliert nachstellen. Deswegen betrachtet man eine Mischung aus Fünfmeren und Lösungsmittelmonomeren. Die Teilchen wechselwirken über ein Lennard–Jones–Potential, benachbarte Teilchen einer Kette zusätzlich über ein FENE–Potential. ε_1, ε_5, ξ, σ_1, σ_5, σ_{1-5} nehmen die gleichen Werte wie in der Simulationen an (vgl. Kapitel 1.1 und 3.1).

Im Folgenden soll Gleichung (4.1) begründet werden. Hierzu betrachtet man die klassische, kanonische Zustandssumme eines Polymeres:

$$Z = \frac{1}{h^3 n!} \int\int e^{-\beta(U(\mathbf{r}) + \Sigma_{i=1}^{nl} \frac{p_i^2}{2m})}\, \mathrm{dp}^{3\mathrm{nl}} \mathrm{dr}^{3\mathrm{nl}}. \tag{4.2}$$

n steht in diesem Ausdruck für die Anzahl der Polymere, l für die Kettenlänge, U für die potentielle Energie, p für die Impulse der Monomere und h für die Planck'sche Konstante. $\beta = \frac{1}{kT}$. Da die Potentiale die Bewegung der einzelnen Monomere nicht wesentlich beschränken (keine starren Bindungen), liefert eine Integration über die Impulse:

$$Z = \frac{1}{\Lambda^{3nl} n!} \int e^{-\beta U(\mathbf{r})} \mathrm{dr}^{3\mathrm{nl}} \quad \text{mit} \quad \Lambda = \frac{\mathrm{h}}{\sqrt{2\pi \mathrm{mkT}}}. \tag{4.3}$$

Gleichung (4.3) kann wie folgt erweitert werden:

$$Z = \left(\frac{V^n}{\Lambda^{3n} n!}\right) \left(\frac{1}{V^{nl}} \int e^{-\beta E^{\mathrm{MONO}}} \mathrm{dr}^{3\mathrm{nl}}\right) \left(\left(\frac{V}{\Lambda^3}\right)^{n(l-1)} \frac{\int e^{-\beta U} \mathrm{dr}^{3\mathrm{nl}}}{\int e^{-\beta E^{\mathrm{MONO}}} \mathrm{dr}^{3\mathrm{nl}}}\right). \tag{4.4}$$

Die erste Klammer entspricht (bis auf einen konstanten Vorfaktor) der Zustandssumme eines idealen Polymergases, der zweite Faktor der Zustandssumme des (unverbundenen) Monomersystems und der dritte dem Beitrag der Verknüpfungsterme. Die Gleichung gilt im Prinzip auch für das Mischsystem, da in dem hier verwendeten Modell lediglich effektive Teilchen berücksichtigt werden. Mit $F = -kT \ln Z$ folgt (4.1).

Als Nächstes werden die einzelnen Terme in (4.1) genauer aufgeschlüsselt. Betrachtet man zunächst den Beitrag des idealen Gases, so ist in der Stirling'schen Approximation:

$$\frac{F^{\mathrm{IDEAL}}}{kT} = -\ln Z^{\mathrm{IDEAL}} = -\ln \frac{V^n}{\Lambda^{3n} n!} \approx n \ln n \frac{\Lambda^3}{V} - n = n(\ln \rho \Lambda^3 - 1). \tag{4.5}$$

Für ein Mischsystem erhält man entsprechend:

$$\frac{F^{\mathrm{IDEAL}}}{NkT} = \left(\sum_{i=1}^{2} x_i \ln \rho x_i \Lambda_i^3 \right) - 1. \tag{4.6}$$

$x_i = \frac{n_i}{N}$ steht hierbei für den Molbruch von Komponente i.

Die Lennard–Jones–Wechselwirkungen der Monomere lassen sich mit Hilfe einer Störungsrechnung ermitteln. Für ein einkomponentiges System ist:

$$\frac{F^{\mathrm{MONO}}}{nkT} = l \frac{F^m}{nlkT} = l \left(\frac{F_0}{nlkT} + \frac{F_1}{nlkT} + \frac{F_2}{nlkT} \right). \tag{4.7}$$

Als Referenzpotential F_0 dient die freie Energie einer Flüssigkeit von harten Kugeln, F_1 und F_2 sind Störterme erster und zweiter Ordnung. Für die Mischung erhält man:

$$\frac{F^{\mathrm{MONO}}}{NkT} = \left(\sum_{i=1}^{2} x_i l_i \right) \frac{F^l}{(n_1 l_1 + n_2 l_2) kT}. \tag{4.8}$$

Das Lösungsmittel wird also nicht explizit durch eine zweite Teilchensorte dargestellt. Vielmehr bildet man, wie bereits angedeutet, effektive Teilchen durch eine Skalierung der Größe und Potentialtiefe.

Im verbleibenden Ausdruck von Gleichung (4.4) (dritte Klammer) sind über die potentielle Energie noch Anteile von Monomer– und Ketten–Wechselwirkungen miteinander verknüpft. Der Quotient der Integrale kann als Mittelwert von $e^{-\beta E^{\mathrm{KETTE}}}$ bezüglich des Monomersystems verstanden werden:

$$\langle e^{-\beta E^{\mathrm{KETTE}}} \rangle_{\mathrm{MONO}} = \langle \prod_{1}^{n(l-1)} e^{-\beta E^{\mathrm{FENE}}} \rangle_{\mathrm{MONO}} \approx (\langle e^{-\beta E^{\mathrm{FENE}}(\mathbf{r_i} - \mathbf{r_j})} \rangle_{\mathrm{MONO}})^{n(l-1)}. \tag{4.9}$$

Vernachlässigt man Korrelationen zwischen den Bindungen, lassen sich beide Wechselwirkungsbeiträge entkoppeln. In dieser Näherung fließt nur noch die Struktur der unverbundenen Monomerflüssigkeit ein, was das Problem erheblich vereinfacht. Mit

$$\langle e^{-\beta E^{\mathrm{FENE}}} \rangle_{\mathrm{MONO}} = \left\langle \int \delta(\mathbf{r} - (\mathbf{r_i} - \mathbf{r_j}))\, e^{-\beta E^{\mathrm{FENE}}} \mathrm{d}r^3 \right\rangle = \frac{1}{V} \int g(r) e^{-\beta E^{\mathrm{FENE}}} \mathrm{d}r^3 \tag{4.10}$$

erhält man insgesamt:

$$\frac{F^{\mathrm{KETTE}}}{kT} = -n(l-1)\ln\frac{\delta}{\Lambda^3}, \qquad \delta = \int g(r)e^{-\beta E^{\mathrm{FENE}}}\mathrm{dr}^3, \tag{4.11}$$

wobei $g(r)$ für die Paar–Korrelationsfunktion des (unverbundenen) Monomersystems steht. Für ein Mischsystem ist entsprechend:

$$\frac{F^{\mathrm{KETTE}}}{NkT} = -\sum_{i=1}^{2} x_i(l_i-1)\ln\frac{\delta_i}{\Lambda_i^3}. \tag{4.12}$$

Durch die Kenntnis der freien Energie ist das System vollständig beschrieben und Ausdrücke für Druck und chemisches Potential lassen sich aus thermodynamischen Überlegungen ableiten. Hierbei ist zu beachten, dass die natürlichen Variablen Temperatur, Molbruch und Dichte sind. So ist:

$$p = -\left(\frac{\partial F}{\partial V}\right)_{\mathrm{T,n_1,n_2}} = kT\rho^2\frac{\partial}{\partial\rho}\frac{F}{NkT}. \tag{4.13}$$

Analog erhält man für das chemische Potential:

$$\mu_i = \left(\frac{\partial}{\partial n_i}F\right)_{\mathrm{T,V,n_j}} = \frac{F}{N} + \frac{p}{\rho} + kT(1-x_i)\frac{\partial}{\partial x_i}\frac{F}{NkT}. \tag{4.14}$$

Bei Phasenkoexistenz sind Druck und chemische Potentiale in beiden Phasen gleich, d.h.:

$$\begin{aligned} p(T,\rho^I,x^I) &= p(T,\rho^{II},x^{II}) \\ \mu_1(T,\rho^I,x^I) &= \mu_1(T,\rho^{II},x^{II}) \\ \mu_2(T,\rho^I,x^I) &= \mu_2(T,\rho^{II},x^{II}). \end{aligned} \tag{4.15}$$

Mit (4.13) und (4.14) können nun die Koexistenzkurven durch das iterative Lösen des Gleichungssystems (4.15) bestimmt werden.

Ein System befindet sich im Gleichgewicht, wenn es in einer freien Energiemulde liegt, d.h. $\mathrm{dF} = 0$ und $\mathrm{d^2F} > 0$. Kleine Auslenkungen bewirken dann ein Moment, welches das System wieder ins Gleichgewicht treibt. Hieraus folgen direkt die Kriterien für mechanisches und chemisches Gleichgewicht (4.15). Im metastabilen und im instabilen Bereich ist $\mathrm{dF} \neq 0$. Das resultierende Moment ist stets so gerichtet, dass das System dem Gleichgewichtszustand entgegenstrebt, also eine Tendenz zur Phasenseparation aufzeigt. Zu Unterscheidungszwecken zieht man daher die Krümmung der freien Energie heran. Im metastabilen Bereich liefert das zweite Moment immer noch einen rücktreibenden Beitrag, d.h. $\mathrm{d^2F} > 0$. Unter Vernachlässigung des ersten Moments wäre das System also stabil. Für $\mathrm{d^2F} = 0$ erhält man die Grenze der Stabilität, die so genannte Spinodale. Die folgenden Betrachtungen beziehen sich auf die natürlichen Einheiten der Theorie $f(T,\rho,x) = \frac{F}{NkT}$:

$$\mathrm{d^2f} = \frac{1}{2}\frac{\partial^2\mathrm{f}}{\partial\rho^2}\mathrm{d}\rho^2 + \frac{1}{2}\frac{\partial^2\mathrm{f}}{\partial\mathrm{x}^2}\mathrm{dx}^2 + \frac{1}{2}\frac{\partial^2\mathrm{f}}{\partial\rho\partial\mathrm{x}}\mathrm{d}\rho\mathrm{dx} + \frac{1}{2}\frac{\partial^2\mathrm{f}}{\partial\mathrm{x}\partial\rho}\mathrm{dx}\mathrm{d}\rho. \tag{4.16}$$

(4.16) lässt sich auch in Matrixform darstellen:

$$\left(\mathrm{d}\rho, \mathrm{dx} \right) \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathrm{d}\rho \\ \mathrm{dx} \end{pmatrix} = A_{11}\mathrm{d}\rho^2 + \mathrm{A}_{21}\mathrm{dx}\mathrm{d}\rho + \mathrm{A}_{12}\mathrm{d}\rho\mathrm{dx} + \mathrm{A}_{22}\mathrm{dx}^2. \quad (4.17)$$

Ein Koeffizientenvergleich mit (4.16) liefert:

$$A = \begin{pmatrix} \frac{\partial^2 f}{\partial \rho^2} & \frac{\partial^2 f}{\partial x \partial \rho} \\ \frac{\partial^2 f}{\partial \rho \partial x} & \frac{\partial^2 f}{\partial x^2} \end{pmatrix} \quad (4.18)$$

In Gleichung (4.17) lassen sich die Terme auch entkoppeln, d.h. es existiert eine Transformation auf eine Eigenwertbasis, so dass gilt:

$$\mathrm{d}^2\mathrm{f} = \frac{1}{2}\frac{\partial^2 \mathrm{f}}{\partial \eta_1^2}\mathrm{d}\eta_1^2 + \frac{1}{2}\frac{\partial^2 \mathrm{f}}{\partial \eta_2^2}\mathrm{d}\eta_2^2. \quad (4.19)$$

Das System überschreitet nun die Grenze zur Instabilität, wenn entweder $\frac{\partial^2 f}{\partial \eta_1^2}$ oder $\frac{\partial^2 f}{\partial \eta_2^2}$ einen Vorzeichenwechsel von Plus nach Minus vollziehen, d.h. die Determinante der Eigenwertmatrix gleich Null wird. Dieses Kriterium ist etwas stärker als das bisher geforderte $\mathrm{d}^2\mathrm{F} = 0$. Im Allgemeinen genügt es also, dass die Krümmung der freien Energie entlang eines Pfades ihr Vorzeichen wechselt. Da die Determinante einer Matrix unabhängig von der Wahl ihrer Basis ist, erhält man an der Spinodalen:

$$\rho^2 \det(\mathrm{A}) = \rho^2 \left(\frac{\partial^2 \mathrm{f}}{\partial \rho^2}\frac{\partial^2 \mathrm{f}}{\partial \mathrm{x}^2} - \frac{\partial^2 \mathrm{f}}{\partial \mathrm{x}\partial \rho}\frac{\partial^2 \mathrm{f}}{\partial \rho \partial \mathrm{x}} \right) = 0. \quad (4.20)$$

ρ^2 dient lediglich Normierungszwecken und macht den Ausdruck dimensionslos. Ähnlich wie bei der Bestimmung der Koexistenz kann (4.20) wieder iterativ bestimmt werden. Kritische Punkte des Mischsystems ergeben sich jeweils als Maximum der isothermen Spinodalen.

Wie bereits erwähnt, soll diese Zusammenfassung lediglich einen Einblick in die Theorie vermitteln. Insbesondere die Ableitung der freien Energieterme in Gleichung (4.8) und die Berechnung der Korrelationsfunktionen in (4.12) mit Hilfe der so genannten „**m**ean **s**pherical **a**pproximation“ (MSA) benötigen einen sehr viel breiter angelegten Einstieg in die Flüssigkeitstheorie, der den Rahmen dieser Einführung sprengen würde. Weiterführende Darstellungen und Referenzen findet man u.A. in [50] und [83].

4.2 Lennard–Jones– und Polymersysteme

Im Folgenden wird zunächst das Verhalten des reinen Fünfmer– und des Monomer–Systems untersucht. Abbildung 4.1 zeigt die mit TPT1 erzeugten Phasendiagramme und die Monte–Carlo–Simulationsdaten. Tabelle 4.1 vergleicht die kritischen Punkte.

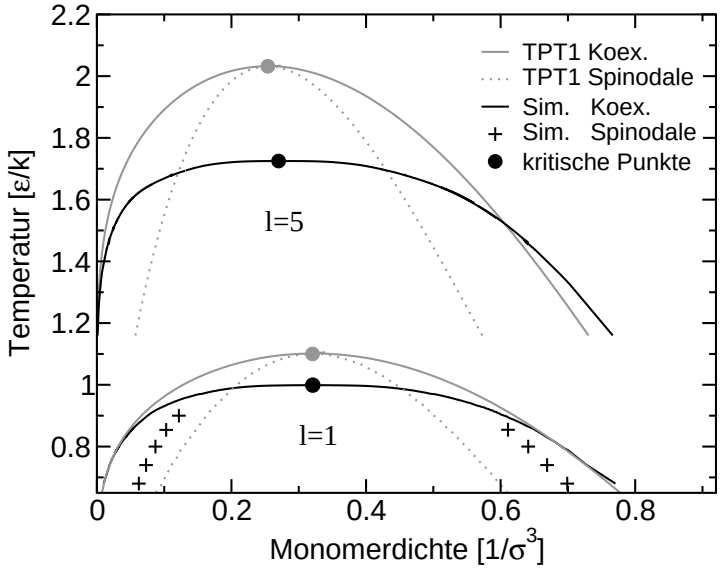

Abbildung 4.1: Koexistenzkurven und Spinodale für das Fünfmer– ($l = 5$) und das Lennard–Jones–System ($l = 1$). TPT1 überschätzt die kritischen Punkte und zeigt in deren Nähe ein „mean field"–artiges Verhalten. Zur Bestimmung der Simulationsspinodale wurde der Kehrwert der Fluktuationen linear aus dem Ein–Phasen–Gebiet in das Zwei–Phasen–Gebiet extrapoliert (siehe Abbildung 4.2).

TPT1 überschätzt die kritische Temperatur um etwa 10% bzw. 15% und die kritischen Drücke um 50% bzw. 100%. Die kritischen Dichten liegen im Vergleich hierzu besser. Die Ergebnisse sind im Einklang mit früheren Untersuchungen eines ähnlichen Systems (bei $l = 10$) [83]. Durch die Näherung in (4.9) wurden Korrelationen höherer Ordnung bei der Berechnung des Kettenbeitrags vernachlässigt. Da auch der Monomeranteil mit Hilfe einer Störungsrechnung bestimmt wurde, erhält man in der Umgebung des kritischen Punktes ein „mean field"–artiges Verhalten mit den entsprechenden kritischen Exponenten ($\beta = 0.5$ statt $\beta = 0.3258$ in der 3d–Ising Universalitätsklasse). Abweichungen von der Simulationskurve im Bereich niedriger Temperaturen sind für das Fünfmer–System größer als für das Monomer–System. Dies lässt sich nach einem Vergleich mit [83] auf die Anwendung der „mean spherical approximation" bei der Berechnung der Paar–Korrelationsfunktionen in (4.11) zurückführen.

In Ergänzung zur Koexistenzkurve wurde für das Lennard–Jones–System eine Abschätzung für die Spinodale bestimmt. Hierzu extrapoliert man bei konstanter Temperatur aus dem Ein–Phasen–Gebiet in das Zwei–Phasen–Gebiet und nimmt an, dass Fluktuationen wie bei einer Mean–Field–Theorie an der Spinodalen divergieren. Der Kehrwert der Schwankungen wurde mittels einer linearen Funktion extrapoliert, d.h.

	Monomer			Fünfmer		
	T_c	ρ_c	p_c	T_c	ρ_c	p_c
MC	0.999	0.32	0.088	1.725	0.27	0.022
TPT1	1.1074	0.3374	0.1392	2.032	0.2545	0.0442

Tabelle 4.1: Kritische Temperaturen T_c $[\frac{\varepsilon}{k}]$, Monomerdichten ρ_c $[\frac{1}{\sigma^3}]$ und Drücke p_c $[\frac{\varepsilon}{\sigma^3}]$ der Reinsysteme in den jeweiligen Lennard–Jones–Einheiten. Vergleiche mit Tabelle 3.1.

man berücksichtigt in der Virialentwicklung lediglich die führenden Terme:

$$\frac{\langle N\rangle}{\langle \delta N^2\rangle} = \frac{\partial \beta p}{\partial \rho} = 1 + 2B\rho + \ldots \tag{4.21}$$

wobei B für den zweiten Virialkoeffizient steht. Diese Annahme lässt sich mit Hilfe von TPT1 überprüfen. Abbildung 4.2 a zeigt, dass auf der Gasseite der Verlauf der Theoriekurve recht gut durch einen linearen Ausdruck beschrieben werden kann. An der Flüssigkeitsseite (Abb. 4.2 b) ist die Kurve gekrümmt – die lineare Näherung reicht nicht aus. In der angenommenen Virialentwicklung müssen also Terme höherer Ordnung berücksichtigt werden.

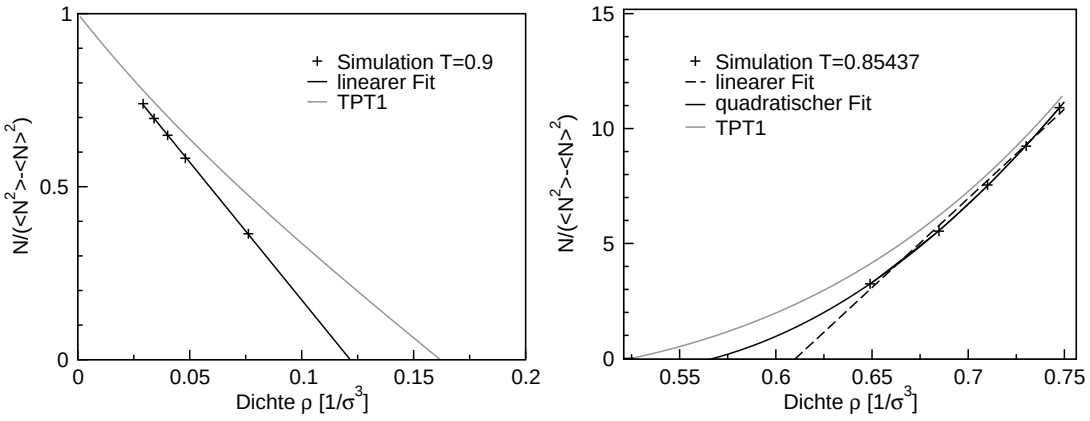

Abbildung 4.2: Extrapolation der Teilchenzahlfluktuationen (Simulation) aus dem Ein–Phasen–Gebiet in das Zwei–Phasen–Gebiet im Vergleich mit TPT1. (a) Gasseite: $T = 0.9\ \frac{\varepsilon_1}{k}$ und $L = 22.5\ \sigma_1$ (b) Flüssigkeitsseite: $T = 0.85437\ \frac{\varepsilon_1}{k}$ und $L = 6.74\ \sigma_1$.

Die mit Hilfe der Simulation bestimmte Spinodale liegt näher an der Koexistenzkurve als die theoretisch berechnete. Wie oben dargelegt, kann dies jedoch nur zum Teil durch die nicht berücksichtigte Krümmung erklärt werden. Generell sollte man die Spinodale eher als Bezugspunkt für ein Gebiet auffassen, in dem Entmischung schnell einsetzt. Bei realen Systemen ist der Übergang verschmiert und nicht scharf

wie in der Mean–Field–Theorie. So gibt es auch in dem von der Theorie vorhergesagten metastabilen Bereich Regionen, in denen Entmischung spontan einsetzt. Die Nukleationbarriere liegt dann in der Größenordnung von thermischen Fluktuationen (vgl. hierzu [80]).

4.3 Polymer–Mischungen

Numerische TPT1–Rechnungen benötigen nicht nur bedeutend weniger Computerzeit. Sie ermöglichen auch eine genauere Untersuchung von Regionen des Phasendiagramms, die der Simulation nur schwer zugänglich sind wie z.B. Bereiche hoher Dichten oder niedriger Temperaturen. Im Folgenden betrachtet man zunächst die Projektionen des dreidimensionalen Phasendiagramms auf die Druck–Temperatur–Ebene.

Abbildung 4.3 vergleicht die TPT1–Rechnungen mit Simulationen für $\xi = 1$. Die kritischen Punkte werden wie bei den reinen Systemen überschätzt – die kritische Linie verläuft oberhalb der Simulationsdaten. Der Grund hierfür liegt abermals in der Vernachlässigung von Fluktuationen in den Störungsrechnungen und dem hieraus resultierenden Mean–Field–Charakter. Skaliert man hingegen die kritischen Punkte der reinen Systeme bei TPT1 auf die der Simulation, wird die kritische Linie unterschätzt (gestrichelte Linie). Bereits hier wird deutlich, dass TPT1 lediglich eine qualitative Beschreibung des untersuchten Systems liefert, wobei die Übereinstimmung in der Nähe des kritischen Punktes des Monomer–Systems noch am besten ist. Die wesentlichen Eigenschaften des Phasendiagramms bleiben jedoch erhalten. So ist die kritische Linie wie in der Simulation geschlossen. Da auch bei sehr tiefen Temperaturen ($T = 0.4\ \frac{\varepsilon_5}{k} \approx 168\ K$ im Experiment) keine Flüssig–flüssig–Entmischung gefunden wurde, lässt sich das System eindeutig Typ I zuordnen.

Für $\xi = 0.9$ (Abbildung 4.4) ändert sich das Phasenverhalten. Bei tiefen Temperaturen ($T \approx 0.65\ \frac{\varepsilon_5}{k}$) erhält man eine Flüssig–flüssig–Entmischung und die dazugehörige kritische Linie. Eine genauere Untersuchung zeigt, dass die zweite kritische Linie in eine Drei–Phasen–Koexistenz einmündet, die über den kritischen Punkt des Monomer–Systems hinaus verläuft (in Abb. 4.4 nicht erkennbar). Dies entspricht dem Verhalten von Typ–IV–Phasendiagrammen (vgl. mit Abb. 3.6 d). In den Simulationen konnte bei hohen Temperaturen eine Änderung der kritischen Zusammensetzung im Vergleich zu $\xi = 1$ festgestellt werden (vgl. Abbildung 3.8 und Anmerkung in Abbildung 3.10). Es war jedoch unklar, ob das Phasenverhalten schon Typ III oder aber Typ IV zugeordnet werden sollte.

Durch ein erneutes Absenken des Mischbarkeitsparameters auf $\xi = 0.886$ erhält man schließlich ein Phasendiagramm vom Typ III. Die kritische Flüssig–flüssig–Linie

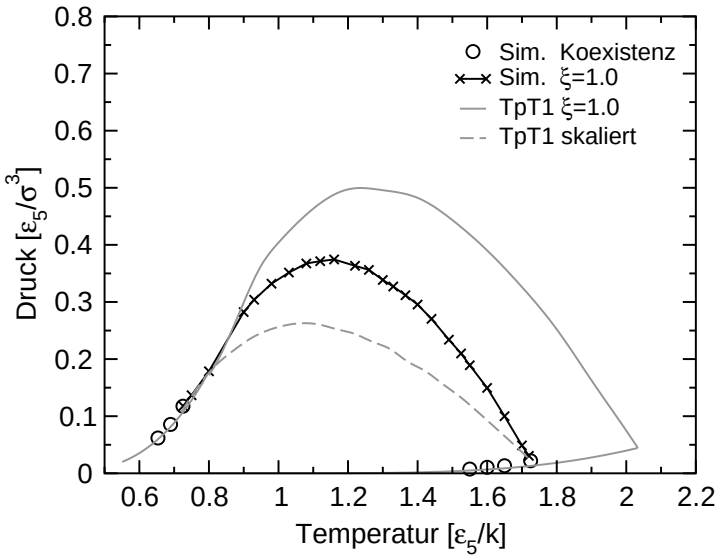

Abbildung 4.3: pT–Projektion des Fünfmer–Monomer–Mischsystem–Phasendiagramms bei $\xi = 1.0$. Bei tiefen Temperaturen tritt keine Flüssig–flüssig–Entmischung auf. Das Phasendiagramm lässt sich Typ I zuordnen. Bei der gestrichelten Linie wurden die kritischen Punkte der Reinsysteme von TPT1 auf die kritischen Punkte der Simulation abgebildet. Der Vergleich zwischen Simulation und Experiment wurde bereits in Abbildung 3.7 vorgenommen.

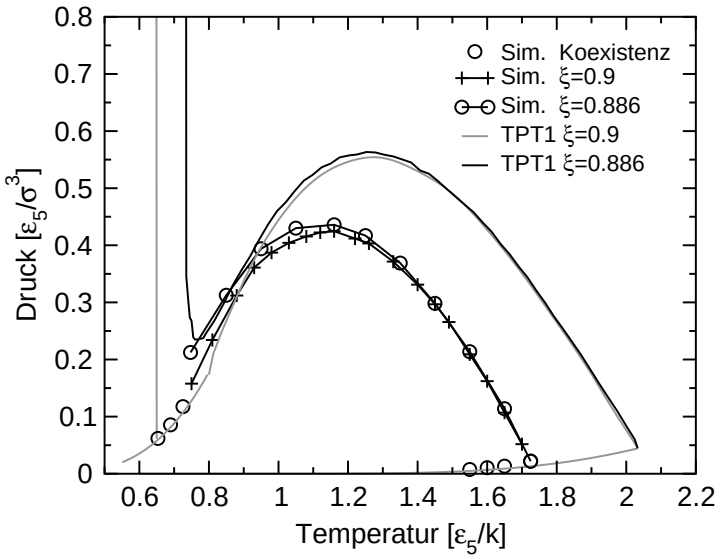

Abbildung 4.4: Mischsystem–Phasendiagramm in der pT–Projektion für $\xi = 0.9$ und $\xi = 0.886$. Bei TPT1 tritt für tiefe Temperaturen eine Flüssig–flüssig–Entmischung auf, die sich mit abnehmender Mischbarkeit zu höheren Temperaturen verschiebt. $\xi = 0.9$ lässt sich Typ IV zuordnen und $\xi = 0.886$ Typ III.

hat sich zu höheren Temperaturen verschoben und mit der zweiten kritischen Linie zu einer geschlossenen Kurve verbunden, die oberhalb der Koexistenzlinie des Monomer–Systems verläuft. Das kritische Verhalten bei hohen Temperaturen wird sowohl in der Simulation als auch in den theoretischen Berechnungen nur wenig von der Änderung in ξ beeinflusst. Der kritische Druck erhöht sich nur unwesentlich. Insgesamt können die Ergebnisse als starkes Indiz gewertet werden, dass auch bei der Simulation ein Phasendiagramm vom Typ III erzeugt wurde.

Neben einem Absenken von ξ kann auch ein Anwachsen der Kettenlänge die Asymmetrie der Mischung vergrößern und das charakteristische Phasenverhalten von Typ III verstärken. Abbildung 4.5 zeigt, wie sich die kritische Linie und die Koexistenzlinie des Polymers mit wachsender Kettenlänge (und konstantem ξ) zu höheren Temperaturen verschiebt. Eine analoge Aussage zur Kettenlängenabhängigkeit findet sich bei

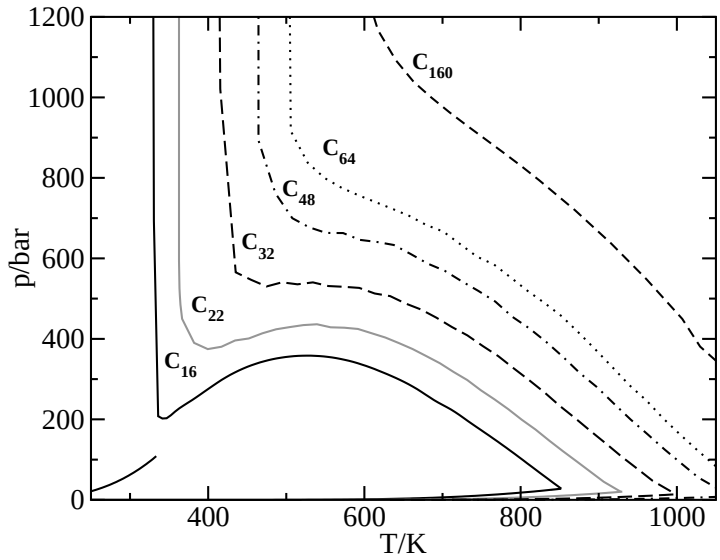

Abbildung 4.5: pT–Projektionen des Alkan–CO_2–Phasendiagramms (TPT1, $\xi = 0.882$) in experimentellen Einheiten. Mit wachsender Kettenlänge verschiebt sich die kritische Linie zu höheren Drücken und Temperaturen. Das für Typ III charakteristische Verhalten tritt verstärkt hervor.

[86] und im Ansatz auch bei [72] und [73]. Die Erstellung von Abbildung 4.5 mittels Simulation wäre sehr aufwendig, da mit steigender Kettenlänge die Akzeptanzrate der großkanonischen Schritte auch bei hohen Temperaturen schnell gegen Null strebt. Da reale Polymer–Mischsysteme wie z.B. Polystyrol in CO_2 sehr lange Ketten aufweisen, zeigen isotherme Schnitte (mit $T > T_c(CO_2)$) bei im Labor zugänglichen Bedingungen mit großer Wahrscheinlichkeit niemals einen kritischen Punkt. Eine Zugabe von Polymeren mittlerer Länge könnte die Asymmetrie aufweichen und die oben beschriebene Tendenz umkehren.

Neben der Projektion des gesamten Phasendiagramms auf die Druck–Temperatur–Ebene wurden auch einzelne isotherme Schnitte verglichen. Für $T = 1.16\ \frac{\varepsilon_5}{k}$ und $\xi = 1$

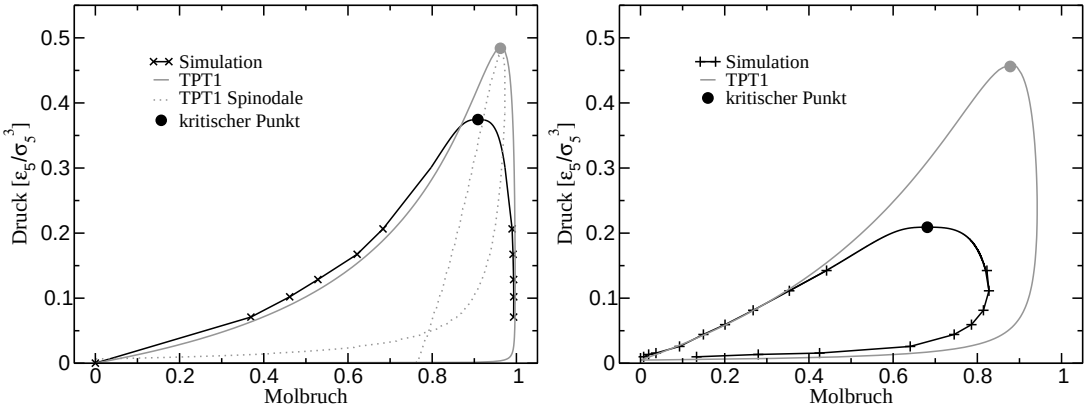

Abbildung 4.6: (a) *px*–Schnitt für eine Fünfmer–Monomer–Mischung bei $T = 1.16\ \frac{\varepsilon}{k}$ und $\xi = 1$, (b) bei $T = 1.55\ \frac{\varepsilon}{k}$ und $\xi = 0.9$. Für $T = 1.16\ \frac{\varepsilon}{k}$ wurde zusätzlich die Spinodale bestimmt. Die Symbole entsprechen den Simulationsdaten. In der Nähe des kritischen Punktes wurde der Verlauf durch eine Finite–Size–Skalenanalyse erstellt, der Rest der schwarzen durchgezogenen Linie ist lediglich ein „guide to the eye“.

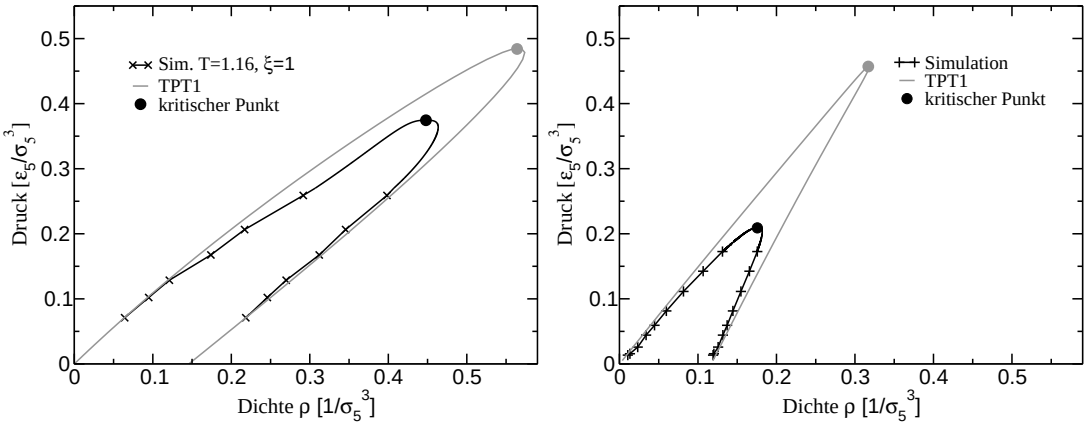

Abbildung 4.7: (a) *pρ*–Schnitt für eine Fünfmer–Monomer–Mischung bei $T = 1.55\ \frac{\varepsilon}{k}$ und $\xi = 0.9$, (b) bei $T = 1.55\ \frac{\varepsilon}{k}$ und $\xi = 0.9$.

(Abbildung 4.6 a und 4.7 a) ist die Übereinstimmung der Koexistenzkurven zwischen Simulation und Theorie relativ gut. Während die Gasseite (rechter Flügel in 4.6 a) bei tiefen Drücken sehr gut mit der Simulation übereinstimmt, wird in der flüssigen Phase (linker Flügel in 4.6 a) der Anteil der Polymere von der Theorie unterschätzt. Eine ähnliche Aussage lässt sich auch für die Zusammensetzung der Koexistenzdichten (Abb. 4.7 a) treffen. In der Nähe des kritischen Punktes weichen die Ergebnisse erwartungsgemäß ab. Mit (4.20) kann die Mean–Field–Spinodale ermittelt werden (gepunktete

Linie in Abbildung 4.6 a). Würde man die Spinodale mittels linearer Extrapolation aus dem Ein–Phasen–Gebiet bestimmen, erwartet man ein ähnliches Ergebnis wie für das Lennard–Jones–System in Kapitel 4.2. Die Spinodale würde abermals näher an der Koexistenzlinie liegen.

Wie bereits in Abbildung 4.4 demonstriert, sind die Abweichungen der kritischen Punkte des Mischsystems bei hohen Temperaturen größer. Für $T = 1.55\ \frac{\varepsilon_5}{k}$ und $\xi = 0.9$ wird der kritische Druck um mehr als 100% überschätzt (Abbildung 4.6 b). Ähnliches gilt auch für die kritische Dichte. Die Zusammensetzung der flüssigen Phase (Abbildung 4.6 b) stimmt bis ungefähr $p = 0.115\ \frac{\varepsilon_5}{\sigma_5^3}$ sehr gut mit den Simulationen überein. Die Übereinstimmung für die Gasseite ist relativ schlecht, was auf einen verstärkten Einfluss der in der Theorie vernachlässigten Dichtefluktuationen hindeutet. Die Koexistenzdichten fernab des kritischen Punktes werden sowohl für die Flüssig– als auch für die Gasphase gut vorausgesagt.

Als Letztes soll untersucht werden, welche Näherungen in erster Linie für die Unterschiede zwischen Theorie und Simulation im Mischsystem verantwortlich sind. Punktuelle Vergleiche mit alten Simulationsdaten in [50] legen nahe, dass für Mischungen von Lennard–Jones–Teilchen eine bessere Übereinstimmung erzielt wird. Abbildung 4.8 zeigt den Verlauf von Druck und Dichte eines solchen Systems bei $T = 1.5\ \frac{\varepsilon_{\mathrm{LJ2}}}{k}$ und verschiedenen chemischen Potentialen als Funktion der Zusammensetzung. Die Übereinstimmung zwischen Simulation und Experiment ist auffallend gut. Das Referenzsystem wird also zumindest bei hohen Temperaturen gut reproduziert, was sich u.U. auch durch den großen Abstand von den kritischen Punkten erklären lässt ($T_{\mathrm{c}}^{\mathrm{LJ1}} = 0.725\ \frac{\varepsilon_2}{k}$ und $T_{\mathrm{c}}^{\mathrm{LJ2}} = 0.999\ \frac{\varepsilon_2}{k}$). Dennoch legen die Resultate eher nahe, dass Abweichungen für Polymermischungen auf den Kettenterm zurückzuführen sind. So fließt in (4.11) z.B. die radiale Verteilungsfunktion $g(r)$ in die „mean spherical approximation“ mit ein, die bereits beim reinen Fünfmer–System eine Abweichung im Phasenverhalten bewirkte (vgl. Kapitel 4.2). Für eine abschließende Bewertung wären jedoch weitere Studien notwendig.

Zusammenfassend kann TPT1 als wertvolles Werkzeug zur effizienten Untersuchung des Phasenverhaltens von Polymermischungen betrachtet werden. Wie der Vergleich mit den Simulationen zeigt, wird sowohl der Typ als auch der qualitative Verlauf eines Mischsystem–Phasendiagramms gut reproduziert. Kritische Punkte werden wie in anderen „mean field“–artigen Theorien überschätzt. Im Laufe der Untersuchungen ergaben sich zudem gezielte Hinweise auf Verbesserungsansätze der Theorie.

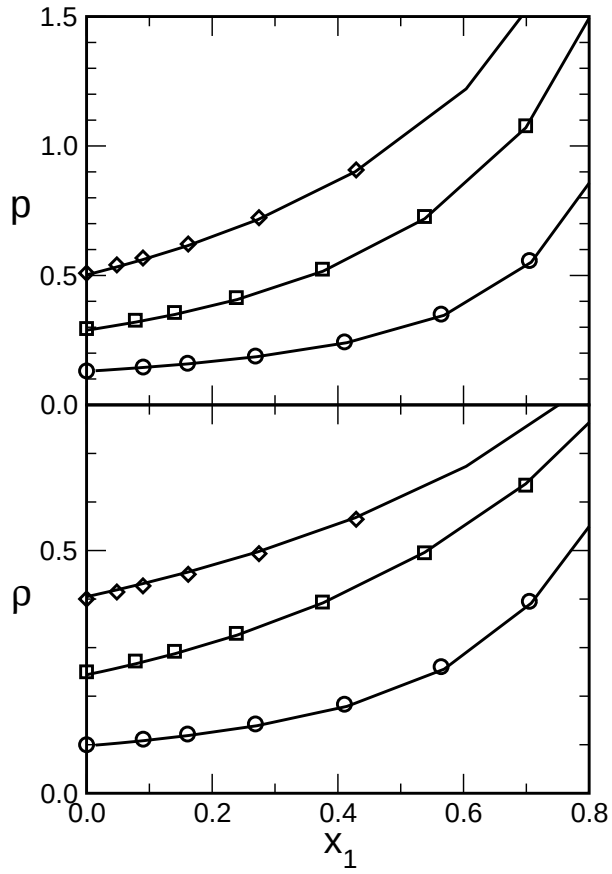

Abbildung 4.8: Druck (oben) und Dichte (unten) einer binären Lennard–Jones–Mischung als Funktion des Molbruchs bei festem $T = 1.5\ \frac{\varepsilon_2}{k}$ und μ_2. Die Symbole sind Simulationsergebnisse (Kreise: $\mu_2 = -3.847$, Quadrate: $\mu_2 = -2.842$, Diamanten: $\mu_2 = -2.186$). TPT1–Berechnungen werden durch Linien gekennzeichnet. Die Wechselwirkungsparameter entsprechen denen der Polymermischung, d.h. $\frac{\sigma_1}{\sigma_2} = 0.816$, $\frac{\varepsilon_1}{\varepsilon_2} = 0.726$ und $\xi = 1$.

Kapitel 5

Untersuchungen zum Nukleationsverhalten in endlichen Systemen

In Kapitel 3 und 4 wurde das Phasenverhalten des Hexadekan–CO_2–Referenzsystems ausführlich diskutiert und die Simulationsergebnisse mit Experimenten und Störungsrechnungen verglichen. Hierbei hat sich u.A. gezeigt, dass das reale System sehr gut durch das hier verwendete Kugel–Feder–Modell beschrieben werden kann. Im letzten Kapitel soll nun das Nukleationsverhalten näher betrachtet werden.

In endlichen Systemen, also in allen Computersimulationen, lässt sich der Übergang von einem mehr oder minder homogenen, übersättigten Gas zu einem einzelnen Tropfen durch einen Phasenübergang erster Ordnung beschreiben. Obwohl dies im Prinzip schon seit geraumer Zeit bekannt ist [87], wurden genauere analytische Betrachtungen für das Ising–Modell erst vor Kurzem durchgeführt [37], [88]. Nach einer kurzen phänomenologischen Motivation (Kapitel 5.1) erfolgt zum ersten Mal ein direkter Nachweis des Übergangs für Lennard–Jones–Flüssigkeiten (Kapitel 5.2). In diesem Zusammenhang zeigt sich, dass der Phasenübergang zu einer Verlangsamung und letztlich zu einer Blockade multikanonischer Simulationen großer Systeme führt (vgl. hierzu [89]). Im Experiment lässt sich Nukleation durch einen kinetischen Prozess beschreiben. Nach einem Sprung ins metastabile Gebiet muss ein Cluster durch Fluktuationen über einen kritischen Radius hinauswachsen und so die Keimbildungsbarriere überwinden. Hierbei hat er genügend Zeit seine Form zu optimieren. Aufbauend auf diesen Überlegungen und der in Kapitel 5.2 vorgenommenen Identifizierung des Clusterbereiches wird in Kapitel 5.3 die freie Energie von kleinen Blasen und Tropfen als Funktion des Radius untersucht und mit Hilfe eines einfachen Modells nach oben abgeschätzt. Alle Betrachtungen wurden hierbei erst durch den gezielten Einsatz

der in Kapitel 2 entwickelten Simulationstechniken ermöglicht. Kapitel 5.4 bietet zum Abschluss einen kurzen, qualitativen Ausblick auf die Kinetik der Entmischung und hierzu in Beziehung stehende Benetzungsphänomene im Hexadekan–CO_2–System.

5.1 Phänomenologische Motivation

Keimbildungsprozesse werden durch klassische Nukleationstheorie nur unzureichend beschrieben (vgl. z.B. mit Ref. [90], [91]). Simulationen bieten sich in diesem Zusammenhang als ideales Werkzeug zur näheren Untersuchung an. Während bei heutigen Experimenten mikroskopische Informationen indirekt aus der Nukleationsrate bestimmt werden [79], sind diese Computersimulationen direkt zugänglich. Da aber alle Simulationen auf endliche Systeme beschränkt sind, kommt einem genauen Studium der Finite-Size–Effekte eine besondere Bedeutung zu. Diese wurden bei bisherigen Betrachtungen weitgehend ignoriert [91], [92].

Zunächst soll eine einfache, phänomenologische Überlegung begründen, wie im Rahmen des multikanonischen Ansatzes Phasenübergänge in endlichen Systemen untersucht werden können. Konkret geht es dabei um die Fragestellung, welche Konfigurationen bei gegebener Dichte und Systemgröße in der Simulationsbox anzutreffen sind und wie sich die entsprechenden Übergänge gestalten. Die hier vorgestellte Theorie bildet zudem die Grundlage für die Analyse der Simulationsdaten in Kapitel 5.3. Ausführlichere Beschreibungen insbesondere zum Gas–Tropfen–Übergang finden sich in [37], [93].

Großkanonische Simulationen werden bei festgelegtem Volumen, konstanter Temperatur und konstantem chemischen Potential durchgeführt. Wie bereits in Kapitel 1.6 ausgeführt, lässt sich die Helmholtz'sche freie Energie aus der großkanonischen Wahrscheinlichkeitsverteilung bestimmen:

$$F = -kT\ \ln(P(n)) + \text{konst.} \tag{5.1}$$

Die freie Energie der homogenen Gasphase ist gegeben durch:

$$F(n_\mathrm{g})(T,V,n) = U - TS = -p_\mathrm{g}V + \mu_\mathrm{g}n_\mathrm{g}. \tag{5.2}$$

In einer kleinen Umgebung von n_g lässt sich F durch eine Taylor–Entwicklung darstellen:

$$F \approx F(n_\mathrm{g}) + F'(n_\mathrm{g})(n-n_\mathrm{g}) + \frac{1}{2}F''(n_\mathrm{g})(n-n_\mathrm{g})^2 \tag{5.3}$$

mit

$$\frac{1}{2}F''(n_\mathrm{g}) = \frac{1}{2}\frac{V}{n_\mathrm{g}^2}\left(\frac{\beta V}{n_\mathrm{g}^2}\left(\frac{\partial n}{\partial \beta\mu}\right)_{\mathrm{T,V}}\right)^{-1} = \frac{1}{2}\frac{V}{n_\mathrm{g}^2}\left(\beta V\frac{\langle\delta n^2\rangle}{n_\mathrm{g}^2}\right)^{-1} = \frac{V}{2n_\mathrm{g}^2\kappa}. \tag{5.4}$$

$F'(n_g) = \mu_{koex} = 0$, da der Koexistenzwert des chemischen Potentials als Nullpunkt betrachtet wird. $V/2n_g^2\kappa$ ist konstant und kann durch einen „Fit“ an $F(n)$ in der Umgebung der Gasphase bestimmt werden. Alternativ kann man auch κ direkt aus der Halbwertsbreite der Wahrscheinlichkeitsverteilung ablesen.

Ab einer gewissen Dichte ist es für das System vorteilhaft, einen Tropfen zu generieren. In einer „Näherung nullter Ordnung“ kann man annehmen, dass der Gashintergrund Koexistenzdichte aufweist und die zusätzlichen Teilchen einen Tropfen mit Flüssigkeitsdichte bilden. Dann ist

$$\frac{4}{3}\pi R^3 \rho_f + \left(V - \frac{4}{3}\pi R^3\right)\rho_g = \rho V \tag{5.5}$$

bzw.

$$R = \sqrt[3]{\frac{V}{\frac{4}{3}\pi}\,\Delta\rho} \quad \text{mit} \quad \Delta\rho := \frac{(\rho - \rho_g)}{(\rho_f - \rho_g)}. \tag{5.6}$$

In diesem Fall ist $\mu_f = \mu_g = \mu_{koex} = 0$ und somit $p_f = p_g = p_{koex}$. Es folgt:

$$F = -p_f V_f - p_g V_g + \mu_g n_g + \mu_f n_f + 4\pi\gamma R^2 = -p_g V + \mu n_g + 4\pi\gamma R^2 = F(n_g) + 4\pi\gamma R^2. \tag{5.7}$$

γ entspricht in der so genannten „Kapillaritätsapproximation“ der Grenzflächenspannung einer flachen Grenzfläche. Es sollte an dieser Stelle angemerkt werden, dass der Gashintergrund nur im Grenzfall $R \to \infty$ Koexistenzdichte aufweist. Für endliches V und endliches R ist er hingegen übersättigt, d.h. $\mu = \partial F/\partial n > \mu_{koex}$. Entwickelt man jedoch $\mu n - pV$ um die Koexistenzwerte, so heben sich die ersten Ableitungen wegen $\rho\Delta\mu = \Delta p$ gerade weg. Für große R sollte die Näherung also gute Resultate liefern. Der Gültigkeitsbereich des hier vorgestellten Modells wird ausführlich in Kapitel 5.3 diskutiert.

Mit weiter anwachsender Dichte bildet das System irgendwann eine gerade Grenzfläche aus (vgl. mit Kapitel 1.6). In einem Teil der Box befindet sich dann Flüssigkeit und im anderen Gas. Es gilt:

$$F = F(n_g) + 2\gamma L^2 \ \text{ unabhängig von } \Delta\rho. \tag{5.8}$$

Für eine gegebene Dichte wird der Zustand mit der kleinsten freien Energie eingenommen. Abbildung 5.1 a zeigt die durch Simulation bestimmte Verteilung im Vergleich zur analytischen Voraussage. Der wahre Verlauf wird lediglich qualitativ reproduziert. So erhält man zunächst ein homogenes Gas, mit ansteigender Dichte einen Tropfen und schließlich eine lineare Grenzfläche. Analog lässt sich bei weiter anwachsender Dichte die Bildung einer Blase motivieren. Die jeweiligen Übergänge werden durch Gleichsetzen von (5.3) mit (5.7) unter Berücksichtigung von (5.6) bzw. durch

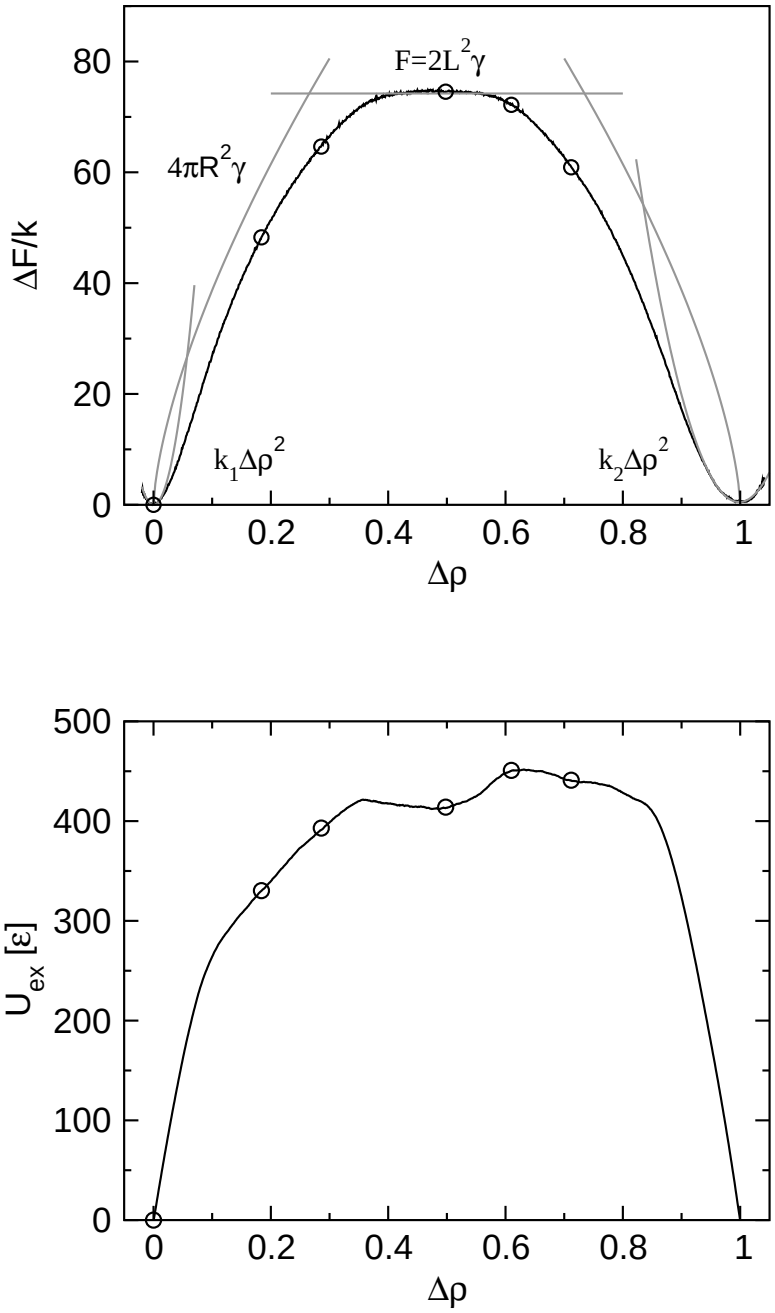

Abbildung 5.1: (a) Freie Energie als Funktion von $\Delta\rho$ ($L = 11.3\,\sigma$, $T = 0.78\,\frac{\varepsilon}{k}$, $\gamma = 0.291\,\frac{\varepsilon}{\sigma^2}$) im Vergleich zur analytischen Berechnung. (b) Exzess–Innere–Energie für das gleiche System. ○ markieren Dichten für die typische Konfigurationen in Abbildung 5.2 visualisiert wurden.

Gleichsetzen von (5.7) mit (5.8) bestimmt. Hierbei ergibt sich für die Übergänge von Gas (Flüssigkeit) zu Tropfen (Blase) ein von der Systemgröße abhängiger Schnittpunkt:

$$\Delta\rho_t = aL^{-\frac{3}{4}} \text{ mit } a = 2\kappa\, 4\pi\gamma \frac{\rho_g^2}{\left(\rho_f - \rho_g\right)^2} \left(\frac{4}{3}\pi\right)^{-\frac{2}{3}}. \tag{5.9}$$

Die Schnittpunkte auf der Gas– und der Flüssigseite in Abbildung 5.1 a unterscheiden sich aufgrund der unterschiedlichen Kompressibilitäten. Für den Übergang vom Tropfen (Blase) zur ebenen Grenzfläche erhält man einen von der Systemgröße abhängigen Wert von $\Delta\rho = 0.266\ (0.734)$. Die Theorie sagt an den Schnittpunkten einen Phasenübergang erster Ordnung voraus, da das chemische Potential $(\partial F/\partial n)$ an dieser Stelle springt. Die Unstetigkeiten sind in dem hier simulierten System verschmiert und treten erst bei sehr großen Boxen deutlich hervor (vgl. mit Kapitel 5.2).

Als Alternative zur freien Energie können auch Ausdrücke herangezogen werden, die die Entropie explizit berücksichtigen. Da sich diese bei der Bildung von Tropfen, Blasen und linearen Grenzflächen ändert, lassen sich die Übergänge auch bei kleinen Boxen leichter identifizieren. Bei der Exzess–Energie (Abb. 5.1 b) handelt es sich um die innere Energie des Systems ($U = F + TS$), die durch eine geeignete Normierung für Gas und Flüssigkeit Null wird:

$$U_{ex} = U - (U_f - U_g)\,\Delta\rho\ - U_g. \tag{5.10}$$

Für hohe Temperaturen ist auch $U_{ex}(n)$ stärker geglättet. Sowohl Abb. 5.1 a als auch Abb. 5.1 b deuten an, dass es sich bei Blasen– und Tropfenbildung um asymmetrische Prozesse handelt.

Zum Schluss dieser Einführung sollen die qualitativen Überlegungen durch eine Auswahl von Schnappschüssen bestätigt werden. Abbildung 5.2 a–f zeigt typische Konfigurationen eines Lennard–Jones–Systems bei Temperatur $T = 0.78\ \frac{\varepsilon}{k}$. Bei $\Delta\rho = 0$ liegt homogenes Gas vor (Abb. 5.2 a). Mit steigender Dichte bildet sich ein einzelner Tropfen (Abb. 5.2 b), der bei kleinen Systemen im Übergangsbereich zur ebenen Grenzfläche über die periodischen Boxenränder hinaus wechselwirkt (Abb. 5.2 c). In der Umgebung von $\Delta\rho = 0.5$ erhält man erwartungsgemäß eine scheibenförmige Konfiguration, bei der die eine Hälfte der Box gasförmig und die andere Hälfte flüssig ist (Abb. 5.2 d). Mit weiter anwachsender Dichte bildet sich bei kleinen Systemen zunächst eine ellipsoide Blase (Abb. 5.2 e), die sich schließlich immer weiter abrundet (Abb. 5.2 f). Bei $\Delta\rho = 1$ ist die Box mit homogener Flüssigkeit gefüllt. Die hier dargestellten graduellen Übergänge (Abb. 5.2 c und e) sind eine Folge der kleinen Systemgröße. Für sehr große Systeme werden die Übergänge scharf (vgl. Kapitel 5.2). Die von der Theorie und der Betrachtung der Exzess–Energie vorhergesagten Regionen und Übergangsbereiche stimmen in etwa mit den tatsächlich gefunden Konfigurationen in der Simulation überein. Während die Existenz einer flachen Grenzfläche schon

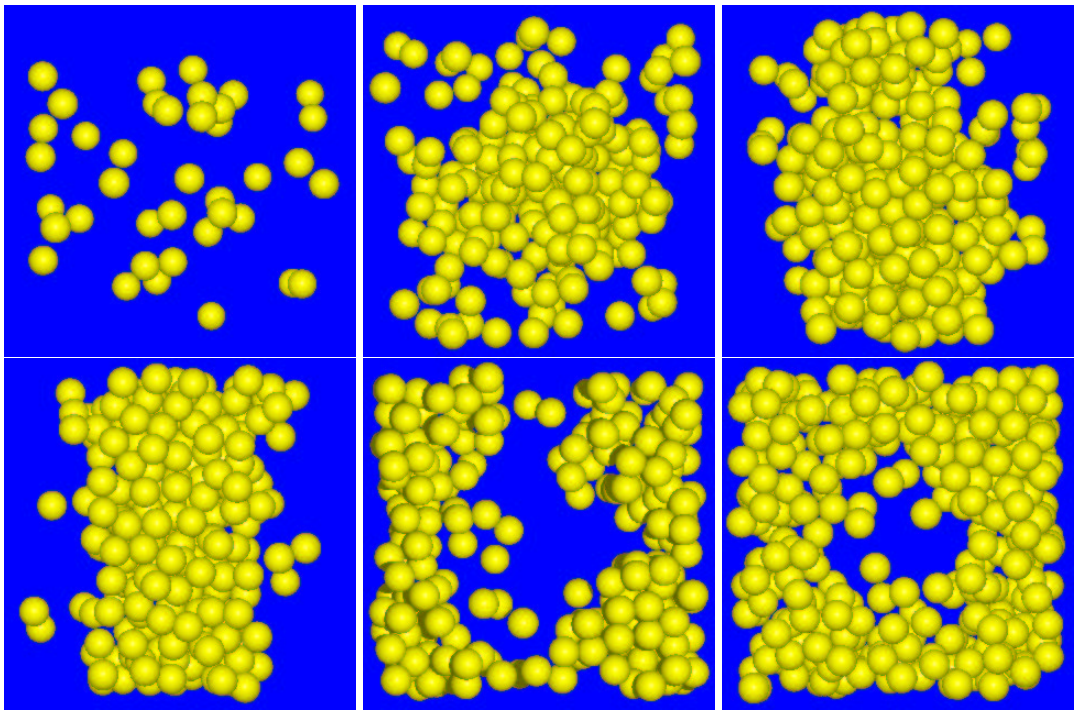

Abbildung 5.2: Typische Konfigurationen eines Lennard–Jones–Systems ($L = 11.3\ \sigma$, $T = 0.78\ \frac{\varepsilon}{k}$, $\gamma = 0.291\ \frac{\varepsilon}{\sigma^2}$) bei ausgewählten Dichten ($\Delta\rho = 0, 0.184, 0.286, 0.498, 0.610, 0.712$; vgl. Abb. 5.1) Bei der unteren Reihe wird aus Gründen der Übersicht lediglich ein Schnitt durch die Hälfte der Simulationsbox gezeigt. Der Radius der Kugeln beträgt $0.56\ \sigma$ (Minimum im LJ–Potential $= 2r \approx 1.12\ \sigma$).

recht lange bekannt ist und bei der Bestimmung der Grenzflächenspannung Anwendung findet [36], wurde der Übergang vom Gas zum Tropfen bisher in erster Linie analytisch untersucht [37], [88]. Im folgenden Unterkapitel soll zum ersten Mal ein direkter Simulationsnachweis des Phasenüberganges für das Lennard–Jones–System geführt werden.

5.2 Der Verdampfungs–/Kondensationsübergang

In einer ersten Versuchsreihe ($L = 6.74\ \sigma$ bis $L = 13.5\ \sigma$) wurden bei $T = 0.68\ \frac{\varepsilon}{k} \approx 0.68\ T_c$ Wahrscheinlichkeitsverteilungen mit Hilfe der in Kapitel 2.3 beschriebenen, neuen Simulationsmethode erzeugt. Für das größte System wurde zusätzlich in der Umgebung des vermuteten Übergangs eine ausführliche multikanonische Simulation durchgeführt und typische Konfigurationen zur späteren Analyse herausgeschrieben.

Es stellte sich jedoch heraus, dass der Übergang erst bei relativ großen Systemen deutlich hervortritt. Aus diesem Grund wurde eine zweite Versuchreihe mit größeren Boxen ($L = 15.8\ \sigma$ bis $L = 22.5\ \sigma$) initiiert, deren Ergebnisse im Folgenden vorgestellt werden. Die Gewichtsfunktionen der zweiten Reihe wurden mit Hilfe des Wang–Landau–Algorithmus erzeugt (vgl. Kapitel 2.2).

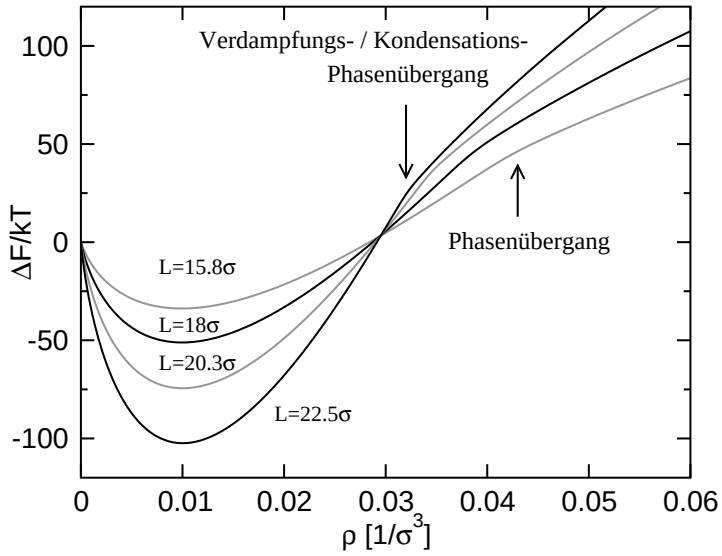

Abbildung 5.3: Freie Energie als Funktion der Dichte für große LJ–Systeme bei Koexistenz und $T = 0.68\ \frac{\varepsilon}{k}$. Der Phasenübergang vom übersättigten Gas zum Tropfen (Pfeile) verschiebt sich mit zunehmender Systemgröße zu kleineren Dichten. Die freie Energie wurde aus der Wahrscheinlichkeitsverteilung der Gewichtsfunktion bestimmt $(F/kT = -\ln P(n))$.

Abbildung 5.3 zeigt die freie Energie als Funktion der Dichte für verschiedene Systemgrößen. Da Simulationen über den dargestellten Dichtebereich sehr rechenintensiv sind, wurde ΔF direkt aus den Gewichtsfunktionen bestimmt. Die Kurven schneiden sich bei $\rho \approx 0.03\ \frac{1}{\sigma^3}$. Liegt ein homogenes Gas vor, so sind bei gegebener Dichte und periodischen Randbedingungen μ und ρ unabhängig von V. Zwei $F(\rho, V)$ Kurven schneiden sich, wenn $\mu\rho(V_1 - V_2) - p(V_1 - V_2) = 0$. Dies ist jedoch immer für $F = 0 \leftrightarrow p = \mu\rho$ erfüllt. Der etwas nach oben verschobene Schnittpunkt erklärt sich durch Grenzflächenbeiträge kleiner Cluster. Für $L = 22.5\ \sigma$ erkennt man deutlich einen Knick in der freien Energie, der mit kleiner werdender Boxgröße immer weiter abflacht und sich wie erwartet zu größeren Dichten verschiebt. Dies kann als erstes Indiz für einen Phasenübergang erster Ordnung gewertet werden. Zur genaueren Untersuchung wurde für das größte System ($L = 22.5\ \sigma$) eine umfangreiche multikanonische Simulation in der direkten Umgebung des Knicks durchgeführt. Abbildung 5.4 a zeigt an dieser Stelle einen Sprung im chemischen Potential, also in der ersten Ableitung der freien Energie. Die zweite Ableitung (Abb. 5.4 b) wird minimal für $n \approx 365$. Gemäß

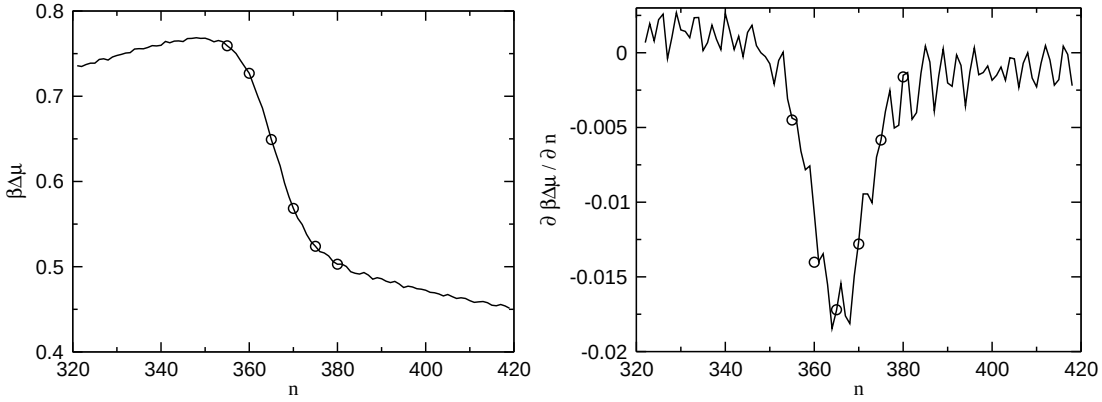

Abbildung 5.4: (a) Sprung im chemischen Potential $(\partial F/\partial n)$ am Verdampfungs– / Kondensationsübergang für ein LJ–System der Größe $L = 22.5\ \sigma$ bei $T = 0.68\ \frac{\varepsilon}{k}$, (b) Ableitung des chemischen Potentials nach der Teilchenzahl. Bei ○ wurden Konfigurationen zur späteren Analyse (Abb. 5.5 bis 5.7) gespeichert.

[93] entspricht diese Stelle gerade dem Übergangspunkt. Der Phasenübergang unterscheidet sich z.B. vom gewöhnlichen Gas–flüssig–Übergang. Bei Letzterem können Gas und Flüssigkeit gleichzeitig in der Simulationsbox koexistieren. Beide Phasen sind dann durch eine wohldefinierte Grenzfläche getrennt (vgl. Abb. 5.2 d). Da sich in dem hier besprochenen Fall der Übergang vom übersättigten Gas mit kleinen Clustern zum Tropfen in Umgebung eines etwas weniger übersättigten Gases vollzieht, gibt es keine Grenzfläche. Vielmehr kann (bei konstanter Dichte) ein einzelner Cluster über eine (systemgrößenabhängige) kritische Größe hinauswachsen und so einen neuen Gleichgewichtszustand (als Tropfen) erreichen.

Zur genaueren Analyse wurden während des Laufs Konfigurationen bei charakteristischen Dichten (○ in Abb. 5.4) gespeichert und im Anschluss analysiert. Die Auswertung zu den Abbildungen 5.5 und 5.7 a wurde von *L.G. MacDowell* vorgenommen. Abbildung 5.5 zeigt die Verteilungen der Clustergrößen im System, die mit Hilfe eines Stillinger–Kriteriums [94] bestimmt wurden. Ein Teilchen gehört zu einem Cluster, wenn der Abstand zu ihm weniger als $1.5\ \sigma$ beträgt. Dies ist durchaus gerechtfertigt, wenn man diesen Wert z.B. mit dem durchschnittlichen Teilchenabstand in der Gas– ($4.64\ \sigma$ bei $T = 0.68\ \frac{\varepsilon}{k}$) und der Flüssigkeitsphase ($1.09\ \sigma$) vergleicht. Im ansteigenden Bereich von Abbildung 5.4 ($n < 350$) fällt $P(N_c)$ monoton mit der Clustergröße (hier nicht gezeigt). Das System ist mehr oder minder homogen. Gelegentlich entstehen durch Fluktuationen kleine Cluster, die wieder zerfallen. Ab $n \approx 355$ (abfallender Bereich in Abb. 5.4) bildet sich in Abbildung 5.5 a ein zweiter Peak bei einer Clustergröße von etwa 120 heraus. Dieser tritt mit steigender Gesamtteilchenzahl immer stärker hervor und verschiebt sich zu größeren Clustern. Vergleicht man die Gesamtverteilung mit der Verteilung der größten Cluster im System (Abbildung 5.5 b), so

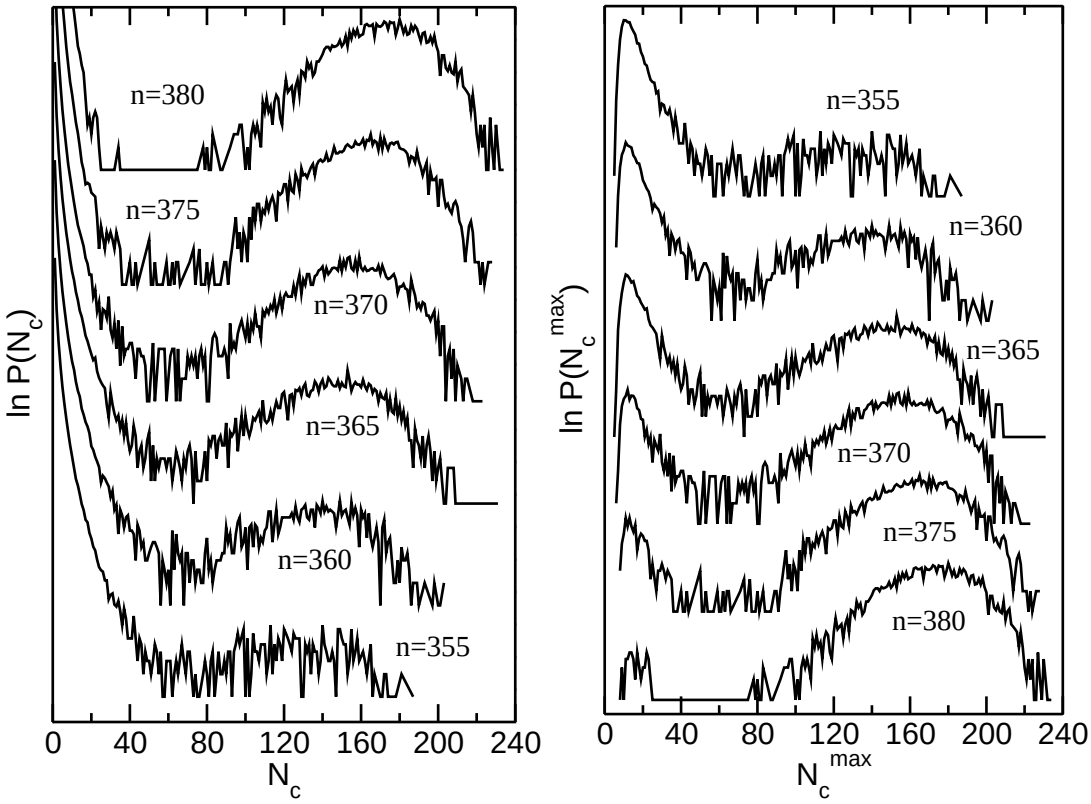

Abbildung 5.5: Verteilung der Clustergrößen für ein Lennard–Jones–System der Größe $L = 22.5\ \sigma$ bei $T = 0.68\ \frac{\varepsilon}{k}$ und verschiedenen Dichten in der Nähe des Verdampfungs–/Kondensationsüberganges. Bei (a) wurden alle Cluster, bei (b) lediglich der größte Cluster in jeder Konfiguration berücksichtigt.

sind die rechten Peaks bei gleichen Dichten deckungsgleich. Die hierzu gehörenden Konfigurationen werden also von einem einzelnen großen Tropfen dominiert und das System befindet sich in der kondensierten Phase. Es gibt allerdings auch Konfigurationen, die keinen großen Cluster aufweisen (linke Maxima in Abb. 5.5 b). In diesem Fall befindet sich das System in der übersättigten Gasphase. Der linke Flügel in Abbildung 5.5 a deutet in diesem Zusammenhang bereits die abnehmende Übersättigung des Gashintergrundes bei zunehmender Tropfengröße in der kondensierten Phase an. Die Wahrscheinlichkeit durch Fluktuationen einen Cluster mittlerer Größe ($N_c \approx 40$) zu bilden nimmt mit wachsender Gesamtteilchenzahl ab. Bei $n = 380$, also bei einer Dichte bei der fast nur noch Konfigurationen mit Tropfen existieren, steht der linke Flügel in 5.5 a nur noch für den homogenen Gashintergrund.

Wendet man zur Lokalisierung des Phasenübergangs eine Regel der gleichen Gewichte an (vgl. Kapitel 1.3), so erhält man Flächengleichheit in Abbildung 5.5 b bei $n \approx 365$. Dies entspricht im Rahmen der statistischen Genauigkeit gerade wieder dem

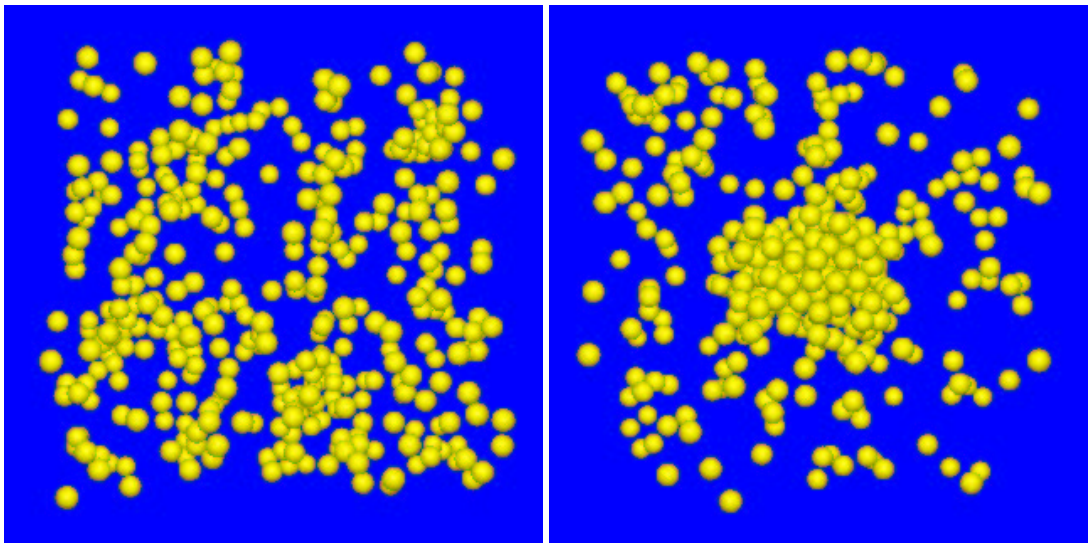

Abbildung 5.6: Schnappschüsse von typischen Konfigurationen eines Lennard–Jones–Systems am Phasenübergang ($n = 365$ Teilchen, vgl. mit Abb. 5.5): (a) übersättigte Gasphase, (b) kondensierte Phase. Der Radius der Kugeln beträgt 0.56 σ (Minimum im LJ–Potential $= 2r \approx 1.12\,\sigma$).

Wert der kleinsten Steigung im chemischen Potential (Abb. 5.4 b). Für diesen Wert wurden in Abbildung 5.6 zwei Konfiguration visualisiert. Die linke Darstellung zeigt die übersättigte Gasphase mit kleinen Clustern. Das rechte Bild entspricht einer typischen Konfiguration in der Tropfenphase.

Die charakteristische bimodale Verteilung lässt sich auch in anderen thermodynamischen Größen nachweisen – z.B. in der Verteilung des chemischen Potentials im übersättigten Gas. Hierzu legt man um das Massenzentrum des größten Clusters im System eine Kugel mit Radius $R = \left(\frac{N}{4/3\pi\rho_f}\right)^{1/3} + \sigma$ (vgl. Gleichung (5.6) mit $\rho_g \ll \rho_f$) und schließt diesen Bereich von der Analyse aus. Im verbleibenden Bereich bestimmt man μ mit Hilfe der „Widom–Teilchen–Einsetzmethode“ [20], [95]. Aus

$$\mu = \left(\frac{\partial F}{\partial n}\right) \approx \frac{F(n+1) - F(n)}{1} = -kT \ln \frac{Z(n+1)}{Z(n)} \tag{5.11}$$

folgt nach Separation der kinetischen Beiträge (vgl. mit Vorgehensweise in Kap. 4.1):

$$\beta\mu = -\ln \frac{V/\Lambda^3}{N+1} + \frac{\int e^{-\beta U} dr^{3(N+1)}}{\int e^{-\beta U} dr^{3N}} = \ln\rho + \ln\Lambda^3 - \ln\langle e^{-\beta\Delta U}\rangle. \tag{5.12}$$

Die Dichte kann direkt gemessen werden und Λ wird gleich Eins gesetzt. Für jede abgespeicherte Konfiguration setzt man zufällig ein Testteilchen in das Gas ein, bestimmt die Zunahme der Energie ΔU und entfernt es wieder. Dieser Vorgang wird vielfach wiederholt und so $\ln\langle e^{-\beta\Delta U}\rangle$ bestimmt.

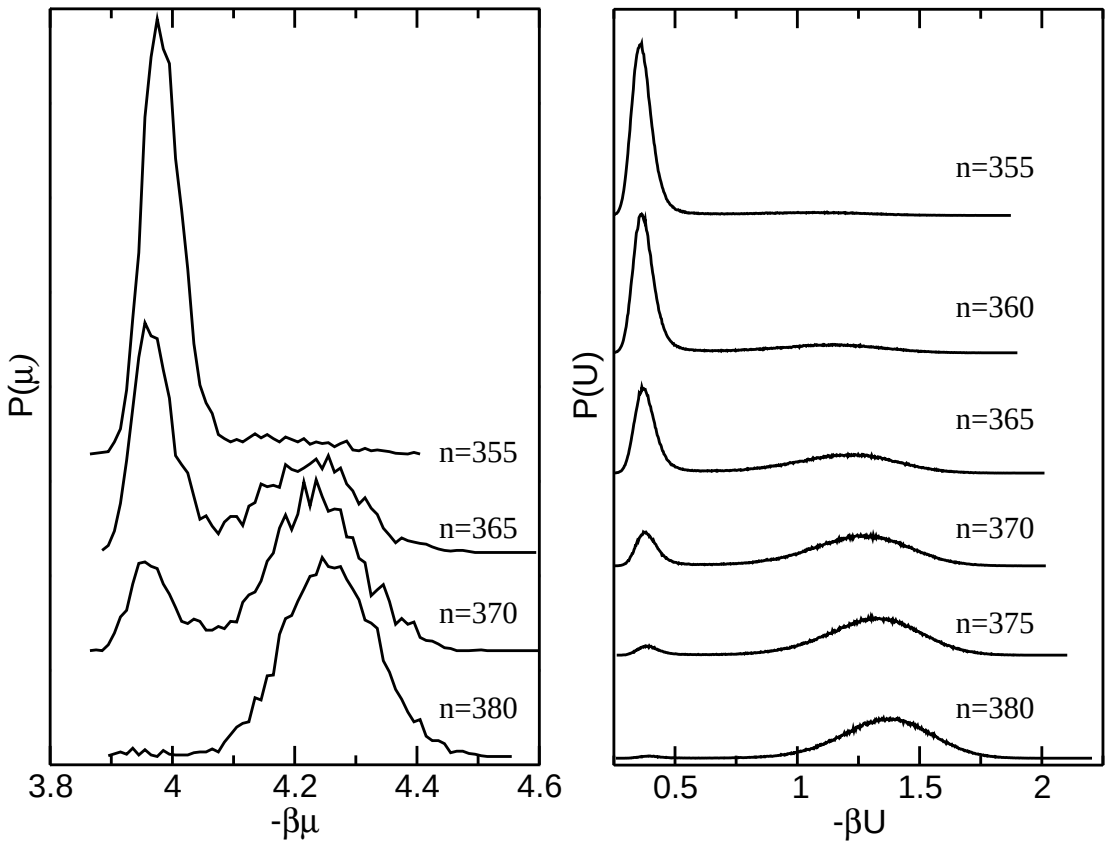

Abbildung 5.7: (a) Verteilung der chemischen Potentiale des übersättigten Gases, (b) Verteilung der Energie pro Teilchen. Die Systemparameter entsprechen denen von Abbildung 5.5.

In der Umgebung eines Tropfens (rechte Maxima in Abbildung 5.7 a) ist das chemische Potential des umgebenden Gases geringer als in der reinen Gasphase. Die Übersättigung nimmt also bei der Kondensation zum Tropfen ab. Wegen $\mu_f = \mu_g$ erhält man für das chemische Potential in der gesamten Box die gleiche Darstellung. Der Abstand der beiden Peaks entspricht dabei genau der Höhe des Sprunges in Abbildung 5.4 a. Die Verteilung der Energie pro Teilchen hat am Phasenübergang ebenfalls zwei Maxima (Abbildung 5.7 b). U wurde während des multikanonischen Laufs direkt aufgezeichnet und nicht aus den gespeicherten Konfigurationen bestimmt, was die bessere Statistik erklärt. Im Tropfen können sich viele Teilchen in der Nähe des Potentialminimums anordnen, in der supersaturierten Gasphase wechselwirken aufgrund der größeren Abstände weitaus weniger Teilchen und die Energie ist folglich größer. Würde man beide Anteile addieren, erhielte man am Phasenübergang wie schon beim chemischen Potential einen Sprung. Dies erklärt u.A. den Verlauf der Exzess–Energie in Abbildung 5.1 b. Die Darstellungen 5.5 und 5.7 sind prinzipiell gleichwertig und liefern die gleiche Übergangsdichte. Eine analoge Analyse kann auch für den Übergang von der Blase zur Flüssigkeit durchgeführt werden. Der Aufwand an Rechenzeit

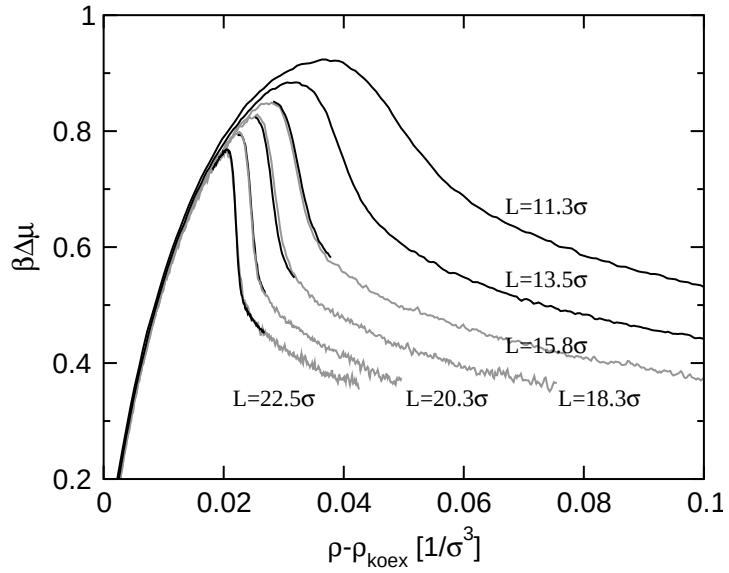

Abbildung 5.8: Chemisches Potential als Funktion der Dichte für verschiedene Systemgrößen.

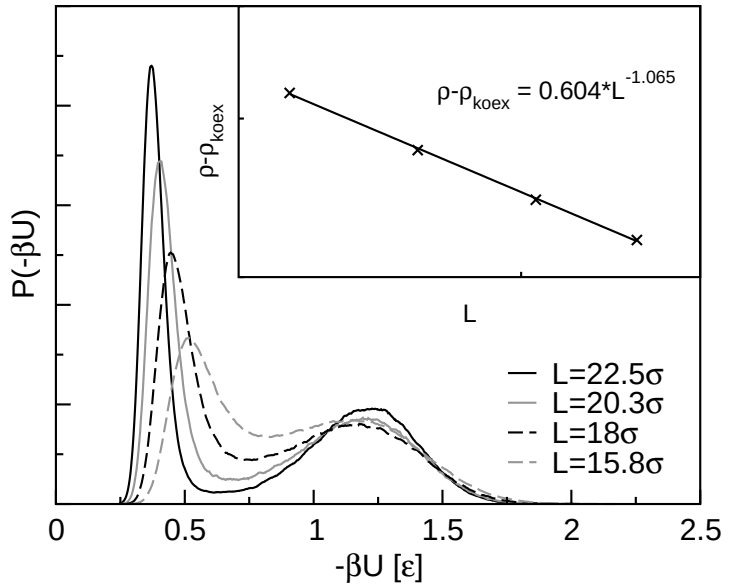

Abbildung 5.9: Verteilung der Energie U pro Teilchen für verschiedene Systemgrößen am Verdampfungs–/Kondensationsübergang. Kleines Bild: $\rho - \rho_{\mathrm{koex}}$ als Funktion der Boxgröße L in doppelt–logarithmischer Auftragung ($L = 15.8\,\sigma : 0.0320\,\frac{1}{\sigma^3}, L = 18\,\sigma : 0.0277\,\frac{1}{\sigma^3}, L = 20.3\,\sigma : 0.0244\,\frac{1}{\sigma^3}, L = 22.5\,\sigma : 0.0220\,\frac{1}{\sigma^3}$).

ist dann jedoch aufgrund der höheren Gesamtdichte ungleich größer.

Zum Abschluss soll kurz die eingangs erwähnte Systemgrößenabhängigkeit diskutiert werden. Abbildung 5.8 zeigt das chemische Potential als Funktion der Dichte für verschieden Systemgrößen. Der Phasenübergang bewegt sich mit steigender Boxgröße in Richtung Koexistenz und wird schärfer ausgeprägt. Für sehr kleine Systeme „verschmiert“ der Übergang und man erhält eine reguläre Van–der–Waals–Schleife. Im homogenen Gasbereich fallen die Kurven erwartungsgemäß aufeinander.

Abbildung 5.9 zeigt die Verteilung der Energie eines Teilchens für verschiedene Boxgrößen. Für große L sind Gas– und Tropfenphase sauber getrennt. Für kleinere Boxen werden beide Maxima breiter und es entsteht ein Überlapp. Nach oberen Überlegungen gilt diese Aussage auch für die Verteilungen der Clustergrößen und des chemischen Potentials des Gashintergrunds. Verkleinert man das System weiter (hier nicht gezeigt), können übersättigte Gasphase und Tropfenphase nicht mehr unterschieden werden. Der linke Peak wandert mit wachsendem L zu geringeren Energien. Dies bedeutet, dass die Wechselwirkungen in der Gasphase und somit der Grad der Übersättigung bei größeren Boxen abnehmen. Die Wechselwirkungen in der flüssigen Phase (rechter Peak) bleiben hingegen mehr oder minder konstant. Trägt man die Differenz zwischen Übergangsdichte und Koexistenzdichte gegen die Boxgröße auf, so erhält man in der doppelt–logarithmischen Darstellung eine Gerade (kleines Bild). Die Abhängigkeit von $\Delta\rho$ und L lässt sich also durch ein Potenzgesetz beschreiben. Eine Bestimmung des Exponenten liefert einen Abfall mit $L^{-1.065}$, was von der theoretischen Voraussage $L^{-3/4}$ in Kapitel 5.1 und [37], [88] abweicht. Eine ähnliche Analyse kann auch für die Breite des Sprungbereiches durchgeführt werden. Definiert man diesen als Differenz zwischen dem Maximum in Abbildung 5.8 und dem Übergangspunkt, so erhält man einen Abfall mit etwa $L^{-3.06}$ statt $L^{-9/4}$ [37], [88]. Bei Letzterem ist die Statistik jedoch sehr schlecht. Zudem sollte in beiden Fällen bedacht werden, dass der untersuchte Bereich gerade einmal ein Zehntel einer Dekade umfasst. Zur genaueren Analyse der Größenabhängigkeit ist es deswegen vorteilhaft, das System mit Hilfe einer Zustandsgleichung numerisch zu lösen. Vorläufige Ergebnisse von *L.G. MacDowell* [93] deuten an, dass der theoretisch vorausgesagte Verlauf nur für sehr große Systeme erreicht wird.

Wie Abbildung 5.9 zeigt, werden Gas– und Flüssigphase durch eine mit L anwachsende Barriere getrennt. Dies ist einerseits typisch für einen Phasenübergang erster Ordnung, andererseits lassen sich hieraus technisch interessante Folgerungen ableiten. Simuliert man eine große Box multikanonisch mit einer Gewichtsfunktion, die lediglich die Teilchenzahl berücksichtigt, so gerät diese am Phasenübergang ins Stocken. Die großkanonischen Schritte sind nicht mehr in der Lage, das System effizient über die Barriere zu treiben (vgl. hierzu auch [89] und Kapitel 2.3.3). Große Simulationsfenster verzögern den Vorgang etwas, da die zweite Phase über den Umweg sich ändernder Dichten (und Verteilungen) erreicht werden kann. Aus diesem Grund wurde

statt des in Kapitel 2.3 vorgeschlagenen Simulationsschemas für die zweite Versuchsreihe die Wang–Landau–Methode (Kapitel 2.2) ohne eine zusätzliche Unterteilung in Unterfenster zur Bestimmung der Gewichtsfunktion herangezogen. Auf diese Weise können Boxen bis $L \approx 22.5\ \sigma$ simuliert werden. Möchte man größere Systeme untersuchen, empfiehlt es sich entweder kanonisch mit einer Gewichtung in der Energie zu simulieren oder aber aber großkanonisch mit einer zweidimensionalen Gewichtsfunktion (n, U).

5.3 Zum Verhalten kleiner Tropfen und Blasen

Experimente werden durch klassische Keimbildungstheorie nur unzureichend modelliert (vgl. z.B. mit [90], [91]). Im Folgenden wird die mittels Simulation bestimmte freie Energie von kleinen Tropfen und Blasen mit einem einfachen Modell verglichen, welches bereits in Kapitel 5.1 vorgestellt wurde. Bei der Keimbildung handelt es sich eigentlich um einen kinetischen Prozess. Dennoch ist die Untersuchung von Gleichgewichtskonfigurationen durchaus interessant. Nach einem Sprung ins metastabile Gebiet hat ein sich bildender Cluster auf dem Weg zur Überwindung der Keimbildungsbarriere in der Regel genügend Zeit, seine Form zu optimieren.

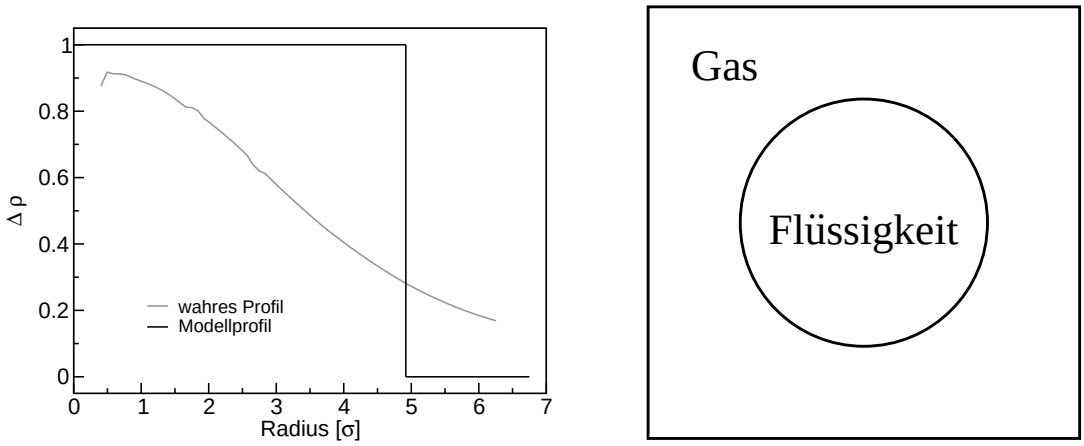

Abbildung 5.10: (a) (Gemitteltes) Dichteprofil eines Lennard–Jones–Clusters bei $T = 0.85437\ \frac{\varepsilon}{k}$ ($L = 13.5\ \sigma$, $\Delta\rho = 0.203$), (b) vereinfachte Modellannahmen. Anm.: In der klassischen Keimbildungstheorie entspricht das Grenzflächenprofil eines Tropfens dem Profil bei „Bulk–Koexistenz“.

Nachdem in Kapitel 5.2 der Phasenübergang vom übersättigten Gas zu einem einzelnen Tropfen eindeutig lokalisiert und erklärt wurde, könnte man nun den Bereich rechts der Übergangsdichte ρ_t und links des Übergangs zur scheibenförmigen Konfiguration genauer analysieren. Dies ist jedoch problematisch, da der Gas–Tropfen–

Übergang erst bei sehr tiefen Temperaturen und großen Systemen deutlich hervortritt. Da sich ρ_t mit wachsender Systemgröße in Richtung der Koexistenzdichte verschiebt, könnte man versuchen wohldefinierte Cluster durch sehr große Systeme bei geringer Dichte zu simulieren. Neben den hiermit verbundenen technischen Schwierigkeiten (Gewichtung in n und U) ist dies jedoch auch grundsätzlich nicht praktikabel, da der Radius eines Tropfen bei Übergangsdichte in etwa mit $L^{3/4}$ wächst ($R \propto L\Delta\rho_t^{1/3}$ und $\Delta\rho_t \propto L^{-3/4}$). In ähnlicher Weise ist der Tropfen durch den Übergang zur scheibenförmigen Konfiguration durch eine mit L anwachsende Schranke nach oben begrenzt ($R_{\max} = L(0.266/(4\pi/3))^{1/3}$). Um experimentell interessante Cluster (im Bereich von 0 bis etwa 200 kT) zu simulieren, muss man also auf kleine Systeme zurückgreifen. Tropfen und übersättigtes Gas unterscheiden sich dann kaum noch bezüglich ihrer freien Energie, der Übergang wird unscharf und der Cluster besitzt keine wohldefinierte Grenzfläche mehr (vgl. hierzu das Dichteprofil in Abbildung 5.10 a).

Im Prinzip könnte man über ein Stillinger–Kriterium zu jedem Zeitpunkt den größten Cluster im System identifizieren (vgl. z.B. [92] oder Kapitel 5.2). Es ist jedoch auch möglich, einer Konfiguration bei gegebener Dichte mittels eines Modells einen Radius zuzuweisen. Aus der Wahrscheinlichkeitsverteilung erhält man über $F(n,L) = -kT\ln(P(n,L))$ die freie Energie als Funktion der Dichte und aus $R(\rho)$ schließlich $F(R)$. Wie bereits in Kapitel 5.1 vorgeschlagen sei der Cluster kugelförmig und mit Flüssigkeit gefüllt. Die Umgebung des Clusters bestehe aus homogenem Gas mit Koexistenzdichte und beide Phasen seien durch eine scharfe Grenzfläche getrennt (Abb. 5.10 b). Ferner gebe es keine Adsorptionseffekte an der Oberfläche des Cluster. Wie bereits in Kapitel 5.1 gezeigt, erhält man dann Gleichung (5.6). Wegen $\mu_g = \mu_f = 0$ und $p_g = p_f$ folgt Gleichung (5.7) und

$$\Delta F = F_{\text{Tropfen}} - F(n_g) = 4\pi\gamma R^2. \tag{5.13}$$

In der so genannten „Kapillaritätsapproximation“ entspricht γ gerade der Oberflächenspannung einer flachen Grenzfläche. Gleichung (5.13) versucht keineswegs die reale Situation vollständig zu beschreiben. So gilt z.B. $\mu_g = \mu_f = \partial F/\partial n > \mu_{\text{koex}}$. Aus der Minimierung der freien Energiedifferenz erhält man ferner einen Druckunterschied zwischen Flüssig– und Gasphase, den so genannten Laplace–Druck. Wie jedoch in Kapitel 5.1 gezeigt wurde, heben sich diese Zusatzterme in erster Ordnung gerade gegenseitig auf. Das hier verwendete Modell ähnelt also durchaus klassischer Nukleationstheorie. Mit steigendem chemischen Potential entfernt man sich von der Binodale und dringt immer weiter ins metastabile Gebiet vor. An der Koexistenzkurve divergiert die Keimbildungsbarriere und man erhält große Tropfen mit scharf definierter Grenzfläche. An der Spinodale verschwindet die Barriere und das Profil wird flach. Je näher man also an der Spinodale liegt, desto „weicher“ wird der Tropfen. Für Abbildung 5.10 a wurde eine Abschätzung dieses Abstandes vorgenommen. Bei $T = 0.85437\,\frac{\varepsilon}{k}$ erhält man $\rho_{\text{koex}} = 0.0512\,\frac{1}{\sigma^3}$ und $\rho_{\text{spin}} = 0.103\,\frac{1}{\sigma^3}$ (vgl. Kapitel 1.5). Berechnet man die Dichte eines übersättigten Gases, welches das gleiche chemische Potential wie der simulierte

Tropfen aufweist, so erhält man $\rho = 0.0663\ \frac{1}{\sigma^3}$. Dies liegt immer noch sehr nahe an der Koexistenzlinie. Der abgebildete Tropfen entspricht also gerade einem kritischen Tropfen bei dieser Dichte. Die einfache Beschreibung in Gleichung (5.13) liefert in der „Kapillaritätsapproximation“ eine obere Schranke für die freie Energie, die direkt berechnet werden kann. Würde man ein solches System in einer Simulation aufsetzen und äquilibrieren, wäre es bestrebt die Grenzfläche (wie in Abb. 5.10 a) aufzuweichen und so seine freie Energie zu minimieren. Im Folgenden soll untersucht werden, in welchem Bereich das einfache Modell eine gute Beschreibung liefert, welche Faktoren zu Abweichungen führen und in welcher Größenordnung diese liegen.

Hierzu betrachtet man zunächst ein Lennard–Jones–System bei $T = 0.85437\ \frac{\varepsilon}{k} = 0.85\ T_c$. Aufgrund der kleinen Systemgrößen und der hohen Temperatur können die Übergänge nicht wie in Kapitel 5.2 eindeutig identifiziert werden. Vorläufig soll daher der Bereich von $\Delta\rho = 0.15 - 0.3$ als Tropfenbereich definiert werden. Die rechte Grenze wird durch das Gleichsetzen der freien Energie von Tropfen und scheibenförmiger Konfiguration in dem verwendeten Modell motiviert. Aus $4\pi R^2\gamma = 2L^2\gamma$ folgt $\Delta\rho = 0.266$ (vgl. Kapitel 5.1). Abbildung 5.11 zeigt die freie Energiedifferenz für verschiedene Systemgrößen als Funktion des Radius. Der für die Tropfenbildung relevante Bereich ($\Delta\rho = 0.15 - 0.3$) ist hierbei für jede Systemgröße verstärkt hervorgehoben und beschreibt eine Einhüllende der Kurven. Hierdurch wird erneut deutlich, dass jede Systemgröße lediglich einen kleinen Bereich von Clustergrößen abdecken kann. Wie erwartet liegt die Modellvoraussage für die freie Energie oberhalb der Einhüllenden. Der qualitative Verlauf ist jedoch ähnlich. In Abbildung 5.12 wird die tatsächlich bestimmte freie Energiedifferenz ins Verhältnis zur theoretisch vorausgesagten gestellt. Man beobachtet, wie bereits in Abbildung 5.11 angedeutet, erhebliche Abweichungen bei kleinen und mittleren Tropfen, die mit zunehmender Tropfengröße abnehmen. Dies erscheint sinnvoll, da große Tropfen durch das Modell besser beschrieben werden als kleine. Verschiebt man die Theoriekurve um $0.56\ \sigma$ nach rechts, fällt diese fast exakt auf die Einhüllende. Dieser um einen konstanten Wert verschobene „effektive“ Radius kompensiert vor allem die Aufweichung der Grenzfläche. Nimmt man z.B. in Gleichung (5.6) an, dass im Inneren des Tropfens die Flüssigkeitsdichte wie in Abbildung 5.10 geringer als bei Koexistenz ausfällt und das umgebende Gas entsprechend dichter wird, so erhält man in erster Ordnung einen konstanten, positiven „offset“ im Radius. Man könnte die Abweichung im Prinzip auch der Grenzflächenspannung zuschlagen und die Theoriekurve in negative y–Richtung verschieben. Eine krümmungsbedingte Erniedrigung um einen Term proportional zu $\frac{1}{R^2}$ führt in diesem Fall jedoch zu einer signifikant schlechteren Übereinstimmung. Eine Kombination beider Korrekturen ist ebenfalls denkbar. Vergleichbare Untersuchungen wurden auch bei einer tieferen Temperatur durchgeführt. Abbildung 5.13 zeigt zudem die Einhüllende für Tropfen bei $T = 0.68\ \frac{\varepsilon}{k}$ (Temperatur der Simulationen in Kapitel 5.2). Aufgrund der höheren Grenzflächenspannung erhält man erwartungsgemäß schon für kleine Tropfen verhältnismäßig große Differenzen in der freien Energie. Die Abweichungen von der theore-

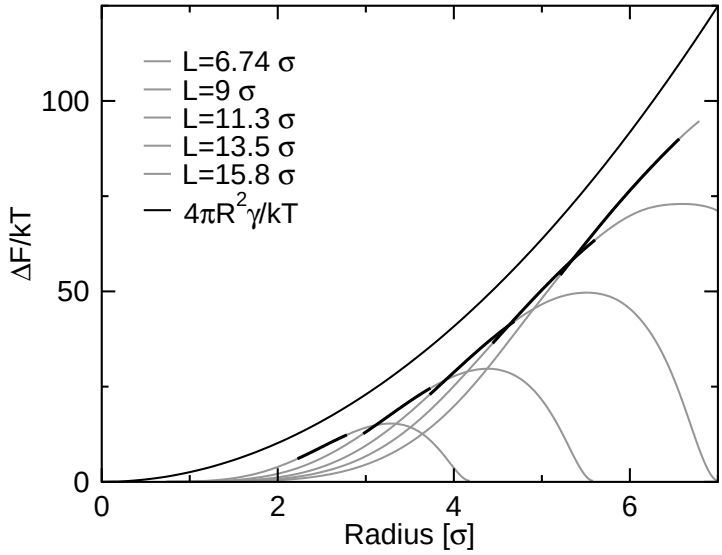

Abbildung 5.11: Freie Energie als Funktion der Tropfenradien für verschiedene Systemgrößen (von links nach rechts: $L = 6.74\ \sigma$, $L = 9\ \sigma$, $L = 11.3\ \sigma$, $L = 13.5\ \sigma$, $L = 15.8\ \sigma$, $\gamma = 0.173\ \frac{\varepsilon}{\sigma^2}$; $\mu = -2.96676$). Die Einhüllende entspricht qualitativ der Voraussage des theoretischen Modells.

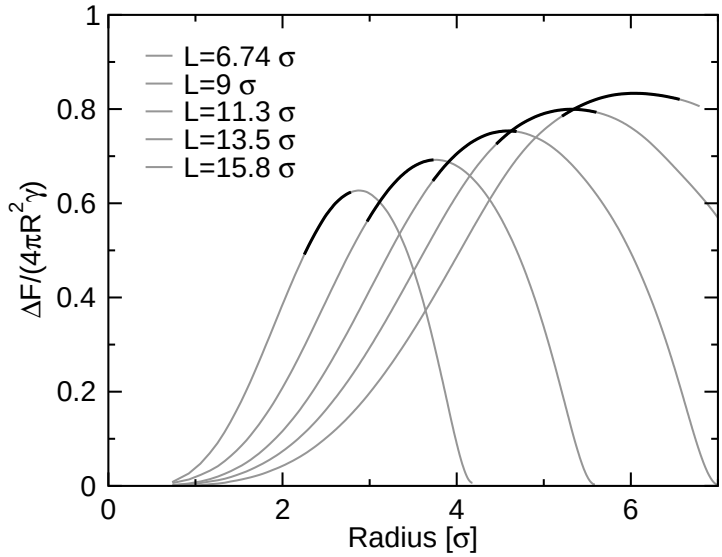

Abbildung 5.12: Quotient aus freier Energie und Modellvoraussage als Funktion der Tropfenradien für verschiedene Systemgrößen (von links nach rechts: $L = 6.74\ \sigma$, $L = 9\ \sigma$, $L = 11.3\ \sigma$, $L = 13.5\ \sigma$, $L = 15.8\ \sigma$). Die Abweichungen nehmen mit zunehmender Tropfengröße ab.

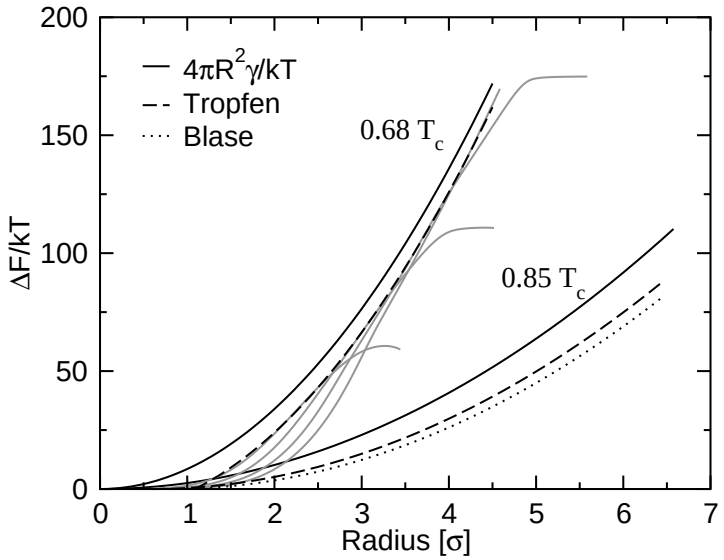

Abbildung 5.13: Freie Energie als Funktion der Tropfenradien für ein Lennard–Jones–System bei $T = 0.68\ \frac{\varepsilon}{k}$ ($\mu = -3.23357$) und $T = 0.85437\ \frac{\varepsilon}{k}$. Für $T = 0.85437\ \frac{\varepsilon}{k}$ wird auch die Einhüllende der Blasen gezeigt.

tischen Vorhersage fallen jedoch geringer aus, da die Fluktuationen mit zunehmendem Abstand vom kritischen Punkt abnehmen. Auch hier kann die Theoriekurve durch eine entsprechende Korrektur auf die Einhüllende abgebildet werden. Diesmal liefert jedoch die Erniedrigung der Grenzflächenspannung eine signifikant bessere Übereinstimmung (gestrichelte Linie in Abb. 5.13). Es scheint als würde der bei der höheren Temperatur dominierende „Aufweichungseffekt“ hier entfallen und lediglich die geringere Korrektur der krümmungsabhängigen Grenzflächenspannung ins Gewicht fallen.

Die Betrachtungen für Tropfen lassen sich auch auf den Blasenbildungsprozess übertragen. Blasen sind in dem hier vorgestellten Modell kugelförmig und weisen eine scharfe Grenzfläche auf. Im Inneren befindet sich Gas bei Koexistenzdichte, die Umgebung besteht aus homogener Flüssigkeit. Analoge Betrachtungen wie beim Tropfen führen zu

$$R = \sqrt[3]{\frac{V}{\frac{4}{3}\pi}\,(1-\Delta\rho)}. \tag{5.14}$$

Die freie Energie entspricht Gleichung (5.13). Der für die Blasenbildung relevante Bereich wird analog zur Tropfenbildung auf $0.7 - 0.85\ \Delta\rho$ festgelegt. Bereits getroffenen Aussagen über Tropfen lassen sich auch auf die Blasenbildung übertragen. Die Einhüllende der $F(R)$–Kurve für $T = 0.85437\ \frac{\varepsilon}{k}$ (gepunktete Linie in Abbildung 5.13) entspricht qualitativ der theoretischen Vorhersage und liegt abermals unterhalb der Theoriekurve. Im Unterschied zur Tropfenbildung sind die Abweichungen jedoch et-

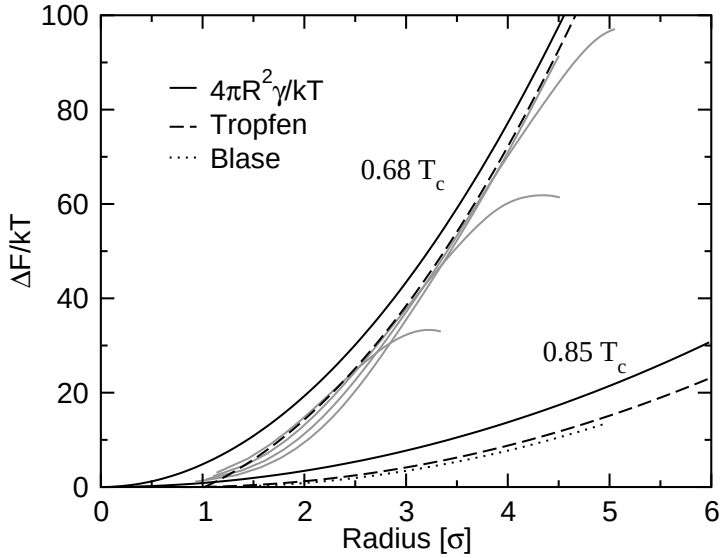

Abbildung 5.14: Freie Energie als Funktion der Tropfenradien für ein Fünfmer–System bei $T = 1.16\ \frac{\varepsilon}{k}$ $(\mu = 36.3113)$ und $T = 1.55\ \frac{\varepsilon}{k}$ $(\mu = 27.0235)$. Für $T = 1.55\ \frac{\varepsilon}{k}$ wird auch die Einhüllende der Blasen gezeigt.

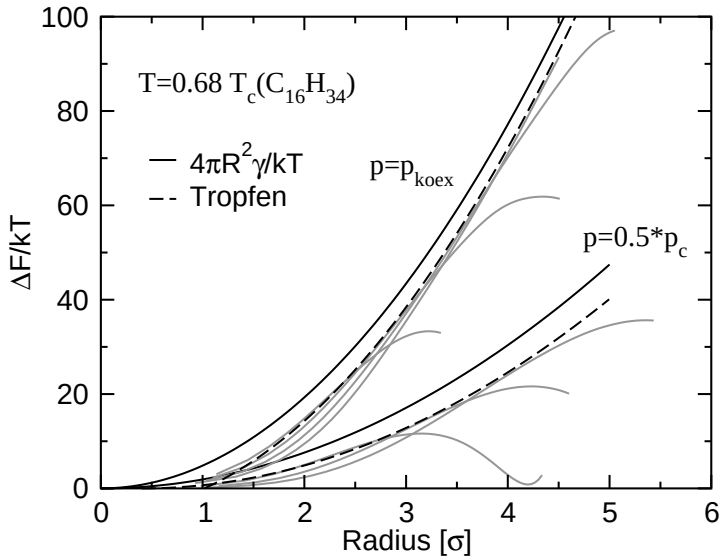

Abbildung 5.15: Freie Energie als Funktion der Tropfenradien für ein Polymer–Lösungsmittel–Mischsystem ($\xi = 0.886$) bei $T = 1.16\ \frac{\varepsilon_5}{k}$, einmal bei Koexistenz des Fünfmers $p \approx 0$ und einmal bei der Hälfte des kritischen Drucks der Isotherme ($\mu_1 = -2.84,\ \mu_5 = 36.6865$). Die Isothermen finden sich in Abb. 3.13 oder 5.16.

was größer. Die Theoriekurve kann durch eine Verschiebung um $0.8\,\sigma$ nach rechts auf die Einhüllende abgebildet werden. Bei Tropfen– und Blasenbildung handelt es sich folglich nicht um vollständig symmetrische Prozesse (vgl. Abb. 5.1 b).

Bei Polymeren (Abb. 5.14) zeigen sich von der Tendenz her die gleichen Abhängigkeiten wie beim Lennard–Jones–System. Die Differenzen in der freien Energie sind jedoch bei vergleichbaren Clusterradien etwas niedriger. Abbildung 5.15 zeigt für einen isothermen Schnitt bei $T = 1.16\ \frac{\varepsilon}{k} \approx 0.68\ T_c(\text{Polymer})$ die Einhüllenden der $F(R)$–Kurven, einmal für $p \approx 0.5\ p_{\text{koex}}$ und einmal für $p = p_{\text{koex}} \approx 0$ (reines Polymer – vergleiche mit Abb. 5.14). Aufgrund der höheren Grenzflächenspannung (bzw. des größeren Abstandes zum kritischen Punkt des Mischsystems) ist für das reine Polymer ΔF größer und die Übereinstimmung mit dem Modell besser.

Zusammenfassend können folgende Aussagen getroffen werden. In einer Simulationsbox gegebener Größe können lediglich Cluster eines bestimmten Größenbereiches simuliert werden. Dieser wird durch zwei Phasenübergänge erster Ordnung (Gas–Tropfen, Tropfen–Scheibe) begrenzt. Kleine Cluster können nur durch kleine Boxen simuliert werden. In diesem Fall sind die Phasenübergänge jedoch nicht mehr scharf und die Grenzfläche der Tropfen stark aufgeweicht. Weist man zur Bestimmung eines Radius dem Cluster und der umgebenden Mutterphase die jeweiligen Koexistenzdichten zu, so bildet die freie Energie der sphärischen Grenzfläche eine obere Abschätzung für F. Abweichungen vom Modell werden mit zunehmender Clustergröße und wachsendem Abstand vom kritischen Punkt geringer. Die Beobachtungen gelten auch für Blasen– und Tropfenbildung komplexerer Systeme. Zwar können die Abweichungen durch eine Einführung einer krümmungsabhängigen Grenzflächenspannung oder einer Vergrößerung des Clusterradius im Rahmen des Modells erklärt werden, dennoch wäre eine weiterreichende Analyse wünschenswert. Ein denkbarer Ansatz wäre die direkte Einbeziehung der „Aufweichungseffekte“.

5.4 Visualisierung der frühen Stadien von Phasenseparation

Zum Abschluss soll ein qualitativer Ausblick auf die Kinetik der Entmischung im Hexadekan–CO_2–System gegeben werden. Die folgenden Betrachtungen beziehen sich auf einen isothermen Schnitt durch das Mischsystem–Phasendiagramm bei 486 K und $\xi = 0.886$. Abbildung 5.16 zeigt den Verlauf der Koexistenzkurve (Simulation und TPT1) und der Spinodalen (TPT1). Zusätzlich wurden die Endpositionen zweier Sprünge in die metastabile und die instabile Region des Zwei–Phasen–Gebiets markiert.

Zur Visualisierung des Blasenbildungsprozesses wurde die Untersättigung des Sys-

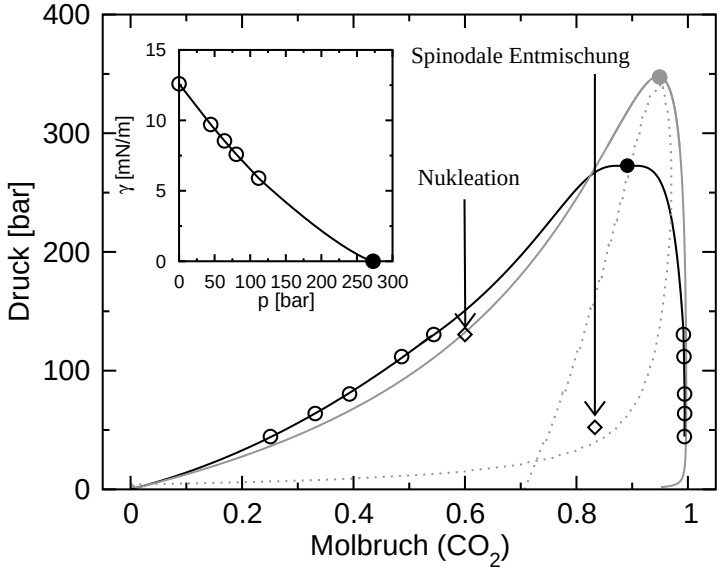

Abbildung 5.16: Isothermer Schnitt durch das Hexadekan–CO_2–Mischsystem–Phasendiagramm bei $T = 486\ K$ und $\xi = 0.886$ (vgl. mit Abb. 3.7). Die simulierte Kurve wurde bereits in Abbildung 3.13 gezeigt. Die durchgezogene graue Linie entspricht der TPT1–Berechnung der Koexistenzkurve und die gepunktete graue Linie der entsprechenden Spinodalen. Das kleine Bild stammt aus Abbildung 3.14. Die beiden Pfeile markieren jeweils die Endpositionen eines „Quenches“ aus dem Ein–Phasen–Gebiet in das Entmischungsgebiet (Abbildung 5.17 und 5.20).

tems durch eine geeignete Wahl der chemischen Potentiale beider Teilchensorten festgelegt. Als Startkonfiguration diente ein homogener, bei hoher Temperatur äquilibrierter Zustand. Die eigentliche Entmischung wurde großkanonisch simuliert. Da die Anzahl der Teilchen in der Zelle fluktuiert und sich die Teilchen nicht gemäß einer realistischen Dynamik bewegen, liefert die Simulation keine Informationen über die Zeitskala des Prozesses. Da die Phasenseparationskinetik in der Nähe der Binodalen jedoch in erster Linie von der Höhe der Nukleationsbarriere bestimmt wird, erwartet man einen ähnlichen Relaxationspfad wie bei einer realistischen Dynamik. Eine entsprechende kanonische Molekulardynamik–Simulation müsste jedoch mit weitaus größere Boxen durchgeführt werden, um die konstante Untersättigung in der Umgebung der Blase auch während des Entmischungsvorgangs zu gewährleisten. Abbildung 5.17 a–f zeigt die zeitliche Entwicklung des Blasenbildungsprozesses. Aus Gründen der Übersichtlichkeit wurde lediglich ein Schnitt der Größe $2\,\sigma$ durch die Box dargestellt. Zu Beginn fluktuiert das System um ein metastabiles Minimum in der freien Energie. Es bilden sich kleine Blasen in der Hexadekan–Matrix (rot), die sichtbar werden wenn der blaue Hintergrund durch den Schnitt hervortritt (Abb. 5.17 b). Die Hohlräume sind jedoch zu klein, um die Nukleationsbarriere zu überwinden und zerfallen wieder in den ur-

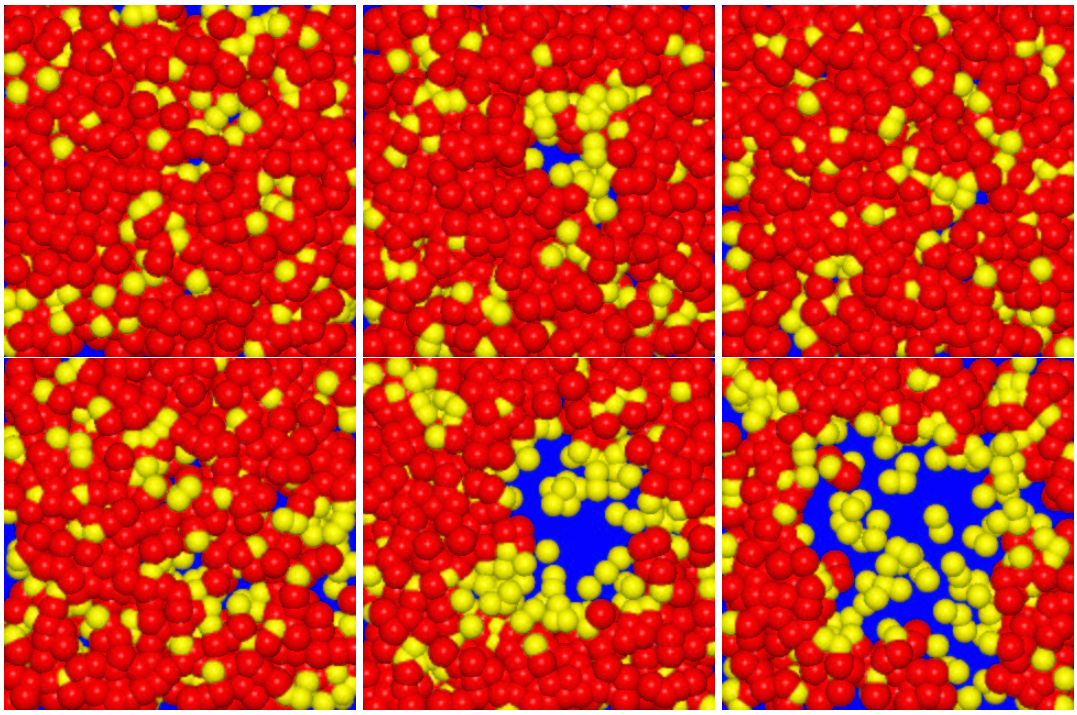

Abbildung 5.17: Zeitliche Entwicklung der Blasenbildung nach einem Sprung ins metastabile Gebiet ($T = 486.2\ K$, $x = 0.60$, $p \approx 130\ bar$ und $L = 22.5\ \sigma_5$ – vgl. mit Abb. 5.16). Zur besseren Übersicht wird lediglich ein Schnitt der Größe $2\ \sigma_5$ dargestellt. Rote Kugeln entsprechen vergröberten Hexadekan–Monomeren und gelbe Kugeln den CO_2–Lösungsmittelteilchen. Der durchscheinende Hintergrund ist blau. Die Teilchen wurden etwas vergrößert: der Radius einer Hexadekan–Kugel beträgt σ_5, der Radius einer CO_2–Kugel $\sigma_1 = 0.816\ \sigma_5$. Eine Animation des Entmischungsvorgangs findet sich auf der beigelegten CD.

sprünglichen homogenen Zustand (Abb. 5.17 c). Nach einer gewissen Zeit gelingt es einer Blase den kritischen Radius zu überschreiten und so die Nukleationsbarriere zu überwinden (Abb. 5.17 e). Von nun an wächst die Blase weiter (Abb. 5.17 f) bis das System komplett separiert ist. Eine Animation des Entmischungsvorgangs findet sich auf der beigelegten CD. Abbildung 5.17 f deutet eine Anlagerung von CO_2–Teilchen an der Grenzfläche an. Eine solche Benetzung würde den Übergang zwischen der dichten Polymerphase und dem Gas aufweichen, die Grenzflächenspannung und somit die Nukleationsbarriere erniedrigen. Die hier gezeigte Visualisierung ist keineswegs die erste ihrer Art. So enthält z.B. bereits [96] eine Reihe von Schnappschüssen, die den Keimbildungsprozess in einem zweidimensionalen Gittergas illustrieren. Das hier diskutierte System ist jedoch weitaus komplexer und zudem von industrieller Relevanz.

Die Benetzung wurde auch an einer flachen Grenzfläche untersucht (Abb. 5.18). Hierzu wurden zunächst verhältnismäßig große Blöcke in der flüssigen Phase erzeugt

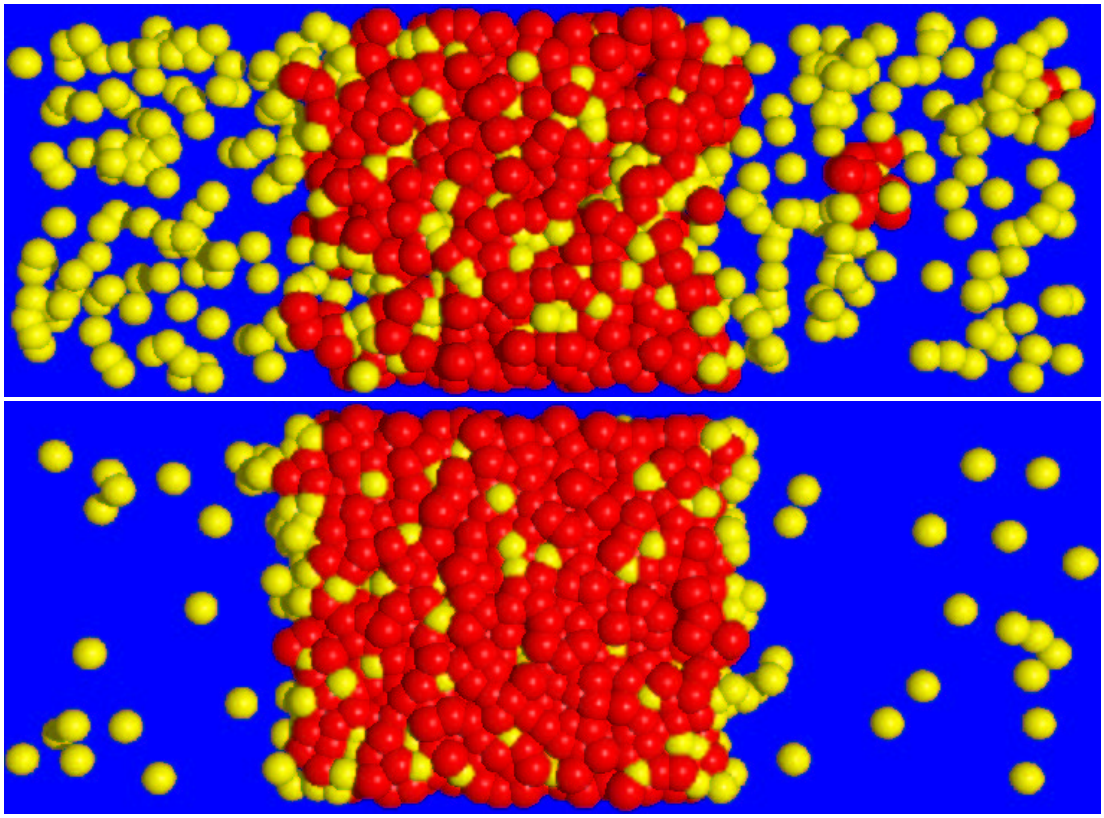

Abbildung 5.18: Schnappschüsse einer flachen Grenzfläche in der Zwei–Phasen–Region bei $T = 486.2\ K$ und $T = 243\ K$ ($\xi = 0.886$). Die Boxgröße beträgt $18\ \sigma_5 \times 18\ \sigma_5 \times 54\ \sigma_5$, gezeigt wird jedoch lediglich ein Schnitt von $2\ \sigma_5$. Die Farbkodierung entspricht der von Abbildung 5.17. Für $T = 243\ K$ sind die Grenzflächen benetzt.

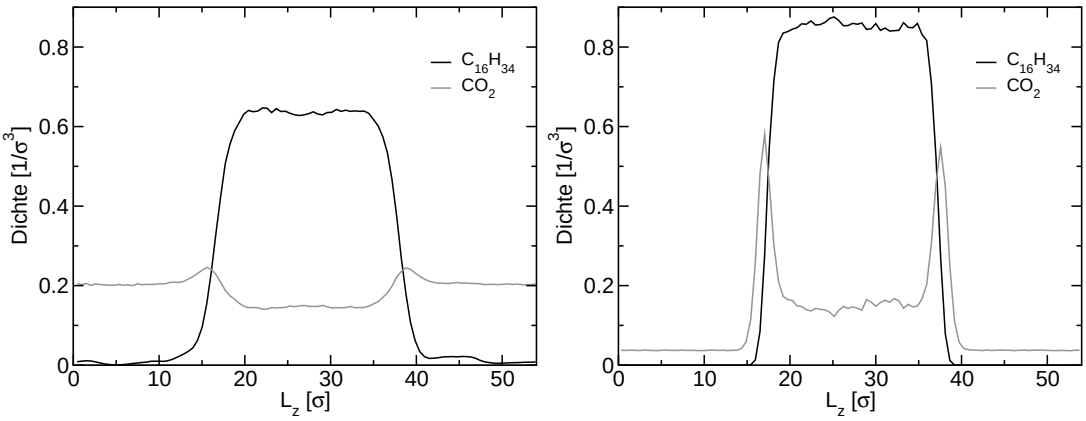

Abbildung 5.19: Dichteprofile für die Konfigurationen in Abbildung 5.18.

und anschließend in einem langgestreckten System kanonisch über einen langen Zeitraum äquilibriert. Die Dichte der CO_2–Teilchen durfte hingegen fluktuieren. Diese Vorgehensweise ermöglicht die Untersuchung von großen Mischsystem–Konfigurationen auch bei sehr tiefen Temperaturen (Abbildung 5.18 b). Abbildung 5.18 a entspricht in Temperatur und Koexistenzdruck Abbildung 5.17 a–f. Die sich andeutende Benetzung der Grenzfläche kann hier visuell nicht identifiziert werden. Eine Betrachtung der Grenzflächenprofile (Abbildung 5.18 a) zeigt jedoch eindeutig einen Benetzungseffekt, der zu einer Reduzierung der Grenzflächenspannung führt. Sowohl die Flüssig–, als auch die Gasphase weisen Koexistenzzusammensetzung und –dichte auf. Untersucht man das gleiche System bei einer sehr viel tieferen Temperatur (knapp unterhalb der Koexistenzlinie von reinem CO_2, vgl. Abbildung 3.7), ist die Benetzung mit bloßem Auge erkennbar. Diese Verstärkung ist auf die Nähe zur Drei–Phasen–Linie zurückzuführen.

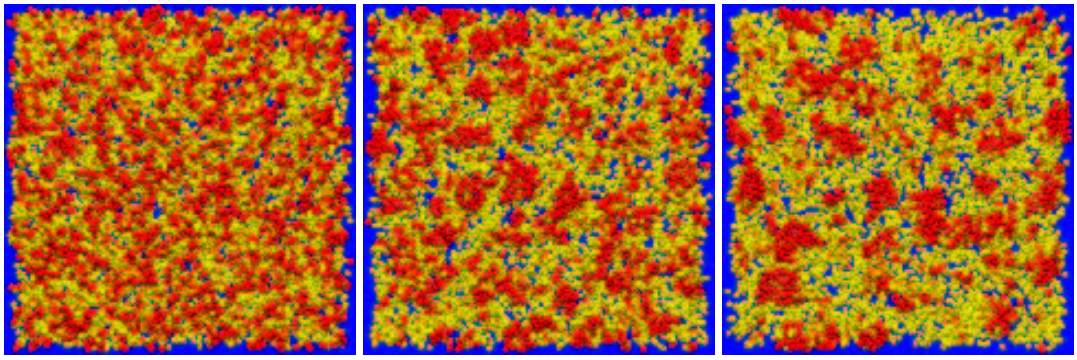

Abbildung 5.20: Zeitliche Entwicklung der spinodalen Entmischung nach einem Sprung ins instabile Gebiet ($T = 486\ K$, $x = 0.833$ und $p \approx 52$ bar, vergleiche mit Abbildung 5.16). (a) Startkonfiguration, (b) nach 125000 und (c) nach 500000 versuchten lokalen Verrückungen (0–0.3 σ_5) pro Teilchen. Die Boxgröße beträgt $91.21\ \sigma_5 \times 91.21\ \sigma_5 \times 17.53\ \sigma_5$ wobei lediglich ein Schnitt von 10 σ_5 gezeigt wird. Die Farbkodierung entspricht der von Abbildung 5.17.

Abbildung 5.20 zeigt zum Abschluss den zeitlichen Ablauf von spinodaler Entmischung nach einem Sprung in die von den analytischen Rechnungen bestimmte instabile Region des Zwei–Phasen–Gebiets (vgl. Abbildung 5.16). Wie zuvor wurde eine homogene Konfiguration mit festgelegter Zusammensetzung und Dichte bei hoher Temperatur äquilibriert und im Anschluss instantan abgekühlt. Der Entmischungsvorgang wurde kanonisch durch lokale Schritte simuliert. Im Gegensatz zur Keimbildung treten die relativen Fluktuationen der gelben Lösungsmittelteilchen verstärkt hervor und bilden ein irreguläres, perkolierendes Netzwerk (Abbildung 5.20 b; vgl. mit [97], [98]). Im Vergleich zur Nukleation verläuft der Entmischungsvorgang schneller, da keine Nukleationsbarriere überwunden werden muss.

Zusammenfassung und Ausblick

In dieser Arbeit wurde das Phasen– und Keimbildungsverhalten von Polymer–Lösungsmittel–Systemen untersucht. Hierzu musste zunächst ein bereits vorhandenes Monte–Carlo–Programm auf Mischungen erweitert werden. Modifizierte Versionen wurden seitdem zur Berechnung des Adsorptionsverhaltens von Copolymerlösungen [99] und der Untersuchung von Benetzungsphänomenen [100] eingesetzt.

Koexistierende Phasen sind im großkanonischen Ensemble in der Regel durch eine Barriere in der freien Energie getrennt. Multikanonische Simulationen überwinden diese durch eine Gewichtung des Hamiltonians in der Monte–Carlo–Abfrage. Hierzu benötigt man jedoch einen Schätzwert für die Wahrscheinlichkeitsverteilung, der in der Regel durch eine mühsame Abfolge von Simulationen und Extrapolationen erzeugt wird. *Wang* und *Landau* schlugen vor nicht allzu langer Zeit ein selbstregulierendes Verfahren für Gittermodelle vor, welches die Gewichtsfunktion während der Simulation erzeugt [41], [46]. Dieser Ansatz wurde in der Arbeit auf Polymermischungen im Kontinuum übertragen und an die speziellen Anforderungen des (μ, V, T)–Ensembles angepasst [101]. Die Methode liefert brauchbare Gewichtsfunktionen und kann prinzipiell an jeder Stelle des Phasendiagramms eingesetzt werden. Eine exakte Bestimmung des Fehlers ist jedoch aufgrund der ständigen Änderung des Hamiltonians nicht möglich. Aufbauend auf diesen Erfahrungen wurde ein neues effizientes Verfahren entwickelt. Ähnlich der Umbrella–Sampling–Technik wird das zu untersuchende Intervall in kleine Abschnitte unterteilt, die nacheinander simuliert werden. Dies ermöglicht eine Extrapolation der Wahrscheinlichkeitsverteilung in das darauf folgende Fenster und somit die Erzeugung einer Gewichtsfunktion. Der mühsame Prozess des Zusammensetzens der verschiedenen Abschnitte wurde automatisiert und in die Simulation integriert. Der Algorithmus ermöglicht eine direkte Bestimmung der Wahrscheinlichkeitsverteilung ohne vorherige Kenntnis der Gewichtsfunktion. Die Detailed–Balance–Bedingung wird nicht verletzt und der Fehler kann exakt ermittelt werden. Dies wurde in einer ausführlichen Analyse nachgewiesen. Hierbei zeigte sich auch, dass entgegen der herrschenden Meinung der Gesamtfehler der Simulation unabhängig von den gewählten Fenstergrößen ist. Das Verfahren ist einfach zu parallelisieren und kann auf Mischungen angewendet werden [101]. Eine Übertragung auf eine Vielzahl von Systemen, beginnend mit einfachen Gittermodellen bis hin zu Kontinuumsmodellen

beliebiger Komplexität, ist mit geringem Aufwand zu realisieren. Mögliche Fragestellungen, die sich mit Hilfe der neuen Methode bearbeiten lassen, umfassen das Studium von Phasenübergängen, der Nukleation und vieles mehr. Innerhalb der Arbeitsgruppe wurde der Algorithmus bereits bei semi–großkanonischen Simulationen von binären Lennard–Jones–Mischungen [102] und großkanonischen Simulationen von Polymermischsystemen an Wänden [100] eingesetzt.

Das Phasenverhalten von Polymer–Lösungsmittel–Mischsystemen wurde anhand des Referenzsystems Hexadekan–CO_2 untersucht [103], [104], [105]. Zunächst musste ein passendes Modell gefunden werden. Hierzu wurde CO_2 auf eine einzelne LJ–Kugel abgebildet und Hexadekan auf eine Kette von fünf LJ–Teilchen. Zur Bestimmung der Parameter im Potential setzte man die kritischen Temperaturen von Simulation und Experiment gleich. Der vergröberte Ansatz basiert auf der Annahme, dass statische Gleichgewichtseigenschaften in erster Linie durch intermolekulare Wechselwirkungen und somit durch die grobe Struktur des Moleküls bestimmt werden. Intramolekulare Wechselwirkungen, die bei realistischeren United–Atom–Modellen berücksichtigt werden, spielen dagegen nur eine untergeordnete Rolle. Die Vermutung wurde durch die Simulationsergebnisse für Hexadekan bestätigt. Die Übereinstimmung von Phasendiagrammen und Grenzflächenspannungen mit den entsprechenden experimentellen Daten ist überraschend gut und reicht an die Qualität von weitaus aufwendigeren, atomistischen Modellen heran. In naher Zukunft sollen weitere, systematische Vergleiche für Alkane unterschiedlicher Kettenlängen folgen [106]. In diesem Zusammenhang wäre es interessant, auch Größen wie den End–zu–End–Abstand in die Betrachtung mit einzubeziehen. Für reines CO_2 fällt die Übereinstimmung erwartungsgemäß etwas schlechter aus. Die Ergebnisse sind jedoch durchaus noch akzeptabel. Die Erweiterung des Modells auf Mischsysteme erfolgt durch die Anwendung der modifizierten Lorentz–Berthelot–Regel. Diese weist einen einzigen freien Parameter ξ auf, der bisher nicht berücksichtigte Effekte (wie die Partialladungen bei CO_2) kompensiert. Hexadekan–CO_2 wird in der Klassifizierung von *Konynenburg* und *Scott* [62], [63] einem Typ–III–Phasendiagramm zugeordnet. Für unterschiedliche Werte von ξ wurden vermutlich zum ersten Mal vollständige Phasendiagramme inklusive kritischer Linien erzeugt. Durch eine Absenkung von ξ und der einhergehenden Reduzierung der CO_2–Hexadekan–Wechselwirkung konnte das System von einem Typ–I–Phasendiagramm in ein Typ–III–Phasendiagramm überführt werden. In diesem Zusammenhang wurde auch eine Drei–Phasen–Koexistenz simuliert, die den Ausgangspunkt für weitere Betrachtungen zur Benetzung von Grenzflächen im Mischsystem bildet [100]. Da die experimentellen Daten zur kritischen Linie stark voneinander abweichen und der richtige Phasendiagrammtyp bereits erreicht wurde, konnte auf eine genaue Einstellung des Wechselwirkungsparameters verzichtet werden.

Der Vergleich der Simulationen mit numerischen TPT1–Störungsrechnungen von *L.G. MacDowell* [50] lieferte eine qualitative Übereinstimmung und ermöglichte eine nähere Charakterisierung des Phasenverhaltens. Bei den reinen Systemen und dem

Mischsystem wurden kritische Punkte aufgrund des „mean field“–artigen Charakters der Theorie überschätzt. Die Koexistenzdichten stimmen für das reine Lennard–Jones–System besser überein als für das Fünfmer–System. Der Übergang von Typ–I– zu Typ–III–Verhalten des Mischsystems erfolgt bei TPT1 bei den gleichen Werten für ξ. Mit steigender Kettenlänge verschiebt sich die kritische Linie zu immer höheren Drücken. Isotherme Schnitte durch das Phasendiagramm stimmen ebenfalls qualitativ mit den Simulationen überein. Abweichungen lassen sich hier in erster Linie durch die Differenzen in den kritischen Punkten erklären. Der Vergleich mit den Simulationen ermöglichte auch eine stufenweise Überprüfung der in der Theorie angewendeten Näherungen. Simulationen von binären Lennard–Jones–Mischungen legen nahe, dass Abweichungen beim Mischsystem in erster Linie auf die Kettenbeiträge in der Zustandsgleichung zurückzuführen sind. Da es sich bei der Spinodalen im eigentlichen Sinn um ein Mean–Field–Konzept handelt, konnte eine Methode zur Abschätzung der Grenzen des instabilen Gebiets bei Simulationen überprüft werden [105]. Diese wurden durch lineare Extrapolation des Kehrwertes der Teilchenfluktationen aus dem Ein–Phasen–Gebiet bestimmt. Hierbei unterstellt man, dass Fluktuationen an der Spinodalen divergieren und lediglich der lineare Teil in der Virialentwicklung des Ausdrucks von Bedeutung ist. Letzteres bestätigt sich im Vergleich mit TPT1 für die Gasseite. An der Flüssigkeitsseite des Phasendiagramms müssen Terme höherer Ordnung berücksichtigt werden. Die mittels Simulation bestimmte Spinodale liegt zwischen der Koexistenzkurve und der Mean–Field–Spinodalen.

Computersimulationen von Keimbildungsphänomenen sind immer durch endliche Boxgrößen limitiert. Aus diesem Grund muss dem Studium von Finite–Size–Effekten eine zentrale Bedeutung eingeräumt werden. In endlichen Volumina lässt sich der Übergang vom übersättigten Gas zum einzelnen Tropfen durch einen Phasenübergang erster Ordnung beschreiben [37]. Dieser tritt nur bei verhältnismäßig großen Systemen deutlich hervor und strebt mit wachsendem Boxenvolumen gegen die Koexistenzdichte. Nach einer kurzen, phänomenologischen Motivation wurde der Übergang zum ersten Mal für ein Lennard–Jones–System explizit nachgewiesen [93], [107]. Dies konnte nur durch einen gezielten Einsatz der in dieser Arbeit entwickelten Simulationsmethoden erreicht werden. Neben eher indirekten Anzeichen wie dem Abknicken der freien Energie bzw. dem Sprung im chemischen Potential, konnten bei der Übergangsdichte bimodale Verteilungen in der Größe des größten Clusters im System und anderen, hiermit in Beziehung stehenden Parametern nachgewiesen werden. Die Abhängigkeit des Übergangs von der Systemgröße wurde ebenfalls demonstriert. Hierbei zeigte sich, dass der von der Theorie vorhergesagte Exponent bei simulierbaren Systemgrößen nicht erreicht wird. Der Phasenübergang ist zudem verantwortlich für die Verlangsamung bzw. Blockade von einfach gewichteten multikanonischen Simulationen großer Systeme. Aufbauend auf diesen Betrachtungen wurde die freie Energie von Tropfen und Blasen kleiner und mittlerer Größe bestimmt. Bei gegebenem Boxvolumen wird die Clustergröße durch die beiden Phasenübergänge vom Gas zum Tropfen und vom

Tropfen zu einer scheibenförmigen Konfiguration begrenzt. Kleine Cluster können nur durch kleine Boxen simuliert werden. In diesem Bereich sind die Phasenübergänge unscharf und die Grenzfläche der Tropfen stark aufgeweicht. Weist man zur Bestimmung eines Radius dem Cluster und der umgebenden Mutterphase jeweils deren Koexistenzdichte zu, so bildet die freie Energie der sphärischen Grenzfläche eine obere Abschätzung für F. Abweichungen zu diesem Verlauf werden mit zunehmender Clustergröße und wachsendem Abstand vom kritischen Punkt geringer. Die Beobachtungen lassen sich auch auf komplexere Systeme, wie Polymer–Lösungsmittel–Mischungen übertragen. Zwar können die Abweichungen durch eine Einführung einer krümmungsabhängigen Grenzflächenspannung oder einer Vergrößerung des Clusterradius eliminiert werden – eine weiterreichende Analyse wäre dennoch wünschenswert [49]. Die Arbeit schließt mit einem kurzen Ausblick auf die Dynamik der Entmischung. Nach einem Sprung in die metastabile Region bilden sich durch Fluktuationen kleine Blasen, die wieder zerfallen. Nach einer gewissen Zeit wächst eine Blase über eine kritische Größe hinaus und das System separiert. Hierbei deutet sich eine Benetzung der Grenzfläche durch CO_2 an, die vermutlich zu einer Absenkung der Grenzflächenspannung und der Nukleationsbarriere führt. Die Benetzung tritt auch bei Konfigurationen mit flacher Grenzfläche auf – besonders ausgeprägt in der Umgebung der Drei–Phasen–Koexistenzlinie. Bisherige Betrachtungen sind jedoch von qualitativer Natur und sollen noch ausgeweitet werden.

Anhang A

Simulationsdaten

T	μ_1	L	ρ_g	ρ_f	γ
0.68	-3.23357	6.74^1	0.01001	0.7706	(0.450645)
0.68	-3.23357	6.74^2	-	-	0.4594
0.78*	-3.06437	11.3^1	0.0273726	0.707583	-
0.82*	-3.00882	11.3^1	0.0389417	0.676734	-
0.85437	-2.96676	11.3^1	0.05059	0.6508	(0.166)
0.85437	-2.9669	9^2	0.05007	0.6495	0.173
0.9*	-2.91711	11.3^1	0.0755217	0.606476	-
0.9	-2.91707	9^2	0.0762538	0.605939	0.10666
0.93*	-2.88805	11.3^1	0.0998297	0.568708	-
0.97	-2.85344	9^2	(0.157049)	(0.497268)	0.021806
0.97	-2.85329	18^2	0.151134	0.503462	0.0218796
0.97*	-2.85458	11.3^1	(0.152472)	(0.501249)	-
0.985	-2.84161	18^2	0.191614	0.458941	0.00764871
0.999	-2.83138	9^2	(0.232861)	(0.412791)	0.00254003
0.999	-2.83085	18^2	0.258453	0.378379	0.00114702
0.999*	-2.83206	11.3^1	(0.221545)	(0.420112)	-

Tabelle A.1: Gas–flüssig–Koexistenz der Monomere. Alle Angaben in Lennard–Jones–Einheiten. Eingeklammerte Werte wurden in den entsprechenden Graphen nicht berücksichtigt. * Zu Beginn der Doktorarbeit vorhandene Simulationen, [1] kubische Simulationsbox ($V = L^3$), [2] gestreckte Box ($V = 2L^3$). Für verschiedene Systemgrößen und –formen können kleine Variationen im chemischen Potential auftreten, insbesondere in der Nähe von T_c.

T	μ_5	L	ρ_g	ρ_f	γ
1	39.7085	6.74^1	1.88893e-06	0.164233	(0.591288)
1.16	36.3113	6.74^2	0.000198571	0.153326	0.445268
1.2	35.4393	6.74^1	0.000323683	0.150034	(0.37027)
1.3	33.1619	9^2	0.000900816	0.14231	0.308422
1.35	31.9778	6.74^1	0.00139297	0.13821	(0.235352)
1.38	31.257	9^2	0.00175906	0.135663	0.241483
1.42	30.2838	9^2	0.00252922	0.131713	0.206018
1.46	29.2949	9^2	0.00344003	0.127803	0.174706
1.5	28.2969	6.74^1	0.00480345	0.123472	(0.120625)
1.5	28.2956	9^2	0.00465485	0.123621	0.144456
1.54	27.2816	9^2	0.00632514	0.118814	0.113209
1.55	27.0235	9^1	0.00681641	0.117374	(0.0884144)
1.55	27.0227	9^2	0.00679553	0.117443	0.105791
1.59	25.9949	9^2	0.00933395	0.111432	0.0766854
1.64	24.687	9^2	0.01475	0.10238	0.0451823
1.65	24.4299	6.74^1	(0.0160208)	(0.101559)	(0.0422495)
1.67	23.8928	9^2	0.0204713	0.0942037	(0.0278729)
1.725	22.4256	13.5^1	(0.03311)	(0.0779605)	(3.20664e-05)

Tabelle A.2: Gas–flüssig–Koexistenz des Fünfmers. Alle Angaben in Lennard–Jones–Einheiten. ρ_g entspricht der Polymer–Teilchenzahldichte. Eingeklammerte Werte wurden in den entsprechenden Graphen nicht berücksichtigt. [1] Kubische Simulationsbox ($V = L^3$), [2] gestreckte Box ($V = 2L^3$).

T	μ_1	μ_5	L	ρ_c	x_c (CO_2)	p_c
0.726	-2.0557	40	9	0.595453	0.999421	0.117725
0.75	-2.05047	41.21	6.74	0.630307	0.989113	0.1367
0.8	-2.03728	41.06	6.74	0.616526	0.97323	0.178644
0.9	-1.99	40.0246	6.74	0.652623	0.949675	0.282355
0.93	-1.995	39.628	9	0.629544	0.949893	0.303906
0.98	-2.02	38.9469	9	0.58773	0.941883	0.332023
1.03	-2.065	38.1942	9	0.546724	0.932606	0.35152
1.08	-2.125	37.3629	9	0.508566	0.924559	0.367433
1.12	-2.195	36.6613	9	0.476607	0.918088	0.371443
1.16	-2.275	35.9198	9	0.447967	0.908529	0.374467
1.22	-2.44	34.7497	9	0.400035	0.891034	0.363364
1.26	-2.56	33.9231	9	0.372169	0.881357	0.356223
1.3	-2.725	33.0682	9	0.339496	0.864056	0.338538
1.33	-2.85	32.4096	9	0.318408	0.853261	0.32714
1.365	-3.005	31.6213	9	0.296234	0.838482	0.311795
1.4	-3.185	30.809	9	0.272497	0.822688	0.295529
1.44	-3.44	29.8566	9	0.243842	0.79704	0.270441
1.49	-3.82	28.6331	9	0.209139	0.757022	0.233951
1.525	-4.125	27.7567	9	0.186393	0.72513	0.209843
1.55	-4.385	27.1195	9	0.170202	0.693263	0.189246
1.6	-5	25.8213	9	0.138929	0.620065	0.149501
1.65	-6	24.485	9	0.104915	0.48462	0.0999967
1.7	-8	23.1177	9	0.0724969	0.237673	0.0488106
1.72	-10.3	22.5673	9	0.0609656	0.0778781	0.0305449

Tabelle A.3: Kritische Linie des Hexadekan–CO_2–Mischsystems für $\xi = 1$. Die Simulationsbox ist kubisch mit $V = L^3$. Alle Angaben in Lennard–Jones–Einheiten.

T	μ_1	μ_5	L	ρ_c	x_c (CO_2)	p_c
0.75	-2.0225	43.6925	6.74	0.759798	0.930539	0.157682
0.81	-1.965	43.0153	6.74	0.694601	0.924906	0.234159
0.88	-1.918	42.1083	6.74	0.65845	0.928297	0.312047
0.93	-1.9	41.3814	9	0.636473	0.928434	0.360941
0.98	-1.92	40.5534	9	0.591037	0.921345	0.387122
1.03	-1.96	39.6604	9	0.547467	0.91295	0.404228
1.08	-2.018	38.7201	9	0.510254	0.906713	0.415528
1.12	-2.075	37.9244	9	0.482554	0.902326	0.422495
1.16	-2.1525	37.0928	9	0.451875	0.89189	0.424568
1.22	-2.3	35.7805	9	0.406986	0.878805	0.411815
1.26	-2.42	34.8694	9	0.378534	0.869473	0.403237
1.33	-2.68	33.2062	9	0.328717	0.848113	0.371492
1.4	-3.025	31.4557	9	0.278691	0.813726	0.33113
1.45	-3.325	30.169	9	0.244774	0.785737	0.29676
1.49	-3.63	29.1006	9	0.215909	0.756485	0.265606
1.55	-4.185	27.4687	9	0.175645	0.69414	0.209037
1.6	-4.825	26.0664	9	0.141517	0.619396	0.162075
1.65	-5.825	24.6307	9	0.10655	0.484826	0.106953
1.7	-7.7	23.1742	9	0.0742509	0.253443	0.0517459
1.725	-11.85	22.4292	9	0.0572967	0.0314325	0.0243509

Tabelle A.4: Kritische Linie des Hexadekan–CO_2–Mischsystems für $\xi = 0.9$. Die Simulationsbox ist kubisch mit $V = L^3$. Alle Angaben in Lennard–Jones–Einheiten.

T	μ_1	μ_5	L	ρ_c	x_c (CO_2)	p_c
0.74711	-1.96518	44.2029	6.74	0.772227	0.922808	0.212405
0.85	-1.89	42.8497	6.74	0.691914	0.923811	0.312442
0.95	-1.874	41.3182	9	0.616431	0.916597	0.393835
1.05	-1.945	39.5093	9	0.540701	0.910242	0.430009
1.16	-2.123	37.2641	9	0.455126	0.891294	0.435982
1.25	-2.355	35.2464	9	0.389425	0.871701	0.417018
1.35	-2.747	32.8203	9	0.314889	0.837237	0.368796
1.45	-3.31	30.2378	9	0.244051	0.785287	0.298333
1.55	-4.162	27.5176	9	0.176017	0.695127	0.213697
1.65	-5.71	24.6627	9	0.109723	0.498106	0.11389
1.7	-7.6	23.1852	9	0.0750793	0.262764	0.0544594

Tabelle A.5: Kritische Linie des Hexadekan–CO_2–Mischsystems für $\xi = 0.886$. Die Simulationsbox ist kubisch mit $V = L^3$. Alle Angaben in Lennard–Jones–Einheiten.

μ_1	μ_5	L	p	ρ_g	ρ_f	x_g (CO_2)	x_f (CO_2)	γ
-2.84	36.6868	6.74[1]	0.208601[2]	0.205057	0.275755	0.991923	0.543956	(0.139787)
-2.84	36.6892	6.74	0.208621[2]	0.204819	0.27609	0.991929	0.543784	0.175209
-3	36.6274	6.74	0.178802[2]	0.172511	0.255394	0.992926	0.486385	0.208586
-3.35	36.5291	6.74	0.128532[2]	0.120321	0.226552	0.993979	0.392856	0.268345
-3.6	36.4793	6.74	0.102145[2]	0.094005	0.210846	0.994103	0.331305	0.302104
-4	36.4281	6.74	0.071154[2]	0.064266	0.193091	0.993696	0.250915	0.343157

Tabelle A.6: Isotherme bei $T = 1.16\ \frac{\varepsilon}{k}$ mit $\xi = 0.886$. Die Simulationsbox ist gestreckt ($V = 2L^3$). Alle Angaben in Lennard–Jones–Einheiten. Eingeklammerte Werte wurden in den entsprechenden Graphen nicht berücksichtigt. [1] Kubische Box, [2] Gasdruck.

μ_1	μ_5	L	p	ρ_g	ρ_f	x_g (CO_2)	x_f (CO_2)	γ
-2.55	35.9631	9	0.258923	0.291633	0.398278	(0.983878)	(0.76583)	(0.05988)
-2.8	36.0137	6.74	0.20644	0.216942	0.345633	0.989744	0.68318	(0.12412)
-3	36.0537	6.74	0.167435	0.173969	0.312007	0.991985	0.621724	(0.17213)
-3.35	36.1199	6.74	0.12862[1]	0.121068	0.269888	0.993347	0.528524	0.237184
-3.6	36.1492	6.74	0.102058[1]	0.094770	0.245872	0.993624	0.461544	0.27955
-4	36.1962	6.74	0.070966[1]	0.064134	0.218478	0.993389	0.369419	0.330222

Tabelle A.7: Isotherme bei $T = 1.16\ \frac{\varepsilon}{k}$ mit $\xi = 1$. Die Simulationsbox ist gestreckt ($V = 2L^3$). Alle Angaben in Lennard–Jones–Einheiten. Eingeklammerte Werte wurden in den entsprechenden Graphen nicht berücksichtigt. [1] Gasdruck.

μ_1	μ_5	L	p	ρ_g	ρ_f	x_g (CO_2)	x_f (CO_2)	γ
-4.5	27.3777	9	0.17266	0.131214	0.175597	(0.811739)	(0.521824)	(0.014034)
-4.8	27.3108	9	0.142436	0.106635	0.165574	0.822666	0.441891	(0.026676)
-5.2	27.2413	9	0.111326	0.081563	0.154753	0.827498	0.353511	0.042023
-5.7	27.1795	9	0.081365	0.059742	0.144494	0.814484	0.267493	0.056715
-6.2	27.1344	9	0.059313	0.044656	0.137184	0.786018	0.200723	0.070582
-6.7	27.104	9	0.044263	0.033904	0.13172	0.745023	0.149364	0.079761
-7.5	27.0701	6.74[1]	0.025883	0.023263	0.12569	0.640601	0.092615	(0.079086)
-9	27.0476	6.74[2]	0.015597	0.013327	0.120272	0.42528	0.036694	(0.105979)
-10	27.053	6.74[2]	0.013521	0.010592	0.119211	0.279609	0.019325	(0.100312)

Tabelle A.8: Isotherme bei $T = 1.55 \frac{\varepsilon}{k}$ mit $\xi = 0.9$. Die Simulationsbox ist gestreckt ($V = 2L^3$). Alle Angaben in Lennard–Jones–Einheiten. [1] Kubische Box, [2] ungewichtete Simulation.

Anhang B

Programmcode

In diesem Teil des Anhangs wird der komplette Code der Monte–Carlo–Simulation vorgestellt. Das ursprüngliche Programm konnte eine multikanonische Simulation von Polymeren durchführen und wurde im Rahmen dieser Arbeit auf Polymer–Lösungsmittel–Mischungen erweitert. Die hier vorgestellte, parallelisierte Version ermöglicht Simulationen auf der Cray–T3E, aber auch auf einzelnen Prozessoren.

Der Code setzt sich aus dem Hauptprogramm (melt.c) und auf verschiedene Dateien verteilte Funktionen zusammen. Das Executable wird mit Hilfe eines Makefiles kompiliert und verlinkt („make mc“). Der Headerfile element.h enthält neben weiteren Systemparametern auch die Definitionen der Datenstrukturen. Die Auswerteroutinen wurden nicht aufgeführt. Das Programm findet sich auch auf der beigelegten CD.

Zum Ausführen des Programmes benötigt man eine Parameterdatei mit Namen „eingabe“:

Steps=40000	Anzahl der Monte–Carlo–Zyklen,
out=10	Analyse der momentanen Konfiguration nach „out“ Schritten,
	Ausgabe in die Ergebnisdatei „histo.dat“,
seed=-1	Anfangswert für den Zufallszahlengenerator,
	-1: Auswahl durch Zeitabfrage,
datei=conf_out	Datei zum Einlesen und Abspeichern von Teilchenpositionen,
reptation=10	Definition eines Monte–Carlo–Zyklus: Anzahl der Reptations–,
local=1	lokalen und
cbgc=100	großkanonischen Schritte.

Die Teilchenpositionen und Angaben zur Temperatur, den chemischen Potentialen und der Simulationsboxgröße befinden sich vor dem Start in „conf_out“. Am Ende der

Simulation wird diese Datei aktualisiert.

T=1.55	Simulationstemperatur in LJ–Einheiten,
CPA=27.5151	chemisches Potential der Polymere,
CPB=-4.165	chemisches Potential der Monomere,
EW=0	
FW=0	
J=0	
LS=11.3 11.3 11.3	Größe der Simulationsbox,
t=0	MC–Zeit bei Start,
76 380	Anzahl der Polymere, Anzahl der „Polymer–Monomere“,
181 181	Anzahl der Lösungsmittelteilchen,
...	Es folgen die Positionsvektoren.

Makefile

```
TARGET  = BIN
SUBTREE =

SRCs= melt.c \
      r250.c \
      sysin.c \
      sysout.c \
      wach.c \
      element.c \
      mcmove.c \
      cbgc.c \
      local.c \
      rep.c \
      pressure.c \
      weight.c

## OBJs:=$(subst .c,.o,$(SRCs))

OBJs= melt.o \
      r250.o \
      sysin.o \
      sysout.o \
      wach.o \
      element.o \
      mcmove.o \
      cbgc.o \
      local.o \
      rep.o \
      pressure.o \
      weight.o

HDRs= element.h \
      r250.h

#CC        = env TARGET=cray-t3e cc
#CFLAGS    = $(STDCFLAGS) -O2 -X16 -I /usr/include/mpp
CC         = cc
STDCFLAGS  =
CFLAGS     = $(STDCFLAGS) -O2
LDFLAGS    = -lm
DEBUG      = $(STDCFLAGS) -g
VERBOSE    =

.PHONY : clean debug verbose backup

say :
echo $(SRCs)
echo $(OBJs)

## Here is how to create the objects:

%.o : %.c $(HDRs)
$(CC) $(CFLAGS) -c $<

mc : $(OBJs)
$(CC) $(CFLAGS) -o $(TARGET) $(OBJs) $(LDFLAGS)

generate : r250.o generate.o
$(CC) $(CFLAGS) -o G r250.o generate.o $(LDFLAGS)

debug   : $(OBJs)
$(CC) -o $(TARGET) $(OBJs) $(LDFLAGS) $(DEBUG)

verbose : $(SRCs)
$(CC) -DVERBOSE -g -o bfl bdibfl.c $(STDCFLAGS) $(LDFLAGS)

backup  : $(SRCs)
zoo auh melt.zoo $(SRCs)

distclean:
rm -ef $(OBJs)

clean :
rm -ef $(OBJs) $(TARGET)

new     :
touch *.c
```

Header: element.h

```
/*************************************************************
 * element.h: header file                                    *
 *            contains substitution macros, function macros, *
 *            definitions of data structures and function    *
 *            prototypes                                     *
 *                                                           *
 * last modification : 23/05/2003                            *
 *************************************************************/
```

```
/*#define PAR*/
/*#define SWITCH*/
/*#define NOSYNC*/
/*#define PRESSURE*/
/*#define T3D */
/*#define FIXLENGTH 0.974732*/
/*#define STRIPE     1 */
/*#define HARDWALLS*/
/*#define REPULSIVE*/

#define PRESSURE

#define TEST
#define NPOLYMAXA  1000    /* A: POLYMER (N=5) */
#define NMONOMAXA  5000
#define NPOLYMAXB  2000    /* B: CO2 (N=1)     */
#define NMONOMAXB  2000

/* number of trial vectors in ccb/rep */
#define W_ALL      25

/* maximal number of monomers in wwlist*/
#define MAX_ANZAHL (NMONOMAXA+NMONOMAXB)

#define NRSUBX     5      /* defines box size      */
#define NRSUBY     5      /* must be larger than 2 */
#define NRSUBZ     5

#define LARGE
/* speeds up calculations for box size>5 */
/* cannot be used for system sizes<4     */

#define MINA 0             /* defines simulation interval */
#define MAXA 500

#define INFENERGY  200
#define JwallN_MAX 1024
#define NPOLYBRUSH 0

#define NBOXX    4
#define NBOXY    4
```

```
/*-------------------------------------*
 | define energy interaction parameters |
 | (EPSILONA and SIGMAA :=1 !)          |
 *-------------------------------------*/

/* Epsilon: depth of LJ potential */
#define EPSILONB   0.726         /* B:  CO2 (N=1)   */
#define EPSILONAB  0.754921913   /* AB: POLYMER-CO2 */

/* Sigma: zero of LJ-potential   */
#define SIGMAB     0.816
#define SIGMAAB    0.908
#define SIGMAB6    0.295216721   /* Sigma^6 */
#define SIGMAAB6   0.56042189


#define OK       0
#define ERROR    1
#define TRUE     1
#define FALSE    0
#define OUT     -1
#define MEASURE  1

#ifdef PAR
#define NOSYNC
#endif

# ifndef T3D
# define PRINTF     printf
# define FPRINTF    fprintf
# define PRINTFS    printf
# define FPRINTFS   fprintf
# else
# define PRINTF     printf
# define FPRINTF    fprintf
# define PRINTFS    printf
# define FPRINTFS   fprintf
# endif

/*-------------------------------------------------------------*/

/*------------------------*
 | Makros fuer Ringlisten |
 *------------------------*/

#define BOXNR(a,b,c) ((a)+NRSUBX*(b)+NRSUB2*(c))
#define IFLOAT(a,L) ((int)((a)/L))
```

```
#define FBOXNR(ax,ay,az) BOXNR(IFLOAT(ax,LSSUBX),\
IFLOAT(ay,LSSUBY),IFLOAT(az,LSSUBZ))

#define VBOXNR(a) FBOXNR((a)->x,(a)->y,(a)->z)


/*-----------------------------*
 | Makros zur Energieberechnung |
 *-----------------------------*/

#define RNULL2INV (1.0/SQUARE(1.5))

/* Offset fuer Bondenergie, diese hat Nullpunkt bei ca. 20 */
#define BOND_OFFSET (-20.0)

#define BOND(x) (-33.75*log(1.0- (x)*RNULL2INV))

/* sets LJ_potential = 0 at RANGE */
#define NULLPUNKT (0.007751464836)

#define LENJAA(x)   ((SQUARE(x)-x)+NULLPUNKT)
#define RANGEAA     (2.0*1.12246204830937298)
#define RANGE2AA    SQUARE(RANGEAA)

#define LENJBB(x)   EPSILONB*(SQUARE(SIGMAB6*x)-\
                            (SIGMAB6*x)+NULLPUNKT)

#define RANGEBB     (2.0*1.12246204830937298*SIGMAB)
#define RANGE2BB    SQUARE(RANGEBB)

#define LENJAB(x)   EPSILONAB*(SQUARE(SIGMAAB6*x)-\
                             (SIGMAAB6*x)+NULLPUNKT)
#define RANGEAB     (2.0*1.12246204830937298*SIGMAAB)
#define RANGE2AB    SQUARE(RANGEAB)
#define RANGE       (MAX(MAX(RANGEAA,RANGEBB),RANGEAB))

/* maximale Verschiebung fuer local displacement*/
#define MAX_STEP    0.3

#define NRSUB2  (NRSUBX*NRSUBY)
#define NRSUB3  (NRSUBX*NRSUBY*NRSUBZ)


/*-------------------------------------*
 | Makros fuer mathematische Funktionen |
 *-------------------------------------*/
```

```
#define PI 3.141592
#define ABS(x) (( (x)>0 )? (x) : -(x) )
#define MAX(x,y) (( (x)>(y) )? (x) : (y) )
#define MIN(x,y) (( (x)<(y) )? (x) : (y) )
#define SQUARE(x) ( (x)*(x) )
#define CUBE(x) ( (x)*(x)*(x) )
#define ASGN(x) (( (x)>0 )? (-1.0) : (1.0) )


/*-------------------------------------------------*
 | Makros fuer mathematische Funktionen mit Vektoren |
 *-------------------------------------------------*/

#define BETRAG(a)    (sqrt(SQUARE((a)->x)+SQUARE((a)->y)+\
                                          SQUARE((a)->z)))

#define BETRAG2(a)   (SQUARE((a)->x)+SQUARE((a)->y)+\
                                       SQUARE((a)->z))

#define DOT(a,b)     (((a)->x)*((b)->x) + ((a)->y) *\
((b)->y)+ ((a)->z) * ((b)->z))

#define CWINKEL(a,b) ((DOT(a,b))/(BETRAG(a)*BETRAG(b)))

#define DIST(a,b)    (sqrt(SQUARE((a)->x - (b)->x)+\
                           SQUARE((a)->y - (b)->y)+\
                           SQUARE((a)->z - (b)->z)))

#define DISTANCE(a,b) (sqrt(SQUARE((a)->x - (b)->x)+\
                            SQUARE((a)->y - (b)->y)+\
                            SQUARE((a)->z - (b)->z)))
#define DISTANCE2(a,b) (SQUARE((a)->x - (b)->x)+\
                         SQUARE((a)->y - (b)->y)+\
                         SQUARE((a)->z - (b)->z))

#define VCOPY(a,b)     (b)->x = (a)->x;\
                       (b)->y = (a)->y;\
                       (b)->z = (a)->z;

#define DIFF(a,b,c)    (c)->x = (a)->x - (b)->x;\
                       (c)->y = (a)->y - (b)->y;\
                       (c)->z = (a)->z - (b)->z

#define ADD(a,b,c)     (c)->x = (a)->x + (b)->x;\
                       (c)->y = (a)->y + (b)->y;\
                       (c)->z = (a)->z + (b)->z
```

```
#define SDOT(n,a,b)    (b)->x = (a)->x*(n);\
                       (b)->y = (a)->y*(n);\
                       (b)->z = (a)->z*(n)

#define VDOT(a,b,c)    (c)->x = (a)->y*\
                                (b)->z-(a)->z * (b)->y;\
                       (c)->y = (a)->z*\
                                (b)->x-(a)->x * (b)->z;\
                       (c)->z = (a)->x*\
                                (b)->y-(a)->y * (b)->x



#define VPRINT(a) printf("%f %f %f\n",(a)->x,(a)->y,(a)->z)


/*----------------------*
 | Strukturdefinitionen |
 *----------------------*/

typedef struct o_element ELEMENT;
typedef struct o_box     BBOX;

typedef struct {
  double x;
  double y;
  double z;
} MYVEC;

struct o_element {
  int monomer;               /* monomer number */
  int type;                  /* monomer type   */
  struct o_element *before;  /* pointer to previous element  */
  struct o_element *after;   /* pointer to next element      */
};


/*                                          element 0     */
/*                                          after         */
/*                     element 1       <------before       */
/*                     after           ------>             */
/*  element 2     <-----before                             */
/*  after         ----->                                   */
/*  before                                                 */

struct o_box {
  struct o_element *first;  /* points somewhere in the list, */
                            /* there is no beginning         */
  int population;           /* number of list elements ,     */
                            /* number of particles in box    */
  int neighbours[125];      /* indices of neighboring boxes  */
                            /* - 27 inner and (125-27) outer */
};

typedef struct DATA {
  double bondangleA;
  double bondangleB;
  double bondlength1A;
  double bondlength1B;
  double bondlength2A;
  double bondlength2B;
  double bondlengthmaxA;
  double bondlengthmaxB;
  double gyrA;
  double gyrB;
  double endA;
  double endB;
  double energy_ljA;
  double energy_ljB;
  double energy_ljAB;
  double energy_feneA;
  double energy_feneB;
  double energy_sqA;
  double energy_sqB;
  double energy_sqAB;
  double energy_wall;
  double msdA;
  double msdB;
  double px;
  double py;
  double pz;
  double pwall1;
  double pwall2;
  double virialx;
  double virialy;
  double virialz;
  int    nrchainsleftA;
  int    nrchainsleftB;
} DATA;


#ifndef FUNKTIONEN
#define FUNKTIONEN
```

```
/*------------------------------------------------------------*/

/*----------------------*
 | Funktionendeklaration |
 *----------------------*/


/*-------------------------------------*
 | element.c: box and energy functions |
 *-------------------------------------*/

void put_element(int ibox, int ident);
/* puts monomer in box */

void del_element(int ibox,int ident);
/* deletes element from box */

void populate(void);
/* fills boxes with monomers from starting configuration */

double elj_energie(int ident, int type, MYVEC *mon);
/* calculates LJ interactions with BOX method */

double fene(int ident, int type, MYVEC *neu);
/* calculates bond energy with all neighbors  */

double eenergie(void);
/* calculates total energy of the system using elj_energie,  */
/* fene, und wenergy, also checks box scheme                 */


/*---------------------------*
 | wach.c: analysis functions |
 *---------------------------*/

void analyse(int zeit,int lauf);
/* updates data-array by calling analysis functions
  - writes results to "histo.dat" */

void bondwinkel();   /* calc. average cos(bondangle)         */
void bondlength();   /* calc. average bondlength, b**2, bmax */
void calc_polymer(); /* calc. (R_e^2) and (R_g^2)            */

void calc_nrchainsleft(void);
/* calc. #chains in the left half of sim. box */

void calc_cm(int pol, MYVEC *start,MYVEC *result);
/* calc. center-of-mass of polymer at pos. start */

void calc_msd();     /* calc. mean-square displacement */
void calc_energy();  /* calc. total LJ, FENE and wall energy */

/* pressure.c */
int  calc_p(void);   /* calc. pressure via virial expression */


/*--------------------------------*
 | mcmove.c: functions for cbgc.c |
 *--------------------------------*/

double bondenergy(int type, double length);
/* calc. LJ+FENE energy of two monomers (same type) */
/* at distance "length"                             */

double calcIntegral(double *, double *);
/* updates rtable[] and itable[] in mcmove.c        */
/* - prerequesit for choose_length()                */

double choose_length(int,double rnd);
/* chooses Boltzmann-distributed bondlength         */

void   make_list(int, int, MYVEC *, double);
/* creates interaction-arrays wwlisteA and wwlisteB */

double lj_energy(int, MYVEC *);
/* calculates LJ interaction of single monomer      */
/* with all monomers in wwlisteA+B                  */


/*------------------------------------------------*
 | cbgc.c: grand-canonical configurational-bias move |
 *------------------------------------------------*/

double cbgc(int type);
/* tries to insert (50%) or delete (50%) chains from sim box */

int insert (int type);
int delete (int type);


/*-----------------------*
 | local.c: local MC move |
 *-----------------------*/
```

```
void lmtrial(MYVEC *new, MYVEC *alt, double ** rnd_ptr_adr);
/* creates trial vector for lmcmove */

int  lmcmove(void);
/* tries nrmonA+nrmonB local moves, */
/* monomers are chosen at random    */


/*-------------------------*
 | rep.c: reptation MC move |
 *-------------------------*/

int reptation(void);              /* calls rep fct */
int rep      (int type);
/* tries nrmonA (if type==0) or nrmonB (if type==1) */
/* reptation moves, monomers are chosen at random   */


/*-------------*
 | io-functions |
 *-------------*/

int sysout(char *,int );
/* reads sim. specifications and monomer positions  */
/* from "conf_out"                                  */

int sysin(char *);
/* writes sim. specifications and monomer positions */
/* to "conf_out"                                    */

void init_weight(void);
/* initializes weights for reweighted simulation    */


/*---------------*
 | wall functions |   -these functions have not been adjusted
 *---------------*/

extern double (*wenergy)(MYVEC *pos);
extern double (*wbenergy)(MYVEC *pos);

double wall_energy(MYVEC *pos);
/* calculates interaction energy with walls */

double bwall(MYVEC *pos);
/* purely repulsive interaction SEwall */

double piston(MYVEC *pos);
/* calculates interaction energy with walls and piston */

void   profile(int status);
#endif

/*

Erklaerungen (wichtigste Makros und Datenstrukturen)
====================================================

BOXNR        : Ermittelt aus (Integer)
               Koordinaten die zugehoerige Boxnummer

IFLOAT       : Rechnet Fliesskommakoordinaten in
               zugehoerige Boxkoordinaten um

BOXNR        : Wie BOXNR, akzeptiert aber
               Fliesskommakoordinaten

BOX_PERIODIC : Ermittelt, mittels Funktion box_periodic,
               fuer Teilchen die zugehoerige Boxennummer

BOND         : FENE-potential aus rad2 berechnen
LENJ         : Lenard Jones Potential aus rad6inv berechnen
DIST         : Abstand zwischen zwei Punkten
BETRAG       : Betrag eines (Pointer auf) Vektors
PRODUKT/DOT  : Vektorprodukt
CWINKEL      : Winkel zwischen zwei Vektoren
DISTANCE2    : Quadrat der Distanz zwischen zwei Punkten
BETRAG2      : Betragsquadrat eines Vektors


(o_)vector   : Datenstruktur fuer Vektoren
(o_)element  : Datenstruktur fuer Ringlisten,
               deren Elemente auf Teilchen zeigen
(o_)box      : Datenstruktur fuer Boxen,
               first zeigt auf ein Element der Ringliste,
               welche die Teilchen enthaelt,
               die sich in der Box befinden

pos          : absolute positions of the monomers
poslat       : periodic boundary conditions applied
*/
```

Hauptprogramm: melt.c

```
/*******************************************
 * melt.c: contains main()                 *
 *                                         *
 * last modification : 23/05/2003          *
 *******************************************/


/* include header files */
#include "element.h"
#include "r250.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>


/* global variables */
int         nrchainsA, nrmonA, polA;
int         nrchainsB, nrmonB, polB;
MYVEC       *posA, *poslatA;
MYVEC       *posB, *poslatB;

double      T, Tinv,J,CPA,CPB,Ewall,Fwall;
double      LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;

#ifdef STRIPE
double      SEwall,SWidth=0;
#endif

int         mcs_start;

MYVEC       *cm0A,*cm0B,pressure;
/* used in wach.c */

double      cenergie=0,e_start;
/* required for a check of the total energy */

double      mx=0,my=0,mz=0,manzahl=0;
/* used in mcmove.c */

ELEMENT     *element_liste;
/* used in element.c, local.c */

BBOX        boxliste[NRSUB3];

double      pcenergie;

DATA        data;
/* used in wach.c to transfer results to main */

double      Jwallwtab[JwallN_MAX]  ;
/* used in switch_Jwall.c, keep global for i/o */

double      w[NPOLYMAXA+1][NPOLYMAXB+1];
/* array to store weighting factors */


int main(int argc, char *argv[]) {

   int     outstep, countstep, seed, j;
   int     mcs, mcsmax, nreptation, local, ncbgc=0, placc;
   int     cbgc_acc0=0,cbgc_acc1=0,cbgc_count0=1,cbgc_count1=1;
   int     lakzeptiert=0,lgesamt=1,rep_acc=0,rep_count=1,\
           out_ctr=0;
   int     pol_print_ctr=360;
   double  e_end;
   double  zbondA,zbondB;
   double  config_ctr=0;
   char    name[80], filenameout[30]="conf_out",dname[30],\
           out_name[30];
   FILE    *eingabe,*log_file;
   time_t  now;

   extern long _MPP_MY_PE, _MPP_N_PES;
   /* only used on T3E machines - number of processors */

   /* test variables */
   double energy1, energy2, lj_energy3=0, lj_energy4=0;
   int i=0,ctr1,ctr2;
   static double *rndtabA=NULL, *rnd_ptrA=NULL;
   int a [2000];
   int dummy1;
   FILE *prob;


   /*-----------------*
    | read parameters |
    *-----------------*/

   time(&now);
   PRINTFS("program starts at %s\n",ctime(&now));
```

```
   if (argc>1) {


     /*-------------------------------------------------*
      | a) for use on a regular single processor machine |
      *-------------------------------------------------*/

#ifndef T3D
     sprintf(name,"%s",argv[1]);              /* 'eingabe' */
     if ((eingabe=fopen(name,"r"))==NULL)
       {
         fprintf(stderr,"couldn't find %s \n",name);
         exit(1);
       }

     /* read parameters from 'eingabe' file */

     fscanf(eingabe,"Steps=%d \n",&mcsmax);
     fscanf(eingabe,"out=%d \n",&outstep);
     fscanf(eingabe,"seed=%d \n",&seed);
     fscanf(eingabe,"datei=%s \n",dname);     /* 'conf_out' */
     fscanf(eingabe,"reptation=%d\n",&nreptation);
     fscanf(eingabe,"local=%d\n",&local);
     fscanf(eingabe,"cbgc=%d\n",&ncbgc);
     fclose(eingabe);

     if(seed<0) time ((time_t *)&seed);
     /* get seed for rnd number generator r250 from cpu-time */
     r250_srandom(seed);
     /* assign seed to rnd number generator */

     sysin(dname);
     /* read sim. specifications and monomer vectors */
     /* from file "conf_out" */

     init_weight();
     /* initializes weights for reweighted simulation */

     /*------------------------*
      | b) for use on T3E only |
      *------------------------*/

#else   /* if T3D is defined */

     sprintf(filenameout,"%s_%d",filenameout,_MPP_MY_PE);
     sprintf(name,"%s_%d",argv[1],_MPP_MY_PE); /* 'eingabe' */
     if ((eingabe=fopen(name,"r"))==NULL)
       {
         fprintf(stderr,"couldn't find %s \n",name);
         exit(1);
       }

     /* read parameters from 'eingabe' file */

     fscanf(eingabe,"Steps=%d \n",&mcsmax);
     fscanf(eingabe,"out=%d \n",&outstep);
     fscanf(eingabe,"seed=%d \n",&seed);
     fscanf(eingabe,"datei=%s \n",dname);     /* 'conf_out' */
     fscanf(eingabe,"reptation=%d\n",&nreptation);
     fscanf(eingabe,"local=%d\n",&local);
     fscanf(eingabe,"cbgc=%d\n",&ncbgc);
     fclose(eingabe);

     sprintf(dname,"%s_%d",dname,_MPP_MY_PE);

     if(seed<0) time ((time_t *)&seed);
     /* get seed for rnd number generator r250 from cpu-time */
     seed += 1711*_MPP_MY_PE;
     /* assign different seed for each processor */
     r250_srandom(seed);

     sysin(dname);
     /* read sim. specifications and monomer vectors */
     /* from file "conf_out" */

     init_weight();
     /* initializes weights for reweighted simulation */
#endif


     /*-----------all parameters read !-----------------*/


     Tinv  = -1.0/T;

     calcIntegral(&zbondA,&zbondB);
     /* update rtable[] and itable[] which are used */
     /* in cbgc.c (choose_length)                   */

     populate();
     /* populate boxes with start configuration */

     /* initialize check for total energy */
     e_start= eenergie();
```

```
cenergie=0;


for (mcs=0;mcs<mcsmax;) {

  /*----------*
   | MC steps |
   *----------*/

  /*-----------------------------------------------*
   | grand-canonical configurational-bias MC moves |
   *-----------------------------------------------*/

  /* adjust CP for configurational-bias move */
  CPA += T*(log(LSX*LSY*LSZ)+(polA-1)*log(zbondA));
  CPB += T*(log(LSX*LSY*LSZ/(SIGMAB*SIGMAB*SIGMAB))+\
           (polB-1)*log(zbondB));  /* SIGMAA := 1 !!! */

  for(i=0;i<ncbgc;i++) {
    cbgc_acc0   += cbgc(0);
    cbgc_acc1   += cbgc(1);
    cbgc_count0 += 1;
    cbgc_count1 += 1;
  }

  /* reajust CP */
  CPA -= T*(log(LSX*LSY*LSZ)+(polA-1)*log(zbondA));
  CPB -= T*(log(LSX*LSY*LSZ/(SIGMAB*SIGMAB*SIGMAB))+\
           (polB-1)*log(zbondB));


  /*---------------*
   | local MC moves |
   *---------------*/

  for(i=0;i<local;i++) {
    lakzeptiert += lmcmove();
    lgesamt     += nrmonA+nrmonB;
  }


  /*--------------------*
   | reptation MC moves |
   *--------------------*/

  for(i=0;i<nreptation;i++) {
    rep_acc   += reptation();


    if (polA==1)
      rep_count += nrchainsB;
    else if (polB==1)
      rep_count += nrchainsA;
    else
      rep_count += nrchainsA+nrchainsB;
  }

  mcs++;   /* MC step counter++ */


  /*------------end of MC circle---------------*/


  /*--------------------------------*
   | analysis after 'outstep' steps |
   *--------------------------------*/

  if (mcs%outstep==0) {

    /* calculate total energy in two ways:  */
    /* 1) eenergie in wach.c                */
    /* 2) sum of energy fcts in element.c   */

    energy1 = eenergie();
    analyse (mcs_start+mcs,1);
    energy2 = data.energy_ljA + data.energy_ljB +
              data.energy_ljAB + data.energy_feneA +
              data.energy_feneB;

    if ( (energy1 - energy2) > 1e-6) {
      fprintf(stderr,"error in energy calculation:"
                     "energy(element.c)=%g energy"
                     "(wach.c)=%g\n", energy1, energy2);}

    /* check if acc. energy differences (cenergie) */
    /* = change in total energy (energie1-e_start) */

    e_end = cenergie-(energy1-e_start);
    e_end = sqrt(e_end*e_end);

    if (e_end > 1e-6) {
      fprintf(stderr,"accumulated energy difference=%g"
                     "total change (e_start-e_end)=%g\n",\
                      cenergie,e_start-energy1);}
  } /* mcstep */
```

```
    }


    /*----------------------------------------*
     | write acceptance rate data to 'log' file |
     *----------------------------------------*/

   if ((log_file=fopen("log_file","a"))==NULL)
      {
        fprintf(stderr,"couldn't find %s \n",name);
        exit(1);
      }

    fprintf(log_file,"gccb: p:%g\tm:%g\t local: %g\trep: %g\n"
            ,cbgc_acc0/(double)cbgc_count0\
            ,cbgc_acc1/(double)cbgc_count1\
            ,lakzeptiert/(double)lgesamt\
            ,rep_acc/(double)rep_count);

    fclose(log_file);

    sysout(dname,mcs_start+mcs);
    /* write sim. specifications and monomer vectors */
    /* for next run to file "conf_out"               */
  }


  free(posA);
  free(posB);
  free(poslatA);
  free(poslatB);
  free(cm0A);
  free(cm0B);
  free(element_liste);

  time(&now);
  PRINTFS("program ends at %s\n",ctime(&now));

  return (OK);
}
```

IO: sysin.c

```
/*******************************************************
 * sysin.c: reads simulation specifications            *
 *          and monomer positions from file "conf_out" *
 *                                                     *
 * last modification: 23/05/2003                       *
 *******************************************************/


#include "element.h"
#include <stdio.h>
#include <stdlib.h>

extern int       nrchainsA, nrmonA, polA;
extern int       nrchainsB, nrmonB, polB, mcs_start;
extern MYVEC     *posA, *poslatA,*posB, *poslatB;
extern double    T,CPA,CPB,Ewall,Fwall,J;
extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern MYVEC     *cm0A,*cm0B;

#ifdef STRIPE
extern double    SEwall,SWidth;
#endif

int sysin(char *name)
{
int    i,ii;
MYVEC  *mon,*mon2;
FILE   *in;
double dummy;

   in = fopen( name,"r");


   /*-----------------------------------------------*
    | read simulation specifications from "conf_out" |
    *-----------------------------------------------*/

   fscanf(in,"T=%lf\n",&T);
   fscanf(in,"CPA=%lf\n",&CPA);
   fscanf(in,"CPB=%lf\n",&CPB);
   fscanf(in,"EW=%lf\n",&Ewall);
   fscanf(in,"FW=%lf\n",&Fwall);

#ifdef STRIPE
```

```
   fscanf(in,"SEW=%lf\n",&SEwall);
   fscanf(in,"SW=%lf\n",&SWidth);
#endif

   fscanf(in,"J=%lf\n",&J);
   fscanf(in,"LS=%lf %lf %lf\n",&LSX,&LSY,&LSZ);
   fscanf(in,"t=%d\n\n",&mcs_start);
   fscanf(in,"%d %d", &nrchainsA, &nrmonA);
   fscanf(in,"%d %d\n\n", &nrchainsB, &nrmonB);

   LSSUBX = LSX/((double)(NRSUBX));
   LSSUBY = LSY/((double)(NRSUBY));
   LSSUBZ = LSZ/((double)(NRSUBZ));


   /*-----------------*
    | allocate memory |
    *-----------------*/

   posA           = (MYVEC *) calloc(NMONOMAXA, sizeof(MYVEC));
   poslatA        = (MYVEC *) calloc(NMONOMAXA, sizeof(MYVEC));
   cm0A           = (MYVEC *) calloc(NPOLYMAXA, sizeof(MYVEC));
   posB           = (MYVEC *) calloc(NMONOMAXB, sizeof(MYVEC));
   poslatB        = (MYVEC *) calloc(NMONOMAXB, sizeof(MYVEC));
   cm0B           = (MYVEC *) calloc(NPOLYMAXB, sizeof(MYVEC));
   if (posA==NULL || poslatA==NULL || cm0A==NULL ||
       posB==NULL || poslatB==NULL ||cm0B==NULL) {
      fprintf(stderr,"ERROR: memory allocation denied"
                     "in function sysin\n");
      exit(ERROR);
   }

   if (nrchainsA==0) {
      polA = NMONOMAXA/NPOLYMAXA;
   }
   if (nrchainsB==0) {
      polB = NMONOMAXB/NPOLYMAXB;
   }


   /*-------------------------------------------------*
    | read typeA monomer positions from "conf_out" and |
    | update posA array                               |
    *-------------------------------------------------*/

   for( i=0,mon=posA ; i<nrchainsA ; i++){
      fscanf( in,"%d", &(polA));
      for ( ii=0 ; ii< polA ; ii++){
         fscanf( in,"%lf%lf%lf",&(mon->x),&(mon->y),&(mon->z));
         fscanf( in,"%lf%lf%lf", &dummy, &dummy, &dummy);
         /* space for velocities -                  */
         /* makes conf_out compatible to MD version */
         mon ++;
      }
   }


   /*-----------------------------*
    | update poslatA and cm0A array |
    *-----------------------------*/

   for(i=0;i<nrchainsA;i++) {
     mon  = posA    + i*polA;
     mon2 = poslatA + i*polA;

     calc_cm(polA,mon,(cm0A+i));  /* see wach.c */

     for (ii=0;ii<polA;ii++,mon++,mon2++) {
      mon2->x = mon->x-LSX*((int)(mon->x/LSX));
      mon2->y = mon->y-LSY*((int)(mon->y/LSY));
      mon2->z = mon->z-LSZ*((int)(mon->z/LSZ));

      if ( mon2->x < 0.0 ) mon2->x += LSX ;
      if ( mon2->y < 0.0 ) mon2->y += LSY ;
      if ( mon2->z < 0.0 ) mon2->z += LSZ ;
     }
   }


   /*-------------------------------------------------*
    | read typeB monomer positions from "conf_out" and |
    | update posA array                               |
    *-------------------------------------------------*/

   for( i=0,mon=posB ; i<nrchainsB ; i++){
      fscanf( in,"%d", &(polB));
      for ( ii=0 ; ii< polB ; ii++){
         fscanf( in,"%lf%lf%lf",&(mon->x),&(mon->y),&(mon->z));
         fscanf( in,"%lf%lf%lf", &dummy, &dummy, &dummy);
         mon ++;
      }
   }
   fclose( in);
```

```
   /*------------------------------*
    | update poslatB and cm0B array |
    *------------------------------*/

   for(i=0;i<nrchainsB;i++) {
     mon  = posB    + i*polB;
     mon2 = poslatB + i*polB;

     calc_cm(polB,mon,(cm0B+i));

     for (ii=0;ii<polB;ii++,mon++,mon2++) {
      mon2->x = mon->x-LSX*((int)(mon->x/LSX));
      mon2->y = mon->y-LSY*((int)(mon->y/LSY));
      mon2->z = mon->z-LSZ*((int)(mon->z/LSZ));

      if ( mon2->x < 0.0 ) mon2->x += LSX ;
      if ( mon2->y < 0.0 ) mon2->y += LSY ;
      if ( mon2->z < 0.0 ) mon2->z += LSZ ;
     }
   }

   return(0);

}/* end sysin */
```

IO: sysout.c

```
/*******************************************************
 * sysout.c: writes sim. specifications and monomer    *
 *           positions for next run to file "conf_out" *
 *                                                     *
 * last modification: 23/05/2003                       *
 *******************************************************/


#include "element.h"
#include <stdio.h>

extern double    T,CPA,CPB,Ewall,Fwall,J,LSX,LSY,LSZ;
extern int       nrchainsA, nrmonA, polA;
extern int       nrchainsB, nrmonB, polB;
extern MYVEC     *posA,*posB;

#ifdef STRIPE
extern double    SEwall,SWidth;
#endif

int sysout(char *name,int mcs)
{
   int i, ii;
   MYVEC *mon;
   FILE *out;


   /*-----------------------------------------------------*
    | write sim. specifications for next run to "conf_out" |
    *-----------------------------------------------------*/

   out = fopen(name,"w");

#ifdef STRIPE
   fprintf(out,"T=%g\nCPA=%g\nCPB=%g\nEW=%g\nFW=%g\n"
               "SEW=%g\nSW=%g\nJ=%g\nLS=%g %g %g\nt=%d\n\n"
               "%d %d\n%d %d\n\n",T,CPA,CPB,Ewall,Fwall,\
               SEwall,SWidth,J,LSX,LSY,LSZ,mcs,nrchainsA,\
               nrmonA,nrchainsB,nrmonB);
#else
   fprintf(out,"T=%g\nCPA=%g\nCPB=%g\nEW=%g\nFW=%g\n"
               "J=%g\nLS=%g %g %g\nt=%d\n\n%d %d\n%d %d\n\n",\
               T,CPA,CPB,Ewall,Fwall,J,LSX,LSY,LSZ,mcs,\
               nrchainsA,nrmonA,nrchainsB,nrmonB);
#endif


   /*-----------------------------------------------*
    | write type A - monomer positions to "conf_out" |
    *-----------------------------------------------*/

   for( i=0,mon=posA ; i<nrchainsA ; i++){
      fprintf( out,"%d\n", polA);
      for( ii=0 ; ii<polA ; ii++){
         fprintf( out,"%f\t%f\t%f\n", mon->x, mon->y, mon->z);
         fprintf( out,"0.000 0.000 0.000\n");
         mon++ ;
      }
      fprintf( out,"\n");
   }

   /*-----------------------------------------------*
    | write type B - monomer positions to "conf_out" |
    *-----------------------------------------------*/
```

```
   for( i=0,mon=posB ; i<nrchainsB ; i++){
      fprintf( out,"%d\n", polB);
      for( ii=0 ; ii<polB ; ii++){
         fprintf( out,"%f\t%f\t%f\n", mon->x, mon->y, mon->z);
         fprintf( out,"0.000 0.000 0.000\n");
         mon++ ;
      }
      fprintf( out,"\n");
   }

   fclose(out);
   return(OK);
}
```

IO: weight.c

```
/************************************************************
 * weight.c: initializes weights for weighted simulation    *
 *                                                          *
 * last modification: 23.05.2003                            *
 ************************************************************/

# include "element.h"
# include <stdlib.h>
# include <stdio.h>
# include <math.h>


/************************************************************
 * weight.c: initializes array w[i][j]= log (e.g. of P(i,j) *
 *           i = number of polymersA, j=number of polymersB)*
 *                                                          *
 *           'w' exists in working directory ?              *
 *           no:  -> simulation will not be weighted,       *
 *                   all w[i][j]=0,                         *
 *           yes: -> simulation will be weighted,           *
 *                   w[i][j] according to 'w'-file.         *
 *                                                          *
 * returns:  pointer to 'w'-array                           *
 *                                                          *
 * idea:     w[i][j] is a guess of (the log of the) prob.   *
 *           distribution P(#polyA,#polyB). In cbgc.c       *
 *           (grand-canonical MC step) the Hamiltonian is   *
 *           modified such that in the ideal case (guess =  *
 *           real distribution) P=constant for all          *
 *           (NPOLYMAXA,NPOLYMAXB).                         *
 *           This enables simulations below the critical    *
 *           point.                                         *
 *                                                          *
 * remark:   One can obtain a 'w'-file by extrapolation of  *
 *           an existing data-set.                          *
 ************************************************************/


void init_weight(void)
{
extern double w[NPOLYMAXA+1][NPOLYMAXB+1];
extern double H[NPOLYMAXA+1][NPOLYMAXB+1];
extern double g;
extern int    adjust_reweight;
extern int    iteration_step;
extern long   _MPP_MY_PE;

FILE *dp=NULL; /* file-pointer to w-file */
char w_file_name[30];
int ctr1, ctr2;
int x,y;


  /*--------------------------------------*
   | initialize 'w'-array: set all w[][]=0 |
   *--------------------------------------*/

  for (ctr1=0;ctr1<=NPOLYMAXA;ctr1++)
  {
    for (ctr2=0;ctr2<=NPOLYMAXB;ctr2++)
      { w[ctr1][ctr2]=0;
      }
  }


  /*---------------------------------*
   | simulation will not be weighted |
   | -> all w[][] remain equal to 0  |
   *---------------------------------*/

#ifdef T3D
  sprintf(w_file_name,"w_%d",_MPP_MY_PE);
  if ((dp =fopen(w_file_name,"r"))==NULL)
    /* 'w'-file does not exist -> simulation will not be */
    /* weighted                                          */
```

```
#else
  if ((dp =fopen("w","r"))==NULL)
#endif

    {
      fprintf(stderr,"No weights available !\n"
                     "Continue with unweighted simulation\n");
    }


  /*-----------------------------------------------------*
   | simulation will be weighted                         |
   | -> overwrite 'w'-array with weights from 'w'-file   |
   *-----------------------------------------------------*/

  else
    {
      for(ctr1=0;ctr1<=NPOLYMAXA;ctr1++)
        for (ctr2=0;ctr2<=NPOLYMAXB;ctr2++)
          {
            fscanf(dp,"%d %d",&x,&y);
            fscanf(dp,"%lg\n",&w[x][y]);
          }
     }
}
```

Energieroutinen: element.c

```
/**************************************
 * element.c: box and energy functions *
 *                                     *
 * last modifications: 23/05/2003      *
 **************************************/

#include"element.h"
#include<stdio.h>
#include<math.h>

extern MYVEC     *posA, *poslatA;
extern int       nrmonA,polA;
extern MYVEC     *posB, *poslatB;
extern int       nrmonB,polB;

extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern double    J,T;
extern BBOX      boxliste[NRSUB3];
extern ELEMENT   *element_liste;

static int box_periodic(int ix, int iy, int iz);
/* help-function for populate() */


/**************************************
 * put_element: put monomer in box by  *
 *              setting pointers in    *
 *              elementliste           *
 *              -> double linked list  *
 *                 for each subbox     *
 *                                     *
 * ibox       = box number             *
 * element    = position of monomer in *
 *              elementliste           *
 **************************************/

void put_element(int ibox, int element) {
  ELEMENT *before, *particle;
  BBOX *pbox;

  particle = element_liste + element;
  pbox     = boxliste + ibox;

  /* monomer is not first particle in box */
  if (pbox->population != 0){
    before=(pbox->first)->before;
    before->after    = particle;
    (pbox->first)->before=particle;
    particle->before = before;
    particle->after  = pbox->first;
    pbox->population++;
   }
  /* monomer is first particle in box */
  else {
    pbox->population++;
    pbox->first      = particle;
    particle->before = particle;
    particle->after  = particle;
   }

}
```

```
/**************************************
 * del_element: delete element from    *
 *              box by setting ptrs in *
 *              elementliste           *
 *                                     *
 * ibox       = box number             *
 * element    = position of monomer    *
 *              in elementliste        *
 **************************************/

void del_element(int ibox, int element) {
  ELEMENT *before,*after,*particle;
  BBOX *pbox;

  particle = element_liste + element;
  pbox     = boxliste + ibox;

  before = particle->before;
  after  = particle->after;

  before->after = after;
  after->before = before;
  pbox->first   = before;
  pbox->population--;

}


/**************************************
 * box_periodic: determines box number   *
 *               for a set of normalized,*
 *               non-periodic, integer   *
 *               box coordinates         *
 *                                       *
 * ix, iy, iz :  box coordinates         *
 *               (0,1, .. (NRSUBX,Y,Z)-1)*
 *                                       *
 * remark:       this function is only   *
 *               called in populate()    *
 **************************************/

static int box_periodic(int ix, int iy, int iz){

  /* non-periodic -> periodic coordinates */
  if (ix<0) ix += NRSUBX;
  if (iy<0) iy += NRSUBY;
  if (iz<0) iz += NRSUBZ;

  if (ix>=NRSUBX) ix -= NRSUBX;
  if (iy>=NRSUBY) iy -= NRSUBY;

#ifdef HARDWALLS
/* HARDWALLS   rturn -1  invalid number */
  if (iz>=NRSUBZ) return(-1);
#else
  if (iz>=NRSUBZ) iz -= NRSUBZ;
  return (BOXNR(ix,iy,iz));
#endif

}


/**************************************
 * populate: populates boxes with       *
 *           start configuration        *
 *                                      *
 * remark:   called at the beginning of *
 *           each run                   *
 **************************************/

void populate(void){
  int     i,ix,iy,iz,ibox;
  int     nx,ny,nz,n;
  MYVEC   *mon;
  BBOX    *pbox;
  ELEMENT *act;

  if ((LSSUBX<RANGE)||(LSSUBY<RANGE)||(LSSUBZ<RANGE)) {
     fprintf(stderr,"ERROR: length of subbox %g %g "
                    "%gsmaller than interaction range %g\n",\
                    LSSUBX,LSSUBY,LSSUBZ,RANGE);
     exit(ERROR);
  }


  /*-----------------------------------------*
   | step 1: set up neighbor list in pbox-array |
   *-----------------------------------------*/

  /* loop over all boxes (determines "specified box") */
  for(ix=0;ix<NRSUBX;ix++)
    for(iy=0;iy<NRSUBY;iy++)
      for(iz=0;iz<NRSUBZ;iz++){
```

```
      ibox = BOXNR(ix,iy,iz);
      pbox = boxliste+ibox;
      pbox->population=0;

      n=0;

      for(nx= -1;nx<=1;nx++)
      /* inner boxes: 3*3*3 around "specified box" */
        for(ny= -1;ny<=1;ny++)
        /* neighbor-array position 0-26 */
          for (nz= -1;nz<=1;nz++) {
            (pbox->neighbours)[n]=\
            box_periodic(ix+nx,iy+ny,iz+nz);
            n++;
          }

      for(nx=-2;nx<=2;nx++)
      /* outer boxes: 5*5*5 around "specified box" */
        for(ny=-2;ny<=2;ny++)
        /* neighbor array position 27-125 */
          for (nz= -2;nz<=2;nz++) {
            if ( (abs(nx)==2)||(abs(ny)==2)||(abs(nz)==2) ) {
              (pbox->neighbours)[n]=\
              box_periodic(ix+nx,iy+ny,iz+nz);
              n++;
             }
          }
    }


/*-------------------------------------------------------*
 | step 2a: - set monomer# and monomer type              |
 |            in elementliste-array for type A particles |
 |          - put monomers of type A in boxes by calling |
 |            put_element()                              |
 *-------------------------------------------------------*/

element_liste  =\
(ELEMENT *)calloc(NMONOMAXA+NMONOMAXB,sizeof(ELEMENT));

act = element_liste;
mon = poslatA;

for(i=0;i<nrmonA;i++,mon++,act++){
  act->monomer = i;  /* set monomer#          */
  act->type    = 0;  /* set monomer type (A!) */

  /* determine box number */
  ibox = FBOXNR(mon->x,mon->y,mon->z);

  if ((ibox<0) || (ibox>=NRSUB3)){
    fprintf(stderr,"ERROR: ibox out of range (%f %f %f)\n",
                   mon->x,mon->y,mon->z);
    exit(ERROR);
  }
  put_element(ibox,i);  /* put element in box */
}


/*-------------------------------------------------------*
 | step 2b: - set monomer# and monomer type              |
 |            in elementliste-array for type B particles |
 |          - put monomers of type B in boxes by calling |
 |            put_element()                              |
 *-------------------------------------------------------*/

act = element_liste+NMONOMAXA;
/* pointer to first typeB particle */

mon = poslatB;

for(i=0;i<nrmonB;i++,mon++,act++){
  act->monomer = i;  /* set monomer#              */
 /* monomer# gives pos. in pos & poslat arrays    */
 /* (two particles (A,B) may have the same number */

  act->type    = 1;      /* set monomer type (B!) */

   /* determine box number */
  ibox = FBOXNR(mon->x,mon->y,mon->z);

  if ((ibox<0) || (ibox>=NRSUB3)){
    fprintf(stderr,"ERROR: ibox out of range (%f %f %f)\n",
            mon->x,mon->y,mon->z);
    exit(ERROR);
  }
  put_element(ibox,i+NMONOMAXA);  /* put element in box */
}


/*-------------------------------------------------*
 | step 3: verify that all particles are put in boxes |
 *-------------------------------------------------*/
```

```
  for (ibox=0,i=0;ibox<NRSUB3;ibox++)
      i += (boxliste+ibox)->population;

  if ( i != nrmonA+nrmonB) {
    fprintf(stderr,"ERROR: number of particles in box %d"
                   "and total number %d differ\n"\
                   ,i,nrmonA,nrmonB);
    exit(ERROR);
  }

}


/**************************************
 * elj_energie: calculates LJ energy    *
 *              of a single monomer     *
 *              "box" method            *
 *                                      *
 * ident = position of monomer in       *
 *         pos and poslat arrays        *
 * type  = A (=0) or B (=1)             *
 * mon   = monomer vector               *
 **************************************/

/*-------------------------------------------------------*
 | The program calculates the LJ interaction of particle  |
 | "mon" with all particles whose distance to "mon" <     |
 | interaction range. They are all located in a 3*3*3     |
 | box with the central subbox containing "mon" (size of  |
 | a subbox = interaction range).                         |
 *-------------------------------------------------------*/

double elj_energie(int ident, int type, MYVEC *mon) {
  int     ibox=0,i=0,j=0,type2=0;
  BBOX    *pbox1=NULL,*pbox2=NULL;
  ELEMENT *akt=NULL;
  MYVEC   *mon1=NULL,*mon2=NULL,diff,vhilf;
  double  rad2=0,rad6inv=0,rad6=0,ljenergie=0;

  ljenergie=0.0;
  mon1 = &vhilf;

  mon1->x = mon->x;          /* -> periodic coordinates */
  mon1->y = mon->y;
  mon1->z = mon->z;

  mon1->x -= LSX*((int)(mon1->x/LSX));
  mon1->y -= LSY*((int)(mon1->y/LSY));
  mon1->z -= LSZ*((int)(mon1->z/LSZ));

  if (mon1->x<0.0) mon1->x+=LSX;
  if (mon1->y<0.0) mon1->y+=LSY;
  if (mon1->z<0.0) mon1->z+=LSZ;


  ibox  = FBOXNR(mon1->x,mon1->y,mon1->z);
  /* determine box number of "mon" particle */

  pbox1 = boxliste+ibox;


  /*-------------------------------------------*
   | scan each of the 3*3*3 subboxes around "mon" |
   *-------------------------------------------*/

  for(i=0;i<27;i++) {

#ifdef HARDWALLS
    if ((pbox1->neighbours)[i]>=0) {
#endif

    pbox2 = (pbox1->neighbours)[i]+boxliste;
    akt = pbox2->first;


    /*-----------------------------------------------*
     | scan double-linked list which contains         |
     | possible interaction monomers in specified box |
     *-----------------------------------------------*/

    for(j=0;j<pbox2->population;j++,akt=akt->after){

      type2 = akt->type;
      /* determine interact. monomer type */

      if ((akt->monomer != ident) || (type2 != type)){
      /* exclude interaction of monomer with itself */

        if (type2==0)
          mon2 = poslatA + akt->monomer;
        else
           mon2 = poslatB + akt->monomer;

        /* determine distance between "mon" */
```

```
        /* and interaction particle        */
        diff.x = mon2->x - mon1->x;
        diff.y = mon2->y - mon1->y;
        diff.z = mon2->z - mon1->z;

        /* Minimum Image Convention */
        diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
        diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
        diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif

        rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);


        /*---------------------*
         | calculate LJ energy |
         *---------------------*/

        if ((type==0)&&(type2==0)){  /* AA interaction */
          /* check if distance < interaction range */
          if (rad2 < RANGE2AA) {
            rad6    = CUBE(rad2);
            rad6inv = 1.0/rad6;
            ljenergie += LENJAA(rad6inv);
          }
        }

        else if ((type==1)&&(type2==1)){  /* BB interaction */
          if (rad2 < RANGE2BB) {
            rad6    = CUBE(rad2);
            rad6inv = 1.0/rad6;
            ljenergie += LENJBB(rad6inv);
          }
        }

        else {  /* AB interaction */
                /* exclude interaction between particles of */
                /* different type at the same position      */
          if ((rad2 < RANGE2AB)) {
            rad6    = CUBE(rad2);
            rad6inv = 1.0/rad6;
            ljenergie += LENJAB(rad6inv);

          }
        } /* in WW range */
      }/* akt o.k. */
    }/* j */

#ifdef HARDWALLS
    }/* pbox2 o.k. */
#endif

  } /* i */

  return (4.0*ljenergie);
}


/*************************************
 * fene: calculates additional FENE-    *
 *       potential of an inserted       *
 *       monomer                        *
 *                                      *
 * ident = position of monomer in       *
 *         pos and poslat arrays        *
 * type  = A (=0) or B (=1)             *
 * neu   = monomer vector               *
 *************************************/

double fene(int ident, int type, MYVEC *neu)
{
int mnr;
double energie,rad2,pol;
MYVEC *mon,*pos;

  if (type==0) {
     pol = polA;
     pos = posA;
  }

  else{
     pol = polB;
     pos = posB;
  }

  if (pol==1) return(0.0);
  /* works also for the monomer system */

  energie=0.0;

  /* calculate monomer position in polymer */
  mnr = ident - pol*((int)((double)(ident)/((double)(pol))));
```

```
  mon = pos + ident;


  /*-----------------------*
   | calculate FENE energie |
   *-----------------------*/

  if (mnr == 0){               /* first monomer in chain */
     rad2 = DISTANCE2(neu,mon+1);
     if (rad2*RNULL2INV<1.0) energie = BOND(rad2);
     else     {              energie = T*INFENERGY; }
  }

  else if (mnr == (pol-1)) {   /* last monomer in chain */
     rad2 = DISTANCE2(neu,(mon-1));
     if (rad2*RNULL2INV<1.0) energie = BOND(rad2);
     else     {              energie = T*INFENERGY; }
  }

  else {                       /* monomer in middle of chain */
     rad2 = DISTANCE2(neu,(mon-1));
     if (rad2*RNULL2INV<1.0) energie = BOND(rad2);
     else     {              energie = T*INFENERGY; }

     rad2 = DISTANCE2(neu,(mon+1));
     if (rad2*RNULL2INV<1.0) energie += BOND(rad2);
     else     {              energie = T*INFENERGY; }
  }

  return(energie);

}


/************************************
 * eenergie: calculates total energy   *
 *           using functions defined in*
 *           element.c                 *
 *                                     *
 * remark:   also checks if monomers   *
 *           are in correct box        *
 ************************************/

double eenergie(void) {
  int i,j,ix,iy,iz,number=0;
  double evalue;
  MYVEC *mon;
  BBOX *pbox;
  ELEMENT *philf;


  /*--------------------------------------*
   | check if monomers are in correct box |
   *--------------------------------------*/

  for(ix=0;ix<NRSUBX;ix++)
    for(iy=0;iy<NRSUBY;iy++)
      for(iz=0;iz<NRSUBZ;iz++){
        pbox    = boxliste+BOXNR(ix,iy,iz);
        number += pbox->population;
        philf   = pbox->first;
        for(j=0;j<pbox->population;j++,philf=philf->after) {
          if (philf->type==0)  mon = poslatA+philf->monomer;
          else                 mon = poslatB+philf->monomer;
          if ((mon->x < (((double)(ix))*LSSUBX))        ||
              (mon->y < (((double)(iy))*LSSUBY))        ||
              (mon->z < (((double)(iz))*LSSUBZ))        ||
              (mon->x > (((double)(ix))*LSSUBX+LSSUBX)) ||
              (mon->y > (((double)(iy))*LSSUBY+LSSUBY)) ||
              (mon->z > (((double)(iz))*LSSUBZ+LSSUBZ))   )
          {fprintf(stderr,"ERROR: monomer number %d : "
                          "%d %d %d\n",philf->monomer,ix,iy,iz);
           VPRINT(mon);}
         }
      }


  /*--------------------------------------*
   | check if number of monomers in box = |
   | total number of monomers             |
   *--------------------------------------*/

  if (number != nrmonA+nrmonB)
     fprintf(stderr,"ERROR: found %d monomer in boxes, "
                    "should be %d\n",number,nrmonA+nrmonB);


  /*-----------------------*
   | calculate total energy |
   *-----------------------*/

  for(i=0,evalue=0;i<nrmonA;i++)
#ifdef HARDWALLS
```

```
    if (i<NPOLYBRUSH*polA)
      evalue += elj_energie(i,0,posA+i) +\
                fene(i,0,posA+i) + 2*wbenergy((posA+i) );
    else
      evalue += elj_energie(i,0,posA+i) + fene(i,0,posA+i) +
                2*wenergy( (posA+i) );
#else
      evalue += elj_energie(i,0,posA+i) + fene(i,0,posA+i);
#endif

  for(i=0;i<nrmonB;i++)
#ifdef HARDWALLS
    evalue += elj_energie(i,1,posB+i) + fene(i,1,posB+i) +
              2*wenergy( (posB+i) );
#else
    evalue += elj_energie(i,1,posB+i) + fene(i,1,posB+i);
#endif

  return (evalue/2); /* each interaction is counted twice ! */
}
```

Monte-Carlo-Routinen: local.c

```
/************************************
 * local.c: local MC move           *
 *          (tries to move monomers) *
 *                                   *
 * last modification: 23/05/2003     *
 ************************************/


#include "element.h"
#include "r250.h"
#include <math.h>
#include <stdio.h>

#define NRND_MAX     5*(NMONOMAXA+NMONOMAXB)
/* (choose monomer, 3 coordinates and metropolis)
   times (total number of monomers)              */

extern int       nrmonA, polA;
extern int       nrmonB, polB;
extern MYVEC     *posA, *poslatA;
extern MYVEC     *posB, *poslatB;
extern double    Tinv,T;
extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern double    cenergie;

#ifdef STRIPE
extern double    SEwall;
#endif


/****************************************
 * lmtrial: creates a trial vector for  *
 *          local MC moves              *
 *                                      *
 * alt = monomer vector to be substituted *
 * new = new trial vector coordinates   *
 *                                      *
 * rnd_ptr_adr = adress of random ptr   *
 * ident       = monomer position of 'alt' *
 *               in pos or poslat array *
 *             = A (=0) or B (=1)       *
 ****************************************/

void lmtrial(MYVEC *new, MYVEC *alt, double ** rnd_ptr_adr)
{
double *rnd_ptr;
double rad;
double ctheta,stheta,phi;

  rnd_ptr = *rnd_ptr_adr;
  /* get pointer to rnd number table */

  rad     = MAX_STEP*(*rnd_ptr++);
  /* determine distance from 'alt'   */

  /* determine arbitrary position on */
  /* sphere around 'alt'             */

  ctheta = 2*(*rnd_ptr++) - 1;
  stheta = 2*(*rnd_ptr++) - 1;

  phi    = ctheta*ctheta + stheta*stheta;
  while (phi >= 1 || phi == 0) {
    ctheta = 2*double_r250() - 1;
    stheta = 2*double_r250() - 1;
    phi    = ctheta*ctheta + stheta*stheta;
  }

  new->z = alt->z + rad*(1.0 - 2*phi);
```

```
  phi    = 2*sqrt(1-phi);
  rad   *= phi;
  new->x = alt->x + rad*ctheta;
  new->y = alt->y + rad*stheta;

  *rnd_ptr_adr = rnd_ptr;
  /* set pointer back */
}


/*****************************************
 * lmcmove: executes nrmonA+nrmonB       *
 *          local moves, monomers are    *
 *          chosen at random, fct. returns *
 *          number of accepted moves     *
 *****************************************/

int lmcmove(void){

static double *rndtab = NULL;
static double *rnd_ptr;
int    ident,box_new,box_old,i_mono,i_poly,acc,type,pol;
MYVEC  *alt,*neu,*mon,*pos,*poslat,neud;
double dE;


  /*----------------------------------------*
   | initialize or update random number array |
   *----------------------------------------*/

  if (rndtab==NULL) {
  /* first call -> initialization    */

        rndtab = (double*) calloc(NRND_MAX,sizeof(double));
        /* allocate memory for NRND_MAX #s */

        double_r250_vector(rndtab,NRND_MAX);
        /* generate all rndnumbers         */

        rnd_ptr = rndtab;
        /* set rnd_ptr to first position   */
     }

  acc = (int)(rnd_ptr-rndtab);
  if ( (acc<0)||(acc>NRND_MAX) ) {
    fprintf(stderr,"ERROR: must generate too many RNDs"
                   "in local.c (%d)\n",acc);
}

else {               /* update random number array      */

  double_r250_vector(rndtab,acc);
  /* refresh rnd #s which have already been used        */

  rnd_ptr = rndtab;  /* set rnd_ptr to first position   */
}

neu = &neud;


/*----------------------------------*
 | loop over nrmonA+nrmonB monomers |
 | which are chosen at random       |
 *----------------------------------*/

for (i_mono=0,acc=0;i_mono<nrmonA+nrmonB;i_mono++) {
   ident  = (int) ( ((double)(nrmonA+nrmonB))*(*rnd_ptr++) );
   /* choose monomer */


   /*----------------------------------------*
    | set variables according to monomer type |
    *----------------------------------------*/

   if (ident<nrmonA) {
     type   = 0;
     pol    = polA;
     pos    = posA;
     poslat = poslatA;
   }
   else {
     type   = 1;
     pol    = polB;
     pos    = posB;
     poslat = poslatB;
     ident  = ident-nrmonA;
   }
   i_poly = ident/pol;
   /* determine position of monomer in polymer */

   if ((i_poly >= NPOLYBRUSH) || (ident%pol > 0) ||
       (type==1) ) { /* not first monomer in a brush */
```

```
        /*--------------------------------------*
         | step 1: create trial vector 'neu' with |
         |         lmtrial-fct and calculate      |
         |         energy difference between old  |
         |         and new configuration          |
         *--------------------------------------*/

        alt = pos + ident;

        lmtrial(neu,alt,&rnd_ptr);
        /* generate trial vector 'neu' */

        dE  = fene(ident,type,neu)-fene(ident,type,alt);
        dE += elj_energie(ident,type,neu)-
              elj_energie(ident,type,alt);

#ifdef HARDWALLS
        if (i_poly>=NPOLYBRUSH) {
        /* monomer's not in a brush */

          dE += wenergy(neu)-wenergy(alt);
        }
        else {
          dE += wbenergy(neu)-wbenergy(alt);
        }
#endif


        /*----------------------*
         | step 2: local MC step |
         *----------------------*/

        if ( (*rnd_ptr++) < exp(Tinv*dE)) {


          /*---------------------------*
           | accepted ? -> update arrays |
           *---------------------------*/

          box_old = VBOXNR(poslat+ident);

          alt->x = neu->x;            /* update pos-array */
          alt->y = neu->y;
          alt->z = neu->z;

          mon = poslat+ident;

          mon->x = alt->x-LSX*((int)(alt->x/LSX));
          mon->y = alt->y-LSY*((int)(alt->y/LSY));
          mon->z = alt->z-LSZ*((int)(alt->z/LSZ));

          /* update poslat-array */
          if ( mon->x < 0 ) mon->x += LSX ;
          if ( mon->y < 0 ) mon->y += LSY ;
          if ( mon->z < 0 ) mon->z += LSZ ;

          box_new = VBOXNR(poslat+ident);

          /* update double-linked subbox list if necessary */
          if (box_new !=box_old) {
            if (type==0) {
              del_element(box_old,ident);
              put_element(box_new,ident);
            }
            else {
              del_element(box_old,NMONOMAXA+ident);
              put_element(box_new,NMONOMAXA+ident);
            }
          }

          cenergie += dE; /* update energy control variable  */
          acc++;          /* number of accepted moves        */
        }
      } /* not first monomer in brush */
    }/* i_mono */

    return(acc);
}


/* undefine local definitions */
#undef NRND_MAX
```

Monte-Carlo-Routinen: rep.c

```
/*********************************************
 * rep.c: reptation MC move                  *
 *                                           *
 * last modification: 30/08/2000             *
 *********************************************/
```

```
#include "element.h"
#include "r250.h"
#include <math.h>
#include <stdio.h>

#define NRND_MAX 5*(NPOLYMAXA+NPOLYMAXB)
/* (choose chain and monomer which will be cut off,         */
/* 2* create new monomer, MC step) * total number of chains */

extern int       nrchainsA, polA;
extern int       nrchainsB, polB;
extern MYVEC     *posA, *poslatA;
extern MYVEC     *posB, *poslatB;
extern double    Tinv,J;
extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern double    cenergie;


/***************************************
 * repatation: calls rep               *
 ***************************************/

int reptation(void) {
int rep_ctr=0;

  if (polA !=1)
     rep_ctr += rep(0);
  if (polB !=1)
     rep_ctr += rep(1);

  return (rep_ctr);
  /* returns # of succesful reptation moves */
}


/***************************************
 * rep: tries nrchains times to cut off *
 *      one end of a polymer and to     *
 *      attach it to the opposite side  *
 *                                      *
 * type: A(=0) or B(=1)                 *
 *                                      *
 * remark: no bondlengths are changed ! *
 ***************************************/

int rep(int type){


static double *rndtab = NULL;
static double *rnd_ptr;

int    rwert = 0;
int    kette, ident, start, stop, dx;
int    an, ab , acc, box_new,box_old;
int    i_poly,i,  ident2;
MYVEC  trials, *trial= &trials;
double energy_old,energy_new,ctheta,stheta,phi;
double prob, length;

int    nrchains, pol;
MYVEC  *pos, *poslat;

  /*---------------------------------------*
   | set variables according to monomer type |
   *---------------------------------------*/

  if (type==0) {
    pol      = polA;
    pos      = posA;
    poslat   = poslatA;
    nrchains = nrchainsA;

    if (nrchainsA >= NPOLYMAXA) {
      fprintf(stderr,"ERROR in rep.c: nrchainsA>=NPOLYMAXA\n");
      return(0);
    }
  }

  else if (type==1) {
    pol      = polB;
    pos      = posB;
    poslat   = poslatB;
    nrchains = nrchainsB;

    if (nrchainsB >= NPOLYMAXB) {
      fprintf(stderr,"ERROR in rep.c: nrchainsB>=NPOLYMAXB\n");
      return(0);
    }
  }

  else {
    fprintf(stderr,"Incorrect 'type' declaration in rep"
                   " - EXITING program ...\n");
    exit(1);
```

```
}


/*----------------------------------------*
 | initialize or update random number array |
 *----------------------------------------*/

if (rndtab==NULL) {  /* first call -> initialization */

  rndtab = (double*) calloc(NRND_MAX,sizeof(double));
  /* allocate memory for NRND_MAX #s */

  double_r250_vector(rndtab,NRND_MAX);
  /* assign random #s to array       */

  rnd_ptr = rndtab;
  /* set rnd_ptr to first position   */
}

acc = (int)(rnd_ptr-rndtab);
if ( (acc<0)||(acc>NRND_MAX) ) {
  fprintf(stderr,"ERROR: must generate too many RNDs"
                 "in rep.c (%d)\n",acc);
}

else {   /* update random number array      */

  double_r250_vector(rndtab,acc);
  /* refresh rnd #s which have already been used */

  rnd_ptr = rndtab;
  /* set rnd_ptr to first position */
}


/*----------------------------------------*
 | try nrchains reptation moves -         |
 | each time a polymer is chosen at random |
 *----------------------------------------*/

for (i_poly=NPOLYBRUSH;i_poly<nrchains;i_poly++) {

  /*-------------------------------------------------------*
   | step 1: choose chain and reptation monomer at random |
   *-------------------------------------------------------*/

  kette = NPOLYBRUSH +
          (int)((nrchains-NPOLYBRUSH) * (*rnd_ptr++) );
          /* choose chain */

ident = pol * kette;


/*--------------------------------------*
 | decide whether first or last monomer  |
 | of the chosen polymer will be cut off |
 *--------------------------------------*/

if ( (*rnd_ptr++) < 0.5){
  an     = ident;
  /* an = monomer on which a new monomer will be put  */

  ab     = ident + pol - 1;
  /* ab = monomer which will be cut off - */
  /* here last monomer of chain           */

  ident2 = ab - 1;
  dx     = -1;            /* compare with step 3 */
}
else {
  an     = ident + pol - 1;
  ab     = ident;
  /* first monomer of chain will be cut off */

  ident2 = ab + 1;
  dx     = 1;
}
start  = ab;
stop   = an;


/*-------------------------------------------------*
 | step 2: create a new monomer on opposite side of |
 |         chain- bondlength remains the same       |
 *-------------------------------------------------*/

length = DISTANCE(pos+ab,pos+ident2);
/* bondlength between "ab" and its neighbor =  */
/* bondlength between new monomer and "an"     */
/* -> function doesn't change bondlengths !    */

/* determine position of new monomer on sphere */
/* around "an" with radius = old bondlength    */
ctheta = 2*(*rnd_ptr++) - 1;
```

```
    stheta = 2*(*rnd_ptr++) - 1;

    phi    = ctheta*ctheta + stheta*stheta;
    while (phi >= 1 || phi == 0) {
      ctheta = 2*double_r250() - 1;
      stheta = 2*double_r250() - 1;
      phi    = ctheta*ctheta + stheta*stheta;
    }
    trial->z = (pos+an)->z + length*(1.0 - 2*phi);
    phi      = 2*sqrt(1-phi)*length;
    trial->x = (pos+an)->x + phi*ctheta;
    trial->y = (pos+an)->y + phi*stheta;

    energy_old  = elj_energie(ab,type,pos+ab);
    /* old energy at position ab without counting monomer ab */

    energy_new  = elj_energie(ab,type,trial);
    /* new energy at position trial without counting ab */

#ifdef HARDWALLS
    energy_old += wenergy((pos+ab));
    energy_new += wenergy(trial);
#endif


    /*--------------------------*
     | step 3: reptation MC step |
     *--------------------------*/

    prob = exp(Tinv*(energy_new-energy_old));
    if ( (prob>1) || ((*rnd_ptr++) <prob)) {
    /* MC move accepted ! */

      rwert++;


      /*--------------------------------------*
       | move along chain and copy monomers to |
       | new positions in chain                |
       *--------------------------------------*/

      for(i=start;i!=stop;i+=dx){
      /* copy old monomers to new positions */

        box_old = VBOXNR(poslat+i);
        VCOPY(pos+i+dx, pos+i);
        /* in pos array (cp pos+i+dx to pos+i)*/

        VCOPY(poslat+i+dx, poslat+i);
        /* in poslat array                    */

        box_new = VBOXNR(poslat+i);

        /* update double-linked subbox list if necessary */
        if (box_old!=box_new && type==0) {
          del_element(box_old, i);
          put_element(box_new, i);
        }
        else if (box_old!=box_new && type==1) {
          del_element(box_old, NMONOMAXA+i);
          put_element(box_new, NMONOMAXA+i);
        }
      }

      VCOPY(trial, pos+an);
      /* copy newly created monomer to first or    */
      /* last position in chain - update pos array */

      trial->x = trial->x - LSX*((int)(trial->x/LSX));
      trial->y = trial->y - LSY*((int)(trial->y/LSY));
      trial->z = trial->z - LSZ*((int)(trial->z/LSZ));

      if ( trial->x < 0 ) trial->x += LSX ;
      if ( trial->y < 0 ) trial->y += LSY ;
      if ( trial->z < 0 ) trial->z += LSZ ;

      box_old = VBOXNR(poslat+an);
      VCOPY(trial, poslat+an);       /* update poslat array */
      box_new = VBOXNR(poslat+an);

      /* update double-linked subbox list if necessary */
      if (box_old!=box_new && type==0) {
        del_element(box_old, an);
        put_element(box_new, an);
      }

      else if (box_old!=box_new && type==1) {
        del_element(box_old, NMONOMAXA+i);
        put_element(box_new, NMONOMAXA+i);
      }

      cenergie += (energy_new - energy_old);
      /* update energy control variable */
```

```
    }
  }
  return(rwert);
  /* return number of succesful reptation moves */
}
```

Monte-Carlo-Routinen: cbgc.c

```
/*************************************************************
 * cbgc.c: grand-canonical configurational-bias MC move      *
 *         (puts chains in box - deletes chains from box)    *
 *                                                           *
 * last modification: 23/05/2003                             *
 *************************************************************/


#include "element.h"
#include "r250.h"
#include <math.h>
#include <stdio.h>

#define NRND_MAX 10*((2*W_ALL+2)*pol+4)

extern int       nrchainsA, nrmonA, polA;
extern int       nrchainsB, nrmonB, polB;
extern MYVEC     *posA, *poslatA;
extern MYVEC     *posB, *poslatB;
extern double    Tinv,CPA,CPB;
extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern double    cenergie;
extern double    w[NPOLYMAXA+1][NPOLYMAXB+1];
extern ELEMENT   *element_liste;


/*******************************************
 * cbgc: attemps to either insert (50% prob)  *
 *       or delete (50% prob) a chain from    *
 *       simulation box                       *
 *                                            *
 * type: A (=0) or B (=1)                     *
 *******************************************/

double cbgc(int type) {
  int count=0;

  if (r250_random()&1)
    {
      if (nrchainsA< MAXA)
         count=insert(type);
         /* try to insert chain with 50% prob. */
    }

  else
    {
      if (nrchainsA> MINA)
         count=delete(type);
         /* try to delete chain with 50% prob. */
    }

  return(count);
}


/*******************************************
 * insert: attemps to insert a chain into     *
 *         simulation box                     *
 *                                            *
 * type:   A (=0) or B (=1)                   *
 *******************************************/

int insert(int type) {
  static double *rndtab = NULL;
  static double *rnd_ptr;
  static int    *box;
  static MYVEC  *new;
  double        ctheta,stheta,phi,length;

  double        energy_new,znew,x,y,z,prn;
                /* znew = Rosenbluth-weight */

  double        energy[W_ALL],prob[W_ALL+1];
                /* energy_new = energy of new chain */

  int           i_mono,i,acc,ident,i_chosen,i_break;
  MYVEC         *mono,trial[W_ALL];

  int           nrchains,pol;
  MYVEC         *pos, *poslat;
  double        CP;
  double        weight_factor;
```

```
 /* initialize arrays for insert and delete at first call */
if(rndtab==0) {
  new     = (MYVEC*) calloc(polA+polB,sizeof(MYVEC));
  box     = (int*)   calloc(polA+polB,sizeof(int));
  /* stores box number of created monomers  */
}

/*--------------------------------------*
 | set variables according to chain-type |
 *--------------------------------------*/

if (type==0) {
  nrchains = nrchainsA;
  pos      = posA;
  poslat   = poslatA;
  pol      = polA;
  CP       = CPA;

  if (nrchainsA >= NPOLYMAXA) {
    fprintf(stderr,"ERROR in cbgc.c: nrchainsA>="
                   "NPOLYMAXA\n");
    return(0);
  }
}

else if (type==1) {
  nrchains = nrchainsB;
  pos      = posB;
  poslat   = poslatB;
  pol      = polB;
  CP       = CPB;

  if (nrchainsB >= NPOLYMAXB) {
    fprintf(stderr,"ERROR in cbgc.c: nrchainsB>="
                   "NPOLYMAXB\n");
    return(0);
  }
}

else {
  fprintf(stderr,"Incorrect 'type' declaration in insert()."
                 "EXITING program ...\n");
  exit(1);
}
```

```
  /*--------------------------------------*
   | initialize or update random number array |
   *--------------------------------------*/

  if (rndtab==NULL) {    /* first call -> initialization    */

    rndtab = (double*) calloc(NRND_MAX,sizeof(double));
    /* allocate memory for NRND_MAX #s */

    double_r250_vector(rndtab,NRND_MAX);
    /* assign random #s to array       */

    rnd_ptr = rndtab;
    /* set rnd_ptr to first position   */
  }

  acc = (int)(rnd_ptr-rndtab);
  if ( (acc<0)||(acc>NRND_MAX) ) {
    fprintf(stderr,"ERROR: must generate too many RNDs "
                   "in cbgc.c (%d)\n",acc);
  }

  else {              /* update random number array    */
    double_r250_vector(rndtab,acc);
    /* refresh rnd #s which have already been used     */

    rnd_ptr = rndtab; /* set rnd_ptr to first position */
  }


  /*-----------------------------------------------*
   | step 1: create first monomer at random position |
   *-----------------------------------------------*/

  ident = nrchains*pol;
  /* assign first free position to first monomer */

  mono  = new;

  mono->x = LSX*(*rnd_ptr++); /* create at random position */
  mono->y = LSY*(*rnd_ptr++);
  mono->z = LSZ*(*rnd_ptr++);

  energy_new = elj_energie(ident,type,mono);
  /* update energy of new chain */

#ifdef HARDWALLS
```

```
  energy_new += wenergy(mono);
#endif

  znew    = exp(Tinv*energy_new);
  /* store Boltzmann-weight of first monomer          */

  VCOPY(mono,pos+ident);
  /* copy monomer vector to last pos. of pos-array    */

  VCOPY(mono,poslat+ident);
  /* copy monomer vector to last pos. of poslat-array */

  box[0]  = VBOXNR(poslat+ident);

  if (type==0) {

    /* update double-linked subbox list */
    put_element(box[0],ident);

    /* update elementliste-array        */
    (element_liste+ident)->monomer = ident;
    (element_liste+ident)->type    = 0;

    nrmonA++;    /* increment # of monomers by one    */
  }

  else {
    put_element(box[0],ident+NMONOMAXA);
    (element_liste+NMONOMAXA+ident)->monomer = ident;
    (element_liste+NMONOMAXA+ident)->type = 1;
    nrmonB++;
  }

  i_break=pol;


  /*---------------------------------*
   | step 2: create rest of the chain |
   *---------------------------------*/

  for (i_mono=1;i_mono<pol;i_mono++) {
  /* loop to create rest of the chain */
    length = choose_length(type,*rnd_ptr++);
    /* choose bondlength */
```

```
    /*-----------------------------------------*
     | 2.1.: create W_ALL trial-monomers on sphere |
     |       around first monomer                  |
     *-----------------------------------------*/

    for (i=0;i<W_ALL;i++) {

      /* determine trial-monomer pos. on sphere */
      /* trial-vectors are evenly distributed ! */
      ctheta = 2*(*rnd_ptr++) - 1;
      stheta = 2*(*rnd_ptr++) - 1;
      phi    = ctheta*ctheta + stheta*stheta;
      while (phi >= 1 || phi == 0) {
        ctheta = 2*double_r250() - 1;
        stheta = 2*double_r250() - 1;
        phi    = ctheta*ctheta + stheta*stheta;
      }
      (trial+i)->z = mono->z + length*(1.0 - 2*phi);
      phi    = 2*sqrt(1-phi)*length;
      (trial+i)->x = mono->x + phi*ctheta;
      (trial+i)->y = mono->y + phi*stheta;
      /* trial vectors are stored in trial[]    */
    }

    /* set up interaction list to speed up      */
    /* energy calculations                      */
    /* remark: start-monomer "mono" is not part */
    /* of the interaction list                  */
    make_list(ident,type,mono,RANGE+length+0.05);



    /*-------------------------------------------------*
     | 2.2.: choose one of the trial vectors according  |
     |       to its Boltzmann-weight                    |
     *-------------------------------------------------*/

    for (i=0,z=0;i<W_ALL;i++) {
      energy[i]  = lj_energy(type,trial+i);
      /* calculate LJ energy of each trial vector */

#ifdef HARDWALLS
      energy[i] += wenergy((trial+i));
#endif

      z         += exp(Tinv*energy[i]);
      /* calc. contribution to Rosenbluth-weight  */
```

```
  /* z is also used as a normalization factor */
  prob[i]    = z;
}

prob[W_ALL]=2;
/* always accept this W_ALL+1 possibility (z==0) ,*/
/* but write warning */

znew *= z;  /* update Rosenbluth-weight for MC step */

i_chosen = 0;


/*-----------------------------------------------------*
 | Each trial vector is assigned a probabilty of being |
 | chosen proportional to its Boltzmann-factor.        |
 | A random number is drawn and normalized. The while  |
 | loop determines to which trial vector the number    |
 | corresponds                                         |
 *-----------------------------------------------------*/

prn      = z*(*rnd_ptr++);
/* draw random number and normalize it */

while(prn>prob[i_chosen]) i_chosen++;
/* choose trial vector                */

if (i_chosen>=W_ALL) {
  fprintf(stderr,"ERROR: run out of trials => rnd==1 "
                 "or z=%lg  resolved to i_chosen=0 and "
                 "break\n",z);
  i_chosen = 0;
  znew     = 0;
  i_break  = i_mono;
  break;
}


/*---------------------*
 | 2.3.: update arrays |
 *---------------------*/

energy_new += energy[i_chosen]+bondenergy(type,length);
/* update energy of new chain */

mono++;
ident++;
```

```
  x = (trial+i_chosen)->x;
  y = (trial+i_chosen)->y;
  z = (trial+i_chosen)->z;

  mono->x           = x;
  mono->y           = y;
  mono->z           = z;

  (pos+ident)->x    = x;        /* update pos-array    */
  (pos+ident)->y    = y;
  (pos+ident)->z    = z;

  x = x - LSX*((int)(x/LSX));   /* update poslat-array */
  if(x<0) x += LSX;
  y = y - LSY*((int)(y/LSY));
  if(y<0) y += LSY;
  z = z - LSZ*((int)(z/LSZ));
  if(z<0) z += LSZ;

  (poslat+ident)->x = x;
  (poslat+ident)->y = y;
  (poslat+ident)->z = z;

  box[i_mono]       = VBOXNR(poslat+ident);

  if (type==0) {

    /* update double-linked subbox list */
    put_element(box[i_mono],ident);

    /* update element_liste             */
    (element_liste+ident)->monomer = ident;
    (element_liste+ident)->type    = 0;
    nrmonA++;      /* increment # of monomers by one   */
  }

  else {
    put_element(box[i_mono],ident+NMONOMAXA);
    (element_liste+NMONOMAXA+ident)->monomer = ident;
    (element_liste+NMONOMAXA+ident)->type = 1;
    nrmonB++;
  }

} /* a new chain has been created ! */
```

```
  /*----------------------*
   | step 3: cbgc MC step |
   *----------------------*/

  if (type==0)
    weight_factor=w[nrchainsA][nrchainsB]-\
                  w[nrchainsA+1][nrchainsB];
  else
    weight_factor=w[nrchainsA][nrchainsB]-\
                  w[nrchainsA][nrchainsB+1];


  if (znew*exp( -Tinv*CP + weight_factor )\
              /(nrchains+1) > (*rnd_ptr++) ) {
  /* MC move is accepted ! */

    cenergie += energy_new;
    if (type==0) nrchainsA++;
    else         nrchainsB++;

    return(1);
  }

  else {  /* MC move is rejected */
    for (i_mono=0,ident=nrchains*pol;i_mono<i_break;\
         i_mono++,ident++) {

    /* delete monomers from boxes */
    if (type==0) del_element(box[i_mono],ident);
    else         del_element(box[i_mono],ident+NMONOMAXA);
    }

    if (type==0) nrmonA -= i_break;
    else         nrmonB -= i_break;
    return(0);
  }

}


/*******************************************
 * delete: attemps to delete a chain from     *
 *         simulation box                     *
 *                                            *
 * type:   A (=0) or B (=1)                   *
 *******************************************/
```

```
int delete(int type) {
  static double *rndtab = NULL;
  static double *rnd_ptr;
  static int    *box;
  MYVEC         *mon0,trial[W_ALL];
  int           i_poly,i_mono,ident,ident2,box1,box2,i,acc;
  double        length,ctheta,stheta,phi,x,y,z,e;

  double        zold,energy_old;
  /* zold = Rosenbluth-weight                   */
  /* energy_old = energy of chain to be deleted */

  int           nrchains, pol;
  MYVEC         *pos, *poslat;
  double        CP;
  double        weight_factor;

  if(rndtab==0) {
    box     = (int*)calloc(polA+polB,sizeof(int));
    /* stores box number of created monomers  */
  }


  /*--------------------------------------*
   | set variables according to chain-type |
   *--------------------------------------*/

  if (type==0) {
    nrchains = nrchainsA;
    pos      = posA;
    poslat   = poslatA;
    pol      = polA;
    CP       = CPA;

    if (nrchains >= NPOLYMAXA) {
      fprintf(stderr,"ERROR in cbgc.c: nrchainsA=NPOLYMAXA\n");
      return(0);
    }
  }

  else if (type==1) {
    nrchains = nrchainsB;
    pos      = posB;
    poslat   = poslatB;
    pol      = polB;
    CP       = CPB;
```

```
  if (nrchains >= NPOLYMAXB) {
    fprintf(stderr,"ERROR in cbgc.c: nrchainsB=NPOLYMAXB\n");
    return(0);
  }
}

else {
  fprintf(stderr,"Incorrect 'type' declaration in delete(). "
                 "EXITING program ...\n");
  exit(1);
}

/*-----------------------------------------*
 | initialize or update random number array |
 *-----------------------------------------*/

if (rndtab==NULL) {  /* first call -> initialization    */

  rndtab = (double*) calloc(NRND_MAX,sizeof(double));
  /* allocate memory for NRND_MAX #s */

  double_r250_vector(rndtab,NRND_MAX);
  /* assign random #s to array       */

  rnd_ptr = rndtab;
  /* set rnd_ptr to first position   */
}

acc = (int)(rnd_ptr-rndtab);
if ( (acc<0)||(acc>NRND_MAX) ) {
  fprintf(stderr,"ERROR: must generate too many RNDs "
                 "in cbgc.c (%d)\n",acc);
}

else {    /* update random number array      */

  double_r250_vector(rndtab,acc);
  /* refresh rnd #s which have already been used  */

  rnd_ptr = rndtab;  /* set rnd_ptr to first position   */
}


/*---------------------------------------------*
 | step 1.1: choose polymer at random which fct |
 |           will attempt to delete             |
 *---------------------------------------------*/
```

```
if (nrchains==NPOLYBRUSH) return(0);
else i_poly = NPOLYBRUSH+\
              r250_random()%(nrchains-NPOLYBRUSH);
ident = i_poly*pol;
/* chosen polymer's first monomer position */
/* in pos and poslat arrays                */


/*-------------------------------------------------------*
 | step 1.2: swap positions with last polymer:           |
 |           newly inserted polymers are always stored   |
 |           at the end of pos, poslat and elementliste- |
 |           arrays                                      |
 |            -> chosen polymer must be swaped with      |
 |               last polymer                            |
 *-------------------------------------------------------*/

if (i_poly!=nrchains-1) {
/* chosen polymer is not last polymer !       */

  ident2 = (nrchains-1)*pol;
  /* position of last polymer in pos and poslat */

  for (i_mono=0;i_mono<pol;i_mono++) {
  /* swap positions of all monomers with        */
  /*  monomers of last polymer                  */

     box1 = VBOXNR(poslat+(ident +i_mono));
     box2 = VBOXNR(poslat+(ident2+i_mono));


     if (type==0) {
     /* swap positions in double-linked subbox lists */
     del_element(box1,ident+ i_mono);
     put_element(box2,ident+ i_mono);
     del_element(box2,ident2+i_mono);
     put_element(box1,ident2+i_mono);
     }

     else {
     del_element(box1,ident+ i_mono + NMONOMAXA);
     put_element(box2,ident+ i_mono + NMONOMAXA);
     del_element(box2,ident2+i_mono + NMONOMAXA);
     put_element(box1,ident2+i_mono + NMONOMAXA);
     }
```

```
    x = (pos+(ident+ i_mono))->x;
    /* swap positions in pos-array */
    (pos+(ident+ i_mono))->x = (pos+(ident2+i_mono))->x;
    (pos+(ident2+i_mono))->x = x;
    y                        = (pos+(ident+ i_mono))->y;
    (pos+(ident+ i_mono))->y = (pos+(ident2+i_mono))->y;
    (pos+(ident2+i_mono))->y = y;
    z                        = (pos+(ident+ i_mono))->z;
    (pos+(ident+ i_mono))->z = (pos+(ident2+i_mono))->z;
    (pos+(ident2+i_mono))->z = z;

    /* swap positions in poslat-array */
    x = (poslat+(ident +i_mono))->x;
    (poslat+(ident+ i_mono))->x =
           (poslat+(ident2+i_mono))->x;
    (poslat+(ident2+i_mono))->x = x;

    y =(poslat+(ident+ i_mono))->y;
    (poslat+(ident+ i_mono))->y =
           (poslat+(ident2+i_mono))->y;
    (poslat+(ident2+i_mono))->y = y;

    z = (poslat+(ident+ i_mono))->z;
    (poslat+(ident+ i_mono))->z =
           (poslat+(ident2+i_mono))->z;
    (poslat+(ident2+i_mono))->z = z;

  } /* swap process succesful ! */
} /* chosen polymer is not last polymer !*/


/*----------------------------*
 | step 2: delete chosen chain |
 *----------------------------*/

ident = (nrchains-1)*pol;
zold=1;
energy_old=0;

/* delete all monomers of chain except first monomer  */
for (i_mono=pol-2;i_mono>=0;i_mono--) {
  box[i_mono+1] = VBOXNR(poslat+(ident+i_mono+1));

  if (type==0) {
  del_element(box[i_mono+1],(ident+i_mono+1));
  /* update double-linked subbox list */
  nrmonA--;
  /* decrement # of monomers by one   */
  }

  else {
  del_element(box[i_mono+1],(ident+i_mono+1+NMONOMAXA));
  nrmonB--;
  }

  mon0   = pos+(ident+i_mono);

  length =  DISTANCE(mon0,(mon0+1));
  /* determine bondlength */

  energy_old += bondenergy(type,length);
  /* update total energy  */

  VCOPY(mon0+1,trial);
  /* copy monomer vector to first position of trial array */


  /* create W_ALL-1 trial vectors   */
  /* trial vectors are evenly       */
  /* distributed on a sphere around */
  /* mon0                           */
  /* trial spheres are created for  */
  /* all monomers except last !     */

  for (i=1;i<W_ALL;i++) {
     ctheta = 2*(*rnd_ptr++) - 1;
     stheta = 2*(*rnd_ptr++) - 1;
     phi    = ctheta*ctheta + stheta*stheta;
     while (phi >= 1 || phi == 0) {
       ctheta = 2*double_r250() - 1;
       stheta = 2*double_r250() - 1;
       phi    = ctheta*ctheta + stheta*stheta;
     }
    (trial+i)->z = mon0->z + length*(1.0 - 2*phi);
    phi    = 2*sqrt(1-phi)*length;
    (trial+i)->x = mon0->x + phi*ctheta;
    (trial+i)->y = mon0->y + phi*stheta;
  }

  make_list(ident+i_mono,type,mon0,RANGE+length+0.05);
  /* set up interaction list         */
  /* to speed up energy calculations */

  for (i=0,z=0;i<W_ALL;i++) {
```

```
      e  = lj_energy(type,trial+i);
     /* calc. LJ energy of each trial vector */

#ifdef HARDWALLS
      e += wenergy((trial+i));
#endif

      if (i==0) {
      /* monomer belongs to chosen chain  */

      energy_old += e;  /* add energy to total energy       */
      }

      z += exp(Tinv*e);
    }

    zold *= z;  /* update Rosenbluth-weight */
  }/* all but first monomer have been deleted */

  if (type==0) nrmonA--;
  else         nrmonB--;

  e = elj_energie(ident,type,poslat+ident);
  /* calc. energy of first monomer */

#ifdef HARDWALLS
  e += wenergy((poslat+ident));
#endif

  energy_old += e;     /* total energy of removed chain */
  zold *=exp(Tinv*e);  /* update Rosenbluth-weight      */
  box[0] = VBOXNR(poslat+(ident));

  /* delete first monomer of chain */
  if (type==0)  del_element(box[0],ident);
  else          del_element(box[0],ident+NMONOMAXA);


  /*----------------------*
   | step 3: cbgc MC step |
   *----------------------*/

  if (type==0)
    weight_factor=w[nrchainsA][nrchainsB]-\
                  w[nrchainsA-1][nrchainsB];
  else
    weight_factor=w[nrchainsA][nrchainsB]-\
                  w[nrchainsA][nrchainsB-1];

  if ( nrchains/(zold*exp(-Tinv*CP - weight_factor) )\
       > (*rnd_ptr++) ) {   /* MC move is accepted */

    cenergie -= energy_old;
    if(type==0)  nrchainsA--;
    else         nrchainsB--;
    return(1);
  }

  else {  /* MC move is rejected */
    for (i_mono=0;i_mono<pol;i_mono++) {
      if (type==0)  put_element(box[i_mono],ident+i_mono);
      else  put_element(box[i_mono],ident+i_mono+NMONOMAXA);
    }
    if (type==0)  nrmonA+=pol;
    else          nrmonB+=pol;
    return(0);
  }

}/* delete */
```

Hilfsroutinen zu cbgc.c: mcmove.c

```
/*********************************************
 * mcmove.c: defines help-functions for cbgc.c *
 *                                             *
 * last modification : 23/05/2003              *
 *********************************************/


#include "element.h"
#include "r250.h"
#include <math.h>
#include <stdio.h>


/* substitution macros for choose_length() */

#define XSTART       (0.85)  /* minimum bondlength */
#define XEND         (1.15)  /* maximum bondlength */
#define MAX_INTEGRAL 1000    /* # steps for integral calc. */

#define DX           ((XEND-XSTART)/MAX_INTEGRAL)
```

```
/* step-size for integral calc. */

extern int       nrmonA;
extern int       nrmonB;
extern MYVEC     *poslatA;
extern MYVEC     *poslatB;
extern double    Tinv,J;
extern double    LSX,LSY,LSZ,LSSUBX,LSSUBY,LSSUBZ;
extern BBOX      boxliste[NRSUB3];

/* rtable[i] = (XSTART+0.5*DX) + DX*i */
static double rtableA[MAX_INTEGRAL+5], rtableB[MAX_INTEGRAL+5];

/* itable[] stores area below P(l) from l=0 to rtable[i] */
static double itableA[MAX_INTEGRAL+5], itableB[MAX_INTEGRAL+5];

static int       wwlisteA[MAX_ANZAHL];
/* interaction list for type A monomers */
static int       wwlistA;

static int       wwlisteB[MAX_ANZAHL];
/* interaction list for type B monomers */
static int       wwlistB;


/*********************************************
 * bondenergy: calculates  LJ+FENE energy of    *
 *             two monomers (same type) at      *
 *             given distance                   *
 *                                              *
 * type        =  A (=0) or B (=1)              *
 * length      =  distance between monomers     *
 *********************************************/

double bondenergy(int type, double length) {
  double benergie = 0;
  double rad2,rad6,rad6inv;

  rad2 = SQUARE(length);

  /* calc. LJ-energy for type A monomers */
  if (type==0 && rad2 < RANGE2AA) {
    rad6      = CUBE(rad2);
    rad6inv   = 1.0/rad6;
    benergie += 4.0*LENJAA(rad6inv);
  }

  /* calc. LJ-energy for type B monomers */
  else if (type==1 && rad2 < RANGE2BB) {
    rad6      = CUBE(rad2);
    rad6inv   = 1.0/rad6;
    benergie += 4.0*LENJBB(rad6inv);
  }

  else if (type!=0 && type!=1) {
  fprintf(stderr,"Wrong 'type' declaration in bondenergy() "
                 "... EXITING program\n");
  exit(1);
  }

  benergie += BOND(rad2);  /* calc. FENE-energy */

  return benergie;

}


/*********************************************
 * calcIntegral: calculates area below         *
 *               probability(length) curve,    *
 *               stores data in arrays         *
 *               rtable[i] (=center of ith x-  *
 *               interval) and itable[i] (area *
 *               from 0 to interval i)         *
 *********************************************/

double calcIntegral(double *flaecheA, double *flaecheB) {
  int i;
  double f0,f1,x;
  double trapez,flaeche;


  /*-------------------------------*
   | update rtableA[] and itableA[] |
   *-------------------------------*/

  x       = XSTART;
  flaeche = 0.0;
  f0      = x*x*exp(Tinv*bondenergy(0,x));

  for (i=0;i<MAX_INTEGRAL;i++) {

    x += DX;    /* increment length by DX    */
    f1 = x*x*exp(Tinv*bondenergy(0,x));
```

```
  /* is proportional to P(l) ! */

  trapez    = (0.5*DX)*(f0+f1);
  /* remark: P(l) is prop. to l^2 * Boltzmann-factor */

  flaeche  += trapez;
  rtableA[i] = x + (0.5*DX); /* save length */

  itableA[i] = flaeche;
  /* save area up to this length */

  f0 = f1;
}

for(i=0;i<MAX_INTEGRAL;i++)  /* normalize area to one */
  itableA[i] /= flaeche;

*flaecheA = (flaeche/(10*DX));
/* division by DX is not necessary ! - shifts CP */

/*--------------------------------*
 | update rtableB[] and itableB[] |
 *--------------------------------*/

x       = XSTART;
flaeche = 0.0;
f0      = x*x*exp(Tinv*bondenergy(1,x));

for (i=0;i<MAX_INTEGRAL;i++) {

  x += DX;  /* increment length by DX    */
  f1 = x*x*exp(Tinv*bondenergy(1,x));
  /* is proportional to P(l) ! */

  trapez    = (0.5*DX)*(f0+f1);
  /* remark: P(l) is prop. to l^2 * Boltzmann-factor */

  flaeche  += trapez;
  rtableB[i] = x + (0.5*DX); /* save length */

  itableB[i] = flaeche;
  /* save area up to this length */

  f0 = f1;
}

for(i=0;i<MAX_INTEGRAL;i++)  /* normalize area to one */
    itableB[i] /= flaeche;

  *flaecheB = (flaeche/(10*DX));

}


/*********************************************
 * choose_length: chooses bondlength such that *
 *                length is P(l) distributed   *
 *                                             *
 * type           = A (=0) or B (=1)           *
 * rnd            = random number (0..1)       *
 *********************************************/

/*-------------------------------------------------------*
 | The normalized area from l=l1 to l=l2 below the       |
 | probability distribution P(length) is equal to the    |
 | probability of choosing l in interval [l1,l2] when l is |
 | P(l)- distributed.                                    |
 *-------------------------------------------------------*/

double choose_length(int type, double rnd) {
  int i;

#ifdef FIXLENGTH
  return(FIXLENGTH);
#else

  if (type==0) {   /* choose bondlength for type A polymer */
    i=0;

    /* -> an interval is chosen according */
    /* to its area below P(l)             */

    while (rnd>itableA[i]) {
      i++;
    }

    return rtableA[i];  /* return chosen bondlength */
  }


  else if (type==1) {
  /* choose bondlength for type B polymer */

    i=0;
```

```
    while (rnd>itableB[i]) {
      i++;
    }

    return rtableB[i];
  }

  else {
  fprintf(stderr,"Wrong 'type' declaration in "
                 "choose_bondlength() ... EXITING program\n");
  exit(1);
  }
#endif

}


/*********************************************
 * make_list: generates interaction list by     *
 *            saving the interaction monomers'  *
 *            type and position (in pos-arrays) *
 *            in wwliste-arrays                 *
 *                                              *
 * ident1   = position of monomer (in posA,B)   *
 *            to which wwliste refers           *
 * type     = A (=0) or B (=1)                  *
 * ort      = vector of monomer to which        *
 *            wwliste refers                    *
 * radius   = interaction radius                *
 *********************************************/

/*********************************************************
 * The function exists in two versions. The first version *
 * is used for systems with a simulation box size of      *
 * 5*5*5 or greater ( -> LARGE is defined). The second    *
 * version is intended for smaller systems ( -> LARGE is  *
 * not defined). It creates an interaction list for all   *
 * W_ALL trial vectors in cbgc.c . These vectors are      *
 * located on a sphere around monomer "ort" and have an   *
 * interaction range determined by RANGE. To consider all *
 * vectors on the sphere the total interaction radius     *
 * adds up to (radius of the sphere + RANGE) <= 2*size of *
 * a subbox. Therefore all possible interaction partners  *
 * are located in a 5*5*5 box around the box which        *
 * contains "ort".                                        *
 *                                                        *
 * 1. The first version loops over all 5*5*5 boxes around *
 *    the central box which contains "ort". It is only    *
 *    possible for simulation boxes >= 5*5*5. For large   *
 *    systems this method will speed up the program       *
 *    considerably.                                       *
 * 2. The second version puts all monomers (!) in the     *
 *    wwliste array. This is the only correct method for  *
 *    system sizes < 5*5*5.                               *
 *********************************************************/


/*---------------*
 | first version |
 *---------------*/

# ifdef LARGE
void make_list(int ident1, int type, MYVEC *ort, double radius)
{
  int i,j,i_box1,i_box2;
  MYVEC ortp,dist,*mon,*poslat;
  double rad2;
  double wwrad2;
  ELEMENT *akt;
  BBOX *p_box1,*p_box2;

  wwlistA = 0;
  wwlistB = 0;
  wwrad2  = SQUARE(radius);

  if (radius>1.732*RANGE)
    fprintf(stderr,"ERROR %g %g\n", radius,RANGE);

  /*--------------------------------*
   | calculate periodic coordinates |
   *--------------------------------*/

  /* "ort" may be located outside the sim. box */
  /* -> put monomer back in box !              */

  ortp.x = ort->x - LSX*((int)(ort->x/LSX));
  ortp.y = ort->y - LSY*((int)(ort->y/LSY));
  ortp.z = ort->z - LSZ*((int)(ort->z/LSZ));

  if ( ortp.x < 0 ) ortp.x += LSX ;
  if ( ortp.y < 0 ) ortp.y += LSY ;
  if ( ortp.z < 0 ) ortp.z += LSZ ;
```

```
  /*--------------------------*
   | generate interaction list |
   *--------------------------*/

  i_box1 = FBOXNR(ortp.x,ortp.y,ortp.z);
  /* determine box number of "ort" */

  p_box1 = boxliste+i_box1;

  for (i=0;i<125;i++) {
  /* loop over inner and outer neighbor-boxes            */
  /* and the box itself (= 5*5*5 = 125 boxes altogether) */

    i_box2 = p_box1->neighbours[i];  /* determine box    */

#ifdef HARDWALLS
    if (i_box2>=0) {
#endif

    p_box2 = boxliste + i_box2;
    akt    = p_box2->first;

    for (j=0;j<p_box2->population;j++,akt=akt->after) {
      /* loop over double-linked list which */
      /* contains all monomers in box       */

      if ((akt->monomer != ident1) || (type!=akt->type)) {
      /* "ort" is not part of the list */

        if (akt->type==0) {
          poslat = poslatA;
        }
        else {
          poslat = poslatB;
        }

        mon = poslat + akt->monomer;

        dist.x = ortp.x - mon->x;
        dist.y = ortp.y - mon->y;
        dist.z = ortp.z - mon->z;

         /* minimum image convention */
        dist.x = dist.x - LSX*((int)(2.0*dist.x/LSX));
        dist.y = dist.y - LSY*((int)(2.0*dist.y/LSY));

#ifndef HARDWALLS
        dist.z = dist.z - LSZ*((int)(2.0*dist.z/LSZ));
#endif

        rad2 = SQUARE(dist.x)+SQUARE(dist.y) + SQUARE(dist.z);

        if (rad2<wwrad2){      /* check if distance <        */
          if (akt->type==0) {  /* total interaction radius   */
            wwlisteA[wwlistA] = akt->monomer;  /* and type=A */
            wwlistA++;  /* save monomer-position in wwlisteA */
          }
          else {
            wwlisteB[wwlistB] = akt->monomer;
            wwlistB++;
          }
        }
      }/* monomer "ort" excluded */
    }/* double-linked list o.k. */

#ifdef HARDWALLS
  }/* i_box2 o.k. */
#endif

  }/* loop over all neighboring boxes o.k. */
}


/*----------------*
 | second version |
 *----------------*/

#else
void make_list(int ident1, int type, MYVEC *ort,double radius)
{
  int i;
  MYVEC ortp,dist,*mon;
  double rad2,rad6,rad6inv;
  double wwrad2;

  wwlistA = 0;
  wwlistB = 0;
  wwrad2  = SQUARE(radius);


  /*--------------------------------*
   | calculate periodic coordinates |
   *--------------------------------*/
```

```
  ortp.x = ort->x - LSX*((int)(ort->x/LSX));
  ortp.y = ort->y - LSY*((int)(ort->y/LSY));
  ortp.z = ort->z - LSZ*((int)(ort->z/LSZ));

  if ( ortp.x < 0 ) ortp.x += LSX ;
  if ( ortp.y < 0 ) ortp.y += LSY ;
  if ( ortp.z < 0 ) ortp.z += LSZ ;


  /*-------------------------------------------------------*
   | generate interaction list: type A interaction monomers |
   *-------------------------------------------------------*/

  mon = poslatA;
  for(i=0;i<nrmonA;i++,mon++){
    if( (type==1) || (type==0 && i!=ident1) ) {

      dist.x = ortp.x - mon->x;
      dist.y = ortp.y - mon->y;
      dist.z = ortp.z - mon->z;

      /* minimum image convention */
      dist.x = dist.x - LSX*((int)(2.0*dist.x/LSX));
      dist.y = dist.y - LSY*((int)(2.0*dist.y/LSY));

#ifndef HARDWALLS
      dist.z = dist.z - LSZ*((int)(2.0*dist.z/LSZ));
#endif

      rad2 = SQUARE(dist.x) + SQUARE(dist.y) + SQUARE(dist.z);
      if (rad2<wwrad2){
        wwlisteA[wwlistA] = i;
        wwlistA++;
      }
    }
  }


  /*-------------------------------------------------------*
   | generate interaction list: type B interaction monomers |
   *-------------------------------------------------------*/

  mon = poslatB;
  for(i=0;i<nrmonB;i++,mon++){
    if( (type==0) || (type==1 && i!=ident1) ) {

      dist.x = ortp.x - mon->x;
      dist.y = ortp.y - mon->y;
      dist.z = ortp.z - mon->z;

      /* minimum image convention */
      dist.x = dist.x - LSX*((int)(2.0*dist.x/LSX));
      dist.y = dist.y - LSY*((int)(2.0*dist.y/LSY));

#ifndef HARDWALLS
      dist.z = dist.z - LSZ*((int)(2.0*dist.z/LSZ));
#endif

      rad2 = SQUARE(dist.x) + SQUARE(dist.y) + SQUARE(dist.z);
      if (rad2<wwrad2){
        wwlisteB[wwlistB] = i;
        wwlistB++;
      }
    }
  }
}
#endif


/*********************************************
 * lj_energy: calculates LJ interaction of      *
 *            monomer "ort" with all monomers   *
 *            in interaction lists "wwlisteA"+B *
 *                                              *
 * type       =  A (=0) or B (=1)               *
 * ort        =  monomer vector                 *
 *********************************************/

double lj_energy(int type, MYVEC *ort) {
  int i;
  MYVEC ortp,dist,*mon;
  double evalue,rad2,rad6,rad6inv;

  evalue = 0.0;

  /*---------------------------------------*
   | calculate periodic coordinates of "ort" |
   *---------------------------------------*/

  ortp.x = ort->x - LSX*((int)(ort->x/LSX));
  ortp.y = ort->y - LSY*((int)(ort->y/LSY));
  ortp.z = ort->z - LSZ*((int)(ort->z/LSZ));

  if ( ortp.x < 0 ) ortp.x += LSX ;
```

```
  if ( ortp.y < 0 ) ortp.y += LSY ;
  if ( ortp.z < 0 ) ortp.z += LSZ ;


  /*---------------------------------------*
   | calculate interaction of monomer "ort" |
   | with type A monomers in wwlisteA       |
   *---------------------------------------*/

  for(i=0;i<wwlistA;i++){
  /* loop over all monomers in wwlisteA */

    /* calculate distance between "ort"   */
    /* and monomer from wwlisteA          */
    mon = poslatA + wwlisteA[i];

    dist.x = ortp.x - mon->x;
    dist.y = ortp.y - mon->y;
    dist.z = ortp.z - mon->z;

    /* minimum image convention */
    dist.x = dist.x - LSX*((int)(2.0*dist.x/LSX));
    dist.y = dist.y - LSY*((int)(2.0*dist.y/LSY));

#ifndef HARDWALLS
    dist.z = dist.z - LSZ*((int)(2.0*dist.z/LSZ));
#endif

    rad2 = SQUARE(dist.x) + SQUARE(dist.y) + SQUARE(dist.z);

    if (type==0) {  /* monomer "ort" is of type A */

      if (rad2<RANGE2AA){
      /* check if distance < interaction range */
      /* ok -> calculate AA interaction        */

        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        evalue += LENJAA(rad6inv);
      }
    }

    else {  /* monomer "ort" is of type B           */

      if (rad2<RANGE2AB){
      /* check if distance < interaction range */
      /* ok -> calculate AB interaction        */

        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        evalue += LENJAB(rad6inv);
      }
    }
  }


  /*---------------------------------------*
   | calculate interaction of monomer "ort" |
   | with type B monomers in wwlisteB       |
   *---------------------------------------*/

  for(i=0;i<wwlistB;i++){
  /* loop over all monomers in wwlisteB */

    mon = poslatB + wwlisteB[i];

    dist.x = ortp.x - mon->x;
    dist.y = ortp.y - mon->y;
    dist.z = ortp.z - mon->z;

    /* minimum image convention */
    dist.x = dist.x - LSX*((int)(2.0*dist.x/LSX));
    dist.y = dist.y - LSY*((int)(2.0*dist.y/LSY));

#ifndef HARDWALLS
    dist.z = dist.z - LSZ*((int)(2.0*dist.z/LSZ));
#endif

    rad2 = SQUARE(dist.x) + SQUARE(dist.y) + SQUARE(dist.z);

    if (type==0) {                  /* AB interaction */
      if (rad2<RANGE2AB){
        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        evalue += LENJAB(rad6inv);
      }
    }
    else {                          /* BB interaction */
      if (rad2<RANGE2BB){
        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        evalue += LENJBB(rad6inv);
      }
    }
```

```
  }
  evalue *= 4.0;

  return(evalue);

}
```

Analyseroutinen: wach.c

```
/*******************************************************
 * wach.c: analysis fcts-update data-array before output *
 *         file "histo.dat" is written                   *
 *                                                       *
 * WARNING : wall routines have not been adjusted !      *
 * last modification: 23/05/2003                         *
 *******************************************************/


#include "element.h"
#include <math.h>
#include <stdio.h>

extern int       nrchainsA, nrmonA, polA;
extern MYVEC     *posA, *poslatA, *cm0A;
extern int       nrchainsB, nrmonB, polB;
extern MYVEC     *posB, *poslatB, *cm0B;
extern double    w[NPOLYMAXA+1][NPOLYMAXB+1];

extern double    LSX,LSY,LSZ;
extern double    J,Ewall;
extern long      _MPP_MY_PE,_MPP_N_PES;
extern DATA      data;
extern double    Jwallwtab[JwallN_MAX];
extern int       switch_iwall;



/***************************************
 * analyse: updates data array by calling *
 *          most analysis functions,      *
 *          updates "histo.dat" output    *
 *          file                          *
 ***************************************/

void analyse(int zeit, int lauf){
```

```
/* zeit = number of MC steps */

  FILE   *datei;
  char   fname[80];
  double weightA, weightB;


  /*-------------------------*
   | call analysis functions |
   *-------------------------*/

  bondwinkel();        /* calc bondangles                   */
  bondlength();        /* bondlength, b**2, bmax            */
  calc_polymer();      /* end-to-end and radius of gyration */

  calc_nrchainsleft();
  /* calc number of chains in the left half of the sim. box */

  calc_msd();          /* calc mean square displacement     */
  calc_energy();       /* calc energy                       */

#ifdef PRESSURE
  calc_p();            /* calc pressure via virial          */
#endif

#ifdef SWITCH
  weight=Jwallwtab[switch_iwall];
#else
  weightA=0;
  weightB=0;
#endif

# ifndef T3D
  sprintf(fname,"histo.dat");
# else
  sprintf(fname,"histo.dat_%d",_MPP_MY_PE);
# endif

  if ((datei=fopen(fname,"a"))==NULL) {
    fprintf(stderr,"ERROR: Couldn't open %s\n",fname);
    exit(ERROR); }


  /*-----------------------------*
   | write results to "histo.dat" |
   *-----------------------------*/
```

```
  /* fprintf(datei,"%d %d %d %g %g %g %g %g\n",
          zeit,nrchainsA,nrchainsB,data.energy_ljA,
          data.energy_ljB,data.energy_ljAB,data.energy_feneA,
          data.energy_feneB); */

  fprintf(datei,"%d %d %d %g %g %g %g %g %g %g %g %g %g %g %g"
                "\n",zeit,nrchainsA,nrchainsB,data.energy_ljA,
                data.energy_ljB,data.energy_ljAB,
                data.energy_feneA,data.energy_feneB,data.px,
                data.py,data.pz,data.virialx,data.virialy,
                data.virialz,w[nrchainsA][nrchainsB]);

  fclose(datei);

  # ifndef T3D
    sprintf(fname,"analyse");
  # else
    sprintf(fname,"analyse_%d",_MPP_MY_PE);
  # endif

  if ((datei=fopen(fname,"a"))==NULL) {
     fprintf(stderr,"ERROR: Couldn't open %s\n",fname);
     exit(ERROR); }

  fprintf(datei,"%d %d %d %g %g %g %g %g %g\n",
                zeit,nrchainsA,nrchainsB,data.bondangleA,
                data.bondlength1A,data.endA,
                data.gyrA,data.msdA,
                w[nrchainsA][nrchainsB]);

  fclose(datei);

}


/****************************************
 * bondwinkel: calculates cosine of     *
 *             bond-angle and averages  *
 *             over all bonds           *
 ****************************************/

void bondwinkel() {
  int i,j;
  double awinkel;
  MYVEC *mon;
  MYVEC bond[2];


/*------------------------------------------------*
 | calculate average cos(bondangle) for polymer type A |
 *------------------------------------------------*/

awinkel=0;

for(i=0;i<nrchainsA;i++){

  mon=posA+i*polA;

  for(j=0;j<(polA-2);j++,mon++) {
    DIFF(mon+1,mon,bond);
    DIFF(mon+1,mon+2,bond+1);
    awinkel+= CWINKEL(bond,bond+1);
    /* cos(bondangle) ! - see def. element.h */
  }
}

if (nrchainsA>0)
  awinkel /= ((double)(nrmonA-2*nrchainsA));
else
  awinkel=0;

/* update data array */
data.bondangleA = awinkel;


/*------------------------------------------------*
 | calculate average cos(bondangle) for polymer type B |
 *------------------------------------------------*/

awinkel=0;

for(i=0;i<nrchainsB;i++){

  mon=posB+i*polB;

  for(j=0;j<(polB-2);j++,mon++) {
    DIFF(mon+1,mon,bond);
    DIFF(mon+1,mon+2,bond+1);
    awinkel+= CWINKEL(bond,bond+1);
  }
}

if (nrchainsB>0)
  awinkel /= ((double)(nrmonB-2*nrchainsB));
```

```
  else
    awinkel=0;

  data.bondangleB = awinkel;

}


/****************************************
 * bondlength: calculates av.bondlenght, *
 *             b**2, bmax                *
 ****************************************/

void bondlength() {
  int i,j;
  double length,abslength,maxlength,distance;
  MYVEC *mon;


  /*--------------------------------------*
   | calculate average bondlength, b**2 and |
   | bmax for type A polymer               |
   *--------------------------------------*/

  length=0;    /* average bondlength */
  abslength=0; /* (average bondlength)^2 */
  maxlength=0; /* maximum bondlength */

  for(i=0;i<nrchainsA;i++){

    mon=posA+i*polA;

    for(j=0;j<(polA-1);j++,mon++) {

      distance   = DIST(mon,mon+1);
      length    += (distance);
      abslength += SQUARE(distance);
      if (ABS(distance)>maxlength) maxlength=ABS(distance);

      if (ABS(distance)>2.0) {
      /* check if chain is broken */

        fprintf(stderr,"WARNING: typeA: chain %d mon "
                       "%i\nlaenge=%f\n",i,j,ABS(distance));

        fprintf(stderr,"WARNING: monomer 1   %f %f %f\n",
                       mon->x,mon->y,mon->z);

        fprintf(stderr,"WARNING: monomer 2   %f %f %f\n",
                       (mon+1)->x,(mon+1)->y,(mon+1)->z);
      }
    }
  }

if ((nrchainsA>0)&&(polA>1))
  data.bondlength1A = length/((double)(nrmonA-nrchainsA));
else
  data.bondlength1A = 0;

if ((nrchainsA>0)&&(polA>1))
  data.bondlength2A = abslength/((double)(nrmonA-nrchainsA));
else
  data.bondlength2A = 0;

data.bondlengthmaxA = maxlength;


/*-----------------------------------*
 | calculate average bondlength, b**2 |
 | and bmax for type B polymer        |
 *-----------------------------------*/

length=0;
abslength=0;
maxlength=0;

for(i=0;i<nrchainsB;i++){

  mon=posB+i*polB;

  for(j=0;j<(polB-1);j++,mon++) {

    distance   = DIST(mon,mon+1);
    length    += (distance);
    abslength += SQUARE(distance);
    if (ABS(distance)>maxlength) maxlength=ABS(distance);
    if (ABS(distance)>2.0) {

      fprintf(stderr,"WARNING: typeB: chain %d mon "
                     "%i\nlaenge=%f\n",i,j,ABS(distance));

      fprintf(stderr,"WARNING: monomer 1   %f %f %f\n",
                     mon->x,mon->y,mon->z);

      fprintf(stderr,"WARNING: monomer 2   %f %f %f\n",
```

```
                    (mon+1)->x,(mon+1)->y,(mon+1)->z);
      }
    }
  }

  if ((nrchainsB>0)&&(polB>1))
    data.bondlength1B = length/((double)(nrmonB-nrchainsB));
  else
    data.bondlength1B = 0;

  if ((nrchainsB>0)&&(polB>1))
    data.bondlength2B = abslength/((double)(nrmonB-nrchainsB));
  else
    data.bondlength2B = 0;

  data.bondlengthmaxB = maxlength;

}


/****************************************
 * calc_polymer: calculates end-to-end   *
 *               distance and radius of  *
 *               gyration                *
 ****************************************/

void calc_polymer(){

  double end,gyration;
  MYVEC *mon1,*mon2;
  int i,j,k;


  /*--------------------------------------*
   | calculate end-to-end distance and    |
   | radius of gyration for type A polymer |
   *--------------------------------------*/

  end      = 0;
  gyration = 0;

  for(i=0;i<nrchainsA;i++){

    mon1 = posA + i*polA;
    end += DISTANCE2(mon1,mon1+polA-1);

    for(j=0;j<polA;j++,mon1++) {
      mon2 = posA +i*polA;
      for(k=0;k<polA;k++,mon2++)
        gyration += DISTANCE2(mon1,mon2);
    }
  }
  if (nrchainsA>0) {
    end      /= nrchainsA;
    gyration /= (2.0*SQUARE(polA)*nrchainsA);
  }
  else {
    end      = 0;
    gyration = 0;
  }

  data.endA = end;
  data.gyrA = gyration;


  /*--------------------------------------*
   | calculate end-to-end distance and    |
   | radius of gyration for type B polymer |
   *--------------------------------------*/

  end      = 0;
  gyration = 0;

  for(i=0;i<nrchainsB;i++){

    mon1 = posB + i*polB;
    end += DISTANCE2(mon1,mon1+polB-1);

    for(j=0;j<polB;j++,mon1++) {
      mon2 = posB +i*polB;
      for(k=0;k<polB;k++,mon2++)
        gyration += DISTANCE2(mon1,mon2);
    }
  }
  if (nrchainsB>0) {
    end      /= nrchainsB;
    gyration /= (2.0*SQUARE(polB)*nrchainsB);
  }
  else {
    end      = 0;
    gyration = 0;
  }

  data.endB = end;
```

```
  data.gyrB = gyration;

}


/***************************************
 * calc_nrchainsleft: calculates #chains *
 *                    in "left" half of  *
 *                    simulation box     *
 ***************************************/

void calc_nrchainsleft() {
  int i_mono,i_poly,nrchainsleft;
  double z;
  MYVEC  *mon;


  /*-------------------------------------------------*
   | calculate #chains in left half for type A polymer |
   *-------------------------------------------------*/

  nrchainsleft=0;
  for (i_poly=0;i_poly<nrchainsA;i_poly++) {

    mon = posA + i_poly*polA;
    z   = 0;

    for (i_mono=0;i_mono<polA;i_mono++,mon++) {
      z += mon->z;
    }

    z  /= polA;
    if (2.0*z < LSZ) nrchainsleft++;
  }
  data.nrchainsleftA = nrchainsleft;


  /*-------------------------------------------------*
   | calculate #chains in left half for type B polymer |
   *-------------------------------------------------*/

  nrchainsleft=0;
  for (i_poly=0;i_poly<nrchainsB;i_poly++) {

    mon = posB + i_poly*polB;
    z   = 0;

    for (i_mono=0;i_mono<polB;i_mono++,mon++) {
      z += mon->z;
    }

    z  /= polB;
    if (2.0*z < LSZ) nrchainsleft++;
  }
  data.nrchainsleftB = nrchainsleft;

}


/***************************************
 * calc_cm: calculates center-of-mass of  *
 *          polymer at position "start"   *
 *          in pos-array                  *
 ***************************************/

void calc_cm(int pol, MYVEC *start, MYVEC *result){

  int j;
  MYVEC *mon;

  result->x = 0;
  result->y = 0;
  result->z = 0;

  mon = start;

  for(j=0;j<pol;j++,mon++) {
    result->x += mon->x;
    result->y += mon->y;
    result->z += mon->z;
  }

  result->x /= pol;
  result->y /= pol;
  result->z /= pol;

}


/***************************************
 * calc_msd: calculates mean-square-      *
 *           displacement                 *
 ***************************************/
```

```
void calc_msd() {

  int i;
  MYVEC cm, *mon, *vhilf;
  double rmove;


  /*-------------------------------------------------------*
   | calculate mean-square-displacement for type A polymer |
   *-------------------------------------------------------*/

  vhilf= cm0A;
  rmove=0;

  for(i=0;i<nrchainsA;i++,vhilf++) {

    mon = posA + i*polA;
    calc_cm(polA,mon,&cm);

    rmove += (SQUARE((vhilf->x - cm.x))+\
              SQUARE((vhilf->y - cm.y))+\
              SQUARE((vhilf->z - cm.z)));
  }

  if (nrchainsA>0)
    data.msdA = rmove/nrchainsA;
  else
    data.msdA = 0;


  /*-------------------------------------------------------*
   | calculate mean-square-displacement for type B polymer |
   *-------------------------------------------------------*/

  vhilf= cm0B;
  rmove=0;

  for(i=0;i<nrchainsB;i++,vhilf++) {

    mon = posB + i*polB;
    calc_cm(polB,mon,&cm);

    rmove += (SQUARE((vhilf->x - cm.x))+\
              SQUARE((vhilf->y - cm.y))+\
              SQUARE((vhilf->z - cm.z)));
  }

  if (nrchainsB>0)
    data.msdB = rmove/nrchainsB;
  else
    data.msdB = 0;

}


/****************************************
 * calc_energy: calculates total FENE,   *
 *              LJ and wall energy       *
 ****************************************/

void calc_energy(){

  int i,j;
  double poten,poly,zcount,wall,rad2,rad6,rad6inv;
  MYVEC *mon1,*mon2,diff;


  /*-------------------------------------------------------*
   | step 1a: calculate FENE potential for type A polymer |
   *-------------------------------------------------------*/

  poly = 0.0;

  for(i=0;i<nrchainsA;i++){

    mon1 = posA + i*polA;
    mon2 = mon1 + 1;

    for(j=0;j<polA-1;j++,mon1++,mon2++){
      rad2 = DISTANCE2(mon1,mon2);
      poly += BOND(rad2);             /* FENE potential */
    }
  }

  data.energy_feneA = poly;


  /*-------------------------------------------------------*
   | step 1b: calculate FENE potential for type B polymer |
   *-------------------------------------------------------*/

  poly = 0.0;

  for(i=0;i<nrchainsB;i++){
```

```
    mon1 = posB + i*polB;
    mon2 = mon1 + 1;

    for(j=0;j<polB-1;j++,mon1++,mon2++){
      rad2 = DISTANCE2(mon1,mon2);
      poly += BOND(rad2);
    }
  }

  data.energy_feneB = poly;


  /*-------------------------------------------------------*
   | step 2a: calculate LJ energy between type A polymers |
   *-------------------------------------------------------*/

  poten  = 0.0;
  zcount = 0.0;
  mon1 = poslatA;

  for(i=0;i<nrmonA;i++,mon1++){

    mon2 = mon1+1;

    for(j=i+1;j<nrmonA;j++,mon2++){
    /* +2 because of bond energy */

      diff.x = mon1->x - mon2->x;
      diff.y = mon1->y - mon2->y;
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */

      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2AA) {
      /* check if distance between monomers < */
      /* interaction range                    */

        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        poten  += LENJAA(rad6inv);
        zcount += 1-0.5*rad6;
      }
    }
  }
  poten *= 4.0;

  data.energy_ljA   = poten;
  data.energy_sqA   = zcount;


  /*-------------------------------------------------------*
   | step 2b: calculate LJ energy between type B polymers |
   *-------------------------------------------------------*/

  poten  = 0.0;
  zcount = 0.0;
  mon1 = poslatB;

  for(i=0;i<nrmonB;i++,mon1++){

    mon2 = mon1+1;

    for(j=i+1;j<nrmonB;j++,mon2++){

      diff.x = mon1->x - mon2->x;
      diff.y = mon1->y - mon2->y;
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */

      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2BB) {
        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        poten  += LENJBB(rad6inv);
        zcount += 1-0.5*rad6;
```

```
      }
    }
  }
  poten *= 4.0;

  data.energy_ljB   = poten;
  data.energy_sqB   = zcount;


  /*-----------------------------------------------------*
   | step 2c: calculate (type A - type B) LJ interaction |
   *-----------------------------------------------------*/

  poten  = 0.0;
  zcount = 0.0;
  mon1 = poslatA;

  for(i=0;i<nrmonA;i++,mon1++){

    mon2 = poslatB;

    for(j=0;j<nrmonB;j++,mon2++){

      diff.x = mon1->x - mon2->x;
      diff.y = mon1->y - mon2->y;
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */

      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2AB && rad2 != 0) {
        rad6    = CUBE(rad2);
        rad6inv = 1.0/rad6;
        poten  += LENJAB(rad6inv);
        zcount += 1-0.5*rad6;
      }
    }
  }
  poten *= 4.0;

  data.energy_ljAB   = poten;
  data.energy_sqAB   = zcount;


  /*-----------------------------------*
   | step 3: calculate wall energy     |
   |                                   |
   | WARNING: this routine has not been |
   |          adjusted to two species ! |
   *-----------------------------------*/

  wall   = 0.0;

/*  mon1 = poslat;

    for (i=0;i<nrmon;i++,mon1++){
      if (i<NPOLYBRUSH*pol) wall += wbenergy(mon1);
      else                  wall += wenergy(mon1);
    }
*/

  data.energy_wall = wall;

}
```

Analyseroutinen: pressure.c

```
/*******************************************
 * pressure.c: measures pressure in the system *
 *                                             *
 * last modification: 23.05.2003               *
 *******************************************/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "element.h"


/* global variables */

extern int        nrchainsA, nrmonA, polA;
extern int        nrchainsB, nrmonB, polB;
```

```
extern double     LSX,LSY,LSZ;
extern MYVEC      *posA,*poslatA;
extern MYVEC      *posB,*poslatB;
extern double     T,Ewall,Fwall;
extern DATA       data;


/*****************************************
 * calc_p: calculates pressure using the *
 *         virial expression             *
 *                                       *
 * P  = (nkT)/V + 1/(3V) *               *
 *      sum (Fx*rx + Fy*ry + Fz*rz)      *
 *                                       *
 * Fx = -dE/dx, Fy = -dE/dy, Fz = -dE/dZ *
 *****************************************/

int calc_p(void)
{
int     i1,ii1,i2,ii2,type1,type1a,type2,type2a;
double  V,dummy,rad2,invrad2,invrad4,invrad8,invrad14;
double  p0,pljx,pljy,pljz,pfex,pfey,pfez,pwall1,pwall2,z1,z2;
double  z1_10,z1_4,z2_10,z2_4,px,py,pz,virialx,virialy,virialz;
MYVEC   *mon1,*mon2,diff;


   /*--------------------*
    | set variables to 0 |
    *--------------------*/

   V      = LSX*LSY*LSZ;
   pljx   = 0;
   pljy   = 0;
   pljz   = 0;
   pfex   = 0;
   pfey   = 0;
   pfez   = 0;
   pwall1 = 0;
   pwall2 = 0;
   virialx =0;
   virialy =0;
   virialz =0;

#ifdef HARDWALLS
   for( i1=0,mon1=pos ; i1<nrchains ; i1++){
     for ( ii1=0 ; ii1< pol ; ii1++){
       z1    = 1.0/mon1->z;
       z2    = 1.0/(LSZ-mon1->z);
       z1_4  = z1*z1*z1*z1;
       z2_4  = z2*z2*z2*z2;
       z1_10 = z1_4*z1_4*z1*z1;
       z2_10 = z2_4*z2_4*z2*z2;
       dummy = 3*Ewall/(LSX*LSY)*
               (3*z1_10-Fwall*z1_4-3*z2_10+Fwall*z2_4);
       if (mon1->z<LSZ/2.0)
         pwall1 += dummy;
       else
         pwall1 -= dummy;
         /* modified pwall1 is the pressure the wall */
         /* exerts on the system                     */

       pwall2 += (mon1->z-0.5*LSZ)*dummy/LSZ;
       /* pwall2 is the viral of the external forces */
       /* (check again)*/

       mon1++;
     }
   }
#endif


   /*---------------------------------*
    | step 1: consider AA interactions |
    *---------------------------------*/

   for (i1=0,mon1=poslatA;i1<nrchainsA;i1++)
   /* loop over all polymers */

   for (ii1=0;ii1<polA;ii1++,mon1++)
   /* loop over all monomers */

   for (i2=0,mon2=poslatA;i2<nrchainsA;i2++)
   /* loop over all interaction partner polymers */

   for (ii2=0;ii2<polA;ii2++,mon2++) {
   /* loop over all interaction partner monomers */
   /* -> all monomers interact with each other   */

     if (i1*polA+ii1>i2*polA+ii2) {
     /* each interaction is only counted once      */

       /* determine distance between interacting monomers */
       diff.x = mon1->x - mon2->x;
       diff.y = mon1->y - mon2->y;
```

```
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */
      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif


      /*---------------------------*
       | compute LJ part of virial |
       *---------------------------*/

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2AA) {
         invrad2  = 1.0/rad2;
         invrad4  = invrad2*invrad2;
         invrad8  = invrad4*invrad4;
         invrad14 = invrad8*invrad4*invrad2;
         p0       = 24*(2*invrad14-invrad8);
         /* compare with LJ potential definition  */
         /* in element.h EPSILONA and SIGMAA := 1 */


         pljx    += diff.x*diff.x*p0; /* = -(dE(LJ)/dx)*x */
         pljy    += diff.y*diff.y*p0; /* = -(dE(LJ)/dy)*y */
         pljz    += diff.z*diff.z*p0; /* = -(dE(LJ)/dz)*z */
      }


      /*----------------------------*
       | compute FENE part of virial |
       *----------------------------*/

      if ((i1==i2)&&(ii1==ii2+1)) {
         p0       = -67.5*RNULL2INV/(1-rad2*RNULL2INV);
         /* compare with FENE potential definition */
         /* in element.h SIGMAA in RNULL2INV :=1   */

         pfex    += diff.x*diff.x*p0; /* = -(dE(FENE)/dx)*x */
         pfey    += diff.y*diff.y*p0; /* = -(dE(FENE)/dy)*y */
         pfez    += diff.z*diff.z*p0; /* = -(dE(FENE)/dz)*z */
      }
    } /* each interaction is only counted once */
  } /* all AA interactions are considered */


  /*---------------------------------*
   | step 2: consider BB interactions |
   *---------------------------------*/

  for (i1=0,mon1=poslatB;i1<nrchainsB;i1++)
  /* loop over all polymers */

  for (ii1=0;ii1<polB;ii1++,mon1++)
  /* loop over all monomers */

  for (i2=0,mon2=poslatB;i2<nrchainsB;i2++)
  /* loop over all interaction partner polymers */

  for (ii2=0;ii2<polB;ii2++,mon2++)  {
  /* loop over all interaction partner monomers */
  /* -> all monomers interact with each other   */

    if (i1*polB+ii1>i2*polB+ii2) {
    /* each interaction is only counted once      */

      /* determine distance between interacting monomers */
      diff.x = mon1->x - mon2->x;
      diff.y = mon1->y - mon2->y;
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */
      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif

      /*---------------------------*
       | compute LJ part of virial |
       *---------------------------*/

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2BB) {
         invrad2  = 1.0/rad2;
         invrad4  = invrad2*invrad2;
         invrad8  = invrad4*invrad4;
         invrad14 = invrad8*invrad4*invrad2;
```

```
      p0       = 24*EPSILONB*(2*SQUARE(SIGMAB6)*invrad14-
                 SIGMAB6*invrad8);
      /* compare with LJ potential definition */
      /* in element.h                         */

      pljx    += diff.x*diff.x*p0;  /* = -(dE(LJ)/dx)*x */
      pljy    += diff.y*diff.y*p0;  /* = -(dE(LJ)/dy)*y */
      pljz    += diff.z*diff.z*p0;  /* = -(dE(LJ)/dz)*z */
    }


    /*-----------------------------*
     | compute FENE part of virial |
     *-----------------------------*/

    if ((i1==i2)&&(ii1==ii2+1)) {
    /* has to be adjusted to 2nd polymer species ! */
    /* -> change RNULL2INV to RNULL2INVB, EPSILONB */
    /* right now it only works for N=1 */

      p0       = -67.5*RNULL2INV/(1-rad2*RNULL2INV);
      /* compare with FENE potential definition */
      /* in element.h                           */

      pfex    += diff.x*diff.x*p0; /* = -(dE(FENE)/dx)*x */
      pfey    += diff.y*diff.y*p0; /* = -(dE(FENE)/dy)*y */
      pfez    += diff.z*diff.z*p0; /* = -(dE(FENE)/dz)*z */
    }
  } /* each interaction is only counted once */
} /* all BB interactions are considered */


/*---------------------------------*
 | step 3: consider AB interactions |
 *---------------------------------*/

for (i1=0,mon1=poslatA;i1<nrchainsA;i1++)
/* loop over all polymers */

for (ii1=0;ii1<polA;ii1++,mon1++)
/* loop over all monomers */

for (i2=0,mon2=poslatB;i2<nrchainsB;i2++)
/* loop over all interaction partner polymers */

for (ii2=0;ii2<polB;ii2++,mon2++) {
/* loop over all interaction partner monomers */
```

```
  /* -> all monomers interact with each other   */

       /* determine distance between interacting monomers */
      diff.x = mon1->x - mon2->x;
      diff.y = mon1->y - mon2->y;
      diff.z = mon1->z - mon2->z;

      /* Minimum Image Convention */
      diff.x = diff.x - LSX*((int)(2.0*diff.x/LSX));
      diff.y = diff.y - LSY*((int)(2.0*diff.y/LSY));

#ifndef HARDWALLS
      diff.z = diff.z - LSZ*((int)(2.0*diff.z/LSZ));
#endif


      /*---------------------------*
       | compute LJ part of virial |
       *---------------------------*/

      rad2 = SQUARE(diff.x)+SQUARE(diff.y)+SQUARE(diff.z);

      if (rad2 < RANGE2AB) {
        invrad2  = 1.0/rad2;
        invrad4  = invrad2*invrad2;
        invrad8  = invrad4*invrad4;
        invrad14 = invrad8*invrad4*invrad2;
        p0       = 24*EPSILONAB*(2*SQUARE(SIGMAAB6)*invrad14-
                   SIGMAAB6*invrad8);
        /* compare with LJ potential definition */
        /* in element.h                         */

        pljx    += diff.x*diff.x*p0;  /* = -(dE(LJ)/dx)*x */
        pljy    += diff.y*diff.y*p0;  /* = -(dE(LJ)/dy)*y */
        pljz    += diff.z*diff.z*p0;  /* = -(dE(LJ)/dz)*z */

        virialx += diff.x*diff.x*p0;
        virialy += diff.y*diff.y*p0;
        virialz += diff.z*diff.z*p0;
      }
  } /* all AB interactions are considered */


  /*---------------------------------------*
   | step 4: calculate pressure in the system |
   *---------------------------------------*/
```

```
   px = (T*(double)(nrmonA+nrmonB) + pljx+pfex)/V;
   /* px, py and pz each account for */

   py = (T*(double)(nrmonA+nrmonB) + pljy+pfey)/V;
   /* total pressure in the system   */

   pz = (T*(double)(nrmonA+nrmonB) + pljz+pfez)/V;
   /* - in a homogeneous system they */
   /* should yield the same result   */

   data.px     = px;  /* save results in data-array     */
   data.py     = py;
   data.pz     = pz;
   data.virialx = virialx/V;
   data.virialy = virialy/V;
   data.virialz = virialz/V;
   data.pwall1 = pwall1;
   data.pwall2 = pwall2;

return(OK);
}
```

Zufallszahlengenerator: r250.h

```
/*************************************************/
/*                                               */
/* File: r250.h                                  */
/*                                               */
/* Prototypes for the R250 random number generator */
/*                                               */
/*************************************************/


/*#define PROBLEM_R250*/
/*#define SERIOUS_R250*/

#ifndef R250_H
#define R250_H

static char
r250_h[]="$Id: r250.h,v 1.5 1991/05/15 11:23:19 "
         "reger Exp reger $";

double   double_r250( void );
float    float_r250( void );
void     double_r250_vector( register double *x, int n );
void     float_r250_vector( register float *x, int n );
float    float_rand( void );
double   gauss_r250( double mean, double variance );
int      int_rand( void );
int      r250_random( void );
void     r250_restart( int table[] );
void     r250_save( int table[] );
void     r250_srandom( unsigned int seed );
void     r250_vector( register int *x, int n );
int      rand_save( void );
/*void   srand( int seed );*/

#endif
```

Zufallszahlengenerator: r250.c

```
/*************************************************/
/*                                               */
/*  File: r250.c                                 */
/*                                               */
/*  The R250 random number generator routines.   */
/*                                               */
/*************************************************/

static char
rcs[] = "$Id: r250.c,v 1.5 1991/08/09 11:25:07 reger Exp $";

#include <math.h>
#include <stdio.h>
#include "r250.h"

#ifndef INT_MAX
#        define INT_MAX                 0x7FFFFFFF
#endif
#if INT_MAX > 0x7FFFFFFF
#        define INT_MAX                 0x7FFFFFFF
#endif

#define INVERSE_INT_MAX          (1.0/INT_MAX)

#define                TYPE_3                  3
/* x**31 + x**3 + 1 */

#define                BREAK_3                128
```

```
#define                DEG_3                   31
#define                SEP_3                    3

#define P 250
#define Q 103

static int m[P]    =
  {  0x7be9c1bd, 0x088aa102, 0x3d38509b, 0x746b9fbe,
     0x2d04417f, 0x775d4351, 0x53c48d96, 0x02b26e0b,
     0x418fedcf, 0x19dbc19e, 0x78512adb, 0x1a1f5e2b,
     0x307d7761, 0x6584c1f0, 0x24e3c36f, 0x2232310f,
     0x2dac5ceb, 0x106e8b5a, 0x5a05a938, 0x5e6392b6,
     0x66b90348, 0x75264901, 0x4174f402, 0x618b18a4,
     0x3a6ab4bf, 0x3c4bc289, 0x2657a9a9, 0x4e68589b,
     0x09648aa6, 0x3fc489bb, 0x1c1b715c, 0x054e4c63,
     0x484f2abd, 0x5953c1f8, 0x79b9ec22, 0x75536c3d,
     0x50b10549, 0x4d7e79b8, 0x7805da48, 0x1240f318,
     0x675a3b56, 0x70570523, 0x2c605143, 0x17d7b2b7,
     0x55dbc713, 0x514414b2, 0x3a09e3c7, 0x038823ff,
     0x61b2a00c, 0x140f8cff, 0x61ebb6b5, 0x486ba354,
     0x0935d600, 0x2360aab8, 0x29f6bbf8, 0x43a08abf,
     0x5fac6d41, 0x504e65a2, 0x1208e35b, 0x6910f7e7,
     0x1012ef5d, 0x2e2454b7, 0x6e5f444b, 0x58621a1b,
     0x077816af, 0x6819306d, 0x4db58658, 0x58291bf8,
     0x3597aa25, 0x45bb60a0, 0x6a6a0f11, 0x1cf1e57c,
     0x361265c4, 0x16ca6054, 0x34c99833, 0x0bee2cd7,
     0x680e7507, 0x6ed37bfa, 0x0f7650d6, 0x49c11513,
     0x02e308f9, 0x7162078c, 0x122cb868, 0x0c18defa,
     0x14c2b244, 0x3c237460, 0x4fb969b9, 0x746f1f85,
     0x0c71da02, 0x61c24d14, 0x5d80176d, 0x1c84c960,
     0x0fe6a1cc, 0x4bdf5bb8, 0x74e6e37b, 0x175eb87b,
     0x33f88c25, 0x429c69d3, 0x6f87d474, 0x6990364a,
     0x0857ca73, 0x59f1e385, 0x06821bc6, 0x3e6a3037,
     0x70bc43d9, 0x3b4bb3fa, 0x4a585d0f, 0x58cab8e0,
     0x2a1f2ff4, 0x59ceade5, 0x228bcdf4, 0x2d0238ee,
     0x4b30b571, 0x34b8865c, 0x391b17e8, 0x5ff367b5,
     0x70dbfabc, 0x08d481a1, 0x5462873b, 0x7d4dd4bf,
     0x6a96ceb6, 0x31e29ea8, 0x19d29e1f, 0x7a7d7082,
     0x7dc1fa60, 0x0eb9819a, 0x11dc28fd, 0x31ba8685,
     0x5155eb6d, 0x0163fd71, 0x1b4abccf, 0x59adb5e0,
     0x5b55e0f6, 0x21ccd896, 0x1817e618, 0x4c1224d0,
     0x5d188c90, 0x62704327, 0x24dcddb0, 0x0737bc84,
     0x3c3ef10c, 0x4768aba4, 0x3439f572, 0x076fa67e,
     0x7c213200, 0x6d550d5a, 0x67630e33, 0x6cfd2cbd,
     0x76298efc, 0x3bc5956e, 0x6a4b017c, 0x60c05db2,
     0x6da83416, 0x041d9f9b, 0x5b3dce34, 0x6b6a2e76,
     0x12d72135, 0x6d19f731, 0x1d24b4fb, 0x642d0ca2,
     0x6e7df4a3, 0x386f71cb, 0x3ddac282, 0x49d3d599,
     0x5a3c4a61, 0x55f2a89a, 0x15e5fa69, 0x3754d6f1,
     0x3862ebc1, 0x3ac2d81a, 0x3e8c9375, 0x74a1dcce,
     0x022b83be, 0x72c688e8, 0x7c11834c, 0x7e4cb5bf,
     0x601b9642, 0x6374917f, 0x6b49e27c, 0x5645253e,
     0x1f3a26ee, 0x5594e3f8, 0x370582f0, 0x0ce25b04,
     0x59b28393, 0x12435124, 0x784c897b, 0x6c89a4c8,
     0x7f5d4856, 0x15713e76, 0x50b6b16a, 0x6ddb3cf9,
     0x4de0b041, 0x0e9173ec, 0x37af1292, 0x281cfaa2,
     0x64841c87, 0x4d950cfc, 0x5f71d193, 0x1ce70848,
     0x0857e516, 0x1dfe6509, 0x1188e516, 0x0a8368d4,
     0x10c4edf1, 0x0d9a6862, 0x08d01e93, 0x70e08433,
     0x710ef9e2, 0x741a010f, 0x4725a972, 0x104920d0,
     0x49aee507, 0x7e2b2c62, 0x1d2b7bd4, 0x2361689a,
     0x106e7d87, 0x1578054f, 0x0feb0d62, 0x0fcbc5dd,
     0x2ae943c6, 0x60a1becc, 0x7da702d6, 0x78c9f407,
     0x6f3332b9, 0x35561568, 0x20e6eeaa, 0x53b74f40,
     0x02eb2264, 0x0058c03d, 0x709e5788, 0x0b43077a,
     0x1e572546, 0x02273c9f, 0x15c6704f, 0x2f1c1337,
     0x0fc1a501, 0x1e968ee2, 0x1ffc976b, 0x00d09ee3,
     0x12b08ff2, 0x672240dd, 0x1119bfb3, 0x5c5f74f9,
     0x654d6d3f, 0x2e453b88, 0x7fc0dd94, 0x75bbeac6,
     0x43bd40d7, 0x0fabeaf6 };

static int *k_static = m + P;
static int *j_static = m + P-Q;
static int *end          = m + P-1;

static  int        randtbl[ DEG_3 ]            =
  { 0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
    0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
    0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
    0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
    0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
    0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
    0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
    0xf5ad9d0e, 0x8999220b, 0x27fb47b9 };

static  int                     *fptr    = randtbl + SEP_3;
static  int                     *rptr    = randtbl;
static  int                     *end_ptr = randtbl + DEG_3;
static  int                     *state   = randtbl;

static  int                     rand_deg = DEG_3;
static  int                     rand_sep = SEP_3;

static int
```

```
random_reger( void )

{
            int                     i;

            *fptr += *rptr;
            i = (*fptr >> 1) & INT_MAX;
            /* chucking least random bit */

            if(  ++fptr  >=  end_ptr  )  {
                     fptr = state;
                     ++rptr;
            }
            else  {
                  if(  ++rptr  >=  end_ptr  )  rptr = state;
            }
            return( i );
}

static void
srandom_reger( unsigned x )

{
            register  int                   i;
                      int random_reger( void );

            state[ 0 ] = x;
            for( i = 1; i < rand_deg; i++ )  {
                    state[i] = (1103515245*state[i - 1] +
                                12345) & INT_MAX;
            }
            fptr = &state[ rand_sep ];
            rptr = &state[ 0 ];
            for( i = 0; i < 10*rand_deg; i++ )  random_reger();
}

void
r250_srandom ( unsigned seed )

{
              register int  i;

              srandom_reger( seed );

              for ( i=0; i<P; ++i) m[i] = random_reger();

 k_static = m + P;
 j_static = m + P-Q;
}

int
r250_random( void )

{

      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;

      return (*k_static ^= *j_static);
}

float
float_r250( void )

{
      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;

      return ( (*k_static ^= *j_static) * INVERSE_INT_MAX );
}

double
double_r250( void )

{
int rnd_int;

      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;
      rnd_int = (*k_static ^= *j_static);
      while (rnd_int==0) {
            if (--k_static < m) k_static = end;
            if (--j_static < m) j_static = end;
            rnd_int = (*k_static ^= *j_static);
      }
      return ( rnd_int * INVERSE_INT_MAX );
}

void
r250_save ( int table[] )

{
      int  i;
```

```
      for ( i = 0; i < P; i++ )
 table[i] = m[i];

      table[P  ] = k_static - m;
      table[P+1] = j_static - m;
}

void
r250_restart ( int table[] )

{
      int  i;

      for ( i = 0; i < P; i++ )
 m[i] = table[i];

      k_static = m + table[P  ];
      j_static = m + table[P+1];
}

double
gauss_r250 ( double mean, double variance )

{
      static int           empty = 1;
      static double        store;
      double               v1, v2, r, root;

      if ( empty ) {

 do { /* Inline double_random() */

      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;
      v1 = (*k_static ^= *j_static) * INVERSE_INT_MAX;

      v1 = 2.0 * v1 - 1.0;

      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;
      v2 = (*k_static ^= *j_static) * INVERSE_INT_MAX;

      v2 = 2.0 * v2 - 1.0;

      r  = v1 * v1 + v2 * v2;

 } while ( r > 1.0 );

 root  = variance * sqrt( -2.0 * log(r) / r );
 store = v2 * root + mean;
 empty = 0;

 return ( v1 * root + mean );

      } else {

 empty = 1;
 return ( store );

      }

}



# ifndef PROBLEM_R250
      # define MAX(a,b) ((a)>(b)?(a):(b))
# else
      # define MAX(a,b) ((((a)>(b))&&((a)<(int *)\
0x80000000  ))?(a):(b))
# endif

void
r250_vector ( register int *x, int n)

{
      register int         *j, *k, *l;

      j = j_static;
      k = k_static;

      while ( n > 0 ) {

 if ( k == m ) k = m + P;
 l = MAX( k-n, m+Q );
 if ( k > l ) {
      n -= k-l;
#pragma ivdep
      while ( k > l )
         *x++ = (*--k ^= *--j);
 }

 if ( j == m ) j = m + P;
 l = MAX( j-n, m+P-Q );
```

```
 if ( j > l ) {
      n -= j-l;
#pragma ivdep
      while ( j > l )
         *x++ = (*--k ^= *--j);
 }

      }

      j_static = j;
      k_static = k;

}



void
float_r250_vector ( register float *x, int n)

{
      register int         *j, *k, *l;

      j = j_static;
      k = k_static;

      while ( n > 0 ) {

 if ( k == m ) k = m + P;
 l = MAX( k-n, m+Q );
 if ( k > l ) {
      n -= k-l;
#pragma ivdep
      while ( k > l )
         *x++ = (*--k ^= *--j) * INVERSE_INT_MAX;
 }

 if ( j == m ) j = m + P;
 l = MAX( j-n, m+P-Q );
 if ( j > l ) {
      n -= j-l;
#pragma ivdep
      while ( j > l )
         *x++ = (*--k ^= *--j) * INVERSE_INT_MAX;
 }

      }

      j_static = j;
      k_static = k;

}

#ifndef SERIOUS_R250
void
double_r250_vector ( register double *x, int n)

{
      register int         *j, *k, *l;
      int                               rnd;

      j = j_static;
      k = k_static;

      while ( n > 0 ) {

 if ( k == m ) k = m + P;
 l = MAX( k-n, m+Q );
 if ( k > l ) {
      n -= k-l;
#pragma ivdep
      while ( k > l ) {
         rnd = (*--k ^= *--j);
         if (rnd==0) rnd=1;
         *x++ = rnd * INVERSE_INT_MAX;
      }
 }

 if ( j == m ) j = m + P;
 l = MAX( j-n, m+P-Q );
 if ( j > l ) {
      n -= j-l;
#pragma ivdep
      while ( j > l ) {
         rnd = (*--k ^= *--j);
         if (rnd==0) rnd=1;
         *x++ = rnd * INVERSE_INT_MAX;
      }
 }

      }

      j_static = j;
      k_static = k;
```

```
}
#else
void
double_r250_vector ( register double *x, int n)
{
double *ptr,rnd;
int      i,rnd_int;

    for (i=0,ptr=x;i<n;i++) {
      if (--k_static < m) k_static = end;
      if (--j_static < m) j_static = end;
      rnd_int = (*k_static ^= *j_static);
      while (rnd_int==0) {
            if (--k_static < m) k_static = end;
            if (--j_static < m) j_static = end;
            rnd_int = (*k_static ^= *j_static);
      }
      rnd  = rnd_int * INVERSE_INT_MAX;
      *x++ = rnd;
    }
}
#endif



/*************************************************/
/*                                               */
/* File: r250.c                                  */
/*                                               */
/* A simple multiplicative generator from        */
/* Kalos & Whitlock.                             */
/*                                               */
/*************************************************/

static int       rand_seed = 1;

int
int_rand( void )

{
rand_seed = ( 1812433253 * rand_seed + 314159265 ) & INT_MAX;
return  ( rand_seed );
}

float
float_rand( void )

{
rand_seed = ( 1812433253 * rand_seed + 314159265 ) & INT_MAX;
return  ( rand_seed * INVERSE_INT_MAX );
}


void
save_rand( int seed )

{
      rand_seed = seed;
}

int
rand_save( void )

{
      return ( rand_seed );
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] S.G. Kazarian, Polym. Sci., Ser. C **42**, 78 (2000).

[2] W-C.V. Wang, E.J. Kramer und W.H. Sachse, Journal of Polymer Science, Polymer Physics Edition **20**, 1371 (1982).

[3] http://www.basf.de/basf/html/plastics/deutsch/pages/schaum/styrodur.htm .

[4] J.D. van der Waals, *Over de Continuiteit van den Gas- en Vloeistoftoestand* (Dissertation, Universität Leiden - deutsche Übersetzung: Leipzig, 1873).

[5] D.E. Young und R.H. Strauss, J. Appl. Phys. **47**, 5081 (1976).

[6] J. Banhart, H. Stanzick, L. Helfen und T. Baumbach, Appl. Phys. Lett. **78**, 1152 (2001).

[7] F.V. Deblasio, Astrophys. J. **452**, 359 (1995).

[8] B. Smit, S. Karaborni und J.I. Siepmann, J. Chem. Phys. **102**, 2126 (1995).

[9] J.J. Potoff, J.R. Errington und A.Z. Panagiotopoulos, Molec. Phys. **97**, 1073 (1999).

[10] F. London, Trans. Faraday Soc. **33**, 8 (1937).

[11] B. Smit und D. Frenkel, J. Chem. Phys. **94**, 5663 (1991).

[12] C. Bennemann, W. Paul, K. Binder und B. Dünweg, Phys. Rev. E **57**, 843 (1998).

[13] K. Kremer und G.S. Grest, J. Chem. Phys **92**, 5057 (1990).

[14] H.A. Lorentz, Annln Phys. **12**, 127 (1881).

[15] D.C. Berthelot, r. hebd. Seanc. Acad Sci., Paris **126**, 1703 (1898).

[16] G.C. Maitland, M. Rigby, E.B. Smith und W.A. Wakeham, *Intermolecular Forces* (Oxford: Clarendon Press, 1987).

[17] C.L. Kong, J. Chem. Phys. **59**, 2464 (1973).

[18] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller und E.Teller, J. Chem. Phys. **21**, 1087 (1953).

[19] D.P. Landau und K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, 2000).

[20] D. Frenkel und B. Smit, *Understanding Molecular Simulation* (Academic, Boston, 1996).

[21] A.K. Kron, Polymer. Sci. USSR **7**, 1361 (1965).

[22] F.T. Wall und F. Mandel, J. Chem. Phys. **63**, 4592 (1975).

[23] B. Smit, Mol. Phys. **85**, 153 (1995).

[24] M.N. Rosenbluth und A.W. Rosenbluth, J. Chem. Phys. **23**, 356 (1955).

[25] K. Binder und D. Landau, Phys. Rev. B **30**, 1477 (1984).

[26] C. Borgs und R. Kotecky, J. Stat. Phys. **60**, 79 (1990).

[27] A.M. Ferrenberg und R.H. Swendsen, Phys. Rev. Lett. **61**, 2635 (1988).

[28] H.E. Stanley, *An Introduction to Phase Transitions and Critical Phenomena* (Oxford University Press, Oxford, 1971).

[29] M.E. Fisher, Rev. Mod. Phys. **46**, 597 (1974).

[30] M. Lee-Koo und M.S. Green, Phys. Rev. A **23**, 2650 (1981).

[31] N.B. Wilding und A.D. Bruce, J. Phys. Condens. Matter **4**, 3087 (1992).

[32] A.M. Ferrenberg und D.P. Landau, Phys. Rev. B **44**, 5081 (1991).

[33] N. Goldenfeld, *Lectures on phase transitions and the renormalization group* (Perseus Publishing, 1992).

[34] K. Binder, Z. Phys. B **43**, 119 (1981).

[35] M. Müller und N.B. Wilding, Phys. Rev. B **51**, 2079 (1995).

[36] K. Binder, Phys. Rev. A **25**, 1699 (1982).

[37] K. Binder, Physica A **319**, 99 (2003).

[38] B.A. Berg und T. Neuhaus, Phys. Rev. Lett. **68**, 9 (1992).

[39] B.A. Berg, U. Hansmann und T. Neuhaus, Z. Phys. **90**, 229 (1993).

[40] N.B. Wilding, Phys. Rev. E **52**, 602 (1995).

[41] F. Wang und D.P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).

[42] G.M. Torrie und J.P. Valleau, J. Comput. Phys. **23**, 187 (1977).

[43] D. Chandler, *Introduction to Modern Statistical Mechanics* (Oxford University Press, New York, 1987).

[44] P. Virnau und M. Müller, eingereicht bei Phys. Rev. Lett. (2002).

[45] B.J. Schulz, K. Binder, D.P. Landau und M. Müller, im Druck bei Phys. Rev. E. (2003).

[46] F. Wang und D.P. Landau, Phys. Rev. E **64**, 056101 (2001).

[47] B.J. Schulz, K. Binder und M. Müller, Int. J. Mod. Phys. C **13**, 477 (2001).

[48] B.A. Berg, J. Stat. Phys. **82**, 323 (1996).

[49] P. Virnau, L.G. MacDowell, M.Müller und K. Binder, in Vorbereitung (2003).

[50] L.G. MacDowell, P. Virnau, M.Müller und K. Binder, J. Chem. Phys. **117**, 6360 (2002).

[51] *CRC Handbook of Chemistry and Physics*, Band 63rd ed. (CRC Press, 1985).

[52] H.L. Swinney und D.L. Henry, Phys. Rev. A **8**, 2586 (1973).

[53] M.R. Moldover, Phys. Rev. A **31**, 1022 (1985).

[54] M.J. Anselme, M. Gude und A.S. Teja, Fluid Phase Equilibria **57**, 317 (1990).

[55] D. Ambrose und C. Tsnopoulos, J. Chem. Eng. Data **40**, 531 (1995).

[56] R. Span und W. Wagner, JPCRD **25**, 1509 (1996).

[57] B.D. Smith und R. Srivastava, *Thermodynamic Data for Pure Compounds: Hydrocarbons and Ketones* (Elsevier, Amsterdam, 1986).

[58] J.P. Hansen, *Theory of simple liquids* (Academic Press, 1986).

[59] G.T. Dee und B.B. Sauer, J. Colloid Interface Sci. **152**, 85 (1992).

[60] G.T. Dee und B.B. Sauer, Adv. Phys. **47**, 161 (1998).

[61] J.P. Nicolas und B. Smit, Molecular Physics **100**, 2471 (2002).

[62] R.L. Scott und P.H. van Konynenburg, Discuss. Faraday Soc **49**, 87 (1970).

[63] P.H. van Konynenburg und R.L. Scott, Phil. Trans. **A298**, 495 (1980).

[64] G.M. Schneider, Adv. Chem. Phys. **27**, 1.

[65] J.S. Rowlinson und F.L. Swinton, *Liquids and Liquid Mixtures*, Band 3rd ed. (Butterworth, 1982).

[66] J.A. Schouten, A. Deerenberg und N.J. Trappeniers, Physica **81A**, 151 (1975).

[67] M. Ruhemann, Proc. R. Soc. **A171**, 121 (1939).

[68] I. Wichterle und R. Kobayashi, J. Chem. Eng. Data **17**, 9 (1972).

[69] G.H. Zenner und L.I. Dana, Chem. Eng. Prog. Symp. Ser. **59**, 36 (1963).

[70] N.K. Muirbrook und J.M. Prausnitz, A.I.Ch.E.Jl. **11**, 1092 (1965).

[71] U.K. Im und F. Kurata, J. Chem. Eng. Data **16**, 412 (1971).

[72] G. Schneider, Z. Alwani, W. Heim, E. Horvath und E.U. Franck, Chem.-Ing.-Techn. **39**, 649 (1967).

[73] A. Galindo und F. J. Blas, J. Phys. Chem. B **106**, 4503 (2002).

[74] R. Enick, G.D. Holder und B.I. Morsi, Fluid Phase Equilibria **22**, 209 (1985).

[75] D.J. Fall und K.D. Luks, J. Chem. Eng. Data **30**, 276 (1985).

[76] J.D. Hottovy, K.D. Luks und J.P. Kohn, J. Chem. Eng. Data **26**, 256 (1981).

[77] U. K. Deiters und I. L. Pegg, J. Chem. Phys. **90**, 6632 (1989).

[78] T. Charoensombut-Amon, R.J. Martin und R. Kobayashi, Fluid Phase Equilibria **31**, 89 (1986).

[79] B. Rathke, Fortschritts-Berichte VDI **770** (2003).

[80] M. Müller, L.G. MacDowell, P. Virnau und K. Binder, J. Chem. Phys. **117**, 5480 (2002).

[81] M.S. Wertheim, J. Chem. Phys. **87**, 7323 (1987).

[82] W.G. Chapman, G. Jackson und K.E. Gubbins, J. Chem. Phys. **65**, 1057 (1988).

[83] L.G. MacDowell, M. Müller, C. Vega und K. Binder, J. Chem. Phys. **113**, 419 (2000).

[84] F.J. Blas und L.F. Vega, Mol. Phys. **92**, 135 (1997).

[85] D. Ghonasgi und W.G. Chapman, AIChE J. **40**, 878 (1994).

[86] L. V. Yelash und T. Kraska, Phys. Chem. Chem. Phys. **1**, 4315 (1999).

[87] K. Kaski, K. Binder und J.D. Gunton, Phys. Rev. B **29**, 3996 (1984).

[88] M. Biskup, L. Chayes und R. Kotecky, Europhys. Lett. **60**, 21 (2002).

[89] T. Neuhaus und J. Hager, eingereicht bei J. Stat. Phys. (2003).

[90] C.M. Stafford, T.P. Russell und T.J. McCarthy, Macromolecules **32**, 7610 (1999).

[91] S. Auer und D. Frenkel, Nature **409**, 1020 (2001).

[92] P.R. ten Walde und D. Frenkel, J. Chem. Phys. **109**, 9901 (1998).

[93] L.G. MacDowell, P. Virnau, M. Müller und K. Binder, in Vorbereitung (2003).

[94] F.H. Stillinger, J. Chem. Phys. **38**, 2808 (1963).

[95] B. Widom, J. Chem. Phys. **39**, 2802 (1963).

[96] K. Binder und D. Staufer, Adv. Phys. **25**, 343 (1976).

[97] J.D. Gunton, M. San Miguel und P.S. Sahni, *Phase Transitions and Critical Phenomena (ed. by C.Domb and J.L Lebowitz)*, Band 8 (Academic Press, London, 1983).

[98] K. Binder und P. Fratz, *Phase Transformations of Materials (ed. by G. Kostorz)* (Wiley-VHC, Weinheim, 2001).

[99] S. Metzger, *Monte Carlo-Simulationen zum Adsorptionsverhalten von Homo- und Copolymerlösungen* (Dissertation, Universität Mainz, 2002).

[100] Y. Lin, M. Müller und K. Binder, in Vorbereitung (2003).

[101] P. Virnau und M. Müller, in Vorbereitung (2003).

[102] R. Vink und K. Binder, in Vorbereitung (2003).

[103] P. Virnau, M. Müller, L.G. MacDowell und K. Binder, Computer Physics Communications **147**, 378 (2002).

[104] P. Virnau, M. Müller, L.G. MacDowell und K. Binder, eingereicht bei New Journal of Physics (2003).

[105] P. Virnau, M.Müller, L.G. MacDowell und K. Binder, in Vorbereitung (2003).

[106] P. Virnau und M. Müller, in Vorbereitung (2003).

[107] P. Virnau, L.G. MacDowell, M. Müller und K. Binder, eingereicht bei Computer Simulation Studies in Condensed Matter Physics XVI (2003).