

# Entwicklung und Evaluierung von Parallelisierungsstrategien für Particle-In-Cell Simulationen auf Multicomputern

Vom Fachbereich Elektrotechnik und Informationstechnik  
der Technischen Universität Darmstadt

zur Erlangung  
der Würde eines Doktor-Ingenieurs (Dr.-Ing.)  
genehmigte

DISSERTATION

von

Dipl.-Ing. Felix Wolfheimer

geboren am 28. April 1978 in Frankfurt/M.

Darmstadt 2008

Referent: Prof. Dr.-Ing. Thomas Weiland

Korreferent: Prof. Dr. rer. nat. Michael Schäfer

Tag der Einreichung: 15.04.2008

Tag der mündlichen Prüfung: 10.07.2008

D 17

Darmstädter Dissertation



---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>5</b>
1.1	Motivation . . . . .	5
1.2	Literaturübersicht . . . . .	6
1.3	Ziele der Arbeit . . . . .	7
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>11</b>
2.1	Die Maxwellsche Theorie der Elektrodynamik . . . . .	11
2.1.1	Modellierung von Vielteilchensystemen . . . . .	14
2.2	Die Methode der Finiten Integration . . . . .	16
2.2.1	Die Gitter-Maxwellgleichungen . . . . .	16
2.2.2	Materialbeziehungen . . . . .	20
2.2.3	Zeitintegration . . . . .	21
2.3	Der Particle-In-Cell (PIC) Algorithmus . . . . .	22
2.4	Paralleles Rechnen . . . . .	25
2.4.1	Architektur von Parallelrechnern . . . . .	25
2.4.2	Programmiermodelle für Parallelrechner . . . . .	29
2.4.3	Analyse paralleler Algorithmen . . . . .	33
2.4.4	Modelle für Parallelrechner . . . . .	44
2.4.5	Im Rahmen der Arbeit verwendetes Rechnermodell . . . . .	47
<b>3</b>	<b>Parallelisierungsstrategien für PIC</b>	<b>51</b>
3.1	Lastverteilung für inhomogene Parallelrechner . . . . .	51
3.2	Das Optimierungsproblem . . . . .	52
3.3	Parallelisierung des Feldlösers . . . . .	53
3.3.1	Rekursive Koordinaten Bisektionierung . . . . .	58
3.3.2	Skalierbarkeitsanalyse . . . . .	61
3.3.3	Verstecken der Latenzzeit . . . . .	63
3.4	Parallelisierung des PIC Algorithmus . . . . .	63
3.4.1	Erweiterung des Optimierungsproblems . . . . .	63
3.4.2	Behandlung des Lastbalancierungsaufwandes . . . . .	70

3.4.3	Dynamische Teilchenzuordnung . . . . .	71
3.4.4	Adaptive Bounding Box . . . . .	82
3.4.5	Statische Gebietszerlegung . . . . .	84
3.4.6	Dynamische Gebietszerlegung . . . . .	87
<b>4</b>	<b>Benchmark- und Simulationsergebnisse</b>	<b>103</b>
4.1	Benchmarkergebnisse . . . . .	103
4.1.1	Feldlöser Benchmarks . . . . .	104
4.1.2	PIC Benchmarks . . . . .	106
4.2	Der PITZ-Injektor . . . . .	119
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>125</b>
<b>A</b>	<b>Notation</b>	<b>127</b>
<b>B</b>	<b>Abkürzungen</b>	<b>131</b>
<b>C</b>	<b>Literaturübersicht</b>	<b>133</b>
C.1	Parallelisierungsstrategien für Feldlöser . . . . .	133
C.1.1	Graphenbasierte Heuristiken . . . . .	134
C.1.2	Geometrische Heuristiken . . . . .	136
C.2	Parallelisierungsstrategien für PIC . . . . .	137
C.2.1	Parallelisierungsstrategien mit statischer Lastbalancierung . . . . .	138
C.2.2	Parallelisierungsstrategien mit dynamischer Lastbalancierung . . . . .	139
<b>D</b>	<b>Optimalitätsbeweis</b>	<b>141</b>
D.1	Konvexität der Menge der zulässigen Lösungen . . . . .	141
D.2	Konvexität der Zielfunktion . . . . .	141
D.3	Beweis der Optimalität . . . . .	142
	<b>Literaturverzeichnis</b>	<b>145</b>
	<b>Stichwortverzeichnis</b>	<b>155</b>
	<b>Danksagung</b>	<b>159</b>
	<b>Lebenslauf</b>	<b>161</b>

---

# 1 Einleitung

## 1.1 Motivation

Bei dem Prozess des Entwurfs, der Entwicklung und der Optimierung moderner technischer Geräte und Anlagen sind rechnergestützte Simulationen nicht nur zu einer wichtigen Informationsquelle und Entscheidungsgrundlage geworden, sondern ermöglichen häufig sogar, den zeitlich wie wirtschaftlich aufwändigen Bau von Prototypen zur Evaluierung verschiedener Designalternativen auf ein Minimum zu reduzieren und so den gesamten Entwicklungszyklus zu verkürzen und kostengünstiger zu gestalten. Zudem ermöglichen Simulationen Einblicke in Systeme, die für Messungen aus verschiedenen Gründen schwer oder gar nicht zugänglich sind, und stören die zu untersuchenden Anordnungen nicht durch das Einbringen von Messgeräten.

Begünstigt durch die rasante Entwicklung der Informationstechnik in den vergangenen Jahren sind heute, mit Hilfe von Rechnern des Massenmarktes, Simulationen in einer Genauigkeit und Geschwindigkeit möglich, die vor einigen Jahren noch undenkbar gewesen sind oder zumindest nur auf spezialisierten Hochleistungsrechnern durchführbar waren. Trotz dieser Entwicklung existieren gegenwärtig viele praxisrelevante Problemstellungen aus dem Bereich der rechnergestützten Simulation, welche ohne den Einsatz von Hochleistungsrechnern unlösbar erscheinen oder zumindest zu unverträglich langen Rechenzeiten führen.

Aufgrund ihres hervorragenden Verhältnisses von Preis zu erzielbarer Rechenleistung sind insbesondere Hochleistungsrechner populär geworden, welche sich aus einer Ansammlung von unabhängigen Einzelrechnern zusammensetzen, die über ein Verbindungsnetzwerk miteinander verbunden sind, sogenannte Cluster. Die Nutzung von Hochleistungsrechnern dieses Typs zur Lösung eines Simulationsproblems erfordert die Parallelisierung der zugrunde liegenden numerischen Algorithmen und, damit verbunden, die Verteilung der Simulationsdaten auf die voneinander unabhängigen Speicher der Einzelrechner. Eine sorgfältige Analyse des von den Algorithmen erzeugten Speicherzugriffsmusters sowie der Abhängigkeit der Berechnungsschritte voneinander sind essentiell, um die von dem Hochleistungsrechner zu Verfügung gestellten Ressourcen effizient nutzen bzw. die gewünschten

Berechnungen überhaupt durchführen zu können.

Für das Gebiet der Elektrodynamik stellt insbesondere die im Rahmen dieser Arbeit betrachtete selbstkonsistente Simulation geladener Teilchen unter dem Einfluss äußerer wie auch von den Teilchen selbst erzeugter elektromagnetischer Felder für viele praxisrelevante Systeme eine große Herausforderung dar. Die selbstkonsistente Simulation solcher Problemstellungen kann mit Hilfe des Particle-In-Cell Algorithmus erfolgen. Dieser führt insbesondere für diejenigen Problemstellungen aus der Beschleunigerphysik, welche aufgrund fehlender Symmetrie zu ihrer Lösung einer Simulation in drei Dimensionen bedürfen, häufig zu einer derart großen Menge an Daten sowie Berechnungsoperationen, dass die Verwendung eines Hochleistungsrechners sinnvoll erscheint, wenn nicht gar unumgänglich ist, um die Simulation überhaupt durchführen zu können. Die große praktische Bedeutung von Simulationen diesen Typs sowie der mit ihnen verbundene immense Rechenaufwand bildeten die Motivation, im Rahmen dieser Arbeit verschiedene Parallelisierungsstrategien für den Particle-In-Cell Algorithmus zu entwickeln, theoretisch zu untersuchen, zu implementieren und deren Performance für ein Simulationsproblem aus der Beschleunigertechnik sowie auf typischen Simulationsproblemen basierten Benchmarks zu vergleichen.

## 1.2 Literaturübersicht

Das bei der Parallelisierung von Algorithmen auftretende Problem, die Rechenoperationen sowie die Daten derart den verfügbaren Prozessoren des verwendeten Parallelrechners zuzuordnen, dass die resultierende Laufzeit bis zum Abschluss der Berechnungen möglichst minimal wird, hat in den vergangenen Jahren große Aufmerksamkeit erfahren. Dies gilt auch für den Bereich der numerischen Simulation elektromagnetischer Felder sowie der selbstkonsistenten Simulation der Dynamik geladener Teilchen mit Hilfe des PIC Algorithmus. Der vorliegende Abschnitt stellt eine Würdigung der wichtigsten Parallelisierungsstrategien für PIC Simulationen dar, die in den vergangenen Jahren in der Fachliteratur vorgeschlagen wurden. Für eine ausführlichere Übersicht sei auf Anhang C verwiesen.

Um durch die Parallelisierung eines Algorithmus die Laufzeit effizient zu verringern, sind im Wesentlichen zwei Grundregeln zu beachten, die einen großen Einfluss auf den schlussendlich erzielten Laufzeitgewinn besitzen. Zum Einen minimiert die Zuordnung der Berechnungsoperationen auf die Prozessoren im Verhältnis der Leistungsfähigkeit derselben die Rechenzeit (balancierte Rechenlast) und ist somit essentiell, um die Ressourcen des Parallelrechners effizient zu nutzen. Zum Anderen führt eine Zuordnung der Daten derart, dass die zwischen den Prozessoren auszutauschende Datenmenge während der Berechnungen möglichst gering wird, zu einer Verringerung der Laufzeit.

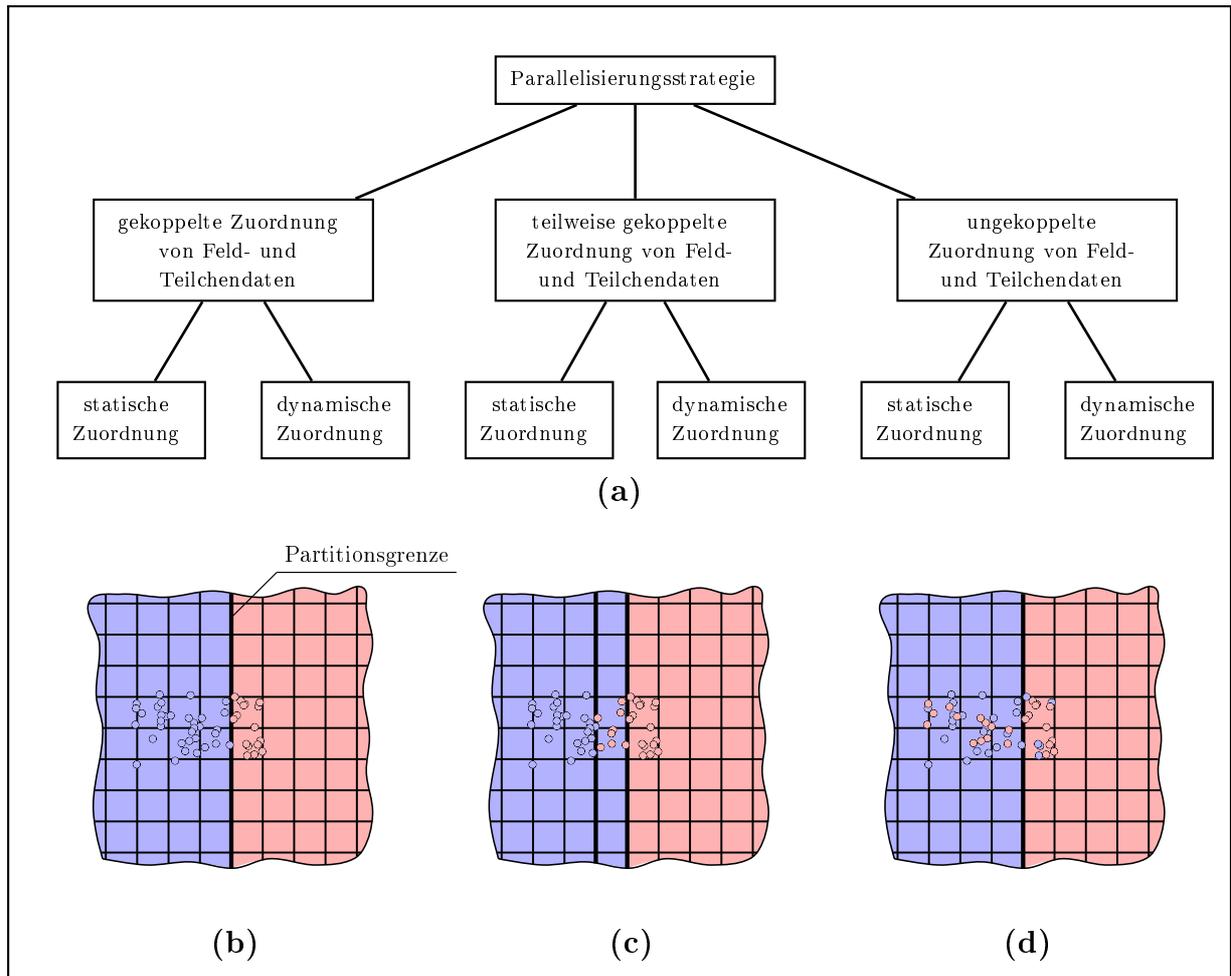
Basierend auf diesen zwei Grundregeln lassen sich die Parallelisierungsstrategien für PIC dahingehend klassifizieren, ob sie vorrangig den Datenaustausch minimieren, oder in erster Linie die Rechenlast balancieren. Zur ersten Kategorie gehören Parallelisierungsstrategien, welche die Zuordnung von Feld- und Teilchendaten zu Prozessoren aneinander koppeln. Zur zweiten Kategorie gehören diejenigen Strategien, die eine solche Kopplung nicht vornehmen. Eine weitere Untergliederung kann dahingehend erfolgen, ob die Strategie die Zuordnung der Daten zu den Prozessoren während der Laufzeit verändert, um auch die zweite Grundregel zu erfüllen. Man spricht in diesem Fall auch von einer dynamischen Zuordnung der Rechenlast. Abbildung 1.1 illustriert diese Klassifizierung. Parallelisierungsstrategien, die auf einer ungekoppelten Zuordnung von Feld- und Teilchendaten basieren, werden in [1, 2, 3, 4] vorgestellt. Für alle diese Strategien erfolgt die Zuordnung der Feld- und Teilchendaten statisch, so dass deren Leistungsfähigkeit durch große Datenmengen, die in jedem Zeitschritt auszutauschen sind, limitiert ist (siehe insbesondere [1, 2]).

Strategien, die auf der gekoppelten Zuordnung von Feld- und Teilchendaten basieren, liefern für Simulationsprobleme mit homogener Teilchenverteilung im Fall statischer Lastzuordnung bereits sehr gute Ergebnisse (siehe z.B. [5, 6, 7, 8]). Für den Fall lokalisierter Teilchenverteilungen wurden Strategien vorgeschlagen, die durch eine dynamische Zuordnung der Feld- und Teilchendaten zu den Prozessoren versuchen, die balancierte Zuordnung der Berechnungsoperationen bei gleichzeitig möglichst geringem Kommunikationsvolumen zu gewährleisten (siehe z.B. [9, 10, 11, 12]). Diese Strategien haben sich für den im Rahmen dieser Arbeit betrachteten Parallelrechner Typ als die erfolgreichsten erwiesen.

Weiterhin existieren Mischformen, welche die Zuordnung von Teilchen- und Felddaten zwar gebietsorientiert, jedoch nicht direkt aneinander gekoppelt vornehmen (siehe z.B. [13, 14, 15, 16]).

## 1.3 Ziele der Arbeit

Obwohl in den vergangenen Jahren einige Parallelisierungsstrategien für Particle-In-Cell Simulationen in der Literatur vorgeschlagen wurden, finden sich kaum theoretische Untersuchungen oder aussagekräftige Benchmarkanalysen derselben, die jedoch beide zur adäquaten Auswahl einer geeigneten Parallelisierungsstrategie für eine konkrete Klasse von Simulationsproblemen unabdingbar sind. Ein wesentliches Ziel der Arbeit ist daher, die theoretische sowie praktische Analyse von Parallelisierungsstrategien mit Hilfe aussagekräftiger Benchmarkprobleme, um die Stärken und Schwächen der verschiedenen Strategien aufzuzeigen. Die im Rahmen der Arbeit untersuchten und implementierten Parallelisierungsstrategien decken die gesamte in Abbildung 1.1 gezeigte Bandbreite ab.



**Abbildung 1.1:** Zur Illustration der Klassifizierung von Parallelisierungsstrategien für PIC Simulationen nach der Zuordnung von Feld- und Teilchendaten auf die Prozessoren. (a) zeigt eine schematische Übersicht über die verschiedenen Strategien, während die unteren Abbildungen für jede Strategie eine mögliche Zuordnung von Feld- und Teilchendaten zeigen. Die farbliche Markierung symbolisiert die Prozessorzugehörigkeit. Bei einer Parallelisierungsstrategie, welche die Zuordnung von Feld- und Teilchendaten aneinander koppelt, wie unter (b) für einen Gitterausschnitt gezeigt, ist die Menge der in jedem Zeitschritt auszutauschenden Daten gering, jedoch verändert sich im allgemeinen die Verteilung der Rechenlast auf die Prozessoren im Verlauf der Simulation, was eine dynamische Neuordnung der Feld- und Teilchendaten erforderlich macht. Parallelisierungsstrategien, die Teilchen- und Felddaten unabhängig voneinander auf die Prozessoren verteilen, wie unter (d) gezeigt, führen zu einer gleichmäßig verteilten Rechenlast, jedoch auch zu einer großen Menge an Daten, welche in jedem Zeitschritt ausgetauscht werden müssen. Gemischte Parallelisierungsstrategien, wie unter (c) gezeigt, versuchen einen Kompromiss zwischen dem Ziel der Lastbalancierung einerseits und der Minimierung des Datenaustausches andererseits zu finden.

Die theoretische Analyse der verschiedenen Parallelisierungsstrategien trägt dabei zum besseren Verständnis des Laufzeitverhaltens derselben bei. So konnte durch die Analyse zweier Parallelisierungsstrategien mit ungekoppelter Zuordnung von Feld- und Teilchen-daten, von denen eine im Rahmen der Arbeit entwickelt wurde, und die zweite die Abwandlung einer in der Literatur vorgeschlagenen Strategie darstellt, gezeigt werden, dass Strategien, die auf diesem Prinzip basieren, an grundsätzlichen Schwierigkeiten leiden, die eine zufriedenstellende Performance häufig verhindern.

Eine Stärke von seriellen PIC Simulationen auf strukturierten Gittern ist, dass die Datenstrukturen sowohl einfach als auch effizient zu implementieren sind. Ein wesentliches Ziel bei den im Rahmen der Arbeit entwickelten Parallelisierungsstrategien war die weitestgehende Bewahrung dieser effizienten Datenstrukturen, um den durch die Parallelisierung erzielten Laufzeitgewinn nicht durch die Einführung unnötig komplizierter Datenstrukturen und deren Verwaltung zu gefährden.

Die Parallelisierungsstrategien wurden alle für Parallelrechner mit Prozessoren beliebiger Leistungsfähigkeit formuliert, so dass eine Implementierung auf Parallelrechnern mit inhomogener Hardwareausstattung möglich wird.



---

## 2 Theoretische Grundlagen

Das folgende Kapitel beinhaltet eine Einführung in die für die Arbeit relevanten Theorien. Abschnitt 2.1 gibt eine Übersicht über die klassische Elektrodynamik unter Berücksichtigung der Modellierung von Problemstellungen, welche eine große Anzahl beweglicher Ladungsträger beinhalten. Die Gleichungen zur Modellierung derartiger Problemstellungen sind für praxisrelevante Anordnungen nur numerisch lösbar. Die Abschnitte 2.2 und 2.3 stellen die im Rahmen der Arbeit verwendeten numerischen Lösungsmethoden vor. Insbesondere wird der Particle-In-Cell Algorithmus erläutert, dessen Parallelisierung den Schwerpunkt der Arbeit bildet. In Abschnitt 2.4 werden schließlich die wichtigsten Begriffe im Zusammenhang mit dem parallelen Rechnen erläutert. Außerdem wird das zur Entwicklung und theoretischen Analyse der parallelen Algorithmen verwendete Rechnermodell vorgestellt.

### 2.1 Die MAXWELLSche Theorie der Elektrodynamik

Die von JAMES CLERK MAXWELL im 19. Jahrhundert entwickelte mathematische Beschreibung des Elektromagnetismus [17, 18] bildet die Grundlage der Modellierung einer Fülle von Problemstellungen der Physik und der Elektrotechnik. In der heute üblichen Schreibweise lassen sich die MAXWELLSchen Gleichungen für den Fall ruhender Medien in ihrer integralen Form angeben als [19]

$$\int_{\partial A} \vec{E}(\vec{r}, t) \cdot d\vec{s} = - \int_A \frac{\partial \vec{B}(\vec{r}, t)}{\partial t} \cdot d\vec{A}, \quad (2.1)$$

$$\int_{\partial A} \vec{H}(\vec{r}, t) \cdot d\vec{s} = \int_A \left( \vec{J}(\vec{r}, t) + \frac{\partial \vec{D}(\vec{r}, t)}{\partial t} \right) \cdot d\vec{A}, \quad (2.2)$$

$$\int_{\partial V} \vec{D}(\vec{r}, t) \cdot d\vec{A} = \int_V \varrho(\vec{r}, t) dV, \quad (2.3)$$

$$\int_{\partial V} \vec{B}(\vec{r}, t) \cdot d\vec{A} = 0. \quad (2.4)$$

Hierbei sind  $\vec{E}$ ,  $\vec{D}$ ,  $\vec{H}$  und  $\vec{B}$  die Feldgrößen, die das elektromagnetische Feld innerhalb eines vorgegebenen Gebietes  $\Omega \subseteq \mathbb{R}^3$  für ein vorgegebenes Zeitintervall  $[t_s; t_e] \subseteq \mathbb{R}$  beschreiben. Die Integralzusammenhänge (2.1)-(2.4) müssen für jedes Volumen  $V \subseteq \Omega$  bzw. jede Fläche  $A \subset \Omega$  erfüllt sein. Die Argumentlisten für die Feldgrößen werden im Folgenden zu Gunsten der besseren Lesbarkeit fortgelassen.

Aus den MAXWELLSchen Gleichungen folgt die Kontinuitätsgleichung

$$\int_{\partial V} \vec{J} \cdot d\vec{A} + \int_V \frac{\partial \rho}{\partial t} dV = 0, \quad (2.5)$$

welche die Erhaltung der elektromagnetischen Ladung innerhalb des Rechengebietes beschreibt.

Ein Zusammenhang zwischen den Feldgrößen  $\vec{E}$  bzw.  $\vec{H}$  und den Flussdichten  $\vec{D}$  bzw.  $\vec{B}$  wird durch sogenannte Materialbeziehungen hergestellt. Während sich im Vakuum die Zusammenhänge

$$\vec{D} = \varepsilon_0 \vec{E} \quad \text{sowie} \quad \vec{B} = \mu_0 \vec{H}. \quad (2.6)$$

ergeben, können unter Anwesenheit von Materialien kompliziertere Zusammenhänge notwendig sein, um den Einfluss des Materials auf das elektromagnetische Feld hinreichend genau zu modellieren. Im Folgenden soll sich die Betrachtung auf lineares, isotropes und stationäres Materialverhalten beschränken, welches zur Modellierung der im Rahmen der Arbeit betrachteten Problemstellungen angenommen werden kann. Zudem werden keine Elektrete und Permanentmagnete betrachtet, welche auch ohne äußeres Feld durch die Ausrichtung von im Material vorhandenen elektrischen beziehungsweise magnetischen Dipolen ein Feld erzeugen können. Unter diesen Annahmen ergibt sich der Zusammenhang zwischen den Feldstärken und Flussdichten zu

$$\vec{D} = \varepsilon(\vec{r}) \vec{E} \quad \text{sowie} \quad \vec{B} = \mu(\vec{r}) \vec{H}. \quad (2.7)$$

Befinden sich in einem Material frei bewegliche Ladungsträger, so wird dies durch eine Leitfähigkeit  $\kappa$  modelliert. In diesem Fall existiert bei Anwesenheit eines elektrischen Feldes ein Anteil von  $\vec{J}$ , der proportional zu  $\vec{E}$  ist und somit nicht als externer Quellterm vorgegeben werden muss. Dieser Anteil wird als Leitungsstromdichte  $\vec{J}_l$  bezeichnet und es gilt

$$\vec{J}_l = \kappa(\vec{r}) \cdot \vec{E} \quad (2.8)$$

für den Zusammenhang dieser Größe mit dem elektrischen Feld.

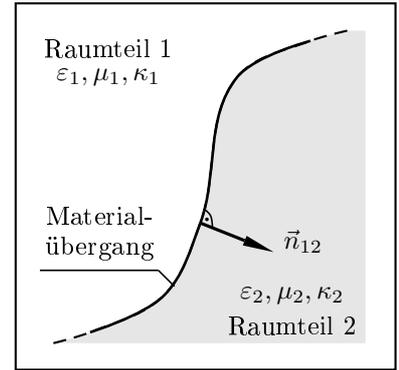
Häufig lassen sich leitfähige Materialien, insbesondere metallische Körper, in guter Näherung als Idealleiter modellieren ( $\kappa \rightarrow \infty$ ). Dies ist insbesondere dann gerechtfertigt, wenn die Materialien ohnehin schon eine hohe Leitfähigkeit besitzen und zudem im Wesentlichen

die Wechselwirkung mit hochfrequenten elektromagnetischen Feldern betrachtet wird. In den Problemstellungen aus der Beschleunigerphysik kann dies für die Strahlrohre und Kavitäten meist in guter Näherung angenommen werden. Bei den im Rahmen dieser Arbeit durchgeführten Simulationen wurden sämtliche leitfähigen Materialien daher als Idealleiter modelliert.

Häufig werden Materialverteilungen in der Weise idealisiert, dass sich Permittivität, Permeabilität und Leitfähigkeit sprunghaft ändern. An solchen idealisierten Materialgrenzflächen müssen die Felder folgende Stetigkeitsbedingungen erfüllen [19]

$$\begin{aligned} \vec{n}_{12} \cdot (\vec{D}_2 - \vec{D}_1) &= \sigma, & \vec{n}_{12} \cdot (\vec{B}_2 - \vec{B}_1) &= 0, \\ \vec{n}_{12} \times (\vec{E}_2 - \vec{E}_1) &= \vec{0}, & \vec{n}_{12} \times (\vec{H}_2 - \vec{H}_1) &= \vec{J}_F. \end{aligned} \quad (2.9)$$

Hierbei ist  $\vec{n}_{12}$  der Normaleneinheitsvektor der Grenzfläche an dem betrachteten Punkt (siehe Abb. 2.1). Die Feldwerte sind als Limites zu interpretieren, wenn man sich dem betrachteten Punkt auf der Grenzfläche von Raumteil 1 bzw. Raumteil 2 aus nähert. Die Größen  $\sigma$  bzw.  $\vec{J}_F$  bezeichnen eine Flächenladungsdichte bzw. Flächenstromdichte, mit denen die Grenzfläche belegt sein kann.



**Abbildung 2.1:** *Idealisierter Materialübergang*

In Idealleitern existieren keine elektromagnetischen Felder, so dass sich die Stetigkeitsbedingungen (2.9) vereinfachen zu

$$\begin{aligned} \vec{n} \cdot \vec{D} &= \sigma, & \vec{n} \cdot \vec{B} &= 0, \\ \vec{n} \times \vec{E} &= \vec{0}, & \vec{n} \times \vec{H} &= \vec{J}_F. \end{aligned} \quad (2.10)$$

Hier ist  $\vec{n}$  der Normaleneinheitsvektor, der an dem betrachteten Punkt auf der Materialgrenzfläche von dem Idealleiter fort zeigt, während die Felder wieder als Limites bei einer Annäherung aus dem Außenraum zu verstehen sind.

Zur Beschreibung des elektromagnetischen Feldes mit Hilfe der MAXWELLSchen Gleichungen ist neben den Materialbeziehungen noch die Feldverteilung im Rechengebiet zum Anfangszeitpunkt  $t_s$  anzugeben (Anfangsbedingung). Außerdem ist der Verlauf der Feldkomponente des elektrischen oder magnetischen Feldes auf dem Rand des Rechengebietes vorzugeben (Randbedingung). Im Fall eines im Unendlichen liegenden Randes wird das Verhalten der Felder durch die SOMMERFELDSchen Abstrahlungsbedingungen beschrieben [20].

Für die im Rahmen dieser Arbeit behandelten Problemstellungen wird der interessierende Raumteil zumindest teilweise durch einen metallischen Körper begrenzt. Bei der Annahme

einer idealen elektrischen Leitfähigkeit dieses Körpers verschwinden gemäß (2.10) die zur Materialoberfläche tangentialen elektrischen Felder, womit deren Werte für alle Zeitpunkte  $t \in [t_s; t_e]$  bekannt sind. Damit bilden Materialgrenzflächen zu Idealleitern automatisch Ränder des Rechengebietes.

Es kann gezeigt werden, dass die MAXWELLSchen Gleichungen zusammen mit den Materialbeziehungen sowie den Anfangs- und Randbedingungen ein mathematisch wohlgestelltes Problem zur Berechnung der Feldgrößen darstellen [20]

### 2.1.1 Modellierung von Vielteilchensystemen

Unter einem Vielteilchensystem soll im Rahmen der Arbeit eine große Anzahl geladener Teilchen verstanden werden, die sich unter dem Einfluss elektromagnetischer Felder im freien Raum bewegen. Dadurch, dass die geladenen Teilchen nicht nur von einem äußeren elektromagnetischen Feld beeinflusst werden, sondern selbst auch ein elektromagnetisches Feld, das sogenannte Raumladungsfeld, erzeugen, müssen zur Modellierung eines Vielteilchensystems die MAXWELLSchen Gleichungen, die die Dynamik des elektromagnetischen Feldes beschreiben, mit den Bewegungsgleichungen der Teilchen gekoppelt werden. Die Kraft auf ein punktförmiges Teilchen der Ladung  $q$  und der relativistischen Masse

$$m := \gamma m_0 \quad \text{mit} \quad \gamma := \frac{1}{\sqrt{1 - \left(\frac{v}{c}\right)^2}}, \quad (2.11)$$

welches sich mit dem mechanischen Impuls  $\vec{p}$  in einem elektromagnetischen Feld bewegt, ist gegeben durch die NEWTON-LORENTZ-Gleichung [19]

$$\vec{F}(t) \equiv \frac{d\vec{p}}{dt} = q \left( \vec{E}_p + \frac{\vec{p}}{m} \times \vec{B}_p \right). \quad (2.12)$$

Bei den Größen  $\vec{E}_p$  und  $\vec{B}_p$  handelt es sich um die Werte des elektromagnetischen Feldes zum Zeitpunkt  $t$  am Ort des Teilchens. Zusammen mit der Beziehung

$$\vec{p} = m \frac{d\vec{r}}{dt} = m\vec{v}, \quad (2.13)$$

wobei  $\vec{r}$  die Position des Teilchens beschreibt, bildet sie die Bewegungsgleichungen für punktförmige, geladene Teilchen in einem elektromagnetischen Feld. Alle in den Gleichungen (2.12) und (2.13) vorkommenden Größen, mit Ausnahme der Ladung  $q$ , sind zeitveränderlich. Die Argumente wurden jedoch zugunsten der besseren Lesbarkeit fortgelassen. Für ein Ensemble von  $P$  punktförmigen Teilchen ergibt sich für die Quellterme  $\rho$  und  $\vec{J}$

in den MAXWELLSchen Gleichungen

$$\begin{aligned} \varrho &= \sum_{p=1}^P q_p \cdot \delta(\vec{r} - \vec{r}_p), \\ \vec{J} &= \sum_{p=1}^P q_p \vec{v}_p \cdot \delta(\vec{r} - \vec{r}_p), \end{aligned} \tag{2.14}$$

wobei  $\delta(\cdot)$  die DIRAC'sche Distribution bezeichnet. Die Menge  $\Gamma \subseteq \mathbb{R}^{6P}$ , welche die möglichen Werte der Orts- und Geschwindigkeitsvariablen enthält, wird als Phasenraum des Systems bezeichnet.

Weder für analytische Berechnungen noch für numerische Lösungen von Vielteilchenproblemen ist es jedoch in den meisten Fällen praktikabel, jedes reale Teilchen durch ein Teilchen im mathematischen Modell zu repräsentieren, da meistens nicht die Dynamik der einzelnen Teilchen von Interesse ist, sondern vielmehr das kollektive Verhalten einer Teilchenverteilung. Man weicht daher auf eine statistische Beschreibung aus, die besser geeignet ist, das interessierende kollektive Verhalten zu repräsentieren.

Führt man eine Wahrscheinlichkeitsdichtefunktion

$$f_P(\vec{r}_1 \dots \vec{r}_P, \vec{p}_1 \dots \vec{p}_P, t) : \Gamma \times \mathbb{R} \rightarrow [0; 1] \tag{2.15}$$

auf dem Phasenraum ein, so dass

$$\int_{\Delta\Gamma} f_P \, d\Gamma \tag{2.16}$$

die Wahrscheinlichkeit angibt, das System zu einem Zeitpunkt  $t$  in dem Bereich  $\Delta\Gamma \subseteq \Gamma$  des Phasenraumes anzutreffen, so genügt  $f_P$  der LIOUVILLE-Gleichung [21]

$$\frac{df_P}{dt} \equiv \frac{\partial f_P}{\partial t} + \sum_{p=1}^P \frac{\partial \vec{r}_p}{\partial t} \cdot \text{grad}_{\vec{r}_p} f_P + \sum_{p=1}^P \frac{\partial \vec{p}_p}{\partial t} \cdot \text{grad}_{\vec{p}_p} f_P = 0. \tag{2.17}$$

Die Funktion  $f_P$  hängt von allen  $6P$  Phasenraumkoordinaten des Systems ab und bietet daher zunächst noch keine Vereinfachung des Problems. Oftmals ist jedoch nur von Interesse, wie groß die Wahrscheinlichkeit ist, irgendeines der  $P$  Teilchen zu einem Zeitpunkt  $t$  in einem Bereich des Phasenraumes anzutreffen. Diese Wahrscheinlichkeit wird beschrieben durch die Einteilchenverteilungsfunktion  $f_1(\vec{r}, \vec{p}, t)$ . Für den häufig auftretenden Fall eines Vielteilchensystems, welches nur gleichartige Teilchen enthält, erfüllt  $f_1$  bei Vernachlässigung von Stoßeffecten die VLASOV-Gleichung, welche aus der LIOUVILLE-Gleichung abgeleitet werden kann [21]

$$\frac{df_1}{dt} \equiv \frac{\partial f_1}{\partial t} + \frac{\vec{p}}{m} \cdot \text{grad}_{\vec{r}} f_1 + q \left( \vec{E} + \frac{\vec{p}}{m} \times \vec{B} \right) \cdot \text{grad}_{\vec{p}} f_1 = 0. \tag{2.18}$$

Die charakteristischen Kurven der Gleichung (2.18) entsprechen möglichen Teilchentrajektorien für ein Teilchenensemble mit der Verteilungsfunktion  $f_1$  [22]. Es sei noch bemerkt, dass die zeitliche Änderung von  $f_1$ , welche einer Bewegung der Teilchen entspricht, die in der Gleichung auftretenden Feldgrößen  $\vec{E}$  und  $\vec{B}$  beeinflusst.

## 2.2 Die Methode der Finiten Integration

Außer für einige Spezialfälle sind die in Abschnitt 2.1 eingeführten MAXWELLSchen Gleichungen (2.1)-(2.4) sowie die VLASOV-Gleichung (2.18) einer analytischen Lösung nicht zugänglich. Für praxisrelevante Problemstellungen kann ihre Lösung daher nur näherungsweise unter Anwendung numerischer Lösungsmethoden erfolgen.

Die Methode der finiten Integration (engl. **F**inite **I**ntegration **T**echnique, FIT) erlaubt für eine Vielzahl von Problemstellungen eine solche näherungsweise Berechnung der elektromagnetischen Felder [23, 24] und, in Verbindung mit dem Particle-In-Cell Algorithmus, auch die Simulation eines Vielteilchensystems [25]. Sie basiert auf der integralen Form der MAXWELLSchen Gleichungen wie in (2.1)-(2.4) angegeben.

Jedes numerische Verfahren zur Lösung einer Integral- oder Differentialgleichung erfordert eine Darstellung der gesuchten Lösung mit Hilfe endlich vieler Freiheitsgrade. Man spricht dabei von der Diskretisierung des kontinuierlichen Problems. Verschiedene numerische Verfahren unterscheiden sich darin, wie diese Darstellung konstruiert wird. Für die Methode der Finiten Integration wird diese durch die sogenannten Gitter-MAXWELLgleichungen gebildet.

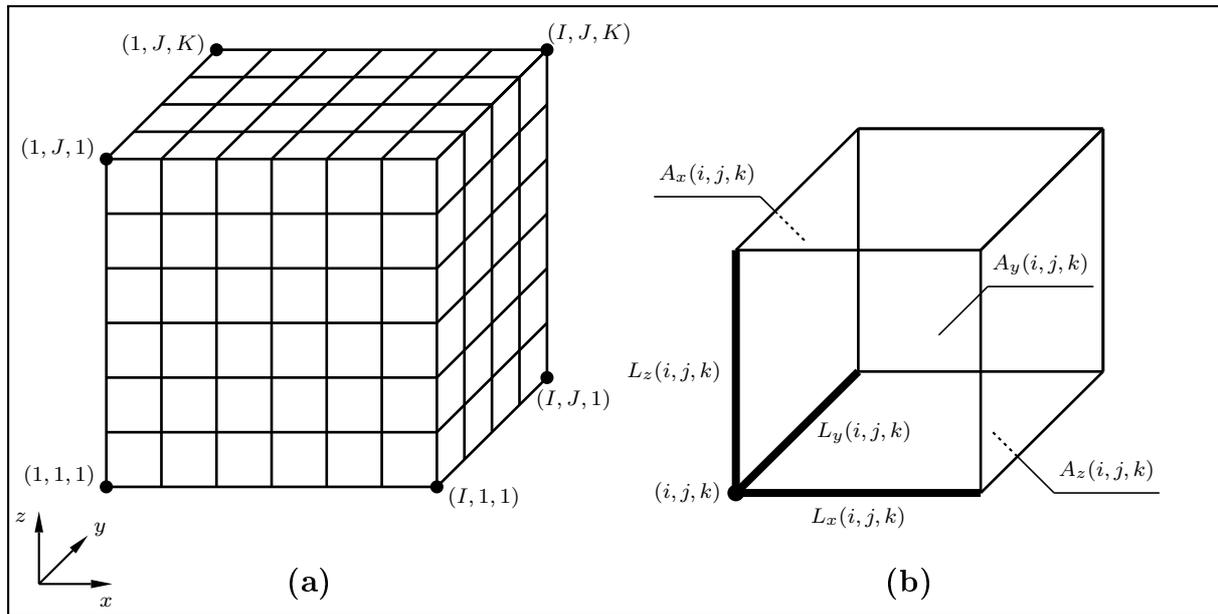
### 2.2.1 Die Gitter-MAXWELLgleichungen

Zur Konstruktion der Gitter-MAXWELLgleichungen wird das Rechengebiet  $\Omega$  in  $N_c$  disjunkte, zusammenhängende Teilgebiete  $\Omega_{1\dots N_c}$  unterteilt. Diese Zerlegung wird als Rechengitter oder einfach als Gitter  $G$  bezeichnet. Die Teilgebiete werden als Gitterzellen bezeichnet. Obwohl die Form der Gitterzellen zunächst vollkommen beliebig ist, haben sich strukturierte Rechengitter, welche sich für drei Raumdimensionen als Tensorprodukt dreier eindimensionaler Gitter konstruieren lassen, für viele Anwendungen bewährt, da sie zu einfachen und effizient zu implementierenden Algorithmen führen. Die Gitterzellen ergeben sich bei einer Diskretisierung in der beschriebenen Form als Quader. Abbildung 2.2(a) zeigt ein nach diesem Prinzip erstelltes Gitter im Falle dreier Raumdimensionen. Durch diese Einschränkung bei der Konstruktion des Gitters ist es im Allgemeinen nicht mehr möglich, das analytische Rechengebiet  $\Omega$  exakt durch das Gitter nachzubilden. Das Gitter

wird dann so groß gewählt, dass es den interessierenden Raumbereich vollständig überdeckt.

Durch die Konstruktion des Gitters als Tensorprodukt eindimensionaler Gitter wird es möglich, alle Kanten, Flächen und Volumina durch ein einfaches Indizierungsschema eindeutig zu referenzieren. Im Fall dreier Raumdimensionen erfolgt dies durch drei Indizes. Die Indizierung für Flächen  $A$ , Kanten  $L$  und Volumina  $V$  ist Abbildung 2.2(b) zu entnehmen.

Die Grundidee der FIT ist das Erzwingen der Integralzusammenhänge (2.1)-(2.4) für die



**Abbildung 2.2:** (a) illustriert ein kartesisches Gitter für drei Raumdimensionen, welches sich aus dem Tensorprodukt dreier eindimensionaler Gitter ergibt. (b) zeigt den Zusammenhang zwischen der Indizierung der Gitterpunkte, -kanten und -flächen.

durch die Gitterzellen festgelegten Flächen und Volumina. Die Integralwerte, welche im Falle von Linienintegralen als Gitterspannungen und im Falle von Flächenintegralen als Gitterflüsse bezeichnet werden, bilden die Freiheitsgrade bei diesem Verfahren. Die elektrischen Gitterspannungen  $\bar{e}$  werden definiert gemäß

$$\bar{e}_\nu(i, j, k) := \int_{L_\nu(i, j, k)} \vec{E} \cdot \vec{e}_\nu \, d\nu, \quad (2.19)$$

und sind somit den Kanten des Gitters zugeordnet, wobei  $\nu \in \{x, y, z\}$  die Raumrichtung bezeichnet. Die magnetischen Gitterflüsse  $\bar{b}$ , welche den Flächen des Gitters zugeordnet

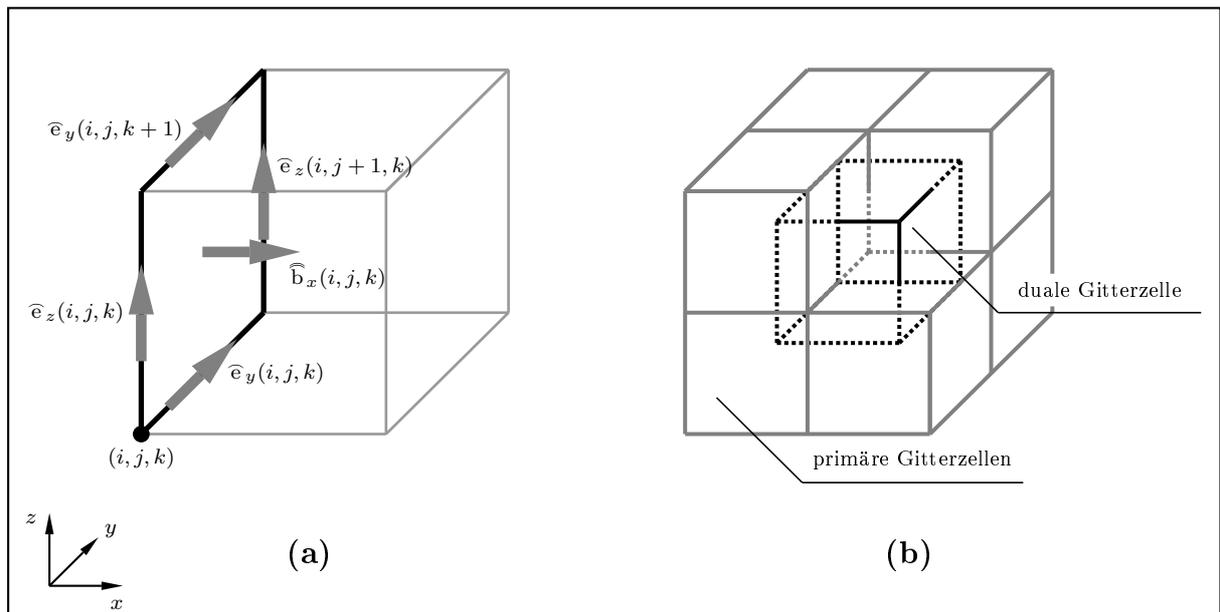
sind, werden definiert als

$$\widehat{\mathbf{b}}_\nu(i, j, k) := \int_{A_\nu(i, j, k)} \vec{B} \cdot \vec{e}_\nu dA. \quad (2.20)$$

Damit lässt sich für jede Gitterfläche ein Zusammenhang zwischen dem magnetischen Gitterfluss, welcher der Fläche zugeordnet ist, und den elektrischen Gitterspannungen, welche den vier die Fläche berandenden Gitterkanten zugeordnet sind, aufstellen. Es folgt beispielhaft für eine Gitterfläche  $A_x(i, j, k)$  mit Normale in  $x$ -Richtung

$$\widehat{e}_y(i, j, k) - \widehat{e}_y(i, j, k + 1) - \widehat{e}_z(i, j, k) + \widehat{e}_z(i, j + 1, k) = -\frac{d}{dt} \widehat{\mathbf{b}}_x(i, j, k). \quad (2.21)$$

Die Vorzeichen in Gleichung (2.21) ergeben sich durch die in den Gleichungen (2.19) und (2.20) eingeführte Orientierung der Gitterkanten und -flächen, wie in Abbildung 2.3(a) illustriert. Eine Gleichung dieser Form lässt sich für jede der Gitterflächen angeben. Wer-



**Abbildung 2.3:** (a) illustriert den in Gleichung (2.21) angegebenen Zusammenhang. (b) illustriert die Konstruktion des dualen Gitters  $\widetilde{G}$  aus dem primären Gitter  $G$ .

den die Gitterspannungen  $\widehat{e}$  bzw. die Gitterflüsse  $\widehat{\mathbf{b}}$  auf bestimmte Weise in Vektoren  $\widehat{\mathbf{e}}$  bzw.  $\widehat{\mathbf{b}}$  angeordnet, so können alle sich ergebenden Gleichungen in einer einzigen vektoriellen Gleichung zusammengefasst werden [26]. Es folgt

$$\mathbf{C}\widehat{\mathbf{e}} = -\frac{d}{dt}\widehat{\mathbf{b}}. \quad (2.22)$$

Die Matrix  $\mathbf{C}$  wählt die Gitterspannungen vorzeichenrichtig aus und wird allein durch die Topologie des Gitters bestimmt.

Auf die gleiche Weise kann auch Gleichung (2.4) für die durch die Gitterzellen gebildeten Volumina angegeben werden. Für eine Gitterzelle  $i, j, k$  ergibt sich

$$\begin{aligned} & \widehat{\mathbf{b}}_x(i+1, j, k) - \widehat{\mathbf{b}}_x(i, j, k) \\ & + \widehat{\mathbf{b}}_y(i, j+1, k) - \widehat{\mathbf{b}}_y(i, j, k) \\ & + \widehat{\mathbf{b}}_z(i, j, k+1) - \widehat{\mathbf{b}}_z(i, j, k) = 0. \end{aligned} \quad (2.23)$$

In einer kompakten Schreibweise mit Hilfe des Vektors  $\widehat{\mathbf{b}}$  und einer topologischen Matrix  $\mathbf{S}$ , welche die entsprechenden magnetischen Gitterflüsse vorzeichenrichtig miteinander kombiniert, ergibt sich

$$\mathbf{S}\widehat{\mathbf{b}} = 0. \quad (2.24)$$

Zur Behandlung der Gleichungen (2.2) und (2.3) wird ein sogenanntes duales Gitter  $\widetilde{\mathbf{G}}$  eingeführt. Zur Unterscheidung des dualen Gitters von dem bereits eingeführten Gitter wird  $\mathbf{G}$  im Folgenden als primäres Gitter bezeichnet. Die Mittelpunkte der Gitterzellen von  $\mathbf{G}$  bilden die Gitterpunkte des dualen Gitters  $\widetilde{\mathbf{G}}$ . Jede primäre Gitterkante durchstößt damit genau eine duale Gitterfläche und umgekehrt (siehe Abbildung 2.3(b)).

Es erfolgt die Einführung von magnetischen Gitterspannungen  $\widehat{\mathbf{h}}$  gemäß

$$\widehat{h}_\nu(i, j, k) := \int_{\widetilde{L}_\nu(i, j, k)} \vec{H} \cdot \vec{e}_\nu \, d\nu, \quad (2.25)$$

sowie von elektrischen Gitterflüssen  $\widehat{\mathbf{d}}$  und Gitterströmen  $\widehat{\mathbf{j}}$  gemäß

$$\widehat{d}_\nu(i, j, k) := \int_{\widetilde{A}_\nu(i, j, k)} \vec{D} \cdot \vec{e}_\nu \, dA \quad \text{und} \quad \widehat{j}_\nu(i, j, k) := \int_{\widetilde{A}_\nu(i, j, k)} \vec{J} \cdot \vec{e}_\nu \, dA, \quad (2.26)$$

wobei die Größen  $\widetilde{L}_\nu$  und  $\widetilde{A}_\nu$  die Kanten und Flächen des dualen Gitters bezeichnen. Die Indizierung der dualen Kanten und Flächen erfolgt derart, dass eine duale Kante den gleichen Index besitzt wie die primäre Gitterfläche, die sie durchstößt. Entsprechendes gilt auch für die dualen Flächen.

Weiterhin wird die in einer dualen Zelle befindliche elektrische Ladung

$$q(i, j, k) := \int_{\widetilde{V}_\nu(i, j, k)} \varrho \, dV \quad (2.27)$$

eingeführt. Nach dem gleichen Prinzip wie schon für die beiden Gleichungen (2.1) und (2.4) können nun die beiden Gleichungen (2.2) und (2.3) für die Zellen des dualen Gitters angegeben werden als

$$\widetilde{\mathbf{C}}\widehat{\mathbf{h}} = \widehat{\mathbf{j}} + \frac{d}{dt}\widehat{\mathbf{d}}, \quad (2.28)$$

$$\widetilde{\mathbf{S}}\widehat{\mathbf{d}} = \mathbf{q}, \quad (2.29)$$

wobei es sich bei  $\tilde{\mathbf{C}}$  und  $\tilde{\mathbf{S}}$  wiederum um topologische Matrizen handelt. Die Gleichungen (2.22), (2.24), (2.28) und (2.29) werden als Gitter-MAXWELLgleichungen bezeichnet. Für die topologischen Matrizen gilt

$$\mathbf{S}\mathbf{C} = 0 \quad \text{sowie} \quad \tilde{\mathbf{S}}\tilde{\mathbf{C}} = 0, \quad (2.30)$$

was als diskretes Analogon zur vektoranalytischen Beziehung

$$\operatorname{div} \operatorname{rot} \equiv 0 \quad (2.31)$$

interpretiert werden kann [26]. Mit Hilfe der Beziehung (2.30) lässt sich das diskrete Analogon zur Kontinuitätsgleichung (2.5) ableiten. Als Zusammenhang zwischen den Gitterströmen  $\widehat{\mathbf{j}}$  und der elektrischen Ladung  $\mathbf{q}$  innerhalb der dualen Gitterzellen ergibt sich

$$\tilde{\mathbf{S}}\widehat{\mathbf{j}} + \frac{d}{dt}\mathbf{q} = 0. \quad (2.32)$$

Die Gitter-MAXWELLgleichungen beinhalten noch keine Näherungen, da sie lediglich eine Spezialisierung der MAXWELLSchen Gleichungen, welche für beliebige Volumina und Flächen gelten, für die vom Gitter festgelegten Flächen und Volumina beinhalten. Nichtsdestotrotz geht bei diesem Schritt bereits Information verloren, da lediglich Integrale über die gesuchten Feldgrößen betrachtet werden, aus denen die tatsächliche Feldverteilung nur noch approximativ berechnet werden kann. Ein solcher Informationsverlust ist jedoch bei einer Repräsentation der Feldverteilung mittels endlich vieler Freiheitsgrade unvermeidbar.

### 2.2.2 Materialbeziehungen

Der Zusammenhang der beiden Gleichungspaare (2.22), (2.24) und (2.28), (2.29) wird, wie bereits im analytischen Fall, durch Materialbeziehungen hergestellt. Im einfachsten Fall wird innerhalb jeder primären Gitterzelle eine homogene Materialverteilung angenommen [26]. Damit können beliebige Materialverteilungen nur noch approximiert werden.

Eine solche Materialapproximation führt zu sprunghaften Änderungen der Materialeigenschaften an den Grenzflächen zwischen primären Gitterzellen. Jedoch ist aus Abschnitt 2.1 bekannt, dass das elektrische Feld tangential sowie die magnetische Flussdichte normal zu Materialgrenzflächen einen stetigen Verlauf besitzt. Dies ermöglicht die eindeutige Definition der Größen  $\widehat{\mathbf{e}}$  und  $\widehat{\mathbf{b}}$  in Gleichung (2.19) bzw. (2.20), obwohl die Integration genau entlang der Materialgrenzflächen erfolgt. Der Zusammenhang zwischen den Vektoren der Gitterspannungen und der Gitterflüsse wird durch sogenannte Materialmatrizen ausgedrückt. In Analogie zu den analytischen Zusammenhängen aus Gleichung (2.7) wird definiert

$$\widehat{\mathbf{d}} = \mathbf{M}_\varepsilon \widehat{\mathbf{e}} \quad \text{und} \quad \widehat{\mathbf{h}} = \mathbf{M}_{\mu^{-1}} \widehat{\mathbf{b}}. \quad (2.33)$$

Eine näherungsfreie Berechnung dieser Zusammenhänge erfordert die Kenntnis des Feldverlaufes auf den Gitterflächen und entlang der Gitterlinien. Da dieser nicht bekannt ist, muss er approximiert werden. Die einfachste Möglichkeit, welche auch im Rahmen der vorliegenden Arbeit zum Einsatz kommt, ist die Annahme eines konstanten Feldverlaufes entlang der Kanten bzw. Flächen. Die Gleichungen (2.33) lassen sich dann komponentenweise angeben zu

$$\widehat{\widehat{\mathbf{d}}}_\nu(i, j, k) = \frac{\bar{\varepsilon}(i, j, k) \widetilde{A}_\nu(i, j, k)}{L_\nu(i, j, k)} \widehat{\mathbf{e}}_\nu(i, j, k) + O(L_\nu^\zeta) \quad (2.34)$$

$$\text{bzw. } \widehat{\widehat{\mathbf{b}}}_\nu(i, j, k) = \frac{\overline{\mu^{-1}}(i, j, k) A_\nu(i, j, k)}{\widetilde{L}_\nu(i, j, k)} \widehat{\mathbf{h}}_\nu(i, j, k) + O(L_\nu^\zeta), \quad (2.35)$$

wobei es sich bei  $\bar{\varepsilon}$  und  $\overline{\mu^{-1}}$  um flächen- bzw. kantengemittelte Größen handelt [26]. Der in den Gleichungen (2.34) und (2.35) eingeführte Fehler wird als lokaler Diskretisierungsfehler der Methode bezeichnet. Der Fehlerterm ist im besten Falle eines äquidistanten Gitters und einer homogenen Materialverteilung von zweiter Ordnung ( $\zeta = 2$ ).

### 2.2.3 Zeitintegration

Bei den Gitter-MAXWELLgleichungen handelt es sich um semidiskrete Gleichungen, da die Gitterspannungen und Gitterflüsse noch immer von der Zeitvariable  $t$  abhängen. Die numerische Berechnung des Zeitverlaufes der Gitterflüsse und -spannungen erfordert die Diskretisierung der Gitter-MAXWELLgleichungen auch in der Zeitvariable  $t$ .

Da die numerische Lösung eines elektromagnetischen Feldproblems häufig die Zeitintegration der Feldfreiheitsgrade über viele Zeitschritte erfordert, ist zu diesem Zweck ein effizienter Algorithmus notwendig. Ein effizientes, explizites Zeitintegrationsschema ist durch den sogenannten Leap-Frog Algorithmus gegeben [26]. Die Approximation des Zeitableitungsoperators geschieht mit Hilfe eines zentralen Differenzenquotienten. Der dabei eingeführte Fehler ist von zweiter Ordnung im Zeitschritt  $\Delta t$ . Um die Verwendung des zentralen Differenzenquotienten zu gewährleisten, sind die elektrischen und magnetischen Gitterspannungen auf der Zeitachse um einen halben Zeitschritt gegeneinander verschoben. Während die magnetischen Feldgrößen zu den Zeitpunkten  $t_s, t_s + \Delta t, t_s + 2\Delta t, \dots, M\Delta t$  berechnet werden, werden die elektrischen Feldgrößen zu den Zeitpunkten  $t_s + \frac{1}{2}\Delta t, t_s + \frac{3}{2}\Delta t, \dots, (M + \frac{1}{2})\Delta t$  berechnet. Die um den Zeitschritt  $\Delta t$  numerisch integrierten Feldwerte lassen sich gemäß

$$\widehat{\mathbf{h}}^{(m+1)} = \widehat{\mathbf{h}}^{(m)} - \Delta t \mathbf{M}_{\mu^{-1}} \widetilde{\mathbf{C}} \widehat{\mathbf{e}}^{(m+\frac{1}{2})} + O(\Delta t^2), \quad (2.36)$$

$$\widehat{\mathbf{e}}^{(m+\frac{3}{2})} = \widehat{\mathbf{e}}^{(m+\frac{1}{2})} + \Delta t \mathbf{M}_\varepsilon^{-1} \left( \widetilde{\mathbf{C}} \widehat{\mathbf{h}}^{(m+1)} - \widehat{\mathbf{j}}^{(m+1)} \right) + O(\Delta t^2) \quad (2.37)$$

bestimmen [26], wobei der hochgestellte Index den Zeitpunkt bezeichnet, für den die berechneten Feldwerte gelten. Das Verfahren ist bedingt stabil für Werte von  $\Delta t \leq \Delta t_{\max}$ . Die obere Grenze für den Zeitschritt  $\Delta t_{\max}$  hängt von der Größe der Gitterzellen sowie der Materialverteilung ab [26]. Hinreichend für die Stabilität der Zeitintegration ist die Wahl des Zeitschrittes  $\Delta t$  gemäß

$$\Delta t \leq \min_{i,j,k} \left\{ \sqrt{\frac{\varepsilon(i,j,k)\mu(i,j,k)}{\left(\frac{1}{L_x(i,j,k)}\right)^2 + \left(\frac{1}{L_y(i,j,k)}\right)^2 + \left(\frac{1}{L_z(i,j,k)}\right)^2}} \right\}. \quad (2.38)$$

Gleichung (2.38) wird als COURANT-FRIEDRICHS-LEVY Bedingung bezeichnet.

In Analogie zum analytischen Fall, beschreiben die Gleichungen (2.22) und (2.28) die zeitliche Dynamik der Gitterspannungen, während die Gleichungen (2.24) und (2.29) automatisch zu jedem Zeitpunkt erfüllt sind, wenn sie zum Anfangszeitpunkt gelten und, im Fall von Gleichung (2.29), die Quellterme  $\mathbf{q}$  und  $\hat{\mathbf{j}}$  der diskreten Kontinuitätsgleichung (2.32) genügen.

Wie im analytischen Fall sind neben den Anfangsbedingungen auch Randbedingungen für die Gitterspannungen zu stellen. Im Rahmen dieser Arbeit wurde lediglich die elektrische Randbedingung verwendet, welche die elektrischen Gitterspannungen tangential zum Rand des Rechengebietes zu Null setzt, was einer Berandung mit elektrisch ideal leitendem Material entspricht.

## 2.3 Der Particle-In-Cell (PIC) Algorithmus

Die Simulation eines Vielteilchensystems, wie in Abschnitt 2.1.1 mittels der VLASOV-Gleichung modelliert, erfordert die Darstellung der Verteilungsfunktion  $f_1$  mittels endlich vieler Freiheitsgrade. Im allgemeinen Fall hängt  $f_1$  von drei Raum- und drei Impulskordinaten sowie der Zeitkoordinate ab. Eine Diskretisierung dieses sechsdimensionalen Phasenraumes verbietet sich für die meisten Problemstellungen, da sie zu einem derart großen Datenvolumen und zu einer so großen Anzahl an Berechnungsoperationen führen würde, dass die Simulationen entweder eine unvertretbar lange Laufzeit benötigen würden, oder aber wegen zu geringer Speicherressourcen der verfügbaren Rechner erst gar nicht durchführbar wären.

Mit Hilfe des PIC Algorithmus können interessierende Charakteristiken der VLASOV-Gleichung approximativ berechnet werden, was mit deutlich geringerem Rechenaufwand möglich ist, als eine Zeitintegration von  $f_1$  im gesamten Phasenraum durchzuführen [27]. Wie in Abschnitt 2.1.1 erwähnt, lassen sich die Charakteristiken der VLASOV-Gleichung als mögliche Trajektorien geladener Teilchen interpretieren, die sich unter dem Einfluss

eines elektromagnetischen Feldes bewegen, und besitzen daher eine direkte physikalische Interpretation [22]. Die Berechnung der Charakteristiken ist äquivalent zur numerischen Integration der Bewegungsgleichungen (2.12) und (2.13). Diese kann mit Hilfe eines von BORIS vorgeschlagenen Algorithmus erfolgen [28]. Die Diskretisierung der Zeitableitungen geschieht dabei mit zentralen Differenzenquotienten. Die Zeitintegration von (2.12) erfolgt in drei Schritten gemäß dem folgenden Schema

$$\vec{p}^- = \vec{p}^{(m-\frac{1}{2})} + \frac{\Delta t}{2} q \vec{E}_p^{(m)} \quad (1. \text{ Beschleunigung}) \quad (2.39)$$

$$\frac{\vec{p}^+ - \vec{p}^-}{\Delta t} = q \cdot \frac{\vec{p}^+ + \vec{p}^-}{2m_0\gamma^{(m)}} \times \vec{B}_p^{(m)} \quad (\text{Drehung}) \quad (2.40)$$

$$\vec{p}^{(m+\frac{1}{2})} = \vec{p}^+ + \frac{\Delta t}{2} q \vec{E}_p^{(m)}. \quad (2. \text{ Beschleunigung}) \quad (2.41)$$

wobei  $\gamma^{(m)}$  definiert wird gemäß

$$\gamma^{(m)} = \sqrt{1 + \left(\frac{|\vec{p}^+|}{m_0 c}\right)^2} = \sqrt{1 + \left(\frac{|\vec{p}^-|}{m_0 c}\right)^2}. \quad (2.42)$$

Die hochgestellten Indizes geben wieder die Zeitpunkte an, zu denen die Größen auf der Zeitachse allokiert sind. Die Größen  $\vec{p}^+$  und  $\vec{p}^-$  sind Hilfsgrößen. Die Zeitintegration von (2.13) wird gemäß

$$\vec{r}^{(m+1)} = \vec{r}^{(m)} + \Delta t \cdot \frac{\vec{p}^{(m+\frac{1}{2})}}{m_0\gamma^{(m+\frac{1}{2})}} \quad (2.43)$$

durchgeführt.

Es fällt auf, dass die Feldwerte des elektromagnetischen Feldes zum Zeitpunkt  $m\Delta t$  am Integrationspunkt  $\vec{r}_p$  benötigt werden. Da bei einer näherungsweise Feldberechnung mittels FIT (siehe Abschnitt 2.2) Gitterspannungen und Gitterflüsse lediglich an bestimmten, durch das Gitter festgelegten Stellen bekannt sind, müssen die Feldwerte an den Integrationspunkten durch eine Interpolation aus diesen Größen berechnet werden. Im Rahmen dieser Arbeit wurde dafür eine trilineare Interpolation verwendet [29]. Im Falle dreier Raumdimensionen werden die Feldwerte somit aus den acht zum Integrationspunkt am nächsten gelegenen Feldwerte berechnet. Um das elektrische Feld zu den Zeitpunkten  $m\Delta t$  zu berechnen, wird die Zeitintegration gemäß (2.37) in zwei Schritten mit dem jeweils halben Zeitschritt vorgenommen [25].

Eine Änderung der Verteilungsfunktion  $f_1$  wirkt sich auf das elektromagnetische Feld aus. Um diesen Einfluss approximativ berechnen zu können, müssen die Ströme  $\widehat{\vec{j}}$  bestimmt werden, welche sich durch die Änderung von  $f_1$ , d.h. durch die Bewegung der Ladungsträger, ergeben. Die Approximation dieser Ströme geschieht mit Hilfe der berechneten

Charakteristiken. Beim in dieser Arbeit verwendeten Cloud-In-Cell Schema [30] werden die berechneten Charakteristiken als Trajektorien von sogenannten Makroteilchen interpretiert. Dabei handelt es sich um Ladungsverteilungen endlicher Ausdehnung, welche ein ganzes Ensemble realer Teilchen modellieren. Aus der Bewegung der Makroteilchen in dem betrachteten Zeitschritt können die Ströme  $\hat{j}$  berechnet werden. Dies kann so erfolgen, dass die diskrete Kontinuitätsgleichung erfüllt wird, und somit auch das diskrete GAUSSsche Gesetz für alle Zeitschritte erfüllt bleibt. In der Arbeit wurde dazu das von BUNEMAN vorgeschlagene Verfahren implementiert [31]. Die Beschaffenheit der Makroteilchen wurde so gewählt, dass während eines Zeitschrittes nur die Ströme beeinflusst werden können, die der Gitterzelle zugeordnet sind, innerhalb derer sich das Teilchen zu Beginn des Zeitschrittes befand, sowie die Ströme der Gitterzellen, die von dieser Zelle höchstens über eine weitere Zelle verbunden sind.

Abbildung 2.4 zeigt das Ablaufdiagramm für den PIC Algorithmus, wie er im Rahmen dieses Abschnittes beschrieben wurde.

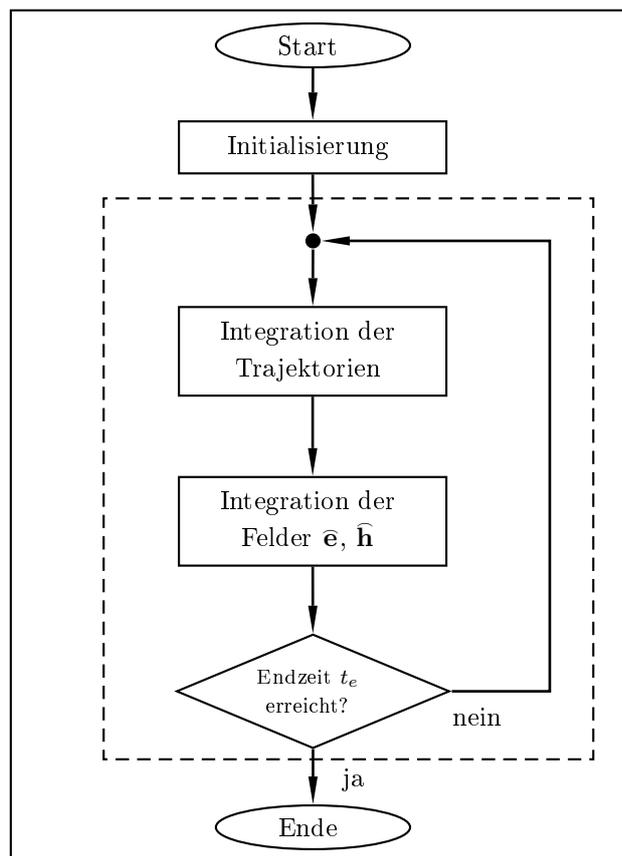


Abbildung 2.4: Das Ablaufdiagramm des PIC Algorithmus.

## 2.4 Paralleles Rechnen

Unter dem Begriff „paralleles Rechnen“ versteht man die Zerlegung eines mathematischen Problems in Teilprobleme und deren simultane Lösung mit Hilfe einer gewissen Anzahl von Verarbeitungseinheiten (Prozessoren). Wird ein Algorithmus dahingehend modifiziert, dass die Abarbeitung seiner Teilschritte simultan durch verschiedene Prozessoren erfolgen kann, so spricht man von der Parallelisierung des Algorithmus.

Viele praxisrelevante mathematische Problemstellungen aus den unterschiedlichsten Bereichen von Wissenschaft, Technik und Wirtschaft erfordern eine derart große Zahl an Rechenoperationen zu ihrer Lösung, dass die Rechenzeit, welche ein einziger Prozessor zur Lösung des Problems benötigen würde, unvertretbar groß wäre. Davon sind insbesondere zeitkritische Berechnungen betroffen, deren Ergebnisse innerhalb eines bestimmten Zeitraumes vorliegen müssen, um von Nutzen zu sein. Jedoch sind auch für Probleme, deren Lösung nicht grundsätzlich zeitkritisch ist, sehr lange Rechenzeiten unerwünscht.

Die Simulation von Vielteilchensystemen mit Hilfe des in den vorangegangenen Abschnitten beschriebenen PIC Algorithmus ist ein Beispiel für ein solches Problem. Die Simulation der Teilchendynamik für die in Kapitel 4 betrachtete Beschleunigerstruktur führte beispielsweise für die dort simulierte Strecke von zwei Metern Länge zu einem Modell mit mehr als einer halben Milliarde Gitterzellen und mehr als einer viertel Million Teilchen. Für die benötigten ca. 100.000 Zeitschritte, um die gewünschte Zeitspanne zu simulieren, benötigte der verwendete Parallelrechner, welcher mit 90 Prozessoren ausgestattet war, eine Laufzeit von etwa sechs Tagen, während ein Einzelprozessor entsprechender Leistungsfähigkeit für dieselbe Berechnung eine Laufzeit von deutlich mehr als einem Jahr benötigt hätte.

Der Steigerung der Verarbeitungsgeschwindigkeit einzelner Prozessoren, wie sie in den vergangenen Jahrzehnten erfolgte, sind prinzipielle physikalische Grenzen gesetzt. Die Parallelverarbeitung mittels mehrerer Prozessoren ist gegenwärtig die einzige Möglichkeit eine gegenüber dem Einzelprozessor deutlich höhere Rechenleistung zu erzielen und somit die Rechenzeit zur Lösung eines Problems zu reduzieren [32].

Das Prinzip der Parallelverarbeitung findet Anwendung in vielen unterschiedlichen Bereichen von Wissenschaft, Technik und Wirtschaft, in denen aufwändige Rechnungen durchgeführt werden müssen. Die Architektur der dabei verwendeten Rechanlagen und die eingesetzten Programmiermodelle sind ebenso vielfältig.

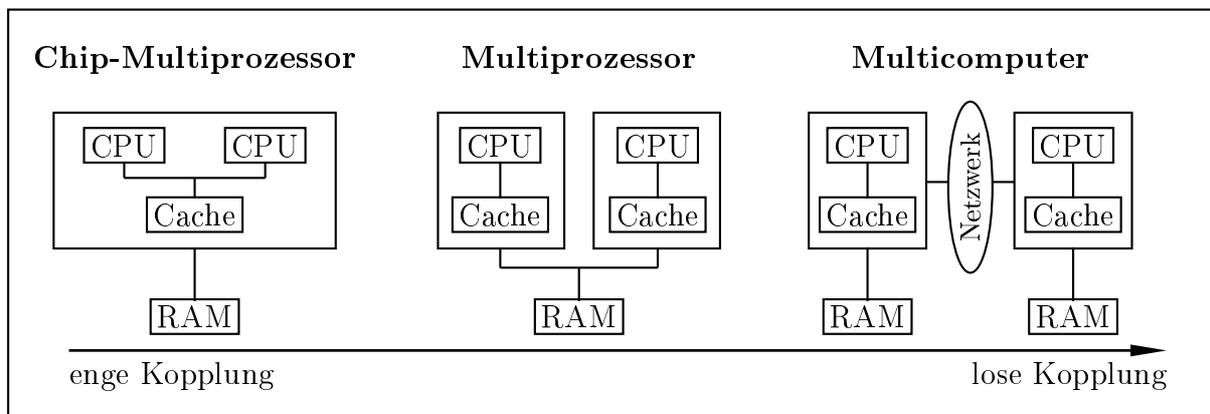
### 2.4.1 Architektur von Parallelrechnern

Zur Klassifikation von Rechanlagen bezüglich ihrer Fähigkeit zur Parallelverarbeitung findet das Schema von FLYNN [33], obwohl recht grob, breite Anwendung. Die Klas-

sifikation der Maschinen erfolgt in diesem Schema über die Anzahl der Befehlsströme und Datenströme, welche gleichzeitig verarbeitet werden können. Es wird unterschieden zwischen **Single Instruction Stream, Single Data Stream (SISD)** Maschinen, welche nicht zur Parallelverarbeitung in der Lage sind, sowie **Single Instruction Stream, Multiple Data Stream (SIMD)** Maschinen und **Multiple Instruction Stream, Multiple Data Stream (MIMD)** Maschinen, welche eine Parallelverarbeitung leisten können. Zur Klasse der SIMD Maschinen gehören sogenannte Vektorrechner, deren Prozessor für die Verarbeitung von Operationen der linearen Algebra optimiert sind. Diese Rechner haben mittlerweile gegenüber den viel flexibleren MIMD Rechnern ihre Bedeutung fast völlig verloren. Zur Klasse der MIMD Maschinen gehören alle Rechner, die in der Lage sind, mehrere verschiedene Instruktionen gleichzeitig auf voneinander verschiedenen Datenelementen auszuführen.

Eine weitere Klassifikation der MIMD Maschinen erfolgt basierend auf der Kopplung der Prozessoren untereinander, sowie der Anbindung des Hauptspeichers (**R**andom **A**ccess **M**emory, RAM) und des Cache Speichers, wie in Abbildung 2.5 verdeutlicht.

Die engste Kopplung besitzen Systeme, bei denen sich die Prozessoren auf dem gleichen Chip befinden. Die Prozessoren benutzen häufig einen gemeinsamen Level 2-Cache Speicher, sowie einen gemeinsamen Hauptspeicher. Man spricht hier auch von Mehrkern (Mul-



**Abbildung 2.5:** Klassifikation von MIMD Rechnern nach dem Kopplungsgrad von Prozessoren (**C**entral **P**rocessing **U**nit, CPU) und Hauptspeicher (nach [32]).

ticore) Prozessoren (engl. **C**entral **P**rocessing **U**nit, CPU).

Bei Multiprozessorrechnern oder einfach Multiprozessoren ist die Kopplung etwas lockerer. Die einzelnen CPUs befinden sich nicht auf dem gleichen Chip, häufig jedoch auf der gleichen Platine, und besitzen unabhängige Cache Speicher. Auf den Hauptspeicher wird mittels eines gemeinsamen Adressraumes zugegriffen. Die Bereitstellung des gemeinsamen Adressraumes und alle Operationen zur Verwaltung des gemeinsamen Speichers sind, wie

beim Chip-Multiprozessor, hardwareseitig implementiert und somit für den Programmentwickler vollkommen unsichtbar.

Die Vorteile dieser Architekturen sind die geringe Latenzzeit beim Zugriff auf den Speicher einerseits und die für den Programmentwickler vollständig unsichtbare Speicherorganisation andererseits, welche die Softwareentwicklung erheblich vereinfacht. Jedoch ist die Anzahl der Prozessoren, aus denen ein Multiprozessor aufgebaut werden kann, begrenzt. Die effiziente Abwicklung und Organisation der Speicherzugriffe auf den gemeinsam genutzten Speicher wird mit wachsender Anzahl von Prozessoren sehr kompliziert und teuer, so dass Multiprozessorsysteme üblicherweise mit deutlich weniger als 100 Prozessoren ausgestattet sind [32].

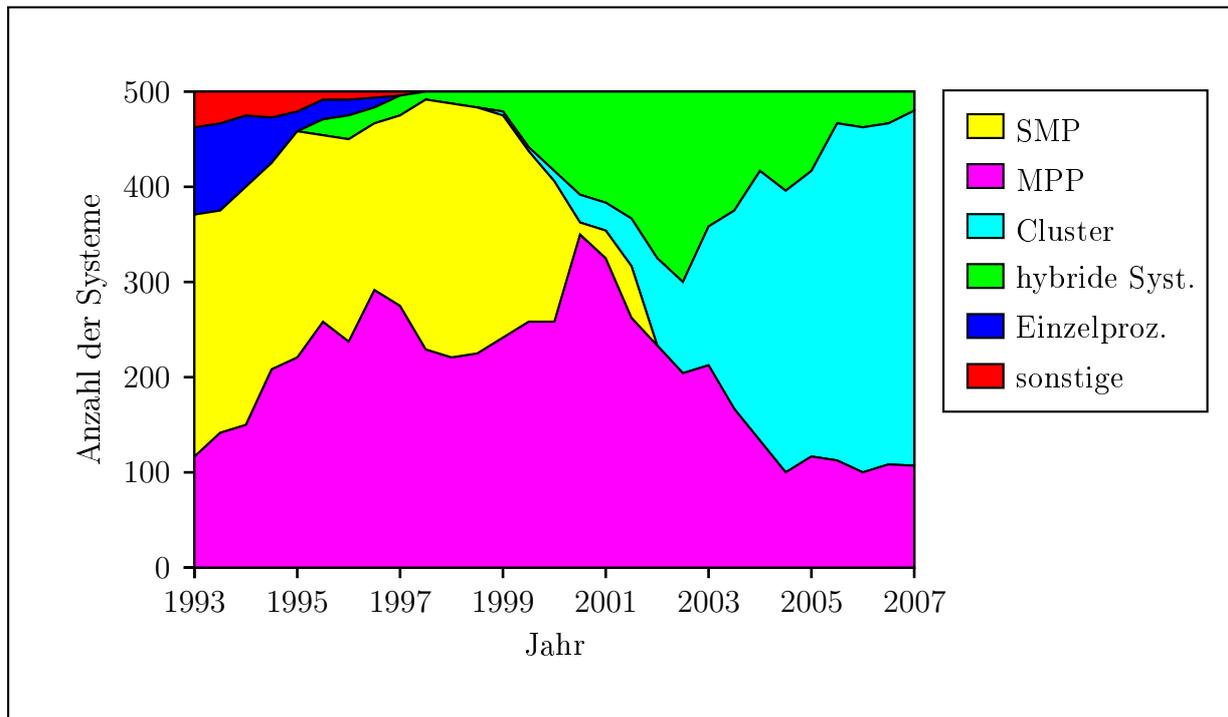
Bei Parallelrechnern, die zur Klasse der Multicomputer gehören, ist jedem der Prozessoren ein eigener Hauptspeicher zugeordnet. Es existiert somit kein hardwareseitig organisierter, gemeinsamer Adressraum. Der Zugriff auf Daten, die in einem nicht lokalen Speichermodul gehalten werden, muss softwareseitig organisiert werden. Die Klasse der Multicomputer umfasst sowohl die sogenannten **Massiv Parallelen Prozessoren** (MPP), bei denen die CPUs über ein spezialisiertes Hochgeschwindigkeitsnetzwerk gekoppelt sind, wie auch die Rechnercluster oder einfach Cluster. Die Spanne der Rechnercluster reicht dabei von spezialisierten, von der Hard- und Softwarekonfiguration meist homogenen Rechnerclustern, welche durch hochoptimierte Einzelrechner und schnelle Verbindungsnetzwerke eine hohe Leistungsfähigkeit besitzen, bis hin zu oftmals heterogenen Systemen, welche aus unterschiedlichen Standardkomponenten des Massenmarktes zusammengestellt sind, bezeichnet als Beowulf Cluster. Ein Multicomputer, welcher sich aus hardware- wie softwaremäßig oftmals heterogenen und geografisch weit voneinander entfernten Rechnern zusammensetzt, wird als Grid bezeichnet. Oft handelt es sich bei den vernetzten Rechnern selber bereits um Parallelrechner, die sich in unterschiedlichen Rechenzentren befinden [34]. Bei einem Grid handelt es sich um die loseste Kopplung von Prozessoren gemäß dem Diagramm in Abbildung 2.5.

Der Preisverfall bei Rechner- und Netzwerkkomponenten bei gleichzeitiger Leistungssteigerung hat Rechnercluster als Architektur für das parallele Rechnen in den vergangenen Jahren stark an Bedeutung gewinnen lassen, da mit einem Cluster ein nahezu unschlagbares Verhältnis von Hardwarekosten zu Rechenleistung erreicht werden kann. Ein weiterer Vorteil dieser Systeme ist ihre fast unbegrenzte Erweiterbarkeit. Mittlerweile existieren Cluster, welche aus deutlich mehr als 100.000 Prozessoren bestehen [35].

Der Erfolg der Cluster als Infrastruktur für das parallele Rechnen lässt sich an der TOP 500 Liste der leistungsfähigsten Parallelrechner ablesen [36]. Mehr als 70% der 500 leistungsfähigsten Rechner der Welt sind zum Zeitpunkt des Entstehens dieser Arbeit Rechnercluster, während ein Anteil von etwa 10-20% als MPP Systeme bezeichnet werden können,

wobei die Bedeutung der Rechnercluster immer noch kontinuierlich zunimmt (siehe Abbildung 2.6).

Der größte Nachteil von Multicomputersystemen ergibt sich aus der verteilten Natur des Systems. Abhängig vom Speicherzugriffsmuster der verwendeten Algorithmen ist es bei parallelen Rechnungen in der Regel unvermeidlich, dass ein Prozessor auf Daten zugreifen muss, die nicht in seinem lokalen Hauptspeicher gehalten werden. Diese Daten müssen



**Abbildung 2.6:** Anteil der verschiedenen Parallelrechnerarchitekturen an den 500 leistungsfähigsten Parallelrechnern der Welt [36].

über das Verbindungsnetzwerk aus einem entfernten Speicherblock geholt werden. Die dafür benötigte Zeit ist sehr viel größer als für einen Zugriff auf den lokalen Speicher. Die tatsächlich benötigte Zeit für einen entfernten Speicherzugriff hängt dabei von der verwendeten Netzwerktechnologie sowie der Netzwerktopologie ab. Dies macht sich umso stärker bemerkbar, je langsamer das Verbindungsnetzwerk ist und je mehr Datenaustausch ein paralleler Algorithmus erfordert.

Ein weiterer Nachteil ergibt sich daraus, dass Multicomputer keine Hardwareunterstützung für den Zugriff auf den verteilten Hauptspeicher anbieten, d.h. es existiert kein gemeinsamer Adressraum. Somit muss die Verteilung der Daten auf die Speichermodule sowie die Zugriffe auf nichtlokale Speichermodule softwareseitig organisiert werden, was dazu

führt, dass sich die Programmentwicklung für einen Multicomputer im Allgemeinen sehr viel komplizierter gestaltet als für einen Multiprozessor. Es existieren zwar einige Ansätze, diese Datenverteilung automatisiert durch einen Compiler vornehmen zu lassen und somit dem Programmentwickler einen einfachen Zugriff auf den Speicher zu bieten, jedoch konnten die mit derartigen Ansätzen erzielten Performanceergebnisse bislang nicht überzeugen, so dass bei dem Großteil der parallelen Programme für Multicomputer der Programmentwickler die Zuordnung der Daten und Berechnungsoperationen auf die Prozessoren manuell vornehmen muss, um eine zufriedenstellende Performance zu erreichen [37].

Die Grenzen zwischen den einzelnen Architekturen sind fließend. So gewinnen beispielsweise hybride Systeme an Bedeutung, bei denen es sich um Multicomputer handelt, die aus Multiprozessoren statt aus Einzelprozessoren zusammengesetzt sind, welche wiederum aus mehreren Kernen bestehen können.

Die im Rahmen der vorliegenden Arbeit vorgestellten Algorithmen sind, wenngleich prinzipiell unabhängig von der Architektur des Parallelrechners, auf dem sie schließlich implementiert und ausgeführt werden, speziell für Multicomputer entwickelt worden, und berücksichtigen deshalb schwerpunktmäßig die Probleme, auf welche man bei Verwendung von Parallelrechnern dieser Architektur stößt.

### 2.4.2 Programmiermodelle für Parallelrechner

Ein Programmiermodell beschreibt ein Rechnersystem aus der Sicht einer Programmiersprache oder einer Programmierumgebung, definiert also die Sicht des Programmentwicklers auf einen Rechner [38]. Ein Programmiermodell spezifiziert somit die Basisfunktionalitäten, auf die bei der Implementierung zurückgegriffen werden kann. Aus diesem Grund muss bereits bei der Entwicklung eines Algorithmus das Programmiermodell berücksichtigt werden, welches später bei der Implementierung genutzt werden soll.

Wie bereits in Abschnitt 2.4.1 erwähnt, gehören die meisten zur Parallelverarbeitung fähigen Rechner zur MIMD Klasse, d.h. es ist grundsätzlich möglich für jeden Prozessor einen eigenen Befehlsstrom, der auf eigenen Daten arbeitet, zu spezifizieren. Bei der Implementierung eines Algorithmus auf einem MIMD Parallelrechner muss somit festgelegt werden, welcher Prozessor welche Verarbeitungsschritte ausführen soll. Weiterhin können Abhängigkeiten zwischen den Verarbeitungsschritten existieren, welche eine Abarbeitung in einer bestimmten Reihenfolge erzwingen. Somit müssen Synchronisationsmechanismen vorhanden sein. Schließlich muss festgelegt werden, wie ein Prozessor auf Daten im Hauptspeicher zugreifen kann.

### 2.4.2.1 Das SPMD Programmiermodell

Um die Komplexität, welche durch die unterschiedlichen Befehlsströme eingeführt wird, zu reduzieren und der Tatsache Rechnung zu tragen, dass ein paralleler Algorithmus oftmals gleiche Operationen auf jeweils unterschiedlichen Daten für die verfügbaren Prozessoren vorsieht, wobei lediglich an gewissen Stellen eine Unterscheidung notwendig ist, basieren heute praktisch alle Programmiermodelle auf dem von DAREMA et. al. [39] vorgeschlagenen **Single-Program-Multiple-Data** (SPMD) Modell. In diesem Modell führen alle an der Rechnung beteiligten Prozessoren zunächst den gleichen Befehlsstrom aus, wobei innerhalb des Befehlsstroms Kontrollstrukturen enthalten sind, die dazu führen, dass die einzelnen Prozessoren unterschiedliche Anweisungen ausführen bzw. die Rechenoperationen auf unterschiedlichen Daten ausgeführt werden, so weit dies benötigt wird.

### 2.4.2.2 Implizit und explizit parallele Programmiermodelle

Inwieweit die Organisation der Verteilung der Daten auf die Speichermodule, der Speicherzugriffe und der Synchronisation vom Programmentwickler vorgenommen werden muss, ist stark unterschiedlich. Es existieren sowohl implizit parallele Programmiermodelle, bei denen dies vollständig vom Compiler übernommen wird und für den Programmentwickler unsichtbar bleibt, wie auch explizit parallele Programmiermodelle, bei denen dies vollständig vom Programmentwickler geleistet werden muss. Daneben finden auch Mischformen, bei denen lediglich ein oder mehrere Teilaspekte der Parallelverarbeitung von dem Programmentwickler organisiert werden muss, Anwendung.

Während implizite Programmiermodelle eine Programmentwicklung erlauben, welche jener für einen seriellen Rechner sehr ähnlich ist, sind sie durch den sehr begrenzten Einfluss, den der Programmentwickler auf die Erstellung der parallelen Programmteile ausüben kann, recht unflexibel und für Algorithmen mit komplizierten Speicherzugriffsmustern verbunden mit einem Parallelrechner mit komplizierter Architektur kaum geeignet, da sie in solchen Fällen zu einem ineffizienten Programm führen. In der Welt der Multicomputer haben sich daher von Anfang an explizit parallele Programmiermodelle etabliert, während für Multiprozessoren eher implizite Programmiermodelle Anwendung finden. Für Multicomputer ist insbesondere die Verteilung der Daten auf den Hauptspeicher, bedingt durch die stark unterschiedliche Zugriffszeit auf lokale und nichtlokale Speicherbereiche, entscheidend für eine zufriedenstellende Performance. Diese wird daher in der Regel explizit vom Programmentwickler vorgenommen, basierend auf den Speicherzugriffsmustern, welche der Algorithmus erzeugt.

Da die im Rahmen dieser Arbeit vorgestellten Algorithmen schwerpunktmäßig für Multi-

computersysteme entwickelt wurden, wurde bei deren Formulierung ein explizit paralleles Programmiermodell verwendet. Die Verteilung der Daten auf den Hauptspeicher und die Organisation des benötigten Datenaustausches werden somit explizit angegeben und deren Optimierung für PIC Algorithmen bilden einen Schwerpunkt der Arbeit.

### 2.4.2.3 Gemeinsamer und verteilter Speicher

Eine weitere Klassifikation von Programmiermodellen für Parallelrechner bezieht sich auf die Art, wie aus Sicht des Programmierentwicklers auf den Hauptspeicher zugegriffen werden kann. Man unterscheidet zwischen Modellen, die einen gemeinsamen globalen Adressraum spezifizieren, und nachrichtenorientierten Programmiermodellen.

Im Falle eines Programmiermodells mit gemeinsamem Speicher erfolgt die physikalische Verteilung der Daten auf den Speicher des Parallelrechners vollkommen unsichtbar für den Programmierentwickler. Auf der Ebene der Programmentwicklung besteht die Illusion eines einzigen großen, homogenen Speichermoduls, auf das alle Instruktionsströme gleichermaßen zugreifen können. Da jedoch mehrere Instruktionsströme auf dem gleichen Speichermodul arbeiten, muss die Konsistenz der Daten im Speicher vom Programmierentwickler sichergestellt werden. Dazu besteht die Möglichkeit einer bedingten oder unbedingten Synchronisation, wodurch eine bestimmte Reihenfolge, in der Operationen auf dem gemeinsamen Speicher ausgeführt werden, erzwungen werden kann. Das Fehlen einer derartigen Reihenfolge führt in der Regel zu einem undefinierten Verhalten des Programms. Außer Synchronisationsoperationen gibt es keine Interaktion zwischen den Instruktionsströmen, die vom Programmierentwickler organisiert werden muss. Für das Modell mit gemeinsamem Adressraum hat sich das 1998 standardisierte und seither kontinuierlich erweiterte OpenMP [40] als Schnittstelle durchgesetzt.

Im Fall des nachrichtenorientierten Programmiermodells ist jedem der Instruktionsströme ein privater Speicherbereich zugeordnet. Die Verteilung der Daten auf die privaten Speicherbereiche muss vom Programmierentwickler vorgenommen werden. Einem Instruktionsstrom ist ein Zugriff auf die Daten außerhalb seines eigenen Speichers nicht möglich. Werden Daten aus einem anderen Speicherbereich benötigt, so müssen diese explizit durch eine sogenannte Nachricht verschickt werden. Ein nichtlokaler Speicherzugriff erfordert also die Interaktion zweier Instruktionsströme, welche vom Programmierentwickler organisiert werden muss. Neben den sogenannten blockierenden Sende- und Empfangsoperationen, die einen Instruktionsstrom so lange anhalten, bis die Übertragung der Nachricht vollständig abgeschlossen ist, existieren auch nicht blockierende Operationen, die dem aufrufenden Instruktionsstrom erlauben, mit der Ausführung anderer Befehle, welche der angeforderten

Daten nicht bedürfen, fortzufahren, während auf das Eintreffen der Nachricht gewartet wird. Der Abschluss der nicht blockierenden Kommunikationsoperation kann dann zu einem späteren Zeitpunkt erfolgen. Somit enthält jede Kommunikationsoperation implizit eine Synchronisation der beteiligten Prozesse. Nicht blockierende Kommunikationsoperationen erlauben lediglich, den Zeitpunkt für die Synchronisation zu verschieben und die Wartezeit bis zum Eintreffen der Nachricht mit nützlichen Berechnungen zu füllen. Somit definieren beim nachrichtenorientierten Programmiermodell die Kommunikationsoperationen verbunden mit ihrer inhärenten Synchronisation die Reihenfolge, mit der Operationen auf dem verteilten Speicher ausgeführt werden.

Das auch im Rahmen dieser Arbeit verwendete **Message Passing Interface** (MPI) spezifiziert eine Schnittstelle, welche die Operationen des nachrichtenorientierten Programmiermodells zur Verfügung stellt. MPI wurde 1994 standardisiert [41] und 1997 erweitert [42]. Freie sowie kommerzielle Implementierungen existieren mittlerweile für zahlreiche Programmiersprachen und die unterschiedlichsten Parallelrechner sowie alle gängigen Betriebssysteme. Heute ist MPI mit weitem Abstand die am häufigsten verwendete Schnittstelle für parallele Programme, welche auf dem nachrichtenorientierten Programmiermodell basieren. Außer Funktionen für den Punkt-zu-Punkt Datenaustausch zwischen genau zwei Prozessoren stellt der MPI Standard auch Funktionen für globale Kommunikationsoperationen zur Verfügung, an denen mehr als zwei Prozessoren beteiligt sein können. Der Datenaustausch erfolgt dann mittels optimierter Kommunikationsmuster.

Da, wie in Abschnitt 2.4.1 erwähnt, auch zunehmend hybride Systeme zum Einsatz kommen, welche für einen Teil der Prozessoren die Multiprozessorarchitektur umsetzen, diese Multiprozessoren aber als Bausteine zum Aufbau von Multicomputern nutzen, werden auch zunehmend Algorithmen entwickelt, welche beide Modelle beinhalten und deren Implementierung MPI ebenso wie OpenMP benutzt.

Obwohl Implementierungen beider Modelle sowohl für Multiprozessoren wie für Multicomputer existieren, hat sich für die Programmierung von Multicomputern weitgehend der nachrichtenorientierte Ansatz durchgesetzt, der dem Programmentwickler die nötige Flexibilität bietet, um die Datenverteilung und den Speicherzugriff optimal organisieren zu können, während sich für Multiprozessoren das Modell, welches den Datenzugriff über einen gemeinsamen Adressraum ermöglicht, etabliert hat. Trotz zahlreicher Bemühungen, für Multicomputer einen gemeinsamen Adressraum auf Softwareebene zu emulieren, welcher zu einer erheblich einfacheren Softwareentwicklung führen würde (siehe [43] für eine Übersicht), ist es selbst angesichts der Entwicklung von Netzwerken mit hoher Bandbreite und geringer Latenz bislang nicht gelungen, eine Performance zu erreichen, die mit derjenigen von optimierten Programmen, die auf dem nachrichtenorientierten Ansatz basieren, konkurrieren kann [37].

Bei der Entwicklung der in dieser Arbeit vorgestellten Algorithmen wurde ein explizit paralleles Programmiermodell zugrunde gelegt. Für die Organisation der Speicherzugriffe wurde ein nachrichtenorientiertes Programmiermodell verwendet. Diese Kombination bietet die Flexibilität, welche zur Formulierung und effizienten Implementierung der Algorithmen auf Multicomputern benötigt wird.

### 2.4.3 Analyse paralleler Algorithmen

Ein wichtiges Qualitätsmerkmal für einen Algorithmus ist sein Bedarf an Ressourcen. Von großem Interesse ist dabei insbesondere die benötigte Rechenzeit sowie der Speicherbedarf und deren Abhängigkeit von der Größe des zu bearbeitenden Problems. Die Problemgröße ist zunächst ein abstrakter Begriff und kann nur kontextabhängig definiert werden. Beispielsweise würde für einen Algorithmus, welcher zum Sortieren einer Liste von Ganzzahlen verwendet wird, unter der Problemgröße sinnvollerweise die Anzahl der Elemente in der Liste verstanden werden, die sortiert werden soll.

Bei einer theoretischen Analyse eines Algorithmus ist insbesondere sein asymptotisches Verhalten von Interesse, d.h. die Abhängigkeit des Ressourcenbedarfs von der Problemgröße für den Grenzfall sehr großer Problemgrößen. Das asymptotische Verhalten wird in der theoretischen Informatik mit Hilfe der Methoden der Komplexitätstheorie beschrieben [44]. Die Sichtweise der Komplexitätstheorie ist zur Bewertung paralleler Algorithmen jedoch nicht ausreichend, da an einen parallelen Algorithmus weitere Qualitätsforderungen gestellt werden müssen. Insbesondere ist das Verhalten des Algorithmus in Abhängigkeit von der Anzahl der verwendeten Prozessoren des Parallelrechners von Interesse. Zur Analyse und Bewertung dieses Verhaltens sind verschiedene Metriken vorgeschlagen worden. Im Folgenden wird eine Auswahl dieser Metriken vorgestellt, welche für die Analyse der im Rahmen dieser Arbeit behandelten Algorithmen von Interesse sind. Eine vollständigere Übersicht findet sich in [45] und [46].

Die theoretische Analyse eines Algorithmus erfordert grundsätzlich ein Modell des Rechners, auf welchem der Algorithmus implementiert werden soll. Die Verbindung eines Algorithmus mit einem Rechnermodell wird als System bezeichnet. Im Falle von parallelen Algorithmen spricht man von einem parallelen System [47].

Im Folgenden soll jedoch zunächst vollkommen von Maschinenmodellen abstrahiert werden, bis in Abschnitt 2.4.4 auf diese Problematik eingegangen wird.

#### 2.4.3.1 Laufzeit

Wie auch bei seriellen Systemen ist eine Analyse der Laufzeit, welche ein paralleles System zur Bearbeitung eines Problems benötigt, von besonderem Interesse, und insbesondere für

zeitkritische Berechnungen ist eine Abschätzung der Laufzeit unverzichtbar, um den für die Berechnungen zur Verfügung stehenden Zeitrahmen nicht zu überschreiten. Anders als bei seriellen Systemen hängt die Laufzeit für ein paralleles System mindestens von den zwei Parametern Problemgröße  $N$  und Anzahl der verwendeten Prozessoren  $N_\pi$  ab.

Obwohl die im Folgenden vorgestellten Metriken wichtige Einblicke in das Verhalten eines parallelen Systems geben können, kann ihre alleinige Anwendung zu falschen Schlussfolgerungen über die Qualität des untersuchten Systems führen [45]. Eine Analyse der schlussendlich erzielten Laufzeit ist daher meist unverzichtbar.

### 2.4.3.2 Speedup und Effizienz

Wie bereits erwähnt, ist das Ziel der Parallelverarbeitung, die zur Lösung eines Problems benötigte Laufzeit gegenüber einer seriellen Bearbeitung zu reduzieren. Der parallele Speedup misst die durch die Parallelverarbeitung erzielte Reduktion der Laufzeit bei der Lösung eines Problems im Vergleich zur Lösung mit Hilfe eines seriellen Algorithmus, welcher auf einem Referenzprozessor ausgeführt wird. Die Definition des Speedup in der Literatur ist nicht einheitlich. Es existieren die beiden folgenden Definitionen. Zum einen wird der Speedup definiert als (siehe z.B. [47])

$$S^*(N_\pi, N) := \frac{T^*(1, N)}{T(N_\pi, N)}. \quad (2.44)$$

Dabei bezeichnet  $T^*(1, N)$  die benötigte Laufzeit für den Fall, dass das Problem mit Hilfe des *bestmöglichen* seriellen Algorithmus auf einem Referenzprozessor gelöst wird, wobei als bestmöglicher Algorithmus in diesem Zusammenhang derjenige angesehen wird, der das Problem in der kürzest möglichen Zeit löst. Die Größe  $T(N_\pi, N)$  bezeichnet die benötigte Laufzeit, welche zur Lösung des Problems mit Hilfe eines parallelen Systems unter Benutzung von  $N_\pi$  gleichartigen Prozessoren, die überdies identisch zu dem Referenzprozessor sind, benötigt wird. Ein Speedup von  $S^*(N_\pi, N) = s$  bedeutet also, dass das parallele System mit Hilfe von  $N_\pi$  gleichartigen Prozessoren das Problem um den Faktor  $s$  schneller löst als der beste sequentielle Algorithmus auf dem Referenzprozessor. Der Parameter  $N$  beschreibt die Problemgröße und muss, wie erwähnt, abhängig von der Problemstellung definiert werden.

Die zweite Definition des Speedup, welche sich in der Literatur findet (siehe z.B. [48]), lautet

$$S(N_\pi, N) := \frac{T(1, N)}{T(N_\pi, N)}. \quad (2.45)$$

Dabei bedeutet  $T(1, N)$  die Laufzeit des parallelen Algorithmus, wenn dieser auf einem einzigen Referenzprozessor ausgeführt wird. Die beiden Definitionen führen dadurch, dass

definitionsgemäß  $T^*(1, N) \leq T(1, N)$  gelten muss, zu unterschiedlichen Bewertungen des gleichen parallelen Algorithmus.

Während die Definition in Gleichung (2.45) den Vorteil besitzt, dass hier nur das zu untersuchende parallele System betrachtet werden muss, muss bei Definition (2.44) jedes denkbare System zur Lösung des Problems berücksichtigt werden, um  $T^*(1, N)$  zu bestimmen. Allerdings birgt Definition (2.45) den Nachteil, dass der durch die Parallelverarbeitung erzielte Gewinn an Rechenzeit überschätzt wird.

Bei dem Studium der relevanten Literatur fällt auf, dass Gleichung (2.44) wesentlich häufiger als Definition des Speedup angeführt wird, allerdings in konkreten Performanceanalysen häufiger Definition (2.45) Anwendung findet. In Anlehnung an [45] werde die in Gleichung (2.44) definierte Größe als realer Speedup und die in Gleichung (2.45) definierte Größe als relativer Speedup bezeichnet.

Da die in dieser Arbeit betrachteten parallelen Algorithmen für den Fall  $N_\pi = 1$  in die für serielle Berechnungen verwendeten Algorithmen übergehen, sind die beiden Größen  $S$  und  $S^*$  in diesem Fall äquivalent. Der Speedup wird daher im Folgenden stets mit  $S$  bezeichnet.

Für den Speedup in Definition (2.44) und (2.45) ergibt sich eine theoretische obere Schranke  $S_{\text{opt}}$  gemäß

$$S_{\text{opt}}(N_\pi, N) := N_\pi, \quad (2.46)$$

da im Allgemeinen die Ungleichung

$$T^*(1, N) \leq T(1, N) \leq N_\pi \cdot T(N_\pi, N) \quad (2.47)$$

erfüllt ist. Es sei jedoch angemerkt, dass unter bestimmten Bedingungen die Ungleichungskette (2.47) verletzt werden und ein Speedup  $S > S_{\text{opt}}$  erreicht werden kann. Man spricht in diesem Fall von einem superlinearen Speedup. Ein superlinearer Speedup kann beispielsweise aufgrund von Effekten auftreten, die durch die Architektur des verwendeten Parallelrechners, insbesondere durch die Speicherhierarchie, bedingt sind. Für eine Analyse von verschiedenen Ursachen für das Auftreten eines superlinearen Speedup sei auf [49] verwiesen.

Die mit dem Speedup eng verwandte Effizienz bildet ein weiteres häufig angewendetes Maß zur Bewertung eines parallelen Systems. Ebenso wie beim Speedup existieren zwei unterschiedliche Definitionen, die sich wiederum darin unterscheiden, ob  $T(1, N_\pi)$  oder  $T^*(1, N_\pi)$  als Vergleichsgröße herangezogen wird. Die Definitionen lauten

$$E^*(N_\pi, N) := \frac{T^*(1, N)}{N_\pi \cdot T(N_\pi, N)} = \frac{S^*(N_\pi, N)}{N_\pi} \quad (2.48)$$

bzw.

$$\boxed{E(N_\pi, N) := \frac{T(1, N)}{N_\pi \cdot T(N_\pi, N)} = \frac{S(N_\pi, N)}{N_\pi}.} \quad (2.49)$$

Die Effizienz beschreibt, welcher Anteil der Laufzeit für sinnvolle Rechnungen verwendet wird. Die übrige Laufzeit wird z.B. für Datenverwaltung verbraucht, welche bei der Ausführung auf einem einzigen Prozessor nicht benötigt wird. Die optimale Effizienz ergibt sich daher zu

$$E_{\text{opt}}(N_\pi, N) = \frac{S_{\text{opt}}(N_\pi, N)}{N_\pi} = 1. \quad (2.50)$$

Mit der gleichen Begründung wie beim Speedup wird im Folgenden lediglich die Definition in Gleichung (2.49) weiter verwendet.

Im Folgenden sollen die Effekte untersucht werden, welche dazu führen, dass der ideale Speedup bzw. die ideale Effizienz nicht erreicht wird. Dazu wird ein fiktiver paralleler Algorithmus betrachtet, welcher jedoch Ähnlichkeiten mit dem in Abschnitt 2.3 vorgestellten PIC Algorithmus besitzt. Die folgenden Ausführungen werden daher bei der Konstruktion der parallelen PIC Algorithmen in Kapitel 3 hilfreich sein.

Der fiktive Algorithmus benötigt zur Berechnung seines Ergebnisses  $N_{\text{it}}$  Stufen, wobei die innerhalb einer Stufe durchzuführenden Rechenschritte das Ergebnis der vorherigen Stufe benötigen. Die Berechnungen, welche innerhalb der  $i$ -ten Stufe durchgeführt werden müssen, benötigen auf einem Referenzprozessor die Rechenzeit

$$T^{(i)}(1, N) = T_{\text{p}}^{(i)}(1, N) + T_{\text{s}}^{(i)}(1, N), \quad (2.51)$$

wobei die Berechnungen, welche die Rechenzeit  $T_{\text{p}}^{(i)}$  benötigen, als parallelisierbar angenommen werden können, d.h. die zur Fertigstellung erforderlichen Rechenoperationen können auf verschiedene Prozessoren verteilt werden. Der Anteil  $T_{\text{s}}^{(i)}$  hingegen wird durch Rechenoperationen verursacht, die nicht parallelisierbar sind, und somit bei einer parallelen Berechnung von jedem Prozessor ausgeführt werden müssen. Damit beträgt die Rechenzeit, welche auf dem Referenzprozessor zur Fertigstellung der Berechnung insgesamt benötigt wird,

$$T(1, N) = \sum_{i=1}^{N_{\text{it}}} T^{(i)}(1, N) = \sum_{i=1}^{N_{\text{it}}} [T_{\text{p}}^{(i)}(1, N) + T_{\text{s}}^{(i)}(1, N)]. \quad (2.52)$$

Wird der Algorithmus nun auf einem Parallelrechner mit  $N_\pi$  Prozessoren ausgeführt, so werden die Berechnungen des parallelisierbaren Anteils des Algorithmus auf die verfügbaren Prozessoren verteilt, so dass sich die Rechenzeit, welche Prozessor  $i_\pi$  für die Berechnungen der  $i$ -ten Stufe benötigt, angegeben werden kann als

$$T^{(i)}(i_\pi, N_\pi, N) = T_{\text{p}}^{(i)}(i_\pi, N_\pi, N) + T_{\text{s}}^{(i)}(1, N). \quad (2.53)$$

Da die Berechnungen, welche  $T_{\text{s}}^{(i)}$  verursachen, nicht parallelisierbar sind, tritt dieser Term in Gleichung (2.53) unverändert in Erscheinung. Die parallelisierbaren Berechnungen wer-

den auf die verfügbaren Prozessoren verteilt, so dass

$$\sum_{i_\pi=1}^{N_\pi} T_p^{(i)}(i_\pi, N_\pi, N) = T_p^{(i)}(1, N) \quad (2.54)$$

gelten muss. Die Berechnungen der  $i$ -ten Stufe benötigen die Ergebnisse der Stufe  $i-1$ . Dies impliziert, dass im Allgemeinen Daten zwischen den Prozessoren nach dem Abschluss der Berechnungen einer Stufe ausgetauscht werden müssen. Zudem können die Berechnungen der Stufe  $i+1$  nicht begonnen werden, bevor nicht die Ergebnisse von allen Prozessoren für die  $i$ -te Stufe vorliegen. Der Prozessor, welcher für die Berechnungen, sowie den darauf folgenden Datenaustausch am meisten Zeit benötigt, bestimmt also die Gesamtzeit, welche auf dem Parallelrechner für die Bearbeitung der  $i$ -ten Stufe des Algorithmus erforderlich ist. Mit  $\tilde{T}^{(i)}(i_\pi, N_\pi, N)$  als Zeit, die Prozessor  $i_\pi$  für den Austausch von Ergebnisdaten in der Stufe  $i$  benötigt, folgt somit für die Gesamtzeit, welche von dem Parallelrechner für die Bearbeitung der  $i$ -ten Stufe des Algorithmus benötigt wird

$$T^{(i)}(N_\pi, N) = \max_{i_\pi \in \Pi} \left\{ T_p^{(i)}(i_\pi, N_\pi, N) + \tilde{T}^{(i)}(i_\pi, N_\pi, N) \right\} + T_s^{(i)}(1, N), \quad (2.55)$$

wobei  $\Pi := \{1 \dots N_\pi\}$  als die Menge der Prozessoren, beschrieben durch eine eindeutige Identifikationsnummer, definiert ist. Die Laufzeit der Gesamtberechnung über alle  $N_{it}$  Stufen ergibt sich zu

$$T(N_\pi, N) = \sum_{i=1}^{N_{it}} T^{(i)}(N_\pi, N). \quad (2.56)$$

Für den parallelen Speedup der Gesamtberechnung folgt somit

$$S(N_\pi) = \frac{\sum_{i=1}^{N_{it}} \left[ T_p^{(i)}(1, N) + T_s^{(i)}(1, N) \right]}{\sum_{i=1}^{N_{it}} \left[ \max_{i_\pi \in \Pi} \left\{ T_p^{(i)}(i_\pi, N_\pi, N) + \tilde{T}^{(i)}(i_\pi, N_\pi, N) \right\} + T_s^{(i)}(1, N) \right]}. \quad (2.57)$$

Aus Gleichung (2.57) lässt sich erkennen, dass drei Faktoren den Speedup eines parallelen Algorithmus gegenüber  $S_{opt}$  reduzieren.

Zum Ersten führt der benötigte Datenaustausch zwischen den Prozessoren, beschrieben durch  $\tilde{T}^{(i)}$ , zu einem Mehraufwand, der bei der seriellen Ausführung nicht auftritt. Dieser Term wird sowohl durch die Menge an Daten, welche ausgetauscht werden müssen, als auch durch die Architektur des Parallelrechners und die Infrastruktur (Software, Protokolle), die dieser zum Austausch von Daten zwischen den Prozessoren anbietet, wie auch durch die konkrete Implementierung des Datenaustausches (Kommunikationsstruktur) beeinflusst. Da die Menge an Daten, welche zwischen den Prozessoren ausgetauscht werden müssen, und damit auch die für den Datenaustausch benötigte Zeit, oftmals direkt damit

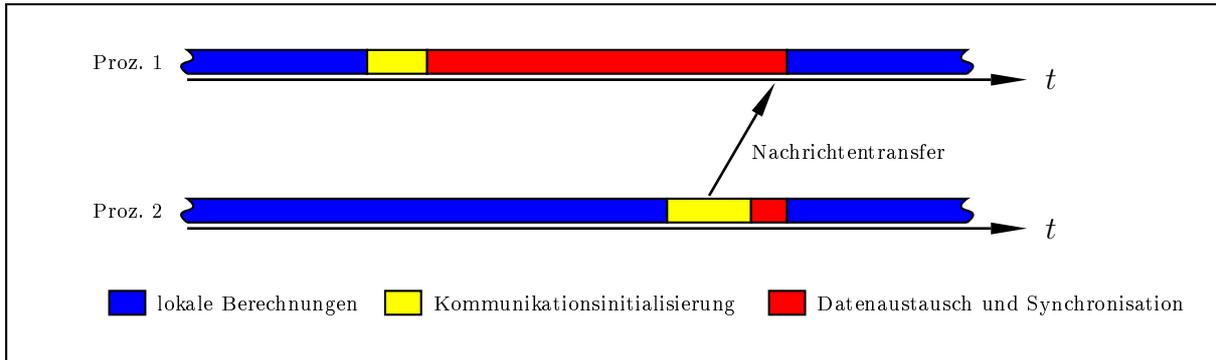
zusammenhängt, wie die parallelisierbaren Berechnungen den Prozessoren zugeordnet werden, *ist ein wesentliches Ziel bei der Entwicklung eines parallelen Algorithmus, basierend auf dessen Speicherzugriffsmuster die Berechnungen so aufzuteilen, dass jeder Prozessor möglichst wenige Daten von anderen Prozessoren benötigt.*

Zum Zweiten führt die durch den Datenaustausch eingeführte Synchronisation nach jeder Stufe des Algorithmus dazu, dass derjenige Prozessor, der für die Ausführung der ihm zugeordneten Berechnungen und Kommunikationsoperationen die längste Zeitspanne benötigt, die Gesamtzeit für die Stufe bestimmt. Ausgedrückt wird dies durch den Ausdruck  $\max\{\cdot\}$  in Gleichung (2.55). Wird die Verteilung der Daten so vorgenommen, dass die für den Datenaustausch benötigte Zeit gegenüber den Rechenzeiten so klein wird, dass sie vernachlässigt werden kann ( $\tilde{T}^{(i)} \approx 0$ ), so folgt aufgrund von Gleichung (2.54), dass

$$\begin{aligned} T^{(i)}(N_\pi, N) &= \max_{i_\pi \in \Pi} \{T_p^{(i)}(i_\pi, N_\pi, N)\} + T_s^{(i)}(1, N) \\ &\geq \frac{1}{N_\pi} T_p^{(i)}(1, N) + T_s^{(i)}(1, N) \end{aligned} \quad (2.58)$$

gilt. Somit ergibt sich, dass im Fall unbalancierter Lastverteilung aufgrund der Synchronisation eine Erhöhung der Rechenzeit auftritt, die den Speedup reduziert. **Die gleichmäßige Verteilung der Rechenlast auf die verfügbaren Prozessoren ist somit in diesem Fall essentiell, um einen möglichst hohen Speedup zu erzielen.** Das Gleichheitszeichen in Gleichung (2.58) gilt genau für den Fall idealer Lastbalancierung. Abbildung 2.7 illustriert den Sachverhalt unbalancierter Lastverteilung für den Fall zweier Prozessoren. Prozessor 1 besitzt weniger Rechenlast als Prozessor 2 und beendet seine Rechnung zu einem deutlich früheren Zeitpunkt. Da er auf die Daten von Prozessor 2 warten muss, kann er aber erst mit seinen Berechnungen fortfahren, nachdem Prozessor 2 seine Berechnung abgeschlossen hat und die benötigten Daten zur Verfügung stellen kann. Prozessor 2 bestimmt hier folglich die Laufzeit der Gesamtberechnung. In vielen Fällen sind die beiden Ziele, den Kommunikationsaufwand zu minimieren, sowie eine optimale Lastbalancierung zu erreichen, nicht miteinander vereinbar, so dass ein Kompromiss zwischen beiden Zielen gefunden werden muss, welcher die Gesamtlaufzeit minimiert. Sei nun noch angenommen, dass Lastbalancierung zu jedem Zeitpunkt der Berechnung gewährleistet ist, so vereinfacht sich der Ausdruck für den Speedup (2.57) zu

$$S(N_\pi, N) = \frac{1 + \frac{T_s(1, N)}{T_p(1, N)}}{\frac{1}{N_\pi} + \frac{T_s(1, N)}{T_p(1, N)}}. \quad (2.59)$$



**Abbildung 2.7:** Ungleich verteilte Last führt zu unterschiedlichen Laufzeiten für die beteiligten Prozessoren. Der am stärksten belastete Prozessor bestimmt, bedingt durch die Synchronisationspunkte, welche durch jeden Datenaustausch eingeführt werden, die Laufzeit.

Da durch die Lastbalancierung alle  $T_p^{(i)}(i_\pi, N_\pi, N)$  gleich werden, konnte der  $\max\{\cdot\}$  Ausdruck fortgelassen werden. Zudem wurde

$$T_p(N_\pi, N) := \sum_{i=1}^{N_{it}} T_p^{(i)}(N_\pi, N) \tag{2.60}$$

und

$$T_s(1, N) := \sum_{i=1}^{N_{it}} T_s^{(i)}(1, N) \tag{2.61}$$

gesetzt. Damit kann der Speedup selbst bei Verwendung von extrem vielen Prozessoren den Wert von

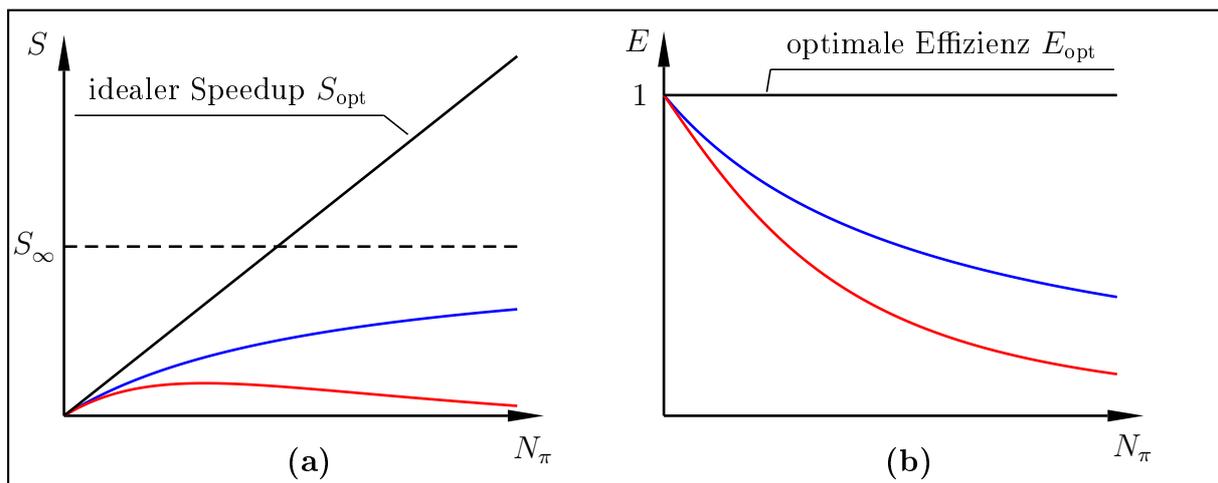
$$S_\infty(N) := \lim_{N_\pi \rightarrow \infty} S(N_\pi, N) = 1 + \frac{T_p(1, N)}{T_s(N)} \tag{2.62}$$

nicht überschreiten. Der Ausdruck in (2.62) ist als Gesetz von AMDAHL bekannt geworden [50] und zeigt die dritte Ursache für einen nicht idealen Speedup: Die seriellen Berechnungen, welche bei der Ausführung des Algorithmus von allen Prozessoren durchgeführt werden müssen, verhindern, dass der Speedup den Wert in (2.62) überschreitet, gleichgültig wie viele Prozessoren benutzt werden. **Ziel muss daher sein, möglichst alle Teile eines Algorithmus zu parallelisieren.** Zur Herleitung dieses Ergebnisses wurde ideale Lastbalancierung vorausgesetzt sowie jeglicher Aufwand für den Datenaustausch zwischen Prozessoren vernachlässigt. Der Speedup einer realen Implementierung nähert sich daher typischerweise nicht asymptotisch diesem Wert an, sondern nimmt bei der Erhöhung der Prozessorenzahl nach dem Erreichen eines Maximums wieder ab, da der Kommunikationsaufwand üblicherweise mit wachsender Prozessorenzahl im Vergleich zum Rechenaufwand ansteigt und zudem, je nach Art des betrachteten Algorithmus, die Lastbalancierung mit

wachsender Anzahl an Prozessoren häufig schwieriger zu gewährleisten ist.

Abbildung 2.8(a) zeigt das typische Verhalten des Speedup in Abhängigkeit von der Prozessorenzahl. Während sich die obere Kurve, welche den Kommunikationsaufwand vernachlässigt, asymptotisch dem in Gleichung (2.62) berechneten Wert nähert, erreicht die untere Kurve, welche den Kommunikationsaufwand berücksichtigt, einen Extremalwert und fällt dann bei weiterer Erhöhung der Prozessorenzahl ab. Die Verläufe, welche sich für die Effizienz in den beiden Fällen ergeben, sind in Abbildung 2.8(b) dargestellt. Die Effizienz besitzt definitionsgemäß ihr Maximum für  $N_\pi = 1$ . Wird  $N_\pi$  erhöht, fällt die Effizienz üblicherweise kontinuierlich ab, obwohl auch hier aus den gleichen Gründen wie für den Speedup in einigen Fällen Werte oberhalb von 1 möglich sind [49].

Ein Kritikpunkt an der Speedup- sowie der Effizienzmetrik ist, dass diese prinzipiell Systeme mit langsamen Prozessoren bevorzugen, d.h. diese Systeme erhalten höhere Speedup- bzw. Effizienzwerte verglichen mit Systemen, welche schnellere Prozessoren besitzen [45]. Dies ist aus Gleichung (2.57) ersichtlich. Für langsame Prozessoren können die Terme  $T_p$



**Abbildung 2.8:** (a) Der typische Verlauf des parallelen Speedup für eine feste Problemgröße. Während sich der Speedup bei Vernachlässigung des Kommunikationsaufwandes einem Maximalwert  $S_\infty$  asymptotisch nähert, erreicht er bei Berücksichtigung desselben einen Extremalwert und fällt danach ab. (b) Der typische Verlauf der Effizienz für konstante Problemgröße. Die Kurven gleicher Farbe in den beiden Bildern korrespondieren miteinander.

sowie  $T_s$  gegenüber dem Term  $\tilde{T}$  groß werden, was zu einem höheren Speedup bzw. zu einer größeren Effizienz führt. Diese besseren Werte werden jedoch durch eine hohe Gesamtlaufzeit erkauft. Somit muss zu einer sinnvollen Bewertung eines parallelen Systems immer auch die Gesamtlaufzeit  $T(N_\pi, N)$  herangezogen werden, um nicht zu falschen Schlussfolgerungen zu gelangen.

### 2.4.3.3 Skalierter Speedup und skalierte Effizienz

Die pessimistische Sicht, welche sich aus dem AMDAHLschen Gesetz (2.62) ergibt, folgt daraus, dass die Problemgröße konstant gehalten wird, während die Anzahl der Prozessoren immer weiter erhöht wird. Somit erhält der serielle Anteil der Laufzeit sowie auch die Zeit, welche für den Datenaustausch benötigt wird, einen immer höheren Anteil an der Gesamtlaufzeit.

Häufig tritt jedoch der Fall auf, dass Instanzen verschiedener Größe ein und desselben Problems gelöst werden müssen. Man denke nur an Konvergenzstudien, die bei vielen Simulationen unerlässlich sind. Vielfach besteht hier der Wunsch, alle Problemgrößen in der gleichen Zeit zu lösen, und, um dies zu erreichen, eine an die Problemgröße angepasste Anzahl an Prozessoren zu benutzen. Dies motiviert die Einführung des skalierten Speedup für eine konstante Laufzeit [51]. Dieser ist definiert als

$$S_T(N_\pi, T_0) := \frac{T(1, N(T_0, N_\pi))}{T_0}. \quad (2.63)$$

Hierbei ist  $T_0$  eine festgelegte Laufzeit. Für eine bestimmte Anzahl an Prozessoren ergibt sich damit die Problemgröße  $N$  aus der Beziehung

$$T(N_\pi, N) \stackrel{!}{=} T_0. \quad (2.64)$$

Ebenso lässt sich die skalierte Effizienz einführen als

$$E_T(N_\pi, T_0) := \frac{S_T(N_\pi, T_0)}{N_\pi}. \quad (2.65)$$

Von GUSTAFSON [52] wurde der speicherskalierte Speedup eingeführt. Dieser ist definiert als

$$S_M(N_\pi, N) := \frac{T(1, N_\pi \cdot N)}{T(N_\pi, N_\pi \cdot N)}. \quad (2.66)$$

Analog lässt sich die speicherskalierte Effizienz definieren als

$$E_M(N_\pi, N) := \frac{S_M(N_\pi, N)}{N_\pi}. \quad (2.67)$$

Wie in Gleichung (2.63) wird auch hier die Problemgröße mit der Anzahl der Prozessoren erhöht. Während in Gleichung (2.63) jedoch die Laufzeit konstant gehalten wird und über Gleichung (2.64) auf die Problemgröße geschlossen wurde, wird diese nun proportional zu der Anzahl der Prozessoren erhöht. Dies kann in dem oft auftretenden Fall, dass der Speicherbedarf des parallelen Systems linear mit der Problemgröße ansteigt, so interpretiert werden, dass der Anteil des insgesamt verfügbaren Arbeitsspeichers, welcher dem parallelen System zur Verfügung gestellt wird, konstant gehalten wird, was dieser Metrik ihren Namen einbrachte.

#### 2.4.3.4 Skalierbarkeit und Isoeffizienzfunktion

Wie in Abschnitt 2.4.3.2 erläutert, kann bei einer konstanten Problemgröße  $N$  durch Erhöhung der Prozessorenzahl  $N_\pi$  die Effizienz  $E$  in der Regel nicht konstant gehalten werden. Jedoch ist zu beobachten, dass eine Erhöhung der Problemgröße bei einer festen Anzahl an Prozessoren oftmals zu einem Ansteigen der Effizienz führt. In einem solchen Fall ist es möglich durch eine gleichzeitige Erhöhung der Problemgröße  $N$  und der Prozessorenzahl  $N_\pi$  die Effizienz konstant zu halten. Parallele Systeme, welche diese Eigenschaft besitzen, werden als skalierbar bezeichnet [47].

Die Skalierbarkeit ist ein wichtiges Qualitätsmerkmal eines parallelen Systems, da es nur für skalierbare Systeme möglich ist, bei einer gegebenen Problemgröße eine Anzahl von Prozessoren auszuwählen, so dass die Ressourcen mit einer hohen Effizienz genutzt werden. Bei nicht skalierbaren Systemen führt eine Erhöhung der Prozessorenzahl zwangsläufig zu einer Verschlechterung der Effizienz. Da eine Erhöhung der Problemgröße in der Regel auch mit einer Vergrößerung des Speicherbedarfs einhergeht, und damit eine Erhöhung der Anzahl der verwendeten Prozessoren/Rechenknoten (auf Multicomputern) unumgänglich macht, sind nicht skalierbare Algorithmen höchstens für kleine Problemgrößen geeignet. Der funktionale Zusammenhang zwischen Problemgröße  $N$  und Prozessorenzahl  $N_\pi$ , welcher sich für eine konstante Effizienz  $\mathcal{E}$  ergibt, wird als Isoeffizienzfunktion bezeichnet [53]. Die Isoeffizienzfunktion ist somit in impliziter Form definiert als

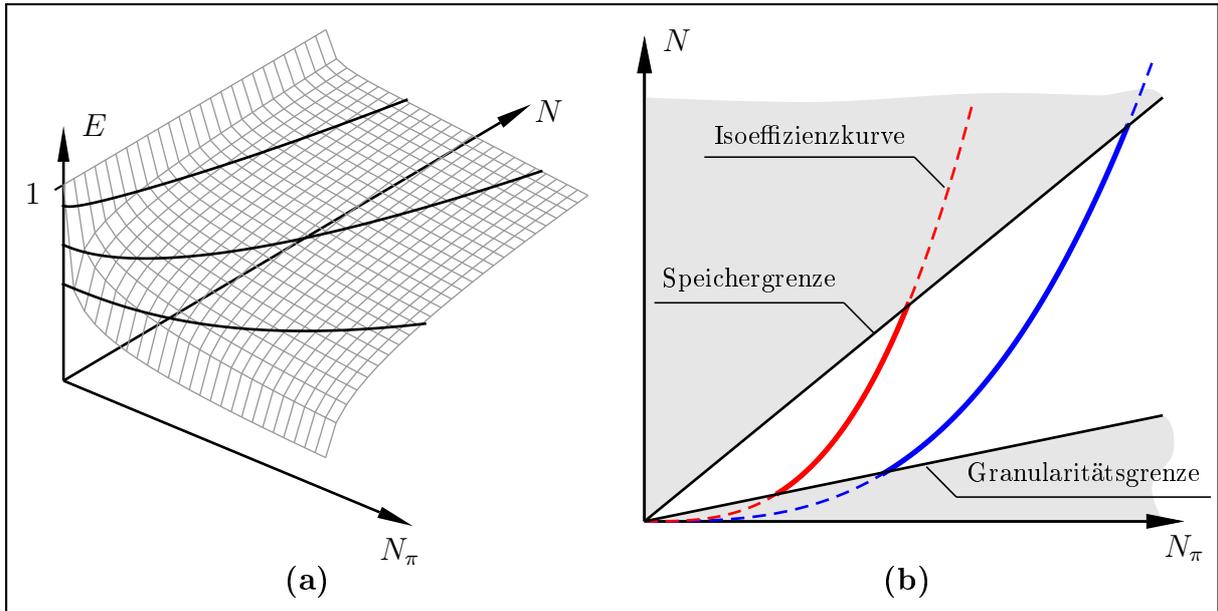
$$E(N_\pi, N) \stackrel{!}{=} \mathcal{E}. \quad (2.68)$$

Abbildung 2.9 zeigt beispielhaft die Konstruktion der Isoeffizienzkurven als Höhenlinien der Funktion  $E(N_\pi, N)$  sowie die Isoeffizienzlinien aufgetragen in der  $N_\pi N$ -Ebene. Der explizite Zusammenhang zwischen  $N$  und  $N_\pi$ , welcher sich durch Auflösen von Gleichung (2.68) nach  $N$  ergibt, werde bezeichnet mit  $\overline{E}_\mathcal{E}$ . Damit ergibt sich

$$N = \overline{E}_\mathcal{E}(N_\pi). \quad (2.69)$$

Man beachte, dass nicht jede beliebige Kombination von Problemgröße  $N$  und Prozessorenzahl  $N_\pi$  mit Hilfe eines parallelen Systems realisiert werden kann.

Mit der Problemgröße  $N$  nimmt im Allgemeinen auch der Speicherbedarf zu. Da der Speicher jedoch eine knappe Ressource ist, existiert für jede Prozessorenzahl  $N_\pi$  eine maximale Problemgröße  $N_{\max}(N_\pi)$ , die nicht überschritten werden kann. Bei Multicomputern bedeutet die Hinzunahme weiterer Prozessoren bzw. Rechenknoten, dass der verfügbare Speicher sich proportional erhöht, sofern alle Rechenknoten einen gleich großen Speicher besitzen. Verhält sich der Speicherbedarf des parallelen Systems ebenfalls proportional zur Problemgröße, so ist die Funktion  $N_{\max}(N_\pi)$  eine lineare Funktion, wie Abbildung 2.9(b) zeigt.



**Abbildung 2.9:** (a) Konstruktion der Iseffizienzkurven als Höhenlinien der Funktion  $E(N_\pi, N)$ . (b) Iseffizienzkurven in der  $N_\pi N$ -Ebene. Die rote Kurve ergibt sich für eine höhere Effizienz als die blaue .

Aufgrund der endlichen Granularität eines parallelen Systems, d.h. die Berechnungen zur Lösung des Problems lassen sich nicht in beliebig viele Teile zerlegen, ist es jedoch auch nicht möglich ein Problem bestimmter Größe auf beliebig vielen Prozessoren zu lösen. Für jede Prozessorenzahl existiert daher auch eine minimale Problemgröße  $N_{\min}(N_\pi)$ . Für den häufig auftretenden Fall, dass die Anzahl der Teilrechnungen, in die sich das Gesamtproblem maximal zerlegen lässt, proportional zu der Problemgröße ist, ergibt sich für  $N_{\min}(N_\pi)$  eine lineare Funktion, wie in Abbildung 2.9(b) gezeigt.

Wie bereits angesprochen, führt ein höherer Wert von  $\mathcal{E}$  zu einer Iseffizienzfunktion mit einer größeren Steigung. Die beiden in Abbildung 2.9(b) aufgetragenen Iseffizienzlinien zeigen dieses Verhalten. Hier wird auch deutlich, dass nur für den Fall

$$N_{\min}(N_\pi) \leq \bar{E}_\mathcal{E}(N_\pi) \leq N_{\max}(N_\pi) \quad (2.70)$$

eine konstante Effizienz  $\mathcal{E}$  mit dem parallelen System für eine beliebige Prozessorenzahl erreicht werden kann. Ist eine der Bedingungen in Gleichung (2.70) verletzt, so wird die Effizienz des parallelen Systems ab einer bestimmten Prozessorenzahl nicht mehr konstant gehalten werden können.

## 2.4.4 Modelle für Parallelrechner

Um die Laufzeit eines konkreten Systems einer theoretischen Analyse zugänglich zu machen, ist ein Modell des Rechners erforderlich, auf welchem der Algorithmus ausgeführt werden soll. Erst ein solches Modell ermöglicht es letztlich, die Laufzeiten, welche in Abschnitt 2.4.3 zunächst als abstrakte Größen eingeführt wurden, zu konkretisieren.

Um von Nutzen zu sein, muss ein Rechnermodell einige Anforderungen erfüllen. Die erste Forderung betrifft die Komplexität des Modells. Das Modell muss einerseits so komplex sein, dass es die dominanten Effekte, welche die Laufzeit eines Systems beeinflussen, abbildet. Jedoch darf die Komplexität nicht so hoch sein, dass keine konkreten Aussagen darüber mehr möglich sind, durch welche Strategien ein System optimiert werden kann. Solche Aussagen werden bei sehr komplexen Modellen häufig hinter zahllosen Modellparametern bzw. einer komplizierten Modellstruktur versteckt.

Eine weitere wünschenswerte Eigenschaft eines Rechnermodells ist seine Portabilität, d.h. seine Eignung zur Modellierung einer möglichst großen Klasse von Systemen. Dies ist in der Regel nur dann möglich, wenn die betrachteten Rechner eine ähnliche Architektur besitzen.

Zur Analyse serieller Systeme hat sich das Modell der **R**andom **A**ccess **M**achine (RAM) weitgehend durchgesetzt [54]. Es handelt sich dabei um einen Rechner, welcher über einen unbegrenzten und homogenen Arbeitsspeicher verfügt und mit einer konstanten Laufzeit die folgenden Operationen ausführen kann, wobei die jeweils benötigte Laufzeit für die verschiedenen Operationen unterschiedlich sein kann:

- Ausführung eines arithmetischen Befehls (Addition, Subtraktion, Multiplikation, Division) zur Kombination von zwei Zahlen.
- Ausführung eines Transportbefehls, d.h. das Überschreiben einer Speicherstelle durch einen Wert, welcher an einer anderen Stelle im Speicher steht.
- Ausführung eines Lese- oder Schreibbefehls, welcher eine Zahl aus dem Speicher einliest oder eine Zahl in den Speicher schreibt.

Für Algorithmen unterschiedlichster Art wurde die benötigte Rechenzeit in Abhängigkeit der Problemgröße für das RAM Modell analysiert und umfangreiche Literatur mit derartigen Analysen ist verfügbar (siehe z.B. [55]). Der große Erfolg des RAM Modells als Basis zur Entwicklung und Analyse von Algorithmen begründet sich daraus, dass es trotz seines hohen Abstraktionsgrades und der damit einhergehenden geringen Komplexität recht genaue Aussagen über die zu erwartende Laufzeit eines Algorithmus in Abhängigkeit von der Problemgröße erlaubt.

Da Parallelrechner gegenüber seriellen Rechnern eine wesentlich kompliziertere Architek-

tur besitzen, ist auch das Aufstellen eines Modells mit den oben genannten Eigenschaften wesentlich komplizierter. Historisch wurde die Entwicklung eines Modells noch dadurch erschwert, dass viele Parallelrechner mit unterschiedlicher Architektur existierten. Diese waren oftmals optimiert für bestimmte Rechenoperationen, während andere Rechenoperationen nur ineffizient ausgeführt werden konnten. Hinzu kam eine Vielzahl unterschiedlicher Netzwerktechnologien, welche bei der Anbindung der Speichermodule an die Prozessoren der Rechner zum Einsatz kamen. Dies betrifft sowohl die Topologie des Netzwerks wie auch Routing Methoden und Netzwerkprotokolle. Obwohl mittlerweile eine gewisse „Konvergenz“ der Rechnerarchitekturen hin zu Parallelrechnern, welche aus Komponenten des Massenmarktes als Multicomputer oder Multiprozessoren aufgebaut sind, zu beobachten ist, gibt es bislang kein Modell für Parallelrechner, welches die gleiche Akzeptanz und Verbreitung gefunden hat wie das RAM Modell für serielle Rechner.

Es sollen daher im Folgenden einige Modelle für Parallelrechner kurz vorgestellt werden, welche eine gewisse Popularität erlangt haben, um im Anschluss daran das im Rahmen dieser Arbeit für die Analyse der untersuchten parallelen Systeme verwendete Modell vorzustellen. Für eine recht umfassende Übersicht über Modelle für Parallelrechner sei auf [56] verwiesen.

#### 2.4.4.1 Parallele RAM

Eines der frühesten Parallelrechnermodelle war eine Erweiterung des RAM Modells, das Modell der **P**arallelen **R**AM (PRAM) [57]. Es handelt sich dabei um eine Maschine mit einer Anzahl an synchron getakteten, gleichartigen Prozessoren, welche auf einem unbeschränkten, gemeinsamen Speicher arbeiten. Der Zugriff auf jede Position des Speichers ist für jeden der Prozessoren in einem Prozessortakt möglich.

Das PRAM Modell war jedoch, obwohl es eine recht weite Verbreitung fand, zur Laufzeitmodellierung paralleler Systeme nicht sehr erfolgreich. Insbesondere zeigte sich, dass die Annahme eines homogenen, gemeinsamen Speichers für die zunehmend populär werden den Multicomputer zu unzutreffenden Voraussagen über die Laufzeit führt, da auf realen Rechnern ein signifikanter Unterschied zwischen der Zugriffszeit für den lokalen und den nicht lokalen Speicher existiert und die Prozessoren üblicherweise asynchron getaktet sind. Obwohl zahlreiche Erweiterungen des PRAM Modells vorgeschlagen wurden, um die Qualität des Modells zu verbessern (siehe z.B. [58] für eine Übersicht), konnte keines dieser Modelle eine große Akzeptanz erreichen.

#### 2.4.4.2 Bulk-Synchronous Parallel Computer

Das Modell des **Bulk-Synchronous Parallel Computers** (BSPC) [59] modelliert die benötigte Zeit für den Austausch von Daten zwischen den Prozessoren eines Parallelrechners als Einflussgröße auf die Laufzeit. Ein BSPC ist eine Maschine bestehend aus einer bestimmten Anzahl Prozessoren, welche in bestimmten festen Zeitintervallen synchronisiert werden. Diese Zeitintervalle werden als Superschritte bezeichnet. Prinzipiell folgt auf eine Phase von Berechnungen, welche ausschließlich mit lokalen Daten auskommen, eine Phase, in der Daten zwischen den Prozessoren ausgetauscht werden. Die beiden Phasen laufen sequentiell ab und können jeweils eine unterschiedliche Anzahl an Superschritten benötigen. Die Modellierung des Datenaustausches geschieht über sogenannte  $h$ -Relationen. Dabei handelt es sich um den Austausch von höchstens  $h$  Datenelementen gleicher Größe zwischen den Prozessoren. Eine  $h$ -Relation benötigt zu ihrer Bearbeitung  $g \cdot h + s$  Zeiteinheiten, wobei  $g$  als Bandbreite des Verbindungsnetzwerks und  $s$  als Zeit, welche für das Vorbereiten der Datenelemente zur Übertragung benötigt wird, interpretiert werden kann. Es spielt dabei keine Rolle, welche der Prozessoren Datenelemente in welcher Reihenfolge austauschen.

Obwohl das Modell des BSPC die Annahme des PRAM Modells hinsichtlich der Synchronisation der Prozessoren zu einer Synchronisation nach jedem Superschritt abschwächt und auch die Zeit für den Austausch von Daten modelliert, ist die Annahme einer Synchronisation in festen Zeitabständen häufig zu restriktiv. Es wurde daher zum LogP Modell weiterentwickelt, welches einen Parallelrechner mit vollkommen asynchron arbeitenden Prozessoren zugrunde legt.

#### 2.4.4.3 LogP Modell

Das LogP Modell [60] verdankt seinen Namen den vier Parametern, die zur Beschreibung des Parallelrechners benutzt werden. Diese vier Parameter sind die Anzahl der Prozessoren ( $P$ ), die minimale Zeit, welche zwischen dem Empfang zweier Datenelemente durch ein und denselben Prozessor vergehen muss ( $g$ ), die benötigte Zeit, um ein Datenelement für die Übertragung vorzubereiten ( $o$ ), sowie die Latenzzeit ( $L$ ), die für die Übertragung eines Datenelementes von einem Prozessor zu einem anderen vergeht. Das LogP Modell trifft keine Annahmen über die Synchronisation der Prozessoren.

Die Inverse des Parameters  $g$  kann als die pro Prozessor zur Verfügung stehende maximale Bandbreite für die Datenübertragung interpretiert werden. Somit wird das Modell sensitiv auf die Kommunikationsstruktur, d.h. die für einen Datenaustausch benötigte Zeit hängt nicht nur von der Menge der verschickten Daten ab, sondern auch davon, zwischen welchen Prozessoren und in welcher Reihenfolge der Datenaustausch erfolgt. So führt bei-

spielsweise eine Kommunikationsoperation, bei der jeder Prozessor gleichzeitig Daten an einen einzigen Prozessor verschickt, unter Umständen dazu, dass die Netzwerkanbindung des Prozessors überlastet wird und als Folge die Daten mit der maximal zur Verfügung stehenden Rate  $1/g$  empfangen werden. Die übrigen Prozessoren müssen so lange warten, bis ihre Nachricht empfangen wurde. Somit ist die vom LogP Modell prognostizierte Laufzeit für eine solche Operation größer als die Laufzeit für einen balancierten Datenaustausch, bei dem jeder Prozessor etwa gleich große Datenmengen sendet und empfängt, und die Netzwerkanbindung der einzelnen Prozessoren gleichmäßig belastet wird.

Für eine einzige Punkt-zu-Punkt Kommunikationsoperation prognostiziert das LogP Modell eine Abhängigkeit der Kommunikationsdauer  $\tilde{T}^{\text{LogP}}$  von der verschickten Datenmenge  $m$  in der Form

$$\tilde{T}^{\text{LogP}} = t_s + t'_w \cdot m, \quad (2.71)$$

wobei  $t_s$  und  $t'_w$  Funktionen der Parameter  $o$ ,  $g$  bzw.  $L$  sind. Gleichung (2.71) legt nahe, dass es von Vorteil ist, den Austausch von Daten, wenn möglich, mit Hilfe einer einzigen großen, anstatt mit einer Vielzahl kleiner Nachrichten durchzuführen. Der konstante Anteil  $t_s$  tritt dann lediglich ein einziges Mal in Erscheinung. In diesem Fall kann der Term  $t_s$  gegenüber dem sehr viel größeren Anteil  $t'_w \cdot m$  vernachlässigt werden und man erhält

$$\tilde{T}^{\text{LogP}} \approx t'_w \cdot m. \quad (2.72)$$

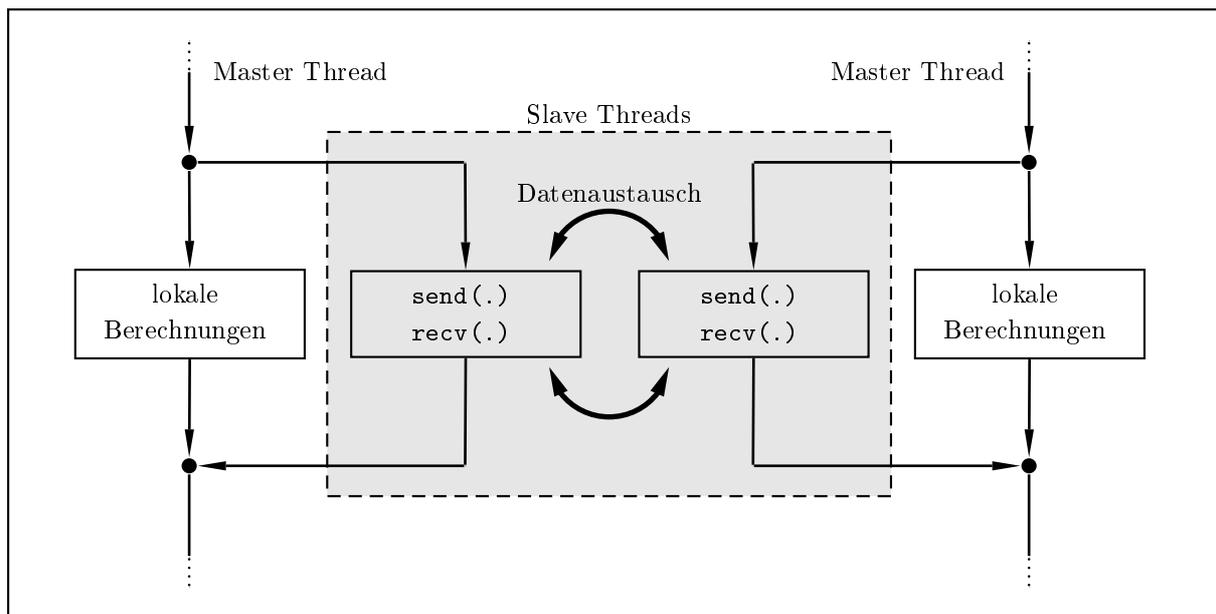
Diese Zeit beinhaltet sowohl einen Anteil für den eigentlichen Datentransfer über das Netzwerk, als auch einen Anteil für Operationen, die für die Vorbereitung des Transfers erforderlich sind. Abbildung 2.11(a) zeigt, dass das Modell eine gute Übereinstimmung mit Messungen der (mittleren) Kommunikationszeit aufweist, welche für zwei Rechnercluster durchgeführt wurden.

### 2.4.5 Im Rahmen der Arbeit verwendetes Rechnermodell

Im Rahmen der Arbeit wurde zur Abschätzung von Laufzeiten das folgende Rechnermodell verwendet. Zur Modellierung von Berechnungsoperationen, welche vollständig auf lokalen Daten arbeiten, wurde das RAM Modell eingesetzt. Für die Modellierung von Kommunikationsoperationen wurde das LogP Modell verwendet. Wie in Abschnitt 2.4.4.3 erläutert, ist es sinnvoll, die auszutauschenden Daten gebündelt in möglichst wenigen Nachrichten zu verschicken. Es wird daher im Folgenden angenommen, dass die verschickten Nachrichten dadurch so groß werden, dass die mittlere zur Übertragung benötigte Zeit gemäß Gleichung (2.72) als proportional zur Nachrichtengröße angenommen werden kann.

An dieser Stelle erscheint es sinnvoll, darauf einzugehen, welche Annahmen in diesem Modell enthalten sind, und welche Effekte somit unberücksichtigt bleiben.

- Überlappen von Datenaustausch und Berechnungen.** Auf einigen Parallelrechnern ist es möglich, dass das Versenden und Empfangen von Nachrichten nicht der direkten Beteiligung der CPU bedarf und daher im Hintergrund erfolgen kann. Durch spezialisierte Hardware, bei der ein Coprozessor das Senden und Empfangen der Nachrichten übernimmt, müssen Sende- und Empfangsoperationen von der CPU lediglich initialisiert und abgeschlossen werden. Während die Daten über das Netzwerk transportiert werden, kann die CPU weitere Rechnungen durchführen, welche der verschickten Daten nicht bedürfen. Dies ist als „Verstecken der Latenzzeit“ (engl. „latency hiding“) bekannt. Dies ist auch bei der Verwendung eines Multicomputers bestehend aus mehreren Prozessoren pro Knoten möglich. Während ein Prozessor Berechnungen durchführt, behandelt ein anderer Prozessor den Austausch der Daten (siehe Abbildung 2.10). Dies muss jedoch explizit implementiert werden. Obwohl diese Technik bei den im Rahmen der Arbeit durchgeführten Implementierungen nicht zum Einsatz kam, wird bei der Vorstellung der parallelen Algorithmen im folgenden Kapitel dennoch an geeigneter Stelle darauf hingewiesen, wie eine solche Implementierung aussehen könnte.

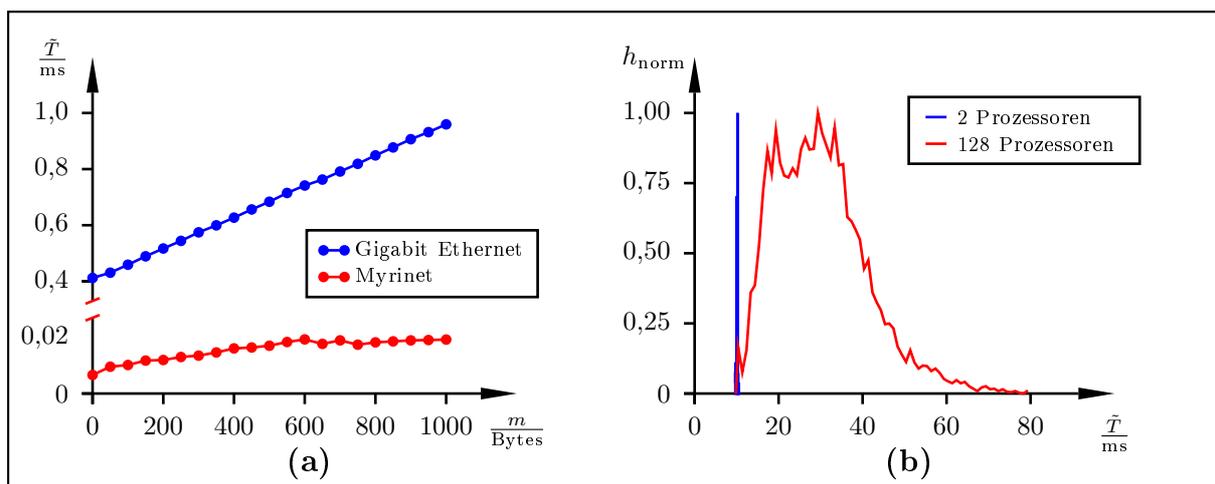


**Abbildung 2.10:** Durch die Verwendung mehrerer Threads lässt sich die Zeit, welche für den Datenaustausch benötigt wird, mit lokalen Berechnungen füllen.

- Netzwerküberlastung.** Der Einfluss der Netzwerklast auf die Nachrichtenübertragungszeiten bleibt unberücksichtigt. Eine Modellierung dieses Effekts ist extrem schwierig, da sich in einem Zustand der Netzwerküberlastung starke Schwankungen der Nachrichtenübertragungszeit beobachten lassen, die sich höchstens mit statisti-

schen Methoden sinnvoll modellieren lassen (siehe z.B. [61, 56] für eine ausführliche Behandlung dieses Effekts). Abbildung 2.11(b) illustriert den Effekt. Abgebildet ist die auf ihren Maximalwert normierte Häufigkeitsverteilung von gemessenen Übertragungszeiten von Punkt-zu-Punkt Nachrichten. Während die gemessenen Zeiten bei einem einzigen kommunizierenden Prozessorpaar sehr stark bei einem Wert konzentriert sind, lässt sich bei 64 gleichzeitig kommunizierenden Prozessorpaaren bereits eine starke Verbreiterung der Verteilung beobachten und die mittlere Kommunikationszeit wächst deutlich an. Je mehr Prozessoren gleichzeitig miteinander kommunizieren, desto breiter wird die gezeigte Verteilungsfunktion.

- **Reihenfolge von Nachrichten.** Es wird nicht die genaue Reihenfolge betrachtet, oder gar eine Reihenfolge erzwungen, in der die Nachrichten verschickt oder empfangen werden. Werden mehrere Nachrichten an unterschiedliche Prozessoren verschickt, so wird die dazu benötigte Zeit als Summe der einzelnen Kommunikationszeiten angenommen.
- **Einfluss fremder Prozesse.** Das Modell berücksichtigt nicht das eventuelle Vorhandensein fremder Prozesse und deren Einfluss auf die verfügbare Prozessorleistung sowie die Netzwerklast. Eine Untersuchung dieser Einflüsse findet sich z.B. in [62].
- **Annahme eines flachen Speichers.** Es wird ein flacher Speicher angenommen, d.h. die Einflüsse der Speicherhierarchie (Cache) werden nicht explizit modelliert. An entsprechenden Stellen wird jedoch auf Effekte, die durch die Speicherhierarchie bedingt sind, hingewiesen, sofern ein starker Einfluss auf die Laufzeit zu beobachten ist.



**Abbildung 2.11:** (a) zeigt die mittlere benötigte Zeit für den Austausch einer Nachricht variabler Größe zwischen zwei Prozessoren für unterschiedliche Netzwerktechnologien. (b) zeigt die normierte Häufigkeitsverteilung, welche sich beim Austausch einer Nachricht der Größe 10 KByte zwischen einer unterschiedlichen Anzahl Prozessoren, welche über ein Gigabit Ethernet Netzwerk verbunden sind, ergibt.



---

## 3 Parallelisierungsstrategien für PIC

Im folgenden Kapitel sollen Strategien zur Parallelisierung des PIC Algorithmus für Multicomputer vorgestellt werden. Zunächst wird in Abschnitt 3.1 hergeleitet, wie eine laufzeitoptimale Zuordnung von Berechnungen im Falle eines Parallelrechners mit Prozessoren beliebiger Leistungsfähigkeit erfolgen kann. In Abschnitt 3.2 wird der Begriff der Parallelisierungsstrategie im Rahmen eines zunächst abstrakten Optimierungsproblems formuliert. In Abschnitt 3.3 wird dieses für den Fall der Parallelisierung des Feldlösers ohne Anwesenheit von Teilchen konkretisiert. Im anschließenden Abschnitt 3.4 wird das Problem unter Anwesenheit von Teilchen formuliert, und verschiedene Parallelisierungsstrategien werden vorgestellt, um das Optimierungsproblem zu lösen. Innerhalb dieses Abschnittes werden im Rahmen dieser Arbeit entwickelte Parallelisierungsstrategien, sowie die Verallgemeinerung einer aus der Literatur bekannten Strategie vorgestellt.

### 3.1 Lastverteilung für Parallelrechner mit Prozessoren unterschiedlicher Leistungsfähigkeit

In diesem Abschnitt soll untersucht werden, wie die Rechenlast, welche bei der Lösung eines Problems anfällt, auf die Prozessoren eines Parallelrechners verteilt werden muss, um die Laufzeit zu minimieren. Betrachtet werde ein Problem, zu dessen Lösung  $C$  voneinander unabhängige, gleichartige Rechenschritte ausgeführt werden müssen. Das Problem soll mit Hilfe eines Parallelrechners mit  $N_\pi$  Prozessoren gelöst werden, wobei Prozessor  $i_\pi$  einen der Rechenschritte in  $\lambda_{i_\pi}$  Zeiteinheiten bearbeiten kann. Die Laufzeit  $T(N_\pi, C)$ , welche zur Lösung des Problems benötigt wird, wird von der Laufzeit des Prozessors bestimmt, welcher für die ihm zugeordneten Berechnungen die meiste Zeit benötigt

$$T(N_\pi, C) = \max_{i_\pi \in \{1 \dots N_\pi\}} \{T(i_\pi, N_\pi, C)\}. \quad (3.1)$$

Gesucht ist die Anzahl an Rechenoperationen  $C_{i_\pi}$ , die Prozessor  $i_\pi$  zugeordnet werden müssen, so dass  $T(N_\pi, C)$  minimal wird. Dazu muss offenbar

$$T(i_\pi, N_\pi, C) = T(j_\pi, N_\pi, C) \quad \forall i_\pi, j_\pi \in \{1 \dots N_\pi\} \quad (3.2)$$

gelten. Anderenfalls gibt es einen Prozessor  $i_\pi \in \{1 \dots N_\pi\}$ , für den

$$T(i_\pi, N_\pi, C) \geq T(j_\pi, N_\pi, C) \quad \forall j_\pi \in \{1 \dots N_\pi\} \quad (3.3)$$

gilt. Durch Abgabe von Berechnungen an die übrigen Prozessoren lässt sich  $T(i_\pi, N_\pi, C)$  und somit die Gesamtlaufzeit senken. Dies ist so lange möglich, bis Gleichung (3.2) erfüllt ist. Mit Hilfe der  $C_{i_\pi}$  und  $\lambda_{i_\pi}$  lässt sich Gleichung (3.2) schreiben als

$$\lambda_{i_\pi} C_{i_\pi} = \lambda_{j_\pi} C_{j_\pi} \quad \forall i_\pi, j_\pi \in \{1 \dots N_\pi\}. \quad (3.4)$$

Wegen

$$\sum_{i_\pi=1}^{N_\pi} C_{i_\pi} = C \quad (3.5)$$

lässt sich sofort der Zusammenhang

$$\boxed{\frac{C_{i_\pi}}{C} = \frac{\frac{1}{\lambda_{i_\pi}}}{\sum_{j_\pi=1}^{N_\pi} \frac{1}{\lambda_{j_\pi}}}} \quad (3.6)$$

angeben, mit dem sich die laufzeitoptimalen Werte der  $C_{i_\pi}$  berechnen lassen.

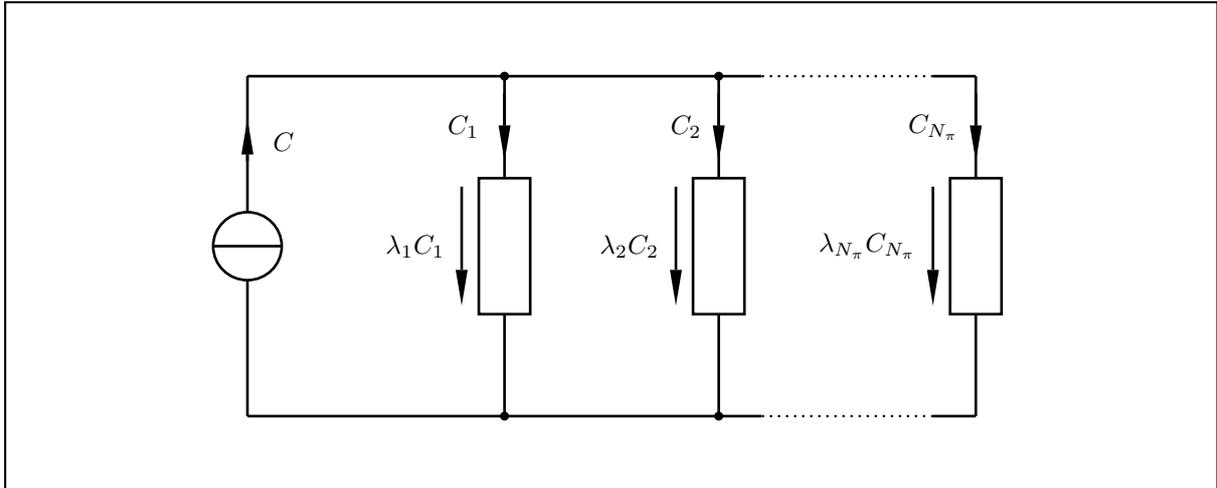
Es besteht eine Analogie zu dem Problem aus der Netzwerktheorie bei bekanntem Gesamtstrom die Ströme durch die Zweige einer Parallelschaltung ohmscher Widerstände zu berechnen. In der Analogie übernehmen die  $C_{i_\pi}$  die Rolle der Ströme und die  $\lambda_{i_\pi}$  die Rolle der ohmschen Widerstände. Die Gleichungen (3.4) und (3.5) sind nichts anderes als die KIRCHHOFFSchen Gesetze und Gleichung (3.6) kann als Stromteilerregel identifiziert werden [63]. Abbildung 3.1 illustriert die beschriebene Analogie.

## 3.2 Das Optimierungsproblem

Wie in Abschnitt 2.4 erwähnt, ist das Ziel jeder Parallelisierung eine Verkürzung der Laufzeit, welche zur Lösung eines Problems benötigt wird. Für eine gegebene Anzahl an Prozessoren  $N_\pi$  des verwendeten Parallelrechners und ein gegebenes Problem  $\mathcal{P}$ , welches durch das parallele System gelöst werden soll, ist also eine Parallelisierungsstrategie  $\mathcal{S}$  gesucht, so dass bei deren Anwendung auf das Problem die Gesamtlaufzeit  $T$  minimal, bzw. der Speedup maximal wird. Das sich ergebende Optimierungsproblem lautet somit

$$\text{minimiere } T_{\mathcal{S}}(N_\pi, \mathcal{P}) \quad \text{über } \mathcal{S}. \quad (3.7)$$

Die Größe  $T_{\mathcal{S}}$  bezeichnet die Gesamtlaufzeit, wenn zur Parallelisierung eine bestimmte Strategie  $\mathcal{S}$  eingesetzt wird. Um die Funktion  $T_{\mathcal{S}}$  explizit ausdrücken zu können, ist eine



**Abbildung 3.1:** Die Abbildung illustriert die beschriebene Analogie aus der Netzwerktheorie für das Problem der Lastzuordnung im Falle eines Parallelrechners mit Prozessoren beliebiger Leistungsfähigkeit.

Präzisierung der Begriffe Problem und Parallelisierungsstrategie erforderlich. Zudem wird ein Modell des Parallelrechners sowie ein Implementierungsmodell benötigt, um zu einem Ausdruck zu gelangen, der einer Analyse zugänglich ist.

Zunächst lässt sich feststellen, dass der PIC Algorithmus aus dem wiederholten Durchlaufen einer Schleife besteht (siehe Abb. 2.4). Ein Schleifendurchlauf ist abhängig von den Ergebnissen des vorherigen Durchlaufes, weshalb die Abarbeitung sequentiell erfolgen muss. Sei  $M$  die Anzahl der insgesamt auszuführenden Zeitschritte (Schleifendurchläufe) und bezeichne  $T_S^{(m)}$  die benötigte Laufzeit für Zeitschritt  $m$ , so lässt sich Gleichung (3.7) präzisieren zu

$$\text{minimiere } T_S(N_\pi, \mathcal{P}) := \sum_{m=1}^M T_S^{(m)}(N_\pi, \mathcal{P}) \quad \text{über } \mathcal{S}. \quad (3.8)$$

Zunächst soll die Zielfunktion für den Fall einer reinen Feldberechnung ohne Anwesenheit von Teilchen, wie in Abschnitt 2.2.3 beschrieben, konkretisiert werden.

### 3.3 Parallelisierung des Feldlösers

Werden nur Simulationen ohne Anwesenheit von Teilchen betrachtet, so ist die Rechenlast lediglich durch die Zeitintegration der Feldfreiheitsgrade, d.h. der Gitterspannungen, gegeben. Zur Parallelisierung des Feldlösers müssen diese Berechnungen auf die verfügbaren Prozessoren verteilt werden. Zunächst soll das Optimierungsproblem für den sehr allgemeinen Fall formuliert werden, bei dem die Zuordnung der Gitterspannungen auf die

Prozessoren zellweise erfolgen kann, d.h. für jede Zelle kann separat eine Prozessorzuordnung angegeben werden. Anschließend wird mit Hilfe der aus der allgemeinen Formulierung gewonnenen Erkenntnisse die Form der Gebiete so festgelegt, dass einerseits die gleichen effizienten Datenstrukturen wie im seriellen Fall verwendet werden können und andererseits die für den Datenaustausch aufgewendete Laufzeit klein gegenüber der Rechenzeit gehalten werden kann.

Im betrachteten Fall strukturierter und nicht adaptiver Gitter kann das Problem  $\mathcal{P}$  mit der Anzahl der Gitterzellen in jeder Raumrichtung vollständig beschrieben werden. Jede Gitterzelle wird durch ihren Index  $(i, j, k)$  referenziert. Die Menge aller Index-Tripel, welche eine Gitterzelle des primären Gitters referenzieren, werde mit  $\mathcal{I} \subset \mathbb{N}^3$  bezeichnet. Für den in dieser Arbeit betrachteten Algorithmus zur Zeitintegration gemäß den Gleichungen (2.36) und (2.37) ergibt sich, dass die mit jedem Freiheitsgrad und somit auch die mit jeder Zelle verknüpfte Rechenlast gleich ist und sich auch im Laufe der Simulation nicht ändert. Die Tatsache, dass einige Freiheitsgrade der Randzellen des Rechengebietes durch die Randbedingungen fixiert sind, bzw. einer abweichenden Behandlung bedürfen, soll hier unberücksichtigt bleiben.

Gleichung (3.8) lässt sich damit vereinfachen zu

$$\boxed{\text{minimiere } T_{\mathcal{S}}(N_{\pi}, \mathcal{P}) = M \cdot T_{\mathcal{S}}^{(m)}(N_{\pi}, \mathcal{P}) \text{ über } \mathcal{S},} \quad (3.9)$$

mit beliebigem  $m \in \{1 \dots M\}$ . Die Verteilung der Berechnungen auf die Prozessoren kann somit ein einziges Mal zu Beginn der Simulation erfolgen. Man spricht hier von einer sogenannten statischen Lastzuordnung [47].

Eine Parallelisierungsstrategie  $\mathcal{S}$  ist nun eine Abbildung von der Menge der Gitterzellen, beschrieben durch ihre Indextripel  $\mathcal{I}$  auf die Menge  $\Pi := \{1 \dots N_{\pi}\}$  der verfügbaren Prozessoren:

$$\mathcal{S} : \mathcal{I} \rightarrow \Pi. \quad (3.10)$$

Diese Zuordnung kann ausgedrückt werden durch Entscheidungsvariablen  $a_{\mathbf{i}_c, i_{\pi}}^{c_{\pi}}$ , für die gilt

$$a_{\mathbf{i}_c, i_{\pi}}^{c_{\pi}} = \begin{cases} 1 & \text{falls Zelle } \mathbf{i}_c \text{ Prozessor } i_{\pi} \text{ zugeordnet wird,} \\ 0 & \text{sonst.} \end{cases} \quad (3.11)$$

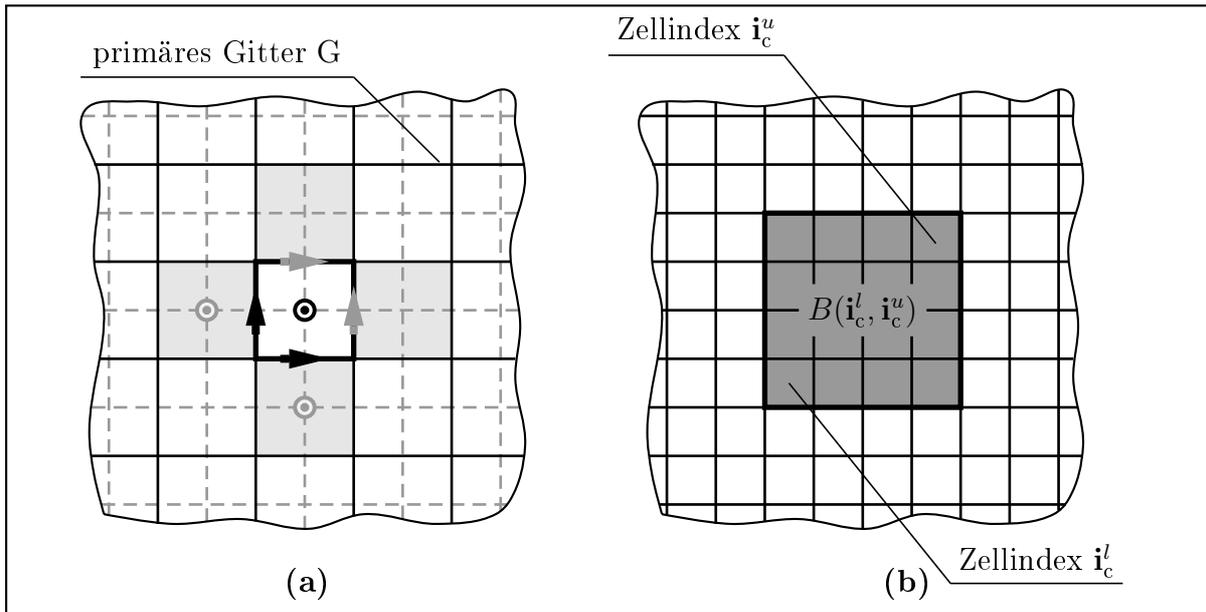
Die Bezeichnung  $\mathbf{i}_c$  steht dabei abkürzend für das Indextripel  $(i, j, k)$  der entsprechenden Gitterzelle. Es sei angenommen, dass die zur numerischen Zeitintegration der einer Zelle zugeordneten Freiheitsgrade benötigte Rechenzeit auf Prozessor  $i_{\pi}$  des parallelen Systems  $\alpha_{i_{\pi}}$  beträgt. Als reine Rechenzeit  ${}^F T_{\mathcal{S}}^{(m)}$  während eines beliebigen Zeitschrittes  $m$  für Prozessor  $i_{\pi}$  ergibt sich dann unter der Annahme einer RAM mit flachem Speicher

$${}^F T_{\mathcal{S}}^{(m)}(i_{\pi}, N_{\pi}, \mathcal{P}) := \alpha_{i_{\pi}} \sum_{\mathbf{i}_c \in \mathcal{I}} a_{\mathbf{i}_c, i_{\pi}}^{c_{\pi}}. \quad (3.12)$$

Die Proportionalitätsfaktoren  $\alpha_{i_\pi}$  modellieren die Leistungsfähigkeit der Prozessoren des Parallelrechners. Im Fall eines Rechners, der mit gleichartigen Prozessoren ausgestattet ist, besitzen alle  $\alpha_{i_\pi}$  den gleichen Wert.

Um eine Prognose für die gesamte Laufzeit pro Zeitschritt aufzustellen, muss zusätzlich der Datenaustausch berücksichtigt werden. Dies erfordert eine Analyse des Speicherzugriffsmusters, welches von dem Algorithmus generiert wird.

Im Falle einer Zeitbereichsrechnung mit FIT entstehen Abhängigkeiten zwischen den Frei-



**Abbildung 3.2:** (a) zeigt die Abhängigkeiten der Freiheitsgrade in einem FIT Gitter in zwei Dimensionen. Zur Zeitintegration der elektrischen und magnetischen Gitterspannungen, welche der dick umrahmten Zelle zugeordnet sind (schwarz eingezeichnet), werden die grau eingezeichneten Gitterspannungen benötigt, die den grau schattierten Nachbarzellen zugeordnet sind. (b) illustriert das Konzept der Bounding Box gemäß Gleichung (3.18).

heitsgraden einer Gitterzelle und den Freiheitsgraden einer anderen Gitterzelle, falls die Zellen eine gemeinsame Fläche (in zwei Dimensionen: eine gemeinsame Kante) besitzen. Die Abhängigkeit wird in Abbildung 3.2(a) für den Fall von zwei Raumdimensionen illustriert. Die schwarz markierten Gitterspannungen (zwei elektrische und eine magnetische) sind der dick umrahmten Zelle zugeordnet. Die Zeitintegration dieser Freiheitsgrade bedarf der grau eingezeichneten Gitterspannungen, die den schattierten Zellen zugeordnet sind. Bezeichne  $Q_{\mathbf{i}_c} \subset \mathcal{I}$  die Menge der Gitterzellen, welche zu einer Zelle  $\mathbf{i}_c$  benachbart sind, d.h. die Berechnungen, welche für die Zelle  $\mathbf{i}_c$  durchzuführen sind, benötigen Daten, welche den in  $Q_{\mathbf{i}_c}$  enthaltenen Zellen zugeordnet sind. Dann lässt sich, unter den in Abschnitt 2.4.5 getroffenen Annahmen betreffend die Modellierung des Datenaustausches und der zusätz-

lichen Einschränkung, dass die benötigte Zeit für den Austausch eines Datenelementes zwischen zwei beliebigen Prozessoren gleich ist, die zum Senden bzw. Empfangen benötigte Laufzeit  ${}^F\tilde{T}_S^{(m)}$  für einen Prozessor  $i_\pi$  zu einem beliebigen Zeitschritt  $m$  abschätzen als

$${}^F\tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) := \xi \sum_{\mathbf{i}_c \in \mathcal{I}} \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} \left( a_{\mathbf{i}_c, j_\pi}^{c\pi} \cdot s \left( \sum_{\mathbf{j}_c \in Q_{\mathbf{i}_c}} a_{\mathbf{j}_c, i_\pi}^{c\pi} \right) \right), \quad (3.13)$$

wobei es sich bei  $\xi$  um einen maschinenabhängigen Proportionalitätsfaktor handelt. Der Wert der Summe entspricht der Anzahl der Gitterzellen, welche zu den Zellen, die Prozessor  $i_\pi$  zugeordnet sind, benachbart sind, jedoch einem anderen Prozessor zugeordnet werden. Bei der Funktion  $s(\cdot)$  handelt es sich um die Einheitssprungfunktion, definiert als

$$s(x) = \begin{cases} 1 & \text{falls } x > 0, \\ 0 & \text{sonst.} \end{cases} \quad (3.14)$$

Eine Abschätzung für die zur Durchführung aller Berechnungen eines Zeitschrittes benötigte Gesamtlaufzeit auf Prozessor  $i_\pi$  ergibt sich aus der Summe der Zeiten  ${}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P})$  und  ${}^F\tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P})$

$$T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) := {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^F\tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}). \quad (3.15)$$

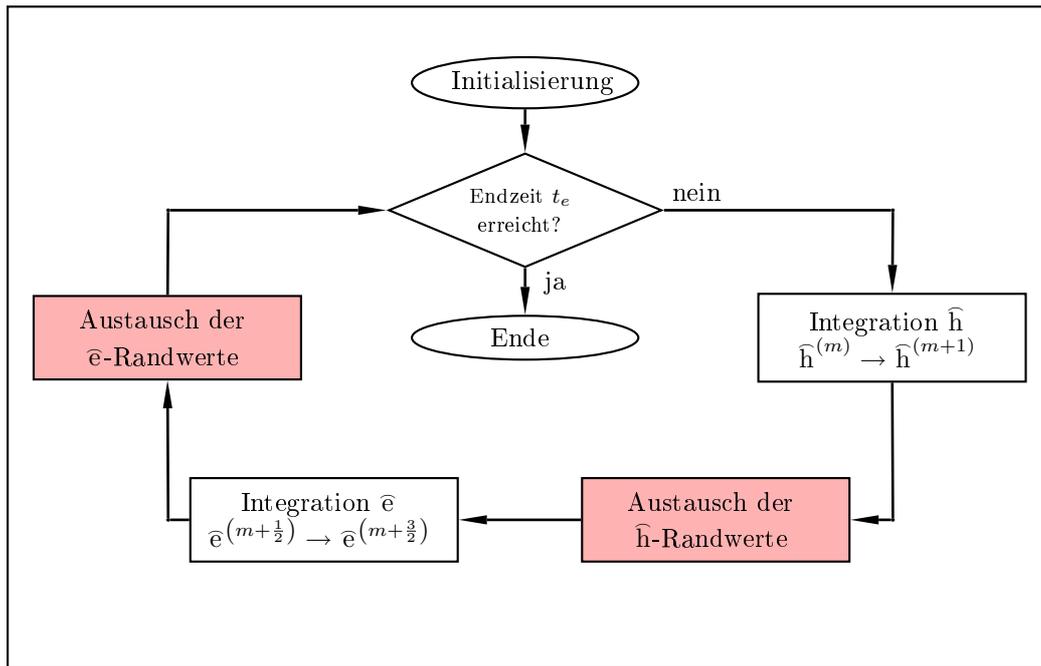
Da die einzelnen Zeitschritte sequentiell abgearbeitet werden müssen, kann ein Zeitschritt erst begonnen werden, wenn alle Berechnungen und Kommunikationsoperationen des vorherigen Schrittes fertig gestellt wurden. Der Prozessor, welcher zur Fertigstellung aller Operationen die meiste Zeit benötigt, wird somit die Laufzeit bestimmen. Es folgt für die Zielfunktion

$$T_S(N_\pi, \mathcal{P}) = M \cdot \max_{i_\pi \in \Pi} \left\{ T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\}. \quad (3.16)$$

Die Zuordnung der Gitterzellen zu den Prozessoren muss eindeutig sein. Dies lässt sich in einer Nebenbedingung an die Entscheidungsvariablen formulieren als

$$\sum_{i_\pi=1}^{N_\pi} a_{\mathbf{i}_c, i_\pi}^{c\pi} = 1 \quad \forall \mathbf{i}_c \in \mathcal{I}. \quad (3.17)$$

Zusammen mit Gleichungen (3.12), (3.13) und (3.15) definiert Gleichung (3.16) die Zielfunktion ausschließlich in Abhängigkeit der gegebenen Parameter  $\mathcal{P}$  und  $N_\pi$  sowie den gesuchten Entscheidungsvariablen  $a_{\mathbf{i}_c, i_\pi}^{c\pi}$ , welche die Zuordnung der Zellen beschreiben. Es handelt sich bei dem formulierten Problem um ein nichtlineares, kombinatorisches Optimierungsproblem [64]. Abbildung 3.3 illustriert das Ablaufdiagramm für die parallelisierte numerische Zeitintegration der Gitterspannungen.



**Abbildung 3.3:** Ablaufdiagramm für den parallelisierten Feldlöser. Die schattierten Kästen beinhalten Kommunikationsoperationen, welche eine Synchronisation der Prozesse erzwingen.

Mehrere Aspekte lassen die Suche nach einem Algorithmus, welcher zu einer optimalen Lösung des in Gleichung (3.16) und (3.17) formulierten Optimierungsproblems führt, für praktische Zwecke unattraktiv erscheinen.

Zum Ersten sind kombinatorische Optimierungsprobleme prinzipiell deutlich schwieriger zu lösen als Optimierungsprobleme mit kontinuierlichen Variablen. Viele kombinatorische Optimierungsprobleme gehören zur Klasse der NP-vollständigen Probleme [65, 64], so dass zu ihrer Lösung nach dem gegenwärtigen Kenntnisstand kein effizienter Algorithmus konstruiert werden kann, der eine optimale Lösung liefert. Unter einem effizienten Algorithmus wird hier ein solcher verstanden, der eine polynomiale Komplexität besitzt.

Zum Zweiten ist das in Gleichung (3.16) und (3.17) gestellte Optimierungsproblem aus einem stark vereinfachten Parallelrechnermodell hervorgegangen, welches nur eine Approximation für die tatsächlich benötigte Laufzeit liefern kann. Die praktische Notwendigkeit eines Algorithmus, der, womöglich unter großem Rechenaufwand, eine optimale Lösung findet, ist somit fraglich.

Aufgrund dieser Überlegungen soll das Augenmerk nicht auf die exakte Lösung des Optimierungsproblems gelegt werden, sondern vielmehr auf die Entwicklung effizienter Heuristiken, die eine für praktische Zwecke hinreichend gute Lösung liefern. Dies ist auch das Vorgehen, welches bei den in der Literatur vorgeschlagenen Ansätzen zur Lösung dieses Problems durchweg verfolgt wird. Für eine Übersicht über die wichtigsten Heuristiken,

welche zur Lösung des Optimierungsproblems in der Literatur zu finden sind, sei auf Anhang C verwiesen.

Eine Betrachtung der Bestandteile (3.12) und (3.13) der Zielfunktion gibt einige Hinweise darauf, welche Eigenschaften eine Parallelisierungsstrategie besitzen muss, um die Zielfunktion tendenziell zu minimieren.

Die Datenmenge, welche in jedem Zeitschritt ausgetauscht werden muss, kann offenbar dadurch reduziert werden, dass den Prozessoren zusammenhängende Bereiche von Gitterzellen zugeordnet werden, was an dem Ausdruck für die Kommunikationszeit in Gleichung (3.13) abgelesen werden kann. Für Parallelisierungsstrategien, welche die Zellen den verfügbaren Prozessoren in zusammenhängenden Partitionen zuordnen, wird ein Datenaustausch nur für die Zellen am Rand der Partition benötigt, so dass die für den Datenaustausch benötigte Zeit  ${}^F\tilde{T}_S^{(m)}$  häufig klein gegenüber der Rechenzeit  ${}^F T_S^{(m)}$  gehalten werden kann. Die Ergebnisse in Kapitel 4 bestätigen diese Überlegung a posteriori.

Um für den parallelisierten Feldlöser und später für die Kopplung mit den Makroteilchen ähnlich effiziente Datenstrukturen wie im seriellen Fall verwenden und gleichzeitig, wie oben als sinnvoll erkannt, den einzelnen Prozessoren zusammenhängende Bereiche zuzuordnen zu können, bietet sich eine Partitionierung an, bei der die einzelnen Partitionen quaderförmige Bereiche des Gitters umfassen, oder aus solchen quaderförmigen Bereichen zusammengesetzt sind. Um solche Partitionen zu beschreiben, wird das Konzept der Bounding Box  $B(\mathbf{i}_c^l, \mathbf{i}_c^u) \subseteq \mathcal{I}$  eingeführt. Diese sei definiert als

$$B(\mathbf{i}_c^l, \mathbf{i}_c^u) := \{\mathbf{i}_c \in \mathcal{I} : i^l \leq i \leq i^u \wedge j^l \leq j \leq j^u \wedge k^l \leq k \leq k^u\}. \quad (3.18)$$

Abbildung 3.2(b) veranschaulicht die Beschreibung eines Gitterausschnittes mit Hilfe der in Gleichung (3.18) eingeführten Schreibweise.

Im folgenden Abschnitt soll eine flexible und effiziente Parallelisierungsstrategie vorgestellt werden, welche unter Berücksichtigung der oben angestellten Überlegungen das Optimierungsproblem heuristisch löst.

### 3.3.1 Rekursive Koordinaten Bisektionierung

Die Methode der Rekursiven Koordinaten Bisektionierung (engl.: **R**ecursive **C**oordinate **B**isection (RCB)) wurde von BERGER und BOKHARI ursprünglich für die Partitionierung von strukturierten Gittern mit lokaler Gitterverfeinerung entwickelt [66]. Der Algorithmus wurde unter der Annahme eines Parallelrechners mit einer Anzahl  $N_\pi = 2^n$ ,  $n \in \mathbb{N}$  gleichartiger Prozessoren formuliert. Er erzeugt eine Partitionierung des Gitters in  $N_\pi$  disjunkte Teile durch eine wiederholte Bisektionierung, wobei jede Bisektionierung durch eine Ebene erfolgt, deren Normale in Richtung einer der Koordinatenachsen zeigt. Durch diese

Beschränkung sind die sich ergebenden Partitionen durch die in Gleichung (3.18) eingeführte Notation beschreibbar.  $B_{i_\pi}$  bezeichne im Folgenden den Gitterausschnitt, welcher Prozessor  $i_\pi$  zugeordnet wird. Der Partitionierungsalgorithmus soll hier in einer weiterentwickelten Form erläutert werden, die sowohl die Behandlung einer beliebigen Anzahl an Prozessoren, wie auch die Berücksichtigung von Prozessoren unterschiedlicher Leistungsfähigkeit erlaubt [67, 68].

Durch die Beschränkung bezüglich der Form der Partitionen kann der Kommunikationsaufwand  ${}^F\tilde{T}_S^{(m)}$  lediglich durch die Wahl der Normale der Partitionierungsebene maßgeblich beeinflusst werden. Die Normale der Partitionierungsebene wird in diejenige Koordinatenrichtung gewählt, in der die Anzahl der durch die Ebene eingeführten Randzellen minimal ist. Die Größe der Partitionen wird so gewählt, dass die Laufzeit  ${}^F T_S^{(m)}$  für den Feldlöser für alle Prozessoren gleich ist:

$${}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = {}^F T_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \quad \forall i_\pi, j_\pi \in \Pi. \quad (3.19)$$

Mit Hilfe des Ausdrucks in Gleichung (3.12) lässt sich die benötigte Rechenzeit für jeden der Prozessoren in Abhängigkeit von der Anzahl der zugeordneten Gitterzellen angeben zu

$${}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = \alpha_{i_\pi} \cdot |B_{i_\pi}|, \quad (3.20)$$

wobei  $|B_{i_\pi}|$  die Anzahl der Elemente der Menge  $B_{i_\pi}$  beschreibt. Es soll zunächst der Fall  $N_\pi = 2$  betrachtet werden. Die beiden gesuchten Partitionen  $B_1$  und  $B_2$  müssen für eine optimale Lastbalancierung so gewählt werden, dass gilt

$$\alpha_1 \cdot |B_1| = \alpha_2 \cdot |B_2|. \quad (3.21)$$

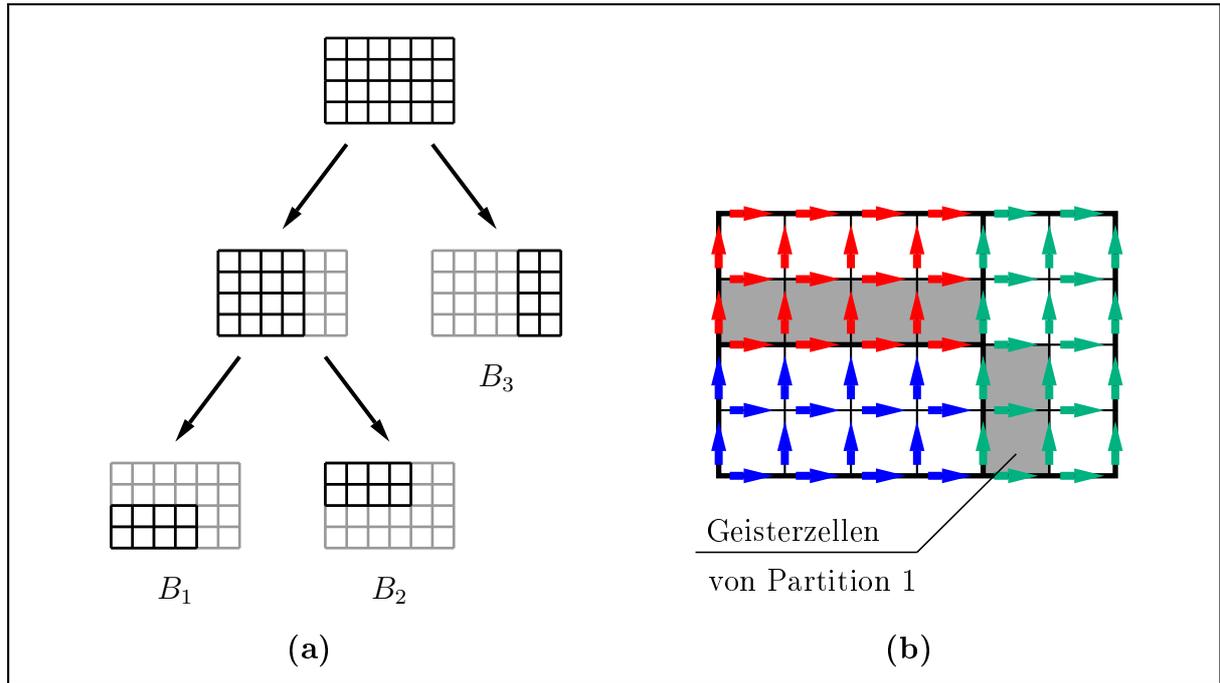
Aufgrund der begrenzten Flexibilität der Partitionierung, bedingt durch die vorgegebene Form der Partitionen, kann Gleichung (3.21) im Allgemeinen nicht exakt erfüllt werden. Die Partitionierungsebene wird dann so gewählt, dass Gleichung (3.21) bestmöglich approximiert wird.

Nun soll das Problem für eine beliebige Anzahl von Prozessoren gelöst werden, wobei von den in Abschnitt 3.1 hergeleiteten Ausdrücken Gebrauch gemacht wird. Zur rekursiven Berechnung der  $|B_{i_\pi}|$  werden nun im ersten Schritt die Prozessoren in zwei disjunkte Mengen  $\Pi_a$  und  $\Pi_b$  aufgeteilt, so dass

$$|\Pi_a| = \left\lceil \frac{N_\pi}{2} \right\rceil \quad \text{und} \quad |\Pi_b| = \left\lfloor \frac{N_\pi}{2} \right\rfloor \quad (3.22)$$

gilt, wobei  $\lceil \cdot \rceil$  und  $\lfloor \cdot \rfloor$  das Aufrunden beziehungsweise Abrunden auf die nächste Ganzzahl bezeichnet gemäß

$$\lceil x \rceil := \min_{k \in \mathbb{Z}, k \geq x} \{k\} \quad \text{und} \quad \lfloor x \rfloor := \max_{k \in \mathbb{Z}, k \leq x} \{k\}. \quad (3.23)$$



**Abbildung 3.4:** (a) zeigt die Partitionierung eines zweidimensionalen Beispieltgitters mit Hilfe der Methode der RCB für  $N_\pi = 3$  im Falle gleichartiger Prozessoren ( $\alpha_{i_\pi} = \alpha \forall i_\pi \in \Pi$ ). Die sich ergebenden Partitionen in den einzelnen Schritten sind fett markiert. (b) zeigt die Zuordnung der elektrischen Gitterspannungen zu den Prozessoren für die Beispielpartitionierung, sowie die Geisterzellen der Partition 1.

Diesen zwei Mengen von Prozessoren kann jeweils ein Proportionalitätsfaktor zugeordnet werden gemäß

$$\alpha_a := \frac{1}{\sum_{i_\pi \in \Pi_a} \frac{1}{\alpha_{i_\pi}}} \quad \text{bzw.} \quad \alpha_b := \frac{1}{\sum_{i_\pi \in \Pi_b} \frac{1}{\alpha_{i_\pi}}}. \quad (3.24)$$

Dies sind die bekannten Formeln zur Berechnung des Gesamtwiderstandes parallelgeschalteter ohmscher Widerstände [63]. Sie folgen sofort aus der in Abschnitt 3.1 gezeigten Analogie. Gemäß der Lastbalancierungsbedingung für den Zweiprocessorfall (3.21) können nun die zwei Gebietsgrößen  $|B_a|$  und  $|B_b|$  für die beiden Prozessorguppen berechnet werden gemäß

$$\boxed{\frac{|B_a|}{|B_b|} = \frac{\alpha_b}{\alpha_a}}. \quad (3.25)$$

In den nächsten Schritten werden die Mengen  $\Pi_a$  und  $\Pi_b$  wieder nach dem obigen Schema partitioniert. Dies wird so lange fortgesetzt, bis jede der Teilmengen nur noch aus einem einzigen Prozessor besteht. Die Entscheidungsvariablen des ursprünglichen Optimierungs-

problems werden durch das Verfahren somit festgelegt zu

$$a_{\mathbf{i}_c, i_\pi}^{c_\pi} = \begin{cases} 1 & \text{falls } \mathbf{i}_c \in B_{i_\pi}, \\ 0 & \text{sonst.} \end{cases} \quad (3.26)$$

Die rekursive Partitionierung des Gitters lässt sich mit Hilfe eines Binärbaumes veranschaulichen. Die Blätter des Baumes zeigen dabei die endgültigen Partitionen, während die anderen Knoten die Zwischenschritte zeigen. Die sich ergebende Partitionierung für ein Beispielgitter im Falle  $N_\pi = 3$  gleichartiger Prozessoren zeigt Abbildung 3.4(a). Die sich daraus ergebende Zuordnung der elektrischen Gitterspannungen zu den Prozessoren ist in Abbildung 3.4(b) dargestellt. Zellen, deren Felddaten benötigt werden, die jedoch einem anderen Prozessor zugeordnet sind, werden als Geisterzellen (engl. Ghostcells) bezeichnet. Dies sind alle Randzellen einer Partition, was sich mit Hilfe der in Abbildung 3.2(a) illustrierten Abhängigkeit zwischen den Gitterzellen ableiten lässt. Die Datenmenge, welche in jedem Zeitschritt zwischen den Prozessoren ausgetauscht werden muss, ist direkt proportional zur Anzahl der Geisterzellen. In Abbildung 3.4(b) sind zur Veranschaulichung die Geisterzellen der Partition links unten schattiert eingezeichnet.

Die Einschränkung bezüglich der Form der Partitionen kann dazu führen, dass die Partitionierungsbedingung (3.25) nur näherungsweise erfüllt werden kann und damit eine suboptimale Verteilung der Gitterzellen auf die Partitionen erfolgt. Eine Verallgemeinerung des Verfahrens, welche die Randzellen einer Partition noch einmal gesondert zuordnet, findet sich z.B. in [69]. Dies ist jedoch eher für kompliziertere Gitter (z.B. strukturierte Gitter mit lokaler Gitterverfeinerung) von Interesse, die ohnehin kompliziertere Datenstrukturen erfordern. Im Rahmen der Arbeit kommt das oben beschriebene Verfahren zum Einsatz.

### 3.3.2 Skalierbarkeitsanalyse für den parallelisierten Feldlöser

Wie in Abschnitt 2.4.3.4 erläutert, ist die Skalierbarkeit eines parallelen Systems ein wichtiges Qualitätsmerkmal. Nur bei einem skalierbaren System ist es möglich, eine Kombination von Prozessorenzahl und Problemgröße zu finden, so dass das parallele System das Problem mit einer gewünschten Effizienz löst. Im Falle des Feldlösers ist die Gesamtzahl der Gitterzellen  $N_c$  ein sinnvolles Maß für die Problemgröße. Da der Rechenaufwand sowie der Kommunikationsaufwand bei einer gegebenen Partitionierung a priori bestimmt werden kann, ist es unter einigen vereinfachenden Annahmen möglich, eine theoretische Skalierbarkeitsanalyse für die RCB Partitionierungs- bzw. Parallelisierungsstrategie durchzuführen. Die Skalierbarkeitsanalyse erfolgt, wie in Abschnitt 2.4.3.4 erläutert, für den Fall eines parallelen Systems mit gleichartigen Prozessoren. Im Folgenden wird daher  $\alpha_{i_\pi} = \alpha \forall i_\pi \in \Pi$  gesetzt.

Die Analyse erfordert das Aufstellen der Isoeffizienzfunktion. Die Effizienz lässt sich zunächst angeben als

$$E(N_\pi, N_c) = \frac{T_S(1, N_c)}{N_\pi \cdot T_S(N_\pi, N_c)}. \quad (3.27)$$

Gemäß Gleichung (3.12) kann die serielle Laufzeit direkt proportional zu der Anzahl der Gitterzellen angenommen werden und es folgt

$${}^F T_S(1, N_c) = \alpha M N_c. \quad (3.28)$$

Wie in Abschnitt 3.3.1 gezeigt, führt die RCB Strategie für ein strukturiertes kartesisches Gitter zu zusammenhängenden, quaderförmigen Partitionen. Unter der Annahme, dass die Gitterzellen durch die RCB Heuristik gleichmäßig auf die verfügbaren Prozessoren verteilt werden, ergibt sich für alle Prozessoren die gleiche Rechenzeit pro Zeitschritt. Die benötigte Zeit für den Datenaustausch kann gemäß Gleichung (3.13) proportional zu der Anzahl der Gitterzellen angenommen werden, für die in jedem Zeitschritt Daten ausgetauscht werden müssen. Diese Anzahl ist wiederum proportional zu der Anzahl der Zellflächen, welche die Partitionsgrenze bilden. Falls die Partitionen so beschaffen sind, dass sie die gleiche Anzahl an Gitterzellen in jeder Raumrichtung besitzen, so bilden  $6 \cdot (N_c/N_\pi)^{2/3}$  Zellflächen diese Grenze. Ansonsten bildet der Ausdruck eine gute Approximation. Da der Zeitaufwand proportional zu der Anzahl auszutauschender Freiheitsgrade angenommen wird, folgt schließlich als Abschätzung für die Gesamtlaufzeit auf  $N_\pi$  Prozessoren

$$T_S(N_\pi, N_c) = \alpha M \frac{N_c}{N_\pi} + \zeta M \left( \frac{N_c}{N_\pi} \right)^{\frac{2}{3}}, \quad (3.29)$$

wobei  $\zeta$  wiederum ein Proportionalitätsfaktor ist, und der  $\max\{.\}$  Ausdruck aus Gleichung (3.16) fortgelassen wurde, da die Rechen- sowie die Kommunikationszeit als für alle Prozessoren gleich angenommen wurde. Es folgt nun mit den Gleichungen (3.28) und (3.29) für die Effizienz

$$E(N_\pi, N_c) = \frac{1}{1 + \frac{\zeta}{\alpha} \left( \frac{N_\pi}{N_c} \right)^{\frac{1}{3}}}. \quad (3.30)$$

Aus Gleichung (3.30) folgt, dass die Effizienz konstant bleibt, falls die Problemgröße proportional zur Anzahl der Gitterzellen erhöht wird. Der parallelisierte Feldlöser ist somit gemäß den Ausführungen in Abschnitt 2.4.3.4 optimal skalierbar.

In [70] wird dieses Ergebnis mit Hilfe graphentheoretischer Methoden allgemein für eine große Klasse von Parallelisierungsstrategien, welche auf dem Bisektionierungsansatz basieren, bewiesen.

### 3.3.3 Verstecken der Latenzzeit

Wie in Abschnitt 2.4.5 erwähnt, ist es mit Hilfe von spezialisierter Hardware oder dem expliziten Einsatz mehrerer Threads bei Multiprozessoren möglich, die für den Datenaustausch benötigte Zeit mit Berechnungen zu füllen, die der ausgetauschten Daten nicht bedürfen.

Da lediglich zur Zeitintegration der Feldfreiheitsgrade der Randzellen nichtlokale Daten benötigt werden, kann der Datenaustausch wie folgt stattfinden. Zunächst werden die magnetischen Gitterspannungen nur für die Randzellen aktualisiert. Für diese aktualisierten Gitterspannungen kann nun der Datenaustausch initialisiert werden. Während dieser durchgeführt wird, können die magnetischen Gitterspannungen für die übrigen Gitterzellen aktualisiert werden. In der gleichen Weise kann mit den elektrischen Gitterspannungen verfahren werden [11].

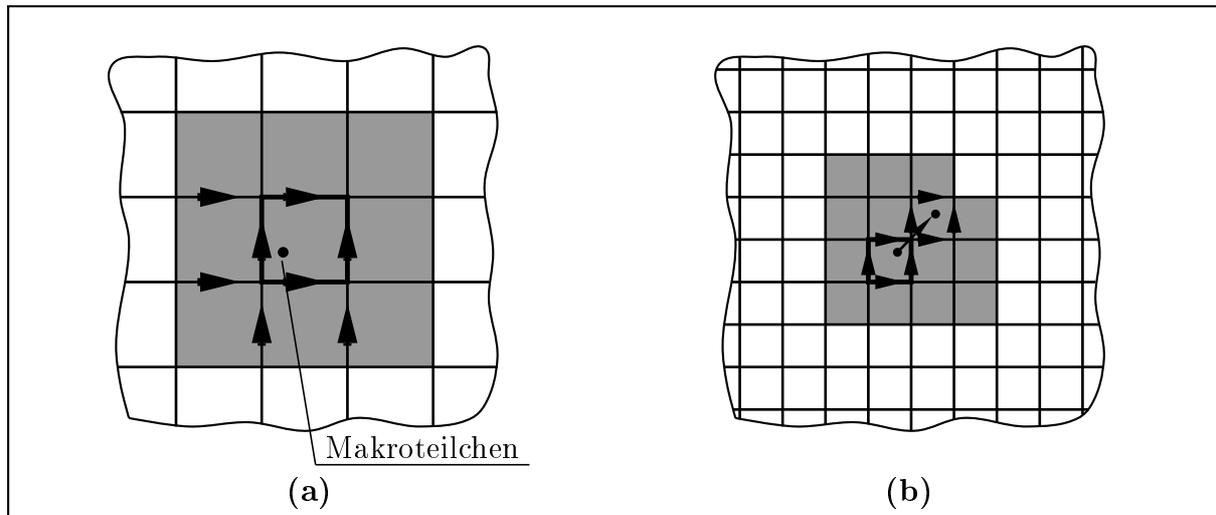
## 3.4 Parallelisierung des PIC Algorithmus

Das Optimierungsproblem soll nun um die durch die Makroteilchen verursachte Rechenlast und den zusätzlich notwendigen Datenaustausch erweitert werden. Dazu ist eine Analyse des Speicherzugriffsmusters erforderlich, welches eine PIC Simulation erzeugt.

### 3.4.1 Erweiterung des Optimierungsproblems

Betrachtet man den vollständigen PIC Algorithmus gemäß Abschnitt 2.3, so lässt sich feststellen, dass sich die zu verteilende Rechenlast aus der Integration der Feldfreiheitsgrade einerseits, und der Integration der Trajektorien der Makroteilchen andererseits zusammensetzt (siehe Abbildung 2.4). Diese beiden Phasen des Algorithmus sind jeweils wechselseitig auf die Ergebnisse der vorangehenden Phase angewiesen und erfolgen daher sequentiell. Die Kopplung der beiden Phasen erfolgt durch die Berechnung des elektromagnetischen Feldes am Ort des Teilchens einerseits, und die Berechnung der Ströme  $\hat{\mathbf{j}}$ , die durch die Bewegungen der Makroteilchen entstehen, andererseits.

Der Wert des elektromagnetischen Feldes am Ort des Teilchens muss durch Interpolation aus den Gitterspannungen bestimmt werden. Welche Gitterspannungen für die Interpolation benötigt werden, hängt zum einen von der Position der Teilchen im Gitter und zum anderen von dem verwendeten Interpolationsverfahren ab. Für die im Rahmen der Arbeit durchgeführten Simulationen kam eine trilineare Interpolation zum Einsatz, welche bei drei Raumdimensionen für jede Feldkomponente die acht (im Falle zweier Raumdimensio-



**Abbildung 3.5:** (a) zeigt die Gitterspannungen welcher Zellen im Allgemeinen für die Berechnung des elektromagnetischen Feldes am Ort eines Teilchens benötigt werden können, wenn eine trilineare Interpolation verwendet wird. Für die eingezeichnete Teilchenposition werden die elektrischen Gitterspannungen benötigt, welche in die Skizze eingetragen sind. (b) zeigt die Ströme welcher Gitterzellen im Allgemeinen von einem Teilchen beeinflusst werden können, welches sich zu Beginn des Zeitschrittes in der umrahmten Zelle befindet, wenn das von BUNEMAN beschriebene Verfahren zur Stromberechnung zum Einsatz kommt. Die eingetragene Teilchenbewegung beeinflusst die eingezeichneten Ströme.

nen die vier) am nächsten zu der Teilchenposition allokierten Gitterspannungen benötigt. Abbildung 3.5(a) zeigt für den zweidimensionalen Fall die an der Interpolation beteiligten elektrischen Gitterspannungen für das eingezeichnete Makroteilchen. Die an der Interpolation beteiligten magnetischen Gitterspannungen ergeben sich entsprechend.

Welche Ströme  $\hat{j}$  durch die Bewegung des Makroteilchens während eines Zeitschrittes beeinflusst werden, hängt einerseits von der Bewegung und der Position des Teilchens im Gitter und andererseits von dem für die Stromberechnung verwendeten Algorithmus ab. Bei dem im Rahmen der Arbeit implementierten Algorithmus nach BUNEMAN [31] kann ein Teilchen während eines Zeitschrittes die Stromwerte beeinflussen, welche der Gitterzelle zugeordnet sind, in der sich das Teilchen zu Beginn des Zeitschrittes befindet, sowie höchstens noch diejenigen Werte, die den Zellen zugeordnet sind, die mit dieser Gitterzelle über eine „Zwischenzelle“ verbunden sind. Dies erklärt sich daraus, dass sich ein Teilchen aufgrund der Limitierung der Zeitschrittweite  $\Delta t$ , welche sich durch die Stabilitätsforderung für den Feldlöser ergibt, in einem Zeitschritt maximal von einer Gitterzelle in eine zu dieser benachbarte Gitterzelle bewegen kann. Diese Aussage bleibt selbst für ein Teilchen, welches sich mit der physikalisch maximal möglichen Geschwindigkeit  $v = c$  bewegt, gültig.

Abbildung 3.5(b) zeigt exemplarisch die von einer Teilchenbewegung beeinflussten Stromwerte. Die Stromwerte, welche den farbig hinterlegten Zellen zugeordnet sind, können im Allgemeinen durch eine beliebige Bewegung des eingezeichneten Teilchens beeinflusst werden.

Auf welche der Gitterspannungen und Gitterströme bei der Berechnung der Trajektorien und Ströme während eines Zeitschrittes zugegriffen werden muss, hängt also von der Verteilung der Teilchen im Rechengebiet ab. Um dies beschreiben zu können, werden Variablen  $\overset{\text{cp}}{n}_{\mathbf{i}_c,p}^{(m)}$  mit folgender Bedeutung eingeführt:

$$\overset{\text{cp}}{n}_{\mathbf{i}_c,p}^{(m)} = \begin{cases} 1 & \text{falls sich Teilchen } p \text{ in Zeitschritt } m \text{ in Zelle } \mathbf{i}_c \text{ befindet,} \\ 0 & \text{sonst.} \end{cases} \quad (3.31)$$

Zur Vereinfachung des Laufzeitmodells soll angenommen werden, dass die zur Berechnung des elektromagnetischen Feldes am Orte des Teilchens benötigten Gitterspannungen von einem fremden Prozessor über das Netzwerk verschickt werden müssen, wenn die Zelle, in der sich das Teilchen befindet, diesem Prozessor zugeordnet ist. Ebenso wird angenommen, dass die berechneten Stromwerte anschließend an diesen Prozessor versendet werden müssen, da er diese im nächsten Zeitschritt zur Integration der Felder benötigt. Diese Annahme vernachlässigt den Fall, dass nicht alle Felddaten, auf welche zur Behandlung eines einzigen Teilchens zugegriffen werden muss, dem gleichen Prozessor zugeordnet sein müssen. Unter der Annahme, dass die Gitterzellen in weitgehend zusammenhängenden Bereichen auf die Prozessoren verteilt werden, wie es in Abschnitt 3.3 als sinnvoll erkannt wurde, tritt dieser Fall jedoch nur dann auf, wenn sich ein Teilchen in einer Gitterzelle nahe einer Partitions Grenze befindet. Da dies in der Regel nur für einen vergleichsweise geringen Teil aller vorhandenen Teilchen auftritt, erscheint diese Vereinfachung gerechtfertigt.

Das Problem  $\mathcal{P}$  wird nun, neben der Anzahl der Gitterzellen in jeder Raumrichtung, definiert durch die Positionen der Teilchen im Gitter  $\overset{\text{cp}}{n}_{\mathbf{i}_c,p}^{(m)}$  zu jedem Zeitschritt  $m$ . Eine Parallelisierungsstrategie  $\mathcal{S}$  ist ein Algorithmus, welcher eine eindeutige Zuordnung der Gitterzellen und Teilchen zu den  $N_\pi$  verfügbaren Prozessoren für alle Zeitschritte  $m \in \{1 \dots M\}$  liefert. Das Optimierungsziel ist wiederum, die Laufzeit  $T_{\mathcal{S}}$  zu minimieren.

Das Laufzeitmodell aus Abschnitt 3.2, welches in Abschnitt 3.3 für den Feldlöser konkretisiert wurde, muss aufgrund der obigen Ausführungen wie folgt erweitert werden. Im Rahmen der Phase des Algorithmus, in der die Trajektorien der Makroteilchen integriert werden, tritt für jedes Teilchen die gleiche Rechenlast auf. Für alle Teilchen, die einem anderen Prozessor zugeordnet werden als die Gitterzelle, in der sie sich befinden, müssen die benötigten Felddaten nach dem Abschluss der Feldlöserphase mit diesem Prozessor ausgetauscht werden. Umgekehrt müssen nach der Berechnung der Teilchenbewegung die sich ergebenden Ströme diesem Prozessor zur Verfügung gestellt werden. Gemäß der oben

diskutierten Vereinfachung soll angenommen werden, dass die verschickte Datenmenge pro Teilchen für alle Teilchen, die sich in einer solchen nichtlokalen Zelle befinden, gleich ist und von dem Prozessor bereitgestellt werden kann, welcher die Felddaten für die Zelle besitzt.

Die Zuordnung der Teilchen zu den Prozessoren kann frei gewählt werden. Sie möge, wie bereits die Zuordnung der Gitterzellen, durch Entscheidungsvariablen  $a_{p,i_\pi}^{p\pi(m)}$  beschrieben werden. Es gelte

$$a_{p,i_\pi}^{p\pi(m)} = \begin{cases} 1 & \text{falls Teilchen } p \text{ in Zeitschritt } m \text{ Prozessor } i_\pi \text{ zugeordnet ist,} \\ 0 & \text{sonst.} \end{cases} \quad (3.32)$$

Da auch hier eine eindeutige Zuordnung der Teilchen zu den Prozessoren erzwungen werden soll, folgt die Nebenbedingung

$$\sum_{i_\pi=1}^{N_\pi} a_{p,i_\pi}^{p\pi(m)} = 1 \quad \forall p \in \{1 \dots P^{(m)}\}, m \in \{1 \dots M\}. \quad (3.33)$$

Die benötigte Laufzeit für die Berechnung der Trajektorien auf einem Prozessor  $i_\pi$  setzt sich zusammen aus der Rechenzeit, welche als proportional zu der Anzahl der Teilchen angenommen werden kann, die dem Prozessor zugeordnet sind, sowie der für den Datenaustausch benötigten Laufzeit. Die für die Trajektorienberechnung während des Zeitschrittes  $m$  von Prozessor  $i_\pi$  benötigte Rechenzeit ergibt sich zu

$$P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) := \beta_{i_\pi} \sum_{p=1}^{P^{(m)}} a_{p,i_\pi}^{p\pi(m)}. \quad (3.34)$$

Die  $\beta_{i_\pi}$  sind wiederum maschinenabhängige Proportionalitätsfaktoren. Als Abschätzung für die für den Datenaustausch benötigte Laufzeit während des Zeitschrittes  $m$  für Prozessor  $i_\pi$  ergibt sich mit dem Proportionalitätsfaktor  $\gamma$

$$\begin{aligned} P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) := & \gamma \sum_{\substack{j_\pi=1 \\ j_\pi \neq i_\pi}}^{N_\pi} \sum_{\mathbf{i}_c \in \mathcal{I}} \sum_{p=1}^{P^{(m)}} a_{p,i_\pi}^{p\pi(m)} \cdot a_{\mathbf{i}_c,j_\pi}^{c\pi(m)} \cdot n_{\mathbf{i}_c,p}^{c\pi(m)} \\ & + \gamma \sum_{\substack{j_\pi=1 \\ j_\pi \neq i_\pi}}^{N_\pi} \sum_{\mathbf{i}_c \in \mathcal{I}} \sum_{p=1}^{P^{(m)}} a_{p,j_\pi}^{p\pi(m)} \cdot a_{\mathbf{i}_c,i_\pi}^{c\pi(m)} \cdot n_{\mathbf{i}_c,p}^{c\pi(m)}, \end{aligned} \quad (3.35)$$

wobei der erste Term proportional zur Anzahl der Teilchen ist, welche Prozessor  $i_\pi$  zugeordnet sind, jedoch Felddaten von einem der anderen Prozessoren benötigen. Der zweite Term ist proportional zur Anzahl der Teilchen, welche nicht Prozessor  $i_\pi$  zugeordnet sind,

jedoch von Prozessor  $i_\pi$  Felddaten benötigen. Bei Ausdruck (3.35) wurde, wie schon beim Datenaustausch für den Feldlöser, angenommen, dass die Daten kollektiv in Nachrichten großer Länge statt einzeln verschickt werden, so dass die dafür benötigte Zeit als direkt proportional zur verschickten Datenmenge angenommen werden kann.

Sowohl die Zuordnung von Teilchen, als auch die Zuordnung von Gitterzellen und den damit verbundenen Gitterspannungen zu den Prozessoren, können durch die Parallelisierungsstrategie während der Simulation verändert werden, wie der hochgestellte Index an den Entscheidungsvariablen  $\overset{p\pi}{a}_{p,i_\pi}^{(m)}$  bzw.  $\overset{c\pi}{a}_{i_c,i_\pi}^{(m)}$  andeutet. Eine solche Veränderung der Zuordnung erfordert den Austausch der mit der Gitterzelle bzw. mit dem Teilchen verbundenen Daten. Die dafür aufzuwendende Laufzeit werde mit  ${}^{LB}\tilde{T}_S^{(m)}$  bezeichnet. Unter der Annahme, dass die benötigte Zeit für den Datenaustausch für jeden Prozessor proportional zur Anzahl der Gitterzellen bzw. Teilchen ist, die dem Prozessor neu zugeordnet werden, beziehungsweise, die er an einen anderen Prozessor abgibt, ergibt sich für einen Prozessor  $i_\pi$  in Zeitschritt  $m$ :

$$\begin{aligned} {}^{LB}\tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) := & \nu \sum_{i_c \in \mathcal{I}} \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} \left( \overset{c\pi}{a}_{i_c,i_\pi}^{(m)} \cdot \overset{c\pi}{a}_{i_c,j_\pi}^{(m+1)} + \overset{c\pi}{a}_{i_c,j_\pi}^{(m)} \cdot \overset{c\pi}{a}_{i_c,i_\pi}^{(m+1)} \right) \\ & + \eta \sum_{p=1}^{P^{(m)}} \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} \left( \overset{p\pi}{a}_{p,i_\pi}^{(m)} \cdot \overset{p\pi}{a}_{p,j_\pi}^{(m+1)} + \overset{p\pi}{a}_{p,j_\pi}^{(m)} \cdot \overset{p\pi}{a}_{p,i_\pi}^{(m+1)} \right), \end{aligned} \quad (3.36)$$

wobei der erste Term proportional zur Anzahl der Gitterzellen ist, die Prozessor  $i_\pi$  neu zugeordnet oder von Prozessor  $i_\pi$  an andere Prozessoren abgegeben werden. Der zweite Term ist proportional zur Anzahl der Teilchen, welche Prozessor  $i_\pi$  neu zugeordnet werden, oder von Prozessor  $i_\pi$  an andere Prozessoren abgegeben werden. Die Größen  $\nu$  und  $\eta$  sind wiederum entsprechende Proportionalitätsfaktoren.

Auf welche Weise die einzelnen Ausdrücke  ${}^F T_S^{(m)}$ ,  ${}^F \tilde{T}_S^{(m)}$ ,  ${}^P T_S^{(m)}$ ,  ${}^P \tilde{T}_S^{(m)}$  und  ${}^{LB}\tilde{T}_S^{(m)}$  zu einer Gesamtlaufzeit kombiniert werden können, hängt von der Implementierung ab. Insbesondere davon, zu welchem Zeitpunkt der Datenaustausch vorgenommen wird, welcher zu einer Synchronisation der Prozesse führt.

Da die beiden Phasen des PIC Algorithmus durch die Kopplung voneinander abhängig sind, liegt die Annahme einer Synchronisation nach jeder Phase nahe. Abbildung 3.6(a) zeigt schematisch das zugehörige Ablaufdiagramm. Die Neuordnung der Feld- und Teilchendaten geschieht nach Abschluss der Berechnungen eines Zeitschrittes in einer separa-

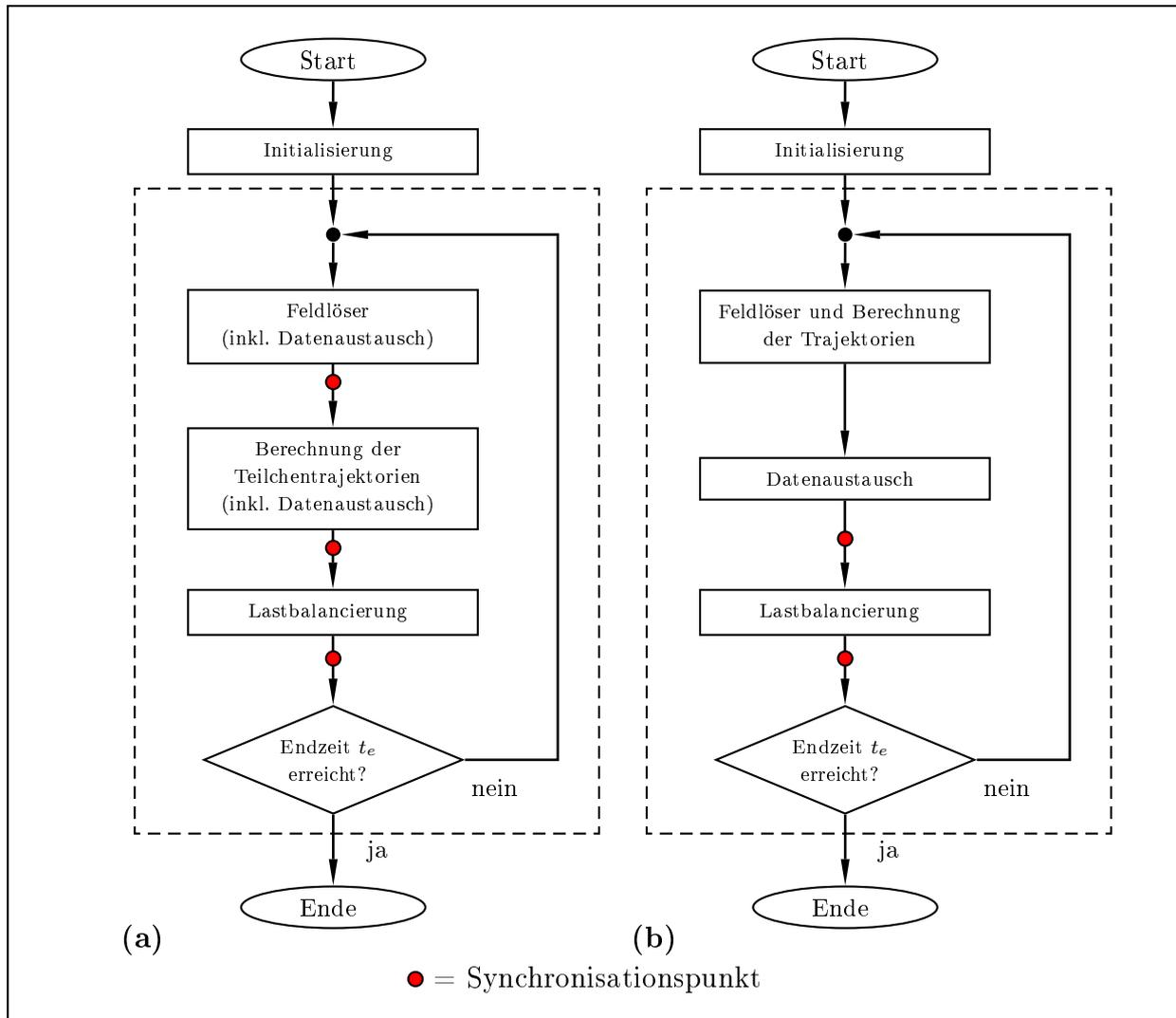
ten Phase. Unter dieser Annahme kann die Laufzeit abgeschätzt werden zu

$$\begin{aligned}
 T_S(N_\pi, \mathcal{P}) = \sum_{m=1}^M & \left( \max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^F \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \right. \\
 & + \max_{i_\pi \in \Pi} \left\{ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \\
 & \left. + \max_{i_\pi \in \Pi} \left\{ {}^{LB} \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \right). \quad (\text{synchroner Fall})
 \end{aligned} \tag{3.37}$$

Durch eine Duplizierung von Berechnungen kann in bestimmten Fällen der gesamte Datenaustausch zu einem Zeitpunkt nach den Berechnungen eines Zeitschrittes durchgeführt werden, so dass einer der Synchronisationspunkte vermieden werden kann. Abbildung 3.6(b) zeigt schematisch den Ablauf des Algorithmus für diesen Fall. Welche Berechnungen zu duplizieren sind, hängt von der Zuordnung der Felddaten zu den Prozessoren ab. Dies soll zu einem späteren Zeitpunkt genauer betrachtet werden (Abschnitt 3.4.6.1). Aufgrund des Wegfalls des Synchronisationspunktes zwischen den beiden Phasen ergibt sich als Abschätzung für die Laufzeit

$$\begin{aligned}
 T_S(N_\pi, \mathcal{P}) = \sum_{m=1}^M & \left( \max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^F \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right. \right. \\
 & + \left. \left. {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \right. \\
 & \left. + \max_{i_\pi \in \Pi} \left\{ {}^{LB} \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \right). \quad (\text{asynchroner Fall})
 \end{aligned} \tag{3.38}$$

Es handelt sich bei dem durch die erweiterte Zielfunktion gegebenen Optimierungsproblem, wie schon im Falle des Feldlösers, um ein nichtlineares kombinatorisches Optimierungsproblem [64]. Bereits für das Optimierungsproblem in Abschnitt 3.3 war die Suche nach einer Strategie, die zu einer Optimallösung führt, aus den dort erläuterten Gründen nicht praktikabel, und es kam eine Heuristik zur Anwendung, welche mit geringem Laufzeitaufwand eine hinreichend gute Lösung lieferte. Durch die Erweiterung der Zielfunktion für den vollständigen PIC Algorithmus ist das zu lösende Optimierungsproblem nochmals schwieriger geworden, so dass auch hier wieder Heuristiken entwickelt werden müssen. Erschwert wird die Lösung durch die Tatsache, dass sich die Verteilung der Teilchen über das Rechengebiet im Verlauf der Simulation in einer Weise ändert, die a priori unbekannt ist. Die zu einer Optimierung im Vorfeld der Simulation notwendige Information der Zuordnung von Teilchen zu Gitterzellen in jedem Zeitschritt der Berechnung ist somit nicht verfügbar und offenbart sich erst im Laufe der Simulation. Somit muss die Zuordnung von Teilchen und Gitterzellen zu den Prozessoren im Laufe der Simulation basierend auf den sich ergebenden Werten der  $\tilde{n}_{i_c, p}^{\text{cp}(m)}$  immer wieder dynamisch angepasst werden. Diese Änderung



**Abbildung 3.6:** Ablaufdiagramm des parallelen PIC Algorithmus. (a) Der Datenaustausch findet nach jeder Berechnungsphase statt. Damit entstehen insgesamt drei Synchronisationspunkte in einem Durchlauf der Schleife. (b) Der Datenaustausch findet nach dem vollständigen Abschluss der Berechnungen statt. Damit verschwindet einer der Synchronisationspunkte, jedoch ist ein größerer Datenaustausch am Rand der einzelnen Teilgebiete notwendig.

der Zuordnung von Objekten zu Prozessoren während der Laufzeit wird als dynamische Lastzuordnung oder auch dynamische Lastbalancierung bezeichnet [47]. Die dafür benötigte Laufzeit fließt über den Term  ${}^{LB}\tilde{T}_S^{(m)}$  in die Zielfunktion ein. Im Gegensatz dazu steht die statische Lastzuordnung, wie sie für den Feldlöser ausreichend war. Dabei erfolgt die Zuordnung der Daten zu den Prozessoren nur ein einziges Mal und ändert sich danach nicht mehr.

Bevor im folgenden Abschnitt unterschiedliche, im Rahmen der Arbeit entwickelte Lö-

sungsheuristiken vorgestellt und soweit wie möglich theoretisch analysiert werden, sollen anhand der Zielfunktionen (3.37) bzw. (3.38), wie bereits für den Feldlöser, noch einige Überlegungen angestellt werden, welche grundsätzlichen Eigenschaften einer Parallelisierungsstrategie dazu führen, dass  $T_S$  tendenziell minimiert wird. Für eine Übersicht über die wichtigsten Parallelisierungsstrategien für PIC, welche in der Literatur zu finden sind, sei auf Anhang C verwiesen.

Bei Betrachtung des Terms  ${}^P\tilde{T}_S^{(m)}$  fällt auf, dass dieser umso kleiner wird, je mehr Teilchen dem gleichen Prozessor zugeordnet werden wie die Gitterzelle, in der sie sich befinden. In dem Fall, dass eine Parallelisierungsstrategie die Zuordnung der Teilchen an die Zuordnung der Gitterzellen koppelt, d.h. ein Teilchen wird immer genau dem Prozessor zugewiesen, dem auch die Gitterzelle zugeordnet ist, in der sich das Teilchen befindet, so wird  ${}^P\tilde{T}_S^{(m)} = 0$  und nimmt damit seinen kleinstmöglichen Wert an. In diesem Fall sind die  ${}^{P\pi}a_{p,i_\pi}^{(m)}$  festgelegt als

$${}^{P\pi}a_{p,i_\pi}^{(m)} = \sum_{i_c \in \mathcal{I}} n_{i_c,p}^{cp(m)} \cdot a_{i_c,i_\pi}^{c\pi(m)}. \quad (3.39)$$

Der Term  ${}^{LB}\tilde{T}_S^{(m)}$ , welcher die benötigte Laufzeit für die Neuordnung von Teilchen und Gitterzellen im Rahmen der dynamischen Lastbalancierung beschreibt, wird umso kleiner, je kleiner die Menge an Daten ist, für welche die Prozessorzuordnung verändert wird. Da  ${}^{LB}\tilde{T}_S^{(m)}$  als einziger Term innerhalb der Zielfunktion von den Entscheidungsvariablen zweier verschiedener Zeitschritte abhängt, nimmt er eine Sonderstellung ein. Seine Behandlung innerhalb einer Parallelisierungsstrategie soll im nächsten Abschnitt genauer untersucht werden.

### 3.4.2 Behandlung des Lastbalancierungsaufwandes

Der Term  ${}^{LB}\tilde{T}_S^{(m)}$  verkoppelt die Entscheidungsvariablen der einzelnen Zeitschritte miteinander. Ohne Berücksichtigung dieses Terms in der Zielfunktion zerfällt das Optimierungsproblem (3.37) bzw. (3.38) in  $M$  voneinander unabhängige Probleme. Die Zuordnung der Gitterzellen und Teilchen zum Zeitschritt  $m$  kann dann ausschließlich auf den  $n_{i_c,p}^{cp(m)}$  basieren. Da sich dadurch die Bewegung großer Datenmengen nicht mehr negativ auf die Zielfunktion auswirkt, besteht zunächst die Gefahr, Lastbalancierungsverfahren zu konstruieren, welche die bestehende Zuordnung der Daten bei der Berechnung einer Neuordnung vollkommen unberücksichtigt lassen. Das kann dazu führen, dass in jedem Schritt eine große Menge an Daten im Rahmen des Lastbalancierungsprozesses auszutauschen ist. Im Wesentlichen existieren zwei Möglichkeiten, dies durch die Konstruktion des Lastbalancierungsalgorithmus zu vermeiden [71].

Zum einen ändert sich die Verteilung der Teilchen im Rechengbiet nur langsam. Führt

nun ein Lastbalancierungsalgorithmus für ähnliche Teilchenverteilungen zu einer ähnlichen Zuordnung von Teilchen und Gitterzellen zu den Prozessoren, so wird automatisch vermieden, dass eine unnötig große Menge an Daten bewegt werden muss, um zu einer guten Lastverteilung zu kommen.

Zum anderen kann ein Lastbalancierungsalgorithmus so konstruiert werden, dass, ausgehend von der bereits vorliegenden Verteilung der Daten, lediglich inkrementelle Änderungen zugelassen werden, so dass die bereits vorliegende Zuordnung von Teilchen und Gitterpunkten implizit berücksichtigt wird.

Für die im Rahmen der vorliegenden Arbeit entwickelten Lastbalancierungsstrategien kommen beide Möglichkeiten zum Einsatz, worauf an entsprechender Stelle hingewiesen wird.

### 3.4.3 Dynamische Teilchenzuordnung

Eine Zuordnung der Teilchen und Gitterzellen ohne die Berücksichtigung der durch  $\tilde{n}_{i_c,p}^{cp(m)}$  beschriebenen Abhängigkeiten führt, trotz idealer Verteilung der Rechenlast, aufgrund eines im Allgemeinen hohen Aufwandes für den Datenaustausch  ${}^P\tilde{T}_S^{(m)}$  nicht zu einer praktikablen Parallelisierungsstrategie wie frühere Untersuchungen zeigten [1, 2].

In diesem Abschnitt soll daher eine im Rahmen dieser Arbeit entwickelte Parallelisierungsstrategie vorgestellt werden, mit deren Hilfe, basierend auf einer bereits gegebenen Zuordnung der Gitterzellen zu den verfügbaren Prozessoren, die Zuordnung der Teilchen so bestimmt werden kann, dass die Zielfunktion minimiert wird. Die Zuordnung der Gitterzellen kann mit dem in Abschnitt 3.3.1 oder einem der in Anhang C beschriebenen Verfahren zunächst unabhängig von der Teilchenverteilung vorgenommen werden.

Es sei angenommen, dass die Zuordnung der Gitterzellen so erfolgte, dass alle Prozessoren die gleiche Zeit  ${}^F T_0$  für den Feldlöser und den damit verbundenen Datenaustausch benötigen und dass die Zuordnung der Gitterzellen im Laufe der Simulation nicht modifiziert wird.

Wie in Abschnitt 3.4.2 angedeutet, soll der Term  ${}^{LB}\tilde{T}_S^{(m)}$  zwar nicht explizit in der Zielfunktion berücksichtigt werden, jedoch wird der Lastbalancierungsalgorithmus derart konstruiert, dass sich für eine nur geringfügig modifizierte Teilchenverteilung auch nur eine geringfügig geänderte Zuordnung der Teilchen zu den Prozessoren ergibt.

Unter Berücksichtigung der beschriebenen Annahmen vereinfacht sich die Zielfunktion zu

$$T_S(N_\pi, \mathcal{P}) = M \cdot {}^F T_0 + \sum_{m=1}^M \max_{i_\pi \in \Pi} \left\{ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\}. \quad (3.40)$$

Die Summanden in (3.40) sind unabhängig voneinander, da jeder der Summanden nur die Entscheidungsvariablen für einen bestimmten Zeitschritt enthält. Somit kann jeder

einzelne Summand separat minimiert werden, und die Betrachtung kann auf die optimierte Zuordnung der Teilchen zu den Prozessoren für einen Zeitschritt reduziert werden. Der konstante Anteil  $M \cdot {}^F T_0$  für den Feldlöser spielt für die Optimierung keine Rolle und wird daher ohne Beschränkung der Allgemeingültigkeit zu Null gesetzt. Für einen beliebigen Zeitschritt  $m$  lautet die Zielfunktion somit

$$T_S^{(m)}(N_\pi, \mathcal{P}) = \max_{i_\pi \in \Pi} \left\{ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\}. \quad (3.41)$$

Da die Optimierung für jeden Zeitschritt getrennt erfolgen kann, wird im Folgenden der hochgestellte Index an allen Größen fortgelassen.

Für die in Gleichung (3.35) auftretenden Terme soll folgende abkürzende Schreibweise verwendet werden:

$${}^{n\pi} a_{i_\pi, j_\pi} := \sum_{\mathbf{i}_c \in \mathcal{I}} \sum_{p=1}^P {}^{p\pi} a_{p, i_\pi} \cdot {}^{c\pi} a_{\mathbf{i}_c, j_\pi} \cdot {}^{cp} n_{\mathbf{i}_c, p}. \quad (3.42)$$

Außerdem sei

$$s_{i_\pi} := \sum_{\mathbf{i}_c \in \mathcal{I}} \sum_{p=1}^P {}^{c\pi} a_{\mathbf{i}_c, i_\pi} \cdot {}^{cp} n_{\mathbf{i}_c, p}. \quad (3.43)$$

Die Größe  ${}^{n\pi} a_{i_\pi, j_\pi}$  beschreibt die Anzahl der Teilchen, welche Prozessor  $i_\pi$  zugeordnet werden und sich in einer Gitterzelle befinden, die Prozessor  $j_\pi$  zugeordnet wurde. Die Größe  $s_{i_\pi}$  beschreibt die Gesamtzahl aller Teilchen, welche sich in Gitterzellen befinden, die Prozessor  $i_\pi$  zugeordnet wurden. Ziel der Optimierung ist, die  ${}^{n\pi} a_{i_\pi, j_\pi}$  basierend auf den Werten von  $s_{i_\pi}$  so zu bestimmen, dass die Zielfunktion (3.41) minimiert wird. Die Auswahl der Teilchen, d.h. die konkrete Wahl der  ${}^{p\pi} a_{p, i_\pi}$ , hat dann lediglich so zu erfolgen, dass sich die berechneten Werte für die  ${}^{n\pi} a_{i_\pi, j_\pi}$  ergeben. Diese Vereinfachung ist möglich, da es sich bei den Teilchen bezüglich der mit ihnen verbundenen Rechenlast um ununterscheidbare Objekte handelt.

Die Zielfunktion lässt sich mit Hilfe der Beziehung

$$\sum_{p=1}^P {}^{p\pi} a_{p, i_\pi} = {}^{n\pi} a_{i_\pi, i_\pi} + \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} {}^{n\pi} a_{i_\pi, j_\pi}, \quad (3.44)$$

welche sich mit Hilfe von Gleichung (3.17) sowie dem Zusammenhang

$$\sum_{\mathbf{i}_c \in \mathcal{I}} {}^{cp} n_{\mathbf{i}_c, p} = 1 \quad \forall p \in \{1 \dots P\} \quad (3.45)$$

aus Gleichung (3.42) ableiten lässt, vollständig in Abhängigkeit der  $a_{i_\pi, j_\pi}^{n\pi}$  formulieren als

$$T_S(N_\pi, \mathcal{P}) = \max_{i_\pi \in \Pi} \left\{ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n\pi} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} a_{i_\pi, j_\pi}^{n\pi} + \gamma \cdot \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} a_{j_\pi, i_\pi}^{n\pi} \right\}, \quad (3.46)$$

was sich durch Einsetzen von Gleichung (3.42) und (3.44) in Gleichung (3.34) und (3.35) ergibt. Die Nebenbedingung aus Gleichung (3.33) lässt sich sofort auf die neuen Variablen übertragen. Außerdem müssen die  $a_{i_\pi, j_\pi}^{n\pi}$  sinnvollerweise positiv sein. Die vollständigen Nebenbedingungen für das Optimierungsproblem lauten somit

$$\sum_{j_\pi=1}^{N_\pi} a_{j_\pi, i_\pi}^{n\pi} = s_{i_\pi} \quad \forall i_\pi \in \Pi, \quad (3.47)$$

$$a_{i_\pi, j_\pi}^{n\pi} \geq 0 \quad \forall i_\pi, j_\pi \in \Pi. \quad (3.48)$$

Für die Optimierung soll angenommen werden, dass  $a_{i_\pi, j_\pi}^{n\pi} \in \mathbb{R}$  gilt. Aus dem diskreten Optimierungsproblem wird damit ein kontinuierliches Optimierungsproblem, welches sich mit dem in diesem Abschnitt beschriebenen Algorithmus optimal lösen lässt. Im Anschluss wird erläutert, wie die diskreten Werte der  $a_{i_\pi, j_\pi}^{n\pi}$  durch Runden aus den kontinuierlichen bestimmt werden können, um eine gute Approximation für die Lösung des diskreten Optimierungsproblems zu erhalten. Diese Vereinfachung lässt sich damit rechtfertigen, dass bei einer PIC Simulation die Anzahl der Makroteilchen üblicherweise als sehr viel größer angenommen werden kann als die Zahl der Prozessoren des verwendeten Parallelrechners. Deswegen ist zu erwarten, dass die  $a_{i_\pi, j_\pi}^{n\pi}$  so groß sein werden, dass sich der relative Fehler, welcher durch das Runden der Werte eingeführt wird, kaum bemerkbar machen wird. Dieses Vorgehen ist eine zur approximativen Lösung diskreter Optimierungsprobleme häufig angewandte Strategie [64].

Ohne Beschränkung der Allgemeingültigkeit sei angenommen, dass

$$\beta_{i_\pi} \cdot s_{i_\pi} \leq \beta_{j_\pi} \cdot s_{j_\pi} \quad \forall i_\pi \leq j_\pi \quad (3.49)$$

gilt. Ist dies nicht der Fall, so kann durch eine Permutation der Prozessoren erreicht werden, dass Bedingung (3.49) erfüllt ist.

Ausgehend von einem Referenzzustand sollen die  $a_{i_\pi, j_\pi}^{n\pi}$  nun iterativ berechnet werden, so dass schließlich die Zielfunktion nach maximal  $N_\pi - 1$  Iterationsschritten ihr Minimum annimmt. Die  $a_{i_\pi, j_\pi}^{n\pi}$ , welche sich im  $n$ -ten Schritt des Optimierungsalgorithmus ergeben, werden mit  $a_{i_\pi, j_\pi}^{n\pi(n)}$  bezeichnet. Für das Argument der  $\max\{\cdot\}$  Funktion im  $n$ -ten Iterationsschritt in Gleichung (3.46) wird im Folgenden kurz  $T_S^{(n)}(i_\pi)$  geschrieben. Man beachte,

dass sich der hochgestellte Index nun auf den Iterationsschritt des Optimierungsalgorithmus bezieht und nicht auf den Zeitschritt der Simulation.

Die  $a_{i_\pi, j_\pi}^{n\pi(n+1)}$  ergeben sich aus den Werten der  $a_{i_\pi, j_\pi}^{n\pi(n)}$  und einer noch zu berechnenden Korrektur  $\Delta a_{i_\pi, j_\pi}^{n\pi(n)}$ . Es gilt

$$\boxed{a_{i_\pi, j_\pi}^{n\pi(n+1)} = a_{i_\pi, j_\pi}^{n\pi(n)} + \Delta a_{i_\pi, j_\pi}^{n\pi(n)}} \quad (3.50)$$

Die Nebenbedingungen aus Gleichung (3.47) und (3.48) gelten natürlich auch für alle im Laufe des Optimierungsverfahrens berechneten Zwischenwerte, so dass

$$\sum_{j_\pi=1}^{N_\pi} a_{j_\pi, i_\pi}^{n\pi(n)} = s_{i_\pi} \quad \forall i_\pi \in \Pi, n \in \mathbb{N}, \quad (3.51)$$

$$a_{i_\pi, j_\pi}^{n\pi(n)} \geq 0 \quad \forall i_\pi, j_\pi \in \Pi, n \in \mathbb{N} \quad (3.52)$$

jederzeit erfüllt sein muss. Aus Gleichung (3.51) ergibt sich sofort, dass für die Korrekturen

$$\sum_{j_\pi=1}^{N_\pi} \Delta a_{j_\pi, i_\pi}^{n\pi(n)} = 0 \quad \forall i_\pi \in \Pi, n \in \mathbb{N} \quad (3.53)$$

gelten muss.

### Initialisierungsschritt:

Die Variablen  $a_{i_\pi, j_\pi}^{n\pi(0)}$  seien gegeben durch

$$a_{i_\pi, j_\pi}^{n\pi(0)} = \begin{cases} s_{i_\pi} & \text{falls } i_\pi = j_\pi \\ 0 & \text{sonst.} \end{cases} \quad (3.54)$$

Die Werte der  $T_S^{(0)}(i_\pi)$  ergeben sich daher zu

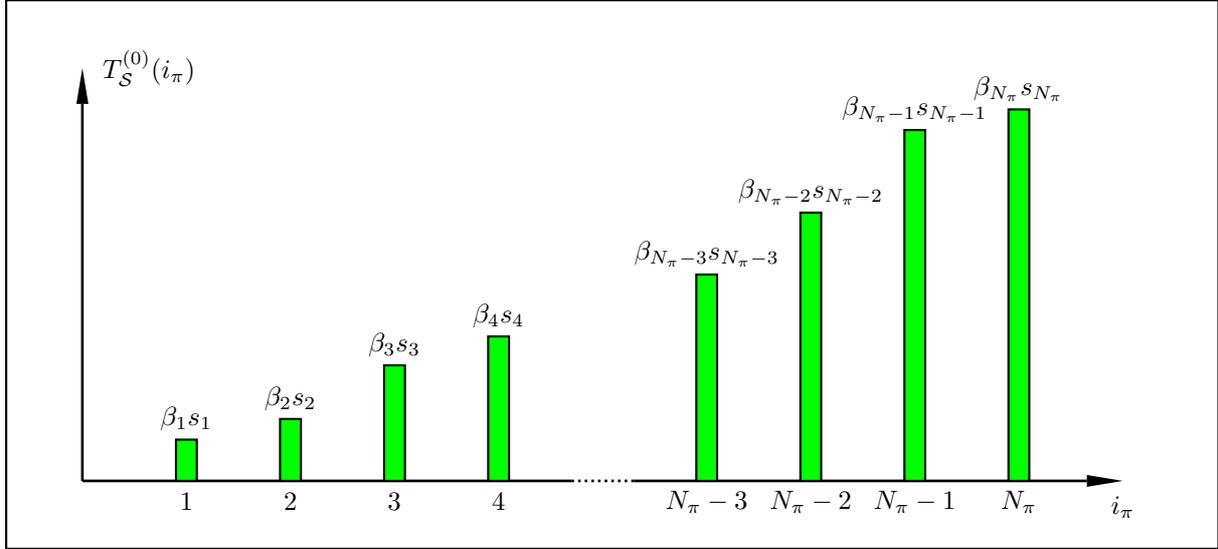
$$T_S^{(0)}(i_\pi) = \beta_{i_\pi} \cdot s_{i_\pi}. \quad (3.55)$$

Abbildung 3.7 zeigt den beschriebenen Referenzzustand.

### Iterationsschritt:

Seien die Größen  $l^{(n)}, u^{(n)} \in \Pi$  so festgelegt, dass gilt:

$$\begin{aligned} T_S^{(n)}(l^{(n)}) &= T_S^{(n)}(i_\pi) \quad \forall i_\pi \leq l^{(n)} \wedge \dots \\ T_S^{(n)}(l^{(n)}) &< T_S^{(n)}(i_\pi) \quad \forall i_\pi > l^{(n)} \end{aligned} \quad (3.56)$$



**Abbildung 3.7:** Ausgehend von dem abgebildeten Referenzzustand, in dem alle Teilchen dem Prozessor zugeordnet werden, dem die Gitterzelle zugeordnet ist, in der sie sich befinden, wird die Optimierung vorgenommen.

sowie

$$\begin{aligned} T_S^{(n)}(u^{(n)}) &= T_S^{(n)}(i_\pi) \quad \forall i_\pi \geq u^{(n)} \wedge \dots \\ T_S^{(n)}(u^{(n)}) &> T_S^{(n)}(i_\pi) \quad \forall i_\pi < u^{(n)}. \end{aligned} \quad (3.57)$$

Abbildung 3.8 illustriert die beiden Größen  $l^{(n)}$  und  $u^{(n)}$ . Da der Wert der Zielfunktion wegen des  $\max\{\cdot\}$  Ausdrucks durch die  $T_S^{(n)}(u^{(n)} \dots N_\pi)$  bestimmt wird, müssen diese zur Reduktion des Wertes der Zielfunktion alle reduziert werden. Das wird dadurch erreicht, dass die Gruppe der Prozessoren  $u^{(n)} \dots N_\pi$  Teilchen an die Prozessoren  $1 \dots l^{(n)}$  abgibt. Dies geschieht so lange, bis entweder  $T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi) \quad \forall i_\pi, j_\pi \in \Pi$  gilt, oder einer der Prozessoren  $u^{(n)} \dots N_\pi$  keine Teilchen mehr besitzt, die abgegeben werden könnten. Damit ergeben sich die folgenden Abbruchbedingungen.

Falls

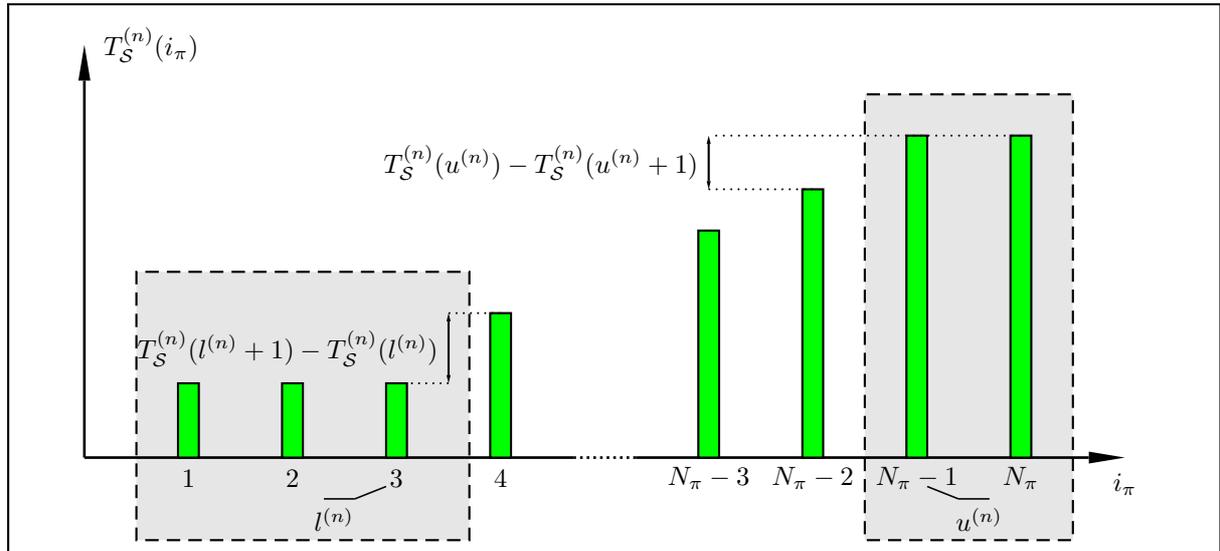
$$T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi) \quad \forall i_\pi, j_\pi \in \Pi \quad (3.58)$$

gilt, dann ist der bestmögliche Zustand erreicht und das Verfahren wird beendet. Ebenso wird das Verfahren beendet, falls es einen Prozessor  $i_\pi$  mit  $i_\pi \geq u^{(n)}$  gibt, für den

$$a_{i_\pi, i_\pi}^{n\pi(n)} = 0 \quad (3.59)$$

oder

$$\beta_{i_\pi} < \gamma \quad (3.60)$$



**Abbildung 3.8:** Alle Prozessoren mit  $i_\pi \geq u^{(n)}$  oder  $i_\pi \leq l^{(n)}$  besitzen den gleichen Wert für  $T_S^{(n)}(i_\pi)$ .

gilt, da auch in diesem Fall der Zustand nicht mehr verbessert werden kann. Ansonsten werden die  $\Delta a_{i_\pi, j_\pi}^{n\pi^{(n)}}$  wie folgt bestimmt.

Zunächst wird die Gesamtzahl der Teilchen berechnet, welche insgesamt von der Gruppe der Prozessoren  $u^{(n)} \dots N_\pi$  an die Prozessoren  $1 \dots l^{(n)}$  abgegeben werden sollen. Dazu muss eine Fallunterscheidung bezüglich  $l^{(n)}$  und  $u^{(n)}$  vorgenommen werden.

**Fall 1:**  $l^{(n)} < u^{(n)} - 1$

Die Größen  $\Delta a_{i_\pi, j_\pi}^{n\pi^{(n)}}$  sollen so bestimmt werden, dass

$$\text{entweder } l^{(n+1)} \geq l^{(n)} + 1 \quad \text{oder} \quad u^{(n+1)} \leq u^{(n)} - 1 \quad (3.61)$$

gilt. Es dürfen nur maximal so viele Teilchen von jedem Prozessor abgegeben werden, dass die Nebenbedingung (3.52) nicht verletzt wird. Diese Bedingung impliziert, dass der Wert von  $T_S^{(n)}(u^{(n)} \dots N_\pi)$  nur um maximal

$$\Delta T_{S \max}^{(n)} := \min_{i_\pi \in \{u^{(n)} \dots N_\pi\}} \left\{ (\beta_{i_\pi} - \gamma) \cdot a_{i_\pi, i_\pi}^{n\pi^{(n)}} \right\} \quad (3.62)$$

reduziert werden kann. Bezeichne  $\Delta a^{n\pi^{(n)}}$  die insgesamt von den Prozessoren der Gruppe  $u^{(n)} \dots N_\pi$  an die Prozessoren der Gruppe  $1 \dots l^{(n)}$  abgegebene Teilchenzahl. Damit sich  $l^{(n+1)} \geq l^{(n)} + 1$  ergibt oder einer der Prozessoren  $u^{(n)} \dots N_\pi$  keine Teilchen mehr be-

sitzt, muss

$$\Delta a_{\textcircled{1}}^{n\pi(n)} = \min \left\{ \Delta T_{\mathcal{S}}^{(n)} \max ; \left( T_{\mathcal{S}}^{(n)}(l^{(n)} + 1) - T_{\mathcal{S}}(l^{(n)}) \right) \right\} \cdot \sum_{i_{\pi}=1}^{l^{(n)}} \frac{1}{\beta_{i_{\pi}} + \gamma} \quad (3.63)$$

sein. Damit  $u^{(n+1)} \leq u^{(n)} - 1$  gilt muss entsprechend

$$\Delta a_{\textcircled{2}}^{n\pi(n)} = \min \left\{ \Delta T_{\mathcal{S}}^{(n)} \max ; \left( T_{\mathcal{S}}^{(n)}(u^{(n)}) - T_{\mathcal{S}}(u^{(n)} - 1) \right) \right\} \cdot \sum_{i_{\pi}=u^{(n)}}^{N_{\pi}} \frac{1}{\beta_{i_{\pi}} - \gamma} \quad (3.64)$$

gelten. Die tatsächliche Anzahl an Teilchen, welche abgegeben wird, wird gemäß

$$\Delta a^{n\pi(n)} := \min \left\{ \Delta a_{\textcircled{1}}^{n\pi(n)} ; \Delta a_{\textcircled{2}}^{n\pi(n)} \right\} \quad (3.65)$$

gewählt.

**Fall 2:**  $l^{(n)} = u^{(n)} - 1$

Falls  $l^{(n)} = u^{(n)} - 1$  gilt, muss die Bestimmung von  $\Delta a^{n\pi(n)}$  etwas anders erfolgen. Wiederum werden nur Teilchen von der Gruppe der Prozessoren  $u^{(n)} \dots N_{\pi}$  an Prozessoren der Gruppe  $1 \dots l^{(n)}$  abgegeben. Der Wert von  $\Delta a^{n\pi(n)}$  wird in diesem Fall so gewählt, dass entweder

$$T_{\mathcal{S}}^{(n+1)}(u^{(n)} \dots N_{\pi}) = T_{\mathcal{S}}^{(n+1)}(1 \dots l^{(n)}) \quad (3.66)$$

gilt, oder ein Prozessor der Gruppe  $u^{(n)} \dots N_{\pi}$  keine Teilchen mehr besitzt. Der Ausdruck in Gleichung (3.62) gibt wiederum den maximalen Wert an, um den  $T_{\mathcal{S}}^{(n)}(u^{(n)} \dots N_{\pi})$  reduziert werden kann. Bei einer Abgabe von insgesamt  $\Delta a^{n\pi(n)}$  Teilchen ändert sich  $T_{\mathcal{S}}^{(n)}(1 \dots l^{(n)})$  gemäß

$$\Delta T_{\mathcal{S}}^{(n)}(1 \dots l^{(n)}) = \frac{\Delta a^{n\pi(n)}}{\sum_{k_{\pi}=1}^{l^{(n)}} \frac{1}{\beta_{k_{\pi}} + \gamma}} \quad (3.67)$$

und  $T_{\mathcal{S}}^{(n)}(u^{(n)} \dots N_{\pi})$  gemäß

$$\Delta T_{\mathcal{S}}^{(n)}(u^{(n)} \dots N_{\pi}) = - \frac{\Delta a^{n\pi(n)}}{\sum_{k_{\pi}=u^{(n)}}^{N_{\pi}} \frac{1}{\beta_{k_{\pi}} - \gamma}}. \quad (3.68)$$

Damit ergibt sich zunächst aus der Bedingung (3.66)

$$T_{\mathcal{S}}^{(n)}(u^{(n)} \dots N_{\pi}) + \Delta T_{\mathcal{S}}^{(n)}(u^{(n)} \dots N_{\pi}) = T_{\mathcal{S}}^{(n)}(1 \dots l^{(n)}) + \Delta T_{\mathcal{S}}^{(n)}(1 \dots l^{(n)}). \quad (3.69)$$

Einsetzen von Gleichung (3.67) und (3.68) in Gleichung (3.69) ergibt

$$\Delta a^{n\pi(n)} = \frac{T_S^{(n)}(u^{(n)} \dots N_\pi) - T_S^{(n)}(1 \dots l^{(n)})}{\frac{1}{\sum_{k_\pi=1}^{l^{(n)}} \frac{1}{\beta_{k_\pi} + \gamma}} + \frac{1}{\sum_{k_\pi=u^{(n)}}^{N_\pi} \frac{1}{\beta_{k_\pi} - \gamma}}}. \quad (3.70)$$

Falls von Gleichung (3.70) Bedingung (3.62) verletzt wird, so muss  $\Delta a^{n\pi(n)}$  stattdessen gewählt werden gemäß

$$\Delta a^{n\pi(n)} = \Delta T_S^{(n)} \max_{i_\pi=u^{(n)}} \sum_{i_\pi=u^{(n)}}^{N_\pi} \frac{1}{\beta_{i_\pi} - \gamma}. \quad (3.71)$$

Damit ist die Fallunterscheidung beendet und aus dem Wert von  $\Delta a^{n\pi(n)}$  können die Werte der  $\Delta a_{i_\pi, j_\pi}^{n\pi(n)}$  bestimmt werden.

Damit die  $T_S^{(n+1)}$  für die Prozessoren der Gruppe  $1 \dots l^{(n)}$  und  $u^{(n)} \dots N_\pi$  um den gleichen Wert erhöht bzw. reduziert werden, müssen die insgesamt von der Gruppe der Prozessoren  $u^{(n)} \dots N_\pi$  an die Gruppe  $1 \dots l^{(n)}$  abzugebenden Teilchen entsprechend verteilt werden. Von den insgesamt von der Gruppe der Prozessoren  $u^{(n)} \dots N_\pi$  abgegebenen Teilchen muss daher ein Anteil von

$$\frac{\Delta a_{\dots, j_\pi}^{n\pi(n)}}{\Delta a^{n\pi(n)}} = \frac{\frac{1}{\beta_{j_\pi} - \gamma}}{\sum_{k_\pi=u^{(n)}}^{N_\pi} \frac{1}{\beta_{k_\pi} - \gamma}} \quad (3.72)$$

auf Prozessor  $j_\pi$  aus dieser Gruppe entfallen, wobei  $\Delta a_{\dots, j_\pi}^{n\pi(n)}$  definiert ist als

$$\Delta a_{\dots, j_\pi}^{n\pi(n)} := \sum_{k_\pi=u^{(n)}}^{N_\pi} \Delta a_{k_\pi, j_\pi}^{n\pi(n)}, \quad (3.73)$$

also als Gesamtanzahl der Teilchen, die Prozessor  $j_\pi$  an die Gruppe der Prozessoren  $1 \dots l^{(n)}$  abgeben muss. Die  $\Delta a_{\dots, j_\pi}^{n\pi(n)}$  von Prozessor  $j_\pi$  abgegebenen Teilchen sollen so auf die Prozessoren  $1 \dots l^{(n)}$  verteilt werden, dass sie den Wert von  $T_S^{(n)}$  für jeden dieser Prozessoren um den gleichen Wert erhöhen. Es folgt daher für einen Prozessor  $i_\pi$  aus dieser Gruppe

$$\frac{\Delta a_{i_\pi, j_\pi}^{n\pi(n)}}{\Delta a_{\dots, j_\pi}^{n\pi(n)}} = \frac{\frac{1}{\beta_{i_\pi} + \gamma}}{\sum_{k_\pi=1}^{l^{(n)}} \frac{1}{\beta_{k_\pi} + \gamma}}. \quad (3.74)$$

Somit ergeben sich die gesuchten Werte der  $\Delta a_{i_\pi, j_\pi}^{n\pi(n)}$  zu

$$\Delta a_{i_\pi, j_\pi}^{n\pi(n)} = \frac{1}{\beta_{i_\pi + \gamma}} \cdot \frac{1}{\beta_{j_\pi - \gamma}} \cdot \Delta a_{i_\pi, j_\pi}^{n\pi(n)} \quad \forall i_\pi \in \{1 \dots l^{(n)}\}, j_\pi \in \{u^{(n)} \dots N_\pi\}. \quad (3.75)$$

Aus Gleichung (3.53) ergibt sich weiterhin, dass

$$\Delta a_{j_\pi, j_\pi}^{n\pi(n)} = - \sum_{i_\pi=1}^{l^{(n)}} \Delta a_{i_\pi, j_\pi}^{n\pi(n)} \quad (3.76)$$

gelten muss. Die verbleibenden  $\Delta a_{i_\pi, j_\pi}^{n\pi(n)}$  werden zu Null gesetzt. Damit ist der Iterationsschritt beendet. Abbildung 3.9 illustriert den Ablauf des Optimierungsverfahrens.

Das beschriebene Verfahren führt zu einer Optimallösung für den Fall  $a_{i_\pi, j_\pi}^{n\pi(n)} \in \mathbb{R}$ . Für den Beweis dieser Behauptung sei auf Anhang D verwiesen.

Aus der Lösung des kontinuierlichen Optimierungsproblems muss nun eine gute Approximation an die Optimallösung des diskreten Optimierungsproblems bestimmt werden. Dies kann für praktische Zwecke dadurch erfolgen, dass die berechneten kontinuierlichen Werte wie folgt modifiziert werden. Werden mit  $a_{i_\pi, j_\pi}^{n\pi}$  nun wieder die diskreten Variablen bezeichnet und mit  $a_{i_\pi, j_\pi}^{n\pi(n)} \in \mathbb{R}$  die Werte, die sich nach Abschluss des Optimierungsverfahrens ergeben, so kann durch

$$a_{i_\pi, j_\pi}^{n\pi} = \lfloor a_{i_\pi, j_\pi}^{n\pi(n)} \rfloor \quad \forall i_\pi \neq j_\pi \quad (3.77)$$

$$a_{i_\pi, i_\pi}^{n\pi} = s_{i_\pi} - \sum_{\substack{j_\pi=1 \\ j_\pi \neq i_\pi}}^{N_\pi} a_{j_\pi, i_\pi}^{n\pi} \quad (3.78)$$

eine zulässige Lösung des diskreten Optimierungsproblems gefunden werden, welche nah an der berechneten Optimallösung liegt.

Wird das Verfahren verwendet, um eine Neuverteilung der Teilchen vorzunehmen, so existiert bereits eine Zuordnung der Teilchen auf die Prozessoren, von der ausgehend der berechnete Zustand erreicht werden muss. Daher ist nun aus den berechneten Werten der  $a_{i_\pi, j_\pi}^{n\pi}$  zu bestimmen, für welche Teilchen die Prozessorzuordnung verändert werden muss. Bezeichne  $a_{i_\pi, j_\pi}^{n\pi \text{ alt}}$  die vorhandene Zuordnung der Teilchen und  $a_{i_\pi, j_\pi}^{n\pi \text{ neu}}$  die gemäß dem

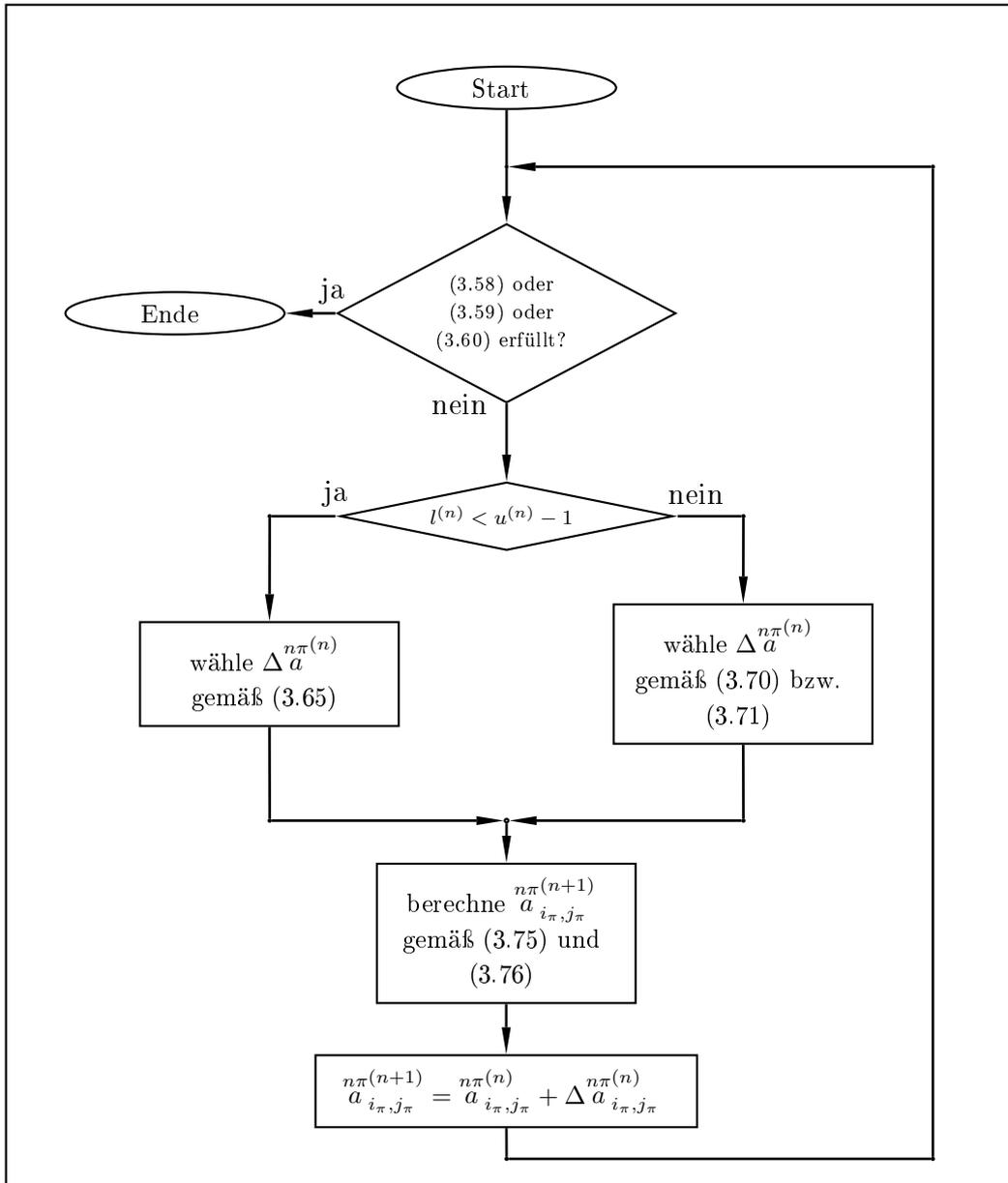
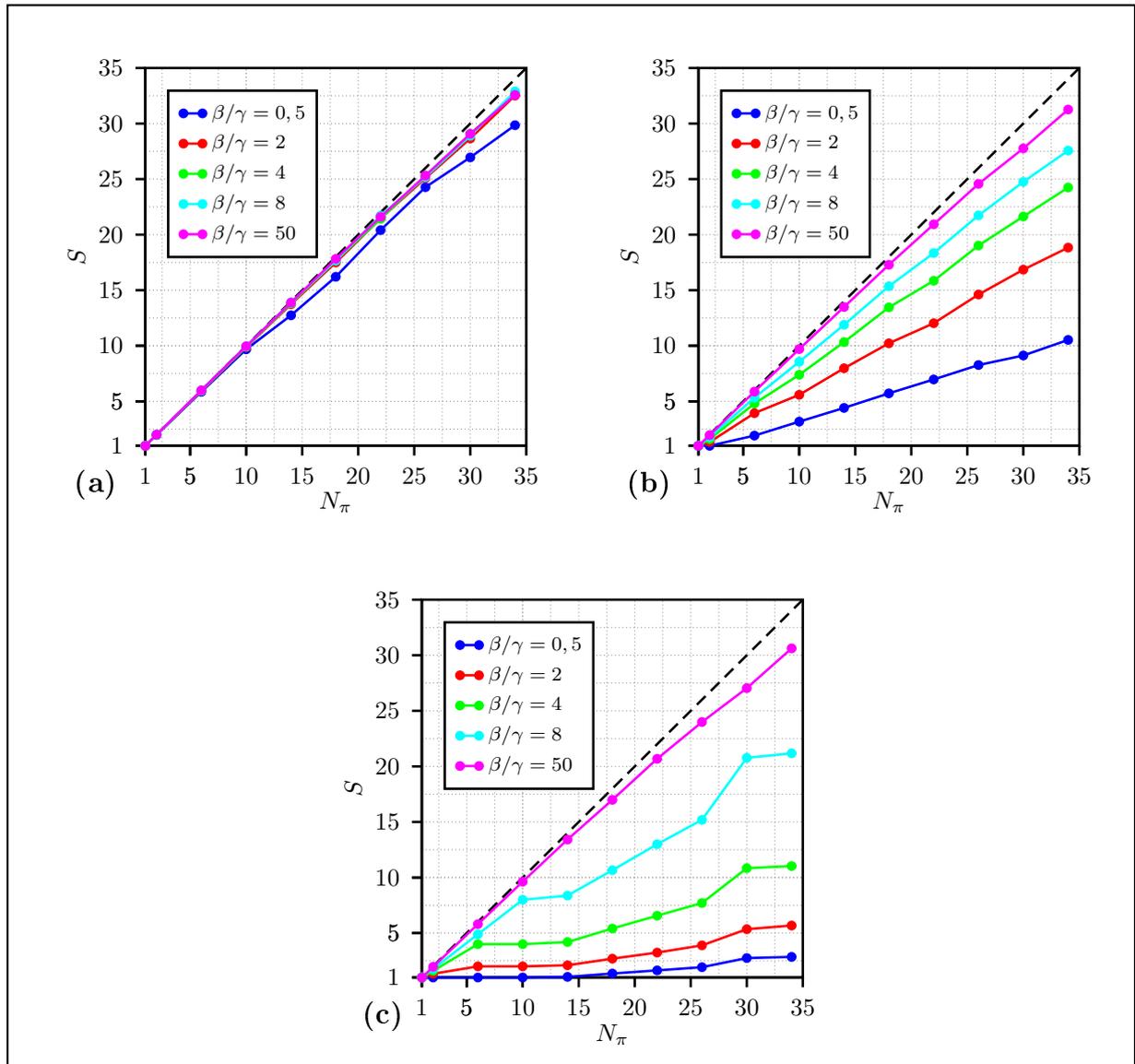


Abbildung 3.9: Das oben stehende Ablaufdiagramm illustriert das in Abschnitt 3.4.3 beschriebene Optimierungsverfahren.

Optimierungsverfahren berechnete neue Zuordnung der Teilchen, so ergibt sich eine Änderung gemäß

$$\Delta a_{i_\pi, j_\pi}^{n\pi} = a_{i_\pi, j_\pi}^{n\pi \text{ neu}} - a_{i_\pi, j_\pi}^{n\pi \text{ alt}}, \quad (3.79)$$

wobei ein negativer Wert von  $\Delta a_{i_\pi, j_\pi}^{n\pi}$  ausdrückt, dass eine entsprechende Anzahl von Teilchen von Prozessor  $i_\pi$  an Prozessor  $j_\pi$  abgegeben werden muss. Ein positiver Wert zeigt an, dass die entsprechende Anzahl an Teilchen von  $j_\pi$  an  $i_\pi$  abgegeben werden muss.



**Abbildung 3.10:** Speedup Werte für die Integration der Teilchentrajektorien vor und nach Anwendung des in Abschnitt 3.4.3 beschriebenen Algorithmus für verschiedene Werte von  $\beta/\gamma$ , wobei es sich bei  $\beta$  um die zur Ausführung der Trajektorienberechnung pro Zeitschritt und pro Teilchen benötigte Laufzeit handelt und  $\gamma$  die Laufzeit beschreibt, welche zum Austausch der Felddaten pro Zeitschritt und pro nichtlokalem Teilchen benötigt wird. Der ideale Verlauf des Speedup ist gestrichelt eingezeichnet. (a) Die Teilchen sind über das komplette Rechengebiet gleichmäßig verteilt. (b) Die Teilchen konzentrieren sich in 10% der Gitterzellen des Rechengebietes. (c) Die Teilchen konzentrieren sich in 1% der Zellen des Rechengebietes.

Da sich die Teilchenverteilung von einem Zeitschritt zum nächsten nur langsam ändert und sich damit auch die  $s_{i_\pi}$  tendenziell langsam ändern, wird das Optimierungsverfahren zu einer ähnlichen Zuordnung der Teilchen führen, wie sie bereits vorliegt

$$a_{i_\pi, j_\pi}^{n\pi \text{ neu}} \approx a_{i_\pi, j_\pi}^{n\pi \text{ alt}}, \quad (3.80)$$

so dass für die  $\Delta a_{i_\pi, j_\pi}^{n\pi}$  und somit auch für  $LB\tilde{T}_S^{(m)}$  kleine Werte zu erwarten sind.

Die Graphen in Abbildung 3.10 zeigen den zu erwartenden Speedup für die Trajektorienberechnung inklusive dem benötigten Datenaustausch, nachdem die Teilchen nach dem oben beschriebenen Algorithmus den Prozessoren zugeordnet wurden. Die Werte wurden über die in der Zielfunktion angenommenen Abhängigkeiten von  $PT_S^{(m)}$  und  $P\tilde{T}_S^{(m)}$  für unterschiedlich stark lokalisierte Teilchenverteilungen bestimmt (siehe Gleichung (3.46)). Die Gebietszerlegung wurde mit Hilfe einer RCB vorgenommen. Es zeigt sich, dass sich die Skalierbarkeit verschlechtert, je stärker sich die Teilchen in einem kleinen Teil des Rechengebietes konzentrieren. Dieses Verhalten resultiert daraus, dass bei einer stark lokalisierten Teilchenverteilung sehr viele Teilchen Felddaten aus einem kleinen Teil des Rechengebietes benötigen. Diese Felddaten sind einigen wenigen oder nur einem einzelnen Prozessor zugeordnet. Für diese Prozessoren entsteht ein „Kommunikationshotspot“, d.h. diese Prozessoren müssen große Datenmengen senden und empfangen, was die erreichbare Performance verschlechtert. Der Effekt wird umso größer, je kleiner das Verhältnis der beiden Größen  $\beta$  und  $\gamma$  zueinander ist.

### 3.4.4 Adaptive Bounding Box

Das im vorigen Abschnitt vorgestellte Verfahren führt bei räumlich stark lokalisierten Teilchenverteilungen, wie in Abbildung 3.10 gezeigt, zu „Kommunikationshotspots“, welche die Performance erheblich verschlechtern können. Bei der Konstruktion des Verfahrens wurde angenommen, dass der Austausch der benötigten Feld- und Stromdaten pro Teilchen erfolgt, was dazu führt, dass für stark lokalisierte Teilchenverteilungen eine große Redundanz bei den ausgetauschten Daten entsteht.

Um dies zu vermeiden, wurde in [72, 73, 4] eine Parallelisierungsstrategie vorgeschlagen und in [74] untersucht, welche den Datenaustausch für diesen Fall reduzieren kann. Bei dieser Variante werden sowohl die Teilchen als auch die Gitterzellen so auf die Prozessoren verteilt, dass die Rechenlast ideal balanciert ist, d.h. es gilt

$$F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = F T_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \quad \forall i_\pi, j_\pi \in \Pi, \quad (3.81)$$

$$P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = P T_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \quad \forall i_\pi, j_\pi \in \Pi. \quad (3.82)$$

Nun wird in jedem Zeitschritt die kleinste Menge an Gitterzellen (Bounding Box)  $B_{\min}^{(m)}$  bestimmt, die alle Gitterzellen enthält, auf deren Feld- bzw. Stromwerte für die Trajektorienberechnung zugegriffen werden muss. Die Felddaten der Gitterzellen in diesem Bereich werden durch eine globale Kommunikationsoperation an alle Prozessoren verschickt. Die Stromdaten werden nach Abschluss der Berechnungen auf allen Prozessoren, wiederum mittels einer globalen Kommunikationsoperation, akkumuliert. In der im Rahmen der Arbeit durchgeführten Implementierung erfolgte der Datenaustausch in beiden Fällen mittels der Funktion `MPI_Allreduce`. Abbildung 3.11 illustriert den Anwendungsfall des Verfahrens.

Bei einer theoretischen Analyse stellt sich heraus, dass der Algorithmus nicht skalierbar ist. Die Skalierbarkeitsanalyse soll im Folgenden vorgestellt werden und erfolgt unter der Annahme eines Parallelrechners mit gleichartigen Prozessoren, das heißt es gilt  $\alpha_{i_\pi} = \alpha$  und  $\beta_{i_\pi} = \beta$ .

Da sowohl die Gitterzellen als auch die Teilchen gleichmäßig den verfügbaren Prozessoren zugeordnet werden, ergibt sich die reine Rechenzeit pro Zeitschritt für den Feldlöser bzw. für die Integration der Trajektorien zu

$${}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = \alpha \cdot \frac{N_c}{N_\pi} \quad {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = \beta \cdot \frac{P^{(m)}}{N_\pi} \quad \forall i_\pi \in \Pi. \quad (3.83)$$

Diese Werte sind aufgrund der idealen Lastbalancierung für jeden der beteiligten Prozessoren gleich. In jedem Zeitschritt müssen die Felddaten für diejenigen Zellen ausgetauscht werden, welche zu  $B_{\min}^{(m)}$  gehören. Somit lässt sich die Laufzeit für den benötigten Austausch der Felddaten einerseits, sowie der Stromdaten andererseits abschätzen zu

$${}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = \kappa |B_{\min}^{(m)}| \log(N_\pi) \quad \forall i_\pi \in \Pi, \quad (3.84)$$

wobei  $\kappa$  ein Proportionalitätsfaktor ist. Bei einer Punkt-zu-Punkt Kommunikationsoperation zwischen den Prozessoren würde  ${}^P \tilde{T}_S^{(m)}$  proportional zu  $N_\pi$  ansteigen. Durch die Verwendung der effizienteren globalen Kommunikationsoperation (`MPI_Allreduce`) kann dies auf den oben angegebenen Zusammenhang reduziert werden [47]. Als Abschätzung für den Term  ${}^F \tilde{T}_S^{(m)}$  folgt mit den Überlegungen aus Abschnitt 3.3.2

$${}^F \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = \zeta \cdot \left( \frac{N_c}{N_\pi} \right)^{\frac{2}{3}} \quad \forall i_\pi \in \Pi. \quad (3.85)$$

Es ergibt sich somit als Abschätzung für die benötigte Laufzeit in Zeitschritt  $m$

$$\begin{aligned} T_S^{(m)}(N_\pi, \mathcal{P}) &= {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^F \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \\ &= \alpha \cdot \frac{N_c}{N_\pi} + \zeta \cdot \left( \frac{N_c}{N_\pi} \right)^{\frac{2}{3}} + \beta \cdot \frac{P^{(m)}}{N_\pi} + \kappa |B_{\min}^{(m)}| \log(N_\pi). \end{aligned} \quad (3.86)$$

Für hinreichend große Simulationsprobleme  $N_c \gg N_\pi$  kann der zweite Summand gegenüber dem ersten vernachlässigt werden. Damit ergibt sich als Abschätzung für die Effizienz in Zeitschritt  $m$  gemäß der Definition (2.49)

$$E_S^{(m)}(N_\pi, \mathcal{P}) = \frac{T_S^{(m)}(1, \mathcal{P})}{N_\pi \cdot T_S^{(m)}(N_\pi, \mathcal{P})} \approx \frac{1}{1 + \frac{\kappa |B_{\min}^{(m)}| / N_c}{\alpha + \beta P^{(m)} / N_c} N_\pi \log(N_\pi)}. \quad (3.87)$$

Um mit Hilfe des Ausdrucks in Gleichung (3.87) die Isoeffizienzfunktion ableiten und somit die Skalierbarkeit untersuchen zu können, muss der Zusammenhang zwischen den Parametern  $N_c$ ,  $P^{(m)}$  sowie  $B_{\min}^{(m)}$  untersucht werden, um zu einem einzigen Parameter zu kommen, der die Problemgröße beschreibt.

Die Anzahl der Gitterzellen, für die Felddaten auszutauschen sind, ist bei einem homogenen Gitter proportional zur Anzahl der gesamten Gitterzellen. Es gilt

$$|B_{\min}^{(m)}| \propto N_c. \quad (3.88)$$

Bei einer sinnvollen Simulation wird die Teilchenanzahl  $P^{(m)}$  nicht unabhängig von der Anzahl der Gitterzellen verändert. Vielmehr wird bei einer besseren räumlichen Auflösung durch eine größere Anzahl an Gitterzellen auch eine größere Anzahl an Teilchen benötigt, so dass das Verhältnis der Anzahl an Teilchen zu der Anzahl an Gitterzellen ungefähr gleich bleibt. Es folgt

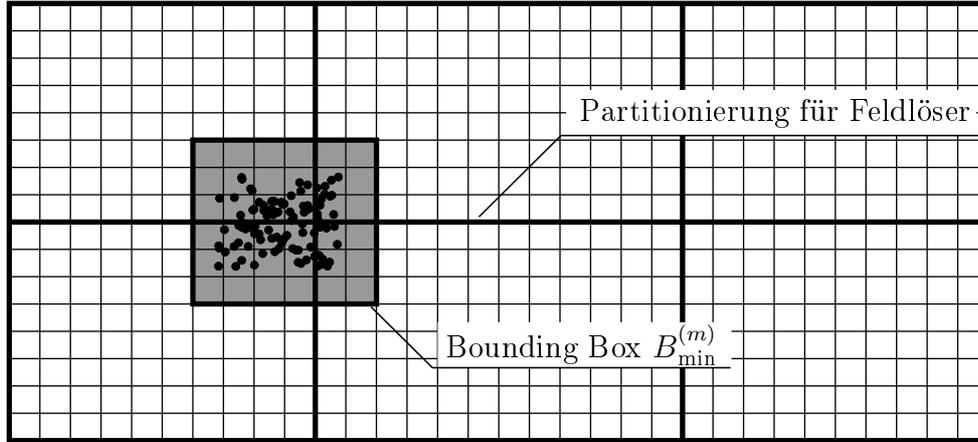
$$P^{(m)} \propto N_c. \quad (3.89)$$

Unter Voraussetzung der Gleichungen (3.88) und (3.89) wird der Ausdruck für die Effizienz in Gleichung (3.87) unabhängig von den Parametern  $N_c$ ,  $P^{(m)}$  sowie  $B_{\min}^{(m)}$ , welche die Problemgröße beschreiben. Somit kann durch eine Erhöhung der Problemgröße bei konstanter Anzahl an Prozessoren nicht erreicht werden, dass die Effizienz ansteigt. Damit ist diese Parallelisierungsstrategie nicht skalierbar und die Isoeffizienzfunktion existiert nicht. Trotz dieses negativen Ergebnisses kann die Strategie für Simulationsprobleme mit räumlich stark lokalisierter Teilchenverteilung recht gute Performanceergebnisse erreichen, wie in Kapitel 4 gezeigt werden wird.

### 3.4.5 Statische Gebietszerlegung

Interessant ist eine Betrachtung der sich mit dem in Abschnitt 3.4.3 beschriebenen Verfahren ergebenden Zuordnung der Teilchen für  $\gamma \ll \min_{i_\pi \in \Pi} \{\beta_{i_\pi}\}$  sowie  $\gamma > \max_{i_\pi \in \Pi} \{\beta_{i_\pi}\}$ .

Der erste Fall modelliert einen Parallelrechner mit einem Verbindungsnetzwerk sehr hoher Bandbreite, während der zweite Fall einen Parallelrechner mit einem Verbindungsnetzwerk



**Abbildung 3.11:** Illustriert das in Abschnitt 3.4.4 beschriebene Verfahren für  $N_\pi = 6$ . Die Felddaten innerhalb des grau schraffierten Bereiches (Bounding Box  $B_{\min}^{(m)}$ ) werden allen Prozessoren über eine globale Kommunikationsoperation bekannt gemacht. Für eine räumlich stark lokalisierte Teilchenverteilung kann dies zu einer Reduktion redundanter Kommunikation führen.

niedriger Bandbreite modelliert. Insbesondere für Parallelrechner aus der Klasse der Multicomputer wird tendenziell eher der zweite als der erste Fall auftreten.

Im ersten Fall kann  $P\tilde{T}_S^{(m)}$  gegenüber  $PT_S^{(m)}$  vernachlässigt werden, und es ist sofort ersichtlich, dass dies im Grenzfall  $\gamma = 0$  zu einer Zuordnung der Teilchen auf die Prozessoren führt, bei der  $PT_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) = PT_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \forall i_\pi, j_\pi \in \Pi$  gilt, bei der also die Rechenlast balanciert ist.

Im zweiten Fall ergibt sich wegen der Abbruchbedingung (3.60), dass das Verfahren zu einem Zustand führt, bei dem die Zuordnung der Teilchen direkt an die Zuordnung der Gitterzellen gekoppelt wird, denn es ergibt sich sofort, dass im resultierenden Zustand

$$a_{i_\pi, j_\pi}^{n_\pi} = \begin{cases} s_{i_\pi} & \text{falls } i_\pi = j_\pi \\ 0 & \text{sonst} \end{cases} \quad (3.90)$$

gilt. Somit wird die Zuordnung der Teilchen gemäß

$$a_{p, i_\pi}^{p_\pi(m)} = \sum_{\mathbf{i}_c \in \mathcal{I}} n_{\mathbf{i}_c, p}^{c_\pi(m)} \cdot a_{\mathbf{i}_c, i_\pi}^{c_\pi(m)} \quad (3.91)$$

vorgenommen und der Term  $P\tilde{T}_S^{(m)}$  in der Zielfunktion verschwindet, was sich durch Einsetzen von Gleichung (3.91) in Gleichung (3.35) leicht überprüfen lässt.

In den vergangenen Jahren wurden zahlreiche PIC Implementierungen vorgestellt, welche auf dem Prinzip einer statischen Gebietszerlegung und einer mit dieser gekoppelten Zuordnung der Teilchen gemäß Gleichung (3.91) basieren (siehe z.B. [6, 7, 5, 8]). Im Folgenden

soll durch eine Skalierbarkeitsanalyse gezeigt werden, dass diese Strategie für bestimmte Teilchenverteilungen problematisch sein kann. Der für die Strategie kritische Fall ist derjenige, bei dem sich ein großer Teil der Teilchen in Gitterzellen befindet, die dem gleichen Prozessor zugeordnet sind.

Bei der folgenden Skalierbarkeitsanalyse wird ein beliebiger Zeitschritt  $m$  betrachtet. In dem betrachteten Zeitschritt seien die Teilchen so im Rechengebiet verteilt, dass sie aufgrund von Gleichung (3.91) einem einzigen Prozessor zugewiesen werden und für keines der Teilchen während des Zeitschrittes die Prozessorzuordnung geändert werden muss. Da bei vielen Simulationsproblemen, insbesondere bei der Simulation der Teilchendynamik innerhalb von Beschleunigerstrukturen, wie z.B. der in Abschnitt 4.2 vorgestellten Anordnung, räumlich stark lokalisierte Teilchenverteilungen auftreten, die sich bei der beschriebenen Parallelisierungsstrategie tendenziell in wenigen oder tatsächlich nur einer einzigen Partition konzentrieren, tritt dieser Fall durchaus bei der Simulation praxisrelevanter Anordnungen in Erscheinung. Außerdem soll angenommen werden, dass die Berechnungen auf einem Parallelrechner mit gleichartigen Prozessoren vorgenommen und die Gitterzellen so den Prozessoren zugeordnet werden, dass

$$\max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^F \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} = \alpha \frac{N_c}{N_\pi} + \zeta \left( \frac{N_c}{N_\pi} \right)^{\frac{2}{3}}, \quad (3.92)$$

angenommen werden kann (vgl. Gleichung (3.29)), was einer gleichmäßigen Verteilung der Gitterzellen auf die Prozessoren mittels einer Partitionierung wie in Abschnitt 3.3.1 entspricht.

Da die Teilchen einem einzigen Prozessor zugewiesen sind und wegen  ${}^P \tilde{T}_S^{(m)} = 0$  folgt, dass

$$\max_{i_\pi \in \Pi} \left\{ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P \tilde{T}_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} = \beta P^{(m)} \quad (3.93)$$

gilt. Es ergibt sich als Abschätzung für die Laufzeit des betrachteten Zeitschrittes

$$T_S^{(m)}(N_\pi, \mathcal{P}) = \alpha \frac{N_c}{N_\pi} + \zeta \left( \frac{N_c}{N_\pi} \right)^{\frac{2}{3}} + \beta P^{(m)}. \quad (3.94)$$

Für Probleme mit  $N_c \gg N_\pi$  kann der zweite Summand in guter Näherung vernachlässigt werden und es folgt

$$T_S^{(m)}(N_\pi, \mathcal{P}) \approx \alpha \frac{N_c}{N_\pi} + \beta P. \quad (3.95)$$

Wie bereits bei der Skalierbarkeitsanalyse in Abschnitt 3.4.4 soll aus den dort erläuterten Gründen auch nun wieder die sinnvolle Annahme getroffen werden, dass die verwendete Anzahl an Makroteilchen mit der Anzahl an Gitterzellen verknüpft ist. Somit gilt mit einem Proportionalitätsfaktor  $k^{(m)}$

$$P^{(m)} = k^{(m)} N_c, \quad (3.96)$$

so dass die Problemgröße nun allein mit  $N_c$  beschrieben werden kann. Somit ergibt sich als Abschätzung für die Effizienz der Berechnungen in Zeitschritt  $m$

$$E^{(m)}(N_\pi, \mathcal{P}) = \frac{T_S^{(m)}(1, \mathcal{P})}{N_\pi \cdot T_S^{(m)}(N_\pi, \mathcal{P})} = \frac{\alpha + \beta k^{(m)}}{\alpha + \beta k^{(m)} N_\pi}. \quad (3.97)$$

Damit lässt sich ein Abfallen der Effizienz durch eine Erhöhung der Problemgröße nicht erreichen und die Parallelisierungsstrategie ist für den betrachteten Fall nicht skalierbar. Da die mit den Teilchen verbundenen Berechnungen einem einzigen Prozessor zugeordnet werden, wirken sich diese wie ein nicht parallelisierbarer Anteil auf die Laufzeit aus. Dies ist genau der Fall, für den in Abschnitt 2.4.3.2 das AMDAHLsche Gesetz hergeleitet wurde [50]. Folglich ergibt sich für den betrachteten Fall ein maximal erzielbarer Speedup  $S_\infty$  in Zeitschritt  $m$  von

$$S_\infty^{(m)} = 1 + \frac{\alpha}{k^{(m)} \beta}. \quad (3.98)$$

Abgesehen von der fehlenden Skalierbarkeit ist der betrachtete Fall noch aus einem anderen Grund kritisch.

Da insbesondere bei Multicomputern die Größe des jedem Prozessor zur Verfügung stehenden Arbeitsspeichers begrenzt ist, führt die betrachtete Situation, dass alle Teilchen einem einzigen Prozessor zugeordnet werden, bei entsprechender Problemgröße dazu, dass die Teilchendaten nicht mehr im Arbeitsspeicher gehalten werden können, was, abhängig vom verwendeten Parallelrechner und/oder Betriebssystem, entweder zu einem Auslagern der Daten auf die Festplatte, was bedingt durch die wesentlich höhere Zugriffszeit die Laufzeit stark ansteigen lässt, oder zu einem vollständigen Abbruch der Berechnungen führen kann.

### 3.4.6 Dynamische Gebietszerlegung

Parallelisierungsstrategien, welche die Zuordnung der Teilchen mit derjenigen der Gitterzellen gemäß Gleichung (3.91) koppeln, besitzen die besonders für Multicomputer mit einem Verbindungsnetzwerk niedriger Bandbreite attraktive Eigenschaft, dass alle zur Berechnung der Teilcentrajektorien benötigten Daten im lokalen Speicher verfügbar sind und somit der Term  ${}^P\tilde{T}_S^{(m)}$  in der Zielfunktion verschwindet. Durch die sich während der Laufzeit ändernde Verteilung der Teilchen im Rechengbiet können jedoch die in Abschnitt 3.4.5 erläuterten Probleme entstehen. Im Folgenden sollen zwei im Rahmen dieser Arbeit untersuchte Parallelisierungsstrategien vorgestellt werden, die diese Probleme durch eine dynamische Neuordnung der Gitterzellen und den mit diesen verbundenen Teilchen

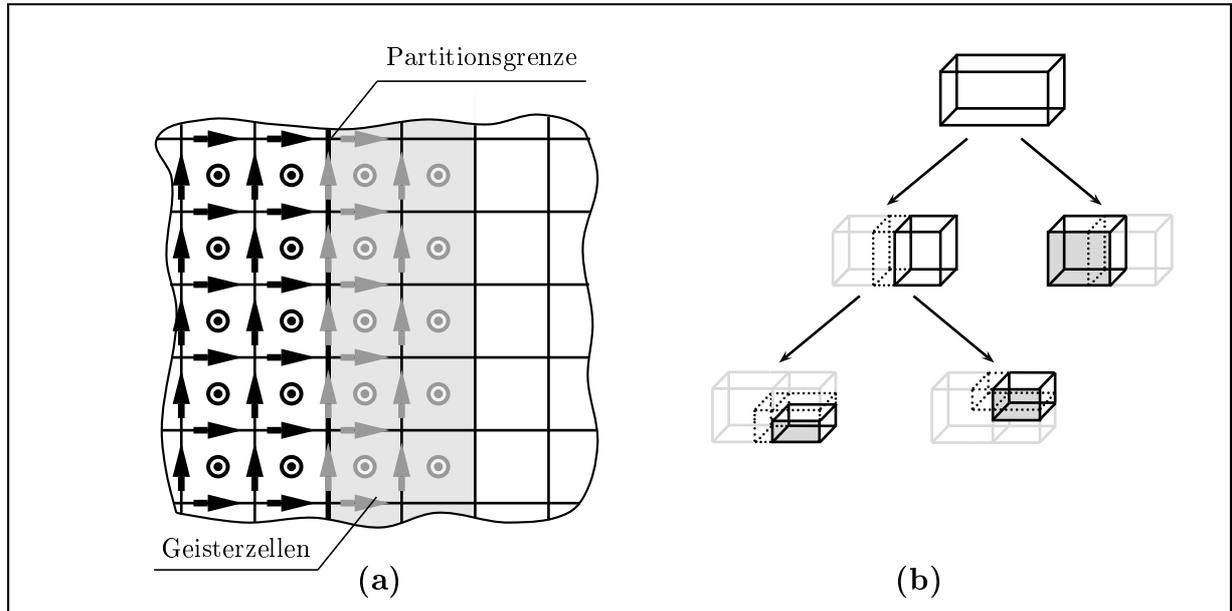
abschwächen können. Die erste der beiden Strategien basiert auf einem von CAMPBELL, CARMONA und WALKER vorgeschlagenen Verfahren [9], welches in [10, 11, 12, 75] zur Parallelisierung verschiedener PIC Codes eingesetzt wurde. Es basiert auf dem in Abschnitt 3.3.1 zur Parallelisierung des Feldlösers beschriebenen Verfahren der Rekursiven Koordinaten Bisektionierung [66]. Die Formulierung erfolgte bislang lediglich für Parallelrechner mit gleichartigen Prozessoren. Es soll hier in verallgemeinerter Form vorgestellt werden, die eine Implementierung für Parallelrechner mit Prozessoren unterschiedlicher Leistungsfähigkeit erlaubt.

Bei der zweiten Strategie handelt es sich um ein im Rahmen der Arbeit entwickeltes Verfahren, welches neben der dynamischen Balancierung der Rechenlast zwischen den Prozessoren durch die geschickte Anordnung der Daten im Speicher die Speicherhierarchie (Caches) moderner Rechner ausnutzen kann, zum Preis eines etwas höheren Aufwandes für den Datenaustausch. Zudem ermöglicht es eine Diskretisierung des Rechengebietes, welche besser auf das Simulationsproblem zugeschnitten werden kann und dadurch weniger Rechnerressourcen benötigt

#### 3.4.6.1 Rekursive Koordinaten Bisektionierung

Zunächst soll erläutert werden, wie durch das Duplizieren eines Teiles der Rechenlast für den Feldlöser erreicht werden kann, dass der Synchronisationspunkt zwischen der Feldlöserphase und der Trajektorienberechnung fortgelassen werden, und der Datenaustausch nach Abschluss aller Berechnungen eines Zeitschrittes erfolgen kann [12]. Dies ermöglicht die Formulierung der Zielfunktion in der in Gleichung (3.38) angegebenen Form.

Um den Datenaustausch für den Feldlöser erst nach Abschluss der Trajektorienberechnung durchführen zu können, muss gewährleistet sein, dass sämtliche Feldwerte, auf die bei der Berechnung der Trajektorien sowie bei der Zeitintegration der Gitterspannungen zugegriffen werden muss, im lokalen Speicher des entsprechenden Prozessors verfügbar sind. Im Falle der verwendeten trilinearen Interpolation bedeutet dies, dass für jede Gitterzelle, die einem Prozessor mitsamt der in ihr enthaltenen Teilchen zugeordnet wurde, die Felddaten derjenigen Zellen lokal verfügbar sein müssen, die an die Zelle angrenzen (vgl. Abbildung 3.5(a)). Um die aktuellen Werte dieser Felddaten zu besitzen, müssen diese entweder nach der Feldlöserphase zwischen den Prozessoren ausgetauscht werden, was die unerwünschte Synchronisation der Prozessoren zur Folge hätte, oder sie müssen lokal berechnet werden. Zu deren lokaler Berechnung werden wiederum die Felddaten weiterer Zellen gemäß der in Abschnitt 3.3 analysierten Abhängigkeiten benötigt. Für die durch eine RCB erstellte Partitionierung kann durch eine Erweiterung der Gebiete mit einer zwei Zellen dicken Schicht von Geisterzellen, deren Felddaten zu Beginn jedes Zeitschrittes von dem



**Abbildung 3.12:** (a) Die Gebiete werden um eine zwei Zellen dicke „Schicht von Geisterzellen“ erweitert. Zu Beginn eines Zeitschrittes werden die den Geisterzellen zugeordneten Felddaten von den entsprechenden Prozessoren geholt. Dies ermöglicht die lokale Berechnung aller Felddaten, welche für die folgende Trajektorienberechnung benötigt werden. (b) illustriert das rekursive Verschieben der Gebietsgrenzen im Rahmen einer dynamischen Lastbalancierung.

entsprechenden Prozessor geholt werden müssen, erreicht werden, dass alle zur lokalen Berechnung der bei der Interpolation benötigten Feldwerte lokal vorhanden sind. Abbildung 3.12 illustriert, für eine einfache Partitionsgrenze, die Felddaten welcher Zellen für die mit der linken Partition verbundenen Berechnungen ausgetauscht werden müssen.

Wie bereits bei dem in Abschnitt 3.4.3 vorgestellten Verfahren soll der Aufwand für die Neuordnung der Teilchen und Gitterzellen in der Zielfunktion nicht explizit berücksichtigt werden. Stattdessen wird die Neuordnung basierend auf der bereits bestehenden Zuordnung durch inkrementelle Veränderungen derselben vorgenommen, und so der Aufwand für den mit der Lastbalancierung verbundenen Datenaustausch gering gehalten. Unter der Annahme  $N_c \gg N_\pi$  soll der Term  ${}^F\tilde{T}_S^v$  vernachlässigt werden. Unter Berücksichtigung dieser Vereinfachungen lautet die Zielfunktion

$$T_S(N_\pi, \mathcal{P}) = \sum_{m=1}^M \max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\}. \quad (3.99)$$

Durch den Wegfall des Terms  ${}^{LB}\tilde{T}_S^{(m)}$  kann jeder Zeitschritt entkoppelt von den restlichen betrachtet werden.

Die den einzelnen Prozessoren in Zeitschritt  $m$  zugeordneten Gebiete seien mit  $B_{i_\pi}^{(m)}$  bezeichnet. Die Zuordnung der Gitterzellen und Teilchen wird durch die Wahl der Gebiete festgelegt gemäß

$$a_{\mathbf{i}_c, i_\pi}^{c\pi(m)} = \begin{cases} 1 & \text{falls } \mathbf{i}_c \in B_{i_\pi}^{(m)} \\ 0 & \text{sonst} \end{cases} \quad (3.100)$$

sowie

$$a_{p, i_\pi}^{p\pi(m)} = \sum_{\mathbf{i}_c \in B_{i_\pi}^{(m)}} n_{\mathbf{i}_c, p}^{cp(m)} \cdot a_{\mathbf{i}_c, i_\pi}^{c\pi(m)}. \quad (3.101)$$

Die Gesamtanzahl der Teilchen, welche Prozessor  $i_\pi$  durch die Wahl der Partitionierung zugeordnet werden, ergibt sich gemäß

$$P_{i_\pi}^{(m)} := \sum_{p=1}^{P(m)} a_{p, i_\pi}^{p\pi(m)} = \sum_{p=1}^{P(m)} \sum_{\mathbf{i}_c \in B_{i_\pi}^{(m)}} n_{\mathbf{i}_c, p}^{cp(m)} \cdot a_{\mathbf{i}_c, i_\pi}^{c\pi(m)}. \quad (3.102)$$

Der Wert des  $m$ -ten Summanden in der Zielfunktion ergibt sich somit unter Berücksichtigung von Gleichung (3.102) sowie dem Zusammenhang

$$|B_{i_\pi}^{(m)}| = \sum_{\mathbf{i}_c \in \mathcal{I}} a_{\mathbf{i}_c, i_\pi}^{c\pi(m)} \quad (3.103)$$

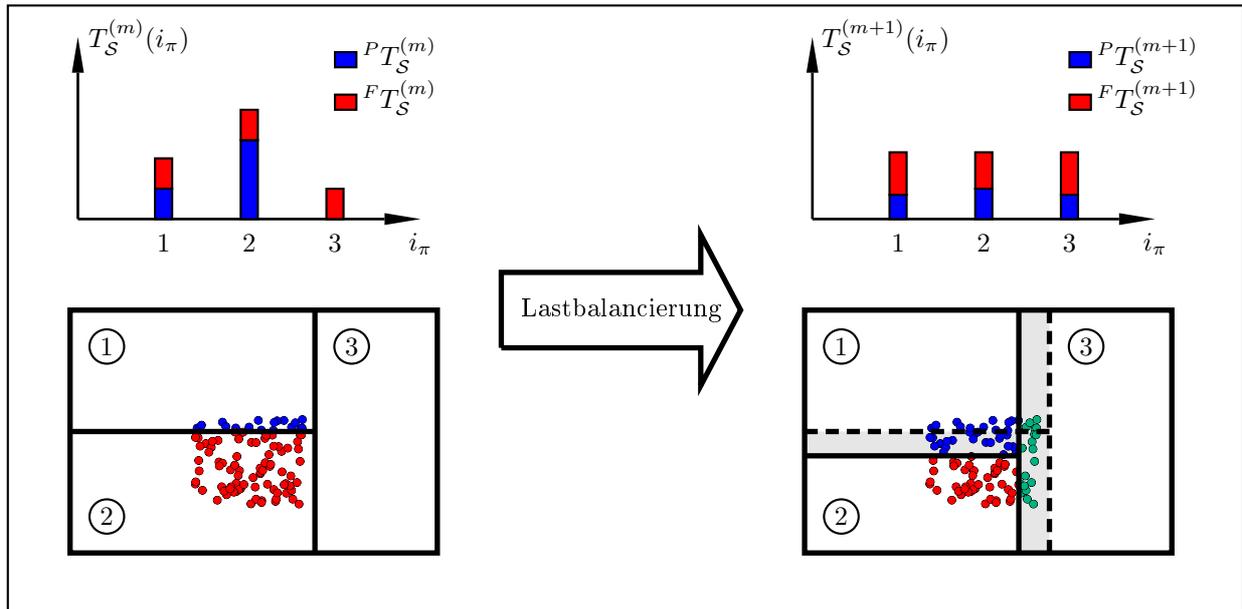
zu

$$\begin{aligned} \max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} &= \max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} |B_{i_\pi}^{(m)}| + \beta_{i_\pi} P_{i_\pi}^{(m)} \right\} \\ &= \max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} \cdot \underbrace{\left( |B_{i_\pi}^{(m)}| + \frac{\beta_{i_\pi}}{\alpha_{i_\pi}} P_{i_\pi}^{(m)} \right)}_{=: W_{i_\pi}^{(m)}} \right\} \\ &= \max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} \cdot W_{i_\pi}^{(m)} \right\}. \end{aligned} \quad (3.104)$$

Es sei angemerkt, dass der Ausdruck  $\frac{\beta_{i_\pi}}{\alpha_{i_\pi}}$  als das Verhältnis der Anzahl der Berechnungsoperationen, welche für ein Teilchen durchzuführen sind zur Anzahl der Berechnungsoperationen, die für die mit einer Gitterzelle verknüpften Gitterspannungen ausgeführt werden müssen, interpretiert werden kann. Die Leistungsfähigkeit der Prozessoren wird also durch die  $\alpha_{i_\pi}$  modelliert, während die  $W_{i_\pi}^{(m)}$  eine davon unabhängige Beschreibung der Rechenlast bilden.

Der Ausdruck in Gleichung (3.104) nimmt sein Minimum an, wenn

$$\alpha_{i_\pi} \cdot W_{i_\pi}^{(m)} = \alpha_{j_\pi} \cdot W_{j_\pi}^{(m)} \quad \forall i_\pi, j_\pi \in \Pi. \quad (3.105)$$



**Abbildung 3.13:** Durch das Verschieben der Partitionierungsebenen wird die Lastbalancierung der Berechnungen durchgeführt. Die Daten, welche dem grau hinterlegten Bereich im rechten Bild zugeordnet sind, müssen im Rahmen der Lastbalancierung zwischen den Prozessoren ausgetauscht werden.

gilt. Wie bereits bei der Parallelisierung des Feldlösers soll zunächst wieder der Zweiprozessorfall betrachtet werden. Die Verallgemeinerung auf den Fall eines Parallelrechners mit einer beliebigen Anzahl an Prozessoren durch Rekursion ist dann ohne Schwierigkeiten möglich.

Im Zweiprozessorfall reduziert sich Bedingung (3.105) auf

$$\alpha_1 \cdot W_1^{(m)} = \alpha_2 \cdot W_2^{(m)}. \quad (3.106)$$

Ist diese Bedingung durch die bestehende Partitionierung nicht erfüllt, so wird die Partitionierungsebene um eine Gitterzelle verschoben, so dass die Partition mit zu großer Last verkleinert wird zugunsten der Partition mit zu kleiner Last. Dies wird so lange wiederholt, bis sich eine Partitionierung ergibt, die Gleichung (3.106) erfüllt, oder zumindest die bestmögliche Approximation an (3.106) liefert, welche durch die RCB Partitionierung erreicht werden kann.

Das Verfahren kann nun durch Rekursion auf den allgemeinen Fall mit einer beliebigen Anzahl an Prozessoren erweitert werden. Im ersten Rekursionsschritt muss die erste Par-

tionierungsebene so verschoben werden, dass

$$\boxed{\alpha_a \cdot W_a^{(m)} = \alpha_b \cdot W_b^{(m)}} \quad (3.107)$$

gilt, wobei sich die  $\alpha_a$  bzw.  $\alpha_b$  gemäß den Gleichungen unter (3.24) bestimmen lassen. Die Zuordnung der Prozessoren zu den beiden Mengen  $\Pi_a$  und  $\Pi_b$  wird gemäß der bestehenden Partitionierung übernommen. In jedem weiteren Rekursionsschritt wird die bestehende Partitionierungsebene wie beschrieben verschoben, so dass für die entstehende neue Partitionierung schließlich die Last so verteilt ist, dass Gleichung (3.105) zumindest näherungsweise erfüllt wird.

Dadurch, dass die bestehenden Partitionierungsebenen lediglich inkrementell verändert werden, verursacht das Verfahren einen recht geringen Datenaustausch. Abbildung 3.13 illustriert beispielhaft für einen Parallelrechner mit drei Prozessoren das Verschieben der Partitionierungsebenen und die resultierende Balancierung der Rechenlast. Man beachte, dass im resultierenden Zustand die Gesamtrechenlast ( ${}^F T_{\mathcal{S}}^{(m)}(i_\pi, N_\pi, \mathcal{P}) + {}^P T_{\mathcal{S}}^{(m)}(i_\pi, N_\pi, \mathcal{P})$ ), nicht jedoch die Rechenlast jeder einzelnen Phase des PIC Algorithmus zwischen den Prozessoren balanciert ist.

Trotz der relativ geringen Datenmenge, die im Rahmen der Lastbalancierungsoperation ausgetauscht werden muss, zeigt sich, dass eine Lastbalancierung in jedem Zeitschritt nicht vorteilhaft ist, da die Operation immer auch mit nicht unerheblichen lokalen Berechnungen im Rahmen der Bestimmung der neuen Partitionierung, sowie mit Aufwand für die anschließende Neuorganisation der lokalen Datenstrukturen verbunden ist [68]. Daher wurde in [68, 9] vorgeschlagen, die Entscheidung, ob eine Lastbalancierung in einem Zeitschritt durchgeführt werden soll oder nicht, von einem Parameter abhängig zu machen, der ein Maß dafür bildet, wie stark die Verteilung der Last von der optimalen Verteilung gemäß Gleichung (3.105) abweicht. Im Rahmen dieser Arbeit wurde

$$\boxed{\sigma^{(m)} := \frac{\max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} \cdot W_{i_\pi}^{(m)} \right\} - \bar{\alpha} \cdot W_{\text{ges}}^{(m)}}{\bar{\alpha} \cdot W_{\text{ges}}^{(m)}}} \quad (3.108)$$

mit

$$\bar{\alpha} := \frac{1}{\sum_{i_\pi=1}^{N_\pi} \frac{1}{\alpha_{i_\pi}}} \quad \text{und} \quad W_{\text{ges}}^{(m)} := \sum_{i_\pi=1}^{N_\pi} W_{i_\pi}^{(m)} \quad (3.109)$$

als Entscheidungsgröße herangezogen. Die Lastbalancierungsoperation wird eingeleitet, sobald der Entscheidungsparameter einen vorher festgelegten Schwellwert  $\sigma_{\text{max}}$  überschreitet.

Im Falle gleichartiger Prozessoren ist die Größe in (3.108) äquivalent zu den in [68, 9] vorgeschlagenen Entscheidungsparametern. Der Entscheidungsparameter  $\sigma^{(m)}$  ist dann direkt verknüpft mit der Effizienz  $E^{(m)}$  und es gilt

$$E^{(m)} = \frac{1}{1 + \sigma^{(m)}}. \quad (3.110)$$

Die Vorgabe eines bestimmten Wertes von  $\sigma_{\max}$  ist also äquivalent zur Vorgabe eines Mindestwertes für die Effizienz der Berechnung.

### 3.4.6.2 Kombinatorische Gebietszerlegung

Die folgende, im Rahmen der Arbeit entwickelte Parallelisierungsstrategie ermöglicht einerseits eine Reihe von Laufzeitverbesserungen, führt aber andererseits zu einer etwas größeren Menge an Daten, die in jedem Zeitschritt zwischen den Prozessoren ausgetauscht werden müssen.

Durch die gekoppelte Zuordnung von Teilchen und Gitterzellen sind die zur Interpolation der Feldwerte und Extrapolation der Ströme während der Trajektorienberechnung benötigten Speicherzugriffe allesamt lokal, wodurch die Zeit für diese Speicherzugriffe reduziert wird. Jedoch wurde kein Augenmerk darauf gelegt, in welcher Reihenfolge diese lokalen Speicherzugriffe erfolgen. Ein Zugriff auf den Arbeitsspeicher eines Rechners dauert üblicherweise mehrere CPU Zyklen, in denen die Verarbeitung angehalten werden muss, bis die benötigten Daten dem Prozessor zur Verfügung stehen. Zur Verringerung der Zugriffszeit existieren schnelle Zwischenspeicher (Caches), die zwar deutlich kleiner als der Arbeitsspeicher sind, jedoch im Vergleich zu diesem eine deutlich geringere Zugriffszeit aufweisen. Diese Zwischenspeicher arbeiten nach dem Lokalitätsprinzip, d.h. bei einem Speicherzugriff wird nicht nur der tatsächlich benötigte Wert in den schnellen Cache Speicher geladen und anschließend zur Verarbeitung zur CPU transportiert, sondern es werden auch Werte, die im Hauptspeicher benachbart zu diesem Wert liegen, in den Cache Speicher übernommen. Dies geschieht in der Erwartung, dass als nächstes auf einen dieser benachbarten Werte zugegriffen werden muss [32]. Ist dies der Fall, so kann die Zugriffszeit ganz erheblich verkürzt werden, da der Wert direkt aus dem schnellen Cache Speicher geholt werden kann. Dies impliziert, dass ein zufälliger Zugriff auf Daten im Hauptspeicher den Cache nur schlecht ausnutzt.

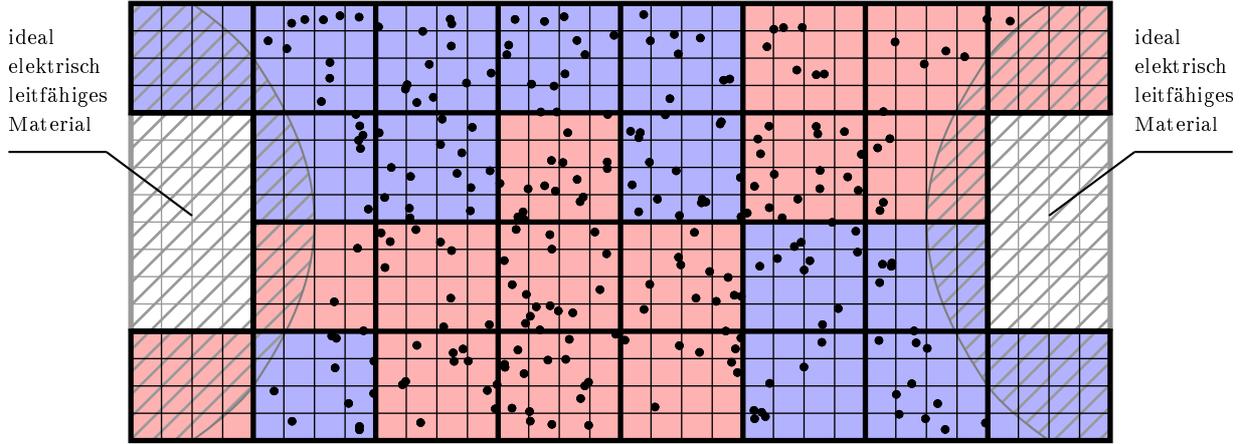
Insbesondere die Speicherzugriffe zur Feldinterpolation und Stromextrapolation bieten bei PIC im Hinblick auf die effiziente Nutzung des Cache Speichers Optimierungspotential. In [76] findet sich eine Untersuchung darüber, wie groß der Einfluss einer effizienten Ausnutzung des Caches auf die Laufzeit der PIC Schleife ist. Für die betrachtete Problemstellung (Plasma Simulation) erhielt der Autor auf dem von ihm verwendeten Rechner eine

Laufzeitverbesserung um 40-70% dadurch, dass er die Teilchen nach den Gitterzellen, in denen sie sich befinden, im Speicher gruppierte. Während die im letzten Abschnitt beschriebene Strategie durchaus um eine derartige Sortierung der Teilchen ergänzt werden und somit ebenfalls von dem Potential des Caches profitieren kann, ist eine solche Sortierung bei dem im Folgenden beschriebenen Verfahren bereits inhärent vorhanden.

Eine weitere Optimierungsmöglichkeit bietet die flexiblere Wahl der Form des Rechengebietes. Häufig sind Anordnungen zu simulieren, bei denen das interessierende Rechengebiet durch Körper begrenzt wird, welche als ideal elektrische Leiter modelliert werden können. Insbesondere bei Problemstellungen aus der Beschleunigerphysik tritt dieser Fall in Erscheinung. Ein Rechengitter, dessen Konstruktion nach dem einfachen in Abschnitt 2.2.1 erläuterten Prinzip erfolgt, überdeckt immer einen quaderförmigen Raumbereich, was zur Folge hat, dass häufig ein großer Teil der Gitterzellen innerhalb der ideal leitfähigen Körper liegen. Die Speicherung der Feldfreiheitsgrade für diese Zellen sowie deren Traversierung im Rahmen der Zeitintegration ist daher eine Verschwendung von Speicherplatz bzw. Rechenzeit. Durch eine flexiblere Diskretisierung, wie im Folgenden vorgestellt, kann dies zum Teil vermieden werden. Eine solche flexiblere Methode der Diskretisierung kommt auch in anderen Anwendungsfeldern der numerischen Simulation zum Einsatz (siehe z.B. [77]).

Die Grundidee der im Folgenden beschriebenen Parallelisierungsstrategie ist die Zerlegung des Rechengebietes in eine Anzahl an Partitionen, die deutlich größer als die Anzahl der Prozessoren ist, welche für die Simulation zum Einsatz kommen sollen. Jede der Partitionen bildet eine separate Datenstruktur, in der die ihr zugeordneten Feld- und Teilchendaten gespeichert werden. Die Partitionen bilden die kleinste Einheit, in der die Feld- und Teilchendaten sowie die mit diesen verknüpften Berechnungen den verfügbaren Prozessoren zugeordnet werden. Eine dynamische Lastbalancierung erfolgt dann durch den Austausch dieser Partitionen zwischen den Prozessoren. Abbildung 3.14 illustriert, wie eine resultierende Partitionierung für  $N_\pi = 2$  aussehen kann. Die farbliche Markierung deutet die Prozessorzuordnung der einzelnen Gebiete in einem bestimmten Zeitschritt  $m$  an. Die vier weiß hinterlegten Partitionen liegen vollständig innerhalb eines als ideal elektrisch leitfähig modellierten Materials. Diese Partitionen können nach Feststellen dieser Tatsache gelöscht werden und spielen bei den folgenden Überlegungen keine Rolle mehr. Eine derartige Form der Partitionierung wurde in [72, 73, 4], im Kontext einer Parallelisierungsstrategie wie in Abschnitt 3.4.4 vorgeschlagen, was jedoch, wie bereits erläutert, zu einem nicht skalierbaren Algorithmus führt, während der im Folgenden erläuterte Algorithmus skalierbar ist.

Das Rechengebiet sei partitioniert in  $G$  Partitionen  $B_1 \dots B_G$ , so dass die Anzahl der Gitterzellen, welche in jeder dieser Partitionen enthalten sind, gleich ist. Durch die Beschränkung, welche durch die Form der Gebiete auferlegt wird, ist diese Bedingung im Allgemeinen nur näherungsweise zu erfüllen. Bezeichne  $K_{i_g}^{(m)}$  die Anzahl der Teilchen,



**Abbildung 3.14:** Illustriert eine mögliche Partitionierung für den Fall  $N_\pi = 2$ . Die farbliche Markierung zeigt die Prozessorzugehörigkeit einer Partition an. Die in weiß hinterlegten Partitionen liegen vollständig innerhalb eines als ideal elektrisch leitfähig modellierten Materials. Eine Diskretisierung dieser Bereiche kann daher unterbleiben und die entsprechenden Partitionen können gelöscht werden.

welche sich in Zeitschritt  $m$  innerhalb der Partition  $B_{i_g}$  befinden:

$$K_{i_g}^{(m)} := \sum_{i_c \in B_{i_g}} \sum_{p=1}^{P(m)} n_{i_c, p}^{\text{cp}(m)}. \quad (3.111)$$

Eine Zuweisung der Gebiete zu den verfügbaren  $N_\pi$  Prozessoren werde durch Entscheidungsvariablen  $a_{i_g, i_\pi}^{g_\pi(m)}$  ausgedrückt, wobei

$$a_{i_g, i_\pi}^{g_\pi(m)} := \begin{cases} 1 & \text{falls } B_{i_g} \text{ im Zeitschritt } m \text{ Prozessor } i_\pi \text{ zugeordnet wird,} \\ 0 & \text{sonst.} \end{cases} \quad (3.112)$$

gilt. Die Parallelisierungsstrategie ist in diesem Zusammenhang ein Algorithmus, welcher die Werte der  $a_{i_g, i_\pi}^{g_\pi(m)}$  festlegt. Die Zuordnung muss eindeutig sein, d.h.

$$\sum_{i_\pi=1}^{N_\pi} a_{i_g, i_\pi}^{g_\pi(m)} = 1 \quad \forall i_g \in \{1 \dots G\}, m \in \{1 \dots M\}. \quad (3.113)$$

Zunächst soll die Zielfunktion in Abhängigkeit der  $a_{i_g, i_\pi}^{g_\pi(m)}$  formuliert werden. Durch die vorgegebenen Gebiete ist der Kommunikationsaufwand  $F\tilde{T}_S^{(m)}$  bereits bestimmt und nicht mehr zu beeinflussen. Dieser Term wird daher in der Zielfunktion nicht mehr berücksichtigt.

Der Aufwand für die Lastbalancierungsoperation soll wieder durch die Konstruktion des Lastbalancierungsverfahrens möglichst gering gehalten werden, indem eine neue Zuordnung der Partitionen zu den Prozessoren durch eine inkrementelle Veränderung der alten Partitionierung konstruiert wird. Eine Lastbalancierung wird nicht in jedem Zeitschritt durchgeführt, sondern nur, wenn die Lastverteilung als ungleichmäßig erkannt wurde, wobei als Entscheidungsgröße wieder der in Gleichung (3.108) eingeführte Parameter  $\sigma^{(m)}$  verwendet wurde. Unter diesen Voraussetzungen lautet die Zielfunktion

$$\begin{aligned}
 T_S(N_\pi, \mathcal{P}) &= \sum_{m=1}^M \left[ \max_{i_\pi \in \Pi} \left\{ {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} + \max_{i_\pi \in \Pi} \left\{ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) \right\} \right] \\
 &= \sum_{m=1}^M \left[ \max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} \sum_{i_g=1}^G a_{i_g, i_\pi}^{g_\pi^{(m)}} \cdot |B_{i_g}| \right\} + \max_{i_\pi \in \Pi} \left\{ \alpha_{i_\pi} \cdot \underbrace{\frac{\beta_{i_\pi}}{\alpha_{i_\pi}}}_{=: \psi} \sum_{i_g=1}^G a_{i_g, i_\pi}^{g_\pi^{(m)}} \cdot K_{i_g}^{(m)} \right\} \right].
 \end{aligned}
 \tag{3.114}$$

Der Faktor  $\beta_{i_\pi}/\alpha_{i_\pi}$  ist, wie in Abschnitt 3.4.6.1 erläutert, für alle Prozessoren gleich, womit die Größe  $\alpha_{i_\pi}$  das alleinige Maß für die Leistungsfähigkeit der einzelnen Prozessoren ist. Da alle Partitionen  $B_1 \dots B_G$  (zumindest näherungsweise) die gleiche Anzahl an Gitterzellen beinhalten, folgt

$$|B_{i_g}| = |B_{j_g}| \quad \forall i_g, j_g \in \{1 \dots G\}.$$
(3.115)

Die Laufzeit für den Feldlöser lässt sich somit minimieren, indem jedem Prozessor  $i_\pi$  eine Anzahl an Gebieten  $G_{i_\pi}$  zugeordnet wird, so dass gilt (vgl. Gleichung (3.6))

$$\frac{G_{i_\pi}}{G} = \frac{\frac{1}{\alpha_{i_\pi}}}{\sum_{j_\pi=1}^{N_\pi} \frac{1}{\alpha_{j_\pi}}}.$$
(3.116)

Die endliche Anzahl an verfügbaren Partitionen führt dazu, dass sowohl Gleichung (3.115) als auch Gleichung (3.116) im Allgemeinen nur näherungsweise erfüllt werden können, was bei einer großen Anzahl an Gebieten oder einem Parallelrechner mit Prozessoren von etwa gleicher Leistungsfähigkeit jedoch kaum ins Gewicht fällt. Sind Gleichung (3.115)

und (3.116) erfüllt, so folgt für die Laufzeit des Feldlösers

$$\begin{aligned}
 {}^F T_{\mathcal{S}}^{(m)}(N_{\pi}, \mathcal{P}) &= \max_{i_{\pi} \in \Pi} \left\{ \alpha_{i_{\pi}} \cdot G_{i_{\pi}} \cdot |B_1| \right\} \\
 &= \max_{i_{\pi} \in \Pi} \left\{ \frac{G \cdot |B_1|}{\sum_{j_{\pi}=1}^{N_{\pi}} \frac{1}{\alpha_{j_{\pi}}}} \right\} \\
 &=: {}^F T_0,
 \end{aligned} \tag{3.117}$$

und nimmt somit seinen optimalen Wert an.

Der Lastbalancierungsalgorithmus soll durch einen Austausch der Gebiete zwischen den Prozessoren die für die Berechnung der Trajektorien benötigte Laufzeit verbessern, ohne die für den Feldlöser benötigte Laufzeit zu erhöhen. Die Anzahl der jedem Prozessor zugewiesenen Gebiete  $G_{i_{\pi}}$  darf sich somit während der Simulation nicht ändern:

$$\sum_{i_g=1}^G a_{i_g, i_{\pi}}^{g_{\pi}(m)} = \sum_{i_g=1}^G a_{i_g, i_{\pi}}^{g_{\pi}(m+1)} = G_{i_{\pi}} \quad \forall m \in \{1 \dots M\}, i_{\pi} \in \Pi. \tag{3.118}$$

Zunächst soll der Lastbalancierungsalgorithmus für den Spezialfall, dass genau ein Prozessor  $o$  existiert, für den

$$P_o^{(m)} > \frac{\frac{1}{\alpha_o}}{\sum_{j_{\pi}=1}^{N_{\pi}} \frac{1}{\alpha_{j_{\pi}}}} \cdot P^{(m)} \tag{3.119}$$

gilt, der also überlastet ist, behandelt werden. Alle anderen Prozessoren seien unterbelastet. Die Berechnung der neuen Gebietszuordnung  $a_{i_g, i_{\pi}}^{g_{\pi}(m+1)}$  erfolgt iterativ in maximal  $G_o$  Iterationen, wobei  $G_o$  die Anzahl der Prozessor  $o$  zugeordneten Gebiete bezeichnet. Das Verfahren basiert auf dem in der kombinatorischen Optimierung wohl bekannten Greedy-Prinzip [64].

Bezeichne  $\mathcal{C}_{i_g, i_{\pi}}^{g_{\pi}(n)}$  die Zuordnung der Gebiete im  $n$ -ten Iterationsschritt des Verfahrens, und  $u_1^{(n)} \dots u_{N_{\pi}-1}^{(n)} \in \Pi$  die aufsteigend nach der für die Trajektorienberechnung benötigten Laufzeit sortierte Liste der unterbelasteten Prozessoren im  $n$ -ten Iterationsschritt des Lastbalancierungsverfahrens.

Seien weiterhin  $l_{i_{\pi}} : \{1 \dots G_{i_{\pi}}\} \rightarrow \{1 \dots G\}$  Abbildungen, welche die Gebiete, die einem beliebigen Prozessor  $i_{\pi}$  zu Beginn des Lastbalancierungsverfahrens zugeordnet sind, aufsteigend nach der Anzahl an Teilchen, die diese Gebiete enthalten, durchlaufen. Es gilt also

$$K_{l_{i_{\pi}}(\kappa_1)}^{(m)} \leq K_{l_{i_{\pi}}(\kappa_2)}^{(m)} \quad \forall \kappa_1 < \kappa_2. \tag{3.120}$$

Da der Zeitschritt  $m$ , in dem die Lastbalancierung durchgeführt wird, keinen Einfluss auf den Ablauf des Lastbalancierungsverfahrens hat, wird der hochgestellte Zeitschrittindex

im Folgenden bei allen Größen fortgelassen. Bei den Größen, die mit einem hochgestellten Index ausgestattet sind, bezeichnet dieser den Iterationsschritt des Lastbalancierungsverfahrens.

**Initialisierungsschritt:**

Setze

$$c_{i_g, i_\pi}^{g^\pi(0)} = a_{i_g, i_\pi}, \quad (3.121)$$

womit der Ausgangszustand für die Lastbalancierung gegeben ist durch die vorliegende Gebietszuordnung. Setze

$$i_o^{(0)} = G_o \quad (3.122)$$

sowie

$$i_{u_1}^{(0)} = \dots = i_{u_{N_\pi-1}}^{(0)} = 1 \quad (3.123)$$

und

$$\kappa = 1. \quad (3.124)$$

**Iterationsschritt:**

**Schritt 1: (Prüfen der Abbruchbedingungen)**

Falls

$$\frac{\alpha_o \cdot \psi \cdot \sum_{i_g=1}^G c_{i_g, o}^{g^\pi(n)} \cdot K_{i_g} - \bar{\alpha} \cdot \psi \cdot P}{\bar{\alpha} \cdot \psi \cdot P + FT_0} < \sigma_{\max} \quad (3.125)$$

oder

$$i_o^{(n)} = 0 \quad (3.126)$$

gilt, so wird das Verfahren beendet und die  $c_{i_g, i_\pi}^{g^\pi(n)}$  enthalten die neue Zuordnung der Gebiete. Im Fall, dass Gleichung (3.125) erfüllt ist, wurde das Lastbalancierungsziel, die Überlastung des Prozessors  $o$  unter die vorgegebene Schwelle  $\sigma_{\max}$  zu reduzieren, erreicht. Im Fall, dass Gleichung (3.126) erfüllt ist, konnte dieses Ziel nicht erreicht werden.

**Schritt 2: (Vergleich/Austausch der Gebiete)**

Falls

$$K_{l_o(i_o^{(n)})} > K_{l_{u_\kappa}^{(n)}(i_{u_\kappa}^{(n)})} \quad (3.127)$$

und

$$\frac{\alpha_{u_\kappa}^{(n)} \cdot \psi \cdot \left( \sum_{i_g=1}^G c_{i_g, u_\kappa}^{g^\pi(n)} \cdot K_{i_g} + K_{l_o(i_o^{(n)})} - K_{l_{u_\kappa}^{(n)}(i_{u_\kappa}^{(n)})} \right) - \bar{\alpha} \cdot \psi \cdot P}{\bar{\alpha} \cdot \psi \cdot P + FT_0} < \sigma_{\max} \quad (3.128)$$

gilt, vertausche die Prozessorzuordnung von Gebiet  $l_o(i_o^{(n)})$  mit derjenigen von  $l_{u_\kappa^{(n)}}(i_{u_\kappa^{(n)}})$ , das heißt es wird gesetzt

$$\begin{aligned} \overset{g\pi}{C}_{l_o(i_o^{(n)}), u_\kappa^{(n)}} &= 1 & ; & \quad \overset{g\pi}{C}_{l_o(i_o^{(n)}), o} = 0 \\ \overset{g\pi}{C}_{l_{u_\kappa^{(n)}}(i_{u_\kappa^{(n)}}), u_\kappa^{(n)}} &= 0 & ; & \quad \overset{g\pi}{C}_{l_{u_\kappa^{(n)}}(i_{u_\kappa^{(n)}}), o} = 1. \end{aligned} \quad (3.129)$$

Die übrigen  $\overset{g\pi}{C}_{i_g, i_\pi}^{(n+1)}$  werden aus dem vorherigen Iterationsschritt übernommen.

Setze

$$i_o^{(n+1)} = i_o^{(n)} - 1 \quad (3.130)$$

sowie

$$i_{u_j^{(n)}}^{(n+1)} = \begin{cases} i_{u_j^{(n)}}^{(n)} + 1 & \text{für } j = \kappa \\ i_{u_j^{(n)}}^{(n)} & \text{sonst.} \end{cases} \quad (3.131)$$

Falls Bedingung (3.127) oder (3.128) nicht erfüllt ist, erhöhe den Wert von  $\kappa$  um Eins und prüfe, ob

$$\kappa \leq N_\pi - 1 \quad (3.132)$$

gilt. Ist dies der Fall, so wird mit Schritt 2 fortgefahren. Anderenfalls wird

$$i_o^{(n+1)} = i_o^{(n)} - 1 \quad \text{und} \quad i_{u^{(n)}}^{(n+1)} = i_{u^{(n)}}^{(n)} \quad (3.133)$$

sowie

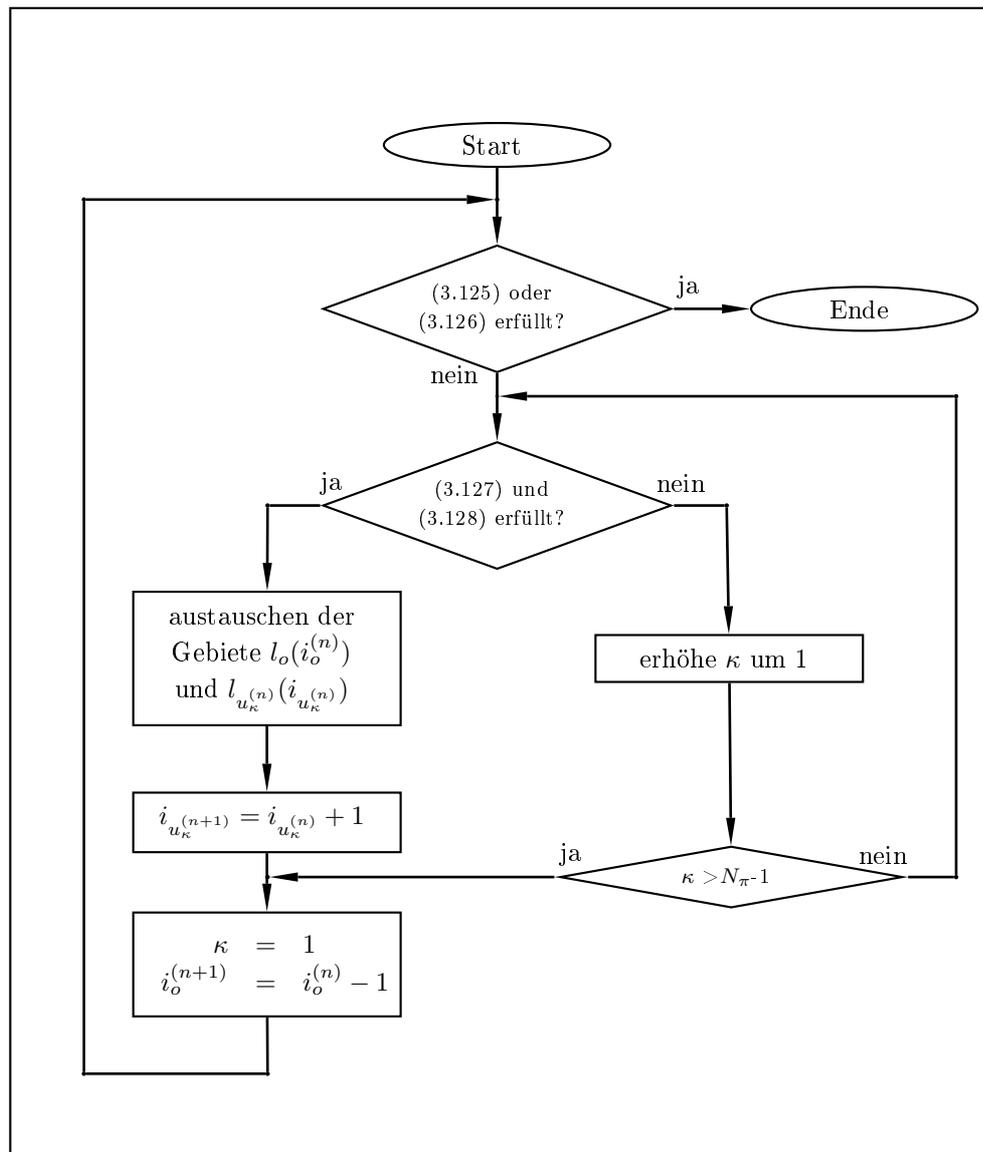
$$\kappa = 1 \quad (3.134)$$

gesetzt und mit Schritt 1 fortgefahren. In Abbildung 3.15 werden die oben beschriebenen Schritte mit Hilfe eines Ablaufdiagramms illustriert.

Für den allgemeinen Fall, in dem eine beliebige Anzahl an Prozessoren überlastet und unterbelastet ist, wird der Algorithmus durch einen Schritt ergänzt, der die Prozessoren vor der Durchführung der oben beschriebenen Schritte derart gruppiert, dass in jeder der gebildeten Gruppen genau ein überlasteter Prozessor existiert und somit innerhalb der entstehenden Gruppen der Algorithmus wie oben beschrieben anwendbar ist. Die Prozessorgruppen werden wie folgt gebildet.

Ohne Beschränkung der Allgemeingültigkeit sei angenommen, dass für die Prozessoren gilt:

$$\alpha_{i_\pi} \cdot \sum_{i_g=1}^G \overset{g\pi}{a}_{i_g, i_\pi} \cdot K_{i_g} \leq \alpha_{j_\pi} \cdot \sum_{i_g=1}^G \overset{g\pi}{a}_{i_g, j_\pi} \cdot K_{i_g} \quad \forall i_\pi < j_\pi, \quad (3.135)$$



**Abbildung 3.15:** Das Ablaufdiagramm illustriert das in Abschnitt 3.4.6.2 beschriebene Lastbalancierungsverfahren.

d.h. die Prozessoren seien proportional zu den Laufzeiten sortiert, welche sie für die Trajektorienberechnung benötigen.

Zunächst wird eine Prozessorgruppe aus Prozessor  $N_{\pi}$  mit der größten Laufzeit und den Prozessoren  $1 \dots v$  mit den geringsten Laufzeiten gebildet, wobei  $v$  der kleinstmögliche

Wert ist, für den gilt

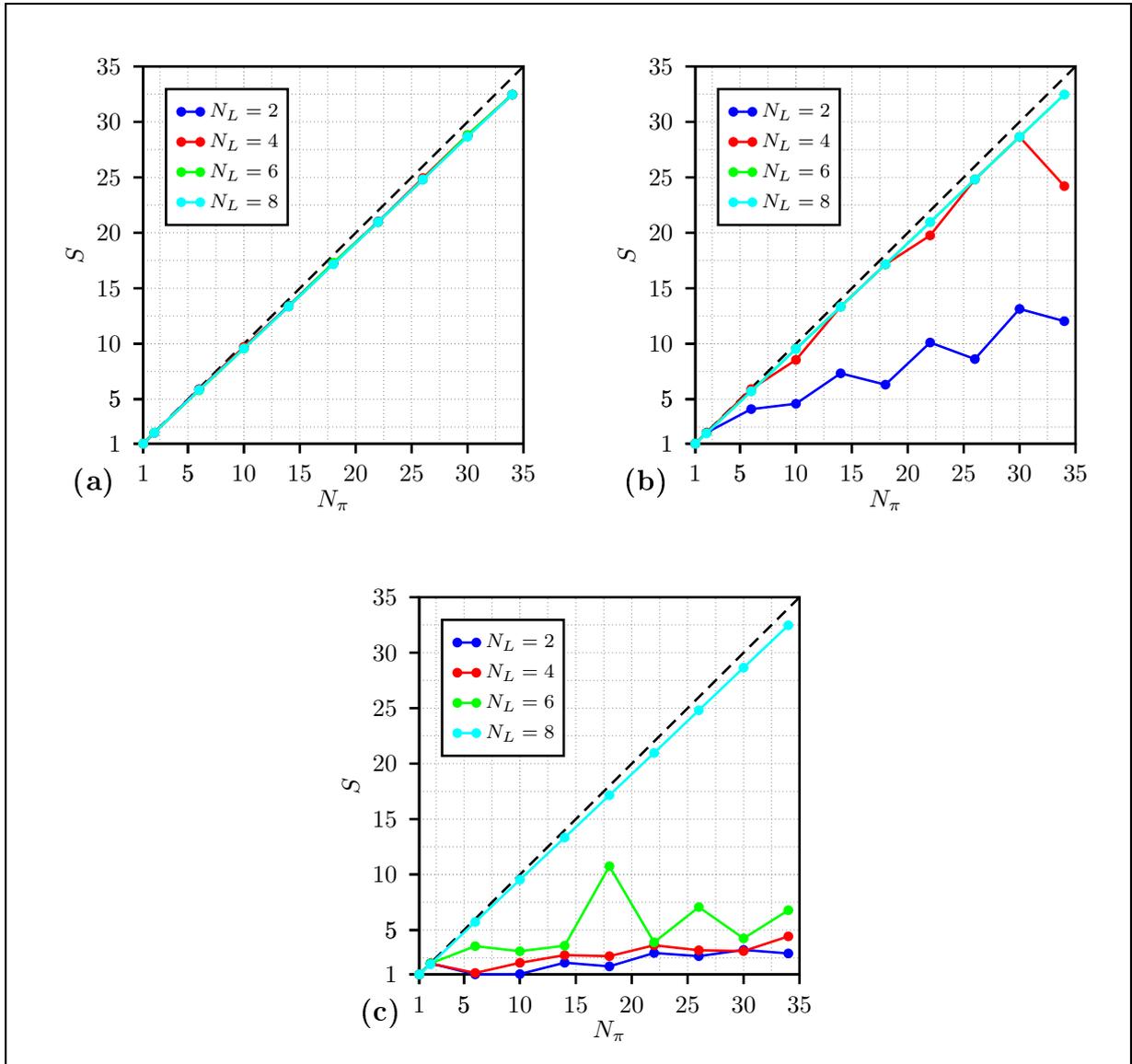
$$\boxed{\frac{\sum_{i_\pi=1}^v \frac{1}{\alpha_{i_\pi}}}{\sum_{j_\pi=1}^{N_\pi} \frac{1}{\alpha_{j_\pi}}} \cdot P - \sum_{i_\pi=1}^v P_{i_\pi} \geq P_{N_\pi} - \frac{1}{\alpha_{N_\pi}} \cdot P} \quad (3.136)$$

d.h. die Belastung der Prozessoren  $1 \dots v$  ist derart, dass die Teilchen, welche die Überlastung von Prozessor  $N_\pi$  verursachen, von diesen aufgenommen werden könnten, ohne dass diese Prozessoren selbst überlastet werden. Die verbleibenden Prozessoren werden nun nach dem gleichen Prinzip weiter gruppiert. Die Lastbalancierungsoperation kann dann, wie oben beschrieben, innerhalb jeder der gebildeten Prozessorgruppen unabhängig von allen anderen Gruppen erfolgen, womit das Lastbalancierungsverfahren selbst wiederum parallelisiert ist.

Es sei angemerkt, dass evtl. für die letzte sich ergebende Prozessorengruppe eine Bedingung wie in Gleichung (3.136) nicht mehr erfüllt werden kann. Daher kann es sinnvoll sein, das Verfahren mehrfach hintereinander anzuwenden. In der im Rahmen der Arbeit durchgeführten Implementierung wurde das Verfahren daher so lange wiederholt, bis nach den oben angegebenen Kriterien keine Gebiete mehr ausgetauscht werden konnten.

Die Graphen in Abbildung 3.16 zeigen den zu erwartenden Speedup für die Integration der Trajektorien für den Fall eines Parallelrechners mit gleichartigen Prozessoren, nachdem die Teilgebiete nach dem oben beschriebenen Algorithmus den Prozessoren zugeordnet wurden. Die Werte wurden über die in der Zielfunktion angenommenen Abhängigkeiten von  ${}^P T_S^{(m)}$  für unterschiedlich stark lokalisierte Teilchenverteilungen bestimmt. Die Anzahl der Gebiete pro Prozessor wurde zu  $G_{i_\pi} = 2^{N_L}$  festgelegt, wobei  $N_L$  ein Parameter ist.

Da lediglich Teilgebiete zwischen den Prozessoren ausgetauscht werden können, um eine gleichmäßige Lastverteilung zu erreichen, ist eine hinreichend gute Auflösung der Teilchenverteilung durch die Teilgebiete notwendig, d.h. die nach Anwendung des Verfahrens verbleibende Lastdifferenz zwischen den Prozessoren ist umso kleiner, je mehr Gebiete existieren, auf die die Rechenlast verteilt ist. Für räumlich sehr stark lokalisierte Teilchenverteilungen erfordert dies eine entsprechend große Anzahl an Gebieten, um eine gleichmäßige Lastverteilung gewährleisten zu können. Dies führt zu einem Anstieg der Anzahl der Randzellen, für die in jedem Zeitschritt Daten ausgetauscht werden müssen, der in den Grafiken in Abbildung 3.16 ebensowenig berücksichtigt wurde wie der Aufwand für die Lastbalancierungsoperation. Wie stark sich diese Effekte bemerkbar machen, wird im folgenden Kapitel mit Hilfe realer Benchmarks untersucht.



**Abbildung 3.16:** Speedup Werte für die Integration der Teilchentrajektorien vor und nach Anwendung des in Abschnitt 3.4.6.2 beschriebenen Algorithmus für verschiedene Anzahl an Untergebieten pro Prozessor  $G_{i_\pi} = 2^{N_L}$ . Die gestrichelte Linie bezeichnet den idealen Speedup. (a) Die Teilchen sind über das komplette Rechengebiet gleichmäßig verteilt. (b) Die Teilchen konzentrieren sich in 10% der Gitterzellen des Rechengebietes. (c) Die Teilchen konzentrieren sich in 1% der Zellen des Rechengebietes.

---

## 4 Benchmark- und Simulationsergebnisse

Im folgenden Kapitel sollen zunächst in Abschnitt 4.1 Benchmarkprobleme, welche zur Performanceevaluierung der Implementierungen der im vorangegangenen Kapitel entwickelten Parallelisierungsstrategien zum Einsatz kamen, vorgestellt und die erhaltenen Ergebnisse diskutiert werden. Im Anschluss wird in Abschnitt 4.2 der **Photoinjektor-Teststand** in **Zeuthen** (PITZ), welcher im Rahmen eines Projektes zum Aufbau eines **Freie-Elektronen-Lasers** (FEL) vom **Deutschen Elektronen Synchrotron** (DESY) betrieben wird, als praxisnahes Anwendungsbeispiel für eine PIC Simulation vorgestellt. Mit Hilfe der im Rahmen der Arbeit entwickelten Software wurde eine 3D Simulation der Teilchendynamik bis zu einer Entfernung von zwei Metern ab der Emissionsstelle durchgeführt.

### 4.1 Benchmarkergebnisse

Um Implementierungen der in Kapitel 3 vorgestellten Parallelisierungsstrategien sinnvoll analysieren zu können, ist die Definition von Benchmarkproblemen notwendig, welche typische Merkmale realer Simulationsprobleme beinhalten. In den folgenden Abschnitten wird das Verhalten von Implementierungen der vorgestellten Parallelisierungsstrategien für solche Benchmarkprobleme analysiert.

Als Testumgebung diente ein Clusterrechner ausgestattet mit insgesamt 90 Einzelrechnern verschiedenen Typs. Die wichtigsten Merkmale der drei Rechnertypen sowie deren Anzahl sind in Tabelle 4.1 aufgelistet. Die Rechner waren weiterhin ausgestattet mit 8 GB Arbeitsspeicher (DDR2) und verbunden über ein Infiniband-Netzwerk (SDR, 10 Gb/s). Für die Benchmarks wurden im Wesentlichen die Rechner des Typs 2 verwendet. Für alle im Folgenden aufgelisteten Ergebnisse mit  $N_\pi \leq 36$  wurde genau eine Instanz des Programms auf den verwendeten Rechnern des Typs 2 ausgeführt. Für alle Benchmarks mit  $N_\pi > 36$  wurden entsprechend Rechner des Typs 3 hinzugenommen.

Der Quellcode wurde kompiliert mit dem Microsoft C/C++ Compiler für x64 Prozessoren in der Version 14.0 mit der Optimierungseinstellung `-O2`. Bei der verwendeten MPI-

Typ	Anzahl	Prozessor	$\frac{\text{Cores}}{\text{Prozessor}}$	$\frac{\text{Prozessoren}}{\text{System}}$	Taktfreq.	Level 2 Cache	RAM
1	24	Intel Xeon	1	1	3,2 GHz	2 MB	8 GB
2	36	Intel Xeon 5130	2	2	2,0 GHz	4 MB	8 GB
3	30	Intel Xeon 5335	4	2	2,0 GHz	8 MB	8 GB

**Tabelle 4.1:** Die drei Typen von Einzelrechnern im verwendeten Clusterrechner.

Bibliothek handelt es sich um die von Microsoft mit dem Betriebssystem Windows Server 2003 in der „Compute Cluster Edition“ mitgelieferten Implementierung. Die Zeitmessung geschah durch Zählen der CPU-Taktzyklen für jede der Operationen mittels der Funktion `__rdtsc`.

Die in den Lastbalancierungsalgorithmen verwendeten Maschinenparameter  $\alpha_{i_\pi}$  und  $\beta_{i_\pi}$  wurden über die sich ergebenden Laufzeiten bei mehrmaliger Ausführung der entsprechenden Operationen bestimmt. Dazu wurden die entsprechenden Operationen auf jedem der an der Simulation beteiligten Prozessoren im Vorfeld der durchzuführenden Simulation mehrmals durchgeführt und aus den gemessenen Laufzeiten durch Mittelwertbildung der entsprechende Wert für jeden der Parameter gewonnen.

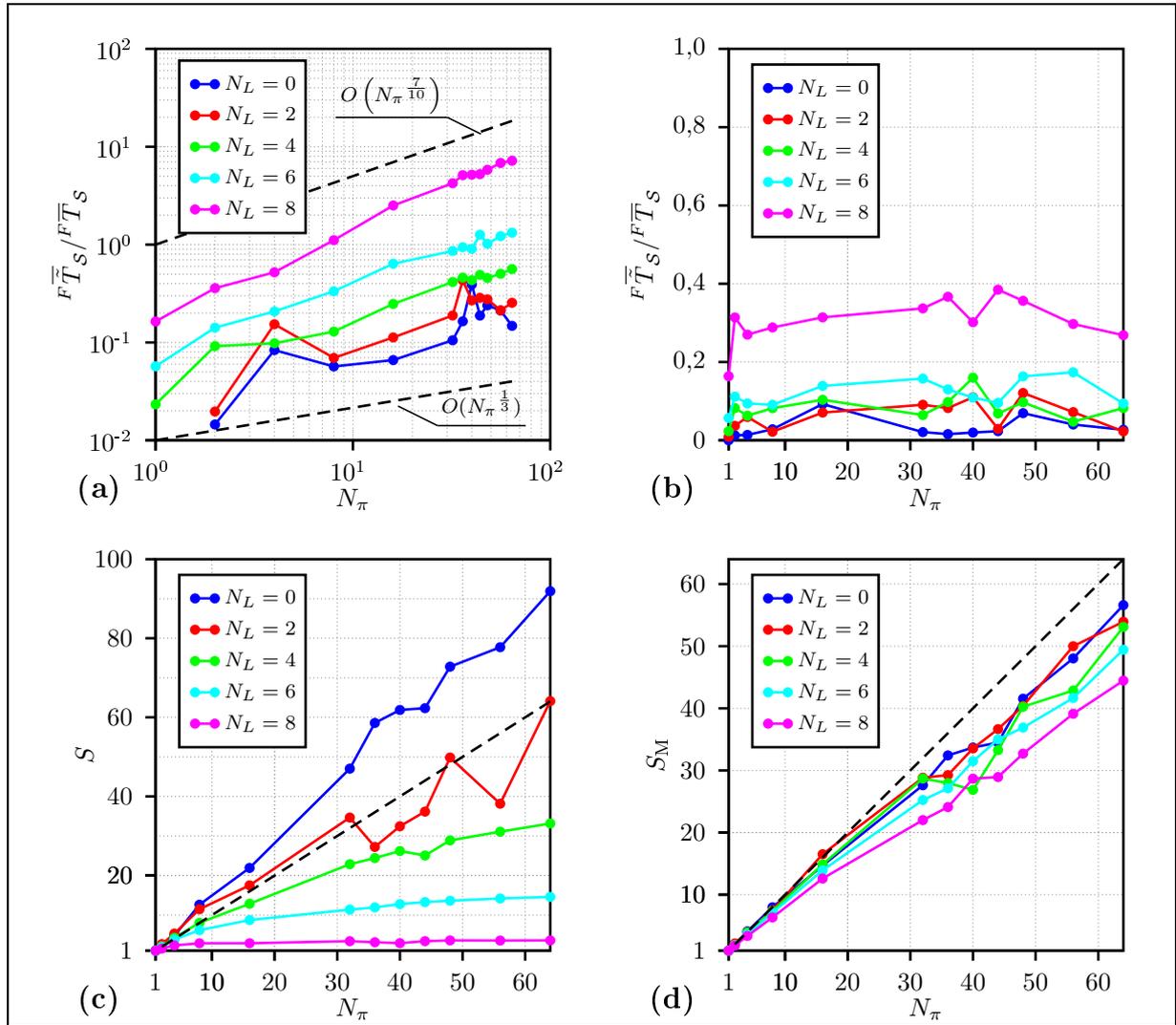
Für die in Abschnitt 3.4.6.2 beschriebene Parallelisierungsstrategie wurden die benötigten  $G$  Partitionen durch eine RCB-Partitionierung erzeugt. Die Anzahl der Gebiete pro Prozessor wurde für die Untersuchungen gemäß

$$G = N_\pi \cdot 2^{N_L} \quad (4.1)$$

mit dem Parameter  $N_L$  festgelegt.

#### 4.1.1 Feldlöser Benchmarks

Zunächst soll das Verhalten des parallelisierten Feldlösers analysiert werden. Als Benchmarkbeispiel wurde ein Rechteckresonator (Kantenlänge 2 m) ausgewählt, welcher in allen drei Raumrichtungen äquidistant diskretisiert wurde. Das Rechengebiet wurde mit Hilfe des RCB-Algorithmus in  $G = N_\pi \cdot 2^{N_L}$  gleich große Gebiete unterteilt, wobei jedem der Prozessoren  $2^{N_L}$  Gebiete zugeordnet wurden. Für  $N_L = 0$  ergibt sich somit die gewöhnliche Partitionierung gemäß Abschnitt 3.3.1. Für die Untersuchungen wurde die Zeitintegration der Gitterspannungen über 100 Zeitschritte mit dem in Abschnitt 2.2.3 beschriebenen Leap-Frog-Algorithmus ausgeführt. Aus den sich ergebenden Laufzeiten  ${}^F T_S^{(m)}$  und  ${}^F \tilde{T}_S^{(m)}$  für jeden Zeitschritt wurden die Mittelwerte  ${}^F \bar{T}_S$  bzw.  ${}^F \bar{\tilde{T}}_S$  gebildet. Der sich für das Verhältnis von  ${}^F \bar{T}_S$  zu  ${}^F \bar{\tilde{T}}_S$  ergebende Verlauf wurde in Abhängigkeit von der Anzahl



**Abbildung 4.1:** Benchmarkergebnisse für den Feldlöser für eine Rechnung mit einer Problemgröße von 10 Millionen Gitterzellen. (a) zeigt das Verhältnis von  $F_{T_S}^{\tilde{}}$  zu  $F_{T_S}$  über der Anzahl der verwendeten Prozessoren. (b) zeigt das Verhältnis für die skalierten Problemgrößen. (c) zeigt den Verlauf des gewöhnlichen Speedup  $S$ . (d) zeigt den Verlauf des skalierten Speedup  $S_M$ .

der verwendeten Prozessoren sowie dem Parameter  $N_L$  in den Abbildungen 4.1(a) und (b) aufgetragen, wobei in Abb. 4.1(a) die Verläufe für eine konstante Problemgröße von 10 Millionen Gitterzellen und in Abb. 4.1(b) die Verläufe für die entsprechenden skalierten Problemgrößen aufgetragen sind, d.h. die Anzahl der Gitterzellen wurde proportional zur Anzahl der Prozessoren erhöht.

Nach den in Abschnitt 3.3.2 angestellten Überlegungen gilt für das Verhältnis  $F_{T_S}^{\tilde{}} / F_{T_S}$

bei konstanter Problemgröße und  $N_L = 0$  der Zusammenhang

$$O\left({}^F\bar{T}_S / {}^F\bar{\tilde{T}}_S\right) = O\left(N_\pi^{\frac{1}{3}}\right), \quad (4.2)$$

welcher für die entsprechende Kurve in Abb. 4.1(a) recht gut erreicht wird. Für die weiteren Kurven ergibt sich durch das Ansteigen der Datenmenge, die in jedem Zeitschritt ausgetauscht werden muss, bei gleichzeitig unveränderten Laufzeiten für die Berechnungen  ${}^F\bar{T}_S$ , ein Anstieg des Verhältnisses (Verschiebung der Kurve nach oben) sowie auch eine größere Steigung der Kurven. Die größere Steigung kann auf den in Abschnitt 2.4.5 erläuterten Effekt zurückgeführt werden, dass die Zeit, welche im Mittel für den Datenaustausch benötigt wird, mit der Anzahl der miteinander kommunizierenden Prozessoren und der Datenmenge ansteigt.

Für die skalierten Problemgrößen ist ein von der Prozessorenzahl unabhängiges Verhältnis von  ${}^F\bar{T}_S$  zu  ${}^F\bar{\tilde{T}}_S$  zu erwarten, da sich die Menge der in jedem Zeitschritt auszutauschenden Daten sowie die Anzahl der Berechnungsoperationen pro Prozessor nicht ändern. Die Kurven in Abb. 4.1(b) zeigen dieses Verhalten recht gut. Ebenso lässt sich erkennen, dass die für den Datenaustausch aufgewendete Laufzeit im Verhältnis zur Rechenzeit selbst für eine große Anzahl an Teilgebieten pro Prozessor recht gering bleibt.

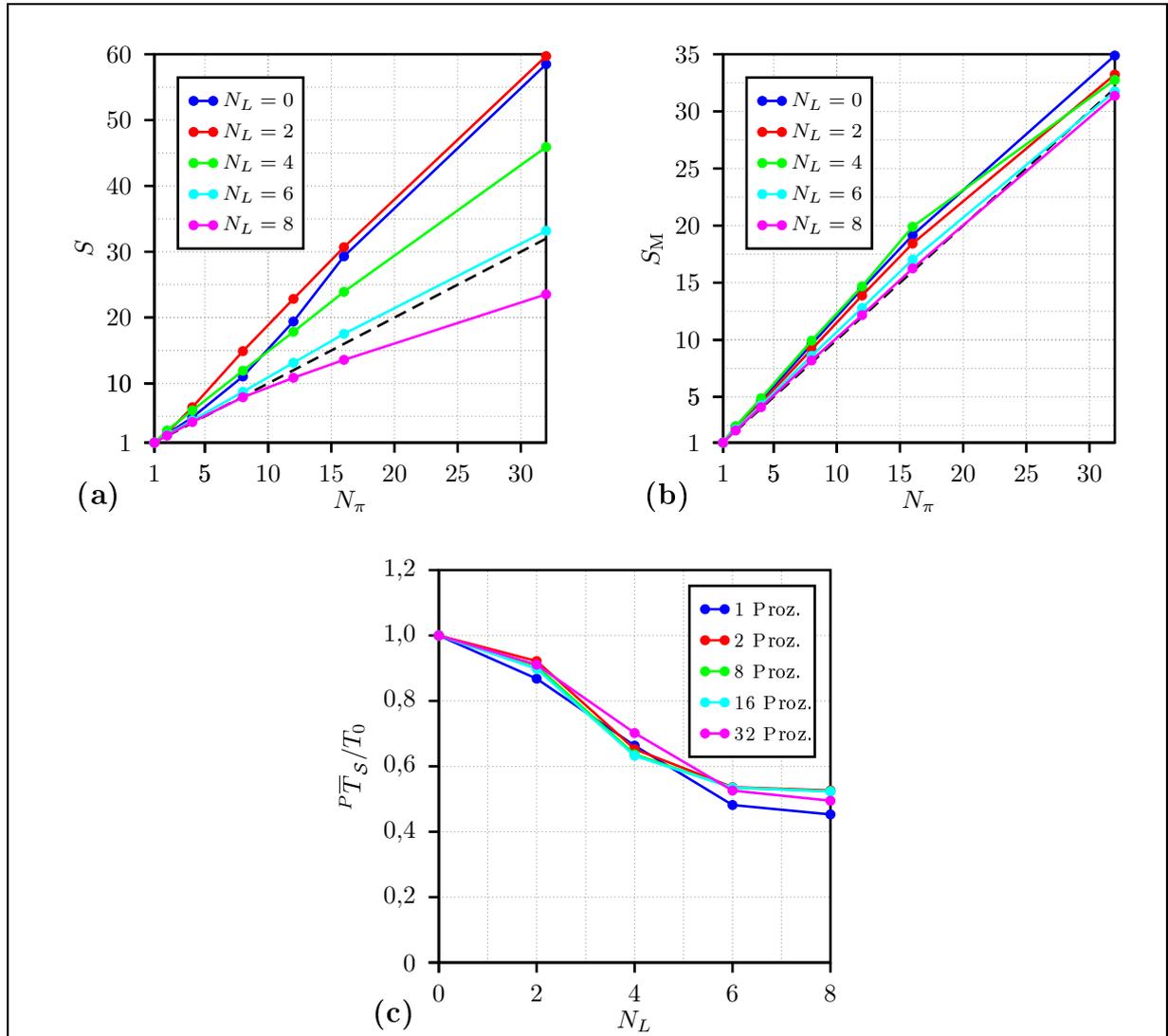
Abbildungen 4.1(c) und (d) zeigen die Verläufe für den gewöhnlichen und den speicherskalierten Speedup. Interessant an Abb. 4.1(c) ist der superlineare Verlauf für die Fälle  $N_L = 0$  sowie  $N_L = 2$ , der auf Cache-Effekte zurückgeführt werden kann. Da die Laufzeit für den Feldlöser maßgeblich durch die Speicherzugriffszeit bestimmt wird, ergibt sich ein deutliches Abfallen derselben, wenn durch die Verringerung der Problemgröße pro Prozessor ein großer Anteil der Gitterspannungen im Cache gehalten werden kann. Dieser Effekt existiert zwar auch für die übrigen Werte von  $N_L$ , wird jedoch durch das Ansteigen der Kommunikationszeit überkompensiert.

Der Verlauf des skalierten Speedup in Abb. 4.1(d) verifiziert die Skalierbarkeitsanalyse für den Feldlöser in Abschnitt 3.3.2. Aufgrund des dort hergeleiteten Ausdrucks ist bei einer proportional zur Anzahl der Prozessoren ansteigenden Anzahl an Gitterpunkten eine konstante Effizienz bzw. ein linear ansteigender Speedup zu erwarten, was durch den erhaltenen Verlauf näherungsweise bestätigt wird.

## 4.1.2 PIC Benchmarks

### 4.1.2.1 Homogene Teilchenverteilung

Für den einfachsten Fall einer räumlich homogenen Teilchenverteilung führen alle Parallelisierungsstrategien im Falle eines Parallelrechners mit gleichartigen Prozessoren zu einer



**Abbildung 4.2:** Benchmarkergebnisse für homogene Teilchenverteilung für eine Basisproblemgröße von einer Million Gitterzellen und durchschnittlich 10 Teilchen pro Gitterzelle. (a) zeigt den gewöhnlichen Speedup  $S$  für eine unterschiedliche Anzahl an Teilgebieten pro Prozessor. (b) zeigt den skalierten Speedup  $S_M$  für die entsprechend skalierten Problemgrößen. (c) zeigt die zur Berechnung der Trajektorien pro Zeitschritt für die skalierten Problemgrößen durchschnittlich benötigte Laufzeit  $P\bar{T}_S$  in Abhängigkeit von der Anzahl der Teilgebiete pro Prozessor. Die Laufzeiten sind normiert auf die Laufzeit  $T_0$ , welche sich für den Fall  $N_L = 0$  ergibt.

gleichmäßigen Zuordnung der Teilchen und Gitterzellen. Die Rechenlast ist ideal balanciert und die in jedem Schritt auszutauschende Datenmenge ist minimal. Somit handelt es sich um den Fall, für den die beste Skalierbarkeit zu erwarten ist. Als Benchmarkproblem wurde

ein quaderförmiger Raumbereich von 2 m Kantenlänge ausgewählt, welcher mit insgesamt einer Million Gitterzellen äquidistant diskretisiert wurde und pro Zelle durchschnittlich zehn Teilchen enthält. Abbildung 4.2(a) zeigt den Speedup pro Zeitschritt, welcher sich für dieses Benchmarkproblem ergibt. Wiederum wird ein deutlich superlinearer Speedup erreicht, was, wie beim Feldlöser, auf Cache-Effekte zurückgeführt werden kann.

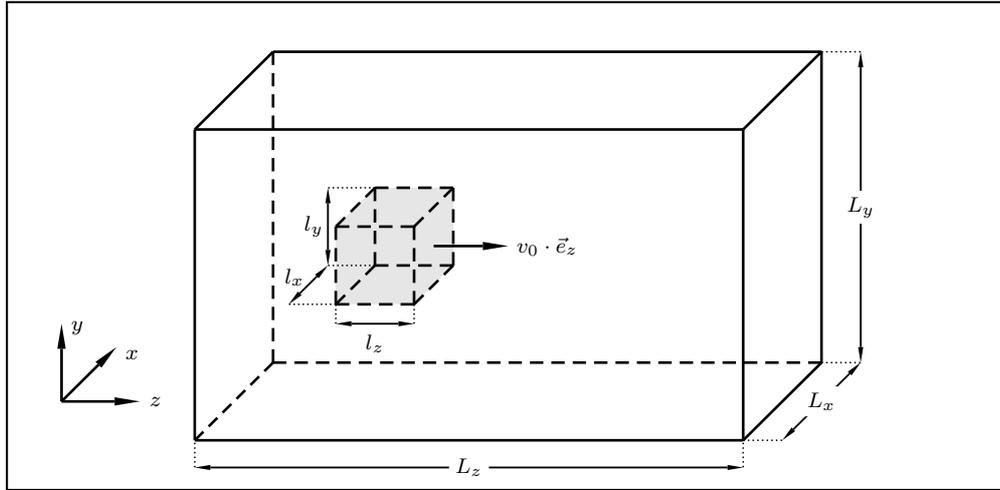
Abbildung 4.2(c) illustriert den Effekt des Cache-Speichers abhängig von der Anzahl der Gebiete, die jedem Prozessor zugeordnet werden, für die skalierten Problemgrößen. Aufgetragen ist die mittlere Laufzeit, welche pro Zeitschritt für die Berechnung der Trajektorien der Teilchen benötigt wurde, normiert auf die entsprechende Laufzeit  $T_0$  für  $N_L = 0$ . Durch die Segmentierung des jedem Prozessor zugeordneten Rechengebietes in Teilgebiete, sowie das gemeinsame Ablegen aller mit einer Partition verbundenen Daten in einer Datenstruktur, wird bei den im Rahmen der Trajektorienberechnung auftretenden Speicherzugriffen der Cache besser ausgenutzt, was sich in einer deutlichen Reduktion der Laufzeit für diese Operation zeigt. Durch diese Segmentierung wird eine Sortierung der Teilchen gemäß ihrer Position im Gitter eingeführt, ähnlich wie durch den in [76] beschriebenen Sortieralgorithmus.

Für den Fall der skalierten Problemgröße ergibt sich erwartungsgemäß ein ähnliches Resultat wie für den Feldlöser, wobei sich der Verlauf aufgrund der durch die Teilchen zusätzlich eingeführten und ideal verteilten Rechenlast gegenüber dem reinen Feldlöser verbessert hat.

### 4.1.2.2 lokalisierte Teilchenverteilungen

Für eine große Klasse von Simulationsproblemen, welche mit dem PIC-Algorithmus gelöst werden können, sind die Teilchen auf einen im Vergleich zum gesamten Rechengebiet kleinen Raumbereich konzentriert. Die Impulse aller Teilchen innerhalb eines solchen Teilchenpaketes variieren in der Regel nur sehr wenig, so dass die Teilchen auch über die gesamte Simulationsdauer in einem kleinen Raumbereich lokalisiert bleiben. Bei der Simulation der Teilchendynamik innerhalb von Beschleunigern tritt dieser Fall beispielsweise in Erscheinung, und auch bei dem in Abschnitt 4.2 vorgestellten PITZ-Injektor handelt es sich um eine Simulation diesen Typs.

Um das Verhalten der Parallelisierungsstrategien (bzw. ihrer Implementierungen) für diese Problemklasse zu analysieren, wurde das in Abbildung 4.3 gezeigte Benchmarkproblem verwendet. Dabei ist eine gewisse Anzahl an Makroteilchen gleichmäßig über einen quaderförmigen Bereich mit einem Volumen von  $l_x \cdot l_y \cdot l_z$  verteilt, der sich in einem Rechengebiet mit einem Volumen von  $L_x \cdot L_y \cdot L_z$  befindet. Alle Teilchen bewegen sich mit der konstan-



**Abbildung 4.3:** Diese Skizze illustriert das Benchmarkproblem, welches bei der Untersuchung der parallelen PIC Algorithmen zum Einsatz kam.

Problem Nr.	$L/m$	$l/m$	$N_c$	$P$	$M$	$v_0/c$
1	2,00	1,60	$1,0 \cdot 10^6$	$1,0 \cdot 10^6$	50	0,9
2	2,00	0,75	$1,0 \cdot 10^6$	$1,0 \cdot 10^5$	50	0,9
3	2,00	1,60	$1,0 \cdot 10^7$	$1,0 \cdot 10^7$	108	0,9
4	2,00	0,75	$1,0 \cdot 10^7$	$1,0 \cdot 10^6$	108	0,9

**Tabelle 4.2:** Die Einstellungen für die vier betrachteten Benchmarkprobleme.

ten Geschwindigkeit  $v_0 \cdot \vec{e}_z$ . Bei den durchgeführten Benchmarks wurde generell

$$L = L_x = L_y = \frac{1}{10} L_z \quad (4.3)$$

sowie

$$l = l_x = l_y = \frac{1}{2} l_z \quad (4.4)$$

gesetzt. Das Rechengebiet wurde mit einem in jeder Raumrichtung äquidistanten Gitter diskretisiert. Es wurden vier Problemstellungen und die zugehörigen skalierten Problemstellungen untersucht. Skaliert heißt in diesem Fall, dass die Problemgröße pro Prozessor konstant gehalten, die Anzahl der Teilchen und Gitterzellen also proportional zur Anzahl der verwendeten Prozessoren erhöht wurde.

Die Tabelle 4.2 zeigt die für die drei Basisproblemgrößen verwendeten Einstellungen. Die größere Anzahl der Zeitschritte für Problem Nr.3 und Nr.4 erklärt sich daraus, dass die Simulationen jeweils das gleiche Zeitintervall umfassen. Da sich durch die feinere Diskretisierung der Zeitschritt aufgrund der Stabilitätsforderung aus Gleichung (2.38) entspre-

chend verkleinert, vergrößert sich folglich die Anzahl der Zeitschritte entsprechend. Diese Anpassung der Anzahl der Zeitschritte wurde ebenso für die skalierten Problemgrößen durchgeführt, so dass alle durchgeführten Simulationen dieselbe Zeitspanne umfassen.

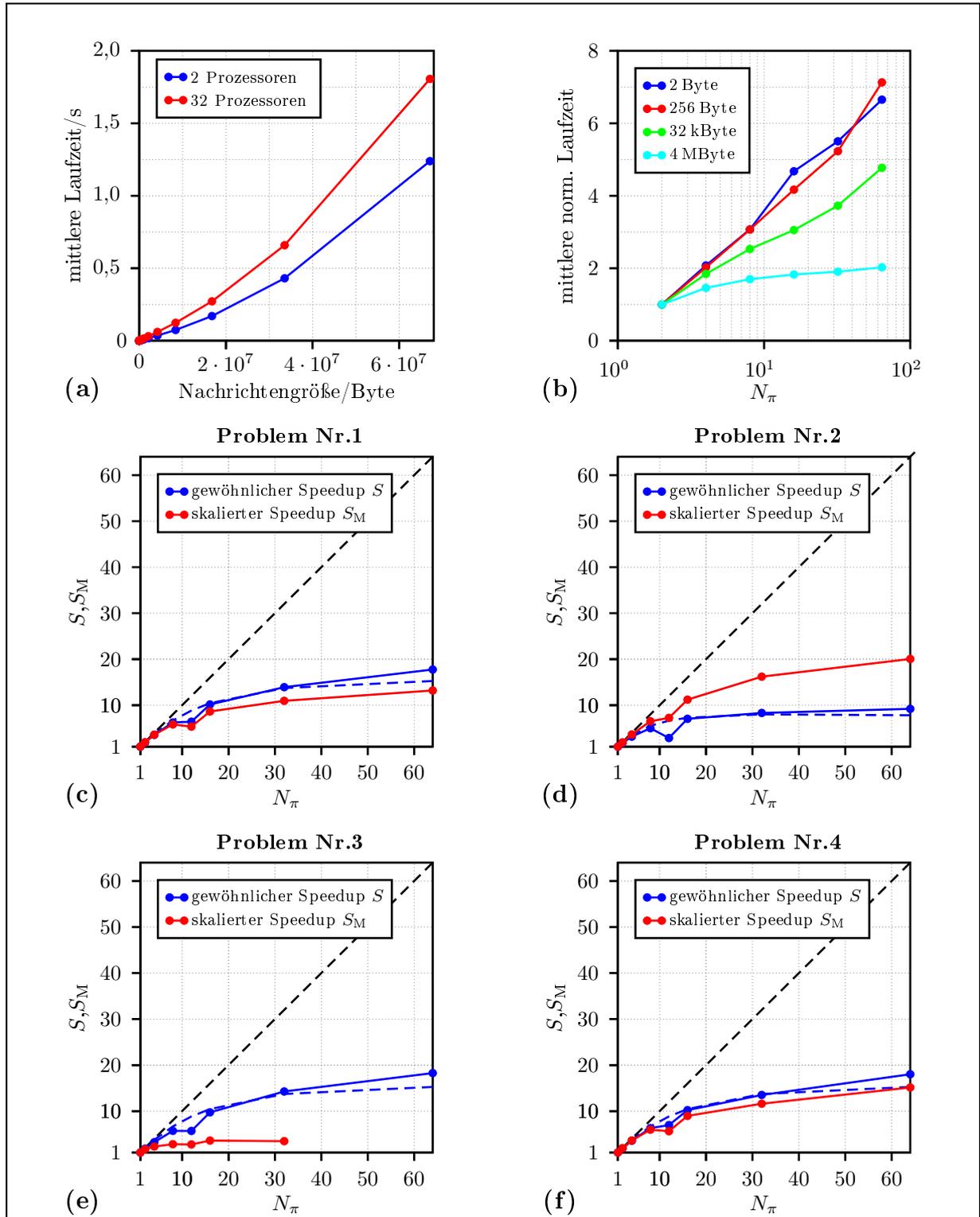
Bei den Problemen 1 und 3 sind 10% des Gesamtvolumens des Rechengebietes mit Teilchen gefüllt, während bei den Problemen 2 und 4 nur ein Anteil von 1% des Rechengebietsvolumens mit Teilchen gefüllt ist. Die Anzahl der Teilchen wurde so gewählt, dass sich bei allen Problemen durchschnittlich 10 Teilchen in jeder Gitterzelle des entsprechenden Bereiches befinden.

Für die in den folgenden Grafiken gezeigten Speedup-Ergebnisse wurde die Gesamtlaufzeit der jeweiligen Simulationen zugrunde gelegt.

**Adaptive Bounding Box** Entscheidend für das Verhalten der Parallelisierungsstrategie ist die Allreduce-Operation [41], mit deren Hilfe die durch die Bounding Box referenzierten Feld- bzw. Stromwerte zwischen den Prozessoren ausgetauscht werden. Um das Verhalten der Implementierung der Parallelisierungsstrategie auf dem verwendeten Parallelrechner besser verstehen zu können, wurde daher zunächst das Verhalten der Allreduce-Funktion untersucht. Dazu wurden Nachrichten unterschiedlicher Länge zwischen einer variierenden Anzahl von Prozessoren ausgetauscht. Für jedes Paar aus Nachrichtengröße und Prozessorenzahl wurden 100 Wiederholungen durchgeführt und die mittlere Zeit pro Übertragung bestimmt.

Abbildung 4.4(a) zeigt die Abhängigkeit der Laufzeit von der Länge der versendeten Nachricht. Es zeigt sich, dass auf dem betrachteten System und für die verwendete MPI-Implementierung die Laufzeit etwas stärker als linear von der Nachrichtenlänge abhängt. Abbildung 4.4(b) zeigt die Abhängigkeit der Laufzeit für die Allreduce-Operation von der Anzahl der beteiligten Prozessoren für unterschiedliche Nachrichtenlängen, wobei die Laufzeiten auf die gemessene Zeit für den Fall zweier Prozessoren normiert wurden. In dem verwendeten logarithmischen Maßstab für die Abszisse sind aufgrund des  $\log(N_\pi)$ -Terms in dem Ausdruck für  ${}^P\tilde{T}_S^{(m)}$  (siehe Gleichung (3.84)) gerade Verläufe zu erwarten. Für kleine Nachrichtenlängen ergibt sich der erwartete Verlauf in sehr guter Näherung, während für größere Nachrichten ein Abflachen der Kurven zu beobachten ist.

Die weiteren Abbildungen 4.4(c)-(f) zeigen die Ergebnisse für die in Tabelle 4.2 spezifizierten Benchmarkprobleme. Die mit durchgezogenen Linien gezeichneten Kurven wurden mit Hilfe von Messwerten bestimmt, während die gestrichelt eingezeichneten Verläufe gleicher Farbe mit Hilfe der analytischen Ausdrücke in Abschnitt 3.4.4 gewonnen wurden. Für den Speedup ist, nach der Herleitung in Abschnitt 3.4.4, für ein Simulationsproblem, bei dem die Gesamtanzahl der Teilchen sowie die Anzahl an Gitterzellen innerhalb der Bounding



**Abbildung 4.4:** Benchmarkergebnisse zur Parallelisierungsstrategie „Adaptive Bounding Box“. (a) zeigt die mittlere Laufzeit zur Durchführung der Allreduce Operation in Abhängigkeit von der verschickten Datenmenge. (b) zeigt die benötigte Laufzeit für die Allreduce Operation in Abhängigkeit von der Anzahl der beteiligten Prozessoren. (c)-(f) zeigen den Verlauf des gewöhnlichen und skalierten Speedup für die in Tabelle 4.2 beschriebenen Benchmarkprobleme.

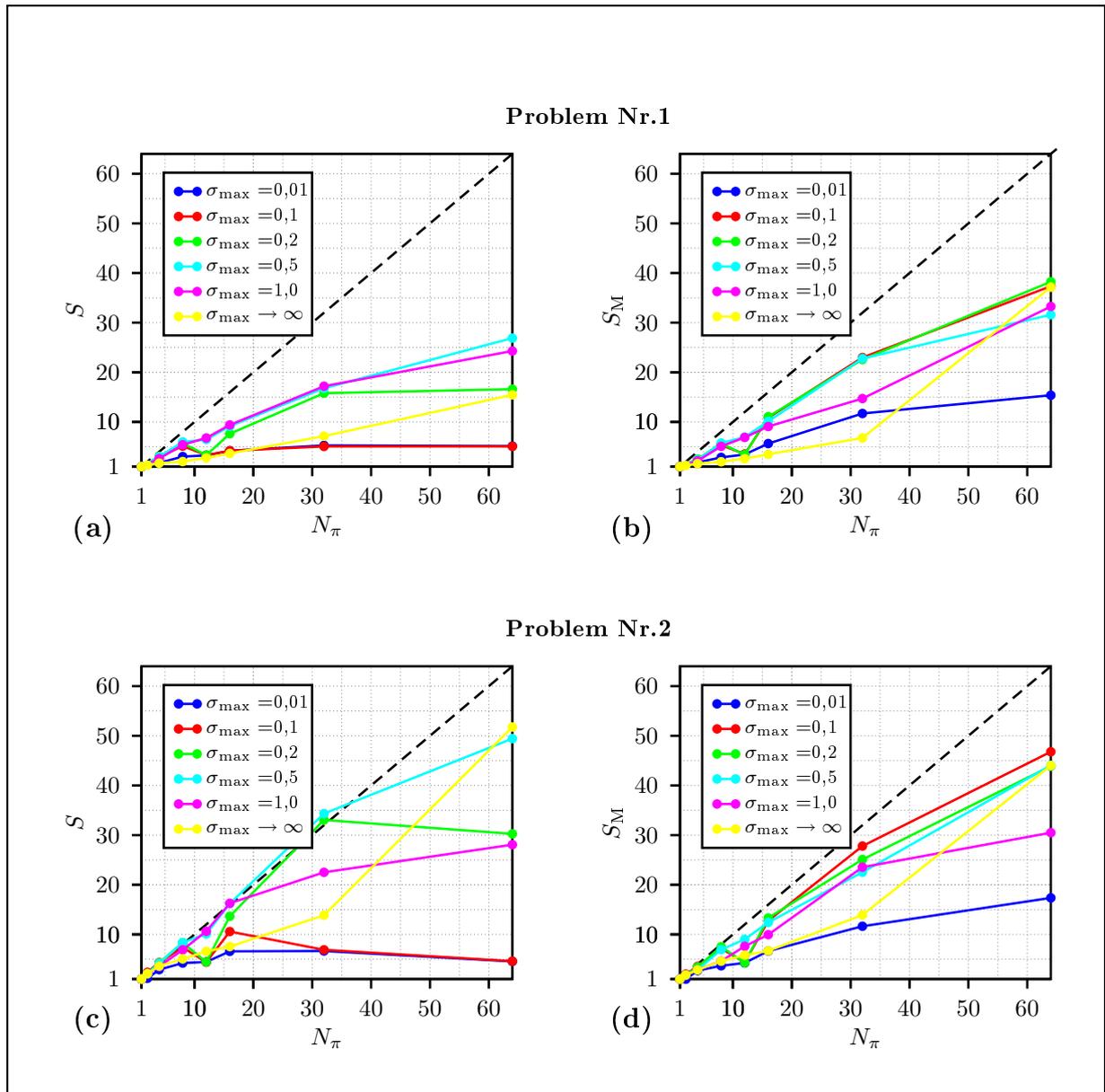
Box  $B_{\min}$  während der Simulation gleich bleibt, ein Verlauf gemäß

$$S(N_\pi, \mathcal{P}) \approx \frac{1}{\frac{1}{N_\pi} + \frac{\kappa|B_{\min}|/N_c}{\alpha + \beta P/N_c} \log(N_\pi)} = \frac{1}{\frac{1}{N_\pi} + k \cdot \log(N_\pi)} \quad (4.5)$$

zu erwarten. Für den speicherskalierten Speedup ergibt sich ein zu (4.5) identischer Ausdruck. Durch eine Vergrößerung der Problemgröße ist folglich keine Verbesserung des Speedup zu erwarten, wie bereits in Abschnitt 3.4.4 bei der dort vorgestellten Skalierbarkeitsuntersuchung festgestellt wurde. Dadurch, dass die Laufzeit für die Allreduce Operation im Gegensatz zu der für die Berechnung der Trajektorien aufgewendeten Laufzeit auf dem betrachteten System überlinear von der Problemgröße abhängt, ist sogar zu erwarten, dass der Verlauf des skalierten Speedup unterhalb des unskalierten Speedup verläuft. Durch das etwas schwächer als im Modell angenommene Ansteigen der Laufzeit für die Allreduce Operation von der Anzahl der Prozessoren, welches in Abbildung 4.4(b) gezeigt wurde, ist zu erwarten, dass der Verlauf des Speedup ein etwas besseres Verhalten zeigt als von Gleichung (4.5) prognostiziert. Anhand der Abbildungen 4.4(c),(e) und (f) erweisen sich beide Aussagen als richtig. Bei den blau gestrichelt eingezeichneten Kurven handelt es sich um mit Hilfe von Gleichung (4.5) prognostizierte Verläufe, welche bis 32 Prozessoren für die Fälle  $N_\pi = 2^n, n \in \mathbb{N}$  den gemessenen Verlauf in guter Näherung abbilden. Für eine noch größere Anzahl verwendeter Prozessoren wird der Verlauf des Speedup etwas unterschätzt. Die Abweichung im Fall  $N_\pi = 12$  lässt sich dadurch erklären, dass die zur Implementierung des Datenaustausches verwendete Allreduce-Operation Algorithmen verwendet, die für den Fall  $N_\pi = 2^n, n \in \mathbb{N}$  optimiert sind und für andere Prozessorenzahlen eine schlechtere Performance liefern [47].

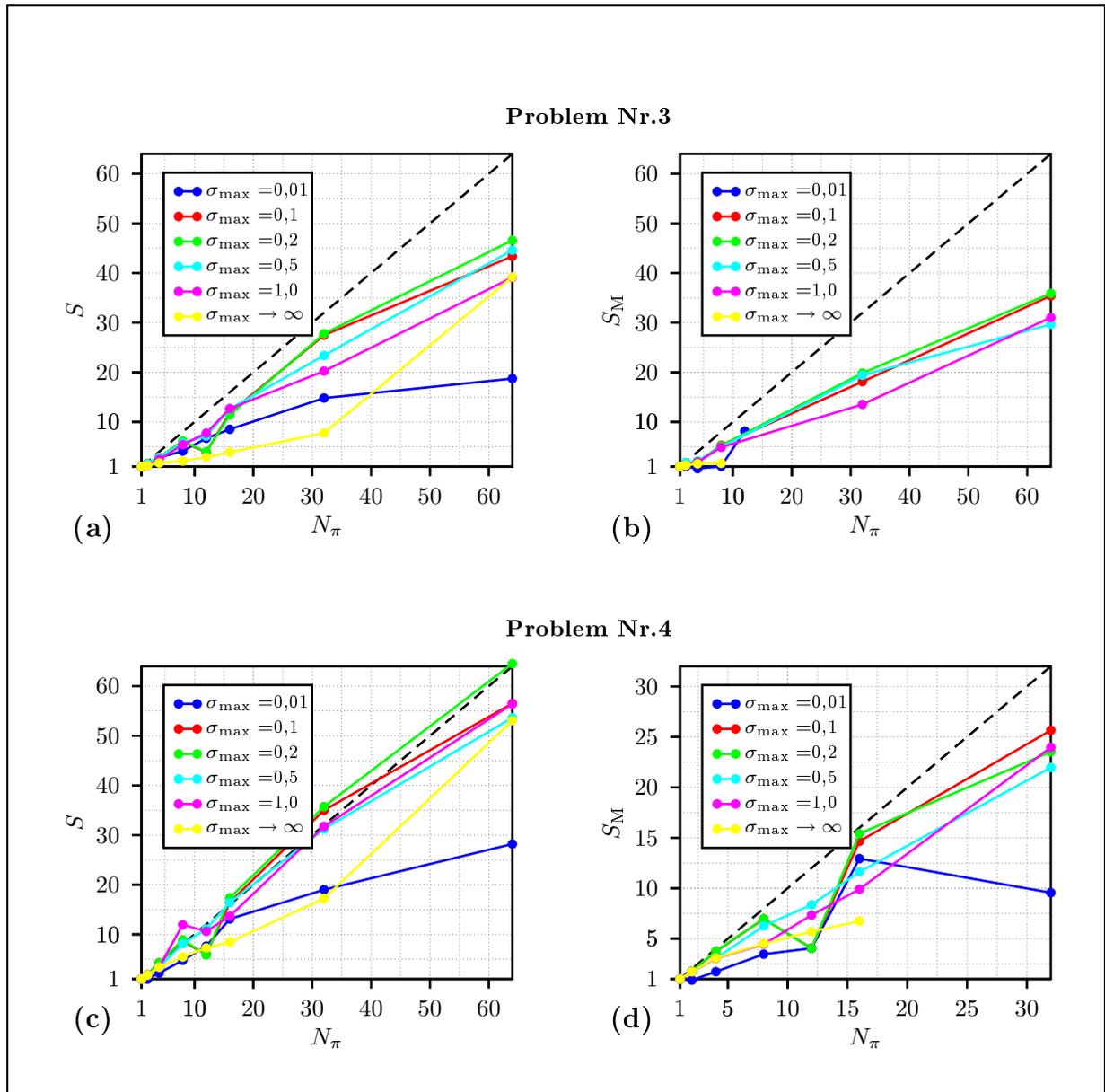
Für das kleinste der betrachteten Probleme (siehe Abbildung 4.4(d)) ergibt sich ein deutlich schlechterer Verlauf des Speedup für das unskalierte Problem als bei den übrigen Benchmarks. Dies kann auf das Verhalten der Allreduce Operation für das betrachtete parallele System zurückgeführt werden. Da sich für kleine Nachrichtenlängen ein deutlich stärkerer Anstieg der Laufzeit für diese Operation ergibt (siehe Abbildung 4.4(b)), resultiert dies in einem schlechteren Verlauf für den Speedup für das relativ kleine Problem. Man beachte, dass der sich ergebende Verlauf für den Speedup besser durch den Ausdruck (4.5) beschrieben werden kann als dies bei den übrigen Problemen der Fall war, bei denen sich das Abflachen der Verläufe in Abbildung 4.4(b) stärker bemerkbar machte. Insgesamt liefert die Parallelisierungsstrategie nur für Probleme, welche sich mit einer kleinen Anzahl an Prozessoren lösen lassen, recht gute Ergebnisse, was aufgrund der Ergebnisse der Skalierbarkeitsuntersuchung zu erwarten war.

**Rekursive Koordinaten Bisektionierung** Abbildungen 4.5 und 4.6 zeigen die mit Hilfe der in Abschnitt 3.4.6.1 beschriebenen Methode für die vier Benchmarkprobleme



**Abbildung 4.5:** Benchmarkergebnisse für die Parallelisierungsstrategie „*Rekursive Koordinaten Bisektionierung*“. (a) und (c) zeigen den Verlauf des gewöhnlichen Speedup  $S$ . (b) und (d) zeigen den Verlauf des skalierten Speedup  $S_M$  für die Benchmarkprobleme Nr. 1 und 2, welche in Tabelle 4.2 spezifiziert sind.

aus Tabelle 4.2 erhaltenen Ergebnisse. Die in den Grafiken eingetragenen Verläufe des gewöhnlichen und des speicherskalierten Speedup wurden für unterschiedliche Werte des Parameters  $\sigma_{\max}$ , welcher über die Durchführung der Lastbalancierungsoperation entscheidet, erhalten. Obwohl ein sehr niedriger Wert von  $\sigma_{\max}$  dazu führt, dass die Rechenlast



**Abbildung 4.6:** Benchmarkergebnisse für die Parallelisierungsstrategie „*Rekursive Koordinaten Bisektionierung*“. (a) und (c) zeigen den Verlauf des gewöhnlichen Speedup  $S$ . (b) und (d) zeigen den Verlauf des skalierten Speedup  $S_M$  für die Benchmarkprobleme Nr. 3 und 4, welche in Tabelle 4.2 spezifiziert sind.

gleichmäßig auf die Prozessoren verteilt bleibt, ergibt sich für diesen Fall ein nicht zufriedenstellender Verlauf des Speedup, der für einige der betrachteten Benchmarks sogar hinter dem Verlauf, welcher sich für den Fall ohne jegliche Lastbalancierung ergibt ( $\sigma_{\max} \rightarrow \infty$ ), zurückbleibt. Bei Simulationen ohne dynamische Lastbalancierung wurde allerdings nach

der Generierung der Anfangsverteilung der Teilchen im Rechengebiet und noch vor den Berechnungen des ersten Zeitschrittes eine Lastbalancierung durchgeführt, da anderenfalls die meisten der Simulationen für Problem 3 und 4 mit skalierten Problemgröße aufgrund der begrenzten Speicherressourcen der Einzelrechner nicht durchführbar gewesen wären. Dennoch sind auch auf diese Weise einige der Simulationen nicht durchführbar, da die sich im Laufe der Simulation verschlechternde Balancierung der Rechenlast und, damit verbunden, die ungleichmäßiger werdende Verteilung der Daten schließlich die Speicherressourcen auf einem der beteiligten Rechner übersteigt, was zum Scheitern der Berechnung führt. Die fehlenden Punkte in den Speedup Kurven für den Fall  $\sigma_{\max} \rightarrow \infty$  in den Abbildungen 4.6(b) und (d) dokumentieren dies und verdeutlichen einmal mehr die Wichtigkeit der dynamischen Lastbalancierung für das Gelingen sehr großer Simulationen.

Die Lastbalancierungsoperation kann nur dann zu einer Verbesserung der Skalierbarkeit im Vergleich zu dem Fall unbalancierter Rechenlast führen, wenn die zur Durchführung der Operation zusätzlich aufzuwendende Laufzeit von dem Laufzeitgewinn, welcher durch die gleichmäßigere Verteilung der Rechenlast erzielt wird, überkompensiert wird. Eine zu häufige Neuverteilung der Daten kann somit in der Summe sogar zu einem Ansteigen der Laufzeit gegenüber dem Fall ohne jegliche Lastbalancierung führen, was für den Fall  $\sigma_{\max} = 0,01$  zu beobachten ist. Für zu große Werte des Entscheidungsparameters hingegen verschlechtert sich die Skalierbarkeit durch die ungleichmäßig auf die Prozessoren verteilte Rechenlast und nähert sich im Grenzfall dem Verlauf, welcher sich für den Fall ohne Lastbalancierung ergibt, an. Der optimale Wert des Entscheidungsparameters ist maschinen- und implementierungsabhängig. Für die durchgeführten Benchmarks lieferte ein Wert von  $\sigma_{\max}$  im Bereich von 0,1 bis 0,2 meist die besten Ergebnisse auf dem betrachteten Parallelrechner.

Generell ist zu beobachten, dass die erreichbare Skalierbarkeit für die Probleme mit konzentrierter Teilchenverteilung (Problem Nr.2 und Nr.4) im Vergleich zu den übrigen besser ist. Dies ist darauf zurückzuführen, dass der Anteil der statischen Rechenlast, welcher mit den Gitterzellen verbunden ist, für diese Problemstellungen größer ist, gegenüber dem dynamischen Anteil, welcher durch die Teilchen repräsentiert wird. Eine ungleichmäßig verteilte Rechenlast kann sich daher weniger stark bemerkbar machen.

Für die superlinearen Werte des gewöhnlichen Speedup können wiederum Effekte verantwortlich gemacht werden, die auf die Speicherhierarchie (Cache) des betrachteten Parallelrechners zurückzuführen sind. Bereits in Abbildung 4.2 wurde gezeigt, dass die Speicherhierarchie moderner Rechner einen großen Einfluss auf die Laufzeit hat. Insbesondere die Anordnung der Teilchen im Speicher des Rechners stellte sich als wichtige Einflussgröße heraus. Während sich eine cacheeffiziente Anordnung der Teilchen für das als „Kombinatorische Gebietszerlegung“ bezeichnete Verfahren mit wachsender Anzahl an Teilgebieten

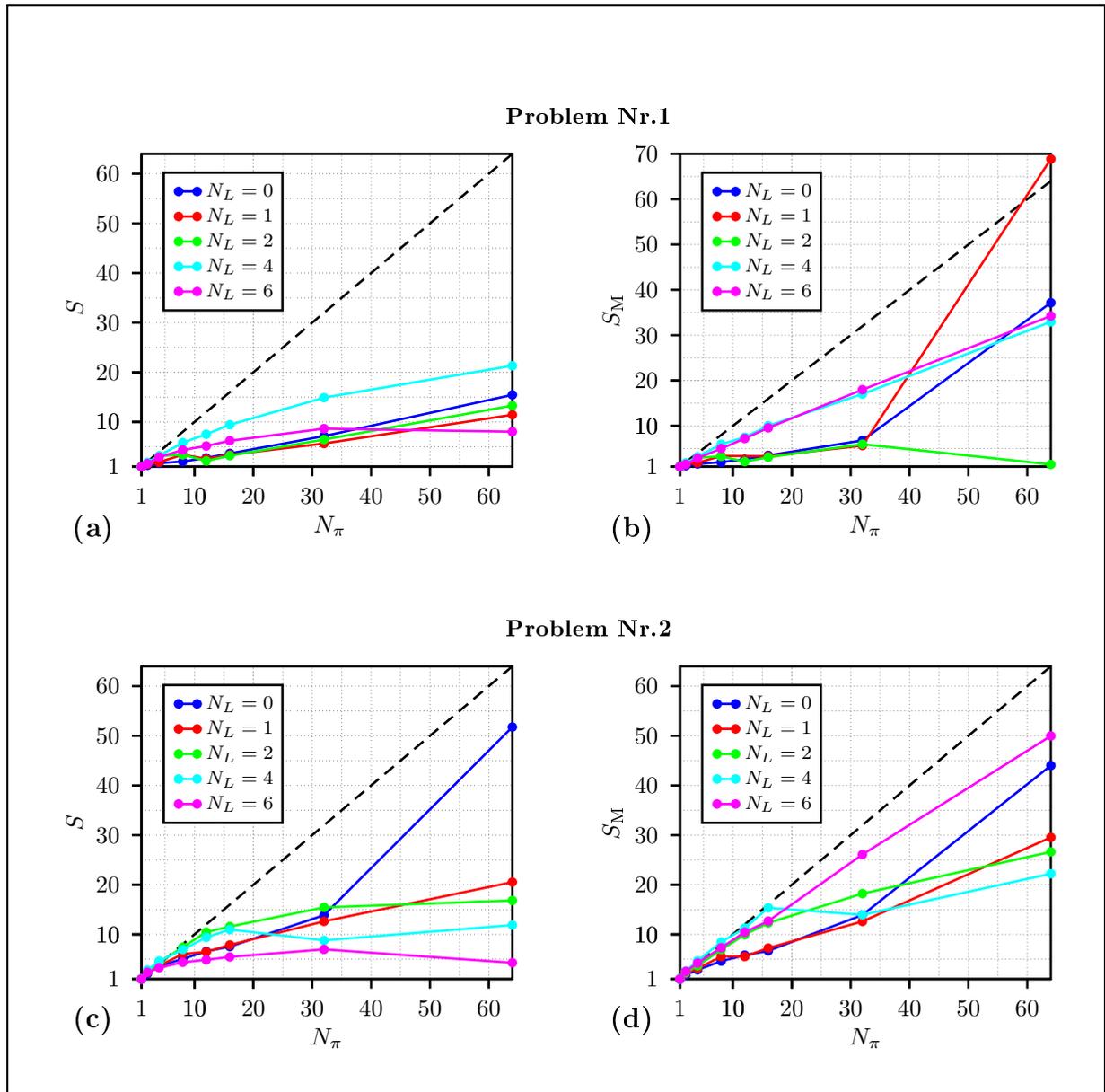
automatisch ergibt, ist dies für das in diesem Abschnitt betrachtete Verfahren nicht der Fall. Um die Lastbalancierungseigenschaften der beiden Verfahren vergleichen zu können, wurde daher bei der Generierung der Anfangsverteilung der Teilchen eine Sortierung gemäß der Position der Teilchen im Rechengebiet durchgeführt, so dass für beide Verfahren ein möglichst cacheeffizienter Zugriff auf die Daten möglich ist. Dadurch wurden die Effekte durch die Speicherhierarchie so weit wie möglich minimiert, womit ein direkter Vergleich der Lastbalancierungseigenschaften der beiden Verfahren möglich wird.

**Kombinatorische Gebietszerlegung** Abbildungen 4.7 und 4.8 zeigen die mit Hilfe der in Abschnitt 3.4.6.2 beschriebenen Methode erhaltenen Ergebnisse. Man beachte hierbei, dass die Vergleichssimulation auf einem einzigen Prozessor, welche zur Berechnung des Speedup verwendet wurde, identisch mit derjenigen ist, welche bereits für die im letzten Abschnitt gezeigten Speedup-Verläufe herangezogen wurde, da die beiden Verfahren für den Fall  $N_\pi = 1$  identisch sind. Somit ist durch den Vergleich der Speedup-Verläufe auch ein direkter Vergleich der Laufzeiten für die einzelnen Benchmarkprobleme möglich. Wie im letzten Abschnitt erläutert, wurden durch die Sortierung der Teilchen die Effekte der Speicherhierarchie abgeschwächt.

Die Kurven wurden für den Fall  $\sigma_{\max} = 0,2$  aufgetragen, da für diesen Wert des Entscheidungsparameters, wie bereits im vorangegangenen Abschnitt, in den meisten Fällen die besten Ergebnisse zu erzielen waren.

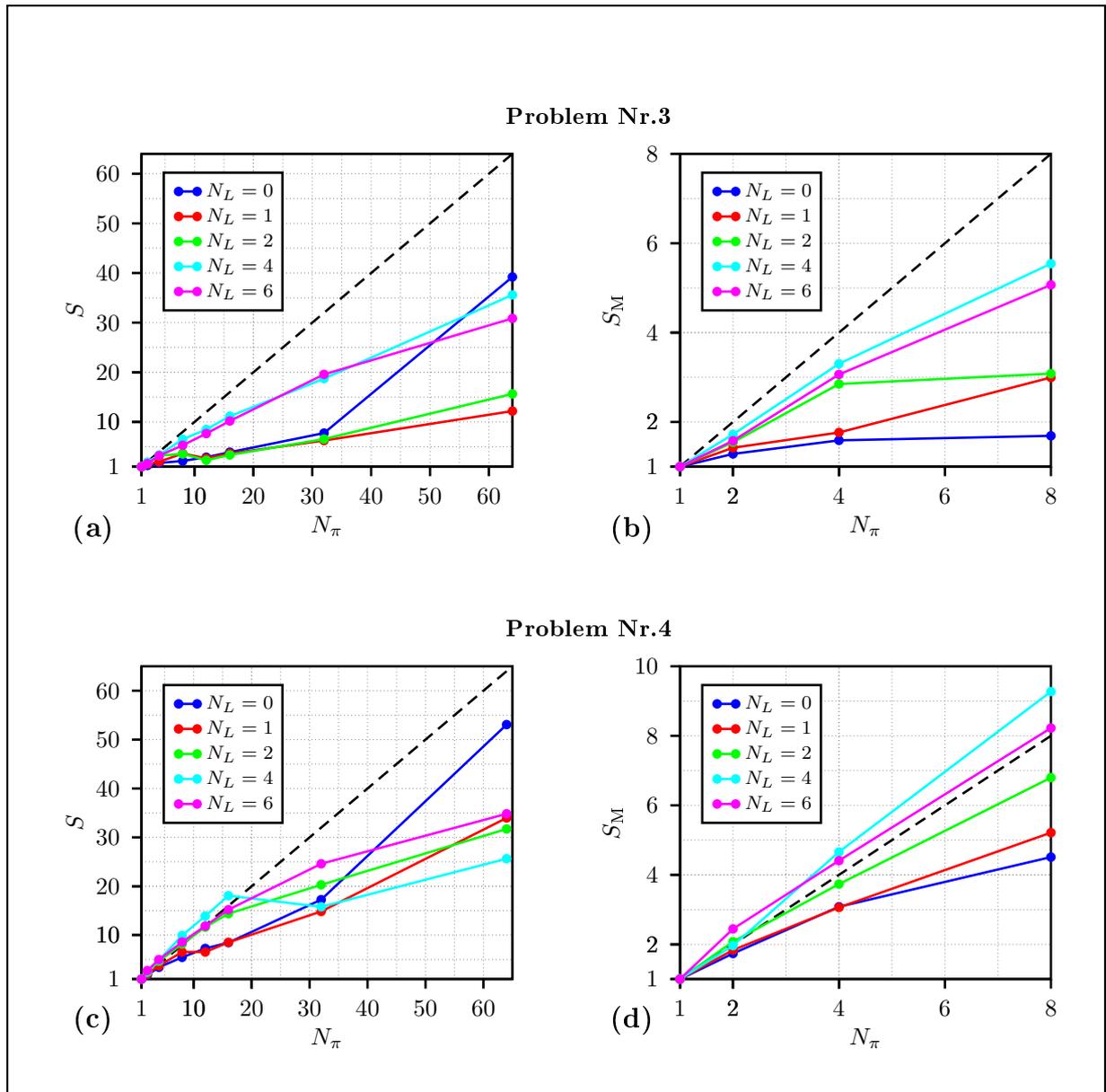
Wie schon bei der im letzten Abschnitt betrachteten Parallelisierungsstrategie zeigt sich auch hier eine etwas bessere Skalierbarkeit für die Benchmarkprobleme mit konzentrierter Teilchenverteilung (Problem Nr.2 und Nr.4), was wiederum darauf zurückzuführen ist, dass für diese Simulationen der dynamische Lastanteil, welcher durch die Teilchen bzw. die durch diese verursachten Berechnungen bedingt ist, im Vergleich zu dem statischen Lastanteil durch die Feldberechnungen niedriger ist im Vergleich zu den Benchmarkproblemen Nr.1 und Nr.3. Dadurch ist eine dynamische Lastbalancierung in diesem Fall weniger wichtig, um eine gute Skalierbarkeit zu erhalten.

Wie aus den theoretischen Überlegungen zu erwarten war, hängt die Skalierbarkeit wesentlich von der Anzahl der Teilgebiete pro Prozessor ab, durch deren Austausch eine Balancierung der Rechenlast erreicht werden kann. Da eine Erhöhung der Gebietsanzahl einerseits zu einer besseren Balancierung der Rechenlast, andererseits jedoch zu einem Ansteigen des Kommunikationsaufwandes führt, ist zu erwarten, dass die Laufzeit mit ansteigendem  $N_L$  zunächst abnimmt, einen Minimalwert erreicht und bei weiterer Erhöhung wieder ansteigt. Dieses Verhalten zeigt sich in den Abbildungen 4.7 und 4.8, wobei meist für  $N_L = 4$  oder  $N_L = 6$  die besten Ergebnisse zu erhalten waren.



**Abbildung 4.7:** Benchmarkergebnisse für die Parallelisierungsstrategie „Kombinatorische Gebietszerlegung“. (a) und (c) zeigen den Verlauf des gewöhnlichen Speedup  $S$ . (b) und (d) zeigen den Verlauf des skalierten Speedup  $S_M$  für die Benchmarkprobleme Nr. 1 und 2, welche in Tabelle 4.2 spezifiziert sind.

**Vergleich der Verfahren** Beim Vergleich der Speedup Verläufe und damit der relativen Laufzeiten für die beiden Parallelisierungsstrategien „Rekursive Koordinaten Bisektionierung“ sowie „Kombinatorische Gebietszerlegung“ zeigt sich, dass für die betrachteten Benchmarkprobleme die Laufzeiten für das letztgenannte Verfahren in fast allen Fällen



**Abbildung 4.8:** Benchmarkergebnisse für die Parallelisierungsstrategie „Kombinatorische Gebietszerlegung“. (a) und (c) zeigen den Verlauf des gewöhnlichen Speedup  $S$ . (b) und (d) zeigen den Verlauf des skalierten Speedup  $S_M$  für die Benchmarkprobleme Nr. 3 und 4, welche in Tabelle 4.2 spezifiziert sind.

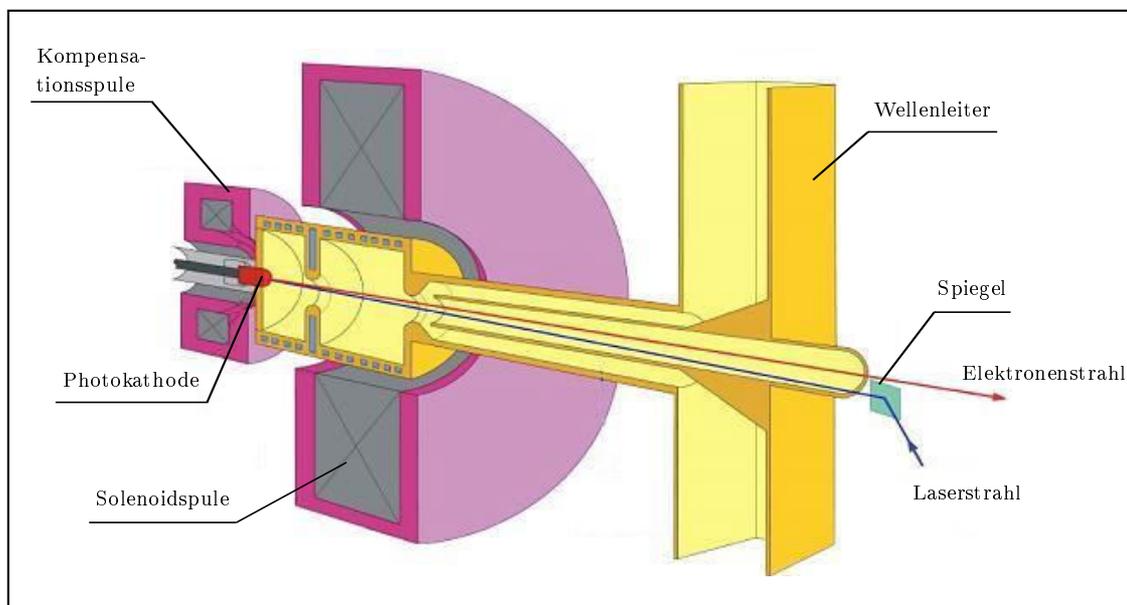
etwas schlechter ausfallen als die entsprechenden Laufzeiten bei Anwendung der ersten Strategie. Dies ist damit zu begründen, dass die Parallelisierungsstrategie „Kombinatorische Gebietszerlegung“ ihren Laufzeitvorteil, welcher sich durch das in Abbildung 3.14 illustrierte Löschen von Teilgebieten, die sich innerhalb von perfekt elektrisch leitendem

Material befinden, ergeben würde, nicht ausspielen kann, da das gesamte Rechengebiet als ein im Vakuum befindlicher Raumbereich angenommen wurde und folglich alle Teilgebiete für die Berechnungen relevant sind. Bei realen Anordnungen wie z.B. der in Abschnitt 4.2 gezeigten Elektronenkanone liegen häufig große Teile des quaderförmigen Rechengebietes außerhalb des relevanten Bereiches, und die entsprechenden Teilgebiete können gelöscht werden. Für solche Simulationsprobleme ist eine Überlegenheit der Parallelisierungsstrategie „Kombinatorische Gebietszerlegung“ zu erwarten.

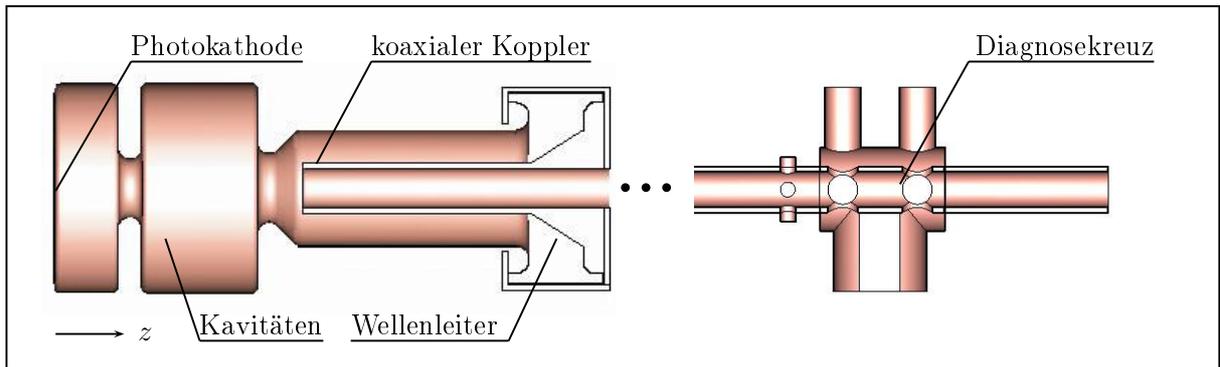
## 4.2 Der PITZ-Injektor

Der PITZ-Injektor wurde zum Studium von Elektronenkanonen und Photoinjektoren, welche hochfrequente elektromagnetische Felder zur Beschleunigung der Ladungsträger nutzen, erstellt [78]. Das spätere Ziel ist die Verwendung solcher Anordnungen als Elektronenquellen für Linearbeschleuniger (LINACs) und Freie-Elektronen-Laser. Abbildung 4.9 zeigt schematisch den Aufbau der Elektronenkanone des Teststandes.

Durch einen Laserstrahl wird die Photokathode angeregt, so dass ein Elektronenpaket aus dem Material herausgelöst wird. Die Elektronen werden durch das in den Kavitäten angeregte hochfrequente elektromagnetische Feld beschleunigt. Durch ein statisches



**Abbildung 4.9:** Die Skizze zeigt den prinzipiellen Aufbau des PITZ Injektors.



**Abbildung 4.10:** CAD-Modell des PITZ-Injektors sowie des Diagnosekreuzes.

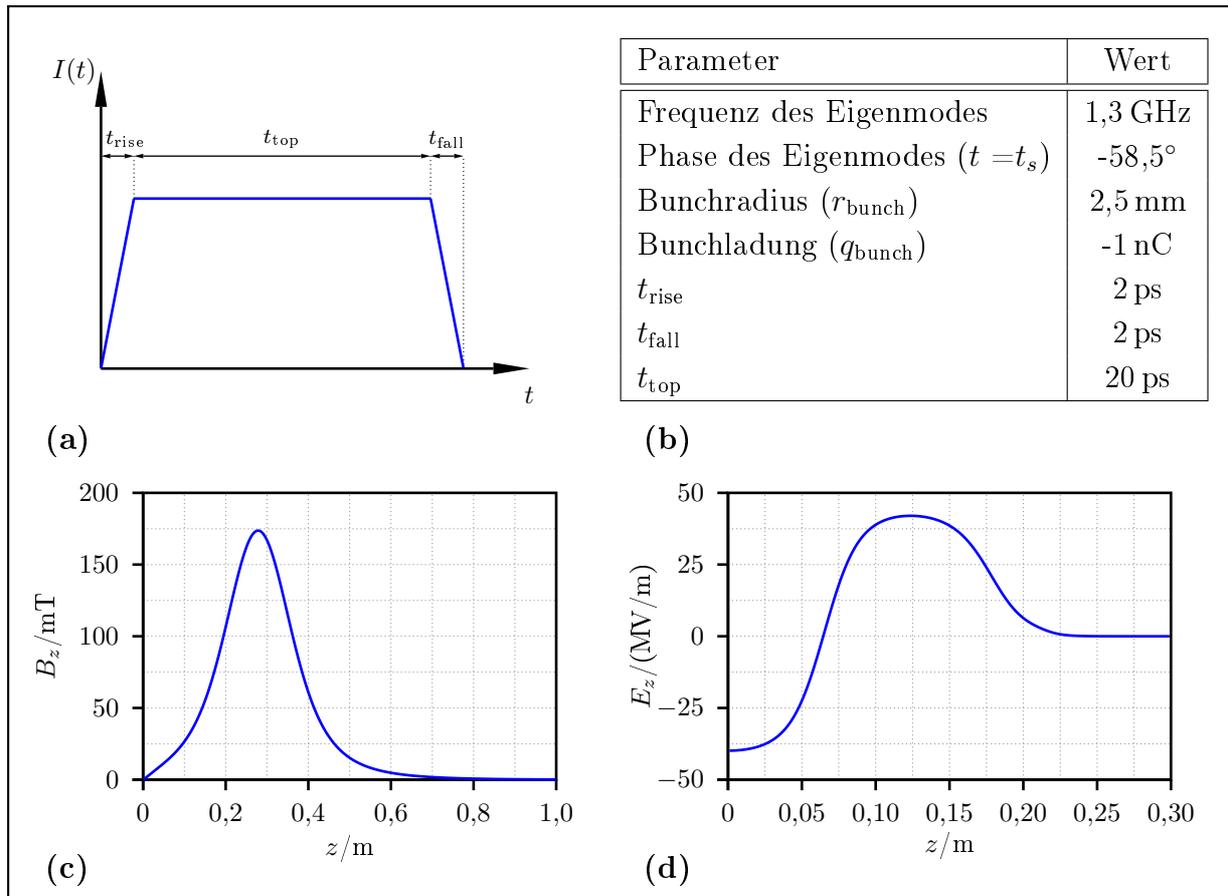
Magnetfeld, welches von den zwei Solenoidspulen erzeugt wird, wird die Aufweitung des Elektronenpaketes verhindert. Zum Design und zur Optimierung einer solchen komplexen Anordnung sind numerische Simulationen unerlässlich.

Durch Elemente, wie z.B. das in Abbildung 4.10 gezeigte Diagnosekreuz, wird die Rotationssymmetrie der Anordnung gebrochen, was eine 3D-PIC-Simulation zur zuverlässigen Berechnung der Teilchendynamik innerhalb des Injektors erforderlich macht. Erschwert wird diese Simulation dadurch, dass die Teilchendynamik nahe der Photokathode eine sehr komplexe Phasenraumdynamik aufweist, in der kurzskalige kollektive Effekte dominant sind. Numerische Tests zeigen, dass eine sehr hohe Auflösung dieses Bereiches mit einer Gitterschrittweite von etwa  $20\mu\text{m}$  in  $z$ -Richtung erforderlich ist, um die Effekte hinreichend genau zu erfassen [79]. Die wichtigsten Simulationsparameter für die durchgeführten Berechnungen sind in Abbildung 4.11(b) zusammengestellt.

Der Verlauf der  $z$ -Komponente des statischen Magnetfeldes, welches durch die Solenoid-Magneten erzeugt wird, entlang der Mittelachse ist in Abbildung 4.11(c) aufgetragen. Der Verlauf der  $z$ -Komponente des elektrischen Feldes für die Phase Null des zur Beschleunigung genutzten hochfrequenten Modes entlang der Mittelachse ist schließlich Abbildung 4.11(d) zu entnehmen. Der Beitrag des statischen Magnetfeldes sowie des beschleunigenden hochfrequenten elektromagnetischen Feldes zu dem Gesamtfeld am Ort der Teilchen wurde aus dem Verlauf der jeweiligen Felder entlang der Mittelachse mit Hilfe einer paraxialen Näherung bestimmt [80].

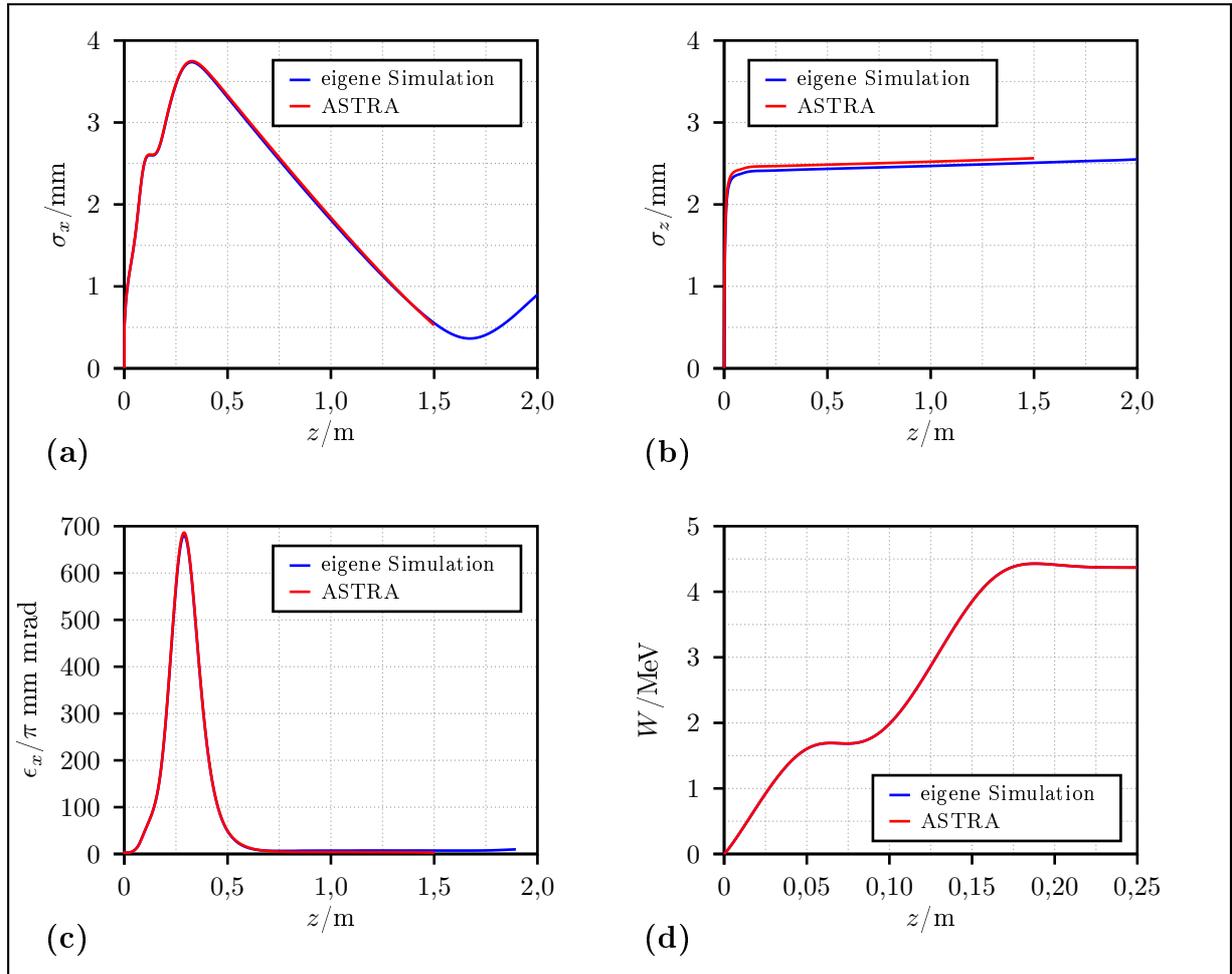
Die emittierte Ladung wird als gleichmäßig verteilt über einen kreisförmigen Bereich mit dem Radius  $r_{\text{bunch}}$  angenommen. Für den aus der Photokathode emittierten Strom wird ein Verlauf gemäß Abbildung 4.11(a) angenommen [81].

Mit Hilfe der im Rahmen der Arbeit entstandenen Software wurde die Teilchendynamik innerhalb der Struktur auf einer Länge von zwei Metern ab der Emissionsstelle simuliert. Die Diskretisierung des Rechengbietes mit hinreichender Genauigkeit führt zu einer Menge an



**Abbildung 4.11:** (a) zeigt den Zeitverlauf des Emissionsstromes als Funktion der Zeit. (b) zeigt die wichtigsten Simulationsparameter für die in Abbildung 4.12 gezeigten Simulationsergebnisse. (c) zeigt den Verlauf der  $z$ -Komponente der magnetischen Flussdichte, welche durch die Solenoid-Magnete erzeugt wird, entlang der Mittelachse. (d) zeigt den Verlauf der  $z$ -Komponente des elektrischen Feldes des beschleunigenden hochfrequenten Modes entlang der Mittelachse für die Phase Null.

Daten und Rechenoperationen, die sich nur mit Hilfe der Ressourcen eines Parallelrechners handhaben lässt. Es wurden Simulationen mit unterschiedlicher Anzahl an Gitterzellen sowie einer entsprechend angepassten Anzahl an Makroteilchen bis zu einer maximalen Problemgröße von insgesamt 650 Millionen Gitterzellen und 250.000 Teilchen durchgeführt, welche auf den in Abschnitt 4.1 genauer beschriebenen Parallelrechner mit 90 Knoten verteilt wurde. Abbildung 4.12 zeigt einige aus der Simulation hervorgegangene Ergebnisse sowie den Verlauf der entsprechenden Größen, der mit dem Simulationsprogramm ASTRA erhalten wurde [82], welche die erhaltenen Ergebnisse verifizieren sollen. Dieses Programm berechnet die Teilchendynamik unter einigen vereinfachenden Annahmen, die sich für die betrachtete Struktur jedoch als gerechtfertigt herausgestellt haben, was sich durch frühere



**Abbildung 4.12:** Simulationsergebnisse: (a) Transversaler RMS-Radius  $\sigma_x$ , (b) Longitudinaler RMS-Radius  $\sigma_z$ , (c) Transversale Emittanz  $\epsilon_x$ , (d) Energie  $W$  des Elektronenpaketes. Alle Größen sind über der Position des Schwerpunktes des Elektronenpaketes aufgetragen.

Vergleichsrechnungen mit dem kommerziellen PIC-Simulationsprogramm MAFIA sowie mit anderen PIC-Simulationsprogrammen, welche am Institut für Theorie Elektromagnetischer Felder der Technischen Universität Darmstadt entwickelt wurden, zeigte, die für den vorderen Teil der Struktur durchgeführt wurden [83, 84]. Als vereinfachende Annahme, welche den Berechnungen von ASTRA zugrunde liegt, ist insbesondere die Vernachlässigung der Materialverteilung zu nennen, wodurch der Einfluss der an den Materialgrenzflächen gestreuten elektromagnetischen Felder vernachlässigt wird. Eine Ausnahme bildet die Emissionsfläche, deren Einfluss mit Hilfe von Spiegelladungen modelliert wird. Eine weitere Annahme ist die Rotationssymmetrie aller Größen, d.h. weder Felder noch Teilchenverteilung besitzen eine azimutale Abhängigkeit.

Die Abbildungen 4.12(a) und (b) zeigen die transversale bzw. longitudinale Ausdehnung

des Elektronenpaketes beschrieben durch den RMS-Radius  $\sigma_x$  bzw.  $\sigma_z$ , welcher für ein Ensemble von  $P$  Teilchen definiert ist als

$$\sigma_x := \sqrt{\sum_{p=1}^P (\vec{r}_p \cdot \vec{e}_x - \vec{r} \cdot \vec{e}_x)^2} \quad \text{mit} \quad \vec{r} := \frac{1}{P} \cdot \sum_{p=1}^P \vec{r}_p, \quad (4.6)$$

während in Abb. 4.12(c) die transversale RMS-Emittanz  $\epsilon_x$ , definiert als [85]

$$\epsilon_x := \sqrt{\overline{r_x^2} \cdot \overline{p_x^2} - \overline{r_x \cdot p_x}^2}, \quad (4.7)$$

aufgetragen ist. Abbildung 4.12(d) zeigt schließlich die Energie des Elektronenpaketes. Alle Größen sind in Abhängigkeit von der Position des Schwerpunktes des Elektronenpaketes in  $z$ -Richtung aufgetragen. Da die Beschleunigung der Elektronen lediglich in den Kavitäten stattfindet und sich die Energie der Teilchen danach nicht mehr wesentlich verändert, wurde der Verlauf der Energie in Abbildung 4.12(d) lediglich für diesen Bereich aufgetragen.

Die mit Hilfe von ASTRA erhaltenen Simulationsergebnisse wurden bis zu einer Entfernung von  $z=1,5$  m ab der Emissionsfläche aufgetragen, da in diesem Bereich die Annahme der Rotationsymmetrie noch erfüllt ist. Die Ergebnisse zeigen eine gute Übereinstimmung in diesem Bereich.

Obwohl in dem gewählten Anwendungsbeispiel ein Simulationsprogramm wie ASTRA mit seinen stark vereinfachenden Annahmen zu gleichwertigen Ergebnissen wie ein 3D-PIC-Programm kommen kann, sei abschließend angemerkt, dass sich das entstandene 3D-PIC-Simulationsprogramm natürlich auf eine größere Klasse an Simulationsproblemen anwenden lässt als ein Programm wie ASTRA, dessen Anwendungsbereich auf Probleme beschränkt ist, für welche die oben beschriebenen vereinfachenden Annahmen das Ergebnis nicht wesentlich beeinflussen.



---

## 5 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden verschiedene Parallelisierungsstrategien des PIC Algorithmus für Multicomputer aus einem vereinfachten Maschinenmodell entwickelt, theoretisch analysiert sowie das Verhalten von Implementierungen der Strategien für repräsentativ ausgewählte Benchmarkprobleme untersucht. Zudem wurde anhand der Simulation einer Elektronenkanone, welche im Rahmen des PITZ Projektes Anwendung findet, gezeigt, dass die im Rahmen der Arbeit entstandene Software zur Lösung praxisrelevanter Problemstellungen geeignet ist, die sich aufgrund der hohen Anzahl an Freiheitsgraden, die für eine hinreichend genaue Rechnung notwendig sind, einer Simulation auf einem Einzelprozessorrechner entziehen.

Die Verteilung der Teilchen im Rechengebiet und das daraus resultierende Speicherzugriffsmuster stellte sich als entscheidend für die Auswahl einer geeigneten Parallelisierungsstrategie heraus. Im Rahmen der durchgeführten Untersuchungen zeigte sich, dass Parallelisierungsstrategien, welche auf einer Gebietszerlegung basieren, in Verbindung mit einer dynamischen Zuordnung der Rechenlast gute Ergebnisse liefern können. Jedoch erwies sich die Wahl des Entscheidungsparameters  $\sigma_{\max}$ , welcher die Durchführung der Lastbalancierungsoperation steuert, als kritisch für das Erreichen eines guten Performanceergebnisses. Eine zu häufige Neuordnung der Rechenlast führt aufgrund des dadurch eingeführten Overheads zu unbefriedigenden Ergebnissen, während eine zu große Wahl dazu führt, dass die Performance durch die ungleichmäßig verteilte Rechenlast vermindert wird.

Die als „kombinatorische Gebietszerlegung“ bezeichnete Parallelisierungsstrategie bietet aufgrund der flexiblen Möglichkeit der Diskretisierung für viele Simulationsprobleme eine interessante Möglichkeit, die Berechnungen zu beschleunigen. Bedingt durch die Art, wie die Rechenlast auf die Prozessoren verteilt werden kann, ist die Strategie jedoch für Simulationen mit räumlich stark lokalisierter Teilchenverteilung tendenziell weniger geeignet, da in diesem Fall die Lastbalancierung für die Trajektorienberechnung nicht zufriedenstellend gewährleistet werden kann.

Für Simulationsprobleme mit räumlich stark lokalisierter Teilchenverteilung wurde eine Parallelisierungsstrategie untersucht, die sich zwar als nicht skalierbar erwies, jedoch für ihren Anwendungsbereich recht gute Ergebnisse liefern kann.

Die vorliegende Arbeit beschränkte sich auf PIC Simulationen in Verbindung mit struk-

turierten, nicht adaptiven Gittern. Der Wunsch nach immer wirklichkeitsnäheren Simulationen hat jedoch die Simulationsprobleme, ebenso wie die verwendeten Algorithmen, immer komplexer werden lassen. Simulationen auf unstrukturierten und/oder adaptiven Gittern, die aufgrund der Größe der behandelten Probleme parallelisiert werden müssen, treten immer häufiger in Erscheinung und erfordern effiziente Parallelisierungsstrategien, die die von dem verwendeten Parallelrechner zur Verfügung gestellten Ressourcen zufriedenstellend ausnutzen. Obgleich Parallelisierungsstrategien zur Behandlung derartiger Simulationen vorgeschlagen wurden, fehlen für diese noch großteils Untersuchungen mit Hilfe aussagekräftiger Benchmarkprobleme, die eine Bewertung bezüglich der praktischen Einsetzbarkeit der Strategien ermöglichen, so dass an dieser Stelle noch weiterer Forschungsbedarf besteht.

Weiterhin bietet die Entwicklung hybrider Programmiermodelle [86] eine interessante Möglichkeit, dem seit einiger Zeit anhaltenden Trend zu Multicomputern, welche aus Multiprozessorrechnern aufgebaut sind, deren Prozessoren häufig auch noch mehrere Rechenkerne besitzen, zu begegnen und für diesen immer wichtiger werdenden Parallelrechnerotyp optimierte Parallelisierungsstrategien zu entwickeln.

---

# A Notation

## Mechanik/Elektrodynamik

$\Omega$	–	Rechengebiet
$\Gamma$	–	Phasenraum
$\varepsilon$	–	Permittivität
$\mu$	–	Permeabilität
$\kappa$	–	Leitfähigkeit
$c$	–	lokale Lichtgeschwindigkeit ( $c := \frac{1}{\sqrt{\varepsilon\mu}}$ )
$\vec{r}$	–	Ortsvektor
$t$	–	Zeit
$\vec{E}$	–	elektrische Feldstärke
$\vec{D}$	–	dielektrische Verschiebungsdichte
$\vec{H}$	–	magnetische Feldstärke
$\vec{B}$	–	magnetische Flussdichte
$\vec{J}$	–	elektrische Stromdichte
$\varrho$	–	elektrische Raumladungsdichte
$\vec{F}$	–	Kraft
W	–	Energie
$m_0$	–	Ruhemasse
$\gamma$	–	relativistischer Faktor ( $\gamma := 1/\sqrt{1 - (\frac{v}{c})^2}$ )
$m$	–	relativistische Masse ( $m := \gamma m_0$ )
$q$	–	elektrische Ladung
$\vec{v}$	–	Geschwindigkeit
$\vec{p}$	–	mechanischer Impuls ( $\vec{p} := m\vec{v}$ )

## Methode der finiten Integration/Particle-In-Cell

$\hat{e}$	– einzelne elektrische Gitterspannung
$\hat{\mathbf{d}}$	– einzelner elektrischer Gitterfluss
$\hat{h}$	– einzelne magnetische Gitterspannung
$\hat{\mathbf{b}}$	– einzelner magnetischer Gitterfluss
$\hat{j}$	– einzelner elektrischer Gitterstrom
$\mathbf{\hat{e}}$	– Vektor aller elektrischer Gitterspannungen
$\mathbf{\hat{d}}$	– Vektor aller elektrischer Gitterflüsse
$\mathbf{\hat{h}}$	– Vektor aller magnetischer Gitterspannungen
$\mathbf{\hat{b}}$	– Vektor aller magnetischer Gitterflüsse
$\mathbf{\hat{j}}$	– Vektor aller elektrischer Gitterströme
$\mathbf{D}_\varepsilon$	– Materialmatrix, verbindet $\mathbf{\hat{d}}$ und $\mathbf{\hat{e}}$ über $\mathbf{\hat{d}} = \mathbf{D}_\varepsilon \cdot \mathbf{\hat{e}}$
$\mathbf{M}_{\mu^{-1}}$	– Materialmatrix, verbindet $\mathbf{\hat{h}}$ und $\mathbf{\hat{b}}$ über $\mathbf{\hat{h}} = \mathbf{M}_{\mu^{-1}} \cdot \mathbf{\hat{b}}$
$\mathbf{C}, \tilde{\mathbf{C}}$	– diskrete Rotationsoperatoren (primäres bzw. duales Gitter)
$\mathbf{S}, \tilde{\mathbf{S}}$	– diskrete Divergenzoperatoren (primäres bzw. duales Gitter)
$L, \tilde{L}$	– Länge einer primären bzw. dualen Gitterkante
$A, \tilde{A}$	– Flächeninhalt einer primären bzw. dualen Gitterfläche
$V, \tilde{V}$	– Volumen einer primären bzw. dualen Gitterzelle
$N_c$	– Gesamtzahl der Gitterzellen der räumlichen Diskretisierung
$\mathbf{i}_c$	– Kurzschreibweise für das Indextripel $(i, j, k)$
$\mathcal{I}$	– Menge aller zulässigen Indextripel $\mathbf{i}_c$
$P$	– Anzahl der Makroteilchen
$\Delta t$	– Zeitschritt
$(\cdot)^{(m)}$	– bezeichnet die entsprechenden Größen zum Zeitschritt $m$

## Bewertung paralleler Algorithmen

$T$	– Laufzeit
$S$	– paralleler Speedup
$S_M$	– speicherskalierter Speedup
$E$	– parallele Effizienz
$\overline{E}_\varepsilon$	– Isoeffizienzfunktion

## Parallelisierungsstrategien für PIC

$\mathcal{S}$	– Parallelisierungsstrategie
$\mathcal{P}$	– Simulationsproblem
$\mathcal{I}$	– Menge aller Gitterzellen beschrieben durch ihre Indextripel $(i, j, k) \in \mathbb{N}^3$
$B$	– Bounding Box (ein quaderförmiger Gitterausschnitt aus einem kartesischen Gitter)
$N_\pi$	– Gesamtanzahl der für eine parallele Rechnung verwendeten Prozessoren
$\Pi$	– Menge aller Prozessoren $\Pi := \{1 \dots N_\pi\}$ , wobei jeder Prozessor durch eine eindeutige Zahl bezeichnet wird.
$T_{\mathcal{S}}$	– Gesamtlaufzeit der Simulation bei Einsatz von Parallelisierungsstrategie $\mathcal{S}$
$T_{\mathcal{S}}^{(m)}$	– Gesamtlaufzeit des Zeitschrittes $m$ bei Einsatz von Parallelisierungsstrategie $\mathcal{S}$
$F T_{\mathcal{S}}^{(m)}$	– Laufzeit für die Zeitintegration der Gitterspannungen in Zeitschritt $m$
$F \tilde{T}_{\mathcal{S}}^{(m)}$	– Laufzeit für den im Rahmen des Feldlösers benötigten Datenaustausch zwischen den Prozessoren in Zeitschritt $m$
$P T_{\mathcal{S}}^{(m)}$	– Laufzeit für die zur Trajektorienberechnung der Makroteilchen benötigten Rechenschritte in Zeitschritt $m$
$P \tilde{T}_{\mathcal{S}}^{(m)}$	– Laufzeit für den im Rahmen der Trajektorienberechnung benötigten Datenaustausch in Zeitschritt $m$
$LB \tilde{T}_{\mathcal{S}}^{(m)}$	– Laufzeit für die in Zeitschritt $m$ durchgeführte Neuordnung der Rechenlast
$\overset{\text{cp}}{n}_{\mathbf{i}_c, p}^{(m)}$	– gibt an, ob sich Teilchen $p$ in Zeitschritt $m$ in Gitterzelle $\mathbf{i}_c$ befindet.
$\overset{\text{pp}}{a}_{p, i_\pi}^{(m)}$	– Entscheidungsvariablen zur Beschreibung der Zuordnung von Teilchen zu Prozessoren
$\overset{\text{cp}}{a}_{\mathbf{i}_c, i_\pi}^{(m)}$	– Entscheidungsvariablen zur Beschreibung der Zuordnung von Gitterzellen zu Prozessoren
$\overset{n\pi}{a}^{(m)}_{i_\pi, j_\pi}$	– Entscheidungsvariable. Gibt an, wie viele Teilchen in Zeitschritt $m$ Prozessor $i_\pi$ zugeordnet sind, für die Berechnung deren Trajektorien Felddaten von Prozessor $j_\pi$ benötigt werden.
$\overset{g\pi}{a}_{i_g, i_\pi}^{(m)}$	– Entscheidungsvariable. Gibt die Zuordnung von Teilgebieten zu Prozessoren an.
$G$	– Gesamtanzahl an Teilgebieten
$G_{i_\pi}$	– Anzahl der Teilgebiete, die Prozessor $i_\pi$ zugeordnet werden.

- $\lceil x \rceil$  – Aufrunden auf die nächst größere Ganzzahl gemäß  $\lceil x \rceil := \min_{k \in \mathbb{Z}, k \geq x} \{k\}$ .
- $\lfloor x \rfloor$  – Abrunden auf die nächst kleinere Ganzzahl gemäß  $\lfloor x \rfloor := \max_{k \in \mathbb{Z}, k \leq x} \{k\}$ .
- $\alpha_{i_\pi}$  – Anzahl der benötigten Zeiteinheiten zur Durchführung der Zeitintegration der Feldfreiheitsgrade pro Zeitschritt und Zelle für Prozessor  $i_\pi$ .
- $\beta_{i_\pi}$  – Anzahl der benötigten Zeiteinheiten zur Durchführung der Trajektorienberechnung pro Zeitschritt und Teilchen für Prozessor  $i_\pi$ .
- $\gamma$  – Proportionalitätsfaktor zur Beschreibung der benötigten Zeit für den Datenaustausch zwischen zwei Prozessoren.
- $\sigma_{\max}$  – Entscheidungsparameter zur Steuerung der Lastbalancierung.
- $N_L$  – Anzahl der Partitionierungsebenen bei der Erstellung der Teilgebiete für die Parallelisierungsstrategie "kombinatorische Gebietszerlegung".
- $G_{i_\pi}$  – Anzahl der Teilgebiete, welche Prozessor  $i_\pi$  zugeordnet sind.

---

## B Abkürzungen

BSPC	—	<b>B</b> ulk- <b>S</b> ynchronous <b>P</b> arallel <b>C</b> omputer
CPU	—	<b>C</b> entral <b>P</b> rocessing <b>U</b> nit
DESY	—	<b>D</b> eutsches <b>E</b> lektronen <b>S</b> ynchrotron
FEL	—	<b>F</b> reie <b>E</b> lektronen <b>L</b> aser
FIT	—	<b>F</b> inite <b>I</b> ntegration <b>T</b> heory (deutsch: Methode der finiten Integration)
MIMD	—	<b>M</b> ultiple <b>I</b> nstruction Stream, <b>M</b> ultiple <b>D</b> ata Stream
PIC	—	<b>P</b> article- <b>I</b> n- <b>C</b> ell
PITZ	—	<b>P</b> hotoinjektor <b>T</b> eststand in <b>Z</b> euthen
PRAM	—	<b>P</b> arallel <b>R</b> andom <b>A</b> ccess <b>M</b> achine
RAM	—	<b>R</b> andom <b>A</b> ccess <b>M</b> achine oder <b>R</b> andom <b>A</b> ccess <b>M</b> emory (kontextabhängig)
RCB	—	<b>R</b> ecursive <b>C</b> oordinate <b>B</b> isection (deutsch: Rekursive Koordinaten Bisektionierung)
SIMD	—	<b>S</b> ingle <b>I</b> nstruction Stream, <b>M</b> ultiple <b>D</b> ata Stream
SISD	—	<b>S</b> ingle <b>I</b> nstruction Stream, <b>S</b> ingle <b>D</b> ata Stream
SPMD	—	<b>S</b> ingle <b>P</b> rogram <b>M</b> ultiple <b>D</b> ata



---

# C Literaturübersicht

Das bei der Parallelisierung eines Algorithmus auftretende Problem der Zuordnung von Rechenoperationen auf die Prozessoren eines Parallelrechners hat in den vergangenen zwei Jahrzehnten viel Beachtung gefunden. Das folgende Kapitel stellt eine Würdigung der in der Literatur untersuchten Parallelisierungsstrategien für PIC Simulationen dar.

## C.1 Parallelisierungsstrategien für Feldlöser

Das Problem, die Gitterzellen eines Rechengitters einer Anzahl von Prozessoren zuzuordnen, so dass die mit den Gitterzellen verknüpften Berechnungen in möglichst kurzer Zeit ausgeführt werden können, ist wohlbekannt. Die mathematische Formulierung dieses Problems führt je nach Modellierung von Rechen- und Kommunikationszeit auf unterschiedliche kombinatorische Optimierungsprobleme, zu deren Lösung ausschließlich Heuristiken eingesetzt werden. Die Gründe, weshalb man sich mit Heuristiken zufrieden gibt, sind zweierlei.

Der erste Grund ist die Schwierigkeit, einen effizienten Algorithmus zu finden, der die exakte Lösung des Optimierungsproblems liefert. Von BOKHARI wurde gezeigt, dass es sich bei dem Lastzuordnungsproblem in seiner populärsten Formulierung (siehe Abschnitt C.1.1) um ein NP-vollständiges Problem handelt [87]. Für NP-vollständige Probleme existiert kein Algorithmus, welcher das Problem exakt löst und dessen Laufzeit eine polynomiale Komplexität aufweist [65]. Vielmehr wächst die benötigte Laufzeit in Abhängigkeit der Problemgröße schneller als jedes Polynom. Somit führt bereits die exakte Lösung eines Problems relativ geringer Größe zu unverträglich langen Laufzeiten.

Der zweite Grund ist, dass die zur Verfügung stehenden Maschinen- und Netzwerkmodelle, mit deren Hilfe die Optimierungsprobleme konstruiert werden, ohnehin nur eine grobe Vorhersage der Laufzeit erlauben. Details der Implementierung, der Maschinenarchitektur, der verwendeten Netzwerkprotokolle etc. und deren Einfluss auf die Laufzeit können nicht berücksichtigt werden (siehe Abschnitt 2.4.5). Eine exakte Lösung der aus diesen relativ groben Modellen hervorgegangenen Optimierungsprobleme erscheint vor diesem Hintergrund nicht unbedingt notwendig, um eine zufriedenstellende Performance zu erreichen.

Das Augenmerk richtet sich also auf die Entwicklung von Heuristiken, die in vertretbarer Zeit eine hinreichend gute Partitionierung finden.

Beim Studium der relevanten Literatur fällt auf, dass für unterschiedliche Gittertypen auch sehr verschiedene Partitionierungsheuristiken entwickelt wurden. Diese lassen sich unterteilen in geometrische und graphenbasierte Heuristiken. Obwohl prinzipiell alle Heuristiken zur Partitionierung beliebiger Gitter anwendbar sind, werden die komplizierteren aber auch flexibleren graphenbasierten Heuristiken eher für unstrukturierte und die einfacheren geometrischen Heuristiken eher für strukturierte Gitter eingesetzt, bei denen sie zu guten Ergebnissen führen. Im Folgenden kann nur eine kurze Übersicht über die populärsten Heuristiken gegeben werden. Für einen ausführlichen Überblick und viele weitere Literaturverweise zum Thema der Gitterpartitionierung sei auf [71] verwiesen.

### C.1.1 Graphenbasierte Heuristiken

Graphenbasierte Heuristiken bilden das Problem der Lastzuordnung auf ein Graphenpartitionierungsproblem ab. Die Knoten eines Graphen stehen dabei für die Objekte, üblicherweise die Gitterzellen, welche den Prozessoren des Parallelrechners zugeordnet werden sollen, während seine Kanten die Abhängigkeiten zwischen den Objekten, und somit den Datenaustausch, modellieren. Mathematisch lässt sich das in der Literatur im Rahmen der Gitterpartitionierung meist verwendete Graphenpartitionierungsproblem wie folgt formulieren [71].

**gegeben:** Ein gewichteter und ungerichteter Graph  $G = (V, E)$  dessen Knoten jeweils Gewichte  $w(v)$ ,  $v \in V$  und dessen Kanten Gewichte  $l(e)$ ,  $e \in E$  besitzen.

**gesucht:** Eine Zerlegung der Knotenmenge in  $N_\pi$  disjunkte Teilmengen  $V_{1..N_\pi}$ , so dass die Summe der Knotengewichte für jede der Teilmengen einen Wert von  $\frac{1}{N_\pi} \sum_{v \in V} w(v) + \varepsilon$  nicht überschreitet, wobei die Summe der Kantengewichte für diejenigen Kanten, welche zwei Knoten aus unterschiedlichen Teilmengen miteinander verbinden, minimal sein soll. Dabei ist  $\varepsilon$  ein sinnvoll zu wählender Parameter. Im einfachsten Fall, dass mit jeder Gitterzelle die gleichen Berechnungsoperationen und für die voneinander abhängigen Zellen die gleichen Operationen für den Datenaustausch durchgeführt werden müssen, ergeben sich die gleichen Gewichte für alle Knoten bzw. Kanten des Graphen.

Dieses Graphenpartitionierungsproblem ist NP-vollständig und kann daher für praktisch interessante Problemgrößen nur heuristisch gelöst werden [87, 65]. Die Idee bei der Konstruktion dieses Graphenpartitionierungsproblems ist es, alle Partitionierungen zu betrachten, die im Falle der Vernachlässigung des Kommunikationsaufwandes die Laufzeit mini-

mieren (Nebenbedingung). Dies sind alle Partitionierungen, bei denen die Gitterzellen gleichmäßig auf die verfügbaren Prozessoren verteilt werden. Innerhalb der Menge dieser Partitionierungen wird dann diejenige gesucht, bei der die über alle Prozessoren aufsummierte Kommunikationszeit, modelliert durch die Anzahl der Kanten, die Knoten aus verschiedenen Knoten miteinander verbinden, minimal wird.

Obwohl eine Partitionierung gemäß der Lösung dieses Graphenpartitionierungsproblems nicht unbedingt die Laufzeit minimiert [88] und ein Versuch zur Lösung des Lastzuordnungsproblems mit Hilfe von Heuristiken zur Lösung des Graphenpartitionierungsproblems somit zunächst fragwürdig erscheint, hat sich dieses Vorgehen für praktische Problemstellungen doch bewährt und liefert hinreichend gute Ergebnisse [71].

Insbesondere Multi-Level Verfahren und Spektralverfahren haben bei der heuristischen Lösung des Lastzuordnungsproblems für Berechnungen auf unstrukturierten Gittern eine große Verbreitung gefunden [89, 90, 91]. Multi-Level Verfahren bestehen aus drei Schritten. Im ersten Schritt wird die Knotenanzahl des zu partitionierenden Graphen durch wiederholtes Zusammenfassen von Knoten zu Großknoten reduziert. Die Zusammenfassung der Knoten und der entstehenden Großknoten wird entweder so lange durchgeführt, bis der resultierende Graph nur noch  $N_\pi$  Großknoten besitzt und somit in trivialer Weise partitioniert werden kann, oder bis die Anzahl der Knoten zumindest so weit abgenommen hat, dass andere Algorithmen zur Partitionierung eingesetzt werden können, die für den Gesamtgraphen entweder zu schlechte Ergebnisse liefern würden oder zu aufwändig wären. Nach der Partitionierung werden die Großknoten schrittweise wieder in ihre Bestandteile zerlegt, wobei überprüft wird, ob durch Austausch von Knoten zwischen den Partitionen der Wert der Zielfunktion verbessert werden kann. Die vorgeschlagenen Heuristiken unterscheiden sich in der Wahl des Algorithmus zur Knotenzusammenfassung, der Berechnung der Partitionierung für den größten Graphen und dem Algorithmus zur Auswahl der Knoten, welche zwischen den Partitionen ausgetauscht werden sollen.

Ebenso sind parallele Multi-Level Partitionierungsalgorithmen entwickelt worden, die die parallele Abarbeitung der oben beschriebenen Schritte erlauben [92, 93].

Neben den Multi-Level Verfahren haben Spektralverfahren eine gewisse Verbreitung gefunden [94]. Spektralverfahren konstruieren eine Partition des Graphen basierend auf dem Eigenvektor zu dem zweitgrößten Eigenwert der LAPLACE Matrix  $\mathbf{L}_G$  des Graphen, deren Einträge wie folgt definiert sind:

$$(\mathbf{L}_G)_{ij} := \begin{cases} 1 & \text{falls } i \neq j \text{ und } \{v_i; v_j\} \in E \\ -\text{deg}(v_i) & \text{falls } i = j \\ 0 & \text{sonst.} \end{cases} \quad (\text{C.1})$$

Der Eigenvektor erlaubt Rückschlüsse darüber wie die Knoten des Graphen miteinander verbunden sind. Gemäß diesen Informationen werden die Knoten den einzelnen Partitionen zugeordnet. Für algorithmische Details sei auf [94] verwiesen.

### C.1.2 Geometrische Heuristiken

Geometrische Heuristiken verwenden zur Partitionierung eines Gitters die Position der Gitterzellen im Raum als zusätzliche Information. Populär sind geometrische Verfahren insbesondere für die Partitionierung strukturierter Gitter geworden. Für deren vergleichsweise einfache Struktur führen sie zu guten Ergebnissen und sind im Vergleich zu graphenbasierten Methoden einfach und schnell. Zudem sind die konstruierten Partitionen häufig von einfacher Form und ermöglichen so die Verwendung ähnlich einfacher Datenstrukturen wie im seriellen Fall, was einen nicht zu unterschätzenden Vorteil darstellt. Aus den genannten Gründen wurde auch für die im Rahmen der vorliegenden Arbeit vorgestellten Parallelisierungsstrategien eine geometrische Heuristik ausgewählt.

Im Fall eines strukturierten, kartesischen Gitters liefert eine einfache Partitionierung, welche das Gitter in Quader gleicher Größe unterteilt, bereits gute Ergebnisse [95]. Die Gebiete ergeben sich bei dieser Partitionierungsheuristik als Tensorprodukt eindimensionaler Partitionierungen in den einzelnen Koordinatenrichtungen. Eine solche Partitionierung ist jedoch aufgrund ihrer fehlenden Flexibilität nur zur statischen Lastzuordnung geeignet.

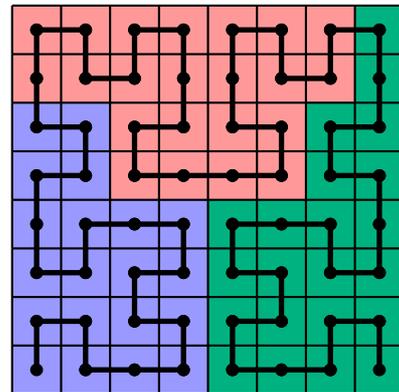
Viele geometrische Verfahren erzeugen eine Partitionierung mittels einer rekursiven Bisektionierung, d.h. eine Zerlegung in  $N_\pi$  Partitionen erfolgt durch eine wiederholte Partitionierung in zwei Partitionen (Bisektionierung). Dies wird so lange durchgeführt, bis die benötigten  $N_\pi$  Gebiete entstanden sind. Obwohl die Lösung des Partitionierungsproblems nicht äquivalent zur Lösung einer Reihe von Bisektionierungsproblemen ist und selbst bei optimalen Bisektionierungen das Ergebnis beliebig schlecht im Vergleich zur Lösung des ursprünglichen Partitionierungsproblems sein kann [70], zeigt sich doch, dass Bisektionierungsverfahren insbesondere bei der Partitionierung strukturierter Gitter Ergebnisse liefern, die durchaus praktikabel sind (siehe Ergebnisse in Kapitel 4). Die Rekursive Koordinaten Bisektionierung (RCB) [66], wie sie in Abschnitt 3.3.1 in verallgemeinerter Form erläutert wurde, ist die wohl bekannteste Variante des Bisektionierungsansatzes.

Eine Verallgemeinerung der RCB Heuristik wurde von NOUR-OMID vorgeschlagen und wird als **R**ecursive **I**nertial **B**isection (RIB) bezeichnet [96]. Jede Gitterzelle wird dabei als Massenpunkt interpretiert. Das Rechengitter wird somit zu einer Verteilung von Massenpunkten im Raum. Von dieser Verteilung wird die Achse berechnet, bezüglich derer die Verteilung das kleinste Trägheitsmoment besitzt. Diese Achse gibt die Normalenrichtung der Partitionierungsebene vor. Im Fall eines strukturierten kartesischen Gitters, welches nach den Koordinatenachsen ausgerichtet ist, fällt diese Achse immer mit einer Koordinatenachse zusammen, und führt somit zum gleichen Ergebnis wie die RCB Heuristik in Abschnitt 3.3.1.

Die Einschränkung bezüglich der Form der Partitionen, kann dazu führen, dass eine suboptimale Verteilung der Gitterzellen auf die Partitionen erfolgt. Eine Verallgemeinerung des

Verfahrens, welche die Randzellen einer Partition noch einmal gesondert zuordnet, findet sich z.B. in [69]. Dies ist jedoch eher für kompliziertere Gitter (z.B. strukturierte Gitter mit lokaler Gitterverfeinerung) von Interesse, die ohnehin kompliziertere Datenstrukturen erfordern.

Insbesondere zur Parallelisierung von Berechnungen auf strukturierten Gittern mit lokaler Gitterverfeinerung hat die Partitionierung mittels raumfüllender Kurven (engl. spacefilling Curves) Popularität erlangt [97, 98]. Dabei wird das Problem der Gitterpartitionierung auf das Problem der Partitionierung einer Liste, d.h. ein eindimensionales Partitionierungsproblem, zurückgeführt. Die Anordnung der Gitterzellen in der Liste erfolgt mit Hilfe einer Kurve, welche mit einer bestimmten Bildungsvorschrift alle Gitterzellen traversiert. Die Liste wird dann in  $N_\pi$  zusammenhängende Teile zerlegt. Alle Gitterzellen innerhalb eines Listenteils werden jeweils einer Partition zugeordnet. Raumfüllende Kurven besitzen die Eigenschaft, dass Gitterzellen, welche im zwei- oder dreidimensionalen Rechengitter nahe beieinander liegen, tendenziell auch in der resultierenden Liste nahe beieinander liegen [99]. Aufgrund dieser Eigenschaft führt die Partitionierung mittels raumfüllender Kurven zu der Zuordnung vieler im Gitter benachbarter Zellen zu derselben Partition und damit zu einem relativ geringen Kommunikationsaufwand. Die am häufigsten verwendete Kurve ist die HILBERT Kurve [100], welche in Abbildung C.1 für ein einfaches Gitter abgebildet ist. Andere raumfüllende Kurven finden jedoch auch vereinzelt Anwendung (siehe z.B. [101]). Dadurch, dass die Zuordnung der Gitterzellen zu den Partitionen zellweise erfolgt, kann die Last sehr gleichmäßig auf die Prozessoren verteilt werden. Allerdings wird dies durch, die im Vergleich zur RCB Heuristik, größere Anzahl an Randzellen und damit einen höheren Kommunikationsaufwand erkaufte [98].



**Abbildung C.1:** *Partitionierung eines kartesischen Gitters mit Hilfe der Hilbert Kurve.*

## C.2 Parallelisierungsstrategien für PIC

Strategien zur Parallelisierung des PIC Algorithmus können im Wesentlichen nach zwei Gesichtspunkten klassifiziert werden. Erstens danach, ob und wie eine Kopplung zwischen der Zuordnung der Gitterzellen und der Zuordnung der Teilchen vorgenommen wird und

zweitens, ob diese Zuordnung im Rahmen einer dynamischen Lastbalancierung während der Rechnung modifiziert wird oder nicht. Eine recht gute Zusammenfassung über die verschiedenen Gesichtspunkte, die bei der Konstruktion einer Parallelisierungsstrategie für PIC eine Rolle spielen, findet sich in [15].

Im folgenden Abschnitt sollen nur solche Parallelisierungsstrategien Erwähnung finden, für die aus der Veröffentlichung ersichtlich ist, dass Implementierungen existieren.

## C.2.1 Parallelisierungsstrategien mit statischer Lastbalancierung

### C.2.1.1 Ungekoppelte Zuordnung von Gitterzellen und Teilchen

In [1, 2, 3] werden Parallelisierungsstrategien vorgeschlagen, welche die Teilchen ungeachtet ihrer Position im Gitter gleichmäßig auf die verfügbaren Prozessoren verteilen. Es ist somit keine Kopplung der Zuordnung von Teilchen und Gitterzellen vorhanden. Die Zuordnung der Gitterzellen geschieht in räumlich zusammenhängenden Blöcken, ähnlich einer strukturierten geometrischen Partitionierung.

Da sowohl die Teilchen wie auch die Gitterzellen gleichmäßig auf die verfügbaren Prozessoren zugeordnet werden, ist die Rechenlast zu jedem Zeitpunkt ideal balanciert und es gilt

$$\begin{aligned} {}^F T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) &= {}^F T_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \quad \forall i_\pi, j_\pi \in \Pi, \\ {}^P T_S^{(m)}(i_\pi, N_\pi, \mathcal{P}) &= {}^P T_S^{(m)}(j_\pi, N_\pi, \mathcal{P}) \quad \forall i_\pi, j_\pi \in \Pi. \end{aligned}$$

Die Zuordnung der Teilchen und Gitterzellen wird während der Simulation nicht verändert. In den jeweiligen Benchmarkuntersuchungen wurden Simulationsprobleme betrachtet, bei denen die Teilchen während der gesamten Simulation gleichmäßig über das Rechengebiet verteilt sind.

Die Ergebnisse zeigen jeweils einen sehr starken Overhead verursacht durch den notwendigen Datenaustausch. Um die Zeit für den Datenaustausch zu reduzieren, schlugen die Autoren in [1] vor, die Zuordnung der Teilchen ganz oder teilweise basierend auf der Gebietszerlegung für den Feldlöser vorzunehmen. Bei einer vollständigen Kopplung der Teilchenzuordnung mit der Gebietszerlegung erreichten die Autoren für ihr Benchmark-Beispiel deutlich reduzierte Laufzeiten für die Feldinterpolation sowie für die Stromextrapolation. Die Laufzeiten für diese Operationen waren bei der Implementierung der Autoren nun in der gleichen Größenordnung wie die Integration der Teilchentrajektorien.

Eine weitere Parallelisierungsstrategie, bei der die Teilchen gleichmäßig auf die Prozessoren verteilt werden, wird in [72, 73, 4] vorgeschlagen. Es handelt sich dabei um das in Abschnitt 3.4.4 theoretisch untersuchte Verfahren. Wie die theoretische Untersuchung

zeigte, ist diese Parallelisierungsstrategie nicht skalierbar, kann aber für Simulationsprobleme mit räumlich stark konzentrierten Teilchenverteilungen zu recht zufriedenstellenden Ergebnissen führen.

### C.2.1.2 Gekoppelte Zuordnung von Gitterzellen und Teilchen

Parallelisierungsstrategien, welche die Zuordnung der Teilchen direkt an die Zuordnung der Gitterzellen koppeln, sind in der Literatur am häufigsten zu finden (siehe z.B. [5, 6, 7, 8]). Für Simulationsprobleme, bei denen die Teilchenverteilung zumindest annähernd homogen ist, führen sie zu guten Ergebnissen. Für Simulationsprobleme mit räumlich konzentrierten Teilchenverteilungen leidet diese Parallelisierungsstrategie jedoch an der ungleichmäßigen Verteilung der Rechenlast auf die Prozessoren (siehe auch die Skalierbarkeitsuntersuchung in Abschnitt 3.4.5).

## C.2.2 Parallelisierungsstrategien mit dynamischer Lastbalancierung

In [13, 14, 15, 16] werden Parallelisierungsstrategien vorgeschlagen, bei denen die Zuordnung der Teilchen und Gitterzellen gebietsorientiert erfolgt, wobei jedoch die Partitionierungen für Teilchen und Gitterzellen nicht übereinstimmen müssen. Während in [15] keine Performanceergebnisse präsentiert werden und in [13] lediglich Ergebnisse für den trivialen Fall einer im gesamten Rechengebiet homogenen Teilchenverteilung präsentiert werden, für den eine Lastbalancierung nicht notwendig ist, werden in [16] einige Performance Ergebnisse für ein Benchmarkproblem mit einer räumlich konzentrierten Teilchenverteilung gezeigt, bei der etwa 50% der Gitterzellen mit Teilchen gefüllt sind, die sich in eine Vorzugsrichtung bewegen. In [14] wird ein reales Simulationsproblem als Benchmark benutzt. In [9, 10, 11, 12, 67] werden Parallelisierungsstrategien vorgeschlagen, welche die Zuordnung der Teilchen und Gitterzellen ebenfalls gebietsorientiert vornehmen, bei denen jedoch die Partitionierungen für Teilchen und Felder übereinstimmen. Eine dynamische Lastbalancierung wird durch die inkrementelle Veränderung der Partitions Grenzen durchgeführt. In Abschnitt 3.4.6.1 wurde eine Verallgemeinerung dieser Strategie vorgestellt.



---

## D Optimalitätsbeweis

Um zu beweisen, dass der in Abschnitt 3.4.3 beschriebene Algorithmus für den kontinuierlichen Fall einen optimalen Zustand findet, soll zunächst gezeigt werden, dass die durch die Nebenbedingungen (3.48) und (3.47) beschriebene Menge konvex ist. Anschließend wird die Konvexität der Zielfunktion (3.46) gezeigt. Aufgrund dieser Eigenschaften der Zielfunktion und der Menge der zulässigen Lösungen kann gefolgert werden, dass jedes lokale Optimum identisch zu dem globalen Optimum ist (siehe Satz 2.35 in [102]). Anschließend wird gezeigt, dass der beschriebene Algorithmus ein lokales Optimum findet, womit bewiesen ist, dass der Algorithmus zu einem globalen Optimum führt. Es sei angemerkt, dass der Beweis bisher nur unter der Voraussetzung eines Parallelrechners mit gleichartigen Prozessoren vollständig geführt werden konnte. In Abschnitt D.3 wird an entsprechender Stelle auf diese Einschränkung hingewiesen.

### D.1 Konvexität der Menge der zulässigen Lösungen

Gemäß Definition 3.1 aus [102] handelt es sich bei der Menge, welche die Nebenbedingungen beschreibt, um ein Polyeder. Gemäß Bemerkung 3.2 aus [102] ist jedes Polyeder konvex. Damit ist die Menge der zulässigen Lösungen konvex.

### D.2 Konvexität der Zielfunktion

Gemäß Definition 2.26 aus [102] ist eine Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  genau dann konvex, wenn für alle  $x_1, x_2 \in \mathbb{R}^n$  und  $\lambda \in (0; 1)$  gilt, dass

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda \cdot f(x_1) + (1 - \lambda) \cdot f(x_2). \quad (\text{D.1})$$

Die Zielfunktion lautet

$$T_S(N_\pi, \mathcal{P}) = \max_{i_\pi \in \Pi} \left\{ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n_\pi} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{i_\pi, j_\pi}^{n_\pi} + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{j_\pi, i_\pi}^{n_\pi} \right\}. \quad (\text{D.2})$$

Es gilt

$$\begin{aligned}
 & \max_{i_\pi \in \Pi} \left\{ \beta_{i_\pi} \cdot \left[ \lambda \cdot a_{i_\pi, i_\pi}^{n\pi} + (1 - \lambda) \cdot a_{i_\pi, i_\pi}^{n\pi'} \right] + \dots \right. \\
 & \quad \dots + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} \left[ (\lambda a_{i_\pi, j_\pi}^{n\pi} + (1 - \lambda) a_{i_\pi, j_\pi}^{n\pi'}) \right] + \dots \\
 & \quad \left. \dots + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} \left[ \lambda a_{j_\pi, i_\pi}^{n\pi} + (1 - \lambda) a_{j_\pi, i_\pi}^{n\pi'} \right] \right\} \\
 = & \max_{i_\pi \in \Pi} \left\{ \lambda \left[ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n\pi} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{i_\pi, j_\pi}^{n\pi} + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{j_\pi, i_\pi}^{n\pi} \right] + \dots \right. \\
 & \quad \left. \dots + (1 - \lambda) \left[ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n\pi'} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{i_\pi, j_\pi}^{n\pi'} + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{j_\pi, i_\pi}^{n\pi'} \right] \right\} \\
 \leq & \lambda \cdot \max_{i_\pi \in \Pi} \left\{ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n\pi} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{i_\pi, j_\pi}^{n\pi} + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{j_\pi, i_\pi}^{n\pi} \right\} + \dots \\
 & \dots + (1 - \lambda) \cdot \max_{i_\pi \in \Pi} \left\{ \beta_{i_\pi} \cdot a_{i_\pi, i_\pi}^{n\pi'} + (\beta_{i_\pi} + \gamma) \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{i_\pi, j_\pi}^{n\pi'} + \gamma \cdot \sum_{\substack{j_\pi \in \Pi \\ j_\pi \neq i_\pi}} a_{j_\pi, i_\pi}^{n\pi'} \right\}, \quad (D.3)
 \end{aligned}$$

was aufgrund der Dreiecksungleichung folgt [103]. Die Konvexität der Zielfunktion ist somit gezeigt.

### D.3 Beweis der Optimalität

Nun soll gezeigt werden, dass der in Abschnitt 3.4.3 vorgeschlagene Algorithmus zu Werten  $a_{i_\pi, j_\pi}^{n\pi}$  führt, für welche die Zielfunktion ein lokales Minimum und, gemäß den Überlegungen in Abschnitt D.1 und D.2, somit ihr globales Minimum annimmt.

Das Verfahren bricht entweder ab, weil ein Prozessor der Gruppe  $u^{(n)} \dots N_\pi$  keine Teilchen mehr besitzt, die abgegeben werden können (siehe Gleichung (3.59)), oder weil  $\beta_{i_\pi} < \gamma$  für einen Prozessor dieser Gruppe gilt (siehe Gleichung (3.60)), oder weil  $T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi)$  gilt (siehe Gleichung (3.58)). Zunächst werden die beiden ersten Fälle betrachtet.

**Fall 1:** Das Verfahren bricht ab, weil für einen Prozessor  $i_\pi$   $a_{i_\pi, i_\pi}^{n\pi(n)} = 0$  gilt.

Aufgrund des  $\max\{\cdot\}$  Ausdrucks in der Zielfunktion müssen, um den Wert der Zielfunktion zu reduzieren, die  $T_S^{(n)}(u^{(n)} \dots N_\pi)$  alle reduziert werden, da diese nach jedem Schritt des Optimierungsverfahrens gleich sind und den Wert der Zielfunktion bestimmen.

Die  $T_S^{(n)}(u^{(n)} \dots N_\pi)$  können jedoch nur dann reduziert werden, wenn die  $a_{i_\pi, i_\pi}^{n\pi^{(n)}}$  der Prozessoren  $u^{(n)} \dots N_\pi$  reduziert werden. Da  $a_{i_\pi, i_\pi}^{n\pi^{(n)}} = 0$  für einen dieser Prozessoren gilt, kann der Wert  $T_S^{(n)}(i_\pi)$  nicht mehr weiter reduziert werden. Damit kann auch der Wert der Zielfunktion nicht weiter reduziert werden und der erreichte Zustand ist optimal.

**Fall 2:** Das Verfahren bricht ab, weil für einen Prozessor  $i_\pi$   $\beta_{i_\pi} < \gamma$  gilt. Wie unter Fall 1 bereits erläutert, müssen die  $T_S^{(n)}(u^{(n)} \dots N_\pi)$  alle reduziert werden, um den Wert der Zielfunktion zu reduzieren. Für einen Prozessor der Gruppe  $u^{(n)} \dots N_\pi$  kann dieser Wert nur durch Abgabe von Teilchen reduziert werden. Gibt ein solcher Prozessor  $i_\pi$  eine Anzahl von  $k$  Teilchen ab, so verändert sich der Wert von  $T_S^{(n)}(i_\pi)$  um  $(\gamma - \beta_{i_\pi}) \cdot k$ . Falls nicht  $\beta_{i_\pi} < \gamma$  gilt, kann  $T_S^{(n)}(i_\pi)$  folglich nicht reduziert werden und ist somit der bestmögliche Wert der Zielfunktion.

**Fall 3:** Das Verfahren bricht ab, weil  $T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi) \forall i_\pi, j_\pi \in \Pi$  gilt.

Für diesen Fall soll die Optimalität nur für den Sonderfall eines Parallelrechners mit gleichartigen Prozessoren

$$\beta_{i_\pi} = \beta \quad \forall i_\pi \in \Pi \quad (\text{D.4})$$

bewiesen werden. Auf den allgemeinen Fall konnte der Beweis bislang nicht erweitert werden. Um zu zeigen, dass ein lokales Optimum vorliegt, soll eine differentiell kleine Änderung in den  $a_{i_\pi, j_\pi}^{n\pi^{(n)}}$  und deren Auswirkung auf den Wert der Zielfunktion untersucht werden. Um den Wert der Zielfunktion durch eine differentielle Änderung der  $a_{i_\pi, j_\pi}^{n\pi^{(n)}}$  zu reduzieren, ist wegen des  $\max\{\cdot\}$  Ausdrucks in der Zielfunktion sowie der Tatsache, dass  $T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi) \forall i_\pi, j_\pi \in \Pi$  gilt, notwendig, dass

$$dT_S^{(n)}(i_\pi) < 0 \quad \forall i_\pi \in \Pi \quad (\text{D.5})$$

gilt. Es werden zunächst die Prozessoren der Gruppe  $u^{(n)} \dots N_\pi$  betrachtet. Die Laständerung für einen Prozessor aus dieser Gruppe ergibt sich zunächst ganz allgemein zu

$$dT_S^{(n)}(i_\pi) = (\gamma + \beta) \cdot \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} da_{i_\pi, j_\pi}^{n\pi^{(n)}} + (\gamma - \beta) \cdot \sum_{\substack{j_\pi = 1 \\ j_\pi \neq i_\pi}}^{N_\pi} da_{j_\pi, i_\pi}^{n\pi^{(n)}}. \quad (\text{D.6})$$

Aufspalten der Summen nach den beiden Prozessorgruppen  $1 \dots l^{(n)}$  und  $u^{(n)} \dots N_\pi$  ergibt

$$\begin{aligned} dT_S^{(n)}(i_\pi) = & (\gamma + \beta) \cdot \sum_{j_\pi=1}^{l^{(n)}} da_{i_\pi, j_\pi}^{n\pi^{(n)}} + (\gamma - \beta) \cdot \sum_{j_\pi=1}^{l^{(n)}} da_{j_\pi, i_\pi}^{n\pi^{(n)}} + \dots \\ & \dots + (\gamma + \beta) \cdot \sum_{\substack{j_\pi = u^{(n)} \\ j_\pi \neq i_\pi}}^{N_\pi} da_{i_\pi, j_\pi}^{n\pi^{(n)}} + (\gamma - \beta) \cdot \sum_{\substack{j_\pi = u^{(n)} \\ j_\pi \neq i_\pi}}^{N_\pi} da_{j_\pi, i_\pi}^{n\pi^{(n)}}. \end{aligned} \quad (\text{D.7})$$

Damit (D.5) erfüllt sein kann, ist es notwendig, dass

$$\sum_{i_\pi=u^{(n)}}^{N_\pi} dT_S^{(n)}(i_\pi) < 0. \quad (\text{D.8})$$

Einsetzen von (D.7) in (D.8) ergibt

$$2\gamma \underbrace{\sum_{i_\pi=u^{(n)}}^{N_\pi} \sum_{\substack{j_\pi=u^{(n)} \\ j_\pi \neq i_\pi}}^{N_\pi} d a_{i_\pi, j_\pi}^{n\pi^{(n)}}}_{\textcircled{1}} + (\gamma + \beta) \cdot \underbrace{\sum_{i_\pi=u^{(n)}}^{N_\pi} \sum_{j_\pi=1}^{l^{(n)}} d a_{i_\pi, j_\pi}^{n\pi^{(n)}}}_{\textcircled{2}} + (\gamma - \beta) \cdot \underbrace{\sum_{i_\pi=u^{(n)}}^{N_\pi} \sum_{j_\pi=1}^{l^{(n)}} d a_{j_\pi, i_\pi}^{n\pi^{(n)}}}_{\textcircled{3}} < 0. \quad (\text{D.9})$$

Wegen der Nebenbedingung  $\bar{a}_{i_\pi, j_\pi}^{n\pi^{(n)}} \geq 0 \quad \forall i_\pi, j_\pi \in \Pi$  folgt, dass alle Terme in den Summen  $\textcircled{1}$  und  $\textcircled{2}$  größer oder gleich Null sein müssen, da für das Optimierungsverfahren immer gilt

$$\begin{aligned} a_{i_\pi, j_\pi}^{n\pi^{(n)}} &= 0 \quad \forall i_\pi, j_\pi \in \{u^{(n)} \dots N_\pi\} \quad (i_\pi \neq j_\pi), \\ a_{i_\pi, j_\pi}^{n\pi^{(n)}} &= 0 \quad \forall j_\pi \in \{1 \dots l^{(n)}\}, i_\pi \in \{u^{(n)} \dots N_\pi\}. \end{aligned}$$

Nun werden die Prozessoren der Gruppe  $1 \dots l^{(n)}$  betrachtet. Wieder folgt aufgrund von (D.5)

$$\sum_{i_\pi=1}^{l^{(n)}} dT_S^{(n)}(i_\pi) < 0 \quad (\text{D.10})$$

als notwendige Bedingung. Es folgt durch Einsetzen des allgemeinen Ausdrucks unter (D.7) in (D.10)

$$2\gamma \underbrace{\sum_{i_\pi=1}^{l^{(n)}} \sum_{\substack{j_\pi=1 \\ j_\pi \neq i_\pi}}^{l^{(n)}} d a_{i_\pi, j_\pi}^{n\pi^{(n)}}}_{\textcircled{4}} + (\gamma + \beta) \cdot \underbrace{\sum_{i_\pi=1}^{l^{(n)}} \sum_{j_\pi=u^{(n)}}^{N_\pi} d a_{i_\pi, j_\pi}^{n\pi^{(n)}}}_{\textcircled{5}} + (\gamma - \beta) \cdot \underbrace{\sum_{i_\pi=1}^{l^{(n)}} \sum_{j_\pi=u^{(n)}}^{N_\pi} d a_{j_\pi, i_\pi}^{n\pi^{(n)}}}_{\textcircled{6}} < 0. \quad (\text{D.11})$$

Durch Vertauschung der Indizes lassen sich die Summen  $\textcircled{5}$  und  $\textcircled{6}$  in die Summen  $\textcircled{3}$  bzw.  $\textcircled{2}$  überführen. Wegen

$$\begin{aligned} a_{i_\pi, j_\pi}^{n\pi^{(n)}} &= 0 \quad \forall i_\pi, j_\pi \in \{1 \dots l^{(n)}\} \quad (i_\pi \neq j_\pi) \\ a_{i_\pi, j_\pi}^{n\pi^{(n)}} &= 0 \quad \forall j_\pi \in \{1 \dots l^{(n)}\}, i_\pi \in \{u^{(n)} \dots N_\pi\}, \end{aligned}$$

folgt, dass alle Terme in den Summen ④ und ⑥ größer Null sein müssen. Mit den Abkürzungen für die Summen lassen sich die beiden Ungleichungen (D.9) und (D.11) schreiben als

$$2\gamma \cdot \textcircled{1} + (\gamma + \beta) \cdot \textcircled{2} + (\gamma - \beta) \cdot \textcircled{3} < 0 \quad (\text{D.12})$$

$$2\gamma \cdot \textcircled{4} + (\gamma - \beta) \cdot \textcircled{2} + (\gamma + \beta) \cdot \textcircled{3} < 0. \quad (\text{D.13})$$

Wegen  $\gamma > 0$  und  $\textcircled{1} > 0$ ,  $\textcircled{4} > 0$  folgt, dass notwendig ist um (D.12) und (D.13) zu erfüllen, dass

$$(\gamma + \beta) \cdot \textcircled{2} + (\gamma - \beta) \cdot \textcircled{3} < 0 \quad (\text{D.14})$$

$$(\gamma - \beta) \cdot \textcircled{2} + (\gamma + \beta) \cdot \textcircled{3} < 0 \quad (\text{D.15})$$

gilt. Diese zwei Ungleichungen sind äquivalent zu der Ungleichungskette

$$\frac{\beta + \gamma}{\beta - \gamma} \cdot \textcircled{3} < \textcircled{2} < \frac{\beta - \gamma}{\beta + \gamma} \cdot \textcircled{3} \quad (\text{D.16})$$

Wegen  $\beta - \gamma > 0$  gilt aber

$$\frac{\beta + \gamma}{\beta - \gamma} > \frac{\beta - \gamma}{\beta + \gamma}. \quad (\text{D.17})$$

Somit kann die Ungleichungskette (D.16) nur dann erfüllt werden, falls  $\textcircled{3} < 0$ , woraus  $\textcircled{2} < 0$  folgen würde im Widerspruch zur Nebenbedingung.

Damit kann durch eine differentielle Änderung der  $a_{i_\pi, j_\pi}^{n\pi(n)}$  der Wert der Zielfunktion in der Umgebung des erreichten Wertes nicht weiter reduziert werden, wenn das Verfahren in einem Zustand abbricht für den  $T_S^{(n)}(i_\pi) = T_S^{(n)}(j_\pi) \forall i_\pi, j_\pi \in \Pi$  gilt. Der erreichte Zustand muss ein lokales und damit aufgrund der Eigenschaften von Zielfunktion und Nebenbedingungen auch ein globales Optimum sein.



# Literaturverzeichnis

- [1] DAVID W. WALKER: *Particle-In-Cell Plasma Simulation Codes on the Connection Machine*. Computer Systems in Engineering, 2(2/3):307–319, 1991.
- [2] IAN T. FOSTER and DAVID W. WALKER: *Paradigms and Strategies for Scientific Computing on Distributed Memory Concurrent Computers*. In *Proceedings of the 1994 Simulation Multiconference on Grand Challenges in Computer Simulation*, pages 252–257, La Jolla, CA, USA, 1994. Simulation Councils, Inc., ISBN 1-56555-073-0.
- [3] B. DI MARTINO, S. BRIGUGLIO, G. VLAD, and P. SGUAZZERO: *Parallel PIC Plasma Simulation Through Particle Decomposition Techniques*. Parallel Computing, 27(3):295–314, 2001.
- [4] WARNER BRUNS: *Parallel Particle In Cell Simulations with GdfidL*. In *Proceedings of the EPAC 2004*, pages 2538–2540, 2004.
- [5] PETER J. MARDAHL and JOHN P. VERBONCOEUR: *Progress in Parallelizing XOOPIC*. In *Proceedings of the 1998 International Computational Accelerator Physics Conference*, pages 218–221, 1998.
- [6] R. G. HEMKER, F. S. TSUNG, V. K. DECYK, W. B. MORI, S. LEE, and T. KATSULEAS: *Development of a Parallel Code for Modeling Plasma Based Accelerators*. In *Proceedings of the 1999 Particle Accelerator Conference*, pages 3672–3674. IEEE, 1999.
- [7] RICARDO A. FONSECA *et al.*: *OSIRIS: A Three-Dimensional, Fully Relativistic Particle in Cell Code for Modeling Plasma Based Accelerators*. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part III*, pages 342–351, London, UK, 2002. Springer-Verlag, ISBN 3-540-43594-8.
- [8] CHET NIETER and JOHN R. CARY: *VORPAL: A Versatile Plasma Simulation Code*. Journal of Computational Physics, 196(22):448–473, 2004.
- [9] PHILIP M. CAMPBELL, EDWARD A. CARMONA, and DAVID W. WALKER: *Hierarchical Domain Decomposition with Unitary Load Balancing for Electromagnetic*

- Particle-In-Cell Codes*. In *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 943–950. IEEE Computer Society Press, 1990.
- [10] JOSEPH. D. BLAHOVEC, LESTER A. BOWERS, JOHN W. LUGINSLAND, GERALD E. SASSER, and JOHN J. WATROUS: *3-D ICEPIC Simulations of the Relativistic Klystron Oscillator*. IEEE Transactions on Plasma Science, 28(3):821–829, 2000.
- [11] PETER MARDAHL, ANDREW GREENWOOD, TONY MURPHY, and KEITH CARTWRIGHT: *Parallel Performance Characteristics of ICEPIC*. In *DOD\_UGC '03: Proceedings of the 2003 DoD User Group Conference*, pages 86–90, Washington, DC, USA, 2003. IEEE Computer Society, ISBN 0-7695-1953-9.
- [12] M. T. BETTENCOURT *et al.*: *Virtual Prototyping of Directed Energy Weapons on Thousands of Processors*. In *Proceedings of the HPCMP Users Group Conference 2006*, pages 259–266. IEEE Computer Society, 2006.
- [13] PAULETT C. LIEWER and VIKTOR K. DECYK: *A General Concurrent Algorithm for Plasma Particle-In-Cell Simulation Codes*. Journal of Computational Physics, 85(2):302–322, 1989, ISSN 0021-9991.
- [14] ROBERT D. FERRARO, PAULETT C. LIEWER, and VIKTOR K. DECYK: *Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code*. Journal of Computational Physics, 109(2):329–341, 1993.
- [15] EDWARD A. CARMONA and LEON J. CHANDLER: *On Parallel PIC Versatility and the Structure of Parallel PIC Approaches*. Concurrency: Practice and Experience, 9(12):1377–1405, 1997.
- [16] STEVEN J. PLIMPTON, DAVID B. SEIDEL, MICHEAL F. PASIK, REBECCA S. COATS, and GARY A. MONTRY: *A Load-Balancing Algorithm for a Parallel Electromagnetic Particle-In-Cell Code*. Computer Physics Communications, 152(3):227–241, 2003.
- [17] JAMES CLERK MAXWELL: *A Treatise on Electricity and Magnetism, Vol I*. Oxford University Press, London, 1873.
- [18] JAMES CLERK MAXWELL: *A Treatise on Electricity and Magnetism, Vol II*. Oxford University Press, London, 1873.
- [19] JOHN DAVID JACKSON: *Klassische Elektrodynamik*. Walter de Gruyter, 2002.
- [20] ARNOLD SOMMERFELD: *Vorlesungen über Theoretische Physik, Band 3*. Akademische Verlagsgesellschaft Geest & Portig K.G., 1949.
- [21] WILHELM H. KEGEL: *Plasmaphysik*. Springer-Verlag Berlin Heidelberg New York, 1998.

- 
- [22] ALBERT COHEN and BENOIT PERTHAME: *Optimal Approximations of Transport Equations by Particle and Pseudoparticle Methods*. SIAM Journal Math. Anal., 32(3):616–636, 2000.
- [23] THOMAS WEILAND: *Eine Methode zur Lösung der Maxwell'schen Gleichungen für sechskomponentige Felder auf diskreter Basis*. Electronics and Communication (AEÜ), 31(3):116–120, 1977.
- [24] THOMAS WEILAND: *Time Domain Electromagnetic Field Computation with Finite Difference Methods*. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 9(4):295–319, 1996.
- [25] ULRICH BECKER and THOMAS WEILAND: *Particle-In-Cell Simulations within the Finite-Integration Method*. Surveys on Mathematics for Industry, 8(4):233–242, 1998.
- [26] THOMAS WEILAND: *Elektromagnetisches CAD*. Technische Universität Darmstadt, 2006.
- [27] CHARLES K. BIRDSALL and A. B. LANGDON: *Plasma Physics via Computer Simulation*. IOP Publishing, 1991, ISBN 0-7503-0117-1.
- [28] J. P. BORIS: *Relativistic Plasma Simulation-Optimization of a Hybrid Code*. In *Proceedings of the Fourth Conference Numerical Simulation of Plasmas*, pages 3–67, 1970.
- [29] J. P. VERBONCOEUR: *Particle Simulation of Plasmas: Review and Advances*. Plasma Physics and Controlled Fusion, 47(5):A231–A260, 2005.
- [30] CHARLES K. BIRDSALL and DIETER FUSS: *Clouds-in-Clouds, Clouds-in-Cells Physics for Many-Body Plasma Simulation*. Journal of Computational Physics, 135(2):141–148, 1997.
- [31] JOHN VILLASENOR and OSCAR BUNEMAN: *Rigorous Charge Conservation for Local Electromagnetic Field Solvers*. Computer Physics Communications, 69:306–316, 1992.
- [32] ANDREW S. TANENBAUM: *Computerarchitektur*. Pearson Studium, 5. Auflage, 2006, ISBN 3-8273-7151-1.
- [33] MICHAEL J. FLYNN: *Some Computer Organizations and Their Effectiveness*. IEEE Transactions on Computers, C-21(9):948 – 960, 1972.
- [34] LUCIANO TARRICONE and ALESSANDRA ESPOSITO: *Grid Computing for Electromagnetics*. Artech House, Inc., Norwood, Massachusetts, USA, 2004.

- [35] A. GARA *et al.*: *Overview of the BlueGene/L System Architecture*. IBM Journal of Research and Development, 49(2):195–212, 2005.
- [36] TOP500 SUPERCOMPUTING SITES: <http://www.top500.org/>.
- [37] INHO PARK and SEON WOOK KIM: *The Distributed Virtual Shared-Memory System based on the InfiniBand Architecture*. J. Parallel Distrib. Comput., 65(10):1271–1280, 2005, ISSN 0743-7315.
- [38] THOMAS RAUBER und GUNDULA RÜNGER: *Parallele und verteilte Programmierung*. Springer-Verlag Berlin Heidelberg New York, 2000, ISBN 3-540-66009-7.
- [39] F. DAREMA, D. A. GEORGE, V. A. NORTON, and G. PFISTER: *A Single-Program-Multiple-Data Computational Model for EPEX/FORTRAN*. Parallel Computing, 7:11–24, 1988.
- [40] OPENMP ARCHITECTURE REVIEW BOARD: *OpenMP Application Program Interface*. 2005. <http://www.openmp.org/>.
- [41] MESSAGE PASSING INTERFACE FORUM: *MPI: A Message-Passing Interface Standard*. Technical report, Knoxville, 1994. <http://www-unix.mcs.anl.gov/mpi>.
- [42] MESSAGE PASSING INTERFACE FORUM: *MPI-2: Extensions to the Message-Passing Interface*. Technical report, Knoxville, 1997. <http://www-unix.mcs.anl.gov/mpi>.
- [43] MARTIN SCHULZ: *Shared Memory Programming on NUMA-based Clusters Using a General and Open Hybrid Hardware/Software Approach*. PhD Thesis, Fakultät für Informatik, Technische Universität München, 2001.
- [44] GERHARD GOOS: *Vorlesungen über Informatik, Band 3*. Springer-Verlag Berlin Heidelberg New York, 2. Auflage, 1997, ISBN 3-540-60655-6.
- [45] SARTAJ SAHNI and VENKAT THANVANTRI: *Performance Metrics: Keeping the Focus on Runtime*. Parallel & Distributed Technology: Systems & Applications, 4(1):43–55, 1996.
- [46] VIPIN KUMAR and ANSHUL GUPTA: *Analyzing Scalability of Parallel Algorithms and Architectures*. Journal of Parallel and Distributed Computing, 22(3):379–391, 1994, ISSN 0743-7315.
- [47] VIPIN KUMAR, ANANTH GRAMA, ANSHUL GUPTA, and GEORGE KARYPIS: *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 2. edition, 2003, ISBN 0-201-64865-2.
- [48] MICHAEL J. QUINN: *Parallel Programming in C with MPI and OpenMP*. McGraw-

- Hill, 2003, ISBN 007-123265-6.
- [49] D. P. HELMBOLD and C. E. MCDOWELL: *Modeling Speedup ( $n$ ) Greater Than  $n$* . IEEE Transactions on Parallel and Distributed Systems, 1(2):250–256, 1990, ISSN 1045-9219.
- [50] GENE M. AMDAHL: *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. AFIPS Conference Proceedings National Computer Conference, pages 483–485, 1967.
- [51] XIAN HE SUN and LIONEL M. NI: *Another View on Parallel Speedup*. In *Supercomputing '90: Proceedings of the 1990 conference on Supercomputing*, pages 324–333, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press, ISBN 0-89791-412-0.
- [52] JOHN L. GUSTAFSON: *Reevaluating Amdahls Law*. Communications of the ACM, 31(5):532–533, 1988.
- [53] ANNANTH Y. GRAMA, ANSHUL GUPTA, and VIPIN KUMAR: *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures*. IEEE Parallel & Distributed Technology, 1(3):12–21, 1993.
- [54] GERHARD GOOS: *Vorlesungen über Informatik, Band 2*. Springer-Verlag Berlin Heidelberg New York, 3. Auflage, 2001, ISBN 3-540-41511-4.
- [55] THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, and CLIFFORD STEIN: *Introduction to Algorithms*. McGraw-Hill, 2. edition, 2001, ISBN 0-262-03293-7.
- [56] DUNCAN A. GROVE: *Performance Modelling of Message Passing Parallel Programs*. Dissertation, University of Adelaide, 2003.
- [57] STEVEN FORTUNE and JAMES WYLLIE: *Parallelism in Random Access Machines*. In *STOC '78: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118, New York, NY, USA, 1978. ACM Press.
- [58] TIM J. HARRIS: *A Survey of PRAM Simulation Techniques*. ACM Comput. Surv., 26(2):187–206, 1994, ISSN 0360-0300.
- [59] LESLIE G. VALIANT: *A Bridging Model for Parallel Computation*. Commun. ACM, 33(8):103–111, 1990, ISSN 0001-0782.
- [60] DAVID CULLER *et al.*: *LogP: Towards a Realistic Model of Parallel Computation*. In *PPOPP '93: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, New York, NY, USA, 1993. ACM Press, ISBN 0-89791-589-5.

- [61] DUNCAN A. GROVE and PAUL D. CODDINGTON: *Communication Benchmarking and Performance Modelling of MPI Parallel Programs on Cluster Computers*. Int. J. of Supercomputing, 34(2):201–217, 2005.
- [62] COSIMO ANGLANO: *Predicting Parallel Applications Performance on non-dedicated Cluster Platforms*. In *ICS '98: Proceedings of the 12th international conference on Supercomputing*, pages 172–179, New York, NY, USA, 1998. ACM Press, ISBN 0-89791-998-X.
- [63] HORST CLAUSERT und GUNTHER WIESEMANN: *Grundgebiete der Elektrotechnik, Band 1*. Oldenbourg, 9. Auflage, 2004.
- [64] CHRISTOS H. PAPADIMITRIOU and KENNETH STEIGLITZ: *Combinatorial Optimization*. Dover Publications, Inc., 1998.
- [65] MICHAEL R. GAREY and DAVID S. JOHNSON: *Computers and Intractability*. W. H. Freeman and Company, 1979, ISBN 0-7167-1045-5.
- [66] MARSHA J. BERGER and SHAHID H. BOKHARI: *A Partitioning Strategy for Nonuniform Problems on Multiprocessors*. IEEE Transactions on Computers, C-36(5):570–580, 1987.
- [67] FELIX WOLFHEIMER: *Simulation geladener Teilchen auf parallel geclusterten Rechnern*. Diplomarbeit, Technische Universität Darmstadt, 2003.
- [68] FELIX WOLFHEIMER, ERION GJONAJ, and THOMAS WEILAND: *A Parallel 3D Particle-In-Cell Code with Dynamic Load Balancing*. Nuclear Instruments and Methods in Physics Research A, 558(1):202–204, 2006.
- [69] J. P. SINGH, C. HOLT, J. L. HENNESSY, and A. GUPTA: *A Parallel Adaptive Fast Multipole Method*. In *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 54–65, New York, NY, USA, 1993. ACM Press, ISBN 0-8186-4340-4.
- [70] HORST D. SIMON and SHANG HUA TENG: *How Good is Recursive Bisection?* SIAM Journal on Scientific Computing, 18(5):1436–1445, 1997, ISSN 1064-8275.
- [71] JACK DONGARRA: *The Sourcebook of Parallel Computing*. Morgan Kaufmann, 2002.
- [72] WARNER BRUNS: *GdfidL on Massive Parallel Systems*. In *Proceedings of the LINAC 2002*, pages 418–420, 2002.
- [73] WARNER BRUNS: *Parallel Particle In Cell Computation of an Electron Gun with GdfidL*. In *Proceedings of the LINAC 2004*, pages 498–500, 2004.

- 
- [74] FELIX WOLFHEIMER, ERION GJONAJ, and THOMAS WEILAND: *Parallel Particle-In-Cell (PIC) Codes*. In *Proceedings of the ICAP 2006*, 2006.
- [75] FELIX WOLFHEIMER, ERION GJONAJ, and THOMAS WEILAND: *Dynamic Domain Decomposition for Parallel PIC Simulations in the Time Domain*. In *Proceedings of 9th International Conference on Electromagnetics in Advanced Applications*, pages 1003–1006, 2005.
- [76] K. J. BOWERS: *Accelerating a Particle-in-Cell Simulation Using a Hybrid Counting Sort*. *Journal of Computational Physics*, 173(2):393–411, 2001, ISSN 0021-9991.
- [77] D.C. STERNEL, D. JUNGLAS, A. MARTIN, and M. SCHAEFER: *Optimisation of Partitioning for Parallel Flow Simulation on Block Structured Grids*. In *Proceedings of the Fourth International Conference on Engineering Computational Technology*, Stirling, United Kingdom, 2004. Civil-Comp Press. paper 93.
- [78] A. OPPELT *et al.*: *Status and First Results from the Upgraded PITZ Facility*. In *Proceedings of the FEL 2005*, pages 564–567, 2005.
- [79] MIN ZHANG and PETRA SCHUETT: *TESLA FEL Gun Simulations with PARMELA and MAFIA*. In *Proceedings of the Computational Accelerator Physics Conference (CAP 1996)*, pages 21–26, 1996.
- [80] MARTIN REISER: *Theory and Design of Charged Particle Beams*. John Wiley & Sons, 1994.
- [81] M. KRASILNIKOV *et al.*: *Optimizing the PITZ Electron Source for the VUV-FEL*. In *Proceedings of the EPAC 2004*, pages 360–362, 2004.
- [82] K. FLÖTTMANN: *ASTRA User Manual*. <http://www.desy.de/~mpyflo/>.
- [83] S. SETZER, R. CEE, M. KRASILNIKOV, and T. WEILAND: *FEL Photoinjector Simulation Studies by Combining MAFIA TS2 and ASTRA*. In *Proceedings of the EPAC 2002*, pages 1664–1666, 2002.
- [84] E. GJONAJ, T. LAU, S. SCHNEPP, F. WOLFHEIMER, and T. WEILAND: *Accurate Modelling of Charged Particle Beams in Linear Accelerators*. *New Journal of Physics*, 8(285):1–21, 2006.
- [85] J. BUON: *Beam Phase Space and Emittance*. In S. TURNER (editor): *CERN Accelerator School - Fifth General Accelerator Physics Course*, volume 1, pages 89–115, 1994.
- [86] ROLF RABENSEIFNER and GERHARD WELLEIN: *Communication and Optimization Aspects of Parallel Programming*. *International Journal of High Performance Computing Applications*, 17(2):49–62, 2003.

- [87] SHAHID H. BOKHARI: *On the Mapping Problem*. IEEE Transactions on Computers, C-30(3):207–214, 1981.
- [88] BRUCE HENDRICKSON and TAMARA G. KOLDA: *Graph Partitioning Models for Parallel Computing*. Parallel Computing, 26(12):1519–1534, 2000.
- [89] BRUCE HENDRICKSON and ROBERT LELAND: *A Multilevel Algorithm for Partitioning Graphs*. Technical report, Sandia National Laboratories, Albuquerque, NM, USA, 1993.
- [90] GEORGE KARYPIS and VIPIN KUMAR: *Analysis of Multilevel Graph Partitioning*. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 29, New York, NY, USA, 1995. ACM, ISBN 0-89791-816-9.
- [91] GEORGE KARYPIS and VIPIN KUMAR: *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*. SIAM Journal on Scientific Computing, 20(1):359–392, 1998.
- [92] GEORGE KARYPIS and VIPIN KUMAR: *A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering*. Journal of Parallel and Distributed Computing, 48(1):71–95, 1998, ISSN 0743-7315.
- [93] C. WALSHAW and M. CROSS: *Parallel Optimization Algorithms for Multilevel Mesh Partitioning*. Parallel Computing, 26(12):1635–1660, 2000.
- [94] BRUCE HENDRICKSON and ROBERT LELAND: *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*. SIAM Journal on Scientific Computing, 16(2):452–469, 1995, ISSN 1064-8275.
- [95] C. GUIFFAUT and K. MAHDJOUBI: *A Parallel FDTD Algorithm Using the MPI Library*. IEEE Antennas and Propagation Magazine, 43(2):94–103, 2001.
- [96] B. NOUR-OMID, A. RAEFSKY, and G. LYZENGA: *Solving Finite Element Equations on Concurrent Computers*. In *Proc. Symposium on Parallel Computations and their Impact on Mechanics*, pages 291–307, 1987.
- [97] HANS SAGAN: *Space-Filling Curves*. Springer-Verlag Berlin Heidelberg New York, 1994, ISBN 3-540-94265-3.
- [98] JOHN R. PILKINGTON and SCOTT B. BADEN: *Dynamic Partitioning of Non-Uniform Structured Workloads with Spacefilling Curves*. IEEE Trans. Parallel Distrib. Syst., 7(3):288–300, 1996, ISSN 1045-9219.
- [99] BONGKI MOON, H. V. JAGADISH, CHRISTOS FALOUTSOS, and JOEL H. SALTZ: *Analysis of the Clustering Properties of the Hilbert Space-Filling Curve*. IEEE Transactions on Knowledge and Data Engineering, 13(1):124–141, 2001.

- [100] DAVID HILBERT: *Über die stetige Abbildung einer Linie auf ein Flächenstück*. Mathematische Annalen, 38:459–460, 1891.
- [101] SRINIVAS ALURU and FATIH E. SEVILGEN: *Parallel Domain Decomposition and Load Balancing Using Space-Filling Curves*. In *HIPC '97: Proceedings of the Fourth International Conference on High-Performance Computing*, pages 230–235, Washington, DC, USA, 1997. IEEE Computer Society, ISBN 0-8186-8067-9.
- [102] MIRJAM DÜR und ALEXANDER MARTIN: *Skriptum zur Vorlesung Optimierung I*. TU Darmstadt, 2005.
- [103] ILJA N. BRONSTEIN, KONSTANTIN A. SEMENDJAJEW, GERHARD MUSIOL und HEINER MÜHLIG: *Taschenbuch der Mathematik*. Verlag Harri Deutsch Thun und Frankfurt/M., 2000.



# Stichwortverzeichnis

- Benchmarkergebnisse
  - Feldlöser, 104
  - Particle-In-Cell, 106
- Benchmarkproblem, 107
- Bounding Box, 58
- Chip-Multiprozessoren, 26
- Cluster, 27
- Courant-Friedrichs-Levy Bedingung, 22
- dynamische Lastbalancierung, 69
  - Entscheidungsparameter, 92
- Effizienz, 35
  - skalierte, 41
- flacher Speicher, 49
- Geisterzellen, 61
- Gesetz von Amdahl, **39**, 87
- Gitter-Maxwellgleichungen, 16
- Grid, 27
- Isoeffizienzfunktion, 42
- Latency Hiding, **48**, 63
- Liouville-Gleichung, 15
- Maxwellsche Gleichungen, 11
  - Kontinuitätsgleichung, 12
  - Materialbeziehungen, 12
  - Stetigkeitsbedingungen, 13
- Methode der Finiten Integration, 16
  - duales Gitter, 19
  - Gitterfluss, 17
  - Gitterspannung, 17
  - Materialbeziehungen, 20
  - Zeitintegration, 21
- MIMD, 26
- MPI, 32
- Multicomputer, 26
- Multiprozessor, 26
- Multiprozessoren, 26
- Newton-Lorentz-Gleichung, 14
- OpenMP, 31
- paralleles Rechnen, 25
- paralleles System, 33
- Parallelisierung, 25
- Parallelisierungsstrategie, 52
  - Adaptive Bounding Box, 82
  - Dynamische Gebietszerlegung, 87
  - Dynamische Teilchenzuordnung, 71
  - Statische Gebietszerlegung, 84
- Particle-In-Cell Algorithmus, 22
- Phasenraum, 15
- PITZ Injektor, 119
- PRAM-Modell, 45
- Programmiermodell
  - mit gemeinsamem Speicher, 31
  - nachrichtenorientiertes, 31
  - SPMD, 30

Rechnercluster, 27

Rechnermodelle

    BSPC-Modell, 46

    LogP-Modell, 46

    PRAM, 45

Rekursive Koordinaten Bisektionierung, 58

SISD, 26

Skalierbarkeit, 42

Speedup, 34

    skalierter, 41

    superlinearer, 35

TOP 500, 27

Vielteilchensystem, 14

Vlasov-Gleichung, 15

# Danksagung

Ich möchte allen Personen, die zum Entstehen der vorliegenden Arbeit beigetragen haben, recht herzlich danken.

Insbesondere möchte ich mich bedanken bei

Herrn Prof. Dr.-Ing. THOMAS WEILAND, der mir die Möglichkeit gegeben hat, diese Arbeit unter hervorragenden Arbeitsbedingungen am Institut für Theorie Elektromagnetischer Felder (TEMF) anzufertigen und mir mit seinem wissenschaftlichen Rat zur Seite stand.

Herrn Dr. rer. nat. ERION GJONAJ, Herrn Dr. rer. nat. THOMAS LAU sowie Herrn Dr.-Ing. DENIS SIEVERS für die gewissenhafte Durchsicht der Arbeit und wertvolle Diskussionen.

Herrn Prof. Dr. ALEXANDER MARTIN (Fachbereich Mathematik der TU Darmstadt) für seine Unterstützung bei meinen Untersuchungen zu der in Abschnitt 3.4.3 vorgestellten Parallelisierungsstrategie.

Meinen Kollegen von TEMF für wissenschaftliche Anregungen, aufbauende Gespräche und eine gute Arbeitsatmosphäre.

Meinen Eltern und Freunden dafür, dass sie immer ein offenes Ohr für mich hatten und mich stets unterstützt und aufgebaut haben.



# Lebenslauf



- 28. April 1978 – geboren in Frankfurt/M.
- 1997 – Abitur am Augustinergymnasium Friedberg/Hessen (Durchschnittsnote: 1,5)
- 1997 - 2000 – Studium des Wirtschaftsingenieurwesens (Fachrichtung Elektrotechnik) an der TU Darmstadt
- März 2000 – Vordiplom im Studiengang Wirtschaftsingenieurwesen (Fachrichtung Elektrotechnik) an der TU Darmstadt (Gesamtnote: gut)
- 2000 - 2003 – Studium der Elektrotechnik an der TU Darmstadt
- September 2001 – Vordiplom im Fach Elektrotechnik an der TU Darmstadt (Gesamtnote: gut)
- Juli 2003 – Hauptdiplom im Fach Elektrotechnik an der TU Darmstadt (Gesamtnote: sehr gut)
- seit Sept. 2003 – wissenschaftlicher Mitarbeiter am Institut für Theorie Elektromagnetischer Felder der TU Darmstadt