Dynamische Lastbalancierung und Modellkopplung zur hochskalierbaren Simulation von Wolkenprozessen

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Diplom-Informatiker (FH) Matthias Lieber
geboren am 22. Februar 1983 in Dresden

Gutachter:	Prof. Dr. Wolfgang E. Nagel, Technische Universität Dresden
	Prof. Dr. Thomas Ludwig, Universität Hamburg
Tag der Einreichung:	28. März 2012
Tag der Verteidigung:	3. September 2012

Danksagung

An dieser Stelle möchte ich mich bei allen herzlich bedanken, die mich auf jegliche Weise bei der Anfertigung dieser Arbeit unterstützt haben.

Zuallererst danke ich meinem Betreuer Prof. Dr. Wolfgang E. Nagel für sein Vertrauen in mich, seine Motivation und seine Anregungen für die Arbeit. Ohne sein Engagement hätte ich die Möglichkeit, am Zentrum für Informationsdienste und Hochleistungsrechnen zu arbeiten und zu promovieren, vielleicht nie erhalten.

Ralf Wolke danke ich für seinen Einsatz für unser gemeinsames Forschungsprojekt, ohne dem diese Arbeit auch nicht möglich gewesen wäre, sowie unsere konstruktive und mittlerweile langjährige Zusammenarbeit. Ihm und seinem Kollegen Oswald Knoth danke ich auch für die grundlegende Idee zu dieser Arbeit. Verena Grützun danke ich für viele Anregungen und Diskussionen und die gute Zusammenarbeit sowie ihr und Martin Simmel für die Unterstützung beim Einarbeiten in die Wolkenmikrophysik und deren Modellierung.

Allen Kollegen vom ZIH danke ich für die angenehme Atmosphäre in den zurückliegenden sechs Jahren. Insbesondere bedanke ich mich bei Matthias Müller für die vielen Anregungen, Diskussionen und die konstruktive Kritik zu meiner Arbeit. Zudem möchte ich mich besonders bei meinem Zimmerkollegen Matthias Jurenz bedanken, der mir nicht wenige VampirTrace-Wünsche erfüllte, bei Guido Juckeland, der mir auf unserem Planetensystem bei Seite stand, bei Bernd Trenkler für seine Anregungen und bei Andreas Knüpfer für die Organisation des ZIH-Doktorandentreffens.

Darüber hinaus bin ich der Deutschen Forschungsgemeinschaft für drei Jahre Förderung im Rahmen des Projekts „Parallel coupling framework and advanced time integration methods for detailed cloud processes in atmospheric models“ dankbar und dem Jülich Supercomputing Centre für den Zugang zur IBM BlueGene/P.

Und schließlich möchte ich ganz besonders meinen Eltern und meiner ganzen Familie für ihre Unterstützung danken. Danke!

Inhaltsverzeichnis

1 Einleitung

Die Simulation atmosphärischer Prozesse ist eine der bedeutendsten Anwendungen der Computersimulation und hat vor allem durch die numerische Wettervorhersage einen festen Platz im heutigen Alltag gefunden. Aus der seit Jahrzehnten stetig wachsenden Rechenkapazität der Computer und dem zunehmenden Wissen über die relevanten physikalischen und chemischen Prozesse ergeben sich zwei Trends, die zu immer komplexeren Simulationsmodellen führen und zur Verbesserung der Vorhersagequalität beitragen [34, 48, 86, 117]: Zum einen werden immer detailliertere und rechenintensivere Teilmodelle zur Beschreibung atmosphärischer sowie damit verbundener Prozesse entwickelt und eingesetzt. Zum anderen werden immer mehr Elemente der Geosphäre in die Simulationen einbezogen, wie beispielsweise bei Klimamodellen neben der Atmosphäre auch Ozean, Meereis, Landoberfläche und Atmosphärenchemie. Die Zusammenführung der Expertise verschiedener geowissenschaftlicher Disziplinen führt hierbei zu einer Kopplung oft vollkommen unabhängig voneinander entwickelter Modelle, welche je einen der genannten Teilaspekte abbilden. Im Allgemeinen wird dabei von sogenannten Multiphysik- und Multiskalen-Modellen gesprochen [57].

Ein Beispiel für zunehmend komplexer werdende Modelle ist die Beschreibung von Wolkenprozessen in atmosphärischen Modellen. Die Darstellung von Wolkenprozessen ist eines der größten Defizite aktueller Wettervorhersage-, Klima- und Luftqualitätsmodelle [10, 18, 86]. Zukünftige hochauflösende Modelle benötigen detailliertere Beschreibungen von Wolkenprozessen als die heute verbreitet eingesetzten Parametrisierungen, um beispielsweise bessere Vorhersagen von Unwettern zu ermöglichen. Insbesondere die Interaktion von Aerosolpartikeln mit Wolken- und Niederschlagsprozessen wird von den Parametrisierungen nicht genügend abgebildet. Dabei ist bekannt, dass die Größenverteilung und chemische Komposition der Partikel einen bedeutenden Einfluss auf Wolkenprozesse ausübt. Spektrale Wolkenmikrophysikmodelle können diese Interaktionen besser beschreiben, da sie die Größenverteilung der Aerosol- und Wolkenpartikel explizit modellieren [36, 48, 83]. Der numerische Aufwand dieser Modelle ist jedoch derzeit zu hoch für einen verbreiteten Einsatz in atmosphärischen Modellen, insbesondere vor dem Hintergrund strenger Laufzeitvorgaben der operationellen Vorhersage. Heute dienen sie zur Untersuchung der Aerosol-Wolken-Interaktion sowie der Bewertung und Verbesserung der Parametrisierungen [58, 68, 78, 98]. Mit der Weiterentwicklung der Rechenkapazität ist jedoch mittelfristig die Anwendung spektraler Wolkenmodelle zur Luftqualitätsprognose, Extremwettervorhersage und gezielten Wetterbeeinflussung denkbar [43, 48, 68].

Der steigenden numerischen und softwaretechnischen Komplexität der Simulationsmodelle steht ein massives Wachstum der Parallelität von Hochleistungsrechnern gegenüber [55]. Die wachsende Zahl von Prozessorkernen erfordert effektive Methoden der Parallelisierung, die den Aufwand zur Synchronisation und zum Datenaustausch zwischen den parallel arbeitenden Prozessen reduzieren. Neben effektiveren Kommunikationsmethoden bedeutet dies auch, die Rechenlast so gleichmäßig wie möglich über die Prozessorkerne zu verteilen. Dabei wird eine dynamische Lastbalancierung notwendig, wenn durch die Beschreibung von immer mehr physikalischen oder chemischen Prozessen in den Simulationsmodellen [75, 94, 121] oder durch Nutzung adaptiver numerischer Methoden [6, 107] der Berechnungsaufwand von den räumlich und zeitlich variierenden Zustandsvariablen abhängt. Für gekoppelte Modelle mit unterschiedlicher Gebietszerlegung

wird zudem der Datenaustausch zwischen den Modellen bei steigender Parallelität immer aufwendiger. Software zur Unterstützung einer effizienten Parallelisierung und Modellkopplung ist daher – auch über den Bereich der Geowissenschaften hinaus – von hoher Bedeutung.

Ziel dieser Arbeit ist die Entwicklung eines Konzepts zur hochskalierbaren Kopplung von spektralen Wolkenmikrophysikmodellen mit atmosphärischen Modellen. Die Beschleunigung dieser vergleichsweise rechenintensiven Modellsysteme erfordert eine hochparallele Ausführung. Jedoch bewirken die derzeit größten Skalierbarkeitsdefizite, die gravierende Lastimbalance und der hohe Kommunikationsaufwand, Effizienzverluste von über 50 %. Die Zielstellung ist, diese Defizite zu beseitigen und eine effiziente Nutzung der heute verfügbaren Parallelität der Größenordnung von 100 000 Prozessorkernen zu erreichen. Auf diese Weise können Simulationen auf hochaufgelösten Gittern deutlich schneller als Echtzeit realisiert werden. Das Konzept und die Implementierung basieren auf einem vom konkreten Modell unabhängigen Softwareframework. Damit sind diese Entwicklungen sowohl in weiteren Bereichen der Geowissenschaften, wie der Simulation atmosphärenchemischer Prozesse, als auch in anderen wissenschaftlichen Disziplinen anwendbar.

Die Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden Grundlagen zu Parallelrechnern, atmosphärischen Modellen und der Modellierung von Wolkenprozessen eingeführt. Zudem wird ein Überblick über bisherige Anwendungen spektraler Wolkenmikrophysik in atmosphärischen Modellen gegeben. Anschließend werden die Skalierbarkeitsdefizite dieser Modelle analysiert und daraus Anforderungen für das in der Arbeit entwickelte Konzept formuliert. Dabei wird die Notwendigkeit der dynamischen Lastbalancierung und der Separation der Datenstrukturen der Wolkenmikrophysik erkannt. Kapitel 3 gibt einen Überblick zum Stand der Forschung über dynamische Lastbalancierung und Kopplung paralleler Simulationsmodelle, insbesondere im Hinblick auf Hochskalierbarkeit und der Anwendung in atmosphärischen Modellen. Basierend auf den Anforderungen wird in Kapitel 4 eine lastbalancierte und hochskalierbare Kopplung für detaillierte Wolkenmodelle entworfen. Das Grundkonzept basiert auf der Trennung der Datenstrukturen und Berechnungen der Wolkenmikrophysik vom atmosphärischen Modell, so dass eine dreidimensionale Dekomposition mit dynamischer Lastbalancierung unabhängig vom atmosphärischen Modell erfolgen kann. Die lose Kopplung der zwei Komponenten erfordert den wechselseitigen Datenaustausch zwischen unterschiedlichen Partitionierungen, wobei die Partitionierung der Wolkenmikrophysik dynamisch ist. Es werden an das Problem angepasste Datenstrukturen, parallele und genaue Lastbalancierungsmethoden sowie skalierbare Methoden zum Datenaustausch zwischen den Partitionierungen entwickelt. Abschließend wird in Kapitel 5 zunächst kurz auf die Implementierung der Konzepte in der Bibliothek FD4 eingegangen. In Anschluss werden Ergebnisse von Leistungsmessungen zur Evaluierung und zum Vergleich der Konzepte diskutiert. Dabei wird gezeigt, dass mit den entwickelten Konzepten sowohl der Kommunikationsaufwand als auch die Lastimbalancen erfolgreich reduziert werden. Trotz des zusätzlichen Aufwands der Modellkopplung und der häufig notwendigen Repartitionierung wird das Ziel einer hohen Skalierbarkeit erreicht. Damit wird eine effiziente Nutzung spektraler Mikrophysikmodelle für Gittergrößen typischer Anwendungen regionaler atmosphärischer Modelle ermöglicht.

2 Parallele Simulation von Wolkenprozessen

In diesem Kapitel werden die wesentlichen Grundlagen für die Arbeit eingeführt und die Defizite für eine effiziente parallele Simulation von Wolkenprozessen analysiert. Der folgende Abschnitt gibt einen kurzen Überblick über die Architektur und Programmierung von Hochleistungsrechnern sowie Leistungsbewertungskriterien für parallele Programme. Danach werden in Abschnitt 2.2 atmosphärische Simulationsmodelle und in 2.3 grundlegende Prinzipien zu deren Parallelisierung eingeführt. Anschließend werden in den Abschnitten 2.4 und 2.5 die Bedeutung und Komplexität von Wolkenprozessen sowie deren Modellierung in atmosphärischen Simulationsmodellen dargelegt. Die Skalierbarkeitsdefizite atmosphärischer Modelle mit einer detaillierten Beschreibung von Wolkenprozessen werden in Abschnitt 2.6 am Beispiel des Modells COSMO-SPECS analysiert und in Abschnitt 2.7 allgemein zusammengefasst.

2.1 Parallelrechner und Leistungsbewertungskriterien

Die hohen Anforderungen der in dieser Arbeit betrachteten atmosphärischen Simulationsmodelle an Speicher und Rechenzeit machen in vielen Anwendungsbereichen die Nutzung von Parallelrechnern notwendig. Parallelrechner sind Rechnersysteme, welche mehrere Operationen zugleich ausführen können, um rechenintensive Aufgaben schneller zu lösen als durch sequentielle Ausführung. Parallelität ist heute sowohl innerhalb von CPUs (Superskalarität, Pipelining, Vektorisierung, Multithreading) als auch durch das Zusammenschalten mehrerer CPUs zu sogenannten MIMD-Systemen gegeben (*multiple instruction, multiple data*). Üblicherweise spricht man nur in letzterem Fall von einem Parallelrechner. Für MIMD-Systeme existieren zwei grundsätzliche Prinzipien der Speicherorganisation [87, Seite 20]: Gemeinsamer Speicher (*shared memory*) und verteilter Speicher (*distributed memory*). Bei Systemen mit gemeinsamem Speicher sind CPUs durch ein Verbindungsnetzwerk an ein oder mehrere Speichermodule gekoppelt, welche einen gemeinsamen Adressraum zur Verfügung stellen. Unter anderem zählen auch die heute üblichen Mehrkernprozessoren (*multi-core processor*) zu dieser Kategorie. Dagegen existiert bei Systemen mit verteiltem Speicher kein gemeinsamer Adressraum. Diese Systeme bestehen aus mehreren sogenannten Knoten, welche über ein Verbindungsnetzwerk gekoppelt sind. Die Knoten bestehen jeweils aus einer CPU mit lokalem Speicher oder einem *shared-memory*-System mit mehreren CPUs. Um Daten zwischen den Knoten zu übertragen, müssen explizit Nachrichten gesendet und empfangen werden (*message passing*). Um möglichst viele Knoten miteinander verbinden zu können, werden nicht vollständig vermaschte Netzwerktopologien genutzt, wie zum Beispiel sogenannte *fat trees*, *hypercubes* oder mehrdimensionale Tori. Insgesamt weisen diese Systeme somit inhomogene Bandbreiten und Latenzen sowohl zwischen den Knoten als auch im Vergleich Knoten-interner und Knoten-externer Kommunikation auf. Der Vorteil von Systemen mit verteiltem Speicher sind ein geringerer Realisierungsaufwand und eine weitaus höhere Skalierbarkeit als Systeme mit gemeinsamem Speicher. Die derzeit leistungsfähigsten Parallelrechner sind *distributed-memory*-Systeme, bei denen bis zu mehrere zehntausend Knoten zu einem Supercomputer verbunden werden. Als Mischform existieren auch Systeme mit physikalisch verteiltem Speicher, welcher logisch einen gemeinsamen Adressraum bildet (*virtual shared memory*). Da die Speicherzugriffszeiten abhängig von der Adresse sind, spricht man in diesem Fall

von sogenannten NUMA-Systemen (*non-uniform memory access*). Diese Architektur vermeidet den Engpass des gemeinsamen Speicherbusses für alle CPUs eines *shared-memory*-Systems.

Aus den zwei allgemeinen Architekturprinzipien ergeben sich das *shared-memory*-Modell und das *message-passing*-Modell als die am weitesten verbreiteten Programmiermodelle für Parallelrechner [27, Seite 26]. Das *shared-memory*-Modell wird heute vor allem durch *OpenMP* [16] realisiert. Es ist jedoch durch die Voraussetzung eines Cache-kohärenten *(virtual-)shared-memory*-Systems in Portabilität und Skalierbarkeit eingeschränkt. Für hochskalierende Anwendungen hat sich das *message-passing*-Modell mit dem Standard *MPI* (*Message Passing Interface*) [70] etabliert. Dieses Programmiermodell ist auf die Architektur von Systemen mit verteiltem Speicher zugeschnitten, schließt jedoch gemeinsamen Speicher nicht aus. Beim *message-passing*-Modell können die unabhängig voneinander laufenden Prozesse Daten nur über den expliziten Aufruf von Kommunikationsoperationen austauschen. Die einfachste Operation, die Punkt-zu-Punkt-Nachricht, ist sehr nah an das *distributed-memory*-Rechnermodell angelehnt und bietet somit sehr wenig Abstraktion. Andererseits bietet MPI aber auch kollektive Operationen an, bei denen eine Gruppe von Prozessen gemeinsam Daten mithilfe einer Vielzahl verschiedener Kommunikationsmuster austauschen kann. Des Weiteren definiert der Standard mehrere Erweiterungen, die über das zugrunde liegende *message passing* weit hinausgehen, wie zum Beispiel parallele Ein-/Ausgabe, einseitige Kommunikation, dynamische Prozessverwaltung und virtuelle Topologien. Im Weiteren wird in dieser Arbeit - soweit nicht anders erwähnt - nur auf das *message-passing*-Modell eingegangen.

Zur Bewertung der Leistung paralleler Programme – also der Qualität der Parallelisierung – existieren eine Reihe von Kriterien [87, Seite 167]. Als Maß des relativen Geschwindigkeitsgewinns wird häufig der *Speedup* $S_P(N)$ genutzt. Dabei wird die Ausführungszeit des optimalen sequentiellen Algorithmus $T^*(N)$ für eine Problemgröße N mit der Laufzeit einer parallelen Implementierung $T_P(N)$ auf P Prozessen in Relation gesetzt:

$$S_P(N) = \frac{T^*(N)}{T_P(N)}$$

Ist $S_P(N) = P$, so wird von linearem oder idealem Speedup gesprochen. Typischerweise steht der optimale sequentielle Algorithmus jedoch nicht zur Verfügung, da dieser zum Beispiel nicht bekannt ist oder die Implementierung zu aufwendig ist. In der Praxis wird dieser strengen Definition daher oft nicht gefolgt. Stattdessen wird für $T^*(N)$ häufig die Laufzeit eines sequentiellen Algorithmus genutzt, auf dem der parallele beruht, oder es wird die Laufzeit des parallelen Algorithmus auf einem Prozess herangezogen. Ein weiteres Leistungsbewertungskriterium stellt die *Effizienz* $E_P(N)$ dar. Sie gibt den Anteil der Laufzeit des sequentiellen Algorithmus an den *Kosten* $C_P(N) = P \cdot T_P(N)$ des parallelen Algorithmus wieder [87, Seite 170]:

$$E_P(N) = \frac{T^*(N)}{C_P(N)} = \frac{T^*(N)}{P \cdot T_P(N)} = \frac{S_P(N)}{P}$$

Bei idealem Speedup ist $E_P(N) = 1$. Speedup und Effizienz eignen sich als Maß für die *Skalierbarkeit*, welche das Verhalten der Leistung eines parallelen Programms bei steigender Prozessanzahl erfasst [87, Seite 171]. Abhängig von der Einbeziehung der Problemgröße N bei der Steigerung der Prozessanzahl P unterscheidet man zwei Fälle von Skalierbarkeit. Wird N konstant gehalten, um so die Laufzeit durch steigende Prozessanzahl P zu reduzieren, so spricht man von *starker Skalierbarkeit* (*strong scalability*). Das *Ahmdahlsche Gesetz* stellt hierfür einen Zusammenhang zwischen dem sequentiellen (nicht parallelisierbaren) Anteil eines parallelen Programms und dem maximal erreichbaren Speedup her [87, Seite 170] und zeigt auf, dass starke Skalierbarkeit sehr hohe Anforderungen an die Güte der Parallelisierung stellt:

$$S_P(N) = \frac{1}{f + \frac{1-f}{P}} \leq \frac{1}{f}$$

Dagegen zielt die *schwache Skalierbarkeit* (*weak scalability*) auf das Lösen wachsender Problemgrößen mit steigender Prozessanzahl ab. Idealer Speedup im Sinne der schwachen Skalierbarkeit wird erreicht, wenn mit zueinander proportional wachsender Problemgröße und Prozessanzahl die Laufzeit konstant bleibt.

2.2 Atmosphärische Simulationsmodelle

Bereits 1950 wurde auf dem ersten elektronischen Universalrechner ENIAC unter Leitung von Jule Charney die erste erfolgreiche numerische Wettervorhersage durchgeführt [67]. Seitdem sind atmosphärische Simulationsmodelle wesentlich komplexer geworden und profitieren von dem Leistungszuwachs von Computern, dem wachsenden Verständnis über physikalische und chemische Prozesse in der Atmosphäre sowie den Fortschritten in der meteorologischen Beobachtung. Heute werden atmosphärische Modelle in vielen Bereichen eingesetzt. Zu den wichtigsten zählen die numerische Wettervorhersage [102, 105], die Vorhersage von Klimaveränderungen auf globaler [86] und regionaler Skala [89], die Untersuchung anthropogener und natürlicher Emissionen [21, 34, 38] sowie der Katastrophenschutz bei Bränden, Chemie- und Nuklearunfällen [45].

Atmosphärische Simulationsmodelle ermöglichen eine zeitliche Prognose des Zustands der Atmosphäre beginnend bei einem bekannten Anfangszustand. Die Dynamik wird durch die sogenannten Grundgleichungen beschrieben [29, Seite 169]. Dies sind partielle Differenzialgleichungen (PDEs) basierend auf den Erhaltungsgleichungen für Masse (Kontinuitätsgleichung), Energie (Erster Hauptsatz der Thermodynamik) und Impuls (Bewegungsgleichungen) sowie der Zustandsgleichung für ideale Gase:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \rho \mathbf{v} \qquad \text{(Kontinuitätsgleichung)}$$

$$\frac{\partial T}{\partial t} = -\nabla \cdot T\mathbf{v} + \frac{1}{\rho c_p}\frac{dp}{dt} + \frac{Q_T}{c_p} \qquad \text{(Erster Hauptsatz der Thermodynamik)}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla \cdot \mathbf{v}\mathbf{v} - 2\Omega \times \mathbf{v} - \nabla\Phi - \frac{1}{\rho}\nabla p \qquad \text{(Bewegungsgleichungen)}$$

$$p = R\rho T \qquad \text{(Zustandsgleichung für ideale Gase)}$$

Darin gelten folgende Definitionen für Symbole:

t	... Zeit	c_p	... Wärmekapazität trockener Luft
ρ	... Dichte	Q_T	... Wärmequellen
T	... Temperatur	Ω	... Winkelgeschwindigkeit der Erde
p	... Druck	Φ	... Geopotential der Erde
$\mathbf{v}$	... Geschwindigkeitsvektor	R	... Universelle Gaskonstante
		∇	... Gradient (Nabla-Operator)

Die Variablen Dichte ρ, Temperatur T, Druck p sowie der Geschwindigkeitsvektor $\mathbf{v}$ sind die Zustandsvariablen der atmosphärischen Dynamik. Überdies enthalten Atmosphärische Modelle Bilanzgleichungen für Wasserdampf und Flüssigwasser, da die oben aufgeführten Gleichungen nur Vorgänge der trockenen Atmosphäre beschreiben. Die aufgeführten Grundgleichungen werden vom sogenannten *dynamischen Kern* numerisch gelöst und bilden die Grundlage jedes atmosphärischen Simulationsmodells.

Um die kontinuierlichen Gleichungen numerisch zu lösen, werden sie räumlich und zeitlich diskretisiert. Für die räumliche Diskretisierung werden meist rechteckige Gitter mit vertikal terrainfolgenden Koordinaten verwendet [102, 105]. Dabei werden die physikalischen Größen im

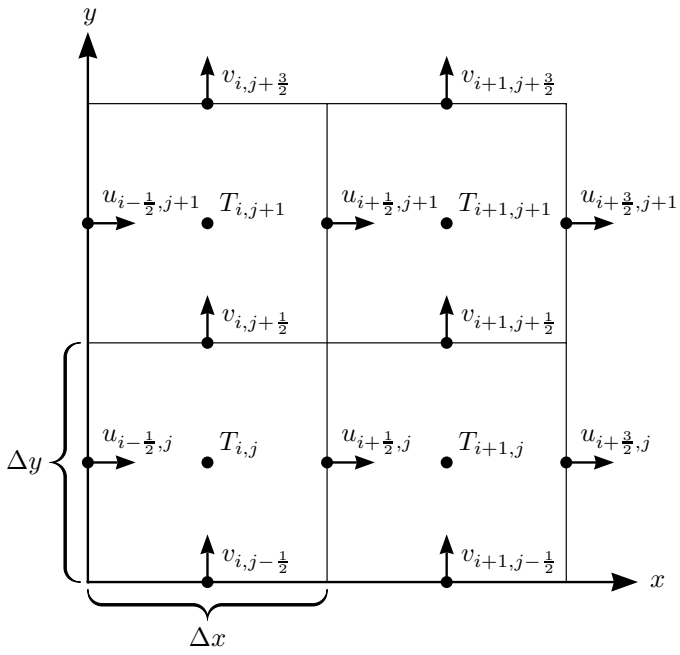


Abbildung 2.1: Versetzte Anordnung von skalaren Größen (T) und vektoriellen Größen (u, v) in einem zweidimensionalen Arakawa-C Gitter (nach [102]).

sogenannten *Arakawa-C Gitter* versetzt angeordnet, da dies eine genauere Repräsentation der Differenzialoperatoren ermöglicht [93]. Skalare Größen werden im Mittelpunkt der Gitterzellen und vektorielle Größen an den entsprechenden Flächen der Zellen definiert. Abbildung 2.1 stellt dieses Schema für den zweidimensionalen Fall dar. Zur Zeitintegration kommen üblicherweise Leapfrog-Verfahren oder Runge-Kutta-Verfahren höherer Ordnung zum Einsatz [93, 102]. Zudem werden oft implizite Zeitintegrationsmethoden in der Vertikalen verwendet, da die feine Auflösung der Höhenschichten zu einer starken Einschränkung der maximalen Schrittgröße expliziter Verfahren führt. Grund ist das CFL-Kriterium (Courant-Friedrichs-Levy), welches eine Bedingung für die numerische Stabilität von Strömungssimulationen darstellt [29, Seite 250].

Atmosphärische Modelle werden auf unterschiedlichen Größenskalen eingesetzt. Auf der lokalen und regionalen Skala umfasst das Rechengebiet je nach Anwendung einzelne Stadtquartiere bis hin zu Kontinenten oder einer ganzen Hemisphäre. Dabei werden typischerweise Maschenweiten im Bereich von 100 m bis 30 km eingesetzt. Das vertikale Gitter besteht aus meist weniger als 60 inhomogen aufgelösten Höhenschichten und reicht bis maximal 30 km Höhe über dem Erdboden. Für globale Modelle, welche die gesamte Erdkugel umfassen, werden auch nicht rechteckige horizontale Gitter oder spektrale Methoden verwendet, da so die Singularitäten an den Polen vermieden werden [112, Seite 387][117].

Viele physikalische Phänomene in der Atmosphäre spielen sich auf Skalen unterhalb der verwendeten Gitterweite ab und werden daher von den Modellen nicht explizit erfasst. Dazu gehören Strahlung und Wolkenmikrophysik, welche Prozesse auf molekularer Ebene darstellen, sowie Turbulenz, konvektive Wolken- und Niederschlagsbildung und orografische Effekte, welche über einem weiten Bereich von Größenskalen auftreten. Diese Prozesse müssen mittels *Parametrisierungen* beschrieben werden [39], welche diese subskaligen Prozesse mithilfe der nur an den Gitterpunkten des Modells vorliegenden Informationen beschreiben. Im einfachsten Fall sind dies nur Abschätzungen, aber meist werden detaillierte Modellvorstellungen für den Ablauf dieser Prozesse entwickelt. Dabei müssen freie Parameter durch empirische Vorgehensweise quantifiziert und die Modelle in langwierigen Verfahren verifiziert werden. Ein weitere Schwierigkeit stellt die Skalentrennung dar: Einige Prozesse, wie zum Beispiel die konvektive Wolken- und

Niederschlagsbildung, werden mit den Gitterweiten der Modelle zum Teil explizit aufgelöst [39]. Die kleinskaligen Anteile müssen jedoch parametrisiert werden. Eine Folge der Skalentrennung ist, dass bei der Änderung der Gitterweite die verwendeten Parametrisierungen überprüft und gegebenenfalls angepasst werden müssen.

Atmosphärische Simulationsmodelle bilden nur dynamische und physikalische Vorgänge in der Atmosphäre ab. Für viele wissenschaftliche Fragestellungen ist jedoch die Wechselwirkung mit weiteren Prozessen der Geosphäre von Interesse. Daher werden in zunehmendem Maße teilweise unabhängig voneinander entwickelte Simulationsmodelle zu komplexen Modellsystemen gekoppelt, um so mehr Aspekte des Gesamtsystems einbeziehen zu können. Ein Beispiel sind Klimamodelle [1, 117], bei denen Modelle zur Beschreibung der Atmosphäre, der Ozeane, des Meereises, der Landoberfläche, der Atmosphärenchemie und weiterer Elemente der Geosphäre gekoppelt werden. Das ultimative Ziel ist die Entwicklung von sogenannten Erdsystemmodellen, welche alle klimarelevanten Prozesse berücksichtigen. Ein weiteres Beispiel für die Modellkopplung sind Chemie-Transport-Modelle (*air quality models*) [21, 34, 38]. Diese Modellsysteme simulieren chemische Reaktionen von Luftschadstoffen und deren Ausbreitung in Abhängigkeit vom Wettergeschehen und der Aktivität von Emissionsquellen.

2.3 Parallele atmosphärische Simulationsmodelle

In vielen Anwendungsbereichen ist der Speicher- und Rechenaufwand atmosphärischer Modelle so groß, dass sie nur auf Parallelrechnern ausgeführt werden können [122]. Zum Beispiel müssen operationelle Wettervorhersagen innerhalb fest vorgegebener Laufzeiten und Klimaprognosen über sehr lange Vorhersagezeiträume mit möglichst feinen Gitterauflösungen berechnet werden. Die notwendige Parallelisierung basiert dabei auf der Zerlegung des Rechengitters in einzelne Teilgebiete (*Partitionen*), welche dann weitestgehend unabhängig voneinander berechnet werden können. Dabei wird das Gitter im Allgemeinen nur in den horizontalen Dimensionen zerlegt, nicht jedoch in der Vertikalen [92, 122]. Gründe sind die hohen vertikalen Abhängigkeiten einiger physikalischer Parametrisierungen, wie Strahlung und Konvektion, sowie die Anforderungen der impliziten Zeitintegrationsverfahren. Die Zerlegung des Rechteckgitters regionaler Modelle erfolgt typischerweise schachbrettartig in M×N annähernd gleich große, rechteckige Partitionen [92, 122]. Zur Lösung der Modellgleichungen müssen zu jedem Zeitschritt mindestens einmal Variablen von den Rändern zwischen benachbarten Partitionen ausgetauscht werden. In Abhängigkeit vom verwendeten numerischen Verfahren ist dafür Kommunikation mit den vier unmittelbar benachbarten Partitionen oder zusätzlich auch mit den vier diagonal benachbarten Partitionen notwendig. Als Programmiermodell wird heute vor allem das *message-passing*-Modell durch MPI in atmosphärischen Simulationsmodellen realisiert. Das *Weather Research & Forecasting Model* (WRF) nutzt aber auch zusätzlich OpenMP zur *shared-memory*-Parallelisierung und kann beide Paradigmen zugleich als *hybride* Parallelisierung einsetzen [72]. Bei globalen atmosphärischen Simulationsmodellen basierend auf spektralen Methoden wird das Gitter typischerweise ebenfalls horizontal in rechteckige Teilgebiete zerlegt [1, 75]. Die jeden Zeitschritt nötige Transformation zwischen dem spektralen und dem physikalischen Raum führt hierbei jedoch zu zusätzlicher globaler Kommunikation.

In der operationellen Vorhersage werden heute Wettermodelle auf maximal mehreren hundert Prozessorkernen parallel berechnet, wie zum Beispiel COSMO, das Modell des *Consortium for Small-scale Modeling*, welches bei vielen europäischen Wetterdiensten zum Einsatz kommt.[1] Höhere Prozessorzahlen sind vor allem Gegenstand von Benchmarks. Für das Modell WRF

[1] `http://www.cosmo-model.org/content/tasks/operational/default.htm` (März 2012)

wurden Messungen mit einem hochaufgelösten Modellgebiet auf bis zu 18 432 Kernen durchgeführt.[2] Zudem wurde mit WRF ein idealisiertes globales Testszenario mit bis zu 65 536 Kernen berechnet, um so Erfahrungen mit hochaufgelösten Simulationen und deren Skalierbarkeit zu sammeln [72]. Für das speziell zur hochskalierbaren Anwendungen in Klimamodellen entwickelte globale atmosphärische Modell CAM/HOMME (*Community Atmospheric Model/High Order Multiscale Modeling Environment*) konnten Taylor et al. [106] eine Skalierbarkeit bis zu 86 200 Prozessorkernen erreichen. Die drei genannten Benchmarks wurden alle auf IBM-BlueGene/L-Systemen durchgeführt.

2.4 Wolken- und Niederschlagsprozesse

Wolken und Niederschlag sind von hoher Bedeutung für die Entwicklung von Wetter und Klima. Sie beeinflussen den Strahlungshaushalt der Erde und sind Teil des Wasserkreislaufs. Wolken reflektieren einen Teil der auf die Erde treffenden Sonnenstrahlung [31], woraus ein abkühlender Effekt resultiert. Andererseits tragen sie aber auch zum natürlichen Treibhauseffekt bei, da sie an der Reflexion der langwelligen Ausstrahlung der Erdoberfläche zur Erde zurück beteiligt sind. Die Bilanz aus kühlenden und wärmenden Effekten durch einen veränderten Grad an Wolkenbedeckung ist eine bedeutende Fragestellung in Bezug auf den Klimawandel [86]. Im Wasserkreislauf speichern Wolken nur einen geringen Anteil des Wassers auf der Erde; die Niederschläge sind jedoch die wichtigste Quelle für Süßwasser und ermöglichen die Bildung von Flüssen, Seen und Grundwasser. Somit sind Wolken- und Niederschlagsprozesse Voraussetzung für einen Großteil des heutigen Lebens auf der Erde.

Die physikalischen Prozesse, die zur Bildung von Wolken und Niederschlag führen, finden auf mikroskopischen Skalen statt und werden als Wolkenmikrophysik bezeichnet. Die Entstehung von Wolken beginnt mit der Bildung von Wolkentropfen oder Wolkeneis aus Wasserdampf bei hinreichend hoher relativer Luftfeuchte. Die häufigste Ursache dafür ist die Hebung von Luftmassen und die damit einhergehende adiabatische Abkühlung, welche zum Ansteigen der relativen Luftfeuchte führt. Zusätzlich spielen *Aerosolpartikel* eine wichtige Rolle, da sie als Kondensationskerne dienen [112, Seite 193]. Aerosolpartikel sind in der Luft schwebende Teilchen, wie zum Beispiel Salzkristalle, Staub, Pollen und Rauch aber auch anthropogene Emissionen aus Verkehr und Industrie. Ihre Größe reicht im Allgemeinen von wenigen Molekülen bis hin zu Radien größer als 100 μm [85, Seite 225]. Für die Bildung von Wolkentropfen dienen vor allem hygroskopische Aerosolpartikel als Kondensationskerne. Bei Temperaturen von unter -10 °C können sich auch direkt aus Wasserdampf mittels Eiskernen Wolkeneispartikel bilden. Als Eiskerne dienen – im Gegensatz zu Kondensationskernen – nur bestimmte wasserunlösliche Aerosole mit einer Eis-ähnlichen Kristallstruktur, wie zum Beispiel Mineralstaub.

Sämtliche Erscheinungsformen flüssigen Wassers (Wolkentropfen, Regentropfen) und gefrorenen Wassers (z. B. Wolkeneis, Schneeflocken, Hagel, Graupel) in der Atmosphäre werden als *Hydrometeore* bezeichnet. Dabei sind die Übergänge zwischen diesen Erscheinungsformen fließend; insbesondere existieren auch gemischte Hydrometeore, welche sowohl aus flüssigem als auch gefrorenem Wasser bestehen. Durch eine Vielzahl mikrophysikalischer Prozesse können Hydrometeore wachsen, miteinander verschmelzen, aufbrechen, den Aggregatzustand wechseln oder die räumliche Struktur ändern. Die Umwandlungsprozesse zwischen Wasserdampf, Wassertropfen und Eispartikeln sind in Abbildung 2.2 schematisch dargestellt. Dabei sind vor allem die Eisprozesse sehr komplex, da zum Beispiel nur etwa einer von einer Million Aerosolpartikeln als Eiskern geeignet ist und Eispartikel in verschiedensten geometrischen Formen mit jeweils unterschiedlichen Eigenschaften existieren [31]. Durch Wachstumsprozesse können Hydrometeore

[2] `http://www.mmm.ucar.edu/wrf/WG2/benchv3/` (März 2012)

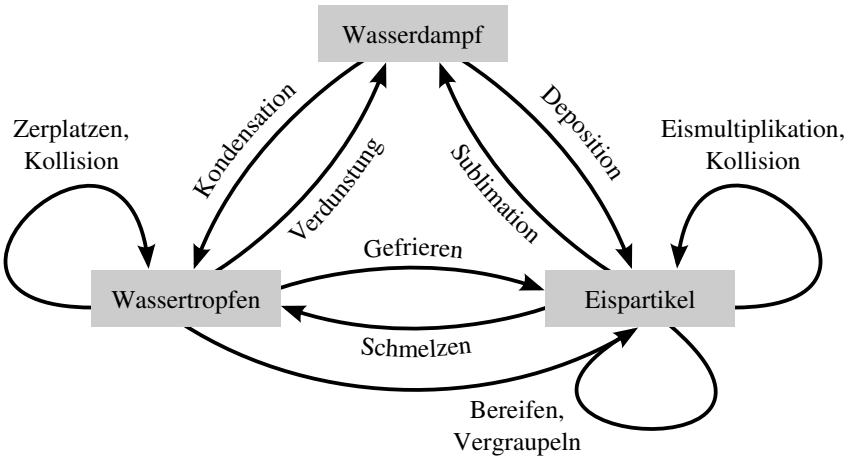


Abbildung 2.2: Schematische Darstellung mikrophysikalischer Prozesse der Wolkenbildung (nach [5]).

schwer genug werden, um als Niederschlag zu Boden zu fallen. Dabei entsteht eine weitere Wechselwirkung mit Aerosolpartikeln, welche von fallenden Hydrometeoren aufgesammelt werden.

Die Anzahl, Größe und chemische Zusammensetzung der Aerosolpartikel hat einen wichtigen Einfluss auf die Wolkenbildung [18, 86]. Zum Beispiel führt eine höhere Anzahl von Kondensationskernen zu einer höheren Anzahl von Wolkentropfen. Da der Gesamtwassergehalt weitgehend durch die Dynamik der Atmosphäre und weniger durch die Mikrophysik bestimmt wird, bleiben in diesem Fall die Tropfen kleiner. Dies beeinflusst sowohl die optischen Eigenschaften als auch die Entwicklung der Wolke und damit Zeit, Ort und Menge des Niederschlags. Die Zusammenhänge sind jedoch sehr komplex und zum Teil noch nicht vollständig verstanden, so dass es nicht möglich ist allein von einer Erhöhung der Aerosolpartikelkonzentration auf eine Abnahme oder Zunahme des Niederschlags zu schließen [51]. Dies hängt sowohl von den Umgebungsbedingungen wie Wind und Luftfeuchte als auch von der Art der Wolke ab.

2.5 Modellierung von Wolkenprozessen

Je nach Art des Modells und der verwendeten Gitterweite werden Wolkenprozesse unterschiedlich detailliert beschrieben [35]. Zum Beispiel liefern Klimamodelle nur statistische Aussagen über die Niederschlagsmenge auf einem meist sehr groben Rechengitter, so dass relativ einfache Parametrisierungen ausreichen. Wettermodelle dagegen nutzen feinere Gitterauflösungen und sollen eine möglichst genaue Vorhersage des Niederschlags liefern. Daher benötigen sie eine detailliertere Beschreibung der Wolkenprozesse als Klimamodelle. Zudem sind auch die Anforderungen an Rechenressourcen ein Kriterium. Zum Beispiel müssen Wettervorhersagemodelle einen festen operationellen Zeitplan einhalten.

Generell werden Wolkenprozesse durch eine mehr oder weniger grobe Einteilung der Hydrometeore in Kategorien, wie zum Beispiel Wolkentropfen und Eispartikel, modelliert. Die Umwandlungsprozesse zwischen diesen Kategorien und dem Wasserdampf werden durch Erhaltungsgleichungen für die Wassermasse mathematisch beschrieben [26, Seite 41]. Prinzipiell kann man zwei Arten der Beschreibung von Wolkenprozessen in atmosphärischen Modellen unterscheiden: Die verbreitet eingesetzten Bulk-Parametrisierungen sowie die detaillierteren spektralen Wolkenmikrophysikmodelle. Der wesentliche Unterschied ist die Herangehensweise bei der Modellierung der Größenverteilung der Hydrometeore.

2.5.1 Bulk-Parametrisierungen

Viele Bulk-Parametrisierungen (*bulk microphysical parameterizations*) basieren auf dem von Kessler entwickelten Modell [66], welches lediglich Wolkentropfen und Regentropfen als Hydrometeorkategorien beschreibt. Heute beinhalten die Parametrisierungen meist noch weitere Kategorien wie Wolkeneis, Schnee und Graupel [63]. Bei sogenannten Ein-Momenten-Verfahren wird jede der Kategorien nur durch die Massenkonzentration als prognostische Größe beschrieben. Dabei werden für die Größenverteilungen der Partikel einfache Annahmen getroffen, zum Beispiel eine logarithmische Normalverteilung. Eine Erweiterung stellen die Zwei-Momenten- oder Mehr-Momenten-Verfahren dar, welche weitere prognostische Variablen je Hydrometeor einführen, wie zum Beispiel die Anzahldichte der Partikel [97]. Dadurch können die Prozesse der Hydrometeore detaillierter beschrieben und die Größenverteilung genauer parametrisiert werden. Jedoch wird auch hier eine bestimmte Verteilungsfunktion vorgegeben. Der Vorteil der Ein-Momenten-Verfahren ist die langjährige Erfahrung und Validierung in operationellen Modellen und der verhältnismäßig geringe Rechenaufwand der Algorithmen. Daher werden sie am weitaus häufigsten eingesetzt.

2.5.2 Spektrale Wolkenmikrophysikmodelle

Bei spektralen Wolkenmikrophysikmodellen [49] (*spectral bin cloud microphysics models*) wird die Größenverteilung der Hydrometeore durch eine explizite Diskretisierung in etwa 20 bis über 100 Größenklassen (*size bins*) wesentlich realistischer abgebildet als bei den oben angeführten Parametrisierungen. Dabei werden auch hier die modellierten Hydrometeore in Kategorien unterteilt und mit separaten Größenspektren beschrieben. Es werden üblicherweise mindestens drei Kategorien verwendet: Aerosolpartikel (als potentielle Kondensationskerne), Wolken- und Regentropfen, sowie Eispartikel. Detailliertere Modelle unterteilen die Eispartikel je nach Erscheinungsform noch in weitere Kategorien, wie etwa das Modell HUCM [50]. Einige Modelle, wie SPECS [99], stellen Aerosolpartikel und Tropfen in einem gemeinsamen Spektrum dar, so dass die Kondensation explizit modelliert werden kann. Die einzelnen Spektren enthalten wiederum mehrere Variablen wie Masse und Anzahlkonzentration, so dass in der Summe hunderte oder sogar mehr als eintausend prognostische Variablen zu Beschreibung der Wolkenmikrophysik je Gitterzelle dienen. Durch die detaillierte Beschreibung der Größenverteilung können viele mikrophysikalische Prozesse explizit durch entsprechende Gleichungen modelliert werden, wie zum Beispiel das Partikelwachstum durch Kollision mithilfe einer stochastischen Gleichung (*stochastic collection equation* [85]). Der Nachteil der spektralen Wolkenmikrophysikmodelle sind die sehr hohen Anforderungen an Rechenzeit und Speicher, was den Einsatz in dreidimensionalen Vorhersagemodellen erschwert. Des Weiteren sind sie durch das Fehlen langjähriger Erfahrungen im praktischen Einsatz weitaus weniger verifiziert als die oft eingesetzten Parametrisierungen.

Anwendung spektraler Wolkenmikrophysikmodelle

Mehrdimensionale atmosphärische Simulationsmodelle mit spektraler Beschreibung der Wolkenmikrophysik finden heute hauptsächlich in wissenschaftlichen Einzelstudien Anwendung, um den Einfluss von Aerosolen auf die Wolken- und Niederschlagsbildung zu untersuchen [36, 48, 83]. Dabei kann die Größenverteilung und Art der Aerosolpartikel wesentlich genauer modelliert werden als bei Bulk-Parametrisierungen. Von hohem Interesse ist hierbei vor allem die Wirkung verschiedener Arten von Aerosolen auf die Entwicklung schwerer Gewitter und tropischer Wirbelstürme. Zudem dienen spektrale Modelle zur Bewertung und Verbesserung der operationell eingesetzten Parametrisierungen [58, 68, 78, 98]. Durch die detaillierten Modelle liegen Referenzdaten vor, welche alle relevanten Variablen umfassen und wesentlich höher aufgelöst

sind als reale Messdaten. Da auch die chemische Zusammensetzung der Aerosolpartikel mit der Wolken- und Niederschlagsbildung in Wechselwirkung steht, wird der spektrale Ansatz auch mit der Modellierung chemischer Spezies kombiniert [43]. Zukünftige Anwendungen in hochaufgelösten Modellen werden in der detaillierten Prognose des räumlichen und zeitlichen Auftretens von Niederschlägen und vor allem Unwettern gesehen, sowie deren Beeinflussung durch gezieltes Einbringen von Aerosolen in die Atmosphäre [48, 68].

Bisherige Anwendungen beschränken sich aufgrund des hohen Aufwands auf ein begrenztes Simulationsgebiet und meist weniger als 24 Stunden Simulationszeit. In den oben zitierten Publikationen zu Studien mit spektralen Modellen wird oft auf den vergleichsweise hohen Rechenaufwand eingegangen. Die Angaben reichen dabei von einem 10- bis 50-mal höherem Aufwand als mit der Ein-Momenten-Parametrisierung [48, 68]. Jedoch wird kaum auf die Rolle der Hochleistungsrechner und Aspekte der Parallelisierung zur Beschleunigung der Simulationsmodelle eingegangen.

Das Mikrophysikmodell SPECS [99] verwendet 66 Größenklassen und wird in einem atmosphärischen Modell zur Simulation einer idealisierten Kumuluswolke genutzt [35]. Dabei werden für die zweistündige Simulation über 25 Stunden Rechenzeit auf 100 Prozessorkernen einer SGI Altix 4700 benötigt. Die Steigerung der Parallelität ergab eine verhältnismäßig geringe Laufzeitreduktion, was auf den relativ hohen Kommunikationsaufwand und Lastimbalancen durch die räumliche Variabilität von Wolken zurückgeführt wird. Das Modell SPECS dient in Abschnitt 2.6 als Grundlage zur Analyse der Defizite bei der effizienten Nutzung spektraler Wolkenmodelle in parallelen atmosphärischen Modellen.

Lynn et al. [68] nutzten für die Simulation eines Gewitters mit dem Wettermodell MM5 und dem spektralen Mikrophysikmodell Fast-SBM eine SGI Origin System mit 32 Prozessoren. Das zum Einsatz in dreidimensionalen atmosphärischen Modellen vereinfachte spektrale Mikrophysikmodell Fast-SBM verfügt über 33 Größenklassen. Zur Simulation von 15 Stunden benötigte das Modellsystem ca. 8 bis 9 Tage Rechenzeit. Khain et al. [48] simulieren mit dem Wettermodell WRF und dem gleichen Mikrophysikmodell die Entwicklung des Hurrikans Katrina, der Ende August 2005 vor allem im Südosten der Vereinigten Staaten schwere Schäden verursachte. Für die Simulation über 3 Tage geben die Autoren 10 Tage Rechenzeit auf einem „Standard-PC-Cluster mit acht Prozessoren“ an. A. Khain berichtet, dass auch bei diesem Modell Lastimbalancen durch die spektrale Mikrophysik auftreten und auf 192 Prozessen die Berechnung in Echtzeit möglich ist (pers. Komm., 7. Juli 2011).

Mit dem Modell DESCAM-3D wurden Simulationen mit spektraler Mikrophysik auf dem bisher größten Rechengitter mit 256×256×80 Gitterpunkten durchgeführt [83]. Das Modell verfügt über 39 Größenklassen. Die Autoren nutzen eine Parallelisierung mit 64 Prozessen, wobei auch hier die Lastimbalancen und der hohe Kommunikationsaufwand als Probleme identifiziert wurden (W. Wobrock, pers. Komm., 8. Juli 2011).

2.6 Skalierbarkeit detaillierter Simulationen von Wolkenprozessen

Durch den hohen Rechenaufwand spektraler Mikrophysikmodelle ist die effiziente Nutzung hochparalleler Rechnersysteme eine wichtige Voraussetzung, um diese Modelle zukünftig verstärkt in atmosphärischen Simulationen einzusetzen. So kann man zum einen die Simulationen beschleunigen, welche heute selten in Echtzeit berechnet werden können. Zum anderen wird es dadurch möglich, hochaufgelöste Simulationen über größere Gebiete durchzuführen, dass heißt also die Gittergröße zu erhöhen. In diesem Abschnitt werden daher anhand des Modellsystems COSMO-SPECS die Defizite untersucht, die bisher eine hohe Skalierbarkeit erschweren.

2.6.1 Das Modellsystem COSMO-SPECS

Das Modellsystem COSMO-SPECS wurde für detaillierte Studien zur Interaktion von Wolken- und Niederschlagsprozessen mit Aerosolen am Leibniz-Institut für Troposphärenforschung Leipzig (IfT) entwickelt [36]. Es basiert auf dem regionalen atmosphärischen Simulationsmodell des *Consortium for Small-scale Modeling*[3] (COSMO-Modell [105], im Folgenden kurz COSMO) und wurde mit dem am IfT entwickelten spektralen Wolkenmikrophysikmodell SPECS (*spectral bin cloud microphysics* [99]) erweitert.

COSMO wurde ursprünglich als „Lokal-Modell“ (LM) beim Deutschen Wetterdienst entwickelt und wird in der regionalen Wettervorhersage operationell eingesetzt. Heute wird es durch das *Consortium for Small-scale Modeling* weiterentwickelt und von den Mitgliedern – vor allem europäische Wetterdienste – für die Wettervorhersage und Forschung eingesetzt. Darüber hinaus wird COSMO auch in vielen weiteren Forschungseinrichtungen genutzt. Wie die meisten regionalen Modelle verwendet COSMO ein rechteckiges Arakawa-C Gitter mit vertikal terrainfolgenden Koordinaten (siehe Abschnitt 2.2). Die Parallelisierung für Systeme mit verteiltem Speicher erfolgt durch Aufteilung des Gitters unter den Prozessen und *message passing* mittels MPI. Die Teilung erfolgt entlang der horizontalen Dimensionen in M×N annähernd gleich große, rechteckige Partitionen. Die Zustandsvariablen und weitere diagnostische Variablen sind jeweils als ein zusammenhängendes Datenfeld je Partition und Variable implementiert. Zur Zeitintegration der Modellgleichungen werden für jeden Gitterpunkt Variablen benachbarter Gitterpunkte als Eingabe benötigt. Um dies auch an den Grenzen der Partitionen zu gewährleisten, erhalten die Datenfelder eine Erweiterung am Rand um sogenannte *Geisterzellen*, welche mit den Kopien der Variablen von den Nachbarpartitionen belegt werden (siehe auch Abschnitt 3.1). Dies erfordert den Austausch von Nachrichten zwischen den Prozessen.

Zur Erweiterung von COSMO mit SPECS wurde die bestehende Bulk-Parametrisierung der Wolkenmikrophysik von COSMO durch SPECS ersetzt [35, 36]. Dadurch wurde eine Vielzahl neuer Variablen zur Beschreibung der spektralen Mikrophysik eingeführt. SPECS unterteilt Partikel in drei Kategorien: Flüssige Partikel (feuchte Aerosolpartikel als Kondensationskeime, Wolken- und Regentropfen), Partikel gemischter Phase (gefrorene oder teilweise gefrorene Partikel wie Wolkeneispartikel, Graupel, Hagel und Schnee) sowie unlösliche Partikel (Eiskerne). Für jede der Kategorien werden Mischungsverhältnisse für Masse und Anzahl der Partikel je Kilogramm Luft modelliert. Zusätzlich werden je nach Kategorie mit weiteren Variablen der Anteil gefrorenen Wassers sowie löslicher und unlöslicher Aerosolpartikel beschrieben. Diese insgesamt 11 Variablen werden nach dem spektralen Ansatz in Größenklassen unterteilt. SPECS nutzt hierbei 66 logarithmisch unterteilte Klassen von 1 nm bis 4,2 mm Radius. In Summe wurden damit 726 zusätzliche Variablen je Gitterpunkt zu COSMO hinzugefügt. Die Aufteilung der Daten über die Prozesse sowie der Nachrichtenaustausch zur Aktualisierung der Geisterzellen erfolgt nach dem gleichen Schema wie für die bestehenden Datenfelder.

Die mikrophysikalischen Prozesse laufen auf kleineren Zeitskalen ab als die dynamischen Prozesse der Atmosphäre. Um die numerische Stabilität zu gewährleisten, erfolgt die Zeitintegration in SPECS daher mit kleineren Zeitschritten als in COSMO. Je Zeitschritt Δt^{dyn} in COSMO werden m kleine Schritte $\Delta t^{mp} = \Delta t^{dyn}/m$ in SPECS berechnet. Abbildung 2.3 stellt die Kopplung von COSMO und SPECS schematisch dar. COSMO berechnet zuerst einen Zeitschritt Δt^{dyn} für das Intervall $[t_d, t_{d+m}]$. Danach folgen m kleine Zeitschritte Δt^{mp} in SPECS für das gleiche Intervall. Dabei werden die Eingabevariablen für den i-ten Schritt Temperatur $T(t_{d+i})$, Druck $p(t_{d+i})$ und Dichte $\rho(t_{d+i})$ über dieses Intervall für $i = 1 \ldots m$ linear interpoliert. Bei der Temperatur wird zusätzlich die Änderung durch die Mikrophysik berücksichtigt. Nach Berechnung der m Schritte in SPECS werden die über das Größenspektrum aufsummierten Mischungsverhältnisse für die

[3] `http://www.cosmo-model.org` (März 2012)

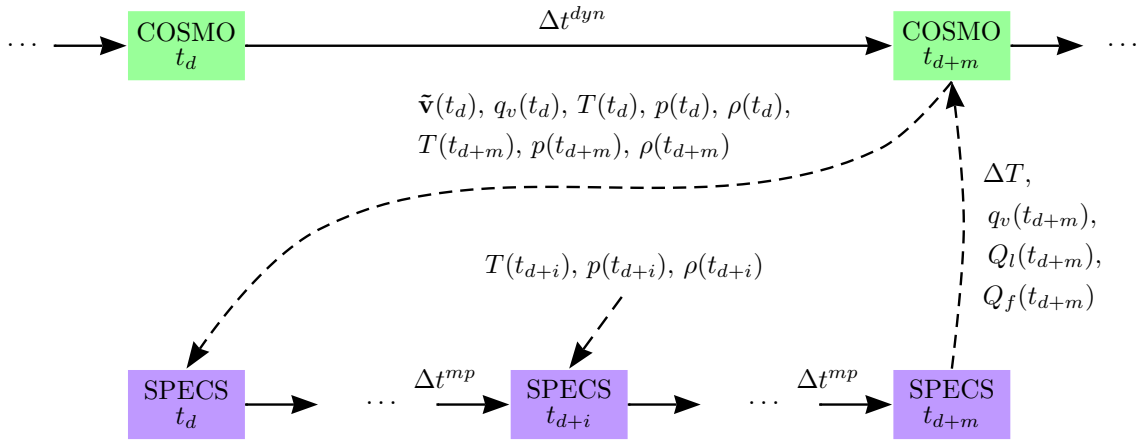


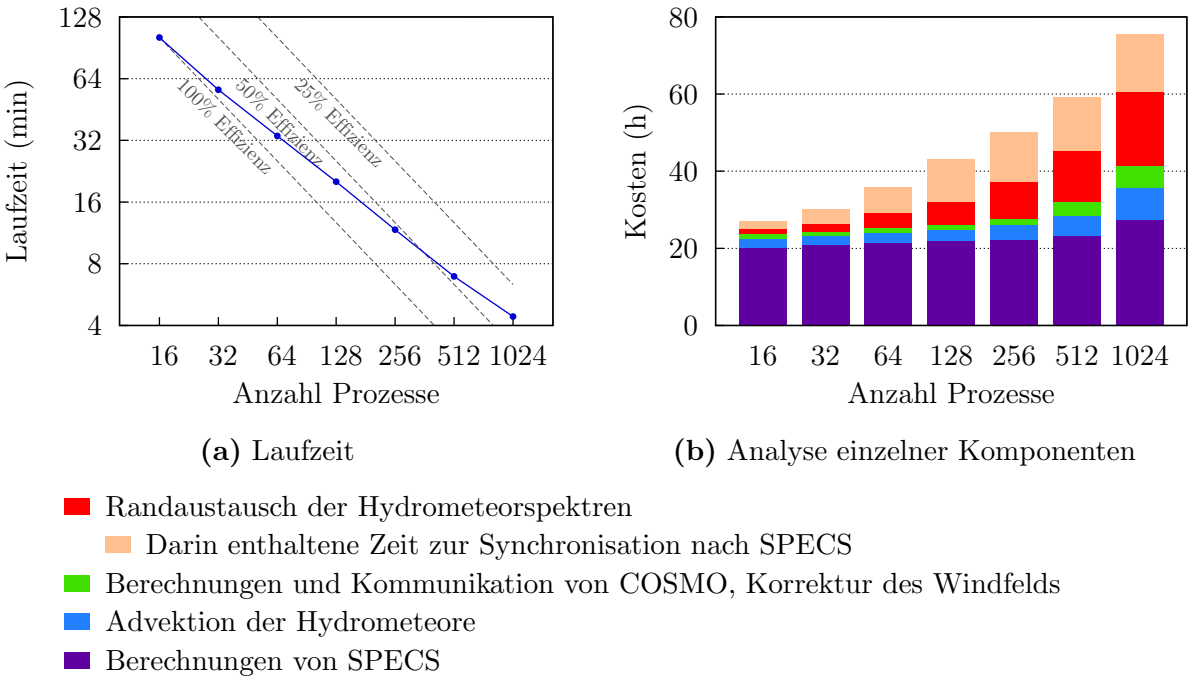
Abbildung 2.3: Schema der Kopplung von COSMO und SPECS für einen Zeitschritt Δt^{dyn} in COSMO (nach [36]).

Masse flüssiger und gefrorener Partikel Q_l und Q_f sowie der Wasserdampfgehalt q_v und die Temperaturänderung ΔT aufgrund mikrophysikalischer Prozesse für den Zeitpunkt t_{d+m} an COSMO zurückgegeben. Vor jedem Aufruf von SPECS wird die Advektion von Wasserdampf und der Hydrometeore für den Zeitschritt Δt^{mp} berechnet. Dafür wird das Windfeld (d. h. der Geschwindigkeitsvektor $\mathbf{v}$) von COSMO herangezogen, für das jedoch die Erhaltung der Masse nicht garantiert ist. Daher wird nach jedem Δt^{dyn} ein modifiziertes Windfeld $\tilde{\mathbf{v}}$ berechnet, für welches die diskrete Massenerhaltungsgleichung auf dem Gitter erfüllt ist [53]. Zusammenfassend werden für jeden kleinen Zeitschritt Δt^{mp} drei Aktivitäten durchgeführt: (1) Austausch des Wasserdampfs und der Hydrometeorspektren zwischen den Partitionsrändern, (2) Berechnung der Advektion von Wasserdampf und der Hydrometeore, (3) Berechnung der Wolkenmikrophysik mit SPECS.

2.6.2 Analyse der Skalierbarkeit

Die Analyse der Skalierbarkeit von COSMO-SPECS wurde mithilfe eines idealisierten Testszenarios durchgeführt [36]. Dabei führt eine mittig im Simulationsgebiet platzierte Wärmeblase zu aufsteigender Luft und infolgedessen zur Bildung einer Kumuluswolke. Das Zentrum der sphärischen Wärmeblase liegt in 1,4 km Höhe. Die maximale Temperaturabweichung beträgt 2 K. Innerhalb der 30 min Simulationszeit bildet sich die Wolke, Eispartikel entstehen und es kommt zu Niederschlag. Zusätzlich ist ein horizontaler Wind vorgegeben. Das dreidimensionale Gitter umfasst 64×64 horizontale Gitterzellen mit einer Auflösung von 1 km und 48 inhomogen aufgelöste Höhenschichten bis 18 km Höhe. Die Schrittweiten für COSMO und SPECS sind $\Delta t^{dyn} = 10\,\text{s}$ und $\Delta t^{mp} = 2{,}5\,\text{s}$.

Die hohe Rechenlast von SPECS zwingt zu einer höheren Parallelität als in atmosphärischen Modellen ohne spektrale Wolkenmikrophysik. Daher wurden Messungen für das oben beschriebene Szenario mit bis zu 1024 Prozessen durchgeführt. In diesem Fall beträgt die Größe einer Partition nur noch 2×2×48 Gitterzellen. Die Laufzeiten auf einer SGI Altix 4700 (siehe Abschnitt 5.2 für eine kurze Systembeschreibung) von 16 bis 1024 Prozesse sind in Abbildung 2.4(a) dargestellt. Die Messung erfolgte über 30 min Simulationszeit ohne die Ausgabe von Ergebnissen in Dateien. Die Zeit für die Initialisierung des Modells ist nicht enthalten. Deutlich zu erkennen ist die starke Abnahme der parallelen Effizienz (bezogen auf 16 Prozesse). Bei 1024 Prozessen beträgt die Effizienz ca. 36 %.



(a) Laufzeit **(b)** Analyse einzelner Komponenten

Abbildung 2.4: Resultate der Skalierbarkeitsmessungen (*strong scaling*) von COSMO-SPECS und Analyse einzelner Komponenten (SGI Altix 4700).

In Abbildung 2.4(b) sind die Kosten (Laufzeit × Anzahl Prozesse) einzelner Teile des Modellsystems dargestellt. Im Idealfall, d. h. bei einem perfekt skalierenden Programm, wären die Kosten für jede Prozessanzahl identisch. Bei COSMO-SPECS ist ein Zuwachs in allen Teilen des Modellsystems zu beobachten. Am deutlichsten ist der Zuwachs beim Austausch der Hydrometeorspektren zwischen Nachbarprozessen. Dieser ist unterteilt in die Wartezeit durch Lastimbalance sowie der Zeit, die nach der Synchronisation in der eigentlichen Kommunikation verbracht wird. In Tabelle 2.1 sind die gemessenen Zeiten bei 16 und 1024 Prozessen gegenübergestellt. Mit diesen Messungen wird auch deutlich, wie stark die Berechnungszeit von der spektralen Mikrophysik dominiert wird.

Der Zuwachs in der Wartezeit ist durch eine steigende Lastimbalance zu erklären. Diese entsteht grundsätzlich durch die starke Abhängigkeit der Laufzeit von SPECS von den aktuellen Werten der Zustandsvariablen der Mikrophysik, insbesondere der Hydrometeorspektren. Auf das Problem der Lastimbalance wird im folgenden Abschnitt detaillierter eingegangen. Die Zeit für den eigentlichen Nachrichtentransfer steigt vor allem aufgrund der Zunahme des Verhältnisses von Kommunikation zu Berechnung. Während der Berechnungsaufwand je Prozess beim *strong scaling* mit steigender Prozessanzahl P um P^{-1} abnimmt, reduziert sich der Kommunikationsaufwand in der zweidimensionalen Partitionierung nur um $P^{-1/2}$ (siehe auch Abschnitt 3.1.1). Zudem verwendet COSMO-SPECS ein ungünstiges Kommunikationsschema: Für jede der 66 Größenklassen wird eine Service-Routine von COSMO für den Randaustausch gerufen, d. h. es werden 66 (verhältnismäßig kleine) Nachrichten zu jeder der vier benachbarten Partitionen gesendet. Das Versenden von vierdimensionalen Feldern, wie die Größenspektren von SPECS, ist in COSMO nicht vorgesehen. Eine Bündelung in eine einzige Nachricht mithilfe einer modifizierten Kommunikationsroutine wäre jedoch mit relativ geringem Aufwand durchführbar.

Die Zunahme der Laufzeit für COSMO und die Modifikation der Windfelder ist durch eine starke Erhöhung des Kommunikationsanteils zu erklären. Des Weiteren werden Berechnungen an Flächen zwischen Gitterzellen an den Partitionsgrenzen doppelt durchgeführt, wodurch sich der Rechenaufwand dafür bei 1024 Prozessen gegenüber 16 Prozessen in etwa verdoppelt. Ein

	16 Prozesse		1024 Prozesse		Faktor
	absolut	relativ	absolut	relativ	(absolut)
Austausch Hydrometeore	5 015 s	5,1 %	69 448 s	25,5 %	13,8
Synchronisation	7 148 s	7,3 %	53 197 s	19,6 %	7,4
COSMO	243 s	0,2 %	7 506 s	2,8 %	30,9
Korrektur Windfeld	3 504 s	3,6 %	14 141 s	5,2 %	4,0
Advektion Hydrometeore	8 445 s	8,7 %	28 600 s	10,5 %	3,4
SPECS	73 135 s	75,0 %	98 939 s	36,4 %	1,4
Gesamt	97 490 s	100,0 %	271 831 s	100,0 %	2,8

Tabelle 2.1: Vergleich der gemessenen Laufzeiten (über alle Prozesse aufsummiert) von Komponenten des Modellsystems COSMO-SPECS bei 16 und 1024 Prozessen.

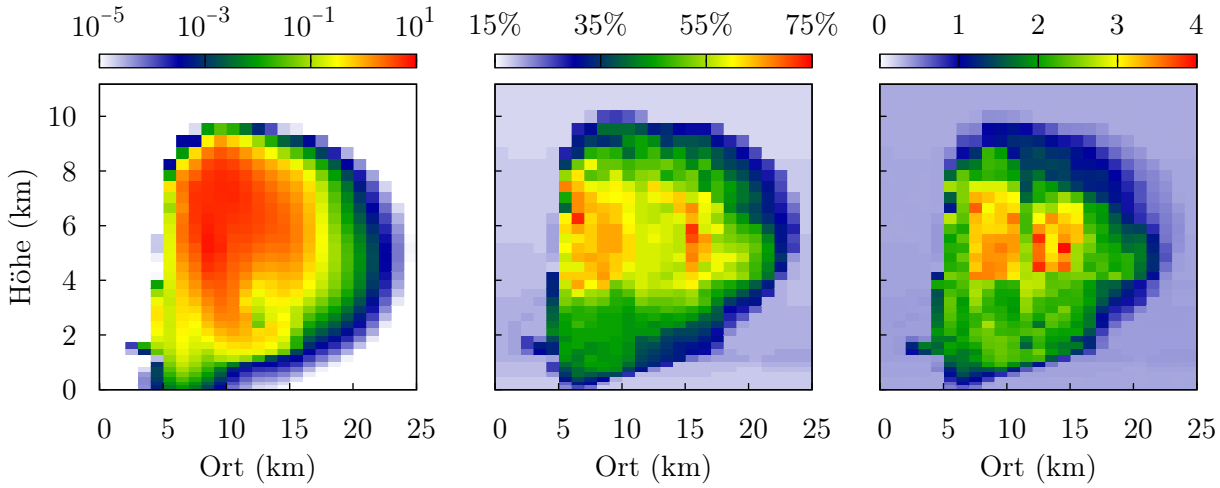
weiterer Grund für die sehr geringe Skalierbarkeit des COSMO-Modells ist zudem auf den gemeinsamen Zugriff hunderter Prozesse auf denselben Programmtext zurückzuführen, was eine Besonderheit des von vielen Prozessoren gemeinsam genutzten Speichers der SGI Altix 4700 darstellt.[4]

Bei der Advektion der Hydrometeore und der Berechnung der Wolkenmikrophysik in SPECS findet keine Kommunikation statt. Der Laufzeiterhöhung liegen hier Cache-Effekte zugrunde. Bei steigender Prozessanzahl erhöht sich der Anteil von Geisterzellen an den Datenstrukturen der Modellvariablen. Für Zugriffe auf Datenfelder, deren Größe den Cache übersteigt, reduziert sich somit die Chance für Cachetreffer. Durch ihre Größe trifft dies insbesondere auf die Hydrometeorspektren zu. Bei der Advektionsberechnung tritt zusätzlich die oben beschriebene Problematik der doppelten Berechnung an Partitionsgrenzen auf.

Die Ursachen für die ungenügende Skalierbarkeit von COSMO-SPECS lassen sich in vier Kategorien zusammenfassen:

- Lastimbalance: Das spektrale Wolkenmikrophysikmodell SPECS, welches die tatsächliche Rechenzeit des Modellsystems klar dominiert, löst starke Lastimbalancen aus. Diese führen zu hohen Wartezeiten in synchronisierenden Kommunikationsmustern und -operationen.
- Kommunikation: Das Verhältnis des Kommunikationsaufwands an den Partitionsrändern zu den Berechnungen innerhalb der Partitionen wird mit steigender Prozessanzahl ungünstiger, was durch die zweidimensionale Dekomposition noch verstärkt wird.
- Datenstrukturen: Bei hoher Parallelität, und somit sehr kleinen Partitionen, wird ein Großteil des Speichers von Geisterzellen eingenommen. Daraus resultiert bei einem Teil der Algorithmen eine geringere Rate an Cachetreffern.
- Algorithmen: Algorithmen, welche Berechnungen an den Flächen der Gitterzellen durchführen, resultieren mit zunehmender Parallelität in höherem Gesamtaufwand. Grund ist die Verdoppelung der Flächen an den Partitionsgrenzen.

[4] Eine detaillierte Analyse zeigte, dass hier der gemeinsame Lesezugriff auf Speicherseiten des Programmtextes die Skalierbarkeit einzelner Programmabschnitte begrenzt, da der Programmtext von allen Prozessen innerhalb einer *shared-memory*-Partition der SGI Altix 4700 gemeinsam genutzt wird. Das COSMO-Modell erleidet als einzige Programmphase einen spürbaren Leistungseinbruch, da es einen vergleichsweise umfangreichen Programmcode aufweist, so dass häufiges Nachladen in den Befehlscache erfolgen muss.



(a) Konzentration von Hydrometeoren (g/kg)

(b) Relative Belegung der Größenklassen

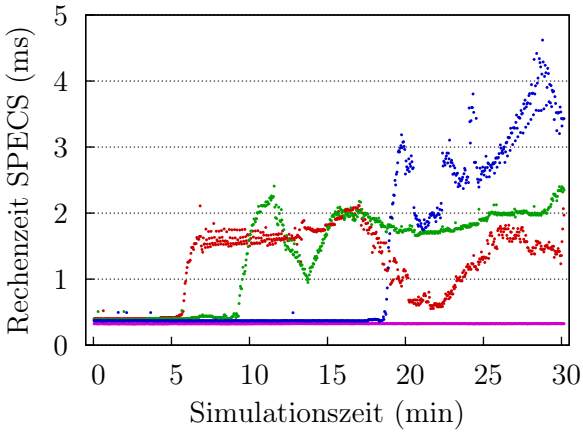
(c) SPECS Rechenzeit je Δt^{mp} (ms)

Abbildung 2.5: Gegenüberstellung der Konzentration von Hydrometeoren in der Luft (Summe über die Masse flüssiger und gefrorener Partikel), der Belegung der Größenklassen beider Partikelkategorien und der Rechenzeit von SPECS in einem Ausschnitt einer vertikalen Schnittfläche nach 30 min Simulationszeit.

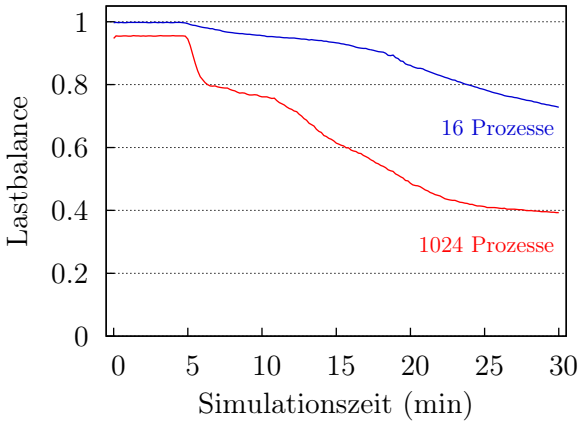
2.6.3 Analyse der Lastbalance

Der hohe Effizienzverlust durch Wartezeiten beim Austausch der Hydrometeorvariablen ist auf Lastimbalancen in dem spektralen Mikrophysikmodell SPECS zurückzuführen. Eine Ursache ist, dass die Modellgleichungen im Allgemeinen nur für Größenklassen gelöst werden müssen, welche Hydrometeore enthalten. Folglich ist der Rechenaufwand von SPECS von der Anzahl belegter Größenklassen für flüssige und gefrorene Partikel abhängig. Zudem wird das Gleichgewicht aus Wasserdampf und kondensiertem Wasser mit einer iterativen Methode bestimmt. Der Aufwand für die Berechnung von Kondensation und Verdunstung flüssiger Partikel sowie Deposition und Sublimation gefrorener Partikel ist somit linear abhängig von der Zahl belegter Größenklassen multipliziert mit der Anzahl der Iterationen. Die Iterationsanzahl erhöht sich mit zunehmendem Umsatz an Wasserdampf, wodurch der Rechenaufwand bei untersättigten Verhältnissen am geringsten ist. Der Aufwand der Algorithmen zur Modellierung des Gefrierens und Schmelzens von Hydrometeoren ist ebenfalls linear von der Zahl belegter Größenklassen abhängig. Dagegen kann die Kollision von Partikeln nur mit quadratischem Aufwand berechnet werden. Zusätzlich spielt die Temperatur eine Rolle für den Rechenaufwand, da Eisprozesse nur unter dem Gefrierpunkt berechnet werden.

Die Abhängigkeit des Rechenaufwands von SPECS von der Hydrometeorkonzentration wird in Abbildung 2.5 verdeutlicht. Dargestellt ist ein Schnitt durch die Kumuluswolke, welche in dem oben beschriebenen Szenario mit COSMO-SPECS simuliert wurde. Die Rechenzeit wurde auf der SGI Altix 4700 mit Intel Itanium 2 9040 Doppelkernprozessoren (1,6 GHz) gemessen. Deutlich zu erkennen ist die starke Variabilität der Rechenzeit. Die minimale und maximale Rechenzeit für einen kleinen Zeitschritt der Mikrophysik Δt^{mp} in dem dargestellten Ausschnitt beträgt 0,33 ms und 3,96 ms. Zwischen der Massenkonzentration an Hydrometeoren und der Rechenzeit ist ein gewisser Zusammenhang erkennbar. Deutlicher ist die Abhängigkeit der Rechenzeit von der Zahl belegter Größenklassen. Der Aufwand der Algorithmen hängt jedoch auch in komplexer Weise von den konkreten Werten der prognostischen Variablen ab und kann daher nicht allein auf Basis dieser Größe bestimmt werden.



(a) Rechenzeit je $\Delta t^{mp} = 2{,}5\,\mathrm{s}$ von vier ausgewählten Gitterzellen



(b) Lastbalance von SPECS

Abbildung 2.6: Veränderung der Lastbalance von SPECS über die Simulationszeit.

Durch die Dynamik der Atmosphäre und der Wolkenprozesse verändert sich die räumliche Verteilung der Last des Mikrophysikmodells SPECS über die Simulationszeit. Die gemessene Rechenzeit für das Kumulus-Szenario ist in Abbildung 2.6(a) für vier Gitterzellen dargestellt. Deutlich zu erkennen ist eine teilweise starke Veränderung der Rechenlast in aufeinanderfolgenden Zeitschritten und der sehr große Bereich, über den die Last insgesamt variiert. Die räumliche und zeitlich variierende Rechenlast beeinflusst die Lastbalance der Partitionierung. Die Lastbalance kann durch folgende Gleichung definiert werden (siehe auch Abschnitt 3.1.1):

$$\text{Lastbalance} = \frac{\text{Durchschnittliche Rechenlast der Partitionen}}{\text{Maximale Rechenlast einer Partition}}$$

Sie gibt den Anteil der für Berechnungen genutzten Zeit von der gesamten Laufzeit (d. h. einschließlich Wartezeiten) über alle Prozesse an. In Abbildung 2.6(b) ist die gemessene Lastbalance für 16 und 1024 Prozesse über die 30 min Simulationszeit des Kumulus-Szenarios dargestellt. Dabei wurde die Rechenzeit aller Berechnungen innerhalb der kleinen Zeitschritte der Mikrophysik herangezogen, dass heißt für die Advektion der Hydrometeore und für SPECS. Deutlich zu erkennen ist ein Einbruch der Lastbalance nach ca. fünf Minuten, welcher den Beginn des Wolkenwachstums markiert. Im weiteren Verlauf sinkt die Lastbalance bis unter 0,75 bei 16 Prozessen und unter 0,4 bei 1024 Prozessen. Die Lastbalance nimmt mit größerer Prozessanzahl typischerweise ab, da bei kleineren Partitionen die Wahrscheinlichkeit steigt, dass einzelne Partitionen einen besonders hohen Anteil an Gitterzellen hoher Rechenlast beinhalten.

2.7 Aktuelle Defizite der hochskalierbaren Simulation von Wolkenprozessen

Im vorigen Abschnitt ist deutlich geworden, dass die ungenügende Lastbalance und der steigende Kommunikationsaufwand die stärksten Skalierbarkeitsengpässe für COSMO-SPECS darstellen. Diese zwei Defizite treffen nicht nur für dieses Modellsystem zu, sondern gelten allgemein bei der Nutzung spektraler Wolkenmikrophysik in atmosphärischen Modellen. Die Lösung dieser fundamentalen Probleme vor dem Hintergrund der Hochskalierbarkeit ist Motivation und Kern dieser Arbeit.

Die Lastimbalance resultiert aus der Abhängigkeit des Aufwands der spektralen Algorithmen von den aktuellen Werten der prognostischen Variablen, wie der Hydrometeorspektren, dem

Wasserdampfgehalt und der Temperatur. Durch die hohe Dynamik der Wolkenprozesse kann eine statische Partitionierung keine gleichmäßige Verteilung der Last gewährleisten. Auf Nachfrage berichten auch die Entwickler der Modelle Fast-SBM (A. Khain, pers. Komm., 7. Juli 2011) und DESCAM-3D (W. Wobrock, pers. Komm., 8. Juli 2011) von Lastimbalance durch die spektrale Mikrophysik.

Das zweite wichtige Problem ist der hohe Kommunikationsaufwand. Dieser entsteht grundsätzlich durch die Vielzahl mikrophysikalischer Variablen des spektralen Ansatzes, welche regelmäßig zwischen benachbarten Partitionen ausgetauscht werden müssen. Die typische zweidimensionale Partitionierung des dreidimensionalen Gitters verstärkt diesen Aufwand zusätzlich, da diese im Vergleich zu einer Partitionierung über alle drei Raumdimensionen ein ungünstigeres Verhältnis von Kommunikation zu Berechnung bietet. Insbesondere für kleine Partitionsgrößen, wie sie für eine Erhöhung der Abarbeitungsgeschwindigkeit nötig sind, ergibt sich dabei ein vergleichsweise hoher Kommunikationsaufwand an den Rändern der Partitionen.

Aus den ermittelten Defiziten lassen sich folgenden Anforderungen an ein Konzept zur effizienten und hochskalierbaren Simulation von Wolkenprozessen mittels detaillierter Modelle formulieren:

- Die räumlichen und zeitlichen Lastvariationen der Berechnungen des Mikrophysikmodells erfordern eine dynamische Lastbalancierung, welche die Partitionierung zur Laufzeit des Simulationsmodells regelmäßig hinsichtlich der Gleichverteilung der Rechenlast anpasst.
- Der Kommunikationsaufwand des Mikrophysikmodells soll durch eine geeignete Dekomposition in drei Dimensionen reduziert werden. Zudem sollte die Partitionierung mit einem Verfahren erstellt werden, welches auf die Reduktion der Oberfläche der Partitionen ausgerichtet ist.
- Da atmosphärische Simulationsmodelle üblicherweise weder über eine dynamische Lastbalancierung noch eine dreidimensionale Dekomposition verfügen, ist eine Trennung der Datenstrukturen und Partitionierungen von atmosphärischem Modell und Mikrophysikmodell erforderlich. Zum Datenaustausch zwischen den zwei separierten Modellen sind Methoden der Modellkopplung erforderlich.

Daher beschäftigt sich das folgende Kapitel mit dem Stand der Forschung dynamischer Lastbalancierungsverfahren und Methoden der Modellkopplung, insbesondere unter den Aspekten der Hochskalierbarkeit und der Anwendung in atmosphärischen Simulationsmodellen.

Eine weitere Anforderung für eine hohe Skalierbarkeit stellt der effiziente Umgang mit den Ausgabedaten dar, welche durch die spektrale Repräsentation der Mikrophysik wesentlich anwachsen. Die Problematik der parallelen Ein-/Ausgabe gilt jedoch durch immer feiner werdende Gitterauflösungen und steigende Parallelität auch allgemein bei vielen Anwendungen atmosphärischer Modelle [122]. Diese Thematik bedarf gesonderter Betrachtung und ist nicht Gegenstand dieser Arbeit.

3 Lastbalancierung und Kopplung paralleler Simulationsmodelle

Den Schwerpunkt der Arbeit stellt die Entwicklung und Analyse einer dynamischen Lastbalancierung in einem gekoppelten atmosphärischen Modellsystem dar. Daher werden in diesem Kapitel der Stand der Forschung zu dynamischen Lastbalancierungsverfahren und zur Kopplung paralleler Simulationsmodelle vorgestellt. Insbesondere wird auf den Einsatz dieser Methoden in atmosphärischen Simulationsmodellen und auf Aspekte der Skalierbarkeit eingegangen.

3.1 Dynamische Lastbalancierung

Viele orts- und zeitabhängige Probleme der Strömungsmechanik werden mithilfe partieller Differenzialgleichungen (PDEs) numerisch gelöst. Auch atmosphärische Simulationsmodelle gehören in diese Kategorie. Bei expliziten numerischen Verfahren zur näherungsweisen Lösung von PDEs werden zur Auswertung der Gleichungen an einem Gitterpunkt nur Informationen von benachbarten Punkten im räumlichen Gitter benötigt [27, Seite 623]. Das konkrete Zugriffsmuster, also welche und wie weit entfernte Nachbarpunkte zur Aktualisierung benötigt werden, wird als *Stencil* bezeichnet und ist abhängig von der numerischen Approximation der Differenzenquotienten. Durch die räumliche Nähe der Punkte des Stencils sind diese Verfahren sehr gut zur Parallelisierung geeignet. Dabei wird das Gitter in einzelne *Partitionen* zerlegt. Die Partitionen werden den Prozessen zur Bearbeitung zugewiesen, so dass jede Partition genau einem Prozess zugeordnet ist. Die gesamte Aufteilung des Gitters wird als *Partitionierung* bezeichnet. Um auch am Rand der Partitionen auf benachbarte Gitterzellen (gemäß dem Stencil) zugreifen zu können werden sogenannte *Geisterzellen* (auch *ghost*, *guard* oder *halo grid points*) genutzt [27, Seite 77]. Bevor ein Zeitschritt berechnet werden kann, müssen die Inhalte dieser Geisterzellen aktualisiert werden. Dies erfordert Synchronisation und Datenaustausch zwischen auf dem Gitter benachbarten Prozessen. Die Berechnung selbst geschieht auf den Partitionen unabhängig voneinander. Die grundsätzliche Struktur einer Iteration zur parallelen Lösung von PDEs mittels expliziter Verfahren besteht also immer aus den zwei abwechselnden Phasen Synchronisation/Kommunikation und Berechnung. Aus dieser Struktur wird sofort ersichtlich, dass die Partitionierung einen hohen Einfluss auf die Effizienz eines parallelen Programms hat. Zum einen beeinflusst sie die Rechenzeit der Prozesse bis zur nächsten Synchronisationsphase und somit den Anteil ungewünschter Wartezeiten an der Gesamtlaufzeit. Zum anderen werden durch die Partitionierung Kommunikationsmuster und -aufwand für den Austausch der Geisterzellen festgelegt.

3.1.1 Anforderungen an Partitionierungsverfahren

Statische Partitionierung

Um höchstmögliche Leistung auf Parallelrechnern zu erzielen, wird eine Reihe von allgemeinen Anforderungen an Methoden zur Ermittlung einer Partitionierung (*Partitionierungsverfahren*) gestellt [14, 107]. Aus dem grundsätzlichen Ziel der Beschleunigung der Berechnung durch Parallelisierung ergibt sich als erste Anforderung ganz offensichtlich:

1. Lastbalance: *Die Rechenlast soll so gleichmäßig wie möglich über den Partitionen verteilt werden.*

Diese Anforderung geht von der Annahme aus, dass die Partitionen jeweils exklusiv auf gleich schnellen Prozessorkernen berechnet werden. In der Literatur finden sich verschiedene Definitionen für die Lastbalance bzw. Lastimbalance [75, 81, 95, 114, 124]. Basierend auf der Definition von Pilkington und Baden [81] gilt für die Lastbalance Λ:

$$\text{Lastbalance} = \frac{\text{Durchschnittliche Rechenlast der Partitionen}}{\text{Maximale Rechenlast einer Partition}} \tag{3.1}$$

Im optimalen Fall beträgt die Lastbalance 1 und im schlechtesten Fall (alle Gitterpunkte befinden sich auf einer Partition) den Kehrwert der Anzahl an Partitionen. Der Wert der Lastbalance befindet sich somit immer im Intervall $(0, 1]$. Der Kehrwert der Lastbalance Λ^{-1} entspricht den Definitionen für Lastimbalance von Schloegel et al. [95] sowie Walshaw und Cross [114].

Eine weitere Anforderung resultiert aus dem durch die Aufteilung des Gitters nötigen Austausch von Geisterzellen. Liegen benachbarte Gitterzellen auf unterschiedlichen Prozessen, so müssen Daten zwischen diesen Prozessen ausgetauscht werden. Da dies im Allgemeinen weitaus mehr Zeit als ein lokaler Kopiervorgang in Anspruch nimmt, lässt sich wie folgt formulieren:

2. Anwendungskommunikation: *Der Aufwand für Interprozesskommunikation für den Austausch von Geisterzellen in dem partitionierten Gitter soll minimiert werden.*

Bei der Reduktion der Anwendungskommunikation ist von hoher Bedeutung, ob alle Dimensionen des räumlichen Gitters zur Zerlegung genutzt werden. Für eine optimale dreidimensionale Partitionierung eines kubischen Gitters lässt sich die Fläche A^{3D}_{comm}, über welche Daten zwischen allen Partitionen ausgetauscht werden, wie folgt berechnen [3]:

$$A^{3D}_{comm}(P) = 6\,P\left(\frac{G}{P}\right)^{\frac{2}{3}} = 6\,G^{\frac{2}{3}}\,P^{\frac{1}{3}} \sim P^{\frac{1}{3}} \tag{3.2}$$

Dabei steht G für die Anzahl der Gitterzellen und P für die Anzahl der Partitionen. Der Einfachheit halber werden periodische Randbedingungen in allen drei Dimensionen angenommen. Dieser Wert stellt die minimale theoretisch erreichbare Fläche dar. Analog lässt sich für eine ideale zweidimensionale Partitionierung formulieren:

$$A^{2D}_{comm}(P) = 4\,G^{\frac{2}{3}}\,P^{\frac{1}{2}} \sim P^{\frac{1}{2}} \tag{3.3}$$

Es wird deutlich, dass eine dreidimensionale Dekomposition eine wesentlich günstigere Skalierbarkeit ausweist, obwohl auch dort der Kommunikationsaufwand insgesamt zunimmt.

Es ist jedoch zu beachten, dass die Anforderung über die Minimierung des Gesamtkommunikationsaufwands (d. h. die Anzahl der zu übertragenden Bytes) hinausgeht. Es muss auch auf eine balancierte Kommunikation geachtet werden. Das bedeutet, dass die Minimierung des maximalen Kommunikationsaufwands unter den Prozessen wichtiger sein kann, als die Minimierung des Gesamtkommunikationsaufwands [107]. Des Weiteren spielt die Struktur des Netzwerks eine wichtige Rolle. Ein optimales Partitionierungsverfahren berücksichtigt die inhomogenen Latenzen und Bandbreiten zwischen den Prozessen auf Parallelrechnern [108]. Zum Beispiel sollten Prozesse, welche auf demselben Knoten laufen, benachbarte Partitionen erhalten, um beim Austausch der Geisterzellen von der typischerweise geringeren Latenz und höheren Bandbreite innerhalb des Knotens profitieren zu können.

Dynamische Lastbalancierung

Für die Berechnung von Problemen, deren Rechenlast nicht ortsabhängig zur Laufzeit schwankt, sind die zwei bisher betrachteten Anforderungen an Partitionierungsverfahren ausreichend. Die *statische Partitionierung* wird vor Beginn der Berechnungen erzeugt und ändert sich zur Laufzeit nicht. Bei atmosphärischen Simulationsmodellen ist dieses Vorgehen typisch [122]. Es gibt jedoch auch Simulationsmodelle, bei denen die Rechenlast ortsabhängig zur Laufzeit so stark schwankt, dass mit einer statischen Partitionierung keine effiziente Ausführung mehr gegeben ist. Die Gründe lassen sich zwei Kategorien zuordnen: adaptive Diskretisierung in Zeit [121] oder Raum (z. B. *adaptive mesh refinement* [25]) sowie ein zustandsabhängig variierender Rechenaufwand des Simulationsmodells je Gitterpunkt. Die Motivation, solche Algorithmen anzuwenden, liegt dabei oft in der Einsparung von Rechenaufwand bzw. bei der Konzentration der vorhanden Rechenkapazität auf zeitliche und räumliche Bereiche, welche von hohem Interesse sind oder welche unbedingt eine höhere Auflösung benötigen, um eine hinreichend exakte Lösung zu erhalten. Bei parallelen Simulationsprogrammen ergibt sich hierbei das Problem, dass der Rechenaufwand der Partitionen zur Laufzeit inhomogen schwankt. Dies führt dazu, dass eine anfänglich ausreichend balancierte Partitionierung nach einer bestimmten Anzahl von Iterationen nicht mehr der Anforderung nach Balance genügt. Daher ist es notwendig, die Partitionierung während der Berechnung kontinuierlich anzupassen, d. h. das Gitter zwischen den Prozessen umzuverteilen. In solchen Fällen wird von *dynamischer Lastbalancierung* [107] anstelle von Partitionierung gesprochen. In der Programmstruktur wird dazu typischerweise eine dritte Phase der *Repartitionierung* zu den zwei bereits genannten Phasen Synchronisation/Kommunikation und Berechnung hinzugefügt. Eine Repartitionierung muss dabei nicht jede Iteration durchgeführt werden, sondern kann entweder nur in festen größeren Intervallen stattfinden oder es wird nach Messung der Lastbalance dynamisch entschieden, ob eine Repartitionierung notwendig ist. Sie ist nur dann hinsichtlich der Laufzeitreduktion lohnenswert, wenn die durch besser balancierte Berechnungen eingesparte Zeit die zusätzliche Zeit der Berechnung einer neuen Partitionierung und der darauf folgenden Umverteilung der Daten zwischen den Prozessen mehr als aufwiegt. Daraus resultieren zwei weitere Anforderungen an Partitionierungsverfahren für dynamische Lastbalancierung (*Repartitionierungsverfahren*), welche zusätzlich zu den zwei oben genannten gelten.

3. Migration: *Der Aufwand der Interprozesskommunikation für die Migration der Daten zwischen aufeinanderfolgenden Partitionierungen soll minimiert werden.*

Hier gelten die gleichen Erläuterungen wie bei Anforderung 2 (Anwendungskommunikation). Zum einen kann die Reduktion des maximalen Kommunikationsaufwands zur Migration unter den Prozessen wichtiger sein als die Minimierung des gesamten Kommunikationsaufwands [96]. Zum anderen hat auch hier die Netzwerkstruktur einen Einfluss auf die Zeit, welche für die Migration der Daten des Gitters benötigt wird.

4. Partitionsberechnung: *Der Aufwand für die Berechnung einer neuen Partitionierung soll minimiert werden.*

Diese Anforderung betrifft den Algorithmus des Repartitionierungsverfahrens selbst, und nicht dessen Resultat wie in den vorigen 3 Anforderungen. Eine Beschleunigung kann zum Beispiel durch Parallelisierung, Reduktion der Ziele bei den anderen Anforderungen oder durch Anwendung eines grundsätzlich anderen (und effizienteren) Repartitionierungsverfahrens erfolgen. Auf Eigenschaften verschiedener Verfahren wird im folgenden Abschnitt eingegangen.

Die Bedeutung der zwei zusätzlichen Anforderungen an Repartitionierungsverfahren im Verhältnis zu den zwei vorigen Anforderungen ist abhängig von dem Anteil, den die Repartitionierung an der Gesamtlaufzeit des Simulationsprogramms hat. Je höher dieser ist, desto wichtiger

sind Optimierungen hinsichtlich dieser zusätzlichen Anforderungen, wobei unter Umständen beispielsweise eine schlechtere Lastbalance (Anforderung 1) in Kauf genommen wird [107]. Dies ist insbesondere bei hochparallelen Anwendungen zu beachten, da das Repartitionierungsverfahren – wenn es überhaupt parallelisiert ist – im Allgemeinen ein anderes Skalierungsverhalten als die Anwendung selbst aufweist.

Um eine Repartitionierung durchzuführen, muss die Rechenlast der Gitterzellen (auch *Gewicht*) in den künftigen Iterationen bekannt sein. Ist es nicht möglich den zukünftigen Aufwand exakt zu bestimmen, wird üblicherweise eine Abschätzung aus den vergangenen Iterationen vorgenommen [118]. Die Rechenlast kann je nach Ursache der Lastschwankungen unterschiedlich bestimmt werden. Für PDE-Lösungsverfahren mit adaptiven Zeitschritten kann die Anzahl der Zeitschritte je Gitterzelle herangezogen werden [121]. Variiert der Rechenaufwand von komplexen Algorithmen zustandsabhängig je Gitterzelle gibt es prinzipiell zwei Methoden der Ermittlung des Aufwands [79, 118]. Zum einen kann eine Messung erfolgen (z. B. Ausführungszeit oder Anzahl arithmetischer Operationen), was kein Wissen über die Algorithmen des Simulationsmodells voraussetzt, jedoch unter Umständen anfällig für systembedingte Schwankungen ist. Zum anderen können aus dem Algorithmus geeignete Indikatoren für das Gewicht gewonnen werden, wie zum Beispiel die Anzahl der Partikel je Gitterzelle beim *particle-in-cell*-Verfahren [120]. Bei PDE-Lösern mit räumlich adaptiven Gittern ändert sich zur Laufzeit die Anzahl der Gitterzellen, so dass hier ebenfalls eine Repartitionierung durchgeführt werden muss, selbst wenn die Last je Gitterzelle nicht variiert. In diesem Fall genügt es, ohne Bestimmung der Gewichte, die Anzahl der Gitterpunkte je Partition zu balancieren [13].

3.1.2 Überblick zu Partitionierungsverfahren

Das Partitionierungsverfahren ermittelt die Verteilung der Gitterpunkte (d. h. des Rechenaufwands) auf die Partitionen gemäß den oben eingeführten Anforderungen. Als Eingabe erhält das Verfahren die Nachbarschaftsbeziehungen oder Koordinaten der Gitterpunkte (im einfachsten Falle ein Rechteckgitter) sowie die Anzahl der zu erzeugenden Partitionen. Die Ausgabe ist die Zuordnung von Gitterpunkten zu Prozessen. Optional sind weitere Eingaben möglich. Aus praktischen Gründen der Implementierung und Performance der Datenstrukturen werden – insbesondere bei strukturierten Gittern – mehrere Gitterpunkte zu Blöcken zusammengefasst. In diesem Fall werden statt einzelner Gitterpunkte Blöcke auf die Partitionen verteilt. Daher wird im Folgenden allgemein von *Objekten* als minimale zwischen den Partitionen zu verteilende Einheit gesprochen. Die Objekte und die Nachbarschaftsbeziehungen können mit Gewichten (auch jeweils mehreren Gewichten) versehen sein und dienen als Kriterium für Rechenaufwand der Objekte und Kommunikationsaufwand zwischen den Objekten. Um den Aufwand der Migration (Anforderung 3) explizit zu berücksichtigen, kann eine bereits bestehende Partitionierung Eingabe des Verfahrens sein. Zudem können Toleranzen bzw. Prioritäten hinsichtlich der genannten Anforderungen hinzukommen, da sie typischerweise nicht alle zugleich optimal erfüllt werden können. Algorithmen, welche hinsichtlich Balance (Anforderung 1) und Anwendungskommunikation (Anforderung 2) optimale Partitionierungen ermitteln, sind im Allgemeinen NP-vollständig [107]. Daher werden in der Praxis Heuristiken genutzt. Zu Vergleichen von Partitionierungsverfahren existiert viel Literatur, zum Beispiel [14, 40, 82, 96, 107].

Abbildung 3.1 zeigt eine Übersicht der wichtigsten Partitionierungsverfahren mit einer Einteilung nach Teresco et al. [107]. Es wird grundsätzlich zwischen geometrischen und graphenbasierten Verfahren unterschieden. Geometrische Verfahren beachten bei der Erstellung der Partitionierung nur die Koordinaten und Gewichte der Objekte. Die tatsächlichen Nachbarschaftsbeziehungen und deren Gewichte werden dabei vernachlässigt. Dabei wird eine balancierte Partitionierung erstellt, bei der Objekte mit geringer Distanz möglichst derselben Partition zugeordnet sind. Die

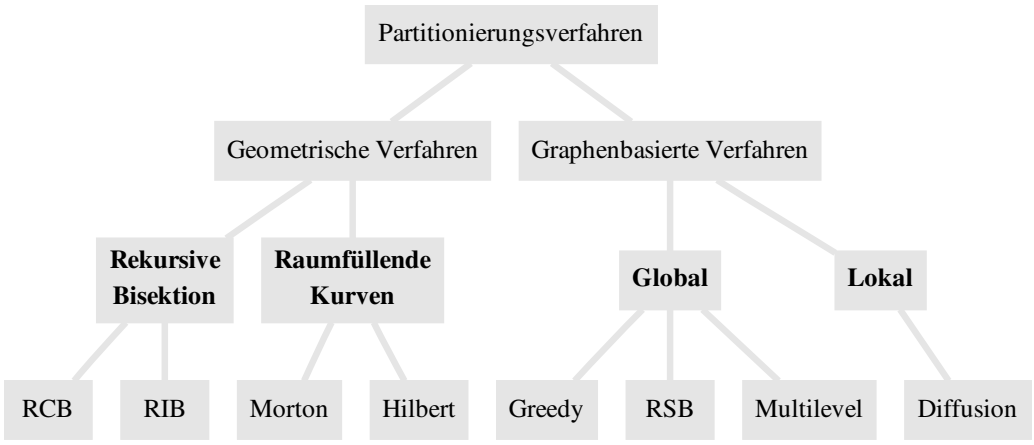


Abbildung 3.1: Übersicht zu Partitionierungsverfahren. Die fett geschriebenen Verfahren werden jeweils in einem Abschnitt näher behandelt.

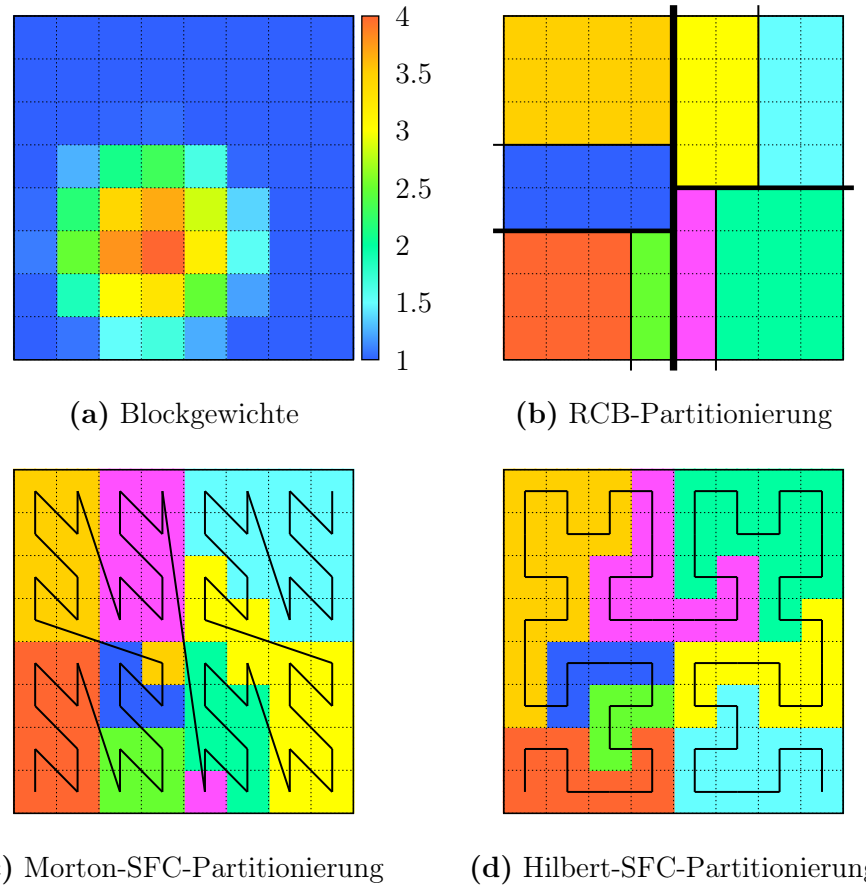
räumliche Nähe ist dabei im Allgemeinen eine gute Näherung für die Nachbarschaftsbeziehungen. Graphenbasierte Verfahren dagegen bilden das Partitionierungsproblem auf die Zerlegung eines gewichteten Graphen ab. Dabei entsprechen die Knoten den Objekten und die Kanten den Nachbarschaftsbeziehungen, welche auf diese Weise explizit berücksichtigt werden. Daher erzeugen graphenbasierte Verfahren im Allgemeinen Partitionierungen mit geringeren Kosten für die Anwendungskommunikation.

Rekursive Bisektionsverfahren

Rekursive Bisektionsverfahren basieren auf der Teilung des Gitters in zwei Teilgitter mit gleicher Gesamtrechenlast. Dieses Verfahren kann so oft rekursiv auf die Teilgitter angewendet werden, bis die benötigte Anzahl an Partitionen erreicht ist. Mit geringen Modifikationen kann die Einschränkung auf Zweierpotenzen als Partitionsanzahl aufgehoben werden [40]. Zwei Varianten rekursiver Bisektionsverfahren sind RCB (*recursive coordinate bisection*) und RIB (*recursive inertial bisection*). RCB [8] teilt das Gitter jeweils entlang der längsten Achse des Koordinatensystems mit einer geraden Schnittfläche, während RIB [84] eine Teilung orthogonal zur Hauptträgheitsachse des Gitters vornimmt. Für rechteckige Gitter sind RCB und RIB identisch. Bei (unstrukturierten) Gittern, welche eine komplexe Geometrie repräsentieren, bietet RIB den Vorteil, dass sich die Schnittachse an der Geometrie und Ausrichtung des Gitters orientiert. Beide Varianten sind sehr schnell in der Berechnung der Partitionierung und können einfach parallelisiert werden. Zur Repartitionierung adaptiver Gitter ist RIB allerdings wenig geeignet, da sich die Hauptträgheitsachse bei Veränderung des Gitters verschieben oder drehen kann [40] und folglich aufeinanderfolgende Partitionierungen große Unterschiede aufweisen. Ein Beispiel einer Partitionierung mittels RCB ist in Abbildung 3.2 (b) dargestellt.

Verfahren basierend auf raumfüllenden Kurven (space-filling curves)

Raumfüllende Kurven (*space-filling curves*, SFC) sind stetige Kurven, welche jeden Punkt eines mehrdimensionalen Gebiets durchlaufen [91, Seite 5]. Damit ermöglichen sie eine Abbildung der Punkte bzw. Objekte eines mehrdimensionalen Gitters auf eine Dimension, d. h. eine Linearisierung. Partitionierungsverfahren nutzen diese Vereinfachung des Partitionierungsproblems, indem sie die Linearisierung in so viele kontinuierliche Abschnitte zerlegen, wie Partitionen erzeugt werden sollen [107]. Damit die Partitionen im ursprünglichen mehrdimensionalen Gitter eine geringe Anwendungskommunikation aufweisen, werden Kurven mit hoher Lokalität benötigt. Lokalität



(a) Blockgewichte **(b)** RCB-Partitionierung

(c) Morton-SFC-Partitionierung **(d)** Hilbert-SFC-Partitionierung

Abbildung 3.2: Exemplarische Darstellung von Partitionierungen erstellt mit den geometrischen Verfahren RCB (*recursive coordinate bisection*), Morton SFC (*space-filling curve*) sowie Hilbert SFC für die gegebene Verteilung der Blockgewichte in einem regulären zweidimensionalen Gitter.

bedeutet, dass Punkte, die in der Linearisierung benachbart sind, auch im mehrdimensionalen Gebiet nah beieinander liegen. In Partitionierungsverfahren werden verbreitet Morton-Kurven (auch Z- oder N-Ordnung genannt) und Hilbert-Kurven eingesetzt. Durch ihre Selbstähnlichkeit lassen sich diese sehr einfach basierend auf einem Startmuster mithilfe einer Zustandstabelle generieren [44]. Das Startmuster wird entsprechend dem Verfeinerungsfaktor k der Kurve und der Zahl an Dimensionen n in k^n Segmente unterteilt (für Morton- und Hilbert-Kurven gilt $k = 2$, für Peano-Kurven $k = 3$). Diese Segmente werden solange rekursiv durch das (ggf. rotierte) Startmuster ersetzt, bis die benötigte Stufe l der Kurve erreicht ist. Im n-dimensionalen Raum durchläuft die Kurve so $(k^l)^n$ Punkte. Kurven mit $k = 2$ können auch durch Bit-Operationen generiert werden [3, 64], wobei die Komplexität beider Verfahren $O(l)$ ist. Die Bezeichnung „raumfüllend“ bezeichnet die Eigenschaft dieser Kurven, dass sie für $l \to \infty$ jeden Punkt im mehrdimensionalen Raum durchlaufen. Abbildung 3.2 veranschaulicht Partitionierungen basierend auf Morton- und Hilbert-Kurven.

Bei der Zerlegung der Kurve in kontinuierliche Abschnitte als Partitionen müssen die Anzahl der Objekte oder gegebenenfalls die Gewichte der Objekte balanciert werden (*eindimensionale Partitionierung*). Im ersten Fall ist das Problem trivial, da sich die Lösung sofort aus der Zahl der Objekte und Anzahl der Partitionen ergibt. Bei gewichteten Objekten kann man zwischen Heuristiken [76, 81] und exakten Algorithmen [82] unterscheiden. Das Ziel ist es, den sogenannten *bottleneck value*, die Summe der Gewichte der maximal belasteten Partition, zu minimieren. Bei Problemen mit stark variierenden Gewichten zeigt sich, dass Heuristiken die optimale Last-

balance teilweise weit verfehlen [82]. Parallele eindimensionale Partitionierungsalgorithmen, bei denen die Blockgewichte über alle Prozesse verteilt bleiben, wurden bisher nur für die Heuristiken entwickelt [76, 81], nicht jedoch für die exakten Verfahren.

Partitionierungsverfahren basierend auf raumfüllenden Kurven sind etwa gleich schnell und generieren hinsichtlich der Anwendungskommunikation nahezu gleich gute Partitionierungen wie rekursive Bisektionsverfahren [81, 107]. Dabei ist die Hilbert-Kurve der Morton-Kurve im Allgemeinen überlegen [44, 77]. Die Selbstähnlichkeit der Kurven erlaubt auch die Verwendung zur Partitionierung adaptiv verfeinerter Gitter. Auch für hochskalierende Anwendungen werden sie in diesem Bereich daher bevorzugt eingesetzt [3, 13]. Zudem liegen ihre Stärken in der Repartitionierung, da bei der Migration Daten typischerweise nur zwischen benachbarten Partitionen entlang der Kurve ausgetauscht werden müssen [81].

Globale graphenbasierte Verfahren

Durch die Betrachtung des Gitters als Graph können die Nachbarschaftsbeziehungen, die durch den Stencil des numerischen Verfahrens zwischen den Gitterpunkten definiert sind, explizit bei der Partitionierung berücksichtigt werden. Damit können globale graphenbasierte Partitionierungsverfahren das Gitter derart balanciert zerlegen, dass die Anwendungskommunikation minimiert wird. In diesem Zusammenhang spricht man üblicherweise vom sogenannten *edge cut* [107], welcher der Zahl durch Partitionsgrenzen geschnittener Kanten entspricht. Werden die Kanten des Graphen zusätzlich entsprechend der Datenmenge gewichtet, welche zwischen den Punkten (Knoten) ausgetauscht werden muss, kann die Optimierung der Kommunikation noch gezielter erfolgen.

Zu Graphenpartitionierung existieren verschiedene Heuristiken [107]. Bei dem sogenannten *greedy algorithm* [30] werden ausgehend von einem Startknoten benachbarte Knoten zur Partition hinzugefügt, bis die gewünschte Größe erreicht ist. Diese Prozedur wird wiederholt, bis alle Partitionen erzeugt wurden. Das Verfahren ist sehr schnell, jedoch wird die Anwendungskommunikation kaum besser optimiert als bei den geometrischen Verfahren und es ist ungeeignet zur Repartitionierung [107]. Ein weiteres Verfahren ist RSB (*recursive spectral bisection*). Hierbei wird der zweitgrößte Eigenvektor der *Laplace-Matrix* [100] des Graphen gebildet, der sogenannte *Fiedler-Vektor*. Die Knoten werden nach den Werten des Fiedler-Vektors sortiert und in zwei Hälften geteilt. Für die so entstandenen Teilgraphen wird diese Methode rekursiv wiederholt. RSB ist sehr gut in der Reduktion der Anwendungskommunikation, jedoch sehr aufwendig in der Berechnung. Etwas weniger Rechenaufwand benötigen *Multilevel-Verfahren*, indem sie den Graph vor dem eigentlichen Partitionieren durch Vereinen benachbarter Knoten vergröbern (*coarsening*). Zur Partitionierung des vereinfachten Graphen können graphenbasierte oder geometrische Verfahren genutzt werden [107]. Bei der anschließenden Verfeinerung (*uncoarsening*) wird die Partitionierung des vergröberten Graphen auf den ursprünglichen Graph abgebildet. Dabei werden lokale Optimierungen hinsichtlich Balance und Anwendungskommunikation vorgenommen [114]. Multilevel-Verfahren erstellen sehr hochwertige Partitionierungen. Es existieren auch parallelisierte Multilevel-Partitionierer [47, 114], jedoch skalieren diese bisher nicht auf mehrere tausend Prozesse [7, 41]. Zur Repartitionierung sowie als Optimierungsverfahren während der *uncoarsening*-Phase [47] ist das *Kernighan-Lin-Verfahren* und die Erweiterung *Fiduccia-Mettheyses* (KL/FM) verbreitet [96]. Hierbei werden, ausgehend von einer existierenden Partitionierung, Optimierungen hinsichtlich des *edge cut* durch den Austausch von Knoten zwischen den Partitionen erwirkt. Dabei können auch lokale Minima durch temporäres Anwachsen des *edge cut* überwunden werden. Des Weiteren existieren sogenannte *multi-constraint*-Partitionierungsverfahren, welche den Graph nach mehreren Kriterien balancieren [95]. Jedem Knoten wird dazu ein Vektor von Gewichten zugeordnet. *Multi-constraint*-Partitionierung ist vor allem für Multiphysik- und Multiphasen-Simulationen von Bedeutung.

Lokale graphenbasierte Verfahren

Lokale graphenbasierte Verfahren balancieren überlappende Teilgraphen, die sich zum Beispiel aus den direkten Nachbarschaftsbeziehungen der Partitionen oder der Netzwerkstruktur des Parallelrechners ergeben [20, 107]. Sie benötigen eine bereits existierende Partitionierung und sind somit ausschließlich zur Repartitionierung geeignet. In Anlehnung an den physikalischen Prozess werden solche Verfahren auch als (lokale) *Diffusionsverfahren* bezeichnet [22, 118]. Sie sind typischerweise in zwei Schritte unterteilt: Zuerst wird festgestellt, wie viel Last zwischen den Partitionen transferiert werden muss, um eine balancierte Partitionierung zu erhalten. Danach werden Knoten zur Migration innerhalb der Teilgraphen ausgewählt, wobei spezifisch nach den oben genannten Anforderungen optimiert werden kann. Diffusionsverfahren sind im Allgemeinen schneller als globale graphenbasierte Verfahren und können relativ einfach parallelisiert werden. Ein Vorteil für hochskalierende Anwendungen ist, dass keine explizite Synchronisation aller Prozesse notwendig ist. Globale Lastbalance wird jedoch unter Umständen erst nach mehreren Zyklen des Verfahrens erreicht [107].

Hierarchische Verfahren

Hierarchische Verfahren ordnen die Prozesse als Blattknoten eines Baumes an, so dass sie in disjunkte Gruppen unterteilt werden. Die Lastbalancierung erfolgt zwischen den Gruppen und innerhalb der Gruppen getrennt. Dabei können in jeder Ebene des Baumes unterschiedliche Partitionierungsverfahren eingesetzt werden. Somit ist dieser Ansatz kein Partitionierungsverfahren im bisher betrachteten Sinne, da er keine eigenständige Strategie zur Zuordnung der Objekte zu Partitionen beinhaltet. Mit hierarchischen Verfahren kann zum einen die Lastbalancierung an die hierarchische Kommunikationsstruktur aktueller Hochleistungsrechner angepasst werden [108]. Dabei wird zwischen den Rechnerknoten die Anwendungskommunikation mit einem aufwendigen Partitionierungsverfahren minimiert (z. B. mit einem globalen graphenbasierten Verfahren) und innerhalb der Knoten ein schnelleres Verfahren genutzt. Zum anderen weisen hierarchische Verfahren eine bessere Skalierbarkeit in Bezug auf Rechenaufwand und Speicherbedarf auf als globale Partitionierungsverfahren und stellen im Gegensatz zu lokalen Verfahren ebenfalls globale Lastbalance her [124]. Damit kann dem mit zunehmendem Grad an Parallelität wachsenden Speicherbedarf eines globalen Partitionierungsverfahrens entgegengewirkt werden.

3.1.3 Software zur dynamischen Lastbalancierung

Es existieren prinzipiell zwei Arten von Software zur dynamischen Lastbalancierung: Implementierungen von Partitionierungsalgorithmen sowie Parallelisierungsframeworks, welche zusätzlich auch die Migration der Objekte und eventuell weitere Funktionalitäten anbieten. Eine tabellarische Übersicht von Partitionierungsalgorithmen in verfügbaren Bibliotheken wird in [96] gegeben.

Die Partitionierungsbibliothek ParMetis [95] enthält parallele Multilevel-Verfahren zur Partitionierung und Repartitionierung von Graphen. Zur Repartitionierung nutzt ParMetis sowohl Diffusionsverfahren als auch eine Abbildung einer komplett neuen Partitionierung auf die vorherige. Diese beiden Methoden werden auf den vergröberten Graphen des Multilevel-Verfahrens angewendet. Danach wird die hinsichtlich vom Nutzer gewählter Parameter (stärkere Reduktion von Anwendungskommunikation oder Migrationskosten) günstigere Zerlegung gewählt und verfeinert. ParMetis ermöglicht auch *multi-constraint*-Partitionierung. Weitere Bibliotheken zur parallelen Graphenpartitionierung mittels Multilevel-Verfahren sind DibaP [71], Jostle [115], KaPPa [41] und PT-Scotch [17]. Von diesen implementieren jedoch nur DibaP und Jostle einen Repartitionierungsalgorithmus.

Zoltan [24] ist ein Parallelisierungsframework, welches unter anderem verschiedene Lastbalancierungsverfahren und Funktionen zur Migration der Objekte anbietet. Dabei kann Zoltan zum einen auf Partitionierungsbibliotheken wie ParMetis und Scotch zugreifen. Zum anderen sind auch eigene Implementierungen geometrischer Verfahren sowie ein Hypergraph-Partitionierungsalgorithmus enthalten. Bei Hypergraphen verbinden die Kanten mehr als zwei Knoten zugleich, womit eine exakte Modellierung der Anwendungskommunikation und Migrationskosten möglich ist [14]. Zudem können in Zoltan die implementierten Partitionierungsverfahren zu einem hierarchischen Verfahren kombiniert werden. Charm++ [46] ist eine Spracherweiterung für C++ zur parallelen Programmierung, welche die Zerlegung der Berechnungen und Daten durch sogenannte *Chares* abstrahiert. Chares sind Objekte, welche von Charm++ transparent zwischen den Prozessen migriert werden können. Zur (Re)Partitionierung stehen eine Vielzahl von Verfahren zur Verfügung, unter anderem globale Graphenpartitionierung, lokale Verfahren sowie hierarchische Verfahren. Raumfüllende Kurven werden in vielen Frameworks für paralleles AMR (*adaptive mesh refinement*) zur Partitionierung genutzt, zum Beispiel in ALPS [13], Chombo [111], GrACE [15], PARAMESH [69] und Racoon [4]. Dabei wird die Selbstähnlichkeit der Kurven ausgenutzt, indem in stärker verfeinerten Gitterbereichen lokal eine höhere Stufe der jeweiligen Kurve zur Linearisierung verwendet wird.

3.1.4 Dynamische Lastbalancierung in atmosphärischen Modellen

In atmosphärischen Simulationsmodellen werden Schwankungen der Rechenlast an den Gitterpunkten vor allem durch physikalische Parametrisierungen ausgelöst. Physikalische Prozesse wie Strahlung und Wolkenbildung treten räumlich und zeitlich nicht einheitlich verteilt auf, so dass Lastimbalancen entstehen können. Xue et al. [122] führen die Parametrisierungen von Kondensation, Strahlung und Turbulenz als Beispiele an. Zudem geben sie an, dass Gewitter die Rechenzeit einer Partition um etwa 20–30 % erhöhen. Schättler und Krenzien [94] unterscheiden zwischen statischen und dynamischen Imbalancen in einem Wettermodell. Statische Imbalancen werden durch die feste Zuordnung von Gitterpunkten zu Land und Meer hervorgerufen, vor allem durch die Beschreibung von Bodenprozessen. Als Ursachen dynamischer Imbalancen geben die Autoren Parametrisierungen für Strahlung, Niederschlag und Konvektion an. Für Konvektion werden Abweichungen im Rechenaufwand von 30 % angeführt. Zudem machen die Autoren mit Messungen deutlich, dass die Effizienzverluste durch Lastimbalancen mit steigender Prozessoranzahl zunehmen. Michalakes und Nanjundiah [75] analysieren Lastimbalancen in einem globalen Klimamodell. Als Hauptursache identifizieren sie die Lastschwankung der Strahlungsberechnung, welche durch den Tagesgang hervorgerufen wird. Durch den Zusammenhang mit dem Tageslicht ist dieses Muster von Imbalancen leicht vorherzusagen. Des Weiteren führen die Autoren die Konvektion als Grund für nicht vorhersagbare Schwankungen in der Rechenlast an. Für das globale Klimamodell ECHAM6 geben Adamidis et al. [1] ebenfalls die Strahlungsberechnung als Hauptverursacher der Lastimbalance an. Trotz der beobachteten Imbalancen verwenden die meisten atmosphärischen Simulationsmodelle keine dynamische Lastbalancierung, da der Aufwand des Partitionierungsverfahrens und des Umverteilens der Daten im Allgemeinen für zu groß angesehen wird [92, 122]. Im Folgenden werden Beispiele der Lastbalancierung in atmosphärischen Simulationsmodellen vorgestellt.

Foster und Toonen [32] vergleichen Lastbalancierungsmethoden für die Strahlungsberechnung in dem globalen Klimamodell PCCM (*Parallel Community Climate Model*). Ausschließlich für die Strahlungsberechnung tauschen die Prozesse untereinander Gitterzellen aus, um eine balancierte Partitionierung zu erstellen. Die besten Ergebnisse von 5,9 % Beschleunigung auf 512 Prozessoren erzielen sie mit einem statischen Verfahren, da die Lastschwankungen durch den Tagesgang ein regelmäßiges und vorhersagbares Muster aufweisen.

Lou und Farrara [65] stellen ein dynamisches Lastbalancierungsverfahren für die physikalische Parametrisierung des globalen atmosphärischen Modells UCLA AGCM (*University of California, Los Angeles, Atmospheric General Circulation Model*) vor. Dabei findet ein paarweiser Austausch von Gitterpunkten zwischen den Partitionen vor der Berechnung der Modellphysik statt. Die Paare werden durch die Rangfolge der P Partitionen bezüglich ihrer Rechenlast bestimmt: Rang i kommuniziert mit Rang $P - i + 1$. Dieses schnelle, lokale Verfahren wird wiederholt, bis die Lastimbalance unter einen spezifizierten Wert sinkt. Mit diesem Schema wird eine Leistungssteigerung des gesamten Modells um 10–15 % auf 240 Prozessen erreicht.

Für das regionale Modell RAMS (*Regional Atmospheric Modeling System*) wurde eine dynamische Lastbalancierung implementiert, welche auf dem *master-worker*-Schema basiert [110]. Dabei wird ausgenutzt, dass der Master regelmäßig alle Daten einsammelt, um sequentiell eine Ausgabedatei zu schreiben. Bei dieser Gelegenheit kann auch eine neue Partitionierung durch den Master vorgenommen werden, indem dieser die Daten – auf Zeitmessung basierend balanciert – neu auf die Worker verteilt. Dieser Ansatz ist jedoch nur experimentell [113] und inhärent nicht skalierbar. Zudem machen die Autoren keine Angaben zum erzielten Leistungsgewinn durch Lastbalancierung.

Das nicht mehr weiterentwickelte regionale Modell MM5 (*PSU/NCAR Mesoscale Model*) ist in der Lage durch die Bibliothek RSL (*Runtime System Library*) dynamische Lastbalancierung durchzuführen [73]. Dabei wird eine Partitionierung für M×N Prozesse mit folgender Methode erstellt [74]: Zuerst wird entlang einer der horizontalen Dimensionen eine Teilung in M (nicht notwendigerweise rechteckige) Teile vorgenommen. Danach werden diese Teile in der anderen horizontalen Dimension in N Teile zerlegt. Dabei werden die Gitterzellen entsprechend einer Zeitmessung der letzten Zeitschritte gewichtet. Durch die Lastbalancierung konnte die Effizienz von MM5 bei der Skalierung von 4 auf 64 Prozesse von 48 % auf 57 % erhöht werden.

Rodrigues et al. [90] stellen eine Methode vor, um dynamische Lastbalancierung in einem (nicht lastbalancierten) Simulationsmodell ohne tief greifende Änderungen der Datenstruktur zu ermöglichen. Die Autoren nutzen AMPI (*Adaptive MPI* [46]), um mittels Prozessorvirtualisierung das regionale Wettermodell BRAMS (*Brazilian developments on the Regional Atmospheric Modeling System*) zu balancieren. Um eine feingranulare Zerlegung des Gitters in migrierbare Blöcke zu erwirken, wird BRAMS mit einer vielfach höheren Zahl an Prozessen gestartet, als Prozessorkerne zu Verfügung stehen. AMPI bildet diese Prozesse auf Threads ab, so dass auf jedem Prozessorkern tatsächlich nur ein MPI-Prozess läuft, und migriert zur Lastbalancierung automatisch Threads zwischen den Prozessen. Durch Cache-Effekte und Ausnutzung überlappender Berechnung und Kommunikation werden schon allein durch die Virtualisierung von 1024 Prozessen auf 64 physische Prozesse ohne Lastbalancierung ca. 25 % Laufzeit eingespart. Zusammen mit Lastbalancierung basierend auf der Hilbert-Kurve wird eine um 32,5 % geringere Ausführungszeit erreicht.

Der Leistungssteigerung durch Lastbalancierung ist in den oben genannten Beispielen mit höchstens 16 % relativ gering. Bei Modellen, welche wesentlich komplexere physikalische bzw. chemische Prozesse einbeziehen, wie zum Beispiel bei Chemie-Transport-Modellen, kann eine dynamische Lastbalancierung lohnenswerter sein. Lastschwankungen entstehen hier aufgrund des Tagesgangs natürlicher und anthropogener Emissionen, Veränderungen der Photolyserate bei Sonnenauf- und -untergang sowie durch Wolkeneffekte [28]. Diese Prozesse führen zu zeitlich und räumlich unterschiedlichen Zeitskalen der chemischen Reaktionen und beeinflussen damit bei Lösern mit adaptiver Zeitschrittgröße den Rechenaufwand. Elbern [28] erzielt durch dynamische Lastbalancierung einen Gewinn von 36 % mit dem Modell EURAD (*European Air pollution Dispersion model*) auf 129 Prozessen. Das Partitionierungsverfahren basiert auf einer Teilung des Gitters in M×N uneinheitliche Rechtecke. Wolke et al. [121] nutzen im Modell MUSCAT (*Multiscale Chemistry Aerosol Transport*) eine Zerlegung des Gitters in Blöcke, wobei diese mit der

graphenbasierten Partitionierungsbibliothek ParMetis auf die Prozesse verteilt werden. Damit erreichen sie 30 % Leistungszuwachs bei 80 Prozessen gegenüber der statischen Partitionierung.

Ein weiterer Grund, dynamische Lastbalancierung einzusetzen, sind adaptiv verfeinerte Gitter, welche jedoch nur selten in atmosphärischen Modellen verwendet werden. Die zur Laufzeit lokal veränderliche Anzahl von Gitterpunkten erfordert nach jedem Adaptionsschritt eine Repartitionierung des Gitters, um optimale Lastbalance zu erhalten. Behrens et al. [6] nutzen in dem Framework amatos (*Adaptive Mesh Generator for Atmosphere and Ocean Simulation*) eine raumfüllende Kurve, um adaptive Dreiecks- und Tetraedergitter zu partitionieren. St-Cyr et al. [104] vergleichen zwei globale Modelle mit adaptiven Gittern, wobei in einem Fall die Bibliothek ParMetis zur Graphenpartitionierung verwendet wird.

3.2 Kopplung paralleler Simulationsmodelle

Viele Simulationsmodelle betrachten nur Teilprozesse eines Gesamtsystems oder nur Prozesse einer speziellen wissenschaftlichen Disziplin, wie zum Beispiel der Strömungsmechanik. Um detaillierte Simulationen zur Untersuchung neuer wissenschaftlicher Fragestellungen zu ermöglichen, müssen ganzheitliche Modelle der Systeme erstellt werden. Daher werden Modelle einzelner Prozesse zu *Modellsystemen* gekoppelt. Der wesentliche Aspekt dabei ist, dass so komplexe Systeme durch mehrere Teilmodelle (Komponenten) und deren Interaktionen gemeinsam simuliert werden [57]. Damit wird auch eine Zusammenführung des Expertenwissens aus unterschiedlichen Disziplinen erreicht. Eine hohe Verbreitung gekoppelter Modelle findet man in den Geowissenschaften. Dazu zählen insbesondere Klimamodelle, welche, wie in Abschnitt 2.2 erwähnt, aus Einzelmodellen zur Simulation der Atmosphäre, der Ozeane, des Meereises, der Landoberfläche, der Atmosphärenchemie und weiterer Elemente der Geosphäre bestehen [117]. Auch zur Prognose der Luftqualität wird typischerweise ein Modellsystem eingesetzt, welches aus einem regionalen Wettermodell und einem Chemie-Transport-Modell besteht [34, 121]. Ein weiterer typischer Einsatzbereich gekoppelter Simulationsmodelle sind die Ingenieurwissenschaften [119]. Beispiele sind die Interaktion von Strömung und Struktur zum Entwurf von Turbinen und Fahrzeugen sowie die Simulation von Prozessen in der Metallverarbeitung. Für gekoppelte Modelle werden auch die Begriffe Multiphysik-Modell (Interaktion verschiedener, einst isoliert betrachteter Phänomene) und Multiskalen-Modell (Interaktion von Phänomenen unterschiedlicher Größenskalen) verwendet [57].

Ein Hauptmerkmal der Modellkopplung sind Datenabhängigkeiten zwischen den beteiligten Teilmodellen. Ein gekoppeltes Modell liegt vor, wenn die Eingabevariablen zur Berechnung einer Iteration eines Modells von den Ausgabevariablen einer Iteration eines anderen Modells abhängen [57]. Somit ist der Datentransfer zwischen den beteiligten Modellen die wesentliche Aufgabe der Modellkopplung. Die Datenabhängigkeiten innerhalb eines Modellsystems lassen sich als Graph darstellen, wobei der Datentransfer zwischen zwei Modellen in nur eine Richtung oder wechselseitig erfolgen kann. Diesbezüglich kann ein Teilmodell somit Konsument, Produzent oder zugleich Konsument und Produzent von Kopplungsdaten sein. Verwenden die Modelle unterschiedliche räumlich Gitterstrukturen oder Auflösungen, so ist eine Transformation der Daten notwendig [56]. Zudem können unterschiedliche Schrittweiten zur Zeitintegration der Modelle eine zeitliche Interpolation oder Mittelung von Daten erfordern. Bei der Kopplung paralleler Simulationsmodelle tritt zudem dass sogenannte *M×N-Problem* auf, welches das Problem des Transfers von unterschiedlich verteilt gespeicherten Daten darstellt [9].

In den folgenden zwei Abschnitten werden das M×N-Problem und Aspekte der Architektur paralleler gekoppelter Modelle näher betrachtet. Danach wird bestehende Software zur Kopplung paralleler Modelle, insbesondere im Bereich atmosphärische Simulation, vorgestellt.

3.2.1 Das M×N-Problem

Bei der Kopplung paralleler Simulationsmodelle erhält der Datentransfer eine zusätzliche Komplexität, da die Partitionierung des Gitters jeweils verschieden sein kann. Diese Problematik wird als M×N-Problem bezeichnet [9], womit die Datenübertragung von einer Partitionierung im M Teilgebiete in eine unterschiedliche Partitionierung in N Teilgebiete gemeint ist. Dies erfordert zuvor eine Abbildung der verschiedenen Partitionierungen untereinander, um Überschneidungen zwischen den Partitionen zu identifizieren, was auch als *handshaking* der Modelle bezeichnet wird [56]. Damit wird festgestellt, von welchen Prozessen eines Modells welche Daten der Partitionen an welche Prozesse eines anderen Modells übertragen werden müssen. Die Lösung des M×N-Problems setzt voraus, dass von jedem Modell eine vollständige Beschreibung der Partitionierung verfügbar ist. Dabei wird häufig ein globaler Ansatz verfolgt, dass heißt die Beschreibung aller Partitionen wird in einem oder mehreren Prozessen vollständig gesammelt (siehe Abschnitt 3.2.3). Dieser globale Ansatz zur Verwaltung der Metadaten der Kopplung ist jedoch bezüglich des Speicheraufwands nicht skalierbar. Des Weiteren ist der Rechenaufwand des *handshaking* nicht skalierbar, wenn alle Partitionen des einen Modells mit denen des anderen Modells verglichen werden. Günstiger ist eine hierarchische Suche [88] oder die Dekomposition der Metadaten, so dass keine globale Suche erfolgen muss [123].

Verwenden die Modelle eine feste Partitionierung, so ist das *handshaking* nur einmal zu Beginn des gekoppelten Simulationslaufs notwendig. In diesem Fall spielt der dazu nötige Aufwand eine untergeordnete Rolle. Wird jedoch eine dynamische Lastbalancierung in einem oder mehreren Modellen des Modellsystems verwendet, so müssen die Überschneidungen nach jeder Repartitionierung neu bestimmt werden. Mit steigender Häufigkeit der Repartitionierung wird somit der Aufwand des *handshaking* in zunehmendem Maße kritisch für die Gesamtleistung des parallelen Modellsystems. Wie in Abschnitt 3.2.3 dargelegt, wird diese Problematik in bisherigen Arbeiten zur Modellkopplung nicht betrachtet.

3.2.2 Architektur paralleler gekoppelter Modellsysteme

Die Architektur paralleler gekoppelter Modellsysteme kann man hinsichtlich verschiedener Kriterien einteilen.

Offline- oder Online-Kopplung

Bei der Offline-Kopplung laufen die Modelle nicht simultan, sondern werden nacheinander abgearbeitet. Dabei erfolgt der Datenaustausch über Dateien, dass heißt ein Modell nutzt die dateibasierte Ausgabe eines oder mehrerer Modelle als Eingabe. Mit der Offline-Kopplung ist kein bidirektionaler Datenaustausch möglich [57]. Das konsumierende Modell kann zu einem beliebigen Zeitpunkt nach der Erzeugung seiner Eingabedaten gestartet werden. Da die Eingabedaten vollständig gespeichert werden, kann das konsumierende Modell beliebig oft – zum Beispiel mit unterschiedlichen Parametern – gestartet werden. Dieses Verfahren wird üblicherweise zur numerischen Wettervorhersage genutzt, bei der ein regionales Wettervorhersagemodell die Ausgabe eines globalen Modells nutzt [2]. Aufgrund des Datenaufkommens ist die Offline-Kopplung für eine enge Kopplung mit einer hohen Frequenz des Datenaustauschs ungeeignet. Zudem spricht der Aufwand des dateibasierten Austauschs gegen eine hochparallele Anwendung. Kommunizieren simultan laufende Modelle ohne Nutzung des Dateisystems, wird von Online-Kopplung gesprochen. Üblicherweise werden die Daten in Form von Nachrichten dem *message-passing*-Prinzip folgend übertragen. Im Gegensatz zur Offline-Kopplung wird somit der Engpass Dateisystem vermieden, was eine Anwendung in hochskalierenden Modellsystemen mit häufigem Datenaustausch ermöglicht. Da die Modelle zeitgleich Berechnung durchführen, ist zudem ein

bidirektionaler Datenaustausch möglich. Dadurch ist eine weitaus engere Kopplung der von den Modellen simulierten Probleme möglich. In dieser Arbeit wird daher nur die Online-Kopplung als Modellkopplung im engeren Sinne betrachtet.

Monolithischer Ansatz oder Nutzung eines Frameworks

Für den Aufbau online gekoppelter Modelle unterscheidet Bulatewicz [12] zwei prinzipielle Vorgehensweisen. Beim monolithischen Ansatz entsteht das gekoppelte Modell durch Vereinigung des Quellcodes mehrerer Modelle. Aus softwaretechnischer Sicht entsteht dabei jedoch ein Konglomerat mit hohem Wartungsaufwand und geringer Wiederverwendbarkeit. Der andere Ansatz basiert dagegen auf Softwareframeworks, welche von konkreten Modellen unabhängig und damit wiederverwendbar sind. Sie enthalten die grundlegenden Funktionalitäten der Kopplung wie das *handshaking*, den Datentransfer und die Transformation. Es können zwei Typen von Frameworks unterschieden werden [12]: Kommunikationsframeworks ermöglichen den Datenaustausch zwischen unabhängigen Modellen durch Instrumentierung des Quellcodes an den Positionen, wo Daten zwischen Modellen gesendet oder empfangen werden sollen [33, 56, 88]. Damit sind nur verhältnismäßig geringe Eingriffe in die ursprünglichen Modelle notwendig. Komponentenframeworks basieren dagegen auf einer Zerlegung in größtenteils unabhängige und wiederverwendbare Komponenten, welche typischerweise deutlich kleinteiliger als die Einzelmodelle bei Kommunikationsframeworks sind [11, 109]. Basierend auf standardisierten Schnittstellen der Komponenten werden deren Ein- und Ausgabedaten verknüpft und so Modellsysteme auf sehr flexible Weise erschaffen.

Organisation des handshaking

Das Ermitteln der Überschneidungen der Partitionen zweier gekoppelter Modelle kann zentralisiert durch einen Prozess der Modelle oder einen zusätzlichen Koppler-Prozess erfolgen [33]. Durch die zentralisierte Speicherung der Metadaten und Berechnung der Überschneidungen ist dieser Ansatz in seiner Skalierbarkeit jedoch beschränkt. Ein andere Möglichkeit ist ein verteiltes *handshaking*, bei dem jeder Prozess nur die für ihn relevanten Überschneidungen berechnet, basierend auf einer globalen Beschreibung der Partitionen des jeweils anderen Modells [56]. Für eine hohe Skalierbarkeit erweitern Zhang et al. [123] diese Methode um die Verteilung der Metadaten, so dass kein Prozess die gesamte Partitionsbeschreibung speichert.

Organisation des Datentransfers

Für den Datentransfer gibt es prinzipiell zwei Organisationsprinzipien [56]: direkter Transfer zwischen den einzelnen Prozessen der gekoppelten Modelle oder über eine Zwischeninstanz. Die Zwischeninstanz kann ein einzelner Prozess eines der Modelle oder ein zusätzlicher Koppler sein. Ein zusätzlicher Koppler bietet hohe Flexibilität bei Modellsystemen mit mehr als zwei Modellen, da die gekoppelten Modelle nur mit dem Koppler kommunizieren und die anderen Modelle dahinter verborgen bleiben. Zudem kann die Transformation der Daten zwischen unterschiedlichen Gitterstrukturen durch den Koppler übernommen werden. Aus Gründen der Skalierbarkeit werden auch parallelisierte Koppler genutzt [42, 88]. Der Vorteil des direkten Transfers ist der geringere Kommunikationsaufwand und die Vermeidung eines potentiellen Engpasses.

Parallele Abarbeitung der Modelle

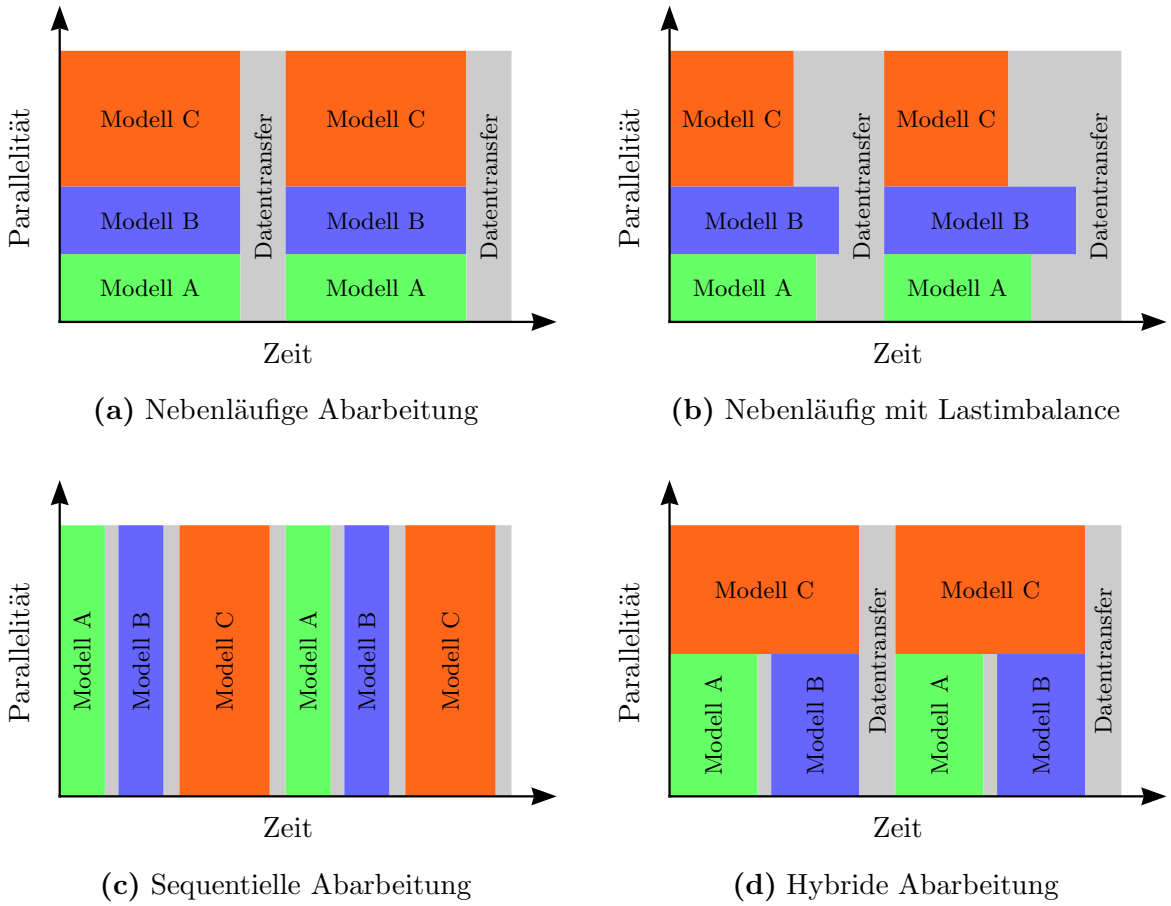
Das Schema der parallelen Abarbeitung der Teilmodelle (*component scheduling*) legt fest, in welcher Reihenfolge die Modelle auf einem Parallelrechner abgearbeitet werden. Grundsätzlich können zwei Varianten unterschieden werden: die *sequentielle Abarbeitung* und die *nebenläufige Abarbeitung* [56]. Bei der sequentiellen Abarbeitung führen alle Prozesse des Programms abwechselnd Berechnungen für jedes der gekoppelten Modelle aus. Zwischen den Berechnungsphasen findet der Datenaustausch zwischen den Modellen statt. Nutzen die Modelle die gleiche Gitterstruktur und Partitionierung, kann auf den Transfer verzichtet werden, wodurch die Modellkopplung aus Sicht der Implementierung stark vereinfacht wird. Bei der nebenläufigen Abarbeitung werden zu Beginn die Prozesse in Gruppen für jedes Modell unterteilt. Auf diesen Prozessgruppen werden die Modelle gleichzeitig abgearbeitet, unterbrochen von Phasen des Datenaustauschs zwischen den Modellen. Bei mehr als zwei gekoppelten Modellen ist auch eine Kombination von sequentieller und nebenläufiger Abarbeitung möglich, die sogenannte *hybride Abarbeitung* [117]. Die drei grundlegenden Abarbeitungsprinzipien sind in Abbildung 3.3 gegenübergestellt.

Der MPMD-Ansatz (*multiple program multiple data*) der nebenläufigen Abarbeitung ermöglicht es, auf das unter Umständen aufwendige zusammenführen eigenständiger Modelle zu einer Programmdatei zu verzichten. Ein zweiter Vorteil gegenüber dem sequentiellen Schema ist, dass durch die zusätzliche Ebene der Parallelität im Allgemeinen eine bessere Skalierbarkeit des Modellsystems zu erwarten ist. Keines der gekoppelten Modelle muss in der Lage sein über die volle Zahl an Prozessen zu skalieren. Voraussetzung für eine hohe Skalierbarkeit ist allerdings eine balancierte Aufteilung der Prozesse zu den Modellen. Wie in Abbildung 3.3(b) gezeigt, führt eine ungünstige Aufteilung zu Lastimbalance zwischen nebenläufig gekoppelten Modellen durch die Synchronisation beim Datentransfer. Aufgrund der unterschiedlichen Skalierbarkeit der Teilmodelle kann die optimale Aufteilung oft nur durch Testen verschiedener Konfigurationen gefunden werden, wie zum Beispiel bei Klimamodellen [23, 117]. Eine effiziente feste Aufteilung kann unmöglich werden, wenn eines der gekoppelten Modelle eine mit der Zeit variierende Rechenlast (temporale Lastimbalance) aufweist [61]. Sivagama et al. [101] stellen einen Ansatz vor, temporale Lastimbalancen bei einem nebenläufig gekoppelten Modell durch dynamische Auslagerung von Berechnungen auf wartende Prozesse eines anderen Modells auszugleichen. Ein anderes Konzept zur Lastbalancierung zwischen nebenläufig arbeitenden Modellen schlagen Kim et al. [52] vor. Dabei ist die Aufteilung der Prozesse zwischen den Modellen variabel und wird zur Laufzeit hinsichtlich der Lastbalance optimiert. Die Modelle müssen dazu jedoch eine Repartitionierung mit Veränderung der Prozessanzahl unterstützen. Bei der sequentiellen Abarbeitung wird die Problematik der Zuordnung der Prozesse zu Teilmodellen vermieden, so dass Lastimbalancen zwischen den Modellen nicht auftreten können. Andererseits ist hier die Skalierbarkeit der Einzelmodelle in stärkerem Maße entscheidend für die Skalierbarkeit des gesamten Modellsystems.

3.2.3 Software zur Kopplung paralleler Simulationsmodelle

Aufgrund der Komplexität des parallelen Kopplungsproblems wurden verschiedene, unabhängige Frameworks zur Modellkopplung implementiert, welche zum Teil unterschiedliche Schwerpunkte in Bezug auf generelle Anwendbarkeit, Flexibilität, Funktionsumfang und Skalierbarkeit setzen. In diesem Abschnitt werden vorhandene Frameworks vorgestellt und Aspekte der Parallelisierung, Verteilung von Metadaten und Skalierbarkeit diskutiert.

Das Kommunikationsframework *Model Coupling Toolkit* (MCT) ermöglicht die sequentielle und nebenläufige Kopplung paralleler Simulationsmodelle mit einem direkten Datentransfer [56]. Die Metadaten zur Kopplung werden global in jedem Prozess gespeichert. MCT wird vor allem in den Geowissenschaften eingesetzt [116]. Zum Beispiel im *Community Climate System Model*



(a) Nebenläufige Abarbeitung

(b) Nebenläufig mit Lastimbalance

(c) Sequentielle Abarbeitung

(d) Hybride Abarbeitung

Abbildung 3.3: Prinzipien der parallelen Abarbeitung gekoppelter Simulationsmodelle.

(CCSM), welches einen zusätzlichen parallelisierten Koppler für den Datenaustausch der Modelle der Atmosphäre, des Ozeans, des Meereises und der Landoberfläche verwendet [42]. Die Teilmodelle kommunizieren mittels MCT mit dem Koppler, welcher auf einer eigenen Partitionierung die Transformation zwischen den Gittern vornimmt. In CCSM ist die nebenläufige, sequentielle oder hybride Abarbeitung der Teilmodelle möglich. Dennis et al. [23] zeigen, dass mit dem hybriden Schema eine Skalierbarkeit bis auf 6380 Prozesse einer Cray XT4 für das gesamte Modellsystem erreicht werden kann.

Ebenfalls im Bereich der Klima- und Erdsystemmodellierung wird das Kommunikationsframework OASIS4 (*Ocean Atmosphere Sea Ice Soil*) eingesetzt [88]. Zur Gittertransformation wird hier wie bei CCSM eine parallelisierte Zwischeninstanz genutzt, wobei nur die nebenläufige Abarbeitung der Teilmodelle möglich ist. Beim *handshaking* speichern die Komponenten an globalen Metadaten nur die Hülle jeder Partition eines anderen Teilmodells, während die genauen Gitterpunkte nur für jene Partitionen gespeichert werden, welche sich mit der eigenen Partition überschneiden. Damit wird eine teilweise, statische Verteilung der Metadaten erreicht.

Das Komponentenframework PALM (*Projet d'Assimilation par Logiciel Multiméthode*) ermöglicht neben einem fest vorgegebenen Ablauf der Modelle (nebenläufig, sequentiell und hybrid) auch eine dynamische Ausführung der Modelle [11]. Dazu können Schleifen und Bedingungen zur Steuerung des Modellsystems mithilfe einer grafischen Benutzeroberfläche definiert werden. Der Datenaustausch zwischen parallelen Komponenten erfolgt direkt.

Das Kommunikationsframework MpCCI (*Mesh-based parallel Code Coupling Interface*) ist vor allem auf die Kopplung ingenieurwissenschaftlicher – auch kommerzieller – Simulationsprogram-

me ausgelegt [33]. Dafür stehen vorgefertigte Schnittstellen zu einer Vielzahl an Programmen bereit. Ähnlich wie bei PALM liegen mehr die Benutzbarkeit und Flexibilität und weniger eine hohe Skalierbarkeit im Fokus von MpCCI [119]. Der Datenaustausch erfolgt bei MpCCI über eine Zwischeninstanz, welche auch die Gittertransformation vornimmt.

Das Kommunikationsframework Seine [123] enthält ein skalierbares Konzept zur Verwaltung der Kopplungsmetadaten, die zur Beschreibung der Partitionierung der Modelle notwendig sind. Die Metadaten werden in einem parallelisierten Koppler mithilfe einer raumfüllenden Kurve unabhängig von der Dekomposition der Modelle verteilt. Auch eine Balancierung der Metadaten durch Umverteilung gemäß der raumfüllenden Kurve ist möglich. Diese Dekomposition ist den gekoppelten Modellen bekannt, so dass deren Prozesse beim *handshaking* nur mit den entsprechenden Prozessen des Kopplers kommunizieren müssen. Für Seine wurden Skalierbarkeitsmessungen bis zu 512 Prozesse durchgeführt, wobei jedoch relativ hohe Kosten für das *handshaking* beobachtet wurden, welche auf zunehmenden Kommunikationsaufwand zurückgeführt werden.

Zur Simulation des Weltraumwetters mit einer Vielzahl gekoppelter Modelle wurde das Komponentenframework SWMF (*Space Weather Modeling Framework*) entwickelt [109]. Mit SWMF wurden Simulationen mit acht hybrid gekoppelten Modellkomponenten auf bis zu 256 Prozessen durchgeführt. An dem Modellsystem sind auch Modelle mit adaptiv verfeinertem Gitter und dynamisch verteilten Blöcken beteiligt. In diesem Fall werden die Kopplungsdaten des gesamten Gitters an alle Prozesse eines solchen Modells gesendet und dort der entsprechende Teil der Daten auf das adaptive Gitter transformiert. Dieser Ansatz umgeht zwar das Problem, dass die sendenden Prozesse die sich dynamisch ändernde Gitterstruktur und Dekomposition speichern müssen, ist jedoch aufgrund der Replikation der globalen Daten nicht skalierbar.

Ein weiteres Modellsystem, bei dem ein Modell mit dynamischer Lastbalancierung beteiligt ist, stellt COSMO-MUSCAT dar [121]. Die Partitionierung des Chemie-Transport-Modells MUSCAT wird regelmäßig mittels Graphenpartitionierung an die lokal variierende Rechenlast angepasst (siehe Abschnitt 3.1.4). Der Datenaustausch mit dem Wettermodell COSMO erfolgt mithilfe der Bibliothek MDE (*Multiblock Data Exchange*) [61]. Mit MDE ist sowohl die nebenläufige als auch die sequentielle Kopplung möglich, wobei die sequentielle Kopplung bei stark schwankender Gesamtlast von MUSCAT bevorzugt wird. Eine Optimierung hinsichtlich der dynamischen Lastbalancierung in MUSCAT findet jedoch nicht statt. Dass heißt, nach jeder Repartitionierung müssen die Kommunikationsbeziehungen vollständig neu berechnet werden. Zudem speichert der Koppler MDE die Metadaten für die Partitionierung global in jedem Prozess. Für COSMO-MUSCAT wurden Skalierbarkeitsmessungen bis 128 Prozesse durchgeführt, wobei sich die Kopplung nicht als Engpass herausstellte.

Die im Kontext dieser Arbeit wichtigen Probleme der Verknüpfung von Kopplung und dynamischer Lastbalancierung sowie der hochskalierbaren Identifikation der Überschneidungen zwischen den Partitionen werden von den bestehenden Lösungen zur Kopplung paralleler Simulationsmodelle nicht genügend behandelt. In den meisten Arbeiten wird nur eine moderate Anzahl an Prozessen, meist unter tausend, genutzt, so dass keine vordergründigen Skalierbarkeitsprobleme durch *handshaking* und globale Metadatenverwaltung auftreten. Keines der bestehenden Frameworks ist für die Kopplung von Modellen mit dynamischer Lastbalancierung optimiert.

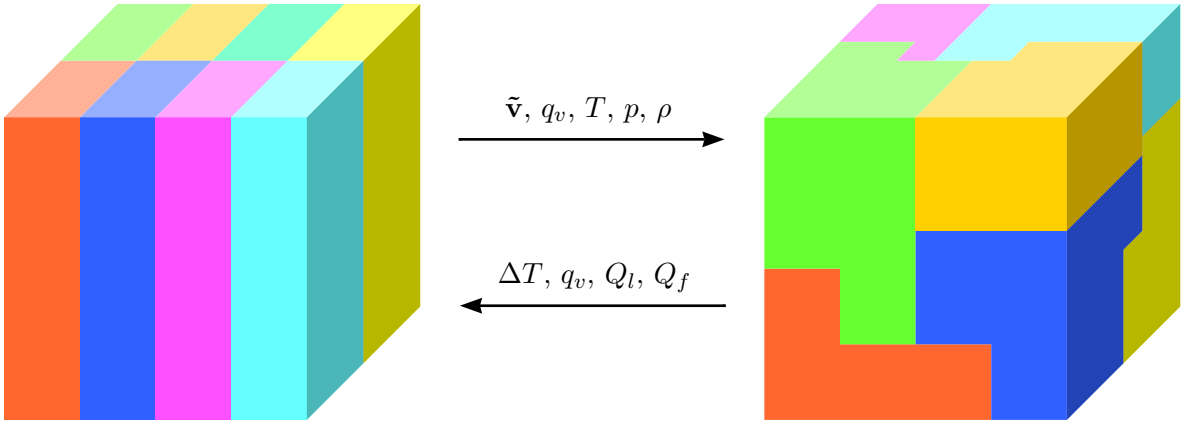
4 Design einer lastbalancierten Kopplung von Wolkenprozessen

Basierend auf der Analyse der Defizite und Anforderungen zur effizienten Nutzung detaillierter Wolkenmodelle in atmosphärischen Simulationsmodellen in Abschnitt 2.7 wird in diesem Kapitel eine hochskalierbare und lastbalancierte Kopplung entworfen. Dabei werden in Kapitel 3 vorgestellte Methoden angewandt und für eine hochskalierende Anwendung erweitert. Im ersten Abschnitt 4.1 wird das grundlegende Konzept der lastbalancierten Kopplung vorgestellt, welches in den darauf folgenden Abschnitten konkretisiert wird. Die parallelisierten Datenstrukturen zur Repräsentation des numerischen Gitters werden in Abschnitt 4.2 eingeführt. Danach wird in Abschnitt 4.3 die hochskalierbare dynamische Lastbalancierung für die detaillierte Mikrophysik entworfen. In Abschnitt 4.4 wird schließlich die Kopplung zwischen dem atmosphärischem Modell und der Wolkenmikrophysik in Verbindung mit der Lastbalancierung konzipiert. Das Konzept der verteilten und dynamisch lastbalancierten Datenstrukturen und der Kopplung wird im Folgenden als *FD4*-Konzept (*four-dimensional distributed dynamic data structures*) bezeichnet.

4.1 Konzept der lastbalancierten Kopplung

Das Konzept der lastbalancierten Kopplung sieht eine neue Strukturierung der Daten und Berechnungen des spektralen Mikrophysikmodells vor, um die bestehenden Defizite zu lösen. Um eine räumlich und zeitlich variierende Rechenlast in Simulationsmodellen gleichmäßig über alle Prozesse zu verteilen, werden Methoden der dynamischen Lastbalancierung eingesetzt, wie sie in Abschnitt 3.1.2 eingeführt worden sind. Diese Konzepte sollen auch für die spektrale Mikrophysik umgesetzt werden. Folglich müssen Teile des Gitters dynamisch den Prozessen zugewiesen werden, um so die Partitionen während eines Simulationslaufes an die sich verändernde Rechenlast anzupassen. Da in atmosphärischen Simulationsmodellen typischerweise keine dynamische Lastbalancierung vorgesehen ist, setzt dies Änderungen an den grundlegenden Datenstrukturen und dem Schema der Partitionierung voraus. Solche tief greifenden Modifikationen bedeuten jedoch praktisch die Neuentwicklung des dynamischen Kerns, was für ein komplexes und von vielen Institutionen operationell oder zur Forschung genutztes Simulationsmodell nicht wünschenswert ist. Zudem ist eine Umverteilung der Berechnungen der atmosphärischen Dynamik nicht nötig, da diese im Allgemeinen keine Lastschwankungen aufweisen.

Das *FD4*-Konzept sieht daher vor, dass keine Änderungen an den Datenstrukturen des atmosphärischen Modells vorgenommen werden. Stattdessen werden die Daten der spektralen Mikrophysik vollständig außerhalb des atmosphärischen Modells verwaltet und können somit unabhängig von diesem dynamisch lastbalanciert werden. Die Berechnungen der spektralen Mikrophysik und der Advektion der Hydrometeore werden somit von dem atmosphärischen Modell getrennt. Wie in Abbildung 4.1 dargestellt, entstehen für das gleiche numerische Gitter zwei unterschiedliche Partitionierungen, zwischen denen regelmäßig Daten ausgetauscht werden. Der Datenaustausch findet immer vor und nach den Berechnungen der spektralen Mikrophysik jedes Zeitschritts des atmosphärischen Modells statt und beinhaltet keine spektralen Variablen. Daher ist davon auszugehen, dass der daraus resultierende Kommunikationsaufwand im Vergleich zum



Atmosphärisches Modell	**Kopplung**	**Spektrale Mikrophysik**
2D Dekomposition	Regelmäßiger	3D Dekomposition
Statische Partitionierung	Datenaustausch	Dynamische Lastbalancierung

Abbildung 4.1: Schematische Darstellung des *FD4*-Konzepts der lastbalancierten Kopplung der detaillierten Wolkenmikrophysik an ein atmosphärisches Modell.

Randaustausch der Hydrometeorspektren innerhalb der Partitionierung der Mikrophysik relativ gering ausfällt.

Das Ergebnis dieser Separation und Wiederverbindung ist vergleichbar mit Modellsystemen, die durch Kopplung eigenständiger Modelle mit jeweils eigenen Datenstrukturen entstehen, wie zum Beispiel Klimamodelle – nur das dort keine dynamische Lastbalancierung berücksichtigt wird. Die parallele Abarbeitung der gekoppelten Modelle wird dabei so belassen wie im ursprünglichen Modellsystem. Dass heißt, jeder Prozess berechnet abwechselnd eine Partition des atmosphärischen Modells und eine des Mikrophysikmodells (sequentielle Abarbeitung). Diese beiden Partitionen sind jedoch nicht identisch. Der grobe Ablauf eines Zeitschritts des atmosphärischen Modells Δt^{dyn} stellt sich nun wie folgt dar:

1. Zeitintegration des atmosphärischen Modells für Δt^{dyn}.
2. Übertragung der Kopplungsdaten in die Partitionierung der Mikrophysik.
3. Zeitintegration des Mikrophysikmodells für $m \times \Delta t^{mp} = \Delta t^{dyn}$, dabei findet eine dynamische Lastbalancierung statt.
4. Übertragung der Kopplungsdaten in die Partitionierung des atmosphärischen Modells.

Die dynamische Lastbalancierung der Wolkenmikrophysik erzeugt einen zusätzlichen Aufwand, da regelmäßig die Lastbalance überprüft, gegebenenfalls eine neue Partitionierung erstellt und Gitterbereiche zwischen den Prozessen migriert werden müssen. Zudem bestimmt die Partitionierung auch den Aufwand des Austauschs der Daten an den Partitionsrändern, welcher eines der zu lösenden Defizite ist. Daher sieht das Konzept vor, eine Aufteilung des Gitters in drei Raumdimensionen vorzunehmen. Dies ermöglicht die Reduktion des Oberfläche-Volumen-Verhältnisses der Partitionen im Vergleich zur zweidimensionalen Partitionierung und somit die Reduktion des Kommunikationsaufwands. Zudem wird so eine feinere Granularität der Gitterzerlegung erreicht, welche notwendig ist, um eine effektive Lastbalancierung zu erreichen. Ein geeignetes Partitionierungsverfahren muss identifiziert werden, welches nicht nur eine hinsichtlich Lastbalance und Kommunikationsaufwand optimierte Partitionierung erstellt, sondern auch zugleich den Aufwand der Lastbalancierung selbst reduziert und hochskalierbar ist.

Die wesentlichen Schwerpunkte des *FD4*-Konzepts lassen sich wie folgt zusammenfassen:

- Die Datenstrukturen und die Parallelisierung des atmosphärischen Modells sind unverändert. Typischerweise wird hier eine statische zweidimensionale Partitionierung eingesetzt.
- Die Berechnungen und Datenstrukturen des Mikrophysikmodells werden von dem atmosphärischen Modell getrennt.
- Für das Mikrophysikmodell wird eine dynamische Lastbalancierung basierend auf einer dreidimensionalen Dekomposition des Gitters eingesetzt. Somit können die Lastschwankungen ausgeglichen und der Kommunikationsaufwand reduziert werden.
- Die Kopplung beider Modellkomponenten auf unterschiedlichen Partitionierungen erfordert einen zusätzlichen Datentransfer zwischen Prozessen.

4.2 Grundlegende Datenstruktur und Parallelisierung

Ein wesentlicher Schwerpunkt des *FD4*-Konzepts ist die Trennung der verteilten Datenstrukturen der spektralen Mikrophysik von denen des atmosphärischen Modells. Auf diese Weise kann eine dynamische Lastbalancierung erfolgen, ohne dass weitreichende Änderungen an dem atmosphärischen Modell nötig sind. Folglich müssen neue Datenstrukturen entwickelt werden, welche hinsichtlich der Anforderungen spektraler Mikrophysikmodelle optimiert sind und eine Abstraktion bieten, welche eine hohe Flexibilität für verschiedene Mikrophysikmodelle sicherstellt. Die Verwaltung der verteilten Datenstrukturen wird dafür vollständig von dem konkreten Modell getrennt.

4.2.1 Dreidimensionale Dekomposition

Die Zerlegung des regulären rechteckigen Gitters erfolgt in allen drei Raumdimensionen, was einen bedeutenden Unterschied zu atmosphärischen Modellen darstellt. Um eine dynamische Lastbalancierung zu ermöglichen, wird das Gitter in eine hohe Zahl von rechteckigen *FD4-Blöcken* zerlegt. Diese dienen als feingranulare, zwischen den Partitionen zu verteilende Einheit. Die Blockzerlegung erfolgt in $B = B_x \times B_y \times B_z$ Blöcke, so dass das Gitter, welches durch die Blöcke gebildet wird (*Blockgitter*), wiederum ein reguläres Rechteckgitter darstellt. Dazu werden $B_d - 1$ Schnittflächen für jede der drei Raumdimensionen $d \in \{x, y, z\}$ durch das Rechengitter gelegt. Um beliebige Gittergrößen zuzulassen, kann die Blockzerlegung in ungleichmäßigen Abständen je Dimension erfolgen, so dass die Gittergröße nicht dem Vielfachen einer festen Blockgröße entsprechen muss. In diesem Fall entstehen Blöcke verschiedener Größe. Die Partition eines Prozesses besteht aus einem oder mehrerer dieser Blöcke.

In den *FD4*-Blöcken werden vierdimensionale Datenfelder gemäß einer Beschreibung der für das Mikrophysikmodell benötigten Variablen bereitgestellt, der *Variablentabelle*. Die vier Dimensionen bestehen aus den drei Raumdimensionen, über welche die Blockdekomposition erfolgt, sowie einer weiteren Dimension zur Abbildung der Größenspektren. In der Variablentabelle werden neben dem Namen der Variable und der Anzahl der Größenklassen weitere Eigenschaften definiert. Diese beinhalten die Zahl der für die Zeitintegration zu speichernden Zeitschritte, der initiale Wert der Variable und eine Angabe darüber, an welchem Punkt innerhalb einer Gitterzelle die Variable angeordnet ist. Diese Angabe ist nötig, um die versetzte Anordnung des in atmosphärischen Modellen üblichen Arakawa-C Gitters zu ermöglichen (siehe Abschnitt 2.2). Skalare Größen, wie die Temperatur oder die Größenspektren der Hydrometeore, werden im Zellmittelpunkt angeordnet. Die Komponenten des Geschwindigkeitsvektors werden dagegen auf die Flächen der Zellen, welche orthogonal zur jeweiligen Flussrichtung stehen, abgebildet. Durch die

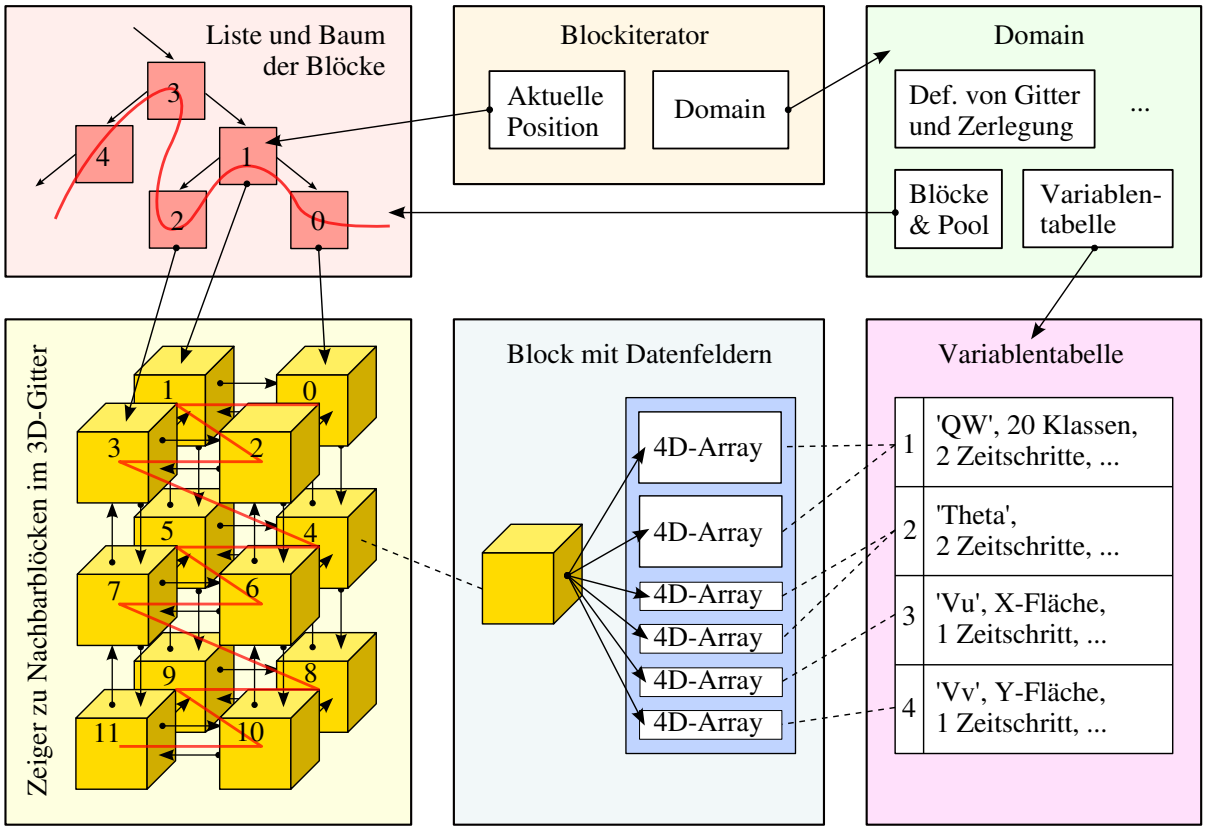


Abbildung 4.2: Die grundlegenden Datenstrukturen des *FD4*-Konzepts der dynamischen Datenverwaltung und dreidimensionalen Blockdekomposition mit einer beispielhaften Variablentabelle. Elemente der Parallelisierung sind aus Gründen der Übersichtlichkeit nicht dargestellt.

Reihenfolge in der Tabelle erhalten alle Variablen einen *Variablenindex*, welcher für den Zugriff auf die entsprechenden Datenfelder in den Blöcken verwendet wird.

Für die Lastbalancierung ist das dynamische Hinzufügen und Entfernen von Blöcken notwendig. Daher sind die Blöcke in einer dynamischen Datenstruktur abgelegt, wobei ein Schlüssel für jeden Block aus seiner Position im Blockgitter erzeugt wird. Die Datenstruktur muss ein schnelles Zugreifen, Einfügen und Entfernen an beliebiger Position sowie schnelles Iterieren gewährleisten. Diesen Anforderungen wird ein balancierter Binärbaum [19] in Kombination mit einer doppelt verketteten Liste der Knoten gerecht. Des Weiteren sind die Blöcke in einer zweiten Datenstruktur enthalten: um einen direkten Zugriff auf lokale Nachbarn im Blockgitter zu ermöglichen, enthalten die Blöcke Zeiger zu den sechs unmittelbar benachbarten Blöcken. Um bei regelmäßiger Repartitionierung ein häufiges Neuanlegen und Entfernen von Blöcken zu vermeiden, werden nicht gebrauchte Blöcke in einem *Pool* zwischengespeichert. Wird zu einem späteren Zeitpunkt ein neuer Block benötigt, kann dieser aus dem Pool entnommen werden. Da die Blöcke prinzipiell unterschiedlich groß sein können, existiert je Blockgröße ein solcher Pool.

Ein Gitter mit zugeordneter Variablentabelle wird von der *FD4-Domain* verwaltet. Im Konzept ist die Domain die zentrale Klasse, welche alle Daten enthält, um die dreidimensionale Blockdekomposition darzustellen. Als Schnittstelle für den Zugriff auf die Daten der Blöcke dient der *Blockiterator*, welcher einen transparenten Zugriff auf die dynamische Datenstruktur der Blöcke gewährleistet. Abbildung 4.2 fasst die grundlegenden Datenstrukturen für die dreidimensionale Blockdekomposition zusammen.

4.2.2 Optimierungen für spektrale Wolkenmikrophysik

Aus den speziellen Anforderungen der spektralen Wolkenmikrophysik geht hervor, dass eine feingranulare Zerlegung des Gitters für eine hohe Leistung und Skalierbarkeit unumgänglich ist. Dass heißt, das Gitter wird in Blöcke unterteilt, welche nur relativ wenige Gitterzellen beinhalten. Dies erschließt sich aus den folgenden zwei Überlegungen: Zum einen benötigt die spektrale Mikrophysik zur Darstellung des Größenspektrums für verschiedene Hydrometeorkategorien eine hohe Zahl an Variablen je Gitterzelle, typischerweise mehrere hundert bis über tausend. Durch die hohe Variablenzahl übertreffen nur Blöcke mit relativ wenigen Gitterzellen nicht die Größe des Prozessorcaches. Zum Beispiel beträgt der Speicherbedarf eines Blocks mit 8^3 Gitterzellen und 1024 Variablen (je 64 bit) 4 MiB.

Die zweite Überlegung basiert auf der Feststellung, dass eine dynamische Lastbalancierung nur dann effektiv sein kann, wenn kein Blockgewicht w_i größer als die optimale Last der Partitionen ist [118]. Folglich ist die maximal effizient nutzbare Anzahl an Prozessen P durch das höchste Blockgewicht w_{max} beschränkt:

$$w_{max} \leq \frac{\sum w_i}{P} \quad \Rightarrow \quad P \leq \frac{\sum w_i}{w_{max}} \tag{4.1}$$

Um eine hohe Skalierbarkeit zu erreichen, welche für die Beschleunigung der aufwendigen spektralen Mikrophysik essenziell ist, muss also das höchste Blockgewicht durch eine feingranulare Blockzerlegung reduziert werden. Um eine Abschätzung für die nötige Anzahl an Blöcken im Verhältnis zur Anzahl der Prozesse zu erhalten, dividiert man Gleichung 4.1 durch die Blockanzahl B und erhält:

$$\frac{P}{B} \leq \frac{\sum w_i}{B\, w_{max}} \quad \Rightarrow \quad B \geq P\, \frac{w_{max}}{avg(w_i)} = P\, \lambda^{-1} \tag{4.2}$$

Gleichung 4.2 sagt aus, dass zur effektiven Lastbalancierung insgesamt soviel Mal mehr Blöcke als Prozesse benötigt werden, wie das Verhältnis aus maximalem und durchschnittlichem Blockgewicht λ^{-1} beträgt. Dabei ist λ^{-1} ein Maß für die Lastimbalance der Blockdekomposition, welches auch (neben dem Lastverhalten des Simulationsmodells) von B abhängt. Für COSMO-SPECS wurde bei der Analyse der Lastbalance in Abschnitt 2.6.3 bezogen auf Gitterzellen als Blöcke (d. h. mit maximal möglichem B) $w_{max}/w_{min} = 12$ ermittelt, was als obere Schranke für λ^{-1} gelten kann. Um das mit 256×256×80 Gitterpunkten verhältnismäßig große Gitter der Studie von Planche et al. [83] (siehe auch Abschnitt 2.5.2) beispielsweise auf 6400 anstelle von 64 Prozessen effizient lastbalanciert auszuführen, müsste mit $\lambda^{-1} = 10$ die maximale Blockgröße $256{\times}256{\times}80/6400/10 \approx 82$ Gitterpunkte betragen.

Eine Zerlegung des Gitters in Blöcke, die nur aus ca. 4^3 Gitterzellen oder weniger bestehen, ist somit Voraussetzung für eine effiziente hochskalierbare Ausführung des lastbalancierten spektralen Mikrophysikmodells. Dass heißt, das *FD4*-Konzept muss eine Optimierung hinsichtlich dieser relativ geringen Blockgröße beinhalten. In dem Zusammenhang sind die Datenstrukturen für die Ränder benachbarter Blöcke von entscheidender Bedeutung. Diese werden für die Approximation der Differenzenquotienten in der parallelisierten Advektionsberechnung benötigt. In atmosphärischen Simulationsmodellen wird – in Abhängigkeit der Ordnung des jeweils eingesetzten Advektionsverfahrens – üblicherweise ein Rand von zwei oder drei Zellen der Nachbarpartitionen benötigt [93, 102]. Überträgt man die bei statischen Partitionierungen übliche Methode der Erweiterung der Datenfelder um einen Rand aus einer bestimmten Zahl von Geisterzellen auf die dreidimensionale Blockdekomposition, so führt dies zu einem enormen Mehrbedarf an Speicherkapazität. Für kubische Blöcke der Kantenlänge b mit g Schichten an Geisterzellen erhöht sich der Speicherbedarf durch Geisterzellen um den Faktor f_M mit:

$$f_M(b, g) = \frac{(2g + b)^3}{b^3} \tag{4.3}$$

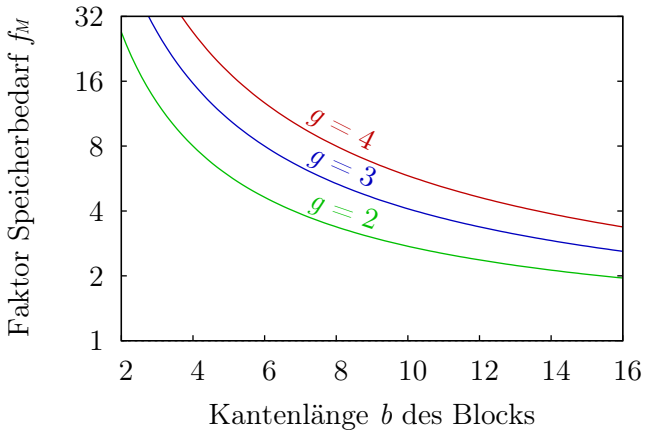


Abbildung 4.3: Faktor der Erhöhung des Speicherbedarfs f_M bei Anwendung von g Geisterzellen in Abhängigkeit der Kantenlänge b der kubischen Blöcke.

In Abbildung 4.3 ist f_M für $g \in \{2, 3, 4\}$ über dem Intervall $b = [2, 16]$ dargestellt. Um beispielsweise zwei Schichten von Geisterzellen um einen Block der Größe 4^3 zu legen, werden insgesamt 8^3 Gitterzellen benötigt, was einer achtfachen Erhöhung des Speicherbedarfs entspricht.

Aufgrund dieses hohen zusätzlichen Speicherbedarfs sieht der Entwurf der dynamisch verteilten Datenstrukturen keine Erweiterung der Blöcke um Geisterzellen vor. Um dennoch auf Variablen benachbarter Blöcke lesend zuzugreifen, müssen zwei Fälle unterschieden werden: Befindet sich der Nachbarblock innerhalb der gleichen Partition, so sind die Daten lokal verfügbar. Im anderen Fall müssen die Variablen von einem anderen Prozess übertragen und lokal geeignet zwischengespeichert werden. Dafür werden *Geisterblöcke* eingeführt, welche anstelle der Nachbarblöcke in das durch Nachbarzeiger gebildete Blockgitter eingefügt werden. Das Konzept berücksichtigt nur Stencil-Operationen ohne diagonale Zugriffe, so dass Geisterblöcke nur für die sechs direkten Nachbarn erforderlich sind. Die Übertragung der Daten von Blöcken zu Geisterblöcken eines anderen Prozesses wird in Abschnitt 4.2.3 erläutert. Durch die Geisterblöcke und Nachbarzeiger besteht zwar ein einheitlicher Zugriff auf Variablen von lokalen Nachbarblöcken und Nachbarblöcken anderer Prozesse. Jedoch ist die Implementierung Stencil-basierter Algorithmen auf diese Weise aufwendig und unportabel, da auf lokale Zellen und Zellen von Nachbarblöcken nicht auf gleiche Weise zugegriffen werden kann. Daher wurden für den Entwurf folgende Strategien des Zugriffs auf die Variablen der lokalen Blöcke und der Nachbarblöcke hinsichtlich Flexibilität und Leistung evaluiert:

- *Kopie*: Einheitlicher Zugriff durch Kopieren des Blocks und der Ränder der sechs Nachbarblöcke in ein größeres, kontinuierliches Datenfeld zur Berechnung.
- *Zeiger*: Einheitlicher, jedoch indirekter Zugriff durch ein kontinuierliches Feld von Zeigern auf Zellen des lokalen Blocks und der Nachbarzellen.
- *Anpassung*: Uneinheitlicher Zugriff mit speziellen Anpassungen im Algorithmus.

Die Strategie *Kopie* bietet die höchste Flexibilität, da Algorithmen für um Geisterzellen erweiterte Datenfelder nicht geändert werden müssen. Aus Sicht des Algorithmus ist die Art des Zugriffs auf die benachbarten Blöcke transparent. Für die Strategie *Zeiger* müssen alle Zugriffe auf gleiche Weise durch den indirekten Zugriff auf des Feld der Zeiger ersetzt werden, was moderaten Aufwand verlangt, jedoch eine für dieses Konzept spezifische Modifikation darstellt. Die Strategie *Anpassung* ist hochgradig unflexibel, da jeder Algorithmus selbst eine unterschiedliche Behandlung lokaler Zellen und Zellen von Nachbarblöcken einführen muss. Hierbei ist ein hoher Aufwand durch den Nutzer nötig, wodurch sich auch die Fehleranfälligkeit erhöht.

Zur Bewertung der drei Strategien bezüglich ihrer Rechenleistung wurde ein Vergleich mit einem synthetischen, seriellen Benchmark durchgeführt. Da die Leistungsfähigkeit der konzipierten Datenstrukturen für den geplanten Einsatzbereich von hoher Bedeutung ist, fließt sie mittels dieser Messungen in Entwurfsentscheidungen ein. Dabei ist insbesondere eine Abhängigkeit von der Größe des Prozessorcaches und den Optimierungsfähigkeiten des verwendeten Compilers zu erwarten. Als Referenz wurden auch eine Version mit um Geisterzellen erweiterten Blöcken (*Geisterzellen*) und eine Version ohne Blockzerlegung (*keine Blöcke*), dass heißt mit einem kontinuierlichen Datenfeld für das gesamte Gitter, in den Vergleich einbezogen. Der Benchmark berechnet 100 Zeitschritte eines Advektionsverfahrens für 800 Variablen je Zelle in einem dreidimensionalen Gitter mit einem 13-Punkt-Stencil ohne diagonale Elemente. Dabei werden für alle Variablen der alte und der neu berechnete Wert im Speicher gehalten, so dass effektiv 1600 Variablen notwendig sind. Hinzu kommen drei Variablen je Zelle für den Windvektor. Die hohe Zahl von Variablen steht hierbei modellhaft für die spektrale Beschreibung der Größenverteilung der Hydrometeore und die geringe Gittergröße von 16^3 für die aus der angestrebten hohe Parallelität resultierende geringe Partitionsgröße. Zur Integration eines Gitterpunkts sind jeweils zwei Nachbarpunkte in jeder Richtung und Dimension notwendig. Die Größe der Blöcke wurde von $2^3 = 8$ bis $16^3 = 4096$ Gitterzellen variiert, was einer Blockanzahl von 512 bis eins entspricht. Wie zu Beginn dieses Abschnitts erläutert, sind insbesondere geringe Blockgrößen bis zu 128 Gitterzellen von besonderem Interesse. Da der Anteil nicht lokaler Speicherzugriffe mit feinerer Blockzerlegung steigt, ist jedoch mit sinkender Leistung bei kleiner werdender Blockgröße zu rechnen. Zudem wächst dabei der Gesamtrechenaufwand, da Berechnungen des Advektionsverfahrens auf den Flächen der Gitterzellen an Blockgrenzen doppelt durchgeführt werden (vgl. Abschnitt 2.6.2). Zu erwarten ist, dass die höchste Leistung bei der größten Blockgröße erreicht wird, bei der der Speicherbedarf aller Variablen eines Blocks (einschließlich der Geisterzellen) die Größe des Prozessorcaches nicht übersteigt.

Die Ergebnisse der Messungen auf vier verschiedenen Plattformen sind in Abbildung 4.4 dargestellt. Es sind deutliche Unterschiede zwischen den Plattformen zu erkennen und keine Strategie kann sich über alle Plattformen hinweg als Leistungsstärkste durchsetzen. Die Strategie *Kopie*, welche auch die höchsten Flexibilität bietet, ist im Vergleich zu den Strategien *Zeiger* und *Anpassung* robuster, dass heißt auf keiner der untersuchten Plattform sind deutliche Leistungseinbußen zu beobachten. Auf drei von vier untersuchten Systemen bietet diese Strategie zudem eine bessere Leistung für Blöcke bis zur Größe von 128 Gitterzellen als die Referenzvariante mit *Geisterzellen*. Daher wird die Strategie *Kopie*, also das Kopieren des Blocks und der Ränder der Nachbarblöcke in ein kontinuierliches Datenfeld vor der eigentlichen Berechnung, für das Konzept der dynamischen Datenstrukturen gewählt.

4.2.3 Parallelisierung und Kommunikation

Um eine hohe Skalierbarkeit erreichen zu können, erfolgt die Parallelisierung der *FD4*-Datenstrukturen gemäß dem *message-passing*-Modell für Rechnersysteme mit verteiltem Speicher, dessen Nutzung mittels MPI auch bei parallelen atmosphärischen Simulationsmodellen weit verbreitet ist. Die Prozesse erhalten jeweils einen oder mehrere Blöcke, welche ihre jeweilige Partition darstellen. Die Partitionen sind untereinander disjunkt. Im *FD4*-Konzept sind drei verschiedene Arten der Kommunikation notwendig:

- Austausch von Variablen am Partitionsrand zwischen benachbarten Partitionen.
- Abstimmung zur dynamischen Lastbalancierung und Migration von Blöcken.
- Austausch von Variablen zur Kopplung mit dem atmosphärischen Modell.

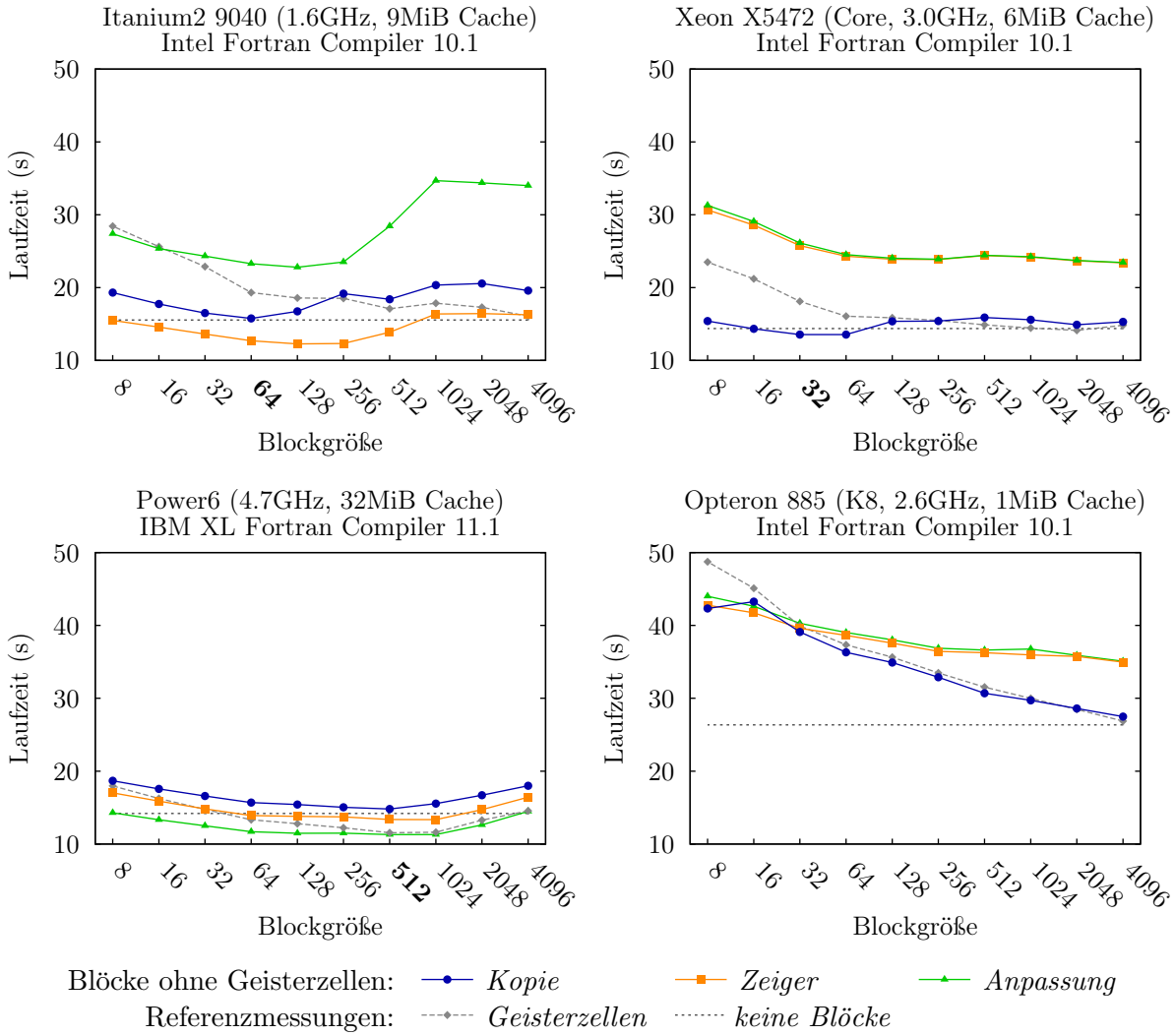
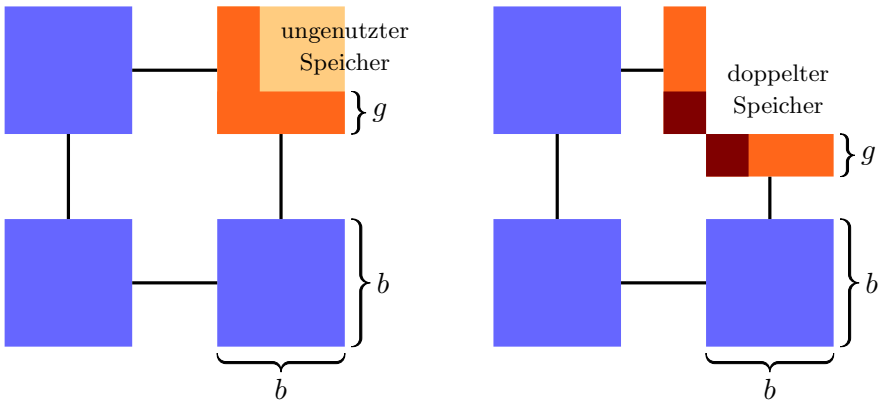


Abbildung 4.4: Vergleich drei verschiedener Zugriffsmethoden auf Variablen von Nachbarblöcken ohne Nutzung von Geisterzellen anhand eines synthetischen Benchmarks. Die fett gedruckte Blockgröße ist jeweils die größte, bei welcher alle Variablen eines Blocks einschließlich der Geisterzellen in den Prozessorcache passen. Der Opteron verfügt dazu über zu wenig Cache.

Dynamische Lastbalancierung und Kopplung werden in den folgenden Abschnitten 4.3 und 4.4 betrachtet. Die Kommunikation zwischen den Partitionsrändern ist Gegenstand dieses Abschnitts. Sie erfolgt mithilfe der bereits in Abschnitt 4.2.2 eingeführten *Geisterblöcke*. Diese ersetzen in dem dreidimensionalen Blockgitter die fehlenden Nachbarblöcke am Rand der Partition. In der Dimension, über die der Randaustausch stattfindet, müssen nur so viele Gitterzellen g übertragen werden, wie der Stencil des Algorithmus verlangt. Daher können die Geisterblöcke in dieser Dimension kleiner als die normalen Blöcke sein. Der umgekehrte Fall, dass g größer als die Kantenlänge b_d eines Blocks für eine der Dimensionen $d \in \{x, y, z\}$ ist, wird nicht betrachtet, da g bei atmosphärischen Modellen typischerweise nicht größer als drei ist. Benötigen zwei oder mehr Blöcke einer Partition Daten vom Rand des gleichen Blocks einer anderen Partition, so könnten sie entweder einen gemeinsamen Geisterblock anlegen oder jeweils einen eigenen. Beide Fälle sind in Abbildung 4.5 für den zweidimensionalen Fall dargestellt. Ein gemeinsamer Geisterblock ist bei Blöcken mit $b_d \approx g$ hinsichtlich der Speichernutzung effizienter. Bei Blöcken mit deutlich größerer Kantenlänge als g sind jedoch separate Geisterblöcke günstiger. Welche Methode effizienter ist, hängt neben der Blockgröße auch vom Anteil gemeinsamer Geisterblöcke



(a) Gemeinsamer Geisterblock **(b)** Getrennte Geisterblöcke

Abbildung 4.5: Illustration der Konzepte zur Behandlung von Geisterblöcken (orange) anhand eines zweidimensionalen Beispiels mit $b = 3\,g$.

ab – insbesondere da diese im Extremfall von bis zu sechs Blöcken einer Partition gemeinsam genutzt werden können.

Da der Austausch zwischen benachbarten Partitionen im Allgemeinen nicht für alle Variablen des Gitters notwendig ist, ist im Konzept die Spezifikation eines *Kommunikationskontextes* durch den Anwender vorgesehen. Darin werden die am Austausch beteiligten Variablen und der jeweilige Zeitschritt, falls mehrere zu einer Variablen gespeichert werden, definiert. Es können beliebig viele Kommunikationskontexte für eine *FD4*-Domain erstellt werden. Die genauen Kommunikationsbeziehungen, also von welchem Block Variablen an welche benachbarte Partition übertragen werden müssen, verändern sich im Laufe der Simulation durch die im Konzept vorgesehene dynamische Lastbalancierung. Unabhängig davon sollen die Daten mit so wenig wie möglich Nachrichten übertragen werden, um die Kommunikationslatenzen so weit wie möglich zu verbergen. Dass heißt, für jedes Paar kommunizierender Prozesse soll je Richtung nur eine Nachricht übertragen werden. Dies ist durch Aggregation der zu übertragenden Daten in einem Zwischenpuffer oder durch Nutzung abgeleiteter Datentypen, wie es der MPI-Standard erlaubt, möglich.

Durch die Abstraktion der Dekomposition in Blöcke, welche aus Sicht der Anwendung keine feste Reihenfolge besitzen, kann auch eine Überlappung von Kommunikation und Berechnung in transparenter Weise stattfinden. Nachdem der Austausch der Partitionsränder mithilfe nichtblockierender Nachrichten initiiert wurde, können Berechnungen bereits auf den inneren Blöcken der Partition durchgeführt werden, dass heißt auf Blöcken, bei denen keiner der sechs Nachbarblöcke einer anderen Partition angehört. Dies kann mithilfe des Iterators transparent geschehen, indem der Iterator zu Beginn über die inneren Blöcke läuft, danach wartet bis die Kommunikation beendet ist und dann schließlich über die äußeren Blöcke iteriert. Dieses Konzept lässt sich noch dahin gehend erweitern, dass der Iterator nur so lange wartet bis mindestens einer der äußeren Blöcke alle Daten für seine Geisterblöcke empfangen hat. Zu diesen Blöcken kann als nächstes iteriert werden. Dieser Zyklus aus Warten und Iterieren wird wiederholt, bis letztlich alle Daten empfangen wurden. Das Verfahren ist natürlich nur dann effektiv, wenn die Partitionen über innere Blöcke verfügen. Dies hängt von der Zahl der Blöcke je Prozess ab (es sind mindestens sieben nötig) und davon, ob das Partitionierungsverfahren Partitionen der notwendigen räumlichen Struktur erzeugt.

Zur Definition von Randbedingungen werden ebenfalls Geisterblöcke verwendet. Diese werden am Rand außerhalb des Gitters angelegt, müssen aber mit nutzerdefinierten Daten belegt werden. Dazu besteht die Möglichkeit über alle lokalen Geisterblöcke am Rand zu iterieren. Eine

Ausnahme sind periodische Randbedingungen. Hier werden die Geisterblöcke mit Variablen von Blöcken am gegenüberliegenden Rand des Gitters belegt. Da keine Unterscheidung der Nachbarschaftsbeziehung zwischen Nachbarn durch periodische Ränder und tatsächlichen räumlichen Nachbarn stattfindet, wird dieser Austausch ebenfalls durch den Kommunikationskontext durchgeführt. Befinden sich periodisch benachbarte Blöcke innerhalb der gleichen Partition, so wird die Nachbarschaftsbeziehung durch Zeiger im Blockgitter realisiert, so dass direkt auf die Daten zugegriffen werden kann. Aus Sicht der Anwendung ist es vollkommen transparent, ob ein Block am Rand oder in der Mitte des Gitters liegt und ob seine Nachbarblöcke lokale Blöcke, Geisterblöcke zur Kommunikation, oder Geisterblöcke für Randbedingungen sind.

4.3 Dynamische Lastbalancierung

Die Aufgabe der dynamischen Lastbalancierung ist es, die *FD4*-Blöcke möglichst gleichmäßig bezüglich der Rechenlast den Prozessen zuzuordnen und diese Zuordnung während der Laufzeit regelmäßig an die sich räumlich und zeitlich verändernde Last anzupassen. Durch die Abstraktion der Blöcke und des Blockiterators ist trotz veränderlicher Partitionierung der Zugriff auf die Daten über die gesamte Laufzeit einheitlich. Bei der Partitionierung sollen alle in Abschnitt 3.1.1 eingeführten Anforderungen für eine effiziente dynamische Lastbalancierung beachtet werden:

- Das Erreichen einer möglichst optimalen Lastbalance ist das wichtigste Ziel der dynamischen Lastbalancierung um so das Hauptdefizit der Skalierbarkeit spektraler Mikrophysik zu beseitigen. Keine der anderen Anforderungen sollte dazu führen, dass die Lastbalance stark beeinträchtigt wird.
- Die Reduktion der Kommunikation zwischen benachbarten Partitionen ist aufgrund des hohen Datenvolumens ebenfalls von hoher Bedeutung. Ein wichtiger Schritt für eine höhere Skalierbarkeit ist der in Abschnitt 4.1 motivierte Übergang von der zweidimensionalen zur dreidimensionalen Zerlegung. Das Partitionierungsverfahren soll die Blöcke derart den Partitionen zuordnen, dass die Kommunikation möglichst minimal ist.
- Wie in Abbildung 2.6 (Seite 19) gezeigt, wurde für die Mikrophysik eine hohe räumliche Variabilität der Rechenlast auf teilweise kurzen Zeitskalen festgestellt, so dass ein relativ häufiges Umverteilen nötig ist und der Aufwand der Migration ebenfalls sehr kritisch ist. Andererseits kann bei einem Modell mit verhältnismäßig hoher Rechenlast ein größerer Aufwand akzeptiert werden, um so eine höhere Optimierung für die Lastbalance zu erreichen. Daher sollte der Aufwand der Migration steuerbar sein.
- Für den Aufwand der Berechnung einer neuen Partitionierung gilt ähnliches wie für die Migration. Zudem sollte das Verfahren eine hohe Skalierbarkeit aufweisen.

4.3.1 Entwurf der dynamischen Lastbalancierung

Durch die hohen Lastschwankungen der spektralen Mikrophysik ist der Einsatz eines globalen Partitionierungsverfahrens notwendig. Ein lokales Verfahren wäre nur nach einer Vielzahl von Wiederholungen mit wechselnden Prozessgruppen in der Lage die Mikrophysik effektiv zu balancieren, was bezüglich der globalen Synchronisation ähnlich wie ein globales Verfahren wirkt. Die kurzen Zeitskalen, auf der die Laständerungen erfolgen, erfordern ein häufiges Prüfen der Lastbalance und gegebenenfalls Repartitionieren. Eine Balancierung während der kleinen Zeitschritte Δt^{mp} der Mikrophysik führt jedoch zu einer zusätzlichen Synchronisation der Prozesse, welche in dem nur durch die Kommunikation zwischen den Nachbarpartitionen sehr lose synchronisiertem Abschnitt nicht wünschenswert ist. Günstiger ist es, die Lastbalancierung höchstens zu jedem

Zeitschritt Δt^{dyn} des atmosphärischen Modells aufzurufen, da hier eine weitaus stärkere globale Synchronisation besteht. Mit der Häufigkeit des Aufrufens der Lastbalancierung wird sowohl der Aufwand des Repartitionierens als auch die erreichte Lastbalance beeinflusst. Zur Reduktion der Laufzeit müssen die Verluste durch Lastimbalance und die Kosten der Repartitionierung gegeneinander abgewogen werden. Beide Größen können durch Zeitmessung zur Laufzeit bestimmt und für die Abschätzung, ob ein Aufruf des Repartitionierungsverfahrens lohnenswert ist, herangezogen werden.

Dynamische Lastbalancierung basiert im Allgemeinen auf dem Wissen um die zukünftige Rechenlast eines Blocks (Gewicht). Eine exakte Vorhersage oder Abschätzung der künftigen Gewichte ist für die spektrale Mikrophysik schwierig, da – wie in Abschnitt 2.6.3 erläutert – die Rechenzeit in komplexer Weise von einer Vielzahl an Zustandsvariablen des Modells abhängt. Zudem besteht auch eine Abhängigkeit der Stärke der Rechenlastschwankungen von der konkreten Systemumgebung, wie dem Prozessor und Compiler. Grund sind die verschiedenen möglichen Ausführungspfade im Mikrophysikmodell, deren Algorithmen von unterschiedlichen Systemen jeweils unterschiedlich effizient abgearbeitet werden. Daher wird hier das Blockgewicht direkt aus der Rechenzeit des vergangenen Zeitschritts bestimmt.

Zur Erstellung der Partitionierung werden raumfüllende Kurven und graphenbasierte Verfahren in Betracht gezogen. Raumfüllende Kurven sind insbesondere hinsichtlich der Anforderungen nach geringen Migrationskosten (durch die zumeist lokale Migration zwischen Nachbarn auf der Kurve) und geringem Berechnungsaufwand sehr günstig. Zudem können mit hinreichend genauen Verfahren für die eindimensionale Partitionierung auf der Kurve sehr gute Werte für die Lastbalance erzielt werden. Die Stärke graphenbasierter Verfahren liegt dagegen in der expliziten Reduktion der Kommunikation zwischen Partitionen. Daher sollen beide Verfahren im Konzept berücksichtigt und schließlich mit Messungen verglichen werden. Die setzt voraus, dass die Lastbalancierung getrennt von den verteilten Datenstrukturen ist und dort keine Annahmen bezüglich des eingesetzten Partitionierungsverfahrens getroffen werden. Zur graphenbasierten Repartitionierung wird eine bestehende parallele Bibliothek genutzt.

Zur Identifikation der Kommunikationspartner beim Randaustausch zwischen benachbarten Partitionen, der Migration von Blöcken sowie der Kopplung werden Informationen über die Partitionen anderer Prozesse benötigt. Es ist daher sinnvoll, dass jeder Prozess eine Beschreibung der globalen Partitionierung speichert. Im allgemeinen Fall muss dafür je Position im Blockgitter der Identifikator des Prozesses gespeichert werden, welcher diesen Block besitzt (*Blockmap*). Bei der Partitionierung mit raumfüllenden Kurven kann die Partitionierung kompakter repräsentiert werden. Es genügt den *Partitionsvektor* s_p zu speichern, welcher je Prozess p dessen Startindex auf der Kurve enthält [81]. Allerdings besteht in diesem Fall ein höherer Aufwand bei der Abfrage der Information. Wenn aus der Position eines Blocks der besitzende Prozess bestimmt werden soll, muss zuerst der Index auf der Kurve berechnet werden. Dies weist eine logarithmische Komplexität in Bezug auf die Größe des Blockgitters auf. Danach muss im Partitionsvektor nach der entsprechenden Partition gesucht werden, was durch die aufsteigende Sortierung des Vektors mit binärer Suche beschleunigt werden kann und somit ebenfalls logarithmische Komplexität aufweist. Um die wiederholte Durchführung dieses Verfahrens für gleiche Blöcke zu vermeiden, können nach erfolgter Berechnung einer neuen Partitionierung gezielt nur jene Teile der Blockmap berechnet werden, welche tatsächlich benötigt werden. Dies betrifft im Einzelnen je Prozess:

1. die Blöcke der eigenen neuen Partition,
2. die neuen Besitzer der von der Partition entfernten Blöcke (für das Senden der Blöcke bei der Migration),

3. die vorigen Besitzer der neu zur Partition hinzugekommenen Blöcke (für das Empfangen neuer Blöcke bei der Migration),
4. die Besitzer der Nachbarblöcke der neuen Partition (für den Randaustausch zwischen benachbarten Partitionen) sowie
5. die Besitzer derjenigen Blöcke, welche sich mit lokalen Partitionen gekoppelter Modelle überschneiden (siehe Abschnitt 4.4).

Der Speicheraufwand dieser Partitionierungsinformation ist mit $O(B/P)$ skalierbar und sie kann mit einem Berechnungsaufwand von $O(log(B) + log(P))$ je Block bestimmt werden. Aus 3. geht hervor, dass sowohl der neu erzeugte als auch der vorige Partitionsvektor für die Berechnung der notwendigen Teile der Blockmap nötig ist. Somit müssen an globalen Informationen über die Partitionierung mindestens zwei Integerwerte je Prozess gespeichert werden. Im Sinne einer hohen Skalierbarkeit sollten dies die einzigen globalen Metadaten sein, wie es bereits im Bereich der hochskalierbaren Verwaltung adaptiver Gitterstrukturen realisiert wurde [13].

4.3.2 Partitionierung mit raumfüllenden Kurven

Hinsichtlich des Kommunikationsaufwands zwischen benachbarten Partitionen werden in der Literatur Hilbert-Kurven bevorzugt. Um den Vergleich verschiedener Kurventypen zu ermöglichen, soll jedoch keine Festlegung auf einen Typ erfolgen. Die Funktion, welche den Index eines Blocks auf der Kurve aus dessen Position im Blockgitter berechnet, soll daher austauschbar sein. Die notwendige Stufe l der Kurve wird aus der Größe des Blockgitters B_d je Dimension $d \in \{x, y, z\}$ mit $l = \lceil log_k(max(B_x, B_y, B_z)) \rceil$ bestimmt. Hierbei gibt k den Verfeinerungsfaktor an, der vom Typ der Kurve abhängt. Da die Kurve $(k^l)^n$ Punkte durchläuft, d. h. im zweidimensionalen Fall ein Quadrat und im dreidimensionalen Fall ein Würfel, wird sie bei typischen Blockgittern beschnitten. Die Eigenschaft der Hilbert-Kurve, in $(k^l)^n$ kontinuierlich zu sein, ist damit im Blockgitter nicht mehr gegeben, so dass die Partitionen aus mehreren, nicht zusammenhängenden Bereichen bestehen können.

Die Qualität der Partitionierung hinsichtlich der Lastbalance wird maßgeblich durch den eindimensionalen Partitionierungsalgorithmus bestimmt. Dieser zerlegt den Vektor von Blockgewichten w_i $(i = 1, 2, \ldots, B)$, welcher durch die raumfüllende Kurve entsteht, in so viele kontinuierliche Abschnitte, wie Partitionen erzeugt werden sollen. Miguet und Pierson [76] schlagen dafür zwei einfache Heuristiken vor. Ausgehend von den Präfixsummen der Blockgewichte $W_j = \sum_{i=1}^{j} w_i$ wird bei der ersten Heuristik *H1* (siehe Algorithmus A.2, Seite 99) der letzte Block des Prozesses p $(p = 1, 2, \ldots, P-1)$ durch den größten Index j_p bestimmt bei dem gilt:

$$W_{j_p} \leq p\,\beta^* \text{ mit } \beta^* = \frac{\sum_{i=1}^{B} w_i}{P} = \frac{W_B}{P}$$

Dabei ist β^* die durchschnittliche Last der Prozesse oder, anders ausgedrückt, der ideale *bottleneck value* für eine hypothetische, vollständig balancierte Partitionierung. Aus den jeweiligen j_p lässt sich mit der Beziehung $s_p = j_{p-1} + 1$ $(p = 2, 3, \ldots P)$ und $s_1 = 1$ der Partitionsvektor erstellen. Die zweite Heuristik *H2* (siehe Algorithmus A.3, Seite 100) verfeinert die Grenzen, indem j_p um eins erhöht wird, wenn $(W_{j_p+1} - p\,\beta^*) < (p\,\beta^* - W_{j_p})$, dass heißt, wenn die aufsummierte Last bis zum folgenden Block näher an der idealen Partitionsgrenze liegt als die des Blocks bei j_p. Diese Verfahren sind sehr schnell, jedoch erreichten sie bei stark variierenden Blockgewichten und hoher Partitionsanzahl bei weitem keine optimale Lastbalance [82]. Exakte Verfahren dagegen lösen das eindimensionale Partitionierungsproblem mit der optimalen Lastbalance. Sie sind jedoch in vielen Fällen weitaus aufwendiger in Bezug auf die Berechnungszeit [82].

Zur skalierbaren Lastbalancierung der Wolkenmikrophysik wird hier ein Verfahren vorgeschlagen, dessen Genauigkeit gesteuert werden kann. Damit ist es möglich, den Berechnungsaufwand gegenüber exakten Verfahren zu verringern und dennoch eine vorgegebene Lastbalance zu garantieren. Das Verfahren basiert auf dem Algorithmus ϵ-*BISECT* von Pınar und Aykanat [82]. Dabei wird eine binäre Suche nach einem möglichst minimalen *bottleneck value* β_{res}, der Summe der Gewichte der maximal belasteten Partition, durchgeführt. Der Partitionsvektor kann anschließend leicht aus β_{res} bestimmt werden, indem in einer Iteration über den Gewichtsvektor den Prozessen sukzessive Abschnitte mit einer Gesamtlast von höchstens β_{res} zugewiesen werden – analog der Heuristik *H1* mit der veränderten Bedingung $W_{j_p} \leq W_{j_{p-1}} + \beta_{res}$ mit $W_{j_0} = 0$ (siehe Algorithmus A.4, Seite 100). Mit einer ähnlichen Methode kann auch festgestellt werden, ob eine Partitionierung für einen vorgegebenen *bottleneck value* β existiert, indem anschließend geprüft wird, ob die Last der letzten Partition höchstens β ist, dass heißt ob $W_B - W_{j_{P-1}} \leq \beta$ gilt. Dieser Algorithmus wird als *PROBE* während der binären Suche nach β_{res} in ϵ-*BISECT* verwendet (siehe Algorithmus A.5, Seite 100). Die Suche findet über dem Intervall

$$I = \left[max(\beta^*, w_{max}), \beta^* + w_{max}\right]$$

statt. Im Fall, dass es kein Blockgewicht größer als die durchschnittliche Last der Partitionen β^* gibt, entspricht die untere Grenze der perfekten Lastbalance. Die obere Grenze wird durch das maximale Blockgewicht beschränkt. Mit *PROBE* wird bei der binären Suche entschieden, ob die obere Grenze auf die Mitte des Intervalls verschoben wird (bei Erfolg von *PROBE*) oder die untere Grenze (bei Misserfolg). Auf diese Weise nähern sich beide Grenzen immer mehr dem tatsächlichen minimalen β_{opt} an, bis die Differenz der Grenzen kleiner oder gleich einer vorgegebenen Genauigkeit ϵ ist. Somit gilt für den erreichten *bottleneck value* $\beta_{res} \leq \beta_{opt}+\epsilon$. Die Komplexität des ursprünglichen Verfahrens von Pınar und Aykanat ist $O(B + P\, log(B/P)\, log(w_{max}/\epsilon))$ und wird durch die initiale Präfixsumme der Blockgewichte mit $O(B)$ und $log(w_{max}/\epsilon)$ Wiederholungen von *PROBE* mit je $O(P\, log(B/P))$ bestimmt. Mit der Genauigkeit ϵ kann der Aufwand des Verfahrens beeinflusst werden.

Das in dieser Arbeit vorgeschlagene Verfahren erweitert ϵ-*BISECT* um die Bestimmung von ϵ aus dem geforderten Verhältnis $r < 1$ der erzielten Lastbalance Λ_{res} zur maximal möglichen Lastbalance Λ_{opt}. Aus der Definition der Lastbalance in Abschnitt 3.1.1

$$\Lambda = \beta^*/\beta \sim \beta^{-1} \tag{4.4}$$

folgt damit $r = \beta_{opt}/\beta_{res}$. Somit entspricht r dem Qualitätsfaktor einer Partitionierung wie ihn Miguet und Pierson [76] definieren. Aus dem Zusammenhang $\beta_{res} - \beta_{opt} \leq \epsilon$ lässt sich nun formulieren:

$$\epsilon \geq \beta_{res} - \beta_{opt} = (1 - r)\, \beta_{res}$$

Wird nun für β_{res} Gleichung 4.4 eingesetzt, so erhält man:

$$\epsilon \geq (1 - r)\, \frac{\beta^*}{\Lambda_{res}}$$

Die einzige Unbekannte der rechten Seite, die Lastbalance der resultierenden Partitionierung Λ_{res}, lässt sich mit $\Lambda_{res} \leq r$ abschätzen, so dass gilt:

$$\epsilon \geq \frac{1 - r}{r}\, \beta^* = \frac{1 - r}{r}\, \frac{W_B}{P} \tag{4.5}$$

Mit Gleichung 4.5 ist nun eine Abschätzung der Genauigkeit des Algorithmus ϵ-*BISECT* aus dem geforderten Verhältnis r zur maximalen Lastbalance möglich.

Pınar und Aykanat [82] zeigen mit den Algorithmus *EXACT-BISECT* wie ϵ-*BISECT* in ein exaktes Verfahren umgewandelt werden kann. Dieser Algorithmus erstellt für $\epsilon = 0$ eine Partitionierung mit $\beta_{res} = \beta_{opt}$, dass heißt mit der optimalen Lastbalance des Gewichtsvektors. Dazu

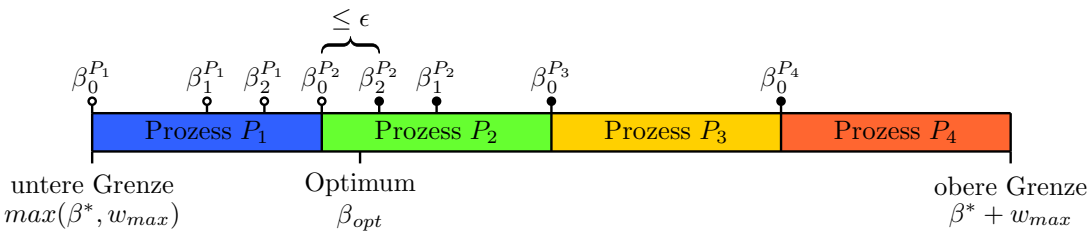


Abbildung 4.6: Illustration der Suche nach dem minimalen gültigen *bottleneck value* des parallelen eindimensionalen Partitionierungsverfahrens *r-PAR-BISECT* bei vier Prozessen. Ein erfolgreicher Aufruf von *PROBE+* ist mit einem ausgefüllten, ein Misserfolg mit einem nicht ausgefüllten Punkt gekennzeichnet. Prozess P_2 findet hier den besten *bottleneck value* $\beta_{res} = \beta_2^{P_2}$ mit $\beta_{res} \leq \beta_{opt} + \epsilon$.

werden bei der binären Suche nach dem minimalen *bottleneck value* die Grenzen des Intervalls auf tatsächlich erreichbare Werte gesetzt und nicht auf solche, die in der Mitte des vorigen Intervalls liegen. Diese Werte werden durch eine Erweiterung des *PROBE*-Algorithmus bestimmt (*PROBE+*, siehe Algorithmus A.6, Seite 101). Da nur endlich viele *bottleneck values* existieren, ist der Suchraum somit ebenfalls endlich und die binäre Suche terminiert auch für $\epsilon = 0$. Wird diese Methode mit der obigen Erweiterung zur Bestimmung von ϵ kombiniert, so erhält man ein Verfahren, das eine beliebige Genauigkeit erzielen kann. Dabei ist das Verfahren für $r = 1$ exakt und für $0 < r < 1$ eine Näherung vorgegebener Genauigkeit. Dieses Verfahren, im Weiteren als *r-BISECT* bezeichnet, ist in Algorithmus A.7 auf Seite 102 dargestellt.

Abgesehen von der Möglichkeit die Präfixsumme des Gewichtsvektors parallel durchzuführen, ist das beschriebene Verfahren vollständig seriell. Eine einfache Möglichkeit der Parallelisierung von *r-BISECT* besteht darin, den summierten Gewichtsvektor auf alle Prozesse zu verteilen und den Prozessen gleich große, disjunkte Teilintervalle I_p mit $I = \cup_{p=1}^{P} I_p$ zur Suche nach dem minimalen *bottleneck value* zuzuweisen. Um die Korrektheit der Abbruchbedingung (Differenz der Grenzen $\leq \epsilon$) zu gewährleisten, führt jeder Prozess zuerst *PROBE+* für seine untere Grenze aus. Nur wenn dies keine gültige Partitionierung ergibt, wird das binäre Suchverfahren, beginnend in der Mitte des jeweiligen Teilintervalls, angewandt. Die Aufteilung des Suchintervalls I und die binäre Suche sind in Abbildung 4.6 dargestellt. Anschließend wird das globale Minimum der gefundenen gültigen *bottleneck values* bestimmt und daraus der Partitionsvektor ermittelt. Dieses Verfahren wird im Folgenden mit *r-PAR-BISECT* bezeichnet (siehe Algorithmus A.8, Seite 102). Durch die kürzeren Teilintervalle wird die Abbruchbedingung nach weniger Iterationen des Suchverfahrens erfüllt als beim seriellen Verfahren *r-BISECT*. Dabei wird die Zahl der Iterationen bei P Prozessen um maximal $\lfloor log_2(P) \rfloor$ reduziert. Dass die Reduktion auch geringer ausfallen kann, ist wie folgt begründet: Durch das Setzen der Intervallgrenzen der binären Suche auf tatsächlich erreichbare *bottleneck values* reduziert sich die Länge der Suchintervalle geringfügig schneller als bei normaler binärer Suche. Daher kann *r-BISECT* nach $\lfloor log_2(P) \rfloor$ Iterationen bereits ein kleineres Suchintervall erreicht haben als die Länge des Startintervalls der P Prozesse bei *r-PAR-BISECT* beträgt. Für $r < 1$ ist es aber auch möglich, dass mit *r-PAR-BISECT* keine binäre Suche nötig ist, da bereits die Länge der Teilintervalle I_p kleiner oder gleich ϵ ist, dass heißt wenn

$$\frac{\beta^* + w_{max} - max(\beta^*, w_{max})}{P} = \frac{min(\beta^*, w_{max})}{P} \leq \epsilon \tag{4.6}$$

gilt. Wird nun zur Abschätzung von ϵ die Beziehung $\lambda^{-1} = B\, w_{max}/W_B$ und Gleichung 4.5 eingesetzt sowie zusätzlich die (zum Erreichen hoher Lastbalance notwendige) Bedingung $\beta^* > w_{max}$ angenommen, so kann daraus formuliert werden:

$$B \geq \frac{r}{1-r}\,\lambda^{-1}$$

Diese Gleichung formuliert aus der Imbalance der Blockgewichte λ^{-1}, der Blockanzahl B und dem gefordertem Qualitätsfaktor r eine hinreichende Bedingung, unter der bei *r-PAR-BISECT* keine binäre Suche nötig ist. Sie ist insbesondere unabhängig von der Prozessanzahl.

4.3.3 Hochskalierbare Partitionierung mit raumfüllenden Kurven

Graphenpartitionierung ist vergleichsweise aufwendig und die parallelen Verfahren skalieren bisher nicht auf mehrere tausend Prozesse [7, 41]. Für eine hochskalierende Lastbalancierung sind daher Methoden basierend auf raumfüllenden Kurven durch die schnellere Berechnung weitaus Erfolg versprechender. Durch die reguläre Struktur kann zudem die gesamte Partitionierung wesentlich kompakter beschrieben werden, so dass weniger globale Metadaten gespeichert müssen (siehe Abschnitt 4.3.1). Der vorgeschlagene eindimensionale Partitionierungsalgorithmus *r-PAR-BISECT* verlangt jedoch, dass die Gewichte aller Blöcke an alle Prozesse verteilt und gespeichert werden. Nur durch die Sicht auf alle Blockgewichte wird mit *r-PAR-BISECT* eine hohe Lastbalance erreicht. Bei einer hohen Anzahl an Prozessen und entsprechend hoher Blockanzahl stellt dies jedoch einen hohen Kommunikationsaufwand und somit wesentlichen Skalierbarkeitsengpass der dynamischen Lastbalancierung dar. Diese Metadaten müssen für hochskalierende Anwendungen unter den Prozessen aufgeteilt bleiben, was einen verteilten Partitionierungsalgorithmus voraussetzt.

Gut zur Parallelisierung eignen sich einfache Heuristiken. Dazu schlagen Miguet und Pierson [76] vor, die Präfixsumme der Blockgewichte parallel durchzuführen. Danach sucht jeder Prozess auf seinem Teil des summierten Gewichtsvektors nach Partitionsgrenzen und sendet diese an die entsprechenden Prozesse. Jedoch besteht hier der bereits angesprochene Nachteil einer unzureichenden Qualität der erzeugten Partitionierung in Bezug auf Lastbalance bei hoher Prozessanzahl. Die Heuristiken können jedoch als erster Schritt für ein zweistufiges, hierarchisches Verfahren dienen, welches im Folgenden vorgeschlagen wird. Für Partitionsprobleme, bei denen das maximale Blockgewicht deutlich kleiner als die durchschnittliche Last einer Partition ist ($max(w_i) \ll \beta^*$), konnten Miguet und Pierson [76] zeigen, dass die Heuristiken nahezu optimale Partitionierungen erzeugen. Wenn diese Bedingung nicht gilt, kann β^* erhöht werden, indem eine Zerlegung in $Q < P$ Partitionen erzeugt wird. Dass heißt, mithilfe einer solchen Heuristik wird als erster Schritt eine grobgranulare Partitionierung hoher Qualität auf vollständig parallele Weise erstellt. Die Q grobgranularen Partitionen müssen anschließend in jeweils P/Q Teile zerlegt werden, welche in ihrer Gesamtheit schließlich die gewünschte Zerlegung in P Partitionen bilden. [1] Für diesen zweiten Schritt wird das oben eingeführte Verfahren *r-PAR-BISECT* eingesetzt, von dem parallel Q Instanzen zugleich laufen. Dabei muss nicht der globale Gewichtsvektor eingesammelt werden, sondern nur der Teil, welcher der jeweiligen grobgranularen Partition von P/Q Prozessen entspricht. Dieses Verfahren wird im Folgenden mit *r-DISTR-BISECT* bezeichnet. Der Ablauf, einschließlich der notwendigen Kommunikation basierend auf MPI, stellt sich im Einzelnen wie folgt dar:

1. *Präfixsumme der Blockgewichte und Ermittlung der Gesamtlast*: Um den lokalen Teil der Präfixsumme der Blockgewichte W_j zu berechnen, wird eine parallele, exklusive Präfixsumme der Gesamtlast $\sum_{i=s_p}^{s_{p+1}-1} w_i$ jedes Prozesses $p = 1, 2, \ldots, P$ durchgeführt. Diese Operation wird von MPI mit `MPI_Exscan` zur Verfügung gestellt. Im Ergebnis erhält dabei jeder Prozess p die aufsummierte Last der Prozesse 1 bis $p-1$, dass heißt $W_{s_p-1} = \sum_{i=1}^{s_p-1} w_i$. Basierend darauf kann der lokale Teil der (globalen) Präfixsumme der Blockgewichte mit $W_j = W_{j-1} + w_j$ für $j = s_p, s_p+1, \ldots, s_{p+1}-1$ berechnet werden. Werden Gleitkommazahlen zur Repräsentation der Blockgewichte verwendet, so ist es anschließend notwendig, dass

[1] Im Folgenden wird zu Vereinfachung der Beschreibung von einem ganzzahligem P/Q ausgegangen, wobei dies grundsätzlich keine Voraussetzung für das Verfahren ist.

jeder Prozess $p > 1$ sein Resultat der parallelen Präfixsumme W_{s_p-1} an den Prozess $p-1$ sendet, da dieser diese Variable auf andere Weise berechnet. Durch Gleitkommaarithmetik verursachte Abweichungen können anderenfalls im weiteren Verlauf des Verfahrens zu einer Blockierung führen. Die zur Erstellung der grobgranularen Partitionierung benötigte aufsummierte Last aller Blöcke $W_B = \sum_{i=1}^{B} w_i$ ist im Prozess $p = P$ verfügbar. Diese teilt er allen anderen Prozessen mit einer `MPI_Bcast`-Operation mit.

2. *Erstellung der grobgranularen Partitionierung*: Jeder Prozess prüft, ob und wo sich in seinen aufsummierten Gewichtsvektor W_j, $j = s_p - 1, s_p, \ldots, s_{p+1} - 1$ eine oder mehrere der $Q-1$ Startindizes der grobgranularen Partitionierung u_q, $q = 2, 3, \ldots, Q$ befinden. Dafür wird die Heuristik *H1* von Miguet und Pierson [76] genutzt, wobei als durchschnittliche Last $\beta^* = W_B/Q$ verwendet wird.

3. *Kommunikation der grobgranularen Partitionsgrenzen an die Gruppen*: Für jede gefundene Partitionsgrenze u_q sendet der Finder die entsprechende Position an den ersten Prozess der beiden anliegenden grobgranularen Partitionen. Diese sind durch die feste Aufteilung in Q Gruppen zu je P/Q aufeinanderfolgenden Prozessen bekannt. Der erste Prozess jeder Partition empfängt die zwei Partitionsgrenzen (ausgenommen in der ersten und letzten Gruppe, wo nur eine Grenze empfangen wird) und verteilt diese und den Identifikator der Finder-Prozesse an alle Prozesse der entsprechenden Gruppe mittels `MPI_Bcast`. Somit ist nun allen Prozessen die Position der eigenen grobgranularen Partition innerhalb des Gewichtsvektors bekannt.

4. *Verteilung der aufsummierten Gewichtsvektoren innerhalb der Gruppen*: Die Prozesse senden alle W_j derjenigen Blöcke, welche einer anderen Gruppe als der eigenen zugeordnet wurden, an den im Prozessvektor naheliegendsten Prozess dieser Gruppe (der erste oder letzte Prozess der Gruppe). Zusätzlich muss dabei auch die Präfixsumme für den Block vor der grobgranularen Partitionierung W_{u_q-1} übertragen werden, um das Blockgewicht w_{u_q} des ersten Blocks dieser Partition ermitteln zu können. Den entsprechenden Empfängern sind die Sender bekannt, da diese aus die im vorherigen Schritt an alle Gruppenmitglieder gesendeten Finder-Prozesse grober Partitionsgrenzen leicht ermittelt werden können. Die Anzahl der so übertragenen Punkt-zu-Punkt-Nachrichten hängt davon ab, wie weit sich die grobgranularen Partitionsgrenzen gegenüber der vorhergegangenen Partitionierung verschoben haben. Liegen die Grenzen immer bei einem Block eines direkt an der Gruppengrenze gelegenen Prozesses, so wird je grobgranularer Partitionsgrenze nur eine Nachricht benötigt. Im Ergebnis besitzen die Prozesse am Rand einer Gruppe gegebenenfalls zusätzliche W_j von Blöcken, welche nicht in ihrer aktuellen Partition, jedoch in der neuen grobgranularen Partition der Gruppe vorhanden sind. Anschließend werden mittels einer kollektiven Kommunikationsoperation innerhalb jeder Gruppe q alle W_j, $j = u_q - 1, u_q, \ldots, u_{q+1} - 1$ der neuen grobgranularen Partition an alle P/Q Gruppenmitglieder verteilt. Hierfür wird die Operation `MPI_Allgatherv` verwendet, von der parallel zueinander Q Instanzen ablaufen. Zur Vorbereitung des nächsten Schrittes werden die globalen Präfixsummen W_j in (bzgl. Gruppe q) lokale Präfixsummen W_i^q umgewandelt, indem W_{u_q-1} von allen Werten subtrahiert wird, dass heißt $W_i^q = W_{i+u_q-1} - W_{u_q-1}$, $i = 1, 2, \ldots, u_{q+1} - u_q$ und $W_0^q = 0$.

5. *Erstellung der Partitionierungen innerhalb der Gruppen*: Mit *r-PAR-BISECT* wird für jede Gruppe unabhängig voneinander eine Partitionierung hoher Qualität erstellt. Als Eingabe wird die Präfixsumme der Blockgewichte W_j^q der Gruppe q und die durchschnittliche Last $\beta^* = W^q_{u_{q+1}-1} Q/P$ verwendet. Als einzige Kommunikation wird zur Ermittlung des minimalen gefundenen *bottleneck value* β_{res} eine `MPI_Allreduce`-Operation innerhalb jeder Gruppe benötigt.

6. *Verteilung der finalen Partitionierung an alle Prozesse*: Um den neuen, globalen Partitionsvektor s'_p in jedem Prozess zu erstellen, verteilt jeder Prozess den Startindex seiner Partition mithilfe von `MPI_Allgather` an alle Prozesse.

Das Verfahren wird im Detail in Anhang A.3 auf Seite 103 dargestellt. Mit *r-DISTR-BISECT* wird zwar keine vollständige Parallelisierung der Partitionsberechnung erreicht, aber die Blockgewichte werden nur innerhalb der Q Prozessgruppen ausgetauscht, was eine deutliche Reduzierung des Kommunikationsaufwands und der Menge an Metadaten bewirkt. Der Speicheraufwand für Blockgewichte hängt von Q ab und ist somit nicht direkt von der Prozessanzahl abhängig.

4.4 Modellkopplung

Die Aufgabe der Modellkopplung im Rahmen des *FD4*-Konzepts ist die Übertragung von Modellvariablen zwischen den Gittern des atmosphärischen Modells und des Wolkenmikrophysikmodells. Die Hauptaufgabe ist es, das M×N-Problem zu lösen, das heißt, Daten zwischen den unterschiedlichen Partitionierungen zu übertragen. Die hohe Komplexität erwächst dabei aus zwei Anforderungen: Zum einen wird eine dynamische Lastbalancierung für das Mikrophysikmodell eingesetzt, so dass regelmäßiges *handshaking* zwischen den Partitionierungen notwendig ist. Zum anderen ist das Ziel der Arbeit, eine hochskalierbare und effiziente Abarbeitung des Modellsystems zu ermöglichen, was beim Entwurf der Kopplung Beachtung finden muss.

Die räumliche und zeitliche Transformation von Daten wird nicht innerhalb dieses Konzepts betrachtet. Beide Transformationen können unabhängig von der Datenübertragung realisiert werden. Die räumliche Transformation müsste in diesem Fall auf Seiten des gekoppelten Modells stattfinden, welches Kopplungsdaten entsprechend der Rechteckstruktur des *FD4*-Gitters sendet bzw. empfängt. In bisherigen Ansätzen der Kopplung detaillierter Wolkenmodelle mit atmosphärischen Modellen sind beide Gitter von identischer Struktur, so dass diese Transformation der Daten nicht notwendig ist. Die zeitliche Interpolation von Variablen kann leicht auf Anwendungsebene realisiert werden.

Wie in Abschnitt 4.1 dargelegt, sieht das *FD4*-Konzept die sequentielle Abarbeitung der parallelen Modelle vor. Jeder Prozess berechnet jeweils abwechselnd eine Partition des atmosphärischen Modells und eine des Mikrophysikmodells. Zwar stellt diese Methode höhere Anforderungen an die Skalierbarkeit der einzelnen Modelle, dafür können jedoch keine Lastimbalancen zwischen den Modellen auftreten. Zudem ist das Kopplungsschema von COSMO-SPECS darauf angewiesen, dass beide Modelle nacheinander abgearbeitet werden. Durch die enge wechselseitige Kopplung bestehen zeitliche Datenabhängigkeiten, welche keine Nebenläufigkeit zulassen. Wie in Abschnitt 2.6.1 erläutert, fließen die Zustandsvariablen von COSMO zum Ende eines Kopplungsintervalls Δt^{dyn} direkt in die Berechnung des gleichen Zeitschritts von SPECS ein. Umgekehrt benötigt COSMO zur Berechnung des folgenden Zeitschritts die Ausgabe des vorigen Schritts von SPECS. Die sequentielle Kopplung hat zur Folge, dass jeder Prozess eine *FD4*-Partition und eine Partition des gekoppelten Modells besitzt. Somit muss das M×N-Problem für jeweils so viele Partitionen der beiden Modelle gelöst werden muss, wie parallel Prozesse existieren. Die dazu notwendige Speicherung der Metadaten über die Partitionierung und das Identifizieren von Überschneidungen von Partitionen wird im Folgenden für den globalen Ansatz beschrieben. Dass heißt, jeder Prozess speichert Kopplungsmetadaten jeder Partition. In dem darauf folgenden Abschnitten 4.4.2 und 4.4.3 werden diese Konzepte auf die verteilte Speicherung der Metadaten erweitert.

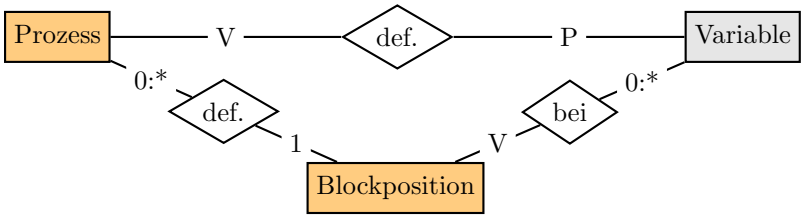
4.4.1 Kopplung mit globalen Metadaten

Die Partitionierung des Gitters der spektralen Mikrophysik ist durch die enge Verknüpfung mit der Lastbalancierung in einem gemeinsamen Konzept auch für die Kopplung bekannt. Daher muss diese Partitionierung nicht zusätzlich für die Kopplung beschrieben werden. Die Datenfelder und die Partitionierung des gekoppelten atmosphärischen Modells müssen jedoch in geeigneter Weise definiert werden. Viele atmosphärische Modelle verwenden eine schachbrettartige Partitionierung der horizontalen Dimensionen in M×N Teilgebiete. Ausnahmen sind MM5, bei dem mithilfe der unabhängigen Bibliothek RSL [73] eine Partitionierung in nicht rechteckige Teile vorgenommen werden kann, und das Chemie-Transport-Modell MUSCAT [121], bei dem eine blockbasierte Dekomposition erfolgt. Um eine hohe Flexibilität und Anwendbarkeit des Konzepts auch in Bereichen außerhalb der atmosphärischen Simulation zu ermöglichen, werden möglichst wenige Annahmen über die Methode der Partitionierung getroffen. Die einzigen Annahmen sind, dass jede Partition durch einen oder mehrere quaderförmige Blöcke des dreidimensionalen Gitters beschrieben werden kann und dass sich diese Partitionierung über die Laufzeit nicht ändert. Jeder dieser Blöcke zur Kopplung mit der *FD4*-Datenstruktur wird im Folgenden als *Koppelfeld* bezeichnet. Dabei ist ein *lokales Koppelfeld* eines Prozesses ein Koppelfeld, welches Teil der gekoppelten Partition dieses Prozesses ist. Dass heißt, dieser Prozess ist der Besitzer dieses Koppelfelds. Für das in diesem Abschnitt vorgestellte Konzept der globalen Verwaltung der Kopplungsmetadaten werden die Beschreibungen aller Koppelfelder auf allen Prozessen vollständig repliziert. Auf diese Weise ist sichergestellt, dass die notwendigen Metadaten auch nach jeder Umverteilung der Blöcke durch dynamische Lastbalancierung vorhanden sind. Da bei der Modellkopplung mit allen Prozessen des gekoppelten Modells die gleichen Variablen ausgetauscht werden, können diese unabhängig von der Beschreibung der Koppelfelder gespeichert werden. Die Metadaten lassen sich somit in Metadaten für Variablen und Metadaten für die gekoppelte Partitionierung einteilen. Letztere existieren für jeden Prozess, wohingegen die Variablen nur einmal definiert werden müssen. Abbildung 4.7(a) stellt ein Datenmodell zur Beschreibung blockbasierter Partitionierungen dar. Jedem Prozess können hier beliebig viele Koppelfelder zugeordnet werden. Wie auch bei COSMO-SPECS besteht bei vielen atmosphärischen Modellen jede Partition aus einem Block. In diesem Fall wird jedem Prozess immer genau eine Blockposition zugeordnet, wie Abbildung 4.7(b) als Datenmodell darstellt. Um eine hohe Flexibilität zu erreichen, soll die Datenstruktur zur Speicherung der Kopplungsmetadaten beide Datenmodelle jeweils speichereffizient realisieren.

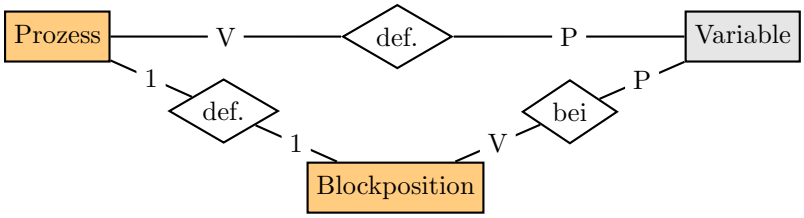
Die Übertragung von Variablen zwischen den Partitionierungen des gekoppelten Modells und der dynamisch lastbalancierten Datenstrukturen kann in beide Richtungen erfolgen: Die Operation *PUT* überträgt Daten von Koppelfeldern in die dynamischen *FD4*-Datenstrukturen während *GET* Daten aus den dynamischen Datenstrukturen in Koppelfelder überträgt. Vor der Datenübertragung wird mit dem *handshaking* festgestellt, welche Prozesse miteinander kommunizieren und von welchen Teilen ihrer Partition Daten gesendet bzw. empfangen werden. Durch die sequentielle Kopplung muss bei *PUT* und *GET* jeder Prozess folgende zwei Überschneidungen identifizieren:

1. Aus der Sicht des gekoppelten Modells: Bestimmung der Überschneidungen lokaler Koppelfelder mit allen Blöcken der *FD4*-Datenstruktur und Bestimmung der Prozesse, welche die jeweiligen *FD4*-Blöcke besitzen. Bei *PUT* betrifft dies den Sender, bei *GET* den Empfänger.

2. Aus der Sicht der *FD4*-Datenstruktur: Bestimmung der Überschneidungen lokaler *FD4*-Blöcke mit allen Koppelfeldern und Bestimmung der Prozesse, welche die jeweiligen Koppelfelder besitzen. Bei *PUT* betrifft dies den Empfänger, bei *GET* den Sender.



(a) Beliebig viele Blöcke je Prozess



(b) Genau ein Block je Prozess

Abbildung 4.7: Datenmodelle zur Beschreibung der Koppelfelder für verschiedene Grade an Flexibilität. Farbig hervorgehobene Objekte sind solche, deren Anzahl proportional mit der Prozessanzahl P steigt. V steht für die Anzahl der zu übertragenden Variablen.

Naive Suchalgorithmen für beide Probleme sind folglich von der Komplexität $O(KB/P)$. Dabei steht B für die Gesamtzahl von *FD4*-Blöcken, K für die Gesamtzahl von Koppelfeldern und P für die Anzahl der Prozesse und es wird vereinfachend davon ausgegangen, dass sowohl die Koppelfelder als auch die Blöcke gleichmäßig über die Prozesse verteilt sind. Da üblicherweise $K \sim P$ gilt kann die Komplexität als $O(B)$ zusammengefasst werden. Dass heißt, der Aufwand skaliert nicht mit steigender Prozessanzahl. Im Gegenteil steigt der Aufwand sogar linear, wenn die Anzahl der Blöcke proportional zur Prozessanzahl ist. Damit sind Verbesserungen notwendig, um das Ziel einer hochskalierbaren Kopplung zu erreichen.

Die Überschneidungen aus Sicht des gekoppelten Modells lassen sich effizient bestimmen, indem nur genau jene Blöcke der globalen Beschreibung der dynamischen Partitionierung untersucht werden, welche sich mit den lokalen Koppelfeldern überschneiden. Da sich weder die Lage der Blöcke noch die Lage der Koppelfelder im Gitter ändert, müssen die in Frage kommenden Blöcke nur einmal nach der Definition aller Koppelfelder (und nicht vor jedem *handshaking* nach geänderter Partitionierung) identifiziert werden. Die Zahl der zu untersuchenden Blöcke je Prozess ist somit direkt abhängig von der Partitionsgröße des gekoppelten Modells und nimmt bei steigender Prozessanzahl ab, so dass die Skalierbarkeit des Verfahrens gegeben ist. Für die Komplexität gilt mit diesem Verfahren $O(B/P)$.

Die Identifikation der Überschneidungen aus Sicht der *FD4*-Datenstruktur kann optimiert werden, indem die quaderförmige Hülle der Koppelfelder jedes einzelnen Prozesses berechnet wird. Diese Hülle ist der kleinste quaderförmige Gitterausschnitt welcher alle Koppelfelder eines Prozesses enthält. Falls je Prozess nur ein Koppelfeld definiert wird, ist diese Hülle sofort verfügbar. Nach der Definition der Koppelfelder wird für alle Blöcke des Gitters die Überschneidung mit den Hüllen der Koppelfelder jedes Prozesses bestimmt. Liegt eine Überschneidung vor, so wird für den Block der entsprechende Prozess vermerkt. Beim eigentlichen *handshaking* müssen dann nur die Überschneidungen für die Koppelfelder der je lokalem Block gespeicherten Prozesse ermittelt werden. Der Aufwand hierfür ist mit $O(B/P)$ skalierbar.

Zur Kopplung zweier Modelle werden verschiedene Ausführungen der Kopplungskommunikation mit jeweils unterschiedlichen zu übertragenden Datenfeldern und unterschiedlicher Richtung

(*PUT* oder *GET*) benötigt. Dies betrifft zum einen die bidirektionale Kopplung zur Laufzeit des Modellsystems. Der Datentransfer findet für jede Richtung zu jeweils unterschiedlichen Zeitpunkten des Programmablaufs für jeweils verschiedene Datenfelder statt. Zum anderen ist zu Beginn für die Initialisierung der Variablen je nach Modellsystem ein einmaliger Datentransfer notwendig. Überdies kann die Ein- und Ausgabe mithilfe des Datentransfers der Kopplung realisiert werden, da so die Daten in eine für parallele Ein-/Ausgabe optimierte Dekomposition gebracht werden können. Um solche verschiedenen Ausführungen abzubilden, wird das Konzept des *Koppelkontextes* eingeführt. Dieser beschreibt die Partitionierung eines gekoppelten Modells und die zu übertragenden Variablen entsprechend der oben eingeführten Datenmodelle (siehe Abbildung 4.7). Verschiedene Koppelkontexte dienen der Beschreibung unterschiedlicher Arten des Datenaustauschs. Zudem können so zugleich verschiedene Modelle gekoppelt werden, so dass Modellsysteme mit mehr als zwei Modellen möglich sind. Wenn mehrere Koppelkontexte auf der gleichen Partitionierung basieren, wie es für den Datenaustausch mit einem gekoppelten Modell der Fall ist, so werden diese Beschreibungen von den Koppelkontexten gemeinsam genutzt, um Speicherplatz einzusparen. Dies betrifft in den Datenmodellen die Blockpositionen und Prozesse.

Die eigentliche Datenübertragung findet für *PUT* und *GET* jeweils in einer direkten Nachricht je miteinander kommunizierendem Prozesspaar statt. Dafür werden die Daten aller sich überschneidenden Bereiche der Koppelfelder des einen Prozesses und der *FD4*-Blöcke des anderen Prozesses mithilfe abgeleiteter MPI-Datentypen zu einer Nachricht gebündelt. Auf diese Weise wird der Einfluss der Latenz des Verbindungsnetzwerks auf die effektive Bandbreite reduziert. Dies ist im Kontext der Kopplung besonders wichtig, da die Kommunikation der Abbildung zweier unabhängiger Partitionierungen entspricht und daher typischerweise ein irreguläres Muster aufweist. Die dabei erstellten MPI-Datentypen speichern das Resultat des *handshaking* jedes Koppelkontextes und werden bei aufeinanderfolgenden Aufrufen der Kopplungskommunikation wiederverwendet. Nur wenn die Partitionierung der *FD4*-Datenstrukturen sich verändert, muss das *handshaking* erneut durchgeführt und die MPI-Datentypen neu aufgebaut werden.

4.4.2 Dekomposition der Metadaten zur Kopplung

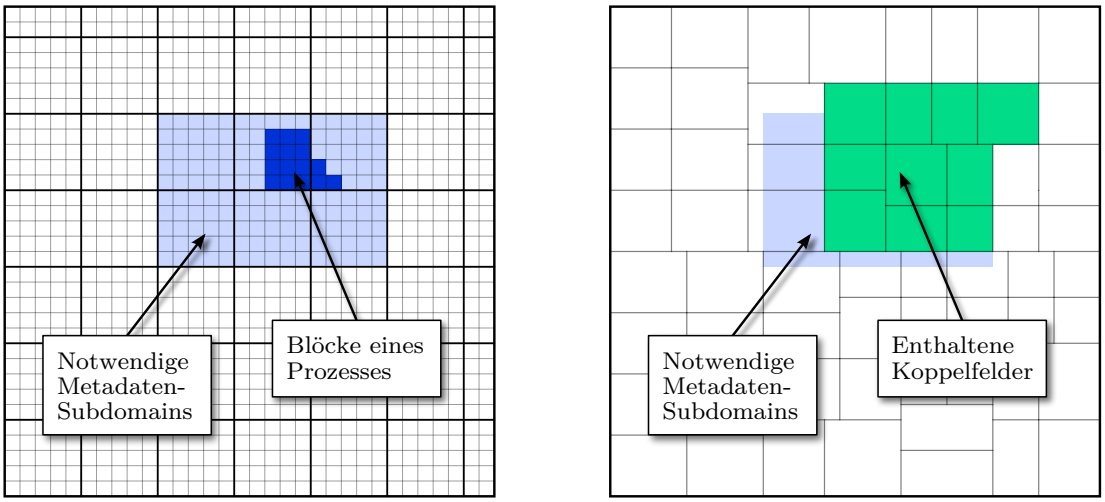
Die globale Verwaltung der Kopplungsmetadaten führt zu einem mit der Prozessanzahl steigenden Speicherbedarf je Prozess. Im oben beschriebenen Konzept sind bei der kompakten Repräsentation der globalen Metadaten zwei nicht skalierbare Metadatenstrukturen vorhanden: die Lage der Koppelfelder aller Prozesse sowie die für alle *FD4*-Blöcke vermerkten Prozesse, deren Koppelfelder sich mit dem jeweiligen Block überschneiden. Für eine hochskalierbare Kopplung ist es daher notwendig, die Metadaten der Kopplung über die Prozesse zu verteilen, so dass jeder Prozess nur die für ihn relevanten Metadaten speichert. Bei der Lastbalancierung der dynamischen Datenstruktur müssen dann zusätzlich auch Metadaten übertragen werden. Unabhängig von den verteilten Metadaten werden zusätzlich die Beschreibungen der lokalen Koppelfelder gespeichert. In diesem Abschnitt wird die Dekomposition der Metadaten beschrieben. Der folgende Abschnitt 4.4.3 widmet sich der dynamischen Umverteilung der Metadaten.

Eine Methode der Dekomposition ist, die Metadaten in den *FD4*-Blöcken zu speichern. Dies bedeutet, dass in der Datenstruktur eines Blocks die Beschreibungen derjenigen Koppelfelder abgelegt werden, welche sich mit dem Block überschneiden. Die Umverteilung wird so bei der Migration der Blöcke realisiert. Ein Nachteil ist die Redundanz der Metadaten, da die Blöcke von feinerer Granularität sind. Die Zahl der Partitionen des gekoppelten Modells entspricht bei der sequentiellen Kopplung der Zahl der Partitionen der dynamischen Datenstruktur, deren Blockanzahl für eine effektive Lastbalancierung deutlich höher ist. In Abschnitt 4.2.2 wurde in diesem Kontext eine mindestens um den Faktor 10 höhere Blockanzahl gegenüber der Zahl der Prozesse für die spektrale Mikrophysik motiviert. Folglich würden in diesem Fall die Beschreibungen der Partitionen des gekoppelten Modells auf durchschnittlich 10 Blöcken repliziert.

Um die Replikation der Block-gebundenen Metadaten zu vermeiden werden die Metadaten in einer unabhängigen, verteilten Datenstruktur mit einer gröberen Dekomposition verwaltet. Die dabei entstehenden Teilgebiete für Metadaten werden im Folgenden als *Metadaten-Subdomains* bezeichnet. Das Rechteckgitter der *FD4*-Blockdekomposition wird in allen drei Raumdimensionen in quaderförmige Metadaten-Subdomains unterteilt. Dass heißt, eine feste Zahl von *FD4*-Blöcken wird immer genau einer Metadaten-Subdomain zugeordnet. Nur den in den drei Raumdimensionen jeweils letzten Metadaten-Subdomains können weniger Blöcke zugeordnet sein, um so beliebige Blockgittergrößen zu ermöglichen. Die Dekomposition der Metadaten wird somit sehr kompakt allein durch die Blockgittergröße und die Größe einer Metadaten-Subdomain repräsentiert. Die Metadaten-Subdomains werden durch ihre Position im Rechteckgitter der Metadaten-Subdomains als $M_{x,y,z}$ mit $x, y, z \geq 1$ eindeutig identifiziert und können somit zum schnellen Zugriff in einem dreidimensionalen Datenfeld organisiert werden. In einer Metadaten-Subdomain werden alle Koppelfelder beschrieben, die innerhalb des von ihr abgedeckten Gitterausschnitts *beginnen*, dass heißt deren erste Gitterzelle bezüglich der drei Raumdimensionen in dieser Metadaten-Subdomain liegt. Damit ist eine Replikation der Koppelfelder in mehreren Metadaten-Subdomains ausgeschlossen. Als Bedingung für die Größe der Metadaten-Subdomains wird gefordert, dass sie in jeder Raumdimension mindestens so groß sind, wie die maximale Größe eines Koppelfelds reduziert um eine Gitterzelle. Auf diese Weise befinden sich die für einen beliebigen Punkt im Gitter relevanten Koppelfelder garantiert in der Metadaten-Subdomain $M_{x,y,z}$ an diesem Punkt und den sieben benachbarten Metadaten-Subdomains die im Gitter direkt und diagonal vor $M_{x,y,z}$ liegen (für $x = 1$, $y = 1$ oder $z = 1$ werden in der entsprechenden Dimension keine benachbarten Metadaten-Subdomains benötigt). Dass heißt, es sind die Metadaten-Subdomains $M_{i,j,k}$ für alle $i \geq 1$, $j \geq 1$, $k \geq 1$ in $i \in \{x-1, x\}$, $j \in \{y-1, y\}$, $k \in \{z-1, z\}$ erforderlich. Die Position dieser Metadaten-Subdomains ist aufgrund der einfachen Darstellung der Metadatendekomposition mit konstantem Aufwand zu berechnen. Liegt darüber hinaus der Sonderfall vor, dass alle Koppelfelder jeweils vollständig innerhalb einer Metadaten-Subdomain liegen, so ist die Einbeziehung der sieben benachbarten Metadaten-Subdomains nicht notwendig. Abbildung 4.8(a) stellt die notwendigen Metadaten-Subdomains für eine beispielhafte Partition dar. Zu beachten ist, dass – anders als bei den Blöcken – eine Metadaten-Subdomain mehr als einem Prozess zugeordnet sein kann. Die in den hervorgehobenen Metadaten-Subdomains enthaltenen Koppelfelder einer gekoppelten Partitionierung sind in Abbildung 4.8(b) illustriert.

Durch die Dekomposition der Metadaten wird das *handshaking* aus Sicht der *FD4*-Datenstruktur nun folgendermaßen verändert: Die im vorigen Abschnitt beschriebene Optimierung, bei der für alle Blöcke des Gitters diejenigen Prozesse vermerkt werden, die diesen Block überschneidende Koppelfelder besitzen, kann entfallen. Stattdessen werden nach oben beschriebener Methode für jeden lokalen Block die jeweils maximal acht Metadaten-Subdomains bestimmt, die die relevanten Koppelfelder enthalten. Nur für diese Koppelfelder muss jeweils die Lage der Überschneidung ermittelt werden, so dass je lokalem Block ein konstanter Aufwand entsteht. Somit ist die Skalierbarkeit des Verfahrens bei einem Gesamtaufwand von $O(B/P)$ je Prozess gegeben. Das *handshaking* aus Sicht des gekoppelten Modells wird durch die verteilten Metadaten nicht verändert.

Wie in Abschnitt 4.4.1 erläutert, können mithilfe des Koppelkontextes mehrere Partitionierungen für die Kopplung definiert werden. Die Definitionen der entsprechenden Koppelfelder werden für jede Partitionierung getrennt in den Metadaten-Subdomains abgelegt. Basieren mehrere Koppelkontexte auf der gleichen Partitionierung, so können sie diese Beschreibung teilen. Abbildung 4.9 fasst die Datenstrukturen zur Dekomposition der Metadaten und das Konzept des Koppelkontextes zusammen. Jede Metadaten-Subdomain kann eine bestimmte Anzahl von Partitionierungen speichern, welche jeweils durch zwei Integer-Datenfelder beschrieben werden. Ein Datenfeld speichert die Lage eines Koppelfelds anhand der zwei Gitterkoordinaten, welche das Koppelfeld aufspannen. Das zweite Feld listet die Identifikatoren der Prozesse auf, welche die Koppelfelder besitzen und indiziert jeweils den Eintrag des ersten Koppelfelds eines Prozesses.



(a) Blöcke und notwendige Metadaten-Subdomains eines Prozesses

(b) In den Metadaten-Subdomains des Prozesses enthaltene Koppelfelder

Abbildung 4.8: Illustration des Konzepts der Metadaten-Subdomains anhand eines zweidimensionalen Beispiels. Das Gitter ist in 32×32 Blöcke unterteilt und die Größe der Metadaten-Subdomains beträgt 5×5 Blöcke. Die gekoppelte Partitionierung wird durch 64 inhomogene Koppelfelder beschrieben.

Ein Koppelkontext enthält den Index der Partitionierung innerhalb der Metadaten-Subdomains, wodurch mehrere Koppelkontexte die gleiche Partitionierung teilen können. Jeder Koppelkontext enthält zudem seine eigene Definition an Variablen für die Kopplung, welche jeweils aus dem *FD4*-Variablenindex und dem Zeitschritt besteht. Des Weiteren werden die Lage der lokalen Koppelfelder und Zeiger zu den eigentlichen Datenfeldern gespeichert.

4.4.3 Dynamisch verteilte Metadaten zur Kopplung

Durch die Dekomposition der Metadaten mithilfe der Metadaten-Subdomains kann das Konzept auf die verteilte Verwaltung erweitert werden. Dabei muss jeder Prozess nur die Metadaten-Subdomains speichern, welche die relevanten Koppelfelder für die lokalen *FD4*-Blöcke enthalten, welche leicht nach der in Abschnitt 4.4.2 beschriebenen Methode bestimmt werden können. Die maximale Anzahl vorhandener Metadaten-Subdomains ist somit proportional zur Zahl lokaler Blöcke. Die Speicherung der Metadaten-Subdomains in einem dreidimensionalen Feld ist in diesem Fall ineffizient, da mit zunehmender Parallelität ein immer kleinerer Teil aller Metadaten-Subdomains benötigt wird. Daher werden bei der verteilten Verwaltung die Metadaten-Subdomains in einem balancierten Binärbaum abgelegt. Der eindeutige Schlüssel kann leicht aus der Position $M_{x,y,z}$ einer Metadaten-Subdomain erzeugt werden.

Bei der Repartitionierung des *FD4*-Gitters müssen neben Blöcken gegebenenfalls auch Metadaten-Subdomains migriert werden. Fehlen einem Prozess für eine neue Partition Metadaten-Subdomains, so erhält er diese von den Prozessen, welche ihm Blöcke übertragen haben. Umgekehrt kann ein Prozess nicht mehr benötigte Metadaten-Subdomains entfernen. Um die Häufigkeit der Migration zu reduzieren, wird ein Zwischenpuffer für nicht benötigte Metadaten-Subdomains eingeführt, wofür ebenfalls ein balancierter Binärbaum dient. Erst wenn eine bestimmte Anzahl gepufferter Metadaten-Subdomains überschritten ist, werden die am längsten nicht mehr benötigten entfernt. Wird eine Partitionierung basierend auf raumfüllenden Kurven verwendet,

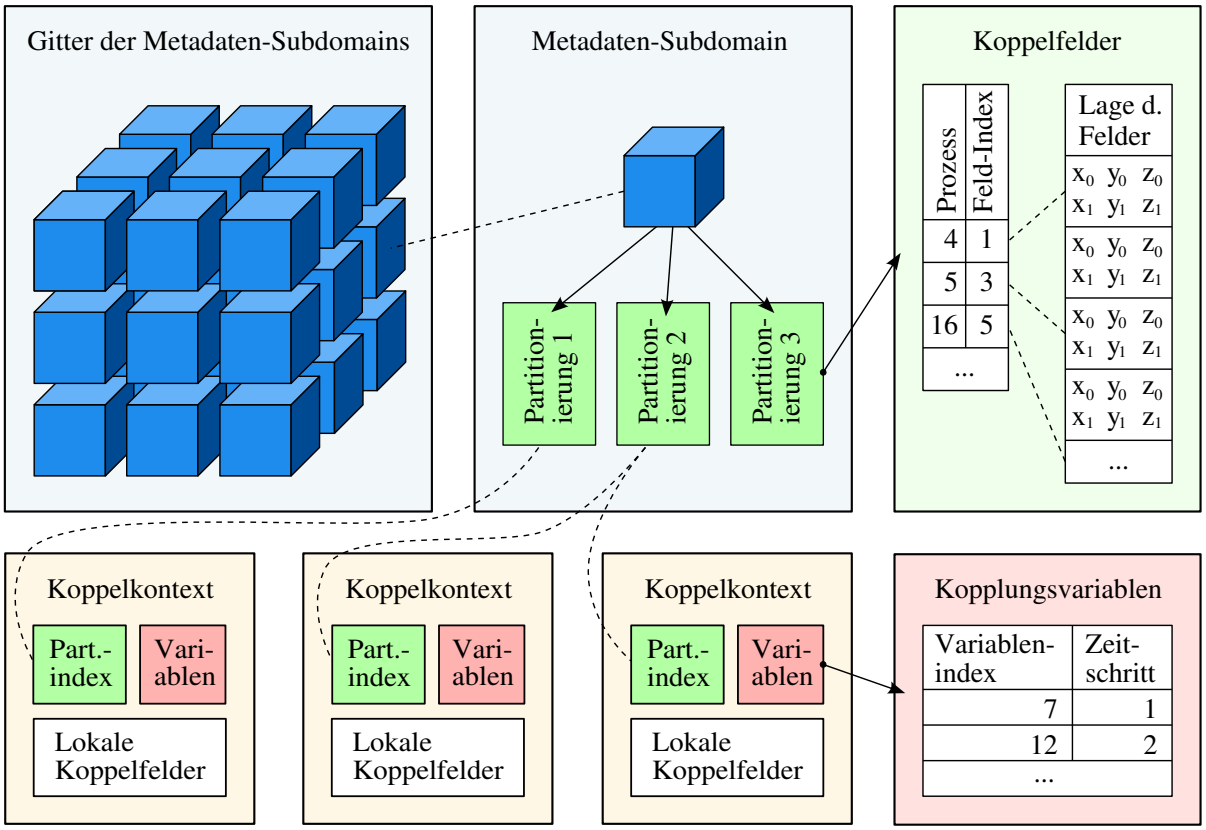


Abbildung 4.9: Darstellung der Datenstrukturen zur Dekomposition der Metadaten und des Konzepts des Koppelkontextes.

so liegen diese Metadaten-Subdomains auf der Kurve am weitesten von der aktuellen Partition entfernt und werden daher von allen gepufferten Metadaten-Subdomains mit der geringsten Wahrscheinlichkeit in Zukunft wieder benötigt. Da alle Prozesse die Metadaten-Subdomains individuell puffern, ist bei der Migration nicht immer bekannt, ob zusätzlich zu den Blöcken auch Metadaten-Subdomains übertragen werden müssen. Daher findet vor der Migration der Blöcke und Metadaten-Subdomains eine Abstimmung statt. Jeder Prozess führt dazu folgende Kommunikationsoperationen durch:

- Denjenigen Prozessen, von denen Blöcke empfangen werden, werden die Schlüssel der benötigten Metadaten-Subdomains gesendet. Dabei wird zuvor sichergestellt, dass sich diese Metadaten-Subdomains nicht im lokalen Zwischenpuffer befinden.
- Von denjenigen Prozessen, an die Blöcke gesendet werden, werden die Schlüssel der angeforderten Metadaten-Subdomains empfangen.

Die Metadaten-Subdomains, welche ein Prozess garantiert besitzt und daher übertragen kann, ergeben sich sofort aus den Sendern von Blöcken bei der Blockmigration, dass heißt aus der alten Partitionierung. Diese zusätzlichen Nachrichten werden nur zwischen Partnern bei der Blockmigration ausgetauscht und die Nachrichtenlänge ist im Vergleich zur Blockmigration sehr gering. Daher ist von einem relativ geringen Aufwand für diese Abstimmung zu rechnen. Die eigentliche Übertragung der Metadaten-Subdomains erfolgt in der gleichen Nachricht, mit der alle zwischen zwei Prozessen migrierten Blöcke übertragen werden.

5 Resultate

In diesem Kapitel wird kurz die Implementierung der Konzepte des vorigen Kapitels in einer unabhängigen Bibliothek erläutert und auf deren Nutzung zur lastbalancierten und hochskalierbaren Kopplung von COSMO und SPECS eingegangen. Anschließend werden in Abschnitt 5.2 Ergebnisse von Messungen präsentiert und diskutiert, welche zur Bewertung und zum Vergleich der Konzepte dienen.

5.1 Implementierung der Konzepte

Zur Evaluierung des Konzeptes der lastbalancierten Kopplung wurde die eigenständige Bibliothek FD4 [59, 62] in Fortran 95 implementiert, welche *message-passing* mittels MPI 2.0 realisiert. Die Bibliothek ist unabhängig vom konkret eingesetzten Simulationsmodell. Folgende Teile des Konzepts wurden in FD4 implementiert:

- Dreidimensionale Blockdekomposition, Variablentabelle, Domain, Block-Pool und Blockiterator (vgl. Abschnitt 4.2.1).
- Optimierung für eine feingranulare Zerlegung durch Geisterblöcke; Zugriff auf benachbarte Blöcke durch Kopieren in temporäres Feld (vgl. Abschnitt 4.2.2).
- Austausch von Variablen zwischen benachbarten Partitionen mittels Kommunikationskontext, jedoch ohne gemeinsame Geisterblöcke und ohne Überlappung von Kommunikation und Berechnung (vgl. Abschnitt 4.2.3).
- Dynamische Lastbalancierung basierend auf raumfüllenden Kurven (Hilbert und Morton) mit den Algorithmen *H1*, *H2*, *r-BISECT*, *r-PAR-BISECT* und *r-DISTR-BISECT* (vgl. Abschnitt 4.3 und Anhang A).
- Dynamische Lastbalancierung durch die parallele Graphenpartitionierungsbibliothek ParMetis [95], welche im Relation zu vergleichbaren Werkzeugen typischerweise eine geringere Berechnungszeit benötigt [14, 17].
- Bidirektionale Modellkopplung mit den Operationen *PUT* und *GET* sowie Koppelkontext (vgl. Abschnitt 4.4.1).
- Dekomposition der Kopplungsmetadaten in Metadaten-Subdomains, jedoch keine dynamisch verteilten Metadaten (vgl. Abschnitt 4.4.2).

5.1.1 Ablauf der dynamischen Lastbalancierung in FD4

Die dynamische Lastbalancierung ist ein zentrales Element der Arbeit und wird in den Messungen in Abschnitt 5.2 genauer bewertet. Daher wird ihr Ablauf in der Implementierung FD4 hier näher erläutert. Die Lastbalancierung in FD4 besteht aus drei Hauptphasen: (1) Synchronisation und Abstimmung, ob eine Repartitionierung nötig ist, (2) Berechnung einer neuen Partitionierung sowie (3) Migration der FD4-Blöcke. Der zeitliche Ablauf dieser Phasen ist in Abbildung 5.1

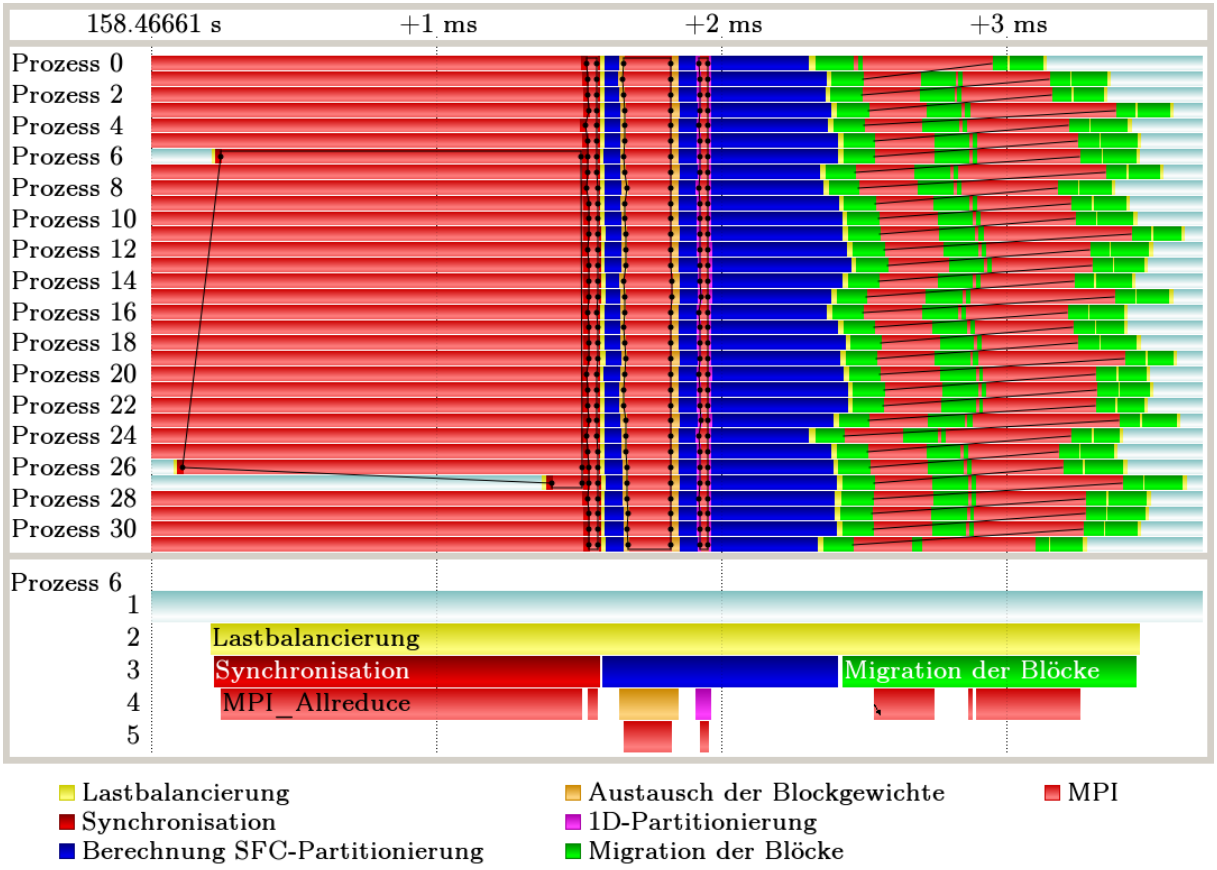


Abbildung 5.1: Darstellung des Ablaufs der FD4-Lastbalancierung anhand einer *Timeline* für 32 Prozesse (oben) und im Detail für einen Prozess anhand des Callstacks (unten). Die Farben entsprechen einzelnen Phasen der Lastbalancierung. Punkt-zu-Punkt-Nachrichten sind durch einfache Linien, kollektive Kommunikationsoperationen durch Linien mit Punkten dargestellt.

anhand einer beispielhaften *Timeline* des Analysewerkzeugs Vampir [54, 80] dargestellt. Im Folgenden werden die drei Phasen genauer beschrieben:

1. *Synchronisation und Abstimmung, ob eine Repartitionierung nötig ist*: In der ersten Phase wird die Lastbalance entsprechend der aktuellen Blockgewichte bestimmt. Die Blockgewichte werden von der Anwendung gesetzt und an FD4 übergeben. Die Last eines Prozesses wird durch die Summe seiner Blockgewichte gebildet. Mit zwei aufeinanderfolgenden `MPI_Allreduce`-Operationen wird das globale Maximum sowie die Summe der Last aller Prozesse bestimmt, woraus entsprechend Formel 3.1 die Lastbalance berechnet wird. Liegt diese über einem vorgegebenen Sollwert wird keine neue Partitionierung berechnet und die Lastbalancierung somit beendet. Aufgrund der globalen Reduktionsoperationen findet eine vollständige Synchronisation der Prozesse statt, wodurch der Zeitbedarf dieser Phase maßgeblich von der Balance der Prozesse vor dem Aufruf der Lastbalancierung abhängt, was auch in Abbildung 5.1 deutlich wird.

2. *Berechnung einer neuen Partitionierung*: Die zweite Phase ist je nach gewähltem Partitionierungsverfahren unterschiedlich:

   - Bei Partitionierung mittels raumfüllender Kurven mit den Verfahren *r-BISECT* oder *r-PAR-BISECT* werden die Gewichte der lokalen Blöcke, sortiert entsprechend der Kurve, an alle anderen Prozesse verteilt. Dafür wird `MPI_Allgatherv` genutzt. Im Ergebnis besitzt jeder Prozess den vollständigen, nach der raumfüllenden Kurve

sortierten Gewichtsvektor. Nach Berechnung der Präfixsumme des Gewichtsvektors wird mit einem eindimensionalen Partitionierungsverfahren der neue Partitionsvektor berechnet, welcher je Prozess dessen Startindex auf der Kurve enthält. In Abbildung 5.1 ist das Verfahren *r-PAR-BISECT* dargestellt, welches mit einer zusätzlichen `MPI_Allreduce`-Operation den minimalen gefundenen *bottleneck value* ermittelt. Anschließend werden aus dem Partitionsvektor notwendige Teile der Blockmap berechnet, dass heißt, die Besitzer derjenigen Blöcke welche für Randaustausch, Migration und Kopplung benötigt werden (siehe Abschnitt 4.3.1). In FD4 wird die Blockmap in einem dreidimensionalen Datenfeld gespeichert, dessen Größe von der Anzahl aller Blöcke des Gitters abhängt.

- Für die Partitionierung mit raumfüllenden Kurven mit dem zweistufigen hierarchischen Verfahren *r-DISTR-BISECT* wird entsprechend der Beschreibung in Abschnitt 4.3.3 vorgegangen, wodurch die Verteilung aller Blockgewichte an alle Prozesse sowie die serielle Berechnung der Präfixsumme entfällt. Anschließend werden auch hier nur die notwendigen Teile der Blockmap berechnet.
- Bei der Partitionierung mittels ParMetis wird der lokale Teil jedes Prozesses des aus den Blöcken gebildeten Graphen erstellt. Dieser Graph, sowie die Knotengewichte (Rechenlast der Blöcke), die Kantengewichte (Volumen des Randaustauschs zwischen Nachbarblöcken) und die Migrationskosten der Knoten (Blockgröße) bilden die Argumente der Funktion `ParMETIS_V3_AdaptiveRepart` von ParMetis. Die zurückgegebene lokale Partitionierung wird mittels `MPI_Allgatherv` allen anderen Prozessen mitgeteilt und daraus die Blockmap für alle Blöcke berechnet.

3. *Migration der Blöcke*: In der dritten Phase werden die Blöcke an die neuen Besitzer übertragen. Jeder lokale Block enthält einen abgeleiteten MPI-Datentyp für dessen Variablen. Durch Aggregation aller Datentypen derjenigen Blöcke, welche zum bzw. vom gleichen Prozess migriert werden müssen, findet die Migration mit genau einer Punkt-zu-Punkt-Nachricht je kommunizierendem Prozesspaar statt. Dabei werden zuerst die Nachrichten nichtblockierend gesendet, danach neue Blöcke angelegt oder, falls möglich, dem Block-Pool entnommen und dann alle migrierten Blöcke empfangen. Zum Schluss werden die nicht mehr benötigten Blöcke und Geisterblöcke entfernt bzw. dem Block-Pool hinzugefügt und neue Geisterblöcke angelegt bzw. dem Block-Pool entnommen. Bei Partitionierungen basierend auf raumfüllenden Kurven findet die Blockmigration typischerweise zwischen benachbarten Prozessen statt. Dieses Kommunikationsmuster ist in Abbildung 5.1 deutlich erkennbar.

Mit der Vorgabe des Sollwerts für die Lastbalance ist es möglich, die Häufigkeit der Repartitionierung zu steuern. Somit kann der Aufwand der Lastbalancierung auf Kosten der Lastbalance reduziert oder umgekehrt die Lastbalance durch häufigeres Repartitionieren verbessert werden. Da jedoch der optimale Sollwert a priori nicht bekannt ist und zudem über die Laufzeit durch verändertes Verhalten der Lastschwankung variieren kann, wurde von Watts und Taylor [118] vorgeschlagen, die Kosten und den Gewinn der Lastbalancierung explizit für die Bestimmung des Zeitpunkts der Repartitionierung heranzuziehen. In FD4 wurde darauf basierend ein Modus implementiert, bei dem die Bibliothek zur Laufzeit selbst abschätzt, ob eine Repartitionierung lohnenswert ist. Dazu wird die Zeit, welche durch die gemessene Imbalance gegenüber der besten Lastbalance nach den letzten n Repartitionierungsaufrufen eingebüßt wurde, gegen die Zeit, welche die letzten n Repartitionierungen minimal benötigt haben, abgewogen. Die Zeit für die Repartitionierung wird durch die Zeit für Partitionsberechnung und Blockmigration, jedoch ohne die Zeit zur Synchronisation der Prozesse bestimmt. Eine Repartitionierung wird somit nur durchgeführt, wenn sich der Aufwand voraussichtlich lohnen wird.

5.1.2 Kopplung von COSMO und SPECS mit FD4

Basierend auf der Bibliothek FD4 wurde die Kopplung zwischen COSMO und SPECS neu implementiert und damit das Modellsystem COSMO-SPECS+FD4 erschaffen [59, 60]. Die umfangreichen Daten des spektralen Mikrophysikmodells SPECS werden hierbei vollständig von FD4 verwaltet und parallelisiert. Auf diesen Datenstrukturen finden die Berechnungen von SPECS und Advektion der Hydrometeore statt. Der konkrete Ablauf des gekoppelten Modellsystems für einen Zeitschritt Δt^{dyn} von COSMO stellt sich gemäß dem Konzept aus Abschnitt 4.1 wie folgt dar:

1. Zeitintegration des atmosphärischen Modells COSMO für Δt^{dyn} auf einer statischen, zweidimensionalen Partitionierung des Gitters.
2. Korrektur des Windfelds $\mathbf{v}$ auf der Partitionierung von COSMO, um Massenerhaltung zu garantieren.
3. Übertragung der Kopplungsdaten in die FD4-Datenstrukturen mit der Operation *PUT* von FD4.
4. Berechnungen auf den FD4-Blöcken für m kleine Zeitschritte $\Delta t^{mp} = \Delta t^{dyn} / m$.
    1. Austausch der Ränder zwischen benachbarten FD4-Partitionen mittels FD4-Kommunikationskontext.
    2. Zeitintegration der Advektion der Hydrometeore und des Wasserdampfs für Δt^{mp}.
    3. Zeitintegration des spektralen Mikrophysikmodells SPECS für Δt^{mp}.
5. Aufruf der dynamischen Lastbalancierung von FD4.
6. Übertragung der Kopplungsdaten in die Partitionierung von COSMO mit der Operation *GET* von FD4.

Die Blockgewichte, welche die Basis für die Lastbalancierung darstellen, werden durch Zeitmessung gewonnen. Dafür wird für jeden Block die benötigt Rechenzeit für Advektion und Mikrophysik über die m kleinen Zeitschritte aufsummiert.

Aus dem oben skizzierten Ablauf sind zwei notwendige Koppelkontexte ersichtlich, welche zur Kopplung vor und nach den Berechnungen auf den FD4-Datenstrukturen benötigt werden. Zusätzlich werden zwei weitere Koppelkontexte ausschließlich für die Initialisierung der Mikrophysik benötigt: Zum einen, um Konstanten wie die Gitterzellengröße an die FD4-Datenstrukturen zu übermitteln, und zum anderen, um nach der Initialisierung der spektralen Variablen die über die Größenklassen aufsummierten Mischungsverhältnisse der Hydrometeore an COSMO zu übertragen. Alle vier Koppelkontexte teilen eine gemeinsame Partitionierung des gekoppelten Modells, so dass nur eine Partitionierung in den Metadaten-Subdomains gespeichert wird.

Das neue Modellsystem COSMO-SPECS+FD4 führt dieselben Berechnungen wie das ursprüngliche COSMO-SPECS durch. Abgesehen von geringen Abweichungen aufgrund der Gleitkommaarithmetik erzeugen beide Versionen des Modellsystems die gleichen Simulationsergebnisse. Diese Abweichungen entstehen durch die geänderte Anordnung der Schleifen über die Gitterzellen bei der Advektionsberechnung. Dadurch werden die die Beiträge der sechs Nachbarzellen zum Nettofluss einer Zelle in einer anderen Reihenfolge aufsummiert. Bedingt durch Auslöschungs- und Absorptionseffekte kann es hierbei zu abweichenden Ergebnissen kommen.

5.2 Bewertung anhand von Benchmarks

Die Leistungsmessungen der implementierten Konzepte wurden auf zwei verschiedenen Hochleistungsrechnern durchgeführt, einer SGI Altix 4700 und einer IBM BlueGene/P. Die SGI Altix 4700 „Mars“ besteht aus 1024 Intel Itanium 2 9040 Doppelkernprozessoren mit 9 MiB Level-3-Cache je Prozessorkern und 1,6 GHz Taktfrequenz. Ein Prozessorkern erreicht eine theoretische Spitzenleistung von 6,4 GFLOPS. Die Knoten mit je einen Prozessor sind mit einem *fat-tree*-Netzwerk des Typs *SGI NUMAlink 4* gekoppelt, welches bis zu 6,4 GB/s Bandbreite je Anschluss bietet. Dabei wird der lokale Speicher von bis zu 256 Prozessoren logisch zu einem gemeinsamen Adressraum von 2 TiB zusammengefasst (*virtual shared memory*). Eine solche Einheit (Partition) wird von einem Betriebssystem verwaltet.

Die IBM BlueGene/P „JUGENE“ ist ein massiv paralleles *distributed-memory*-System mit 73 728 Knoten. Jeder Knoten verfügt über einen IBM PowerPC 450 Vierkernprozessor mit 8 MiB gemeinsamen Level-3-Cache und 850 MHz Taktfrequenz sowie 2 GiB Hauptspeicher. Ein Prozessorkern erreicht eine theoretische Spitzenleistung von 3,4 GFLOPS. Die Knoten sind mit zwei spezialisierten Netzwerken für die Interprozesskommunikation verbunden [103]: ein dreidimensionaler Torus mit maximal 425 MB/s Bandbreite je Verbindung für Punkt-zu-Punkt-Nachrichten und ein Binärbaum mit maximal 850 MB/s Bandbreite je Verbindung für globale kollektive Operationen. Darüber hinaus steht ein weiteres dediziertes Netzwerk für globale Barrieren zur Verfügung. Aufgrund der hohen Parallelität des Systems mit insgesamt 294 912 Prozessorkernen wurde die IBM BlueGene/P für diese Arbeit zur Untersuchung der Hochskalierbarkeit eingesetzt.

5.2.1 Vergleich eindimensionaler Partitionierungsalgorithmen

Die folgenden Messungen dienen zum Vergleich der in Abschnitt 4.3.2 vorgestellten eindimensionalen Partitionierungsalgorithmen hinsichtlich erzielter Lastbalance und der Ausführungszeit. Diese Verfahren stellen den Kern der Partitionierung basierend auf raumfüllenden Kurven dar. Es werden die Heuristiken *H1* und *H2* von Miguet und Pierson [76] mit denen in dieser Arbeit entwickelten Verfahren *r-BISECT* und *r-PAR-BISECT* verglichen. Letztere basieren auf dem Algorithmus *EXACT-BISECT* von Pınar und Aykanat [82]. Bei den Messungen wurden verschiedene Werte des Qualitätsfaktors r der Verfahren *r-BISECT* und *r-PAR-BISECT* berücksichtigt, um so dessen Einfluss auf Lastbalance und Ausführungszeit zu ermitteln.

Der Vergleich erfolgte anhand gemessener Berechnungszeiten von COSMO-SPECS+FD4 für das in Abschnitt 2.6.2 vorgestellte Szenario, jedoch mit einer horizontalen Gittergröße von 32×32 Gitterzellen. Die Zeiten zur Berechnung der Advektion der Hydrometeore und des Mikrophysikmodells SPECS je COSMO-Zeitschritt $\Delta t^{dyn} = 10\,\mathrm{s}$ wurden für Blöcke von je 2×2×2 Gitterzellen gemessen. Die Messung erfolgte auf der SGI Altix 4700. Auf diese Weise wurden 6144 Blockgewichte für jeden der 180 COSMO-Zeitschritte gewonnen. Davon wurden nur die letzten 100 Schritte für den Vergleichsbenchmark genutzt, da zu Beginn eine geringere Imbalance besteht. Die Lastimbalance der Blockgewichte $\lambda^{-1} = B\, w_{max} / \sum w_i$ liegt für die gewählten Schritte zwischen 4,32 und 6,61 und im Mittel bei 5,58. Diese Blockgewichte wurden in der Horizontalen schachbrettartig auf ein um 8×8 vergrößertes Gitter mit nunmehr 128×128×24 = 393 216 Blöcken repliziert. Um Wiederholungsmuster zu vermeiden, wurden alle Gewichte um bis zu ±2,5 % zufällig modifiziert. Abbildung 5.2 stellt einen Teil des daraus resultierenden Gewichtsvektors, welcher durch Linearisierung mit einer Hilbert-Kurve erzeugt wurde, grafisch dar. Die in diesem Teil auftretenden Muster setzen sich auch für den Rest des Gewichtsvektors in ähnlicher Weise fort. Die Präfixsumme dieses Vektors ist die Eingabe des eindimensionalen Partitionierungsalgorithmus.

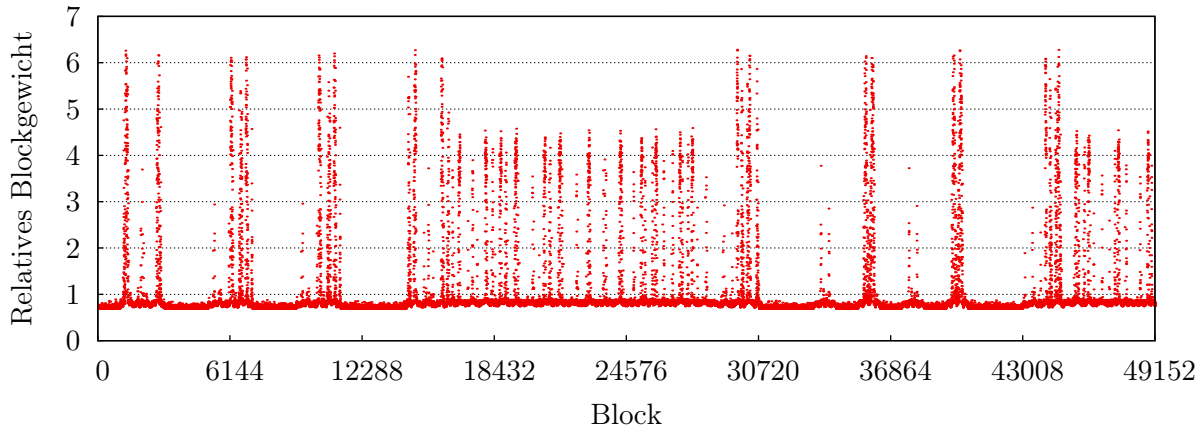
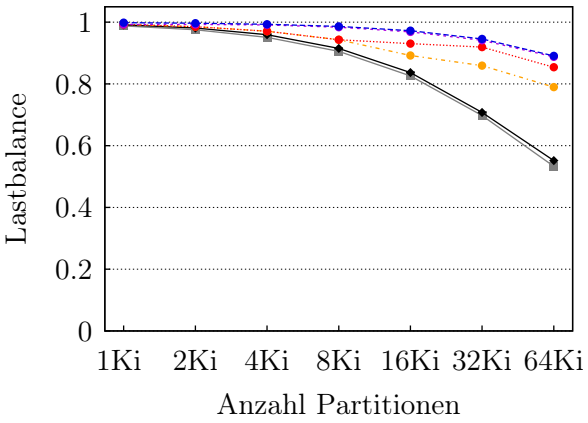


Abbildung 5.2: Darstellung des Gewichtsvektors für den Letzten der 100 Schritte zum Vergleich eindimensionaler Partitionierungsalgorithmen. Es sind die Gewichte der ersten 49 152 Blöcke (1/8 von insgesamt 393 216 Blöcken) relativ zum durchschnittlichen Gewicht aller Blöcke abgebildet.

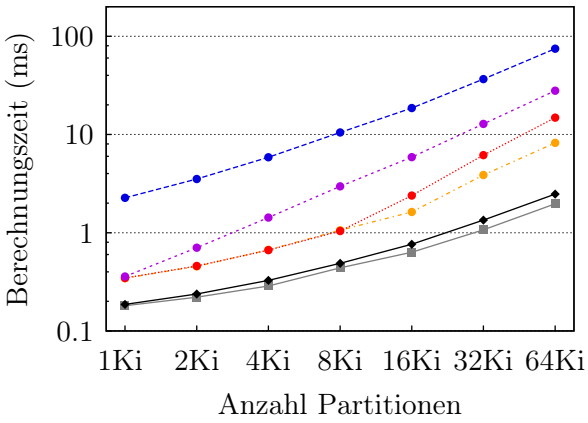
Abbildung 5.3 zeigt die Messergebnisse der seriellen Verfahren auf der SGI Altix 4700 für 1024 bis 65 536 Partitionen. Dargestellt sind jeweils Durchschnittswerte über die 100 Schritte des Benchmarks für die Lastbalance, die Rechenzeit sowie den Anteil migrierter Blöcke an der Gesamtblockzahl. Die Lastbalance nimmt bei allen Verfahren mit zunehmender Partitionszahl ab, da die durchschnittliche Anzahl von Blöcken je Partition abnimmt. In letztem Fall liegt diese Zahl bei 6, womit bei $\lambda^{-1} > 6$ selbst theoretisch keine Lastbalance von 1,0 möglich ist (siehe Abschnitt 4.2.2). Die beiden Heuristiken erreichen eine deutlich schlechtere Balance als *r-BISECT*, sind jedoch bei der Berechnung in allen Fällen schneller. Interessant ist der Vergleich von *r-BISECT* mit $r = 0{,}99$ und $r = 1{,}0$. In letztem Fall wird immer die optimale Balance für das eindimensionale Partitionierungsproblem erreicht, wofür jedoch eine hohe Zahl an Suchschritten des Bisektionsverfahrens nötig ist. Bei 64 Ki Partitionen werden im Durchschnitt 18,8 Suchschritte benötigt. Dagegen wird bei $r = 0{,}99$ mit deutlich weniger Aufwand eine nur unwesentlich geringer Lastbalance erreicht. Hier sind durchschnittlich 7 Suchschritte bei 64 Ki Partitionen notwendig. Damit bietet *r-BISECT* bei $r = 0{,}99$ den Vorteil einer nahezu exakten Lösung bei deutlich vermindertem Berechnungsaufwand gegenüber dem exakten Verfahren.

Der Vergleich des Anteils migrierter Blöcke in Abbildung 5.3(c) zeigt einen Vorteil für die Heuristiken. Bei *r-BISECT* mit $r < 1{,}0$ werden bei 64 Ki Partitionen nahezu alle Blöcke bei einer Repartitionierung migriert. Im Schnitt bleibt nur ca. 1 % der Blöcke in der gleichen Partition. Die Ursache für dieses Verhalten ist, dass die Heuristiken die Partitionsgrenzen innerhalb des Gewichtsvektors allein basierend auf dem idealen *bottleneck value* $\beta^* = \sum w_i / P$ positionieren, dessen Wert während eines Benchmarklaufs nur von der Gesamtlast abhängt. Dagegen nutzt *r-BISECT* mit $r = 1{,}0$ zur Erstellung der Partitionierung den minimalen tatsächlich möglichen *bottleneck value* $\beta_{opt} \geq \beta^*$, welcher zusätzlich von den einzelnen Blockgewichten (und deren Reihenfolge im Gewichtsvektor) abhängt und damit stärkeren Schwankungen ausgesetzt ist. Bei $r < 1{,}0$ verstärkt sich der Effekt, da der jeweils erreichte *bottleneck value* β_{res} zusätzlich über das Intervall $[\beta_{opt}, \beta_{opt}/r]$ variiert. Für *r-BISECT* mit $r = 0{,}99$ bei 64 Ki Partitionen wurde bei 21 der 100 Schritte ein sinkender Wert für β_{res} beobachtet, obwohl die Gesamtlast in nur einem der Schritte sinkt.

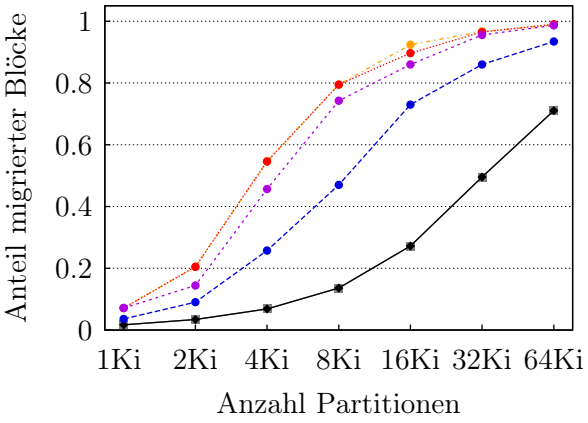
In Abbildung 5.4 ist die Berechnungszeit des parallelen Verfahrens *r-PAR-BISECT* den Zeiten der seriellen Verfahren gegenübergestellt. Diese Messungen erfolgten auf der IBM BlueGene/P, wobei für *r-PAR-BISECT* die gleiche Zahl an Prozessen genutzt wurde, wie Partitionen erzeugt wurden. Die Zeit für das Ermitteln des minimalen gültigen *bottleneck value* mittels ei-

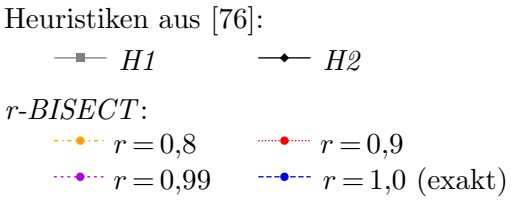


(a) Durchschnittliche Lastbalance



(b) Durchschnittliche Berechnungszeit





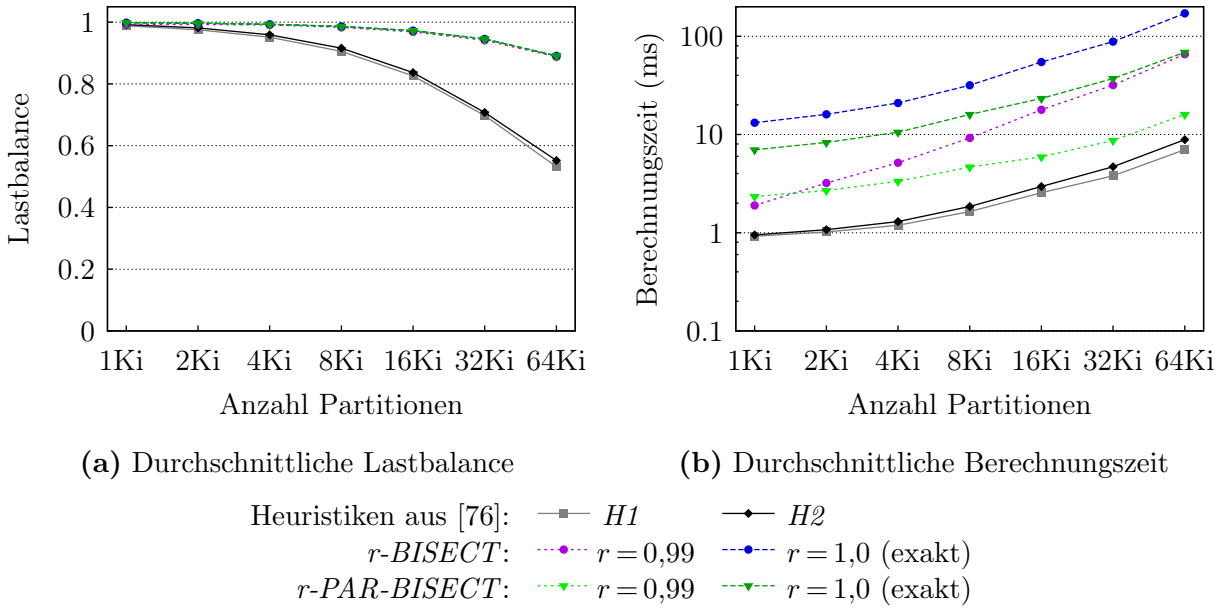
(c) Durchschnittlicher Anteil migrierter Blöcke an der Gesamtblockanzahl

Abbildung 5.3: Vergleich der in Abschnitt 4.3.2 eingeführten seriellen eindimensionalen Partitionierungsalgorithmen für 393 216 gewichtete Blöcke (SGI Altix 4700).

ner `MPI_Allreduce`-Operation ist in den Messwerten des parallelen Verfahrens enthalten. Die verbesserte Skalierbarkeit von *r-PAR-BISECT* gegenüber dem seriellen *r-BISECT* führt dazu, dass für $r = 0{,}99$ ab 16 Ki Partitionen nur noch etwa doppelt so viel Zeit als bei den einfachen Heuristiken benötigt wird. Die Zahl der Suchschritte bei 64 Ki Partitionen beträgt für $r = 1{,}0$ im Durchschnitt 6,8 und für $r = 0{,}99$ immer 1. Das bedeutet, dass jeder Prozess nur einen möglichen *bottleneck value* am unteren Ende seines Teilintervalls prüft und keine binäre Suche notwendig ist, da Aufgrund der hohen Zahl kleiner Teilintervalle die notwendige Genauigkeit bereits erreicht ist (siehe Gleichung 4.6, Seite 50). Das Verfahren *r-PAR-BISECT* bietet hinsichtlich der Abwägung von erreichter Lastbalance gegenüber der Berechnungszeit die günstigsten Eigenschaften für die Lastbalancierung der spektralen Mikrophysik. Die Heuristiken sind zwar schnell in der Berechnung und führen zu geringerer Blockmigration, erreichen jedoch bei hoher Prozessanzahl eine nur ungenügende Lastbalance. Daher wird *r-PAR-BISECT* mit $r = 0{,}99$ für die folgenden Messungen mit Partitionierung basierend auf raumfüllenden Kurven gewählt.

5.2.2 Vergleich von COSMO-SPECS mit COSMO-SPECS+FD4

Zum Vergleich von COSMO-SPECS mit COSMO-SPECS+FD4 wurden die Messungen aus Abschnitt 2.6.2 ebenfalls mit COSMO-SPECS+FD4 auf der SGI Altix 4700 durchgeführt. Die



(a) Durchschnittliche Lastbalance **(b)** Durchschnittliche Berechnungszeit

Abbildung 5.4: Vergleich der seriellen eindimensionalen Partitionierungsalgorithmen mit dem parallelen Algorithmus für 393 216 gewichtete Blöcke (IBM BlueGene/P).

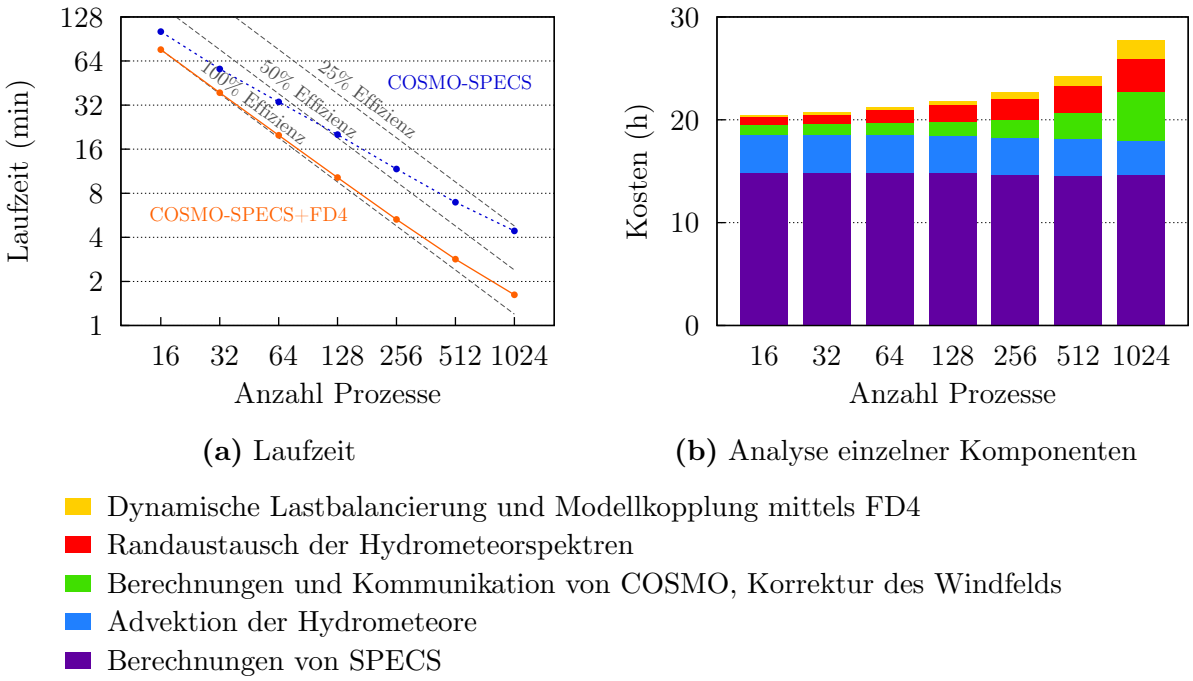
dynamische Lastbalancierung erfolgte mithilfe der Hilbert-Kurve und dem Algorithmus *r-PAR-BISECT* mit $r = 0{,}99$. Dabei wird das in Abschnitt 5.1.1 eingeführte Verfahren zur Abschätzung, ob der Aufruf der Lastbalancierung lohnenswert ist, genutzt. Ob am Ende eines COSMO-Zeitschritts dynamische Lastbalancierung durchgeführt wird entscheidet FD4 somit selbständig anhand von Zeitmessungen aus vergangenen Schritten. Die FD4-Blockgröße wurde mit 2×2×4 festgelegt, so dass das Gitter in 32×32×12 = 12 288 Blöcke zerlegt wird.

Abbildung 5.5(a) vergleicht die Laufzeit des originalen Modellsystems mit der lastbalancierten Version. Deutlich werden vor allem die weitaus geringeren Effizienzverluste bei der Skalierung von COSMO-SPECS+FD4. Dadurch benötigt es bei 1024 Prozessen weniger als 37 % der Zeit von COSMO-SPECS. Schon bei 16 Prozessen ist das neue Modellsystem um den Faktor 1,3 schneller, was auf die günstigeren Datenstrukturen für die Hydrometeore, welche ein Kopieren der Größenspektren vor und nach dem Aufruf von SPECS unnötig machen, sowie eine geringere Kommunikationszeit zurückzuführen ist.

Die Analyse der Skalierbarkeit einzelner Komponenten von COSMO-SPECS+FD4 in Abbildung 5.5(b) zeigt, dass der Kommunikationsaufwand beim Randaustausch der Hydrometeorspektren im Vergleich zum Original nur mäßig zunimmt (vgl. Abbildung 2.4(b), Seite 16). Der Aufwand nimmt um Faktor 4,8 zu, was nur leicht über dem bei einer dreidimensionalen Dekomposition idealerweise zu erwartenden Faktor vier liegt. Dieser lässt sich basierend auf Formel 3.2 mit $1024^{\frac{1}{3}}/16^{\frac{1}{3}}$ bestimmen. In einer zweidimensionalen Dekomposition, wie bei dem ursprünglichen COSMO-SPECS, ist dagegen bestenfalls mit Faktor acht zu rechnen. Zudem profitiert der Kommunikationsaufwand von der verbesserten Lastbalance, so dass deutlich geringere Wartezeiten auftreten.

Durch die stark reduzierte Ausführungszeit bei hoher Parallelität fällt bei COSMO-SPECS+FD4 die unzureichende Skalierbarkeit von COSMO und der Windfeldkorrektur wesentlich stärker ins Gewicht. Die Ursachen dieser ungenügenden Skalierbarkeit sind in Abschnitt 2.6.2 dargelegt.

Die Laufzeitanteile von FD4 für die dynamische Lastbalancierung und für die Kopplung betragen bei 1024 Prozessen 5,7 % bzw. 0,6 %. Im Durchschnitt verbringt ein Prozess 6,2 s in FD4. Der zusätzliche Aufwand für das Erreichen einer besseren Lastbalance ist lohnenswert,



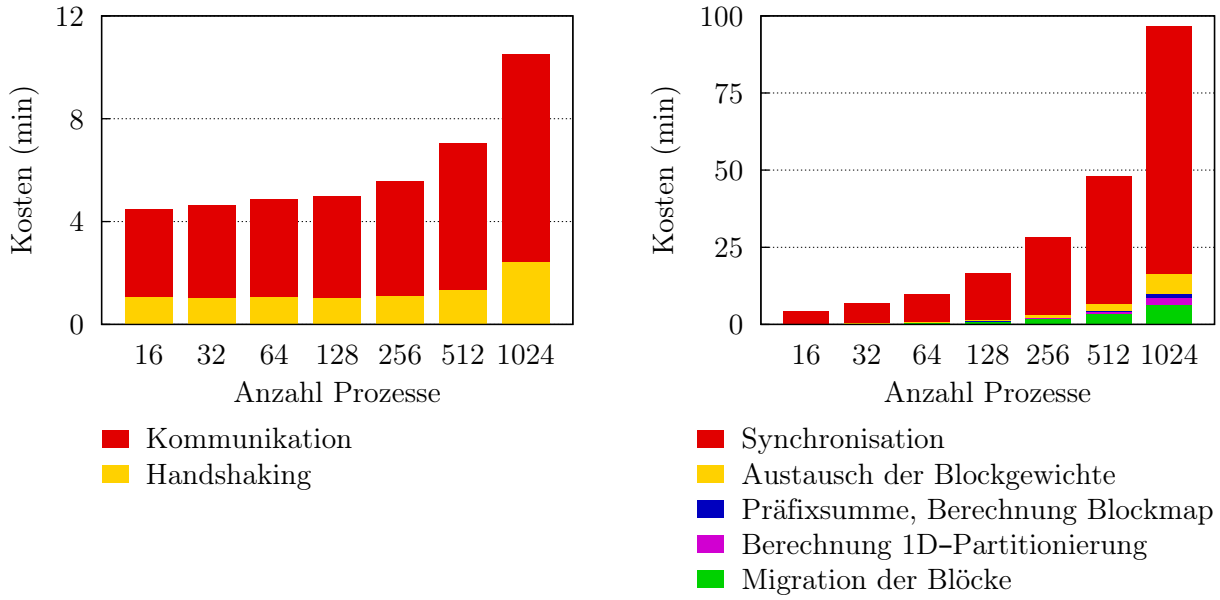
(a) Laufzeit **(b)** Analyse einzelner Komponenten

Abbildung 5.5: Vergleich der Laufzeit von COSMO-SPECS+FD4 mit COSMO-SPECS und Skalierbarkeitsanalyse einzelner Komponenten von COSMO-SPECS+FD4 (SGI Altix 4700).

da diesem durchschnittlich 51 s Wartezeit durch Lastimbalance im originalen Modellsystem gegenüberstehen.

Abbildung 5.6(a) stellt die Skalierbarkeit der FD4-Kopplung dar. Bis 512 Prozesse zeigen das *handshaking*, also die Identifikation überschneidender Partitionen, und die eigentliche Datenübertragung eine sehr gute Skalierbarkeit. Für das *handshaking* entspricht dies dem erwarteten Ergebnis, da nur Überschneidungen im Bereich der lokalen Partition berechnet werden. Der starke Zuwachs der Dauer des *handshaking* bei 1024 Prozessen ist nicht durch den Algorithmus verursacht. Er resultiert aus dem zeitgleichen Zugriff hunderter Prozesse auf den gleichen Programmtext im gemeinsamen Speicher der SGI Altix 4700. Dies wurde bereits in Abschnitt 2.6.2 als systemspezifisches Skalierbarkeitsproblem des COSMO-Modells identifiziert. Insgesamt erhöht sich von 16 zu 1024 Prozessen der Gesamtaufwand der Kopplung um den Faktor 2,3.

Die Skalierbarkeit der FD4-Lastbalancierung ist in Abbildung 5.6(b) dargestellt. Dabei wird deutlich, dass die Lastbalancierung kaum skaliert, da sich der Gesamtaufwand über alle Prozesse um Faktor 22,4 erhöht. Abgesehen von der Synchronisation lässt sich das auf die Abhängigkeit der Komplexität der Algorithmen von der Prozessanzahl zurückführen. Bei der steigenden Synchronisationszeit, welche den Gesamtaufwand der Lastbalancierung dominiert, spielen zwei Ursachen eine Rolle: Zum einen nimmt die Granularität der Blockdekomposition relativ zur Prozessanzahl ab. Bei 16 Prozessen stehen je Partition im Schnitt 768 Blöcke zur Verfügung, wohingegen es bei 1024 Prozessen nur noch 12 Blöcke sind. Zum anderen wird die Synchronisationszeit indirekt durch den steigenden Aufwand der Partitionsberechnung und Migration erhöht. Grund ist, dass FD4 bei steigender Prozessanzahl seltener eine Repartitionierung vornimmt, da der zunehmende Aufwand diese immer weniger lohnenswert werden lässt. Werden bei 16 Prozessen in 119 von 180 Zeitschritten eine Lastbalancierung durchgeführt, ist dies bei 1024 Prozessen nur noch in 40 Zeitschritten der Fall. Dass heißt, FD4 nimmt eine geringere Lastbalance in Kauf, um den Aufwand für die Repartitionierung nicht größer als die Zeitersparnis durch balancierte Berechnungen werden zu lassen. Damit erhöht sich jedoch die Synchronisationszeit zu Beginn des Lastbalancierungsverfahrens. Wird dagegen bei 1024 Prozessen eine Lastbalancierung zu



(a) FD4-Modellkopplung **(b)** FD4-Lastbalancierung

Abbildung 5.6: Analyse der Skalierbarkeit der FD4-Modellkopplung und FD4-Lastbalancierung (SGI Altix 4700). Es ist jeweils die aufsummierte Zeit aller Prozesse aufgeschlüsselt nach Anteilen einzelner Komponenten angegeben.

jedem Zeitschritt durchgeführt, so erhöht sich die Laufzeit von 97 s auf 100 s obwohl sich die durchschnittliche Lastbalance von 90,6 % auf 93,7 % verbessert. Der Verlauf der Lastbalance in diesen zwei Modi ist in Abbildung 5.7 der Lastbalance des ursprünglichen Modellsystems gegenübergestellt. Darin wird deutlich, dass durch die dynamische Lastbalancierung das starke und beständige Absinken der Lastbalance verhindert wird. Bei automatischer Entscheidung durch FD4 über den Zeitpunkt der Lastbalancierung zeigt sich ein charakteristisches „Sägezahnmuster“ mit über mehrere Zeitschritte stark fallender Lastbalance bis zu einem Punkt, an dem FD4 abschätzt, dass eine Repartitionierung lohnenswert ist. Die Balance fällt dabei nie unter 80 %.

Insgesamt zeigen die Vergleichsmessungen, dass durch die Realisierung des Konzepts der lastbalancierten Kopplung erfolgreich die Lastimbalancen minimiert worden sind. Zwar skaliert das Lastbalancierungsverfahren *r-PAR-BISECT* nur unbefriedigend, dennoch ist die dynamische Lastbalancierung auch bei einer in Bezug zur Gittergröße hohen Parallelität profitabel. Die Hochskalierbarkeitsmessungen, welche in Abschnitt 5.2.4 präsentiert werden, unterstreichen dieses Resultat.

5.2.3 Vergleich raumfüllender Kurven mit Graphenpartitionierung

Zum Vergleich der Lastbalancierungsmethoden basierend auf raumfüllenden Kurven und der Graphenpartitionierung wurden mit COSMO-SPECS+FD4 Leistungsmessungen für das gleiche Szenario wie im vorigen Abschnitt durchgeführt. Für eine äquivalente Vergleichsbasis wurde jedoch die Repartitionierung in jedem COSMO-Zeitschritt durchgeführt. Mit der Hilbert-Kurve und der Morton-Kurve wurden zwei Typen von raumfüllenden Kurven betrachtet. Für die Graphenpartitionierung basierend auf der Bibliothek ParMetis wurden Messungen mit einer tolerierten Imbalance $ubvec = 1{,}05$ durchgeführt, so wie in der Dokumentation empfohlen. Um eine höhere Lastbalance zu erzielen wurden zusätzlich Messungen mit $ubvec = 1{,}01$ durchgeführt. Der Parameter *itr* zur Gewichtung der Kommunikation zwischen Nachbarpartitionen gegenüber

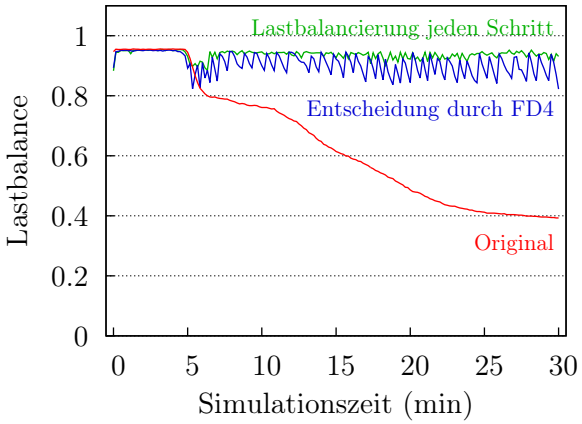


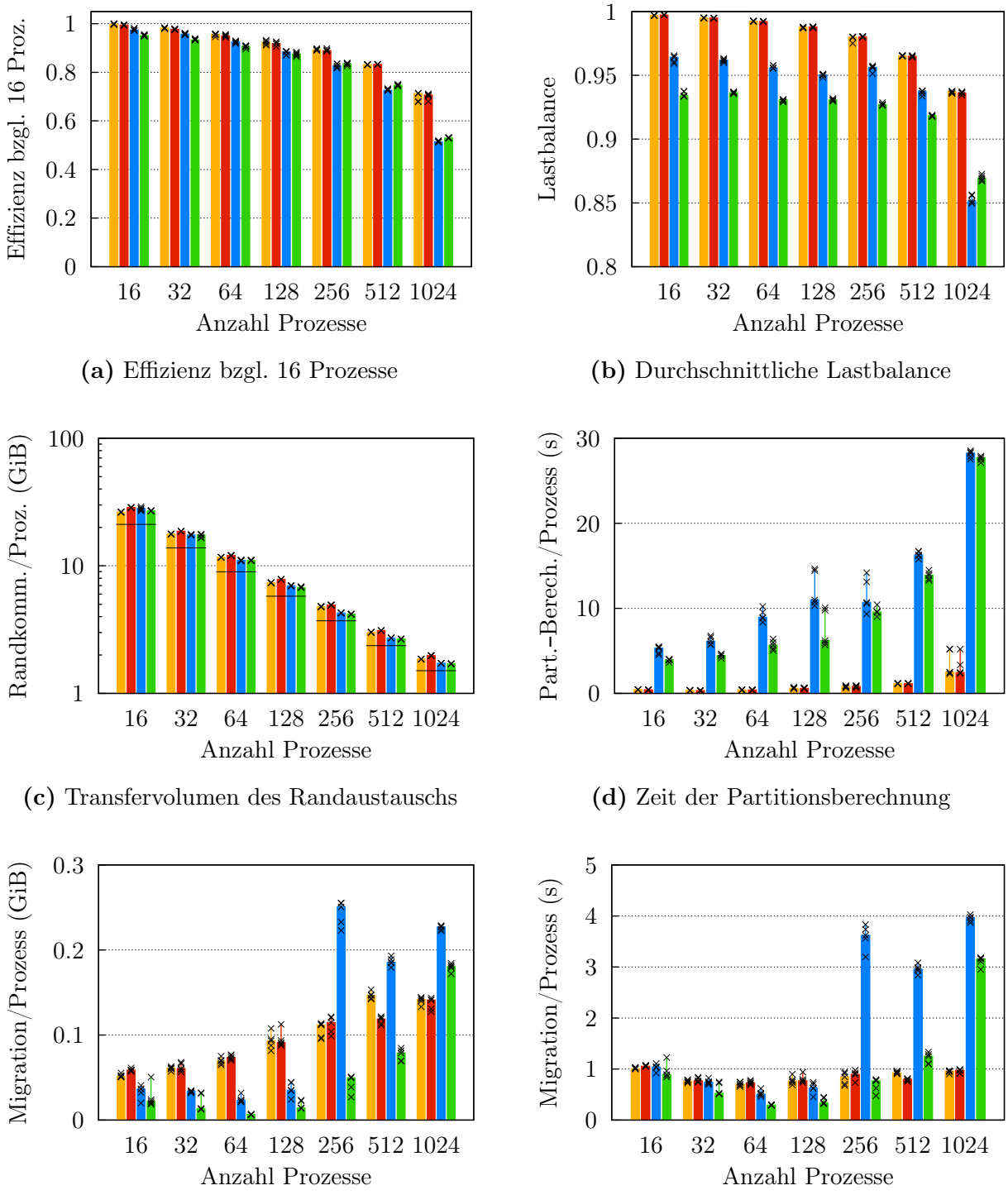
Abbildung 5.7: Vergleich des Verlaufs der Lastbalance über die Simulationszeit bei 1024 Prozessen (SGI Altix 4700). Für COSMO-SPECS+FD4 ist die Lastbalance sowohl bei Aufruf der Lastbalancierung in jedem Zeitschritt als auch bei automatischer Entscheidung durch FD4 dargestellt.

der Migration wurde mit 1000,0 festgelegt. Bei Tests wurde kein entscheidender Einfluss dieses Parameters auf die Partitionsqualität und die Berechnungszeit festgestellt.

Die Messergebnisse bzgl. sechs verschiedener Metriken sind in Abbildung 5.8 dargestellt. Diese Metriken lassen einen Vergleich hinsichtlich der in Abschnitt 3.1.1 eingeführten vier Anforderungen an Partitionierungsverfahren zu. Es wurden je Konfiguration (Partitionierungsverfahren und Prozessanzahl) fünf Messungen durchgeführt, wobei vor allem bei ParMetis variierende Resultate für einige Metriken beobachtet wurden. Daher ist jeweils der Median der fünf Messungen dargestellt und zur Verdeutlichung der Streuung zusätzlich der Wert jeder individuellen Messung.

Abbildung 5.8(a) stellt die parallele Effizienz von COSMO-SPECS+FD4 bzgl. des Laufs mit 16 Prozessen und Partitionierung mittels einer Hilbert-Kurve dar. Mit steigender Parallelität wird hier ein zunehmender Vorteil der gleichauf liegenden SFC-basierten Methoden deutlich. Ein Grund liegt in der geringeren Lastbalance der von ParMetis erzeugten Partitionierungen, wie in Abbildung 5.8(b) dargestellt. Die genauen Ursachen dafür sind nicht klar. Jedoch wurde dieses Verhalten von ParMetis auch bei anderen Anwendungen beobachtet [7]. Nahe liegend ist, dass die expliziten Optimierungen bezüglich anderer Anforderungen an die Partitionierung, wie etwa der Kommunikation zwischen Partitionen, zu einer geringeren Lastbalance beitragen.

Abbildung 5.8(c) stellt die Anwendungskommunikation dar, dass heißt, das Kommunikationsvolumen des Austauschs der Daten von Partitionsrändern zwischen Nachbarprozessen des gesamten Simulationslaufs. Das theoretische Minimum, basierend auf Gleichung 3.3 (Seite 22), ist durch schwarze Linien angedeutet. Die Grafik zeigt, dass ParMetis ab 64 Prozessen Partitionen geringerer Oberfläche erstellt als raumfüllende Kurven. Bei 1024 Prozessen erzeugen die Partitionierungen der Hilbert-Kurve in der Summe ca. 109 % des Kommunikationsvolumens der Partitionierungen von ParMetis mit $ubvec = 1{,}05$. Die Hilbert-Kurve erweist sich bezüglich des Kommunikationsvolumens immer als leicht günstiger im Vergleich zur Morton-Kurve. Der Vorteil des geringeren Kommunikationsvolumens durch ParMetis wird jedoch nicht in geringere Zeit für die Kommunikation umgesetzt, da die höhere Lastimbalance in dem Fall zu längeren Synchronisationszeiten zwischen Kommunikationspartnern führt. Zudem ist das Kommunikationsmuster weitaus irregulärer, wie der Vergleich der Nachrichtenstatistiken in Abbildung 5.9 zeigt. Für die SFC-Methoden liegt der Schwerpunkt der Anwendungskommunikation in der Nähe der Diagonalen, womit typischerweise eine geringere Belastung des Netzwerks einhergeht.



(a) Effizienz bzgl. 16 Prozesse

(b) Durchschnittliche Lastbalance

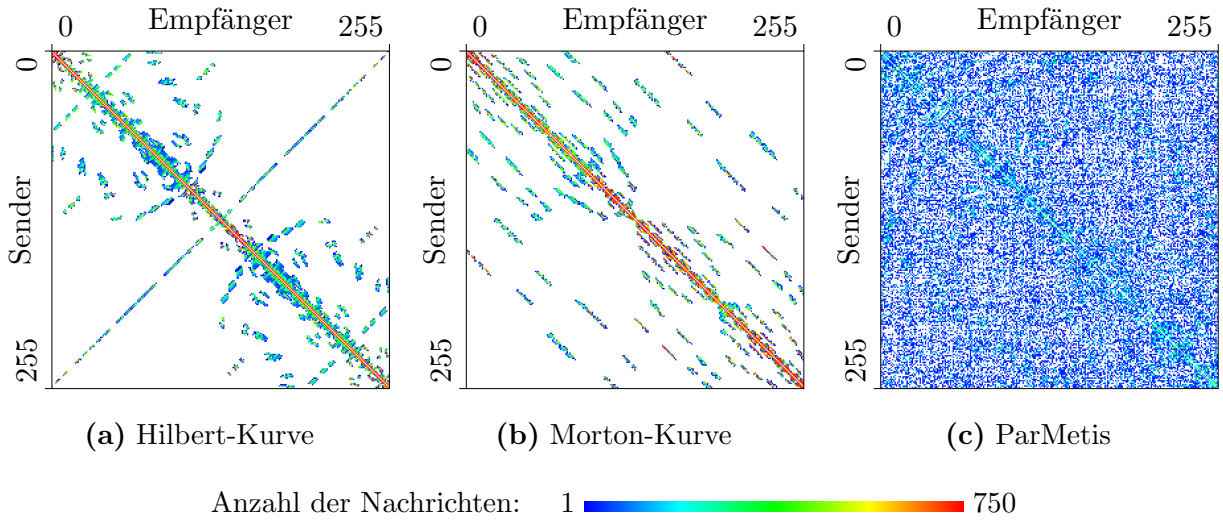
(c) Transfervolumen des Randaustauschs

(d) Zeit der Partitionsberechnung

(e) Transfervolumen der Migration

(f) Kommunikationszeit der Migration

Abbildung 5.8: Vergleich von raumfüllenden Kurven und Graphenpartitionierung zur Lastbalancierung in COSMO-SPECS+FD4 (SGI Altix 4700). Die Balken zeigen den Median von fünf Messungen je Konfiguration, welche jeweils durch Kreuze einzeln dargestellt sind.



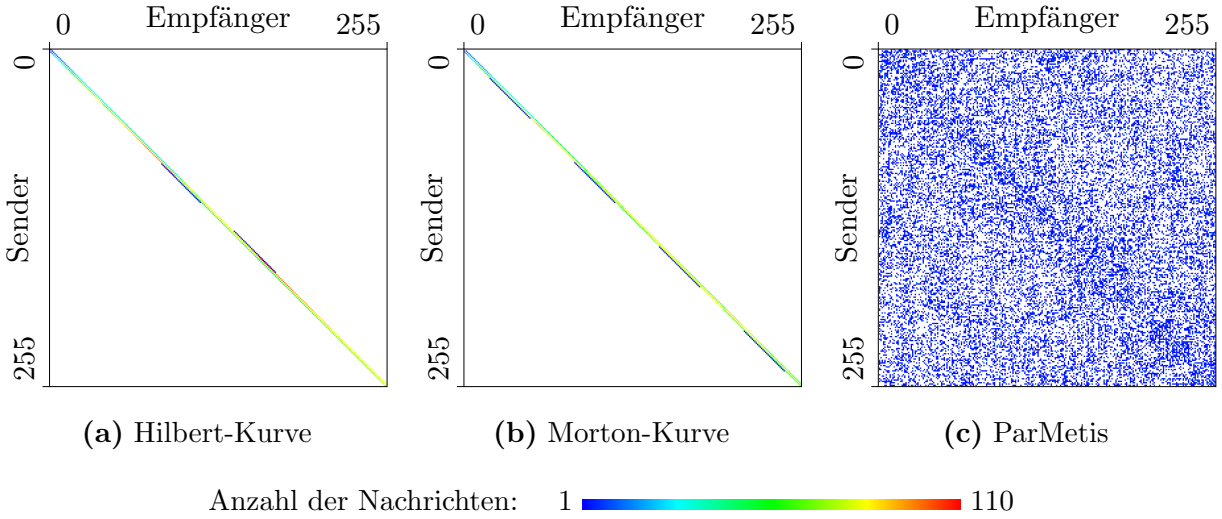
(a) Hilbert-Kurve **(b)** Morton-Kurve **(c)** ParMetis

Abbildung 5.9: MPI-Nachrichtenstatistiken für den Randaustausch der FD4-Partitionierung der drei untersuchten Partitionierungsverfahren. Dargestellt ist die Anzahl der Nachrichten je Prozesspaar aufsummiert über einen vollständigen Simulationslauf mit 256 Prozessen.

Die Zeit, welche je Prozess durchschnittlich zur Berechnung aller Partitionierungen eines Laufs benötigt wird, ist in Abbildung 5.8(d) dargestellt. Die Berechnungen für beide Lastbalancierungsmethoden skalieren nur sehr schlecht. Jedoch ist die SFC-Partitionsberechnung um ein vielfaches schneller als ParMetis. Bei 1024 Prozessen verbringt COSMO-SPECS+FD4 im Falle der Graphenpartitionierung ca. 20 % der Ausführungszeit in der Partitionsberechnung. Bei Partitionierung mit raumfüllenden Kurven sind dies maximal 5 %. Somit trägt die Geschwindigkeit und Skalierbarkeit der Partitionsberechnung wesentlich zur Gesamtlaufzeit und parallelen Effizienz des Modellsystems bei.

Das aufsummierte Transfervolumen und die Zeit für die Migration der Blöcke je Prozess sind in den Abbildungen 5.8(e) und 5.8(f) dargestellt. Das teilweise deutlich geringere Transfervolumen der Graphenpartitionierung im Vergleich zu den raumfüllenden Kurven führt oft nur zu einem geringen Vorteil bei der Übertragungszeit. Grund sind die sehr unterschiedlichen Kommunikationsmuster, welche in Abbildung 5.10 gegenübergestellt sind. Bei der SFC-Partitionierung werden die Blöcke zum großen Teil zwischen unmittelbar benachbarten Prozessen (bezogen auf den MPI-Rank) migriert, da die Repartitionierung nur eine Verschiebung der Prozessgrenzen im eindimensionalen Blockvektor darstellt. Im Vergleich zu dem irregulären Kommunikationsmuster der Migration durch ParMetis ermöglicht dies deutlich höhere Bandbreiten. Das Transfervolumen bei ParMetis steigt bei hoher Parallelität stark an, so dass bei 1024 Prozessen die Übertragungszeit wesentlich über der der SFC-Partitionierung liegt.

Zusammenfassend zeigt der Vergleich, dass die Graphenpartitionierung den raumfüllenden Kurven in nur einer der vier Anforderungen an Partitionierungsverfahren leicht überlegen ist, und zwar der Reduktion der Kommunikation zwischen Nachbarpartitionen. Andererseits wird durch den Einsatz raumfüllender Kurven eine höhere Lastbalance bei deutlich geringerer Berechnungszeit für die Partitionierung erreicht. Beim Aufwand der Migration kann sich kein Verfahren klar abgrenzen; jedoch zeigt sich mit steigender Parallelität ein Vorteil für die SFC-basierten Methoden. Für eine hochfrequente dynamische Lastbalancierung für skalierbare Anwendungen, wie sie im Kontext der Lastbalancierung für detaillierte Wolkenmikrophysik gefordert ist, bietet die Partitionierung mittels raumfüllender Kurven insgesamt eine höhere Leistung. Der Vergleich zwischen den zwei untersuchten Kurventypen zeigt bzgl. der Anwendungskommunikation einen geringen Vorteil für die Hilbert-Kurve, wohingegen bei den anderen drei Anforderungen keine Unterschiede zu beobachten sind.



(a) Hilbert-Kurve **(b)** Morton-Kurve **(c)** ParMetis

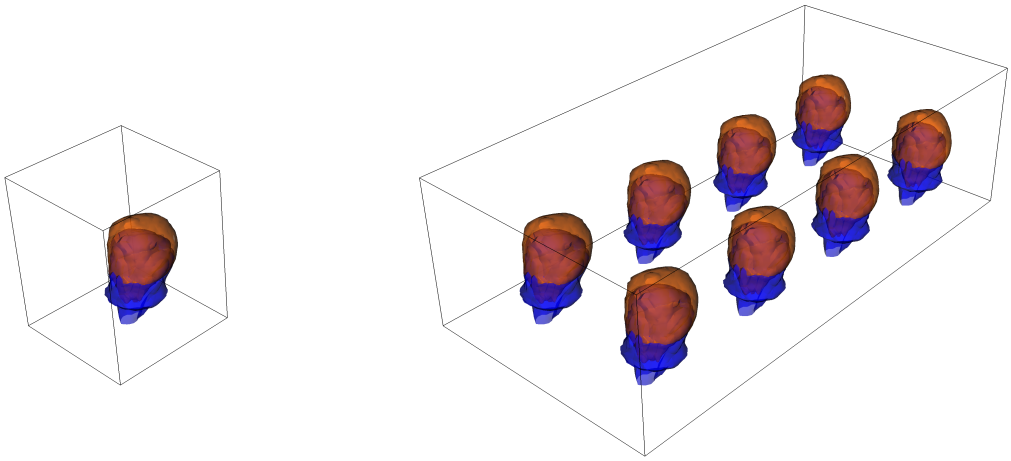
Abbildung 5.10: MPI-Nachrichtenstatistiken für die Migration der FD4-Blöcke der drei untersuchten Partitionierungsverfahren. Dargestellt ist die Anzahl der Nachrichten je Prozesspaar aufsummiert über einen vollständigen Simulationslauf mit 256 Prozessen.

5.2.4 Hochskalierbarkeitsmessungen

Im Folgenden wird das Skalierbarkeitsverhalten von COSMO-SPECS+FD4 bis auf über 100 000 parallele Prozesse auf der IBM BlueGene/P analysiert. Dabei steht die Frage im Mittelpunkt, ob der zusätzliche Aufwand für die dynamische Lastbalancierung und Kopplung auch bei hoher Parallelität im Vergleich zur statischen Partitionierung lohnenswert ist. Es ist insbesondere zu erwarten, dass der Aufwand der Lastbalancierung mit zunehmender Parallelität wächst, da die Komplexität des Austauschs der Blockgewichte aller Prozesse, der Berechnung der Präfixsumme des Gewichtsvektors sowie des eindimensionalen Partitionierungsverfahrens von der Prozess- und Blockanzahl abhängig ist.

Für diese Messungen wurde das bisher verwendete Szenario aus Abschnitt 2.6.2 für *weak scaling* abgewandelt, so dass ein weiter Bereich an Parallelität untersucht werden kann. Um sowohl Gittergröße als auch Rechenlast proportional mit der Prozessanzahl zu skalieren, wurde die Methode des *replication scaling* [111] angewandt. Dabei werden das Gitter und die initialen Zustandsvariablen entsprechend der sich jeweils verdoppelnden Prozessanzahl repliziert und so eine Verdopplung der Problemgröße erreicht. Das so zu lösende Problem besteht damit aus identischen Kopien eines Basisproblems, wobei weder den Modellen COSMO und SPECS noch den Lastbalancierungs- und Kopplungsalgorithmen in FD4 diese Regelmäßigkeit „bewusst“ ist. Bei dem Basisproblem wird analog zu dem bisher betrachteten Szenario die Entstehung einer Kumuluswolke durch eine initiale Wärmeblase angeregt. Der einzige Unterschied ist die verringerte horizontale Gittergröße von 32×32 Gitterpunkten. Dieses Basisproblem wurde mit 256 Prozessen ausgeführt. Bei jeder Verdoppelung der Prozessanzahl wurde auch die Gittergröße abwechselnd in einer der beiden horizontalen Dimensionen verdoppelt und damit auch die Zahl der simulierten Wolken. In Abbildung 5.11 sind die simulierten Wolken für die Konfigurationen für 256 bzw. 2048 Prozesse dargestellt. In der größten Konfiguration für 131 072 Prozesse werden so 32×16 Wolken auf einem Gitter der Größe 1024×512×48 Zellen simuliert. In Tabelle 5.1 sind alle zehn Konfigurationen aufgeführt.

Für COSMO-SPECS+FD4 wurde das Gitter gleichmäßig in Blöcke der Größe 2×2×4 Gitterzellen zerlegt, wodurch sich im Durchschnitt 12 Blöcke je Prozess ergeben. Zur dynamischen Lastbalancierung wurde die Hilbert-Kurve und das Verfahren *r-PAR-BISECT* genutzt. Die Lastbalancierung wurde mit einem Sollwert für die Lastbalance von 1,0 bei jedem der 180 COSMO-



(a) Bei 256 Prozessen **(b)** Bei 2048 Prozessen

Abbildung 5.11: Visualisierung der Wolken nach 30 min Simulationszeit des Szenarios für die Hochskalierbarkeitsmessungen mit *replication scaling*. Es sind Isoflächen für Wolkenwasser (blau) und Wolkeneis (orange) bei der Konzentration von 10^{-3} g/kg dargestellt.

Schritte gerufen. Die Kopplungsmetadaten wurden so zerlegt, dass eine Metadaten-Subdomain genau 2×2 Partitionen in der horizontalen Partitionierung von COSMO beinhaltet. Dass heißt, die Anzahl der Metadaten-Subdomains ist immer ein Viertel der Prozessanzahl.

Die gemessenen Laufzeiten von COSMO-SPECS+FD4 sind in Tabelle 5.1 den Laufzeiten von fünf Konfigurationen von COSMO-SPECS gegenübergestellt. Trotz eines zunehmenden Aufwands der dynamischen Lastbalancierung liegt die Laufzeit von COSMO-SPECS+FD4 auch bei hoher Parallelität deutlich unter der des ursprünglichen Modellsystems. Die Hauptursache dieser Differenz sind die Lastimbalancen in COSMO-SPECS, welche zu Wartezeiten bei der Kommunikation zwischen Nachbarpartitionen führen. Mit diesen Ergebnissen wird gezeigt, dass das vorgestellte Konzept der lastbalancierten Kopplung auch im hochparallelen Einsatz zu einer deutlichen Leistungssteigerung der detaillierten Wolkensimulation führt. Die Zunahme des Anteils der Lastbalancierung ist durch die angesprochene Abhängigkeit des Aufwands von der Prozess- und Blockanzahl zurückzuführen.

Abbildung 5.12 zeigt eine detaillierte Analyse verschiedener Messwerte der zehn Läufe mit COSMO-SPECS+FD4. Aus Abbildung 5.12(a) geht hervor, dass die Zunahme der Laufzeit mit steigender Parallelität zum Großteil auf die Lastbalancierung und Kopplung zurückzuführen ist. Jedoch skalieren die Kommunikation zwischen den FD4-Partitionen und die Berechnung des COSMO-Modells ebenfalls nicht ideal, was zum einen auf die leicht sinkende Lastbalance (siehe Abbildung 5.12(d)) und zum anderen auf einen stark ansteigenden Kommunikationsanteil innerhalb von COSMO zurückzuführen ist.

Abbildung 5.12(b) stellt die zunehmende Laufzeit der FD4-Lastbalancierung dar. Die gezeigten Phasen entsprechen dabei denen aus Abbildung 5.1. Bis etwa 16 384 Prozesse wird die Laufzeit der FD4-Lastbalancierung von der Synchronisationszeit dominiert. Sowohl die kommunikationsintensiven Phasen (Austausch der Blockgewichte und Migration) als auch insbesondere die berechnungsintensiven Phasen (Präfixsumme und eindimensionale Partitionierung) nehmen jedoch mit steigender Parallelität deutlich an Laufzeit zu. Dabei steht eine 512-fache Steigerung der Prozess- und Blockanzahl einer ca. 60-fachen Steigerung der Zeit für die Lastbalancierung ohne Synchronisation gegenüber. Ursache ist, dass die Komplexität der Algorithmen der Lastbalancierung von der Prozess- und Blockanzahl abhängt. Der Aufwand der Berechnung der Präfixsumme des Gewichtsvektors und des eindimensionalen Partitionierungsverfahrens steigt linear mit der

Anzahl Prozesse	Gittergröße (horizontal)	Anzahl Wolken	COSMO-SPECS+FD4 Blöcke	Laufzeit	Anteil Lb.	Original Laufzeit
256	32×32	1×1	3 072	354 s	0,2 %	814 s
512	64×32	2×1	6 144	356 s	0,3 %	–
1 024	64×64	2×2	12 288	363 s	0,4 %	–
2 048	128×64	4×2	24 576	359 s	0,5 %	809 s
4 096	128×128	4×4	49 152	365 s	0,7 %	–
8 192	256×128	8×4	98 304	368 s	1,1 %	812 s
16 384	256×256	8×8	196 608	383 s	1,7 %	–
32 768	512×256	16×8	393 216	392 s	2,8 %	822 s
65 536	512×512	16×16	786 432	409 s	5,0 %	–
131 072	1024×512	32×16	1 572 864	431 s	8,3 %	822 s

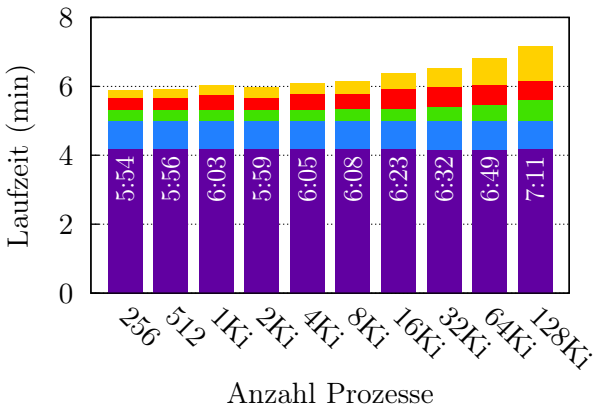
Tabelle 5.1: Konfiguration der Hochskalierbarkeitsmessungen von COSMO-SPECS und COSMO-SPECS+FD4 sowie gemessene Laufzeiten und Anteile der FD4-Lastbalancierung (ohne Synchronisation) auf der IBM BlueGene/P.

Block- bzw. Prozessanzahl. Eine genauere Analyse ergab, dass von 256 zu 131 072 Prozessen der Zeitbedarf zur Berechnung der Präfixsumme um den Faktor 526 (von 26 ms auf 13,8 s) und des eindimensionalen Partitionierungsverfahrens um den Faktor 288 (von 14 ms auf 4,1 s) steigt. Dagegen steigt der Aufwand zur Berechnung der notwendigen Teile der Blockmap nur vergleichsweise wenig von 49 ms auf 92 ms, was durch die logarithmisch wachsende Komplexität der Berechnung der raumfüllenden Kurve verursacht wird. Insgesamt wird deutlich, dass eine weitergehende Parallelisierung durch die parallele Berechnung der Präfixsumme bzw. durch das in Abschnitt 4.3.3 beschriebene verteilte Verfahren der Partitionsberechnung *r-DISTR-BISECT* lohnenswert und für eine noch höhere Skalierbarkeit unabdingbar ist.

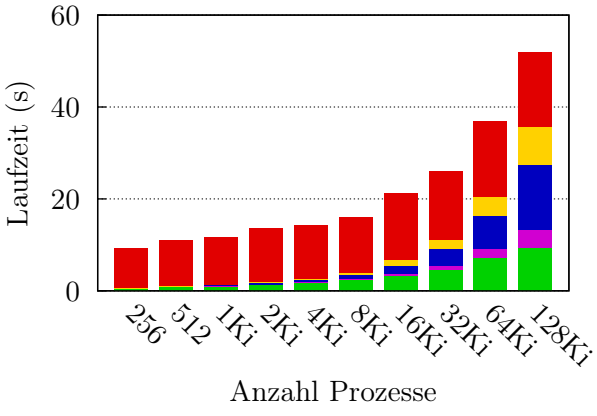
Die Skalierbarkeit der Modellkopplung ist in Abbildung 5.12(c) dargestellt. Wie erwartet skaliert das *handshaking*, dass heißt die Berechnung der Überschneidungen zwischen den COSMO- und FD4-Partitionen, nahezu perfekt. Dass dennoch eine geringe Zunahme auftritt, ist auf die sinkende Zahl an Zeitschritten ohne Partitionsänderung zurückzuführen, wodurch immer seltener das *handshaking*-Resultat des vorhergehenden Zeitschritts wiederverwendet werden kann. Bereits ab 1024 Prozesse wird jeden Zeitschritt die globale Partitionierung geändert, wobei sich jedoch nicht immer für alle Prozesse die lokale Partition ändert. Die Zunahme des eigentlichen Datentransfers der Kopplung ist bedingt durch das sehr irreguläre Kommunikationsmuster. Bei zunehmender Parallelität führt dies zu einer höheren Belastung des Verbindungsnetzwerks und somit zu sinkenden Bandbreiten.

Das Transfervolumen zwischen benachbarten FD4-Partitionen je Prozess hängt, wie Abbildung 5.12(e) zeigt, nicht von der Anzahl der Prozesse ab, was durch das *weak scaling* bedingt ist. Basierend auf Gleichung 3.2 (Seite 22) für 729 Variablen und zwei Reihen an Geisterzellen liegt das theoretische Minimum bei 1,51 GiB. Der tatsächlich erreichte Wert von 1,84 GiB entspricht 122 % dieses Minimums. Diese Abweichung lässt sich darauf zurückführen, dass das Minimum nur bei quaderförmigen Partitionen erreicht wird, welche durch raumfüllende Kurven typischerweise nicht erzeugt werden.

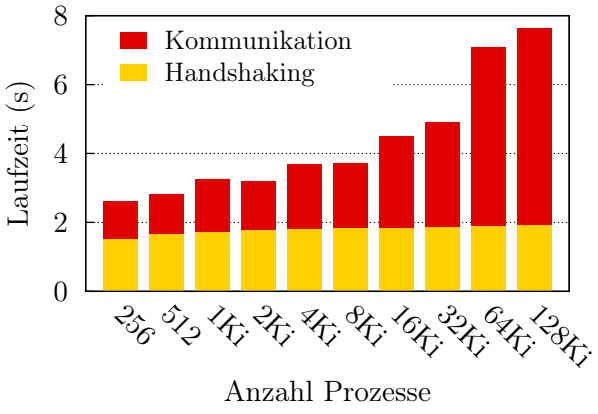
Im Vergleich zum Randaustausch ist das Transfervolumen der Migration deutlich geringer. In Abbildung 5.12(f) ist dargestellt, dass dieses jedoch mit der Prozessanzahl zunimmt. Dabei wird fast das theoretische Maximum von 0,44 GiB erreicht, welches sich unter Annahme der Umverteilung aller Blöcke je COSMO-Schritt berechnen lässt. Die Zunahme lässt sich damit erklären, dass der Bereich, über den die untere und obere Grenze einer Partition auf der raumfüllenden Kurve während eines Laufs variieren, mit steigender Parallelität zunimmt. Dabei werden jedoch



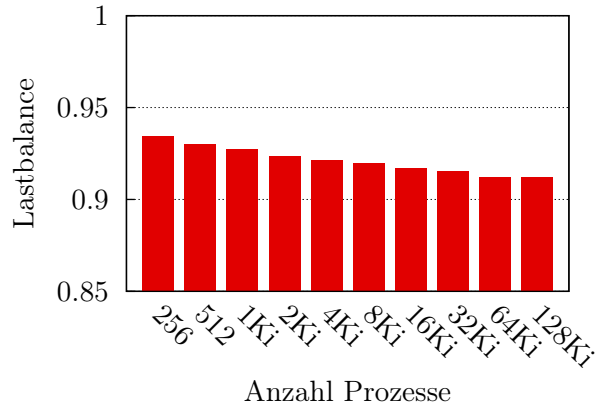
(a) Gesamtlaufzeit COSMO-SPECS+FD4



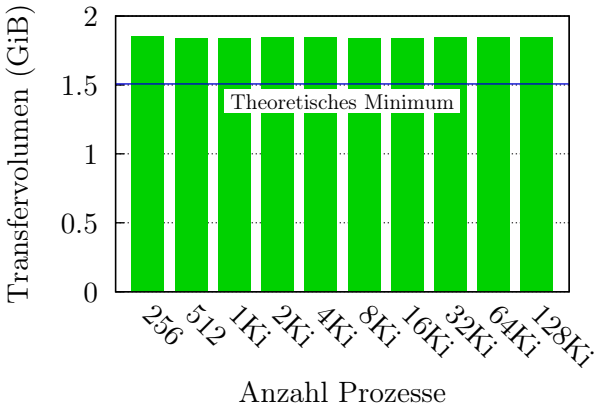
(b) Laufzeit der FD4-Lastbalancierung



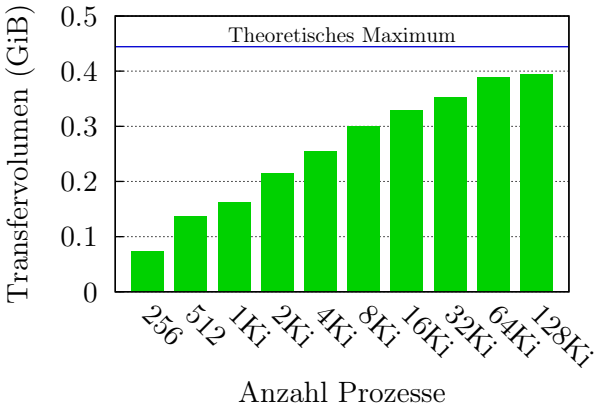
(c) Laufzeit der FD4-Modellkopplung



(d) Durchschnittliche Lastbalance



(e) Transfervolumen des Randaustauschs



(f) Transfervolumen der Migration

Abbildung 5.12: Skalierbarkeit (*weak scaling*) von COSMO-SPECS+FD4 (IBM BlueGene/P). Dargestellt sind Durchschnittswerte über alle Prozesse (außer Lastbalance).

die meisten Blöcke zwischen direkten Nachbarprozessen (bezogen auf den MPI-Rank) oder nahe benachbarten Prozessen ausgetauscht. Dadurch entsteht das in Abschnitt 5.2.3 beobachtete lokale Kommunikationsmuster, welches typischerweise hohe Bandbreiten ermöglicht. Somit ist der Aufwand der Migration trotz Umverteilung fast aller Blöcke je Zeitschritt auch bei 131 072 Prozessen mit 2,2 % moderat. Für eine höhere Skalierbarkeit sollte die Zahl der migrierten Blöcke jedoch verringert werden. Dies ermöglicht *r-DISTR-BISECT*, wie im folgenden Abschnitt gezeigt wird. Darüber hinaus kann die Migration durch stabilere Blockgewichte reduziert werden, welche weniger stark durch die hier genutzte Zeitmessung zufällig variieren.

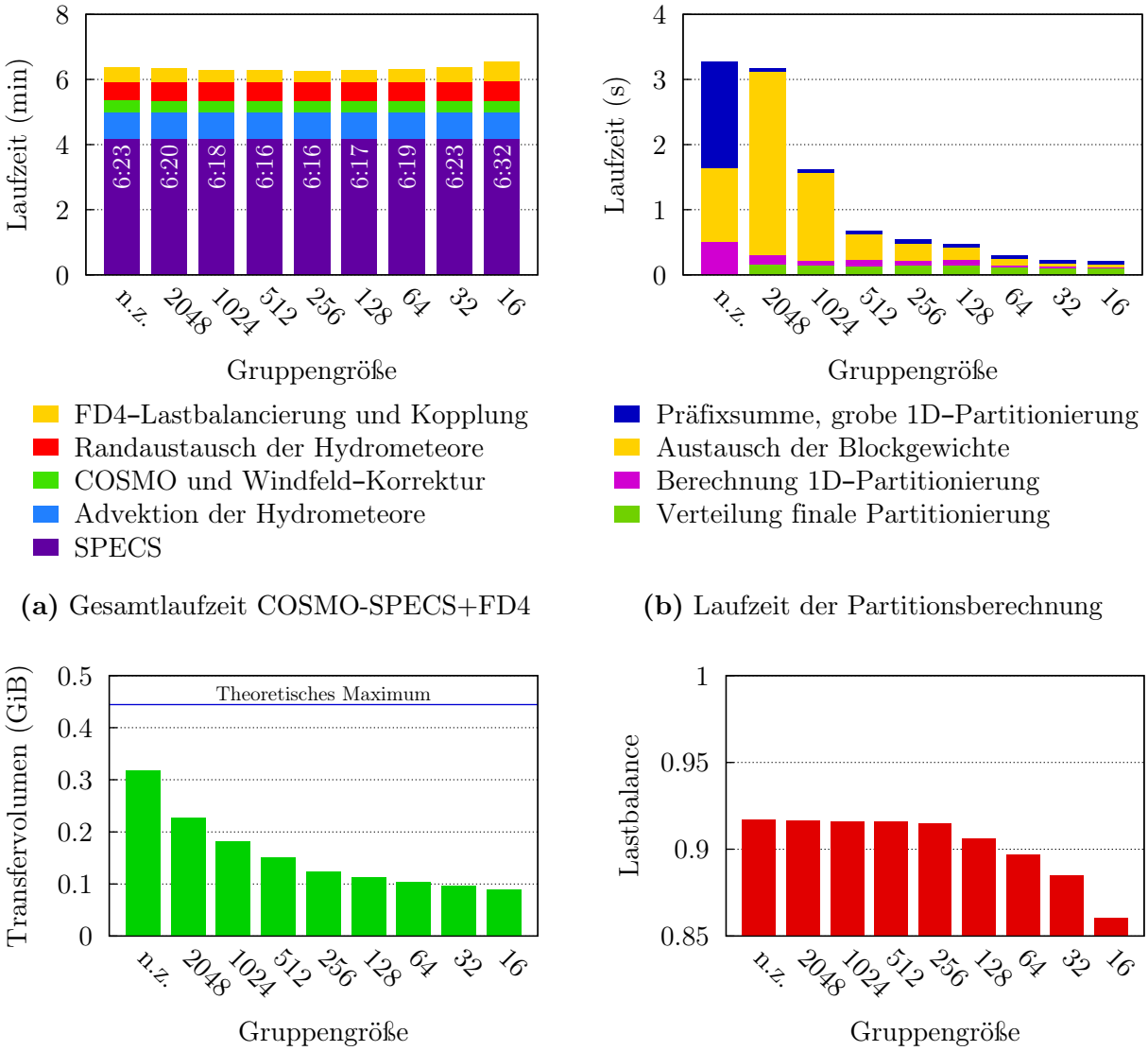
5.2.5 Evaluierung des verteilten 1D-Partitionierungsverfahrens

Die Skalierbarkeit des in Abschnitt 4.3.3 eingeführten verteilten eindimensionalen Partitionierungsverfahrens *r-DISTR-BISECT* wurde mit derselben Modellkonfiguration von COSMO-SPECS+FD4 wie im vorigen Abschnitt evaluiert. Zum Vergleich mit dem bisher betrachteten Verfahren *r-PAR-BISECT* und verschiedener Gruppengrößen für *r-DISTR-BISECT* wurden Messungen mit 16 384 Prozessen auf der IBM BlueGene/P durchgeführt. Die Gruppengröße P/Q wurde dabei von 2048 bis 16 variiert, womit eine Gruppenanzahl Q von 8 bis 1024 untersucht wurde. Die Ergebnisse dieser Messungen sind in Abbildung 5.13 für die Laufzeit des Modells sowie dreier Metriken für den Leistungsvergleich der Partitionierungsverfahren dargestellt. Die erste der Säulen stellt dabei jeweils die Messwerte von *r-PAR-BISECT* dar.

Abbildung 5.13(a) zeigt, dass eine Reduktion der Gesamtlaufzeit durch das verteilte Partitionierungsverfahren erfolgt, wobei diese insbesondere durch einen geringeren Zeitbedarf von FD4 zurückzuführen ist. Insgesamt beträgt der Anteil der FD4-Lastbalancierung (Partitionsberechnung und Migration, ohne Synchronisation) an der Gesamtlaufzeit von COSMO-SPECS+FD4 1,6 % mit *r-PAR-BISECT* und 0,4 % mit *r-DISTR-BISECT* bei einer Gruppengröße von 256. Die geringste Gesamtlaufzeit wird bei den Gruppengrößen 256 und 512 erreicht. Sowohl bei größer werdenden als auch bei kleiner werdenden Gruppen erhöht sich die Laufzeit von FD4. Die Ursachen dafür gehen aus den Abbildungen 5.13(b) bis 5.13(d) hervor. Bei Verringerung der Gruppengröße reduziert sich die Laufzeit der Partitionsberechnung und zudem das Transfervolumen der Migration. Andererseits sinkt ab einer Gruppengröße kleiner als 256 die erreichte Lastbalance deutlich und somit steigt die Zeit für die Synchronisation zur Lastbalancierung.

Die Reduktion der Zeit für die Partitionsberechnung gegenüber *r-PAR-BISECT* ist vor allem auf die parallele Berechnung der Präfixsumme und dem mit der Gruppengröße skalierenden Aufwand für den Austausch der Blockgewichte zurückzuführen. Der Austausch der Blockgewichte zwischen allen 16 384 Prozessen ist jedoch schneller als innerhalb der Gruppen der Größe 1024 und 2048 Prozesse, was zuerst unlogisch erscheint. Dieses Verhalten ist dadurch zu erklären, dass das Baumnetzwerk der IBM BlueGene/P nur von kollektiven Kommunikationsoperationen verwendet wird, an denen alle Prozesse beteiligt sind [103]. Andernfalls wird die Kommunikation über das Torusnetzwerk durchgeführt, welches weniger für kollektive Operationen mit hoher Prozessanzahl geeignet ist.

Das Verhalten des Migrationsaufwands ist ein Resultat des steigenden Einflusses der Heuristik *H1* gegenüber dem Verfahren *r-PAR-BISECT*. Mit Verringerung der Gruppenzahl gewinnt *H1* für die Partitionierung an Bedeutung. In Abschnitt 5.2.1 wurde der geringere Migrationsaufwand der Heuristiken im Vergleich zu *r-BISECT* bereits diskutiert. Die sinkende Lastbalance ist ebenfalls auf den steigenden Einfluss der Heuristik *H1* zurückzuführen, welche bei einer hohen Anzahl an grobgranularen Partitionen eine sinkende Lastbalance liefert. Auch dieses Ergebnis ist nach den Betrachtungen in den Abschnitten 4.3.3 und 5.2.1 nicht unerwartet.



(a) Gesamtlaufzeit COSMO-SPECS+FD4

(b) Laufzeit der Partitionsberechnung

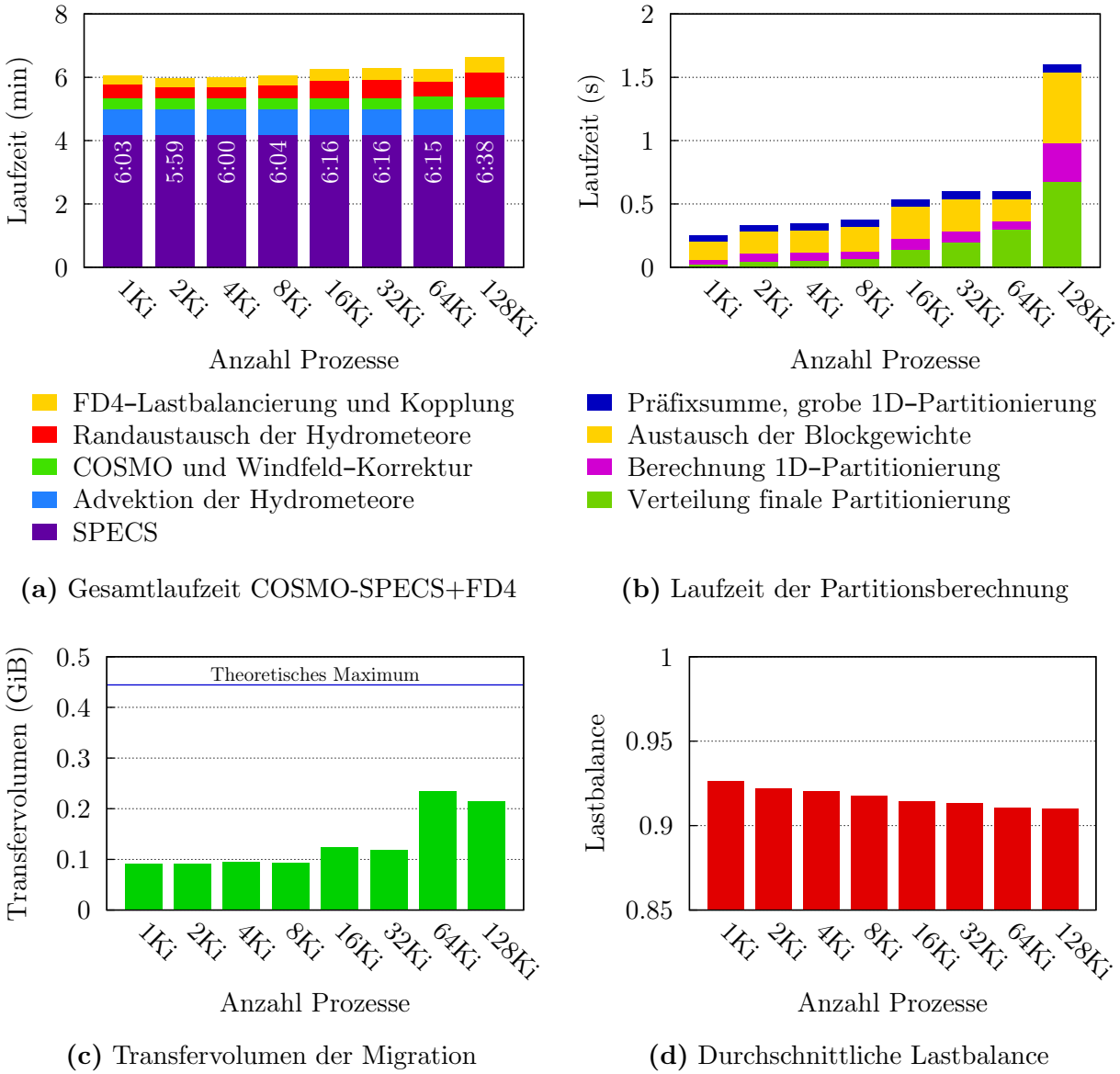
(c) Transfervolumen der Migration

(d) Durchschnittliche Lastbalance

Abbildung 5.13: Vergleich von *r-PAR-BISECT* (erste Säule) und *r-DISTR-BISECT* mit verschiedenen Gruppengrößen mit COSMO-SPECS+FD4 bei 16 384 Prozessen (IBM BlueGene/P).

Die Skalierbarkeitsmessungen des vorherigen Abschnittes wurden mit *r-DISTR-BISECT* und einer festen Gruppengröße von 256 Prozessen wiederholt. Abbildung 5.14(a) zeigt eine Verbesserung der Skalierbarkeit gegenüber der Partitionierung mit *r-PAR-BISECT*, was auf einen niedrigeren Anteil von FD4 zurückzuführen ist. Der Anteil der Lastbalancierung ohne Synchronisation wurde bei 131 072 Prozessen durch das verteilte Verfahren von 8,3 % auf 1,0 % reduziert.

Eine genauere Analyse der Phasen des Partitionierungsverfahrens *r-DISTR-BISECT* zeigt Abbildung 5.14(b). Die ersten drei Phasen des Verfahrens (parallele Präfixsumme, Berechnung der groben Partitionierung, Kommunikation der grobgranularen Partitionsgrenzen – siehe Abschnitt 4.3.3) zeigen eine sehr gute Skalierbarkeit, was vor allem durch die hocheffiziente Implementierung von `MPI_Exscan` auf der IBM BlueGene/P ermöglicht wird. Damit konnte der größte Engpass der Implementierung von *r-PAR-BISECT* in FD4, die serielle Berechnung der Präfixsumme, beseitigt werden. Durch die feste Gruppengröße weist der sich daran anschließende Austausch der Blockgewichte (d. h. der aufsummierten Gewichtsvektoren) innerhalb der Gruppen ebenfalls eine sehr gute Skalierbarkeit auf. Nur bei der höchsten Prozessanzahl steigt der An-



(a) Gesamtlaufzeit COSMO-SPECS+FD4

(b) Laufzeit der Partitionsberechnung

(c) Transfervolumen der Migration

(d) Durchschnittliche Lastbalance

Abbildung 5.14: Skalierbarkeit (*weak scaling*) von COSMO-SPECS+FD4 mit dem Partitionierungsverfahren *r-DISTR-BISECT* bei Gruppengröße von 256 (IBM BlueGene/P).

teil deutlich; die Ursache dafür ist jedoch unklar. Die Zeit für die Partitionsberechnung innerhalb der Gruppen steigt bei 131 072 Prozessen ebenfalls stark an. Eine detaillierte Performanceanalyse in Abbildung 5.15 zeigt, dass der asynchrone Austritt aus dem Austausch der Blockgewichte hohe Wartezeiten bei dem in *r-PAR-BISECT* gerufenen `MPI_Allreduce` verursacht, womit dieses Verhalten erklärt werden kann. Die einzige nicht skalierbare Phase von *r-DISTR-BISECT*, die Verteilung der finalen Partitionierung an alle Prozesse mittels `MPI_Allgather`, steigt im Aufwand ca. um das 29-fache bei einer 128-fachen Erhöhung der Prozessanzahl. Aber auch hier führen Imbalancen zwischen den Gruppen zu erhöhter Synchronisationszeit.

Der zweite Grund der Beschleunigung gegenüber *r-PAR-BISECT* ist der reduzierte Migrationsaufwand. Abbildung 5.14(c) zeigt, dass das Transfervolumen etwa auf die Hälfte zurückgeht. Dies führt zu einer noch stärkeren Reduktion der Kommunikationszeit, da sich die Partitionsgrenzen um weniger Blöcke bzw. Partitionen verschieben. Somit werden seltener Blöcke weiter als bis zum Nachbarprozess gesendet, was im Mittel höhere Bandbreiten generiert.

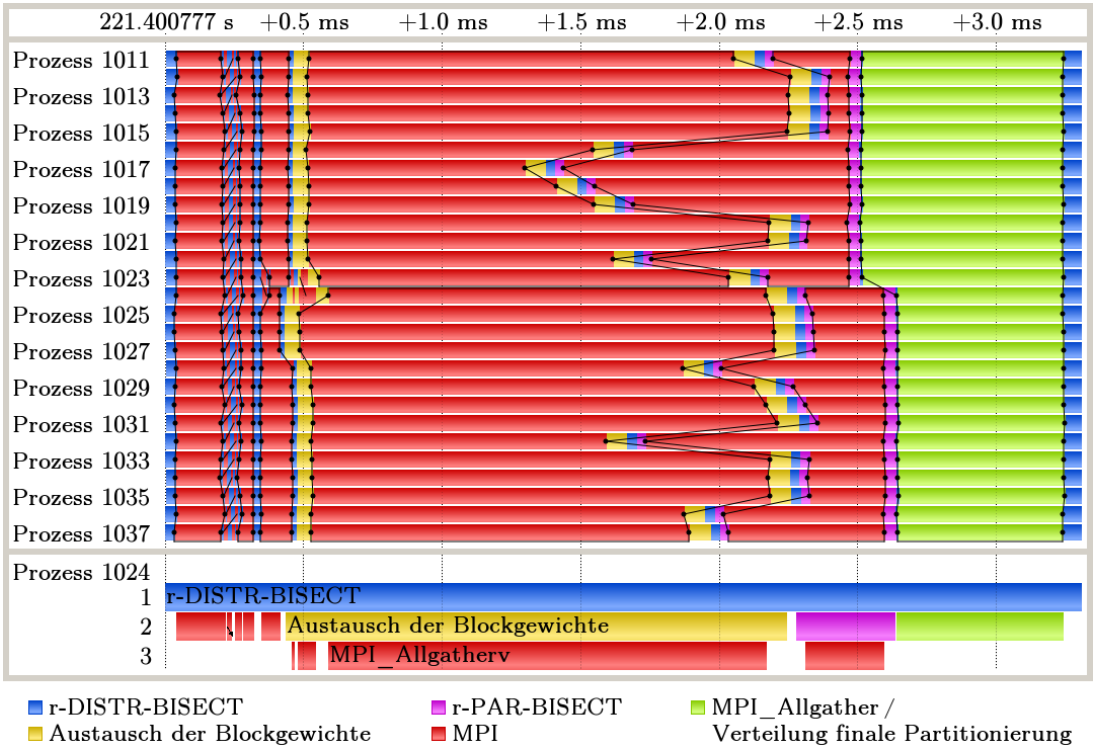


Abbildung 5.15: Detaillierte Analyse von *r-DISTR-BISECT* mit Gruppengröße 256 anhand eines Ausschnitts einer *Vampir-Timeline* von COSMO-SPECS+FD4 bei 16 384 Prozessen. Zwischen Prozess 1023 und 1024 befindet sich eine Gruppengrenze. Die Farben entsprechen einzelnen Phasen der Partitionsberechnung. Punkt-zu-Punkt-Nachrichten sind durch einfache Linien, kollektive Operationen durch Linien mit Punkten dargestellt.

Die in Abbildung 5.14(d) dargestellte Lastbalance verhält sich nahezu identisch wie bei den Messungen im vorigen Abschnitt. Trotz der Kombination mit einer weniger genauen Heuristik zur Partitionsberechnung liegt die Balance bei den durchgeführten Messungen mit *r-DISTR-BISECT* nie unter 99,7 % des Werts von *r-PAR-BISECT*.

Diese Messergebnisse zeigen, dass das zweistufige Verfahren *r-DISTR-BISECT* zur verteilten Partitionsberechnung die Skalierbarkeit effektiv und nahezu ohne Abstriche bei der erreichten Lastbalance verbessert. Dies wird durch die Verknüpfung einer skalierbaren Heuristik mit einem sehr genauen Suchalgorithmus zu einem hierarchischen Verfahren erreicht. Sowohl der Aufwand der Partitionsberechnung als auch der Migration wurde gegenüber der ausschließlichen Verwendung des genauen Verfahrens *r-PAR-BISECT* deutlich verringert. Auf 131 072 Prozessen konnte dadurch die Laufzeit von COSMO-SPECS+FD4 von 431 s auf 398 s reduziert werden.

6 Zusammenfassung und Ausblick

In dieser Arbeit wird eine neue Art der Kopplung spektraler Wolkenmikrophysikmodelle an atmosphärische Modelle entworfen, mit dem Ziel, die Skalierbarkeit und Effizienz auf aktuellen Hochleistungsrechnern deutlich zu verbessern. Dies wird durch eine dynamische Lastbalancierung der Mikrophysik erreicht, welche aufgrund des räumlich und zeitlich stark variierenden Rechenaufwands unumgänglich ist. Da die Lastbalancierung unabhängig vom atmosphärischen Modell ist, sind keine Änderungen an den Datenstrukturen des atmosphärischen Modells notwendig. Basierend auf einer Analyse der Defizite anhand eines bestehenden Modells mit spektraler Mikrophysik werden an die Problemstellung angepasste Datenstrukturen, Lastbalancierungsverfahren und Kopplungsmechanismen entworfen, die aufeinander abgestimmt sind und eine hohe Skalierbarkeit erreichen. Ein großer Teil der Konzepte werden in der Bibliothek FD4 implementiert, welche zur Optimierung der Kopplung des Modellsystems COSMO-SPECS genutzt wird. Mit Leistungsmessungen wird gezeigt, dass dadurch die Ausführungszeit mehr als halbiert wird, was vor allem auf das Beseitigen der Lastimbalancen und die Optimierung der Kommunikation zurückzuführen ist. Trotz des zusätzlichen Aufwands der dynamischen Lastbalancierung und der Kopplung zwischen unterschiedlichen Partitionierungen wird eine Skalierbarkeit bis 131 072 Prozessorkerne nachgewiesen. Zudem erweisen sich beim Vergleich zweier grundlegender Lastbalancierungsverfahren raumfüllende Kurven als weitaus skalierbarer als graphenbasierte Verfahren. Die wesentlichen Beiträge der Arbeit lassen sich wie folgt zusammenfassen:

- Es wird ein integriertes Konzept zur Dekomposition, Parallelisierung und dynamischen Lastbalancierung von detaillierten Wolkenmikrophysikmodellen sowie deren Kopplung an atmosphärische Simulationsmodelle entworfen. Es beruht auf der Separation der beiden Modellkomponenten und ist auch in weiteren Bereichen anwendbar.
- Ein bestehender eindimensionaler Partitionierungsalgorithmus zur Lastbalancierung mittels raumfüllender Kurven wird dahingehend erweitert, dass eine spezifizierte Mindestanforderung an die Lastbalance eingehalten wird. Damit kann der Rechenaufwand im Vergleich zu exakten Verfahren deutlich reduziert, aber dennoch eine vorgegebene Lastbalance garantiert werden.
- Es wird ein hierarchischer eindimensionaler Partitionierungsalgorithmus entwickelt, welcher durch Kombination mit einer skalierbaren Heuristik das genaue Partitionierungsverfahren verteilt berechnet. Dadurch wird die Skalierbarkeit der Repartitionierung – nahezu ohne Abstriche bezüglich der erzielten Lastbalance – deutlich verbessert.
- Die Arbeit zeigt erstmals die hochskalierbare Verknüpfung von dynamischer Lastbalancierung und Modellkopplung, wofür ein effizientes und skalierbares Verfahren für die regelmäßige Bestimmung der Überschneidungen zwischen den unterschiedlichen Partitionierungen der gekoppelten Modelle entwickelt wird.

Durch die effiziente Nutzung von Hochleistungsrechnern ermöglichen diese Ergebnisse die Anwendung spektraler Wolkenmikrophysik zur Simulation realistischer Szenarien auf hochaufgelösten Gittern. Die effizienten Verfahren zur Lastbalancierung und Kopplung sind auch bei weniger rechenintensiven spektralen Modellen lohnenswert. Die entwickelten Konzepte sind darüber hinaus auch für weitere Bereiche relevant, bei denen eine Kombination aus Modellkopplung und Lastbalancierung zum Erreichen hoher Skalierbarkeit notwendig ist. Diese Anwendungen finden

sich vor allem im Bereich der Multiphysik- und Multiphasen-Simulation, wie zum Beispiel die Simulation chemischer Prozesse in der Atmosphäre [121], die Simulation reaktiver Strömungen im Grundwasser [37] sowie ingenieurwissenschaftliche Simulationen zum Entwurf von Turbinen, Triebwerken und Verbrennungsmotoren.

Aufbauend auf der Arbeit sind weitere Entwicklungen zur Verbesserung der Skalierbarkeit möglich und für zukünftige Rechnergenerationen sinnvoll. Für die dynamische Lastbalancierung sollten Methoden untersucht werden, welche die Replikation des vollständigen Partitionsvektors auf allen Prozessen vermeiden. Zum einen entfällt so eine nichtskalierbare Datenstruktur und zum anderen kann auf die Kommunikation des kompletten Partitionsvektors an alle Prozesse verzichtet werden, was als der stärkste Skalierbarkeitsengpass des verteilten Partitionierungsverfahrens identifiziert wurde. Des Weiteren sollte die Zahl migrierter Blöcke durch verbesserte Methoden zur Bestimmung der Gewichte reduziert werden, welche geringeren zufälligen Schwankungen ausgesetzt sind als eine Zeitmessung. Eine relativ einfache Methode wäre beispielsweise die Einführung eines Schwellwerts für die relative Gewichtsänderung eines Blocks bei aufeinanderfolgenden Zeitschritten. Darüber hinaus könnte auch untersucht werden, inwiefern mit einem Performance-Modell für die spektrale Mikrophysik die Abschätzung der zukünftigen Rechenlast möglich ist. Damit ließe sich sowohl die Migration als auch die Lastimbalance weiter reduzieren.

Eine weitere Verbesserung der Skalierbarkeit des Wolkenmikrophysikmodells kann durch eine hybride Parallelisierung erreicht werden, welche die bestehende *distributed-memory*-Parallelisierung um eine *shared-memory*-Parallelisierung erweitert. Dafür ließe sich beispielsweise die bereits bestehende feingranulare Blockzerlegung nutzen. Durch die hybride Parallelisierung werden der Aufwand der dynamischen Lastbalancierung und die Menge globaler Metadaten reduziert. Ein besonderes Skalierbarkeitsproblem des *strong scaling* stellt die in Relation zum Berechnungsaufwand steigende Kommunikation zwischen benachbarten Partitionen dar. Dem kann mit der Überlappung von Kommunikation und Berechnung begegnet werden, wie es in Abschnitt 4.2.3 vorgeschlagen wird.

Das Thema parallele Ein-/Ausgabe wird in dieser Arbeit nicht behandelt, ist jedoch für eine hohe Skalierbarkeit sehr wichtig. In diesem Kontext ist bereits eine Implementierung basierend auf den entwickelten Verfahren der Kopplung erfolgt. Dabei werden die irregulär verteilten Daten in FD4 in eine für parallele Ausgabe geeignete Dekomposition transformiert und parallel in eine Datei geschrieben. Eine Evaluierung und weitere Optimierung dieser Methode, etwa durch automatische Ermittlung der günstigsten Dekomposition, steht noch aus.

Zwei-Momenten- oder Mehr-Momenten-Verfahren sind im Vergleich zu spektralen Ansätzen wesentlich schneller, führen jedoch ebenfalls zu Lastimbalancen. Zu deren effizienten Balancierung müssen deutlich schnellere Methoden entwickelt werden. Dabei wird es kaum möglich sein globale Lastbalance sicherzustellen, da die dafür notwendige globale Abstimmung voraussichtlich zu viel Zeit beansprucht. Daher sind lokale Verfahren in diesem Kontext besser geeignet.

Eine Leistungssteigerung atmosphärischer Modelle mit spektraler Wolkenmikrophysik kann durch Zeitintegrationsverfahren mit adaptiver Schrittweite erfolgen, um so die heterogene Dynamik von Wolkenprozessen zu berücksichtigen. Zudem ist es denkbar, für jede Gitterzelle zur Laufzeit zu entscheiden, ob die Mikrophysik mit einem spektralen Modell oder einer Parametrisierung berechnet werden soll, um bei untersättigten Verhältnissen sowohl Rechenzeit als auch Speicher einzusparen [62]. Beide Methoden führen zu einer weiteren Steigerung der möglichen Lastungleichgewichte zwischen einzelnen Gitterzellen und setzen somit für eine effiziente Anwendung das hier entwickelte Konzept zur dynamischen Lastbalancierung zwingend voraus.

Abbildungsverzeichnis

Abkürzungsverzeichnis

COSMO	*Consortium for Small-scale Modeling* Internationales Konsortium, welches ein gleichnamiges atmosphärisches Modell entwickelt.
FD4	*Four-dimensional distributed dynamic data structures* Softwareframework, in dem Konzepte der vorliegenden Arbeit implementiert und bewertet wurden.
MM5	*Fifth-generation Penn State/NCAR Mesoscale Model* Atmosphärisches Modell, welches an der Pennsylvania State University und dem National Center for Atmospheric Research entwickelt wurde.
MPI	*Message Passing Interface* Programmierstandard für Parallelrechner mit verteiltem Speicher.
MUSCAT	*Multiscale Chemistry Aerosol Transport* Chemie-Transport-Modell des Leibniz-Instituts für Troposphärenforschung Leipzig.
RSL	*Runtime System Library* In MM5 genutzte Bibliothek zur Parallelisierung.
SFC	*Space-filling curve* Raumfüllende Kurven sind stetige Kurven, welche jeden Punkt eines mehrdimensionalen Gebiets durchlaufen.
SPECS	*Spectral bin cloud microphysics* Spektrales Wolkenmikrophysikmodell des Leibniz-Instituts für Troposphärenforschung Leipzig.
WRF	*Weather Research & Forecasting Model* Atmosphärisches Modell, welches als Gemeinschaftsprojekt vor allem von US-amerikanischen Organisationen entwickelt wird.

Symbolverzeichnis

b_d	Kantenlänge eines Blocks in Gitterzellen je Dimension $d \in \{x, y, z\}$
β	Summe der Gewichte der maximal belasteten Partition (*bottleneck value*)
β^*	Idealer *bottleneck value* $\beta^* = \sum_{i=1}^{B} w_i / P$ (alle Partitionen haben gleiche Last)
B	Gesamtzahl der Blöcke
B_d	Anzahl der Blöcke je Raumdimension $d \in \{x, y, z\}$ im Blockgitter
Δt^{dyn}	Zeitschrittgröße des atmosphärischen Modells
Δt^{mp}	Zeitschrittgröße des Wolkenmikrophysikmodells
ϵ	Maximal erlaubter Abstand eines *bottleneck value* β vom minimal möglichen *bottleneck value* β_{opt}, $\epsilon \geq \beta - \beta_{opt}$
g	Anzahl der Schichten an Geisterzellen für den Randaustausch
k	Verfeinerungsfaktor einer raumfüllenden Kurve, üblicherweise $k \in \{2, 3\}$
l	Stufe einer raumfüllenden Kurve
K	Gesamtzahl der Koppelfelder
λ^{-1}	Lastimbalance der Blockgewichte, $\lambda^{-1} = B\, w_{max} / \sum_{i=1}^{B} w_i$
Λ	Lastbalance einer Partitionierung, $\Lambda = \beta^* / \beta$
m	Anzahl der Zeitschritte des Wolkenmikrophysikmodells je Zeitschritt des atmosphärischen Modells, $m = \Delta t^{dyn} / \Delta t^{mp}$
P	Anzahl der Prozesse bzw. Partitionen
Q	Anzahl grobgranularer Partitionen bzw. Gruppen des verteilten 1D-Partitionierungsverfahrens *r-DISTR-BISECT*, $Q < P$
r	Verhältnis der erzielten Lastbalance zur optimalen Lastbalance, $r = \Lambda_{res} / \Lambda_{opt}$
s_p	Partitionsvektor, enthält für jeden Prozess $p = 1, 2, \ldots, P$ dessen Startindex auf dem durch eine raumfüllende Kurve gebildeten Blockvektor, $s_{P+1} = B + 1$
u_q	Partitionsvektor der Gruppen des Verfahrens *r-DISTR-BISECT*, enthält für jede Gruppe $q = 1, 2, \ldots, Q$ deren Startindex auf dem Blockvektor, $u_{Q+1} = B + 1$
w_i	Gewicht des i-ten Blocks für $i = 1, 2, \ldots, B$
w_{max}	Maximales Blockgewicht in w_i
W_j	Präfixsumme des Gewichtsvektors, $W_j = \sum_{i=1}^{j} w_i$, $j = 1, 2, \ldots, B$, $W_0 = 0$
W_j^q	Präfixsumme des Gewichtsvektors einer Gruppe q des Verfahrens *r-DISTR-BISECT*, $W_j^q = \sum_{i=u_q}^{u_q+j-1} w_i$, $j = 1, 2, \ldots, u_{q+1} - u_q$, $W_0^q = 0$

Literaturverzeichnis

[1] Adamidis, P., I. Fast und T. Ludwig: *Performance Characteristics of Global High-Resolution Ocean (MPIOM) and Atmosphere (ECHAM6) Models on Large-Scale Multicore Cluster*. In: Malyshkin, V. (Hrsg.): *Parallel Computing Technologies*, Bd. 6873 d. Reihe *Lecture Notes in Computer Science*, S. 390–403. Springer, 2011.

[2] Adrian, G. und D. Frühwald: *Design der Modellkette GME/LM*. Promet: Die neue Modellkette des DWD I, 27(3/4):106–110, 2002.

[3] Aftosmis, M., M. Berger und S. Murman: *Applications of Space-Filling Curves to Cartesian Methods for CFD*. In: *42nd Aerospace Sciences Meeting and Exhibit*, 2004.

[4] Arnold, L., C. Beetz, J. Dreher, H. Homann, C. Schwarz und R. Grauer: *Massively Parallel Simulations of Solar Flares and Plasma Turbulence*. In: Bischof, C. et al. (Hrsg.): *Parallel Computing: Architectures, Algorithms and Applications*, Bd. 15 d. Reihe *Advances in Parallel Computing*, S. 467–474. IOS Press, 2008.

[5] Beheng, K. D. und U. Wacker: *Über die Mikrostruktur von Wolken*. Promet: Wolkenphysik und Wolkendynamik I, 23(1/2):10–15, 1993.

[6] Behrens, J., N. Rakowsky, W. Hiller, D. Handorf, M. Läuter, J. Päpke und K. Dethloff: *amatos: Parallel adaptive mesh generator for atmospheric and oceanic simulation*. Ocean Modelling, 10(1-2):171–183, 2005.

[7] Bekas, C., A. Curioni, P. Arbenz, C. Flaig, G. H. Van Lenthe, R. Müller und A. J. Wirth: *Extreme scalability challenges in micro-finite element simulations of human bone*. Concurrency and Computation: Practice and Experience, 22(16):2282–2296, 2010.

[8] Berger, M. und S. Bokhari: *A Partitioning Strategy for Nonuniform Problems on Multiprocessors*. IEEE Transactions on Computers, C-36(5):570–580, 1987.

[9] Bertrand, F., R. Bramley, A. Sussman, D. E. Bernholdt, J. A. Kohl, J. W. Larson und K. B. Damevski: *Data Redistribution and Remote Method Invocation in Parallel Component Architectures*. In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS 2005)*, S. 40b. IEEE Computer Society, 2005.

[10] Böhme, T., N. Van Lipzig, L. Delobbe, E. Goudenhoofdt und A. Seifert: *Evaluation of microphysical assumptions of the COSMO model using radar and rain gauge observations*. Meteorologische Zeitschrift, 20(2):133–144, 2011.

[11] Buis, S., A. Piacentini und D. Déclat: *PALM: a computational framework for assembling high-performance computing applications*. Concurrency and Computation: Practice and Experience, 18(2):231–245, 2006.

[12] Bulatewicz, T.: *Support for model coupling: An interface-based approach*. Dissertation, University of Oregon, 2006.

[13] Burstedde, C., O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox und S. Zhong: *Scalable adaptive mantle convection simulation on petascale supercomputers*. In: *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, S. 62:1–62:15. IEEE Press, 2008.

[14] Catalyurek, U. V., E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy und L. A. Riesen: *A repartitioning hypergraph model for dynamic load balancing*. Journal of Parallel and Distributed Computing, 69(8):711–724, 2009.

[15] Chandra, S., X. Li, T. Saif und M. Parashar: *Enabling scalable parallel implementations of structured adaptive mesh refinement applications.* Journal of Supercomputing, 39(2):177–203, 2007.

[16] Chapman, B., G. Jost und R. van der Pas: *Using OpenMP*. MIT Press, 2007.

[17] Chevalier, C. und F. Pellegrini: *PT-Scotch: A tool for efficient parallel graph ordering.* Parallel Computing, 34(6-8):318–331, 2008.

[18] Chin, M., R. A. Kahn und S. E. Schwartz (Hrsg.): *Atmospheric Aerosol Properties and Climate Impacts.* U.S. Climate Change Science Program and the Subcommittee on Global Change Research, NASA, 2009.

[19] Cormen, T. H., C. E. Leiserson, R. L. Rivest und C. Stein: *Introduction to Algorithms*. MIT Press, 3. Aufl., 2009.

[20] Corradi, A., L. Leonardi und F. Zambonelli: *Diffusive load-balancing policies for dynamic applications.* IEEE Concurrency, 7(1):22–31, 1999.

[21] Cuvelier, C. et al.: *CityDelta: A model intercomparison study to explore the impact of emission reductions in European cities in 2010.* Atmospheric Environment, 41(1):189–207, 2007.

[22] Cybenko, G.: *Dynamic load balancing for distributed memory multiprocessors.* Journal of Parallel and Distributed Computing, 7(2):279–301, 1989.

[23] Dennis, J. M., M. Vertenstein und A. P. Craig: *Performance Evaluation of Ultra-High-Resolution Climate Simulations.* In: *10th LCI International Conference on High-Performance Clustered Computing*, 2009.

[24] Devine, K., E. Boman, R. Heaphy, B. Hendrickson und C. Vaughan: *Zoltan Data Management Services for Parallel Dynamic Applications.* Computing in Science and Engineering, 4(2):90–97, 2002.

[25] Diachin, L. F., R. D. Hornung, P. Plassmann und A. M. Wissink: *Parallel Adaptive Mesh Refinement*. In: Heroux, M. A., P. Raghavan und H. D. Simon (Hrsg.): *Parallel processing for scientific computing*, S. 143–162. Cambridge University Press, 2006.

[26] Doms, G. et al.: *A Description of the Nonhydrostatic Regional COSMO-Model, Part II: Physical Parameterization*. Consortium for Small-scale Modelling, 2011. `http://www.cosmo-model.org/content/model/documentation/core/default.htm`.

[27] Dongarra, J., I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon und A. White (Hrsg.): *Sourcebook of Parallel Computung*. Morgan Kaufmann, 2003.

[28] Elbern, H.: *Parallelization and load balancing of a comprehensive atmospheric chemistry transport model.* Atmospheric Environment, 31(21):3561–3574, 1997.

[29] Etling, D.: *Theoretische Meteorologie*. Springer, 3. Aufl., 2008.

[30] Farhat, C. und M. Lesoinne: *Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics.* International Journal for Numerical Methods in Engineering, 36(5):745–764, 1993.

[31] Feichter, J.: *Aerosole und das Klimasystem.* Physik in unserer Zeit, 34:72–79, 2003.

[32] Foster, I. T. und B. R. Toonen: *Load-Balancing Algorithms for Climate Models.* In: *Proceedings of the IEEE Scalable High-Performance Computing Conference*, S. 674–681, 1994.

[33] Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Sankt Augustin: *MpCCI 4.1.0 Documentation*, 2011. `http://www.mpcci.de/mpcci/manuals.html`.

[34] Grell, G. A., S. E. Peckham, R. Schmitz, S. A. McKeen, G. Frost, W. C. Skamarock und B. Eder: *Fully coupled "online" chemistry within the WRF model.* Atmospheric Environment, 39(37):6957–6975, 2005.

[35] Grützun, V.: *Influence of Aerosol Particles on Deep Convective Clouds: Investigations with the New Model LM–SPECS*. Dissertation, Leibniz-Institut für Troposphärenforschung, Leipzig, 2009.

[36] Grützun, V., O. Knoth und M. Simmel: *Simulation of the influence of aerosol particle characteristics on clouds and precipitation with LM–SPECS: Model description and first results.* Atmospheric Research, 90(2-4):233–242, 2008.

[37] Hammond, G., P. Lichtner und C. Lu: *Subsurface multiphase flow and multicomponent reactive transport modeling using high-performance computing.* Journal of Physics: Conference Series, 78(1):012025, 2007.

[38] Heinold, B., I. Tegen, R. Wolke, A. Ansmann, I. Mattis, A. Minikin, U. Schumann und B. Weinzierl: *Simulations of the 2010 Eyjafjallajökull volcanic ash dispersal over Europe using COSMO-MUSCAT.* Atmospheric Environment, 48(0):195 – 204, 2012.

[39] Heise, E.: *Parametrisierungen.* Promet: Die neue Modellkette des DWD I, 27(3/4):130–141, 2002.

[40] Hendrickson, B. und K. Devine: *Dynamic load balancing in computational mechanics.* Computer Methods in Applied Mechanics and Engineering, 184(2-4):485–500, 2000.

[41] Holtgrewe, M., P. Sanders und C. Schulz: *Engineering a Scalable High Quality Graph Partitioner.* In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS 2010)*. IEEE Computer Society, 2010.

[42] Jacob, R., J. Larson und E. Ong: *M x N Communication and Parallel Interpolation in Community Climate System Model Version 3 Using the Model Coupling Toolkit.* International Journal of High Performance Computing Applications, 19(3):293–307, 2005.

[43] Jacobson, M. Z. und D. L. Ginnebaugh: *Global-through-urban nested three-dimensional simulation of air pollution with a 13,600-reaction photochemical mechanism.* Journal of Geophysical Research, 115:D14304, 2010.

[44] Jin, G. und J. Mellor-Crummey: *SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion.* ACM Transactions on Mathematical Software, 31:120–148, 2005.

[45] Jones, A., D. Thomson, M. Hort und B. Devenish: *The U.K. Met Office's Next-Generation Atmospheric Dispersion Model, NAME III.* In: Borrego, C. und A.-L. Norman (Hrsg.): *Air Pollution Modeling and Its Application XVII*, S. 580–589. Springer, 2007.

[46] Kale, L. V. und G. Zheng: *Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects.* In: Parashar, M. und X. Li (Hrsg.): *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*, Kap. 13, S. 265–282. John Wiley & Sons, Inc., 2009.

[47] Karypis, G. und V. Kumar: *Parallel Multilevel series k-Way Partitioning Scheme for Irregular Graphs.* SIAM Review, 41:278–300, 1999.

[48] Khain, A., B. Lynn und J. Dudhia: *Aerosol Effects on Intensity of Landfalling Hurricanes as Seen from Simulations with the WRF Model with Spectral Bin Microphysics.* Journal of the Atmospheric Sciences, 67(2):365–384, 2010.

[49] Khain, A., M. Ovtchinnikov, M. Pinsky, A. Pokrovsky und H. Krugliak: *Notes on the state-of-the-art numerical modeling of cloud microphysics.* Atmospheric Research, 55(3-4):159–224, 2000.

[50] Khain, A. und A. Pokrovsky: *Simulation of Effects of Atmospheric Aerosols on Deep Turbulent Convective Clouds Using a Spectral Microphysics Mixed-Phase Cumulus Cloud Model. Part II: Sensitivity Study.* Journal of the Atmospheric Sciences, 61(24):2983–3001, 2004.

[51] Khain, A. P.: *Notes on state-of-the-art investigations of aerosol effects on precipitation: a critical review.* Environmental Research Letters, 4(1):015004, 2009.

[52] Kim, D., J. W. Larson und K. Chiu: *Toward Malleable Model Coupling.* Procedia Computer Science, 4:312–321, 2011.

[53] Knoth, O. und R. Wolke: *An explicit-implicit numerical approach for atmospheric chemistry-transport modeling.* Atmospheric Environment, 32(10):1785–1797, 1998.

[54] Knüpfer, A., H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller und W. E. Nagel: *The Vampir Performance Analysis Tool-Set.* In: Resch, M. et al. (Hrsg.): *Tools for High Performance Computing*, S. 139–155. Springer, 2008.

[55] Kogge, P. M. und T. J. Dysart: *Using the TOP500 to trace and project technology and architecture trends.* In: *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, S. 28:1–28:11. ACM, 2011.

[56] Larson, J., R. Jacob und E. Ong: *The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models.* International Journal of High Performance Computing Applications, 19(3):277–292, 2005.

[57] Larson, J. W.: *Ten organising principles for coupling in multiphysics and multiscale models.* In: Read, W., J. W. Larson und A. J. Roberts (Hrsg.): *Proceedings of the 13th Biennial Computational Techniques and Applications Conference, CTAC-2006*, Bd. 48 d. Reihe *ANZIAM J.*, S. C1090–C1111, 2009.

[58] Li, X., W.-K. Tao, A. P. Khain, J. Simpson und D. E. Johnson: *Sensitivity of a Cloud-Resolving Model to Bulk and Explicit Bin Microphysical Schemes. Part I: Comparisons.* Journal of the Atmospheric Sciences, 66(1):3–21, 2009.

[59] Lieber, M., V. Grützun, R. Wolke, M. S. Müller und W. E. Nagel: *FD4: A Framework for Highly Scalable Load Balancing and Coupling of Multiphase Models.* AIP Conference Proceedings, 1281(1):1639–1642, 2010.

[60] Lieber, M., V. Grützun, R. Wolke, M. S. Müller und W. E. Nagel: *Highly Scalable Dynamic Load Balancing in the Atmospheric Modeling System COSMO-SPECS+FD4.* In: Jónasson, K. (Hrsg.): *Applied Parallel and Scientific Computing*, Bd. 7133 d. Reihe *Lecture Notes in Computer Science*, S. 131–141. Springer, 2012.

[61] Lieber, M. und R. Wolke: *Optimizing the coupling in parallel air quality model systems.* Environmental Modelling & Software, 23(2):235–243, 2008.

[62] Lieber, M., R. Wolke, V. Grützun, M. S. Müller und W. E. Nagel: *A framework for detailed multiphase cloud modeling on HPC systems.* In: Chapman, B. et al. (Hrsg.): *Parallel Computing: From Multicores and GPU's to Petascale*, Bd. 19 d. Reihe *Advances in Parallel Computing*, S. 281–288. IOS Press, 2010.

[63] Lin, Y.-L., R. D. Farley und H. D. Orville: *Bulk Parameterization of the Snow Field in a Cloud Model.* Journal of Climate and Applied Meteorology, 22(6):1065–1092, 1983.

[64] Liu, X. und G. Schrack: *Encoding and decoding the Hilbert order.* Software: Practice and Experience, 26:1335–1346, 1996.

[65] Lou, J. Z. und J. D. Farrara: *Performance analysis and optimization on a parallel atmospheric general circulation model code.* Concurrency: Practice and Experience, 10(7):549–565, 1998.

[66] Lüpkes, C.: *Parametrisierung wolkenmikrophysikalischer Prozesse.* Promet: Wolkenphysik und Wolkendynamik I, 23(1/2):34–40, 1993.

[67] Lynch, P.: *The origins of computer weather prediction and climate modeling.* Journal of Computational Physics, 227(7):3431–3444, 2008.

[68] Lynn, B. H., A. P. Khain, J. Dudhia, D. Rosenfeld, A. Pokrovsky und A. Seifert: *Spectral (Bin) Microphysics Coupled with a Mesoscale Model (MM5). Part II: Simulation of a CaPE Rain Event with a Squall Line.* Monthly Weather Review, 133:59–71, 2005.

[69] MacNeice, P., K. M. Olson, C. Mobarry, R. de Fainchtein und C. Packer: *PARAMESH: A parallel adaptive mesh refinement community toolkit.* Computer Physics Communications, 126:330–354, 2000.

[70] Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard, Version 2.2*, 2009. `http://www.mpi-forum.org`.

[71] Meyerhenke, H.: *Dynamic Load Balancing for Parallel Numerical Simulations Based on Repartitioning with Disturbed Diffusion.* In: *Proceedings of the 15th International Conference on Parallel and Distributed Systems (ICPADS '09)*, S. 150–157. IEEE Computer Society, 2009.

[72] Michalakes, J., J. Hacker, R. Loft, M. O. McCracken, A. Snavely, N. J. Wright, T. Spelce, B. Gorda und R. Walkup: *WRF Nature Run.* In: *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, S. 59:1–59:6. ACM, 2007.

[73] Michalakes, J. G.: *MM90: A scalable parallel implementation of the Penn State/NCAR Mesoscale Model (MM5).* Parallel Computing, 23(14):2173–2186, 1997.

[74] Michalakes, J. G.: *RSL: A parallel runtime system library for regional atmospheric models with nesting.* In: Baden, S. B. et al. (Hrsg.): *Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, Bd. 117, S. 59–74. Springer, 2000.

[75] Michalakes, J. G. und R. S. Nanjundiah: *Computational Load in Model Physics of the Parallel NCAR Community Climate Model.* Techn. Ber. ANL/MCS-TM-186, Argonne National Loboratory, 1994.

[76] Miguet, S. und J.-M. Pierson: *Heuristics for 1D rectilinear partitioning as a low cost and high quality answer to dynamic load balancing.* In: Hertzberger, B. und P. Sloot (Hrsg.): *High-Performance Computing and Networking*, Bd. 1225 d. Reihe *Lecture Notes in Computer Science*, S. 550–564. Springer, 1997.

[77] Moon, B., H. V. Jagadish, C. Faloutsos und J. 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.

[78] Morrison, H. und W. W. Grabowski: *Comparison of Bulk and Bin Warm-Rain Microphysics Models Using a Kinematic Framework.* Journal of the Atmospheric Sciences, 64(8):2839–2861, 2007.

[79] Muszala, S., G. Alaghband, J. Hack und D. Connors: *Natural Load Indices (NLI) for scientific simulation.* The Journal of Supercomputing, S. 1–22, 2010.

[80] Nagel, W. E., A. Arnold, M. Weber, H.-C. Hoppe und K. Solchenbach: *VAMPIR: Visualization and Analysis of MPI Resources.* Supercomputer 63, XII(1):69–80, 1996.

[81] Pilkington, J. und S. Baden: *Dynamic partitioning of non-uniform structured workloads with spacefilling curves.* IEEE Transactions on Parallel and Distributed Systems, 7(3):288–300, 1996.

[82] Pinar, A. und C. Aykanat: *Fast optimal load balancing algorithms for 1D partitioning.* Journal of Parallel and Distributed Computing, 64(8):974–996, 2004.

[83] Planche, C., W. Wobrock, A. I. Flossmann, F. Tridon, J. V. Baelen, Y. Pointin und M. Hagen: *The influence of aerosol particle number and hygroscopicity on the evolution of convective cloud systems and their precipitation: A numerical study based on the COPS observations on 12 August 2007.* Atmospheric Research, 98(1):40–56, 2010.

[84] Pothen, A.: *Graph Partitioning Algorithms with Applications to Scientific Computing.* In: Keyes, D. E., A. Sameh und V. Venkatakrishnan (Hrsg.): *Parallel Numerical Algorithms*, S. 323–368. Kluwer Academic Publishers, 1997.

[85] Pruppacher, H. R. und J. D. Klett: *Microphysics of Clouds and Precipitation.* Kluwer Academic Publishers, 2. Aufl., 1997.

[86] Randall, D. A., R. A. Wood, S. Bony, R. Colman, T. Fichefet, J. Fyfe, V. Kattsov, A. Pitman, J. Shukla, J. Srinivasan, R. J. Stouffer, A. Sumi und K. E. Taylor: *Climate Models and Their Evaluation.* In: Solomon, S. et al. (Hrsg.): *Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change.* Cambridge University Press, 2007.

[87] Rauber, T. und G. Rünger: *Parallel Programmierung.* Springer, 2. Aufl., 2007.

[88] Redler, R., S. Valcke und H. Ritzdorf: *OASIS4 - a coupling software for next generation earth system modelling.* Geoscientific Model Development, 3(1):87–104, 2010.

[89] Rockel, B., A. Will und A. Hense: *Special Issue: Regional climate modelling with COSMO-CLM (CCLM).* Meteorologische Zeitschrift, 17(4), 2008.

[90] Rodrigues, E. R., P. O. A. Navaux, J. Panetta, A. Fazenda, C. L. Mendes und L. V. Kale: *A Comparative Analysis of Load Balancing Algorithms Applied to a Weather Forecast Model.* In: Moreira, J. E. et al. (Hrsg.): *22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2010)*, S. 71–78, 2010.

[91] Sagan, H.: *Space-filling curves*. Springer, 1994.

[92] Schättler, U.: *Nutzung moderner Höchstleistungsrechner für die NWV*. Promet: Die neue Modellkette des DWD II, 28(1/2):33–39, 2002.

[93] Schättler, U., G. Doms und C. Schraff: *A Description of the Nonhydrostatic Regional COSMO-Model, Part VII: User's Guide*. Consortium for Small-scale Modelling, 2009. `http://www.cosmo-model.org/content/model/documentation/core/default.htm`.

[94] Schättler, U. und E. Krenzien: *The parallel 'Deutschland-Modell' – A message-passing version for distributed memory computers.* Parallel Computing, 23:2215–2226, 1997.

[95] Schloegel, K., G. Karypis und V. Kumar: *Parallel static and dynamic multi-constraint graph partitioning.* Concurrency and Computation: Practice and Experience, 14(3):219–240, 2002.

[96] Schloegel, K., G. Karypis und V. Kumar: *Graph Partitioning for High-Performance Scientific Simulations*. In: Dongarra, J. et al. (Hrsg.): *Sourcebook of Parallel Computung*, Kap. 18, S. 491–541. Morgan Kaufmann, 2003.

[97] Seifert, A. und K. D. Beheng: *A two-moment cloud microphysics parameterization for mixed-phase clouds. Part 1: Model description.* Meteorology and Atmospheric Physics, 92:45–66, 2006.

[98] Seifert, A., A. Khain, A. Pokrovsky und K. Beheng: *A comparison of spectral bin and two-moment bulk mixed-phase cloud microphysics.* Atmospheric Research, 80:46–66, 2006.

[99] Simmel, M. und S. Wurzler: *Condensation and activation in sectional cloud microphysical models.* Atmospheric Research, 80(2-3):218–236, 2006.

[100] Simon, H.: *Partitioning of unstructured problems for parallel processing.* Computing Systems in Engineering, 2(2-3):135–148, 1991.

[101] Sivagama, S. M., S. S. Vadhiyar und R. S. Nanjundiah: *Dynamic Component Extension: a Strategy for Performance Improvement in Multicomponent Applications.* International Journal of High Performance Computing Applications, 23(1):84–98, 2009.

[102] Skamarock, W. C. und J. B. Klemp: *A time-split nonhydrostatic atmospheric model for weather research and forecasting applications.* Journal of Computational Physics, 227(7):3465–3485, 2008.

[103] Sosa, C. und B. Knudson: *IBM System Blue Gene Solution: Blue Gene/P Application Development*. IBM Redbooks. IBM, 2009.

[104] St-Cyr, A., C. Jablonowski, J. M. Dennis, H. M. Tufo und S. J. Thomas: *A Comparison of Two Shallow-Water Models with Nonconforming Adaptive Grids.* Monthly Weather Review, 136(6):1898–1922, 2008.

[105] Steppeler, J., G. Doms, U. Schättler, H. Bitzer, A. Gassmann, U. Damrath und G. Gregoric: *Meso-gamma scale forecasts using the nonhydrostatic model LM.* Meteorology and Atmospheric Physics, 82:75–96, 2003.

[106] Taylor, M. A., J. Edwards und A. St-Cyr: *Petascale atmospheric models for the Community Climate System Model: new developments and evaluation of scalable dynamical cores.* Journal of Physics: Conference Series, 125(1):012023, 2008.

[107] Teresco, J. D., K. D. Devine und J. E. Flaherty: *Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations.* In: Bruaset, A. M. und A. Tveito (Hrsg.): *Numerical Solution of Partial Differential Equations on Parallel Computers*, Bd. 51 d. Reihe *Lecture Notes in Computational Science and Engineering*, S. 55–88. Springer, 2006.

[108] Teresco, J. D., J. Faik und J. E. Flaherty: *Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation*. In: Dongarra, J., K. Madsen und J. Wasniewski (Hrsg.): *Applied Parallel Computing*, Bd. 3732 d. Reihe *Lecture Notes in Computer Science*, S. 911–920. Springer, 2006.

[109] Tóth, G. et al.: *Space Weather Modeling Framework: A new tool for the space science community*. Journal of Geophysical Research, 110(A9):A12226, 2005.

[110] Tremback, C. J. und R. L. Walko: *The Regional Atmospheric Modeling System (RAMS): Development for parallel processing computer architectures*. In: *3rd RAMS Users Workshop*, 1997.

[111] Van Straalen, B., J. Shalf, T. Ligocki, N. Keen und W.-S. Yang: *Scalability challenges for massively parallel AMR applications*. In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS 2009)*. IEEE Computer Society, 2009.

[112] Visconti, G.: *Fundamentals of Physics and Chemistry of the Atmosphere*. Springer, 2001.

[113] Walko, R. L. und C. J. Tremback: *RAMS Regional Atmospheric Modeling System, Version 6.0 – Model Input Namelist Parameters*. ATMET, 30. Oktober 2006. `http://www.atmet.com/html/docs/rams/ug60-model-namelist-1.4.pdf`.

[114] Walshaw, C. und M. Cross: *Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm*. SIAM Journal on Scientific Computing, 22:63–80, 2000.

[115] Walshaw, C. und M. Cross: *JOSTLE – Multilevel Graph Partitioning Software: An Overview*. In: Magoules, F. (Hrsg.): *Mesh Partitioning Techniques and Domain Decomposition Techniques*, Kap. 2, S. 27–58. Saxe-Coburg Publications, 2007.

[116] Warner, J. C., N. Perlin und E. D. Skyllingstad: *Using the Model Coupling Toolkit to couple earth system models*. Environmental Modelling & Software, 23(10-11):1240–1249, 2008.

[117] Washington, W. M., L. Buja und A. Craig: *The computational future for climate and Earth system models: on the path to petaflop and beyond*. Philosophical Transactions A, 367(1890):833–846, 2009.

[118] Watts, J. und S. Taylor: *A Practical Approach to Dynamic Load Balancing*. IEEE Transactions on Parallel and Distributed Systems, 9:235–248, 1998.

[119] Wolf, K. (Hrsg.): *10th MpCCI User Forum Proceedings*. Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Sankt Augustin, 2009.

[120] Wolfheimer, F., E. Gjonaj und T. Weiland: *A parallel 3D particle-in-cell code with dynamic load balancing*. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 558(1):202–204, 2006.

[121] Wolke, R., O. Knoth, O. Hellmuth, W. Schröder und E. Renner: *The Parallel Model System LM-MUSCAT for Chemistry-Transport Simulations: Coupling Scheme, Parallelization and Applications*. In: Joubert, G. et al. (Hrsg.): *Parallel Computing - Software Technology, Algorithms, Architectures and Applications*, Bd. 13 d. Reihe *Advances in Parallel Computing*, S. 363–369. Elsevier, 2004.

[122] Xue, M., K. K. Droegemeier und D. Weber: *Numerical Prediction of High-Impact Local Weather: A Driver for Petascale Computing*. In: Bader, D. A. (Hrsg.): *Petascale Computing: Algorithms and Applications*, Kap. 6, S. 103–124. Chapman & Hall/CRC, 2008.

[123] Zhang, L. und M. Parashar: *Seine: a dynamic geometry-based shared-space interaction framework for parallel scientific applications*. Concurrency and Computation: Practice and Experience, 18(15):1951–1973, 2006.

[124] Zheng, G., A. Bhatele, E. Meneses und L. V. Kale: *Periodic hierarchical load balancing for large supercomputers*. International Journal of High Performance Computing Applications, 2011.

A Algorithmen zur 1D-Partitionierung

A.1 Grundlegende Algorithmen zur 1D-Partitionierung

```
PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
  if W_guess > sum then                                  /* Schätzung war zu hoch */
    i := guess − 1
    while i ≥ start and W_i > sum do i := i − 1
    end := max(start, i)
  else                                                   /* Schätzung war zu gering oder genau richtig */
    i := guess + 1
    while i ≤ B and W_i ≤ sum do i := i + 1
    end := i − 1
  guess = min(end + end − start, B)                      /* schätze Ende der folgenden Partition */
  return end
```

Algorithmus A.1: *PARTITION_SEARCH* sucht in der exklusiven Präfixsumme W des Blockgewichtsvektors der Länge B nach der höchsten Position *end* mit $end \geq start$ und $W_{end} < sum$. Die lineare Suche wird bei *guess* begonnen; einer neuer Schätzwert für die Suche nach der folgenden Partition (mit gleicher Last) wird in *guess* zurückgegeben und beim nächsten Aufruf von *PARTITION_SEARCH* verwendet. Die Komplexität ist $O(1 + \frac{B}{P}(\alpha - 1))$ wobei α davon abhängt, wie stark die Blockgewichte aufeinanderfolgender Partitionen voneinander abweichen. Im ungünstigsten Fall ist $\alpha = w_{max}/w_{min}$; typischerweise gilt jedoch $1 \leq \alpha < w_{max}/w_{min}$. Im Vergleich zu dem von Pınar und Aykanat [82] genutzten Algorithmus mit durchschnittlich $O(log(B/P))$ wurden für die Messungen in Abschnitt 5.2.1 geringere Laufzeiten für *PARTITION_SEARCH* gemessen, was auf $\alpha \ll w_{max}/w_{min}$ hindeutet.

```
H1 (in: W, B, P, out: s)
  start := 0
  β* := W_B/P
  sum := β*
  guess := B/P
  for p := 1 to P − 1 do
    end := PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
    s_{p+1} := end + 1
    sum := sum + β*
    start := end
  s_1 := 1
  s_{P+1} := B + 1
```

Algorithmus A.2: Der 1D-Partitionierungsalgorithmus *H1* nach [76] erzeugt einen Partitionsvektor s für P Partitionen. Die Blockgewichte sind als exklusive Präfixsumme W gegeben.

```
H2 (in: W, B, P, out: s)
  start := 0
  β* := W_B/P
  sum := β*
  guess := B/P
  for p := 1 to P − 1 do
      end := PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
      if end < B and W_{end+1} − sum < sum − W_end then      /* Einzige Änderung zu H1 */
          end := end + 1
      s_{p+1} := end + 1
      sum := sum + β*
      start := end
  s_1 := 1
  s_{P+1} := B + 1
```

Algorithmus A.3: Der 1D-Partitionierungsalgorithmus *H2* nach [76] erzeugt einen Partitionsvektor s für P Partitionen. Die Blockgewichte sind als exklusive Präfixsumme W gegeben.

```
PARTITION_VECTOR (in: W, B, P, β, out: s)
  start := 0
  sum := β
  guess := B/P
  for p := 1 to P − 1 do
      end := PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
      s_{p+1} := end + 1
      sum := W_end + β
      start := end
  s_1 := 1
  s_{P+1} := B + 1
```

Algorithmus A.4: *PARTITION_VECTOR* erzeugt einen Partitionsvektor s für P Partitionen mit der maximalen Last β (*bottleneck value*). Die Blockgewichte sind als exklusive Präfixsumme W gegeben.

```
PROBE (in: W, B, P, β)
  start := 0
  sum := β
  guess := B/P
  for p := 1 to P − 1 do
      end := PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
      if end = B then exit
      sum := W_end + β
      start := end
  if W_B ≤ sum then return true
  else return false
```

Algorithmus A.5: *PROBE* prüft, ob der Blockvektor der Länge B in P Partitionen mit der maximalen Last β (*bottleneck value*) zerlegt werden kann. Die Blockgewichte sind als exklusive Präfixsumme W gegeben.

```
PROBE+ (in: W, B, P, β, out: β⁻_res, β⁺_res)
  start := 0
  sum := β
  guess := B/P
  β⁺_res := W_B
  β⁻_res := 0
  for p := 1 to P − 1 do
      end := PARTITION_SEARCH (in: W, start, B, sum, inout: guess)
      β⁻_res := max(β⁻_res, W_end − W_start)
      if end = B then exit
      β⁺_res := min(β⁺_res, W_end+1 − W_start)
      sum := W_end + β
      start := end
  β⁻_res := max(β⁻_res, W_B − W_start)
  β⁺_res := min(β⁺_res, W_B − W_start)
  if W_B ≤ sum then return true
  else return false
```

Algorithmus A.6: *PROBE+* erweitert *PROBE* um die Bestimmung der tatsächlichen Maximallast $\beta^-_{res} \leq \beta$ (bei Erfolg) sowie die nächsthöhere tatsächlich mögliche Maximallast $\beta^+_{res} > \beta$.

A.2 1D-Partitionierungsalgorithmen mit vorgegebener Genauigkeit

```
r-BISECT (in: W, B, P, w_max, r, out: s)
  ε = W_B(1 − r)/(P r)
  β* = W_B/P
  β_l := max(β*, w_max)
  β_u := β* + w_max
  while β_u − β_l > ε do
      β := (β_l + β_u)/2
      if PROBE+ (in: W, B, P, β, out: β⁻_res, β⁺_res) then
          β_u = β⁻_res
      else
          β_l = β⁺_res
  β_res = β_u
  PARTITION_VECTOR (in: W, B, P, β_res, out: s)
```

Algorithmus A.7: Der 1D-Partitionierungsalgorithmus *r-BISECT* erzeugt einen Partitionsvektor s für P Partitionen. Die Blockgewichte sind als exklusive Präfixsumme W gegeben. Für den *bottleneck value* der erzeugten Partitionierung β_{res} garantiert dieses Verfahren in Bezug zum bestmöglichen *bottleneck value* β_{opt} das Verhältnis $\beta_{res} \leq \beta_{opt}/r$ mit $0 < r \leq 1$. Für $r = 1$ ist das Verfahren exakt.

```
r-PAR-BISECT (in: W, B, P, w_max, r, p, out: s)
  ε = W_B(1 − r)/(P r)
  β* = W_B/P
  b_l := max(β*, w_max)
  b_u := β* + w_max
  β_l := b_l + (p − 1) (b_u − b_l)/P        /* Teilintervall des Prozesses p: I_p = [β_l, β_u] */
  β_u := b_l + p (b_u − b_l)/P
  if PROBE+ (in: W, B, P, β_l, out: β⁻_res, β⁺_res) then     /* Prüfe untere Grenze β_l */
      β_res = β⁻_res
  else
      β_l = β⁺_res
      β_res = b_u
      while β_u − β_l > ε do                                  /* binäre Suche in I_p */
          β := (β_l + β_u)/2
          if PROBE+ (in: W, B, P, β, out: β⁻_res, β⁺_res) then
              β_u = β⁻_res
              β_res = β⁻_res
          else
              β_l = β⁺_res
  b_res := MPI_Allreduce (Minimum von β_res)
  PARTITION_VECTOR (in: W, B, P, b_res, out: s)
```

Algorithmus A.8: Der parallele 1D-Partitionierungsalgorithmus *r-PAR-BISECT* basiert auf *r-BISECT*. Das Suchintervall für β_{res} wird unter den P Prozessen aufgeteilt. Wie bei *r-BISECT* gilt für die erzeugte Partitionierung $\beta_{res} \leq \beta_{opt}/r$. Der Prozessidentifikator des rufenden Prozesses ist in p mit $1 \leq p \leq P$ gegeben.

A.3 Verteilter 1D-Partitionierungsalgorithmus

r-DISTR-BISECT (**in:** s, w, B, P, r, p, **out:** s')
  Aufteilen der P Prozesse in Q aufeinanderfolgende Gruppen zu je P/Q Prozessen
  $p^q :=$ Prozessidentifikator des Prozesses p innerhalb seiner Gruppe, $1 \leq p^q \leq P/Q$
  PREFIXSUM (**in:** s, w, B, P, p, **out:** W, *sumweight*)
  COARSE_PARTITION (**in:** s, W, Q, *sumweight*, p, **out:** $\tilde{u}$, $qfirst$)
  COARSE_PARTITION_COMM (**in:** B, P, Q, s, $\tilde{u}$, p, **out:** u, $pfirst$, $plast$)
  EXCH_WEIGHTS (**in:** s, B, P, Q, W, $qfirst$, $\tilde{u}$, u, $pfirst$, $plast$, p, **out:** W^q, w_{max})
  r-PAR-BISECT (**in:** W^q, $|W^q| - 1$, P/Q, w_{max}, r, p^q, **out:** s^q)
  `MPI_Allgather` (**in:** erster Block dieses Prozesses aus s^q, **out:** $s'_i, i = 1, 2, \ldots, P$)
  $s'_{P+1} = B + 1$

Algorithmus A.9: Der parallele 1D-Partitionierungsalgorithmus *r-DISTR-BISECT* kombiniert *H1* und *r-PAR-BISECT* zu einem hierarchischen Verfahren. Im Unterschied zu den bisher betrachteten Verfahren wird hier nicht der vollständige Vektor der Präfixsumme aller Blockgewichte benötigt. Als Eingabe werden stattdessen nur die lokalen Blockgewichte w des Prozesses p übergeben. Die aktuelle Partitionierung wird mit dem Partitionsvektor s übergeben. Der neue Partitionsvektor wird in s' zurückgegeben. Es gelten folgende Annahmen: P/Q sei ganzzahlig, $Q \geq 2$ und $P/Q \geq 2$ (und somit $P \geq 4$).

PREFIXSUM (**in:** s, w, B, P, p, **out:** W, *sumweight*)
  $load := \sum_{i=s_p}^{s_{p+1}-1} w_i$ /* Summe der lokalen Blockgewichte */
  $W_{s_p-1} :=$ `MPI_Exscan` (Summe von *load*) /* parallele exklusive Präfixsumme */
  if $p > 1$ **then** `MPI_Isend` (W_{s_p-1} an Prozess $p-1$)
  /* berechne lokalen Abschnitt der Präfixsumme aller Blockgewichte */
  for $j := s_p$ **to** $s_{p+1} - 1$ **do** $W_j := W_{j-1} + w_j$
  if $p < P$ **then** `MPI_Recv` ($W_{s_{p+1}-1}$ von Prozess $p+1$)
  if $p > 1$ **then** `MPI_Wait` ()
  if $p = P$ **then** $sumweight := W_B$; /* aufsummierte Last aller Blöcke */
  `MPI_Bcast` (*sumweight* von Prozess P an alle Prozesse)

Algorithmus A.10: *PREFIXSUM* berechnet die Präfixsumme der Blockgewichte W auf parallele Weise und ermittelt die Gesamtlast aller Blöcke *sumweight*. Der Vektor W bleibt dabei entsprechend der aktuellen Partitionierung über alle Prozesse verteilt, wobei er sich zwischen benachbarten Prozessen um einen Eintrag überschneidet: W_{s_p-1} enthält im Prozess p die Gesamtlast vor den eigenen Blöcken im Blockvektor und im Prozess $p-1$ die Gesamtlast inklusive der eigenen Blöcke. Um trotz verschiedenen Weges der Berechnung dieser Werte deren Identität zu garantieren, werden sie mit Punkt-zu-Punkt Nachrichten zwischen benachbarten Prozessen ausgetauscht. Andernfalls können durch Gleitkommaarithmetik verursachte Abweichungen dazu führen, dass grobe Partitionsgrenzen doppelt oder gar nicht gefunden werden. Dadurch würde im weiteren Verlauf des Verfahrens eine Blockierung entstehen.

```
COARSE_PARTITION (in: s, W, Q, sumweight, p, out: ũ, qfirst)
  β* := sumweight/Q
  start := s_p − 1
  guess := start
  exactmatch := false
  qfirst := −1
  for q := max(1, ⌈W_{s_p−1}/β*⌉ − 1) to min(Q − 1, ⌊W_{s_{p+1}−1}/β*⌋ + 1) do
      sum := q β*
      end := PARTITION_SEARCH (W, start, s_{p+1} − 1, sum, guess)
      if sum ≥ W_end then
          if end = s_p − 1 and sum = W_end then
              qfirst := q + 2          /* Prozess p − 1 findet diese Partitionsgrenze ebenfalls */
          else if end < s_{p+1} − 1 or sum = W_end then
              qfirst := min(q + 1, qfirst)
              ũ_{q+1} := end + 1
      start := end
  if qfirst < 0 then qfirst := max(1, ⌈W_{s_p−1}/β*⌉) + 1
```

Algorithmus A.11: *COARSE_PARTITION* sucht in dem lokalen Abschnitt der Präfixsumme der Blockgewichte W nach Partitionsgrenzen $\tilde{u}$ der grobgranularen Partitionierung in Q Partitionen. Die Suche erfolgt gemäß der Heuristik *H1*. Partitionsgrenzen mit $sum = W_{s_p-1}$ werden verworfen (exakte Treffer vor dem ersten eigenen Block), da der Prozess $p-1$ diese Grenze an seinem letzten Block registriert. In $qfirst$ wird diejenige grobgranulare Partition $2 \leq qfirst \leq Q$ zurückgegeben, welche an der ersten lokal gefundenen Grenze beginnt. Falls keine Grenze im lokalen W gefunden wird, ist $\tilde{u}$ leer und $qfirst$ die nächste nach den lokalen Blöcken beginnende grobgranulare Partition.

```
COARSE_PARTITION_COMM (in: B, P, Q, s, ũ, p, out: u, pfirst, plast)
  foreach ũ_q in ũ do
      MPI_Isend (ũ_q als untere Grenze an den ersten Prozess der q-ten Gruppe)
      MPI_Isend (ũ_q als obere Grenze an den ersten Prozess der (q − 1)-ten Gruppe)
  if p ist erster Prozess einer Gruppe q then
      if q > 1 then MPI_Recv (untere Grenze u_q, merke Sender in pfirst)
      else u_1 := 1; pfirst := 1
      while pfirst < P and s_{pfirst+1} ≤ u_q do pfirst := pfirst + 1
      if q < Q then MPI_Recv (obere Grenze u_{q+1}, merke Sender in plast)
      else u_{Q+1} := B + 1; plast := P
      while plast > 1 and s_plast > u_q − 1 do plast := plast − 1
  MPI_Bcast ({u_q, u_{q+1}, pfirst, plast} vom Gruppenersten an alle Prozesse der Gruppe)
  MPI_Waitall ()
```

Algorithmus A.12: *COARSE_PARTITION_COMM* verteilt die gefundenen groben Partitionsgrenzen $\tilde{u}$ an die beiden jeweils anliegenden Partitionen. Der gezeigte Algorithmus kann leicht dahingehend erweitert werden, dass keine Punkt-zu-Punkt-Nachricht übertragen wird, falls der erste Prozess einer grobgranularen Partition selbst eine der Grenzen dieser Partition in findet. In u_q und u_{q+1} werden der erste Blockindex der Gruppe q des rufenden Prozesses und der darauf folgenden Gruppe $q+1$ zurückgegeben. $pfirst$ und $plast$ sind der erste und letzte Prozess, die der Gruppe q zugeteilte Blöcke besitzen.

EXCH_WEIGHTS (**in:** s, B, P, Q, W, $qfirst$, $\tilde{u}$, u, $pfirst$, $plast$, p, **out:** W^q, w_{max})
  /* Senden der W_j an ersten oder letzten Prozess der neuen Gruppe */
  $start := s_p$
  $qlast := qfirst + |\tilde{u}|$
  for $q := qfirst$ **to** $qlast$ **do**
    if $q < qlast$ **then** $end := \tilde{u}_q - 1$
    else $end := s_{p+1} - 1$
    if $start \leq end$ **and** q *ist nicht Gruppe von Prozess* p **then**
      if *Block bei start ist erster einer groben Partition* **then** $start := start - 1$
      $pg :=$ derjenige Prozess aus Gruppe $q - 1$ mit minimalem $|pg - p|$
      `MPI_Isend` ($\{W_j,\ j = start, start + 1, \ldots, end\}$ an pg)
    $start := end + 1$
  /* Erste und letzte Prozesse einer Gruppe empfangen die W_j */
  if p *ist erster oder letzter Prozess einer Gruppe* q **then**
    if p *ist erster Prozess der Gruppe* q **then**
      $pstart := pfirst$
      $pend := min(plast, p - 1)$
      $start := u_q$
    if p *ist letzter Prozess der Gruppe* q **then**
      $pstart := max(pfirst, p + 1)$
      $pend := plast$
      $start := max(u_q - 1, s_{p+1})$
    for $pr{:=}pstart$ **to** $pend$ **do**
      $end := min(s_{pr+1}, u_{q+1}) - 1$
      if $start \leq end$ **then**
        if *Block bei start ist der erste der groben Partition* **then** $start := start - 1$
        `MPI_Irecv` ($\{W_j,\ j = start, start + 1, \ldots, end\}$ von pr)
      $start := end + 1$
  `MPI_Waitall` ()
  /* Verteilen der W_j innerhalb der Gruppe */
  Erzeuge Parameter für `MPI_Allgatherv`, um alle W_j der eigenen Gruppe auszutauschen
  `MPI_Allgatherv` ($\{W_j, j = u_q - 1, u_q, \ldots, u_{q+1} - 1\}$ innerhalb der eigenen Gruppe)
  /* Berechen der Gruppen-lokale Präfixsumme und des maximalen Blockgewichts */
  $w_{max} := W_{u_q} - W_{u_q - 1}$
  $W_0^q := 0$
  for $i = 1$ **to** $u_{q+1} - u_q$ **do**
    $W_i^q := W_{i+u_q-1} - W_{u_q-1}$
    $w_{max} := max(w_{max}, W_i^q - W_{i-1}^q)$

Algorithmus A.13: *EXCH_WEIGHTS* verteilt die lokalen aufsummierten Gewichtsvektoren innerhalb der Gruppen. Dabei werden nur die Gewichte derjenigen Blöcke mit Punkt-zu-Punkt Nachrichten übertragen, welche die grobgranulare Partition wechseln. Alle anderen werden innerhalb der Gruppe mit einer kollektiven Operation verteilt. In W^q wird die Gruppen-lokale Präfixsumme und in w_{max} das maximale Blockgewicht der eigenen Gruppe zurückgegeben.