Forschungszentrum Jülich
in der Helmholtz-Gemeinschaft

Zentralinstitut für Angewandte Mathematik

Strategien zur Kopplung und Datenreduktion bei der Online-Visualisierung von parallelen Simulationsrechnungen mit verteilter Datenhaltung

Wolfgang Frings

Jül-4021

Strategien zur Kopplung und Datenreduktion bei der Online-Visualisierung von parallelen Simulationsrechnungen mit verteilter Datenhaltung

Wolfgang Frings

Berichte des Forschungszentrums Jülich ; 4021
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-4021

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461/61-5220 · Telefax: 02461/61-6103 · e-mail: zb-publikation@fz-juelich.de

Zusammenfassung

Bei der Online-Visualisierung werden ein Simulationprogramm und eine Visualisierungsanwendung direkt miteinander verbunden, so daß schon während des Ablaufs der Simulationsrechnung Zwischenergebnisse graphisch betrachtet werden können und steuernd in die Simulationrechnung eingegriffen werden kann (Computational Steering). Insbesondere bei Simulationsrechnungen auf großen Parallelrechnern mit verteilter Datenhaltung entstehen dabei zwei Probleme: Zum einen die große Datenmenge, die von der Simulation zur Visualisierung übertragen werden muß, und zum anderen die verteilte Datenhaltung innerhalb des Simulationsprogramms.

In dieser Arbeit wird zur Lösung des ersten Problems eine Reduzierung der Datengröße mit Hilfe eines Komprimierungsverfahren auf Basis der Wavelet-Transformation eingesetzt, mit dem die Daten in verschiedene Auflösungsstufen zerlegt und progressiv zur Visualisierung übertragen werden können. Für die Lösung des zweiten Problems, der verteilten Datenhaltung, werden in der Arbeit unterschiedliche Kopplungsstrategien vorgestellt, welche die verteilten Datenhaltung des Simulationsprogramms berücksichtigen und dabei die Wavelet-zerlegten Daten der einzelnen Prozessoren übertragen.

Zur Bewertung der verwendeten Komprimierungsverfahren und Kopplungsstrategien wurden Modelle für deren Laufzeitverhalten und deren Einfluß auf ein paralleles Simulationsprogramm entwickelt und an Hand von Messungen verifiziert. Dazu wurde in dieser Arbeit die Bibliothek *LVISIT* und der Code-Generator *visitcg* entwickelt, welche die im Forschungszentrum entwickelte Kommunikationsbibliothek *VISIT* benutzen und dem Benutzer eine einfache Anwendung der verschiedenen Kopplungsstrategien und der Komprimierung auf Basis der Wavelet-Transformation ermöglichen.

Abstract

Visualizing data just being calculated by a simulation program running on a remote computer is called online visualization. This method allows an immediate visualization and a direct control of simulation parameters (computational steering). Applying online visualization to large-scale applications on parallel supercomputers with distributed memory lead to two major problems: on one hand the huge amount of data which have to be transfered between simulation and visualization and on the other hand the distributed data management within the simulation program.

The first problem is tackled by a compression procedure on the basis of a wavelet transformation which is implemented in this thesis. This allows to decompose the data in different resolution steps and to transfer them progressively to the visualization. To overcome the second problem coupling strategies for the distributed data management are introduced, which take into account the parallel structure of the simulation program and ensure a reliable transfer of the decomposed data located on the single processors.

The benchmarking of the applied compression techniques and coupling strategies with respect to the run time behavior and to the effects on a parallel simulation program was done by simplified models and validated by real time measurements. For that purpose a library *LVISIT* and a code generator *visitcg* were developed, which use the communication library *VISIT* and provide the coupling techniques introduced here together with a compression based on wavelet transformation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die *Online-Visualisierung* und die interaktive Steuerung von Simulationsprogrammen (*Computational Steering*) sind im wissenschaftlichen Rechnen etablierte Werkzeuge, welche die effiziente Nutzung von Rechner-Ressourcen unterstützen. Der Benutzer kann mit diesen Werkzeugen frühzeitig auf falsche oder unerwünschte Ergebnisse durch Verändern von Parametern bzw. auch durch Abbruch reagieren.

Mit der stetig wachsenden Leistungsfähigkeit und Kapazität von Parallelrechnern steigt auch die Größe der von Simulationsprogrammen produzierten Datensätze. Bei großen Parallelrechnern liegt die Gesamtgröße des auf die einzelnen Prozessoren verteilten Hauptspeichers im TByte-Bereich. Typische Simulationsergebnisse erreichen dabei oft die Größe von mehreren GByte.

Die Bandbreite der Netzverbindungen und die Leistungsfähigkeit der für die Online-Visualisierung genutzten Workstations steigen aber nicht in diesem Maße. Insbesondere durch den begrenzten Hauptspeicher einer Visualisierungsworkstation ist die direkte Verarbeitung und Anzeige von sehr großen Datensätzen schwierig.

Die gegenüber Desktop-Maschinen um Faktoren höhere Leistungsfähigkeit von Parallelrechnern erzwingt deshalb neben der Komprimierung auch den Einsatz neuer Methoden zur Datenreduzierung. Es genügt nicht nur, die Daten für den Transport zur Visualisierungsworkstation zu komprimieren, sondern auch der Umfang der Daten muß für die Visualisierung reduziert werden.

Bei den derzeit verfügbaren Tools zur Online-Visualisierung und Steuerung von Simulationsprogrammen werden diese Probleme nicht gezielt angegangen. Methoden zur Datenreduzierung sind zwar aus dem Bereich des Post-Processing bekannt, müssen aber bei der Online-Visualisierung unter neuen Gesichtspunkten betrachtet werden. So muß z.B. der Aufwand und der Ressourcen-Bedarf für die Reduzierung der Daten begrenzt sein, da diese begleitend zur Simulationsrechnung ablaufen soll. Wichtig ist, daß die aktuellen Daten auf dem Bildschirm dargestellt sind, bevor das Simulationsprogramm schon wieder neue Daten berechnet hat. Anderenfalls ist ein Reagieren auf die aktuelle Situation schwierig. Zudem muß sichergestellt werden, daß die für die Online-Visualisierung nötige Datenreduzierung die eigentliche Simulationsrechnung nur in begrenztem Maße belastet. Ist dies nicht der Fall, führt der Einsatz des *Computational Steering* nicht zu einem Gewinn. Die durch das direkte Reagieren mögliche Schonung von Ressourcen wird dann durch den erhöhten Verbrauch von Ressourcen bei der Datenreduzierung aufgehoben.

In dieser Arbeit wird für die Datenreduzierung bei der Online-Visualisierung die in vielen anderen Bereichen genutzte *Wavelet-Transformation* eingesetzt. Für diese lineare Transformation gibt es schnelle Algorithmen, die in ihrem Aufwand begrenzt sind und sich somit für den Einsatz im *Computational Steering* eignen. Im Gegensatz zur *Fourier*-Transformation, welche die Daten vom

Orts- in den Frequenzraum transformiert, überführt die *Wavelet-Transformation* die Daten in eine Darstellung, welche die Vorteile beider Räume ausnutzt. Dabei werden die Daten in verschiedene Frequenzbänder zerlegt, so daß einzelne Bänder in Abhängigkeit von ihrem Informationsgehalt ausgewählt und *progressiv* zur Visualisierungsworkstation übertragen werden können.

Diese Datenreduzierung mit Hilfe der *Wavelet-Transformation* wird in der vorliegenden Arbeit bei der Ankopplung von parallelen Simulationsprogrammen an eine Visualisierung genutzt. Die verteilte Datenhaltung innerhalb eines parallelen Simulationsprogramms wirft dabei neue Fragen auf. So muß z.B. sichergestellt sein, daß auch die Datenreduzierung verteilt ablaufen kann und die Daten nicht komplett in einem Rechenknoten vorliegen müssen. Moderne Parallelrechner bestehen immer häufiger aus vielen Standard-Rechnern, die über ein schnelles Kommunikationsnetz verbunden sind. Jeder Rechner besitzt dabei einen eigenen lokalen Hauptspeicher. Nur durch den Zusammenschluß vieler solcher Rechner erhält man den für den Einsatz von Simulationsprogrammen notwendigen großen Hauptspeicher. Da ein direkter Zugriff nur auf den lokalen Hauptspeicher möglich ist, muß auf entfernte Speicher über das Kommunikationsnetz und mit Hilfe von speziellen Kommunikationsbibliotheken (PVM, MPI) zugegriffen werden. Die Untersuchung verschiedener Strategien zur Einbettung der für die Online-Visualisierung nötigen Kommunikation in die interne Kommunikation des Simulationsprogramms ist Gegenstand des zweiten Teils der Arbeit.

Ob die in die verschiedenen Kopplungsstrategien eingebundene Komprimierung eine Zeitersparnis liefert, hängt von vielen Parametern ab. Um deren Einfluß voraussagen zu können, werden in dieser Arbeit für die verschiedenen Verarbeitungszeiten und auch für den Einfluß der Ankopplung an ein paralleles Programm (Speedup) Modelle entwickelt und an Hand der implementierten Verfahren durch Zeitmessungen überprüft.

Für die Implementierung der Steering-Komponenten werden das in der Wissenschaft weit verbreitete Visualisierungswerkzeug *AVS/Express* [1] und die im Forschungszentrum Jülich entwickelte Kopplungsbibliothek *VISIT* [2] benutzt.

Bei der Kopplung eines Simulationsprogramms mit einer Visualisierung entstehen viele Datenströme, welche die darzustellenden Daten, aber auch Kontrollinformation zur Steuerung der Simulation enthalten. Die Anzahl und die Art der Datenströme hängt stark von der Anwendung ab. Auch die Verteilung der Daten innerhalb der Simulationsprogramme auf die lokalen Speicher der Prozessorelemente ist abhängig vom Design der Programme. Im Rahmen dieser Arbeit ist zur einfachen Spezifikation der Datenströme, der Komprimierungs- und Datenreduktionsmethode und der Gebietsaufteilung der Daten eine Beschreibungssprache und dazu ein *Code-Generator* entwickelt worden, der ausgehend von der Spezifikation der Datenströme sowohl eine auf das Simulationsprogramm angepaßte Schnittstelle als auch eine Beispiel-Applikation (AVS/Express) auf der Visualisierungsseite generiert.

Als Anwendungsbeispiele für die Komprimierung und Ankopplung werden zwei Simulationsprogramme untersucht, die unterschiedliche Arten von Simulationsdaten und Verteilungsstrategien der Daten auf den verteilten Speicher eines Parallelrechners verwenden. Dabei handelt es sich um die beiden Programme Trace [3] und Partrace [4], die eine Grundwasserströmung mit Hilfe eines gebietszerlegten FE-Gitters simulieren und die Schadstoffausbreitung im Grundwasser mit einer *Particle Tracking*-Methode berechnen.

Kapitel 2

Computational Steering

In einer Vielzahl von Forschungsbereichen werden Simulationsrechnungen durchgeführt, deren Ergebnisdaten eine solche Größe erreichen, daß selbst für deren Analyse (*Post-Processing*) ein *Supercomputer* erforderlich ist. Dabei ist nicht die Rechenleistung der lokalen Workstation der begrenzende Faktor, vielmehr ist es die Größe der Daten, die eine Übertragung und lokale Speicherung schwierig machen. Lieferanten von großen Datensätze sind z.B. Simulationen im Bereich der *Computational Fluid Dynamics (CFD)*, der Schadstoffausbreitung, der Wetter- und Klimaforschung, Molekulardynamik und der Astrophysik.

In Teilbereichen der Simulationsrechnungen stellen die *Online-Visualisierung* und das *Computational Steering* eine Alternative zu dem heute üblichen Verfahren des *Post-Processing* dar. Der Wissenschaftler betrachtet dabei schon während der Simulationsrechnung die Zwischenergebnisse (Online-Visualisierung) und kann dabei ggf. mit dem Simulationsprogramm direkt interagieren (Computational Steering). Da er im Gegensatz zum *Post-Processing* schon während der Rechnung die Ergebnisse von Zwischenschritten betrachten und analysieren kann, ist er frühzeitig in der Lage, durch Anpassung von Programmparametern auf fehlgeleitete Simulationsrechnungen zu reagieren. So können unnötige und kostspielige Simulationsrechnungen vermieden werden. Zudem ist es bei diesem Modell nicht mehr nötig, präventiv eine Vielzahl von Zwischenergebnissen für das *Post-Processing* auf Platte abzuspeichern. Vielmehr kann das Abspeichern von interessanten Datensätzen durch den Anwender direkt gesteuert und so die Empfehlung von M. Cox zur Verwaltung von großen Datenmengen [6] umgesetzt werden:

> „Das Problem mit großen Datenmengen kann dadurch umgangen werden, indem man sie nicht erzeugt."

2.1 Art und Dimensionierung der Daten

Große Datenmengen können z.B. dann entstehen, wenn bei numerischen Simulationen von physikalischen Phänomenen das Berechnungsgebiet diskretisiert wird. Grundlage einer numerischen Simulation ist in der Regel ein mathematisches Modell, welches das Phänomen mehr oder weniger exakt charakterisiert. Häufig wird das mathematische Modell durch ein System von linearen oder nicht-linearen partiellen Differentialgleichungen dargestellt, die nur in einfachen Fällen analytisch lösbar sind. Hilfe bietet dann eine numerische Lösung auf Basis einer Diskretisierung, die häufig große Datenmengen entstehen läßt.

Bei der Diskretisierung wird das Simulationsgebiet durch ein Gitter ersetzt. Bei der Simulation zeitabhängiger Phänomene entsteht dadurch ein System von gewöhnlichen Differentialgleichun-

gen, das an jedem Knoten des Gitters zu lösen ist. Für die Diskretisierung des Berechnungsgebiets können die *Finite Difference Methode* (FD), die *Finite Element Methode* (FEM) oder die *Finite Volume Methode* (FV) benutzt werden[7]. Zum numerischen Modell gehören Randbedingungen, die z.B. die physikalischen oder numerischen Bedingungen an den Randknotenpunkten des Gitters festlegen. Bei Strömungssimulationen sind dies z.B. die Dirichlet'sche (vorgegebener Wert) oder die Neumann'sche Randbedingung (vorgegebener Gradient).

Je nach benötigter Genauigkeit der Lösung muß der Gitterabstand entsprechend gering gewählt werden. Dies kann zu einer hohen Anzahl von Gitterpunkten und damit zu großen Ergebnisdatenmengen führen.

In einer weiteren Gruppe von Simulationrechnungen fallen größere Datenmengen an, die nicht an ein Gitter gebunden sind. Zu diesen gehören z.B. solche, die *Particle-Tracking* oder *Monte Carlo-Methoden* verwenden. Benutzt werden diese Methoden häufig bei der Molekulardynamik [5], der Astrophysik oder auch im Bereich der Schadstoffausbreitung, zu dem auch das in dieser Arbeit verwendete Programm *Partrace* zu zählen ist. Bei den Daten handelt es sich häufig um Objekte (Partikel, Atome, Sterne, usw.), die mit Attributen (Ort, Geschwindigkeit, Art, ...) versehen sind. Häufig verwenden die Simulationsprogramme intern doch ein Gitter, das den Simulationsbereich in Teilbereiche zerlegt. Für diese Teilbereiche werden dann lokale Maße wie z.B. die Konzentration oder Gesamtenergie ermittelt und für die weitere Rechnung genutzt. Insbesondere bei einer sehr hohen Anzahl von Objekten wird diese Strategie häufig angewendet.

2.2 Die verschiedenen Gittertypen

Je nach Anwendungsgebiet ist die Art des zu verwendenden Gitters unterschiedlich. Gitter kann man in *strukturierte* und *unstrukturierte* Gitter unterteilen. In einem strukturierten Gitter gibt es ein festes Verbindungsschema der Knoten, bei einem unstrukturierten Gitter sind die Knoten beliebig miteinander verbunden. Die Anordnung der Gitterknoten in einem strukturierten Gitter ist zwar festgelegt, die Position der Gitterknoten im Berechnungsgebiet ist aber noch frei. Spezialfälle strukturierter Gitter sind z.B. *uniforme* und *geradlinige* (*rectlinear*) Gitter. Bei den erstgenannten sind die Abstände zwischen den Knoten in Richtung einer Koordinatenachse immer gleich. Bei geradlinigen Gittern darf auch dieser Abstand von Knoten zu Knoten unterschiedlich sein[1]. Abbildung 2.1 auf der nächsten Seite veranschaulicht die verschiedenen Gittertypen.

Die Anordnung und auch die Einschränkung der Position der Gitterknoten im Raum erlauben Vereinfachungen bei der Speicherung der im Gitter angeordneten Daten. Dies gilt sowohl für die Speicherung der Verbindungsinformationen der einzelnen Knoten als auch für die Speicherung der Position der Knoten im Raum.

Bei unstrukturierten Gittern müssen zusätzlich zu den Daten Informationen darüber abgespeichert werden, welche Knoten miteinander verbunden sind. Bei strukturierten Gittern ergibt sich diese Information direkt aus der Anordnung der Knoten. Das Verbindungsschema ist also implizit gegeben.

Ähnliches gilt auch für die Position der Knoten im Raum: In einem uniformen Gitter müssen für die Beschreibung des Gitters nur Dimension und Position zweier Eckpunkte gespeichert werden. Alle anderen Informationen lassen sich daraus berechnen. In einem geradlinigen Gitter kommen für jede Achse noch Abstandsinformationen hinzu.

[1] Die oben eingeführten Bezeichnungen für die Gitter sind aus den Beschreibungen des in dieser Arbeit eingesetzten Visualisierungssystem AVS/Express abgeleitet [1].

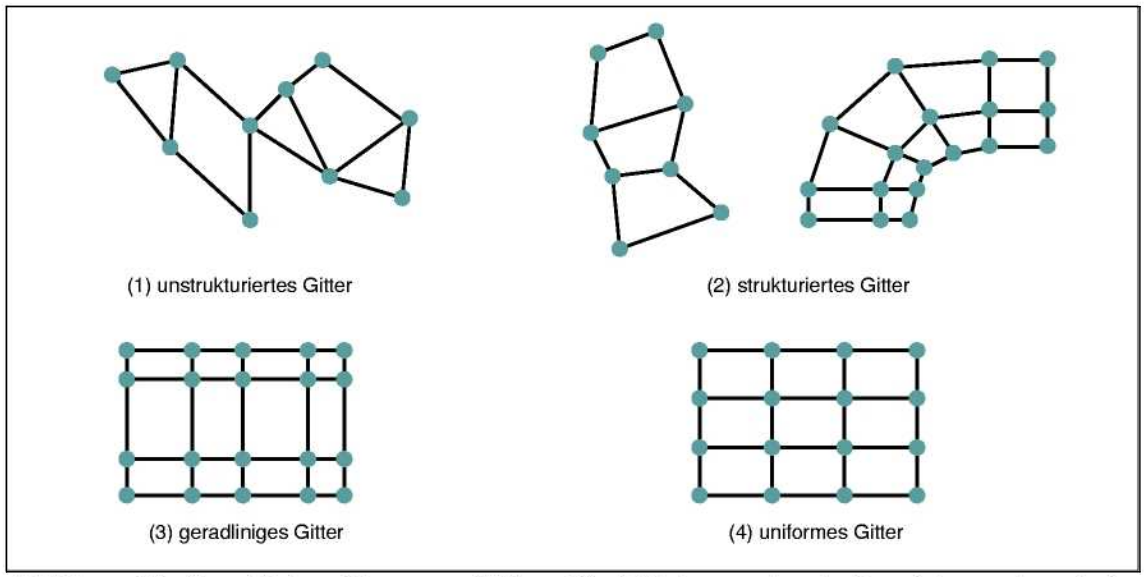


Abbildung 2.1: Verschiedene Typen von Gittern: Die Gittertypen unterscheiden sich zum einen darin, wie die Knoten untereinander angeordnet sind, und zum anderen darin, wie die Position der Knoten im Raum festgelegt ist. Das allgemeinste Gitter ist das unstrukturierte Gitter (1). Bei den strukturierten Gittern (2) gibt es eine regelmäßige Verbindung der einzelnen Knoten. Nur bei dem geradlinigen (3) und uniformen Gitter (4) gibt es für die Position der Knoten Einschränkungen.

Bei strukturierten Gittern können die Daten in n-dimensionalen Feldern abgespeichert werden. Die Zuordnung der Daten zu den Knoten des Gitters ergibt sich implizit. Bei unstrukturierten Gittern müssen die einzelnen Knoten in Listen (eindimensionale Felder) gespeichert werden. So ergeben sich zwei Listen: die Liste der Knoten und die Liste der Verbindungskanten. Die bei vielen Rechnungen benötigten Informationen über räumliche Nachbarn müssen zusätzlich in Nachbarschaftslisten gespeichert werden.

Der zusätzliche Aufwand für unstrukturierte Gitter bei der Speicherung und auch bei den Rechnungen auf diesen Gittern schränkt deren Einsatzgebiet ein. So werden solche Gitter immer dann eingesetzt, wenn es sich um stark unregelmäßige Anordnungen handelt (z.B. Strömungssimulationen). Auch in adaptiven Simulationsrechnungen werden unstrukturierte Gitter eingesetzt. Dort wird z.B. das Gitter nur an den Stellen verfeinert, an denen der Fehler bei der numerischen Berechnung noch zu groß ist. Dies kann dann zu einem speziellen Gitter führen, das aus einer Anzahl von Teilgittern besteht, die selbst wieder strukturiert sind. Viele Visualisierungssysteme unterstützen solche *Arrays of Fields*.

Die einfache Datenhaltung und der schnelle Zugriff auf die einzelnen Daten in einem uniformen Gitter erklären die weite Verbreitung dieses Gittertyps bei numerischen Simulationen.

Pro Gitterknoten können beliebige Daten gespeichert sein. Üblich sind skalare Daten (z.B. Druck- oder Dichtewerte), Vektor-Daten (z.B. Geschwindigkeitsvektoren, Tensoren) oder wieder mehrdimensionale Daten (z.B. Matrizen in der Quantenchromodynamik).

Diese oben beschriebenen gitterbasierten Daten müssen im Falle der Online-Visualisierung komprimiert, zur Visualisierungsworkstation übertragen und dort dargestellt werden. Der Gittertyp ist hierbei auch entscheidend für den Aufwand und die Art der möglichen Komprimierung.

2.3 Größe der Simulationsdaten

Die Größe der von Simulationsprogrammen berechneten Daten steigt mit der Leistungsfähigkeit der dabei verwendeten Rechner. Für das Jahr 1999 wurde in [6] aufgeführt, daß bei durchschnittlichen Simulationen auf Supercomputern Daten in der Größe von 350 GByte bis 10 TByte erzeugt

werden. Für das Jahr 2004 soll nach Untersuchungen im Rahmen des ASCI-Initiative die durchschnittliche Ausgabemenge von Simulationsprogrammen auf 12 TByte steigen [8]. Auch auf den Rechnern des John von Neumann Institute für Computing (NIC) sind solche Datengrößen zu erwarten. Bei einer Hauptspeichergröße von über 250 GByte des CRAY T3E-1200 können z.B. bei Strömungssimulationen pro Zeitschritt Ausgabedaten im Bereich von mehreren Gigabytes erzeugt werden.

Sollen solche Daten nach Ablauf der Simulation analysiert und visualisiert werden, müssen sie entweder auf eine Visualisierungsworkstation übertragen und dort für die weitere Analyse gespeichert werden oder vor Ort auf dem Supercomputer verarbeitet werden. Im ersteren Fall müssen auf der Visualisierungsworkstation genügend große Platten und für die Verarbeitung entsprechend viel Hauptspeicher vorhanden sein. Ist dies nicht der Fall, können die Daten in Originalgröße nur auf dem Supercomputer direkt verarbeitet werden, was aber sehr kostspielig ist, zumal Supercomputer nicht für Visualisierungsaufgaben ausgelegt sind.

Aber auch bei der direkten Visualisierung der Daten ohne Zwischenspeicherung (Computational Steering) treten bei diesen Datengrößen Probleme auf. Ggf. steht für die Übertragung der Daten zwar noch genügend Bandbreite zur Verfügung, aber die Datenhaltung und die Visualisierung auf der lokalen Workstation wird bei dieser Datengröße nicht mehr möglich sein. Auch sollte der Prozeß der Übertragung und Visualisierung eine gewisse Zeitdauer nicht überschreiten. Die Daten des aktuellen Rechenschritts sollten auf dem Bildschirm zu sehen sein, bevor der Supercomputer den nächsten Rechenschritt beendet hat. Ansonsten ist ein Reagieren auf die aktuelle Situation schwierig.

Bei der Anbindung einer Steering-Komponente an eine große Simulation entstehen also mehrere Probleme (Speichergröße, Bandbreite, Verarbeitungsgeschwindigkeit der Visualisierung), die aber alle von der Datengröße verursacht werden. Sinnvoll ist es daher, die Ursache der Probleme durch eine angepaßte Reduzierung der Datengröße zu beseitigen.

2.4 Eignung einer Simulation für Computational Steering

Die Anbindung einer Visualisierungs- und Steuerungskomponente an ein Simulationsprogramm macht nur bei bestimmten Randbedingungen Sinn. So sollte die Berechnung in einzelne Rechenschritte aufgeteilt sein, so daß am Ende eines jeden Rechenschritts ein Zwischenergebnis vorliegt. Das Zwischenergebnis kann dann zur Visualisierungskomponente übertragen und angezeigt werden. Kann die Berechnung nicht in mehrere Schritte aufgeteilt werden, liegt das Ergebnis auch erst am Ende der Rechenzeit vor. Eine Online-Visualisierung bringt in diesem Fall keine Vorteile.

Die Rechenzeit pro Simulationsschritt muß in gewissen Grenzen liegen. Dauert ein Rechenschritt zu lange, ist die direkte Verfolgung der Ergebnisse am Bildschirm zu langwierig. Die Beobachtung, inwieweit Parameteränderungen Auswirkungen zeigen, ist damit unmöglich. Die Online-Visualisierung kann dann nur noch dazu genutzt werden, fehlgeleitete Simulationsrechnungen vorzeitig abzubrechen. Ist im Gegensatz dazu die Rechenzeit pro Simulationsschritt zu gering, steht dem Anwender nicht mehr genügend Zeit zur Verfügung, auf die Ergebnisse des aktuellen Simulationsschritts zu reagieren. Bevor die Daten übertragen und angezeigt werden können, liegen auf der Simulationsseite wieder neue Daten vor. Parameter-Anpassungen und Ergebnisse von Zwischenschritten sind damit nicht mehr synchron.

Natürlich sollten die der Online-Visualisierung zur Verfügung stehenden Daten in einer Form vorliegen, die eine graphische Darstellung erfordern. So können z.B. skalare Werte direkt auf dem Terminal ausgegeben werden.

Vor dem Einsatz einer Visualisierungskomponente sollte auch überprüft werden, ob während der Simulationsrechnung Daten erzeugt werden, bei deren Visualisierung neue Erkenntnisse erzielbar sind und damit die Simulationsrechnung positiv beeinflußt werden kann. Obwohl in manchen Simulationen zwar mehrdimensionale Daten erzeugt werden, kann z.B. die Güte einer Simulationsrechnung nur an globalen Werten, z.B. der Gesamtenergie eines Systems, erkannt werden. Dann sollte auch nur dieser skalare Wert beobachtet werden. Der Einsatz von Computational Steering ist immer dann sinnvoll, wenn Parameter erst während der Rechnung effektiv bestimmt werden können.

Neben der Online-Visualisierung ist auch die Steuerung der Simulation wichtig. Dazu muß die Simulationsrechnung so parametrisiert sein, daß während der Laufzeit Parameter verändert und so die Rechnung beeinflußt werden kann. Fehlen solche Parameter, steht dem Anwender nur noch die Möglichkeit des Abbruchs der Simulation zur Verfügung.

2.5 Visualisierungstechniken bei Computational Steering

Gegenüber der Visualisierung im *Post-Processing* sollte die Visualisierung im Falle des Computational Steerings anderen Anforderungen genügen. Sinn und Zweck des Computational Steerings ist die Beobachtung und Steuerung der Simulation. Die Visualisierung soll hier dazu beitragen, daß Probleme oder falsch eingestellte Parameter der Simulation frühzeitig erkannt werden können. Wichtiger als eine High-End-Visualisierung ist also eine schnelle und aktuelle Darstellung der Zwischenergebnisse. Kompromisse bei der Darstellung sollten und müssen dabei eingegangen werden. Im Gegensatz zur High-End-Visualisierung sollte das Computational Steering von beliebigen Orten und Rechnern aus möglich sein. Eine spezielle Graphik-Hardware ist dabei nicht immer vorhanden. Trotzdem sollte die Beobachtung der Simulation möglich sein. Daneben steht der Visualisierung für die Anzeige der Daten oft nur wenig Zeit zur Verfügung. In der Zeit, welche die Simulation für die Berechnung eines Simulationsschritts braucht, müssen die Daten angezeigt werden und der Betrachter die Simulationsparameter auch verändern können. Die beiden Bedingungen schränken die Darstellungmöglichkeiten und Visualisierungstechniken zwangsläufig ein.

Wie schon aus den vorhergehenden Kapiteln hervorgeht, handelt es sich bei den von Simulationsprogrammen gelieferten Daten häufig um gitterbasierte Daten, die oft auf einem dreidimensionalen Simulationsgebiet beruhen. Dies gilt insbesondere, wenn das Hauptaugenmerk – wie in dieser Arbeit – auf Simulationsrechnungen mit sehr großen Datenmengen gelegt wird. Die zu visualisierenden Daten stellen also einen vom Simulationsprogramm berechneten und auf einem Gitter diskretisierten Zustand eines oder mehrerer physikalische Parameter (z.B. Strömunggeschwindigkeit, Druck, ...) dar. Dabei können die Daten diesen Zustand entweder für die einzelnen Knotenpunkte des Gitters oder für die durch das Gitter definierten einzelnen Raumelemente beschreiben. Handelt es sich um ein geradliniges Gitter, beschreiben die einzelnen Raumelemente einen Quader oder Würfel und werden als *Voxel* bezeichnet. Werte an einem Gitterpunkt beschreiben den Zustand genau an der Stelle des Knotenpunkts, während Voxelwerte einen Mittelwert über alle möglichen Orte innerhalb des Raumelements darstellen.

Man kann die Visualisierungstechniken für solche Daten in zwei Gruppen unterteilen: In der ersten Gruppe wird auch in der Darstellung die Diskretisierung beibehalten (siehe Abb. 2.2 auf der nächsten Seite); für jeden Gitterpunkt bzw. für jedes Voxel wird ein graphisches Element erzeugt, das durch Größe, Farbe, Form oder Richtung den Wert an dieser Stelle repräsentiert. In der zweiten Gruppe werden aus den Daten graphische Elemente abgeleitet, die eine von der Diskretisierung unabhängige Darstellung der Daten erlauben (siehe Abb. 2.3 auf der nächsten Seite).

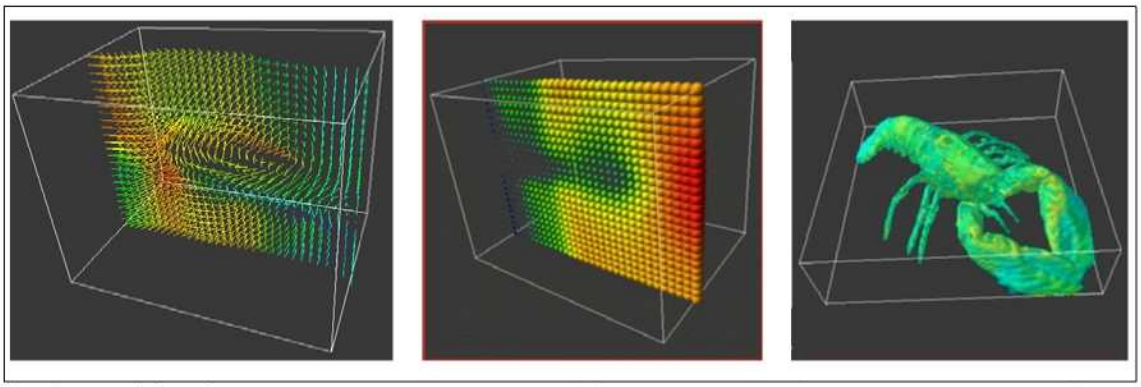


Abbildung 2.2: Diskrete Darstellung: Das erste Bild zeigt einen Ausschnitt aus einem Vektorfeld. An jedem Gitterknoten wird ein Vektor erzeugt, der aus den drei Richtungkomponenten der Daten gebildet wird. Die Farbe gibt dabei die Länge des Vektors wieder. Das mittlere Bild zeigt eine mögliche Darstellung für skalare Daten an jedem Gitterknoten. Hier wird der Wert durch die Größe der Kugel und deren Farbe repräsentiert. Das dritte Bild zeigt die Darstellung eines skalaren Feldes mit Hilfe eines *Volume Renderers*. Neben einer Farbe wird hier jedem Element auch ein *Alpha*-Wert zugeordnet, der die Durchsichtigkeit bestimmt.

Die direkte diskrete Darstellung der Simulationsergebnisse erlaubt nicht immer eine schnelle Interpretation der Daten. Zum Beispiel können interessante Bereiche des Simulationsgebiets durch weniger wichtige Daten verdeckt sein. Zum Teil hilft dabei die Einschränkung der darzustellenden Objekte auf einen Teilbereich (*Slice*, *Zoom*) oder auf Elemente, die Daten über einen gewissen Schwellwert beschreiben (*Threshold*). Alternativ zur diskreten Darstellung können die Daten weiter verarbeitet und daraus graphische Objekte abgeleitet werden, die eher die globale Struktur der Daten beschreiben. Dies sind z.B. Schnittebenen, Iso-Oberflächen oder -Konturen oder Strömungslinien eines Geschwindigkeitsfeldes.

Beide Darstellungsarten bieten Vor- und Nachteile. So steigt z.B. die Anzahl der graphischen Objekte bei der ersten Gruppe linear mit der Anzahl der Knoten des Gitters. Die Anzahl der noch darstellbaren Objekte hängt dabei stark von der Graphik-Hardware ab, ist aber in jedem Fall nach oben begrenzt. Auch bei der zweiten Gruppe spielt die Anzahl der Gitterknoten eine wesentliche Rolle. Für die Berechnung einer Iso-Oberfläche muß der Marching-Cube-Algorithmus [9] z.B. auch alle Gitterpunkte bzw. Voxel betrachten. Im Unterschied zur ersten Gruppe muß dies aber

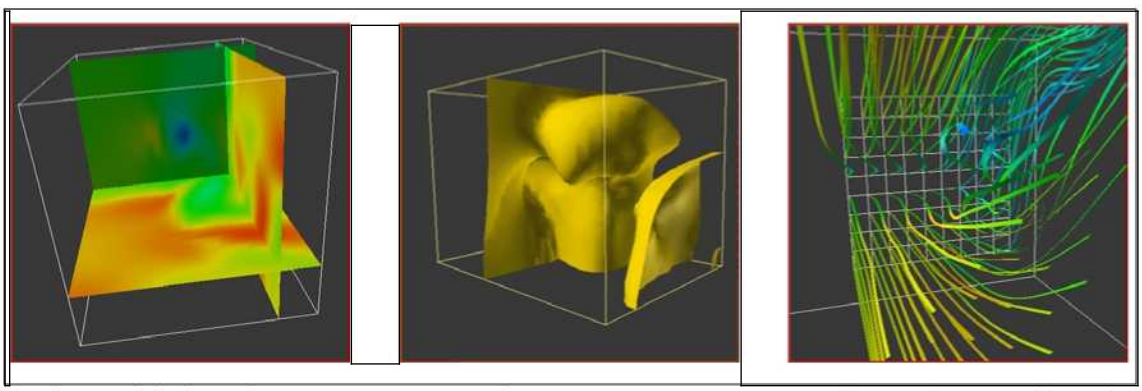


Abbildung 2.3: Aus diskreten Daten abgeleitete Darstellung: Das erste Bild zeigt die Darstellung der Daten mit Hilfe von Schnittebenen. Die Farbgebung der Ebenen stellt dabei die aus den Gitterdaten interpolierten Werte an den jeweiligen Stellen dar. Die Schnittebenen müssen nicht orthogonal angeordnet sein, sondern können beliebige Winkel annehmen. Im zweiten Bild ist eine Iso-Oberfläche dargestellt. Die Oberfläche wird durch diejenigen Punkte im Raum definiert, deren Daten einen vorgegebenen Wert haben. Das dritte Bild zeigt die Darstellung mit Hilfe sogenannter *Streamlines*. Diese Bänder beschreiben die Flugbahnen von virtuellen Partikeln in einem Strömungsfeld. Der Betrachter bestimmt dabei die Startpunkte der einzelnen Partikel.

nur bei einer Aktualisierung der Daten und nicht bei jeder Veränderung der Betrachterposition geschehen. Die resultierende Oberfläche ist von ihrer Komplexität eher geringer und erzeugt bei der Darstellung weniger Performance-Engpässe. Die hohe Anzahl graphischer Elemente der ersten Gruppe muß dagegen bei jeder Veränderung des Betrachterstandpunktes verarbeitet werden.

Die in den nachfolgenden Kapiteln beschriebenen Strategien zur Ankopplung von Visualisierungskomponenten an ein Simulationsprogramm mit verteilter Datenhaltung beinhalten die Möglichkeit, Teile eines Simulationsgebiets einzeln zu übertragen und der Visualisierung zu übergeben. Hier können die Visualisierungstechniken der ersten Gruppe vorteilhafter eingesetzt werden, da dort die Teilgebiete in der graphischen Verarbeitungskette getrennt und unabhängig voneinander bearbeitet werden können.

Den gitterbasierten Daten stehen solche gegenüber, die nicht an ein Gitter gebunden sind und zum Beispiel bei Particle-Tracking, Monte Carlo- oder Molekulardynamik-Simulationen anfallen. Es handelt sich häufig um Objekte (Partikel, Atome), die mit Attributen (Ort, Geschwindigkeit, Art, ...) versehen sind. Es bietet sich also an, diesen Objekten jeweils ein graphisches Element (Kugel, Vektor, ...) zuzuordnen und somit die Daten darzustellen. Die Attribute können, wie in der zuvor beschriebenen ersten Darstellungsart, in Farbe, Form und Durchsichtigkeit der graphischen Objekte überführt werden. Man wird aber auch hier sehr schnell an die obere Grenze der noch darstellbaren graphischen Elemente kommen. Ein Übergang zu einer globalen Darstellungsart aus der zweiten Gruppe stellt sich hier aber schwieriger dar, da keine Diskretisierung der Ortskoordinaten vorhanden ist. Für eine Darstellung z.B. von Dichteverteilungen von Partikeln muß vorher ein Gitter erzeugt werden und dieses mit den Informationen über die Anzahl der Partikel in dem jeweiligen Volumenelement gefüllt werden. Im Falle des Computational Steering kann diese aufwendige und speicherintensive Operation sowohl auf der Simulations- als auch auf der Visualisierungsseite ausgeführt werden.

Eine andere Art, die Komplexität der Daten zu reduzieren, ist die des *Level-of-Detail*. Dabei werden je nach Betrachterstandpunkt für die einzelnen Teilobjekte jeweils andere Auflösungen gewählt. So werden Objekte, die sich nahe am Betrachterstandpunkt befinden, höher aufgelöst als diejenigen, die sich davon weiter weg befinden. Ein typisches Beispiel dafür ist die Visualisierung von Molekülen, bei denen einzelne Strukturen nur teilweise eingeblendet werden. Diese Methode wird aber nicht von der jeder Visualisierung unterstützt.

2.6 Parallele Programme mit verteilter Datenhaltung

Die für große Simulationsrechnungen benötigte Rechenleistung und Hauptspeichergröße können von Einzelprozessorsystemen nur bedingt bereitgestellt werden. Die Rechenleistung eines Einzelprozessorsystems wird durch eine Vielzahl von Faktoren begrenzt. Zum Beispiel kann ein Prozessor nur eine gewisse Anzahl von Gleitpunktoperationen pro Sekunde ausführen, die von der Taktrate und der Anzahl der Floating Point-Einheiten im Prozessor abhängen. Darüber hinaus ist auch der Hauptspeicher und dessen Zugriffsgeschwindigkeit (Bandbreite) ein begrenzender Faktor. Große Simulationsrechnungen können deshalb oft nur auf Vektor- oder Parallelrechner ausgeführt werden. Die durch optimierte Zugriffsmechanismen und Pipeline-Verarbeitung innerhalb der Prozessors auf die Rechnungen mit Vektoren spezialisierten Vektorrechner werden dabei immer mehr durch Parallelrechner abgelöst. Ein Grund für diese Entwicklung ist, daß Parallelrechner sehr einfach aus vielen Einzelprozessorsystemen aufgebaut werden können und dadurch wesentlich kostengünstiger sind als die mit Spezialprozessoren ausgerüsteten Vektorrechner. Die Gesamtkapazität eines Parallelrechners ist dabei im wesentlichen von der Anzahl der Einzelprozessoren abhängig und kann durch Hinzufügen weiterer Prozessoren sehr einfach erhöht werden. Diesen Vorteilen der geringen Kosten und der Skalierbarkeit der Rechenleistung steht als Nachteil

ein komplizierteres Programmiermodell entgegen. Da das Simulationsprogramm nun gleichzeitig auf einer Menge von Prozessoren läuft, muß die Verteilung der Arbeit auf die einzelnen Prozessoren, die Koordination des Programmablaufs und der Zugriff auf gemeinsame Daten geregelt sein. Der Aufbau eines parallelen Simulationsprogramms, die dabei benutzte Verteilung der Daten und die Art des Parallelrechners bestimmen auch die Strategie, die bei der Ankopplung einer Online-Visualisierungs- und Steuerungskomponente angewendet werden kann.

2.6.1 Aufbau eines Parallelrechners

Der Aufbau eines Parallelrechners wird im wesentlichen durch die drei Komponenten Prozessor, Hauptspeicher und das verbindende Kommunikationsnetzwerk bestimmt. Aus diesen Komponenten lassen sich verschiedenartige Parallelrechner zusammenstellen. Generell kann man dabei zwischen *Shared-Memory* und *Distributed Memory* Systemen unterscheiden [10].

Bei einem *Shared-Memory*-System teilen sich alle Prozessoren einen gemeinsamen Hauptspeicher (Abb. 2.4). Der Zugriff auf diesen Hauptspeicher wird dabei durch ein Kommunikationsnetzwerk geregelt, das insbesondere auch dafür sorgen muß, daß bei der Verwendung von lokalen Caches die Konsistenz der Daten sichergestellt ist. Der ggf. aus mehreren Komponenten aufgebaute Hauptspeicher besitzt einen gemeinsamen Adreßraum, auf den jeder Prozessor zugreifen kann.

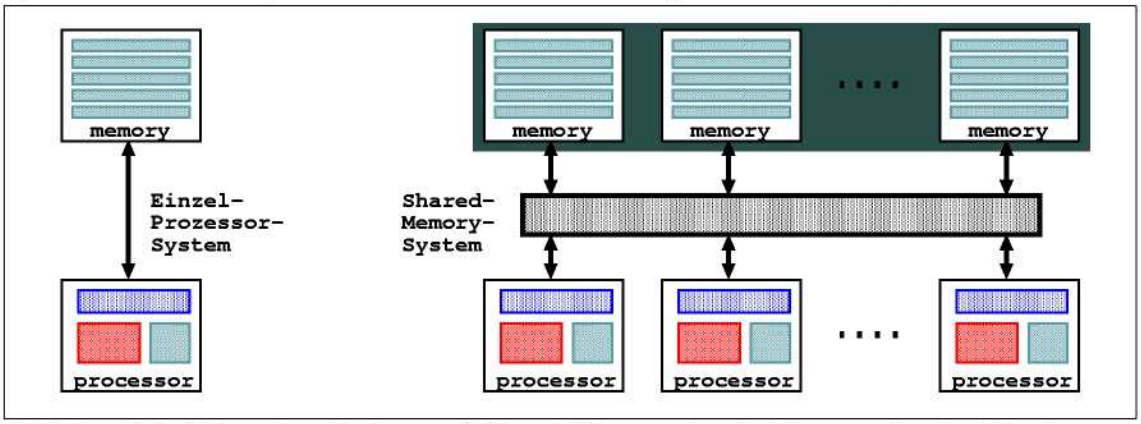


Abbildung 2.4: Aufbau eines Rechners mit Shared-Memory: Aus den Komponenten eines Einzelprozessorsystems (links) wird ein Parallelrechner mit gemeinsamen Hauptspeicher aufgebaut (rechts). Die direkte Verbindung zwischen Prozessor und Hauptspeicher muß bei dem Parallelrechner durch ein aufwendigeres Verbindungsnetzwerk ersetzt werden.

Die Komplexität des Verbindungsnetzwerks erhöht sich mit der Anzahl der Prozessoren, die auf den gemeinsamen Hauptspeicher zugreifen. Typische Größen solcher Parallelrechner sind 16 bzw. 32 Prozessoren. Eine weitere Erhöhung der Prozessoranzahl erzwingt ein anderes Verbindungsdesign. Das Verbindungsnetzwerk kann z.B. in mehrere hierarchisch angeordnete Stufen aufgeteilt werden (NUMA-Architektur, *nonuniform memory access*). Bei größeren Prozessoranzahlen erhöht sich dabei aber nicht nur die Komplexität, sondern auch die Zugriffszeit auf den Hauptspeicher.

Da es einen gemeinsamen Hauptspeicher gibt, ist die Programmierung für ein Shared-Memory-System sehr an die eines Einzelprozessorsystems angelehnt. Wird z. B. für die Programmierung OpenMP [11] verwendet, muß ein sequentielles Programm nur mit Compiler-Direktiven versehen werden, welche die parallelen Teile (Schleifen) des Programms sowie die privaten und globalen Variablen markieren. Die Parallelisierung und der Zugriff auf die im gemeinsamen Hauptspeicher liegenden Daten wird dann vom Compiler festgelegt.

Die Parallelisierung kann auch auf Basis von *Threads* [12] erfolgen. Hierbei werden mehrere Kopien eines Prozesses gestartet. Der Zugriff auf gemeinsame Daten und die Koordination der einzelnen Prozesse erfolgt über sogenannte *Semaphor*-Variablen, die ein blockierendes und damit

konsistentes Schreiben eines Variablenwerts erlauben. Diese aufwendigere Parallelisierung liegt dabei in den Händen des Programmierers.

Eine Steigerung der Leistungsfähigkeit von Shared-Memory-Systemen ist durch die Komplexität des Kommunikationsnetzwerks für den Zugriff auf den gemeinsamen Hauptspeicher begrenzt. Diese Komplexität wird bei den Parallelrechnern mit verteiltem Speicher vermieden (siehe Abbildung 2.5).

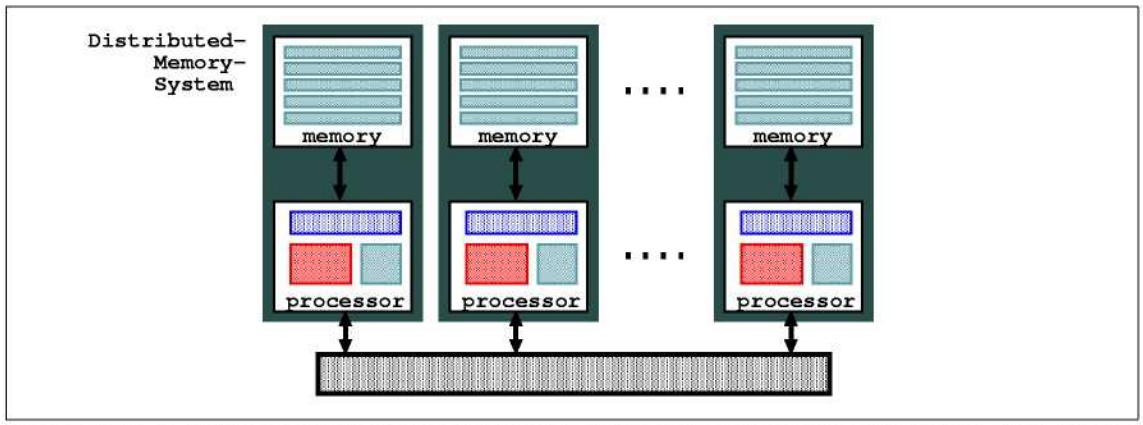


Abbildung 2.5: Aufbau eines Rechners mit Distributed-Memory: Jeder Prozessor besitzt einen eigenen Hauptspeicher und kann auf entfernten Hauptspeicher nur indirekt und unter Zuhilfenahme des entfernten Prozessors zugreifen.

Jeder Prozessor besitzt einen eigenen lokalen Hauptspeicher, auf den nur er zugreifen kann. Daten können nur mit Hilfe von expliziten Nachrichten zwischen den einzelnen Prozessoren ausgetauscht werden.

Ein Vorteil solcher Systeme ist der einfache Aufbau. Im einfachsten Fall können Einzelprozessorsysteme verwendet werden, die durch ein schnelles Netz (z.B. Ethernet, Myrinet, ...) miteinander verbunden sind. Da nur Standard-Komponenten verwendet werden, sind die Kosten wesentlich geringer als bei Shared-Memory-Systemen. Eine Erweiterung des Rechners erfolgt durch Hinzufügen weiterer Einzelprozessorsystemen (PCs, Workstations). Begrenzender Faktor ist hier die Bandbreite des Kommunikationsnetzwerks. Derzeit werden Parallelrechner mit mehr als 10000 Prozessoren realisiert. Die Programmierung solcher Systeme ist jedoch aufwendiger und komplizierter. Da es keinen gemeinsamen Hauptspeicher gibt, müssen die gemeinsamen Daten innerhalb des Programms auf die Prozessoren verteilt, dort bearbeitet und später wieder eingesammelt werden. Der Austausch von Nachrichten muß dabei explizit im Programm kodiert werden. Zur Anwendung kommen dabei die Kommunikationbibliotheken MPI (Message-Passing Interface, [13]) oder PVM (Parallel Virtual Machine, [14]). Eine Portierung eines sequentiellen Programms auf einen Parallelrechner erfordert dabei eine vollständige Anpassung der logischen Struktur an die neuen Bedingungen. Trotzdem haben sich solche Systeme gerade wegen der guten Skalierbarkeit und der preisgünstigen Einzelkomponenten durchgesetzt.

Auch bei den Distributed-Memory-Systemen gibt es Spezialrechner, die z.B. die Übertragungsgeschwindigkeit zwischen den Prozessoren mit komplexer Hardware erhöhen. Ein Beispiel dafür ist der Rechner CRAY T3E-1200 (512 Prozessoren), auf dem auch ein Teil der in dieser Arbeit vorgestellten Testläufe durchgeführt worden sind (siehe Abb. 2.6 auf der nächsten Seite). Dieser Rechner benutzt zwar Standard-Prozessoren (Alpha-Chip), verbindet diese aber über spezielle Kommunikationsprozessoren und ein dreidimensionales Netzwerk (3D-Torus) miteinander. Durch die Spezialprozessoren kann eine Bandbreite bis zu 320 MB/s zwischen zwei Prozessoren erreicht werden. Durch den Torus steht diese Kommunikationsleistung gleichzeitig für mehrere Prozessorpaare zur Verfügung.

Derzeit setzen sich Systeme durch, die eine Kombination von Shared-Memory-Systemen und Distributed-Memory-Systemen darstellen. Dabei werden eine Vielzahl von Shared-Memory-Rech-

nern (SMP-Knoten, Symetric Multiprocessor) mit einem schnellen Netz zusammengefügt. Die SMP-Knoten besitzen dabei nur eine geringe Anzahl von Prozessoren (4-32). Unter anderem ist auch der durch Einsatz von SMP-Knoten geringere Platz- und Stromverbrauch ein Grund für diese Entwicklung. Die Kommunikation innerhalb der SMP-Knoten kann über Shared-Memory erfolgen. Die Kommunikation zwischen zwei Knoten muß über das Verbindungsnetzwerk erfolgen. Auch im Programmiermodell schlägt sich diese zweistufige Hierarchie nieder: Innerhalb des SMP-Knoten kann für die Parallelisierung z.B. OpenMP, zwischen den SMP-Knoten MPI oder PVM benutzt werden. Dieses Modell wird als *hybrides* Programmiermodell bezeichnet. Ein Beispiel für einen solchen Rechner ist das in dieser Arbeit verwendete und im ZAM aufgebaute Linux-Cluster ZAMpano. Es besteht aus insgesamt 32 Prozessoren, die in je 8 SMP-Knoten angeordnet und über ein Myrinet-Netzwerk (1.5 GBit/s) miteinander verbunden sind.

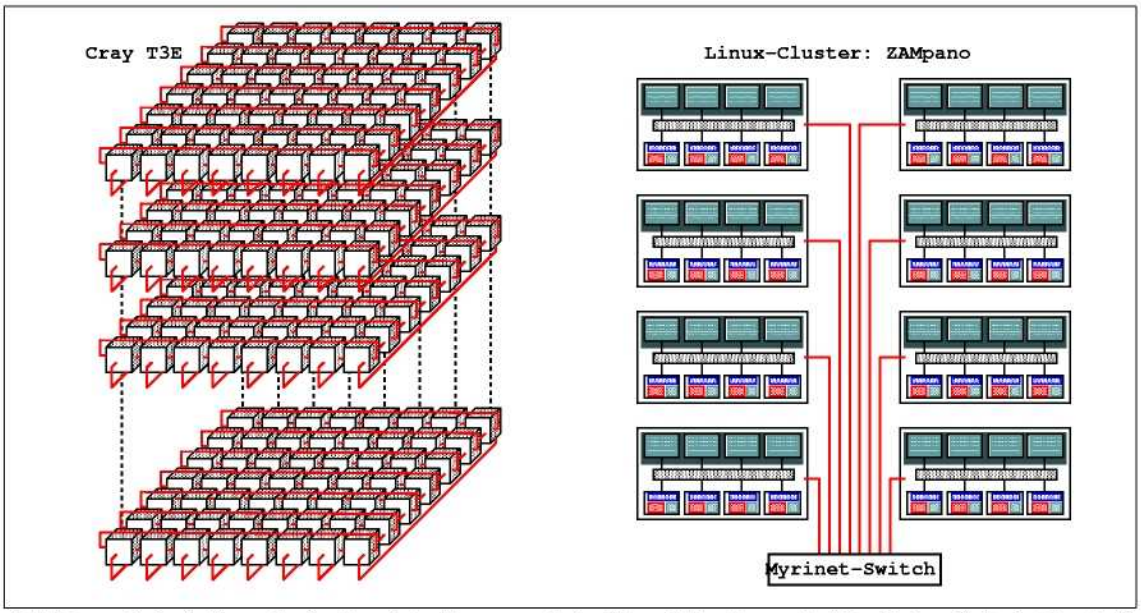


Abbildung 2.6: Aufbau der in der Arbeit verwendeten Parallelrechner: Auf der linken Seite ist der Aufbau des CRAY T3E gezeigt. Es handelt sich dabei um einen 3D-Torus. Jeder Prozessor ist mit 6 Nachbarn verbunden. Auf der rechten Seite ist ein im ZAM aufgebautes Linux-Cluster darstellt, das aus 8 4-Wege-SMP-Knoten besteht.

Nach der Klassifikation nach Flynn [15] kann ein Einzelprozessorsystems als ein *single instruction stream - single data stream*-Computer (SISD) bezeichnet werden. Es gibt nur einen Programmcode, der mit einem Datenstrom arbeitet. Bei einem Parallelrechner wird auf jedem Prozessor ein eigener Programmcode abgearbeitet. Es kann sich dabei um eine Kopie eines einzigen Programmcodes oder um jeweils verschiedene Programmcodes handeln. Bei Rechnern mit verteiltem Speicher arbeiten die Prozessoren auf verschiedenen Daten. Im Falle eines gemeinsamen Programmcodes kann der Rechner als *single instruction stream - multiple data stream*-Computer (SIMD) klassifiziert werden kann. Werden auf den einzelnen Prozessoren verschiedene Programmcodes ausgeführt, wird der Rechner als *multiple instruction stream - multiple data stream*-Computer (MIMD) bezeichnet.

2.6.2 Verteilung der Daten in einer parallelen Anwendung

Bei der Portierung von Simulationsprogrammen, die für ihre Rechnungen große Datenmengen benötigen, auf Parallelrechner mit verteiltem Speicher muß nicht nur die Rechnung in Teile zerlegt und auf die einzelnen Prozessoren verteilt werden, sondern auch die damit verbundenen Daten. Bei der *Gebietszerlegung* oder *Partitionierung* wird das der Simulationsrechnung zugrunde liegende Simulationsgebiet in Teilbereiche aufgeteilt. Jeder Prozessor erhält einen solchen Teilbereich und führt die Simulationsrechnung darauf aus. Die Eingabedaten müssen vor der Berechnung auf die Prozessoren verteilt und die Ergebnisdaten nach der Berechnung wieder eingesammelt werden.

Diese Aufgabe wird typischerweise von einem als *Master* bezeichneten Prozessor ausgeführt. Er ist für die Verteilung der Aufgaben, die Koordinierung der *Slave*-Prozessoren sowie für das Einsammeln der Daten zuständig. Diese Art der Zusammenarbeit wird daher als *Master-Slave*-Verfahren bezeichnet.

Bei vielen Simulationsrechnungen ist das Simulationsgebiet so groß, daß es nur verteilt gespeichert werden kann. Der Hauptspeicher eines einzelnen Prozessors genügt dafür nicht, so daß die Ergebnisdaten nicht von einem *Master*-Prozessor eingesammelt werden können. Üblicherweise schreibt in solchen Fällen jeder Prozessor die Ergebnisdaten in eigene Ausgabedateien. Da auch das Einlesen der Eingabedaten auf diese Weise geschieht, ist der *Master* dann nur noch für die Koordination der Rechnung notwendig. Meistens wird seine Rechenleistung mit genutzt, in dem er in die verteilte Berechnung eingebunden wird. Für die Anbindung einer Visualisierung bedeutet dies, daß auch hier das gesamte Simulationsgebiet mangels Hauptspeicher nicht auf dem *Master*-Prozessor eingesammelt werden kann. Hier können die Simulationsergebnisse der Teilgebiete nur nacheinander von den *Slave*-Prozessoren abgeholt und verschickt werden. Abbildung 2.7 veranschaulicht dieses Prinzip.

In dieser Arbeit liegt der Fokus auf solchen Simulationsprogrammen mit großem Speicherbedarf. Die Größe der Daten verhindert, daß die Daten gesammelt und direkt zur Visualisierung geschickt werden können. Um auch hier eine Visualisierung anbinden zu können, müssen die Daten in ihrer Größe reduziert und komprimiert werden.

Die bei vielen Simulationsprogrammen während der Rechnung durchgeführte und für den Lastausgleich wichtige Neuverteilung der Teilgebiete erzwingt bei der Anbindung der Visualisierung eine gewisse Dynamik. Dazu sollten die Größen und Positionen der Teilgebiete innerhalb des Visualisierungsprogramms nicht festgelegt sein, sondern in jedem Simulationsschritt mit übertragen werden.

Ein weiterer Punkt, der bei der Ankopplung an Simulationsprogramme mit verteilter Datenhaltung berücksichtigt werden muß, ist, daß die Teilgebiete in vielen Fällen etwas größer definiert werden und mit den benachbarten Teilgebieten überlappen. Diese Ränder werden bei bestimmten iterativen Berechnungen wie z.B. beim parallelen konjugierten Gradienten-Verfahren (CG) in jedem Iterationschritt ausgetauscht und ermöglichen so eine globale Berechnung des Ergebnisses. Bei der Darstellung solcher Teilgebiete muß diese Überlappung berücksichtigt und ggf. ein Mittelwert von mehrfach vorhandenen Datenelementen berechnet werden.

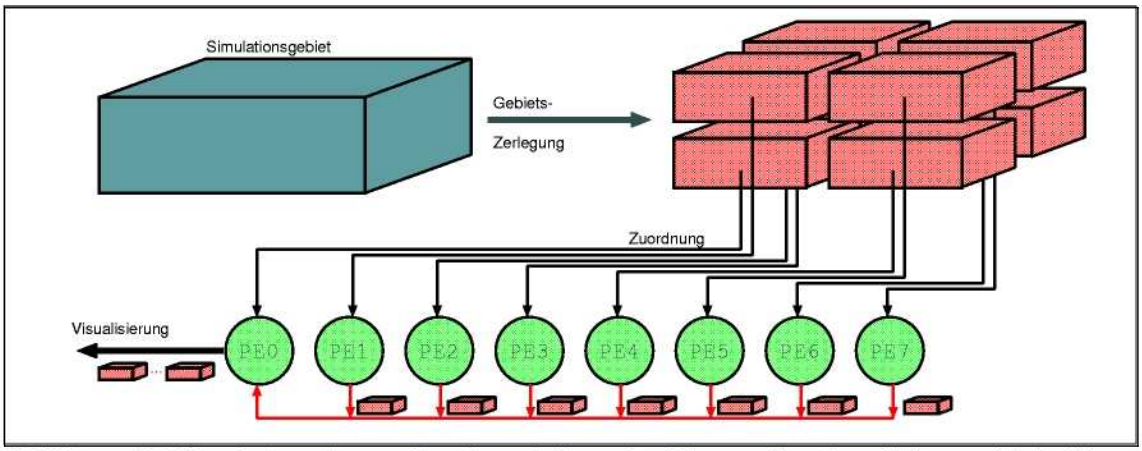


Abbildung 2.7: Simulation mit verteilter Datenhaltung: Bei der verteilten Datenhaltung wird das Simulationsgebiet in eine der Anzahl von Prozessoren entsprechende Anzahl von Teilgebieten zerlegt und den Prozessoren zugeordnet. Bei der Ankopplung der Visualisierung muß wegen des begrenzten lokalen Hauptspeichers dafür gesorgt werden, daß sich auf einem Prozessor nie mehr als ein solches Teilgebiet befindet. Eine mögliche Lösung für dieses Problem ist das sequentielles Verschicken der Teilgebiete über den ersten Prozessor (*Master*, PE0).

2.7 Tools für Computational Steering

Bei aufwendigen Simulationsrechnungen ist es oft zweckmäßig, Zwischenergebnisse bereits während der Laufzeit des Simulationsprogramms zu kontrollieren. Entsprechende Libraries und Tools zur Unterstützung des Computational Steering entstanden zuerst aus Anwendungen heraus. So wurden spezielle Visualisierungs- und Steuerungs-Tools für Strömungssimulationen oder der Simulation für die Molekulardynamik entwickelt [16]. Mit der Generalisierung solcher Tools entstanden eine Vielzahl von Systemen, die sich in ihrem Anwendungsbereich, der Architektur und den Benutzerinterfaces unterscheiden.

Die meisten Systeme basieren auf einer Client-Server-Architektur, welche die Verbindung zum Simulationsprogramm und zur Visualisierung herstellen. Vielfach besitzt der Client-Prozeß selbst schon Visualisierungsfunktionen. Beispiele für solche Systeme sind Progress (Magelan) [17, 18], CUMULVS [19, 20], VIPER [21], CSE [22] und VISIT [2].

Andere Systeme bieten eine integrierte Entwicklungsumgebung an, mit der sowohl die Visualisierungs- und Steuerungs-Funktionalität als auch die Simulationsrechnungen möglich sind. Da hier beide Seiten des Computational Steering in einem Tools vereint sind, können sehr leicht komplexe Steuerungs- und Visualisierungsprozesse entwickelt werden. Als Beispiel hierfür ist SCIrun [23, 24] zu nennen, daß ähnlich zu dem in dieser Arbeit benutztem Visualisierungssystem AVS/Express [25] ein Datenflußmodell zur Definition der Applikation benutzt. Weitere Systeme, die Funktionen für eine Simulationsrechnung mit in die Visualisierungsumgebung integrieren, sind z.B. COVISE [26] und AMIRA [27, 28].

Systeme, die über eine Client-Server-Architektur hinausgehen, basieren häufig auf CORBA oder Java-Web-Applikationen. Zu nennen sind hier z.B. Ovid [29], VisWiz [30] und 3D-Streaming-Systeme [31]. In diesen Tools wird für den Transport der Daten ein vorhandenes Kommunikationsprotokoll wie z.B. HTTP oder CORBA benutzt.

Zwei wichtige Punkte, die eine Anbindung an ein paralleles Simulationsprogramm vereinfachen, sind die Unterstützung bei verteilten Daten und deren Reduktion, bevor diese zur Visualisierung übertragen werden. Nur in CUMULVS gibt es die Möglichkeit, eine Verteilung der Daten innerhalb des parallelen Simulationsprogramms zu spezifizieren. CUMULVS arbeitet mit der Kommunikationsbibliothek PVM und nutzt diese sowohl für die Kommunikation zwischen Visualisierung und Simulation als auch innerhalb des Simulationsprogramms.

Das Problem der Reduktion von großen Datensätze auf eine für die Visualisierung beherrschbare Größe wird in den oben aufgeführten Tools nicht direkt angegangen. Nur bei den integrierten Systemen ist dieses Problem implizit gelöst, da alle Module der Applikation auf die gemeinsamen Daten zugreifen können. Andererseits müssen dabei Simulations- und Visualisierungs-Funktionen auf einen gemeinsamen Speicher zugreifen. Generell wird erwartet, daß die Datenreduktion vom Programmierer der Simulation übernommen wird oder das die zu visualisierenden Daten schon in reduzierter Form vorliegen.

Gerade im Scientific Computing haben Simulationsprogramme eine lange Historie und es wurde sehr viel Arbeit in die Entwicklung der Programme investiert. Daher sollte eine Anbindung einer Steering-Komponente immer so erfolgen, daß möglichst wenige Änderungen am Quell-Code der Simulation durchgeführt werden müssen. Dieser Grund spricht z.B. gegen Analyse-Umgebungen, bei denen das Simulationsprogramm aus den Knoten eines Verarbeitungsnetzwerkes neu konstruiert werden muß.

Im folgenden Abschnitt wird eine Übersicht über die Konzepte der in dieser Arbeit für die Implementierung einer Computational Steering-Funktionalität verwendeten Bibliothek VISIT gegeben.

2.7.1 Visualization Interface Toolkit (VISIT)

Das Visualization Interface Toolkit VISIT [2] ist im Rahmen des DFN-Projekts "Gigabit Testbed West" [32] für die Kopplung von Visualisierungs- und Simulationsprogrammen entwickelt worden. Bei dem Entwurf wurde besonders auf eine einfache Handhabung von VISIT und eine minimale Beeinträchtigung der Performance des Simulationsprogramms geachtet. Weiterhin wurde darauf geachtet, daß im Zusammenspiel von Visualisierung und Simulation das Simulationsprogramm bevorzugt wird, so daß durch Fehler und Performance-Engpässe im Netzwerk auf der Visualisierungsseite die Rechnung nicht gestört wird. VISIT ist als Client-Server-Architektur aufgebaut, wobei aber entgegen anderen Implementierungen der VISIT-Client in das Simulationsprogramm integriert und der VISIT-Server auf der Visualisierungsseite etabliert ist (siehe Abb. 2.8).

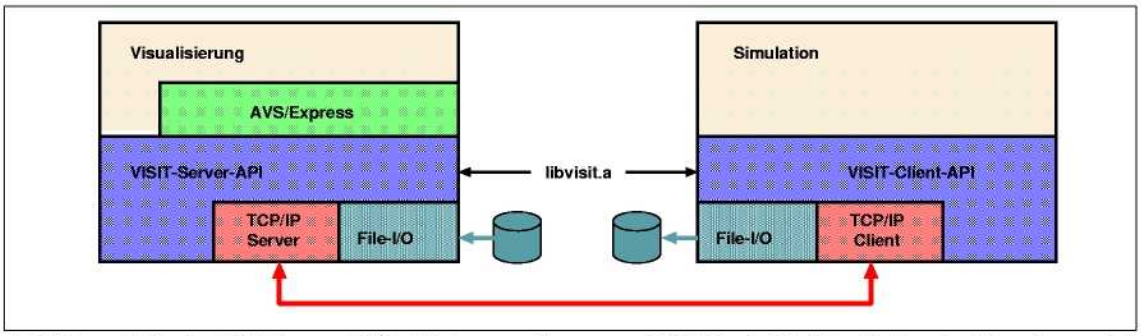


Abbildung 2.8: Architektur und Verbindungsschema von VISIT: Auf beiden Seiten der Kopplung wird für die Verbindung dieselbe VISIT-Bibliothek benutzt. Diese bietet neben der Verbindung über TCP/IP-Sockets auch die Speicherung in Dateien bzw. das Lesen aus Dateien an. So können z.B. Daten mitgeschnitten und später off-line erneut betrachtet werden.

Die Rolle des Servers, der im wesentlichen auf Anforderungen des Clients wartet und diese dann erfüllt, ist in VISIT auf der Visualisierungsseite etabliert worden. In den Architekturen bisher bekannter Steering-Tools agiert die Simulation als eine Art Datenserver, der Daten für die Abholung durch die Visualisierung bereithält. Dieses Verfahren erzwingt, daß die Simulation auf Anforderungen der Visualisierung warten und reagieren muß. Eine Störung der Simulationsrechnung z.B. durch Interrupts ist damit unumgänglich. In VISIT wird diese Behinderung der Simulation durch die Ankopplung vermieden, in dem die Visualisierung die Rolle des Servers übernimmt und auf die Daten der Simulation wartet. Dies hat den Vorteil, daß das Simulationsprogramm alle Steering-Aktionen (z.B. Übertragen von Daten) initiiert und damit auch die Kontrolle über den Startzeitpunkt erhält.

Als Schnittstelle steht auf der Seite des Simulationsprogramms ein C- oder FORTRAN-API zur Verfügung. Auf der Visualisierungsseite stehen neben dem C-Server-API auch entsprechende Module des Visualisierungssystems AVS/Express bereit, so daß auf dieser Seite keine explizite Programmierung nötig ist. AVS/Express arbeitet nach dem Datenfluß-Modell. Dazu können Funktions-Module mit Hilfe von Ein- und Ausgabe-Ports miteinander zu einem Netzwerk zusammengefügt werden. In diesem Netzwerk gibt es typischerweise einen Datenfluß von den Eingabe-Modulen über gewisse Verarbeitungsmodule (z.B. Generierung von Iso-Oberflächen) bis hin zu den Anzeige-Modulen für die Anzeige der Daten. Mit Hilfe eines graphischen Editors kann das Netzwerk ohne zusätzliche Programmierung interaktiv zu einer sehr leistungsfähigen Visualisierungsanwendung zusammengestellt werden.

Für die Datenübertragung können u.a. TCP/IP-Sockets benutzt werden, die auf sehr vielen Plattformen implementiert sind und damit eine Kopplung verschiedenster Rechner erlauben. Auf der Client-Seite sind die Funktionsaufrufe an die von MPI angelehnt. So können verschiedene Datentypen (Integer, Float, Strings, ...) übertragen und mit numerischen IDs versehen werden. Durch diese IDs können auf der Server-Seite verschiedene Datenströme zwischen Simulation und Visua-

lisierung identifiziert werden. Die Daten können entweder aus mehrdimensionalen Feldern oder aus einfachen Strukturen bestehen, so daß für den Transport von der Simulation zur Visualisierung die Felder direkt übertragen werden können. Für den Transport in umgekehrter Richtung können z.B. Steuerungsinformationen in Strukturen zusammengefaßt und damit einfacher übertragen werden. Die bei verschiedenen Hardware-Plattformen nötige Datenkonvertierung wird von VISIT automatisch durchgeführt. Da die meisten modernen Plattformen das IEEE-Gleitkommaformat unterstützen, ist die Konvertierung nur bei unterschiedlicher Byte-Order (Big- bzw. Little-Endian) oder unterschiedlicher Genauigkeit der Zahlendarstellung notwendig.

Für den Aufbau einer Verbindung mit TCP/IP muß der Client die Internet-Adresse des Servers und die dort von VISIT benutzte Port-Nummer kennen. Der Server überwacht diesen Port und kann bei einer Anfrage des Clients über diesen Port eine Verbindung zwischen Client und Server herstellen. Zur einfachen Verwaltung der für den Verbindungsaufbau wichtigen Internet-Adresse und Port-Nummer steht bei VISIT ein Name-Server (*service announcement protocol*, SEAP) zur Verfügung, der auf einer festgelegten Workstation läuft. Für den Verbindungsaufbau hinterlegt die Visualisierung ihre Adresse und Port-Nummer unter einem *Service*-Namen auf dem SEAP-Server. Die Simulation selbst kennt nur den *Service*-Namen und muß die für den Verbindungsaufbau wichtige Informationen mit Hilfe dieses Namens vom SEAP-Server abholen. Dieser auf den ersten Blick etwas umständliche Verbindungsaufbau über ein dritte Instanz hat den Vorteil, daß weder Simulation noch Visualisierung die Adresse des anderen Partners kennen müssen. Damit ist es z.B. möglich, sich von verschiedenen Workstations aus mit dem Simulationsprogramm zu verbinden, ohne daß dieses direkt vom Wechsel der Workstation informiert werden muß. VISIT sollte so in die Simulation eingebunden sein, daß in gewissen Abständen eine Anfrage an den SEAP-Server gestartet wird. Ist dort eine Visualisierung registriert, kann die Verbindung dorthin aufgebaut werden. Dadurch ist es aber auch möglich, sich nur zeitweise mit der Simulation zu verbinden. In den restlichen Zeiten kann die Simulation ohne Verbindung zu einer Visualisierung weiterrechnen (siehe Abb. 2.9).

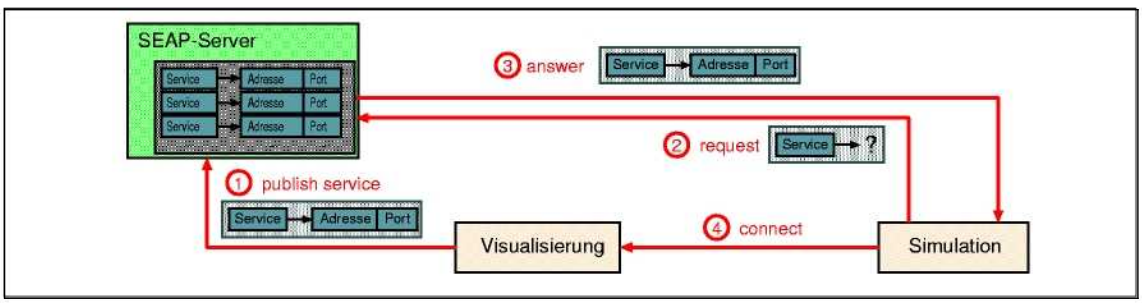


Abbildung 2.9: Verbindungsaufbau bei VISIT: In vier Schritten wird bei VISIT eine Verbindung aufgebaut, ohne daß die beiden Partner ihre Adresse kennen. Dabei ist der SEAP-Server eine unabhängige Komponente, die unter einer sowohl der Simulation als auch der Visualisierung bekannten Adresse zu erreichen ist. Die Visualisierung hinterlegt unter einem eindeutigen Service-Namen ihre Adresse und Port-Nummer im SEAP-Server. Die Simulation kann diese dort abholen und damit die Verbindung zur Visualisierung aufbauen.

In der neuesten Version von VISIT ist die Anzahl der Clients, die sich mit einem VISIT-Server verbinden können, nicht mehr auf einen Client begrenzt. Vielmehr können sich kollektiv mehrere Clients gleichzeitig mit einem VISIT-Server verbinden, was dazu genutzt werden kann, ein paralleles Simulationsprogramm auch über mehrere Kommunikationpfade an die Visualisierung anzukoppeln. In dieser Arbeit wird untersucht, in welchen Fällen dann mit einem Performance-Gewinn zu rechnen ist.

Kapitel 3

Strategien zur Datenkomprimierung und Datenreduktion

Bereits in vorigen Kapitel wurde dargelegt, daß eine effektive Datenreduktion und Komprimierung für die Anwendung des Computational Steerings bei vielen Simulationsrechnungen wichtig sind. Die Verfahren zur Komprimierung von Daten können in zwei Klassen aufgeteilt werden. Die erste Klasse umfaßt diejenigen Verfahren, die *verlustfrei* arbeiten und dadurch nach der Komprimierung auch eine vollständige Rekonstruktion der Originaldaten ermöglichen. Im Gegensatz dazu sind die Verfahren der zweiten Klasse *verlustbehaftet.* Diese trennen die Daten in „wichtige“ und “unwichtige“ Informationen und komprimieren nur die wichtigen. Dies führt im allgemeinen zu einer höheren Komprimierungsrate, die aber durch den Nachteil erkauft wird, daß die Daten nach der Rekonstruktion nur eine Näherung an den Originaldatensatz darstellen.

Im Umfeld des Computational Steering müssen die Komprimierungsverfahren unter anderen Gesichtspunkten betrachtet werden. So spielt hier neben der erzielbaren Komprimierungsrate auch die Komplexität der Verfahren und die benötigte Zeit für Komprimierung eine große Rolle. Bei einer angekoppelten Visualisierung soll die Komprimierung „on-the-fly“ geschehen und das Simulationsprogramm nicht wesentlich beeinträchtigen.

Im vorangegangen Kapitel wurde schon der Aspekt angesprochen, daß bei großen Datenmengen nicht nur der Transport der Daten vom Simulationsrechner zum Visualisierungsrechner einen Engpaß bildet. Vielmehr bringen die großen Datenmengen auch die Visualisierung selbst an die Grenze des Möglichen. Das heißt, daß die verlustfreien Verfahren der ersten Klasse hier nicht alleinige Abhilfe bringen. Sie reduzieren zwar die benötigte Bandbreite für den Transport der Daten, aber nach der Rekonstruktion haben die Daten wieder den für die Visualisierung zu großen Umfang der Originaldaten. Eine mögliche Lösung für dieses Problem ist die Kombination mit einem verlustbehafteten Verfahren, das nur die wichtigen Daten verwendet und so die zu übertragenden und darzustellenden Datenmengen drastisch reduziert.

Gerade beim Computational Steering stellt der Einsatz von verlustbehafteten Verfahren kein Problem dar. Durch die Online-Visualisierung sollen die Daten nur in einer solchen Genauigkeit angezeigt werden, daß sie als Entscheidungsbasis für Parameteränderungen oder sogar für den Abbruch der Simulationsrechnung dienen können. Für die High-End-Visualisierung im Post-Processing sollten zusätzlich ausgewählte Datensätze auf dem Simulationsrechner abgespeichert werden.

Durch den im World-Wide-Web immer höher werdenden Übertragungsbedarf von Bilddaten haben sich in der Klasse der verlustbehafteten Verfahren diejenigen durchgesetzt, die eine progressive Übertragung der Bilder erlauben. Dabei werden die Bilder vor der Übertragung in verschiedene Auflösungsstufen aufgeteilt und nacheinander übertragen. Der Betrachter sieht dabei zuerst ein

grobes Bild, das nach und nach weiter verfeinert wird. Bei vielen Bildformaten wird dabei das Bild nicht neu gerastert, sondern vom Ortsraum in den Frequenzraum übertragen. Die grobe Auflösung entspricht dabei den niedrigen Frequenzbändern, die durch Hinzufügen von weiteren Frequenzbändern zum hochaufgelösten Bild vervollständigt wird. Bekannt ist hier die JPEG-Kodierung, welche die gefensterte Fouriertransformation benutzt und derzeit im neuen Standard JPEG2000 [33] die Wavelet-Transformation einsetzt.

Eine wichtige Rolle bei den verlustbehafteten Verfahren spielt die Quantisierung. Sie sorgt dafür, daß die bei Simulationsrechnungen typischerweise als Gleitkommazahlen vorliegenden Datenwerte auf einen ganzzahligen Bereich abgebildet werden. Die Quantisierung erreicht dadurch schon eine beachtliche Komprimierungsrate. Für die Speicherung werden dadurch statt der 4- bzw. 8-Byte für eine Gleitkommazahl nur noch 1-2 Byte für den Ganzzahlwert benötigt.

Die folgenden Abschnitte beschreiben die einzelnen Teilschritte der in dieser Arbeit gewählten Komprimierungsverfahren. Es handelt sich dabei um die Wavelet-Transformation mit anschließendem Abschneiden von Koeffizienten, die Quantisierung, die Kodierung der ganzzahligen Werte und deren verlustfreie Komprimierung. Bei den Verfahren wird jeweils deren Eignung für den Einsatz im Computational Steering untersucht und insbesondere bei der Kodierung der quantisierten Daten ein Modell für den Speicherplatzbedarf der verschiedenen Kodierungsmöglichkeiten entwickelt.

3.1 Die Wavelet-Transformation

Für die Komprimierung von Bildern gibt es eine Anzahl verschiedener Algorithmen, die sich durch die erreichbare Komprimierungrate und den Berechnungsaufwand unterscheiden. Im alten JPEG-Standard, der vielfach für die Kodierung und Komprimierung von Bildern zur Übertragung im Internet genutzt wird, findet z.B. die gefensterte Fouriertransformation [34] ihre Anwendung (STFT, *Short Time Fourier Transform*). Hier wird für die Beschreibung des Bildinhaltes vom Ortsraum in den Frequenzraum gewechselt. Dies geschieht mit Hilfe der Fouriertransformation:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i t\omega}\, dt = \int_{-\infty}^{\infty} ft(\sin(2\pi t\omega + \frac{\pi}{2}) - i\sin(2\pi t\omega))dt \tag{3.1}$$

Der Bildinhalt stellt sich nach der Transformation als ein Frequenzspektrum dar, das sich aus Anteilen von Sinus-Kurven verschiedener Frequenz zusammensetzen läßt. Das ursprüngliche Bild ergibt sich durch die Überlagerung der einzelnen Sinus-Kurven. Nachteil dieser Methode ist der Mangel an Informationen über den Ort. Eine Änderung (z.B. Einfügen einer Kante) an einer Stelle im Bild verändert das komplette Spektrum der Fouriertransformierten. Dieser Nachteil wird in der Methode der gefensterten Fouriertransformierten vermieden, indem man eine Fensterfunktion $g(\tau)$ einfügt, welche die Transformation auf einen bestimmten Bereich im Ortsraum einschränkt:

$$F_g(\omega, \tau) = \int_{-\infty}^{\infty} f(t) g(t-\tau) e^{-2\pi i t\omega}\, dt \tag{3.2}$$

Als Fensterfunktion wird oft die Gaußfunktion genommen. Ihr Integral soll endlich und verschieden von Null sein. Dieses Verfahren liefert nun für verschiedene Punkte im Ortsraum jeweils ein eigenes Frequenzspektrum, welches das Bild an dieser Stelle (Umgebung) beschreibt. Auch dieses Verfahren weist dann Nachteile auf, wenn Funktionen mit lokalen Besonderheiten beschrieben

werden sollen. Die Größe des Ausschnitts im Ortsraum ist immer gleich. Sowohl bei niedrigen als auch bei hohen Frequenzen wird immer die gleiche Fenstergröße gewählt. Es wäre sinnvoller, bei niedrigen Frequenzen einen großen Ausschnitt und bei hohen Frequenzen einen kleinen Ausschnitt zu wählen. Dazu müßte die Frequenz auch als Parameter in die Fensterfunktion eingehen und dort zur Skalierung genutzt werden.

Diese von der Frequenz abhängigen Fensterfunktionen finden in der Wavelet-Transformation [35, 36, 37] ihre Anwendung. Die Wavelet-Transformation $W_{\tau,s}(f)$ einer Funktion f ist folgendermaßen definiert:

$$W_{\tau,s}(f) =< f, \psi_{\tau,s} >= \int_{-\infty}^{\infty} f(t)\psi_{\tau,s}(t)\,dt \qquad \tau, s \in \mathbb{R}, s \neq 0 \tag{3.3}$$

Bei dieser Integraltransformation (Faltung) wird anschaulich gesehen die Funktion f mit einer Fensterfunktion multipliziert und das Ergebnis aufsummiert. Die Fensterfunktion $\psi_{\tau,s}(t)$ wird als Wavelet bezeichnet und besitzt zusätzlich zum Translation-Parameter τ (Dilatation) einen Skalierungs-Parameter s. Man kann diese Faltung mit der Berechnung der Korrelation beider Funktionen vergleichen. Das Ergebnis der Korrelationsrechnung ist umso größer, je genauer die beiden Funktionen in ihrer Form übereinstimmen. Entsprechendes gilt auch für die Wavelet-Transformation. $W_{\tau,s}(f)$ beschreibt also, wie genau die Funktion $f(t)$ an der Stelle τ so geformt ist wie eine um s skalierte Wavelet-Funktion.

Anschaulich gesehen wird bei der Wavelet-Transformation die Funktion $f(t)$ mit einer Menge von Wavelet-Funktionen $\psi_{\tau,s}(t)$ verglichen. Die als Ergebnis entstehenden Koeffizienten beschreiben nun die Funktion im Frequenzraum. Die Wavelet-Funktionen werden dabei aus einem Prototyp abgeleitet, der auch als *Mother*-Wavelet bezeichnet wird. Mit Hilfe der beiden Parameter τ, s wird aus dem *Mother*-Wavelet eine Funktionsfamilie definiert:

$$\{\psi_{\tau,s}\} := \frac{1}{\sqrt{|s|}}\psi\left(\frac{t-\tau}{s}\right) \qquad \tau, s \in \mathbb{R}, s \neq 0 \tag{3.4}$$

Gerade durch die Skalierung der Wavelet-Funktion wird bei der Wahl der Fenstergröße (Fensterung) Rücksicht auf den gewählten Frequenzbereich genommen: Für hohe Frequenzen ist die Fensterfunktion schmal, für niedrige ist sie breiter. Daraus folgt, daß man für niedrige Frequenzen eine gute Frequenz- und eine schlechte Ortsauflösung und bei hohen Frequenzen eine schlechte Frequenz- und eine gute Ortsauflösung erhält. Diese Charakteristik entspricht aber in der Praxis auch der Charakteristik vieler Signale, bei denen tiefe Frequenzen über einen längeren Zeitraum und hohe Frequenzen nur in einem kurzen Zeitraum auftreten [38]. Abbildung 3.1 auf der nächsten Seite zeigt diesen wesentlichen Unterschied zur Fouriertransformation und zur gefensterten Fouriertransformation.

Bisher gibt es noch keine Festlegung, wie das *Mother*-Wavelet aussehen soll. Es wird nur eine Forderung an die Funktionen gestellt: Die Wavelet-Funktion $\psi \in L_2(\mathbb{R})$ muß folgende Konvergenzbedingung erfüllen:

$$0 < C_\psi := \int_{\mathbb{R}} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi < \infty \tag{3.5}$$

$\hat{\psi}$ ist dabei die Fourier-Transformierte von ψ. C_ψ wird als Zuverlässigkeitskonstante bezeichnet. Die Menge $L_2(\mathbb{R})$ kennzeichnet den Raum der meßbaren quadratintegrierbaren eindimensionalen Funktionen endlicher Energie. Dabei muß die L_2-Norm endlich sein:

$$\|f(t)\|_2 = \int_{-\infty}^{+\infty} |f(t)|^2 dt < \infty \tag{3.6}$$

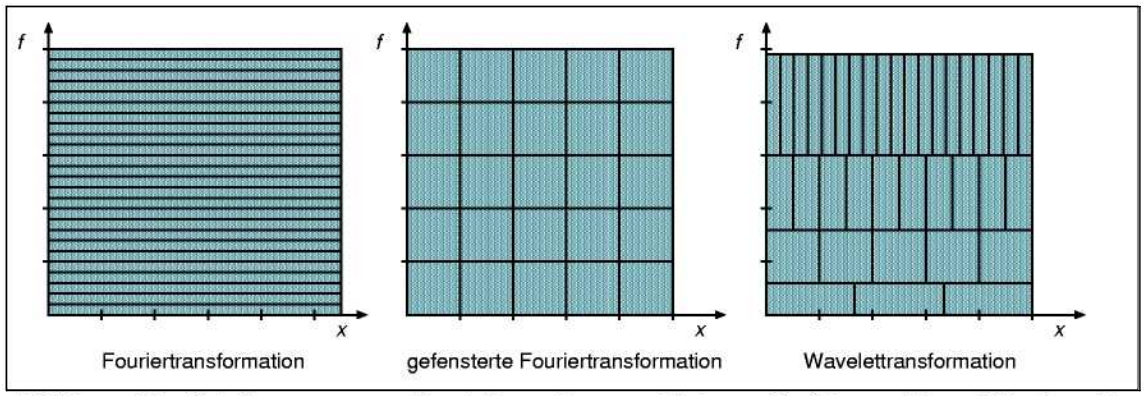


Abbildung 3.1: Ortsfrequenzraum-Darstellung der verschiedenen Verfahren: Man erhält einen Zusammenhang zwischen der Orts- und Frequenzauflösung, wenn man die Transformationen in der Ortsfrequenzebene aufträgt. Jeder Kasten im Diagramm entspricht einem Koeffizienten der Transformation und wird als Heisenbergkästchen bezeichnet [38]. Die Ausdehnung eines Kästchens in x-Richtung beschreibt dessen Ortsauflösung, die Ausdehnung in f-Richtung die Frequenzauflösung. So besitzt die Fouriertransformation (links) eine gute Frequenzauflösung. Da bei der Transformation der Ort nicht festgelegt ist, besitzen die Kästchen in x-Richtung eine unendliche Ausdehnung. In der Fensterfunktion der STFT (Mitte) geht die Frequenz nicht mit ein, so daß sich hier gleichartige und von der Frequenz unabhängige Kästchen ergeben. Nur bei der Wavelettransformation (rechts) ist die Fensterbreite von der Frequenz abhängig.

Anschaulich bedeutet dies, daß die Funktion betragsmäßig schnell genug abfallen muß. Im allgemeinen muß zusätzlich zur Wavelet-Funktion auch die zu approximierende Funktion $f(t)$ aus dieser Menge $L_2(\mathbb{R})$ sein.

Die Konvergenzbedingung bedeutet, daß zum einen das *Mother*-Wavelet keinen Gleichanteil besitzt und zum anderen kompakt ist. Die erste Bedingung bedeutet, daß $\int_{-\infty}^{\infty} \psi(t)dt = 0$ gilt. Die zweite Bedingung besagt, daß die Funktion nur in einem sehr begrenzten Bereich um den Nullpunkt Werte verschieden von Null aufweist. Beide Eigenschaften zusammen ergeben das Bild einer kleinen, um den Nullpunkt oszillierende Welle, die dieser ihren Namen *Wavelet* gibt (Wavelet = *kleine Welle*). Abbildung 3.2 zeigt einige Beispiele für Wavelet-Funktionen.

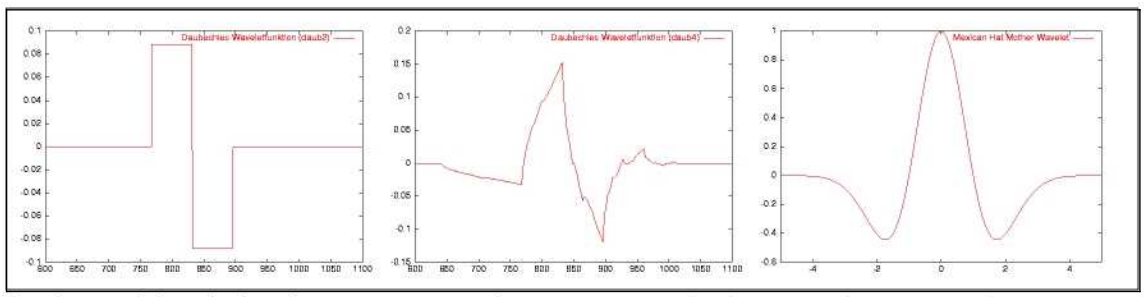


Abbildung 3.2: Beispiele für Wavelet-Funktionen: Haar-Wavelet (links), Daub4 (Mitte) und Mexican Hat (rechts) sind Beispiele für Funktionen, die der oben beschriebenen Konvergenzbedingung genügen. Allen gemeinsam ist, daß sie kompakt sind, also nur für einen bestimmten Bereich des Definitionsgebiets Werte verschieden von Null aufweisen.

Durch die Konvergenzbedingung (3.5) ist sichergestellt, daß auch die Umkehrung der Transformation definiert ist:

$$f(t) = \frac{1}{C_\psi} \int_{\mathbb{R}^2} W_{\tau,s}(f)\psi_{\tau,s}(t)\,\frac{d\tau ds}{s^2} \tag{3.7}$$

Damit ist es möglich, aus der Wavelet-Transformierten wieder die Originalfunktion zu berechnen.

In der Berechnung der Wavelet-Transformation wird ein Skalarprodukt von Funktionen verwen-

det, das wie folgt definiert ist:

$$< f(t), g(t) > = \int_{-\infty}^{\infty} g(t)\, f(t) dt \tag{3.8}$$

Daraus ergibt sich die Eigenschaft der *Linearität*. Durch das verwendete Skalarprodukt ist die Wavelet-Transformation eine lineare Transformation. Für die Wavelet-Zerlegung sind vor allem solche Wavelet-Funktionen interessant, die orthogonal zueinander sind. Die Funktionen sind damit auch linear unabhängig. Werden bei der Wavelet-Transformation einer Funktion orthogonale Wavelet-Funktionen verwendet, ist dadurch sichergestellt, daß die Information, die durch einen Koeffizienten dargestellt wird, nicht an anderer Stelle nochmal kodiert wird. Es gibt also bei den Koeffizienten der Wavelet-Funktionen keine Redundanz. Sind die Wavelet-Funktionen zusätzlich orthonormal, bildet die Funktionsfamilie $\{\psi_{\tau,s}\}$ eine Orthonormalbasis des $L_2(\mathbb{R})$. Die Orthonormalität ist gegeben, wenn die L_2-Norm der Funktionen den Wert 1 besitzt.

Sehr wichtig für die Anwendung der Wavelet-Transformation ist die Eigenschaft der *Energieerhaltung*. Es gilt $\|f\|_2 = \|W(f)\|_2$. Durch den Faktor $\frac{1}{\sqrt{|s|}}$ in der Definition der Wavelet-Funktionsfamilie (Gleichung 3.4) wird die Energie sogar beim Wechsel zwischen verschiedenen Frequenzskalen erhalten.

3.1.1 Die diskrete Wavelet-Transformation

Die im vorausgehenden Abschnitt vorgestellte Wavelet-Transformation arbeitet auf kontinuierlichen Funktionen und benutzt dabei für die Verschiebung und die Skalierung der Wavelet-Funktion $\psi_{\tau,s}(t)$ kontinuierliche Parameter τ und s. Dies führt zu einer unendlichen Menge von Koeffizienten. Üblicherweise liegen die Daten von Simulationsrechnungen nicht als kontinuierliche Funktionen vor. Vielmehr sind die Daten diskretisiert und stehen in Form von Datenvektoren zur Verfügung. Eine Anwendung der kontinuierlichen Wavelet-Transformation würde bei solchen Daten zu einer Überabtastung führen. Daher wird für die Transformation von diskreten Daten auch eine diskrete Darstellung der Wavelet-Funktionen gesucht. Für die Bildung einer diskreten Wavelet-Funktionsfamilie wird üblicherweise eine Skalierung mit Faktoren der Form 2^j zu Grunde gelegt:

$$\psi_{j,k}(t) := \psi(2^j t - k)\ , j = -1..\infty \tag{3.9}$$

Die Parameter j, k für Dilatation und Skalierung nehmen jetzt nur noch diskrete Werte an. Wird diese Form auf orthonormale Wavelets angewendet, ergibt die Funktionsfamilie wiederum eine Orthonormalbasis.

Die Wavelet-Transformation kann wie bisher berechnet werden:

$$c_{j,k} = \int_{-\infty}^{\infty} f(t)\psi_{j,k}(t)\, dt \tag{3.10}$$

Da die Koeffizienten der Wavelet-Funktionen nun als diskrete Werte vorliegen, erfolgt die Berechnung der Originalfunktion aus den Koeffizienten der Wavelet-Transformation über Summen:

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_{j,k}\psi_{j,k}(t) \tag{3.11}$$

Ein Beispiel für eine Funktionsfamilie, die oft bei diskreten Daten (Bildern) angewendet wird, ist das Haar-Wavelet, das auch unter dem Namen Daub2-Wavelet (siehe Abb. 3.2) bekannt ist. Hier

ist das *Mother*-Wavelet folgendermaßen definiert:

$$\psi_H(t) = \begin{cases} 1 & : \quad 0 < t < 1/2 \\ -1 & : \quad 1/2 \leq t < 1 \\ 0 & : \quad sonst \end{cases} \tag{3.12}$$

Eine um 2^j skalierte und um k verschobene Wavelet-Funktion ist definiert durch:

$$\psi_{j,k}(t) := \psi(2^j t - k) = \begin{cases} 1 & : \quad k/2^j < t < (k+1)/2^j \\ -1 & : \quad 1/2^j \leq t < (k+1)/2^{j-1} \\ 0 & : \quad sonst \end{cases} \tag{3.13}$$

Die zu transformierende Funktion $f(x)$ wird nun als Kombination der Haar-Wavelets dargestellt. Der jeweilige Anteil des Wavelets an der Funktion wird durch die Koeffizienten $c_{i,k}$ bestimmt. Die Haar-Wavelets zeichnen sich dadurch aus, daß ihr Integral 0 ist. Besitzt die diskrete Funktion $f(x)$ einen Gleichanteil, ist dieser nicht mehr mit Hilfe des Haar-Wavelets darzustellen. Für die vollständige Approximation der Funktion ist daher eine zusätzliche Skalierungsfunktion $\Phi(x)$ definiert:

$$\phi(t) = \begin{cases} 1 & : \quad t \in [0,1] \\ 0 & : \quad sonst \end{cases} \tag{3.14}$$

Erst zusammen mit den Koeffizienten dieser auch als *Father*-Wavelet bezeichneten Funktion kann aus den Wavelet-Koeffizienten die Originalfunktion wieder hergestellt werden. $\psi_{j,k}(t)$ und $\phi(t)$ bilden eine Basis für die endlich quadratisch integrierbaren Funktionen, die nur auf dem Intervall $[0,1]$ von Null verschieden sind.

3.1.2 Die schnelle Wavelet-Transformation (FWT)

Die Berechnung der Wavelet-Transformation für diskrete Daten ist aufwendig, da für jeden Koeffizienten ein Integral gelöst werden muß. Mit der auf St. G. Mallat zurückgehenden schnellen Wavelet-Transformation (FWT, Fast Wavelet Transformation) steht ein Hilfsmittel zur Verfügung, das diese Aufgabe wesentlich effizienter löst [39].

Bei der schnellen Wavelet-Transformation werden zwei Filter eingesetzt. Ein Hochpaßfilter g und ein Tiefpaßfilter h werden auf den Datenvektor angewendet. Die aus der Filterung mit dem Hochpaßfilter entstehenden Daten enthalten nur die hochfrequenten Anteile. Der Tiefpaßfilter h eliminiert genau diese Anteile hoher Frequenz, so daß als Ergebnis eine grobere Darstellung der Originaldaten entsteht. Im wesentlichen stimmen die Ergebnisse dieser beiden Filter mit den Ergebnissen der Wavelet-Transformation überein. Die Koeffizienten der Skalierungsfunktionen (*Father*-Wavelets) stellen die Daten in einer reduzierten Auflösung dar und entsprechen somit den Ergebnissen des Tiefpaßfilters. Die Koeffizienten der *Mother*-Wavelets entsprechen den hochfrequenten Anteilen und somit den Ergebnissen des Hochpaßfilters.

Im einfachsten Fall der schnellen Wavelet-Transformation (Haar-Wavelet) berechnet der Hochpaßfilter die Differenz und der Tiefpaßfilter die Durchschnittswerte zweier benachbarter Vektorelemente. Dabei werden die Filter nacheinander auf alle Paare von Vektorelementen angewendet. Als Resultat des Tiefpaßfilters erhält man einen Wert, der zunächst an der Stelle des ersten Paarelements abgespeichert wird. Nach dem Filtervorgang wird in dem sogenannten *Downsampling*-Schritt jeder zweite Wert im Ergebnisvektor entfernt, so daß man einen Vektor der halben Länge erhält. Das gleiche Verfahren wird auch für den Hochpaßfilter angewendet. Hier wird nur die Differenz der beiden Vektorelemente abgespeichert. Abbildung 3.3 auf der nächsten Seite verdeutlicht diesen Vorgang.

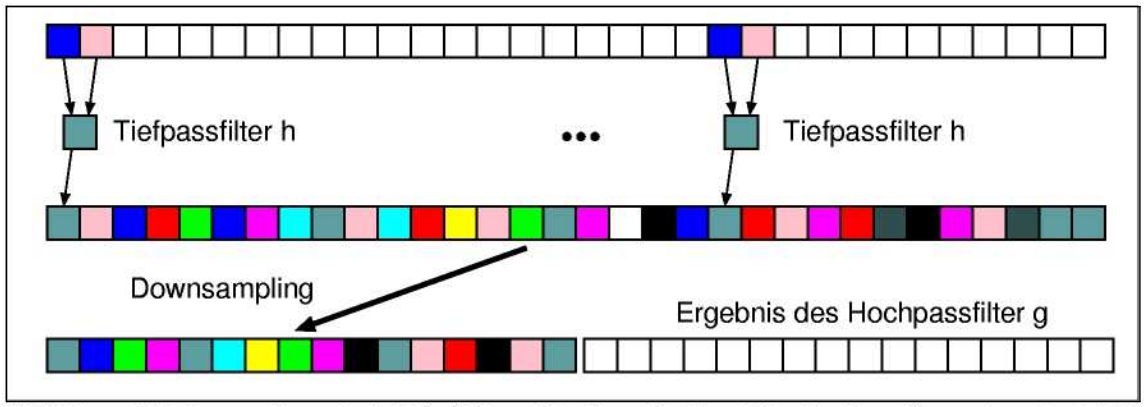


Abbildung 3.3: Anwendung des Tiefpaßfilters bei der schnellen Wavelet-Transformation: Der Tiefpaßfilter wird nacheinander auf alle Paare von Vektorelementen angewendet. Im anschließenden *Downsampling*-Schritt kann jedes zweite Element des Ergebnisvektors entfernt werden. Das gleiche Verfahren wird dann nochmals mit dem Hochpaßfilter durchgeführt. So entstehen dann zwei Teilvektoren, die jeweils halb so lang sind wie der Originalvektor. Bei der technischen Realisierung wird meistens nur jedes zweite Element des Datenvektors (als Beginn eines Paares) betrachtet und somit das *Downsampling* vermieden. Zusätzlich werden beide Filter gleichzeitig angewendet und die beiden Filterergebnisse in den zwei betrachteten Vektorelementen abgespeichert. Anschließend muß der Vektor nur noch umsortiert werden. Ein zusätzlicher Speicherplatz für Zwischenergebnisse ist somit nicht mehr nötig.

Durch das oben beschriebene *Downsampling* geht keine Information verloren, da aus dem Mittelwert (c) und der Differenz (d) zweier Elemente (a, b) diese selbst wieder hergestellt werden können:

$$(a,b) \rightarrow (c,d): \qquad c = g(a,b) = (a+b)/2 \qquad d = h(a,b) = a - b \tag{3.15}$$

$$(c,d) \rightarrow (a,b): \qquad a = g^*(c,d) = c + d/2 \qquad b = h^*(c,d), c - d/2 \tag{3.16}$$

Der Filtervorgang kann nun für den halb so langen Ergebnisvektor des Tiefpaßfilters erneut durchgeführt und zusammen mit dem *Downsampling* solange wiederholt werden, bis nur noch ein Wert als Ergebnis der letzten Anwendung des Tiefpaßfilter übrig bleibt. Dieser Wert repräsentiert dann den Mittelwert des gesamten Datenvektors und bildet mit der Reihe von Ergebnisvektoren des Hochpaßfilters das Resultat der schnellen Wavelet-Transformation. Abbildung 3.4 stellt dieses als Pyramidenalgorithmus bezeichnete Verfahren nochmals schematisch dar.

Für die Rekonstruktion der Originaldaten aus den transformierten Daten müssen inverse Filter (g^*, h^*) angewandt werden. Statt des *Downsampling* erfolgt nun ein *Upsampling*, das einen Vektor der doppelten Länge erzeugt. Das inverse Filterpaar kann aus Filtern für die Hintransformation abgeleitet werden (siehe Gleichung 3.16).

Bei den vom Hochpaßfilter erzeugten Teilvektoren stellt der kürzeste Vektor die Koeffizienten der Wavelet-Funktionen mit der niedrigsten Auflösung und der längste Vektor die Koeffizienten der Wavelet-Funktionen mit der höchsten Auflösung dar. Gerade dieses Anordnung erlaubt es, Wavelet-transformierte Daten progressiv zu übertragen. Es wird dabei zuerst das Ergebnis der letzten Anwendung des Tiefpaßfilters übertragen und anschließend die Teilvektoren des Hochpaßfilters. Bei der Rekonstruktion entstehen dann Vektoren höheren Auflösung.

Bei der schnellen Wavelet-Transformation sind die beiden Filter als Vektoren (g_k und h_k) aufgebaut, die im Falle des Haar-Wavelets aus jeweils zwei Komponenten bestehen. Der Filtervorgang besteht somit aus einer fortgesetzten Bildung des Skalarprodukts der Filtervektoren mit den entsprechenden Teilvektoren aus den Originaldaten. Generell können die Filter aus einer beliebigen Anzahl von Vektorelementen bestehen. Besteht der Filter z.B. aus L Vektorelementen, so müssen die aus den Originaldaten gebildeten Vektoren auch aus L Elementen bestehen. Mit der Anzahl der Elemente im Filtervektor steigt auch die Anzahl der Elemente im Datenvektor, die zur Bildung

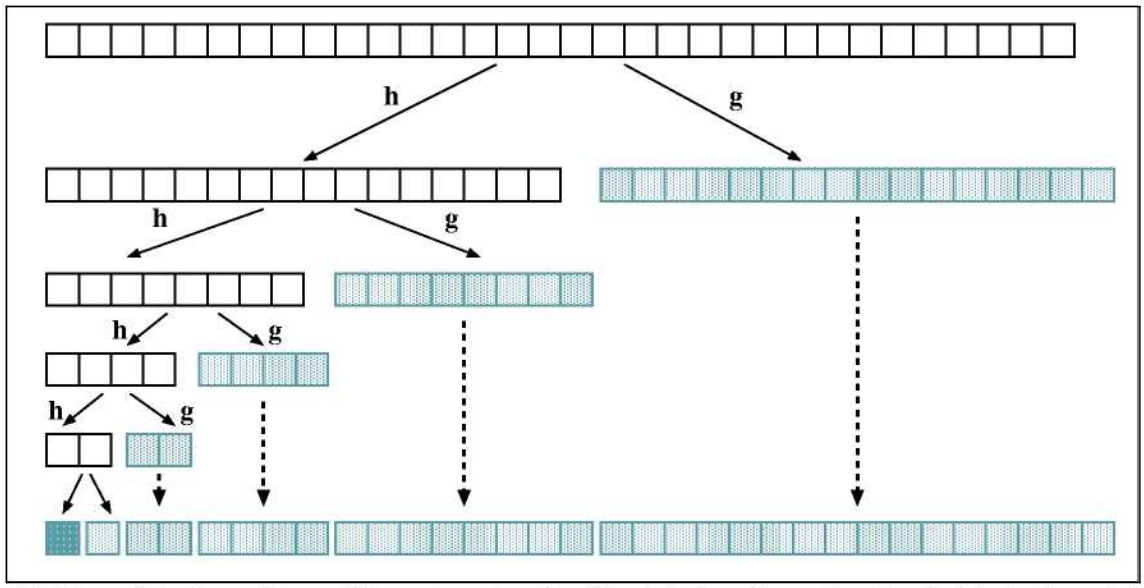


Abbildung 3.4: Pyramidenverfahren der schnellen Wavelet-Transformation: Der Originalvektor wird in mehreren Schritten mit Hilfe zweier Filter h (Tiefpaß) und g (Hochpaß) in Teilvektoren zerlegt. In jedem Schritt entstehen zwei gleich große Teilvektoren, von den der rechte (Ergebnis des Hochpaßfilters) direkt in das Ergebnis einfließt. Der linke Vektor (Ergebnis des Tiefpaßfilters) stellt den Datenvektor in einer geringeren Auflösung dar und wird im nächsten Schritt weiter verarbeitet. Wird der Transformationsschritt oft genug wiederholt, bleibt als Ergebnis des Tiefpaßfilters nur noch ein Wert übrig, der dann den Mittelwert aller Elemente des Datenvektors darstellt.

des Mittelwerts bzw. des hohen Frequenzanteils herangezogen werden. Anschaulich bedeutet dies, daß mit einer steigenden Anzahl von Elementen die Glättung des Signal zunimmt.

Wie bei den Wavelets besteht für die Wahl der beiden Filter g_k und h_k eine gewisse Freiheit. Eingeschränkt wird diese nur durch folgende Bedingungen, die für die eindeutige Umkehrung der Transformation nötig sind:

$$g_k = (-1)^k h_{L-1-k}, \; k = 0, ..., L-1 \tag{3.17}$$

$$\sum_k h_k = \sqrt{2} \tag{3.18}$$

$$\sum_k g_k = 0 \tag{3.19}$$

$$\sum_k h_k h_{k+2m} = \sigma_{0,m} \tag{3.20}$$

Mit der ersten Bedingung (3.17) werden die beiden Filter miteinander verknüpft: Die Vektorelemente von g stehen in dem Vektor h in umgekehrter Reihenfolge, wobei jedes zweite Element ein umgekehrtes Vorzeichen erhält. Die zweite und dritte Bedingung definieren, daß h einen Gleichanteil besitzt und g keinen Gleichanteil besitzt. Auffällig ist, daß der Gleichanteil des Tiefpasses nicht, wie bei der Bildung eines Mittelwerts üblich, gleich 1 ist (3.18). Begründet wird dies durch die Forderung, daß die Wavelet-Transformation energieerhaltend sein soll. Nur durch den Faktor $1/\sqrt{2}$ im Tiefpaßvektor des Haar-Filters ergibt sich bei dem halb so großen Ergebnisvektor die gleiche Energie wie beim Originalvektor (siehe auch Gleichung 3.6).

Wenn die oben beschriebenen Einschränkungen erfüllt sind, können die Filter für die Umkehrung der schnellen Wavelet-Transformation (g^* und g^*) aus g und h gebildet werden. Dazu gelten folgende Regeln:

$$g_k^* = g_{L-1-k}, \qquad h_k^* = h_{L-1-k} \tag{3.21}$$

Der Zusammenhang zwischen den Wavelet-Funktionen und dem Filterpaar g und h sowie die oben geforderten Eigenschaften der Filter ergeben sich aus der Multiresolution-Analysis. Die Filterkoeffizienten können aus dem *Mother*-Wavelet hergeleitet werden. Der üblichere Weg ist aber, zuerst die Filterkoeffizienten zu definieren und dann daraus die Wavelet-Funktion herzuleiten. Für die Berechnung der Wavelet-Transformation mit Hilfe der FWT ist es nicht nötig, daß das *Mother*-Wavelet bekannt ist.

Als erstes Beispiel für die Filter der schnellen Wavelet-Transformation sollen die Vektoren des Haar-Wavelets vorgestellt werden. Hier bildet der Tiefpaßfilter einen Mittelwert und der Hochpaßfilter die Differenz zweier benachbarten Vektorelemente. Dazu bestehen die Filter h und g jeweils aus zwei Elementen:

$$h = (1/\sqrt{2}, 1/\sqrt{2}) \qquad g = (1/\sqrt{2}, -1/\sqrt{2}) \tag{3.22}$$

Die Umkehrfilter für die inverse Transformation können mit den Gleichungen (3.21) folgendermaßen berechnet werden:

$$h^* = (1/\sqrt{2}, 1/\sqrt{2}) \qquad g^* = (-1/\sqrt{2}, 1/\sqrt{2}) \tag{3.23}$$

Abbildung 3.5 auf der nächsten Seite zeigt die Transformation der Mexican-Hat-Funktion als Beispiel für die Anwendung der eindimensionalen Wavelet-Transformation mit Hilfe des Haar-Filters.

Als zweites Beispiel soll ein schnelle Transformation mit Daubechies-Wavelets vorgestellt werden, deren Filtervektoren jeweils aus vier Vektorelementen bestehen (daub4):

$$h = (-\frac{1-\sqrt{3}}{4\sqrt{2}}, \frac{3-\sqrt{3}}{4\sqrt{2}}, -\frac{3+\sqrt{3}}{4\sqrt{2}}, \frac{1-\sqrt{3}}{4\sqrt{2}}) \tag{3.24}$$

$$g = (\frac{1+\sqrt{3}}{4\sqrt{2}}, \frac{3+\sqrt{3}}{4\sqrt{2}}, \frac{3-\sqrt{3}}{4\sqrt{2}}, \frac{1-\sqrt{3}}{4\sqrt{2}}) \tag{3.25}$$

In jedem Berechnungsschritt werden die Filtervektoren mit 4 aufeinanderfolgenden Vektorelementen multipliziert. Da also mindestens vier Vektorelemente für die Multiplikation mit den Filtern benötigt werden, muß der Pyramidenalgorithmus bei diesen Filtern früher enden. Als Ergebnis bleiben hier zwei Elemente übrig, die zusammen einen Mittelwert des Datenvektors bilden.

Ein weiteres Problem, das bei diesem Filter ersichtlich wird, ist die Größe des Vektors. Bisher wurde implizit davon ausgegangen, daß die zu transformierenden Datenvektoren aus 2^i Elementen bestehen. Nur dann ist eine wiederholte Anwendung der Wavelet-Transformation möglich. Ist dies nicht der Fall, kann z.B. beim Haar-Filter das letzte Element des Datenvektors nicht mehr betrachtet werden.

Eine weitere Ausnahme zeigt sich bei der Berechnung der Koeffizienten für die letzten Vektorelementen, da dort nicht genügend weitere Vektorelemente zur Verfügung stehen. Hier stehen zwei Möglichkeiten zur Auswahl: Anhängen des Vektoranfangs oder Anhängen von konstanten Werten (Null). Dieses Problem wird in Abschnitt 3.4.1 auf Seite 42 dieser Arbeit noch genauer angegangen.

Der Aufwand der FWT kann mit $O(n)$ abgeschätzt werden: In jedem Schritt des Pyramidenalgorithmus wird die Anzahl der zu bearbeitenden Daten halbiert. Für die Berechnung der Ergebnisse bei einer Vektorlänge N und einer Filterlänge K müssen im ersten Schritt $N * K$ Multiplikationen ausgeführt werden. Ergebnisse, die durch das *Downsampling* eliminiert werden, brauchen nicht berechnet zu werden. Daher geht jeder Filter mit $N/2$ Elemente in diese Rechnung ein. Für den

kompletten Pyramidenalgorithmus ergibt sich dann folgendes:

$$(N + N/2 + N/4 + ...)K = K \sum_{i=0}^{\log_2 N} \frac{N}{2^i}$$

$$< K \sum_{i=0}^{\infty} \frac{N}{2^i} = KN \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = KN \frac{1}{1-1/2} = 2KN \tag{3.26}$$

Insgesamt werden also weniger als $2 * K * N$ Multiplikationen benötigt, was für die FWT eine Komplexität von $O(N)$ ergibt. Im Gegensatz dazu liegt die Fast Fourier Transformation bei einer Komplexität von $O(N \log_2 N)$.

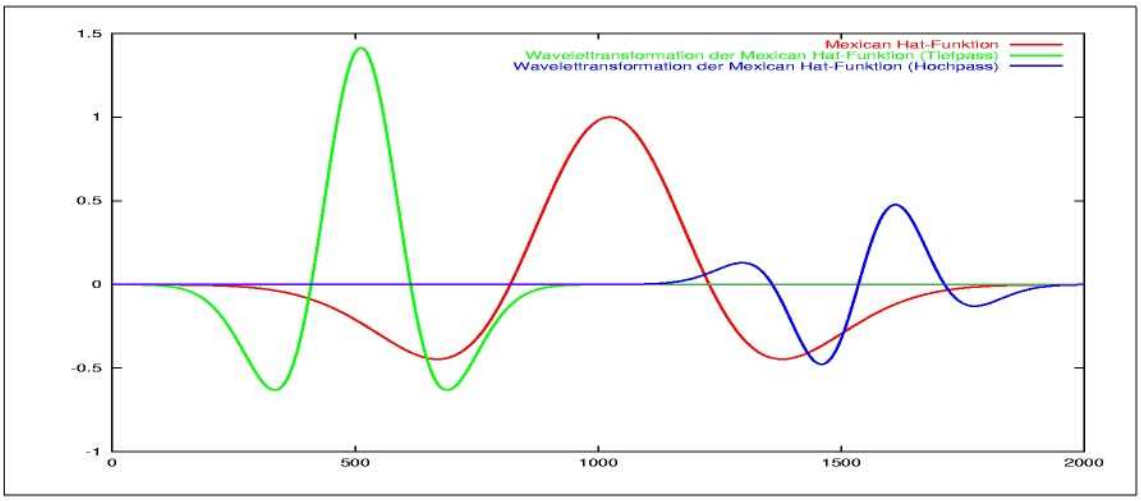


Abbildung 3.5: Beispiel für eine Transformation mit dem Haar-Filter (daub2): Die Ausgangsdaten für dieses Beispiel wurden von der Mexican Hat Funktion $x^2 \exp\left(-\frac{x^2}{2}\right)$ gewonnen. Auf diesem Datenvektor wurde der Pyramidenalgorithmus einmal angewendet. Im ersten Teil der Ergebnisfunktion sieht man wiederum die Mexican Hat-Funktion, aber in geringerer Auflösung. Im zweiten Teil der Ergebnisfunktion sieht man das Ergebnis des Hochpaßfilters, der die Differenz von aufeinanderfolgenden Vektorelementen berechnet. Zur besseren Darstellung wurden die Werte um den Faktor 100 vergrößert.

3.1.3 Mehrdimensionale schnelle Wavelet-Transformation

Mit der oben vorgestellten schnellen Wavelet-Transformation und dem dazugehörenden Pyramidenalgorithmus können eindimensionale Datenvektoren effizient transformiert werden. Für die Anwendung der FWT auf Bilder oder Volumendaten muß dieses Verfahren auf mehrere Dimensionen erweitert werden. Eine einfache Erweiterung ist immer dann möglich, wenn die bei der Transformation verwendete Skalierungsfunktion $\Phi(x, y)$ separierbar ist. Im zweidimensionalen Fall muß dann folgende Bedingung erfüllt sein:

$$\Phi(x, y) = \Phi_x(x) * \Phi_y(y) \tag{3.27}$$

Dann ist es möglich, die zweidimensionale Wavelet-Transformation auf eine Hintereinanderausführung zweier eindimensionaler Wavelet-Transformationen abzubilden. Diese mehrstufige Anwendung der Wavelet-Transformation soll im folgenden am Beispiel von zweidimensionalen Daten erläutert werden.

Ein häufiges genutztes Beispiel für zweidimensionale Daten sind digitalisierte Fotographien. Abbildung 3.6 auf der nächsten Seite veranschaulicht die Anwendung des Pyramidenalgorithmus auf solche Bilddaten. Bei der Benutzung einer separierbaren Skalierungsfunktion können die Hoch-

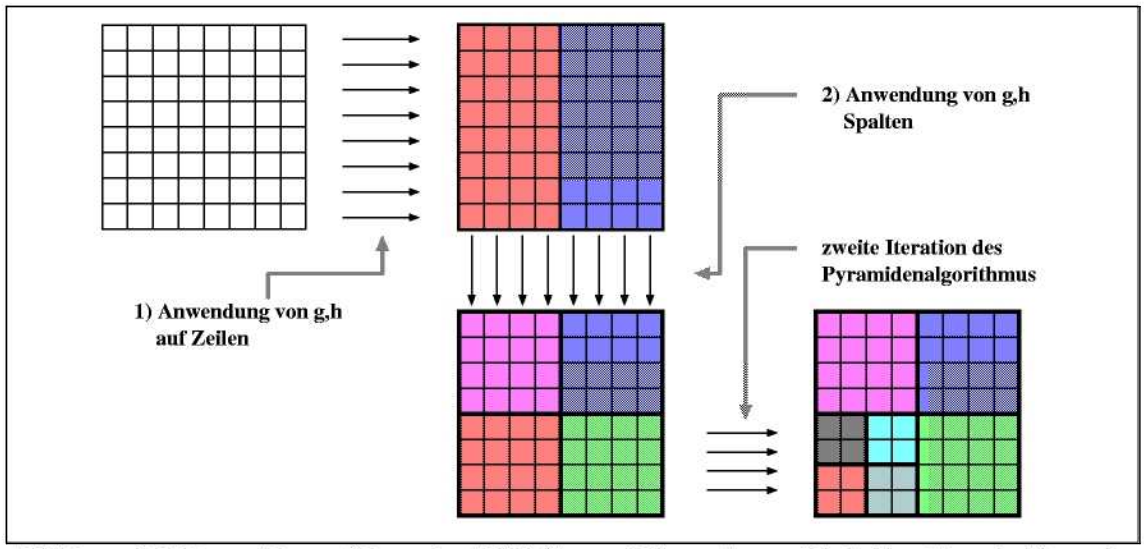


Abbildung 3.6: Pyramidenverfahren der FWT für zwei Dimensionen: Die beiden Filter (g, h) werden zuerst auf die Zeilen des Bildes angewandt. Danach ist das Bild in zwei Streifen unterteilt. Der linke Streifen enthält das in X-Richtung reduzierte Bild, der rechte Streifen beschreibt das Differenzbild, also die hohen Frequenzanteile in X-Richtung. Danach werden beide Filter auf alle Spalten des Bildes angewandt. Man erhält somit vier Teilbilder, von denen drei Differenzbilder und eins das in der Auflösung reduzierte Bild darstellen. In weiteren Iterationschritten wird dieses reduzierte Teilbild weiter zerlegt.

und Tiefpaßfilter nacheinander auf die einzelnen Dimensionen angewendet werden. Für Bilddaten heißt dies, daß zuerst die Zeilen und dann die Spalten des Bildes transformiert werden. Dabei werden bei der Transformation der Spalten die Ergebnisse beider vorher auf die Zeilen angewendeten Filter berücksichtigt (s. Abb. 3.7). In jedem Schritt entstehen somit 3 Teilbilder mit den Ergebnissen des Hochpaßfilters, die als Differenzbilder entsprechender Richtung (X, Y, diagonal) betrachtet werden können. Die Rücktransformation der Daten erfolgt nach gleichen Prinzip. Die inversen Filter werden jeweils nacheinander auf die Zeilen und Spalten angewendet.

Erweitert man das Schema auf drei Dimensionen, müssen die Filter nacheinander auf Datenvektoren in X-, Y- und Z-Richtung angewendet werden. In jedem Schritt des Pyramidenalgorithmus entstehen nun sieben Teilblöcke, die jeweils die Differenzen in X-, Y- und Z-Richtung, sowie in den Diagonalen in XY-, XZ-, YZ- und XYZ-Richtung enthalten. Abbildung 3.8 auf der nächsten Seite zeigt dazu ein Beispiel.

Abbildung 3.7: Beispiel: Anwendung der FWT auf ein Bild: Auf das linke Bild wurde er Pyramidenalgorithmus der FWT einmal angewendet. Das rechte Bild zeigt das Ergebnis der FWT. Die Koeffizienten der drei Differenzbilder sind zur besseren Darstellung um den Faktor 50 vergrößert worden. In den Differenzbilder treten besonders diejenigen Kanten hervor, die senkrecht zur Transformations-Richtung liegen.

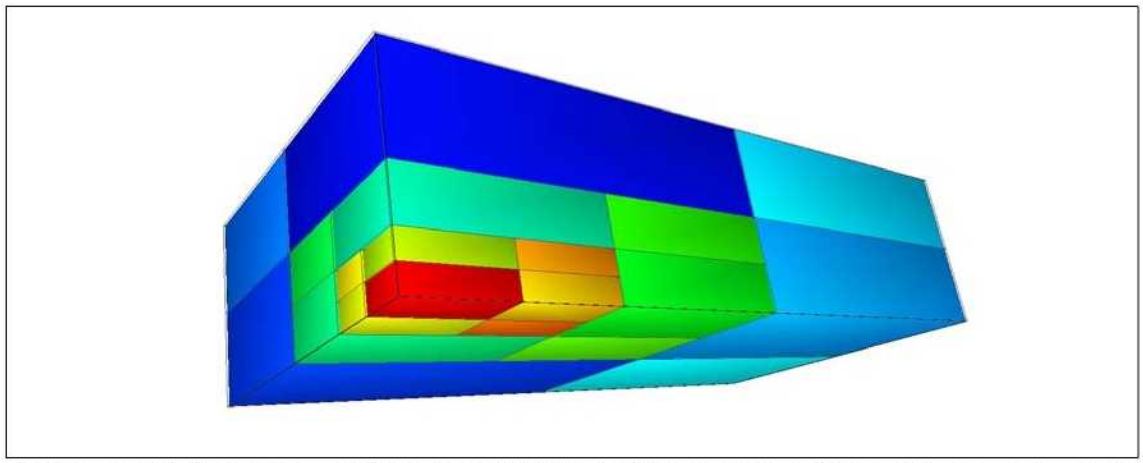


Abbildung 3.8: Pyramidenverfahren der FWT für drei Dimensionen: Dieses Schema zeigt, wie ein Volumen durch die FWT in Teilblöcke zerlegt wird. Hier wurden drei Iterationen des Pyramidenalgorithmus durchgeführt. Das nun in der Auflösung um den Faktor acht reduzierte Volumen befindet sich im vorderen roten Block.

Abbildung 3.9 zeigt die Anwendung des dreidimensionalen Pyramidenalgorithmus an einem Simulations-Datensatz.

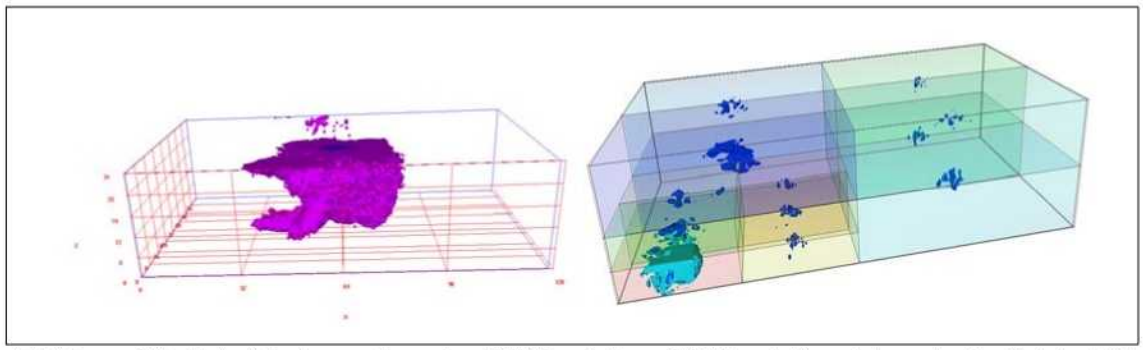


Abbildung 3.9: Beispiel: Anwendung der FWT auf einen 3D-Simulationsdatensatz: Im Originalbild (links) wird eine Schadstoffwolke gezeigt, die sich im Grundwasser ausbreitet und von einem Brunnen wieder aufgenommen werden soll. Das von dem Simulationsprogramm Partrace berechnete Konzentrationsfeld wurde in zwei Iterationen Wavelet-transformiert. Die dabei entstehenden Differenzvolumen (rechts) zeigen die hohen Frequenzanteile der Schadstoffwolke.

Die in diesem Beispiel zu erkennende starke Lokalität der Daten innerhalb des Simulationsgebiets ist nicht immer typisch für Simulationsdaten. Vielfach sind die Daten über das gesammte Simulationsgebiet verteilt, wie z.B. bei dem von Partrace verwendeten Geschwindigkeitsfeld oder dem bei der Erzeugung des Geschwindigkeitsfeldes verwendeten und statistisch erzeugten K-Feld (Durchlässigkeit des Bodens).

3.1.4 Progressive Übertragung von Wavelet-transformierten Daten

Die Wavelet-Transformation bietet in der bisher dargestellten Form noch keine Hilfe im Sinne der Datenkomprimierung. Durch die Transformation bleibt die Größe der Daten unverändert. Ein Volumen mit der Kantenlänge x,y,z wird auch wieder in ein Volumen mit der Größe x,y,z transformiert. Ein Vorteil liegt aber in der Aufteilung in verschiedene Frequenzbänder. Als Frequenzband wird das Ergebnis des Hochpaßfilters bezeichnet. Bei der eindimensionalen Transformation ist dies ein Vektor; bei den zwei- bzw. dreidimensionalen Transformationen besteht ein Frequenzband aus 3 bzw. 7 Datenblöcken. Nach der Transformation stehen damit die Daten in verschiedenen Auflösungsstufen zur Verfügung. Das Resultat der letzten Anwendung des Tiefpaßfilter kann

direkt angezeigt werden. Durch die Wavelet-Transformation enthält dieses Teilbild eine in der Auflösung reduzierte Darstellung des Originalbildes. Die Art der Reduzierung hängt dabei stark von den angewendeten Wavelets ab. So erhält man z.B. mit den Haar-Wavelets eine Mittelwertbildung. Bei höherwertigen Daubechies-Wavelets erhält man zusätzlich eine Glättung der Daten. Bei der Wavelet-Rücktransformation wird nun, ausgehend von diesen reduzierten Daten, jeweils ein Frequenzband hinzugenommen und durch Anwendung der inversen Filter die um eine Auflösungsstufe höhere Darstellung der Daten rekonstruiert (siehe Abb. 3.10). Es werden also zuerst die niedrigen Frequenzen betrachtet, danach die hohen Frequenzen. Diese Eigenschaft kann man sich bei der Einbindung in ein Steering-Tool zu Nutzen machen, in dem man die Frequenzbänder nacheinander zur Visualisierung-Workstation überträgt [40]. Dort kann die Volumendarstellung schrittweise aufgebaut werden. Auf dem Bildschirm verfeinert sich die Darstellung immer weiter. Eine solche progressive Übertragung von Daten wird, wie schon erwähnt bei der Übertragung vom Bildern im World Wide Web benutzt. Die wesentliche, aber grob gerasterte Information wird am Anfang übertragen, die nachfolgenden Verfeinerungen bringen nur Änderungen in höheren Frequenzen.

Durch die progressive Versenden und die direkte Rekonstruktion der Daten kann die Übertragung nach jedem Frequenzband abgebrochen werden. So kann die FWT für eine Datenreduktion genutzt werden. Je nach Leistungsfähigkeit der Visualisierung-Workstation und Größe des verfügbaren Speichers können die Daten damit in einer angepaßten Auflösung angezeigt werden. Steht die maximal mögliche Auflösung der Visualisierung-Workstation schon vorher fest, können die nicht benötigten höheren Frequenzbänder direkt verworfen werden.

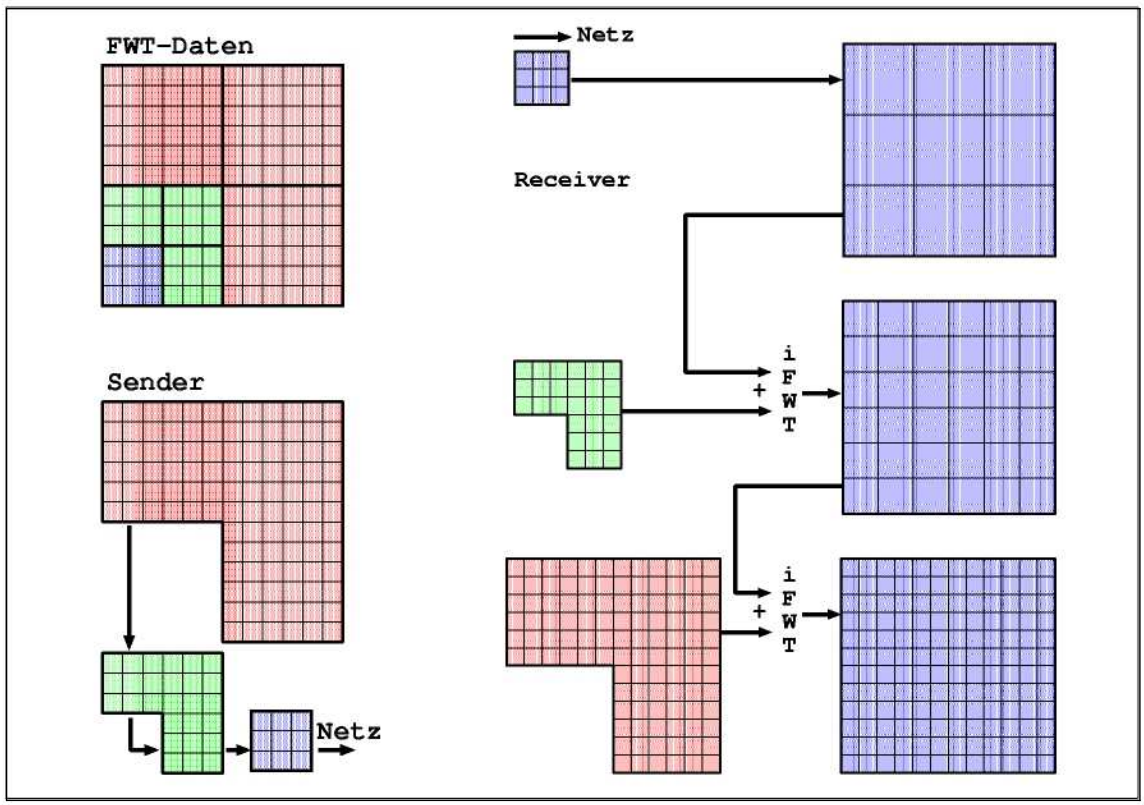


Abbildung 3.10: Progressive Übertragung Wavelet-transformierter Daten: Die Wavelet-transformierten Daten (FWT-Daten) werden auf der Sende-Seite in die einzelnen Frequenzbänder zerlegt und hintereinander über das Netz verschickt. Auf der Empfängerseite werden in jedem Schritt Daten erzeugt, die angezeigt werden. Dazu kann im ersten Schritt das niedrigste Frequenzband direkt genutzt werden. In den darauffolgenden Schritten muß jeweils eine Iteration der inversen FWT durchgeführt werden, wobei die bisher erhaltenen Daten mit dem neuen Frequenzband verknüpft werden.

Die Eigenschaft, daß bei der Rekonstruktion der Daten jeweils die zuvor übertragenden Daten wieder benutzt werden, unterscheidet die progressive Übertragung von Wavelet-transformierten Daten von dem einfacheren Ansatz, den Datensatz in mehreren Iterationen einem Mittelwertfilter zu unterwerfen. Hier wird ähnlich der Wavelet-Transformation eine Menge von Teilbildern erzeugt, die sich in der Auflösung unterscheiden. Nur müssen dabei in jeder Auflösungsstufe alle Daten zur Visualisierung-Workstation übertragen werden. Bei Wavelet-transformierten Daten werden jeweils nur die Daten übertragen, um die sich die beiden letzten Auflösungsstufen unterscheiden. Insgesamt werden bei einem Bild der Größe $n * n$ bei der Wavelet-Transformation nur $n*n$ Datenelemente übertragen (siehe auch Abb. 3.10 auf der vorherigen Seite). Bei dem einfacheren Ansatz reichen dazu die Datenelemente der niedrigeren Auflösungsstufen $(\frac{n}{2})^2, (\frac{n}{4})^2, \ldots)$.

Der zusätzliche Aufwand für die Wavelet-Transformation bleibt mit $O(n)$ (n: Anzahl der Datenelemente) in einem für das Computational Steering akzeptablen Rahmen. Bei der Ankopplung einer Visualisierung an ein Simulationsprogramm muß die komplette Verarbeitung der Daten eines Zeitschrittes (Transformation, Übertragung und Rekonstruktion) in einer Zeit möglich sein, die wesentlich kleiner ist als die Rechenzeit, welche die Simulation für die Berechnung eines Zeitschrittes benötigt. Viele Komprimierungs- und Reduktions-Verfahren sind bezüglich ihres Aufwands nicht symmetrisch. Vielfach muß die Komprimierung nur einmal zur Speicherung der Daten geschehen, die Entkomprimierung muß aber bei jedem Betrachten der Daten durchgeführt werden. Die Rechenzeit für die Komprimierung spielt dann keine wesentliche Rolle. Wichtig ist, daß die Entkomprimierung bzw. Rekonstruktion der Daten sehr schnell ist. Die Wavelet-Transformation ist in diesem Sinne ein symmetrisches Verfahren, da für die Transformation und Rücktransformation jeweils gleichartige Filter benutzt werden.

Ein für den Einsatz im Computational Steering wichtiger Vorteil der Wavelets ist, daß für die Transformation keine Zwischenspeicherung der Daten nötig ist. Bei der Zerlegung können die gefilterten Daten wieder direkt an der Speicherstelle der Originaldaten abgelegt werden. Dies ist aber nur dann möglich, wenn die Originaldaten vom Simulationsprogramm selbst nicht mehr benötigt werden. Fließen die Daten wieder in die Rechnungen zum nächsten Simulationsschritt ein, müssen sie vor der Wavelet-Transformation umkopiert werden.

Nachteil der Wavelet-Transformation mit Hilfe des Pyramidenalgorithmus ist die Reihenfolge, in der die Zerlegung geschieht. In den ersten Iterationen werden die hochfrequenten Frequenzbänder erzeugt. Erst in der letzten Iteration des Pyramidenalgorithmus werden diejenigen Daten erzeugt, welche die niederfrequenten Anteile enthalten und damit zuerst verschickt werden müssen. Eine Überlappung des Versendens der Daten und deren Transformation ist also nicht möglich.

Es bleibt die Frage offen, in welchem Maß die in den Daten enthaltene Information mit der oben beschriebenen Datenreduktion durch Weglassen der hohen Frequenzbänder verändert wird. Durch die Normierung der Filter ist die L_2-Norm der einzelnen Frequenzbänder gleich. Durch das Weglassen von kompletten Frequenzbändern wird diese Norm nicht verändert. Ein Beispiel dafür, daß auch durch das Weglassen kompletter Frequenzbändern Information verloren geht, sind schraffierte Flächen in einem Bild, deren Linienabstand genau der Auflösung des höchsten Frequenzbands entspricht. Wird das Haar-Wavelet zur Transformation benutzt und nach der Transformation das höchste Frequenzband weggelassen, erscheint die schraffierte Fläche im nächst niedrigen Frequenzband nur noch als eine homogene Fläche. Im Gegensatz dazu wird eine einzelne Kante (Linie) in einem Bild durch die Transformation mit Hilfe der Haar-Wavelet und das Weglassen des höchsten Frequenzbands nicht gelöscht. Die Linie führt durch die Differenzbildung im höchsten Frequenzband zu einer Kante, bzw. Linie. Aber auch der Mittelwert zweier benachbarten Pixel, von den eines zu der Linie gehört, ist höher als der Mittelwert anderer Pixel. Dadurch ist die Linie auch in den niederfrequenteren Bändern zu erkennen.

3.1.5 Anwendung der FWT bei parallelen Simulationsprogrammen

Ein Ziel dieser Arbeit ist es, die Ankopplung der Visualisierung an das Simulationsprogramm so zu gestalten, daß es möglichst wenig beeinträchtigt wird. Dieses Ziel kann nur dann erreicht werden, wenn fast alle der für die Ankopplung nötigen Berechnungen parallel ablaufen können. Nur dann werden Wartezeiten auf den nicht an der FWT beteiligten Prozessoren vermieden und die Effizienz des parallelen Programms nicht beeinträchtigt. Für den Einsatz der FWT bei der Ankopplung bedeutet das, daß auch sie parallel ablaufen muß. Die Parallelisierung der schnellen Wavelet-Transformation ist zwar möglich, da es sich im wesentlichen um die Anwendung von skalaren Filtern handelt. Die Verteilung der Berechnungen auf die einzelnen Prozessoren ist aber insbesondere bei dreidimensionalen Problemstellungen aufwendig. Auch die Anforderungen an die Kommunikationsleistung der Rechner ist nicht gering, da Datenblöcke oft zwischen den Prozessoren ausgetauscht werden müssen [41, 42].

Bei der Parallelisierung arbeiten alle Prozessoren gemeinsam an einer Wavelet-Transformation. Die dabei benötigten Berechnungen müssen auf die einzelnen Prozessoren aufgeteilt werden und es muß dafür gesorgt werden, daß für die Berechnungen jeweils die aktuellen Daten bzw. Zwischenergebnisse vorliegen. Bei dem in dieser Arbeit beschriebenen Anwendungsszenario gibt es zwei Gründe gegen diese parallele Berechnung einer Wavelet-Transformation auf dem Gesamtgebiet. Zum einen wird in dieser Arbeit von parallelen Simulationsprogrammen mit verteilter Datenhaltung ausgegangen. Das heißt, daß jeder Prozessor einen eigenen Teilbereich der Daten verwaltet. Eine Parallelisierung der Wavelet-Transformation hängt damit stark von der Art der Datenverteilung ab. Zum anderen besteht auch die aus Sicht der Parallelisierung günstigere Möglichkeit, daß jeder Prozessor eine Wavelet-Transformation seines Teilgebiets vornimmt und das Ergebnis separat zur Visualisierung überträgt. Erst bei der Rekonstruktion auf der Visualisierungsseite werden dann die einzelnen Teilgebiete zu einem Gesamtgebiet zusammengefügt. Abbildung 3.11 auf der nächsten Seite zeigt das Prinzip.

Es ist sicherlich vorteilhaft, daß bei der getrennten FWT die Prozessoren unabhängig voneinander arbeiten können. Jeder Prozessor transformiert sein Gebiet und schickt es zur Visualisierung. Es ist weder eine Synchronisierung noch ein Datenaustausch während der Wavelet-Transformation nötig. Bei vielen Simulationsprogrammen, die keinen perfekten Lastausgleich herstellen können, ist die asynchrone Wavelet-Transformation von Vorteil.

Auf der Empfängerseite erhält die Visualisierung pro Frequenzband nun mehrere Datensätze. Bei der Rekonstruktion der Daten sollen diese aber wieder zu einem Datensatz zusammengefügt werden. Bei den zuerst gesendeten niederfrequenten Ergebnissen des letzten Durchlaufs des Tiefpasses kann dies durch einfaches Zusammenfügen geschehen (siehe auch Abb. 3.11 auf der nächsten Seite). Bei der Anwendung der inversen FWT muß darauf geachtet werden, daß diese für jedes Teilgebiet einzeln durchgeführt wird. Ein ähnliches Zusammenfügen wie bei den niederfrequenten Anteilen ist hier nicht erlaubt, da auf der Sendeseite an den Rändern der Teilgebiete die Daten jeweils zyklisch erweitert oder mit Null-Werten fortgesetzt wurden. Es können dann Artefakte entstehen, wenn die Ergebnisse der Hochpaßfilters einfach aneinandergefügt werden und die inverse Wavelet-Transformation dann für das Gesamtgebiet einmal ausgeführt wird. Daher sollte die inverse FWT auf der Visualisierungsseite auch für jedes Teilgebiet einzeln ausgeführt werden. Die Komplexität wird durch die mehrmalige, dafür aber auf kleineren Gebieten ausgeführte inverse FWT, nicht erhöht.

3.1.6 Reduzierung durch Abschneiden von kleinen Koeffizienten

Der bei der Wavelet-Transformation angewendete Übergang vom Orts- in den Frequenzraum kann für die Anwendung im Computational Steering zu weiteren Optimierungen genutzt werden. Ein

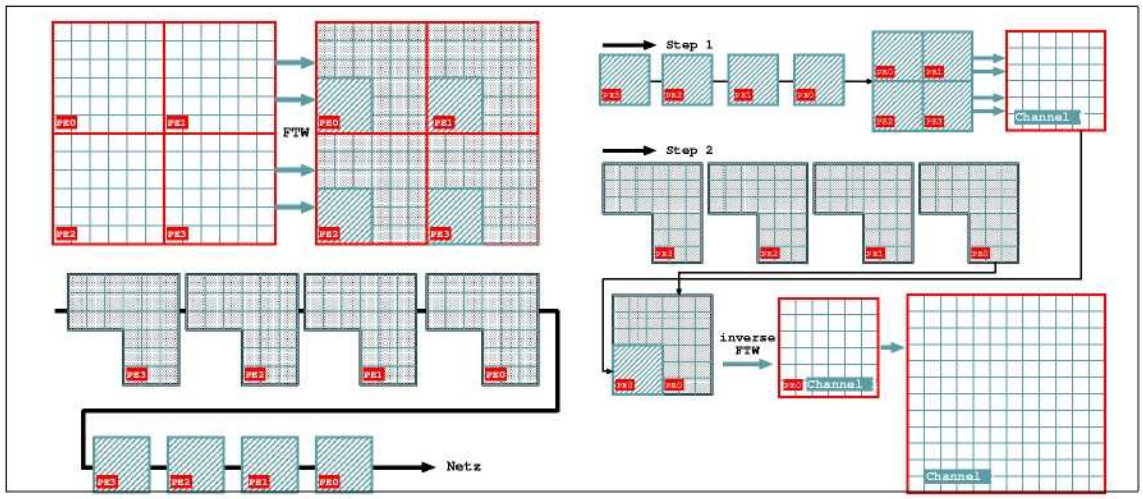


Abbildung 3.11: Progressive Übertragung von verteilten und Wavelet-transformierter Daten: Die in diesem Beispiel zweidimensionalen Daten sind blockweise auf vier Prozessoren verteilt. Vor dem Versenden der Daten führt jeder Prozessor auf seinem Teilgebiet die schnelle Wavelet-Transformation durch (linker Teil des Abbildung). Beim Versenden der Daten werden zuerst alle niederfrequenten Anteile zur Visualisierung übertragen. Die Übertragung der weiteren Frequenzbänder erfolgt wieder für alle Prozessoren gemeinsam. Erst bei der Rekonstruktion werden die Teilgebiete der einzelnen Prozessoren zusammengefügt. Die inverse FWT muß dabei für jedes Teilgebiet einzeln ausgeführt werden.

Ansatz ist die oben beschriebene Reduzierung der Daten durch einfaches Weglassen von ganzen Frequenzbändern. Die dabei zu erzielenden Reduktionsraten sind sehr hoch, da in jeder Iteration die Größe in jeder Dimension um den Faktor zwei verkleinert wird. Nachteil dieser Methode ist, daß alle Koeffizienten des betreffenden Frequenzbandes unberücksichtigt bleiben. Ein Ansatz, der diesen Nachteil ausräumt, ist die Reduzierung durch Weglassen nur von einzelnen Koeffizienten. Dabei werden nur diejenigen Koeffizienten eines Frequenzbandes übertragen, die einen gewissen Informationsgehalt besitzen. Aus Sicht der Implementierung werden alle nicht interessanten Koeffizienten auf Null gesetzt und anschließend eine Komprimierung durchgeführt, die z.B. mit Hilfe der Run-Length-Encoding-Methode (RLE) lange Sequenzen von Null-Werten effizient komprimieren kann (siehe auch Abschnitt 3.3 auf Seite 37).

Abbildung 3.13 auf der nächsten Seite zeigt ein Beispiel für die Häufigkeitsverteilung von Wavelet-Koeffizienten in den vom Hochpaßfilter gelieferten Frequenzbändern. Man erkennt die für Wavelet-Transformation typische Häufung der Koeffizienten um den Nullpunkt. Sie ist immer dann vorhanden, wenn in den Originaldaten homogene Bereiche enthalten sind. Dort unterscheiden sich benachbarte Werte nicht oder nur geringfügig und erzeugen so bei der Wavelet-Transformation in höheren Frequenzbändern nur kleine Koeffizientenwerte. Durch das Abschneiden dieser kleinen Werte wird das Ergebnis nur wenig verfälscht. Gegenüber dem Weglassen des ganzen Frequenzbandes werden nun große Koeffizienten dort berücksichtigt.

Eine wichtige Einflußgröße für die Qualität der resultierenden Daten ist der *Cut-Level*, der die Obergrenze bestimmt, bis zu der Koeffizienten abgeschnitten werden. Betrachtet wird dabei der Absolutwert des Koeffizienten, so daß im Histogramm ein symmetrischer Bereich um den Nullpunkt abgeschnitten wird. Eine direkte Angabe dieses Schwellwertes führt zu einer schwierigeren Kontrolle des durch das Abschneiden entstehenden Fehlers in den Daten. Das Histogramm und insbesondere dessen Minimal- und Maximalwerte sind durch die Originaldaten beeinflußt, so daß im Falle des Computational Steering bei jeder Aktualisierung der Daten der Cut-Level neu angepaßt werden muß. Statt der direkten Angabe eines Wertes sollte hier der Anteil der zu löschenden Koeffizienten (in Prozent der Anzahl) oder der maximal erlaubte relative L_2-Fehler benutzt werden.

Abbildung 3.12: Abschneiden von Koeffizienten bei einem Wavelet-transformierten Bild: Auf das linke Bild wurde die FWT zweimal angewendet und anschließend 97% der Koeffizienten zu Null gesetzt. Die Datengröße des so reduzierte Bild (rechts) hat sich dabei um den Faktor 10 verringert.

Der Aufwand, der für das Löschen aller Koeffizienten unterhalb eines Schwellwert erbracht werden muß, liegt bei n Koeffizienten bei O(n), da für jeden Koeffizient direkt über sein Löschen entschieden werden kann. Wird der Schwellwert in Abhängigkeit aller Koeffizienten definiert, wie das bei einem Prozent-Anteil oder einem maximalen Fehler der Fall ist, muß dieser vor dem eigentlichen Löschvorgang bestimmt werden. Dazu ist eine Sortierung der Koeffizienten nach ihrer absoluten Größe notwendig, die sich in der Aufwandsberechnung mit O(n log n) niederschlägt.

Steht ein Histogramm der Koeffizienten zur Verfügung, kann eine Näherungslösung für den Cut-Level gefunden werden, ohne die Koeffizienten zu sortieren. Im Computational Steering ist die exakte Bestimmung des Schwellwertes weniger wichtig als der durch die Abschätzung erhaltene Performancegewinn. Der Aufwand für die Erstellung des Histogramms liegt bei O(n) und die Bestimmung der Abschätzung hängt nur von der Anzahl der im Histogramm benutzten Teilintervalle ab. Das Histogramm kann schon während der Berechnung der FWT mitgeführt werden und muß damit nicht in einem eigenen Bearbeitungsschritt berechnet werden.

In einem Histogramm wird der von den Koeffizienten benutzte Wertebereich $[d_{\min}, d_{\max}]$ der Koeffizienten in m Intervalle zerlegt. Die Histogrammfunktion $H(i)$ beschreibt dann die Anzahl der

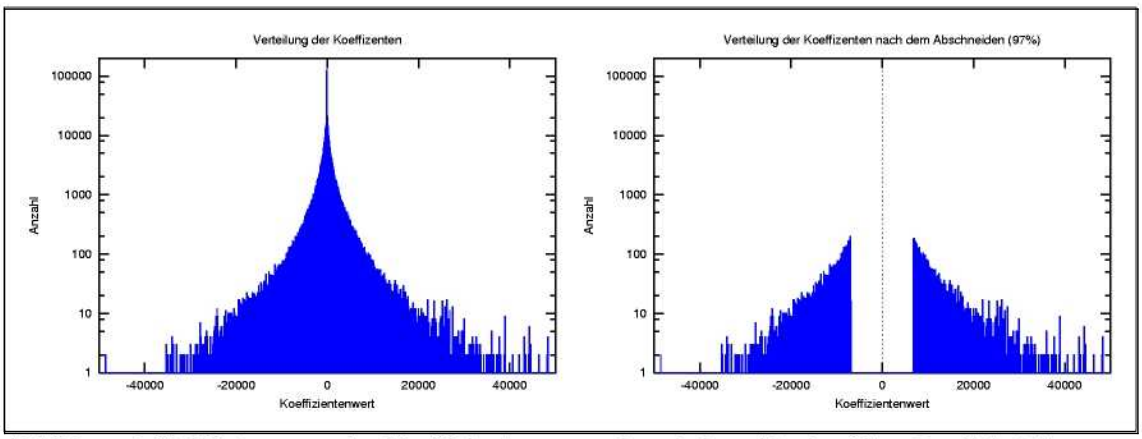


Abbildung 3.13: Histogramm der Koeffizienten vor und nach dem Abschneiden: Das linke Diagramm zeigt die absolute Häufigkeit von Koeffizienten in den vom Hochpaßfilter berechneten Frequenzbändern des in Abb. 3.12 dargestellten Bildes. Das rechte Diagramm zeigt die Häufigkeitsverteilung der Koeffizienten nach Abschneiden von 97% der Koeffizient um den Nullpunkt. Zur besseren Darstellung sind die Werte logarithmisch aufgetragen.

Elemente, die im Intervall $[d_i, d_{i+1}[$ liegen, wobei für die Intervallgrenzen folgendes gilt:

$$d_i = d_{\min} + \frac{d_{\max} - d_{\min}}{m}$$

Wenn z.B. n_l der n Koeffizienten gelöscht werden sollen, kann die Abschätzung für den Schwellwert s wie folgt bestimmt werden:

$$s = d_{j+1} + (d_{j+2} - d_{j+1}) * \frac{H(j+1) - D(j+1, n_l)}{H(j+1)} \tag{3.28}$$

$$j = \max\{l \in \{0..m\} | \sum_{k=0}^{l} H(k) < n_l\}, \qquad D(j,g) = \sum_{k=0}^{j} H(k)) - g$$

Die Genauigkeit des Schwellwertes hängt damit von der Anzahl der im Histogramm benutzten Teilintervalle ab, da nur ein Teil der im letzten Intervall $H(j+1)$ liegenden Koeffizienten gelöscht werden darf und dieser Anteil nur abgeschätzt werden kann. Für die Bestimmung des Schwellwertes über den maximal erlaubten Fehler muß neben dem Histogramm nicht die Anzahl der in jedem Teilintervall liegenden Koeffizienten wiedergeben werden, sondern die Summe der L_2-Fehler , die das Abschneiden der Koeffizienten bedingen würde. Die Gewichtung erfolgt dann über diese Fehlersummen. Abbildung 3.12 auf der vorherigen Seite zeigt ein Beispiel für die Anwendung des Koeffizienten-Abschneidens bei einem Bild.

Das Löschen bringt nur Vorteile für die bei der Übertragung genutzte Komprimierung der Daten, da die Null-Sequenzen für den Transport sehr gut komprimiert werden können, aber auf der Visualisierungsseite wieder in der entsprechend höheren Auflösung vollständig rekonstruiert werden müssen. Die Auflösung und damit die Komplexität der Daten hat sich damit nicht geändert. Bei einem Volumen erhöht sich die Anzahl der Voxel bei jedem zusätzlichen Frequenzband um den Faktor acht. Für die Anzeige der Daten bringt diese Methode nur dann Vorteile, wenn dabei abgeleitete Darstellungsarten, wie z.B. Isosurfaces, genutzt werden können. Diese profitieren dann von homogenden Bereichen, da dort z.B. Oberflächen durch weniger Graphikelemente (Dreiecke) dargestellt werden können.

Die in den Wavelets eingebaute Normierung erlaubt es, beim Abschneiden von einzelnen Koeffizienten alle Frequenzbänder gleichzeitig zu betrachten. Sie sorgt dafür, daß bei der Reduzierung der Auflösung die Koeffizienten höher gewichtet werden und somit ggf. über dem Schwellwert liegen können.

3.1.7 Abschneiden von kleinen Koeffizienten bei verteilten Daten

Das Löschen der Koeffizienten kann bei verteilten Daten in einem parallelen Simulationsprogramm auch parallel ablaufen. Der Schwellwert kann dabei entweder lokal oder global für alle Teilbereiche gemeinsam bestimmt werden.

Soll der Schwellwert global bestimmt werden, müssen die lokal berechneten Histogramme angeglichen werden. Dazu müssen vor der Berechnung der Histogramme der Minimal- und Maximalwert des kompletten Datensatzes berechnet werden, damit die im Histogramm definierten Teilintervalle gleiche Größe und Position im Wertebereich besitzen. Dabei ist eine globale Kommunikation zwischen allen Prozessoren nötig. Danach kann die Anzahl der Koeffizienten pro Teilintervall lokal auf jedem Prozessor bestimmt und global aufsummiert werden. Die Berechnung des Schwellwertes kann wieder lokal erfolgen.

Bei einer lokalen Bestimmung des Schwellwertes und einer nicht gleichmäßigen Verteilung der kleinen Koeffizienten kann es starke Schwankungen der lokalen Schwellwerte geben. Das führt

bei der Visualisierung zu unterschiedlichen Darstellungen der Teilbereiche und bei abgeleiteten Darstellungsarten ggf. zu Artefakten.

3.2 Quantisierung

Die bei Simulationsrechnungen anfallenden Daten liegen häufig als Gleitkommazahlen vor. Ihre Speicherung ist insbesondere bei der vielfach in Simulationsprogrammen benutzen doppelten Genauigkeit sehr aufwendig. Da Gleitkommazahlen im allgemeinen keine hohe Komprimierungsraten zulassen, sollten sie für den Transport zur Visualisierung in ganzzahlige Werte umgerechnet werden, welche für die Speicherung und Komprimierung der Daten wesentlich günstiger sind. Diese als *Quantisierung* bezeichnete Umrechnung bedarf einiger Vorüberlegungen. So sollte z.B. wie bei der Bildung des Histogramms im vorigen Abschnitt der Wertebereich der Datenelemente eingegrenzt werden. Dieser Bereich kann dann in Intervalle eingeteilt werden. Jedem Datenelement kann nun ein Teilintervall zugeordnet werden und statt der Speicherung des Wertes selbst ist nur noch die Speicherung der laufenden Intervallnummer (Index) notwendig. Bei der Rekonstruktion der Werte auf der Visualisierungsseite kann aus diesem Index, der Anzahl der Teilintervalle sowie dem Minimal- und Maximalwert eine Gleitkommazahl generiert werden. Mathematisch gesehen handelt es sich bei Quantisierung um eine Treppenfunktion, die jeder reellwertigen Zahl eine natürliche Zahl zuordnet:

$$Q : \mathbb{R} \rightarrow \mathbb{N},\ Q(x) = i \qquad \text{falls} \qquad d_i \leq x < d_{i+1} \tag{3.29}$$
$$\text{mit} \qquad d_i = d_{\min} + \frac{d_{\max} - d_{\min}}{m}$$

Für die Rekonstruktion wird für die neue Gleitkommazahl der Mittelpunkt zwischen den zum entsprechenden Teilintervall gehörenden Intervallgrenzen benutzt:

$$R : \mathbb{N} \rightarrow \mathbb{R},\ Q^{-1}(i) = r_i \qquad \text{mit} \qquad r_i = \frac{d_i + d_{i+1}}{2} \tag{3.30}$$

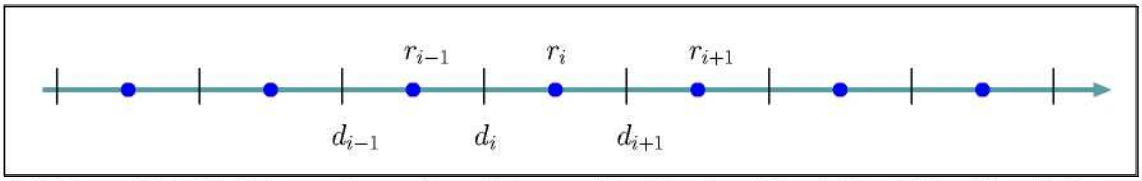


Abbildung 3.14: Einfache uniforme Quantisierung: Der Wertebereich wird in gleichmäßige Teilintervalle unterteilt. Jedem Datenelement wird ein Intervall zugeordnet. Bei der Rekonstruktion wird für das Datenelement der Wert des Mittelpunktes der Intervallgrenzen festgelegt.

Diese einfachste Form der Quantisierung (Abb. 3.14) erzeugt bei der Rekonstruktion einen sehr großen L_2-Fehler , da r_i immer auf die Mitte des Teilintervalls gelegt wird. Günstiger ist es, diesen Punkt anhand der im Intervall gelegenen Datenelemente zu orientieren. Dazu müssen während der Quantisierung die Datenwerte für jedes Teilintervall summiert werden. Der Rekonstruktionswert r_i kann dann durch die Summe gewichtet werden:

$$R : \mathbb{N} \rightarrow \mathbb{R},\ Q(i) = r_i \qquad \text{mit} \qquad r_i = \frac{1}{n_i} \sum_{i=1}^{n_i} x_i \quad , \quad d_i \leq x_i < d_{i+1} \tag{3.31}$$

Es kann hergeleitet werden, daß für die äquidistanten Intervallgrenzen die so berechneten Schwerpunkte den kleinsten L_2-Fehler ergeben [35].

Im häufig verwendeten Lloyd-Max-Quantisierer [43] wird die hier beschriebene Gewichtung der

Rekonstruktionswerte dahingehend erweitert, daß mit Hilfe der Intervall-Schwerpunkte r_i die Intervallgrenzen neu bestimmt werden:

$$d_i = \frac{r_{i-1} + r_i}{2} \tag{3.32}$$

Dadurch werden die Koeffizienten jeweils zu einem Wert r_i rekonstruiert, der am nächsten zu diesem liegt. Die so verschobenen Intervallgrenzen führen wiederum zu anderen Schwerpunkten r_i, die erneut berechnet werden müssen. Der Lloyd-Max-Quantisierer wiederholt diese beiden Schritte (Berechnung von d_i und r_i) so lange, bis ein Konvergenzkriterium erreicht worden ist. Hier können zum Beispiel die Veränderungen der d_i, der r_i oder der L_2-Fehlers selbst betrachtet werden. Der Algorithmus verschiebt die Intervallgrenzen so, daß sie in Bereichen, in denen das Histogramm große Steigungen aufweist, nahe aneinanderrücken, und in Bereichen, in denen die Werte gleichmäßig verteilt sind, weiter voneinander entfernt stehen (Abb. 3.15).

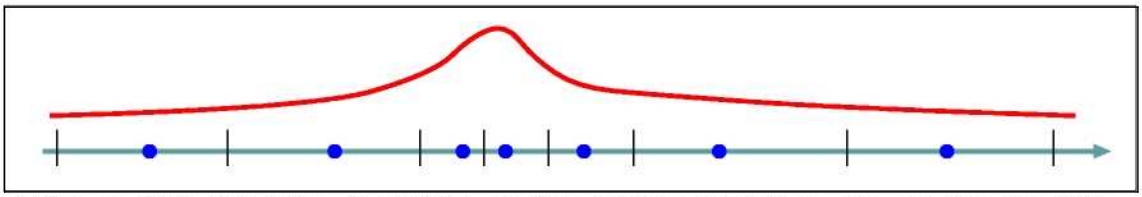


Abbildung 3.15: Lloyd-Max-Quantisierung: Schematische Darstellung der von dem Lloyd-Max-Quantisierer festgelegten Intervallgrenzen und Rekonstruktionswerte. Die obere Kurze zeigt die Verteilung der Koeffizienten, die senkrechten Markierungen die Intervallgrenzen und die Punkte die r_i, die bei der Rekonstruktion der Koeffizienten genutzt werden.

Für die Anwendung im Computational Steering eignet sich das Lloyd-Max-Verfahren nur bedingt. Die Intervallgrenzen sind in den weiteren Iterationen des Verfahren nicht mehr äquidistant verteilt. Daher kann das zu einem Koeffizienten gehörende Intervall nicht mehr direkt bestimmt werden. Vielmehr muß es in der Menge der Teilintervalle gesucht werden, das z.B. bei binärem Suchen einen zusätzlichen Aufwand von O(log n) pro Koeffizient bedeutet. Insgesamt erhöht sich damit die Komplexität der Quantisierung von O(n) auf O(n log n). Zudem müssen in jeder Iteration zur Bestimmung der neuen Rekonstruktionswerte alle Koeffizienten neu untersucht werden. Es ist daher empfehlenswert, in diesem Anwendungsfall nur die Gewichtung der Rekonstruktionswerte vom Lloyd-Max-Verfahren zu übernehmen.

Für die Rekonstruktion der Daten müssen auf der Visualisierungsseite sowohl die Index-Werte der einzelnen Koeffizienten als auch die Rekonstruktionwerte r_i vorliegen. Die Anzahl der Indizes ist durch die Anzahl der Koeffizienten (n) begrenzt. Die Anzahl der Rekonstruktionwerte ist durch die Bit-Anzahl (q) der für die Speicherung der Indizes verwendeten Zahlenformate bestimmt. Wird z.B. für die Speicherung eines Index ein Byte ($q = 8$) benutzt, können maximal $2^8 = 256$ Teilintervalle adressiert werden. Übliche Werte für q liegen im Bereich von 8 bis 16 Bits. Die Anzahl der zu übertragenden Rekonstruktionwerte ist fest und unabhängig von n und kann bei der Betrachtung des Aufwands vernachlässigt werden (Abb. 3.16 auf der nächsten Seite).

Bei der Bestimmung der Intervallgrenzen können bekannte Eigenschaften der Daten berücksichtigt werden. Dazu gehört zum Beispiel, daß durch das Abschneiden von kleinen Koeffizienten in einem Bereich um den Nullpunkt keine Werte zu finden sind. Diese *Deadzone* kann bei der Bildung der Teilintervalle ausgeschlossen werden. Die Größe der restlichen Teilintervalle wird damit kleiner und damit die Genauigkeit der rekonstruierten Daten höher. Da der Null-Wert genau in dieser Deadzone liegt, muß für ihn eine getrennte Indizierung erfolgen.

Bei der Quantisierung können die Koeffizienten der einzelnen Frequenzbänder zusammen betrachtet werden. Ausnahme ist nur das niedrigste Frequenzband, das vom letzten Durchlauf des Tiefpaßfilters erzeugt worden ist. In diesem Frequenzband sind die Werte direkt und nicht die

Abbildung 3.16: Beispiel: Effekt der Quantisierung auf die Qualität eines Bild: Die drei Bilder sind jeweils mit unterschiedlicher Anzahl von Bits quantisiert worden. Für Speicherung der Datenwerte sind im linken Bild 4, im mittleren 8 und im rechten Bild 16 Bits benutzt worden. Schon bei einer 8-Bit-Quantisierung sind deren Effekte kaum noch zu erkennen.

Koeffizienten der Wavelets enthalten, so daß das Histogramm unterschiedlich zu dem der Koeffizienten ist.

Eine leistungsfähige aber auch aufwendigere Alternative zu der hier vorgestellten skalaren Quantisierung ist die Vektorquantisierung. In diesem Verfahren werden nicht nur einzelne Datenwerte, sondern eine Menge (Vektor) dieser Werte betrachtet. Bei einem Bild können dies z.B. Teilbilder mit 4x4 oder 8x8 Pixel sein. Die Menge der im Datensatz enthaltenen Vektoren wird in einem *Codebook* verwaltet, so daß statt des Vektors (Teilbild) nur der Index des Vektors im Codebook gespeichert werden muß. Der Aufwand für diese Quantisierung hängt stark von der Anzahl der verschiedenen Vektoren ab und wird auch wesentlich von der Suche im Codebook bestimmt. Zusätzlich muß zu den Indizes auch das Codebook übertragen werden. Im Falle des Computational Steering können sich die Datensätze von Simulationsschritt zu Simulationsschritt stark ändern, so daß das Codebook entweder jeweils erweitert oder neu erstellt werden muß [44].

3.2.1 Quantisierung bei verteilten Daten

Auch die Quantisierung kann bei parallelen Simulationsprogrammen mit verteilter Datenhaltung parallel ablaufen. Jeder Prozessor besitzt dabei ein Teilgebiet, das unabhängig von den anderen Prozessoren Wavelet-transformiert und quantisiert werden kann. Wie im vorigen Abschnitt „Reduzierung durch Abschneiden von kleinen Koeffizienten“ beschrieben, müssen aber vor der Bestimmung der Gewichte die Minimal- und Maximalwerte der Koeffizienten global abgestimmt werden. Danach benutzen alle Prozessoren denselben Bereich für die Bestimmung der Teilintervalle.

Werden die Gewichte global bestimmt, braucht nur ein Prozessor diese zu übertragen. Dazu müssen pro Teilintervall die Anzahl der darin enthaltenen Koeffizienten (n_i) und ihre Summe global aufsummiert werden. Die Berechnung der Gewichte erfolgt dann wieder gemäß Gleichung (3.31).

Wenn die Gewichte r_i nur lokal bestimmt werden, müssen sie mit jedem Teilgebiet zur Visualisierung übertragen werden. Dieser erhöhte Aufwand bei der Übertragung erzielt eine höhere Genauigkeit der Rekonstruktionwerte, da die Gewichte nur für Teilmengen der Koeffizienten berechnet werden. Jedoch treten auch hier Probleme an den Rändern der Teilgebiete auf, da dort benachbarte Werte unterschiedlich quantisiert worden sein können. Besonders bei abgeleiteten Darstellungsformen (Iso-Oberflächen) führt das zu Artefakten.

3.3 Kodierung der transformierten und quantisierten Daten

Bevor die Wavelet-transformierten und quantisierten Daten zur Visualisierung übertragen werden, sollten diese in einem weiteren Schritt komprimiert werden. Durch die letzten drei Verarbeitungs-

schritte stehen die Daten in Form von ganzzahligen Werten (1-2 Byte) zur Verfügung und enthalten durch die Elimination von kleinen Koeffizienten im allgemeinen sehr viele Nullwerte. Zusätzlich sind die Daten in verschiedene Frequenzbänder zerlegt, die sich in ihrer Auflösung unterscheiden.

Diese durch die Wavelet-Transformation und Quantisierung verursachten Eigenschaften können für die Komprimierung ausgenutzt werden. Das heißt, daß vor der eigentlichen Komprimierung mit Hilfe eines Standard-Tools (RLE, Lempel-Ziv, Huffman) eine an diese Eigenschaften angepaßte Verarbeitung vorgeschaltet werden soll. Es gibt mehrere Möglichkeiten, solche mit Null-Werten durchsetzten Daten zu kodieren. Ihre Komprimierungsrate hängt dabei stark von der Anzahl der in den Daten enthaltenen Null-Werten ab.

Befinden sich in einem Frequenzband sehr viele Null-Werte, empfiehlt es sich, nur die von Null verschiedenen Koeffizienten aus den Daten herauszusuchen und diese zur Visualisierung zu übertragen. Für die Rekonstruktion müssen zu diesen Werten auch deren Positionen innerhalb des Frequenzbandes übertragen werden, was zusätzlichen Aufwand bedeutet und damit die Einsparungen durch das Abschneiden der Null-Werte zumindest teilweise wieder aufhebt.

Eine Speichermethode, die dann sinnvoll ist, wenn weniger Null-Koeffizienten in den Daten vorhanden sind, ist die Umsortierung anhand der Zusammengehörigkeit der einzelnen Koeffizienten. Bei einer Iteration des Pyramidenalgorithmus werden benachbarte Datenelemente in zwei Frequenzbereiche zerlegt. Im eindimensionalen Fall werden dabei zwei Werte jeweils in einen Mittelwert und einen Differenzwert zerlegt. Im zwei- bzw. dreidimensionalen Fall entstehen drei bzw. sieben Differenzwerte und ein Mittelwert. Die so zu einem Datenelement im reduzierten Frequenzband gehörenden Koeffizienten können nun zusammen abgespeichert werden. Durch eine vorangestellte Bitmaske werden dabei diejenigen Koeffizienten markiert, die verschieden von Null sind [45, 46]. Bei dieser zweiten Speichermethode entsteht zusätzlicher Speicherbedarf durch die jeweils abzuspeichernden Bitmasken. Im Gegenzug dazu brauchen die Null-Koeffizienten nicht mit gespeichert werden[1].

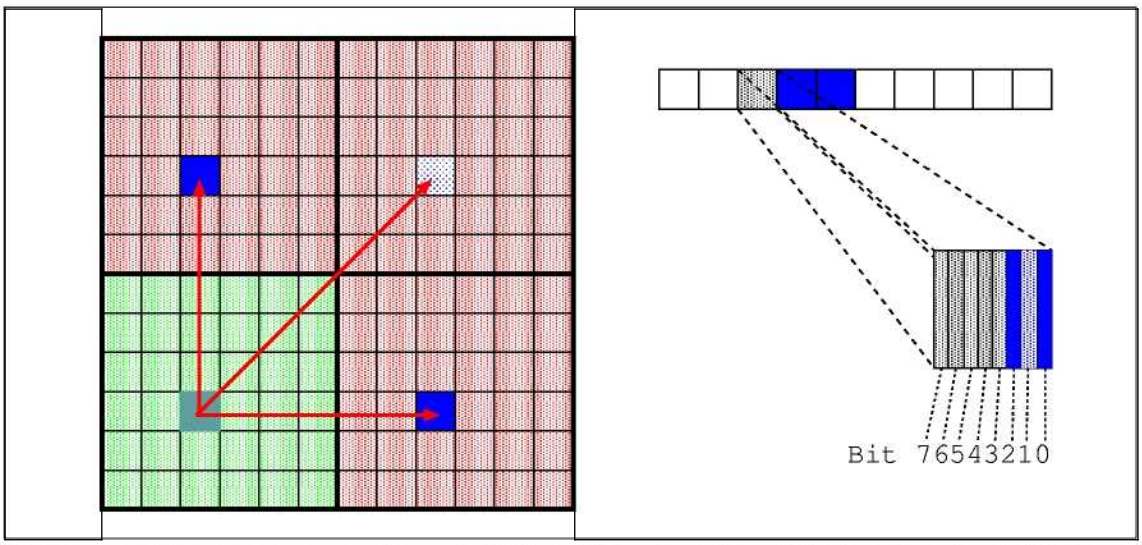


Abbildung 3.17: Gruppierung der Koeffizienten bei der Kodierung: Die zu einem Datenelement im niedrigeren Frequenzband gehörenden Koeffizienten aus dem nächst höheren Frequenzband werden zusammen kodiert. Durch das Voransetzen einer Bitmaske werden die Null-Koeffizienten markiert und müssen daher nicht mehr in die komprimierten Daten eingebunden werden.

[1] Eine getrennte Speicherung von Bitmaske und Koeffizienten würde bei den Bitmasken eine höhere Packungsdichte erlauben, da bei der obigen Methode im dreidimensionalen Fall ein und zweidimensionalen Fall fünf Bit unbenutzt bleiben. Nachteil dabei ist, daß zwei Datenströme (Bitmasken und Koeffizienten) gespeichert und übertragen werden müssen. Auch eine direkte Zuordnung der Bitmaske zu den Koeffizienten ist nicht mehr möglich. Zumal im dreidimensionalen Fall der Gewinn eher gering ist, wird die getrennte Speicherung hier nicht weiter berücksichtigt.

Ist der Anteil der Null-Koeffizienten zu gering, lohnen sich die beiden Speicherungsmethoden nicht. Hier sollten alle Werte übertragen werden. Die wenigen Null-Werte können dann der nachgeschalteten Standard-Komprimierung überlassen werden.

Die Auswahl von einem der drei oben beschriebenen Verfahren zur Vorverarbeitung der Daten kann in Abhängigkeit von der Anzahl der in den Frequenzbändern enthaltenen Null-Koeffizienten geschehen. In der folgenden Rechnung wird die Gesamtanzahl der Koeffizient mit n und die Anzahl der Null-Koeffizienten mit n_0 bezeichnet. Für den Anteil der Null-Koeffizienten an der Gesamtanzahl ergibt sich somit $f_0 = n/n_0$.

Bei der direkten Speicherung der Koeffizienten entspricht der Speicherbedarf für einen Null-Koeffizienten dem eines normalen Koeffizienten, da die Null-Koeffizienten nicht gesondert betrachtet werden. Es ergibt sich folgender Speicherbedarf:

$$S_1(n, f_0, m) = (1 - f_0)\, n\, n_{\text{quant}} + f_0\, n\, n_{\text{quant}} = n\, n_{\text{quant}} \tag{3.33}$$

n_{quant} ist dabei die Anzahl der Bytes, die für die Speicherung eines quantisierten Koeffizienten benötigt wird (z.B. 1 oder 2 Byte).

Im zweiten Verfahren werden Koeffizienten zu Gruppen zusammengefaßt und mit einer Bitmaske versehen. Es ergibt sich folgender Bedarf an Speicherplatz:

$$S_2(n, f_0, m) = (1 - f_0)\, n\; n_{\text{quant}} + \frac{n}{2^m} n_{\text{mask}} = n(\,(1 - f_0) n_{\text{quant}} + \frac{1}{2^m} n_{\text{mask}}) \tag{3.34}$$

m bezeichnet in dieser Formel die Anzahl der Dimensionen. Der zweite Term beschreibt den für die Bitmaske benötigten Speicherplatz. Da dieser nur für die Datenelemente des niedrigeren Frequenzbandes benötigt wird, ist er um den Faktor $1/2^m$ kleiner. Die Speicherung der Maske selbst benötigt bei bis zu drei Dimensionen nur ein Byte (n_{mask}=1).

Die Extraktion der von Null verschiedenen Koeffizienten im dritten Verfahren benötigt eine zusätzliche Speicherung der Positionen im Frequenzband. Es wird davon ausgegangen, daß die Positionen der Koeffizienten linear adressierbar sind, so daß für die Speicherung ein ganzzahliger Wert ausreicht:

$$S_3(n, f_0, m) = n\; (1 - f_0)\, (n_{\text{int}} + n_{\text{quant}}) \tag{3.35}$$

Abbildung 3.18 zeigt den Verlauf der durch die Gleichungen (3.33), (3.34) und (3.35) beschriebenen Speicherplatzanforderungen. Je nach Anteil der Null-Koeffizienten sollte das Verfahren mit

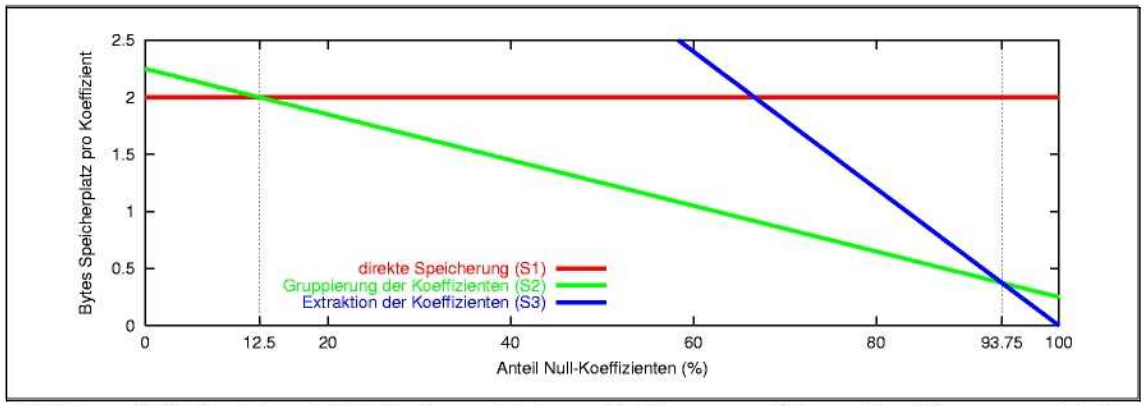


Abbildung 3.18: Speicherplatzbedarf verschiedener Kodierungsverfahren: Das Diagramm zeigt den durchschnittlichen Speicherplatzbedarf der drei Verfahren für einen Koeffizienten. Die Parameter haben dabei folgende Werte: m=2, n_{quant}=2, n_{mask}=1 und n_{int}=4. Besonders durch das zweite Verfahren kann der Speicherplatzbedarf bei einer mittleren Anzahl von Null-Werten verkleinert werden.

dem niedrigsten Speicherplatzbedarf benutzt werden. Zu den Schnittpunkten können aus den Gleichungen folgende Bedingung aufgestellt werden:

$$S_1 < S_2 \Leftrightarrow f_0 < p_{12}, \quad p_{12} = \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{quant}}} \tag{3.36}$$

$$S_2 < S_3 \Leftrightarrow f_0 < p_{23}, \quad p_{23} = 1 - \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{int}}} \tag{3.37}$$

Die beiden Schnittpunkte p_{12} und p_{23} unterteilen den möglichen Wertebereich von f_0 in drei Abschnitte und bestimmen somit das günstigste Verfahren. Es sind immer dann drei Abschnitte vorhanden, wenn folgende Bedingung gilt:

$$p_{12} < p_{32} \quad \Leftrightarrow \quad \frac{n_{\text{mask}}}{n_{\text{quant}}} + \frac{n_{\text{mask}}}{n_{\text{int}}} < 2^m \quad \Leftrightarrow \quad \frac{1}{n_{\text{quant}}} + \frac{1}{n_{\text{int}}} < \frac{2^m}{n_{\text{mask}}} \tag{3.38}$$

Diese Bedingung ist im allgemeinen erfüllt, da die linke Seite der Ungleichung für alle möglichen Größen für $n_{\text{int}} > 1$ und $n_{\text{quant}} \geq 1$ kleiner zwei ist und für die Bitmaske (n_{mask}) erst ab vier Dimensionen mehr als ein Byte benötigt wird und damit die rechte Seite der Ungleichung größer oder gleich zwei ist.

Wenn jeweils das günstigste Verfahren gewählt wird, ergibt sich für den Speicherplatzbedarf der Koeffizienten eines Frequenzbands folgende Gleichung:

$$S = \begin{cases} n\, n_{\text{quant}} & \text{falls} \quad f_0 \leq \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{quant}}} \\ n\,((1-f_0)n_{\text{quant}} + \frac{1}{2^m}n_{\text{mask}}) & \text{falls} \quad \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{quant}}} < f_0 \leq 1 - \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{int}}} \\ n\,(1-f_0)(n_{\text{int}} + n_{\text{quant}}) & \text{falls} \quad 1 - \frac{1}{2^m}\frac{n_{\text{mask}}}{n_{\text{int}}} < f_0 \end{cases} \tag{3.39}$$

3.3.1 Verlustfreie Komprimierungsverfahren

In jedem der oben beschriebenen Fälle sollten die so modifizierten Daten mit einem verlustfreien Verfahren weiter komprimiert werden. Ein Grund dafür sind die gerade in der zweiten Methode entstehenden Muster in den Daten. So folgen dort z.B. einer ein Byte langen Bitmaske jeweils null bis sieben Koeffizienten. Es wird sehr häufig vorkommen, daß alle der sieben Koeffizienten null sind und dadurch nur das Null-Byte der Bitmaske abgespeichert wird. Die daraus entstehenden Folgen von Null-Bytes sollten in jedem Fall noch komprimiert werden. Ein Einsatz einer weiteren Komprimierung ist auch dann sinnvoll, wenn für die Quantisierung eine andere Anzahl von Bits als 8 oder 16 benutzt worden ist. Dadurch, daß die Speicherung auf Byte-Grenze erfolgt, bleiben Bits ungenutzt und können durch Komprimierungsverfahren vermieden werden, die auf Bit-Ebene arbeiten. Bei der ersten Methode, die zu jedem Koeffizienten noch dessen Position innerhalb des Frequenzbandes abspeichert, ist nicht mit wiederkehrenden Muster zu rechnen, da die Positionen zwangsläufig alle voneinander verschieden sind.

Die für die Komprimierung von Wavelet-transformierten und quantisierten Daten in Frage kommenden Verfahren können generell in die Klasse der *statistischen Verfahren* und in Klasse der *substituierenden Verfahren* unterteilt werden [47].

Statistische Komprimierungsverfahren

Das klassische Beispiel für ein statistisches Verfahren ist der oft genutzte *Huffman-Kodierer*, der ein sehr einfaches aber elegantes Verfahren zur Komprimierung darstellt [48]. Er nutzt dazu die Eigenschaft aus, daß die Eingabedaten aus verschiedenen Symbolen bestehen (ein oder mehrere Bytes), die zusammen ein Alphabet beschreiben. Den einzelnen Symbolen des Alphabets wird

jeweils, abhängig von der Häufigkeit ihres Vorkommens, ein Code zugeordnet. Statt der Symbole selbst müssen dann nur diese Codes abgespeichert werden. Eine Komprimierung der Daten wird nun dadurch erreicht, daß die einzelnen Codes unterschiedlich lang sind und den Symbolen, die häufig vorkommen, kurze Codes zugewiesen werden. Die durchschnittliche Länge der Codes ist damit kleiner als die durchschnittliche Größe der Symbole in den Eingabedaten. Das Verfahren arbeitet intern mit einem Binärbaum, der von den Blätter zur Wurzel hin aufgebaut wird. In ihm sind alle Symbole des Alphabets in den Blättern abgespeichert, jeder Kante des Baumes sind Eins- oder Null-Bits zugeordnet, so daß der Weg von der Wurzel zum Blatt den resultierenden Code ergibt. Je länger der Weg ist, desto weniger häufig tritt das Symbol in der Eingabe auf und desto länger ist auch der resultierende Code. Die Komprimierung von Daten mit Hilfe des Huffman-Kodierers ist immer dann sinnvoll, wenn die Anzahl der einzelnen Symbole nicht gleichverteilt ist. Bei Gleichverteilung werden in den komprimierten Daten kurze Codes genauso häufig benutzt wie die langen Codes. Im Mittel gleichen sich diese unterschiedlichen Längen wieder aus, so daß keine hohe Komprimierungsrate erzielt werden kann.

Bei den hier verwendeten Daten besteht das Alphabet aus ein bzw. zwei Byte langen Symbolen. Es ist dabei nicht von einer gleichmäßigen Verteilung der Häufigkeiten auszugehen, da es zum einem bei den Bitmasken immer wiederkehrende Muster geben wird und zum anderen die Koeffizienten nicht gleichverteilt sind sondern in ihrer Häufigkeit um den Nullpunkt herum zunehmen. Problematisch können die ggf. vorhandenen Null-Byte-Folgen sein. Diesen wird durch ihre hohe Anzahl zwar ein kurzer Code zugewiesen, der aber trotzdem in den Ausgabedaten entsprechend oft wiederholt werden muß. Es ist also zwingend notwendig, daß vor der Anwendung des Huffman-Kodierers ein *Run Length Encoding* durchgeführt wird, der eine lange Folge gleicher Symbole durch ein Wertetupel (Symbol, Anzahl) ersetzt.

Der Aufwand für die Huffman-Kodierung hängt im wesentlichen linear von Größe der Daten ab. Die Daten müssen für die Kodierung zweimal durchlaufen werden. Im ersten Durchlauf werden die Häufigkeiten bestimmt, im zweiten Durchlauf werden die Symbole durch die entsprechenden Codes ersetzt.

Eine Weiterentwicklung des Huffman-Kodierers ist der *Arithmetische Kodierer*, der im wesentlichen die Symbole der Eingabe zu Gruppen zusammenfaßt und diesen jeweils eine Gleitkommazahl zwischen 0.0 und 1.0 zugeordnet wird. Durch eine geschickte Auswahl der Gleitkommazahl kann erreicht werden, daß die durchschnittliche Länge der Codes geringer ist als deren Länge beim Huffman-Kodierer. Die Implementierung des Arithmetischen Kodierers ist komplizierter und bringt nur dann Vorteile, wenn die Häufigkeit der einzelnen Symbolen nicht wie in dem für den Huffman-Kodierer optimalen Fall um den Faktor $1/2$ abnimmt.

Substituierende Komprimierungsverfahren

Als substituierende Komprimierungsverfahren werden solche Verfahren bezeichnet, die wiederkehrende Symbolfolgen im Eingabestrom durch einen Verweis auf ihr erstes Vorkommen ersetzen. Diese Verfahren sind einfach, effizient und generell für beliebige Eingabedaten einsetzbar. Die meisten derzeit bekannten Verfahren dieser Art beruhen auf dem Lempel-Ziv Algorithmus, der unter dem Namen LZ77 [49] bekannt ist. Es gibt eine ganze Familie von darauf aufbauenden Komprimierungsalgorithmen, die in vielen bekannten Komprimierungstools eingebaut sind.

Der LZ77-Kodierer arbeitet mit einem Sliding Window, das dazu führt, daß nur in den letzten m vorangegangenen Symbolen nach einem Vorkommen der aktuellen Symbolfolge gesucht wird. Typische Werte für m liegen dabei zwischen 2000 und 16000 Bytes. Es wird versucht, Wiederholungen von möglichst langen Symbolfolgen zu finden und durch Referenzen zu ersetzen. Als Referenz wird ein Tupel eingesetzt, das zum einem die Position des ersten Vorkommens und zum

anderen die Länge der Symbolfolgen enthält. Der Aufwand für die Komprimierung hängt nicht nur von der Größe der Daten, sondern auch von der Größe des Sliding Window ab. Jede Symbolfolge muß mit allen möglichen im Sliding Window enthaltenen Symbolfolgen verglichen werden. Eine möglichst effiziente Speicherung der im Sliding Window enthaltenen Symbolfolgen ist sehr wichtig (z.B. Suchbäume). Ein bekanntes Unix-Tool, das diesen Algorithmus benutzt, ist *gzip*. Bekannte Weiterentwicklungen sind der *LZ78* und *LZW* -Varianten, die jeweils Dictionaries für die Verwaltung und Übertragung von bekannten Symbolfolgen benutzen.

Auch die oben schon angesprochene Verfahren des *Run Length Encoding* kann als ein substituierendes Komprimierungsverfahren angesehen werden, da Folgen gleicher Symbole durch nur ein Symbol und dessen Wiederholungsfaktor ersetzt werden.

3.4 Sonderfälle und Erweiterungen

Beim Einsatz der Wavelet-Transformation bei der Ankopplung an ein Simulationsprogramm fällt eine Voraussetzung der schnellen Wavelet-Transformation als sehr störend auf: Durch die in jeder Iteration des Pyramidenalgorithmus vorgenommen Halbierung der Seitenlänge entsteht implizit die Forderung, daß die Kantenlänge eine Potenz von zwei sein muß (TwoUp-Kantenlänge). Ist dies nicht der Fall, müssen vor oder während der Anwendung des Pyramidenalgorithmus Modifikationen an der Größe der Simulationsdaten vorgenommen werden. Im nächsten Teilabschnitt werden dazu einige Strategien vorgestellt.

In dem bisher vorgestellten Anwendungsszenario können Simulationsdaten, die typischerweise zwei- oder dreidimensional sind, Wavelet-transformiert, quantisiert und effektiv für den Transport zur Visualisierungs-Workstation komprimiert werden. Durch die Trennung der Wavelet-transformierten Daten in einzelne Frequenzbänder ist es sogar möglich, die Auflösung der Daten zu steuern, so daß sie nicht nur an die Leistungsfähigkeit des Verbindungsnetzes, sondern auch an die Leistungsfähigkeit der Visualisierungs-Workstation angepaßt werden können. Störend dabei ist, das die Auflösung von Frequenzband zu Frequenzband nicht in einem moderaten Maß zunimmt. Geht man z.B. von der Darstellung eines Volumens (3D) aus, steigt die Anzahl der darzustellenden Voxel jeweils um den Faktor 2^3. Eine feinere Abstufung wäre hier wünschenswert, damit die Auflösung besser an die Leistungsfähigkeit der Visualisierungs-Workstation angepaßt werden kann. Als ein Ansatz dazu wird im übernächsten Teilabschnitt die Octree-Zerlegung vorgestellt.

3.4.1 Behandlung von nicht-*TwoUp*-Kantenlängen

Es wird im allgemeinen nicht so sein, daß in Simulationsrechnungen die Kantenlängen der Simulationsgebiete eine Potenz von zwei sind. Dies kann z.B. durch algorithmische Randbedingungen oder Einschränkungen bei der Implementierung oder Optimierung des Simulationsprogramms vorgegeben sein. Zum Beispiel sollten für einen schnellen Zugriff auf den Hauptspeicher sogenannte *Strides* vermieden werden. Diese Zugriffe auf Daten, die im Hauptspeicher den Abstand einer Zweierpotenz besitzen, führen bedingt durch die Aufteilung des Speichers in Memory-Bänke auf vielen Plattformen zu zusätzlichen Wartezyklen.

Beim Computational Steering ist es im allgemeinen nicht notwendig, daß beim Pyramidenalgorithmus alle Iterationen durchlaufen werden, bis im niedrigsten Frequenzband nur noch ein Datenelement übrig bleibt. Vielmehr kann der Pyramidenalgorithmus abgebrochen werden, wenn die verbleibenden Daten eine genügend kleine Größe für den Transport und die Anzeige besitzen. Dann werden nur noch m Iterationen des Pyramidenalgorithmus ausgeführt, was dazu führt, daß die Kantenlängen jeweils auch nur noch m-mal durch zwei teilbar sein müssen. Diese Anforderung an die Datengröße kann eher zutreffen. Dennoch sollte auch die Behandlung von Daten möglich sein, deren Kantenlänge nicht dieser Bedingung genügt.

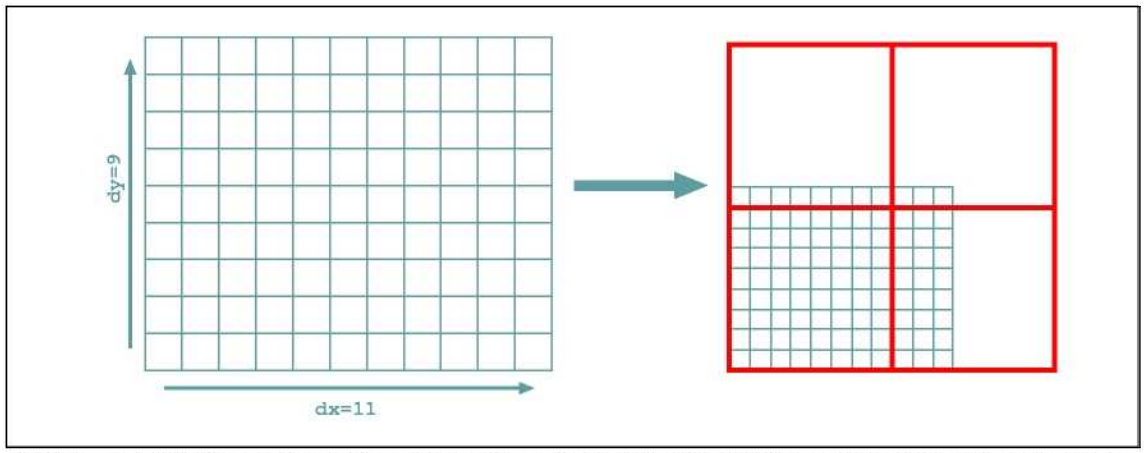


Abbildung 3.19: Umkopieren eines Datensatzes in ein TwoUp-Gebiet: Schematische Darstellung der Einlagerung eines Bildes mit ungünstigen Kantenlängen in ein Bild mit für die FWT günstigen Ausmaßen. Bei diesem Beispiel ist nach dem Auffüllen nur etwas mehr als ein Viertel des Bildes mit „Nutzdaten" gefüllt. Die restlichen Bereiche bleiben leer, müssen aber bei der Wavelet-Transformation mit berücksichtigt werden.

Ein offensichtliche Lösung des Problems ist, die Daten in ein entsprechend größeres Gebiet mit einer den Anforderungen genügenden Kantenlänge zu kopieren und die dort unbenutzten Bereiche mit Null-Werten aufzufüllen. Dieses Gebiet kann entsprechend oft vom Pyramidenalgorithmus verkleinert werden. Wird zusätzlich das höchste Frequenzband nur für die Transformation benutzt und nicht übertragen, entsteht keine höhere Anforderung an die Leistungsfähigkeit des Verbindungsnetzes und der Visualisierungs-Workstation. Abbildung 3.19 zeigt einen sehr ungünstigen Fall für dieses Umkopieren. Größter Nachteil ist dabei nicht der durch das größere Gebiet höhere Aufwand bei der Wavelet-Transformation, sondern die Tatsache, daß für die Kopie weiterer Hauptspeicher benötigt wird. Dies kann bei Simulationsprogrammen, die selbst schon für die Speicherung ihrer Daten die Verteilung auf mehrere Prozessoren in Kauf nehmen, einen erheblichen Engpaß darstellen.

Die Größe des zusätzlichen mit Null-Werten aufgefüllten Bereichs kann wie folgt abgeschätzt werden:

$$n_z < (2^k - 1)^m * (2^m - 1) \tag{3.40}$$

Dabei ist k die Anzahl der Iterationen des Pyramidenalgorithmus und m die Dimension der Daten. Der erste Term $(2^k - 1)$ ergibt die maximale Anzahl von Null-Werten, die in einer Dimension hinzugenommen werden müssen. Potenziert mit m ergibt dies die Anzahl für alle Dimensionen. Bei einem zweidimensionalen Gebiet ist dies ein Quadrat, das nun an drei Seiten an die Originaldaten angefügt werden muß. Der zweite Term beschreibt das mehrmalige Anfügen mit Null-Daten.

Das Umkopieren in einen größeren Bereich kann durch eine Modifikation im Verfahren der FWT vermieden werden. Im Pyramidenalgorithmus wird die Wavelet-Transformation jeweils nacheinander für jede Dimension ausgeführt. Sie arbeitet dabei jeweils auf eindimensionalen Datenvektoren, die aus den Zeilen bzw. Spalten der mehrdimensionalen Daten gebildet werden. Statt die mehrdimensionalen Daten zu kopieren und zu vergrößern, reicht es aus, die eindimensionalen Datenvektoren zu vergrößern. In den meisten Implementierungen der FWT werden zur Vermeidung von aufwendigen Indexberechnungen die Daten vor der Anwendung der Filter aus den mehrdimensionalen Daten herauskopiert und die transformierten Daten wieder ins Original hineinkopiert. Die Erweiterung des eindimensionalen Datenvektors fällt nun bei der Betrachtung des Speicherverbrauchs nicht mehr ins Gewicht. Beim Zurückkopieren werden nur die Ergebnisdaten des Tiefpasses berücksichtigt, so daß damit implizit das höchste Frequenzband weggelassen wird und die Daten wieder in den Original-Speicherbereich passen.

Als Nachteil bleibt weiterhin der Mehraufwand bei der Wavelet-Transformation sowie die zusätzliche komplizierte Verwaltung der Datengrößen und die Erweiterung des Pyramidenalgorithmus um den Sonderfall in der ersten Iteration.

Statt der Erweiterung des Simulationsgebiets auf ein TwoUp-Gebiet kann auch nur ein Ausschnitt mit einem entsprechend kleinen TwoUp-Gebiet zur Visualisierung übertragen werden. Hier besteht jedoch die Gefahr, daß in den nicht übertragenen Randgebieten wichtige Informationen vorhanden sind. Wenn bei der Anbindung der Visualisierung an das Simulationsprogramm eine zur Simulationsrechnung asynchrone Interaktion möglich ist, kann zur Vermeidung dieses Risikos z.B. eine interaktive „Lupe“ eingebaut werden, die es erlaubt, den Ausschnitt über das Simulationsgebiet zu verschieben. Nachteil dabei ist, daß die Daten mehrmals transformiert und übertragen werden müssen.

3.4.2 Octree-Zerlegung

Durch die Zerlegung der Daten in Frequenzbänder wird zwar eine auflösungsabhängige Darstellung möglich. Die schnelle Wavelet-Transformation bedingt aber, daß die Auflösung von Frequenzband zu Frequenzband in zu großen Schritten ansteigt (z.B. Faktor 8 bei Volumendaten). Eine Betrachtung von solchen Daten ist insbesondere dann schwierig, wenn die Originaldaten nur mit wenigen von Null verschiedenen Werten besetzt sind. Für die Darstellung müssen dann auch die hohen Frequenzbänder herangezogen werden, was auf der Visualisierungsseite wieder zu sehr großen Speicheranforderungen führt. Für den Transport können die Daten zwar durch die Extraktion der wichtigen Koeffizienten effektiv gespeichert werden. Für die Anzeige müssen die Daten wieder rekonstruiert und mit hochaufgelösten Gittern dargestellt werden. Dabei sind in den rekonstruierten Daten viele Null-Werte vorhanden, die keinen Beitrag zur Darstellung liefern.

Die Anzeige solcher Daten könnte dadurch vereinfacht werden, daß innerhalb der Visualisierung die Speicherung der Null-Werte vermieden wird. Ein Verfahren, das sich bei der Darstellung von Volumina dazu anbietet, ist die *Octree*-Zerlegung. Dabei wird das Volumen rekursiv in immer kleinere Teilblöcke zerlegt, die dann getrennt voneinander übertragen und angezeigt werden können. Blöcke, die keine relevanten Daten enthalten, brauchen dabei nicht übertragen werden und verbrauchen dadurch auch keinen Speicherplatz innerhalb der Visualisierung.

Bei der Octree-Zerlegung wird ein Volumen in mehreren Iterationen in kleinere Teilblöcke zerlegt. Dazu wird in jeder Iteration jeder Block in acht Teilblöcke zerlegt (siehe Abb. 3.20 auf der nächsten Seite). Die Teilblöcke stehen in einer hierarchischen Beziehung, die für ihre Speicherung ausgenutzt werden kann. Der dabei entstehende Baum enthält in jeder Ebene Teilblöcke der gleichen Auflösungsstufe. Bei der Zerlegung ist es nicht notwendig, daß diese solange fortgeführt wird, bis Teilblöcke mit der Kantenlänge eins entstanden sind. Vielmehr kann die Zerlegung dann abgebrochen werden, wenn die Teilblöcke eine genügend kleine Größe für die Visualisierung erreicht haben.

Diese Zerlegung ähnelt sehr stark der Zerlegung mit Hilfe der schnellen Wavelet-Transformation. Im Gegensatz zu dieser können hier die Teilblöcke unterschiedliche Größen besitzen. Damit ist auch eine Zerlegung von Volumina möglich, deren Kantenlängen keine Zweierpotenz ist. Außerdem kann die Entscheidung über eine weitere Zerlegung für jeden Teilblock einzeln getroffen werden, so daß sie z.B. von dem Informationsgehalt des Teilblocks abhängig gemacht werden kann. Man erreicht mit dieser Art der Zerlegung einen adaptiven und vom Informationsgehalt abhängigen Speicherbedarf, wenn die Auflösung der Teilblöcke mit der Entfernung vom Wurzelknoten der Baumstruktur zunimmt. Das führt dazu, daß Blöcke, die wegen eines zu geringen Informationsgehalts nicht mehr weiter zerlegt worden sind und sich damit in einer höheren Ebene des Baumes befinden, in einer groberen Auflösung gespeichert werden. Die interessanten Informationen steckt nach der Zerlegung in den feiner aufgelösten unteren Ebenen der Baumstruktur (siehe Abb. 3.21).

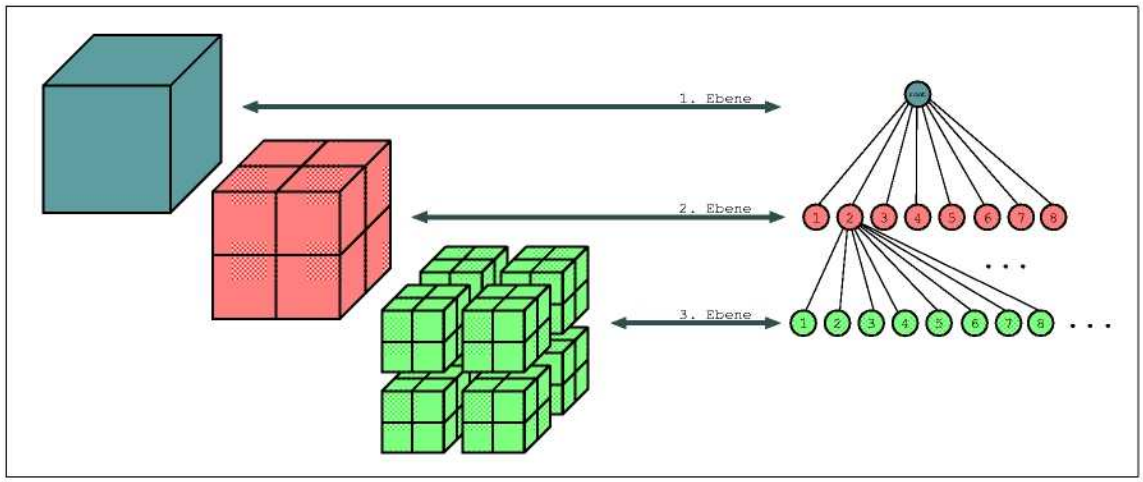


Abbildung 3.20: Octree-Zerlegung: Die einzelnen Teilblöcke können in eine hierarchische Beziehung gebracht werden. Dabei entsteht ein Baum, dessen Ebenen jeweils Teilblöcke einer Größe beschreiben.

Im Prinzip ist für die Generierung der Octree-Zerlegung eines Volumens auch die Anwendung des Pyramidenalgorithmus möglich. Beginnend mit dem niedrigsten Frequenzband können die Koeffizienten des nächst höheren Frequenzbandes umsortiert und zu einer Baumstruktur zusammengefügt werden. Führt man die Sortierung rekursiv für alle Ebenen durch, erhält man eine der Octree-Zerlegung entsprechende Baumstruktur. Ein solcher Aufbau des Octrees ist aber aus mehreren Gründen nicht sinnvoll. Zum einen wird auf der Seite der Simulation das Gesamtgebiet mit einer FWT transformiert. Da aber auf der Visualisierungsseite eine Menge von Teilblöcken unterschiedlicher Auflösung vorliegen, müssen diese dort jeweils einzeln mit einer FWT rekonstruiert werden. Probleme bereiten hier die Randelemente der Teilblöcke, die durch die Transformation des Gesamtgebiets auch Anteile von Elementen benachbarter Teilblöcke enthalten. Für eine Octree-Generierung mit Hilfe des Pyramidenalgorithmus müßte dieser bis zur letzten Iteration durchlaufen werden, damit eine Wurzel des Baumes generiert werden kann. Das wiederum erfordert, daß die Kantenlängen des Volumen eine Zweierpotenz sind. Ein entscheidender Vorteil der Octree-Zerlegung ist damit aufgehoben.

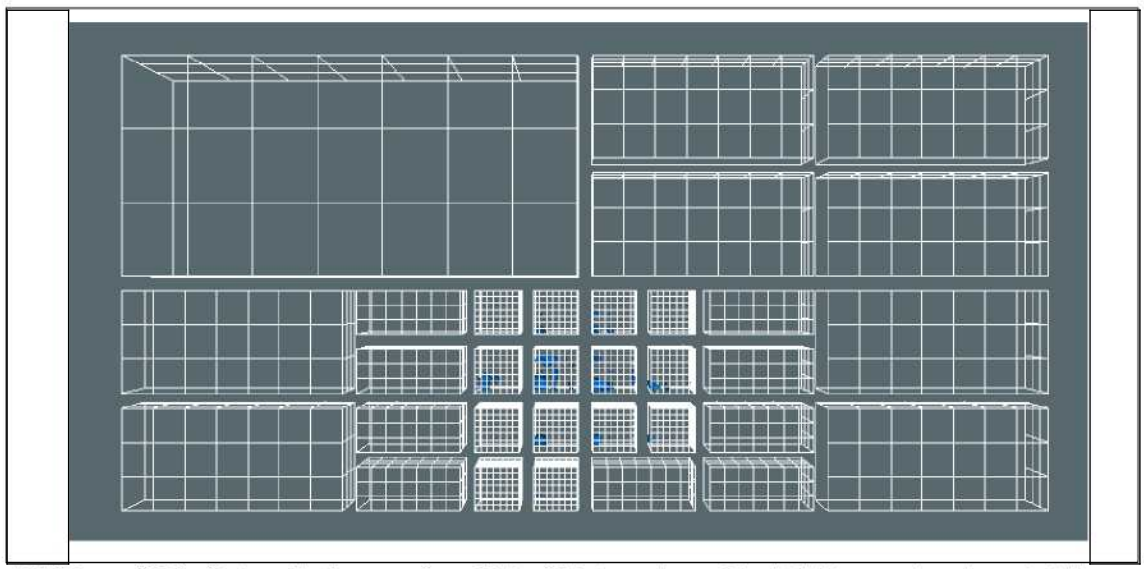


Abbildung 3.21: Octree-Zerlegung eines Beispieldatensatzes: Die Abbildung zeigt einen dreidimensionaler Datensatz, der ein von Partrace berechnetes Konzentrationfeld enthält und in eine Octree-Baumstruktur zerlegt worden ist. Die wesentliche Information befindet sich im mittleren Bereich, die äußeren Bereiche bestehen jeweils aus großen und gering aufgelösten Teilblöcken. Zur Mitte hin werden die Teilblöcke kleiner und besitzen eine höhere Auflösung.

Durch die Aufteilung des Volumens in Teilblöcke ist auch eine progressive Übertragung der Daten möglich, wobei die Übertragungsreihenfolge durch den Informationsgehalt der Teilblöcke bestimmt werden kann. Das heißt, daß die Reihenfolge aus der Baumstruktur abgeleitet werden kann, so daß die wichtigsten Blöcke mit der höchsten Auflösung aus der untersten Ebene des Baumes zuerst entnommen werden. Auch ist eine Benutzer-Interaktion möglich, bei der nur die vom Benutzer ausgewählten Teilblöcke übertragen und angezeigt werden. Zur Bestimmung des Informationsgehalts der einzelnen Teilblöcke kann z.B. die L_2-Norm herangezogen werden.

Als ein Nachteil der Octree-Methode ist sicherlich der höhere Verwaltungsaufwand durch die Baumstruktur und die Aufteilung in Teilblöcke zu nennen. Durch die unterschiedliche Auflösung der Teilblöcke ergibt sich aber auch eine zusätzliche Anforderung an das Visualisierungsprogramm. Dies muß nämlich in der Lage sein, eine durch die Teilblöcke gebildete Liste von Teilgittern mit unterschiedlichen Auflösungen darzustellen. Durch die unterschiedliche Auflösung und Größe der Blöcke können diese nicht mehr zu einem uniformen Gitter zusammengefügt werden und müssen daher vom Visualisierungsprogramm getrennt voneinander behandelt werden. Schwierig wird dann die Anzeige mit Hilfe von globalen abgeleiteten Darstellungsmethoden (z.B. Iso-Oberflächen), da an den Stoßkanten der einzelnen Teilblöcke unterschiedliche Auflösungen aufeinandertreffen. AVS/Express verfügt über derartige Möglichkeiten [50].

Durch die Octree-Zerlegung erreicht man zwar eine vom Informationsgehalt abhängige Steuerung der Auflösung. Eine zusätzliche Komprimierung ist aber insbesondere für die im unteren Bereich der Baumstruktur angesiedelten hochaufgelösten Teilblöcke nötig. Es bietet sich dort wiederum das in dieser Arbeit vorgestellte Verfahren der Wavelet-Transformation mit anschließender Quantisierung und Kodierung an [51]. Dabei wird die Wavelet-Transformation auf jedem Teilblock einzeln ausgeführt. Da es sich nur um Teilblöcke handelt, müssen nur zwei bis drei Iterationen des Pyramidenalgorithmus durchgeführt werden. Die resultierenden Frequenzbänder können nun pro Teilblock gemeinsam oder zusätzlich zur progressiven Übertragung der Teilblöcke nach und nach zur Visualisierung übertragen werden. Der Informationsgehalt der einzelnen Teilblöcke ist weiterhin mit Hilfe der L_2-Norm bestimmbar, da die Wavelet-Transformation energieerhaltend ist. Die Reduktion der Auflösung bei höher im Baum angesiedelten Teilblöcken kann nun durch das Weglassen des jeweils höchsten Frequenzbandes erfolgen. Die L_2-Norm der einzelnen Frequenzbänder kann dabei einen Aufschluß über die Anzahl der wegzulassenden Frequenzbänder liefern.

Kapitel 4

Modellierung der Kopplung

Mit der Wavelet-Transformation und anschließenden Quantisierung und Kodierung wurde in vorherigen Kapitel eine Komprimierungmethode vorgestellt, die beim Computational Steering auch die Übertragung und Anzeige von großen Datensätzen ermöglichen soll. Ob eine solche neue, in den Übertragungsprozeß integrierte Komprimierungskomponente eine Zeitersparnis bringt, hängt von vielen Parametern ab. So wird der Datensatz durch die Komprimierung zwar verkleinert, aber auch für die Komprimierung selbst muß Rechenzeit erbracht werden. Für eine genauere Betrachtung müssen Rechenzeit und Zeitersparnis durch kleinere Datenmengen in Abhängigkeit von anderen Größen wie z.B. Datengröße oder Komprimierungsqualität gebracht werden. Ein so aufgestelltes Modell kann dann eine Voraussage darüber geben, unter welchen Randbedingungen es sich lohnt, eine Komprimierung einzusetzen. Weitere solcher Fragen stellen sich insbesondere bei der Ankopplung einer Visualisierung an ein paralleles Simulationsprogramm.

Es gibt mehrere Faktoren, die das Modell der Ankopplung beeinflussen können. Der erste Faktor ist der Einsatz der oben beschrieben Komprimierungskomponente selbst. Weitere sind z.B. die Art der Ankopplung oder die Art und Häufigkeit der Übertragung. Bei einem parallelen Simulationsprogramm ist neben der üblichen Ankopplung über einen Master-Knoten auch die Ankopplung aller Rechnenknoten oder die Ankopplung über einen zusätzlichen, nur für die Übertragung zuständigen Knoten möglich. Durch die Zerlegung der Daten in verschiedene Frequenzbänder ist weiterhin auch die Möglichkeit gegeben, die Daten progressiv, also nach und nach zur Visualisierungsworkstation zu übertragen.

Die Modellierung gliedert sich in mehrere Abschnitte, die jeweils den Einfluß der oben beschriebenen Faktoren behandeln. Die dabei entwickelten Modelle werden im nächsten Kapitel mit der in dieser Arbeit implementierten Komprimierungkomponente unter verschiedenen Kopplungsstrategien überprüft.

4.1 Definitionen der Modellparameter

Die einzelnen Schritte der Verarbeitung werden im Modell als Komponenten bzw. als Knoten eines Netzwerks betrachtet. Die Knoten können Daten empfangen, diese verarbeiten und über die Kanten an andere Knoten weiter versenden. Sowohl in den Knoten als auch an den sie verbindenden Kanten können Wartezeiten entstehen, die im wesentlichen durch die Übertragung und Verarbeitung der Daten begründet sind. Für die Bewertung der Modelle spielen gerade diese Zeiten eine wichtige Rolle, da für die Kopplung immer diejenige Strategie ausgewählt werden soll, welche die schnellsten Kommunikationsablauf ergibt und die Simulationsrechnung am wenigsten behindert.

Das Modell sollte also die für die Kopplung benötigte Zeit in Beziehung zu den verschiedenen Randbedingungen setzen.

Ein Maß für die Bewertung ist also die Zeit, die für die Übertragung und Anzeige der Daten benötigt wird. Ein anderes Maß kann auch die Zeit sein, um die das Simulationsprogramm durch die Ankopplung verlangsamt wird. Häufig wird hier dafür der Anteil der Kommunikationszeit an der gesamten Rechenzeit betrachtet. Je kleiner dieser Anteil ist, desto effizienter arbeitet das Simulationsprogramm.

Bei parallelen Simulationsprogrammen hängt deren Beeinflussung von der Art der Ankopplung ab. Hier werden als Maß häufig der Speedup oder die Effizienz des parallelen Programms angewandt. Der Speedup gibt dabei an, um welchen Faktor ein paralleles Programm schneller läuft als das entsprechende sequentielle Programm. Als Grenze für diesen Wert ist hier ein linearer Speedup zu sehen, der nur dann erreichbar ist, wenn die Arbeit optimal verteilt und jegliche Kommunikation vermieden werden kann. Die Effizienz beschreibt, zu welchem durchschnittlichen Anteil die Prozessoren mit Rechenarbeit beschäftigt sind und wird durch den Quotienten aus Speedup und Prozessoranzahl definiert.

Simulationsprogramme, bei denen eine Online-Visualisierung und Steuerung sinnvoll ist, rechnen meistens auf Basis von Simulationsschritten, mit denen die Simulationszeit diskretisiert wird. Eine Übertragung der Daten ist dabei erst nach der Beendigung eines Simulationsschritts möglich. Auf der Seite der Simulation entsteht dadurch eine Abfolge von zwei Teilschritten (Berechnung und Versenden der Daten). Die Bewertung eines Modells betrachtet jeweils einen Zeitschritt und geht dabei bei den Parametern von Durchschnittwerten für alle Zeitschritte aus.

In den einzelnen Modellen werden Konstanten und Symbole benutzt, die an dieser Stelle zusammenhängend aufgeführt werden sollen:

s	Größe eines Datenpakets, das zum Visualisierungsrechner übertragen wird,
n	Anzahl der in einem Datenpaket enthaltenen Zahlen,
b	Bandbreite bzw. Übertragungskapazität einer Verbindungskante,
t_l	Latenz, *StartUp*-Zeit einer Übertragung,
t_A	Verarbeitungszeit für die Berechnung im Knoten A,
t_a	Kommunikationszeit für die Datenübertragung über die Verbindungskante a,
t_v	Visualisierungszeit zur Darstellung von s,
T_S	durchschnittlicher Zeitaufwand für einen Simulationsschritt eines sequentiellen Simulationsprogramms,
$T_\mathrm{S,v}$	durchschnittlicher Zeitaufwand für einen Simulationsschritt eines sequentiellen Simulationsprogramms mit angekoppelter Visualisierung,
T_P	durchschnittlicher Zeitaufwand für einen Simulationsschritt eines parallelen Simulationsprogramms,
$T_\mathrm{P,v}$	durchschnittlicher Zeitaufwand für einen Simulationsschritt eines parallelen Simulationsprogramms mit angekoppelter Visualisierung,
T_K	durchschnittliche Kommunikationszeit in einem Simulationsschritt: $T_K = T_\mathrm{S,v} - T_\mathrm{S}$,
N	Anzahl der Prozessoren eines parallelen Simulationsprogramms,
$S(N)$	Speedup bei einem parallelen Programm. Quotient aus der Laufzeit des seriellen und des parallelen Programms. Optimal ist ein linearer Speedup ($S(N) = N$),
$E(N)$	Effizienz der Parallelisierung: $E(N) = S(N)/N$

4.2 Modell für die direkte Übertragung

Als erstes Modell wird in diesem Abschnitt der einfachste Fall für eine Kopplung zwischen einer Visualisierung und einer Simulation vorgestellt. Dabei wird davon ausgegangen, daß die Simulation sequentiell, also auf einem Ein-Prozessor-Rechner abläuft und daß sich Simulation und Visualisierung auf getrennten Rechnern befinden. Die Daten werden von der Simulation direkt, also ohne eine weitere Konvertierung zum Visualisierungsrechner übertragen. Das dem Modell zugrunde liegende Netzwerk besteht damit aus nur zwei Knoten, nämlich der Simulation und der Visualisierung (siehe Abb. 4.1). Im einfachsten Fall kann davon ausgegangen werden, daß sowohl für das Verschicken als auch für das Empfangen der Daten in den entsprechenden Knoten keine zusätzlichen Verarbeitungszeiten entstehen. Auf der Simulationsseite befinden sich die Daten typischerweise in einem Speicherbereich, der direkt an die Versende-Routinen übergeben werden kann. Auf der Visualisierungsseite muß für das Empfangen der Daten ein entsprechend großer Speicherbereich zur Verfügung gestellt werden, der durch die Empfangsroutinen gefüllt wird.

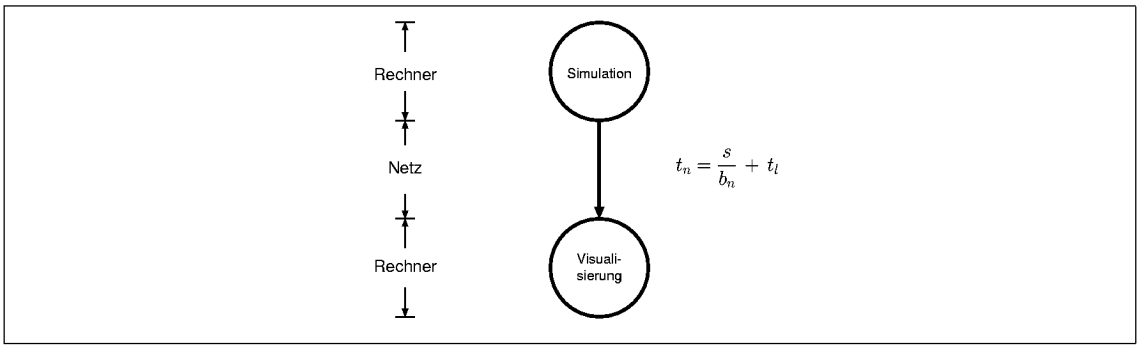


Abbildung 4.1: Netzstruktur bei direkter Übertragung: Die einzelnen Komponenten des Übertragungsnetzes werden als Knoten dargestellt, die mit Kanten untereinander verbunden sind. In diesem Beispiel entstehen nur an der einen Verbindungskante durch die Übertragung der Daten eine Wartezeiten.

Die Zeitdauer der Übertragung wird durch die Leistungfähigkeit des dazwischen liegenden Netzwerks bestimmt. Kenndaten des Netzwerks sind dabei die Bandbreite (b_n) und die Latenz (t_l). Die Bandbreite beschreibt die Menge von Daten, die in einer bestimmten Zeiteinheit übertragen werden können. Typischerweise werden dabei die Einheiten MB/s oder MBit/s verwendet. Geht man davon aus, daß Simulation und Visualisierung über TCP/IP und FastEthernet verbunden sind, liegt die Bandbreite im Bereich von 5-10 MB/s. Wenn man für die Größe s der zu übertragenden Daten ein Wert im Bereich von 0.1-100 MB annimmt, kommen Übertragungszeiten im Bereich von mehreren Sekunden zustande. Die Latenz ist ein Maß dafür, wie lange die Initialisierung einer Übertragung dauert. Dies ist die Zeit, die vom Start der Übertragung auf der Sendeseite bis zu dem Zeitpunkt vergeht, zu dem das erste Byte auf der Empfängerseite vorliegt. Gemessen wird die Latenz häufig mit der Übertragung einer leeren Nachricht. Somit ergibt sich für die Übertragungzeit:

$$t_n = \frac{s}{b_n} + t_l \tag{4.1}$$

Hierbei wird das Netzwerk zwischen den Knoten als eine Verbindungsstrecke betrachtet, für die Bandbreite und Latenz bekannt sind. Besteht die Strecke – wie im Normalfall üblich – aus mehreren Segmenten, die durch Router oder Switches miteinander verbunden sind, berechnet sich die Bandbreite als das Minimum der Bandbreiten der Teilstrecken und die Latenz als die Summe der Latenzen in den Teilstrecken, den Switches und Routern. Auch der Fall, daß die Strecke nicht dediziert für die Anwendung des Computational Steering zur Verfügung steht, beeinflußt die Bandbreite und die Latenz. Im Modell wird dies aber nicht berücksichtigt.

Die obige Abschätzung (4.1) zeigt, daß typische Übertragungszeiten für große Datenmengen im Computational Steering im Bereich von mehreren Sekunden liegen. Der Anteil der Zeit, die durch

die Latenz bedingt wird, liegt dabei bei Entfernungen unter 1000 km im Millisekundenbereich und ergibt daher nur einen relativ geringen Einfluß auf das Modell. Die Latenz ist zudem nicht entscheidend für die Modellierung, da die Übertragung insbesondere für große Datenmengen von der Simulation in Richtung Visualisierung geht und eher einen asynchronen Charakter besitzt.

Für den Vergleich der verschiedenen Modelle soll in erster Linie die Zeit in Betracht gezogen werden, um die die Simulation mit angekoppelter Visualisierung langsamer läuft als ohne Ankopplung. Diese durch die Kommunikation mit der Visualisierung bedingte und mit T_K bezeichnete Zeit kann für die direkte Kopplung wie folgt bestimmt werden:

$$T_{K,\text{direkt}} = t_n = \frac{s}{b_n} + t_l \tag{4.2}$$

Im einfachsten Fall wird durch die Übertragung der Daten zur Visualisierung das Simulationsprogramm pro Simulationsschritt für die Zeitdauer T_K blockiert (siehe Abb. 4.2). Geht man von einer durchschnittlichen Rechenzeit von T_S für einen Simulationsschritt ohne Ankopplung aus, berechnet sich der Anteil der für die Simulationrechnung genutzten Zeit (Effizienz) an der Gesamtzeit zu:

$$E_{\text{direkt}} = \frac{T_S}{T_{\text{S,v}}} = \frac{T_S}{T_S + T_{K,\text{direkt}}} \tag{4.3}$$

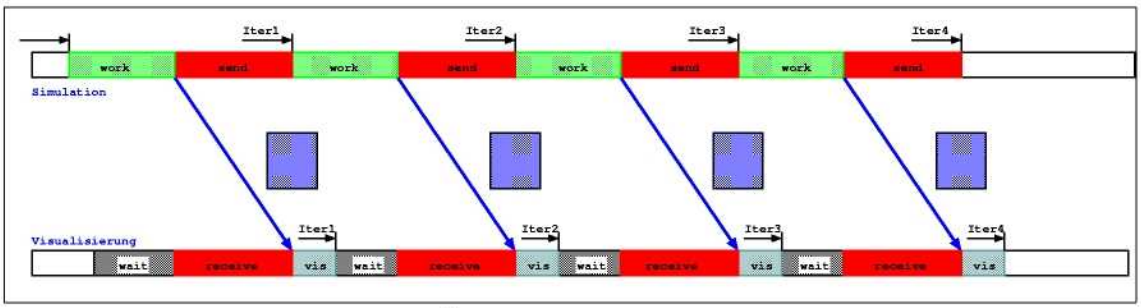


Abbildung 4.2: Beispiel für die direkte Übertragung: Die beiden Balken stellen den Zustand der Komponenten Simulation (oben) und Visualisierung (unten) dar. Die diagonalen Verbindungslinien zwischen den beiden Balken markieren die versendeten Datenpakete. In diesem Beispiel befindet sich die Simulation zu einem größeren Teil im Zustand *send*. Die Simulation wird also durch die Ankopplung der Visualisierung stark behindert.

Berücksichtigung der begrenzten Größe des Socket-Buffers im Modell

Bei der in dieser Arbeit verwendeten Kopplungsbibliothek VISIT wird die Übertragung der Daten mit Hilfe von TCP/IP-Sockets (Stream) realisiert. Dabei werden vom Betriebssystem jeweils für das Versenden und Empfangen von Daten sogenannte Socket-Buffer bereitgestellt. Beim Versenden werden die Daten beim Aufruf der Send-Funktion zuerst dort abgelegt und dann vom Betriebssystem zum Zielrechner übertragen. Gleiches gilt auf der Empfängerseite: Die Daten werden vom Betriebssystem zuerst in den Buffer abgelegt. Eine vom Anwenderprogramm aufgerufene Receive-Funktion kopiert die Daten dann vom Buffer in den Programm-eigenen Speicher. Die Größe der Buffer (s_{SO}) ist begrenzt und liegt typischerweise bei maximal 1 MB. Daher kann es auf beiden Seiten zu Blockierungen kommen, falls die zu versendenden Daten zu groß sind oder mehrere Datenpakete direkt hintereinander versendet werden.

Ist z.B. der Buffer auf der Empfängerseite zu klein oder voll, muß der Sender solange warten, bis dieser geleert worden ist. Erst dann können neue Daten übertragen werden. Ist auf der Sendeseite der Buffer noch gefüllt oder zu klein, muß die Send-Funktion solange warten, bis der Buffer geleert

ist. Wenn die Daten größer als der Socket-Buffer sind, werden die Daten stückweise in den Buffer kopiert und versendet.

Dieses Verhalten ist nur dann bei der Modellierung zu berücksichtigen, wenn nur kleine Datenpakete übertragen werden und die durchschnittliche Simulationsschrittlänge T_S groß genug ist. Dann ist zu erwarten, daß die Send-Funktion die Daten direkt in den Buffer kopieren kann und keine Wartezeiten im Simulationsprogramm erzeugt. Die Übertragung der Daten wird dann vom Betriebssystem im Hintergrund erledigt. Wenn die Zeitdauer zwischen zwei Übertragungen lang genug ist, ist der Buffer beim nächsten Aufruf der Send-Funktion geleert. In diesem Fall gilt:

$$T'_{K,\text{direkt}} = \begin{cases} 0 & \text{wenn } s \leq s_{SO} \\ \frac{s - s_{SO}}{b_n} + t_l & \text{wenn } s > s_{SO} \end{cases} \tag{4.4}$$

Auch beim Versenden großer Datenblöcke reduziert sich die Wartezeit des Simulationsprogramms ein wenig, da seine Sende-Funktion nicht auf das Versenden des letzten Teilblocks warten muß. Im Normalfall aber ist nicht zu erwarten, daß die zu visualisierenden Daten unmittelbar in den Socket-Buffer passen. Wenn $s \gg s_{SO}$ gilt, kann auf die Modellierung des Buffers verzichtet werden.

Asynchrone Übertragung

Eine Möglichkeit, die Wartezeit des Simulationsprogramms zu reduzieren, ist die asynchrone Übertragung der Daten. Dabei wird für die Übertragung eines Datenpakets nur dessen Speicherbereich der Sendefunktion übergeben und das Versenden nur angestoßen. Das Simulationsprogramm kann unmittelbar nach dem Start der Übertragung mit der Simulationrechnung fortfahren und muß nicht auf das Ende der Übertragung warten. Bevor neue Daten übertragen werden, muß sichergestellt sein, daß die vorherige asynchrone Übertragung abgeschlossen ist. Dazu kann das Ende der Übertragung entweder durch *polling* des Simulationsprogramms erfragt oder durch ein Signal dem Simulationsprogramms mitgeteilt werden.

Es gibt mehrere Gründe, die gegen den Einsatz einer solchen asynchronen Übertragung sprechen. Zum einen ist nicht bei allen Parallelrechnern die Möglichkeit gegeben, für die Socket-Kommunikation asynchronen I/O zu nutzen. Zum anderen müssen die zu versendenden Daten während der ganzen Übertragung in Programm-eigenen Speicher vorgehalten werden. Da sich Übertragung und Simulationsrechnung überlappen, kann der Speicher nicht für die Simulationsrechnung genutzt werden. Daher muß der Speicherbereich doppelt zur Verfügung stehen, einmal für die Simulationsrechnung, einmal für die Übertragung. Bei Simulationsprogrammen, die aus Speicherplatzgründen nur noch auf Parallelrechnern mit verteiltem Speicher laufen können, ist nicht zu erwarten, daß für eine solche doppelte Speicherung der Daten Platz vorhanden ist.

Eine Möglichkeit, die Übertragung trotzdem asynchron zu gestalten besteht darin, einen zusätzlichen Prozeß zu starten, der nur für die Kommunikation zuständig ist. Bei einem Parallelrechner würde dieser Prozeß auf einem zusätzlichen Rechenknoten ablaufen, bei einem Einzelprozessorsystem würden Simulations- und Kommunikationsprozeß auf demselben Rechner ablaufen. Das Simulationsprogramm müßte dann nur noch die Daten an den Kommunikationsprozeß übergeben und könnte danach mit der Rechnung fortfahren. Auf Einzelprozessorsystemen wird auch hier das Simulationsprogramm indirekt beeinflußt. Da die Arbeit beider Prozesse von einem Prozessor erledigt wird, muß nun das Betriebssystem dafür sorgen, daß beiden Prozessen abwechselnd der Prozessor zugeteilt wird (Scheduling). Dadurch wird sich die effektive Rechenzeit pro Simulationsschritt erhöhen. Da beide Prozesse sich auf demselben Rechner befinden, müssen sich beiden den Hauptspeicher teilen, was bei dem doppelten Speicherbedarf der asynchronen Übertragung zu Engpässen führen kann. Bei Parallelrechnern kann dagegen die Hinzunahme eines Kommuni-

kationsknotens sinnvoll sein. Im Abschnitt „Modell für die Übertragung über einen zusätzlichen Kommunikationsknoten“ 4.6.3 auf Seite 67 wird dazu ein Modell entwickelt.

Berücksichtigung der Verarbeitungszeiten in der Visualisierung

In der obigen Gleichungen 4.2 und 4.4 wurden davon ausgegangen, daß die Visualisierung jederzeit für den Empfang von Daten bereit steht. Dadurch, daß die Visualisierung auch interaktiv genutzt wird, muß dies aber nicht der Fall sein. Durch eine Prioritätensteuerung kann zwar teilweise erreicht werden, daß der Empfang von Daten bevorzugt und die interaktive Bearbeitung der visualisierten Daten während dieser Zeit angehalten wird.

Weiterhin dazu müssen die empfangenen Daten für die Anzeige aufbereitet werden. Insbesondere bei abgeleiteten Darstellungsarten wie z.B. den Iso-Oberflächen wird dazu eine nicht zu vernachlässigende Zeit benötigt. Bei kleinen Datenmengen kann das Problem durch den Socket-Buffer gelöst werden: Die Daten können vom Betriebssystem des Visualisierungsrechners angenommen werden, ohne daß das Visualisierungsprogramm dazu benötigt und das Simulationsprogramm als Sender der Daten blockiert wird. Bei großen Datenmengen ist dieser Vorteil wegen der zu geringen Größe der Socket-Buffer nicht nutzbar.

Daher ist im Modell die Bearbeitungszeit im Visualisierungsknoten zu berücksichtigen. Innerhalb der Zeitdauer eines Simulationsschritts müssen von dem Visualisierungsprogramm die Daten empfangen und verarbeitet werden. Dazu wird für das Empfangen die Zeitdauer t_n und für das Bearbeiten der Daten die Zeitdauer t_v benötigt. Zusätzlich kann noch eine Zeit t_i einfließen, welche die Zeit für nicht unterbrechbare interaktive Arbeit spezifiziert. Nur wenn die Summe aller drei Zeiten kleiner ist als die Zeit, die das Simulationsprogramm für die Berechnung und das Versenden der Daten eines Simulationsschritts benötigt, ist nicht mit einer zusätzlichen Wartezeit t_{delay} auf der Simulationsseite zu rechnen:

$$t_{\text{delay}} = \begin{cases} 0 & \text{wenn } t_v + t_i \leq T_S \\ t_v + t_i - T_S & \text{sonst} \end{cases} \tag{4.5}$$

Die Verarbeitungszeit t_v der empfangenen Daten ist u.a. von deren Größe abhängig. Sie hängt dabei sehr stark von der gewählten Darstellungsart ab. Im Fall der Schnittebene muß nur ein Teil der Daten betrachtet und visualisiert werden. Bei der diskreten Darstellung wird jedes Datenelement zu einem darstellbaren Objekt, so daß man hier eine lineare Abhängigkeit von den Datengröße erhält und die Verarbeitungszeit mit Hilfe einer Geschwindigkeits- bzw. Verarbeitungsbandbreite spezifizierbar. Bei den abgeleiteten Darstellungen, wie z.B. den Oberflächen oder Streamlines, kann die Datengröße die Verarbeitungszeit noch stärker beeinflussen.

In die für die Kopplung benötigte Zeit $T_{K,\text{direkt}}$ fließt die Wartezeit als zusätzlicher Summand ein:

$$T''_{K,\text{direkt}} = \frac{s}{b_n} + t_l + t_{\text{delay}} \tag{4.6}$$

Bei der in dieser Arbeit verwendeten Kopplungsbibliothek VISIT kann diese zusätzliche Wartezeit dadurch verhindert werden, daß in dem Fall, daß das Visualisierungsprogramm nicht für das Empfangen neuer Daten bereit ist, die Daten zu dem neuen Simulationsschritt nicht übertragen werden. Dazu wird in VISIT nach jeder Nachricht eine Bestätigung (ACK2) an den Sender der Daten zurückgeschickt, sobald die Visualisierung zur Entgegennahme neuer Daten bereit ist. Nur wenn diese Bestätigung auf der Sendeseite vorliegt, wird eine neue Nachricht versendet.

4.3 Modell für die Übertragung mit verlustfreier Komprimierung

Wesentliches Ziel der Modellbildung ist es, eine Voraussage über den Einfluß der Kopplung auf die Laufzeit der Simulationsrechnung zu ermöglichen. Durch den Einsatz der im ersten Teil der Arbeit beschriebenen Verfahren zur Komprimierung und Reduzierung der Daten mit Hilfe der Wavelet-Transformation soll dieser Einfluß verringert werden. In diesem Abschnitt soll zunächst ein Modell für eine einfache, verlustfreie Komprimierung entwickelt werden. Im folgenden Abschnitt wird dieses Modell für die nicht mehr verlustfreie Wavelet-Transformation verfeinert.

In diesem Szenario werden die Daten vor der Übertragung komprimiert und danach wieder entkomprimiert. Durch das Hinzufügen der neuen Komponenten wird keine Veränderung der Daten entstehen, da hier verlustfreie Komprimierungs-Verfahren verwendet werden. Verfahren, die eine solche verlustfreie Komprimierung durchführen, sind in Abschnitt 3.3.1 „Verlustfreie Komprimierungsverfahren“ beschrieben (RLE, Huffman, Lempel-Ziv).

Das Netzwerk des Modells muß also um zwei Knoten für die Komprimierung und Dekomprimierung erweitert werden (siebe Abb. 4.3). Durch die Erweiterung können sowohl in den Knoten als auch an den hin- und wegführenden Kanten zusätzliche Verarbeitungszeiten entstehen. Nur wenn diese zusätzlichen Zeiten durch eine Zeitersparnis bei der Übertragung der komprimierten Daten ausgeglichen werden können, ist der Einsatz einer Komprimierung sinnvoll.

Der Komprimierungsfaktor r ist abhängig von den Daten und dem verwendeten Komprimierungsalgorithmus. Handelt es sich dabei z.B. um Volumendaten, in denen große Bereiche mit Null oder einem anderen konstanten Wert besetzt sind, können diese Bereiche sehr gut zusammengefaßt werden. Für die weiteren Rechnungen wird davon ausgegangen, daß der Komprimierungsfaktor nicht von der Problemgröße abhängt und die komprimierten Datensätze um diesen Faktor kleiner sind als die originalen Datensätze:

$$s_{\text{komp}} = r * s \tag{4.7}$$

Der Aufwand für die Komprimierung und Entkomprimierung der Daten hängt vom Komprimierungsalgorithmus und der Größe der Daten ab. Auch hier soll für die weiteren Rechnungen von einer linearen Abhängigkeit des Aufwands von der Datengröße ausgegangen werden. Eine lineare

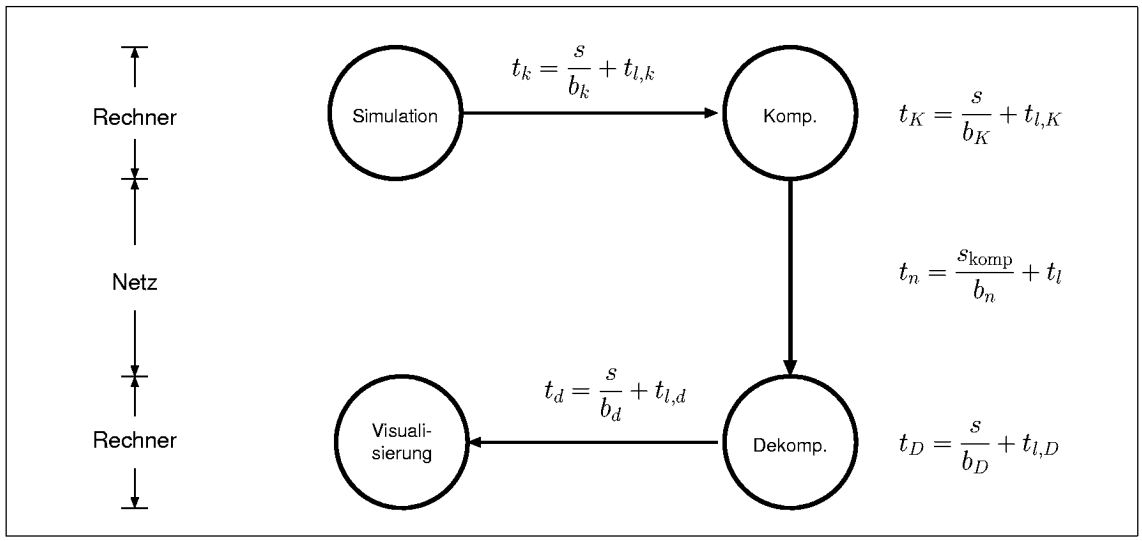


Abbildung 4.3: Netzstruktur bei einfacher Komprimierung: Es werden jeweils vor und nach der Übertragung der Daten über das Netz Komponenten für die Komprimierung und Dekomprimierung der Daten eingebaut. Die Abbildung zeigt an den Knoten und Kanten jeweils die Zeiten, die dort während der Übertragung bzw. während der Komprimierung und Dekomprimierung entstehen.

Abhängigkeit kann dadurch erklärt werden, daß für die Komprimierung ein Datenvektor sequentiell durchlaufen und dabei ein Ausgabevektor mit den komprimierten Daten aufgebaut wird. Bei dem Run-Length-Encoding (RLE) wird dabei nur ein Wiederholungszähler mitgeführt, was keinen zusätzlichen Aufwand bedeutet. Bei Komprimierung mit Hilfe eines Huffman-Algorithmus muß das Alphabet in einen Baum aufgebaut und verwaltet werden. Bei der Lempel-Ziv-Komprimierung muß für jede Position im Datenvektor untersucht werden, ob eine dort beginnende Zeichenfolge schon an einer zurückliegenden Position im Datenvektor zu finden ist. Beide Fälle verursachen einen höheren Aufwand. Bei der in dieser Arbeit verwendeten Komprimierung mit Hilfe des LZO-Verfahrens ist die Mustersuche des Lempel-Ziv-Algorithmus durch ein Hash-Verfahren implementiert, was eine mit der Datengröße linear ansteigende Komprimierungszeit ergibt.

Die lineare Abhängigkeit kann als eine „Komprimierungsbandbreite" formuliert werden, wobei für die Komprimierung und Entkomprimierung jeweils eine eigene Bandbreite spezifiziert wird (b_K und b_D). Viele Komprimierungsverfahren sind insofern unsymmetrisch, daß der Aufwand für die Komprimierung der Daten viel höher ist als der für die Dekomprimierung. Die Verarbeitungszeiten in den jeweiligen Knoten berechnet sich damit zu:

$$t_K = \frac{s}{b_K} + t_{l,K} \qquad t_D = \frac{s}{b_D} + t_{l,D} \tag{4.8}$$

Die beiden Latenzen $t_{l,K}$ und $t_{l,D}$ können dabei als Initialisierungszeiten für das Komprimierungs- bzw. Dekomprimierungs-Verfahren ausgefaßt werden.

Im allgemeinen können sich alle vier Komponenten des Netzwerks auf verschiedenen Rechnern befinden. Daher müssen auch für alle Verbindungskanten im Modell Übertragungszeiten mit Bandbreite und Latenz spezifiziert werden. Es ergeben sich für die beiden neuen Kanten folgenden Zeiten:

$$t_k = \frac{s}{b_k} + t_{l,k} \qquad t_d = \frac{s}{b_d} + t_{l,d} \tag{4.9}$$

Im Normalfall wird die Komprimierung auf dem Simulationsrechner und die Entkomprimierung auf dem Visualisierungsrechner angesiedelt sein, da ansonsten die beiden neuen Kanten Netzverbindungen darstellen würden und die nicht komprimierten großen Datenmengen über externe Kommunikationpfade übertragen werden müßten. Sinnvoll ist dies nur, wenn zwischen den Rechnern ein sehr schnelles Verbindungsnetzwerk wie z.B. bei Parallelrechnern existiert. Wenn Simulation und Komprimierung auf einem Prozessor ausgeführt werden, können diese Komponenten nicht parallel arbeiten. Das heißt, daß die Rechnung der Simulation solange unterbrochen ist, bis der Komprimierungs-Knoten die Daten verarbeitet und an die nächste Komponente abgegeben hat. Gleiches gilt für die Visualisierungsseite.

Für die Bewertung des Netzwerks soll wiederum die Zeit berechnet werden, um welche die Simulation bei angekoppelter Visualisierung langsamer läuft. Sie ergibt sich als Summe der einzelnen Verarbeitungs- bzw. Übertragungszeiten.

$$T_{K,\text{komp}} = t_k + t_K + t_n + t_D + t_d \tag{4.10}$$

Diese Gleichung gilt in dem Fall, daß die Simulation erst nach der kompletten Verarbeitung der Daten mit der Rechnung fortfahren kann. Im allgemeinen ist dies aber schon dann möglich, wenn die Daten den Komprimierungknoten verlassen haben. Bedingung dafür ist aber, daß die Gegenseite, also die Dekomprimierung bzw. die Visualisierung, bereit ist, die Daten anzunehmen. Dies ist dann immer der Fall, wenn die Aufgaben auf der Visualisierungsseite innerhalb eines Simulationsschritts erledigt werden können:

$$T_S > t_D + t_d + t_v + t_i \tag{4.11}$$

Die beiden letzten Summanden der Ungleichung beschreiben dabei die Verarbeitungszeiten innerhalb der Visualisierung (siehe auch Abschnitt 4.2 auf Seite 52). Ist sie erfüllt, reduziert sich $t_{K,\text{komp}}$ entsprechend:

$$T_{K,\text{komp}} = t_k + t_K + t_n = \frac{s}{b_k} + \frac{s}{b_K} + \frac{s\,r}{b_n} + t_l + t_{l,\text{komp}}$$
$$\text{mit} \quad t_{l,\text{komp}} = t_{l,k} + t_{l,K} \tag{4.12}$$

4.3.1 Vergleich mit dem Modell ohne Komprimierung

Mit den Gleichungen 4.2 und 4.12 können die beiden Kommunikationszeiten der bisher vorgestellten Modelle miteinander verglichen werden. Eine Komprimierung lohnt sich nur dann, wenn durch sie die Kommunikationszeit verringert werden kann. Es muß also gelten:

$$T_{K,\text{komp}} < T_{K,\text{direkt}} \Leftrightarrow \frac{s}{b_k} + \frac{s}{b_K} + \frac{s\,r}{b_n} + t_l + t_{l,\text{komp}} < \frac{s}{b_n} + t_l \tag{4.13}$$

$$\Leftrightarrow \frac{s}{b_k} + \frac{s}{b_K} + t_{l,\text{komp}} < \frac{s(1-r)}{b_n} \tag{4.14}$$

Die für den Vergleich wichtigsten Größen sind dabei die Übertragungsbandbreite b_n, die Komprimierungsbandbreite b_K und der Komprimierungsfaktor r. Laufen Komprimierung und Simulation auf demselben Rechner ab, ist die Übertragungsbandbreite zwischen diesen beiden Knoten sehr hoch, so daß die Übertragungszeit dort vernachlässigt werden kann. Wird zusätzlich davon ausgegangen, daß die Startup-Zeit für die Komprimierung sehr viel kleiner ist als die Komprimierungszeit $\frac{s}{b_K}$, ergibt sich folgende Abschätzung:

$$\frac{s}{b_k} + t_{l,\text{komp}} \ll \frac{s}{b_K} \Rightarrow$$
$$T_{K,\text{komp}} < T_{K,\text{direkt}} \Leftrightarrow b_K > \frac{b_n}{1-r} \tag{4.15}$$

Diese Relation kann nun nach jeder der drei Größen aufgelöst werden. Sind zum Beispiel der Komprimierungsfaktor r und die Verarbeitungsgeschwindigkeit des Komprimierungsverfahren bekannt, kann nun die Bandbreite bestimmt werden, ab der es sich lohnt, die Komprimierung bei der Übertragung der Daten einzusetzen. Ist z.B. das Komprimierungsverfahren aufwendig und damit dessen Bandbreite b_K klein, ist die Grenze für die Bandbreite, ab der sich der Einsatz der Komprimierung lohnt, sehr niedrig.

4.4 Modell für die Übertragung mit verlustbehafteter Komprimierung

In diesem Abschnitt wird ein Modell für eine Übertragung mit verlustbehafteter Komprimierung am Beispiel des in Kapitel 3 vorgestellten Verfahrens zur Wavelet-Transformation entwickelt. Die von dieser Transformation gelieferten Daten werden vor der Übertragung zum Visualisierungsrechner zusätzlich in verschiedenen Verarbeitungsschritten weiter reduziert. Dazu wird das aus dem vorigen Abschnitt bekannte Modell für eine Übertragung mit verlustfreier Komprimierung an einigen Stellen erweitert. So müssen die einzelnen Verarbeitungsschritte der Datenreduktion einzeln modelliert und in das Modell eingefügt werden. Dabei muß neben der Aufwandsabschätzung auch die Datengröße s differenzierter betrachtet werden, da diese durch die Datenreduktion auf der Sender- und Empfängerseite unterschiedlich ist.

Die verschiedenen Verarbeitungsschritte spiegeln sich im Modell als zusätzliche Knoten des Netzwerks wieder. Im einzelnen sind das die Wavelet-Transformation mit Hilfe des Pyramidenalgorithmus, das Abschneiden von kleinen Koeffizienten, die Quantisierung, die Kodierung der Daten sowie deren verlustfreie Komprimierung, die schon im letzten Abschnitt behandelt wurde.

4.4.1 Modellparameter für die schnelle Wavelet-Transformation

Durch den Pyramidenalgorithmus werden die Daten in mehreren Iterationen in verschiedene Frequenzbänder zerlegt. Die Zerlegung wird dabei durch zwei Filter (Hoch- und Tiefpaß) realisiert, welche die Daten in einen niederfrequenten und einen hochfrequenten Bereich aufteilen. Im eindimensionalen Fall wird der Datenvektor dabei halbiert. In der nächsten Iterationen werden die Filter nur noch auf den niederfrequenten Teil der Daten angewendet. Bei einer Filterlänge K und einer Vektorlänge N müssen dabei jeweils $N * K$ Multiplikationen ausgeführt werden. Mit Hilfe der Gleichung 3.26 auf Seite 26 kann der Aufwand mit $2KN$ für alle Iterationen des Pyramidenalgorithmus abgeschätzt werden. Im mehrdimensionalen Fall müssen die Filter jeweils für jede Dimension ausgeführt werden. Bei m Dimensionen erhöht sich damit der Aufwand auf $2mKN$ Multiplikationen. Da es eine lineare Abhängigkeit zwischen Datengröße und Aufwand gibt, kann der Zeitaufwand im Modell als eine Verarbeitungsbandbreite in Relation zur Datengröße beschrieben werden. Für die schnelle Wavelet-Transformation ergibt sich dann folgende Verarbeitungsbandbreite für in den Daten enthaltenen Werte:

$$b'_{\text{fwt}} = f_{\text{fwt}} 2\, m\, K \tag{4.16}$$

Dabei ist f_{fwt} die durchschnittliche Zeitdauer, die auf dem entsprechenden Rechner für eine Operation benötigt wird. Sie ist maschinenabhängig und beinhaltet neben den Multiplikationszeiten auch die Zeiten, die für andere Operationen auf den Daten (z.B. Additionen) benötigt wird. An dem für die schnelle Wavelet-Transformation zuständigen Netzknoten entsteht damit folgender Zeitaufwand:

$$t_{\text{fwt}} = \frac{s}{b_{\text{fwt}}} + t_{l,\text{fwt}} \tag{4.17}$$

In dieser Gleichung gibt $t_{l,\text{fwt}}$ eine von der Datengröße unabhängige Latenz an, die z.B. durch eine Initialisierungsphase bedingt ist. In diesem Schritt werden die Daten auf Basis der Gleitpunktzahlen verarbeitet. In die Berechnung des Aufwands fließt also nicht die Gesamtgröße s der Daten ein, sondern die Anzahl der darin enthalten Gleitpunktzahlen. Damit bei den weiteren Berechnungen die Bandbreite immer in Bezug zur Größe der Daten gesetzt werden kann, sollte die Umrechnung auf die Anzahl der in den Daten enthaltenen Werte in die Gleichung für die Bandbreite aufgenommen werden:

$$b_{\text{fwt}} = \frac{s}{n}\, b'_{\text{fwt}} = n_{\text{double}}\, b'_{\text{fwt}} \tag{4.18}$$

Die FWT zerlegt die Daten in verschiedene Frequenzbänder, die eine unterschiedliche Größe besitzen. Bei maximal möglicher Anzahl (i) von Iterationen ergibt sich folgende Summe der Größen der einzelnen Frequenzbänder:

$$1 + 1 + \cdots + (2^m - 1)\frac{s}{4^m} + (2^m - 1)\frac{s}{2^m} = 1 + \sum_{j=1}^{i} \frac{(2^m - 1)s}{2^{(m\,j)}} = s \tag{4.19}$$

Die schnelle Wavelet-Transformation verändert also nicht die Gesamtgröße der Daten. Werden für die weiteren Verarbeitungsschritte alle Frequenzbänder verwendet, kann die Größe der Ausgabedaten des Knoten gleich der Größe der Eingabedaten gesetzt werden:

$$s_{\text{fwt}} = s \tag{4.20}$$

Wird aber in diesem Modell schon eine Datenreduktion durch das Weglassen der obersten j Frequenzbänder eingebaut, ergibt sich die Größe der Ausgabedaten zu:

$$s_{\text{fwt}} = \frac{s}{2^{(mj)}} \tag{4.21}$$

4.4.2 Modellparameter für das Abschneiden von kleinen Koeffizienten

In diesem Verarbeitungsschritt werden diejenigen Koeffizienten in den Frequenzbändern gesucht und zu Null gesetzt, die nur eine geringe Information tragen. Dabei wird die als Entscheidungskriterium genutzte Schranke für den Absolutwert der Koeffizienten durch ein zuvor berechnetes Histogramm bestimmt. Die Daten müssen in diesem Verarbeitungsschritt also zweimal durchlaufen werden, einmal für die Berechnung des Histogramms und einmal für das Suchen und Zurücksetzen der Koeffizienten. Die Initialisierung und Analyse des Histogramms ist vom Aufwand her unabhängig von s und kann daher als Latenz $t_{l,\text{cut}}$ beschrieben werden, da es nur aus einer festen Anzahl von Teilintervallen besteht (z.B. 256 oder 1000). Im wesentlichen steigt damit die Verarbeitungszeit für diesen Knoten linear mit der Größe der Daten und kann damit wiederum mit Hilfe einer Verarbeitungsbandbreite definiert werden:

$$t_{\text{cut}} = \frac{s}{b_{\text{cut}}} + t_{l,\text{cut}} \tag{4.22}$$

In der Formel wird nicht berücksichtigt, daß in diesem Verarbeitungsschritt nur die vom Pyramidenalgorithmus erzeugten Koeffizienten betrachtet werden. Ausgenommen sind dabei die Elemente des untersten Frequenzbandes, welches vom Tiefpaßfilter erzeugt worden ist. Die Elemente dieses Frequenzbands sind Mittelwerte und werden bei dem Abschneiden von Koeffizienten nicht berücksichtigt. Typischerweise wird der Pyramidenalgorithmus nicht bis zur letztmöglichen Iteration ausgeführt, sondern dann abgebrochen, wenn die Größe des niedrigsten Frequenzbands hinreichend klein ist. Die obige Formel beschreibt damit eine obere Schranke für die Verarbeitungszeit in diesem Schritt.

Da in diesem Verarbeitungsschritt die Koeffizienten nur verändert werden, bleibt die Größe der Daten gleich:

$$s_{\text{cut}} = s \tag{4.23}$$

Die Null-Koeffizienten werden erst im Kodierungsschritt aus den Daten entfernt.

4.4.3 Modellparameter für die Quantisierung

Durch die Quantisierung werden die Daten von Gleitpunktzahlen in ganzzahlige Werte umgerechnet. Gegenüber der Speicherung in Form von Gleitpunktzahlen können ganzzahlige Werte (Integer-Zahlen) kompakter gespeichert werden. Der Gewinn an Speicherplatz wird dabei durch einen Genauigkeitverlust erkauft. Bei der Quantisierung wird der von den Daten benutzte Wertebereich in Intervalle unterteilt. So kann jedem Wert ein Intervall zugeordnet werden. Statt der Gleitpunktzahlen muß dann nur noch die Nummer des Intervalls gespeichert werden. Für den Aufwand bedeutet dies, daß vor der Umsetzung der Zahlen deren Minimal- und Maximalwert bekannt sein muß. Diese Werte können von dem vorhergehenden Verarbeitungsschritt übernommen werden, da dort ein Histogramm der Daten berechnet worden ist, wofür die Minimal- und Maximalwerte berechnet werden mußten.

Die Umsetzung selbst bedarf auch nur einen Durchlauf der Daten. Für jede Zahl kann die Nummer des zugehörigen Intervalls direkt berechnet werden. Das heißt, daß ein Aufwand von O(n) entsteht.

Insgesamt kann damit auch die Quantisierungszeit mit Hilfe einer Quantisierungsbandbreite und einer Latenz angegeben werden:

$$t_{\text{quant}} = \frac{s}{b_{\text{quant}}} + t_{l,\text{quant}} \tag{4.24}$$

Die Größe der Daten verringert sich um dem Faktor, um den sich die Größe einer Zahl durch die Umrechnung in eine Integer-Zahl verringert. Bei dem üblichen Speicherbedarf von n_{double}=8 Byte für eine Double-Zahl ergibt sich der Faktor 4 bei einer 16-Bit-Quantisierung (n_{quant}=2) und der Faktor 8 bei einer 8-Bit-Quantisierung (n_{quant}=1) aus:

$$s_{\text{quant}} = \frac{n_q}{n_{\text{double}}} s \tag{4.25}$$

Ein zusätzlicher Aufwand entsteht, wenn für die Rekonstruktion der Double-Werte nicht die Mittelpunkte der Teilintervalle, sondern gewichtete Werte benutzt werden. Für deren Berechnung muß für jedes Teilintervall die Summe der darin enthaltenen Werte mitgeführt werden, um daraus die gewichteten Rekonstruktionswerte zu bestimmen. Bei einer 16-bit-Quantisierung besteht dieses Histogramm immerhin aus 2^{16}=65536 Einträgen und sollte daher in der Latenz mit berücksichtigt werden:

$$t'_{l,\text{quant}} = t_{l,\text{quant},rest} + \frac{2^{8\,n_{\text{quant}}}}{f_{\text{quant\_hist}}} \tag{4.26}$$

Mit $f_{\text{quant\_hist}}$ wird in dieser Formel die Verarbeitungsgeschwindigkeit für die Berechnung der gewichteten Rekonstruktionswerte beschrieben.

Die gewichtigen Rekonstruktionswerte werden auf der Visualisierungsseite zum Wiederherstellen der Gleitpunktzahlen benötigt. Dazu müssen sie dorthin übertragen und damit zu der Größe der Ausgabedaten des Knoten hinzugezählt werden:

$$s'_{\text{quant}} = \frac{n_q}{n_{\text{double}}} s + n_{\text{double}} * 2^{8\,n_{\text{quant}}} \tag{4.27}$$

4.4.4 Modellparameter für die Kodierung

Die Kodierung speichert die von der Quantisierung gelieferten ganzzahligen Daten so, daß möglichst wenige der Null-Koeffizienten übertragen werden müssen. Im Abschnitt 3.3 wurden neben dem direkten Speichern der Daten dafür zwei weitere Verfahren vorgestellt, die jeweils für verschiedene hohe Anteile von Null-Koeffizienten effektiv sind. Der Anteil der Null-Koeffizienten (f_0) hängt zum einen von den Daten selbst ab. Sind dort z.B. große homogene Bereiche enthalten, werden viele Koeffizienten, die im wesentlichen Differenzen benachbarter Werte darstellen, gleich Null sein. Zum anderen werden in dem hier vorgestellten Verfahren in dem Verarbeitungsschritt „Abschneiden von kleinen Koeffizienten“ weitere Null-Koeffizienten erzeugt. Insgesamt ergibt sich die Anzahl und die Größe der Null-Koeffizienten als Summe zweier Anteile:

$$f_0 = f_{\text{cut}} + f_{\text{daten}}, \qquad s_0 = f_0 * s \tag{4.28}$$

Während der Faktor f_{cut} durch den Benutzer bestimmt werden kann, ist der zweite Faktor f_{daten} von den Daten abhängig und kann deshalb im Modell nicht weiter spezifiziert werden.

Durch die Kodierung der Daten wird die Größe verändert. Alle drei Verfahren benötigen dafür nur einen Aufwand von O(n), da jeweils alle Werte nur einmal betrachtet werden müssen. Also ist

die Kodierungszeit t_{code} wiederum durch eine Gerade mit Verarbeitungsbandbreite und Latenz als Parameter modellierbar:

$$t_{\text{code}} = \frac{s}{b_{\text{code}}} + t_{l,\text{code}} \tag{4.29}$$

Für den durch die Kodierung reduzierten Speicherbedarf wurde in Abschnitt 3.3 die Gleichung 3.39 aufgestellt. Die Größe der kodierten Daten ist laut dieser Gleichung von der Anzahl der Koeffizienten (n), dem Anteil der Null-Koeffizienten (f_0) und der Anzahl der Dimensionen (m) abhängig. Für die Auswahl des Kodierungsverfahrens wird zusätzlich der Speicherbedarf für eine Intervallnummer der Quantisierung (n_{quant}), die Bitmaske (n_{mask}) und einer Offset-Angabe (n_{int}) benötigt. Der in Abbildung 3.18 auf Seite 39 dargestellte Verlauf des Speicherplatzbedarfs für verschiedene Anteile von Null-Koeffizienten zeigt, daß bei typischen Werten für die einzelnen Parameter in einem weiten Bereich von f_0 zwischen 12.5 % und 93.75 % das zweite Verfahren ausgewählt wird, das die Koeffizienten gruppiert und mit einer Bitmaske versieht. Im dreidimensionalen Fall (m=3) ist dies sogar für f_0 zwischen 6.25 % und 96.88 % der Fall. Dies erlaubt die Vereinfachung, in den weiteren Rechnungen für die Berechnung der reduzierten Datengröße nur das zweite Verfahren zu berücksichtigen. Für ein Frequenzband ergibt sich folgender Speicherbedarf:

$$s'_{b,\text{code}} = n\,(n_{\text{quant}}(1 - f_0) + n_{\text{mask}}\frac{1}{2^m}) \tag{4.30}$$

Da die Bitmaske in jedem Frequenzband benötigt wird, ergibt für deren Speicherplatz bei maximal möglicher Anzahl von Frequenzbändern i:

$$s'_{\text{mask}} = n\,n_{\text{mask}} \sum_{j=1}^{i} \frac{1}{2^{mj}} = n\,n_{\text{mask}} \frac{1 - \frac{1}{2^{mj}}}{2^m - 1} \lesssim n\,n_{\text{mask}} \frac{1}{2^m - 1} \tag{4.31}$$

Die Größe aller quantisierten und kodierten Frequenzbänder kann wie folgt abgeschätzt werden:

$$s'_{\text{code}} \lesssim \frac{s}{n_{\text{double}}} (n_{\text{quant}}(1 - f_0) + n_{\text{mask}}\frac{1}{2^m - 1}) \tag{4.32}$$

4.4.5 Zusätzlicher Aufwand auf der Visualisierungsseite

Der Empfänger muß die Reduktionsschritte in umgekehrter Reihenfolge und inverser Wirkung ausführen. Dabei muß nach der Entkomprimierung als erstes die Kodierung der Daten aufgelöst, dann die ganzzahligen Werte wieder durch Gleitpunktzahlen ersetzt und darauf die inverse Wavelet-Transformation ausgeführt werden. Da die Rekonstruktionswerte mit den kodierten Daten geliefert werden, ist bei der inversen Quantisierung die Berechnung eines Histogramms nicht mehr notwendig.

Für die beiden ersten Schritte ergibt sich daher eine lineare Abhängigkeit der Verarbeitungszeit von der Datengröße.

$$T_{D,\text{wave}} = \frac{s'_{\text{code}}}{b_D} + \frac{s_{\text{quant}}}{b_{\text{i\_code}}} + \frac{s}{b_{\text{i\_quant}}} + \frac{s}{b_{\text{fwt}}} + t_{l,\text{fwt}} + t'_{l,\text{i\_quant}} + t_{l,\text{i\_code}} + t_{l,D} \tag{4.33}$$

4.4.6 Zusammenfassung der Teilschritte

Da die einzelnen Verarbeitungsschritte nacheinander ausgeführt werden müssen, kann im Modell für jeden Schritt ein Knoten in den Übertragungsweg eingefügt werden. Es wird davon ausgegangen, daß sich alle neu hinzugefügten Knoten auf demselben Rechner befinden, so daß die Zeiten

für die Übertragung zwischen den Knoten vernachlässigt werden können (siehe Abb. 4.4 auf der nächsten Seite). Der Zeitverbrauch in allen Teilschritten der verlustbehafteten Komprimierung steigt linear mit der Größe der Daten an. Es ergibt sich damit folgender Zeitverbrauch auf der Seite der Simulation:

$$\begin{aligned} T_{K,\text{wave}} &= t_{\text{fwt}} + t_{\text{cut}} + t_{\text{quant}} + t_{\text{code}} + t_K + t_n \\ &= \frac{s}{b_{\text{fwt}}} + \frac{s_{\text{fwt}}}{b_{\text{cut}}} + \frac{s_{\text{cut}}}{b_{\text{quant}}} + \frac{s_{\text{quant}}}{b_{\text{code}}} + \frac{s'_{\text{code}}}{b_K} + \frac{r\, s'_{\text{code}}}{b_n} \\ &\quad + t_{l,\text{fwt}} + t_{l,\text{cut}} + t'_{l,\text{quant}} + t_{l,\text{code}} + t_{l,K} + t_l \end{aligned} \tag{4.34}$$

Die Latenzen der Teilschritte können addiert werden:

$$t_{l,\text{wave}} = t_{l,\text{fwt}} + t_{l,\text{cut}} + t'_{l,\text{quant}} + t_{l,\text{code}} + t_{l,K} \tag{4.35}$$

Auch die Bandbreiten der einzelnen Teilschritte können zusammengefaßt werden, da in allen Termen s ausgeklammert werden kann. Als Gesamtbandbreite der verlustbehafteten Komprimierung ergibt sich damit:

$$\frac{1}{b_{\text{wave}}} = \frac{1}{b_{\text{fwt}}} + \frac{1}{b_{\text{cut}}} + \frac{1}{b_{\text{quant}}} + \frac{n_q}{n_{\text{double}}}\frac{1}{b_{\text{code}}} + \frac{(1-f_0)n_{\text{quant}} + \frac{1}{2^m-1}n_{\text{mask}}}{n_{\text{double}}}\frac{1}{b_K} \tag{4.36}$$

Die Größe der durch die Quantisierung und Komprimierung veränderten Daten ergibt sich zu:

$$s_{\text{wave}} = r_{\text{wave}}\, s \qquad\qquad r_{\text{wave}} = r\frac{(1-f_0)n_{\text{quant}} + \frac{1}{2^m-1}n_{\text{mask}}}{n_{\text{double}}} \tag{4.37}$$

Die Zeit, die mit dem Komprimieren und Versenden der Daten verbracht wird, kann damit wie folgt bestimmt werden:

$$T_{K,\text{wave}} = \frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{b_n} + t_l \tag{4.38}$$

Um entscheiden zu können, ob sich der Einsatz dieser Komprimierungsmethode lohnt, muß die von vielen Bandbreiten und anderen Parametern abhängige Gleichung mit der Kommunikationszeit der unkomprimierten Übertragung verglichen werden. Wird davon ausgegangen, daß die Startup-Zeit $t_{l,\text{wave}}$ für die Komprimierung sehr viel kleiner ist als die Komprimierungszeit, ergibt sich für die Ungleichung $T_{K,\text{wave}} < T_{K,\text{direkt}}$ folgende Abschätzung:

$$T_{K,\text{wave}} < T_{K,\text{direkt}} \quad\Leftrightarrow\quad \frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{b_n} + t_l < \frac{s}{b_n} + t_l \tag{4.39}$$

$$t_{l,\text{wave}} \ll \frac{s}{b_{\text{wave}}} \quad\Rightarrow\quad b_{\text{wave}} > \frac{b_n}{1 - r_{\text{wave}}} \tag{4.40}$$

Der Einsatz der Komprimierung mit Hilfe der Wavelet-Transformation kann sogar dann sinnvoll sein, wenn die dadurch entstehende Kommunikationszeit größer ist als die der direkten Übertragung. Durch die Zerlegung der Daten in einzelne Frequenzbänder ist eine Datenreduktion durch das Weglassen der oberen Frequenzbänder möglich. Erst dadurch ist das Computational Steering bei großen Datenmengen möglich. Auch die im nächsten Abschnitt behandelte progressive Übertragung ist erst durch die Aufteilung in einzelne Frequenzbänder möglich.

Es bleibt zu beachten, daß beim Einsatz einer Komprimierung mehr Speicherplatz benötigt wird. Neben den Originaldaten müssen auch die komprimierten Daten im Speicher gehalten werden. Das heißt, daß wenigstens ein zusätzlicher Speicherplatz der Größe s_{code} bereitgehalten werden muß. Sollen die Originaldaten im Speicher nicht zerstört werden, müssen diese zusätzlich vor der Wavelet-Transformation kopiert werden, womit nochmals ein Speicherplatzbedarf der Größe

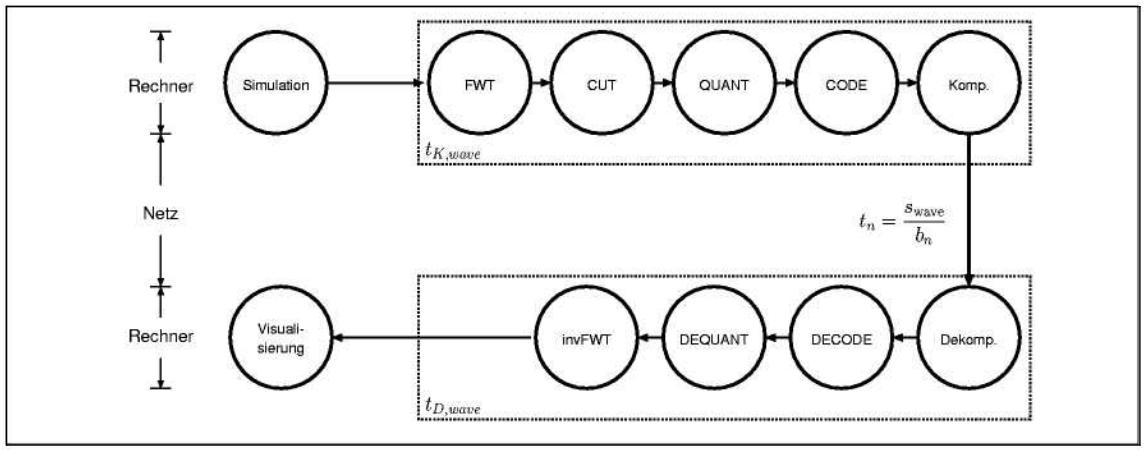


Abbildung 4.4: Netzstruktur bei verlustbehafteter Komprimierung: Vor und nach der Übertragung der Daten über das Netz werden in mehreren Schritten die Daten mit Hilfe der Wavelet-Transformation, der Quantisierung und Kodierung in ihrer Größe reduziert und komprimiert. Auf der Visualisierungsseite müssen diese Schritte in umgekehrter Reihenfolge und mit inverser Wirkung ausgeführt werden.

s hinzukommt. Würde bei der Wavelet-Transformation das höchste Frequenzband nicht übertragen, müßten dessen Koeffizienten auch nicht gespeichert werden, wodurch sich der zusätzliche Speicherbedarf auf $\frac{s}{2^m}$ reduziert.

4.5 Modelle für die progressive Übertragung

Durch die Aufteilung der Daten in einzelne Frequenzbänder mit Hilfe des Pyramidenalgorithmus besteht die Möglichkeit, die Daten nicht in einem Schritt, sondern nach und nach zur Visualisierung zu übertragen. Beginnend mit dem niedrigsten Frequenzband werden zuerst die grob aufgelösten Daten übertragen und angezeigt. In jedem weiteren Schritt werden dann zusätzliche Frequenzbänder übertragen, welche die Auflösung der Daten erhöhen (siehe Abb. 4.5).

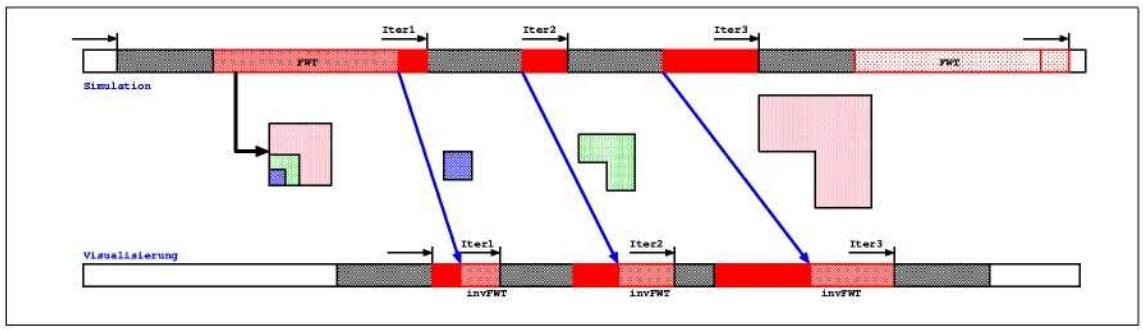


Abbildung 4.5: Ablauf einer progressiven Übertragung: Die durch den Pyramidenalgorithmus zerlegten Daten können bei der progressiven Übertragung in mehreren Teilen zu der Visualisierung übertragen werden. Die Abbildung zeigt den Verlauf der Übertragung eines in drei Frequenzbänder aufgeteilten Datensatzes. Dabei wird nur in jeder dritten Iteration ein neuer Datensatz transformiert. In den dazwischenliegenden Iterationen wird jeweils ein Teil des zuvor transformierten Datensatzes übertragen.

Durch dieses Verfahren kann die Übertragung eines in q Frequenzbänder zerlegten Datensatzes auf q Iterationen des Simulationsprogramms ausgedehnt werden, wodurch die Kommunikationszeit T_K im Durchschnitt um den Faktor $1/q$ verringert werden kann. Gegenüber der nicht progressiven Übertragung sieht der Betrachter insgesamt weniger Datensätze, da nur in jeder q-ten Iteration für die Übertragung neue Daten aus dem Simulationsprogramm entnommen werden. Die Datensätze, die vom Simulationsprogramm in den dazwischenliegenden Iterationen berechnet werden, werden in diesem Verfahren nicht übertragen, wodurch ggf. kurzzeitige Veränderungen in den Daten nicht erkannt werden.

Die gleiche Verringerung der Übertragungszeit könnte auch damit erreicht werden, daß nur in jeder q-ten Iteration die kompletten Daten zur Visualisierung übertragen werden. Die progressive Übertragung hat dagegen den Vorteil, daß auf der Visualisierungsseite zuerst nur ein kleiner Datensatz angezeigt werden muß, der nach und nach verfeinert wird. Der Betrachter sieht dadurch früher die wesentlichen Information zu den aktuellen Ergebnissen. Zusätzlich besteht bei der progressiven Übertragung auch die Möglichkeit, die Übertragung der höheren Frequenzbänder abzubrechen. Diese Dynamik führt im Zusammenspiel mit VISIT zu einer automatischen Anpassung an die Leistungsfähigkeit des Netzes und des Visualisierungsrechners. Ist die Visualisierung noch mit der Darstellung der bisher gesendeten Frequenzbänder beschäftigt, findet die Übertragung des nächst-höheren Frequenzbandes erst gar nicht statt. Situationen, in denen die Visualisierung und die Simulation mit der Übertragung und Aufbereitung der neuen Daten beschäftigt und damit lange blockiert sind, können mit diesem Mechanismus weitgehend vermieden werden. Für die Anpassung an verschiedene Simulationsprogramme und damit an verschieden große Rechenzeiten pro Simulationsschritt sollte bei der Ankopplung einer Visualisierung immer die Möglichkeit bestehen, eine vorgegebene Anzahl von Simulationsschritten zu überspringen. Im Modell wird diese interne kleinere Aufteilung in Simulationsschritte nicht berücksichtigt: Aus Sicht des Modell besteht eine Iteration aus der Zeitspanne von einer Übertragung zur Visualisierung bis zur nächsten.

Bei dem in dieser Arbeit vorgestellten Modell der Kopplung wird die Übertragungszeit für einen durchschnittlichen Simulationsschritt definiert. Da bei der progressiven Übertragung die Größe der Frequenzbänder immer weiter zunimmt, ist die Kommunikationszeit für die q Simulationsschritte der Übertragungszyklus sehr unterschiedlich. Bei der ersten Iteration des Zyklus kommt weiterhin die Zeit für die Transformation der Daten hinzu. Für den Vergleich mit anderen Modellen kann nur die durchschnittliche Kommunikationszeit der q Iterationen einer Übertragungsphase herangezogen werden.

Diese durchschnittliche Kommunikationszeit für die q Schritte der Übertragung eines kompletten Datensatzes ergibt sich aus der Kommunikationszeit der nicht progressiven Variante:

$$T_{K,\text{wave,prog}} = \frac{1}{q}\left(\frac{s}{b_{\text{wave}}} + \frac{s_{\text{wave}}}{b_n} + t_{l,\text{wave}}\right) + t_l \tag{4.41}$$

Dies ist dadurch begründet, daß innerhalb der q Simulationsschritte ein Datensatz der gleichen Größe transformiert, kodiert und übertragen werden muß wie in jedem Simulationsschritt der nicht progressiven Variante. Für eine bessere Einsicht soll die Verteilung dieser Kommunikationszeit auf die verschiedene Teilschritte aufgeführt werden. Der Hauptteil der Zeit fällt in die Transformation der Daten und damit in den ersten Teilschritt:

$$T_{K,\text{wave,prog},1} = \frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + t_{n,1} + t_l \tag{4.42}$$

In den weiteren Schritten fällt nur noch die Übertragungszeit der komprimierten Daten über das Netz ins Gewicht:

$$T_{K,\text{wave,prog},i} = t_{n,i} + t_l \qquad 1 < i \leq q \tag{4.43}$$

Diese Übertragungszeit $t_{n,i}$ hängt von der Größe der einzelnen Frequenzbänder ab und kann folgendermaßen berechnet werden:

$$t_{n,1} = \frac{1}{b_n}\frac{s_{\text{wave}}}{2^{m(q-1)}} + t_l \qquad t_{n,i} = \frac{1}{b_n}\, s_{\text{wave}}\, \frac{2^m - 1}{2^{m(q-i+1)}} + t_l \quad \text{mit } 1 < i \leq q \tag{4.44}$$

Dabei wird implizit davon ausgegangen, daß die Komprimierung der einzelnen Frequenzbänder die gleiche Komprimierungsrate ergibt. Dann ist das Größenverhältnis der komprimierten Frequenzbänder gleich dem Größenverhältnis der nicht komprimierten Frequenzbänder.

Da bei der progressiven Übertragung die Daten in mehreren Simulationsschritten übertragen werden müssen, ist eine Speicherung der Daten auch nach der Beendigung eines Simulationsschritts nötig. Da die Frequenzbänder einzeln quantisiert, kodiert und komprimiert werden können, ist es möglich, die Daten nur in komprimierter und damit sehr viel kleineren Speicherbedarf für die Übertragung in den nachfolgenden Simulationsschritten zu speichern. Damit ist der zusätzliche Speicheraufwand für die progressive Übertragung begrenzt.

4.6 Ankopplung an parallele Simulationsprogramme

Die in dieser Arbeit berücksichtigten parallelen Simulationsprogrammen arbeiten auf Parallelrechnern mit verteiltem Speicher. Daher sind die für die Visualisierung notwendigen Daten auf die Hauptspeicher der einzelnen Rechenknoten verteilt. Bei der Ankopplung einer Visualisierung an ein solches Simulationsprogramm müssen daher die Übertragungszeiten, die sich durch das ggf. nötige Verlagern der Daten von einem Prozessor zu einem anderen Prozessor ergeben, mit berücksichtigt werden.

Für die Ankopplung der Visualisierung sind drei verschiedene Modelle vorstellbar. Im ersten Modell wird nur der erste, auch als Master-Knoten bezeichnete Rechenknoten mit der Visualisierung verbunden. Das zweite Modell geht von einer parallelen Ankopplung aller Rechenknoten aus. Die Anbindung über einen zusätzlichen, nur für Kommunikation mit der Visualisierung zuständigen Knoten wird im dritten Modell behandelt. Auch hier stellt sich wieder die Frage, für welche Randbedingungen welches Modell am günstigsten ist. Dabei spielen die Bandbreiten der einzelnen Übertragungswege innerhalb und außerhalb des Rechners die Hauptrolle. Ein Maß für die Beeinflussung des parallelen Simulationsprogramms ist neben der für die Ankopplung nötigen Kommunikationszeit auch die Beeinflussung des Speedups des parallelen Programms, der unter anderem auch dadurch verringert wird, daß die Arbeit ungleichmäßig auf die einzelnen Prozessoren verteilt wird. Eine auf einen Prozessor beschränkte Kommunikation verringert damit auch den Speedup.

4.6.1 Modell für die Übertragung über den Master-Knoten

Das in vielen Simulationsprogrammen angewendete Prinzip der Master-Slave-Hierarchie der Prozesse, bei dem ein Prozeß die Steuerung aller anderen Prozesse übernimmt, kann bei der Ankopplung der Visualisierung in der Art ausgenutzt werden, daß der Master-Knoten auch die Kommunikation mit der Visualisierung übernimmt. In einigen Fällen wird der Master-Knoten nur für das Verteilen der Daten und das Einsammeln der Ergebnisse eingesetzt und nicht mit in die Berechnungen einbezogen. Dann steht ihm noch Zeit für Kommunikation mit der Visualisierung zur Verfügung. In den meisten Fällen aber wird der Master-Prozeß bei den Simulationsrechnungen mitarbeiten, so daß eine zusätzliche Kommunikation mit der Visualisierung auch eine zusätzliche Belastung der gesamten parallelen Simulationsrechnung ergibt. In der Modellierung soll von diesem Fall ausgegangen werden.

Bei der Modellierung müssen zwei Übertragungswege berücksichtigt werden. Zuerst müssen die Daten von den Slave-Prozessen zum Master-Prozeß übertragen werden und dann von dort weiter zur Visualisierung. Die Slave-Prozesse können erst dann mit der Arbeit fortfahren, wenn der Master-Prozeß die Übertragung zur Visualisierung beendet hat (siehe Abb. 4.6 auf der nächsten Seite).

Die Daten der einzelnen Prozessoren müssen vom Master-Prozessor sequentiell nacheinander angenommen werden. Daher geht in die für die Kopplung benötigte Kommunikationszeit $T_{K,\mathrm{master}}$

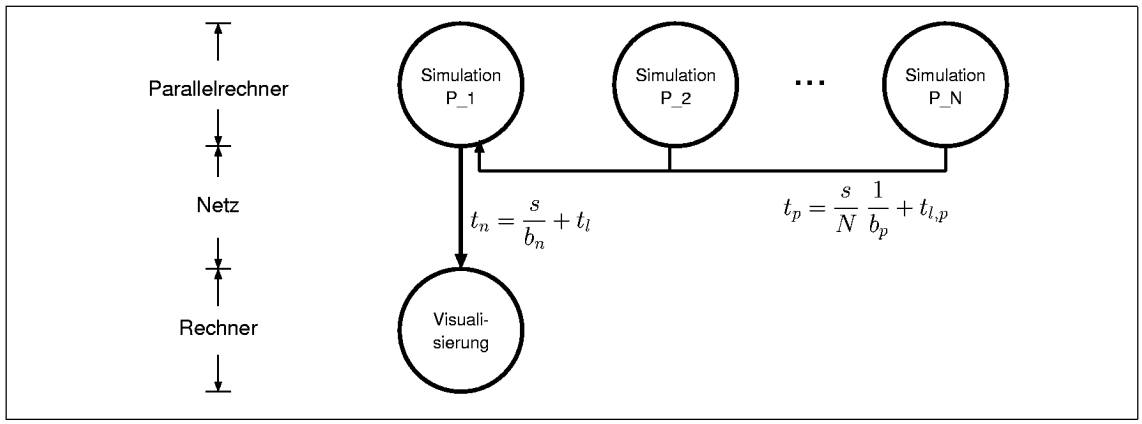


Abbildung 4.6: Netzstruktur bei der Anbindung an ein paralleles Simulationsprogramm über einen Master-Knoten: Bevor die Daten zur Visualisierung übertragen werden können, müssen diese vom Master-Prozeß (P_1) von den Slave-Prozessen (P_2 ... P_N) eingesammelt werden. Jeder Prozessor muß nur seinen Anteil zum ersten Prozessor übertragen, der bei einer gleichmäßigen Verteilung die Größe s/N besitzt.

die Übertragungszeit der einzelnen Teilpakete mit dem Faktor N ein. Im allgemeinen muß auch die Übertragungszeit zu der Visualisierung voll mit eingerechnet werden:

$$T_{K,\text{master}} = t_n + Nt_p = \frac{s}{b_n} + t_l + N\,\frac{s}{Nb_p} + N\,t_{l,p} = s(\frac{1}{b_n} + \frac{1}{b_p}) + t_l + N\,t_{l,p} \qquad (4.45)$$

Im Vergleich zu der Übertragung bei einem seriellen Simulationsprogramm ändert sich die Kommunikationszeit um die Zeit, die innerhalb des Parallelrechners für die Übertragung der Daten zum Master-Knoten benötigt wird. Insbesondere geht in diese Zeit die Latenz $t_{l,p}$ mit dem Faktor N ein, da auf dem Master-Knoten für jede Nachricht eine neue Initialisierung stattfinden muß. Es ist zu erwarten, daß die Bandbreite innerhalb des Parallelrechners (b_p) um ein Vielfaches höher ist als die zwischen Simulation und Visualisierung.

Eine Komprimierung der Daten kann schon vor der Übertragung zum Master-Prozeß durchgeführt werden, so daß alle Prozesse an dieser Arbeit beteiligt werden und nur noch die komprimierten Daten übertragen werden müssen. Da in die Gleichungen für den Zeitverbrauch der Komprimierungsschritte diese bis auf den konstanten Anteil $t_{l,\text{wave}}$ jeweils linear eingeht, ist die für die Komprimierung benötigte Zeit durch den Einsatz von N Prozessoren um den Faktor $1/N$ geringer als bei einem sequentiellen Programm. Die Startup-Zeit $t_{l,\text{wave}}$ für die Komprimierung muß auch nur einmal berücksichtigt werden, da die Komprimierung auf allen Prozessoren gleichzeitig abläuft. Bei der Komprimierung mit Hilfe der Wavelet-Transformation ergibt sich dann folgende Kommunikationszeit:

$$T_{K,\text{master,wave}} = \frac{1}{N}\frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + s_{\text{wave}}(\frac{1}{b_n} + \frac{1}{b_p}) + t_l + N\,t_{l,p} \qquad (4.46)$$

Auf der Seite der Visualisierung ändert sich bei der Ankopplung an ein paralleles Simulationsprogramm über den Master-Knoten aus Sicht der Kommunikation nichts, da es weiterhin nur einen Kommunikationspartner auf der anderen Seite gibt. Unterschiedlich ist nur die Aufteilung der Daten. Da der Master-Knoten die von den Slave-Prozessen gelieferten Daten der einzelnen Teilgebiete nicht mehr zu einem Gesamtgebiet zusammenfügt, erhält die Visualisierung nun N komprimierte Teildatensätze. Der Aufwand für die Entkomprimierung ändert sich dadurch aber nicht.

Einfluß auf den Speedup des parallelen Programms

Der Speedup beschreibt den Faktor, um den ein auf N Prozessoren parallel ausgeführtes Programm schneller ist als ein sequentielles Programm, das mit der gleichen Problemgröße arbeitet. Nur wenn innerhalb des parallelen Programms keine Kommunikation stattfindet und außerdem die Arbeit gleichmäßig verteilt ist, kann ein linearer Speedup erreicht werden. Der Verlauf der Speedup-Kurve einer realen Anwendung ist schwer vorherzusagen, da dieser z.B. von dem Anteil der nicht parallelisierbaren Rechenarbeit, der verwendeten Parallelisierungsstrategie und dem Kommunikationsverhalten der Anwendung abhängt.

Den Einfluß der nicht parallelisierbaren Rechenarbeit auf den Speedup einer Anwendung beschreibt die folgende als *Amdahl's Law* bezeichnete Gleichung [52]:

$$S(N) = \frac{T_S}{f\,T_S + (1-f)T_S/N} \qquad \lim_{N\to\infty} S(N) = \frac{1}{f} \tag{4.47}$$

Der serielle Anteil (f) einer parallelen Anwendung bestimmt nach diesem Gesetz den maximal möglichen Speedup.

Für die Modellierung der Kopplung wird das oben beschriebene Gesetz um einen weiteren Anteil ergänzt, der die Abhängigkeit der für die interne Kommunikation benötigten Zeit von der Datengröße s beschreibt. Im allgemeinen kann über die Menge der auszutauschenden Daten in einem parallelen Simulationsprogramm keine Aussage gemacht werden. Daher wird sie als eine nicht weiter spezifizierte Funktion $F(s)$ in die Gleichung eingebracht. Dabei wird nur die reine Übertragungszeit der Daten berücksichtigt. Die Latenz wird implizit dem seriellen Anteil des Programms zugeschlagen. Dann ergibt sich für den Speedup folgende Gleichung:

$$S(N) = \frac{T_S}{T_P} = \frac{T_S}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p}} \tag{4.48}$$

Als Beispiel für die Funktion F kann eine Abschätzung der internen Kommunikation des in dieser Arbeit benutzten Simulationsprogramms Trace aufgeführt werden. Die einzelnen Rechenknoten von Trace tauschen in jedem Iterationschritt die Randschichten der Teilgebiete mit den jeweiligen Nachbarn mehrmals aus. Bei würfelförmigen Teilgebieten hat ein innenliegender Knoten in etwa $6(n/N)^{\frac{2}{3}}$ Randelemente, die ausgetauscht werden müssen. Für die Funktion F(s) ergibt sich damit:

$$F_{\text{Trace}}(s) \approx 6\left(\frac{s/n_{\text{double}}}{N}\right)^{\frac{2}{3}} \tag{4.49}$$

Durch die Parallelisierung wird die für einen Prozessor zu erledigende Arbeit mit der Erhöhung der Prozessoranzahl immer geringer. Bei Trace nimmt auch die für die Kommunikation benötigte Zeit ab, so daß der serielle Anteil $f\,T_S$ bei hohen Prozessoranzahlen den Verlauf der Speedup-Kurve bestimmt.

Für die Berechnung des Speedups bei einem an eine Visualisierung angekoppelten Simulationsprogramm wird die entsprechende Zeitdauer eines sequentiellen ($T_{\mathrm{S,v}}$) und eines parallelen Simulationsschritts ($T_{\mathrm{P,v}}$) benötigt:

$$T_{\mathrm{S,v}} = T_S + t_n \qquad T_{\mathrm{P,v}}(N) = f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + t_n + N\,t_p \tag{4.50}$$

Dann kann der Speedup eines angekoppelten Simulationsprogramms wie folgt bestimmt werden:

$$\begin{aligned} S_v(N) &= \frac{T_{\mathrm{S,v}}}{T_{\mathrm{P,v}}(N)} = \frac{T_S + t_n}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + t_n + N\,t_p} \\ &= \frac{T_S + \frac{s}{b_n} + t_l}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + \frac{s}{b_p} + N\,t_{l,p} + \frac{s}{b_n} + t_l} \end{aligned} \tag{4.51}$$

Werden die Daten zusätzlich auf den Rechenknoten komprimiert, muß der Zeitaufwand dafür in den Zeitdauer eines parallelen Simulationsschritts eingerechnet werden. Bei der Übertragung muß dann nur noch die geringere Größe s_{wave} berücksichtigt werden:

$$S_{v,\mathrm{wave}}(N) = \frac{T_S + \frac{s}{b_{\mathrm{wave}}} + t_{l,\mathrm{wave}} + \frac{s_{\mathrm{wave}}}{b_n} + t_l}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + \frac{s/N}{b_{\mathrm{wave}}} + t_{l,\mathrm{wave}} + \frac{s_{\mathrm{wave}}}{b_p} + N\,t_{l,p} + \frac{s_{\mathrm{wave}}}{b_n} + t_l} \tag{4.52}$$

Für die Bewertung des Verlaufs der Speedup-Kurve ist es sinnvoll, den Speedup für große Prozessorzahlen zu betrachten. Wird dieser in dem Bereich der Kurve konstant oder nimmt er dort sogar ab, gibt es für ihn eine Obergrenze, die den maximal möglichen Speedup beschreibt. Eine weitere Vereinfachung der Formel wird unter der Annahme möglich, daß die Bandbreite innerhalb des Parallelrechners gegenüber der externen Bandbreite sehr hoch und die Latenz sehr gering ist. Bei der Abschätzung ist der Term $N\,t_{l,p}$ problematisch, da hier eine hohe Prozessoranzahl mit einer sehr kleinen Latenz multipliziert wird. Bei den heute verfügbaren Rechnern ist für die rechnerinterne Latenz ein Wert im Bereich von μs möglich, so daß selbst bei einer hohen Prozessoranzahl von z.B. 1000 der Term nur einen Beitrag im Millisekundenbereich liefert und damit in der gleichen Größenordnung wie die Latenz der externen Kommunikation liegt. Dann ergibt sich:

$$N \gg 1,\ b_p \gg b_n,\ \Rightarrow\ S_v(N) < \frac{T_S + \frac{s}{b_n} + t_l}{f\,T_S + \frac{s}{b_n} + t_l + N\,t_{l,p}} < \frac{1}{f} \tag{4.53}$$

Obwohl also $N\,t_{l,p}$ das asymptotische Verhalten von $S_v(N)$ bestimmt, kann dieser Term in den hier betrachteten Modellen vernachlässigt werden.

4.6.2 Modell für die parallele Übertragung

Durch die Ankopplung der Visualisierung an den Master-Knoten des Simulationsprogramms stellt sich ein Ungleichgewicht bei der Arbeitsverteilung ein, da die Prozessoren erst mit den Berechnungen fortfahren können, wenn der Master-Knoten die Daten zur Visualisierung übertragen hat. Dieser Nachteil kann vermieden werden, wenn alle Knoten direkt mit der Visualisierung verbunden sind. Eine Übertragung der Daten innerhalb des Parallelrechners entfällt damit.

Diese Kopplungsstrategie scheint gegenüber der vorher vorgestellten wesentlich günstiger. Bei genauerer Betrachtung wird jedoch der Engpaß durch die parallele Ankopplung nur weiter in Richtung Visualisierung verlegt. Die Daten können zwar parallel aus dem Parallelrechner übertragen werden, auf der Visualisierungsseite steht aber typischerweise nur ein Rechner zur Verfügung, auf dem die ankommenden Datenpakete sequentiell verarbeitet werden müssen. Wenn auch das Visualisierungsprogramm eine gewisse Parallelität bei der Datenannahme durch z.B. mehrere Threads besitzt, kann die parallele Übertragung ggf. bis in das Visualisierungsprogramm hinein bestehen. Dann ist ein Vorteil durch die parallele Ankopplung zu erwarten.

Bei vielen Parallelrechnern ist zwar interne Kommunikation in einer sehr hohen Geschwindigkeit möglich, für die Kommunikation zu externen Rechnern steht häufig aber nur eine geringere Bandbreite zur Verfügung. So wird innerhalb des Parallelrechners CRAY T3E die gesamte Kommunikation nach außen über einen Knoten geleitet. Durch die parallele Kopplung wird daher der Engpaß

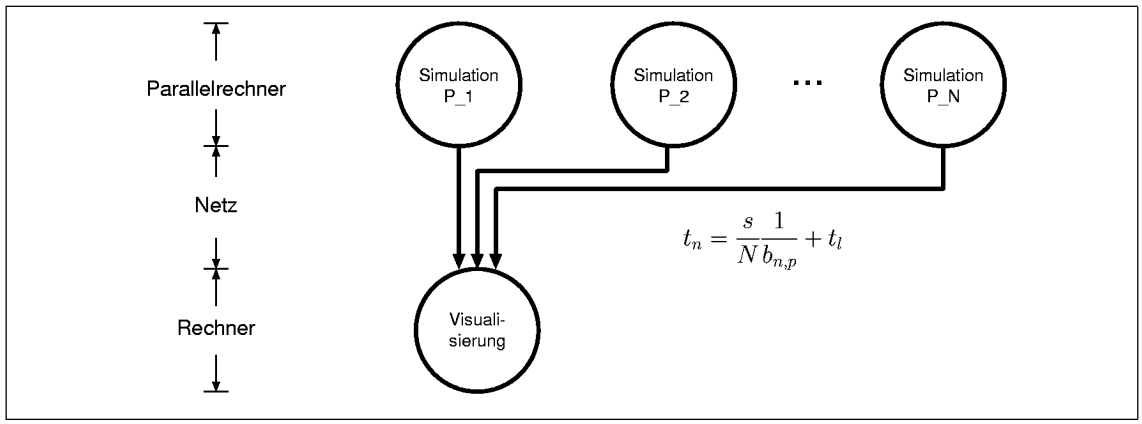


Abbildung 4.7: Netzstruktur bei paralleler Anbindung: Jeder Prozessor ist in dieser Anordnung direkt mit der Visualisierung verbunden. Die Daten können damit parallel zur Visualisierung übertragen werden. Da jeder Prozessor nur seinen Anteil übertragen muß, reduziert sich die Übertragungszeit entsprechend.

vom Master-Knoten der Anwendung zu diesem Kommunikationsknoten verlagert. Da damit die für die parallele Anwendung zur Verfügung stehende Bandbreite durch diesen Knoten begrenzt ist, gilt in etwa $b_{n,p} = b_n/N$. Auch bei dem zweiten in dieser Arbeit benutzten Rechner, dem Linux-Cluster ZAMpano, wird die gesamte externe Kommunikation über den FrontEnd-Knoten geleitet.

Im allgemeinen muß die Bandbreite und Latenz für dieses Modell aus den oben genannten Gründen in Abhängigkeit von N definiert werden. Damit gilt für die Kommunikationszeit:

$$T_{K,\text{parallel}} = t_n = \frac{1}{N}\frac{s}{b_{n,p}(N)} + t_l(N) \tag{4.54}$$

Werden die Daten vor der Übertragung komprimiert, muß diese Zeit in der Formel mit berücksichtigt werden:

$$T_{K,\text{parallel,wave}} = \frac{1}{N}\left(\frac{s}{b_{\text{wave}}}\right) + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{N}\frac{1}{b_{n,p}(N)} + t_l(N) \tag{4.55}$$

Wird in dieser Gleichung die Bandbreite $b_{n,p}$ durch die entsprechend geringere Bandbreite b_n/N ersetzt, was die einem Parallelrechnern mit einer über einen Knoten gerouteten externe Kommunikation entspricht, ergibt sich $T_{K,\text{parallel,wave}} \approx T_{K,\text{master,wave}}$ und somit kein Vorteil gegenüber dem Modell, das die Datenpakete intern in eine sequentielle Reihenfolge bringt.

Bei idealer Anbindung ohne Engpaß ergibt sich folgender Speedup (ohne Komprimierung):

$$S_v(N) = \frac{T_S + \frac{s}{b_n} + t_l}{f\,T_S + (1-f)\frac{T_S}{N} + \frac{F(s)}{b_p} + \frac{s}{N}\frac{1}{b_{n,p}(N)} + t_l(N)} \tag{4.56}$$

4.6.3 Modell für die Übertragung über zusätzlichen Kommunikationsknoten

Die Übertragung der Daten zur Visualisierung kann durch das Hinzufügen eines zusätzlichen Prozessors von der eigentlichen Simulationsrechnung entkoppelt werden. Dieser zusätzliche Prozessor empfängt auf der einen Seite die Daten von allen anderen Prozessoren der Simulation und überträgt die gesammelten Daten dann weiter zur Visualisierung. Da der zusätzliche Prozessor nicht an der Simulationsrechnung teilnimmt, steht ihm die Zeit T_S auch für die Übertragung zur Visualisierung zur Verfügung. Die Übertragung läuft damit asynchron zur Simulation.

Da der zusätzliche Prozessor nicht für die Simulationsrechnung benötigt wird, muß die externe Übertragungszeit t_n nicht in der Berechnung der Kommunikationszeit berücksichtigt werden. Die

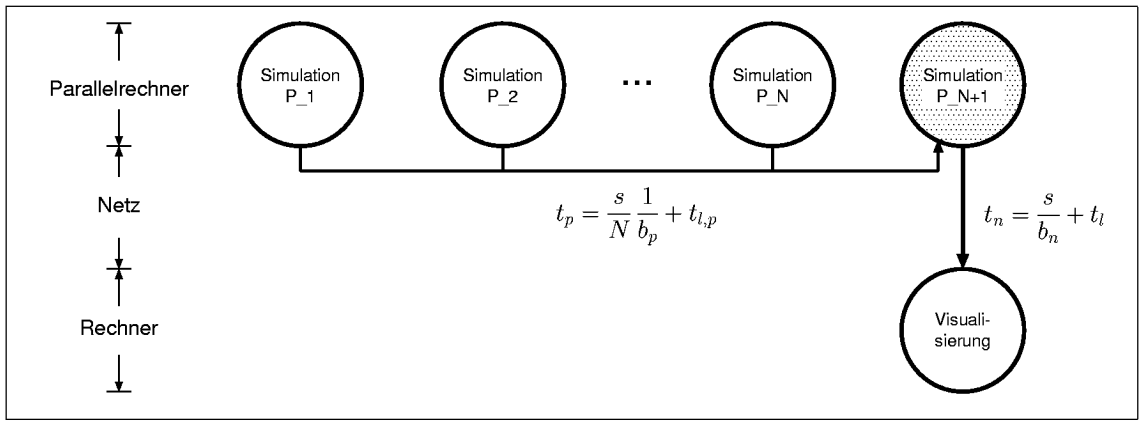


Abbildung 4.8: Netzstruktur bei direkter Übertragung und parallelen Simulationsprogramm über einen zusätzlichen Prozessor: Bevor die Daten zur Visualisierung übertragen werden können, werden sie von einem zusätzlichen Prozessor eingesammelt. Wenn der Speicherbedarf des zusätzlichen Prozessors für die Speicherung aller Daten eines Simulationsschritts ausreicht, kann die Übertragung asynchron zur Simulationsrechnung ablaufen und damit diese Übertragungzeit versteckt werden.

Datenpakete der einzelnen Knoten müssen aber von dem zusätzlichen Knoten nacheinander angenommen werden. Daraus ergibt sich:

$$T_{K,\text{node}} = N\,t_p = N(\frac{s}{N}\frac{1}{b_p} + t_{l,p}) = \frac{s}{b_p} + N\,t_{l,p} \tag{4.57}$$

Dies gilt aber nur, wenn die Übertragungszeit zur Visualisierung nicht größer als die Zeitdauer eines Simulationsschritts ist. Anderenfalls wäre der zusätzliche Prozessor nach der Beendigung des Simulationsschritts noch mit dem Versenden der Daten aus dem letzten Simulationsschritt beschäftigt. Bei allen Prozessoren würde dann wieder eine Wartezeit entstehen. Dann gilt:

$$T_{K,\text{node}} = \begin{cases} \frac{s}{b_p} + N\,t_{l,p} & \text{falls} \quad t_n \le T_P \\ \frac{s}{b_n} + t_l + \frac{s}{b_p} + t_{l,p} & \text{falls} \quad t_n > T_P \end{cases} \tag{4.58}$$

Eine zusätzliche Wartezeit kann dadurch entstehen, daß der zusätzliche Prozessor nicht in der Lage ist, alle Daten lokal zwischenzuspeichern. Dann muß er seinen Speicher zuerst entleeren, indem er die bisher empfangenen Daten direkt zur Visualisierung überträgt. Der Vorteil des asynchronen Datenübertragung ist in diesem Fall aufgehoben. Diese Situation kann teilweise dadurch verhindert werden, daß die Daten bereits auf den einzelnen Prozessoren komprimiert werden. Die Größe der Daten ist damit reduziert und die Wahrscheinlichkeit, daß die Daten eines Simulationsschritts in den Hauptspeicher passen, viel höher. Besitzt der zusätzliche Prozessor sehr viel Hauptspeicher und ist die Zeitdauer eines Simulationsschritts groß genug, kann die Komprimierung der Daten auch auf dem zusätzlichen Prozessor geschehen. Dies ist aber eher ein Ausnahmefall, da meistens die Knoten eines Parallelrechners gleichartig ausgebaut sind.

Wenn jeder Prozessor die Daten vor der Übertragung zum zusätzlichen Prozessor komprimiert und die Zeit für die Kommunikation mit der Visualisierung gering genug ist, ergibt sich folgende Kommunikationszeit:

$$T_{K,\text{node,wave}} = \frac{1}{N}\frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + s_{\text{wave}}\frac{1}{b_p} + N\,t_{l,p} \tag{4.59}$$

Bei der Berechnung des Speedups muß berücksichtigt werden, daß zwar ein Prozessor mehr benötigt wird, dieser aber nicht für die Simulationsrechnung eingesetzt werden kann. Der Speedup berechnet sich für den günstigeren Fall zu:

$$S_{P,v}(N+1) = \frac{T_S + \frac{s}{b_n} + t_l}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + \frac{s}{b_p} + N\,t_{l,p}} \quad \text{für} \;\; t_n \le T_P \tag{4.60}$$

Wenn die Übertragung zur Visualisierung innerhalb eines Simulationsschritts abgeschlossen werden kann, wird der Anstieg der Speedup-Kurve nur durch die Übertragungszeit der Daten innerhalb des Parallelrechners beeinflußt. Da ein Einfluß durch die viel langsamere externe Kommunikation nicht mehr gegeben ist, wird der Speedup weniger stark beeinträchtigt sein als ohne zusätzlichen Knoten.

Im ungünstigen Fall ($t_n > T_P$) ist der Speedup fast konstant, da er nur noch durch die Kommunikation des zusätzlichen Knotens bestimmt wird. Hier hängt nur noch die Latenz für die Kommunikation mit den Rechenknoten von der Prozessoranzahl N ab:

$$S_{P,v}(N+1) = \frac{T_S + t_n}{t_n + N\,t_p} = \frac{T_S + \frac{s}{b_n} + t_l}{\frac{s}{b_n} + t_l + \frac{s}{b_p} + N\,t_{l,p}} \qquad \text{für } t_n > T_P \tag{4.61}$$

Der Schnittpunkt beider Kurven ist durch folgende Gleichung bestimmt:

$$t_n = T_P \Leftrightarrow N_{\max} = \frac{(1-f)\,T_s}{\frac{s}{b_n} + t_l - f\,T_S - \frac{F(s)}{b_p}} \tag{4.62}$$

Je länger die Übertragung der Daten zur Visualisierung dauert, desto früher wird der maximal mögliche Speedup erreicht.

Wenn die Rechenknoten die lokal gespeicherten Daten vor der Übertragung komprimieren, ergeben sich folgende Speedup-Kurven:

$$\begin{aligned} S_{v,\text{wave}}(N+1) &= \frac{T_S + \frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{b_n} + t_l}{f\,T_S + (1-f)\,\frac{T_S}{N} + \frac{F(s)}{b_p} + \frac{s/N}{b_{\text{wave}}} + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{b_p} + N\,t_{l,p}}, \; t_n \le T_P \\ S_{v,\text{wave}}(N+1) &= \frac{T_S + \frac{s}{b_{\text{wave}}} + t_{l,\text{wave}} + \frac{s_{\text{wave}}}{b_n} + t_l}{\frac{s_{\text{wave}}}{b_n} + t_l + \frac{s_{\text{wave}}}{b_p} + N\,t_{l,p}} \qquad t_n > T_P \end{aligned} \tag{4.63}$$

Insbesondere verschiebt sich die Grenze $N_{\max}$ zu größeren Prozessorzahlen hin, da bei der externen Kommunikation nur noch die Größe der komprimierten Daten s_{wave} eingeht:

$$N_{\max,\text{wave}} = \frac{(1-f)\,T_s + \frac{s}{b_{\text{wave}}}}{\frac{s_{\text{wave}}}{b_n} + t_l - f\,T_S - \frac{F(s)}{b_p} - t_{l,\text{wave}}} \tag{4.64}$$

Begrenzender Faktor wird hier dann nicht diese Übertragung, sondern die Rekonstruktion der Daten auf der Visualisierungsseite sein, da dort gegenüber dem Parallelrechner nur ein Prozessor für diese Aufgabe zur Verfügung steht.

4.6.4 Auswirkung der progressiven Übertragung auf die parallelen Modelle

Auch bei der parallelen Ankopplung an ein Simulationsprogramm können die Daten progressiv übertragen werden. In allen drei Fällen der parallelen Ankopplung wird die Komprimierung der Daten auf allen an der Simulationsrechnung beteiligten Prozessoren für den lokal gespeicherten Teil des Simulationsgebiets ausgeführt. Wie bei der Ankopplung an ein serielles Programm werden die Daten durch den Pyramidenalgorithmus in q einzelne Frequenzbänder zerlegt, die in q Schritten nach und nach zur Visualisierung versendet werden können.

Im Durchschnitt wird sich die für die Ankopplung benötigte Kommunikationszeit T_K um den Faktor $1/q$ reduzieren, da die Daten eines Simulationsschritts nun in p Iterationen versendet werden. In der entsprechenden Gleichung für den Speedup des angekoppelten Simulationsprogramms wird die Datengröße s durch s/q ersetzt.

4.6.5 Vergleich der parallelen Modelle

Die in den vorherigen Abschnitten aufgestellten Gleichungen für den Speedup können nun verglichen werden. Dazu zeigt Abbildung 4.9 den Verlauf der einzelnen Speedup-Kurven. Bei der Ankopplung über den Master-Knoten oder über einen zusätzlichen Knoten ist der Speedup jeweils durch die externe Kommunikation begrenzt. Bei der parallelen Ankopplung ist im idealen Fall ohne Engpaß keine Begrenzung durch die externe Kommunikation gegeben. Andernfalls gibt es entweder im Parallelrechner oder auf der Visualisierungsseite einen Engpaß, der dazu führt, daß die externe Bandbreite reduziert wird und das Verhalten der Ankopplung mit der über einen Master-Knoten vergleichbar ist. Gibt es solche idealen Bedingungen, sollte die parallele Ankopplung gewählt werden. Ansonsten sind die beiden anderen Kopplungsstrategien günstiger. Eine ähnliche Untersuchung der Anbindung einer Visualisierung an ein paralleles Simulationsprogramm wurde von Olbrich [53] durchgeführt. Darin werden jedoch weder die Latenzen noch Komprimierungsverfahren berücksichtigt.

Die Speedup-Kurven der beiden verbleibenden Modelle nähern sich für zunehmende Prozessoranzahlen zunächst an einen durch die externe Kommunikation bestimmten Wert. Wie in Abschnitt 4.6.1 diskutiert, ist $N\,t_{l,p}$ für die hier betrachteten N klein gegen t_l, so daß dieser für große N dominierende Term hier keine Rolle spielt.

$$N \gg 1,\ b_p \gg b_n \quad \Rightarrow \quad S_{v,\mathrm{master}}(N) \quad < \frac{T_S + \frac{s}{b_n} + t_l}{f\,T_S + \frac{s}{b_n} + t_l + N\,t_{l,p}} \tag{4.65}$$

$$t_n > T_P \quad \Rightarrow \quad S_{P,\mathrm{node}}(N+1) \quad = \frac{T_S + \frac{s}{b_n} + t_l}{\frac{s}{b_n} + t_l + \frac{s}{b_p} + N\,t_{l,p}} \tag{4.66}$$

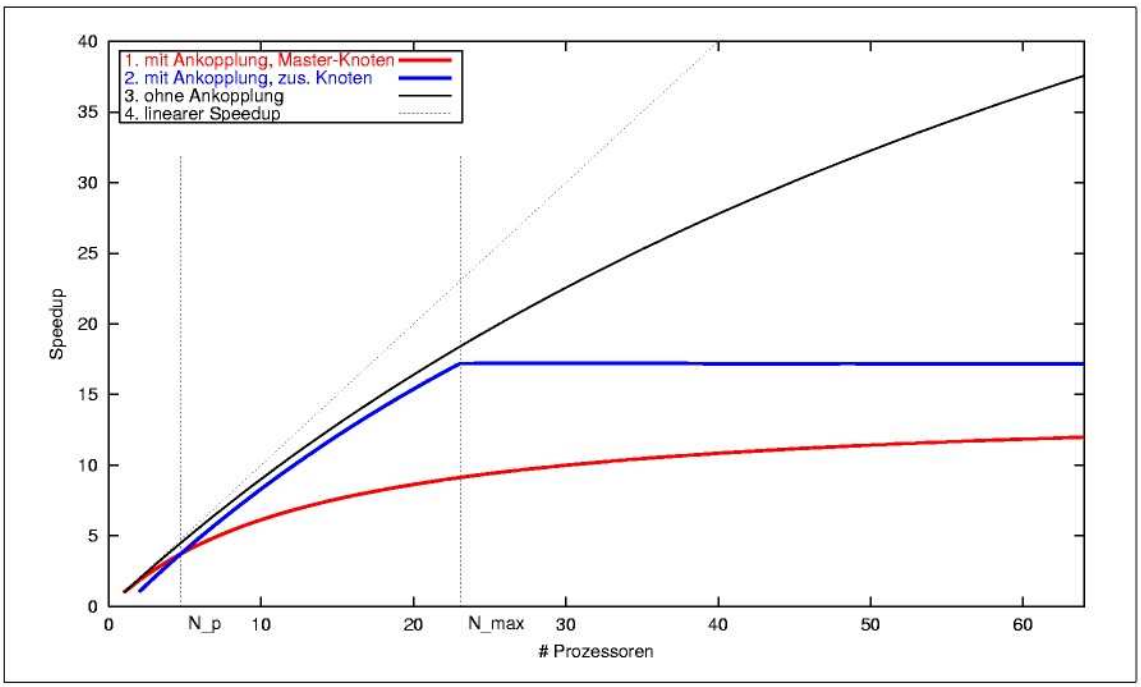


Abbildung 4.9: Speedup-Kurven für die verschiedenen Modelle ohne Komprimierung: Das Diagramm zeigt den Verlauf der Speedup-Kurven für die nicht angekoppelte Simulation, sowie für die Ankopplung über den Master-Knoten oder einen zusätzlichen Knoten. Der Schnittpunkt N_p beschreibt die Prozessoranzahl, ab welcher der Einsatz des zusätzlichen Kommunikationsknotens einen höheren Speedup ergibt. Ab dem Schnittpunkt $N_{\max}$ wird die Kommunikation über den zusätzlichen Knoten von der externen Kommunikation begrenzt, so daß ab dieser Prozessoranzahl der Speedup nicht mehr weiter ansteigt und konstant bleibt. Für die Parameter wurden folgende Werte benutzt: T_S=10.0 s, f=0.001, s = 5.0 MB, b_n= 9 MB/s, t_l= 7 ms, b_p= 100 MB/s, $t_{l,p}$ = 40 μs .

Da bei der Ankopplung über einen zusätzlichen Knoten dieser Grenzwert durch die Zeitdauer der externen Kommunikation bestimmt wird, geht in diese Gleichung der serielle Anteil der Anwendung nicht ein. Genau dieser Anteil unterscheidet die Grenzwerte der beiden Verfahren. Da bei der Ankopplung der Anteil im Nenner erscheint, wird dieser Speedup bei hohen Prozessoranzahlen niedriger sein als der bei einem zusätzlichen Knoten.

Weitere Unterschiede sind im Bereich kleinerer Prozessoranzahlen zu sehen. Die zweite Kurve ($S_{P,\text{node}}$) beginnt erst weiter rechts, da ein zusätzlicher Prozessor benutzt wird, der nicht für die Simulation zur Verfügung steht. Da in dieser Kurve die Übertragungzeit zur Visualisierung nicht einfließt, steigt sie schneller an als die erste Kurve. Von dem Schnittpunkt (N_P) der beiden Kurven an ist der Einsatz eines zusätzlichen Prozessors sinnvoll. Es gilt:

$$N_P = \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{(1-f)\,T_s}{t_{l,p} + \frac{s}{b_n} + t_l}} \tag{4.67}$$

Die beiden markanten Punkte N_P und $N_{\max}$ der beiden Speedup-Kurven verschieben sich z.B. in Richtung Nullpunkt, wenn sich die Zeit für eine sequentielle Iteration oder die Datengröße erhöht. Sie verschieben sich in die entgegengesetzte Richtung, wenn die Bandbreite der externen Kommunikation zunimmt.

Abbildung 4.10 zeigt den Verlauf der Speedup-Kurven, wenn die Daten vor der Übertragungzeit mit dem in dieser Arbeit beschriebenen Verfahren komprimiert werden. Durch die geringere Größe der komprimierten Daten werden die Speedup-Kurven nicht mehr so stark durch die externe Kommunikation beeinträchtigt.

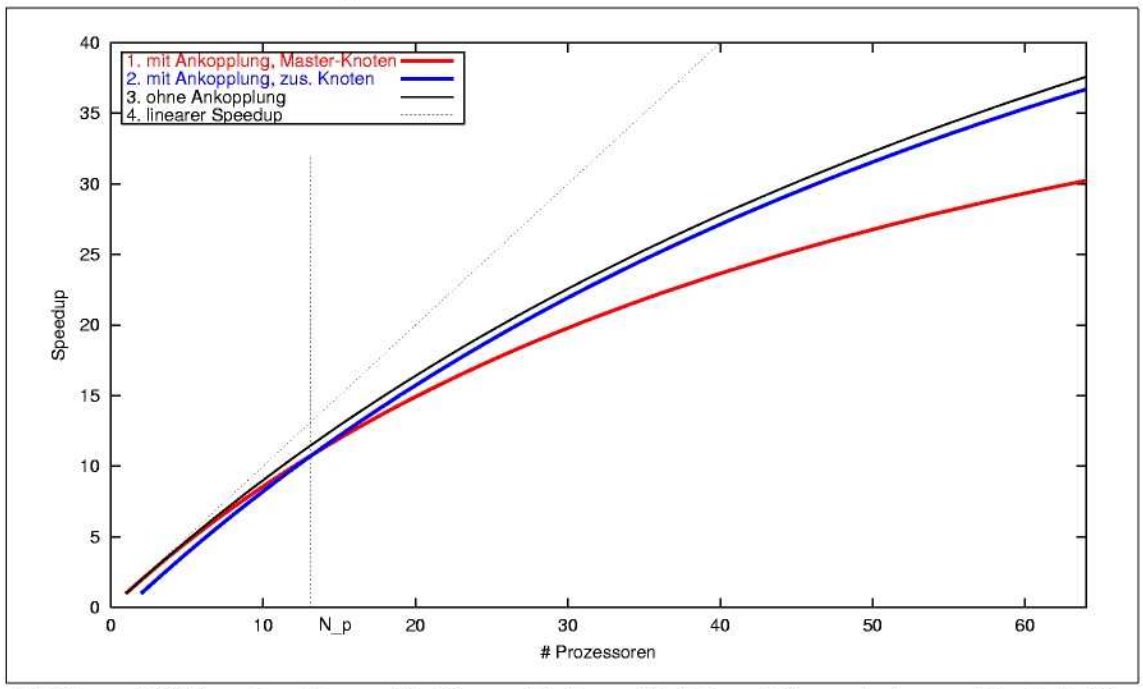


Abbildung 4.10: Speedup-Kurven für die verschiedenen Modelle mit Komprimierung: Durch die Hinzunahme der Komprimierung mit Hilfe des Pyramidenalgorithmus werden die Speedup-Kurven weniger stark beeinflußt. Da nur noch die komprimierten Daten zur Visualisierung übertragen werden müssen, verringert sich diese Übertragungzeit entsprechend, so daß die Speedup-Kurve des Modell mit zusätzlichem Knoten erst bei einer sehr viel höheren Prozessoranzahl ggf. begrenzt wird. Für die Parameter wurden folgende Werte benutzt: T_S=10.0 s, f=0.001, s = 5.0 MB, b_n= 9 MB/s, t_l= 7 ms, b_p= 100 MB/s, $t_{l,p}$ = 40 μs , b_{wave}=20 MB/s $s_{\text{wave}} = 0.1 * s$, $t_{l,\text{wave}}$= 1 ms.

Im Gegensatz zu dem ersten Modell kann bei der Verwendung des zusätzlichen Prozessors die Abflachung der Speedup-Kurve verhindert werden, indem nur dann Daten von dem Simulation-

prozessoren zu dem zusätzlichen Prozessor übertragen werden, wenn dieser im *Idle*-Zustand ist. Ansonsten werden die Daten des aktuellen Simulationsschritts nicht zur Visualisierung übertragen. Die Aktualisierungsrate der Visualisierung ist damit geringer als die Rate, in der von der Simulation neue Datensätze erzeugt werden.

Die Verwendung eines zusätzlichen Prozessors macht nur dann Sinn, wenn die Visualisierung zu einem großen Teil der Rechenzeit an die Simulation angekoppelt ist. Ansonsten wäre er ungenutzt und blockiert damit Ressourcen des Parallelrechners. Wird die Visualisierung nur sporadisch an die Simulation angekoppelt, sollte das erste Modell vorgezogen werden, da in diesen kurzen Zeiten zwar die Simulation stärker beeinträchtigt wird, diese insgesamt aber nur wenig beeinflußt wird.

Die Hinzunahme eines weiteren Prozessors kann bei Simulationsprogrammen, die 2^n Prozessoren verwenden müssen, und darauf angepaßten Batch-Klassen zu einer großen Fragmentierung des Rechners führen (z.B. CRAY T3E in Jülich). Durch den zusätzlichen Prozessor kann das Programm nun nicht mehr in der Batch-Klasse mit 2^n Prozessoren gestartet werden, sondern muß in der nächst höheren Batch-Klasse mit 2^{n+1} Prozessoren gestartet werden, bei der im allgemeinen längere Wartezeiten auftreten. Abhilfe könnte hier die in MPI-2 implementierte Funktionalität zur dynamischen Anbindung weiterer Prozessoren an eine laufende Anwendung bringen.

Nachdem im nächsten Kapitel der Entwurf und die Implementierung der in dieser Arbeit entwickelten Werkzeuge zu der hier beschriebenen Kopplung und Datenreduzierung vorgestellt werden, werden Messungen durchgeführt, um zum einen die lineare Abhängigkeit der verschiedenen Verarbeitungszeiten zu überprüfen, und zum anderen die in diesem Kapitel definierten Modellparameter auch quantitativ für eine typische Simulationsumgebung zu bestimmen.

Kapitel 5

Entwurf und Implementierung der Kopplung

Bei dem Entwurf und der Implementierung der für die Kopplung einer Visualisierung an ein paralleles Simulationsprogramm nötigen Komponenten stehen zwei Bereiche im Blickfeld: Zum einen müssen die in Kapitel 3 beschriebenen Teilschritte der Datenkomprimierung und Datenreduktion in den Übertragungsweg der Daten eingebracht werden. Zum anderen entstehen durch die Datenreduktion, die progressive Übertragung und insbesondere durch die Ankopplung an parallele Programme eine Vielzahl von Datenströmen, die auf beiden Seiten der Übertragung koordiniert und kontrolliert werden müssen.

Für die Datenübertragung wird in dieser Arbeit die Kommunikations-Bibliothek VISIT benutzt (siehe auch Abschnitt 2.7.1 auf Seite 15). Sie stellt für die Datenübertragung die unterste Schicht dar. Auf sie setzen die in dieser Arbeit entwickelten Übertragungsschichten für komprimierte und verteilte Daten auf. Für die Visualisierung der Daten wird das in der wissenschaftlichen Visualisierung weit verbreitete Werkzeug AVS/Express [25] eingesetzt, für das eine Schnittstelle zu VISIT existiert. Im Abschnitt 5.1 werden die für die Implementierung wichtigen Details beider Tools vorgestellt.

Die im Rahmen dieser Arbeit entwickelten, auf VISIT [2] aufsetzenden Schichten für die Datenkomprimierung und Verwaltung von verteilten Daten sind in der Bibliothek LVISIT zusammengefaßt (Abschnitt 5.2) und für die Ankopplung an parallele Simulationsprogramme erweitert (Abschnitt 5.3). Um die Schnittstelle zu dieser Bibliothek auf beiden Seiten möglichst einfach zu halten, werden Teile der Bibliothek dynamisch an die im Simulationsprogramm benutzten Daten angepaßt. Zu diesem Zweck wurde ein Code-Generator entwickelt, der aus einer gemeinsamen Spezifikation der für die Kopplung nötigen Datenströme eine angepaßte Schnittstelle für das Simulationsprogramm, entsprechende AVS/Express-Makros und eine Beispiel-Applikation für die Einbettung in die Visualisierung generiert (Abschnitt 5.4). Der Code-Generator garantiert eine konsistente Generierung der Schnittstellen auf beiden Seiten der Kommunikation und ermöglicht so eine einfache Anwendung der Bibliothek durch den Benutzer.

5.1 VISIT und AVS/Express

Das Visualization Interface Toolkit VISIT ermöglicht die Übertragung von skalaren Werten oder mehrdimensionalen Feldern verschiedener Datentypen zwischen zwei verschiedenen Programmen (siehe auch Abschnitt 2.7.1 auf Seite 15). Die Visualisierung übernimmt bei der Übertragung die Rolle des Servers, der auf Daten oder Kontrollinformationen von der Simulation (Client) wartet. Für die Einbettung in das Simulationsprogramm stellt VISIT im wesentlichen die Funktionen

`visit_connect`, `visit_disconnect`, `visit_send` und `visit_recv` zur Verfügung. Die ersten beiden Funktionen dienen zum Auf- bzw. Abbau der Verbindung. Mit den beiden anderen Funktionen können Daten zur Visualisierungsseite gesendet oder von dort angefordert werden. Für die Datenübertragung müssen diese durch eine Identifikation-Nummer (Id), den Datentyp, ihre Größe und einen Zeitstempel (Simulationzeit) genauer spezifiziert werden. Durch die Id können verschiedene Daten auf der Empfängerseite unterschieden und somit bei der Übertragung verschiedene Datenströme genutzt werden. Durch die Angabe des Datentyps ist VISIT in der Lage, die Daten auf das interne Datenformat der Zielplattform zu konvertieren. Dies wird zum Beispiel beim Wechsel vom Zahlenformat Big nach Little Endian bei numerischen Daten notwendig. Der Zeitstempel kann für Zuordnung der Daten zu einem Simulationsschritt (Simulationszeit) genutzt werden. Abbildung 5.1 zeigt die Einbettung in ein Simulationsprogramm.

```
vcd=visit_connect(service,...);
...
while(SimTime) {
  /* id=2, dim=1 */
  visit_recv(vcd,2,SimTime,&parm,VISIT_INT,1,1)
  work(...);
  /* id=1, dim=3 */
  visit_send(vcd,1,SimTime,data,VISIT_DOUBLE,3,nx,ny,nz);
}
...
visit_disconnect(vcd);
```

Abbildung 5.1: Einbettung der VISIT-Funktionen in ein Simulationsprogramm: Vor dem Beginn der Simulationsrechnung (While-Schleife) wird die Verbindung zur Visualisierung aufgebaut. Dabei wird eine sowohl in der Visualisierung als auch im Simulationsprogramm bekannte Bezeichnung (`service`) angegeben, über die eine eindeutige Zuordnung zwischen den beiden Kommunikationspartnern möglich ist. Innerhalb der Schleife werden zu Beginn neue Parameterwerte von der Visualisierung gelesen und am Ende der Schleife die zuvor berechneten Daten zur Visualisierung übertragen.

Auf der Visualisierungsseite wird von VISIT neben einer C- und Perl-Schnittstelle auch eine Einbettung der Server-Seite in AVS/Express angeboten. Dabei handelt es sich nicht um Funktionsauf-

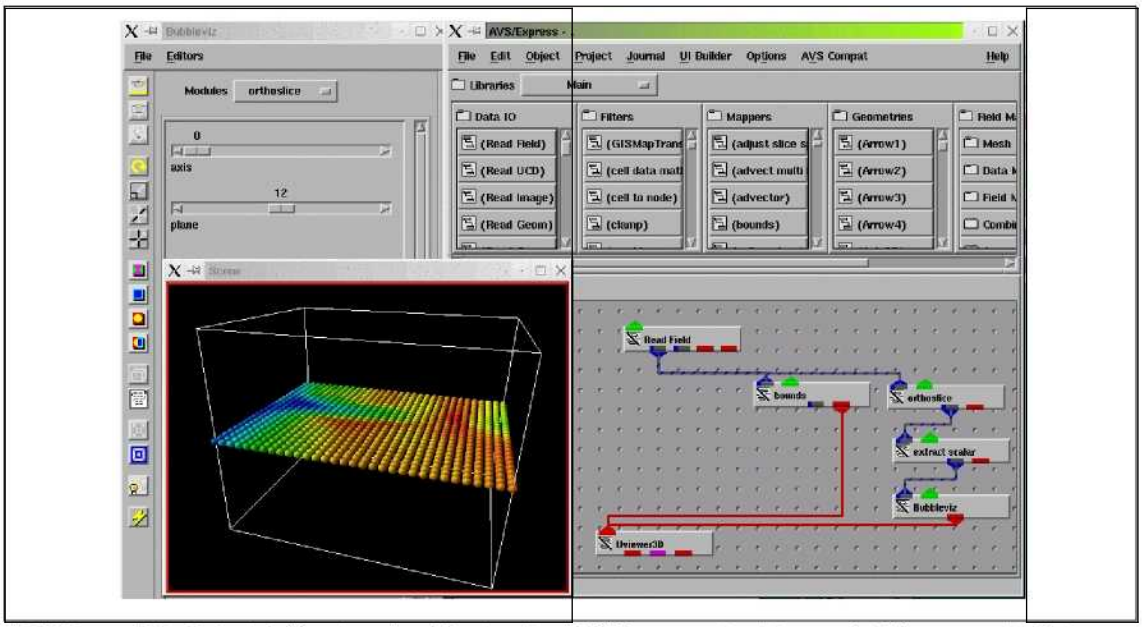


Abbildung 5.2: Beispiel für eine Applikation in AVS/Express: Im Netzwerk-Editor von AVS/Express (rechts unten) ist der Datenfluß für eine einfache Anwendung zu erkennen. Von dem Einlese-Modul (Read_Field) gelangen die Daten über verschiedene Verarbeitungsmodule (z.B. orthoslice) zu dem Anzeige-Modul (Uviewer). Jedes Modul kann in einem separaten Fenster GUI-Elemente besitzen, über die modulspezifische Einstellungen vorzunehmen sind.

rufe, sondern vielmehr um Module, die in das Datenfluß-Modell von AVS/Express eingebunden werden können [25]. Abbildung 5.2 auf der vorherigen Seite zeigt eine Beispiel-Applikation in AVS/Express.

In der Abbildung 5.3 ist das Netzwerk so verändert worden, daß die Daten nicht mehr von einer Datei gelesen, sondern von VISIT-Modulen geliefert werden. Die Visit-Module von AVS/Express sind in der Programmiersprache C implementiert und nutzen intern die C-API von VISIT. Für jeden Datenstrom, der auf der Simulationsseite durch eine Id gekennzeichnet wird, muß auf der Visualisierungsseite ein `visitreader`- oder `visitwriter`-Modul vorhanden sein, das die Daten entgegennimmt bzw. zurückgibt. Das Lese-Modul von VISIT stellt die Daten an einem der Ausgänge des Moduls zur Verfügung. Die Ausgänge des Moduls unterscheiden sich durch den Datentyp der auszugebenden Daten und deren Dimension (Skalar, Feld). Damit muß bei der Erstellung des Netzwerks sichergestellt werden, daß zu jeder von der Simulation genutzten Id ein Lese- bzw. Schreib-Modul in das Netzwerk eingebunden wird und die richtigen Ausgänge der Module gewählt werden.

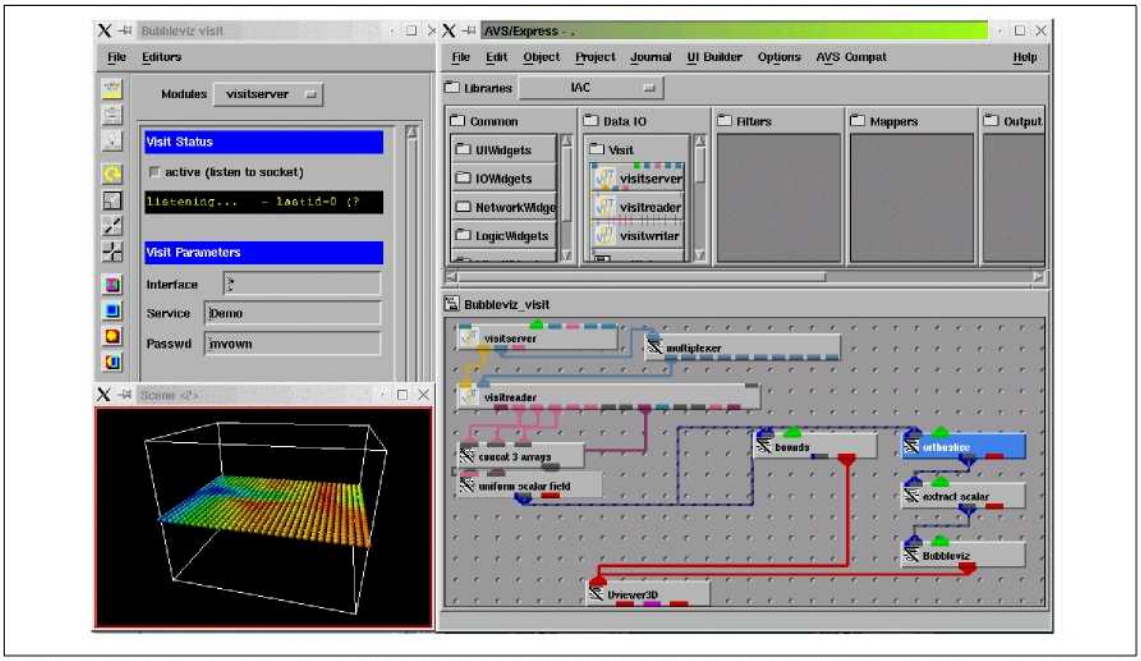


Abbildung 5.3: Beispiel für eine VISIT-Applikation in AVS/Express: In dem zu dieser Applikation gehörenden Netzwerk ist das Einlese-Modul durch VISIT-Module ersetzt worden. Das `visitserver`-Modul ist für die Verwaltung der Verbindung zum Simulationsprogramm zuständig. Wird vom Simulationsprogramm eine Übertragung von Daten initiiert, empfängt dieses Modul die zu den Daten gehörende Id und aktiviert über den Multiplexer das zugehörigen `visitreader`-Modul. Dieses empfängt die Daten und gibt sie an die Verarbeitungs-Module von AVS/Express weiter. Links oben ist die Eingabemaske des Visitserver-Moduls zu sehen, mit dem u.a. der Service-Name verändert werden kann. Rechts oben stellt AVS/Express in seiner Modulbibliothek auch die VISIT-Module für die Konstruktion von Netzwerken zur Verfügung.

5.2 Bibliothek für die Datenreduzierung (LVISIT)

Für die Datenreduzierung müssen vor dem Versenden und nach dem Empfangen der Daten Verarbeitungsschritte zwischengeschaltet werden. Auf der Simulationsseite bedeutet dies, daß die Visit-Funktionen durch die entsprechenden, in dieser Arbeit entwickelten LVISIT-Funktionen ersetzt werden. Diese reduzieren und komprimieren die Daten und versenden sie anschließend mit Hilfe der VISIT-Funktionen. Abbildung 5.4 auf der nächsten Seite beschreibt die erweiterte Ar-

chitektur der Kopplungsbibliothek. Innerhalb von AVS/Express können die VISIT-Module zum Verwalten der Verbindung und zum Senden und Empfangen der Daten weiter genutzt werden. Nur bei den komprimierten Datenströmen müssen zwischen die `visitreader`-Module und die Verarbeitungs-Module von AVS/Express LVISIT-Module zwischengeschaltet werden, welche die Daten wieder entkomprimieren.

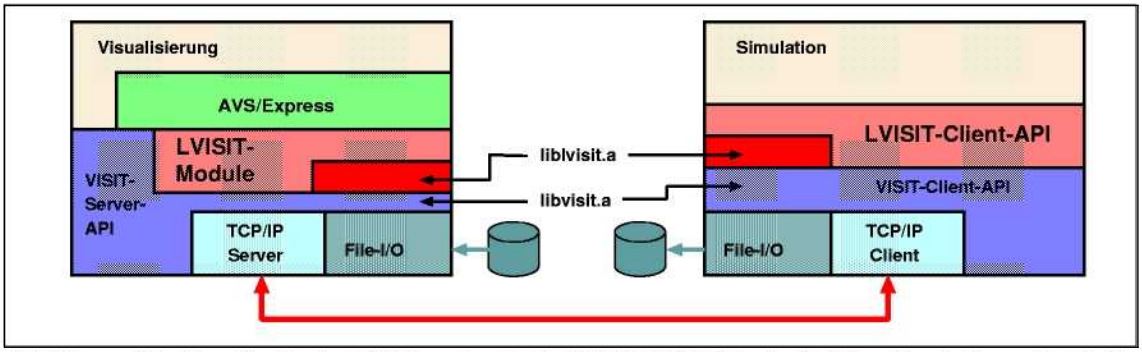


Abbildung 5.4: Erweiterte Architektur durch die LVISIT-Bibliothek: Auf der Simulationsseite ist für das Simulationsprogramm nur die Schnittstelle der neuen Bibliothek LVISIT sichtbar. In AVS/Express werden neben den VISIT-Modulen nun auch LVISIT-Module bereitgestellt, welche die Daten vor der Übergabe an die AVS/Express-Module verarbeiten. Auf beiden Seiten werden Routinen aus der LVISIT-Bibliothek benutzt. Die Schnittstelle auf der Simulationsseite enthält dynamisch generierte Wrapper-Routinen, die zusätzlich zum Programmcode und den Bibliotheken hinzugeladen werden müssen.

Die einzelnen Verarbeitungsschritte der Datenreduktion erlauben eine Vielzahl von Steuerungsmöglichkeiten. So kann z.B. die Anzahl der Iterationen des Pyramidenalgorithmus oder die Schranke für das Abschneiden von kleinen Koeffizienten variiert werden. Auch für die Datenübertragung gibt es verschiedene Modi: die Daten können direkt, komprimiert oder progressiv übertragen werden. Bei der Implementierung wurde Wert darauf gelegt, daß der Anwender diese für die Nutzung der Bibliothek wichtige Parameter sehr einfach interaktiv (in AVS/Express) einstellen kann. Diese Parameter müssen vor der eigentlichen Übertragung der Daten von der Visualisierungsseite zur Simulation übermittelt werden. Folgende Tabelle führt die in dieser Arbeit implementierten und für die Datenreduktion notwendigen Parameter auf.

Name	zul. Werte	Beschreibung
type	direkt, wtn, progressive	Art der Übertragung (wtn: Wavelet-Transformierten; progressive: Übertragung einzelner Frequenzbänder)
cut	0..100%	Schranke für das Abschneiden von kleinen Koeffizienten
cutmethod	hist, no_hist	Bestimmung des Schwellwert für das Abschneiden von kleinen Koeffizienten über ein Histogramm
iter	1-5	Anzahl der Iterationen des Pyramidenalgorithmus
wtstep	pwt*i*	Auswahl einer Wavelet-Funktion, Haar (2), Daubechies (4-32)
nquant	8-16	Anzahl der Bits bei der Quantisierung
quantcorr	ja/nein	Gewichtung der Rekonstruktionswerte über ein Histogramm
quantfunction	linear/log	linear oder logarithmische Abbildung der Werte auf die Teilintervalle bei der Quantisierung
wtndebug	ja/nein	Debug-Modus für die Wavelettransformation
active[6]	ja/nein	Auswahl von Frequenzbändern

Die oben beschriebenen Parameter sind in LVISIT in der Datenstruktur `wdesc` zusammengefaßt. Sie enthält zusätzlich noch einige weitere Einträge, die z.B. die Größe und den Status der komprimierten Daten beschreiben. Die Datenstruktur wird zusätzlich zu der Übertragung der Parameter-Einstellung zur Simulation auch für die Beschreibung der komprimierten Daten bei der Übertragung zur Visualisierung benutzt.

Bei der Datenkomprimierung entstehen zwei verschiedene Datenströme. Nach der Quantisierung und Kodierung stehen die Daten selbst als eine Folge von Byte-Werten für die Übertragung bereit. Die Quantisierung erzeugt zusätzlich eine Folge von Gleitpunktzahlen, welche für die Rekonstruktion der Daten benötigt werden (gewichtete Mittelwerte der Quantisierung). Diese Double-Werte können nicht an den Byte-Strom angehängt werden, da sich bei einer Übertragung zu einer anderen Maschine ggf. die interne Zahlendarstellung ändert und VISIT den Byte-Strom ohne Konvertierung überträgt. Insgesamt besteht die Übertragung komprimierter Daten damit aus mehreren einzelnen Übertragungen, deren Ablauf in Abbildung 5.5 dargestellt ist.

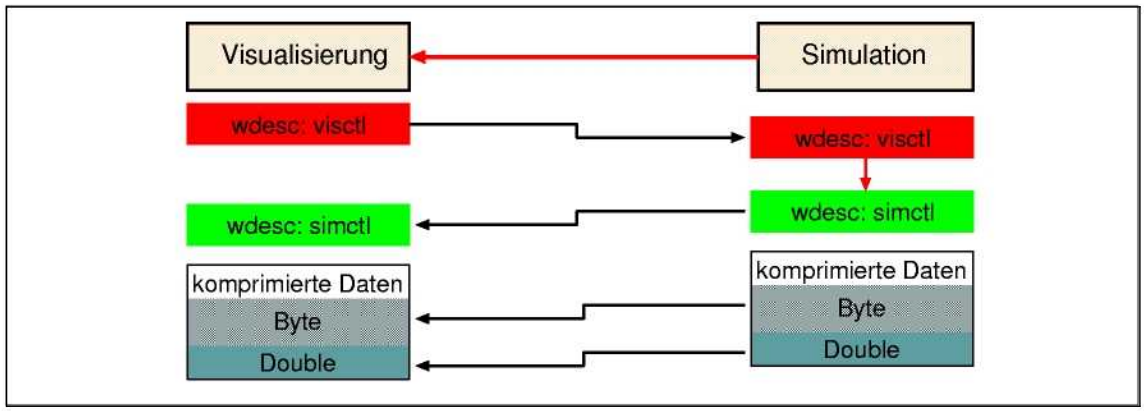


Abbildung 5.5: Übertragung komprimierter Daten mit LVISIT: Die wdesc-Datenstruktur wird bei der Übertragung von komprimierter Datensätzen zweimal benutzt. Zum einen werden mit ihr die auf der Visualisierungsseite eingestellten Parameter für die Komprimierungsroutinen bei einer Anforderung der Simulation zu dieser übertragen (visctl). Zum anderen beschreibt sie nach der Komprimierung die dabei verwendeten Einstellungen (simctl).

Auf der Visualisierungsseite muß das Netzwerk von AVS/Express um ein Schreib- und ein Lese-Modul erweitert werden. Das erste dient dazu, die wdesc-Datenstruktur zum Simulationsprogramm zu übertragen. Das zweite Modul empfängt die wdesc-Struktur, welche die nachfolgenden komprimierten Daten beschreibt. Für die Einstellung der Parameter in der wdesc-Struktur steht dem Benutzer ein Panel zur Verfügung (siehe Abb. 5.6 auf der nächsten Seite). Da dort auch die Einträge der wdesc-Struktur während der Übertragung geändert werden, kann diese lokale Struktur nicht für die Rekonstruktion der auf der Simulationsseite komprimierten Daten genutzt werden. Eine Konsistenz zwischen der wdesc-Struktur und den Daten ist nur durch die Rückübertragung der benutzen Einstellungen zu erreichen.

Bei der Anwendung des Pyramidenalgorithmus werden die Daten in verschiedene Frequenzbänder aufgeteilt, die bei der einfachen Übertragung zusammen und bei der progressiven Übertragung nacheinander in mehreren Simulationsschritten zur Visualisierung übertragen werden. Das Modul `waveletprogressive` wird in beiden Fällen für die Rekonstruktion der Daten benutzt. Als Eingabe dient diesem Modul neben den komprimierten Daten (Byte, Double) auch die von der Simulation gelieferte wdesc-Struktur. Die verschiedenen Auflösungsstufen der Frequenzbänder spiegeln sich in den verschiedenen Größen der Daten an den Ausgängen des Moduls wieder. Je nach Art der Übertragung und der Anzahl der Iterationen des Pyramidenalgorithmus werden dort die Daten in den entsprechenden Auflösungstufen nach der Ankunft eines neuen Datenpaketes und dessen Entkomprimierung zur Verfügung gestellt. Mit Hilfe der beiden anderen LVISIT-Makros `WTprogcontrol` und `switchchannel` kann die Anzahl der darzustellenden Frequenzbändern ausgewählt und somit die Auflösung der Darstellung gesteuert werden.

Die Abbildung 5.6 auf der nächsten Seite zeigt die aufwendige Verknüpfung der einzelnen Module. Für die Übertragung eines komprimierten Datensatzes müssen mehrere VISIT-Lese- und Schreib-Module und einige LVISIT-Module miteinander verknüpft werden. Es muß weiterhin

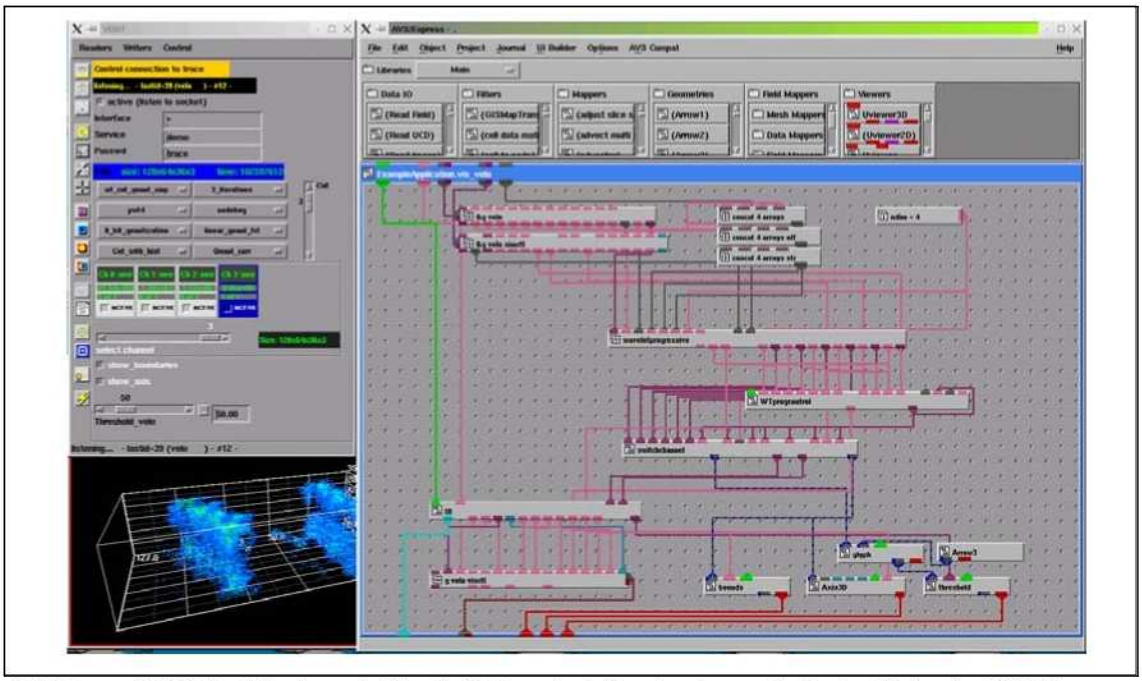


Abbildung 5.6: Beispielnetzwerk für die Rekonstruktion der komprimierten Daten in AVS/Express: Die Daten und die wdesc-Struktur werden von den VISIT-Modulen in den AVS/Express-Groups `g_velo` und `g_vel_simctl` abgespeichert und in diesem Teil des Netzwerks dem LVISIT-Modul `wavelet-progressive` übergeben. Das Makro `UI` erzeugt und verwaltet die Eingabemaske für die Komprimierungseinstellungen (links). Das Makro `WTprogcontrol` zeigt in der linken Eingabemaske die Daten zu den einzelnen Frequenzbändern der durch den Pyramidenalgorithmus zerlegten Daten an. Eines dieser Frequenzbänder kann für die Anzeige ausgewählt werden; diese Aufgabe übernimmt das Makro `switch-channel`.

dafür gesorgt werden, daß die für die Übertragung genutzten Ids auf der Simulations- und der Visualisierungsseite übereinstimmen. Um diese fehleranfällige Aufgabe zu automatisieren, wurde im Rahmen dieser Arbeit der Code-Generator `visitcg` entwickelt. Er nutzt die Eigenschaft von AVS/Express aus, daß die Netzwerke in einer eigenen einfachen Programmiersprache *V* abgespeichert werden. Der Code-Generator ist so in der Lage, Teilnetze für die Verarbeitung der komprimierten Daten zu liefern, die in AVS/Express zu Makros zusammengefaßt sind. Darüber hinaus generiert `visitcg` auch eine komplette Beispiel-Applikation, welche die Daten von der Simulation entgegennehmen und anzeigen kann. Diese Beispiel-Applikation kann als Ausgangsbasis für die weitere Entwicklung der Visualisierungs-Applikation genutzt werden. Der Generator benutzt für die Erzeugung eine Konfigurationsdatei, welche die einzelnen Datenströme der Kopplung beschreibt. Der Generator verwendet diese Datei auch für die Erzeugung einer an die dort aufgeführten Datenströme angepaßten Schnittstelle auf der Simulationsseite. Die einzelnen Datenströme werden in der Datei mit Namen versehen und in den Schnittstellen über diese identifiziert, so daß die Ids für den Benutzer verborgen bleiben und Fehler durch eine falsche Zuordnung der Datenströme (Ids) vermieden werden. Da die Konfigurationsdatei des Code-Generators für die Generierung aller Schnittstellen genutzt wird, ist sie die einzige Beschreibung der zu übertragenden Daten, die vom Benutzer geliefert werden muß.

Abbildung 5.7 auf der nächsten Seite veranschaulicht die Einbettung der LVISIT-Schnittstelle in ein Simulationsprogramm. Gegenüber der Schnittstelle von VISIT (siehe Abb. 5.1) wird in jedem Schleifendurchlauf durch die Routine `lvisit_trace_check_connection` überprüft, ob eine geöffnete Verbindung zu der Visualisierung existiert oder ob eine neue Verbindung aufgebaut werden soll. Besteht keine Verbindung zu einer Visualisierung, werden die Aufrufe der LVISIT-Routinen zum Senden und Empfangen der Daten direkt beendet. Mit diesem Mechanismus ist es

```
...
lvisit_trace_init();
while(SimTime) {
  lvisit_trace_check_connection();
  lvisit_trace_parm_recv(&parm);
  work(...);
  lvisit_trace_velo_send(data_velo,nx,ny,nz,3);
}
...
lvisit_trace_close();
```

Abbildung 5.7: Einbettung der LVISIT-Funktionen in ein Simulationsprogramm: Die LVISIT-Funktionen sind dynamisch generiert und an die in der Konfigurationsdatei des Code-Generators spezifizierten Datenströme angepaßt. In den Funktionsnamen ist daher der Name der Kopplung (`trace`) und bei den Send- und Receive-Funktionen der Name des Datenstroms enthalten (`velo` bzw. `parm`). Daher müssen für die Spezifikation der Daten keine Ids mehr verwendet werden. Der für den Verbindungsaufbau benötigte Service-Name wird nicht mehr fest im Code angegeben, sondern von der Init-Routine aus einer Konfigurationsdatei (`trace.rc`) gelesen.

möglich, daß sich eine Visualisierung erst während der Laufzeit eines Simulationsprogramms an dieses ankoppelt. Dies ist mehrfach und durch den Einsatz des SEAP-Servers sogar von verschiedenen Rechnern aus möglich (siehe auch Abb. 2.9 auf Seite 16).

Um eine Anpassung der Bibliothek an andere Systeme zu ermöglichen, ist der größte Teil der Funktionalität in die LVISIT-Bibliothek verlagert worden, so daß der für das Visualisierungssystem spezifische Programmcode einen geringen Umfang hat. Auch auf der Simulationsseite werden von den dynamisch generierten Wrapper-Routinen im wesentlichen Funktionen der LVISIT-Bibliothek aufgerufen. Das sin insbesondere die Funktionen `lvisit_wavelet` (Simulationsseite) und `lvisit_invwavelet` (Visualisierungsseite). Sie sind für die Komprimierung bzw. Entkomprimierung der Daten zuständig. Abbildung 5.8 veranschaulicht das Prinzip. Diese beiden Funktionen benutzen dann für die Konvertierung der Daten weitere Routinen aus der LVISIT-Bibliothek.

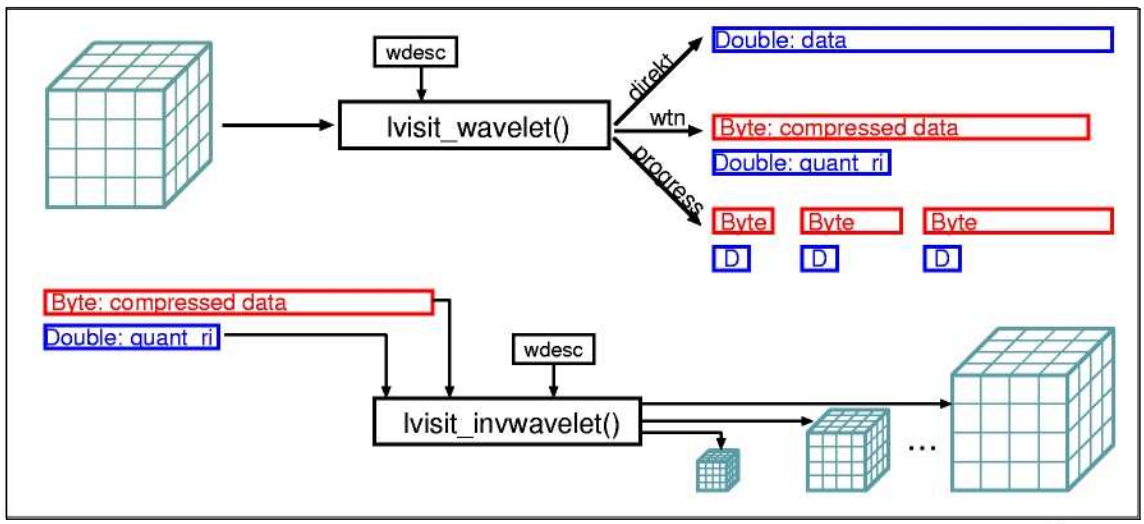


Abbildung 5.8: Umwandlung der Daten durch die LVISIT-Funktionen: Je nach gewünschter Übertragungsart werden die Daten in verschiedene Pakete von Double-Zahlen bzw. Byte-Werten zerlegt. Bei der progressiven Übertragung werden die Felder zusätzlich auf die einzelnen Frequenzbänder des Pyramidenalgorithmus aufgeteilt, so daß die Bänder einzeln zur Visualisierung übertragbar sind. Auf der Empfängerseite werden aus den beiden Datenströmen die Daten wieder rekonstruiert und bei einer Aufteilung in verschiedene Frequenzbänder auf verschiedene Ausgabefelder verteilt.

Die wichtigsten Routinen der LVISIT-Bibliothek davon sollen kurz vorgestellt werden:

`wtn()` führt mehrere Iterationen des Pyramidenalgorithmus durch. Da die inverse Wavelet-Transformation bei der FWT nach dem gleichen Prinzip, nur mit einem anderen Filterpaar durchgeführt wird, kann diese Routine sowohl auf der Simulations- als auch auf Visualisierungsseite benutzt werden. Die Richtung der Transformation wird dabei als Parameter übergeben. Bei der Implementierung wurde als Basis die entsprechende Routine aus „Numerical recipes in C“ [54] benutzt. Sie ermöglicht, daß für die Transformation das Filterpaar während der Laufzeit ausgewählt werden kann. Für den Einsatz im Computational Steering wurde die Performance dieser Routine bei der Behandlung der Ränder und dem Kopieren der Daten optimiert.

`analyse_data()` analysiert die Original- bzw. die Wavelet-transformierten Daten und berechnet ggf. Histogramme, welche die Verteilung der Daten oder Koeffizienten über ihren Wertebereich beschreiben.

`cut_coeff()` setzt in den Wavelet-transformierten Daten diejenige Koeffizienten auf Null, die unter einer vorgegebenen Schranke liegen. Diese Schranke wird ggf. mit dem von `analyse_data()` berechneten Histogramm bestimmt.

`band_compress_quant()` führt die restlichen Schritte der Datenkomprimierung aus, welche aus Speicherplatzgründen in einem Durchlauf ausgeführt werden. Im einzelnen sind dies die Quantisierung, die Kodierung und die Komprimierung des Byte-Datenstroms. Für die Komprimierung wurde das Paket `minilzo` [55] benutzt, das eine sehr schnelle Komprimierung und Entkomprimierung von Daten mit Hilfe eines Lempel-Ziv-Algorithmus erlaubt (siehe auch Abschnitt 3.3.1 auf Seite 41).

Mit Hilfe der in dieser Arbeit entwickelten LVISIT-Bibliothek und des Code-Generators werden alle nötigen Hilfsmittel für eine konsistente und einfache Kopplung zwischen einer in AVS/Express implementierten Visualisierung und einem Simulationsprogramm bereitgestellt. Bisher wurde nur die Ankopplung an ein serielles oder an einen Knoten eines parallelen Simulationsprogramms vorgestellt. Im nächsten Abschnitt wird auf die Erweiterungen von LVISIT und des Code-Generators für die Anbindung von parallelen Simulationsprogrammen eingegangen.

5.3 Erweiterungen für die Ankopplung an parallele Programme

Zur Unterstützung der Ankopplung an parallele Simulationsprogramme mit verteilter Datenhaltung sind im wesentlichen auf der Simulationsseite Erweiterungen in der Schnittstelle notwendig. In der LVISIT-internen Verarbeitung sind auf beiden Seiten Erweiterungen notwendig, da nun die Daten als Teilgebiete zur Visualisierung übertragen und dort wieder zusammengesetzt werden.

Im Abschnitt 4.6 „Ankopplung an parallele Simulationsprogramme“ wurden drei verschiedene Möglichkeiten der Ankopplung an ein paralleles Simulationsprogramm aufgezeigt. Dies sind die Übertragung der Daten über den Master-Knoten oder über einen zusätzlichen und nur für die Kommunikation zuständigen Knoten sowie die parallele Ankopplung aller Knoten an die Visualisierung.

Bei der Implementierung wurde davon ausgegangen, daß das Simulationsprogramm mit Hilfe der weit verbreiteten MPI-Bibliothek parallelisiert worden ist. Für die Anbindung an Programme, die mit anderen Kommunikationsbibliotheken parallelisiert worden sind, ist eine Umstellung der LVISIT-internen Kommunikation auf die jeweilige Kommunikationsbibliothek nötig.

Für die parallele Ankopplung kann eine Erweiterung der VISIT-Bibliothek genutzt werden, die es erlaubt, mehrere VISIT-Clients mit einem VISIT-Server zu verbinden. Dabei verhält sich das AVS/Express-Modul `visitserver` so wie bei einer einfachen Ankopplung. Senden mehrere

Clients gleichzeitig Daten zu diesem Server, werden diese sequentiell entgegengenommen und an die VISIT-Lese-Module weitergegeben. Bei dieser Anbindung, aber auch bei der Anbindung über einen Rechenknoten bzw. zusätzlichen Knoten des Simulationsprogramms muß die Herkunft der Daten bzw. deren Position im Gesamtgebiet mitgeliefert werden. Bei der Übertragung der komprimierten Daten werden diese Informationen mit in den Byte-Datenstrom eingebunden, so daß beim Entpacken der Daten diese direkt an der richtigen Position in den Ausgabedaten abgelegt werden können. Bei der direkten, nicht komprimierten Übertragung werden diese Informationen in zusätzliche, von VISIT unterstütze Felder der Nachricht (*offset* und *stride*) eingetragen und auf der Empfängerseite wieder ausgelesen.

Da auf der Visualisierungsseite bei der Ankopplung an ein Simulationsprogramm mit verteilter Datenhaltung die Daten in Teilpaketen empfangen und wieder zusammengesetzt werden müssen, ist es notwendig alle Routinen so zu erweitern, daß sie auf einem Ausschnitt eines Feldes arbeiten können. Dazu erhalten die Routinen neben der Größe des Gesamtgebiets zusätzlich die Größe und Anfangsposition des Ausschnitts. Da die Daten erst auf der Visualisierungsseite zusammengefügt werden, müssen die Routinen zur Entkomprimierung, inversen Quantisierung und inversen FFT für jedes Teilgebiet aufgerufen werden. Insgesamt bleibt im Vergleich zur Ankopplung an ein serielles Simulationsprogramm der Aufwand in der gleichen Größenordnung, da die Größe der Teilgebiete in der Summe wieder die Größe des Gesamtgebietes ergibt.

Im Gegensatz zu den Parametern der Komprimierung kann die Kopplungsstrategie bei parallelen Programmen nicht dynamisch während der Laufzeit verändert werden. Insbesondere bei der Datenübertragung mit Hilfe eines zusätzlichen Knotens wären dazu Erweiterungen aus der Version 2 von MPI (*process attach*) notwendig, die aber derzeit auf den in dieser Arbeit verwendeten Rechnern nicht genutzt werden können. Die Kopplungsstrategie wird in der Konfigurationsdatei des Code Generators festgelegt. Die LVISIT-Schnittstelle hängt dabei nicht von der Kopplungsstrategie ab, d.h. ein Wechsel der Strategie erfordert keine Modifikation am Quell-Code der Simulation (siehe Abb. 5.9).

Da einige der LVISIT-Funktionen für die Anbindung an ein paralleles Simulationsprogramm die zuvor vorgestellten LVISIT-Funktionen für ein serielles Programm benutzen, erhalten die neu-

```
mlvisit_trace_split();
...
MPI_Comm_size(mlvisit_COMM_WORLD(),&numprocs);
MPI_Comm_rank(mlvisit_COMM_WORLD(),&myid);

mlvisit_trace_init();
while(SimTime) {
  mlvisit_trace_check_connection();
  mlvisit_trace_parm_recv(&parm);
  work(...);
  mlvisit_trace_velo_send(data_velo,nx,ny,nz,3,
                                    ox,oy,oz,3,sx,sy,sz,3);
}
...
mlvisit_trace_close();
...
```

Abbildung 5.9: Einbettung der LVISIT-Funktionen in ein paralleles Simulationsprogramm: Die LVISIT-Funktionen für parallele MPI-Programme beginnen jeweils mit `mlvisit`. Neu hinzugekommen sind die zusätzlichen Parameter der send-Funktion für das Versenden eines Ausschnitts des Vektorfeldes. Die beiden Funktionen `mlvisit_trace_split` und `mlvisit_COMM_WORLD` werden für die Ankopplung über einen zusätzlichen Kommunikationsknoten benötigt (siehe Text).

en Funktionen eigene Namen (beginnend mit `mlvisit`). Die Sendefunktionen erhalten darüber hinaus zusätzliche Parameter für die Spezifikation des lokal gespeicherten Teilgebiets. Für die Erhaltung einer konsistenten Kommunikation mit der Visualisierung ist es notwendig, daß auf allen Rechenknoten die LVISIT-Funktionen in der gleichen Reihenfolge aufgerufen werden. Eine wesentliche Änderung an der Schnittstelle ist die Funktion `mlvisit_trace_split`, die bei der Ankopplung über einen zusätzlichen Kommunikationsknoten benötigt wird. Mit ihrer Hilfe wird der globale MPI-Kommunikator `MPI_COMM_WORLD` in zwei neue Kommunikatoren aufgespalten. Der erste Kommunikator umfaßt alle Rechenknoten des Simulationsprogramms und der zweite nur den zusätzlichen Kommunikationsknoten. Diese Aufspaltung ist notwendig, da der zusätzliche Knoten nicht an der Simulationsrechnung teilnehmen soll. Dazu muß bei allen im Simulationsprogramm verwendeten MPI-Befehlen der globale Kommunikator durch den neuen, einen Prozessor weniger umfassenden Kommunikator ersetzt werden. Dieser Kommunikator wird durch die LVISIT-Funktion `mlvisit_COMM_WORLD` bereitgestellt. Wird die Ankopplung über den Master-Knoten oder parallel betrieben, erzeugt der Code-Generator nur eine leere Funktion für `mlvisit_trace_split` und `mlvisit_COMM_WORLD` liefert den globalen Kommunikator `MPI_COMM_WORLD` zurück. Da bei der Kommunikation über einen zusätzlichen Knoten dieser nicht an der Rechnung teilnehmen soll, wird auf diesem Knoten innerhalb von `mlvisit_trace_split` direkt in eine Warteschleife verzweigt, die für die wechselseitige Übertragung der Daten zuständig ist.

Bei der parallelen Ankopplung ist innerhalb der Sende-Funktionen kein zusätzlicher Aufwand nötig, da alle Knoten über eine direkte Verbindung zur Visualisierung verfügen. Die `mlvisit`-Funktionen rufen in diesem Fall direkt die entsprechenden `lvisit`-Funktionen auf. Bei der Ankopplung über den Master-Knoten muß dieser die Daten von den anderen Prozessoren entgegennehmen und über die VISIT-Verbindung zur Visualisierung senden. Da alle Prozessoren solange blockiert sind, bis auch der Master-Knoten wieder an der Rechnung teilnehmen kann, ist ein asynchroner Transfer der Daten mit einer Zwischenspeicherung im Master-Knoten nicht sinnvoll, zumal dieser nur einen begrenzten Speicherplatz für die Zwischenspeicherung bereitstellen kann. Der Knoten empfängt daher jeweils ein Datenpaket von einem Slave-Knoten über MPI und sendet dieses sofort weiter an die Visualisierung.

Eine asynchrone Datenübertragung ist erst durch den Einsatz eines zusätzlichen Kommunikationsknotens möglich, der dabei als eine Art Proxy-Knoten wirkt. Um eine möglichst geringe Störung der Simulationsrechnung zu erhalten, werden die Daten der einzelnen Rechenknoten zuerst nur im Proxy-Knoten in eine Container-Struktur (`pcontainer`) zwischengespeichert. Erst wenn alle Rechenknoten ihre Daten abgeliefert haben, beginnt der Proxy mit der Übertragung der gesammelten Daten zur Visualisierungsseite. Die Rechenknoten sind bei dieser Übertragung nicht beteiligt und können daher mit der Simulationsrechnung fortfahren (siehe auch Abb. 5.10 auf der nächsten Seite).

Da nur der Proxy-Knoten eine Verbindung zur Visualisierung unterhält, müssen alle LVISIT-Sende- und Lese-Aufrufe an diesen Knoten weitergereicht werden. Da dieser asynchron zu der Simulationsrechnung läuft, muß der über die nächste Aktion informiert werden. Dazu schickt jeder Knoten über MPI die entsprechende Id der nächsten LVISIT-Übertragung an den Proxy-Knoten. Bei einem Sende-Befehl werden danach die Daten zum Proxy-Knoten gesendet. Bei einem Lese-Befehl können die von der Visualisierung erhaltenen Daten mit Hilfe des MPI-Broadcast-Befehls verteilt werden. Der Proxy-Knoten führt im wesentlichen eine Schleife aus, in der zu Anfang mit dem MPI-Probe-Befehl auf eine MPI-Nachricht eines Rechenknotens gewartet wird. Ist eine solche Nachricht eingetroffen, wird die entsprechende VISIT-Übertragung für alle Rechenknoten komplett abgearbeitet. Da auf allen Rechenknoten die LVISIT-Funktionen in der gleichen Reihenfolge aufgerufen werden, können durch dieses Verfahren auf der Proxy-Seite keine Dead-Locks auftreten.

Auch auf der Visualisierungsseite ist zur Vermeidung von Wartezeiten eine Speicherung der Daten mit Hilfe der Container-Struktur `pcontainer` implementiert worden. Dort werden alle Teilblöcke der einzelnen Rechenknoten zwischengespeichert, bis alle Blöcke übertragen worden sind. Erst dann werden die Daten entkomprimiert und den Verarbeitungsmodule von AVS/Express übergeben. Ein Blockieren der Rechenknoten insbesondere bei der parallelen Ankopplung oder bei der Ankopplung über den Master-Knoten kann so verhindert werden.

5.4 Entwurf des Code-Generators visitcg

Bei der Entwicklung des Code-Generators `visitcg` sind mehrere Ziele berücksichtigt worden. So wird sowohl auf der Simulationsseite als auch auf der Visualisierungsseite jeweils eine an das Problem angepaßte und möglichst einfache Schnittstelle angeboten. Zusätzlich kann die konsistente Vergabe von Ids und die korrekte Verarbeitung der Daten auf beiden Seiten der Übertragung nur dann garantiert werden, wenn für die Generierung der Funktionen bzw. Module und Makros nur eine Definition benutzt wird. Besonders auf der Visualisierungsseite ist die Verknüpfung der Makros und Module für die Verarbeitung der von VISIT entgegengenommenen Daten aufwendig und fehleranfällig.

Der Code-Generator liest aus einer Konfigurationsdatei die Beschreibung der zu übertragenden Daten und generiert, je nach Optionen beim Aufruf des Generators, Schnittstellen für AVS/Express auf der Server-Seite und für die Programmiersprachen C und FORTRAN auf der Client-Seite. Für AVS/Express wird eine V-Datei erzeugt, die in ein bestehendes AVS/Express Projekt-Verzeichnis abgelegt wird. Zusätzlich werden die Projekt-spezifischen V-Dateien in diesem Verzeichnis so verändert, daß in den Modul-Katalogen von AVS/Express alle wesentlichen Module und Groups aufgeführt werden. Weiterhin wird dort eine AVS/Express-Applikation abgelegt, die ein komplettes Netzwerk für die Kommunikation mit der Simulation und die Anzeige der empfangenen Daten enthält. Der Benutzer kann diese Applikation als Basis für eigene Weiterentwicklungen der Visualisierung nutzen. Für die Einbindung der Schnittstelle auf der Simulationsseite werden verschiedene C-Header- und C-Source-Dateien erzeugt und in einem separaten Verzeichnis abgelegt. Zusätzlich wird dort auch ein Makefile abgelegt, das die Schnittstellen-Funktionen kompiliert und in einer Bibliothek ablegt (z.B. `liblvisit_trace`). Bei der Kompilation der Anwendung müssen ggf. die C-Header-Dateien eingebunden und die Aufrufe der Schnittstellen-Funktionen im

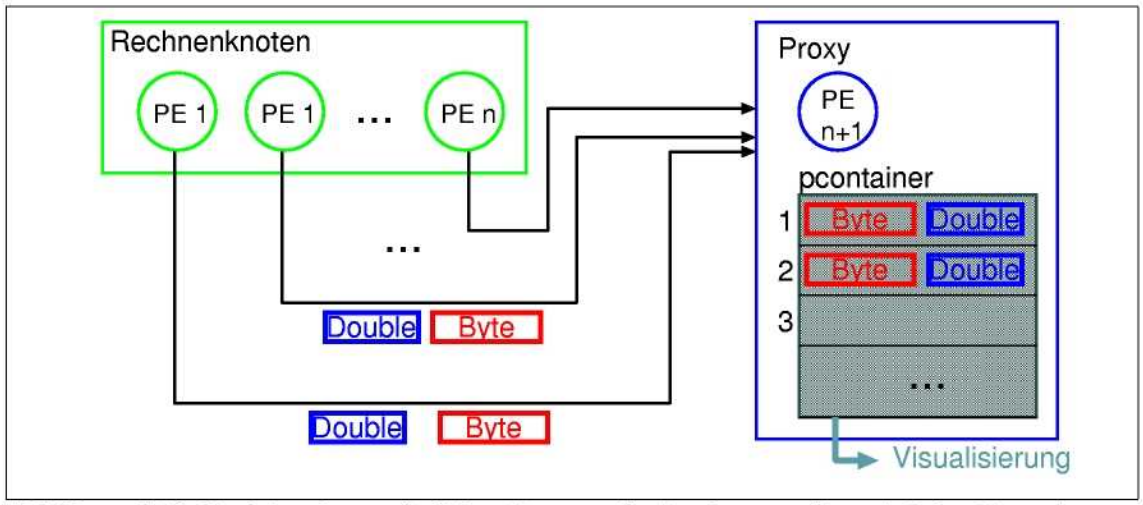


Abbildung 5.10: Funktion des zusätzlichen Kommunikationsknotens (Proxy): Beim Versenden von komprimierten Daten werden die Teilblöcke der einzelnen Rechenknoten auf dem Proxy-Knoten zuerst zwischengespeichert, um dann gemeinsam und asynchron von der Simulationsrechnung zur Visualisierung weitergesendet zu werden.

Programm-Code eingefügt werden. Beim Compiler-Aufruf muß die LVISIT-Bibliothek und zusätzlich die durch den Code-Generator erzeugte Schnittstellen-Bibliothek angegeben werden (z.B. `-llvisit -llvisit_trace`).

Der Code-Generator ist in der Script-Sprache Perl implementiert. Sie ist u.a. wegen der guten Unterstützung der String-Verarbeitung durch reguläre Ausdrücke für die innerhalb des Generators anfallende Verarbeitung von Code-Fragmenten gut für diese Aufgabe geeignet. Der Generator ist in mehrere Perl-Module aufgeteilt, die für die Generierung der jeweiligen Schnittstellen zuständig sind. Für die Erstellung des AVS/Express-Netzwerks wurde ein Perl-Modul *Vtree* entwickelt, das für den Aufbau der Netzwerke deren baumartige Definition innerhalb der V-Sprache ausnutzt. Für die Erstellung der verschiedenen Header- und Source-Dateien wurde ein Template-Mechanismus entwickelt, der es erlaubt, die Schnittstellen-Funktionen allgemein zu beschreiben. Die folgenden Abschnitte beschreiben neben dem Aufbau der Konfigurationsdatei diese beiden Perl-Module.

5.4.1 Beschreibung der Datenströme

Der Code-Generator benötigt für die Generierung der Schnittstellen eine Beschreibung der Datenströme. Ein Datenstrom beschreibt dabei entweder ein Feld, einen skalaren Wert oder eine Struktur von skalaren Werten bzw. fest dimensionierten Feldern. Jeder Datenstrom muß bei der Definition mit einer Id versehen werden, die auf beiden Seiten der Kopplung eine eindeutige Zuordnung der Daten zu einem Datenstrom ermöglicht. In der vom Code-Generator benötigten Konfigurationsdatei werden diese Datenströme in einzelnen Abschnitten beschrieben. Die Datei muß derzeit als ASCII-Datei vorliegen. Eine Erweiterung des Code-Generators um eine zusätzliche Komponente mit interaktiver Oberfläche könnte durch die Verwendung dieser Konfigurationsdatei ohne Änderungen am Code-Generator entwickelt werden.

Im ersten Abschnitt der Konfigurationsdatei werden allgemeine Angaben zur Schnittstelle definiert. Dazu gehört ein Name, der sich z.B. auch in den Namen der generierten Wrapper-Funktionen

```
general:
    name  = trace
    cdir        = ../trace/lvisit
    expressdir  = ../waveview
    mpi   = yes
    mpicollective   = no
    mpiaddnode   = yes
dataset velo:
        active        = yes
        id            = 39
        datatype      = double
        direction     = sim -> vis
        dimension     = 3    # dimension of the field
        veclen        = 3    # number vector elements per node
        compression   = waveletprog
        distribution  = domain
        vistype       = bounds,axis3d,glyph

dataset stat:
        active       = yes
        id           = 3
        datatype      = compound
        compound_datatypes = (
                                integer step,    string  info,
                                double x[100],   int    tempnumx[100]
                             )
        direction    = sim -> vis
 ...
```

Abbildung 5.11: Konfigurationsdatei des Code-Generators (`trace.api`): Der obere Teil der Konfigurationsdatei enthält generelle Informationen über die Ankopplung. Dazu gehören unter anderem die gewünschte Kopplungsstrategie für die Anbindung an das parallele Simulationsprogramm (Trace) und die verschiedenen Verzeichnisse, in denen die Source-Dateien der LVISIT-Funktionen abgelegt werden sollen. In den nachfolgenden Abschnitten werden die einzelnen Datenströme definiert, zu denen das Geschwindigkeitsfeld einer Grundwasserströmung gehört. Es handelt sich dabei um ein dreidimensionales Feld, dessen Feldelemente aus Vektoren der Länge 3 bestehen (x,y,z-Komponente der Geschwindigkeit). Der zweite Eintrag zeigt, wie eine Struktur von skalaren Daten und Feldern als ein Datenstrom definiert werden kann.

wiederfindet, die Verzeichnisse, in denen die zu erzeugenden Dateien abgelegt werden sollen, sowie Angaben über die Kopplungsstrategie bei der Verbindung zu einem parallelen Simulationsprogramm, die für alle Datenströme gleich gehalten wird (siehe Abb. 5.11 auf der vorherigen Seite).

Die weiteren Abschnitte beschreiben die einzelnen Datenströme. Dazu gehören als wesentliche und zwingende Einträge der Datentyp, die Richtung der Übertragung und die Dimension der Daten. Die Größe der Daten wird nicht in dieser Datei festgelegt, sondern bei den Aufrufen der Schnittstellen-Funktionen als Parameter während der Laufzeit des Simulationsprogramms übergeben. Der Id-Eintrag legt die bei den VISIT-Funktionen benutzte numerische Identifikation der Daten fest. Sie wird nur innerhalb der LVISIT-Bibliothek und den Schnittstellen-Funktionen benutzt und kann auch weggelassen werden. In diesem Fall wird die Id von dem Code-Generator festgelegt.

5.4.2 V-Code-Generierung mit dem Perl-Modul Vtree

In AVS/Express werden die Verarbeitungs-Netzwerke aus Modulen und Makros zusammengestellt. Dabei bestehen die Makros wieder aus einer Menge von Modulen oder Makros, so daß sich daraus eine baumartige Anordnung der Module ergibt. Die Verbindung der Ein- und Ausgänge der Module untereinander beschreibt den Datenfluß innerhalb des Netzwerks. Diese Verbindungen sind an die Baumstruktur des Netzwerks gebunden, so daß mit Hilfe der Symbole "<-." und "." zu höheren Ebenen des Baums bzw. zu den Elementen eines untergeordneten Makros verwiesen werden kann. Abbildung 5.12 zeigt ein Beispiel für die Kodierung eines Netzwerks in der Sprache V. An diesem Beispiel ist auch zu erkennen, daß die manuelle Erstellung eines solchen Codes schwierig ist, da für die Definition der Verbindungen jeweils viele Ebenen des Baumes durchlaufen werden.

```
APPS.MultiWindowApp Bubbleviz {
   UI {
      Modules {
         IUI {
            optionList {
               cmdList => {
                  <-.<-.<-.<-.Read_Field.read_field_ui.panel.option,
                  <-.<-.<-.<-.bounds.UIpanel.option,
                  <-.<-.<-.<-.isosurface.UIpanel.option
   }; }; }; }; };
   GDM.Uviewer3D Uviewer3D {
      Scene {
         Top {
            child_objs => {
               <-.<-.<-.bounds.out_obj,<-.<-.<-.isosurface.out_obj};
         };
   };
   MODS.Read_Field Read_Field {
      read_field_ui {
         filename = "./data.fld";
      };
      DVread_field {
         Mesh_Unif+Node_Data Output_Field;
     };
   };
   MODS.bounds bounds {
      in_field => <-.Read_Field.field;
   };
   MODS.isosurface isosurface {
      in_field => <-.Read_Field.field;
      IsoParam {
         iso_level => 32.24;
   }; }; };
```

Abbildung 5.12: Beispiel für eine Netzwerkdefinition in Programmiersprache V: In V wird die Verknüpfung der Module und deren Gruppierung in Makros definiert. Durch die Zusammenfassung von Modulen oder Makros entsteht eine Baumstruktur, die in der V-Datei durch die Klammerung der Definitionen repräsentiert wird. Die Verknüpfung der Ein- und Ausgänge der einzelnen Module ist davon unabhängig und wird in V mit Hilfe des Dereferenzierungssymbols "<-." realisiert. Das Beispiel zeigt die Definition des Netzwerks, mit dem die Anzeige der in Abb. 2.3 auf Seite 8 dargestellten Iso-Oberflächen möglich ist.

Um Fehler bei der Definition der Module und deren Verbindungen zu vermeiden, ist das Perl-Modul *Vtree.pm* entwickelt worden, das ein Vtree-Objekt definiert. Dieses Objekt besitzt Methoden, die das Generieren von neuen, dem Objekt im Baum untergeordneten Objekten (Makros) und das Verbinden dieser Objekte unterstützt. Abbildung 5.13 zeigt ein Beispiel dafür, wie der Code-Generator das Perl-Modul benutzt.

```
$top=Vtree->new("APPS.MultiWindowApp","Bubbleviz");
...
$read=$top->new("MODS.Read_Field","Read_Field");
$readfield=$read->new("","field");
...
$iso=$top->new("MODS.isosurface","isosurface");
$infield=$iso->new("","in_field");
$infield->connect_to($readfield);
...
$iso_level-value(32.24);
...
$code=$top->dumpit();

print $code;
```

Abbildung 5.13: Beispiel für eine Benutzung des Vtree-Moduls: Mit der `new`-Methode wird ein neues Vtree-Objekt generiert, das dem aktuellen Objekt untergeordnet ist. Auf diese Weise kann die baumartige Struktur eines AVS/Express-Netzwerk aufgebaut werden. Mit der Methode `connect_to` können zwei Objekte miteinander verbunden werden, ohne daß der Weg durch den Baum explizit angegeben werden muß. Die Methode `dumpit` erzeugt nach der Definition des Netzwerks den zugehörigen V-Code.

Durch das Vtree-Modul ist es nicht mehr nötig, innerhalb des Code-Generators direkt V-Code zu erzeugen. Dies hilft nicht nur bei der Vermeidung von Fehlern bei der Definition des Netzwerks, sondern auch bei der Anpassung des Code-Generators an neue Versionen von AVS/Express.

5.4.3 C-Code-Generierung mit einem Template-Mechanismus

Auch bei der Generierung der Schnittstellen auf der Client-Seite ist es aus Gründen der Fehleranfälligkeit und aufwendigen Maskierung von Sonderzeichen nicht sinnvoll, den Quell-Code direkt aus dem Code-Generator heraus zu generieren. Daher werden hier dem Code-Generator Templates für die einzelnen Typen von Schnittstellen-Funktionen zur Verfügung gestellt. In den dort definierten Vorgaben für die Schnittstellen-Funktionen müssen vom Code-Generator nur noch die einzelnen Platzhalter durch die in der Definitionsdatei angegeben Datenströmen ersetzt werden.

Abbildung 5.14 auf der nächsten Seite zeigt ein Beispiel für ein Template einer Schnittstellen-Funktion. Der Quell-Code, aus dem die Funktionen für die C-Schnittstelle generiert werden, ist mit XML-ähnlichen Markierungen (TAGS) versehen, die den Templates jeweils Namen zuordnen. Mit weiteren TAGs werden z.B. Einschränkungen angegeben, welche die Anwendung des Templates nur unter bestimmten Bedingungen erlauben. Weiterhin wird unterschieden, ob ein Template nur einmal für die gesamte Schnittstellen angewendet werden soll oder ob die mit dem Template beschriebene Funktion für jeden in der Konfigurationsdatei beschriebenen Datenstrom angewendet werden soll. Ein Beispiel für den ersten Fall ist die weiter oben beschriebene Funktion `lvisit_trace_check_connection`. Ein Beispiel für den zweiten Fall ist die in der Abbildung 5.14 auf der nächsten Seite gezeigte Funktion zum Empfangen von skalaren Daten.

Innerhalb des Templates können Variablen verwendet werden, die bei der Anwendung des Templates durch den Code-Generator ersetzt werden. Dadurch können z.B. an die Dimension des Datenstroms angepaßte Parameterlisten benutzt werden oder Templates definiert werden, die unabhängig von dem verwendeten Datentyp des Datenstroms sind. Dazu werden Variablen der Form `$name` benutzt (siehe auch Abb. 5.16 auf Seite 88).

```
<TEMPLATE H_RECV_SCALAR_MPIADDNODE>
<ITERATOR $group>
<CONSTRAINT scalar recv mpi addnode>
int mlvisit_$name_$group_recv($parm);
</TEMPLATE>

<TEMPLATE C_RECV_SCALAR_MPIADDNODE>
<ITERATOR $group>
<CONSTRAINT scalar recv mpi addnode>
int mlvisit_$name_$group_recv ($parm) {
  int rc=0, intdata=0;

  /* send Id to proxy */
  MPI_Send(&intdata,1,MPI_INT,mpi_proxy,$id,MPI_COMM_WORLD);

  /* receive data from proxy */
  MPI_Bcast(data_${group},sizeof($cdatatype), MPI_BYTE, mpi_proxy,
            MPI_COMM_WORLD);
  &apply_template(PROXY_PRINT_VISCTL);
  return(1);
}
</TEMPLATE>
```

Abbildung 5.14: Beispiel für ein Template einer Schnittstellen-Funktion: Dieses Template beschreibt die Schnittstellen-Funktionen, die auf der Simulationsseite für die Übertragung von skalaren Daten von der Visualisierung zur Simulation zuständig ist. Das Template wird nur dann angewendet, wenn die Schnittstelle für eine Ankopplung an ein paralleles Programm unter Hinzunahme eines zusätzlichen Kommunikationsknoten genutzt wird. Dazu sendet die Funktion zuerst mit Hilfe einer MPI-Funktion eine Dummy-Nachricht an den Proxy-Knoten, um diesen über die nachfolgende Übertragungsaktion zu informieren. Dieser erkennt am MPI-TAG die Id des Datenstroms und initiiert dann die VISIT-Übertragung. Nach deren Beendigung werden die Daten mit Hilfe des MPI-Broadcast-Befehls an alle Rechenknoten verteilt.

Die Anwendung eines Templates kann auch rekursiv erfolgen. So können innerhalb eines Templates wieder andere Templates benutzt werden. Dazu dient der Befehl `&apply_template`, der vom Code-Generator erkannt und durch das entsprechende Template ersetzt wird.

Durch einen `INCLUDE`-TAG können weitere Dateien mit Template-Definitionen eingelesen werden. Daher wird vom Code-Generator zu Beginn nur eine Template-Datei (`templates.c` bei C-Quellcode) eingelesen. Über das TAG `FILENAME` wird dem Code-Generator mitgeteilt, daß der durch dieses Templates erzeugte Quellcode in der angegebenen Datei abgelegt werden soll. Abbildung 5.15 zeigt, wie dadurch eine C-Header-Datei erzeugt wird.

```
 <TEMPLATE C_HEADER_FILE>
 <FILENAME lvisit_$name.h>
 #ifndef LVISIT_$NAME_H
 #define LVISIT_$NAME_H
 #include <lvisit.h>

 &apply_template(H_INITUTS)
 &apply_template(H_INIT)
 &apply_template(H_CHECK)
 &apply_template(H_SEND_SCALAR)
 &apply_template(H_SEND_FIELD)
 &apply_template(H_SEND_WAVELET)
 &apply_template(H_RECV_SCALAR)
 &apply_template(H_CLOSE)
 #endif
</TEMPLATE>
```

Abbildung 5.15: Beispiel für ein Template einer C-Header-Datei: Durch das TAG `FILENAME` wird dem Code-Generator mitgeteilt, daß der erzeugte Quellcode in der angegebenen Datei abgelegt werden soll. Von diesem Template aus werden alle Templates angewendet, welche die Funktions-Deklarationen der Schnittstellen beschreiben.

Durch den hier vorgestellten Template-Mechanismus ist die Kodierung des Quell-Codes von dem eigentlichen Perl-Script des Code-Generators abgekoppelt. Dies erleichtert nicht nur die Definition dieser Schnittstelle, sondern auch die Erweiterung der Code-Generators für andere Programmiersprachen wird dadurch wesentlich einfacher.

Die Anbindung an Simulationsprogramme, die in FORTRAN 90 implementiert sind, ist durch Wrapper-Funktionen realisiert worden, welche die mit dem Code-Generators erzeugten C-Schnittstellen-Funktionen aufrufen. Die Wrapper-Funktionen sind dazu in einer eigenen Template-Datei definiert und durch das `INCLUDE`-TAG zu den bestehenden Templates hinzugefügt worden. Diese Schnittstelle wird z.B. für die Anbindung an das in dieser Arbeit verwendete Simulationsprogramm Trace benutzt.

<TEMPLATE *name*>	Anfang einer Template-Definition
</TEMPLATE>	Ende eines Template-Definition
<ITERATOR group>	Wiederholung des Templates für alle Datenströme (groups)
<CONSTRAINT cstr1 ...>	Einschränkung auf verschiedene Bedingungen (Bedingungen werden und-verknüpft)
scalar, field	nur wenn Datenstrom Skalar bzw. Feld ist
send, recv	Übertragungsrichtung
normalfield	keine Komprimierung
waveletprog	progressive Wavelet-Transformation
(no)mpi	Anbindung an paralleles Simulationsprogramm (MPI)
(no)collective	(keine) parallele Ankopplung
(no)addnode	(keine) Ankopplung über zusätzlichen Knoten
<FILENAME *fname*>	Speicherung des Quell-Codes in Datei *fname*
<INCLUDE *iname*>	Lese weitere Template-Definitionen aus Datei *iname*
&apply_templates(*tname*)	Anwendung des Templates *tname* an dieser Stelle
&eval("*body*","*var* (*range*)")	Schleifenausführung
$name	Name der Schnittstelle
$group	Name des Datenstroms
$id	Numerische Identifikation (Id) des Datenstroms
$parm	Parameterliste der Schnittstellen-Funktion
$dim	Dimension der Daten

Abbildung 5.16: Liste der wichtigsten Template-Ersetzungen

Kapitel 6

Verifikation des Modells und Messung der Modellparameter

Mit der im vorherigen Kapitel vorgestellten neuen LVISIT-Bibliothek und dem Code-Generator wurden wichtige Werkzeuge bereitgestellt, um die verschiedenen Ankopplungsmöglichkeiten einer Visualisierung an bestehende Simulationsprogramme sowie die unterschiedlichen Datenreduzierungs- und Komprimierungsmethoden einfach anzuwenden. Dadurch wird der Einsatz des Computational Steering auch bei Simulationsprogrammen mit sehr großen Datenmengen möglich. Mit diesen Werkzeugen ist es aber auch möglich, durch Zeitmessungen von realen Simulationsprogrammen in einer für diese typischen Umgebung (siehe unten) zu prüfen, ob die in Kapitel 4 entwickelten Modelle das Verhalten der Programme richtig beschreiben und aus den Messungen die Modellparameter (Bandbreiten, Latenzen und Verarbeitungsgeschwindigkeiten) quantitativ zu bestimmen.

Diese Messungen werden exemplarisch für zwei verschiedene Szenarien durchgeführt. Als Parallelrechner konnten dazu das vom Zentralinstitut für Angewandte Mathematik (ZAM) betriebene Linux-SMP-Cluster ZAMpano und der Parallelrechner CRAY T3E-1200 genutzt werden [56]. Auf der Visualisierungsseite kam vorwiegend ein Linux-Notebook zum Einsatz, welches das typische Einsatzgebiet des Computational Steering widerspiegelt, in dem die Kontrolle des Simulationsprogramms auch vom normal ausgestatteten Arbeitsplatz aus und nicht nur von speziellen Graphik-Workstations aus möglich sein soll.

Der Vergleich der Modelle mit den Ergebnissen der Testläufe gliedert sich in mehrere Stufen. Zuerst müssen die grundlegenden Parameter der Modelle festgelegt werden. Dazu gehören die Übertragungsbandbreiten und Latenzen der in den Szenarien benutzten Netzverbindungen (Abschnitt 6.1) und die Verarbeitungsgeschwindigkeiten für die einzelnen Schritte der Datenreduzierung auf den eingesetzten Rechnern (Abschnitt 6.2). Aus diesen gemessenen Modellparametern kann an Hand der im Modell entwickelten Gleichungen eine Abschätzung für die durch die Kopplung verursachte zusätzliche Kommunikationszeit berechnet und mit der Messung verglichen werden. Anschließend wird die Auswirkung der Kopplung auf die Laufzeit von realen Simulationsprogrammen betrachtet (Abschnitt 6.3). Zuletzt wird noch die Auswirkung der verlustbehafteten Komprimierung auf die Daten betrachtet (Abschnitt 6.4).

6.1 Messungen der Netzparameter

Für die Messung von Bandbreite und Latenz der Verbindung zwischen Visualisierungs- und Parallelrechner wird eine Nachricht zuerst in eine Richtung gesendet und nach der vollständigen Übertragung wieder zurückgesendet. Bei diesem als Pingpong-Verfahren bezeichneten Test kann die Länge der Nachricht variiert und so die Laufzeit der Nachricht als Funktion der Nachrichtenlänge ermittelt werden. Als Laufzeit wird dabei die Hälfte der Zeit betrachtet, die für eine vollständige Hin- und Rückübertragung benötigt wird. Durch diese Mittelung wird ggf. ein unsymmetrisches Verhalten der Übertragungsstrecke bezüglich der Übertragungszeit ausgeglichen. Die Latenz der Verbindung ist diejenige Zeit, die für Initiierung der Nachricht benötigt wird. Sie kann durch die Laufzeit einer sehr kleinen Nachricht (1-Byte) bestimmt werden, bei der die reine Übertragung der Daten vernachlässigbar ist. Der Austausch der Nachrichten mit Hilfe des Pingpong-Verfahrens wird dabei für jede Nachrichtenlänge mehrfach wiederholt und deren Mittelwert bestimmt.

Die Messung der beiden Verbindungsstrecken zu den Parallelrechnern CRAY T3E und ZAMpano wurde jeweils zweimal durchgeführt. Im ersten Fall wurde die vorhandene Anbindung des Linux-Notebooks über FastEthernet genutzt. Im zweiten Fall wurde ein Repeater zwischengeschaltet, der die Übertragungsgeschwindigkeit auf 10 MBit/s reduziert (siehe Abb. 6.1). Die Messung mit dieser geringeren Geschwindigkeit soll den Fall repräsentieren, daß der Visualisierungsrechner nicht vor Ort steht und die Übertragung über viele Teilstrecken führt, die gleichzeitig auch von anderen Teilnehmern genutzt werden (z.B. Visualisierung in einer anderen Institution). Auch die Nutzung von Computational Steering über Wireless LAN stellt eine solche Einschränkung dar, da dort die Bandbreite begrenzt und mit allen Teilnehmern des Funknetzes mit gleicher Basisstation geteilt werden muß. Die meisten Messungen fanden auf nicht dedizierten Rechnern und Netzen statt. Da die in der Messung benutzen Strecken in der Regel nur gering belastet sind, werden die Ergebnisse nur geringfügig durch anderen Netzverkehr beeinflußt. Zum Nachweis wurde eine Messung auf der CRAY T3E innerhalb einer dedizierten Testzeit durchgeführt. In Abbildung 6.2 auf der nächsten Seite ist die Übertragungsgeschwindigkeit bei diesem Testlauf zusätzlich dargestellt.

Der Datenaustausch wurde zuerst mit den Funktionen der VISIT-Bibliothek implementiert und gemessen, da diese auch für die Implementierung der Kopplung verwendet werden. Zum Vergleich und zur Bestimmung des zusätzlichen Overheads dieser Bibliothek wurden Pingpong-Messungen auch direkt mit Hilfe der TCP/IP-Sockets durchgeführt. Abbildung 6.2 auf der nächsten Seite zeigt die Ergebnisse. Auffällig ist, daß insbesondere bei der CRAY T3E der Overhead der Biblio-

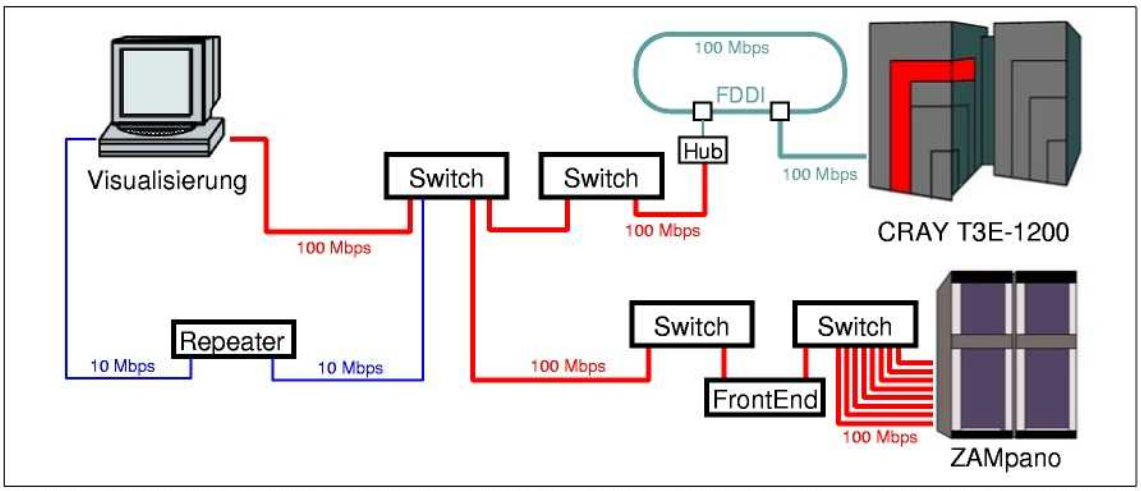


Abbildung 6.1: Netzverbindung der für die Messung benutzten Rechner: Der Visualisierungsrechner kann alternativ über Ethernet (10 MBit/s) oder FastEthernet (100 MBit/s) angebunden werden. Zum Linux-SMP-Cluster ZAMpano führt der Weg über mehrere Teilstrecken mit jeweils 100 MBit/s FastEthernet. Der Weg zum Parallelrechner CRAY T3E führt über einen FDDI-Backbone (100 MBit/s).

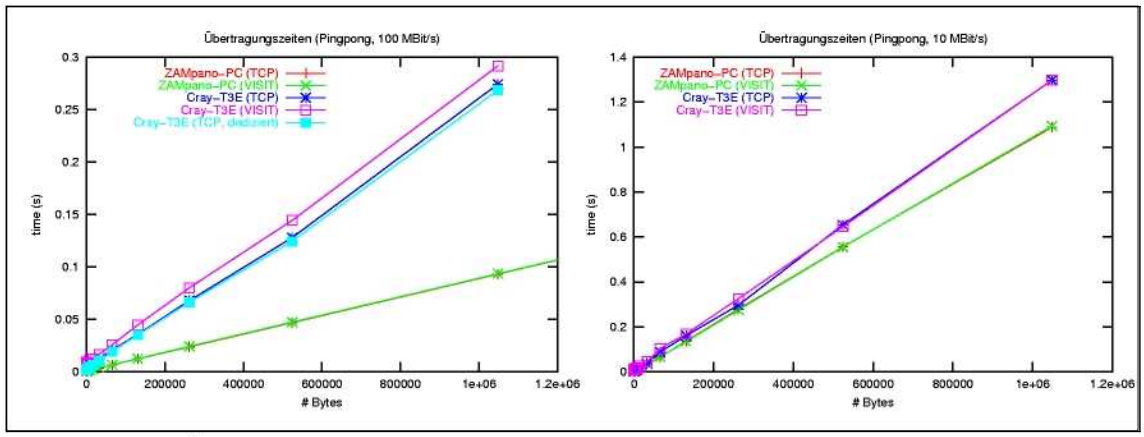


Abbildung 6.2: Übertragungszeit beim Austausch von Nachrichten (Pingpong): Das linke Diagramm zeigt die Übertragungszeiten für Nachrichtenlängen von 1 Byte bis zu 1 MByte bei einer Anbindung über FastEthernet. Im rechten Diagramm sind die Übertragungszeiten für den Umweg über den 10 MBit/s-Repeater aufgetragen. Sowohl bei der Übertragung über TCP/IP-Sockets als auch über VISIT gibt es eine lineare Abhängigkeit der Übertragungszeit von der Nachrichtenlänge. Im linken Diagramm ist zusätzlich die Übertragungszeit bei einer dedizierten Messung auf der CRAY T3E aufgetragen, bei der es nur geringfügige Unterschiede zur Messung im normalen Betrieb des Rechners gibt.

thek mehrere Millisekunden beträgt. Die Ursache dafür liegt in dem Kommunikationsprotokoll der VISIT-Bibliothek (siehe Abbildung 6.3 auf der nächsten Seite (links)). Im Gegensatz zu der Kommunikation über TCP/IP wird bei VISIT eine Nachricht in eine numerischen Kennung (Id), eine Datenbeschreibung (Request) und die Daten selbst zerlegt. Die Id wird über den Control-Socket übertragen, während die anderen Daten mittels eines Datensockets ausgetauscht werden. Nach Abschluß des Lesens einer Nachricht schickt der Empfänger über den Control-Socket eine Bestätigung an den Sender (ACK2). Da die Id und die anderen Nachrichten über verschiedene Sockets versendet werden, ergeben sich so für die Übertragung drei verschiedene Pakete (ACK2, ID und Request+Daten). Das erklärt die dreifach höhere Latenz der Übertragung im Vergleich zu TCP/IP. Die TCP/IP-Latenz ist auf der CRAY T3E aus verschiedenen Gründen höher als auf ZAMpano. Zum Beispiel wird innerhalb des Parallelrechners die externe Kommunikation über einen separaten Kommunikationsknoten geleitet, der auf dem Übertragungsweg zu einem zusätzlichen Zwischenstop führt. Da die Knoten des Linux-SMP-Clusters ZAMpano aber direkt mit einem Switch verbunden sind, ist hier eine niedrigere Latenz zu erwarten.

Die Zeitmessungen zeigen, daß es eine lineare Abhängigkeit der Übertragungszeit von der Nachrichtenlänge gibt, und bestätigen somit die im Modell spezifizierte Formel für die Übertragungszeit:

$$t_n = \frac{s}{b_n} + t_l \tag{6.1}$$

Die in der nachfolgenden Tabelle aufgeführten Werte für die Modellparameter der Kommunikation wurden durch lineare Regression aus den Meßwerten bestimmt.

		ZAMpano		CRAY T3E	
		10 MBit/s	100 MBit/s	10 MBit/s	100 MBit/s
Latenz	TCP/IP	0.21 ms	0.15 ms	2.03 ms	2.27 ms
	VISIT	1.06 ms	0.77 ms	9.42 ms	6.99 ms
maximale	TCP/IP	7.35 MBit/s	86.27 MBit/s	6.17 MBit/s	29.49 MBit/s
Bandbreite	VISIT	7.21 MBit/s	86.09 MBit/s	6.23 MBit/s	27.72 MBit/s

6.2 Zeitmessungen zu den einzelnen Verarbeitungsschritten

Bei der Entwicklung der Modelle für die Datenreduzierung wurden Gleichungen für die Verarbeitungszeit der einzelnen Teilschritte der Datenreduzierung aufgestellt. In allen Teilschritten wird im Modell eine lineare Abhängigkeit der Verarbeitungszeit von der Datengröße angenommen, so daß die Verarbeitungszeit aus einer Verarbeitungsbandbreite (Geschwindigkeit) und einer Latenz berechnet werden kann. Durch die Messung der Verarbeitungszeiten der einzelnen Teilschritte wird diese Annahme überprüft und die Modellparameter bestimmt.

Für die Messung der Verarbeitungsgeschwindigkeiten wurde statt des Simulationsprogramms ein vereinfachtes Simulationsprogramm (Client) benutzt, das einen Beispieldatensatz einliest und mit den LVISIT-Routinen an einen Server sendet. Auch für die Server-Seite wurde statt der Visualisierung ein einfacherer Server benutzt, der die Daten entgegennimmt und mit den LVISIT-Routinen rekonstruiert. Durch diese vereinfachten Programme auf Client- und Server-Seite ist es möglich, die Messung mit verschiedenen Parametern für die Übertragung zu wiederholen und Zeiten zu ermitteln. Bei der Messung wurde der Client jeweils auf einem Knoten der beiden Parallelrechner CRAY T3E und ZAMpano ausgeführt. Der Server, der die Funktionalität der Visualisierung repräsentiert, wurde auf einem Linux-Notebook ausgeführt, das mit 100 MBit/s oder über den Repeater über 10 MBit/s angeschlossen war.

Als Beispieldatensatz wurde ein von dem Simulationsprogramm Trace erzeugtes Geschwindigkeitsfeld der Größe 256x128x64 benutzt. In diesem dreidimensionalen Datensatz sind pro Gitterknoten die drei Komponenten des Geschwindigkeitsvektors der Grundwasserströmung in X-, Y- und Z-Richtung abgespeichert. Im Simulationsgebiet befinden sich vier Brunnen, die sich im Geschwindigkeitsfeld als Senken repräsentieren. Mit zusätzlich eingebundenen Wasserquellen wird ein inhomogenes Geschwindigkeitsfeld der Grundwasserströmung erzeugt. Zur Messung der Verarbeitungsgeschwindigkeit der einzelnen Komprimierungsschritte werden die benötigte Zeit für verschieden große Datensätze gemessen. Dazu wird der Beispieldatensatz, der in einer festen Größe vorliegt, vergrößert, indem das Simulationsgebiet entsprechend oft in jede Richtung wiederholt wird. Dabei wird das Gebiet jeweils gespiegelt und die Elemente mit einem Faktor multipliziert. Dieses ist für die Messung des letzten Komprimierungsschrittes notwendig, da die dort verwendete LZO-Komprimierung sehr empfindlich auf wiederkehrende Muster reagiert. Eine Skalierung des Datensatzes mit Hilfe einer Interpolation ergäbe zwar eine höhere Auflösung. Durch die linea-

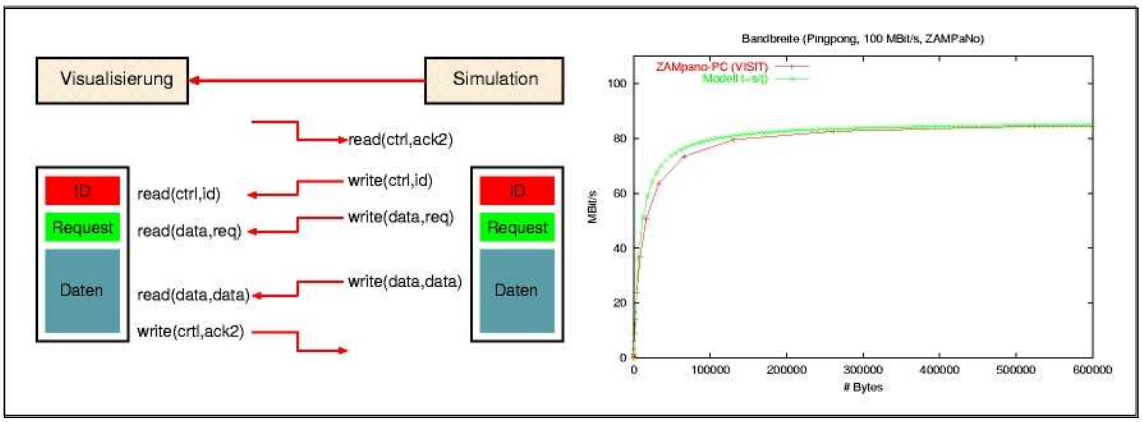


Abbildung 6.3: Datenaustausch bei VISIT und Bandbreite der Übertragung: Die linke Abbildung zeigt, wie die einzelnen Teile einer Nachricht bei VISIT versendet werden. Wichtig ist hier, daß vor dem Absenden einer Nachricht überprüft wird, ob die zuletzt versendete Nachricht schon verarbeitet wurde (ACK2). Das rechte Diagramm zeigt, daß die zwischen dem Linux-Notebook und dem ZAMpano bei einer FastEthernet-Verbindung gemessene Bandbreite und die aus dem Modell berechnete Bandbreite sehr gut übereinstimmen, was das Modell bestätigt.

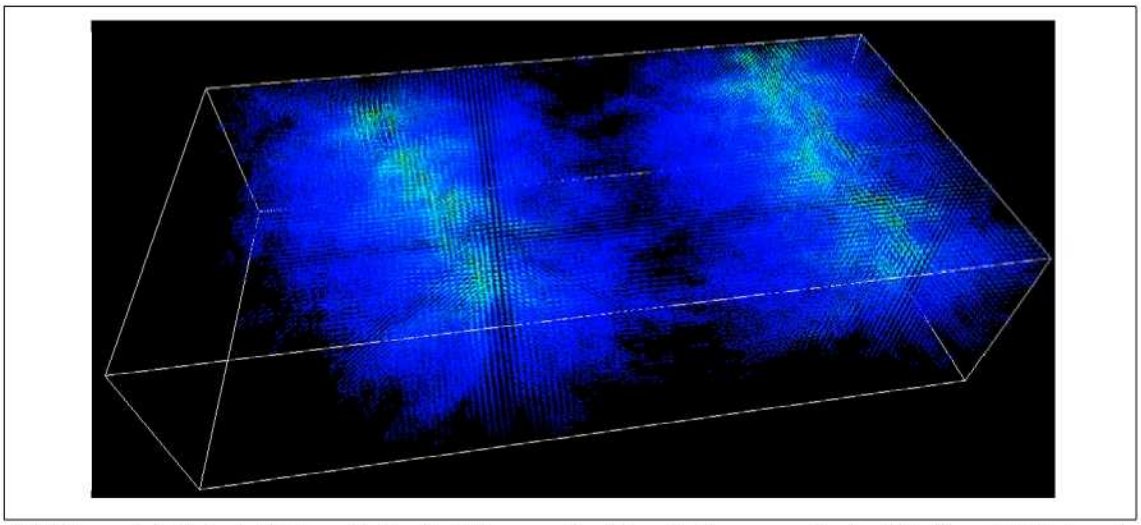


Abbildung 6.4: Beispieldatensatz für die Messung der Verarbeitungsgeschwindigkeiten: Als Datensatz wurde in der Messung dieses von Trace berechnete Geschwindigkeitsfeld benutzt, das durch die im Simulationsgebiet vorhandenen Brunnen (links) und Wasserquellen (rechts) starke Inhomogenitäten besitzt.

re Interpolation der in der höheren Auflösung vorhandenen zusätzlichen Punkte erhöht sich der Informationsgehalt des Datensatzes jedoch nicht. Eine solche Skalierung führt z.B. bei der LZO-Komprimierung nur zu größeren Mustern, die mit einer entsprechend höheren Komprimierungsrate gespeichert werden können, was letztendlich kein realistisches Beispiel für die Anwendung wiederspiegelt. Bei der Messung wurde für jeden Datensatz eine Iteration des Pyramidenalgorithmus ausgeführt, danach 80 % der Koeffizienten auf Null gesetzt und eine 8-Bit Quantisierung angewendet.

Bei der Bewertung der in diesem Kapitel gemessenen Verarbeitungsgeschwindigkeiten müssen die Leistungsdaten der dabei genutzten Rechner berücksichtigt werden, die in der folgenden Tabelle zusammengefaßt sind:

	CRAY T3E	ZAMpano	Notebook
Prozessortyp	DEC Alpha 21164	Intel Pentium III Xeon	Intel Pentium III
Taktfrequenz	600 MHz	550 MHz	1 GHz
Peak-Leistung je Prozessor	1.2 GFlop/s	550 MFlop/s	1 GFlop/s
Hauptspeicher je Knoten	512 MB	2 GB	512 MB
Cache-Größe (Level 2)	96 KB	512 KB	256 KB
Anzahl Prozessoren	512	8 * 4	1
Gesamtleistung	614 GFlop/s	19.8 GFlop/s	
Gesamthauptspeicher	262 GB	16 GB	
Bandbreite des internen Netzwerks	312 MB/s	(int/ext) 175/130 MB/s	
Latenz des Netzwerks (MPI)	3.75 μs	2.5/16 μs	
Betriebssystem	UNICOS/mk	SUSE Linux 7.2	SUSE Linux 7.3

Der Parallelrechner CRAY T3E besitzt zusätzlich zu den 512 Rechenknoten noch weitere 28 Knoten, die für das Betriebssystem und das interaktive Arbeiten genutzt werden. Das Betriebssystem bietet dabei ein Single-System-Image. Das heißt, daß der CRAY T3E sich dem Benutzer als ein Rechner präsentiert. Dagegen ist das Linux-SMP-Cluster in 8 Knoten mit je vier Prozessoren aufgeteilt, auf denen jeweils ein Betriebssystem läuft. Für die Messung der Verarbeitungsgeschwindigkeiten wurde dort jeweils ein Knoten benutzt, auf dem ein Prozessor den Simulations-Client ausführt und die restlichen drei Prozessoren nicht genutzt werden. Auf dem CRAY T3E wurde einer der 512 Rechenknoten für die Ausführung des künstlichen Clients benutzt. Die Messungen wurden im laufenden Betrieb des Rechner durchgeführt. Dadurch können insbesondere bei den Übertragungszeiten kleinere Schwankungen auftreten, da die Kommunikationspfade gleichzeitig auch durch andere Anwendungen genutzt werden.

6.2.1 Verarbeitungszeit der schnellen Wavelet-Transformation

Im ersten Komprimierungsschritt wird mit Hilfe des Pyramidenalgorithmus die schnelle Wavelet-Transformation (FWT) ausgeführt. Wesentlich ist hierbei, daß für einen dreidimensionalen Datensatz pro Iteration die eindimensionale FWT jeweils für alle Vektoren in X-, Y- und Z-Richtung ausgeführt werden muß. Bei dem als Beispieldatensatz genutzten Geschwindigkeitsfeld wird die FWT für die Komponenten des Geschwindigkeitsvektors jeweils einzeln ausgeführt. Das heißt, daß sich das Geschwindigkeitsfeld für die FWT als drei getrennte skalare Felder darstellt.

Abbildung 6.5 zeigt die Verarbeitungsgeschwindigkeiten für die beiden Parallelrechner. Es ist zu erkennen, daß bei beiden Rechnern die Verarbeitungszeit im wesentlichen linear mit der Größe der Daten ansteigt. Die im Modell angenommen Linearität wird damit bestätigt.

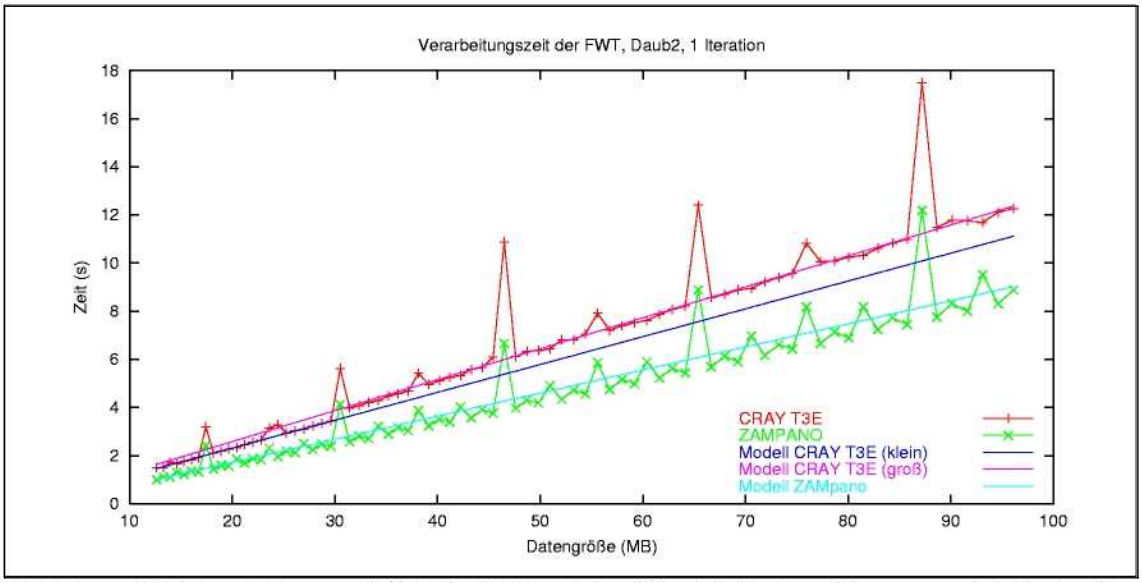


Abbildung 6.5: Verarbeitungszeit für die FWT auf der Client-Seite: Das Diagramm zeigt die Ausführungszeit für den ersten Schritt der Komprimierung für die beiden Rechner CRAY T3E und ZAMpano. Es wurde Peaks dabei eine Iteration des Pyramidenalgorithmus ausgeführt. Die sind eine Folge des Stride-Effekts (siehe Text).

Beide Kurven enthalten aber markante Punkte, an denen sich die Verarbeitungszeit beträchtlich erhöht (Peaks). Zur genaueren Erläuterung dieser Peaks sind in Abbildung 6.6 auf der nächsten Seite die Verarbeitungszeit der drei Durchläufe des Pyramidenalgorithmus (in X-, Y- und Z-Richtung) einzeln aufgeführt. Bei beiden Rechnern werden die meisten Verzögerungen durch den dritten Durchlauf (Z-Vektoren) verursacht, bei dem nacheinander auf im Speicher weit auseinander liegende Daten zugegriffen wird. Wird dabei ein Abstand benutzt, der einem Vielfachen von 2^i entspricht, kann es Stride-Effekten kommen, die den Ablauf stark verzögern und durch den Cache verursacht werden.

Für das Verständnis dieser Stride-Effekte ist eine genauere Betrachtung des Cache-Mechanismus notwendig, die hier am Beispiel der T3E-Knoten erfolgt: Bei dem CRAY T3E sind die Knoten mit einem dreifach assoziativen Second-Level-Cache der Größe 96 KByte ausgerüstet. Dies bedeutet, daß er in 512 dreifach ausgelegte Cache-Zeilen mit jeweils 64 Byte aufgeteilt ist. Beim Zugriff auf Daten, die sich nicht im Cache befinden, werden immer Blöcke dieser Größe (64 Byte) vom Speicher in eine Cache-Zeile geladen. Es gibt eine feste Zuordnung der Speicheradressen zu den Cache-Zeilen. Da der Cache-Speicher sehr viel kleiner ist als der Hauptspeicher, gibt es eine Mehrfachzuordnung zu den Cache-Zeilen, so daß bei einer neuen Belegung einer Cache-Zeilen alte, sich dort befindende Daten ausgelagert werden. Die Beschleunigung des Speicherzugriffs durch den Cache wird immer dann ausgehebelt, wenn bei nachfolgenden Zugriffen immer dieselbe der

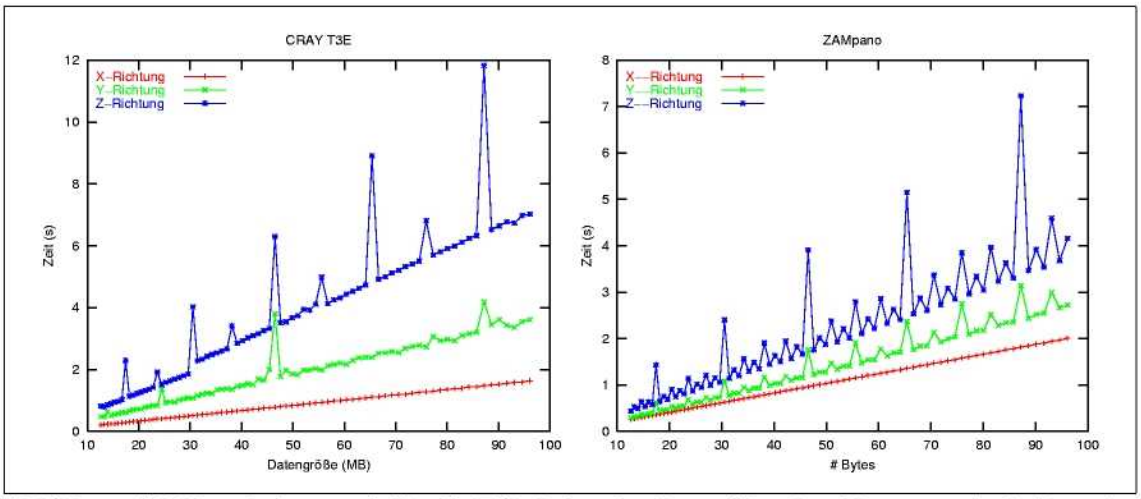


Abbildung 6.6: Verarbeitungszeit für die Teilschritte des Pyramidenalgorithmus: In den beiden Diagrammen ist jeweils die Verarbeitungszeit für den Durchlauf des Pyramidenalgorithmus in X-, Y- und Z-Richtung aufgetragen.

512 Speicherstellen des Caches ausgewählt wird. Da die Zuordnung der Speicherstellen zu den Cache-Zeilen zyklisch erfolgt, treten diese Stride-Effekte genau dann auf, wenn bei dem Zugriff auf den Speicher ein Abstand von 512 * 64 Byte = 32 KByte oder ein Vielfaches davon gewählt wird.

Genau dieser Abstand ergibt sich bei der Messung bei dem FWT-Durchlauf in Z-Richtung bei s=46.5 MB. Da damit der Cache den Zugriff auf den Speicher nicht beschleunigen kann, verlangsamt sich die Ausführung, was zu einem Peak in der Kurve an dieser Stelle führt. Diese Effekte treten in vermindertem Ausmaß auch dann auf, wenn der Abstand die Hälfte oder ein Viertel des oben beschriebenen Abstands beträgt. Dann wird bei jedem zweiten oder vierten Zugriff dieselbe Cache-Zeile ausgewählt. Auf diese Weise lassen sich auch die anderen Peaks in der Kurve erklären. Auch beim zweiten Durchlauf durch die Daten treten solche Peaks auf, die wiederum durch Stride-Effekte begründet sind. Bei den Messungen der Verarbeitungszeit auf dem ZAMpano treten diese Stride-Effekte bereits bei kleineren Abständen auf.

An dem Verlauf der Verarbeitungszeit auf dem CRAY T3E ist zu erkennen, daß sich der Zugriff auf die Daten im Hauptspeicher mit einem großen Zugriffsabstand im dritten Durchlauf des Pyramidenalgorithmus auf dem CRAY T3E ab etwa 32 MB deutlich verlangsamt, wobei beide Bereiche sich durch eine Gerade beschreiben lassen. Eine Ursache dafür ließ sich nicht eindeutig zuordnen. In der Herstellerliteratur [57, 58] wird darauf hingewiesen, daß Zugriffe mit sehr großen ungeraden Strides, wie sie hier auftreten, auf dem CRAY T3E zu einer geringeren Memory-Bandbreite führen können. Für die Bestimmung der Modellparameter sollten daher für den CRAY T3E die Bandbreite und Latenz für die beiden Bereiche einzeln bestimmt werden. Insbesondere bei der Ankopplung an parallele Simulationsprogramme mit verteilter Datenhaltung liegen die Datengrößen der lokalen Teilgebiete bei den folgenden Messungen stets im Bereich der ersten etwas flacheren Gerade. Für die beiden Geraden des CRAY T3E ergeben sich folgende, mit Hilfe der linearen Regression bestimmte Werte für die Modellparameter: Bandbreite b_{fwt}=7.77 MB/s bzw. 8.64 MB/s und Latenz $t_{l,\text{fwt}}$=1.1 ms bzw. 12.3 ms.

Auch bei der auf ZAMpano gemessenen Kurve ist – bis auf die durch Stride-Effekte erzeugten Peaks – ein linearer Verlauf zu erkennen. Bei der Anpassung der Modellgeraden durch die lineare Regression ergibt jedoch sich für den Modellparameter $t_{l,\text{fwt}}$ unerwarteter Weise ein negativer Wert (-202 ms). Das liegt daran, daß bei Datengrößen kleiner 4 MB die gemessene Kurve nicht mehr linear ist, wie das linke Diagramm der Abbildung 6.7 auf der nächsten Seite zeigt. Ursache dafür ist die Benutzung des Caches, der insgesamt die Verarbeitung der Daten beschleunigt und

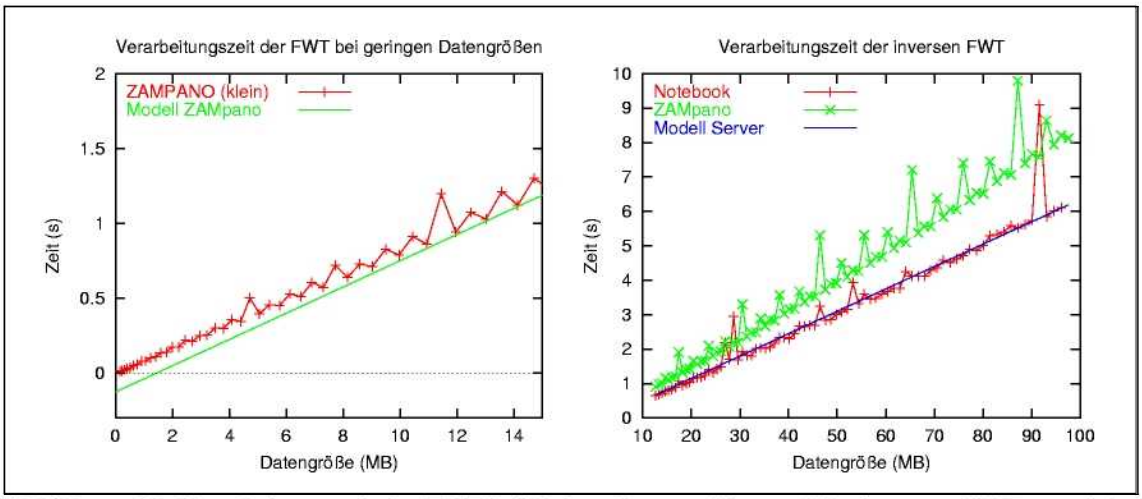


Abbildung 6.7: Verarbeitungszeit der FWT bei kleinen Datengrößen und der inversen FWT: Das linke Diagramm zeigt die Verarbeitungszeit der FWT bei sehr kleinen Datenmengen. Durch den Cache wird die Verarbeitungszeit so verringert, daß eine Abschätzung der Verarbeitungszeit mit Hilfe einer Geraden zu einer negativen Latenz führt. Das rechte Diagramm zeigt die Verarbeitungszeit der inversen FWT auf dem Notebook und zum Vergleich auf einem Knoten des ZAMpano.

so bei allen Messungen die Zeiten verringert. Da die gemessene Kurve in dem für diese Arbeit wichtigen Bereich der großen Datenmengen linear ist und gut mit einer Geraden modellierbar ist, wird die negative Latenz, die sich formal aus der linearen Regression ergibt, in den weiteren Rechnungen benutzt. Es ergibt sich damit für ZAMpano eine Latenz von $t_{l,\mathrm{fwt}}$=-202s und eine Bandbreite von b_{fwt}=11.41 MB/s.

Im rechten Diagramm der Abbildung 6.7 ist die Verarbeitungszeit der inversen FWT auf der Server-Seite aufgetragen. Zum Vergleich wurde dieser Verarbeitungsschritt auch auf ZAMpano ausgemessen. Die fast doppelt so hohe Taktrate des Notebooks gleicht den nur halb so großen L2-Cache des Notebook-Prozessor gegenüber den Xeon-Prozessoren des ZAMpano aus und führt insgesamt zu einer geringeren Ausführungszeit. Für die Modellparameter ergibt sich auf dem Notebook eine Bandbreite von b_{ifwt}= 15.2 MB/s und ähnlich wie beim ZAMpano eine negative Latenz von $t_{l,\mathrm{ifwt}}$=-197 ms.

6.2.2 Verarbeitungszeit für das Abschneiden kleiner Koeffizienten

Für das Abschneiden kleiner Koeffizienten wird vom Benutzer als Maximalgrenze der Anteil der abzuschneidenden Koeffizienten angegeben. Für die Umrechnung dieses Prozentsatzes auf einen Absolutwert, unter dem die abzuschneidenden Koeffizienten liegen müssen, wird die Verteilung der Koeffizientenwerte benötigt. Die Berechnung dieses Histogramms wird in einem eigenen Schritt durchgeführt. Abbildung 6.8 auf der nächsten Seite zeigt die Verarbeitungszeit beider Teilschritte.

Die Verarbeitungszeit für die Histogrammberechnung liegt bei beiden Rechnern dabei höher als die Zeit zum Abschneiden der kleinen Koeffizienten. Dies liegt daran, daß bei Histogrammberechnung während des Durchlaufs auf zwei verschiedene Bereiche im Speicher (Daten und Histogramm) zugegriffen werden muß. Das kann dazu führen, daß die Histogramm-Werte immer wieder aus dem Cache verdrängt werden und beim nächsten Zugriff aus dem Hauptspeicher neu geladen werden müssen. Es ergeben sich als Summe beider Teilschritte für CRAY T3E b_{cut}=28.05 MB/s und $t_{l,\mathrm{cut}}$=2.6 ms sowie für ZAMpano b_{cut}=10.6 MB/s und $t_{l,\mathrm{cut}}$=1.1 ms.

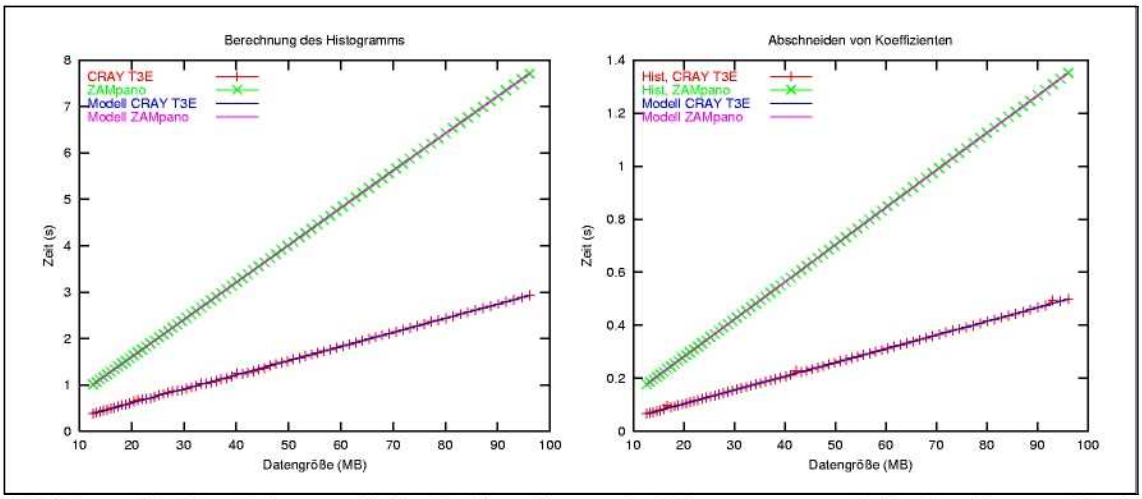


Abbildung 6.8: Verarbeitungszeit für die Generierung des Histogramms und Abschneiden von Koeffizienten: Die Kurven in beiden Diagrammen können mit einer Geraden abgeschätzt werden. Da in beiden Fällen die Datenelemente in ihrer Speicherreihenfolge durchlaufen werden können, steigt die Verarbeitungszeit linear mit der Größe.

6.2.3 Verarbeitungszeit der Quantisierung und Kodierung

Bei der Implementierung wurden die beiden Verarbeitungsschritte der Quantisierung und der Kodierung in einem Durchlauf zusammengefaßt, so daß die quantisierten Daten vor der Kodierung nicht mehr zwischengespeichert werden müssen. Abbildung 6.9 auf der nächsten Seite zeigt die entsprechende Verarbeitungsgeschwindigkeit für diesen Durchlauf. Auch hier nimmt die Zeit linear mit der Datengröße zu und bestätigt damit die Modellannahmen.

Mit den in der Messung benutzten Einstellungen wurde der Pyramidenalgorithmus einmal durchlaufen, so daß bei der Transformation zwei Frequenzbänder entstehen. Neben Quantisierung und Kodierung der höheren Frequenzbänder wird in diesem Schritt auch das unterste Frequenzband berücksichtigt. Da dieses keine Koeffizienten enthält, die potentiell auf Null gesetzt werden können, entfällt hier die Kodierung durch eine Bitmaske. Bei der Kodierung der in den höheren Frequenzbändern enthaltenen Koeffizienten werden jeweils die in den sieben Differenzbildern enthaltenen Koeffizienten einem Element aus dem nächst niedrigeren Frequenzband zugeordnet (siehe auch Abschnitt 3.3 auf Seite 37). Das heißt, daß bei dem Durchlauf der entsprechende Hauptspeicherbereich nicht linear durchlaufen wird. Vielmehr wird bei dem Zugriff auf die sieben Koeffizienten ein Stride benutzt, welcher der Kantenlänge des nächst niedrigeren Frequenzbands entspricht. Dadurch lassen sich ähnliche Stride-Effekte erklären, wie sie auch bei der Verarbeitungszeit der FWT auf den beiden Rechnern auftreten.

Für CRAY T3E ergibt sich eine Bandbreite von $b_{\text{quant\_code}}$=57.18 MB/s und eine Latenz von $t_{l,\text{quant\_code}}$=6.0 ms sowie für ZAMpano $b_{\text{quant\_code}}$=50.85 MB/s und $t_{l,\text{quant\_code}}$=1.6 ms.

Auf beiden Rechnern werden für die Speicherung einer Gleitkommazahl acht Byte benötigt. Bei einer 8-Bit Quantisierung ergibt sich dadurch eine Reduzierung der Datengröße um den Faktor 8. Im Verarbeitungsschritt „Abschneiden von Koeffizienten“ wurden in der Messung 80 % der Koeffizienten auf Null gesetzt, was bei der Kodierung zu einer zusätzlichen Reduzierung der Datengröße führt. Abbildung 6.10 auf der nächsten Seite zeigt die nach der Quantisierung und Kodierung erhaltene Datengröße. Sie ist für die in der Messung benutzten Daten um den Faktor 18.8 geringer als die Größe der Originaldaten. Da innerhalb der Messung eine Iteration des Pyramidenalgorithmus ausgeführt wurde, ist ein Frequenzband mit Koeffizienten und ein Frequenzband

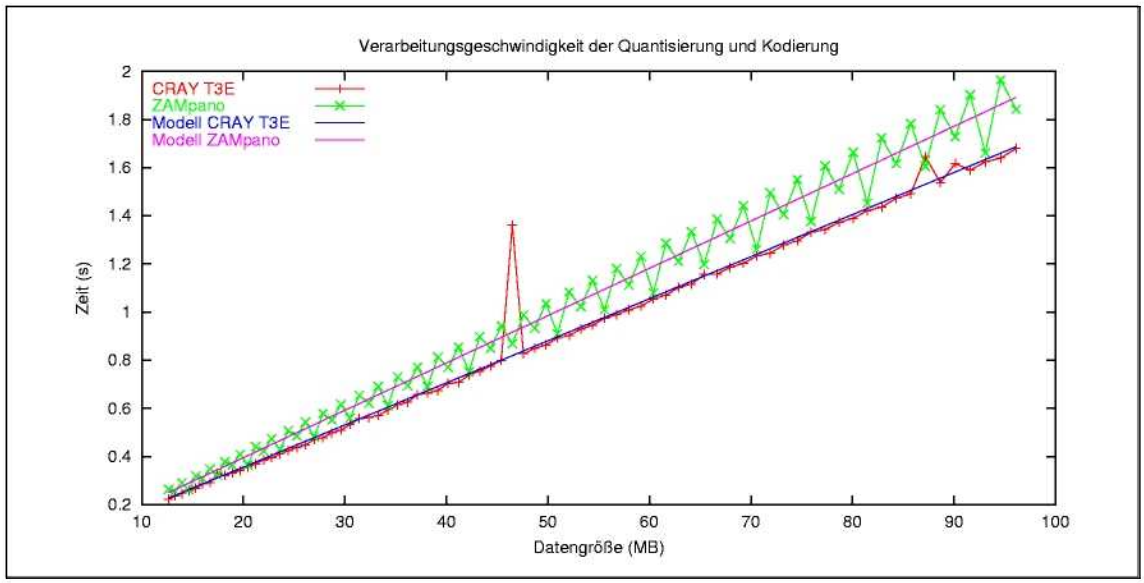


Abbildung 6.9: Verarbeitungszeit der Quantisierung und der Kodierung: Die beiden Verarbeitungsschritte werden wegen des geringeren Speicherbedarfs in einem Durchlauf abgearbeitet. Die Messungen ergeben für beide Parallelrechner einen linearen Verlauf. Ausnahme bilden hier nur die Stellen, an denen mit einem ungünstigen Stride auf die Daten im Hauptspeicher zugegriffen wird.

mit gemittelten Daten entstanden. Für die Kompressionrate r_{wave} ergibt sich somit:

$$r_{\text{wave}} = r\frac{1}{n_{\text{double}}}\left(0.2\frac{7}{8}\,n_{\text{quant}} \;+\; \frac{1}{8}\,n_{\text{mask}} \;+\; \frac{1}{8}\,n_{\text{quant}}\right) \tag{6.2}$$

Dabei entspricht der erste Term der Klammer den von Null verschiedenen quantisierten Koeffizienten, der zweite Term den zugehörigen Bitmasken und der dritte Term den quantisierten Werten aus dem unteren Frequenzband. Der Faktor r beschreibt die LZO-Kompressionrate des nachfolgenden Verarbeitungsschrittes. Mit den Werten n_{double}=8, n_{quant}=n_{mask}=1 ergibt sich für r_{wave}=$r*1/18.824$. Wird mit dem Pyramidenalgorithmus die maximal mögliche Anzahl von Iterationen berechnet, kann entfällt der dritte Term aus der obigen Gleichung und die beiden restlichen Terme nähern sich an die in der Modellierung angegebenen Terme an (Glg. 4.32 auf Seite 59).

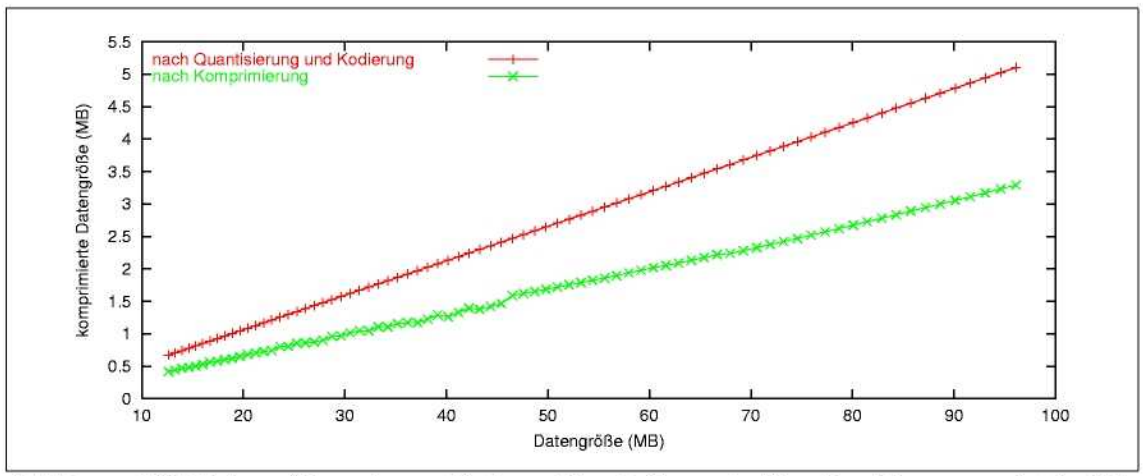


Abbildung 6.10: Datengröße nach verschiedenen Verarbeitungschritten: Das Diagramm zeigt die Größe der Daten nach der Quantisierung und Kodierung sowie nach der Komprimierung mit Hilfe des LZO-Verfahren. Nach der Quantisierung und Kodierung sind die Daten um den Faktor 18.8 kleiner als die Originaldaten. Die LZO-Komprimierung verringert die Datengröße nochmals um einen Faktor von $r = 1.6$.

6.2.4 Verarbeitungszeit der Komprimierung

Im letzten Verarbeitungsschritt werden die quantisierten und kodierten Daten mit Hilfe des LZO-Verfahrens weiter komprimiert. Bei diesem Verfahren wird der zu komprimierende Byte-Stream nach wiederkehrenden Mustern untersucht, um diese nur einmal speichern zu müssen. Dazu wird das erste Muster in die Ausgabedaten kopiert und an den nachfolgenden Stellen nur ein Verweis auf die erste Kopie gespeichert. In der in dieser Arbeit benutzen LZO-Implementierung wird die Mustersuche mit Hilfe eines Hash-Verfahrens realisiert, so daß ein mit der Datengröße in etwa linear anwachsender Zeitverbrauch zu erwarten ist. Im Detail kann die Verarbeitungszeit von Anzahl, Größe und Häufigkeit der in den Daten enthaltenen Muster abhängen, so daß Abweichungen von einem linearen Verlauf der Kurve zu erwarten sind. Die Zeitmessungen auf jeweils einem Prozessor von ZAMpano und CRAY T3E zeigen im wesentlichen dieses Verhalten (siehe Abb. 6.11). Für die Verarbeitungsgeschwindigkeit können für den CRAY T3E b_K=3.1 MB/s und $t_{l,K}$=19 ms sowie für ZAMpano b_K=14.20 MB/s und $t_{l,K}$=11.8 ms angenommen werden. Die Komprimierungsrate des LZO-Verfahrens liegt bei 1/1.6.

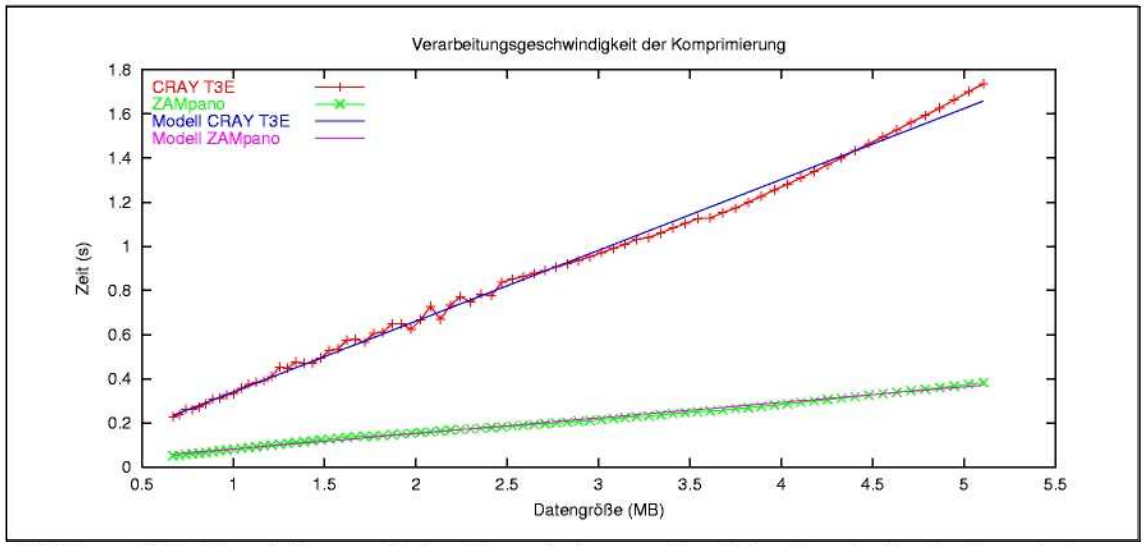


Abbildung 6.11: Verarbeitungszeit der Komprimierung: Der Zeitverbrauch für die Komprimierung steigt auf beiden Rechnern in etwa linear mit der Datengröße an.

6.2.5 Zusammenfassung der Teilergebnisse

Aus den oben ausgeführten Messungen der einzelnen Verarbeitungsschritte ergeben sich folgende Bandbreiten und Latenzen für die beiden Parallelrechner:

	CRAY T3E		ZAMpano	
	Bandbreite (MB/s)	Latenz (ms)	Bandbreite (MB/s)	Latenz (ms)
FWT	7.77 / 8.64	1.1 / 12.3	11.41	-202.0
Histogramm	32.85	2.0	12.46	0.7
Abschneiden	192.9	0.6	71.10	0.4
Quant.&Kodierung	57.18	6.0	50.85	1.6
Komprimierung	3.11	19.0	14.20	11.8
Senden (10 MBit/s)	0.77	9.4	0.90	1.1
Senden (100 MBit/s)	3.47	7.0	10.76	0.8

Die Gesamtbandbreite b_{wave} für die Komprimierung und Übertragung der Daten ergibt durch das

Einsetzen der in der Tabelle aufgeführten Werte in folgende aus 4.36 abgeleitete Gleichung:

$$\frac{1}{b'_{\text{wave}}} = \frac{1}{b_{\text{fwt}}} + \frac{1}{b_{\text{cut}}} + \frac{1}{b_{\text{quant\_code}}} + r_{\text{wave}}\frac{1}{b_K} + r\,r_{\text{wave}}\frac{1}{b_n} \tag{6.3}$$

Für die beiden Parallelrechner ergibt sich bei r_{wave}=1/18.824 und r=1/1.6 folgende Werte für Bandbreite und Latenz:

	ZAMpano		CRAY T3E	
	10 MBit/s	100 MBit/s	10 MBit/s	100 MBit/s
Bandbreite b'_{wave}	4.12 MB/s	4.79 MB/s	4.13 MB/s	4.80 MB/s
Latenz $t_{l,\text{wave}}$	-186.1 ms	-186.4 ms	38.10 ms	35.70 ms

Die beiden Abbildungen 6.12 und 6.13 auf der nächsten Seite vergleichen die aus den obigen Modellparametern abgeleiteten Geraden mit den gemessenen Gesamt-Verarbeitungszeiten auf jeweils einem Prozessor der beiden Parallelrechner. Die Messungen bestätigen somit die Modellannahme,daß sich die unkomprimierte wie auch die komprimierte Datenübertragung durch eine Gerade beschreiben lassen, die durch Bandbreite und Latenz bestimmt wird.

Die hier bestimmten Modellparameter für die einzelnen Verarbeitungsschritte werden im nächsten Abschnitt auch für die Bestimmung der Modelle für die Speedup-Kurve paralleler und an eine Visualisierung angekoppelter Simulationsprogramme herangezogen. Bei dem Vergleich mit der Bandbreite und der Latenz von der Übertragung ohne Komprimierung (b_h=0.9 MB/s und t_l=1.1 ms) ergeben sich klare Vorteile für den Einsatz der Komprimierung. Im Durchschnitt ist die Bandbreite beim Einsatz der Komprimierung um den Faktor 5 größer als ohne Komprimierung.

Bei einem Vergleich der komprimierten Übertragung zu einer unkomprimierten Übertragung bei einem 100 MBit/s-Anschluß des Notebooks ist beim ZAMpano die direkte Übertragung schneller, da es sich um einen direkten, nur durch Switches unterbrochenen Übertragungsweg zum Notebook handelt. Auf dieser Strecke kann eine Übertragungsrate von über 10 MB/s erreicht werden. Ein solcher dedizierter Übertragungsweg steht üblicherweise nur innerhalb lokaler Netze zur Verfügung. Bei dem CRAY T3E bringt die Komprimierung sogar bei einem 100 MBit/s-Anschluß des Notebooks Vorteile, da die Bandbreite durch die I/O-Leistung des Parallelrechners bereits auf ca. 3 MB/s begrenzt wird.

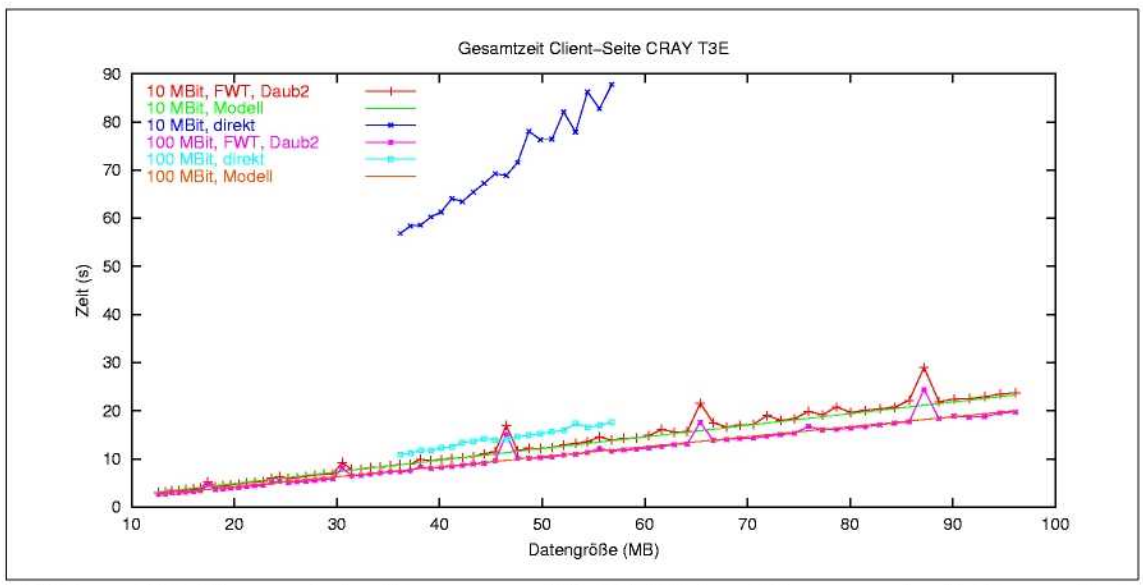


Abbildung 6.12: Gesamtzeit der Übertragung (CRAY T3E): Das Diagramm zeigt die Kommunikationszeit, die für die Übertragung der Daten von dem CRAY T3E zur Visualisierung benötigt werden. Die Schwankungen der Übertragungzeit bei einem 10 MBit/s-Anschluß erklären sich dadurch, daß die Messungen während des laufenden Benutzerbetriebs des Parallelrechners stattfanden.

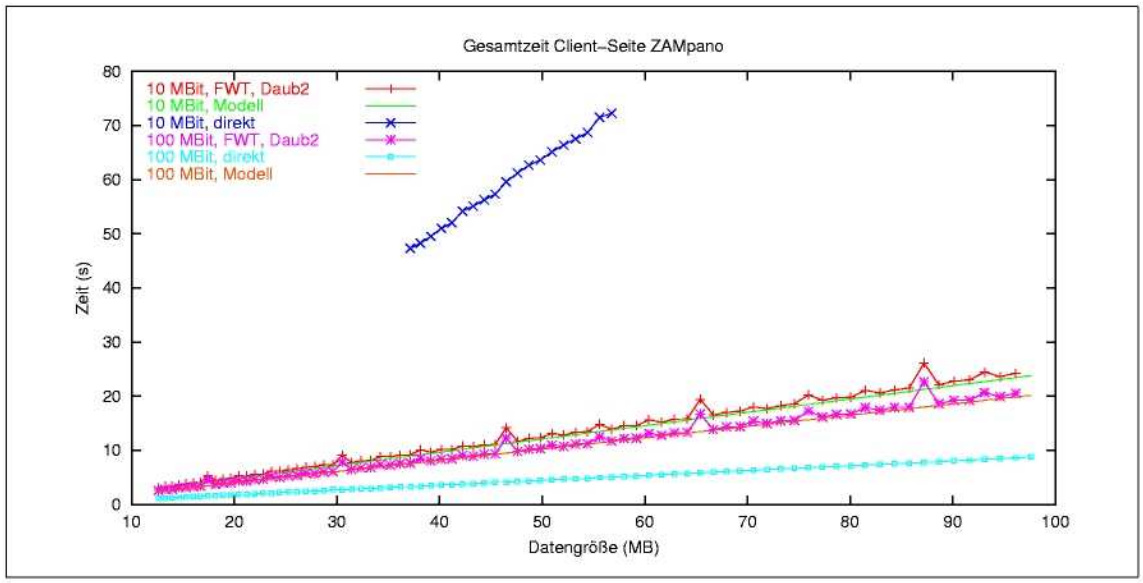


Abbildung 6.13: Gesamtzeit der Übertragung (ZAMpano): Das Diagramm zeigt die Kommunikationszeit für die Übertragung der Daten von dem Parallelrechner ZAMpano zum Notebook. Die Kurven für die komprimierte Übertragung bei einem 10 und einem 100 MBit/s-Anschluß des Notebook unterscheiden sich deshalb nur geringfügig, da die komprimierten Daten eine geringe Größe besitzen und in beiden Fällen die Übertragung nur einen kleinen Beitrag zu der Gesamtzeit liefern.

6.3 Messungen bei einem parallelen Simulationsprogramm

Im vorigen Abschnitt wurde eine lineare Abhängigkeit der Verarbeitungszeit der Datenreduzierung von der Datengröße entsprechend den Voraussagen des Modells nachgewiesen. Es bleibt weiterhin zu zeigen, daß bei einer Ankopplung an ein paralleles Simulationsprogramm dessen Beeinflussung mit dem im Modell entwickelten Speedup-Verlauf übereinstimmt. Aus den für die Untersuchungen zur Verfügung stehenden Simulationsprogrammen wurde dazu die Anwendung Trace ausgewählt, deren Ergebnisdaten bereits im vorherigen Abschnitt für die Ausmessung der einzelnen Verarbeitungsschritte benutzt worden sind.

6.3.1 Beschreibung des Simulationsprogramms Trace

Das im Institut für Agrosphäre (FZJ/ICG IV) entwickelte Simulationsprogramm Trace [59] berechnet für einen Bodenausschnitt die Strömung im Grundwasserleiter. Die Eigenschaften des Bodens sowie die Randbedingungen für das in der Rechnung benutzte Simulationsgebiet werden aus Feld- und Laborexperimenten gewonnen und als statistische Parameter in die Simulationsrechnung eingebracht. Für die Parallelisierung der in Trace verwendeten Finite-Elemente-Methode wird das Simulationsgebiet auf die einzelnen Prozessoren verteilt. Diese bilden dann das physikalische Simulationsgebiet so ab, daß sich die Teilgebiete um jeweils einen Gitterknoten überlappen. In der iterativen Berechnung des Geschwindigkeitsfeldes wird ein paralleles Verfahren der konjugierten Gradienten (CG) benutzt, für das in jeder Iteration die Ränder des lokal gespeicherten Teilgebiets mit den Nachbarknoten ausgetauscht werden müssen.

Als Testbeispiel für die Speedup-Vergleiche wurde ein Simulationsgebiet mit 254 * 126 * 62 FE-Knoten ausgewählt. Innerhalb des Simulationsgebiets wurden mehrere Brunnen und Wasserquellen angeordnet, die zu einem heterogenen und zeitabhängigen Geschwindigkeitsfeld führen (siehe auch Abb. 6.4 auf Seite 93). Die Größe und die Auflösung des Simulationsgebiets wurde so gewählt, daß für kleine Prozessorzahlen eine noch akzeptable Laufzeit erreicht wird und auch bei

höheren Prozessorzahlen (128 bzw. 256 auf dem CRAY T3E) noch genügend große Teilgebiete für die einzelnen Prozessoren entstehen. Für die Vergleiche werden jeweils 7 Simulationsschritte durchgeführt, die einer Simulationszeit von einem Tag entsprechen. Innerhalb jedes Zeitschritts werden für die nötige Genauigkeit der Lösung im Mittel 95 CG-Iterationen benötigt, bei denen jeweils der Rand der lokalen Simulationsgebiete ausgetauscht wird.

Trace ist in FORTRAN 90 implementiert, so daß für die Ankopplung der LVISIT-Routinen deren Fortran-Interface genutzt wird. Für die Tests ist in der Konfigurationsdatei des Code-Generators ein Datenstrom für das von Trace berechnete Geschwindigkeitsfeld definiert. In diesem dreidimensionalen Feld sind für jeden FE-Knoten die drei Komponenten des Geschwindigkeitsvektors abgespeichert. Auf der Visualisierungsseite wurde wie im vorherigen Abschnitt ein künstlicher Server benutzt, der wie ein Visualisierungsprogramm die Daten von der Gegenseite annimmt und rekonstruiert. Durch die Verwendung des künstlichen Servers ist es möglich, die Messungen für verschiedene Parameter und Anordnungen zu wiederholen, ohne daß eine Interaktion mit der Visualisierungsoberfläche nötig ist. Bei den Messungen wurde das Simulationsprogramm auf dem CRAY T3E ausgeführt, der durch seine hohe Anzahl von Einzelprozessoren gegenüber ZAMpano eine größere Variation der Prozessorzahlen ermöglicht.

6.3.2 Messung ohne Ankopplung

Zur Bestimmung der Modellparameter wurde zuerst der Verlauf der Speedup-Kurve für einen Lauf des Simulationsprogramms ohne Ankopplung bestimmt. Aus den Meßwerten konnten Näherungslösungen für die Modellparameter T_S und f durch Einsetzen in die auf Amdahl's Law beruhende Modellgleichung 4.48 auf Seite 65 gefunden werden. Eine direkte Bestimmung der sequentiellen Laufzeit des Programms ist nicht möglich, da innerhalb von Trace im sequentiellen Fall andere Algorithmen benutzt werden. Daher wurden für die Bestimmung der Modellparameter die gemessenen Speedup-Werte bei 2 und 128 Prozessoren zugrunde gelegt. Es ergeben sich damit T_S=612.9 s und f= 0.000792 (siehe Abb. 6.14). Dabei geht in die in Gleichung 4.49 beschriebene Abschätzung für die interne Kommunikationszeit von Trace eine Datengröße von 50 MB ein. Dies entspricht der Summe aller lokalen Teilfelder bei einer Speicherlänge von 8 Byte für eine Gleitkommazahl.

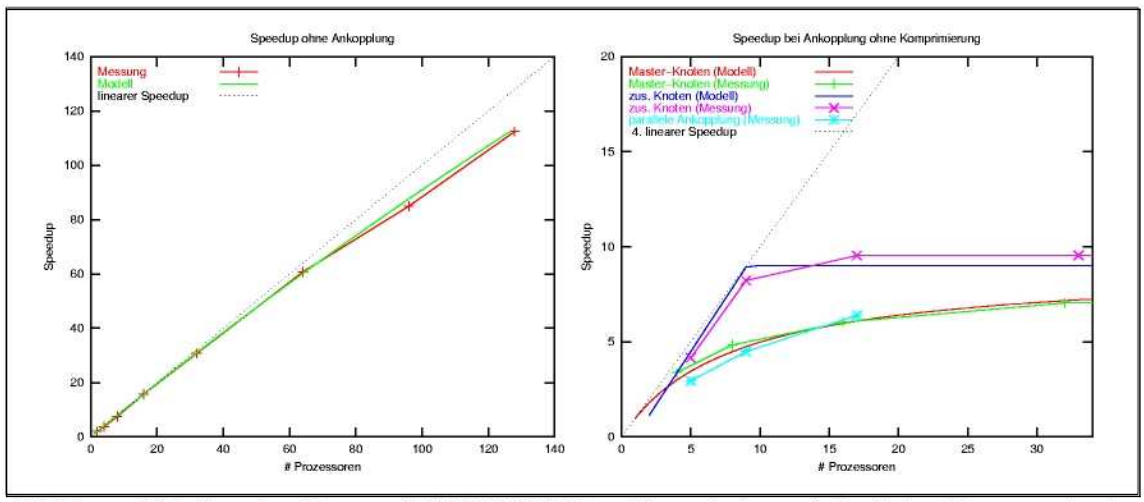


Abbildung 6.14: Speedup Trace auf CRAY T3E (ohne Komprimierung): Im linken Diagramm ist der Verlauf der gemessenen und des aus der Modellgleichung bestimmten Speedups aufgetragen. Die beiden Kurven stimmen sehr gut überein. Im rechten Diagramm sind die Speedup-Kurven bei verschiedenen Kopplungsstrategien ohne Komprimierung der Daten aufgetragen. Bei allen drei Kopplungsstrategien wird der Speedup durch die unkomprimierte Datenübertragung auf einen Wert unter 10 begrenzt.

6.3.3 Messung bei Ankopplung ohne Komprimierung

In der nächsten Messung wurde bei der Ankopplung der Visualisierung keine Komprimierung der Daten verwendet. Das bedeutet, daß nach jedem Simulationsschritt 50 MB Daten zu der Visualisierungsworkstation übertragen werden müssen. Bei der in der Messung verwendeten 10 MBit/s-Variante der Verbindung zum Notebook ergibt dies eine Übertragungszeit von etwa 80 Sekunden. Diese Zeit muß unabhängig von der verwendeten Prozessorzahl aufgebracht werden. Bei der Ankopplung über einen zusätzlichen Kommunikationsknoten kann diese Zeit teilweise mit den Simulationsrechnungen des nächsten Zeitschritts überlagert werden. Dadurch ergeben sich für den Speedup dieser Kopplungsstrategie bessere Werte (siehe Abb. 6.14 auf der vorherigen Seite).

Bei den zu übertragenden Daten handelt es sich bei dieser unkomprimierten Übertragung um Double-Werte. Um die unterschiedliche interne Darstellung der Werte auf CRAY- und Notebook-Seite auszugleichen, werden die Daten im VISIT-Server konvertiert. Da diese Konvertierungszeit mit zu der Übertragungszeit gezählt wird, liegt die Übertragungsbandbreite für Double-Werte unter der für die in der Ausmessung der VISIT-Bandbreite verwendeten Byte-Daten, bei denen keine Konvertierung nötig ist. Für die Verbindung zwischen CRAY T3E und Notebook mit zwischengeschalteten 10 MBit/s-Repeater reduziert sich die Bandbreite von 6.23 MBit/s auf 5.23 MBit/s bei Double-Werten. Bei dem Vergleich der Speedup-Kurven mit den Modellgleichungen wurde diese reduzierte Bandbreite benutzt. Das rechte Diagramm der Abbildung 6.14 auf der vorherigen Seite zeigt den gemessenen und den modellierten Verlauf der Speedup-Kurven. Auch bei der parallelen Übertragung der Daten ist der Speedup begrenzt, da die Gesamt-Bandbreite zum Notebook auf 10 MBit/s begrenzt ist. In 4.54 auf Seite 67 4.54 entspricht dies dem Fall $b_{n,p} = 10/N$ MBit/s. Messungen mit mehr als 16 Prozessoren bei der parallelen Ankopplung waren auf dem Rechner CRAY T3E nicht möglich, da für die Socket-Kommunikation Ressourcen (z.B. Buffer auf den Betriebssystemsknoten) benötigt werden, die mit anderen gleichzeitig ablaufenden Programmen geteilt werden müssen und der an den Benutzerbetrieb angepaßten Konfiguration der Maschine nur begrenzt zur Verfügung stehen. Insgesamt wird der Speedup bei allen drei Kopplungsstrategien auf einen Wert begrenzt, der unter 10 liegt.

6.3.4 Messung bei Ankopplung mit Komprimierung

Dieser ungünstige Effekt bei den Speedup-Kurven wird durch die in dieser Arbeit beschriebene und implementierte Datenreduzierung und anschließende Komprimierung der Daten stark verringert. Da durch die Datenreduzierung die Größe der zu übertragenden Daten verringert wird, fällt die dabei entstehende Übertragungzeit bei der Betrachtung der Speedup-Kurven nicht mehr so stark ins Gewicht. Im vorigen Abschnitt wurde gezeigt, daß Komprimierungsfaktoren von bis zu 30 möglich sind. Dem dadurch erreichten Zeitgewinn stehen aber zusätzliche Verarbeitungszeiten durch die Komprimierungszeit gegenüber. In den folgenden Messungen wird gezeigt, daß diese Zeit aber nur ein Bruchteil der Übertragungzeit der nicht komprimierten Daten ausmacht.

Für die Messung der Ankopplung mit Komprimierung wurde bei allen drei Kopplungsstrategien eine Iteration des Pyramidenalgorithmus ausgeführt, danach 80 % der dabei entstandenen Koeffizienten abgeschnitten und anschließend eine Quantisierung mit 8 Bit Genauigkeit durchgeführt. Diese Einstellungen entsprechen denjenigen, die bei der Ausmessung der einzelnen Verarbeitungsschritte benutzt worden sind. Damit können die dort bestimmt Modellparameter auch für die Modellierung der Speedup-Kurven benutzt werden. Abbildung 6.15 auf der nächsten Seite zeigt die Meßergebnisse und die modellierten Speedup-Kurven für die Ankopplung mit Komprimierung.

Bei der Ankopplung über den Master-Knoten ist eine deutliche Reduzierung des Speedups durch die Datenübertragung zur Visualisierung zu erkennen, die mit wachsender Prozessorzahl zunimmt.

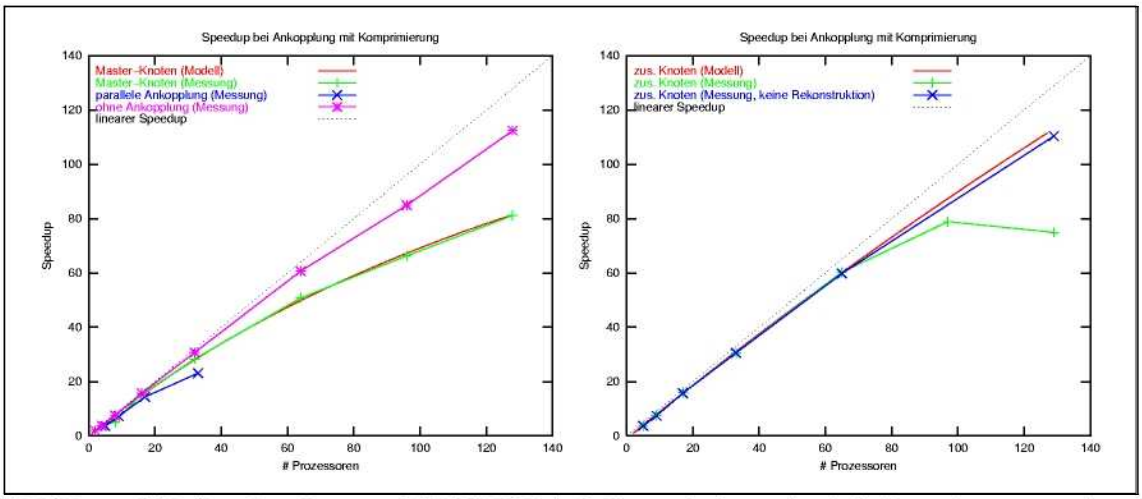


Abbildung 6.15: Speedup Trace auf CRAY T3E (mit Komprimierung): Die beiden Diagramme zeigen die Speedup-Kurven für verschiedene Kopplungsstrategien. Nur bei der Ankopplung über einen zusätzlichen Kommunikationsknoten ist ein Speedup zu erreichen, der dem ursprünglichen Speedup des Programms nahe kommt. Die Simulationsrechnung wird erst dann durch die Ankopplung wesentlich gestört, wenn die Visualisierungsseite die Daten nicht schnell genug rekonstruieren kann (rechtes Diagramm).

Die Ursache dafür ist, daß der Master-Knoten die Daten aller Prozessoren an die Visualisierung senden muß. Während dieser Zeit können die restlichen Rechenknoten nicht mit ihrer Arbeit fortfahren. Mit wachsender Prozessorzahl verringert sich die Zeitdauer eines Simulationsschritts, so daß sich der Anteil der von der Prozessorzahl unabhängigen Kommunikationszeit erhöht und damit den Speedup weiter verringert. Bei der Ankopplung über einen zusätzlichen Knoten übernimmt dieser die Aufgabe der Datenübertragung, so daß im Gegensatz zur Übertragung über den Master-Knoten die Rechenknoten mit ihrer Arbeit fortfahren können (siehe auch Abb. 6.16 auf der nächsten Seite). Wenn bei hohen Prozessorzahlen die Dauer eines Simulationsschritts immer geringer wird, reicht diese Zeit nicht aus, um die Übertragung der Daten zur Visualisierung und dort die Rekonstruktion der komprimierten Daten zu beenden. In diesem Fall muß der zusätzliche Knoten bei der nächsten Datenübertragung solange warten, bis daß die Visualisierung wieder für das Empfangen von Daten bereit ist. Dies führt in den nächsten Simulationsschritten dann auch zu Wartezeiten bei den Rechenknoten, so daß insgesamt die Speedup-Kurve abfällt. Dieser Effekt wird durch die beiden gemessenen Speedup-Kurven im rechten Diagramm der Abb. 6.15 verdeutlicht. Wie bei der nicht-komprimierten Übertragung konnte die parallele Ankopplung nur für kleine Prozessorzahlen ausgemessen werden. Die Messungen zeigen aber, daß die Speedup-Kurve noch stärker abfällt als bei der Ankopplung über den Master-Knoten. Dies liegt im wesentlichen an der Vielzahl von kleinen Datenpaketen, die bei dieser Strategie über verschiedene Verbindungen zur Visualisierung gelangen und dort jeweils einzeln behandelt werden müssen. Jeder Rechenknoten muß in jedem Schritt die aktuellen Komprimierungsparameter von der Visualisierung anfordern (*visctl*) und zwei Datenpakete (Byte, Double) sowie die benutzten Komprimierungsparameter zur Visualisierung zurücksenden. Da es bei der parallelen Ankopplung keine geordnete Reihenfolge der Abarbeitung auf der Visualisierungsseite gibt, kann es z.B. vorkommen, daß diese bereits mit der Verarbeitung der Datenpakete eines Rechenknotens beschäftigt ist und noch nicht alle Anfragen für Komprimierungsparameter bearbeitet hat. In diesem Fall warten diese Knoten auf die Parameter und können die Komprimierung der Daten dadurch erst später beginnen. Die Kommunikationszeit verlängert sich dadurch und der Speedup sinkt entsprechend. Bei den beiden anderen Kopplungsstrategien werden die einzelnen Datenpakete von dem Master-Knoten bzw. zusätzlichen Kommunikationsknoten vor der Übertragung zu einem Paket zusammengefaßt. Zusätzlich werden die Komprimierungsparameter nur von einem Knoten des Simulationsprogramms von bzw. zu der Visualisierung übertragen. Der Aufwand dafür ist entsprechend geringer.

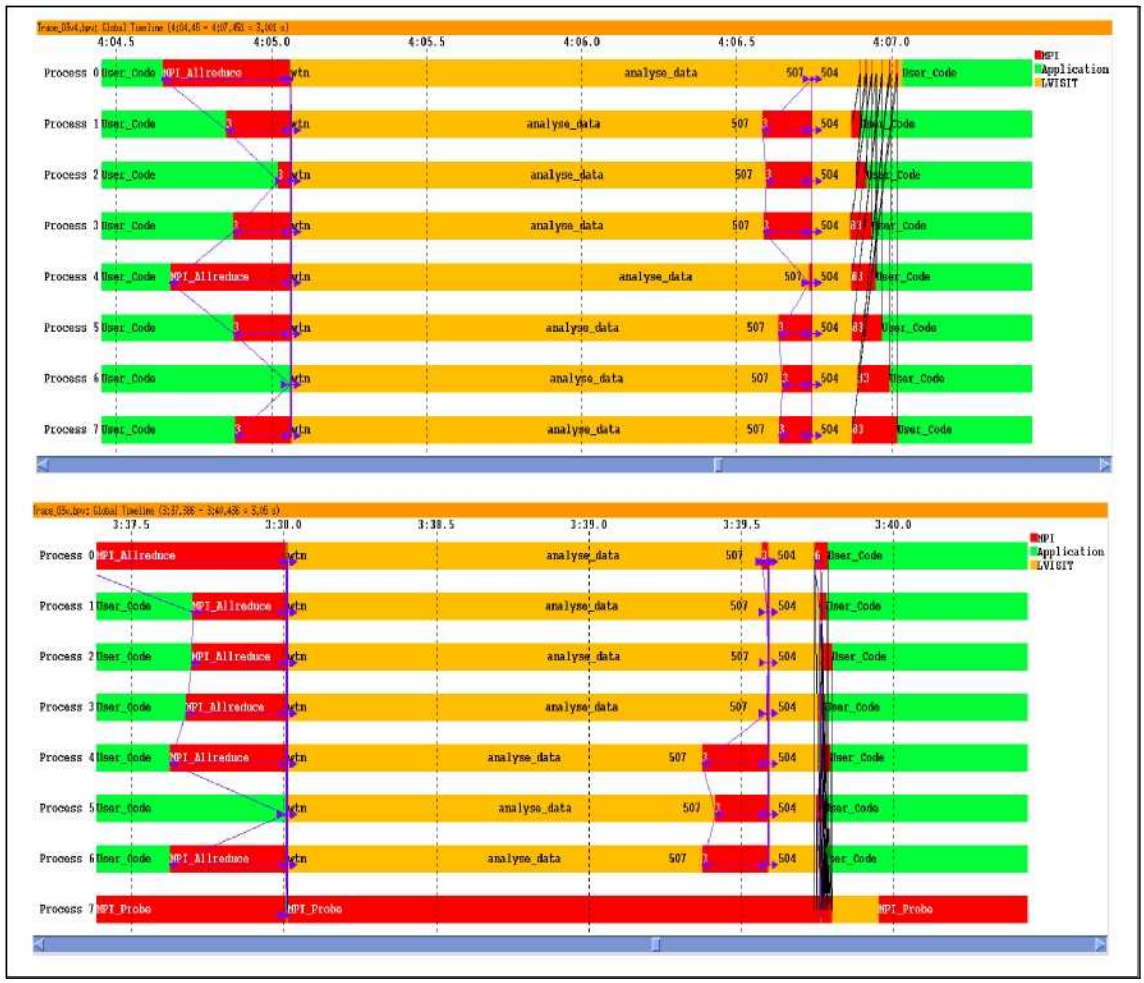


Abbildung 6.16: Kommunikationsablauf bei Ankopplung über Master-Knoten (oben) bzw. zusätzlichen Knoten (unten): Dieser mit Vampir [60] dargestellte Ablauf der Kommunikation innerhalb von Trace zeigt die Aktivitäten der acht Prozessoren während der Komprimierung und Übertragung der Daten zur Visualisierung. Die Zeit für die Komprimierung der lokalen Daten unterscheidet sich geringfügig auf den Prozessoren. Durch die ersten kollektiven MPI-Operationen (Abgleich der Minimal- und Maximalwerte der Histogramme) werden die Prozessoren wieder synchronisiert. Zum Abschluß der Komprimierung sendet jeder Knoten die Daten zum Master-Knoten bzw. zum zusätzlichen Kommunikationsknoten, der sie in geordneter Reihenfolge annimmt und dann als ein Paket an die Visualisierung weiterschickt.

6.4 Auswirkung der Komprimierung auf die Daten

Wie in den vorherigen Abschnitten gezeigt wurde, lassen sich durch die Komprimierung der Daten die Performance-Einbußen einer Simulationsrechnung in Folge der Ankopplung eines Visualisierungsprogramms erheblich reduzieren. Im Gegenzug müssen Fehler in den visualisierten Daten durch die verlustbehaftete Komprimierung hingenommen werden. In diesem Abschnitt wird untersucht, wie dieser Fehler von den beiden Parametern abhängt, welche die Qualität der rekonstruierten Daten beeinflussen. Es sind dies der Anteil der abgeschnittenen kleinen Koeffizienten und die Anzahl der bei der Quantisierung benutzten Bits zur Repräsentation eines Zahlenwerts. Als ein einfaches Maß für die Bewertung des durch diese beiden Verarbeitungsschritte entstandenen Fehlers wird hier die L_2-Norm der Differenz zwischen den Originaldaten und den aus den komprimierten Daten rekonstruierten Daten betrachtet.

Im diskreten Fall hat die L_2-Norm die folgende Form:

$$||f||_2 = \sqrt{\sum_{i=1}^{n} f_i^2} \tag{6.4}$$

Da die Wavelet-Transformation energieerhaltend ist und die für die diskrete Wavelet-Transformation benutzten Wavelets orthonormal zueinander sind, läßt sich die L_2-Norm in der gleichen Weise auch aus den Wavelet-Koeffizienten bestimmen:

$$||f||_2 = ||W(f)||_2 = \sqrt{\sum_{k,j} c_{j,k}^2} \tag{6.5}$$

Die L_2-Norm des Fehlers, der durch das Nullsetzen von Koeffizienten erzeugt wird, läßt sich also als Wurzel aus der Summe der Quadrate dieser Koeffizienten berechnen läßt. Abbildung 6.17 zeigt dazu das Ergebnis einer Rechnung mit dem von Trace gelieferten Beispieldatensatz. Man erkennt, daß durch das in den Messungen durchgeführte Abschneiden von 80% der Koeffizienten ein Fehler erzeugt wird, dessen L_2-Norm weniger als 0.1% der L_2-Norm des Datensatzes beträgt. Dies macht deutlich, daß bei den in der Messung verwendeten Daten durch das Abschneiden kleiner Koeffizienten eine hohe Komprimierungsrate erzielt werden kann, ohne dabei einen großen Fehler in den komprimierten Daten zu verursachen. Im oberen rechten Diagramm der Abbildung 6.18 auf der nächsten Seite ist dazu die Komprimierungsrate bei verschiedenen Anteilen von abgeschnittenen Koeffizienten gezeigt.

Bei dem Verfahren der Quantisierung werden in jedem Teilintervall die Originalwerte durch das arithmetische Mittel der in diesem Teilintervall liegenden Werte ersetzt. Als Fehlermaßstab wird daher die L_2-Norm der Abstände zwischen Originalwert und Mittelwert benutzt. Mit Hilfe der in

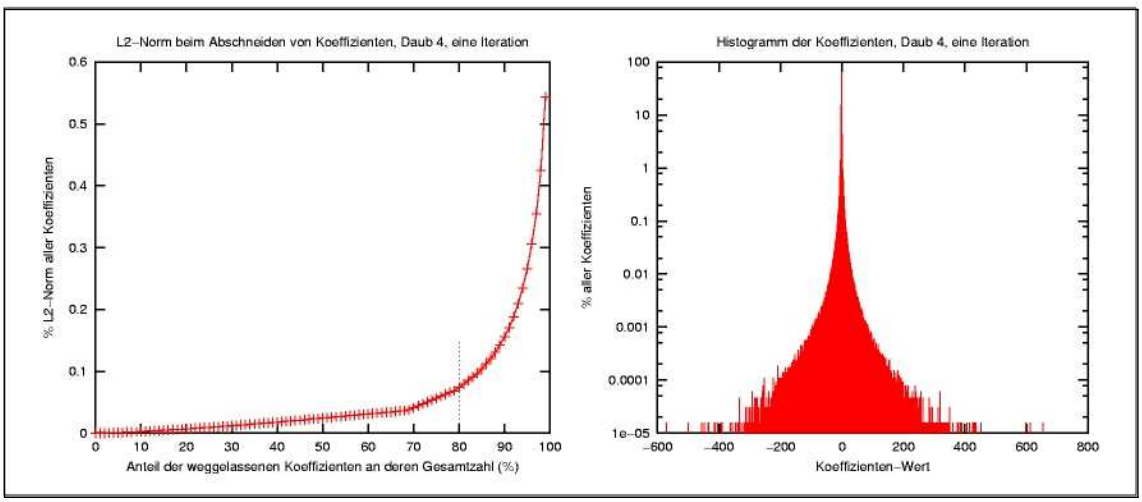


Abbildung 6.17: L_2-Norm der abgeschnittenen Koeffizienten: Das linke Diagramm zeigt die Entwicklung der L_2-Norm des Fehlers, der durch das Abschneiden von kleinen Koeffizienten entsteht. Aufgetragen ist der relative Fehler, also der Anteil an der L_2-Norm aller Koeffizienten. Sogar bei dem Abschneiden von 99% der Koeffizienten im ersten Frequenzband bleibt der Fehler unter einem Prozent. Im rechten Diagramm ist die Verteilung der Koeffizienten in einem Histogramm zu sehen, die zeigt, daß sich die Mehrzahl der Koeffizienten nahe bei Null befinden und daher beim Abschneiden auch nur einen kleinen Fehlerbeitrag liefern.

Abschnitt 3.2 auf Seite 35 eingeführten Bezeichnungen ergibt sich:

$$L_2(f_{\text{quant}}) = \sqrt{\sum_{i=0}^{2^q} \sum_{x \in [d_i, d_{i+1}]} (x_j - r_i)^2} = \sqrt{\sum_{i=0}^{2^q} \left(\sum_{x \in [d_i, d_{i+1}]} x^2 - n_i\, r_i^2 \right)} \tag{6.6}$$

Dabei ist q die Anzahl der Quantisierungsbits und n_i die Anzahl der Koeffizienten im Teilintervall i. Durch die zweite Umformung kann der Fehler direkt während der Rechnung bestimmt werden. Abbildung 6.18 zeigt den Verlauf der L_2-Norm des Fehlers als Prozentanteil der L_2-Norm aller Koeffizienten für verschiedene Quantisierungsstufen. Da die Anzahl der Teilintervalle mit 2^q exponentiell anwächst, nimmt der Fehler bei der Erhöhung der Anzahl der Quantisierungsbits exponentiell ab. Durch die für Performance und Komprimierungsrate günstige 8-Bit-Quantisierung wird bei den hier verwendeten Daten nur ein Fehler von ca. einem Prozent verursacht. Das rechte untere Diagramm der Abbildung 6.18 zeigt, daß die Komprimierungsrate mit steigender Anzahl der Quantisierungsbits wie zu erwarten abnimmt. Ursache für die starke Abnahme der Komprimierungsrate bei neun Quantisierungsbits ist, daß ab dieser Anzahl ein zweites Byte für die Speicherung der Intervallnummer benötigt wird.

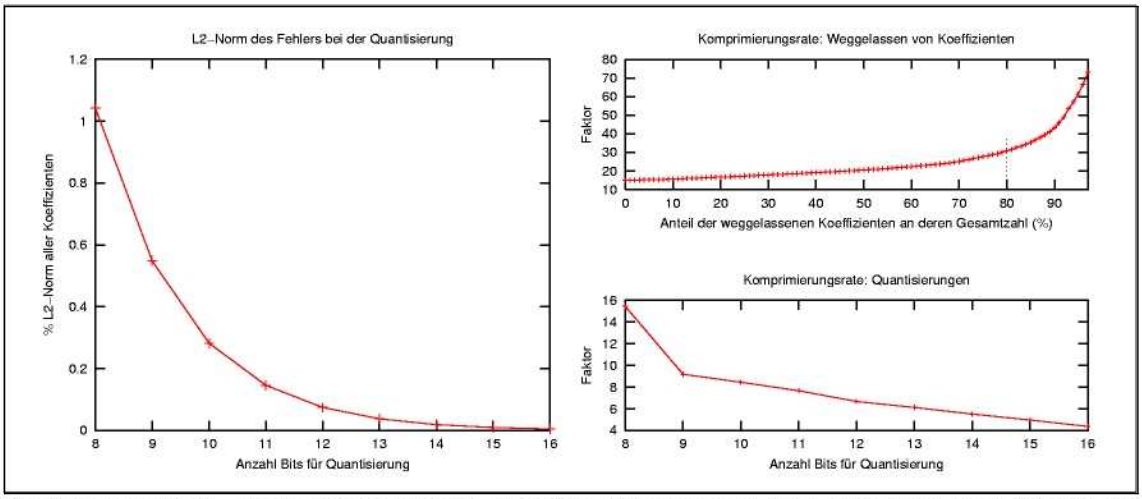
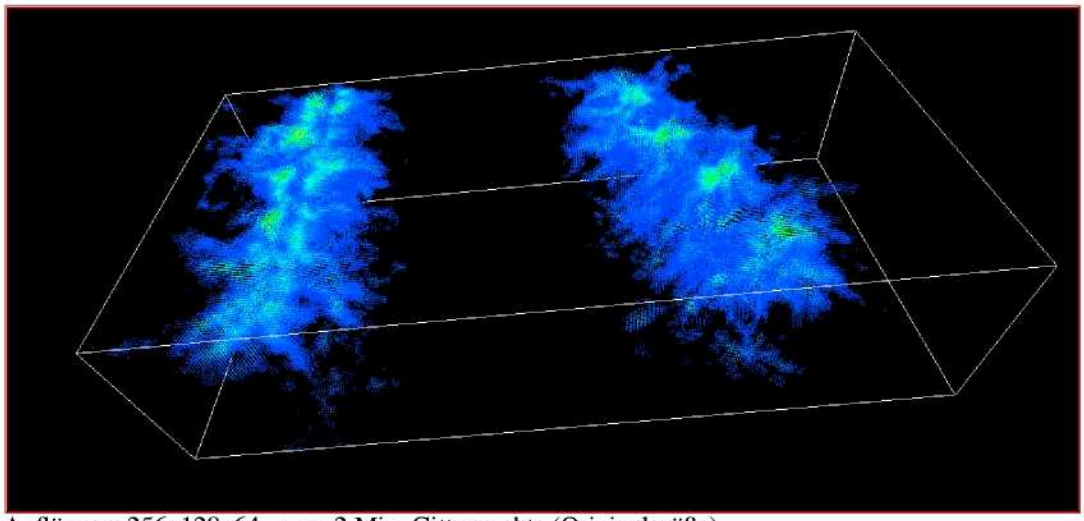


Abbildung 6.18: L_2-Norm des Fehlers bei der Quantisierung: Das linke Diagramm zeigt den Verlauf der L_2-Norm des durch die Quantisierung verursachten Fehlers als Anteil an der L_2-Norm aller Koeffizienten. Auf der linken Seite sind die Komprimierungsraten für die beiden in diesem Abschnitt untersuchten Verfahren aufgeführt. Um die Messung nicht zu verfälschen, wurden bei der Messung der Rate für die Quantisierung im vorhergehenden Verarbeitungsschritt keine Koeffizienten abgeschnitten.

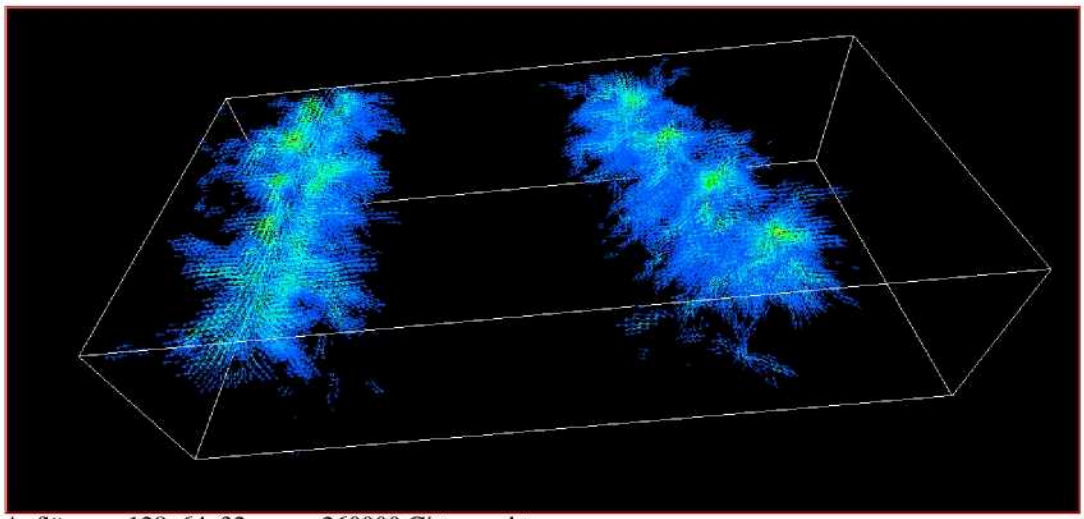
Diese Ergebnisse zeigen, daß mit den beiden hier betrachteten Hilfsmitteln eine effiziente und schnelle Komprimierung von Daten möglich ist, ohne dabei einen großen Fehler in der L_2-Norm zu verursachen. Daher ist gerade im Bereich des Computational Steering der Einsatz der in dieser Arbeit implementierten schnellen Wavelet-Transformation mit diesen beiden nachgeschalteten Komprimierungsschritten sinnvoll.

Beispiel für die Auswirkung der Rekonstruktion in verschiedenen Auflösungsstufen

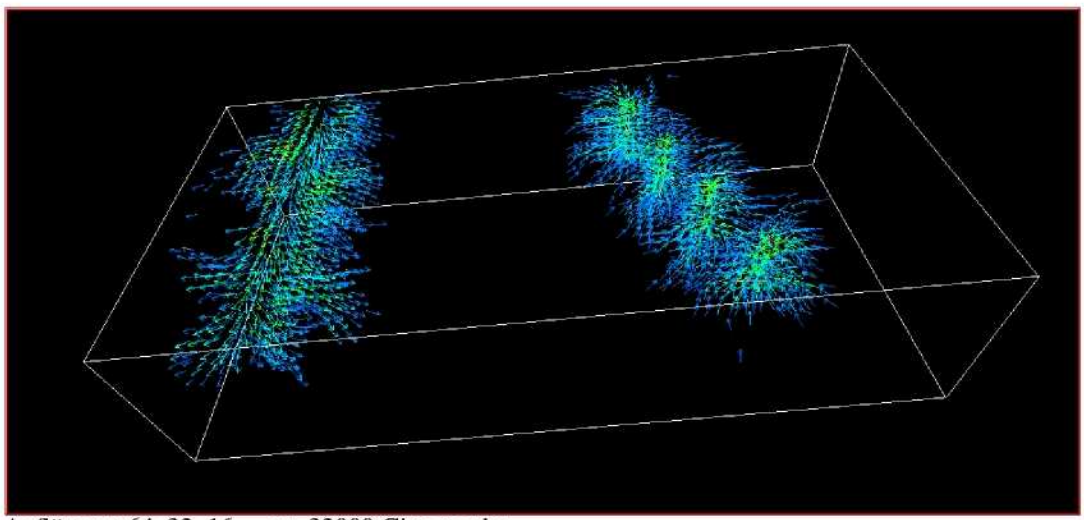
Abbildung 6.19 auf der nächsten Seite zeigt den bei diesen Messungen benutzten Beispieldatensatz in den verschiedenen Auflösungsstufen, die sich durch die Zerlegung der Daten in einzelne Frequenzbänder ergeben. Bei der Betrachtung mit der durch den Code-Generator erzeugten AVS/Express-Applikation kann der Benutzer interaktiv zwischen diesen verschiedenen Auflösungsstufen wechseln und zusätzlich die Anzahl der zu übertragenden Frequenzbänder begrenzen. Die aufwendige Verarbeitung der Visualisierungsdaten in Originalgröße, die insbesondere wegen des hohen Hauptspeicherbedarfs an die Leistungsgrenze des Notebooks stößt, kann dadurch vermieden werden.



Auflösung: 256x128x64 → ca. 2 Mio. Gitterpunkte (Originalgröße)



Auflösung: 128x64x32 → ca. 260000 Gitterpunkte



Auflösung: 64x32x16 → ca. 32000 Gitterpunkte

Abbildung 6.19: Beispieldatensatz nach Rekonstruktion in verschiedenen Auflösungsstufen: Die drei Bilder zeigen das Ergebnis der komprimierten Übertragung des Beispieldatensatzes, bei der zwei Iterationen des Pyramidenalgorithmus ausgeführt und damit drei Frequenzbänder erzeugt wurden. Nach der Übertragung werden die Frequenzbänder wieder so zurück transformiert, daß für jede Auflösungsstufe ein Bild entsteht. Bei der progressiven Übertragung der Daten werden die Frequenzbänder nacheinander übertragen, so daß die Bilder – beginnend mit der niedrigsten Auflösungsstufe – auch nach und nach in der Anzeige aktualisiert werden.

Kapitel 7

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, Strategien zu entwickeln, welche die Online-Visualisierung auch bei parallelen Simulationsrechnungen mit verteilter Datenhaltung ermöglichen. Insbesondere die Größe der von parallelen Simulationsprogrammen erzeugten Daten stellt dabei ein Problem dar, durch das eine direkte Übertragung zum Visualisierungsrechner nicht mehr möglich ist. Ein weiteres Problem ist die verteilte Datenhaltung, die bei der Ankopplung einer Visualisierung an das Simulationsprogramm berücksichtigt werden muß.

Zur Reduzierung der Datengröße und damit als Lösung für das erste oben aufgeführte Problem wurde in dieser Arbeit ein Komprimierungsverfahren auf Basis der schnellen Wavelet Transformation entwickelt, das zusammen mit weiteren Komprimierungsschritten wie dem Abschneiden kleiner Koeffizienten, der Quantisierung und der Kodierung bei der Online-Visualisierung nicht nur die Daten für den Transport verkleinert, sondern diese auch so in der Auflösung reduziert, daß eine schnelle, aber dennoch weitgehend unverfälschte Darstellung möglich ist.

Für die Lösung des zweiten Problems wurden in der Arbeit verschiedene Kopplungsstrategien vorgestellt, die zum einen die parallele Struktur des Simulationsprogramms berücksichtigen und zum anderen die Übertragung der auf den einzelnen Prozessoren verteilten Zwischenergebnisse zum Visualisierungsrechner ermöglichen. Dabei wurden mehrere mögliche Kopplungsstrategien behandelt: So können die Daten über einen ausgezeichneten Prozessor (Master-Prozessor) des Simulationsprogramms oder einen zusätzlichen nur für diese Aufgabe zuständigen Prozessor zur Visualisierung geführt oder parallel und damit gleichzeitig von allen Prozessoren übertragen werden. Bei diesen Strategien wurden insbesondere die Eigenschaften der schnellen Wavelet-Transformation ausgenutzt, die eine separate Transformation der lokal gespeicherten Daten zuläßt, so daß diese erst auf der Visualisierungsseite zusammengeführt werden müssen. Durch diese parallele Verarbeitung der Daten kann die für die Online-Visualisierung benötigte Zeit um ein Vielfaches reduziert werden. Weiterhin wurde bei der Entwicklung der Strategien ausgenutzt, daß die Daten durch die schnelle Wavelet-Transformation in Frequenzbänder verschiedener Auflösungsstufe aufgeteilt werden, was eine progressive Übertragung der Daten ermöglicht. Auf der Visualisierungsseite ergibt sich dadurch eine schrittweise Verfeinerung der angezeigten Daten.

Zur Bewertung der Leistungsfähigkeit der vorgestellten Verfahren wurden Modelle für deren Laufzeitverhalten entwickelt. Diese Modelle wurden auf parallele Programme verallgemeinert, um den Einfluß der oben beschriebenen Kopplungstrategien auf den Speedup vorhersagen zu können. Die wichtigste Vorhersage der Modelle ist, daß die Ankopplung über einen zusätzlichen Knoten über weite Parameterbereiche den anderen Kopplungsstrategien überlegen ist. Aufgrund der benutzten Algorithmen wird dabei insgesamt eine lineare Abhängigkeit der für die Reduzierung und Übertragung der Daten benötigten Zeit von deren Größe erwartet. Eine weitere, wenig überraschende Prognose ist, daß eine Komprimierung der Daten vor der Übertragung nur dann eine Zeitersparnis

bringt, wenn die durch eine Komprimierungsbandbreite ausgedrückte Geschwindigkeit der Komprimierung die Übertragungsbandbreite übertrifft.

Um die Modellvorhersagen qualitativ und quantitativ zu überprüfen, wurden die entwickelten Komprimierungs- und Kopplungsstrategien implementiert und in das parallele Simulationsprogramm Trace integriert. Dieses stellt den aufwendigsten Teil der hier vorgestellten Arbeiten dar, da keine auf Trace zugeschnittene Einzellösung, sondern ein allgemeiner einsetzbares Werkzeug entwickelt wurde.

Bei der Implementierung der Datenreduzierung und der für die verschiedenen Kopplungstrategien nötigen Funktionen konnte auf die im Forschungszentrum Jülich entwickelte Kommunikationsbibliothek VISIT zurückgegriffen werden. Darauf aufbauend wurde in vorliegenden Arbeit für die Datenreduzierung die Bibliothek LVISIT entwickelt, welche Funktionen für die Wavelet-Transformation, das Abschneiden von Koeffizienten, die Quantisierung der Daten und deren Kodierung und Komprimierung zur Verfügung stellt. Für die Bereitstellung einer an die Simulationsrechnung angepaßten Schnittstelle zur Ankopplung einer Online-Visualisierung wurde zudem der Code-Generator *visitcg* entwickelt, der auf der Simulationsseite Schnittstellenfunktionen in der jeweiligen Programmiersprache und für die zu übertragenden Daten generiert. Darüber hinaus stellt der Code-Generator auch für die Visualisierungsseite eine an die Kopplung angepaßte Schnittstelle zur Verfügung. Dazu wird für das in der wissenschaftlichen Visualisierung weit verbreitete Werkzeug AVS/Express neben den für die Ankopplung wichtigen Kommunikationsmakros auch eine komplette Applikation generiert, welche die Aufgabe der Übertragung und Anzeige der Daten übernimmt und die vom Anwender mit Hilfe des AVS-Editors sehr einfach interaktiv für seine Fragestellungen verfeinert werden kann.

Mit Hilfe der in dieser Arbeit entwickelten LVISIT-Bibliothek und des Code-Generators werden damit alle benötigen Hilfsmittel bereitgestellt, die es für den Anwender sehr einfach machen, ein Simulationsprogramm durch minimale Änderungen mit einer Visualisierung konsistent zu koppeln. Dabei sind einerseits – im Sinne des Computational Steerings – Parameter-Änderungen durch den Betrachter möglich und andererseits die in dieser Arbeit vorgestellte und für die speziellen Anforderungen des Computational Steerings angepaßten Datenreduzierungs- und Komprimierungs-Methoden sehr einfach anwendbar.

Anschließend wurden der Code-Generator und die Bibliothek für die Realisierung einer Ankopplung am Beispiel des parallelen Anwenderprogramms Trace eingesetzt. Mit Hilfe dieses Simulationsprogramms wurden die Modellgleichungen für den Speedup-Verlauf verifiziert. Dabei konnte gezeigt werden, daß durch den Einsatz des in dieser Arbeit entwickelten Komprimierungsverfahrens eine Online-Visualisierung der Ergebnisse ohne eine signifikante Beeinträchtigung der Rechnung möglich ist und sich zudem eine weit geringere Netzbelastung einstellt.

Die hier entwickelten Werkzeuge wurden so entworfen, daß sich Erweiterungen leicht integrieren lassen. Denkbar sind z.B. die Unterstützung anderer Kommunikationbibliotheken als MPI (PVM oder Threads) oder die Unterstützung weiterer Programmiersprachen auf der Simulationsseite sowie anderer Visualisierungssysteme (z.B. VTK). Die Komprimierungsverfahren wurden in dieser Arbeit vor allem auf Laufzeiteffizienz optimiert. Sinnvoll kann auch die Optimierung des Speicherplatzbedarfs sein. Ein Ansatzpunkt für Verbesserungen wäre auch, Koeffizienten von nicht benötigten Frequenzbändern nicht abzuspeichern. Die Komprimierungsverfahren der Bibliothek unterstützen uniforme Gitter und damit eine wichtige Klasse von Simulationsprogrammen. Für eine Erweiterung auf adaptive Gitteralgorithmen können die implementierten Verfahren genutzt werden.

Literaturverzeichnis

[1] ADVANCED VISUAL SYSTEMS INC.: *AVS Visualization Techniques*, 1998. Waltham.

[2] T. Eickermann und W. Frings: *VISIT - a Visualization Interface Toolkit, Version 1.0.* Technical Report, Forschungszentrum Jülich, Zentralinstitut für Angewandte Mathematik, 2000.

[3] R. W. Seidemann: *Untersuchungen zum Transport von gelösten Stoffen und Partikeln durch heterogene Porengrundwasserleiter.* Dissertation, Universität Bonn, 1997.

[4] O. Nitzsche, H. Vereecken und H. Hardelauf: *TRACE, a parallel reactive particle tracking code for soil-groundwater systems.* In: *Abstracts to the EGS XXVI General Assembly.* Nice, France, 2001.

[5] N. Attig, M. Lewerenz, G. Sutmann und R. Vogelsang: *DMMD - a Modular Molecular Dynamics Program for MPP Systems.* In: *Fifth European SGI/Cray MPP Workshop*, Bologna, 1999.

[6] M. Cox: *Large data management for interactive visualization design.* In SIGGRAPH '99 System Designs for Visualizing Large-Scale Scientific Data, course notes, 1999.

[7] H.-U. Schlageter: *Entwurf eines integrierten Systems zur Visualisierung von Ergebnissen numerischer Berechnungsverfahren für massiv parallele Rechnerarchitekturen.* Dissertation, Fakultät für Energietechnik der Universität Stuttgart, 2000.

[8] P. D. Heermann: *ASCI visualization: One Teraflop and beyond.* In: *NSF/DOE Workshop on Large Scale Visualization and Data Management.* Salt Lake City, Utah, 1999.

[9] W. E. Lorensen und H. E. Cline: *Marching cubes: A high resolution 3D surface construction algorithm.* In: *Proceedings of SIGGRAPH '87*, 163–169, 1987.

[10] B. Wilkinson und M. Allen: *Parallel Programming - techniques and applications using networked workstations and parallel computers.* Prentice Hall, 1999.

[11] L. Dagum und R. Menon: *OpenMP: An Industry-Standard API for Shared-Memory Programming.* IEEE Computational Science & Engineering, 5(1):46–55, 1998.

[12] B. Nichols, D. Buttlar und J. Proulx Farrell: *Pthreads Programming, a POSIX Standard for Better Multiprocessing.* O'Reilly, 1996.

[13] M. Snir, St. W. Otto, St. Huss-Lederman, D. W. Walker und J. Dongarra: *MPI The Complete Reference.* The MIT Press, Cambridge, Massachusetts, 1996.

[14] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek und V. Sunderam: *PVM 3 User's Guide and Reference Manual.* Technical Report, Oak Ridge National Laboratory, 1994.

[15] M. J. Flynn: *Very High-Speed Computing Systems.* In: *Proceedings of the IEEE*, Band 54/12, 1901– 1909, 1966.

[16] J.D. Mulder, J.J. van Wijk und R. van Liere: *A survey of computational steering environments.* Technical Report SEN-R9816, Centrum voor Wiskunde en Informatica, 1996.

[17] J. Vetter und K. Schwan: *Progress: A toolkit for interactive program steering.* In: *Proceedings of the 1995 International Conference on Parallel Processing*, 139–142, Oconomowoc, WI, 1995.

[18] J. Vetter und K. Schwan: *Models for Computational Steerin.* In: *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDS '96), Annapolis, Maryland*, 1996.

[19] J. A. Kohl: *CUMULVS.* Tutorial, Oak Ridge National Laboratory, September 2000.

[20] J. A. Kohl und P. M. Papadopulos: *Efficient and flexible fault tolerance and migration of scientific Simulations using CUMULVS.* In: *Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools*, Welches, Oregon, August 1998.

[21] S. Rathmayer: *On-line Visualisierung und interaktive Steuerung paralleler wissenschaftlich-technischer Anwendungen.* Vortrag, Sommerschule Parallele Algorithmen und Rechnerarchitekturen, Technische Universität München, März 1997.

[22] R. van Liere, J.D. Mulder und J.J. van Wijk: *Computational steering.* Future Generation Computer Systems, 12(5):441–450, 1997.

[23] S. Parker: *The SCIRun Problem Solving Environment and Computational Steering Software System.* PhD thesis, University of Utah, 1999.

[24] C. Johnson, S. Parker und D. Weinstein: *Large-Scale Computational Science Applications Using the SCIRun Problem Solving Environment.* Supercomputer 2000, Heidelberg, 2000.

[25] J. Vroom: *AVS/Express: A new Visual Programming Paradigm.* Advance Visual Systems Inc., 1998.

[26] A. Wierse und U. Lang: *COVISE.* WWW: http://www.hlrs.de/organization/vis/covise.

[27] A. Merzky: *Visuelle Kontrolle und Steuerung von verteilten Anwendungen auf Höchstleistungsrechnern.* In: *Tagungsbroschüre zum Workshop "Wissenschaftliche Anwendungen auf Höchstleistungsrechnern".* RRZN/Universität Hannover, September 1999.

[28] B. Henderson: *3D Visualization From Molecules to Immersion.* Exhibitor Tech Talk at SIGGRAPH 2002, San Antonio, Texas, July 2002.

[29] S. Rathmayer: *Visualization and Computational Steering in Heterogeneous Computing Environments.* In: *Proceedings of Euro-Par, München*, 2000.

[30] C. Michaels und M. Bailey: *VisWiz: A Java Applet for Interactive 3D Scientific Visualization on the Web.* Proceedings IEEE Visualization 97, 261–267, 1997.

[31] S. Olbrich und H. Pralle: *Verteilte Visualisierung von ingenieur-wissenschaftlichen Simulationsergebnissen.* In: *ICCES-Kolloquium, Geometrische Modellierung, Visualisierung, Software, Hannover*, 2001.

[32] T. Eickermann Hrsg.: *Gigabit Testbed West - Abschlußbericht des DFN-Projektes.* Technical Report, Forschungszentrum Jülich, Zentralinstitut für Angewandte Mathematik, 2000.

[33] M. Rabbani und R. Joshi: *An overview of the JPEG 2000 still image compression standard.* Signal Processing: Image Communication, Elsevier, 17:3–48, 2002.

[34] A. Klingert, C. Knörzer und K. Saar (Hrsg.): *Multimedia Datenformate.* Universität Karlsruhe, Institut für Betriebs- und Dialogsysteme, 1994/95.

[35] M. M. Gutzmann und R. Scholz: *Wavelets: Grundlagen und Anwendungen in der Bildkompression.* Band 3 11, Friedrich-Schiller-Universität Jena, 1997.

[36] Parker, J. R.: *Algorithms for Image Processing and Computer Vision.* Wiley Computer Publishing, 1996.

[37] T. Lehmann, W. Oberschelp, E. Pelikan und R. Repges: *Bildverarbeitung für die Medizin.* Springer-Verlag Berlin, 1997.

[38] M. Weis: *Verwendung von Texturparametern bei der Klassifizierung von hochaufgelösten Satellitenbildern.* Diplomarbeit, Universität Hannover, 2000.

[39] S. G. Mallat: *A theory for multiresolution signal decomposition: The wavelet representation.* IEEE Transactions on pattern analysis and machine intelligence, 11:674–693, 1989.

[40] A. Trott, R. Moorhead und J. McGinley: *Wavelets Applied to Lossless Compression and Progressive Transmission of Floating Point Data in 3-D Curvilinear Grids.* In: *IEEE Visualization '96*, 385–388, 1996.

[41] O. Møller Nielsen und M. Hegland: *A Scalable Parallel 2D Wavelet Transform Algorithm.* Technical Report TR-CS-97-21, Canberra 0200 ACT, Australia, 1997.

[42] O. Møller Nielsen und M. Hegland: *Parallel Performance of Fast Wavelet Transforms.* International Journal of High Speed Computing, 11(1):55–74, 2000.

[43] J. Max: *Quantizing for Minimum Distortion.* IRE Transactions on Information Theory, IT-6(1):7–12, März 1960.

[44] P. C. Cosman, R. M. Gray und M. Vetterli: *Vector Quantization of Image Subbands: A Survey.* IEEE Transactions on Image Processing, 5(2):202–225, 1996.

[45] R. Westermann: *A Multiresolution Framework for Volume Rendering.* In: *1994 Symposium on Volume Visualization, Washington, D.C.*, 51–58, 1994.

[46] T. Klein: *Compression Domain Volume Rendering.* Hauptseminar Visualisierung, Universität Stuttgart, 2001.

[47] A. L. Semon: *The Effects of Cascading Popular Text Compression Techniques.* Diplomarbeit, East Stroudsburg University, Pennsylvania, 1993.

[48] D. A. Huffman: *A Method for the Construction of Minimum-Redundancy Codes.* Proceedings of the IRE, 40(9):1098–1101, September 1952.

[49] J. Ziv und A. Lempel: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23(3):337–349, 1977.

[50] J. M. Favre: *Accelerating the AVS/Express Multi-block Visualization Macros.* In: *Proceedings of IEEE Visualization, San Francisco*, 1999.

[51] Z. Zhu, R. Machiraju, B. Fry und R. Moorhead: *Wavelet-based Multiresolution Representation of Computational Field Simulation Datasets.* In: *Proceedings of IEEE Visualization '97, Phoenix*, 151–158, 1997.

[52] G. Amdahl: *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities.* In: *AFIPS Conference Proceedings*, 483–485, 1967.

[53] S. Olbrich, u.a.: *Bericht zum DFN-Projekt Tele-Immersion*, 2001.
WWW: http://www.rvs.uni-hannover.de/projekte/tele-immersion/.

[54] W. H. Press, S. A. Teukolsky, W. T. Vetterling und B. P. Flannery: *Numerical recipes in C: The art of scientific computing.* Cambrigde University Press, 1992.

[55] M. F. Oberhumer: *LZO – a real-time data compression library*, 2000.
WWW: http://www.oberhumer.com/opensource/lzo/.

[56] *Rechnersysteme des Zentralinstituts für Angewandte Mathematik.* Forschungszentrum Jülich. WWW: http://www.fz-juelich.de/zam/CompServ/services/config.html.

[57] J. Brooks: *Single PE Optimization Techniques for the CRAY T3E System.* Technical Report, Cray Research, 1996.

[58] E. Anderson, J. Brooks und T. Hewitt: *The Benchmarker's Guide to Single-processor Optimization for CRAY T3E Systems.* Technical Report, Cray Research, 1997.

[59] H. Vereecken, G. Lindenmayr, O. Neuendorf, U. Döring und R. Seidemann: *TRACE, A mathematical model for reactive transport in 3D variable saturated porous media.* Technical Report, Forschungszentrum Jülich, 1994.

[60] Pallas GmbH: *Vampir: Visualization and Analysis of MPI Programs.*
http://www.pallas.de/pages/vampir.htm.

Danksagung

Die vorliegende Arbeit wurde als Diplomarbeit am Lehrgebiet für Technische Informatik I an der FernUniversität Hagen erstellt. Ich möchte dem Inhaber des Lehrgebiets Prof. Dr. W. Schiffmann herzlich für sein Interesse an der Arbeit und die mir gewährte Unterstützung danken. Herrn Prof. Dr. F. Hoßfeld, Leiter des Zentralinstituts für Angewandte Mathematik im Forschungzentrum Jülich danke ich für die Übernahme des Zweitgutachtens und für die technischen Möglichkeiten, die ich im seinem Institut für diese Arbeit nutzen durfte. Herrn Dr. R. Esser danke ich dafür, die Arbeit thematisch in ein Aufgabengebiet seiner Abteilung eingliedern zu dürfen.

Dr. Thomas Eickermann danke ich für die ausgezeichnete Betreuung, fruchtbare Diskussionen und wertvollen Anregungen.

Bei Jörg Striegnitz, Dr. Godehard Sutmann und Volker Sander möchte ich mich für die konstruktiven Diskussionen bedanken. Herrn Dr. Reiner Vogelsang danke ich für seine Hilfstellung bei den Performance-Messungen auf der CRAY-T3E.

Beate Schäfer, Dr. Norbert Attig und insbesondere Dr. Rudolf Theisen danke ich für das unermüdliche Korrekturlesen der Arbeit. Den Techno-Mathematik-Diplomanden des ZAMs danke ich für die motivierende Unterstützung während meiner Diplomarbeit.

Mein ganz besonderen Dank gilt meiner Frau Ulrike, die mich mit viel Verständnis und Geduld durch die Zeit der Diplomarbeit begleitet hat.

Forschungszentrum Jülich
in der Helmholtz-Gemeinschaft

Jül-4021
Dezember 2002
ISSN 0944-2952