H L R S

Institut für
Höchstleistungsrechnen

FORSCHUNGS – UND ENTWICKLUNGSBERICHTE

DIE BERECHNUNG VON WIEDEREINTRITTSPHÄNOMENEN AUF HIERARCHISCHEN SUPERCOMPUTERN MIT EINEM EFFIZIENTEN PARALLELEN MULTIBLOCKVERFAHREN

Thomas Bönisch

ISSN 0941 - 4665 Februar 2007 HLRS-04

Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. M. Resch
Nobelstrasse 19 - 70569 Stuttgart
Institut für Höchstleistungsrechnen

DIE BERECHNUNG VON WIEDEREINTRITTSPHÄNOMENEN AUF HIERARCHISCHEN SUPERCOMPUTERN MIT EINEM EFFIZIENTEN PARALLELEN MULTIBLOCKVERFAHREN

von der Fakultät Maschinenbau der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

Thomas Bönisch
aus Welzheim

Hauptberichter:	Prof Dr.- Ing. Michael Resch
MItberichter:	Prof. Dr. rer. nat. habil. Ulrich Maas
Tag der Einreichung:	17. Oktober 2005
Tag der mündlichen Prüfung:	28. April 2006

ISSN 0941 - 4665 Februar 2007 HLRS-04

Zusammenfassung

Wie in vielen Ingenieurwissenschaften sind auch in der Weltraumforschung Computersimulationen ein wichtiger Bestandteil der Forschung geworden. Auch bei der Entwicklung von Raumfahrzeugen spielt die Simulationsrechnung für die Systemauslegung eine wichtige Rolle. Gerade hier ist die Simulationsrechnung besonders wichtig, da Experimente und Messungen mit enormem Aufwand und extremen Kosten verbunden sind. Allerdings ist auch die Simulation von Raumfahrzeugen, hier speziell die Simulation des Wiedereintritts von Orbitern, erheblich aufwändiger als die Simulation eines "normalen" Flugzeugs, da neben den aerodynamischen Effekten auch chemische Reaktionen auftreten und in der Simulation berücksichtigt werden müssen. Für die Berechnung solcher Wiedereintrittsströmungen wurde am Institut für Raumfahrtsysteme der Universität Stuttgart das Programmpaket URANUS entwickelt. Dieses Programm besaß allerdings den Schwachpunkt, dass bisher nur sogenannte C-Netze zur Berechnung eingesetzt werden konnten, mit denen sich komplexere Raumgleiterkonfigurationen, wenn überhaupt, nur sehr schwer vernetzen lassen. Da die Berechnung von Wiedereintrittsströmungen auch erhebliche Anforderungen an Rechenleistung und Speicherbedarf besitzt, soll das neue Programm auch auf modernsten Supercomputerplattformen ohne Leistungsverlust einsetzbar sein.

Um diese Schwachpunkte zu beheben wurde das Programm zur Verwendung von sogenannten Multiblocknetzen erweitert. Dies bedingt allerdings eine völlige Überarbeitung des vorhandenen Simulationsprogramms und zwar sowohl hinsichtlich der verwendeten Datenstrukturen als auch bezüglich des Programmablaufs. Dazu wurden die Eigenschaften der Multiblocknetze genau untersucht und daraus die notwendigen Änderungen des Programms spezifiziert. Zur Integration der Multiblocknetze wurde eine neue Datenstruktur entwickelt, die, soweit möglich, bereits zukünftige Erweiterungen und Verbesserungen zum Beispiel in Richtung Mehrgitterverfahren ermöglicht und berücksichtigt. Die vorhandene Programmstruktur wurde in großen Teilen umgestellt und vor allem die Randbehandlung wurde so erweitert, dass die Randbedingungen unabhängig von der Lage des Blocks im Netz auf all seinen Seiten angewendet werden können. Dies erforderte eine allgemeine Formulierung der Randbedingungen. Für den Einsatz auf einer möglichst breiten Palette moderner Supercomputerplattformen wurde

das Multiblockprogramm von vorneherein auf die Verwendung massiv paralleler Systeme ausgelegt. Aber auch eine bereits vorhandene Optimierung für Vektorsysteme wurde weitergeführt. Durch die verschiedenen Größen der in Multiblocknetzen auftretenden Netzblöcke wurde hierbei eine komplexe Lastverteilungsstrategie notwendig. Hierzu wurden Algorithmen entwickelt, die Blöcke je nach Anforderung zerlegen können. Welcher Block welchem Prozess zur Berechnung zugeteilt wird, entscheiden Partitionierungsalgorithmen, die als Tool bereits verfügbar waren und über eine Schnittstelle in das Programm integriert worden sind. Die Schnittstelle dazu wurde so gestaltet, dass hier jederzeit neue Algorithmen einfach integriert werden können.

In einem weiteren Teil der Arbeit werden Technologien vorgestellt, die für URANUS entwickelt wurden, um das Strömungssimulationsprogramm effizient in einer Metacomputingumgebung einsetzen zu können.

Mit Hilfe des neuen parallelen Multiblock URANUS Verfahrens wurden bereits wichtige Simulationsergebnisse für den Wiedereintritt von modernen Raumgleitern erzielt, die ohne dieses Werkzeug nicht möglich gewesen wären.

Abstract

Simulation technology has become a important part of research within the engineering disciplines. This is also true for space science, where simulation plays a very important role for example in the development and design of space vehicles. The reason is, that experiments and measurements in this area result in an enormous effort in work as well as in costs. But, the simulation of space crafts, especially during the reentry phase is quite more complicated than the simulation of an ordinary plane as not only the aerodynamic effects but also chemical reactions occuring during the reentry and radiation have to be taken into account. For the calculation of these kind of flows, a special simulation program URANUS is developed at the institute for space systems at the University Stuttgart. In this activity, the URANUS program has been enabled to use so called multiblock meshes which allow for an easier meshing and the meshing of complex topologioes compared to the C-meshes used before. In additon, special technologies have been developed to enable the URANUS program for an efficient use in a metacomputing environment.

The introduction of multiblock meshes into the URANUS program results in a complete revision of the existing program concerning the workflow as well as the data structures. The new data structures reflects the requirements of the multiblock meshes as well as requirements for future extensions in the direction of multigrid methods and adaptive refinements. Therefore, these planned extensions should be possible with a limited effort as the data structures are already prepared. The existing program structure has also been changed and adapted. Especially, the handling of the boundary conditions has been made possible on all sides of a block by developing a general formulation. For the use of a broad spectrum of modern supercomputer architectures, the multiblock program is targeting massively parallel platfroms. But, an already existing optimization for vector parallel systems has been continued, too. Due to the different mesh block sizes possible in multiblock meshes, a new complex load balancing strategy has become necessary. For this, algorithms have been developed which devide too large blocks into smaller pieces. The distribution of the mesh blocks to the processors is decided by load balancing algorithms which have been already available as tools. For these tools, interfaces have been integrated into the program. These

interfaces have been made flexible allowing an easy integration of future load balancing tools.

The provided results show that URANUS is now able to provide good simulation results even for complex topologies on current supercomputer architectures. The parallel efficiency of the program is in the range of 82 to 97%, depending on the problem size and the number of used processors. Therefore, a scaling of the program into thousands of processors should be possible.

The program is applicable on nearly all available parallel platforms. Calculations have been performed on massively parallel systems as well as on PC-Clusters and vector parallel systems.

The use of this flow solver in a metacomputing environment has shown, that even CFD-programs can benefit from such an environment even as they are not predestined for this due to the strong coupling. The introduced changes like the less communicating solver, the removal of collective communication and the message pipelining decrease the influence of the weak external network connectivity. Especially, the influence of the high latency which occurs from the large distance between the available supercomputer systems is decreased. Calculations on a world wide distributed metacomputer have shown the usability of the URANUS program in such an environment with more than 1000 CPU's. However, as long as one single, most powerful system is available and can be used, metacomputing should not be taken into account.

The new parallel multiblock URANUS program has already been used to calculate results for the reentry of modern space orbiters into the earths atmosphere. This caculations would not have been possible without this newly developed tool.

Danksagung

Mein Dank gilt zuallererst Herrn Professor Michael Resch vom Höchstleistungrechenzentrum Stuttgart (HLRS) für die Übernahme des Hauptberichts und vor allem für seine Betreuung und Unterstützung dieser Arbeit, erst als Kollege, später als Vorgesetzter und Direktor des HLRS. Besonders danken möchte ich auch Herrn Professor Ulrich Maas vom Institut für Technische Thermodynamik der Universität Karlsruhe für die Übernahme des Zweitberichts.

Mein Dank gilt ebenso Herrn Professor Roland Rühle der mit dem HLRS ein Einrichtung geschaffen hat, die mit ihren internationalen Beziehungen, ihren Projekten und Ihrer Ausstattung diese Arbeit erst ermöglicht hat.

Danken möchte ich auch Herrn Dr. Alfred Geiger, der den Einstieg in die Promotion möglich gemacht hat.

Diese Arbeit entstand im Rahmen des SFB 259 "Hochtemperaturprobleme rückkehrfähiger Raumtransportsysteme" in enger Zusammenarbeit mit dem Institut für Raumfahrtsysteme. Mein Dank gilt hier insbesondere Herrn Dr. Markus Fertig, Herrn Dr. Farid Infed und Herrn Dr. Ferdinand Olawsky sowie allen Kollegen aus der Arbeitsgruppe Aerothermodynamik und deren ehemaligen Leiter, Herrn Dr. Hans-Heiner Frühauf, für die wissenschaftlichen Diskussionen. Die offene und vertrauenvolle Zusammenarbeit der verschiedenen Disziplinen in dieser Kooperation war und ist die Grundlage für eine gute wissenschaftliche Arbeit, besonders im Bereich der Simulation auf modernsten Höchstleistungsrechnern.

Danken möchte ich auch allen Kollegen des HLRS für die gute und offene Zusammenarbeit, die vielen guten Gespräche über Wissenschaft, Technik und den ganzen Rest. Den Kollegen aus der Betriebsgruppe darüberhinaus herzlichen Dank für das Verständnis und ihren Einsatz zu allen passenden und unpassenden Zeiten ohne den die Arbeiten im Bereich Metacomputing nicht möglich gewesen wären.

Hierzu gilt mein Dank auch den Kollegen vom Pittsburgh Supercomputing Center (PSC), dem Manchester Computing Center und den verschiedenen Netzwerkadministratoren auf den Wegen zwischen den Zentren für Ihre Unterstützung.

Meiner Familie und meinen Freunden ganz herzlichen Dank für die so wich-

tige mentale Unterstützung und Begleitung der Arbeit, insbesondere meinen Eltern auch für die tatkräftige Mitarbeit beim Korrekturlesen "beyond the spell checker".

Stuttgart, im Januar 2007 Thomas Bönisch

Inhaltsverzeichnis

1 Einleitung **15**
1.1 Wiedereintrittsprobleme 16
1.2 Simulation von Wiedereintrittsströmungen 17
1.3 Anforderungen an eine detaillierte Simulation 18

2 Stand der Technik **20**
2.1 Das Simulationspaket URANUS 20
2.2 Lösungsansätze für die Vernetzungsproblematik 21
2.2.1 Unstrukturierte Gitter 21
2.2.2 Chimera-Technik 24
2.2.3 Multiblockverfahren 24
2.3 Ansätze zur Erhöhung der Leistungsfähigkeit 25
2.3.1 Parallelisierung 25
2.3.2 Die Bibliothek CLIC 26
2.3.3 Die Bibliothek LOCO 26
2.3.4 Mehrgitterverfahren 26
2.3.5 Adaptive Netzverfeinerung 27
2.4 Programmiermodelle 28

3 Multiblockverfahren für Parallelrechner **30**
3.1 Ansatz 32
3.2 Konzeption 32
3.2.1 Eigenschaften von Multiblocknetzen 32
3.2.2 Bedeutung der Eigenschaften für das parallele Programm . 36
3.3 Programmentwurf 37
3.3.1 Datenstrukturen für die Blockverarbeitung 38
3.3.2 Datenstrukturen für die Randbehandlung 41
3.3.3 Behandlung physikalischer Ränder 42
3.3.4 Innere Ränder und Kommunikation 43
3.3.5 Organisation der Blockstruktur auf einem Parallelrechner . 44
3.3.6 Lastverteilung 46
3.3.7 Darstellung des Programmablaufs 52
3.4 Automatisches Multilevelverfahren 54

3.4.1 Generierung der Netze 54
3.4.2 Behandlung der Netzgrenzen und paralleles Verfahren . . . 55
3.4.3 Umschaltung zwischen den Netzleveln 56

4 Praktische Untersuchungen 57
4.1 Verwendete Hardware 57
4.1.1 Systeme mit verteiltem Hauptspeicher 58
4.1.2 Systeme mit gemeinsamem Hauptspeicher 59
4.1.3 Heterogene Systeme 61
4.2 Verwendete Testfälle 65
4.2.1 Colibri Kapsel 65
4.2.2 X38 - Prototyp für das CRV 67
4.3 Ergebnisse 68
4.3.1 Messmethode 69
4.3.2 Validierung des Verfahrens 69
4.3.3 Lastverteilung 70
4.3.4 Leistung 73
4.3.5 Multilevelverfahren 88
4.3.6 X38 92
4.4 Fazit 92

5 Konzeption für das Metacomputing 94
5.1 Allgemeine Einführung in das Metacomputing 95
5.2 Beschreibung der Kopplungssoftware PACX-MPI 95
5.2.1 Realisierung von PACX-MPI 96
5.3 Beschreibung der Testumgebungen 97
5.4 Erste Testergebnisse 99
5.5 Anpassung des parallelen Strömungslösers 105
5.5.1 Einschränkung der Kommunikation im Löser 105
5.5.2 Einfluß der Netzaufteilung im Metacomputing 114
5.5.3 Elimination der kollektiven Kommunikation 114
5.5.4 Message Pipelining 118
5.5.5 Weitere Ergebnisse 123
5.5.6 Änderungen im Pre- und Postprocessing 127
5.6 Diskussion und Folgerungen 128
5.7 Neue Ergebnisse 130
5.8 Fazit 131

6 Zusammenfassung und Ausblick 132
6.1 Ausblick 133

Tabellenverzeichnis

4.1 Leistungskennzahlen Prozessoren 63
4.2 Leistungskennzahlen Knoten 63
4.3 Durchsatz Netzwerk 64
4.4 Latenz Netzwerk 64
4.5 Vergleich der Rechenkosten 80
4.6 Scaleup auf der Cray T3E 81
4.7 Scaleup auf dem Cray Strider 0 82
4.8 Scaleup auf der NEC SX-6 82
4.9 Verlauf der Blockauf- und -umverteilung für den beschriebenen Fall 84

5.1 Leistungsfähigkeit der Verbindungsnetzwerke 100
5.2 Ergebnisse der ersten Messungen mit URANUS auf Metacomputern 101
5.3 Speedup von Standard URANUS auf einem Metacomputer 101
5.4 Laufzeitdifferenzen gemessen mit URANUS auf Metacomputern . 102
5.5 Kommunikationszeiten für Startup und Datenrücktransport im Metacomputing 103
5.6 Verteilung des Gitters auf die Prozesse 104
5.7 Bandbreite und Latenz der internen Netzwerkverbindungen auf der IBM RS/6000 SP 109
5.8 Ergebnisse von URANUS auf einem Metacomputer mit einem schwächer koppelnden Löser 111
5.9 Laufzeitdifferenzen mit neuem Löser 111
5.10 Ergebnisse von URANUS mit einem schwächer koppelnden Löser und verbesserter PACX-MPI Bibliothek 112
5.11 Laufzeitdifferenzen mit neuem Löser und neuer PACX-MPI Bibliothek 112
5.12 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing 113
5.13 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation 117
5.14 Abschätzung für die Laufzeit des Iterationsteils von URANUS bei einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation 118

5.15 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining . . . 120
5.16 Abschätzung für die Laufzeit des Iterationsteils von URANUS bei einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation mit Message-Pipelining . . . 121
5.17 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining auf der 300 MHz Partition mit 10 und 20 Iterationen . . 122
5.18 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining, mit speziellem Messverfahren im Rechenteil . . . 122
5.19 Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation und mit Message Pipelining mit unterschiedlicher Prozessorzahl . 123
5.20 Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, einmal ohne und einmal mit Message Pipelining, spezielle Messung des Iterationsteils . . . 124
5.21 Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, ohne Message Pipelining, mit unterschiedlicher Zellverteilung in den verwendeten Netzen . . . 125
5.22 Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, mit Message Pipelining auf einem anders aufgeteilten Netz mit 2x256 Prozessoren . . . 126
5.23 Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining, gemessen an unterschiedlichen Tagen . . . 129
5.24 Ergebnisse von Multiblock URANUS mit einem 1,536 Millionen Zellen Netz im Metacomputing . . . 130
5.25 Ergebnisse von Multiblock URANUS mit einem 5,12 Millionen Zellen Netz im Metacomputing . . . 131

Abbildungsverzeichnis

2.1 Strukturiertes Gitter für die X-38 22
2.2 Unstrukturiertes Gitter 23

3.1 C-Netz um die Nase eines Wiedereintrittskörpers 31
3.2 Indizierung in einem Multiblocknetz 34
3.3 Grund für die Seitenteilnumerierung 42
3.4 Netz-Kommunikationsvergleich 46
3.5 Kommunikationsvergleich 47
3.6 Vergleich der Blockschnittstellen nach Aufteilung 48

4.1 Multiblocknetz um den Colibrikörper 66
4.2 Fotomontage der X38 während eines Wiedereintritts 67
4.3 Oberflächennetz des Netzes zur Simulation der Strömung um die X38 68
4.4 Netz um die X-38, durch Zellverschmelzung von jeweils acht Zellen vergröbert 70
4.5 Netz um die X-38, große Blöcke durch URANUS zur Vermeidung von Lastungleichgewicht zerlegt 71
4.6 Netz um die X-38, nach Lastverteilung 72
4.7 Speedup des Multiblock URANUS Verfahrens auf der Cray T3E in Stuttgart 74
4.8 Speedup des Multiblock URANUS Verfahrens auf dem Cray Strider mit unterschiedlichen Problemgrößen 75
4.9 Speedup des Multiblock URANUS Verfahrens auf dem Cray Strider 0 mit unterschiedlichen Prozesszahlen pro Knoten 76
4.10 Speedup des Multiblock URANUS Verfahrens im Vergleich AMD Opteron, IA-64 77
4.11 Speedup des Multiblock URANUS Verfahrens auf der NEC-SX6 . 78
4.12 Laufzeitvergleich des Programms auf Strider, IA-64 und NEC SX-6 79
4.13 Darstellung eines kompletten Programmlaufs mit Vampir 83
4.14 Darstellung einer Iteration 86
4.15 Kommunikationsvolumen in einer Iteration 87
4.16 Ablauf einer Iteration bei einem Lauf auf 6 Prozessoren 89

4.17 Konvergenz einer Simulationsrechnung mit dem Prototyp des Multilevelverfahrens 90
4.18 Ergebnis einer Simulationsrechnung auf der Cray T3E mit dem Idealgasverfahren bei Mach 6 und 40 Grad Anstellung 91
4.19 Ergebnis einer Simulationsrechnung mit dem Nichtgleichgewichtsverfahren auf der NEC SX-5, Eintrittsgeschwindigkeit Mach 19,8 bei 40 Grad Anströmung 93

5.1 PACX-MPI logische Architektur 97
5.2 PACX-MPI, Schichtenkonzept 98
5.3 Konfiguration des interkontinentalen Metacomputers 99
5.4 Konvergenzvergleich der beiden Löservarianten 108
5.5 Vergleich der beiden Löservarianten mit unterschiedlichen Netzwerken 110

Kapitel 1

Einleitung

Die Weltraumfahrt beschäftigt die Menschen schon seit Jahrhunderten. Seit dieser Zeit setzt sich die Wissenschaft bereits mit den verschiedensten Problemen auseinander, die das Verlassen der Erde, ein Weltraumflug und die Rückkehr zur Erde mit sich bringen [80] [81]. Mit den ersten Weltraumflügen in den fünfziger und sechziger Jahren des vorigen Jahrhunderts sowie mit den Mondflügen der NASA waren diese Probleme bei weitem nicht alle zufriedenstellend gelöst. Nach dem Wettlauf um das Erreichen des Mondes und den ersten Raumstationen im Weltall verschob sich der Fokus der Weltraumfahrt von einer politisch motivierten Weltraumfahrt mit allen Mitteln hin zu einer mehr kosten-/ nutzenorientierten Wissenschaft, die neben dem Nachweis des Machbaren auch verwertbare Ergebnisse als Daseinsberechtigung liefern muß. Ein wesenlicher Punkt dabei ist eine gewisse Kosteneffizienz, die verlangt, daß Lösungen für Probleme bezahlbar bleiben. Ein erster Schritt in diese Richtung stellt das Space Shuttle der NASA dar, das erstmals eine gewisse Wiederverwendbarkeit von Raumfahrzeugen ermöglicht. Die Wiederverwendbarkeit des seit inzwischen über 20 Jahren verwendeten Shuttlesystems ist allerdings doch sehr begrenzt. Nach Informationen aus Quellen der NASA kostet jeder Spaceshuttlestart etwas über 500 Millionen Dollar – eigentlich sollte es aber auch für 50 Millionen Dollar gehen. Neben dem Treibstoff und den anderen Verbrauchsmitteln schlägt vor allem die technische Neuausrüstung zu Buche. Neben dem Haupttank, der als einzige Komponente des Shuttlesystems nicht wiederverwendbar ist, da er nach Abwurf in der Erdatmosphäre verglüht, verursachen vor allem die notwendigen Wartungs- und Reparaturarbeiten zur erneuten Flugvorbereitung des Shuttles enorme Kosten. Einer der wesentlichen Punkte dabei ist das Hitzeschutzsystem. Ein guter Teil der dazu verwendeten hitzeresistenten Keramikplatten muß nach jedem Flug ausgetauscht werden. Das war mit ein Grund für die Luft- und Raumfahrtwissenschaftler in Stuttgart, sich näher mit dem Thema der Rückkehr wiederverwendbarer Raumfahrzeuge auseinanderzusetzen.

1.1 Wiedereintrittsprobleme

Die Rückkehr eines Raumfahrzeugs zur Erde wirft erhebliche Probleme auf. Hauptproblem ist, daß ein Raumfahrzeug, das sich in der Umlaufbahn befindet oder von einer interplanetarischen oder gar interstellaren Reise zurückkehrt, auf die Rotationsgeschwindigkeit der Erdoberfläche abgebremst werden muß. Diese ist aber erheblich geringer als die Orbitalgeschwindigkeit oder die Reisegeschwindigkeit eines Raumfahrzeugs. Um also zur Erdoberfläche zurückkehren zu können, muß das Raumfahrzeug entsprechend abgebremst werden. Dieser Bremsvorgang kann zum Beispiel komplett durch Impuls entgegen der Flugrichtung vorgenommen werden. Da aber jedes Quentchen Treibstoff von Beginn der Reise an mitgeführt werden muß, suchte man nach effizienteren Methoden zur Abbremsung. Als günstigste Alternative hat sich von Anfang der Raumfahrt an die Ausnutzung der Erdatmosphäre als Bremsmedium herausgestellt. Dabei wird das Raumfahrzeug durch wenige kurze Impulse mit den Brems- und Steuerraketen aus der Umlaufbahn gelenkt. Beim folgenden Sturzflug auf die Erde zu wird die Reibung der Atmosphäre dazu verwendet, das Raumfahrzeug abzubremsen. Dabei gilt es natürlich zu beachten, daß der Wiedereintrittswinkel so gewählt wird, daß das Fahrzeug weder zu steil eintaucht und dabei das Hitzeschutzsystem thermisch überlastet wird und das Fahrzeug verglüht, noch sollte der Winkel zu flach sein, da sonst das Fahrzeug auf der Atmosphäre abprallen könnte und einen ungeplanten Kurs weg von der Erde nehmen würde. Da für dann notwendige Kurskorrekturen nicht genügend Treibstoff an Bord ist, würde das Fahrzeug mitsamt seiner Besatzung verloren gehen. Heute versucht man, diese Rückkehr von mehreren Seiten her zu optimieren, vor allem aber im Hinblick auf die Hitzeschutzsysteme und die Wiederverwendbarkeit der Raumtransporter hin.

Physikalisch gesehen muß die im System Raumfahrzeug vorhandene Lage- und Bewegungsenergie aus dem System herausgenommen werden. Diese Energie wird während des Wiedereintritts in Wärmeenergie umgewandelt, wobei sich die das Fahrzeug umgebende Atmosphäre und das Fahrzeug selbst stark erwärmen. Um das Fahrzeug und die Insassen vor den dabei auftretenden Temperaturen zu schützen, wurden verschiedene Hitzeschutzsysteme entwickelt. Bei den nur einmal verwendbaren Systemen der fünfziger bis siebziger Jahre des zwanzigsten Jahrhunderts wurde der Hitzeschutz durch ein Hitzeschild erzielt, der während des Wiedereintritts verdampfte bzw. pyrolisierte und durch die dafür notwendige Energie einen Kühleffekt für das Fahrzeug erreichte. Bei wiederverwendbaren Systemen ist es extrem teuer solch ein Verfahren anzuwenden, da das Hitzeschild nach jedem Flug komplett zu ersetzen ist. Für wiederverwendbare Fahrzeuge lag es da nahe, nach Materialien zu suchen, die eine hohe isolierende Wirkung haben ohne selbst Schaden zu nehmen. Aufgrund intensiver Forschung stehen hier inzwischen Materialen zur Verfügung, die sowohl leicht sind, als auch durch entsprechende Hitzebeständigkeit mehrfach verwendet werden können [21]. Durch die zusätzliche Anwendung spezieller Schutzschichten [44], kann für den Normal-

flug die Hitzebeständigkeit sogar so weit verbessert werden, daß die Materialen für zwanzig und mehr Wiedereintritte verwendet werden können. Allein die Schutzschicht ist vor einer neuerlichen Verwendung durch einfaches Anstreichen zu erneuern.

Ein weiterer Angiffspunkt zur Verbesserung des Wiedereintritts ist die Flugbahn. Die bestimmenden Materialeigenschaften für den Eintritt sind die maximal mögliche Temperatur des Hitzeschutzsystems und die Standzeit des Materials bei der jeweiligen Temperatur. Mit Hilfe von Flugbahnoptimierung [35] versucht man nun, die Wiedereintrittsbahn so zu optimieren, dass das Material möglichst wenig belastet wird, auch wenn das eine höhere integrale Wärmelast bedeutet. Die Wärmelast auf der Fahrzeugoberfläche ist dabei auch nicht gleichverteilt sondern hängt von der Geometrie des Fahrzeugs ab. An Staupunkten, wie zum Beispiel der Fahrzeugnase, treten dabei höhere Lasten auf, als zum Beispiel an der Fahrzeugoberseite. An diesen Stellen kann dann Material eingesetzt werden, das in der Herstellung einfacher und billiger ist, aber eben dafür nur geringeren Wärmelasten standhält. Um die Materialverteilung allerdings richtig vornehmen zu können ist es notwendig, die Wärmelasten auf der Oberfläche des Fahrzeugs in allen relevanten Flugzuständen möglichst genau zu kennen. Da Messungen aufgrund der hohen Temperaturen und der seltenen Messmöglichkeiten nur sehr schwierig und für neue Geometrien gar unmöglich sind, ist die einzige Möglichkeit diese Wärmelasten vorherzusagen, die Simulation der Strömungsgrößen auf einem Rechner. Natürlich sind auch Windkanalexperimente in entsprechenden Hochgeschwindigkeitswindkanälen möglich. Dort können aber nur Experimente an Modellen und auch nicht für einen gesamten Wiedereintritt gefahren werden. Damit lassen sich zwar Strömungen, aber nicht der Wärmestrom für den tatsächlichen Wiedereintritt bestimmen. Somit ist die Simulation die einzige Möglichkeit der Vorhersage.

1.2 Simulation von Wiedereintrittsströmungen

Die Berechnung der Strömung um in die Erdatmosphäre wiedereintretende Raumfahrzeuge ist auch heute noch ein nicht vollständig gelöstes Problem. Zur Auslegung der Materialien für die Fahrzeugoberfläche, die das Hitzeschutzsystem bilden, ist aber eine genügend genaue Berechnung der Wäermeströme auf der Fahrzeugoberfläche erforderlich. Dies bedingt wiederum eine genauere Berechnung der Wiedereintrittsströmung als mit den sonst üblichen 2-5%. Es ist vielmehr eine Genauigkeit von unter einem Prozent erforderlich.

Durch die hohe Eintrittsgeschwindigkeit ist die Berechnung der Strömung mit idealem Gas durch den auftretenden Schock, der mathematisch gesehen eine Unstetigkeit ist, bereits sehr komplex und erfordert entsprechend angepasste Verfahren, die aber heute an sich verfügbar sind. Eine zusätzliche Problematik stellt sich aber durch das Fluid, Luft, das sich während der Fahrzeugumströmung laufend ändert. Ursache dafür sind die bei der hohen Wiedereintrittsgeschwindig-

keit auftretenden Temperaturen, bei denen die Fluidbestandteile, hauptsächlich Stickstoff und Sauerstoff, anfangen miteinander zu reagieren. Problematisch daran ist, daß diese chemische Reaktionen in einem ähnlichen Geschwindigkeitsbereich ablaufen wie die Strömung selbst. Wäre die Reaktionsgeschwindigkeit höher, könnte man in etwa einen Umschlagspunkt bestimmen und vor und nach diesem Umschlagspunkt mit konstanten Fluidwerten rechnen. Wäre die Reaktionsgeschwindigkeit geringer, wäre es dagegen möglich, von einer quasi konstanten Fluidzusammensetzung auszugehen. Somit könnte man ein gewöhnliches Überschallverfahren einsetzen. Da sich aber die Fluidzusammensetzung während der gesamten Fahrzeugumströmung nicht vernachlässigbar ändert, leider auch nicht schnell genug, um einen Umschlagspunkt festlegen zu können, müssen die für die Fluidzusammensetzung relevanten chemischen Reaktionen und ihre Spezies in der Strömungssimulation mit berücksichtigt, also mit berechnet werden.

Einen erheblichen Einfluß auf diese Reaktionen hat darüberhinaus die Beschaffenheit, also das Material, des Hitzeschutzsystems des Fahrzeugs. Genauer gesagt, dessen chemisches Verhalten. Moderne Hitzeschildmaterialen wie SiO2, SiC oder C/C-SiC wirken hier voll- bzw. teilkatalytisch, was dann aber im Simulationsverfahren berücksichtigt werden muß, wenn die Strömung und damit die Wärmeströme auf der Fahrzeugoberfläche mit genügender Genauigkeit berechnet werden sollen.

1.3 Anforderungen an eine detaillierte Simulation von Wiedereintrittsströmungen

Für die geforderte genaue Berechnung der Wärmeströme auf der Fahrzeugoberfläche sind vom Simulationsprogramm weitere Eigenschaften gefordert. Die beschriebene Berücksichtigung der chemischen Reaktionen hat zur Folge, daß der Berechnungsaufwand erheblich steigt, da die Quellterme für die Reaktionen und die komplexere Randbehandlung zusätzlich ins Gewicht fallen. Darüberhinaus steigt der Speicherbedarf quadratisch mit der Anzahl der Spezies an, die zusätzlich zu den notwendigen Strömungsdaten mit abgelegt werden müssen.

Zur Berechnung der Strömung werden die zu berechnenden Körper mit einem Netz umgeben, auf dessen Zellen dann die notwendigen Gleichungen gelöst werden. Um die geforderte Genauigkeit zu erreichen ist es notwendig, fein genug zu diskretisieren, also in jede der drei Raumdimensionen genügend solche Zellen zu haben. Dazu muß das Netz fein genug aufgelöst sein.

Mit steigender Zellenzahl steigt aber auch der Berechnungsaufwand an. In Abhängigkeit des verwendenten numerischen Verfahrens hängt der Berechungsaufwand pro Iteration zumindest linear von der Anzahl der Netzzellen ab. Darüber hinaus nimmt aber bei feineren Netzen die Konvergenzgeschwindigkeit des Verfahrens selbst auch ab. Damit ist also bis zu einer "gleich guten" Lösung, das

heißt bis zum gleichen Residuum, verglichen mit einem gröberen Netz, ein erheblich höherer Rechenaufwand notwendig.

Die Vernetzung der Wiedereintrittskörper selbst verursacht ebenfalls große Schwierigkeiten. Einerseits sollen auch an sowohl für die Vernetzung als auch für die Berechnung kritischen Stellen wie Körperecken und Kanten sowie in der Nähe des Staupunkts keine unförmigen Netzstellen entstehen, da sie im numerischen Verfahren zu zusätzlichen Schwierigkeiten führen. Dies schlägt sich im günstigen Fall in geringerer Konvergenzgeschwindigkeit nieder, führt aber meist eher zur Divergenz des ganzen Verfahrens. Andererseits sollen es die Netze erlauben auch den Fahrzeugnachlauf aufzulösen. Darüberhinaus soll es möglich sein, auch komplexere Geometrien als ein einfaches Raumfahrzeug, so zum Beispiel eine komplette X-38 Konfiguration mit angelenkter Steuerklappe zur Einstellung des Wiedereintrittswinkels, zu vernetzen und damit berechnen zu können.

In dieser Arbeit sollen nun diese zuletzt beschriebenen Probleme angegangen werden. Das heißt, es werden Lösungen gesucht, die es erlauben, die Berechungen für entsprechend komplexe Topologien mit einem Feinheitsgrad durchzuführen, der es erlaubt alle relevanten Phänomene aufzulösen. Dazu müssen die Probleme der Vernetzung der Körper und des hohen Berechungsaufwands bei der Simulation reagierender Strömungen um diese Körper angegangen werden.

Kapitel 2

Stand der Technik

In diesem Kapitel wird die technische Ausgangslage zu Beginn der Arbeit dargestellt. Dies umfaßt neben einer Darstellung des Standes des Programmpakets URANUS eine Übersicht über die Möglichkeiten zur Lösung der am Ende des ersten Kapitels dargestellten Problematik sowie eine Diskussion dieser Möglichkeiten.

2.1 Das Simulationspaket URANUS

Seit über zehn Jahren wird am Institut für Raumfahrtsysteme (IRS) der Universität Stuttgart inzwischen am URANUS (Upwind Relaxation Algorithm for Nonequilibrium Flows of the University Stuttgart) Verfahren gearbeitet. Ziel ist, einen Simulationscode zur Verfügung zu haben, der genau die Vorhersage der im letzten Kapitel erwähnten Oberflächenwärmeströme für verschiedene Oberflächen und in einem großen Geschwindigkeitsbereich mit der erforderlichen Genauigkeit möglichst dreidimensional leistet.

Hervorgegangen aus einem Programm zur Berechnung von Strömungen in Flugzeugturbinen [74] wurde URANUS im Sonderforschungsbereich 259 (Hochtemperaturprobleme rückkehrfähiger Raumtransportsysteme) schrittweise an die neue Aufgabe angepasst. Neben der Umstellung auf C-Netze für die Körperumströmung wurden hauptsächlich neue Verfahren für die Behandlung der chemischen Reaktion des Fluids vor allem im Zusammenhang mit den chemischen Reaktionen an der Körperoberfläche entwickelt und in den zweidimensionalen Programmcode eingeführt [16] [48] [49] [52] .

Zu Beginn dieser Arbeit lagen folgende Programmversionen von URANUS vor:

- 3-D Verfahren single Block C-mesh für Euler
- 2-D Verfahren für die reagierenden Strömungen ohne Ionisierung

- 2-D Randbehandlung des Festkörpers mit katalytischer bzw. teilkatalytischer Oberfläche (CVCV-Modell zur Vibrationskopplung).

Parallel zur Lösung der Rechenzeit- und Netzproblematik, die in dieser Arbeit dargestellt wird, wurde URANUS in folgenden Bereichen weiterentwickelt:

- Erweiterung des Verfahrens für reagierende Strömungen ohne Ionisierung auf den 3-D Fall
- Entwicklung eines Verfahrens für reagierende Strömungen mit Ionisierung (vorerst 2-D) zur Abdeckung des Hochgeschwindigkeitswiedereintritts mit Geschwindigkeiten um Mach 20 bei der Rückkehr von interplanetaren Missionen.
- Entwicklung eines effizienten numerischen Lösungsverfahrens für stationäre Hyperschallströmungen unter Berücksichtigung des thermochemischen Nichtgleichgewichts.

Nach einer Diplomarbeit [3] war das 3-D Einblockverfahren für C-Netze in einer ersten parallelen Version verfügbar.

In den folgenden Unterkapiteln werden nun die verschiedenen möglichen Ansätze zur Lösung der Vernetzungs- und Rechenzeitproblematik untersucht.

2.2 Lösungsansätze für die Vernetzungsproblematik

Um das Problem der unförmigen Netzzellen an Kanten und anderen exponierten Stellen sowie die Vernetzung komplexer Topologien zu lösen, existieren verschiedene Ansätze. Allein das Erzeugen entsprechend geeigneter Netze setzt ein großes Maß an Erfahrung und die Verwendung von hochwertigen Netzgeneratoren voraus. Allein in die Entwicklung dieser Netzgeneratoren und der notwendigen Mathematik auf denen diese basieren, wurden hunderte Mannjahre investiert. Außerdem ist die Netzgenerierung nach wie vor das Thema aktueller Forschung [75]. Zur Lösung der Problematik in URANUS stehen im Prinzip drei Möglichkeiten zur Auswahl, die wir nachfolgend genauer untersuchen wollen.

2.2.1 Unstrukturierte Gitter

Bisher werden in URANUS sogenannte strukturierte Gitter eingesetzt, die aus, bisweilen verzerrten, Quadern bestehen, die aneinander gereiht sind. Dabei hat jeder dieser Quader auf jeder seiner sechs Seiten einen eindeutig bestimmten Nachbarquader, oder einen Rand des Rechengebiets, den wir hier als physikalischen Rand bezeichnen, da hier für die Rechnung eine Randbedingung aufgrund

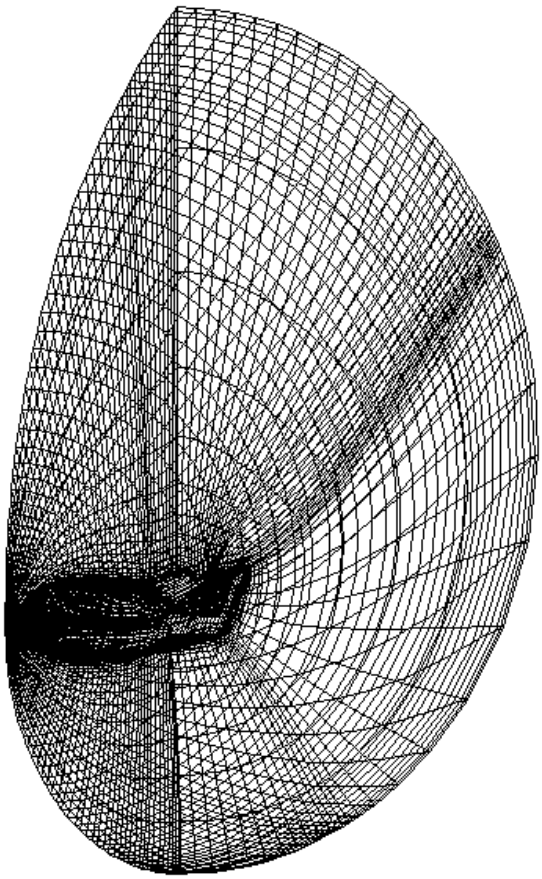

Abbildung 2.1: Strukturiertes Gitter für die X-38

der Physik vorgegeben wird. Ein Gitter wird als karthesisches Gitter bezeichnet, wenn die Quader nicht verzerrt sind, also alle dieselbe Größe und rechte Winkel besitzen. Ein strukturiertes Gitter zeichnet sich nun dadurch aus, dass es sich durch eine Abbildung in ein karthesisches Gitter überführen läßt. Abbildung 2.1 zeigt ein solches strukturiertes Gitter.

Unstrukturierte Netze hingegen können aus beliebigen geometrischen Objekten bestehen, die einfach aneinander gereiht werden. Im einfachsten Fall werden hier im zweidimensionalen Fall Dreiecke oder in 3-D Tetraeder verwendet. Das Netz wird hier durch die Nachbarschaftsbeziehung der Netzelemente beschrieben.

Mit solchen unstrukturierten Gittern sind selbst komplexe Topologien recht einfach zu vernetzen, wie das Oberflächennetz um ein Flugzeug, hier speziell der Ausschnitt um die Triebwerkskonfiguration, in Abbildung 2.2 zeigt. Allerdings bergen sie andere Schwierigkeiten: Solche unstrukturierten Gitter sind von der Numerik und Algorithmik her erheblich aufwendiger zu behandeln [41] und auch die Optimierung des Rechenprogramms ist sehr kompliziert. Allerdings wurden

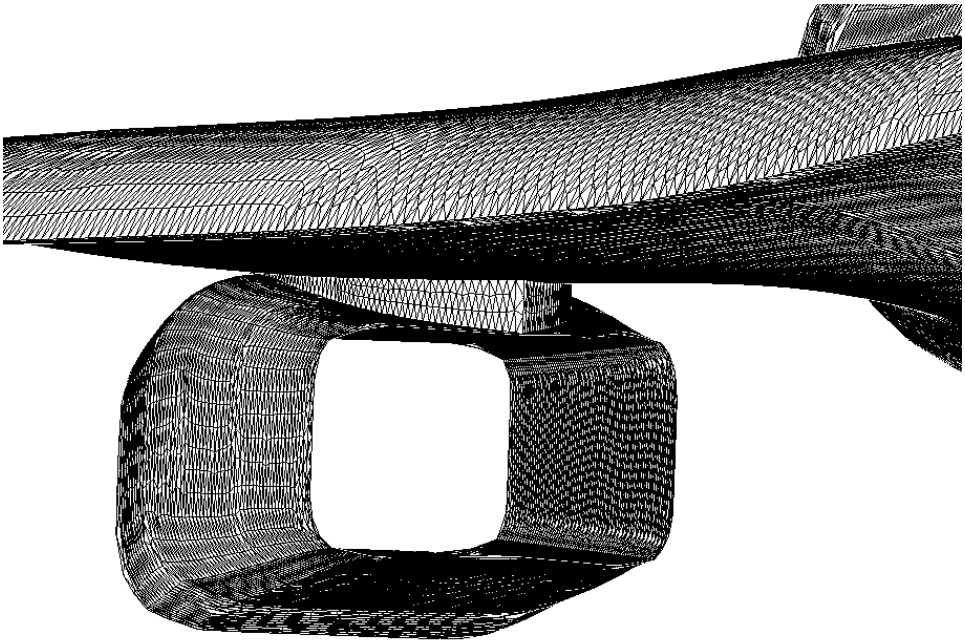

Abbildung 2.2: Unstrukturiertes Gitter

hier in den letzten Jahren erhebliche Anstrengungen unternommen, die zu gut optimierten Programmen geführt haben. Diese Programme laufen naturgemäß auf cachebasierten Systemen im Verhältnis besser als auf Vektorsystemen. Hier sind Programme, die strukturierte Netze verwenden, nachwievor im Vorteil. Dies macht aber den Nachteil der aufwendigen Netzgenerierung oft nicht wett, so dass viele Programme heute nur unstrukturierte Netze verwenden [40]. Programme, die beide Netztypen unterstützen sind selten, da hier der Pflege- und Weiterentwicklungsaufwand fast doppelt so hoch ist.

Für URANUS kommt noch eine weitere Schwierigkeit hinzu, da hier nicht nur Strömung, sondern auch chemische Reaktionen, insbesondere an der Oberfläche berechnet werden. Die gesamte Entwicklung der hierzu verwendeten Verfahren basiert auf strukturierten Gittern. Eine Umsetzung dieser Verfahren auf unstrukturierte Gitter ist bisher weder geplant, noch ist der Aufwand für eine solche Umsetzung abgeschätzt. Eine Verwendung unstrukturierter Gitter für URANUS kam daher zur Lösung der Vernetzungsproblematik nicht in Frage. Die Lösung des Problems muss somit zwangsweise auf strukturierten Gittern basieren. Die hier möglichen Verfahren werden in den beiden nächsten Abschnitten vorgestellt.

2.2.2 Chimera-Technik

Die Chimera-Technik wurde bereits in den achtziger Jahren des vorigen Jahrhunderts entwickelt und erstmals verwendet [79]. Die grundlegende Idee hinter dem Verfahren ist recht einfach, wenn auch die Umsetzung erheblich komplexer ist. Die Idee ist, voneinander relativ unabhängige Teile der Topologie getrennt mit jeweils einem (strukturierten) Gitter zu vernetzen und diese einzelnen Teile bei der Berechnung dann zu überlagern. Natürlich muss dabei im Überlagerungsbereich jeweils eine Schnittstelle definiert und dort zwischen den Netzen interpoliert werden. Die jeweiligen Netzteile, in denen sich der andere Körper befindet, wird dabei nicht berücksichtigt, also quasi abgeschaltet. Besonders bewährt hat sich dieser Ansatz bei der Berechnung von Topologien, bei denen sich Teile während der Berechnung bewegen oder sonst ihre Position gegeneinander verändern. So wird diese Technologie zum Beispiel bei der Berechnung von Strömungen um Helikopter oder zur Simulation von Trennungs- bzw. Abwurfvorgängen eingesetzt.

Für URANUS ist dieser Ansatz leider auch nicht geeignet. Zwar können mit diesem Ansatz Teile der Topologie elegant aufgelöst werden, wie zum Beispiel die Klappen an Raumfahrzeugen, aber die Hauptproblematik, die ungünstigen Zellzerrungen an Nasen und Tragflächenkanten, ist hiermit nicht zu umgehen. Weiterhin stellt sich auch hier die Problematik der Interpolation für den Chemieanteil, insbesondere im Bereich der Oberflächen, der für die genaue Berechnung des Wärmestroms ja insbesondere relevant ist. Der Einsatz von der Chimeratechnologie kommt für das gestelle Problem also auch nicht in Frage.

2.2.3 Multiblockverfahren

Mit Multiblockverfahren wird versucht, die Vorteile der unstrukturierten und strukturierten Gitter zu verbinden. Multiblocknetze bestehen aus beliebig angeordneten Quadern. Wobei an Kanten nicht unbedingt immer vier Quader und an Ecken acht Quader aneinander stossen müssen. Vielmehr ist hier eine gewisse unstrukturierte Anordnung der Quader erlaubt. Die Quader selbst hingegen bestehen aus einem strukturierten Gitter, wobei die Anzahl der Gitterpunkte auf einander benachbarten Flächen der Quader gleich sein muß. Somit hat jede innere Gitterzelle auch in einem Multiblocknetz genau sechs Nachbarn, mit denen sie eine gemeinsame Fläche hat.

Mit Multiblocknetzen lassen sich alle Topologien vernetzen, ohne dass dazu Zellen zu stark verzerrt werden müssen. Allerdings ist der Netzgenerator entsprechend aufwendig und auch die Netzgenerierung erfordert einiges an Erfahrung. Der Vorteil für URANUS liegt aber darin, daß die bereits entwickelten und getesteten Verfahren insbesondere zur Berechnung der chemischen Reaktionen in der Strömung und auf der Oberfläche einfach übertragen werden können.

2.3 Ansätze zur Erhöhung der Leistungsfähigkeit des Programms

Zur Erhöhung der Leistungsfähigkeit des Programms gibt es grundsätzlich gesehen zwei Möglichkeiten: Die Anpassung des Programms für den effizienten Einsatz auf den leistungsstärksten Rechnern also modernsten Supercomputerplattformen und die Verbesserung der Algorithmik durch den Einsatz modernster Lösungsverfahren. Beide Möglichkeiten sollen hier näher betrachtet werden.

2.3.1 Parallelisierung

Alle Rechner der höchsten Leistungsklasse arbeiten heutzutage massiv parallel, wie sich in der Top 500 Liste [17] der leistungsfähigsten Rechner schnell feststellen läßt. Das bedeutet, daß in diesen Rechnern eine große Anzahl an Prozessoren verfügbar ist, auf denen das Programm gleichzeitig abläuft. Dabei ist der verfügbare Hauptspeicher, zumindest physikalisch, einzelnen Prozessoren oder Prozessorgruppen zugeordnet. Zwar ist bei manchen Systemen eine globale Adressierung möglich, jedoch ist dabei die Zugriffszeit auf den Speicher stark davon abhängig, wem der Speicher zugeordnet ist. Ist der adressierte Speicher lokal, so erfolgt der Zugriff erheblich schneller als wenn auf Speicher einer anderen Prozessorgruppe zugegriffen wird.
Um URANUS auf solchen Plattformen einsetzen zu können, muß das Programm auf jeden Fall parallelisiert sein. Im Prinzip gibt es auch hier wieder zwei Möglichkeiten. Die Parallelisierung mittels Direktiven, die eine Arbeitsverteilung definieren und eine komplette Parallelisierung der gesamten Programmstruktur durch eine Gebietszerlegung und Verteilung der Gebiete auf die verschiedenen Prozessoren oder Prozessorgruppen. Die Parallelisierung allein mittels Direktiven führt aber nicht zum gewünschten Ziel, dem effizienten Einsatz des Programms auf den schnellsten Rechnern. Die Gründe dafür sind vielfältig: erstens ist die dargestellte globale Adressierung bei weitem nicht auf allen Systemen verfügbar, vor allem die größten und schnellsten Systeme unterstützen diese nicht. Die cachekohärente, globale Adressierbarkeit des Hauptspeichers ist aber die Voraussetzung für den Einsatz der Direktiven zur Parallelisierung. Zweitens ist nachgewiesen [15], daß die Art der Parallelisierung durch Arbeitsverteilung effizient nur bis zur einer gewissen Anzahl von sogenannten Threads gut funktioniert. Bei der Verwendung größerer Prozessorzahlen muß also auch bei der Verwendung von Direktiven in der Art der Gebietszerlegung programmiert werden. Somit ist es also günstiger, das Programm gleich mittels Gebietszerlegung und Parallelisierung der ganzen Programmstruktur für Rechner mit verteiltem Hauptspeicher auszulegen. Damit ist ein effizienter Einsatz des Programms auf Rechnern der höchsten Leistungsklasse auf jeden Fall sichergestellt. Der parallelen Programmierung widmet sich ein weiterer Abschnitt dieses Kapitels, zuerst wollen wir aber noch einen Blick auf

zwei Programmbibliotheken werfen, die zu Beginn der Arbeit bereits verfügbar waren und die Parallelisierung von Strömungsprogrammen unterstützen sollen.

2.3.2 Die Bibliothek CLIC

CLIC wurde seit 1986 am SCAI (Institut für Algorithmen und Wissenschaftliches Rechnen) der GMD (Gesellschaft für Mathematik und Datenverarbeitung, inzwischen in die Fraunhofer-Gesellschaft integriert) entwickelt. Ziel war es, die Parallelisierung von Simulationsprogrammen zu vereinfachen, die dreidimensionale Multiblocknetze verwenden [72]. Während CLIC eine ausgefeilte Behandlung der Multiblocknetze besitzt, steht die Verteilung der Blöcke auf die vorhandenen Prozessoren im Hintergrund. Eine eventuell notwendige Zerlegung bzw. Zusammenfassung von Blöcken wird auf einen Preprocessing Schritt verschoben, der jeweils vor der eigentlichen Rechnung, unabhängig von dieser, sequentiell durchgeführt wird. Die Verteilung der dabei entstandenen Blöcke auf die im Lauf vorhandenen Prozesse findet dann nach einer einfachen Heuristik statt. Somit kann die Lastverteilung bei weitem nicht als optimal bezeichnet werden. Die zu Beginn der Arbeit vorliegende Version von CLIC bot auch keinerlei Unterstützung für eine adaptive Verfeinerung von Blöcken über Blockgrenzen hinaus. Ebenso war in diesem Fall keine dynamische Änderung der Lastverteilung möglich. Das größte Problem war jedoch, daß CLIC noch nicht in einer Version vorlag, die intern MPI, also den Message Passing Standard, verwendete.

2.3.3 Die Bibliothek LOCO

Im Gegensatz zu CLIC liegt der Schwerpunkt der Bibliothek LOCO auf der dynamischen Lastverteilung und der adaptiven Verfeinerung [51]. Hierzu handhabt die Bibliothek die Verteilung der Blöcke auf die Prozessoren sowie die Kommunikation zwischen den Blöcken. Allerdings arbeitet das Verfahren nur mit Programmen, die bereits in der Lage sind Multiblocknetze zu verwenden, darüberhinaus bisher nur im zweidimensionalen Fall. Außerdem ist die Bibliothek nur für die native NX-Bibliothek auf Intel Systemen und für PVM verfügbar.

Neben der reinen Parallelisierung bietet sich auch die Verwendung moderner numerischer Algorithmen zur Lösung der partiellen Differentialgleichungen zur Erhöhung der Leistungsfähigkeit an. Die erfolgversprechensten Lösungsansätze werden hier kurz vorgestellt.

2.3.4 Mehrgitterverfahren

Mehrgitterverfahren haben in den letzten Jahren vor allem bei der schnellen Lösung elliptscher Gleichungen weite Verbreitung gefunden. Weitaus weniger Erfahrung gibt es hingegen bei der Behandlung von Überschallströmungen mit

Mehrgitterverfahren. Vor allem der Stoßbereich hat sich bei der Verwendung dieser Verfahren als problematisch erwiesen und benötigt eine aufwendige Sonderbehandlung [32] [63]. Die Arbeiten an Mehrgitterverfahren bei der Simulation von chemisch reagierenden Strömungen steckten seinerzeit auch noch in den Kinderschuhen und erste Ergebnisse zeigten noch keine bahnbrechenden Erfolge [76]. Daher wird in URANUS vorerst kein Mehrgitterverfahren eingesetzt. Allerdings wurde aus Gründen der Zukunftssicherheit die Datenstruktur bereits so entworfen und implementiert, dass das Verfahren später relativ einfach um ein Mehrgitterverfahren erweitert werden kann. Ein anderes mehrgitterartiges Verfahren sind kaskadische Mehrgitter- und Mehrebenenverfahren [11]. Hierbei werden die unterschiedlichen Netzebenen nicht wie im Mehrgitterverfahren zur Korrektur des Fehlers eingesetzt, sondern es werden echt Lösungen auf den Gittern berechnet. Bei einem Kaskadenverfahren wird auf einem groben Gitter begonnen und dies dann schrittweise bis zum Zielgitter verfeinert. Dabei werden wie in einem klassischen Mehrgitterverfahren die Ergebnisse des jeweiligen Netzes prolongiert. Obwohl auch hier fast noch keine Erfahrung mit dem Einsatz dieser Methodik für die Simulation von Überschallströmungen besteht, so läßt das Prinzip doch einiges an Potential erwarten, denn das implizite URANUS Verfahren besitzt eine lange Anrechenphase, in der vor allem der Stoß vom Körper wegläuft und positioniert wird. Da ist ein solches Verfahren zumindest zur Verkürzung der Anrechenphase erfolgversprechend.

2.3.5 Adaptive Netzverfeinerung

Den Mehrgitterverfahren eng verwandt ist die sogenannte adaptive Netzverfeinerung [12]. Hierbei werden mit dem bereits implementierten Simulationsverfahren Lösungen auf einem relativ groben Gitter berechnet, das allerdings in einigen Bereichen schon hinreichend genaue Lösungen liefert. Anschließend werden nur Teilgitter, auf denen die Lösung noch nicht hinreichend genau ist, bei Bedarf mehrfach verfeinert. Je nach Behandlung der unterschiedlichen Teilnetze in den verschiedenen Verfeinerungsstufen existieren bereits vom Originalvorschlag unterschiedliche Ansätze [57], die sich nur in kleinen, aber je nach Anwendung sehr wirkungsvollen, Details unterscheiden. Ein Teil der Verfahren wurde bereits auch zur effizienten Verwendung auf Parallelrechnern angepaßt [39].

Weitere Ansätze wie zum Beispiel die Leistungssteigerung durch anders geartetet Lösungsverfahren werden in weiteren Arbeiten innerhalb des URANUS Projekts durchgeführt.

2.4 Programmiermodelle für die Parallele Programmierung

Die Erstellung einer portablen parallelen Applikation setzt die Verwendung eines standardisierten parallelen Programmiermodells voraus. Nur so ist gesichert, daß die Applikation auf möglichst vielen aktuellen und zukünftigen parallelen Plattformen jedwelcher Art eingesetzt werden kann. In den letzten Jahren wurden in diesem Bereich zwei Standards verabschiedet: MPI [58] und OpenMP [61].

OpenMP ist das Standard-Programmiermodell zur direktivenbasierten parallelen Programmierung auf Systemen mit gemeinsamem Hauptspeicher (SMP's) und Systemen mit, zumindest logisch gesehen, gemeinsamem Speicher (cc-NUMA Systeme). OpenMP verwendet vom Programmierer einzufügende Direktiven zur Steuerung der Parallelisierung. In vom Benutzer zu spezifizierenden parallelen Bereichen werden dann die mit Direktiven versehenen Schleifen parallel berechnet, das heißt der Schleifenrumpf wird auf die Prozessoren kopiert und dort jeweils für einen Teilbereich der Schleifenindizes ausgeführt. OpenMP basiert also hauptsächlich auf einer einfachen Schleifenparallelisierung, obwohl auch andere Bereiche, sogenannte Parallel Sections, zur parallelen Bearbeitung im Programm ausgewiesen werden können.

Für den gesamten Bereich der Message Passing Programmiermodelle hat sich seit 1995 MPI (Message Passing Interface) als Standard [58] etabliert, der verschiedenste herstellerabhängige Programmiermodelle und auch Public Domain Systeme wie PVM ersetzt. Message Passing wurde für Massiv Parallele Prozessoren (MPP's) entwickelt, die typischerweise aus vielen Rechenknoten bestehen. Diese Rechenknoten sind für sich gesehen quasi eigenständige Rechner also ein oder mehrere Prozessoren mit ihrem Hauptspeicher und der notwendigen zusätzlichen Hardware. Diese Knoten sind miteinander mittels eines schnellen Netzwerkes verbunden. Durch den Austausch von Nachrichten (Messages) zwischen den Rechenknoten können diese zusammen an einem Problem arbeiten. MPI erlaubt durch entsprechende Bibliotheksfunktionen das relativ einfache Versenden und Empfangen von Nachrichten, die aus zusammenhängenden Speicherblöcken bestehen. Für die parallele Programmierung stehen daneben aber auch viele sehr mächtige und umfangreiche Funktionen zur Verfügung, welche die Programmierung erheblich vereinfachen. Die Broadcastfunktion erlaubt es zum Beispiel, dass ein Speicherblock nicht nur an einen, sondern an alle am Programmlauf beteiligten Rechenknoten verschickt wird.

MPI wurde von den Herstellern aber auch für die Plattformen mit gemeinsamem Hauptspeicher implementiert, so daß es quasi auf jeder parallelen Plattform zur Verfügung steht. Für Workstation bzw. PC-Netzwerke und Cluster existieren Public Domain Implementierungen, so daß MPI auch auf dieser Art von Syste-

men verfügbar ist. So konnte die Entwicklung der parallelen Applikation in dieser Arbeit schon allein aus Gründen der Portabilität nur mittels MPI erfolgen, da ein Ziel die Verwendbarkeit der Applikation auf möglichst vielen parallelen Plattformen ist.

High Performance Fortran (HPF) [42], ein direktivengesteuerter Parallelisierungsansatz, zielt darauf ab, Datenstrukturen zu verteilen. Die bei der Berechnung dann notwendige Kommunikation wird implizit, also ohne weiteren Eingriff des Programmierers, ausgeführt. HPF tendiert dabei allerdings dazu, zuviel und an ungünstigen Stellen zu kommunizieren, so daß die damit entwickelten Programme von der Leistung her gesehen nicht mit den mittels MPI parallelisierten Programmen konkurrieren können. Daher konnte sich HPF bisher nicht durchsetzen und wurde für diese Arbeit nicht weiter in Betracht gezogen.

Kapitel 3

Ein effizientes Multiblockverfahren für Parallelrechner

Die zu Beginn dieser Arbeit vorhandene Version des Programms URANUS löst die Navier-Stokes-Gleichungen auf sogenannten C-Netzen. Der Name für diese Netzklasse leitet sich aus der Netzform ab, da sich im Zweidimensionalen die Netze C-förmig um den zu berechnenden Körperschnitt legen. Sollen Körper in ihrer räumlichen Ausdehnung dreidimensional berechnet werden, so ist dies ebenfalls mit C-Netzen möglich, wobei diese dann quasi um den Körper rotiert werden. Eine Verwendung anderer Netzarten war mit URANUS nicht möglich. Eben diese C-Netze besitzen allerdings mehrere unerwünschte Eigenschaften. Bereits bei der Vernetzung relativ einfacher Körper werden die einzelnen Netzzellen zum Teil stark verformt, was sich bei der Berechnung in einer oft erheblich verringerten Konvergenzgeschwindigkeit bemerkbar macht. Diese Zellen treten immer im Bereich der Rotationsachse auf (vgl. Abbildung 3.1), die darüberhinaus eine Sonderbehandlung der direkt anliegenden Zellen notwendig macht. Aber nicht nur an der Rotationsachse, sondern auch an sonstigen Kanten werden bei C-Netzen Zellen stark verformt. Hauptproblem ist aber, daß sich mit C-Netzen nicht alle Körper vernetzen lassen. Hauptsächlich Klappenkonfigurationen sowie mehrteilige Körper und alle Körper mit Spalten zwischen einzelnen Teilen, lassen sich nicht vernetzen. Die Vernetzung eines Nachlaufbereichs ist im allgemeinen ebenfalls nicht möglich. Da aber moderne Raumgleiter solche Bereiche aufweisen, lassen sie sich nicht in gewünschter Weise mit C-Netzen berechnen. Dies führte zum Entschluß, URANUS so zu erweitern, daß Multiblocknetze zur Berechnung verwendet werden können.

Das in URANUS verwendet implizite Lösungsverfahren wirft ein weiteres Problem auf. Es ermöglicht zwar bei sauberer Umsetzung eine quadratische Konvergenz, was durch entsprechende Untersuchungen der Projektpartner inzwischen

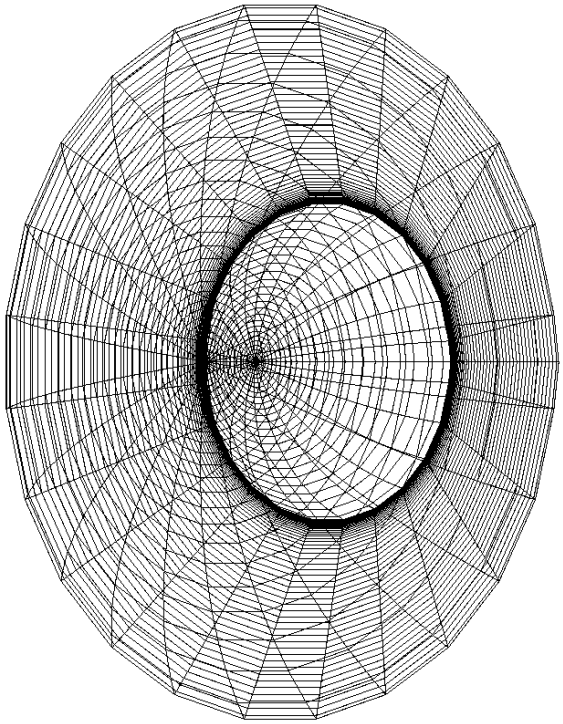

Abbildung 3.1: C-Netz um die Nase eines Wiedereintrittskörpers

auch nachgewiesen ist [62], erhöht aber auch den Speicherbedarf des Programms. Weiterhin ist der Rechenzeitbedarf bei der geforderten hohen Auflösung der Rechennetze um die Raumfahrzeuge erheblich. Die Erweiterung des Verfahrens auf die Berechnung der dreidimensionalen Nichtgleichgewichtsströmungen, also die Einführung der Berechung der ebenfalls ablaufenden chemischen Reaktionen im Fluid erhöht den Rechenzeitbedarf abermals erheblich. Um also mit dem Verfahren die Strömung um aktuelle Raumgleiter mit vernünftiger Auflösung im chemisch thermischen Nichtgleichgewicht berechnen zu können, ist also nicht nur die Verwendung von Multiblocknetzen, sondern auch der Einsatz entsprechend leistungsfähiger Rechensysteme notwendig.

Diese Leistungsfähigkeit sowohl in Hinblick auf notwendige Rechenleistung und ebenso Speicherkapazität bieten nur Supercomputer der obersten Leistungsklasse wie NEC SX-5, Cray T3E und Hitachi SR8000. Diese Rechner sind typischerweise als Systeme mit verteiltem Hauptspeicher ausgeführt, wobei die einzelnen Knoten des Systems sehrwohl Rechner oder Rechenelemente sein können, bei denen intern wiederum mehrere Prozessoren auf einem gemeinsamen Speicher (Shared Memory) arbeiten. Um diese Maschinen effizient nutzen zu können ist allerdings ein paralleles Simulationsprogramm notwendig, dass auf einem Programmiermodell für Parallelrechner mit verteiltem Speicher wie MPI aufsetzt. Die für eine hohe parallele Effizienz dabei empfehlenswerte Verteilung der Arbeit

auf die Prozessoren bzw. Knoten auf Basis einer Gebietszerlegung bedingt meist eine komplette Überarbeitung des gesamten Simulationsprogramms.

3.1 Ansatz

Programme, die von anderen Institutionen zur Strömungsberechnung eingesetzt werden, lagen schon vor ihrem Einsatz auf parallelen Supercomputern in einer Multiblockversion vor. Diese Programme wurden dann für die Verwendung auf diesen Supercomputerarchitekturen angepasst, also parallelisiert. Da das URANUS Programm zu Projektbeginn nur in einer Einblockversion aber auch schon parallel vorlag [3], konnte für das geforderte parallele URANUS Multiblockverfahren ein anderer Ansatz erfolgen. Die Gebietszerlegung für das parallele Einblockverfahren berechnet ja schon Lösungen auf einem Multiblocknetz, wenn auch auf einem sehr einfachen. Die Idee ist nun, das parallele Verfahren so zu erweitern, anzupassen und auszubauen, daß die allgemeine Klasse der Multiblocknetze berechnet werden kann. Dazu werden notwendige Erweiterungen für die Blockbehandlung mit denen der Parallelisierung verbunden, so daß ein neues paralleles Multiblockverfahren entsteht, das die Parallelität durch die einzelnen Netzblöcke ausnutzt bzw. diese Parallelität sogar durch weitere Zerlegung der Netzblöcke vergrößert. Durch die Multiblocknetze notwendig gewordene Erweiterungen wie zum Beispiel die Behandlung der unterschiedlichen Indexrichtungen auf den Blöcken konnten so mit in dem Datenaustausch integriert und damit einheitlich behandelt werden. Dadurch ist die Parallelisierung untrennbar mit der Multiblockberechnung verbunden. Der Ansatz bietet damit aber die Möglichkeit zukünftige Entwicklung von parallelen Multiblocksimulationsprogrammen erheblich zu vereinfachen. Ein Ingenieur braucht nur noch den Berechnungskern zu erstellen, also die notwendigen Berechnungen für einen Block, sowie für alle möglichen auftretenden Randarten und Randbedingungen. Die Parallelisierung (inklusive einer Anpassung für das Metacomputing) und die Möglichkeit Multiblocknetze zu verwenden, wird durch die hier erstellten Programmteile abgedeckt.

3.2 Konzeption

Das zu entwickelnde Programm soll möglichst die gesamte allgemeine Klasse der Multiblocknetze abdecken. Dazu muß diese Klasse zuerst genauer untersucht und deren Eigenschaften ermittelt werden. Daraus sind dann die notwendigen Programmeigenschaften abzuleiten.

3.2.1 Eigenschaften von Multiblocknetzen

Wie wir bereits festgestellt haben, sind Multiblocknetze nichts anderes als unstrukturiert angeordnete Blöcke strukturierter Netzteile. Diese Netzteile müssen

allerdings Quaderform aufweisen, da sie nur so einen strukturierten Block ergeben. Das bedeutet aber, wie im rein strukturierten Fall, daß jede Zelle an jeder ihrer Flächen genau eine Nachbarzelle hat, an die sie grenzt. Dies gilt insbesondere auch für Zellen, deren Nachbarn auf einem anderen, benachbarten Block liegen. Das bedeutet aber, daß auf Zellebene bei der Kommunikation keine Interpolation notwendig ist, so lange nicht einer der Blöcke eine andere Auflösung besitzt. Dies kommt aber erst vor, wenn wir die lokale NetzVerfeinerung einführen.

Da mit der Neukonzeption des Programms zur Berechnung der Multiblocknetze eine komplette Überarbeitung der Datenstrukturen notwendig ist, wurde die adaptive Verfeinerung beim Entwurf der neuen Datenstrukturen, sowohl für den Block als auch für die Kommunikation bereits berücksichtigt. Ebenso fiel, quasi als Nebenprodukt, die Mehrgitterfähigkeit der Datenstruktur ab.

Blockgröße

Gegenüber dem einfachen Multiblocknetz, das in der bisherigen parallelen Version durch Gebietszerlegung aus einem C-Netz entstanden ist, werden die Multiblocknetze von komplexen Gittergeneratoren erzeugt, wie wir im vorigen Kapitel schon besprochen haben. Damit werden aber die verschiedenen Blöcke in einem Multiblocknetz nicht notwendigerweise gleich groß. Es treten vielmehr erhebliche Unterschiede in der Blockgröße auf. In einem der Netze für die X-38 zum Beispiel enthält der größte Block etwa 250000 und der kleinste etwa 9000 Gitterzellen. Auch die Zahl der Blöcke des erzeugten Multiblocknetzes korreliert im allgemeinen in keiner Weise mit der Anzahl der für die Berechnung zur Verfügung stehenden Prozessoren. Es sei denn man startet das parallele Programm genau mit einer vorher entsprechend (manuell oder automatisch) bestimmten Anzahl von Prozessoren. Genau das wollen wir hier aber nicht annehmen. Das parallele Programm soll mit einer Anzahl von Prozessen gestartet werden, die von der Topologie und Art des Problem gänzlich unabhängig ist und nur durch äußere Umstände wie Maschinenbelegung und Leistungsfähigkeit des Rechensystems sowie der Problemgröße insgesamt bestimmt wird. Oder anders ausgedrückt: die Anzahl der verwendeten Prozessoren soll nur von der Anzahl der zur Verfügung stehenden Prozessoren und der gewünschten Laufzeit bestimmt werden.

Indexrichtungen der Blöcke

In einem strukturierten Gitter werden die Gitterzellen einfach durch Indizierung in den drei Raumrichtungen bezeichnet und auch so abgespeichert. Parameter und Ergebniswerte der Zellen werden jeweils in Felder abgelegt, in denen sie durch die jeweilige Position im Datenfeld genau der jeweiligen Zelle zugeordnet sind. Folglich ist auch der Zugriff auf die Parameter der Nachbarzellen einfach, da auch deren Position im Netz bekannt ist und sich die jeweiligen Werte in einem

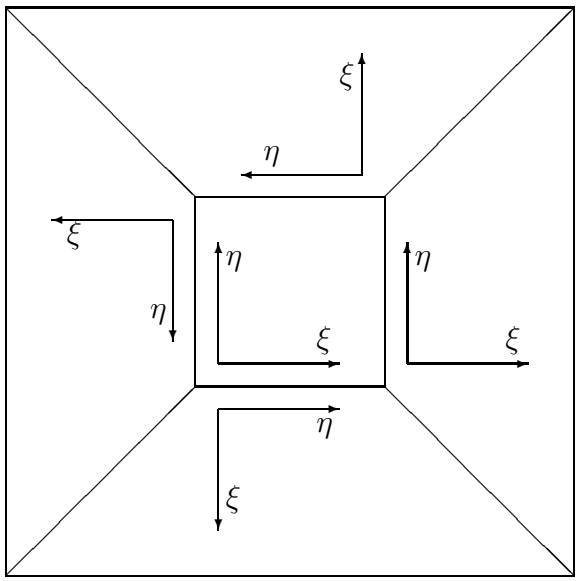

Abbildung 3.2: Das Auftreten der Indizierungsrichtungen in den unterschiedlichen Blöcken eines Multiblocknetzes

jeweils fest definierten Abstand zur Position des Werts der jeweiligen Zelle befinden. Sollte die Zelle am Rand des Rechengebiets liegen, so liegt sie auch am Rand des Feldes und ein Nachbarwert existiert dann in der entsprechenden Richtung nicht. Diese Indizierung ist der Hauptgrund für eine effiziente Behandlung dieser Netzart auf vektorbasierten Rechnern, da die Speicherzugriffe immer linear erfolgen können. Auch auf cachebasierten Systemen ist diese Netzklasse gegenüber den unstrukturierten Netzen im Vorteil, weil eine Wiederbenutzung des Caches und damit eine hohe Effizienz bei der Berechnung leichter zu erreichen ist, als bei unstrukturierten Netzen. Bei unstrukturierten Netzen ist nicht allein durch die Position eines Parameters ersichtlich, wer die Nachbarzelle ist, geschweige denn, wo im Speicher ihre Daten liegen.

In einem Multiblocknetz können die strukturierten Netzblöcke unstrukturiert angeordnet sein. Das bedeutet erst einmal, daß die Speicherung der Werte innerhalb eines Blocks wie bei einem strukturierten Netz erfolgen kann. Interessant ist hingegen der Übergang zwischen benachbarten Netzblöcken. Da diese beliebig auf eine unstrukturierte Art und Weise angeordnet sein können, kann die interne Indizierung erst einmal beliebig sein. Nun könnte man erwarten, daß man dies durch eine Umindizierung inklusive einer entsprechenden Umspeicherung aller Parameter reparieren und alle Blöcke somit gleich indizieren könnte. Dem ist aber leider nicht so, da die Netzblöcke untereinander so angeordnet sein können, daß die Abhängigkeiten eine Umindizierung nicht erlauben. Am einfachsten wird dies in Abbildung 3.2 klar, die in einem einfachen, exemplarischen Fall aufzeigt, warum eine gleiche Indizierung der Nachbarblöcke auch nicht durch Umindizie-

rung und Umspeicherung erzwungen werden kann. Nehmen wir im Beispiel die Indexrichtungen (ξ, η) des rechten Blocks als Ausgangspunkt. Die Nachbarn dieses Blocks sollen alle diesselbe Indizierungsrichtung besitzen wie er selbst. Dies ist auch erfüllt, wenn man die Lage der Schnittstelle zu den Nachbarn berücksichtigt. Somit haben dann aber der Mittelblock und sein oberer Nachbar ein unterschiedliches Indexkoordinatensystem. Also können nicht alle Nachbarblöcke eines beliebigen Blocks zwingend dieselbe Indizierungsrichtung besitzen.

Physikalische Ränder

Ein weiterer wichtiger Punkt ist die Behandlung der Zellen an den Aussenseiten des Netzes, also an den Seiten, an denen die Zellen mit speziellen Randbedingungen aus den physikalischen Gegebenheiten behandelt werden müssen. Dazu gehören Festkörperränder, Zu- und Abströmränder sowie Symmetrieränder. In der Einblockversion von URANUS sind diese Ränder fest an einen speziellen Index gebunden. So befindet sich zum Beispiel bei j=max immer ein Zuströmrand. Bei einem Multiblocknetz kann dies nicht mehr garantiert werden. Erstens kann der Netzgenerator die Blöcke (fast) frei mit den Indizes für die Richtungen versehen. Somit könnte man zwar durch Drehen der Blöcke für einen der Ränder diese Bedingung garantieren, aber bei Blöcken, bei denen drei physikalische Ränder aneinanderstoßen, also quasi an den "Ecken" des Netzes, muß dieses Verfahren im allgemeinen scheitern. Desweiteren kann es in Multiblocknetzen vorkommen, daß an zwei Blockseiten eines Blockes zwei gleichartige Ränder auftreten, also zum Beispiel zwei Blockseiten demselben Zuströmränd zugeordnet sind. Auch solche Netze könnten dann nicht behandelt werden. Ein Multiblockverfahren für allgemeine Multiblocknetze muß also die Behandlung von physikalischen Rändern an allen sechs Blockseiten unabhängig von der Indizerung des Blockes erlauben.

Nachbarschaftsbeziehungen

Die letzte wichtige Eigenschaft von Multiblocknetzen betrifft die Nachbarschaftsbeziehungen der Blöcke untereinander. Wird ein Einblocknetz bei der Gebietszerlegung für ein paralleles Verfahren zerlegt, sorgt man der Einfachheit halber dafür, daß dies regelmäßig geschieht, sich also an jeder der sechs Seiten des neuen Blocks höchstens ein Nachbarblock befindet. Bei einem allgemeinen Multiblocknetz ist dies aber nicht zwingend. Hier kann ein Block an jeder seiner sechs Seiten auch mehrere Nachbarn vorfinden, mit denen er dann während der Berechnung Daten austauschen muß. Dadurch ist es wesentlich einfacher, komplizierte Topologien zu vernetzen. Müßte man garantieren, daß es nur einen Nachbarn auf jeder Blockseite gibt, würde dies zu einer sehr starken Zerlegung des Netzes in Blöcke und zu einer großen Blockanzahl führen. Denn jede Blockgrenze, die aus topologischen Gründen notwendig ist, müßte durch das ganze Netz als Blockgrenze geführt werden, was dann auch zur Zerlegung von Blöcken führt, die ansonsten

nicht zerlegt werden müßten. Diese Blöcke können, falls es während der Programmausführung überhaupt notwendig ist, vom Gebieteszerleger des parallelen Programmes zerlegt werden. Wie wir später sehen werden, kann der Gebietszerleger dabei den Block in einer Art und Weise zerlegen, die für den Programmlauf einen Vorteil bietet.

3.2.2 Bedeutung der Eigenschaften für das parallele Programm

Blockgrößen

Die möglichen unterschiedlichen Blockgrößen haben für das parallele Programm in zweierlei Hinsicht Konsequenzen. Um mit dem parallelen Programm einen Parallelrechner möglichst effizient zu nutzen, ist eine gleichmäßige Auslastung der verwendeten Prozessoren wünschenswert. Wird die Auslastung zu unausgewogen, kann die parallele Berechnung relativ gesehen sehr schnell extrem teuer werden, wenn viele Prozesse warten, bis auch der letzte Prozeß seine Berechnungen beendet hat. Hier muß es also eine Möglichkeit geben, die Last gleichmäßig zu verteilen. Das bedeutet aber, daß ein Lastverteiler verwendet werden muß, der eine möglichst optimale Verteilung bestimmt.

Damit kann es aber vorkommen, daß Blöcke zu groß werden, um auf einem Prozeß berechnet zu werden. Diese Blöcke müssen also entsprechend oft geteilt werden, bis die Teile klein genug sind, daß sie nicht zu einer Überlast auf einem Prozessor führen. Es muß also ein Zerlegungsalgorithmus entwickelt und in das Programm integriert werden.

Weiterhin können Blöcke viel zu klein sein, um einen Prozessor wirklich auszulasten. Hier ist also dafür zu sorgen, daß ein Prozessor nicht nur einen sondern mehrere Blöcke gleichzeitig berechnen kann. Dies wiederum bedeutet einen kompletten Umbau der Datenstrukturen im Multiblockprogramm verglichen mit der parallelen Programmversion für Einblocknetze.

Indexrichtungen

Die unterschiedlichen Indexrichtungen auf den einzelnen Blöcken führen dazu, daß das Multiblockprogramm nicht davon ausgehen kann, daß zwei Nachbarblöcke in derselben Art- und Weise im Speicher abgelegt sind. Bei einem Datenaustausch zwischen zwei Blöcken können damit also nicht einfach Speicherinhalte kopiert werden, sondern es ist in Abhängigkeit von den unterschiedlichen Richtungen der Indizes auf den beiden Blöcken eine Umsortierung der Speicherinhalte vorzunehmen, damit der Datenaustausch zwischen den Blöcken klappt und jede Randzelle an den inneren Rändern auf die richtigen Werte ihrer Nachbarzelle zugreifen kann.

Physikalische Ränder

Das mögliche Auftreten von physikalischen Rändern an allen sechs Blockseiten, erfordert, daß dies auch vom Programm behandelt werden kann. Dazu sind die Randbedingungen statt bisher für eine, nun für alle sechs Seiten zu berechnen, und das mit den jeweils richtigen Indizes. Eine sechsfache Programmierung der Randroutinen, also einmal für jede Seite, ist aufwendig, sowohl in der Programmierung als auch später in der Programmwartung oder bei Erweiterungen. Außerdem führen die Programmverzweigungen, die notwendig sind, um zu entscheiden, welche Seite nun wie zu behandeln ist, auf den modernen Architekturen grundsätzlich zu einem unnötigen Leistungsverlust. Durch eine geschickte Programmierung, in dem für alle Seiten eines physikalischen Typs, zum Beispiel für den Festkörperrand, dasselbe Unterprogramm genutzt wird, kann dies vermieden werden. Dabei wird im entsprechenden Unterprogramm jeweils eine Schleife über alle entsprechenden Ränder durchlaufen. Dazu müssen jedoch die Randparameter so abgespeichert werden, daß eine Behandlung in einer Schleife ohne eine Verzweigung möglich wird.

Nachbarschaftsbeziehungen

Die komplizierteren Nachbarschaftsbedingungen, die in Multiblocknetzen möglich sind, erfordern gegenüber dem parallelen Einblockverfahren einen kompletten Umbau der Programmteile für die Kommunikation. Die bisherige quasi strukturierte Aufteilung des Block in seine Unterblöcke für deren parallele Abarbeitung sorgte dafür, daß der Prozess des Nachbarblocks implizit bekannt war, da er anhand der Richtung in der der Nachbarblock liegt und aus der eigenen Prozeßnummer bestimmt werden konnte. Dies ist nun nicht mehr möglich. Die Blöcke werden jetzt von einem beliebigen Prozeß bearbeitet, insbedondere kann aufgrund der unstrukturieretn Anordnung der Blöcke nicht mehr vom eigenen Prozess auf den Prozess des Nachbarblocks geschlossen werden. Damit muß ein Verfahren entwickelt werden, mit dem bestimmt werden kann, auf welchem Prozeß der Nachbarblock liegt. Weiterhin muß bei der Kommunikation über die Randfläche dafür gesorgt werden, daß unterschiedliche Teile der Randfläche mit unterschiedlichen Nachbarn ausgetauscht also verschickt und auch empfangen werden können. Eigentlich sollte es für die Anzahl der Nachbarn pro Randfläche keine zahlenmäßige Begrenzung geben.

3.3 Programmentwurf

Im Programmentwurf werden die spezifizierten Eigenschaften des parallelen Multiblockprogramms in Datenstrukturen, Abläufe und eine Implementierung umgesetzt. Als grundlegend ist hier die Entwicklung tragfähiger Datenstrukturen zu sehen, da sie das Fundament des gesamten Programms sind. Sie müssen daher

nicht nur den Anforderungen der derzeit geplanten Implementierung gerecht werden, sondern bereits die Bedürfnisse zukünftiger Erweiterungen berücksichtigen.

3.3.1 Datenstrukturen für die Blockverarbeitung

Wie in der Konzeption beschrieben, kann es vorkommen, daß auf einem Prozeß mehrere Blöcke bearbeitet werden müssen. Daher muß die Datenstruktur in der Lage sein, mehrere Blöcke pro Prozeß zu verwalten. Bei den bisherigen Multiblockprogrammen wurde dazu eine der folgende Strategien angewandt[56]:

- Es wird für jeden Block der gleiche Speicherplatz reserviert. Dieser Speicherplatz muß sich am größten Block orientieren, damit auch dieser abgelegt werden kann. Das bedeutet, daß die kleineren Blöcke den ihnen zur Verfügung gestellten Speicherplatz nicht ausnutzen und daher ein erheblicher Speichermehrverbrauch, verglichen mit dem tatsächlich benötigten, auftritt.

- Die Blöcke werden nacheinander im Speicher abgelegt. Dies erforderte aber unter FORTRAN77 die Entwicklung eines eigenen Speichermanagements, das diese Datenspeicherung verwaltet und die Zugriffe auf die jeweils richtigen Blöcke organisiert. Weiterhin hängt der Speicherplatzbedarf auf einem Prozeß immer noch von der Anzahl der verwendeten Prozessoren für den Programmlauf und von der Problemgröße ab, erfordert also entweder die statische Belegung des gesamten verfügbaren Speichers pro Prozessor oder ein Neuübersetzen des Programms für jede Problemgröße und jede Prozessorzahl. Die erste Lösung ist nur auf Maschinen mit verteiltem Hauptspeicher ratsam, auf denen der Speicher von anderen Prozessoren nicht auch genutzt werden kann, er also sonst ungenutzt bliebe. Sobald jedoch gemeinsamer Speicher verfügbar ist, kann der freie Speicher von anderen Prozessen genutzt werden. Damit wäre also jeweils eine Neuübersetzung des Programms pro Problemgröße und eingesetzter Prozessorzahl notwendig.

- Implementierung der Speicherroutinen in C. In C war es, im Gegensatz zu Fortran, von Anfang an möglich Speicher dynamisch zu verwalten. Das Problem an dieser Lösung ist, daß die C-Programmteile von Fortran aus aufgerufen werden müssen, was zwar auf allen Plattformen möglich ist, aber je nach Plattform unterschiedlich zu bewerkstelligen ist. Solch eine Lösung schränkt die Protabilität erheblich ein und setzt für die Portierung ein umfassendes Wissen über die Compiler und Linker auf den jeweiligen Plattformen voraus.

Mit der dynamischen Speicherverwaltung von Fortran90 ist es inzwischen aber möglich, eine saubere, speichereffiziente, reine Fortranlösung zu entwickeln, ohne als Programmierer das Speichermanagement selbst implementieren zu müssen. Daher viel die Entscheidung zugunsten einer reinen Fortranlösung in Fortran90.

Blockliste

Für die Speicherung der Blockdaten auf den jeweiligen Prozessen wird im vorliegenden Programm eine doppelt verkettete Liste verwendet. Ein Listenelement enthält dabei sämtliche Informationen und Variablen, die der Prozeß für die Bearbeitung eines Blocks benötigt. Dazu zählen neben Identnummer und Informationen über die Blocknachbarn, die Informationen über die Blockseiten, die Felder zur Aufstellung der Gleichungssysteme für den Block, verschiedene Puffer und Handles für den Datenaustausch, die Verwaltungsinformation und natürlich die gesamte Netzhierarchie des Blocks.

Ein Block enthält im Moment zwar nur ein Netz. Wie oben beschrieben ist es aber äußerst sinnvoll, bereits die geplanten Erweiterungen wie Netzverfeinerung und Multilevelverfahren zu berücksichtigen. Dazu enthält der Block Zeiger auf das Basisnetz, also das Originalnetz, so wie es eingelesen wurde, und das Netz, das vom Block gerade verwendet wird. Der Zugang zu den anderen Netzen des Blocks ist nicht direkt möglich sondern erfolgt über die Netzhierarchie.

Bei Multilevel- und Multigridverfahren wird das Netz gleichmäßig gröber gemacht, in dem jeweils acht Zellen, also zwei in jeder Dimension, zusammengefaßt werden. Bei der Verfeinerung oder auch Vergröberung durch den adaptiven Algorithmus, darf in URANUS im Gegensatz zu den bisherigen Verfahren aber auch jeweils nur eine Richtung in der Feinheit verändert werden. Aus dem so entstehenden Netz müssen, falls ein Mehrgitteralgorithmus verwendet wird, auch noch gleich die entsprechenden gröberen Netze, wie eben beschrieben, erzeugt werden.

Um die Netze klassifizieren zu können wurde das Netzlevel eingeführt. Da die Feinheit der Netze in jede der drei Indexrichtungen in URANUS unterschiedlich sein kann, muß für jede Indexrichtung ein eigener sogenannter Richtungslevel angegeben werden. Dieser bezeichnet die Feinheit des Netzes in der jeweiligen Richtung. Dabei bekommt das Basisnetz in jeder Richtung den Level null zugeordnet. Wird das Netz in einer Richtung durch Teilen einer Zelle in zwei Teile feiner gemacht, steigt der Richtungslevel um eins, wird vergröbert, sinkt er entsprechend. Dadurch ist jedes Netz, ausgehend vom Basisnetz, durch seine drei Richtungslevel bestimmt. Aus den drei Richtungsleveln wird nun ein Gesamtlevel gebildet, das sich einfach aus der Summe der drei Richtungslevel errechnet. Folglich identifiziert dieses Gesamtlevel ein Netz nicht mehr eindeutig. Es eignet sich aber sehr gut, um als Hashwert für die Speicherung der Netze herangezogen zu werden.

Die Netzhierarchie ist nun in zwei einfach verketteten Listen gespeichert. Eine für Netze mit positivem und eine für Netze mit negativem Gesamtlevel. Ein Element dieser Listen enthält neben dem Gesamtlevel nur einen Zeiger auf die Liste der Netze mit diesem Level. Damit stellt das Gesamtlevel für den Zugriff ein äußerst einfaches Hashverfahren dar. Die Netze selbst, genauer gesagt ein Listenelement Netz, enthält dann alle notwendigen Daten eines Netzes. Also seine gesamte geometrische Beschreibung mit allen für die Strömungsberechnung not-

wendigen Größen wie Zelllängen und Ableitungen. Auch der letzte berechnete Erhaltungsvektor auf dem Netz ist hier mit abgelegt. Außerdem enthält das Netz noch zwei Zeiger auf das nächst feinere und das nächst gröbere Netz, das durch Verfeinerung, bzw. Vergröberung in allen Richtungen aus diesem Netz entstanden ist. Dadurch ist ein schneller Zugriff auf die Nachbarnetze für Mehrgitter und Multilevelverfahren sichergestellt.

Die vorgestellte Datenstruktur wird allen bisher denkbaren Erweiterungen für die Multiblocknetze gerecht. Außerdem verwendet sie nur den Speicher, der wirklich für die Bearbeitung des Blocks benötigt wird. Andererseits ist der Aufwand zum Auffinden eines bestimmten Netzes nicht überproportional hoch. Leistungseinbußen durch die verwendete Listenstruktur sind nicht zu erwarten. Zwar können die einzelnen Listenelemente quer im Speicher verstreut liegen, aber die Anzahl der Operationen, die auf einem strukturierten Listenelement mit linearen Speicherzugriffen durchgeführt werden können ist erheblich, verglichen mit den gelegentlich notwendigen Umschaltungen und den damit verbundenen Sprüngen auf einen anderen Block und damit wahlfrei in einen anderen Speicherbereich. Die Umschaltungen selbst erfordern auch nicht mehr Aufwand als in den anderen vorgestellten Verfahren.

Die vorgestellten verwendeten Datenstrukturen sind also sowohl für Vektorrechner als auch für cachebasierte Systeme optimal.

Programmablauf

Diese neue Datenstruktur führt natürlich auch zu Änderungen im Programmablauf, denn alle Programmteile und Unterprogramme, die bisher nur auf einem einzigen Block gearbeitet haben, müssen nun für mehrere Blöcke verwendbar sein. Dabei ist grundsätzlich zwischen zwei Situationen zu unterscheiden. Programmteile, die nur auf einem Block an sich ablaufen, wie zum Beispiel die Berechnung der Ableitungen oder das gesamte Aufstellen des lokalen Gleichungssystems, laufen ab wie bisher, nur daß sie eben mehrere Male aufgerufen werden, für jeden Block einmal. Dabei können manche Unterprogrammaufrufe sogar zusammengefaßt werden, so daß eine Folge von Unterprogrammen auf einem Block ablaufen kann, bevor zum nächsten Block gewechselt wird.

Unterprogramme, die neben dem Block auch seine Beziehung mit Nachbarn beeinflussen, in denen also kommuniziert wird, müssen dagegen so umgearbeitet werden, daß sie mit der Blockliste zurecht kommen. In diesen Unterprogrammen muß dann auch mehrmals zwischen den Blöcken umgeschaltet werden, da zum Beispiel alle Blöcke eines Prozesses quasi gleichzeitig kommunizieren können müssen. Vorteil dieses Umstandes ist aber, daß auch lokale Blöcke miteinander durch Nachrichtenaustausch kommunizieren können. Das vereinfacht die Handhabung der lokalen Blöcke erheblich, da zwischen lokalen und entfernten Nachbarn kein Unterschied gemacht werden muß, vielmehr spielt die tatsächliche Position des Nachbarn, also der Prozeß auf dem er liegt, für die Berechnung und auch die

Implementierung keine weitere Rolle. Nur bei der Kommunikationsgeschwindigkeit kann es natürlich Unterschiede geben, sofern das unterliegende Kommunikationssystem nicht für lokale Nachrichten optimiert ist. Die mehrfache Umschaltung zwischen den Blöcken fällt bei der Kommunikation nicht ins Gewicht, da schon die Latenz durch das Message Passing System erheblich größer ist, als der Aufwand, auf einen anderen Block zu schalten.

3.3.2 Datenstrukturen für die Randbehandlung

Die Behandlung der Ränder spielt in jedem gitterbasierten Verfahren eine große Rolle. Hier werden spezielle Datenstrukturen eingeführt, um den bereits beschriebenen Gegebenheiten an den Rändern in Multiblocknetzen Rechnung zu tragen und Ränder effizient bearbeiten zu können. An jedem der sechs Ränder eines Blocks liegen entweder ein, oder mehrere physikalische Ränder oder Blocknachbarn vor. Zur Speicherung der Situation an den Rändern existieren in jedem Blocklistenelement sechs Vektoren, für jede Seite einer. In diesen Vektoren wird beim Einlesen des Netzes und der Verbindungsdaten eines Blockes für jedes seiner Seitenflächenteile abgespeichert, was für eine Situation dort vorliegt.

Neben der genauen Seitenteilflächenbeschreibung in Indizes der Blockoberfläche, wird bei einem physikalischen Rand abgelegt, welcher Art und welchen Untertyps er ist. Der Untertyp wurde eingeführt, damit die Randtypen weiter unterschieden werden können. Damit ist es zum Beispiel möglich, nicht nur einen Zuströmrand zu definieren, sondern mehrere Zuströmränder, die sich in den Randbedingungen, zum Beispiel der Zuströmgeschwindigkeit unterscheiden. Im verwendeten URANUS-Verfahren wird diese Untertypisierung zukünftig am Festkörperrand zur Unterscheidung verschiedener Oberflächenmaterialien eingesetzt werden. Das ist notwendig, da die Katalyseeigenschaften der Oberfläche je nach Material unterschiedlich sind. Diese beeinflussen aber die ablaufenden chemischen Reaktionen und letztlich die Strömung und die Wärmelast auf der Oberfläche.

Liegt an der Seitenteilfläche ein Nachbar, so wird neben der Blocknummer des Nachbarn und seiner Orientierung im Indexraum auch die Nummer der Seitenfläche des Nachbarn abgelegt, an den die Blockseitenteilfläche grenzt. Dies ist notwendig, da aus der eigenen Seitenflächenausrichtung, also an welcher der sechs Blockseiten die Seitenteilfläche liegt, und der Orientierung des Nachbarblocks nicht sicher auf die richtige Seitenteilfläche des Nachbarblocks geschlossen werden kann. Denn schließlich können zwei verschiedene Seitenteilflächen einer Blockseite eines Blocks an zwei ebenfalls verschiedene Seitenteilflächen derselben Seite eines anderen Blocks grenzen. Zur Verdeutlichung ist dies in Abbildung 3.3 an einem Beispiel gezeigt.

Die beschriebenen Vektoren mit den jeweiligen Seiteninformationen können nun auch bei einer Blockverfeinerung oder bei einem Vergröberungsschritt leicht auf die neue Situation umgerechnet werden.

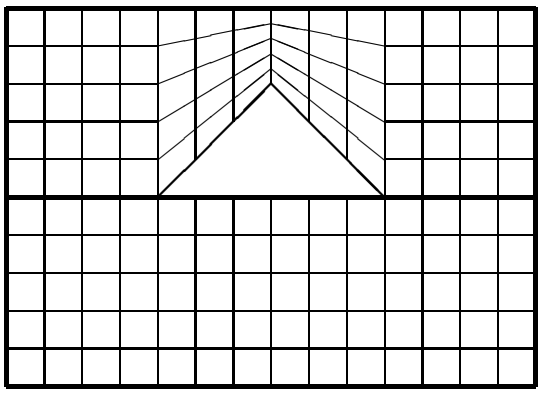

Abbildung 3.3: Beispielhafte Darstellung der Notwendigkeit der Seitenteilnumerierung

Aus diesen Vektoren werden jeweils nach Festlegen des Netzes neue Vektoren berechnet. Dabei wird für jede Seite ein Vektor berechnet, der die für die Kommunikation mit allen Nachbarn dieser Seite notwendigen Informationen enthält (vgl. Kapitel 3.3.4). Weiterhin wird für jeden der im Moment auftretenden vier Randtypen ein Vektor gebildet, der alle Seitenteilflächen des Blocks enthält, die an einen entsprechenden physikalischen Rand grenzen. Mit Hilfe dieser Vektoren ist dann eine effiziente Behandlung der jeweiligen Ränder möglich, da alle notwendigen Informationen zusammen abgelegt sind.

3.3.3 Behandlung physikalischer Ränder

Die Spezifikation sieht für das Multiblockverfahren eine kompakte Behandlung der physikalischen Ränder in einem Unterprogramm pro Randtyp vor. So wird ohne aufwendige Verzweigungen eine möglichst effiziente Behandlung der physikalischen Ränder erreicht. Voraussetzung dafür ist, daß alle Informationen über alle Ränder eines Randtyps eines Blocks gemeinsam so abgelegt werden, daß ohne Verzweigung in einer Schleife über diese Informationen die Randbehandlung in den jeweiligen spezifischen Unterprogrammen erfolgen kann. Daher wurden die bereits erwähnten Randvektoren für die physikalischen Ränder entwickelt, die genau diese Eigenschaft besitzen. Sie enthalten alle Informationen über den jeweiligen Rand wie Seitenfläche, Lage des Rands dort und auch seinen Untertyp.

Die Unterprogramme zur Randbehandlung wurden so umgearbeitet, daß sie die Behandlung des Rands unabhängig von seiner Indexposition auf allen sechs Seiten eines Blocks erlauben. Die dazu notwendigen Indexinformationen sind jeweils im Randvektor verfügbar.

Einzige Ausnahme ist die inzwischen von den Ingenieuren entwickelte äußerst komplexe Behandlung der verschiedenen katalytischen Festkörperränder, die als einzige im Moment nur bei Index j=0 erfolgen kann. Dies bedeutet aber in der Realität der Anwendung keine Einschränkung, da dies der einzige so geartete

Rand ist. Damit kann die Indexrichtung der Randblöcke aber jeweils so gedreht werden, daß sich der Festkörperrand bei j=0 befindet. Einziges Problem wären Netzblöcke, die auf zwei verschiedenen Seiten einen Festkörperrand besitzen. Dies läßt sich aber durch eine angepaßte Netzgenerierung gut umgehen. Außerdem besaßen bisher alle Netze, die von anderen Institutionen für Wiedereintrittsrechnungen mit URANUS zur Verfügung gestellt oder vorgegeben wurden, ebenso diese Eigenschaft.

Mit Hilfe dieser Vektoren die alle Informationen über den jeweiligen Rand wie Seitenfläche, Lage des Rands dort und auch seinen Untertyp enthalten, konnte die Randbehandlung relativ kompakt gestaltet werden.

3.3.4 Innere Ränder und Kommunikation

Die Seitenränder eines Blocks, an denen ein anderer Block angrenzt nennt man auch "innere Ränder". Für die Berechnung der Werte in Zellen, die an ein einem solchen inneren Rand liegen, werden also die Werte der auf einem Nachbarn liegenden Nachbarzellen benötigt. Diese müssen vom Nachbarblock beschafft, also auf einem Parallelrechner per Kommunikation verschickt werden. Um nun möglichst selten Daten zwischen benachbarten Blöcken austauschen zu müssen, besitzt jeder Block eine zwei Zellen tiefe Schicht aus sogenannten Halozellen. Diese Zellen gehören eigentlich zu einem Nachbarblock. Ihre Daten werden aber nochmals auf dem lokalen Block als Kopie gehalten. Damit müssen Daten zwischen den Nachbarn nur ausgetauscht werden, wenn diese sich ändern. Dies ist in URANUS nur bei der Gleichungslösung und nach der jeweiligen Neuberechnung des Erhaltungsvektors der Fall. Zwar erfordert diese Lösung einen etwas höheren Berechnungsaufwand auf dem lokalen Prozess, dies ist aber gegenüber einer mehrmaligen Kommunikation bei weitem die effizientere Lösung. Zum Beispiel wurden selbst auf einem Prozessor einer Cray T3E etwa 13.000 Operationen durchgeführt, bis auch nur das erste Byte einer verschickten Nachricht auf dem Nachbarprozessor angekommen ist. Bei heutigen Rechnern liegt dieses Verhältnis leider noch weitaus ungünstiger für die Kommunikation. Auch der geringe Speichermehrbedarf dieser Lösung rechtfertig nicht einen Umstieg auf eine Strategie, die statt der Kopie den Wert immer durch Kommunikation zur Verfügung stellt.

Behandlung der Indexrichtungen

Diese Halozellen besitzen jeweils die Indexrichtungen und damit die Orientierung des Blocks, an den sie angegliedert sind, unabhängig von der originalen Ausrichtung der Zellen auf ihrem Heimatblock. Damit kann während der Berechnung direkt auf die Werte dieser Zellen zugegriffen werden, ohne auf Speicherstrukturen Rücksicht nehmen zu müssen. Für die jeweilige Anpassung der Indexrichtungen zwischen den Blöcken wird dann beim Datenaustausch durch Umsortieren gesorgt. Grenzen nach Implementierung der lokalen Verfeinerung zwei Blöcke mit

unterschiedlicher Auflösung, das heißt unterschiedlich groben Zellen in den entsprechenden Richtungen, aneinander, kann das Problem der Interpolation an den Blockschnittstellen in gleicher Art und Weise gelöst werden. Die Auflösung der Halozellen orientiert sich dabei ebenso an der des Blockes an den sie angegliedert sind. Die tatsächliche Prolongation oder Restriktion findet dann beim Datenaustausch statt.

Um die beschriebenen Transformationen während des Datenaustauschs vornehmen zu können, werden die entsprechenden Informationen über den Nachbarblock benötigt. Dazu enthält der in Abschnitt 3.3.2 beschriebene Vektor neben der lokalen Position und der Identität der angrenzenden Nachbarseiten und des Nachbarblocks auch die Information über die Orientierung und bereits auch die Auflösung des Nachbarblocks.

3.3.5 Organisation der Blockstruktur auf einem Parallelrechner

In vielen Teilen der letzten Abschnitte wurde bereits das Thema Kommunikation angeschnitten. Sie wird durch die Verwendung eines Parallelrechners mit verteiltem Speicher notwendig, da hier bei dem verwendeten Programmiermodell Informationen zwischen den Blöcken auf den einzelnen Prozessoren nur über Nachrichten ausgetauscht werden können. Nachdem wir im vorigen Unterkapitel bereits die Datenstruktur für die inneren Ränder, an denen Kommunikation notwendig ist, vorgestellt haben, wollen wir uns hier mit dem genauen Entwurf der dazu notwendigen Programmteile beschäftigen.

Die Kommunikation stellt das einzige Verbindungsglied zwischen den einzelnen Blöcken während des Programmlaufs dar. Um einen Informationsfluß zwischen den Blöcken zu ermöglichen, müssen die einzelnen Blöcke die empfangenen Daten auch interpretieren können. Wie bei der Kommunikation zwischen Rechnern und Anwendungen über externe Netzwerke, für die entsprechende Protokolle existieren, ist es also sinnvoll auch bei der Kommunikation in einer parallelen Anwendung, diese Protokolle vor dem Simulationsteil zu verbergen. Das bedeutet, daß sich die Kommunikationsebene um das Protokoll und die Darstellung der Informationen kümmert, so daß sie der Simulationsteil direkt verarbeiten kann. Dazu werden während der Kommunikation an den ausgetauschten Daten verschiedene Änderungen und Anpassungen vorgenommen:

- Änderung der Speicherreihenfolge, falls die Nachbarblöcke, zwischen denen Daten ausgetauscht werden, unterschiedliche Indexrichtungen besitzen. Die Informationen über die Indexrichtungen des jeweiligen Nachbarn liegen im Datenvektor für die Seite vor.

- Interpolation bzw. Prolongation, falls die Auflösung der Netzzellen auf Sender- und Empfängerseite unterschiedlich ist.

Um einen möglichst hohen Durchsatz zu erreichen und die Berechnung so wenig wie möglich zu stören wurde auf globale Synchronisationspunkte soweit möglich verzichtet. Nur zur Berechnung der Residuen ist noch eine kollektive Kommunikation notwendig. Während aller anderen Kommunikationsschritte hängt die Möglichkeit zum Datenaustausch nur vom jeweiligen Nachbarn ab. Der Synchronisationspunkt durch die kollektive Kommunikation hätte vermieden werden können, wenn MPI die Möglichkeit der asynchronen kollektiven Kommunikation bieten würde. Die Programmstruktur von URANUS hätte die Verwendung solcher Konstrukte direkt ermöglicht. Leider hat die asynchrone kollektive Kommunikation den Weg in MPI 2.0 nicht gefunden und wird bis auf weiteres auch nicht zur Verfügung stehen.

Beim Entwurf wird davon ausgegangen, daß alle Prozesse in etwa gleich ausgelastet sind und daher die Routinen für die Kommunikation mehr oder minder gleichzeitig aufrufen. Es wird also nicht versucht, Berechnung und Kommunikation zu überlappen. Im allgemeinen Fall wird vielmehr die Kommunikation asynchron durchgeführt und versucht, die einzelnen Kommunikationsschritte der Blöcke hintereinander zu verstecken. Dazu sollten die Kommunikationsroutinen für alle Blöcke gleichzeitig aufgerufen werden. Die Kommunikation zwischen Blöcken, die auf ein und demselben Prozeß liegen, findet ebenfalls mittels MPI statt. Dies hat den enormen Vorteil, daß nirgends zwischen lokalen und externen Blöcken unterschieden werden muß und wird. Ein Block enthält also nirgends direkt die Information, daß sich ein Nachbarblock auf demselben Prozeß befindet. Dies vereinfacht auch die Blockumverteilung im Falle eines Lastungleichgewichts. Aufgrund dessen müssen allerdings die Kommunikationsroutinen auf der Blockliste und nicht auf einem einzelnen Block arbeiten.

Wird eine Kommunikationsroutine von einem Prozess aufgerufen, werden zuerst für alle Nachbarn aller Blöcke eines jeden Prozessors die entsprechenden asynchronen Empfangsvorgänge gestartet, danach, wieder für alle Blöcke und alle Nachbarn, die Sendevorgänge, ebenfalls asynchron. Nun wird für den jeweils ersten Block auf das Beenden einer der offenen Empfangsvorgänge gewartet und zwar über alle Nachbarseiten. Sobald eine Nachricht empfangen wurde, wird sie bearbeitet. Mit großer Wahrscheinlichkeit läuft während dieser Zeit eine weitere Nachricht ein, so daß nicht wirklich gewartet werden muß. Sobald also der erste Nachbar des Blocks etwas geschickt hat, kann diese Nachricht bearbeitet und die möglicherweise notwendige Umspeicherung der Werte auf die lokale Indexrichtung vorgenommen werden. So läßt sich der Aufwand für die Kommunikation unter dem Aufwand für die notwendigen Verwaltungsarbeiten für die Kommunikation verstecken.

Derzeit werden die Blöcke eines Prozesses nacheinander abgearbeitet. Sollte sich zeigen, daß das Verfahren noch nicht ausreicht, und während der Kommunikation immer noch zu lange gewartet werden muß, können auch alle Blöcke gleichzeitig behandelt werden, so lange sie auf Nachrichten warten. Damit ist die Wahrscheinlichkeit noch größer, daß eine Nachricht für eine der Seiten eines

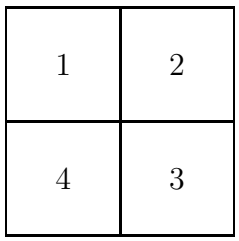

Abbildung 3.4: Einfaches 2-D Blocknetz

Blockes des Prozesses ankommt, während die gerade angekommene Nachricht bearbeitet wird. Um die Zuordnung der Nachrichten zu den Blöcken des Prozesses zu erleichtern wird der Tag der MPI-Nachricht zur Identifikation des Blocks und der Blockseite verwendet. Die Abbildung von Block und Blockseite auf das Tag ist dabei eineindeutig.

Wenn alle Nachrichten für alle Blöcke eines Prozesses angekommen und bearbeitet sind, wird noch auf die Beendigung aller Sendevorgänge gewartet, ehe aus der Kommunikationsroutine die Kontrolle an den übergeordneten Programmteil zurückgegeben wird. Abbildung 3.5 zeigt für das einfache 2-D Netz aus Abbildung 3.4 den Ablauf mehrerer Kommunikationsschritte hintereinander. Dabei werden Daten über die Ränder auf die jeweils benachbarten Blöcke übertragen. Ebenso wird in Abbildung 3.5 vergleichend der Fall gezeigt, daß die Verwaltungsarbeiten wie die Konvertierung auf der Senderseite durchgeführt werden. Der Zeitaufwand für alle Konvertierungen soll hier gleich sein, ebenso der Aufwand für die einzelnen Kommunikationsschritte.

Wie zu sehen ist, hat das Programm, das die verschickten Daten empfängerseitig bearbeitet, stets Laufzeitvorteile gegenüber seinem Pendant, das die Daten senderseitig behandelt. Der dargestellte Vorteil für die gezeigten vier Prozesse läßt sich auch einfach methematisch nachweisen. Ein allgemeiner Nachweis für beliebige Prozeßzahlen würde den Rahmen dieser Arbeit aber sprengen.

3.3.6 Lastverteilung

Um die im letzten Unterkapitel angesprochene Blockverteilung so vorzunehmen, daß der Parallelrechner möglichst effizient genutzt wird, muß die Last möglichst gleichmäßig auf die Prozessoren verteilt werden. Das bedeutet, daß die Last so zu verteilen ist, daß die Prozessoren alle zum selben Zeitpunkt miteinander kommunizieren können und ihre dazu notwendige Rechenarbeit beenden. Natürlich möglichst so, daß auch keiner auf die Kommunikation mit seinem Nachbarn warten muß.

Mit den im letzten Abschnitt beschriebenen Methoden zur Kommunikation der Blöcke untereinander und mit der Möglichkeit auch mehr als einen Block pro Prozessor zu berechnen, besitzen wir nun das Handwerkszeug, die Last auch vom Netz her gleichmäßig verteilen zu können. Die Programmteile, die jetzt noch zu

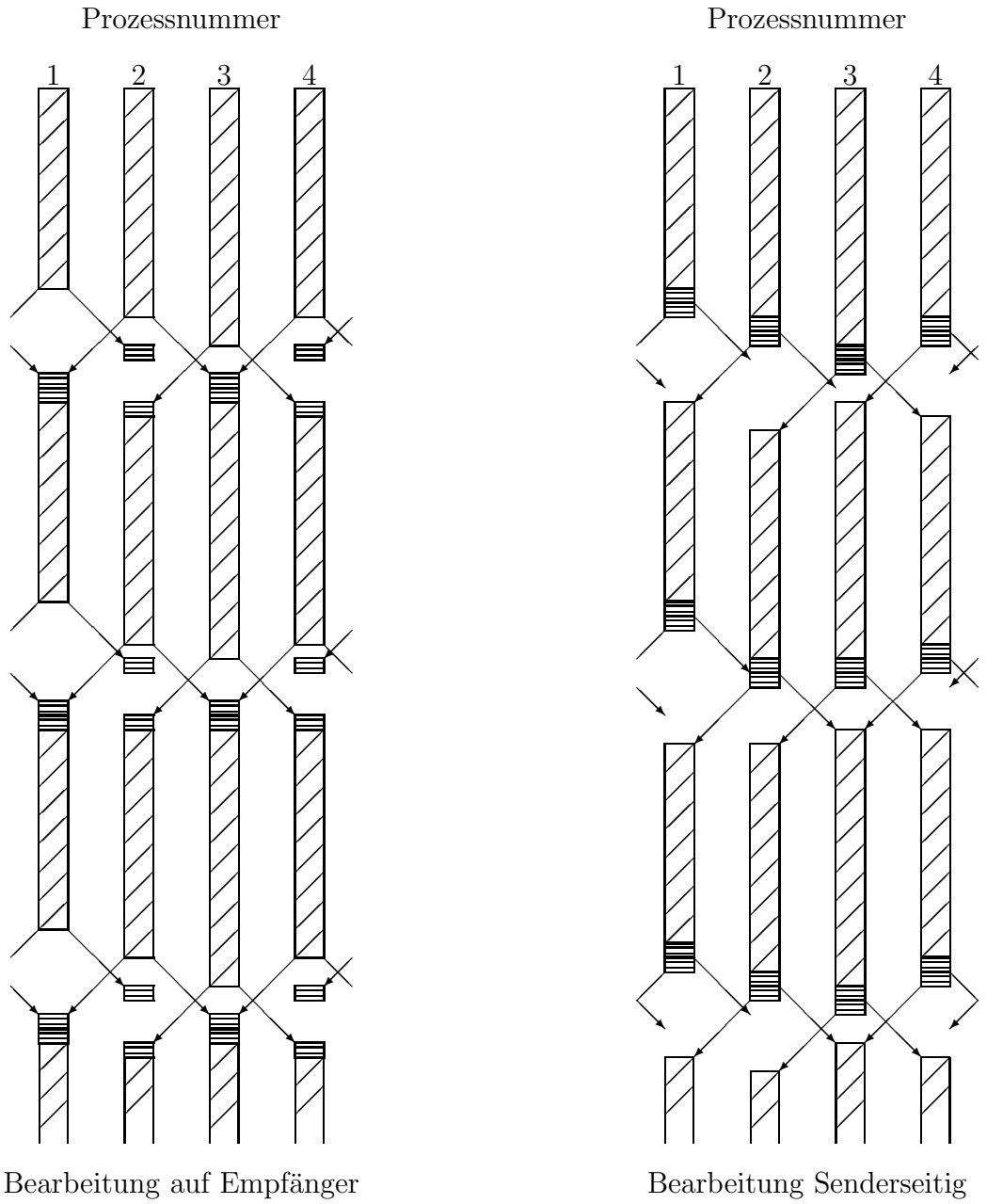

Abbildung 3.5: Vergleichende Darstellung mehrerer Kommunikationsvorgänge für ein einfaches Netz

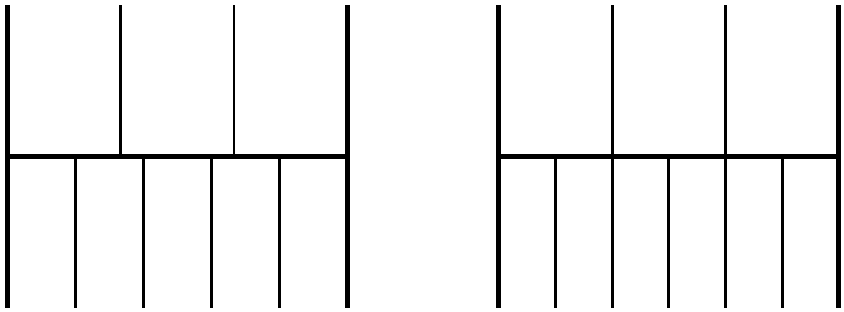

Abbildung 3.6: Vergleich der Blockschnittstellen nach Aufteilung

einer funktionierenden Lastverteilung fehlen, sind ein Algorithmus zur Aufteilung der Blöcke, die für einen Prozeß zu groß sind, und einer zur Verteilung der Last an sich. Als Verteilungsalgorithmus werden verfügbare Programmpakete eingesetzt. Zur Zeit ist eine Schnittstelle für Jostle [82] vorhanden. Daneben wurde noch ein sehr einfacher Algorithmus direkt implementiert, der die vorhandenen, eventuell schon vom Aufteilungsalgorithmus geschnittenen Blöcke reihum (Round Robin) auf die Prozessoren verteilt. Dies ist notwendig, damit das Programm auch auf Plattformen lauffähig ist, für die sonst kein Lastverteiler verfügbar oder lizensiert ist.

Blockaufteilungsalgorithmus

Der Blockaufteilungsalgorithmus ist ein ziemlich komplexer Programmteil, da er bei der Blockaufteilung verschiedenste Vorgaben berücksichtigen muß. Zuerst muß festgestellt werden, in wieviele Teile ein Block, der zur Berechnung auf einem Prozessor zu groß ist, zerteilt werden muß. Dazu wird die Zahl der pro Prozessor zu berechnenden Zellen bestimmt. Diese Zahl entsteht durch Division der Summe der Zellzahlen aller Netzblöcke durch die Prozessorzahl. Durch Teilen der Zellzahl des jeweiligen Blockes durch diese Größe erhält man den ersten Anhaltspunkt, in wieviele Teile der Block mindestens zu zerlegen ist. Wir lassen hier jedoch auch eine einstellbare Übergröße von derzeit 3% zu, um nicht auch Blöcke, die knapp zu groß sind, aufwendig zu teilen.

Diese dann notwendige Teilezahl für den Block wird weiter bearbeitet, denn als Teilerzahlen für den Block wurde hier festgelegt, nur Potenzen von zwei und drei und deren Kombinationen zuzulassen. Damit ist sichergestellt, daß regelmäßige Teilungen in allen drei Raumrichtungen möglich sind. Dies wäre zum Beispiel bei Primzahlen unmöglich. Weiterhin sind dann auch die Teilerzahlen für die Seiten Potenzen von 2 und 3 und deren Kombinationen. Da auch die Nachbarn der Blöcke in gleicher Art und Weise geteilt werden, läuft dort die Teilung der Nachbarseite ähnlich ab. Durch die ausschließliche Verwendung der Potenzen von zwei und drei ist die Wahrscheinlichkeit sehr hoch, daß bei einer Aufteilung von zwei Nachbarblöcken die Teiler der beiden Blockseiten am gemeinsamen inneren Rand

einen größten gemeinsamen Teiler haben, der größer als eins ist. Damit kann eine zu starke Zersplitterung der Schnittstellen zwischen den Blocknachbarn verhindert werden, wie sie sonst möglich wäre. Dies ist in Abbildung 3.6 nochmals verdeutlicht. Eine Zersplitterung der Blockschnittstellen würde den Aufwand für die Kommunikation unnötig erhöhen, da der Block auf dieser Seite dann mit mehr Nachbarn Daten austauschen müßte. Außerdem würde zwischen diesen Blöcken zusätzlich synchronisiert werden. Der Kommunikationsalgorithmus würde aber trotzdem auch solch eine aufgesplittete Schnittstelle richtig behandeln und abarbeiten.

Verteilen der Schnitte auf die Seiten

Die nun folgende Verteilung der notwendigen Schnitte auf die drei Seiten ist ebenfalls aufwendig, da hier einige Möglichkeiten und Sonderfälle zu berücksichtigen sind. Die Erfahrung mit der parallenen Version des 3-D Einblockverfahrens von URANUS hat gezeigt, daß es günstig ist, wenn die zu erstellenden Blöcke ein Kantenverhältnis von 2:1:1 besitzen, also die Zellzahl in der ersten Dimension doppelt so groß ist, wie in den beiden anderen. Grund dafür ist, daß die Zellen in der ersten Dimension hintereinander im Cache und Speicher liegen und so eine Abarbeitung effizient wird. Wird das Verhältnis größer, gewinnt hingegen wieder die Oberfläche, die ja mit den Nachbarn auszutauschen ist, im Verhältnis zum Volumen des Blocks an Einfluß, was wieder die Kommunikationszeiten erhöht. Die beschriebene Größe hat sich bei verschiedenen Messungen in den letzten Jahren auf den verschiedensten Plattformen als vorteilhaft erwiesen. Der Blockaufteilungsalgorithmus versucht also, möglichst solch ein Verhältnis herzustellen. Dabei muß aber nach wie vor berücksichtigt werden, wie oft der Block insgesamt geteilt werden darf und auch, daß als Teiler der Seiten nur Potenzen von zwei und drei in Frage kommen.

Daneben gibt es aber noch eine ganze Reihe weiterer Kriterien, die zum Teil auch K.O.-Kriterien für die Aufteilung sind. So muß ein Block in jeder seiner drei Dimensionen mindesten sechs Zellen besitzen. Grund dafür ist nicht nur, daß der Block sonst zu klein würde, sondern auch ein Algorithmus innerhalb der Oberflächenberechnung geht davon aus, daß er auf sechs Zellen senkrecht zur Oberfläche direkt zugreifen kann. Da bei der Aufteilung über die Randbedingungen nichts bekannt ist, war dies eine einfache und elegante Lösung. Sollte durch die vorhin gewählte Teilung nun ein Block in einer Richtung weniger als die notwendige Zellzahl besitzen, müssen die Schnitte entsprechend auf die anderen Seiten umgelegt werden. Und zwar so, daß das Seitenverhältnis noch so gut es geht gewahrt bleibt.

Sollte ein Umlegen der Schnitte aber nicht möglich sein, muß beachtet werden, daß das noch lange nicht heißt, daß der Block nicht entsprechend oft geteilt werden kann. Denn manchmal kann der Block nicht in diese bestimmte Anzahl von Teilen, aber öfter zerlegt werden. Dies soll das folgende Beispiel verdeutlichen.

Nehmen wir an, wir haben einen Block, der in jeder seiner drei Raumrichtungen zwölf Zellen groß ist, also insgesamt 1728 Zellen besitzt. Dieser Block soll nun in sechs Blöcke zerlegt werden. Der Aufteilungsalgorithmus würde in erster Iteration eine Aufteilung von 1x2x3 vorschlagen. Wie klar zu sehen ist, würde die zweifache Aufteilung der dritten Dimension aber Blöcke erzeugen, die nur vier Zellen tief sind. Der Algorithmus stellt schnell fest, daß auch die erste und zweite Dimension nicht entsprechend oft geteilt werden können. Trotzdem kann der Block aufgeteilt werden. Zerlegt man ihn nämlich in acht Teile, sind diese zwar kleiner, aber eine Aufteilung von 2x2x2 erzeugt nur Blöcke, die auch in jeder Dimension groß genug sind. Und genau so geht der Algorithmus auch vor.

Nachdem die Aufteilung für alle Blöcke bestimmt ist, tauschen die Nachbarn untereinander ihre jeweilige neue Aufteilung aus. Dabei wird den Nachbarn jeweils mitgeteilt, welche Blocknummer die neuen Blöcke haben in die der eigene Block zerfällt, außerdem ihre Größe und aus welchem Teil des bisherigen Blocks sie hervorgehen. Mit diesen Informationen kann der jeweilige Block nun nicht nur in seine eben bestimmten Teile aufgeteilt werden, sondern es liegen damit auch die Informationen zur Berechnung der neuen Informationsvektoren für den Datenaustausch zwischen den neuen Blöcken vor. Die neu generierten Blöcke werden erst einmal auf dem Prozeß ihres Ursprungsblocks belassen.

Lastverteilungsschritt

Nun erfolgt die Berechnung der notwendigen Eingabedaten für den Lastverteiler, aufgrund dessen Ausgabe dann die Blöcke auf die Prozessoren umverteilt werden. Als Lastverteiler können verfügbare Tools wie Jostle oder Metis [50] eingesetzt werden. Diese Lastverteiler sind fertige Programme oder Bibliotheksaufrufe, die aus der Applikation heraus entsprechend aufgerufen werden müssen. Sie besitzen keinerlei Informationen über die Art der Anwendung an sich, was aber auch nicht notwendig ist. Lastverteiler sind Optimierungsverfahren, die auf einer allgemeinen Struktur arbeiten, auf die alle Probleme, die mit ihrer Hilfe behandelt werden sollen, abgebildet werden können. Diese allgemeine Struktur sind Graphen. Das hat den Vorteil, daß für den Lastverteiler bekannte Methoden aus der Graphentheorie eingesetzt werden können.

Bei der Lastverteilung im vorliegenden Fall ist allerdings zu beachten, daß bereits der gesamte Einlese- und Aufteilungsvorgang parallel stattfindet. Daher muß auch der Lastverteiler in der Lage sein, parallel zu arbeiten. Das heißt, er bekommt auf jedem Prozess nur die dortigen Blöcke mit ihren Nachbarschaftsbeziehungen zur Verfügung gestellt und wird dann auf allen Prozessen auch parallel aufgerufen. Dabei stellt diese Zerlegung in Teilnetze bereits eine Eingangsverteilung dar, auf deren Basis der Lastverteiler weiter optimiert. Eine zentrale Sicht auf alle Blöcke des Netzes existiert dabei zu keinem Zeitpunkt. Die oben erwähnten Lastverteiler sind beide in der Lage, dies zu leisten. Für die vorliegende Arbeit wurde Jostle verwendet, da mit diesem Lastverteiler bereits gute Erfahrungen

vorlagen.

Die Aufgabe der Lastverteiler ist aber nicht nur, die Prozessoren gleichmäßig auszulasten. Lastverteilung umfaßt mehr. Ziel der Lastverteilung ist auch, auf dem parallelen Rechensystem eine möglichst geringe Rechenzeit zu erreichen. Dazu ist eine gleichmäßige Auslastung der verwendeten Prozessoren sicher ein wichtiger Punkt. Daneben muß aber auch noch die Kommunikation in Betracht gezogen werden, die den notwendigen Informationsaustausch zwischen den Netzteilen auf den verschiedenen Prozessoren sicherstellt. Eine Verminderung dieser Kommunikation ist damit ebenfalls von Bedeutung. Daneben versucht ein Lastverteiler auch noch die durch die Umverteilung notwendige Kommunikation zu minimieren. Das fällt jedoch erst bei Anwendung der adaptiven Netzverfeinerung ins Gewicht. Leider ist das Problem der optimalen Lastverteilung NP-Vollständig [46] und so wird auch ein Lastverteiler niemals nachweislich die beste Verteilung berechnen. Allerdings bieten die Lastverteiler gute Algorithmen und Heuristiken, so daß in relativ kurzer Zeit meist gute Annäherungen an ein Optimum gefunden werden.

Für den Einsatz des Lastverteilers werden aus dem vorhandenen Netz und seiner Verteilung auf die Prozessoren der entsprechende Graph bzw. die entsprechenden Teilgraphen für die Übergabe an den Lastverteiler berechnet. Dabei wird jeder Block mit einem Knoten im Graphen identifiziert. Die Blockgröße in Zellen wird dabei als Knotengewicht in den Graphen übertragen. Nachbarschaftsbeziehungen zwischen Blöcken werden als Kanten zwischen den entsprechenden Knoten abgebildet. Diese sind hier nicht gewichtet, da die Latenz bei der durchschnittlichen Interfacegröße mit etwa 50% der Kommunikationkosten zu Buche schlägt und so die meist geringen Unterschiede in den Interfacegrößen und damit im Volumen der auszutauschenden Daten fast keine Rolle spielen.

Theoretisch kann der Lastverteiler auch berücksichtigen, daß die Prozessoren unterschiedliches Leistungsvermögen haben können. Da einer parallelen Applikation aber an sich keinerlei Informationen über die verwendete Hardware zur Verfügung stehen und das Programm hauptsächlich auf Systemen der obersten Leistungsklasse eingesetzt werden soll, wird hier von einem homogenen System ausgegangen, bei dem alle Prozessoren die gleiche Leistung besitzen. Diese Voraussetzung trifft auch für die meisten derzeit verwendeten PC-Cluster zu, die alle in einem Stück angeschafft wurden. Für die Verwendung in über mehrere Jahre gewachsenen heterogenen Systemen mit Prozessoren unterschiedlicher Leistungsklassen und für die Verwendung in heterogenen Metacomputingumgebungen ist hier eine entsprechende Erweiterung notwendig.

Die Aufarbeitung der Netzdaten geschieht in einem eigenen Interfacemodul, so daß der verwendete Lastverteiler, Jostle, auch einfach gegen einen anderen ausgetauscht werden kann. Anhand der Ausgabe des Lastverteilers werden dann die Blöcke mit Hilfe spezieller Blockaustauschroutinen auf die neu bestimmten Prozessoren umverteilt.

Lastverteilung für das Simulationsprogramm bei der Behandlung komplexer Oberflächen

Sobald in das Nichtgleichgewichtsprogramm die aufwendige Behandlung katalytischer und teilkatalytischer Oberflächen integriert wird, hat dies auch Auswirkungen auf die Lastverteilung. Der Mehraufwand in den oberflächennahen Zellen führt zu einem erheblichen Berechnungsmehraufwand in den entsprechenden Zellen. Diese erhöht natürlich den Aufwand für die Berechnung der entsprechenden Blöcke. Wird dies nicht berücksichtigt ergibt sich ein Lastungleichgewicht und damit sinkt die parallele Effizienz schnell erheblich ab, da die Prozesse, die keine Blöcke mit dieser Oberflächenbehandlung zu berechnen haben, während der nächsten Kommunikation auf diese langsameren Prozesse warten müssen.

Um dieses Lastungleichgewicht zu verhindern, muß der Mehraufwand für die Randbehandlung an der Oberfläche bei der Lastverteilung berücksichtigt werden. Dazu erhalten die entsprechenden Blöcke einfach einen dem Mehraufwand entsprechend gewählten Zuschlag auf die Anzahl der Zellen, für die eine Körperoberflächenberechnung durchgeführt werden muß.

3.3.7 Darstellung des Programmablaufs

Hier wird kurz der Programmablauf des parallelen Multiblock-Simulationsprogramms vorgestellt, wobei vor allem auf die Änderungen und Erweiterungen eingegangen wird.

Aufsetzen der parallelen Berechnungsumgebung

Nach dem Aufspannen der parallelen Umgebung liest der erste Prozeß aus dem Konfigurationsfile die Anzahl der zu bearbeitenden Blöcke ein. Ab diesem Zeitpunkt ist während des Startups für jeden Block des zu lesenden Originalnetzes ein Prozessor zuständig. Sollte es weniger Blöcke als Prozesse geben, was üblicherweise der Fall ist, warten die nicht benötigten Prozesse während dieser Zeit. Ansonsten bearbeitet jeder Prozessor einen oder mehrere Blöcke. Da die Arbeit pro Block während der Initialisierungsphase unabhängig von der Größe des Blocks ist und die Arbeit für einen Block auch nicht an mehrere Prozessoren verteilt werden kann, ist dies die günstigste Lösung. Die für Blöcke zuständigen Prozesse lesen dann den Kopf des jeweiligen Netzfiles parallel ein, um die Größe der Blöcke und damit des gesamten Netzen bestimmen zu können. Nach Durchlaufen der Aufteilungs- und Lastverteilungsprozedur ist die Blockverteilung für die folgenden Rechenschritte hergestellt.

Erst jetzt lesen die Prozesse parallel die für sie relevanten Daten aus dem Netzfile der jeweils zu berechnenden Blöcke ein. Hier wurde absichtlich noch kein MPI-I/O verwendet, da dies noch nicht auf allen Plattformen zur Verfügung steht. Durch das gleichzeitige Lesen von verschiedenen Dateien sollte aber auch so eine

gute I/O-Performance zu erreichen sein, sofern das I/O-System diese Leistung überhaupt zur Verfügung stellen kann. Dies ist natürlich auch für ein schnelles Einlesen der Daten mittels MPI-I/O Bedingung.

Preprocessing

Nach dem Einlesen der Blöcke werden die zur Berechnung der Netzgeometrie notwendigen Daten an den Blockrändern ausgetauscht. Nach dieser Berechnung müssen nochmal Werte für die Eigenschaften der Netzzellen im Überlappungsbereich ausgetauscht werden. Mit der folgenden Initialisierung des Erhaltungsvektors ist das Preprocessing beendet und die Hauptsimulationsschleife kann durchlaufen werden.

Simulationsteil

Die Hauptsimulationsschleife zerfällt strukturell gesehen in zwei Teile: den ersten, in dem das lineare Gleichungssystem aus den physikalischen und numerischen Gegebenheiten berechnet wird und den folgenden Lösungsschritt. Durch die Verwendung des Überlappungsbereichs an den Blockgrenzen, wird im ersten Teil eine Kommunikation zwischen den Blöcken komplett vermieden. Kommunikation ist damit nur während des Lösungsschritts notwendig. Der aus dem parallelen Einblockverfahren gegebene Löser wurde an die Gegebenheiten des Multiblockverfahrens angepaßt. Der Hauptunterschied zwischen dem Multiblockverfahren und dem parallelen Einblockverfahren ist in den unterschiedlichen Indexrichtungen begründet. Dies ist im parallelen Einblockverfahren ja nicht der Fall, da hier alle Netzblöcke für die Parallelisierung aus einem Block entstehen. Damit konnten im parallelen Linienrelaxationsverfahren im Einblockfall nach der Berechnung jeder Linie die Ergebnisdaten an den Rändern ausgetauscht werden. Im Multiblockverfahren ist dies nicht mehr möglich. Daher werden hier die Randwerte nach jedem vollständigen Subiterationsschritt, also nach der Berechnung aller drei Linien des Blockes in einem Subiterationsschritt, ausgetauscht. Dadurch müssen im Extremfall gegenüber dem Einblockverfahren geringfügig höhere Subiterationszahlen gewählt werden, um eine entsprechende Konvergenz zu erreichen. Bei Vergleichen mit dem Einblockverfahren an Hand einfacher Körper hat sich jedoch gezeigt, dass dieser Nachteil durch die Verwendung der besseren Multiblocknetze und dem damit verbundenen Wegfall der metrischen Singularität beim verwendeten URANUS Verfahren von der Konvergenz des Gesamtverfahrens her mehr als ausgeglichen wird.

Postprocessing

Nach Beendigung der Simulation müssen die Ergebnisse der Simulation gespeichert werden. Dazu werden die Rechenergebnisse der Blöcke in jeweils ein File

geschrieben. Dabei werden die Blöcke aus dem Aufteilungsalgorithmus zu Grunde gelegt. Es werden also nicht die Daten blockweise bezogen auf das ursprüngliche Netz zurückgeschrieben, sondern bezogen auf die aktuelle Blockverteilung. Es werden aber die Informationen, zu welchem Block aus dem Originalnetz der Block gehört hat, mit abgelegt. Dadurch ist für die Auswertung eine Rekonstruktion der Daten und eine Darstellung auf Basis des ursprünglichen Netzes mit seiner Blockverteilung jederzeit möglich. Vorteil dieser Lösung ist, daß für einen Folgerechendurchlauf auf Basis der berechneten Ergebnisse, der häufig mit gleicher Prozessorzahl auf demselben Rechner durchgeführt wird, eine neue Berechnung der Zerlegung nicht notwendig ist. Da diese Folgerechnungen eher die Regel als die Ausnahme sind, ist dieses Verfahren gegenüber dem Ablegen der Daten in der ursprünglichen Blockstruktur gewählt worden. Ein Nachteil bei der Auswertung entsteht dadurch nicht, da diese sowieso sequentiell erfolgt und selbst bei einer vielleicht in Zukunft möglichen (teil-)parallelen Auswertung kein Nachteil entsteht.

3.4 Automatisches Multilevelverfahren

Die zur Lösung der Strömungsprobleme verwendeten Netze werden so erstellt, daß sich alle relevanten Strömungsphänomene sauber auflösen lassen. Dies führt zu einem enormen Rechenaufwand durch die Vielzahl der Zellen. Während der Anrechenphase ist aber eine so hohe Genauigkeit noch nicht notwendig, vor allem da hier erst einmal der Stoß, der in den betrachteten Überschallströmungen auftritt, richtig positioniert werden muß. Die dafür notwendige Iterationszahl hängt aber von der Zahl der Zellen ab, die zwischen Körperoberfläche und der späteren Stoßposition liegen. Mit der Iterationszahl steigt aber der zeitliche Aufwand. Hier kann ein Verfahren gewählt werden, das mit einem gröberen Netz bei geringerer Genauigkeit weniger Rechenzeit benötigt.

Die Idee ist, hierfür ein Multilevelverfahren einzusetzen. Im Gegensatz zu manch anderen Programmen, die eine ähnliche Strategie verwenden [64], dafür aber die Eingabe der ganzen Netzhierarchie benötigen, soll hier das Simulationsprogramm die notwendigen Netze aus dem Eingabenetz selbst generieren und auch für die Anpassung dieser Netze an das parallele Verfahren sorgen. Es sollen lediglich die Anzahl der Vergröberungsschritte also die Zahl der Level und gegebenenfalls noch die maximale Anzahl der Iterationen pro Netzlevel eingestellt werden müssen.

3.4.1 Generierung der Netze

Für das Multilevelverfahren werden Netze verschiedener Auflösung verwendet. Diese Netze können entweder vom Benutzer bereitgestellt oder aus dem vorhandenen (feinen) Netz berechnet werden. Um das Verfahren möglichst einfach hand-

haben und einsetzen zu können, sollen hier die Netze vom Programm selbst generiert werden. Die Netze sollen möglichst einfach auseinander hervorgehen. Hier wurde dazu ein einfaches Verfahren verwendet, bei dem aus jeweils 8 Zellen in 3D durch Zusammenlegen eine neue Zelle entsteht. Es werden also in jeder Raumrichtung jeweils zwei Zellen zusammengelegt. Beim ursprünglichen Einblockverfahren kann diese Methode leicht angewendet werden, da von vorneherein keinerlei Nachbarschaftsbeziehungen zu berücksichtigen sind. So können einfach die entsprechenden Zellen zusammengelegt werden. Sollte es an den Blockrändern nicht aufgehen, weil die Zellzahl ungeradzahlig ist, so werden einfach in der jeweiligen Richtung drei Zellen zusammengelegt.

Im Multiblockverfahren kann so nicht vorgegangen werden, da bereits beim Programmstart mehrere Blöcke mit ihren Nachbarschaftsbeziehungen vorliegen, und diese Nachbarschaftbeziehungen verhindern im allgemeinen das für das Einblockverfahren beschriebene Vorgehen, zum Beispiel wenn zwei Blöcke an eine Seite eines anderen angrenzen und beide Blöcke an der Blockschnittstelle eine ungerade Zellenzahl aufweisen. Da die hier verwendeten Netze bereits unter dem Gesichtspunkt eines Einsatzes für ein Multilevelverfahren generiert werden, haben die Blöcke und vor allem die Schnittstellen nur gerade Zellenzahlen. Desweiteren wird bei der Netzgenerierung auf eine durch vier oder acht teilbare Zellenzahl geachtet, womit Netzhierarchien mit drei bzw. vier Netzstufen einfach zu realisieren sind. An den Schnittstellen zwischen den Netzblöcken gibt es hier keine Probleme, da so sichergestellt ist, daß immer eine Zelle eines Blocks direkt an eine Zelle des Nachbarblocks angrenzt. Ein tiefer gehendes, aufwendigeres Verfahren, das zum Beispiel an den Netzgrenzen mit Interpolation arbeitet, wurde daher nicht in Betracht gezogen.

3.4.2 Behandlung der Netzgrenzen und paralleles Verfahren

An den Netzgrenzen müssen die Informationen, die die Schnittstelle realisieren, an das Multilevelverfahren angepaßt werden. Dazu werden die Informationen in den Rand- und Kommunikationsvektoren bei der Generierung der Netzhierarchie entsprechend umgerechnet. Für das parallele Verfahren bedeutet die Verwendung eines Multilevelverfahrens, daß die Blöcke für die Parallelisierung auch das im vorigen Abschnitt beschriebene Aussehen haben, also entsprechende Zellzahlen an den Seiten und Schnittstellen aufweisen. Dies erreicht man am einfachsten dadurch, daß die Zerlegung der Netzblöcke für die Parallelisierung auf dem gröbst möglichen Netz durchgeführt wird. Dies hat dann allerdings wieder den Nachteil, daß sich die Blockgrößen der zu generierenden Blöcke stärker unterscheiden, was sich nicht vorteilhaft auf die Lastverteilung auswirkt. Umgehen ließe sich dies durch eine separate Blockaufteilung und Lastverteilung für jedes Level, was dann allerdings wieder zusätzlichen Berechnungs- und Kommunikationsaufwand mit

sich bringt.

3.4.3 Umschaltung zwischen den Netzleveln

Die Umschaltung zwischen den Netzleveln erfordert, daß die Lösungen vom groben auf das feine Netz transportiert, also prolongiert werden. Dazu exisitieren aus den Multigridverfahren mehrere zum Teil recht aufwendige Verfahren [33]. Hier wurde für den Anfang ein einfaches Verfahren verwendet, bei dem die Ergebniswerte einer Zelle aus dem groben Netz einfach auf die Zellen im feinen Netz, aus denen die "grobe" Zelle hervorging, kopiert werden. Schon mit dieser Methode lassen sich recht bemerkenswerte Ergebnisse erzielen. Da die Umschaltung jeweils zum Ende eines Iterationsschritts stattfindet, brauchen nur Werte aus dem Erhaltungsvektor und die des Änderungsvektors übertragen zu werden, letzterer auch nur, um für die Lösungsiterationen bereits eine bessere Startlösung verwenden zu können.

Kapitel 4

Praktische Untersuchungen

Das neu entwickelte parallele Multiblockverfahren wurde mit mehreren Testfällen verifiziert und auf einer großen Zahl unterschiedlicher Hardwareplattformen getestet. Dabei wurden nicht nur klassische massiv parallele Rechner wie die Cray T3E eingesetzt, die ja ursprünglich die Zielarchitektur für die Parallelisierung war, sondern es wurden auch große Systeme mit gemeinsamem Hauptspeicher (Shared Memory Systeme) verwendet. Darüber hinaus wurde das Programm auf der neuen Klasse der hybriden Systeme (Distributed Shared Memory), wie der Hitachi SR8000 getestet, die inzwischen verfügbar waren. Außerdem wird das beschriebene URANUS Programm als Benchmark am HLRS verwendet, so dass für viele neue und auch prototypische Systeme Ergebnisse vorliegen. Diese dürfen jedoch größtenteils nicht veröffentlicht werden.

4.1 Verwendete Hardware

In diesem Kapitel soll ein Überblick über die verwendeten Hardwarearchitekturen gegeben werden. Die verwendeten Systeme sind dabei grob nach der Speicherhirarchie, also verteilter Hauptspeicher, gemeinsamer Hauptspeicher und heterogene Systeme, klassifiziert. Unabhängig von der Architektur wird die Applikation jedoch immer wie auf einem System mit verteiltem Hauptspeicher übersetzt und verwendet, da ansonsten weitere Änderungen im Quellcode notwendig werden würden, von denen dann aber nicht alle Plattformen profitieren. Trotzdem kann die Leistungsfähigkeit der verwendeten Systeme zu einem sehr großen Teil ausgeschöpft werden, da auf einem System mit gemeinsamem Speicher die Datenübertragung zwischen den Prozessoren über den Hauptspeicher mit hoher Bandbreite und geringer Latenz geschieht. Dazu muß allerdings die verwendete MPI Bibliothek entsprechend optimiert sein. Dazu wird jeweils die entsprechend angepaßte Bibliothek des Hardwareherstellers verwendet. Auf hybriden Systemen wird das Programm wie auf massiv parallelen Prozessoren eingesetzt. Eine etwaige Unterstützung der Heterogenität ist momentan nicht implementiert. In Kapitel

sechs sind allerdings bereits ein paar Ideen skizziert, wie eine entsprechende Erweiterung solch eine Plattform noch besser nutzen könnte. Vor einer Umsetzung wäre allerdings abzuschätzen, ob der Aufwand mit dem zu erwartenden Gewinn in einer vertretbaren Relation steht. Die effiziente Nutzung hybrider Systeme durch die Verwendung eines Shared Memory und eines Distributed Memory Programmiermodells verlangt vom Entwickler neben Kenntnissen und guter Erfahrung mit beiden Modellen auch den Umgang mit der zusätzlichen Komplexität. Außerdem muß das Programmiermodell für die Nutzung des verteilten Speichers, fast ausschließlich kommt hier MPI zur Anwendung, die Programmierung mit Threads auf einem Knoten des hybriden Systems unterstützen, also "threadsafe" sein, damit ein entsprechendes Programmiermodell, meist OpenMP, überhaupt sinnvoll verwendet werden kann.

Die Programmierung von URANUS wurde so portabel wie möglich gestaltet. Voraussetzung für die Verwendung des Programms auf einem Rechensystem ist lediglich, dass dieses Fortran90 und MPI unterstützt. Dies ist heute auf jedem System, das für technisch wissenschaftliche Simulationen in Frage kommt, gegeben. Auch alle entsprechenden Systeme mit gemeinsamem Hauptspeicher, ebenso wie die hybriden Systeme, erfüllen diese Voraussetzung. Deshalb wurde die Applikation auf einer großen Anzahl von Plattformen getestet. Da die jeweilige Architektur natürlich trotz der portablen Programmierung einen erheblichen Einfluss auf die Leistung des Programms hat, werden diese hier kurz klassifiziert und vorgestellt.

4.1.1 Systeme mit verteiltem Hauptspeicher

Systeme mit verteiltem Hauptspeicher bestehen aus einzelnen Rechenknoten, die durch ein schnelles Netzwerk miteinander verbunden sind. Die einzelnen Knoten sind quasi eigenständige Rechner, insbesondere besitzt jeder Knoten seinen eigenen Hauptspeicher, die nur gemeinsam verwaltet werden. Dabei können Knoten für Spezialaufgaben dediziert sein. Es kann also in einem System zum Beispiel Knoten zum Login, Knoten für den I/O und Knoten für die Bearbeitung der Applikationen geben. Dabei kann das Betriebssystem entsprechend der Aufgaben angepaßt sein. In diese Klasse gehören auch PC- und Workstation-Cluster, sofern sie pro PC bzw. Workstation nur einen Prozessor besitzen.

Cray T3E

Die Cray T3E ist der letzte wirklich monolithische Massiv Parallele Rechner (MPP), der als ein System geliefert wurde. Das Testsystem war die T3E am HLRS in Stuttgart, bestückt mit 512 DEC Alpha EV5 21164 Prozessoren, die mit einer theoretischen CPU-Leistung von 900 MFlops bei 450 MHz Taktrate der T3E zu einer Systemleistung von 460 GFlops verhelfen. Das Verbindungsnetzwerk ist

als 3D Torus ausgeführt. Jeder Prozessor hat also sechs direkte Nachbarn mit jeweils einer direkten Verbindung. Das Netzwerk erlaubt eine reale Bandbreite von 300 MB/s je Verbindung bei einer Latenz von ca. 15 μs. Neben der T3E wurde URANUS noch auf folgenden Massiv Parallelen Rechnern getestet:

- IBM RS/6000 SP
- Hitachi SR 2201

4.1.2 Systeme mit gemeinsamem Hauptspeicher

Bei Systemen mit gemeinsamem Hauptspeicher sind prinzipiell zwei Arten zu unterscheiden. Dabei handelt es sich um die Systeme mit tatsächlich gemeinsamem Hauptspeicher und solche mit virtuell gemeinsamem Hauptspeicher. Während bei ersteren Systemen der Hauptspeicher tatsächlich von allen Prozessoren gleichartig angesprochen werden kann, also der Hauptspeicher physikalisch allen Prozessoren in gleicher Weise zugeordnet ist (Uniform Memory Access, UMA), befindet sich bei letzteren der Hauptspeicher bei einem Prozessor oder einem Prozessorpaar, wird aber durch Hardware- und Softwareunterstützung auch den anderen Prozessoren zur Verfügung gestellt. Dies führt aber dazu, dass die Zugriffszeit, also die Speicherlatenz, von der Lage des Hauptspeichers abhängig ist. Der Zugriff des lokalen Prozessors erfolgt dabei mit geringerer Latenz als der Zugriff eines entfernteren Prozessors. Je nach System kann der Latenzunterschied bis zu einem Faktor drei betragen. Diese Systeme werden als NUMA-Systeme (Non Uniform Memory Access) bezeichnet. Darüberhinaus kann, je nach Architektur, die Speicherbandbreite in den lokalen Speicher größer sein als in den entfernten.

Ein Problem der NUMA-Systeme ist der Cache, der üblicherweise nicht von allen Prozessoren des Systems gemeinsam benutzt wird. Typischerweise ist der Level-1 Cache CPU-lokal, der Level-2 und Level-3 Cache werden, falls vorhanden, oft von einer geringen Anzahl von Prozessoren gemeinsam verwendet. Um diese Systeme auch so einfach wie UMA-Maschinen programmieren zu können, ist es unumgänglich, dass die Caches der Prozessoren koherent gehalten werden. Dazu werden zusätzliche Hardware und Cache-Koherenzprotokolle eingesetzt. Diese Systeme werden dann als cc-NUMA bezeichnet. NUMA-Systeme ohne Cache-Koherenz werden nur noch äußerst selten eingesetzt, da hier der Programmierer auf diesen Umstand Rücksicht nehmen muss. Meist werden solche Systeme wie Systeme mit verteiltem Speicher verwaltet, und die MPI-Software sowie andere maschinenspezifische Software nutzen den globalen Adressraum, um eine höhere Leistung zu erzielen. Eigentlich ist auch die Cray T3E ein NUMA System, wurde aber hauptsächlich wie ein Massiv Paralleles System programmiert und verwendet. Auch das Systemmodell war eher das eines Massiv Parallelen Systems, da Prozesse trotz des globalen Adressraums nicht einfach zwischen den Prozessoren verschoben werden konnten.

Der gleichförmige Speicherzugriff auf UMA Systemen wird meist durch einen gemeinsamen Speicherbus oder einen Switch realisiert, der jede CPU auf jede gewünschte Speicherbank aufschalten kann.

NEC SX-6+

Die NEC SX-6+ ist eine klassische UMA Maschine, wenn man einmal davon absieht, dass sich die Systeme auch noch mittels eines speziellen Hochgeschwindigkeitsverbindungsnetzwerks clustern lassen. Allerdings besitzt die SX-6+ gegenüber den anderen verwendeten Systemen einen weiteren gravierenden Unterschied. Bis auf die SX-6+ basieren alle verwendeten Systeme auf Mikroprozessoren. Die SX-6+ besitzt Vektorprozessoren, ist also ein Vektorrechner, während alle anderen Systeme auf Mikroprozessoren basieren. Im Gegensatz zu den mikroprozessorbasierten Systemen darf man auf der SX-6+ durch die ausgefeilte Vektortechnologie und eine entsprechend hohe Speicherbandbreite eine hohe Effizienz bei der CPU-Nutzung erwarten. Je nach Applikation sollten 30-50% der theoretischen Spitzenleistung eines Prozessors tatsächlich beim Programmablauf für die gesamte Applikation gemessen werden. Einzelne Programmteile erreichen sogar 75 bis fast 100% der theoretischen Spitzenleistung, der sogenannten Peak Performance. Die verwendeten Systeme in Stuttgart besitzen jeweils 8 Vektorprozessoren mit je 9 GFlops Spitzenleistung, die über ein mehrstufiges Speichernetzwerk mit den 4096 Speicherbänken verbunden sind. Jeder der Rechenknoten besitzt also eine Maximalleistung von 72 GFlops. Die Speicherbandbreite beträgt etwa 36 GByte/s pro Prozessor, die aufgrund des Switches und einer entsprechenden Anzahl Leitungen auch mit der Anzahl der Prozessoren pro Knoten skaliert. Ein SX-6+ Knoten besitzt somit eine Speicherbandbreite von 288 GByte/s. Damit kann also ein halbes Speicherwort pro Fliesskommaoperation pro Prozessor aus dem Hauptspeicher geladen werden.

Die MPI Bibliothek auf der SX-6+ wurde von NEC speziell an die Architektur angepaßt, so dass mit dieser, dank der hohen Speicherbandbreite, eine ausgezeichnete Performance auch für MPI-paralleleisierte Anwendungen erreichbar ist.

HP ZX 5800

Bei der ZX 5800 handelt es sich ebenfalls um ein UMA System mit gemeinsamem Hauptspeicher, das sich aber in wesentlichen Zügen von der SX-6+ unterscheidet. Der Itanium2 Prozessor (Madison, 1,3 GHz Takt) ist ein 64-Bit Prozessor von Intel. Der interne Aufbau entspricht dem eines typischen Microprozessors mit grossen Caches, wobei die vier Funktionseinheiten schon vektorartig aufgebaut sind. Allerdings kann mit einer Maschineninstruktion nach wie vor nur eine Operation pro Funktionseinheit ausgeführt werden. Auf der ZX 5800 arbeiten nun vier dieser CPU's auf dem 48 GB großen Hauptspeicher, an den sie über einen gemeinsamen Bus angebunden sind. Das bedeutet, dass zu einem Zeitpunkt im-

mer nur eine CPU auf den Hauptspeicher oder das I/O-System zugreifen kann. Die Speicherbandbreite des Systems beträgt 3,2 GByte/s und ist damit genau so groß wie die Speicherbandbreite eines der Prozessoren.

Außer auf den vorgestellten Systemen wurde URANUS auch auf folgenden Shared Memory Systemen getestet:

- UMA
  - HP V-class
  - SUN 10k
- cc-NUMA
  - SGI Origin
  - IBM p690
  - NEC Azusa und Asama

4.1.3 Heterogene Systeme

Fast alle neueren Systeme gehören allerdings zu einem Zwittertyp, der beide bisher vorgestellten Rechnerklassen miteinander vereinigt. Diese heterogenen Systeme bestehen aus Rechnerknoten, die mit einem schnellen Netzwerk wie in einem System mit verteiltem Speicher gekoppelt sind. Allerdings sind diese Knoten keine Rechner mehr, die aus einem Prozessor mit seinem Hauptspeicher bestehen, sondern komplette Shared Memory Systeme mit mehreren Prozessoren. Dabei können die Bauarten der Knoten, je nach Hersteller, sowohl im Hinblick auf die Zahl der Prozessoren als auch auf die weitere Speicherorganisation (NUMA oder UMA) völlig unterschiedlich sein.

Von einem MPP aus gesehen sind die heterogenen Systeme also MPP's, bei denen der Einzelrechner, das Processing Element, durch einen Shared Memory Rechner ersetzt wurde. Aus Sicht der SMP's sind die heterogenen Systeme einfach Cluster derselben. Je nach der Richtung, auf der sich die Hersteller auf diese heterogenen Systeme zu bewegen, fallen hier die Knoten unterschiedlich aus: die Hersteller von großen SMP-Systemen (z.B. NEC) setzen weiterhin auf diese und setzen damit große und leistungsfähige Knoten ein, die von einem speziellem Höchstleistungnetzwerk verbunden werden. Dafür fällt die Knotenzahl noch relativ moderat aus, mit z.B. 640 Knoten im Earth Simulator [19]. Die Hersteller, die bisher eher der Workstationtechnologie verbunden waren, setzen hier auf kleinere SMP-Knoten mit z.B. zwei oder vier Workstationprozessoren, die auch über ein schnelles Netzwerk verbunden werden. Allerdings ist dieses von der Einzelverbindung her gesehen erheblich schwächer als das der vorher genannten Maschinen. Dafür werden hier große Knotenzahlen von 2000 (Mare Nostrum) und mehr erreicht.

NEC SX-6M48

Die NEC SX-6M48 besitzt mit den SX-6+ Systemen sehr starke Knoten. Diese sind über ein spezielles proprietäres Hochgeschwindigkeitsnetzwerk gekoppelt, das einen Datentransfer mit 8 GB/s je Knoten und Richtung erlaubt. Damit besteht das System in Stuttgart aus 6 Knoten mit einer Gesamtspitzenleistung von fast 434 GFlops und 384 GByte Hauptspeicher. Da auch viele Programme auf dem System laufen, die nur einen Teil eines Knotens oder gar nur einen Prozessor verwenden, stehen für ein paralleles Programm maximal 4 Knoten mit insgesamt 32 Prozessoren zur Verfügung.

Cray Strider

Der Cray Strider ist der Prototyp für die Cray XT3, mit der Cray wieder einen rein Massiv Parallelen Rechner auf den Markt bringt. Der Strider dient hauptsächlich zur Validierung des Softwarestacks und besteht im Gegensatz zum endgültigen Produkt aus Zweiprozessorknoten mit gemeinsamem Hauptspeicher. Die einzelnen Knoten sind von der Architektur her gesehen Standardserver PC's mit zwei AMD Opteronprozessoren mit je 2,0 GHz Taktrate und 4 GByte Hauptspeicher. Dabei ist jeweils die Hälfte des Hauptspeichers einem Prozessor zugeordnet, da der auch das Speichermanagementmodul enthält. Allerdings hat jeder der Prozessoren über eine direkte Verbindung (Hypertransport) einen direkten Zugriff auf den Hauptspeicher des anderen Prozessors. Damit liegt die Spitzenleistung eines Knotens bei 8 GFlops bei einer maximalen Speicherbandbreite von 12,8 GByte/s. Das System in Stuttgart besteht aus 129 solchen Knoten, wobei allerdings zwei für I/O und einer als Frontend dediziert sind. Alle Knoten sind für den Nachrichtenaustausch mit Myrinet untereinander verbunden. Damit stehen für die Applikationen 126 Knoten mit einer Gesamtleistung von knapp einem Teraflops und 504 Gigabyte Hauptspeicher zur Verfügung.

Als Zusammenfassung und zur leichteren Orientierung sind die Leistungsdaten der verwendeten Parallelrechnersysteme in den folgenden Tabellen 4.1 bis 4.4 zusammengefaßt. Dabei sind sowohl die theoretische Maximalleistung pro CPU und pro Knoten als auch die Speicherbandbreiten sowie die mit MPI gemessenen Netzwerkbandbreiten und -latenzen angegeben.

Da neben der reinen Leistung auch die Ausgewogenheit des Rechnersystems einen Einfluß auf die erreichbare Leistung mit URANUS hat, sind in den vier Tabellen aus den ermittelten und berechneten Leistungsdaten Kennzahlen für die relative Leistungsfähigkeit, bezogen auf die theoretische Maximalleistung, berechnet worden. Dabei wird die Latenz in "Anzahl Fließkommaoperationen" angegeben, die während der Zeit von einem Prozessor bzw. vom Knoten durchgeführt werden können. Damit läßt sich also die mögliche Arbeit während der Zeitverzöge-

Rechner	**Maximalleistung Proz. [MFlops]**	**Speicherband-breite [MB/s]**	**rel. Bandbreite [Byte/Flop]**
T3E	900	150	0,17
HP ZX 5800	5200	3200	0,62
Strider	4000	6400	1,6
SR 8000	1000	4000	4,0
NEC SX-5	4000	32000	8,0
NEC SX-6+	9000	36000	4,0

Tabelle 4.1: Leistungskennzahlen Prozessoren

Rechner	**Proz. pro Kn.**	**Max. Leistung Knoten [MFlops]**	**Speicherbandbr. Kn. [MB/s]**	**rel. Bandbr. [B/Flop]**
T3E	1	900	150	0,17
HP ZX 5800	4	20800	3200	0,15
Strider	2	8000	12800	1,6
SR 8000	8	8000	32000	4,0
NEC SX-5	16	64000	512000	8,0
NEC SX-6+	9	72000	288000	4,0

Tabelle 4.2: Leistungskennzahlen Knoten

Rechner	**Max. Leistung Knoten [MFlops]**	**Netzwerkband breite [MB/s]**	**rel. Bandbreite [B/Flop]**
T3E	900	300	0,33
HP ZX 5800	10400	230	0,022
Strider	8000	230	0,029
SR 8000	8000	1000	0,125
NEC SX-6+	72000	8000	0,111

Tabelle 4.3: Durchsatz Netzwerk

Rechner	**Max. Leistung Knoten [MFlops]**	**Latenz [μs]**	**rel Latenz [Flop]**
T3E	900	15	13500
HP ZX 5800	10400	7	72800
Strider	8000	7	56000
SR 8000	8000	8	64000
NEC SX-6+	72000	6	432000

Tabelle 4.4: Latenz Netzwerk

rung durch die Kommunikation bestimmen. Die Bandbreite ist in Byte/s pro Flops angegeben. Ein Wert von 1 heißt hier also, daß pro Fließkommaoperation in der Sekunde ein Byte übertragen werden kann. Dies ist damit ein Maß dafür, ob die Daten schnell genug auf den Prozessor bzw. Knoten gelangen oder wieviele Operationen die CPU ausführen könnte, während eine bestimmte Datenmenge übertragen wird.

Aus der relativen Speicherbandbreite (Tabelle 4.1 und Tabelle 4.2) ist zu ersehen, dass nur auf der NEC SX-5 pro Operation ein 64-bit Wort aus dem Speicher in den jeweiligen Prozessor geladen werden kann, auf der SX-6 und der Hitachi SR8000 ist es noch ein halbes, auf den anderen Systemen noch erheblich weniger. Da bei großen Simulationen wie URANUS, die keine hohe Datenlokalität besitzen, hauptsächlich der Datenfluß zwischen dem Hauptspeicher und der CPU die Leistung begrenzt, sind diese Werte ausschlaggebend für die Leistung des Systems mit URANUS auf einem Knoten. Zu bemerken wäre noch, dass die relative Speicherbandbreite des ZX 5800 Knotens geringer ist als die pro Prozessor, da die Prozessoren über einen Bus an den Hauptspeicher angebunden sind und sich so die Hauptspeicherbandbreite teilen.

Die Netzwerkleistung in den Tabellen 4.3 und 4.4 zeigt, dass die T3E gegenüber der CPU-Leistung ein enorm starkes Netzwerk besessen hat. Nur in der relativen Bandbreite können die SX-6 und die SR8000 mit stark einem Drittel noch in etwa mithalten. Bei der relativen Latenz hingegen kann der T3E keines der Systeme das Wasser reichen. Inbesondere die SX-6 fällt hier hinter alle anderen Systeme zurück, obwohl sie, absolut gesehen, die beste Latenz der betrachteten Systeme besitzt. Dem steht aber die enorme Leistung der angeschlossenen Knoten gegenüber. Hier werden vor allem kleine Nachrichten, wie sie bei kleinen Blöcken vorkommen können, zu Leistungseinbußen führen.

4.2 Verwendete Testfälle

Für den Test des neu entwickelten parallelen Programms wurden hauptsächlich zwei Testfälle verwendet, die im folgenden näher vorgestellt werden sollen. Als erstes wurde die Strömung um die Colibrikapsel berechnet. Als weiterer Testfall, fast schon Produktionstest, stand ein Netz für die X-38 zur Verfügung.

4.2.1 Colibri Kapsel

Bei der Colibri Kapsel handelt es sich um eine geplante Wiedereintrittskapsel, die als Experiment unter anderem zur Erprobung von keramischen Hitzeschutzmaterialien geflogen werden sollte. Leider konnte das Experiment selbst aus Geldmangel nicht weiter verfolgt werden. Trotzdem ist es jedoch für den Test des URANUS Programms ein hervorragender Kandidat, da für die Colibri-

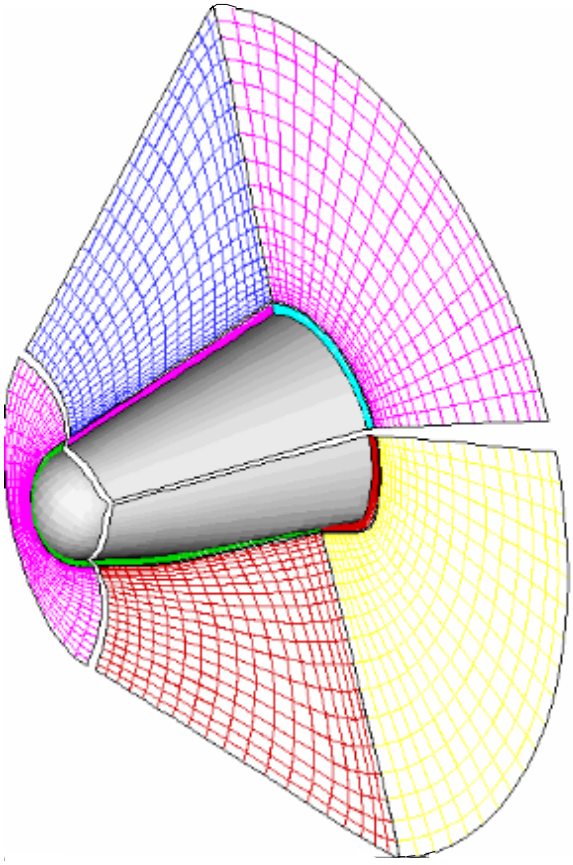

Abbildung 4.1: Multiblocknetz um den Colibrikörper

kapsel sowohl eine Vielzahl von Windkanalmessungen als auch bereits Idealgas-Simulationsergebnisse mit dem Singleblock URANUS-Verfahren vorliegen. Für das Singleblockverfahren wurden allerdings andersartige Netze, sogenannte C-Netze verwendet, bei denen die dritte Dimension durch die Rotation eines 2-D Netzes um die Körpermittelachse erlangt wird. Daher waren für das Multiblockverfahren entsprechende Multiblocknetze mit einem Gittergenerator, Gridpro [36], zu erzeugen.

Wie in Abbildung 4.1 zu sehen ist, besteht das neue Kolibrinetz aus fünf Blöcken. Vier Blöcke sind um den Körper herum angeordnet und enden an einer Schnittstelle zum Frontblock, der den Nasenbereich der Kapsel umhüllt. Dabei unterscheidet sich das Netz in der Topologie vor allem durch diesen eingesetzten Frontblock vom entsprechenden C-Netz. Das Colibri Netz fällt außerdem auch in die Klasse der sogenannten Vorkörpernetze, da sich das Netz nur um den Körper oder allgemein den vorderen Teil eines Körpers erstreckt und so vor allem die Strömung im Nachlauf nicht mit betrachtet wird. Ausgehend von dem generierten Basisnetz wurden durch Vergröberung und einfache Verfeinerung ohne Interpola-

Abbildung 4.2: Fotomontage der X38 während eines Wiedereintritts

tion eine ganze Netzhierarchie generiert, vor allem, um das parallele Programm auch mit größeren Zellenzahlen testen zu können. Neben den Leistungsmessungen wurde anhand dieses Netzes das neue Verfahren auch validiert.

4.2.2 X38 - Prototyp für das CRV

Als weiteres Netz für ein reales Problem wurden Netze für die X38 zu Test- und Produktionsrechnungen verwendet. Dazu standen Netze der DLR und von der TU München zur Verfügung. Die X38 ist der Prototyp des sogenannten "Crew Rescue Vehicles" (CRV, vgl. Abbildung 4.2), also eines Rettungsfahrzeugs. Die CRV besitzen dabei bis auf einen kleinen Manövertreibsatz keinen eigenen Antrieb und sind vor allem nicht zum Selbststart von der Erde aus vorgesehen. Vielmehr sollen sie vom Space Shuttle zur Internationalen Raumstation (ISS) transportiert und dann dort als Rettungsgerät zur eventuellen Evakuierung der ISS verankert werden. Natürlich besitzt das CRV einen Hitzeschutz, der das Fahrzeug und seine bis zu neun Insassen vor den bei einem Wiedereintritt auftretenden Temperaturen schützt. Das Hitzeschutzsystem für das CRV besteht aus modernen Hitzeschutzmaterialen wie C-Si/C, die, wie auch das vorliegende Simulationsprogramm, im Sonderforschungsbereich (SFB) 259 [78] entwickelt wurden.

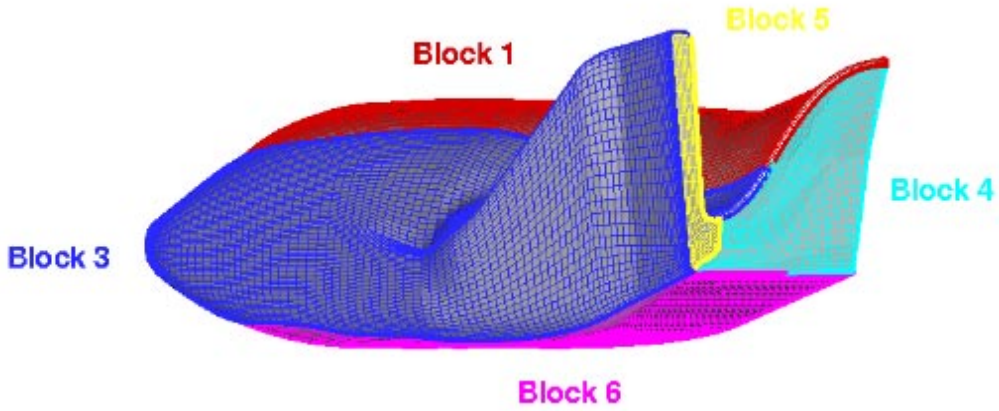

Abbildung 4.3: Oberflächennetz des Netzes zur Simulation der Strömung um die X38

Abbildung 4.3 zeigt das Oberflächennetz, also nur den Oberflächenteil des verwendeten Netzes. Schon daraus kann man dessen Aufbau erkennen, der deutlich komplizierter als der des vorhin vorgestellten Colibrinetzes ist. Das Netz ist ein Halbkörpernetz, da die X38 Spiegelsymmetrisch ist, und besteht aus sechs Blöcken mit insgesamt 559.104 Zellen. Der zweite Block, der den Nasenbereich umschließt ist in der Abbildung nicht zu erkennen. Die Komplexität des Netzes äußert sich darin, dass an den meisten Blockschnittstellen die Indexrichtungen gegeneinander gedreht sind. Darüber hinaus weist das Netz stark unterschiedlich große Blöcke auf, der größte besitzt mit 263.424 Zellen genau 28 mal soviele wie der kleinste mit 9408. Damit ist dieses Netz auch für die Lastverteilung eine Herausforderung. Um das Verhalten des Programms nicht gleich mit einer großen Prozessorzahl, wie sie für die Berechnung und vor allem die Speicherung dieses Netzes notwendig ist, erproben zu müssen, wurde durch Verschmelzung von jeweils acht Zellen ein gröberes Netz erzeugt, das dann für die Tests besser geeignet war. Von diesem Netz ist der Oberflächenteil in Abbildung 4.4 zu sehen.

Um auch Strömungen um einen kompletten Körper berechnen zu können, wurde das vorgestellte, spiegelsymmetrische Netz durch Spiegelung zu einem Ganzkörpernetz ergänzt. Dieses Netz umfaßt dann 1.099.392 Zellen in insgesamt acht Blöcken.

4.3 Ergebnisse

Im folgenden Abschnitt werden nun die wichtigsten Messungen und Ergebnisse der verschiedenen Programmläufe und Tests vorgestellt. Natürlich wurden mit

URANUS während der Entwicklung und den Tests weitaus mehr Rechnungen durchgeführt, die aber hier aus Platzgründen nicht dargestellt werden können.

4.3.1 Messmethode

Zur Darstellung der Simulationsergebnisse wurden diese vom Format her konvertiert und mit Hilfe von Tecplot visualisiert. Auf diese Art wurden alle hier gezeigten Strömungs- und Netzdarstellungen erzeugt. Für die Zeitmessungen wurden die in MPI verfügbaren Routinen zur Abfrage der Systemuhr verwendet (Wallclock), woraus durch Differenzbildung die jeweiligen Programmlaufzeiten bestimmt wurden. Dabei wurde neben der Gesamtlaufzeit auch die Laufzeit für den reinen Rechenteil bestimmt. Die so gewonnenen Ergebnisse sind natürlich nur relevant, wenn das Programm nicht durch andere Programme in seinem Ablauf gestört wird. Da aber mit Ausnahme der NEC SX-6 auf allen Systemen die Prozessoren dediziert für die Programmläufe vergeben wurden, kann diese Voraussetzung als gegeben angenommen werden. Einzig im I/O und bei der Kommunikation könnten sich Programme noch gegenseitig stören. Dabei sollten sich die Kommunikationsvorgänge verschiedener Programme auf den meisten Systemen allerdings nicht gegenseitig stören, da die Netzwerktopologie entsprechend ausgelegt ist. Gegenseitige Behinderungen beim I/O können dagegen vorkommen, sollten aber aufgrund der seltenen Benutzung des I/O Systems während der Programmlaufzeit in URANUS wie auch bei den meisten anderen Programmen nur sehr selten vorkommen. Im Ausnahmefall müssten Ausreißer in den Ergebnissen durch wiederholte Rechnung bestätigt werden.

Auf der NEC SX-5 und SX-6 wurde ein anderes Verfahren angewendet, da bei diesem System die Prozessoren im Zeitscheibenverfahren von mehreren Programmen gleichzeitig genutzt werden und die Maschine zwecks besserer Auslastung überbucht betrieben wird. Allerdings stehen auf den SX Systemen einige Werkzeuge zur Verfügung, welche die eigenen Messungen unnötig machen. Mit diesen Werkzeugen kann neben der tatsächlichen Programmlaufzeit zum Beispiel auch die Megafloprate und der Vektorisierungsgrad gemessen werden. Mit diesen Werkzeugen wurden daher die entsprechenden Messungen auf der NEC SX-6 durchgeführt, sofern die verwendeten Knoten nicht dediziert belegt werden konnten.

4.3.2 Validierung des Verfahrens

Anhand des Colibri Netzes wurde das gesamte neue Multiblockverfahren für URANUS validiert. Dies gestaltete sich recht einfach, da bereits Rechnungen mit einer prototypischen Implementierung durchgeführt worden waren. In dieser Implementierung konnten allerdings ausschließlich Multiblocknetze, die mit GridPro erzeugt worden waren, verarbeitet werden. Der Vergleich einer Rechnung für das

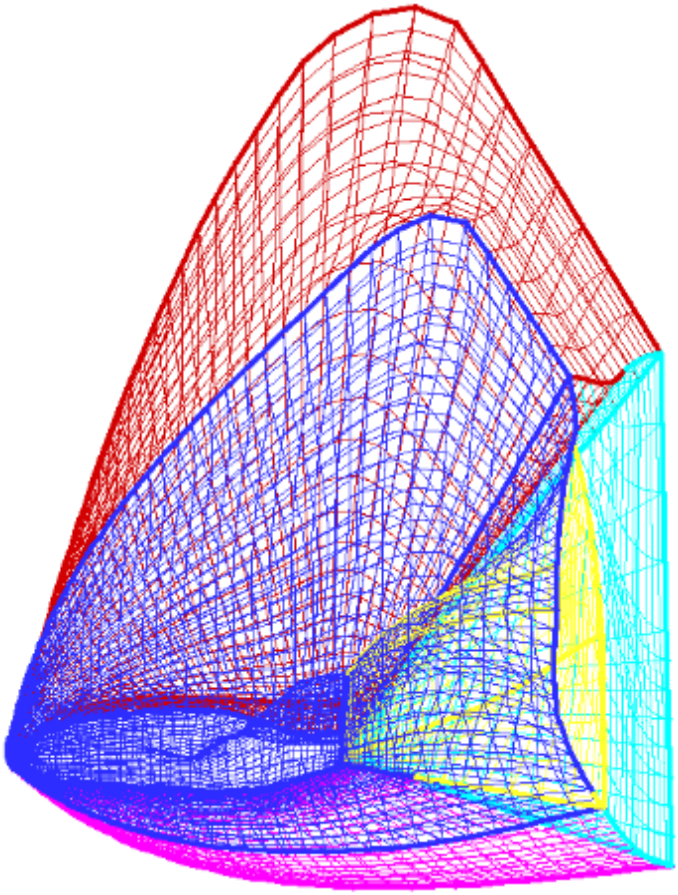

Abbildung 4.4: Netz um die X-38, durch Zellverschmelzung von jeweils acht Zellen vergröbert

Colibrinetz mit diesem Verfahren zeigte eine genaue Übereinstimmung der Simulationsergebnisse. Dabei wurde bei der Implementierung des Prototypsystems in [83] das dortige Ergebnis durch Interpolation und Vergleich mit einer Berechnung aus dem URANUS Singleblockverfahren validiert. Die Ergebnisse des Singleblockverfahrens waren zuvor von den Kollegen des IRS anhand von Windkanalmessungen und Vergleichen mit Rechnungen anderer Einrichtungen überprüft worden. Natürlich werden auch die Rechenergebnisse für andere Topologien, vor allem für die X-38, mit Ergebnissen aus Simulationen anderer Institute und vorliegenden Messungen aus dem Windkanal verglichen und von den Strömungsmechanikern auf Plausibilität überprüft.

4.3.3 Lastverteilung

Die Funktion des Lastverteilers läßt sich am besten am Netz der X38 überprüfen, da es aus mehreren Blöcken besteht, die in der Größe stark variieren. Um für

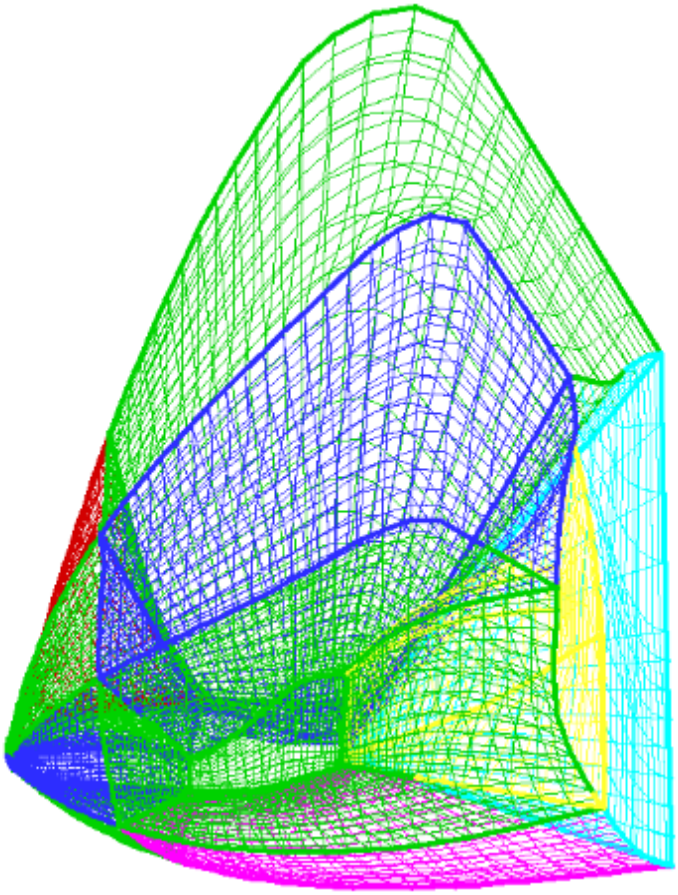

Abbildung 4.5: Netz um die X-38, große Blöcke durch URANUS zur Vermeidung von Lastungleichgewicht zerlegt

den Test nicht allzuviele Prozessoren belegen zu müssen wurde das vergröberte X38-Netz (Halbkörper) mit insgesamt 68.712 Zellen in sechs Blöcken verwendet, so wie es auch in Abbildung 4.4 zu sehen ist. Hier enthält der größte Block (blau) 32.256 und der kleinste Block (gelb) 1.176 Zellen. Wiederum ist der relativ kleine aber dicht vernetzte Frontblock über der Nase des Raumgleiters im Bild nicht erkennbar.

Für den Test wurde die CRAY T3E verwendet und URANUS für dieses Problem auf neun Prozessoren gestartet. Noch vor dem tatsächlichen Einlesen der gesamten Netzgeometrie liest URANUS die Größe der Netzblöcke ein und zerlegt die Blöcke, die allein durch Ihre Größe zu einem Lastungleichgewicht führen würden, in entsprechend viele Teile. Dadurch entstehen aus den sechs ursprünglichen Blöcken elf, da der blaue Block in vier sowie der rote und violette Block jeweils in zwei neue Blöcke zerlegt werden. Das sich ergebende Netz nach der Blockteilung ist in Abbildung 4.5 dargestellt. Die sich neu ergebenden Blöcke sind entweder in der bisherigen Blockfarbe oder grün eingefärbt. An dem in der

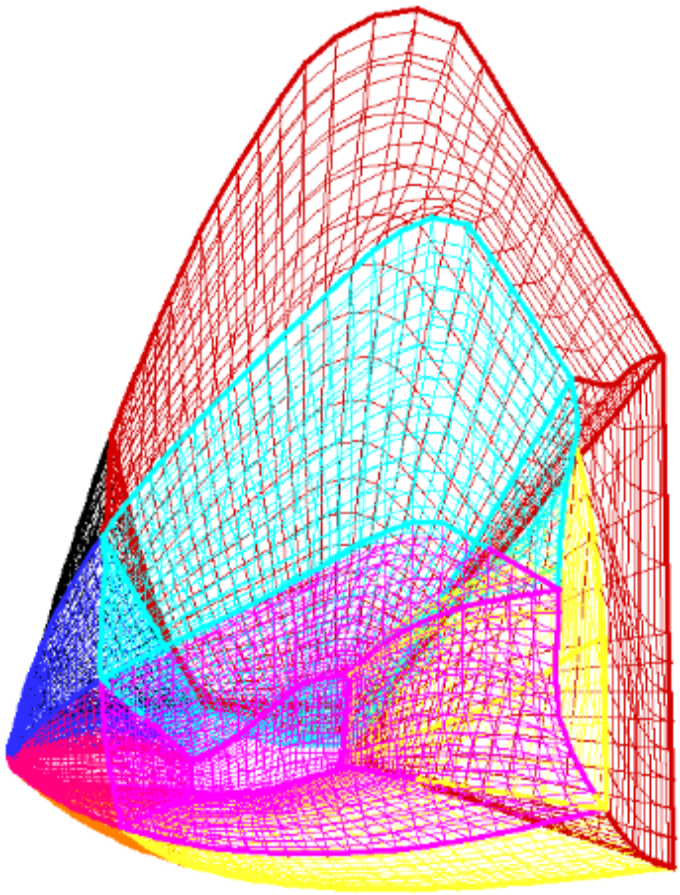

Abbildung 4.6: Netz um die X-38, nach Lastverteilung

Darstellung nicht erkennbaren Frontblock ändert sich nichts.

Diese elf nun verfügbaren Blöcke werden dann vom Lastverteiler den verfügbaren neun Prozessoren zugeordnet und die Blockdaten von URANUS entsprechend auf die Prozessoren verteilt. Dann erst werden von den Prozessoren die Topologiedaten der jeweils zugeordneten Blöcke eingelesen. Das Ergebnis der Lastverteilung zeigt Abbildung 4.6, wobei gleichfarbige Blöcke vom selben Prozess verarbeitet werden. Im einzelnen sind dies zwei Prozesse (in der Abbildung rot und gelb eingefärbt), die jeweils zwei Blöcke zu behandeln haben, während die anderen Prozesse jeweils einen Block simulieren. Der Prozess mit der größten Last hat dabei insgesamt 9198 Zellen, der Prozess mit der geringsten Last 6174 Zellen zu verarbeiten. Die durchschnittliche Last pro Prozessor beträgt 7765 Zellen. Dies bedeutet in diesem Fall ein Lastungleichgewicht von 18%. Das heißt, dass gegenüber einer völlig ausgeglichenen Zerlegung 18% mehr Zeit für die Simulation benötigt wird. Natürlich wäre es besser, wenn der Lastverteiler diese voll ausgeglichene Verteilung erreichen würde, aber dies stellt ein erhebliches Problem dar, da wir im strukturierten Fall ja nicht einfach einzelne Zellen zuteilen können,

sondern es gilt die quaderartige Struktur der Blöcke zu erhalten. Außerdem besitzt der Programmteil für die Blockzerteilung keinerlei Wissen über die globale Topologie der Blöcke und eine mögliche Verteilung. Dies wäre aber notwendig, um eine "bessere" Lastverteilung zu erreichen. Dazu müsste aber das Zerteilungsverfahren mit dem Lastverteiler interagieren, was von den Programmbibliotheken aber im Moment leider nicht vorgesehen ist.

4.3.4 Leistung

Im folgenden Kapitel wird die erzielte Rechenleistung samt Speedup und Scaleup für die verwendeten Supercomputerplattformen dargestellt. Gleich zu Beginn ist aber zu erwähnen, dass das Programm nicht für eine der Plattformen oder auch allgemein optimiert wurde. Allerdings wurde der ursprüngliche sequentielle URANUS Programmcode für die Verwendung auf Vektorcomputern entwickelt. Dies macht sich heute noch auf Vektorsystemen positiv bemerkbar, wohingegen cachebasierte Systeme ohne eine weitergehende Unterstützung, wie zum Beispiel Softwarepipelining, etwas darunter zu leiden haben.

Für die Leistungsmessungen wurde die erwähnte Netzhierarchie für den Colibri Wiedereintrittskörper verwendet. Dieses Netz ist einerseits noch recht einfach aufgebaut, bietet aber bereits die Problematik unterschiedlicher Blockgrößen und auch unterschiedlicher Indexrichtungen auf den verschiedenen Blöcken. Vier der fünf Blöcke besitzen die gleiche Größe, während einer, der Frontblock, genau doppelt so viele Zellen enthält wie die anderen. Das Ausgangsnetz besitzt 24000 Zellen. Daraus wurden Netze mit 192000 und 1536000 Zellen, jeweils durch halbieren jeder Zelle in jeder Richtung, erstellt. Außerdem existiert ein Netz mit 5120000 Zellen, bei dem allerdings alle Blöcke die gleiche Größe haben. Diese andere Größenverteilung wurde notwendig, da das Netz mit 1,5 Mio. Zellen für einige (Metacomputing-)Tests zu klein, das nächstfolgende in der Reihe mit 12,3 Mio. Zellen aber auf einigen der derzeit verfügbaren Systemen nicht mehr handhabbar gewesen wäre.

Speedup

Mit dem Speedup wird die Beschleunigung eines parallelen Programms untersucht, wenn es auf immer mehr Prozessoren eines Parallelrechners verteilt wird. Dabei bleibt das zu lösende Problem dasselbe, das gilt insbesondere auch für die berechnete Problemgröße. Eine genauere Definition des Speedup und eine detaillierte Einführung findet sich zum Beispiel in [30]. Die erwähnte konstante Problemgröße bringt aber auch einige Probleme mit sich. Selbst das Netz mit 24000 Zellen erfordert mehr Speicher als auf einem Prozessor der T3E zur Verfügung steht. Damit ist ein Vergleich mit einem Einprozessorlauf gar nicht möglich. Weiterhin erfordert das numerische Verfahren, dass der Block, der im Endeffekt auf

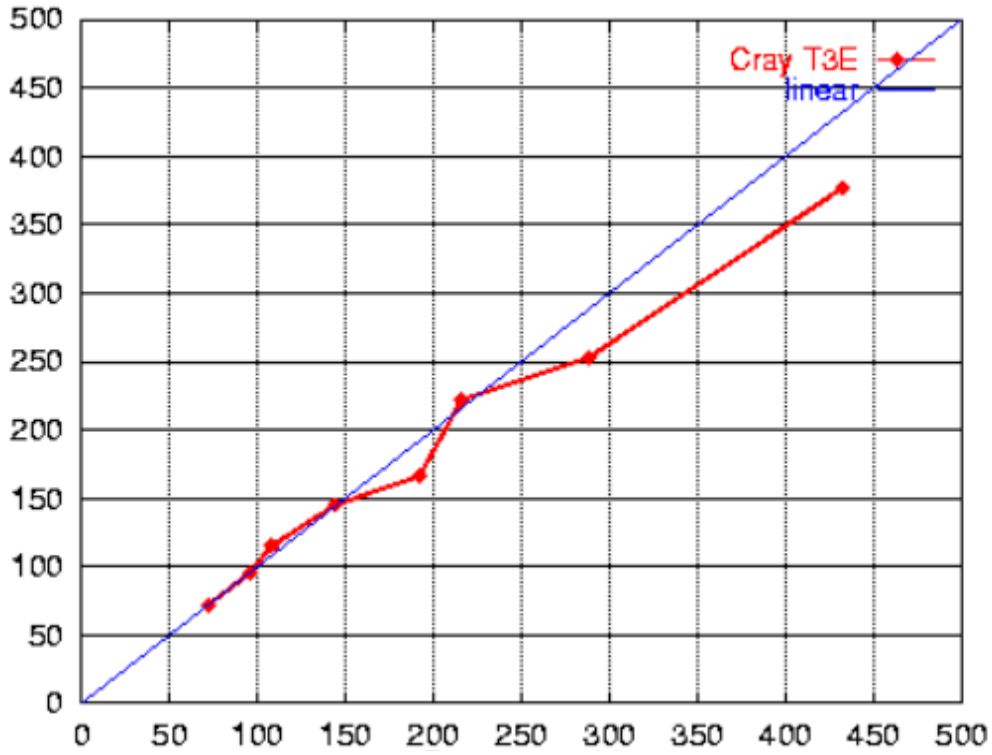

Abbildung 4.7: Speedup des Multiblock URANUS Verfahrens auf der Cray T3E in Stuttgart

einem Prozessor berechnet wird, mindestens sechs Zellen in Festkörperrichtung aufweist. Da wir auf die Zerteilung und Anordnung der Blöcke von außen keinen Einfluß nehmen, müssen die Blöcke also aus mindestens 216 Zellen bestehen. Damit könnte das 24000er Netz auf höchstens 54 Prozessoren verteilt werden. Dies erlaubt dann aber leider keine Aussage für den Speedup auf großen Prozessorzahlen. Daher wurde hier der Speedup mit dem 192000 Zellennetz bestimmt, das allerdings mindestens 72 Prozessoren benötigt, um überhaupt genügend Speicher für die Berechnung dieser Problemgröße zur Verfügung zu haben. Die in Abbildung 4.7 dargestellte Meßreihe wurde auf den Lauf mit 72 Prozessoren bezogen. Die Prozessorzahlen für die dargestellten Messpunkte wurden so gewählt, um eine möglichst perfekte Lastverteilung zu erreichen. Trotzdem zeigen sich Schwankungen und teilweise auch Messungen mit superlinearem Speedup. Der superlineare Speedup ist von Messungen mit URANUS auf der T3E bereits bekannt [4] und tritt in einem begrenzten Bereich, wie hier bei der Verwendung von 120 bis 210 Prozessoren, auf. Ursache hierfür ist hauptsächlich ein durch die kleiner werdenden Problemgrößen auf jedem Prozessor sich verbesserndes Cacheverhalten. Die Ursache für die Schwankungen sind letztendlich auch in einer mehr oder weniger effizienten Nutzung der Caches und Streamsbuffer begründet. Der Grund hierfür

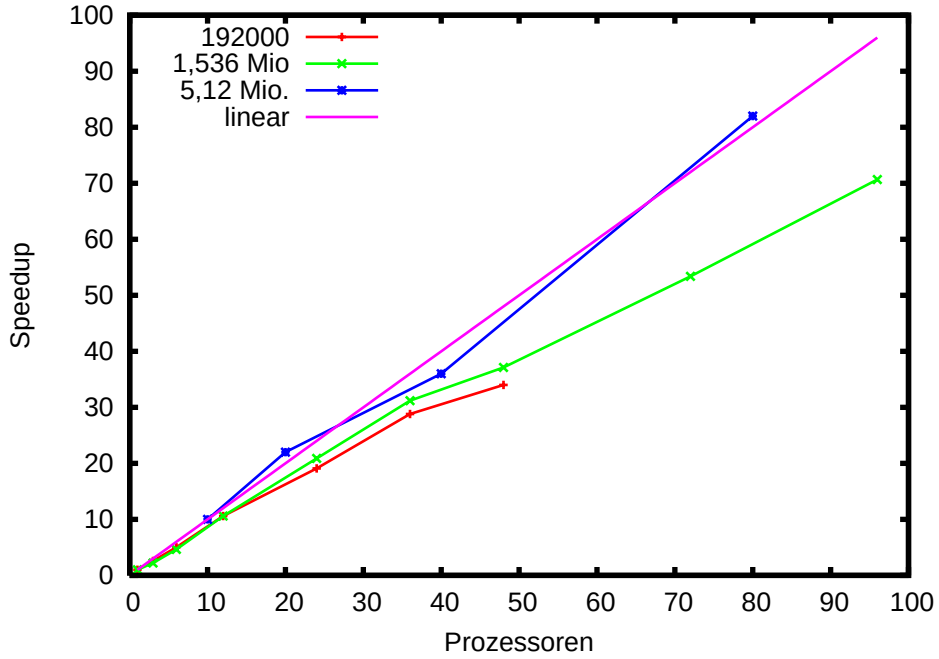

Abbildung 4.8: Speedup des Multiblock URANUS Verfahrens auf dem Cray Strider mit unterschiedlichen Problemgrößen

ist allerdings nicht die sich ändernde Problemgröße sondern die Problemverteilung. Mit der sich ändernden Prozessorzahl ändert sich auch die Anzahl der Teile, in die ein Block zerlegt werden muß. Und je nach der Anzahl der neu entstehenden Blöcke ändert sich die Anzahl der Zellen in den drei Raumrichungen auf diesen und damit auch die Aufteilung auf die Schleifen im Programm und damit auch das Cacheverhalten. Zwar versucht der Algorithmus, wie im vorigen Kapitel beschrieben, eine möglichst als ausgewogen bekannte Verteilung in den Raumrichtungen zu erreichen, dies ist aufgrund der zu generierenden Blockzahlen und der anderen erwähnten Nebenbedingungen aber leider oft nicht möglich. Der sich ergebende Speedup von 380 für den Lauf auf 432 Prozessoren ergibt eine parallele Effizienz von 88% verglichen mit dem Lauf auf der kleinstmöglichen Partition mit 72 Prozessoren.

Die T3E in Stuttgart wurde im Jahr 2004 durch den Strider ersetzt. Neben einer erheblich leistungsfähigeren CPU besitzt nun ein Knoten 4 GB Hauptspeicher, genug um auch größere Fälle auf einem Knoten berechnen zu können. Bis auf das 5,12 Mio.-Zellen Netz können die zur Messung verwendeten Probleme nun auf einem Knoten berechnet werden. Für dieses Netz mussten auf dem Strider mindestens 5 Knoten verwendet werden. Die Ergebnisse beziehen sich bei diesem Netz dann auf den Referenzlauf mit 10 Prozessen auf 10 Prozessoren verteilt auf 10 Knoten, da der Lauf auf 5 Knoten einen Zeitmehrbedarf von etwa 20% zeigte.

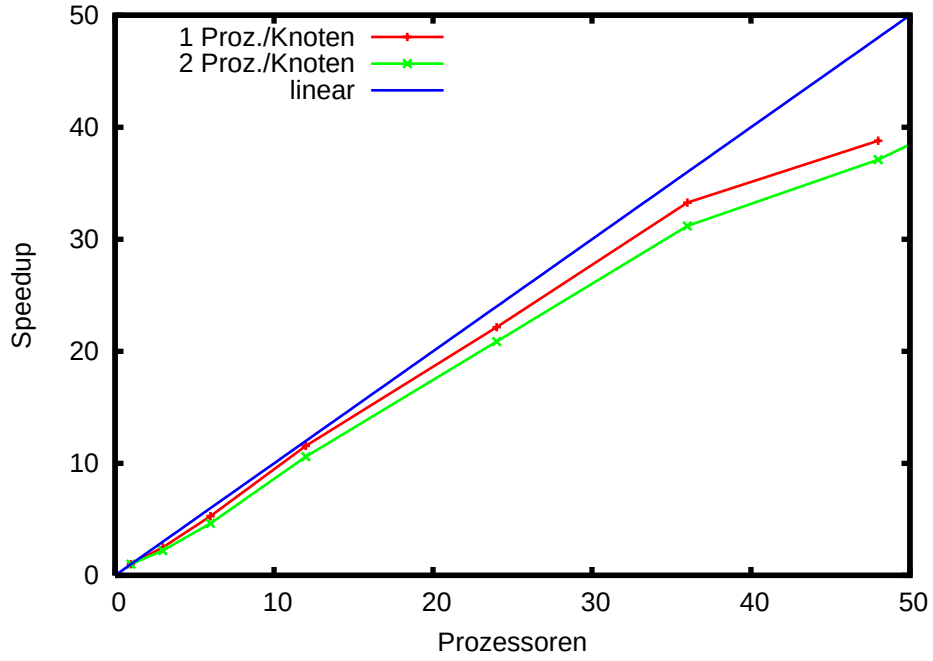

Abbildung 4.9: Speedup des Multiblock URANUS Verfahrens auf dem Cray Strider 0 mit unterschiedlichen Prozesszahlen pro Knoten

Abbildung 4.8 zeigt die erreichten Speedups auf dem Strider für die unterschiedlichen Netzgrößen. Hier fällt insbesondere der leicht superlineare Speedup für das größte Netz auf. Die Gründe hierfür sind vor allem in einer anscheinend günstigeren Aufteilung der Blöcke bei der Verwendung von 20 und 80 Prozessoren im Zusammenhang mit einer besseren Nutzung des Cache zu suchen. Weiter ist ab 24 Prozessoren der Speedup für den kleinen Fall mit 192.000 Zellen geringer als der für den mittleren Fall. Hier beginnt das Netz für den Prozessor zu klein zu werden, der Aufwand für die Kommunikation wird im Verhältnis zum Rechenaufwand größer und macht sich in der Laufzeit bemerkbar. Es sei noch bemerkt, dass die Effizienz des parallelen Multiblockverfahrens für 72 und 96 Prozessoren mit dem 1,536 Mio. Zellennetz noch 74% beträgt und im Vergleich zu 48 Prozessoren mit 77% Effizienz über steigende Prozessorzahlen nur geringfügig abnimmt.

Da der Knoten nur zwei Prozessoren enthält, bietet es sich an, auszuprobieren, wie sich URANUS auf einem entsprechenden System verhält, das nur einen Prozessor pro Knoten besitzt. Abbildung 4.9 zeigt die Ergebnisse dieser Messungen. Bei der Verwendung von drei Prozessoren ist fast noch kein Unterschied festzustellen, während sich ab 6 Prozessoren leichte Vorteile für den Einprozessorknoten ergeben. Diese sind aber zu gering, als dass ein Einprozessorsystem in der vorgestellten Konfiguration den höheren Preis im Verhältnis zur erreichten Rechenleistung rechtfertigen würde. Offensichtlich reicht bei den gemessenen Pro-

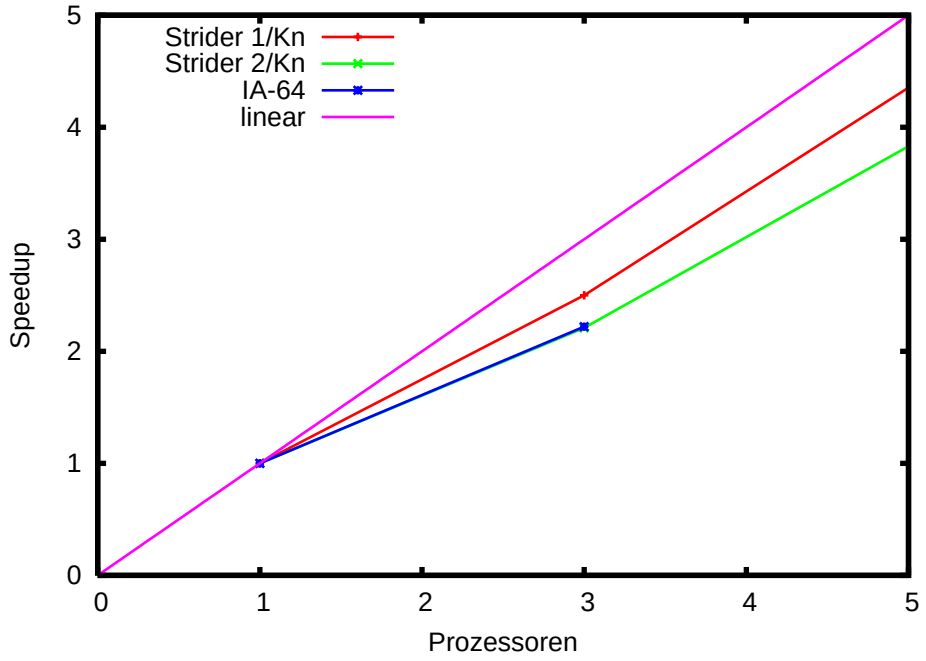

Abbildung 4.10: Speedup des Multiblock URANUS Verfahrens im Vergleich AMD Opteron, IA-64

zessorzahlen die verfügbare Bandbreite aus, um die notwendige Kommunikation effizient abzuwickeln. Natürlich spielt hier auch eine Rolle, dass die Kommunikation zwischen den beiden Blöcken auf einem Knoten natürlich lokal abgehandelt werden kann und dass sich auf dem Strider bei der Verwendung der beiden Prozessoren auf dem Knoten nicht nur die Rechenleistung sondern auch die Speicherbandbreite verdoppelt. Allerdings unterstützt die auf dem Strider derzeit verwendete MPI-Bibliothek, MPICH, nicht den direkten Datentransfer über den Hauptspeicher, wenn zwischen den beiden lokalen Prozessen eine Kommunikation stattfindet. Stattdessen wird dieser Vorgang von der Netzwerkschnittstelle abgehandelt.

Dies ist wiederum auf einem Itaniumsystem nicht der Fall, hier arbeiten alle Prozessoren eines Knotens über einen Bus auf dem gemeinsamen Speicher und somit steigt die Speicherbandbreite nicht an, wenn man zwei oder vier Prozessoren statt einem verwendet. Allerdings kann auf dem hier verwendeten Vierprozessorsystem dann die Kommunikation über den Hauptspeicher erfolgen.

Um diese Auswirkungen zu überprüfen, wurden Messungen auf solch einem System durchgeführt. Abbildung 4.10 zeigt das Ergebnis. Überraschenderweise ergibt sich zwischen Strider und IA-64 kein Unterschied, sofern man nur zwei Striderkknoten verwendet. Wenn man das Experiment mit drei Striderknoten und nur einem Prozess pro Knoten durchführt, liegt der Strider vorne. Hieraus

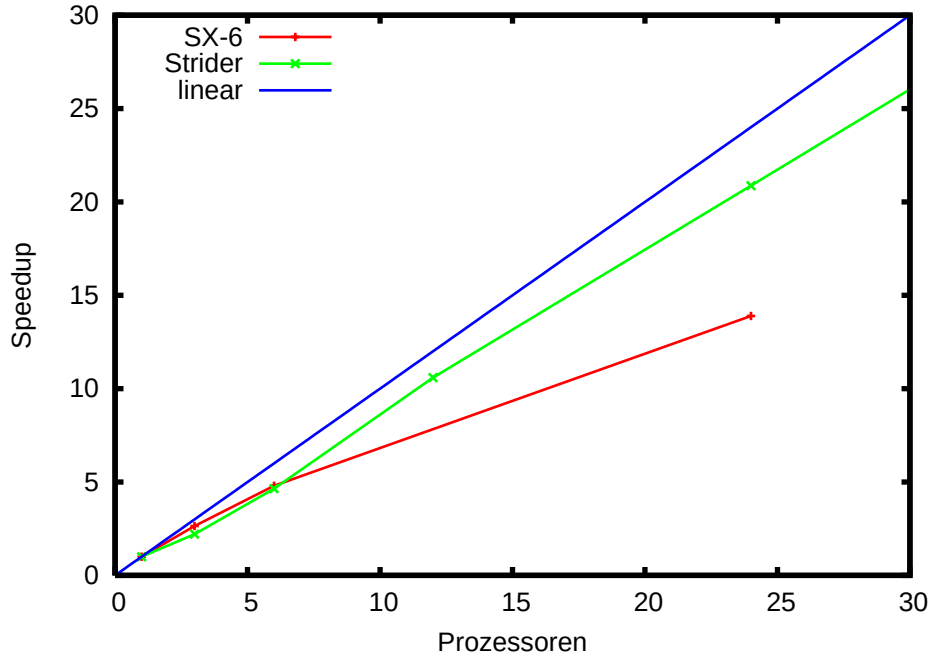

Abbildung 4.11: Speedup des Multiblock URANUS Verfahrens auf der NEC-SX6

ergeben sich zwei mögliche Folgerungen: entweder der Strider hat doch ein Problem mit dem Speichermanagement, oder der konkurrierende Zugriff von zwei Prozessen auf das Netzwerkinterface führt noch zu Problemen. Letzteres scheint der wahrscheinlichere Fall zu sein, denn wenn sich zwei Prozesse die Netzwerkschnittstelle teilen, kann es vorkommen, dass der eine sendet, für den anderen aber Daten einlaufen. Dies muss vom Betriebssystem und vor allem auch von der proprietären Netzwerkhardware (Myrinet) und auch deren Treibern sauber unterstützt werden, damit es hier nicht zu Problemen kommt. Da sowohl das Betriebssystem als auch die Netzwerkhardware ursprünglich für Systeme mit einem Prozessor entwickelt wurden, ist die Wahrscheinlichkeit sehr hoch, dass sich hier noch Probleme verbergen, die zwar die Funktionalität nicht beeinflussen, aber Nachteile bei der erzielbaren Leistung zeigen. Wie die Ergebnisse zeigen, gleicht der Vorteil der hohen Speicherbandbreite auf dem Strider gegenüber dem IA-64 den Nachteil beim Netzwerkzugriff gerade eben wieder aus. Leider steht im Moment kein entsprechendes vierprozessor Opteronsystem für einen fairen vergleichenden Test zur Verfügung.

Nun wollen wir noch einen Blick auf die SX-6 werfen. Hier, in Abbildung 4.11, zeigt sich im Vergleich zum Strider ein auf den ersten Blick erstaunliches Bild. Der Speedup ist ab ca. 6 Prozessoren erheblich geringer als auf dem Strider und die Effizienz liegt auf 24 Prozessoren bei gerade mal 60%. Bei genauerer Untersuchung ist die Ursache schnell geklärt. Durch die Aufteilung des Rechengebiets

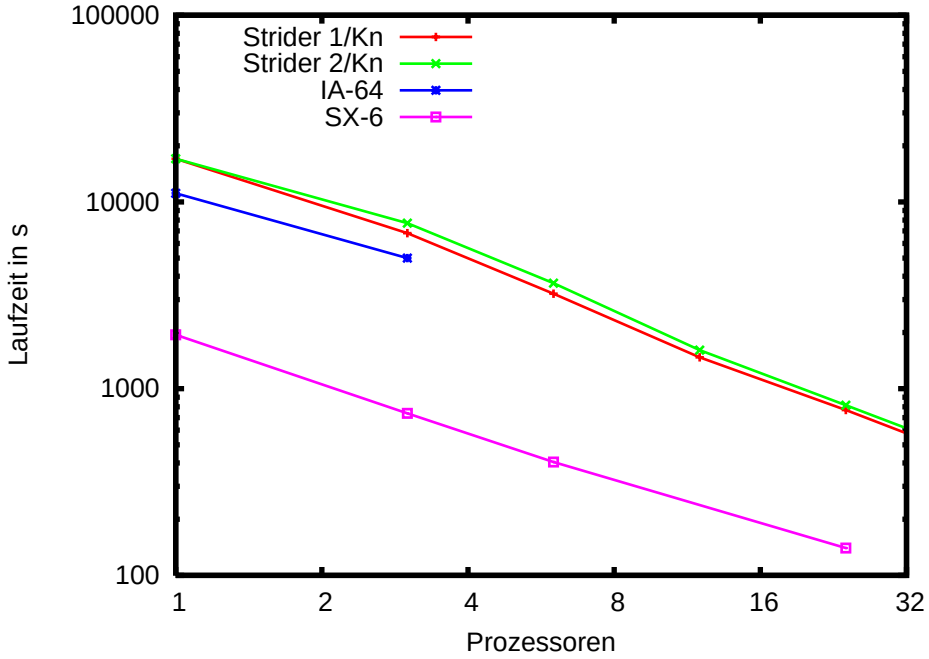

Abbildung 4.12: Laufzeitvergleich des Programms auf Strider, IA-64 und NEC SX-6

sinkt die Vektorlänge erheblich ab und erreicht bald eine Untergrenze, ab der die Vektorprozessoren eines Vektorrechners nicht mehr richtig ausgelastet werden. Damit sinkt aber deren Effizienz. Der zusätzliche Leistungsverlust ist also nicht im Verfahren begründet sondern liegt daran, dass das Problem für diese leistungsstarke Maschine bereits zu klein wird. Hier muß also unbedingt der Scaled Speedup oder Scaleup betrachtet werden.

Zuvor wollen wir uns aber noch mit den absoluten Laufzeiten des Programms auf den einzelnen Systemen auseinandersetzen. Denn der Speedup ist ja nur die halbe Wahrheit, wenn man Systeme miteinander vergleicht. Hierzu sind unbedingt auch die absoluten Laufzeiten eines Programms relevant. Diese sind in Abbildung 4.12 aufgezeigt. Sofort fällt auf, dass die SX-6 natürlich erheblich kürzere Laufzeiten bei der gleichen Anzahl der verwendeten Prozessoren aufweist. Um auf dem Strider die gleiche Laufzeit wie auf den 3 Prozessoren der SX zu erreichen muss man hier fast 28 Prozessoren einsetzen. Es ist also mehr als die neunfache Prozessorzahl für die gleiche Leistung notwendig. Nun kann man natürlich einwenden, dass die SX noch auf dem gemeinsamen Speicher arbeitet und sich das Problem dort nicht in der gleichen Art und Weise skalieren läßt wie auf dem Strider. Aber auch ein Versuch mit dem großen Netz auf 10 Prozessoren SX zeigt, dass zur Erzielung der gleichen Leistung mehr als 120 Prozessoren auf dem Strider notwendig wären. Dies liegt daran, dass die großen Blöcke auf der SX erheblich

System	Preis CPU-Stunde	Laufzeit [s]	CPU-Stunden	Kosten Euro
Strider	ca. 1,–	7710	6,425	ca. 6,40
Itanium	ca. 5,–	5033	4,194	ca. 21,–
SX-6	ca. 6,–	738	0,615	ca. 3,70

Tabelle 4.5: Vergleich der Rechenkosten

besser laufen als die etwas kleineren und der Strider auch noch einen ordentlichen Verlust durch das im Verhältnis etwas schwache Netzwerk aufweist. Das Verhältnis zwischen Strider und SX könnte sich wieder etwas zugunsten des Strider verbessern, wenn man auf der SX noch mehr Knoten verwenden möchte oder muß. Aber besser als etwa 1 zu 10 wird es sehr wahrscheinlich nicht werden. Für obige Berechnung wurde übrigens das Rechenzeitergebnis der SX in die Hyperbel eingetragen, die durch die gemessenen Werte für 20 und 80 Strider-Prozessoren auf dem großen Netz aufgespannt wird. Die Messdaten für die genannten Fälle finden sich bei der Berechnung des Scaleups auf den nächsten Seiten.

Aber auch das IA-64 System benötigt in etwa nur zwei Drittel der Zeit des Striders bei gleicher Problemgröße und Prozessorzahl.

Zur Kosteneffizienbetrachtung muss man diese Verhältnisse noch mit dem Preis der Systeme in Relation setzen. Die dazu verwendeten Kosten für eine CPU-Stunde wurden aus den Verrechnungspreisen des HLRS für die benutzten Systeme, Preisstand Dezember 2004, gebildet. Auf Basis der Laufzeit für die Rechnungen auf drei Prozessoren wurden die Kosten für diese Simulationen errechnet und in Tabelle 4.5 eingetragen. Wie zu sehen ist, kann das Itaniumsystem hier bei weitem nicht mit dem Strider mithalten und überraschenderweise bietet die SX-6 nicht nur die kürzeste Laufzeit sondern auch noch den geringsten Preis für die durchgeführte Rechnung.

Um den obigen Vergleich zwischen Strider und SX-6 fortzuführen setzen wir hier nochmals die CPU-Preise für das Leistungsverhältnis von eins zu 10 zwischen Strider und SX-6 an. Damit besitzt die SX-6 mit dem verwendeten URANUS-Programm einen Kostenvorteil von etwa 40% gegenüber dem Strider.

Scaleup

Im Gegensatz zum Speedup wird beim Scaleup (oder auch Scaled Speedup) mit der Prozessorzahl auch die Problemgröße entsprechend erhöht. Eine genauere Definition findet sich zum Beispiel wieder in [30]. Da beim URANUS Verfahren die Rechenzeit linear mit der Zahl der Zellen steigt (es sich also um ein Verfahren

Netzgröße	Blockzahl	Prozessor-zahl	Simulationszeit [s]	Effizienz
24.000	5	18	255,3	1,0
192.000	5	144	285,7	0,893

Tabelle 4.6: Scaleup auf der Cray T3E

der Ordnung O(n), wenn n die Zellenzahl beschreibt, handelt), erhöht sich hier mit der Prozessorzahl entsprechend die Anzahl der Zellen oder umgekehrt. Damit bleibt aber die Größe der zu lösenden Teilprobleme pro Prozessor konstant. Somit können die Einflüsse von Cacheeffekten und ähnlichem, die durch die Änderung der lokalen Problemgröße bedingt sind, größtenteils ausgeschlossen werden. Für die Scaleupmessung auf der T3E wurde das 24000 Zellen Colibrinetz als Basis herangezogen und auf 18 Prozessoren gerechnet. Das nächst größere Netz war dann das mit 192000 Zellen auf 144 Prozessoren. Das erzielte Ergebnis ist in Tabelle 4.6 zusammengestellt. Es zeigt sich, dass wir hier für den größeren Fall eine Effizienz von knapp 90% gegenüber dem kleineren erreichen. Allerdings hat sich die Konfiguration der lokalen Problemgröße doch geändert, da der Lastverteiler für die große Zellenzahl die Blöcke natürlich in einer anderen Art und Weise zerlegt. Damit können leider die Einflüße des Caches und der Streamsbuffer wieder nicht ausgeschlossen werden. Ein Scaleup auf einer größeren Prozessorzahl konnte auf der T3E nicht mehr gemessen werden. Dafür wären mindestens 576 Prozessoren notwendig gewesen, da das Netz mit 192000 Zellen wie bereits erwähnt, mindestens 72 Prozessoren benötigt. Wir können aber immerhin aus dem Ergebnis die Annahme ableiten, dass auch die neue URANUS Multiblockversion zwar mit gewissen Verlusten, aber doch bis auf mehrere tausend Prozessoren verwendbar sein wird, wenn die Probleme groß genug sind.

Auch auf dem Strider wurde eine Scaleupmessung durchgeführt. Durch die großzügige Speicherbestückung konnte hier sogar eine Scaleupmessung mit drei verschiedenen Netzvarianten durchgeführt werden. Hierzu wurden die Netze mit 192.000, 1,536 Millionen sowie 5,12 Millionen Zellen verwendet. Vom kleinen auf das mittlere Netz wurde die Zellzahl auf jedem Block in jeder Richtung verdoppelt, das mittlere Netz ist also genau achtmal größer als das kleine. Das große Netz wiederum ist 3,3 mal so groß wie das mittlere und besitzt ein etwas anderes Layout, da hier alle Netzblöcke die gleiche Zellzahl besitzen. Tabelle 4.7 zeigt die erzielten Ergebnisse. Zwischen dem kleinen und dem mittleren Fall ist die Leistungseinbuße nur gering, während sich zum großen Fall hin schon ein merkliches Nachlassen der Effizienz auf 89 % bemerkbar macht. Die Blockgröße pro Prozess unterscheidet sich zwischen dem mittleren und großen Netz bis auf eine Ausnahme nicht. Allerdings ist hier zu berücksichtigen, dass es durch die Zerlegung beim

Netzgröße	**Blockzahl**	**Prozessor-zahl**	**Simulationszeit [s]**	**Effizienz**
192.000	5	3	794	1,0
1,536 Mio.	5	24	816	0,97
5,12 Mio.	5	80	897	0,89

Tabelle 4.7: Scaleup auf dem Cray Strider 0

Netzgröße	**Blockzahl**	**Prozessor-zahl**	**Simulationszeit [s]**	**Effizienz**
1,536 Mio.	5	3	710	1,0
5,12 Mio.	5	10	614	1,16

Tabelle 4.8: Scaleup auf der NEC SX-6

großen Netz nunmehr eine größere Anzahl Blöcke gibt. Dadurch existieren nun aber Blöcke, die auf jeder Seite einen Nachbarn haben, also auf allen sechs Blockseiten kommunizieren müssen. Dies ist beim mittleren Netz noch nicht der Fall, hier ist mit maximal fünf Nachbarn eine Kommunikation notwendig. Berücksichtig man das, kann man davon ausgehen, dass der Scaleup für ein noch größeres Netz nicht mehr ganz so stark nachlassen wird. Man kann also auf etwa 2000 Prozessoren noch mit einer Effizienz von mindestens 80 % rechnen, wenn das Kommunikationsnetzwerk des verwendeten Rechners leistungsfähig genug ist.

Schauen wir uns zum Schluß noch einmal die SX-6 an. Hier wurde die Scaleupmessung nur mit dem mittleren und dem großen Fall durchgeführt. Die Ergebnisse in Tabelle 4.8 sind auf den ersten Blick wieder etwas überraschend, denn der Scaleup ist superlinear und das, obwohl wir den großen Fall auf 10 Prozessoren berechnen und so eine Kommunikation über das externe (proprietäre) Netzwerk stattfinden muß, während sich der Datenaustausch im mittleren Fall, berechnet auf drei Prozessoren, mit der schnellen Kommunikation über den gemeinsamen Speicher durchführen lässt. Ein Grund hierfür ist, dass die Blöcke im aufgeteilten großen Netz mit einer Ausnahme doppelt so groß sind wie im ersten. Dies führt zu einer erheblich größeren Vektorlänge, da das dreidimensionale Netz für den Vektorrechner linearisiert ist. Dadurch steigt die Effizienz des Vektorprozessors.

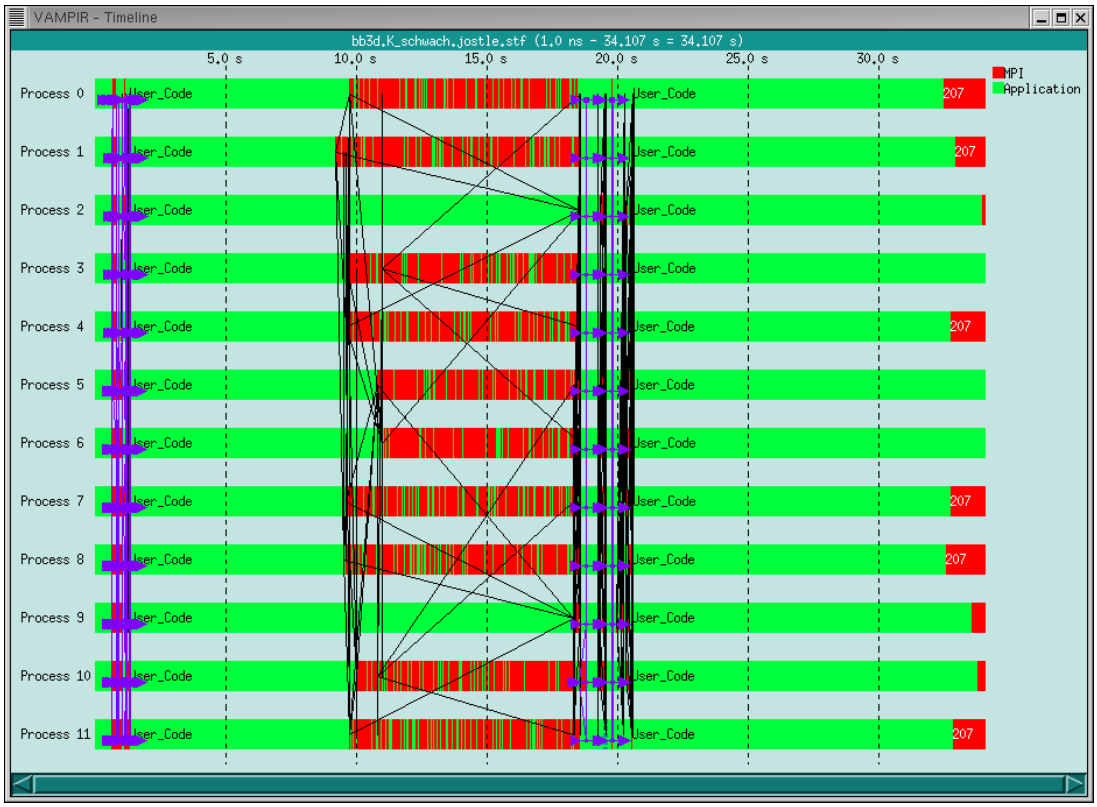

Abbildung 4.13: Darstellung eines kompletten Programmlaufs mit Vampir

Untersuchung des Kommunikationsverhaltens

Nachdem wir uns nun Speedup und Scaleup angesehen haben, wollen wir das Kommunikationsverhalten des parallelen Programms noch genauer ansehen. Dazu wurde das Programm mit VampirTrace [14] instrumentiert. Am Ende eines Laufs des instrumentierten Programms wird ein Tracefile geschrieben, das die Kommunikationsinformationen mit einem Zeitstempel enthält. Auf Basis dieses Tracefiles wird nun mittels Vampir die Kommunikationsstruktur graphisch sichtbar. Die Untersuchungen, die in diesem Abschnitt vorgestellt werden, wurden auf der SX-6 in Stuttgart mit Programmläufen auf sechs und zwölf Prozessoren durchgeführt. Abbildung 4.13 zeigt den Gesamtablauf des Programms auf zwölf Prozessoren, wobei die Iterationszahl auf zwei begrenzt wurde, damit das Tracefile noch eine einigermassen handhabbare Größe besitzt. Jedem der zwölf Prozesse ist ein eigener Balken zugeordnet, der im Verlauf von links nach rechts den Ablauf des Programms über der Zeit darstellt. Rote Bereiche im Balken zeigen dabei den Aufruf und das Verweilen in MPI-Aufrufen, grüne Bereiche den restlichen Programmcode, der von Vampir nicht weiter dargestellt wird. Die Pfeile zeigen den Verlauf der Kommunikation, schwarze Pfeile für eine Punkt zu Punkt Kommunikation, wie zum Beispiel mittels MPI_Send und MPI_Recv, blaue Pfeile für

Block (bisher)	1		2		3				4		5	
Block (neu)	1	6	2	7	3	8	9	10	4	11	5	12
Prozess	7	0	8	1	5	6	9	2	10	3	11	4

Tabelle 4.9: Verlauf der Blockauf- und -umverteilung für den beschriebenen Fall

kollektive Kommunikation an der mehrere oder alle Prozesse partizipieren, wie zum Beispiel einen Broadcast (MPI_Bcast).

Betrachten wir den Ablauf genauer, zerfällt der Gesamtablauf in mehrere Teile. Zu Beginn ist nach einem kurzen grünen Bereich gleich ein Teil mit erheblicher, vor allem kollektiver, Kommunikation zu sehen. Mittels dieser Kommunikation werden die globalen Parameter verteilt und die Blockgrößen bekanntgemacht. Die Ergebnisse der folgenden lokalen Blockzerlegung werden den Nachbarn mitgeteilt, damit diese ihre Kommunikationsparameter entsprechend anpassen können. Auch die Kommunikation der Lastverteilung darauf und die Umverteilung der Parameter der einzelnen Blöcke läuft in diesem Kommunikationsbereich ab. In dem darauf folgenden großen grünen Bereich werden die Netzdaten für die Blöcke von den jeweiligen Prozessen eingelesen und die Ableitungen und festen Größen der Netzgeometrien berechnet. Für die Berechnung der Werte im Überschneidungs- oder Halobereich werden dann einige der berechneten Werte an den jeweiligen Blockrändern noch ausgetauscht. Dies findet im großen Bereich in etwa zwischen Sekunde 9 und 18 statt. Dieser Bereich ist nur deshalb so groß, weil Prozeß 2 und 9 für den Präprozessingteil mehr Zeit benötigen. Dies wirft die Frage nach der Ursache hierfür auf. Um diese jedoch genauer bestimmen zu können, müssen wir zuerst den Verlauf der Blockauf- und -umverteilung nachvollziehen, damit wir wissen, aus welchem ursprünglichen Block die neuen Blöcke hervorgegangen sind und welcher Prozess welchen Block berechnet. Dies ist in Tabelle 4.9 dargestellt.

Wie sich jetzt zeigt, berechnen die Prozesse 2 und 9 die Blöcke 10 und 9, die beide aus dem zentralen Mittelblock des Netzes, Block 3, hervorgegangen sind. Die Nachbarschaftsbeziehungen und ein etwa daraus resultierender Mehraufwand an den Rändern kommen als Ursache nicht in Frage, da sonst auch die Prozesse 5 und 6 diesen Mehraufwand zeigen müssten, da sie ebenfalls aus dem Mittelblock hervorgegangen sind und ähnliche Nachbarschaftsbeziehungen besitzen. Es muß also am anderen großen Teil, dem Lesen der Blockdaten, liegen. Dafür spricht auch, daß die Zeit, die Prozeß 2 und 9 bis zur Kommunikation brauchen fast doppelt so lange ist, wie die der restlichen beiden Prozessoren. Nun ist die Datei des mittleren Blocks aber genau doppelt so groß, wie die der anderen Blöcke. Auch hier müßten dann aber wieder auch Prozeß 5 und 6 dieses Verhalten zeigen. Da aber der ehemalige mittlere Block im Gegensatz zu den anderen auch in

der dritten Dimension geteilt wird und die Daten der Dimension nach abgelegt sind, müssen von den Prozessen, deren Blöcke Teile des mittleren Blocks waren, nur zwei die komplette Datei einlesen. Die zwei anderen können den Lesevorgang nach der Hälfte der Datei beenden. Somit gibt es also zwei Blöcke, die doppelt so viele Daten einlesen müssen, wie die anderen. Interessanterweise scheint es keine Rolle zu spielen, wieviele Prozesse gleichzeitig aus einer Datei Daten einlesen, denn aus der Datei des ehemals mittleren Blocks lesen vier aus den anderen Dateien jeweils nur zwei Prozesse. Ausserdem liegen die Prozesse, die jeweils Daten von einem Block einlesen jeweils gleichmäßig verteilt auf den beiden verwendeten Knoten. Dieses Leseverhalten sollte nochmals genauer untersucht werden, um das Verhalten des Rechnersystems, genauer gesagt seines globalen parallelen Dateisystems, beim konkurierenden Zugriff aus einer verteilten parallelen Applikation heraus besser zu verstehen. Und zwar wie im hiesigen Fall beim formatierten Lesen (und Schreiben) als auch bei unformatierter Ein- und Ausgabe, die auf den SX-Systemen erfahrungsgemäß eine weitaus höhere Leistung bietet.

Nach dem Berechnen und Austauschen der netzabhängigen Parameter beginnt die Simulationsrechnung mit ihren Iterationen. Die folgenden Datenaustauschschritte gehören dann hierzu. Der Ablauf einer Iteration wird im folgenden Abschnitt noch genauer betrachtet. Nach den zwei Iterationen werden dann die Erhaltungsvektorwerte am Rand für die Ausgabe nochmals ausgetauscht, bevor jeder Prozess seinen Block in eine Datei schreibt. Da diese Blöcke alle gleich groß sind, dauert das auch in etwa gleich lange. Kleine Unterschiede haben ihren Grund sicher im Zugriff auf das Dateisystem. Am Ende warten die Prozesse noch aufeinander, da der letzte Aufruf, der die parallele Umgebung beendet (MPI_Finalize), blockierend ist.

Schauen wir uns aber nun eine Iteration genauer an. Abbildung 4.14 zeigt die zweite Iteration, die aus dem Gesamtbild in Abbildung 4.13 quasi herausvergrößert wurde. Eine Iteration beginnt natürlich auf den verschiedenen Prozessen zu unterschiedlichen Zeitpunkten, je nachdem wann die Kommunikation der letzten Iteration beendet ist. Als Ende einer Iteration wurde hier das Beenden der Kommunikation im Löser definiert, also, wenn der letzte Prozess den Datenaustausch beendet hat.

Im ersten Kommunikationsblock nach Beginn der Iteration werden die neu berechneten Erhaltungsvektorwerte ausgetauscht. Interessant ist hier, dass die Zeit bis dorthin auf den einzelnen Prozessen äußerst unterschiedlich ausfällt. Hierfür gibt es zwei Gründe: Nach der Kommunikation wird ein zweiter Lösungsschritt durchgeführt, der je nach Anzahl der notwendigen Subiterationen unterschiedlich lang sein kann. Dies werden wir nachher noch genauer diskutieren. Der zweite Grund ist die Randbehandlung zu Beginn der neuen Iteration, die je nach Position des Blocks im Netz unterschiedlich aufwendig ist. Schön ist hier zu sehen, dass Prozess 2 im Gegensatz zu Prozess 1 und 3 nur einen physikalischen Rand, eine

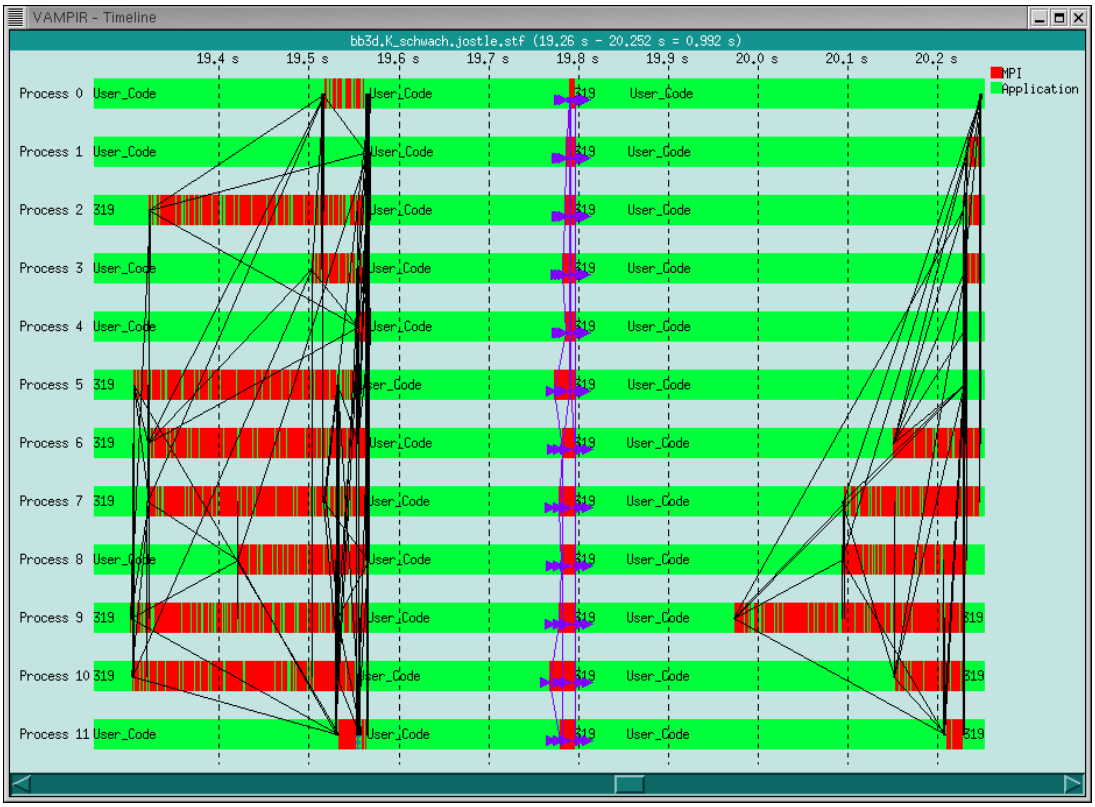

Abbildung 4.14: Darstellung einer Iteration

Fläche, zu bearbeiten und deshalb seine Arbeit erheblich schneller erledigt hat. Außerdem ist eine Randbehandlung ja nur für einzelne Flächen bzw. Kanten und Ecken notwendig, so dass dieser Teil auf einem Vektorrechner relativ ineffizient ist, da der Zugriff auf den Speicher mit großen Sprüngen erfolgt und meist keinen vektorartigen Datentranfer aus dem Speicher erlaubt. Diese Teile brauchen hier daher relativ viel Zeit. Außerdem zeigt sich damit auch, dass das Problem für den Rechner bzw. die Anzahl der verwendeten Prozessoren bereits zu klein ist.

Nach dem Transfer des Erhaltungsvektors werden die Einträge für die Jacobimatrix errechnet. Hier wird auf allen Blöcken über alle Zellen gerechnet. Da alle Blöcke neben der gleichen Größe auch dasselbe Größenverhältnis besitzen, ist der Aufwand und die Berechnungszeit auf allen Blöcken gleich. Die kollektive Kommunikation (blaue Pfeile) wird daher von allen Prozessen auch fast gleichzeitig aufgerufen. Die geringfügigen Unterschiede rühren hauptsächlich daraus, dass die letzte Kommunikation ja nicht gleichzeitig beendet wurde. Die kollektive Kommunikation dient zur Berechnung der globalen Summen und des globalen Maximums für die Residuen der unterschiedlichen Normen. Nach der Residuenberechnung erfolgt der Lösungsschritt, der auf den unterschiedlichen Blöcken wieder unterschiedlich lange dauern kann, da je nach den Werten in der lokalen Matrix unterschiedlich viele Subiterationen zur Lösung des lokalen Systems

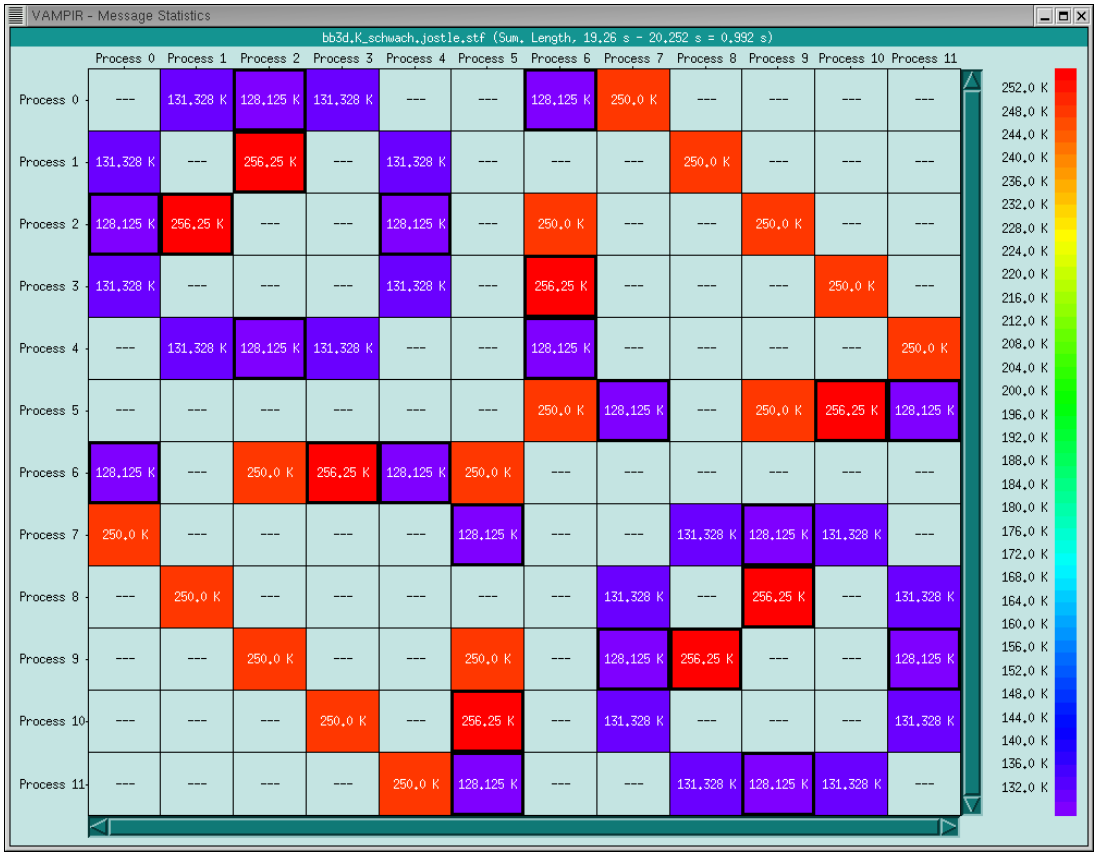

Abbildung 4.15: Kommunikationsvolumen in einer Iteration

bis zur gewünschten Genauigkeit notwendig sein können. Diese Unterschiede in den Subiterationen treten hauptsächlich zu Beginn des Verfahrens auf, solange, bis sich der Stoß beginnt abzuheben und die Störungen in alle Zellen propagiert sind. Diese Laufzeitunterschiede sind damit äußerst dynamisch und selbst von einer dynamischen Lastverteilung praktisch kaum zu egalisieren, da sie sich quasi von Iteration zu Iteration ändern können. Eine dynamische Lastverteilung würde daher hier vor allem durch den nochmals zusätzlichen Aufwand für die Umverteilung mehr schaden als nutzen.

Nun wird, wie bereits erwähnt, nochmals ein Lösungsschritt durchgeführt, bevor auf den jeweiligen Prozessoren die nächste Iteration beginnt.

Abbildung 4.15 zeigt die Matrix der Datenvolumina, die während der gezeigten Iteration zwischen den Prozessen verschickt wurden. Dabei ist schön die Größe der Schnittstellen zu sehen. Da im verwendeten Netz alle Blöcke die Größe 80x40x40 haben besitzen die Schnittflächen alle die Größe 80x40 oder 40x40. Wegen der physikalischen Ränder kann es auch in jeder Richtung noch höchstens eine Zelle mehr sein. Und natürlich bedingt eine doppelte so große Schnittstelle zwischen zwei Blöcken in der Summe ein doppelt so hohes Datentransfervolumen,

wie es auch in der Matrix zu sehen ist. Zum Beispiel besitzen alle Schnittstellen, die zwischen zwei neuen Blöcken liegen, die aus einem "alten" Block hervorgegangen sind, eine große Schnittfläche untereinander. Dies ist in der Matrix zwischen den Prozessen 7 und 0, 8 und 1, 10 und 3 sowie 11 und 4 zu sehen. Aus Block 3 sind die neuen Blöcke 3, 8, 9 und 10 entstanden, die auf den Prozessen 5, 6, 9 und 2 liegen. Jeder dieser Blöcke hat nun zwei Nachbarn, die auch aus Block 3 hervorgegangen sind. Und mit jedem dieser Nachbarn besitzt er eine große Schnittfläche. Dies sieht man zum Beispiel an Block 10 auf Prozess 2, der eine solche Schnittstelle mit Block 8 und 9 auf Prozess 6 und 9 hat.

Wenn man die Blöcke untersucht, die nicht aus dem Mittelblock drei entstanden sind, fällt auf, dass manche mit vier andere aber mit fünf Nachbarn kommunizieren, obwohl alle Blöcke an zwei Randflächen physikalische Ränder besitzen. Der Grund hierfür ist die Aufteilung des Blocks drei. Er wurde, wie alle "alten" Blöcke, in der zweiten, aber eben auch in der dritten Dimension geteilt. Damit haben nun insgesamt vier der acht neuen Blöcke, die nicht aus dem ehemaligen Mittelblock hervorgegangen sind, über eine ihrer Schnittstellen Kontakt zu zwei neuen Blöcken, über die Daten ausgetauscht werden müssen. Dies ist zum Beispiel bei Block 1 auf Prozess 7 der Fall. Er kommuniziert mit Block 3 und 9 auf den Prozessen 5 und 9, während sein Nachbar, Block 2 auf Prozess 8, nur mit Block 9 auf Prozess 9 Daten austauscht. Dafür hat dieser Austausch das doppelte Datenvolumen, da die Schnittfläche ja doppelt so groß ist.

Zum Vergleich mit der gerade betrachteten Iteration wollen wir uns nun noch eine Iteration aus einem Lauf mit sechs Prozessen, dargestellt in Abbildung 4.16, ansehen. Im Vergleich zum Lauf auf zwölf Prozessoren fällt sofort auf, dass hier das Verhalten zwischen den Prozessoren weitaus ausgeglichener ist. Grund ist, dass hier nicht nur alle Blöcke fast den gleichen Aufwand für die Randbehandlung haben, sondern sie liegen alle ebenso am Festkörperrand des Netzes. Damit partizipieren aber alle gleich zu Beginn am sich ausbildenden Stoß und haben somit auch im Löser in etwa denselben Berechnungsaufwand. Außerdem laufen die sechs Prozesse alle auf einem Knoten, was Vorteile bei der Kommunikation mit sich bringt, da diese lokal über den gemeinsamen Speicher abgewickelt werden kann. Aus diesen Gründen beträgt der Kommunikationsanteil (Anteil der in MPI-Aufrufen verbrachte Zeit) gegenüber der Gesamtlaufzeit des Programms bei diesem Lauf etwa 2,5 %, während er im vorhin betrachteten bei knapp 19 % lag.

4.3.5 Multilevelverfahren

Bevor wir zum Abschluß noch einen Blick auf die Ergebnisse einiger Simulationsrechnungen werfen, sollen noch die Möglichkeiten des Multilevelverfahrens dargestellt werden. Leider konnte das Multilevelverfahren aus Zeitgründen nicht mehr in den Programmcode für das Multiblockverfahren integriert werden, obwohl vor allem von der Datenstruktur her die Möglichkeiten dafür bereits gegeben

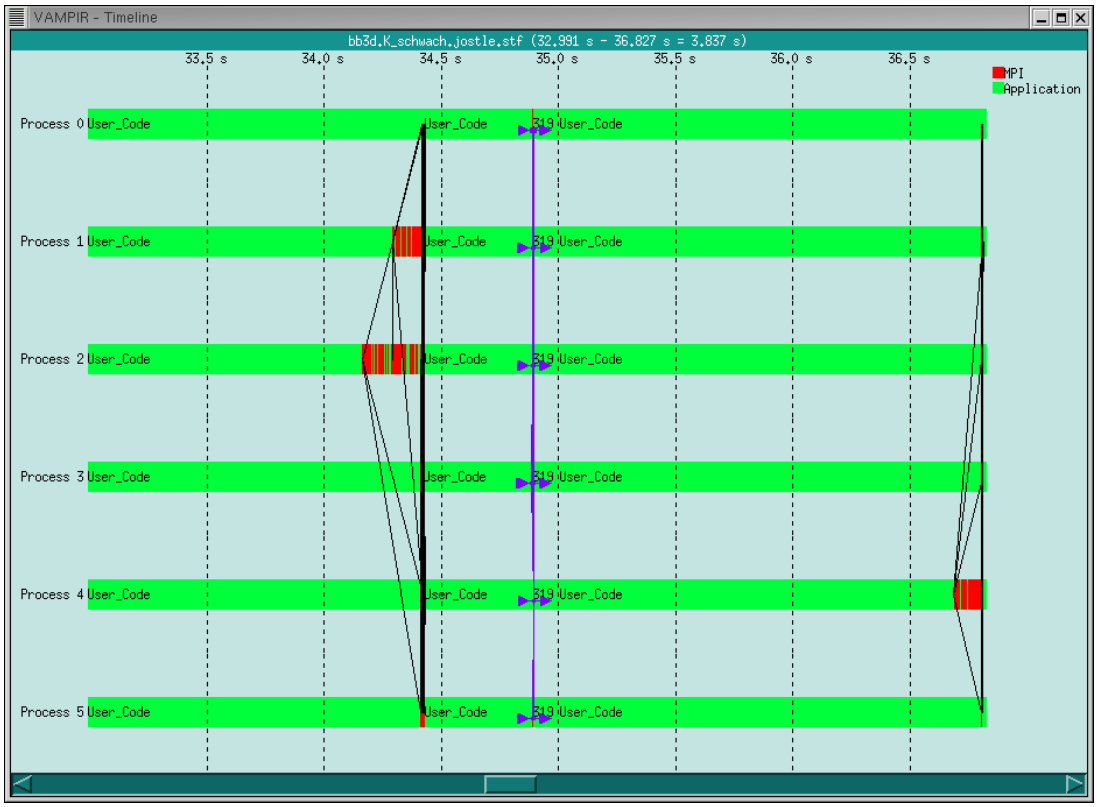

Abbildung 4.16: Ablauf einer Iteration bei einem Lauf auf 6 Prozessoren

sind. Trotzdem möchte ich ein Ergebnis hier darstellen, um das Potential dieser Methode zu unterstreichen. Um dieses Potential für das URANUS-Verfahren zu untersuchen, wurde die parallelisierte Einblockversion zur Verwendung des Multilevelverfahrens erweitert. Mit diesem Programm wurde die dargestellte Berechnung durchgeführt. Als Referenz ist das Ergebnis einer Rechnung mit der herkömmlichen Methode angegeben. Als Testfall wurde die Strömung um Hermes, einen ehemals geplanten europäischen Orbiter, mit einem Netz aus 221.184 Zellen gerechnet. Es handelt sich hierbei also nicht um ein einfaches Testnetz, sondern um einen realen Anwendungsfall, der auch die eingangs beschriebenen Probleme bei der Netztopologie besitzt. Für das Multilevelverfahren wird dieses Netz nach Programmstart beliebig oft, hier zweimal, automatisch vergröbert, indem in jeder Dimension zwei Zellen zusammengelegt werden. Es wurden also immer acht Zellen zu einer Zelle des neuen Netzes verschmolzen. Mit dem gröbsten Netz startet dann die Simulation. In Abbildung 4.17 sind von der Rechnung der Konvergenzverlauf und die CFL-Zahl angegeben. Im dargestellten Graph fallen sofort die Sprünge bei der Umschaltung auf das jeweils nächst feinere Netze auf, die im Prototyp nach einer vorgegebenen Anzahl von Iterationen durchgeführt wird. Zur Prolongation wurde das einfachst mögliche Verfahren verwendet, indem einfach die errechneten Werte einer Zelle auf alle Zellen, die aus dieser Zelle

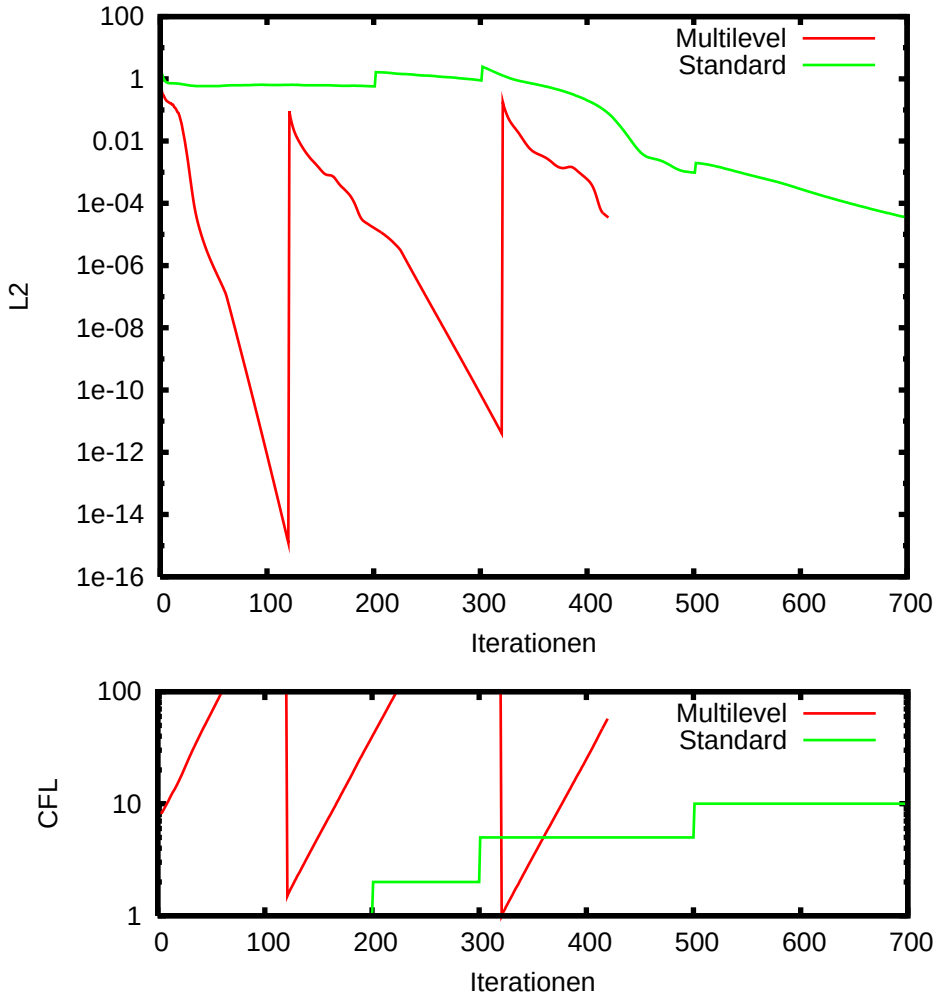

Abbildung 4.17: Konvergenz einer Simulationsrechnung mit dem Prototyp des Multilevelverfahrens

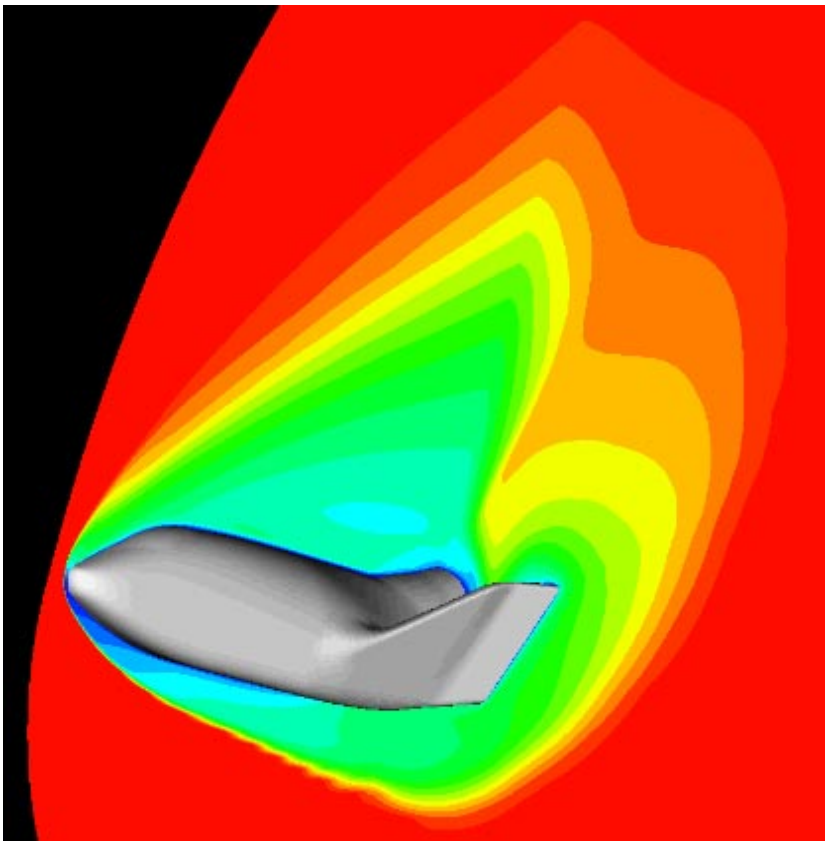

Abbildung 4.18: Ergebnis einer Simulationsrechnung auf der Cray T3E mit dem Idealgasverfahren bei Mach 6 und 40 Grad Anstellung

im feineren Netz hervorgehen, übertragen werden. Trotzdem erreicht man mit dem Multilevelverfahren bereits nach 420 Iterationen einen Residuumswert, der mit dem normalen Verfahren erst nach 700 Iterationen erreicht wird. Hierbei ist allerdings zu beachten, das von den 420 Iterationen nur 100 mit dem feinen Netz durchgeführt wurden, dass im normalen Verfahren für alle 700 Iterationen verwendet wurde. 200 Iterationen wurden auf dem gröberen Netz mit nur einem Achtel der Zellen und 120 Iterationen gar nur auf einem Netz mit einem Vierundsechzigstel der Zellen durchgeführt. Der Gesamtaufwand entspricht also nur etwa dem Aufwand von 127 Iterationen auf dem feinen Netz und damit weit weniger als einem Fünftel gegenüber dem normalen Verfahren. Berücksichtigt man nun noch, dass die Erstellung der groben Netze auch Rechenzeit benötigt und das vor allem beim groben Netz der relative Kommunikationsaufwand doch erheblich ist, kann man noch von einer Rechenzeitersparnis von etwa 75% ausgehen.

4.3.6 X38

An den Ergebnissen für die X38 bestand seitens der Kollegen aus der Strömungsmechanik großes Interesse. Hier wird daher neben einem Ergebnis mit dem Idealgasverfahren auch bereits das Ergebnis einer Produktionsrechnung mit dem Nichtgleichgewichtsverfahren gezeigt.

Simulationsergebnis mit dem Idealgasverfahren

Mit dem Idealgasverfahren wurde ein kompletter Simulationslauf mit dem 888.832 Zellen großen X38 Gesamtkörpernetz auf der T3E durchgeführt, wegen Problemen an den Rändern allerdings nur erster Ordnung. Abbildung 4.18 zeigt das konvergierte Ergebnis für eine Fluggeschwindigkeit von Mach sechs bei einem Anstellwinkel von 40 Grad. Das Ergebnis wurde mit 600 Iterationen erreicht, die auf 128 Prozessoren der Cray T3E zu einer Laufzeit von 101 Minuten führten. Die angesprochenen Probleme am Rand sind, wie sich dann herausstellte, durch eine ungenaue Formulierung begründet, die sich vor allem im hinteren Bereich des Körpers auswirkt. Dies wurde von den Kollegen im Idealgasverfahren nicht mehr geändert, da dieses nur noch zum Testen der parallelen Version eingesetzt wird. Daher können hier keine Ergebnisse aus Rechnungen mit zweiter Ordnung gezeigt werden.

Berechnungsergebnisse mit dem vollen Nichtgleichgewichtsverfahren

Das neue parallele Multiblockverfahren wird inzwischen auch für Produktionsrechnungen eingesetzt. Hauptaugenmerk der Simulation ist dabei die genaue Berechnung der auftretenden Wärmeströme auf der Oberfläche des Wiedereintrittsfahrzeugs. Dazu wurde das neue parallele Multiblock URANUS Verfahren gemeinsam mit den Kollegen vom Institut für Raumfahrtsysteme der Universität Stuttgart (IRS) um die Behandlung der Nichtgleichgewichtseffekte und katalytischen Oberflächen ergänzt, die dort zwischenzeitlich entwickelt wurden. Abbildung 4.19 zeigt ein mit dieser Version mit 6000 Iterationen in 120 Stunden auf der NEC SX-5 berechnetes Ergebnis. Dargestellt ist die Machzahlverteilung für einen Wiedereintritt mit Mach 19,8 bei 40 Grad Anstellwinkel.

4.4 Fazit

Die Ergebnisse dieses Kapitels zeigen, dass die entwickelte parallele Mutliblockversion des URANUS Verfahrens den gestellten Ansprüchen gerecht wird. Neben der erreichten und nachgewiesenen Portabilität hat das parallele Verfahren auch gezeigt, dass es, mit aureichend großen Fällen eingesetzt, keine größeren Leistungsverluste gibt, die durch die Verwendung mehrerer auch vieler Prozesse

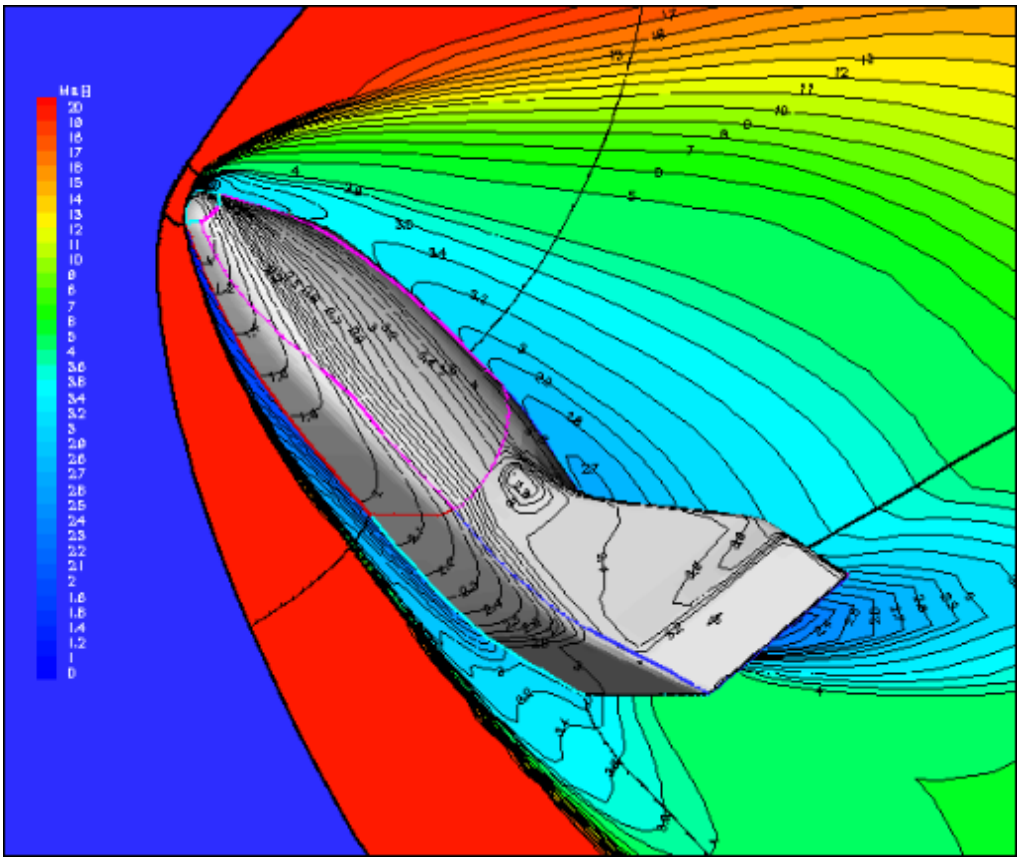

Abbildung 4.19: Ergebnis einer Simulationsrechnung mit dem Nichtgleichgewichtsverfahren auf der NEC SX-5, Eintrittsgeschwindigkeit Mach 19,8 bei 40 Grad Anströmung

und der notwendigen Kommunikation entstehen. Somit kann davon ausgegangen werden, dass es mit den entsprechenden Fällen auch auf mehreren tausend Prozessoren effizient einsetzbar ist.

Dabei werden die Blöcke aus dem eingelesenen Multiblocknetz automatisch zerlegt und vom Lastverteiler auf die verfügbaren Prozessoren verteilt. Dieses Verfahren arbeitet mit zufriedenstellenden Ergebnissen, wenn auch öfters eine bessere Lastverteilung durch eine Integration dieser beiden Schritte möglich wäre.

Die Untersuchung zeigt auch, dass Vektorrechner im Moment die kosteneffizienteste Plattform für das Programm darstellen.

Der Einsatz des hier entwickelten Verfahrens durch die Ingenieure des IRS hat bereits zu Simulationsergebnissen für eine komplette X-38 Konfiguration im chemisch-thermischen Nichtgleichgewicht geführt, die sonst nicht möglich gewesen wären [47].

Kapitel 5

Erweiterung des Konzepts zur Verwendung des Programms in einer Metacomputing Umgebung

Zur Durchführung großer Simulationen zur Erlangung relevanter Ergebnisse, werden derzeit noch immer die größten verfügbaren Rechnersysteme benötigt. Entweder, um die genauen Modelle oder eine entsprechend hohe Auflösung in der Topologie zu ermöglichen, oder auch beides. Leider stehen für Strömungssimulationen nicht immer die allergrößten Rechner zur Verfügung. Eine Idee, trotzdem eine entsprechende Rechnerleistung verfügbar zu machen, liegt im sogenannten Metacomputing. Hier wird mittels einer Zwischenschicht (Middleware) einem Anwendungsprogramm ein einheitlicher virtueller Größtrechner zur Verfügung gestellt. Tatsächlich besteht der Größtrechner aus zwei oder mehreren Großrechenanlagen, die aber jede für sich zu klein wären, entsprechende Ressourcen, sei es Rechenleistung oder Speicher, zur Verfügung zu stellen. Natürlich können von dem virtuellen Größtrechner nicht die gleichen Systemparameter verlangt werden, wie von den unterliegenden Großrechnern. Denn durch die notwendige Benutzung systemexterner Verbindungsnetzwerke, ist vor allem die Kommunikationsleistung eines solchen virtuellen Größtrechners eingeschränkt. Und zwar sowohl bezüglich Bandbreite, als auch bezüglich der Latenz. Soll solch ein virtuelles Rechensystem also benutzt werden, ist es von Vorteil, wenn die Anwendungsprogramme auf diese Gegebenheiten eingestellt sind. In diesem Kapitel werden wir uns daher ausführlich mit dem Thema Metacomputing für eine Strömungsanwendung auseinandersetzen. Nach einer Einführung in das Metacomputing werden die verwendeten Testumgebungen beschrieben. Hier standen sowohl eine rechenzentrumslokale als auch eine globale Testumgebung zur Verfügung. In der globalen Testumgebung wurden dabei kontinentale Entfernungen überbrückt. Da die ersten Ergebnisse vielversprechend waren, aber eindeutig auch die Schwächen einer solchen stark gekoppelten Anwendung in einer Metacomputingumgebung aufzeigten, wurde der Strömungslöser auch an diese Gegebenheiten angepaßt. Die erzielten Ergebnisse

zeigen, dass selbst ein solch eng gekoppelter Strömungslöser wie der hier untersuchte, von einer Metacomputingumgebung profitieren kann.

5.1 Allgemeine Einführung in das Metacomputing

In der wissenschaftlichen Welt hat sich bis heute noch keine genaue Definition für den Begriff Metacomputing herausgebildet. An sich bedeutet der Begriff: über das Computing, also die normalen Berechnungen, die auf einem Rechner durchgeführt werden, hinausgehend. Michael Resch hat in seiner Dissertation [65] Metacomputing speziell aus Sicht der Ingenieurswissenschaften definiert und gegen die Begriffe Verteiltes Rechnen, Cluster Computing und Grid Computing abgegrenzt. An dieser Stelle werden die wichtigsten Eigenschaften des Metacomputing nochmals aufgezeigt, die für die in dieser Arbeit durchgeführten Untersuchungen von Belang sind.

Resch definiert: *Metacomputing ist die Kopplung mehrerer sehr leistungsfähiger und verteilter Rechner und Instrumente, die über schnelle Netze so gekoppelt werden, dass sie dem Benutzer wie eine einzige Ressource erscheinen.*

Besonders wichtig und hervorzuheben ist, daß es sich bei Metacomputing um die Kopplung sehr leistungsfähiger Rechnersysteme, also typischerweise um Höchstleistungsrechner handelt. Da mehrere solche Anlagen meist nicht an einem Ort zur Verfügung stehen, muß diese Kopplung natürlicherweise über größere Entfernungen stattfinden. Und um diese Kopplung von technischer Seite möglichst leistungsfähig auszuführen, müssen schnellst-mögliche Netze dafür verwendet werden. Bei einem Metacomputer handelt es sich also um ein Netzwerk aus Höchstleistungsrechnern, die dem Benutzer wie ein einziger Rechner erscheinen. Um dieses Rechnernetzwerk als eine Resource ansprechen und verwenden zu können, ist zusätzliche Software notwendig. Diese soll im nächsten Kapitel vorgestellt werden.

5.2 Beschreibung der Kopplungssoftware PACX-MPI

Die Kopplungssoftware PACX-MPI [28] stellt der Anwendung den beschriebenen Metacomputer quasi als ein Rechnersystem zur Verfügung. Sie ist dabei auf Anwendungen beschränkt, die für die Verwendung auf Parallelrechnern mit verteiltem Hauptspeicher (Distributed Memory Systeme) mittels der Message Passing Bibliothek MPI parallelisiert wurden. Dies ist aber keine Einschränkung, da für eine Verwendung auf einem Metacomputer sowieso nur Applikationen interessant sind, die bisher schon auf den größten und schnellsten verfügbaren Systemen ge-

laufen sind und diese liegen heute alle mit MPI parallelisiert vor. PACX-MPI ist für die Applikation völlig transparent. Das Anwenderprogramm sieht also eine normale MPI-Schnittstelle und braucht im Prinzip von der unterliegenden Topologie nichts zu wissen. Natürlich ist es für eine Optimierung der Applikation auf die Plattform sinnvoll, die Topologie zu berücksichtigen. Dies ist aber nicht unbedingt nötig. Das heißt, dass jede beliebige mit MPI parallelisierte Anwendung mittels PACX-MPI auf einem entsprechend konfigurierten Metacomputer lauffähig ist.

Im folgenden wird nun eine kurze Übersicht über die Konzeption und Funktionalität von PACX-MPI gegeben.

5.2.1 Realisierung von PACX-MPI

Da PACX-MPI mit allen MPI-Programmen verwendbar sein soll, muß es die komplette Funktionalität des MPI-1.2 Standards abbilden. Zumindestens aber den Teil, den die meisten Programme benutzen. Tatsächlich ist PACX-MPI an den Anforderungen gewachsen, die die jeweiligen Anwendungen an die MPI-Bibliothek für den Metacomputer hatten, bis dann der komplette Standard umgesetzt war.

Bei der tatsächlichen Realisierung der Kommunikation muß PACX-MPI zwischen der internen Kommunikation, also der Kommunikation zwischen zwei Prozessoren auf demselben Rechner, und der externen Kommunikation unterscheiden. Die externe Kommunikation zwischen den verschiedenen Rechnern wird von zwei ausgezeichneten Prozessen auf zwei dedizierten Prozessoren eines jeden Rechners durchgeführt, die damit für die Anwendung jeweils nicht zur Verfügung stehen (vgl. Abbildung 5.1). Dabei ist jeweils ein Prozeß für die eingehende und der andere für die abgehende Kommunikation zuständig. Nur diese beiden Prozesse kennen die jeweils anderen Rechensysteme. Auch auf jedem Applikationsprozeß läuft eine kleine Zwischenschicht von PACX-MPI. Diese entscheidet für jede Punkt-zu-Punkt-Kommunikation, wie diese gehandhabt werden soll: ist der Empfänger ein lokaler Prozessor, wird die Nachricht über das interne Verbindungsnetzwerk des Rechners unter Verwendung der normalerweise optimierten MPI-Version des Hardwareherstellers an den Zielprozessor weitergeleitet. Ist der Empfängerprozessor dagegen auf einem anderen Rechensystem beheimatet, leitet die lokale Zwischenschicht die Nachricht an den lokalen Prozessor für die abgehende Kommunikation weiter. Dieser bestimmt dann den Rechner des Empfängerprozessors und schickt die Nachricht mittels TCP/IP über die externe Netzwerkverbindung an den dortigen Kommunikationsprozessor für eingehende Nachrichten. Jener nimmt die Nachricht entgegen und leitet sie dann wieder über das interne schnelle Verbindungsnetzwerk an den tatsächlichen Empfängerprozessor weiter. Die gesamte notwendige Verwaltungsarbeit, wie das Umnumerieren der Prozessoren von der globalen Anwendungssicht auf die lokale Systemsicht, wird dabei von PACX-MPI übernommen. Kollektive Kommunikation und alle anderen MPI-

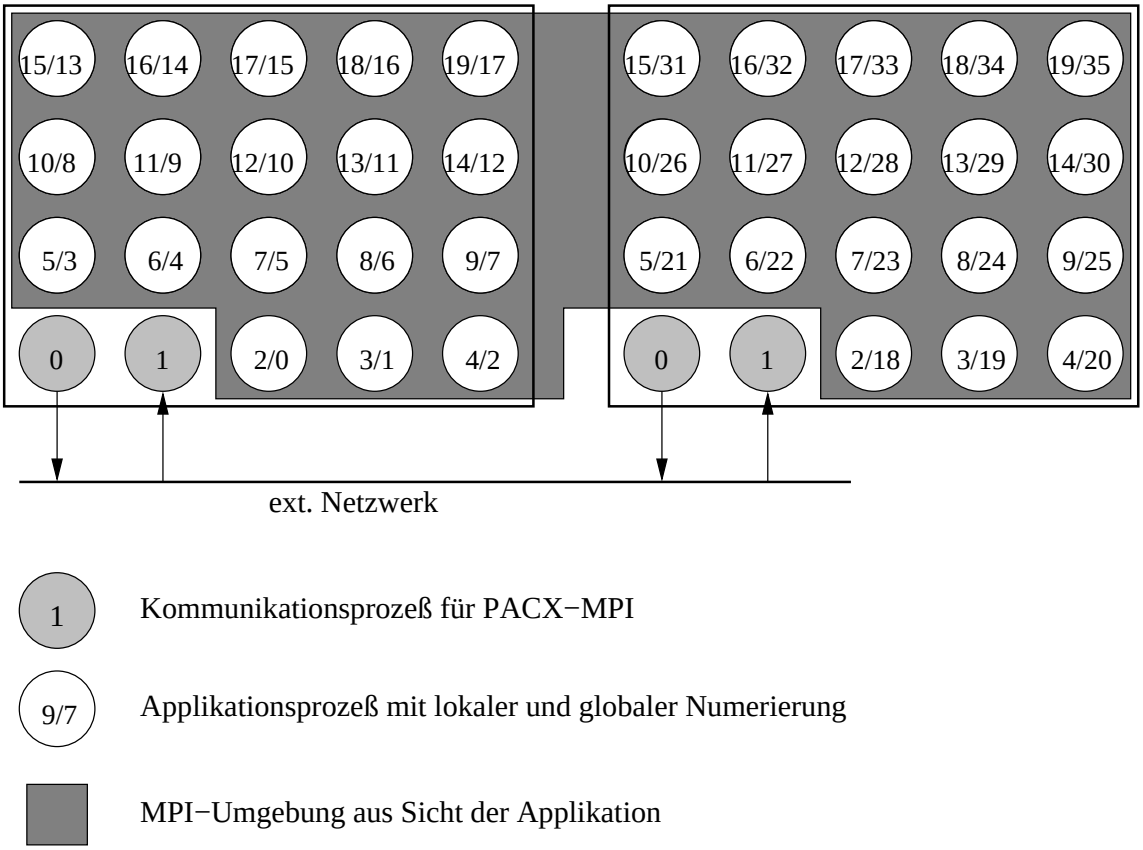

Abbildung 5.1: PACX-MPI logische Architektur

Funktionalitäten, die für den Einsatz in der Metacomputingumgebung verändert werden müssen, werden von der neuen Kommunikationsschicht abgefangen, entsprechend umgesetzt und dabei speziell auf das heterogene Kommunikationsumfeld hin optimiert. Abbildung 5.2 zeigt das aus der eben beschriebenen Vorgehensweise resultierende Schichtenmodell von PACX-MPI sowohl für die Anwendungs- als auch für die Kommunikationsprozesse.

5.3 Beschreibung der Testumgebungen

Um die Eigenschaften des Programms bei der Verwendung in einer Metacomputingumgebung zu bestimmen, wurden mehrere Testszenarien verwendet. Da zum Zeitpunkt der Messung die Kommunikationsbibliothek in der Zwischenschicht, PACX-MPI, noch nicht in der Lage war, heterogene Systeme zu koppeln, wurden die Messungen auf einem homogenen Metacomputer durchgeführt. Das heißt, die Grossrechensysteme, aus denen die Metacomputingumgebung aufgebaut wurde, waren vom selben Typ. Dies bietet außerdem den Vorteil, daß die Leistung aller verwendeter Prozessoren gleich ist. Damit kann der derzeitige Lastverteilungs-

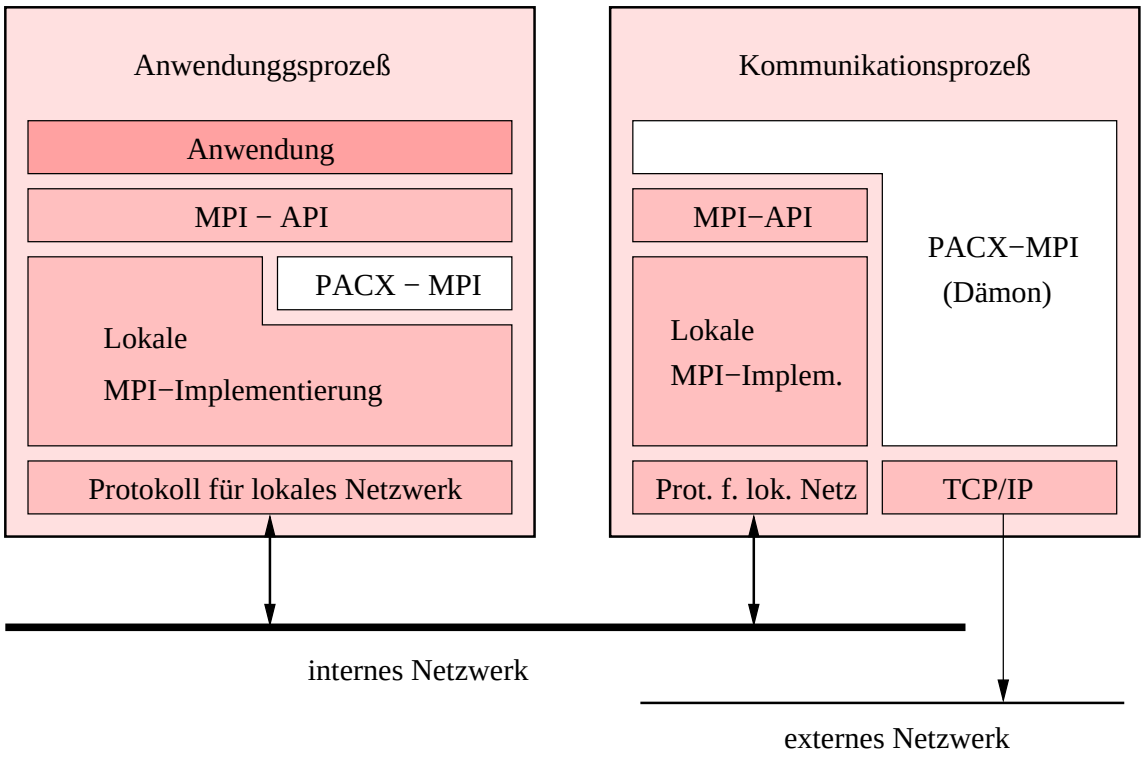

Abbildung 5.2: PACX-MPI, Schichtenkonzept

algorithmus beibehalten werden. Die Messungen und Versuche für diese Arbeit wurden auf Massiv Parallelen Rechnern des Typs T3E der Firma Cray durchgeführt, die im vorigen Kapitel bereits vorgestellt wurden.

Als lokale Testumgebung wurde die T3E in Stuttgart verwendet, allerdings in einer anderen Konfiguration: um einen virtuellen Metacomputer zu simulieren, wurde das Programm in zwei verschiedenen Partitionen der Maschine gestartet, die untereinander nur über das Netzwerkinterface der Maschine und das Netzwerkprotokoll TCP/IP Verbindung aufnehmen konnten. Damit läßt sich eine Metacomputingumgebung gut simulieren, da so die Netzwerkverbindung zwischen den Partitionen bereits um Klassen schlechter ist, als die Verbindung der Knoten innerhalb einer Partition über das schnelle interne Verbindungsnetzwerk. Im Rahmen eines Projekts der G7-Staaten konnten erstmals auch Rechner auf zwei Kontinenten gekoppelt werden. Dabei wurden die T3E's in Pittsburgh (USA) und Stuttgart über eine Hochgeschwindigkeitsverbindung gekoppelt, ebenfalls mit dem Netzwerkprotokoll TCP/IP. In Abbildung 5.3 ist die Konfiguration dieses transkontinentalen Metacomputers dargestellt. Hieraus wird vor allem der notwendige Aufwand für die Netzwerkverbindung zwischen Stuttgart und Pittsburgh ersichtlich, die zwischen Stuttgart und Chicago eine dedizierte Bandbreite von 2MBit/s aufwies. Innerhalb des vBNS zwischen Chicago und Pittsburgh konnte keine Bandbreite reserviert werden, jedoch war dieses Netz 1997 bereits so gut

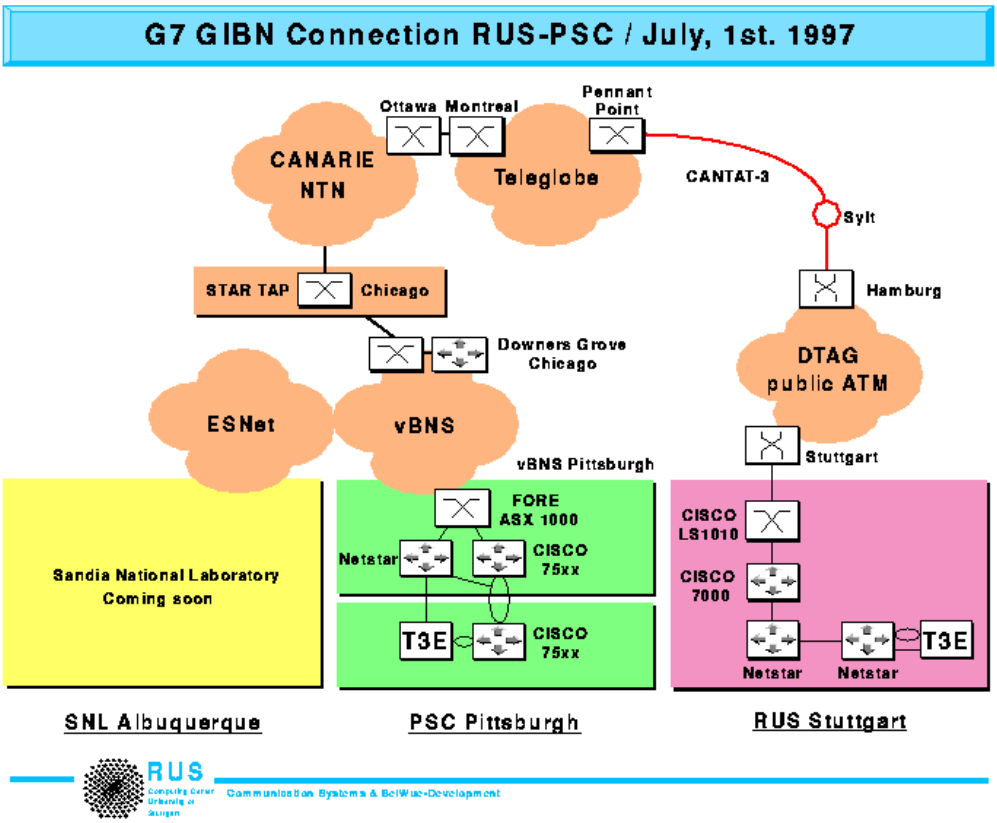

Abbildung 5.3: Konfiguration des interkontinentalen Metacomputers

ausgebaut, dass dadurch keine Verminderung der nutzbaren Bandbreite auf der gesamten Verbindung zu befürchten war. Dieser transkontinentale Metacomputer wurde dann für die folgenden Versuche und Messungen als Testsystem verwendet. Dies führte gegenüber dem lokalen Testsystem hauptsächlich zu einer weiteren Erhöhung der Latenz und einer nochmaligen Verminderung der Bandbreite. In Tabelle 5.1 sind die unterschiedlichen Leistungsdaten im Vergleich zum internen Verbindungsnetzwerk der T3E nochmals zusammengefaßt.

Für abschliessende Messungen konnte dann noch ein drittes T3E System in Manchester/Großbritannien mit verwendet werden, so daß Messungen für Rechenläufe auf einem Metacomputer bestehend aus drei Supercomputern mit insgesamt bis zu 1536 Prozessoren durchgeführt werden konnten.

5.4 Erste Testergebnisse

Die gesamten Messungen und Entwicklungen, die im folgenden dargestellt werden, wurden noch mit der Einblockversion von URANUS durchgeführt, da seiner-

Bibliothek	**Verbindung**	**Latenz** [μs]	**Bandbreite** [MB/s]
Cray MPI	internes Netzwerk	16	307
PACX-MPI	eine Partition	19	297
PACX-MPI	zwei Partitionen (TCP/IP)	4000	0.25
PACX-MPI	zwei Systeme (HLRS - PSC)	75000	0.15

Tabelle 5.1: Leistungsfähigkeit der Verbindungsnetzwerke

zeit die parallele Multiblockversion noch nicht lauffähig zur Verfügung gestanden hat. Für die ersten Messungen wurde das Simulationsprogramm in der Version verwendet, wie sie für die Massiv Parallelen Rechner programmiert ist und dort ebenso verwendet wird. Es wurden also keinerlei Anpassungen und Veränderungen vorgenommen. Die Tests wurden mit einem kleinen Netz mit 36.400 Netzpunkten (26 x 56 x 25) auf 16, 32 und 64 Prozessoren durchgeführt. Es wurden jeweils 10 Iterationen gerechnet, wobei im Löser 5 Subiterationen voreingestellt war. In Tabelle 5.2 sind die erzielten Laufzeiten zusammengefaßt.

Die Berechnungen wurden in Stuttgart auf der Cray T3E (hwwt3e) und im Fall der Kopplung auch auf der Cray T3E des Pittsburgh Supercomputer Centers (jaromir) durchgeführt. In der Tabelle sind außer der jeweils verwendenten Prozessorzahl und der Gesamtlaufzeit noch der Zeitbedarf für das Aufstellen des Gleichungssystems (GA), das Lösen des Gleichungssystems (GL) und für den gesamten Iterationsteil (Proces.) angegeben.

Bei der Analyse der Ergebnisse fällt zunächst auf, dass die Simulationszeiten durch die Verwendung eines Metacomputers sofort um eine Größenordnung schlechter werden. Selbst wenn der Metacomputer durch die Verwendung von zwei Partitionen auf einem Parallelrechner simuliert wird, trifft dies bereits zu. Der starke Anstieg der Latenz und die Verminderung der Bandbreite haben also erheblichen Einfluß auf das Programm. Es zeigt sich auch, dass der Latenzunterschied zwischen der Messung auf dem lokal simulierten Metacomputer und einem realen von etwa einem Faktor 10 nicht mehr so stark ins Gewicht fällt, obgleich die Simulationszeiten nochmals schlechter geworden sind. Außerdem fällt auf, dass im Metacomputingfall durch eine Erhöhung der Prozessorzahl keine Verminderung der Laufzeit mehr zu erreichen ist. Daraus läßt sich ableiten, dass der Fall im Metacomputing für 32 Prozessoren bereits zu klein ist. Wird dagegen lokal gerechnet, sind, wie in Tabelle 5.3 zu sehen ist, auch für 64 Prozessoren noch Verbesserungen feststellbar. Diese sind sogar erheblich, wenn man nur den Iterationsteil der Applikation betrachtet.

System	Anzahl Prozesse	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
hwwt3e	16	10,5	12,9	27,5	35,9
	32	5,2	5,3	12,2	20,4
	64	2,9	2,9	6,6	14,9
hwwt3e - hwwt3e	2x8	10,6	176,4	200,7	211,4
	2x16	5,3	159,9	192,6	205,1
jaromir - hwwt3e	2x8	10,5	254,1	289,2	310,7
	2x16	5,2	220,1	277,7	311,6
	2x32	2,9	149,4	265,4	323,7

Tabelle 5.2: Ergebnisse der ersten Messungen mit URANUS auf Metacomputern

System	Anzahl Prozesse	GA	GL	Proces.	Gesamt
hwwt3e	16	1,00	1,00	1,00	1,00
	32	2,02	2,43	2,25	1,76
	64	3,62	4,45	4,17	2,41
hwwt3e - hwwt3e	2x8	1,00	1,00	1,00	1,00
	2x16	2,00	1,10	1,04	1,03
jaromir - hwwt3e	2x8	1,00	1,00	1,00	1,00
	2x16	2,02	1,15	1,04	1,00
	2x32	3,62	1,70	1,09	0,96

Tabelle 5.3: Speedup von Standard URANUS auf einem Metacomputer

System	**Anzahl Prozesse**	**Proces. – (GA+GL)** [s]	**Gesamt – Proces.** [s]
hwwt3e	16	4,1	8,4
	32	1,7	8,2
	64	0,8	8,3
hwwt3e - hwwt3e	2x8	13,7	10,7
	2x16	27,4	12,5
jaromir - hwwt3e	2x8	24,6	21,5
	2x16	52,4	33,9
	2x32	113,1	58,3

Tabelle 5.4: Laufzeitdifferenzen gemessen mit URANUS auf Metacomputern

Wenn wir die Ergebnisse nun genauer untersuchen, werden die Ursachen für das schlechte Verhalten von URANUS im Metacomputing klarer. Betrachten wir dazu für die Messungen aus Tabelle 5.2 die Differenz zwischen der Gesamtlaufzeit und der Laufzeit für den gesamten Iterationsteil. Dies ist genau der Zeitbedarf für das das Pre- und Postprocessing. Auerdem wollen wir jeweils die Differenz zwischen der Laufzeit des gesamten Iterationsteils und dem Zeitbedarf für das Aufstellen (GA) und Lösen (GL) der Gleichungen untersuchen. Genau diese Zeit wird für die Berechnung der Residuen benötigt. Beide berechneten Zeiten sind in Tabelle 5.4 angegeben.

Es ist sofort zu erkennen, dass das Pre- und Postprocessing im lokalen Fall immer in etwa gleich lang dauert. Im Pre- und Postprocessing werden die Gitterdaten aus einer Datei von einem Prozeß eingelesen, dann wird das Gitter zerteilt und jedem Prozeß sein Teil zugeschickt. Dann berechnet jeder Prozeß noch einige Geometriedaten für seinen Gitterteil, bevor die Iterationen zur Lösung beginnen. Das Postprocessing besteht nur aus zwei Teilen, dem Sammeln der Ergebnisse auf einem Prozeß, der diese dann in die Ergebnisdatei schreibt. Das Pre- und Postprocessing ist im lokalen Fall also eindeutig vom I/O dominiert. Der Zeitaufwand für das Verteilen des Gitters und das Einsammeln der Ergebnisse steigt mit der Anzahl der Prozessoren. Er fällt aber wie sich zeigt gegenüber dem Lesen und Schreiben der Dateien nicht ins Gewicht.

Anders dagegen im Metacomputing. Hier steigt der Zeitbedarf für die erwähnte Kommunikation erheblich an. Das war zu erwarten, da ja jeweils die Hälfte der Daten über die langsame Verbindung zwischen den Rechnern transferiert werden

System	Anzahl Prozesse	Gesamt − Proces. abzgl. I/O [s]
hwwt3e - hwwt3e	2x8	2,4
	2x16	4,2
jaromir - hwwt3e	2x8	13,2
	2x16	25,6
	2x32	50,0

Tabelle 5.5: Kommunikationszeiten für Startup und Datenrücktransport im Metacomputing

muß. Die Zeit für den I/O kann in allen Fällen als gleich angenommen werden, da in allen aufgeführten Fällen nur ein Prozessor den I/O durchführt. Damit läßt sich die Zeit für den I/O aus den Werten der letzten Spalte für ein System allein in Tabelle 5.4 mit etwa 8,3 Sekunden bestimmen. Subtrahiert man diesen Wert nun von den anderen Werten derselben Spalte, erhält man den Zeitaufwand für die Kommunikation in den Metacomputingfällen. Diese sind in Tabelle 5.5 aufgeführt. Aus den Ergebnissen ist dann sofort ersichtlich, dass der Zeitaufwand für die Kommunikation linear mit der Anzahl der Prozessoren ansteigt. Das kommunizierte Datenvolumen bleibt dabei nahezu konstant, es wird ja jedesmal die Hälfte der Gitterdaten auf die andere Partition übertragen. Nur die Information für die jeweiligen Halos wird ein wenig mehr, da die Anzahl der Halos ansteigt. Allerdings vergrößert sich die Anzahl der Nachrichten, da für jeden Prozessor auf der anderen Partition sowohl beim Verteilen des Gitters als auch beim Einsammeln der Ergebnisdaten eine Nachricht geschickt werden muß.

Daraus folgt, dass die Datenübertragung zu Beginn und zum Ende eindeutig von der Latenz dominiert sind. Bei der Wertung und dem Einfluß dieses Anteils auf die Gesamtrechenzeit muß man natürlich berücksichtigen, daß die hier durchgeführten zehn Iterationen nicht einem realen Simulationslauf entsprechen. Bei diesem ist von Iterationszahlen von 200 bis über 2000 auszugehen, je nach interner Steuerung und Anwendervorgabe. Das Pre- und Postprocessing fällt also bei einem realen Anwendungsfall bei weitem nicht so stark ins Gewicht, wie es bei den hier durchgeführten Messungen auf den ersten Blick scheint.

Ein ähnliches Bild zeigt sich, wenn wir die Spalte Proces.-(GA+GL) in Tabelle 5.4 nochmals betrachten. Hier ist jeweils die Zeitdifferenz zwischen der Gesamtlaufzeit des Iterationsteils und dem Zeitbedarf für das Aufstellen und Lösen der Gleichungen aufgetragen. Diese Zeit enthält damit den Aufwand für die Be-

Anzahl Prozesse	**Verteilung**
16	4 x 4 x 1
32	4 x 8 x 1
64	4 x 16 x 1

Tabelle 5.6: Verteilung des Gitters auf die Prozesse

rechnung der drei Residuen. Dabei wird erst der jeweils lokale Residuenanteil berechnet, der dann mittels einer kollektiven Operation kommuniziert und weiterverarbeitet wird. Hierzu werden Aufrufe von MPI_Allreduce bzw. MPI_Reduce verwendet. Wie zu sehen ist, besaß die Metacomputingbibliothek zu jener Zeit noch keine optimierte Unterstützung für diese Aufrufe, denn der Zeitbedarf steigt ebenfalls wieder linear mit der Prozessorzahl. Hier ist die Latenzabhängigkeit noch viel offensichtlicher als im vorigen Fall, da hier wohl viele kurze Nachrichten versandt werden.

Gehen wir nun noch einmal zu Tabelle 5.2 zurück und betrachten den Zeitaufwand für das Aufstellen und Lösen der Gleichungen. Beim Aufstellen der Gleichungen zeigt sich kein Laufzeitunterschied, da in diesem Programmteil nicht kommuniziert wird. Die notwendigen Werte von Nachbarzellen liegen in den Halozellen am Rand des jeweiligen Rechengebiets als Kopien vor. Diese Kopien werden nach dem Lösen jeweils auf den neuesten Stand gebracht. Die Laufzeiten für den Löser in Spalte GL zeigen dagegen ein interessantes Bild, denn die Laufzeit für den Löser sinkt. Ursache dafür ist gegenüber dem Pre- und Postprocessing eine andere Kommunikationsstruktur. Während im Preprocessing ein Prozeß die Gitterdaten an alle anderen schickt (one to many) und beim Postprocessing die Ergebnisse wieder einsammelt (many to one), tauschen hier immer Prozessorpaare Daten untereinander aus. Dabei spielt natürlich die Verteilung des Rechengitters auf die Prozesse eine erhebliche Rolle. Diese hing in den getesten Fällen nur von der Prozessorzahl und nicht von den verwendeten Rechnern ab und ist in Tabelle 5.6 angegeben. Die Verteilung bezieht sich dabei auf die drei Dimensionen des Rechengitters. Die Verteilung 4x8x1 heißt damit zum Beispiel, dass die Zellen der ersten Dimension des Rechengitters dreimal unterteilt, die Zellen in der zweiten Dimension siebenmal unterteilt und in der dritten Dimension gar nicht unterteilt wurden. Damit berechnet ein Prozessor also bei dem verwendeten Netz mit 26x56x25 Zellen ein Netz der Größe 6(bzw. 7)x7x25.

Trotz allem ist der Zeitbedarf für das Lösen gegenüber den Läufen auf einem Massiv Parallelen Rechensystem natürlich erheblich und ist damit die Hauptursache für die inakzeptabel hohen Laufzeiten auf einem Metacomputer.

Fassen wir die Ergebnisse dieses Kapitels nochmals kurz zusammen. Das parallele Strömungssimulationsprogramm ist in der vorliegenden Form für Metacomputing nicht geeignet. Die Ursachen dafür sind:

- Das zentrale Konzept für das Pre- und Postprocessing führt zu einem hohen Kommunikationsaufwand, der mit der Anzahl der Prozessoren ansteigt.
- Der Aufwand für die kollektive Kommunikation zur Bestimmung der Residuen ist im Metacomputing hoch und steigt linear mit der Anzahl der Prozessoren.
- Der eingesetzte parallele Gleichungslöser ist durch seinen hohen Kommunikationsaufwand die Hauptursache für die Laufzeitprobleme des eingesetzten Programms im Metacomputing.

5.5 Anpassung des parallelen Strömungslösers

Nach dem schlechten Abschneiden von URANUS bei den ersten Tests in einer Metacomputingumgebung und der Ursachenanalyse wurde nach Lösungsmöglichkeiten gesucht, um Strömungssimulationsprogramme trotzdem in Metacomputingumgebungen einsetzen zu können. Diese sind in den folgenden Abschnitten dargestellt.

5.5.1 Einschränkung der Kommunikation im Löser

Wie oben beschrieben liegt im großen Kommunikationsbedarf des Gleichungslösers das Hauptproblem für URANUS im Metacomputing. Der Gleichungslöser basiert auf einer Idee von Hofhaus [45]. Die Grundidee ist relativ einfach und sei im folgenden anhand eines Tridiagonalsystems gezeigt. Die zu lösende Gleichung sei

$$A\vec{u} = \vec{r}$$

Numerische Gleichungslöser errechnen nun explizit oder iterativ daraus die Inverse A^{-1} der Matrix A. Leider sind diese Löser per se nicht so einfach parallelisierbar. Hofhaus schlägt nun eine Splittung

$$A = L + M$$

der Matrix A vor. Damit erhalten wir

$$(L + M)\vec{u} = \vec{r}$$

beziehungsweise

$$L\vec{u} + M\vec{u} = \vec{r} \tag{5.1}$$

Die Gestalt von L und M werden dabei durch die Parallelisierung bestimmt. L enthält alle Elemente der Hauptdiagonalen die lokal zum Prozessor sind und alle Elemente der Nebendiagonalen, die zwischen diesen Elementen koppeln. M enthält die restlichen Elemente, dies sind genau die Elemente der Nebendiagonalen, die die Kopplung zwischen den Hauptdiagonalelementen verschiedener Prozessoren herstellen. Wenn eine Matrix A mit 15 Elementen auf drei Prozessoren verteilt wird, ergibt sich zum Beispiel die folgende Aufteilung:

$$L = \left(\begin{array}{ccccc|ccccc|ccccc}
d_1 & r_1 & & & & & & & & & & & & & \\
l_2 & d_2 & r_2 & & & & & & & & & & & & \\
 & l_3 & d_3 & r_3 & & & & & & & & & & & \\
 & & l_4 & d_4 & r_4 & & & & & & & & & & \\
 & & & l_5 & d_5 & & & & & & & & & & \\
\hline
 & & & & & d_6 & r_6 & & & & & & & & \\
 & & & & & l_7 & d_7 & r_7 & & & & & & & \\
 & & & & & & l_8 & d_8 & r_8 & & & & & & \\
 & & & & & & & l_9 & d_9 & r_9 & & & & & \\
 & & & & & & & & l_{10} & d_{10} & & & & & \\
\hline
 & & & & & & & & & & d_{11} & r_{11} & & & \\
 & & & & & & & & & & l_{12} & d_{12} & r_{12} & & \\
 & & & & & & & & & & & l_{13} & d_{13} & r_{13} & \\
 & & & & & & & & & & & & l_{14} & d_{14} & r_{14} \\
 & & & & & & & & & & & & & l_{15} & d_{15}
\end{array}\right)$$

und

$$M = \left(\begin{array}{ccccc|ccccc|ccccc}
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & r_5 & & & & & & & & \\
\hline
 & & & & & l_6 & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & r_{10} & & & & \\
\hline
 & & & & & & & & & l_{11} & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & &
\end{array}\right)$$

Offensichtlich ist M dabei sehr schwach besetzt. Dies wird nun ausgenutzt um Gleichung 5.1 wie folgt umzuformen

$$L\vec{u} = \vec{r} - M\vec{u}$$

und daraus entsteht zur Lösung

$$\vec{u} = L^{-1}(\vec{r} - M\vec{u})$$

Zwar steht die Lösung jetzt auch auf der rechten Seite, dies kann aber durch ein iteratives Schema behoben werden

$$\vec{u}^{(n)} = L^{-1}(\vec{r} - M\vec{u}^{(n-1)})$$

wobei $\vec{u}^{(0)} = \vec{0}$ ist. Vorteil dieses Verfahrens ist, dass als lokaler Löser zur Invertierung der Matrix L der bereits implementierte Löser eingesetzt werden kann. Nachteilig ist, dass der Löser durch das iterative Schema ungenauer wird und durch die zusätzliche Iteration mehr Rechenzeit beansprucht.

Bei der Parallelisierung von URANUS wurde der Löser erst wie beschrieben implementiert, wobei allerdings die Matrix A heptadiagonal ist. Um nicht unnötig Rechenzeit zu verwenden, wurde das beschriebene parallele Lösungsverfahren in einem zweiten Schritt in den Jacobi Linienrelaxationslöser integriert. In diesem Löser werden dann nach Lösung jeder Linie der Ergebnisvektor ausgetauscht und die jeweilige rechte Seite damit neu berechnet. Durch diese Maßnahme verliert der Löser gegenüber der sequentiellen Version kaum Genauigkeit, hat nur äußerst geringen Mehrbedarf an Rechenzeit für das Ändern der rechten Seite, aber er benötigt erheblich mehr Kommunikation. Es müssen in jeder der fünf Subiterationen dreimal die Ergebnisse mit den Nachbarn ausgetauscht werden. Auf einem Massiv Parallelen Rechner mit einer ausgewogenen Netzwerkverbindung wie der Cray T3E und der Intel Paragon führt dies zu keinem Problem. Im Metacomputing aber fällt mit diesem Lösungsverfahren die gesamte Simulation.

Um die Kommunikation im Löser zu vermindern liegt es daher nahe, auf das ursprüngliche Verfahren zurückzugehen. Das hat den Vorteil, dass während des Lösens die Zwischenergebnisse nur wenige Male ausgetauscht werden, statt wie bisher fünfzehn mal. Das dazu notwendige Mehr an Rechenzeit sollte dabei durch die Einsparung bei der Kommunikation bei weitem aufgefangen werden können. Bevor aber genauerer Messungen im Metacomputing durchgeführt wurden, sollen hier erst die beiden Löservarianten noch kurz in der Praxis verglichen werden. Dazu wurden mit dem Testbeispiel Rechnungen mit beiden Lösern lokal durchgeführt. Abbildung 5.4 zeigt dabei die Unterschiede in der Konvergenz zwischen den beiden Lösern. Der ursprüngliche Löser ist dabei rot, der für das parallele URANUS veränderte blau dargestellt. Wie zu sehen ist, haben beide Löser einen nahezu identischen Konvergenzverlauf. Am rechten unteren Eck der Grafik kann man gerade noch erkennen, dass der ursprüngliche Löser eine etwas geringere Konvergenzgeschwindigkeit besitzt, die knapp 1% mehr Iterationen erfordert, um das Ergebnis mit der gleichen Genauigkeit zu erzielen. Das zeigt sich auch darin, dass er auf Störungen etwas verspätet reagiert, wie ebenfalls in der Abbildung zu sehen ist.

Untersuchen wir nun das Verhalten der Löser mit unterschiedlichen Netzwerken. Dazu bestand auf einer IBM RS/6000 SP die Gelegenheit, die sowohl mit

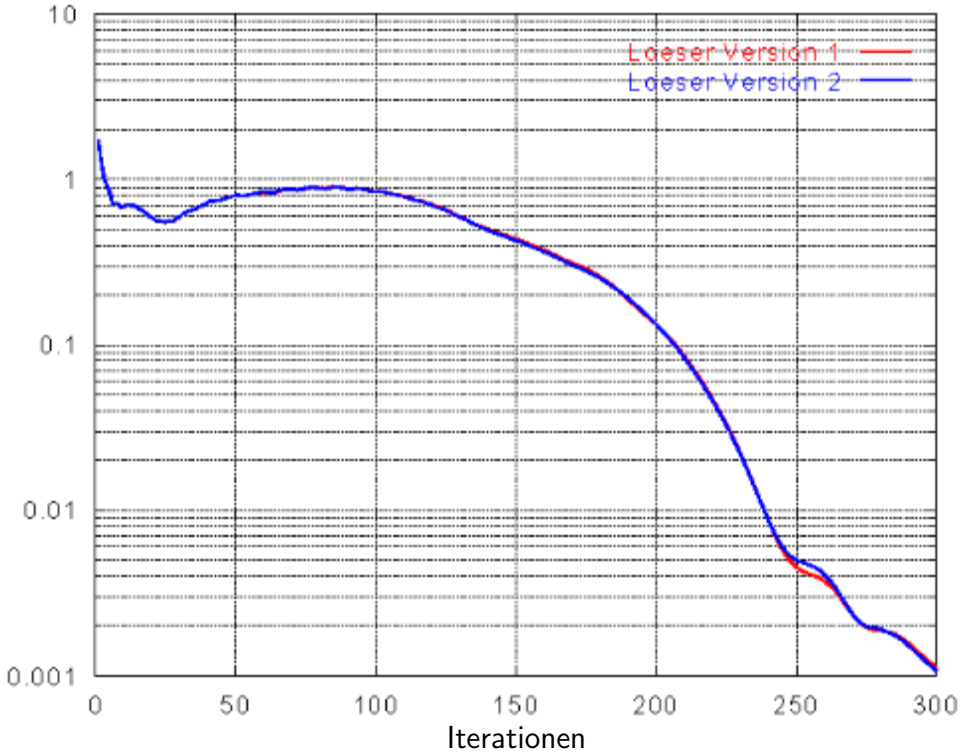

Abbildung 5.4: Konvergenzvergleich der beiden Löservarianten

dem High Performance Switch als auch mit Standard Ethernet vernetzt war. In Tabelle 5.7 sind Latenz und Bandbreite für die zwei unterschiedlichen Netzwerke aufgetragen. Somit ließ sich genau der Einfluß des Netzwerks bestimmen, da die Tests auf ein und derselben Maschine durchgeführt werden konnten. Für die Messungen wurden Strömungsberechnungen mit dem jeweiligen Gleichungslöser auf demselben Problem mit jeweils 200 Iterationen durchgeführt. In Abbildung 5.5 ist die dabei erzielte Laufzeit über der verwendeten Prozessorzahl aufgetragen. Es wurden insgesamt vier Messungen durchgeführt. Jeweils eine mit jedem Löser auf jedem der beiden Netzwerke.

Als erstes fällt dabei der Laufzeitunterschied der beiden Löservarianten auf. Er zeigt die starken 10% Mehraufwand, die Löser 1 mit der geringeren Kommunikation mit sich bringt. Bei der Verwendung von zwei Prozessoren ist noch kein großer Unterschied zu sehen. Hier kommen noch beide Netzwerkarten mit dem grösseren Kommunikationsbedarf des zweiten Lösers zurecht. Bei der Verwendung von vier Prozessoren beginnt sich das schwache Ethernetnetzwerk beim zweiten Löser schon auszuwirken, aber die Laufzeit ist noch besser als mit dem ersten Löser. Verwendet man hingegen 8 Prozessoren, ist mit dem stark kommunizierenden zweiten Löser nur noch ein geringer Laufzeitgewinn zu verzeichnen. Demgegenüber werden bei der Verwendung von 8 Prozessoren und dem ersten Löser zwar auch geringe Geschwindigkeitseinbußen sichtbar, doch diese sind bei

Netzwerk	**Latenz in μs**	**Bandbreite** in MB/s
Ethernet	440	1,0
HighPerformance-Switch	< 45	> 35

Tabelle 5.7: Bandbreite und Latenz der internen Netzwerkverbindungen auf der IBM RS/6000 SP

weitem nicht so stark wie mit dem zweiten Löser. Und wie ebenfalls zu erkennen ist, lohnt es sich bei diesem Problem und der Verwendung von Ethernet schon ab 8 Prozessoren auf den ersten Löser mit der schwachen Kopplung umzusteigen.

Verwenden wir nun die erste, ursprüngliche Löserversion für unsere Metacomputingfälle und führen die vorigen Messungen wieder in unserer bereits bekannten Testumgebung mit diesem Löser durch. Vergleichen wir nun die dabei erzielten Meßergebnisse in Tabelle 5.8 mit denen der ersten Messung in Tabelle 5.2. Bei der Messung auf einem System sehen wir, dass der Mehraufwand bei einer geringen Prozessorzahl mit fast 20% noch erheblich ist, dann aber bei steigender Anzahl der Prozessoren stark abnimmt, so dass er bei 64 Prozessoren fast unerheblich ist. Auf einem Metacomputer sind dagegen die Gewinne durch den ursprünglichen Löser erheblich. Auf dem lokalen Metacomputer beträgt der Zeitgewinn etwa einen Faktor von vier, während er auf dem realen transkontinentalen Metacomputer immerhin noch den Faktor drei beträgt. Im direkten Vergleich sind die Laufzeiten im realen Metacomputing verglichen mit denen auf einem lokalen System noch mehr als doppelt so hoch, selbst wenn man den I/O und die Initialisierungsphase nicht berücksichtigt und die Zeiten für die innere Iteration miteinander vergleicht. Daher liegt der Schluß nahe, dass der gewählte Simulationsfall für einen Einsatz des Metacomputing zu klein ist.

Bevor wir uns aber einem größeren Fall zuwenden, wollen wir die erzielten Ergebnisse weiter untersuchen. Bei der Analyse der Zeiten für die innere Iterationen stellt man fest, dass es selbst auf dem transkontinentalen Metacomputer einen tatsächlichen Speedup gibt, wenn er auch mit 1,44 gegenüber der Berechnung auf einer Partition recht bescheiden ist. Bei der Untersuchung der Zeiten für das Lösen des Gleichungssystems scheinen diese dagegen nicht mehr verläßlich zu sein, denn für 32 und 64 Prozessoren stimmen diese mit den auf einer Partition erzielten überein. Die Erklärung dafür liegt in der geänderten Kommunikationsstruktur im Löser und der Messmethode. Alle angegebenen Zeiten wurden auf dem Prozeß gemessen, der in der MPI-Umgebung den Rank 0 hat. Dieser übernimmt für das Dateneinlesen und Schreiben und die Partitionierung sozusagen eine Serverfunktion. Daher mißt er auch die Laufzeiten. Durch den Wegfall der

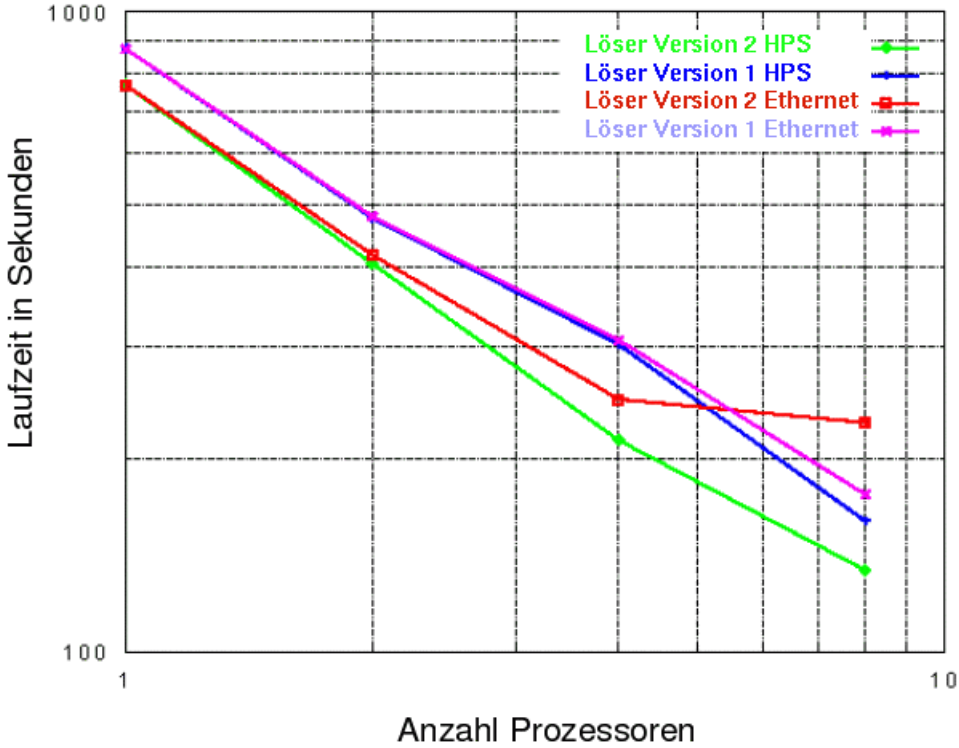

Abbildung 5.5: Vergleich der beiden Löservarianten mit unterschiedlichen Netzwerken

häufigen Kommunikation im Löser, fällt nun aber die starke Synchronisation zwischen den Teilgittern, die auf den einzelnen Prozessoren berechnet werden fast weg. Sie hat bei zwei Kommunikationsschritten im Löser nur noch Auswirkungen auf den Nachbarprozeß eines Prozesses, der während einer Kommunikation auf die Ergebnisse seines Nachbarn lange warten muß, da der Nachbar auf einer anderen Partition liegt. Bei der Verwendung von 16 Prozessoren ist der Prozeß 0 aber, wie jeder andere Prozeß auch mindestens direkter Nachbar eines solchen Prozesses. Bei der Verwendung von 32 und 64 Prozessoren trifft dies aber nicht mehr zu und damit sind die Löserlaufzeiten auf Prozeß 0 mit denen auf einer Partition erzielten identisch. Die Zeiten für den gesamten Iterationsdurchlauf sind aber tragfähig, da hier durch die kollektive Kommunikation quasi synchronisiert wird.

Wenn wir nun noch einen Blick auf den I/O werfen, der in Tabelle 5.9 sichtbar wird, stellen wir fest, dass er wieder ca. 8 Sekunden benötigt. Die Bildung der Differenzen wie im ersten Testfall zeigt, dass sich hier gegenüber dem ersten Test nahezu keine Veränderungen ergeben haben. Dies bestätigt nochmal die obigen Ausführungen zum Versenden der Gitter im Preprocessing sowie zum Rücktransport der Ergebnisdaten.

System	Anzahl Prozesse	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
hwwt3e	16	10,5	17,8	33,8	41,8
	32	5,2	6,1	14,1	22,5
	64	2,9	3,2	6,8	15,1
hwwt3e - hwwt3e	2x8	10,6	25,8	48,1	58,4
	2x16	5,3	6,1	35,4	47,9
jaromir - hwwt3e	2x8	10,5	31,8	73,5	95,8
	2x16	5,2	6,0	58,4	91,7
	2x32	2,9	3,1	50,9	99,9

Tabelle 5.8: Ergebnisse von URANUS auf einem Metacomputer mit einem schwächer koppelnden Löser

System	Anzahl Prozesse	Gesamt-Proces. [s]	Gesamt-Proces. exkl. I/O [s]
hwwt3e	16	8,0	-
	32	8,4	-
	64	8,3	-
hwwt3e - hwwt3e	2x8	10,3	2,0
	2x16	12,5	4,2
jaromir - hwwt3e	2x8	22,3	14,0
	2x16	32,3	26,0
	2x32	49,0	40,7

Tabelle 5.9: Laufzeitdifferenzen mit neuem Löser

System	Anzahl Prozesse	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
hwwt3e - hwwt3e	2x8	10,6	19,9	43,7	54,5
	2x16	5,3	6,0	31,6	44,5
jaromir - hwwt3e	2x8	10,5	27,0	63,9	83,7
	2x16	5,2	6,0	52,5	81,2
	2x32	2,9	3,1	46,9	88,7

Tabelle 5.10: Ergebnisse von URANUS mit einem schwächer koppelnden Löser und verbesserter PACX-MPI Bibliothek

System	Anzahl Prozesse	Gesamt-Proces. [s]	Gesamt-Proces. exkl. I/O [s]
hwwt3e - hwwt3e	2x8	10,8	2,5
	2x16	12,9	4,6
jaromir - hwwt3e	2x8	19,8	11,5
	2x16	28,7	20,4
	2x32	41,8	33,5

Tabelle 5.11: Laufzeitdifferenzen mit neuem Löser und neuer PACX-MPI Bibliothek

Zu diesem Zeitpunkt wurde von den Entwicklern der PACX-MPI Bibliothek eine neue Version der Kopplungsbibliothek zur Verfügung gestellt, die durch eine Optimierung der Einstellungen an den Sockets eine Verbesserung der externen Kommunikation verspricht. Die mit dieser neuen PACX-MPI Version gemessenen Zeiten sind in Tabelle 5.10 dargestellt. Wie sofort zu erkennen ist, führt die neue Version tatsächlich zu einigen Verbesserungen. Die Zeit für das Gleichungslösen nimmt ab, was man aber aufgrund der beschriebenen Problematik nur an den Messungen mit 2x8 Prozessoren direkt sehen kann. Besser erkennt man das daran, dass auch die Zeiten für die gesamten Iterationen abnehmen. Die Zeit für den Datentransport zu Beginn und nach Ende der Berechnung wurde in Tabelle 5.11 nochmals getrennt dargestellt. Hier erkennt man dann genau, dass die Änderungen erhebliche Verbesserungen vor allem im realen Metacomputing über grosse

System	**Anzahl Prozesse**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
hwwt3e	128	40,5	40,3	102,4	272,2
hwwt3e - hwwt3e	2x64	40,8	39,7	102,7	346,9
jaromir - hwwt3e	2x64	37,2	40,2	156,7	508,4

Tabelle 5.12: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing

Entfernungen bringen, die hier auftretende hohe Latenz also nicht mehr ganz so gravierend ins Gewicht fällt. Auf dem lokal simulierten Metacomputer haben für den Datentransport am Beginn und am Ende die Änderungen in PACX-MPI dagegen keinen positiven Effekt. Sie werden sogar ein klein wenig schlechter. Was aber durch die Verbesserung während der Berechnungsphase wieder mehr als ausgeglichen wird.

Eine weitere Messung wurde nun mit einem erheblich größeren Rechenbeispiel durchgeführt. Dabei wurde eine Simulation des seinerzeit geplanten europäischen Raumtransporters Hermes verwendet. Das Netz besteht aus 880 000 Gitterpunkten. Aufgrund des Speicherplatzbedarfs muß eine Simulation auf mindestens 96 Prozessoren durchgeführt werden, da sonst der lokale Speicher nicht für den lokal zu berechnenden Netzanteil ausreicht. Alle hier angegebenen Messungen wurden daher auf 128 Prozessoren durchgeführt. Es wurde dabei in allen Fällen der Löser mit der schwachen Kopplung und in den beiden Metacomputingmessungen die socketoptimierte PACX-MPI Bibliothek verwendet. Die dabei erzielten Ergebnisse, ersichtlich in Tabelle 5.12, ergeben ein interessantes Bild. Vergleicht man den Rechenlauf auf einer Partition mit dem Rechenlauf auf zwei Partitionen einer Maschine, so ist für den Teil der Iterationen, also den Teil, in dem die eigentliche Simulation abläuft, kein Laufzeitunterschied mehr festzustellen. Damit konnte hier wohl der gesamte Mehraufwand an Kommunikation erfolgreich hinter Berechnungen oder sonstiger Arbeit versteckt werden, so dass er sich nicht mehr in der Laufzeit niederschlägt. Einzig im Zeitunterschied für den Datentransfer zu den Prozessoren schlägt sich die schlechtere Netzwerkverbindung nieder. Wenn wir nun nur den inneren Schleifenteil der Applikation ansehen (Proces.) und damit den I/O und das Verteilen bzw. Einsammeln der Daten einmal beiseite lassen, stellen wir fest, dass selbst das interkontinentale Metacomputing nur einen Mehraufwand von starken 50% bedeutet. Damit kann es sich in manchen großen Fällen bereits lohnen, Metacomputer einzusetzen.

5.5.2 Einfluß der Netzaufteilung im Metacomputing

Die Verteilung des Netzes auf die Prozessoren spielt bei der Verwendung des Programms auf einem normalen Massiv Parallelen Rechner mit entsprechend leistungsfähigem Netzwerk nur eine untergeordnete Rolle. Es ist nur dafür zu sorgen, dass die Zellzahl in der ersten Dimension der Blöcke jeweils einigermaßen groß genug ist, so dass durch eine entsprechende Länge der inneren Schleifen Caches und insbesondere Features wie beispielsweise die Stream Buffers auf der T3E gut genutzt werden. Für den Einsatz des Programms im Metacomputing sind hingegen wieder ganz neue Überlegungen anzustellen. Dabei ist zu berücksichtigen, dass die externe Netzwerkverbindung in Bandbreite und Latenz etwa drei Größenordnungen schwächer ist als das interne Netz eines Massiv Parallelen Rechners wie der T3E. Daher sollte also die externe Verbindung so wenig wie möglich belastet werden. Es gilt also, die Kommunikation weit möglichst zu vermeiden und die unbedingt zu transferierenden Daten möglichst in großen Paketen zu verschicken, damit die Latenz möglichst selten auftritt. Dies kann man jetzt bei der Festlegung der Aufteilung des Netzes auf die Prozessoren berücksichtigen.

Am besten wäre es natürlich, all diese Vorgaben über die Verbindungsnetzwerke, im heterogenen Fall ebenso über die Prozessorleistung, wie auch die Informationen über den notwendigen Datenaustausch zwischen den Blöcken einem Lastverteilungswerkzeug zu übergeben. Dieses berechnet dann aus diesen Informationen die optimale, in der Realität eine der optimalen möglichst nahe kommende, Verteilung der Blöcke auf die Prozessoren. Leider sind solche Werkzeuge speziell für den Einsatz im Metacomputing heute erst in der Entwicklung und standen zur Zeit der hier beschriebenen Versuche noch nicht einmal im Ansatz zur Verfügung.

Daher beschränken wir uns hier darauf, die Anzahl der Prozesse, die Daten über die schwache externe Netzwerkverbindung miteinander austauschen, möglichst klein zu halten. Somit wurde in allen hier gezeigten Fällen nur eine zweidimensionale Verteilung des Netzes auf die Prozessoren gewählt. Da dann die Trennung zwischen den Partitionen in der zweiten Aufteilungsdimension erfolgt, entspricht die Anzahl der Prozessoren in der ersten genau der Zahl der Blöcke, die Daten über die externe Netzwerkverbindung in jeder Iteration austauschen. Die gewählten Verteilungen für das kleinere Netz wurden in Tabelle 5.6 bereits dargestellt, für das größere Netz beträgt sie 8x16x1=128, wenn es in den jeweiligen Tabellen nicht anders angeben ist.

5.5.3 Elimination der kollektiven Kommunikation

Wie schon bei der Auswertung der ersten Testergebnisse erwähnt, stellt die kollektive Kommunikation für die Strömungssimulation mit URANUS im Metacomputing ein Problem dar. Der Zeitbedarf für die kollektive Operation ist wie gemessen erheblich und er steigt auch noch fast linear mit der Zeit. Die Ursachen vor allem

für den fast linearen Anstieg sind unklar, da der Kommunikationsaufwand auf der Verbindung zwischen den Partitionen mit einer geeignet gewählten Implementierung in der Kopplungsbibliothek gegenüber dem kleineren Fall eigentlich konstant gehalten werden kann. Allerdings waren zum Zeitpunkt der Messungen die kollektiven Operationen in PACX-MPI noch nicht optimiert, was in den heute verwendeten Versionen natürlich der Fall ist. Jedenfalls ist der Aufwand für die kollektiven Operationen in dieser Applikation im Metacomputing zu hoch.

Eine einfache Möglichkeit, das Problem hier zu lösen wäre die Verwendung von nichtblockierenden kollektiven Operationen gewesen. Die kollektive Kommunikation wird zur Berechnung der Residuen benötigt. Die Werte der rechten Seite, die für die Berechnung verwendet werden stehen aber schon vor dem Lösen der Gleichungen zur Verfügung, während die Residuen selbst erst nach dem Lösen der Gleichungen benötigt werden. Damit könnte man elegant die kollektive Kommunikation hinter den Berechnungen und Punkt zu Punkt Nachrichten beim Gleichungslösen verstecken. Leider stehen diese nichtblockierenden kollektiven Kommunikationsroutinen im MPI-1 Standard [58] nicht zur Verfügung. Aufgrund der hier festgestellten Notwendigkeit für solche Kommunikationskonstrukte in MPI vor allem für Metacomputing und bei Verwendung von Verbindungsnetzwerken mit hoher Latenz, wurden von uns die nichtblockierenden kollektiven Routinen für den MPI-2 Standard vorgeschlagen. Leider folgte das Standardisierungskomitee nicht unserem Vorschlag und stufte die Notwendigkeit als gering ein, so dass die gewünschten Routinen nicht Teil der MPI-2 Standards [59] geworden sind. Immerhin wurden sie aber in das MPI-2 Journal of Development [60] aufgenommen.

Lösung für URANUS

Für das Simulationsprogramm URANUS bedeutet dies nun, dass es innerhalb der Applikation eine eigene Lösung für dieses Problem geben muss. Prinzipiell gibt es hierfür zwei Möglichkeiten: entweder es wird eine eigene nichtblockierende kollektive Kommunikation aus den nichtblockierenden Punkt zu Punkt Routinen implementiert, oder es wird versucht, die kollektive Kommunikation weitgehendst aus dem Programm zu eliminieren. Ersteres ist möglich, erfordert aber ein wenig Hintergrundwissen, um eine einigermaßen effiziente Kommunikationsstruktur mit Hilfe einer Art binären Baums aufzustellen. Allerdings wäre für eine effiziente Implementierung auf Metacomputern Information über die jeweilige Topologie des gerade verwendeten Metacomputers notwendig, die aber in der Applikation nicht vorhanden ist und auch nicht zur Verfügung gestellt werden kann, da auch hierfür keine standardisierte Schnittstelle existiert. Also blieb noch nach Möglichkeiten zu suchen, die kollektive Kommunikation zu entfernen.

Um die kollektive Kommunikation weitgehend entfernen zu können, müssen Möglichkeiten entwickelt werden, die Steuerparameter und den Konvergenzfort-

schritt auf andere Weise zu bestimmen. Die Idee ist nun, statt der globalen Konvergenz, den lokalen Konvergenzfortschritt für die Steuerung zu verwenden. Beim Updateparameter stellt dies kein Problem dar, da nichts dagegenspricht, den Anteil der neuen Lösung an der Gesamtlösung für einen Teil des Lösungsvektors anders zu wählen als für einen anderen. Etwas schwieriger ist dies bei der CFL-Zahl, die ja den Zeitschritt bestimmt. Hier kommt uns aber zugute, dass URANUS nur ein quasi instationäres Verfahren ist, das zwar instationär rechnet, aber auf eine stationäre Lösung hin. Damit kann auch der Zeitschritt lokal unterschiedlich sein, da nie ein Zwischenergebnis als Momentaufnahme eines Strömungszustands in der Berechnung verwendet wird. Falls aber eine stationäre Lösung existiert, was URANUS voraussetzt, wird diese auch mit lokal unterschiedlichen Zeitschritten erreicht. Damit dürfen auch die CFL-Zahlen lokal unterschiedlich sein und können somit lokal berechnet werden. Verschiedene Tests haben gezeigt, dass die Konvergenzgeschwindigkeit durch diese Änderungen im Verfahren fast nicht beeinflußt wird.

Bliebe noch die Verwendung der Residuen als Abbruchkriterium. In der Praxis hat sich bisher gezeigt, dass die Berechnungsingenieure lieber mit festen Iterationszahlen arbeiten, als die Abbruchkriterien zu verwenden. Also können wir diesen Aspekt außer Betracht lassen. Sollte sich in einer späteren Entwicklung zeigen, dass doch ein Abbruchkriterium notwendig ist, braucht man die dafür notwendigen globalen Residuen ja nicht in jeder Iteration zu bestimmen. Bei Iterationszahlen von bis zu 5000 im Falle der Berechnung von Lösungen für Hermes, würde es zum Beispiel sicher reichen, die globale Konvergenz nur alle 25 Iterationen zu bestimmen. Damit verteilt sich der Mehraufwand im Metacomputing auf 25 Iterationen und der durchschnittlich notwendige Mehraufwand für die Iterationen vom theoretischen Abbruchspunkt bis zum tatsächlichen liegt bei 12,5 Iterationen und damit bei etwa 0,25%. Daher wurde für die hier durchgeführten Messungen die kollektive Kommunikation vollständig entfernt und durch eine lokale Residuenauswertung ersetzt.

In Tabelle 5.13 sind die gemessenen Laufzeiten mit URANUS ohne kollektive Kommunikation für den Fall mit 880000 Zellen eingetragen. Die gewonnenen Laufzeiten dürfen nun allerdings nicht einfach als fixe Größen verwendet und mit den Messungen aus Tabelle 5.12 verglichen werden. Ursache dafür ist gerade die Elimination der kollektiven Kommunikation. Dadurch entfallen die Synchronisationspunkte in den Iterationen und damit dürfen die Messungen der Zeit für die Durchführung der Iterationen nicht mehr als global angesehen werden. Sie sind damit nur noch für den Prozess, auf dem gemessen wurde, gültig. Einzig die Gesamtlaufzeit darf noch als global gültige Zeit verwendet werden. Der Verlust der globalen Synchronisationspunkte hat einen weiteren Effekt: die T3E in Pittsburgh war inzwischen mit Knoten unterschiedlicher Performance ausgestattet worden. Es standen sowohl Knoten mit 300 MHz Prozessoren wie seinerzeit in Stuttgart, als auch Knoten mit 450 MHz Prozessoren zur Verfügung. In Tabelle 5.12 war

System	Anzahl Prozesse	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
hwwt3e	128	40,0	51,0	91,2	269,4
hwwt3e - hwwt3e	2x64	40,3	51,3	92,1	357,9
jaromir(300) - hwwt3e	2x64	40,1	72,4	112,8	586,9
jaromir(450) - hwwt3e	2x64	37,2	63,9	101,5	487,6

Tabelle 5.13: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation

darauf keine Rücksicht genommen woden, da durch den globalen Synchronisationspunkt ein Abgleich in jeder Iteration gegeben war. Daher werden nun alle Messungen auf dem interkontinentalen Metacomputer zweimal durchgeführt und zwar jeweils einmal mit 300 MHz CPU's und einmal mit 450 MHz CPU's in Pittsburgh. Die Frequenzsteigerung um 50% führt natürlich nicht zu einem Leistungsgewinn von ebenfalls 50% für die Applikation. Die Ursache dafür liegt in der Speicherbandbreite, die durch die Frequenzsteigerung der CPU allein nicht ebenso steigt, in diesem Fall vielmehr gleich geblieben ist. Trotzdem profitiert die Applikation ein wenig von der höheren CPU-Leistung, wie ebenfalls in Tabelle 5.12 bereits zu sehen ist. Der Teil, in dem das Gleichungssystem aufgestellt wird und der ohne Kommunikation abläuft, wird um etwa 9% schneller.

Es zeigt sich, dass der Wegfall der kollektiven Kommunikation auf einer lokalen Partition und im transkontinentalen Metacomputing die Gesamtlaufzeit verringert. Im Falle des interkontinentalen Metacomputing sogar um rund 20 Sekunden, was zwei Sekunden pro Iteration entspricht. Die Verwendung der 300 MHz Knoten in Pittsburgh hat natürlich auch für den sequentiellen Teil, der zu Beginn auf diesem System läuft, seine Auswirkungen. Daher ist diese Messung bezüglich der Gesamtlaufzeit nicht mit der aus Tabelle 5.12 vergleichbar. Darüberhinaus zeigen die Messungen an jenem Tag einen geringeren Netzwerkdurchsatz, was Vergleichsmessungen mit der Version aus dem nächsten Kapitel eindeutig zeigen. Leider stand nicht genügend Messzeit zur Verfügung, um alle Messungen, die Schwankungen zeigten, entsprechend zu wiederholen. Die Ursache für die erhöhte Laufzeit auf zwei Partitionen desselben Rechners ist unklar.

Um nun die Laufzeiten für den Iterationsteil und damit den Gewinn durch den Wegfall der kollektiven Kommunikation für diesen relevanten Programmteil wenigstens in etwa bestimmen zu können, müssen wir auf die Differenz der Gesamtlaufzeit zwischen den Messungen mit und ohne kollektive Kommunikaton

System	Anzahl Prozesse	Proces. bisher [s]	Diff. Gesamt [s]	Proces. neu [s]
hwwt3e	128	91,2	2,8	99,6
hwwt3e - hwwt3e	2x64	92,1	11,0	91,7
jaromir(300) - hwwt3e	2x64	-	-	151,5
jaromir(450) - hwwt3e	2x64	101,5	20,8	135,9

Tabelle 5.14: Abschätzung für die Laufzeit des Iterationsteils von URANUS bei einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation

zurückgreifen. Dieser wird dann von den Laufzeiten für die Iterationen im Fall mit kollektiver Kommunikation abgezogen. So erhält man dann in etwa die Laufzeit für den Iterationsteil in der letzten Messung. In Tabelle 5.14 sind daher die Laufzeitdifferenzen der Gesamtlaufzeit und die daraus bestimmte Laufzeit für den Iterationsteil mit eingetragen. Bei der Analyse der Werte wird in etwa der Gewinn deutlich. Allerdings führt bei dieser Berechnungsweise jede Schwankung in der Kommunikationsgeschwindigkeit beim Datentransfer zu Beginn und am Ende der Simulation zu bisweilen erheblichen Ungenauigkeiten bei der Bestimmung der tatsächlichen Zeit für die Iterationen, was bei der lokalen Messung auf zwei Partitionen auch deutlich zu sehen ist. Hier kann zum Beispiel eine anderweitige Nutzung der Netzwerk-Schnittstelle zu den Ungenauigkeiten führen. Sicher ist nur, dass der Verzicht auf die kollektive Kommunikation eine je nach Latenz erhebliche Verbesserung der Laufzeit zur Folge hat. Die 136 Sekunden für den Berechnungsteil bedeuten eine etwa um 33% höhere Rechenzeit, aber zumindest verspricht dies schon einen Gewinn durch die Verwendung des Metacomputers. Für die Messung auf den 300 MHz CPU's wurde die Zeit für den Processingteil aufgrund der oben genannten Probleme durch eine extra Messung für diesen Programmteil gewonnen. Diese Messung ändert allerdings durch die notwendige Synchronisation das Kommunikationsverhalten ein wenig, so dass das Ergebnis nur qualitativ zu werten ist und eine Obergrenze für die tatsächlich notwendige Zeit für den Iterationsteil darstellt.

5.5.4 Message Pipelining

Um eine weitere Verbesserung der Programmlaufzeit im Iterationsteil zu erreichen und diese weiter in Richtung auf die lokal erreichbaren Zeiten zu drücken, bleibt nun nur noch eine nochmalige Verbesserung der Kommunikation im Löser. Um die Transferzeit weiter hinter Berechnungen verstecken zu können, kann nur der

Rechenaufwand pro Kommunikation weiter erhöht werden. Betrachten wir dazu den Kommunikationsaufwand im Löser genauer:

Bei der gewählten Aufteilung tauschen 8 Prozessoren mit jeweils einem Nachbarprozessor auf der anderen Partition des Metacomputers Daten aus. Die Teilgitter haben dabei eine Größe von 18x96 Zellen, was durch die Randwerte und den fünfelementigen Erhaltungsvektor bei vier Byte Fließkommazahlen zu einem Datenvolumen von 39.200 Byte beim Lösen und von 78.400 Byte beim Ergebnisaustauch, jeweils pro Prozessor, führt. Mit den gemessenen Leitungsdaten unter MPI aus Tabelle 5.1 bedeutet das eine Transferzeit von 0,77 s pro Prozessor für den interkontinentalen Metacomputer. Fügen wir noch die Latenz hinzu, dauert der Transfer also 0,92 s pro Prozessor und Iteration. Bei acht Datentransfers zwischen den Metacomputern also 7,36 Sekunden pro Iteration. Bei der lokalen Simulation des Metacomputers ergibt sich für diese Rechnung eine Gesamttransferzeit von 3,76 Sekunden. Während sich letztere gerade noch in der Zeit pro Lösungsdurchlauf von etwa 4 Sekunden (vgl. Tabelle 5.12) verbergen lässt, stehen wir im ersteren Fall vor einem erheblichen Problem. Dieses Verhalten ist schon bei den Messungen in Tabelle 5.12 zu erkennen. Die Durchlaufzeit für den Iterationsteil weist hier ja keinen Unterschied zwischen der Rechnung auf einer Partition und der auf zwei lokalen Partitionen auf. Im Fall des interkontinentalen Test ist diese Laufzeit aber erheblich länger. Der dabei festzustellende Zeitmehrbedarf liegt bei 54 Sekunden. Wie wir aus Tabelle 5.14 wissen, gehen etwa 20 Sekunden davon auf das Konto der kollektiven Kommunikation. Damit bleibt ein Zeitmehrbedarf durch den Löser von etwa 34 Sekunden, bei 10 Iterationen also 3,4 Sekunden pro Durchlauf. Und das ist wiederum genau die Zeitdifferenz zwischen dem Aufwand für den Datentransfer von 7,36 Sekunden und der Zeit pro Löserdurchlauf von 4 Sekunden. Unsere Überlegungen decken sich also genau mit den Messergebnissen.

Damit wird nun klar, dass die benötigte Transferzeit für die Daten im Löser nicht hinter den Berechnungen während des Lösens versteckt werden kann. Also bleibt nur, diesen Transfer auch hinter den Berechnungen beim Aufstellen des Gleichungssystems zu verbergen. Dazu muss aber die gesamte Kommunikationsstruktur geändert werden. Außerdem ist in der Applikation im vorhinein ja nicht bekannt, welcher Datentransfer über die langsame Verbindung stattfindet. Daher wurde für den Datenaustausch beim Lösen folgendes Konzept entwickelt: sobald nach der Berechnung verfügbar, schickt jeder Prozeß seine Randergebnisse ab und startet den Empfang der entsprechenden Werte vom Nachbarn. Sollten diese Werte nicht verfügbar sein, wenn sie benötigt werden, wird einfach auf entsprechende ältere Werte des Nachbarn aus einer früheren Iteration zugegriffen. Diese müssen natürlich nach dem Empfang so lange bereit gehalten werden. Dazu sind die Sende- und Empfangspuffer auf jedem Prozess als Ringpuffer organisiert, der zusätzlich zum Platz für den aktuellen Eintrag Platz für eine einstellbare Anzahl älterer Werte hat. Hier werden dann die Werte einfach in einem logischen Ring zwischengespeichert bis sie abgerufen werden. So kann der Empfänger immer auf

System	**Anzahl Prozesse**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
hwwt3e - hwwt3e	2x64	40,4	36,6	77,3	332,7
jaromir(300) - hwwt3e	2x64	40,4	40,0	80,4	562,7
jaromir(450) - hwwt3e	2x64	36,9	37,2	74,5	482,8

Tabelle 5.15: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining

ältere Werte zugreifen. Damit braucht die Applikation aber auch kein Wissen über die Topologie des Metacomputers. Jeder Prozeß greift einfach die neuesten verfügbaren Werte ab. Liegt der Nachbar auf der gleichen Partition, sind die Werte mit hoher Wahrscheinlichkeit die aktuellen aus der derzeitigen Iteration, ist der Nachbar dagegen entfernt, werden sie mit hoher Wahrscheinlichkeit aus einer vorigen Iteration stammen.

Da die Größe der Ringpuffer und der zugehörigen Datenstrukturen einfach einstellbar implementiert wurde, kann das maximale Alter der verwendeten Werte entsprechend einfach festgelegt werden. Sind dabei die Werte des vorgegebenen maximalen Alters nicht verfügbar, wird auf sie gewartet. Da bei diesem Verfahren die Nachrichten wie am Fließband nacheinander verschickt und bei Verfügbarkeit abgerufen werden, wurde das Verfahren "Message Pipelining" getauft.

Natürlich kann die Konvergenzrate mit diesem Verfahren nicht aufrecht erhalten werden. Daher wurden mehrere Simulationsläufe durchgeführt, bei denen das zulässige Alter der Nachbarwerte, die in der aktuellen Iteration noch verwendet werden dürfen, variiert wurde. Setzt man diesen Wert auf null, so dürfen nur Werte aus der aktuellen Iteration verwendet werden und das neue Verfahren verhält sich wie das bisherige. Wird dagegen zugelassen, dass Werte aus der letzten Iteration verwendet werden dürfen, so sinkt die Konvergenzrate um etwa 30 % ab. Weitere Versuche bei denen dieser Parameter auf 2 und höher gesetzt wurde, waren leider nicht mehr erfolgreich: in all diesen Simulationsläufen divergierte das Verfahren. Hier war damit also die Kopplung zu schwach geworden.

Mit dem auf das Message Pipelining angepaßten Löser wurden dann wieder Messungen auf dem Hermesbeispiel mit 880.000 Zellen durchgeführt. Dabei wurde der Löser so eingestellt, dass Nachbarwerte der letzten Iteration verwendet werden dürfen. Die erzielten Laufzeiten sind in Tabelle 5.15 dargestellt, wobei die Messung auf den 300 MHz Prozessoren nicht die beste ist, sondern in der gleichen Sitzung gemessen wurde, wie die in Tabelle 5.13. Wie wir feststellen können,

System	**Anzahl Prozesse**	**Proces. bisher** [s]	**Diff. Gesamt** [s]	**Proces. neu** [s]
hwwt3e - hwwt3e	2x64	77,3	14,2	88,5
jaromir(300) - hwwt3e	2x64	81,1	24,2	127,3
jaromir(450) - hwwt3e	2x64	74,5	25,6	131,1

Tabelle 5.16: Abschätzung für die Laufzeit des Iterationsteils von URANUS bei einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation mit Message-Pipelining

bringt das Message-Pipelining auf jeden Fall eine weitere Verbesserung. Um eine Abschätzung für den Gewinn zu bekommen, bilden wir wieder die Differenz der Gesamtlaufzeit und zwar wieder zu den in Tabelle 5.12 dargestellten Messungen, bei denen wir für den Processingteil noch von einer sinnvollen Messung sprechen können. Damit erhalten wir die in Tabelle 5.16 dargestellten Ergebnisse für den Processingteil. Für die Messung auf den 300 MHz Knoten in Pittsburgh wurde ein anderes Verfahren gewählt. Hier nehmen wir die Abschätzung für den Processingteil aus Tabelle 5.14 zu Hilfe und verwenden die Laufzeitdifferenz zu der Messung aus Tabelle 5.13. Dies sollte ebenso zu einer vertretbaren Abschätzung führen. Damit liegt die Zeit für den Berechnungsanteil bei etwa 130 Sekunden, gegenüber etwa 135 Sekunden ohne kollektive Kommunikation und ohne Message-Pipelining. Interessant ist hier auch die Messung auf den lokalen zwei Partitionen: es ergibt sich eine Laufzeit von 88,5 Sekunden für den Rechenteil. Das sind 11 Sekunden weniger als für die Rechnung ohne Message-Pipelining und ohne kollektive Kommunikation auf einer Partition der Stuttgarter T3E (vgl. Tabelle 5.14). Unter Einbeziehung der 30% Konvergenzverlust, der durch eine entsprechende Iterationszahl ausgeglichen werden muss, bedeutet das eine Äquivalenzlaufzeit von 1,3x88,5=115,05 Sekunden. Dies entspricht gegenüber dem erwähnten Vergleichswert einem Mehraufwand von 15,5%, wenn man davon ausgeht, dass die Schwankungen der Gesamtlaufzeit lokal nicht so erheblich sind. Das würde also bedeuten, dass sich Metacomputing schon lohnen kann, wenn man verschiedene Systeme hat und den Rechenraum netzmäßig gesehen nicht verläßt. Aber da ist es sicher besser, die Systeme durch Zusammenschalten der Hochgeschwindigkeitsnetzwerke zu verbinden oder gleich ein entsprechend größeres monolithisches System anzuschaffen.

Aber zurück zum transkontinentalen System. Um den Zeitanteil für den Berechnungsteil etwas abzusichern, wurde in derselben Sitzung mit der 300 MHz Partition in Pittsburgh Messungen mit jeweils 64 Prozessoren pro Partition durch-

System	**Anzahl Iterationen**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
jaromir(300) - hwwt3e	10	40,4	40,0	81,1	519,1
jaromir(300) - hwwt3e	20	80,8	87,4	169,0	648,4

Tabelle 5.17: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining auf der 300 MHz Partition mit 10 und 20 Iterationen

System	**Anzahl Prozesse**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
jaromir(300) - hwwt3e	2x64	40,2	39,9	139,0	628,5
jaromir(450) - hwwt3e	2x64	37,0	37,5	116,7	507,1

Tabelle 5.18: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining, mit speziellem Messverfahren im Rechenteil

geführt, und zwar jeweils mit 10 und 20 Iterationen im Berechnungsteil. Damit sollte man aus der Laufzeitdifferenz ebenso Rückschlüsse auf den Zeitanteil für die Iterationen ziehen können. Da die Konfiguration denen aus den vorigen Messungen vergleichbar ist, sollten auch die Meßwerte vergleichbar sein. Tabelle 5.17 zeigt die gewonnenen Meßwerte, wobei diesmal Messungen aus den besten Sitzungen verwandt wurden. Bildet man nun die Differenz der Gesamtlaufzeit, erhalten wir 129,3 Sekunden, was wieder in etwa in derselben Größe liegt wie unser voriges Ergebnis. Außerdem wurden die Messungen aus Tabelle 5.15 nochmals durchgeführt, diesmal allerdings mit der oben erklärten speziellen Meßmethode für den Iterationsteil. Damit erhalten wir die in Tabelle 5.18 dargestellten Ergebnisse. Die Messung zeigt ein sehr gutes Verhalten mit 116,7 Sekunden für die 450 MHz Partition und mit 139,0 Sekunden wieder eine Abschätzung für die 300 MHz Partition.

Ursache für den erheblich besseren Wert für die 450 MHz Partition ist, dass durch den Performanceunterschied die durch die Messung hervorgerufene Synchronisation nicht mehr ins Gewicht fällt. Sicher ist, dass wir mit dem Message-Pipelining bei Zeiten zwischen 116 und 130 Sekunden pro 10 Iterationen im Rechenteil liegen. Gegenüber etwa 135 bzw. 150 Sekunden ohne kollektive Kommunikation und ohne Message-Pipelining ist dies zwar einiges besser, aber leider

Anzahl Prozesse	Aufteilung	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
2x64	8x16x1	40,1	39,9	80,4	549,0
2x128	16x16x1	21,7	24,9	46,8	619,9

Tabelle 5.19: Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation und mit Message Pipelining mit unterschiedlicher Prozessorzahl

doch nicht schnell genug, um die 30% Verlust durch die schlechtere Konvergenz aufzufangen.

5.5.5 Weitere Ergebnisse

Natürlich wurden auch noch Messungen auf höheren Prozessorzahlen durchgeführt. Als erstes auf zwei Partitionen mit jeweils 128 und dann auf zwei Partitionen mit jeweils 256 Prozessoren, jeweils auf Prozessoren mit 300 MHz in Pittsburgh. In Tabelle 5.19 sind aus der gleichen Messsitzung auf dem interkontinentalen Metacomputer jeweils ein Ergebnis auf 2x64 und 2x128 Prozessoren gezeigt, jeweils ohne kollektive Kommunikation und mit Message-Pipelining.

Wie sofort zu sehen ist, bringt gemäß der Gesamtlaufzeit die Erhöhung der Prozessorzahl keinen Vorteil, im Gegenteil, die Gesamtlaufzeit vergrößert sich sogar. Das liegt aber nicht unbedingt an einer Erhöhung des Zeitbedarfs im Iterationsteil. Vielmehr spielt hier auch eine erhebliche Rolle, dass nun die Daten für die doppelte Anzahl von Blöcken erst einmal über den Atlantik transferiert werden muss, die Latenz also doppelt so oft anfällt, was natürlich auch für den Datenaustausch während des Iterationsteils gilt. Um hier Gewißheit zu erlangen, wurden weitere Messungen mit 2x128 Prozessoren durchgeführt, wobei diesmal jeweils die Programmversion mit der speziellen Messmethode für den Iterationsteil zum Einsatz kam. Die gewonnenen Meßergebnisse sind in Tabelle 5.20 gezeigt, wobei die erste Messung mit einem Programmlauf ohne kollektive Kommunikation durchgeführt wurde, die zweite auch mit Message-Pipelining.

Wie gewohnt dürfen wir durch die zusätzliche Synchronisation die Gesamtlaufzeiten nicht mit den Messungen aus Tabelle 5.19 vergleichen. Dafür besitzen wir nun bessere Aussagen über den Zeitanteil des Iterationsteils. Für den Lauf nur ohne kollektive Operationen benötigen die zehn durchgeführten Schleifendurchläufe 113,4 Sekunden, mit zusätzlichem Message-Pipelining gewinnen wir weitere 6,4 Sekunden. Wie wir sehen erreichen wir gegenüber den Messungen auf 2x64 Prozessoren noch eine Verbesserung um 38 Sekunden nur ohne kollektive

Anzahl Prozesse	**Aufteilung**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
2x128	16x16x1	21,6	75,0	113,4	646,7
2x128	16x16x1	21,7	30,1	107,0	633,1

Tabelle 5.20: Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, einmal ohne und einmal mit Message Pipelining, spezielle Messung des Iterationsteils

Kommunikation (vgl. Tabelle 5.14), bzw. um 32 Sekunden mit Message-Pipelining (vgl. Tabelle 5.18).

Überlegen wir uns, wie hoch der Aufwand für den Datentransfer tatsächlich ist. Dazu verwenden wir wieder die gemessene Latenz und Bandbreite aus Tabelle 5.1. Durch die geänderte Aufteilung erhalten wir nun unter Berücksichtigung der Ränder ein Datenvolumen von 21560 Byte für den Datenaustausch während der Lösung und von 43120 Byte für den Austausch nach dem Lösungsschritt. Zusammen mit der Latenz kommen wir so auf einen Zeitbedarf von 0,57 Sekunden pro Prozessor Schnittstelle. Da nun 16 Schnittstellen Daten über den Atlantik austauschen bedeutet dies einen Zeitbedarf von 9,13 Sekunden pro Iteration.

Bedenkt man nun, dass bei der Messung ohne Message-Pipelining eigentlich keine Überschneidungsmöglichkeit zwischen dem Datenaustausch im Löser und dem Programmteil, in dem die neuen Gleichungen aufgestellt werden, besteht, stimmen die erzielten Ergebnisse wieder fast genau mit den theoretisch bestimmten überein, da der Datenaustausch etwa 91 Sekunden benötigt, der Teil zum Gleichungen aufstellen etwa 22 Sekunden, womit wir schon die erzielten 113 Sekunden für den gesamten Lösungsteil erreicht hätten. Das Message Pipelining bringt hier wieder eine geringfügige Verbesserung natürlich auch wieder auf Kosten der Konvergenzgeschwindigkeit.

Interessant ist nun auch folgende Messung, die mit 2x256 Prozessoren durchgeführt wurde. Allerdings mußte hierzu das zu berechnende Netz geändert werden, da sonst in der zweiten Dimension nur 2 Zellen pro Block geblieben wären. Wenn man dann noch bedenkt, dass der Überlappungsbereich in jeder Richtung ebenfalls zwei Zellen beträgt, ist leicht einzusehen, dass so eine Aufteilung nicht mehr sinnvoll ist. Daher wurde das Netz wie folgt geändert. Bedingung war dabei, dass sich die Netzgröße insgesamt nicht ändert, um die Ergebnisse wenigstens mit dem Lauf auf einem Rechner vergleichen zu können. Daher wurde beim bestehenden Netz mit einer Zellaufteilung von 144x64x96 die Zellzahl in der zweiten Dimension verdoppelt und in der dritten dafür halbiert. Wir verwenden nun also ein Netz der Größe 144x128x48. Damit kann das Netz nun auf 16x32x1 Pro-

Netzgröße	**Aufteilung**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
144x64x96	16x16x1	86,5	419,8	507,5	1071,9
144x128x48	16x16x1	77,2	180,1	257,8	788,6

Tabelle 5.21: Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, ohne Message Pipelining, mit unterschiedlicher Zellverteilung in den verwendeten Netzen

zessoren verteilt werden. Eine Verteilung des Netzes auf die Prozessoren in drei Dimensionen kommt ja aus bereits vorgestellten Gründen nicht in Frage. Wichtig bei dieser neuen Aufteilung ist nun, dass damit auch das über den Atlantik zu übertragende Datenvolumen sinkt. Der Schnitt bzw. die Querschnittsfläche an der die Schnittstellen liegen, die einen Nachbarn auf der anderen Seite des Ozeans haben, werden durch die erste und dritte Dimension bestimmt, da in der zweiten geschnitten wird. In der dritten Dimension wird das Netz ja nicht verteilt. Und da das Netz anhand der Prozessornummer vergeben wird, liegt die erste Hälfte der zweiten Dimension auf dem einen Rechner und die zweite auf dem anderen. Da aber die zweite Dimension in der Zellzahl verdoppelt werden muß, kann nur die erste oder dritte Dimension in der Zellzahl halbiert werden. Damit vermindert sich aber das über den Atlantik zu übertragende Datenvolumen während der Berechnung um die Hälfte.

Natürlich hätte man auch die Zellzahl in der ersten Dimension erhöhen können und die in der dritten vermindern, dann wäre zwar das Datenvolumen gleich geblieben, aber die Anzahl der Prozessoren, die Nachbarn auf dem anderen Rechner des Metacomputers haben, hätte sich für den Fall mit 2x256 Prozessoren verdoppelt. Dies hätte sich dann durch die Latenz wieder negativ auf die Berechnung ausgewirkt. Damit war die gewählte Methode die für das Metacomputing besser geeignete.

Tabelle 5.21 zeigt eine Vergleichsrechnung mit den beiden unterschiedlichen Netzen jeweils auf 2x128 Prozessoren, jeweils 300 MHz Patitionen in Pittsburgh, diesmal 40 Iterationen mit dem Löser ohne kollektive Kommunikation aber auch ohne Message-Pipelining. Auf den ersten Blick fällt sofort der dramatische Unterschied in der Gesamtlaufzeit auf. Bis auf die Schwankungen durch die Netzwerkleistung dürfte sich aber der Zeitaufwand für den Datentransport zu Beginn und am Ende der Berechnung jeweils nicht unterscheiden, da beides mal das gleiche Datenvolumen auf die gleiche Anzahl von Prozessoren über die langsame externe Verbindung übertragen werden muß. Damit kommt der Zeitvorteil sicher aus dem Berechnungsteil, wie es ja aus der Vorüberlegung zu erwarten war.

Netzgröße	**Aufteilung**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
144x128x48	16x32x1	43,3	125,7	247,5	942,0

Tabelle 5.22: Ergebnisse von URANUS mit einem 880000 Zellen Netz im interkontinentalen Metacomputing ohne kollektive Kommunikation, mit Message Pipelining auf einem anders aufgeteilten Netz mit 2x256 Prozessoren

Für die Messung mit 2x256 Prozessoren wird nun wieder eine Messung mit der Programmversion mit der speziellen Synchronisation für die Zeitmessung verwendet, um einfach Aussagen über die Laufzeit des Iterationsteils zu erhalten. Das Programm wurde dabei nach wie vor auf der amerikanischen Seite auf den 300 MHz CPU's gestartet. Tabelle 5.22 zeigt die Ergebnisse. Die Gesamtlaufzeit läßt sofort vermuten, dass sich diese hohe Prozessorzahl nicht lohnt, obwohl darin auch der Aufwand für die Übertragung der Daten zu und von doppelt sovielen Prozessoren zu Beginn beziehungsweise nach Beendigung des Berechnungsteils enthalten ist. Überlegen wir wieder kurz den Aufwand für die Datenübertragung: insgesamt werden pro Block und Iteration 11000 + 22000 Byte übertragen. Dies führt unter Verwendung wiederum der Meßwerte aus Tabelle 5.1 zu einem Anteil von etwa 0,21 Sekunden. Zuzüglich der ebenfalls bestimmten Latenz von zwei Übertragungsschritten kommen wir auf 0,36 Sekunden pro Iteration und Block. Da jeweils 16 Prozessoren einer Partition Daten über das externe Netzwerk miteinander austauschen, bedeutet das einen Aufwand von 5,76 Sekunden pro Iteration, bei 40 Iterationen also von 230 Sekunden. Die gemessenen 247,5 Sekunden für den Processingteil werden also durch die Datenübertragung dominiert und stimmen bis auf einen Fehler von 7,5% mit den Messwerten überein.

Da wir aus vorangegangenen Messungen wissen, dass im Optimalfall das Aufstellen und Lösen der Gleichungen in etwa gleich lange dauert, können wir den realen Aufwand für die Berechnung in diesem Lauf mit etwa 85 Sekunden abschätzen, da der Messwert des Gleichungen-Aufstellens nicht durch irgendwelche Synchronisationszeiten beeinflußt wird. Dadurch bestimmt sich aber der Overhead durch die Kommunikation zu 160 Sekunden, also fast doppelt so lang, wie die eigentlichen Berechnungen dauern. Damit ist klar, dass die Benutzung von 2x256 Prozessoren für das gewählte Problem ineffizient ist. Allerdings kann man aus obigen Messungen mit 2x64 Prozessoren folgern, dass dies nicht gilt, wenn mit einer geeigneten Aufteilung ein Problem gelöst wird, dass groß genug ist, also besonders dann, wenn ein Problem auf einem Superrechner nicht mehr zu lösen ist.

Immerhin hat die eben durchgeführte Messung aber gezeigt, dass das Message-Pipelining in der Lage ist, die Berechnung fast vollständig hinter der Kommuni-

kation zu verstecken, denn sonst hätte der Zeitaufwand für den Berechnungsteil in der gerade betrachteten Messung noch erheblich höher ausfallen müssen.

5.5.6 Änderungen im Pre- und Postprocessing

Bliebe noch das Pre- und Postprocessing. Dessen Anteil wird gegenüber den hier vorgestellten Testläufen bei einem realen Rechenlauf zwar erheblich kleiner, es ist aber dennoch von Vorteil, wenn sich der Aufwand dafür noch verkürzen ließe.

Die Idee besteht hier in einer einfachen Entkopplung des Datentransfers von der tatsächlichen Rechnung. Dazu wird die notwendige Eingabedatei mit dem Rechengitter einfach auf beiden Teilrechnern, die später den Metacomputer bilden, hinterlegt. Das geschieht, indem man sie über die normale Netzverbindung von Hand auf den anderen Rechner kopiert. Ebenso werden die Ergebnisse in zwei Dateien geschrieben und hinterher von Hand wieder zusammengefaßt. Die Anpassungen im Simulationsprogramm bestehen nun darin, dass auf jedem der beiden Rechner ein Prozeß die Daten einliest und verteilt. Um sicherzustellen, dass diese beiden ausgezeichneten Prozesse auch auf unterschiedlichen Rechnern liegen, wird dazu der erste Prozess (Rank 0) und der letzte Prozeß (Rank Anzahl-1) verwendet. Jeder dieser beiden liest nun die Datei ein und versorgt die untere bzw. die obere Hälfte der Prozessoren mit dem jeweiligen Teil des Rechennetzes. Falls die Prozessoren des Metacomputers symmetrisch auf die beiden realen Systeme verteilt sind, ist im Pre- und Postprocessing keine Kommunikation mehr notwendig. Ist die Prozessorverteilung asymmetrisch, ist Kommunikation erforderlich, die aber gegenüber der vorher notwendigen erheblich geringer ist.

Besteht der Metacomputer aus mehr als zwei realen Systemen, ist auf jeden Fall Kommunikation erforderlich. Allerdings ist für diese der Zeitaufwand gegenüber der alten Programmversion erheblich geringer, da jetzt von zwei Seiten aus die notwendigen Daten verschickt werden. Werden zum Beispiel drei Rechner mit jeweils gleicher Anzahl von Prozessoren verwendet, versorgt jeder der beiden I/O-Prozesse "sein" System lokal, das dritte wird dann von beiden Seiten aus über typischerweise verschiedene Weitverkehrsverbindungen quasi parallel versorgt. So sinkt der Zeitaufwand für das Pre- und Postprocessing gegenüber der bisherigen Programmversion in diesem Fall um den Faktor vier.

Leider existieren für diese Massnahme keine tragfähigen Ergebnisse, da zu diesem Zeitpunkt die dedizierte Netzwerkverbindung geändert wurde. Nach der Änderung kam es zu Paketverlusten, was Messungen für Metacomputingläufe gänzlich unmöglich machte. Messungen über Standardinternet sind wegen der enormen Schwankungsbreite der Laufzeiten leider ebenfalls nicht aussagekräftig. Die Erkenntnisse aus diesen Metacomputingversuchen führten aber zu einem gänzlich anderen Vorgehen für das Pre- und Postprocessing in der in den vorigen Kapiteln vorgestellten Multiblockversion von URANUS:

1. Das Netz ist nicht mehr in einer Datei gespeichert, sondern jeder Block

des Multiblocknetzes besitzt eine eigene Datei. So kann aus verschiedenen Dateien parallel gelesen werden.

2. Jeder Prozessor liest "seinen" Rechennetzteil selbst direkt aus der zugehörigen Blockdatei. So können auch die Dateien parallel gelesen werden, sofern die Hardware das erlaubt. Außerdem ist diese Methode relativ einfach auf paralleles Einlesen mit MPI-I/O umzustellen, sobald der MPI-2 Standard auf allen Plattformen unterstützt wird.

3. Nach dem Rechenlauf schreibt jeder Prozeß sein Ergebnis und seinen Gitteranteil in jeweils eine eigene Datei. So wird ein Datenrücktransport vermieden. Und außerdem kann parallel geschrieben werden. Darüber hinaus wird ein Fortsetzen der Berechnung mit der gleichen Prozessorzahl so erheblich vereinfacht, da die Daten nicht neu verteilt werden müssen. Für die Auswertung werden die Dateien dann vom Visualisierungwerkzeug, falls vom Benutzer gewünscht, bei der Konvertierung zusammengefaßt. Dies erfordert keinen Mehraufwand, da bei der Ergebniskonvertierung sowieso die gesamten Daten des Ergebnisses durchgegangen werden müssen.

5.6 Diskussion und Folgerungen

Um nochmals die Schwierigkeiten und die Problematik der ganzen Metacomputingmessungen aufzuzeigen -und gleichzeitig eine Problematik des gesamten verteilten Rechnens über große Distanzen- möchte ich noch einige Messungen vorstellen. Hier sind es Messungen mit dem schwach koppelnden Löser ohne kollektive Kommunikation mit Message-Pipelining jeweils mit 10 Iterationen im Berechnungsteil und auf der 300 MHz Partition in Pittsburgh. Der einzige Unterschied ist der Tag, an dem gemessen wurde. Tabelle 5.23 zeigt die Ergebnisse. Wie leicht zu sehen ist, zeigt sich je nach Tag ein erheblicher Unterschied in der Gesamtlaufzeit. Dieser beträgt zwischen der besten und der schlechtesten Messung immerhin fast 94 Sekunden, also über 18%. Dies zeigt nochmals das Problem, daraus die richtigen Werte für die Abschätzung des Aufwands im Rechenteil ziehen zu können. Gleichzeitig zeigt es aber auch den Einfluß des Netzwerkes. Da die beiden Maschinen für die Messungen nicht komplett reserviert waren, könnte ein anderer Prozeß auf gemeinsam genutzte Resourcen wie Netzwerk oder Festplatten zugegriffen haben. Da bei lokalen Messungen nie solche Schwankungen aufgetreten sind, scheiden die Festplatten aber als Ursache aus. Bleibt nur noch das Netzwerk. Da die Atlantikleitung mit einer festen Bandbreite von 2 MBit/s gebucht war, kommen dafür nur die Teilstrecken innerhalb des Rechenzentrums in Stuttgart und die Strecke zwischen Chicago und dem PSC in frage. Beide Teile sollten aber eigentlich ausreichend Bandbreitenreserven bieten. Trotzdem sind diese doch recht erheblichen Schwankungen aufgetreten. Hieraus läßt sich folgern,

System	**Tag der Messung**	**GA** [s]	**GL** [s]	**Proces.** [s]	**Gesamt** [s]
jaromir(300) - hwwt3e	10.10.97	40,4	40,0	81,1	519,1
jaromir(300) - hwwt3e	22.10.97	40,1	39,9	80,4	562,7
jaromir(300) - hwwt3e	29.10.97	40,1	39,9	80,4	612,9
jaromir(300) - hwwt3e	5.11.97	40,1	39,9	80,4	549,0

Tabelle 5.23: Ergebnisse von URANUS mit einem 880000 Zellen Netz im Metacomputing ohne kollektive Kommunikation und mit Message Pipelining, gemessen an unterschiedlichen Tagen

dass selbst eine als genügend angesehene Bandbreite eine fest zugesicherte Servicequalität von Endpunkt zu Endpunkt nicht ersetzen kann. Am Rande sei noch bemerkt, daß es von zusätzlichen Messungen mit den ganzen Maschinen, keine sinnvollen Ergebnisse gibt.

Ursache dafür war wiederum die Netzwerkverbindung: für diese Messungen wurde die Netzwerkbandbreite auf der gebuchten Verbindung von 2 MBit/s auf 10 MBit/s erhöht. Leider trat ab jenem Zeitpunkt ein Paketverlust von etwa 5% der Pakete auf, der sich innerhalb der Zeit, in der die Leitung gebucht war, auch nicht beseitigen ließ, obwohl sich alle beteiligten Netzwerkprovider und Institute (insgesamt 8!) redlich darum bemühten. Leider hat selbst dieser geringe Paketverlust einen erheblichen Einfluß auf die nutzbare Bandbreite der Verbindung, da viel Zeit mit dem Wiederaufsetzen nach einem Paketverlust und der notwendigen Sendewiederholung verbracht wird. Dies führte bei uns dazu, dass für die Messungen nur eine maximale Bandbreite von etwa 30 kB/s ermittelt werden konnte. Außerdem war diese Bandbreite je nach aktueller Verlustrate sehr stark schwankend, so dass keine weiteren, wirklich zuverlässigen Messungen durchgeführt werden konnten.

Auch dieses Vorkommnis zeigt wieder die Anfälligkeit des Metacomputing vor allem auf Einflüsse durch das Verbindungsnetzwerk, aber auch, welch ein manueller Organisationsaufwand noch immer hinter solchen Rechnungen steckt. Zukünftig müssen daher Netzwerkverbindungen mit garantierter Bandbreite sicher funktionieren und automatisch gebucht werden können, ebenso wie die entsprechend notwendigen Computerresourcen.

System	Anzahl Prozesse	GA [s]	GL [s]	Proces. [s]	Gesamt [s]
jaromir - turing - hwwt3e	3x32	92,2	175,0	598,0	611,0
jaromir - turing - hwwt3e	3x64	43,7	103,9	299,4	311,0

Tabelle 5.24: Ergebnisse von Multiblock URANUS mit einem 1,536 Millionen Zellen Netz im Metacomputing

5.7 Neue Ergebnisse

Die Ergebnisse aus den Untersuchungen und Versuchen sind bereits in die Konzeption des Multiblockprogramms eingeflossen. Vor allem die Ideen für die Erhöhung der Parallelität im Preprocessing und für den I/O, wie sie in Kapitel 3 dargestellt wurden, fußen auf den Ergebnissen und Entwicklungen für das Metacomputing. Daher wurden mit der Multiblockversion des Programms nochmals einige Messungen durchgeführt. In Tabelle 5.24 sind mit der Standard Multiblockversion des Programms und zyklischer Lastverteilung (vgl. Kapitel 4) erzielte Ergebnisse zu sehen. Dabei wurde auf 3 verschiedenen T3E's in Stuttgart (hwwt3e), Pittsburgh (jaromir) und Manchester (turing) gemessen. Dabei verfügten alle T3E's über 450 MHz Prozessoren. Verwendet wurde nochmals ein Colibrinetz aus dem Benchmark mit 5 Blöcken und insgesamt 1.536.000 Zellen. Es zeigt sich, dass der Zeitbedarf für den I/O durch die parallele Abwicklung auf dem jeweils lokalen System zwar abhängig von der Zahl der Blöcke im Netz zum Start (hier 6) ist, aber durch das parallele Einlesen und die parallele Ausgabe ist der Zeitbedarf dafür doch recht gering. Vor allem wenn man bedenkt, dass wiederum nur 10 Iterationen gerechnet wurden und hier der Zeitanteil für das Pre- und Postprocessing schon fast nicht mehr ins Gewicht fällt. Äußerst interessant ist aber das Verhalten der zwei unterschiedlichen Läufe auf dem Metacomputer. Es zeigt sich nämlich, dass trotz der Verwendung der doppelten Zahl an Prozessoren und damit der doppelten Zahl von Kommunikationen über die externen Netzwerkverbindungen zwischen den Systemen ein fast idealer Speedup zu beobachten ist. Und dies obwohl für diese Messungen keine dedizierte Netzwerkverbindung mehr zur Verfügung stand. Allerdings wurde diese Messung im Jahr 2001 und damit vier Jahre nach den vorhin vorgestellten durchgeführt und außerdem von den USA aus zu einer Uhrzeit, zu der in Europa, bedingt durch die fortgeschrittene Tageszeit, nicht mehr soviel Netzwerkverkehr zu beobachten ist.

Zum Abschluß zeigen wir in Tabelle 5.25 noch eine Messung auf insgesamt 320 Prozessoren, die mit einem 5,12 Mio. Zellennetz durchgeführt wurde. Gegenüber der Messung in der vorigen Tabelle wurde das Netz um den Faktor drei vergrößert,

System	Anzahl	GA	GL	Proces.	Gesamt
jaromir - hwwt3e - turing	107+107+106	234,8	232,9	713,7	765,8

Tabelle 5.25: Ergebnisse von Multiblock URANUS mit einem 5,12 Millionen Zellen Netz im Metacomputing

allerdings hat das Netz, bedingt durch die entsprechende Vergrößerung, nun eine etwas andere Konfiguration, denn es besteht aus 5 gleichgroßen Blöcken. Wenn man die Messungen vergleicht, fällt auf, dass der Aufwand für Pre-, Postprocessing und I/O nicht skaliert sondern aufgrund der gleichen Blockanzahl steigt, aber mit 52 Sekunden immer noch im erträglichen Rahmen bleibt. Vergleicht man die Zeit für den Berechnungsteil mit dem Ergebnis aus Tabelle 5.24 für 3x32 Prozessoren, stellt man einen Rechenzeitmehrbedarf von 116 Sekunden fest. Dieser liegt aber auch in der geänderten Konfiguration begründet. Da das Netz aber mit dem gleichen Faktor skaliert wurde wie die Anzahl der verwendeten Prozessoren, kann man trotz der geänderten Konfiguration Rückschlüsse auf die Skalierung ziehen. Der erreichte Scaleup ist mit 84% zwar nicht mehr sehr gut, aber doch noch im guten Mittelfeld.

5.8 Fazit

Die Anstrengungen, eine CFD-Applikation wie URANUS in einer Metacomputingumgebung effizient zum Laufen zu bringen haben auch positive Auswirkungen auf die weitere Entwicklung des Programms in der Multiblockversion gehabt. Aber auch die Entwickler der Kopplungssoftware [29] haben von der engen Zusammenarbeit profitiert und PACX-MPI an entscheidenden Stellen optimieren können. Selbstverständlich sind, mit Ausnahme vielleicht der trivialparallelen Applikationen ohne größere Kommunikation, alle Applikationen im Metacomputing auf eine effiziente Bibliothek für die Kopplung der Systeme angewiesen. Die Versuche haben aber auch gezeigt, dass für die Zukunft weitere Verbesserungen der sogenannten Middleware, vor allem im Managementbereich notwendig sind. Denn nur, wenn die notwendigen Partitionen auf allen benötigten Rechnern sowie die Netzwerkverbindungen einfach buchbar und verfügbar sind, werden sich solche gekoppelten Anwendungen in Zukunft sinnvoll einsetzen lassen.

Außerdem hat sich gezeigt, dass der Einsatz von Metacomputing für relativ eng gekoppelte Probleme wie die dargestellten CFD Rechnungen möglich ist. Sinnvoll ist der Einsatz der Supercomputerkopplung aber nur für ausgesuchte Fälle, die nicht mehr auf einem Rechner allein gelöst werden können, deren Lösung aber entweder unbedingt notwendig oder trotz des Aufwandes wirtschaftlich sinnvoll ist.

Kapitel 6

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das bestehende URANUS Programm zur Berechnung von Wiedereintrittsströmungen so erweitert, dass Multiblocknetze zur Berechnung verwendet und so Lösungen selbst für komplexe Topologien berechnet werden können. Dabei wurde die Programmierung so durchgeführt, dass zur Berechnung Supercomputer der höchsten Leistungsklasse effizient verwendet werden können.

Das erstellte Programm berücksichtigt die Eigenschaften der Multiblocknetze, die fast beliebig um das zu berechnende Wiedereintrittsfahrzeug angeordnet sein können. Dies umfaßt vor allem die Behandlung der physikalischen Ränder, die verschiedenen Schnittstellen der Blöcke untereinander, als auch die unterschiedlichen Größen der Blöcke, die vor allem bei einem Einsatz des Programms auf modernen Parallelrechnerarchitekturen die Verwendung eines Lastverteilers notwendig machen. Die Einführung der Multiblocknetze bedingte die Schaffung völlig neuer Datenstrukturen im Programm, die bereits zukünftige Erweiterungen wie die adaptive Verfeinerung einzelner Blöcke und Mehrgitterverfahren berücksichtigt, so dass diese ohne große Änderungen an der vorliegenden Daten- und Programmstruktur einfach integriert werden können. Die Programmstruktur wurde so verändert, dass nun ein Prozess Simulationsrechnungen auf mehreren Blöcken simultan durchführen kann. Hierzu werden den Prozessen Blocklisten übergeben, die diese dann abarbeiten. Die notwendige Kommunikation zwischen den Blöcken findet dabei ebenfalls unter der Verwendung von Blocklisten statt, so dass es für den einzelen Block unerheblich ist, ob seine Nachbarn auf demselben Prozess oder von einem anderen berechnet werden. Um eine gute Verteilung der Last auf die verfügbaren Prozessoren zu erreichen, wurden Tools und Algorithmen zur Gitterpartitionierung in das Programm integriert, die eine quasi optimale Verteilung der Blöcke auf die Prozesse berechnen, um so für das jeweilige Problem eine möglichst geringe Programmlaufzeit zu erreichen.

Die dargestellten Ergebnisse zeigen, dass URANUS nun in der Lage ist, auf modernen Architekturen mit hoher Leistung gute Simulationsergebnisse selbst für komplexe Toplogien zu liefern. Der relative Leistungsverlust durch die Paral-

lelisierung beträgt dabei je nach Problemgröße und Anzahl der verwendeten Prozessoren zwischen drei und 12%. Eine Skalierung des Programms bis auf mehrere tausend Prozessoren sollte daher möglich sein.

Das Programm ist flexibel einsetzbar, da es auf fast allen verfügbaren parallelen Plattformen lauffähig ist. Testrechnungen wurden sowohl auf Massivparallelen Systemen als auch auf Vektorrechnern und PC-Clustern durchgeführt.

Der Einsatz des Strömungsprogramms in einer Metacomputingumgebung hat gezeigt, dass auch CFD-Programme von einer solchen Umgebung profitieren können, obwohl sie durch die relativ starke Kopplung nicht von vorneherein für eine solche Umgebung prädestiniert erscheinen. Die vorgestellten Änderungen und Anpassungen mindern aber den Einfluß der schwachen externen Netzwerkverbindung und vor allem der hohen Latenzen, die hauptsächlich durch die räumliche Entfernung zwischen den Rechnersystemen entstehen. Testrechnungen auf einem weltweit verteilten Metacomputer haben gezeigt, dass das URANUS Programm auch mit mehr als tausend Prozessoren einsetzbar ist.

Trotzdem ist es stets sinnvoll, soweit nur irgend möglich, homogene Systeme einzusetzen, die über ein möglichst starkes Verbindungsnetzwerk verfügen, da sich nur so größere Leistungsverluste vermeiden lassen.

6.1 Ausblick

Während des Entwurfs und der Verwirklichung des Programms entstanden bereits viele Ideen, wie das Programm weiterentwickelt werden kann. Manche wurden schon in der Darstellung des Entwurfs erwähnt. Die wichtigsten sollen hier vorgestellt werden.

Adaptive Netzverfeinerung

Bei der adaptiven Netzverfeinerung werden Blöcke oder Teilblöcke nach Maßgabe des Strömungslösers weiter in der Auflösung verfeinert. Die dazu notwendigen Voraussetzungen sind bereits in der Datenstruktur des parallelen Multiblocklösers geschaffen. Grenzen Blöcke unterschiedlicher Auflösung aneinander, können die notwendigen Interpolationen während der Kommunikation durchgeführt werden, so dass von der Blockseite her keine Änderungen am Löser notwendig sind.

Alleinige Berechnung von Blöcken in denen Änderungen notwendig sind

Bei den Versuchen für das Metacomputing wurde auch untersucht, wie sich während der gesamten Rechenzeit die Informationen auf den einzelnen Blöcken ändern. Dabei wurde festgestellt, dass nur während etwa der Hälfte der Rechenzeit sich Änderungen auf allen Blöcken gleichzeitig ergeben. Daraus ergibt sich die Idee,

Blöcke, auf denen sich momentan nichts ändert, die also vorübergehend einen stabilen Zustand erreicht haben, für eine bestimmte Anzahl von Iterationen einfach abzuschalten. Im Extremfall kann man dies so lange tun, bis wieder Änderungen auf diesen Block propagiert werden. Während der Block abgeschaltet ist, verwenden die Nachbarn, sofern sie aktiv sind, einfach die letzten verfügbaren Werte des abgeschalteten Nachbarblocks, anstatt Daten mit diesem auszutauschen. Diese Werte, die der Nachbarblock schicken würde, ändern sich ja auch nicht. Ergeben sich Änderungen auf einem der Nachbarblöcke, die an den abgeschalteten Nachbarn propagiert werden müssen, so wird dieser wieder aktiviert. Hierbei könnte man auch das Abbruchkriterium als Grenzwert für zu propagierende Änderungen an den Rändern verwenden.

Das Verfahren dürfte nach der Beobachtung in etwa eine Rechenzeitverkürzung von 25% erlauben. Allerdings ist dazu auch eine dynamische Lastverteilung notwendig, damit Prozesse, die bisher nun deaktivierte Blöcke berechnet haben, andere Blöcke zur Berechnung zugewiesen bekommen. Die Zeit, die für die Lastverteilung, vor allem den notwendigen Transfer der Blöcke, benötigt wird, sollte natürlich möglichst geringer sein, als die Rechenzeitersparnis. Weiterhin muß der Rechner genügend Hauptspeicher besitzen, da die deaktivierten Blöcke auch noch Platz im Speicher beanspruchen, wenn dieser auch einiges geringer ist, als der für aktive Blöcke benötigte. Dies sollte aber gegeben sein, da ja auch alle Blöcke aktiv sein könnten. Es ist aber bei der Lastverteilung darauf zu achten, dass der Hauptspeicher auf den jeweiligen Prozessoren bzw. Knoten nicht überläuft, weil dort zu viele inaktive Blöcke abgelegt sind.

Message Pipelining für Netze mit geringer Leistungsfähigkeit

Wie wir in Kapitel 4 festgestellt haben, ist die relative Netzwerkbandbreite auf den neueren Systemen weitaus geringer als auf Maschinen der älteren Generationen. Da die Prozessorleistung nach wie vor weitaus schneller steigt als die der Netzwerke, wird sich dieser Trend weiter fortsetzen. Und die Tendenz, in Zukunft Mehrprozessorkerne einzusetzen, verschärft dieses Problem weiter. Da die Leistungsfähigkeit des parallelen Programms ja zu einem guten Teil von einer guten Netzwerkverbindung abhängt, bietet es sich auf Systemen mit einer besonders geringen relativen Netzwerkleistung an, die eigentlich für das Metacomputing entwickelten Technologien auch auf diesen meist clusterartigen Systemen einzusetzen. Vor allem das Message Pipelinening wird sich hier positiv auswirken. Allerdings muß vor dem Einsatz abgeschätzt werden, ob die Leistungsverbesserung den Nachteil im numerischen Verfahren mit dem etwa 30% höheren Aufwand bis zur gleichen Konvergenz wettmachen kann.

Lastverteilung für Metacomputing und Cluster

Für den Einsatz auf Systemen mit einer heterogenen Struktur ist eine weitere Verbesserung äußerst sinnvoll. Systeme mit heterogener Struktur können hierbei sowohl Metacomputer sein, bei denen die einzelnen Komponenten typischerweise unterschiedliche Architektur und damit Leistungsfähigkeit besitzen, als auch Clustersysteme, die nach und nach vergrößert oder nachgerüstet wurden. Auf solchen Systemen ist es von Vorteil, wenn die Lastverteilung die Heterogenität des Systems berücksichtigt. Dazu benötigt der Lastverteiler allerdings die Kenndaten der verschiedenen unterschiedlichen Systemkomponenten. Am einfachsten und sichersten ist es, sich diese Werte selbst zu besorgen, das heißt, mittels eines typischen Rechenabschnitts dessen Laufzeit und damit die Prozessorleistung zu bestimmen. Die Netzwerkgeschwindigkeit kann man entsprechend mit typischen Testnachrichten ausmessen. Anhand dieser Werte kann der Lastverteiler dann eine angepaßte Verteilung bestimmen. Da die Leistungsbestimmung nur zu Beginn der Berechnung notwendig ist, lohnt sich der geringe Aufwand dafür in nahezu allen Fällen.

Ansätze für eine zusätzliche Shared Memory Parallelisierung auf Threadebene

Die meisten modernen Rechner haben einen clusterartigen Aufbau mit verteiltem Hauptspeicher, auf denen allerdings die einzelnen Knoten aus Mehrprozessorsystemen mit gemeinsamem Hauptspeicher bestehen. Auf solchen Systemen besteht die Möglichkeit, die Parallelisierung für den Knoten mittels OpenMP vorzunehmen. Der Standardansatz für diese Art der Parallelisierung ist es, die äußersten Schleifen parallel auf den verschiedenen Prozessoren des Knotens abzuarbeiten, sofern dies die Datenabhängigkeiten erlauben. In URANUS wird dies für das Berechnen des zu lösenden Gleichungssystems relativ einfach funktionieren, da in diesen Schleifen kaum Datenabhängigkeiten zwischen den einzelnen Schleifendurchläufen bestehen. Problematisch wird das ganze dagegen im Löser, da hier bei der Lösung des Heptadiagonalsystems, bzw. bei der Lösung der jeweiligen Tridiagonalsysteme durch das Thomasverfahren Abhängigkeiten zwischen den einzelnen Schleifendurchläufen bestehen.

Das blockstrukturierte Verfahren im Multiblock URANUS erlaubt aber auch eine andere Vorgehensweise. Die Idee ist, hier auch auf Blockbasis zu parallelisieren und den einzelnen Threads dann die Blöcke aus der Blockliste des Prozesses zuzuordnen. Hier muß dann nur sichergestellt werden, dass der Lastverteiler dem Prozess genügend Blöcke zuteilt. Dies kann man aber relativ einfach sicherstellen. Die Kommunikation wird nur vom Masterthread durchgeführt, was sich ohne Probleme mit fast allen MPI-Bibiotheken realisieren läßt. Der knoteninterne Datenaustausch erfolgt durch einfaches Kopieren der Werte an den Blockrändern oder ebenfalls über MPI. Dieser Ansatz bietet den Vorteil einer hohen Daten-

lokalität und ist relativ einfach zu realisieren. Wie bereits in anderen Projekten nachgewiesen wurde, ist durch die Datenlokalität und die Weiterverwendung des Parallelisierungsansatzes für verteilten Speicher mit einer guten Performance zu rechnen.

Dieses Vorgehen hat gegenüber der reinen Schleifenparallelisierung einen Leistungsvorteil. Ist allerdings die MPI-Bibliothek gut an die Shared Memory Systeme angepaßt, sollte der Leistungsgewinn nicht allzu groß sein, da auch hier die MPI-Kommunikation quasi einem direkten Kopieren gleichkommt.

Literaturverzeichnis

[1] T. Beisel, E. Gabriel, M.Resch: *An Extension to MPI for Distributed Computing on MPPs'*, in M. Bubak, J. Dongarra, J. Wasniewski (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', Lecture Notes in Computer Science, Springer, 1997, pp. 75-83.

[2] H.M. Bleecke et al: *Benchmarks and Large Scale Examples*, in A. Schüller 'Portable Parallelization of Industrial Aerodynamic Applications (POPINDA), Notes on Numerical Fluid Mechanics, Volume 71, Vieweg, 1999.

[3] T. Bönisch, A. Geiger: *Implementierung eines 3D Strömungscodes auf Parallelrechnern*, RUS - 35, Forschungs- und Entwicklungsberichte, Rechenzentrum der Universität Stuttgart, 1996, ISSN 0941-4665.

[4] T. P. Bönisch, R. Rühle: *Portable Parallelization of a 3-D Flow-Solver*, in Emerson et. al. (Eds.) 'Parallel Computational Fluid Dynamics', Recent Developments and Advances Using Parallel Computers, North-Holland, 1998, pp. 457-464.

[5] T. P. Bönisch, Roland Rühle: *Adaptation of a 3-D Flow-Solver for use in a Metacomputing Environment*, in C.A. Lin, A. Ecer, N. Satofuka, P. Fox, J. Periaux (Eds.), 'Parallel Computational Fluid Dynamics' Development and Applications of Parallel Technology, North Holland, 1999, pp. 119-125.

[6] T. P. Bönisch, R. Rühle: *Efficient Flow Simulation with Structured Multiblock Meshes on Current Supercomputers*, in ERCOFTAC Bulletin No. 50: Parallel Computing in CFD, 2001.

[7] T. P. Bönisch, R. Rühle: *Migrating from a Parallel Single Block to a Parallel Multiblock Flow Solver*, in P. Wilders, A. Ecer, J. Periaux, N. Satofuka, P. Fox (Eds.), 'Parallel Computational Fluid Dynamics' Practice and Theory, North Holland, 2002.

[8] T. P. Bönisch, R. Rühle: *Implementation of an Integrated Efficient Parallel Multiblock Flow Solver*, in G.R. Joubert, A.Murli, F.J. Peters, M. Vanneschi (Eds.), 'Parallel Computing' Advances and Current Issues, Imperial College Press, London, 2002, pp. 58-65.

[9] T. Bönisch, J.D. Chen, A.Ecer, Y.P. Chien, H.U. Akay: *Dynamic Load Balancing in International Distributed Heterogeneous Workstation Clusters*, in C. B. Jenssen, T. Kvamsdal, H. I. Andersson, B. Pettersen, A. Ecer, J. Periaux, N. Satofuka, P. Fox (Eds.),'Parallel Computational Fluid Dynamics' Trends and Applications, North Holland, 2001.

[10] Bönisch T., Resch M., Berger H.: *Benchmark Evaluation of the Message-Passing Overhead on Modern Parallel Architectures*, Proceedings of ParCo97 at Bonn/Germany.

[11] Folkmar A. Bornemann and Peter Deuflhard: *The cascadic multigrid method for elliptic problems* Numerische Mathmatik, Volume 75, No. 2, pp. 135-152, 1996.

[12] A. Brandt: *Multi-Level Adaptive Solutions to Boundary-Value Problems* Mathematics of Computation, 21(138), pp. 333-390, April 1977.

[13] Thomas Bräunl: *Parallele Programmierung*, Vieweg, Braunschweig, 1993.

[14] Brunst, H., Winkler, M., Nagel, W.E., Hoppe, H.-C.: *Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach.* In: Alexandrov, V.N., Dongarra, J.J., Juliano, B.A., Renner, R.S., Kenneth Tan, C.J. (eds.): Computational Science - ICCS 2001, Part II, Vol. 2074 in LNCS, pp. 751-760, Springer, 2001.

[15] R.B. Ciotti, J.R. Taft, J.Petersohn: *Early Experiences with the 512 Processor Single System Image Origin2000*, Proceedings of the 42nd CUG Conference (CUG Summit 2000), 2000 (CDROM).

[16] Daiss, A.: *Modellierung katalytischer Reaktionen auf Siliziumdioxidoberflächen unter Berücksichtigung von Verdünnungseffekten* Dissertation, Fakultät für Luft- und Raumfahrttechnik, Universität Stuttgart, 1997.

[17] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon: *Top 500 Supercomputer Sites* www.top500.org

[18] A.Ecer, Y.P. Chien, J.D. Chen, T. Bönisch, H.U. Akay: *Dynamic Load-Balancing for Distributed Heterogeneous Computing of Parallel CFD Problems*, Proceedings of NASA High Performance Computing and Communications Computational Aero Science Workshop (CD-ROM), February 15-17 2000, California, USA.

[19] The Earth Simulator Center: *System Configuration of the Earth Simulator* available at http://www.es.jamstec.go.jp/esc/eng/Hardware/system.html

[20] Luciano Fornasier: *HISSS A High-Order Panel Method for Subsonic and Supersonic Attached Flow about Arbitrary Configurations* in: Panel Methods in Fluid Mechanics with Emphasis on Aerodynamics, Notes on Numerical Fluid Mechanics, Vol. 21, Vieweg, Braunschweig, Germany, 1987.

[21] M. Frieß, T. Ullmann: *Materialentwicklung und Herstelltechnologie für C/C-SiC-Bauteile* in Sonderforschungsbereich (SFB) 259, Hochtemperaturprobleme rückkehrfähiger Raumtransportsysteme, Arbeits- und Ergebnisbericht 2001.

[22] H.-H. Frühauf, M. Fertig, F. Olawsky, and T. Bönisch: *Upwind Relaxation Algorithm for Re-entry Nonequilibrium Flows* in E. Krause, W. Jäger 'High Performance Computing in Science and Engineering '99 ', Springer, Berlin 2000, pp. 365-378.

[23] Frühauf, Daiss, Gerlinger, Knab, Schöll: *Computation of Reentry Nonequilibrium Flows in a Wide Altitude and Velocity Regime* American Institute of Aeronautics and Astronautics (AIAA) Paper 94-1961, 1994.

[24] H.-H. Frühauf, T. Bönisch, A. Daiss, M. Fertig, A. Geiger, U. Gerlinger, S. Kanne, F. Olawsky, M. Resch, S. Rips: *Numerische Wiedereintritts-Aerothermodynamik* Arbeits- und Ergebnisbericht, Sonderforschungsbereich 259, Teilprojekt C3, S. 313-341, 1998.

[25] H.-H. Frühauf, M. Fertig, S. Kanne: *Validation of the Enhanced URANUS Non-Equilibrium Navier-Stokes Code* in AIAA Journal of Spacecrafts and Rockets, Vol. 37, No. 2, March-April 2000.

[26] H.-H. Frühauf, M. Resch, T. Bönisch, M. Fertig, F. Infed, F. Olawsky: *Numerische Wiedereintritts-Aerothermodynamik* Arbeits- und Ergebnisbericht, Sonderforschungsbereich 259, Teilprojekt C3, S. 355-390, 2001.

[27] H.-H. Frühauf, O. Knab, A. Daiss, U. Gerlinger: *The URANUS code, an advanced simulation tool for re-entry nonequilibrum flow simulations*, Journal of Flight Sciences and Space Research, 19, 219-227, 1995.

[28] E. Gabriel, M.Resch, R.Rhle: *Implementing MPI with Optimized Algorithms for Metacomputing* in Anthony Skjellum, Purushotham V. Bangalore, Yoginder S. Dandass, 'Proceedings of the Third MPI Developer's and Users's Conference', MPI Software Technology Press, Starkville Mississippi, 1999.

[29] E. Gabriel: *Optimierung und Einsatz einer Kommunikationsbibliothek für Metacomputing.* Forschungs- und Entwicklungsberichte, RUS-54, ISSN 0941-4665, Mai 2002.

[30] A. Geiger: *Parallelrechner, Architectur und Programmierung.* Forschungs- und Entwicklungsberichte, RUS-19, ISSN 0941-4665, März 1995.

[31] U. Gerlinger, H.-H. Frühauf, T. Bönisch: *Implicit Upwind Navier-Stokes Solver for Re-entry Nonequilibrium Flows* American Institut of Aeronautics and Astronautics Paper 97-2547, June 1997.

[32] Gerlinger, P. and Brüggemann, D.: *Multigrid Convergence Acceleration for Turbulent Supersonic Flows* International Journal of Numerical Methods in Fluids, Vol. 24, pp. 1019-1035, 1997.

[33] Gerlinger, P., Möbus, H., Brüggemann, D.: *An Implicit Multigrid Method for Turbulent Combustion* Journal of Computational Physics, 167, pp. 247-276, 2001.

[34] E. A. Gerteisen and A. Jameson: *Massive Parallel Implementation of the Aircraft Euler Method and Performance Tests on Different Computational Platforms* GAMNI/SMAI Conference on Parallel CFD93, Paris/France, May 10-12, 1993.

[35] Gräßlin, M. Burkhradt, J., Schöttle, U.M.: *Optimierung und Sensitivitätsanalysevon Rückkehrbahnen des Raumgleiters X-38*, DGLR-JT99-1000, Deutscher Luft- und Raumfahrtkongreš 1999, Berlin.

[36] Gittergenerator *Gridpro*, weitere Informationen im Internet unter http://www.gridpro.com

[37] W. Gropp, E. Lusk, N. Doss, A. Skjellum: *A high-performance portable imple-mentation of the MPI message-passing interface standard*, Parallel Computing, 22, 789-828, 1996.

[38] Gropp, Lusk, Skjellum: *Using MPI, Portable Programming with the Message-Passing Interface*, The MIT Press, Cambridge Massachusetts, USA, 1994.

[39] L. Hart and S. McCormick: *Asynchronous Multilevel Adaptive Methods for Solving Partial Differential Equations on Multiprocessors: Basic ideas* Parallel Computing, 12, pp. 131-144, 1989.

[40] Stefan Hein, Ingo Rosenboom, Gero Günter, Volker Hannemann: *Erste Untersuchungen zur Anwendbarkeit des DLR-TAU-Codes zur Grundströmungsberechnung für nachfolgende Instabilitätsanalysen mit NOLOT/PSE* DLR-IB 223-99 A30, Göttingen 2000.

[41] C. Helf, U. Küster, R. Rühle: *Automatic, Self-Adaptive Grid-Generation for Complex Geometries in 3D*, in C.A. Lin, A. Ecer, N. Satofuka, P. Fox, J. Periaux (Eds.), 'Parallel Computational Fluid Dynamics' Development and Applications of Parallel Technology, North Holland, 1999.

[42] High Performance Fortran Forum: *High Performance Fortran Language Specification* Scientific Programming, Vol. 2, no. 1-2 (Spring and Summer 1993), pp. 1-170, John Wiley and Sons, 1993.

[43] Ch. Hirsch: *Numerical Computation of Internal and External Flows* Volume 1: Fundamentals of Numerical Discretization, John Wiley & Sons, Chichester, 1988.

[44] Hirsch K, Altmann I, Bauer G, Kollermann T, Roth B, Schneider J, Schumacher U, Stirn R, Tawfik A: *Development and Investigation of Erosion Protection Layers for Heat Shilding of Re-Entry Space Vehicles* VII European Conference on Atomic and Molecular Physics, April, 2.-6. 2001, Berlin, Germany.

[45] J. Hofhaus: *Entwicklung einer effizienten, parallelisierten und impliziten Lösung der Navier-Stokes-Gleichungen für dreidimensionale, inkompressible und instationäre Strömungen* Diplomarbeit an der RWTH Aachen, Fakultät für Maschinenwesen, 1993.

[46] John E. Hopcroft, Jeffrey D. Ullman: *Introduction to automata theory, languages, and computation* - Reading, Mass. : Addison-Wesley, 1979, ISBN 0-201-02988-X.

[47] F. Infed, F. Olawsky and M. Auweter-Kurtz: *Stationary Coupling of 3D Hypersonic Nonequilibrium Flows and TPS Structure with URANUS* AIAA-2003-3984, 16th AIAA Computational Fluid Dynamics Conference, Orlando, Florida, June 23-26, 2003.

[48] Jonas, S.: *Implizites Godunov-Typ-Verfahren zur voll gekoppelten Berechnung reibungsfreier Hyperschallströmungen im thermo-chemischen Nichtgleichgewicht* Dissertation, Fakultät für Luft- und Raumfahrttechnik, Universität Stuttgart, 1992.

[49] Kanne, S.: *Zur thermo-chemischen Relaxation innerer Freiheitsgrade durch Stoß- und Strahlungsprozesse beim Wiedereintritt* Dissertation, Fakultät für Luft- und Raumfahrttechnik, Universität Stuttgart, 1999.

[50] G. Karypis and V. Kumar: *A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering* Journal of Parallel and Distributed Computing, Vol. 48, pp. 71-85, 1998.

[51] J. De Keyser, K. Lust, and D. Roose: *Run-time load balancing support for a parallel multiblock Euler/Navier-Stokes code with adaptive refinement on distributed memory computers* Parallel Computing, 20:1069-1088, 1994.

[52] Knab, O.: *Konsistente Mehrtemperatur-Modellierung von thermochemischen Relaxationsprozessen in Hyperschallströmungen* Dissertation, Fakultät für Luft- und Raumfahrttechnik, Universität Stuttgart, 1996.

[53] N. Kroll, B. Eisfeld, H.M. Bleecke: *FLOWer* in A. Schüller 'Portable Parallelization of Industrial Aerodynamic Applications (POPINDA)', Notes on Numerical Fluid Mechanics, Volume 71, Vieweg, 1999

[54] U. Lang, J. P. Peltier, P. Christ, S. Rill, D. Rantzau, H. Nebel, A. Wierse, R. Lang, S. Causse, F. Juaneda, M. Grave, P. Haas: *Perspectives of collaborative supercomputing and networking in European aerospace research and industry*, Journal of Future Generation Computing Systems, 11, 419-430, 1995.

[55] M.A. Leschziner, F.S. Lien: *Computation of Physically Complex Turbulent Flows on Parallel Computers with a Multiblock Algorithm* in Emerson et. al. (Eds.) 'Parallel Computational Fluid Dynamics' Recent Developments and Advances Using Parallel Computers, North-Holland, 1998, pp. 3-14.

[56] F.S. Lien, W.L. Chen and M.A. Leschziner: *A Multiblock Implementation of a Non-Orthogonal, Collocated Finite Volume Algorithm for Complex Turbulent Flows* International Journal for Numerical Methods in Fluids, Vol. 23, pp. 567-588, 1996.

[57] S. McCormick and J. Thomas: *The FAst Adaptive Composite Grid (FAC) Method for Elliptic Equations* Mathematics of Computation, 46, pp. 439-456, 1986.

[58] Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard.* International Journal of Supercomputer Applications, 8, 159-416, 1994.

[59] Message Passing Interface Forum: *MPI-2: Extensions to the Message Passing Interface Standard.* Document for a Standard Message-Passing Interface, University of Tennessee, 1997.

[60] Message Passing Interface Forum: *MPI-2 Journal of Development* Supplement to the MPI-2 standard document, University of Tennessee, 1997.

[61] OpenMP Forum: *OpenMP Fortran Application Program Interface* Version 2.0, November 2000, Accessible via http://www.openmp.org/specs/

[62] Olawsky, Ferdinand: *Effizientes Newton-Verfahren zur Berechnung stationärer Hyperschallströmungen im thermochemischen Nichtgleichgewicht* Dissertation an der Fakultät für Luft- und Raumfahrttechnik der Universität Stuttgart, 2005.

[63] Radespiel, R. and Swanson, R. C.: *Progress with Multigrid Schemes for Hypersonic Flow Problems* Journal of Computational Physics, Vol. 116, pp. 103-112, 1995.

[64] Radespiel R, Longo JMA, Brück S: *Efficient numerical simulation of complex 3D flows with large contrast* in: 77th AGARD Fluid Dynamics Panel Meeting and Symposium on Progress and Changes in CFD Methods and Algorithms, Sevilla, October 1995, AGARD-CP-578, pp. 33-1 – 33-11.

[65] M. Resch : *Metacomputing für Simulationsanwendungen* Forschungs- und Entwicklungsberichte, RUS-51, Juni 2001.

[66] M. Resch, D. Rantzau, H. Berger, K. Bidmon, R. Keller, E. Gabriel: *A Metacomputing Environment for Computational Fluid Dynamics* in C.A. Lin, A. Ecer, N. Satofuka, P. Fox, J. Periaux (Eds.), 'Parallel Computational Fluid Dynamics' Development and Applications of Parallel Technology, North Holland, 1999, pp. 135-142.

[67] M. Resch, H. Berger and T. Bönisch: *A Comparison of MPI Performance on Different MPP's* in Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', pp. 25-32, Springer, 1997.

[68] M. Resch, T. Beisel, T. Bönisch, B. Loftis, R. Reddy: *Performance Issues of Intercontinental Computing* Cray User Group Meeting, San Jose (CA), May 1997.

[69] Michael Resch, Hans Babowsky: *Workstation Clustering: A powerful tool for numerical simulation in flight sciences and space research* Journal of Flight Sciences and Space Research, 19, 253-258, 1995.

[70] Michael Resch, Alfred Geiger, Jörg Zikeli: *Message-Passing-Systems on Workstation Clusters and Parallel Computers, The Impact of Software- and Network-Architectures on Applications* in: Gentzsch W. and Harms U. (Eds.), High Performance Computing and Networking, Vol. 2, Springer, Berlin Heidelberg, 260-266, 1994.

[71] Michael M. Resch, Alfred Geiger, Manuela Sang: *Developing PVM-Code on Various Hardware Platforms: Portability and Performance* First European PVM User Group Meeting, Rome/Italy, October 10-11, 1994.

[72] H. Ritzdorf: *The High-Level Communications Library CLIC* in A. Schüller 'Portable Parallelization of Industrial Aerodynamic Applications (POPINDA), Notes on Numerical Fluid Mechanics, Volume 71, pp. 33-50, Vieweg, 1999

[73] E. Schöll, H.-H. Frühauf: *An Accurate Efficient Implicit Upwind Solver for the Navier-Stokes Equations* in F.-K. Hebecker, R. Ranacher, G. Wittum (Eds.) Notes on Numerical Fluid Mechanics, Numerical Methods for the Navier-Stokes Equations, Vieweg, 1994.

[74] E. Schöll: *Ein dreidimensionales Rechenverfahren für reibungsbehaftete Strömungen in den Schaufelreihen von Turbomaschinen* VDI Forschrittsbericht Reihe 7, Nummer 287, VDI Verlag, 1996.

[75] D. Sharov, H. Luo, J.D. Baum and R. Löhner: *Unstructured Navier-Stokes Grid Generation at Corners and Ridges* Int. J. Num. Meth. Fluids 43, 717-728 (2003).

[76] Sheffer, S. G., Jameson, A. and Martinelli, L.: *A Multigrid Method for High Speed Reactive Flows* AIAA-papre 97-2106, 1997.

[77] L. Smarr and C.E. Catlett: *Metacomputing* Communications of the ACM 35/6, 45-52, 1992.

[78] Sonderforschngsbereich 259: *Hochtemperaturprobleme rückkehrfähiger Raumtransportsysteme* Arbeits- und Ergebnisbericht, 2001.

[79] Steger, J. L., Dougherty, F. C. and Benek, J. A.: *A Chimera Grid Scheme* Advances in Grid Generation, K. N. Ghia and U. Ghia, eds., ASME FED-Vol. 5, June, 1983.

[80] Jules Verne: *Von der Erde zum Mond* erschienen 1865.

[81] Jules Verne: *Reise um den Mond* erschienen 1879.

[82] C. Walshaw, M. Cross and M. Everett: *Parallel Dynamic Graph Partitioning for Adaptive Unstructered Meshes* Journal of Parallel and Distributed Computing 47(2), pp. 102-108, 1997.

[83] Weimar, L.: *Erweiterung des parallelisierten Uranus Navier-Stokes-Rechenverfahrens auf blocktrukturierte Netze.* Studienarbeit. Institut für Raumfahrtsysteme, Universität Stuttgart, Januar 1998.

[84] Andreas Wierse: *Performance of the COVISE visualization system under different conditions*, in: Georges G. Grinstein, Robert F. Erbacher (Eds.), Proceedings of SPIE 2410, San Jose/California, 218-229, 1995.