

FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Performance-Analyse paralleler Programme:
Die PARvis-Visualisierungsumgebung

*Alfred Arnold, Jost Bernert,
Wolfgang E. Nagel, Martin Röth*

KFA-ZAM-IB-9530

Dezember 1995
(Stand 14.12.95)

Erschienen in: PARS-Mitteilungen, 14. PARS-Workshop 1995, Stuttgart

Performance-Analyse paralleler Programme - Die *PARvis*-Visualisierungsgebung -

A. Arnold, J. Bernert, W. E. Nagel, M. Röth
(a.arnold, j.bernert, w.nagel, m.roeth)@kfa-juelich.de
Zentralinstitut für Angewandte Mathematik (ZAM)
Forschungszentrum Jülich GmbH (KFA)
D-52428 Jülich

Kurzfassung

PARvis ist eine Visualisierungsgebung, die eine gegebene Trace-Datei in eine Reihe verschiedener graphischer Darstellungen, z.B. Momentaufnahmen, Statistiken oder auch Zeitachsendarstellungen, transferiert. Dies erleichtert die Programmoptimierung, wodurch der Entwicklungszyklus auf massiv-parallelen Rechnersystemen deutlich verkürzt wird. PARvis unterstützt die gängigen Programmiermodelle (physikalisch/virtuell gemeinsamer Speicher, Message-Passing) und ist auf einer breiten Palette von Workstations ablauffähig.

1 Einführung

Bei der Programmierung massiv-paralleler Computer zeigt sich immer wieder, daß mit realen Codes nur ein Bruchteil der vom Hersteller angegebenen Peak-Performance erreicht werden kann. Grund für diese Differenzen ist die im Vergleich zu „klassischen“ Parallelrechnern mit physikalisch gemeinsamem Speicher deutlich komplexere Struktur dieser Systeme, die eine Optimierung der Anwendungen unter genauer Kenntnis der unterliegenden Maschinenarchitektur erfordert. Selbst wenn diese Kenntnisse vorhanden sind, ist es aber auch für einen erfahrenen Programmierer immer noch schwierig, die Auswirkungen von Programmänderungen auf den Programmablauf abzuschätzen bzw. nur anhand der Ausführungszeiten von Programmteilen Problemstellen zu identifizieren. Profiling-Werkzeuge, wie sie von sequentieller Programmierung her bekannt sind und auch auf Parallelrechnern angeboten werden, geben mit ihren summarischen Angaben nur eine begrenzte Hilfestellung. Dies liegt daran, daß insbesondere für das auf massiv-parallelen Systemen weit verbreitete Programmiermodell des Message-Passing die zeitlichen Verhältnisse der einzelnen Instruktionsströme das Programmverhalten entscheidend beeinflussen. Dies äußert sich in der Praxis darin, daß ein paralleles Programm mit z.B. fehlerhafter Synchronisation trotz gleicher Prozessorzahl und gleichen Eingabedaten bei einem Lauf korrekte Ergebnisse liefert, bei einem anderen jedoch nicht: Bereits die Lastdifferenzen, die auf im Multiprogramming laufenden Parallelrechnern auftreten, beeinflussen das Programm in einer weder vorhersagbaren noch reproduzierbaren Weise. Dies führt einerseits dazu, daß die bei sequentieller Programmierung gern genutzte Fehlersuchmethode des sukzessiven Einkreisens meist nicht zum Ziel führt, andererseits aber auch das klassische Debugging von Programmen durch Anhalten und schrittweise Ausführung nur anwendbar ist, wenn der Fehler nicht in der Kommunikation der Prozessoren untereinander zu suchen ist. Das folgende Beispiel zeigt, wie das Anhalten eines Prozessors ein Programmverhalten ins Gegenteil verkehren kann: Zwei Prozessoren führen eine parallele Anweisungssequenz (z.B. eine Schleife)

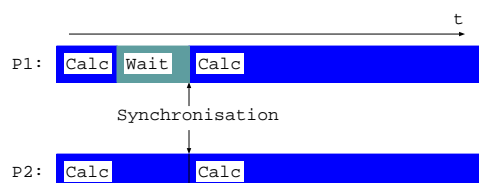


Abbildung 1: Programmablauf ohne Debugger

aus, aufgrund von Lastinbalancen ist aber der erste Prozessor früher fertig und muß an einer Synchronisationsstelle auf den zweiten warten, ehe er mit dem nächsten Konstrukt fortfahren kann (Abbildung 1). Im

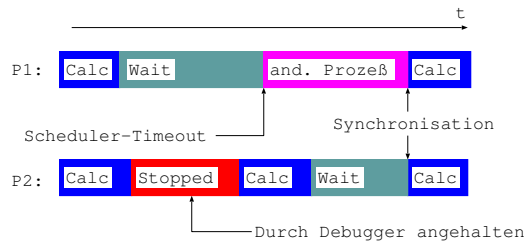


Abbildung 2: Programmablauf mit Debugger

zweiten Fall wurde zu Untersuchungszwecken der zweite Prozessor durch einen Debugger angehalten (Abbildung 2). Die Folge davon ist, daß P1 nach einer gewissen Wartezeit der Anwendung vom Betriebssystem entzogen wird und nun Teile eines anderen Programmes ausführt. Wird das unterbrochene Programm auf P2 fortgesetzt, so muß jetzt P2 auf P1 warten, da die Kontext-Rückschaltung auf P1 nicht sofort geschehen kann: Die Verhältnisse haben sich umgekehrt, P2 wartet jetzt auf P1.

Parallele Debugger, wie sie von den Herstellern massiv-paralleler Systeme angeboten werden (z.B. *ipd* auf der Intel Paragon[6]) können Effekte wie den eben gezeigten zwar abmildern, indem alle Prozessoren einer parallelen Anwendung gleichzeitig angehalten werden. Man muß sich allerdings vor Augen halten, daß diese Gleichzeitigkeit auch nur im Rahmen der Netzwerklatenzen gegeben ist und daß der gleichzeitige Wiederanlauf eine implizite Synchronisationsstelle in das Programm einfügt, den Programmablauf also auch verändern kann.

Zusammenfassend ergibt sich, daß die von sequentieller Programmierung her bekannten Methoden des Debuggings und der Performance-Analyse nur begrenzt auf parallele Programme übertragbar sind. Es werden Methoden gebraucht, die zum einen den Ablauf des Programmes möglichst wenig beeinflussen, andererseits aber auch einen detaillierten Einblick in die dynamischen Vorgänge zur Laufzeit gewähren. Die erste Forderung läßt sich durch *offline*-Verfahren erreichen, die während des Programmablaufes vorher festgelegte Komponenten des Programmverhaltens protokollieren und für eine spätere Auswertung in einer Datei speichern. Zu diesen Methoden zählt einerseits das bereits angesprochene Profiling, das zur Laufzeit Ausführungszeiten aufsummiert und diese für eine spätere statistische Auswertung zur Verfügung stellt, andererseits das im folgenden besprochene *Tracing*: Tracing basiert darauf, daß das zu untersuchende Pro-

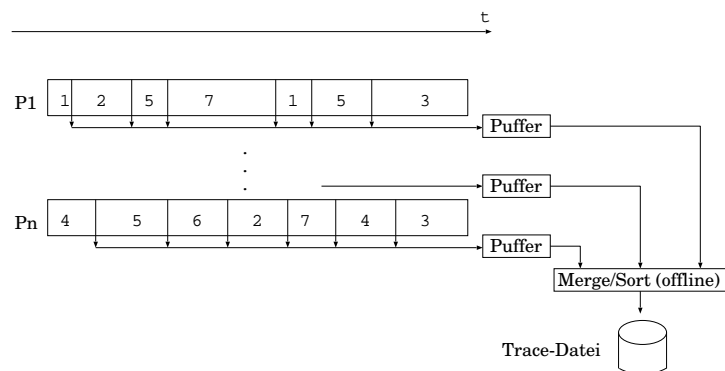


Abbildung 3: Prinzipielle Arbeitsweise des Tracing

gramm vor der Ausführung *instrumentiert* wird. Bei dieser Instrumentierung werden vom Benutzer vorgegebene Programmteile so erweitert, daß sie bei Ausführung Datensätze in ein Protokoll schreiben (Abb. 3). Diese Datensätze enthalten einen Zeitstempel, eine Kennzeichnung, die Rückschlüsse auf den Ereignistyp zuläßt, sowie meist noch eine vom Typ abhängige Liste zusätzlicher Parameter. Instrumentierbare Ereignisse sind z.B. Unterprogramm- und -ausprünge sowie das Versenden bzw. Empfangen von Nachrichten. Die Beeinflussung des zu analysierenden Programmes wird im allgemeinen dadurch reduziert, daß die Ereignis-Records in einen Puffer im lokalen Speicher des jeweiligen Prozessors geschrieben werden, der erst bei Überlauf auf Platte geleert wird. Nach dem Ende des Programmes werden die einzelnen Ereignisspuren chronologisch zu einer Datei zusammensortiert und können dann off-line analysiert werden.

Problematisch ist jedoch die auf diese Weise entstehende Datenmenge. Da bei der ersten Analyse eines Programmes meist noch nicht genau bekannt ist, wo die kritischen Stellen im Programm liegen, wird man in einem ersten Ansatz das komplette Programm mit allen möglichen Events instrumentieren. Die dabei entstehenden Trace-Daten können leicht die Größenordnung von vielen Megabyte pro Prozessor erreichen. Was benötigt wird, sind leistungsfähige Werkzeuge, die einerseits einen schnellen Überblick über die Datenflut gewähren, andererseits aber auch detaillierte Analysen bis auf die Ebene einzelner Ereignisse erlauben.

2 Die PARvis-Visualisierungsumgebung

PARvis wurde am Forschungszentrum Jülich mit der Zielsetzung entwickelt, dem Anwender ein leistungsfähiges, aber trotzdem ohne große Vorkenntnisse nutzbares Werkzeug zur Analyse seiner parallelen Anwendung zur Verfügung zu stellen. Zu diesem Zweck liest PARvis die vom Tracing-System gelieferten Trace-Dateien ein und transformiert sie in eine Reihe graphischer Darstellungen. Durch die X11/Motif-Basierung und Implementierung in Ansi-C ist PARvis auf praktisch jeder UNIX-Workstation einsetzbar. Implementationen existieren bisher für UNIX-Workstations der Firmen Sun (SunOS oder Solaris), DEC (Ultrix oder OSF/1), IBM sowie Silicon Graphics; Versionen für Linux bzw. HP-UX sind in Vorbereitung. Es existiert ein quellcodebasiertes Fortran-77-Instrumentierungswerkzeug *PARvis.inst*, das für PARvis verwendbare Traces erzeugt und für beliebige Message-Passing-Bibliotheken anpaßbar ist[5]. Konkrete Anpassungen existieren z. Zt. für Intel Paragon und Cray T3D. Weiterhin können über einen Konverter die vom Paragon-eigenen Instrumentierer gelieferten Tracefiles visualisiert werden.

Bei der Entwicklung von PARvis wurde Wert auf eine einheitliche und leicht zu handhabende Oberfläche gelegt, die es ermöglicht, ohne eine lange Einarbeitungszeit erste Ergebnisse zu erzielen: Alle Konfigurationsparameter lassen sich interaktiv einstellen, und alle Anzeigen mit ähnlicher Funktion werden auf gleiche Weise bedient. Im einzelnen gliedern sich die Anzeigen von PARvis in folgende Gruppen:

- Momentaufnahmen: Diese zeigen einen Teil des Systemzustandes zu einem bestimmten Zeitpunkt. Um den Systemzustand zu einem anderen Zeitpunkt zu sehen, kann man sich in der Zeitachse vor und zurück bewegen.
- Animationen: Die regelmäßige Bewegung in der Zeitachse im Zusammenspiel mit Momentaufnahmen ergibt einen animierten Eindruck von den dynamischen Veränderungen des Systemzustandes.
- Statistiken: Die Akkumulation von Zeiten oder Aufrufen über einen wählbaren Zeitraum erlaubt eine schnelle Übersicht über die Zeitanteile einzelner Unterprogramme an der Gesamtlaufzeit. PARvis schließt auf diese Weise Funktionen eines Profilers ein.
- Zeitachsendarstellungen: Diese stellen eine Komponente des Systemzustandes über der Zeitachse dar.

Insbesondere die letzte Klasse von Anzeigen stellt mit den gebotenen Zooming-Möglichkeiten den entscheidenden Vorteil gegenüber anderen Visualisierungswerkzeugen wie *Pablo*[7] oder *ParaGraph*[6] dar, die sich als reine Animationstools verstehen und es vergleichsweise schwierig machen, von der globalen Übersicht über das ganze Programm zur Detailanalyse zu gelangen: Durch einfaches Aufzoomen des gewünschten Zeitabschnittes ist diese in PARvis „on the fly“ möglich.

Die mit PARvis visualisierbaren Systemzustände können grob in drei Gruppen eingeordnet werden:

- Prozessorzustände
- Nachrichtenverkehr
- Seitenverteilung bei virtuell gemeinsamem Speicher

Auf diese drei Gruppen wird in den folgenden Abschnitten genauer eingegangen. Naturgemäß werden in der Praxis, je nach unterliegendem Programmiermodell, nicht alle Komponenten genutzt. Sofern PARvis am Trace-File erkennen kann, daß eine dieser Komponenten nicht auftaucht, werden die entsprechenden Menüpunkte automatisch nicht mehr angeboten.

3 Visualisierung von Prozessorzuständen

Da die Prozessoren eines Parallelrechners mit ihren momentanen Zuständen die wichtigste, zu visualisierende Komponente darstellen, dient ein großer Teil der Displays in PARvis ihrer Visualisierung. Bei der Visualisierung von Programmen, die auf einer Shared-Memory-Architektur ablaufen, sind dies sogar die einzig notwendigen Displays.

In PARvis werden Prozessorzustände durch die Kombination eines Zustandsnamens und einer optionalen zusätzlichen Nummer dargestellt. Während der Name eine grobe Klassifizierung des Zustandes erlaubt (z.B. „rechnend/wartend/blockiert“), ist mit der Nummer eine Feinunterscheidung möglich (z.B. die Angabe des momentan ausgeführten Unterprogrammes). Diese Information wird in dem einfachsten aller Zustands-Displays, der globalen Betrachtung aller Prozessoren, dargestellt (Abb. 4). Jeder Prozessor wird dabei als

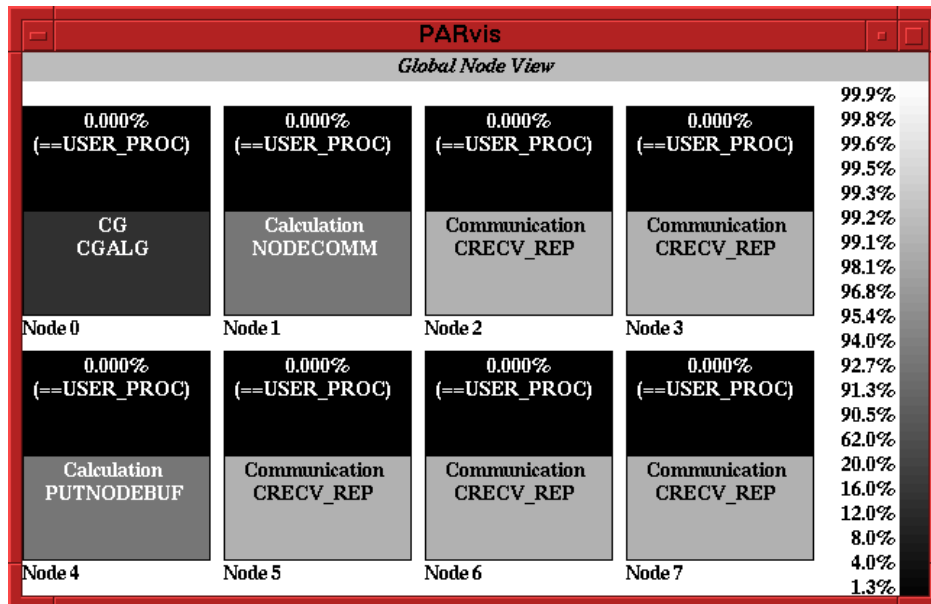


Abbildung 4: Darstellung von Prozessorzuständen

ein Kästchen dargestellt, in dessen unterer Hälfte die momentane Aktivität zu sehen ist. In diesem Beispiel wurde von der Möglichkeit Gebrauch gemacht, einem Namen/Nummern-Paar einen symbolischen Namen zuzuordnen, so daß man einen Bezug zum Quellcode des Programmes erhält. In der oberen Hälfte wird der akkumulierte Zeitanteil eines Zustandes an der bisherigen Laufzeit dargestellt. Beide Felder sind zur schnelleren Erkennung farblich hinterlegt, wobei die Farbe des Auslastungsfeldes mit der Farbskala am rechten Rand korrespondiert.

Eine Verallgemeinerung des Auslastungsfeldes stellt die Statistik über alle Zustände dar, die in Abbildung 5 gezeigt ist. Diese Darstellungsform ist sowohl für alle als auch für individuelle Prozessoren vorhanden und kann in weiten Grenzen variiert werden:

- Darstellung als Kuchendiagramm, Histogramm oder Tabelle;
- logarithmische oder lineare Skalierung im Histogramm;
- Sukzessives Ausblenden des Maximalwertes, um so die anderen Anteile besser einschätzen zu können;
- Statistik über die gesamte Programmlaufzeit oder einen Ausschnitt;
- Ausgabe der akkumulierten Zeitanteile oder der Aufrufzahlen;
- Statistik über alle Zustände oder nur einen einzelnen Zustand;
- Wertangaben wahlweise absolut oder relativ.

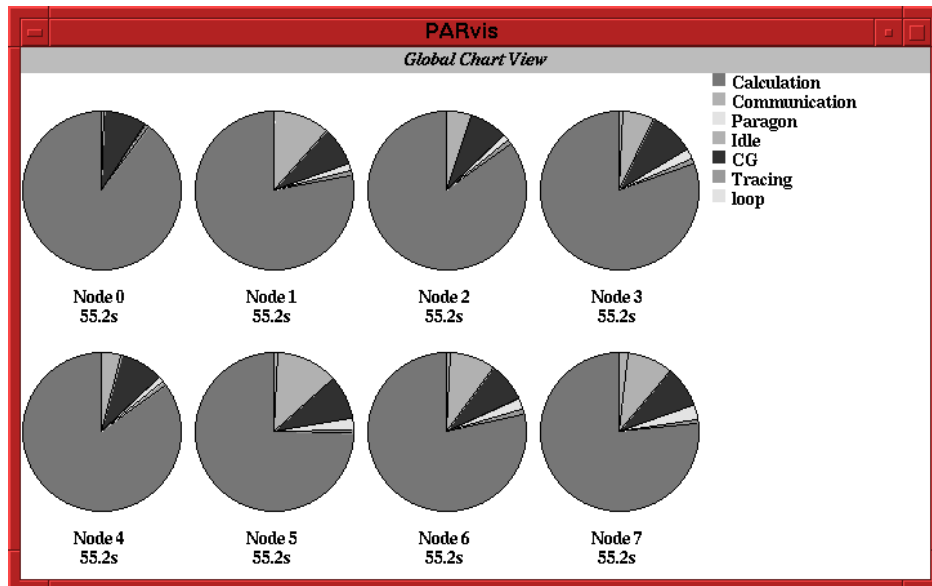


Abbildung 5: Statistik über Prozessorzustände

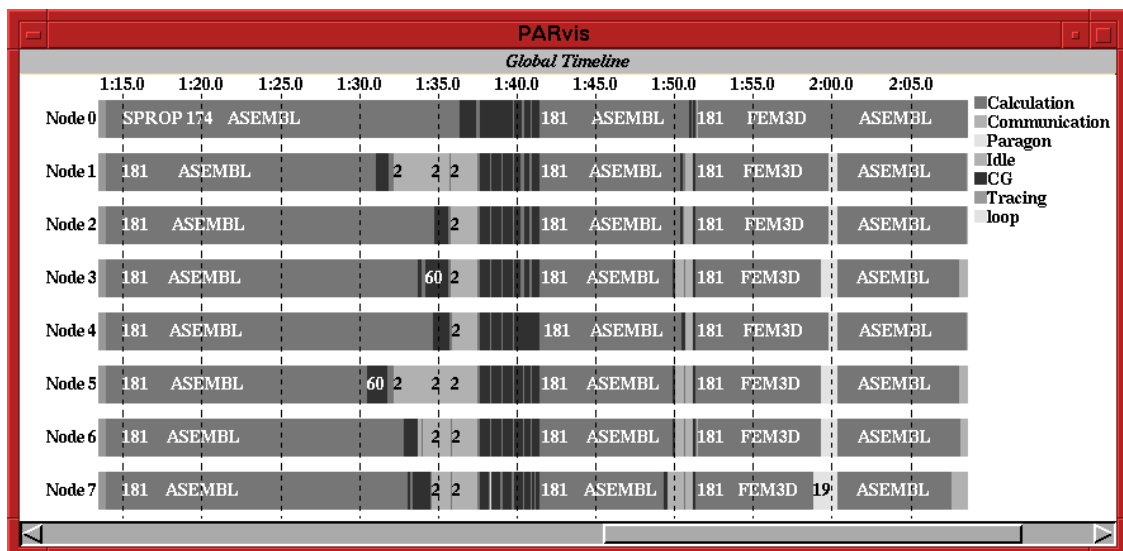


Abbildung 6: Zeitachsendarstellung von Prozessorzuständen

Die Zeitachsendarstellung der Prozessoraktivitäten (Abb. 6) erlaubt es, die Zustandswechsel der einzelnen Prozessoren in einem zeitlichen Zusammenhang zu verfolgen. Die Farben, die PARvis dabei zur Kennzeichnung der unterschiedlichen Aktivitäten nutzt, sind identisch mit denjenigen, die in den beiden vorher gezeigten Displays benutzt werden. Das Bild zeigt bereits eine aufgezoomte Darstellung eines Ausschnittes aus der Gesamtprogrammlaufzeit und eine im Programm identifizierte Problemstelle: durch unbalancierte Arbeitsverteilung müssen einige Prozessoren mehrere Sekunden auf die Ergebnisse ihrer Nachbarn warten. Zooming kann bis zu einem beliebigen Detail erfolgen; PARvis speichert dabei die vorherigen Ausschnitte, um mit einer *Undo*-Funktion wieder zurückgehen zu können.

4 Darstellung von Message-Passing

Das Programmiermodell des expliziten Versendens von Nachrichten ist das momentan am weitesten verbreitete Modell für massiv-parallele Rechner, da es auf die unterliegende Architektur dieser Systeme mit verteiltem Speicher am einfachsten abgebildet werden kann. Dieser Einfachheit in der Implementierung stehen allerdings deutliche Komforteinbußen in der Nutzung entgegen: Da kein globaler Adreßraum vorhanden

ist, ist der Programmierer gezwungen, seine Daten explizit auf die einzelnen lokalen Speicher zu verteilen und das Datenaustauschmuster zu programmieren. Die dabei möglichen Programmierfehler sind demgemäß vielfältig; sie reichen von Performance-Einbußen durch Versenden zu vieler kurzer Nachrichten bis zu schwer auffindbaren Fehlern im Kommunikationsmuster, die häufig unregelmäßig und schwer reproduzierbar auftreten. PARvis hilft bei der Suche nach solchen Fehlern durch zwei Anzeigeformen: Raum-Zeit-Diagramme und Statistiken. Abbildung 7 zeigt ein Beispiel für das Raum-Zeit-Diagramm. Nachrichten werden dabei als

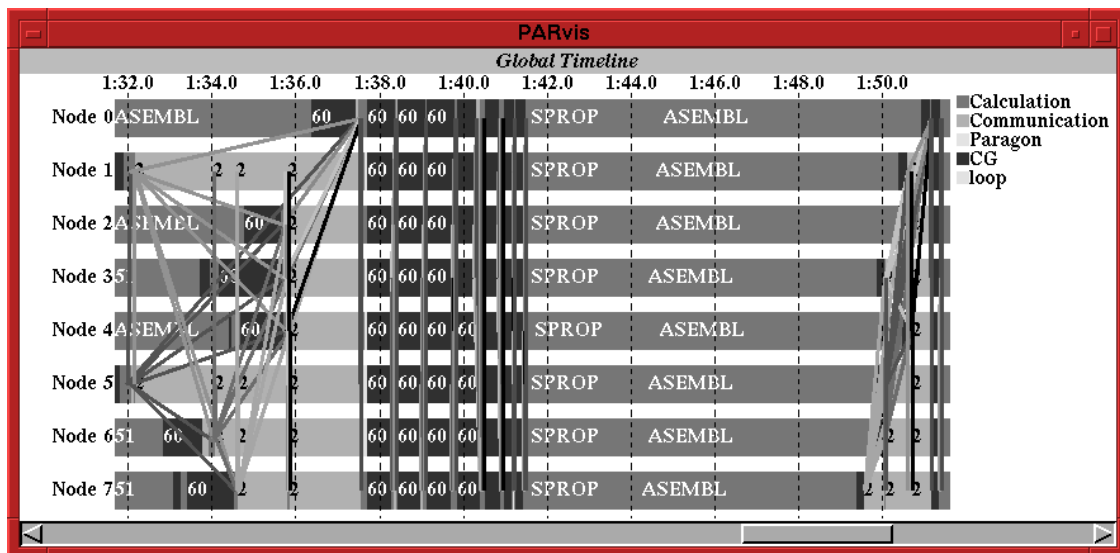


Abbildung 7: Raum-Zeit-Diagramm

Linien über der bereits angesprochenen Zeitachsendarstellung dargestellt, die vom Sender zum Empfänger laufen und deren Endpunkte Sende- und Empfangszeit repräsentieren. Die sich daraus ergebenden Muster erlauben, im Zusammenspiel mit der Zooming-Funktion und der Möglichkeit, eine einzelne Nachricht mit Typ und Länge auf Mausklick zu identifizieren, eine schnelle Erkennung fehlerhafter Kommunikationsmuster. Diese kann noch dadurch erleichtert werden, daß Nachrichten anhand ihres Typs unterschiedlich eingefärbt werden können; eine Variation der Linienbreite anhand der Länge der Nachricht ist ebenfalls möglich. Zum

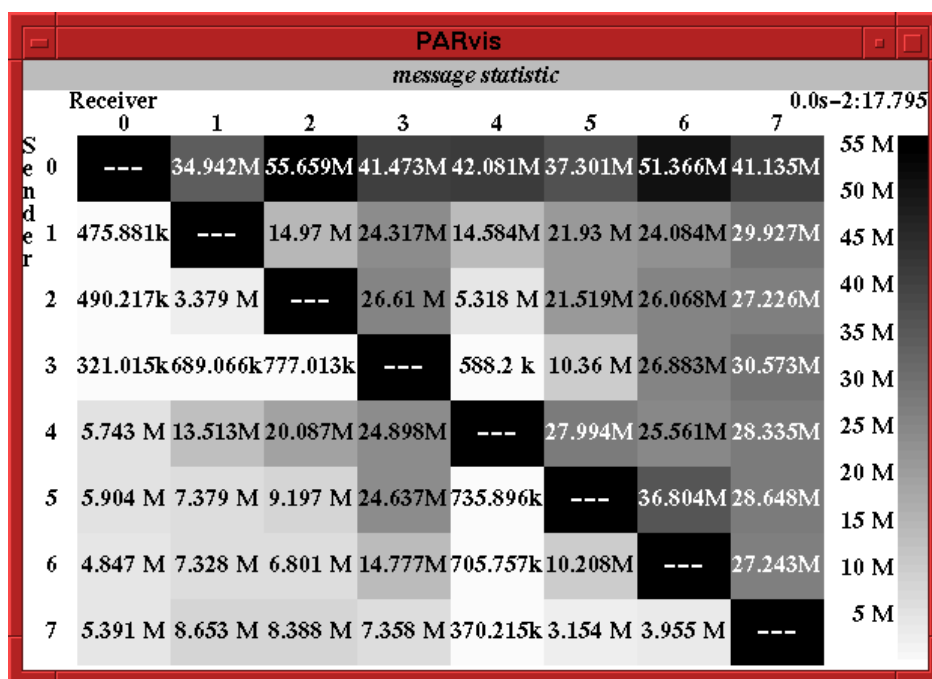


Abbildung 8: Statistik über Message-Passing

Erkennen unbalancierter oder ineffizienter Kommunikation bietet PARvis eine Statistik in Matrixform (Abb.

8) an. Diese kann für alle Sender-Empfänger-Paare folgende statistischen Werte berechnen:

- Gesamtzahl der Nachrichten;
- Gesamtlänge aller Nachrichten;
- mittlere, minimale und maximale Nachrichtenlänge;
- mittlere, minimale und maximale Datentransferrate.

Im Bild ist die durchschnittliche Datenrate über die Gesamtlaufzeit des Programmes dargestellt (auch hier ist es möglich, Statistiken nur über einen bestimmten Zeitabschnitt zu bilden). Man kann erkennen, daß Nachrichten, die von niedrigeren zu höheren Prozessornummern laufen, im Schnitt eine höhere Datenrate erreichen. Eine nähere Analyse mit dem Raum-Zeit-Diagramm ergibt, daß auch dies eine Folge der unbalancierten Arbeitsverteilung zwischen den Prozessoren ist (Abbildung 9): Da Prozessoren mit höheren Nummern früher mit ihrem Anteil fertig werden, müssen diese mit dem Versenden ihrer Ergebnisse warten, bis die vorderen Prozessoren ihre Rechenarbeit abgeschlossen haben und ihrerseits ihre Teilergebnisse versenden. Dieses Versenden geht aber deutlich schneller, da auf diese Nachrichten bereits gewartet wird.

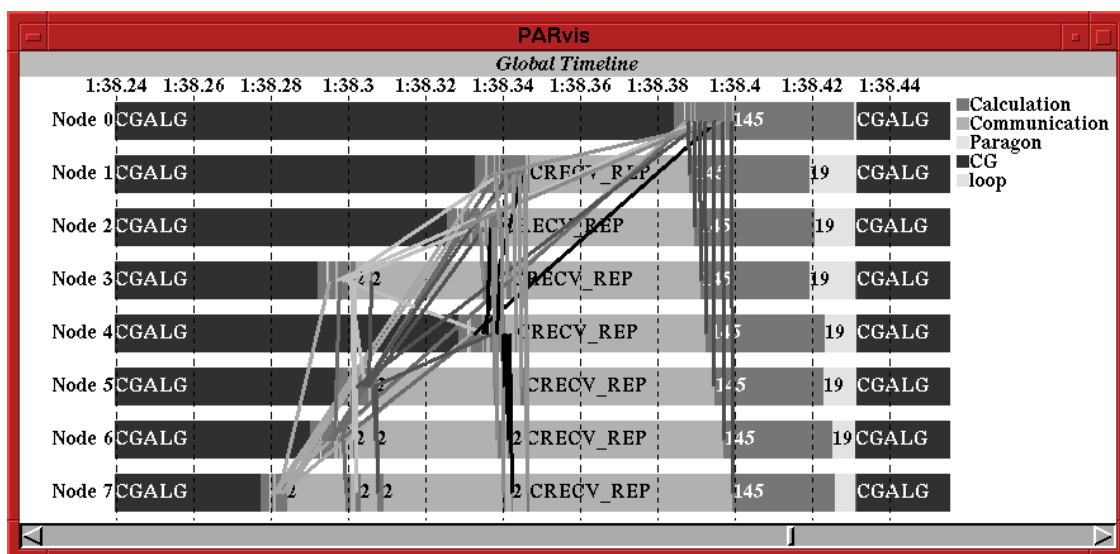


Abbildung 9: Unsymmetrische Kommunikation durch ungleiche Lastverteilung

5 Visualisierung von SVM

Aufgrund der Probleme des Message-Passing-Programmiermodells in der Praxis gibt es eine Reihe von Bestrebungen, auf Rechnern mit physikalisch verteiltem Arbeitsspeicher einen globalen Adreßraum zur Verfügung zu stellen. Neben Ansätzen wie HPF, die die Umsetzung der Speicherzugriffe in Nachrichten im Compiler vornehmen, wird auch häufig *Shared Virtual Memory* eingesetzt. SVM löst die nicht-lokalen Zugriffe zur Laufzeit des Programmes auf und eignet sich daher auch für Anwendungen mit unregelmäßigen oder dynamisch veränderlichen Datenzugriffen, die von einem Compiler nur schwierig umzusetzen sind. Bild 10 zeigt das Funktionsprinzip von SVM. Ähnlich wie bei der virtuellen Speicherverwaltung auf sequentiellen Rechnern wird der logische Adreßraum eines Prozessors in Seiten gleicher Größe unterteilt. Im Unterschied dazu wandern die Seiten hier jedoch nicht zwischen Arbeits- und Hintergrundspeicher, sondern *zwischen* den lokalen Speichern der einzelnen Prozessoren. Die Kohärenz wird üblicherweise dadurch gewährleistet, daß für eine Seite entweder nur eine Kopie mit Schreibrecht oder eine beliebige Zahl von Kopien mit Leserecht existieren dürfen. Die Einhaltung dieser Bedingung wird von einem speziellen Prozeß, dem sogenannten *Manager* überwacht. Dieser nimmt Anfragen von Seiten entgegen und steuert Transfer und Invalidierung von Seitenkopien.

Dieser Mechanismus sorgt dafür, daß man einen Parallelrechner mit SVM theoretisch genauso programmieren kann, als ob er einen physikalisch gemeinsamen Speicher hätte. In der Praxis führt die Seitenstruktur

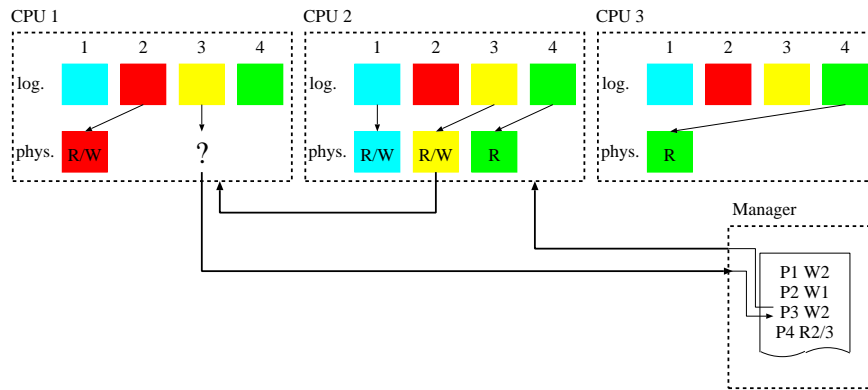


Abbildung 10: Arbeitsweise von SVM

des Speichers jedoch zu erheblichen Performance-Einbußen, wenn nicht auf die SVM-Verwaltung Rücksicht genommen wird. Ein großes Problem stellt dabei das sogenannte *False Sharing* dar. Beim False Sharing greifen zwei Prozessoren teilweise schreibend auf unterschiedliche Daten in der gleichen Seite zu. Das Ergebnis ist, daß obwohl logisch keine Datenkonflikte vorhanden sind, die betroffene Seite ständig zwischen den Prozessoren hin- und herwandert, was drastische Performance-Verschlechterungen zur Folge hat.

PARvis bietet eine Reihe von Anzeigeformen an, die die momentane Seitenverteilung und Seitenwanderungen in einem SVM-basierten Programm visualisieren. Die einfachste Darstellung ist die Seitenübersicht (Abb. 11), die alle Seiten als Kästchen darstellt: Im Beispiel werden die momentanen Eigentümer der Seite

PARvis																	
Page Overview																	
P1 O6	P2 O0	P3 O1	P4 O2	P5 O4	P6 O5	P7 O7	P8 O9	P9 O0	P10 O6	P11 O1	P12 O2	P13 O4	P14 O5	P15 O9	P16 O7	P17 O0	Node 0
P18 O1	P19 O2	P20 O8	P21 O9	P22 O5	P23 O5	P24 O7	P25 O3	P26 O6	P27 O1	P28 O2	P29 O9	P30 O8	P31 O5	P32 O0	P33 O6	P34 O3	Node 1
P35 O1	P36 O9	P37 O2	P38 O8	P39 O5	P40 O0	P41 O6	P42 O3	P43 O4	P44 O1	P45 O2	P46 O7	P47 O5	P48 O0	P49 O6	P50 O9	P51 O3	Node 2
P52 O1	P53 O2	P54 O7	P55 O5	P56 O4	P57 O3	P58 O9	P59 O8	P60 O2	P61 O7	P62 O5	P63 O7	P64 O4	P65 O3	P66 O9	P67 O8	P68 O8	Node 3
P69 O2	P70 O7	P71 O6	P72 O0	P73 O4	P74 O3	P75 O9	P76 O8	P77 O2	P78 O5	P79 O7	P80 O0	P81 O0	P82 O0	P83 O0	P84 O0	P85 O0	Node 4
P86 O0	P87 O1	P88 O1	P89 O1	P90 O1	P91 O1	P92 O1	P93 O1	P94 O1	P95 O1	P96 O2	P97 O2	P98 O2	P99 O2	P100 O2	P101 O2	P102 O2	Node 5
P103 O3	P104 O3	P105 O3	P106 O3	P107 O3	P108 O3	P109 O3	P110 O3	P111 O4	P112 O4	P113 O4	P114 O4	P115 O4	P116 O4	P117 O4	P118 O4	P119 O5	Node 6
P120 O5	P121 O5	P122 O5	P123 O5	P124 O5	P125 O5	P126 O5	P127 O6	P128 O6	P129 O6	P130 O6	P131 O6	P132 O6	P133 O6	P134 O6	P135 O7	P136 O7	Node 7
P137 O7	P138 O7	P139 O7	P140 O7	P141 O7	P142 O7	P143 O8	P144 O8	P145 O8	P146 O8	P147 O8	P148 O8	P149 O8	P150 O8	P151 O9	P152 O9	P153 O9	Node 8
P154 O9	P155 O9	P156 O9	P157 O9	P158 O9	P159 O9	P160 O9	P161 O0	P162 O1	P163 O2	P164 O3	P165 O4	P166 O5	P167 O6	P168 O7	P169 O8	P170 O9	Node 9
P171 O0	P172 O1	P173 O2	P174 O3	P175 O4	P176 O5	P177 O6	P178 O7	P179 O8	P180 O9	P181 O0	P182 O1	P183 O2	P184 O3	P185 O4	P186 O5	P187 O6	Node 0
P188 O7	P189 O8	P190 O9	P191 O0	P192 O1	P193 O2	P194 O3	P195 O4	P196 O5	P197 O6	P198 O7	P199 O8	P200 O9	P201 O0	P202 O1	P203 O2	P204 O3	Node 1
P205 O4	P206 O5	P207 O6	P208 O7	P209 O8	P210 O9	P211 O0	P212 O1	P213 O2	P214 O3	P215 O4	P216 O5	P217 O6	P218 O7	P219 O8	P220 O9	P221 O0	Node 2
P222 O1	P223 O2	P224 O3	P225 O4	P226 O5	P227 O6	P228 O7	P229 O8	P230 O9	P231 O0	P232 O1	P233 O2	P234 O3	P235 O4	P236 O5	P237 O6		Node 3

Abbildung 11: Seitenübersicht

dargestellt, es ist aber möglich, andere Seitenparameter wie zum Beispiel die akkumulierte Transferzeit für diese Seite darzustellen. Insbesondere diese Angabe erlaubt das schnelle Auffinden von Seiten, für die False Sharing auftritt. Ein Beispiel für die Visualisierung der Wanderungsbewegungen für eine solche Seite zeigt Abbildung 12. Hier wandert Seite 16 ständig zwischen Prozessor 0 und 7 hin und her, die Prozessoren warten häufiger, als daß sie rechnen können. Die beiden gezeigten Darstellungen sind nur eine Auswahl der in PARvis möglichen SVM-Visualisierungen; eine komplette Beschreibung kann in [2] nachgelesen werden.

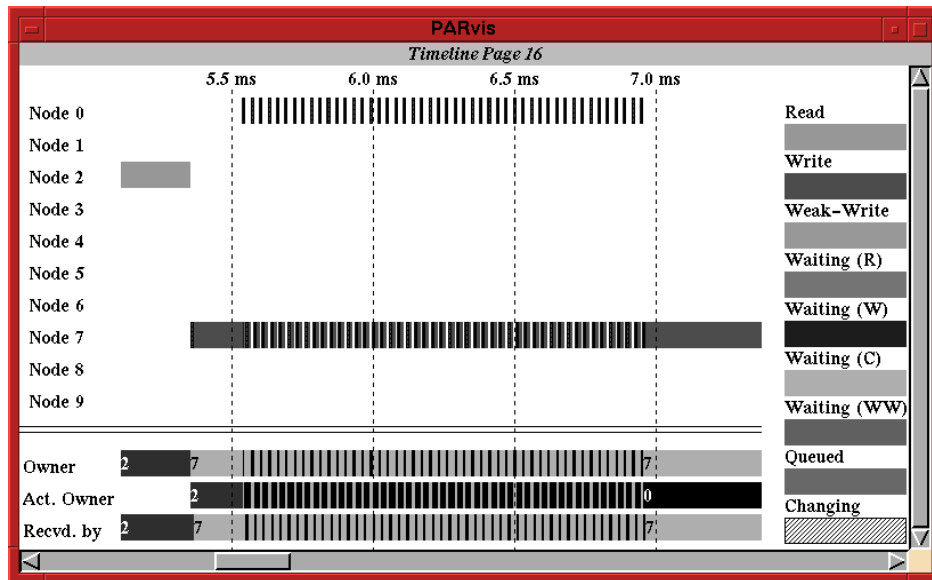


Abbildung 12: Visualisierung von Seitenwanderungen

6 Metacomputing

Die Tatsache, daß eine Reihe von Anwendungen in Teile zerlegt werden können, die auf unterschiedlichen Parallelrechnerarchitekturen verschieden gut implementiert werden können, hat im Zusammenhang mit dem Auftauchen schneller WAN-Netze wie Hippi oder ATM zu Bestrebungen geführt, Anwendungen verteilt auf miteinander verbundenen Parallelrechnern ablaufen zu lassen. Da aber Transfers zwischen verschiedenen Rechnern immer langsamer sein werden als die rechnerinterne Kommunikation, ist es erforderlich, bei der Performance-Analyse eines solchen, als *Metacomputer* bezeichneten Rechnerverbundes die Gruppierungen darstellen zu können. PARvis besitzt zu diesem Zweck Erweiterungen, die automatisch in Aktion treten, wenn im Tracefile entsprechende Gruppierungen der Prozessoren angegeben werden. Die Benennung der Prozessoren erfolgt dann nicht mehr streng linear, sondern in Gruppen gestaffelt. In der Zeitachsendarstellung (Abb. 13) ist es mit dieser Unterteilung möglich, einzelne Gruppen (wie hier die SP2-Prozessoren) für Detailuntersuchungen vergrößert darzustellen. Alternativ können einzelne Gruppen temporär ganz aus der Darstellung entfernt werden.

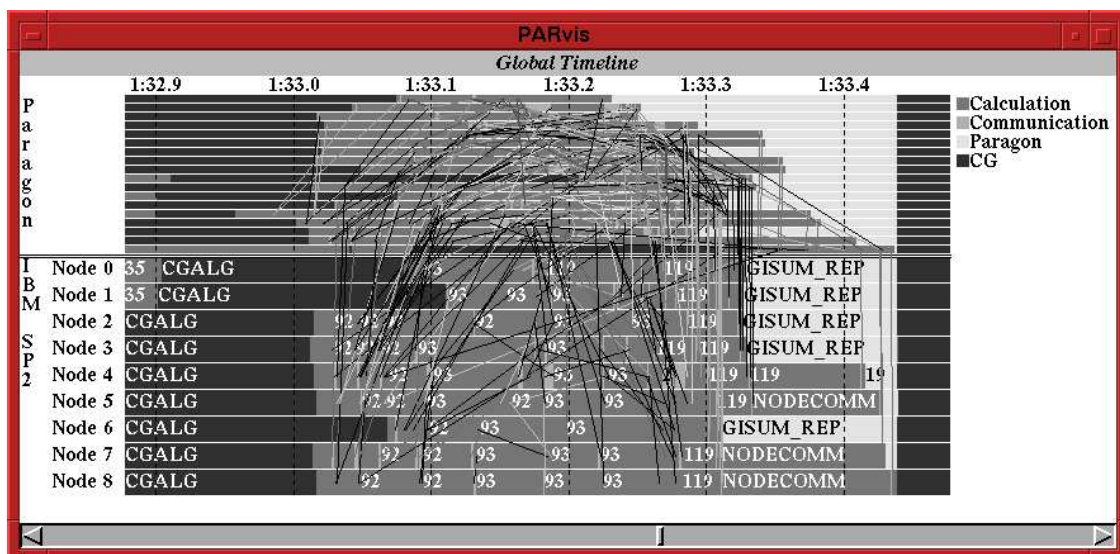


Abbildung 13: Zooming von Gruppen

7 Zusammenfassung

Massiv-parallele Rechner haben bis heute mit dem Problem zu kämpfen, daß ihre Programmierung durch eine unzureichende Unterstützung mit Werkzeugen erschwert wird. Insbesondere fehlen Werkzeuge, die einen leicht verständlichen Einblick in die dynamischen Abläufe in einem parallelen Programm gewähren. Die PARvis-Visualisierungsumgebung schließt diese Lücke: Aus einer zur Programmlaufzeit geschriebenen Trace-Datei werden graphische Darstellungen des Programmablaufes generiert, die einen intuitiven Zugang zum Programmverhalten ermöglichen. Der Zeitaufwand für Debugging und Optimierung kann auf diese Weise drastisch verkürzt werden. Die Anpassung an eine Reihe unterschiedlicher Programmiermodelle sowie die Portierbarkeit garantieren ein weites Einsatzfeld für PARvis.

Literatur

- [1] A. Arnold, *PARvis: Eine X-basierte Umgebung zur Visualisierung von parallelen Programmen in Multiprozessorsystemen*, Jül-2848, Forschungszentrum Jülich (KFA), 1993
- [2] Ch. Müllender, *Visualisierung der Speicheraktivitäten von parallelen Programmen in Systemen mit virtuell gemeinsamem Speicher*, Jül-2911, Forschungszentrum Jülich (KFA), 1994
- [3] W. E. Nagel und A. Arnold, *PARvis: Ein Werkzeug zur Visualisierung von parallelen Programmen auf Mehrprozessorsystemen*, Proc. 7. ITG/GI Fachtagung MMB '93 (Kurzberichte und Werkzeugvorstellung), S. 178-187, 1993
- [4] R. Williams and W. E. Nagel, *Optimization of output bandwidth from a Paragon*, Technical Report CCSF-44, Caltech Concurrent Supercomputing Facilities, Pasadena, CA, 1994
- [5] A. Arnold, U. Detert, W. E. Nagel, *Performance Optimization of Parallel Programs: Tracing, Zooming, Understanding*, Proc. Cray User Group Conference Spring 1995, pp. 252-258
- [6] *Paragon application tools user's guide*, Intel Corporation, 1993
- [7] D. A. Reed, R. A. Aydt, T. M. Madhyastha, R. J. Noe, K. A. Shields, and B. W. Schwartz, *An overview of the Pablo performance analysis environment*, Technical Report, Dept. of Computer Science, University of Illinois, Urbana-Champaign, 1992