High Performance Fortran: Die Sprache und ihre Compiler

Thomas Brandes, Clemens-August Thole
Gesellschaft für Mathematik und Datenverarbeitung (GMD)
Schloß Birlinghoven, D-53757 Sankt Augustin

Abstract

Das datenparallele Programmiermodell hat sich als ein Programmiermodell erwiesen, das sich für nahezu alle parallelen Architekturen eignet und somit Portierbarkeit garantiert. Da es zudem einfach und benutzerfreundlich ist, hat es große Chancen, das Software-Problem für Parallelrechner zumindestens für eine bestimmte Klasse von Anwendungen zu verkleinern.

Dieses Programmiermodell wird mittlerweile auch durch die Programmiersprache High Performance Fortran unterstützt. Es handelt sich dabei um Spracherweiterungen und -modifikationen von Fortran 90, die durch das High Performance Fortran Forum festgelegt worden sind.

In diesem Aufsatz wird kurz das datenparallele Programmiermodell und seine Unterstützung mittels High Performance Fortran vorgestellt. Ferner soll durch erste Erfahrungen mit einem Prototyp-Compiler beleuchtet werden, wie effizient dieses Modell mittels Compiler auf Parallelrechnern mit verteiltem Speicher realisiert werden kann.

1 Das datenparallele Programmiermodell

Im datenparallelen Programmiermodell wird die inhärente Parallelität von gleichartigen Operationen auf den Elementen großer Felder (Arrays) genutzt, um entsprechende Leistungssteigerungen zu erzielen. Hierzu werden die Arrays auf die unterschiedlichen Prozessoren verteilt und die entsprechenden Operationen auf den jeweiligen Teilfeldern ausgeführt, wobei gegebenenfalls Daten zwischen den Prozessoren ausgetauscht werden müssen.

Aufgrund der globalen Sicht der Daten und durch die Tatsache, daß nur ein Kontrollfluß existiert, ist dieses Programmiermodell wesentlich anwendungsfreundlicher als das Modell der sequentiellen Prozesse mit explizitem Nachrichtenaustausch, auch Message-Passing Programmiermodell genannt.

Obgleich der Anwender in seinem datenparallelen Programm stets die ganzen Arrays sieht, ist es für eine effiziente Ausführung derartiger Programme auf Parallelrechnern mit verteiltem Speicher sehr wichtig, eine hohe Datenlokalität mit möglichst geringem Kommunikationsaufwand zu erreichen. Hierzu gibt der Benutzer in der Regel über Direktiven dem Compiler entsprechende Hilfestellungen.

Das folgende Programm zeigt z.B. eine LU-Zerlegung ohne Pivot-Strategie als datenparalleles Programm in Fortran 90, wobei die Array-Operationen an sich die Parallelität explizit festlegen.

```
 real a(N,N)
 ...
 do K = 1, N-1
    a(K+1:N,K) = a(K+1:N,K) / a(K,K)
    a(K+1:N,K+1:N) = a(K+1:N,K+1:N) -
$                    SPREAD (A(K+1:N,K), DIM=2, NCOPIES = N-K) *
$                    SPREAD (A(K,K+1:N), DIM=1, NCOPIES = N-K)
 end do
```

Das datenparallele Programmiermodell hat durch kommerzielle SIMD-Architekturen (single instruction, multiple data) wie die Connection Machine und MasPar eine sehr große Verbreitung gefunden. Mittlerweile existieren zahlreiche Anwendungen in CM Fortran [Thi91] und MP Fortran. Beides sind Spracherweiterungen, die im wesentlichen auf Fortran 90 basieren. Trotz der durch den einfachen Kontrollfluß implizierten SIMD-Semantik kann das Programmiermodell an sich auch auf MIMD-Architekturen eingesetzt werden [Fox91]. Dieses ist u.a. auch ein Grund dafür, daß sich ein Forum zur Definition von High Performance Fortran gebildet hat.

2 High Performance Fortran

Auf Inititative von Prof. Dr. Ken Kennedy, Rice University, haben sich seit Januar 1992 regelmäßig Vertreter der meisten Parallelrechnerhersteller, verschiedener Universitäten und Forschungseinrichtungen getroffen, um Spracherweiterungen und -modifikationen des Fortrtan 90 Standards zu spezifizieren, mit deren Hilfe das datenparallele Programmiermodell unterstützt wird.

Nach einer Reihe von Treffen dieses High Performance Fortran Forums ist mittlerweile die Version 1.0 von High Performance Fortran (HPF) als endgültige Sprachdefinition verabschiedet worden [Hig93a]. Es wurde versucht, Abweichungen zu anderen Standards und direkte Konflikte mit Fortran 77 oder Fortran 90 zu vermeiden bzw. zu minimieren.

2.1 Überblick

In HPF erlauben Compilerdirektiven die Spezifikation der Abbildung von Datenobjekten auf Prozessoren. Einige wenige Spracherweiterungen und eine umfangreiche Bibliothek von Unterroutinen ermöglichen es, Operationen auf Daten so zu formulieren, daß sie ähnlich zu den Array-Operationen von Fortran 90 parallel ausgeführt werden können.

HPF enthält nicht die Möglichkeit, die Abbildung von Anweisungen auf die Prozessoren festzulegen. Diese Abbildung ist Aufgabe des Compilers unter Berücksichtigung der aktuellen Datenverteilung.

Ein spezielles Interface erlaubt den Aufruf von Unterprogrammen, die in einem anderen Programmiermodell (z.B. im Message-Passing Programmiermodell) geschrieben worden

Abbildung 1: HPF-Modell zur Abbildung von Datenobjekten auf Prozessoren

sind. Da das datenparallele Programmiermodell gewisse Grenzen besitzt, insbesondere was die funktionale Parallelität anbetrifft, kann dieses sehr wichtig für viele interessante Anwendungen sein.

Um möglichst kurzfristig Compiler zur Verfügung zu haben und damit eine schnelle Verbreitung dieses Programmiermodells zu garantieren, hat man sich zudem auf eine Untermenge von High Performance Fortran und Fortran 90 geeinigt. Diese Untermenge, die als 'HPF Subset' bezeichnet wird, umfaßt insbesondere die Array-Notation und die dynamischen Arrays aus Fortran 90 sowie die Direktiven zur Verteilung und die parallelen Schleifen. Verzichtet wird allerdings auf das gesamte Modulkonzept und auf dynamische Umverteilungen.

Die derzeitige Sprachdefinition berücksichtigt kaum Spracherweiterungen, welche die asynchrone Arbeitsweise von MIMD-Architekturen ausnutzen. Auch die I/O-Problematik ist unzureichend gelöst. Aus diesem Grunde existiert ein weiteres Dokument [Hig93b], das als Basis für die Fortentwicklung dieser Sprache dient. Anfang 1995 soll ein neues Forum an einer Erweiterung (HPF-2) arbeiten.

2.2 Datenverteilungen

Die Speicherzugriffskosten zwischen lokalen und globalen Zugriffen variieren aufgrund der Speicherhierarchie stark voneinander. Dies gilt auch für parallele Systeme, für die ein globaler Adressraum bereitgestellt wird. Daher ist die Verteilung der Daten auf die Prozessoren von enormer Wichtigkeit für die Effizienz. Da automatische Techniken zur Verteilung der Daten nicht ausreichen, wird der Benutzer diese Datenverteilung in der Regel selber vornehmen. Hierzu sieht HPF eine Reihe von Möglichkeiten und Direktiven vor. Die Abbildung 1 zeigt das grundsätzlich zugrunde liegende mehrstufige Modell.

Alignment. Arrays können mittels der ALIGN-Direktive zu einer Gruppe zusammengefaßt werden, indem sie auf ein entsprechendes Ziel-Array abgebildet werden. Falls kein natürliches Ziel-Array existiert, kann ein sogenanntes Template verwendet werden, das selber keinen Speicherplatz beansprucht. Die Abbildung erfolgt über einfache lineare Funktionen, allerdings können auch alle Elemente einer oder mehrerer Dimensionen auf ein Element des Zielobjekts abgebildet werden oder entlang von Dimensionen des Zielobjekts repliziert werden. Mit diesen Direktiven wird sichergestellt, daß zueinander gehörende Daten lokal sind und damit unnötige Kommunikation vermieden wird.

```
!HPF$  TEMPLATE GRID (200,200,200)
       REAL F(2:198,1:200)
       REAL, DIMENSION(1:200,1:200) :: U1, U2
!HPF$  ALIGN F(I,J) WITH GRID(I,*,J)
!HPF$  ALIGN (I,J) WITH GRID(I,*,J) :: U1, U2
```

Verteilen von Datenobjekten. Ein Template oder ein Array und die ihm zugeordneten Arrays werden mittels der DISTRIBUTE-Direktive auf die abstrakten Prozessoren einer Prozessorkonfiguration verteilt. HPF sieht zyklische (CYCLIC) und auch Block-Zerteilungen (BLOCK) für die zu verteilenden Dimensionen vor.

Abstrakte Prozessor Konfigurationen. Durch die PROCESSORS-Direktive können die dem Problem angemessenen Prozessorkonfigurationen in Form von Arrays festgelegt werden. Verschiedene Prozessorkonfigurationen werden in der Regel auf die gleichen physikalischen Prozessoren abgebildet. Deren Konfiguration läßt sich über entsprechende HPF-Intrinsics abfragen.

```
!HPF$  PROCESSORS R (8, NUMBER_OF_PROCESSORS()/8)
!HPF$  DISTRIBUTE GRID(BLOCK, *, BLOCK) ONTO R
```

Dynamische Abbildungen. Die Spezifikation von Aligment und Verteilungen behält in der Regel nur für ein Unterprogramm Gültigkeit. Ändert sich die Verteilung zwischen aufrufendem und aufgerufenem Unterprogramm, so sind die Daten entsprechend umzuverteilen. HPF sieht allerdings Mechanismen vor, aktuelle Verteilungen zu vererben bzw. auch Unterprogramme für Parameter mit einer beliebigen Verteilung zu definieren.

Mit Hilfe der Direktive DYNAMIC wird spezifiziert, daß sich die aktuelle Verteilung der hiermit spezifizierten Objekte dynamisch zur Laufzeit ändern kann. Auf diese Art und Weise sind dynamische Umverteilungen auch ganzer Gruppen von Objekten möglich.

Storage und Sequence Association. HPF hat sich um weitgehende Kompatibilität zu Fortran 90 bemüht. Bestimmte Arten, formale Parameter von Unterprogrammen an aktuelle Parameter der aufrufenden Routine zu binden, oder aber bestimmte Arten von EQUIVALENCE und COMMON Anweisungen machen Annahmen über die Anordnung und Größe der Elemente von Arrays im Speicher (Sequence und Storage Association). HPF stellt Direktiven zur Verfügung, die für die benannten Variablen die vollen Eigenschaften von Fortran 90 garantieren und nur eingeschränkte Abbildungen der Datenobjekte auf die Prozessoren erlauben. Fehlen diese Direktiven, so ist Storage und Sequence Association nicht gewährleistet.

2.3 Parallele Sprachkonstrukte

Im datenparallelen Programmiermodell wird die Parallelität an sich explizit festgelegt, wenngleich mittels Abhängigkeitsanalyse und automatischen Parallelisierungstechniken auch implizite Parallelität genutzt werden kann. Diese explizite Parallelität wird im wesentlichen durch die Array-Operationen und Array-Anweisungen festgelegt, wo die Anweisungen parallel für alle Elemente ausgeführt werden. HPF sieht jedoch weitergehende Möglichkeiten zur Spezifizierung von Parallelität vor.

FORALL-Anweisung und FORALL-Konstrukt. Die FORALL-Anweisung entspricht einer (maskierten) Arrayzuweisung. Dadurch, daß der Indexraum dieser Anweisung jedoch explizit verfügbar ist, lassen sich verschiedene Anweisungen überhaupt erst oder einfacher formulieren. Sie hat sich in CM Fortran [Thi91] sehr bewährt. Das FORALL-Konstrukt ist eine vereinfachte Schreibweise für eine Folge von FORALL-Anweisungen.

Die LU-Zerlegung kann mittels paralleler Schleifen auch wie folgt definiert werden:

```
      do K = 1, N-1
         forall (J=K+1:N) a(J,K) = a(J,K) / a(K,K)
         forall (J=K+1:N, I=K+1:N)
            a(J,I) = a(J,I) - a(J,K) * a(K,I)
         end forall
      end do
```

PURE Unterprogramme. Spezielle Unterprogramme, die das Attribut PURE besitzen und damit zustandsfrei und frei von jeglichen Seiteneffekten sein müssen, können innerhalb einer FORALL-Schleife aufgerufen werden. Derartige Unterprogramme unterliegen einer Reihe von syntaktischen Einschränkungen.

INDEPENDENT Direktive. Mit der INDEPENDENT-Direktive sichert der Anwender zu, daß zwischen den Iterationen einer Schleife keine Abhängigkeiten existieren. Eine parallele Ausführung der Iterationen wird dadurch ermöglicht.

```
!hpf$ independent (I)
      do I = 1, N
         A(P(I)) = B(I)
      end do
```

HPF Standard-Bibliothek. Die Erfahrung mit massiv parallelen Rechnern hat gezeigt, daß es einige wichtige elementare Operationen gibt, die hochgradig parallel sind und in Fortran 90 nur mit Schwierigkeiten formuliert werden können. Beispiele hierfür sind Scatter- und Gather-Operationen, die indirekten Zugriff auf Daten spezifizieren. Die HPF Standard-Bibliothek enthält die Definitionen für verschiedene Klassen von derartigen Operationen.

3 Adaptor

Adaptor (Automatic Data Parallelism Translator) ist ein an der GMD entwickeltes Werkzeug, mit dessen Hilfe datenparallele Fortran Programme automatisch partitioniert werden können, d.h. es wird ein entsprechendes Fortran 77 Programm mit explizitem Nachrichtenaustausch generiert, das anschließend auf einer Reihe von unterschiedlichen Hardware-Architekturen ablauffähig ist [Bra93].

Wesentliches Ziel der Entwicklung von Adaptor war die automatische Portierung datenparalleler CM Fortran Programme auf andere Message-Passing Architekturen. Adaptor wurde aber auch dafür eingesetzt, erste Erfahrungen mit High Performance Fortran zu sammeln, insbesondere was Effizienz, Optimierungs- und Einsatzmöglichkeiten anbetrifft.

3.1 Aufbau von Adaptor

Adaptor selbst besteht aus einem interaktiven 'Source-to-Source' Transformationssystem und einer Bibliothek von Unterprogrammen für den Nachrichtenaustausch und globaler Operationen auf Arrays (z.B. globale Reduktionen, Transpose, indirekte Adressierung auf verteilten Arrays, Randaustausch, Shift, Replizieren und Umverteilten von Daten).

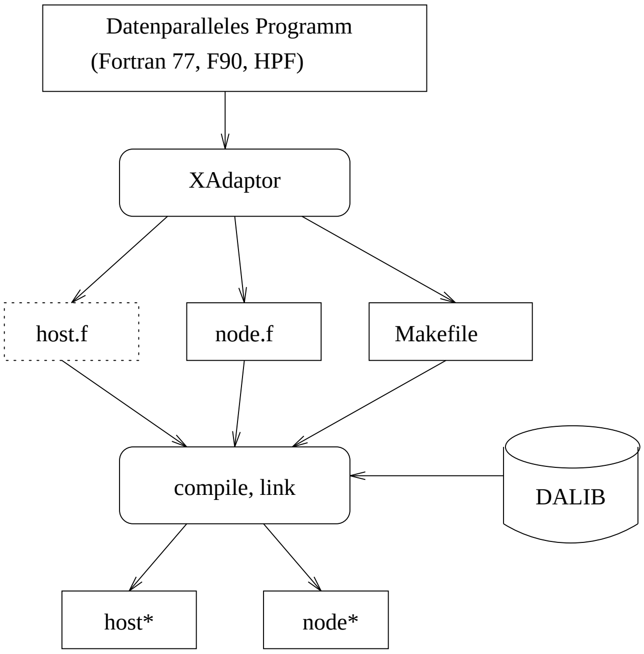

Abbildung 2: Aufbau von Adaptor

Quellprogramme für Adaptor, Beschreibungen in PostScript und eine große Anzahl von Beispielprogrammen sind mittels 'anonymem ftp' über Internet verfügbar.

```
ftp.gmd.de        (129.26.8.90)
im Verzeichnis    gmd/adaptor
```

3.2 Funktionsweise

Adaptor generiert aus dem datenparallelen Programm in der Regel ein entsprechendes SPMD Program (single program, multiple data), das auf allen Knoten gestartet wird.

Der globale Namensraum wird dadurch emuliert, daß skalare Daten und ausgewählte Arrays repliziert werden. Andere Arrays wiederum werden entsprechend den Direktiven verteilt.

Analog wird der skalare Code auf alle Knoten repliziert, d.h. bei einem Zugriff auf ein verteiltes Element wird der aktuelle Wert an alle Prozessoren gesendet. Array-Operationen und parallele Schleifen werden auf die jeweils lokalen Daten eingeschränkt, wobei gegebenenfalls benötigte nicht-lokale Daten ausgetauscht werden müssen (implizite Kommunikation).

Für eine Reihe von Array-Operationen (Reduktionen, Shiften von Arrays, Transpose) werden optimierte Library-Routinen zur Verfügung gestellt, unter anderem auch für gewisse Routinen aus der HPF Standard Bibliothek. Obgleich diese Routinen in 'C' geschrieben sind und ein portables Message-Passing Interface verwenden, bieten diese Routinen ein sehr hohes Potential für maschinenabhängige Optimierungen.

3.3 Erste Ergebnisse

Zur Untersuchung der Leistungsfähigkeit wurden Programme aus der High Performance Fortran Benchmark Suite verwendet, die an der Syracuse University, New York, zusammengestellt worden ist [MFL+92]. Die CM Fortran Version wurde leicht modifiziert, so daß eine Partitionierung mittels Adaptor möglich war.

3.3.1 Leistungsgewinn und Effizienz

Die nachfolgende Tabelle zeigt den Leistungsgewinn und die Effizienz für einige einfache datenparallele Anwendungen aus der oben erwähnten Benchmark Suite in einfacher Genauigkeit auf dem iPSC/860 am ZAM, Forschungszentrum Jülich. Die von Adaptor generierten Programme wurden dabei mittels des Fortran 77 Compilers (if77/NX Sun4 Rel 4.0, Flags '-O2 -Mvect') übersetzt. Die Ergebnisse belegen insbesondere die Skalierbarkeit der datenparallelen Anwendungen auch für MIMD Architekturen (iPSC/860).

Problem	8 Knoten	16 Knoten	32 Knoten
Trapez-Regel	7.96 (99.5 %)	15.79 (98.7 %)	31.25 (97.6 %)
Reduktionsfunktion 1	7.41 (92.6 %)	14.08 (88.0 %)	28.17 (88.0 %)
Reduktionsfunktion 2	6.44 (80.5 %)	12.88 (80.5 %)	25.75 (80.5 %)
Reduktionsfunktion 3	6.08 (76.0 %)	13.16 (82.2 %)	26.13 (81.6 %)
Einfache Suche	7.12 (89.0 %)	14.14 (88.4 %)	27.91 (87.2 %)
Lineare Gleichungen	4.86 (60.7 %)	8.75 (54.7 %)	16.17 (50.5 %)
Lagrange Interpolation	7.97 (99.6 %)	15.76 (98.5 %)	31.17 (97.4 %)
Dividierte Differenzen	7.61 (95.2 %)	13.92 (87.0 %)	25.90 (80.9 %)
Finite Differenzen	7.97 (99.6 %)	15.40 (96.3 %)	28.88 (90.2 %)
Fourier's Momente	7.94 (99.3 %)	14.90 (93.1 %)	29.20 (91.3 %)
Array-Konstruktion	6.84 (85.5 %)	10.95 (68.4 %)	16.85 (52.6 %)
Floating-Point Arithmetik	7.91 (98.9 %)	15.80 (98.7 %)	31.71 (99.0 %)
Integration	7.20 (90.0 %)	14.00 (87.5 %)	25.20 (78.8 %)
Chebyshev Interpolation	7.48 (93.5 %)	14.84 (92.7 %)	28.75 (89.8 %)

3.3.2 Vergleich von HPF mit Message-Passing Programmen

In der Benchmark Suite waren für die obigen Probleme ebenso von Hand codierte Message-Passing Programme verfügbar, die auf dem Message-Passing Interface PICL basieren. Der Vergleich der von Adaptor generierten Programmen mit diesen Programmen gibt Aufschluß darüber, inwieweit High Performance Fortran konkurrenzfähig zum Message-Passing Programmiermodell ist.

Beim Vergleich der Laufzeiten stellte sich jedoch heraus, daß die automatisch generierten Message-Passing Programme immer schneller waren, manchmal sogar bis zu viermal so schnell. Dies belegt allerdings nur, daß die handgeschriebenen Programme sehr ineffizient sind. In der Tat wird bei Adaptor die Parallelität der Array-Operationen auch für die Vektorisierung genutzt, was bei den anderen Programmen nicht so sehr der Fall ist. Ebenfalls zeigte sich, daß bei datenparallelen Programmen oft nur kleine Änderungen erforderlich waren, um wesentliche Leistungssteigerungen zu erzielen. Für die gleiche Optimierung wird dagegen bei den hand-codierten Programmen wesentlich mehr Umstellungsaufwand benötigt.

3.3.3 SIMD- und MIMD-Ausführung

Die CM 5 bietet derzeitig sowohl das datenparallele Programmiermodell als auch das Message-Passing Programmiermodell an. Hierzu wurden die 14 Anwendungen zum einen als CM Fortran Programme vom CM Fortran Compiler übersetzt (SIMD). Diese Laufzeiten wurden mit denen der von Adaptor generierten Fortran 77 Programme mit explizitem Nachrichtenaustausch verglichen (MIMD). Die Vektor-Einheiten auf jeweils einem Knoten wurden nicht verwendet, da diese von Adaptor derzeit nur mit Einschränkungen genutzt werden können.

Wenngleich die folgenden Ergebnisse auf einer CM-5 mit 64 Knoten als sehr vorläufig zu betrachten sind, insbesondere weil der CM Fortran Compiler noch in einer sehr frühen Version verwendet wurde und die Programme für Adaptor leicht modifiziert worden sind, wird deutlich, daß trotz SIMD Semantik die Ausführung in einem MIMD-Modus gegenüber dem SIMD-Modus, der eine wesentlich stärkere Synchronisierung der Prozessoren bedingt, deutliche Vorteile bringen kann.

Beschreibung	Problemgröße	SIMD	MIMD
Trapez-Regel	1048576	16.8 s	8.8 s
Reduktionsfunktion 1	1024 x 1024	6.2 s	1.0 s
Reduktionsfunktion 2	1024 x 1024	14.6 s	0.6 s
Reduktionsfunktion 3	524288	25.8 s	6.3 s
Einfache Suche	128 x 4092	43.4 s	19.2 s
Lineare Gleichungen	262144	98.7 s	- s
Lagrange Interpolation	10 x 32768	24.8 s	2.7 s
Dividierte Differenzen	131072 x 4	22.8 s	10.1 s
Finite Differenzen	512 x 512	8.4 s	1.7 s
Fourier's Momente	262144	34.2 s	15.9 s
Array-Konstruktion	1023 x 511	28.1 s	4.8 s
Floating-Point Arithmetik	262144	56.0 s	21.5 s
Integration	262144	4.9 s	1.8 s
Chebyshev Interpolation	16384	14.1 s	3.5 s

Wenn die Vektor-Einheiten benuzt werden, ergeben sich wesentlich kürzere Laufzeiten. Allerdings gehen die Vorteile von MIMD weitgehend verloren, da jeder einzelne Knoten selber im SIMD-Modus arbeitet.

3.4 Portierung größerer Anwendungen

HYDFLO ist ein dreidimensionaler Code für kompressible Hydrodynamik. Er basiert auf einem expliziten 2-Schritt Lax-Wendroff finite Differenzen Verfahren auf einem regulären Netz in einem regulären Gebiet. Der Code lief auf einer CM effizient ab und konnte erfolgreich mittels Adaptor portiert werden. Es ist prinzipiell davon auszugehen, daß datenparallele Fortran Programme, die auf SIMD-Architekturen wie CM und MasPar effizient ablaufen, nach Umstellung auf High Performance Fortran effizient auf MIMD-Architekturen ablaufen werden.

Der ESM (Earth System Model) Code ist Teil eines dreidimensionalen gitterbasierten Wettermodells. Der in Fortran 90 geschriebene Code konnte zwar erfolgreich partitioniert werden, Leistungsgewinne konnten allerdings nicht erzielt werden. Dieses ist im wesentlichen darauf zurückzuführen, daß die Datenverteilung nicht konform zu den parallelen Schleifen ist. Das folgende Programmsegment z.B. ist sehr ineffizient, da die Parallelität bzgl. der ersten Dimension des Arrays A gegeben ist, jedoch eine Verteilung des Feldes bzgl. der zweiten Dimension spezifiziert ist.

```
      real A(N,N), B(N)
!hpf$ distribute A(*,BLOCK)
!hpf$ distribute B(BLOCK)
      ....
      do J = 1, N
         A(1:N,J) = B(J) * A(1:N,J)
      end do
```

In einer Kooperation zwischen GMD und ECMWF (European Center for Medium-Range Weather Forecasts) soll der Produktionscode für mittelfristige Wettervorhersage IFS (Integrated Forecasting System) parallelisiert werden. Eine zweidimensionales Modell des IFS, das bereits alle relevanten Datenstrukturen und algorithmischen Komponenten des 3D-Modells enthält, konnte mit Hilfe von Adaptor automatisch partitioniert werden und entsprechende Leistungsgewinne erzielen. Es wurde genau die Parallelisierungsstrategie implementiert, die in einer von Hand vorgenommenen Parallelisierung verwendet worden ist [GJS93]. Obgleich für Adaptor Datenstrukturen von Hand umdefiniert werden mußten und die Laufzeiten bis zu zweimal so lang sind, konnte der Code mit wesentlich geringerem Aufwand parallelisiert werden. Die folgende Tabelle zeigt die Resultate, die für ein kleineres Problem auf einer CM 5 ohne Verwendung der Vektor-Einheiten gemessen worden sind.

Resultate auf einer CM-5 (ohne VU)	1 Knoten	2 Knoten	4 Knoten
Hand-codierte Version	5.7 s	3.1 s	1.7 s
Adaptor Version	6.8 s	4.3 s	3.1 s

Bei dem Vergleich ist allerdings mit zu berücksichtigen, daß das von Adaptor generierte Programm ca. 2.5-mal so viel Speicherplatz benötigt.

4 Zusammenfassung und Ausblick

Im Vergleich zum Message Passing Programmiermodell können mittels High Performance Fortran und dem datenparallelen Programmiermodell unter wesentlich geringerem Entwicklungsaufwand parallele Codes erstellt werden, die zudem einfacher lesbar und in der Regel auch einfacher portierbar sind.

Für eine große Anzahl von Anwendungen lassen sich damit Parallelrechner wesentlich einfacher und effektiver nutzen als bisher. Inwieweit die Leistungen an hand-geschriebene Message-Passing Programme herankommen, wird von den jeweiligen Compilern und den entsprechenden Optimierungen abhängen.

Deutliche Schwächen treten bei High Performance Fortran immer dann auf, wenn es nicht möglich ist, Lastverteilungen und Datenverteilungen konform zu spezifizieren. Dies ist bei irregulären Problemen und Datenstrukturen leider oft der Fall.

Es bleibt zu hoffen, daß in möglichst kurzer Zeit hinreichend gute Compiler zur Verfügung stehen werden. Denn es ist jetzt sehr wichtig, Erfahrungen mit HPF zu sammeln und auch die bisherigen Erfahrungen in der Parallelisierung größerer Anwendungen zu nutzen. Diese Erfahrungen sind für die Optimierung und Weiterentwicklung von HPF und dessen Compilern von enormer Bedeutung.

Literatur

[Bra93] Brandes T.: Automatic Translation of Data Parallel Programs to Message Passing Programs. Aus *International Workshop on Automatic Distributed Memory Parallelization, Automatic Data Distribution and Automatic Parallel Performance Prediction*, Saarbrücken, Maerz 1993.

[Fox91] Fox G.: Achievements and prospects for parallel computing. *Concurrency: Practice and Experience*, 3(6):725–739, Dezember 1991.

[GJS93] Gärtel U., Joppich W. und Schüller A.: Parallelizing the ECMWF's Weather Forecast Program: The 2D Case, Technical Documentation and Results for the IFS-2D Model. Arbeitspapiere der GMD 740, Gesellschaft für Mathematik und Datenverarbeitung mbH, Maerz 1993.

[Hig93a] High Perforamnce Fortran Forum: High Performance Fortran Language Specification. Final Version 1.0, Department of Computer Science, Rice University, Mai 1993.

[Hig93b] High Performance Fortran Forum: High Performance Fortran - Journal of Development. Version 1.0, Department of Computer Science, Rice University, Mai 1993.

[MFL+92] Mohamed A., Fox G., Laszewski G., Parashar M., Haupt T., Mills K., Lu Y., Lin N. und Yeh N.: Applications Benchmark Set for Fortran-D and High Performance Fortran. Technical Report 327, Northeast Parallel Architectures Center, 1992.

[Thi91] Thinking Machines Corporation: CM Fortran Programming Guide, Version 1.0. Manual, TMC, Januar 1991.