ハイパフォーマンス
コンピューティング 52－2
（1994．7．22）

HPFコンパイラの実装とAP1000を用いた評価

萩原純一　金城ショーン　土肥実久　岩下英俊　進藤達也†
(株)富士通研究所　並列処理研究センター††

分散メモリ型並列計算機のソフトウェア環境では，逐次型既存言語，特にFortranを拡張した並列用拡張言語が実用的なアプローチとされ，研究・開発が進められている．中でも，High Performance Fortran (HPF) は標準並列Fortran言語として注目されている．今回，我々はAP1000用HPFコンパイラを実装した．本実装では，VPP Fortranを中間表現とみなし，HPFプログラムをVPP Fortran相当のコンパイラ内部表現に変換する．これにより，すでに開発しているAP1000用VPP Fortranコンパイラの最適化技術や通信ライブラリを直接利用でき，HPFを効率良く実行できる．本稿では，HPFコンパイラの実装方法について述べ，HPFで書かれた行列乗算とSPECベンチマークの tomcatv の性能をAP1000上で評価する．

The implementation and evaluation of a HPF compiler for the AP1000

Junichi Hagiwara　Shaun Kaneshiro　Tsunehisa Doi
Hidetoshi Iwashita　Tatsuya Shindo
Parallel Computing Research Center, Fujitsu Laboratories Ltd.

Many Fortran-based parallel languages have been proposed for distributed memory parallel supercomputers. In this study, we implement a High Performance Fortran (HPF) compiler for the AP1000, and evaluate its performance based on several benchmark programs.

†E-mail:{jhagiw, syk, micky, hideto, shindo}@flab.fujitsu.co.jp
††211 川崎市中原区上小田中1015

1 はじめに

分散メモリ型並列計算機はハードウェアの拡張性に富むなどの利点を持っているが，ソフトウェアから見た場合，プログラミングの難しさや既存プログラムの移植に対する困難さが問題となる．そのため，現在では逐次型既存言語，特にFortranを拡張した並列用拡張言語が実用的なアプローチとされ，研究・開発が進められている [1] [2] [3] [5]．近年これらの並列用拡張言語の標準化の動きがあり，もっとも有力視されているのがHigh Performance Fortran (HPF) [1]である．

今回，我々が開発しているコンパイラ**FLoPS**上にAP1000用HPFコンパイラを実現した．本稿では，その実装方法について述べ，評価結果を示す．

以下，2章でHPFコンパイラ開発の背景を説明し，3章で，HPFコンパイラの実装方法について述べる．4章でAP1000 [4] で実行した結果を示し，その性能について評価する．5章で今後の課題を示し，6章で HPF に関連する研究について述べる．最後に 7章で本稿をまとめる．

2 HPFコンパイラ開発の背景

2.1 FLoPSプロジェクト

FLoPSは**F**ujitsu **L**aboratories **O**ptimizing and **P**arallelizing **S**ystem の略で，VPP Fortran [5] やHPF の言語拡張を含むFortranを対象とした並列化コンパイラである．**FLoPS**の構成を 図1 に示す．**FLoPS** は source to source のコンバータとして構成され，プロセッサ間通信ライブラリ呼び出しを含むSPMD型Fortranプログラムを生成する．

図1における楕円はコンパイラの各機能を実現するプログラム(以下，パスと呼ぶ)を示す．各パス間の情報の受渡しは中間形式ファイルを用いる．また，すべてのパス間において中間形式ファイルのフォーマットは同一である．そのため，中間形式ファイルを用いることでパス毎の独立性を高め，コンパイラ構成の複雑度を下げるとともに，複数の開発者や複数のプロジェクトの混在を可能とする．

我々は，図1のVPP Fortran側のパスに示す，AP1000用のSPMD型プログラムを生成するコンパイラをすでに実現している [6]．Fortran Parser は入力されたFortran プログラムをパーズし，中間形式ファイルを出力する．VPP Fortran Directive Parser はコメントの中から VPP Fortran のディレクティブを抽出し，中間形式ファイルを更新する．Code Generator は通信ライブラリ呼び出しを含むFORTRAN77プログラムを再構成する．出力されたFORTRAN77プログラムは，バックエンドのFORTRAN77コンパイラによりコンパイルされ，各対象並列計算機のオブジェクトコードとなる．

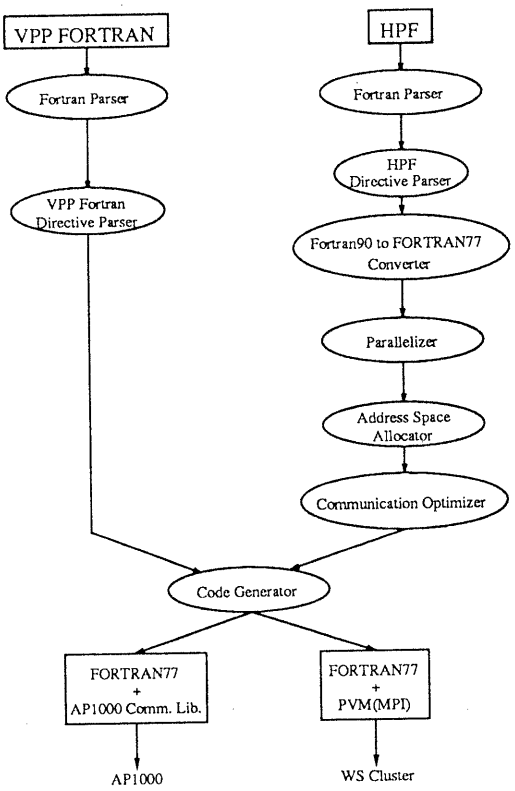

図 1: **FLoPS**の構成

今回，図1に示したHPF側のパスを追加することで，HPFコンパイラを実現した．HPFコンパイラの実装のために開発したこれらのパスの詳細については，3章で述べる．なお，今回の実装ではAP1000のみを対象とした．

2.2 中間表現としての VPP Fortran

VPP Fortranは，HPF，FortranD [2] などがサポートしているデータ分散表現や並列実行ループ指定記述に加え，以下のチューニングのための機能を提供する [5]．これらの機能はプログラマによるチューニングを可能とする．

- グローバルアドレス空間とローカルアドレス空間のサポート
- プロセッサ間通信の一括化記述

グローバルアドレス空間のサポートは，複数プロセッサにわたって分散されたメモリ空間を連続したアドレス空間として見せることで，プログラミングを容易にする．一方，ローカルアドレス空間のサポートは，プロセッサ間通信を必要としないアクセスを明示できるため，プログラムのチューニングを容易にする．さらに，グローバルアドレス空間とローカルアドレス空間とのEQUIVALENCE をとることにより，同一データを両空間で共有できる．

一方，プロセッサ間通信の一括化記述は，通信のブロック化を明示する．一般に分散メモリ型並列計算機ではプロセッサ間通信の立ち上がりのオーバヘッドが大きいため，転送するパケットを大きくし，通信をまとめることが性能を上げるために重要である．VPP Fortranではこのようなプロセッサ間通信の一括化を，send/receive のような計算機に依存した表記を用いることなく記述できる．この記述には2種類あり，1つは SPREAD MOVE と呼び，もう1つは OVERLAPFIX と呼ぶ．SPREAD MOVE は，DOループと同様の記述を用い，配列変数間の一括転送を実現する．OVERLAPFIX は，隣接プロセッサ間で共有した変数の一部(オーバラップエリアと呼ぶ [5] [3])のみの更新を意味する．

このようなチューニングのための記述はプログラマがVPP Fortran を用いて書く以外に，コンパイラによる自動最適化時の中間表現として適している．なぜなら，これらの記述は対象とする並列計算機に依存しない抽象化された表現であり，複数の並列計算機を対象とする場合の長所となるからである．そこで，今回の HPF の実装では，VPP Fortran を中間表現とみなし，HPF から VPP Fortran 相当の中間形式ファイルを生成するパスを開発した．また，このアプローチでは Code Generator をHPF側のパスと VPP Fortran側のパス間で共有でき，すでに実装しているコード生成レベルの最適化 [10] を利用できる．

3 HPFコンパイラの実装

HPFプログラムをVPP Fortran 相当の記述に変換する上で必要な機能を，以下のパスとして実装した．

1. HPF Directive Parser
2. Fortran90 to FORTRAN77 Converter
3. Parallelizer
4. Address Space Allocator
5. Communication Optimizer

以下の各節でこれらの機能を説明する．なお，例として，List1 に示す行列乗算の主要部分のプログラムを入力したときの各パスの出力を，対応する VPP Fortran で表現した†．斜体の部分がパスにより追加された記述である．

List 1 HPFによる行列乗算

```
1          PARAMETER (NPROC=64)
2          PARAMETER (L=500, M=500, N=500)
3          REAL*8 A(L,M), B(M,N), C(L,N)
4     !HPF$ PROCESSORS P(NPROC)
5     !HPF$ DISTRIBUTE A(BLOCK,*) ONTO P
6     !HPF$ DISTRIBUTE B(*,BLOCK) ONTO P
7     !HPF$ DISTRIBUTE C(BLOCK,*) ONTO P
8
9     !HPF$ INDEPENDENT, NEW(K,J)
10         DO I = 1, L
11           DO J = 1, N
12             DO K = 1, M
13               C(I,J) = C(I,J)
14        &                 + A(I,K) * B(K,J)
15             ENDDO
16           ENDDO
17         ENDDO
```

3.1 HPF Directive Parser・Fortran90 to FORTRAN77 Converter

HPF Directive Parser ではコメントの中からHPFのディレクティブを抽出し，中間形式ファイルを更新する．

また，Fortran90 to FORTRAN77 Converter は，HPF に含まれる Fortran90 の配列代入文などの機能をFORTRAN77 で実行できる形に変換する．

本パスの出力を List2 に示す．

List 2 HPFによる行列乗算(パーズ後)

```
1          ...
2          REAL*8 A(L,M), B(M,N), C(L,N)
3     !XOCL PROCESSOR P(NPROC)
4     !XOCL GLOBAL A(/(P),:)
5     !XOCL GLOBAL B(:,/(P))
6     !XOCL GLOBAL C(/(P),:)
7          ...
```

3.2 Parallelizer

Parallelizer はデータ依存解析により並列実行可能なDOループを検出する．さらに，検出されたループやINDEPENDENT ディレクティブにより並列実行指定されたループに対して，そのループを実行するプロセッサを決定する．

DOループのプロセッサへの割り付け方法はさまざまに提案されているが，今回の実装ではス

†実際には VPP Fortranではなく，対応する中間形式ファイルが出力される

プレッドーバリア実行モデル [5]を採用した．スプレッドーバリア実行モデルとは，並列実行可能なDOループに対し，各イタレーションを実行するプロセッサをあらかじめ決定し，ループ内の各ステートメント間で同期を取ることなく実行するモデルである．このモデルは Owner Computes Rule と比較して，分割されたDOループの上下限の計算に関する最適化 [3]などの必要がない．また，各ステートメント間で同期の必要がないため，高速な実行が可能である．今回の実装では，ループ内に含まれるすべての配列変数の添字式のうち，インデックス変数の一次式で表現される次元の分割形状を調べ，一番出現頻度の高い分割形状をDOループの分割形状とした．

また，データのプロセッサ間にわたる定義・参照方法は Direct Remote Data Access (DRDA) [6] を用いた．グローバルアドレス空間において，DRDAでは配列要素への定義・参照は送受信のペアによる実現ではなく，単独のリード・ライトとして実現される．そのため，プロセッサ間通信によるデッドロックはなく，どのようにDOループを分割しても正常な実行が保証される．

本パスの出力を List3 に示す．

List 3 HPFによる行列乗算(並列化後)

```
1         ...
2    !XOCL SPREAD DO /(P)
3          DO I = 1, L
4            DO J = 1, N
5              ...
6            ENDDO
7          ENDDO
8    !XOCL END SPREAD
```

3.3 Address Space Allocator

HPFでは主としてグローバルアドレス空間を用いてプログラミングする．しかし，効率良く実行するためには，ローカルアドレス空間に存在するデータに関してはグローバルな変数としてではなく，ローカルな変数として定義・参照することが重要である．このような最適化を実現するため，Address Space Allocatorでは変数に対してグローバル名とローカル名を生成し，EQUIVALENCEをとる．

本パスの出力を List4 に示す．

3.4 Communication Optimizer

Communication Optimizer では，グローバルな変数へのアクセスを削減するために，以下

List 4 HPFによる行列乗算(ローカル名生成後)

```
1          ...
2          REAL*8 A(L,M), B(M,N), C(L,N)
3          REAL*8 AL(L,M), CL(L,N)
4    !XOCL LOCAL AL(/(P),:)
5    !XOCL LOCAL CL(/(P),:)
6          EQUIVALENCE (A, AL)
7          EQUIVALENCE (C, CL)
8          ...
```

の2つの方法を実装した．

- グローバル名のローカル名への置換
- プロセッサ間データ転送の一括化

これらの最適化について説明する．

3.4.1 ローカル名への置換

グローバル名を3.3節で述べた Address Space Allocator により生成されたローカル名に置換することで，プロセッサ間通信を伴わない参照が可能となる．さらに，EQUIVALENCE されているため物理的な位置が同一であるので，ローカルな変数への定義はグローバルな変数への定義を意味する．今回の実装では，DOループの分割形状とDOループ内で定義・参照される変数の分割形状が完全に一致している場合，グローバル名をローカル名に置換する．

本パスの出力を List5 に示す．

List 5 HPFによる行列乗算(ローカル名置換後)

```
1          ...
2    !XOCL SPREAD DO /(P)
3          DO I = 1, L
4            ...
5                CL(I,K) = CL(I,K)
6          &           + AL(I,J) * B(K,J)
7            ...
8          ENDDO
9    !XOCL END SPREAD
```

3.4.2 データ転送の一括化

データ転送の一括化は，並列実行可能なDOループ内で，インデックス変数に沿って連続 または 一定間隔に定義・参照されるグローバルな配列変数に関する最適化である．データ転送の一括化の構成を以下に示す．

- ループ内のグローバルな配列変数の各要素をループ実行前に新たに生成したローカルな配列変数に代入する
- ループ内では，生成されたローカルな配列変数に対してのみ定義・参照を許す
- ローカルな配列変数への定義があった場合，その各要素をグローバルな配列変数に

書き戻す

今回の実装では，3.4.1節で述べたローカル名への置換が適用できないグローバルな配列変数すべてに対して，データ転送の一括化を適用した．

本パスを通した最終出力を List6 に示す．

List 6 HPFによる行列乗算(転送一括化後)

```
1          PARAMETER (NPROC=64)
2          PARAMETER (L=500, M=500, N=500)
3          REAL*8 A(L,M), B(M,N), C(L,N)
4          REAL*8 AL(L,M), CL(L,N)
5          REAL*8 BTMP(M,N)
6    !XOCL PROCESSOR P(NPROC)
7    !XOCL GLOBAL A(/(P),:)
8    !XOCL GLOBAL B(:,/(P))
9    !XOCL GLOBAL C(/(P),:)
10   !XOCL LOCAL  AL(/(P),:)
11   !XOCL LOCAL  CL(/(P),:)
12         EQUIVALENCE (A, AL)
13         EQUIVALENCE (C, CL)
14
15   !XOCL SPREAD MOVE
16         DO K = 1, M
17           DO J = 1, N
18             BTMP(K,J) = B(K,J)
19           ENDDO
20         ENDDO
21   !XOCL END SPREAD
22   !XOCL SPREAD DO /(P)
23         DO I = 1, L
24           DO J = 1, N
25             DO K = 1, M
26               CL(I,K) = CL(I,K)
27        &          + AL(I,J) * BTMP(K,J)
28             ENDDO
29           ENDDO
30         ENDDO
31   !XOCL END SPREAD
```

4 評価

行列乗算とSPECベンチマークの tomcatv をHPFで記述し，AP1000で実行した場合の実行時間を測定した．また，比較のため，人手により最適化された VPP Fortran プログラム [6]の実行時間も測定した．

なお，図2・図3 では，縦軸はAP1000の1プロセッサで逐次実行したときの実行時間に対する比(速度向上率)を表し，横軸はプロセッサ数を表す．また，実線は HPF プログラムの結果を示し，点線は VPP Fortran プログラムの結果を示す．

4.1 行列乗算

List1 に示した行列乗算のプログラムを実行した．実行結果を 図2 に示す．

プロセッサ台数に比例して性能が向上し，64プロセッサで約47倍の速度向上率となった．しかし，プロセッサ台数が多くなるにしたがって，人手により最適化されたVPP Fortran プログラムとの性能の差が大きくなっている．これは，人手による最適化では，一括化されたデータ転送のタイミングをプロセッサ毎にずらし，転送の競合を減らしているためである．

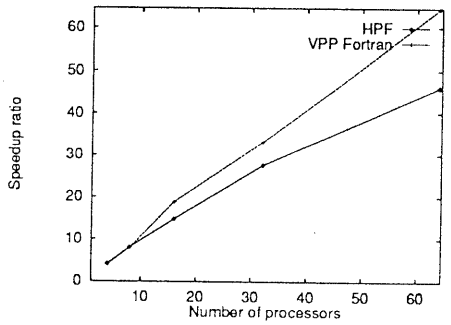

図 2: 行列乗算の実行結果

4.2 tomcatv

tomcatv では，すべての配列を2次元目でBLOCK分割した．結果を 図3に示す．

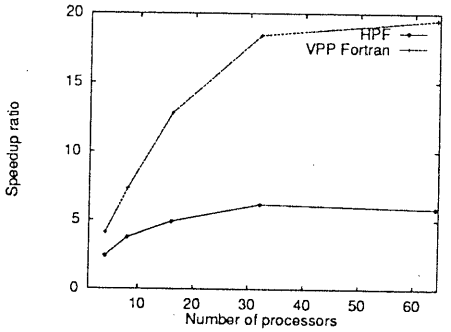

図 3: tomcatvの実行結果

全体としてHPFプログラムの性能は人手により最適化された VPP Fortranプログラムの性能よりも低く，最高でも約30%の性能 (32プロセッサのとき)であった．これは，人手による最適化ではオーバラップエリアを利用し，プロセッサ間通信を大幅に削減しているためである．現状のHPFコンパイラでは，オーバラップエリアを利用した最適化をしないため，このような大きな差が生じたと考えられる．

5 今後の課題

今回の実装では，VPP Fortranを中間表現として用いた．そのため，現在の VPP Fortran で表現できないHPFの機能は実行することができない．例えば，幅付きCYCLIC分割や動的な

データの再分散などである．今後，この制約をなくし full HPF を実現したい．

また，オーバラップエリアを利用した最適化や，転送のタイミングを考慮するなどのより高度なデータ転送の一括化を実現し，GENESIS [11] をはじめとするHPFベンチマークプログラムや，大規模実プログラムでの効率良い実行を目指す．

6 関連する研究

分散メモリ型並列計算機用の並列Fortranコンパイラに関する研究は盛んで，Fortran D [2], SUPERB [3]などが提案されている．これらのコンパイラでは，ループの実行方式は Owner Computes Rule であり，プロセッサ間データ転送には明示的な通信を必要とする．これらの方式では，データの受信待ちや冗長な同期が多い．また，インデックス配列が存在した場合に性能が著しく低下することが知られている．一方，**FLoPS**ではループの実行方式にスプレッドーバリア実行方式を採用し，プロセッサ間データ転送にはDRDAによるリード・ライトを用いることで，これらの問題を解決している．

また，HPF コンパイラの実装に関しては，Applied Parallel Research の xhpf, Pacific-Sierra Research の VAST/HPF, The Portland Group のシステム [7]，日本IBMの HPFコンパイラ [8]，Bee-Fortran [9] などが報告されている．これらの HPFコンパイラでは，データ転送の一括化をはじめとするプロセッサ間通信最適化の重要性が指摘されているが，詳細に関する検討は明らかではない．一方，我々はグローバル名の置換とデータ転送の一括化を**FLoPS**に実装し，さらに，オーバラップエリアによる最適化の重要性を指摘している．

7 まとめ

本稿では，まず，我々の開発しているコンパイラ **FLoPS**について説明した．次に，**FLoPS**におけるHPFコンパイラの実装方法について述べ，出力コードの性能を評価した．

HPFコンパイラの実装では，VPP Fortran を中間表現とみなし，HPFプログラムをVPP Fortran相当の中間形式ファイルに変換するパスを開発することで，HPFのAP1000上での実行を可能にした．

また，行列乗算とtomcatvをHPFで記述し，AP1000上で実行した．行列乗算では最適化の効果は良好で，プロセッサ台数が少ない場合には人手により最適化された VPP Fortran プログラムとほぼ同等の性能を出すことができた．一方，tomcatv では，最高でも人手により最適化された VPP Fortran プログラムの約30%の性能であった．これは，現在のHPFコンパイラではオーバラップエリアを用いた最適化ができないためである．

8 謝辞

本研究の遂行に当たり，御指導頂きました石井光雄氏，白石博氏，佐藤弘幸氏，池坂守夫氏に感謝いたします．

参考文献

[1] High Performance Fortran Language Specification Version 1.0, High Performance Fortran Forum, May. 1993.

[2] S. Hiranandani, K. Kennedy, and C. Tseng, "Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines," in Proc. of Supercomputing'91 pp.86-100, Nov. 1991.

[3] H. Gerndt and H. Zima, "OPTIMIZING COMMUNICATION IN SUPERB," in Proc. of CONPAR'90, LNCS457, pp.300-311, Sept. 1990.

[4] 石畑, 稲野, 堀江, 清水, 池坂: "高並列計算機AP1000のアーキテクチャ", 信学論(D-I),J75-D-I, No.8, pp.637-645, Aug. 1992.

[5] 岩下, 進藤, 岡田: "VPP Fortran: 分散メモリ型並列計算機言語", 並列処理シンポジウム JSPP'94, pp.153-160, May. 1994.

[6] 進藤, 岩下, 土肥, 荻原: "AP1000を対象としたVPP Fortran処理系の実現と評価", 情処研報 HPC93-48, pp.9-16, Aug. 1993.

[7] R. Babb II, A. Choudhary, L. Meadows, S. Nakamoto and V. Schuster, "Retargetable High Performance Fortran Compiler Challenges," in Proc. of CompCon'93, pp.137-146, Feb. 1993.

[8] T. Nakatani, "Compiling HPF for A Cluster of Workstations,"並列処理シンポジウム JSPP'93, pp.1-6, May. 1993.

[9] 小前, 杉森, 大谷, 渦原, 安村: "SIMD 型超並列計算機用HPF の実装とその性能評価", 信学技報 SS93-46, pp.9-16, Mar. 1994.

[10] 土肥, 林, 進藤: "ストライドデータ転送機構を用いたコード生成", SWoPP'94(発表予定)

[11] V. Getov, I. Glendinning, A. Hey, R. Hockney and I. Wolton, "The GENESIS Distributed-Memory Benchmark Suite Release 2.3," University of Southampton, Apr. 1994.