ハイパフォーマンス
コンピューティング 57−20
(1995. 8. 25)

HPF処理系の実現と評価

郷田 修, 大澤 暁, 小松秀昭, 菅沼俊夫, 小笠原武史, 石崎一明, 中谷登志男

日本アイ・ビー・エム(株)東京基礎研究所

本論文ではIBM RS/6000のためのHPF(High Performance Fortran)の処理系について述べる。本処理系はHPFプログラムを入力し,SPMDプログラムに変換するコンパイラと,実行時にプロセッサ間の通信を行う実行時ライブラリからなる。コンパイラはHPFの標準サブセットをサポートし、Fortran 77ループを自動的に並列化する。また,プロセッサ数,配列サイズ等が実行時に決まる場合につても効率の良いコードを生成する。ベンチマーク・プログラムの実行結果はコンパイラのコード生成が効果的であることを示している。

Design, Implementation, and Evaluation of an HPF Compiler

Osamu Gohda, Gyo Ohsawa, Hideaki Koamtsu, Toshio Suganuma, Takeshi Ogasawara, Kazuaki Ishizaki, Toshio Nakatani

IBM Japan, Tokyo Researcg Laboratory

This paper presents the implementaion and evaluation of an HPF compiler for IBM RS/6000. The compiling system is composed of a compiler and a runtime libarary: the compiler accepts HPF programs as input and translates them into SPMD programs, and the runtime library performs inter-processor communications at runtime. The compiler supports the officail subset definde by the HPF language specification, and automatically paralelizes Fortran 77 loops. The compiler generates efficient code even in the cases where the number of processors or the array size are unknown at compile-time. The benchmark result shows the effectiveness of the compiler.

1 はじめに

近年, 分散メモリマシンの普及にともない, 標準プログラミング言語として HPF (High Performance Fortran) [1] が注目を集めている。現在, 各種の処理系が開発され一部は商用に供されている [3] [6]。我々は言語仕様がドラフトの時点から HPF が分散メモリマシン環境での標準言語になると考え,IBM RS/6000 SP(SP2) 上での並列プログラミング環境の一部として, 処理系の開発を進めてきた [2]。処理系の開発にあたっての我々の目標は以下のとおりである。

標準サブセットのサポート ポータビリティを考慮して HPF 言語で定義されたサブセット（標準サブセット）をサポートすることとし, 独自の言語仕様の拡張は行わない。

F77 ループの並列化 IF 文を含むループ, 複数の代入文を含むループ, IF 文と GOTO 文で構成されたループ等, 並列化の困難なループの並列化を行う。そのために, 依存性解析, ループ検出, インダクション変数の検出機能等を活用する。

現在, 処理系は標準サブセットのすべての機能をサポートする。また, 配列演算, Forall といった明示的に並列性を記述する文ばかりでなく,Fortran 77 の DO 文や IF 文により構成されるループも並列化するとともに, プロセッサ数, 配列サイズ等が実行時に変化する動的な場合についても並列化を行なう。

2 処理系の構成

本コンパイラは IBM Risc System/6000 用の Fortran コンパイラに, 図 1に示すように SPMDizer と呼ぶコンポーネントを追加する形で実現されている。SPMDizer はフロントエンドが生成する中間語を入力して SPMD 形式のノード・プログラムに変換する。変換された中間語は既存のバックエンドに供給され目的プログラムとなる。なお, 本処理系の計算分割の方法は基本的には owner-computes rule による。

3 プログラムの解析／変換

プログラム解析では依存性解析, ループ解析といったプログラムの基本情報を収集するとともに, ループ並列化および通信解析をより効果的に行なうためのデータの収集およびプログラムの変換を行なう。

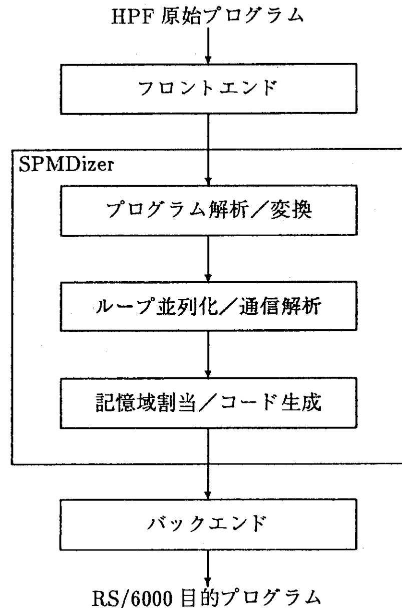

図 1: HPF コンパイラの構成

プログラムの解析および変換では以下の処理を行なう。

1. 依存性解析
2. ループ解析
3. 定数プロパゲーション
4. インダクション変数の検出
5. プライベート変数の検出
6. ループ・ディストリビューション
7. リダクション・ループの検出

このうちプライベート変数の検出, ループ分割は,HPF プログラムでは逐次プログラムの場合と比較して重要度が高い。プライベート変数の検出では, これによりループ繰り返し間の依存関係をなくし並列化の機会を増大させることができる。また, ループ・ディストリ

ビューションによりループ中の代入文の数が減少し並列化の機会が増大する。

4 ループの並列化と通信解析

ループの並列化および通信の解析はプログラム解析によって得られる情報およびHPFディレクティブで指定される配列の分割情報をもとにして行い, ループの分類および各オペランドに必要な通信の分類を行う。なお, これらについて本稿では紙面の都合から詳しくは述べないが (詳しくは [4] [7] を参照), 以下にループと通信がどのように分類されるかを示す。

ループの分類 ループは, その中に含まれるオペランドの形式や, オペランド間の依存関係によりつぎの4種類に分けられる。

dopar: 各プロセッサが独立して並列に実行可能なループ。

dopipe: 独立して並列実行はできないが, パイプライン実行が可能なループ。

doreduce: リダクション演算を行うループ.

dosync: 上記以外のループ。

通信の分類 配列要素の参照などの各オペレンドは以下の4書類に分けられる。

prefetch: ループ中で参照されるオペランドで, ループを開始する前に他のプロセッサから取得しておくことが可能なオペランド。

pipline: パイプライン実行ループ中でループ繰り返しにまたがる依存関係を持つオペランド。

sync: prefetch, pipeline 以外の通信を必要とするオペランド

no comm: 実行にあたって, 通信を必要としないオペランド

ループとオペランドの分類の例 ループとオペランドがどのように分類されるかをいくつかの例を使って示す。

```
      real a(100), b(100), c(100)
*hpf$ processors p(10)
*hpf$ distribute (block) onto p :: a, b, c
      do i=1,99
        a(i) = b(i) + c(i+1)
      end
```

この例では, ループ繰り返し間の依存関係がないためループは dopar となる。オペランドについては, 右辺の第一オペランド b(i) は左辺 a(i) と同じプロセッサーにあるため通信は必要なく, no comm となる。c(i+1) については通信は必要とするが, 必要な要素をループ開始前に前もって取得しておくことができるので prefetch となる。

```
      real a(100), b(100)
*hpf$ processors p(10)
*hpf$ distribute (block) onto p :: a, b
      do i=2,100
        a(i) = a(i-1) + b(i-1)
      end
```

この例では, ループは a についてのループ繰り返し間の依存関係があるため dopipe となる。また、a(i-1) は同様の理由から pipeline と分類され, b(i-1) は prefetch と分類される。

```
      real a(100), m(100)
*hpf$ processors p(10)
*hpf$ distribute (block) onto p :: a, b, c
      do i=1,100
        a(m(i)) = a(i)
      end
```

この例では代入文の左辺に間接参照が使用されているため, コンパイル時には並列実行可能かどうかの決定できず, ループは dosync となる。

4.1 コンパイル時および実行時データ構造

本処理系の特徴として, コンパイル時と実行時で共通のデータ構造とそのデータ構造に対するサポートルーチンを使用していることがあげられる。一般に, コンパイラはループの上下限や添字といったパラメータがコンパイル時に既知（静的）である場合と, そうでない（動的な）場合とでは非常に異なったコードを生成する。これは静的な場合と動的な場合ではコード生成の方式がまったくことなることを意味し, コンパイラの負担が増加する。本処理系では配列の分割情報, ループの上下限情報, 通信領域の情報といった基本的なデータ構造を定め, データ構造の値をコンパイル時に（静的に）計算するか, 実行時に（動的に）計算するかという点を除いて, 原則として同一の目的コードを生成する。こうすることにより, Fortran D コンパイラ [5] のようにプロセッサ数, ループの上下限が定数でなければならないといった制限を避けることができた。

5 記憶域の割当とコード生成

プログラム解析の結果に基づいて記憶域の割当, SPMD コードの生成を行う。

5.1 記憶域の割当

主に配列（オーバーラップ領域を含む）およびバッファ領域のサイズの決定, それらの割り当てを行う。

HPF では, 並列ループ中またはパイライン実行可能なループ中で参照されるオペランドの一部を, 前もって他のプロセッサから取得しておく必要があるが, 取得したデータを置く場所が問題になる。これには, 一般にオーバーラップ領域を使う場合とバッファ領域を使う場合があり, これをどのように使い分けるかが問題になる。

すべてをオーバーラップ領域を使うようにすれば自プロセッサの持ち分を複写する必要がなくなり実行効率は良くなるが, 最悪の場合すべてのプロセッサが配列の全域を持つ必要があり HPF の特徴である記憶域を複数プロセッサに分散できるという利点を失う。また, すべてバッファを使うと記憶域の使用量の点ではよいが, 逆に自プロセッサの持ち分をバッファに複写する必要が出てくる。

本処理系ではこれを通信解析の結果に基づいて次のように行う。

1. 各配列参照について, 通信解析の結果をもとに, 他のプロセッサからプリフェッチしなければならない範囲をもとめる。

2. 前ステップで求めた範囲がオーバーラップ領域の許容範囲内であれば, その配列参照については, 他のプロセッサからのデータをオーバーラップ領域にプリフェッチし参照する。

3. オーバーラップ領域を使用できない場合は, その配列参照についてはバッファ領域を割り当てる。

4. バッファ領域を使用するオペランドが pipeline 通信の場合には, 対応する左辺のオペランドについても, そのバッファ領域 (左辺バッファ) を使用するようにする。

記憶域割当の例として以下のプログラムを考える。

```
      real a(100,100),b(100, 100),
           c(100,100)
!hpf$ processors p(10,10)
!hpf$ distribute (block,block) onto p ::
     *    a, b, c
      do j=1,99
        do i=1,100
          a(i,j)=b(i,j+1)+c(i,100)
        end do
      end do
```

この例では, b については必要なオーバーラップ領域の幅は 1 なので配列 b に割り当てられたオーバーラップ領域が使用される。一方, c については, もしオーバーラップ領域を使用するとすると, プロセッサ 1 では必要なオーバーラップ領域の幅が 90 となってしまうので, そのかわりにバッファーが使用される。

5.2 コード生成

プログラム解析, 記憶域割当の結果をもとにずいて, コード生成を行なう。実際には, SPMD 化される前, すなわち逐次プログラムの中間語に対して以下の操作を行なう。

1. 通信の挿入

2. ループの上下限およびステップの局所化

3. 配列参照中のサブスクリプトの局所化

4. 副プログラム境界における配列の再分割

6 その他の最適化

本コンパイラではこれまでに述べた以外にも, さまざまな最適化を行っている。ここでは, そのうちの主なものについて述べる。

6.1 実行時オーバーヘッドの削減

前に述べたように, 本コンパイラではコンパイル時と実行時に共通のデータ構造を使用しており, コンパイル時に解決できなかった値については実行時に計算するようになっている。しかし, これらの計算の中にはかなり複雑な計算を必要とするものがある。本処理系では, 計算結果をキャッシングすることにより, 前回と同じものについては再計算を行わないようにしている [8]。

6.2 通信の最適化

HPF の処理系では通信をいかに効率的に行うか, また通信回数と通信量をいかに少なくするかが, 目的プログラムの効率の点から非常に重要である。このため, 本処理系では以下のような通信についての最適化を行っている。

メッセージ・ベクトル化 通信解析で得られた情報をもとに, ループ中で必要な通信をループ開始前に一括して行う。

メッセージ融合 同一または共通部分を参照する複数のオペランドに対するメッセージを一つのメッセージにまとめる。

メッセージ構造化 通信回数を減らすため，同一プロセッサに送られる複数のオペランドを同一パケットに入れる。

6.3 スカラ変数に関する最適化

スカラ変数を含むループについても以下の最適化を行なう。

リダクション・ループ 総和を求めるといったリダクション・ループをプロセッサの持ち分に対する演算を行うループと，その結果を縮約するメッセージ交換ライブラリへの呼び出しに変換する。

スカラへの代入を含むループ 本処理系では，スカラ変数は全プロセッサが重複してもつので，ループ中のスカラへの代入文の右辺に配列参照があると，参照される範囲をすべてのプロセッサに送る必要が生じる。これを避けるため，ループを開始する前にスカラ所有者を1台だけにし，ループ終了後結果をすべてのプロセッサに送ることにより，全プロセッサ所有に戻している。

6.4 ループ・スカラライズ

Fortran 90 の配列代入等に対して生成されるテンポラリへの代入を，依存関係が発生しない場合は，左辺への直接代入に置き換える。また，テンポラリへの代入を削除した場合に依存関係発生する場合でも，ループの繰り返し順序を反転することにより依存関係が取り除けるものについては同様の処理を行なう。

7 RS/6000 SP(SP2) による評価

本処理系の評価を，分散メモリ型並列計算機 SP2 を使って行った。使用したプログラムは Applied Pararell Reserch 社の作成した HPF ベンチマークから選んだ，tomcatv, shallow77 , grid, x42 の 4 つで，配列の分割は grid が (block,block,*)，その他は (*,block) である。図 2は本処理系でコンパイルした場合の実行時間 (Tp) と，同じプログラムを逐次コンパイラでコンパイルした場合の実行時間 (Ts) を比較したもので，横軸にプロセッサ数 (p)，縦軸にスピードアップ (Sp=Tp/Ts) を示した。また，図 3は shallow 77 について，同じプログラムをメッセージ通信ライブラリ (EUI) をつかった手書きプログラムと比較したものである。

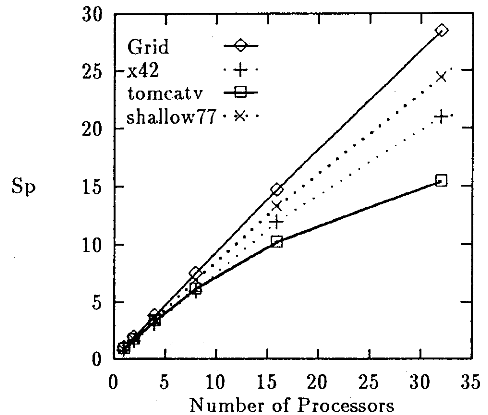

図 2: 逐次プログラムとの比較

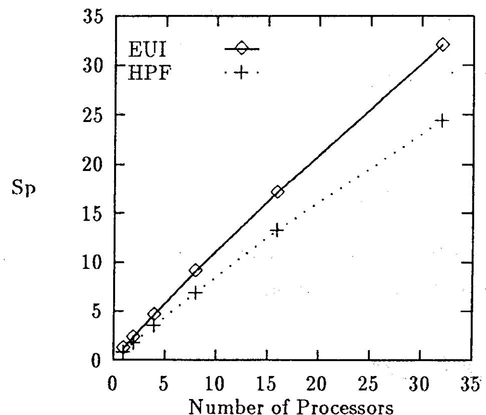

図 3: 手書きプログラムとの比較

8 関連研究

Rice 大学で開発された Fortran D コンパイラは分散型並列メモリ用自動並列化コンパイラの初期のものである。通信のベクトル化等，さまざまな最適化を行なっているが，プロセッサ数の一次元の分散のみをサポートし，プロセッサ数もコンパイル時定数でなければならない。HPF の処理系についてもいくつか研究

結果が報告されている。ADAPTOR [9] と PGI のコンパイラ [3] は配列代入, Forall 等の配列化を行なうが,Fortran 77 ループの並列化は行なわない。言語仕様的には HPF のすべての機能をサポートする処理系は報告されておらず, 大部分は HPF 標準サブセットまたはその一部をサポートしている。また, コンパイラの並列化機能を補うために, APR [6] や NEC [10] の処理系のように独自のディレクティブや文を追加しているものも多い。

9 まとめ

本論文では, SP2 のための HPF 処理系の実現とベンチマーク・プログラムによる評価結果について述べた。コンパイラは HPF 標準サブセットのすべての仕様をサポートしており高いポータビリティを有する。また, 依存解析, ループ解析, リダクション・ループの検出等を利用した並列化機能を有し,Fortran 77 のループの並列化を行なう。配列の大きさやプロセッサ数等が実行時に決まる場合についても, コンパイル時と実行時に共通のデータ構造を使用し, コンパイル時にこれらの値が決まっている場合と同様な効率の良いコードを生成する。ベンチマークの結果は本処理系の並列化および最適化機能が有効であることを示している。

10 謝辞

本研究をサポートして頂いた IBM 東京基礎研究所 CSI 担当上村部長に感謝します。

参考文献

[1] High Performance Fortran Forum, "High Performance Fortran Language Specification (Version 1.0)," Rice University, May 3, 1993.

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

[3] Z. Bozkus, L. Meadows, S. Nakamoto, and V. Schuster, "Retergettable HPF Compiler Interface," Proceedings of Forth International Workshop on Compilers for Parallel Computers, pp.494-502, December 1993.

[4] 石崎一明, 小松秀昭, "分散並列計算機のための並列性抽出法," 信学技報, CPSY94-39, pp.97-104, 1994.

[5] C. W. Tseng, "An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines," PhD thesis, Rice University.

[6] Appried Parallel Research "Forge 90 Distributed Memory Parallelizer," User's Guide,Placerville, CA, USA, 1992.

[7] 石崎一明, 小松秀昭,"HPF コンパイラにおける並列化手法," SWoPP '95 ハイパーフォーマンス研究会 (発表予定).

[8] 小笠原武史, 石崎一明, 小松秀昭,"HPF における実行時通信解析オーバーヘッドの削減手法," SWoPP '95 ハイパーフォーマンス研究会 (発表予定).

[9] T. Brandes, "ADAPTOR Language Reference Manual, Version 2.0," German National Research Center for Computer Science, Schloss Birlinghven, Germany, 1994.

[10] 末広 他,"HPF 処理系における計算マッピング," 並列処理シンポジウム JSPP '95, July 1995.