2004－HPC－99　(46)
2004／8／1

SPEC OMP の HPF 化に関する研究

森井 宏幸*，坂上 仁志**，新居 学**，高橋 豐**
*姫路工業大学　大学院工学研究科　電気系工学専攻, **兵庫県立大学　大学院工学研究科　電気系工学専攻

HPF を使うとデータ分散を指示するだけでコンパイラが処理を分割し，データ通信するプログラムを自動生成するので並列処理を容易に実現することができる．しかし，単純に指示文を挿入するだけでは，並列化できなかったり並列化できても効率が悪い場合があるため，その問題を改善する方法について研究を行ってきた．研究対象の一つに引数の次元数不一致問題があり，動的配列の確保，解放およびコピーにより問題を回避することで並列化する方法を提案したが並列化の効率が悪かった．そこで本稿では，新しい解決法として間接的アドレス参照法と大域的な並列化よる並列化法を検討し評価した．

Evaluation of Parallel Performance with HPF

Hiroyuki MORII*, Hitoshi SAKAGAMI**, Manabu MANABU** and Yutaka TAKAHASHI**
*Division of Computer Engineering, Graduate school of Engineering, Himeji Institute of Technology
**Division of Computer Engineering, Graduate school of Engineering, University of Hyogo

Once user explicitly specifies data distribution with directives, the HPF compiler divides a job into many tasks and allocates each task to different processors, and automatically generates data transfer codes. Thus user can easily parallelize their programs with HPF. We have rewriten some SPEC OMP 2001 benchmark programs with HPF. In this paper, we have rewrite one of the programs again with another methods to improve parallel efficiency and evaluated parallel performance and programming applicability.

1．はじめに

現在，実用的な並列プログラミング環境として MPI[1,2]，OpenMP[3]，HPF[4-6] が存在する．MPI(Message Passing Interface) は，多くの異なったアーキテクチャの並列計算機で利用可能であるが，ユーザがプロセッサ間のデータ転送をプログラム実行の流れを意識した上で明示的に記述する必要があり，一般ユーザには扱い難い．一方，OpenMP は一般ユーザーにも扱い易いように，従来の逐次プログラムに最小限の付加的な指示文を挿入することでプログラムの並列実行を可能とする記述性の高い指示文ベース処理を用いているが，共有メモリ型の並列計算機でしか利用できない．

これに対して，HPF は OpenMP と同様に指示文ベース処理を用いたデータ並列言語であり，一般ユーザにも扱い易く，共有メモリ型の並列計算機だけではなく分散メモリ型の並列計算機でも利用できる．そのため，HPF は今後並列計算機を一般ユーザでも容易に扱えるツールとするために必要なプログラミング言語と言える．

そこで，HPF と同様に扱い易い指示文ベースである OpenMP で記述された，共有メモリ型並列計算機システムの性能を評価するためのベンチマークプログラムである SPEC OMP2001[7] を HPF に書き換え，並列性能を比較する研究を行っている．本稿では以前に HPF により並列化を行ったプログラムである MGRID の従来法による並列化法の問題点を示してその解決法を検討した．なお，実行環境には大阪大学サイバーメディアセンターの NEC SX-5 と地球シミュレータセンターの PC クラスタを用いた．

2．MGRID の並列化

MGRID はマルチグリッドで 3 次元のポテンシャル場を計算するプログラムである．

MGRID では，大きな 1 次元配列を主プログラムで用意し，副プログラムではその 1 次元配列の一部を 3 次元配列として用いている．しかし，HPF は分散配列を副プログラムに渡すときに次元数変換す

ることには対応していない．そのため，MGRID は HPF の指示文を追加するだけでは並列化できなかった．この問題を引数の次元数不一致問題と呼ぶ．

引数の次元数不一致問題は，従来法では動的配列の確保，解放とコピーによって回避したために OpenMP ほどの性能は得られていない [8]．そこで，よりよい解決法，簡単なソース書き換えにより問題を回避する方法の検討を行った．

２．１　間接的アドレス参照法

従来法の問題点である動的配列の確保・解放とコピーを避けるため，主プログラムにおいて必要な配列を副プログラム側で用いる次元数であらかじめ定義し分散しておくことが考えられる．しかし，単純にそのように書き換えを行おうとすると以下に示すように，副プログラムの呼出し時に if 文によりインデックスの値に対応させて配列を変更することで呼び分ける必要がある．このため，大きくプログラムを書き換える必要があり，煩雑な方法である．

```
C      dimension a(M),is(k),il(k)
      dimension a1(n1,n1,n1),...,ak(nk,nk,nk),il(k)
!HPF$ PROCESSORS proc(NP)
!HPF$ DISTRIBUTE (*,*,BLOCK) ONTO proc :: a1,a2,..,ak
      •••
      do i = 1 , k
C        call sub(a(is(i)),il(i))
        if(i .eq. 1)then
          call sub(a1,il(1))
        else if(i .eq. 2)then
          call sub(a2,il(2))
        •••
        else if(i .eq. k)then
          call sub(ak,il(k))
        endif
      enddo
      •••
      stop
      end

      subroutine sub(a,m)
      dimension a(m,m,m)
      ••• (計算)
      return
      end
```

そこで，改善案として Intel Fortran Compiler（以下，ifc）の loc() と%val() を用いて以下のように書き換えることで DO ループを展開することなく，動的配列の確保・解放，データのコピーなしに並列化を行う方法を考えた．loc は引数のアドレスを返す関数で，%val は FORTRAN でアドレス参照ではなくデータ参照による副プログラム呼び出しを行うために用いる関数である．

これらの関数により larg=loc(arg) で arg のアドレスを larg に代入し，%val(larg) で larg の値を参照することで arg のアドレスを手続きに渡すことができる．これを応用することで if 文による呼び

分けを行わなくても元のプログラムと同様にアドレスを渡すことを実現している．この方法を，間接的アドレス引用法（ Indirect Address Quoting Method : IAQM）と呼ぶ[9].

```
C      dimension a(M),is(k),il(k)
       dimension a1(n1,n1,n1),...,ak(nk,nk,nk),il(k)
!HPF$ PROCESSORS proc(NP)
!HPF$ DISTRIBUTE (*,*,BLOCK) ONTO proc :: a1,a2,...ak
       dimension 1a(k)
       la(1)=loc(a1)
       •••                    (追加)
       la(k)=loc(ak)
       do i = 1 , k
C         call sub(a(is(i)),il(i))
          call sub(%val(la(i)),il(i))
       enddo
       •••
       stop
       end

       subroutine sub(a,m)
       dimension a(m,m,m)
       ••• (計算)
       return
       end
```

HPF では引数としてデータを渡す場合，プリコンパイル時に引数に関する情報が新たに引数として追加される．データ分散されてない変数を引数として渡す場合は，データの型に関する情報が引数として追加される．データ分散された配列を引数として渡す場合は，型だけでなく配列のデータ分散に関する情報も記憶している変数が引数として追加される．しかし，上記の書き換えでは副プログラムに分散されている配列を渡すのではなくアドレス値を渡しているため，データ分散についての情報ではなくアドレスを格納している変数の型に関する情報を渡すようにプリコンパイルされる．そこで，コンパイラの代わりにプリコンパイル後にデータ分散の情報を副プログラムに渡すように修正を加える必要がある．配列 X の分散に関する情報は X$sd1 に記憶されているためにデータの型を渡している pghpf_type の代わりに X$sd1 を渡せばよいが，直接渡すと if 文による呼び分けが必要となるためにプリコンパイル前の修正と同様に loc() と%val() を用いて以下のように書き換えを行った．

```
       dimension la2(k)
       la2(1) = loc(a1$sd1)
       la2(2) = loc(a2$sd1)   (追加)
       •••
       la2(k) = loc(ak$sd1)
       do i = 1, k
C         call sub(%val(la(i)),il(i),pghpf_type(25),pghpf_type(25))
          call sub(%val(la(i)),il(i),%val(la2(i)),pghpf_type(25))
       enddo
```

また，SX-5 の FORTRAN コンパイラでは，上記の loc()，%val() 機能が実装されていないために地球

シミュレータセンターにあるPCクラスタを用いてプログラムの実行を行った．IAQMを用いたプログラムと従来法の実行時間と高速化率を表1に示す．

表1 IAQMの実行結果

		実行時間 [sec]		高速化率	
		従来法	IAQM	従来法	IAQM
プロセッサ数	1	14960.9	6841.4	1.00	1.00
	2	8252.9	4265.3	1.81	1.60
	4	4458.3	2387.7	3.36	2.87
	8	2511.3	1443.0	5.97	4.74
	16	1682.1	1026.5	8.89	6.66

表1より，IAQMを用いることで約15000秒から約7000秒と大幅に性能を改善することができた．現在は，どのコンパイラでも対応できる方法ではないが，有効な手段であるのでコンパイラに実装されるべきだと考えられる．加速率においては従来法の方が高い性能を出しているが，これは動的配列の確保にかかる時間が，プロセッサ数が増えたことにより１プロセッサあたり確保する領域が減ることで確保にかかる時間が減少する影響が大きい．

２．２　大域的な並列化

これまでの並列化の方針ではOpenMPと比較し易いようにOpenMPで局所的に並列されている部分を並列化してきたが，MGRIDでは性能が得られなかった．また，前節のIAQMはifcに依存しているために利用できないシステム環境もある．加えてプリコンパイル後のソースに手を加える必要があり，通常はこのような並列化は行わない．そこで，これまでの方針とは異なりOpenMPの並列化を参考にすることなく大域的に並列化を行う方法を考えた．

MGRIDでは，ndata個の入力データを入力して以下のように各データに対して処理を行う部分があり計算時間の大部分を占めていた．

```
      dimension u(M),v(M),r(M)
!  前処理
     ・・・
      do i = 1 , ndata
! 各データに対する計算部分
        call zero30(u,v,n,n,n)
        call norm2u3(r, n, n, n, j1(i), j2(i))
        ・・・
      enddo
```

そこで，１データに対して行われる処理をまとめて一つのサブルーチンとし，PURE手続きを用いて並列呼出しを行うように書き換えた．これにより，これまで各データに対する処理部分を並列化するときに必要であったデータ転送や動的配列と静的配列間の複写を取り除くことができ効率よく並列化を行える．

```
      dimension u(M),v(M),r(M)
      allocatable j1(:),j2(:),...
!HPF$ PROCESSORS proc(NP)
!HPF$ DISTRIBUTE (BLOCK) ONTO proc :: j1,j2,...
       ・・・
      interface
        pure extrinsic('FORTRAN','LOCAL')subroutine
     &            puresub(u,v,r,n,xx,...,j1,j2,...)
      dimension u(:),v(:),r(:),...
       ・・・
        end subroutine
      end interface

!  前処理
      ・・・
!HPF$ INDEPENDENT, NEW(u,v,r,...
      do i = 1 , ndata
!HPFJ ON HOME(j1(i)) , LOCAL BEGIN
        call puresub(u,v,r,...,j1(i),j2(i),...)
!HPFJ END ON
      enddo
       ・・・

      subroutine puresub(u,v,r,...,j1,j2,...)
       integer j1,j2
       ・・・
! 各データに対する計算部分
        call zero30(u,v,n,n,n)
        call norm2u3(r, n, n, n, j1, j2)
        ・・・
```

しかし，並列呼び出し部分では主要な計算領域である u,v,r を NEW 変数として宣言し，各プロセッサ毎にメモリを確保する必要がある．このため，本来なら一組で済む u,v,r がプロセッサ毎に独立して確保されるために全体として大きなメモリを必要とする．

表２に大域的に並列化を行った MGRID の SX を用いた実行時間と高速化率を示す．表２の結果より，大域的な並列化により多くのメモリを必要とするが効率の良い並列化を行えたことが分る．

表 2 MGIRD 分散処理の計測結果

		実行時間 [秒]		高速化率	
		従来法	大域的	従来法	大域的
プロセッサ数	1	285.3	275.2	1.00	1.00
	2	216.2	149.7	1.32	1.84
	3	176.1	105.5	1.62	2.61
	4	160.9	76.0	1.77	3.62
	8	138.8	44.2	2.06	6.22

4. まとめ

SPEC OMP2001 ベンチマーク中の MGRID の従来法による並列化では，引数の次元数不一致問題に対応するために必要な次元数の配列を動的に確保し，その配列にデータを複写することで回避できたが高い高速化率を得られなかった．その改善案として ifc の loc() と%val() という関数を用いる IAQM によりプログラムに修正を加えることで配列の動的確保・解放とデータのコピー，DO ループを展開を行うことなく並列化できた．また，各データに対する処理部分を部分的に取り出し，ピュアな副プログラムとして並列呼出しするように書き換えることで大域的な並列化を実現した．しかし，大域的並列化を行うと必要なメモリが膨大になる問題があるが，従来法に比べ高い高速化率を得ることができた．

謝辞

本研究の実施において，並列計算機 NEC SX-5 の利用環境を提供していただいた大阪大学サイバーメディアセンターと PC クラスタの利用環境を提供していただいた地球シミュレータセンターに謝意を表する．また，地球シミュレータセンターの村井均氏の協力に謝意を表する．

参考文献

[1]Message Passing Interface Forum:A Massage-Passing Interface Standard, Int. J. Supercomputing. Applications and High Performance Computing, Vol.8, No.3/4, pp.165-416 (1994).
[2]Message Passing Interface Forum:Extensions to the Message-Passing Interface (1997).
[3]OpenMP Architecture Review Board:“OpenMP Fortran Application Program Interface”，富士通株式会社 訳，(1999).
[4]High Performance Fortran Forum:High Performance Fortran language specification, version2.0 (1996).
[5]High Performance Fortran Forum:High Performance Fortran2.0 公式マニュアル，シュプリンガー・フェアラーク東京株式会社，(1999).
[6]http://www.hpfpc.org/jahpf/.
[7]http://www.specbench.org/.
[8] 森井宏幸，坂上仁志，新居学，高橋豊:HPF の性能評価と応用に関する研究，情報処理学会研究報告 2003-HPC-95，情報処理学会，pp.143-148 (2003).
[9] 森井宏幸，坂上仁志，新居学，高橋豊:OpenMP ベンチマークプログラムの HPF 化と並列化に関する研究，姫路工業大学大学院工学研究科研究報告 No56，姫路工業大学 (2003).