High Performance Fortranによる並列プログラミング

核融合科学研究所
基礎物理シミュレーション研究系
坂上仁志
sakagami.hitoshi@nifs.ac.jp

プロセッサ性能向上の壁*

- 命令レベル並列性の壁
  - SSE命令,アウト・オブ・オーダー実行,分岐予測,投機的実行等をプロセッサに実装してきた.
  - 命令依存やデータ依存のため,効果が非常に限定的であった.
- 消費電力の壁
  - 熱密度の点から,消費電力は限界近くに達している.
  - 動作周波数を高くするために電圧を上げることはできない.
- メモリの壁
  - メモリのアクセス速度の向上は,プロセッサの演算性能の向上よりずっと遅く,両者の乖離は著しい.
  - キャッシュだけでは,その溝を埋めることができない.

*B. J. Smith, Int. Supercomputing Conf. 07, keynote, Dresden, Germany, June 26-29 (2007).

並列プログラミングの必要性

- 命令レベル並列性の壁
  - スレッドレベルやプロセスレベルの並列性が重要である .
- 消費電力の壁
  - 動作周波数を下げてマルチコア化することにより,全体の演算性能は維持しつつ消費電力を低下させる.
- メモリの壁
  - 高速にアクセスできるローカライズされた分散メモリ上のデータとコアの組み合わせが重要となる .

分散メモリを前提とした並列プログラミングが
これからのスーパーコンピューティングには必須である.

MPIによる並列プログラミング

- 並列プログラミングにおけるデファクトスタンダードは,アセンブラに匹敵すると言う人もいるMPIである.
- しかし,一般のユーザにとって,MPIでプログラミングすることは,大きなストレスとなる.
  - バグの原因が,そもそものプログラムにあったのか,MPIを用いたことによって混入したのかを判断することが難しい.
  - 単純なデバック出力でさえも,プログラミングには大きな労力を必要とする.
- 特に開発途上のコードでは,非常に大きなストレスとなる.
  - 新しい物理モデルの導入,新しいアルゴリズムの採用によるコードの改変が頻繁にある.

並列プログラミングのジレンマ

- プログラムのソフトウェアとしての継承や開発の持続性およびそのコストを考えると,MPIによるプログラミングだけでは,どう考えても厳しい.
  - Life is too short for MPI.

    T-shirts message@WOMPAT2001
- しかし,並列計算機アーキテクチャーの分散メモリ化やプロセッサのマルチコア化の動向が,今後更に加速されそうな現実を考えると,並列プログラミングは必須である.
  - The free lunch is over.

    H. Sutter, Dr. Dobb's Journal, 30(3), (2005).

並列プログラミング環境

- OpenMP系
  - OpenMP+ccNUMA
  - OpenMP+Software Distributed Shared Memory
  - Cluster OpenMP
- 分散メモリ系
  - Co-Array Fortran
  - Unified Parallel C
  - Chapel, X10, ~~fortress~~
    - High Productivity Computing Systems (DARPA)
  - HPF(High Performance Fortran)

OpenMP系

- 分散メモリをユーザにはあたかも共有メモリのように見せるので,プログラミングは楽であるが…
- 性能を出すためには,超高速なネットワークが必須である.
  - ハードウェアが非常に高価になる.
- リモートメモリへのアクセスが発生した瞬間に,性能が急激に低下してしまう.
  - しかし,アクセスのローカリティを保証できない.
  - first touchだけでは,無理がある.
  - ページ単位でのメモリ割り付けにも,無理がある.
- 大規模並列での性能は,まったく期待できない.

Co-Array Fortran(UPC)

- データ転送は,GET/PUT的な片方向通信を原則としており,高い並列性能が得られるかどうかはインタコネクトのアーキテクチャに大きく依存している.
- プロセッサ間のデータ転送は,すべて明示的に記述しなければならない.
  - MPIによるプログラミングと,手間はほとんど同じ.
- 2005年にFortran2008の標準仕様として一旦採用されたが,まだ最終仕様や標準実装について,もめていた.
  - 2009年8月に仕様をダウングレードして決着した.
- 今までのFortranコンパイラでは,エラーになる.

サンプルプログラム

```
real :: r[*]     ! Scalar co-array
real :: x(n)[*] ! Array  co-array
! Co-arrays always have assumed co-size

real :: t        ! Local scalar
integer :: p     ! Local scalar

! Remote array references
! MPI_GET communication
t = r[p]
x(:) = x(:)[p]

! Reference without [] is to local part
! MPI_PUT communication
x(:)[p] = r
```

Chapel

- Cascade High-Productivity Language
- アドレス空間はグローバルであり,通信はコンパイラが自動的に生成する.
- データ並列,タスク並列を抽象化して記述方法する.
  - localeとdomain
- 考え方は,ほとんどHPFと同じ?
  - ただし, HPFとは逆に, OpenMPと同様にデータ分散より処理分担を優先している.
- まったく異なった言語であり,記述方法もFortranやCとは異なっている.
  - 既存のHPCユーザに受け入れられるか?

サンプルプログラム

```
var N: integer = 1000;
var A, B: [1..N] float;

// forall specify parallel execution
forall i in 2..n-1 do
   A(i) = B(i-1) + B(i+1);
```

```
var N: integer = 1000;
var CompGrid: [1..N] locale;
var D: domain(2) distributed(Block(2), CompGrid);
var A, B: [1..N] float;

forall i in D on B(i) do
   A(i) = B(i);
```

HPFの利点

- HPFによるプログラミングは,通常のFortranプログラムに指示文を挿入するだけである.
  - プログラミングそのものが容易である.
  - 通常のFortranではコメント行として扱われるので,原則として逐次プログラムと互換性がある.
- HPF指示文を増やすことで,段階的に並列化ができる.
  - ある程度の満足できる並列性能が得られれば,並列化の作業をそこで止めればよい.
  - 問題があった場合,前の段階にすぐに戻れる.
- 並列プログラミングを試すには,非常にお手軽である.

プログラムの比較

MPI

HPF

```
parameter(n=100)
real a(n), b(n)
read(*,*) a, b
aipd = 0.0
do i = 1, n
  aipd = aipd + a(i) * b(i)
end do
write(*,*) 'aipd = ', aipd
stop
end
```

```
parameter(n=100)
real a(n), b(n)
call MPI_INIT ( ierr )
call MPI_COMM_SIZE ( MPI_COMM_WORLD, np, ierr )
call MPI_COMM_RANK ( MPI_COMM_WORLD, id, ierr )
if( id .eq. 0 ) then
  read(*,*) a, b
  do i = 1, np-1
    call MPI_SEND ( a, ...
    call MPI_SEND ( b, ...
  end do
else
  call MPI_RECV ( a, ...
  call MPI_RECV ( b, ...
end if
is = ( n / np ) * id + 1
ie = ( n / np ) * ( id + 1 )
aipdt = 0.0
do i = is, ie
  aipdt = aipdt + a(i) * b(i)
end do
call MPI_REDUCE ( aipdt, aipd, ...
if( id .eq. 0 ) write(*,*) 'aipd = ', aipd
call MPI_FINALIZE ( ierr )
stop
end
```

```
      parameter(n=100)
      real a(n), b(n)
!HPF$ PROCESSORS proc(number_of_processors())
!HPF$ DISTRIBUTE (BLOCK) ONTO proc :: a,b
      read(*,*) a, b
      aipd = 0.0
!HPF$ INDEPENDENT, REDUCTION(+:aipd)
      do i = 1, n
        aipd = aipd + a(i) * b(i)
      end do
      write(*,*) 'aipd = ', aipd
      stop
      end
```

並列プログラミング手法の比較

- 並列プログラミングで大事な三つのポイントについて,それぞれの手法を比較する.

	データの分散	処理の分担	通信の生成
MPI			
OpenMP+DSM	※分散共有メモリ		※分散共有メモリ
Cluster OpenMP			※分散共有メモリ
HPF			
CAF(UPC)	※Co-Array		※代入文
Chapel			※グローバルメモリ

自動(不要)

自動＋
手動の最適化

手動(指示行)

手動

JAHPF

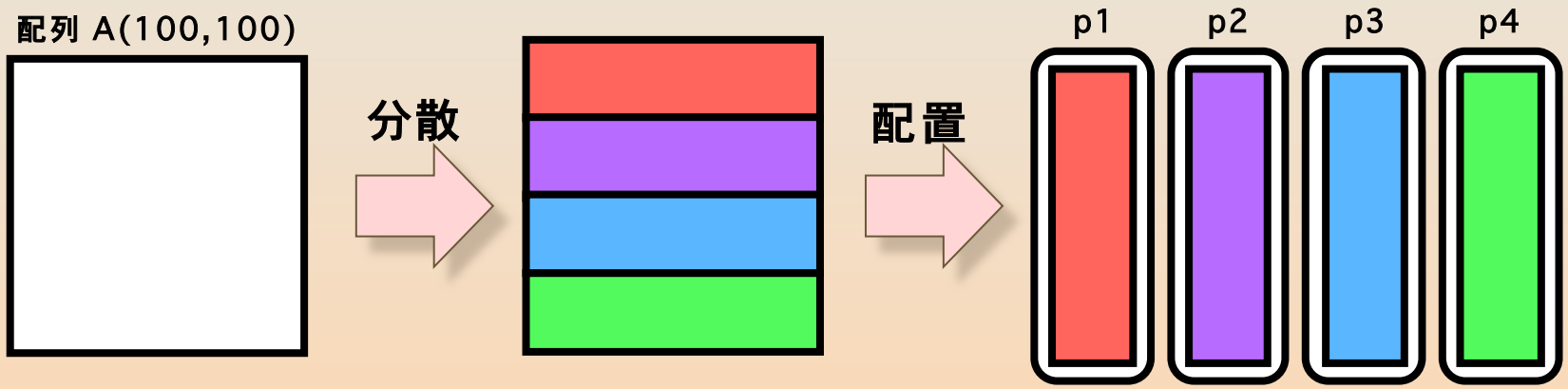
- HPF合同検討会（Japan Association for HPF）
  - http://www.hpfpc.org/jahpf/
  - 1997年1月29日に発足し,活動を開始した.
- HPFをベースとし,以下の3条件を満たす並列言語仕様を確立する.
  - 標準性：多用なプラットフォーム上で動作
  - 適用性：科学技術計算スキームの実装
  - 高速性：ハードウェア性能の活用
- HPF/JA仕様を策定した.

HPFPC

- HPF推進評議会（HPF Promoting Consortium）
  - http://www.hpfpc.org/
  - JAHPFを発展解消し,2001年7月9日に設立総会を開催して,任意団体としての活動を開始した.
- 活動の中心を言語仕様の開発から,実用アプリケーションのHPF化促進,実環境でのHPFの評価にフォーカスする.
  - HPF利用を支援するために,会員所有のコードをHPF化するための個別相談や講習会を行う.
- フリーのHPFコンパイラを提供している.
  - 富士通製 fhpf

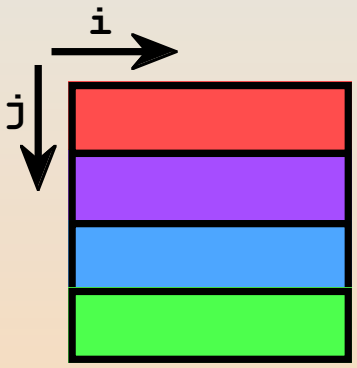
データの分散メモリ上への配置

- ユーザは,Fortran配列をどのように複数のプロセッサ上へ分散して配置するかだけを明示的に指定する.（データ分散）



データ分散に従った処理の分担

- 更新データを保持するプロセッサが処理を行うように並列化する.(処理分担)
  - Owner Computes Rule




```
      dimension a(100,100),b(100,100)
!HPF$ DISTRIBUTE (*,BLOCK) :: a,b
      do j = 2, 100
        do i = 1, 100
          a(i,j) = a(i,j)*b(i,j)+b(i,j-1)
        end do
      end do
```

```
p1:j=2,25
p2:j=26,50
p3:j=51,75
p4:j=76,100
```

必要なデータ転送は,
コンパイラが自動で面倒を見る.

ループに対する並列実行の指示

- コンパイラだけでは判断できない場合にループを並列に実行しても問題のないことを明示する.

```
!HPF$ INDEPENDENT
      do i = 1, 100
        a(ind(i)) = a(ind(i)) + b(i)
      end do
```

  - ind(i)に重なりがあった場合,並列実行すると回帰参照により正しい実行結果が得られないが,コンパイラはind(i)に重なりがあるかどうか判断できないのでループを並列化しない.
  - そこで,ind(i)に重なりがない場合,コンパイラにループの並列実行を指示する.

集計演算の指示

- ループ中に総和等の集計演算(reduction)がある場合,単純に並列実行しただけでは演算結果が不正になる.
- そこで,各プロセッサが部分的な結果を求め,最後に全プロセッサで最終的な結果を求める特別な指示をする.

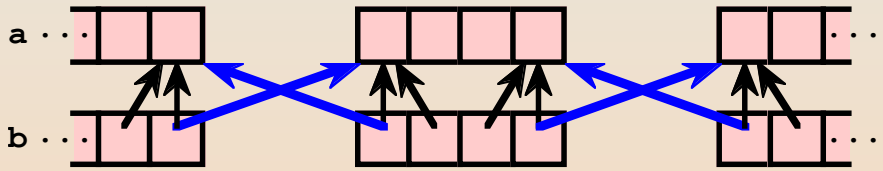
```
      s = 0.0
!HPF$ INDEPENDENT, REDUCTION(+:s)
      do i = 1, 100
        s = s + a(i) * b(i)
      end do
```

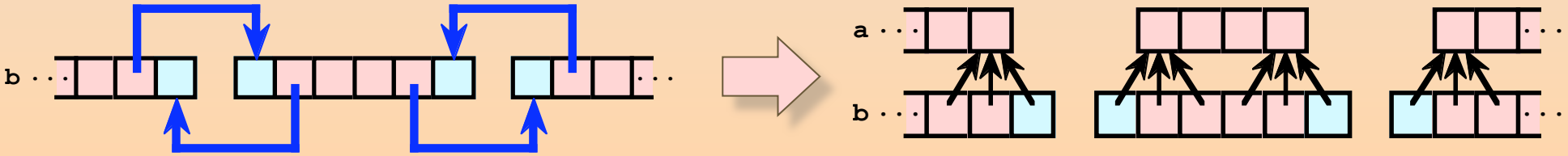
  - 総和計算であることを明示する.

袖領域の定義とその通信

- 分散配列に袖領域を定義して,その部分を一括通信すると効率が良い.

```
      real a(100), b(100)
!HPF$ DISTRIBUTE(BLOCK) ONTO linear::a,b
!HPF$ SHADOW b(1)
      do i = 1, 100
         b(i) = …
      end do
!HPF$ REFLECT b
!HPF$ INDEPENDENT
      do i = 2, 99
        a(i) = b(i-1) + b(i) + b(i+1)
      end do
```

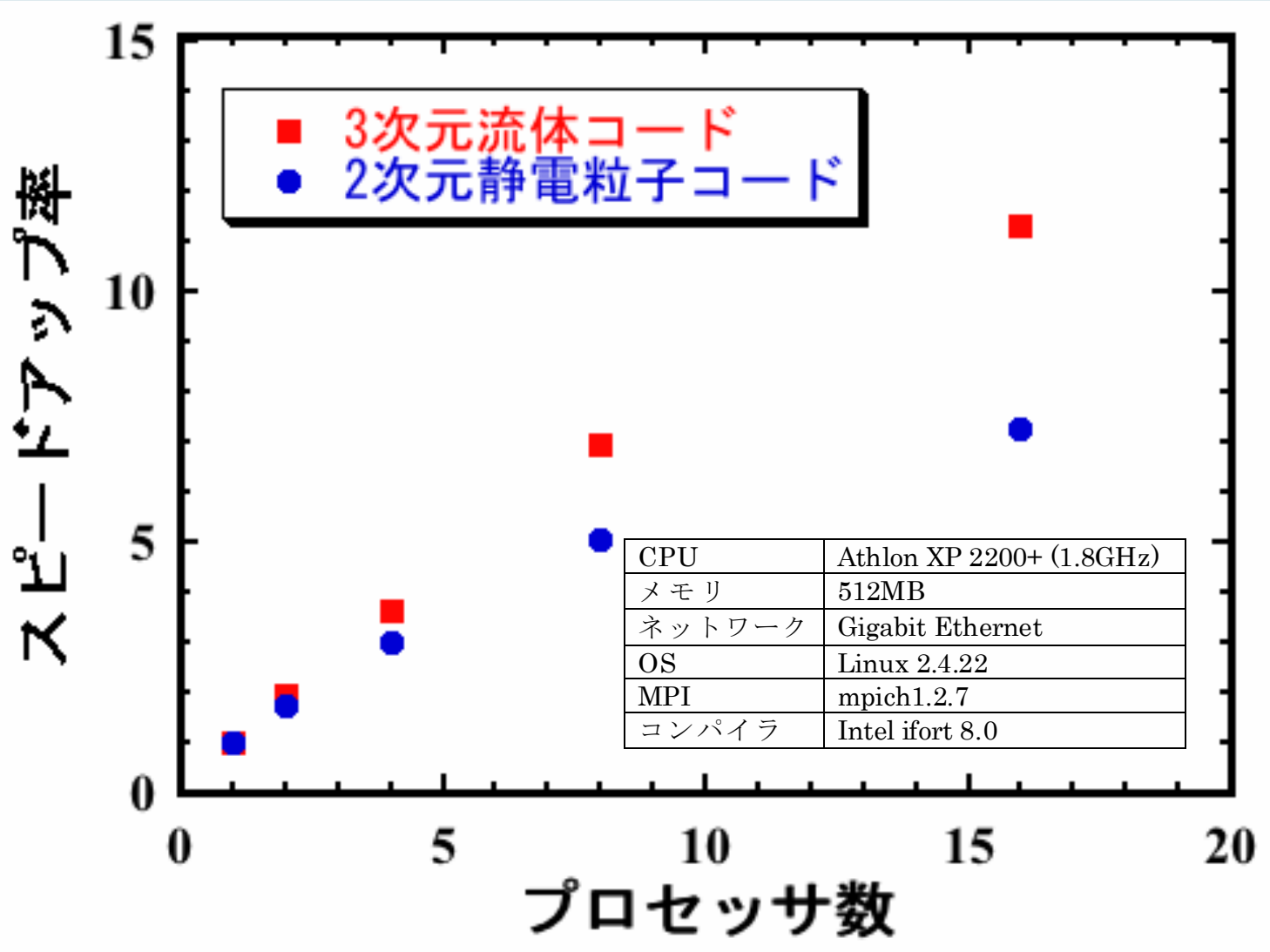




HPFによるプログラミング

1. プログラムにおいて並列化する部分を決定する.
   - 10%のサブルーチンがCPU時間の90%を費やす.
2. その部分において,並列化のためのデータ転送が最も少いデータ分散方法を決定する.
   - PROCESSORS, DISTRIBUTE指示文
3. コンパイラが自動並列化できないループについて明示的に並列化を指示する.
   - INDEPENDENT指示文
4. 各種最適化を行う.
   - ループ処理分担の追加補足のための指示
   - データ転送を最適化するための指示

fhpf+PCクラスタによる性能



CPU	Athlon XP 2200+ (1.8GHz)
メモリ	512MB
ネットワーク	Gigabit Ethernet
OS	Linux 2.4.22
MPI	mpich1.2.7
コンパイラ	Intel ifort 8.0

HPFの現状

- ユーザが指示文によりデータ分散を書けば,残りの作業(処理の分担と通信の生成)をコンパイラが自動的に生成するというのは,既存コードとの親和性や段階的並列化を考えると理想的だったが…
  - 仕様に対する完全性追求とコンパイラ実装の困難性
  - Fortran90をベースとした失敗,過度な期待と失望
  - 自動生成によるチューニングの困難性
  - レファレンス実装の不在とユーザへの啓蒙／教育不足
- 現在活動しているのは,ほぼ日本だけであり,今後広く普及する可能性は低いと言わざるを得ない.
  - 富士通のVPP Fortranと同様に,Fortranの方言として利用される可能性が高い.

まとめ

- とりあえず,手軽にやってみる!
  - 最初から苦労してMPIを用いて不可逆的なプログラミングをするより,まず,HPFによる並列化を試してみる価値は,十分にある!
- HPFでは,挿入する指示文を順次増やすことで,ステップバイステップに段階的に並列化やチューニングができる.
  - 並列化の手間を考慮して,得られた並列性能に納得できれば,並列化をそこで止めればよい.
  - 問題があった場合,すぐに前の段階に戻れる.
- 規則的なデータ構造のプログラムなら,HPFでも十分な並列性能が得られることが多い.
  - できるだけ通信が起こらないデータ分散を考える.

FYI

- HPF推進評議会（HPF Promoting Consortium）
  - http://www.hpfpc.org/
  - フリーのHPFコンパイラやサンプルソースコードを配布している.
- PCクラスタで並列プログラミング
  High Performance Fortranで楽々並列化
  - 津田孝夫 監修,HPF推進協議会 編
  - 岩下英俊・坂上仁志・妹尾義樹・林康晴 共著
  - 出版社: 培風館
  - 発売日：2011年3月28日
  - ISBN978-4-563-01586-2
  - 定価：3400円+税