HPCプログラミングと
性能可搬性

広島市立大学
窪田昌史

HPCシステムの動向

- アクセラレータ(GPGPUなど)の普及
  - ➢参照の局所性を抽出する必要性
- スパコンの大規模化：
  - ➢ノード間、ノード内、アクセラレータへの
    - 並列性の振り分け
    - データ配置の最適化

HPCプログラミング向け言語・API

- 分散メモリ
  - MPI
  - HPF
  - CAF, XcalableMP, X10, Chapel, UPC, Fortress
- 共有メモリ
  - pthreads
  - OpenMP
- GPGPU
  - CUDA, OpenCL, OpenACC

その他…

性能可搬性

新たなHPCシステムが登場しても、（大幅に）プログラムを書き変えず、高い性能を得たい

Write once, run anywhere with high performance

- ディレクティブ
  - 並列性とデータ配置の指定
- 自動チューニング
- 自動並列化、自動データ配置最適化

今日の主な話題

- OpenACC

  ディレクティブによるアクセラレータの実行制御

- 自動データ配置最適化

  窪田,北村：配列整合解析に基づく自動データ分割手法、情報処理学会論文誌プログラミングVol.3, No.1, pp.41-53 (Mar.2010)

OpenACC

- ディレクティブによって、アクセラレータでの実行部分を指定

  例：`# pragma acc kernels`

- アクセラレータとホスト間で転送するデータは明示的に指定

  例：`copyin(a[0:n]), copyout(b[0:n])`

- Specification 1.0(Nov.2011), 2.0(Jul.2013)
- PGIコンパイラなどで使用可能

行列積による評価

行列積による正/逆ルジャンドル陪変換

- 全球大気モデルのスペクトル変換法などで使用

実装方法

1GPUに収まるデータサイズを想定

- OpenACC実装
- CUDA実装（cuBLASの行列積を呼び出す）

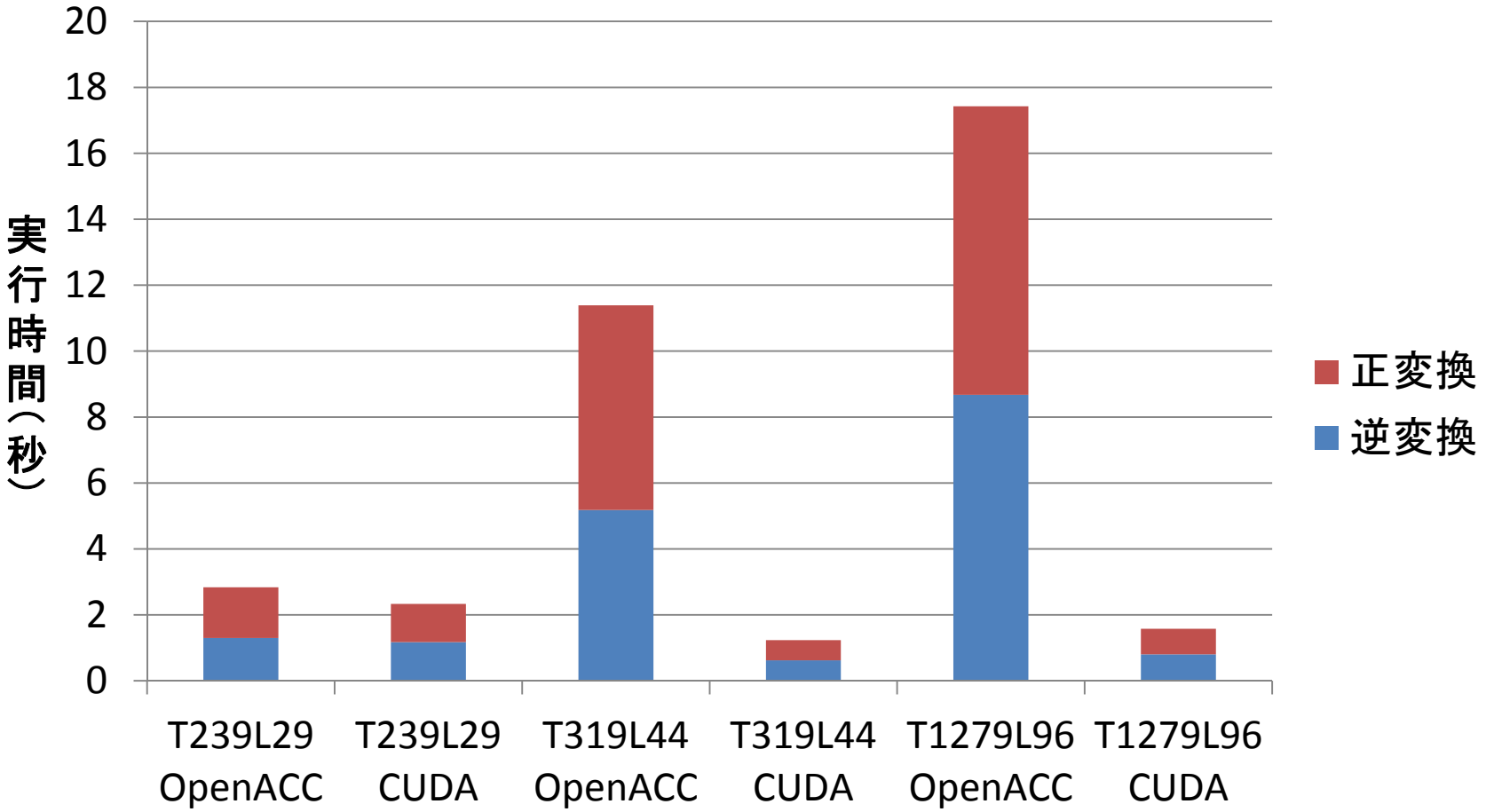
ディレクティブの挿入の例

```
for (m=0; m<MP; m++) {
  ...(略)...
#pragma acc kernels for
copyin(spect_ilt[0:size_2h][0:size_lmp],
alp_ilt[0:size_lmp][0:size_l],ft_ilt[0:size_2h][0:size_l])
copyout(ft_ilt[0:size_2h][0:size_l])
    for (ilev=0; ilev<size_2h; ilev++) {
      for (lm=0; lm<size_lmp; lm++) {
        for (ilat=0; ilat<size_l; ilat++) {
          ft_ilt[ilev][ilat] += spect_ilt[ilev][lm]
                              * alp_ilt[lm][ilat];
        }
      }
    }
...(略)...
}
```

GPUワークステーションの仕様

PCの仕様	
CPU	Core i7 980 3.33GHz 6cores
Memory	12GB DDR3
OS	CentOS 6.3 x86_64
GPU	Tesla C2070 6GB Memory
Compute Capability	2.0
CUDA	5.0
Compiler	PGI 13.3

GPUを用いた行列積の実行結果

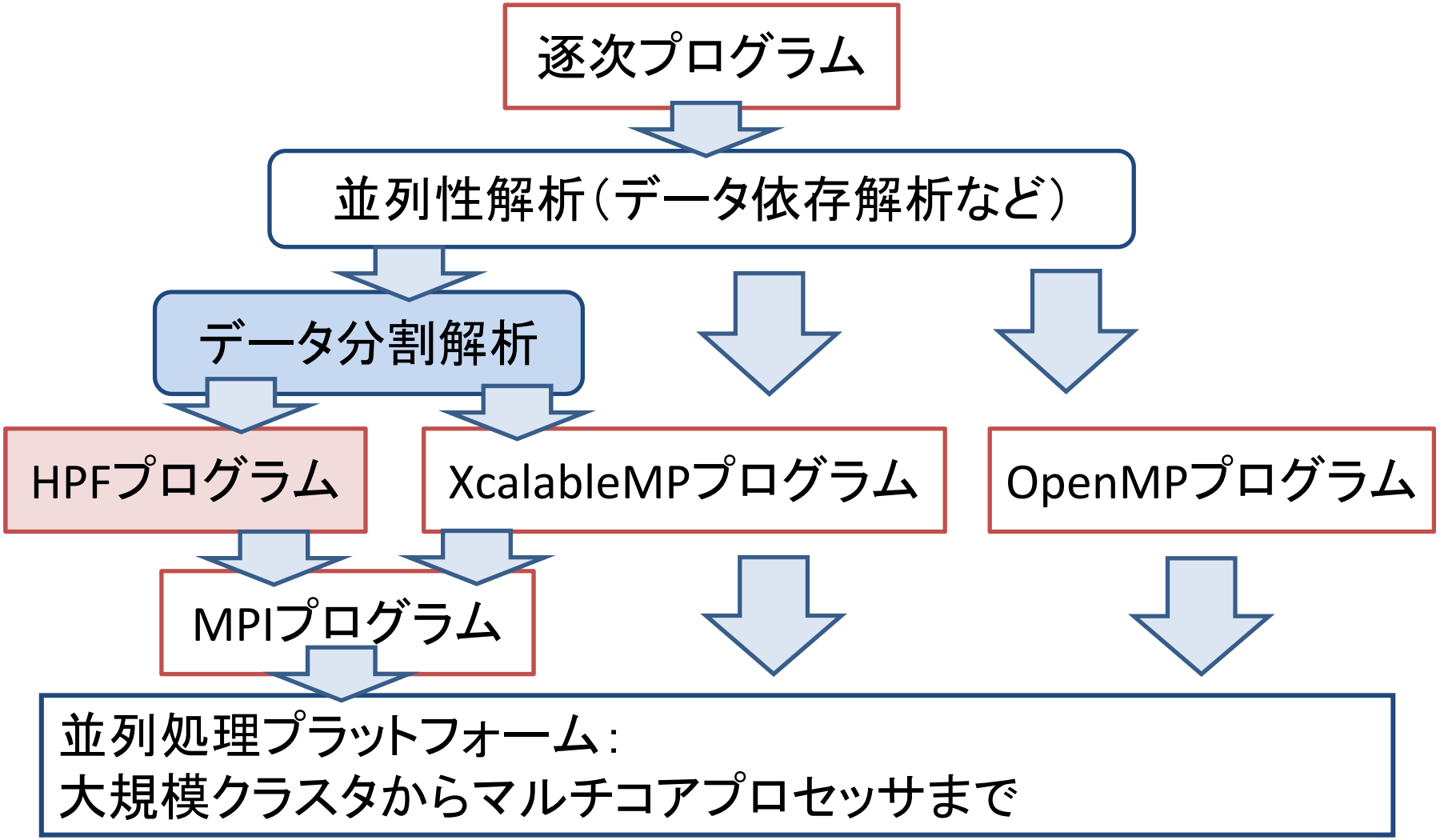


考察

- データサイズが小さい場合、OpenACCで、CUDA(cuBLAS)並みの性能
- CUDA用のコードをOpenACCで書き換えたために高性能

- 性能可搬性の観点から:
  - ディレクティブを無視すると汎用CPUでも動作
  - 他のアーキテクチャへ移植する場合、修正が必要か？

自動並列化を目指して


逐次プログラム
並列性解析（データ依存解析など）
データ分割解析
HPFプログラム
XcalableMPプログラム
OpenMPプログラム
MPIプログラム
並列処理プラットフォーム：
大規模クラスタからマルチコアプロセッサまで

関連研究：データ分割

- NP困難な問題
- CAG(Component Affinity Graph)ベース

  [Li and Chen 90][Gupta and Banerjee 93] [Kennedy and Kremer 95] [辰巳ほか 96]
- 配列のストライドに着目 [松浦ほか 00]
- 分散共有メモリへの対応 [廣岡 and 太田 00]

- 解析に要する時間が長い
- 解析できるコードに制約がある

ループの並列性とデータ分割

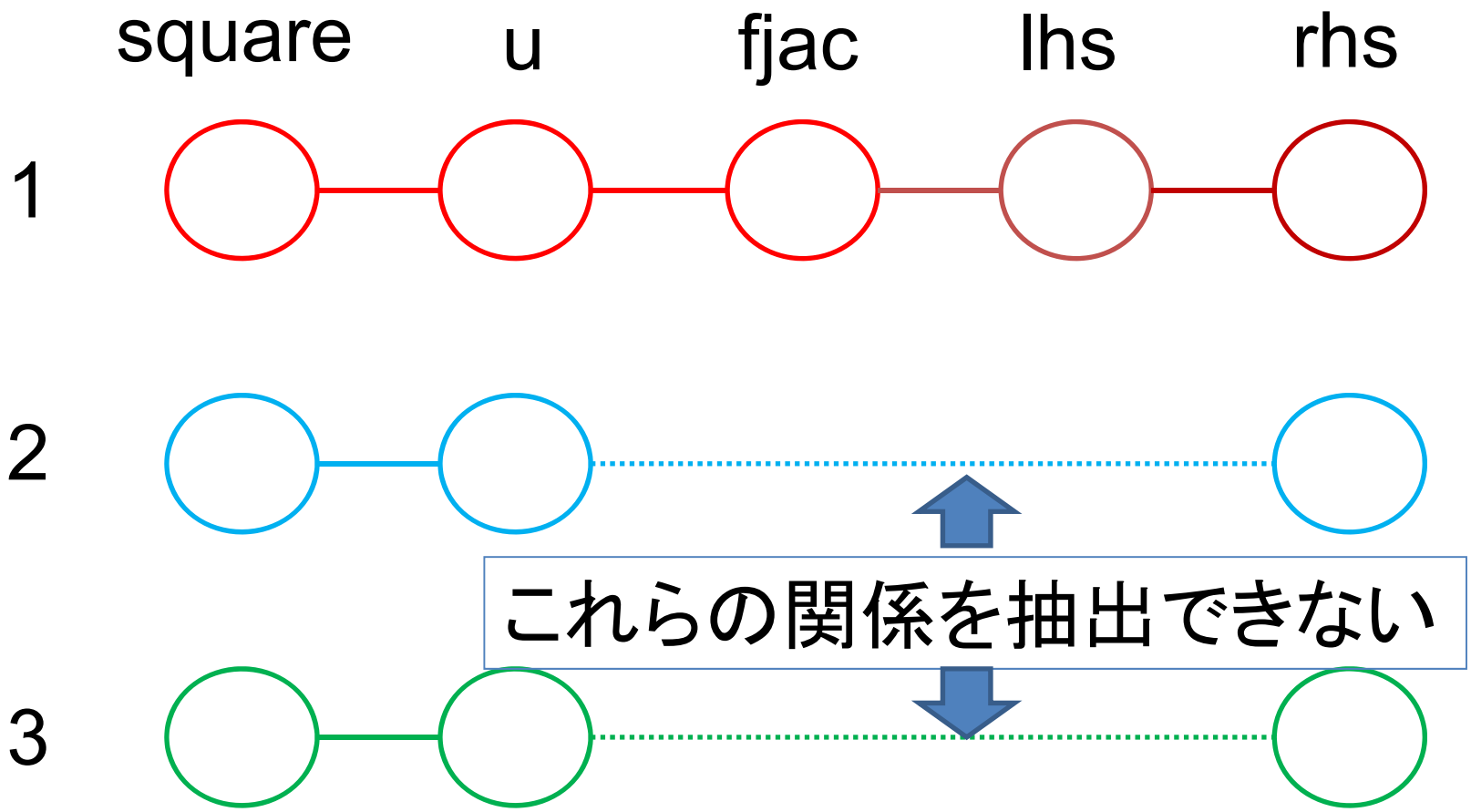
```
DO J=1,N
  DO I=1,N
    A(I,J)=B(I-1,J)+B(I,J)+B(I+1,J)
  ENDDO
ENDDO
```

- I,Jどちらのループでも並列化可能
- 配列A,Bの1,2次元目のどちらの次元でも分割可能
- 配列A,Bの2次元目を分割し、Jループを並列化するのが望ましい

HPFの指示文

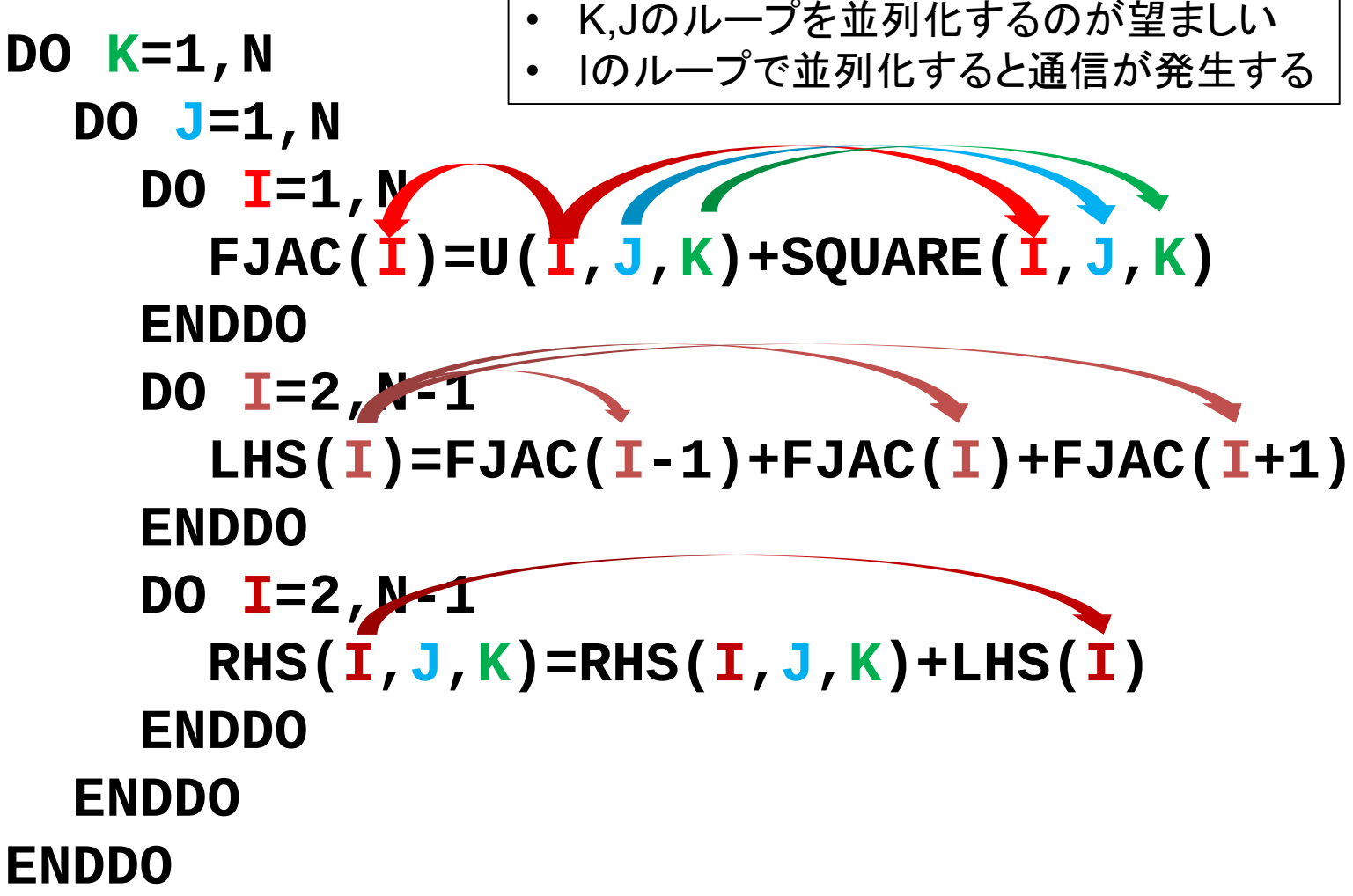
```
!HPF$     DISTRIBUTE A(*,BLOCK)
!HPF$     DISTRIBUTE B(*,BLOCK)
```

NPB 3.2-SER BT x_solve の CAG



NPB 3.2-SER BT x_solve(一部)

- K,Jのループを並列化するのが望ましい
- Iのループで並列化すると通信が発生する

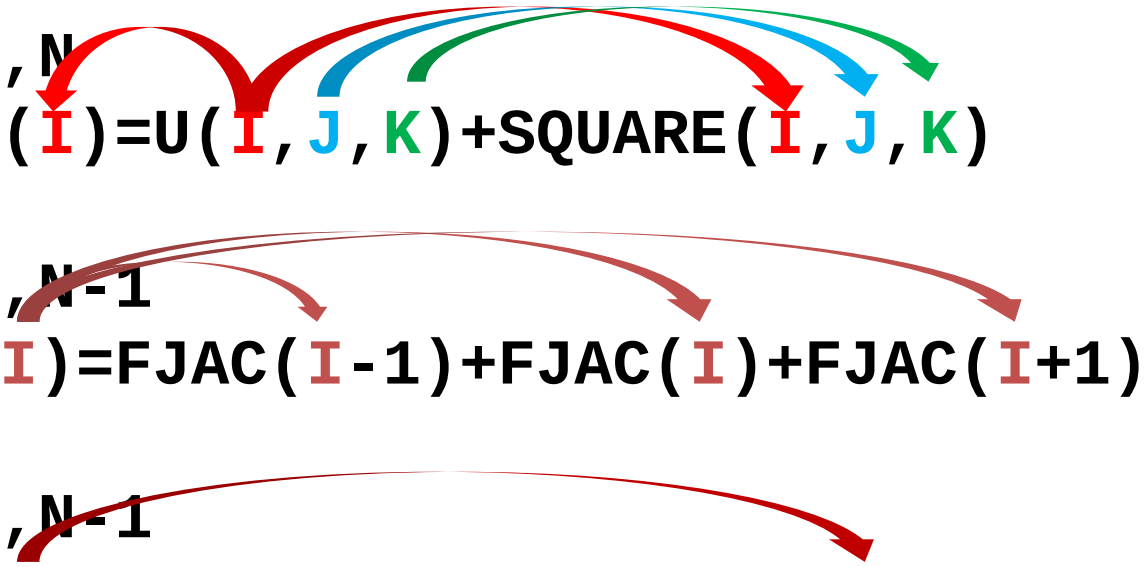


提案手法:配列整合解析による分割次元決定

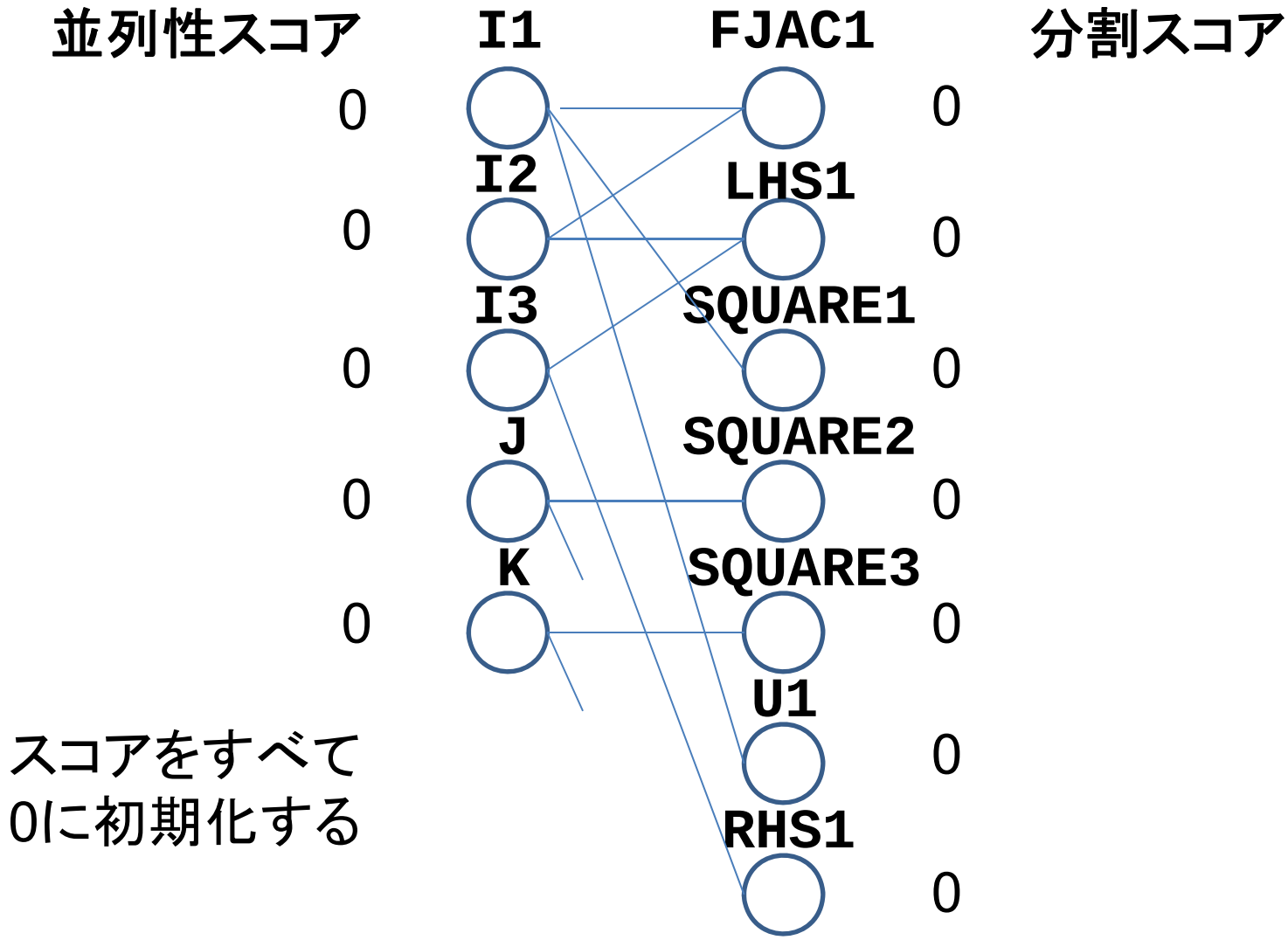
- 並列性スコア:0以上の数値
  - 0の場合は並列実行に支障がない
  - スコアが大きいほど並列実行に支障が出る
- 分割スコア:0以上の数値
  - 0の場合は分割に支障がない
  - スコアが大きいほど分割に支障が出る

NPB 3.2-SER BT x_solve(一部)

```
DO K=1,N
  DO J=1,N
    DO I=1,N
      FJAC(I)=U(I,J,K)+SQUARE(I,J,K)
    ENDDO
    DO I=2,N-1
      LHS(I)=FJAC(I-1)+FJAC(I)+FJAC(I+1)
    ENDDO
    DO I=2,N-1
      RHS(I,J,K)=RHS(I,J,K)+LHS(I)
    ENDDO
  ENDDO
ENDDO
```



スコア設定(1/3)

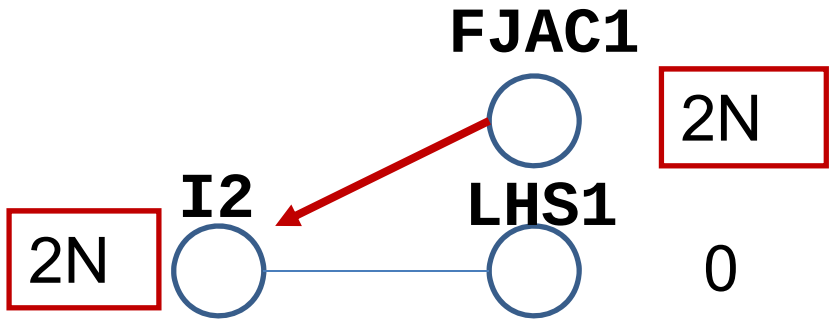


スコアをすべて
0に初期化する

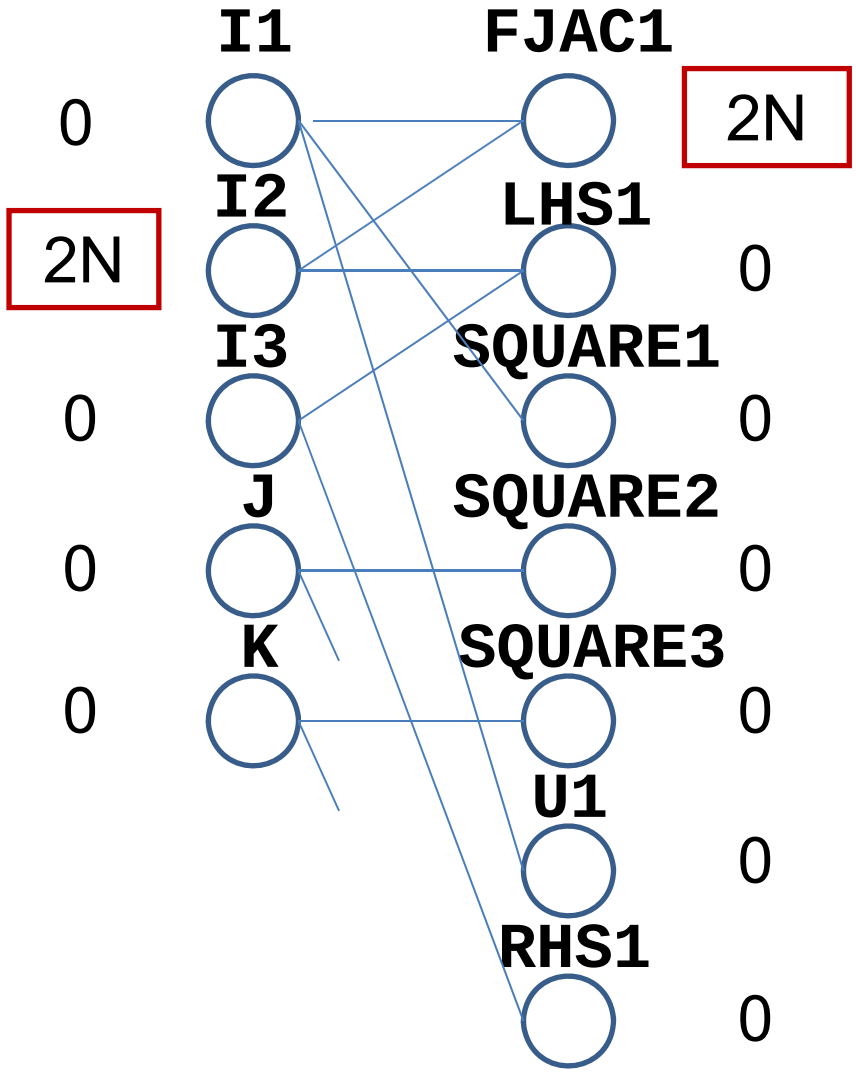
スコア設定(2/3)

```
DO K=1,N
  DO J=1,N
    …
    DO I=2,N-1
      LHS(I)=FJAC(I-1)+FJAC(I)+FJAC(I+1)
    ENDDO
    …
  ENDDO
ENDDO
```

通信量2N



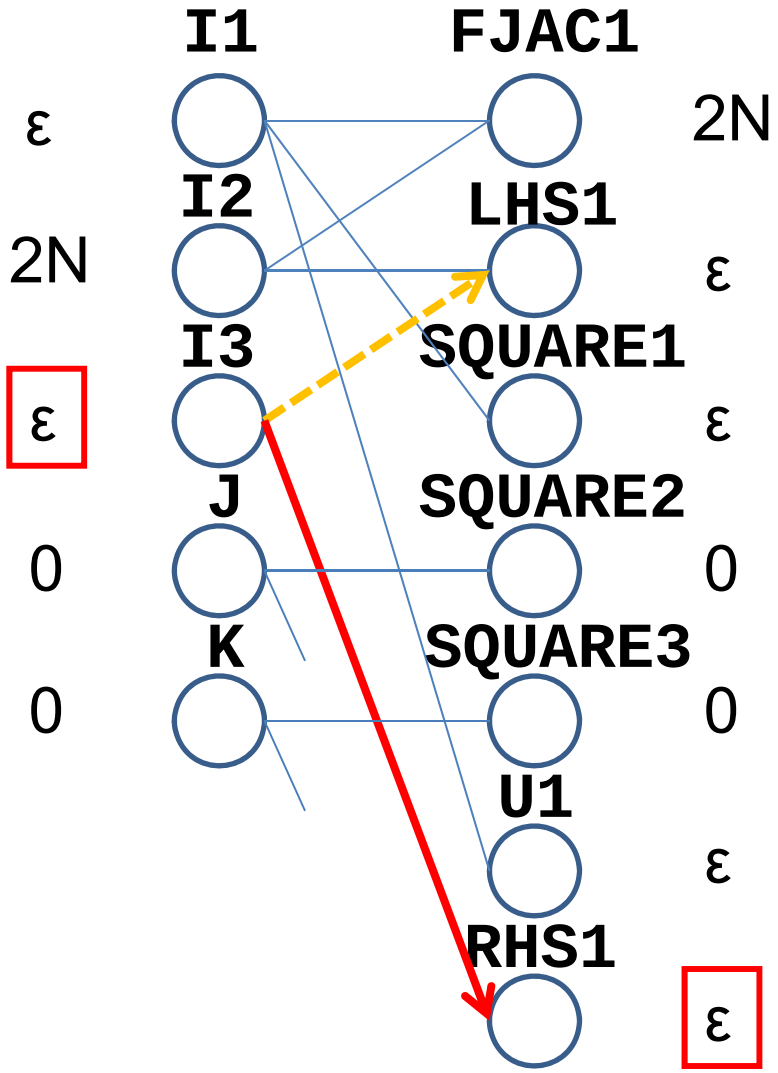
スコア設定(3/3)


I1
FJAC1
0
2N
2N
I2
LHS1
0
I3
SQUARE1
0
0
J
SQUARE2
0
0
K
SQUARE3
0
0
U1
0
RHS1
0

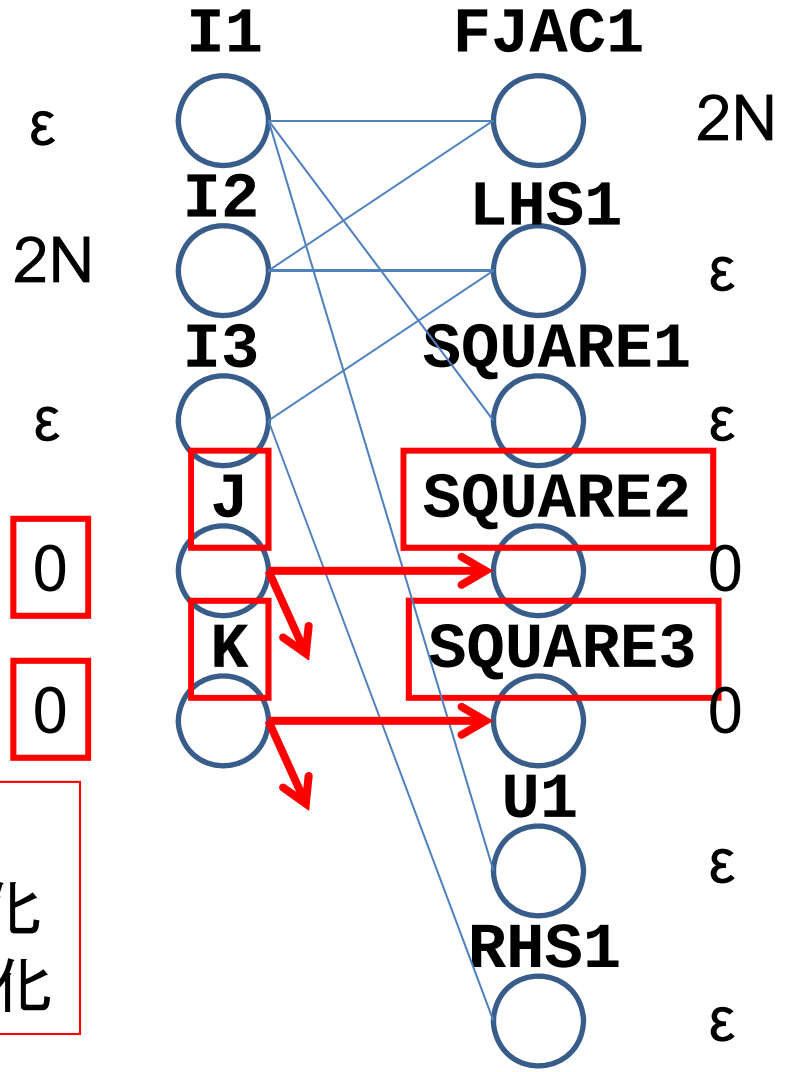
スコア伝播(5/5)

並列性スコアから
分割スコアへの
スコア伝播

これ以上スコアは
更新されず
アルゴリズム停止



分割次元の決定



並列化候補
- ループJでの並列化
- ループKでの並列化

分割次元候補
SQUAREの
- 2次元目
- 3次元目

U,RHSも同様

分割次元の決定

- K,Jの両ループが並列化候補
  - Kループを並列化：U,SQUARE,RHSの3次元目を分割
  - Jループを並列化：U,SQUARE,RHSの2次元目を分割
- Kループの並列化を選択：Fortranが列優先であるため、より高い次元での分割を選択

  （再分割のプロセス間通信のパック･アンパック操作を少なくするため）

性能評価

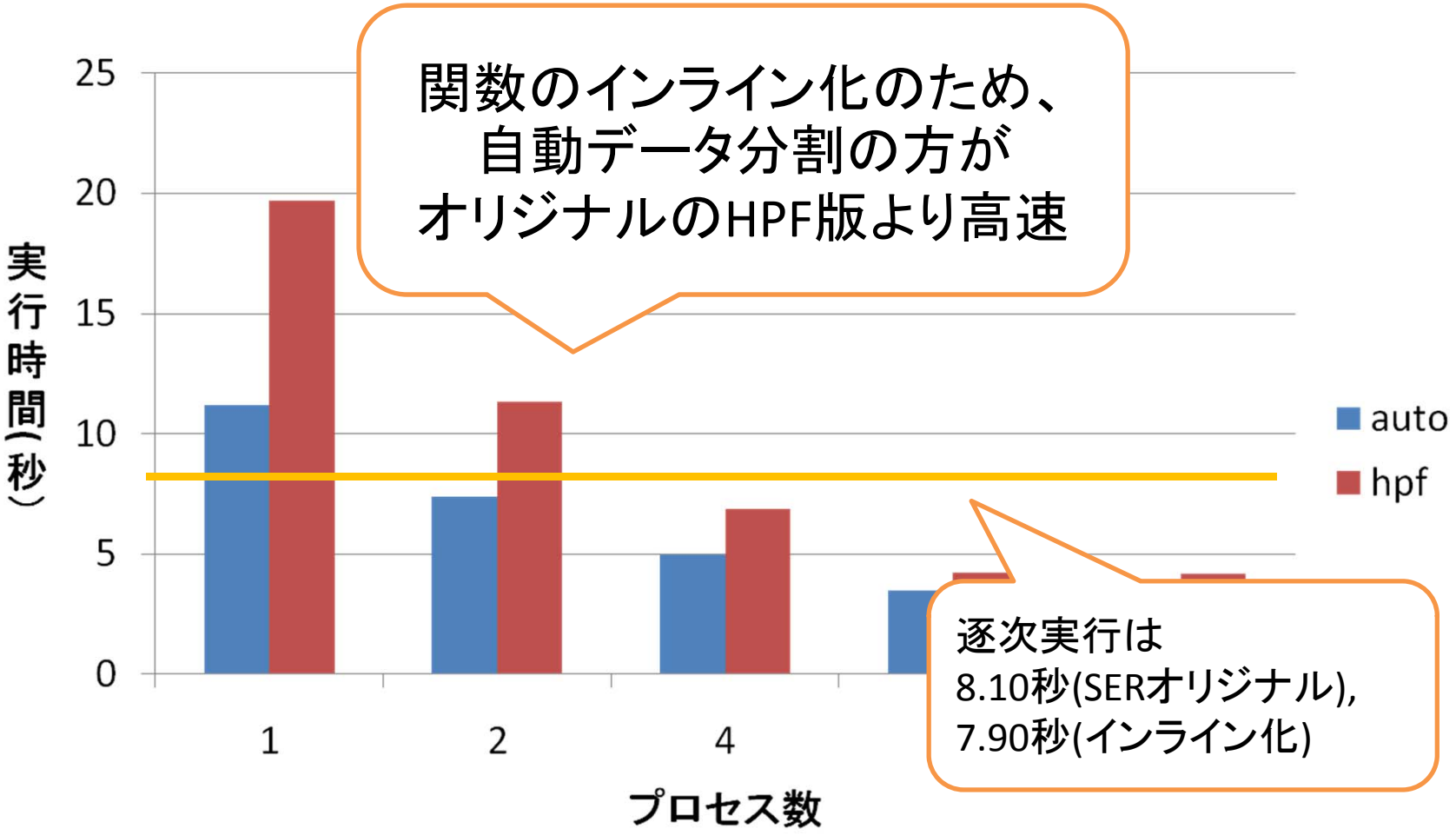
- NPB3.2-SER BT Class W,Aに提案手法で自動データ分割して実行
  - NPB3.0-HPF BTと比較
- 他の分割と比較

- 解析手法の制約から、matvec_sub, matmul_sub, binverhs, binvrhsなどのサブルーチンを手動でインライン展開

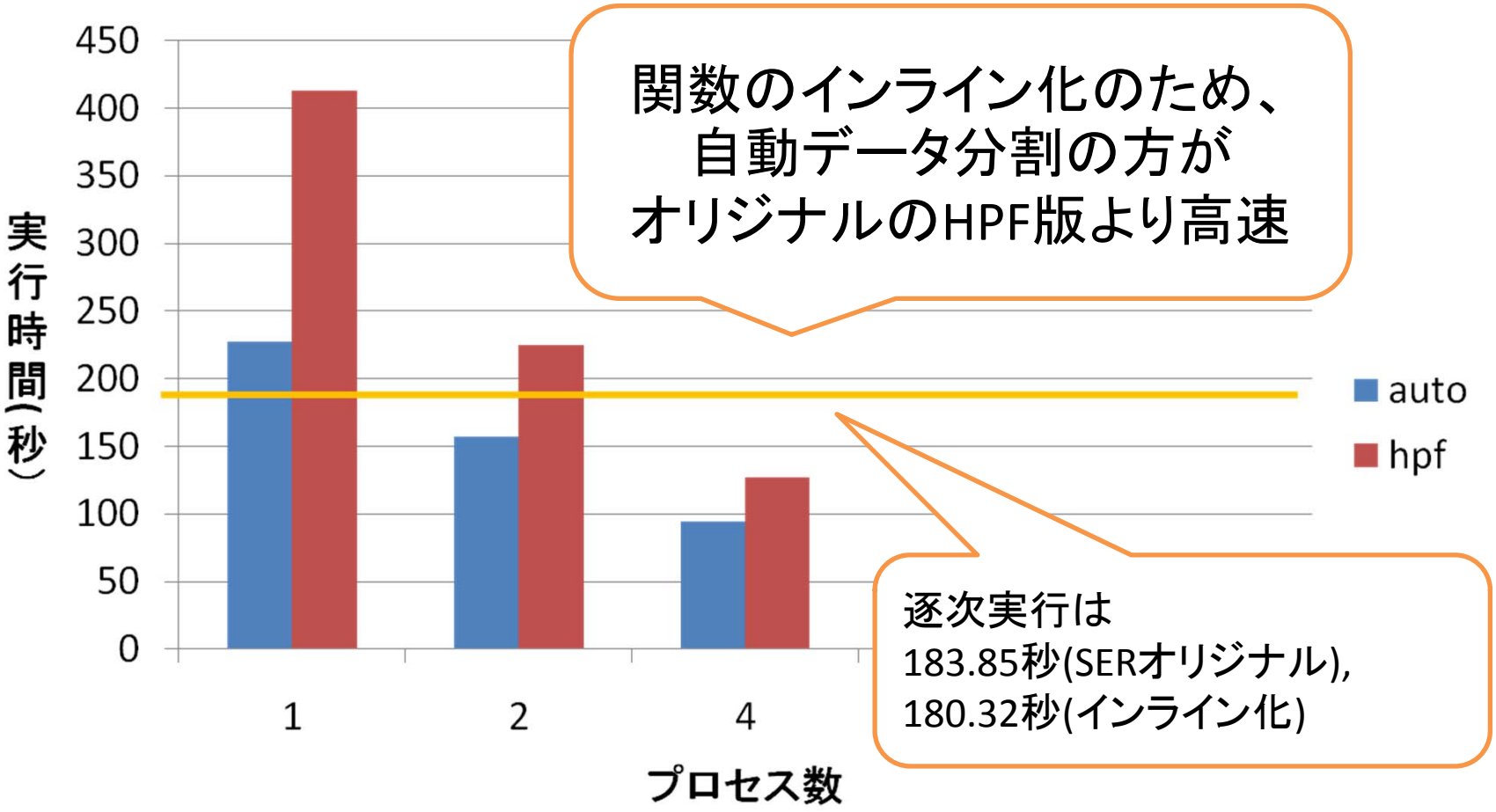
PCクラスタの仕様

CPU	
CPU	Intel Xeon 2.8GHz
L2 Cache	512KB
ノード	
CPU数	2
主記憶	1GB
Network	Myrinet-2000 双方向 2.0Gbps
OS	Fedora Core 3
Compiler	PGI HPF 7.2
Score	5.8.3
ノード数	8

NPB3.2-SER BT Class W



NPB3.2-SER BT Class A

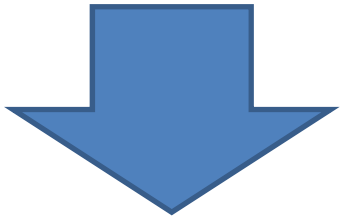


まとめ

- ループの並列性と配列の次元の分割を統一的に扱うためにアクセス関数を用いた配列整合解析手法を提案
- NPB3.2-SER BTで評価を行い、提案手法の有効性を確認
- Fortranが列優先であることを考慮した配列次元選択の有効性を確認

今後の課題

- XcalableMPプログラムを自動生成する処理系の開発
- 多次元分割への対応
- GPGPUなどのアクセラレータへの対応



自動並列化・最適化による性能可搬性の実現