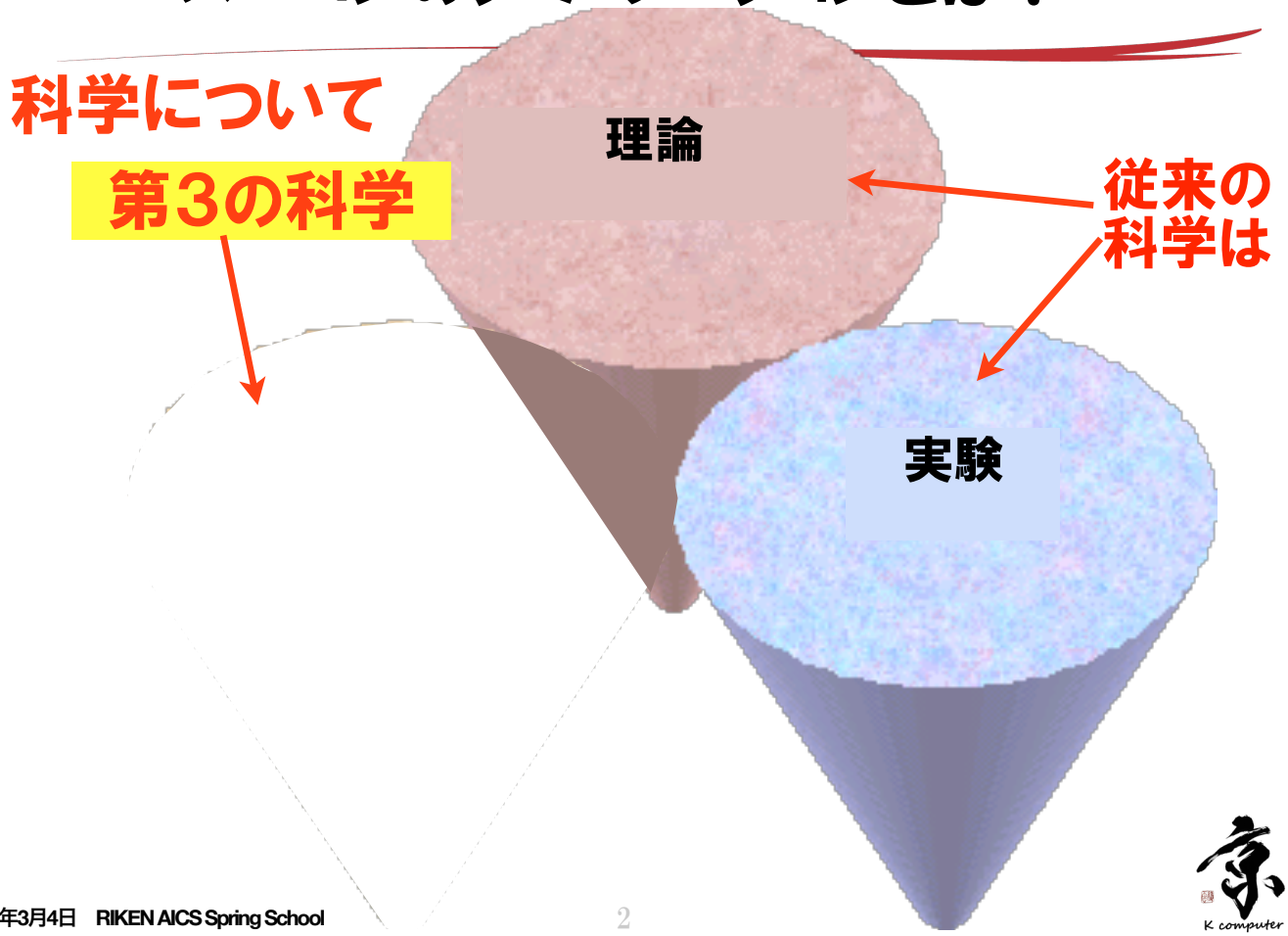
スーパーコンピュータを使う技術（概説）

2016年3月8日

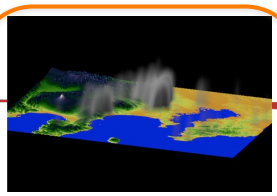
独立行政法人理化学研究所
計算科学研究機構 運用技術部門
ソフトウェア技術チーム チームヘッド

南 一生
minami_kaz@riken.jp

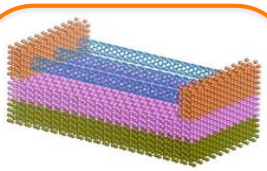
スパコンのシミュレーションとは？



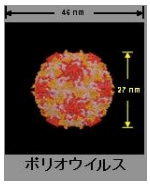
具体的なアプリケーション



難しい大気現象の解明、また正確な台風の進路・強度の予測による気象への貢献



次世代のデバイス全体のシミュレーションによるエレクトロニクスへの貢献



ポリオウイルス

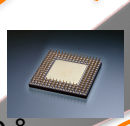
水中のウィルスの丸ごとシミュレーションによる医療への貢献

10^{21}m

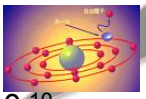
10^{7}m

10^{2}m

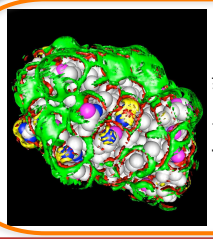
10^{0}m



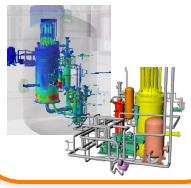
10^{-8}m

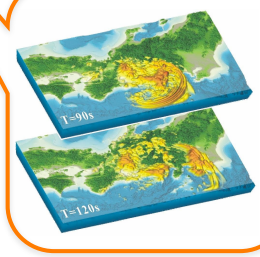


10^{-10}m



セルロース分解酵素のシミュレーションによる安価なバイオ燃料の提供等エネルギーへの貢献





短周期地震波動の地震波シミュレーション、構造物の耐震シミュレーションを組み合わせた防災への貢献

現代のスパコン利用の難しさ

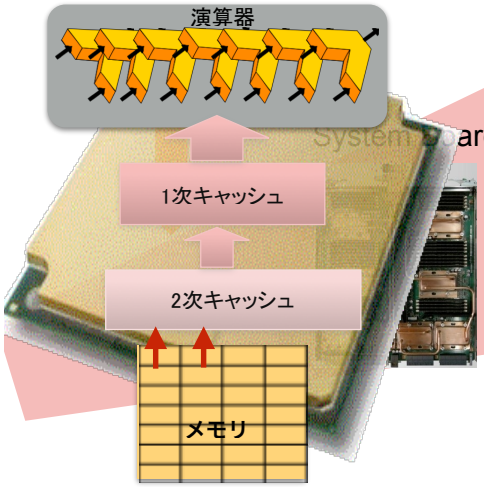
アプリケーションの性能最適化

アプリケーションの超並列性を引き出す

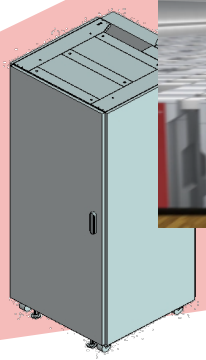
プロセッサの単体性能を引き出す

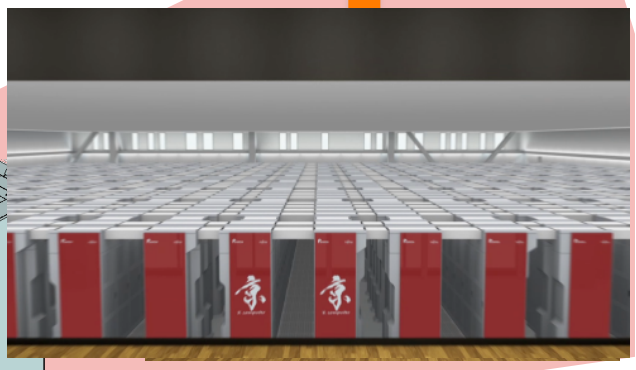
現代のスーパーコンピュータシステム

現代のプロセッサ



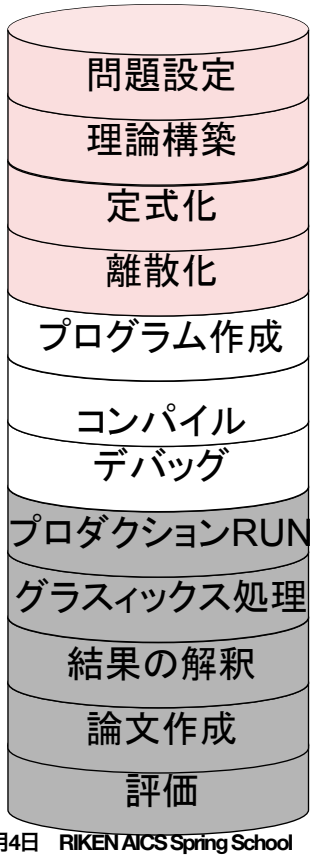
Rack





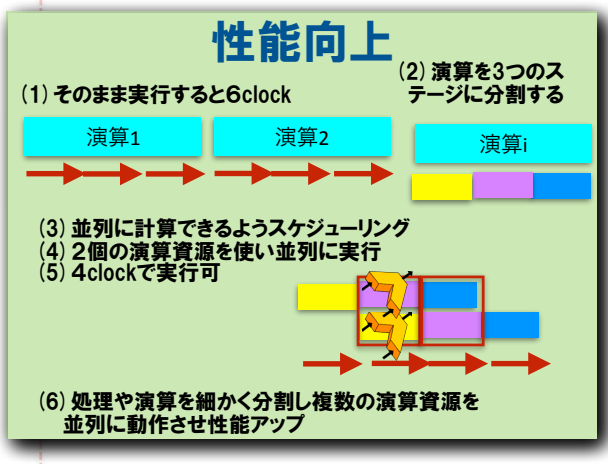
計算科学

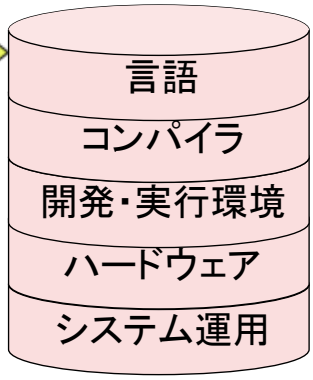
計算機科学



理論に忠実な分かりやすいコーディング

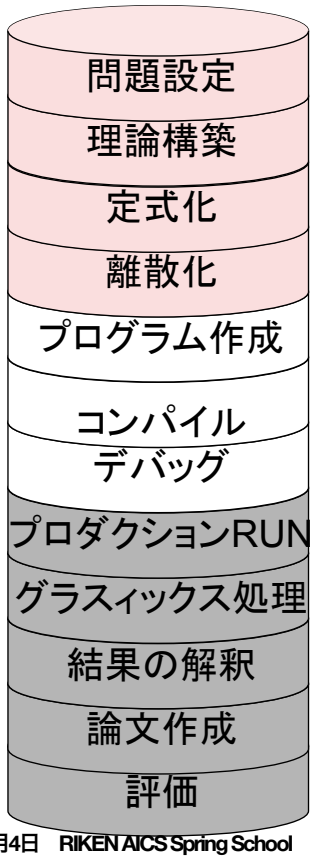
プログラム





計算科学

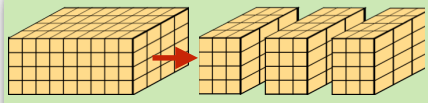
計算機科学



理論に忠実な分かりやすいコーディング

プログラム

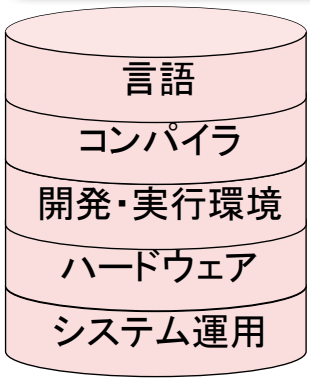
現代のプログラミング（スパコンを使う場合）



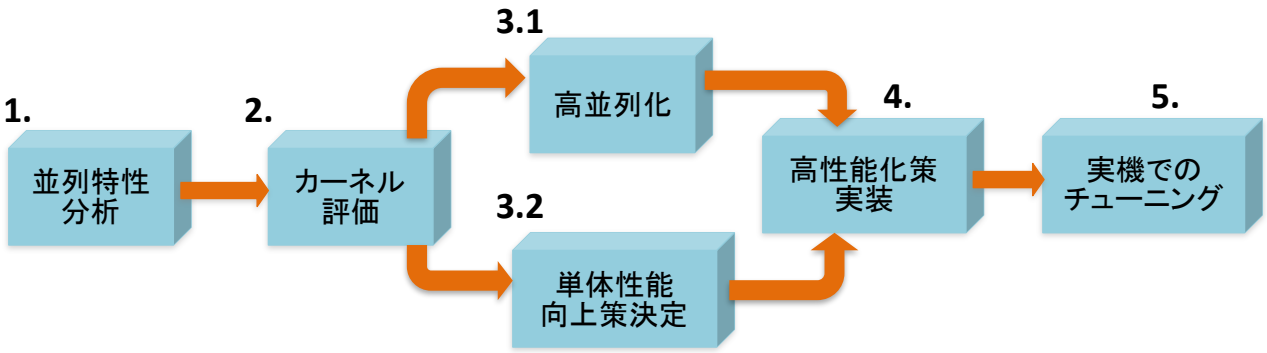
- データを各プロセッサへ分割
- 処理を各プロセッサへ割当
- プロセッサ間での通信を記述
- キャッシュの有効利用プログラミング
- 演算器有効利用プログラミング
- メモリ性能有効利用プログラミング
- コンパイラ有効利用プログラミング

プログラム

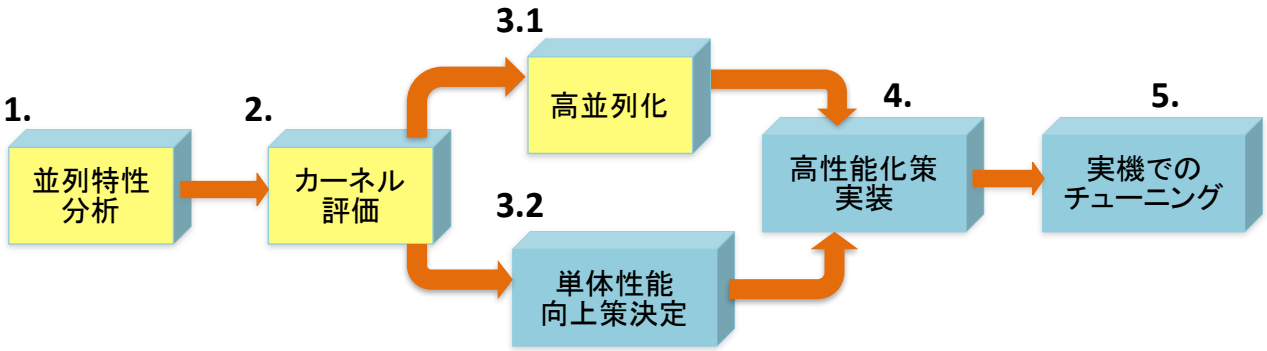
高並列化・高性能コーディング



アプリケーションの性能最適化のステップ

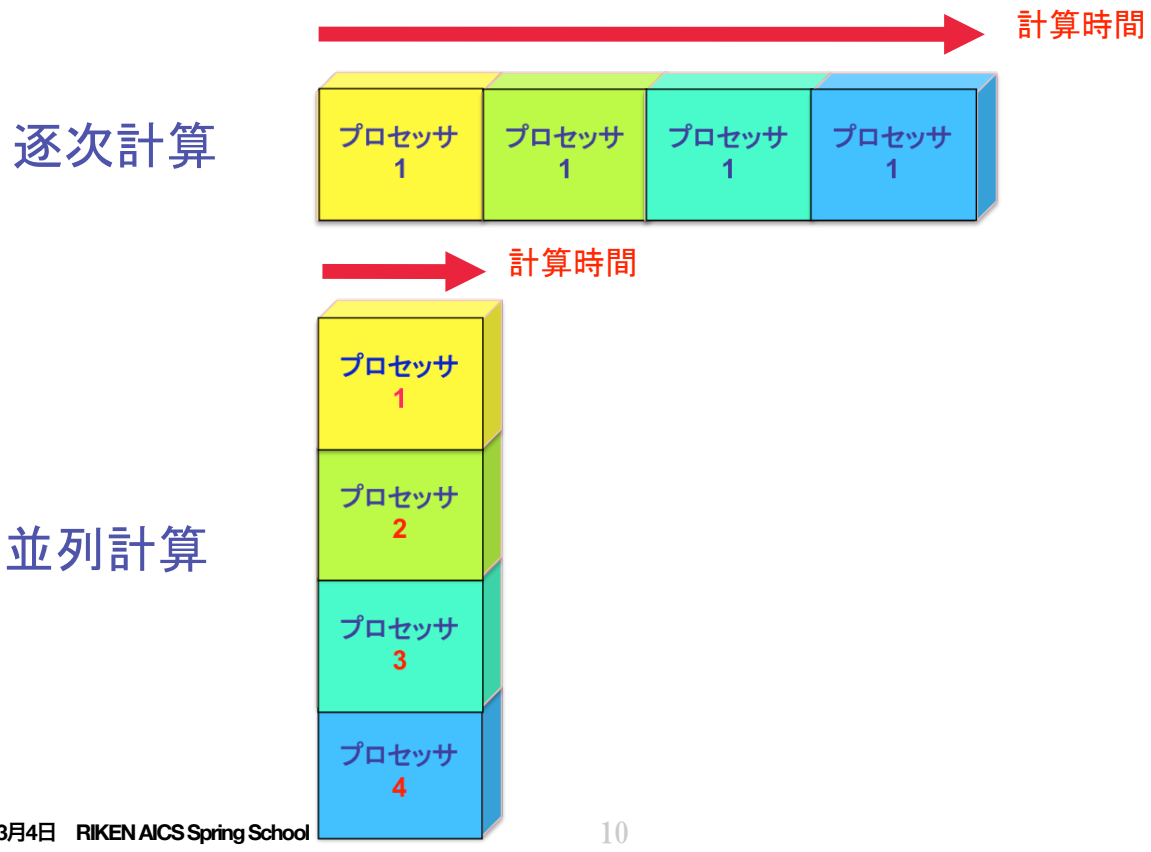


並列特性分析・カーネル評価

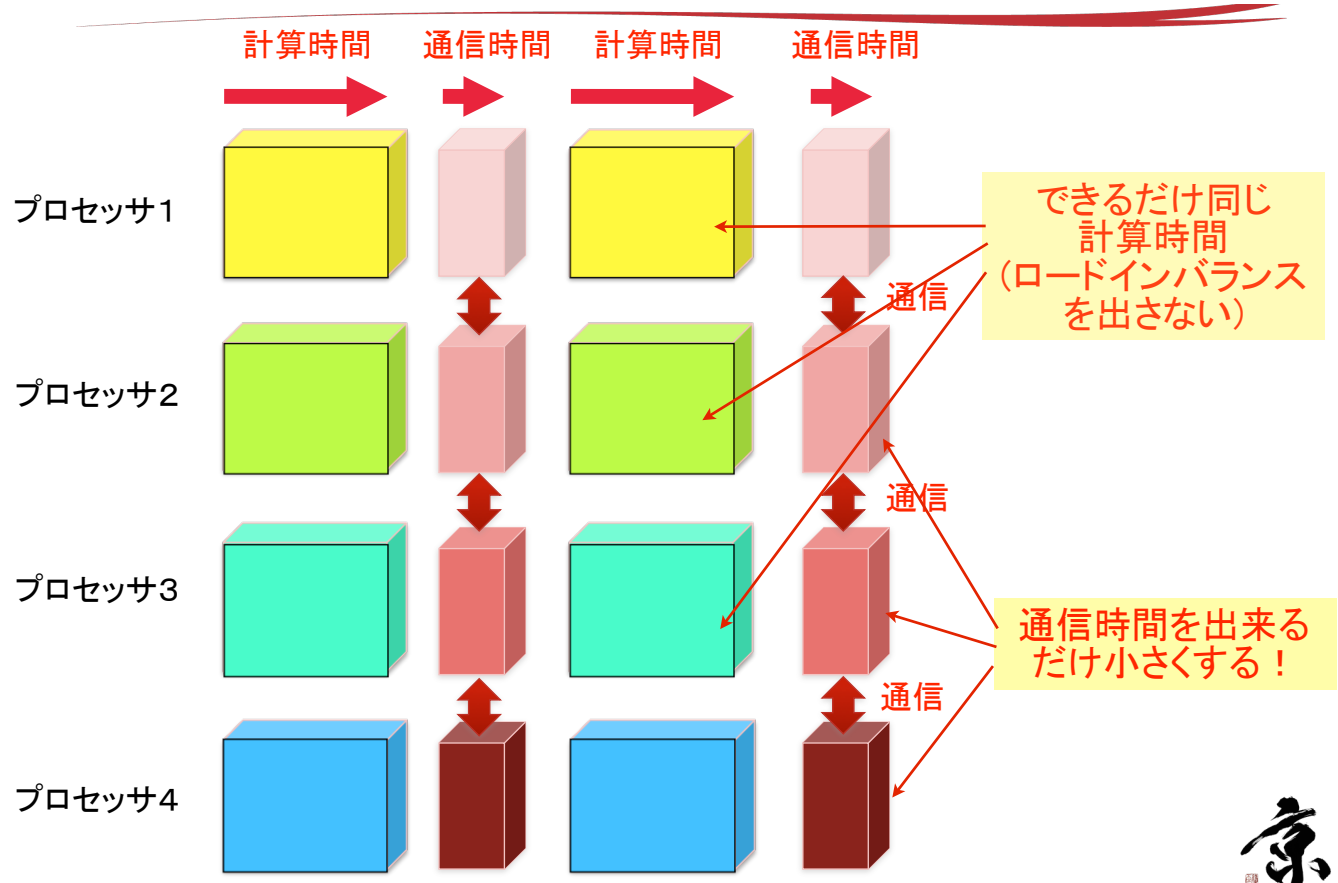


高並列化のための重要点

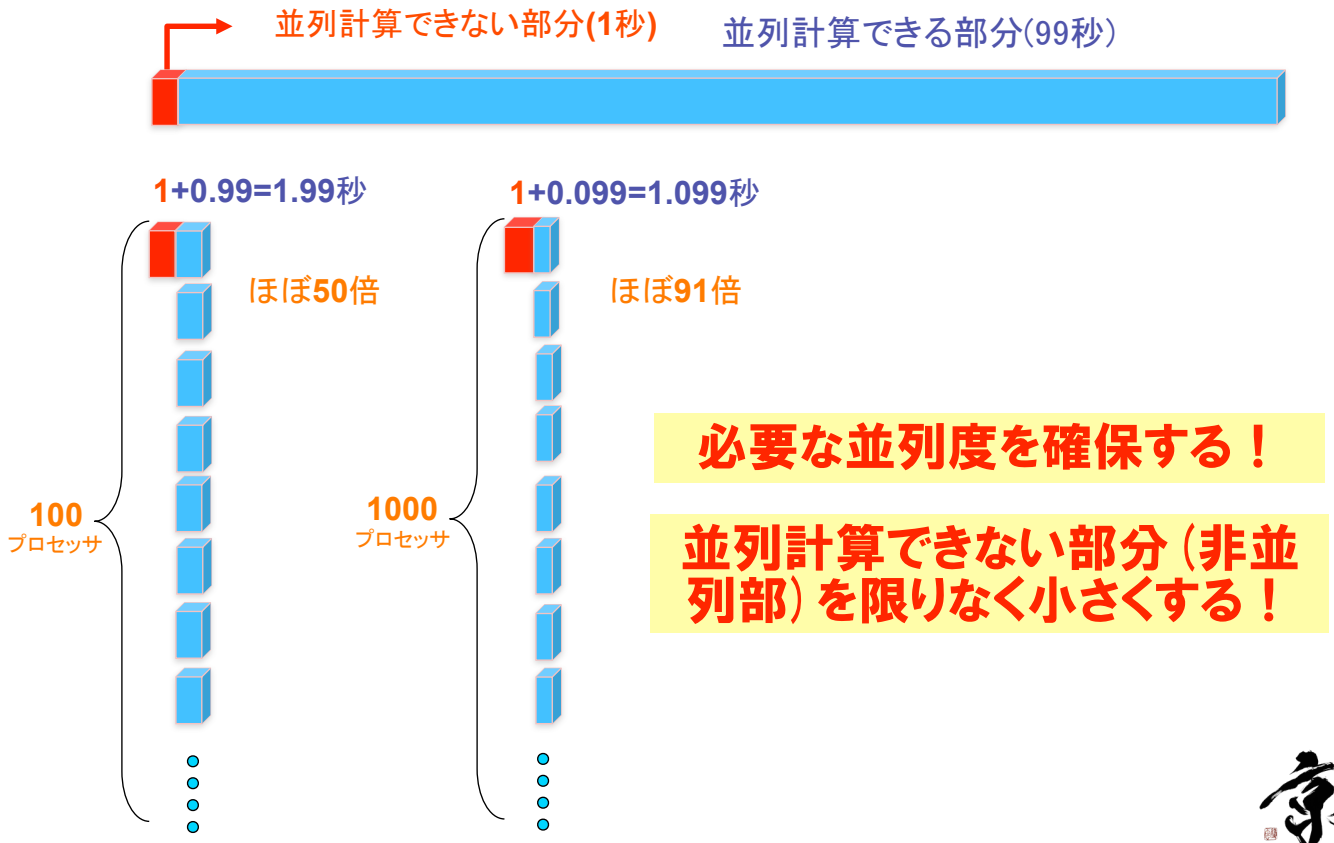
そもそも並列化とは？（1）



そもそも並列化とは？(2)



そもそも並列化とは？(3)

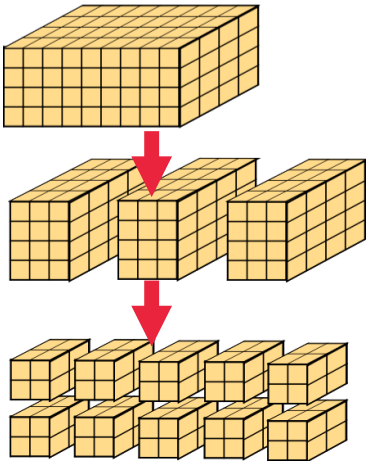


必要な並列度を確保する！

並列計算できない部分（非並列部）を限りなく小さくする！

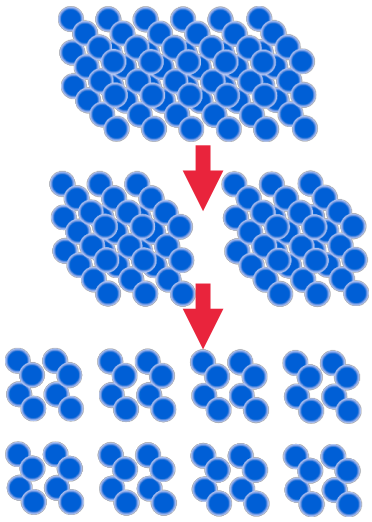
必要な並列度を確保するとは？

領域分割



・原理的にメッシュ数以上には分割できない
・実際的にはそんなに分割すると通信ばかりになる
・以下の手順で分割数を見積もる事が重要
(1) 解きたいメッシュ数を設定し (2) 実行時間を見積もる
(3) 解きたい時間を設定し (4) 分割数を設定する
(5) 分割数が多すぎる場合，並列数の拡大を検討
(5) については全てのケースでできる訳ではないが後の講義でテクニックを例示する．

原子分割



・原理的に原子数以上には分割できない
・実際的にはそんなに分割すると通信ばかりになる
・後は領域分割と同様

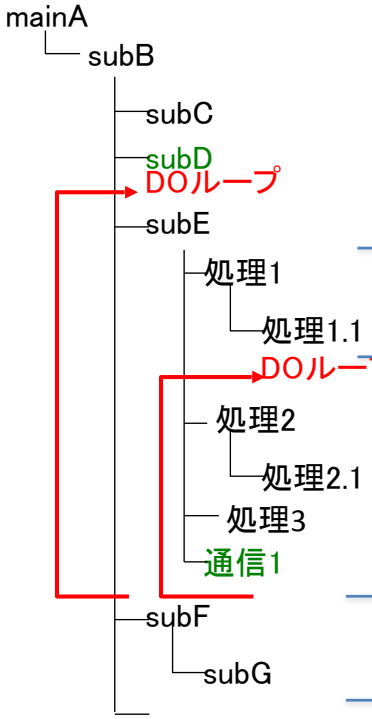
アプリケーションを高並列性を評価するためには？

- ✓ 十分な並列度を得る並列化手法を採用する
- ✓ 非並列部分を最小化する
- ✓ 通信時間を最小化する
- ✓ ロードインバランスを出さない

**最初の並列特性分析のフェーズ
および高並列化のステップで
アプリケーションの高並列を阻害する
要因を洗い出す事が重要**

1. 並列特性分析
(処理構造分析・ブロック特性分析)

(1)コードの構造を分析し物理に沿った処理ブロック(計算/通信)に分割
(2)コードの実行時間の実測
(3)プログラムソースコードの調査
(4)処理ブロックの物理的処理内容を把握
(5)計算ブロック毎の計算特性把握(非並列/完全並列/部分並列、Nに比例/N**2に比例等)
(6)通信ブロック毎の通信特性把握 (グローバル通信・隣接通信、隣接面に比例&隣接通信/体積に比例、等)



	実行時間・スケーラビリティ	物理的処理内容	演算・通信特性	演算・通信見積り	カーネル
ブロック1（計算）			部分並列	Nに比例	
ブロック2（計算）			完全並列	N**3に比例	○
（通信）			隣接通信	隣接面に比例	○
ブロック3					

2. カーネル評価

(1)計算・通信ブロックについて物理的処理内容・コーディングの評価を行い同種の計算・通信ブロックを評価し異なる種類の計算・通信ブロックをカーネルの候補として洗い出す
(2)並列特性分析の結果から得た問題規模に対する依存性の情報を元にターゲット問題実行時に、また高並列実行時にカーネルとなる計算・通信カーネルを洗い出す

超高並列を目指した場合の留意点–ブロック毎に以下を評価する

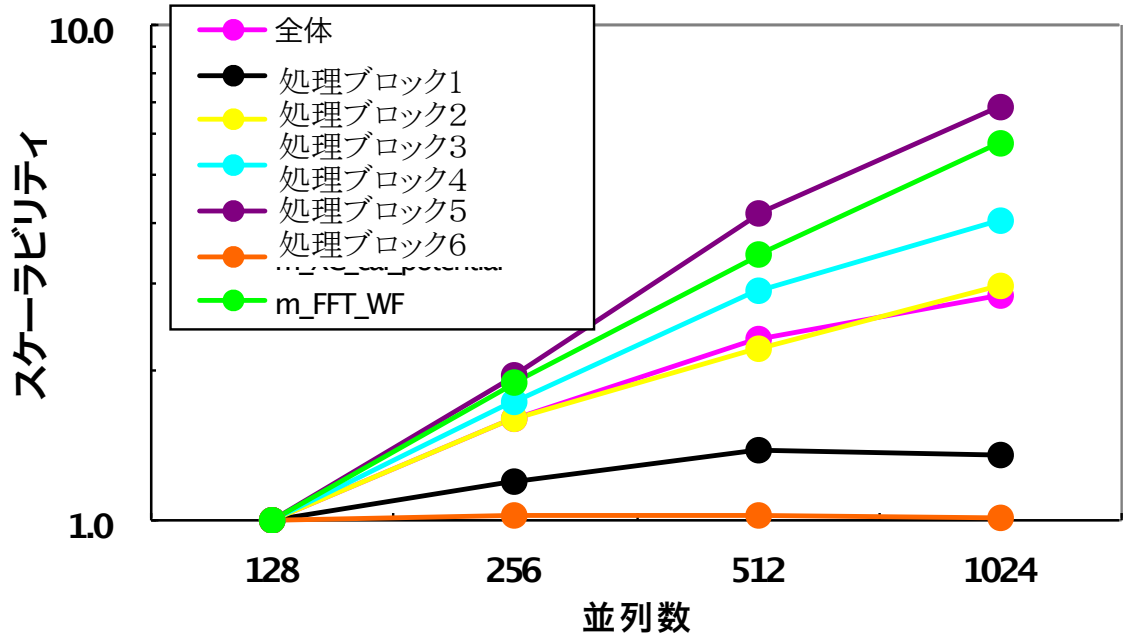
- 非並列部が残っていないか？残っている場合に問題ないか？
- 隣接通信時間が超高並列時にどれくらいの割合を占めるか？
- 大域通信時間が超高並列時にどれくらい増大するか？
- ロードインバランスが超高並列時に悪化しないか？

→ これらの評価が重要

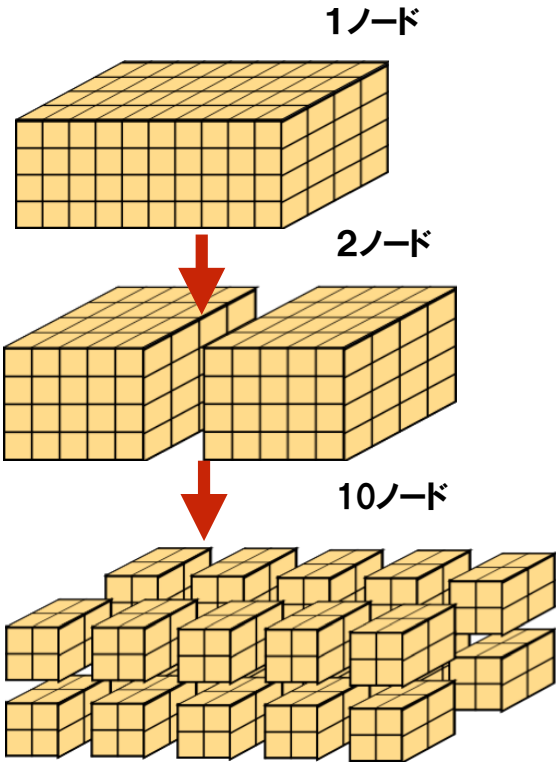
そのために

ブロック毎の実行時間とスケーラビリティ評価(例)

→従来の評価はサブルーチン毎・関数毎等の評価が多い
→サブルーチン・関数は色々な場所で呼ばれるため正しい評価ができない

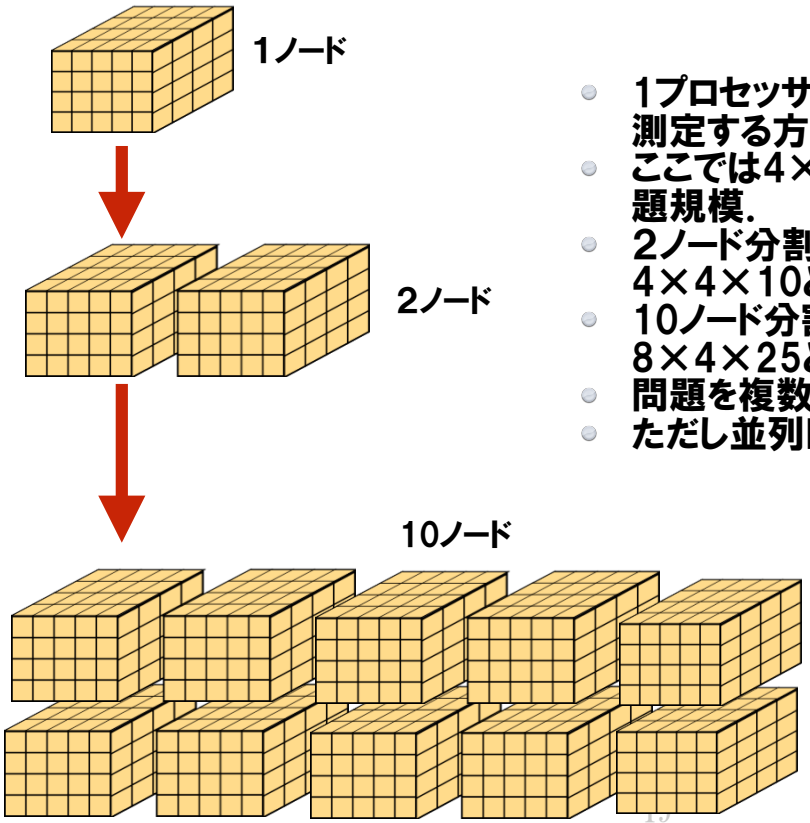


ストロングスケール測定



- 全体の問題規模を一定にして測定する方法.
- ここでは4×4×10が全体の問題規模.
- 2ノード分割では1ノードあたりの問題規模は4×4×5となる.
- 10ノード分割では1ノードあたりの問題規模は2×2×2となる.
- 問題を1種類作れば良いので測定は楽である.
- 並列時の挙動は見えにくい.

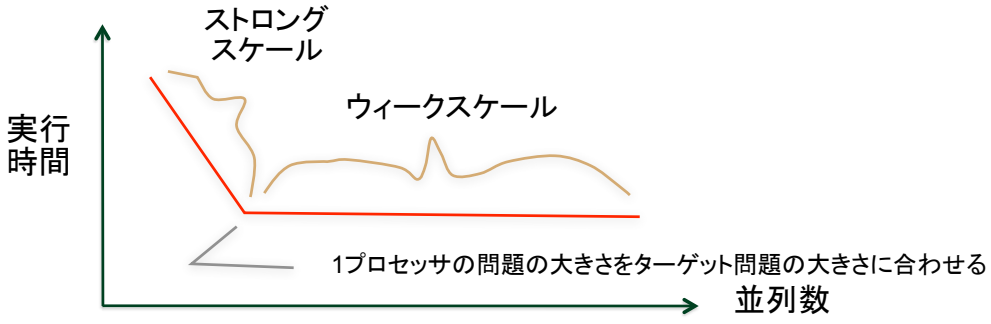
ウィークスケーリング測定



- 1プロセッサあたりの問題規模を一定にして測定する方法.
- ここでは4×4×5が1プロセッサあたりの問題規模.
- 2ノード分割では全体の問題規模は4×4×10となる.
- 10ノード分割では全体の問題規模は8×4×25となる.
- 問題を複数作る必要が測定は煩雑である.
- ただし並列時の挙動が見え易い.

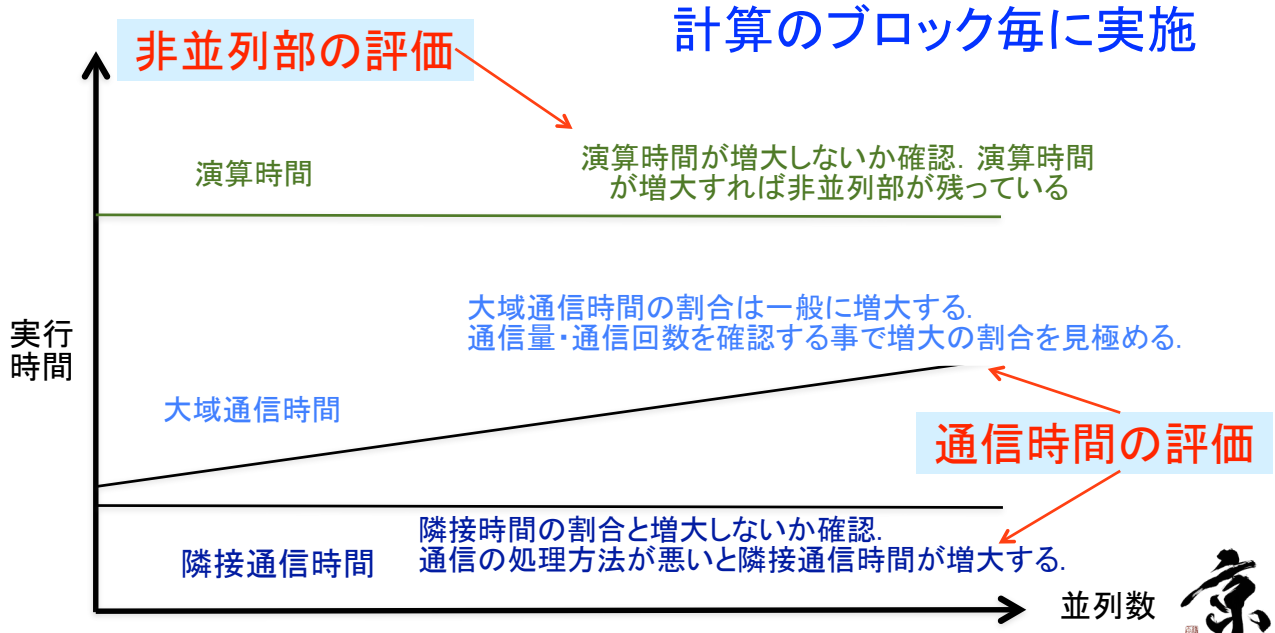
ストロングスケールとウィークスケーリング測定を使う

- ターゲット問題を決める
- 1プロセッサの問題規模がターゲット問題と同程度となるまでは、ストロングスケーリングで実行時間・ロードインバランス・隣接通信時間・大域通信時間を測定・評価する(100から数百並列まで)
- 上記もウィークスケーリングでできればなお良い
- 上記の測定・評価で問題が有れば解決する
- 問題なければ並列度を上げてウィークスケーリングで大規模並列の挙動を測定する

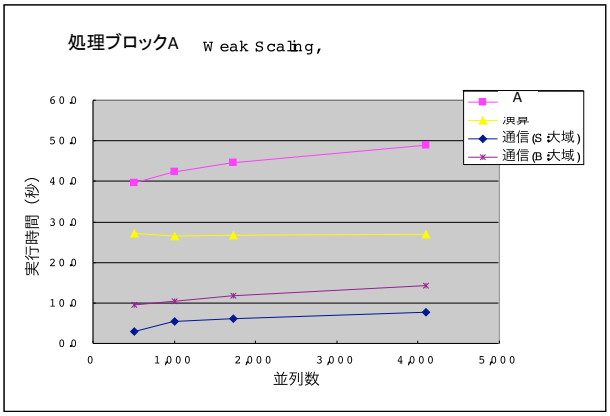


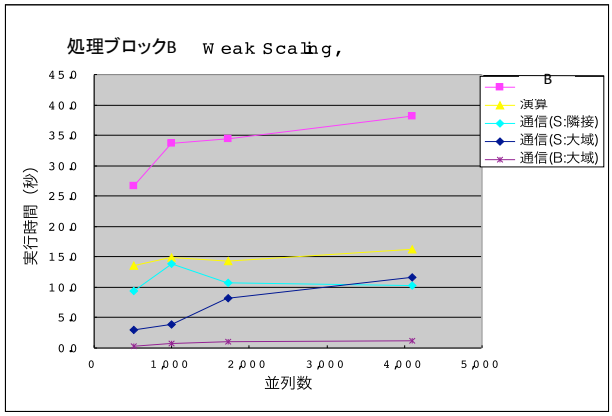
大規模並列のウィークスケーリング評価

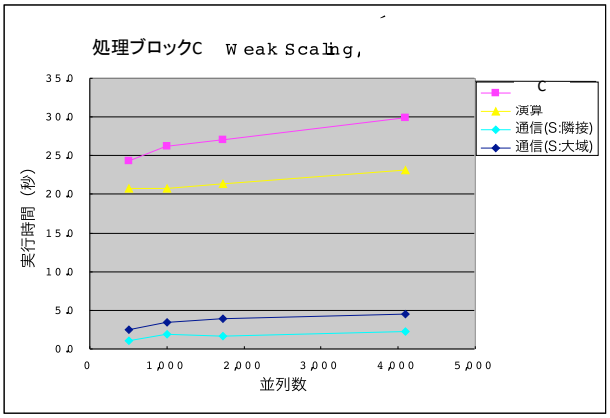
- 現状使用可能な実行環境を使用し100程度/1000程度/数千程度と段階を追ってできるだけ高い並列度で並列性能を確認する(ウィークスケーリング測定).
- ウィークスケーリングが難しいものもあるが出来るだけ測定したい. 難しい場合は, 演算時間をモデル化して実測とモデルとの一致を評価する.

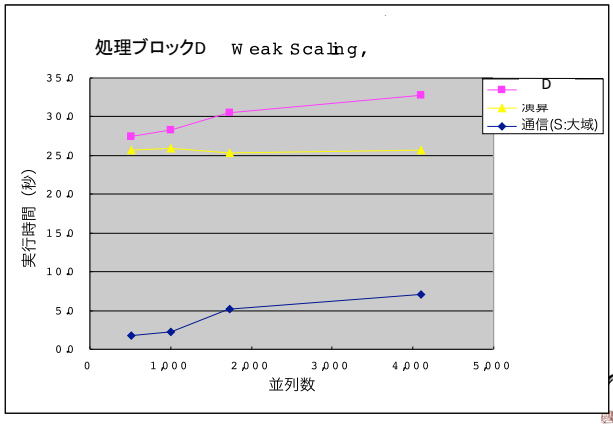


Weak Scaling測定例

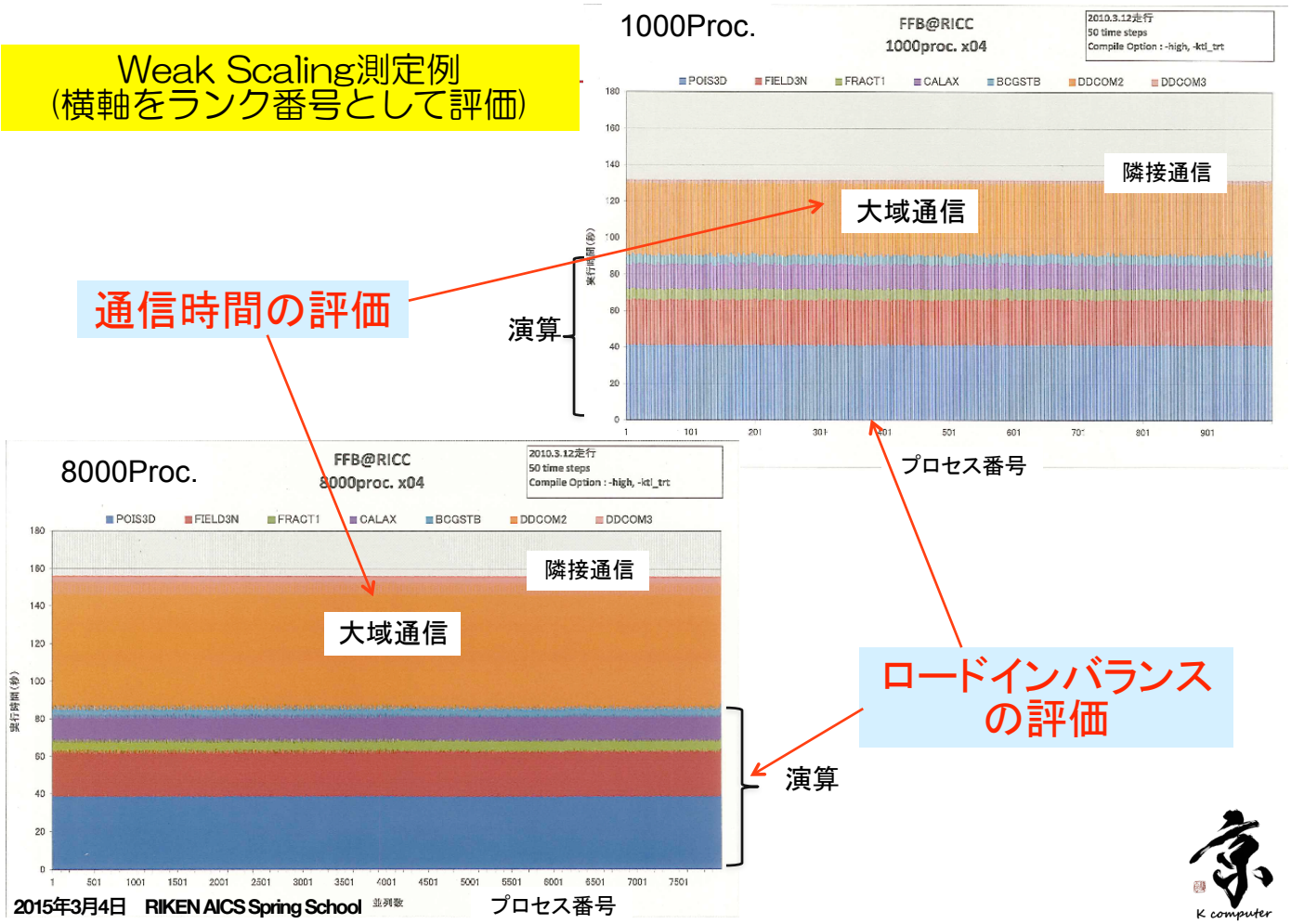




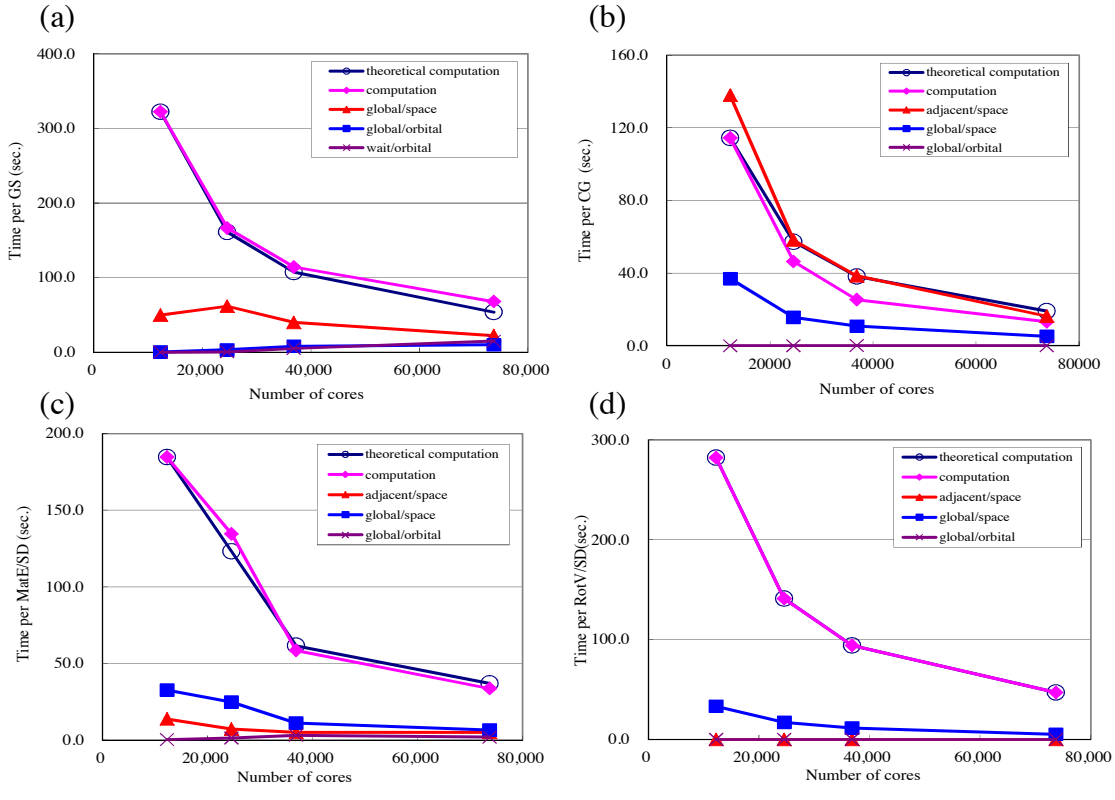




Weak Scaling測定例
(横軸をランク番号として評価)



ストロングスケーリングだけどスケールをモデル化した例

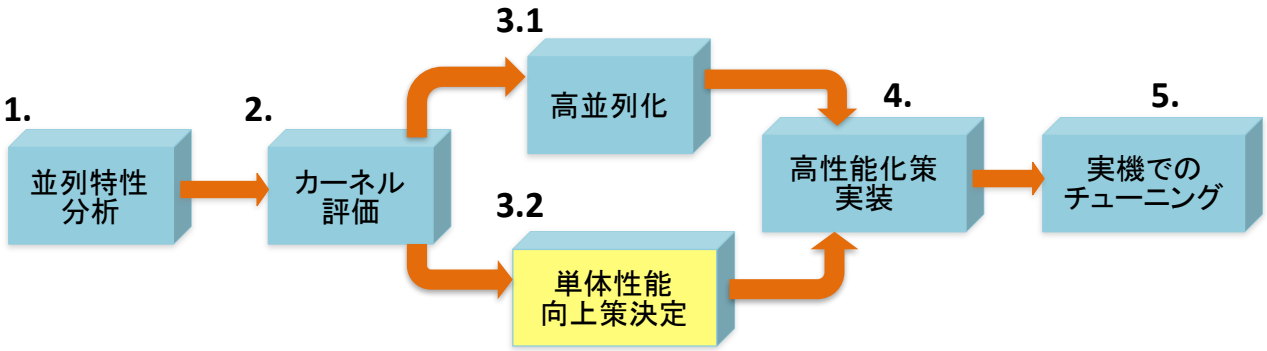


並列性能上のボトルネック

- 今まで示した調査を実施することにより処理ブロック毎に並列性能上の問題がある事が発見される.
- それらを分析するとだいたい以下の6点に分類されると考える.

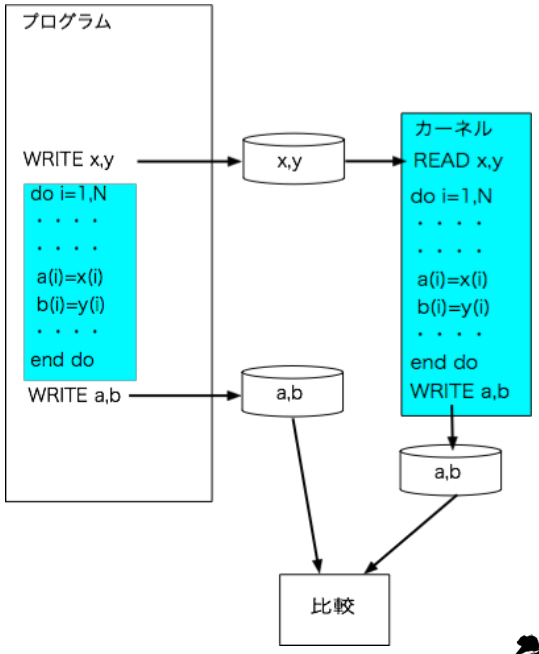
1	アプリケーションとハードウェアの並列度のミスマッチ (アプリケーションの並列度不足)
2	非並列部の残存
3	大域通信における大きな通信サイズ、通信回数の発生
4	フルノードにおける大域通信の発生
5	隣接通信における大きな通信サイズ、通信回数の発生
6	ロードインバランスの発生

単体性能向上策決定



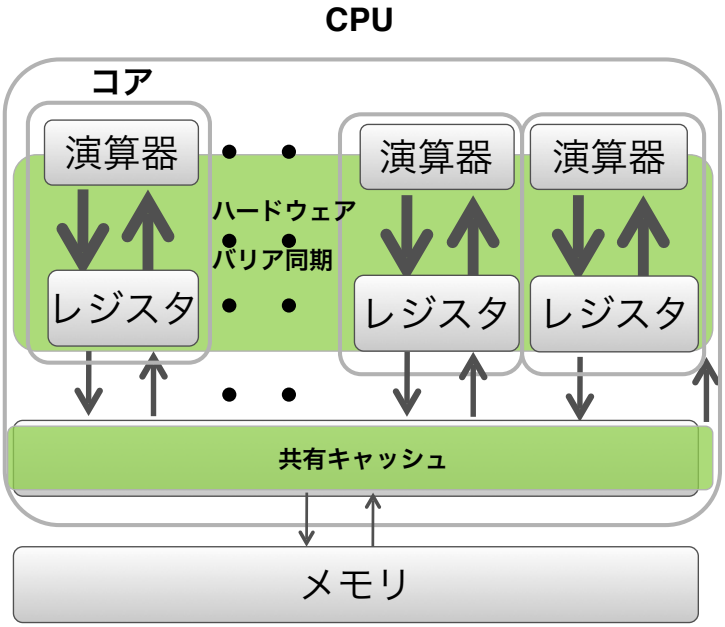
カーネルの切り出しと性能向上策の試行

- 計算カーネルの切り出し→計算カーネルを独立なテストプログラムとして切り出す.
- 性能向上策の試行→切り出したテスト環境を使用し様々な性能向上策を試行する.
- 性能向上策の評価・決定→試行結果を評価し性能向上策の案を策定する.
- 作業量見積り→性能向上策を実施した場合のコード全体に影響する作業を洗い出す.
- それら作業を実施した場合の作業量を見積る.
- それらを評価し最終的に採用する案を決定する.



スレッド並列化

スレッド並列の概要



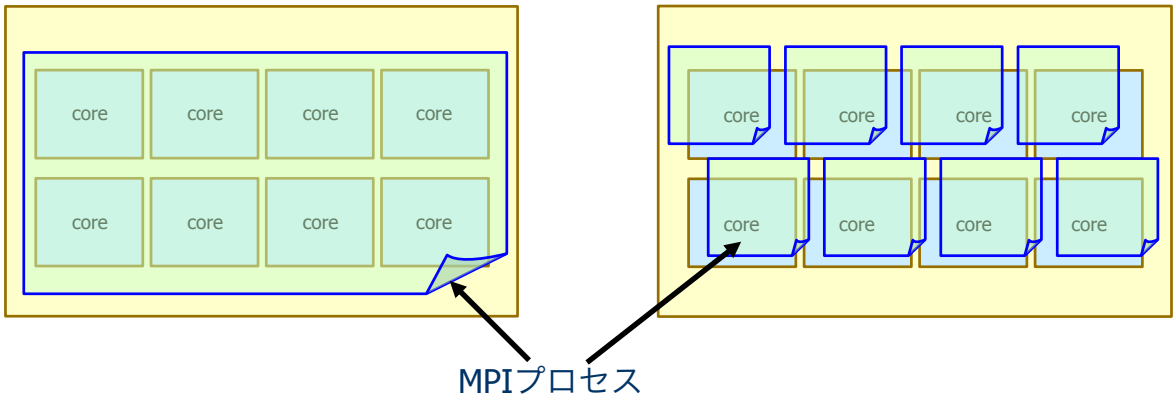
- 京の場合1CPUに8コア搭載.
- 8コアでL2キャッシュを共有.
- MPI等のプロセス並列に対しCPU内の8コアを使用するためのスレッド並列化が必要.
- スレッド並列化のためには自動並列化かOpenMPが使用可.
- 京の場合はハードウェアバリア同期機能を使用した高速なスレッド処理が可能.

ハイブリッド並列とフラットMPI並列

- MPIプロセス並列とスレッド並列の組み合わせをハイブリッド並列という.
- 各コアにMPIプロセスを割り当てる並列化をフラットMPI並列という.
- 京では通信資源の効率的利用，消費メモリ量を押さえる観点でハイブリッド並列を推奨している.

1プロセス8スレッド
（ハイブリッドMPI）の場合

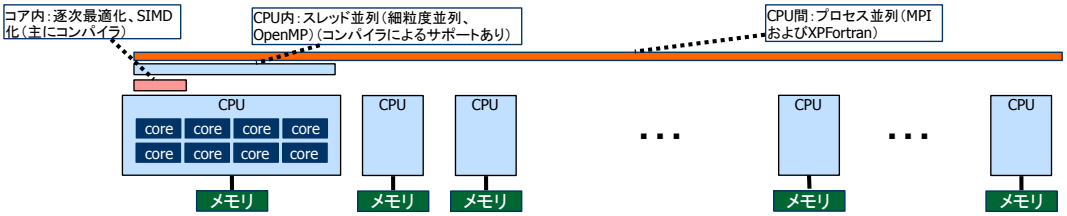
8プロセス（フラットMPI）の場合



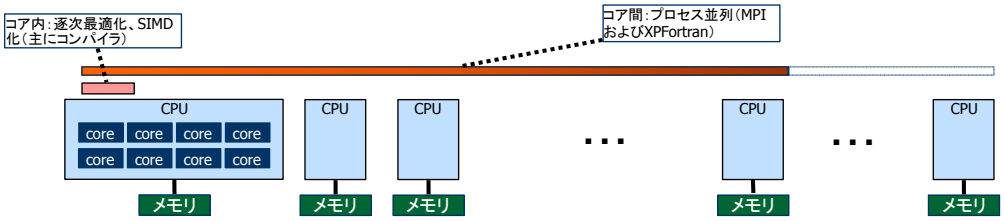
ハイブリッド並列とフラットMPI並列（京の場合）

- スレッド並列＋プロセス並列のハイブリッド型
  - コア内：コンパイラによる逐次最適化，SIMD化
  - CPU内：スレッド並列(自動並列化：細粒度並列化†，OpenMP)
  - CPU間：プロセス並列(MPI、XPFortran)

†細粒度並列化
高速バリア同期を活用し，内側ループをコア間で並列化
ベクトル向け（内側ループが長い）コードの高速化が可能



- プロセス並列型
  - コア内：コンパイラによる逐次最適化，SIMD化
  - コア間：プロセス並列(MPI、XPFortran)

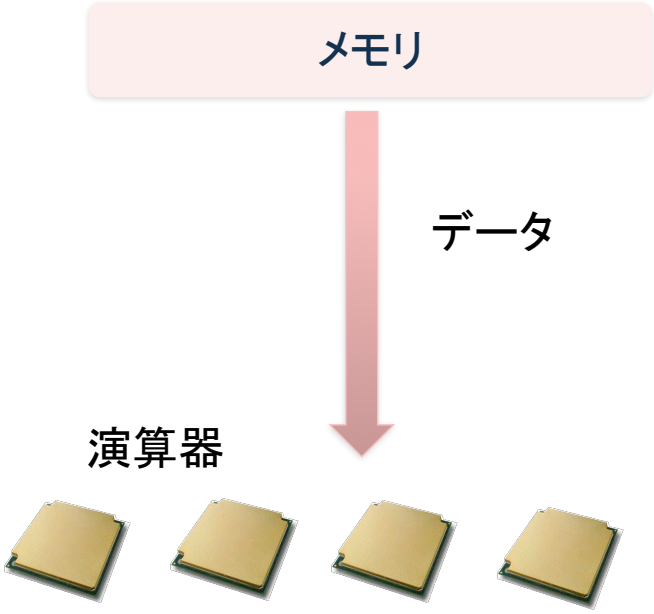


実行効率の観点から、ハイブリッド型を推奨

単体性能向上のための重要点

プロセッサの単体性能を引き出す（1）

- かつては研究者やプログラマーは物理モデル式に忠実に素直にプログラミングすることが一般的であった



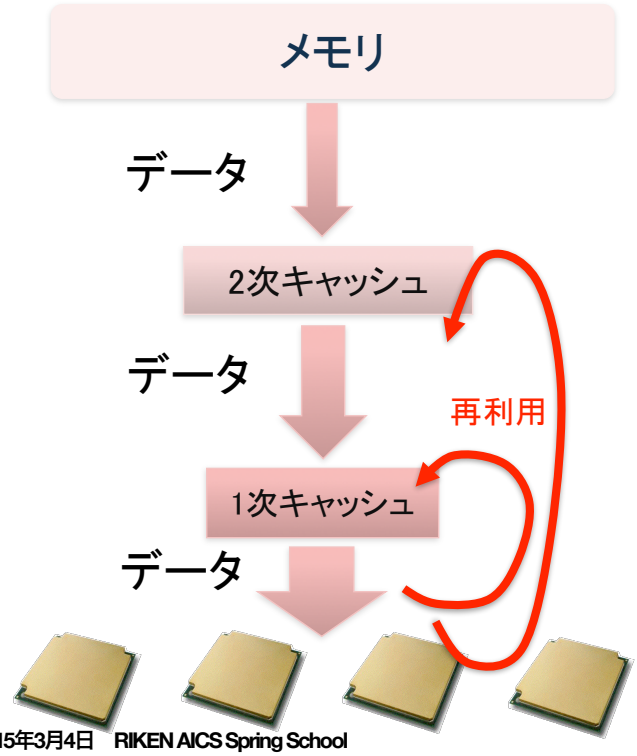
メモリウォール問題

✓ 昔の計算機はメモリのデータ供給能力と演算器の能力がバランスしていた

- 現代の計算機は演算器の能力が高くなりメモリのデータ供給能力が相対的に不足している

プロセッサの単体性能を引き出す（2）

メモリウォール問題への対処



・データ供給能力の高いキャッシュを設ける
・キャッシュに置いたデータを何回も再利用し演算を行なう
・こうする事で演算器の能力を十分使い切る

「アプリケーションが要求するByte/Flop値が低い」タイプの計算(*1)

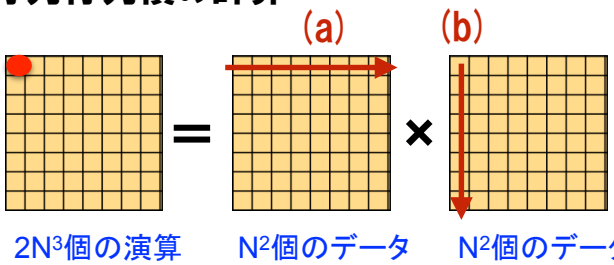
例えば行列行列積
($2N^2$個のデータで$2N^3$個の演算可)

(*1)演算量(Flop) に比べデータの移動量(Byte)が小さい計算

キャッシュの有効利用

もう少し詳しく説明すると

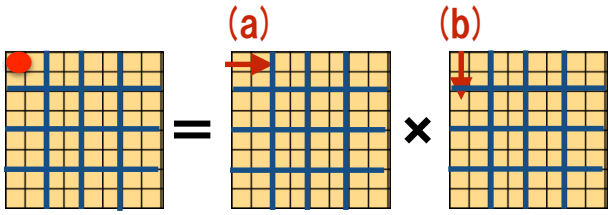
行列行列積の計算



B/F値
=移動量(Byte)/演算量(Flop)
=$2N^2/2N^3$
=1/N

原理的にはNが大きい程小さな値

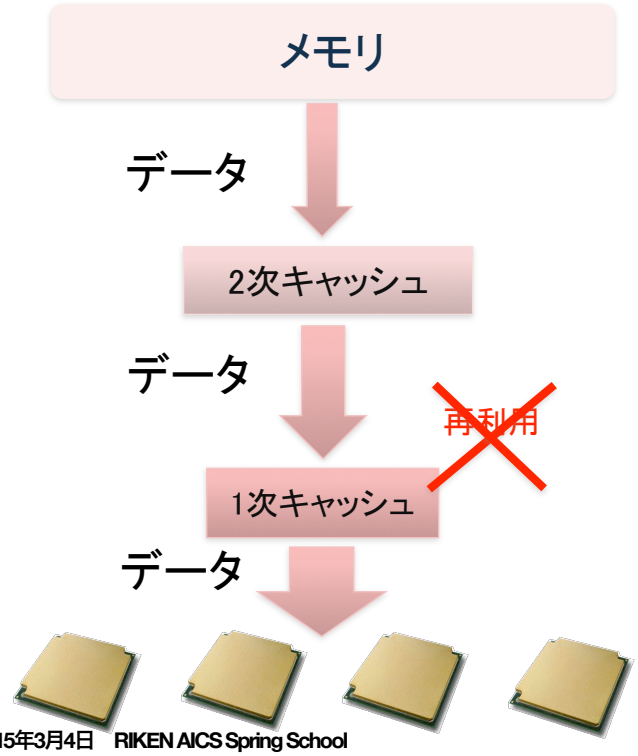
- **現実のアプリケーションではNがある程度の大きさになるとメモリ配置的には (a) はキャッシュに乗っても (b) は乗らない事となる**



- **そこで行列を小行列にブロック分割し (a) も (b) もキャッシュに乗るようにしてキャッシュ上のデータだけで計算するようにする事で性能向上を実現する.**

プロセッサの単体性能を引き出す (3)

とは言っても・・・・



・再利用出来ない問題もある
・こうなる演算器の能力を十分使い切る事が出来ない

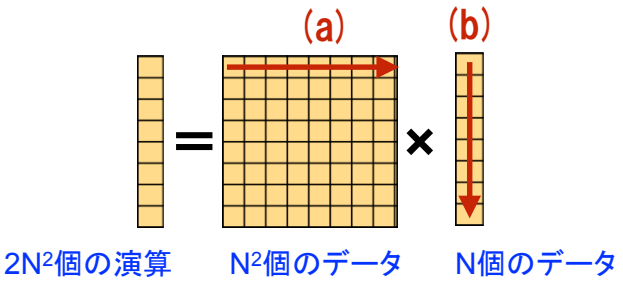
「アプリケーションが要求するByte/Flop値が高い」タイプの計算(*2)

(*2)演算量(Flop) に比べデータの移動量(Byte)が大きい計算

キャッシュの有効利用が難しい

例えば

行列ベクトル積の計算



B/F値
=移動量(Byte)/演算量(Flop)
=$(N^2+N)/2N^2$
≒1/2

原理的には1/Nより大きな値

- **行列を小行列にブロック分割して(a)も(b)もキャッシュに乗るようにしてもB/F値は大きいので性能向上はできない.**

CPU単体性能を上げるための5つの要素

CPU単体性能を上げるための5つの要素

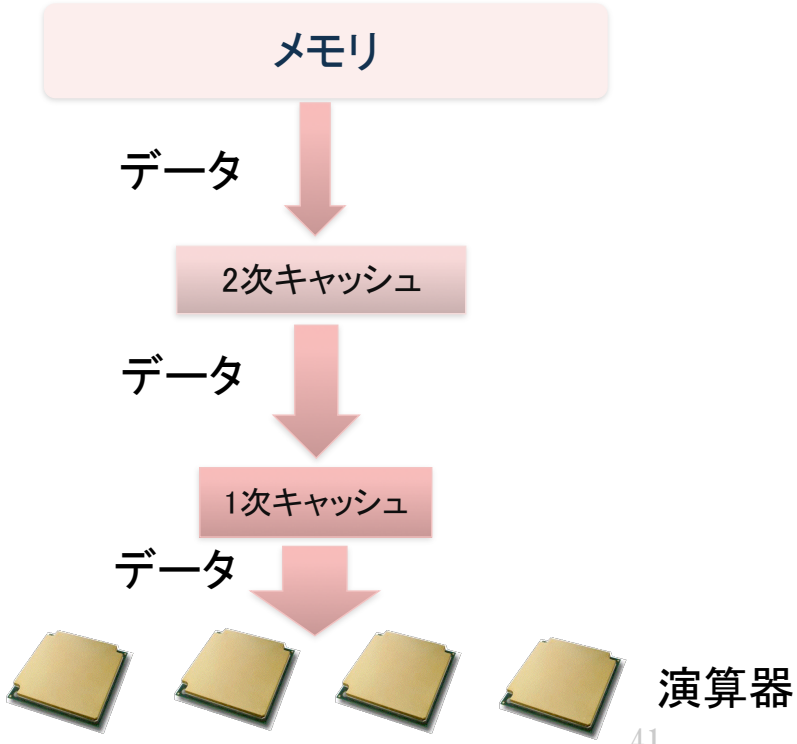
CPU内の複数コアでまずスレッド並列化が
できていると事は前提として

(1) ロード・ストアの効率化
(2) ラインアクセスの有効利用
(3) キャッシュの有効利用
(4) 効率の良い命令スケジューリング
(5) 演算器の有効利用

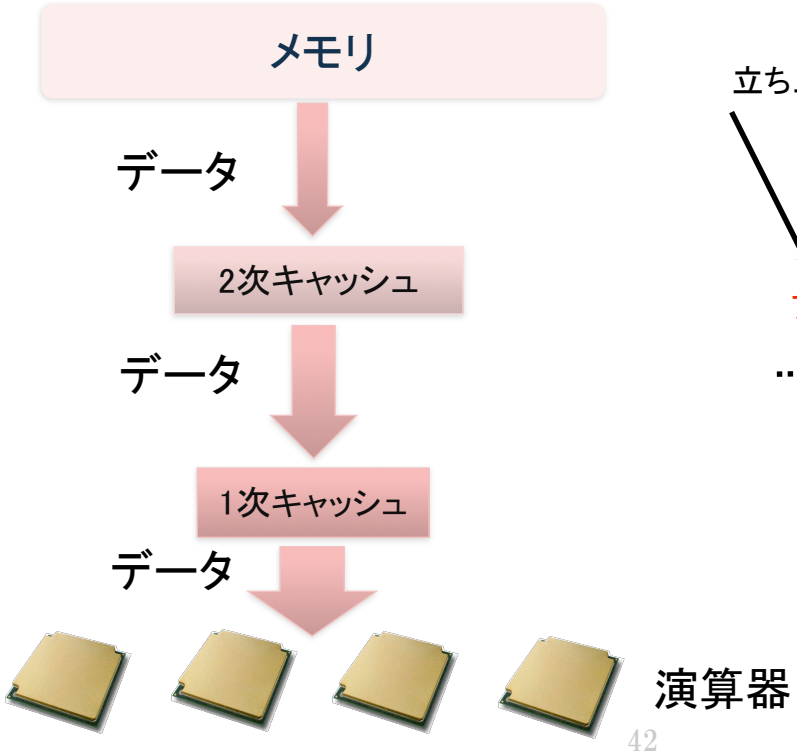
(1) ロード・ストアの効率化

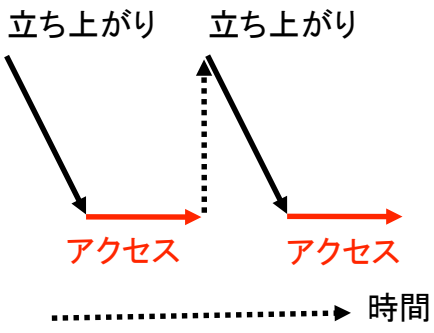
- プリフェッチの有効利用
- 演算とロード・ストア比の改善

レイテンシ(アクセスの立ち上がり)

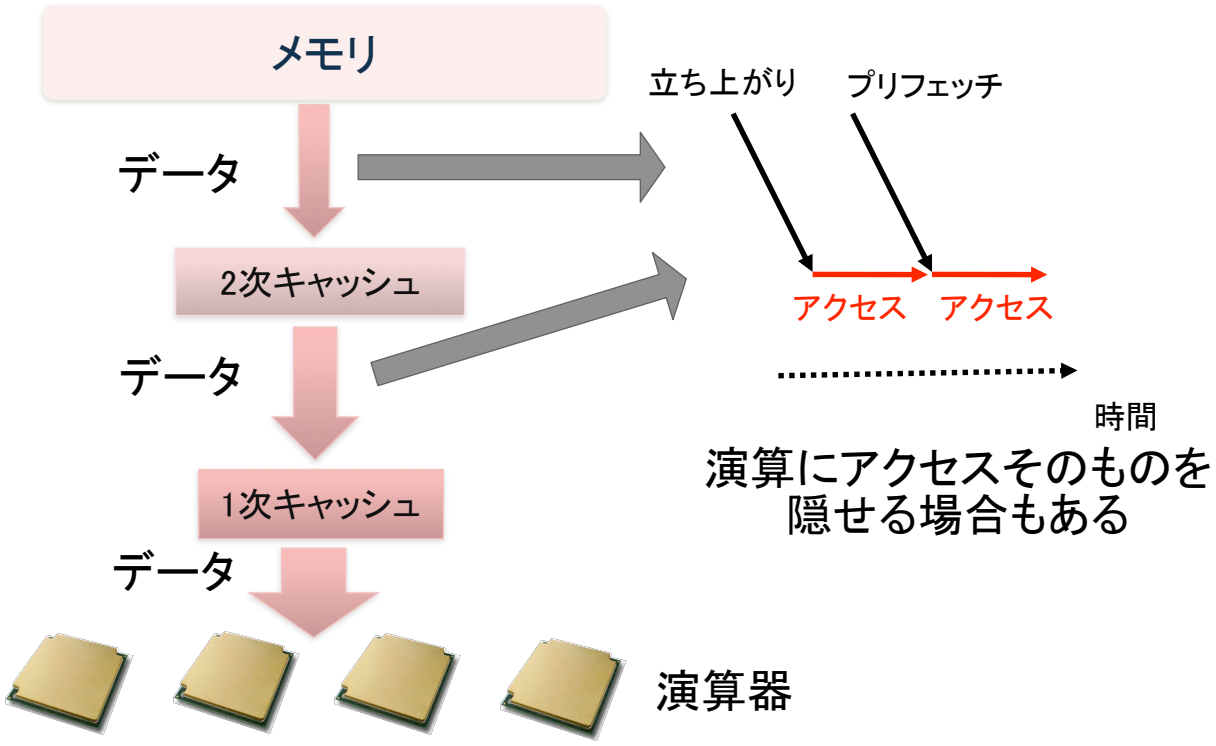


レイテンシ(アクセスの立ち上がり)





プリフェッチの有効利用

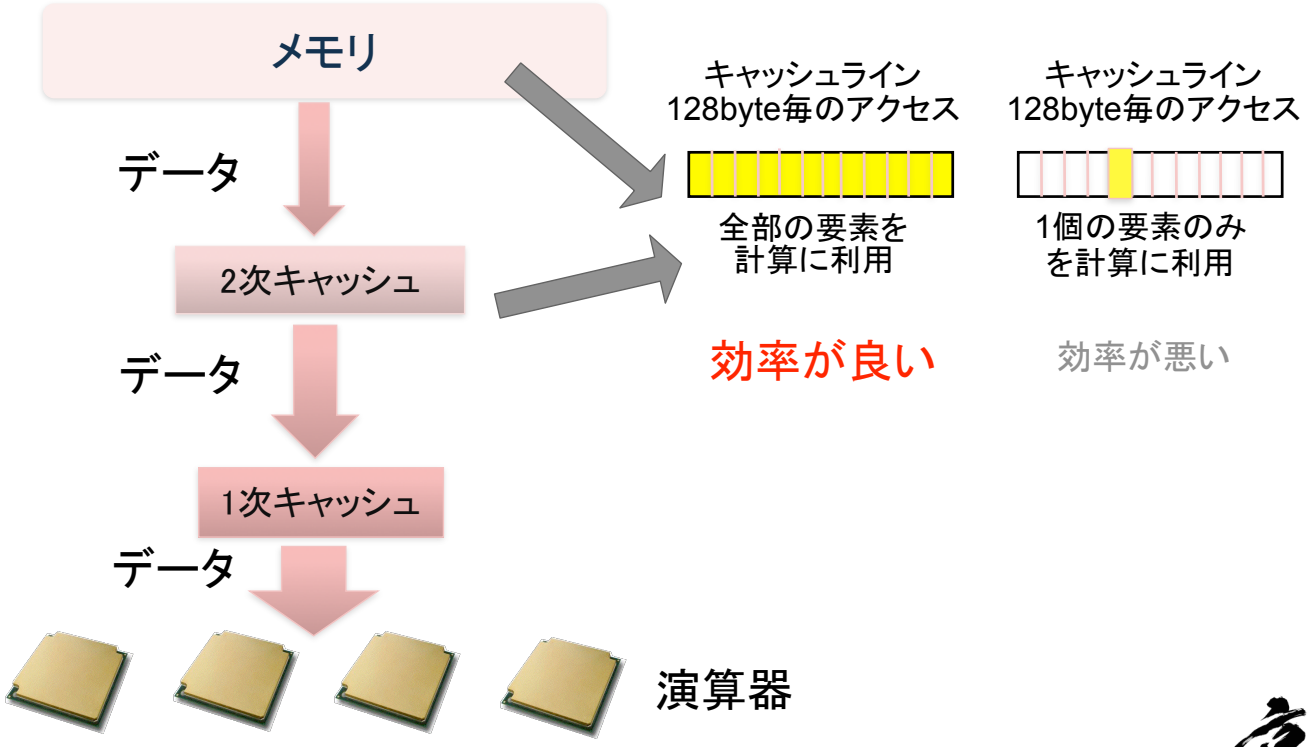


演算とロード・ストア比の改善

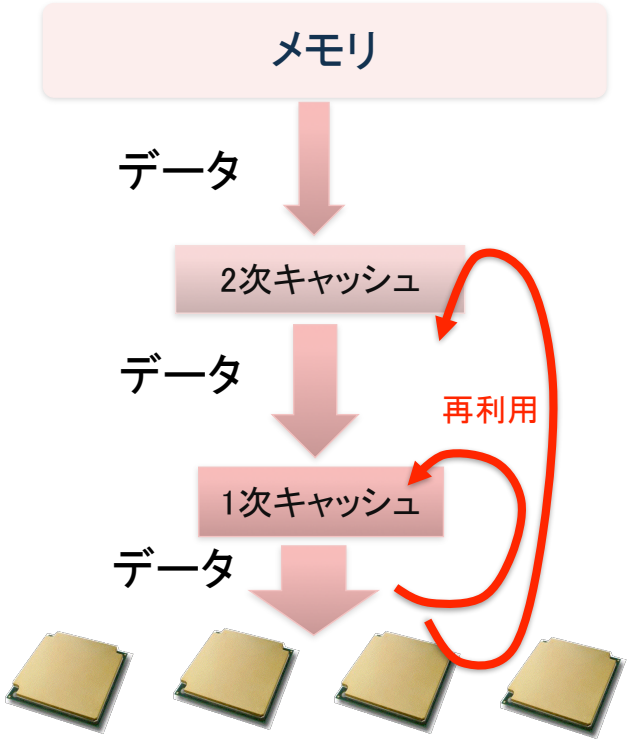
- 以下のコーディングを例に考える.
- 以下のコーディングの演算は和2個，積2個の計4個.
- ロードの数はx,a(i),a(i+1)の計3個.
- ストアの数はxの1個.
- したがってロード・ストア数は4個
- 演算とロード・ストアの比は4/4となる.
- なるべく演算の比率を高めロード・ストアの比率を低く抑えて演算とロードストアの比を改善を図る事が重要.

```
do  j=1,m
do  i=1,n
  x(i)=x(i)+a(i)*b+a(i+1)*d
end  do
```

(2) ラインアクセスの有効利用



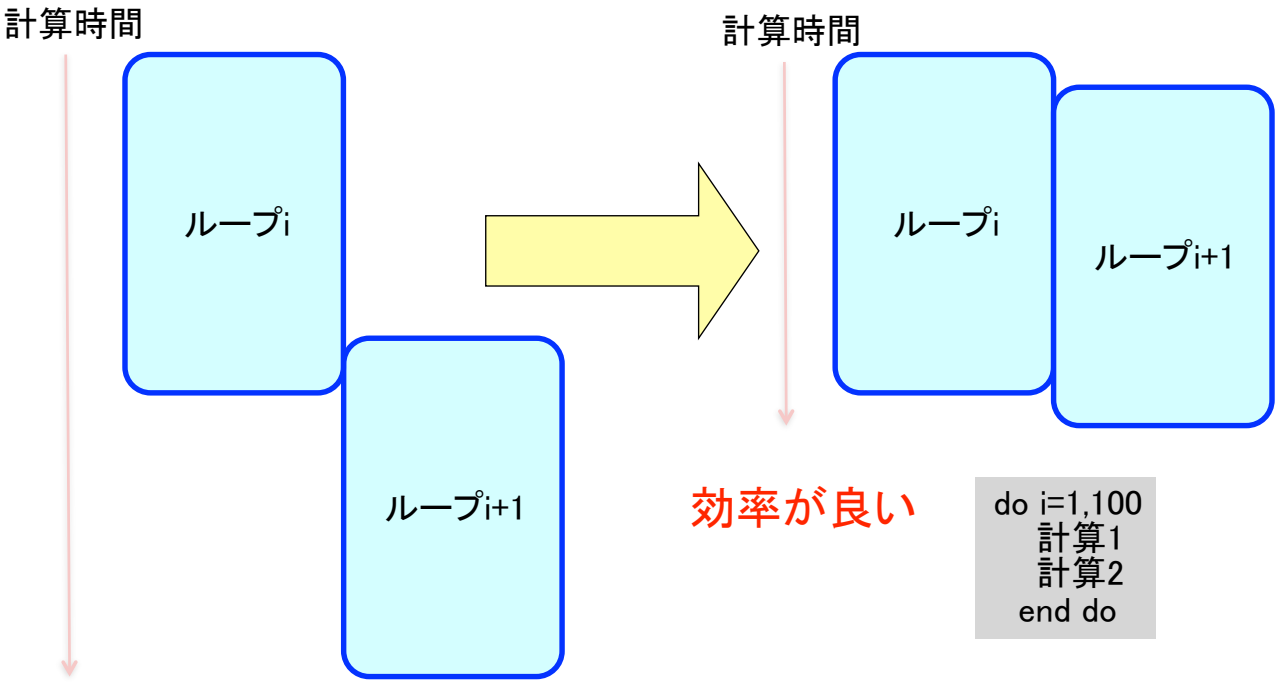
(3) キャッシュの有効利用



・データ供給能力の高いキャッシュを設ける
・キャッシュに置いたデータを何回も再利用し演算を行なう
・こうする事で演算器の能力を十分使い切る

- **ブロッキング**

(4) 効率の良い命令スケジューリング



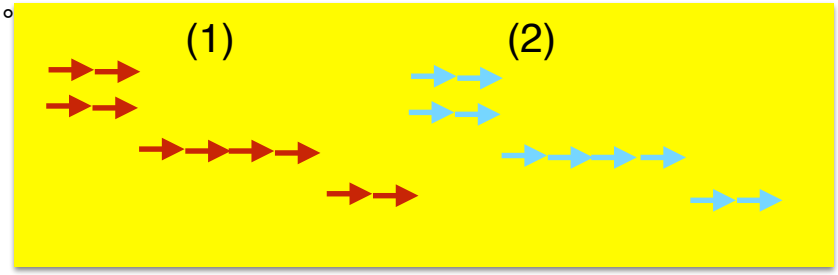
並列処理と依存性の回避

< ソフトウェアパイプラインニング> **コンパイラ**

<前提>
・ロード2つorストアと演算とは同時実行可能
・ロードとストアは2クロックで実行可能
・演算は4クロックで実行可能
・ロードと演算とストアはパイプライン化されている

例えば以下の処理を考える。

```
do i=1,100
   a(i)のロード
   b(i)のロード
   a(i)とb(i)の演算
   i番目の結果のストア
end do
```



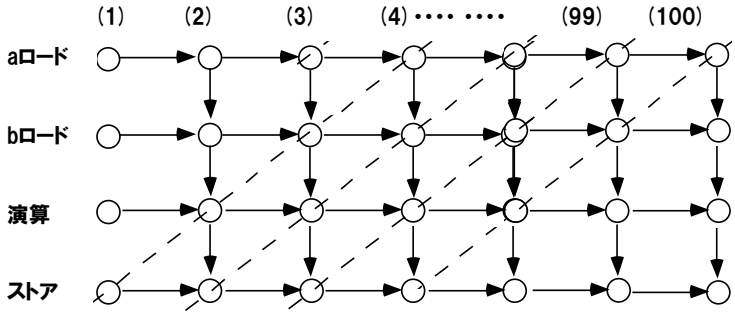
<実行時間>
・8クロック×100要素=800クロックかかる

並列処理と依存性の回避

< ソフトウェアパイプラインニング>　**コンパイラ**

左の処理を以下のように構成し直す事を
ソフトウェアパイプラインニングという

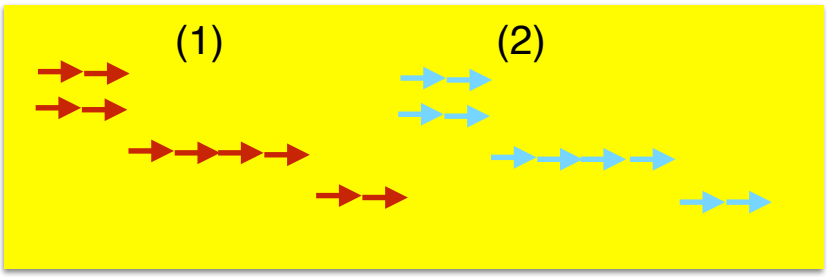
a(1)a(2)a(3)のロード
b(1)b(2)のロード
(1)の演算
do i=3,100
　a(i+1)のロード
　b(i)のロード
　(i-1)の演算
　i-2番目の結果のストア
end do
b(100)のロード
(99)(100)の演算
(98)(99)(100)のストア



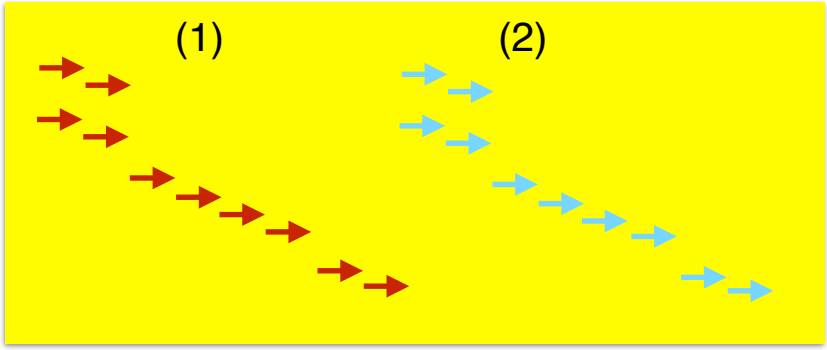
並列処理と依存性の回避

< ソフトウェアパイプラインニング>　**コンパイラ**

do i=1,100
　a(i)のロード
　b(i)のロード
　a(i)とb(i)の演算
　i番目の結果のストア
end do



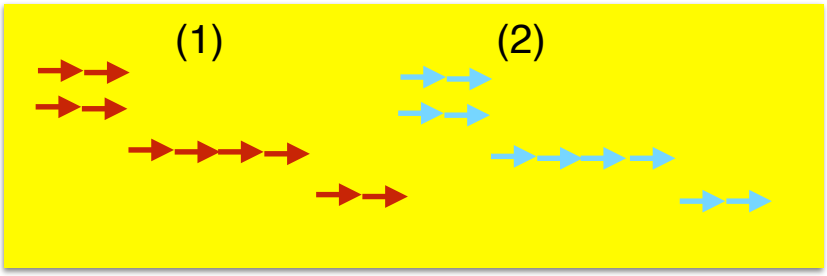
do i=1,100
　a(i)のロード
　b(i)のロード
　a(i)とb(i)の演算
　i番目の結果のストア
end do



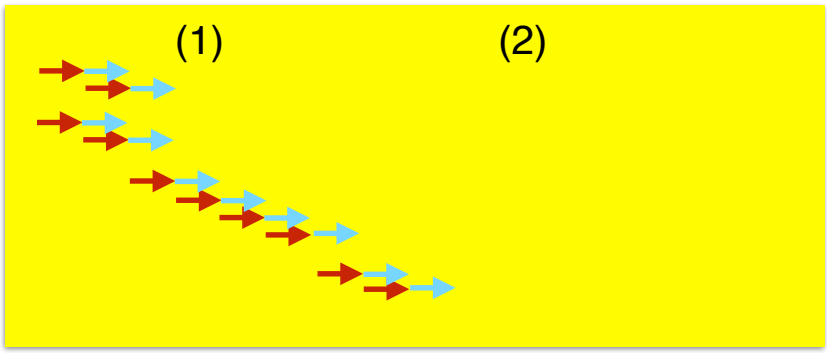
並列処理と依存性の回避

< ソフトウェアパイプラインニング>　　**コンパイラ**

```
do i=1,100
      a(i)のロード
      b(i)のロード
      a(i)とb(i)の演算
      i番目の結果のストア
end do
```




```
do i=1,100
      a(i)のロード
      b(i)のロード
      a(i)とb(i)の演算
      i番目の結果のストア
end do
```

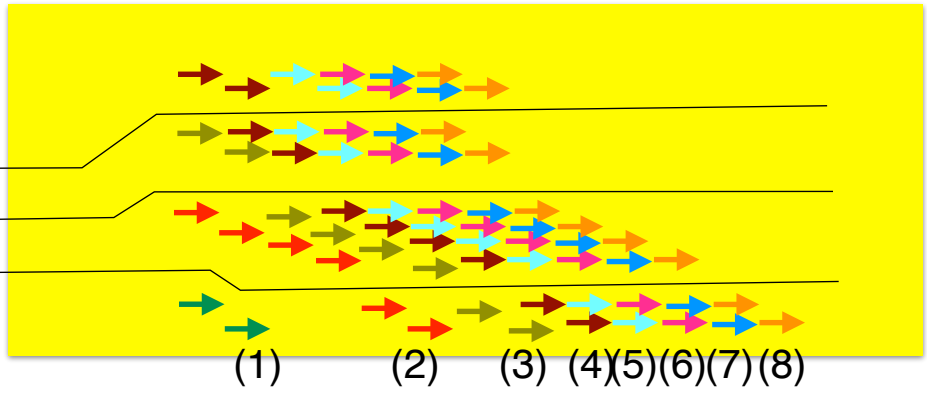


並列処理と依存性の回避

< ソフトウェアパイプラインニング>

左の処理を以下のように構成し直す事を
ソフトウェアパイプラインニングという

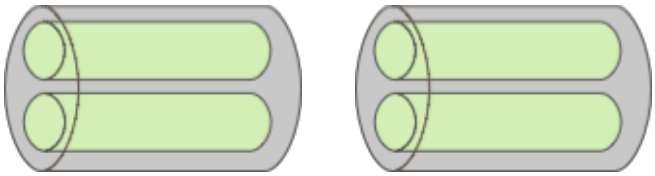
```
a(1)a(2)a(3)のロード
b(1)b(2)のロード
(1)の演算
do i=3,100
      a(i+1)のロード
      b(i)のロード
      (i-1)の演算
      i-2番目の結果のストア
end do
b(100)のロード
(99)(100)の演算
(98)(99)(100)のストア
```



<実行時間>
・前後の処理及びメインループの立ち上がり部分を除くと
・1クロック×100要素=100クロックで処理できる

(5) 演算器の有効利用

2 SIMD Multi&Add演算器×２本



乗算と加算を4個同時に計算可能

(1 + 1) × 4 = 8

この条件に
近い程高効率

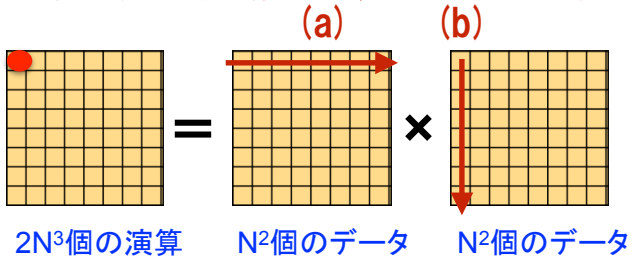
1コアのピーク性能：8演算×2GHz =16G演算/秒

要求B/F値と5つの要素の関係

要求B/F値と5つの要素の関係

アプリケーションの要求B/F値の大小によって性能チューニングにおいて注目すべき項目が異なる

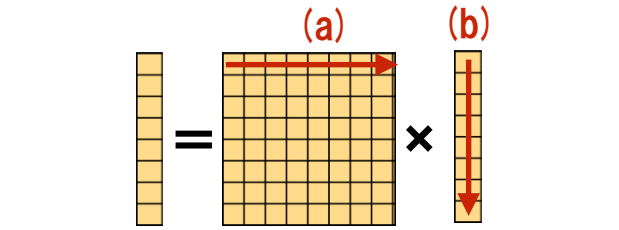
行列行列積の計算（要求B/F値が小さい）



B/F値
=移動量(Byte)/演算量(Flop)
$=2N^2/2N^3$
$=1/N$

原理的にはNが大きい程小さな値

行列ベクトル積の計算（要求B/F値が大きい）



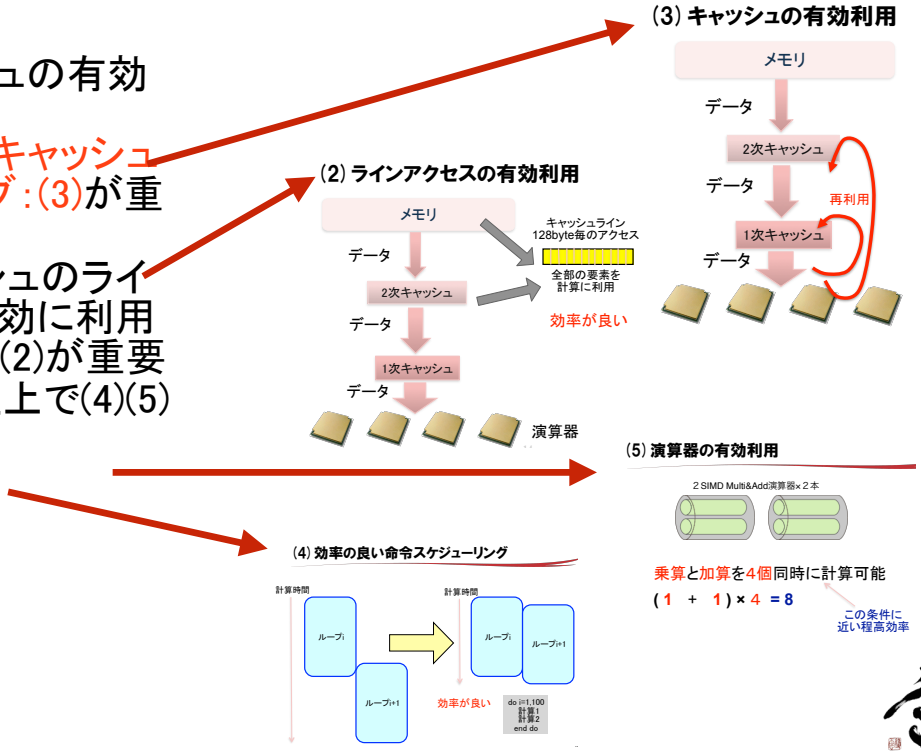
B/F値
=移動量(Byte)/演算量(Flop)
$=(N^2+N)/2N^2$
$\fallingdotseq 1/2$

原理的には1/Nより大きな値

要求B/F値と5つの要素の関係

要求するB/Fが小さいアプリケーションについて

- 原理的にキャッシュの有効利用が可能
- まずデータをオンキャッシュにするコーディング：(3)が重要
- つぎに2次キャッシュのライン上のデータを有効に利用するコーディング：(2)が重要
- それが実現できた上で(4)(5)が重要

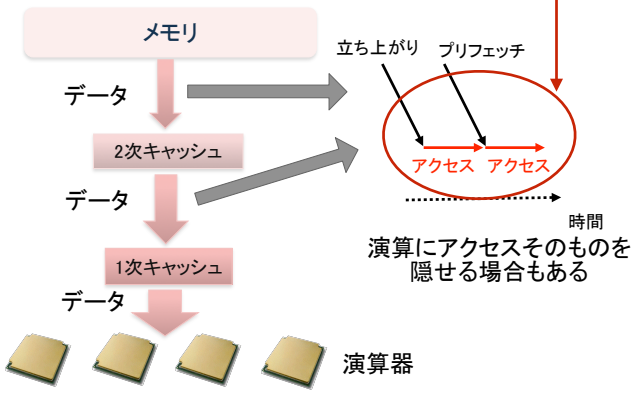


要求B/F値と5つの要素の関係

要求するB/Fが大きいアプリケーションについて

・メモリバンド幅を使い切る事が大事

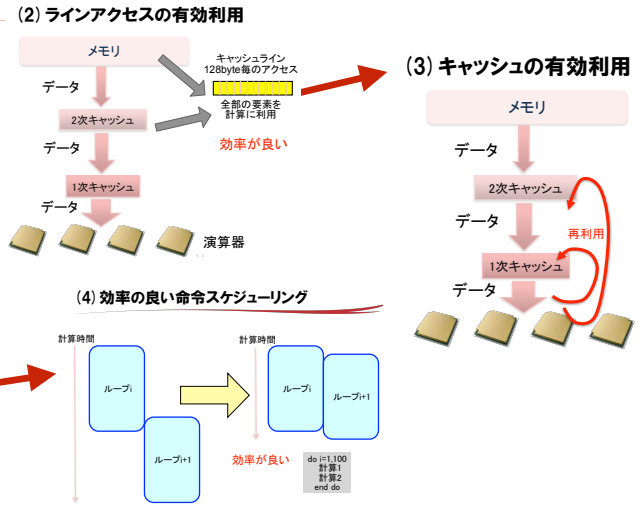
レイテンシーが隠れた状態にする事．この状態でメモリバンド幅のピーク(京の場合の理論値は64GB/sec)が出せる．レイテンシーが見えているとメモリバンド幅のピーク値は出せない．



要求B/F値と5つの要素の関係

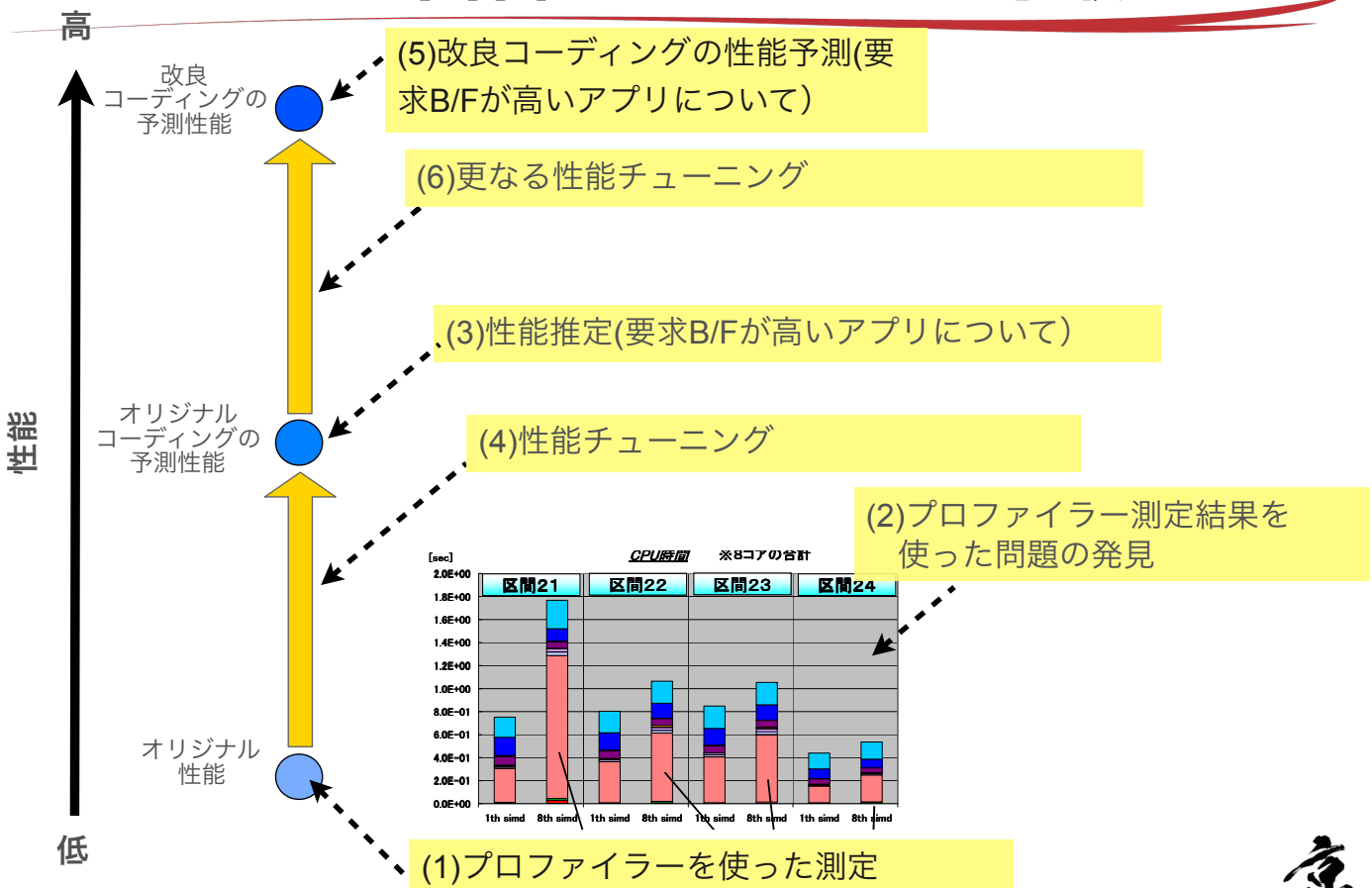
要求するB/Fが大きいアプリケーションについて

- 一番重要なのは(1)(2)
- 次にできるだけオンキャッシュする(3)が重要
- これら(1)(2)(3)が満たされ計算に必要なデータが演算器に供給された状態で、それらのデータを十分使える程度に(4)のスケジューリングができて、さらに(5)の演算器が有効に活用できる状態である事が必要

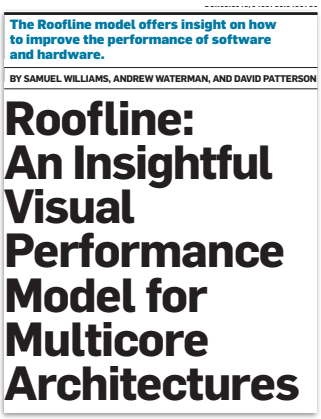


性能予測手法
(要求B/F値が大きい場合)

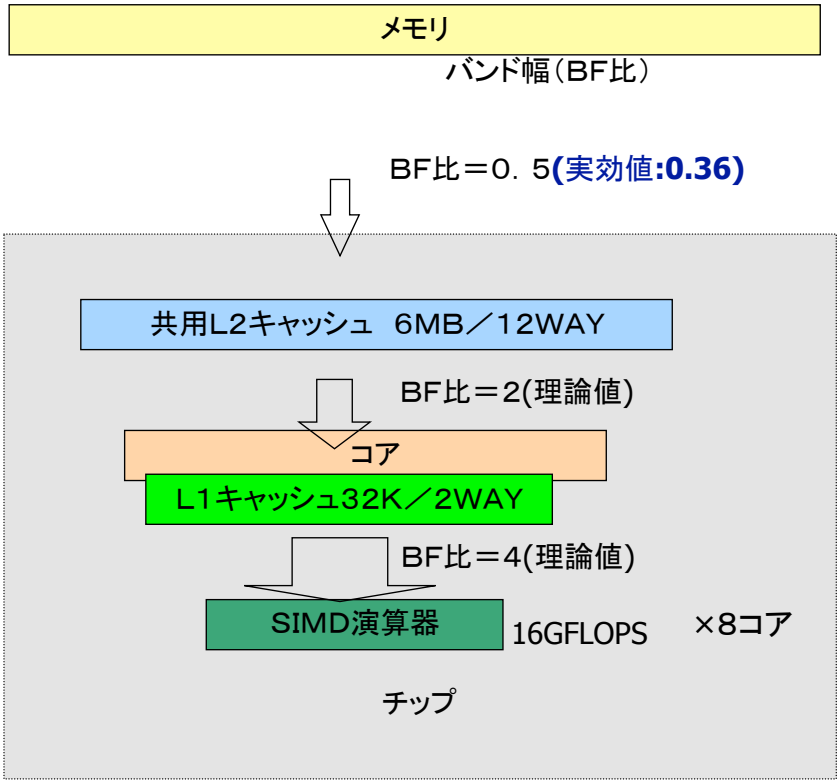
CPU単体性能チューニング手順

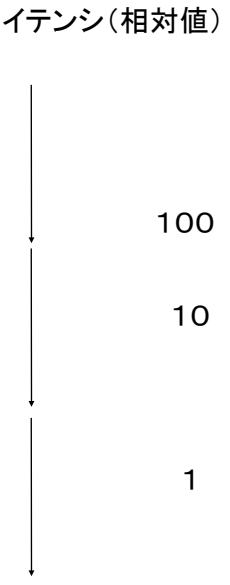


ルーフラインモデル

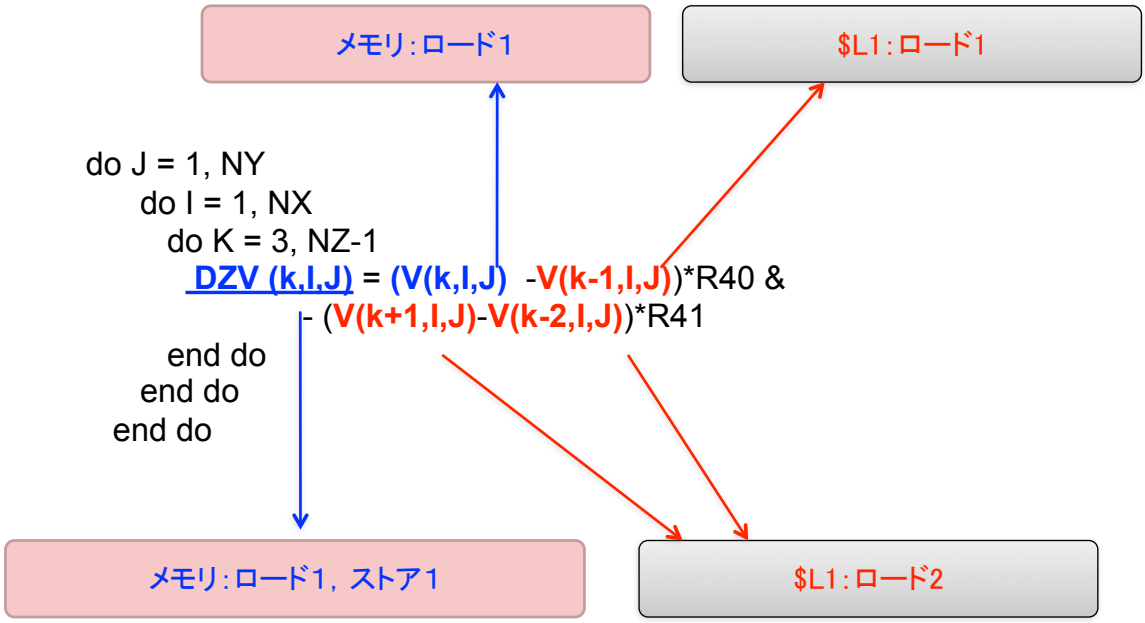


ベースとなる性能値



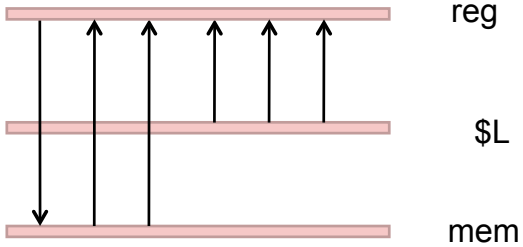


メモリとキャッシュアクセス (1)



メモリとキャッシュアクセス (2)

```
do J = 1, NY
  do I = 1, NX
    do K = 3, NZ-1
      DZV (k,I,J) = (V(k,I,J)  -V(k-1,I,J))*R40 &
                  - (V(k+1,I,J)-V(k-2,I,J))*R41
    end do
  end do
end do
```



	Store	Load	バンド幅比($L1)	データ移動時間の比(L1)	バンド幅比($L2)	データ移動時間の比(L2)
$L	1	5	11.1 (8*64G/s)	0.5= 6/11.1	5.6 (256G/s)	1.1=6/5.6
M	1	2	1(46G/s)	3=3/1	1 (46G/s)	3=3/1

データ移動時間の比を見るとメモリで律速される
→ メモリアクセス変数のみで考慮すれば良い。

性能見積り

```
do J = 1, NY
    do I = 1, NX
      do K = 3, NZ-1
        DZV (k,I,J) = (V(k,I,J)  -V(k-1,I,J))*R40 &
                    - (V(k+1,I,J)-V(k-2,I,J))*R41
      end do
    end do
  end do
```

- 最内軸(K軸)が差分
- 1ストリームでその他の3配列は$L1に載っており再利用できる。

要求B/F	12/5 = 2.4
性能予測	0.36/2.4 = 0.15
実測値	0.153

要求Byteの算出:

1store,2loadと考える

4x3 = 12byte

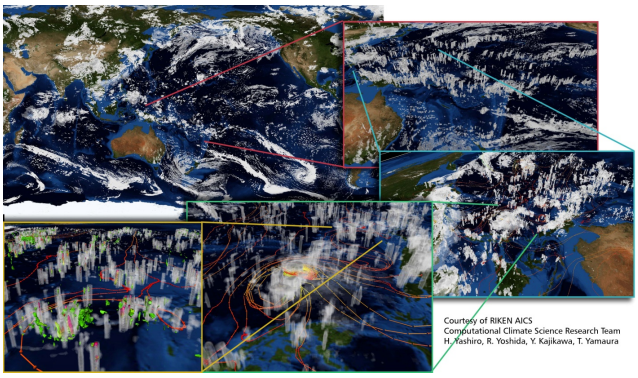
要求flop:

add：3 mult：2 = 5

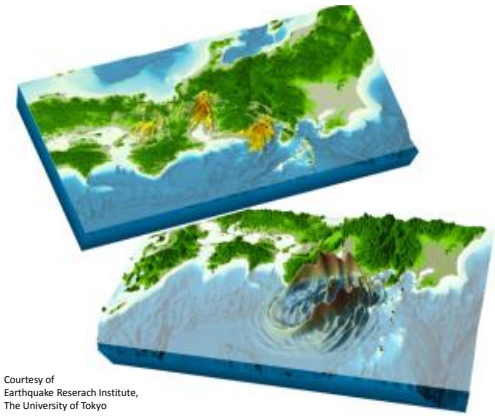
京の開発時に理研が性能最適化対象としたアプリケーション

プログラム名	分野	アプリケーション概要	期待される成果	手法
NICAM	地球科学	全球雲解像大気大循環モデル	大気大循環のエンジンとなる熱帯積雲対流活動を精緻に表現することでシミュレーションを飛躍的に進化させ，現時点では再現が難しい大気現象の解明が可能となる.	FDM（大気）
Seism3D	地球科学	地震波伝播・強震動シミュレーション	既存の計算機では不可能な短い周期の地震波動の解析・予測が可能となり，木造建築およびコンクリート構造物の耐震評価などに応用できる.	FDM（波動）
PHASE	ナノ	平面波展開第一原理電子状態解析	第一原理計算により，ポスト35nm世代ナノデバイス，非シリコン系デバイスの探索を行う.	平面波DFT
FrontFlow/Blue	工学	Large Eddy Simulation（LES）に基づく非定常流体解析	LES解析により，エンジニアリング上重要な乱流境界層の挙動予測を含めた高精度な流れの予測が実現できる.	FEM（流体）
RSDFT	ナノ	実空間第一原理電子状態解析	大規模第一原理計算により，10nm以下の基本ナノ素子(量子細線，分子，電極，ゲート，基盤など)の特性解析およびデバイス開発を行う.	実空間DFT
LatticeQCD	物理	格子QCDシミュレーションによる素粒子・原子核研究	モンテカルロ法およびCG法により，物質と宇宙の起源を解明する.	QCD

Earth Science

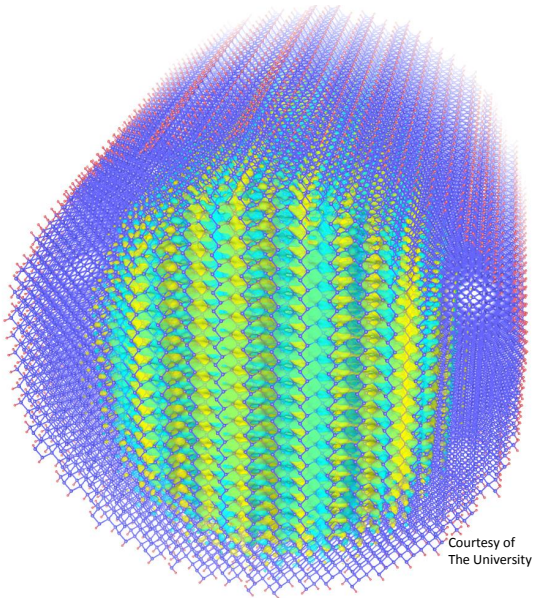


NICAM
全球雲解像大気大循環モデル

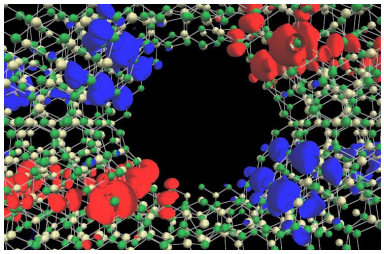


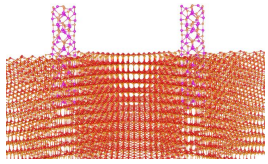
Seism3D
地震波伝播・強震動
シミュレーション

Nano Science



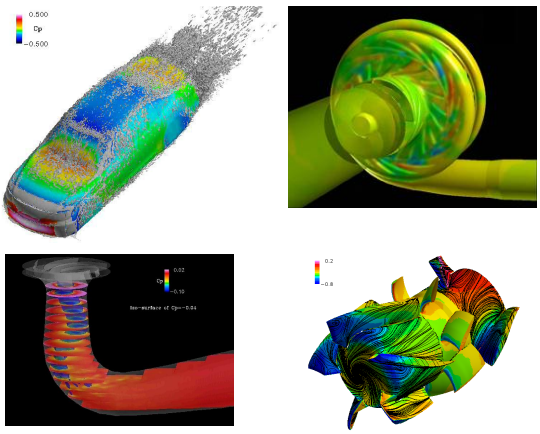
RSDFT
実空間第一原理電子状態解析





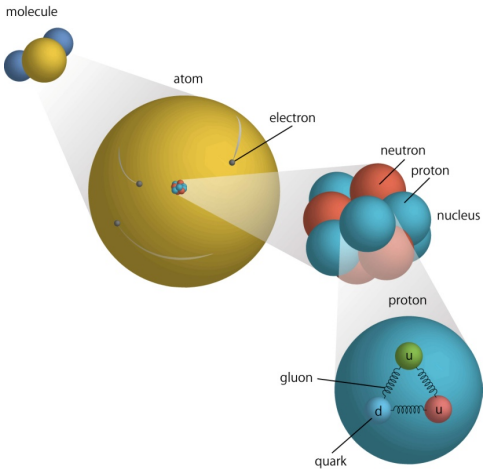
PHASE
平面波展開第一原理
電子状態解析

Engineering/Physics



FrontFlow/Blue

Large Eddy Simulation (LES)
に基づく非定常流体解析



LatticeQCD

格子QCDシミュレーションによる
素粒子・原子核研究

PHASEの問題点

1	アプリケーションとハードウェアの並列度のミスマッチ (アプリケーションの並列度不足)
2	非並列部の残存
3	大域通信における大きな通信サイズ、通信回数の発生
4	フルノードにおける大域通信の発生
5	隣接通信における大きな通信サイズ、通信回数の発生
6	ロードインバランスの発生

PHASEの高並列化・高性能化

固有値方程式

$$H\varphi_{ik}(G) = \varepsilon_i \varphi_{ik}(G)$$

φ_{ik}：電子軌道（=波動関数）
i：電子準位（=エネルギーバンド量子数）
G：波数格子
k：k点

- オリジナルはエネルギーバンドで並列化されていた
  →並列度の不足
- エネルギーバンド(B)に加え波数(G)の並列を実装
  →２軸並列による並列度の拡大
- 行列ベクトル積の行列積化を実装
  →単体性能の大幅な向上

PHASEの高並列化・高性能化

- 青字部が並列化されていない→非並列部の存在
- ２軸並列化の実装→非並列部の並列化

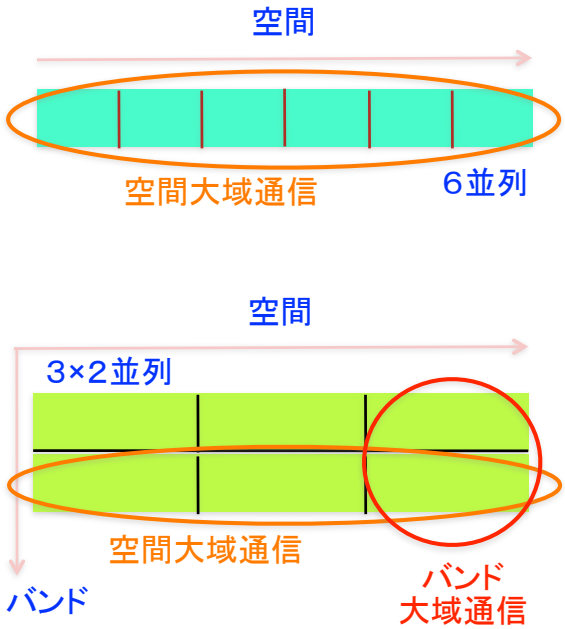
```
subroutine m_es_vnonlocal_w(ik,iksnl,ispin,switch_of_eko_part)
  +-call tstatc0_begin
  | loop_ntyp: do it = 1, ntyp
  |   loop_natm : do ia = 1, natm ----------原子数のループ
  |   +-call calc_phase
  |   | T-do lmtt2 = 1, ilmt(it)
  |   | +-call vnonlocal_w_part_sum_over_lmt1
  |   |   +-call add_vnlph_1_without_eko_part
  |   |     subroutine add_vnlph_1_without_eko_part()
  |   |     | T-if(kimg == 1) then
  |   |     | | T-do ib = 1, np_e  ----------エネルギーバンド並列部
  |   |     | |   T-do i = 1, iba(ik)
  |   |     | |   V-end do
  |   |     | | V-end do
  |   |     | +-else
  |   |     | | T-do ib = 1, np_e  ----------エネルギーバンド並列部
  |   |     | |   T-do i = 1, iba(ik)
  |   |     | |   V-end do
  |   |     | | V-enddo
  |   |     | V-end if
  |   |     end subroutine add_vnlph_1_without_eko_part
  |   | V-end do
  |   V-end do loop_natm
  | V-end do loop_ntyp
end subroutine m_es_vnonlocal_w
```

非並列部が波数で並列化できる

RSDFTの高並列化

並列軸拡張の効果

- 並列軸を増やす事で空間の分割粒度を増やすことが出来る
- 10万並列レベルに対応可能
- 空間並列のみの場合は全プロセッサ間の大域通信が必要
- 通信時間の増大を招く
- 2軸並列への書換で空間に対する大域通信が一部のプロセッサ間での通信とできる
- バンドに対する大域通信も同様
- 大域通信の効率化が実現可



高並列化・高性能化の成果

Program Name	Field	Achieved number of parallel nodes (core)	Performance (ratio to peak performance)
NICAM	earth science	81920 (655360)	0.8 Pflops(8%)
Seism3D	earth science	82944 (663552)	1.9 Pflops (18%)
FrontFlow/ Blue	engineering	80000 (640000)	0.3 Pflops(3.2%)
PHASE	material science	82944 (663552)	2.1 Pflops (20%)
RSDFT	material science	82944 (663552)	5.5Pflops (52%)
LatticeQCD	physics	82944 (663552)	1.6Pflops (16%)