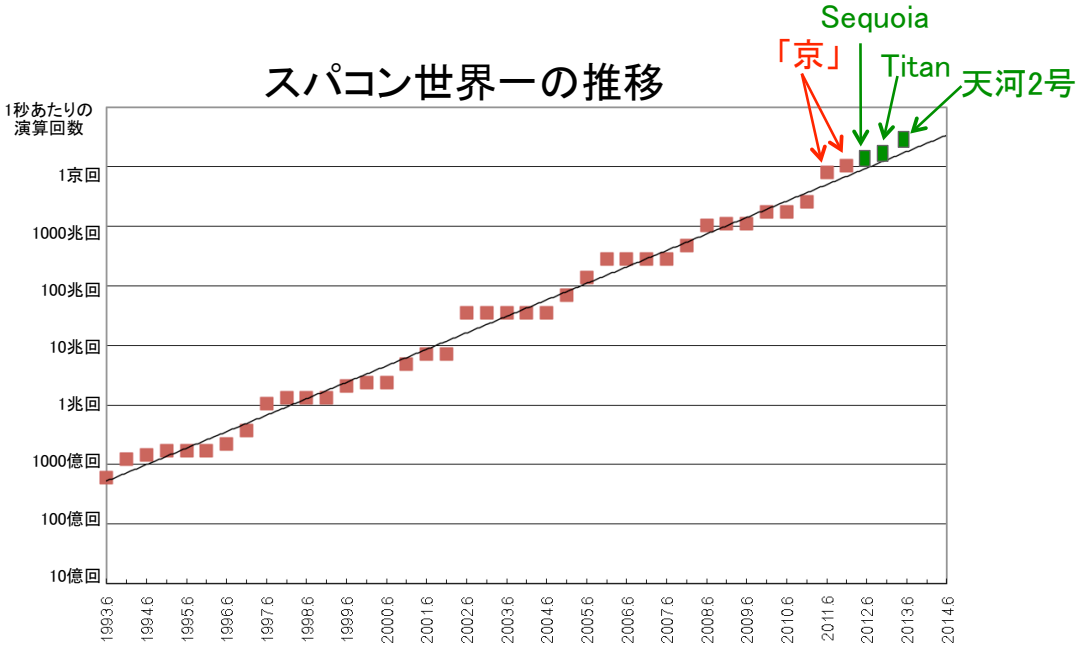
AICSサマースクール
−最適化II−

2013年8月

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

スパコン世界一をめぐる競争



平均すると1年に約1.9倍の性能向上！

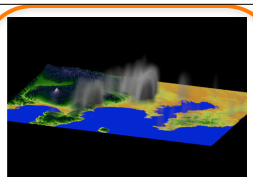
現在世界最速のスパコンは、世界初のスパコンCRAY-1（1976年）の
約2億1000万倍！

ちなみに

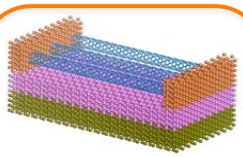
世界初のスパコンCRAY-1の演算性能は、約160メガフロップス
一方、iPhone4Sの演算性能は、約140メガフロップス

1976年　　2011年

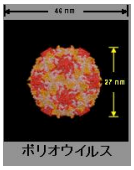
「京」の応用分野



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



ポスト35nm世代のデバイス全体のシミュレーションによ



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

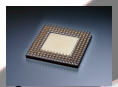
10^{21}m

10^{7}m

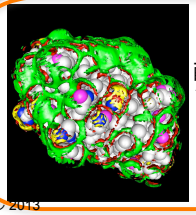
10^{2}m

10^{0}m

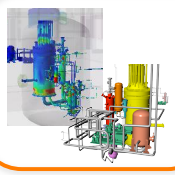
10^{-8}m



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

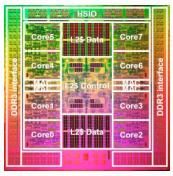


短周期地震波動の地震波シミュレーション、構造物の耐震シミュレーションを組み合わせた防災への貢献

プロセッサ構成

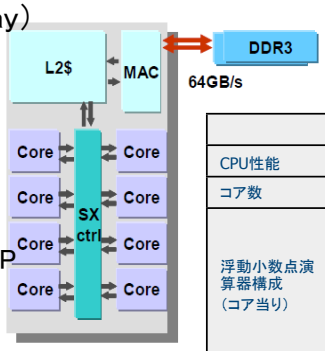
- ✓ 8コア構成，各コア256本の浮動小数点レジスタを備えたスーパースカラ方式
  - ✓ SIMD拡張（積和演算器2個 x 2セット）
  - ✓ コア当り16GFLOPS，CPU当り128GFLOPS
- ✓ コア共有の2次キャッシュ（6MB，12way）
  - ✓ ハードウェアバリア機構
  - ✓ プリフェッチ機構
  - ✓ セクタキャッシュ機能
- ✓ データ供給能力
  - ✓ レジスタ-L1キャッシュ間：4B/FLOP
  - ✓ L1キャッシュ-L2キャッシュ間：2B/FLOP
  - ✓ L2キャッシュ-主記憶間：0.5B/FLOP

SPARC64™ VIIIfx



提供：富士通（株）

動作周波数：2GHz
チップサイズ：22.7mm x 22.6mm
トランジスタ数：760 M トランジスタ
消費電力：58W（水冷，30℃時）

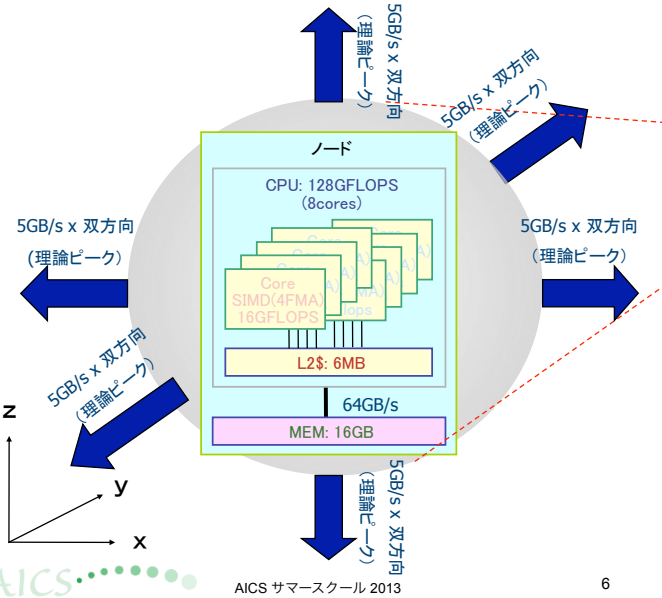


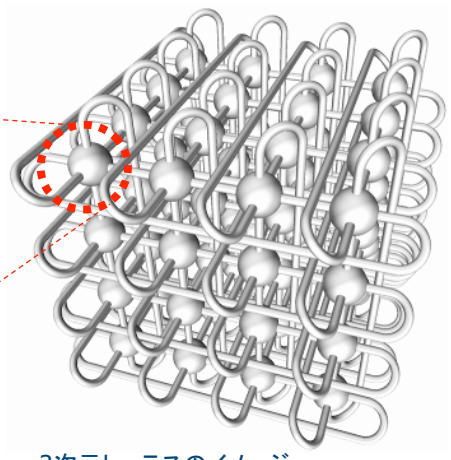
	仕　様
CPU性能	128GFLOPS（16GFLOPSx8コア）
コア数	8個
浮動小数点演算器構成（コア当り）	積和演算器：2×2個（SIMD） （逆数近似命令：SIMD動作） 除算器：2個 比較器：2個
	浮動小数点レジスタ(64ビット)：256本 グローバルレジスタ(64ビット)：188本
キャッシュ構成	1次命令キャッシュ：32KB(2way) 1次データキャッシュ：32KB(2way) 2次キャッシュ：6MB(12way)コア間共有
メモリバンド幅	64GB/s(0.5B/F)

より詳細な情報は，「SPARC64™ VIIIfx Extensions」を参照のこと
http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64viiifx-extensions.pdf

計算ノードとインターコネクトの構成

- 計算ノードの構成
  - CPU（8コア）　：1個
  - ICC（インターコネクト用LSI）　：1個
  - メモリ　：16GB
- インターコネクトの構成
  - ユーザービューは3次元トーラス
  - 帯域：3次元の正負各方向にそれぞれ5GB/s x 2（双方向）【理論ピーク】
  - ケーブル：約200,000本，約1000km





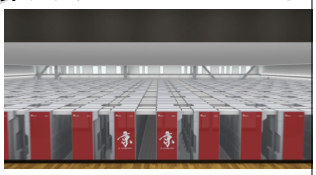
3次元トーラスのイメージ

提供：富士通（株）

「京」のシステム構成

システム全体

計算ラック×864=82944ノード



1京回/秒≒10.6ペタフロップス

1PB以上

計算ラック群

計算ラック×8

ディスクラック×2

98.4兆回/秒

12TB

計算ラック

システムボード×24

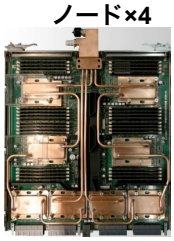
IOシステムボード×6

12.3兆回/秒

1.5TB

システムボード

ノード×4



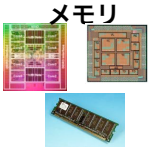
5120億回/秒

ノード

CPU×1

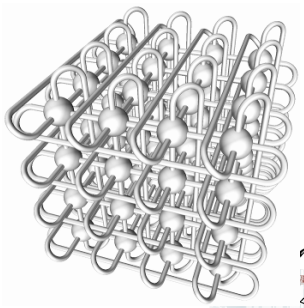
ICC×1

メモリ



演算性能：1280億回/秒　64GB

メモリ容量：16GB

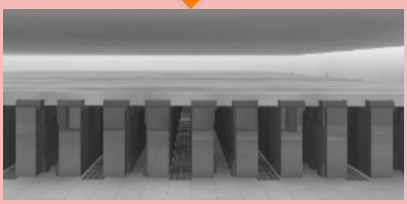


現代のスパコン利用の難しさ

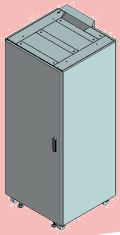
アプリケーションの
超並列性を引き出す

プロセッサの単体性能
を引き出す

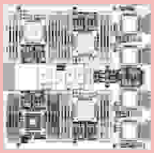
System

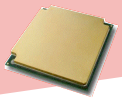


Rack



System Board

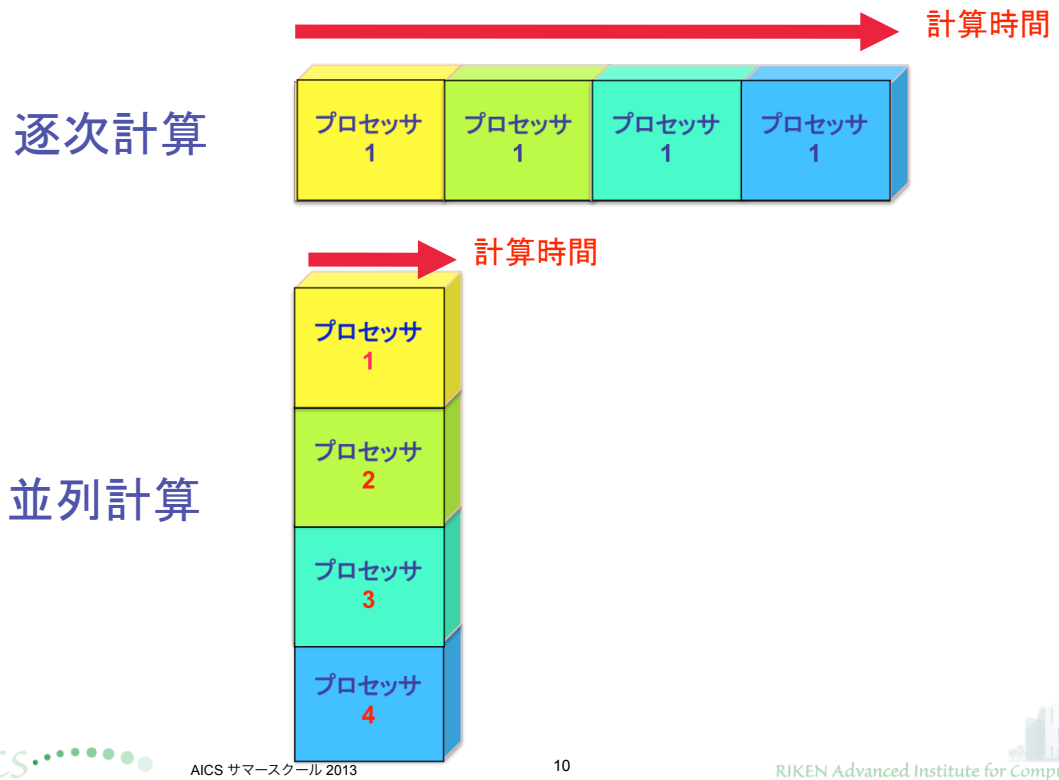




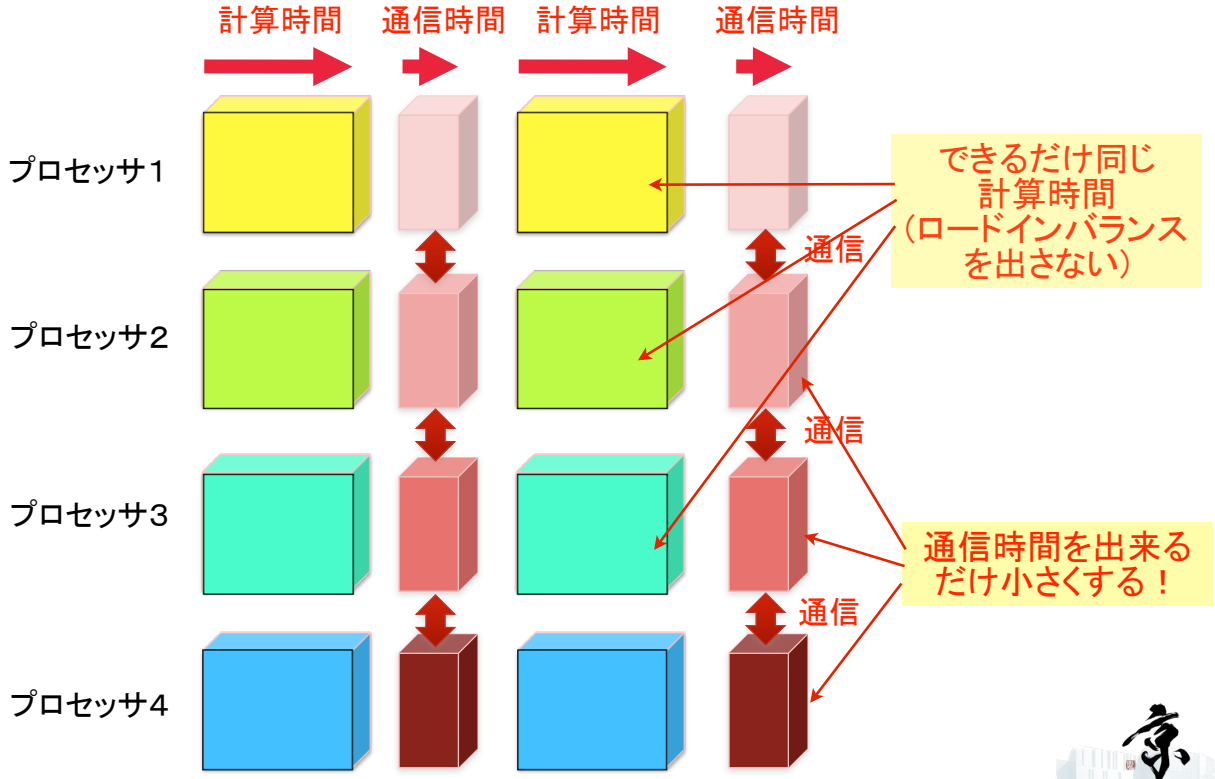
画像提供：富士通株式会社

高並列化のための重要点

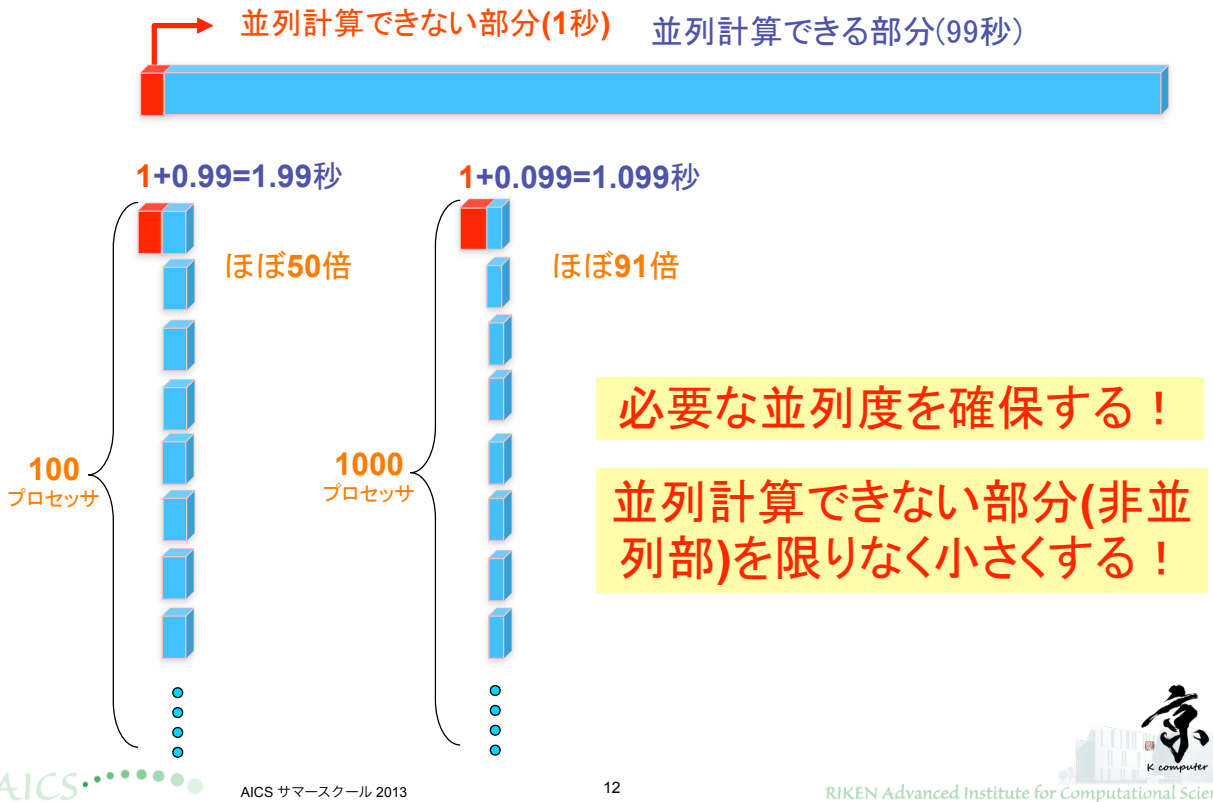
そもそも並列化とは？(1)



そもそも並列化とは？(2)



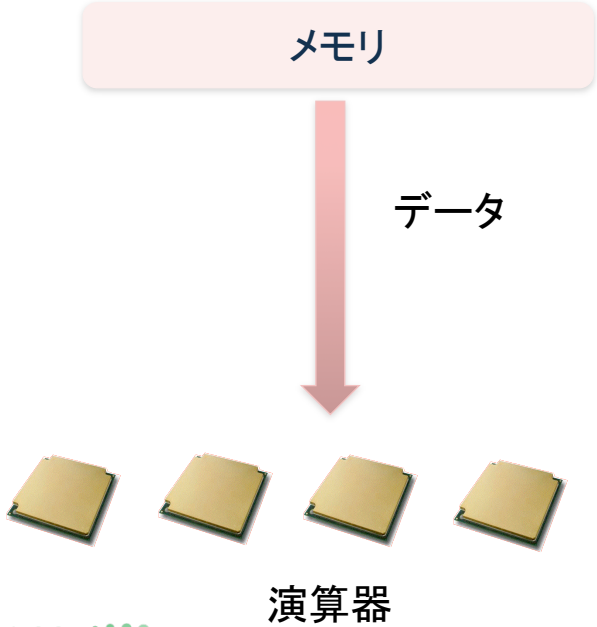
そもそも並列化とは？(3)



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



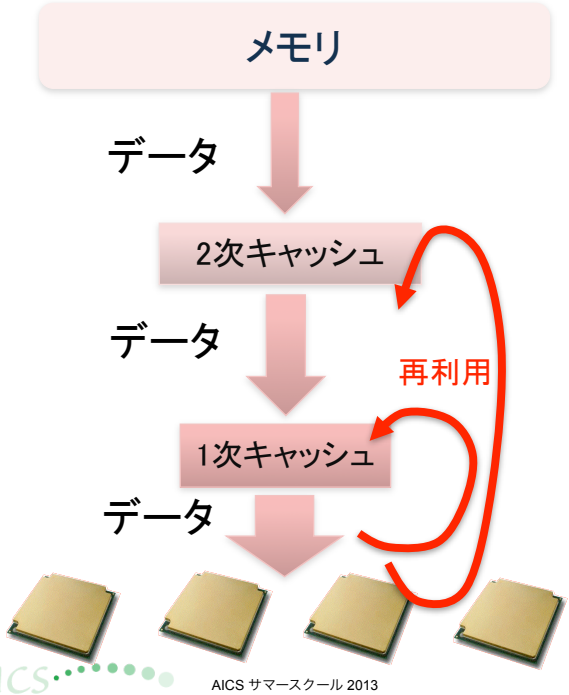
メモリウォール問題

✓ 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)

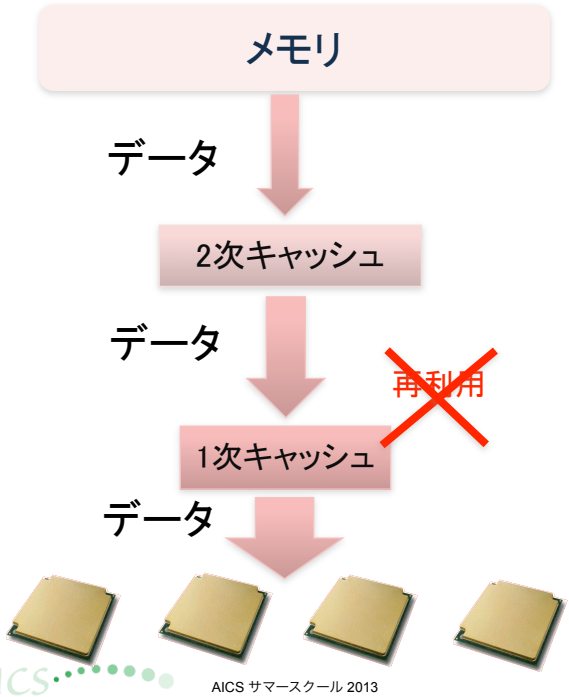
例えば行列行列積
($2N^2$個のデータで$2N^3$個の演算可)

(*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

プロセッサの単体性能を引き出す(3)

とは言っても・・・・



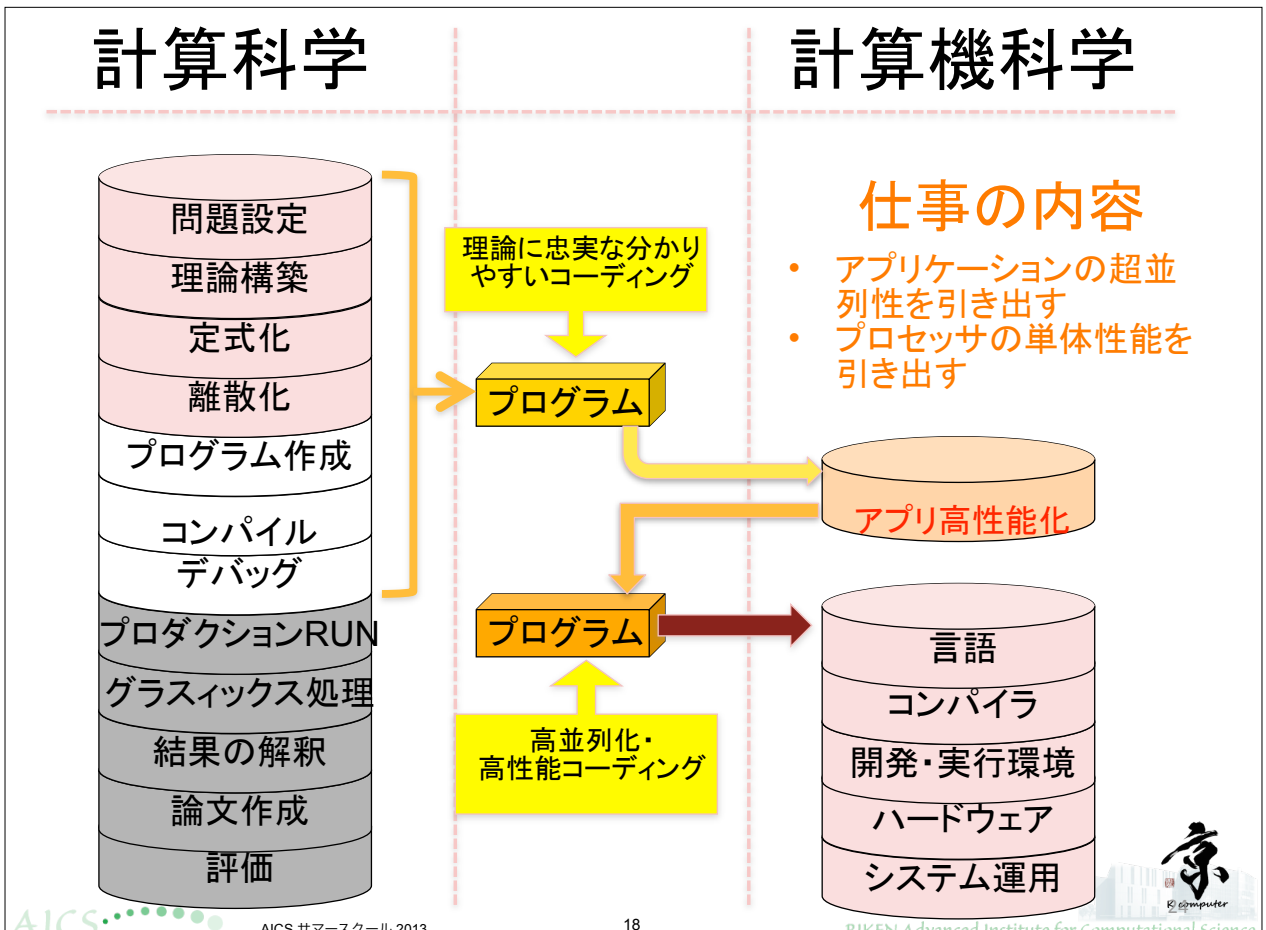
・再利用出来ない問題もある
・こうなる演算器の能力を十分使い切る事が出来ない

「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

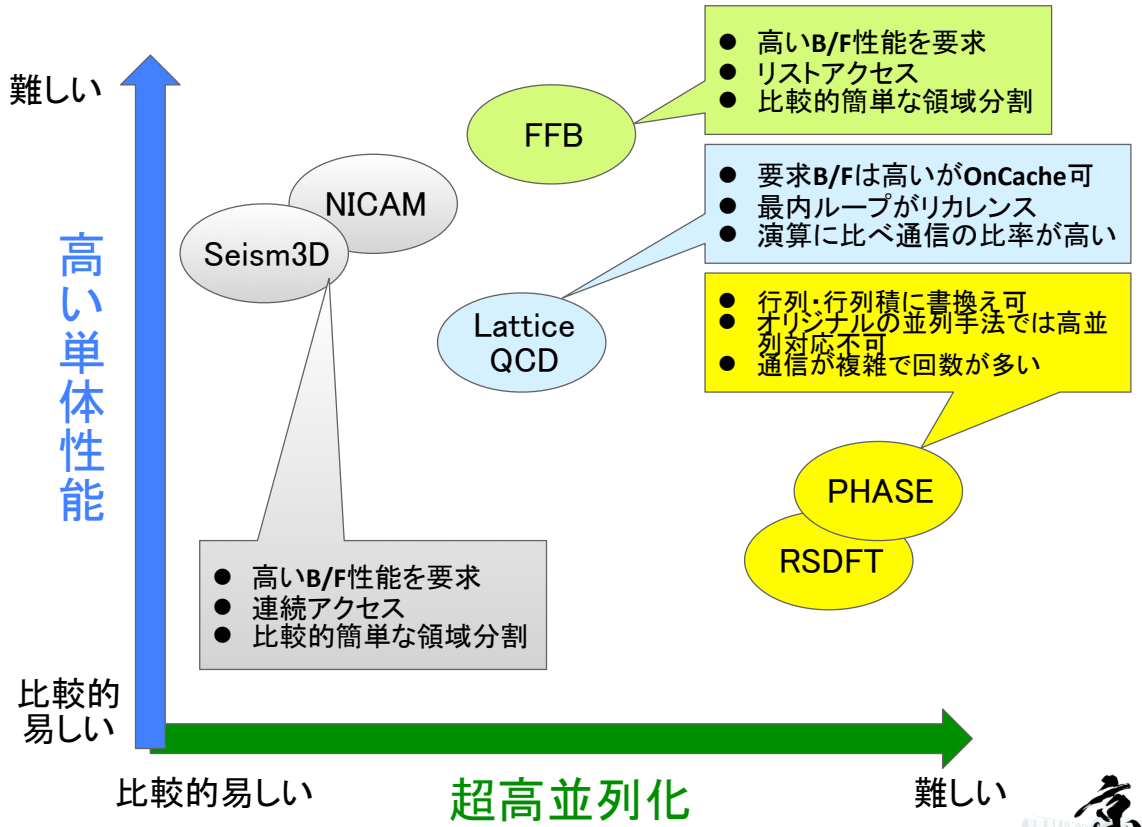
キャッシュの有効利用が難しい

理研で進めたアプリケーション の高性能化



対象アプリケーション

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



ここから

- 理研で進めたアプリケーションを例題に
  - 高並列化手法について説明
  - 単体性能向上手法について説明

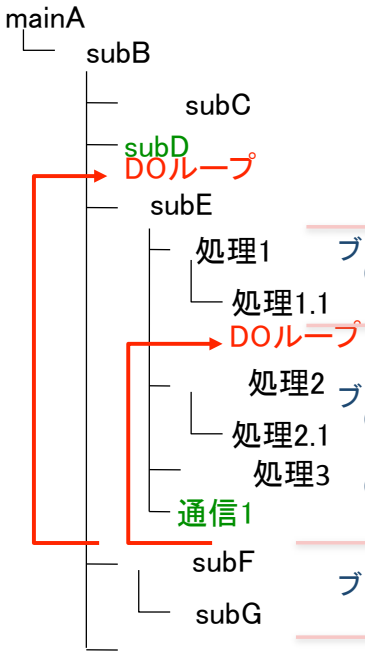
アプリケーションを高並列にするためには どうしたら良いか

- ✓ 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

最初に

アプリケーションの高並列阻害する要因を洗い出す事(並列特性分析)が重要

並列特性分析

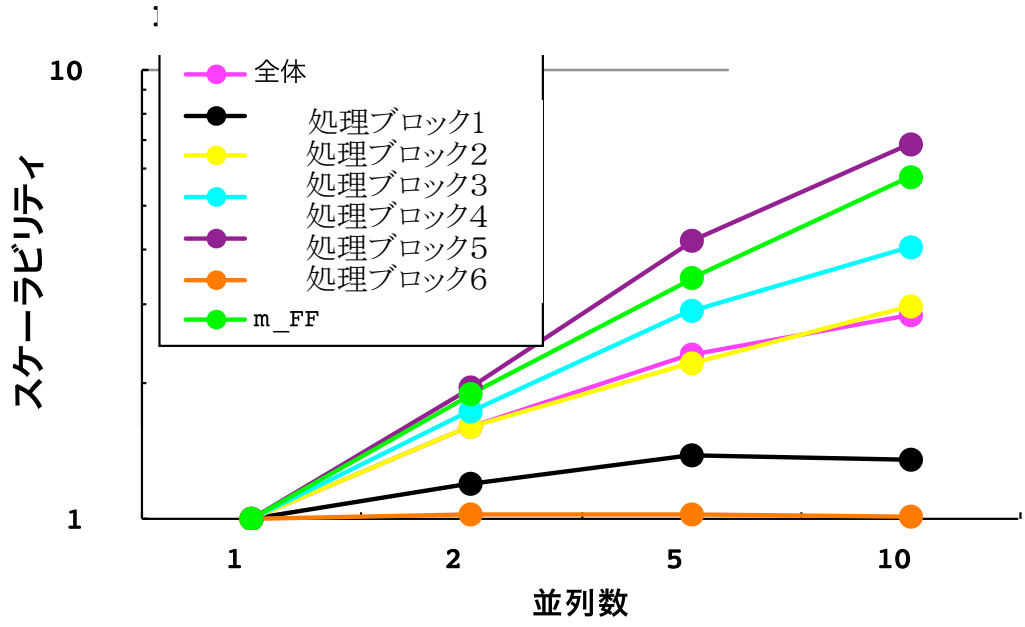


	実行時間・スケーラビリティ	物理的処理内容	演算・通信特性	演算・通信見積り	カーネル
ブロック1（計算）			部分並列	Nに比例	
ブロック2（計算）			完全並列	N**3に比例	○
（通信）			隣接通信	隣接面に比例	○
ブロック3					

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



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？

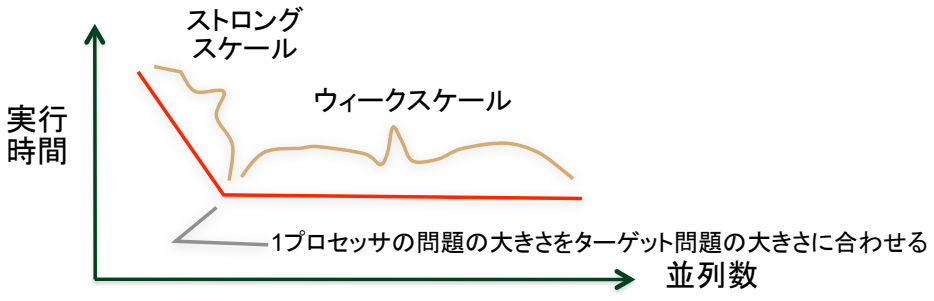
→ これらの評価が重要

ストロングスケーリング：
　　全体の問題規模を一定にして並列数を増やし測定する方法

ウィークスケーリング　：
　　1プロセッサで実行する問題規模を一定にし並列数を増やし測定する方法

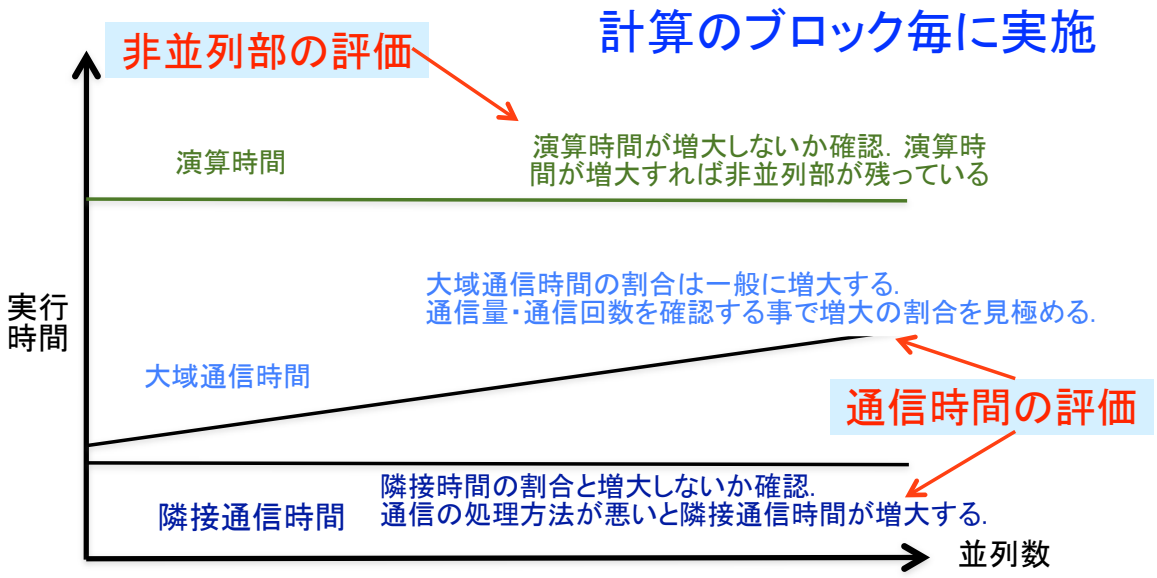
そのために

- 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

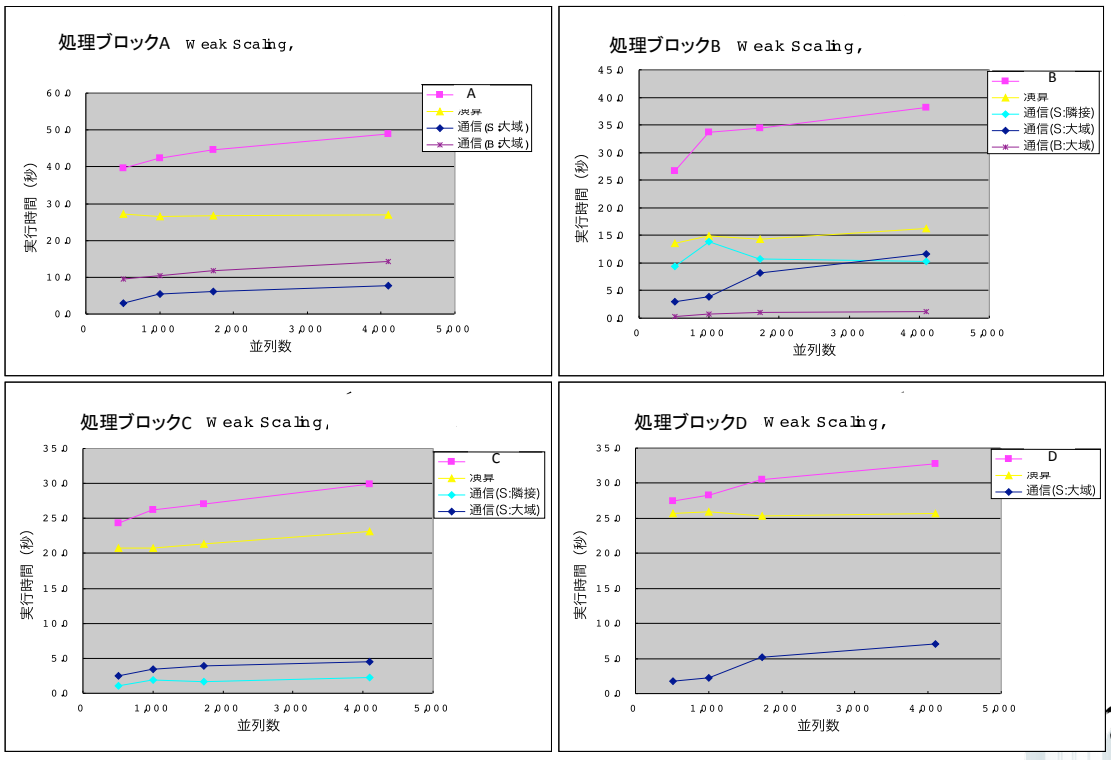


ウィークスケーリング評価

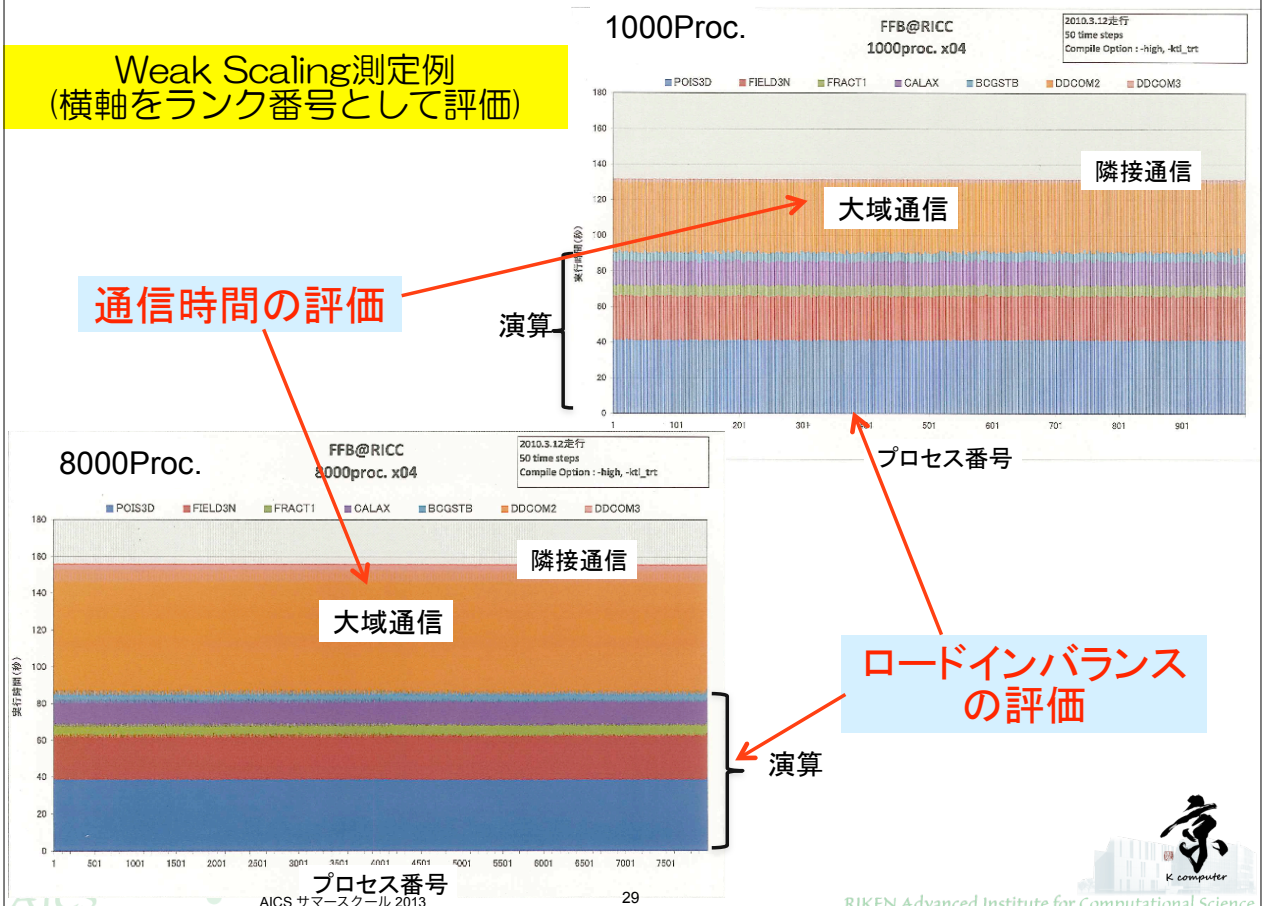
- 現状使用可能な実行環境を使用し100程度/1000程度/数千程度と段階を追ってできるだけ高い並列度で並列性能を確認する(ウィークスケーリング測定).
- ウィークスケーリングが難しいものもあるが出来るだけ測定したい. 難しい場合は, 演算時間をモデル化して実測とモデルとの一致を評価する.



Weak Scaling測定例



Weak Scaling測定例
(横軸をランク番号として評価)



十分な並列数の確保

■ RSDFT/PHASEの例

対象とする系(分子・固体)に含まれる電子の数だけの波動関数φを求める
電子状態(バンド構造，電荷密度，状態密度)がわかる
密度汎関数法(DFT:Density Functional Theory)に基づきKohn-Sham方程式を解く

Kohn-Sham方程式 (H:ハミルトニアン r:座標 n:エネルギーバンド k:波数)

$$[-\frac{1}{2}\nabla^2 + V_{LOC}[\rho(\phi)](r) + V_{non-LOC}]\phi_n(r) = \varepsilon_n\phi_n(r)$$

$$H\phi_n(r) = \varepsilon_n\phi_n(r) \quad :\text{RSDFT}$$

$$H\phi_n(k) = \varepsilon_n\phi_n(k) \quad :\text{PHASE}$$

オリジナルの並列軸
拡張した並列軸
オリジナルの並列軸
拡張した並列軸
並列軸の拡張

非並列部分最小化

■ PHASEの例

波数に関する並列軸の拡張で並列化

プログラム分析により非並列部分が見つかる

```
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
   |   +-call calc_phase
   |   |  T-do lmt2 = 1, ilmt(it)
   |   |  +-call vnonlocal_w_part_sum_over_lmt1
   |   |  |  +-call add_vnlph_l_without_eko_part
   |   |  |  |  subroutine add_vnlph_l_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_l_without_eko_part
   |   |  |  V-end do
   |   |  V-end do loop_natm
   |   V-end do loop_ntyp
end subroutine m_es_vnonlocal_w
```

オリジナルの並列部

通信時間最小化

■ RSDFTの例

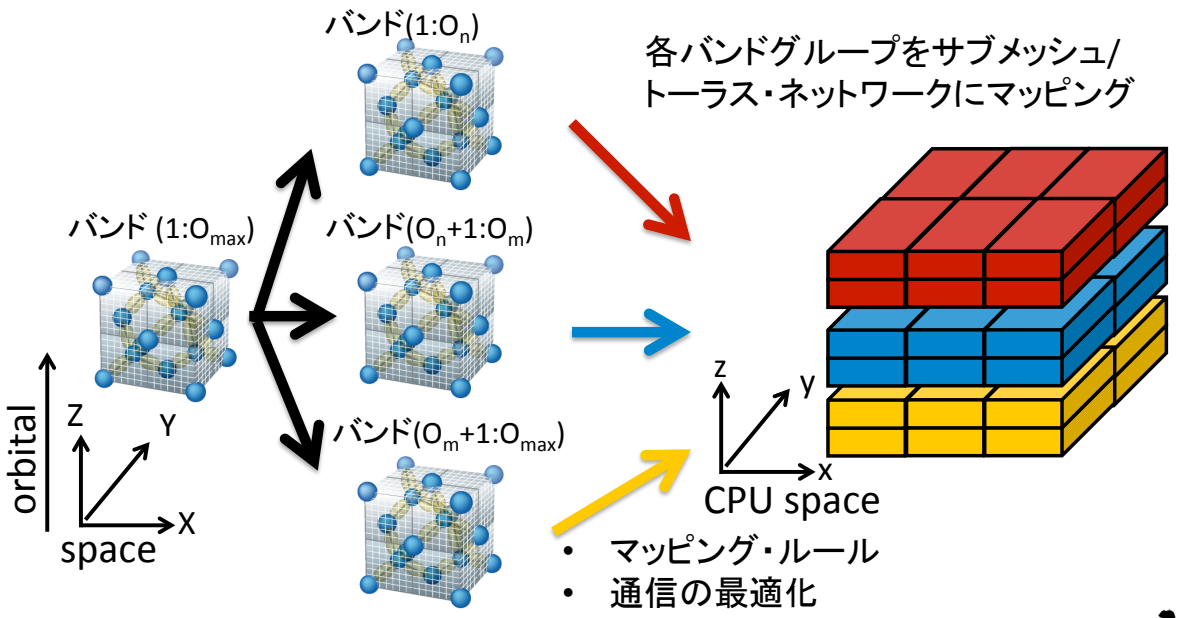
RSDFTの計算手順

1. 波動関数Φの初期値を与える
2. CG法により、波動関数φを更新する
3. 波動関数Φを規格直交化（グラムーシュミット）する
4. 局所ポテンシャルV_{LOC}を更新する。更新前後で変化が無ければ計算終了
5. ハミルトニアンを更新
6. 部分対角化（MB×MB空間で）を行い、1.に戻る

通信時間最小化

■ RSDFTの例

空間並列 | 空間並列＋バンド並列 | Tofuネットワークへのマッピング

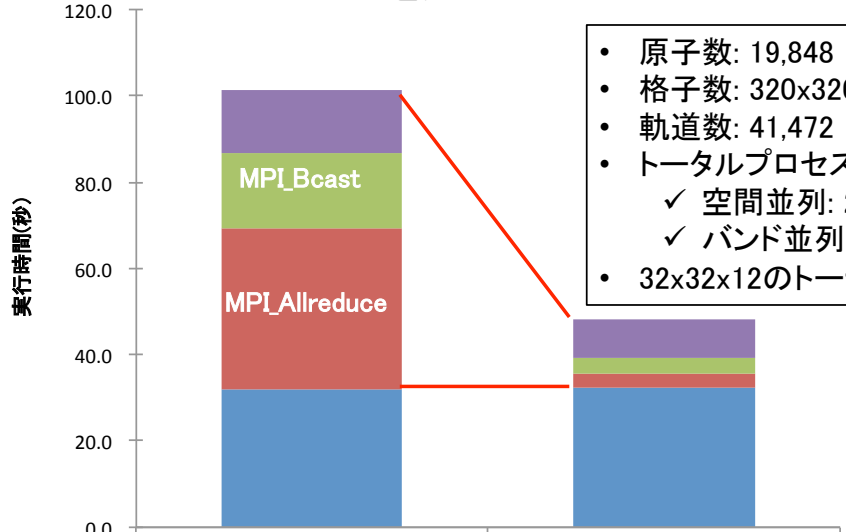


- マッピング・ルール
- 通信の最適化

サブメッシュ/トーラス内で通信が閉じられる

最適マッピングの効果- Gram-Schmidt -

最適マッピング → サブコミュニケータ間のコンフリクトが発生しない
MPI通信でTofu向けアルゴリズムが選択される



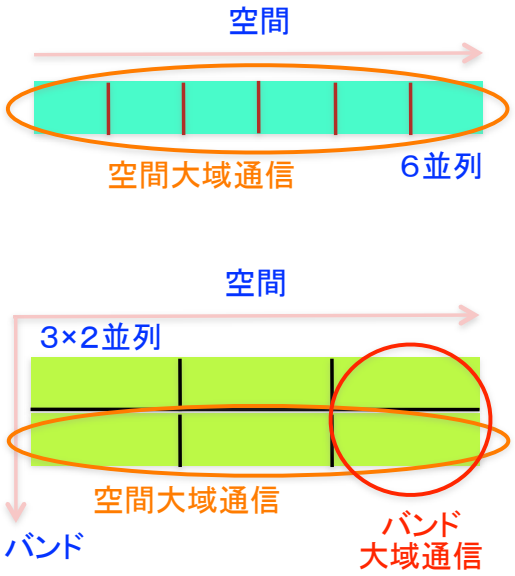
	マッピングなし	最適マッピング
通信 同期/軌道	14.473	8.658
通信 大域/軌道	17.364	3.827
通信 大域/空間	37.246	3.336
演算	32.095	32.191

- 原子数: 19,848
- 格子数: 320x320x120
- 軌道数: 41,472
- トータルプロセス数: 12,288
  - ✓ 空間並列: 2,048(32x32x2)
  - ✓ バンド並列: 6
- 32x32x12のトーラスにマッピング

HPCS2012

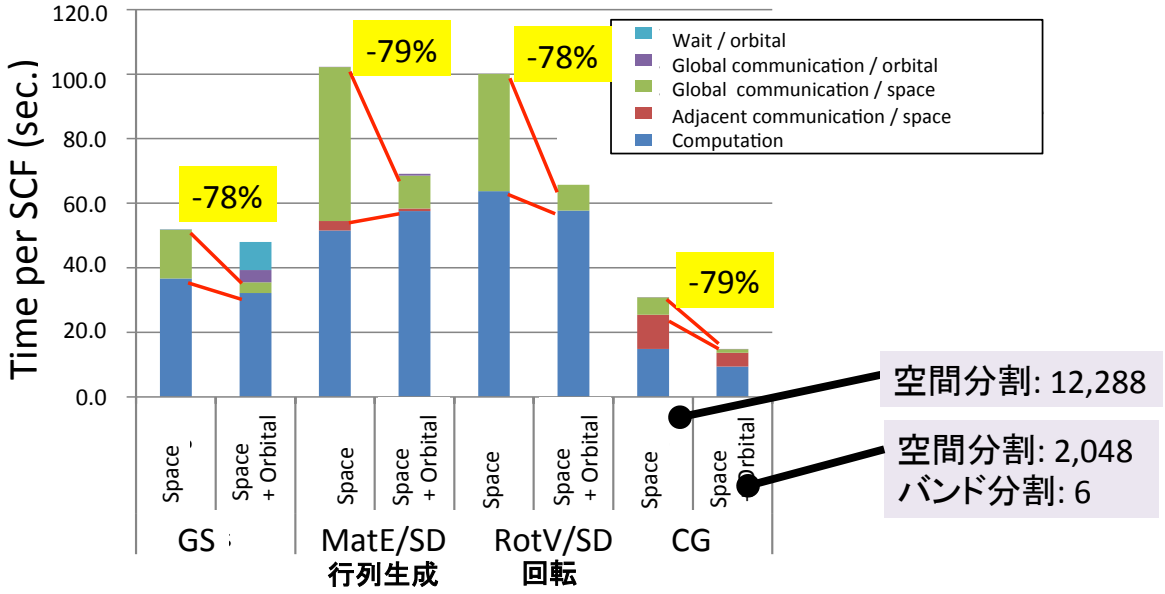
2軸並列の結果大域通信時間が減少

- ✓ 空間並列から空間+エネルギーバンド並列(2軸並列)へ書換
- ✓ 並列軸を増やす事で10万並列レベルに対応可能となった
- ✓ 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



その効果

SiNW, 19,848 原子, 格子数:320x320x120, バンド数:41,472
トータル並列プロセス数は12,288で固定



大域通信時間を大幅に削減

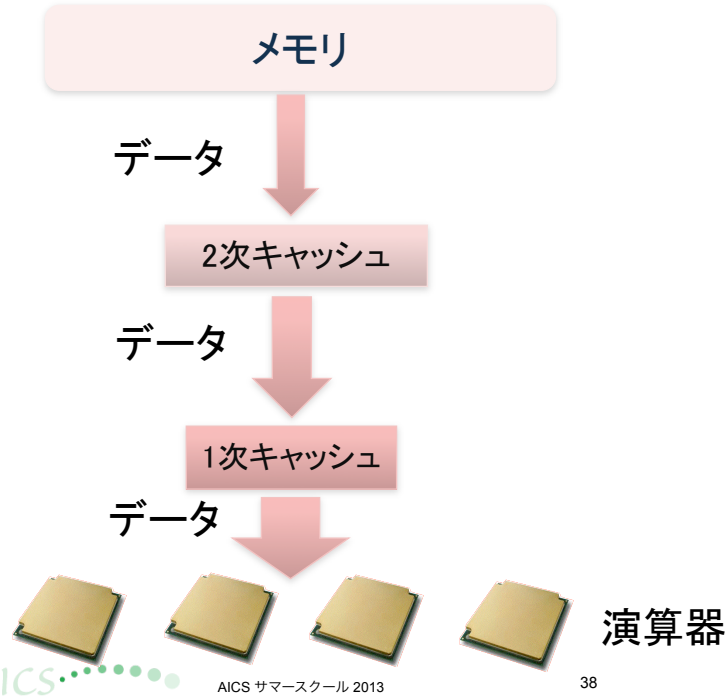
ここからCPU単体性能を上げる話

CPU単体性能をあげるためには？

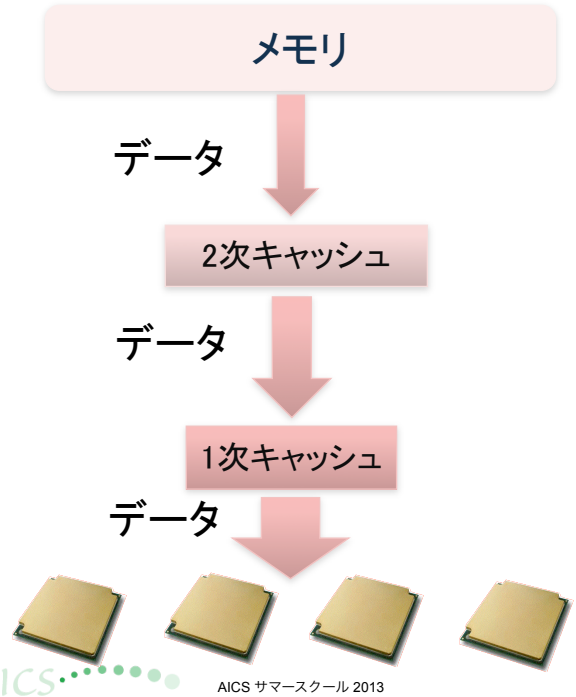
CPU内の複数コアでまずスレッド並列する事
は前提として

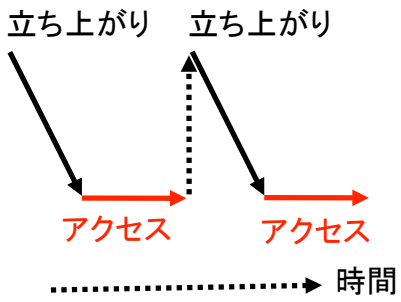
(1)プリフェッチの有効利用
(2)ラインアクセスの有効利用
(3)キャッシュの有効利用
(4)効率の良い命令スケジューリング
(5)演算器の有効利用

レイテンシ(アクセスの立ち上がり)



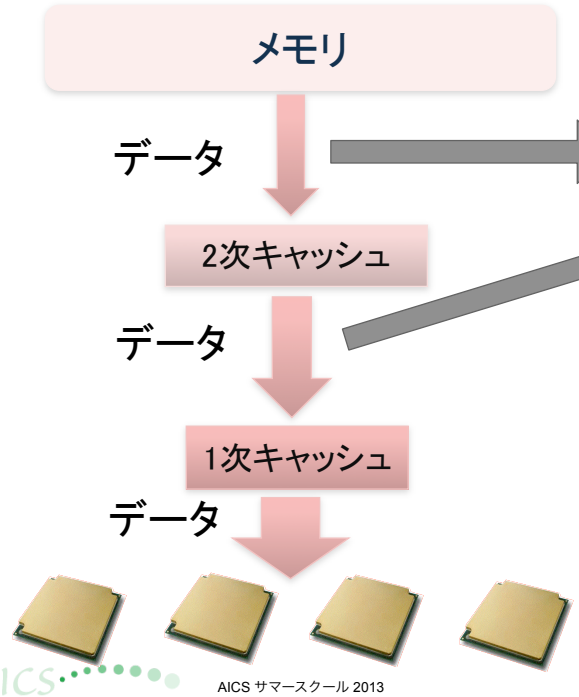
レイテンシ(アクセスの立ち上がり)

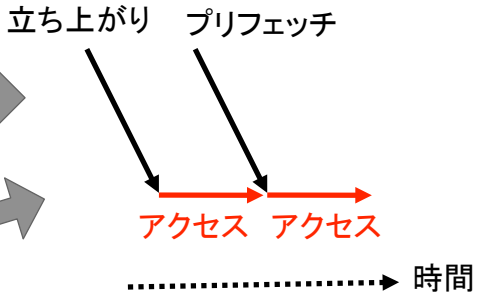




演算器

(1)プリフェッチの有効利用

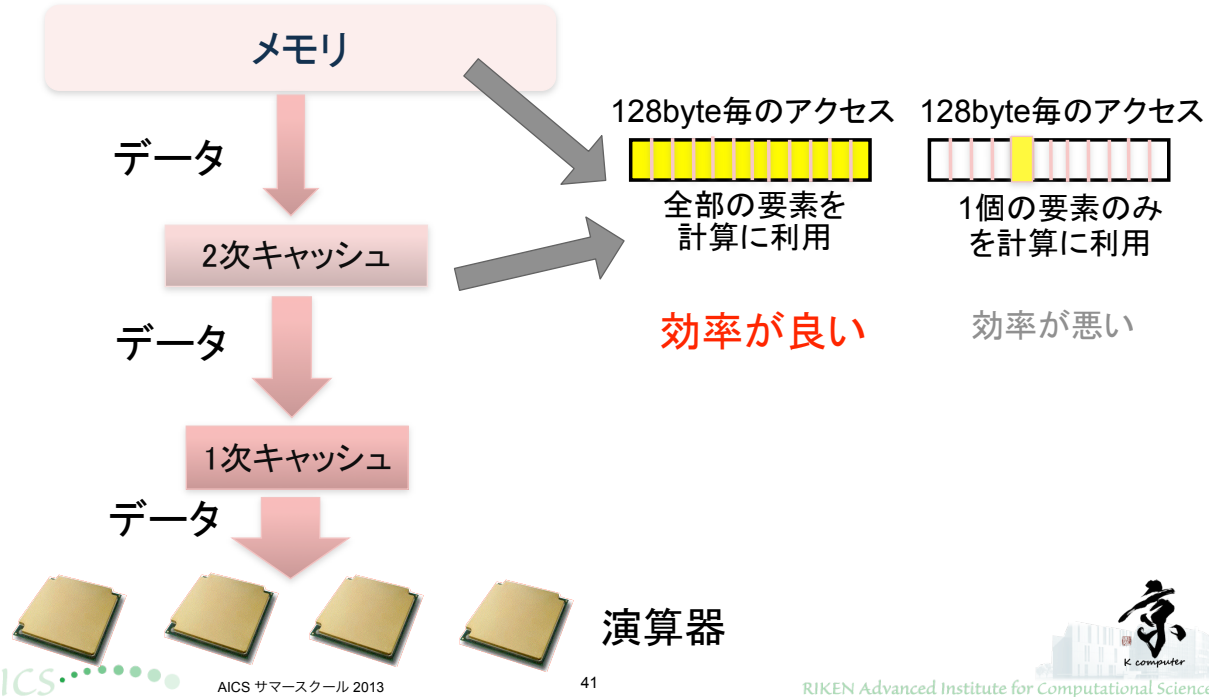




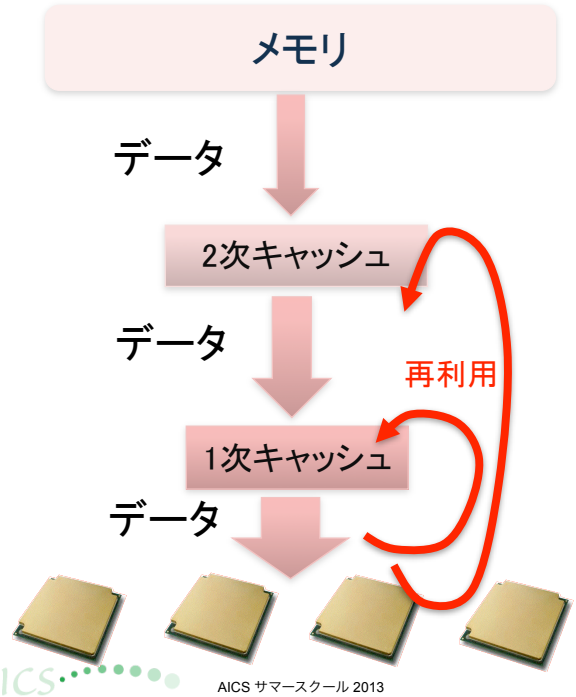
演算にアクセスそのものを
隠せる場合もある

演算器

(2)ラインアクセスの有効利用

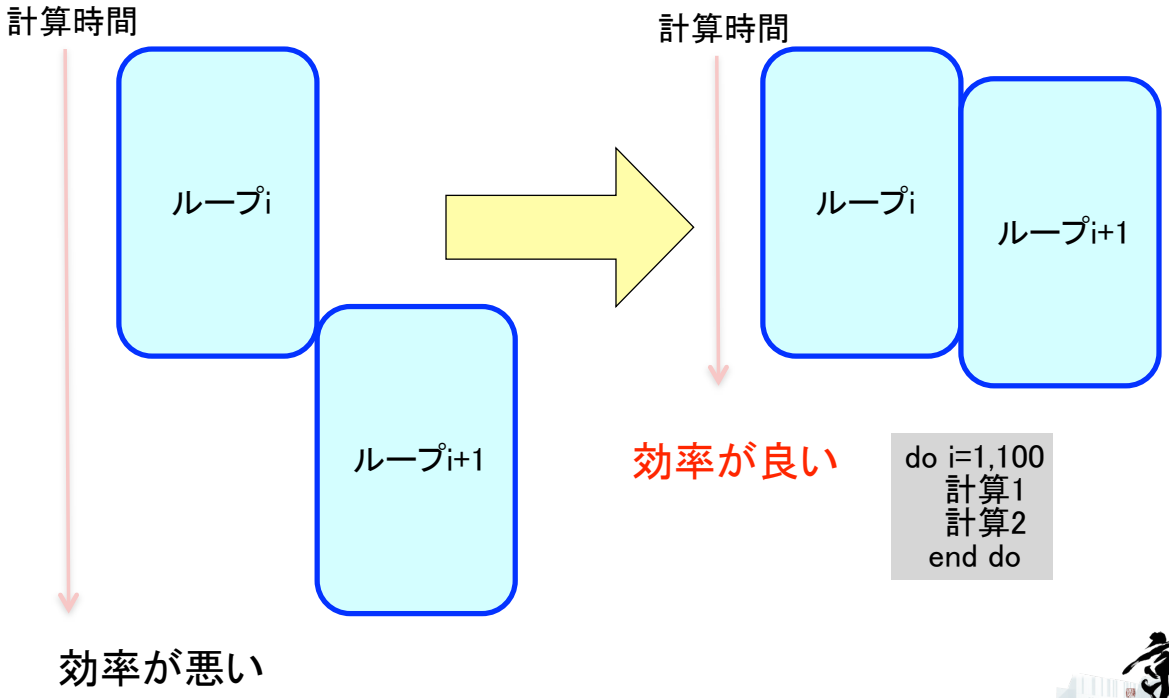


(3)キャッシュの有効利用



・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

(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

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値と性能の関係

高い性能を得るための要素と要求B/F値の関係

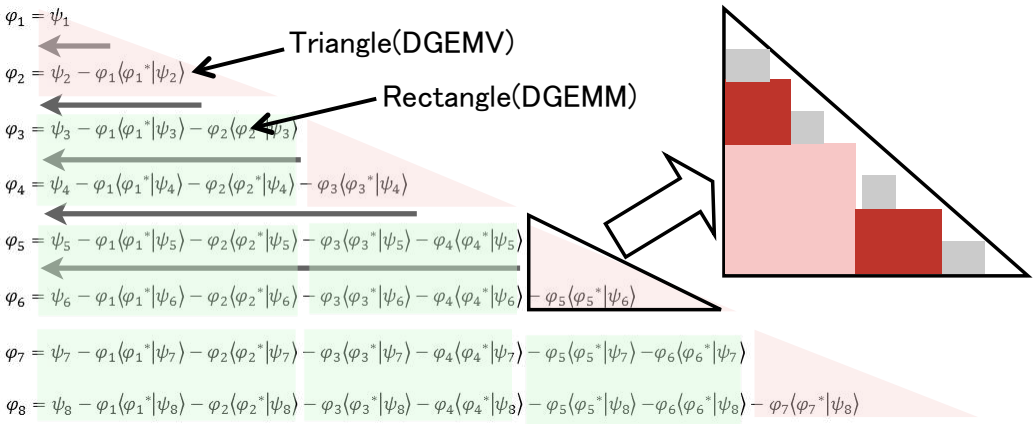
要求するB/Fが小さいアプリケーションについて
・原理的にキャッシュの有効利用が可能
・まずデータをオンキャッシュにするコーディング:(3)が重要

・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

キャッシュブロック:
n個のデータをキャッシュに置いてデータの使い回しによりnの2乗回の計算をする

計算コアの行列積化 – RSDFT Gram-Schmidt –

行列ベクトル積を行列・行列積に変換　　**再帰分割法**



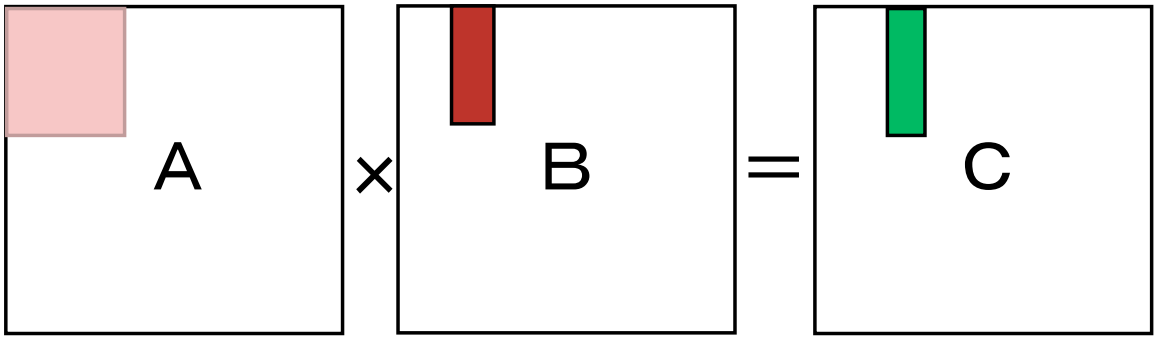
- 依存関係のある部分とない部分にブロック化して計算
- 再帰的にブロック化することで四角部を多く確保

DGEMMの性能

✓実行効率：96.6%

✓セクタキャッシュの効果：12%程度の性能向上

行列積(C=AB)　　行列をブロックに分け、L1D、L2キャッシュにのせる



L2にのせる

L1Dにのせる
セクタキャッシュ機能で追い出しを防止

高い性能を得るための要素と要求B/F値の関係

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
・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

Seism3D,FFB

◆ Seism3DとFFBは基本的に疎行列とベクトルの積であるので，その観点で議論する

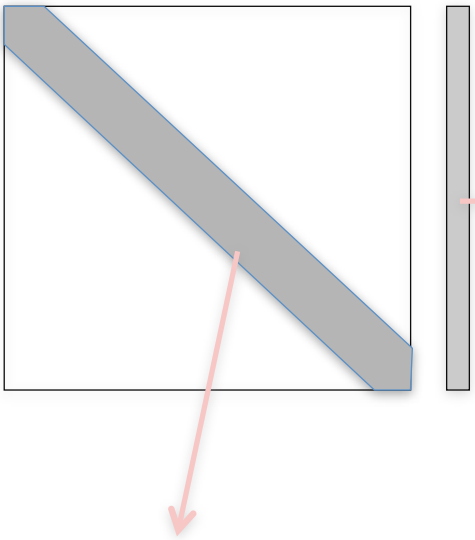
疎行列とベクトルの積の問題点

◆ CPU単体性能チューニングにおいては，対象とするコーディングの限界性能値が分からない
◆ 要求B/F値が高いコーディングに対するスカラチューニング

ここでの議論の狙い

◆どの段階までチューニング作業を進めるかの判断基準を設ける事を目的に疎行列とベクトルの積のコーディングから性能を予測する手法を提案
◆Seism3D及びFFBを対象に「京」において更なる性能向上のための具体的なチューニング手法を提案

疎行列とベクトルの積の特徴

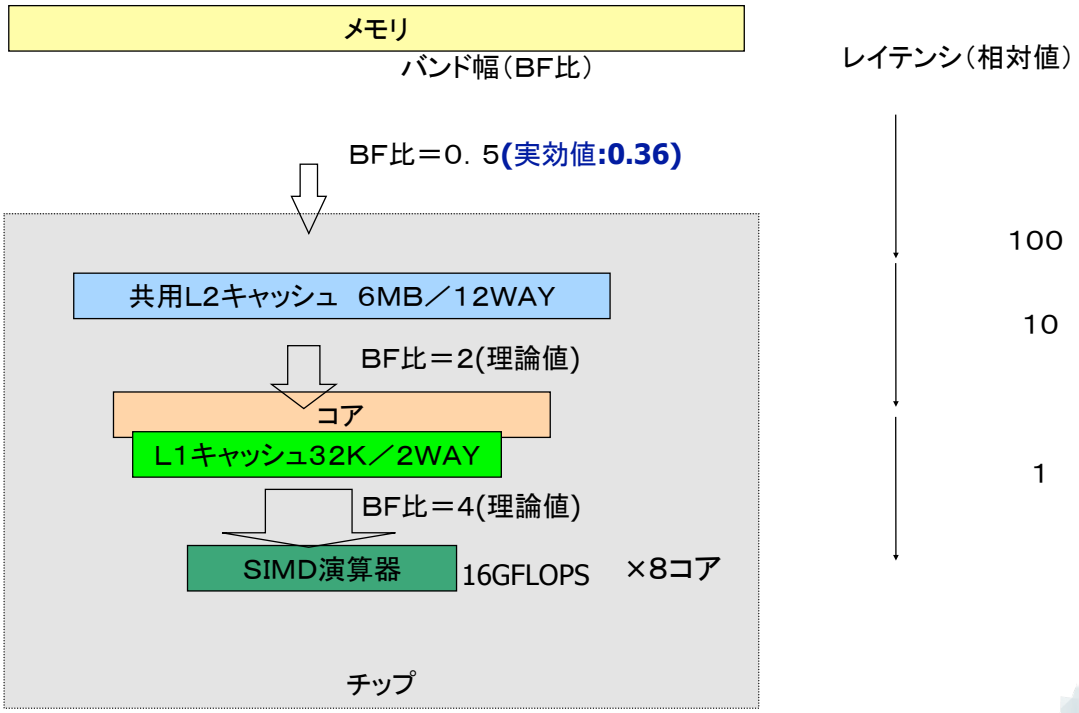


- ベクトルは一般的に3次元配列である
- しかしコーディング的には1次元や2次元配列で表現されている場合もある
- ベクトルは行列の列幅程度の再利用性がある
- したがって量的には行列の列幅分の1程度の大きさである
- メモリバンド幅を消費するが再利用性を生かせるかが重要
- 一次元の量としてリストアクセスする場合が有る
- その場合はリストは行列と同じ量が必要となりメモリバンド幅を消費する

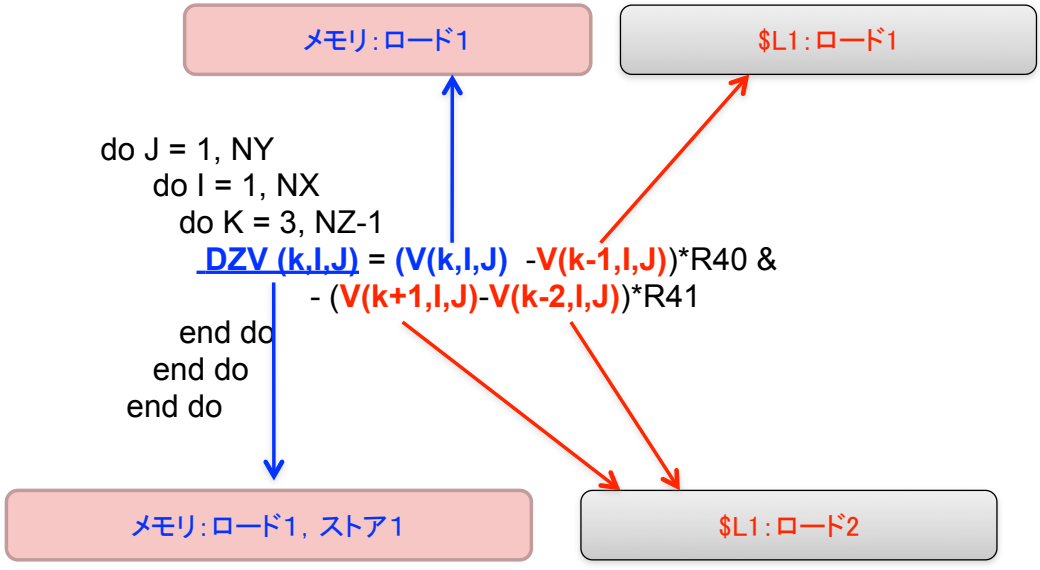
- 行列は一般的に3次元の配列である
- しかしコーディング的には1次元や2次元配列で表現されている場合もある
- ただ量的には大きいのでメモリバンド幅を消費する場合が一般的
- しかし物理的に2次元の量である場合やスカラー量である場合がある

性能予測手法

ベースとなる性能値

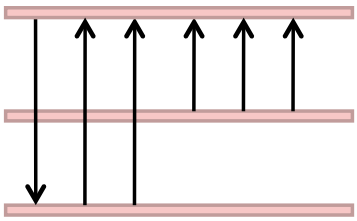


メモリとキャッシュアクセス(1)



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Byteの算出:

1store,2loadと考える

4x3 = 12byte

要求flop:

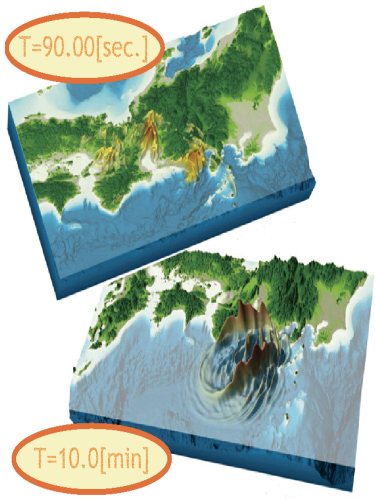
add：3 mult：2 = 5

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

Seism3D

- 有限差分法により数値的に粘弾性方程式を時間発展させる
- 地震伝播と津波を連動して解く
- 大規模な並列化に対応しているアプリケーション
- 以下の6つの計算部分より構成

a)応力空間微分計算
b)速度空間微分計算
c)応力時間積分計算
d)応力時間積分吸収計算
e)速度時間積分計算
f) 速度時間積分吸収計算



Seism3Dのチューニング

空間微分Z方向の計算a)b)(1次元目の差分)

```
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

空間微分X方向の計算a)b)(2次元目の差分)

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

- 第2軸(I軸)が差分
- 1ストリームでその他の3配列は$L1or$L2に載っており再利用できる
- 従って1次元目が差分のパターンと同じ性能になる

要求Byteの算出:

P12より、メモリコストだけを考慮する。

1sore,2loadと考える

4x3 = 12byte

要求flop:

add：3 mult：2 = 5

要求B/F	12/5 = 2.4
性能予測	0.36/2.4 = 0.15
実測値	0.135

- 実測値が13.5%と少し低い
- 実測したメモリバンド幅は42.9GB/sec
- この値で性能予測をすると14.0%となる
- 14.0%に比較すれば13.5%は良い値
- ループ長やコンパイラのバージョンによりプリフェッチ命令の利き方に差が出るためこれ以上は追求しない

空間微分Y方向の計算a)b)(3次元目の差分)

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

■ 第3軸が差分 → 再利用性なし

要求flop:

add：3 mult：2 = 5

要求B/F	24/5 = 4.8
性能予測	0.36/4.8= 0.075
実測値	0.076

要求Byteの算出:

1store/5loadより

(5+1) * 4byte = 24

空間微分Z方向の計算a)b)(ZXループ融合)

要求B/F値を下げる

```
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))*R42 &
                   - (V(k+1,I,J)-V(k-2,I,J))*R43
       DXV (k,I,J) = (V(k,I,J)  -V(k,I-1,J))*R40&
                   - (V(k,I+1,J)-V(k,I-2,J))*R41
     end do
   end do
 end do
```

■ K,I軸差分のループを融合することにより、V(K,I,J)のロードを共通化でき、プログラムの要求B/F値を下げる。

要求Byteの算出:

2store,3loadより、

(2+3)*4 = 20

要求flop:

add：6 mult：4 = 10

要求B/F	20/10 = 2
性能予測	0.36/2 = 0.18
実測値	0.174

空間微分Y方向の計算a)b)
(3次元目をcyclicでスレッド並列化)

```
!$OMP DO SCHEDULE(static,1),PRIVATE(I,J,K)
 do J = 1, NY
     do I = 1, NX
       do K = 1, NZ
         DYV (k,I,J) = (V(k,I,J)  -V(k,I,J-1))*R40 &
                     - (V(k,I,J+1)-V(k,I,J-2))*R41
       end do
     end do
  end do
```

キャッシュに載せる

- 第3軸をcyclic分割 → 1ストリームで3配列がL2に乗る(説明次項)
- 性能が 2 倍になる

要求B/F	12/5 = 2.4
性能予測	0.36/2.4 = 0.15
実測値	0.136

要求Byteの算出:

1sore,2loadと考える

4x3 = 12byte

要求flop:

add : 3 mult : 2 = 5

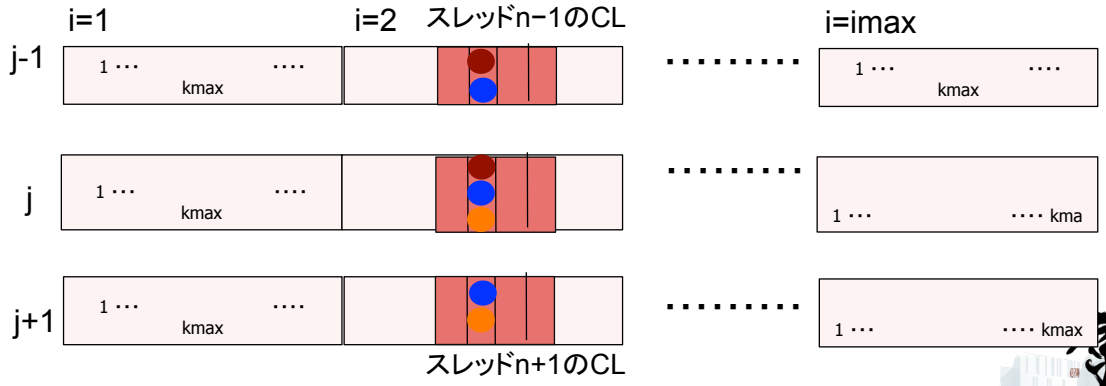
(cyclic分割スレッド並列の説明)

プログラム例

```
Do j=1,jmax
  do i=1,imax
    do k=1,kmax
      a(k,i,j)=・・・c0*v(k,i,j-1)+c1* v(k,i,j)+c2* v(k,i,j+1)・・・
    end do
  end do
end do
```

● :スレッドn−1で参照するデータ
● :スレッドnで参照するデータ
● :スレッドn+1で参照するデータ

キャッシュライン(CL)



空間微分Y方向の計算a)b)
(ZXYループ融合cyclicスレッド並列)

```
!$OMP DO SCHEDULE(static,1)
   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))*R42 &
                     - (V(k+1,I,J)-V(k-2,I,J))*R43
         DXV (k,I,J) = (V(k,I,J)  -V(k,I-1,J))*R40&
                     - (V(k,I+1,J)-V(k,I-2,J))*R41
         DYV (k,I,J) = (V(k,I,J)  -V(k,I,J-1))*R40 &
                     - (V(k,I,J+1)-V(k,I,J-2))*R41
       end do
     end do
   end do
```

要求B/F	28/15 = 1.86
性能予測	0.36/1.86 = 0.19
実測値	0.177

要求B/F値を下げる
キャッシュに載せる

- K,I,J軸差分のループを融合することにより、V(K,I,J)のロードを共通化でき、プログラムの要求B/F比を下げる。

要求Byteの算出:

Store 3 +4 load と考えると、

(3+4)*4 = 28byte

要求flop:

add：9 mult：6 = 15

速度時間積分の計算 e)

オリジナルコードの結果

要求B/F	72/52=1.38
性能予測	0.36/1.38 = 0.26
実測値	0.240

Seism3D全体のチューニング結果

(*)通信を含む性能

The number of node	Elapse time(sec)	Ratio to peak performance
16	48.8	17.1%
256	48.8	17.6%
4096	48.9	17.7%
16384	48.8	17.8%
36864	48.6	17.9%
64512	48.5	17.9%
82944	48.5	17.9%

- 8万ノードまでの良好なウィークスケーラビリティを得られた
- フルノードでトータル性能**1.9Pflops**を達成

Front Flow/blueのチューニング

Front Flow/blue(FFB)の概要

- 有限要素法を用いた流体計算のプログラム
- 有限要素法には2つのタイプの計算方法がある
  - 全体剛性マトリクスを構築するタイプ
  - 全体構成マトリクスを構築せずに要素剛性マトリクスのみで計算を進めるタイプ（エレメント・バイ・エレメント法）
- FFBは新バージョンにおいて両方のソルバに対応
- 本講演の疎行列とベクトルの積は前者のソルバで使用される計算カーネル

オリジナルコードの性能予測(理想的なケース)

オリジナルコード

```
ICRS=0
    DO 110 IP=1,NP
      BUF=0.0E0
      DO 100 K=1,NPP(IP)
        ICRS=ICRS+1
        IP2=IPCRS(ICRS)
        BUF=BUF+A(ICRS)*S(IP2)
100   CONTINUE
      AS(IP)=AS(IP)+BUF
110 CONTUINE
```

リスト　ベクトル　行列

要求B/F	8/2 = 4
性能予測	0.36/4 = 0.09

(スレッド並列を仮定しピーク性能128Gflopsに対して)

- ベクトルの部分がL1キャッシュに載っていると仮定した場合
- ベクトルのメモリへのアクセスを全く無視してよい
- メモリからのロードは行列とリストのみ

要求Byteの算出:

単精度：2 load なので

2*4 = 8byte

要求flop:

add：1 mult：1 = 2

オリジナルコードの性能予測と実測 (スレッド並列なし：1コア)

- メモリバンド幅を1コアで占有する場合のSTREAMベンチマークの結果は20GB/秒
- 1コアの理論ピーク性能は16GFLOPS
- 従って理論的なB/F値は20GB/16GFLOPで1.25

要求Byteの算出：2loadより 2* 4byte = 8
要求flop：1(add)+1(mult) = 2

要求B/F	8/2 = 4
性能予測	1.25/4= 0.313
実測値	0.059(六面体) 0.024(四面体)

(スレッド並列なしピーク性能**16Gflops**に対して)

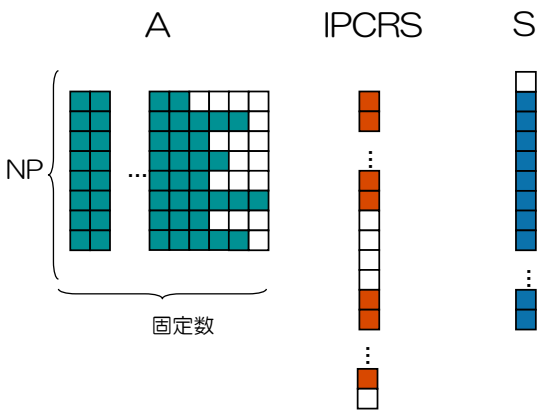
- ベクトルがリストアクセス
- 連続アクセスでないためプリフェッチが効きにくい
- メモリアクセスのレイテンシが見える
- 1ラインのうち1要素しか使用しない事による大きなペナルティが発生
- 著しい性能低下が発生
- L2オンキャッシュでも同様のペナルティが発生

チューニング1: フルアンロール

狙い：
- スケジューリングの改善(演算待ちの削減)

変更点:
- 行列要素の配列に0を代入
- ベクトルインデックスの配列に0を代入
- 余分な配列同士は0*0の演算を実施

```
 ICRS=0
       DO 110 IP=1,NP
         BUF=0.0E0
!        DO 100 K=1,NPP(IP)   MAX_NZ=27
           BUF=BUF+A(ICRS+ 1)*S(IPCRS(ICRS+ 1))
     &           +A(ICRS+ 2)*S(IPCRS(ICRS+ 2))
     &           +A(ICRS+ 3)*S(IPCRS(ICRS+ 3))
     &           +A(ICRS+ 4)*S(IPCRS(ICRS+ 4))
      ・・・・・・・・・・・・・・・・・・・・
      ・・・・・・・・・・・(省略)・・・・・
      ・・・・・・・・・・・・・・・・・・・・
     &           +A(ICRS+24)*S(IPCRS(ICRS+24))
     &           +A(ICRS+25)*S(IPCRS(ICRS+25))
     &           +A(ICRS+26)*S(IPCRS(ICRS+26))
     &           +A(ICRS+27)*S(IPCRS(ICRS+27))
           ICRS=ICRS+27
! 100     CONTINUE
           AS(IP)=AS(IP)+BUF
  110 CONTINUE
```

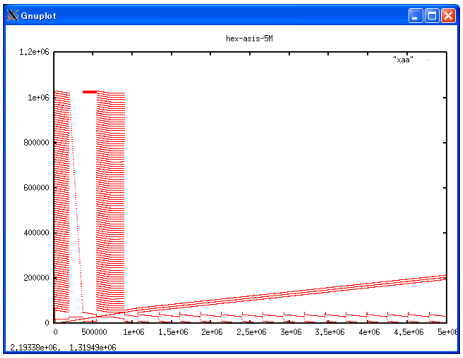


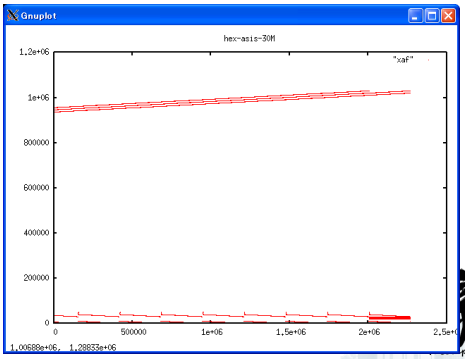
チューニング2: リオーダリング (1/4)

狙い：
■ベクトルデータ(S)のブロック化によるL1,L2キャッシュミスの削減
オリジナルデータの特徴：
■６面体(総回転数: 約2700万)
- 最初の1M回のSへの広範囲なランダムアクセス
- それ以降は二極化するが、局所的アクセス

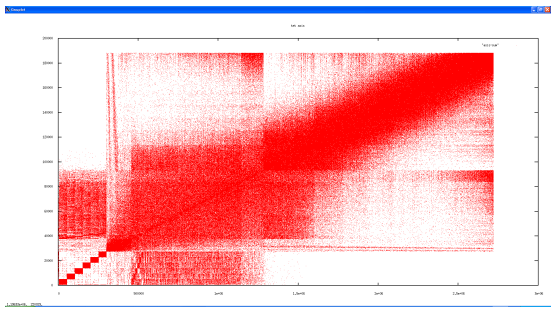
■４面体(総回転数:約270万)
- 全アクセスとも、広範囲なランダムアクセス

■ ６面体 オリジナルデータ





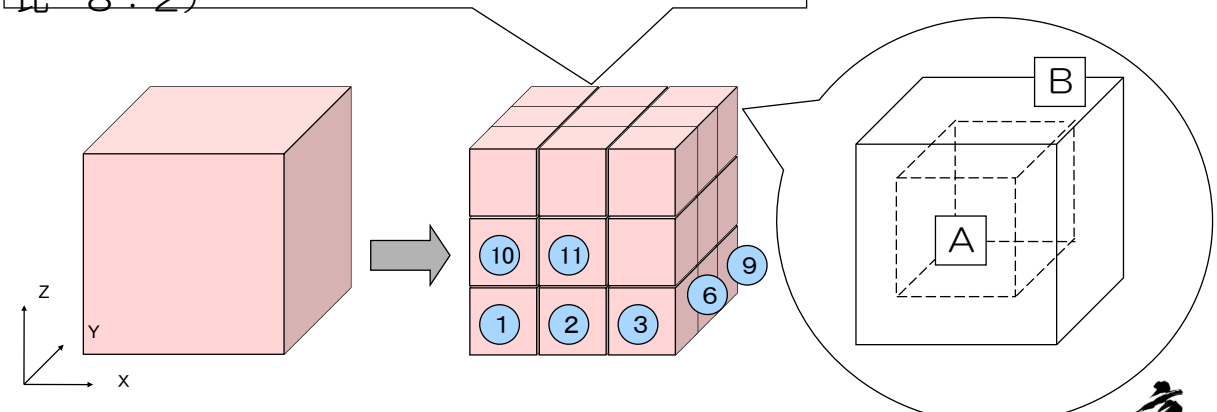
■ ４面体 オリジナルデータ



チューニング2: リオーダリング (2/4)

節点番号のリオーダリング:
■オリジナルデータを各軸分割しブロックを作成
■各ブロックを外と内に分割し物理座標に基づき内側・外側の順にナンバリング

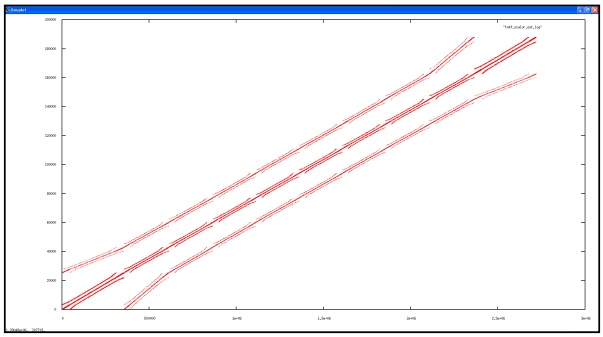
ひとつのブロックを内と外に分け、内側(A)のナンバリング後、外側(B)のナンバリングを実施(内:外の比　８：２)



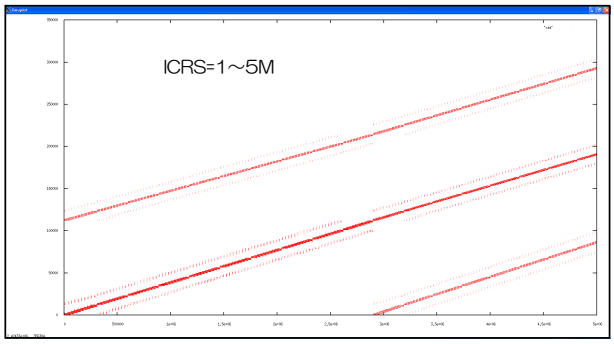
チューニング2: リオーダリング (3/4)

- 物理的に近い節点が配列の並びとしても近い位置に配置される事を期待
- 一要素を構成する節点の番号が近くなる
- 一箱の大きさを調整することによりベクトルのリストアクセスの多くに対しL1オンキャシュのデータを利用できる

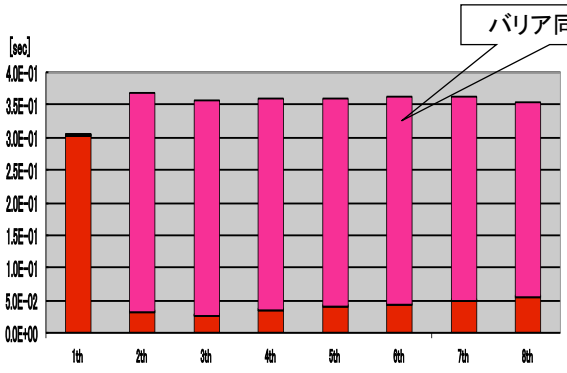
■ 4面体 リオーダリング結果



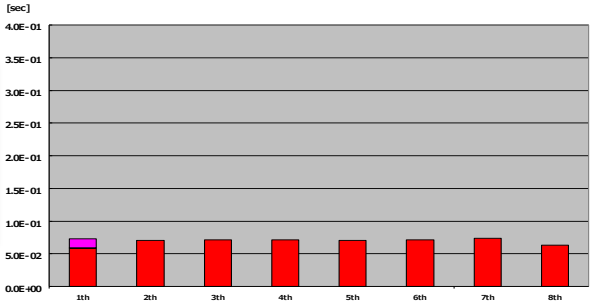
■ 6面体 リオーダリング結果



チューニング2: リオーダリング (4/4)



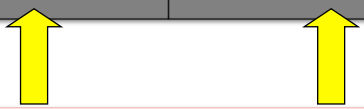
リオーダリング前
■マスタースレッドが担当する範囲にランダムアクセスが集中しメモリアクセスコストが増大
■スレーブスレッドが担当する範囲は局所アクセスが多く結果としてバリア同期待ちのコストが支配的



リオーダリング後
■全てのスレッドが均一的に局所アクセス化
■メモリアクセスバランスの改善と同期コストの大幅な削減

FFBカーネルの結果まとめ

	６面体	４面体
オリジナル(1core)	5. 9%	2. 4%
フルアンロール (1core)	10. 8%	4. 2%
フルアンロール (8core)	5. 4%	3. 0%
フルアンロール ＋ リオーダリング (1core)	10. 2%	10. 2%
フルアンロール ＋ リオーダリング (8core)	8. 1%	7. 7%



L1 オンキャッシュである時の理論性能値である9%に近い性能値を実現

まとめ

- 「京」の概要を示した
- 理研で進めているアプリ高性能化の概要を示した
- 高並列化のための重要点を示した
- 高い単体性能を得るための重要点示した
- RSDFT，PHASE，Seism3D,FFB等を例として高並列化，チューニング手法を示した.