FUJITSU

shaping tomorrow with you

京アプリケーション高速化の現状

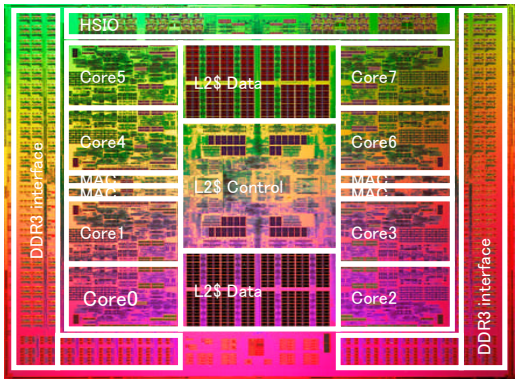
2014年1月23日

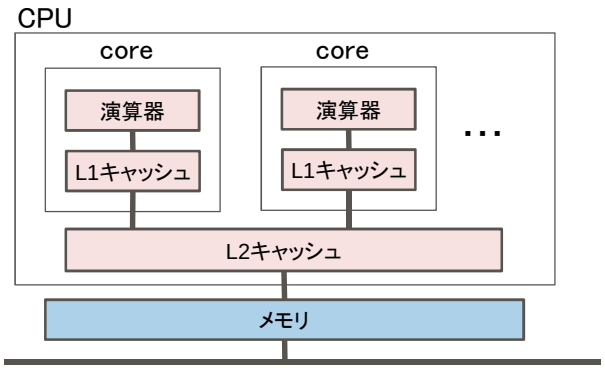
富士通株式会社

FUJITSU

1. スパコンは速い？

1-1. CPU（SPARC64™ VIIIfx）





- アーキテクチャ
  - 8 コア
  - 6 MB の共有L2キャッシュ
  - クロック 2 GHz
- 45nm CMOS
  - 22.7mm x 22.6mm
  - 760M トランジスタ
  - 信号ピン数 1271
- ピーク性能
  - 演算性能　128GFlops
  - メモリスループット　64GB/s
- 消費電力
  - 58W（TYP, 30℃）
  - リーク電流削減

PC並?

1-2. CPUのクロック周波数（単体CPU）

FUJITSU

他社の動向

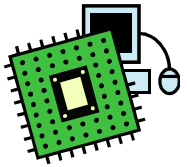
Intel Core i7-4770 ：3.9GHz，4コア，8Mキャッシュ

AMD FX-9590 ：4.7GHz，8コア，8Mキャッシュ

クロック→限界！
キャッシュ→限界！

10年前と比較してクロックの向上率は低下している

既存アーキの延長では、大きな性能向上は困難



既存マシンを「効率的に使用」する

ハードを有効活用するためには、チューニングが必要！

（「現状のマシンを100%使いこなす」という意気込み）

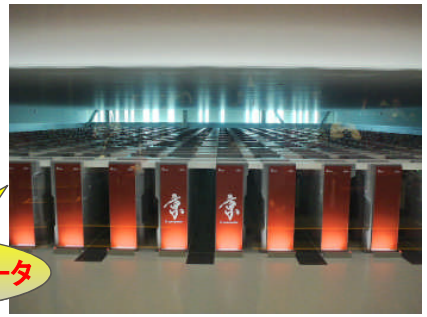
1-3. CPUの個数（ノード構成）

FUJITSU

京コンピュータは、約8万個のCPUで構成されている

ピーク性能　：10.62Pflops
メモリ総量　：1.26PB
ディスク総量：30PB
ネットワーク：5GB/s×2(双方向)

スーパーコンピュータ



京コンピュータを有効に使う

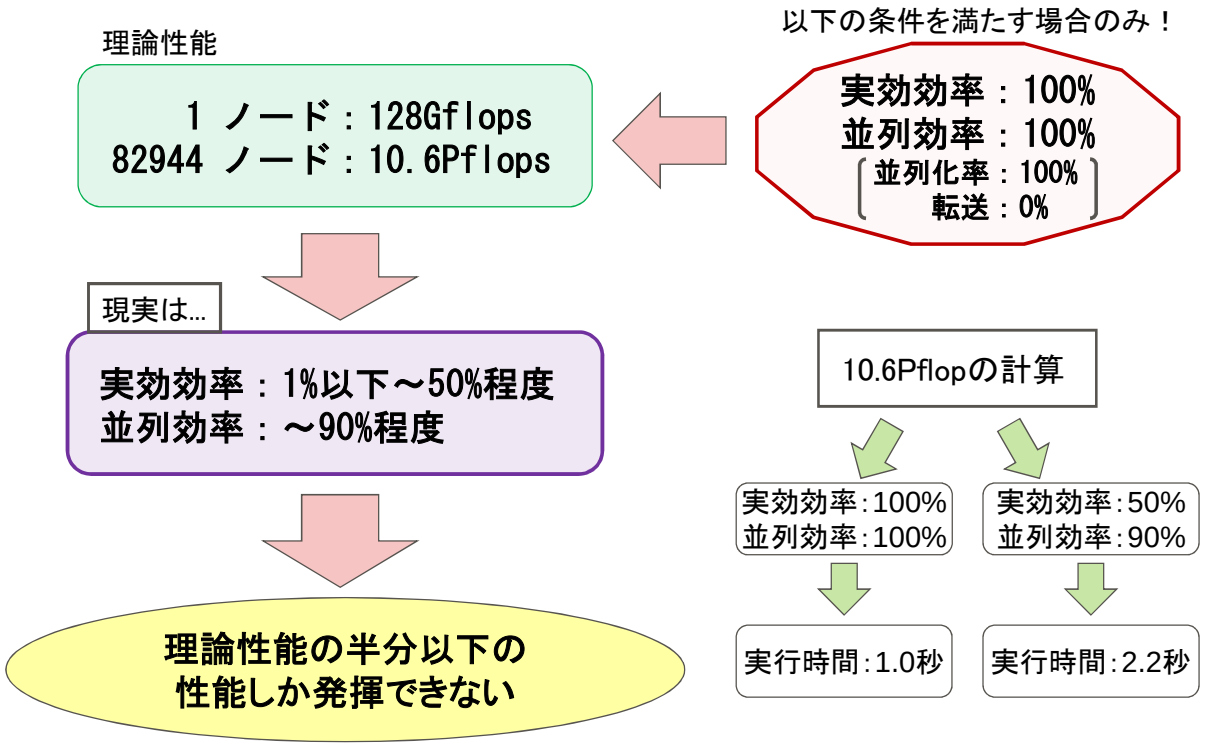
8万個のCPUを効率的に使用する必要がある。
（ノード間データ転送も考慮する必要アリ）

チューニングが必要

京コンピュータを有効活用するためには、
単体CPUとノード間通信の両方のチューニングが必要！

1-4. 実効効率について（チューニングの必要性）

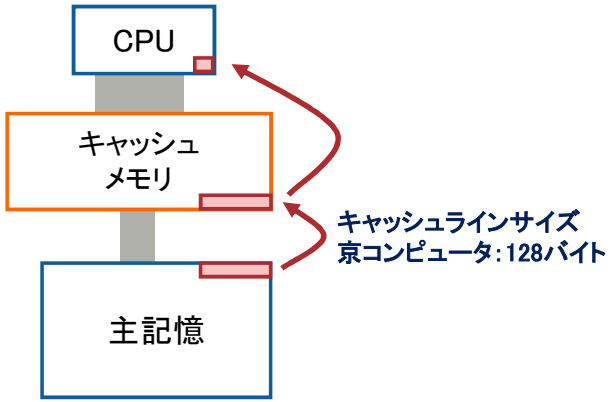
FUJITSU



1-5. なぜ実効効率が悪い？（キャッシュラインの例）

実効効率100%を発揮できない原因は、同時に多数存在する場合が一般的です。
そのうち、皆様に意識して頂きたい点を一つだけ、お知らせします。

キャッシュメモリと主記憶間のデータの移動は、決まった大きさの連続領域（キャッシュライン）単位で行われる。



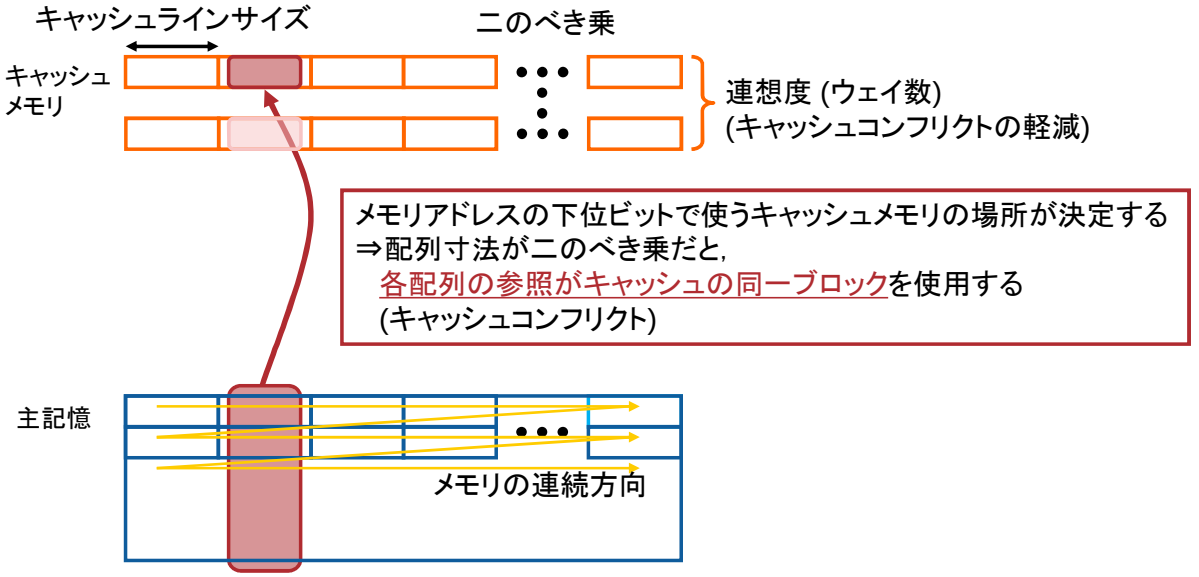
ランダムに配列アクセスすると、メモリとキャッシュ間で無駄なデータ転送が発生するため、実効効率が悪くなる。

1-5. なぜ実効効率が悪い？ (マッピングの例)

FUJITSU

キャッシュメモリの使用場所の決定方法

2冪サイズの配列 → キャッシュ競合の可能性アリ → 実効効率悪化の可能性アリ



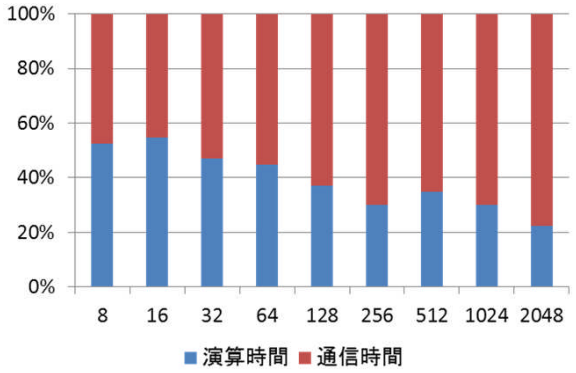
1-6. なぜ実効効率が悪い？ (通信の例)

FUJITSU

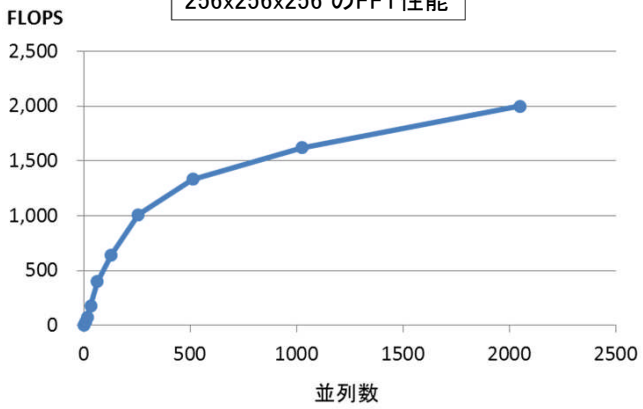
フーリエ変換(FFT)の並列特性を以下に示す。右グラフを見ると、1000並列以上では性能が向上しない様子が分かる。通信がボトルネックになり、演算性能が劣化する。

Alltoall通信がボトルネック

256x256x256 FFTの演算と転送の比率



256x256x256 のFFT性能



FUJITSU

2. チューニング

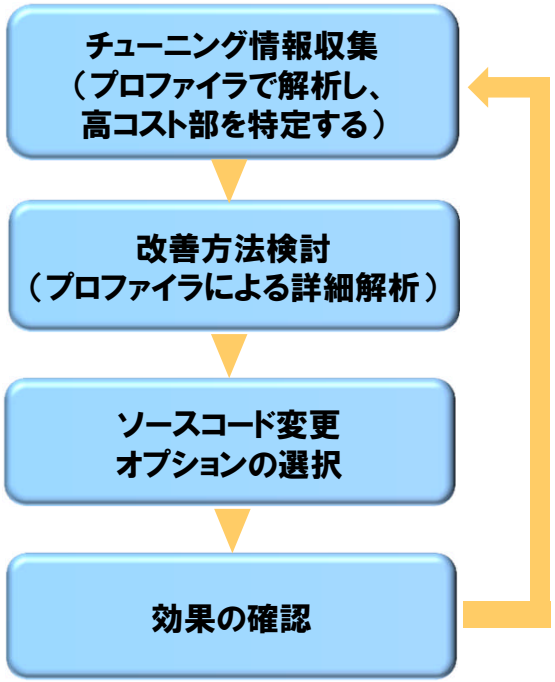
FUJITSU

2.1. プログラムのチューニング

ハードおよびコンパイラの最適化を意識してプログラミングする。

2-1-1. チューニングの流れ

FUJITSU



プログラムの実行時間を短縮することをチューニングという。

プロファイラを使用し、高コスト処理を特定し、必要な対策を施すことで、実行時間を短縮する。

2-1-2. チューニング事例(パディングの例)

FUJITSU

下記のようなプログラムでプロファイルを採取する

```
!--- Parameters -------
   integer,  parameter :: NMAX =256
!---- Data ------------
 double precision :: A(NMAX,NMAX,NMAX+1)
 double precision :: B(NMAX,NMAX,NMAX+1)
 double precision :: C(NMAX,NMAX,NMAX+1)
 double precision :: X(NMAX,NMAX,NMAX)
 double precision :: Y(NMAX,NMAX,NMAX)

 do k = 1, NMAX
   do j = 1, NMAX
     do i = 1, NMAX
       X(i,j,k) = A(i,j,k) + B(i,j,k)
       Y(i,j,k) = X(i,j,k) + B(i,j,k+1)*C(i,j,k)
     enddo
   enddo
 enddo
```

fippでコスト分布を採取し、左記の処理コストが高いと特定したとする。

2-1-3. チューニング事例(プロファイル採取)

FUJITSU

fappを使用し、「-Hevent=Performance」で採取した情報で、最も時間を要しているイベントを探す。
今回の場合は、キャッシュアクセス待ちに約70%の時間を要している。
キャッシュアクセスに問題があることが分かる。

Kind	Elapsed(s)	2-4icmit(S)	1i_cmit(S)	op_wait(S)	cache_wait(S)
AVG	2.4309	0.5700	0.0435	0.0835	1.7226
MAX	2.4328	0.5700	0.0435	0.0835	1.7245
MIN	2.4290	0.5700	0.0435	0.0835	1.7207

Kind	Elapsed(s)	Mem_wait(S)	fetch_wait(S)	other_wait(S)
AVG	2.4309	0.0001	0.0024	0.0089
MAX	2.4328	0.0001	0.0024	0.0089
MIN	2.4290	0.0001	0.0024	0.0089

2-1-4. チューニング事例(チューニング)

FUJITSU

パディングしたプログラムでプロファイルを採取する

```
!--- Parameters -------
   integer,  parameter :: NMAX =256
   integer,  parameter :: NPAD =2
!---- Data ------------
 double precision :: A(NMAX+NPAD,NMAX,NMAX+1)
 double precision :: B(NMAX+NPAD,NMAX,NMAX+1)
 double precision :: C(NMAX+NPAD,NMAX,NMAX+1)
 double precision :: X(NMAX+NPAD,NMAX,NMAX)
 double precision :: Y(NMAX+NPAD,NMAX,NMAX)

 do k = 1, NMAX
   do j = 1, NMAX
     do i = 1, NMAX
       X(i,j,k) = A(i,j,k) + B(i,j,k)
       Y(i,j,k) = X(i,j,k) + B(i,j,k+1)*C(i,j,k)
     enddo
   enddo
 enddo
```

配列の1次元目をパディングする
これで、キャッシュラインがずれる

2-1-5. チューニング事例(効果の確認)

fappを使用し、「-Hevent=Performance」で採取した情報で、キャッシュアクセス待ちの割合を見る。
対策を施すと、キャッシュアクセス待ちは約10%程度に減少し、実行時間も低減する。

Kind	Elapsed(s)	2-4icmit(S)	1i_cmit(S)	op_wait(S)	cache_wait(S)
AVG	1.1317	0.5703	0.1681	0.1253	0.1323
MAX	1.1318	0.5703	0.1681	0.1253	0.1323
MIN	1.1317	0.5703	0.1681	0.1253	0.1323

Kind	Elapsed(s)	Mem_wait(S)	fetch_wait(S)	other_wait(S)
AVG	1.1317	0.0002	0.0007	0.1349
MAX	1.1318	0.0002	0.0007	0.1349
MIN	1.1317	0.0002	0.0006	0.1349

2-1-6. その他のチューニング

FUJITSU

チューニング手法は多数あり、これらを組み合わせて高速化を実現する。
（経験と勘が必要？？）

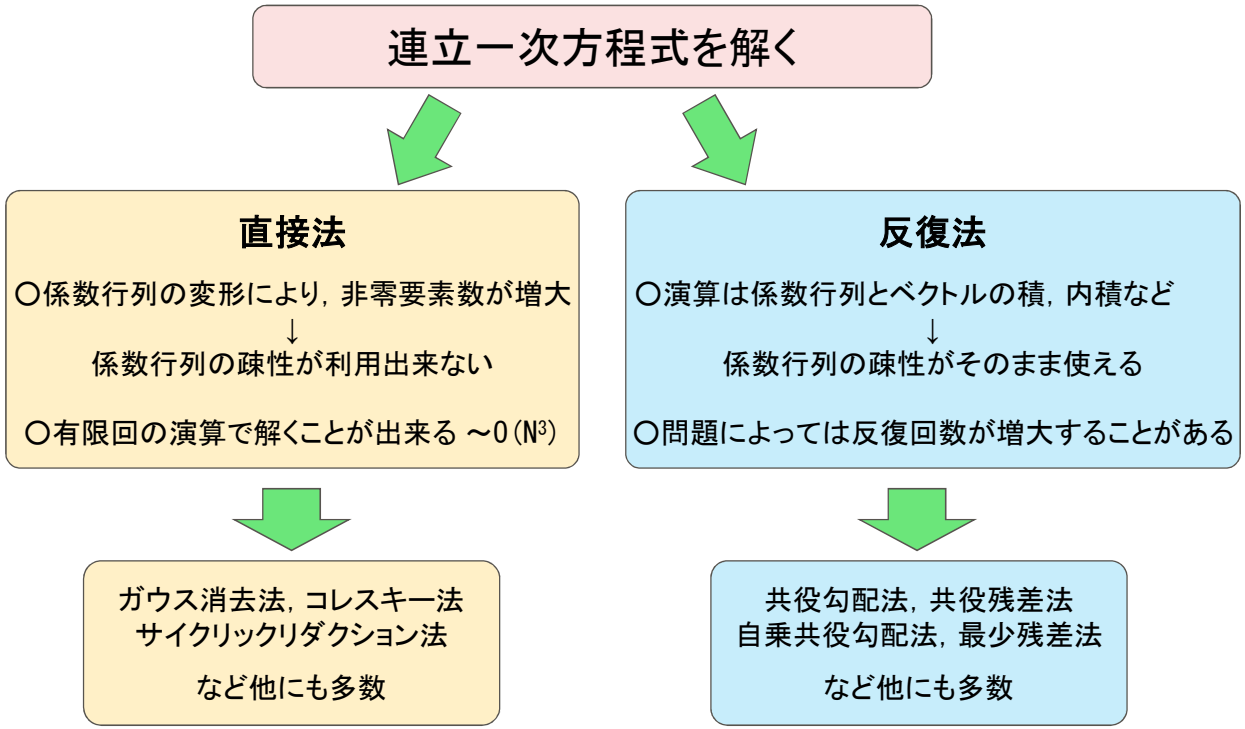
項目	目的	説明
連続アクセス化	キャッシュミスを低減する。 プリフェッチを有効に利用する。	データはメモリからキャッシュライン単位で読み込まれる。オンキャッシュの状態でキャッシュラインの全要素を参照できるとベスト。
パディング	キャッシュ競合を回避する。	配列をパディングすることで キャッシュへのマッピング状況を変えて、競合が発生しないようにする。
ループ融合	キャッシュ上の値を参照することで効率を向上させる。	ループ間で作業配列を使用 している場合は、作業配列を 削除することで、高速化する。
作業配列の次元縮小	メモリアクセスを低減する。	不要なメモリアクセスを無くすことで高速化する。
配列のマージ	ロード回数を低減する。	メモリからの読み込み回数を低減することで高速化する。

FUJITSU

2.2. アルゴリズムの検討

2-2-1. 数値解析手法を検討する

FUJITSU



2-2-2. シミュレーション手法を再検討する

FUJITSU

高並列で実行する必要がある。（数千～数万並列のレベル）

↓

高並列に対応した計算手法を検討する

（第一原理：平面波展開法 → 実空間法
分子動力学：PME法 → FMM法）

↓

研究者の皆様、頑張ってください！

研究を効率よく推進できるよう、我々も頑張ります

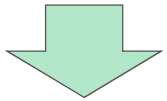
FUJITSU

3. まとめ

3. まとめ

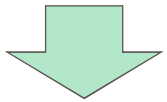
FUJITSU

既存アーキテクチャの延長では、
クロック，キャッシュ共に限界に達している



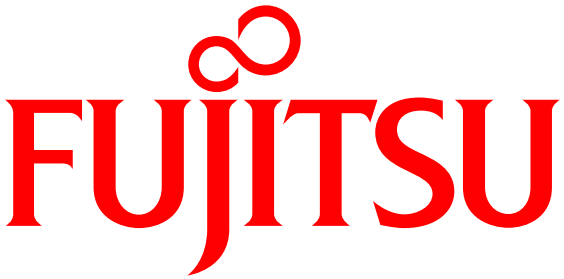
ブレイクスルーが無ければ

既存コンピュータを有効に活用する



どうすれば良い？？

ハードを有効活用するためには、チューニングが必要！



shaping tomorrow with you