チューニング入門

2007年6月13日

中島研吾

並列計算プログラミング（616-2057）・先端計算機演習I（616-4009）

- チューニングとは
- ベクトルプロセッサとスカラープロセッサ
- 具体的チューニング例
  - スカラープロセッサー
  - ベクトルプロセッサー

- チューニングとは
- ベクトルプロセッサとスカラープロセッサ
- 具体的チューニング例
  - スカラープロセッサー
  - ベクトルプロセッサー

チューニングとは

- チューニングの目的
  - 計算時間の削減
  - 最適化
- チューニングの実施方法
  - アルゴリズムの変更
  - ハードウェアに応じた修正，最適化
  - もちろん計算結果に影響を及ぼすようなものであってはならない
    - またはその効果を承知していなければならない

「チューニング」との向き合い方・・・

- そもそもどういうタイミングでチューニングをやるのか？
  - プログラムができてしまってから，チューニングをするのは（誰がやっても）大変な作業
- 最初に作るときから，ある程度，チューニングに関することを考慮するべきである
  - いくつかの心得
- 「わかりやすい」，「読みやすい」プログラムを作ること
  - バグの出にくいプログラムを作る・・・ということでもある

- 最初の授業で「良い並列プログラム＝良いシリアルプログラム」と言ったこととつながる。
- チューニング⇒高速⇒研究サイクルの効率化・・・

チューニング：参考文献

- スカラープロセッサ
  - 寒川「RISC超高速化プログラミング技法」，共立出版，1995.
  - Dowd（久良知訳）「ハイ・パフォーマンス・コンピューティング-RISCワークステーションで最高のパフォーマンスを引き出すための方法」，トムソン，1994.
  - Goedecker, Hoisie “Performance Optimization for Numerically Intensive Codes”, SIAM, 2001.
- 自動チューニング
  - 片桐「ソフトウェア自動チューニング」，慧文社，2004.
- ベクトルプロセッサ
  - 長島，田中「スーパーコンピュータ」，オーム社，1992.
  - 奥田，中島「並列有限要素解析」，培風館，2004.

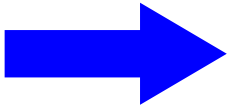
「チューニング」の心得

- 特に大規模データに関しては，メモリアクセスパターンに配慮する。
- 最内側ループでサブルーチン，関数コールは避ける。
  - 最近のコンパイラでは「インライン展開」が可能であるが，うまく働かない場合もある。
  - IF文もできるだけ避ける。
- 多重DOループを避ける。
- 除算，組み込み関数の多用を避ける。
- 無駄な計算はなるべく排除する。
  - 記憶できるところは記憶しておく。
  - メモリ容量との兼ね合い。
- **H/W，コンパイラ依存性は残念ながら大きい。**
  - **実際に試して速い手法を採用するべき**

例：多重DOループ

- 1つのDOループに到達するごとに，ループカウンタの初期設定というオーバーヘッドが生じる。
  - 例えば，下左の例では，最内ループに1,000,000回到達するため1,000,000回のオーバーヘッドが生じる。
  - 右のように展開してしまった方が良い。

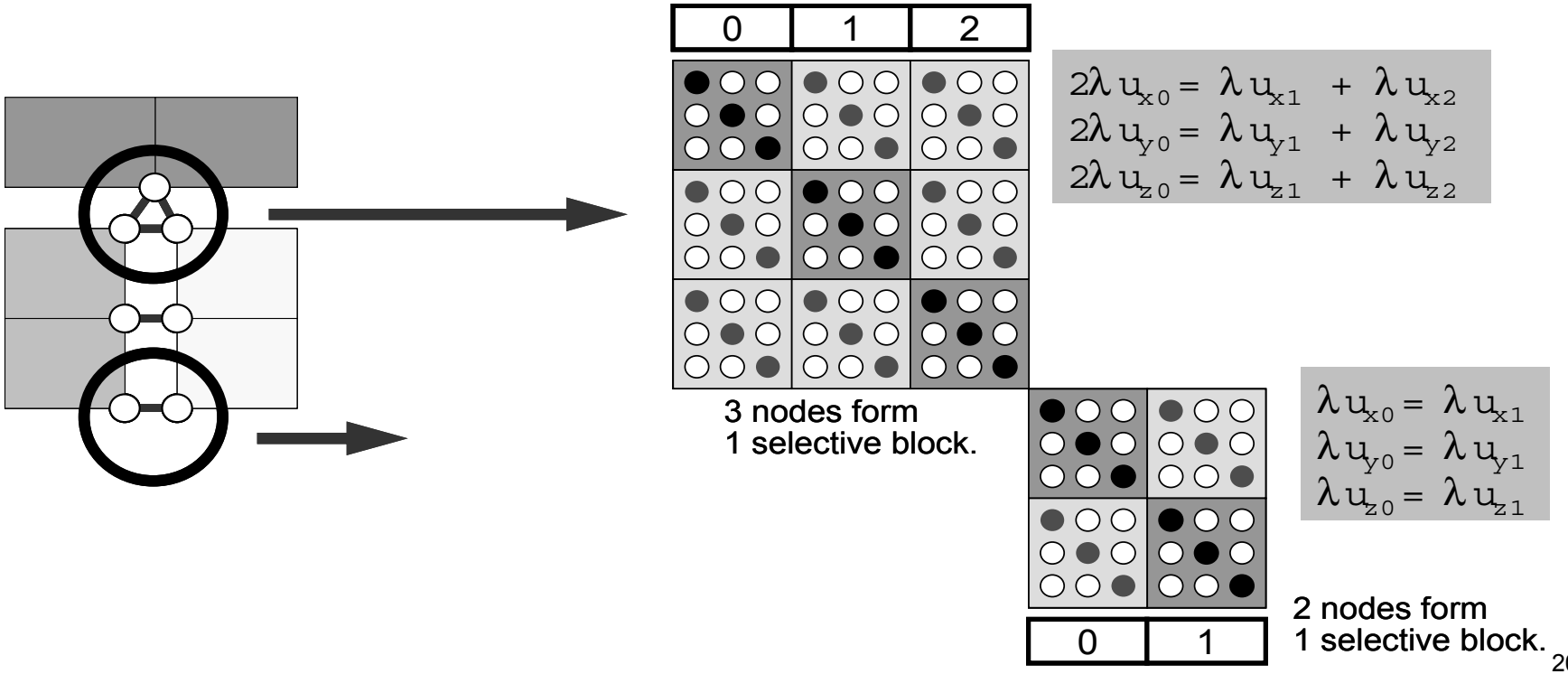
```
real*8 AMAT(3,1000000)
. . .
do j= 1, 1000000
  do i= 1, 3
    A(i,j)= A(i,j) + 1.0
  enddo
enddo
. . .
```



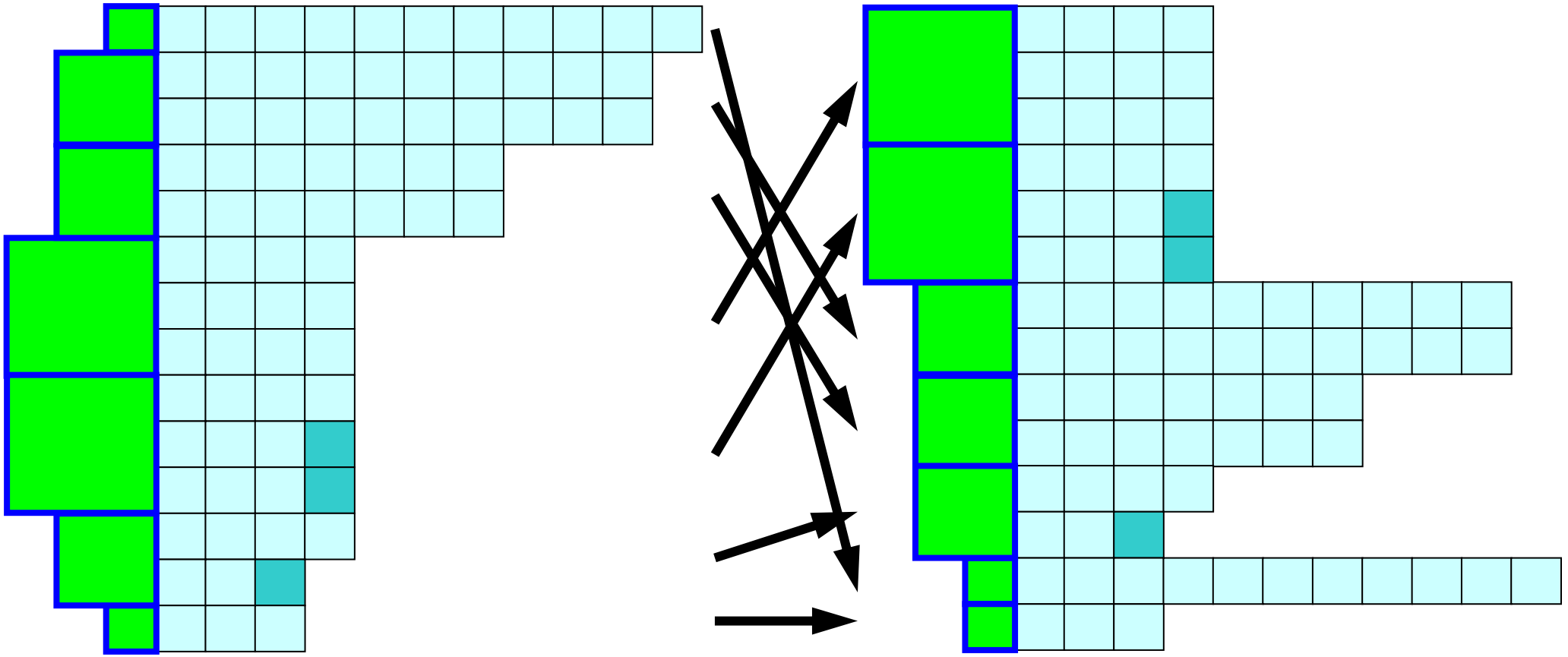
```
real*8 AMAT(3,1000000)
. . .
do j= 1, 1000000
  A(1,j)= A(1,j) + 1.0
  A(2,j)= A(2,j) + 1.0
  A(3,j)= A(3,j) + 1.0
enddo
. . .
```

IF文について・・・

- 例えば，「地球シミュレータ」では，最内ループにIF文があってもベクトル化される，と言われているが，性能は明らかに低下する。
  - 接触問題（CS06で話した）の例（Selective Blocking）



Selective Blockのサイズに従って順番入れ替え（re-ordering）・・・



何故並べ替え(re-ordering)が必要か?

- “IF-THEN-ELSE” 文を排除できる
- “IF-THEN-ELSE” 文を含んでいてもベクトル化はされるが，性能は悪い。
  - かってはベクトル化すらできなかった。

並べ替え無し

```
do i= 1, NN
  if (BLKsiz=1) then
    (...)
   else if
     (BLKsiz=2) then
    (...)
   else
    (...)
  endif
end
```

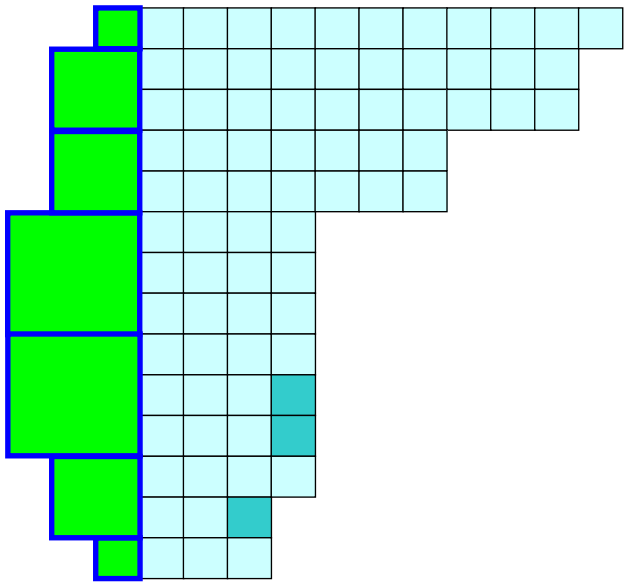
並べ替えあり

```
do i= 1, NN(1)
  (...)
enddo

do i= NN(1)+1, NN(2)
  (...)
enddo

do i= NN(2)+1, NN(3)
  (...)
enddo
```

何故並べ替え(re-ordering)が必要か?

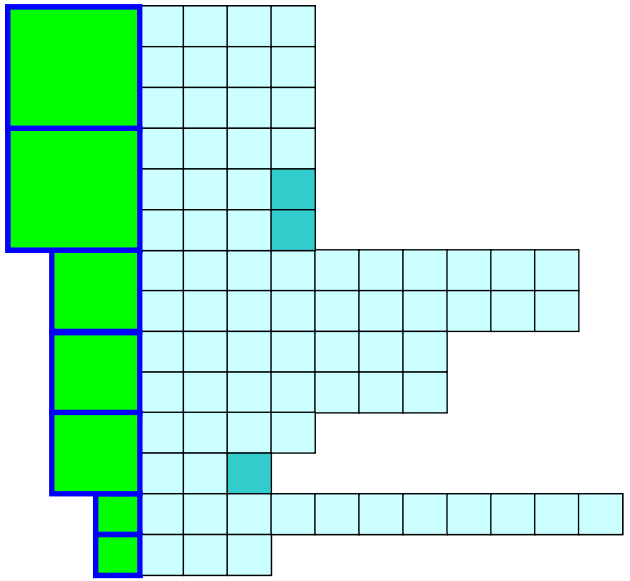
並べ替え無し

```
do i= 1, NN
  if (BLKsiz=1) then
    (...)
   else if
     (BLKsiz=2) then
    (...)
   else
    (...)
  endif
end
```



並べ替えあり

```
do i= 1, NN(1)
  (...)
enddo

do i= NN(1)+1, NN(2)
  (...)
enddo

do i= NN(2)+1, NN(3)
  (...)
enddo
```

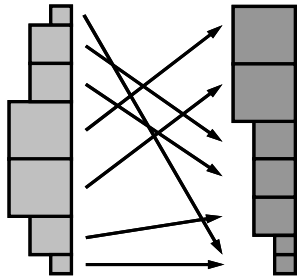


Selective Blocking：並べ替えの効果

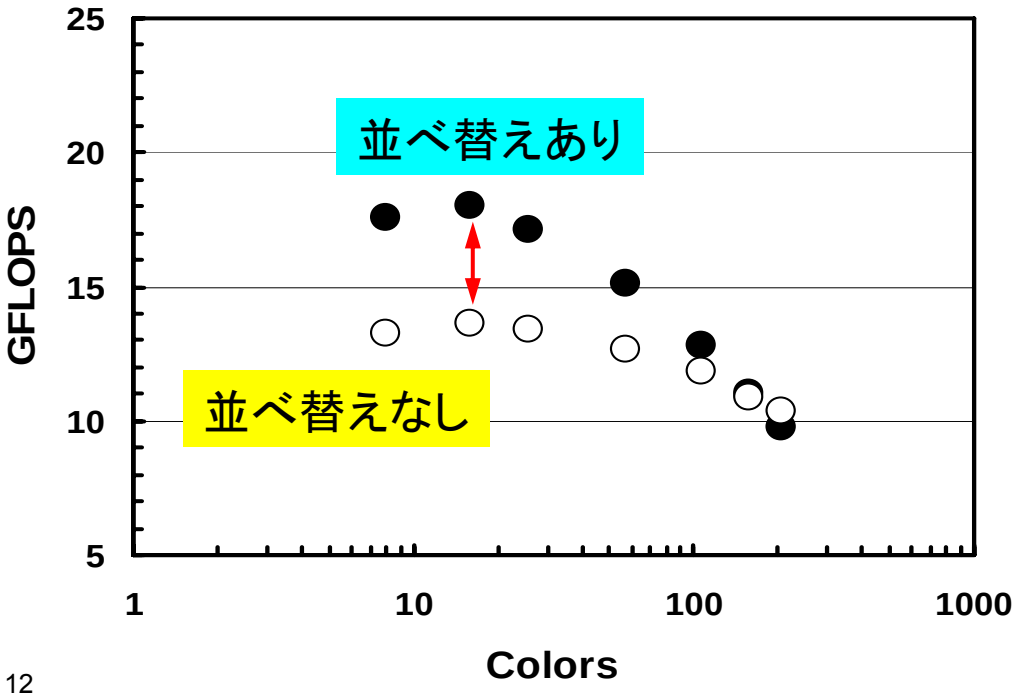
1 SMP Node of ES, 64 GFLOPS Peak Performance

● with Reordering, ○ without Reordering

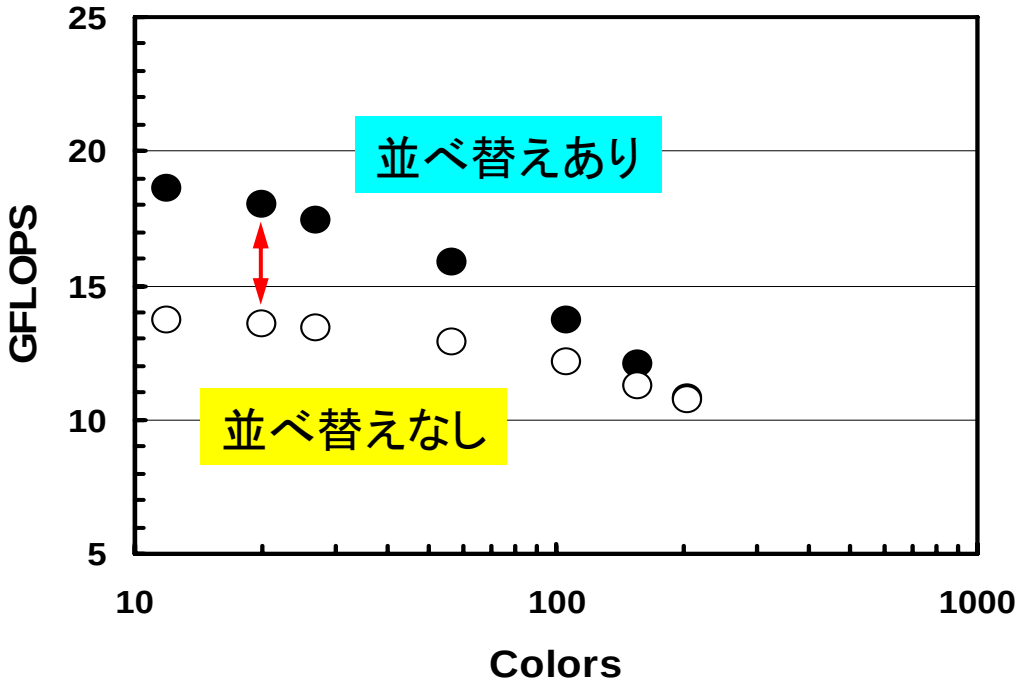
並べ替えをした方が概して性能が良い



Simple Block



西南日本モデル



簡単に実施できること:パフォーマンス測定

- 「time」コマンド
- 「timer」サブルーチンによる測定
- 「プロファイリング（profiling）」ツールの使用
  - ホットスポットの特定
  - gprof（UNIX）
  - pgprof: PGIコンパイラ専用
  - trace:「地球シミュレータ」

ファイルコピー：今日はFORTRANのみ

簡単なプログラムなのでCプログラマは自分で作ってください

```
>$ cd <$07S>

FORTRAN
>$ cp /home/nakajima/class/2007summer/F/s4-f.tar .
>$ tar xvf s4-f.tar
```

直下に「/S4」というディレクトリができている。
<$07S>/S4を<$S4>と呼ぶ。

gprof

- サブルーチンごとにCPU時間，呼び出し回数が表示される。
  - どのサブルーチンから何回コールされ，どのサブルーチンを何回コールしたかが表示される。
- 「gmon.out」が生成される。

```
$> cd <$S4>

$> pgf90 -O3 -pg test.f
$> a.out
$> ls gmon.out
   gmon.out

$> gprof > out.lst
```

pgprof

- 「gprof」のPGIコンパイラ版。
- X-Windows環境であれば，GUIを使用して様々な機能を使用できる。ここではコマンドラインからの使用。
  - インタラクティヴに処理できる。
- MPIを使用した場合の，PEごとの評価などもできるはずなのであるが・・・うまく行かない。
- 利用法
  - コンパイルオプション「-Mprof=lines,func」を付けてコンパイル
  - 「pgprof.out」が生成される。
  - 「pgprof」と打ち込むと，コマンドライン環境に入る。

pgprof（続き）

```
$> pgf90 -O3 -Mprof=lines,func test.f
$> a.out

$> ls pgprof.out
   pgprof.out

$> pgprof
```

pgprofの利用法（1/4）

```
pgprof> help
    load - load a new data set
           Usage: lo[ad] [<data_file>]
   merge - merge the profile data in data_file into the current dataset
           Usage: m[erge] <data_file>
   shell - fork a shell using the given arguments
           Usage: sh[ell] <arg1,...,argn>
    sort - sort the data based on the given key
           Usage: so[rt] [by] [max|avg|min|all] calls|time/call|time|cost|name
  select - set selection filter key and cutoff level (decimal integer)
           Usage: sel[ect] calls|time/call|time|cost|all [[>] <cutoff>]
   times - specify whether times are displayed as raw values or percentages
           Usage: t[imes] raw|pct
   print - print the selected and sorted function level data
           Usage: p[rint] [[>] <filename>]
  srcdir - add a directory to the source search path
           Usage: src[dir] <directory_path>
   lines - display line level data for the named function
           Usage: l[ines] <function_name> [[>] <filename>]
 process - specify the process in a multiprocessing environment
           Usage: pro[cess] <process_num>
singleprocess - focus on a single process in a multithreaded multiprocessor run
           Usage: si[ngleprocess] <process_num>
  thread - specify the thread in a multithreaded multiprocessor run
           Usage: th[read] <thread_num>
 display - specify which multiprocessing fields are displayed
           Usage: d[isplay] <options>|all|none
    stat - specify which multiprocessing items are displayed
           Usage: s[tat] [no]min|[no]avg|[no]max|[no]proc|[no]thread|[no]all
    help - provide information about commands
           Usage: he[lp] [<command>]
 history - display list of previous commands
           Usage: hi[story] [<size>]
       ! - re-execute previous command
           Usage: !<num> or !-<num> or !<str> or !?<str>
      !! - re-execute previous command
           Usage: !!
       ? - same as help
           Usage: ? [<command>]
    quit - quit the profiler
           Usage: q[uit]
```

pgprofの利用法(2/4)

pgprof> select all 常に最初にタイプしておくと良い

pgprof> times raw times raw:実時間, times pct:時間比(%)

pgprof> print print:サブルーチンごとの経過時間

```
Profile output - Fri Jun 02 17:44:21 JST 2006
Program                                 : a.out
Datafile                                : pgprof.out
Process                                 : 0
Total Time for Process                  : 0.027933 secs
Sort by max time
Select all


                    Routine  Source  Line
Calls   Time(s)        Name    File   No.

    1  0.019516          cg  test.f   655
    1  0.007136        MAIN  test.f     1
  624  0.001107     csearch  test.f   624
  100  0.000174      jacobi  test.f   816
```

pgprofの利用法(3/4)

```
pgprof> times pct
```

times raw:実時間，times pct:時間比(%)

```
pgprof> print


Profile output - Fri Jun 02 17:44:21 JST 2006
Program                                 : a.out
Datafile                                : pgprof.out
Process                                 : 0
Total Time for Process                  : 0.027933 secs
Sort by max time
Select all


                    Routine  Source  Line
Calls   Time(s)        Name    File   No.

    1  0.019516          cg  test.f   655
    1  0.007136        MAIN  test.f     1
  624  0.001107     csearch  test.f   624
  100  0.000174      jacobi  test.f   816
```

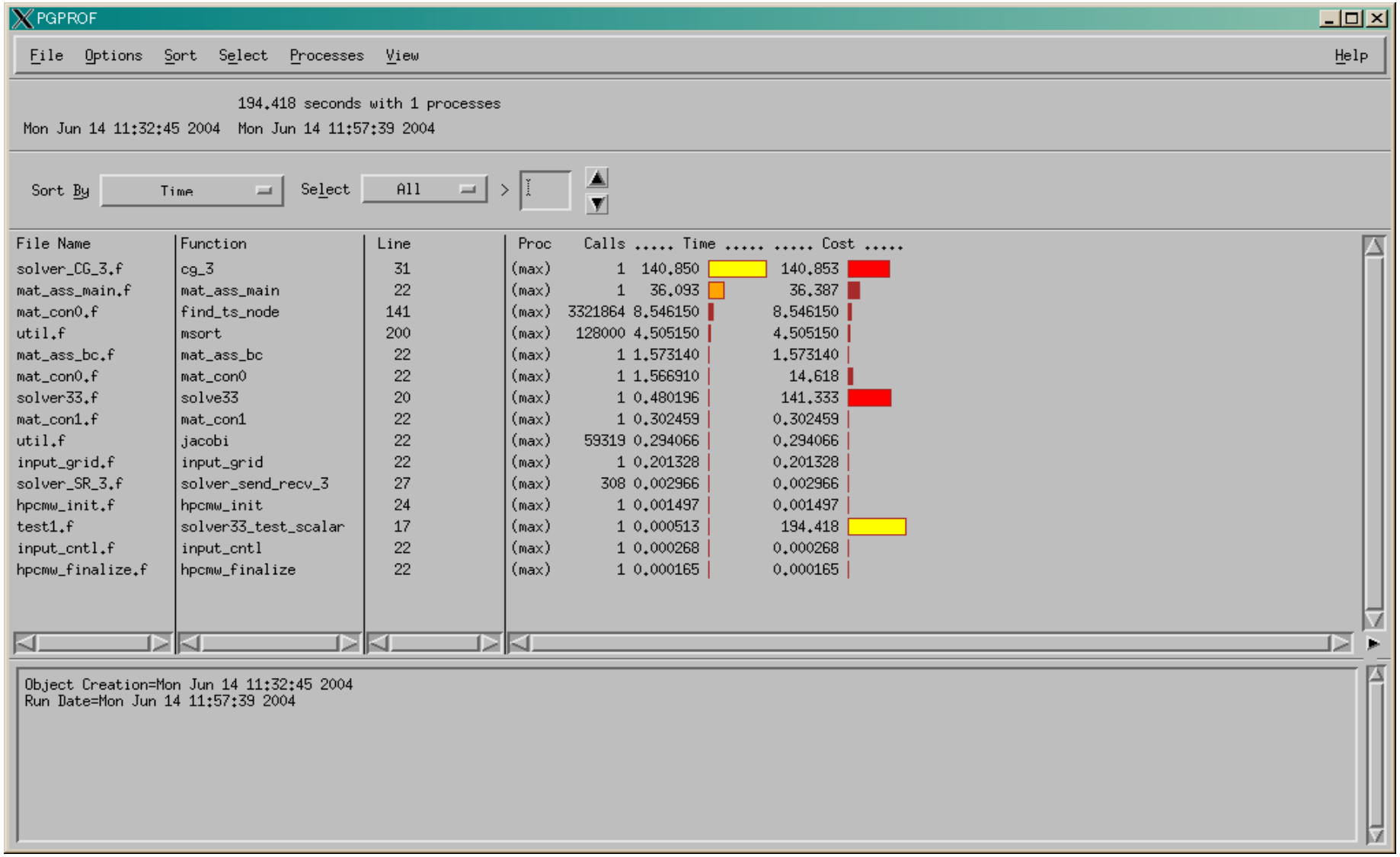
pgprofの利用法(4/4)

```
pgprof> lines cg
```

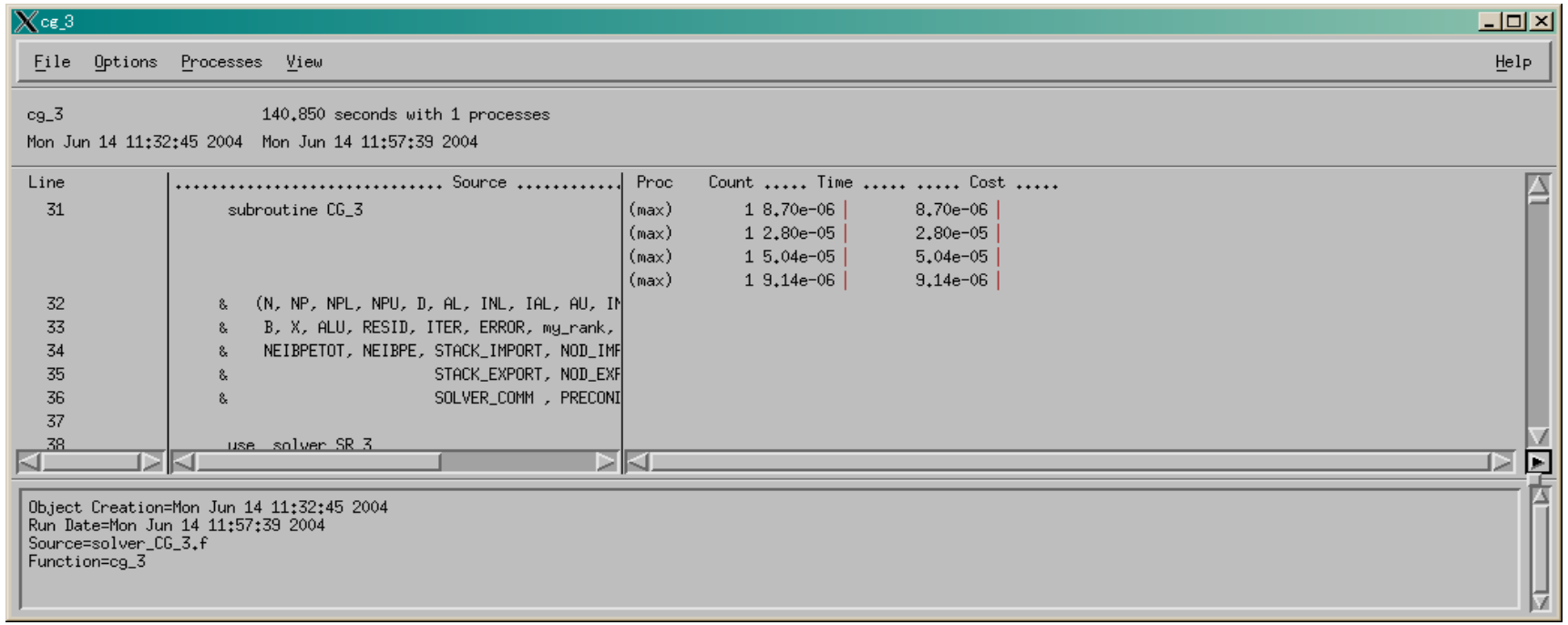
lines <関数名>:関数の処理時間を行単位で表示

```
pgprof> quit
```

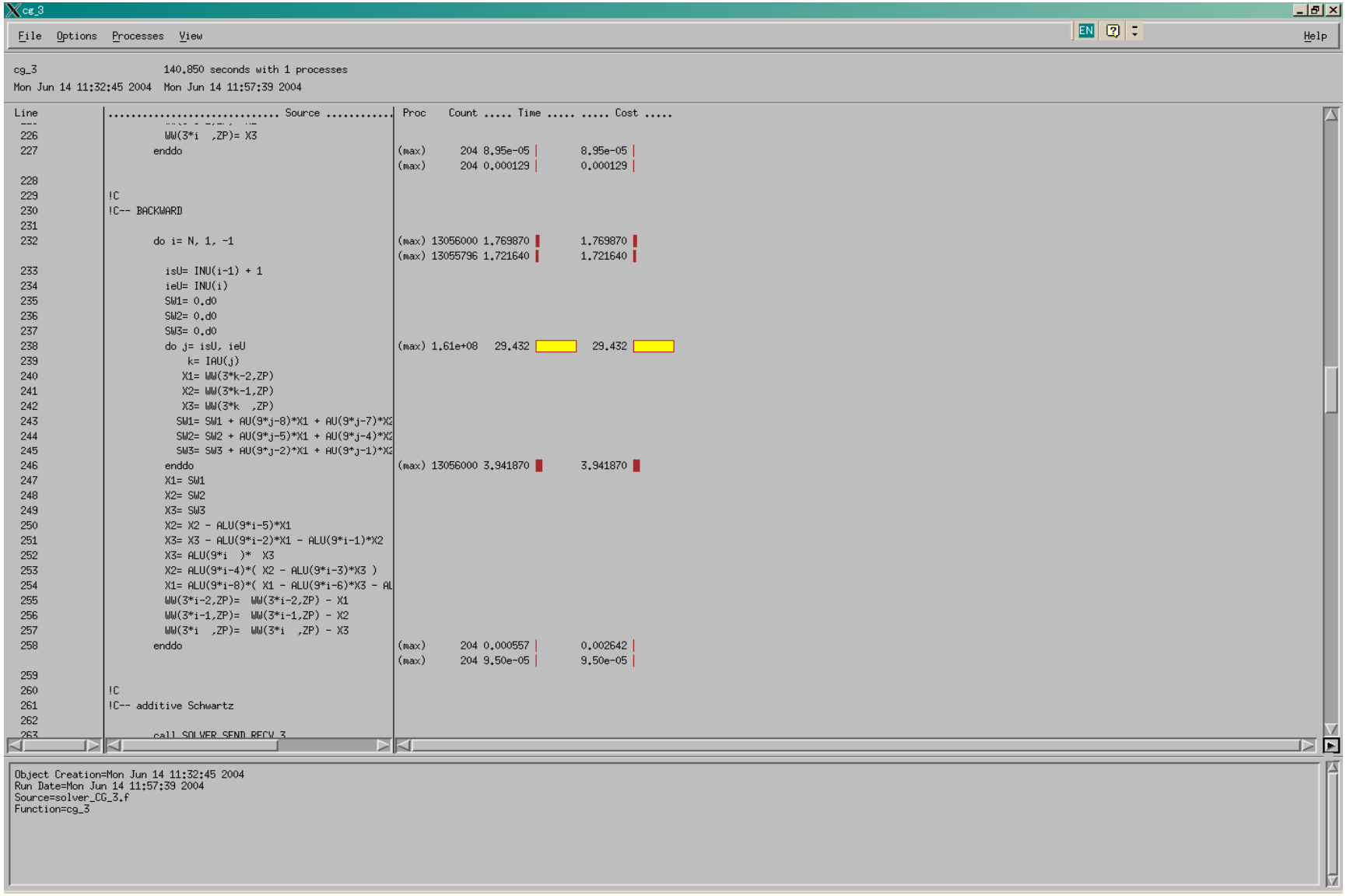
pgprof on X-window



pgprof on X-window

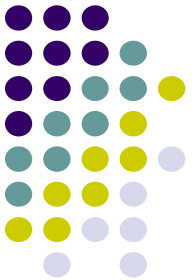


pgprof on X-window



- チューニングとは
- ベクトルプロセッサとスカラープロセッサ
- 具体的チューニング例
  - スカラープロセッサー
  - ベクトルプロセッサー

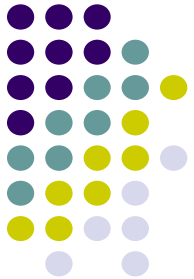
プロセッサの動向：スカラーとベクトル



- スカラープロセッサ
  - クロック数とメモリバンド幅のギャップ
    - 改善はされつつある
  - 低い対ピーク性能比
    - 例：IBM Power-3，Power-4，FEM型アプリケーション → 5-8 %
- ベクトルプロセッサ
  - 高い対ピーク性能比
    - 例：地球シミュレータ，FEM型アプリケーション → >35 %
  - そのためには・・・
    - ベクトルプロセッサ用チューニング
    - 充分長いベクトル長（問題サイズ）
  - 比較的単純な問題に適している

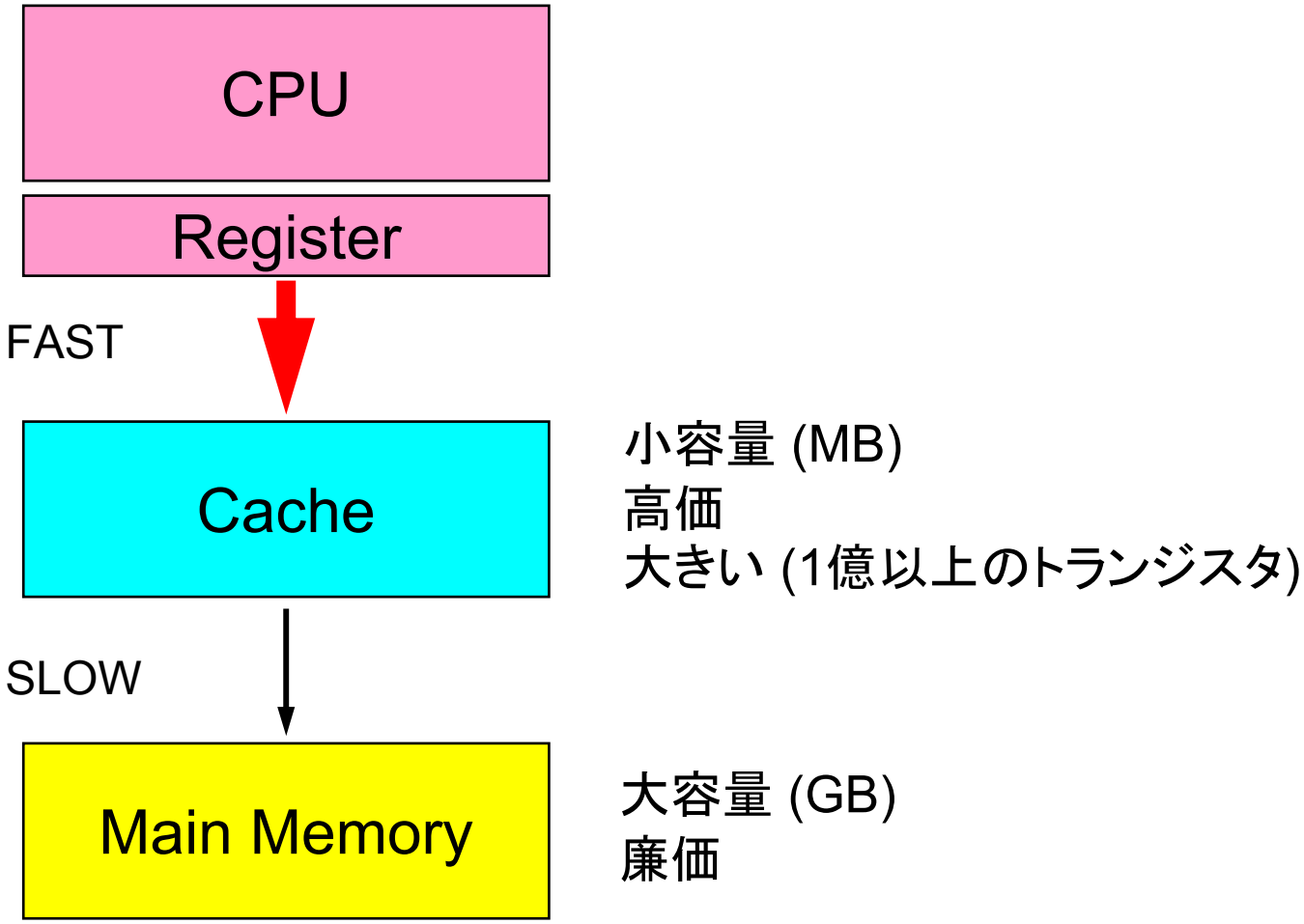
各種ハードウェアの比較

	Hitachi SR8000/MPP	IBM SP-3 NERSC/LBNL	Earth Simulator
PE#/node	8	16	8
Clock rate	450 MHz	375 MHz	500 MHz
Peak performance/PE	1.80 GFLOPS	1.50 GFLOPS	8.00 GFLOPS
Memory/node	16 GB	16GB ~ 64 GB	16 GB
Memory-PE Bandwidth	32 GB/sec/node	16 GB/sec/node	256 GB/sec/node



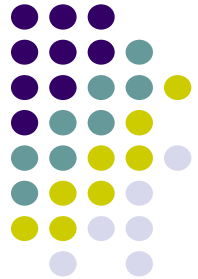
スカラープロセッサ

CPU−キャッシュ−メモリの階層構造

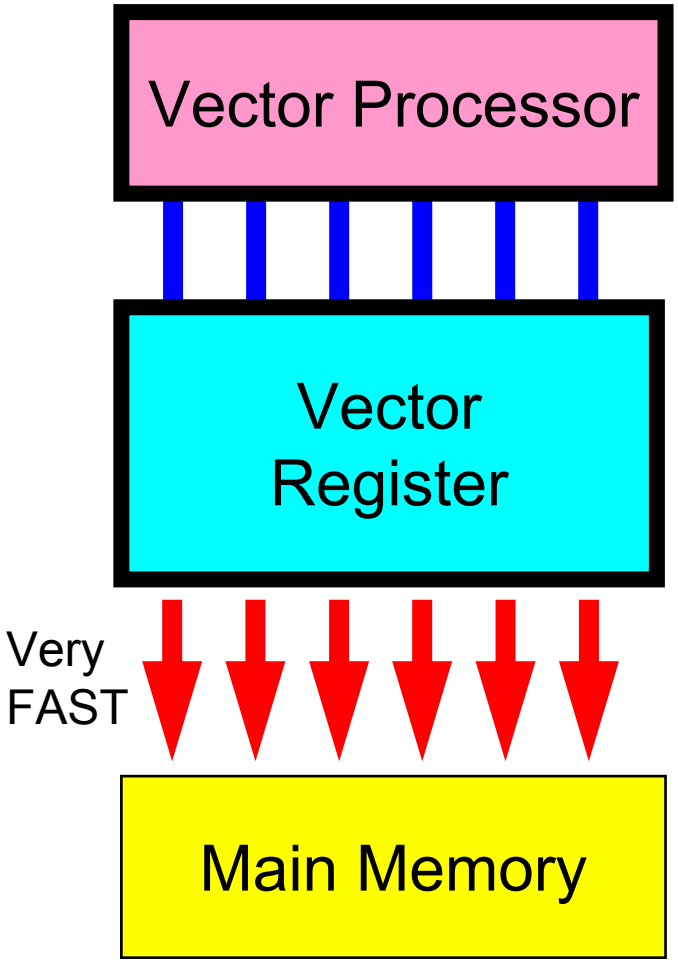


小容量 (MB)
高価
大きい (1億以上のトランジスタ)

大容量 (GB)
廉価



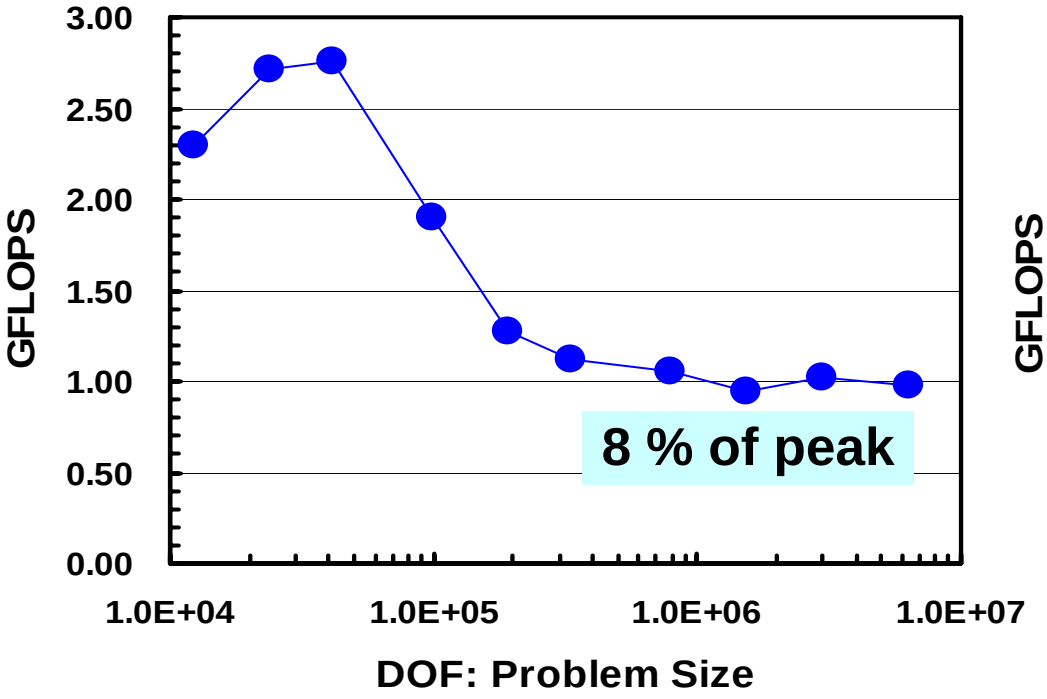
ベクトルプロセッサ
ベクトルレジスタと高速メモリ

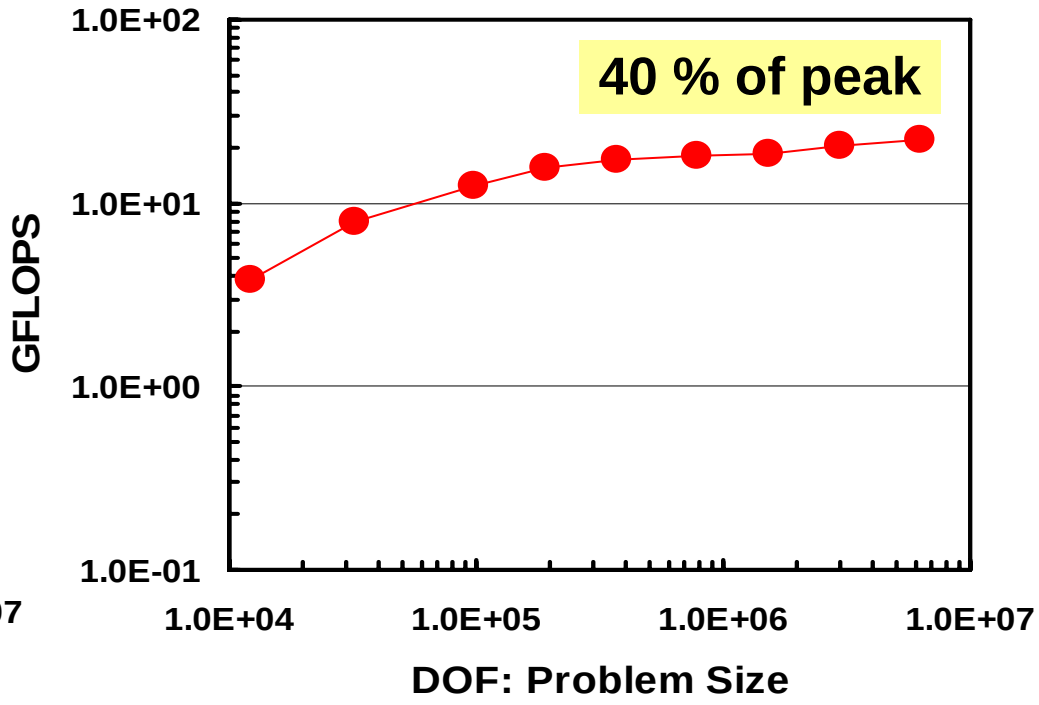


- 単純構造のDOループの並列処理
- 単純，大規模な演算に適している

```
do i= 1, N
  A(i)= B(i) + C(i)
enddo
```

典型的な挙動



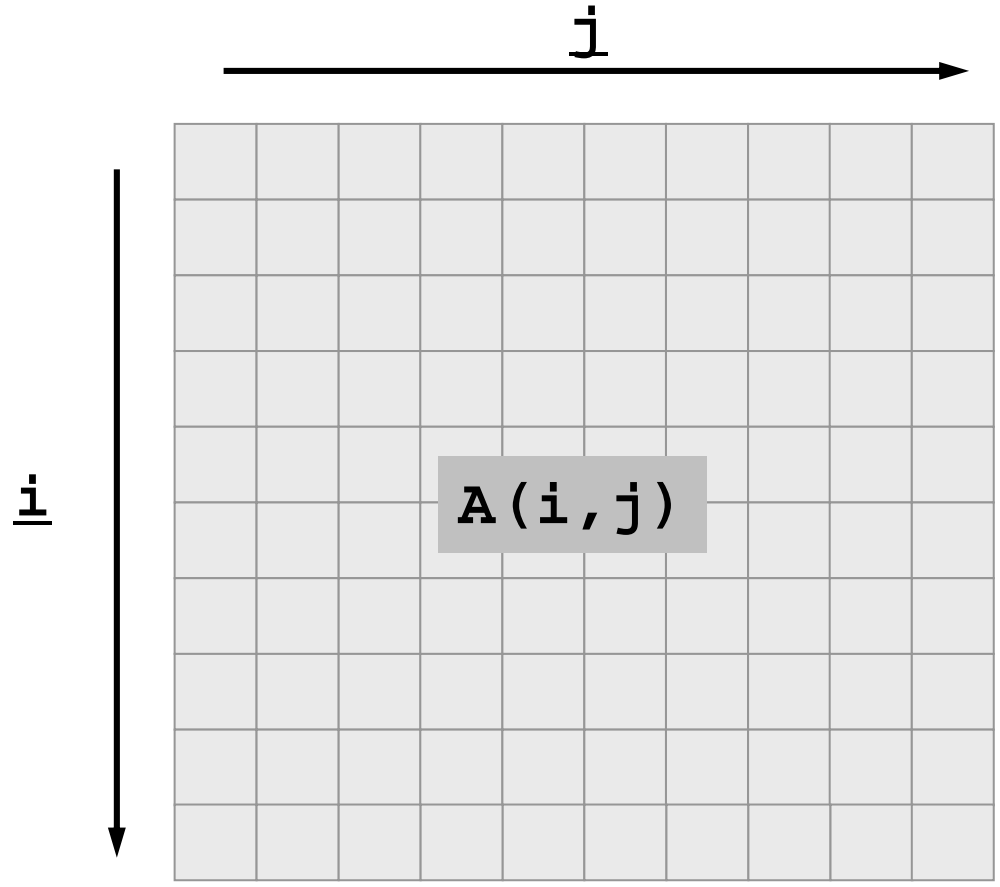


IBM-SP3:
問題サイズが小さい場合はキャッシュの影響のため性能が良い

Earth Simulator:
大規模な問題ほどベクトル長が長くなり，性能が高い

プロセッサに応じたチューニング

- メモリ参照の最適化･･･に尽きる



プロセッサに応じたチューニング（続き）

- ベクトルプロセッサ
  - ループ長を大きくとる。
- スカラープロセッサ
  - キャッシュを有効利用，細切れにしたデータを扱う。
  - PCクラスタなどでは，キャッシュサイズを大きくできない一方で，メモリレイテンシ（立ち上がりオーバーヘッド）の減少，メモリバンド幅の増加の傾向。
- 共通事項
  - メモリアクセスの連続性
  - 局所性
  - 計算順序の変更によって計算結果が変わる可能性について注意すること

- チューニングとは
- ベクトルプロセッサとスカラープロセッサ
- 具体的チューニング例
  - スカラープロセッサー
  - ベクトルプロセッサー

スカラープロセッサの代表的なチューニング

- ループアンローリング
  - ループのオーバーヘッドの削減
  - ロード・ストアの削減
- ブロック化
  - キャッシュミスの削減

ループアンローリング
ループオーバーヘッドの削減

<$S4>/t1.f

- ループ処理に対する演算の割合が増す。

```
NN= 100000000

do i= 1, NN
  X(i)= 1.d0
enddo

do i= 1, NN-1, 2
  X(i  )= 1.d0
  X(i+1)= 1.d0
enddo

do i= 1, NN-3, 4
  X(i  )= 1.d0
  X(i+1)= 1.d0
  X(i+2)= 1.d0
  X(i+3)= 1.d0
enddo

do i= 1, NN-7, 8
  X(i  )= 1.d0
  X(i+1)= 1.d0
  X(i+2)= 1.d0
  X(i+3)= 1.d0
  X(i+4)= 1.d0
  X(i+5)= 1.d0
  X(i+6)= 1.d0
  X(i+7)= 1.d0
enddo
```

```
$> cd <$S4>
$> mpif90 -O3 t1.f

$> <modify “go.sh”>

$> qsub go.sh

1-lev.     8.203125E-01
2-lev.     5.234375E-01
4-lev.     5.078125E-01
8-lev.     5.273438E-01
FORTRAN STOP
```

ループアンローリング
ロード・ストアの削減（1/4）

- ループ処理に対する演算の割合が増す。

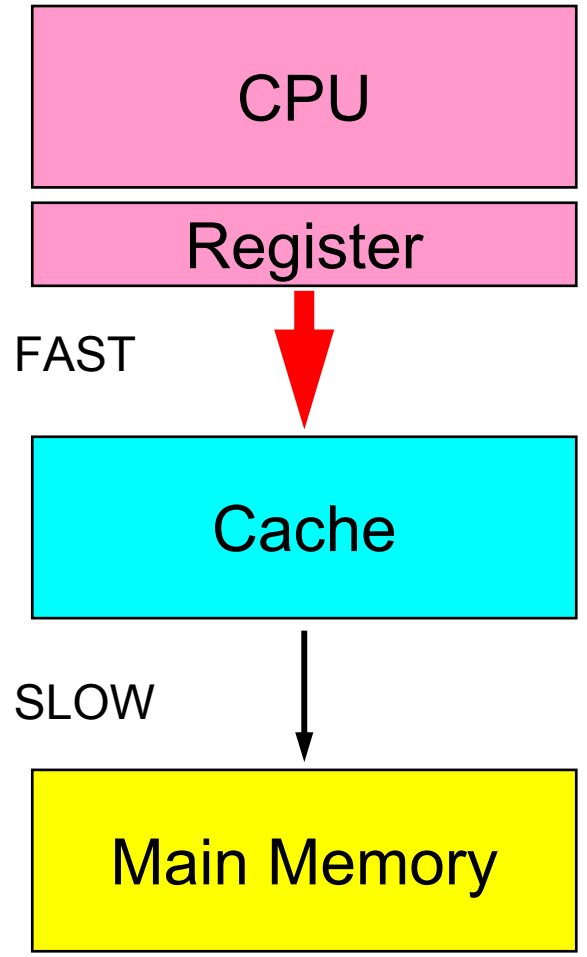
<$S4>/t2.f

```
N= 10000

do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
  enddo
enddo

do j= 1, N-1, 2
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
  enddo
enddo

do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
    A(i)= A(i) + B(i)*C(i,j+2)
    A(i)= A(i) + B(i)*C(i,j+3)
  enddo
enddo
```

```
$> cd <$S4>
$> mpif90 -O3 t2.f

$> <modify “go.sh”>

$> qsub go.sh

    4.023438E-01
    3.085938E-01
    2.617188E-01
FORTRAN STOP
```

ループアンローリング
ロード・ストアの削減（2/4）

- ロード：メモリ⇒キャッシュ⇒レジスタ
- ストア：ロードの逆
- ロード・ストアが少ないほど効率良い



ループアンローリング
ロード・ストアの削減（3/4）

```
do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
   ストア  ロード   ロード ロード
  enddo
enddo
```

- A(i,j)に対して，各ループでロード・ストアが発生する。

ループアンローリング

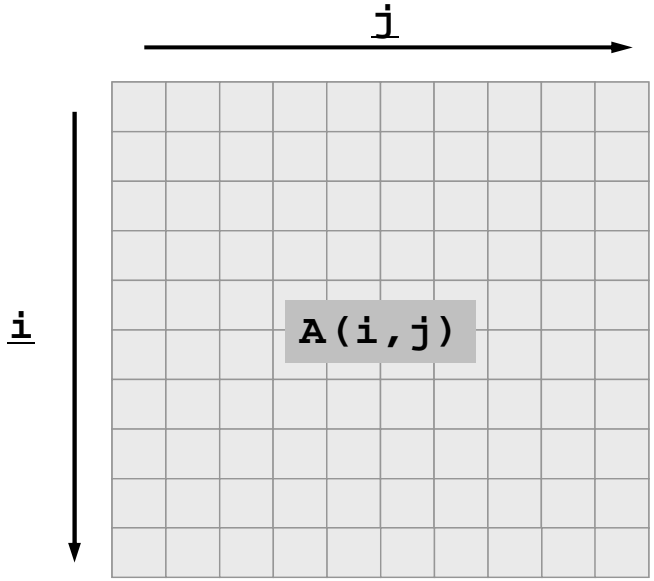
ロード・ストアの削減（4/4）

```
do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
           ロード  ロード ロード
    A(i)= A(i) + B(i)*C(i,j+1)
    A(i)= A(i) + B(i)*C(i,j+2)
    A(i)= A(i) + B(i)*C(i,j+3)
    ストア
  enddo
enddo
```

- 同一ループ内で複数回表れている場合には，最初にロード，最後にストアが実施され，その間レジスターに保持される。
- 計算順序に注意。

ループ入替によるメモリ参照最適化(1/2)

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```



- FORTRANでは，A(i,j)のアドレスは，A(1,1), A(2,1), A(3,1),…, A(N,1), A(1,2), A(2,2),…, A(1,N), A(2,N),…, A(N,N)のように並んでいる
  - Cは逆
- この順番にアクセスしないと効率悪い(ベクトル，スカラーに共通)。

ループ入替によるメモリ参照最適化(2/2)

<$S4>/2d-1.f

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```

```
$> cd <$S4>
$> mpif90 -O3 2d-1.f
$> qsub go.sh
### N ###      512
WORSE      3.125000E-02
BETTER     3.906250E-03
### N ###     1024
WORSE      2.343750E-01
BETTER     7.812500E-03
### N ###     1536
WORSE      3.476563E-01
BETTER     1.562500E-02
### N ###     2048
WORSE      9.296875E-01
BETTER     3.125000E-02
### N ###     2560
WORSE      9.687500E-01
BETTER     4.687500E-02
### N ###     3072
WORSE      2.152344E+00
BETTER     7.421875E-02
### N ###     3584
WORSE      1.921875E+00
BETTER     9.765625E-02
### N ###     4096
WORSE      3.804688E+00
BETTER     1.250000E-01
FORTRAN STOP
```

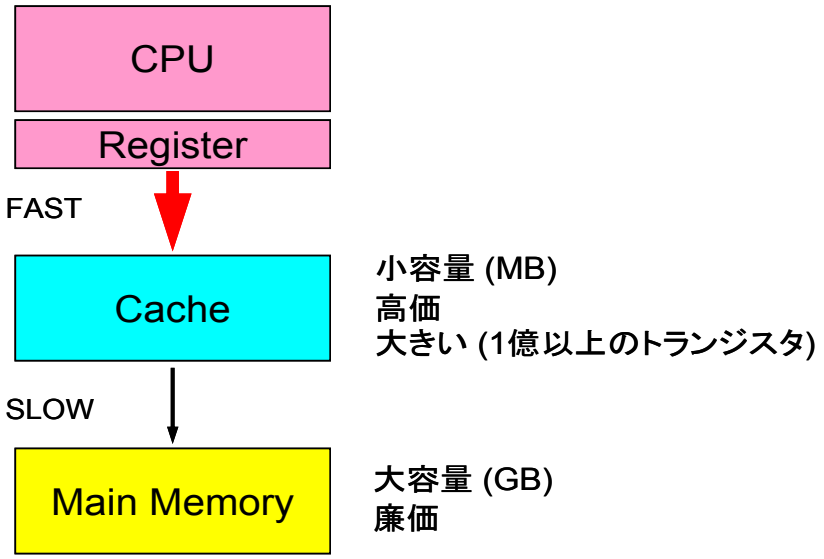
ブロック化によるキャッシュミス削減（1/7）

```
do i= 1, NN
  do j= 1, NN
    A(j,i)= A(j,i) + B(i,j)
  enddo
enddo
```

- 計算の処理の都合でこのような順番で計算せざるを得ない場合。

キャッシュの有効利用

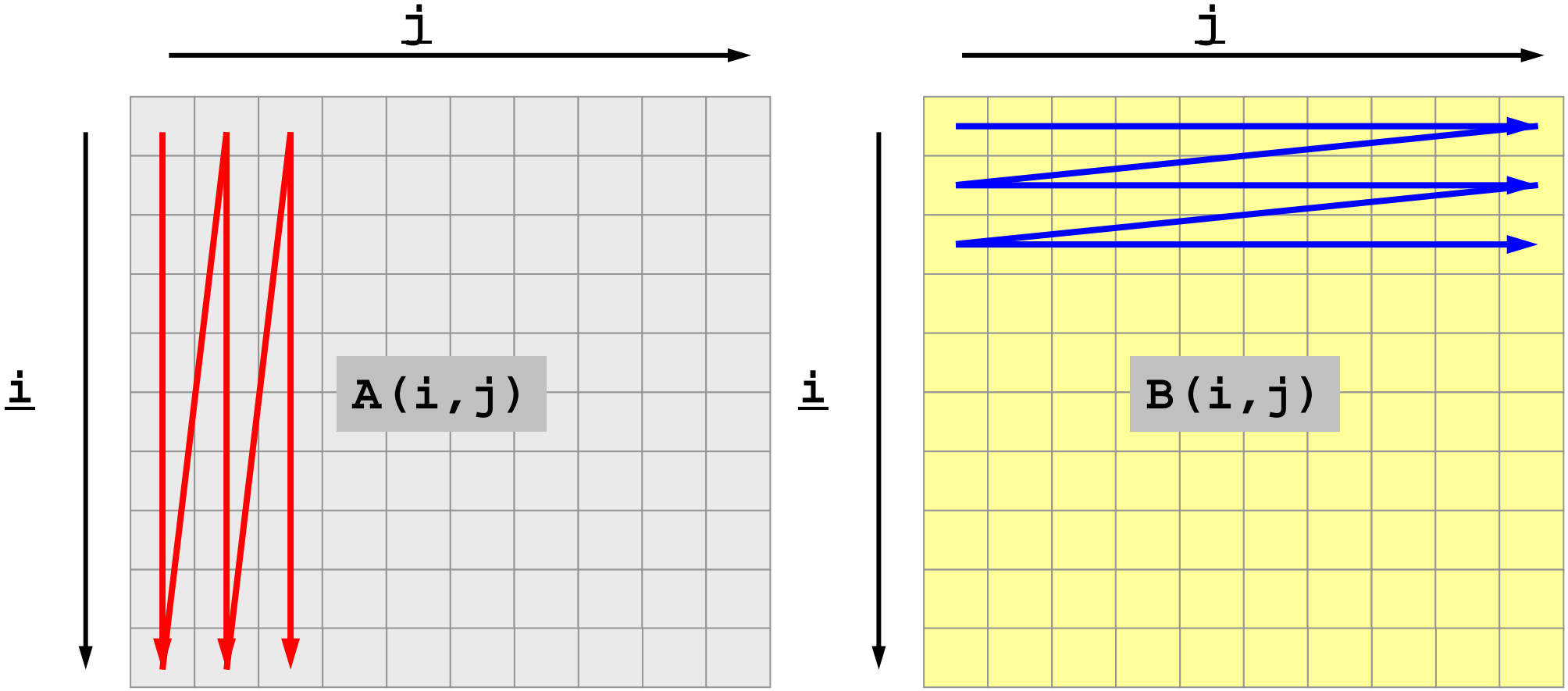
- キャッシュ
  - cenjuでは1PEあたり1MB
  - 実際は64byte～128byteの細かいキャッシュラインに分かれている。
  - キャッシュライン単位でメモリへのリクエストが行われる。



- TLB（Translate Lookaside Buffer）
  - アドレス変換バッファ
    - 仮想アドレスから実アドレスへの変換機能
  - TLB用のキャッシュ
    - 通常128×8 kbyte程度：リンク時に可変
  - 「キャッシュ」が有効に使われていればTLBミスも起こりにくい

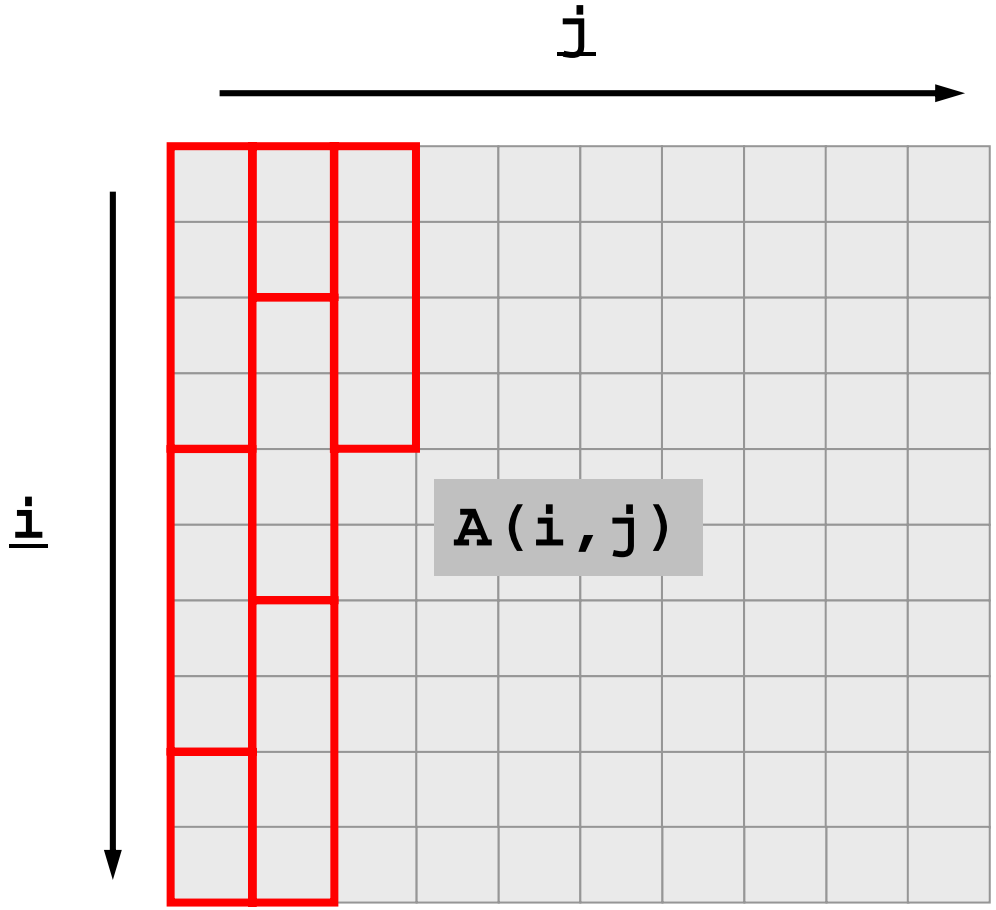
ブロック化によるキャッシュミス削減（2/7）

- A, Bでメモリアクセスパターンが相反
  - 特にBは効率が悪い



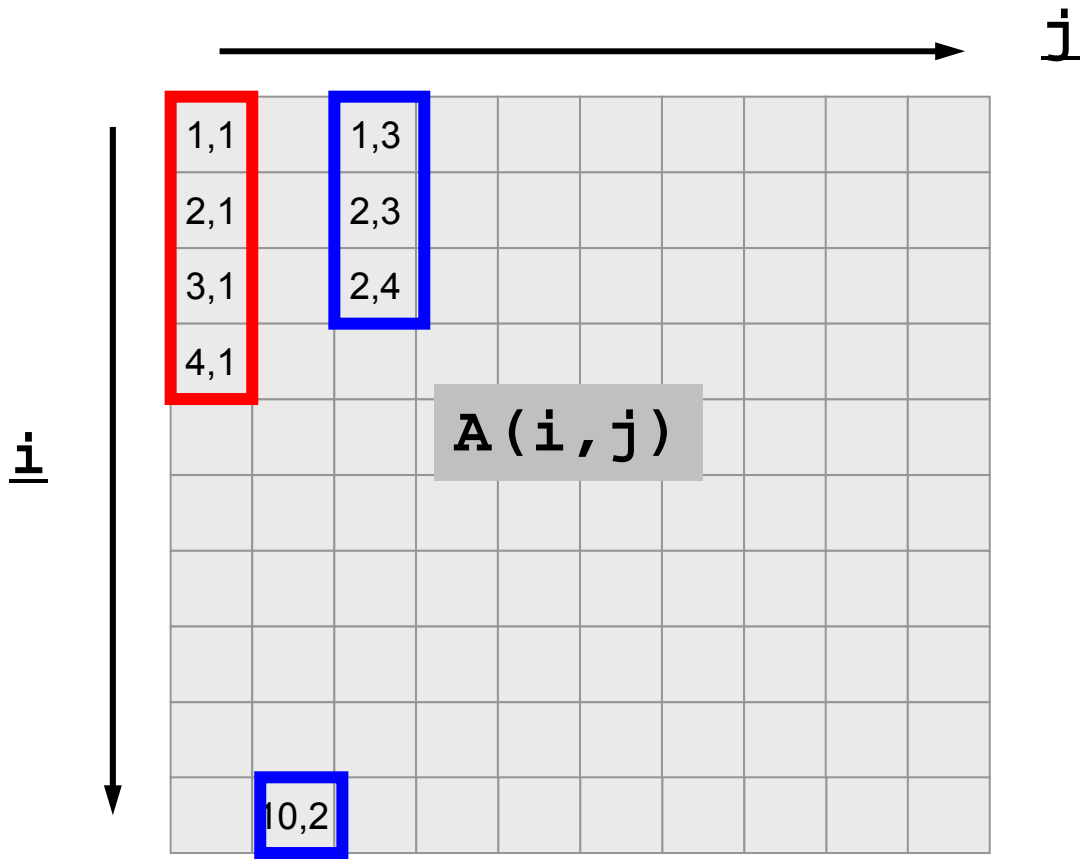
ブロック化によるキャッシュミス削減（3/7）

- 例えば，キャッシュラインサイズを4ワードとすると配列の値は以下のようにキャッシュに転送される。



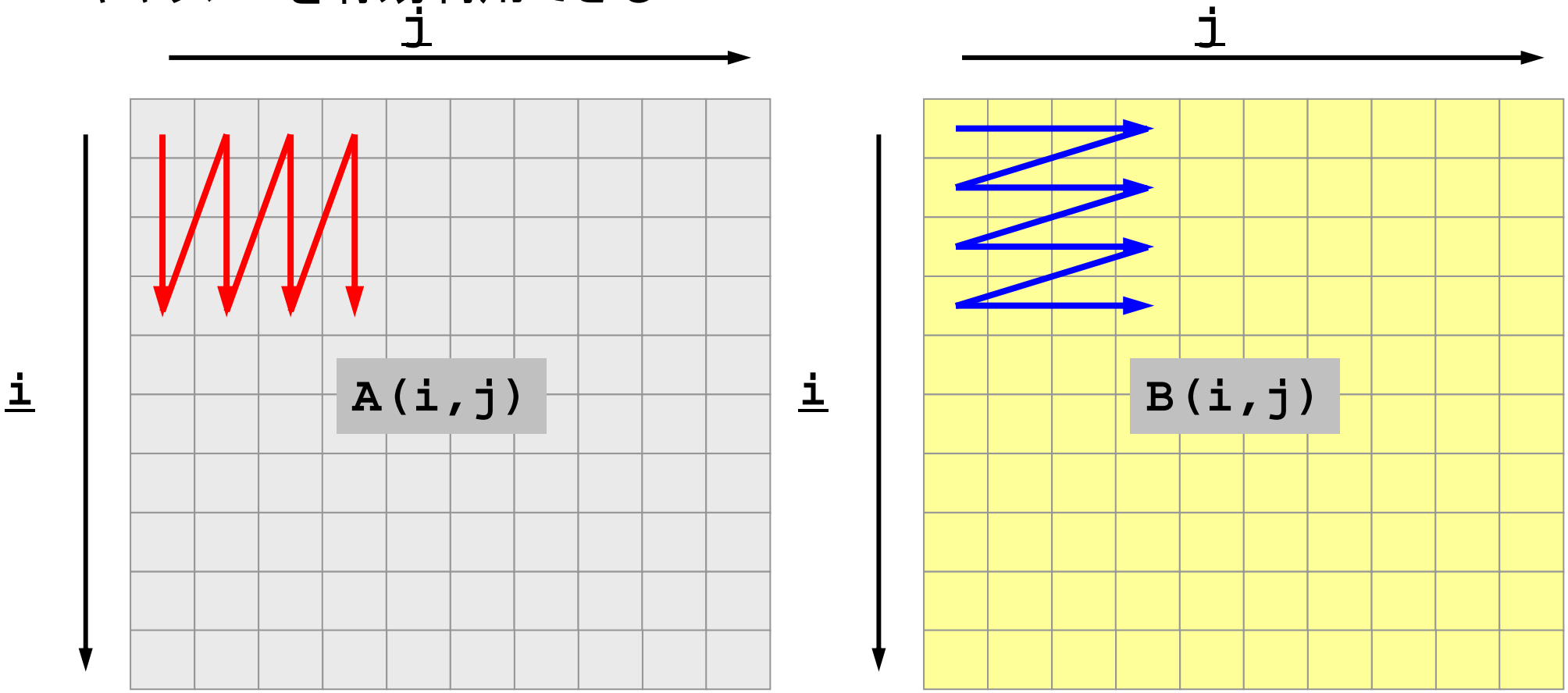
ブロック化によるキャッシュミス削減（4/7）

- したがって，A(1,1)をアクセスしたら，A(1,1), A(2,1), A(3,1), A(4,1)が，A(10,2)をアクセスしたらA(10,2), A(1,3), A(2,3), A(3,3)がそれぞれキャッシュ上にあるということになる。



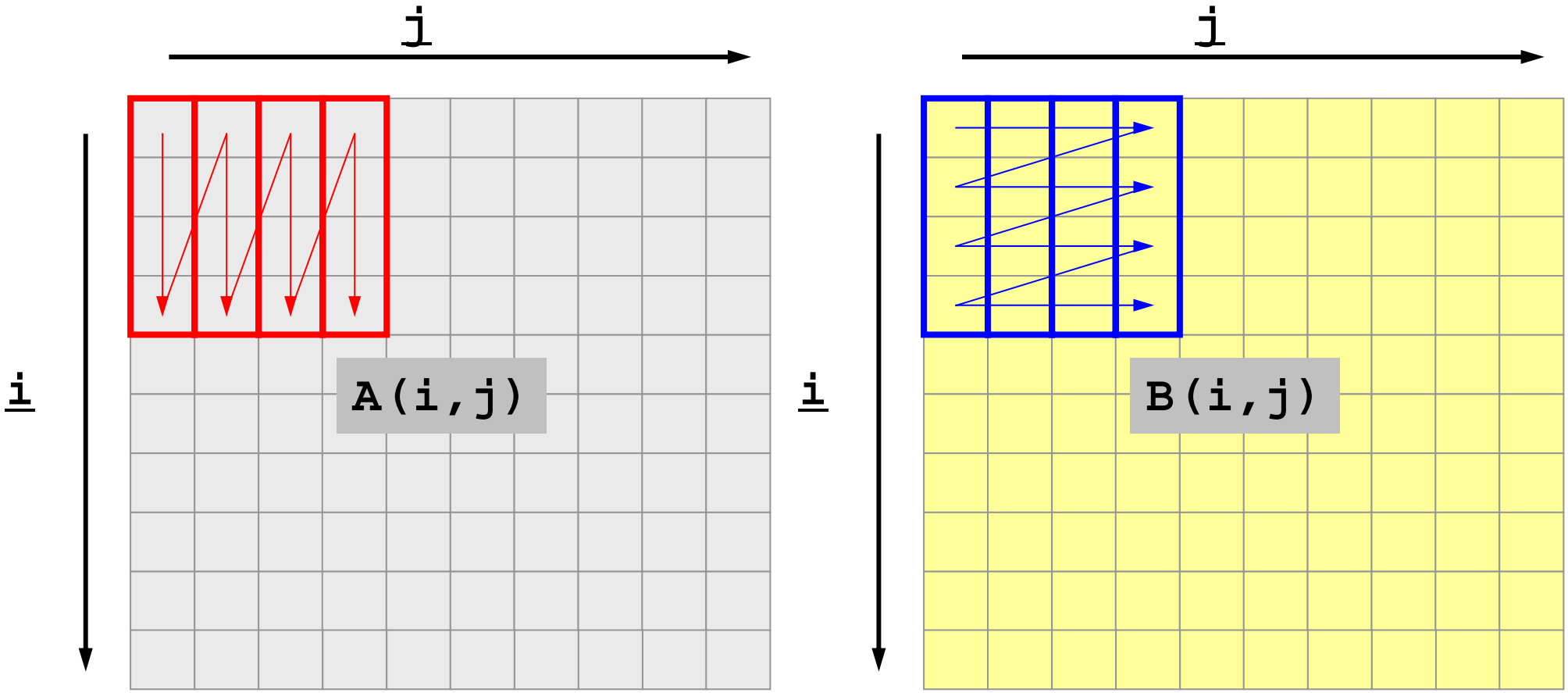
ブロック化によるキャッシュミス削減（5/7）

- したがって，以下のようなブロック型のパターンでアクセスすれば，キャッシュを有効利用できる



ブロック化によるキャッシュミス削減（6/7）

- □，□で囲んだ部分はキャッシュに載っている。



ブロック化によるキャッシュミス削減（7/7）

- ２×２ブロック

<$S4>/2d-2.f

```
do i= 1, NN
  do j= 1, NN
    A(j,i)= A(j,i) + B(i,j)
  enddo
enddo


do i= 1, NN-1, 2
  do j= 1, NN-1, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo

do i= 1, NN-1, 2
  do j= 1, NN/2, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo

do i= 1, NN-1, 2
  do j= NN/2+1, NN-1, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo
```

ループ分割もTLBミス，
キャッシュミス削減に
有効と言われているが・・・

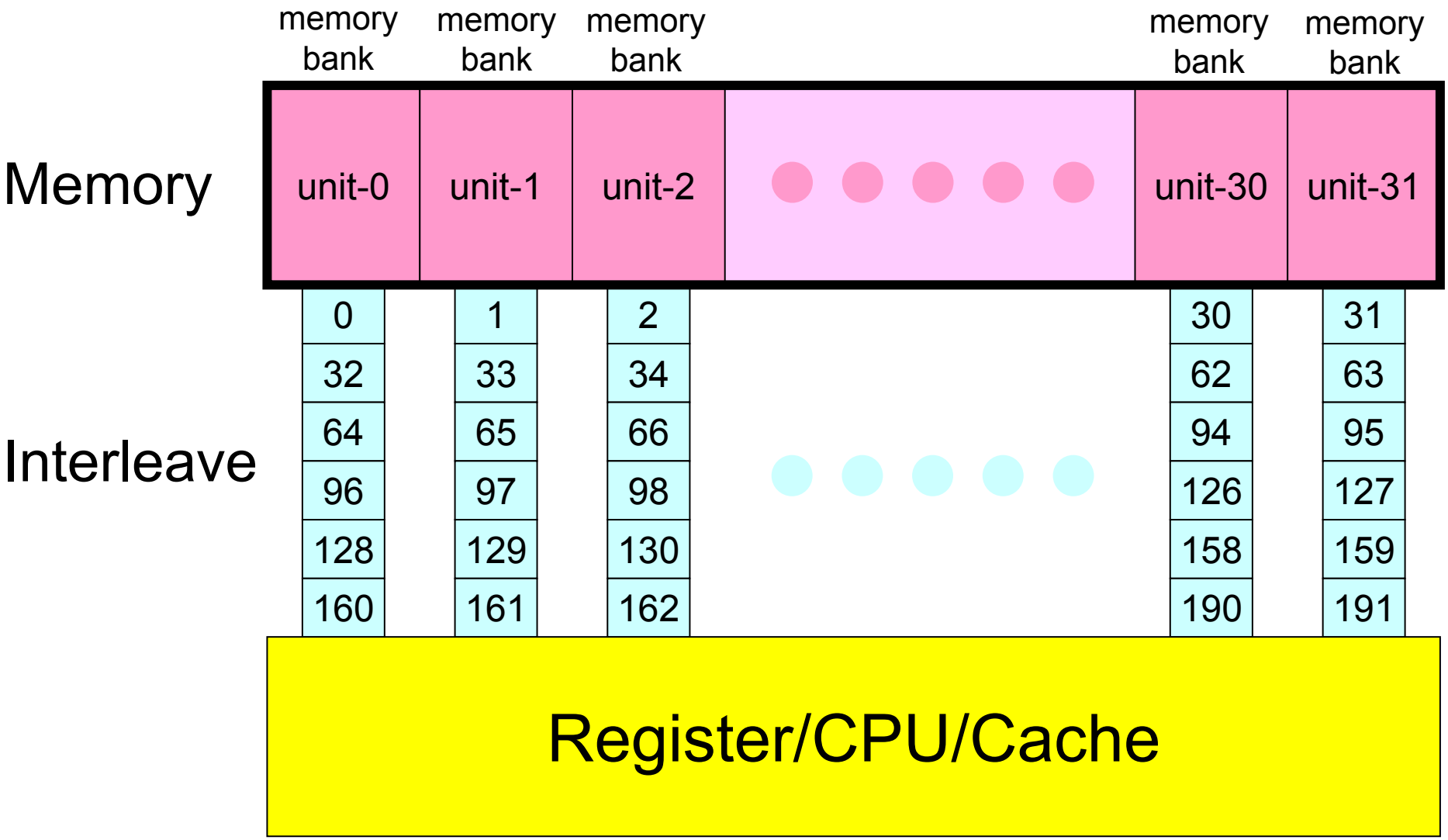
```
$> cd <$S4>
$> mpif90 -O3 2d-2.f
$> qsub go.sh

### N ###     1024
BASIC      2.734375E-02
2x2        1.562500E-02
2x2-b      1.562500E-02
### N ###     1536
BASIC      6.250000E-02
2x2        3.515625E-02
2x2-b      3.515625E-02

...

### N ###     3072
BASIC      2.578125E-01
2x2        1.484375E-01
2x2-b      1.484375E-01
### N ###     3584
BASIC      3.710938E-01
2x2        2.031250E-01
2x2-b      2.031250E-01
### N ###     4096
BASIC      8.437500E-01
2x2        4.375000E-01
2x2-b      4.335938E-01
```

Memory Interleaving と Bank Conflict

- メモリインターリーブ（memory interleaving）
  - メモリのデータ転送を高速化する技術の一つ。複数のメモリバンクに同時並行で読み書きを行なうことにより高速化を行なう手法。
- メモリーバンク，バンク（memory bank）
  - メモリコントローラがメモリを管理するときの単位となる，一定の容量を持ったメモリの集合。
    - 通常は 2^n個の独立したモジュール
  - 同一のメモリバンクに対しては，同時には1つの読み出しまたは書き込みしかできないため，連続して同じグループのデータをアクセスすると性能が低下。
  - 例えば，一つの配列を32要素飛び(又は32の倍数要素飛び)にアクセスすると，バンク競合（コンフリクト）が発生する。これを防ぐためには，配列宣言を奇数になるように変更するか，ループを入れ換えて，配列のアクセスが連続するように変更する。

Bank Conflict

32とびにアクセスすると1つのバンクしか使えない



Bank Conflictの回避

×

```
REAL*8 A(32,10000)

k= N
do i= 1, 10000
   A(k,i)= 0.d0
enddo
```

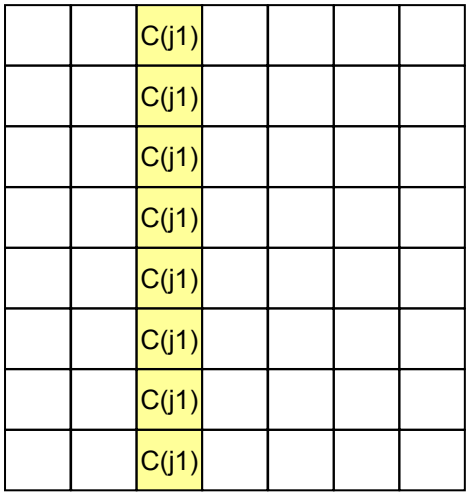
○

```
REAL*8 A(33,10000)

k= N
do i= 1, 10000
   A(k,i)= 0.d0
enddo
```

- 「地球シミュレータ」では，内側ループで，メモリ番地が飛び飛びになるようなアクセスパターンでは，概して「Bank Conflict」の値が大きくなる。
  - 「Bank Conflict」が解消されて遅くなる場合もある。

Bank Conflictの例（追加）

```
   do k= 1, M
!CDIR NODEP
     do i= 1, N
       j= index(i,k)
       A(i)= A(i) + B(i,k)*C(j)
     enddo
   enddo
```



- このような場合，ベクトルレジスタのサイズ（ESでは256）の範囲で同じC(j)を頻繁にアクセスする場合もバンクコンフリクトが生じる可能性がある。

Memory Interleaving と Bank Conflict

- 例えば，「2d-2.f」では，「allocate (A(NN,NN), B(NN,NN))」としているが，NNは512の倍数なので，この影響を受けやすいと考えられる。

allocate (A(NN+k,NN+k), B(NN+k,NN+k)) とした場合

k=0

```
### N ###     3584
BASIC      5.273438E-01
2x2        2.929688E-01
### N ###     4096
BASIC      1.023438E+00
2x2        5.429688E-01
```

k=1

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.757813E-01
### N ###     4096
BASIC      4.218750E-01
2x2        3.203125E-01
```

k=2

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.718750E-01
### N ###     4096
BASIC      4.179688E-01
2x2        2.382813E-01
```

k=3

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.679688E-01
### N ###     4096
BASIC      4.101563E-01
2x2        2.343750E-01
```

k=4

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.679688E-01
### N ###     4096
BASIC      4.179688E-01
2x2        2.226563E-01
```

k=8

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.718750E-01
### N ###     4096
BASIC      4.179688E-01
2x2        2.226563E-01
```

k=16

```
### N ###     3584
BASIC      3.359375E-01
2x2        1.835938E-01
### N ###     4096
BASIC      4.375000E-01
2x2        2.421875E-01
```

k=32

```
### N ###     3584
BASIC      3.359375E-01
2x2       1.835938E-01
### N ###     4096
BASIC      4.492188E-01
2x2       2.460938E-01
```

k=64

```
### N ###     3584
BASIC      4.218750E-01
2x2        2.421875E-01
### N ###     4096
BASIC      6.679688E-01
2x2        3.750000E-01
```

k=63

```
### N ###     3584
BASIC      3.164063E-01
2x2        1.679688E-01
### N ###     4096
BASIC      4.101563E-01
2x2        2.226563E-01
```

k=65

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.679688E-01
### N ###     4096
BASIC      4.101563E-01
2x2        2.265625E-01
```

k=128

```
### N ###     3584
BASIC      5.585938E-01
2x2        3.085938E-01
### N ###     4096
BASIC      7.421875E-01
2x2        4.140625E-01
```

k=127

```
### N ###     3584
BASIC      3.164063E-01
2x2        1.679688E-01
### N ###     4096
BASIC      4.101563E-01
2x2        2.265625E-01
```

k=129

```
### N ###     3584
BASIC      3.125000E-01
2x2        1.718750E-01
### N ###     4096
BASIC      4.140625E-01
2x2        2.304688E-01
```

- チューニングとは
- ベクトルプロセッサとスカラープロセッサ
- 具体的チューニング例
  - スカラープロセッサー
  - ベクトルプロセッサー

ベクトルプロセッサの代表的なチューニング

- 連続メモリアクセス。
- 最内ループ長を大きくとる。
  - キャッシュの有効利用（細切れ）とは逆の発想
- そのためのリオーダリング（reordering）。
  - 計算順序によって答えが変わることがあるので注意。

- スカラープロセッサの場合，チューニングの効果は高々数倍程度，ベクトルプロセッサでは数百倍にも及ぶことがある。

Vectorization: Primary (1)

Original Code

```
      NN= 10 000 000
!CDIR NODEP
      do i= 1, NN
        AA(i)= dfloat(i)
        BB(i)= dfloat(i)
      enddo

      do i= 1, NN
        do k= 1, 4
!CDIR NODEP
        do j= 1, 4
          CC(j,k)= dfloat ((k-1)*NN + j)
        enddo
        enddo

        do k= 1, 4
!CDIR NODEP
        do j= 1, 4
          AA(i)= AA(i) + CC(j,k)
          BB(i)= BB(i) + CC(j,k)
        enddo
        enddo
      enddo

      stop
      endcc
```

EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO
3.520(100.0)	3520.189	389.3	90.9	2.92

Vectorization: Primary (2)

Improved Code

```
!CDIR NODEP
      do i= 1, NN
          CC(1,1)= dfloat ((1-1)*NN + 1)
          CC(1,2)= dfloat ((2-1)*NN + 1)
          CC(1,3)= dfloat ((3-1)*NN + 1)
          CC(1,4)= dfloat ((4-1)*NN + 1)
          CC(2,1)= dfloat ((1-1)*NN + 2)
          CC(2,2)= dfloat ((2-1)*NN + 2)
          CC(2,3)= dfloat ((3-1)*NN + 2)
          CC(2,4)= dfloat ((4-1)*NN + 2)
          CC(3,1)= dfloat ((1-1)*NN + 3)
          CC(3,2)= dfloat ((2-1)*NN + 3)
          CC(3,3)= dfloat ((3-1)*NN + 3)
          CC(3,4)= dfloat ((4-1)*NN + 3)
          CC(4,1)= dfloat ((1-1)*NN + 4)
          CC(4,2)= dfloat ((2-1)*NN + 4)
          CC(4,3)= dfloat ((3-1)*NN + 4)
          CC(4,4)= dfloat ((4-1)*NN + 4)

          AA(i)= AA(i) + CC(1,1) + CC(2,1) + CC(3,1) + CC(4,1) &
     &                 + CC(1,2) + CC(2,2) + CC(3,2) + CC(4,2) &
     &                 + CC(1,3) + CC(2,3) + CC(3,3) + CC(4,3) &
     &                 + CC(1,4) + CC(2,4) + CC(3,4) + CC(4,4)

          BB(i)= BB(i) + CC(1,1) + CC(2,1) + CC(3,1) + CC(4,1) &
     &                 + CC(1,2) + CC(2,2) + CC(3,2) + CC(4,2) &
     &                 + CC(1,3) + CC(2,3) + CC(3,3) + CC(4,3) &
     &                 + CC(1,4) + CC(2,4) + CC(3,4) + CC(4,4)
      enddo
```

```
      do i= 1, NN
        do k= 1, 4
!CDIR NODEP
        do j= 1, 4
          CC(j,k)= dfloat ((k-1)*NN + j)
        enddo
        enddo

        do k= 1, 4
!CDIR NODEP
        do j= 1, 4
          AA(i)= AA(i) + CC(j,k)
          BB(i)= BB(i) + CC(j,k)
        enddo
        enddo
      enddo
```

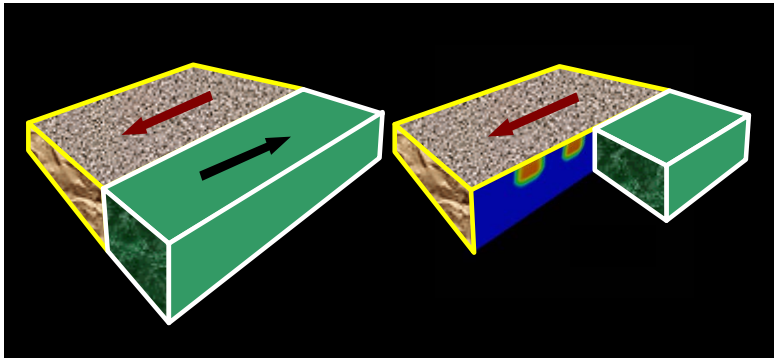
EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO
0.015(100.0)	15.022	6719.6	1331.4	99.07

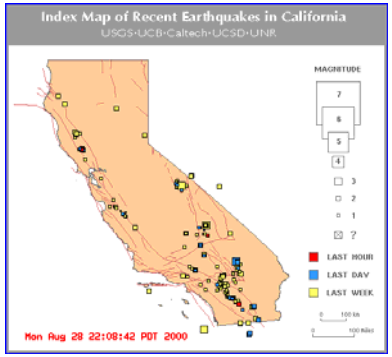
ケーススタディによる事例

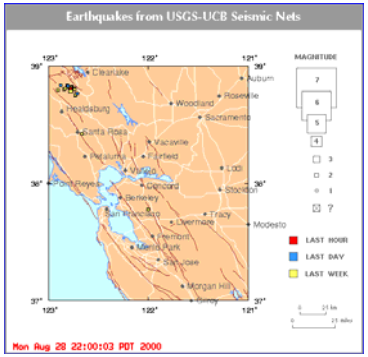
- 中島研吾，橋本千尋，松浦充宏（2002），『地球シミュレータ』による横ずれ断層の大規模シミュレーション，日本地震学会2002年度秋季大会

- 境界積分法（境界要素法）
  - 横ずれ断層における応力蓄積

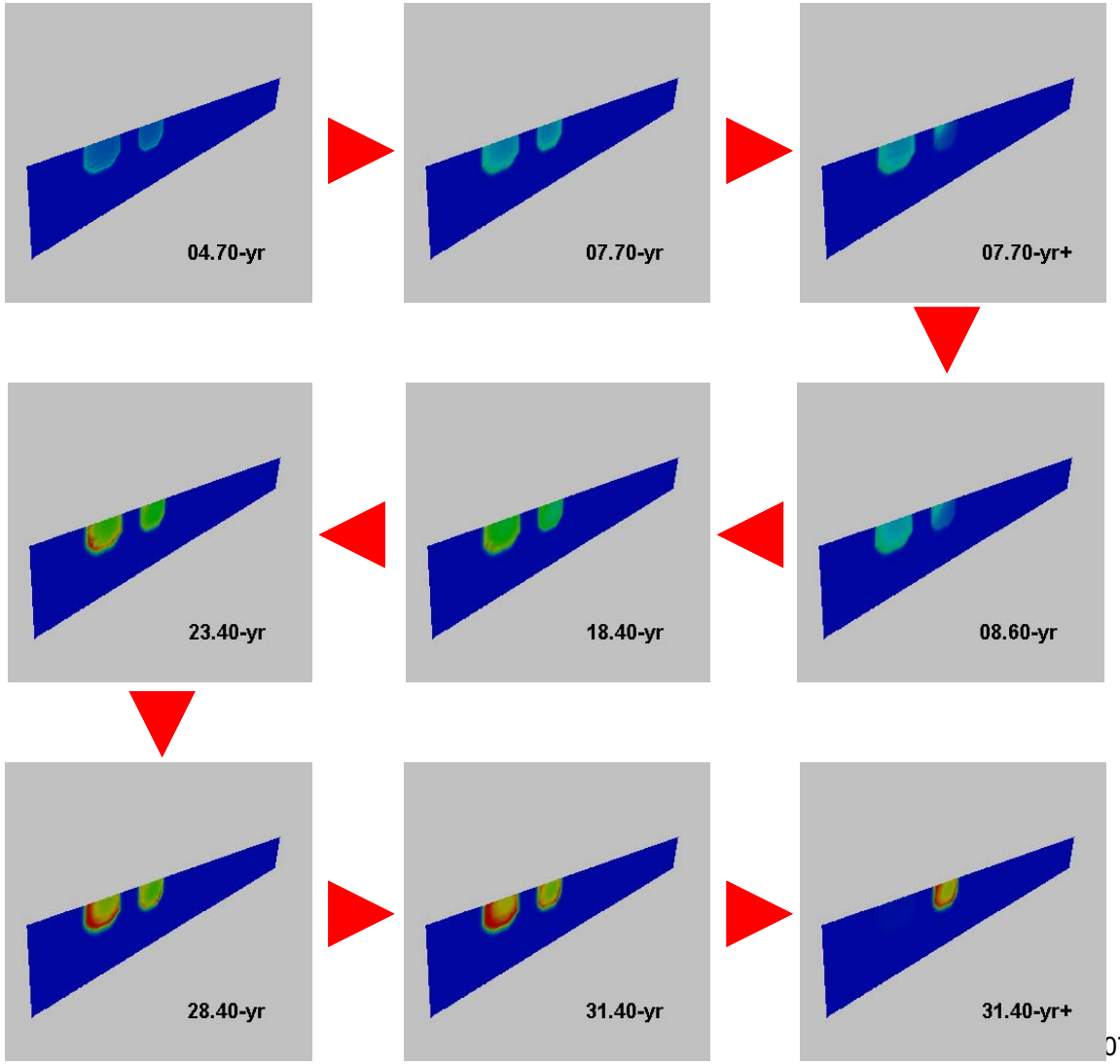
断層面上におけるせん断応力分布履歴
断層長さ= 450 km



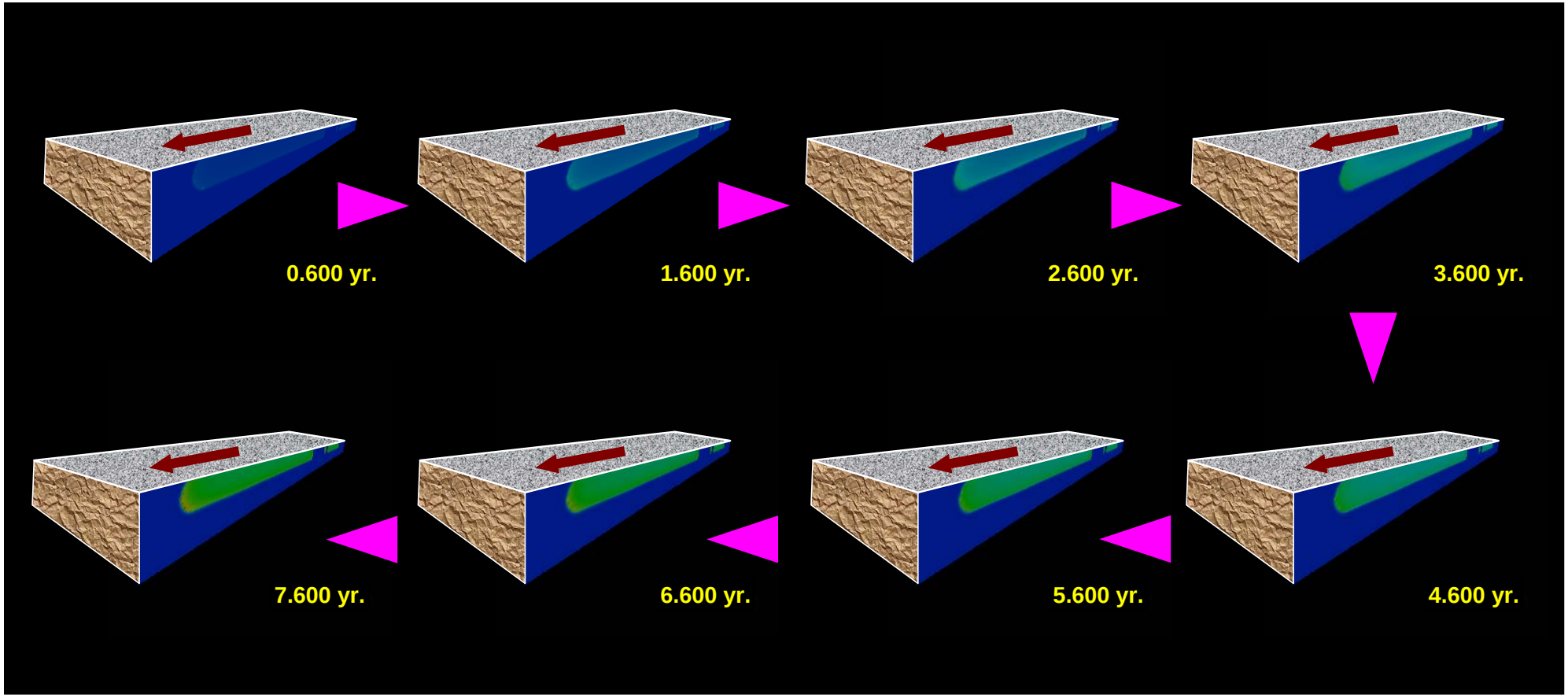




Transcurrent Plate Boundaries
San Andreas Faults, CA, USA
US Geological Survey

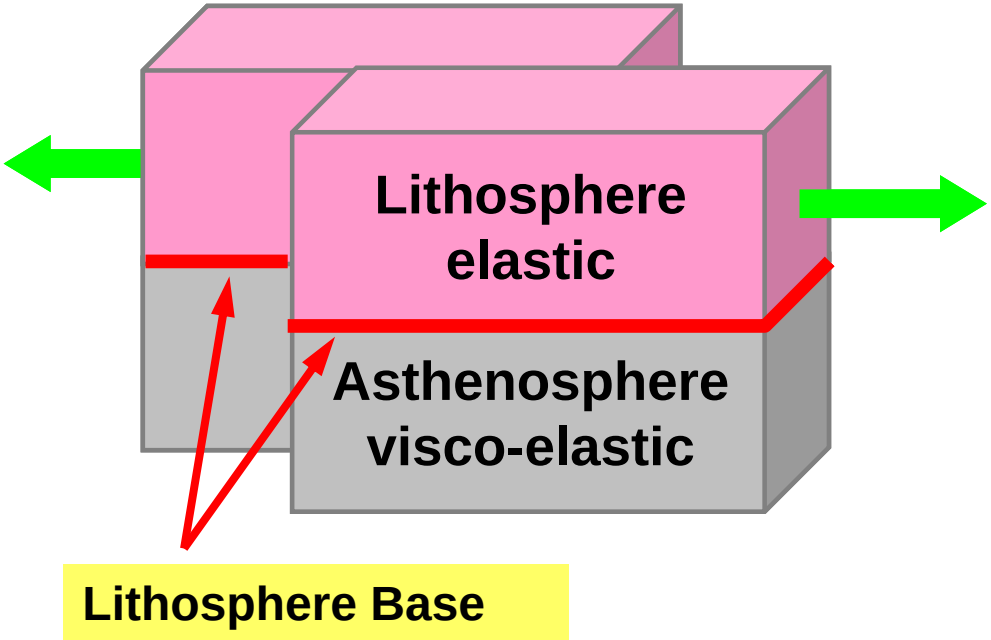


断層面上におけるせん断応力分布履歴
断層長さ= 1200 km, 16 PE`s on ES



横ずれ断層における地震発生サイクルのシミュレーションモデル (~1,200km fault length)

- Hashimoto & Matsu'ura's Method
  - Sato & Matsu'ura's Kinematic Model
  - すべり量，応力のカップルした非線形方程式を最小二乗法（Levenberg-Marquardt法）で解く。
  - 境界積分法（BIEM），
  - 密行列，直接法
- 2-way Parallelization for Different Points (Parmeter/Data) by Flat MPI
- Vector Optimization
- ~64 PEs of the Earth Simulator, nice parallel/vector efficiency

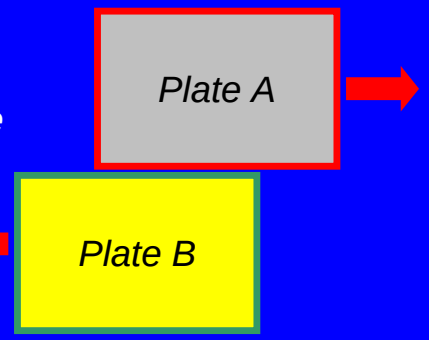


Effect of visco-elastic part is included in the source terms to the lithosphere base line. Only elastic part is modeled for the simulation.

Numerical Method
Hashimoto & Matsu'ura

Elastic lithosphere divided into 2 parts

infinitely long vertical interface
interaction between the 2 parts : fault slip



Fault Slip "w"

v_{pl} *steady plate motion rate*
u *perturbation*

$$w(\mathbf{x},t) = v_{pl} + u(\mathbf{x},t)$$

Shear Stress (σ) due to Fault Slip(w)

H *internal viscoelastic func. to a unit step slip on the plate boundary*

1st term : steady plate motion
2nd term: slip perturbation

$$\sigma(\mathbf{x},t) = \underbrace{\sigma_0(\mathbf{x},t)}_{\text{steady plate motion}} + \underbrace{\int_0^t \sum_s \int \frac{\partial u(\xi,\tau)}{\partial \tau} H(\mathbf{x}, t-\tau; \xi, 0)\, d\xi\, d\tau}_{\text{slip perturbation}}$$

Constitutive Relation between "w" and "u"

Aochi and Matsu'ura (1999) : New Model

$$\sigma(\mathbf{x},t) = f\left[w(\mathbf{x},t); \mathbf{x}\right]$$

Spatial Discretization

- *Slip "u" is defined by combination of "M" Cubic Spline Basis.*

$$u(\mathbf{x},t) = \sum_{m=1}^{M} a_m(t)\Psi_m(\mathbf{x})$$

Solve Equations

- *Construct Nonlinear Coupled Equation*
- *Linealized by Levenberg-Marquardt Method using "N" points: N>M*
- *Solve linearized equation for "$a_m(t)$" at each iteration*
- *Requires M*M dense matrix inversion*

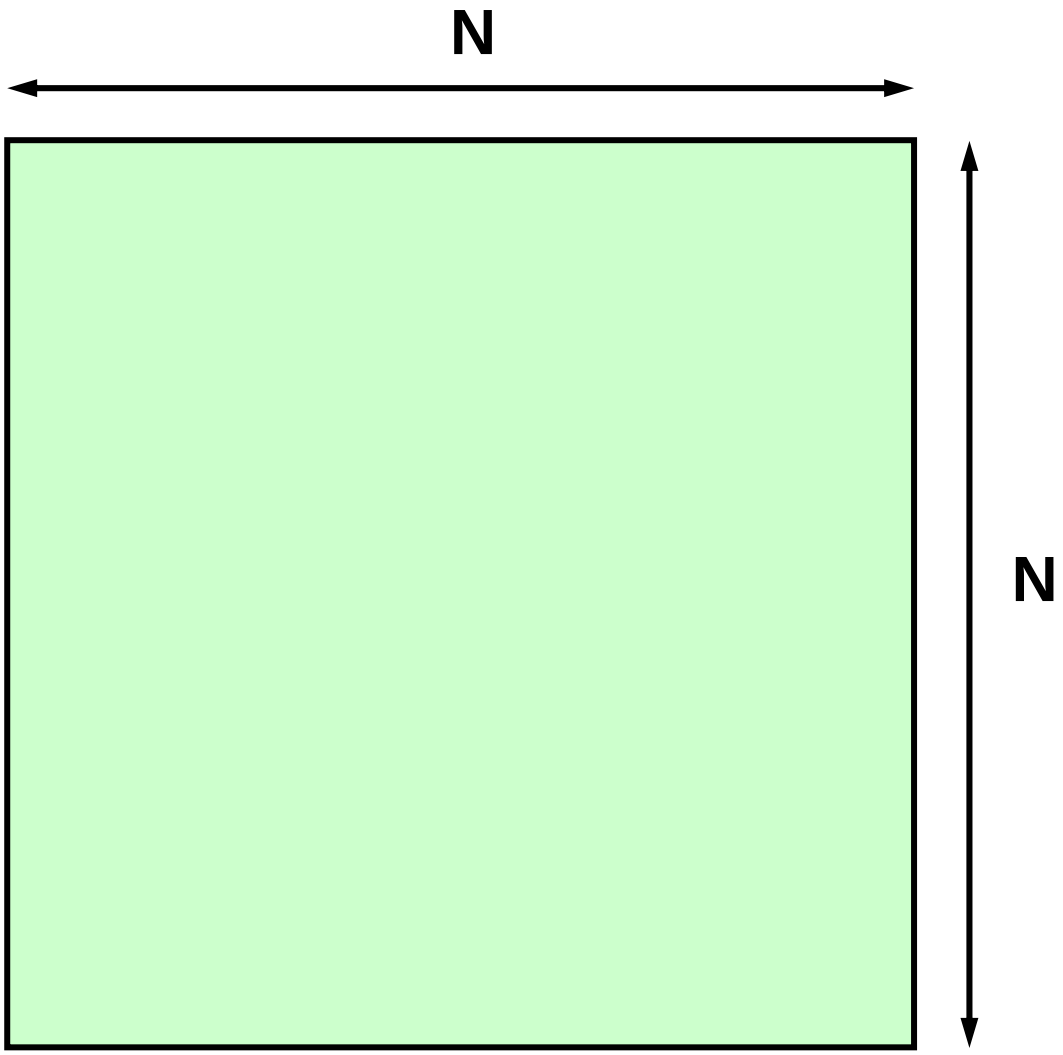
Results on Earth Simulator

Single PE, 15 steps for 150km length region

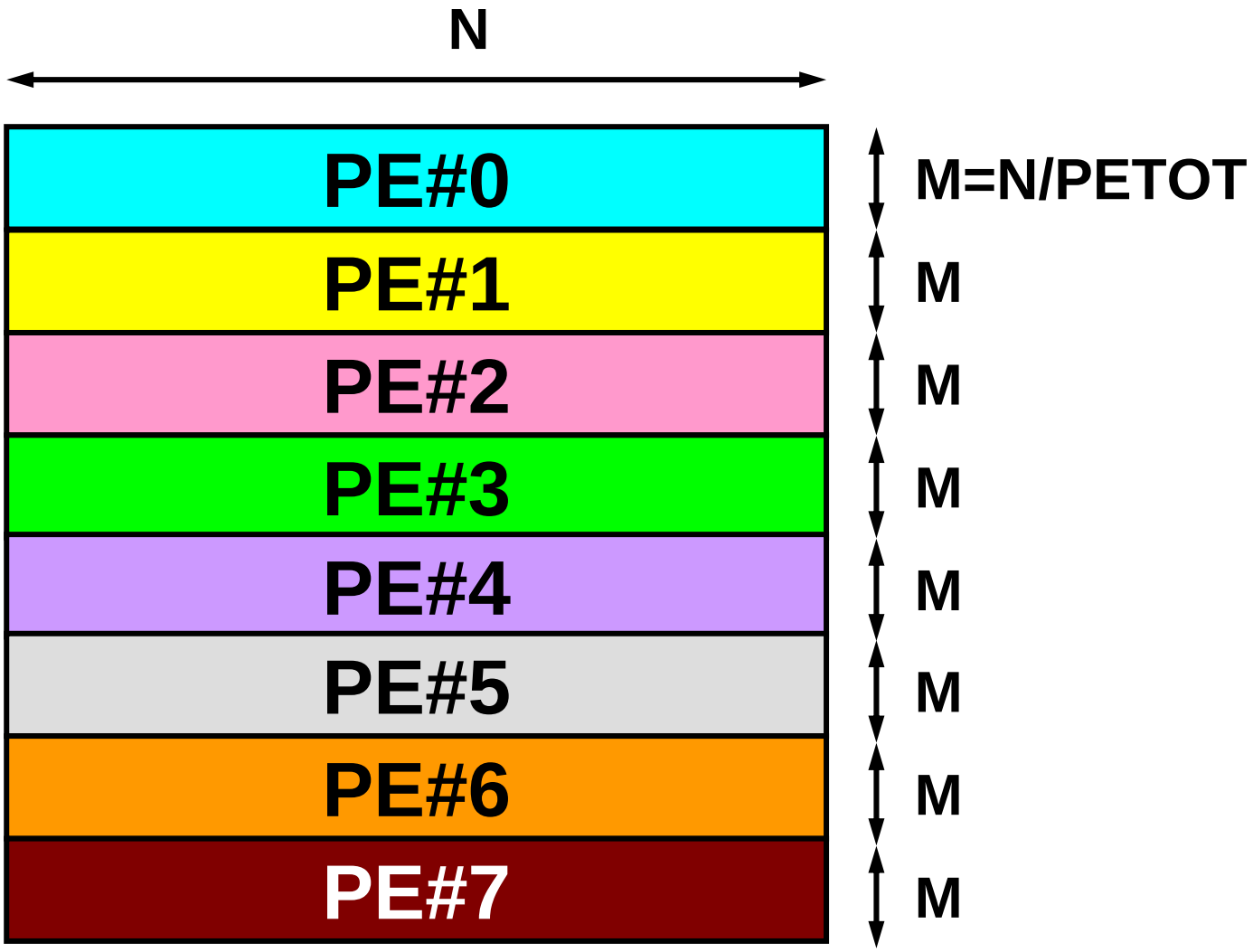
PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN
mrqcof	162	385.226(64.3)	2377.937	5669.4	1240.2	98.49	234.1
funcs	1125252	169.319(28.3)	0.150	839.0	261.9	77.30	54.8
srcinput	1	22.228(3.7)	22227.847	171.0	0.9	1.22	178.8
pgauss	108	12.887(2.2)	119.327	4260.5	1966.9	98.64	198.0
quasi_static	1	4.495(0.8)	4494.601	178.6	4.6	5.15	250.7
consti_parameter	1125252	4.445(0.7)	0.004	203.1	47.8	0.00	0.0
mrqmin	108	0.117(0.0)	1.085	3751.8	2.6	98.33	234.7
total	2250884	598.717(100.0)	0.266	3986.7	914.8	97.01	202.0

- mrqcof
  - 連立一次方程式の係数計算(最小二乗法適用部)
- funcs
  - Slip Perturbation計算部:mrqcofから呼び出される

係数行列の並列計算方法



係数行列の並列計算方法



Parallel Matrix Assembling for Linear EQN's: MRQCOF: original

```
  do ip= 1, PETOT
    is= (ip-1)*gN
    if (iflagM.eq.1) then
      do j= 1, gN
        wt= dydamatP(is+j)*sig2imatP(ip)
!CDIR NODEP
        do k= 1, gM
                k1= gMTBL(k)
          gA2(j,k)= gA2(j,k) + wt*dydamatP(is+k1)
        enddo
        gB2(j)= gB2(j) + dymatP(ip)*wt
      enddo
    endif
    chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
  enddo
```

- **gM=gN/PETOT**
- 最も内側のループの長さが短くなっている
  - **gA2**へのアクセスも連続でない

Parallel Matrix Assembling for Linear EQN's: MRQCOF: original

```
  do ip= 1, PETOT
    is= (ip-1)*gN
    if (iflagM.eq.1) then
      do j= 1, gN
        wt= dydamatP(is+j)*siq2imatP(ip)
!CDIR NODEP
        do k= 1, gM
                k1= gMTBL(k)
          qA2(j,k)= qA2(j,k) + wt*dydamatP(is+k1)
        enddo
        gB2(j)= gB2(j) + dymatP(ip)*wt
      enddo
    endif
    chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
  enddo
```

- **gM=gN/PETOT**
- 最も内側のループの長さが短くなっている
  - **gA2**へのアクセスも連続でない

Parallel Matrix Assembling for Linear EQN's: MRQCOF: optimized

```
 if (iflagM.eq.1) then
   do ip= 1, PETOT
     is= (ip-1)*gN
      k= 1
     k1= gMTBL(k)
!CDIR NODEP
     do j= 1, gN
       wt= dydamatP(is+j)*sig2imatP(ip)
       gA2(j,k)= gA2(j,k) + wt*dydamatP(is+k1)
       gB2(j)  = gB2(j)   + wt*dymatP  (ip)
     enddo

     do k= 2, gM
       k1= gMTBL(k)
!CDIR NODEP
       do j= 1, gN
         wt= dydamatP(is+j)*sig2imatP(ip)
         gA2(j,k)= gA2(j,k) + wt * dydamatP(is+k1)
       enddo
     enddo

     chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
   enddo
  else
!CDIR NODEP
   do ip= 1, PETOT
     is   = (ip-1)*gN
     chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
   enddo
endif
```

- gM=gN/PETOT
- 最も内側のループの長さをgNにしてある。
  - gA2へのアクセス連続
- 全体的な計算量は増加している。
  - wt

Parallel Matrix Assembling for Linear EQN's: MRQCOF: optimized

```
 if (iflagM.eq.1) then
   do ip= 1, PETOT
     is= (ip-1)*gN
      k= 1
     k1= gMTBL(k)
!CDIR NODEP
     do j= 1, gN
       wt= dydamatP(is+j)*sig2imatP(ip)
       gA2(j,k)= gA2(j,k) + wt*dydamatP(is+k1)
       gB2(j)  = gB2(j)   + wt*dymatP  (ip)
     enddo

     do k= 2, gM
       k1= gMTBL(k)
!CDIR NODEP
       do j= 1, gN
         wt= dydamatP(is+j)*sig2imatP(ip)
         gA2(j,k)= gA2(j,k) + wt * dydamatP(is+k1)
       enddo
     enddo

     chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
   enddo
  else
!CDIR NODEP
   do ip= 1, PETOT
     is   = (ip-1)*gN
     chisq= chisq + dymatP(ip)*dymatP(ip)*sig2imatP(ip)
   enddo
endif
```

- gM=gN/PETOT
- 最も内側のループの長さをgNにしてある。
  - gA2へのアクセス連続
- 全体的な計算量は増加している。
  - wt

Results on Earth Simulator

Single PE, 15 steps for 150km length region

Original

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	O-CACHE MISS	BANK CONF
mrqcof	162	385.226(64.3)	2377.937	5669.4	1240.2	98.49	234.1	4.8410	2.8251	0.1084
funcs	1125252	169.319(28.3)	0.150	839.0	261.9	77.30	54.8	1.7660	4.5700	6.4117
srcinput	1	22.228(3.7)	22227.847	171.0	0.9	1.22	178.8	3.4421	0.1314	0.0000
pgauss	108	12.887(2.2)	119.327	4260.5	1966.9	98.64	198.0	0.0922	0.2183	0.0098
quasi_static	1	4.495(0.8)	4494.601	178.6	4.6	5.15	250.7	0.3769	0.0692	0.0000
consti_parameter	1125252	4.445(0.7)	0.004	203.1	47.8	0.00	0.0	0.6829	0.7551	0.0000
mrqmin	108	0.117(0.0)	1.085	3751.8	2.6	98.33	234.7	0.0041	0.0025	0.0000
total	2250884	598.717(100.0)	0.266	3986.7	914.8	97.01	202.0	11.2052	8.5715	6.5299

Optimized

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	O-CACHE MISS	BANK CONF
funcs	1125252	168.392(46.2)	0.150	843.6	263.3	77.30	54.8	0.9748	4.4880	6.4117
mrqcof	162	153.722(42.1)	948.903	7102.9	3494.7	98.99	233.3	2.4568	3.9489	0.1084
srcinput	1	20.673(5.7)	20672.844	183.8	1.0	1.22	178.8	2.1630	0.1307	0.0000
pgauss	108	12.907(3.5)	119.510	4254.0	1963.9	98.64	198.0	0.1104	0.2287	0.0104
consti_parameter	1125252	4.538(1.2)	0.004	198.9	46.8	0.00	0.0	0.9358	0.6159	0.0000
quasi_static	1	4.464(1.2)	4464.177	179.8	4.6	5.15	250.7	0.3026	0.1485	0.0000
mrqmin	108	0.117(0.0)	1.084	3752.9	2.6	98.33	234.7	0.0043	0.0025	0.0000
total	2250884	364.813(100.0)	0.162	3549.2	1664.3	96.18	180.2	6.9477	9.5631	6.5306

- 全体的に計算時間は減少
- “MRQCOF” は計算量が増えているにも関わらず，計算時間減少
- “FUNCS”のバンクコンフリクトの問題

FUNCSにおけるBank Conflict

配列へのアクセスパターンに問題あり

```
  idX1= idint(zz_d)
  do p= 1, ma
    ipp= idnint(dabs(kk(p)-xx_d))
    if (ipp.gt.xmax0) then
      uu(p)= 0.d0
!CDIR NODEP
      do it= 0, itcnt                itcntは数10程度
        gss(p,it)= 0.d0
      enddo
     else
      idX2= idint(ll(p)/3.d0)
      uu(p)= u(ipp, idX1, idX2)
!CDIR NODEP
      do it= 0, itcnt
        gss(p,it)= gs(ipp, idX1, idX2, it)
      enddo
    endif
  enddo
```

- gss，gsについてメモリへのアクセスが不連続

FUNCSにおけるBank Conflict

配列へのアクセスパターンに問題あり

```
  idX1= idint(zz_d)
  do p= 1, ma
    ipp= idnint(dabs(kk(p)-xx_d))
    if (ipp.gt.xmax0) then
      uu(p)= 0.d0
!CDIR NODEP
      do it= 0, itcnt                    itcntは数10程度
        gss(p,it)= 0.d0
      enddo
     else
      idX2= idint(ll(p)/3.d0)
      uu(p)= u(ipp, idX1, idX2)
!CDIR NODEP
      do it= 0, itcnt
        gss(p,it)= gs(ipp, idX1, idX2, it)
      enddo
    endif
  enddo
```

- gss，gsについてメモリへのアクセスが不連続

FUNCSにおけるBank Conflict

改良版

```
  idX1= idint(zz_d)
  it= 0
!CDIR NODEP
  do  p= 1, ma
    ipp= idnint(dabs(kk(p)-xx_d))
    if (ipp.gt.xmax0) then
       uu (p)    = 0.d0
       gss(p,it)= 0.d0
      else
       idX2= idint(ll(p)/3.d0)
       uu (p)    = u (ipp,      idX2, idX1)
       gss(p,it)= gs(ipp, it, idX2, idX1)
     endif
   enddo

   do it= 1, itcnt
!CDIR NODEP
     do  p= 1, ma
       ipp= idnint(dabs(kk(p)-xx_d))
       if (ipp.gt.xmax0) then
         gss(p,it)= 0.d0
        else
         idX2= idint(ll(p)/3.d0)
         gss(p,it)= gs(ipp, it, idX2, idX1)
       endif
     enddo
   enddo
```

FUNCSにおけるBank Conflict

改良版

```
  idX1= idint(zz_d)
  it= 0
!CDIR NODEP
  do  p= 1, ma
    ipp= idnint(dabs(kk(p)-xx_d))
    if (ipp.gt.xmax0) then
       uu (p)    = 0.d0
       gss(p,it)= 0.d0
     else
      idX2= idint(ll(p)/3.d0)
      uu (p)    = u (ipp,      idX2, idX1)
      gss(p,it)= gs(ipp, it, idX2, idX1)
    endif
   enddo

   do it= 1, itcnt
!CDIR NODEP
     do  p= 1, ma
       ipp= idnint(dabs(kk(p)-xx_d))
       if (ipp.gt.xmax0) then
         gss(p,it)= 0.d0
        else
         idX2= idint(ll(p)/3.d0)
         gss(p,it)= gs(ipp, it, idX2, idX1)
       endif
     enddo
   enddo
```

- 最内側ループ
  “it” ⇒ “p” for “gss(p,it)”.

FUNCSにおけるBank Conflict

改良版

```
  idX1= idint(zz_d)
  it= 0
!CDIR NODEP
  do  p= 1, ma
    ipp= idnint(dabs(kk(p)-xx_d))
    if (ipp.gt.xmax0) then
       uu (p)    = 0.d0
       gss(p,it)= 0.d0
      else
       idX2= idint(ll(p)/3.d0)
       uu (p)    = u (ipp,      idX2, idX1)
       gss(p,it)= gs(ipp, it, idX2, idX1)
     endif
   enddo

   do it= 1, itcnt
!CDIR NODEP
     do  p= 1, ma
       ipp= idnint(dabs(kk(p)-xx_d))
       if (ipp.gt.xmax0) then
         gss(p,it)= 0.d0
        else
         idX2= idint(ll(p)/3.d0)
         gss(p,it)= gs(ipp, it, idX2, idX1)
       endif
     enddo
   enddo
```

- アクセスパターン
  “gss(ipp,idX1,idX2,it)” ⇒
  “gss(ipp,it,idx1,idX2)”.

Results on Earth Simulator

Single PE, 15 steps for 150km length region

Original

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	O-CACHE MISS	BANK CONF
mrqcof	162	385.226(64.3)	2377.937	5669.4	1240.2	98.49	234.1	4.8410	2.8251	0.1084
funcs	1125252	169.319(28.3)	0.150	839.0	261.9	77.30	54.8	1.7660	4.5700	6.4117
srcinput	1	22.228(3.7)	22227.847	171.0	0.9	1.22	178.8	3.4421	0.1314	0.0000
pgauss	108	12.887(2.2)	119.327	4260.5	1966.9	98.64	198.0	0.0922	0.2183	0.0098
quasi_static	1	4.495(0.8)	4494.601	178.6	4.6	5.15	250.7	0.3769	0.0692	0.0000
consti_parameter	1125252	4.445(0.7)	0.004	203.1	47.8	0.00	0.0	0.6829	0.7551	0.0000
mrqmin	108	0.117(0.0)	1.085	3751.8	2.6	98.33	234.7	0.0041	0.0025	0.0000
total	2250884	598.717(100.0)	0.266	3986.7	914.8	97.01	202.0	11.2052	8.5715	6.5299

Optimized

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	O-CACHE MISS	BANK CONF
funcs	1125252	168.392(46.2)	0.150	843.6	263.3	77.30	54.8	0.9748	4.4880	6.4117
mrqcof	162	153.722(42.1)	948.903	7102.9	3494.7	98.99	233.3	2.4568	3.9489	0.1084
srcinput	1	20.673(5.7)	20672.844	183.8	1.0	1.22	178.8	2.1630	0.1307	0.0000
pgauss	108	12.907(3.5)	119.510	4254.0	1963.9	98.64	198.0	0.1104	0.2287	0.0104
consti_parameter	1125252	4.538(1.2)	0.004	198.9	46.8	0.00	0.0	0.9358	0.6159	0.0000
quasi_static	1	4.464(1.2)	4464.177	179.8	4.6	5.15	250.7	0.3026	0.1485	0.0000
mrqmin	108	0.117(0.0)	1.084	3752.9	2.6	98.33	234.7	0.0043	0.0025	0.0000
total	2250884	364.813(100.0)	0.162	3549.2	1664.3	96.18	180.2	6.9477	9.5631	6.5306

Final

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	O-CACHE MISS	BANK CONF
mrqcof	162	151.462(60.2)	934.950	7208.9	3546.8	98.99	233.3	4.2596	0.6754	0.1108
funcs	1125252	57.366(22.8)	0.051	5323.4	1284.5	98.56	230.9	2.3050	1.0004	0.1203
srcinput	1	21.015(8.4)	21014.591	180.8	1.0	1.22	178.8	2.4005	0.1383	0.0000
pgauss	108	12.832(5.1)	118.812	4279.0	1975.5	98.64	198.0	0.0939	0.1742	0.0104
quasi_static	1	4.634(1.8)	4633.743	173.2	4.4	5.15	250.7	0.5050	0.1040	0.0000
consti_parameter	1125252	4.221(1.7)	0.004	213.9	50.3	0.00	0.0	1.0785	0.0555	0.0000
mrqmin	108	0.117(0.0)	1.083	3758.9	2.6	98.33	234.7	0.0041	0.0025	0.0000
total	2250884	251.646(100.0)	0.112	5794.3	2529.4	98.52	231.2	10.6466	2.1503	0.2415

チューニング：まとめ

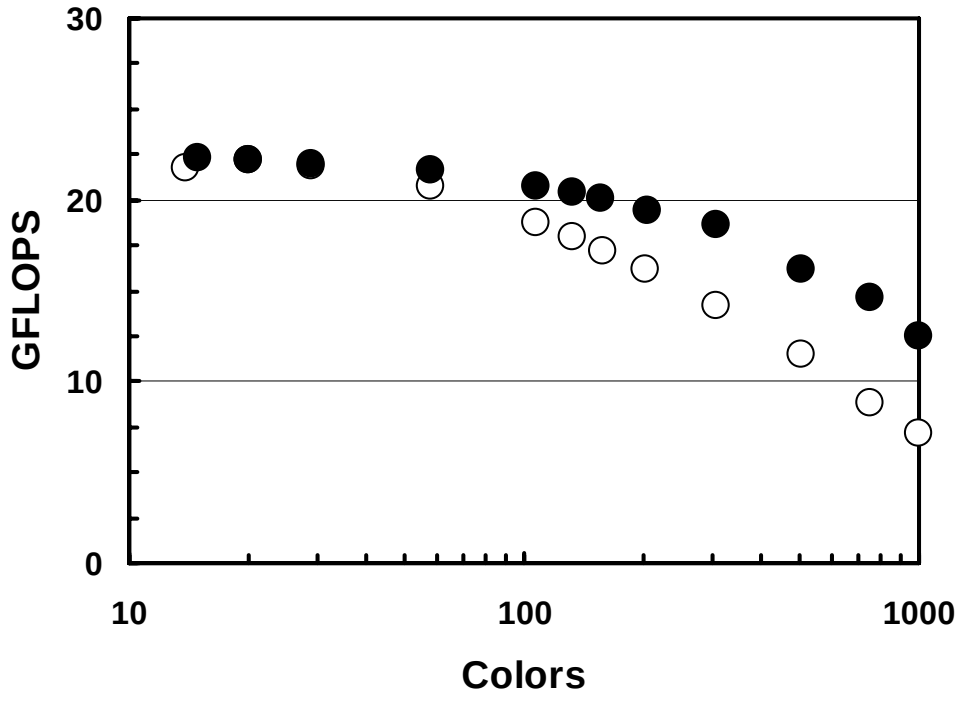
- 共通事項
  - メモリアクセスの連続性
  - 局所性
  - 計算順序の変更によって計算結果が変わる可能性について注意すること
- ベクトルプロセッサ
  - ループ長を大きくとる。
- スカラープロセッサ
  - キャッシュを有効利用，細切れにしたデータを扱う。
  - 自動チューニング：最適ブロックサイズの自動決定など
    - 片桐（東大情報基盤センター）他「AT（自動チューニング）研究会」
- スカラーとベクトル
  - 一方に対する最適化が逆効果に働くことがある。

チューニング：まとめ

- もう一件，話したいことがあるのだが，今日は時間が足りない。最終講義（7月18日）の日に少し時間が余る予定なので，そのときに話す。

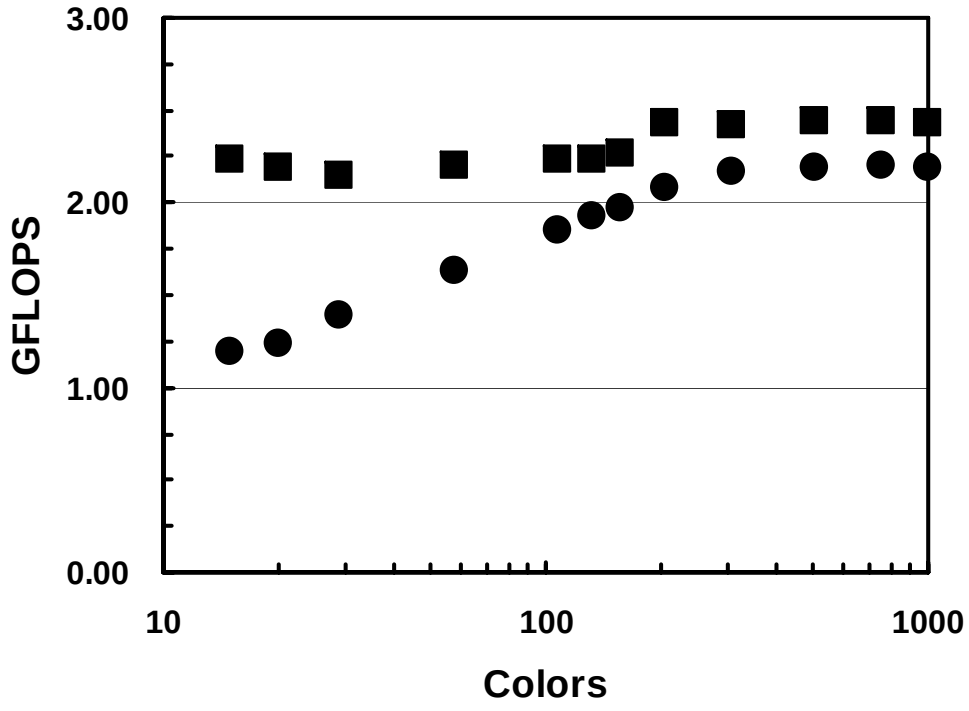
Multicolor オーダリング（後期に扱う）の例

地球シミュレータ：色数（グループ数）を増やすと，一色あたりのループ長さが短くなり性能低下



Multicolor オーダリング（後期に扱う）の例

Cenju：色数（グループ数）を増やすと，一色あたりの
問題サイズが小さくなりキャッシュを有効利用可能



IBM BlueGene/L: Double FPU

- 各プロセッサに2つのFPU（Floating-Point Unit）
- スカラープロセッサであるが，ベクトル機的な扱いが適している
  - 現在研究中

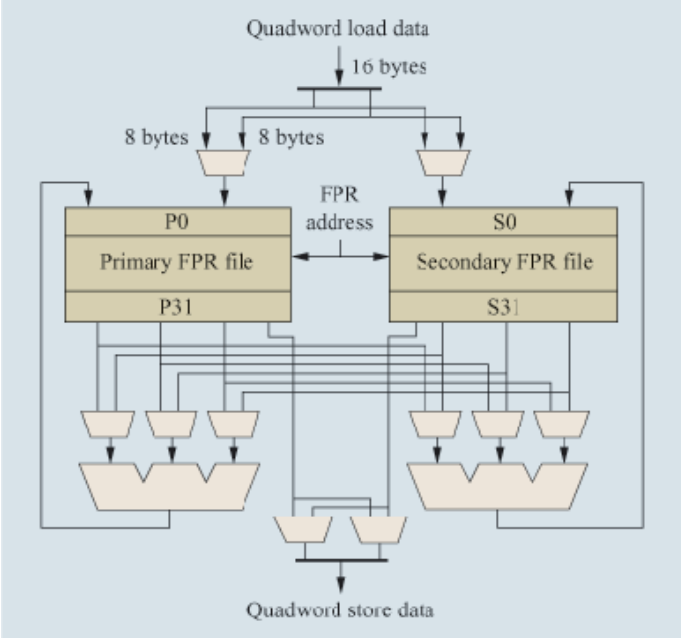


Figure 1

Architecture of the IBM PowerPC 440 FP2—primary and secondary data flow.

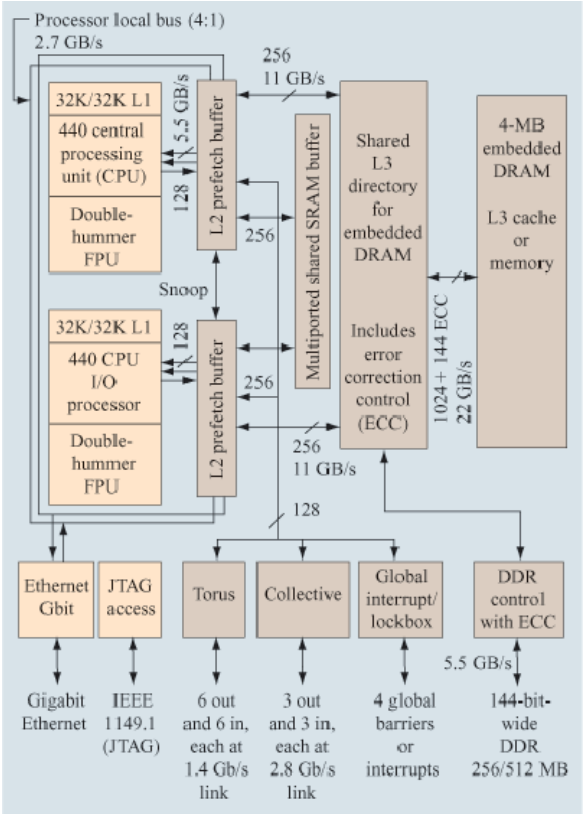


Figure 1

Blue Gene/L compute (BLC) chip architecture. Yellow shading indicates off-the-shelf cores. ©2002 IEEE. Reprinted with permission from G. Almasi et al., "Cellular Supercomputing with System-on-a-Chip," *Digest of Technical Papers*, 2002 IEEE International Solid-State Circuits Conference.