チューニング入門

2012年度冬学期

中島研吾

科学技術計算Ⅱ（4820-1028）・コンピュータ科学特別講義（4810-1205）

（並列有限要素法）

- 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
- チューニング例：スカラープロセッサ
- メモリ性能

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
- チューニングされたライブラリを使う
- 良い並列プログラム＝良いシリアルプログラム
- 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.

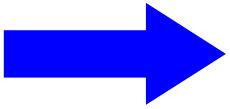
「チューニング」の心得

- メモリアクセスパターンに配慮
- 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**
  - **オプション，ディレクティヴ：実際に試して速い手法を採用するべき**
  - **今日の話は一般的なもの**

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
. . .
```

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）
  - 他にも処理系，コンパイラに応じたプロファイラがある
    - pgprof：　PGIコンパイラ
    - Vtune：　Intel
    - https://oakleaf-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal_u.ja/index.cgi

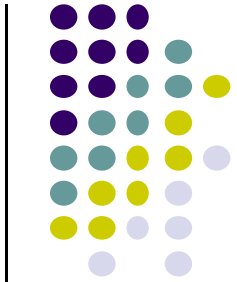
ファイルコピー：Fortranのみ

簡単なプログラムなので，自分で作ってください

```
>$ cd <$O-fem2>

FORTRAN
>$ cp /home/z30088/fem2/s3.tar .
>$ tar xvf s3.tar

確認
>$ cd mpi/S3

このディレクトリを本講義では <$O-S3> と呼ぶ。
<$O-S3> = <$O-fem2>/mpi/S3
```

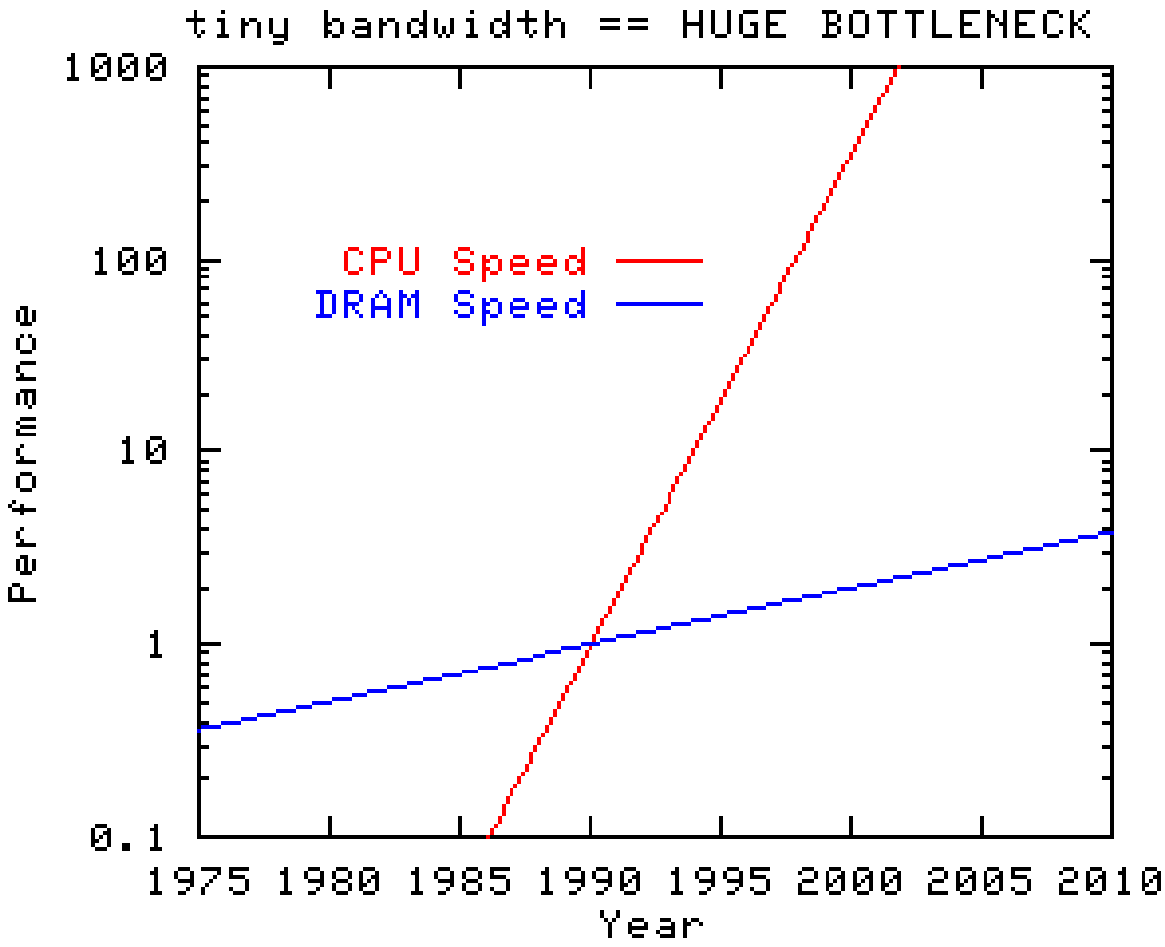
- チューニングとは
- **ベクトルプロセッサとスカラープロセッサ**
- チューニング例：スカラープロセッサ
- メモリ性能

スカラー／ベクトルプロセッサ



- 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　→　マルチコア，メニーコア
  - 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

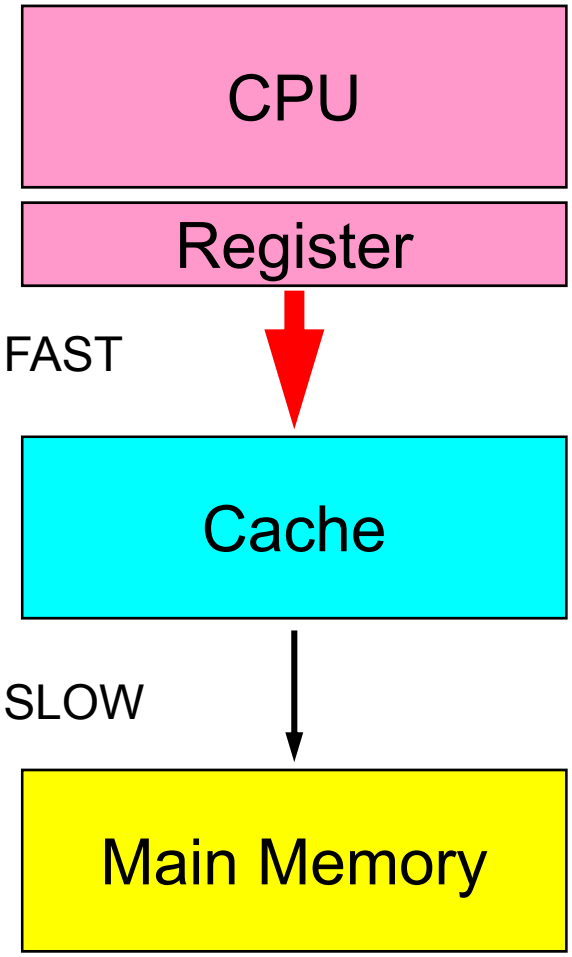
マイクロプロセッサの動向
CPU性能，メモリバンド幅のギャップ



http://www.streambench.org/

スカラープロセッサ

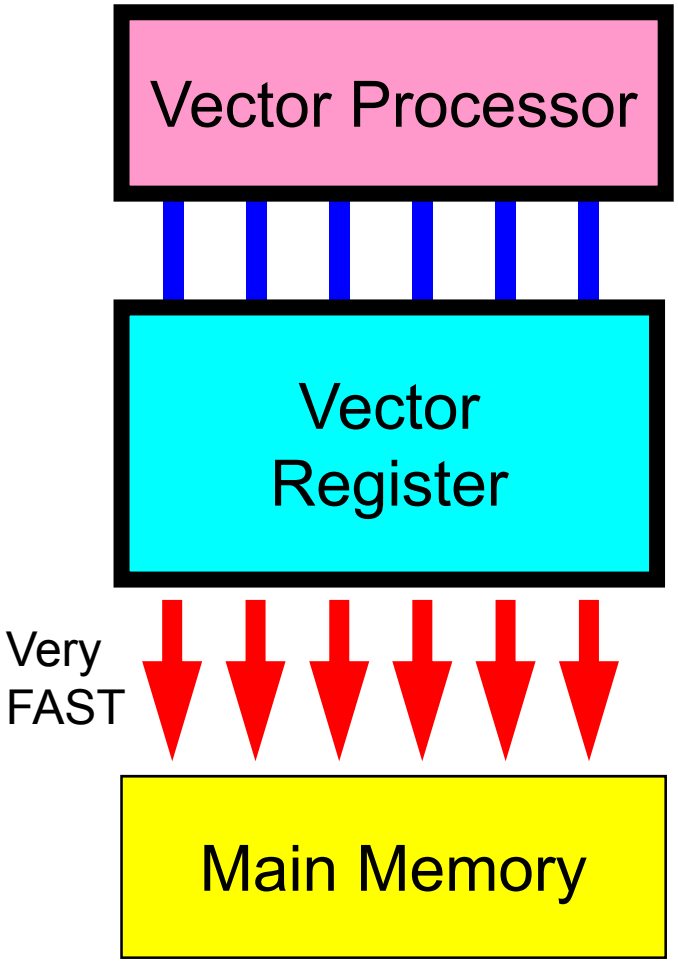
CPU–キャッシュ–メモリの階層構造



小容量 (MB)
高価
大きい (数億以上のトランジスタ)
命令キャッシュ（プログラム），データキャッシュ
多階層：L1,L2,L3

大容量 (GB)
廉価

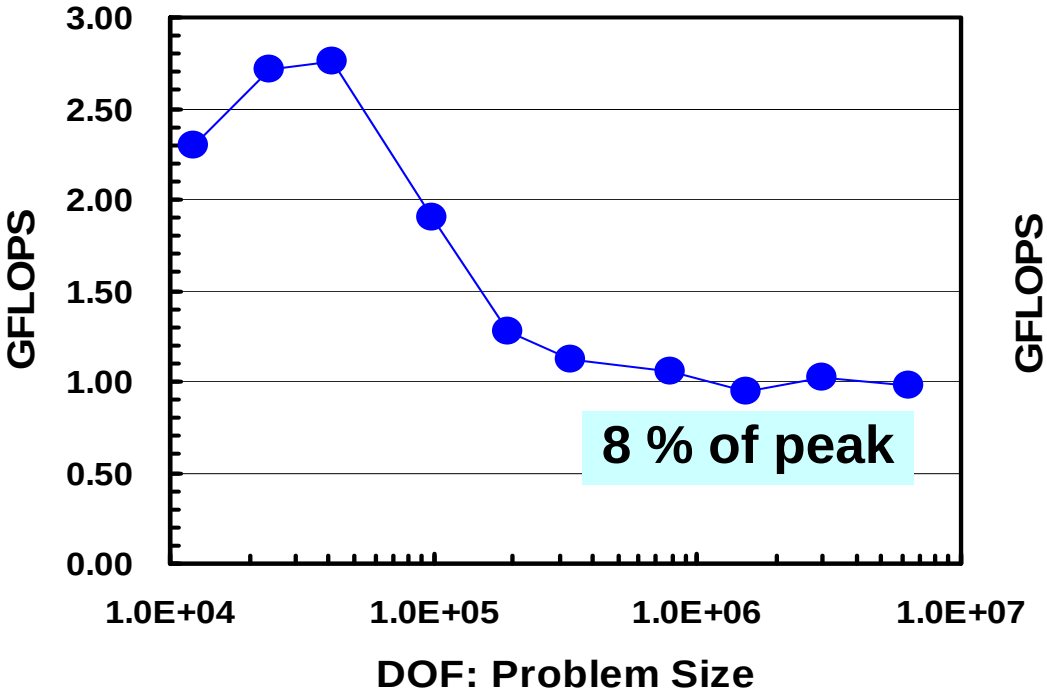
ベクトルプロセッサ
ベクトルレジスタと高速メモリ

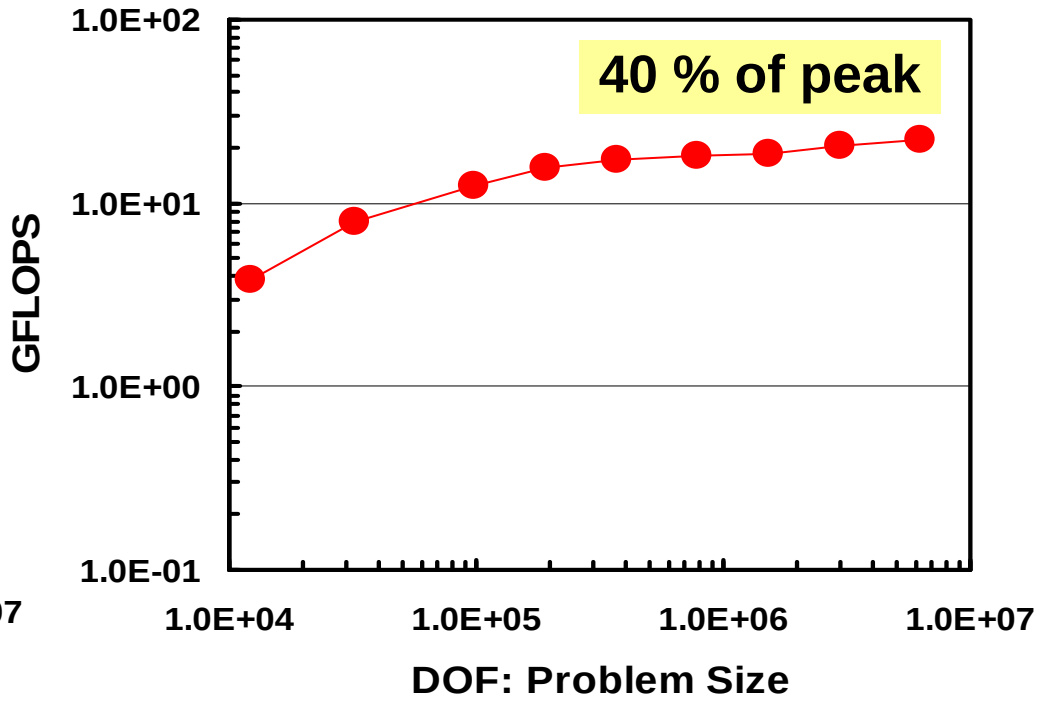


- 単純構造のDOループの並列処理
- 単純，大規模な演算に適している

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

典型的な挙動

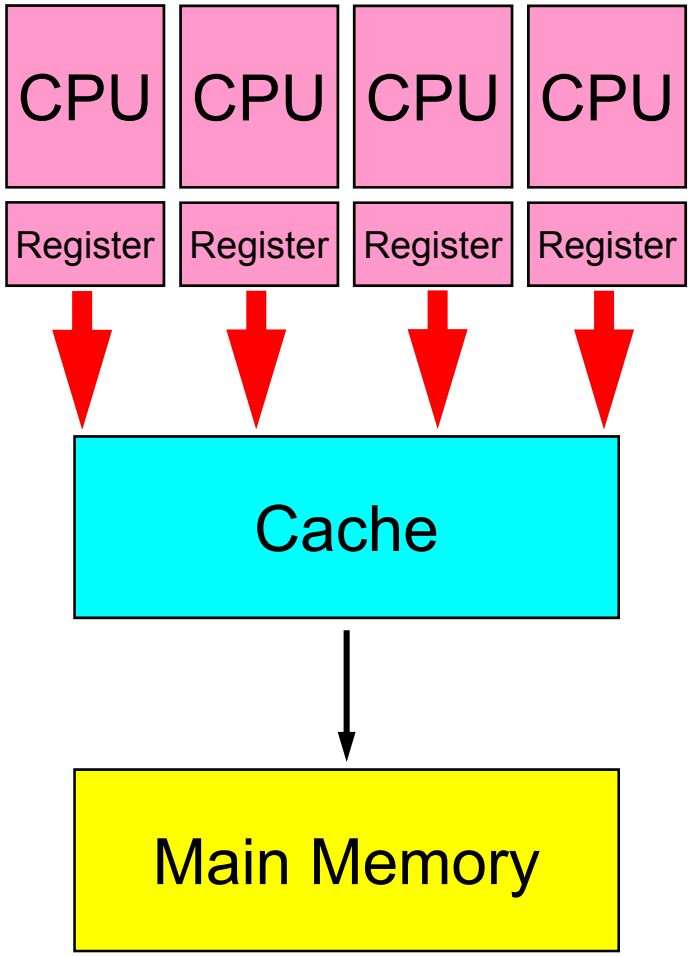


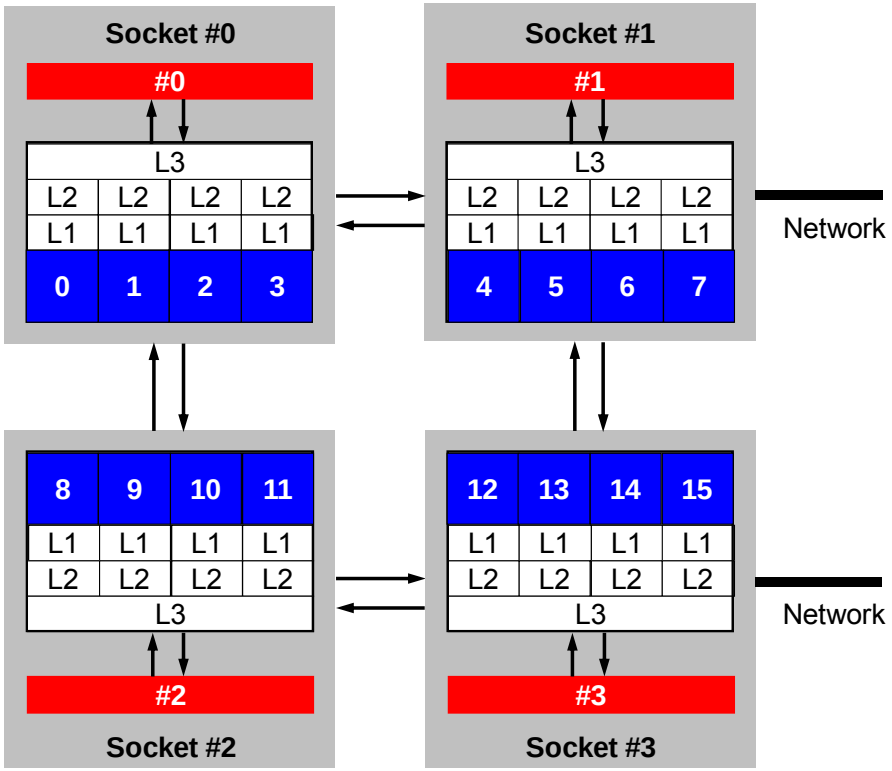


IBM-SP3:
問題サイズが小さい場合はキャッシュの影響のため性能が良い

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

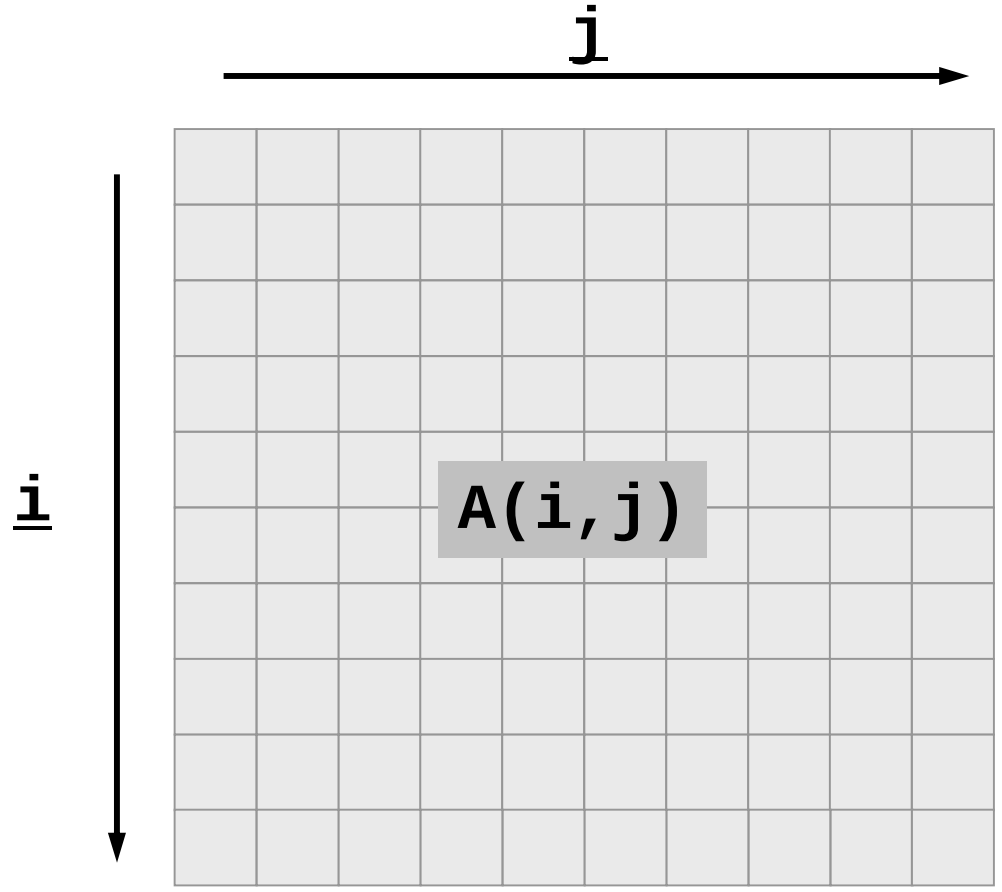
マルチコア，メニーコア
複数のコアでメモリ，キャッシュを共有
→競合，性能低下





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。
- 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
- **チューニング例：スカラープロセッサ**
- メモリ性能

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

BLAS: Basic Linear Algebra Subprograms

- ベクトル，（密）行列の基本的な演算に関するライブラリAPI

- レベル1：ベクトル演算（内積，DAXPY）
- レベル2：（行列×ベクトル）積
- レベル3：（行列×行列）積

- LINPACK等で利用
  - DGEMM：行列×行列（レベル3）

Scalar

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。

<$O-S3>/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
```

```
$> cd <$O-S3>
$> mpifrtpx -Kfast t2.f

<modify “go1.sh”>
$> pjsub go1.sh （1プロセス）

    1.560717E-01
    1.012069E-01
    7.471654E-02
```

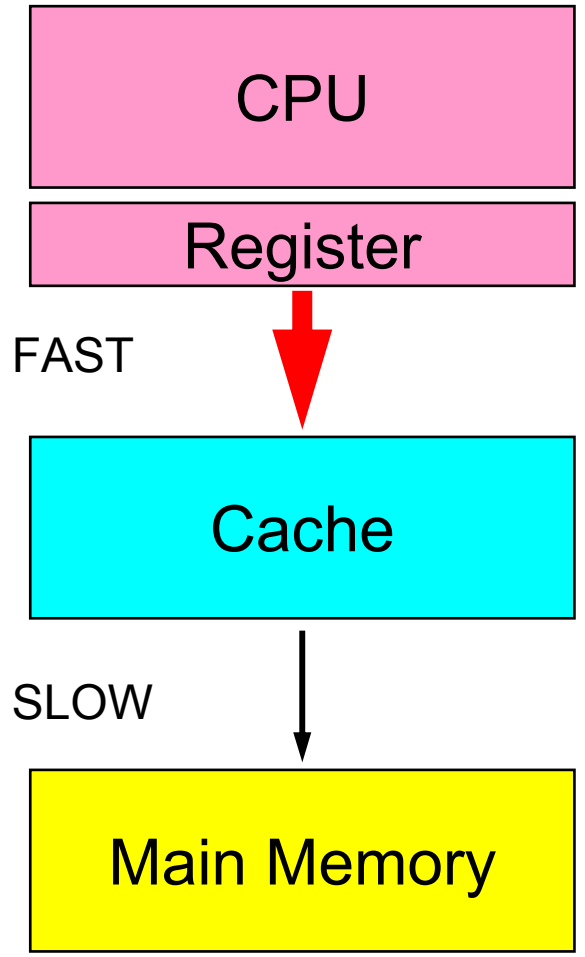
ジョブスクリプト：go1.sh 1コア用

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt64"
#PJM -j
#PJM -o "test.lst"
#PJM --mpi "proc=1"

mpiexec ./a.out
```

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



Scalar

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), B(i), C(i,j)に対して，各ループでロード・ストアが発生する。

Scalar

ループアンローリング

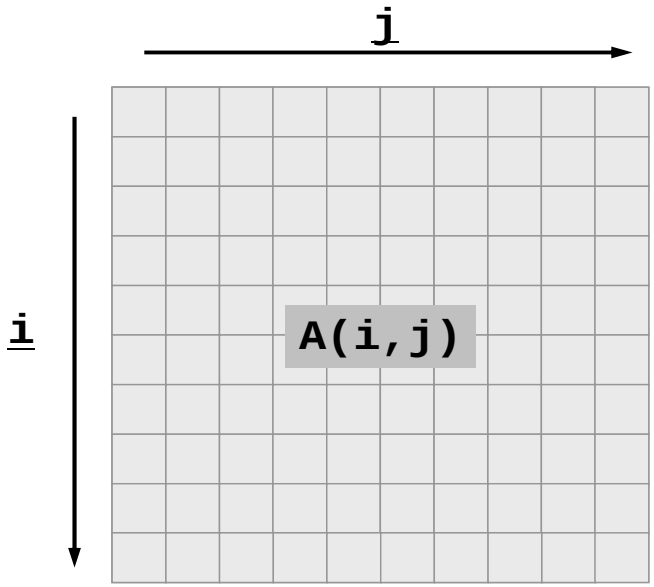
ロード・ストアの削減（4/4）

```
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。
- 計算に対するメモリアクセス（ロード・ストア）の割合を減らせる。
- 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：A[0][0], A[0][1], A[0][2], …, A[N-1][0], A[N-1][1],…,A[N-1][N-1]
- 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)。

Scalar

ループ入替によるメモリ参照最適化（2/2）

<$O-S3>/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
```

```
TYPE-A
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    A[i][j]= A[i][j] + B[i][j];
  }
}

TYPE-B
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    A[i][j]= A[i][j] + B[i][j];
  }
}
```

```
$> cd <$O-S3>
$> mpifrtpx -O1 2d-1.f
$> pjsub go1.sh
### N ###      500
WORSE      3.730428E-03
BETTER     9.525069E-04
### N ###     1000
WORSE      1.642722E-02
BETTER     3.771729E-03
### N ###     1500
WORSE      4.304052E-02
BETTER     8.295263E-03
### N ###     2000
WORSE      6.198836E-02
BETTER     1.455921E-02
### N ###     2500
WORSE      1.180517E-01
BETTER     2.305677E-02
### N ###     3000
WORSE      1.675602E-01
BETTER     3.311505E-02
### N ###     3500
WORSE      2.324806E-01
BETTER     4.509363E-02
### N ###     4000
WORSE      2.594637E-01
BETTER     5.803503E-02
```

ループ入替によるメモリ参照最適化（2/2）
-Kfastにすると，高速化，差は無くなる

```
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
```

```
TYPE-A
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    A[i][j]= A[i][j] + B[i][j];
  }
}

TYPE-B
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    A[i][j]= A[i][j] + B[i][j];
  }
}
```

```
$> cd <$O-S3>
$> mpifrtpx -Kfast 2d-1.f
$> pjsub go1.sh
### N ###      500
WORSE      3.017990E-04
BETTER     3.012992E-04
### N ###     1000
WORSE      1.213297E-03
BETTER     1.189598E-03
### N ###     1500
WORSE      2.675394E-03
BETTER     3.808392E-03
### N ###     2000
WORSE      4.778489E-03
BETTER     4.779590E-03
### N ###     2500
WORSE      7.354485E-03
BETTER     7.370183E-03
### N ###     3000
WORSE      1.060798E-02
BETTER     1.060808E-02
### N ###     3500
WORSE      1.444077E-02
BETTER     1.443657E-02
### N ###     4000
WORSE      1.977946E-02
BETTER     1.977636E-02
```

Scalar

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。

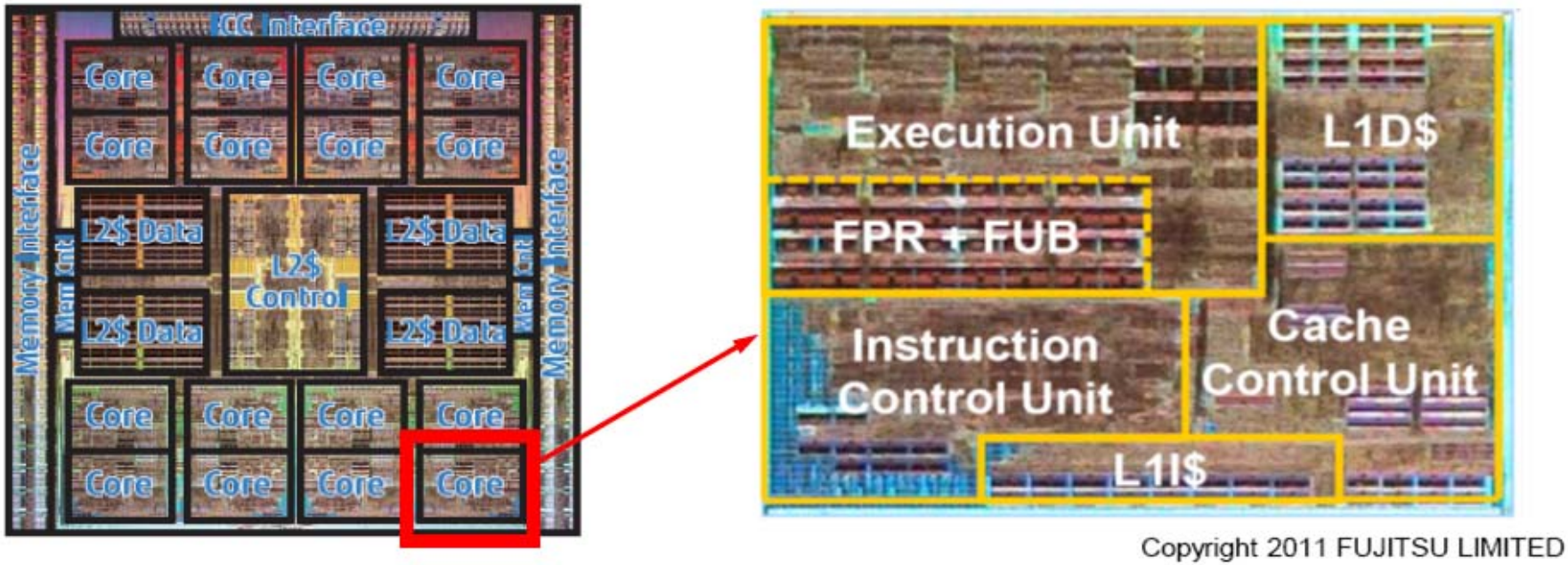
キャッシュの有効利用

- **SPARC64™ IXfxのキャッシュ**
  - L1: 32KB／コア（命令，データ各）
  - L2: 12MB／ノード
  - 実際は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。

- Sector Cache
  - Software Controllable Cache
  - キャッシュに残しておきたいデータを指定できる
    - キャッシュ容量には上限があるので，後からどんどんデータが入ってくると，最初にロードされたデータは追い出されてしまう
  - 本講義では扱わない

SPARC64™ IXfx

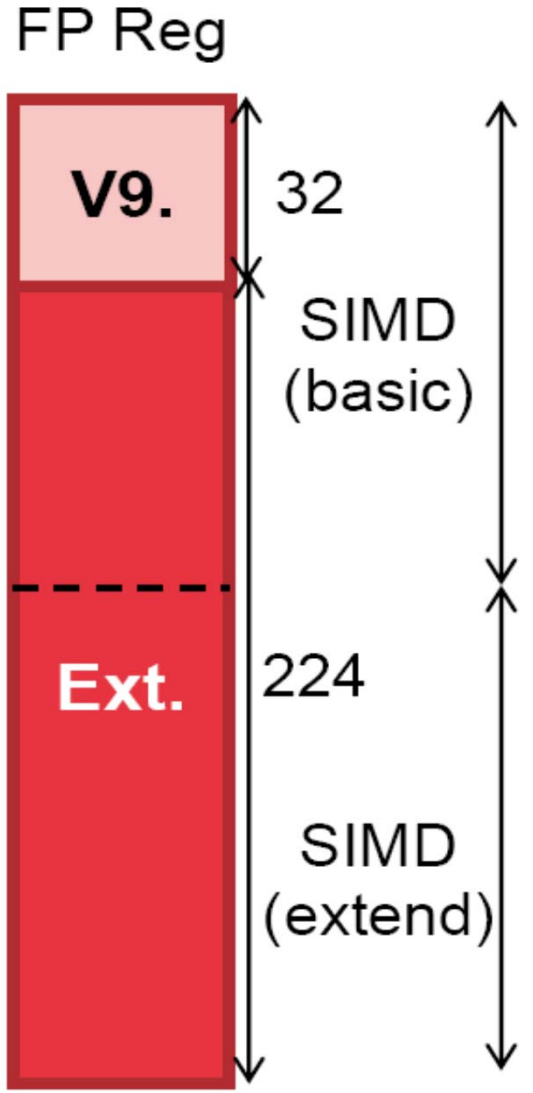
- HPC-ACE (High Performance Computing – Arithmetic Computational Extensions)
- UMA, not NUMA
- H/W barrier for high-speed synchronization of on-chip cores



HPC-ACE

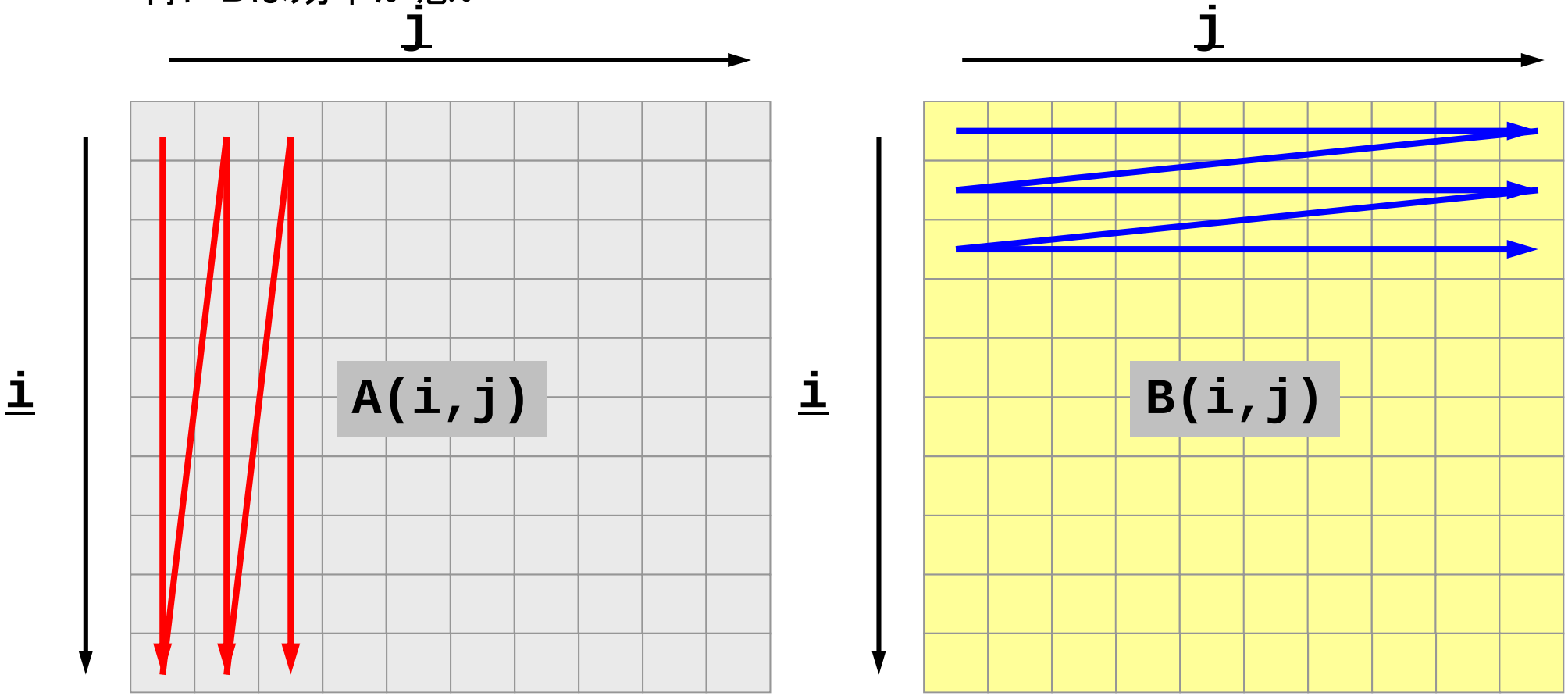
High Performance Computing – Arithmetic Computational Extensions

- Enhanced instruction set for the SPARC-V9 instruction set arch.
  - High-Performance & Power-Aware
- Extended Number of Registers
  - FP Registers: 32→256
- S/W Controllable Cache
  - “Sector Cache”
  - for keeping reusable data sets on cache
- High-Performance, Efficient
  - Optimized FP functions
  - Conditional Operation



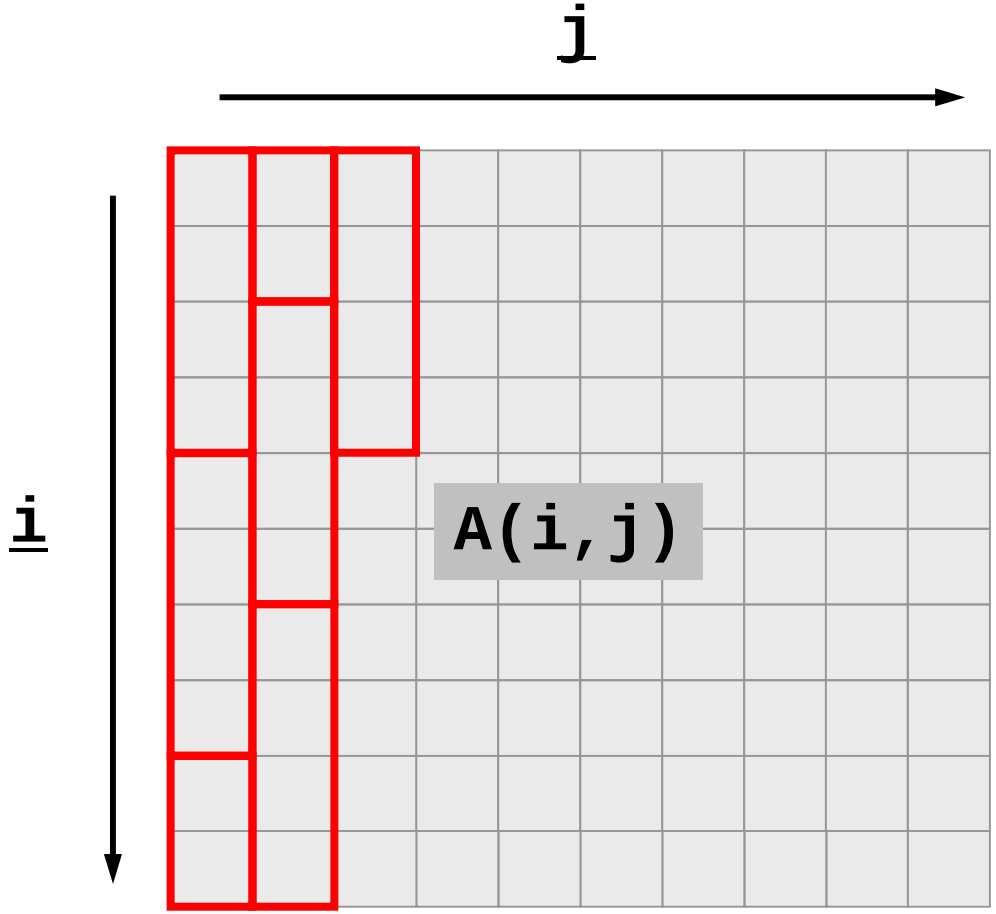
ブロック化によるキャッシュミス削減（2/7）

- 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



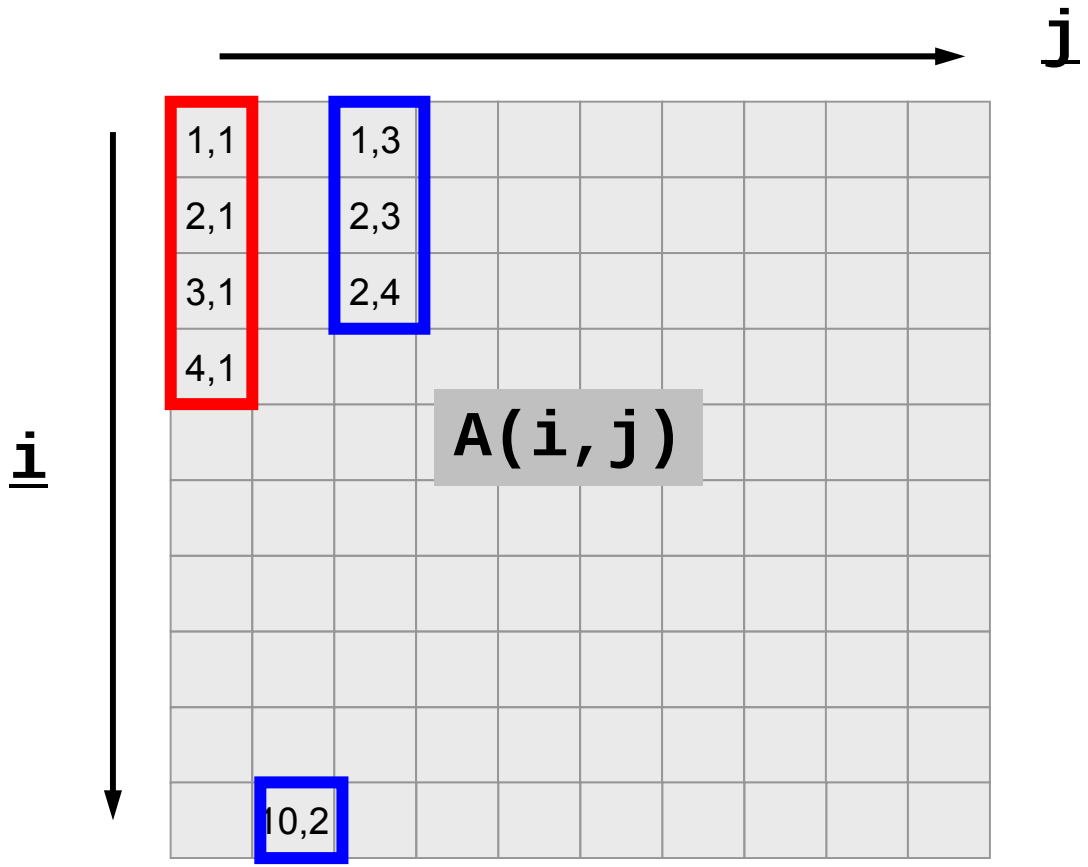
ブロック化によるキャッシュミス削減（3/7）

- 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。



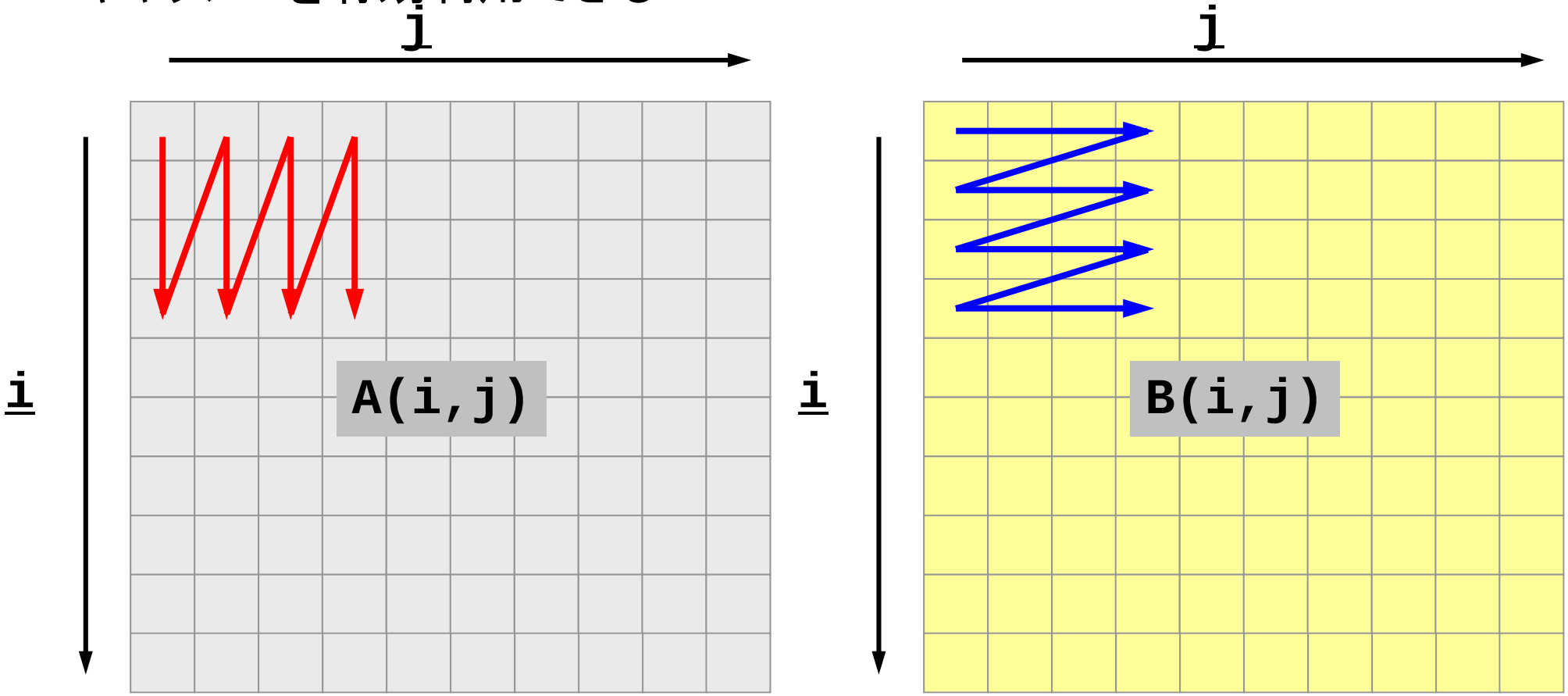
ブロック化によるキャッシュミス削減（4/7）

- 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。



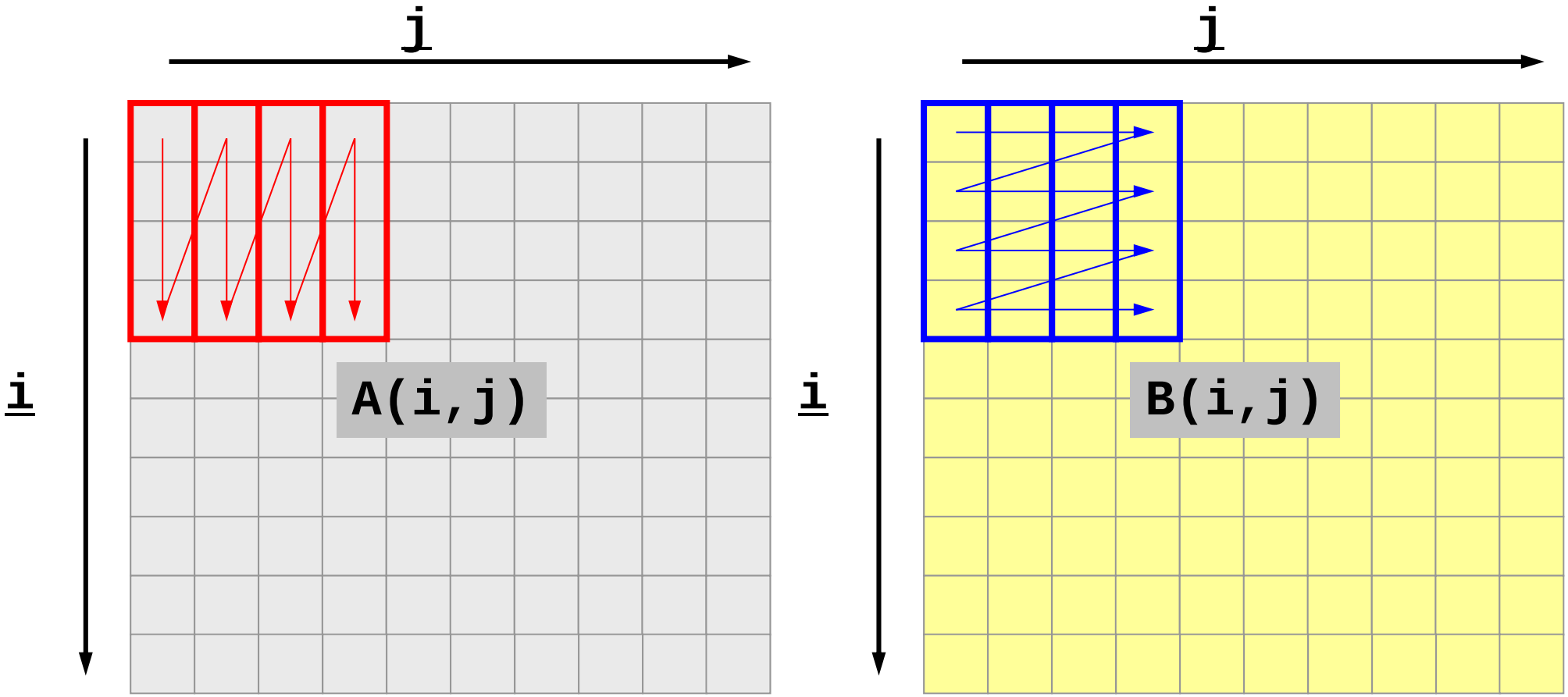
ブロック化によるキャッシュミス削減（5/7）

- 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

```
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

<$O-S3>/2d-2.f

```
$> cd <$O-S3>
$> mpifrtpx -O1 2d-2.f
$> pjsub go1.sh

### N ###      500
BASIC     2.838309E-03
2x2       1.835505E-03
2x2-b     1.538004E-03
### N ###     1000
BASIC     1.167853E-02
2x2       9.495229E-03
2x2-b     9.299729E-03

...

### N ###     4000
BASIC     2.081746E-01
2x2       1.294938E-01
2x2-b     1.317760E-01
### N ###     4500
BASIC     3.203001E-01
2x2       2.335151E-01
2x2-b     2.354205E-01
### N ###     5000
BASIC     3.517879E-01
2x2       2.478577E-01
2x2-b     2.492195E-01
```

-Kfastにすると，高速化，差は無くなる

- 2×2ブロック

```
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

<$O-S3>/2d-2.f

```
$> cd <$O-S3>
$> mpifrtpx -Kfast 2d-2.f
$> pjsub go1.sh

### N ###      500
BASIC     1.870605E-03
2x2       1.602004E-03
2x2-b     1.579705E-03
### N ###     1000
BASIC     7.570124E-03
2x2       7.771923E-03
2x2-b     9.585428E-03

...

### N ###     4000
BASIC     1.276466E-01
2x2       1.236477E-01
2x2-b     1.216945E-01
### N ###     4500
BASIC     1.625757E-01
2x2       1.761032E-01
2x2-b     1.732303E-01
### N ###     5000
BASIC     2.037693E-01
2x2       1.944027E-01
2x2-b     1.913424E-01
```

チューニング：まとめ

- スカラープロセッサ
- 密行列：BLAS

- 本講義で主として扱う疎行列の場合はもっと難しい（研究途上の課題）
  - しかし，基本的な考え方は変わらない
  - メモリアクセスの効率化

Scalar

疎行列と密行列

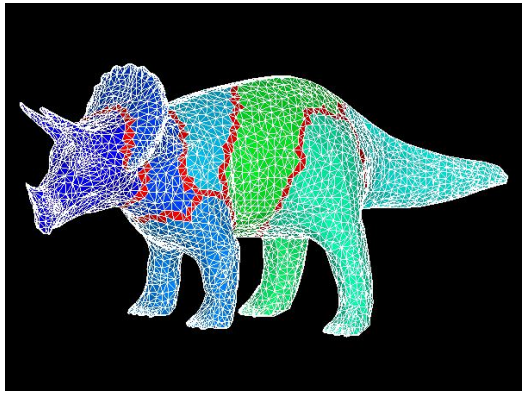
```
do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```

```
do j= 1, N
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

- 右辺のX
  - 密行列：メモリ上で連続しているのでキャッシュを上手く使える
  - 疎行列：メモリ上で連続している保証が無いので，そのたびにメモリアクセスが必要になる場合がある
    - 疎行列計算の性能はメモリの性能が支配：memory-bound
- 疎行列における最も有効な対策：ブロック化
  - 並び替え：ブロック性の抽出
  - 物理的特徴の利用：1要素，1節点に複数自由度

チューニング：まとめのちょっと余談

- 疎行列
  - データの並び方によってチューニングの戦略が変わる
  - データの並び方によってプログラムが変わる場合もある
- 密行列
  - 規則性
  - マシンパラメータ，問題サイズで性能は決まる
    - 他にもコンパイルオプションの影響などもある
    - 自動化（自動チューニング）の余地がある
  - ライブラリ
    - ATLAS（自動チューニング）
      - http://math-atlas.sourceforge.net/
    - GoToBLAS（手動チューニング）
      - 後藤和茂氏（現Microsoft）



- 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
- チューニング例：スカラープロセッサ
- **メモリ性能**

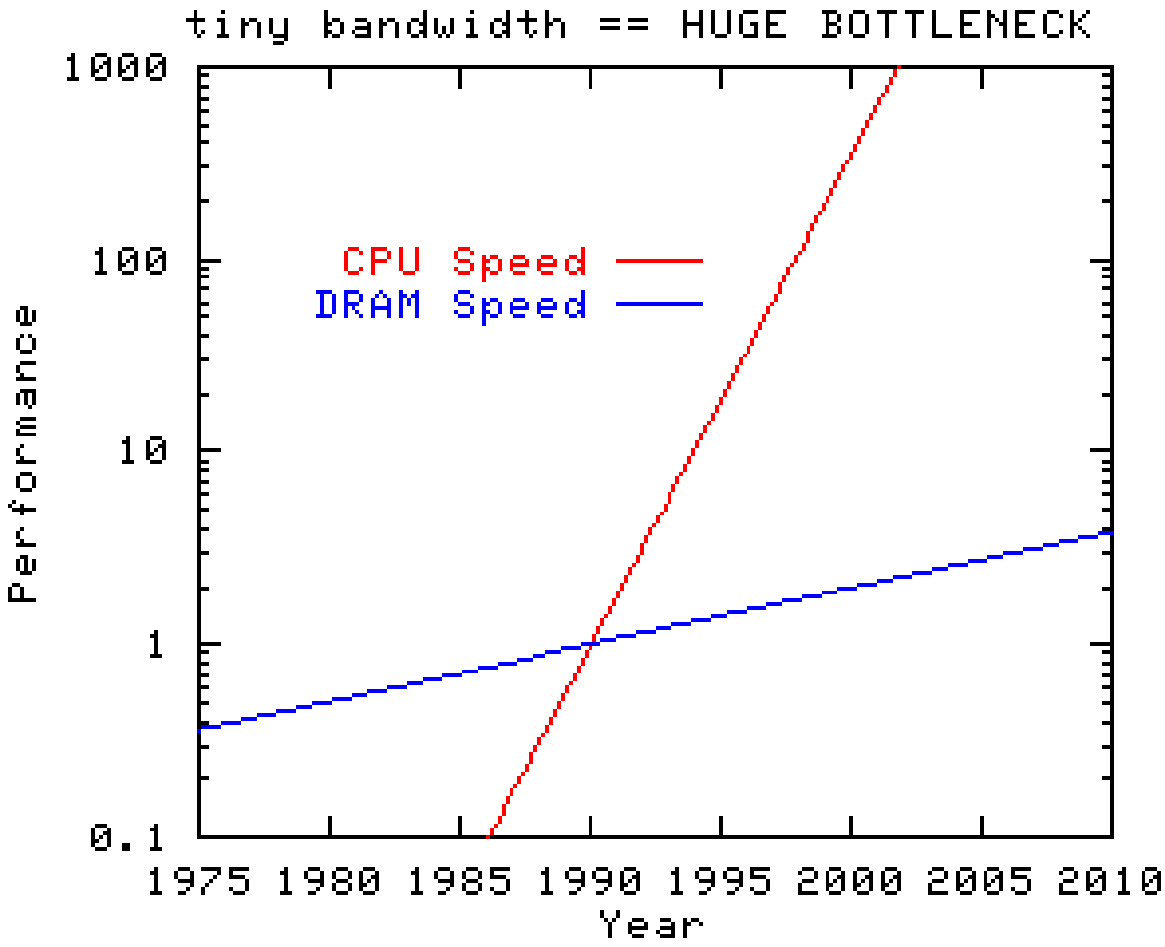
STREAM ベンチマーク

http://www.streambench.org/

- メモリバンド幅を測定するベンチマーク
  - Copy: c(i)= a(i)
  - Scale: c(i)= s*b(i)
  - Add: c(i)= a(i) + b(i)
  - Triad: c(i)= a(i) + s*b(i)

```
-------------------------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
-------------------------------------------------------------
 -------------------------------------------------------------
 STREAM Version $Revision: 5.6 $
 -------------------------------------------------------------
 Array size =   80000512
 Offset     =          0
 The total memory requirement is 1831 MB
 You are running each test  10 times
 --
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
 -------------------------------------------------------------
 -------------------------------------------------------------
Function     Rate (MB/s)  Avg time   Min time  Max time
Copy:      60139.1643      0.0213      0.0213      0.0213
Scale:     59975.9088      0.0214      0.0213      0.0214
Add:       64677.4224      0.0297      0.0297      0.0297
Triad:     64808.5886      0.0297      0.0296      0.0297
```

マイクロプロセッサの動向
CPU性能，メモリバンド幅のギャップ



http://www.streambench.org/

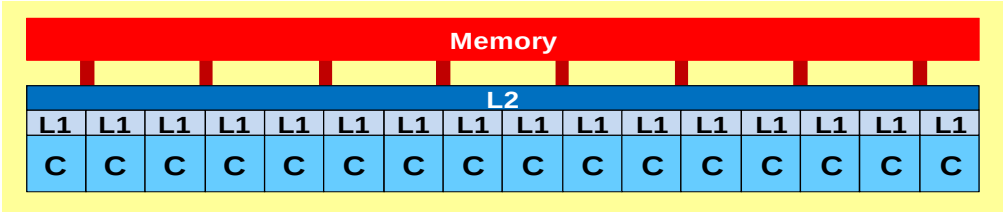
実行：OpenMPバージョン

```
>$ cd <$O-fem2>/mpi/S3/stream
>$ pjsub go.sh
```

go.sh

```
#!/bin/sh
#PJM -L "rscgrp=lecture"
#PJM -L "node=1"
#PJM -L "elapse=10:00"
#PJM -j

export PATH=…
export LD_LIBRARY_PATH=…
export PARALLEL=16
export OMP_NUM_THREADS=16              スレッド数（1-16）

./stream.out > 16-01.lst 2>&1          出力ファイル名
```

Triadの結果 <$O-S3>/stream/*.lst



ピーク性能は85.3 GB/sec.,
75%程度

Thread #	MB/sec.	Speed-up
1	8606.14	1.00
2	16918.81	1.97
4	34170.72	3.97
8	59505.92	6.91
16	64714.32	7.52

実　習

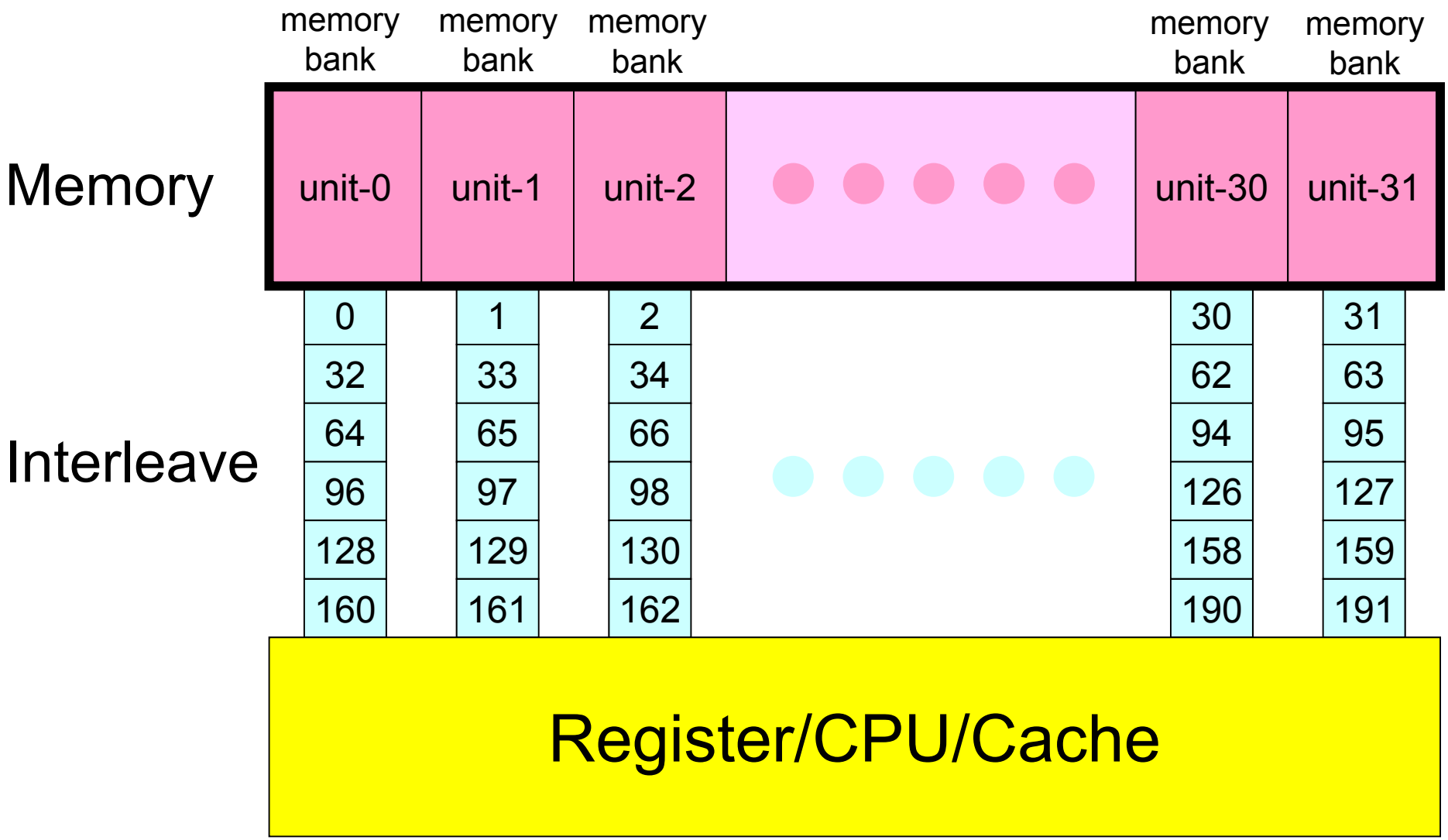
- 実際にやってみよ
- スレッド数を変える
- 1CPU版，MPI版もある
  - Fortran，C
  - STREAMのサイトを参照

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，同時に同じバンク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。（次ページ）

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
```

- 2^nを避ける（32だけがダメというわけではない）

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
- 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」
- ベクトルプロセッサ（本講義では省略）
  - ループ長を大きくとる。
- 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。
- Oakleaf-FXのプロファイラ（基本・詳細）：利用者ポータル