ハイパフォーマンスコンピューティング(2)

HPCとは

- HPC（High-Performance Computing）
  - 非常に計算量が多い計算処理
  - 主な用途として、地球全体の気象など、人間の手で制御することができない現象や、自動車の衝突シミュレーション等の現象の解析
  - HPCでは、スーパーコンピュータ、ワークステーション、PCクラスタ、GPGPUなどを使うのが一般的

⇒「TOP500」は世界中の高性能な
コンピュータシステム上位500機のリスト。

1秒間に1京（10^{16}）回

2012／6更新

Rank	System	Vendor	Total Cores	Rmax (TFlops)	Rpeak (TFlops)	電力 (kW)
1	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	IBM	1,572,864	16,325	20,133	7,890
2	K computer（京）, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	富士通	705,024	10,510	11,280	12,660

※TFLOPS：「TERA Floating-Point Operations Per Second」の略
⇒1秒間に1兆（10^{12}）回の浮動小数点演算

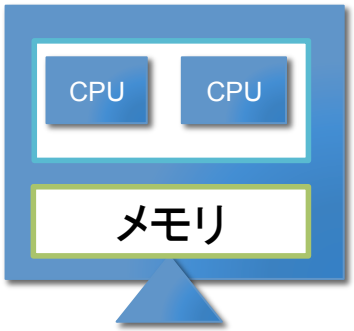
高速化の方法

- アルゴリズム

- プログラムの実装方法
  - キャッシュ
  - メモリアクセス
  - ループアンローリング

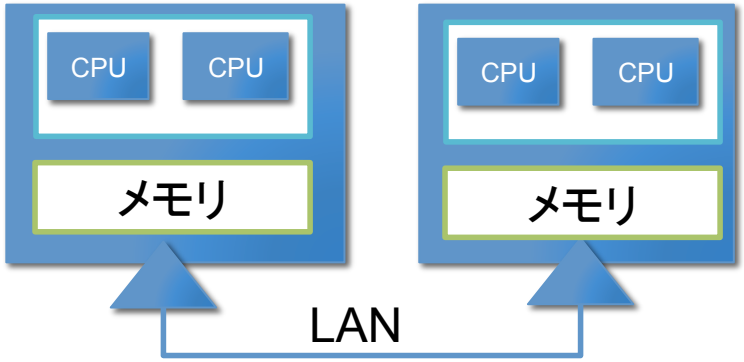
- 並列化
  - 共有メモリ型
  - 分散メモリ型

・共有メモリ型



共有メモリ型：1台のマシンに複数CPUやコアがあるマシンを利用。

・分散メモリ型



ネットワークでつながれた複数のマシンを利用して計算。

メモリアクセスの例

- 試しにMatlabを用いてメモリへのアクセス方法の違いによってどれくらい計算速度が変わるか調べてみよう。

- 配列データのコピーを行ごと（row-wise）及び列ごと（column-wise）に実行する関数mem1.m（次のページ）を作成し、下記のようにコマンドウィンドウで実行してみよう。

```
>> n=1000; loop = 10; mem1(n,loop)
```

mem1.m

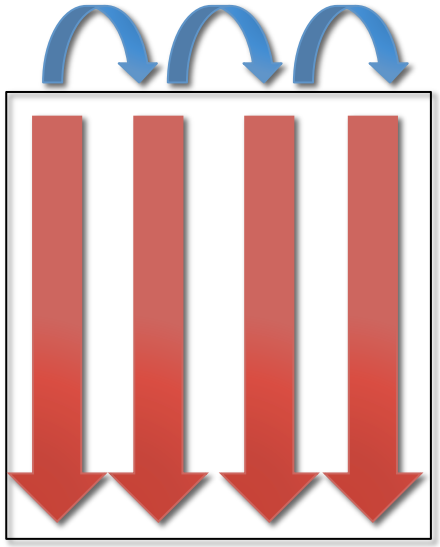
```
function mem1(n,maxloop)
A = randn(n); % random matrix
B = zeros(n);

disp('row-wise')
tic
for loop=1:maxloop
    for i=1:n
        B(i,:) = A(i,:);
    end
end
toc

disp('column-wise')
tic
for loop=1:maxloop
    for j=1:n
        B(:,j) = A(:,j);
    end
end
toc
```

何が違うのか？

- メモリ上のデータは連続的にアクセスするほうが速い。
- Matlabの場合、多次元配列については、最も内側のインデックスが連続になる（Fortranと同じ。C言語は逆で、最も外側が連続）。

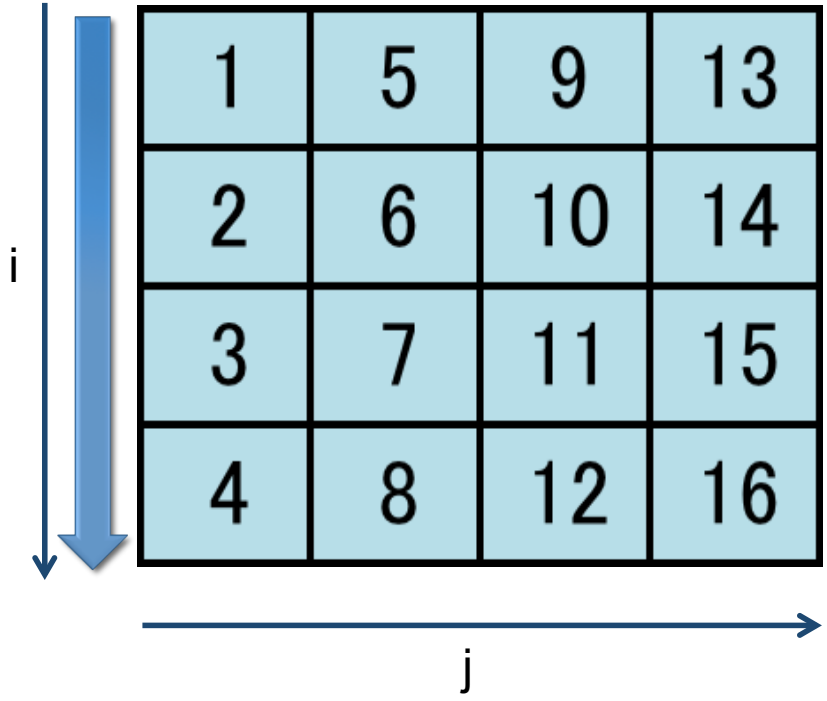


- column-wiseのほうは、連続的なアクセスとなる。
- row-wiseのほうは、非連続。

メインメモリでの配列の格納方式

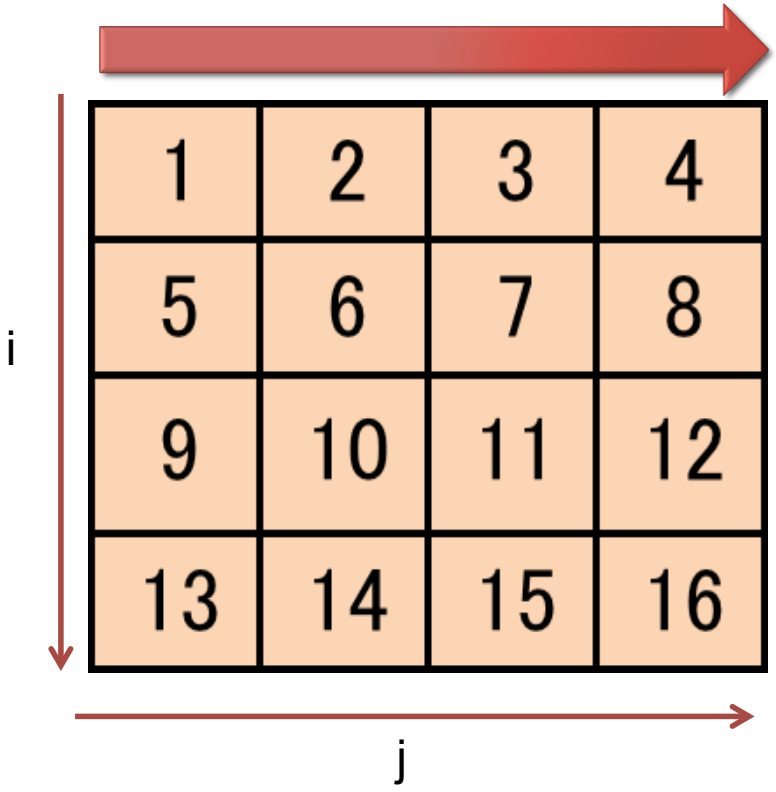
・MATLABの場合（Fortranも同じ）

A(i,j)



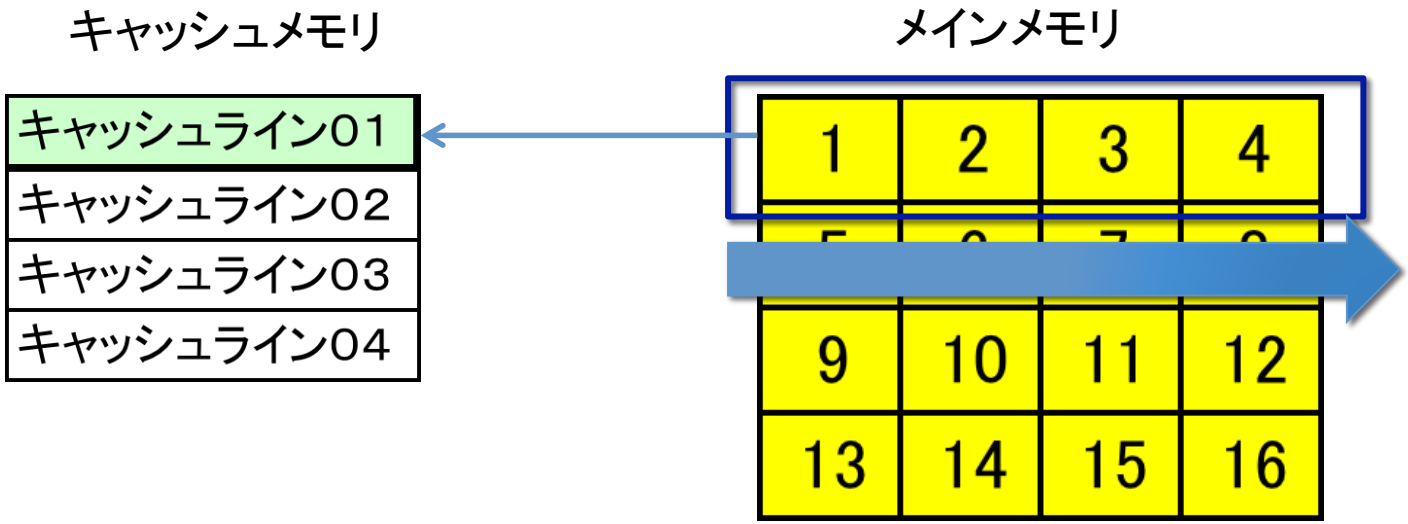
・C言語の場合

A[i][j]



格納方向に従い、アクセスする場合

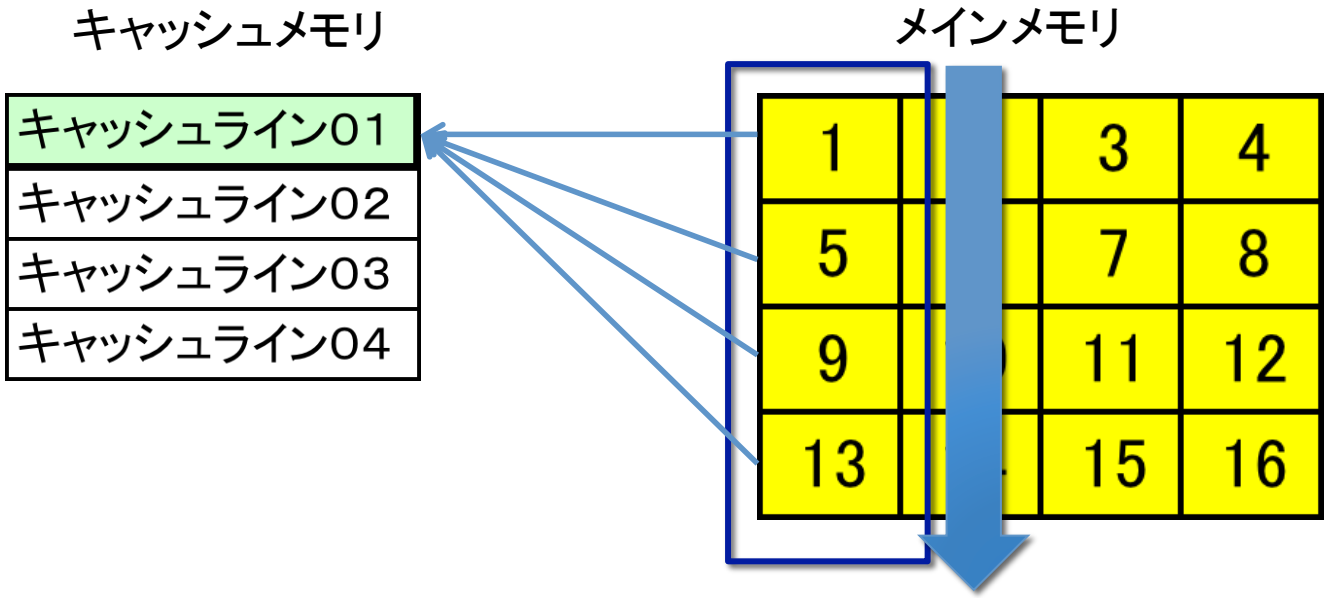
・格納方向に従い、連続アクセスするため、データアクセス時間の短縮され、効率性が高くなる。



格納方向と逆方向にアクセスした場合

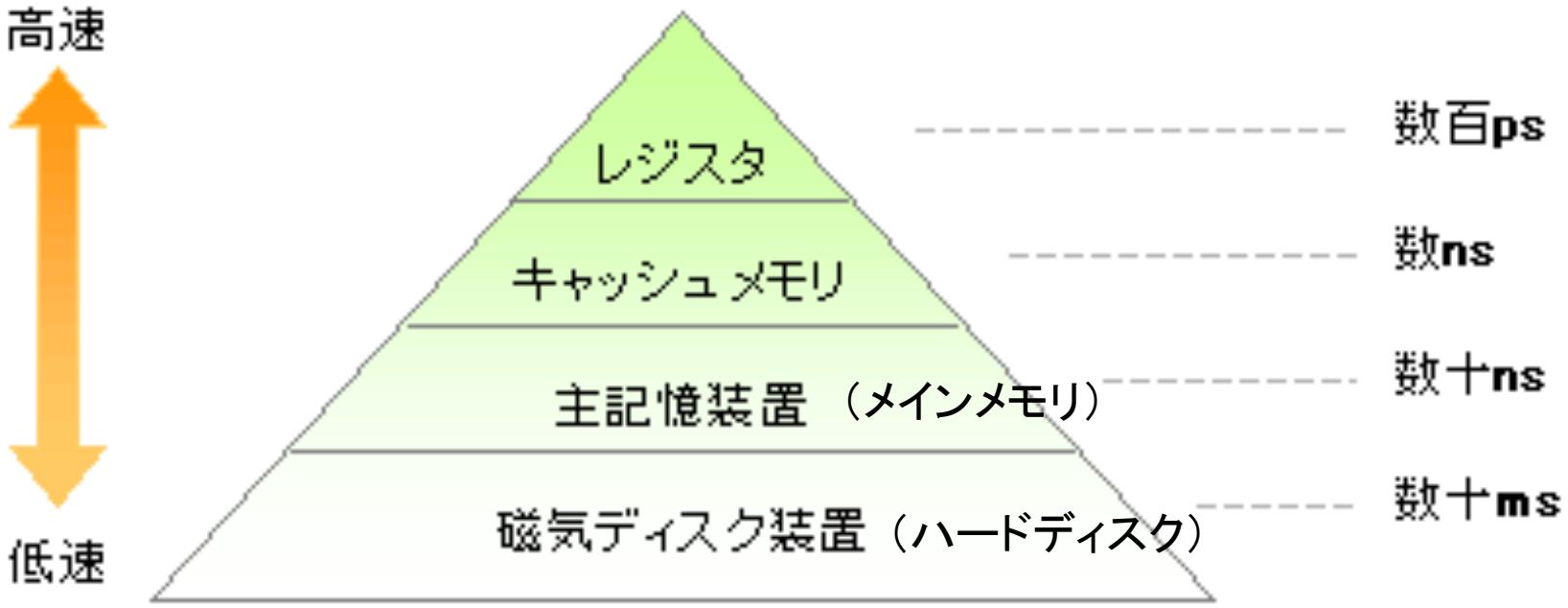
・メモリから逐次、データを読み出し、キャッシュを更新する必要があるため、効率が悪い。

⇒高速に処理できない



【参考】メモリ階層構造

・アクセス時間は、CPUの中心部に向かうほど速く、CPUから離れていくほど遅くなります。

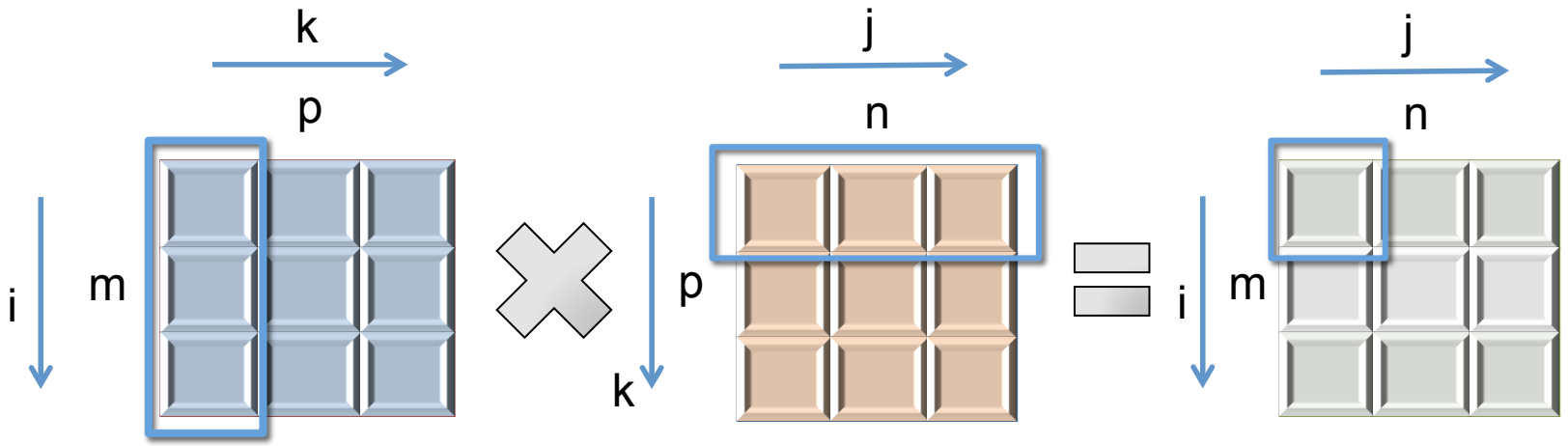


http://www.jitec.jp/

【実習】行列積（C言語 or Fortran）

- C言語やFortranで行列積を計算してみよう。

⇒アクセスの方向（i,j,kの組み合わせで全6パターン）別に3重ループを作成し、メモリ連続アクセスとなる場合と、不連続となる場合の性能と処理速度を確認する。



【実習】プログラム作成のポイント

- 行列A,Bは乱数を使用：#include<stdlib.h>，rand()
- 時間の計測：#include<time.h>，clock()を使用する
- 処理速度(Flops)：1秒間に何回の浮動小数点数演算できるかの指標(floating-point operations per second)

⇒行列積: 計算量は$2n^3$ flopsなので、MFlops=$2n^3$/時間/10^6(M)とする

＜実行例＞
Input n　1000
(i,j,k) Time(sec)=sss, MFlops=○○○
(i,k,j) Time(sec)=sss, MFlops=○○○
(j,k,i) Time(sec)=sss, MFlops=○○○
(j,i,k) Time(sec)=sss, MFlops=○○○
(k,i,j) Time(sec)=sss, MFlops=○○○
(k,j,i) Time(sec)=sss, MFlops=○○○

コンパイルオプション

- コンパイル時に最適化オプションを付けることによって、計算処理性能を向上させることができる。
- gccの場合（他の多くのコンパイラでも同様）

  gcc –O2 xxx.c
  - O0, O1, O2, O3の順に最適化レベルが上がる（デフォルトは、O0）。
- MS Visual C++の場合

  「ビルド」→「構成マネージャ」を選択。「アクティブ ソリューション構成」で「Release」を選ぶ。デフォルトは「Debug」。

- コンパイラによっては過剰な最適化（計算順序の変更・計算の省略）を行い、最適化を行わない場合と計算結果が異なる場合もあるので、注意が必要。過剰な最適化を抑制するオプションもある。

【実習】行列積プログラムへの適用(1)

- 既に作成した行列積のプログラムをコンパイルするときに、最適化オプションを変更して計算時間の変化を確認しよう。

- 最適化オプション O1, O2, O3 の違いは何かを man などで調べてみよう。

- 他にも、コンパイラやアーキテクチャ(CPUなど)に固有のコンパイラオプションがあるので、色々と調べて試してみよう。

ループアンローリング（ループの展開）

- ループを展開して、１回のループで複数回分の処理をすると、ループのオーバーヘッド（条件判定など）の影響が小さくなり、処理速度が向上する場合がある。
- 展開の数を増やしていくと性能は向上していくが、上限はある。
- １回のループでループカウンタ（i, jなど）が展開の数だけ飛ぶので、端数処理をする必要がある。

```
/* アンローリングなし */
s = 0.0;
for (i = 0; i < n; i++) {
    s += a[i];
}
```

```
/* アンローリングあり（2回） */
s1 = 0.0;  s2 = 0.0;
for (i = 0; i <= n - 2; i += 2) {
    s1 += a[i];  s2 += a[i+1];
}
s = s1 + s2;
for ( ; i < n; i++) { /* 端数処理 */
    s += a[i];
}
```

【実習】アンローリングの例

```
#include <stdio.h>

int main(void)
{
  int i, n;

  printf("n = \n");
  scanf("%d", &n);

  printf("unrolling = 2\n");
  for (i = 0; i <= n - 2; i += 2) {
    printf("  i = %d\n", i);
    printf("i+1 = %d\n", i + 1);
  }
  printf("reminder\n");
  for ( ; i < n; i++) {
    printf("  i = %d\n", i);
  }
  /* 右へつづく */
```

```
  /* つづき */

  printf("unrolling = 4\n");
  for (i = 0; i <= n - 4; i += 4) {
    printf("  i = %d\n", i);
    printf("i+1 = %d\n", i + 1);
    printf("i+2 = %d\n", i + 2);
    printf("i+3 = %d\n", i + 3);
  }
  printf("reminder\n");
  for ( ; i < n; i++) {
    printf("  i = %d\n", i);
  }

  return 0;
}
```

【実習】行列積プログラムへの適用(2)

- 既に作成した行列積のプログラムにループアンローリングを適用してみよう。
- 6パターンすべてに適用するのは大変なので、6パターンの内、一番高速だったものについて適用しよう。
- 最適化オプションによっては自動的にアンローリングを行うものもあるので、はじめは最適化オプションは付けないで試してみよう。
- 展開の数を、2, 4, 8のように増やしていき、性能向上の上限を探そう。
- 最適化オプションをつけたときにどうなるかも見てみよう。