応用数理概論

準備

端末上で

```
cd ~/
mkdir cppwork
cd cppwork
wget http://271.jp/gairon/main.cpp
wget http://271.jp/gairon/matrix.hpp
```

とコマンドを記入.

```
ls
```

とコマンドをうち，main.cppとmatrix.hppがダウンロードされていることを確認.

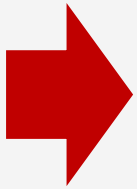
準備

コンパイル

```
c++ -I. -std=c++0x -O3 main.cpp
```

実行

```
./a.out
```



```
2  1  1  1  1
1  1  1  1  1
1  1  1  1  1
1  1  1  1  1
1  1  1  1  1
```

と表示されれば成功!

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

行列A, B, Cを作成

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

時間計測用のオブジェクト

t.tic(); // 時間計測開始
プログラム
t.toc();//時間計測終了

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

Arow：行列Aの行サイズ
Acolumn :行列Aの列サイズ

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

Brow : 行列Bの行サイズ
Bcolumn :行列Bの列サイズ

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

全成分1のArow×Acolumnサイズの行列Aを作成.

今はこんな感じ...

Acolumn

Arow

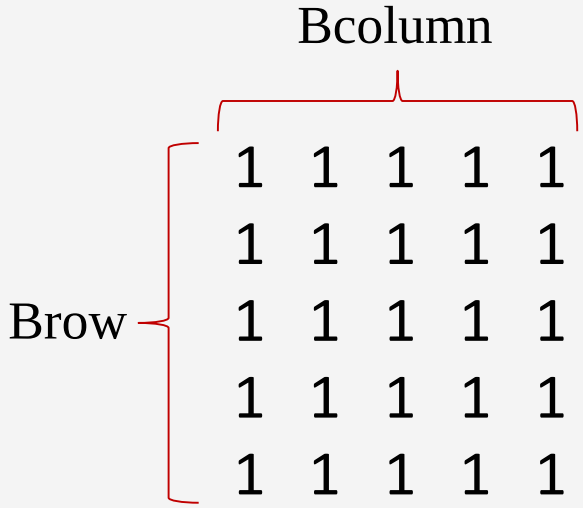
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

全成分1のBrow×Bcolumnサイズの行列Bを作成.

今はこんな感じ...



matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

全成分0のArow×Bcolumnサイズの行列Cを作成.

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

行列の要素へのアクセス

今はこんな感じ...

行列Aの(0,0)成分に2を代入

```
2 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

matrixクラスの概説

```
matrix< double > A, B, C;
Time t;
int Arow = 5, Acolumn = 5;
int Brow = Acolumn, Bcolumn = 5;

A.ones(Arow,Acolumn);
B.ones(Brow,Bcolumn);
C.zeros(Arow,Bcolumn);

A(0, 0) = 2.0;

std::cout << A << std::endl;
```

行列Aを出力

行列サイズを大きく指定したら
コメントアウトをしましょう
例
```
//std::cout << A << std::endl;
```

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){

                }
        }
}
t.toc();
```

t.tic();
t.toc();
で行列積の時間計測

ここに必要なプログラムを記入してみよう!!
(答えは次のスライドにあるので,
わからない場合は確認を…)

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
```

実行してみよう!!

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
```

for文を入れ替えた行列積を追加しよう!!

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
C.zeros(Arow,Bcolumn);
t.tic();
for (int k = 0; k < Acolumn; k++){
        for (int j = 0; j < Acolumn; j++){
                for (int i = 0; i < Arow; i++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
```

自作行列積の作成

実行速度を速くするためには…?

1.キャッシュヒット率
2.並列化
3.SIMD拡張命令

自作行列積の作成

実行速度を速くするためには…?

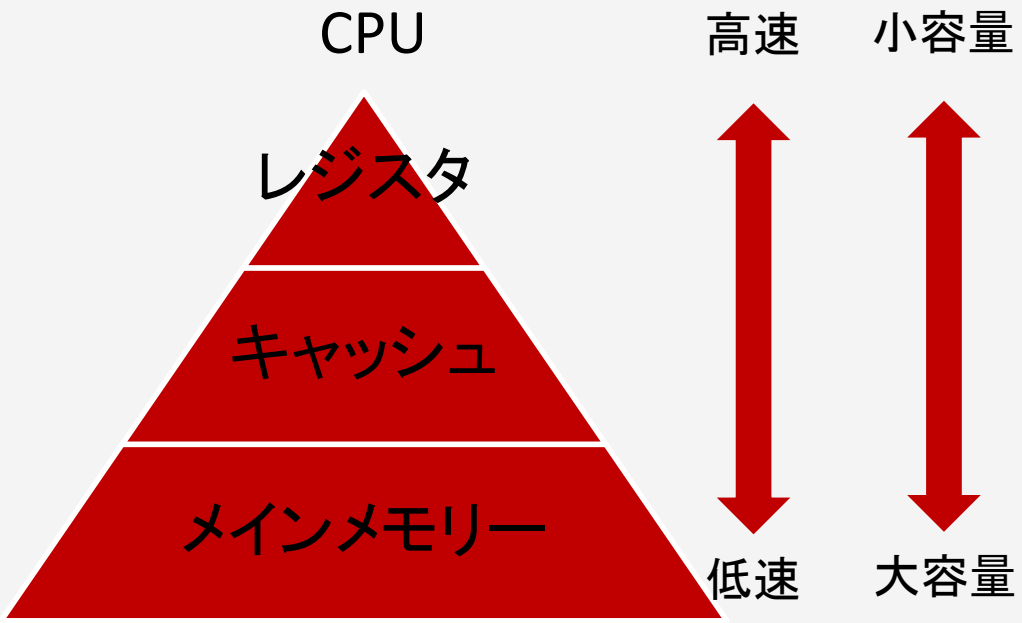
1.キャッシュヒット率

2.並列化

3.SIMD拡張命令

キャッシュヒット

データはメインメモリーに保存されている.
必要に応じてキャッシュやレジスタにデータを転送する.
CPUはレジスタからデータを取り出し，計算する.



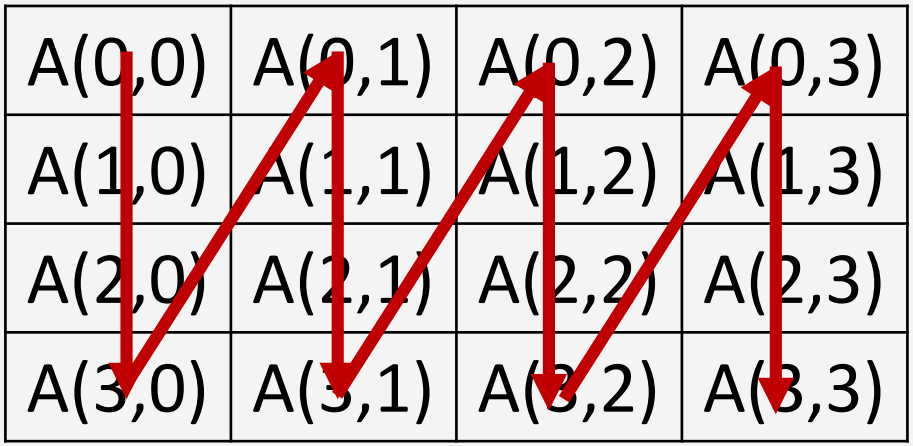
自作行列積の作成

自作matrixはどのように
メインメモリーに格納されている?

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)
A(3,0)	A(3,1)	A(3,2)	A(3,3)

自作行列積の作成

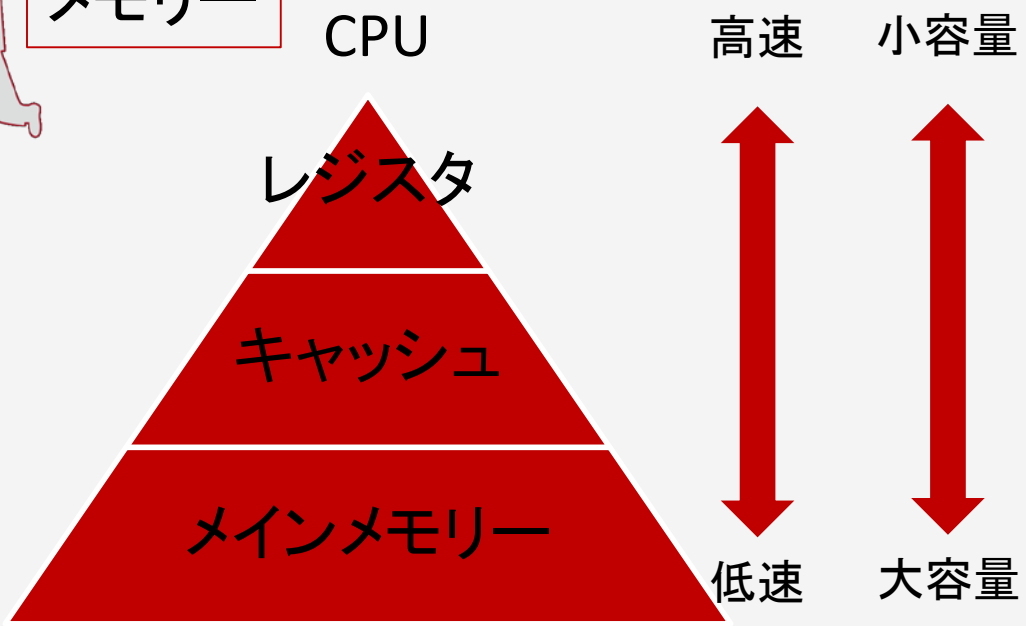
自作matrixはどのように
メインメモリーに格納されている?



矢印の順に格納されている!!

キャッシュヒット

現代のCPUは非常に速度が速いため，メインメモリーとキャッシュのデータの転送がボトルネックになっている…



キャッシュライン

メインメモリーからキャッシュには一度に何Byteかのデータが転送される(キャッシュライン).

例

Intel Core i7-4790

キャッシュライン 64Byte

(倍精度浮動小数点8個分)

キャッシュライン

キャッシュラインが倍精度浮動小数点数4個分ならば…

例 double A[8]

メモリー

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]

キャッシュ

キャッシュライン

キャッシュラインが倍精度浮動小数点数4個分ならば…

例 A[1]を呼び出すと…

メモリー

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]

キャッシュにないので，
メモリーから転送

キャッシュ

A[1]
A[2]
A[3]
A[4]

キャッシュライン

キャッシュラインが倍精度浮動小数点数4個分ならば…

例 続いてA[2]を呼び出すと…

メモリー

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]

キャッシュ

A[1]
A[2]
A[3]
A[4]

キャッシュにあるので，
メモリーから転送しない

再利用できている!!

キャッシュライン

キャッシュラインが倍精度浮動小数点数4個分ならば…

例 続いてA[0]を呼び出すと…

メモリー

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]

キャッシュ

A[1]
A[2]
A[3]
A[4]

キャッシュにないので，
メモリーから転送しない

再利用できていない…

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
C.zeros(Arow,Bcolumn);
t.tic();
for (int k = 0; k < Acolumn; k++){
        for (int j = 0; j < Acolumn; j++){
                for (int i = 0; i < Arow; i++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
```

再利用できていない…

再利用できている!

自作行列積の作成

実行速度を速くするためには…?

1.キャッシュヒット率

2.並列化

3.SIMD拡張命令

コンピュータと並列化

現在のコンピュータは大きく分けて…

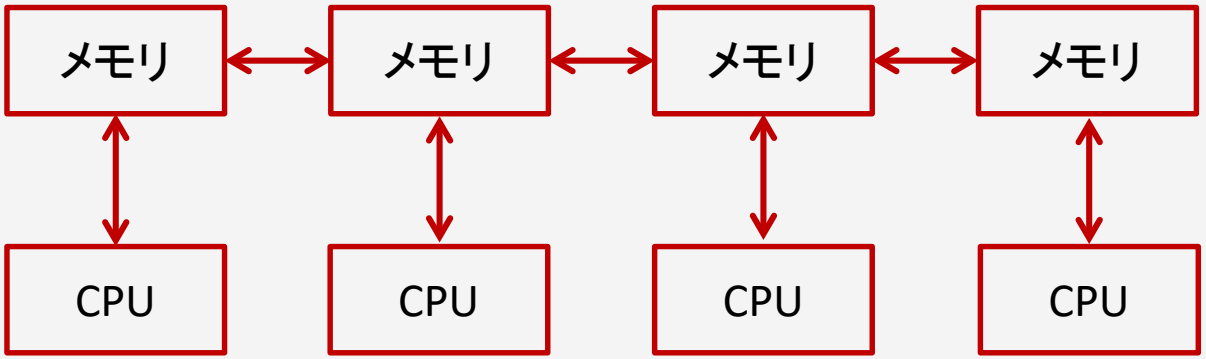
- 分散メモリ型コンピュータ

- 共有メモリ型コンピュータ

コンピュータと並列化

現在のコンピュータは大きく分けて…

・分散メモリ型コンピュータ

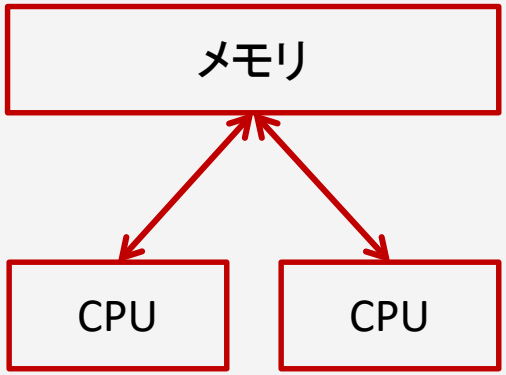


たくさんのPCがあり，PC間でデータ通信を行うイメージ!!

コンピュータと並列化

現在のコンピュータは大きく分けて…

・共有メモリ型コンピュータ



いくつかのCPUで一つのメモリを共有するイメージ!!

コンピュータと並列化

現在，利用しているPCは多くは，共有メモリ型で
複数個のコアやスレッドを持つ!!

例えば
Intel Core i7-6700K コア数 4

コンピュータと並列化

現在，利用しているPCは多くは，共有メモリ型で
複数個のコアやスレッドを持つ!!

例えば
Intel Core i7-6700K コア数 4

しかし，今までのプログラミングでは，
1つのコアしか使われていない...

コアを有効活用するプログラムが必要!!

自作行列積の作成

実行速度を速くするためには…?

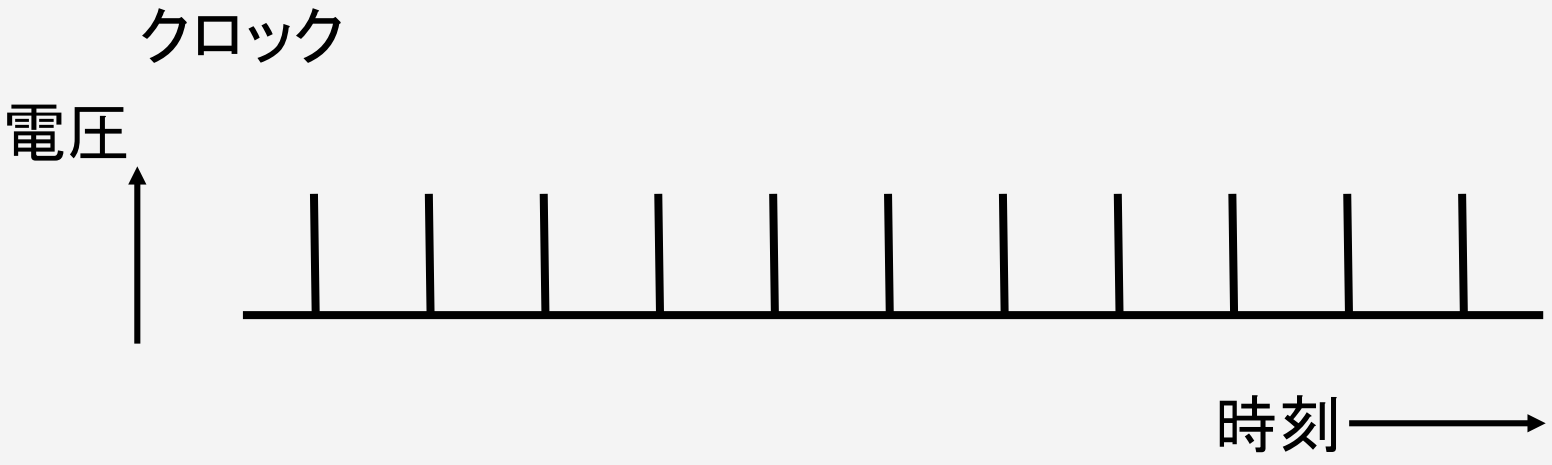
1.キャッシュヒット率

2.並列化

3.SIMD拡張命令

SIMD演算とは

通常のCPUによる演算

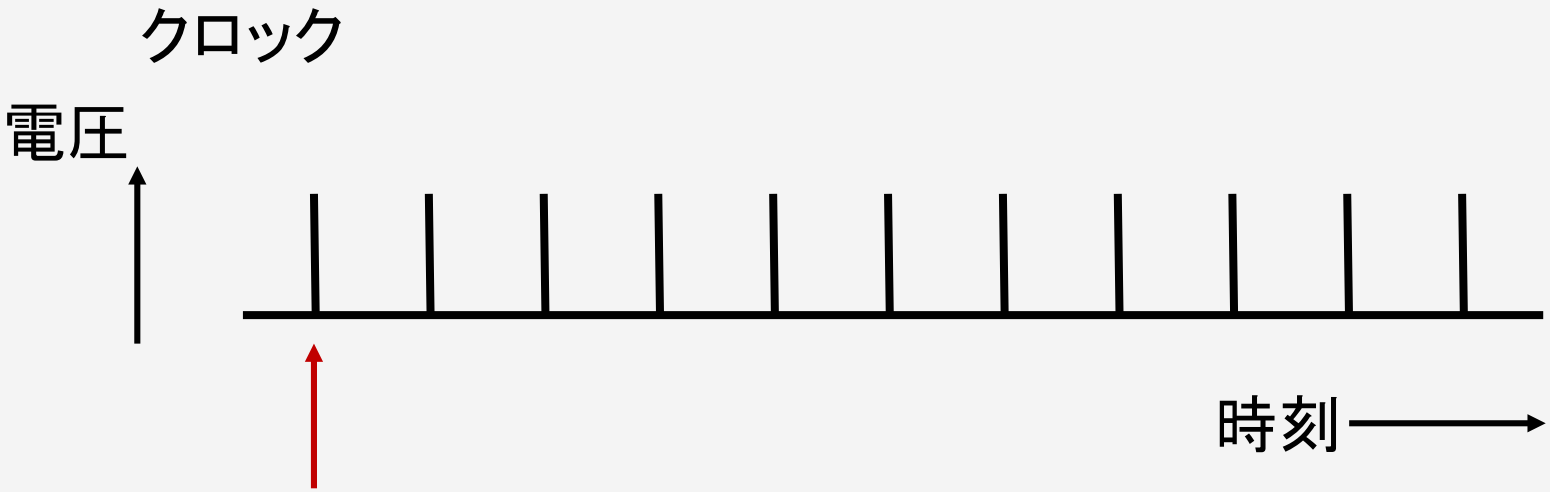


Intel Core i7-6700K 4.0GHz

1秒間に4×10^9回，電圧が上昇する

SIMD演算とは

通常のCPUによる演算



倍精度浮動小数点数(C言語のdouble)
同士の演算を1回行える!!

```
double a=1.0, b=2.0, c;
c = a+b;
```

これが1回の演算

SIMD演算とは

SIMDとは，Single instruction multiple dataの略
多数のデータを1つの命令(クロック)で処理する!!

例えばIntel社製CPUでは…

- Sandy Bridge 世代以降(2000番台) 2011年発売
  AVX(Intel Advanced Vector Extensions)と呼ばれるSIMD拡張命令があり，浮動小数点数をサポート．256bit対応．
- Haswell世代以降(4000番台) 2013年発売
  AVX2と呼ばれるSIMD拡張命令があり，整数型をサポート．さらに浮動小数点数の積和演算をサポート．256bit対応．

(※AVX以前はSSE4と呼ばれるSIMD拡張命令があり，128bit対応)

SIMD演算とは

SIMDとは，Single instruction multiple dataの略
多数のデータを1つの命令(クロック)で処理する!!

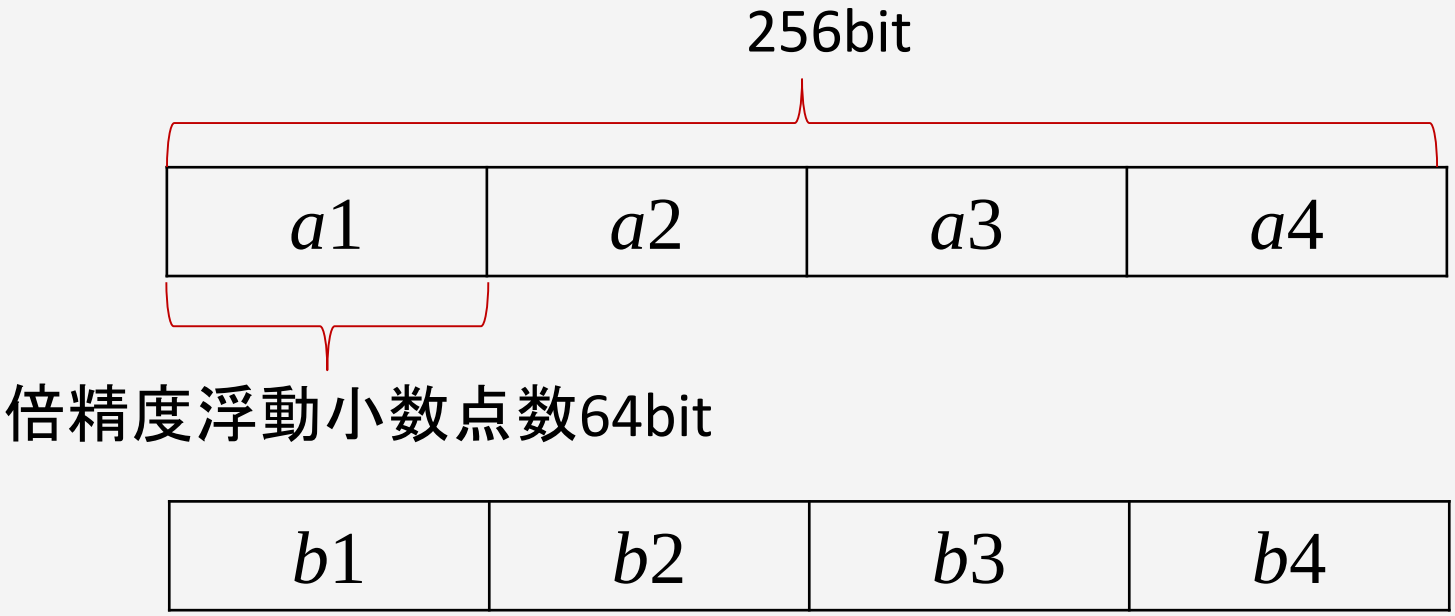
例えばIntel社製CPUでは…

AVX及びAVX2は256bitのレジスタを持つため
倍精度浮動小数点数(64bit)を4つ格納できる!!

SIMD演算とは

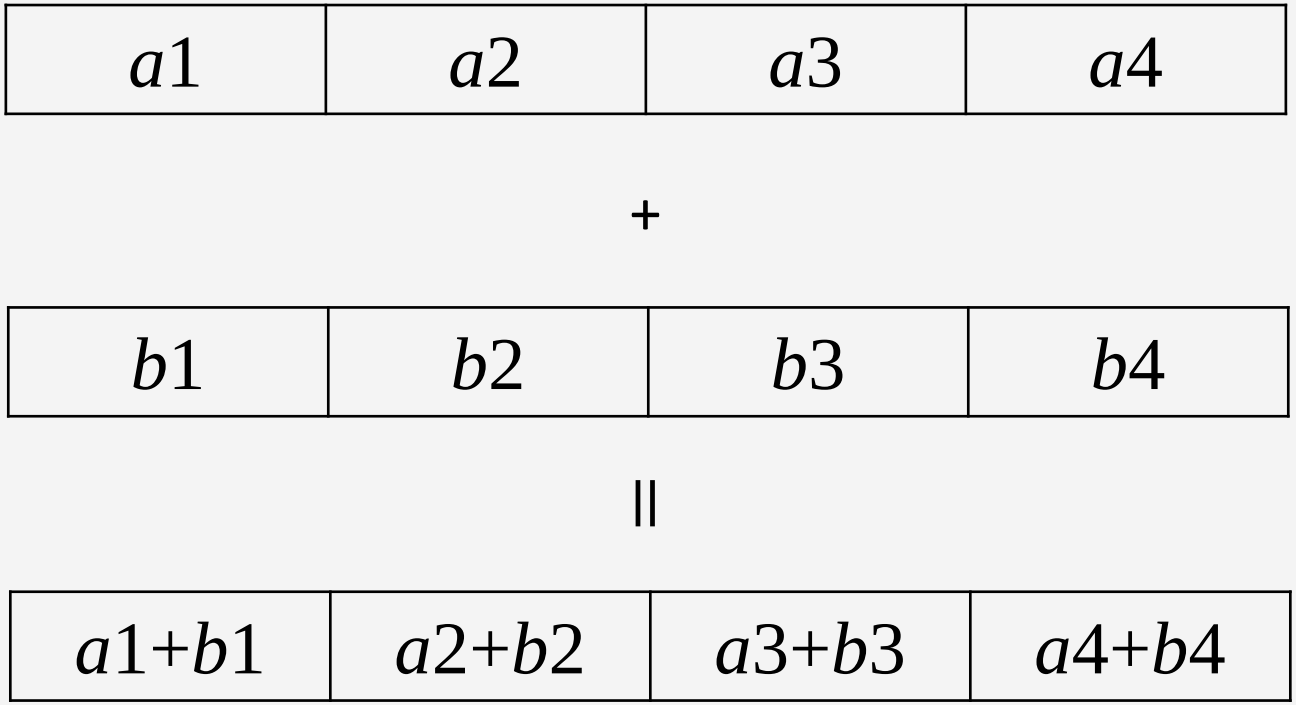
SIMDとは，Single instruction multiple dataの略

多数のデータを1つの命令(クロック)で処理する!!



SIMD演算とは

SIMDとは，Single instruction multiple dataの略
多数のデータを1つの命令(クロック)で処理する!!



SIMD演算とは

SIMDとは，Single instruction multiple dataの略
多数のデータを1つの命令(クロック)で処理する!!

$a1$	$a2$	$a3$	$a4$

本来は4クロック必要な演算が
1クロックで処理可能

||

$a1+b1$	$a2+b2$	$a3+b3$	$a4+b4$

SIMD演算とは

SIMDとは，Single instruction multiple dataの略
多数のデータを1つの命令(クロック)で処理する!!

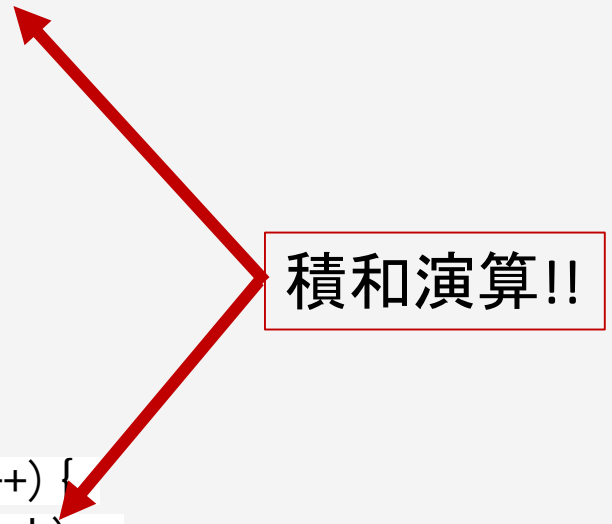
AVX2では積和演算(FMA : Fused Multiply-Add)

$a1+b1\times c1$	=	$a1$	+	$b1$	×	$c1$
$a2+b2\times c2$		$a2$		$b2$		$c2$
$a3+b3\times c3$		$a3$		$b3$		$c3$
$a4+b4\times c4$		$a4$		$b4$		$c4$

8クロック必要な演算が
1クロックで処理可能

自作行列積の作成

```
t.tic();
for (int i = 0; i < Arow; i++){
        for (int j = 0; j < Acolumn; j++){
                for (int k = 0; k < Acolumn; k++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
C.zeros(Arow,Bcolumn);
t.tic();
for (int k = 0; k < Acolumn; k++){
        for (int j = 0; j < Acolumn; j++){
                for (int i = 0; i < Arow; i++){
                        C(i,k) += A(i,j)*B(j,k);
                }
        }
}
t.toc();
```



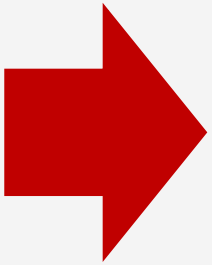
自作行列積の作成

実行速度を速くするためには…?

1.キャッシュヒット率

2.並列化

3.SIMD拡張命令



高速な行列積を自作してみましょう!!
…とはいいません.

プロが作ったツールを使いましょう!!

BLASとは

BLAS :

Basic Linear Algebra Subprogramsの略.

ベクトルや行列に関する演算に関する関数,

サブルーチンが組み込まれている.

例

dgemm('n','n',an,bm,am,alpha,A,an,B,am,beta,C,an)

行列積のサブルーチン

C = alpha*A*B + beta*C

BLASとは

BLASは様々な人，企業が開発している:

- Reference BLAS

  基準として作られたBLAS(無料)．速くない．

- Intel MKL

  Intel社が開発(有料)．

  CPU毎に設計されて非常に速い．

- OpenBLAS

  後藤和茂先生が作成したGotoBLASが

  引き継がれたBLAS．速い(無料)．

- ATLAS (Automatically Tuned Linear Algebra Software)

  自動でチューニングするBLAS．CPUに依存しない．

BLASとは

BLAS :

関数名，サブルーチン名や引数，役割が同じであるため，どのBLASを使っても動作は同じ．
しかし，BLASによって速度が変わる!!

① 計算時間がかかるところをBLASで作成

② 使用者がBLASを選択し，実行する

コンピュータ毎に最適な選択し，高速に!

BLASとは

BLASは演算ごとにレベル分けされている:

Level 1:

ベクトル-ベクトルの演算

Level 2:

行列-ベクトルの演算

Level 3:

行列-行列の演算

Lapackとは

Lapack:

Linear Algebra PACKageの略

線形代数ライブラリ.

連立一次方程式や固有値問題などが解ける.

例えば

dgesv(An,Bn,An,ipiv,b,bn,info)

連立一次方程式のサブルーチン

$$Ax = b$$

の解xをbに代入して出力される.

Lapackとは

Lapack:

LapackはFortranで記述されており，内部でBLASを用いているためBLASを差し替えることでCPUに依存した最適化が可能!

世界中で利用されており，信頼性も高い!!

現在はバージョン3.5.0

Lapackのリファレンス:

http://www.netlib.org/lapack/

Lapackとは

Lapackは変数の型ごとに関数，サブルーチン名が変わる:

ge : 一般行列

gb :一般帯行列

tr : 三角行列

など…

例 :

dgesv (引き数)

dgesv, dgbsv

連立一次方程式

☆文法 ?gesv(n, nrhs, A, lda, ipiv, b, ldb, info)

n : integer型. Aの次元(n×n), bの行数.

nrhs : integer型. bの列数.

A : ?型のn×n配列.

lda : max(1,n)

b : ?型のn×nrhs配列

ipiv : integer型. n次元の配列.

info : integer型.

info = 0なら正常

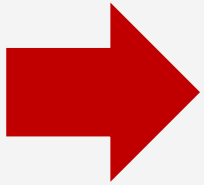
info > 0 正則でない可能性

info < 0 info番目の値が不正

Matlab

BLASやLapackなどのライブラリを使えば世界最高速のツールが使えるが…

もっと手軽に世界最高のツールを利用したい!!



Matlabを利用する!!

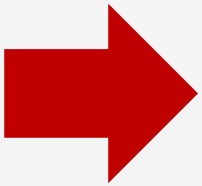
MatlabはIntel MKLをはじめとした世界の
様々なライブラリを搭載したプログラミング言語.
そのため，世界最高峰の行列積や連立一次方程式
の近似解法が簡単に使用可能!!

Matlab

端末上で

matlab

とコマンドをうつ.



Matlabが起動

```
>> n = 5
>> A = ones(n,n);
>> B = ones(n,n);
>> C = A*B;
```

とMatlabコマンドをうつ.

Matlab

端末上で

matlab

とコマンドをうつ.

Matlabが起動

```
>> n = 5
>> A = ones(n, n);
>> B = ones(n, n);
>> C = A*B;
```

nに5を代入

全要素1のn×n
行列AとBを作成

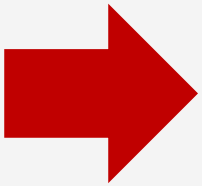
セミコロン(;)があ
ると非表示になる.

Matlab

端末上で

matlab

とコマンドをうつ.



Matlabが起動

```
>> n = 5
>> A = ones(n,n);
>> B = ones(n,n);
>> C = A*B;
```

A×Bの行列積を実行.
Intel MKL のBLAS
dgemmが呼ばれる!!

Matlab

```
>> n = 1000, A = ones(n,n); B = ones(n,n);
>> tic; C = A*B; toc
```

Matlab

表示する場合はカンマで区切る

```
>> n = 1000, A = ones(n,n); B = ones(n,n);
>> tic; C = A*B; toc
```

一行で書くことも可

実行時間の計測

自作行列積と実行時間を比べよう!

Matlab

```
>> n = 10, A = rand(n,n); b = A*ones(n,1);
>> tic; x = A¥b, toc
```

Matlab

n×nの乱数行列Aを作成

```
>> n = 10, A = rand(n,n); b = A*ones(n,1);
>> tic; x = A¥b, toc
```

Aと全要素1のベクトルの行列ベクトル積

連立一次方程式Ax = bを満たすxを求める.
Lapackの連立一次方程式を求める関数が
呼び出される.
答えはb = A*ones(n,1)であるため,
大体x=ones(n,1)になる.

Matlab

```
>> n = 100, A = rand(n,n); A = A + A' ;
>> tic; lambda = eig(A), toc
```

Matlab

A'はAの転置行列. A+A'は対称行列になる

```
>> n = 100, A = rand(n,n); A = A + A' ;
>> tic; lambda = eig(A), toc
```

行列Aの固有値問題A*x = lambda*xを満たす固有値lambdaを求める関数.
これもLapackの固有値問題の関数が呼ばれている(対称行列用).
Aが対称行列のため, 固有値はすべて実数になる.

おまけ:精度保証付き数値計算

今まで利用してきた浮動小数点数とは簡単に言うと
超高速な近似計算である. そのため

```
>> n = 3000, A = rand(n,n); b =A*ones(n,1);
>> tic; x = A¥b; toc
```

を計算した場合, 実に10000000000回以上の近似計算を行う. このとき, 正しい結果は得られているのであろうか?

```
>> format long
>> x(1)
```

とみると大体1に近いためあっていることが推測される.

おまけ:精度保証付き数値計算

```
>> n = 3000, A = randmat(n,10^15); b =A*ones(n,1);
>> tic; x = A¥b; toc
>> format long
>> x(1)
```

とすると1に近いとは言えなくなる.

このように，近似計算を利用している限り，

結果が必ず正しい解に近いという保証がない.

精度保証付き数値計算とは数値計算の手間に対し，

検算を行うことで近似解が正しい解の近くにあることを保証する数値計算法である.