KOBE HPC サマースクール（初級）
2019 講義資料
2019/08/26

兵庫県立大学シミュレーション学研究科
安田　修悟

連立一次方程式の解法1

- LU分解

  行列*A*を下三角行列*L*（対角成分は1）と上三角行列*U*に分解する。

$$A = LU$$

Aが3X3行列の場合：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ b_{21} & 1 & 0 \\ b_{31} & b_{32} & 1 \end{pmatrix} \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{pmatrix}$$

連立一次方程式の解法1

連立一次方程式　$A\vec{x} = \vec{y}$

- LU分解を行う。 $LU\vec{x} = \vec{y}$

  ※LU分解の方法については補足資料参照.

- $\vec{z} = U\vec{x}$ と置くと、 $L\vec{z} = \vec{y}$ と書ける。

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ b_{21} & 1 & 0 \\ b_{31} & b_{32} & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

連立一次方程式の解法1

連立一次方程式　$A\vec{x} = \vec{y}$

解法の手順

1. $L\vec{z} = \vec{y}$ を解いて、$\vec{z}$ を求める。
2. $U\vec{x} = \vec{z}$ を解いて、元の方程式の解 $\vec{x}$ を求める。

連立一次方程式の解法1

1. $L\vec{z} = \vec{y}$ を解いて、$\vec{z}$ を求める。

$$\begin{pmatrix} 1 & 0 & 0 \\ b_{21} & 1 & 0 \\ b_{31} & b_{32} & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

(1) $z_1 = y_1$

(2) $z_2 = y_2 - b_{21} z_1$

(3) $z_3 = y_3 - b_{31} z_1 - b_{21} z_2$

z_1から順次，z_2, z_3と求められる。

連立一次方程式の解法1

2. $U\vec{x} = \vec{z}$ を解いて、元の方程式の解 $\vec{x}$ を求める。

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

(1) $x_3 = z_3 / c_{33}$

(2) $x_2 = (z_2 - c_{23}x_3) / c_{22}$

(3) $x_1 = (z_1 - c_{12}x_2 - c_{13}x_3) / c_{11}$

x_3, x_2, x_1と順次，求められる。

連立一次方程式の解法2

- ヤコビの反復法（Jacobi method)

$$A\boldsymbol{x} = \boldsymbol{b}$$

$$A = D + U + L$$

D:diagonal matrix,
U:upper triangular matrix:
L: lower triangular matrix

$$\boldsymbol{x} = D^{-1}\left(\boldsymbol{b} - (U + L)\boldsymbol{x}\right)$$

$$\boldsymbol{x}^{(k+1)} = D^{-1}\left(\boldsymbol{b} - (U + L)\boldsymbol{x}^{(k)}\right)$$

$$\boldsymbol{x}^{(k)} \rightarrow \boldsymbol{x}^{*}$$

- ヤコビの反復法の形式

$$A\boldsymbol{x} = \boldsymbol{b}$$

$$A = D + U + L$$

D:diagonal matrix,
U:upper triangular matrix:
L: lower triangular matrix

$$\boldsymbol{x} = D^{-1}\left(\boldsymbol{b} - (U + L)\boldsymbol{x}\right)$$

$$x_i = c_0 x_0 + c_1 x_1 + \ldots + c_{i-1} x_{i-1} + c_{i+1} x_{i+1} + \ldots + c_N x_N$$

1. ***x*** の*i*成分について，成分*i*以外のベクトル要素の線形結合で表す.
2. *i=0,…,N*の*N+1*個の連立一次方程式を反復法で解く.

- ヤコビ反復法 （３×３行列）

$$\begin{pmatrix} 5 & 2 & -1 \\ 1 & 3 & 1 \\ 1 & -1 & 5 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 4 \end{pmatrix}$$

非対角成分を除いて右辺に移行

$$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ -2 \\ 4 \end{pmatrix} - \begin{pmatrix} 0 & 2 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \left[\begin{pmatrix} 5 \\ -2 \\ 4 \end{pmatrix} - \begin{pmatrix} 0 & 2 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right]$$

- ヤコビの反復法（３×３行列）

適当な初期値を使って
反復計算を実行.

$$\text{initial value } (x^{(0)}, y^{(0)}, z^{(0)}) = (x_0, y_0, z_0)$$

$$\begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \\ z^{(k+1)} \end{pmatrix} = \begin{pmatrix} \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \left[\begin{pmatrix} 5 \\ -2 \\ 4 \end{pmatrix} - \begin{pmatrix} 0 & 2 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \end{pmatrix} \right]$$

$$\begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \end{pmatrix} \to \begin{pmatrix} x^* \\ y^* \\ z^* \end{pmatrix} \quad (k \to \infty)$$

解が収束するまで反復する.

(x,y,z)が収束する場合には、その値は連立一次方程式の解を与える.

- ヤコビの反復法 （3×3行列）
  - 収束判定の例

$$\begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix} = \begin{pmatrix} 5 & 2 & -1 \\ 1 & 3 & 1 \\ 1 & -1 & 5 \end{pmatrix} \begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \end{pmatrix} - \begin{pmatrix} 5 \\ -2 \\ 4 \end{pmatrix}$$

$$\sqrt{e_x^2 + e_y^2 + e_z^2} \ll 1$$

元の方程式との残差を求めて
十分小さくなるまで反復する.

サンプルコード yabobi_3by3.c

連立一次方程式の解法

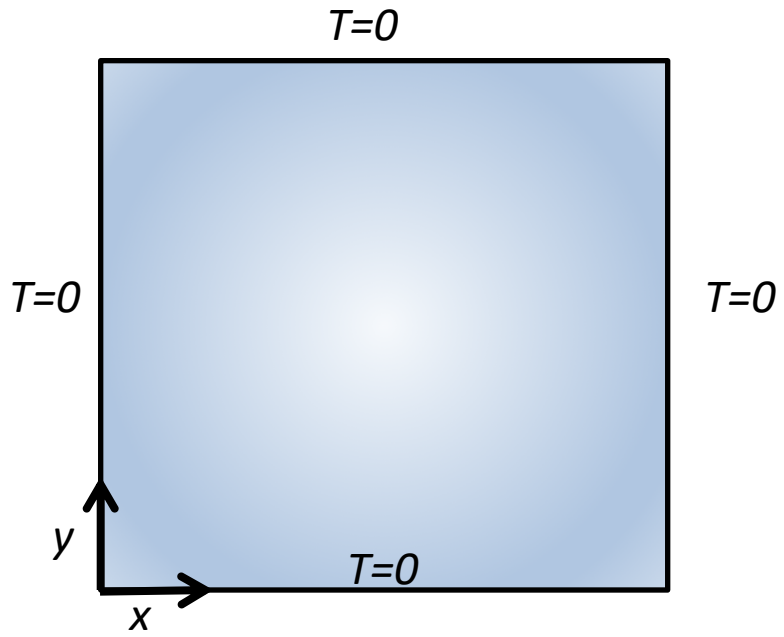
- サンプルコード
  次の連立一次方程式をヤコビ法とLU分解を使ってそれぞれ解くプログラムがyacobi_solver.c, lu_solver.cです.

$$\begin{pmatrix} 5 & 1 & 2 & 0 & -1 \\ 0 & 2 & -1 & 1 & 2 \\ 1 & 1 & 3 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ -2 \\ 0 \\ 4 \end{pmatrix}$$

熱伝達問題の差分解法

- 温度場を計算する。(ラプラス方程式)

$$-\Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 10$$



- 温度場を計算する。(ラプラス方程式)

$$-\Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 10$$

- 差分化する。

$$T(x_i, y_j) = T_{i,j}\ ,$$

$$x_i = y_i = \frac{i}{N} \quad (i = 0, \ldots, N),$$

$$\Delta h = 1/N$$

- 温度場を計算する。(ラプラス方程式)

$$-\Delta T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 10$$

- 差分化する。

$$T_{i,j} = \frac{1}{4}\left(10\Delta h^2 + T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1}\right)$$
$$(i, j = 1, \dots, N-1)$$

- $T_{i,j}$の線形連立方程式

$$T_{0,j} = T_{N,j} = 0, (j = 0, \dots, N)$$
$$T_{i,0} = T_{i,N} = 0, (i = 0, \dots, N)$$

境界条件として固定

ヤコビの反復法

$$T_{i,j}^{(k+1)} = \frac{1}{4}\left(10\Delta h^2 + T_{i-1,j}^{(k)} + T_{i+1,j}^{(k)} + T_{i,j-1}^{(k)} + T_{i,j+1}^{(k)}\right)$$
$$(i, j = 1, \dots, N-1)$$

サンプルコード laplace.c

演習4

- 熱伝達問題のコードで以下のことについて調べよ.
  - gnuplotで温度分布を可視化せよ.

    gnuplot
    >splot “laplace.dat”
    >set pm3d map
    >replot

  - 水平参照と垂直参照(2重ループの順番)を入れ替えて、計算時間を比較せよ.
  - 次頁で紹介するSIMD命令の効果について確認せよ.

ベクトル化

- SIMD（Single Instruction Multiple Data）
  - 1命令で複数要素の演算を行う.

```
for(i=0;i<n;i++){
    C[i]=A[i]+B[i];
}
```

- [SSE]: float4要素を一度に計算
- [AVX]: float8要素
- [AVX-512]: float 16要素
  (兵県大の計算機は未対応）

	A[0]	A[1]	A[2]	A[3]
+	B[0]	B[1]	B[2]	B[3]
	C[0]	C[1]	C[2]	C[3]

コンパイラーによるベクトル化

- ループの自動ベクトル化（-xまたは-axオプション）
  - 最適化レポート（-qopt-reportオプション）でベクトル化状況を確認

  icc -xhost laplace.c -qopt-report

  - -xhost: コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードを生成.
  - laplace.optrptにベクトル化状況を報告.
  - -no-vec: コンパイラーによるベクトル化を無効にする.

（補足）LU分解の方法

$$A = LU$$

Aが3X3行列の場合：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ b_{21} & 1 & 0 \\ b_{31} & b_{32} & 1 \end{pmatrix} \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{pmatrix}$$

具体的に書くと、

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ b_{21}c_{11} & b_{21}c_{12} + c_{22} & b_{21}c_{13} + c_{23} \\ b_{31}c_{11} & b_{31}c_{12} + b_{32}c_{22} & b_{31}c_{13} + b_{32}c_{23} + c_{33} \end{pmatrix}$$

（補足）LU分解の方法

1. 行列Aを次のようにA'に変換する。

L, U成分での表現は？

$$A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21}/a_{11} & a_{22} - a'_{21}a_{12} & a_{23} - a'_{21}a_{13} \\ a_{31}/a_{11} & a_{32} - a'_{31}a_{12} & a_{33} - a'_{31}a_{13} \end{pmatrix} \Rightarrow \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ b_{21} & c_{22} & c_{23} \\ b_{31} & b_{32}c_{22} & b_{32}c_{23} + c_{33} \end{pmatrix}$$

2. 行列A'をさらに次のようにA''変換するとL，Uの全ての要素が決定する。

$$A'' = \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} \\ a'_{21} & a'_{22} & a'_{23} \\ a'_{31} & a'_{32}/a'_{22} & a'_{33} - a''_{32}a'_{23} \end{pmatrix} \Rightarrow \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ b_{21} & c_{22} & c_{23} \\ b_{31} & b_{32} & c_{33} \end{pmatrix}$$

（補足）LU分解の方法

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \rightarrow \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ b_{21} & c_{22} & & c_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NN-1} & c_{NN} \end{pmatrix}$$

行列AをLU分解の要素行列に変換するプログラム。

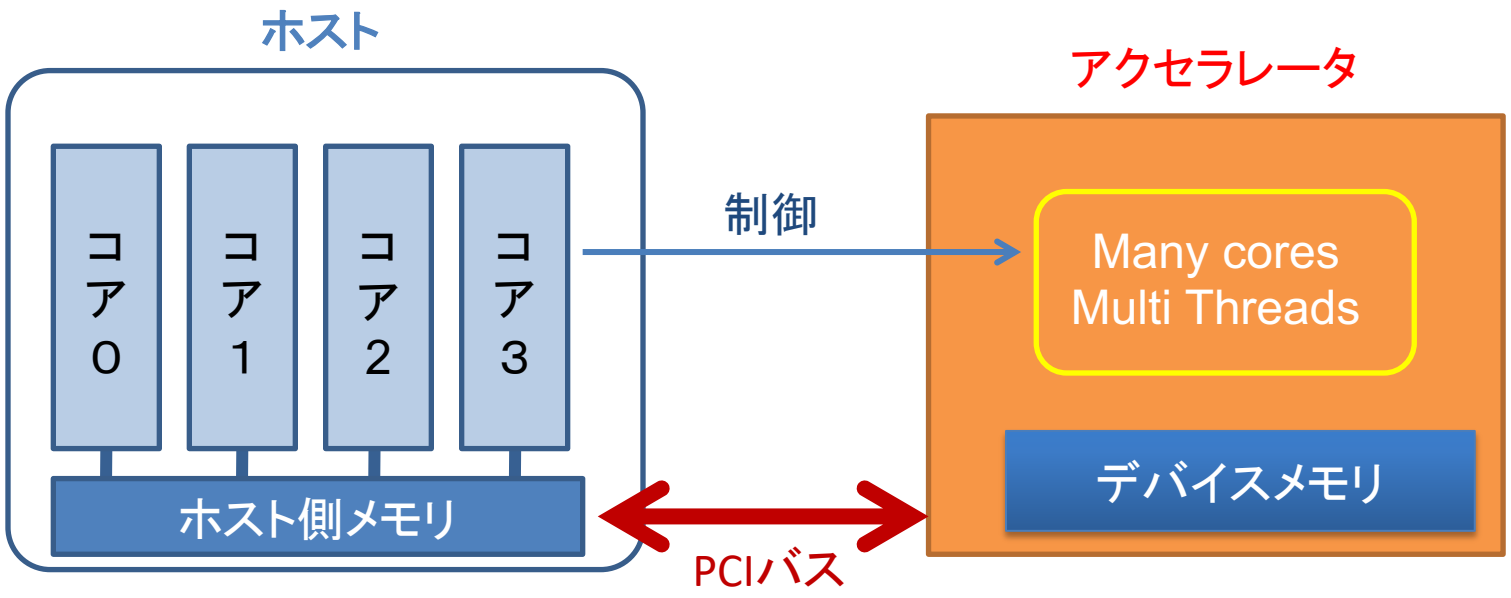
```
for(k=0;k<N-1;k++){
  w=1/a[k][k];
  for(i=k+1;i<N;i++){
    a[i][k] *= w;
    for(j=k+1;j<N;j++){
      a[i][j] -= a[i][k]*a[k][j];
    }
  }
}
```

サンプルコード lu_decomp.c

※ U行列の対角成分c_{ii}にゼロや小さな値が出てくる場合には、更に、行や列の入れ替えを伴う複雑な手順を考える必要がある。

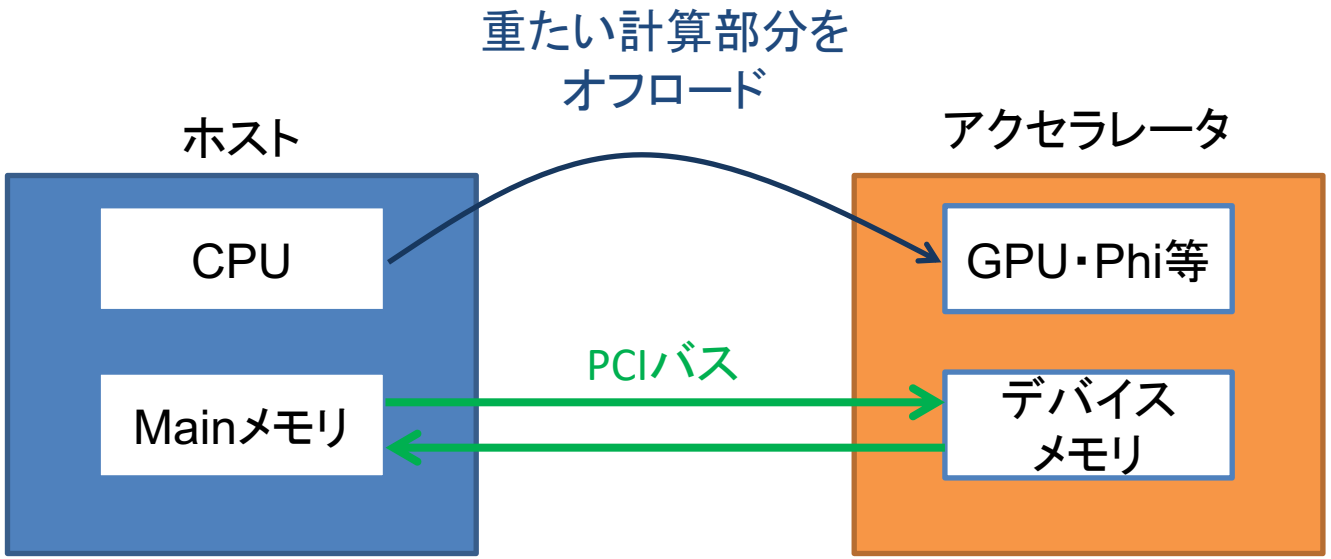
アクセラレータとは？

- CPUの処理を代替して処理の効率を向上させる装置（ディバイス）。
  - GPGPU (NVIDIA), Xeon Phi (Intel), APU (AMD)など



アクセラレータの実行イメージ

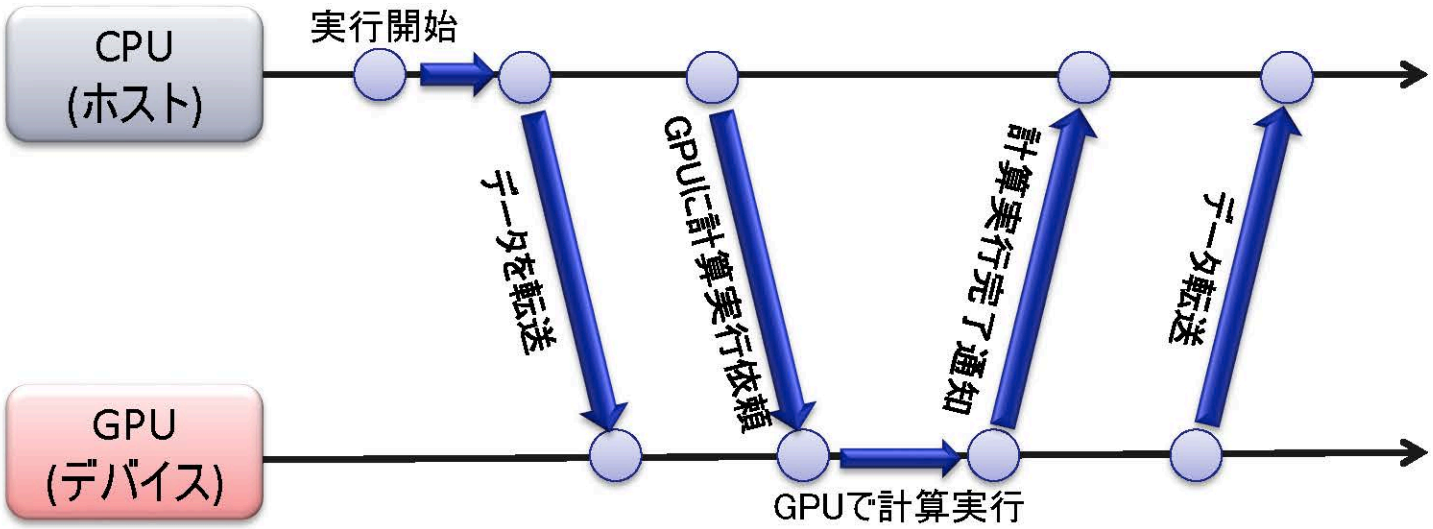
ハイブリット構成 （CPU＋アクセラレータ）



※データのオフロード通信がボトルネックとなる

- CPUのメモリ空間とディバイスのメモリ空間が異なる
- 一般的なメモリ帯域幅に比べてPCIバスでの低速な転送

GPUでのプログラム実行イメージ



- GPU(デバイス)ではCPU(ホスト)から制御をします。
- GPU(デバイス)の計算で必要なデータはCPU(ホスト)から転送します。
- GPU(デバイス)で計算した結果はCPU(ホスト)に転送します。

OpenACC

- Accelerators用のAPIの一つ。
- CPUからディバイスに処理をオフロードする際に使用する。
- Fortran/C/C++言語でディレクティブを指定（OpenMPと同様）
- CUDA等の専用プログラムに比べて、導入が容易。

OpenACCプログラミング

- 基本フォーマット

```
#pragma␣acc␣ディレクティブ␣節1␣節2␣・・・
{
        構造化ブロック（途中で抜け出したり、終了しない）
}

例
#paragma ␣acc ␣kernels␣copy(a)
for(i=0;i<imax;i++){
        a[i]=a[i]*a[i];
        ・・・・
}
```

OpenACCプログラミング

- 基本フォーマット (Fortranの場合)

```
!$acc␣ディレクティブ␣節1␣節2␣・・・
        構造化ブロック
!$acc end ディレクティブ

!$acc ␣kernels␣copy(a)
do i=0, imax
        a[i]=a[i]*a[i];
        ・・・・
end do
!$acc end kernels
```

基本的なディレクティブ構文

① Accelerator compute構文
- オフロードするループ対象部分を指定.
- kernels構文とparallel構文.

② DATA構文
- ホストとアクセラレータのデータ転送を明示的に指示.
- 高速化の鍵を握る.

③ Loop構文
- ①のオフロード領域のループのベクトル長や並列分割の詳細を明示的に指示することができる.

PGIコンパイラー

- Module環境の設定
  module␣load␣pgi
  - module␣listでintelが設定されている場合
    module␣unload␣intel
    でIntel環境をunloadする。
- OpenACCのコンパイル
  pgcc␣-acc␣<prog.c> ␣-o␣acc.out
  pgfortran –acc <prog.f90>
  - “-acc”でOpenACCディレクティブを有効にする.

PGIコンパイラー

- OpenMPのコンパイル
  pgcc␣-mp␣program.c ␣-o␣mp.out
  “-mp”はOpenMPディレクティブを有効にするためのオプション。
- 最適化オプション
  - “-fastsse” : 一般的な最適化オプション
  - “-O3”, “-O4”: より高度な最適化オプション
    （”-fastsse”より後に入れる
    例 pgcc␣-fastsse␣-O3␣program.c）

PGIコンパイラー

- 環境変数（実行前に設定）
  - PGI_ACC_TIME
    - 実行後に簡易プロファイル情報を標準出力に出力。
    - 0（ゼロ）を指定すると機能を抑制（デフォルト）、ゼロ以外の整数値で機能する。
  - PGI_ACC_NOTIFY
    - ディバイスの実行イベントを表示する
    - 整数値１：Kernel launch のイベントを出力、整数値２：データ転送のイベントの出力、など
    - 0（ゼロ）を指定すると機能を抑制（デフォルト）

GPUキューの実行

- インタラクティブ実行
  qsub␣-I␣-q␣G
  GPUを搭載したノードにてプログラムを実行。
- ホストプログラムの並列化に応じてジョブスクリプトを作成.　(以下の例は，ホストが並列化無しの場合）

```
#!/bin/bash
#PBS␣-q␣G
#PBS␣-l␣select=1:ncpus=1:mem=1gb
#PBS␣-N␣job_gpu
#PBS␣-j␣oe
source␣/etc/profile.d/modules.sh
module␣purge
module␣load␣pgi
cd␣${PBS_O_WORKDIR}
dplace␣./a.out
```

・現在のモジュール環境を全て破棄
・PGIコンパイラの環境設定

OpenACCの演習1

- pi1.cとpi2.cのループにOpenACC指示文
  #pragma acc kernels
  を入れて書き換え，CPU計算とGPU計算の時間を比較せよ.

- pi1.cとpi2.cのオフロード処理の違いに着目して、GPUでの計算時間の違いについて考察しなさい.

DATA構文

- #pragma␣acc␣data␣節
- 節には以下のようなものがある.（Kernel構文の節にも使える.）
  - copyin(a,b): 配列a, bをホストからアクセラレータにコピーする.
  - copyout(c): 配列cをアクセラレータからホストにコピーする.
  - copy(A)：copyinとcpyoutの双方を行う.
  - create(B): アクセラレータでローカルに使用する配列Bの領域を作成.
  - present(C): 対象となる処理に入ったときにすでにデバイスメモリにある配列Cのデータを使用する.

DATA構文

- pi2.cのOpenACCプログラムに対して,

1. GPU内だけで必要な変数を予め作成する.

```
data create(f,x,y){
.......
}
```

2. Kernel構文でデータ属性を指示する.
   - kernel present(f) copyin(dh,nmax)
   - kernel present(f) copy(sum) copyin(nmax)

OpenACCの演習2

- ラプラス方程式のOpenACCプログラム

1. laplace.cの主要なループをkernel構文を使って書き換える.
2. データ構文を使ってオフロード処理を出来るだけ抑制するコードを作成する.
3. シリアル計算とGPU計算の時間を比較する.

OpenACCの演習3

- セルオートマトンのOpenACCプログラム
  (セルオートマトンについては補足資料を参照)

1. r184.cの主要なループをkernel構文を使って書き換える.
2. データ構文を使ってオフロード処理を出来るだけ抑制するコードを作成する.
3. シリアル計算とGPU計算の時間を比較する.

（補足）セルオートマトン

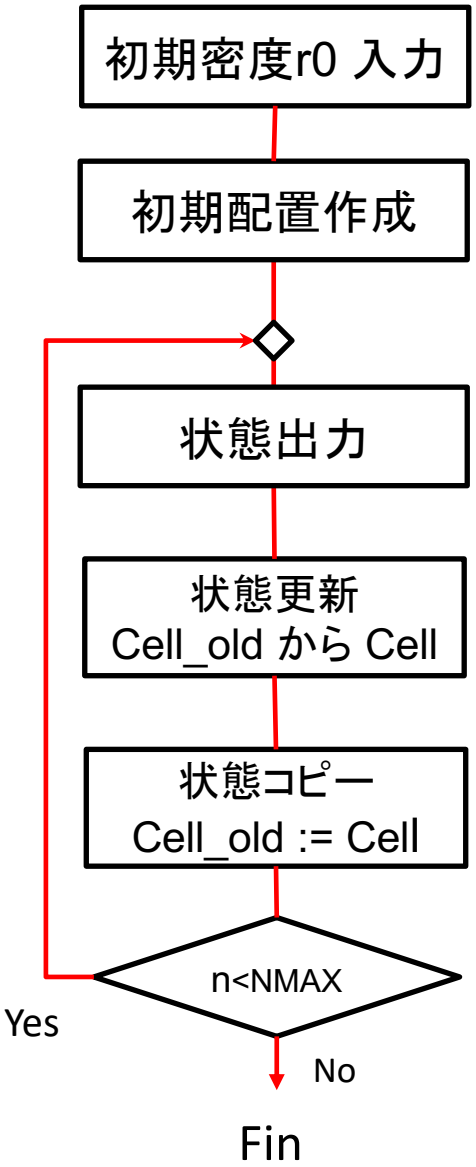
- 格子状のセルと単純な規則による，離散的計算モデルである．（ウィキペディアより）
- 交通流モデルの一つであるルール184を扱う．
  - 1つのセルに入ることのが出来る車両は最大1台．
  - 車両は1タイムステップで右に1セル分だけ進める．
  - 1つ先のセルに車両が存在すれば，次のタイムステップでは先のセルに進めず，そのセルに留まる．
  - 1つ先のセルに車両が存在しなければ，次のタイムステップに先のセルに進める．

（補足）セルオートマトン

- 境界条件
  - 周期境界条件・・・右端のセルから出て行く車両は左端のセルに移動する．円周上に並んだセルを考えることができる．
  - 開放境界条件・・・左端のセルに車両が入ってくる確率と，右端のセルから車両が出て行く確率を予め設定する．

全体の流れ

（補足）フローチャート



IMAX: セルサイズ
NMAX: タイムステップ数

Cell[IMAX]; 現在の状態
Cell_old[IMAX]; 過去の状態

状態更新の方法は？