KOBE HPCサマースクール（初級）2023 講義資料 2023/09/18

兵庫県立大学 情報科学研究科
安田　修悟

背景と講義目的

- 背景
  「富岳」のような大規模並列スーパーコンピュータによるシミュレーションを活用するためには、並列アプリケーションの開発に必要な基本的な並列計算法についての知識が必要となる。

- 講義目的
  並列計算についての知識と基礎的な並列計算のプログラミング技術の習得を目的とする。
  - 各種並列計算（スレッド並列、プロセス並列、アクセラレータ）について、それぞれ具体的なプログラミング法を学習する。
  - 高速化のためのプログラミング技術についても学習する。

受講対象者

1. C言語やFortran言語などのプログラミング言語の経験があること。
2. EmacsやVim等のエディタによるファイル編集ができること。
3. Linux/Unixの基本的なコマンドを使えること。（準必須）
4. データ計算科学の研究においてスパコンおよび並列処理の知識・技術の習得が必要であること。

講義内容

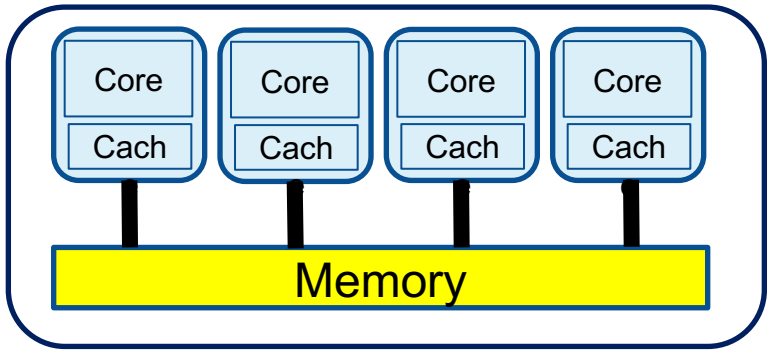
1. 計算機サーバーの環境設定と使い方　（担当　兵県大・安田）
2. シリアルプログラムの高速化
3. 熱伝達問題の差分計算
4. スレッド並列とは　（担当　R-CCS・今村先生）
5. OpenMPによるループ処理の並列化
6. 差分化された偏微分方程式の並列化
7. （招待講演：芝隼人先生）
8. アムダール法則と並列化効率の評価
9. 分散メモリ型並列計算機とは何か？　（担当　神大・三宅先生）
10. 1対1通信関数、集団通信関数
11. 並列計算性能の評価方法
12. 熱伝導問題のプロセス並列計算
13. ハイブリッド並列
14. アクセラレータとは　（担当　兵県大・安田）
15. OpenACCプログラミング
16. まとめと確認テスト

並列計算機と並列計算の種類

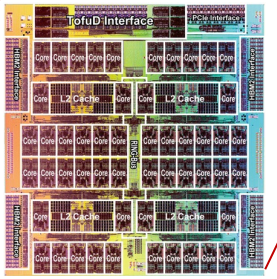
並列計算機

- 計算機アーキテクチャ

SMP（Symmetric Multiprocessing) 共有メモリ型

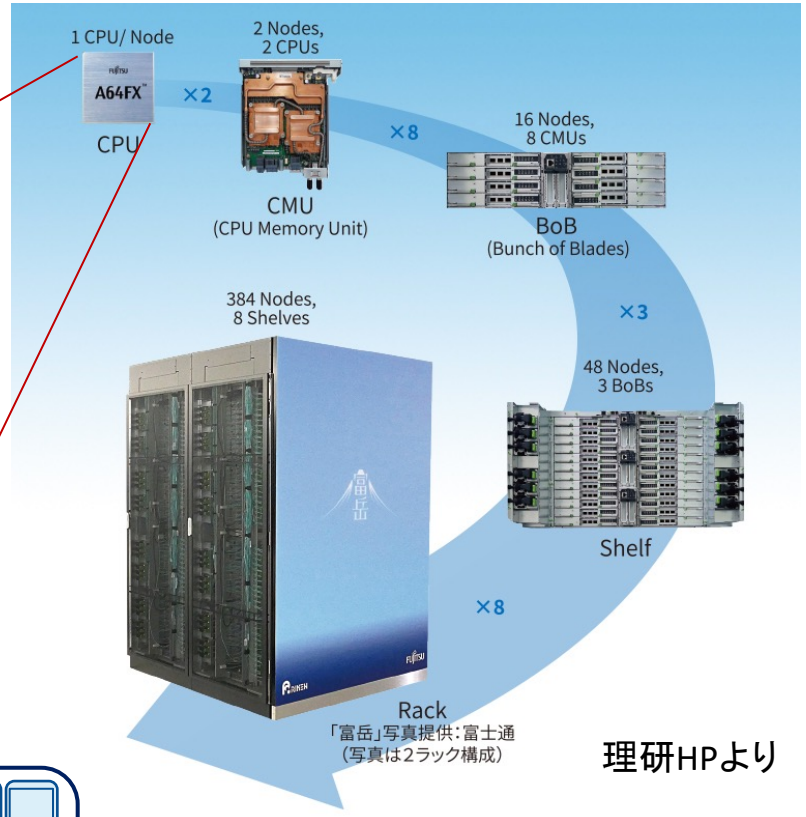


複数のコアが均一に並列処理



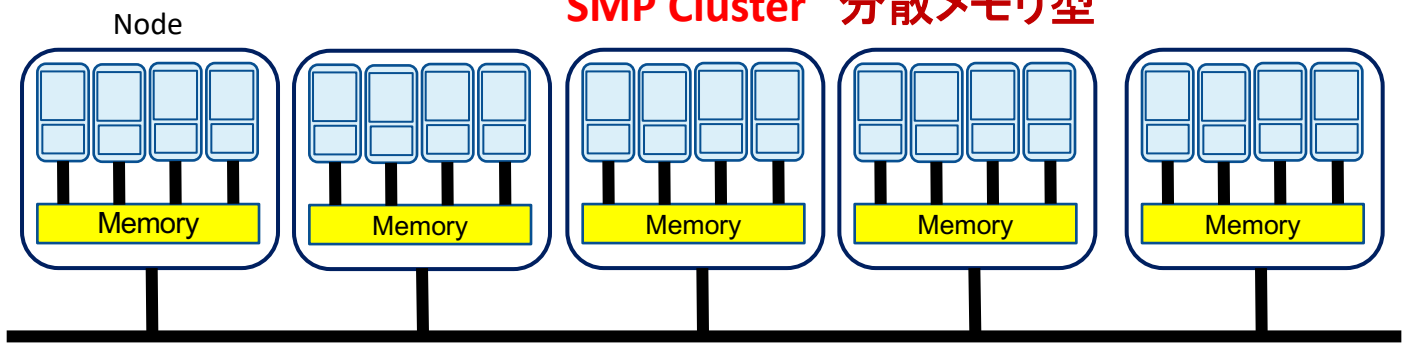
A64FX CPUダイ

富岳の構成



理研HPより

SMP Cluster　分散メモリ型

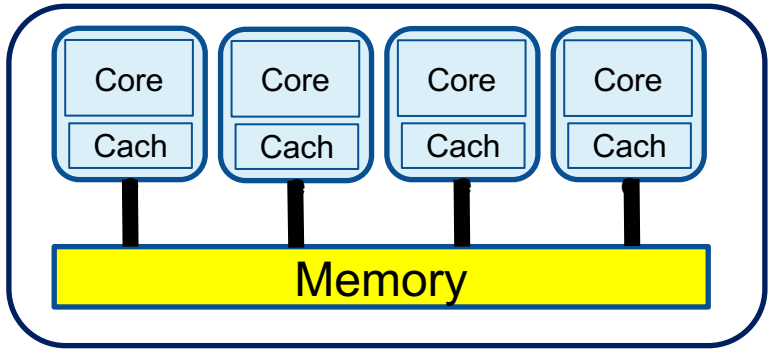


複数のプロセッサがネットワークを介して通信

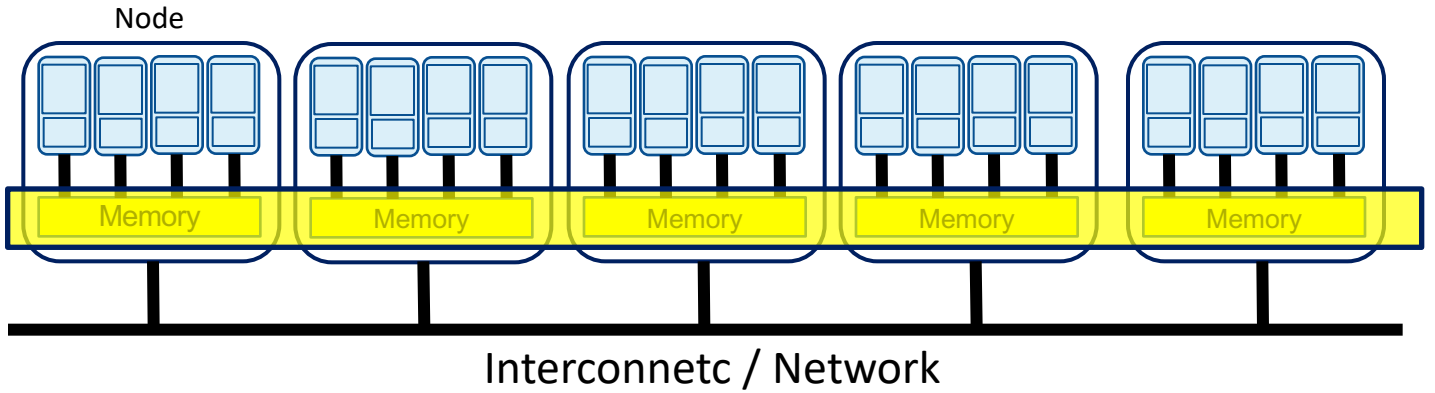
並列計算機

- 計算機アーキテクチャ

SMP（Symmetric Multiprocessing) 共有メモリ型



NUMA (Non-Uniform Memory Access) 共有メモリ型マルチプロセッサ計算システム

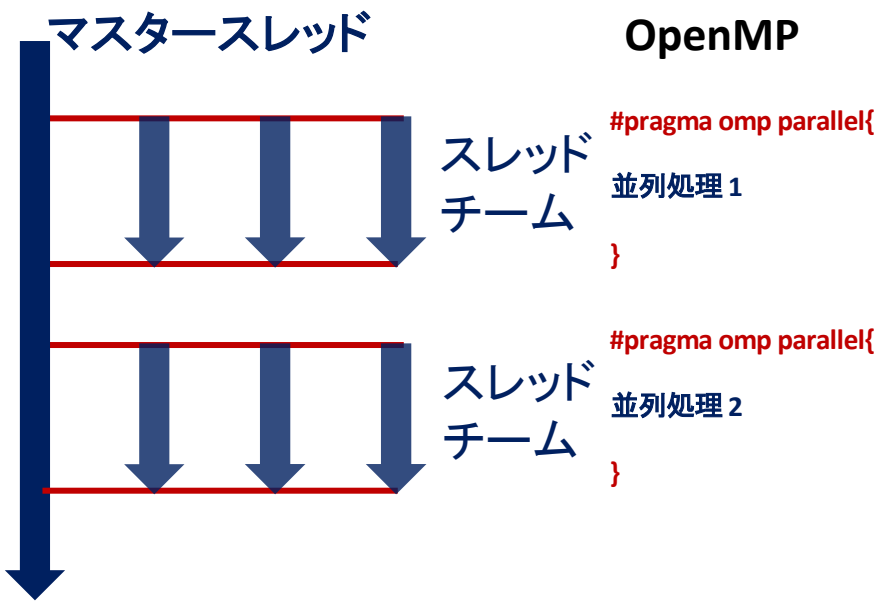


並列計算とは

- 並列計算・・・演算を分割して、複数のプロセッサに割り当てて処理する
  - スレッド並列（共有メモリ型）
    - 分割された演算（スレッド）が、メモリ空間を共有できる。
    - 複数のスレッドが均一に並列処理をする。
  - プロセス並列（分散メモリ型）
    - 分割された演算（プロセス）が、それぞれ独立のメモリ空間を参照。
    - プロセス間にデータ転送が必要。

並列計算の実行イメージ

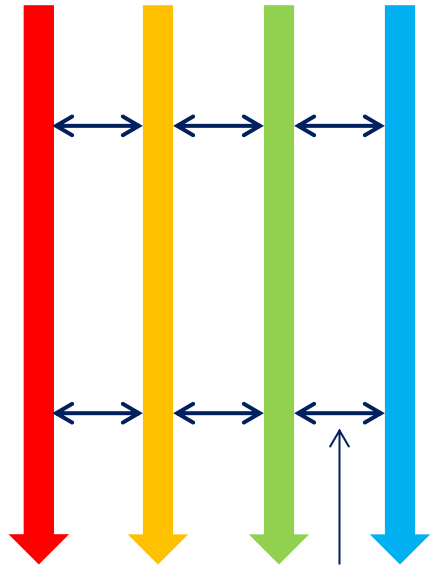
- スレッド並列



OpenMP

```
#pragma omp parallel{
並列処理 1
}
#pragma omp parallel{
並列処理 2
}
```

- プロセス並列



MPI

```
MPI_Comm_size(MPI_COMM_WORLD,&nprocs)
MPI_Comm_rank(MPI_COMM_WORLD,&myrank)

/*プロセス間データ通信*/
MPI_Send(&message,count,datatype,dest,tag,comm)
MPI_Recv(&message,count,datatype,source,tag,comm)

/*プロセス間データ通信*/
MPI_Send(&message,count,datatype,dest,tag,comm)
MPI_Recv(&message,count,datatype,source,tag,comm)
```

プロセス間通信

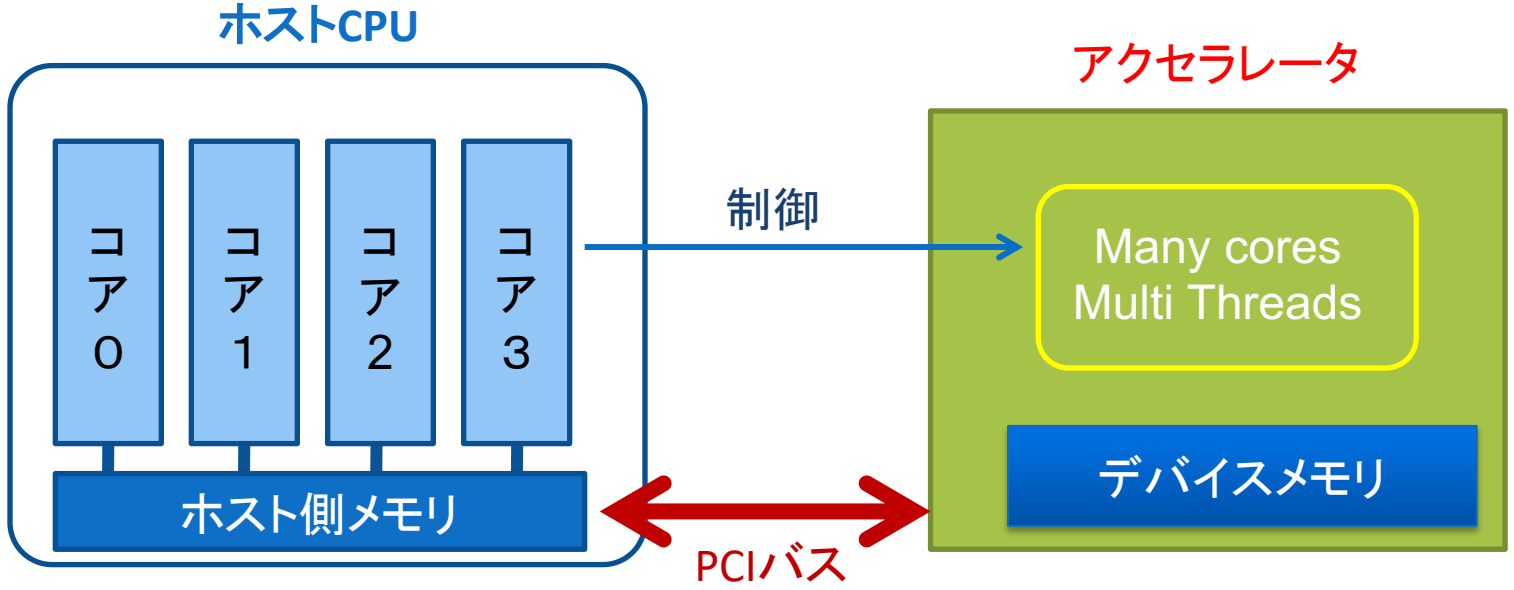
並列計算機と並列計算の種類

	SMP	NUMA	Cluster
特徴	• メモリを共有 • データ転送が不要	• 論理的にメモリ共有 • データ転送が不要	• ローカルにメモリ • ノード間データ転送が必要
プログラミング	• 容易 自動並列化、OpenMPが可能	• 容易 自動並列化、OpenMPが可能	• 難しい MPIによるノード間通信が必要。 （自動並列化、OpenMPはノード内でのみ可能）
スケーラビリティ	中	高い	非常に高い

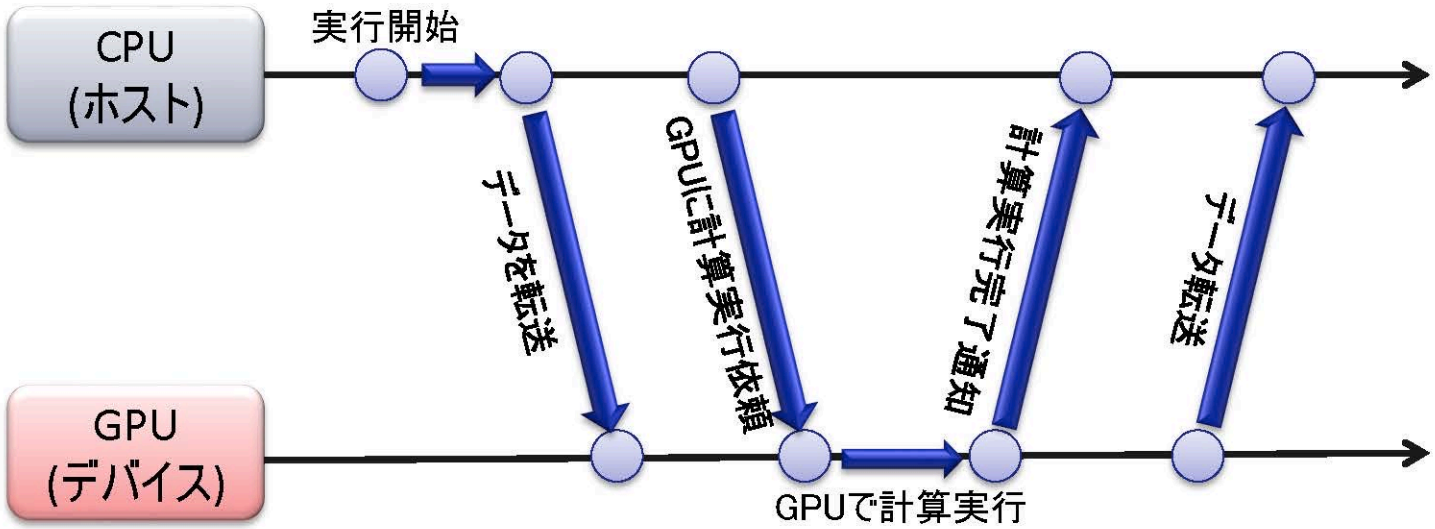
アクセラレータ

- CPUの処理を一部代替して全体の処理の効率を向上させる装置

ハイブリット構成 （CPU＋アクセラレータ）



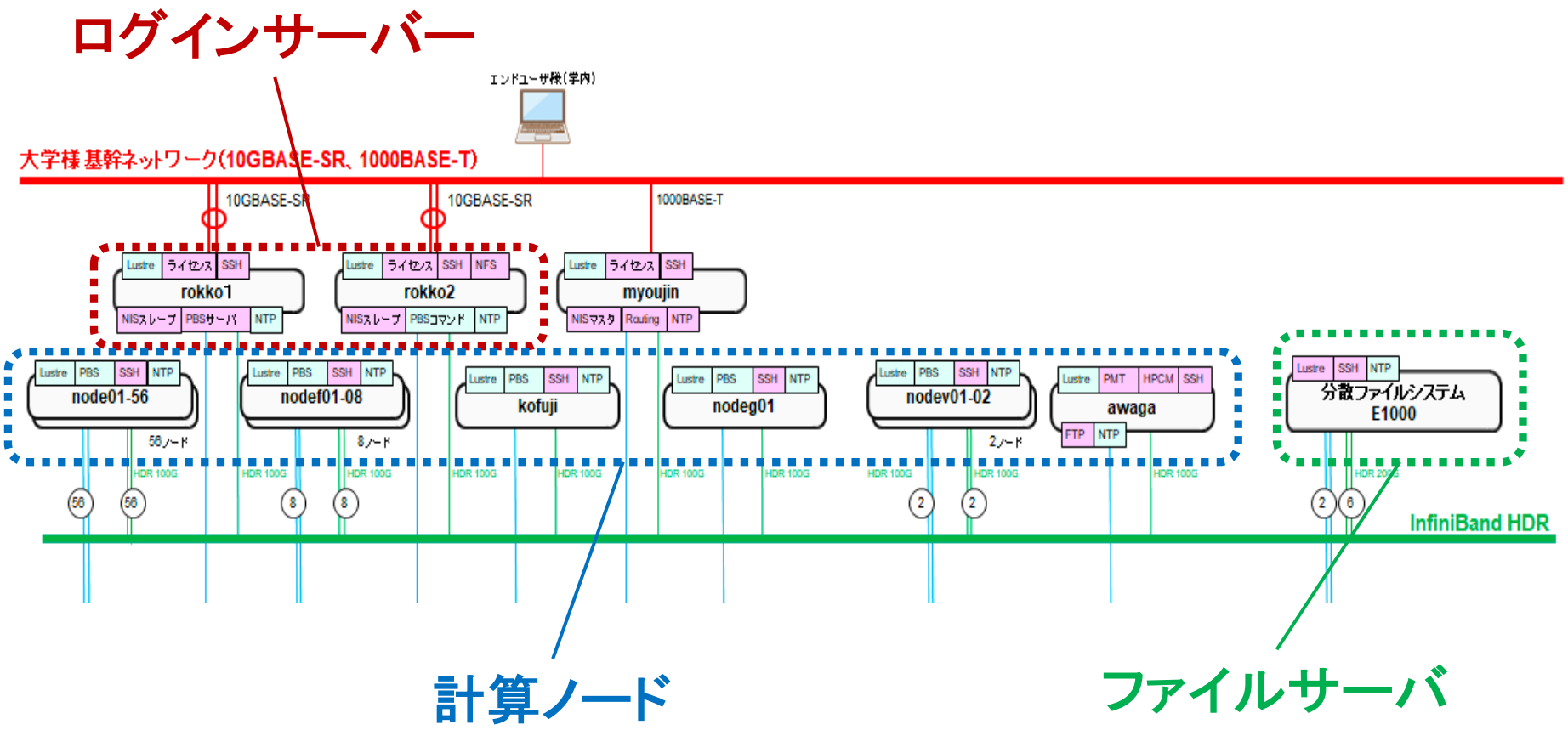
アクセラレータの実行イメージ



- 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。

計算機サーバと環境設定

兵庫県大・情報科学キャンパス
計算機サーバシステム概要



計算機サーバシステム概要

	Host	System	CPU/Accelerator	Memory
フロントエンドサーバ	rokko1 rokko2	HPE ProLiant DL380 Gen10	Intel Xeon Gold 6248 2.5GHz (20cores x2)	768GB
CPUノード（Thin/Fatノード）	node01~56 nodef01~08	HPE Apollo 2000 Gen10	Intel Xeon Gold 6248 2.5GHz (20cores x2)	192GB (Thin)/ 768GB (Fat)
GPUノード（gpu搭載ノード）	nodeg01	HPE Apollo 6500 Gen10	Intel Xeon Gold 6248 2.5GHz (20cores x2) / NVIDIA V100 32GB SXM2 x8	768GB
VEノード（VE搭載ノード）	nodev01~02	HPE Apollo 6500 Gen10	Intel Xeon Gold 6248 2.5GHz (20cores x2) / NEC Vector Engine Accelerator Module x8	768GB
共有メモリノード	kofuji	HPE ProLiant DL560 Gen10	Intel Xeon Gold 6248 2.5GHz (20cores x2) x4 (Total 80cores)	3TB

フロントエンドサーバ(rokko)へのログイン

- アカウントシート：アカウント名と初期パスワード

- フロントエンドサーバ IPアドレス
  rokko1: 172.25.61.33
  rokko2: 172.25.61.34
  （学籍番号が奇数の人はrokko1へ偶数の人はrokko2へ）

- SSHコマンド
  ssh␣-Y␣アカウント名@172.25.61.33
  （-YはX11転送を有効にするオプション。GUI操作を可能にする）

  ⇒ログインできたらコマンド “xclock” を入力。時計が出てきたらOK。

プログラミング環境設定

- moduleコマンド
  - Module環境のロード／アンロード
    module␣load␣intel / module␣unload␣intel
    module␣load␣gnuplot / module␣unload␣gnuplot
  - 現在の設定
    module␣list
  - 利用可能なmoduleの確認
    module␣avail
  - Module環境の初期化
    module␣purge

- 「利用者マニュアル_rev1.4」の12頁参照
  （マニュアルは当日に送付します．）

プログラミング環境設定

1. “.module”ファイルの作成
   > vim␣.module

```
source␣/etc/profile.d/modules.sh
module␣load␣intel
module␣load␣gnuplot
```

2. “~/.bashrc”の最下部に次の1行を追記.
   > vim␣.bashrc

```
.␣~/.module
```

3. 端末に “bash”と入力．Module環境の確認.
   > module␣list

プログラミング環境

- コンパイラ(「利用者マニュアル」30頁以降)
  - Intel
    icc␣-O3␣hello.c␣-o␣hello.out
    オプション　　実行ファイルの指定
- 実行はPBSジョブ管理システムを使います.
  (「利用者マニュアル」14頁)
  - ジョブスクリプトの利用
    qsub␣<ジョブスクリプト>
  - インタラクティブジョブ
    qsub␣-I

PBSジョブ管理システム

- バッチスクリプトの作成
  vim␣sample.sh

/tmp/khpc2023/20230918/sample.sh

```
#!/bin/bash
#PBS␣-q␣T                          ・キューの指定
#PBS␣-l␣select=1:ncpus=1           ・計算リソースの指定
#PBS␣-N␣sample_job                 ・任意のジョブ名の指定
#PBS␣-j␣oe                         ・標準出力と標準エラー出力を統合

cd␣${PBS_O_WORKDIR}                ・投入ディレクトリに移動

echo “Hello KOBE HPC Summer”       ・ジョブ内で実行するコマンド
```

- ジョブ投入
  qsub␣sample.sh

PBSジョブ管理システム

- キュー構成

キュー名		ジョブのコア数制限	最長実行時間	使用できるノード	アクセラレータ数	優先度	インタラクティブ
T		1-40	30min	CPU ノード(Thin)	-	150	可
TF		1-40	30min	CPU ノード(Fat)	-	150	可
C	S	1-80	12h	CPU ノード(Thin)	-	130	可
	M	81-200	24h	CPU ノード(Thin)	-	90	-
	L	201-2240	24h	CPU ノード(Thin)	-	70	-
	H	2241-2560	24h	CPU ノード(Thin, Fat)	-	50	-
SF		1-80	12h	CPU ノード(Fat)	-	130	可
MF		81-160	24h	CPU ノード(Fat)	-	90	-
LF		161-320	24h	CPU ノード(Fat)	-	70	-
LONG		1-40	168h	CPU ノード(Thin)	-	130	-
G		1-40	24h	GPU 搭載ノード	GPU 8	130	可
V		1-40	24h	VE 搭載ノード	VE 8	130	可
VL		41-80	24h	VE 搭載ノード	VE 16	70	-
SMP		1-80	24h	共有メモリノード	-	130	可
SMP-LONG		1-80	168h	共有メモリノード	-	130	-

- キュー名 : ジョブ投入の際に指定するキュー名です.
- ノード/コア数制限 : ジョブ投入の際に予約できる最小・最大ノード/コア数です.
- 最長実行時間 : ジョブ投入の際に予約できる最大実行時間です.
- 使用できるノード : 各キューで利用可能なノードです.
- アクセラレータ数 : 各キューで利用可能なアクセラレータ数です.
- 優先度 : キューの優先順位. 大きい値の方が優先度が高くなります.

サマースクールの期間中限定
WSキュー: 15 CPUノード

PBSジョブ管理システム

- 結果表示
  cat␣sample_job.o<ジョブID>
- 実行ジョブの確認（「利用者マニュアル」28頁）
  qstat

表 5 qstat コマンドの主なオプション

オプション	説明
-u␣<user_name>	指定したユーザのジョブの状態を表示します.
-f␣<ジョブ ID>	特定のジョブの詳細な状況を表示します. ジョブ ID を省略すると, すべてのジョブについての詳細な状況を表示します.
-Q	キューの状況を表示します.
-s	ジョブコメントおよびその他の情報が表示されます. キューイング中のジョブは実行待ちになっている理由が表示されます.

- ジョブのキャンセル（「利用者マニュアル」29頁）
  qdel␣<ジョブID>

PBSジョブ管理システム

- インタラクティブキュー
  - **qsub␣-I␣-q␣WS**　クラスタノードにログイン.
    “-q”で利用するキューの指定.
  - Job IDが割り当てられる. **qstat**で確認.
  - 作業ディレクトリに移動して作業を実行
    module␣load␣intel
    icc␣hello.c
    ./a.out
  - **exit** コマンドで終了.（ノードからログアウト. ）
    必ずexitでログアウトして下さい!!

演習0

- hello.cをインタラクティブキューを使って実行する.
  1. qsub -I -q WS
  2. cd　作業ディレクトリ
  3. icc hello.c -o hello.out
  4. ./hello.out

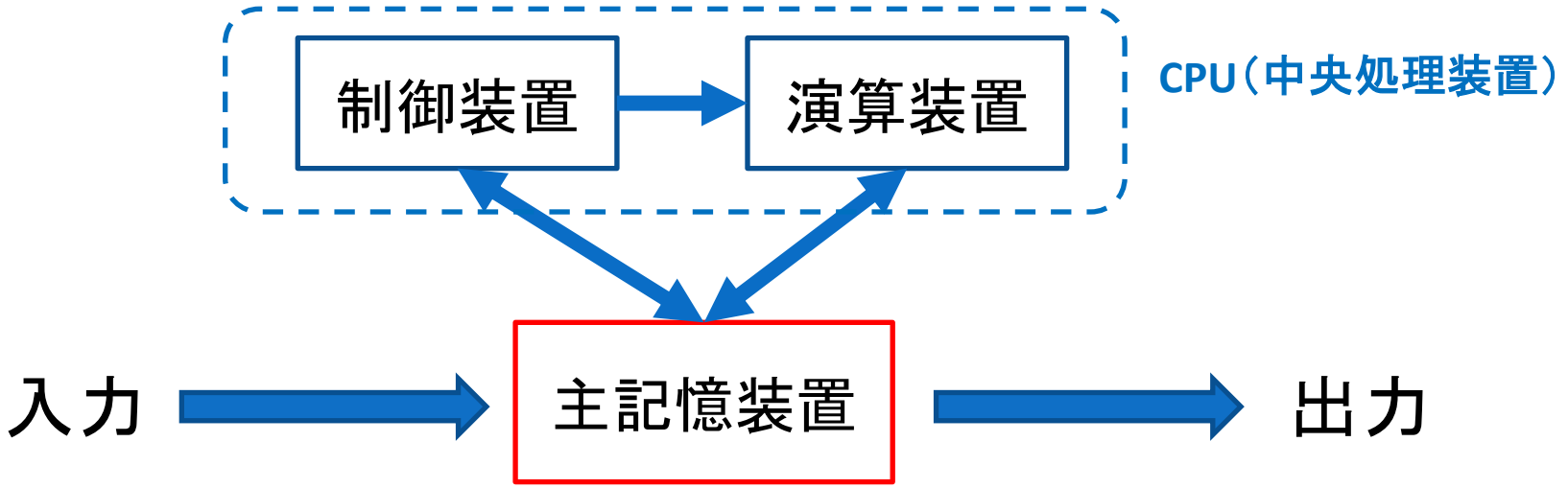
- バッチスクリプトを使って実行する.

※ Linux環境構築やネットワーク設定等については補足資料を参照すること.

逐次プログラムの高速化

計算機の基本構造

- フォン・ノイマン型 (von Neumann architecture)



1. 主記憶装置（メモリ），1次元のアドレス空間をもつ，
2. 命令とデータがともに記憶装置に記憶される．
3. 演算は制御装置からの命令信号によって実行される．
4. 命令は，プログラムカウンタにより逐次的に実行される．

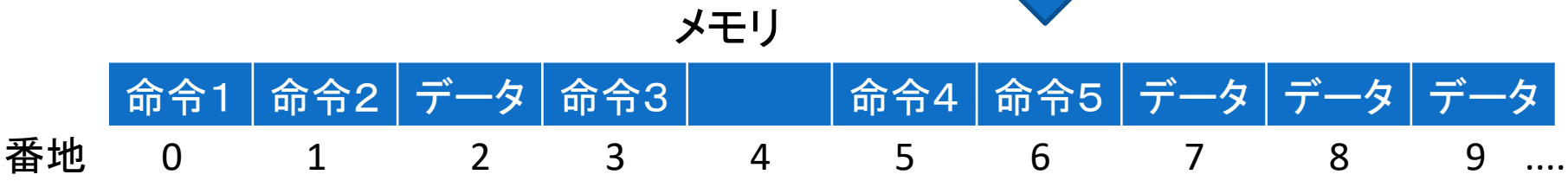
計算機の動作原理

- チューリングマシン

2．ヘッダ（情報の読み書き）
3．内部状態メモリ

- テープの情報の読み書き.
- 機械の内部状態の変更.
- ヘッドの移動.

- ノイマン型

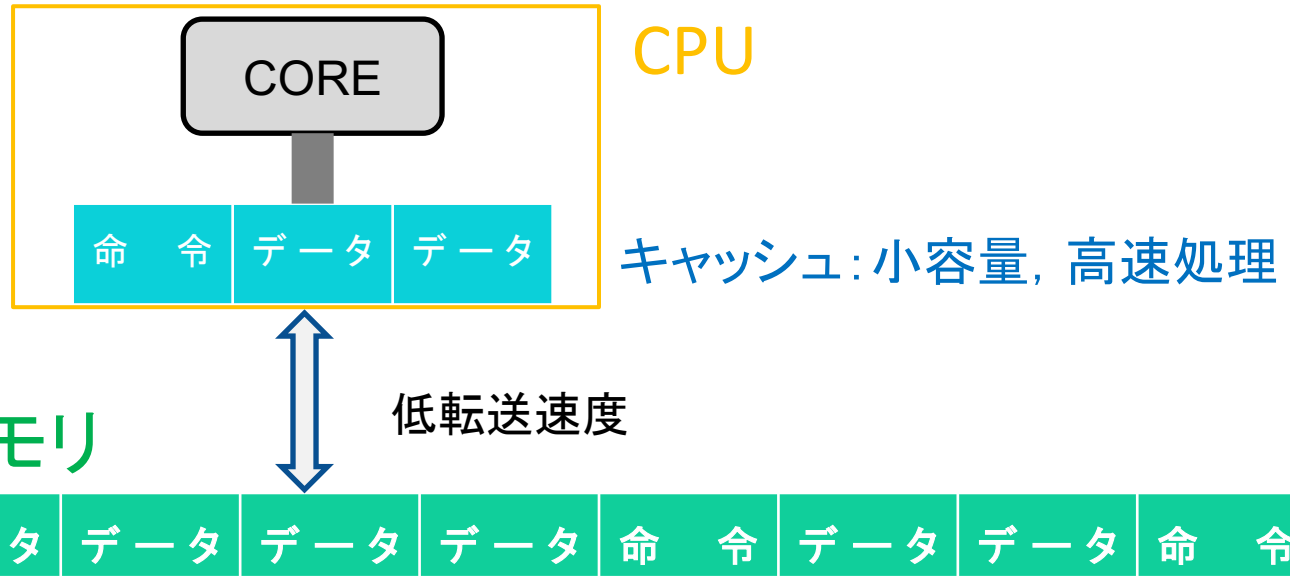


プログラムカウンタ　->　実行する命令の番地を示す

0 1 3 5 ~~6~~1　　※分岐命令でカウンタを書き換える.

メモリアクセス

- キャッシュメモリ



主記憶装置（メモリ）：大容量，1次元アドレス空間

番地で区切られたデータ量をメモリとキャッシュでやりとりする.

逐次計算の高速化

- ループ展開
  - 繰り返し処理で毎回発生する終了条件のチェックの低減
  - ループ制御のカウンタやポインタの更新回数の低減

行列積の計算

```
for(i=0;i<IMAX;i++)
    for(j=0;j<IMAX;j++)
        for(k=0;k<IMAX;k++)
            c[i][j]+=a[i][k]*b[k][j];
```

2つ目のループについて展開

```
for(i=0;i<IMAX;i++)
    for(j=0;j<IMAX;j+=8)
        for(k=0;k<IMAX;k++){
            c[i][j]+=a[i][k]*b[k][j];
            c[i][j+1]+=a[i][k]*b[k][j+1];
            c[i][j+2]+=a[i][k]*b[k][j+2];
            c[i][j+3]+=a[i][k]*b[k][j+3];
            c[i][j+4]+=a[i][k]*b[k][j+4];
            c[i][j+5]+=a[i][k]*b[k][j+5];
            c[i][j+6]+=a[i][k]*b[k][j+6];
            c[i][j+7]+=a[i][k]*b[k][j+7];
        }
```

逐次計算の高速化

- 一時変数の活用
  - ループ内で同じ変数へのアクセスが多い場合，一時変数を用いると冗長な命令を省く事ができる．

```
for(i=0;i<IMAX;i++)
    for(j=0;j<IMAX;j+=8)
        for(k=0;k<IMAX;k++){
            c[i][j]+=a[i][k]*b[k][j];
            c[i][j+1]+=a[i][k]*b[k][j+1];
            c[i][j+2]+=a[i][k]*b[k][j+2];
            c[i][j+3]+=a[i][k]*b[k][j+3];
            c[i][j+4]+=a[i][k]*b[k][j+4];
            c[i][j+5]+=a[i][k]*b[k][j+5];
            c[i][j+6]+=a[i][k]*b[k][j+6];
            c[i][j+7]+=a[i][k]*b[k][j+7];
        }
```

配列a[i][k]へ冗長なアクセス

```
for(i=0;i<IMAX;i++)
    for(j=0;j<IMAX;j+=8)
        for(k=0;k<IMAX;k++){
            t=a[i][k];
            c[i][j]+=t*b[k][j];
            c[i][j+1]+=t*b[k][j+1];
            c[i][j+2]+=t*b[k][j+2];
            c[i][j+3]+=t*b[k][j+3];
            c[i][j+4]+=t*b[k][j+4];
            c[i][j+5]+=t*b[k][j+5];
            c[i][j+6]+=t*b[k][j+6];
            c[i][j+7]+=t*b[k][j+7];
        }
```

演習1 (ex01.c)

- ループ展開無し，ループ展開で2回処理，4回処理，8回処理の場合について実行速度を比較せよ.
- ループ展開で8回処理した後に，さらに一時変数を用いることで実行速度が速くなるか確認せよ.

※コンパイルオプションを”-O0”（最適化無し）としてコンパイルする.

逐次計算の高速化

- 水平参照と垂直参照
  - 水平参照の方がキャッシュミスが少ない.

2次元配列 **A[I][J]**　　A[i][j]は1次元メモリ空間で $i \times JMAX + j$ 番目のデータ.

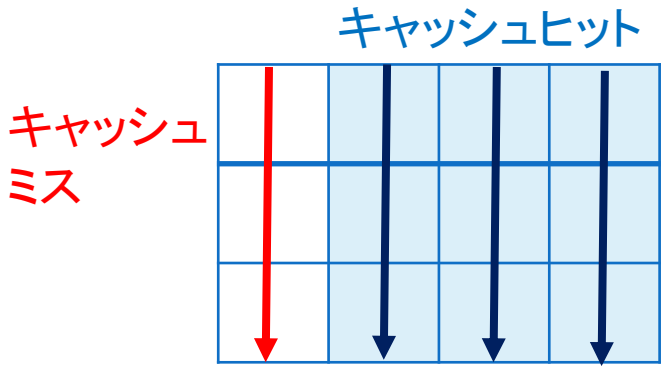
0	1	2	3	.....	J - 4	J - 3	J - 2	J - 1
J	J + 1	J + 2	J + 3	.....	2J-4	2J-3	2J-2	2J-1
⋮				⋮				
			[i][j]					

A[i][j]のデータが必要な場合，その前後のデータをまとめキャッシュにあげる.

FortranではA(i,j)は$j \times IMAX + i$の順番になるので注意!!

逐次計算の高速化

- サブブロック分割



0	1	2	3	.....	J - 4	J - 3	J - 2	J - 1
J	J + 1	J + 2	J + 3	.....	2J-4	2J-3	2J-2	2J-1
⋮				⋮				

演習2

- 2つの高速化処理についてベンチマーク実行.
  （コンパイルの最適化オプションは “-O0”を指定）
  a. 垂直参照と水平参照の比較
     ex02a.cを水平参照のコードに改良せよ.

  b. 垂直参照とサブブロック分割の比較.
     ex02b.cでoffsetのサイズを変更して比較せよ.
     offset=4, 8, 16, 32, 64, 128

演習3

- 事前読込（プリフェッチ）処理
  - ex01.cではkについてのループで配列b[k][j]に毎回キャッシュミスが起こる．ex03.cでは配列b[k][j]について事前読込することで，キャッシュミスを低減させている．
  - ex01.cとex03.cを比較して事前読込処理についてベンチマーク．

※コンパイルオプションを”-O0”（最適化無し）としてコンパイルする．

熱伝達問題の差分計算

連立一次方程式の解法1

- 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 yacobi_3by3.c

連立一次方程式の解法2

- LU分解による解法

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。
（LU分解の方法は補足資料を参照．）

$$A = LU$$

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2

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2

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2

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2

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。

演習4

- サンプルコード(ex04_1.cとex04_2.c)を使って，次の連立一次方程式を解きなさい.

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

（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];
    }
  }
}
```

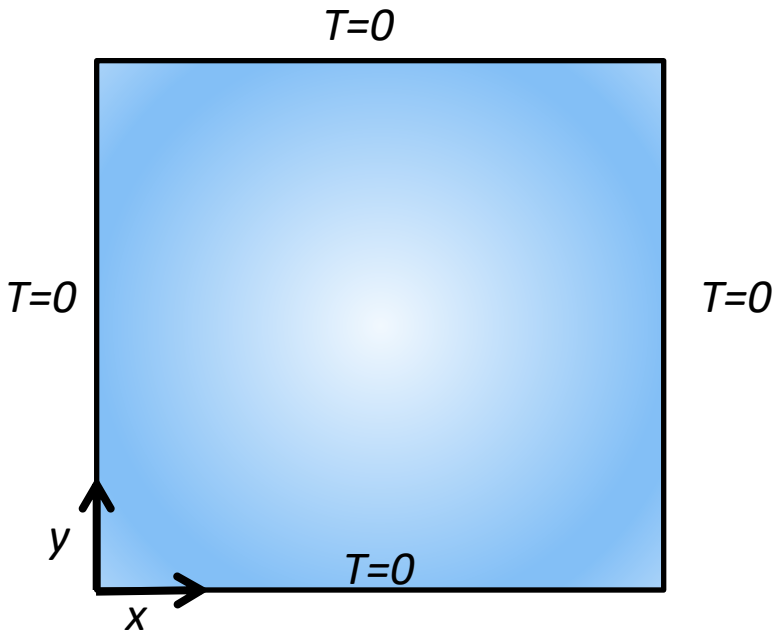
サンプルコード lu_decomp.c

※ *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。

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)

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



- 温度場を計算する。(ラプラス方程式)

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

- 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)

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

- 差分化する。

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

- $T_{i,j}$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

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

	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.

演習5

- 熱伝達問題のコードを使って以下を実施せよ.
  - SIMD命令の効果の確認.
    ベクトル化を有効にした場合(-xsse4.2, -xavx2, -xcore-avx512)と無効にした場合(-no-vec)について計算速度を比較せよ.

  - Gnuplotによる温度分布の可視化.
    gnuplot
    >splot “laplace.dat”
    >set pm3d map
    >replot

（発展）ガウス･ザイデル法

- 熱伝達問題の差分式を次の反復式によって計算する.

for i=1 to i=N-1
　for j=1 to j=N-1

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

- 反復計算（laplace.cの43行目）の右辺を
  Told → T
  に変更する.
- 反復計算の収束速度はi,jの走査方向に依存する.

サンプルコード laplace_gs.c

（発展）ガウス・ザイデル法

- ヤコビ法に比べて収束が早い.
- 後方依存関係がある.

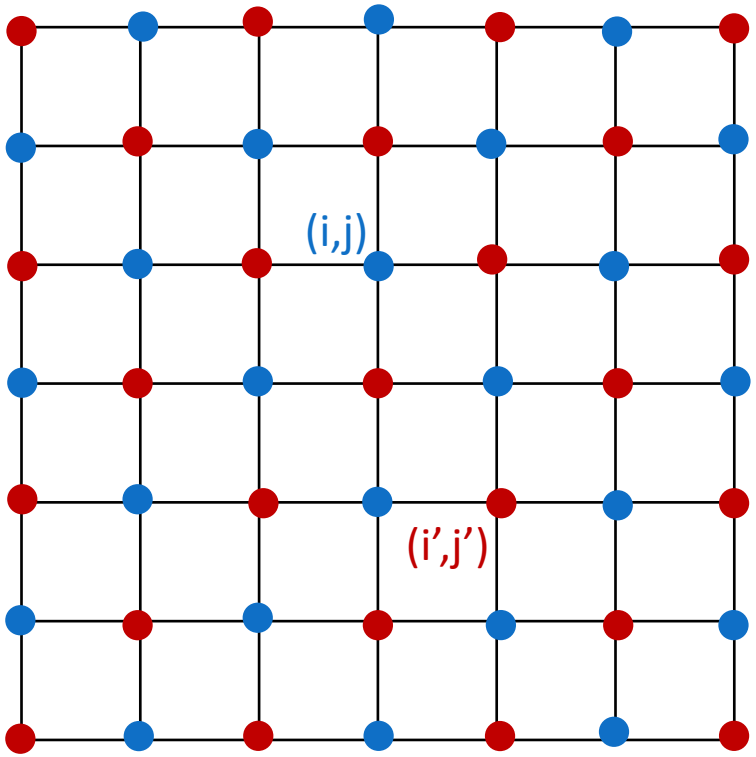
$$T_{i,j}^{(k+1)} \leftarrow T_{i-1,j}^{(k+1)}, T_{i,j-1}^{(k+1)}$$

- 依存関係がベクトル化を阻害していることが確認できる.
  icc –xhost laplace_gs.c –qopt-report
- 後方依存の関係を解消する方法を考える.

（発展）ガウス･ザイデル法

- 前方･後方依存の解消（計算順序の組み替え）

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



- はじめに青を計算する.

$$T_{i,j}^{(k+1)} \leftarrow T_{i\pm1,j\pm1}^{(k)}$$

- 次に赤を計算する.

$$T_{i',j'}^{(k+1)} \leftarrow T_{i'\pm1,j'\pm1}^{(k+1)}$$

自習問題

- 前のページで解説した計算順序の入替を実装し，修正版ガウス・ザイデル法のコードを作成しなさい．

- 元の方法と修正版の方法について，収束速度，計算速度，ベクトル化の観点で比較しなさい．