KOBE HPC サマースクール（初級）
2021 講義資料
2021/09/06

兵庫県立大学 情報科学研究科
安田　修悟

スタッフ

- 講師
  - 横川先生（神大・ECCSE），寺尾先生（理研・R-CCS），安田（兵県大・情報科学）
- アシスタント
  - 清水，中井，上本，中澤（神大）
    木村，谷口，柳生（兵県大）
- 計算機・アカウント管理
  - 島，安田（兵県大・情報科学）

講義内容

1. 計算機サーバーの環境設定と使い方　（担当　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　寺尾）
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. アムダール法則と並列化効率の評価
8. 分散メモリ型並列計算機とは何か？　（担当　横川）
9. 1対1通信関数、集団通信関数
10. 並列計算性能の評価方法
11. 熱伝導問題のプロセス並列計算
12. ハイブリッド並列
13. アクセラレータとは　（担当　安田）
14. OpenACCプログラミング
15. まとめと確認テスト

神戸大学・三宅先生から提供して頂いた資料

Zoom講義の進め方

Zoom基本機能を確認しましょう！

1. 聴講中（発言時以外）はミュートしましょう.

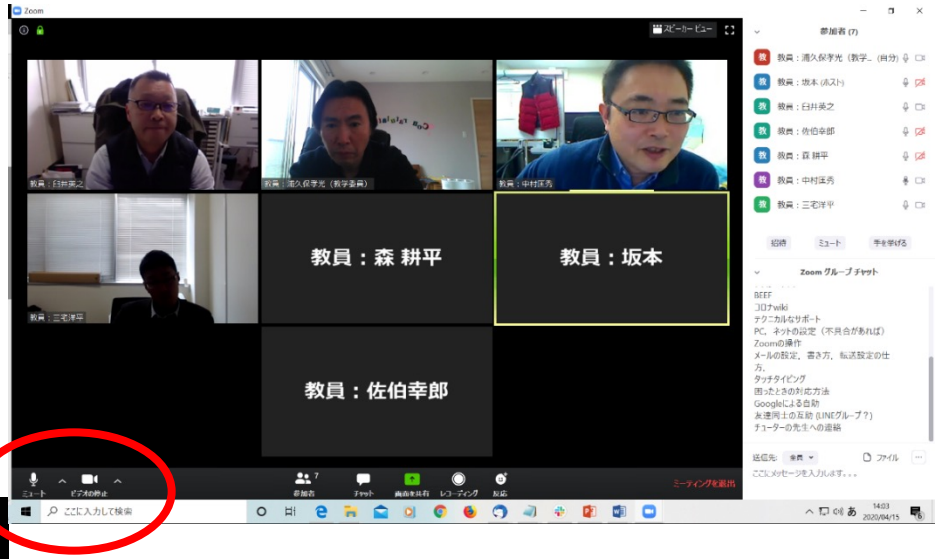
2. 参加者は名前を，「受講番号（2桁）_氏名」として下さい.（入室後に「名前の変更」機能で修正可能です）.

3. 質問や声が聞こえないなどの場合は，チャットで伝えてください.
   遠隔授業TAが対応します.「全員宛て」にして書き込んでください.

4. 「挙手」機能を確認しましょう.

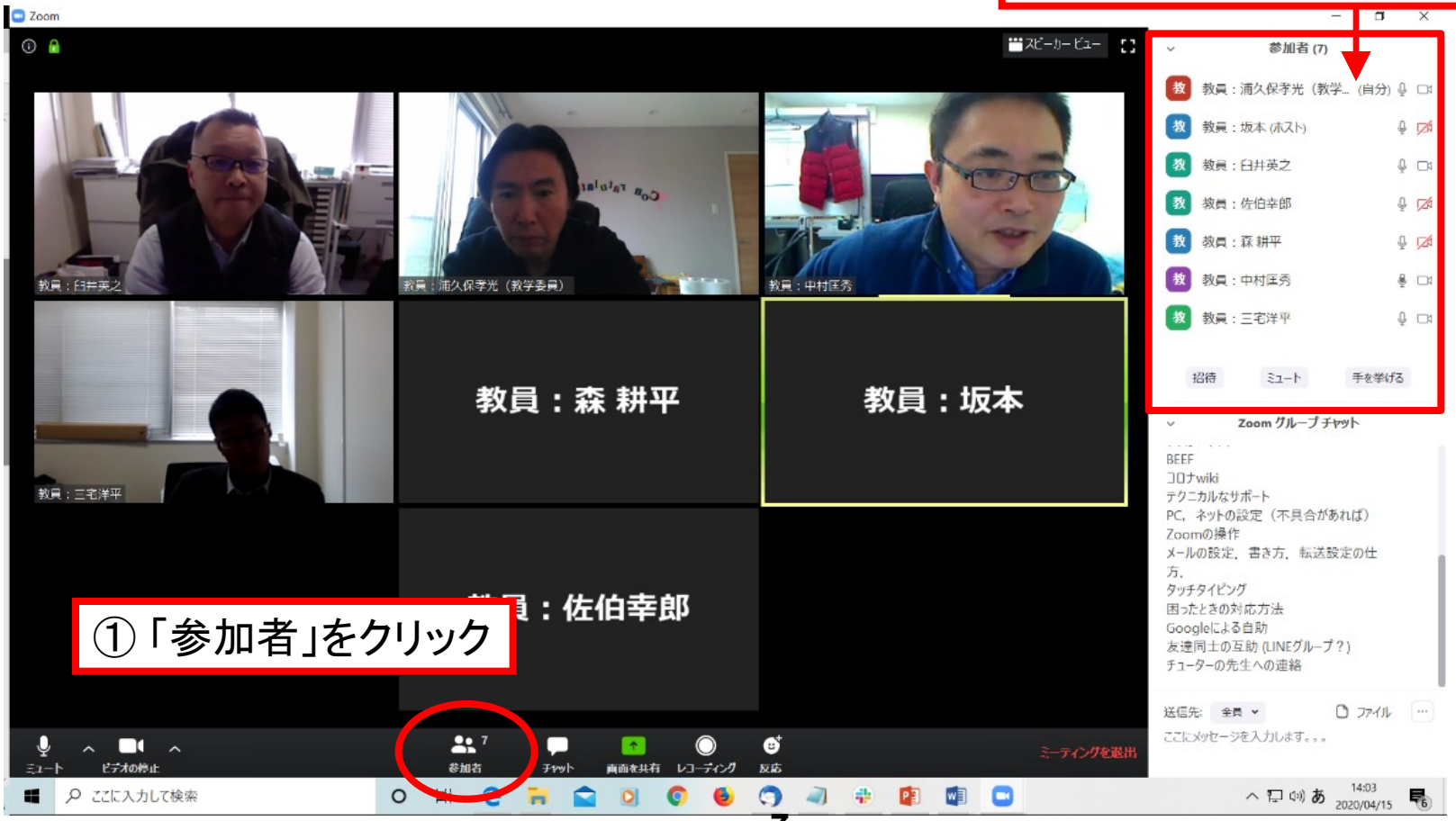
1. ミュートしましょう.
発言するときミュートを解除してください.
発言終了時には再びミュートしてください.

それぞれクリックすることで,
・ミュート / ミュート解除
・ビデオ開始 / ビデオ停止
を切り替えられます.



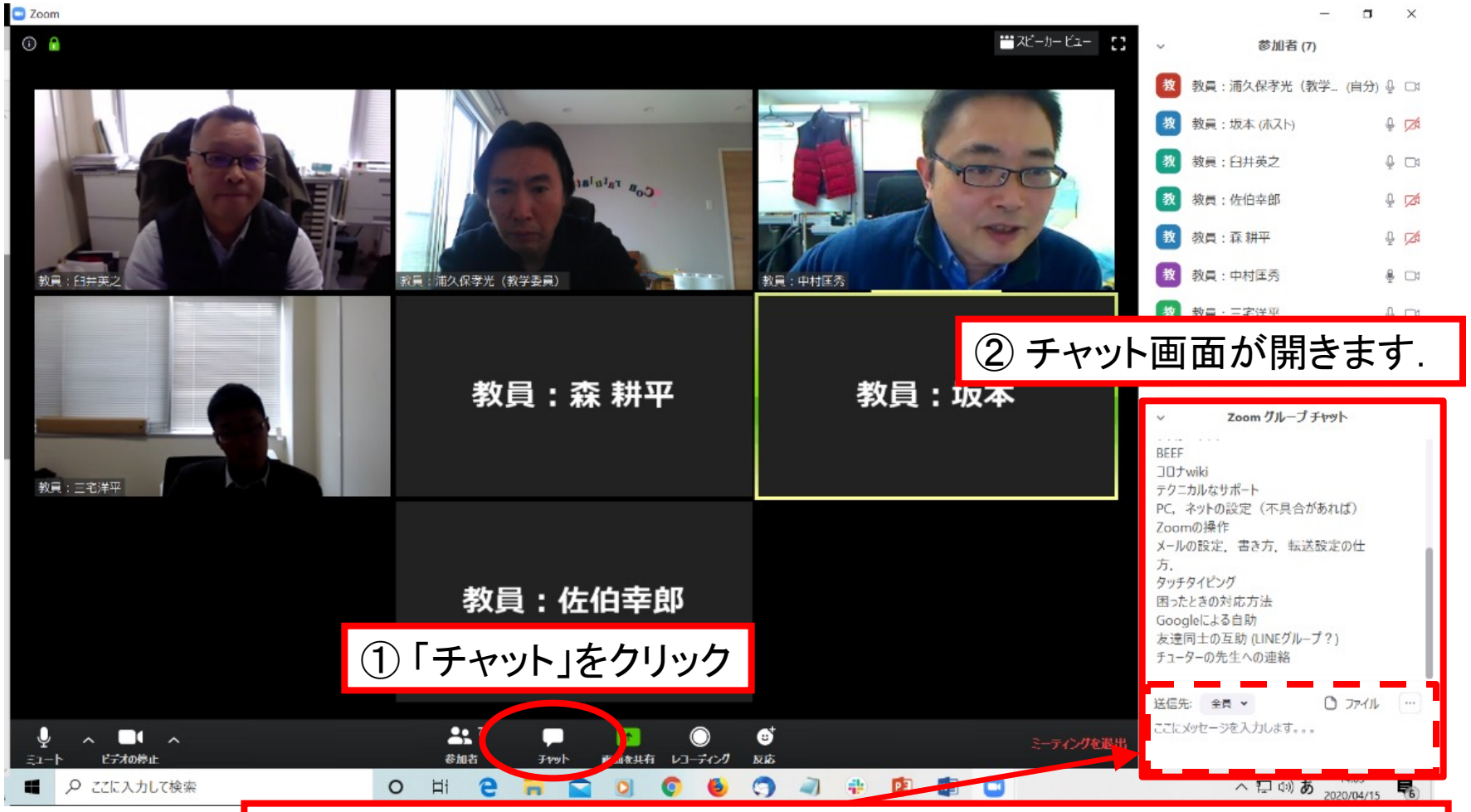
2. 自分の氏名を確認しましょう.

② 参加者のリストの中に
＊＊＊（自分）が表示されます.



2. 「名前の変更」が必要な場合.　（手順2）

3. 声が聞こえないなどの場合は，チャットで伝えてください．



4. 「挙手」機能を確認しましょう.

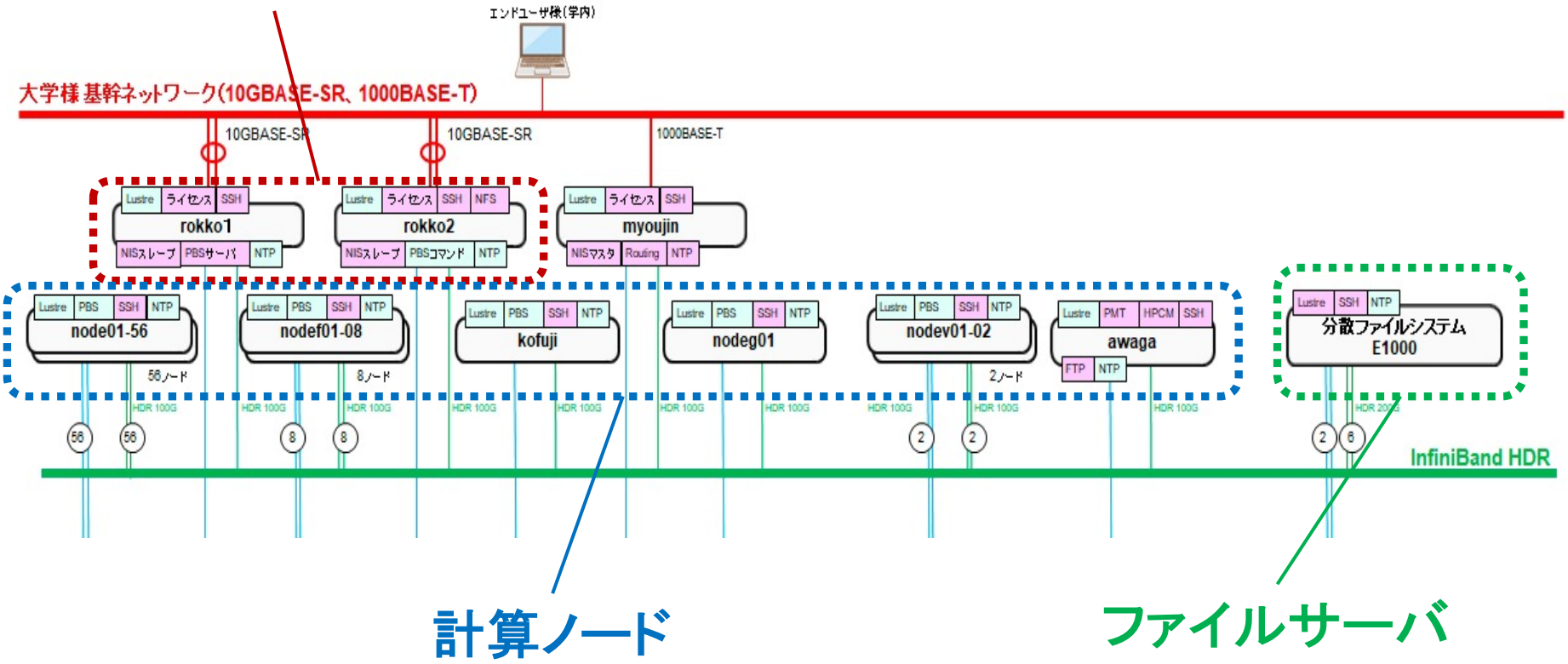
「手を挙げる」をクリック
同様に「手を降ろす」ことができる

質問がある時

1. 直接口頭で質問 or 挙手をしてください.
2. 講師又はTAが応答します.
3. 必要に応じてTAと個別のチャットで対応します.
4. さらに必要に応じて，ブレークアウトルーム（BOR）へ誘導します.

計算機資源について

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

- アカウント名
  “guest受講者番号（2桁）”となっています.
  例： guest01, guest31, ...
- ログインパスワードはアカウント名と同一です.
- インターネットによる接続方法は島先生による資料を参考にしてください.

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
  - 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_rev1.2」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
```

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. エラーが無ければOK.

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/khpc2021/20210906/sample.sh

```
#!/bin/bash
#PBS␣-q␣S                          ・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 sample job.”           ・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.

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␣T**　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 T
  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.

（補足）環境構築

- GUI(Graphical User Interface) と CUI(Character User Interface)
  - スパコンではCUIの操作が必須。

- Linuxコマンド
  - WindowsではCygwinとMacではターミナルを使う。

Cygwinのインストール（Windows）

a. http://www/cygwin.com/からインストーラをダウンロード
b. インストーラを起動。「Chose Installation Type」で[「Install from Internet」を選択。Cygwin Mirror Sitesにある日本のサイト（例えば、http://ftp.jaist.ac.jp）を選択し、「次へ」。
c. パッケージの取得。「Select Package」で「View」を「Full」に設定、「Search」を使って、”gcc-core”と”nedit”を探す。“gcc-core”と”nedit”の「New」の欄のプルダウンを使って、「Skip」を最新のバージョンに変更。
d. 次に「View」を「Category」に変更して、表示される一覧から”X11”のプルダウンを”Install”に変更。
e. 「次へ」をクリックしてインストールの完了を待つ。

Cygwinのインストール（Windows）

f. インストールが完了したら、Cygwinを開いて、ターミナルに
gcc –v
と入力。バージョンなどの情報が返ってきたらOK。

g. Cygwinのサイトからダウンロードしたインストールファイル（setup-x86_64）を使うと、Packageの追加やアンインストールなどが出来るのでそのまま残すと便利。

homebrewによる環境構築（Mac）

a. Homebrewのインストール。以下参考サイト。
   https://brew.sh/index_ja
   https://fukatsu.tech/homebrew
   https://qiita.com/pypypyo14/items/4bf3b8bd511b6e93c9f9

b. AquaTermのインストール。
   https://sourceforge.net/projects/aquaterm/

c. X11のインストール。https://www.xquartz.org

homebrewによる環境構築（Mac）

f. ターミナルで次を入力してgccをインストール。
brew install gcc

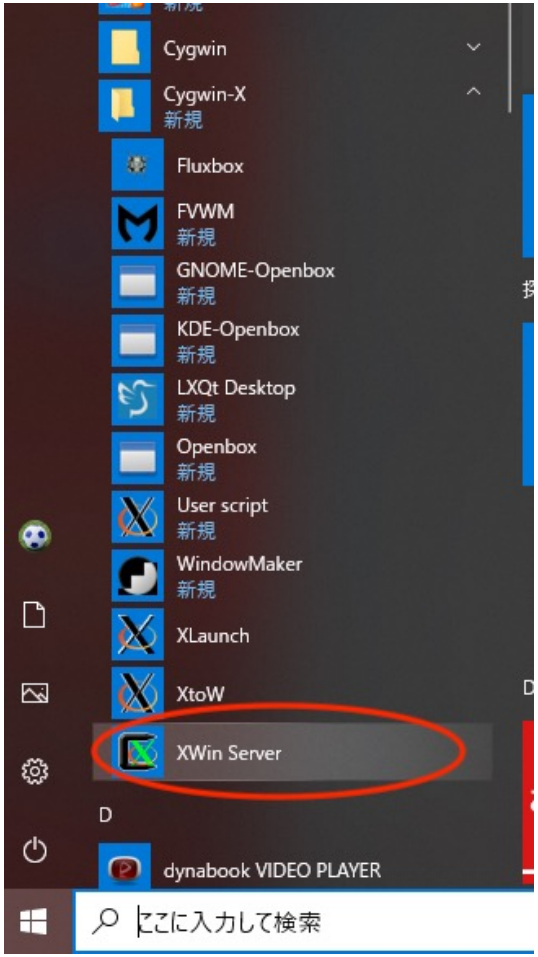
g. インストールが完了したら、ターミナルに”gcc -v”と入力して、バージョンなどの情報が表示されればOK。

h. 次に、neditをインストール。
brew install nedit
ターミナルに”nedit “と入力してエディタが起動すればOK。

Cygwin

Cygwinを使おう

スタートメニュー
→ Cygwin-X
→Xwin server
でXサーバを起動



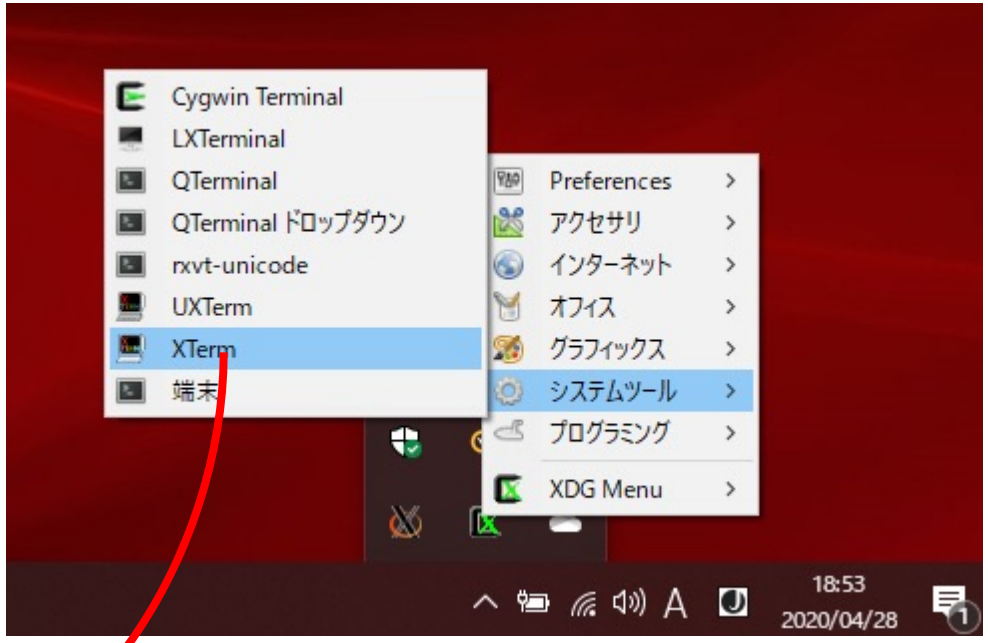
Cygwin

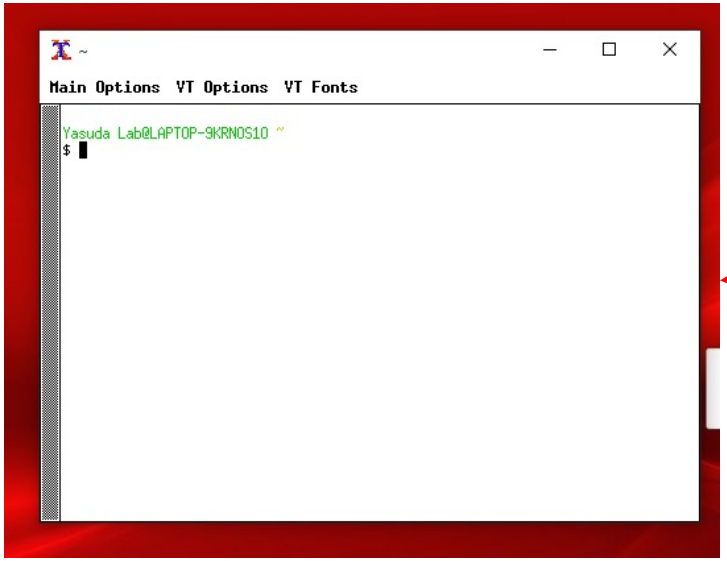
Xtermの起動

Xwin Server

→ システムツール

→ Xterm



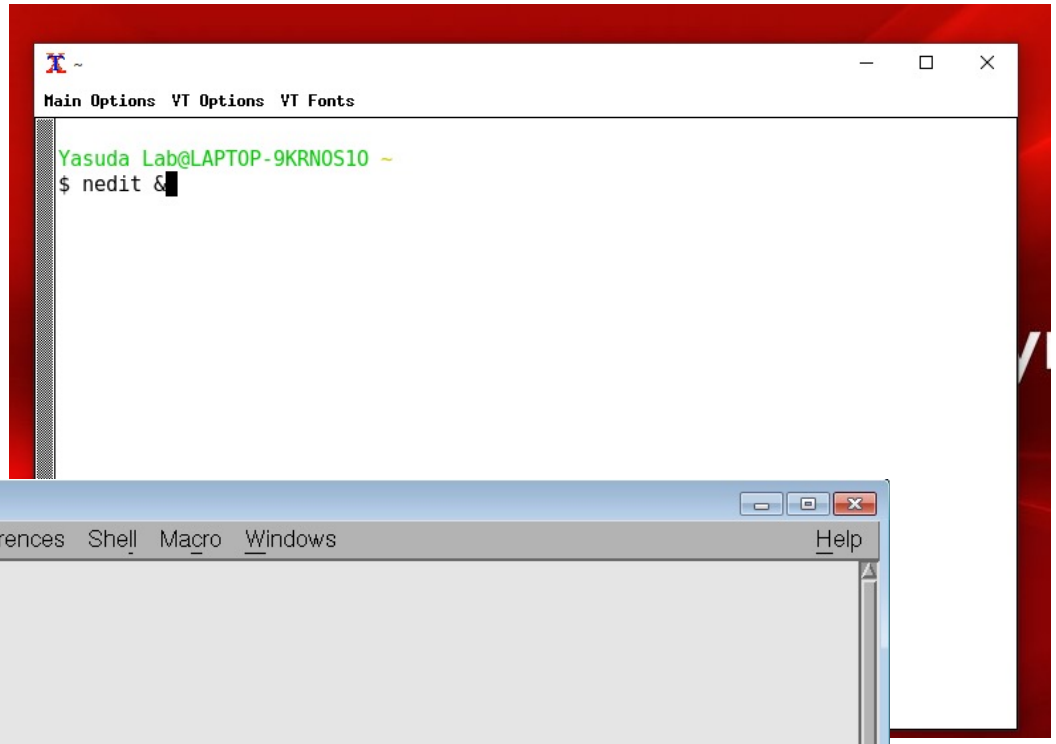


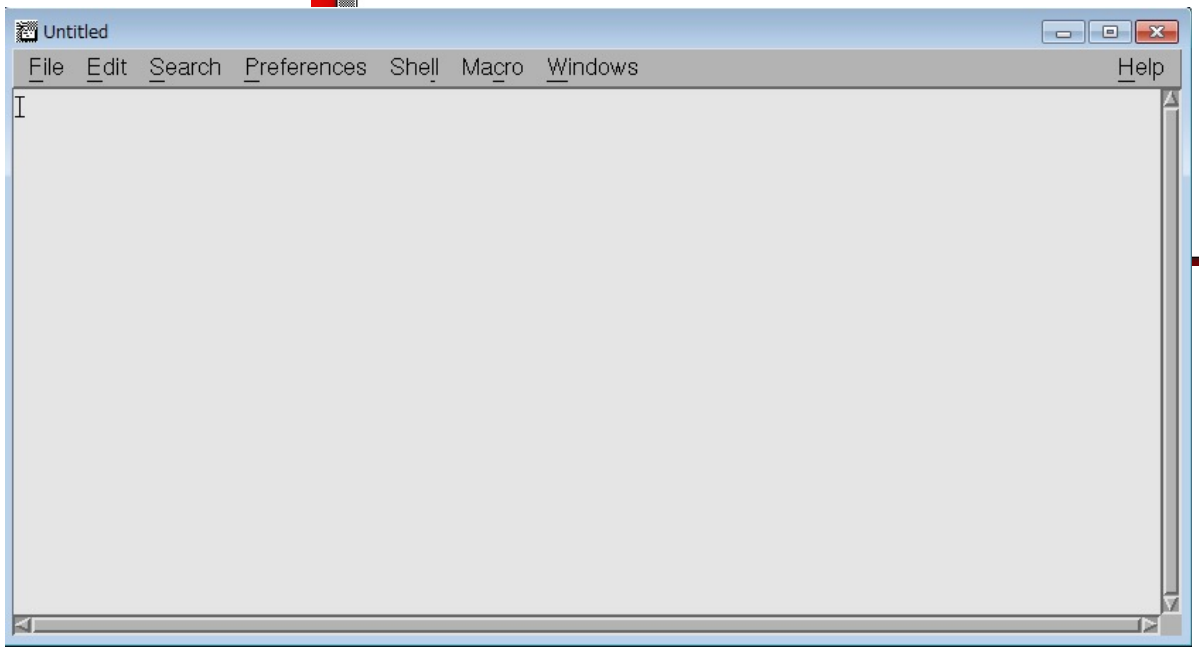
Cygwin エディタの起動

この中で

nedit␣&⏎

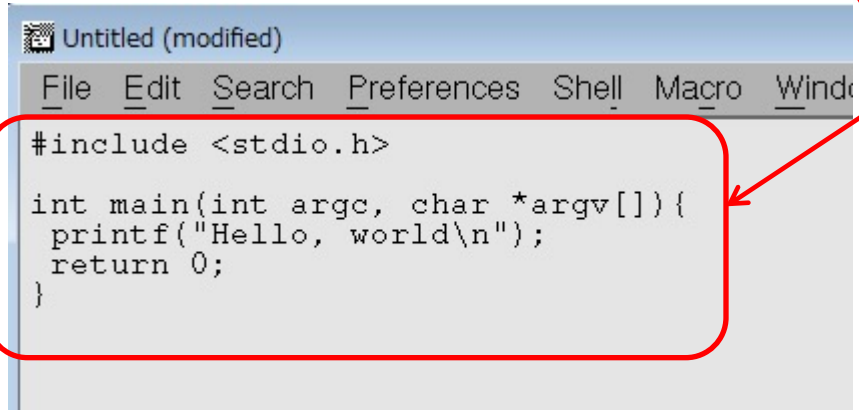
と入力。





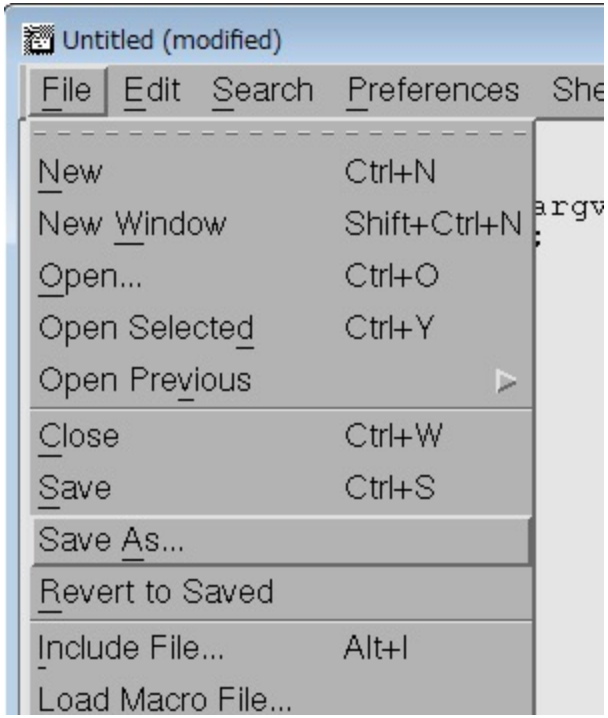
入力・コンパイル・実行

```
#include␣<stdio.h>⏎
⏎
int␣main(int␣argc,␣char␣*argv[]){⏎
 printf("Hello,␣world¥n");⏎
 return␣0;⏎
}⏎
```



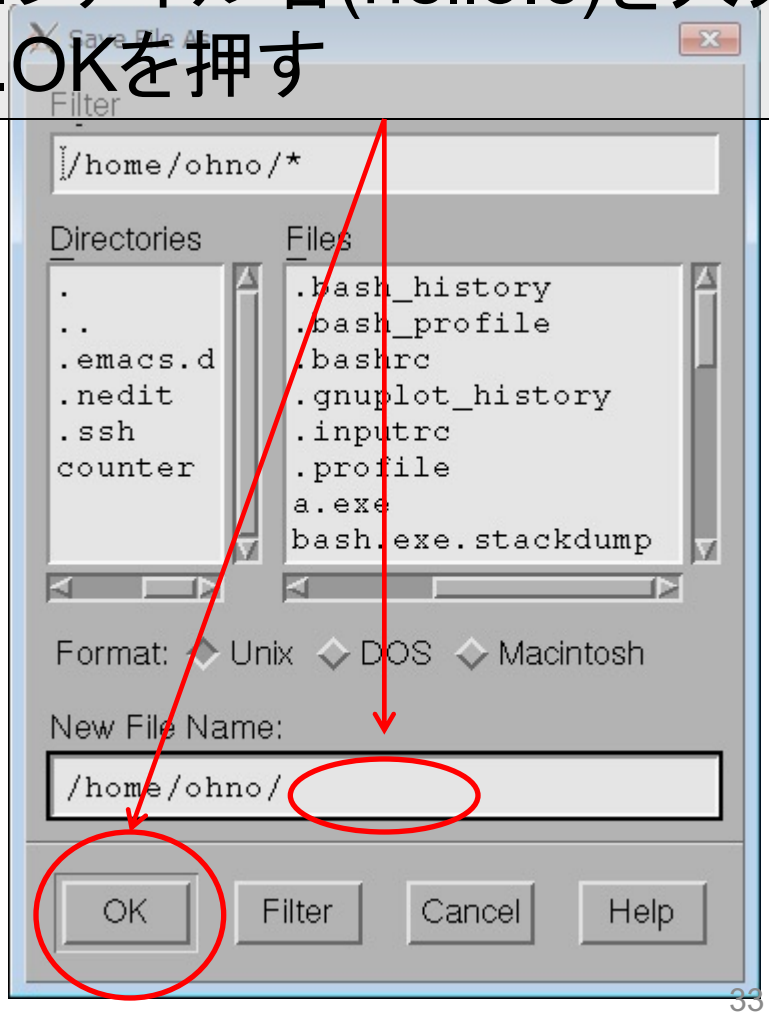
入力しよう
⏎:改行を表す
␣:スペースを表す
エディタの中では
¥はバックスラッシュになる

入力・コンパイル・実行



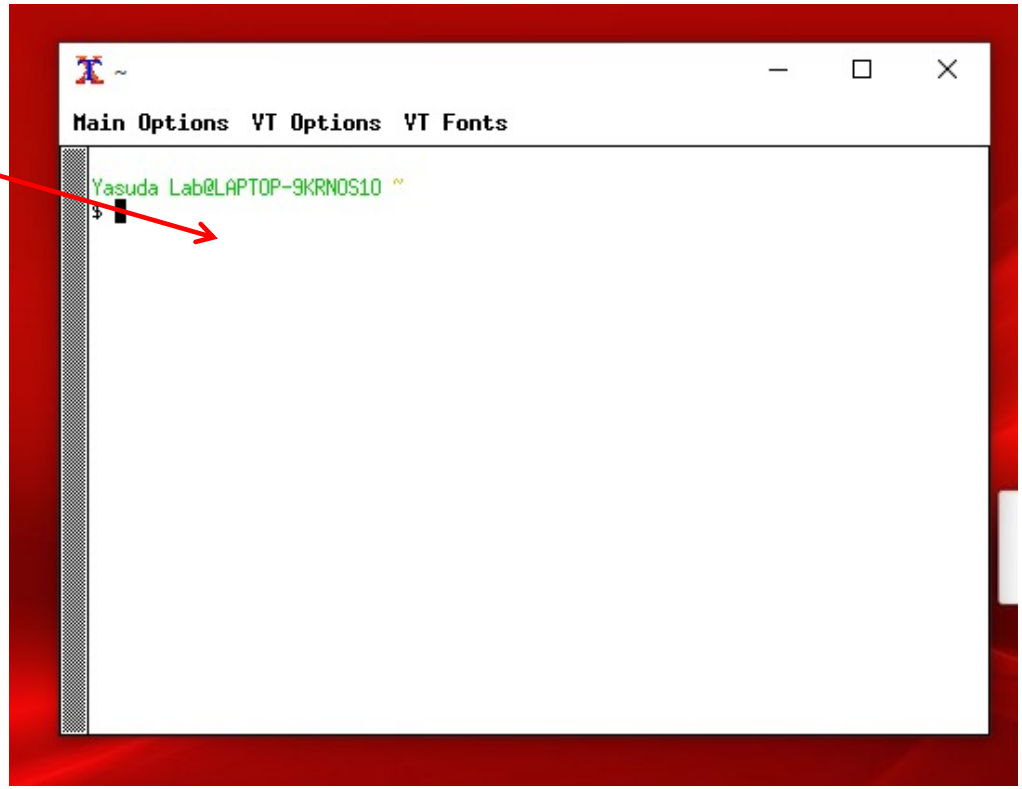
保存しよう
File -> Save As

1.ファイル名(hello.c)を入力し
2.OKを押す



入力・コンパイル・実行

この窓の中で、
gcc␣hello.c⏎
とすると、a.exeという
実行ファイルが出来る。

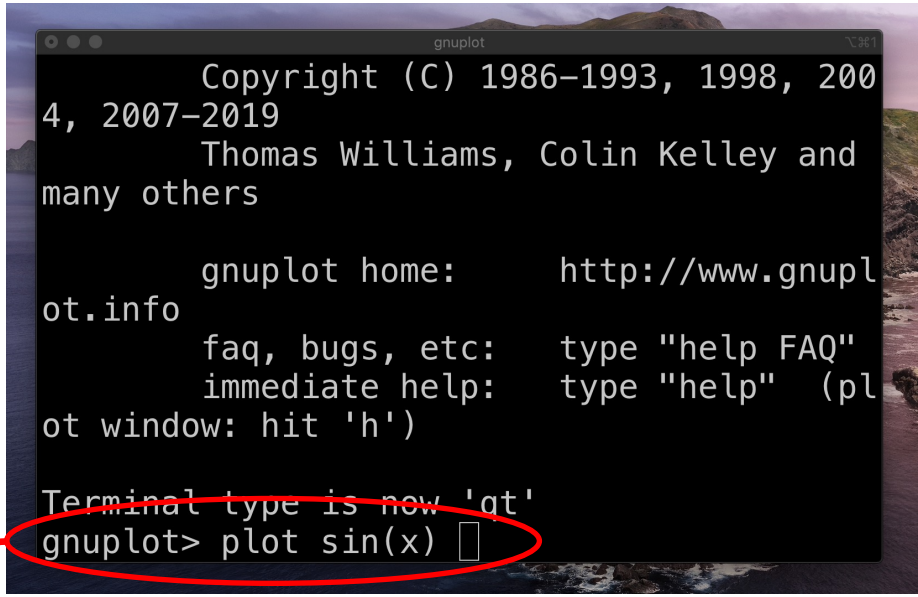


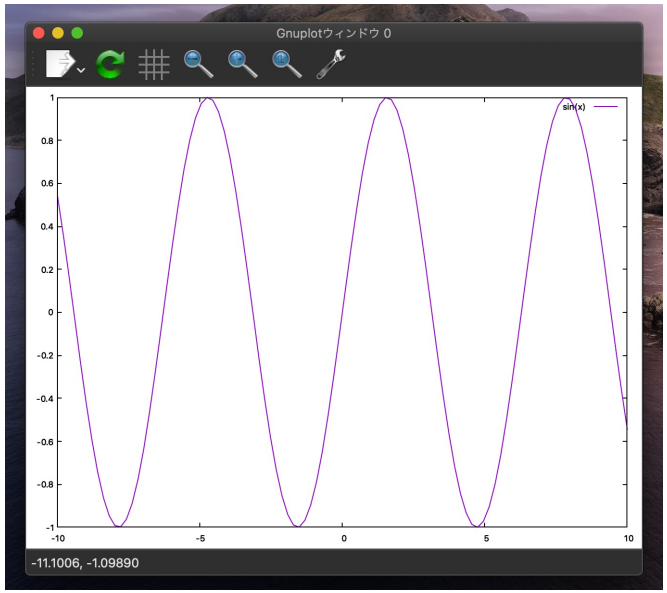
これを実行するには
./a.exe⏎

普通のLinuxでは、a.exeではなく、a.outという実行ファイルができる

Gnuplotの起動

- CygwinのsetupファイルまたはHomebrewを使ってgnuplotをインストール。
- ターミナルでgnuplotを起動して、plot␣sin(x) ⏎





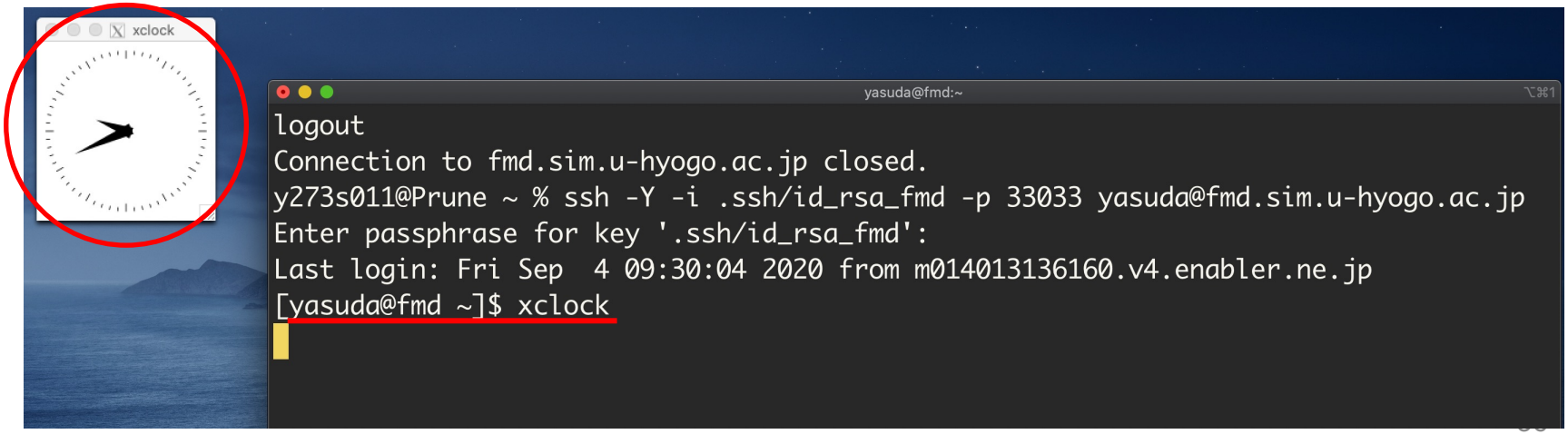
Macの場合

x11アプリ

- x11転送
  - Cygwin/X (Win)又はXquartz (Mac)を起動
  - ”-Y”オプション

ssh –Y –i ~/.ssh/id_rsa_guest40 –p 33033 guest40@fmd.sim.u-hyogo.ac.jp

- サーバーでx11アプリケーションを起動
  - xclock



トンネリング

- fmdに接続する時に下記オプションを追加.

  -L (ローカルポート指定):172.25.61.33:22

- fmdから見えるIPアドレス172.25.61.33の22ポートと指定した(ローカルポート)が繋がります.

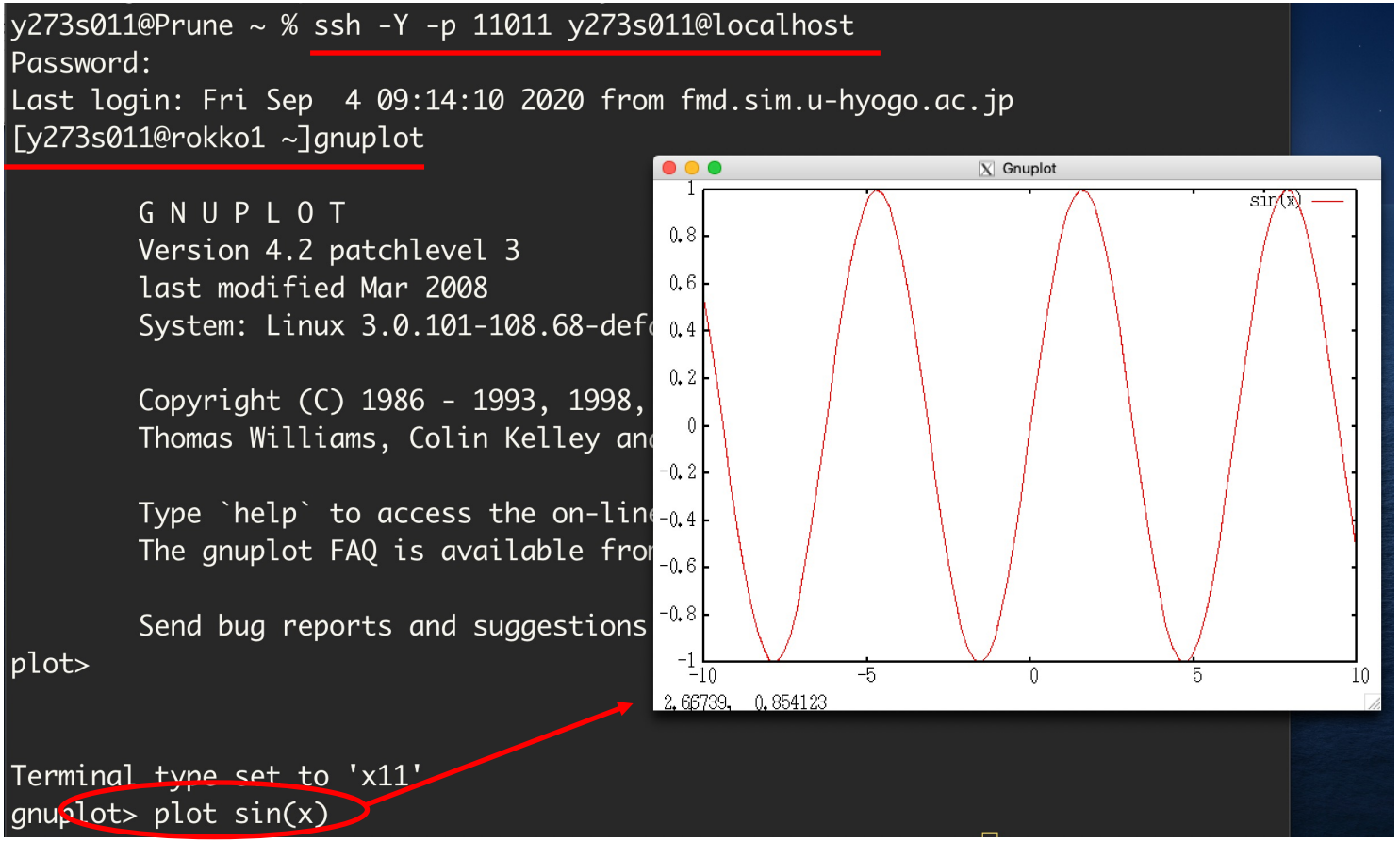
```
y273s011@Prune ~ % ssh -Y -i .ssh/id_rsa_fmd -p 33033 -L 11011:172.25.61.11:22 yasud
a@fmd.sim.u-hyogo.ac.jp
Enter passphrase for key '.ssh/id_rsa_fmd':
Last login: Fri Sep  4 09:39:28 2020 from m014013136160.v4.enabler.ne.jp
[yasuda@fmd ~]$
```

ローカルポート11011番とrokko1の22番ポート(172.25.61.11:22)が踏み台サーバをトンネルして繋がります.

- 接続を確保するため xclockでも起動させておく.

トンネリング

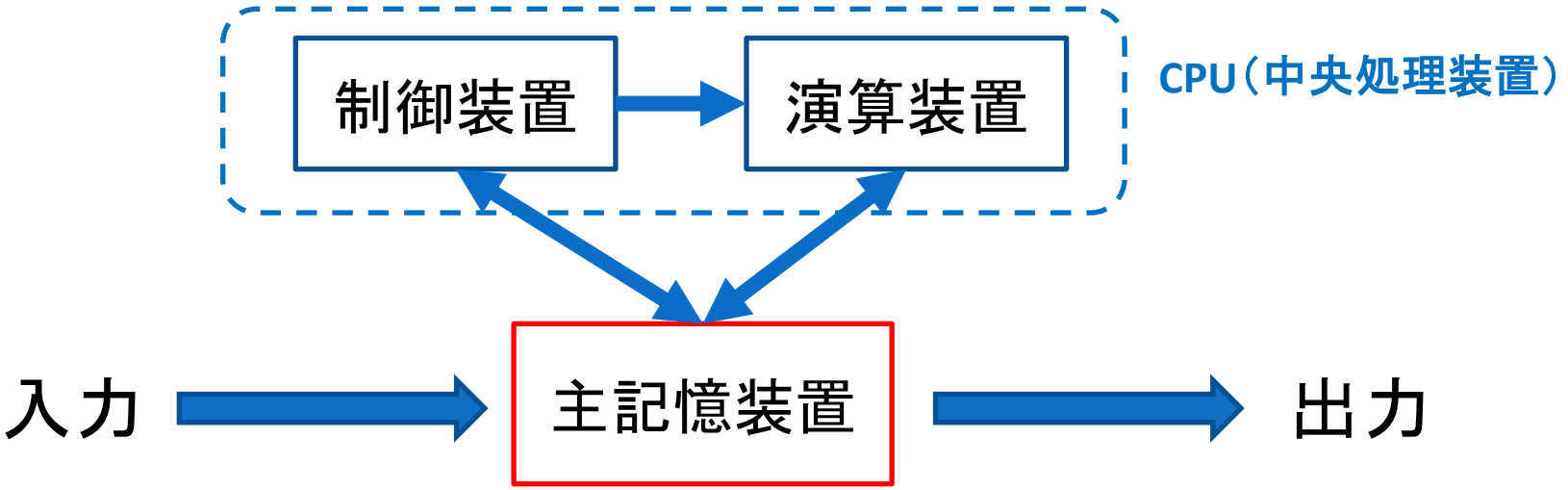
- ターミナルを起動して，
  ssh –Y　アカウント@localhost –p 11011
- rokko1のx11アプリを起動.



シリアル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

メモリ

命令1	命令2	データ	命令3		命令4	命令5	データ	データ	データ
番地 0	1	2	3	4	5	6	7	8	9 ....

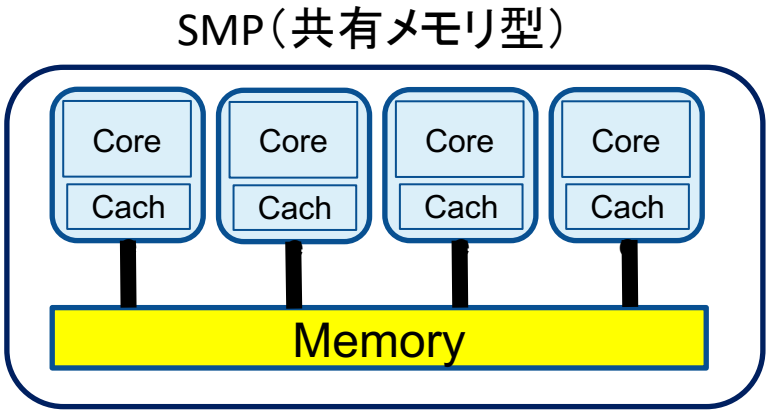
プログラムカウンタ －＞ 実行する命令の番地を示す

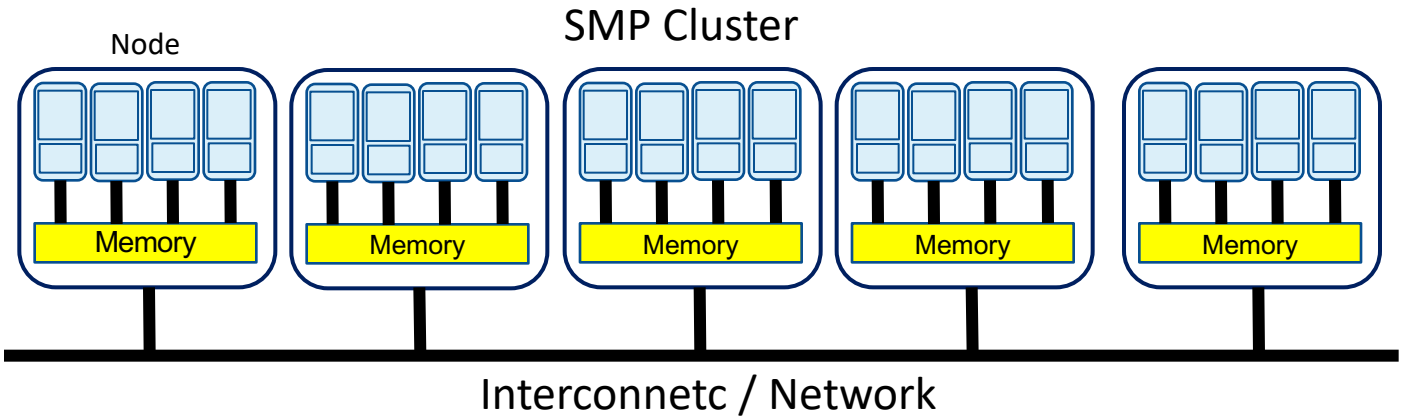
0 1 3 5 ~~6~~1

※分岐命令でカウンタを書き換える.

並列計算機

- 並列計算機の種類





並列計算

- 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
  - プロセス並列
    - 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する。
    - プロセス間にデータ転送が必要。
  - スレッド並列
    - 分割された演算（スレッド）は、メモリ空間を共有できる（複数のスレッドで変数を共有できる）。
    - 各スレッドは他のスレッドからはアクセスできない固有の変数も用意できる。

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];
        }
```

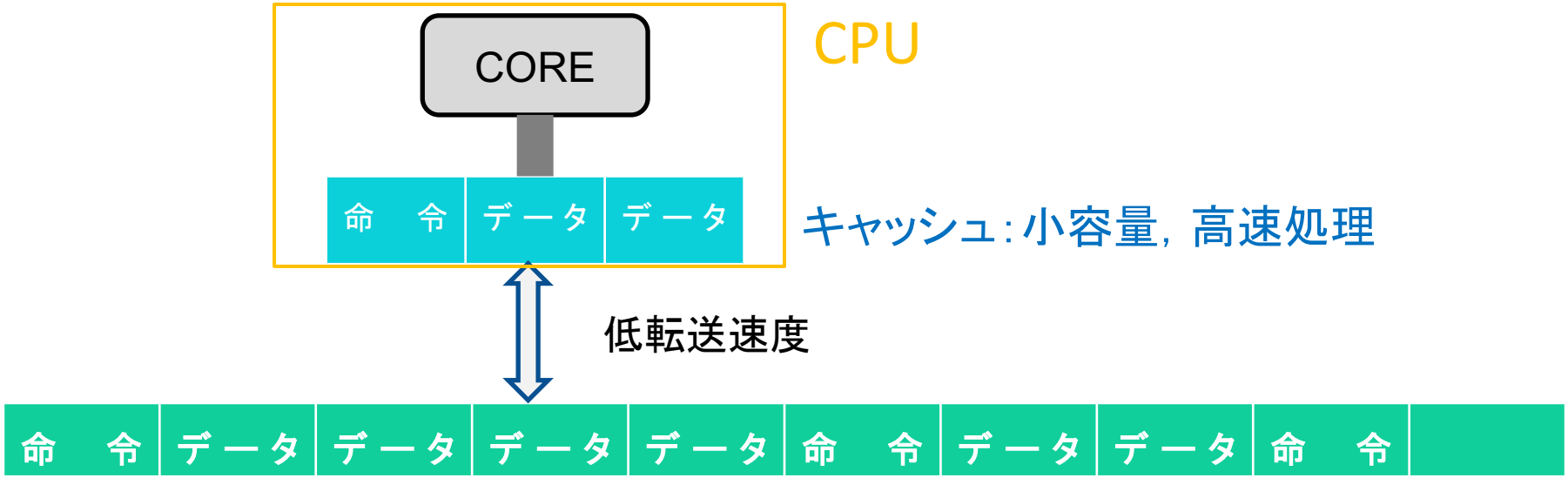
演習1 (ex01.c)

- 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

- キャッシュメモリ
  - データアクセスの時間を短縮する.



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

1. 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.

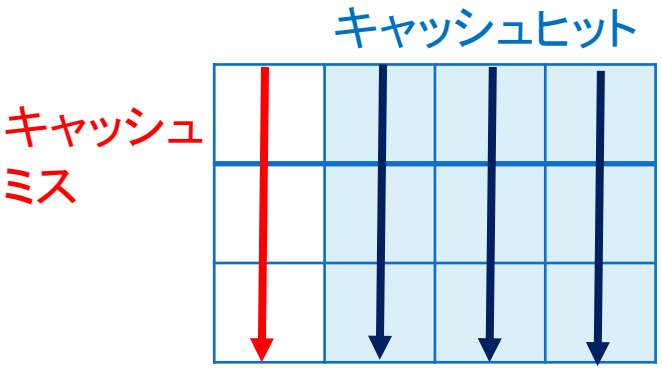
0	1	2	3	.....	J - 4	J - 3	J - 2	J - 1
J	J + 1	J + 2	J + 3	.....	2J-4	2J-3	2J-2	2J-1
⋮				⋮				
			[i][j]					

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

2. 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];
    }
  }
}
```

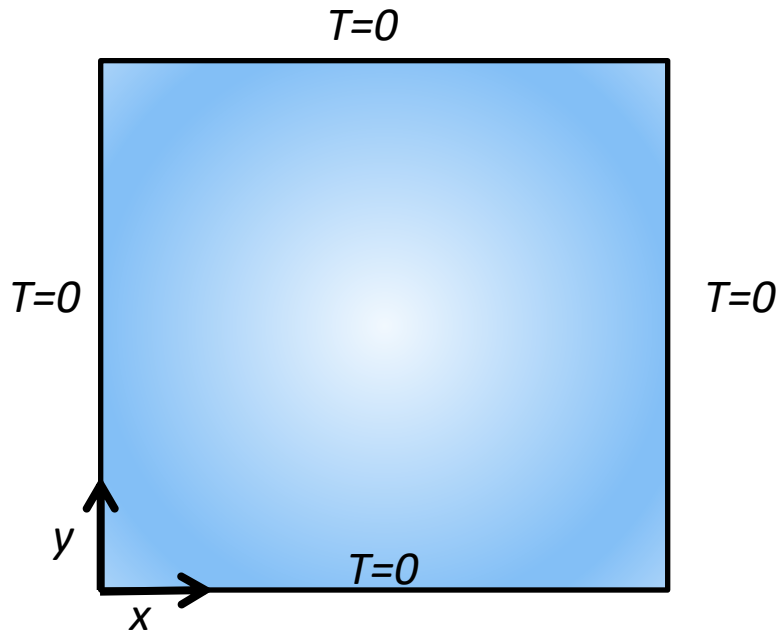
サンプルコード lu_decomp.c

※ *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)

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



- 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。

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

- $T_{i,j}$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

(補足)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

(補足)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.

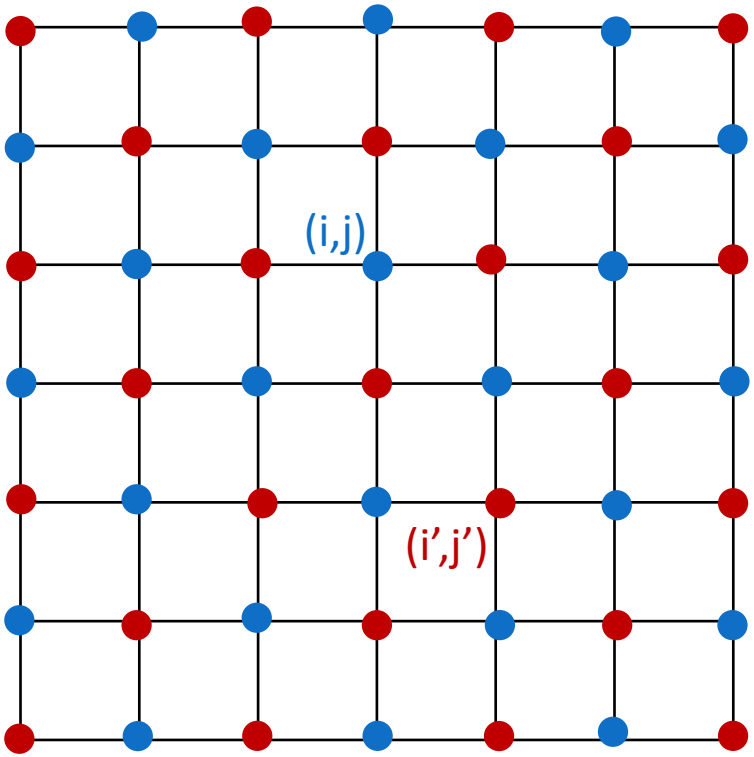
$$T_{i,j}^{(k+1)} \leftarrow T_{i-1,j}^{(k+1)}, T_{i,j-1}^{(k+1)}$$

- 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.

(補足)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．