KOBE HPC サマースクール（初級）
2019 講義資料
2019/08/25

兵庫県立大学シミュレーション学研究科
安田　修悟

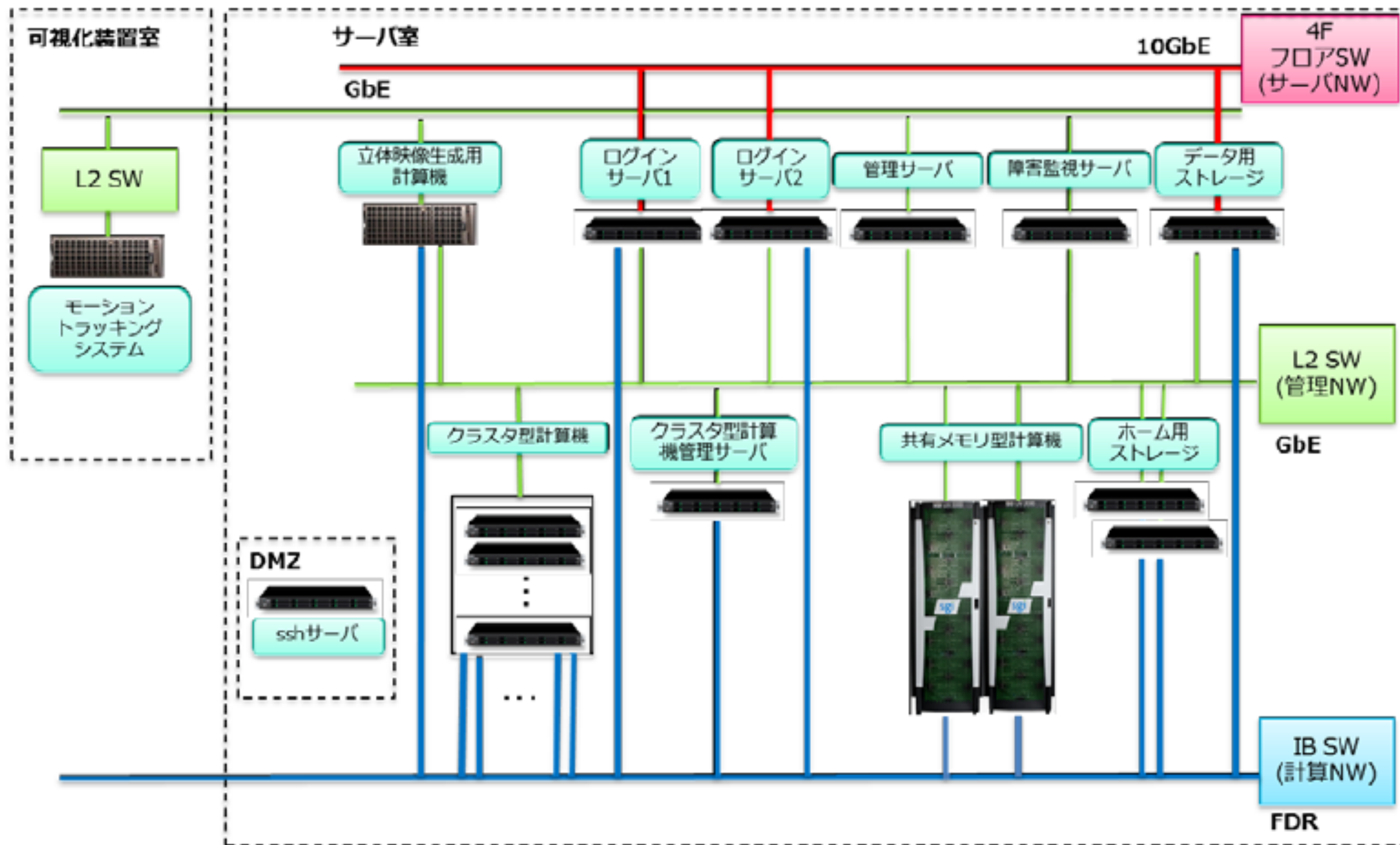
講義内容

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. アクセラレータとは
5. OpenACCプログラミング
6. スレッド並列とは　（担当　八木）
7. OpenMPによるループ処理の並列化
8. 差分化された偏微分方程式の並列化
9. アムダール法則と並列化効率の評価
10. 分散メモリ型並列計算機とは何か？　（担当　横川）
11. 1対1通信関数、集団通信関数
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. まとめと確認テスト　（担当　安田）

利用環境

- 注意事項
  - 飲食禁止。
  - 指定の端末を利用。
- 起動とログイン
  - 端末PCの電源ON後，Linuxを選択。
  - 「ユーザーID」と「パスワード」でログイン。
- ブラウザ（Fire Fox)設定
  - 詳細 -> ネットワーク
  - Proxy: 172.25.50.4: 8080

兵庫県大・シミュレーション学研究科 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 [172.25.61.11] rokko2 [172.25.61.11]	SGI Standard-Depth Server C1110-GP2	Intel Xeon E5-2667v3 3.2GHz (8coresx2)	128GB
クラスタ型（通常ノード）	r03n01~r03n28, r04n01~r04n24	SGI Standard-Depth Server	Intel Xeon E5-2650v3 2.3GHz (10cores x2)	128GB
クラスタ型（mic搭載ノード）	r03p01~r03p08		Xeon Phi 5110P (1.053GHz 60cores /240threads)	8GB
クラスタ型（gpu搭載ノード）	r04g01~r04g08		NVIDIA Tesla K40m (754MHz 192cores x15)	12GB
共有メモリ型	kofuji1 kofuji2	SGI UV3000	Intel Xeon E5-4627v3 2.6GHz (10coresx8)	2TB (256GB x8)

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.
- 接続用アドレス
  IP: 172.25.51.93, Port: 50001 => rokko1 （番号奇数）
  IP: 172.25.51.93, Port: 50002 => rokko2 （番号偶数）
- 接続方法（例：受講者番号2）
  ssh␣-Y␣guest02@172.25.51.93␣-p␣50002
- ファイル<hello.c>をサーバのホームディレクトリにコピー.
  scp␣-P␣50002␣./hello.c␣guest02@172.25.51.93:~/

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
- その他module設定について
  「利用者マニュアル_rev2.0」(UserMaual.pdf)
  の13頁参照

プログラミング環境設定

1. “.module”ファイルの作成
   > gedit␣.module

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

2. “~/.bashrc”の最下部に次の1行を追記.
   > gedit␣.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」37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」15頁)
  - インタラクティブジョブ
    qsub␣-I
  - ジョブスクリプトの利用
    qsub␣<ジョブスクリプト>

PBSジョブ管理システム

- バッチスクリプトの作成
  gedit␣single_run.sh

```
#!/bin/bash
#PBS␣-q␣S                                    ・キューの指定
#PBS␣-l␣select=1:ncpus=1:mem=1gb             ・計算リソースの指定
#PBS␣-N␣single_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
source␣/etc/profile.d/modules.sh             ・moduleを利用する為の環境設定
module␣load␣intel                            ・Intelコンパイラの環境設定
cd␣${PBS_O_WORKDIR}                          ・投入ディレクトリに移動
dplace␣./hello.out                           ・dplaceコマンドの指定
```

- ジョブ投入
  qsub␣single_run.sh

PBSジョブ管理システム

- キュー構成

表 3 キュー編成

キュー名		ノード/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 ノード	30min	クラスタ通常ノード		150	可
C	S	1-4 ノード	24h	クラスタ通常ノード		130	可
	M	5-10 ノード	24h	クラスタ通常ノード		90	
	L	11-52 ノード	24h	クラスタ通常ノード		70	
	H	53-68 ノード	24h	クラスタ全ノード	GPU 8 Phi 8	50	
LONG		1-4 ノード	168h	クラスタ通常ノード		130	
G		1-2 ノード	24h	GPU 搭載ノード	GPU 2	130	可
GL		3-8 ノード	24h	GPU 搭載ノード	GPU 8	70	
P		1-2 ノード	24h	MIC 搭載ノード	Phi 2	130	可
PL		3-8 ノード	24h	MIC 搭載ノード	Phi 8	70	
SMP1		1-80 コア	24h	共有メモリ kofuji1		130	可
SMP2		1-80 コア	24h	共有メモリ kofuji2		130	可
SMP-LONG		1-80 コア	168h	共有メモリ kofuji2		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␣single_job.o<ジョブID>
- 実行ジョブの確認（「利用者マニュアル」31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」32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(S or T)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実行
    dplace␣./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!!

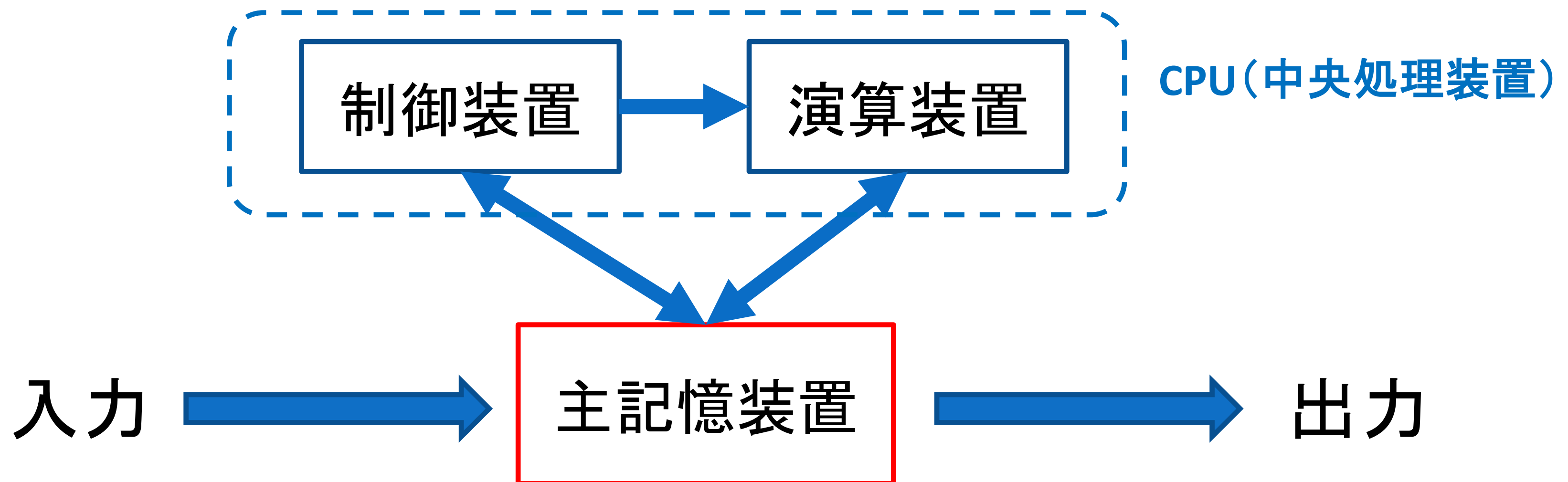
演習0

- 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 *任意の文字列*

- バッチスクリプトを使って実行する.

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.

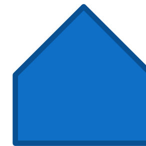
計算機の動作原理

- チューリングマシン

1．無限に長いテープ

・・・・・・・ ・・・・・・・

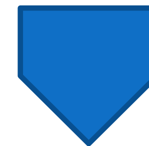
  - テープの情報の読み書き.
  - 機械の内部状態の変更.
  - ヘッドの移動.



2．ヘッダ（情報の読み書き）

3．内部状態メモリ

- ノイマン型



メモリ

番地　0　1　2　3　4　5　6　7　8　9　....

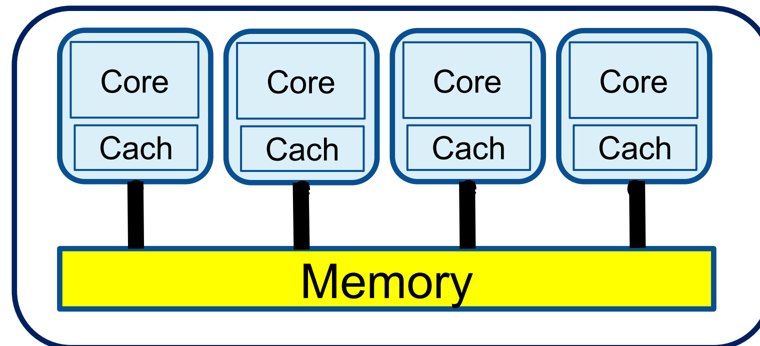
プログラムカウンタ　->　実行する命令の番地を示す

0 1 3 5 ~~6~~1　　※分岐命令でカウンタを書き換える.

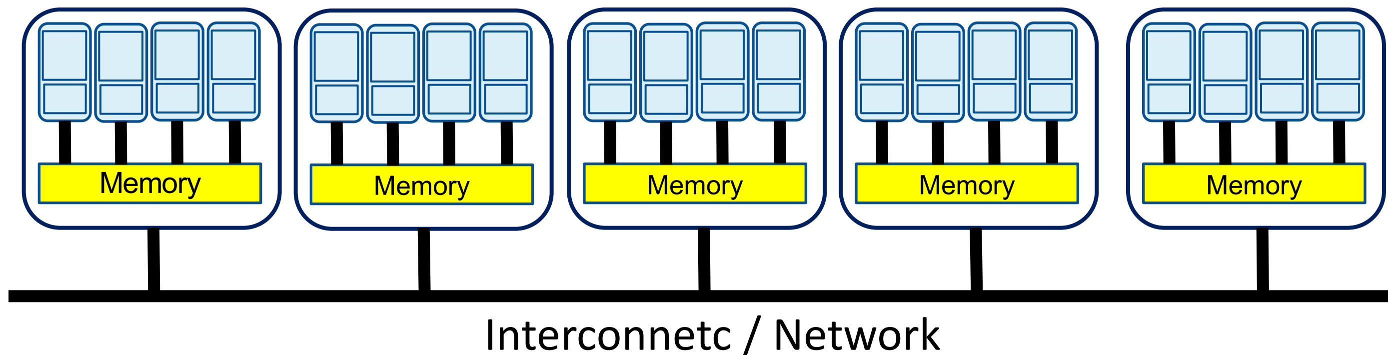
並列計算機

- 並列計算機の種類

SMP（共有メモリ型）



SMP Cluster



Interconnetc / Network

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];
        }
```

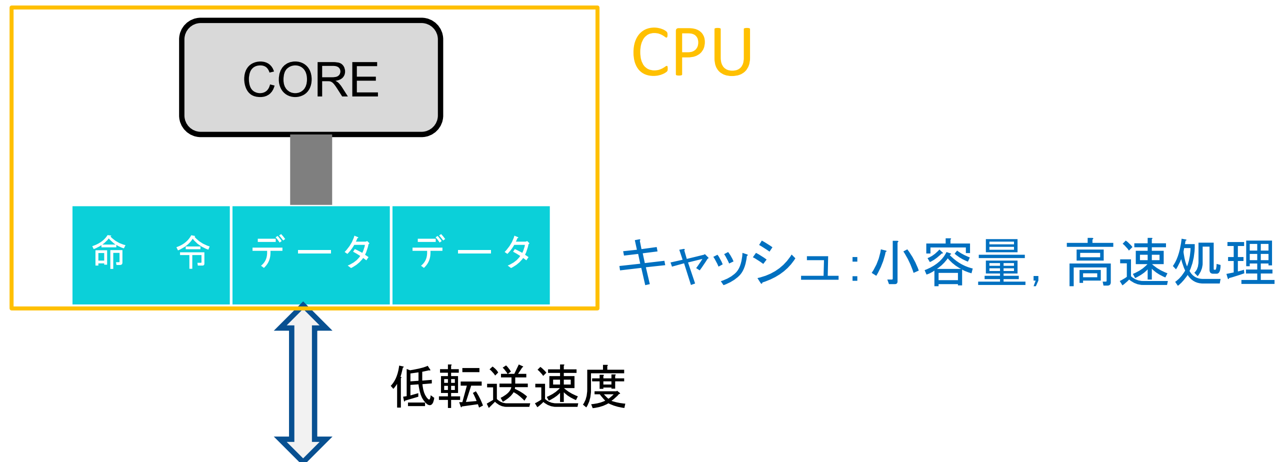
演習1 (ex01.c)

- 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.



命　令　データ　データ　データ　データ　命　令　データ　データ　命　令

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	J2-1
⋮				⋮				
			[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

キャッシュヒット

キャッシュミス

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	J2-1
⋮				⋮				

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.

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．