

# スパコンに通じる 並列プログラミングの基礎

---

宮武勇登（大阪大学 サイバーメディアセンター）

2020年9月2日

# はじめに：この講習会の対象者

プログラミング/数値計算の経験はあるけれど、  
スパコンや並列計算にも興味がある、 という方

- ① 普段はWindowsやMac
- ② 研究・開発などで大規模計算が必要になりそう
- ③ 少しでも計算が速いと嬉しい
- ④ とはいっても、並列計算の全体像がよく分からない  
どのような技術？難しいの？コストは？
- ⑤ 普通のPCで試せることがある？

# より詳しく知りたくなったら

より詳しい講習会へ！

---

9月3日 スパコン利用説明会

9月10日 スパコン利用入門\*

9月16日 SX-ACE 高速化技法の基礎\*

9月17日 並列コンピュータ 高速化技法の基礎\*

9月24日 並列プログラミング入門(MPI)\*

---

\* 一週間使える無料お試しアカウント付き！

[http://www.hpc.cmc.osaka-u.ac.jp/lecture\\_event/lecture/](http://www.hpc.cmc.osaka-u.ac.jp/lecture_event/lecture/)

# 今日お話しすること

## ◎ CUI (と Unix)

- いわゆる「黒い画面+白い文字」
- CUIの最低限の操作に慣れる必要があります

## ◎ 並列計算

- スパコンの概略
- 並列計算とは？どんな技術なのか？
- 普通のPCで試してみると…？

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# 目次

## 1. CUI入門

### 1.1 コマンドで操作するCUIとは

### 1.2 Unixコマンド入門

## 2. 並列計算入門

### 2.1 スーパーコンピュータ

### 2.2 計算の高速化手法, 概要とハードウェア

### 2.3 並列計算の効率限界

### 2.4 どのようなソフトウェアを使うべきか?

### 2.5 まとめ

# GUIとCUI, OSの関係



GUI (Graphical User Interface)

```
hoge -- bash -- 80x24
Last login: Wed May 27 17:48:07 on ttys001
The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT200050.
- $ cd Desktop/
-/Desktop $ mkdir hoge
-/Desktop $ cd hoge/
-/Desktop/hoge $ touch hoge1.txt
-/Desktop/hoge $ touch hoge2.txt
-/Desktop/hoge $ ls
hoge1.txt hoge2.txt
-/Desktop/hoge $
```

CUI (Character User Interface)

- OS (Windows, Mac, Unix) の GUI の見かけはほぼ同じ
- CUI / CLI (Command Line Interface) の充実度は OS により異なる
- Unix (系のOS) は CUI / CLI が非常に充実 (Unixの強み)
- Unix コマンド : CUI で使うコマンド

# GUIとCUI

## GUI

- 俯瞰的. 同時並行表示. 多くの情報量.
- 直感的.
- 環境や状況への高い依存性.
- 自動化しづらい. ソフトウェア間の連携がしづらい.

## CUI

- 局所的. 表示情報は原則一度に一つ. 情報はミニマム.
- 理論的.
- 環境や状況への依存性は低い.
- 自動化しやすい. ソフトウェア間の連携がしやすい.

# 遠隔操作で役に立つ！



手元の端末



ネットワーク



Unix サーバー

- ◎ 遠隔操作はネットワークに負荷 → 通常はCUIで.
- ◎ 通常は ssh というプロトコルが使われる.

# CUI ≈ Google View



GUI ≈ Google Map



CUI ≈ Google View

CUI では

- 常に「今どこにいるか」を認識する必要がある
- 他の場所の情報は  
「自分が移動するか取り寄せるか」して取得する必要がある

# CUI の操作 = shell への命令

CUI の操作 = shell (とよばれるソフトウェア) への命令



# 使えるエディタは主に二種類 (のみ)

CUI で使われるエディタは事実上

**Emacs or vi**

- ◎ 普通は (まだ) とっつきやすい **Emacs** がお勧め (らしい)
- ◎ サーバーの管理人は **vi** が使えないと困る (ことも)
- ◎ **vim** は vi の派生版
- ◎ vscodeのようなエディタでも拡張機能を導入すれば  
利用・練習できる

# 目次

## 1. CUI入門

1.1 コマンドで操作するCUIとは

### 1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法, 概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか?

2.5 まとめ

# これだけ知っていれば戦える!

- ◎ ディレクトリ操作
- ◎ ファイル操作
- ◎ ファイルの中身の操作
- ◎ ファイル編集（エディタ）

# Unix コマンド：ディレクトリ操作

---

pwd	今何処のディレクトリに居るかを表示
cd ..	上階層ディレクトリへ移動
cd hoge	hoge ディレクトリへ移動
mkdir hoge	ここに hoge ディレクトリを作成
rmdir hoge	この hoge ディレクトリを削除
mv hoge pokο	この hoge ディレクトリを pokο ディレクトリに移動 or 名前変更*

---

\* 既に pokο ディレクトリがあるなら移動。なければ名前変更

# Unix コマンド：ファイル操作

---

ls 今のディレクトリの中のファイルリストを表示

touch hoge hoge というファイルを作成

rm hoge hoge というファイルを削除

mv hoge pokobogo というファイルを  
poko ディレクトリに移動 or 名前変更\*

---

\* 既に pokobogo ディレクトリがあるなら移動. なければ名前変更

# Unix コマンド：ファイルの中身の操作

---

less hoge      hoge というファイルの中身を表示  
                  more, cat もほぼ同様の動作

grep kore \*    ディレクトリ内で  
                  kore という文字列を含むファイルを表示

---

# ファイル編集：Emacs

`emacs hoge` `hoge` というファイルを読み込んで起動

以下、C- は Ctrl キー同時押し、M- は Esc キーを押してから

---

C-x C-f	ファイル読み込み
C-x C-s	保存
C-x C-c	終了
C-g	emacs がしていることを止める (困ったらこれ)
C-s hoge	<code>hoge</code> という文字列を探す
C-スペースキー	選択開始
M-w	コピー
C-w	カット
C-y	ペースト

---

# ファイル編集：vi

`vi hoge` `hoge` というファイルを読み込んで起動

以下、**文字挿入モード** と **コマンドモード** を切り替えて使用

`i` 文字挿入モードへ切り替え

`Esc` コマンドモードへ切り替え

以下、コマンドモードで

---

`h, j, k, l` 左, 下, 上, 右へ移動

`:wq` 保存して終了

`:q!` 保存せず終了

`x, dd` 1 文字, 1 行カット

`yy` 1 行コピー

`p` ペースト

---

# CUI 入門のまとめ

- ◎ CUI を理解する幾つかのコツ
  - 遠隔操作でよく使われる
  - CUI の性質は “Google View” に近い
  - CUI の操作 = shell への命令
  - エディタは事実上 Emacs or vi
- ◎ 今回紹介したコマンドである程度戦える  
(必要に応じて 書籍 / web で検索)
- ◎ 薄くてよいので本があると便利

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# これからの中身

## 話すこと

- ◎ スーパーコンピュータの概要（歴史）
- ◎ 高速化の仕組み、手法
- ◎ 原理的にどれくらい速くなるのか？
- ◎ どんなハードウェアとソフトウェアの組み合わせが？

## 話さないこと

- ◎ 各種ソフトウェアのプログラミングの詳細  
→それぞれの講習会へ！
- ◎ 並列計算のチューニングなど

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# スーパーコンピュータとは

## 大規模、並列計算処理

100万単位のプロセッサコア

Top 500 (2020.06) <https://www.top500.org/>

rank	system	コア数	Rmax (PFlops / s)
1	<b>Fugaku</b> (理研)	730万	415
2	Summit (米)	240万	149
4	神威・太湖之光 (中)	1.06千万	93
10	Piz Daint (スイス)	39万	21
12	ABCi (産総研)	39万	20

(Rmax: 実効性能値)

# スーパーコンピュータとは

## 巨大メモリ

Fugaku: **4.8P** バイト,

Summit: **2.8P** バイト, 神威: **1.3P** バイト,

Piz Daint: **365T** バイト, ABCI: **418T** バイト

## 大電力 (困るけど...)

Fugaku: **28.3 MW**,

Summit: **10.1MW**, 神威: **15.4 MW**,

Piz Daint: **2.4MW**, ABCI: **1.6MW**

\* 淀川水系水力発電所 29 中,

発電力が 10MW を越えるものはたった 3 つ...

# ちなみに：阪大のSX-ACEは…

## 計算性能

3 クラスタ合計 423 TFlops = 0.423 PFlops, コア数 6144

\* 合計だと 2020 年 6 月の TOP500 の 500位(1.23 PFlops) の 約35%  
(一つのプログラムで動かせるのは 1 クラスタまで)

## メモリ

96 TB  $\cong$  0.1 PB

## 電力

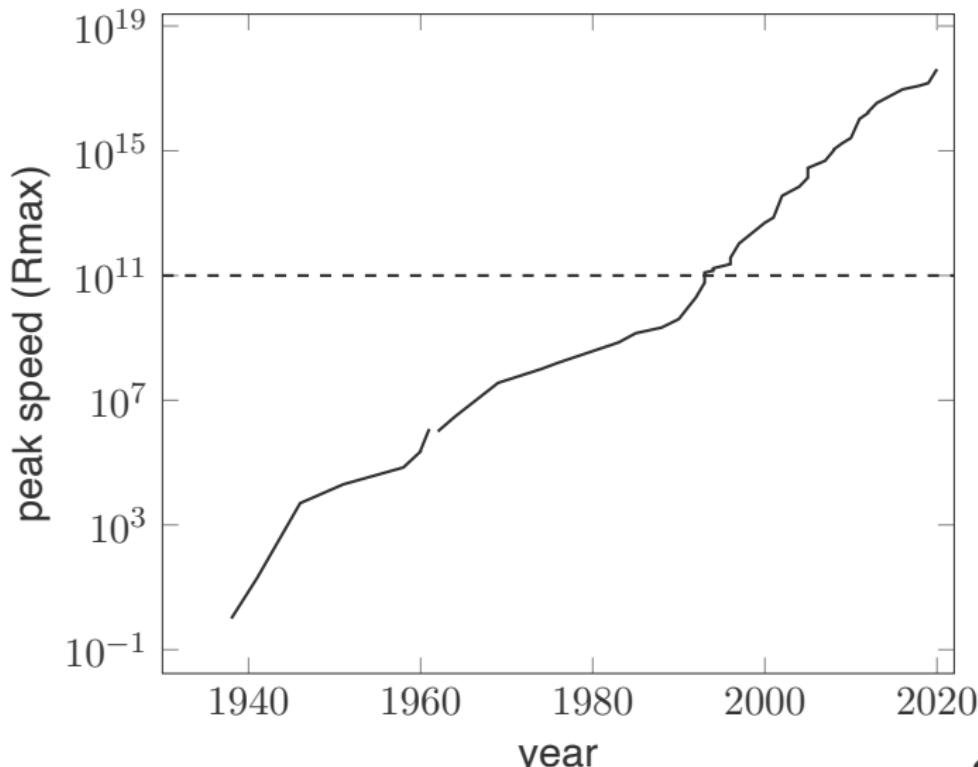
(わずか？) 700 KW \* だいたい、コタツ 1000 個分

Fugaku と比べて、性能 0.10%, メモリ 2%, 電力消費量 2.5%, くらい

Macbook Air と比べて、性能 5000倍, メモリ 6000倍, くらい

# スーパーコンピュータの歴史 I

スーパーコンピュータの性能の変遷



# スーパーコンピュータの歴史 II

- ◎ **ILLIAC I, II, III** (真空管 1952, トランジスタ 1962, SIMD 1966)  
スパコン黎明期. 並列計算の概念. 技術の基礎が登場.
- ◎ **Cray-1** (商用スパコン, 1976, 80-160MFLOPS)  
計 80 台以上が飛ぶように売れた.
- ◎ **地球シミュレータ**  
(SX-5, 41 TFLOPS, 2002: SX-9, 131 TFLOPS, 2009)  
TOP500 1 位 2002.06-2004.06.  
当時としては「化け物」(登場時, 2 位の 5 倍の性能)
- ◎ **Blue Gene** (2004) シンプルにコア数を増やす戦略.  
登場時で既に 32,768 コア. 2007 年には 212,992 コア.
- ◎ **京** (2011) 10.62 PFLOPS の超並列機.
- ◎ **天河 2 号** (2013) CPU を数多く繋ぐ. 33.8 PFLOPS.
- ◎ **神威・太湖之光** (2016) CPU をとにかく数多く. 93 PFLOPS.
- ◎ **Summit** (2018) 122 PFLOPS.
- ◎ **Fugaku** (2020) 415PFLOPS.

# スーパーコンピュータの歴史 III

トップ性能のスパコンの所持・開発に絡んでいる国

① アメリカ (113 / top 500)

CPU 数を増やして能力を稼ぐ。Cray 社と IBM 社が強い。  
CPU を自国で開発できる強みあり。

② 日本 (29)

ベクトル型スパコンを作る唯一の国か。  
NEC と富士通、日立が主なメーカー。  
CPU は自国開発ものと輸入物の混在。

③ 中国 (226)

ほぼアメリカ製だったが、近年は CPU の自力開発も。  
この数年急激に数を増やしていたが、減速気味。

④ フランス (19), ドイツ (16), オランダ (15), アイルランド (14),  
カナダ (13), イギリス (10)

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# そもそも計算とは？



高速化するためには

- ◎ 計算のアルゴリズムを工夫する。  
演算回数が  $1/2$  になれば、計算時間もおよそ  $1/2$
- ◎ 高性能な計算機を使う
- ◎ **並列計算**：分担して処理

# アルゴリズムの工夫

## アルゴリズムそのものの改善

「より速い」 or 「並列化により適している」アルゴリズムへ

例： $1+2+\cdots+100$  ?

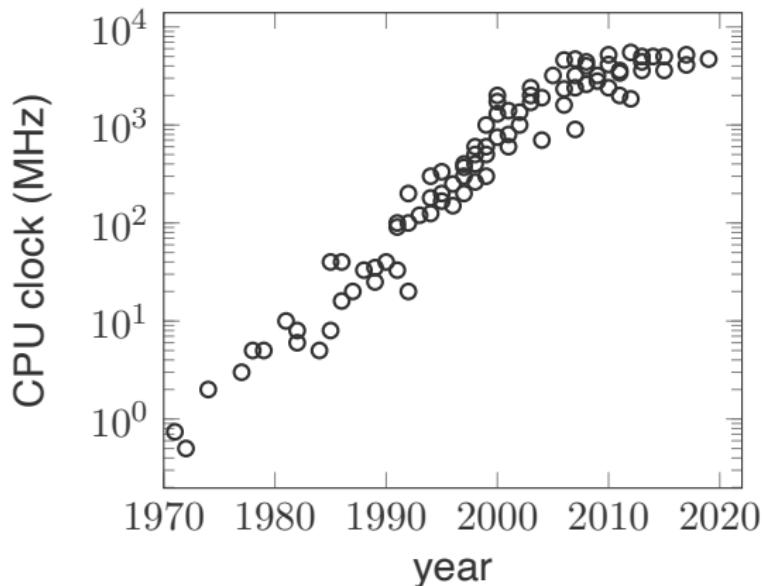
- ◎ `for i = 1:100 do result += i;`
- ◎ `result = (100 + 1) * 100 / 2`

四則演算の回数を大幅に削減！

# 高性能な計算機を使う

速く回す！

CPU, メモリ, HDD などのスピードを単純に上げる。



単純なスピードアップはそろそろ限界？

# 並列計算 I-1

## 一度に沢山！

似たような処理を一度に沢山行う（ハードウェアで）

- ◎ ベクトル計算
- ◎ SIMD (Single Instruction Multiple Data)  
单一命令・複数データ流

(注)

- ◎ ベクトル計算機：  
パイプラインによるベクトル演算が可能な計算機
- ◎ SIMDはPCのCPU等にも搭載

## 並列計算 I-2

スカラー計算：1つずつ順番に計算していく



$a[1] = b[1] + c[1]$	$a[2] = b[2] + c[2]$	$\dots$	$a[100] = b[100] + c[100]$
----------------------	----------------------	---------	----------------------------

ベクトル演算：一度に全部計算できる

$$\begin{array}{c|c|c} a[1] & b[1] & c[1] \\ \hline a[2] & b[2] & c[2] \\ \hline & & \\ \hline a[100] & b[100] & c[100] \end{array} = \begin{array}{c|c} \hline & + \\ \hline \end{array}$$

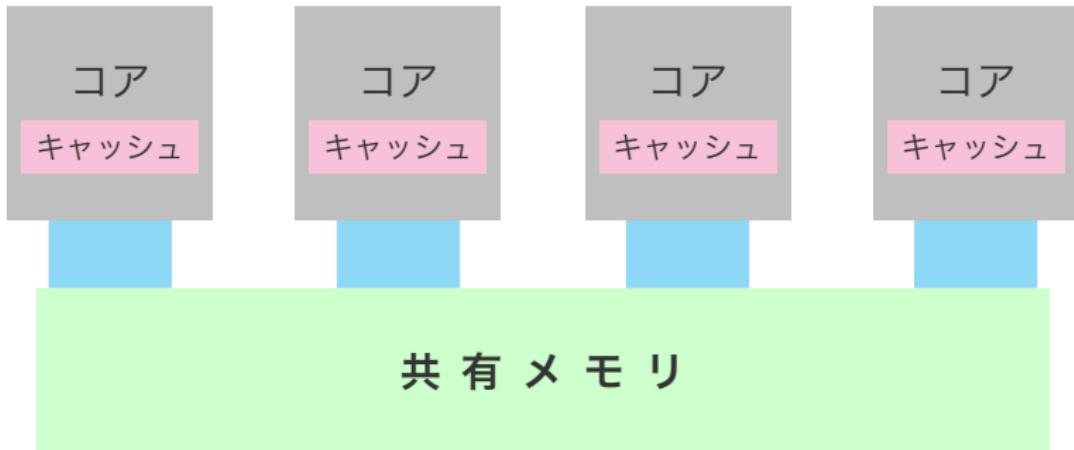
# 並列計算 II-1

## 皆でやる！メモリ共有型

メモリ（データ）を共有した状態で、処理を並列化。

実装しやすいが、メモリアクセス競合が起きやすく、  
高速化しにくい。

## 並列計算 II-2



メモリ共有型計算機の概念図

- ◎ 一つの CPU に複数のコア
- ◎ ノード：共有メモリを持つハードウェアの構成単位

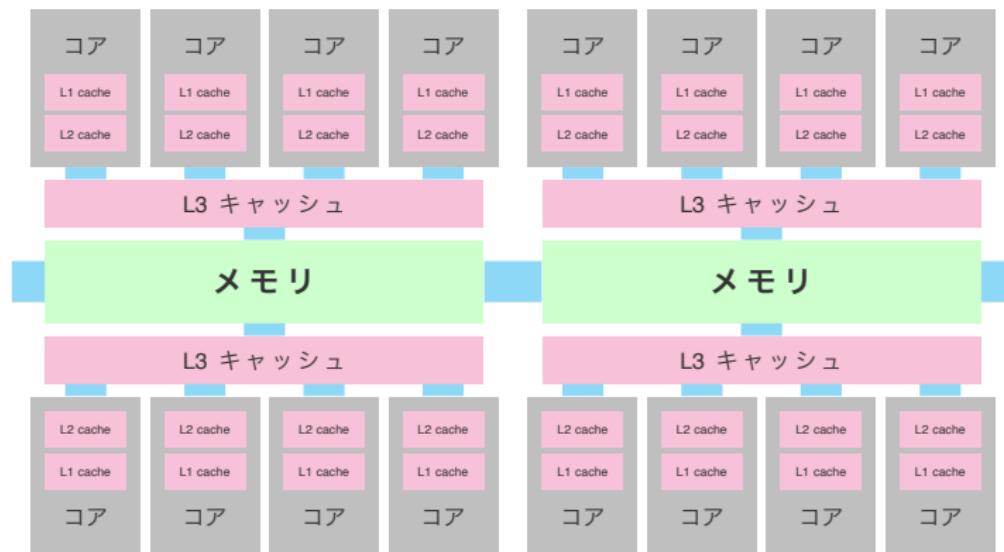
## 皆でやる！メモリ分散型

メモリ（データ）を分散して、処理を並列化。

メモリアクセス競合が起きにくく、高速化しやすい。

- ◎ 「メモリ+CPU」がたくさん繋がっている場合
- ◎ ノードがたくさん繋がっている場合

並列計算 III-2



## メモリ分散型計算機の概念図

# 計算の高速化手法の現状

- ◎ 「単純に速く」というのは困難になりつつある.
- ◎ ベクトル型スパコンは今や NEC の SX シリーズぐらいか.  
PC のチップや GPU 計算は SIMD をサポート.
- ◎ 多くのスパコンは超並列 (つなぎ方がすごい！)  
メモリは、多段分散型.  
つまり、分散しているが近くとは共有のキャッシュがあるような多段構造が多い.

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

### 2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# 並列計算でどれくらい速くなる？

## アムダール則

プログラム中、割合  $a$  の部分を  $n$  個のプロセッサで並列化し、  
残りの  $1 - a$  の部分を並列化しない場合、全体は

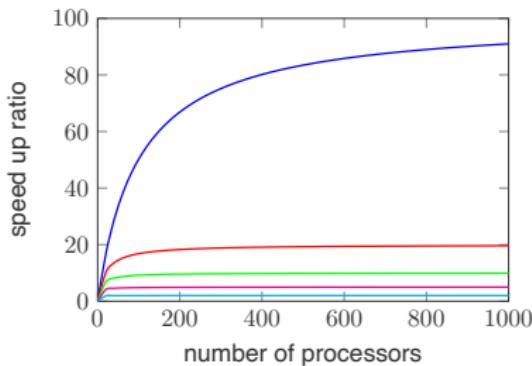
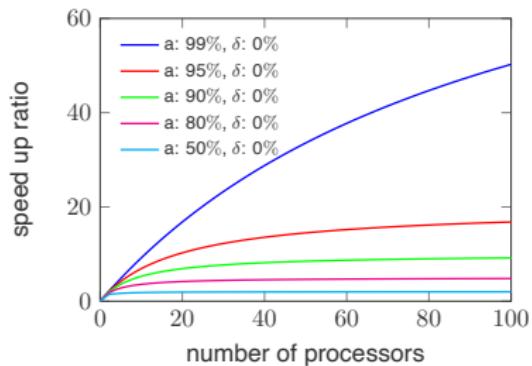
$$\frac{1}{(1 + \delta)\frac{a}{n} + (1 - a)} \text{ 倍}$$

に高速化される

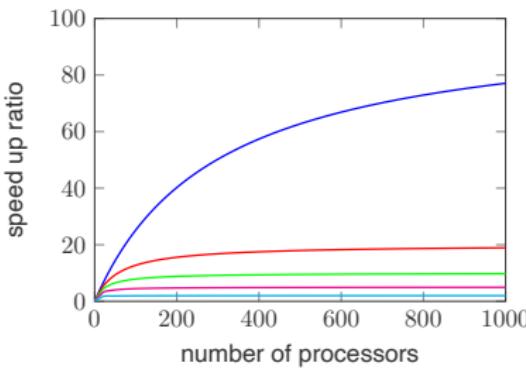
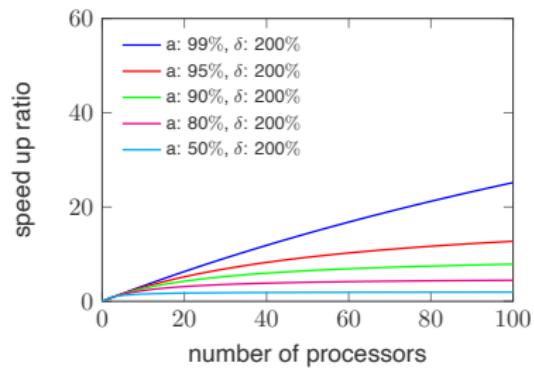
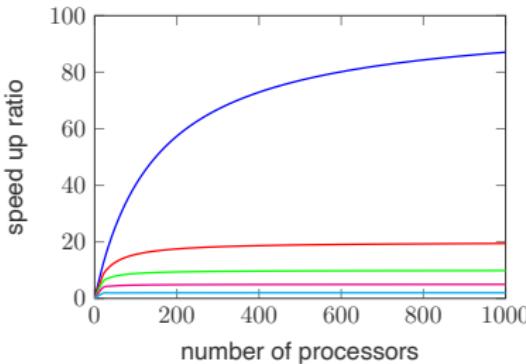
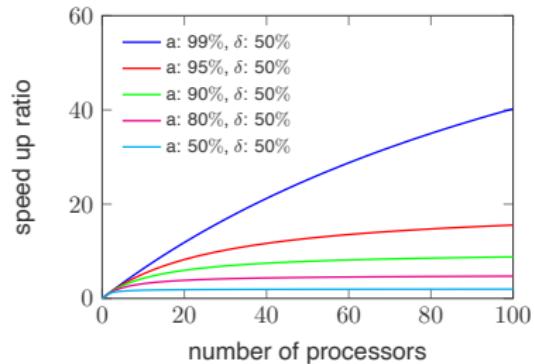
\* $\delta$ : 並列化に伴って発生する通信などによる遅延率

(例)  $a = 0.8, n = 8, \delta = 0.2$  だと、3.125 倍

# 遅延がない場合



# 遅延がある場合



# 結局のところ

- ◎ 並列化できない箇所が信じられないくらい足を引っ張る
- ◎ 並列化に伴う通信等で遅延があると、  
全体をじわじわと遅くする
- ◎ ただ並列化するだけでは効率は悪いかもしれない

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# ハードウェアとソフトウェアの組み合わせ

ハードウェア的な結合度の強い方から並べると…

結合方法	メモリ	ソフトウェア等
SIMD / ベクトル化	共有	各種ライブラリ, CUDA 等
CPU内並列	共有	OpenMP (Multi Processing), 他
CPU間並列	共有	OpenMP (Multi Processing), 他
マシン間並列	分散	MPI (Message Passing Interface), 他

- ◎ 一般に、上の方がプログラミングは容易
- ◎ 上2つはラップトップなどでも試せる
- ◎ ハードウェアは下の方がスケールしやすい

# SIMD / ベクトル化

- ◎ ハードウェア, ソフトウェア, ライブラリの「準備」をしさえすれば容易に利用できる.
- ◎ プログラミング的な意味での特殊なテクニックはほぼ不要.  
「ここはベクトル化する / しない」という強制的な  
ディレクティブ（必須ではない）を入れたりするぐらい.
- ◎ アムダールの法則があるので,  
ベクトル化率を高めるための変更は必要かつ重要.

# SIMD / ベクトル化：内積計算の例

- ◎ C / C++ on SX-ACE:

```
#pragma vdir nodep ← #pragma vdir ... をつけるだけ  
for(i=1;i<length(x);i++){ s += x[i] * y[i]; }
```

- ◎ Fortran90/SX on SX-ACE: 自動

- ◎ Julia lang:

```
@simd for i=1:length(x) ← @simd をつけるだけ  
    @inbounds s += x[i] * y[i]  
end
```

実際に、6~7倍高速 (1万次元ベクトル, 1万回平均)

---

Macbook Air	12.689 μs → 1.967 μs
-------------	----------------------

---

Mac mini	9.175 μs → 1.398 μs
----------	---------------------

# CPU 内 / 間 並列化：メモリ共有

- ① プログラミングは比較的容易。  
「ここからここまで並列化」というディレクティブを  
書き入れるくらい。(thread を用いると少し難しい)
- ② 移植性は高い。
- ③ 解説書は多い。
- ④ やや高速化しにくい。
- ⑤ ソフトウェアとしては
  - Open MP
  - OS や言語の用意するライブラリ
    - Grand Central Dispatch (MacOS X 10.6, FreeBSD),
    - intel TBB, Google Go, Rust など

# サンプル：OpenMP 的な並列化 (1)

## Fortran の例

```
program hello  
:  
!$omp parallel  
並列化したい計算部分  
!$omp end parallel  
:  
end
```

- ◎ 並列化したいところをディレクティブで挟むのが基本.
- ◎ C/C++ の場合は `#pragma omp parallel` など.
- ◎ 並列度等はコンパイラや環境変数で指定.

# サンプル：OpenMP 的な並列化 (2)

## Julia lang の例（コイントス）

```
nheads = @distributed (+) for i=1:200000000  
    Int(rand(Bool))  
end
```

Macbook Air	デュアルコア	1.654 s → 1.010 s	約1.6倍高速
Mac mini	6 コア	1.271 s → 250 ms	約5倍高速

SIMD, OpenMP の並列化手法ならば、通常のプログラミング能力で十分。

# マシン間並列化：分散メモリ

- ◎ 通信 (Message) :

並列して動く各部分同士の情報のやりとり

- ◎ プログラムの動作を並列化用に設計する必要あり

- ◎ プログラムは面倒

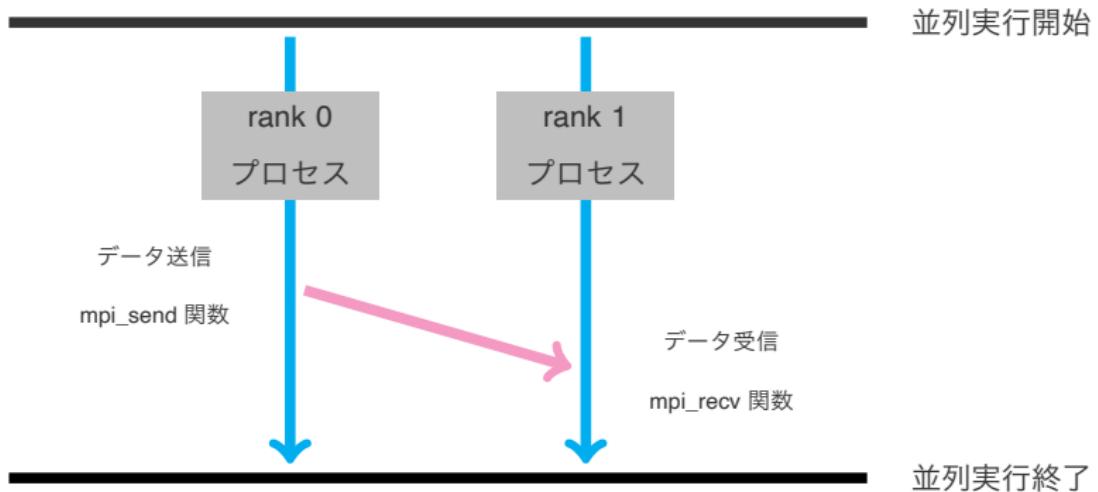
- ◎ ハードウェアへの依存性はやや高い (移植性はやや低い)

- ◎ 解説書は多い

- ◎ スーパーコンピュータはほぼこれ

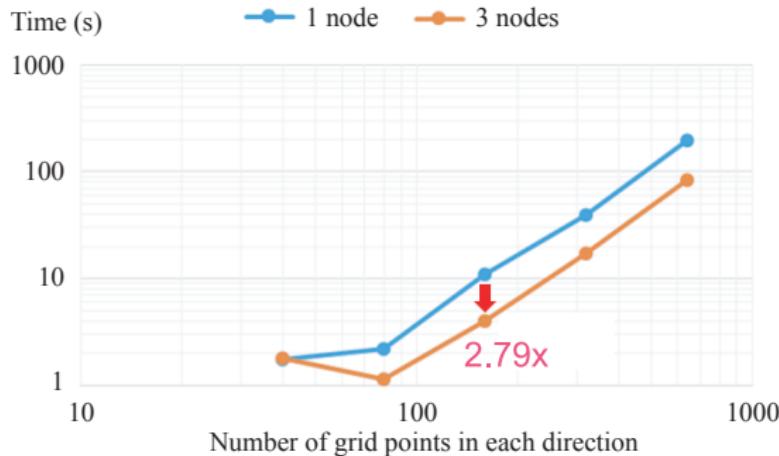
- ◎ ソフトウェアは **MPI** (Message Passing Interface) という  
共通規格に沿った言語、ライブラリを利用

- データは分散。他プロセスのデータには触れない。
- データはお互いに「明示的に」通信する必要がある。



# MPI : 例

非線形Schrödinger方程式の数値計算 (Reedbush-U @ 東大)



T. Sakai, S. Kudo, H. Imachi, Y. Miyatake, T. Hoshi, Y. Yamamoto  
A parallelizable energy-preserving integrator MB4 and its application to quantum-mechanical wavepacket dynamics  
to appear in Japan J. Indust. Appl. Math. (arXiv:2003.04345)

# 目次（この講習会の内容）

## 1. CUI入門

1.1 コマンドで操作するCUIとは

1.2 Unixコマンド入門

## 2. 並列計算入門

2.1 スーパーコンピュータ

2.2 計算の高速化手法、概要とハードウェア

2.3 並列計算の効率限界

2.4 どのようなソフトウェアを使うべきか？

2.5 まとめ

# まとめ

- ハードウェアによって並列化の方法は異なる。  
→ ソフトウェアはそれに合わせて選択。
- (他に比べ) MPI はやや敷居が高い。
- SX-ACE (阪大スパコン) はベクトル型計算機を束ねたもの。  
1 ノード (1 cpu, 4 core) でおさまる計算ならば  
テクニック的には難しいことはない。  
(ベクトル化率を高める為の工夫はまた別に必要)
- 普通のPCで 4 コア持っていたりする。  
→ 4 倍 (弱) くらいまでの並列化は容易にできたりする。
- ベクトル化, MPI についてはサイバーで講習会あり。