

スパコンに通じる並列プログラミングの基礎

降旗 大介

大阪大学サイバーメディアセンター

2014.06.03

はじめに

この講習の対象者

- 普段は Windows, Mac を使っていて, Unix についてはあまり…
- 研究対象についてはよく知っている.
- 普通にプログラミングは出来る.
- 計算対象がやや大規模になりそうだ.
- 少しでも計算が速いと有難い.
- 並列計算に興味はあるが, まず全体像がよくわからない. どのような技術があるのか? 難しいのか簡単なのか? 高価につくのか?

0432-J

Part I

Unix 入門

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ
- ごく簡単な入門

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ
- ごく簡単な入門

Part II

並列計算入門

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ
- ごく簡単な入門

Part II

並列計算入門

- 原理的に並列計算でどれくらい高速化が可能か

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ
- ごく簡単な入門

Part II

並列計算入門

- 原理的に並列計算でどれくらい高速化が可能か
- そもそも高速化の手法には何があるのか

Part I

Unix 入門

- GUI と CUI: Unix, Windows, Mac OS の同じところと違うところ
- ごく簡単な入門

Part II

並列計算入門

- 原理的に並列計算でどれくらい高速化が可能か
- そもそも高速化の手法には何があるのか
- 各種並列計算の紹介

詳しく知りたくなったら？

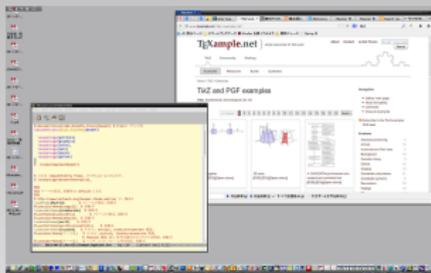
ぜひ、それぞれのソフトウェアの講習会 (下記: 2014 年度) へ！

- 6月3日 スパコンに通じる並列プログラミングの基礎 (本日)
- 6月9日 スーパーコンピュータ概要とスーパーコンピュータ利用入門
- 6月10日 IDL 利用入門
- 6月17日 スーパーコンピュータと並列コンピュータの高速化技法の基礎
(by NEC)
- 6月24日 **MPI** プログラミング入門
- 6月26日 **HPF** プログラミング入門
- 8月8日 Gaussian 講習会
- 9月16日 AVS 可視化処理入門
- 9月17日 AVS 可視化処理応用
- 9月頃 並列計算入門
- 12月～2月 スーパーコンピュータ概要とスーパーコンピュータ利用入門
- 12月～2月 スーパーコンピュータと並列コンピュータの高速化技法の基礎
- 12月～2月 **MPI** プログラミング入門
- 12月～2月 **HPF** プログラミング入門

Part I: UNIX 入門

コマンドで操作する CUI とは

GUI と CUI, OS の関係



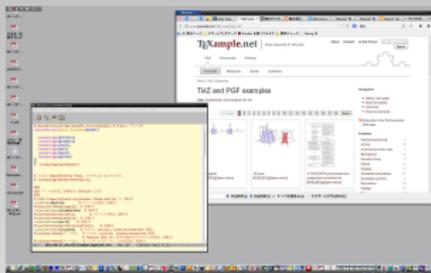
GUI

- b Forces a "break" from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.
- c Commands are read from the following argument (which must be present, and must be a single argument), stored in the `command` shell variable for reference, and executed. Any remaining arguments are placed in the `argv` shell variable.
- d The shell loads the directory stack from `~/cshdirs` as described under `Startup and shutdown`, whether or not it is a login shell. (i)
- Dname[=value]
Sets the environment variable `name` to `value`. (Domain/OS only) (+)
- e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

```
omodaru@paon> <102>
```

CUI

GUI と CUI, OS の関係



GUI

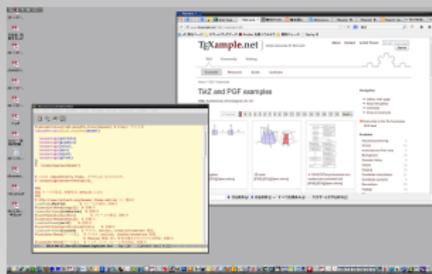
- b Forces a "break" from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.
- c Commands are read from the following argument (which must be present, and must be a single argument), stored in the `command` shell variable for reference, and executed. Any remaining arguments are placed in the `argv` shell variable.
- d The shell loads the directory stack from `~/cshdirs` as described under `Startup and shutdown`, whether or not it is a login shell. (+)
- Dname[=value] Sets the environment variable `name` to `value`. (Domain/OS only) (+)
- e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

omodaru@paon> <102>

CUI

- 今の OS (Windows, MacOS X, Unix) の GUI (Graphical User Interface) の見かけはあまり変わらない。

GUI と CUI, OS の関係



GUI

```
-b Forces a "break" from option processing, causing any further
shell arguments to be treated as non-option arguments. The remain-
ing arguments will not be interpreted as shell options. This may
be used to pass options to a shell script without confusion or pos-
sible subterfuge. The shell will not run a set-user ID script
without this option.

-c Commands are read from the following argument (which must be
present, and must be a single argument), stored in the command
shell variable for reference, and executed. Any remaining argu-
ments are placed in the argv shell variable.

-d The shell loads the directory stack from /.cshdirs as described
under Startup and shutdown, whether or not it is a login shell. (+)

-Dname[=value]
Sets the environment variable name to value. (Domain/OS only) (+)

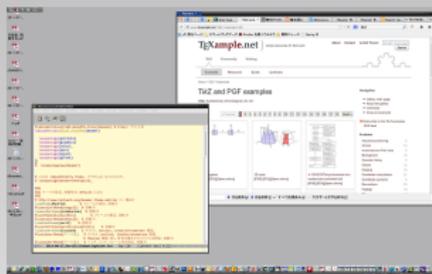
-e The shell exits if any invoked command terminates abnormally or
yields a non-zero exit status.

omodaru@paon> <102>
```

CUI

- 今の OS (Windows, MacOS X, Unix) の GUI (Graphical User Interface) の見かけはあまり変わらない。
- CUI (Character User Interface)/ CLI (Command Line Interface) の充実度は OS によって異なる。

GUI と CUI, OS の関係



GUI

```
-b Forces a "break" from option processing, causing any further
shell arguments to be treated as non-option arguments. The remain-
ing arguments will not be interpreted as shell options. This may
be used to pass options to a shell script without confusion or pos-
sible subterfuge. The shell will not run a set-user ID script
without this option.

-c Commands are read from the following argument (which must be
present, and must be a single argument), stored in the command
shell variable for reference, and executed. Any remaining argu-
ments are placed in the argv shell variable.

-d The shell loads the directory stack from ~/cshdirs as described
under Startup and shutdown, whether or not it is a login shell. (+)

-Dname[=value]
Sets the environment variable name to value. (Domain/OS only) (+)

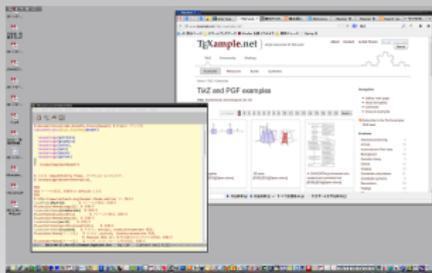
-e The shell exits if any invoked command terminates abnormally or
yields a non-zero exit status.

omodaru@paon> <102>
```

CUI

- 今の OS (Windows, MacOS X, Unix) の GUI (Graphical User Interface) の見かけはあまり変わらない。
- CUI (Character User Interface)/ CLI (Command Line Interface) の充実度は OS によって異なる。
- Unix は CUI/CLI が非常に充実、かつ、Unix の強みはそこに。

GUI と CUI, OS の関係



GUI

```
-b Forces a "break" from option processing, causing any further
shell arguments to be treated as non-option arguments. The remain-
ing arguments will not be interpreted as shell options. This may
be used to pass options to a shell script without confusion or pos-
sible subterfuge. The shell will not run a set-user ID script
without this option.

-c Commands are read from the following argument (which must be
present, and must be a single argument), stored in the command
shell variable for reference, and executed. Any remaining argu-
ments are placed in the argv shell variable.

-d The shell loads the directory stack from /.cshdirs as described
under Startup and shutdown, whether or not it is a login shell. (+)

-Dname[=value]
Sets the environment variable name to value. (Domain/OS only) (+)

-e The shell exits if any invoked command terminates abnormally or
yields a non-zero exit status.
```

omdsrv@paon> <102>

CUI

- 今の OS (Windows, MacOS X, Unix) の GUI (Graphical User Interface) の見かけはあまり変わらない。
- CUI (Character User Interface)/ CLI (Command Line Interface) の充実度は OS によって異なる。
- Unix は CUI/CLI が非常に充実、かつ、Unix の強みはそこに。
- Unix がよくわからない = CUI/CLI がよくわからない。

GUI

- 俯瞰的. 同時並行表示. 情報量多し.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

CUI

- 局所的. 表示情報は原則 1 度に 1 つ. 情報はミニマム.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

CUI

- 局所的. 表示情報は原則 1 度に 1 つ. 情報はミニマム.
- 理論的な操作.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

CUI

- 局所的. 表示情報は原則 1 度に 1 つ. 情報はミニマム.
- 理論的な操作.
- 環境・状況依存性低し.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

CUI

- 局所的. 表示情報は原則 1 度に 1 つ. 情報はミニマム.
- 理論的な操作.
- 環境・状況依存性低し.
- 自動化しやすい.

GUI

- 俯瞰的. 同時並行表示. 情報量多し.
- 直感的に使える.
- 環境・状況依存性高し.
- 自動化しにくい.
- ソフトウェア間の連携しにくい.

CUI

- 局所的. 表示情報は原則 1 度に 1 つ. 情報はミニマム.
- 理論的な操作.
- 環境・状況依存性低し.
- 自動化しやすい.
- ソフトウェア間の連携しやすい.

CUI を理解するコツ I

GUI は 地図を見ながらの操作で, CUI は 現場での操作 である!



GUI = 地図



CUI = 現場

CUI では「今何処にいるか」が重要。
見たいもの(ファイル等)があったらそこまで移動しないとイケない。

CUI を理解するコツ II

CUI の操作 = 執事 (shell) への命令 である。



CUI 操作 = シェルとよばれるソフトウェアへの命令.
付き合いにくいシェルもあるので、好みで変えよう!

CUI は遠隔操作でよく使われる



手元の端末



ネットワーク



Unix サーバ

遠隔操作はネットワークに負荷がかかるので、普通は CUI で。
通常は ssh というプロトコルが使われる。

0435-J

Unix コマンド入門
これだけ知っていれば戦える

- `pwd` 今何処のディレクトリに居るかを表示.

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.

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.

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 ディレクトリを作る.

- `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 ディレクトリを作る.
- `rmdir hoge` この hoge ディレクトリを削除.

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 ディレクトリを作る.
- `rmdir hoge` ここの hoge ディレクトリを削除.
- `mv hoge poko` ここの hoge ディレクトリを poko ディレクトリに移動 or 名前変更.
 - poko ディレクトリが既に有るならその中へ移動,
 - 無いならその名前に変更.

0435-J

- `ls` 今のディレクトリに有るファイルのリスト表示.

- `ls` 今のディレクトリに有るファイルのリスト表示.
- `touch hoge` hoge というファイルを作る.

- `ls` 今のディレクトリに有るファイルのリスト表示.
- `touch hoge` hoge というファイルを作る.
- `rm hoge` hoge というファイルを削除.

- `ls` 今のディレクトリに有るファイルのリスト表示.
- `touch hoge` hoge というファイルを作る.
- `rm hoge` hoge というファイルを削除.
- `mv hoge poku` hoge というファイルを poku ディレクトリに移動
or 名前変更.
 - poku ディレクトリが既に有るならその中へ移動,
 - 無いならその名前に変更.

- `less hoge` `hoge` というファイルの中身を表示.
ほぼ同様の動作をするコマンド: `more`, `cat`

Unix コマンド: ファイル中身 操作

- `less hoge` `hoge` というファイルの中身を表示.
ほぼ同様の動作をするコマンド: `more`, `cat`
- `grep kore *` このディレクトリで `kore` という文字列を含むファイルを表示.

0435-J

- `emacs hoge` `hoge` というファイルを読み込んで起動.

- `emacs hoge` `hoge` というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- C-x C-s 保存.

- `emacs hoge` `hoge` というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- C-x C-s 保存.
- C-x C-c 終了.

- `emacs hoge` `hoge` というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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.

Unix エディタ (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, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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 hoge という文字列を探す.

- `emacs hoge` `hoge` というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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 hoge という文字列を探す.
- C-スペースキー 選択開始.

- `emacs hoge` `hoge` というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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 hoge 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.

- `emacs hoge` hoge というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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 hoge 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 (削除).

- `emacs hoge` hoge というファイルを読み込んで起動.

(以下, emacs 中で.

C- は Ctrl キー同時押し, M- は Esc キーを押してから.)

- 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 hoge 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 (削除).
- C-y ペースト (貼付け).

Unix エディタ (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.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(以下, コマンドモードで.)

- h, j, k, l 左, 下, 上, 右へ移動.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(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.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(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.

- vi hoge hoge というファイルを読み込んで起動.

(以下, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(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 (削除).

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(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 (削除).
- yy 1行コピー.

Unix エディタ (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, vi 中.

文字挿入モードと, コマンドモードを切り替えて使う.)

- i 文字挿入モードへ切替え.
- Esc キー コマンドモードへ切替え.

(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 (削除).
- yy 1行コピー.
- p ペースト (貼付け).

- Unix がわからない = CUI に慣れてないというだけのことが多い.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.
 - CUI はその性質が Google View によく似ている.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.
 - CUI はその性質が Google View によく似ている.
 - CUI の操作 = 執事 (shell) への命令.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.
 - CUI はその性質が Google View によく似ている.
 - CUI の操作 = 執事 (shell) への命令.
 - CUI は遠隔操作でよく使われる.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.
 - CUI はその性質が Google View によく似ている.
 - CUI の操作 = 執事 (shell) への命令.
 - CUI は遠隔操作でよく使われる.
 - CUI で使われるエディタは事実上 Emacs か vi に限られている.

- Unix がわからない = CUI に慣れてないというだけのことが多い.
- CUI を理解するにはいくつかコツが有る.
 - CUI はその性質が Google View によく似ている.
 - CUI の操作 = 執事 (shell) への命令.
 - CUI は遠隔操作でよく使われる.
 - CUI で使われるエディタは事実上 Emacs か vi に限られている.
- コマンドは沢山あるが、今回紹介したものがわかれば充分戦える.

- Unix がわからない = CUI に慣れてないというだけのことが多い。
- CUI を理解するにはいくつかコツが有る。
 - CUI はその性質が Google View によく似ている。
 - CUI の操作 = 執事 (shell) への命令。
 - CUI は遠隔操作でよく使われる。
 - CUI で使われるエディタは事実上 Emacs か vi に限られている。
- コマンドは沢山あるが、今回紹介したものがわかれば充分戦える。
- 薄いものでいいので Unix の本を買って持っておこう。

Part II: 並列計算入門

この Part の構成

話すこと

この Part の構成

話すこと

- スーパーコンピュータの歴史

この Part の構成

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法

この Part の構成

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法
- 原理的に, 並列計算でどれくらい速くなるのか?

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法
- 原理的に, 並列計算で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?

この Part の構成

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法
- 原理的に, 並列計算で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

この Part の構成

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法
- 原理的に, 並列計算で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 GO!

この Part の構成

話すこと

- スーパーコンピュータの歴史
- 高速化の仕組み, 手法
- 原理的に, 並列計算で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 GO!
- 並列計算のチューニング等

スーパーコンピュータについて

スーパーコンピュータとは

- 科学技術計算を主目的とする大規模コンピュータのこと.

スーパーコンピュータとは

- 科学技術計算を主目的とする大規模コンピュータのこと.
- **大規模, 超並列処理**

スーパーコンピュータとは

- 科学技術計算を主目的とする大規模コンピュータのこと.
- **大規模, 超並列処理**
 - ベクトル演算 (SIMD): 1 クロックで数百の演算を同時実行
 - SX-9 (NEC) は 1 チップ 1 コアプロセッサで 100GFlops 以上 (2007 年当時世界最高速).
 - 数百万規模のプロセッサコア
 - 京 (Top500 1 位: 2011.06-12): コア数 705,024.
 - 天河 2 号 (中国, Top500 1 位: 2013.06-): コア数 3,120,000

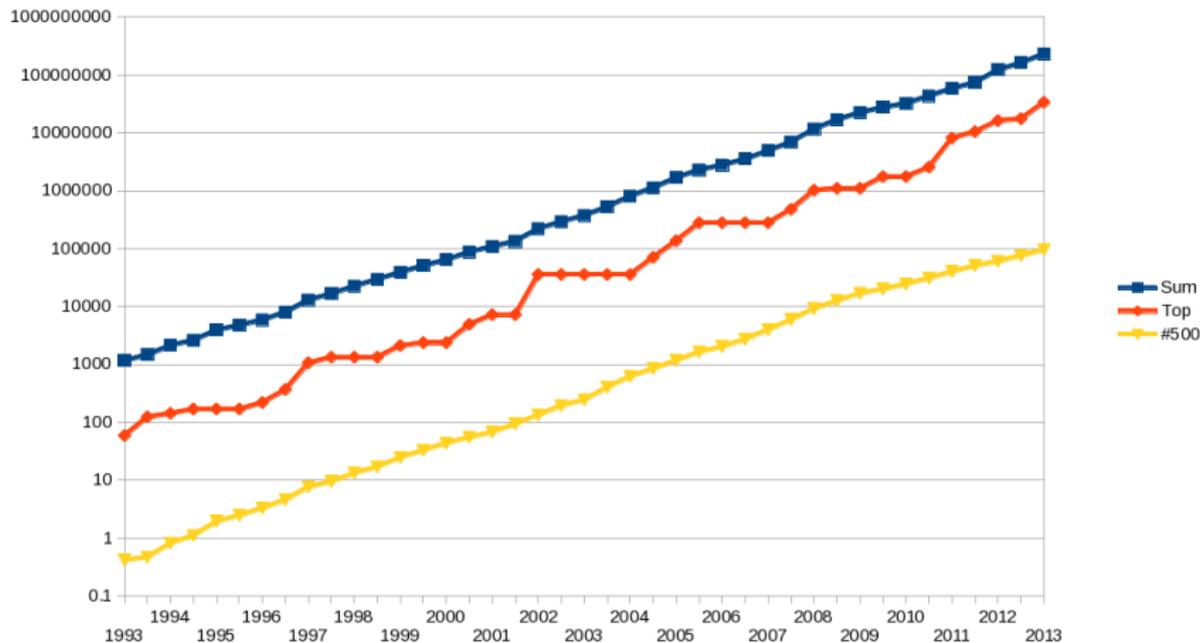
スーパーコンピュータとは

- 科学技術計算を主目的とする大規模コンピュータのこと.
- **大規模, 超並列処理**
 - ベクトル演算 (SIMD): 1 クロックで数百の演算を同時実行
 - SX-9 (NEC) は 1 チップ 1 コアプロセッサで 100GFlops 以上 (2007 年当時世界最高速).
 - 数百万規模のプロセッサコア
 - 京 (Top500 1 位: 2011.06-12): コア数 705,024.
 - 天河 2 号 (中国, Top500 1 位: 2013.06-): コア数 3,120,000
- **巨大メモリ**
 - 京: 1.26 P バイト, PrimeHPC FX10(京の商用版) は spec 上 6Pbyte 積める

スーパーコンピュータとは

- 科学技術計算を主目的とする大規模コンピュータのこと.
- **大規模, 超並列処理**
 - ベクトル演算 (SIMD): 1 クロックで数百の演算を同時実行
 - SX-9 (NEC) は 1 チップ 1 コアプロセッサで 100GFlops 以上 (2007 年当時世界最高速).
 - 数百万規模のプロセッサコア
 - 京 (Top500 1 位: 2011.06-12): コア数 705,024.
 - 天河 2 号 (中国, Top500 1 位: 2013.06-): コア数 3,120,000
- **巨大メモリ**
 - 京: 1.26 P バイト, PrimeHPC FX10(京の商用版) は spec 上 6Pbyte 積める
- **大電力** (… 困るが)
 - 京は 12,659KW, 天河 2 号は 17,808 KW. 淀川水系にある 30 の水力発電所のうちでも 10,000 KW を越えるものはたった 3 つ.

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

(creative commons -attribution, share alike 3.0 by A.I.Graphic)

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.

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.

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」. アメリカでは Japanese 'Computenik' 扱い.

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」. アメリカでは Japanese 'Computenik' 扱い.
- Blue Gene (2004)
シンプルにコア数を増やす戦略. 登場時点で既に 32,768 コア, 2007 年には 212,992 コア.

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」. アメリカでは Japanese 'Computenik' 扱い.
- Blue Gene (2004)
シンプルにコア数を増やす戦略. 登場時点で既に 32,768 コア, 2007 年には 212,992 コア.
- 京 (2011)
10.62 PFLOPS の超並列機.

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」. アメリカでは Japanese 'Computenik' 扱い.
- Blue Gene (2004)
シンプルにコア数を増やす戦略. 登場時点で既に 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)
intel CPU をとにかかく数多く繋ぐ. 33.8 PFLOPS.

トップ性能のスーパーコンピュータに絡んでいる国は次の3つ.

トップ性能のスーパーコンピュータに絡んでいる国は次の3つ.

- 中国： 速い商用 CPU を数多く使う戦略.

トップ性能のスーパーコンピュータに絡んでいる国は次の3つ.

- 中国： 速い商用 CPU を数多く使う戦略.
- アメリカ： CPU 数を多くというシンプルな戦略は同じ.
Cray 社と IBM 社が強い.
CPU を自国で開発できる強みあり.

スーパーコンピュータの歴史 III

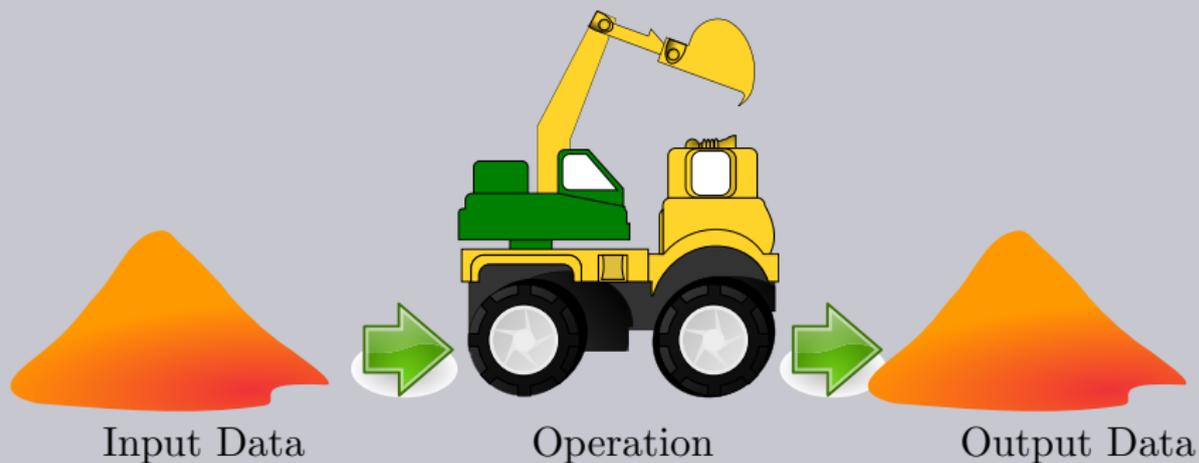
トップ性能のスーパーコンピュータに絡んでいる国は次の3つ.

- 中国： 速い商用 CPU を数多く使う戦略.
- アメリカ： CPU 数を多くというシンプルな戦略は同じ.
Cray 社と IBM 社が強い.
CPU を自国で開発できる強みあり.
- 日本： 今や世界で唯一のベクトル型スーパーコンピュータを作る国.
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.

0434-J

計算の高速化手法, 概要とハードウェア

そもそも計算とは



アルゴリズムそのものの改善

(理想的) 「より速い」 or 「並列化により適している」アルゴリズムへ



例： $1 + 2 + 3 + \dots + 100 = 5050$ に対して、

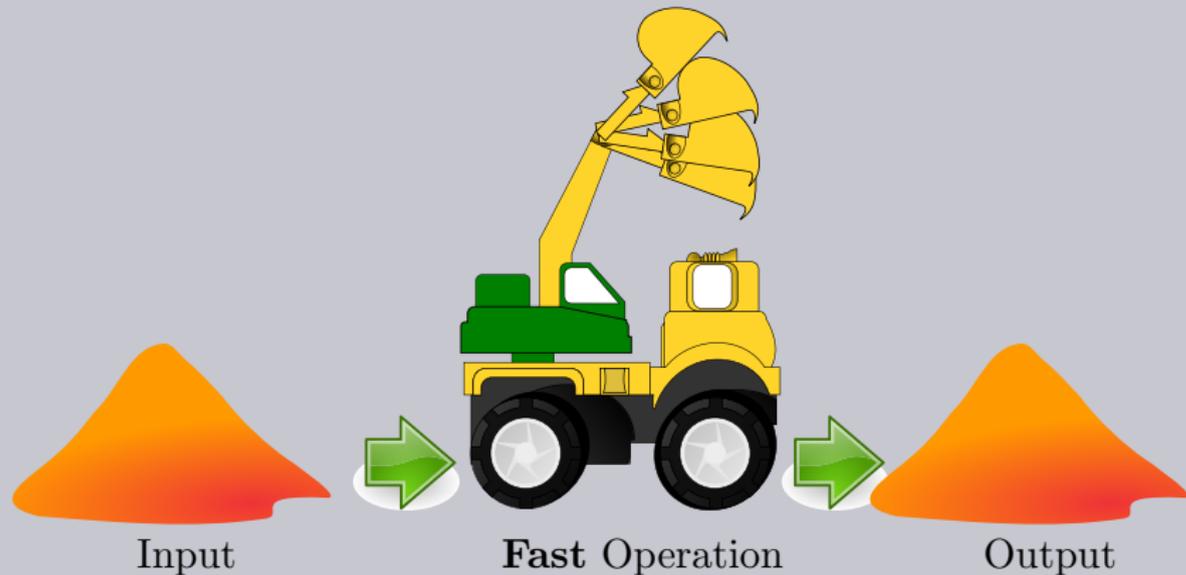
```
for i := 1 to 100 do result += i;
```

を改善して、次のように。

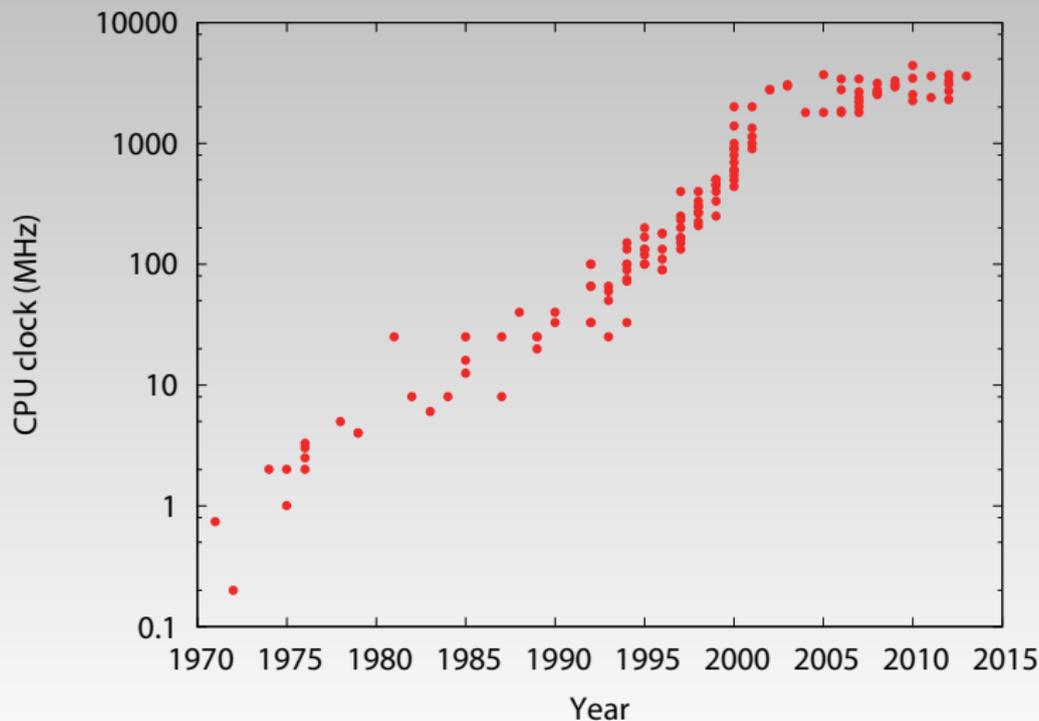
$$(100 + 1) * 100 / 2$$

速く回せ！

CPU, メモリ, HDD などのスピードを単純に上げる.



計算の高速化手法 III-2



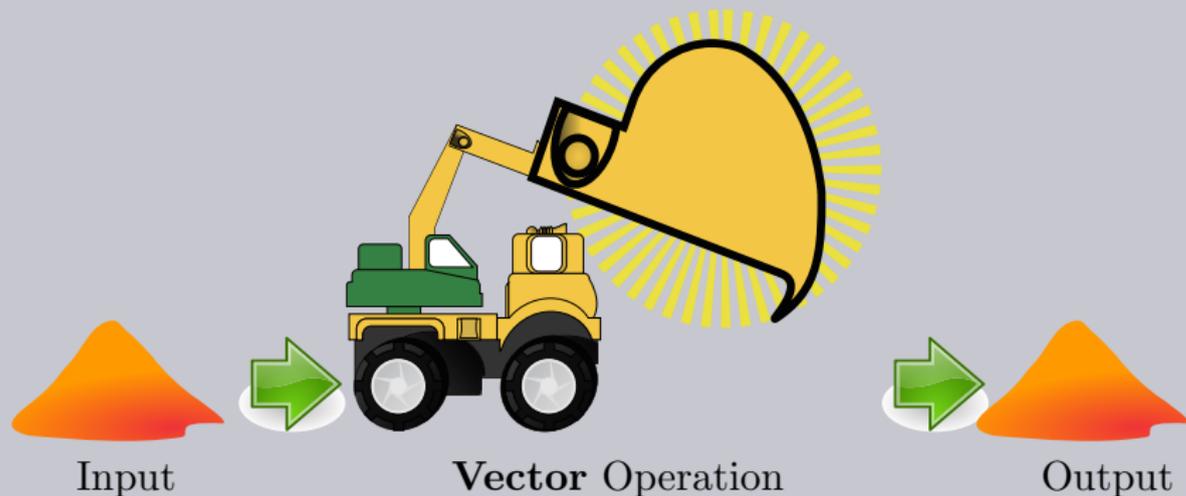
単純なスピードアップはそろそろ限界？

計算の高速化手法 IV-1

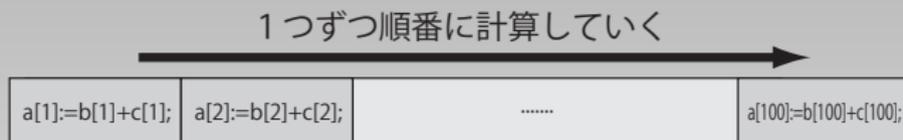
一度に沢山行え！

似たような処理をまとめて一度に行う。

- ベクトル計算
- SIMD (Single Instruction Multiple Data)



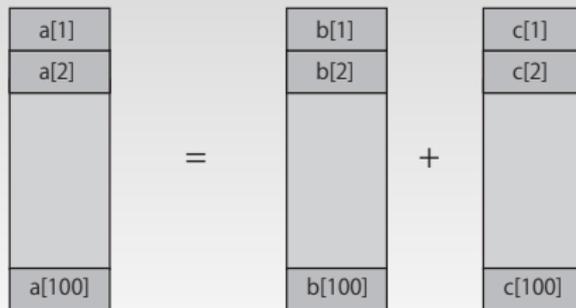
計算の高速化手法 IV-2



スカラー計算



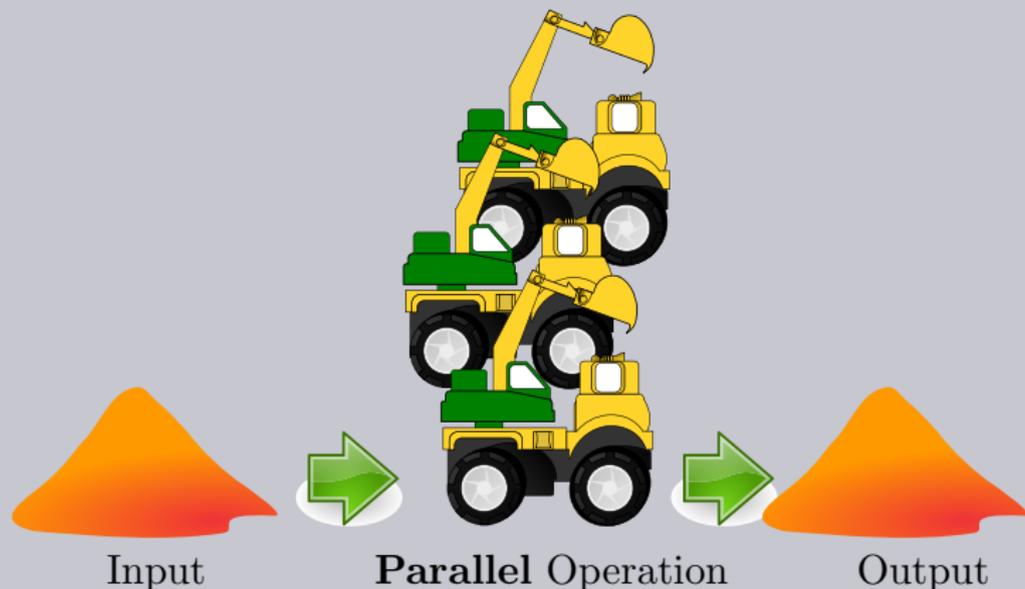
一度に全部計算できる



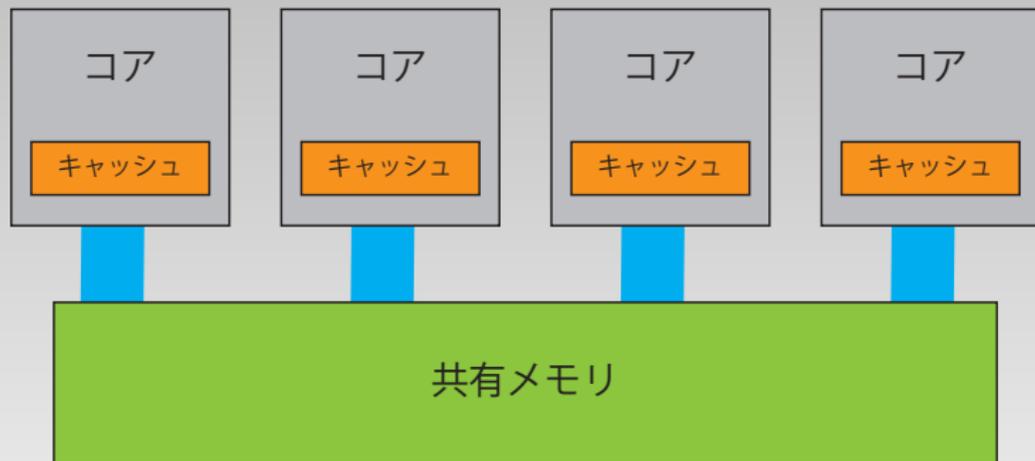
ベクトル計算

皆でやれ！ 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．



メモリ共有型計算機概念図

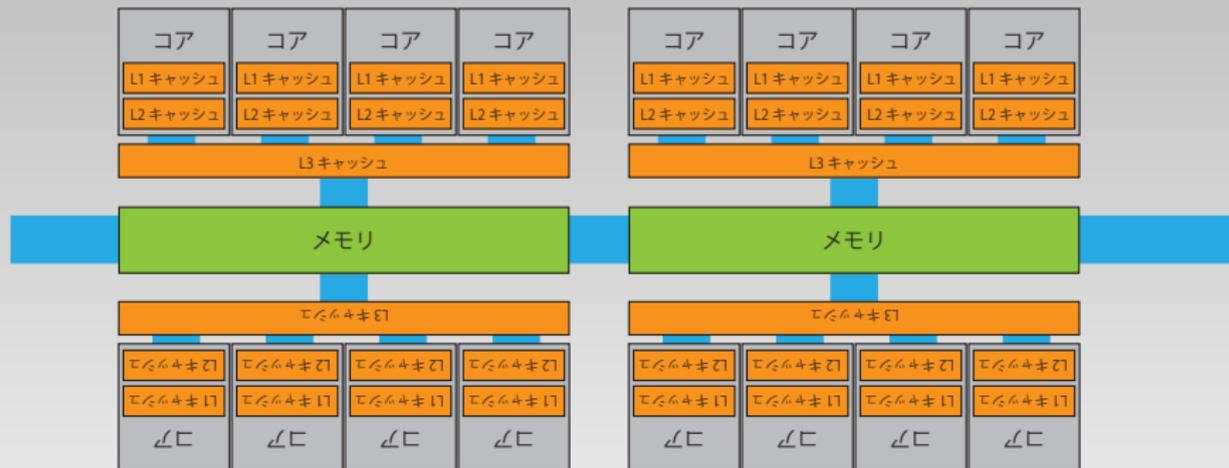
皆でやれ！ メモリ分散型

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, うまくいけば高速化しやすい.

計算の高速化手法 VI-2



メモリ分散型計算機 の概念図

(ついでに) 平行化

プログラムを、「どういう順序で実行しても良いように分割する」ことを平行化などと言う。

- これができれば，並列化にとっても役に立つ。
- 難しい。

0434-J

現状は

- 「単純に速く」というのは段々困難に.

現状は

- 「単純に速く」というのは段々困難に.
- ベクトル型スーパーコンピュータは今や 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.

現状は

- 「単純に速く」というのは段々困難に.
- ベクトル型スーパーコンピュータは今や 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.
- 多くのスーパーコンピュータは超並列 (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.

0434-J

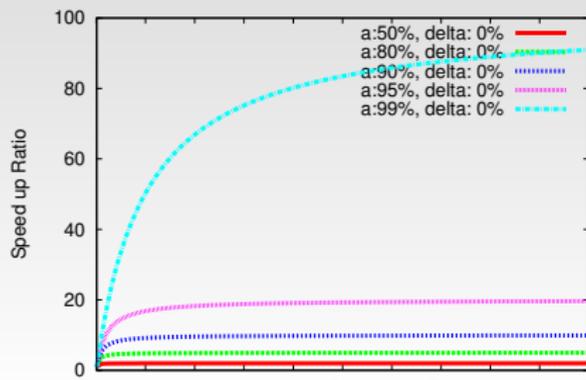
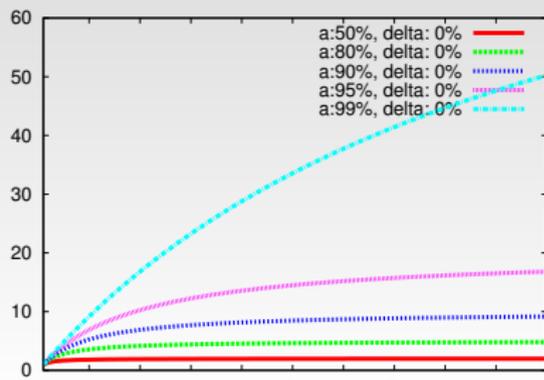
並列計算の効率**限界**

並列計算でどれくらい速くなるのか: アムダール則 I

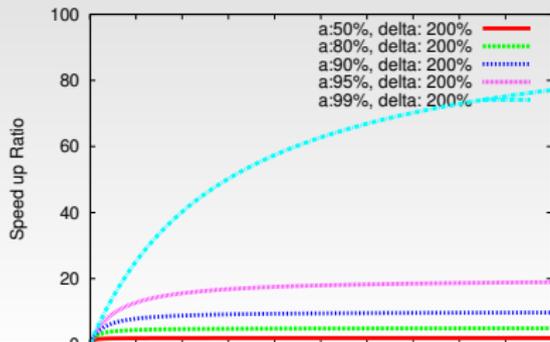
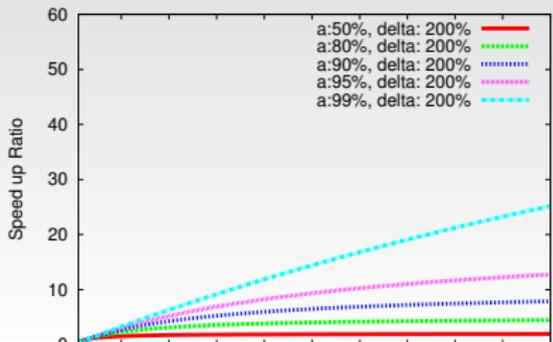
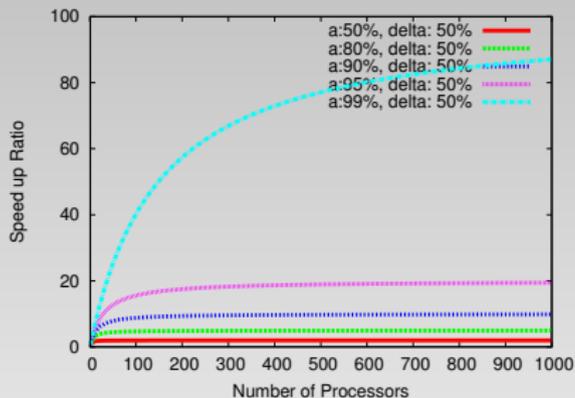
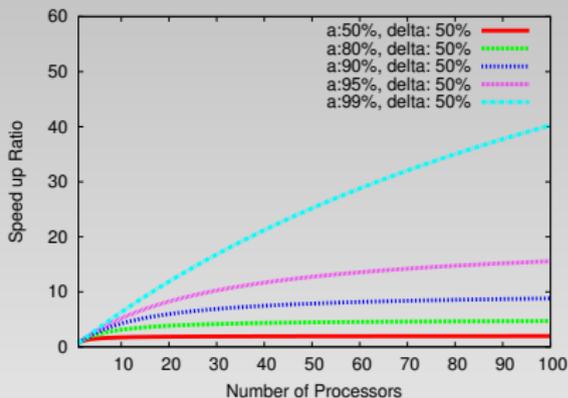
プログラム中、割合 a の部分を n 個のプロセッサで並列化し、残り $1 - a$ の部分を並列化しない場合、全体は

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

に高速化される。 δ は並列化に伴って発生する通信等による遅延率。



並列計算でどれくらい速くなるのか II



並列計算でどれくらい速くなるのか III

結局…

結局…

- 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」.

結局…

- 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.

結局…

- 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…

0434-J

どのようなソフトウェアを使うべきか

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緩い方から並べると…

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緩い方から並べると…

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

一般的に、下のほうがプログラミングは容易。
ハードウェアは上のほうがスケールしやすい。

並列計算ソフトウェア I

小規模もしくは、使いやすい方から紹介する.

ベクトル化, SIMD

- 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…

小規模もしくは、使いやすい方から紹介する.

ベクトル化, SIMD

- 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…
- 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.

小規模もしくは、使いやすい方から紹介する。

ベクトル化, SIMD

- 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…
- 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。

並列計算ソフトウェア I

小規模もしくは、使いやすい方から紹介する。

ベクトル化, SIMD

- 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…
- 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。

例えば,

```
for i:=1 to 10000 do a[i] := 2*i;
```

というコードはコンパイルするだけで1クロックで実行される。

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.
- 移植性は高い.

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.
- 移植性は高い.
- 解説書多し.

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.
- 移植性は高い.
- 解説書多し.
- やや高速化しにくい.

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.
- 移植性は高い.
- 解説書多し.
- やや高速化しにくい.
- ソフトウェアとしては

CPU 内 / 間 並列化: メモリ共有

- プログラミングは比較的容易.
「ここからここまで並列」というディレクティブをプログラムに書き入れるぐらいで済む.
- 移植性は高い.
- 解説書多し.
- やや高速化しにくい.
- ソフトウェアとしては
 - OpenMP
 - OS や言語の用意するライブラリ
 - Grand Central Dispatch (MacOS X 10.6, FreeBSD),
 - intel TBB など.

OpenMP

Fortran の例:

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

上のように、並列化したいところをディレクティブで挟むのが基本。
並列度等はコンパイラや環境変数で指定。

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- プログラムの動作を並列化用に設計する必要あり.

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- 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.

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- 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. そのため, 移植性はやや低い.

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- 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. そのため, 移植性はやや低い.
- 解説書多し.

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- 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. そのため, 移植性はやや低い.
- 解説書多し.
- スーパーコンピュータはほぼこれ (NEC 除く).

マシン間並列化: 分散メモリ

- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- 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. そのため, 移植性はやや低い.
- 解説書多し.
- スーパーコンピュータはほぼこれ (NEC 除く).
- 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.

MPI

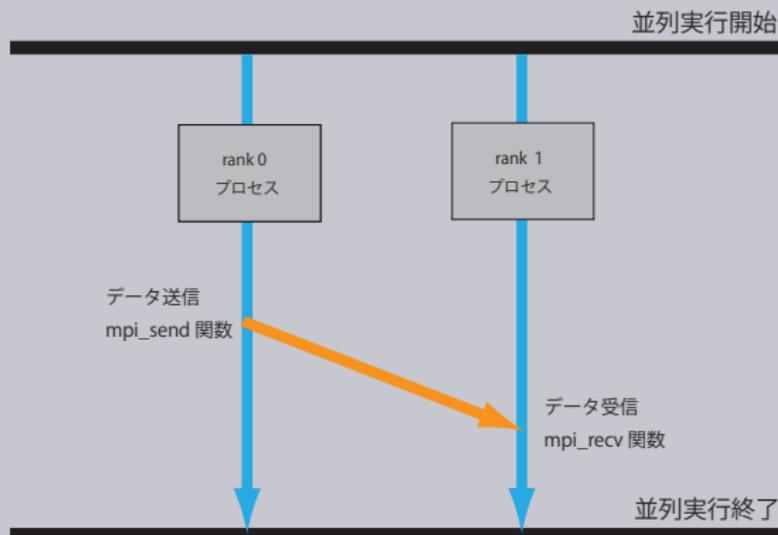
- データは分散しており，他プロセスのデータに触れない.

MPI

- データは分散しており，他プロセスのデータに触れない.
- データはお互いに「明示的に」通信する必要がある.

MPI

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



- ハードウェアによって並列化の方法が異なるので，ソフトウェアもそれに合わせて選択する.

並列計算のまとめ

- ハードウェアによって並列化の方法が異なるので，ソフトウェアもそれに合わせて選択する.
- 他のソフトウェアに比較すると，MPI はプログラムを書く人が並列化を考えねばならず，やや敷居が高い.

並列計算のまとめ

- ハードウェアによって並列化の方法が異なるので、ソフトウェアもそれに合わせて選択する。
- 他のソフトウェアに比較すると、MPI はプログラムを書く人が並列化を考えねばならず、やや敷居が高い。
- 阪大のスーパーコンピュータ (SX-9) はベクトル型なので、プログラムテクニ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だが。

並列計算のまとめ

- ハードウェアによって並列化の方法が異なるので、ソフトウェアもそれに合わせて選択する.
- 他のソフトウェアに比較すると、MPI はプログラムを書く人が並列化を考えねばならず、やや敷居が高い.
- 阪大のスーパーコンピュータ (SX-9) はベクトル型なので、プログラムテクニ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だが.
- 実は普通の PC でも 4 コア持っていたりするので、4 倍ぐらいまでの並列化は容易にできたりする.

並列計算のまとめ

- ハードウェアによって並列化の方法が異なるので、ソフトウェアもそれに合わせて選択する。
- 他のソフトウェアに比較すると、MPI はプログラムを書く人が並列化を考えねばならず、やや敷居が高い。
- 阪大のスーパーコンピュータ (SX-9) はベクトル型なので、プログラムテクニ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だが。
- 実は普通の PC でも 4 コア持っていたりするので、4 倍ぐ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。興味のある方はぜひそちらへ。

UNIX

- (F) “応用数理学 7”, 降旗 大介.
- (K)§1 “UNIX の復習”, 中村 匡秀.

並列化について

- (K)§3 “並列計算とは”, 山本 有作.

ベクトル化

- “ベクトル型計算機のためのベクトル計算”, 菊池 誠 (阪大 サイバーメディアセンター), スパコン 2011 配布資料.
- “パソコン & スーパーコンピュータで計算するためのベクトル & 並列入門”, 福田 優子 (阪大 レーザー研).

OpenMP

- (K)§4 “OpenMP を用いた並列計算”, 谷口 隆晴.
- (T) “科学技術計算のためのマルチコアプログラミング入門”, 中島 研吾.

MPI

- (K)§5 “MPI を用いた並列計算”, 山本 有作.
- (T) “MPI 基礎: 並列プログラミング初級入門”, 片桐 孝洋.

その他

- (O) “大規模計算機システムガイドブック”.
- (O) “パソコン&スーパーコンピュータで計算するための基礎知識”, 福田 優子 (阪大 レーザー研).
- (O) “スーパーコンピュータ利用入門 (講習会入門編資料) ”.

F: (降旗) 応用数理学 7 授業用 web.

<http://www.cas.cmc.osaka-u.ac.jp/paoon/Lectures/2012-7Semester-AppliedMath7/>

K: 神戸大学大学院システム情報学研究科・計算科学専攻の計算科学演習用 web.

http://exp.cs.kobe-u.ac.jp/wiki/comp_practice/index.php?%B7%D7%BB%BB%B2%CA%B3%D8%B1%E9%BD%AC

O: 大阪大学サイバーメディアセンター大規模計算機システム

<http://www.hpc.cmc.osaka-u.ac.jp/j/index.html>

T: 東京大学情報基盤センタースーパーコンピューティング部門講習会資料

http://www.cc.u-tokyo.ac.jp/support/kosyu/schedule_kosyu.html

Thank You!

Thank You!

0434-J