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

2019.08.29

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？

0444-J

構成

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

0444-J

詳しく知りたくなったら？

ぜひ，より詳しい講習会 (下記: 本日以降) へ！

8 月 29 日 (本日)	スパコンに通じる並列プログラミングの基礎
9 月 5 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 *
9 月 11 日	SX-ACE 高速化技法の基礎 *
9 月 12 日	並列コンピュータ 高速化技法の基礎 *
9 月 19 日	並列プログラミング入門 (MPI) *
9 月 25 日	AVS 可視化処理入門
9 月 26 日	AVS 可視化処理応用 / 特別相談会

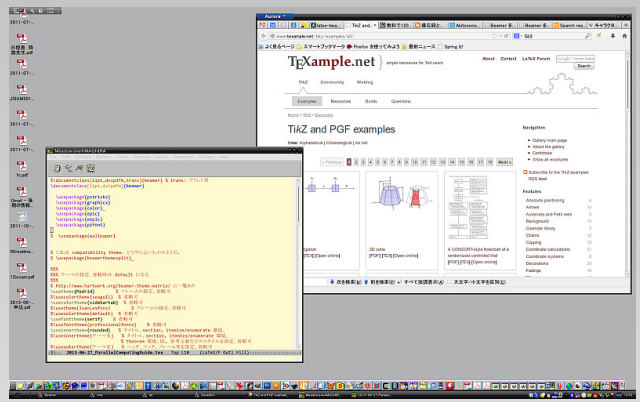
* 一週間使える無料お試しアカウントつきです！

http://www.hpc.cmc.osaka-u.ac.jp/lecture_event/lecture/

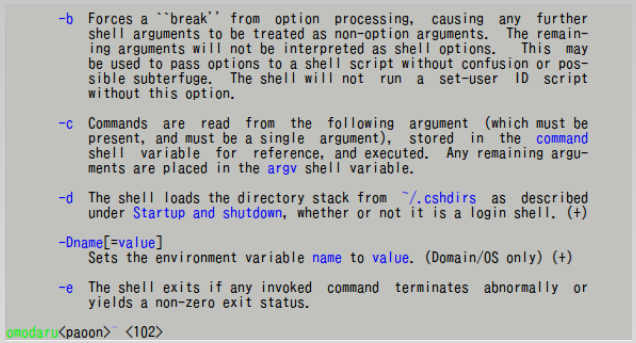
Part I: UNIX 入門

コマンドで操作する CUI とは

GUI と CUI, OS の関係



GUI



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.

0445-J

GUI と CUI

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 地図で，**CUI** は 写真である！

GUI = 地図

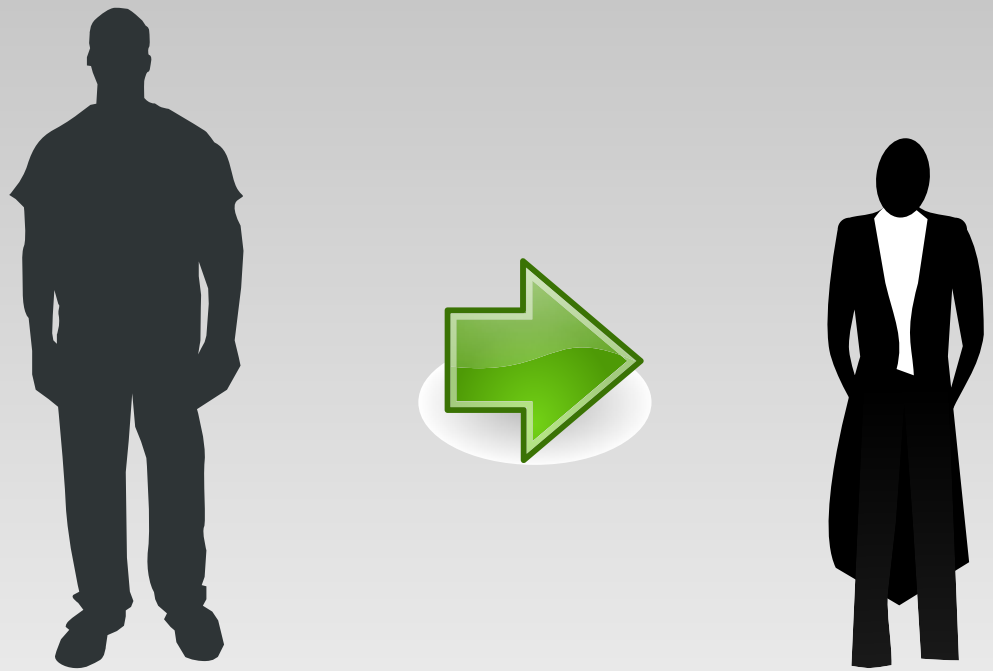
CUI = 写真

CUI では常に「今どこにいるか」を認識する必要がある．
CUI では他の場所の情報は「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．

0445-J

CUI を理解するコツ II

CUI の操作 = 執事 **(shell)** への命令 である.



ユーザー　　命令　　シェル

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いるので，好みで変えよう!

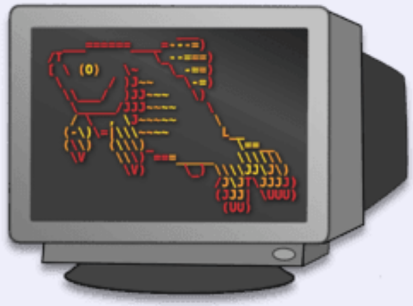
0445-J

CUI を理解するコツ II-2

おすすめシェル: **fish**

Finally, a command line shell for the 90s

fish is a smart and user-friendly command line shell for Linux, macOS, and the rest of the family.



- 補完が賢いよ (マニュアルの情報も同時に見せてくれたりする)
- 文法がまともだ (esac なんて打つ必要はない)
- 文法にあわせて色を付けるよ
- 設定は web browser で
- コマンドに合わせた動作をするよ

CUI を理解するコツ III

CUI は遠隔操作でよく使われる

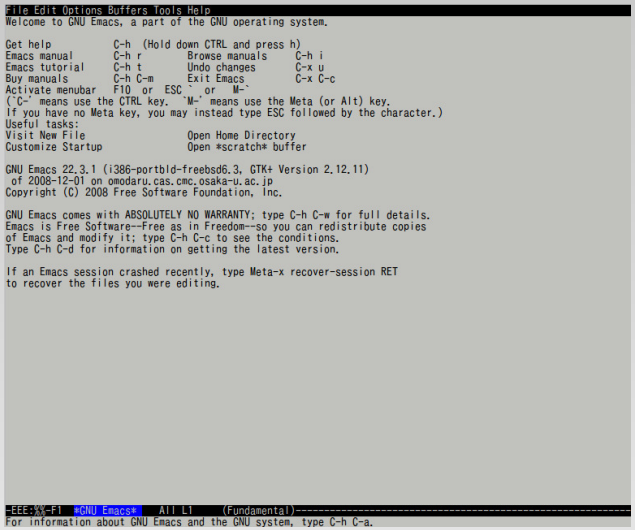
手元の端末　　ネットワーク　　Unix サーバ

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．

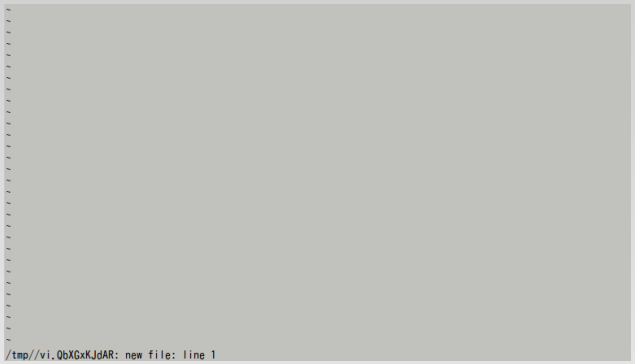
0445-J

CUI を理解するコツ IV

CUI で使われるエディタは事実上 **Emacs** か **vi** に限られている



Emacs



vi

普通は (まだ) とっつきやすい emacs がお勧め.
管理人は vi が使えないと困るよ.

0445-J

Unix コマンド入門
これだけ知っていれば戦える

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.

0445-J

Unix コマンド: ファイル操作

- `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 poko`　hoge というファイルを 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.

0445-J

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.

0445-J

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-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`　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).

0445-J

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).

おまけ: Emacs + vi = spacemacs: 速くて強力

2014.10 first release. 現在は release 版は ver.0.200.13, 開発版は 0.300.0.

0445-J

Unix 入門のまとめ

- **Unix** がわからない = **CUI** に慣れてないだけということが多い.
- CUI を理解するにはいくつかコツが有る.
  - CUI はその性質が “写真” 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.

0445-J

Part II: 並列計算入門

この Part の構成

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，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

0491-J

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

- 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.
  - 百万単位のプロセッサコア (Top500 2019.06 時点)

    Summit (アメリカ, IBM Power9, 1 位): **149 PFlops**, コア数 2,414,592.
    神威・太湖之光 (中国, Sunway, 3 位): **93 PFlops**, コア数 10,649,600.
    Piz Daint(スイス, Xeon, 6 位): **21 PFlopes**, コア数 387,872.
    ABCI (日本 産総研, Xeon, 8 位): **19.9 PFlops**, コア数 391,680.

- 巨大メモリ
  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 (… 困るが)
  Summit: **10.1MW**, 神威: **15.4 MW**, Piz Daint: **2.4MW**, ABCI: **1.2MW**
  * 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

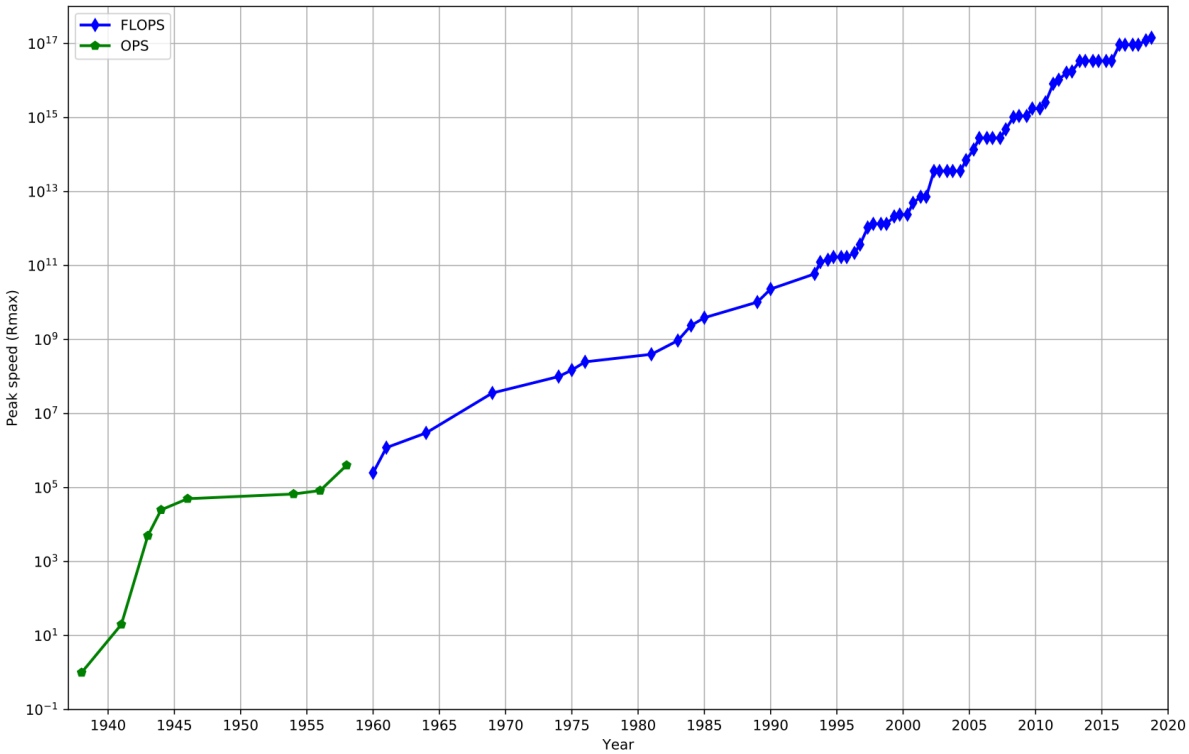
- 計算性能
  3 クラスタ合計 423 TFlops = **0.423 PFlops**, コア数 6144.

  * 合計だと 2019 年 6 月の TOP500 の 500 位 (1.02 PFlops) の 4 割.
  (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
  96TB ≅ 0.1 PB.
- 電力
  (わずか？) 700 KW.
  * だいたい、コタツ 1000 個分.

… Summit に比して，性能 0.35%, メモリ 3.5%, 電力消費量 8%, ぐらいか.

0491-J

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 CC0 1.0 Universal Public Domain Dedication)

0491-J

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 をとにかく数多く繋ぐ．93 PFLOPS.
- Summit(2018) CPU 数や電力に配慮が… 122 PFLOPS (現在は 149).

0491-J

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の所持・開発に絡んでいる国は…

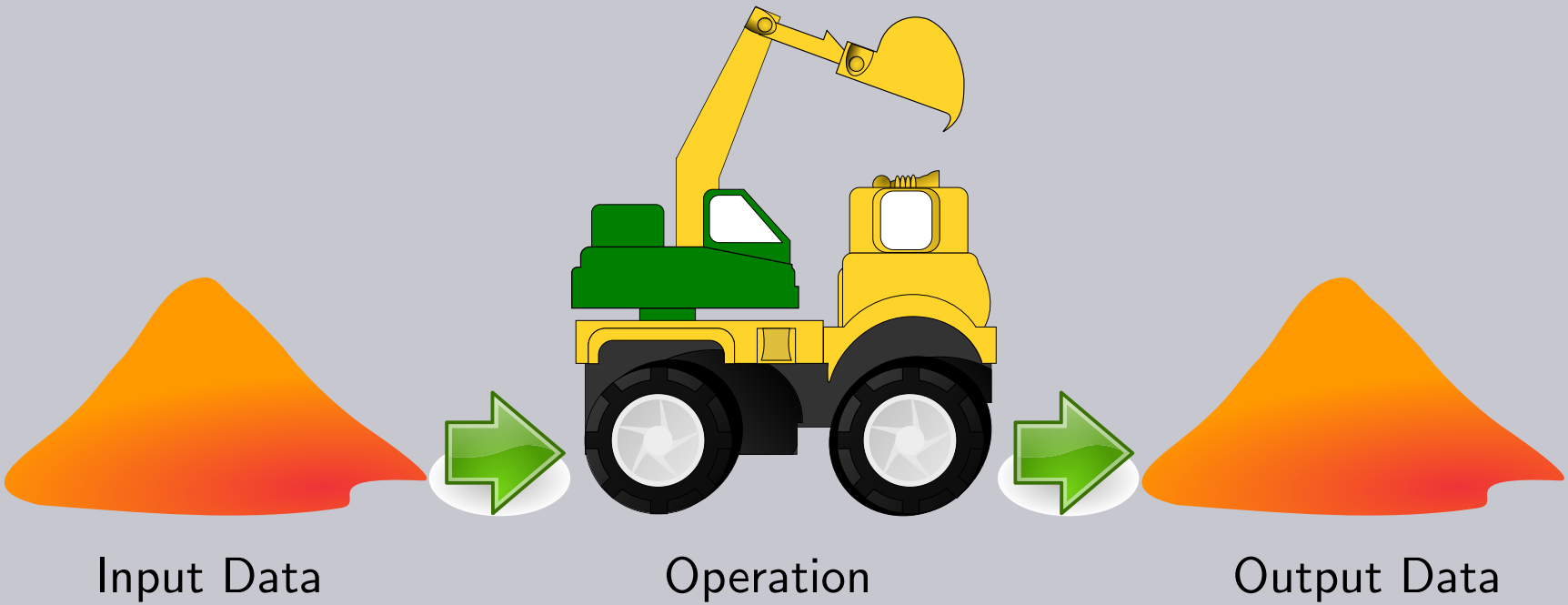
- アメリカ (116/top 500, 2019.06, 2 位)： CPU 数を増やして能力を稼ぐ. Cray 社と IBM 社が強い．CPU を自国で開発できる強みあり.
- 日本 (同 28, 3 位)： ベクトル型スーパーコンピュータを作る唯一の国か. NEC と富士通，日立が主なメーカー．CPU は自国開発ものと輸入物の混在.
- 中国 (同 220, 1 位)： ほぼアメリカ製だったが、近年は CPU の自力開発も. この数年急激に数を増やしていたが，減速.
- フランス (20), イギリス (18), アイルランド (13), オランダ (13), ドイツ (13), 残りの国は 10 未満 (top500 中).

0491-J

計算の高速化手法, 概要とハードウェア

計算の高速化手法 I

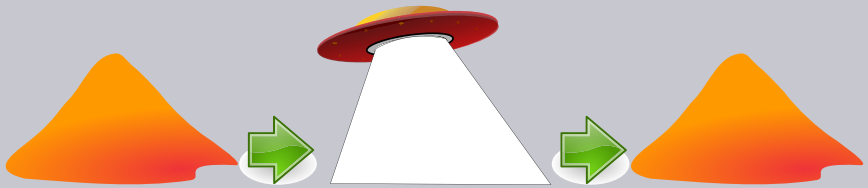
そもそも計算とは



計算の高速化手法 II

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： $\mathbf{1+2+3+\cdots+100=5050}$ に対して，

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

を改善して，次のように．

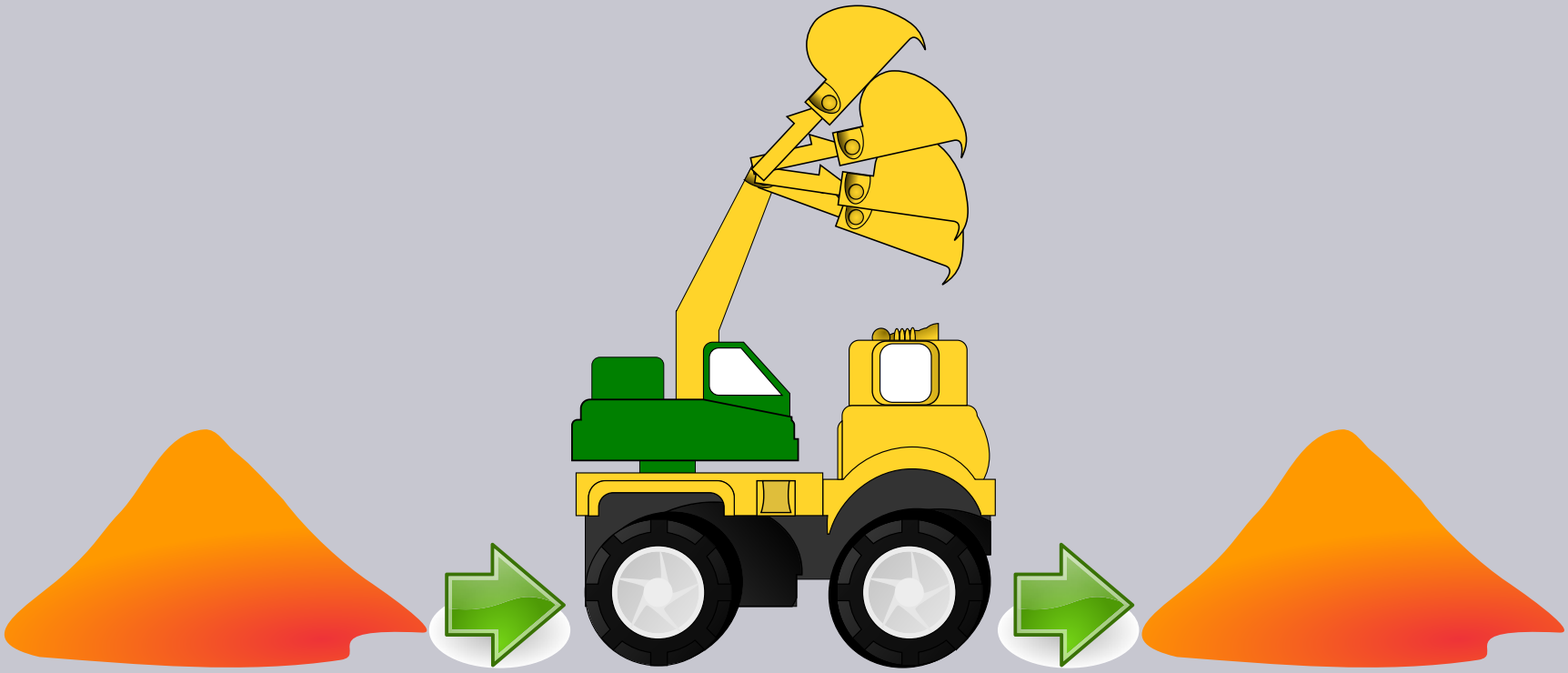
```
( 100 + 1 ) * 100 / 2
```

0491-J

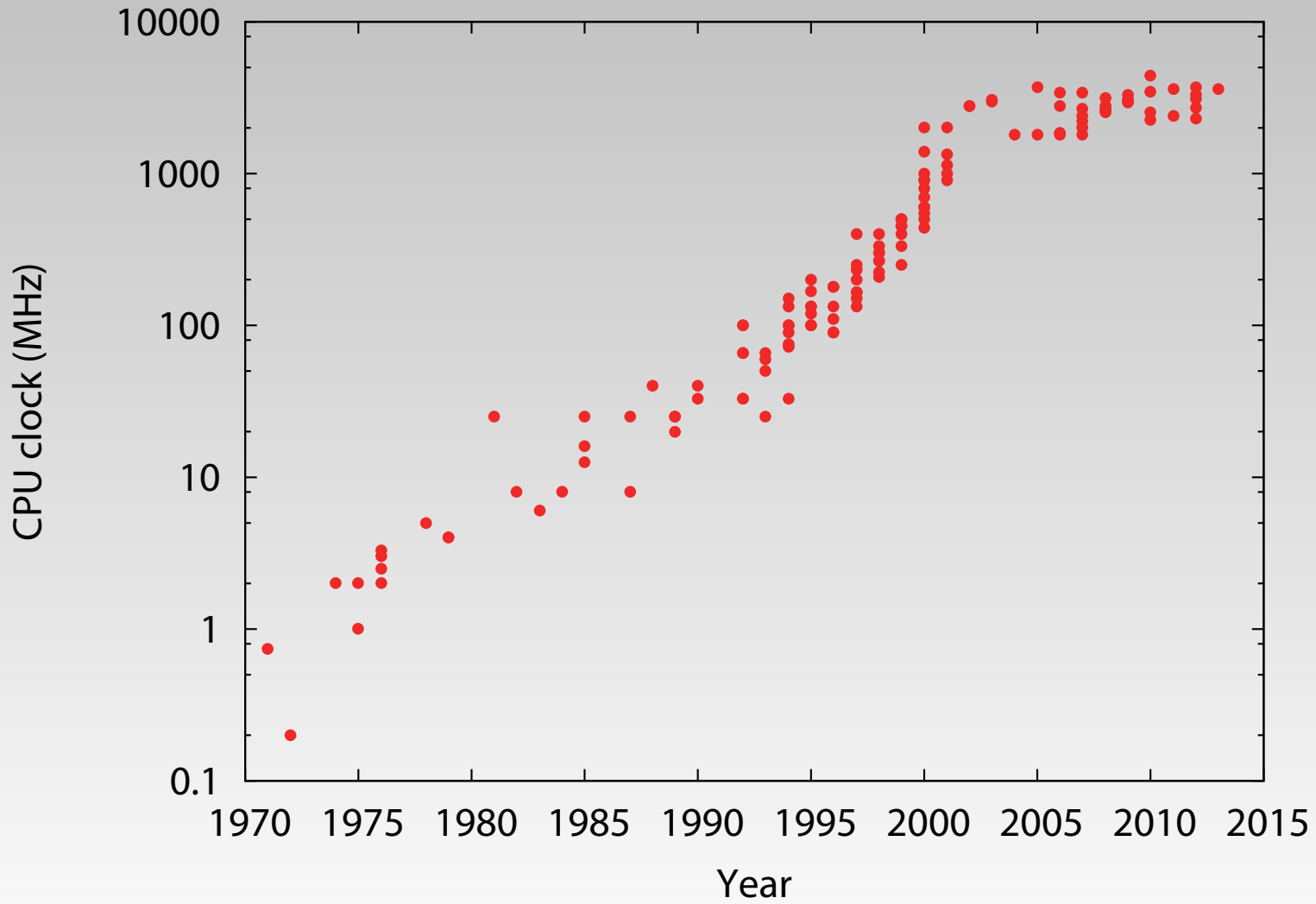
計算の高速化手法 III-1

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



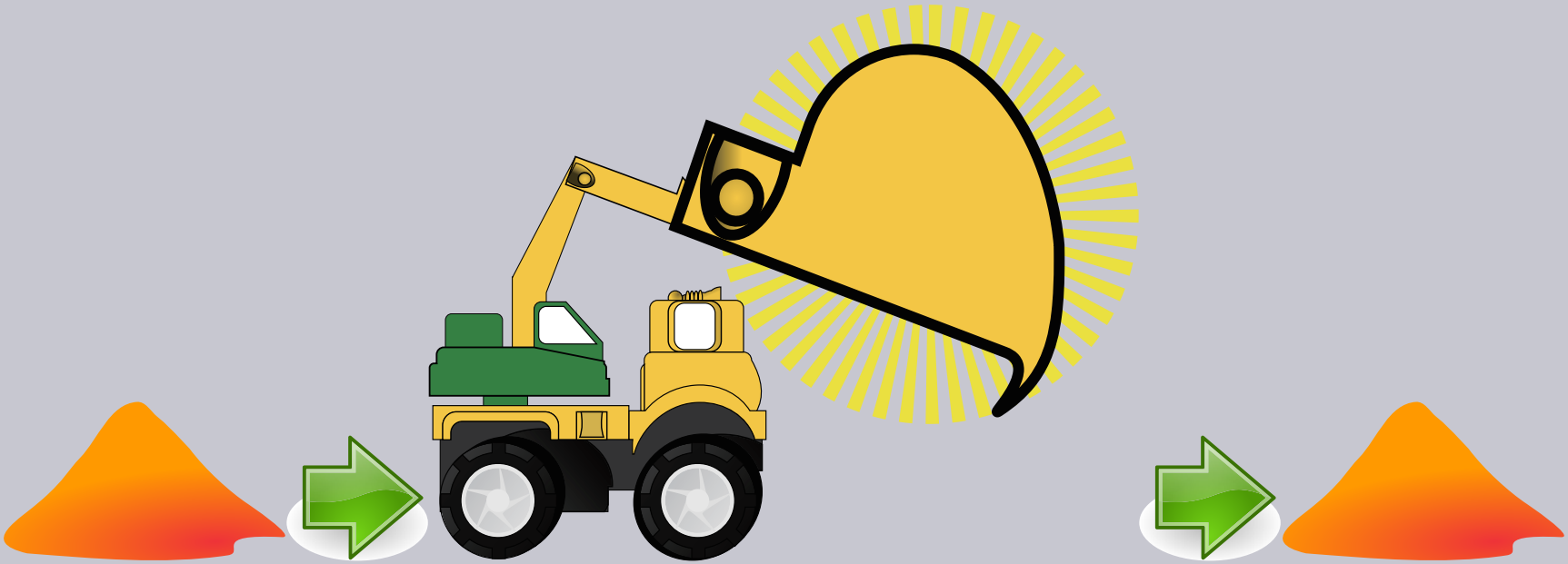
単純なスピードアップはそろそろ限界？

計算の高速化手法 IV-1

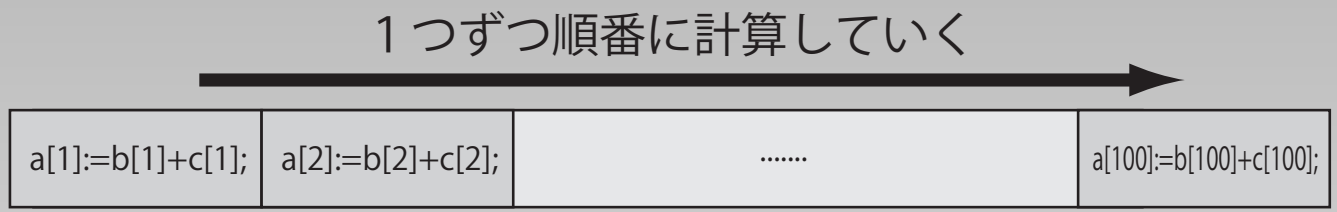
一度に沢山行え！

似たような処理をまとめて一度に行う (ハードウェアで).

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

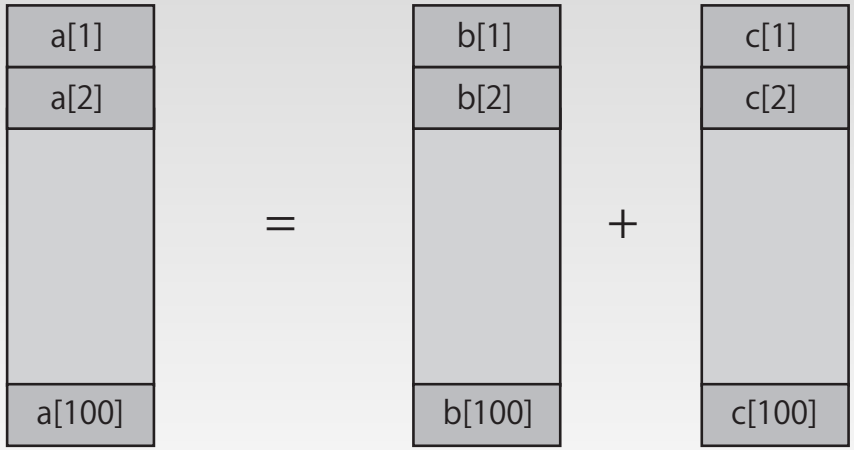


計算の高速化手法 IV-2



スカラー計算

一度に全部計算できる

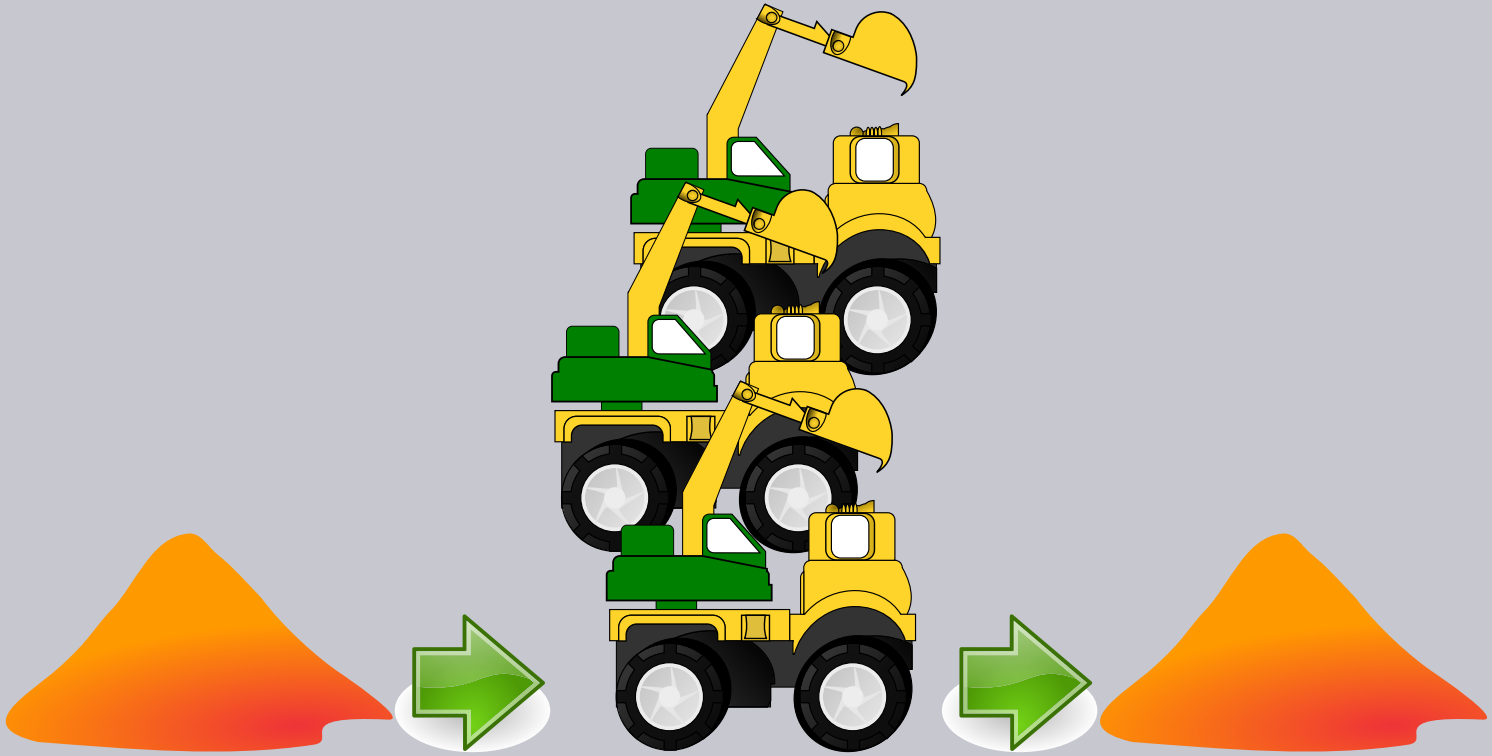


ベクトル計算

計算の高速化手法 V-1

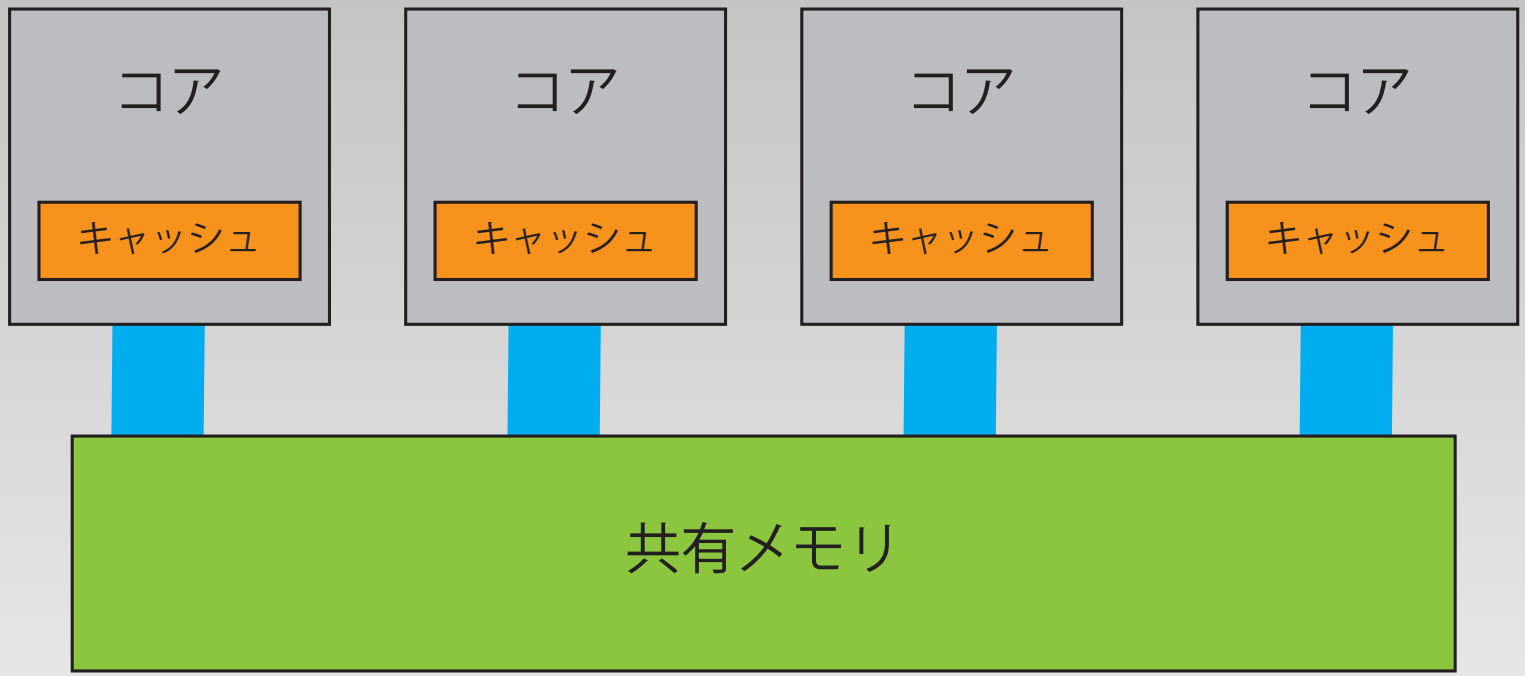
皆でやれ！ メモリ共有型

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.

計算の高速化手法 V-2



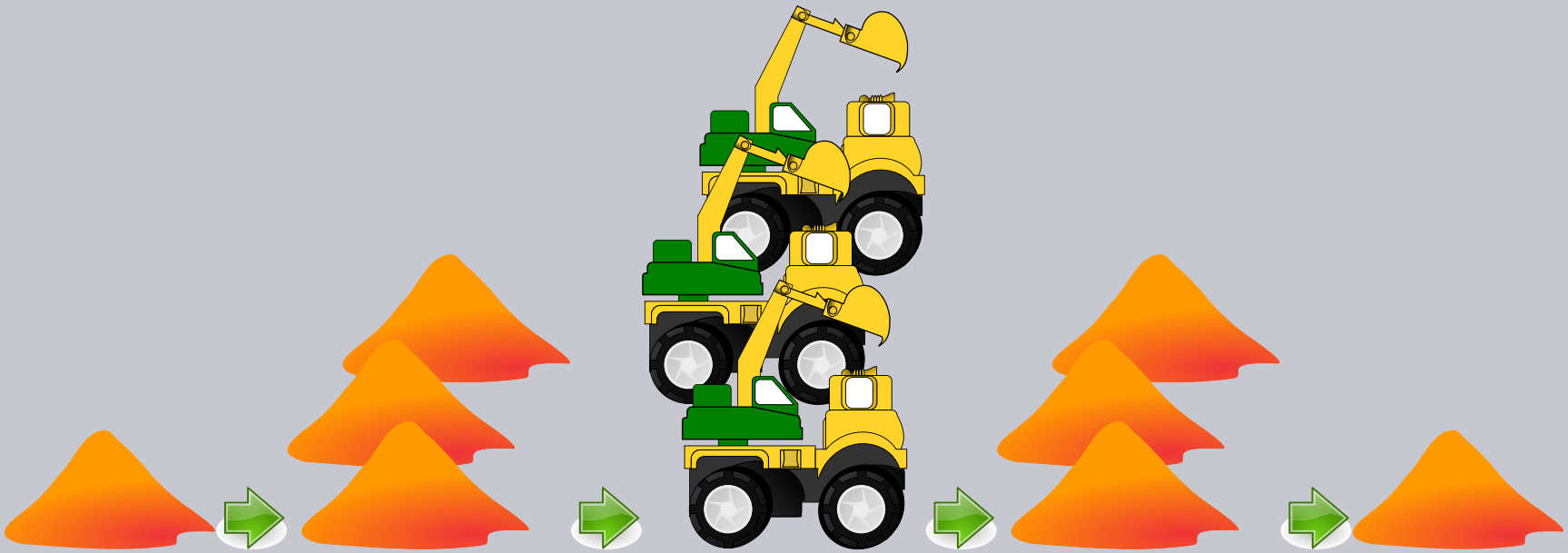
メモリ共有型計算機の概念図

0491-J

計算の高速化手法 VI-1

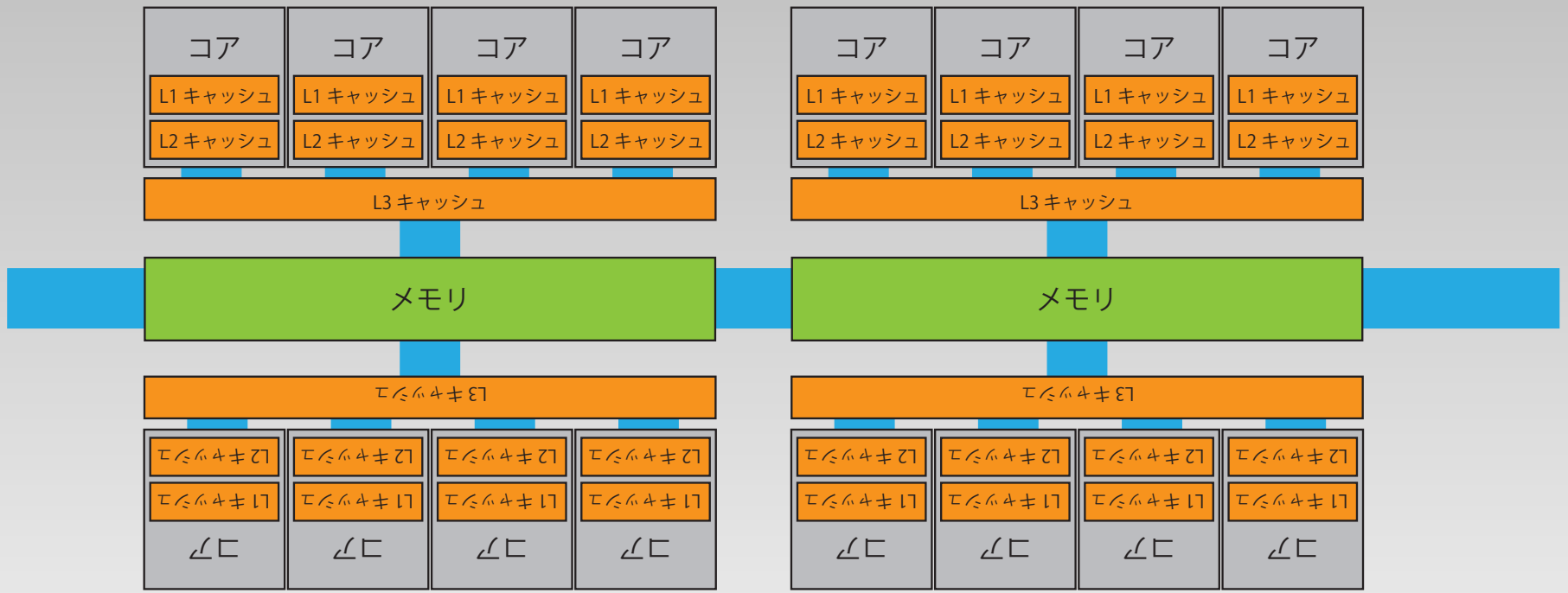
皆でやれ！ メモリ分散型

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

0491-J

計算の高速化手法 VII

(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，並列化にとても役に立つ.
- 難しい.

0491-J

計算の高速化手法 VIII

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.

0491-J

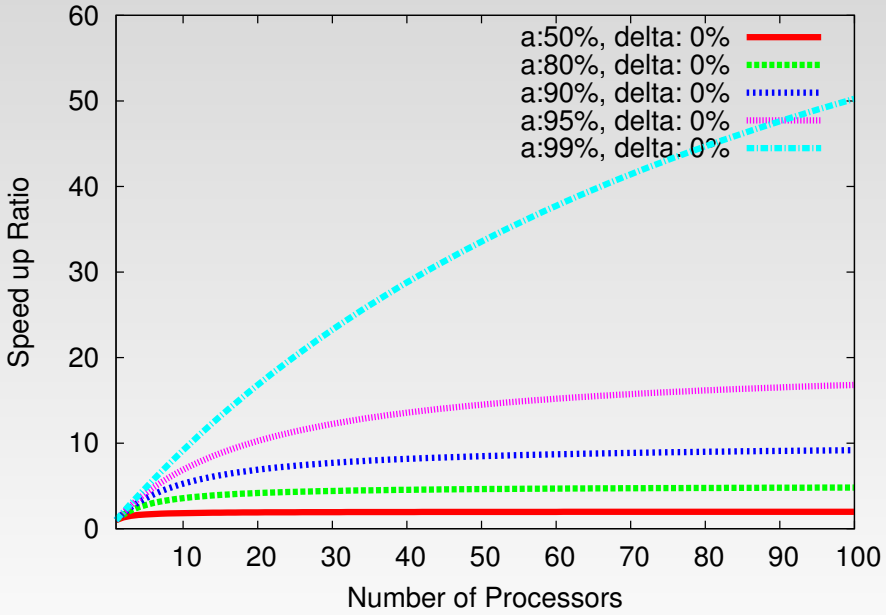
並列計算の効率限界

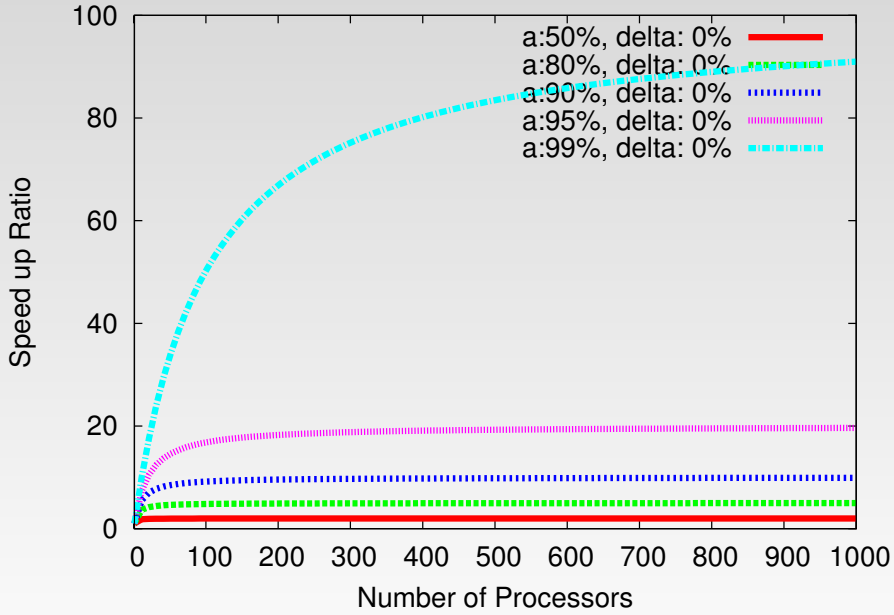
並列計算でどれくらい速くなるのか: アムダール則 I

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

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

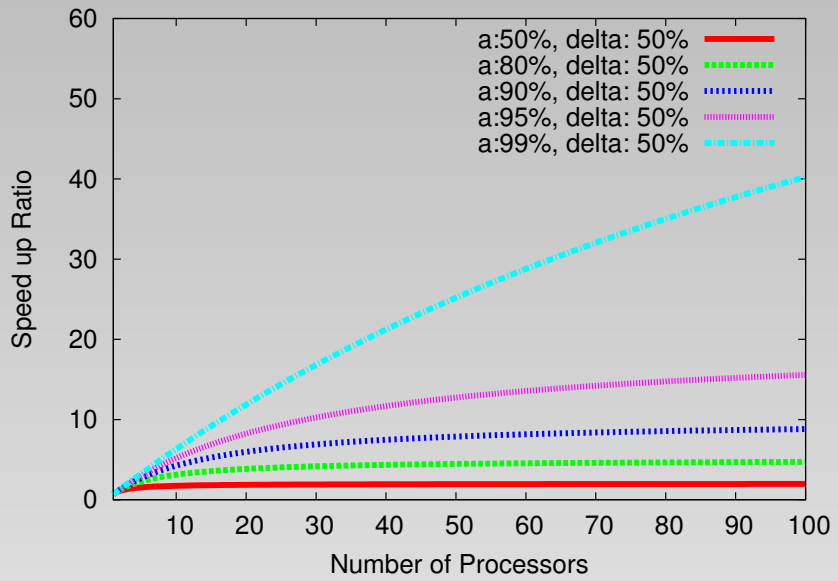
に高速化される．δ は並列化に伴って発生する通信等による遅延率.

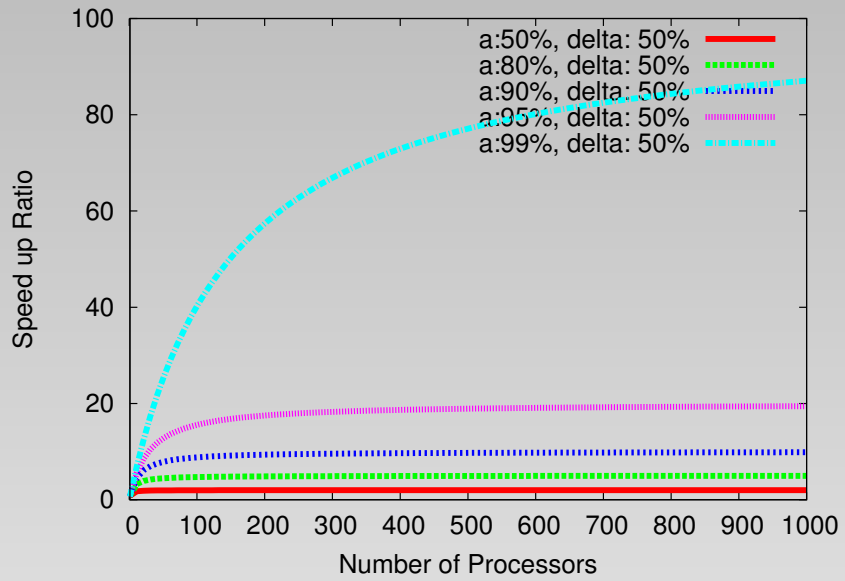


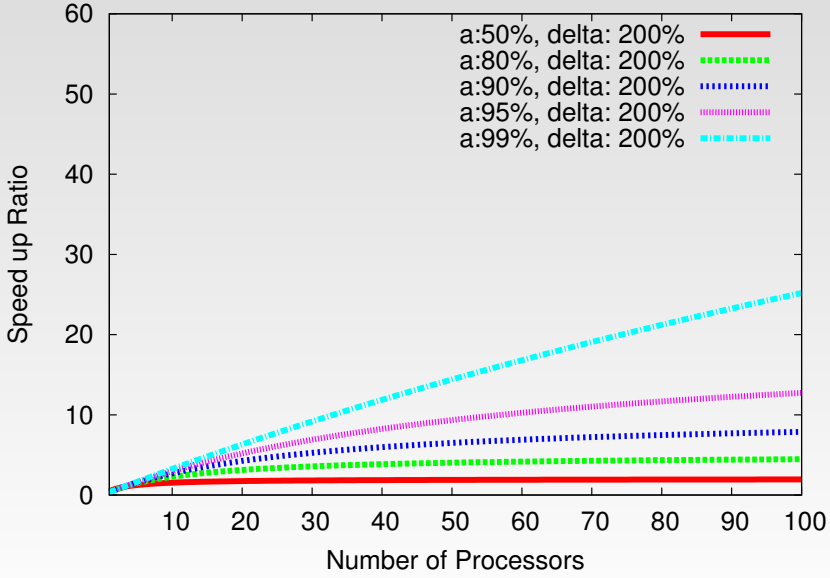


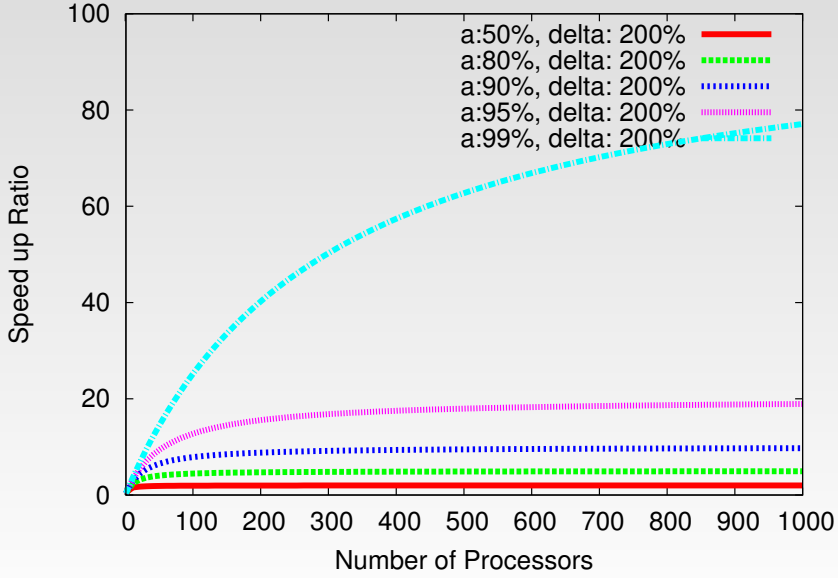
0491-J

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…

- 並列化できない箇所が「信じられないぐらい足を引っ張る」.
- 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…

0491-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…

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), 他
— 大きな壁 —		
CPU 間並列 (同じ MB 上で)	共有	OpenMP (Multi Processing), 他
CPU 内並列	共有	OpenMP, 他沢山
SIMD/ ベクトル化	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…
- 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．

0491-J

サンプル: SIMD / ベクトル計算: 大変容易.

SIMD / ベクトル 計算の例: 内積

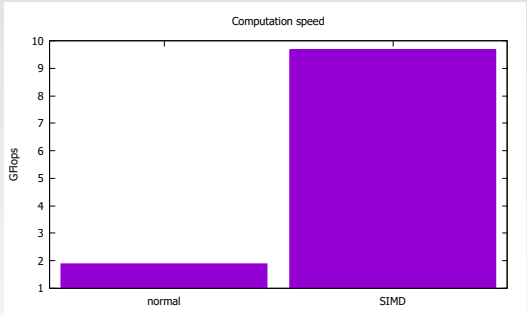
C/C++ on SX-ACE:

```
#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
```



SIMD を使うことで (4 core CPU で) 計算速度が 5 倍以上に!!

* ノート PC (vaio, 4core), 1000 次元ベクトル, 100000 回平均.

並列計算ソフトウェア II-1

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. (thread を用いると少し難しい)
- 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, Google Go, Rust など.

0491-J

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
```

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.
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.

0491-J

サンプル: OpenMP 的な並列化 (2)

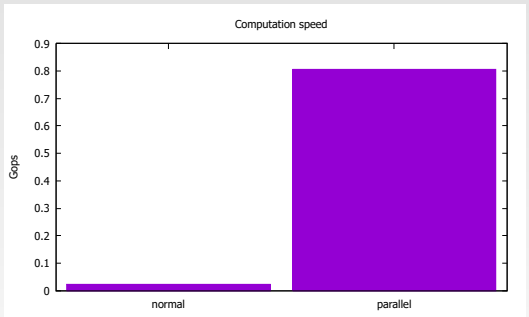
Parallel プログラムの例: コイントス

Julia lang:

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

C/C++ on SX-ACE:

compile option に `-Pauto` をつけるだけでも良い…



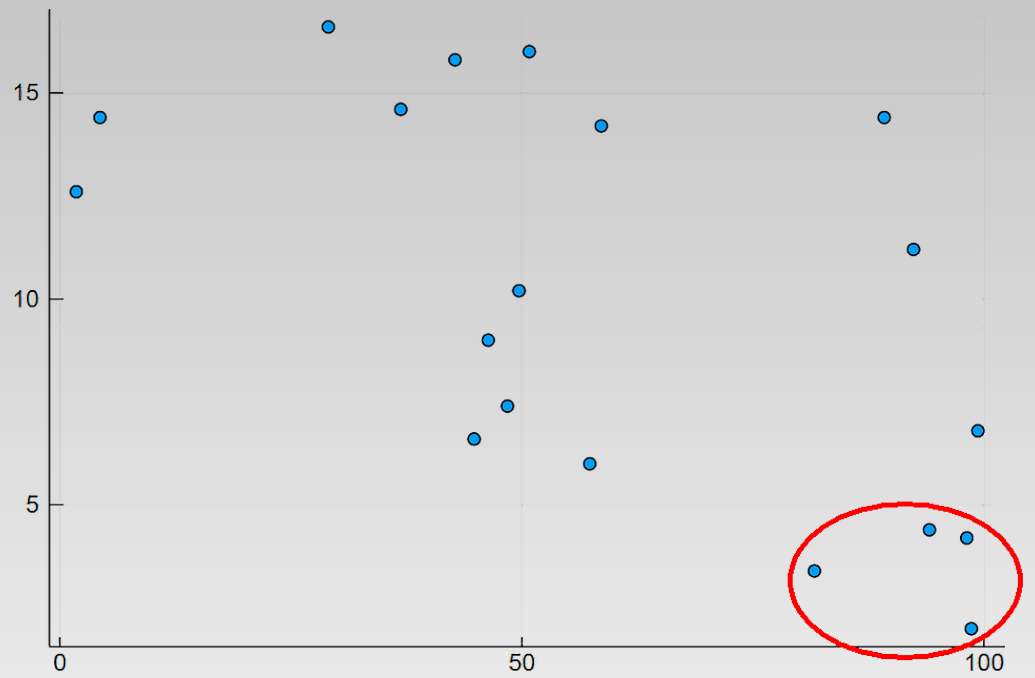
並列計算を使うことで (4 core CPU で) 計算速度が 33 倍以上に!!

* ノート PC (vaio, 4core).

サンプル: OpenMP 的な並列化 (3)

スパコン 2017 にて、高校生 20 チームが SX-ACE を使ったところ…

ただし、SIMD と 自動 OpenMP 相当まで.



横軸: 5 題平均ベクトル化率, 縦軸: 5 題平均順位

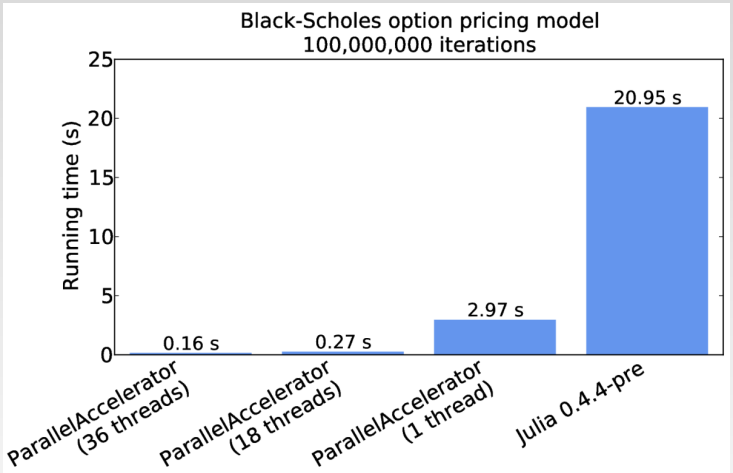
- 強いチームはベクトル化も出来ている.
- SIMD, OpenMP の並列化手法ならば、通常のプログラミング能力で十分.

0491-J

サンプル: OpenMP 的な並列化 (4)

Parallel プログラムの例: Black-Scholes 問題 (by Julia language)

```
using ParallelAccelerator ← Intel Labs 製.  最適化 & OpenMP.
@acc begin
   …プログラム本体…
end
```



http://julialang.org/blog/2016/03/parallelaccelerator より.
並列計算を使うことで (36 core CPU で) 計算速度が 130 倍以上に!!

0491-J

並列計算ソフトウェア III-1

マシン間並列化: 分散メモリ

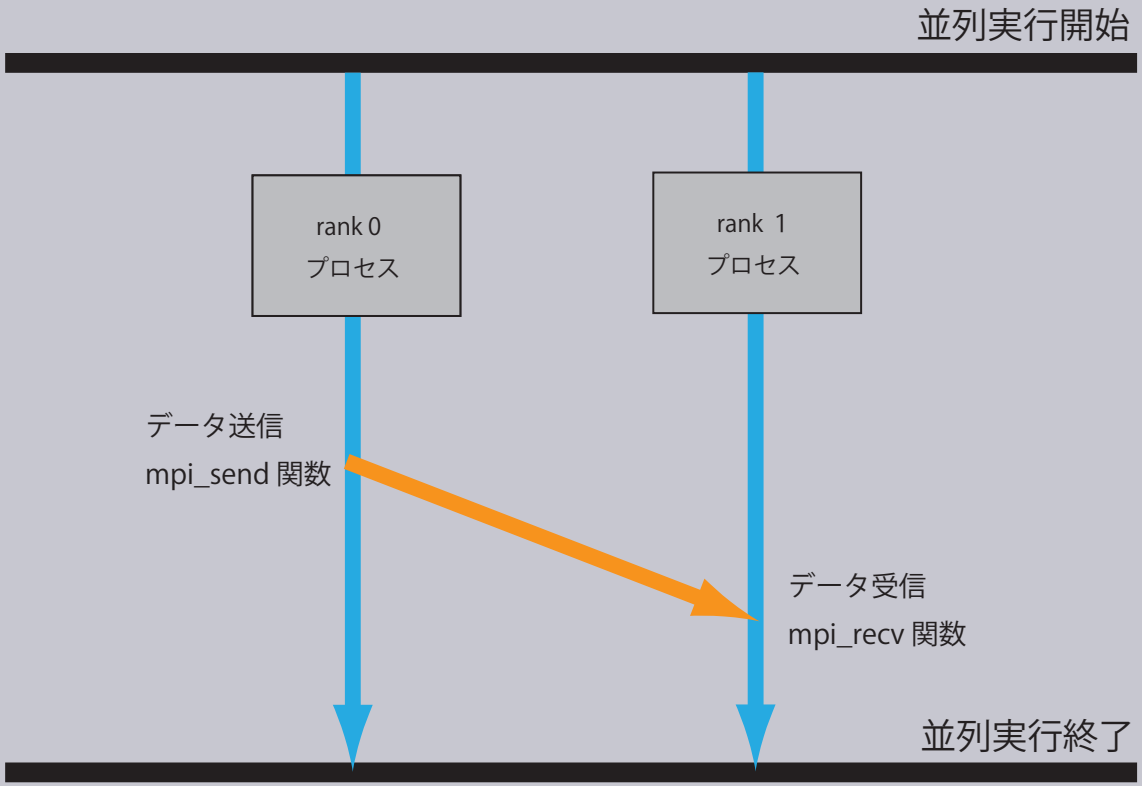
- 並列して動く各部分同士の情報のやりとりを 通信 (Message) という形で行う.
- プログラムの動作を並列化用に設計する必要あり.
- プログラムは面倒.
- ハードウェアへの依存性はやや高い．そのため，移植性はやや低い.
- 解説書多し.
- スーパーコンピュータはほぼこれ (NEC 除く).
- ソフトウェアは MPI (Message Passing Interface) という共通規格に沿った言語，ライブラリを利用.

0491-J

並列計算ソフトウェア III-2

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．



サンプル: MPI 並列化

Fortran の例: $n = 1000$ までの n の和を並列処理する

```
call MPI_INIT(ierr) !!  MPI 並列開始. nprocs 数だけのプロセス生成. 分身！
    call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr) !!  自分のプロセス rank は？
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr) !!  全体のプロセス数は？

    n = 1000 !!  n を nprocs で割り切れる前提のもとで.
    istart = (n/nprocs)*myrank + 1 !!  それぞれのプロセスで計算範囲を分割
    iend = (n/nprocs)*(myrank + 1)

    isum = 0
    do i = istart, iend !!  それぞれのプロセスで部分和を計算
      isum = isum + i
    end do

    !!  結果をプロセス 0 に集めて足す特別な命令 (プロセス 0 は他のプロセスより特別).
    call MPI_REDUCE(isum,sum,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)
    if (myrank .eq.  0) print *,sum
call MPI_FINALIZE(ierr) !!  MPI 並列終了.
```

上のように，各プロセスにさせる別々の仕事を「人間が」プログラムする (自動じゃない！).
C/C++ の場合もほぼ同様．並列度等はコンパイラや環境変数で指定.

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-ACE) はベクトル型計算機を束ねたものなので，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．
- コアの数より多めにプログラムを単純に並列するだけで早くなることも．
- 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．

0491-J

Thank You!

Thank You!

0491-J