

並列コンピュータがもっと速くなる方法

胡 曜

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研究部門

2024年10月26日

アウトライン

- 背景：計算機環境の変化
- 並列計算の基礎
- 並列計算の高速化
- 結論と今後の課題

背景：ムーア法則の終焉

- 「半導体チップの集積度は、約18ヶ月で2倍になる」

トランジスタの数
と性能は必ずしも
イコールではない

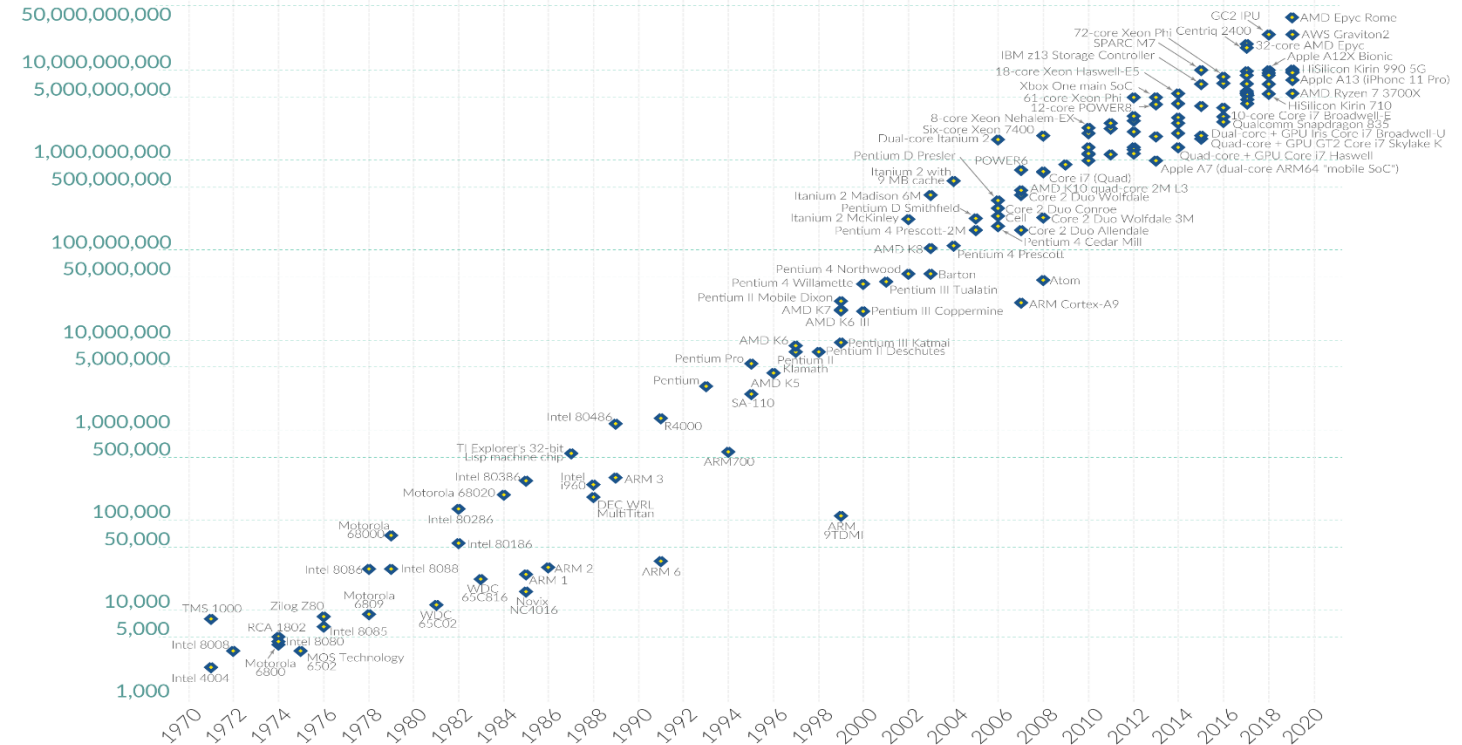
性能アップのため
には、様々な改良
が必要

Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World
in Data

Transistor count



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count) Year in which the microchip was first introduced

OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

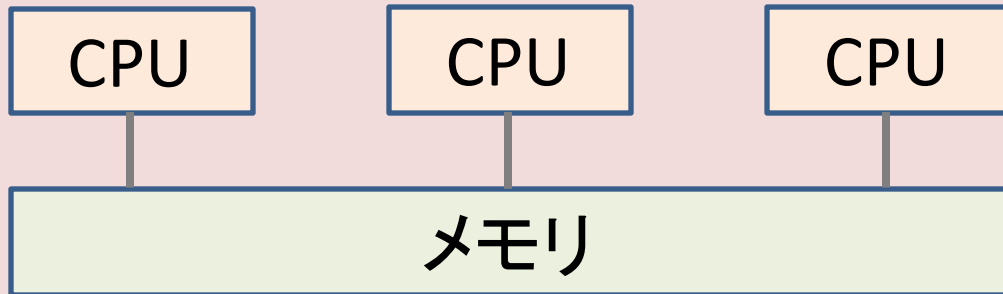
CPU並列計算の分類

共有メモリ型

(SMP, Symmetric MultiProcessor)

◆メモリを共有

- キャッシュとメモリの整合性が必要
- CPUの同時アクセス不可

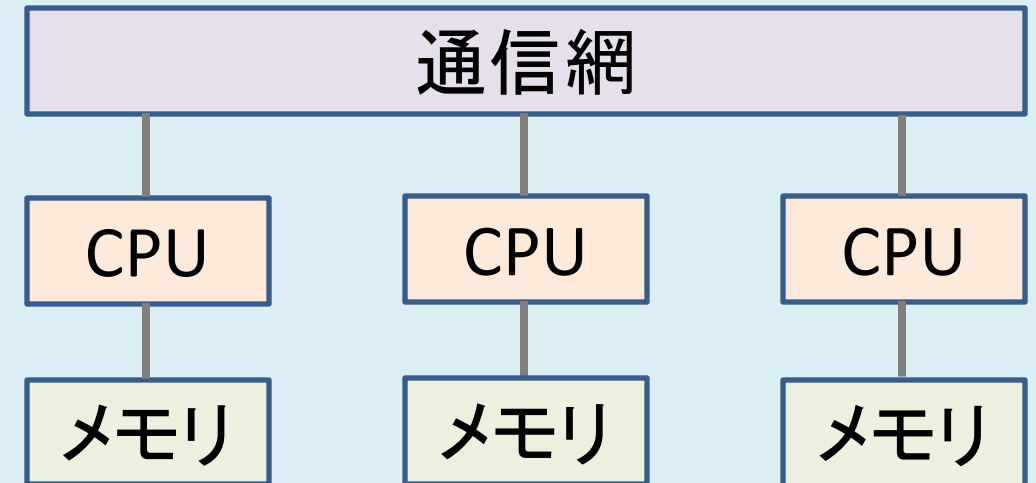


分散メモリ型 ← 現在の主流

(DSM, Distributed Shared Memory)

◆独立したメモリ

- 通信網でデータ交換
- 並列化に強い

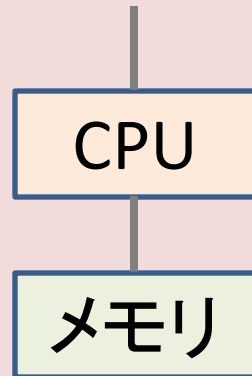


CPU並列計算の規格

OpenMP

スレッド並列化
(単一メモリ)

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main() {
5      #pragma omp parallel
6      {
7          printf("Hello, World from thread %d\n",
8              omp_get_thread_num());
9      }
10     return 0;
11 }
```

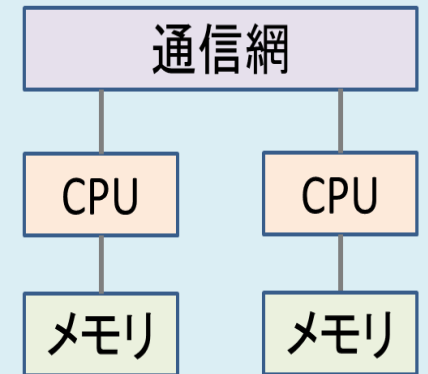


OpenMPは並列化する部分を囲む

MPI

メモリをまたいだ並列化
(メモリコピー含む)

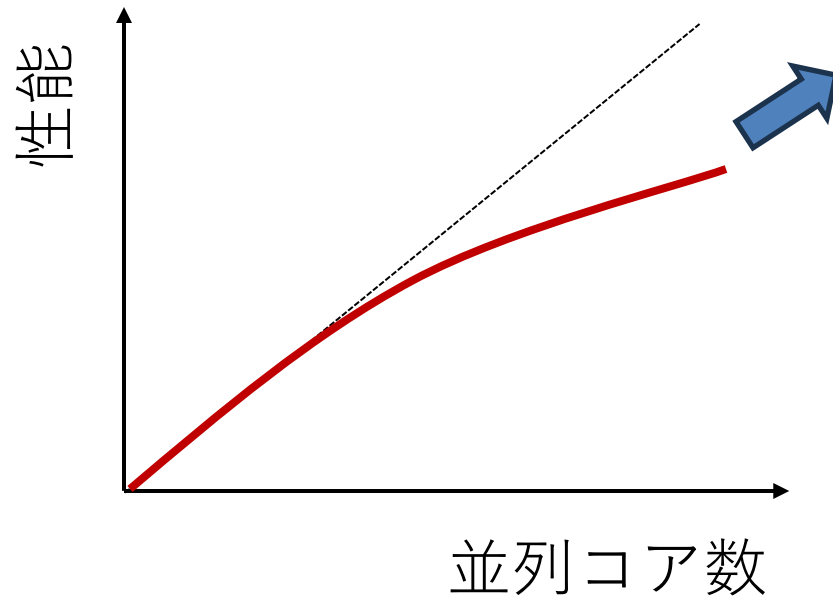
```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char** argv) {
5      MPI_Init(&argc, &argv);
6
7      int world_rank;
8      MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
9
10     int world_size;
11     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
12
13     printf("Hello, World from process %d out of %d\n",
14         world_rank, world_size);
15
16     MPI_Finalize();
17     return 0;
18 }
```



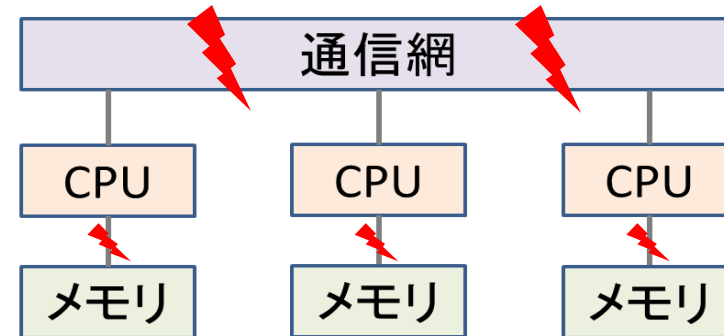
データ通信の命令が必要

CPU並列計算のボトルネック

- CPUコア数を上げるだけだと性能は線形に向上しない
- スケーラビリティ：並列数に対する性能向上の指標



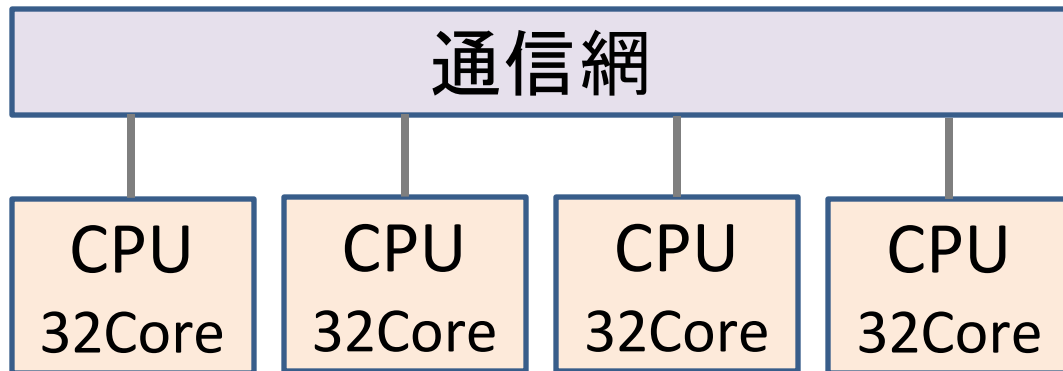
計算機能以外のメモリアクセス & 通信によってスケーラビリティが低下する



全てのノードのメモリ内容を揃えるために通信時間が発生する

CPU並列計算からGPU並列計算へ

- マルチコア：一般的には、コアが8~32個あるCPUを1つのノードとして並列化する



合計 $32 \times 4 = 128$ のコア数となるが、ノード間通信がボトルネック

- GPU計算：ビデオメモリによる超並列計算（別名：アクセラレータ）
 - 周波数は超遅いが、10万コアの並列計算も可能
 - メモリとコアが隣接している

ヘテロジニアス構成における計算プロセス

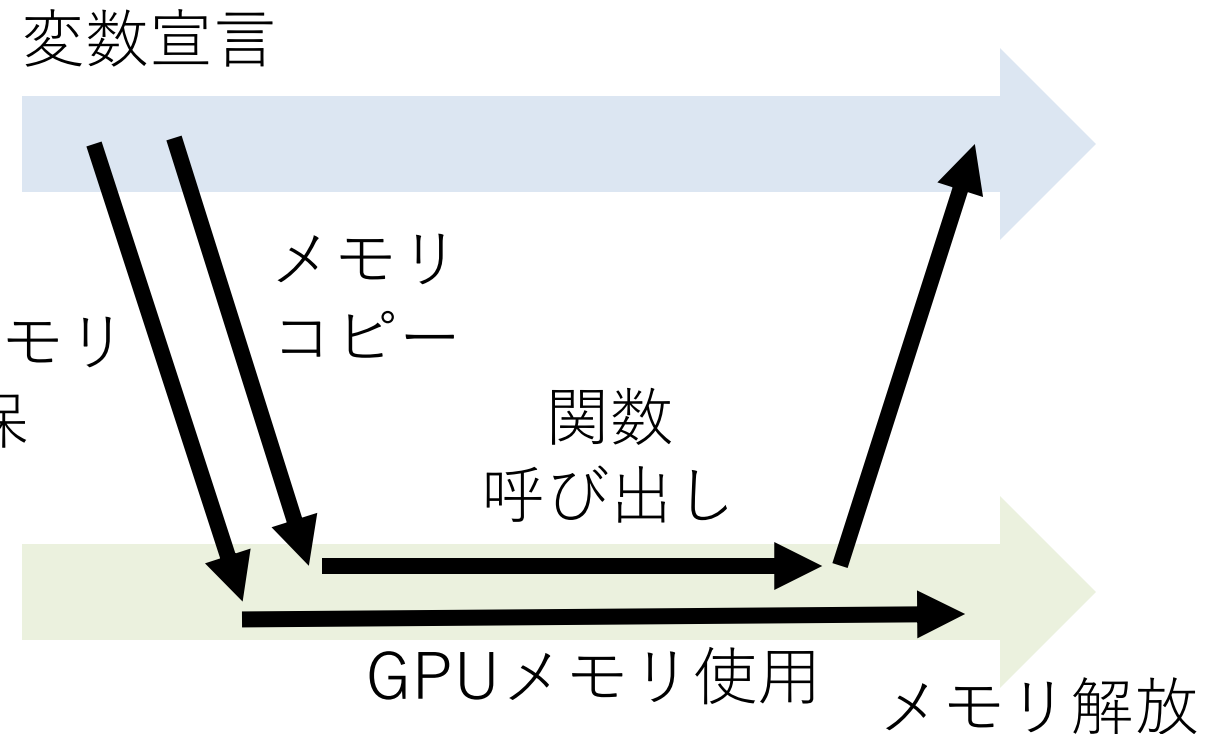
- GPU計算は関数化して使用される
- 並列化しにくい部分はCPUで処理するので、**CPUの性能は必須**



CPU
(&メインメモリ)



GPU
(&ビデオメモリ)



並列計算の高速化

- ムーア法則に従った汎用プロセッサの「計算」の性能向上が望めない中で、「通信」が性能向上の鍵となる
 - ネットワークを対象とすれば通信速度
 - 計算と通信の両者を含んだシステム性能として応答速度
- コスト削減のため、実用的観点から自動チューニング技術へ期待
 - 「性能」は計算機（対象マシン）の計算速度に限定しない
 - 「自動」は機械学習を意味するが、人間の介入を否定しない



並列計算の高速化方法

研究者人数

アプリ作者



MPIプログラムで生じる通信パターンと全体構成を正確に理解する必要がある

難！

アプリユーザー



システムベンダー



プロセスのナンバリングの更新など
ルーティングアルゴリズムやジョブ
マッピングなど

目標

ネットワークベンダー

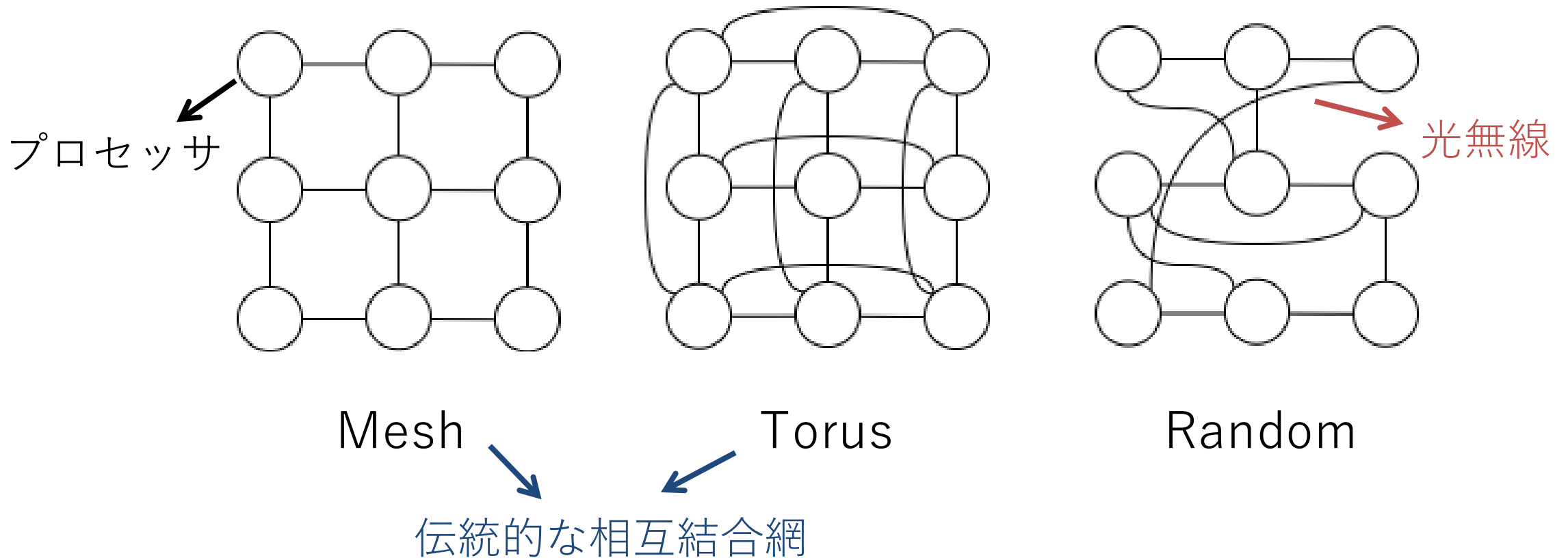


集団通信の一部をネットワークスイッチにオフロードするMPI実装（NVIDIAのSHArPやAllreduceなど）

難！

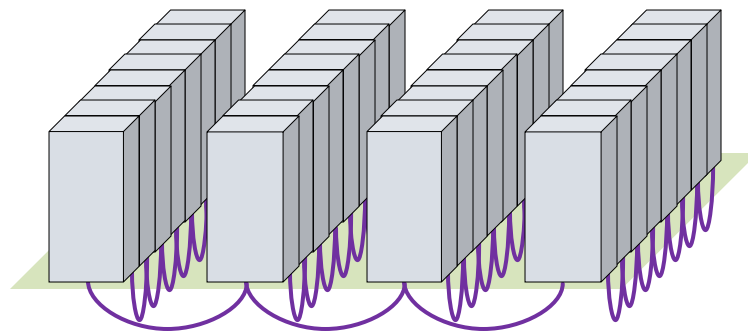
並列計算アプリ実行：ネットワークタイプ

- 並列コンピューターとアプリケーション実行のプロセストポロジの物理構造を定義する

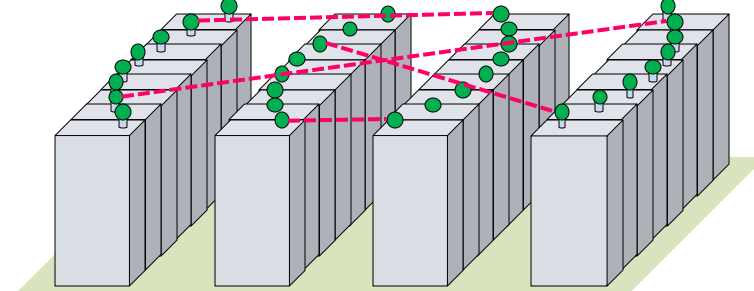


並列計算アプリ実行：トポロジ最適化

- 生成されたランダムトポロジに基づいて、設計パラメータを考慮して最適化する

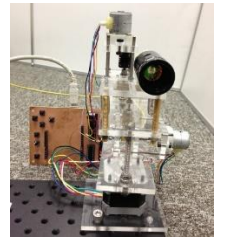


ケーブル



光無線

ランダムトポロジ構成



- 例
 - 最大の二分帯域幅（Bisection Bandwidth）を持つトポロジを作成する
 - 最小の平均最短経路長（ASPL）を持つトポロジを作成する

並列計算アプリ実行：プロセスマッピング

- マッピング問題はNP完全問題である
- ノードの番号付け（ナンバニング）に従う単純な解決策を取る
 - プロセッサ間通信を最小限に抑える

A_0

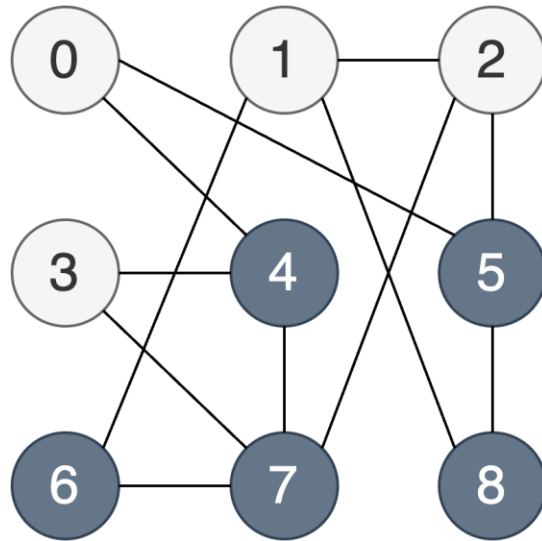
0	3	2	2
3	0	1	3
2	1	0	4
2	3	4	0

ASPL = 2.5

A_1

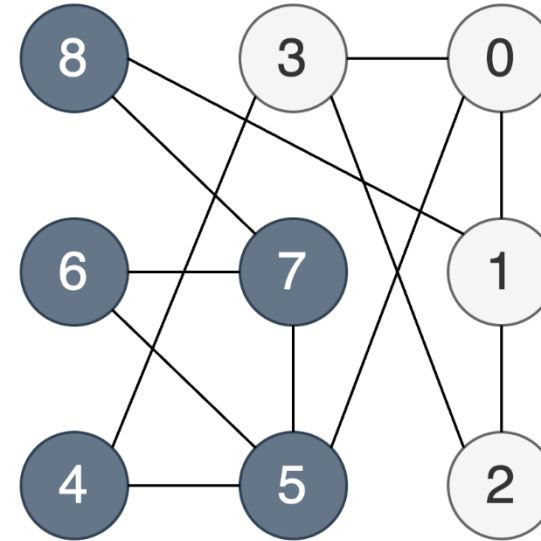
0	2	2	1	3
2	0	3	3	1
2	3	0	1	2
1	3	1	0	3
3	1	3	3	0

ASPL = 2.1



Task 0 
Task 1 

プロセス0とプロセス1との通信には3ホップが必要



A_0

0	1	2	1
1	0	1	2
2	1	0	1
1	2	1	0

ASPL = 1.3

A_1

0	1	2	2	3
1	0	1	1	2
2	1	0	1	2
2	1	1	0	1
3	2	2	1	0

ASPL = 1.6

プロセス0とプロセス1との通信には1ホップが必要

並列計算アプリ実行：MPI実装

- MPIプログラマーにとって、特定の状況で適切なMPI実装を選択することは困難である



OPEN MPI

MPICH

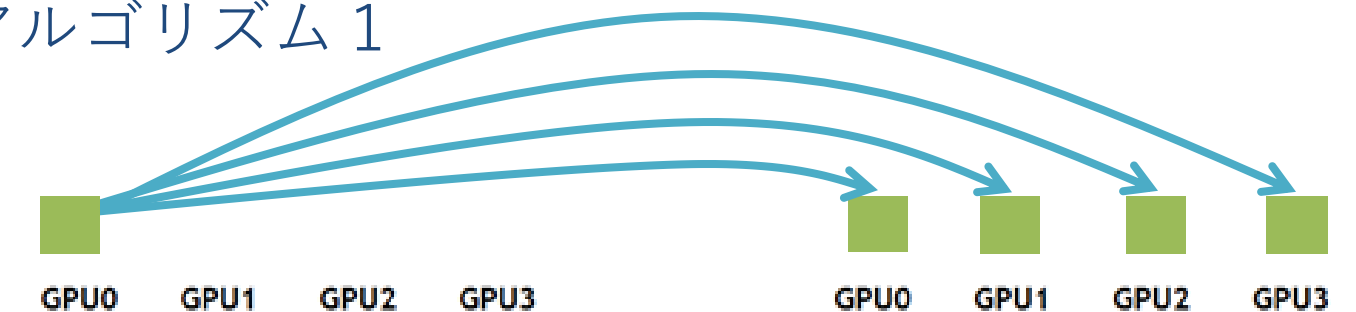


MVAPICH

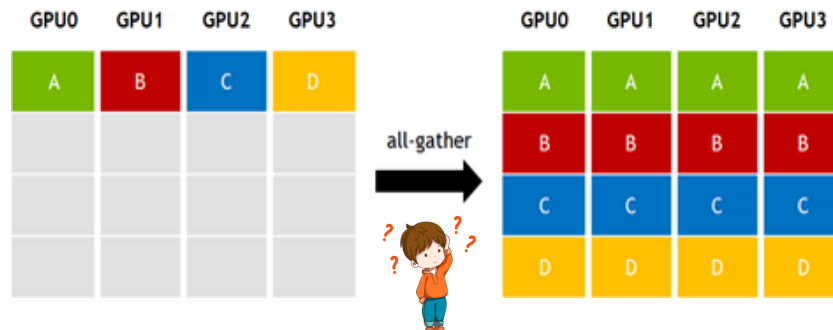
並列計算アプリ実行：集団通信アルゴリズム

- MPI実装の間で大きく異なる

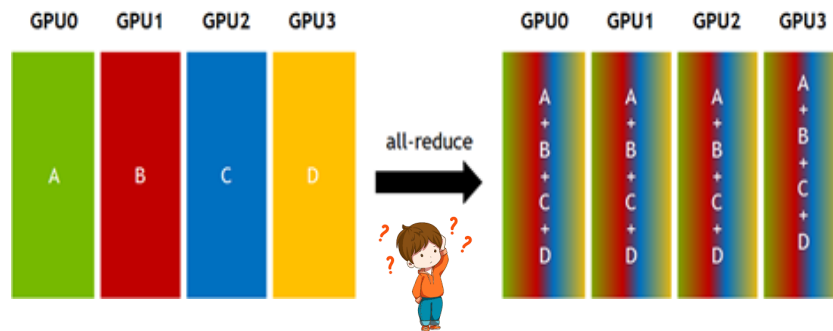
アルゴリズム 1



アルゴリズム 2



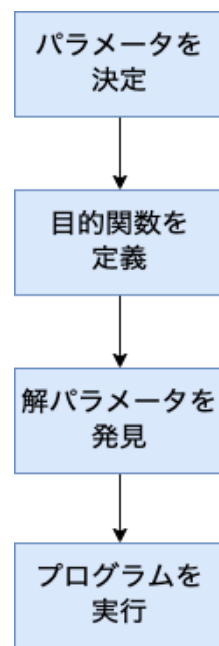
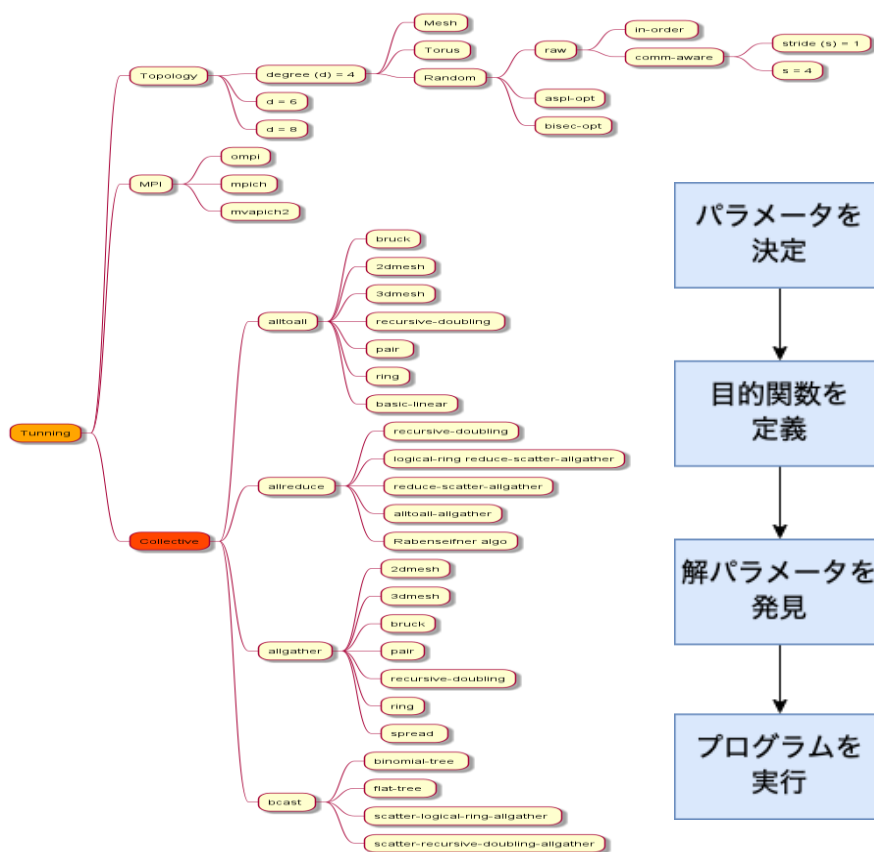
アルゴリズム？



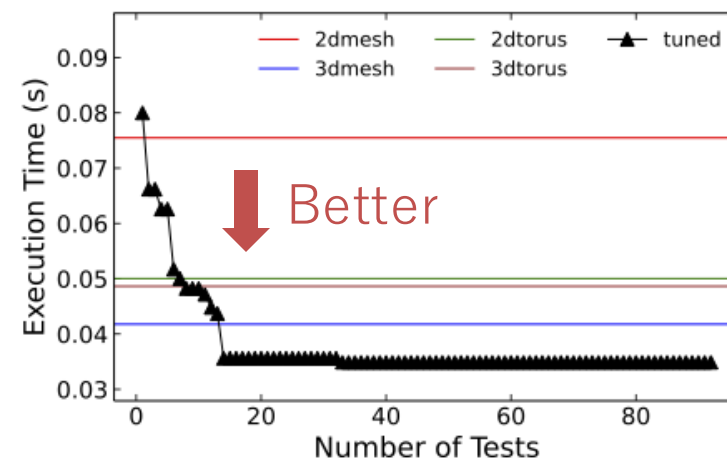
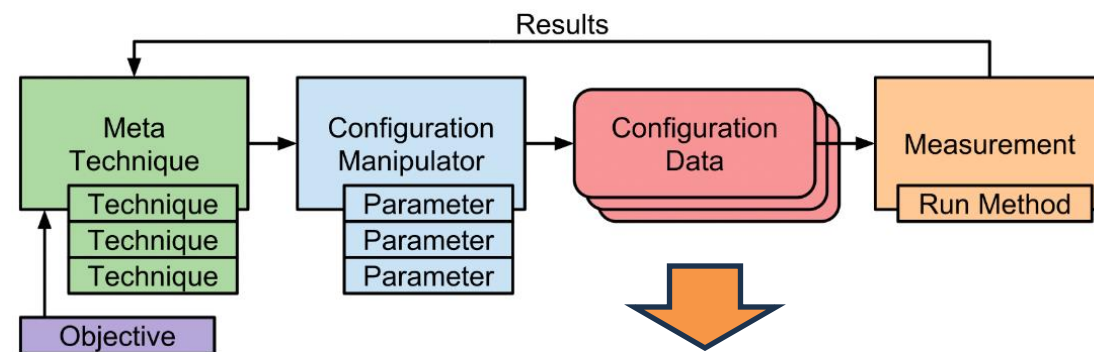
アルゴリズム？

並列計算の高速化のための自動チューニング

- 目的：プログラムを試行実行することで、**準最適なパラメータの組み合わせ**を発見する



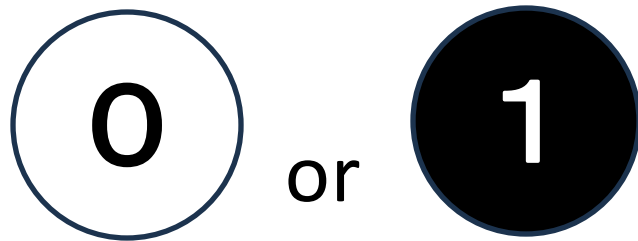
マルチグリッド
(MG)
並列アプリ



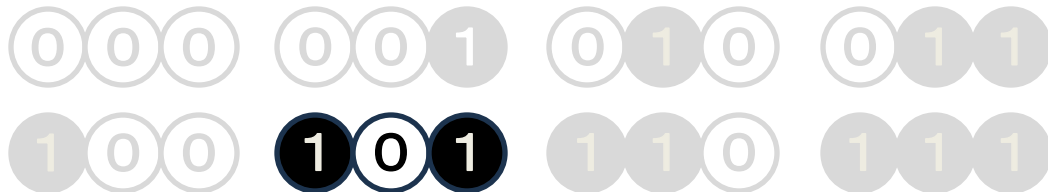
その他：量子計算

古典ビット（古典コンピュータ）

0 か 1 の**いずれか**の状態

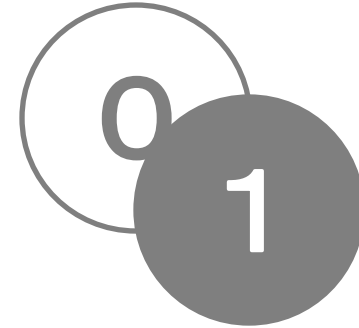


3 ビットで 8 つの状態の**いずれか**を表す

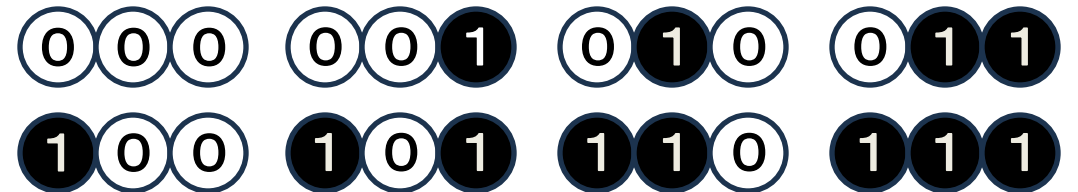


量子ビット（量子コンピュータ）

0 と 1 を重ね合わせて**同時に**表す



3 ビットで 8 つの状態を**同時に**を表す



量子コンピュータは重ね合わせ状態を上手に利用する

結論と今後の課題

- 先進的なアプリの実行に**並列コンピュータ**が必要不可欠
- 並列アプリの速度へ影響を与える要素が多数
 - コスト削減のため、実用的観点から**自動チューニング技術**へ期待
- 将来、スパコンのアクセラレータとして**量子コンピュータ**が登場！

