柏キャンパス一般公開2025

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
2025年10月25日

アウトライン

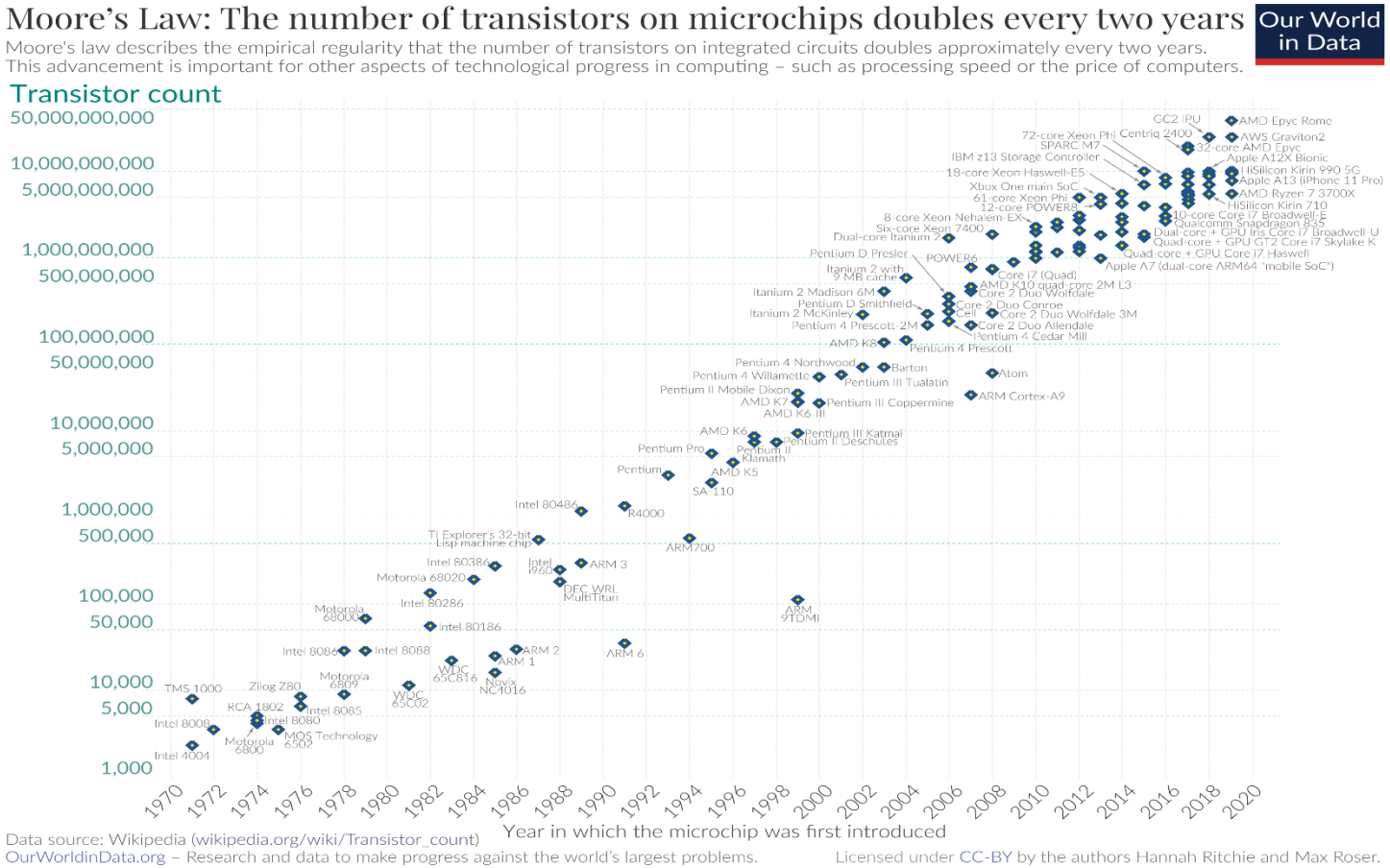
- 背景：計算機環境の変化
- 並列計算の基礎
- 並列計算の高速化
- 結論と今後の課題

計算能力：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



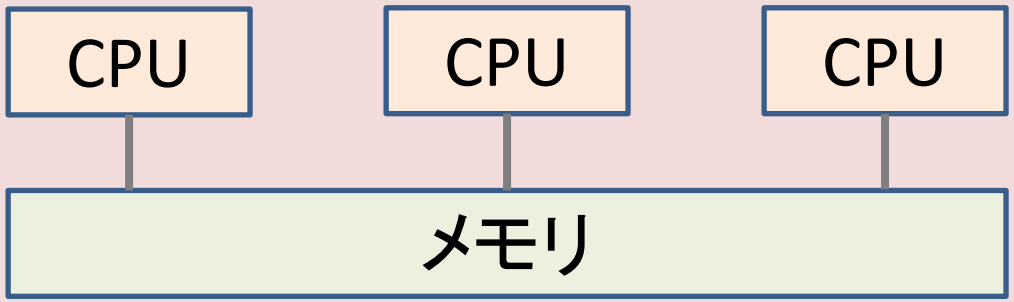
CPU並列計算の分類

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

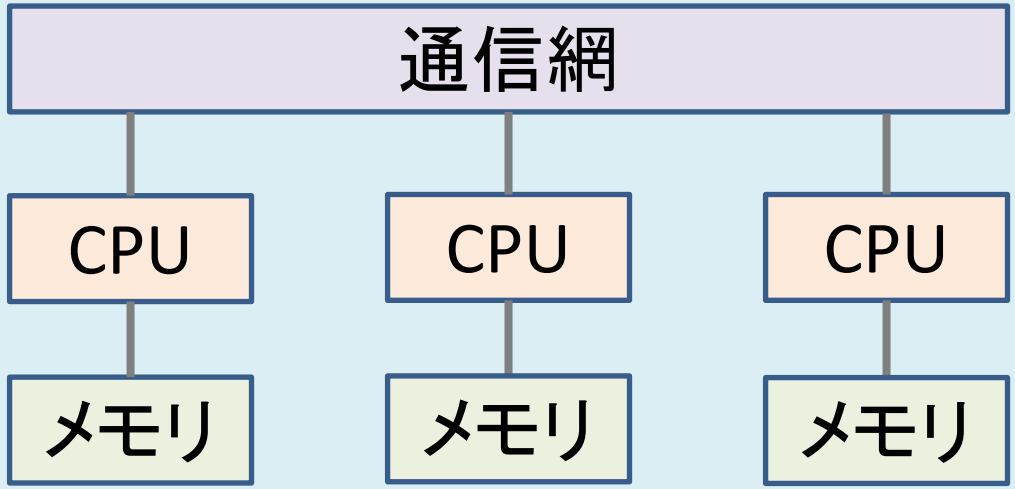


分散メモリ型 ← **現在の主流**

（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

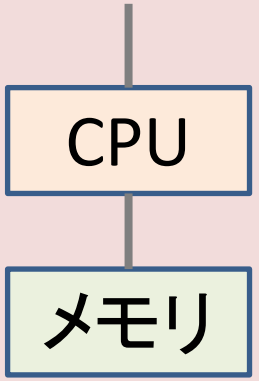


CPU並列計算の規格

OpenMP

スレッド並列化
（単一メモリ）

```c
#include <stdio.h>
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        printf("Hello, World from thread %d\n",
            omp_get_thread_num());
    }
    return 0;
}
```

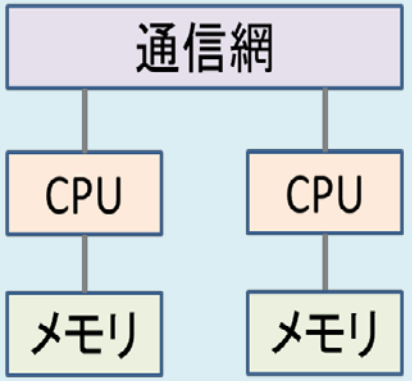


OpenMPは並列化する部分を囲む

MPI

メモリをまたいだ並列化
（メモリコピー含む）

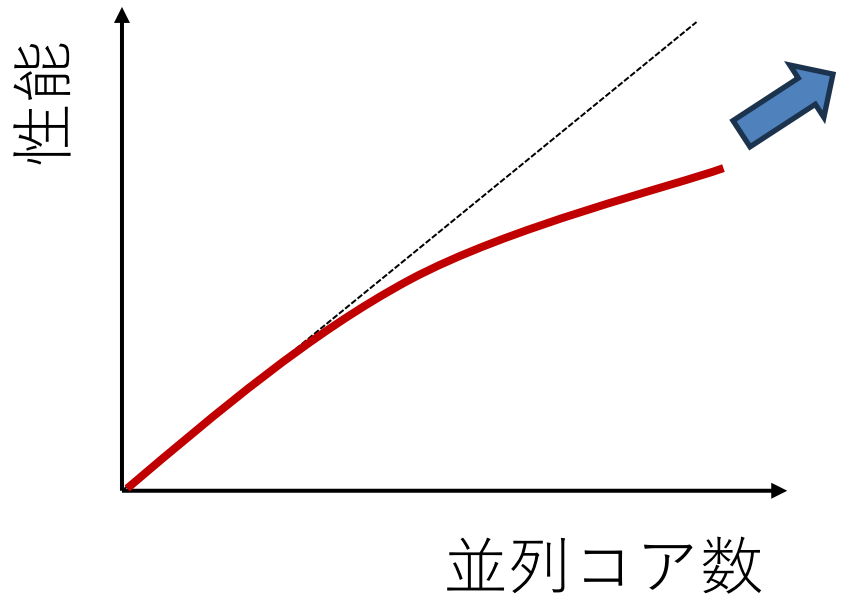
```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    printf("Hello, World from process %d out of %d\n",
        world_rank, world_size);

    MPI_Finalize();
    return 0;
}
```



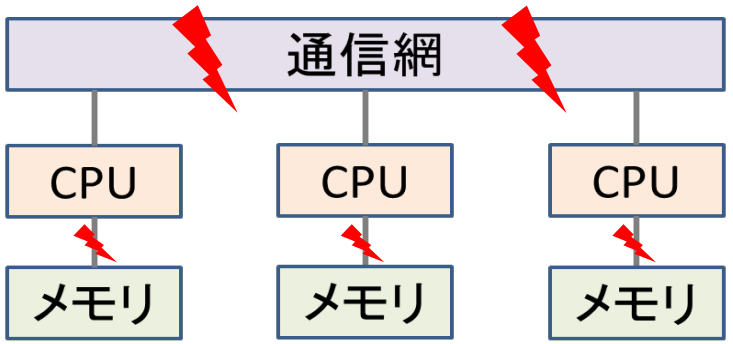
データ通信の命令が必要

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



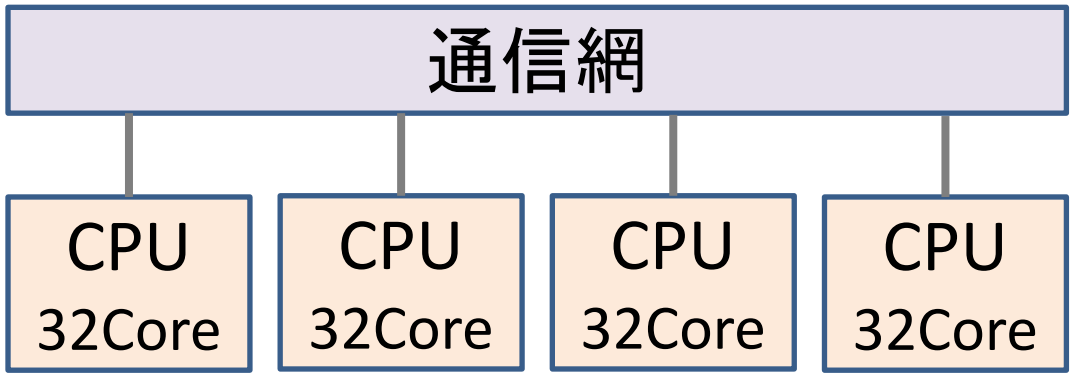
計算機能以外の**メモリアクセス＆通信**によってスケーラビリティが低下する



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32×4=128のコア数となるが、ノード間通信がボトルネック

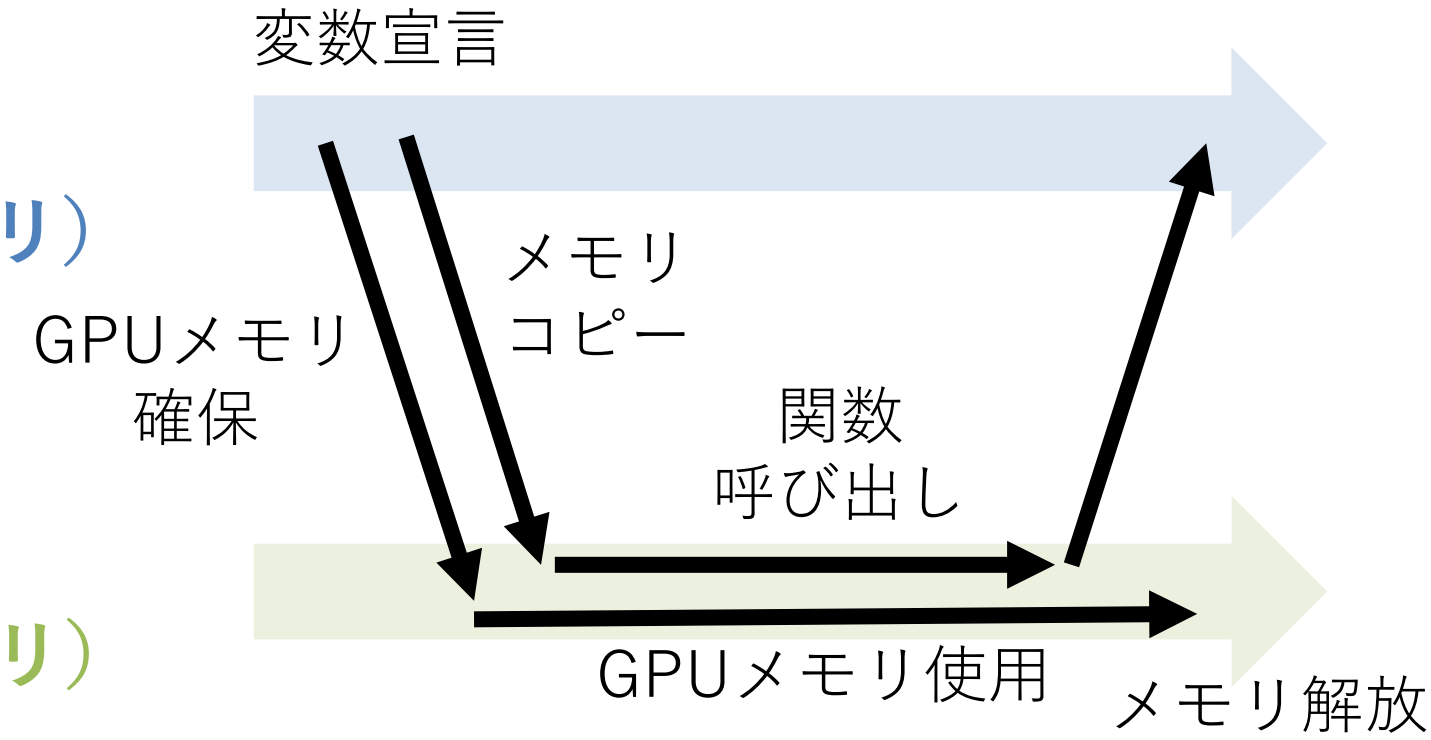
- GPU計算：ビデオメモリによる超並列計算（別名：アクセラレータ）
  - 周波数は超遅いが、10万コアの並列計算も可能
  - メモリとコアが隣接している

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）

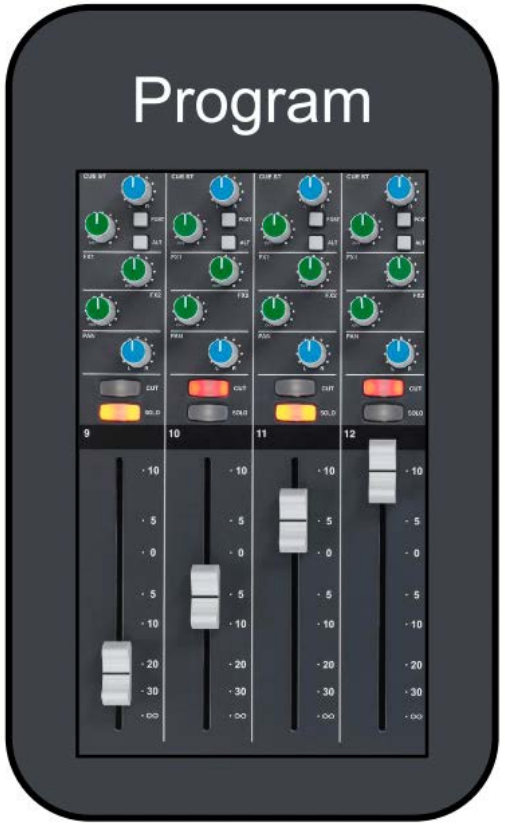
GPU
（＆ビデオメモリ）



並列計算の高速化

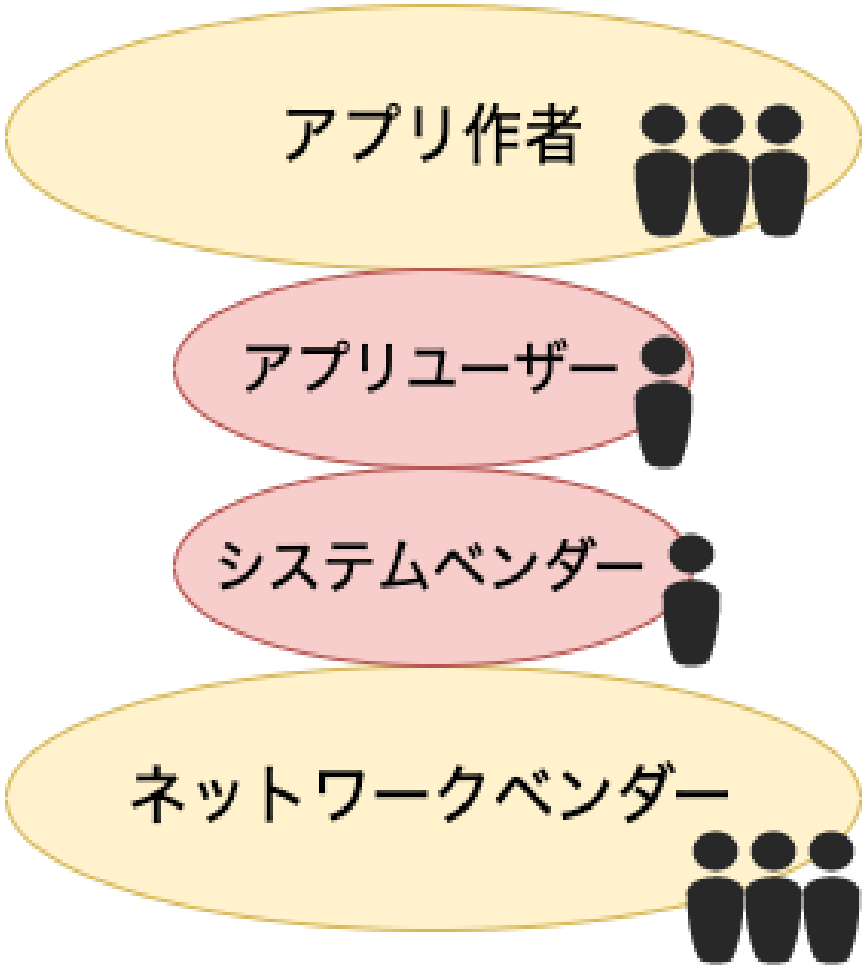
- **ムーア法則に従った汎用プロセッサの「計算」の性能向上が望めない中で、「通信」が性能向上の鍵となる**
  - ネットワークを対象とすれば通信速度
  - 計算と通信の両者を含んだシステム性能として応答速度

- **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、人間の介在を否定しない



並列計算の高速化方法

研究者人数



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！

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

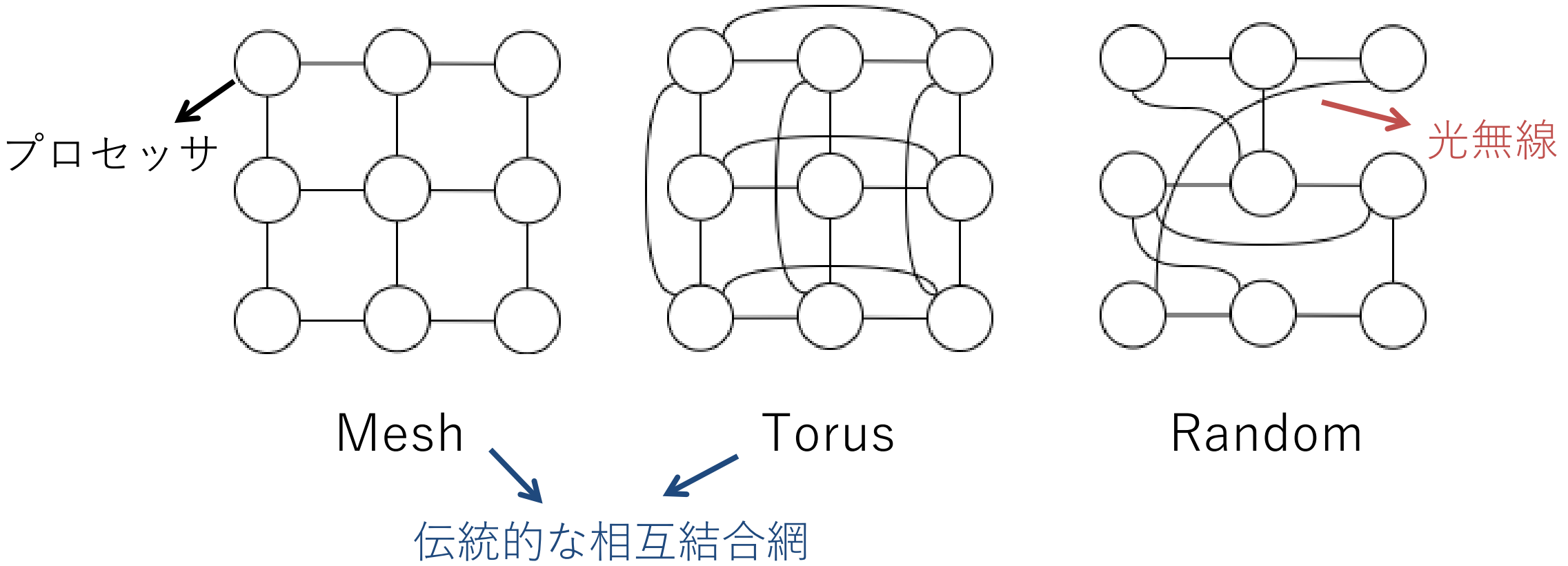
目標

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！

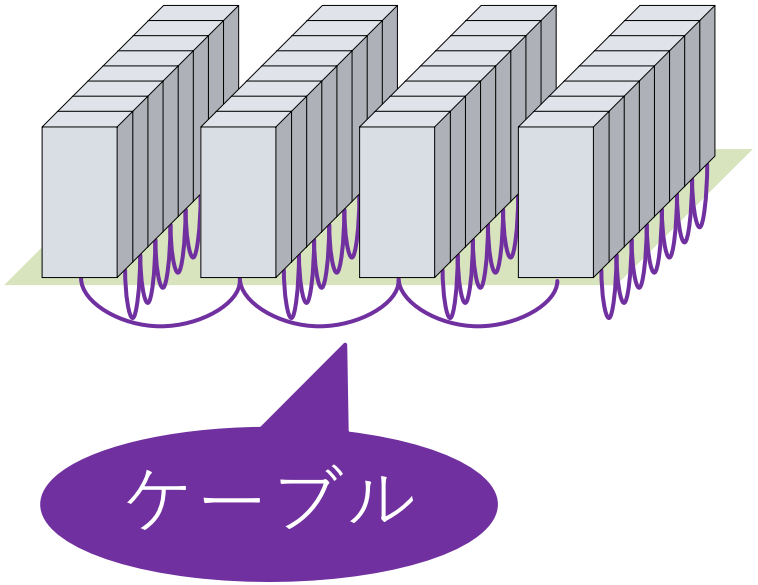
並列計算アプリ実行：ネットワークタイプ

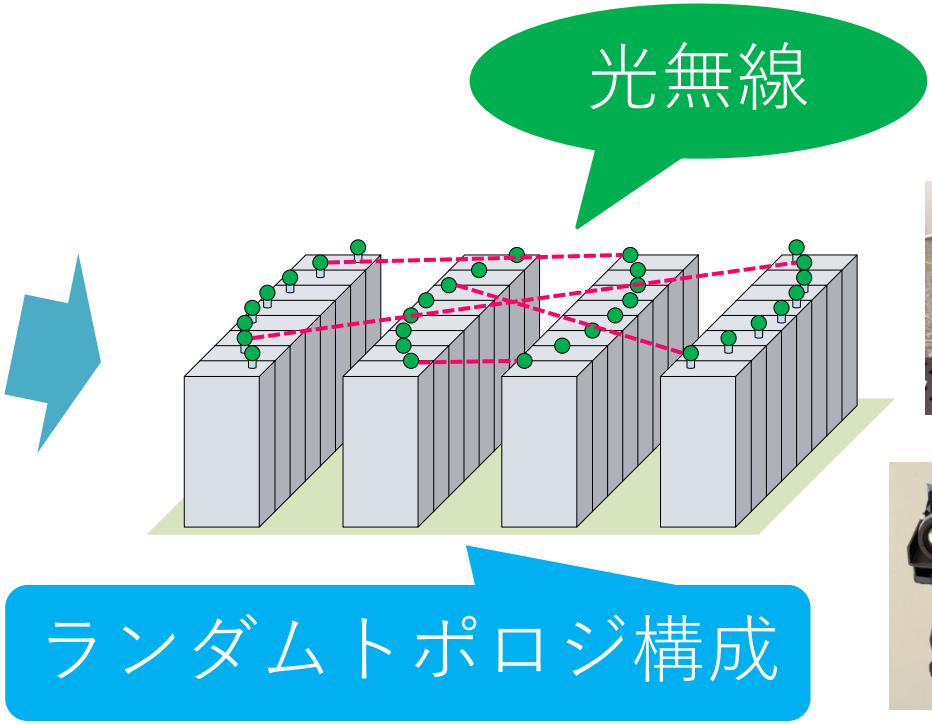
- 並列コンピューターとアプリケーション実行のプロセストポロジの物理構造を定義する

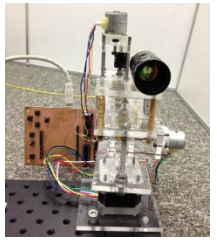


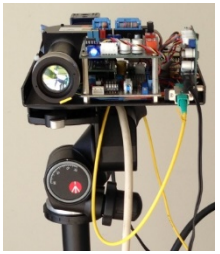
並列計算アプリ実行：トポロジ最適化

- 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





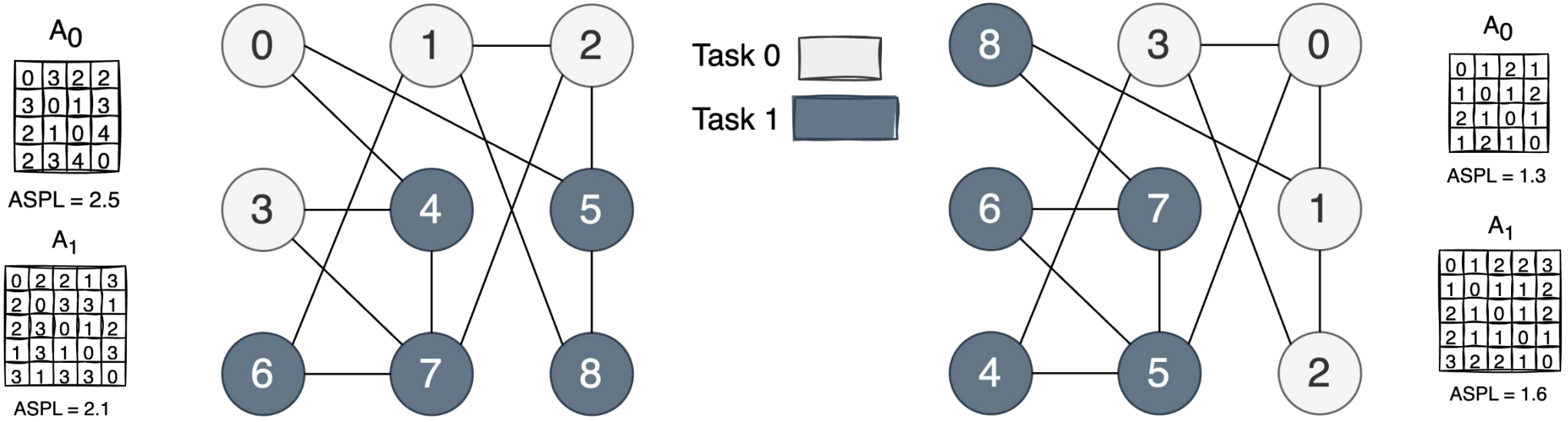




- 例
  - 最大の二分帯域幅（Bisection Bandwidth）を持つトポロジを作成する
  - 最小の平均最短経路長（ASPL）を持つトポロジを作成する

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



プロセス0とプロセス1と
の通信には３ホップが必要

プロセス0とプロセス1と
の通信には1ホップが必要

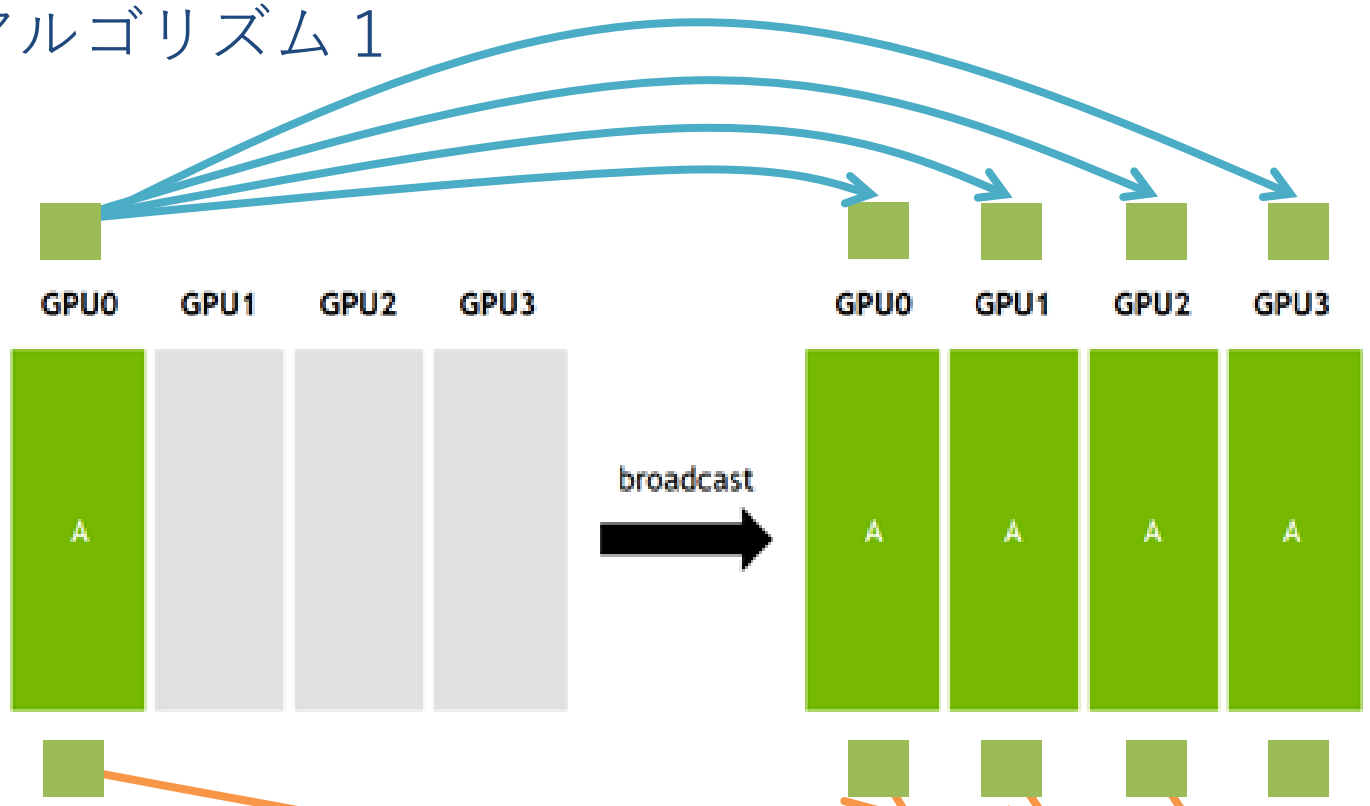
並列計算アプリ実行：MPI実装

- 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

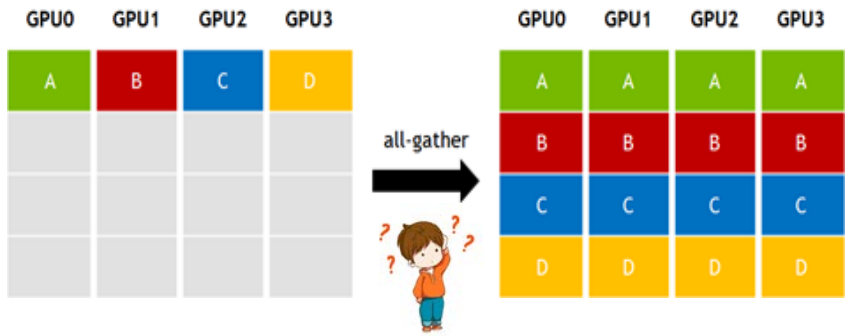
並列計算アプリ実行：集団通信アルゴリズム

- MPI実装の間で大きく異なる

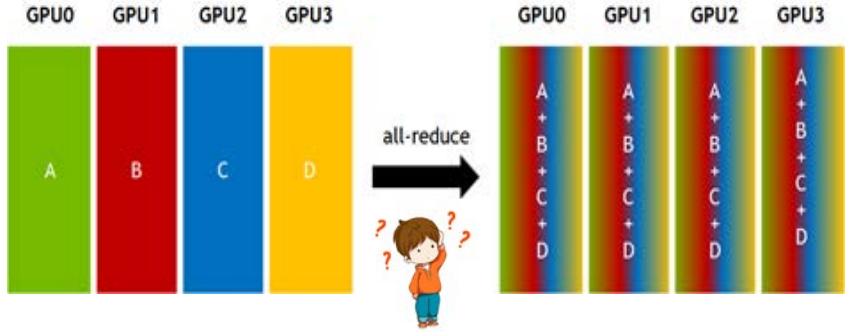
アルゴリズム１



アルゴリズム２



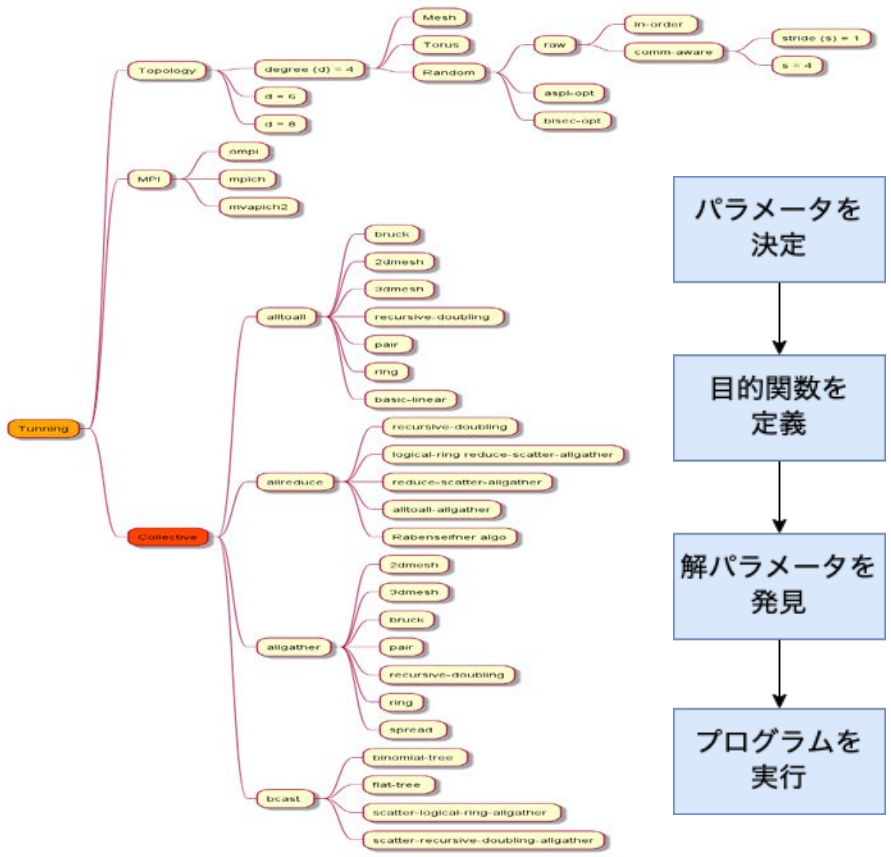
アルゴリズム？



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

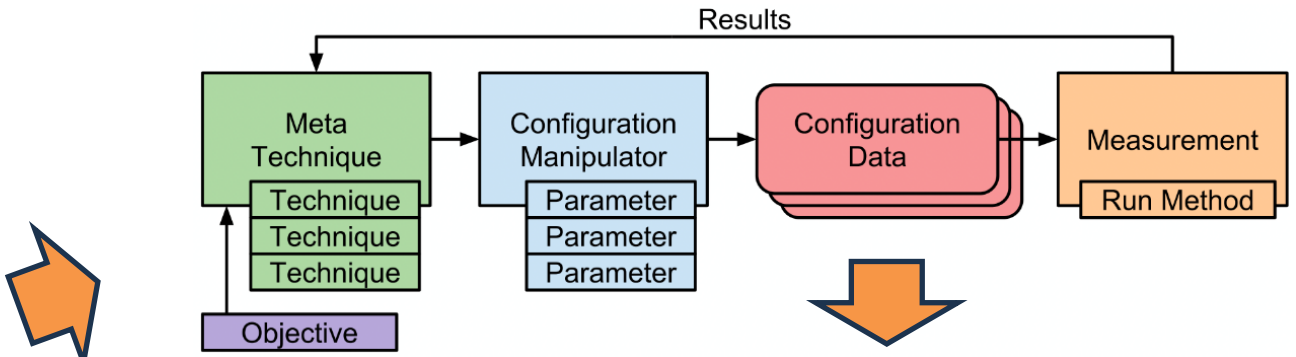


パラメータを決定

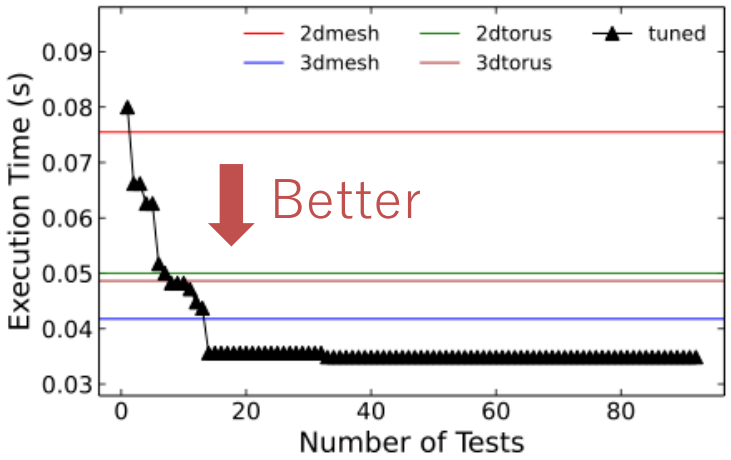
目的関数を定義

解パラメータを発見

プログラムを実行



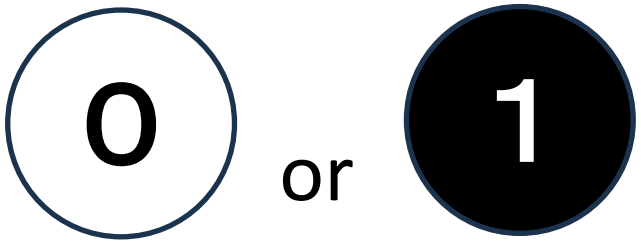
マルチグリッド（MG）並列アプリ



その他：量子計算

古典ビット（古典コンピュータ）

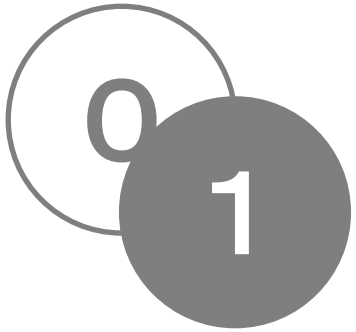
0 か 1 の**いずれか**の状態



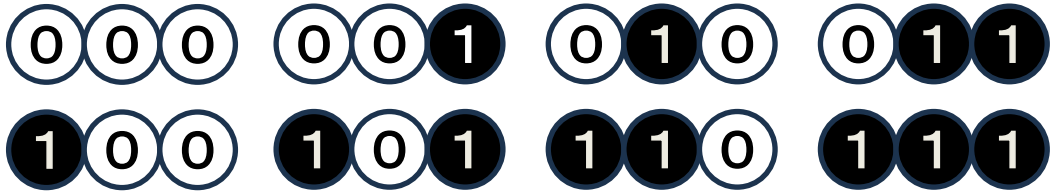
3 ビットで 8 つの状態の**いずれか**を表す

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！