計算科学演習I　第5回講義
「並列計算とは」

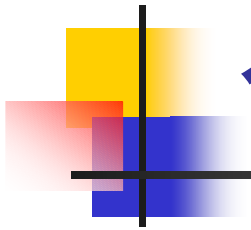
2012年5月17日

山本有作

ブラウザを立ち上げて

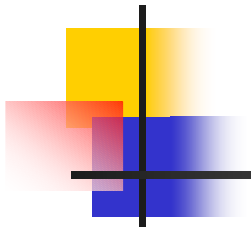
http://exp.cs.kobe-u.ac.jp/wiki/comp_practice/

の「3. 並列計算とは」の項目を見ること



今回の講義の概要

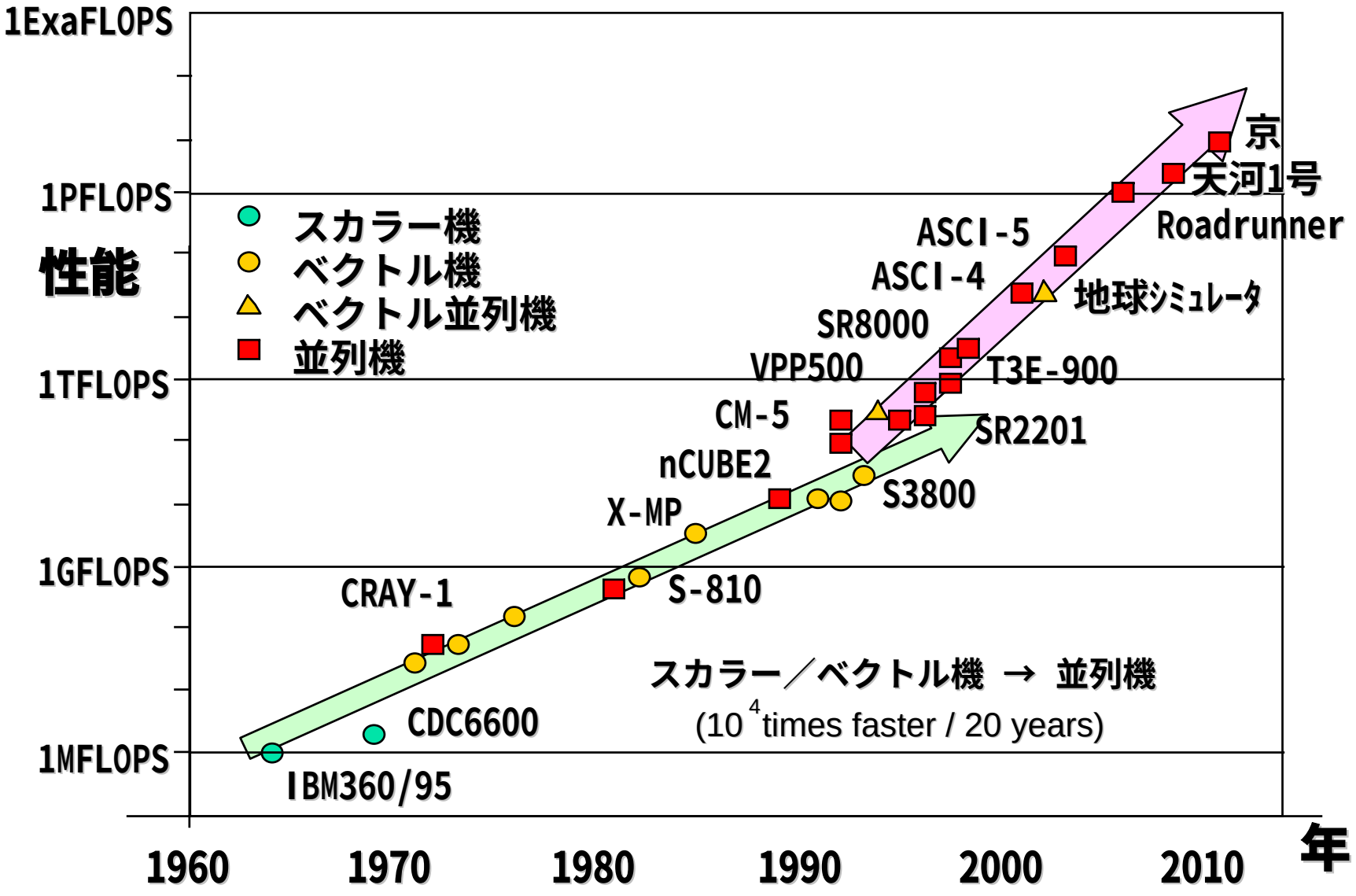
1. 並列計算機の種類と特徴
2. 基本的な並列化方法
3. 数値計算アルゴリズムとその並列化
4. 並列性能の向上のために

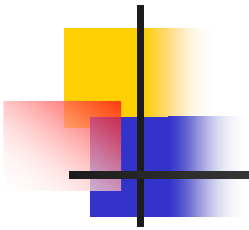


1. 並列計算機の種類と特徴

- さまざまな並列計算機
- 並列計算機のアーキテクチャ
  - SIMD と MIMD
  - 共有メモリ型と分散メモリ型
  - ホモジニアス型とヘテロジニアス型
- 本演習で用いる並列計算機

スーパーコンピュータの性能動向





並列計算機の登場

- 並列計算機の普及の背景
  - プロセッサの動作周波数向上の飽和
  - 専用スーパーコンピュータの設計コストの増加
- 並列計算機の特長
  - プロセッサ数を増やすことでピーク性能を無制限に向上可能
  - 分散メモリ型並列機では，プロセッサ数に比例した大きなメモリ空間が利用可能
  - 汎用のプロセッサを使うことで設計コストを大幅に削減可能
- 並列計算機の課題
  - 多数のプロセッサを効率良く使うには，良い並列化アルゴリズムが必要
  - OpenMP，MPI などを使ったプログラムの修正/書き換えが必要

さまざまな並列計算機

- 研究室レベルのPCクラスタからスーパーコンピュータに至るまで，様々な並列計算機が科学技術計算に活用されている

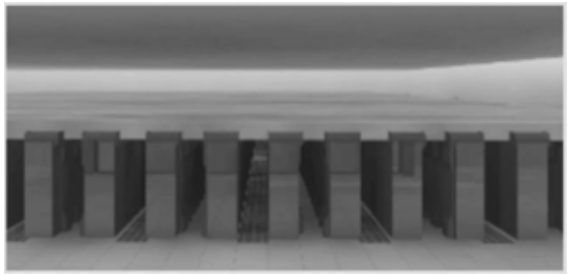
現在のスパコン

地球シミュレータ

天河1号 (中国)

PCクラスタ

次世代スパコン「京」



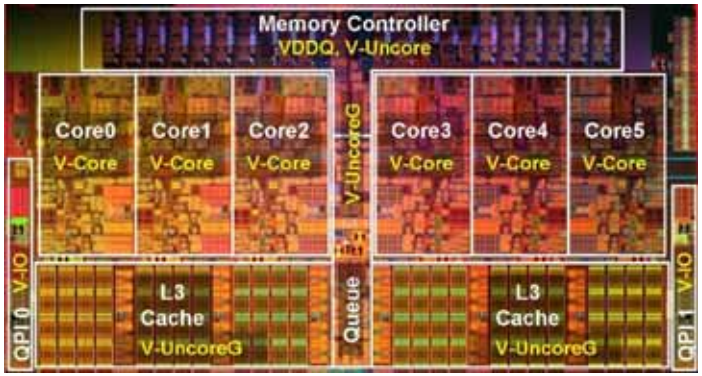
完成予想図 (今年6月完成)

建屋 (ポートアイランド)

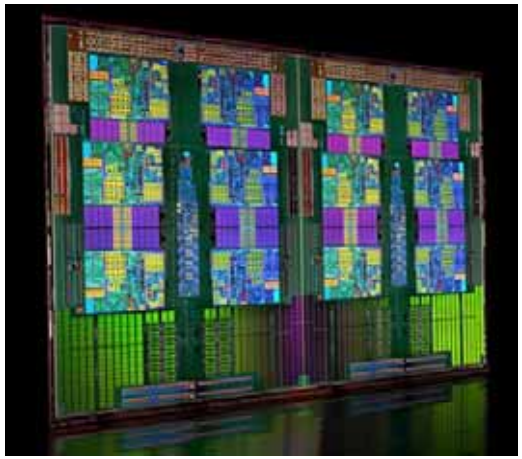
チップレベルでの並列処理

- マルチコアプロセッサの普及
  - チップ上に2〜12個のプロセッサコアを載せたマルチコアプロセッサが一般化
  - チップレベルでの共有メモリ型並列計算機

⇒ パソコンレベルの計算機でも，並列計算のメリットを得られる時代に



Intel Core i7-980X **プロセッサ**
6**コア / チップ**



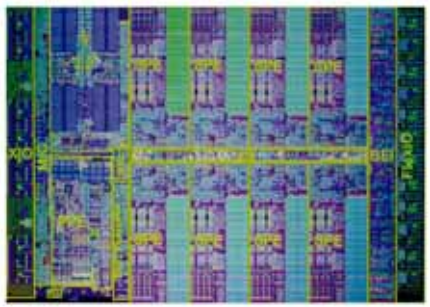
AMD Opteron 6100 **プロセッサ**
12**コア / チップ**

チップレベルでの並列処理 (続き)

- **ゲーム機用プロセッサ/グラフィックスプロセッサの高性能化**
  - PlayStation3**用プロセッサ** (Cell)： 9**個の**CPU**を集積**
  - GTX480 **グラフィックプロセッサ** (GPU)：480**個の演算コアを集積**

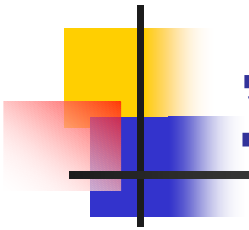
⇒ **・これらを数値計算に活用できれば，超高速の計算が可能**
・超大規模シミュレーションのための新しい手段

PlayStation3



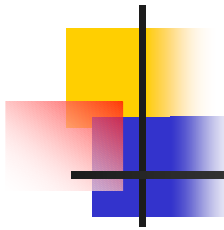
Cell **ﾌﾟﾛｾｯｻ写真**
(9**個の**CPU**を内蔵**)

グラフィックスプロセッサ
GeForce GTX480



並列計算機のアーキテクチャ

- SIMD と MIMD
  - プロセッサ群の命令実行方法に着目した分類法
- 共有メモリ型と分散メモリ型
  - メモリ空間に着目した分類法
- ホモジニアス型とヘテロジニアス型
  - ハードウェアの均質性に着目した分類法

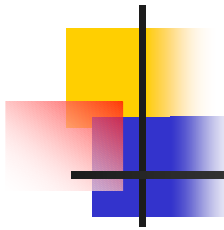


SIMD と MIMD

- SIMD (Single Instruction Multiple Data) **型並列計算機**
  - **全プロセッサがそれぞれ異なるデータに対し，同じ命令を実行**
    - **例：ベクトルの加算 $c = a + b$ において，各プロセッサがそれぞれの要素の加算を実行**

- SIMD **の特徴**
  - **命令実行の制御回路は1組で済むため，プロセッサ数増加が容易**
  - **命令実行に柔軟性がないため，有効な用途が限られる**

- SIMD **の例**
  - **初期 (～1990年頃) の商用並列計算機**
  - **グラフィックスプロセッサ (各画素に対して同じ演算を実行)**
  - **ベクトル型スーパーコンピュータ**

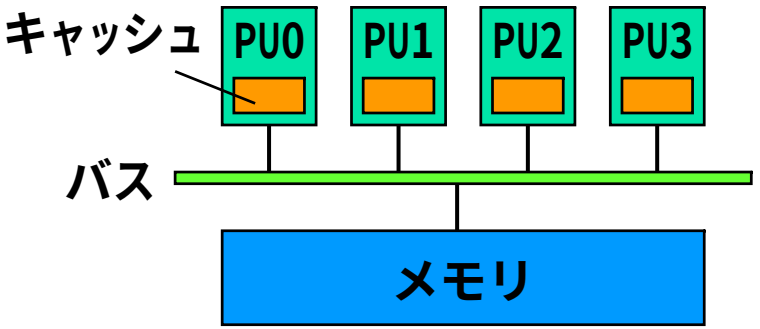


SIMD と MIMD (続き)

- MIMD (Multiple Instruction Multiple Data) **型並列計算機**
  - **各プロセッサがそれぞれ異なるデータに対し，異なる命令を実行**
    - **例：マルチコアのPCで，1つのコアで文書を作成しながら，もう1個のコアでウイルスチェックを行う**
- MIMD **の特徴**
  - **各プロセッサに命令実行の制御回路が必要**
  - **命令実行の柔軟性は非常に高い**
- MIMD **の例**
  - **最近のほぼすべての商用並列計算機**
  - **マルチコアプロセッサ**

共有メモリ型と分散メモリ型

- **共有メモリ型並列計算機**
  - **複数のプロセッサ (PU) がバスを通してメモリを共有**
  - **どのPUも同じメモリ領域にアクセスできる**

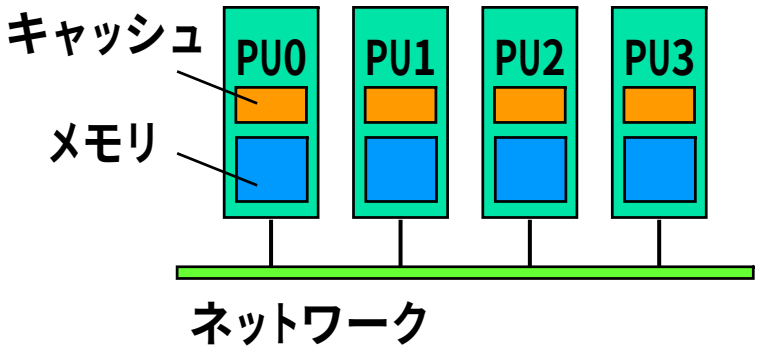


- **特徴**
  - **メモリ空間が単一のためプログラミングが容易**
  - **PUの数が多すぎると，アクセス競合により性能が低下**
    → 2〜16台程度の並列が多い

- **プログラミング言語**
  - OpenMP (FORTRAN/C/C++ **＋ 指示文**) **を使用**
    - **メモリ領域を分割し，MPI (次頁参照) を利用することも可能**

共有メモリ型と分散メモリ型 (続き)

- 分散メモリ型並列計算機
  - 各々がメモリを持つ複数のPUを ネットワークで接続
  - 各PUはそれぞれ自分の持つメモリ のみにアクセス可能

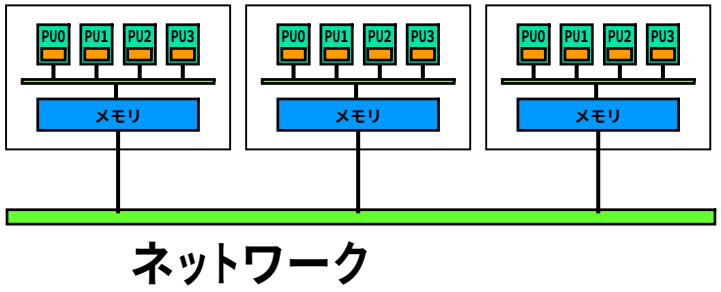


- 特徴
  - 数千～数万PU規模の並列が可能
  - PU間へのデータ分散を意識したプログラミングが必要

- プログラミング言語
  - FORTRAN/C/C++ ＋ MPI を使用

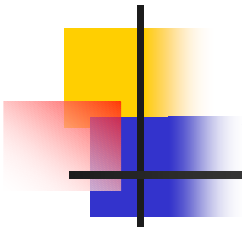
共有メモリ型と分散メモリ型 (続き)

- SMPクラスタ
  - 複数の共有メモリ型並列計算機 (SMP) をネットワークで接続



- 特徴
  - 各ノードの性能を高くできるため，比較的少ないノード数で高性能を達成できる
  - プログラミングは，ノード内部の計算では共有メモリ型並列機として，ノードをまたがる計算では分散メモリ型並列機として行う
  - 最近のハイエンドスパコンはほとんどがこのタイプ

- プログラミング
  - MPI と OpenMP とを組み合わせて使用
    - MPI のみでプログラミングすることも可能

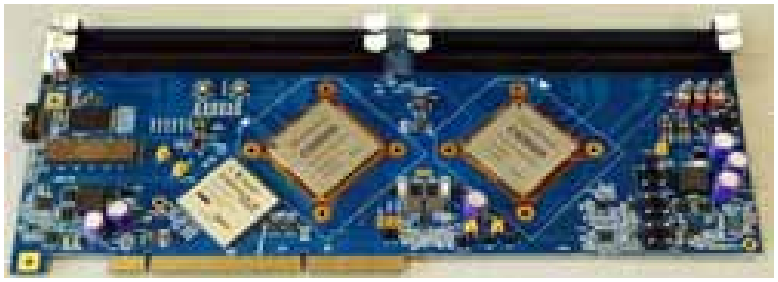


ホモジニアス型とヘテロジニアス型

- ホモジニアス型の並列計算機
  - 全プロセッサが同じ能力を持つ
  - 各プロセッサの持つメモリも同じ (分散メモリ型の場合)
- ホモジニアス型の特徴
  - 均質な仕事を並列化するのに向く
- ホモジニアス型の例
  - 今までの説明は，すべて暗黙のうちにホモジニアス型を仮定
  - マルチコアプロセッサ，PCクラスタ，SMPクラスタなど

ホモジニアス型とヘテロジニアス型

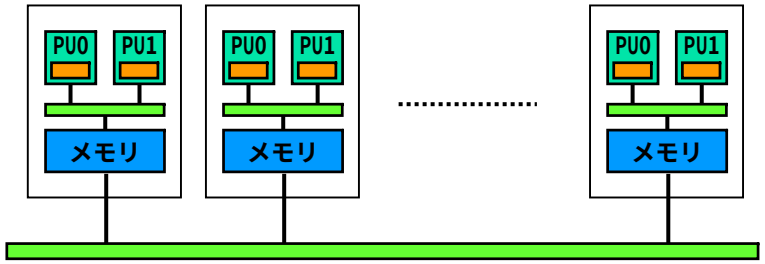
- ヘテロジニアス型の並列計算機
  - アーキテクチャ，能力，メモリ量などの異なるプロセッサが混在
- ヘテロジニアス型の特徴
  - 特定の計算に特化したプロセッサを用いて，その計算を加速可能
- ヘテロジニアス型の例
  - 数値計算用アクセラレータ
  - グラフィックス専用プロセッサ (GPU)
  - スペックの異なるPCをつなげて作ったPCクラスタ



数値計算用アクセラレータ
ClearSpeed 社 CSX600
(96コア，48GFLOPS)

本セミナーで用いる並列計算機

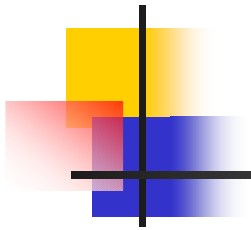
- NEC　Express 5800/i120Rg-1
  - 6.4GFLOPS／**プロセッサ**
  - 2**プロセッサ／ノード**
  - **メモリ**2GB ／**ノード**
  - **全**72**ノード**



ネットワーク

Express 5800/i120Rg-1
(1**ノード**)

- **本セミナーでの利用法**
  - **各ノードを使い,** OpenMP **の練習**
  - **複数ノードを使い,** MPI **の練習**
    - **この場合, ノード内も含めて** MPI **で並列化**



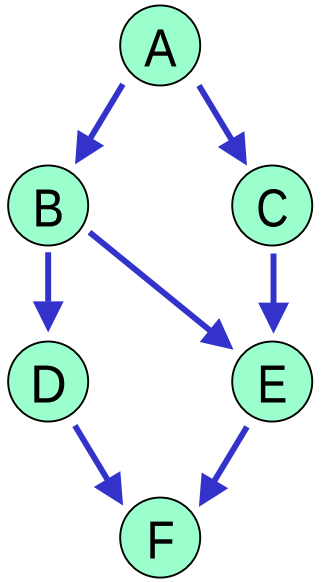
2. 基本的な並列化方法

- 制御フローグラフ
- ガントチャート
- データ並列化
- タスク並列化
- パイプライン型の並列化
- 並列化できない演算

制御フローグラフ

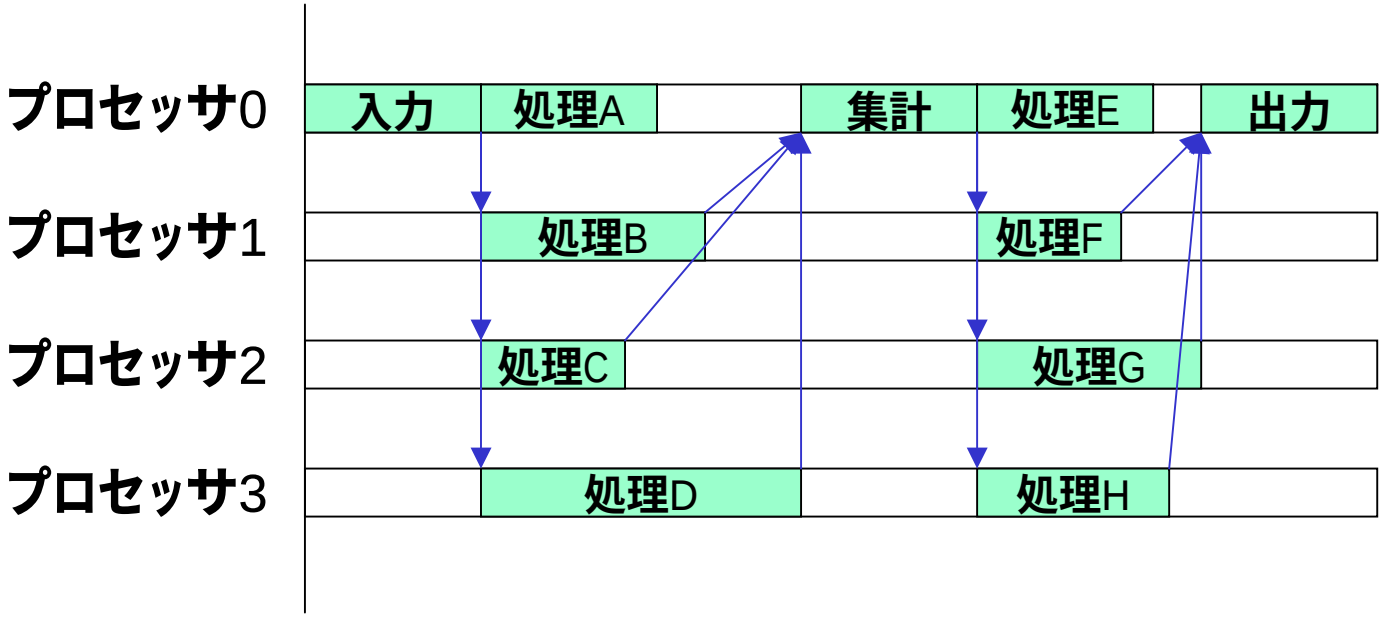
- 処理間の依存関係
  - 2つの処理A，Bがあるとする。Aを終えてからでないとBが開始できないとき，処理Bは処理Aの実行に依存するという。
    - 例：2個の数a，bの平均を求める場合，和を求める処理と2で割る処理
    - ここでいう「処理」は，サブルーチンのように大きい単位でも，四則演算のように小さい単位でもよい

- 制御フローグラフ
  - 処理A，B，C，… を丸で表し，BがAに依存するとき，AからBに矢印を描いて作ったグラフ
  - 依存関係にある処理は同時に（並列に）実行できないため，処理間の並列性を知るのに便利



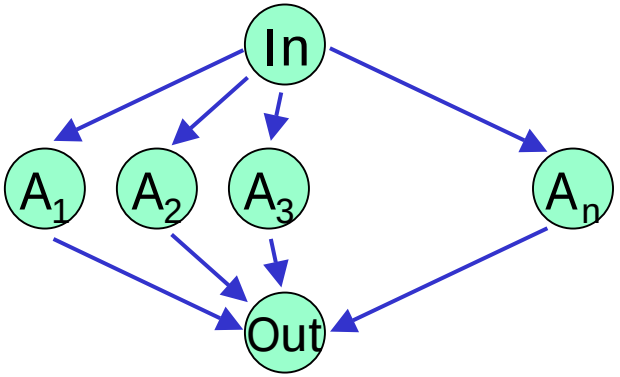
ガントチャート

- 定義
  - 時間を横軸，プロセッサ番号を縦軸に取り，各時間帯で各プロセッサが行う処理の内容を示した図をガントチャートと呼ぶ
    - 並列処理に限らず，一般の工程管理でも用いられる
  - 処理間の依存関係を矢印で書き入れることもある
  - プロセッサの利用効率などを解析するのに便利



データ並列化

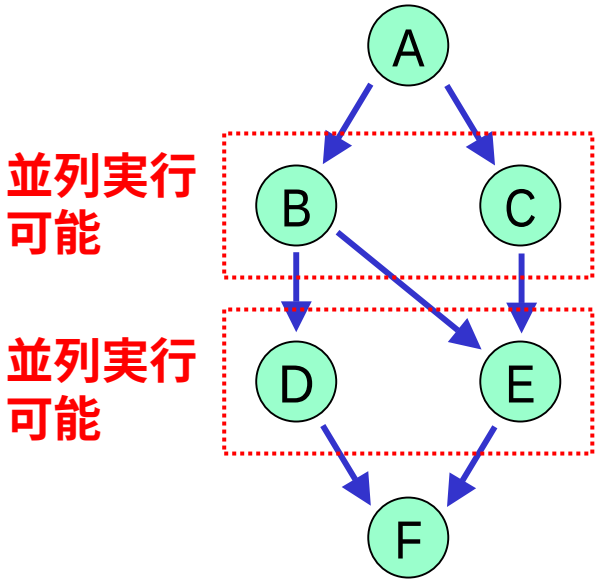
- 定義
  - 多数のデータに対して独立に同じ演算を行う場合に，各データに対する演算をそれぞれプロセッサに割り当てて並列化
    - 1個のプロセッサが複数のデータを担当してもよい
- 例
  - ベクトルの加算 $c = a + b$ を要素ごとに独立に計算
  - 科学技術計算･行列計算で最も多用される並列化方法
  - 本演習でも，この並列化方法を中心に扱う



n個のデータに対する処理A_1～A_nをそれぞれ別のプロセッサで実行

タスク並列化

- **定義**
  - **異なる処理A, B, C, … があるときに，それらの依存関係を解析し，同時に実行可能な処理をそれぞれプロセッサに割り当てて並列化**
    - **1個のプロセッサが複数の処理を担当してもよい**

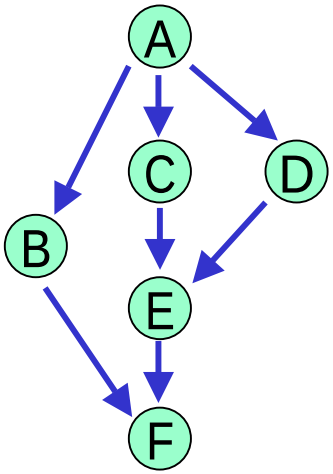


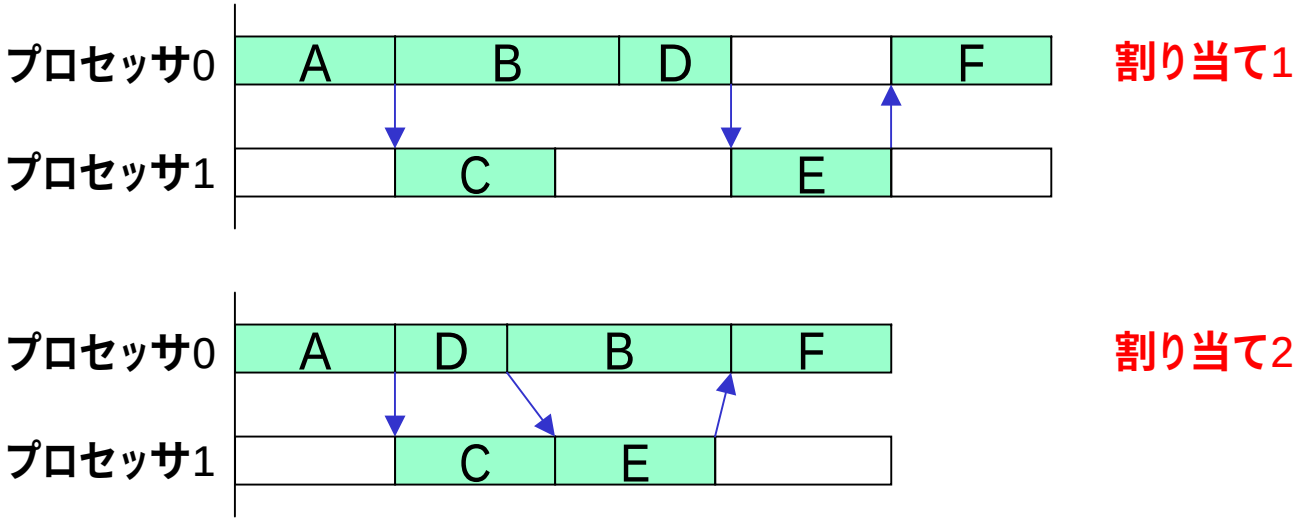
・BとCは同時に実行可能
・DとEも同時に実行可能

・BとDをプロセッサ0，CとEをプロセッサ1に割り当てれば，並列実行可能
・ただし，プロセッサ1はEを開始する前にプロセッサ0のBの終了を待つ必要あり。

タスク並列化 (続き)

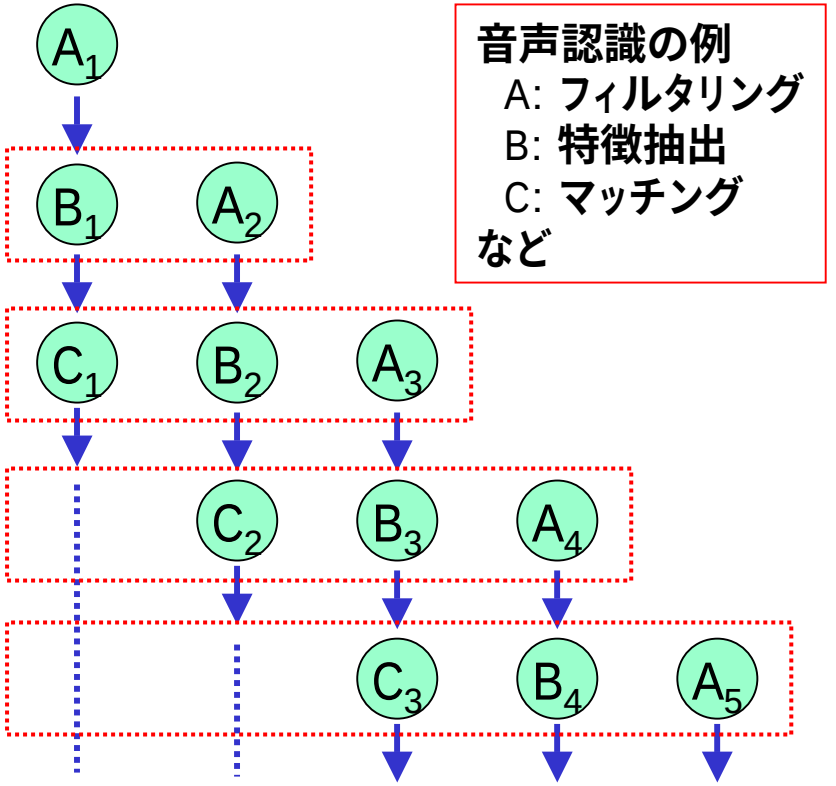
- スケジューリング
  - 処理のプロセッサへの割り当て方は，一般に複数存在する
  - 各処理の長さ，依存関係を考慮して割り当てを行うことにより，並列実行時間を短縮できる
    - 一般には難しい組合せ最適化問題
    - データ通信時間，共有資源へのアクセスなどを考慮しなくてはならない場合もあり





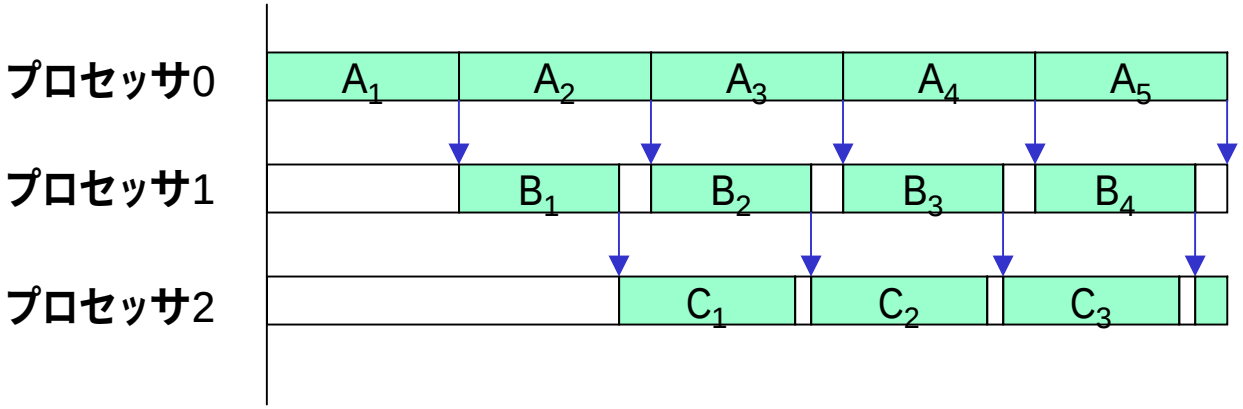
パイプライン型並列化

- 定義
  - 順番にやってくるデータ1，2，3，… に対し，それぞれ処理A，B，C，… を順番に施す場合，各処理に対して1個のプロセッサを割り当てることにより並列化できる。これを**パイプライン型並列化**という。
- データ並列化との比較
  - 各データの処理が独立ならば，データ並列化を用いてもよい
  - しかし，各処理に対して特別なハードウェアあるいはデータが必要な場合は，パイプライン型並列化が有効
- 応用
  - 信号処理など



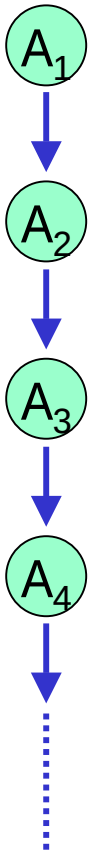
パイプライン型並列化 (続き)

- 適用に当たっての注意
  - プロセッサの利用効率を上げるには，パイプラインの各処理の実行時間をほぼ同じにする必要あり
  - パイプライン型並列化では，データ処理のスループットは向上するが，1個のデータに対する処理時間は短縮されないことに注意



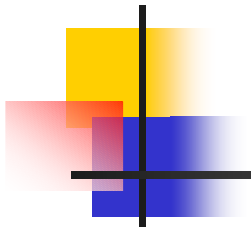
並列化できない演算

- 逐次的な計算
  - 処理 A_n が終わらないと処理 A_{n+1} が開始できない場合
  - 制御フローグラフは直線



- 例
  - 擬似乱数の生成： $x_{n+1} = ax_n + b \pmod{m}$

しかし，このような制御フローグラフに対しても，アルゴリズムの変更により並列化できる場合がある（後述）

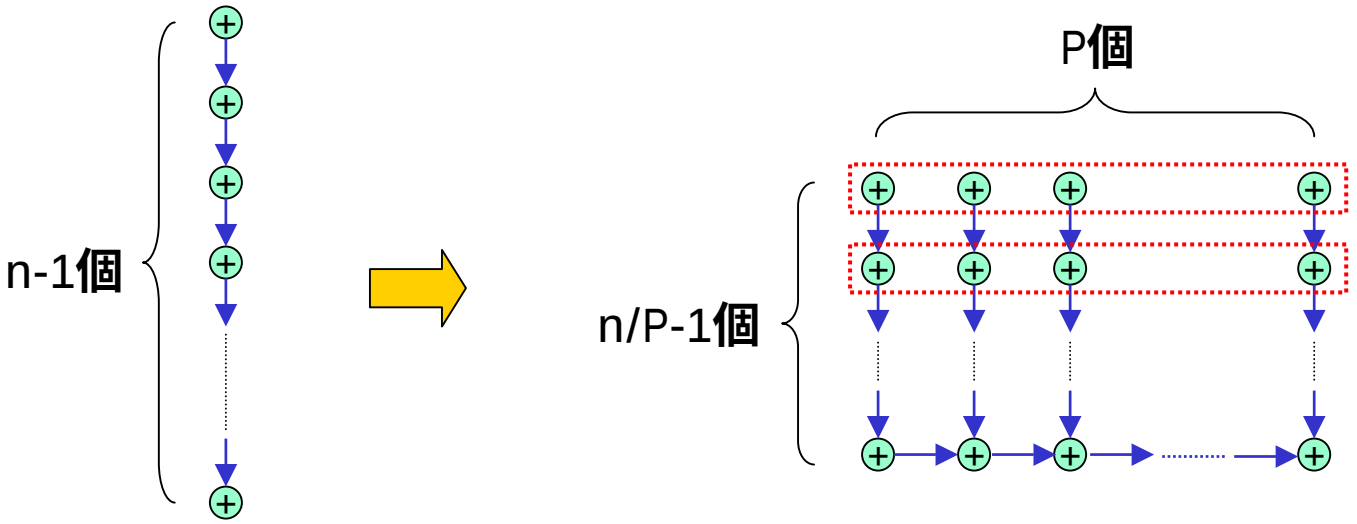


3．数値計算アルゴリズムとその並列化

- 総和演算
- 基本行列演算
- 合同乗算法による擬似乱数生成
- 2次元の温度分布の計算

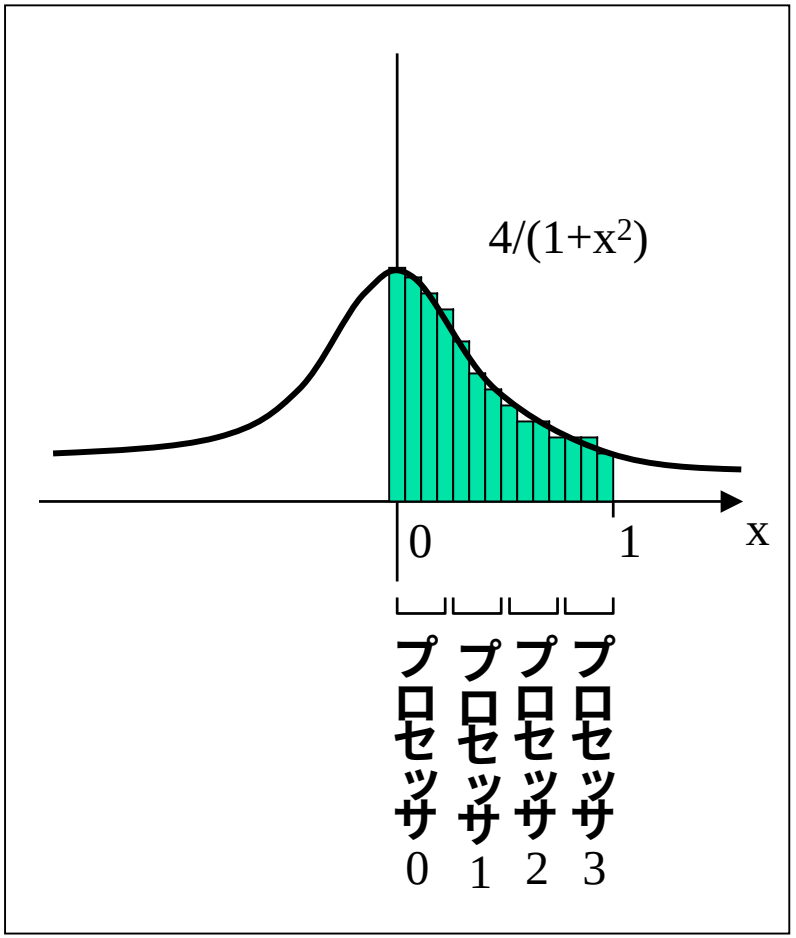
総和演算

- **問題**
  - **長さnの配列** a(1), a(2), …, a(n) **の要素の和を求める**
- **並列化方法**
  - **配列をP (プロセッサ台数) 個の部分配列に分割**
  - **各プロセッサで部分和を求め，最後に1プロセッサで集計**



総和演算 (続き)

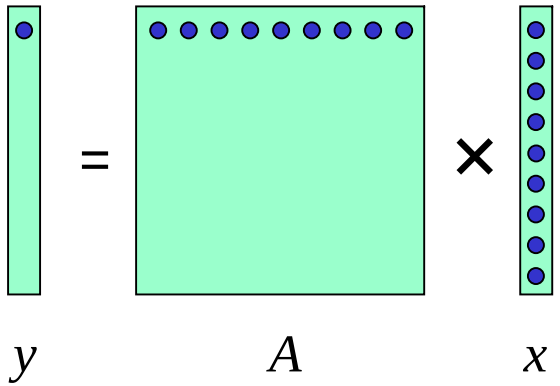
- 数値積分によるπの計算
  - $\pi = \int_0^1 4/(1+x^2)\,dx$ の積分区間をn等分し，中点則により計算
- 計算の並列化
  - n個の長方形を4個のプロセッサに割り当てる
  - 各プロセッサは，担当するn/4個の長方形それぞれの面積を求め，その部分和を計算する
  - プロセッサ0は，各プロセッサの求めた部分和を足し合わせる



基本行列演算

- **行列とベクトルの積** $y = Ax$

```
do i=1, n
   s = 0.0
   do j=1, n
      s = s + a(i,j) * x(j)
   end do
   y(i) = s
end do
```

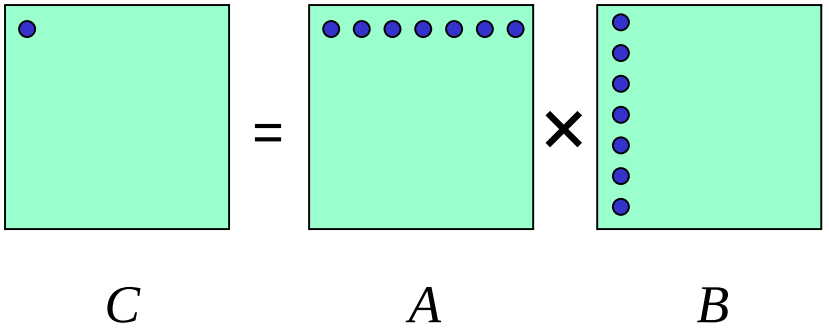


- **並列化方法**
  - **各 i についての計算は独立なので, i のループを並列化可能**
    - **ベクトル y の各要素は, 他とは独立に計算できる**
  - **j のループは総和なので, これを並列化することも可能**

基本行列演算 (続き)

- **行列どうしの積** $C = AB$

```
do i=1, n
   do j=1, n
      s = 0.0
      do k=1, n
         s = s + a(i,k) * b(k,j)
      end do
      c(i,j) = s
   end do
end do
```



- **並列化方法**
  - **各 i についての計算は独立なので, i のループを並列化可能**
  - **各 j についての計算は独立なので, j のループも並列化可能**
  - **k のループは総和なので, これを並列化することも可能**

合同乗算法による擬似乱数生成

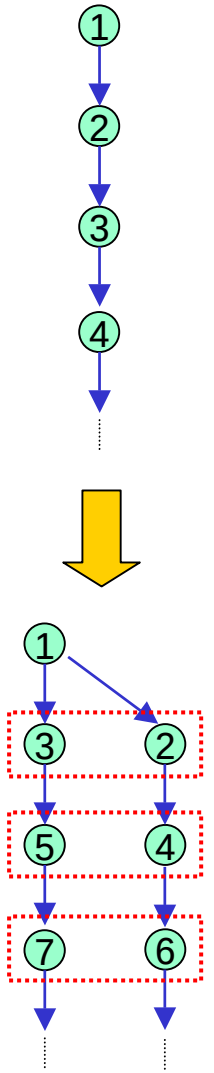
- 計算式

$$x_{n+1} = ax_n + b \pmod{m}$$

- 並列化方法
  - 次のように式変形を行う

$$\begin{aligned} x_{n+1} &= a(ax_{n-1} + b) + b \pmod{m} \\ &= a^2 x_{n-1} + (a+1)b \pmod{m} \\ &= a' x_{n-1} + b' \pmod{m} \end{aligned}$$

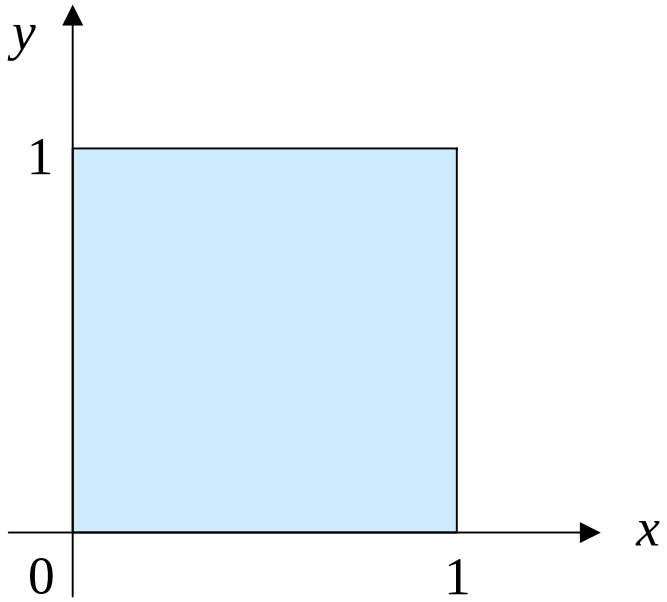
  - これにより奇数項と偶数項を並列に計算可能
  - この変形を再帰的に適用することで，さらに並列性を向上できる
  - この技法を recursive doubling と呼ぶ



2次元の温度分布の計算

- **問題**
  - **2次元正方形領域** $[0,1]\times[0,1]$ **での熱伝導を考える**
  - **境界をすべて0℃に固定**

$$u(0, y) = 0$$
$$u(1, y) = 0$$
$$u(x, 0) = 0$$
$$u(x, 1) = 0$$



  - **領域全体に一定の熱を加える**

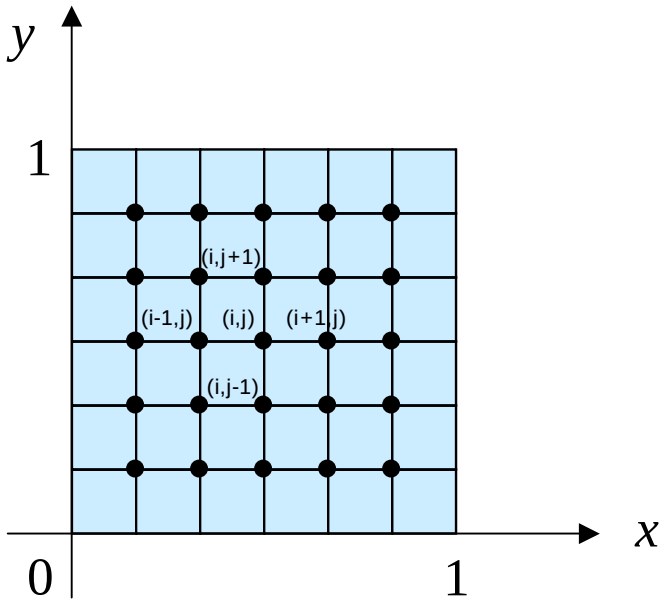
このとき，十分な時間が経った後での温度分布はどうなるか？

2次元の温度分布の計算 (続き)

- **離散化**
  - 領域内を格子に区切り，格子点上での温度のみを考える
  - さらに，離散的な時間ステップでの温度のみを考える

- **温度の従う方程式**
  - 時間ステップ n での格子点 (i, j) の温度を $u_{ij}^{(n)}$ とすると，

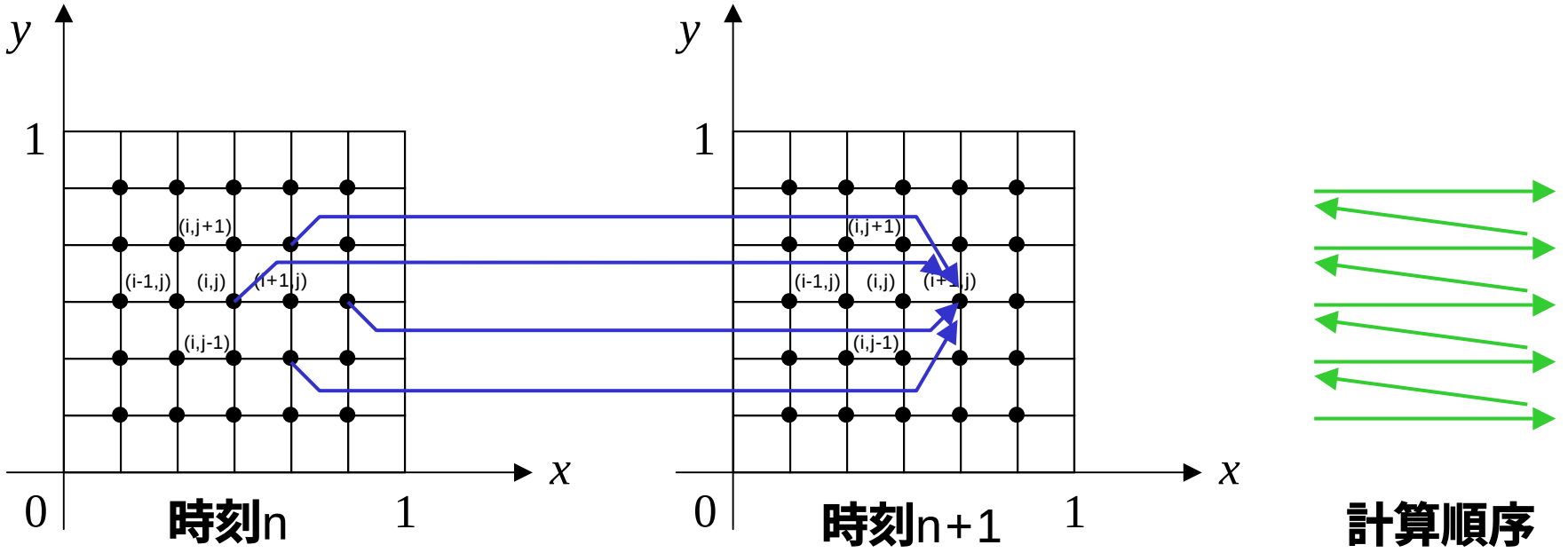
$$u_{ij}^{(n+1)} = (u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)}) / 4 + f_{ij}$$



2次元の温度分布の計算 (続き)

- **時間発展のアルゴリズム (ヤコビ法)**

```
do j=1, N
    do i=1, N
        u_ij^(n+1) = (u_i-1,j^(n) + u_i+1,j^(n) + u_i,j-1^(n) + u_i,j+1^(n) ) / 4 + f_ij
    end do
end do
```



2次元の温度分布の計算 (続き)

- **並列化方法**
  - **ステップ $n+1$ での値 $u_{ij}^{(n+1)}$ は, ステップ n での値 $u_{ij}^{(n)}$ のみを使って計算されている**
  - **したがって, N^2 個の $u_{ij}^{(n+1)}$ はすべて独立に計算可能**
  - **プログラムでは, i のループと j のループがそれぞれ並列化可能**

```
do j=1, N
    do i=1, N
        u_ij^(n+1) = (u_i-1,j^(n) + u_i+1,j^(n) + u_i,j-1^(n) + u_i,j+1^(n) ) / 4 + f_ij
    end do
end do
```

2次元の温度分布の計算 (続き)

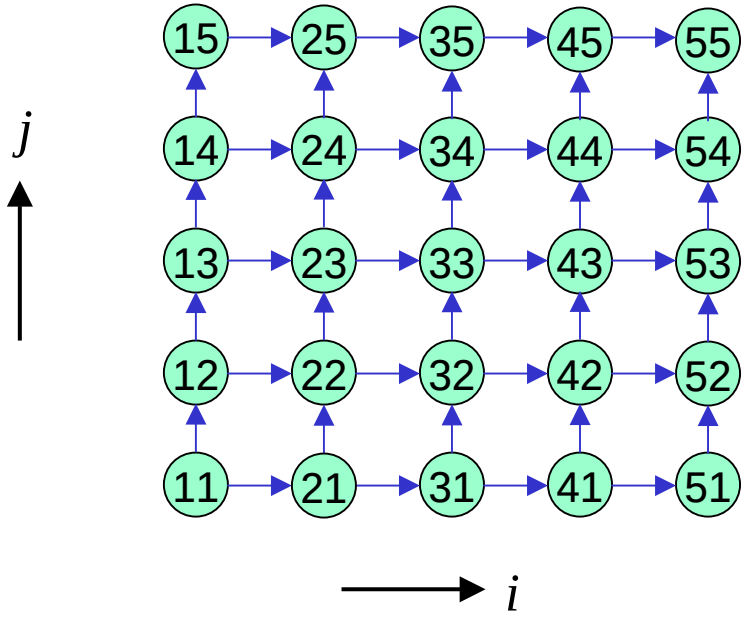
- 収束を加速するアルゴリズム (ガウス-ザイデル法)
  - ヤコビ法において，計算が左下隅から順に行われることに着目
  - $u_{ij}^{(n+1)}$ を計算する時点では，左および下の点については，既に新しい値 $u_{i-1,j}^{(n+1)}$，$u_{i,j-1}^{(n+1)}$ が計算されている
  - そこで，これらの点について新しい値を使うことで，最終状態への収束を加速する
    - 理論的には，収束が2倍速くなることが示せる

```
do j=1, N
    do i=1, N
        u_ij^(n+1) = (u_i-1,j^(n+1) + u_i+1,j^(n) + u_i,j-1^(n+1) + u_i,j+1^(n) ) / 4 + f_ij
    end do
end do
```

2次元の温度分布の計算 (続き)

- ガウス-ザイデル法の並列性
  - 同じ行内では，左の結果を使って計算を行うので，並列性はない
  - 同じ列内では，下の結果を使って計算を行うので，並列性はない

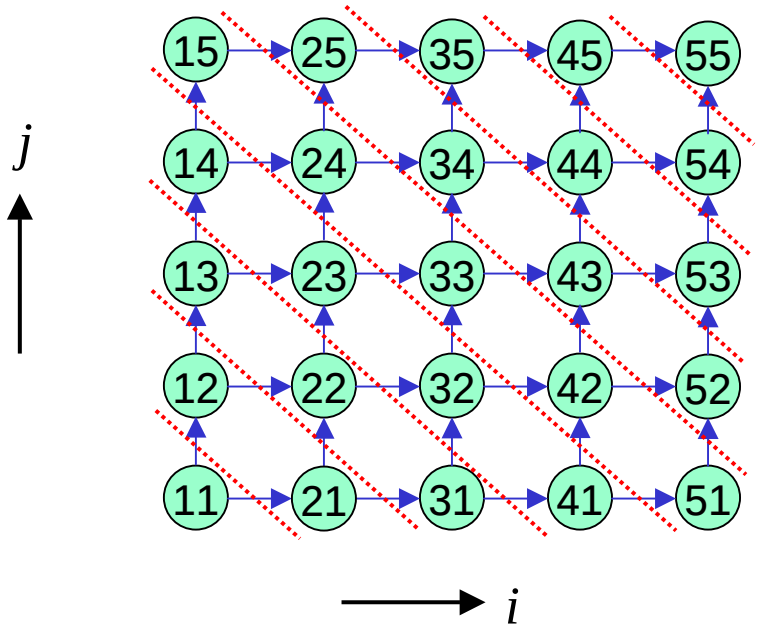
⇒ では，ガウス-ザイデル法は並列化不可能か？

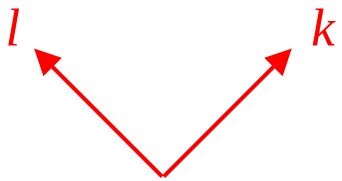


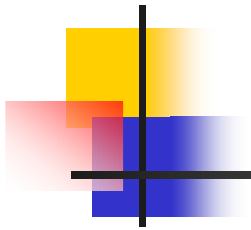
ij ： $u_{ij}^{(n+1)}$ の計算

2次元の温度分布の計算 (続き)

- ループ組み換えによる並列化
  - 制御フローグラフを調べると，対角線に平行な要素の間には，依存関係がないことがわかる
  - そこで，i と j についての2重ループを組み替え，反対角線方向 (k) と対角線方向 (l) のループにすれば，l について並列化が可能
  - これを超平面法と呼ぶ





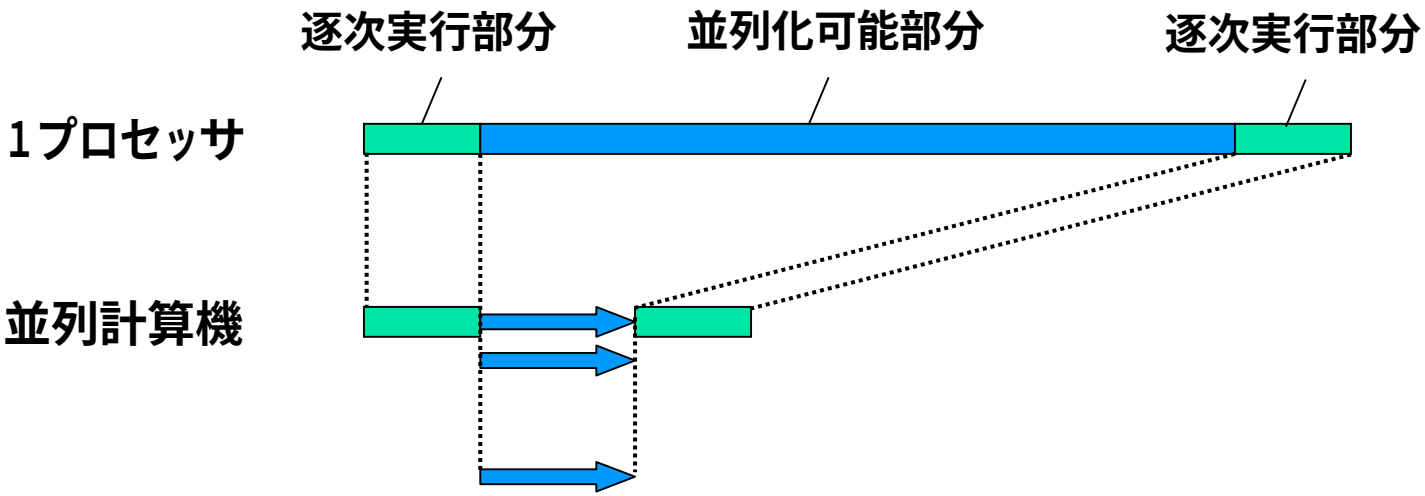


4. 並列性能の向上のために

- 並列処理による高速化
- アムダールの法則
- 並列性能の向上のために

並列処理による高速化

- 並列化可能部分と逐次実行部分
  - プログラム中で独立に実行可能な部分（並列化可能部分）を複数のプロセッサで分担することにより，実行時間を短縮
  - 並列化可能部分の割合が大きいほど，並列化の効果が大きい
  - 逐次実行部分の割合が大きいと，並列化の効果は小さい



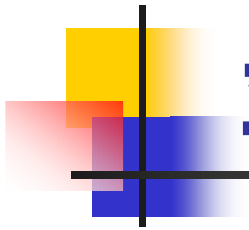
アムダールの法則

- 加速率と並列化効率
  - プロセッサ数が1，P のときの実行時間をそれぞれ T_1，T_P とするとき，T_1 / T_P を**加速率**と呼ぶ
  - $T_1 / (P * T_P)$ を**並列化効率**と呼ぶ

- アムダールの法則
  - プログラムの実行時間のうち，並列化可能部分が占める割合を q とすると，

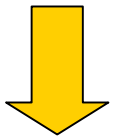
$$\text{加速率の上限} = 1/((1-q) + q/P)$$

  - 実際は，負荷不均衡やスレッド起動コストなどで，より低い値
  - 逐次実行部分が10%あると，10プロセッサを使っても最大5倍程度の加速率しか得られない



並列性能の向上のために

- 並列計算機の性能を引き出すプログラムを作るには，以下の点に関する考慮が重要
  - プロセッサ間の負荷分散
  - 並列粒度と同期回数
  - プロセッサ間通信の回数と量



- これらについては，以下の演習の中で順に説明してゆく