並列プログラミング方法論:
アルゴリズム

2016 年

Rev 1.0

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

コースの目的

- シリアル・コードから並列アプリケーションを設計する
- アプリケーションの適切な分割方法を決定する
- 実装する並列モデルを選択する

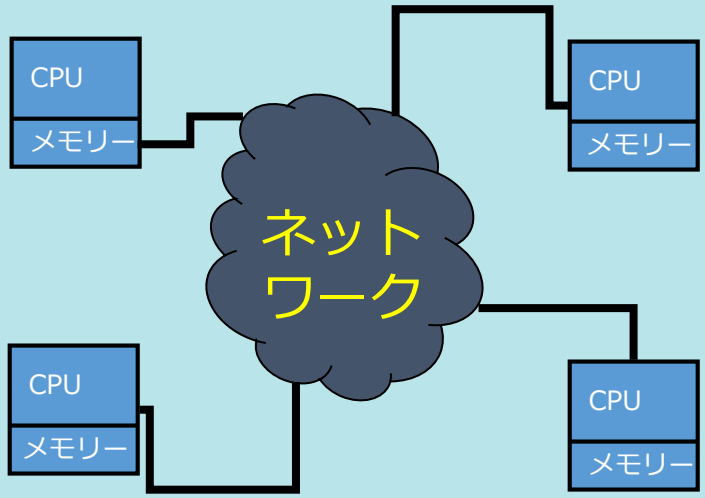
© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

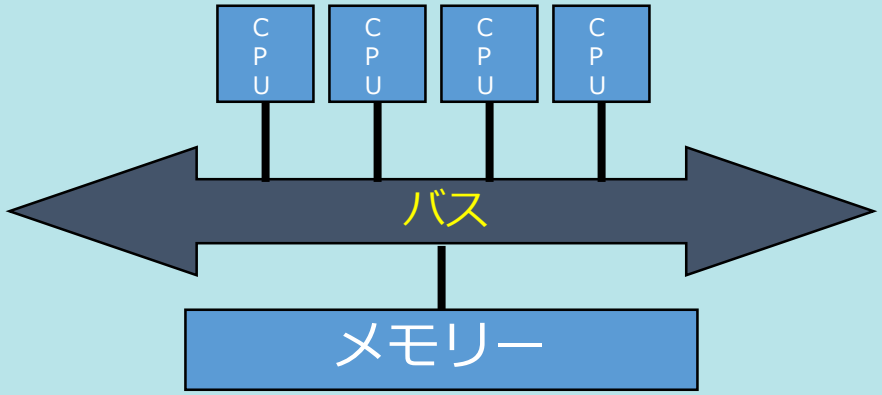
コースの内容

1. 分散コンピューティングと共有コンピューティング
2. 並列プログラム設計方法論
   - パーティショニング　　(分割)
   - コミュニケーション　　(通信)
   - アグロメレーション　　(結合)
   - マッピング　　(割り当て)
3. 分散と共有の例

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

分散メモリーと共有メモリー





メッセージ・パッシング

マルチプロセス

メッセージとデータを共有

MPI

スレッド

シングルプロセス

　並行実行

共有メモリーとリソース

OpenMP*、Cilk™ Plus、TBB、など

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

Foster の設計方法論

Ian Foster 著
『Designing and Building Parallel Programs』より

4 つのステップ:

- 演算とデータを分割
- 演算間のデータを共有
- タスクをグループ化してパフォーマンスを向上
- プロセッサー/スレッドにタスクを割り当て

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

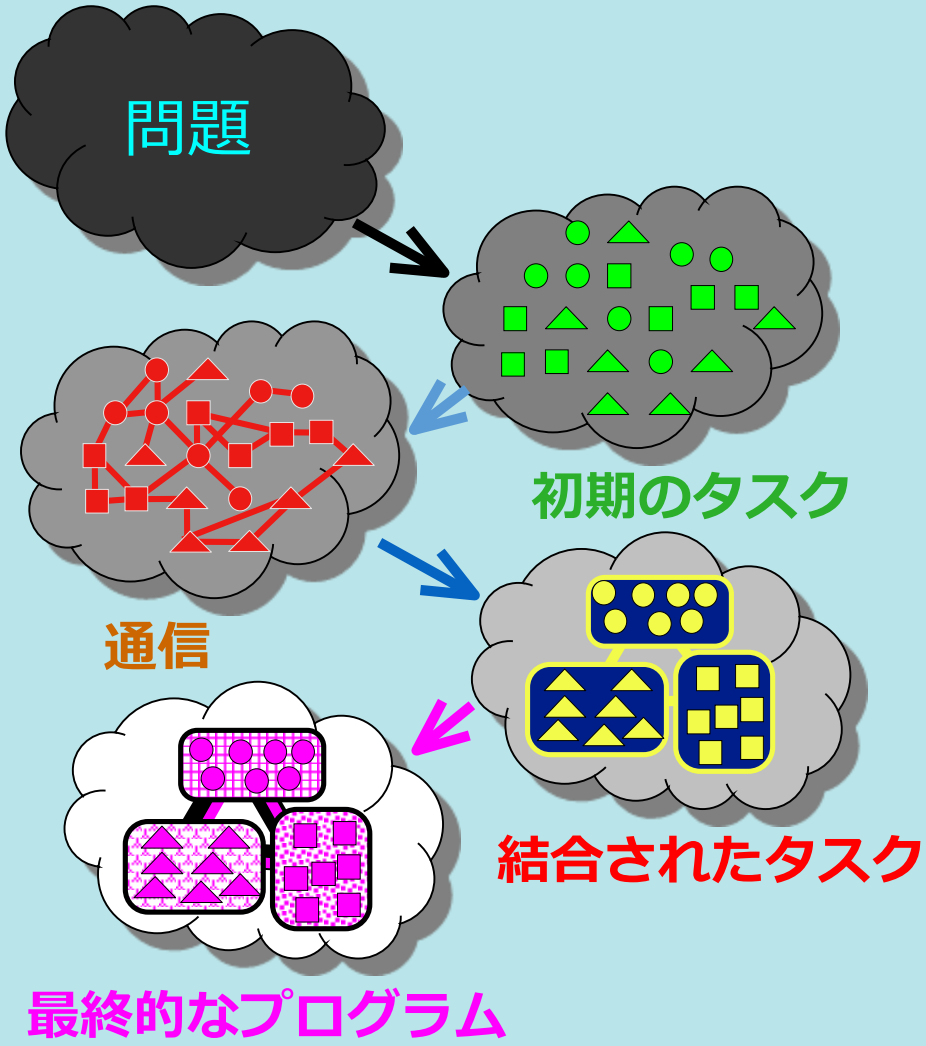
並列プログラムの設計

パーティショニング
- 問題をタスクに分割

通信
- 通信の量とパターンを決定

アグロメレーション（結合）
- タスクを結合

マッピング
- 結合されたタスクを
  物理プロセッサーに割り当て



© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

1. パーティショニング

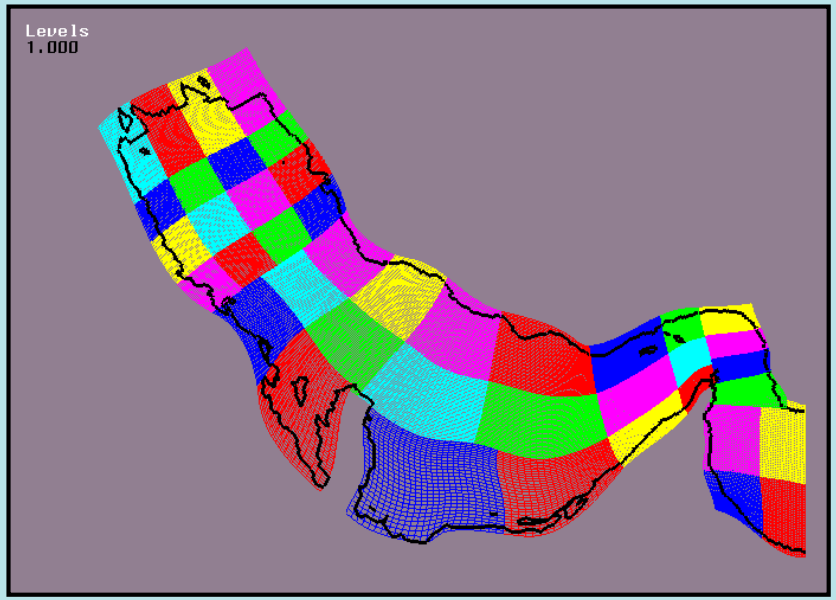
- 可能な限り並列処理を見つける
  - 独立した演算およびデータ
  - プリミティブ・タスクの数を最大限に
- 機能分割
  - 演算を分割してからデータを関連付け
- ドメイン分割
  - データを分割してから演算を関連付け

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

分割メソッド

機能分割

– 問題の構造を明らかにできる演算に注目



Phu V. Luong 博士（沿岸水力研究所、技術者研究開発センター）の許可によりグリッドを転載

ドメイン分割

– 最大のデータ構造、または最も頻繁にアクセスされるデータ構造に注目

– データの並列処理

  – すべてのデータに同じ演算を適用

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列でないものとは?

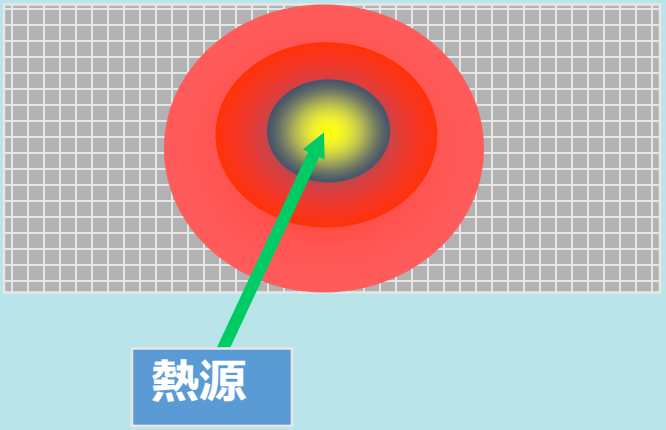
- “ステート” を伴うサブプログラム
  - 擬似乱数生成
  - ファイル I/O ルーチン
- 依存関係があるループ
  - ある反復で書き込まれ、別の反復で読み取られる変数
  - 簡単な依存性テスト: ループの反復を逆にする

ループキャリー：	値をある反復から次の反復に運ぶ
帰納変数：	ループごとにインクリメントされる
リダクション：	配列を単一データに変換する
循環：	次の反復に情報を伝える

補足資料を参照

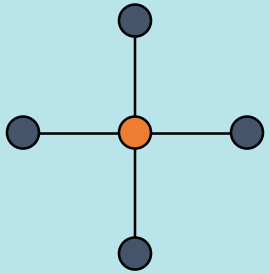
© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

例: 熱散逸



5 ポイントのステンシルを使用して各ポイントの curr 値を計算

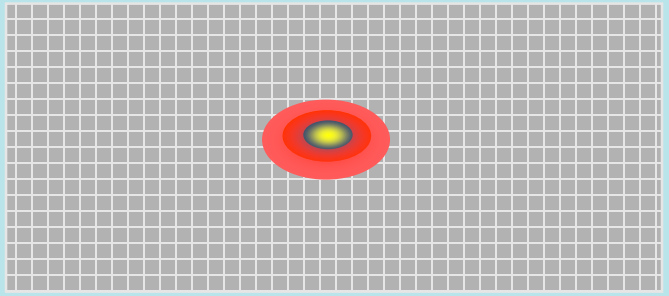
```
float curr[*][*], prev[*][*];

Initialize curr;
while (!stable) {
  copy curr => prev;  //swap pointers
  for (i = …)
    for (j = …)
      Compute curr;
  for (i = …)
    for (j = …)
      Sum up differences;
  stable = (avg diff < tolerance);
}
```



```
curr[i][j] = (prev[i][j] +
              prev[i][j+1] + prev[i][j-1] +
              prev[i+1][j] + prev[i-1][j]) / 5.0;
```


© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

例: パーティショニング



プリミティブ・タスクは?

ポイントの初期化
ポインタのコピー
ステンシルの演算

```
float curr[*][*], prev[*][*];

Initialize curr;
while (!stable) {
  copy curr => prev;  //swap pointers
  for (i = …)
    for (j = …)
      Compute curr;
  for (i = …)
    for (j = …)
      Sum up differences;
  stable = (avg diff < tolerance);
}
```

- 差の演算
- 差を合計に追加
- 平均の演算

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

2. 通信

- プリミティブ・タスク間の通信パターンを決定
  - 共有するデータは?
- 1 対 1 通信
  - あるスレッドから別のスレッド
- 集合通信
  - データを共有するスレッドのグループ
- 実行順の依存関係は通信

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

データの共有

メッセージ・パッシング

- メッセージを送受信
- ブロードキャスト
- スキャッター、ギャザー

スレッド

- 共有メモリー領域に値を格納
  - 同期アクセスが必要

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

データの保護

メッセージ・パッシング

- 別のプロセスからメモリーを読み取ることができない

スレッド

- スレッド・ローカル・ストレージ（TLS）
- スレッド・スタックと関数からのスコープ
- OpenMP* スコープ
- ミューテックス（Mutex）

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

データの競合

- 複数のスレッドが共有データにアクセス
- 実行順は仮定されているが保証されていない
- 診断が困難
  - 不確定
  - デバッグ手段がデータの競合を隠匿してしまう
  - 数値偏差による再現性

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

クリティカル・セクション

- アトミックに実行する必要があるコード・セクション
  - 割り込みなしで共有変数を読み取りまたは更新
- ミューテックス（排他処理）を行うために使用される同期オブジェクト:
  - ロック
  - セマフォ
  - 条件変数
  - CRITICAL_SECTION
  - Mutex
  - イベント

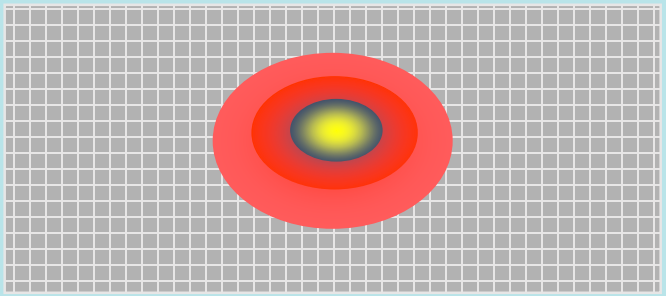
© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

実行の同期

- スレッドはプログラムで同期する必要がある
  - タスク B の開始前にタスク A が終わることを保証する
- バリアー
  - スレッドはすべてのスレッドがバリアーに進むまで休止
- イベント、シグナル、条件変数
  - スレッドは処理を進める前にシグナル（メッセージ）を待つ

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

例: 通信



プリミティブ・タスク間の通信を制限するのは?

- *curr* を計算するには、*prev* から 5 ポイントのステンシル・データが必要
- 実行順序の依存関係

```
float curr[*][*], prev[*][*];

Initialize curr;
while (!stable) {
   copy curr => prev;  //swap pointers
   for (i = …)
      for (j = …)
         Compute curr;
   for (i = …)
      for (j = …)
         Sum up differences;
   stable = (avg diff < tolerance);
}
```

- 初期化/コピー
- *curr* の計算
- 差の合計
- 平均の演算

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

3. アグロメレーション（結合）

- 次のようにプリミティブ・タスクをグループ化
  - パフォーマンス/粒度を向上
  - 通信を集中
    - 通信するタスクを同じグループにする
  - 設計上のスケーラビリティーを維持
    - データセットのサイズまたはプロセッサー数による変更を細かく制御
  - プログラミングとメンテナンスを単純化

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

作業のレプリケーション

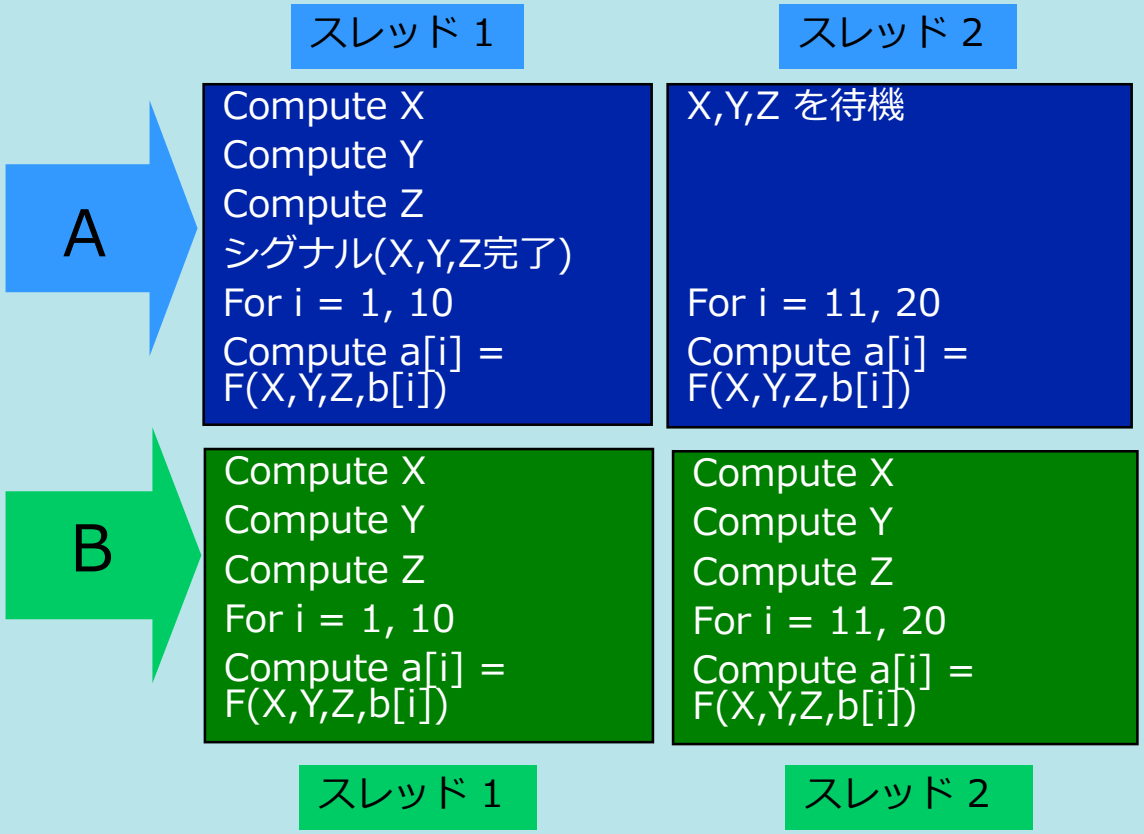
通信を減らすための演算の複製とのトレードオフ

順次アルゴリズム:

```
Compute X
Compute Y
Compute Z
For i = 1, 20
Compute a[i] = F(X,Y,Z,b[i])
```

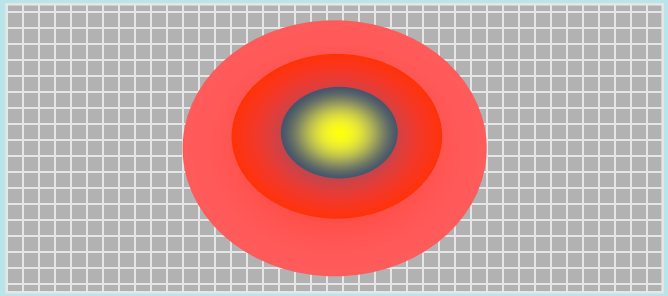
どちらのタスクのアグロメレーションの方が同期が減るか?

メモリーのトレードオフに注意!



© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

実習: アグロメレーション（結合）



どのようにタスクを組み合わせれば通信を減らしてスレッドのオーバーヘッドを少なくできるか?

グリッドポイントを論理単位にグループ化

```
float curr[*][*], prev[*][*];

Initialize curr;
while (!stable) {
   copy curr => prev;  //swap pointers
   for i = …
      for j = …
         Compute curr;
   for i = …
      for j = …
         Sum up differences;
   stable = (avg diff < tolerance);
}
```


© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

4. マッピング

- 次のようにプロセッサにタスクを割り当てる
  - プロセッサの使用率を最大化
  - プロセッサ間の通信を最小化
- プロセッサごとに 1 つのタスクか、複数のタスクか?
- 静的割り当てか、動的割り当てか?
- 大部分はメッセージ・パッシングに適用可能
  - 開発者はスレッドにタスクをマップできる

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

静的スケジューリング

- アプリケーションの最初にデータ/タスクを分割
- 割り当てはスレッドの数に基づく
- データの分散方法は?

メッセージ・パッシング	スレッド
• 単一ソースからデータを送信 – 単一プロセスで I/O を処理 • 個別入力	• 共有メモリ領域に値を格納 – 同期アクセス • 個別入力

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

動的スケジューリング

- 実行中にデータ/タスクを割り当て
- 不明または可変
- データ/タスクの分散方法は?

メッセージ・パッシング	スレッド
• 単一キューから取得 – ボス-ワーカー • 静的配分、動的再バランス	• ボス-ワーカー • OpenMP* 動的スケジューリング • OpenMP* WorkQueue – インテル® コンパイラーでサポート

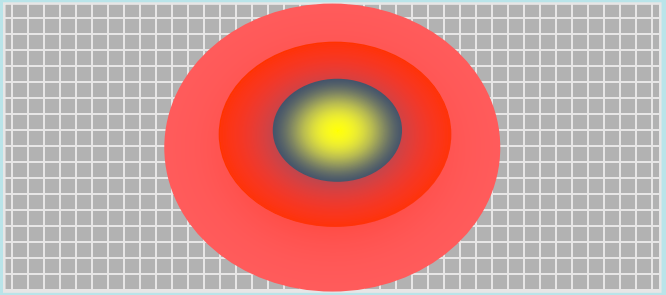
© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

スレッドプール

- 小さなトランザクションを処理するアプリケーションを考える
- なぜ新しいトランザクションを処理するたびに「一時的な」スレッドを作成しないのか?
  - スレッド作成と破棄のオーバーヘッド
  - 他のリソースがスレッドによって使用される
- より良いソリューション: スレッドプール
  - スレッドの数を制限してスポーン
  - 制御するスレッドのトランザクションをキューに入れる
- インテルの OpenMP* はこの方式を採用

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

実習: マッピング



結合したタスクをスレッドに
どのように割り当てられるか?

```
float curr[*][*], prev[*][*];

Initialize curr;
while (!stable) {
   copy curr => prev;  //swap pointers
   for i = …
      for j = …
         Compute curr;
   for i = …
      for j = …
         Sum up differences;
   stable = (avg diff < tolerance);
}
```

メッセージ・パッシング
- グリッド数をプロセス数で割る
- ゴーストセルを使用して境界の *prev* 値を保持
- 集団平均

- 明示的なスレッド
  - メッセージ・パッシングと同じ、ゴーストセルなし
- OpenMP*
  - ループの周囲にプラグマを配置
  - 平均にリダクションを使用

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

並列プログラムにおける問題点

- シリアルコードでは発生しない他の問題
  - デッドロック
  - 丸め誤差

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

デッドロック

- 決して発生しないイベント/オブジェクト/メッセージを待つスレッド

メッセージ・パッシング	スレッド
• 実行していないプロセスからのメッセージを待つ • 不適切な送信と受信操作の組み合わせ • 異なるバリアで待機	• 正しくない階層のロック – T1 が A をロック、B が必要 – T2 が B をロック、A が必要 • 同期オブジェクトの保持を終了したスレッド • 異なるバリアで待機

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

丸め誤差

- 有限桁数で行われるコンピューター演算には誤差が含まれる
- 並列処理では、操作が連続した順に行われないことがある
  - 結果が異なる可能性がある

ε（1.0 + ε = 1.0 のような非常に小さな数）を計算する

浮動小数点演算では、次のようになる

-1.0 + (1.0 + ε) = -1.0 + 1.0 = 0.0

つまり、

(-1.0 + 1.0) + ε = 0.0 + ε = ε

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

パーティショニングのチェックリスト

- 分割の品質の評価
  - プロセッサー数よりもプリミティブ・タスクの数は多いか?
  - 冗長演算およびデータ格納領域は最小限にできたか?
  - プリミティブ・タスクはほぼ同じサイズか?
  - タスクの数は問題のサイズに基づいているか?

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

通信のチェックリスト

- 通信の品質の評価
  - 通信操作はバランスが取れているか?
  - 各タスクは隣接する少数のタスクとのみ通信しているか?
  - タスクは通信を同時に実行できるか?
  - タスクは演算を同時に実行できるか?

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

アグロメレーションのチェックリスト

- アグロメレーションの品質の評価
  - 通信の局所性は増加したか?
  - 結合されたタスクの演算と通信コストは同じであるか?
  - 複製された演算は置換された通信よりも時間がかからないか?
  - 複製されたデータの量はアルゴリズムがスケーリング可能な量か?
  - コード修正のトレードオフは適切か?

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

マッピングのチェックリスト

- マッピングの品質の評価
  - プロセッサ設計について 1 つのタスクと複数のタスクの両方が考慮されたか?
  - 静的割り当てと動的割り当ての両方が評価されたか?
  - 動的の場合、マネージャー・スレッドがボトルネックではないか?
  - 静的の場合、ロードバランスが考慮されたか?

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

まとめ

- 並列アプリケーション設計の 4 つのプロセス
  - データまたはタスク別に問題を分割する
  - タスク間の通信を定義する
  - 通信を保持するように結合する
  - タスクをプロセス/スレッドにマップする
- 並列プログラミング・エラーに注意

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

アルゴリズムの戦略：
SPMD (Single Program Multiple Data) の設計パターン

- P が大きくなる可能性のある処理要素上で、同じプログラムを実行する
- 一連のタスクを選択、そして任意のデータ構造を管理するためランクを利用 ... ランクは 0 から (P-1) の ID

このパターンは非常に一般的であり、多くのアルゴリズム戦略のパターン（全てではない）をサポートするために利用されてきた

MPI プログラムはほとんどこのパターンを使用 ... 並列プログラミングの歴史の中で最も一般的に利用されてきたパターン

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

例:単純な SPMD 並列 pi プログラム

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{           int i, nthreads;  double pi, sum[NUM_THREADS];
            step = 1.0/(double) num_steps;
            omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   {
          int i, id,nthrds;
           double x;
           id = omp_get_thread_num();
           nthrds = omp_get_num_threads();
           if (id == 0)   nthreads = nthrds;
           for (i=id, sum[id]=0.0;i< num_steps; i=i+nthrds) {
                     x = (i+0.5)*step;
                     sum[id] += 4.0/(1.0+x*x);
           }
    }
            for(i=0, pi=0.0;i<nthreads;i++)pi += sum[i] * step;
}
```

競合を避けるため、スカラー変数に変わりスレッド数分の配列を定義

複数のフレッドが同じアドレスへ書き込む際に、競合を起こさないよう 1 つのスレッドのみがグローバル変数を安全に更新できるようにする

これは、ループ反復の周期的な分配を行う際における SPMD プログラムの一般的な手法

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

一般的なリソース情報

- IA Software User Sciety (iSUS)

  http://isus.jp

- インテル® Developer Zone
  https://software.intel.com/en-us/ （英語）

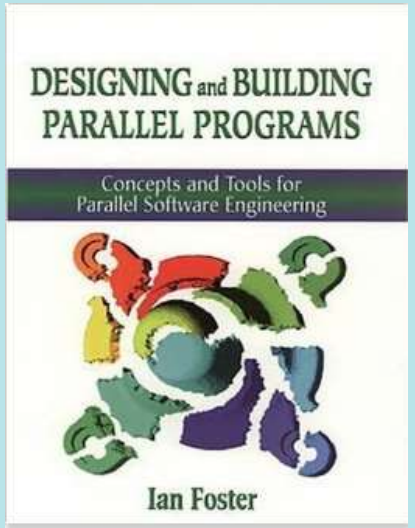
- インテル® ソフトウェア開発製品
  http://www.xlsoft.com/jp/products/intel/index.html

- 各コースのセッション資料の最後にあるトピック特有の情報を参照

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

参考文献

Foster, Ian T.著
『Designing and Building Parallel Programs』
Boston: Addison-Wesley, 1995



この本の内容は www-unix.mcs.anl.gov/dbpp から無料でダウンロード可能

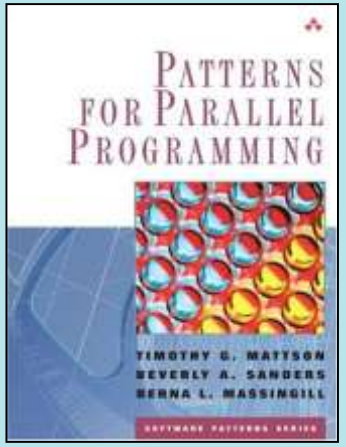
MPIによるプログラミング概要 RIKEN AICS HPC Summer School 2014:
https://www.google.co.jp/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&uact=8&ved=0ahUKEwi3ncaY_-jLAhXEF5QKHcK5AicQFgg9MAU&url=https%3A%2F%2Fwww.aics.riken.jp%2Faicssite%2Fwp-content%2Fuploads%2F2014%2F08%2Fss14_MPIprog-C1_rev.pdf&usg=AFQjCNGIvM7YfWDyKgIKvYYS4tno00p5ng

MPI API の使い方: http://www.cv.titech.ac.jp/~hiro-lab/study/mpi_reference/index.html

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

参考文献

- 並列アルゴリズム設計におけるパターン言語について解説
- MPI、OpenMP そして Java による例題を提示
- プログラマーがどのように並列プログラミングについて考えるのか、著者の仮説を述べている



Patterns for Parallel Programming, Timothy G. Mattson, Beverly A. Sanders, Berna L. Massingill, Addison-Wesley, 2005, ISBN 0321228111

- 構造化並列プログラミングによる効率良い計算を行うためのパターン
- 並列プログラミングにおけるデザインパターンを示す手引書です
- パターンの概念を文書や図表だけで理解いただくのは容易でないため、本書では随所にサンプルコードが紹介されています。C/C++、そして一部 Fortran を利用する開発者の方々に役立つ書籍です

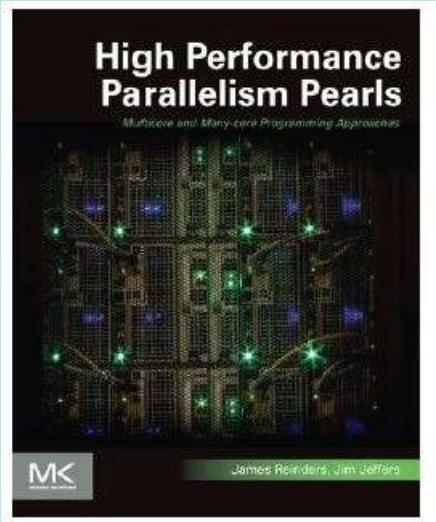
カットシステム 版
Michael McCool, Arch D. Robinson, James Reinders 著
菅原 清文／エクセルソフト株式会社 訳
2013 年 ISBN 978-4-87783-305-3

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

参考文献（新刊）

Multithreading in Visual Effects

DreamWorks Animation、Pixar、Side Ef fects、AMD、Intel から集まった著者達による、SIGGRAPH 2013 コース「Multithreading in Visual Effects」 をベースにした書籍です。コースで使用された資料を大幅に拡張/ 更新し、 Pixar の Manuel Kraemer 氏により OpenSubdiv に関する章が追加されています。Ron Henderson 氏は、 FLUX ガス・シミュレーション・システム (第 5 章) の開発が評価され、2014 年 2 月に Technical Achievement Award を受賞しました。



High Performance Parallelism Pearls

簡単にインテル® Xeon Phi™ コプロセッサー・ファミリーの優れた並列性を利用してコードを実行できるため、最適化に集中し、ハイパフォーマンスを実現することが可能です。並列処理を細かくチューニングすることで、正しいアプリケーションを正しく効率良いアプリケーションにすることができます。
インテル コーポレーションの並列プログラミング・エバンジェリストであるJames Reinders とインテル コーポレーションのエンジニアである Jim Jeffers により執筆された最新の書籍は、69 人の専門家の実際の経験を基に、インテルのマルチコアおよびメニーコア・プロセッサーを最大限に利用するための創意工夫を紹介しています。

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

補足資料: 並列でない例

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

状態付きルーチン

- 多くのルーチンは呼び出しの状態を維持する
  - メモリー割り当て
  - 擬似乱数生成
  - I/O ルーチン
  - グラフィック・ライブラリー
  - サードパーティ・ライブラリー

- これらのルーチンへの並列アクセスは同期されていない限り安全ではない

- スレッドの安全性を決定する特定の関数についてのドキュメントを確認する

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

ループにおける反復間の依存関係

- wrap が 1 つの反復から次の反復に依存性を持っているため、このループは並列ではない

```
wrap = a[0] * b[0];
for (i=1; i<N; i++) {
  c[i] = wrap;
  wrap = a[i] * b[i];
  d[i] = 2 * wrap;
}
```

- wrap が各反復で使用される前に定義されるように再構成する

```
for (i=1; i<N; i++) {
  wrap = a[i-1] * b[i-1];
  c[i] = wrap;
  wrap = a[i] * b[i];
  d[i] = 2 * wrap;
}
```


© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

帰納変数

- 帰納変数は各ループの反復毎にインクリメントされる

- インクリメント式をループ・インデックスの純関数に置き換える

```
i1 = 0
i2 = 0
DO I=1,N
  i1 = i1 + 1
  B(i1) = ...
  i2 = i2 + I
  A(i2) = ...
ENDDO
```

```
DO I=1,N
 B(I) = ...
 A((I**2 + I)/2)= ...
ENDDO
```


© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

リダクション

- リダクションは、アソシエーティブ演算により配列データをスカラデータに変換する

```
do i=1,n
  sum = sum + c(i)
  maxx = max(maxx,c(i))
enddo


for (i=0; i<n; i++)
   sum += c[i];
```

- アソシエーティビティーを利用して、プライベート領域の部分和または極大値を計算する
- 次に、アクセスが同期するように注意しながら、部分的な結果を共有結果と組み合わせる

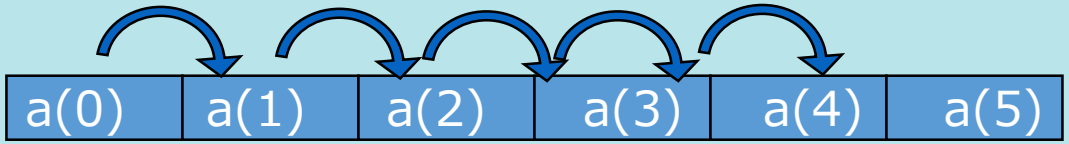
© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

循環

- 循環関係は、ある反復から次の反復に情報を伝える
  - 時間ステップのループ
  - 収束ループ

```
do i=1,n
  a(i) = a(i-1) + b(i)
enddo
```

大部分の循環は完全に並列化できない。 代わりに、より外側のループまたはより内側のループを探す



© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

Thank you !!

© 2016 iSUS. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。