実践編 II

担当：坂本 尚久

計算科学演習A2

2016年7月28日

※本資料は昨年の陰山先生の資料に坂本が加筆したものです。

リソースグループ

- 今日13:00から17:00の間だけ諸君だけでlargeキューを占有
- π-computerの演習用アカウント（ID）は当分の間有効だが、年度末までのどこかのタイミングで予告なく削除される。
- 研究等で継続利用したい場合は、指導教員と相談の上、利用申請を行う。

準備

- 作業ディレクトリの作成

```
$ cd                    ホームディレクトリに移動（場所はどこでもOK）
$ mkdir pract02         作業ディレクトリの作成（名前は何でもOK）
```

- サンプルコードをコピー

```
$ cd pract02            作業ディレクトリに移動
$ cp /tmp/160728/* .    サンプルコードをコピー
```

先週の課題の解答

- さきほどコピーした heat3.f90 をみよ。

時間計測

- heat5.f90
  - これまで使ってきたheat4…f90に、system_clock()関数を使った時計計測モジュールstopwatch_mを組み込んだ。

```
!
! heat5.f90
!   + module stopwatch, to monitor time.
!   + many calls to stopwatch__stt and .._stp.
!   - data output calls for profile 1d and 2d (commented out.)
!!   usage (on pi-computer)
!!    1) mkdir ../data  (unless there is already.)
!!    2) mpifrtpx -O3 heat5.f90  (copy un to u is slow in default.)
!!    3) pjsub heat5.sh
```

演習1

- heat5.f90を次のコマンドでコンパイルし、実行せよ。

```
$ mpifrtpx heat5.f90
$ pjsub heat5.sh
```

  - どこで時間がかかっているか調べよ。
  - 次のページに答えを載せる。

答え

- stopwatch出力の copy un to u ラベルの部分が遅い。ソースコードでは以下の部分。

```
u(1:NGRID,jj%stt:jj%end)=un(1:NGRID,jj%stt:jj%end)
```

  - コンパイルオプションで-O3をつければ速くなる。

```
$ mpifrtpx -O3 heat5.f90
```

演習2

- heat5.f90で最も時間のかかる

  `u(1:NGRID,jj%stt:jj%end)=un(1:NGRID,jj%stt:jj%end)`

  の部分を2重do-loopに展開し、それをheat5b.f90とせよ。
- heat5b.f90を次のコマンドでコンパイル＆実行せよ。

```
$ mpifrtpx heat5b.f90
$ pjsub heat5.sh
```

  – 実行時間はどう変わったか？

実行例

```
##################################################
job start at Tue Jul 15 21:07:29 JST 2015
##################################################
 # myrank= 3 jj%stt & jj%end = 751 1001
 # myrank= 0 jj%stt & jj%end = 1 250
 # myrank= 2 jj%stt & jj%end = 501 750
 # myrank= 1 jj%stt & jj%end = 251 500
 //==============<stop watch>===============\\
        profile 1d:    0.000 sec
         main loop:    8.334 sec
      mpi sendrecv:    0.409 sec
             jacobi:    4.103 sec
      copy un to u:    3.799 sec
  ---------------------------------------
              Total:    8.386 sec
 \\==============<stop watch>===============//
##################################################
job end at Tue Jul 15 21:07:39 JST 2015
```

演習３

- heat5b.f90で時間がかかるのはstopwatch出力のjacobiとcopy un to uラベルの二カ所（どちらも２重do loop）である。この部分をOpenMPでスレッド並列化し、heat6.f90として保存せよ。
  - MPIとOpenMPのハイブリッド並列化！
- ハイブリッド並列化ようのジョブスクリプト（heat6.sh）は用意した。
  - OpenMPスレッド数を1,2,4,8,16と変えて自動投入する。
- コンパイル＆実行

```
$ mpifrtpx -Kopenmp heat6.f90
$ pjsub heat6.sh
```

ジョブスクリプト

- heat6.sh（中心部分）

```
#!/bin/bash
#PJM -N "heat6”
#PJM -L "rscgrp=small”
#PJM -L "node=4”
#PJM -L "elapse=02:00”
#PJM -j
export FLIB_CNTL_BARRIER_ERR=FALSE
.
.
for opn in 1 2 4 8 16
do
export OMP_NUM_THREADS=$opn
echo "# omp_num_threads = " $opn
mpiexec -n 4 ./a.out
done
```

課題

- スケーリングの確認
  1. heat6.f90のジュブキューの指定をsmallからlargeに変更せよ。今日の17時まではlargeキュー占有。
  2. heat6.f90を使い
     a. 1ノードM（≤ 16）スレッドのハイブリッド並列で
     b. N（≤ 84）ノードを使い
     c. スレッド総数P （= M × N） v.s. 計算速度Sのグラフ（Sはstopwatch moduleの出力の”Total”で表示されている秒数の逆数と定義する）を、gnuplotで描け。

【評価基準】
Pの値が大きく、並列化スケーリングが線形に近いものほど高い評価とする。
ヒント：並列化スケーリングが悪い時は、格子点数NGRIDを増やす。

提出方法

- 以下の内容を書いてメールで提出せよ。
  a) 氏名・学籍番号
  b) 使用したNGRIDの値、ノード数N、ノードあたりのスレッド数M（= OMP_NUM_THREADS）の値
  c) gnuplotで描画したグラフのキャプチャ図
- 提出先
  kobeuniv.compra1@gmail.com
  - ファイルフォーマット：PDF（表紙なし・1ページ・圧縮しない）
  - メールタイトル：計算科学演習 レポート
  - 締め切り：8月3日23:59　※受け取ったら数日以内に返信する。

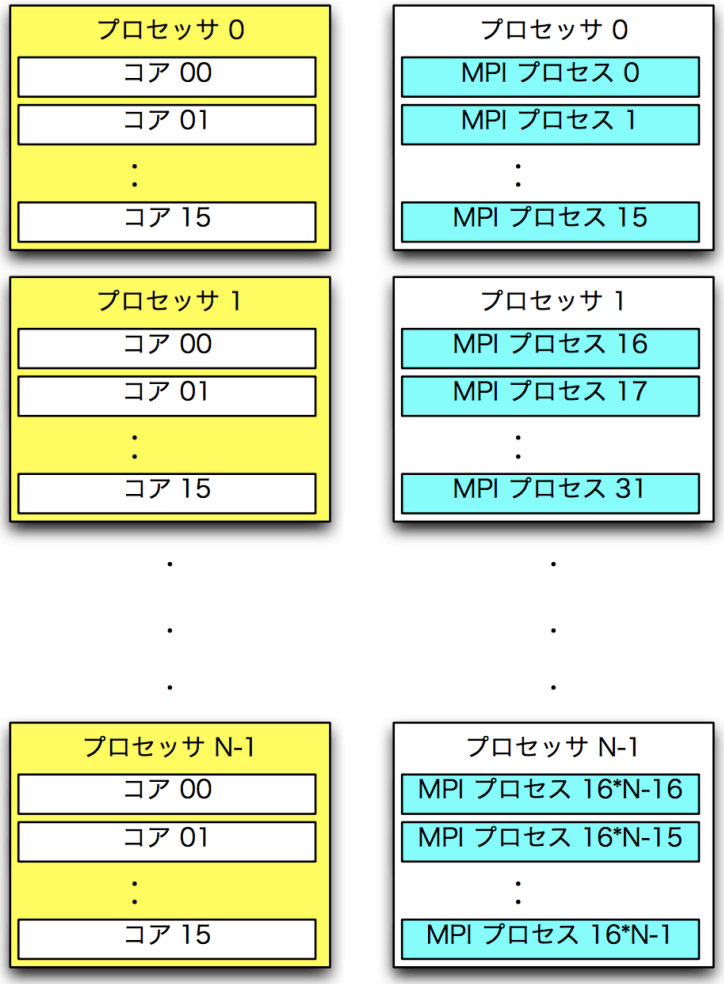
【注意】largeキューを占有して使えるのは今日の17時まで。

補遺A：Flat MPI並列化

Flat MPI並列化

- これまでは1ノード（1プロセッサ、16コア）に一つだけMPIプロセスを動かしていた。
- OpenMPを使ったハイブリッド並列化をしない場合、他の15個のコアは遊んでいた。
- 1ノード（1プロセッサ）に16個のMPIプロセスを走らせることも可能。
- 例えば、4ノード使う場合には、合計4×16=64MPIプロセスで並列化。
- このような並列化をFlat MPIという。OpenMPを使わないのでプログラムはその分簡単になるが、計算速度は一般にはハイブリッド並列化に劣るので推奨しない。

Flat MPI並列化

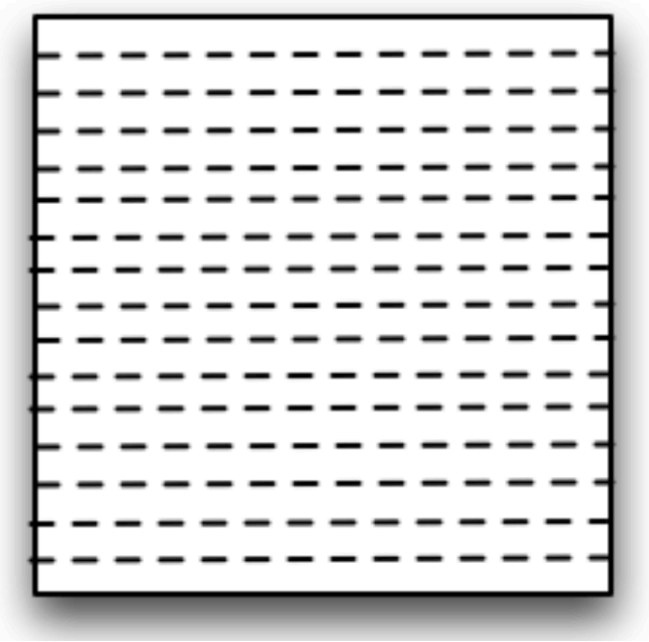

プロセッサ 0
コア 00
コア 01
コア 15
プロセッサ 0
MPI プロセス 0
MPI プロセス 1
MPI プロセス 15
プロセッサ 1
コア 00
コア 01
コア 15
プロセッサ 1
MPI プロセス 16
MPI プロセス 17
MPI プロセス 31
プロセッサ N-1
コア 00
コア 01
コア 15
プロセッサ N-1
MPI プロセス 16*N-16
MPI プロセス 16*N-15
MPI プロセス 16*N-1

補遺B：1次元領域分割と2次元領域分割

補遺C：2次元並列化

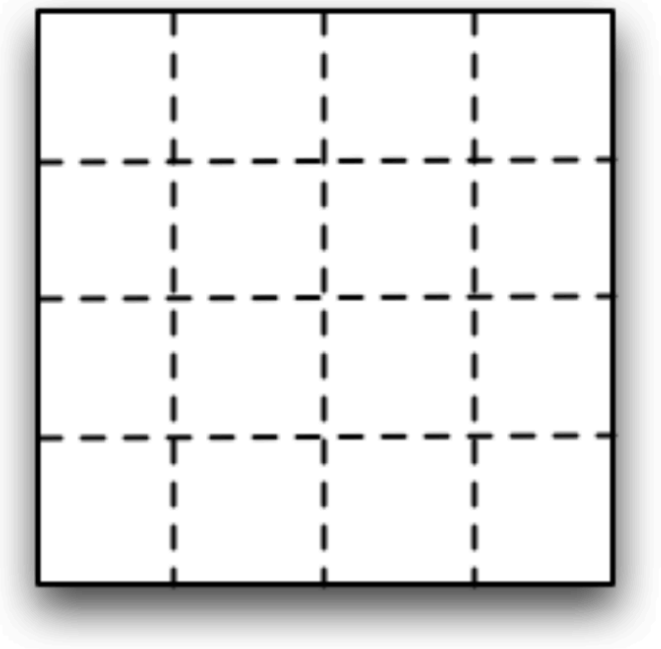
1次元並列化

- 引き続き、正方形領域の熱伝導問題（平衡温度分布）を考える。これまでの並列化：1 次元領域分割による並列化：16 並列



2次元並列化

- これも 16 並列。どちらが速いか?1 次元領域分割と 2 次元領域分 割どちらを採用すべきか?



1次元分割ができない場合

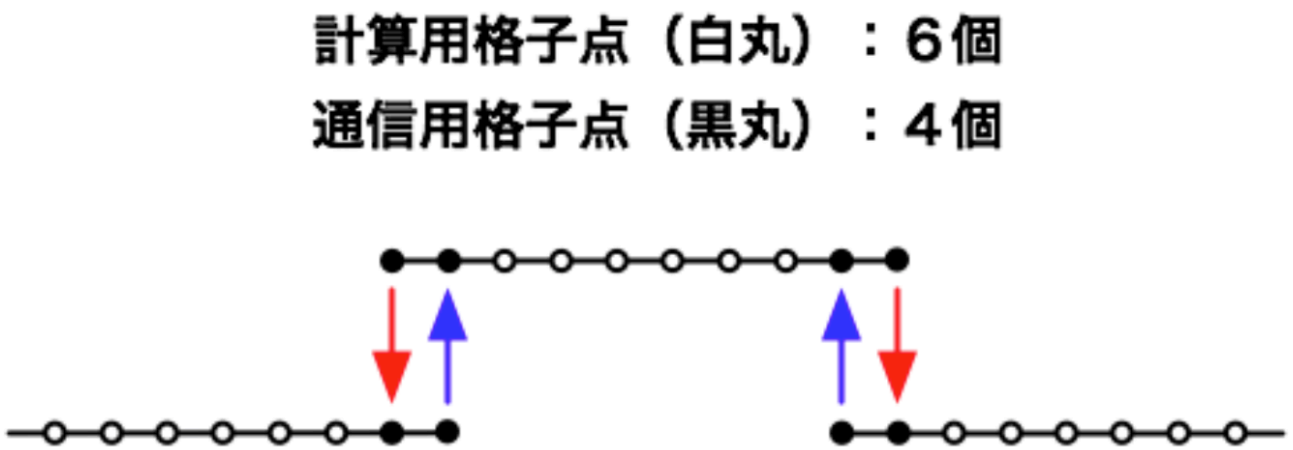
- 格子数NGRID61
- 並列プロセス数100
  - 1次元分割不可能
  - 2次元分割なら可能（総格子点数3721）

1次元分割と2次元分割の違い

- 格子点数 NGRID が十分大きければ 1 次元分割と 2 次元分割は同じか?
  - プロセスあたりの計算量は同じ
  - 通信料が違う

計算と通信

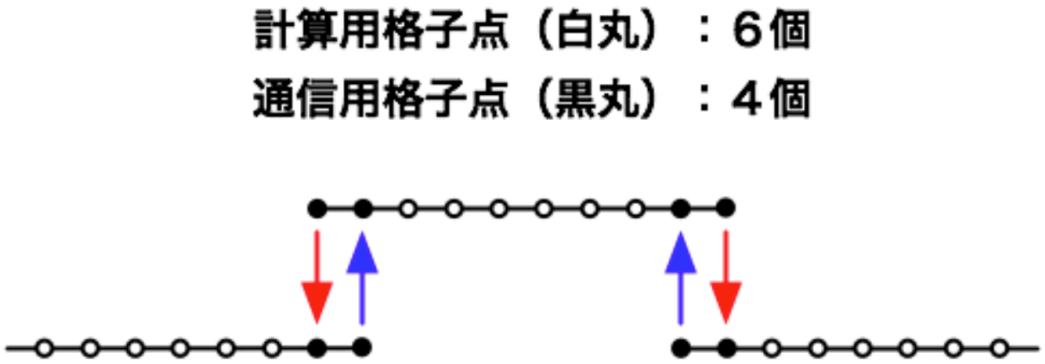
- 1 次元空間を格子点で離散化した上で、MPI でプロセス間通信を行う場合を考える。



計算と通信

- 計算格子には2種類ある。
  - その上で計算だけを行う格子
  - MPI通信のデータを送受信する格子

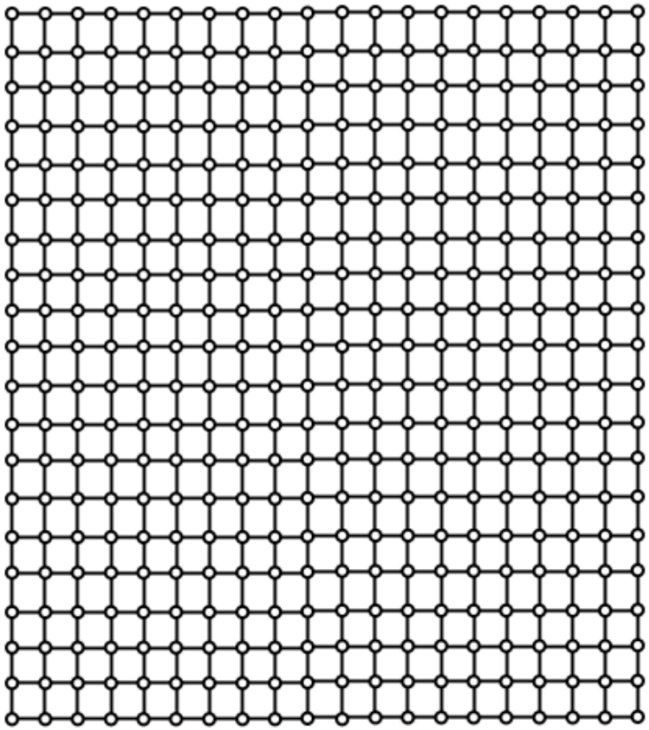
    ※一番外側から2番目の格子は計算も通信も行う



通信は時間がかかるので、通信を行う格子点は少ない方が望ましい。

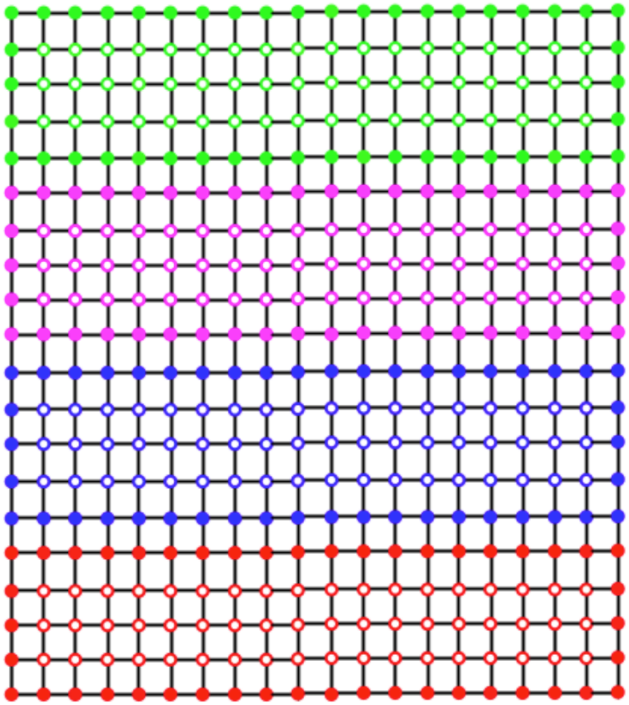
2次元領域分割

- 正方形領域を 400 個の格子点で離散化した場合



2次元領域分割

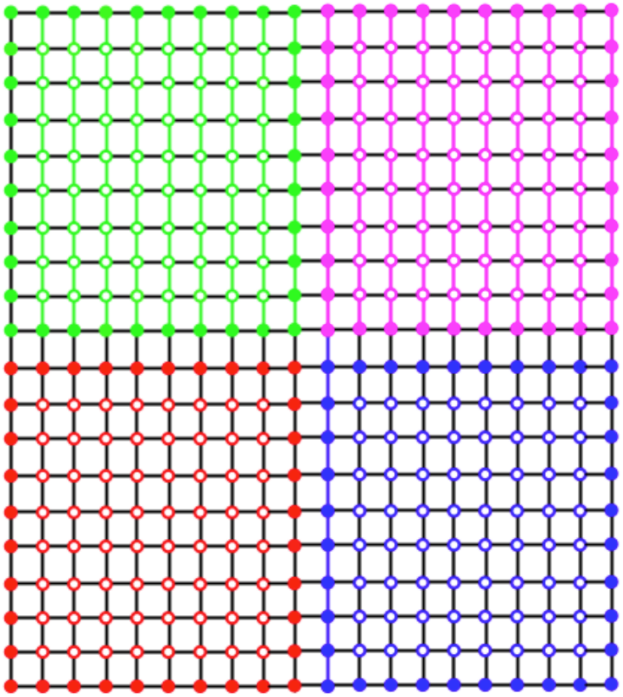
- 4つの MPIプロセスで並列化
  1次元領域分割



赤のプロセスの通信担当格子点の数:46

2次元領域分割

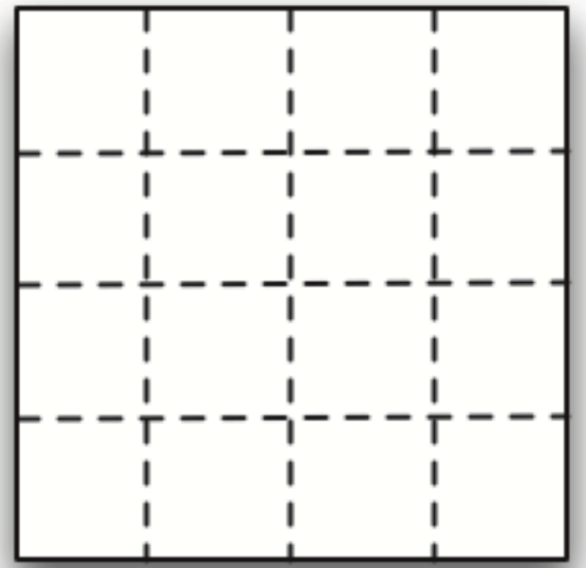
- 2 次元領域分割の場合
  同じく4 つの MPIプロセスで並列化

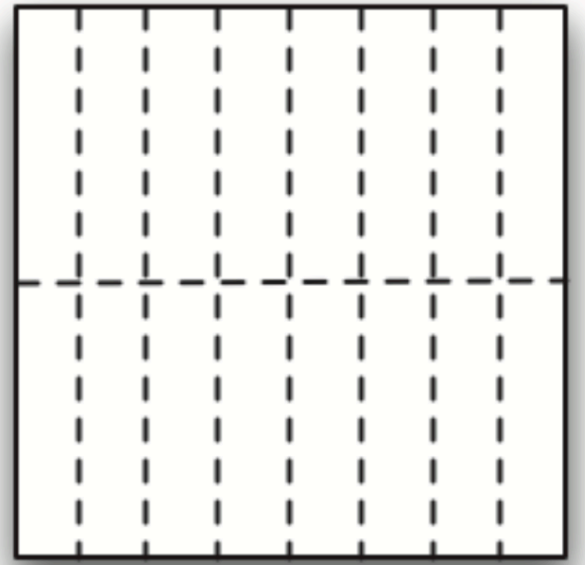


赤のプロセスの通信担当格子点の数:38

2次元領域分割の方法

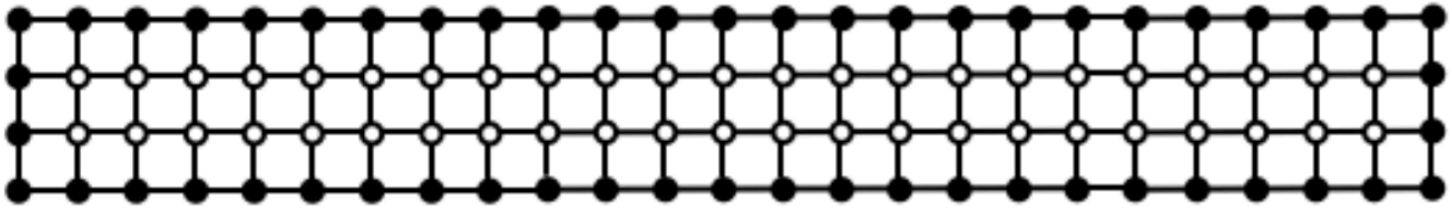
- どちらが良いか？





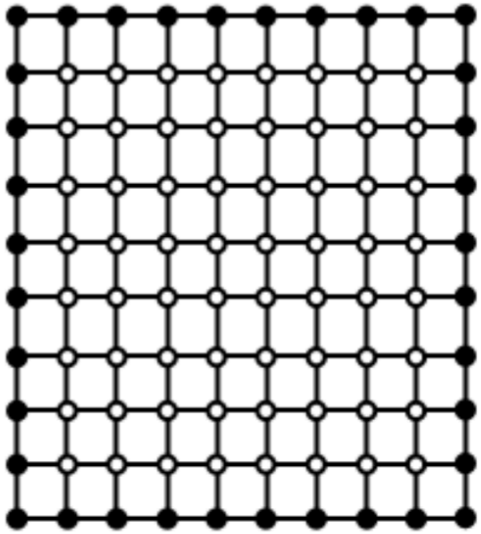
2次元領域分割の方法

計算格子（白丸）：46 個
通信格子（黒丸）：54 個
合計：100 個



2次元領域分割の方法

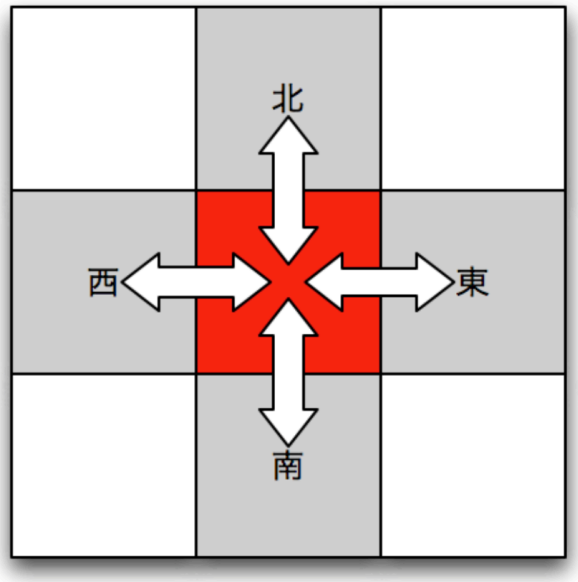
計算格子（白丸）：64 個
通信格子（黒丸）：36 個
合計：100 個



面積の等しい長方形の中で、4辺の長さの合計が最も小さいものは正方形。

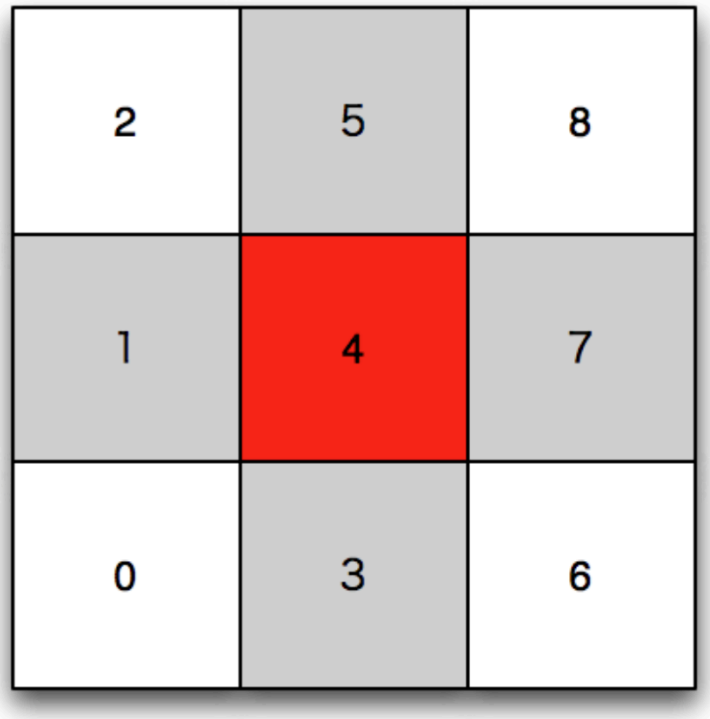
2次元領域分割による並列コード

- 領域分割による並列化を行うときに注意すべき点の一つは、MPIプロセスの配置方法。
- 隣同士の通信がもっとも通信速度的に「近い」位置にプロセスを配置することが望ましい。



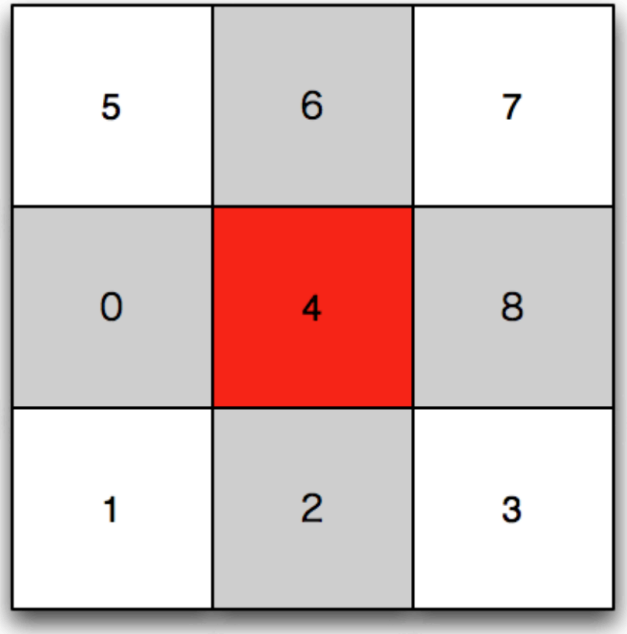
2次元領域分割による並列コード

- 4 番のプロセスはランク番号 1,3,5,7 のプロセスと頻繁に通信する。



2次元領域分割による並列コード

- もしも使用している並列計算機のネットワークの構成上、4 番のプロセスはむしろランク番号 0,2,6,8 のプロセスと通信した方が 速い場合には、以下のようにプロセスを配置する方が望ましい。



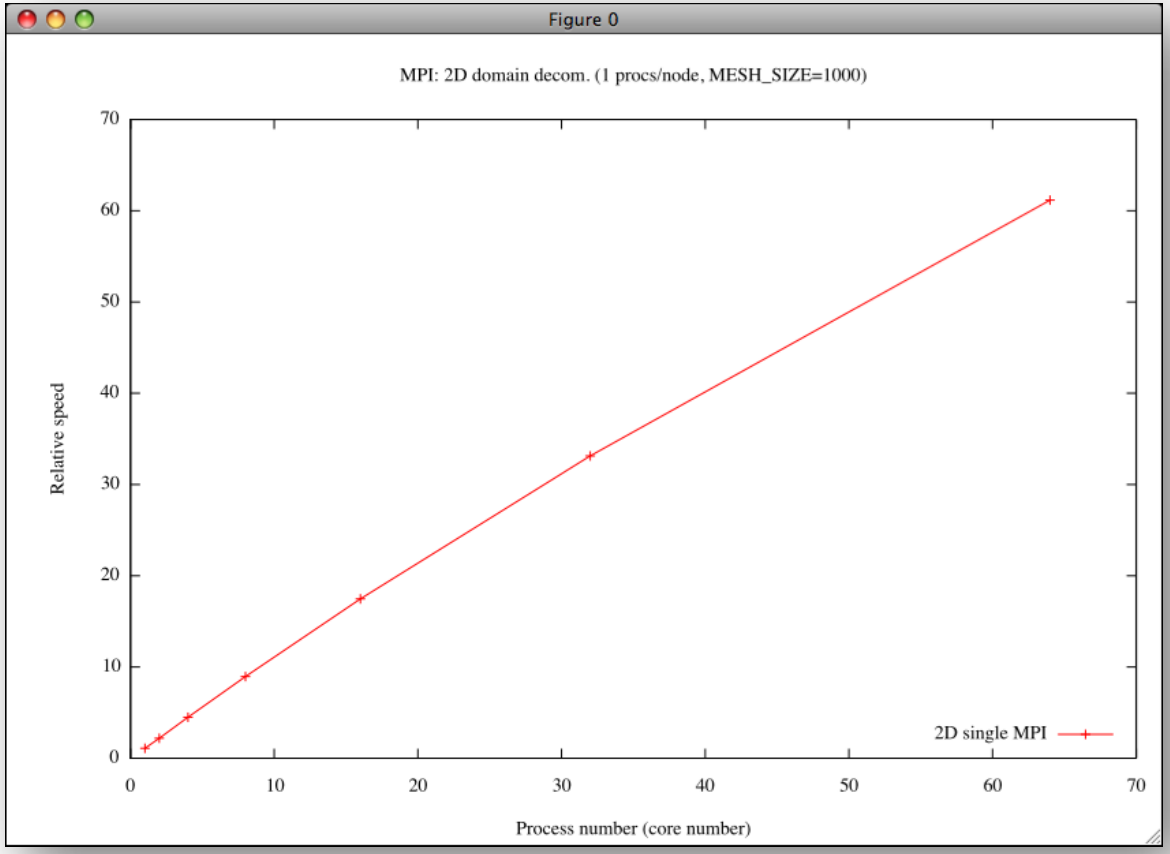
MPI_CART_CREATE

- MPI 関数の一つ MPI CART CREATE を使うと（使用する計算機 がネットワークの通信性能の関する情報を提供している場合に は）通信効率の点で最適な配置でプロセスを自動的に分配してくれる。

速度のスケーリング

- 【参考】
  π-computer とは別のシステムにおける測定結果

MPIプロセス数と計算速度の関係



1次元領域分割と2次元領域分割の比較