総合演習第1ラウンド資料
No.3

滝澤 真一朗

トピック

- 演習の進め方についての補足
- N体問題の課題の説明
- 配列データのMPIプログラムでの取り扱い

プログラム引数読み込みの注意点

- プログラム引数は、MPI_Init実行後に処理すること

良い例

```
MPI_Init(&argc, &argv);
…
loop = atoi(argv[1]);
…
```

悪い例

```
loop = atoi(argv[1]);
…
MPI_Init(&argc, &argv);
```

- 悪い例の場合、すべてのプロセスで値が読み込めるか、MPIの処理系依存
  - ある環境では動く（かもしれない）が、環境が変わると動かなくなるコードのできあがり！

⇒ ポータビリティがない

ファイル転送のこつ (1/2)

- scpのオプションが長いので、ちょっと楽したい
  - TSUBAMEのホームディレクトリに同じ名前でコピー

```
$ scp mpi_pi_mc.c USERNAME@login.cc.titech.ac.jp:
```

  - TSUBAMEのホームディレクトリ以下の別ディレクトリに同じ名前でコピー（ディレクトリはあらかじめ作っておく）

```
$ scp mpi_pi_mc.c USERNAME@login.cc.titech.ac.jp:enshu1
```

  - Mac上のカレントディレクトリ（コマンドを実行したディレクトリ）に、TSUBAME上のファイルを同じ名前でコピー

```
$ scp USERNAME@login.cc.titech.ac.jp:mpi_pi_mc.c .
```

ファイル転送のこつ (2/2)

- 複数のファイルをまとめて1度で送りたい
  - まとめ方

```
$ tar zcvf files.tar.gz FILE1, FILE2, …
```

    - files.tar.gzという名前で、FILE1, FILE2, …をまとめた1つのファイルを作成
    - ディレクトリを与えても良い
  - 展開方法

```
$ tar zxvf files.tar.gz
```

    - files.tar.gzをカレントディレクトリに展開

オプション	意味
z	zip(gzip)圧縮する
f	ファイル名を指定する
c	ファイルを1つにまとめる(compress)
x	ファイルを展開する(extract)
v	進行状況を表示

N1GEのオプション

- ジョブ名の指定 -> 出力ファイル名から出力内容を把握

```
$ n1ge -N JOBNAME PROGRAM PARAM1 PARAM2 …
```

  - JOBNAME.oJOBID (プログラム出力)、JOBNAME.poJOBID (スケジューラ出力) ファイルが出力される
  - ジョブ名を指定しないと、自動的に割り振らる（OTHERS）

- ジョブ実行開始、終了のメール通知

```
$ n1ge -mail ADDRESS PROGRAM PARAM1 PARAM2 …
```

N体（多体、n-body）問題

- N個の質点間における相互作用の力を解くことによって、例えば宇宙空間に散らばる惑星間の引力、物質を構成する分子間の引力のシミュレーション等を行なう。

- 解いてもらう問題は、東工大スーパーコンピューティングコンテスト2001の問題

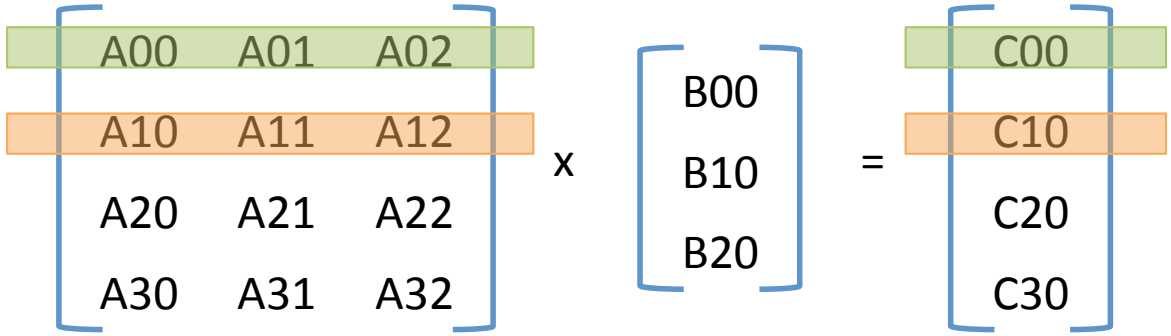
滝澤のサンプル結果

実行時間の推移. 5000点

プロセス数	実行時間(秒)
1	13.86
2	7.81
4	3.79
8	2.52
10	2.12
20	1.28
40	0.88
50	0.74

配列データの取り扱い

- 行列・ベクトル積のMPI並列化



- 行列積の規則

$$Cxy = \Sigma\ (Axk * Bky)$$

- 結果の各要素に依存関係がない
  - 容易にデータ分割して並列化が可能

ただし、単純な行列・ベクトル積は計算量が少なく、並列化する意味が少ない

逐次実装 (コア部分のみ)

M_ROW	行の数
M_COL	列の数

```
long *matrix, *vector, *answer;

matrix = (long*)malloc(M_ROW * M_COL * sizeof(long));
vector = (long*)malloc(M_COL * sizeof(long));
answer = (long*)malloc(M_ROW * sizeof(long));
init_array(matrix, M_ROW * M_COL, 0);
init_array(vector, M_COL, 1);

for (i = 0; i < M_ROW; i++) {
  answer[i] = 0;
  for (j = 0; j < M_COL; j++) {
    answer[i] += matrix[M_COL*i+j] * vector[j];
  }
}
```

行列・ベクトルの初期化

*(matrix + M_COL*i+j)

mv_mul.cとして配布

- MPI並列化を考慮して、matrixは連続メモリ領域として確保
  - 連続メモリ領域として確保しないと、MPIでの通信の実装が困難

並列化のための考察

- answerの計算処理を分割すればよい
  - 各プロセスに M_ROW / #process の数の計算をさせる
- vectorは全プロセスで共通して持たせよう
- matrixは、answer計算で必要な部分のみ、持てば十分
  - 全プロセスでmatrix全体を持たせても良いが、メモリ領域の無駄、通信オーバーヘッドが発生

MPI並列化 – 初期化部分

```
long *matrix, *vector, *answer;
long *each_mat, *each_ans;
int nmyrow;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
nmyrow = M_ROW / nprocs;
```

各プロセスの計算担当数を求める

```
each_mat = (long*)malloc(nmyrow * M_COL * sizeof(long));
vector = (long*)malloc(M_COL * sizeof(long));
each_ans = (long*)malloc(nmyrow * sizeof(long));
```

全プロセスで共通のメモリ領域を確保

```
if (rank == 0) {
  matrix = (long*)malloc(M_ROW * M_COL * sizeof(long));
  answer = (long*)malloc(M_ROW * sizeof(long));
  init_array(matrix, M_ROW * M_COL, rank);
  init_array(vector, M_COL, rank + 1);
}
```

データの生成、計算結果の集計は1つのプロセスが担当

MPI並列化 – 行列・ベクトル配布

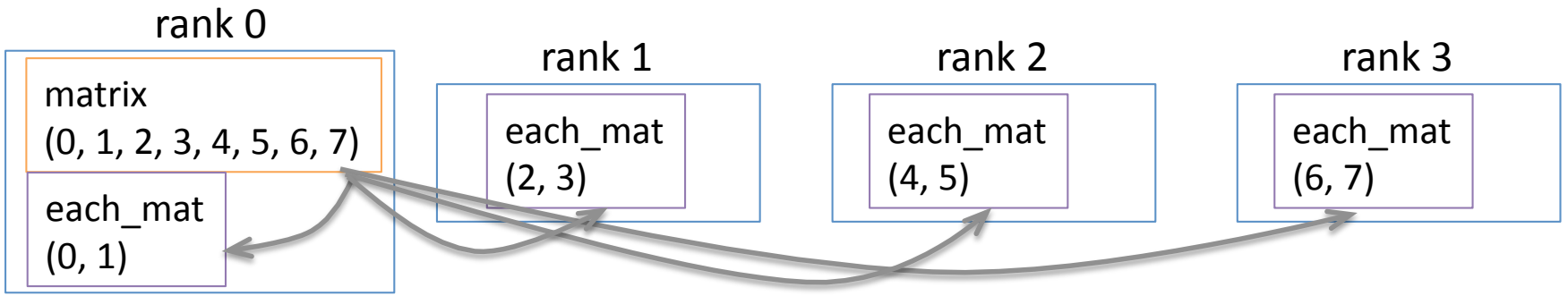
```
MPI_Scatter(matrix,   nmyrow * M_COL, MPI_LONG,
            each_mat, nmyrow * M_COL, MPI_LONG,
            0, MPI_COMM_WORLD);
MPI_Bcast(vector, M_COL, MPI_LONG, 0, MPI_COMM_WORLD);
```

連続データの分割配布のための集団通信関数

```
MPI_Scatter(送信するデータメモリ領域へのポインタ,
            各プロセスに送信するデータ数,
            送信するデータの型,
            受信するメモリ領域へのポインタ,
            受信メモリ領域のサイズ,
            受信するデータ型,
            送信プロセスのランク,
            MPI_COMM_WORLD);
```

全送信データ量でないことに注意！！

これらは等しいことが多い



MPI並列化 - 計算・データ集計

```
for (i = 0; i < nmyrow; i++) {
  each_ans[i] = 0;
  for (j = 0; j < M_COL; j++) {
    each_ans[i] += each_mat[M_COL*i+j] * vector[j];
  }
}
```

部分行列、部分計算結果を保存する変数に書き換え

```
MPI_Gather(each_ans, nmyrow, MPI_LONG,
           answer,   nmyrow, MPI_LONG,
           0, MPI_COMM_WORLD);
```

全受信データ量で
ないことに注意！！

```
MPI_Gather(送信するデータメモリ領域へのポインタ,
           送信メモリ領域のサイズ,
           送信するデータの型,
           受信するメモリ領域へのポインタ,
           各プロセスから受信するデータ量,
           受信するデータの型,
           受信プロセスのランク,
           MPI_COMM_WORLD);
```

完成プログラムは
mpi_mv_mul.cとして配布

課題

- N体問題の逐次実装のプログラムを並列化せよ
  - 締め切り:11/9
  - 途中経過を次回講義までに滝澤にメール報告

- 次回: 11/2 or 10/29
  - 領域分割法、粒子登録法を用いたN体問題の実装の紹介(の予定)

N体問題課題の進め方 (1/2)

- makeを用いたコンパイル
  - コンパイルルール（依存関係）をMakefileに記述し、ルールにしたいがソースコードをコンパイル
  - Makeターゲット

make	逐次、MPIプログラムの両方を生成
make main	逐次プログラム（main）を生成
make mpi_main	MPIプログラム（mpi_main）を生成
make clean	コンパイル生成ファイルを削除

- mpi_main.cを編集すること
  - 最初、main.cとmpi_main.cは同じ内容の、逐次プログラム

N大問大課題の進め方(2/2)

- 実行方法
  - メインプログラムは引数に問題番号を受け取る

```
$ mpirun –np X mpi_main 0
$ mpirun –np X mpi_main 1
$ mpirun –np X mpi_main 2
```

  - 問題番号2は回答が用意されていない

問題番号	粒子数	ステップ数
0	1024	100
1	5000	100
2	20000	200

- レポートには、問題番号1の結果をまとめること
  - プロセス数も1から順に増やして実行すること
  - 問題番号2もまとめると良い