2002 年 5 月 24 日

長谷川秀彦

情報処理演習 II
Parallel Computing on Distributed Memory Machine

1. 放送

前回は、プロセス間でメッセージをやりとりする方法について演習した。今回は、すべてのプロセスに同じ情報を伝えたり、各プロセスの情報を集約してその情報を共有する方法についてふれる。 MPI の場合、プロセス間のメッセージ交換は MPI_Send-MPI_Recv で送り先のプロセス、差し出し元のプロセス、メッセージタイプなどを指定して情報のやりとりするのに対し、プロセス 0 からそれ以外のすべてのプロセスに情報を送るのが MPI_Bcast (broadcast; 放送) である。放送に対して、ふつうのメッセージ交換を「通信」という。 MPI_Bcast は相手を細かく指定することはできないが、システムが提供するメッセージパッシングライブラリに該当プロセスへ効率よく情報を届けるアルゴリズムが組み込まれている。broadcast と同様の機構を用いれば、部分和から総和を計算する MPI_Reduce, 最大値・最小値の計算など各プロセス情報の集約が効率よく実行できる。

MPI_Bcast, MPI_Reduce を用いて数値積分を実行する例が Integral.c である。これは、f120 翫さんのレポートを手直ししたもので、関数 4/(1+x^2) を0から1まで積分して円周率πを得る(本当かな？)。このプログラムでは、 [0, 1] 区間を n 等分し、細かく区切られた区間を一つずつ順に演算を担当するプロセスに割り当てる。このような割り当て方法を cyclic distribution という。実際は、プログラムに仕組みが組み込まれているので、MPI_Bcast で分割数 n だけを全プロセスに伝えればよい。プロセスは与えられた小区間に対して、関数の下側の領域を近似する矩形の面積を加え合わせている。各プロセスでの部分和が計算できたら、 MPI_Reduce を用いて総和を計算する。 bnode, mask, allnds などの使い方に注意してプログラムをながめてほしい。

このプログラムでは、分割数 n を大きくして区間を細かくすればするほど(限度はあるが)正しい値に近づく。並列処理を用いることで、時間をかけることなく高品質の解が得られる。一般の逐次プログラムとは、部分和を取ってから総和をとるという違いがあるので、数値に敏感な計算の場合は注意がいる。とはいっても、部分和をとってから総和をとるほうが、丸め誤差の影響がでにくいので、神経質になることはない。

2. 時間測定

並列化効果を調べるには、最初から最後まで動いているプロセス（多くの場合はプロセス０）がどれくらいかかったかを、壁時計（Wall Clock time）で測定する。あるいは、仕事の開始時点と終了時点に同期（ synchronization ）を入れてその間の実行時間を測定する。複数プロセスが動いているからといって、すべてのプロセスの実行時間を合計するようなことはしない。並列処理ではメッセージの通信時間や待ち時間（オーバーヘッド）が生じるが、 CPU 時間にはそのような影響が間接的にしか反映しない。測定の際は、混雑の影響がでないように十分注意し、少なくとも 2,3 回の測定の平均をとる。

オーバーヘッドのため、個々のプロセスの CPU 時間を合計したものは、逐次プログラムより CPU 時間がかかるはずである。台数を増やしたとき、１台あるいは２台（並列版のプログラムが１台では実行できない場合）に対する時間の短縮率の逆数を並列化効率という。台数が増えたとき、傾きが１の直線にそって並列化効率がよくなるものが理想的な並列プログラムで、並列化効率が線形であるという（傾きが０に近いと恥ずかしくていえない)。

時間計測の例 timing.c はプロセス０から右回りに最終プロセスまでメッセージを送り、プロセス０がそのメッセージを受け取って終了するというリングを N 周繰り返して、隣のプロセスへメッセージを送るのに必要な平均時間を求めている。当然ながら、プロセス数が増えれば１周に必要な時間は長くなる。プログラムからもわかるように、ほとんどがメッセージ待ちの時間になる。

時間計測の際は、 MPI_Wtime を使い、２回の MPI_Wtime 呼び出しをすればその間の時間がわかる。また、時間計測の際には、「入出力を測定しない」ように注意しなければならない。入出力は CPU での処理に比べて 1000 倍以上（もっと？）遅いので、MPI_Wtime の間に入出力があると何を計測したのかわからない結果となる。

3. データ分散方法とプログラミングスタイル、負荷分散

処理時間がかかる例として、２種類の行列積プログラム MatMul1.c, MatMul2.c を作った。わかり易さを優先したため、インターフェースや性能には問題がある。このプログラムでは、 C = AB を計算する際、行列 B をすべてのプロセスに持たせ、 A を横切りにした BLOCK × n の小行列 A_1, A_2, …, A_l を各プロセスに送って計算させ、結果の BLOCK × n の小行列 C_1, C_2, …, C_l をプロセス０に集めている。プロセス間で A と C の連続的な領域が受け渡しできるよう、行ブロックとしてデータを分散させている。このようなデータ分割を block distribution という。高速計算のためにも、よけいなデータ移動を避けるためにも、メモリの連続アクセスが基本である。一方、これまでの例では cyclic

distribution を使った。

二つのプログラムはまったく同じ計算をしているが、 MatMul1.c では

```
if( my_id == 0 ) {          プロセス 0 の処理
}
else {                      その他のプロセスの処理
}
```

が全体を通して１組になっているのに対し、 MatMul2.c では何組も出現している。MatMul1.c ではプロセス０の仕事とその他の仕事という形式で記述されているが、MatMul2.c ではある send に対応した receive という形式で記述されている。後者のやり方のほうがバグを作り込みにくくなるためか、複雑なプログラムでは後者の書き方をすることが多いように思う。どちらのやり方でも全く同じことが記述できるので、作者と自分の考え方を対応づけながらプログラムを読むことが大事である。並列処理では（意味的に）プログラムが複数になる。複数のプログラムを作るとメンテナンスも大変なので、一つのプログラムで複数のプロセス用の機能を実現したSPMD（ Single Program Multiple Data ）形式のプログラムが好まれる。しかし、プログラムが複雑になるなら、 SPMD などやめて別プログラムにしてしまえということだってありである。いずれにせよ、メンテナンスの問題を含めて、並列化の結果はすべてプログラマが責任を負う必要がある。並列化にはじゅうぶんな注意が必要である。

この行列積プログラムは行列サイズ N とプロセス数の組み合わせによって、プロセス１台がまるまる空いてしまったり、極端に仕事が少なくなったりする。このような場合、負荷分散（ load balancing ）に問題があるという。よい並列プログラムは、すべてのプロセスを均等に働かせることと、計算と通信を同時に実行して通信を隠蔽する。この例では、最適なブロックサイズ BLOCK をどう決めるとか、プロセス０がデータの送信・受信ばかりでなく積の計算に加わるとか、データを行ごとに cyclic に送るとか、いろいろと検討の余地がある

本日の演習メニュー

(0) 教材ファイルのコピーを行う

```
clover% cp -rp ~hasegawa/IPP2/May24   .
```

(1) Integral.c をそのまま実行して正しく動作することを確認せよ

(2) √（1 - x^2 ）を 0 から 1 まで積分すると π/4 が得られる。これをプログラムとして実現しなさい

(3) 「関数の下側の領域を近似する矩形の面積を加え合わせる」代わりに「近似する台形の面積を加え合わせる」ようなプログラムを作りなさい

(4) 被積分関数、分割数、矩形か台形かなどを変えて、結果がどう変わるかを調べなさい。πの正確な値はインターネットで探そう（時間に余裕があるときに)。

(5) 条件を変えて timing.c を実行せよ

(6) プロセス 0 で printf を実行し、どのくらい遅くなるかを調べなさい

(7) 前回の Binary Tree の例を（無意味だけど）繰り返し、その間の実行時間を２種類の方法で測定しなさい

これ以上、新たな話題は持ち出さないつもりなので、これまでのプリントをゆっくりと理解してほしい。問題に出くわしたとき、知人に聞いてもいいし、本やマニュアルを調べるのもいいだろう。 211 研究室を訪れるもよし、メールで聞くもよしであるが、時間に余裕をもって、色々と試行錯誤することが必要になる。ゆっくり考えれば、必ずできる！

なお、プログラムはプロセス数に関係なく正しい結果が得られることが必要だが、時として無理なこともある。なぜうまくいかないのかをよく考え、論理的にものごとを説明する習慣をつけよう。エラーチェックなどは致命的でなければ、今回のレポートでは多少省いてもよい。並列プログラムではデッドロックや、データの待ち続けなどが起こりやすいので、じゅうぶんな注意がいる。あと、数週間楽しんで！

参考文献

1) 湯浅太一、安村道晃、中田登志之編．はじめての並列処理，共立出版， 1998, 244p. bit 別冊

2) William Gropp, Ewing Lusk, and Anthony Skjellum. Using MPI, Second Edition, The MIT Press, 1999

3) Peter S. Pacheco. Parallel Programming with MPI, Morgan Kaufmann Publishers, 1997(実は日本語訳がでている)

4) Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. MPI: The Complete Reference, The MIT Press, 1996

まとめの課題

＊＊　ご希望に応じて、問題候補を考えますが、とりあえず　＊＊

課題 A の並列処理として、(1) - (5) のいずれかを 6 月 21 日までに完成させ、関連するリスト、結果などを A4 版縦サイズのレポートにまとめて提出しなさい。実験レポートの観点から、目的、経過、結果、考察などは必ず含め、図表を活用すること。ソースプログラムにはコメント・インデントをつけ、データやプログラムのファイル名もきちんとまとめておくこと。質問・ヒントは hasegawa@ulis.ac.jp まで。

(1) 計算時間が多くかかるアプリケーションを文献などから探し出し、そのプログラムの実行時間を短縮するような並列化プログラムを作成しなさい。なお、 clover では思うような性能がでなくともよい。例えば、行列計算、ソート、グラフ、最適化などに大規模問題がある。

(2) 行列積の計算 MatMul1.c と MatMul2.c は時として正しく動作しない。原因はメッセージの到着順所が予想と異なるためである。プロセス数と n を変えても動くようプログラムを改善しなさい。

(3) 行列積プログラム MatMul1.c と MatMul2.c のデータ分散方法などを変更し、より高性能なプログラムに改良しなさい。また、プログラムの読みやすさ（書き易さ？）が性能がどのように変化するかを考察しなさい。

(4) 最初に作った単体版（ CPU 1 台用）のプログラムを N が大きくなっても、性能が落ちず、概ね 300MFLOPS 以上をコンスタントにだせるように改善しなさい。

(5) メッセージパッシングに必要な時間を測定・評価しなさい。
- 小さいメッセージを何回も送るより、大きなメッセージを 1 回送るほうが高速？
- MPI_Send-MPI_Recv で実現した放送や集約とシステム提供のプログラムでは？
- 通信時間は「通信の起動時間」＋「データ量に比例した時間」になっている？