卒業研究報告書

題目

MPI を用いた並列計算

指 導 教 員 石水 隆 助教

報告者

04-1-47-088

延山 周平

近畿大学理工学部情報学科

平成 19 年 2 月 4 日提出

概要

現在、様々な分野で計算処理の高速化が求められている。高速処理を行うためには、複数のプロセッサを持つ並列計算機(Parallel Computer)[6]が用いられる。しかし、一般に並列計算機は非常に高価であり、容易に用いることはできない。そこで、複数の計算機をネットワーク接続して 1 台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が現在重視されている。

本研究は、一台のコンピュータ上で MPI(Message Passing Interface)[1][2]を用い、擬似的に並列計算プログラムを動かすことによって、仮想並列計算の有用性を示すことを目的とする。

MPI とは **Message Passing Interface** の略であり、分散メモリ型並列計算機(Distributed Memory Parallel Computer)において、複数のプロセッサ間で、データのやりとりをするために用いる、メッセージ通信操作の**仕様標準**である。

本研究では、MPI の実装には、**MPICH**[5]を用いた。これは、MPI 規格を実装したアルゴンヌ国際研究所[4]より配布されているフリーのライブラリ群である。

今回使用した環境としては、LINUXOS の PC 一台で、MPI を用い、仮想的な並列計算環境を構築し、検証した。これにより、並列計算において最もネックである通信時間は、最小に近い値になる。MPI の仕様言語としては、C や Fortran がよく用いられるが、今回は慣れ親しんだ **Java** を使うため、**mpiJava**[3]を用いた。

本研究では、MPI の性能を検証するためにシンプルな加算プログラムを作り、MPI 上での実行時間を測定することにより MPI の有用性を実験的に検証した。これは、1 から 10^n（n=2,3,4,5）までの和を求めるプログラムである。また、プロセッサ数は 4 台と仮定している。

目次

第1章 序論

1.1. 仮想並列計算機(Parallel Virtual Computer)

大容量の記憶デバイスの登場や、ネットワークの高速化などにより、近年、大量のデータを高速に処理することが求められている。取り扱われる情報の量は日々増大しており、その処理時間を短縮することは計算機を使用する上での重大な課題である。データの高速処理には、複数のプロセッサを持つ並列計算機が必要とされる。しかし一般に並列計算機は非常に高価であるため容易に利用できない。そこで現在、複数の計算機をネットワーク接続して接続された計算機全体を仮想計算機(Virtual Machine)として用いるクラスタ処理(Cluster Computing)[7]やより大きな規模のネットワークで用いられるグリッド処理（Grid Computing）[7]といった手法が注目されている。仮想並列計算機を構築するソフトウェアの中には無償で提供されているものもあるため、安価に並列計算環境を構築することが可能である。代表的な仮想並列計算環境を構築するソフトウェアとしては、PVM(Parallel Virtual Machine)[8][9]や MPI(Message Passing Interface)[1][2]などが存在する。

1.2. PVM (Parallel Virtual Machine)

PVM(Parallel Virtual Machine)[8][9]は、1991 年に米国のオークリッジ国立研究所(Oak Ride National Laboratory)[10]を中心に異機種間の分散処理が目的に開発された、メッセージパッシングによる並列処理を行うための並列化ライブラリである。

PVM はワークステーションクラスタなため、TCP/IP の通信ライブラリで一般的に使用されている LAN 環境があれば並列処理が実行出来るので多くのユーザが利用している。また、異機種間の通信も考慮されているため、対応する計算機は家庭にあるパーソナルコンピュータからスーパーコンピュータなど多くの種類で PVM による並列処理が出来る。

PVM の問題点として、PVM は多くの並列計算機に移植されるようになったとき、各並列計算機ベンダが独自にチューニングを行なった PVM を開発してしまい。PVM で作成をしたプログラムの移植性が乏しくなってしまったことが挙げられる。

1.3. MPI (Message Passing Interface)

MPI (Message Passing Interface)[1][2]は分散メモリ型並列計算機(Distributed Memory Parallel Computer)において、複数のプロセッサ間で、データのやりとりをするために用いる、メッセージ通信操作の仕様標準である。1990 年の初頭に、それまでベンダ独自で行っていたメッセージパッシングによる通信の仕組みを共有化することを目的に MPI の規格作成が開始され、1994 年に MPI のヴァージョン 1.0 がまとめられた。翌年 1995 年には新しい機能の拡張を考慮した MPI-2 が検討され、1997 年に規格がまとめられた。

MPI による仮想並列計算環境においての通信は TCP/IP などのネットワークを用いて行われる。仮想並列計算機を構成する各計算機はアーキテクチャにより通信方法が異なり、それに伴い実装も異なる。そのため、ユーザが通信方式の差異等を気にせずにすむように MPI では「MPI ライブラリ」が用意されている。

MPI は専用の並列計算機からワークステーション、パーソナルコンピュータに至るまで幅広くサポートしている、無料で提供されている主な実装は MPICH[5]や LAM[11] [12]といったものがある。

MPI のサポートするプログラミング言語は多く、C 言語や Fortran そして最近では Java[3]などに対応している。

1.4. PVM と MPI の差異

MPI は規格を共有化することを目的としているため、PVM よりも移植性が優れている。しかし、PVM は異機種間での並列計算が可能であるのに対し、MPI は高いレベルのバッファ操作が可能であり高速にメッセージの受け渡しが可能な反面、異機種間の並列処理ができないことが欠点である。

現在では、高速なメッセージ処理および移植性の高さから、MPI の方が主流となりつつある。

1.5. 本研究の目的

本研究では、無料提供されている仮想並列計算環境を構築するソフトウェアの一つである MPICH2[5]を用いて簡単な加算演算を行い、その性能を実験的に評価することにより、MPI による仮想並列計算の有用性を検証する。MPICH2 はアルゴンヌ国際研究所[4]において無料提供されているソフトウェアであり、同研究所の MPICH2 のページ[5]からダウンロードすることにより容易に使用することが可能である。

1.6. 本報告書の構成

第 2 章節以降では、まず 2.1 で MPI の実装環境を示す。次に 2.2 で使用したプログラムの説明をしている。第 3 章では、実行結果を表にし、それを元に考察をしている。

次に 0 節で結論と今後の課題について述べている。それ以降は、謝辞、参考文献、実行したプログラムとその実行結果になっている。

第2章研究内容

2.1. MPI の実装

本研究を始めるにあたり、まず MPI による仮想並列計算環境を構築する必要がある。

本研究では、その実装には MPICH[5]を用いた。これは、MPI 規格を実装したフリーのライブラリー群であり、MPICH のホームページ[5]よりパッケージをダウンロードし、インストールを行うことができる。

現時点では、MPICH を記述する言語は C や Fortran が多く用いられている。しかし、本研究では MPI の記述言語として JAVA を用いるために更に mpiJava[3]を導入した。

MPICH は、Windows、LINUX 共に実装することが出来る。しかし、mpiJava は、現在のところ LINUX ベースの OS にしか対応していない。よって本研究では、LINUXOS の PC を用いた。

2.2. 検証用プログラム

本研究では、MPI の性能を検証するためにシンプルな加算プログラムを用いた。これは、1 から 10^n（n=2,3,4,5）までの和を一万回求めるプログラムである。また、プロセッサ数は 4 台と仮定している。付録に本研究で用いた加算プログラムを示す。

本研究で用いたプログラムは、逐次的に加算していく部分と、擬似並列的に加算していく部分とに分かれる。その二つの実行時間を比較し、並列の有用性を検討する。

擬似並列の部分においては、同性能のプロセッサ 4 台を想定している。故に 1 から 10^n までを 4 つに分け、一台ずつに計算を負担させるため、プロセッサ 2～4 の計算時間を最後に引くという方法を取っている。また、通信時間も一回分になるようにしている。

第3章 結果・考察

本研究では、MPI を用いた加算演算の並列実行時間と、プロセッサ 1 台による逐次実行時間の実行時間差から、MP による仮想並列計算の有用性の検証を行った。

本研究で用いた MPI による加算プログラムの実行結果を表 1 に示す。並列の実行時間とは、並列プログラムの実際の実行時間から並列動作部分の時間を引いたものである。これにより、仮想的ではあるが、並列プログラムの実行時間を出すことができる。最大整数が 100 から 1000 までの間は、実行時間において、逐次プログラムよりも遅かった並列プログラムであるが、1000 を超えてからは速くなっている。また、数値の伸びから、更に数を増やすことにより、差は開いていくと容易に推測できる。

表 1 MPI による和演算の実行時間と逐次時間との比較(秒)

	1~100	1~1000	1~10000	1~100000
逐次の 実行時間	0.019	0.159	1.52	15.191
並列の 実際の実行時間	0.878	0.803	2.188	16.289
並列に 動作している時間	0.2	0.281	1.321	11.689
並列の 実行時間	0.678	0.522	0.867	4.6

第4章 結論・今後の課題

本研究では、MPI を用いた加算演算の並列実行時間と、プロセッサ 1 台による逐次実行時間の実行時間差から、MP による仮想並列計算の有用性の検証を行った。

本研究で用いたプログラムにおいては、4 台の計算機を用いて仮想並列計算を行った場合、実行時間は、問題のサイズが小さい(データ数 1000 以下)の範囲では計算機 1 台による逐次での実行時間よりも遅くなった。しかし、問題のサイズが充分に大きい(データ数 10000 以上)の場合の実行時間は、逐次の実行時間の 4 分の 1 に近づいていった。従って、問題のサイズが大きい場合は MPI による仮想並列計算は有用性が高いと言える。MPI はネットワークを通じて計算機間で通信を行うため、どうしても通信時間を取られてしまう。つまり、通信回数と一台のプロセッサの計算の負担率、これが大きなトレードオフになってくる。故にプロセッサの性能、通信速度、これらを考慮した設計をする必要がある。本研究の結果が示すように、MPI による仮想並列計算の有用性は通信に対して計算の負担が大きくなる場合、すなわちデータ数が多いときにより顕著であると言える。

近年、通信環境やプロセッサの性能の向上は著しい。また PC の世代交代も早いため、複数台持つ家庭も増えてくると考えられる。そういった資源を使用するという意味でも、仮想並列計算の重要性は年々増加してゆくと思われる。つまり、アルゴリズムや MPI を実装する環境作りも、これからの重要な課題である。

謝辞

拙い小論の成るについても多くの方々の恩顧が思われますが、特に石水隆先生には時に厳しく、時に優しく、いつもあたたかくご指導ご鞭撻を賜り、心からの感謝の念を表して謝辞とさせていただきます。

参考文献

[1] P.パチェコ 著,秋葉博 訳：MPI 並列プログラミング, 培風館（2001）
[2] 渡邉真也 著：MPI による並列プログラミングの基礎, 培風館（2001）
[3] 梶原 広輝, 廣安 知之, 三木 光範：mpiJava の利用法範, http://mikilab.doshisha.ac.jp/dia/research/report/2004/0809/005/report20040809005.html
[4] Argonne National Laboratory, http://www.mcs.anl.gov/index.php
[5] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/
[6] J.JáJá：An Introduction to Parallel Algorithms ,Addison Wesley(1992)
[7] 日本アイ・ビー・エム システムズ・エンジニアリング株式会社 著：グリッド・コンピューティングとは何か, ソフトバンクパブリッシング株式会社(2004)
[8] PVM, Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/
[9] PVM, http://erpc1.naruto-u.ac.jp/~geant4/pvm/pvm.html
[10] OAK RIDGE National Laboratory, http://ww.ornl.gov/
[11] LAM / MPI Parallel Computing, http://www.lam-mpi.org/
[12] 船山正樹他 訳, オハイオスーパーコンピュータセンタ著：MPI 入門 / LAM による開発, http://phase.hpcc.jp/phase/mpi-j/mpiprimj.pdf

付録

本研究で用いたプログラムと実行結果を以下に示す。

付録1　　本研究で作成したプログラム

```
/*  一台のプロセッサを使い擬似的に                                    */
/*   1 から 10 の n 乗までの整数を                                    */
/*  遂次と並列に足していき、その結果を比較するプログラム              */
/*    n は 2,3,4,5 の値を取る。                                       */
/*  並列計算機のプロセッサ数は 4 台を想定している。(仮想的に)         */

import mpi.*; //mpi パッケージのインポート

public class Add{
    public static void main(String args[]) throws MPIException{

        /*MPI による初期化*/
        MPI.Init(args);

        /*  変数  */
        //int rank; //現在のプロセッサのランク(ID)
        //int size; //プロセッサの総数
        int[] tmp = new int[1];//要素数 1 の int 型配列、データの転送に使用
        int sum =0; //合計
        final int maxSize=1000000;//整数の限界(最大値、定数)
        int maxNum=100;//合計する整数のうち最大の数
        int[] data = new int[maxSize];//合計する整数が入る配列

        tmp[0]=0; //初期化
        double btime=0 ;//計算前の時間
        double time=0; //計算後の時間
        double pbtime=0; //並列計算前の時間  **これらは、仮想的に並列計算している時間を計算し
        double ptime=0; //並列計算後の時間    **time から引くために使う

        /*これらは、今回のプログラムでは使用しない。
          //ランクを得る
          rank = MPI.COMM_WORLD.Rank();
          //サイズを得る
          size = MPI.COMM_WORLD.Size();
        */

        //データの作成。
        for(int i = 1;i<=maxSize;i++){
            data[i-1]=i;
        }

        for(maxNum=100;maxNum<=maxSize;maxNum*=10){ //最大数を 10 倍していく
            System.out.println("=========================================");
```

```
System.out.println("1~"+maxNum);

/****************************************/
/*              遂次計算部分                */
/****************************************/

System.out.println("Normal loop Start ");
btime = System.currentTimeMillis();//計算前の時間取得

for(int j =0; j <=10000;j++){ //一万回繰り返し、誤差を少なくする
    sum=0;//合計の初期化
    for(int i=1;i<=4;i++){//4 回のループは後の並列のプログラムに近づけるためのもの
        tmp[0]=0;
        /*1~25*n 26~50*n 51~75*n 76~100*n の足し算  */
        for(int k=(i-1)*maxNum/4+1;k<=(i)*maxNum/4;k++){
            tmp[0] += data[k-1];
        }
        sum +=tmp[0];
    }
 }

time = System.currentTimeMillis();//計算後の時間取得

System.out.print("    Normal loop is finished. sum = "+ sum);
//時間を、秒になおし出力。
System.out.println(" time：" + (time - btime)/1000 + "seconds");


/****************************************/
/*              仮想並列部分                */
/****************************************/
System.out.println("Parallel loop Start");
ptime =0;
btime = System.currentTimeMillis();//計算前の時間取得

for(int j =0; j <=10000;j++){//一万回の繰り返しのためのループ
    sum=0;
    pbtime=System.currentTimeMillis();//並列計算前の時間取得
    for(int i=1;i <= 4;i++){
        tmp[0]=0;

        /*1~25*n 26~50*n 51~75*n 76~100*n の足し算  */
        for(int k=(i-1)*maxNum/4+1;k<=(i)*maxNum/4;k++){
            tmp[0] += data[k-1];
        }

        if(i==3){ //4 回目の計算+送信の一回分、時間取得。
            ptime+=(System.currentTimeMillis()-pbtime);
```

```
                    }

                    if(i !=4 ){//4 番目以外は、データを送信
                        MPI.COMM_WORLD.Send(tmp,0,1,MPI.INT,0,0);
                    }else{
                        sum+=tmp[0];
                    }
                }

                /*データを受け取り、sum に足していく*/
                MPI.COMM_WORLD.Recv(tmp,0,1,MPI.INT,0,0);
                sum+=tmp[0];
                MPI.COMM_WORLD.Recv(tmp,0,1,MPI.INT,0,0);
                sum+=tmp[0];
                MPI.COMM_WORLD.Recv(tmp,0,1,MPI.INT,0,0);
                sum+=tmp[0];

            }
            time = System.currentTimeMillis(); //計算後の時間取得
            System.out.print("Parallel loop is finished. sum = "+ sum);
            //実際かかった時間を出力

            System.out.println(" time : " + (time - btime)/1000 + "seconds");
            //仮想的な並列時間とそれを実際の時間から引いた時間を出力。
            System.out.println("Parallel time : " + ptime /1000+"seconds    Time - Parallel Time : " + (time
- btime - ptime)/1000+"seconds");
        }
    }
}
```

付録2　　実行結果.

```
1~100
Normal loop Start
  Normal loop is finished. sum = 5050 time : 0.019seconds
Parallel loop Start
Parallel loop is finished. sum = 5050 time : 0.878seconds
Parallel time : 0.2seconds    Time - Parallel Time : 0.678seconds
========================================
1~1000
Normal loop Start
  Normal loop is finished. sum = 500500 time : 0.159seconds
Parallel loop Start
Parallel loop is finished. sum = 500500 time : 0.803seconds
Parallel time : 0.281seconds    Time - Parallel Time : 0.522seconds
========================================
1~10000
Normal loop Start
```

```
   Normal loop is finished. sum = 50005000 time : 1.52seconds
Parallel loop Start
Parallel loop is finished. sum = 50005000 time : 2.188seconds
Parallel time : 1.321seconds    Time - Parallel Time : 0.867seconds
========================================
1~100000
Normal loop Start
   Normal loop is finished. sum = 705082704 time : 15.191seconds
Parallel loop Start
Parallel loop is finished. sum = 705082704 time : 16.289seconds
Parallel time : 11.689seconds    Time - Parallel Time : 4.6seconds
========================================
1~1000000
Normal loop Start
   Normal loop is finished. sum = 1784293664 time : 154.62seconds
Parallel loop Start
Parallel loop is finished. sum = 1784293664 time : 156.94seconds
Parallel time : 116.975seconds    Time - Parallel Time : 39.965seconds
```