卒業研究報告書

題目

MPI を用いた並列計算処理

指 導 教 員

石水 隆 助教

報告者

06-1-037-0041

福崎伸哉

近畿大学理工学部情報学科

平成 22 年 2 月 5 日提出

概要

近年、高速インターネット回線の普及や大容量の記憶デバイスの登場により、膨大なデータをより速く処理し、短時間で解くことが必要となっており、１台のプロセッサからなる逐次計算機を用いた処理では、非常に大きな時間がかかってしまう。

この問題を解決するために、プロセッサの性能を向上させる方法と、複数のプロセッサを用いて並列処理が挙げられる。しかし、処理速度の向上には限界があり、また、多くの資金や時間がかかってしまうため、複数のプロセッサを用いた並列計算機(Parallel Computer)による並列処理が注目されている。しかし、一般的に並列計算機は高価なものであるため、本研究では複数の計算機をネットワークに接続し、仮想並列計算を行うソフトウェアの一つである MPI(Message Passing Interface) [1][2][5]を用いて、どれほどの処理時間の短縮を行うことが出来るかの検証を行った。

目次

1 序論

1.1 仮想並列計算(Parallel Virtual Computing)

近年、高速インターネット回線の普及や大容量の記憶デバイスの登場により、扱うデータ量は日々増大しており、計算の効率化は重要な課題の一つとなっている。この問題を解決するためには、並列計算機能を持つスーパーコンピュータを利用する方法も挙げられる。しかし、一般ユーザにとって価格、プログラミング技術の点で容易に利用することが出来ないため、無料で利用可能な仮想並列計算(Parallel Virtual Computing)の方が一般的に利用し易いと考えられる。仮想並列計算とは、ネットワーク接続された個々のコンピュータをネットワークを通して結合し、仮想的な一つのシステムとみなし、そこで大きな計算を並列的に実行させるという方法である。このようなシステム構築のための提案として、MPI(Message Passing Interface)[1][2][5]や、PVM(Parallel Virtual Machine) [6][7]などが挙げられる。

本研究では、無料で提供されている MPI を利用し、基本問題の一つである行列積演算を行い、その性能を実験的に検証する。

1.2 PVM(Parallel Virtual Machine)

PVM(Parallel Virtual Machine) [6][7]は、1991 年にアメリカの Oak Ridge 国立研究所[8]によって無料で提供されているフリーソフトウェアであり、異機種間の分散処理を目的として開発された、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 ネットワークで接続された複数台のコンピュータを、仮想的に一台のマシンであると捉え、並列プログラムを実行できる環境を提供するソフトウェアである。そのため、LAN 環境があれば並列処理を実行することが出来るので、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と、様々な種類で並列処理を行うことが出来る。また、PVM は並列計算中に仮想並列計算機を構成する計算機のうち 1 台が故障し、停止してしまった場合、その計算機を仮想並列計算機内から迅速に削除されるようになっているため、耐故障性に優れている。

しかし、問題点として挙げられるのが、移植性の低さである。各並列計算機ベンダが独自にチューニングした PVM 開発を行ったため、PVM で作成したプログラムの移植性が乏しくなってしまったのである。

1.3 MPI(Message Parallel Interface)

MPI(Message Passing Interface) [1][2][5]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行うために用いる、メッセージ通信操作の仕様標準である。1990 年初め、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とし 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。その翌年、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のである。

MPI による通信は、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違いなどを気にせず行えるよう、MPI では「MPI ライブラリ」が用意されている。

また、MPI は PVM より後に設計されたため、PVM の問題点を改善するように作られている点が多い。MPI 初期の段階では、プロセス管理の機能において、PVM のようにアプリケーションの中から動的にプロセスの生成を行ったり、停止したりすることは出来なかった。しかし、MPI の新規格である MPI-2 ではそれが取り入れられている。さらに、MPI は Web や書籍などで情報を手に入れやすいため、一般的に利用し易いと言える。

1.4 PVM と MPI の違い

先にも述べたように、MPI は PVM の問題点を学んで作られている点が多い。そのため、PVM と比べると、MPI は高レベルなバッファ操作が可能であり、高速なメッセージの受け渡しが可能である。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 ではそれが不可能であるのが欠点である。

今現在では、複数の信頼性の高いライブラリの存在、MPI フォーラムの存在、移植性の良さという点で、MPI の方が主流になってきている。

1.5 本研究の目的

MPI は移植性が高く、現在主流になってきており、研究が盛んに行われている。そのため、本研究では MPI を用いて仮想並列計算環境を構成する。本研究では、仮想並列計算環境を構成するソフトウェアの一つである MPICH2[4][5]を用いて行列積演算を行い、仮想並列計算における有用性を実験的に評価する。評価方法は、計算機 1 台による逐次計算で行った場合と、計算機 4 台による仮想並列計算で行った場合で比べ、どの程度時間短縮を行うことが出来たかを検証する。

1.6 本報告書の構成

第 2 章節以降では、2.1 で本研究において使用した機器について示す。次に、2.2 で使用した MPI のソフトウェアのインストール方法と環境設定の説明。2.3 で Visual C++[9]のインストール方法と環境設定の説明。2.4 で本研究において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、それを基に考察を述べている。

第 4 章では、結論と今後の課題について述べており、それ以降は、謝辞、参考文献、実行したプログラムについて記している。

2　研究内容

2.1 使用機器

本研究では、仮想並列環境の構築に 4 台のコンピュータを使用する。図 1 に本研究で使用した計算機ネットワークの概念図を示す。それぞれのコンピュータは、イーサーネットケーブルとハブにより、ネットワークが構成されている。本研究では、1 台のコンピュータをホストコンピュータ、他の 3 台のコンピュータをサブコンピュータとして扱う。表 1 に、本研究で用いたコンピュータとその性能を示す。

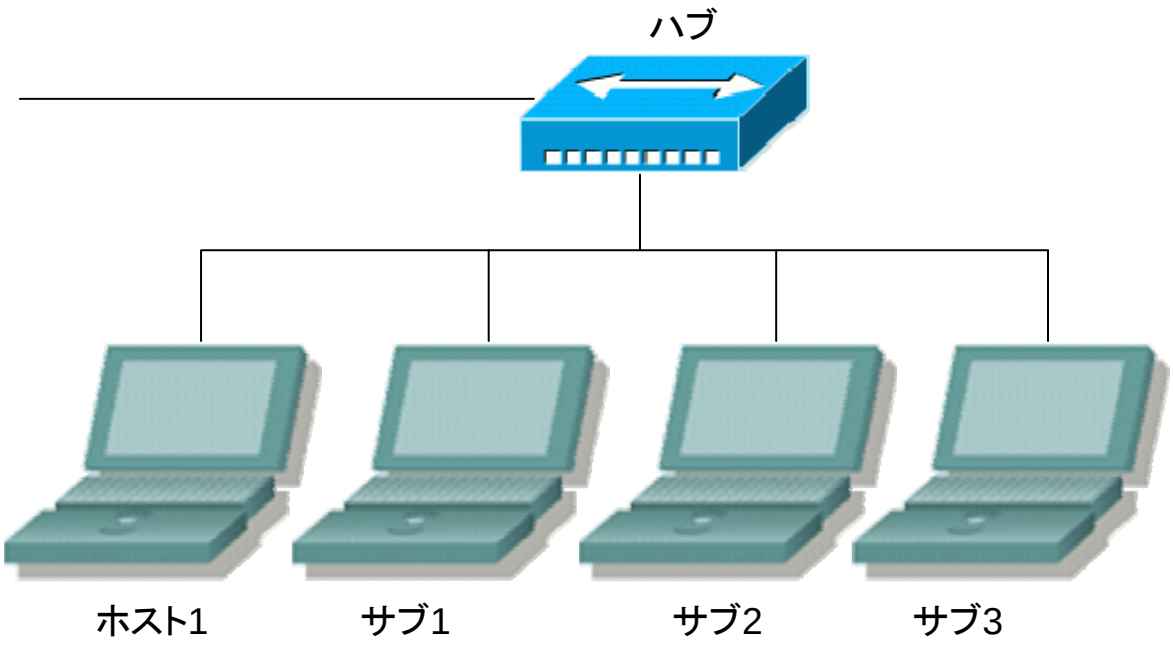

図 1. 本研究で使用した計算機ネットワークの概念図

表 1. 本研究で使用した計算機一覧

	ホスト	サブ 1	サブ 2	サブ 3
OS	Windows XP	Windows XP	Windows XP	Windows XP
CPU	Processor 1.6GHz	Processor 1.6GHz	Processor 1.6GHz	Processor 1.6GHz
メモリ	504MB	504MB	504MB	504MB

2.2 MPICH のインストールと環境設定

本研究を始めるにあたり、まずは MPI による仮想並列環境を構成する必要がある。本研究では、MPICH の最新のヴァージョンである MPICH2[4]を使用するため、そのパッケージを各計算機にダウンロードし、インストールを行う。MPICH2 は、アルゴンヌ国際研究所[3]の MPICH2 の公式ホームページ[4]において、無償で提供されている。

インストール後、環境変数を用いて MPICH2 のバイナリのあるフォルダに対し、PATH を指定する必要がある。本研究では、フォルダ”C:¥Program Files¥MPICH2¥bin”に対し環境変数 PATH の指定を行った。PATH の指定が正しく行われているか確認するためには、新規にコマンドプロンプトを立ち上げ、”mpiexec”と入力することで、引数の Usage と表示されるか否かにより確認できる。表示されれば PATH は正しく行われていると言える。

次に、各計算機にネットワークを通じて共有することが出来るフォルダを設定する。本研究では、各計算機でフォルダ”C:¥mpi”を作成し、このフォルダのプロパティでネットワークを通じて共有できるように設定を行った。プロセス実行の際には、この共通フォルダにある実行ファイルを使用するのである。また、Windows の OS で行うため、使用する計算機に交通のアカウント名とパスワードを持ったユーザを作成しなければならない。本研究では、各計算機にユーザ名”mpi”を作成し、それぞれ共通のパスワードを設定し、ワークグループを「HEIRETSU」として統一した。

2.3 Visual C++のインストール

本研究では、C/C++言語を用いる。その環境を作るために、Visual C++のインストールを行う。

方法としては、Visual C++2005 Express Edition が、マイクロソフトの公式ページ[9]より配布されているので、その仮想 CD をダウンロードすることで、インストールすることが出来る。

また、Visual C++のツールオプションより、MPICH2 のインクルードファイルおよびライブラリファイルのあるフォルダ”C:¥Program Files¥MPICH2¥include” および ”C:¥Program Files¥MPICH2¥lib”を追加し、リンカ入力の依存ファイル”mpi.lib”を追加することにより、MPICH2 を用いて並列プログラムを作成する環境を作る。

2.4 検証用プログラム

本研究では、MPI の性能を評価するため、膨大な計算を必要とする行列積の計算を行うプログラムを用いて、1 台の逐次計算による処理と、複数台による並列計算による処理とを行列のサイズ毎に場合分けし、処理時間にどれほどの差が生まれるか検証を行う。本研究で用いる行列積は、8 個の正方行列 $A_1, A_2, \ldots, A_8$ の行列積 $\prod_{k=1}^{8} A_k$ とし、4 台のコンピュータを用いて計算し、その計算時間を測定する。また、行列のサイズは 10*10、100*100、500*500、1000*1000 と変化させてサイズ毎に計算時間の測定を行う。

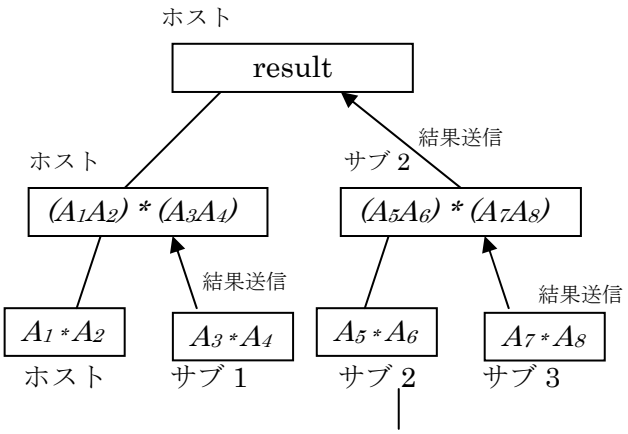

図 2.MPI を用いた行列積計算の概要

図 2 に計算の概要図を示す。処理方法は、ホストコンピュータが全ての行列を保持しており、生成した行列を他のサブコンピュータ 3 台に 2 つずつ送信する。各コンピュータが受け取った行列の積を計算し、結果を上位コンピュータに渡す。この計算を繰り返す。全ての計算が終われば、ホストコンピュータはサブコンピュータから計算結果を受信し、その結果を表示する。

この作業を 1 台の計算機で行った場合と、4 台の計算機で行った場合に分け、行列のサイズ毎に処理時間を測定する。処理時間は、計算を開始してから結果を表示するまでの時間を計測している。

3　結果・考察

表 2.MPI による行列積計算の処理時間(秒)

計算機数＼行列サイズ	10*10	100*100	500*500	1000*1000
1 台	0.015	0.066	10.87	167.2
4 台	0.012	0.051	7.29	86.3
向上率	1.25 倍	1.29 倍	1.49 倍	1.94 倍

上記の表 2 に、行列の大きさ毎に行列積計算の処理に要した時間をまとめた。

計算機 1 台で処理したときと比較すると、計算機 4 台で処理した場合のいずれも速度が向上していることが分かる。また、行列のサイズを増やし、処理するデータを大きくするほど、処理時間の向上率は上がっている。これは、処理数の多い場合よりも少ない場合のほうが、内部計算に比べて処理の送受信や同期にかかる時間の割合が大きいため、通信による遅延の影響を受けやすいからであろうと考えられる。

4　結論・今後の課題

本研究では、MPI による仮想並列計算環境の下で複数の計算機を使用し、行列積計算を行った。それにより、１台のコンピュータで処理を行う場合より、処理速度が大幅に向上したことを検証することが出来た。さらに、処理するデータが膨大であるほど、MPI による仮想並列計算は有用性が高いと言える。

また、今回は行列積計算の処理であったが、この並列環境を用いることで、処理時間が膨大である様々な問題も、大幅な処理速度の向上を図ることが出来るであろう。

謝辞

本研究を行うにあたり、並列処理の基礎知識から研究内容まで様々な指導、助言をいただきました石水隆先生、同じ研究室の皆様に深く感謝いたします。また、共同研究者の金久君、坂東君には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] 渡邊真也 著：MPI による並列プログラミングの基礎,
http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/PDF/chapter02.pdf
[3] Argonne National Laboratory, http://www.anl.gov/
[4] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/
[5] MPI 並列プログラム入門,
http://visualtechnology.jp/jp/support/man/pdf/mpi_parallel_programming_guide.pdf
[6] PVM, Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/
[7] PVM, http://www-geant4.kek.jp/~yoshidah/pvm/pvm.html
[8] Oak Ridge National Laboratory, http://www.ornl.gov/
[9] Visual Studio 2008 Express Editions,
http://www.microsoft.com/japan/msdn/vstudio/express/default.aspx
[10] C++入門, http://www.asahi-net.or.jp/~yf8k-kbys/newcpp0.html

付録

以下に、本研究に使用した行列計算の C 言語プログラムを示す。

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"


#define N 10// 行列サイズの定義
#define L 10
#define M 10

typedef double matNL[N][L];// 型の定義
typedef double matLM[L][M];
typedef double matNM[N][M];

void multiply(matNM c, matNL a, matLM b) // 行列の掛け算
{
        int i, j, k;
        double s;

        for (i = 0; i < N; i++)
                for (j = 0; j < M; j++) {
                        s = 0;
                        for (k = 0; k < L; k++) s += a[i][k] * b[k][j];
                        c[i][j] = s;
                }
}


#include <stdio.h>
#include <stdlib.h>

void matprint(int nrow, int ncol, double *a) // 行列表示
{
        int i, j;

        for (i = 0; i < nrow; i++) {
                for (j = 0; j < ncol; j++) printf("%15.1f", *a++);
                printf("¥n");
        }
}

int main(int argc,char *argv[])
{
        int num,myrank;
        int i, j,k,l;
```

```
int s = 100;// 送信サイズ設定
double start;
double finish;//時間計測用
static matNL a;// プログラム中常に有効な変数
static matLM b;
static matNM c;

static matNL d;// 2個目
static matLM e;
static matNM f;

static matNL a2;
static matLM b2;
static matNM c2;
static matNL d2;
static matLM e2;
static matNM f2;

static matNM g;
static matNM g2;

static matNM res;

MPI_Status status;

MPI_Init(&argc, &argv);//MPI命令の開始
MPI_Comm_size(MPI_COMM_WORLD, &num);//プロセス数を求める
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);//ランクを求める

start = MPI_Wtime();// 時間計測開始

if(myrank==0){


        for (i = 0; i < N; i++)//AとBの乱数生成
                for (j = 0; j < L; j++)
                        a[i][j] = rand() / (RAND_MAX / 10 + 1);// 行列Aに乱数を入れる
        for (i = 0; i < L; i++)
                for (j = 0; j < M; j++)
                        b[i][j] = rand() / (RAND_MAX / 10 + 1); // 行列Bに乱数を入れる
                        //      printf("A¥n");  matprint(N, L, (double *)a);
                        //      printf("B¥n");  matprint(L, M, (double *)b);

        for (k = 0; k < N; k++)// DとEの乱数生成
                for (l = 0; l < L; l++)
                        d[k][l] = rand() / (RAND_MAX / 10 + 1);// 行列Dに乱数を入れる
        for (k = 0; k < L; k++)
                for (l = 0; l < M; l++)
```

```
                        e[k][l] = rand() / (RAND_MAX / 10 + 1); // 行列Eに乱数を入れる
                        //      printf("D¥n");  matprint(N, L, (double *)d);
                        //      printf("E¥n");  matprint(L, M, (double *)e);


        // 繰り返し
        for (i = 0; i < N; i++)
                for (j = 0; j < L; j++)
                        a2[i][j] = rand() / (RAND_MAX / 10 + 1);// 行列A2に乱数を入れる
        for (i = 0; i < L; i++)
                for (j = 0; j < M; j++)
                        b2[i][j] = rand() / (RAND_MAX / 10 + 1); // 行列B2に乱数を入れる
                        //      printf("A2¥n");  matprint(N, L, (double *)a2);
                        //      printf("B2¥n");  matprint(L, M, (double *)b2);


        for (k = 0; k < N; k++)// 2回目
                for (l = 0; l < L; l++)
                        d2[k][l] = rand() / (RAND_MAX / 10 + 1);// 行列D2に乱数を入れる
        for (k = 0; k < L; k++)
                for (l = 0; l < M; l++)
                        e2[k][l] = rand() / (RAND_MAX / 10 + 1); // 行列E2に乱数を入れる


        // それぞれのコンピュータにデータを送信
        MPI_Send(d,s,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
        MPI_Send(e,s,MPI_DOUBLE,1,1,MPI_COMM_WORLD);
        MPI_Send(a2,s,MPI_DOUBLE,2,2,MPI_COMM_WORLD);
        MPI_Send(b2,s,MPI_DOUBLE,2,3,MPI_COMM_WORLD);
        MPI_Send(d2,s,MPI_DOUBLE,3,4,MPI_COMM_WORLD);
        MPI_Send(e2,s,MPI_DOUBLE,3,5,MPI_COMM_WORLD);


        multiply(c, a, b);// 行列AとBの計算をCに代入
        //printf("AB¥n");  matprint(N, M, (double *)c);

        MPI_Recv(f,s,MPI_DOUBLE,1,6,MPI_COMM_WORLD,&status);

        multiply(g, c, f);// 行列CとFの計算をGに代入
        //printf("(AB)*(DE)¥n");  matprint(N, M, (double *)g);

        MPI_Recv(g2,s,MPI_DOUBLE,2,8,MPI_COMM_WORLD,&status);

        multiply(res,g,g2);//行列GとG2の計算をRESに代入
        //printf("{(AB)*(DE)}*{(AB2)*(DE2)}¥n");  matprint(N, M, (double *)res);
        printf("result¥n");
}
```

```
	else if(myrank == 1)
	{
		MPI_Recv(d,s,MPI_DOUBLE,0,0,MPI_COMM_WORLD,&status);
		MPI_Recv(e,s,MPI_DOUBLE,0,1,MPI_COMM_WORLD,&status);

		multiply(f, d, e);// 行列DとEの計算をFに代入
		//	printf("DE¥n");  matprint(N, M, (double *)f);
		MPI_Send(f,s,MPI_DOUBLE,0,6,MPI_COMM_WORLD);
	}


	else if(myrank == 2)
	{
		MPI_Recv(a2,s,MPI_DOUBLE,0,2,MPI_COMM_WORLD,&status);
		MPI_Recv(b2,s,MPI_DOUBLE,0,3,MPI_COMM_WORLD,&status);
		multiply(c2, a2, b2);//行列A2とB2の計算をC2に代入
		//	printf("AB2¥n");  matprint(N, M, (double *)c2);

		MPI_Recv(f2,s,MPI_DOUBLE,3,7,MPI_COMM_WORLD,&status);

		multiply(g2, c2,f2);//行列C2とF2の計算をG2に代入
		//	printf("AB2*DE2¥n");  matprint(N, M, (double *)g2);

		MPI_Send(g2,s,MPI_DOUBLE,0,8,MPI_COMM_WORLD);
	}

	else if(myrank == 3)
	{
		MPI_Recv(d2,s,MPI_DOUBLE,0,4,MPI_COMM_WORLD,&status);
		MPI_Recv(e2,s,MPI_DOUBLE,0,5,MPI_COMM_WORLD,&status);
		multiply(f2, d2, e2);//行列D2とE2の計算をF2に代入
		//	printf("DE2¥n");  matprint(N, M, (double *)f2);
		MPI_Send(f2,s,MPI_DOUBLE,2,7,MPI_COMM_WORLD);
	}
	MPI_Barrier(MPI_COMM_WORLD);//時間計測のため同期をとる
	if(myrank == 0){
		finish = MPI_Wtime();
		printf("処理時間：%10.6f seconds¥n",finish - start);
	}
	MPI_Finalize();//MPI命令終了
	return EXIT_SUCCESS;
}
```