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

０６-１-０３７-００３８

金久　英之

近畿大学理工学部情報学科

平成 22 年 2 月 5 日提出

概要

近年では、計算機がなくてはならない存在になっており、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、大量のデータを高速に処理することが求められている。それらを処理するには、膨大な時間がかかる。高速な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）が用いられる。しかし、一般的に並列計算機は高価であるために、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仮想的並列計算機とする手法が注目されている。本研究では、MPI〔Message Passing Interface〕[1][2]を用いて仮想並列計算を行い、その実用性を検証する。MPI は無料提供されているソフトウェアであり、MPICH2 のページ[3]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。

目次

第1章 序論

1.1. 本研究の背景

近年、様々な情報や記憶デバイスが計算機で取り扱われている。取り扱われる情報の量などは日々増大しており、その処理時間を高速化することは計算機を使用する上での重大な課題である。高速な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）が用いられる。しかし、一般的に並列計算機は高価であるために、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仮想的並列計算機とする手法が注目されている。本研究では、MPI〔Message Passing Interface〕[1][2]を用いて仮想並列計算を行い、その実用性を検証する。

1.2. 並列計算機

並列計算機(Parallel Computer)とは、単一 CPU の計算機の計算能力を越えるために複数台の CPU を用いて構成された計算機であり、問題の計算にかかる処理時間の高速化を図るために、複数のプロセッサを用いて並列処理を行える計算機である。並列計算機は大きく２つに分類される。図 1 のように全てのプロセッサが共通のメモリを使用して読み書きを行い、他のプロセッサ間で通信を行う共有メモリ型並列計算機(Shared Memory Parallel Computer)と、図 2 のように各プロセッサがそれぞれメモリを持ちネットワークを通じて通信を行う分散メモリ型並列計算機(Distributed Memory Parallel Computer)である。

図１と図２に、共有メモリ型並列計算機と分散メモリ型並列計算機の概念図を示す。プロセッサ間の通信については、共有メモリ型並列計算機ではメモリを通して行われ、分散メモリ型並列計算機ではネットワークを通して行われる。共有メモリ型並列計算機は、専用の並列計算機を必要とするのに対し、分散メモリ型並列計算機は、複数の計算機をネットワーク接続して、仮想的に並列計算機を構築できる利点がある。そこで、本研究では、分散メモリ型並列計算機を用いる。

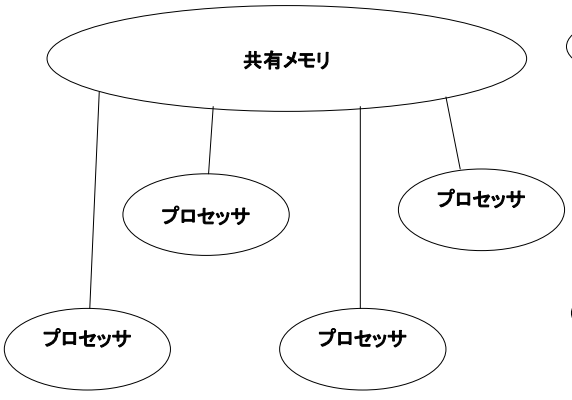

図１：共有メモリ型並列計算機

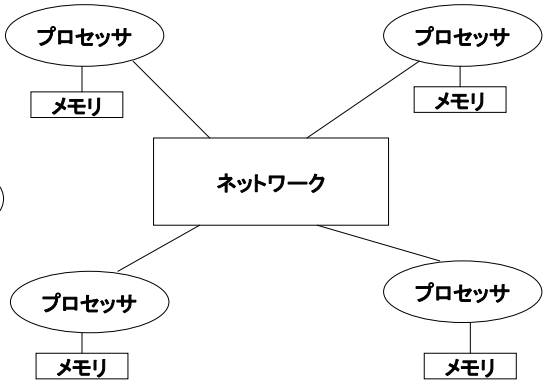

図２：分散メモリ型並列計算機

1.3. MPI と MPICH

分散メモリ型仮想並列計算の構築には、MPI(Message Passing Interface)[1][2]のソフトウェアを用いて行うことができる。MPI は Message Passing Interface の略称であり、並列計算を利用するために標準化された規格である。メッセージ通信のプログラムを記述するために広く使われる「標準」を目指して開発されたもので、複数の計算機が情報をバイト列からなるメッセージとして送受信することで協調動作を行います。ライブラリレベルでの並列化であるため、言語を問わず利用でき、プログラマが細密なチューニングを行えます。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 MPI(Message Passing Interface)[1][2]を用いて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評価する。MPI は無料提供されているソフトウェアであり、MPICH2 のホームページ[3]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。

MPICH は、ゴードン国立研究所というアメリカの研究所が開発を行い、無償でソースコードを配布したライブラリであり、移植しやすさを重視した作りになっている。この MPICH は UNIX や Linux に限らず、Windows 系へのサポートもしており、OS への対応が充実している。本研究では、MPICH の最新のヴァージョンである MPICH 2 を使用して研究を進めていく。

1.4. 本報告書の構成

2 節以降では、まず 2.1 で使用機器を示す。次に 2.2、2.3、2.4 で MPI、Visual C++、Microsoft Platfrom SDK のインストールと環境設定を説明し、2.5 ではワークグループの設定について述べる。次に 3 節で目的と計算方法、4 節では結果と考察を述べ、5 節では、結論と今後の課題について述べている。また、付録として実行したプログラムとその実行結果を示す。

準備

1.5. 使用機器

本研究では、仮想並列環境の構築するためにコンピュータ 4 台を使用します。それぞれのコンピュータは、イーサネットケーブルとハブによりネットワーク環境が構築されている。図 3 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。本研究では、1 台をホストコンピュータとして、残り 3 台をサブコンピュータとして扱う。表 1 に、そのコンピュータの性能を示す。

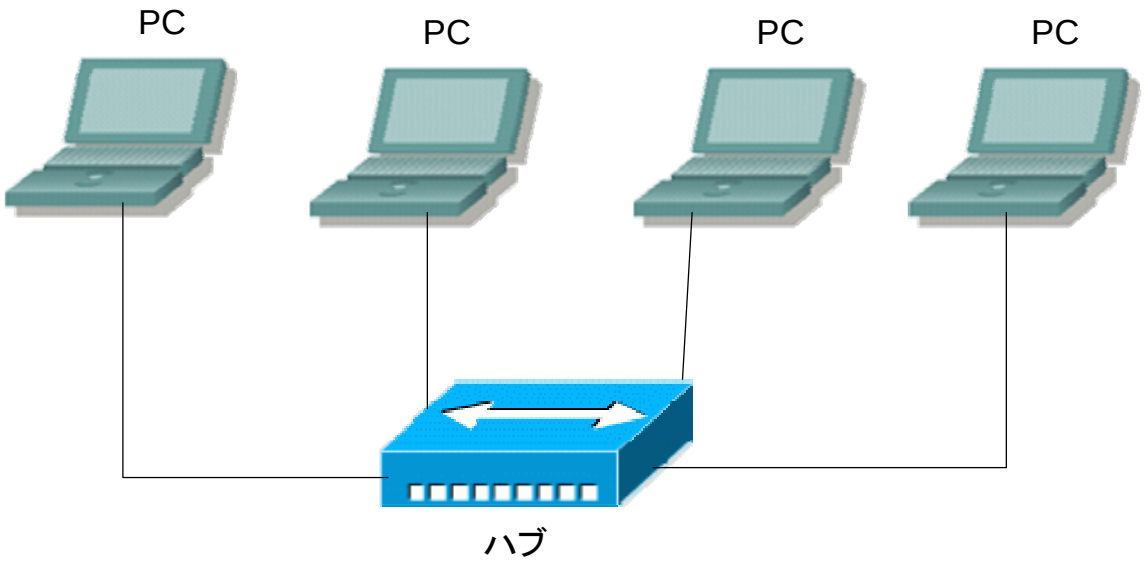

図 3 ：使用した計算機ネットワークの概念図

表 1 実験で使用した計算機

	ホストコンピュータ	サブコンピュータ 1	サブコンピュータ 2	サブコンピュータ 3
OS	Windows XP	Windows XP	Windows XP	Windows XP
CPU	Intel Pentium 1.60GHz	Intel Pentium 1.60GHz	Intel Pentium 1.60GHz	Intel Pentium 1.60GHz
Memory	504MB	504MB	504MB	504MB

1.6. MPICH のインストールと環境設定

仮想並列計算環境を作るため、MPICH のホームページ[3]よりパッケージをダウンロードし、インストールを行い、インストール後に、環境変数を用いて MPICH プログラムのあるフォルダへの PATH を指定をする必要がある。この指定は、マイコンピュータのプロパティから行うことができる。プロパティで表示されるシステム環境変数のリストから、変数名 path を選択し、変数値の最後に;C:¥Program Files¥MPICH2¥bin¥を追加する。PATH の指定が正しくできているかの確認は、新規にコマンドプロンプトを立ち上げ、mpiexec と入力したときに、引数の Usage が表示されるかどうかで確認することができる。

1.7. Visual C++のインストール

次に、C 言語の環境を作るために、Visual C++のインストールを行う。
VisualC++2008 Express Edition が、マイクロソフトの公式ページ[6]より配布されているので、ダウンロードしインストールを行うことができる。

1.8. Microsoft Platform SDK のインストール

Visual C++の開発環境をあげるために、マイクロソフトの公式ページ[7]より配布されている Platform SDK のインストールを行う。

Platform SDK のインストール後、Visual C を起動し、Platform SDK の Executable、Include、Library に PATH を通し、更新を行う。これにより、Visual C++の環境が向上される。

1.9. ワークグループの設定

OS が Windows の場合には、使用する計算機に共通のアカウント名とパスワードを持つユーザを設定しておく必要がある。並列環境の作成のため、それぞれのコンピュータに mpi アカウントを作りパスワードを共通のものとし、ワークグループを「HEIRETSU」として統一する。これにより、コンピュータ同士でのネットワークが構築される。

また、プロセスの実行の際には、すべての使用コンピュータに実行ファイルを置く必要があるため、mpi 共有フォルダをそれぞれのコンピュータに作成する。

第2章　目的・方法

2.1. 目的

本研究では、MPI の性能を研究するために、膨大な行列積計算を１つのコンピュータで計算する場合と、MPI を用いて複数のコンピュータで計算する場合を比較し、処理時間にどれだけ差があるかを計測する。

2.2. 計算方法

本研究では、８個の正方行列 $A_1\ A_2 \ldots A_8$ の行列積 $\prod_{k=1}^{8} A_k$ を 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。このとき、行列のサイズを 10*10,100*100,300*300,500*500,700*700,1000*1000 と変化させ、サイズごとに測定を行う。

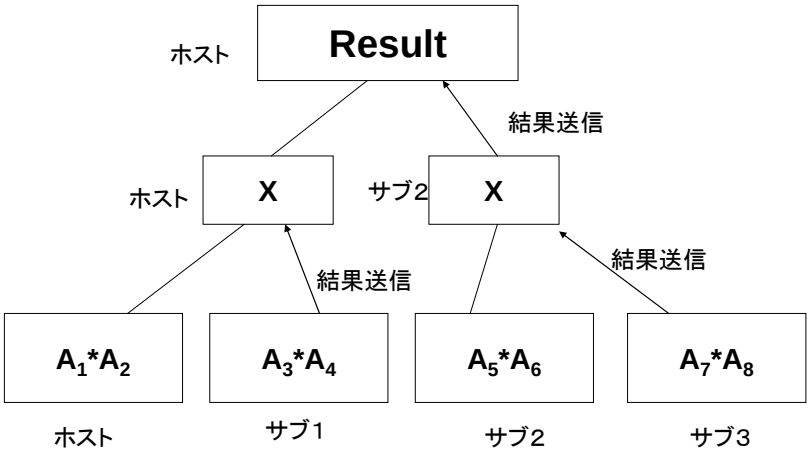

図４　MPI を用いた行列積計算の概念図

図４に計算の概念図を示す。初期状態では、ホストコンピュータが全ての行列を保持している。ホストコンピュータは、他のサブコンピュータに生成した行列を２つずつ送信する。各コンピュータにて、受け取った行列同士の積を計算し、その結果を上位コンピュータに渡し、また計算を繰り返す。全ての計算が終わった後、ホストコンピュータは他の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。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、この作業を１台で行ったときと、複数台で行ったときの計算時間を測定し、比較する。

第3章　結果・考察

表２　行列計算の処理時間の平均(秒)

CPU＼行列数	10	100	300	500	700	1000
1台	0.015	0.066	0.197	10.87	15.31	167.2
4台	0.012	0.051	0.152	7.291	10.21	86.3
速度向上率	1.25 倍	1.294 倍	1.296 倍	1.49 倍	1.5 倍	1.93 倍

表２に MPI を用いて行列積計算を行ったときの処理時間の平均を示す。表２より、CPU 数 1 台のときと比べると、どれも大幅に速度が向上しているのが見て取れる。特に、処理の数が大きければ大きいほど、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少ない処理の場合のほうが、処理全体の時間に対するデータの送受信や、同期の時間にかかる時間の割合が大きいため、それの影響を受けやすいからであろうと考えられる。この並列環境を用いれば、今まで処理時間が膨大だった問題も、大幅に速度を向上して、さらにたくさんの情報を扱うことができると考えられる。

第4章　結論・今後の課題

本研究により、並列計算環境を用いて複数のコンピュータを使って膨大なデータの処理を行うことと、1 台のコンピュータでその処理を行った時を比べると、大幅に処理速度が向上することを確認出来た。この並列環境を用いれば、今まで処理時間が膨大だった問題も、大幅に速度を向上して、さらにたくさんの情報を扱うことができると考えられる。また、少量のデータの処理においても仮想並列計算環境を用いれば、処理時間の向上が認められたことから、身近なことにも利用することが出来るであろうと考えられる。

謝辞

研究を進めていくにあたり、石水隆先生、同じ研究室の皆様に、基礎知識から研究内容まで指導や助言をいただきましたことを心より感謝を申し上げます。

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] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/

[4] MPICH2 on Windows Local,
http://ums.futene.net/wiki/Paralell/4D5049434832206F6E2057696E646F7773204C6F63616C.html

[5] Argonne National Laboratory,
http://www.mcs.anl.gov/research/projects/mpich2/indexold.html

[6] Visual Studio, http://www.microsoft.com/japan/msdn/vstudio/express/

[7] Windows® Server 2003 SP1 Platform SDK Web Install,
http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en

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。

matrixMultiplication.c

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

#define N 10
#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;
}
```