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

07-1-037-0065

清水周

近畿大学理工学部情報学科

平成 23 年 1 月 28 日提出

概要

並列計算とは、複数のマイクロプロセッサなどに処理を分散して割り当て、同時に計算・処理を行うことで、システム全体の処理性能を向上させる技術であり、また、そのような環境を効率的に活用するためのソフトウェアやプログラミングの手法の総称である。並列計算には、複数のプロセッサを持つ並列計算機が必要になる。しかし専用の並列計算機は非常に高価であり容易に利用できない。

このため、ネットワーク接続した複数台の計算機を一つの仮想的な並列計算機として扱う手法が最近注目されている。仮想並列計算では専用の高価な計算機を使わず複数の計算機を使うため安くすみ、複数であるため部分的な故障が全体のダウンに結びつかない。またデータを共有しやすい。仮想並列計算の欠点としてはネットワークの信頼性が必要であり、セキュリティの対策も万全に施さないといけないという点が挙げられる。

本研究では、仮想並列計算機を実現するためのソフトウェアの一つである MPI(Message Passing Interface) [1] を用い仮想並列計算の有用性を検証する。MPI とは、世界標準とされている分散メモリ型並列処理におけるメッセージ交換のためのライブラリである。このライブラリの基本はメッセージパッシング (message passing)であり、あるプロセスから他のプロセスへデータを明示的に送る方法で、極めて効率の良い並列プログラムを書くことができる。検証方法としては、性能検証のための問題に対して MPI を用いて並列処理した場合、逐次処理した場合と比べてどの程度処理時間が短縮できるかを計測する。本研究では、性能検証用の問題として円周率計算を用いる。

本研究の結果より、MPI を用いた計算時間が短縮され効果があったアルゴリズムがあり。並列計算の有用性が判明した。

目次

1. 序論

1.1 本研究の背景

近年、計算機の性能は向上し続けており、ハードディスクの記憶容量なども増加し続けている。計算機の性能向上に伴い、一度に扱うデータの量は増大して来ている。しかし、計算機が扱うデータ量は今後も増大していくと予想される一方、計算機自体の性能の向上は近い将来頭打ちになることが予想される。膨大なデータに対する処理の高速化の方法として、複数のプロセッサを持つ並列計算機(Parallel Computer)を用いた並列処理(Parallel Processing)がある。並列計算機の種別には、論理的にメモリを共有する共有メモリ型並列計算機(Shared Memory Parallel Computer)、全プロセッサがデータを共有しているためプログラミングしやすいが CPU はなるべく高速なメモリアクセスを必要とするためハード的な負担は大きい、上記の特性により、サーバマシンに適している。分散共有メモリ型計算機(Distributed Memory Parallel Computer)、共有メモリ型と比較した場合ハードウェアへの負担は小さくなるがプログラミングが複雑である。

並列処理を行うためには、並列計算機が必要となる。しかし、専用の並列計算機は高価であるため、ネットワーク接続した複数台の計算機を一つの仮想的な並列計算機として扱うクラスタ処理(Cluster Computing)が現在注目されている。

仮想並列計算機を構築するソフトウェアは様々なものが開発されており、無料で提供されているものもある。このため、仮想並列計算機を個人で使用することも容易になっており、今後並列計算機の重要性はより拡大していくと考えられる。

1.2 仮想並列計算機を構築するソフトウェア

無料で提供されている仮想並列計算機を構築するソフトウェアとしては、MPI(Message Passing Interface)[1], PVM(Parallel Virtual Machine)[4], OpenMP[5], Score[6]等がある。以下に、これらについての説明をする。

MPI(Message Passing Interface)[1]は世界標準とされている分散メモリ型並列処理におけるメッセージ交換のためのライブラリである。このライブラリの基本はメッセージパッシング (message passing)であり、あるプロセスから他のプロセスへデータを明示的に送る方法で、 極めて効率の良い並列プログラムを書くことができる。

PVM(Parallel Virtual Machine)[4]は並列計算を行うためのソフトウェアである。PVM はアメリカのオークリッジ国立研究所のメンバーが中心となって開発され、動作するマシンの種類が多いのが特徴である。PVM ソフトウェアシステムの構成は、デーモンとルーチンライブラリの大きく 2 つに分けられる

OpenMP[5]は並列コンピューティング環境を利用するために用いられる標準化された基盤である。OpenMP の利点としては、移項のたやすさ、移植性の高さ、単一プロセッサ環境との共存の容易さが挙げられる。

Score は[6]は経済産業省が設立した超並列処理研究推進委員会である新情報処理開発機構(Real World Computing Partnership, RWCP)にて開発された Linux 用クラスタ計算機用超並列プログラム実行環境である。主に流体行動解析、量子化学計算、物理学計算等の科学技術計算分野で使用されている。

1.3 MPI(Message Passing Interface)

本研究では、並列処理環境を実現するために MPI(Message Passing Interface)[1]を用いる。MPI を用いた理由は、世界標準とされている分散メモリ型並列処理におけるメッセージ交換のためのライブラリでありどんな計算機でも動くからである。フリーであらゆるアーキテクチュアをサポートしている MPICH[7]とい

うソフトウェアが存在することも理由として挙げられる。

1.4 本研究の目的

本研究では、MPI を用いた並列計算の有用性を検証する。検証方法としては、性能検証のための問題に対して MPI を用いて並列処理した場合、逐次処理した場合と比べてどの程度処理時間が短縮できるかを計測する。

MPI の性能検証用の問題として本研究では円周率の計算を行なう。この処理を 1 台の計算機で行なった場合と、複数台で処理した場合との比較を行ないどの程度処理速度の向上が見られるかを検証する。

1.5 円周率の計算

円周率の計算は古来より様々な方法が考えられてきた。

数学が発達する以前には、円に外接および内接する多角形の周の長さから円周率の近似値を得ていた。紀元前 3 世紀に古代ギリシアの哲学者 Archimedes (前 287-前 212)は円に接する正 96 角形の周の長さから、円周率を小数点以下 2 桁まで求めた。同様の手法で 5 世紀に中国の天文学者祖沖之(429-500)は正 24576 角形(推定)を用いて小数点以下 7 桁まで、17 世紀にドイツの数学者 Ludolf van Ceulen (1539-1610)はその生涯を掛けて正 2^{62} 角形を用いて小数点以下 70 桁まで計算した。(ただし後に 36 桁目が間違っていたことが判明する。) 日本では、1663 年に村松茂清 (1608-1695)が正 2^{15} 角形を用いて小数点以下 7 桁まで計算した。

数学の発達により、14 世紀には円周率が無限級数和や無限乗積として表されることが判明する。14 世紀にインドの数学者 Madhava (1350-1425)は後に Leibniz の公式と呼ばれる無限交差級数

$$\frac{\pi}{4}=\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots$$

を発見した。しかしこの級数は収束が遅く、小数点以下 10 桁を計算するためには 100 億個以上の項数が必要となる。1655 年イギリスの数学者 Jhon Wallis (1616-1703)は Wallis 積と呼ばれる無限乗積

$$\frac{\pi}{2}=\prod_{n=1}^{\infty}\frac{(2n)^2}{(2n-1)(2n+1)}=\left(\frac{2^2}{1\cdot 3}\right)\cdot\left(\frac{4^2}{3\cdot 5}\right)\cdot\left(\frac{6^2}{5\cdot 7}\right)\cdot\left(\frac{8^2}{7\cdot 9}\right)\cdot\cdots$$

を発見した。1671 年スコットランドの天文学者 James Gregory (1638-1675)は Gregory-Leibniz 級数

$$\arctan x=\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}x^{2n+1}=x-\frac{1}{3}x^3+\frac{1}{5}x^5-\frac{1}{7}x^7+\frac{1}{9}x^9-\cdots$$

を発見した。1706 年イギリスの天文学者 John Machin (1686-1751)は Machin の公式

$$\frac{\pi}{4}=4\arctan\frac{1}{5}-\arctan\frac{1}{239}$$

を発見し、この式と Gregory-Leibniz 級数を用いて円周率を小数点以下 100 桁まで計算した。以降様々な円周率を求める無限級数和・無限乗積が発見され、1872 年にはイギリスの数学者 William Shanks (1812-1882)が小数点以下 527 桁まで計算した。

20 世紀後半からは計算機により円周率を計算できるようになった。表 1 に計算機による円周率計算の代表的な結果を示す。1949 年に George Reitwiesner らが 世界最初の計算機 ENIAC を用いて 70 時間かけて

小数点以下 2037 桁まで計算した。 1982 年に田村が MELCOM900II を用いて 2,097,144 桁まで計算した。1997 年 8 月に高橋，金田らが HITACHI SR2201 と 1024 台の演算機(並列計算)を用いて、51,539,600,000 桁まで計算した。

今日では、円周率の計算は計算機の性能評価の指標となっている。今日の計算機の性能は非常に高く、一般に手に入るパーソナルコンピュータでも円周率を計算できる。2010 年 1 月、Fabrice Bellard がパソコンで 131 日かけ 2 兆 6999 億 9999 万桁まで計算した[16]。また、2010 年 8 月、近藤茂と Alexander J.Yee は、自作パソコンを用いて 90 日かけ 5 兆桁まで計算した[17]。円周率を計算するソフトウェアも多く公開されており、誰でも簡単に円周率を計算できる。代表的なソフトウェアとして、東京大学で開発されたスーパーπ[12]がある。このソフトウェアを用いて最大 3355 万桁まで計算することができる。

計算機を用いて円周率を計算する場合、どのような公式を用いるかが重要となる。用いる公式は、できるだけ早く値が収束するものが望ましいとされる。円周率を計算する際に用いられる代表的な公式を以下に示す。なお、各公式名の右に表記されている比率はチュドノフスキーの公式の計算時間を 1 とした時の、その公式の計算時間の比率である。

チェドノフスキーの公式(比率 1)

$$\pi = \frac{426880\sqrt{10005}}{\sum_{n=0}^{\infty} \frac{(6n)!(545140134n + 13591409)}{(n!)^3(3n)!(-640320)^{3n}}}$$

ラマヌジャンの公式(比率 1.60)

$$\pi = \frac{1}{2\sqrt{2}\sum_{n=0}^{\infty} \frac{(4n)!(1103 + 26390n)}{4^{(4n)}(n!)^4 99^{(4n+2)}}}$$

マチンの公式(比率 3.96)

$$\pi = 16tan^{-1}\frac{1}{5} - 4tan^{-1}\frac{1}{239}$$

シュトルマーの公式(比率 3.96)

$$\pi = 176tan^{-1}\frac{1}{57} + 28tan^{-1}\frac{1}{239} - 48tan^{-1}\frac{1}{682} + 96tan^{-1}\frac{1}{12943}$$

表 1 計算機によるπ計算の歴史

氏名	年・月	計算桁	使用機種
リトワイズナー等	1949	2,037	ENIAC
ニコルソン，ジーネル	1954	3,092	NORC
フェルトン	1957	7,480	Pegasus
ジェニュイス	1958	10,000	IBM 704
フェルトン	1958	10,020	Pegasus
ギュー	1959	16,167	IBM 704
シャンクス，レンチ	1961	100,265	IBM 7090

ギュー，フィリャトル	1966	250,000	IBM 7030
ギュー，ディシャン	1967	500,000	CDC 6600
ギュー，ブーエ	1973	1,001,250	CDC 7600
三好，金田	1981	2,000,036	FACOM M-200
ギュー	1981-82	2,000,050	未確認
田村	1982	2,097,144	MELCOM 900II
田村，金田	1982	4,194,288	HITAC M-280H
田村，金田	1982	8,388,576	HITAC M-280H
金田，吉野，田村	1983	16,777,206	HITAC M-280H
後，金田	1983.10	(*)10,013,395	HITAC S-810/20
ゴスパー	1985.10	17,526,200	Symbolics 3670
ベイリー	1986.1	29,360,111	CRAY-2
金田，田村	1986.9	33,554,414	HITAC S-810/20
金田，田村	1988.1	204,326,551	HITAC S-820/80
チュドノフスキー兄弟	1989.5	480,000,000	CRAY-2 IBM-3090/VF
チュドノフスキー兄弟	1989.6	535,339,270	IBM 3090
金田，田村	1989.7	536,870,898	HITAC S-820/80
チュドノフスキー兄弟	1989.8	1,011,196,691	IBM 3090
金田，田村	1989.11	1,073,740,799	HITAC S-820/80
チュドノフスキー兄弟	1991.8	2,260,000,000	自作の計算機
高橋，金田	1995.6	3,221,220,000	HITAC S-3800/480
高橋，金田	1995.8	4,294,960,000	HITAC S-3800/480
高橋，金田	1995.10	6,442,450,000	HITAC S-3800/480
高橋，金田	1997.8	51,539,600,000	HITACHI SR2201 と 1024 台の演算機(並列計算)
高橋，金田	1999.5	68,719,470,000	HITACHI SR8000 と 64 台の演算機(並列計算)
高橋，金田	1999.9	206,158,430,000	HITACHI SR8000 と 128 台の演算機(並列計算)

(*) スーパーコンピューターを利用した最初の計算のため参考記録

1.6 本研究への準備

MPICH2 のインストール、及び環境変数の設定を行なった。またソースファイルの作成に当たり Microsoft Visual Studio 2008 を使用したため、そのインストールを行なった。

1.7 本報告書の構成

本報告書は 4 つの章から構成されている。第 0 章で MPI を用いた実験環境および円周率を求めるアルゴリズムと、そのアルゴリズムを基にした MPI プログラムについて述べる。続く、3 章では本研究で得られた結果と考察を記述する。また、4 章では結論および今後の課題について述べる。最後に、付録に本研究で作成した円周率を計算する MPI プログラムを掲載する。

表 2 本研究で使用した計算機のスペック

	スペック		
	OS	プロセッサ	メモリ(GB)
計算機	Windows Vista	Intel®Core™2DUO CPU L7300 1.40GHz	RAM 1.00
	Windows XP	Intel®Core™2CPU 6300 1.86 GHz	RAM 2.00
	Windows Vista	Intel®Core™i5CPU750 2.67 GHz	RAM 4.00

2 研究内容

2.1 MPI(Message Passing Interface)

MPI (Message Passing Interface)[1]は1991年に計算機間のメッセージ通信の標準規格として開発され第一章で述べたように世界標準のためのライブラリであり、メッセージパッシング (message passing)である。メッセージパッシングは分散メモリ環境において現在、最も主流の方式であり、かなり本格的な並列プログラミングが実現することができる。この方式をプログラムで実現するためのライブラリが，メッセージパッシングライブラリである 。

無料で提供されているMPIの主な実装として、MPICH2[7]、LAM[8]、OpenMPI[9]等がある。

MPICHはMPIを実装するためのソフトウェアとしてArgonne National Laboratory[15]で開発された。2005年にはMPICHの後継としてMPICH2が開発され､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ためプログラムのソースコードを変更することなく、分散メモリ環境、共有メモリ環境のマシンで動作させることが可能である。

LAMはネットワーク接続された計算機のための並列処理環境および開発システムであり、拡張モニタリングおよびデバッグツールによってサポートされたプログラミング標準である。

OpenMPIは高性能メッセージパッシングライブラリーである。 OpenMPIはコミュニティー，研究機関，パートナー企業によって開発,維持されているオープンソースMPI-2 を実装されている。OpenMPIの特徴は,完全な MPI-2 規格準拠，スレッドセーフと並行性，ダイナミックなプロセスのスワッピング，ネットワークと耐障害性の処理，異種ネットワークのサポート，シングルライブラリのサポート，ランタイムとしての役割， 多数のジョブスケジューラーのサポート，多数の OS のサポート，アクティブなメーリングリスト，BSD ライセンスに基づくオープンソースライセンス，などである。

本研究では、MPIを実装するソフトウェアとして、現在最も幅広く使用されているMPICH2を用いる。

2.2 実験の環境

本研究では、異なるスペックをもつ計算機 3 台をネットワークで繋ぎ MPI 環境を構築する。本研究で使用する計算機のスペックを表 2 に示す。本研究では、計算機 3 台を 100Base-TX による LAN 接続し、MPI 環境の構築を行った。図 1 に本研究で用いた LAN の構成図を示す。

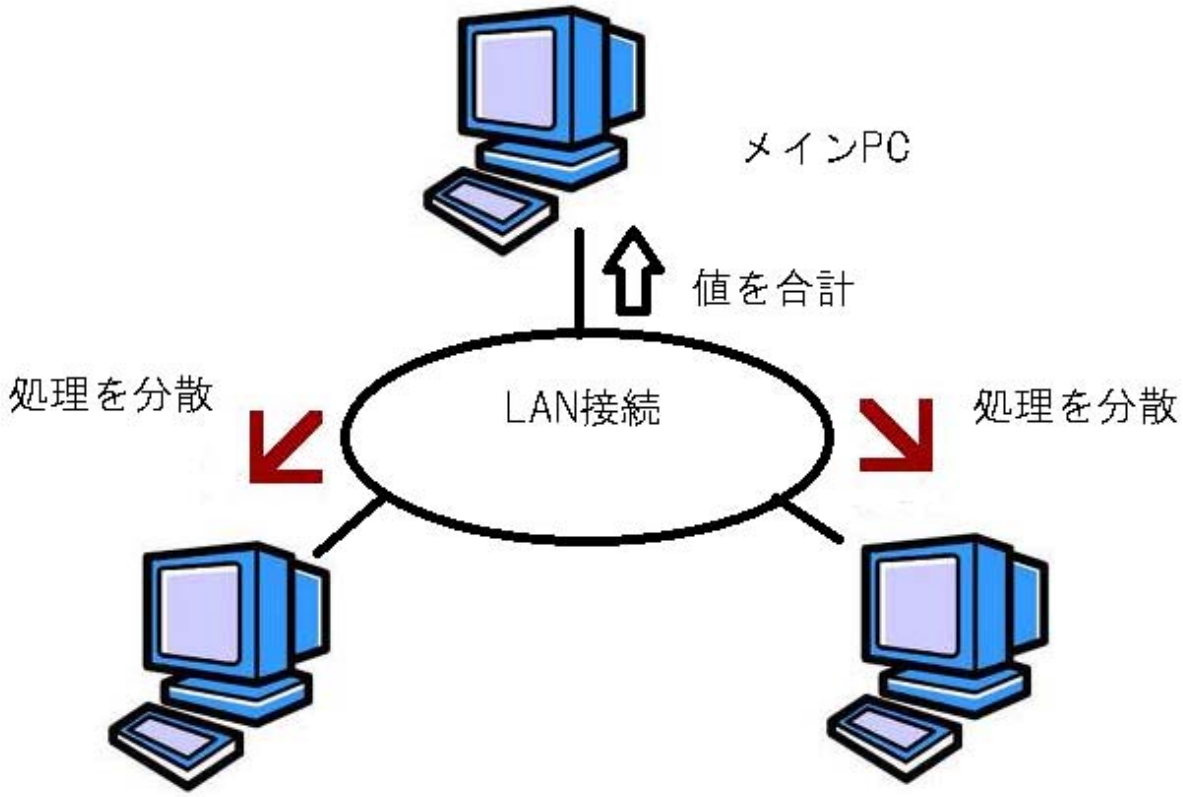

図 1　ネットワークの構成図

2.3 MPICH2 のインストール

本節では、MPI 環境を構築するために MPICH2 のインストール方法について述べる。MPICH2 を使用するためには、MPI 環境を構築する全ての計算機に MPICH2 をインストールする必要がある。MPICH2 は MPICH2：downloads[11]よりダウンロードすることができる。本研究では OS として Windows 系 OS を用いたので、Windows 版 MPICH2 のインストーラ(mpich2-1.3.1-win-ia32.msi)を各計算機に実行形式のイストーラファイルをダウンロードした。

ダウンロードしたインストーラファイルを実行することによりインストールを行うことができる。インストールするフォルダの指定と自分専用か他ユーザーも使うのか聞かれるので自分に適したモノを選択し、Next を押し進めていく。

また、MPICH2 を使用するためには、環境変数 PATH に MPICH のバイナリファイルがあるフォルダを設定する必要である。環境変数を変更するために、「マイコンピュータ」を右クリック、「プロパティ」-「詳細設定」-「環境変数」を開き、「システム環境変数」のリストから「path」を選択し、「編集」を押し、変数値

の最後に「C:¥Program Files¥MPICH2¥bin¥」を追加する。

並列計算実行時に、ファイヤーウォールを無効にしなければ実行できないのですべてのマシンで無効にしておく。

2.4 数値積分アルゴリズム

円周率の計算は、級数展開を利用して積分を行うことで求められる。積分により円周率を求める式としては以下に示す式などがある。

$$\pi = \int_{-1}^{1} \sqrt{1 + (y')^2}\, dx \tag{1}$$

$$\pi = \int_{-\infty}^{\infty} \frac{1}{1 + x^2}\, dx \tag{2}$$

$$S = \sum_{i=0}^{n} \frac{4}{1 + {x_i}^2} \cdot h \qquad \left(x_i = h \cdot (i - 0.5),\ h = \frac{1}{n} \right) \tag{3}$$

$$\arctan(x) = \int_{0}^{x} \frac{1}{1 + t^2}\, dt \tag{4}$$

$$\arctan(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n + 1} x^{2n+1} \tag{5}$$

$$\arcsin(x) = x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdots (2n - 1)}{2 \cdot 4 \cdots (2n)} \frac{x^{2n+1}}{2n + 1} \tag{6}$$

本研究では、円周率を求める式として(3)を用いる。この式は他の式のような積分を必要とせず、簡単な四則演算で計算できるものであったため使用した。(3)式で求められる値 S はシグマの総和計算により求められた半径 1 の円の面積であり円周率の近似値となる。

2.5 モンテカルロ法によるアルゴリズム

モンテカルロ法とは乱数を用いたシミュレーションを何度も行うことにより近似解を求める計算手法である。

円周率をモンテカルロ法で求める場合、1 辺の長さが 2 の正方形の中からランダムに 1 点を選択し(ランダムな x 座標を y 座標の組を求め)、その座標の正方形の中心からの距離が 1 以下であるかどうかを判断する。ランダムに設定した n 個の点のうち、正方形の中心からの距離が 1 以下のものが k 個だった場合、円の面積は $\pi = 4 \times k/n$ と求められる。

2.6 円周率を計算する MPI プログラム

本研究では、数値積分アルゴリズムおよびモンテカルロ法によるアルゴリズムの 2 つに対して、C++を用いて MPI 上でプログラム化を行った。付録に本研究で作成した MPI プログラムを示す。

以下に本研究で作成した MPI プログラムについて記述する。

[プログラム A: 数値積分による MPI プログラム]

入力：積分範囲 *n*

出力：円周率の近似値

rank とはプロセスの番号であり、size とは総プロセス数である。

入力 n から各プロセスに mpi = MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD)により、ランク 0 から整数をメイン PC 以外に送信。以下の for ループにより、各プロセッサで積分計算を行なう。

```
for (i = rank+1; i<=n; i+= size){/*  各プロセスで(分割された)数値積分を実行*/
                x = h * (i - 0.5);
                sum = sum + 4.0 / (1.0 + x*x);
    }
```

MPI_Reduce(&rankpi,&pi,1,MPI_DOUBLE,/MPI_SUM,0,MPI_COMM_WORLD);により格プロセッサの計算結果をランク 0 に送信する。

[プログラム B: モンテカルロ法による MPI プログラム]

入力：乱数の個数

出力：円周率の近似値

各プロセッサで割り振られた回数、乱数によって決められた点をうつ。

p.x、p.y はそれぞれ座標の x 軸、y 軸である。

```
for (i = 0; i < end-start; i++){
   p.x = (double)rand() / (double)RAND_MAX;
   p.y = (double)rand() / (double)RAND_MAX;
   r = sqrt(p.x * p.x + p.y * p.y);
   length[i]=r;
}
```

その座標の正方形の中心からの距離が 1 以下であるかどうかを判断する。1 以下だった場合、cnt(直径 1 の円内の点の数)を加算していく。これが各プロセッサで計算された半径 1 の四分円内の点の数(面積)sendbuf となる。

```
for (i = 0; i < end-start; i++){
   r=length[i];
   if ( r < 1.0 )cnt++;
}
sendbuf = cnt;
recvbuf =0;
```

MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD)により各プロセスの計算結果 sendbuf の「四分円内の点の数」を計算 (MPI_SUM)し、 0 番のプロセスの変数 recvbuf へ送信する。

上記 2 つのプログラム A,B を MPI 上で実行させるには、コマンドプロンプトを立ち上げ mpiexec コマンドで Microsoft Visual Studio 2008 によりリリースし作成された exe ファイルを指定すると出来る。リリースとは、デバッグとは対になる語であり、最適化された exe ファイルが作成される。

本研究では、上記の 2 つのプログラム A,B を用いて、計算機 1〜3 台を用いて MPI 上で円周率の計算を行った。本研究では、プログラム A に対しては項数 100 の場合と 100 万の場合で計算を行い、小数点第 6 桁目と 1 2桁目まで、プログラム B に対しては生成する点の個数が 2 万個の場合と 2 億個の場合で計算を行い、小数点第 3 桁目と 6 桁目までを求めた。また、プログラム A と同様の手法で Leibniz 和と Wallis 積を用いても計測する。Leibniz の公式により項数 100 の場合と 100 万の場合で計算を行ない、少数第 1 桁目と 5 桁目まで、Wallis の公式により項数 100 の場合と 100 万の場合で計算を行ない、少数第 1 桁目と 5 桁目まで求めた。

3　結果・考察

2.6 節で示した 2 つの MPI プログラム A,B 及び Leibniz の公式のWallis の公式を用いて、MPI 上で円周率を計算した時の計算時間を表 3 に示す。表 3 より、プログラム B の処理時間は計算機台数の増加に伴い入力の違いに依らず時間短縮が見られたことから、MPI による並列計算の有用性を示せている。しかし、プログラム A 及び Leibniz の公式・Wallis の公式の入力が少ない場合には処理時間は計算機台数の増加に伴い逆に処理時間が延びていることが示される。処理時間が延びてしまった事については、計算機台数を増やしたことにより計算機間の通信が増えたためと考えられ、入力による内部計算よりも分散する処理の方が時間がかかったためと考えられる。

このことから、プログラム A 及び Leibniz の公式・wallis の公式については、積分範囲をもっと広げればさらに桁数が上げられると思われる。プログラム B については、乱数により生成される点の数を増やせば桁数が増えると思われる。

表 3　MPI による計算時間と計算機台数の関係

アルゴリズム	項数 生成点数	有効桁数 (少数点以下)	PC（台数）		
			1	2	3
数値積分	項数 100	6	0.000096	0.000215	0.000597
数値積分	項数 100 万	12	0.043853	0.034987	0.029730
モンテカルロ法	生成点数 2 万	3	6.612981	3.875726	3.516391
モンテカルロ法	生成点数 2 億	6	65923.304931	37943.917841	32397.407234
Leibniz 和	項数 100	1	0.000060	0.000089	0.000269
Leibniz 和	項数 100 万	5	0.039402	0.026691	0.023523
Wallis 積	項数 100	1	0.000058	0.000091	0.000328
Wallis 積	項数 100 万	5	0.040141	0.031893	0.024005

(m 秒)

4　結論・今後の課題

本研究では MPI の有効性を検証するために、MPI を用いて円周率の計算を行った。また、本研究の結果からは MPI の有用性が示せたと考えられる。ただしプログラム A 及び Leibniz の公式の wallis の公式の入力が少なかった場合、積分範囲が小さすぎ通信時間の方が大きくなり、また性能の違った複数台の PC を利用したため計測が遅くなったのではないかと思われる。今後の課題としては、入力桁数のさらなる追加による求める桁数の増加を行ないたい。また、円周率を求める他のやり方についても検討していきたい。

謝辞

この研究を卒業論文として形にすることが出来たのは、担当して頂いた石水隆助教の熱心なご指導や、同研究室メンバーのアドバイスや協力していただいたおかげです。協力していただいた皆様へ心から感謝の気持ちと御礼を申し上げたく、謝辞にかえさせていただきます。

参考文献

[1] P.Pacheco 著, 秋葉博訳, MPI 並列プログラム, 培風館,(2001).

[2] 渡邉　真也著, PC クラスタ超入門 2000,
http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/index.html

[3] 奥田洋司,ハイパフォーマンスコンピューティング,
http://nihonbashi.race.u-tokyo.ac.jp/lectures/H18_HPC/

[4] Geist, A., Beguelin, A., Dognarra, J., Jiang, W., Manchek, R. and Sunderam, V.著,
"PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing," The MIT Press, (1994.)

[5] Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D. and McDonald, J.著, "Parallel Programming in OpenMP," Morgan Kaufmann Publishers, (2000.)

[6] 石川裕・佐藤三久・堀淳史・住元真司・原田浩・長谷川篤史・清水正明・亀山豊久著,「Linux で並列処理をしよう －SCore Version 6 で作るスーパーコンピュータ-」, 共立出版 ,(2007)

[7] MPICH2 User's Guide
http://www.mcs.anl.gov/research/projects/mpich2/documentation/index.php?s=docs

[8] LAM/MPI Parallel Computing
http://www.lam-mpi.org/

[9] Edgar Gabriel 他, Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation,(2004)

[10] 野崎昭弘著「πの話」、岩波書店、(1974)

[11] MPICH2：downloads
http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads

[12] πの部屋
http://www1.coralnet.or.jp/kusuto/PI/

[13] 円周率ものがたり
http://www5f.biglobe.ne.jp/~tsuushin/sub1.html

[14] 数学の泉 円周率を求めて, 数泉編集部, 2007
http://www.chikyo.co.jp/maths/pdf/free02.pdf

[15] Argonne National Laboratory ... for a brighter future
http://www.anl.gov/

[16] パソコンで円周率計算の世界記録を更新、フランス, AFPBB News, 2010 年 1 月 10 日
http://www.afpbb.com/article/environment-science-it/science-technology/2681124/5146651

[17] 円周率 5 兆桁、PC で計算 長野の会社員,3 ヵ月かけ, 朝日新聞, 2010 年 8 月 5 日
http://www.asahi.com/science/update/0804/TKY201008040488.html

付録：円周率を計算する MPI プログラム

以下に本研究で用いた円周率を求める MPI プログラムのソースファイルを示す。

(1) 数値積分による円周率計算

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
void main(int argc, char* argv[]){
double rankpi,pi,h,sum,x;
double kaishi,syuuryou;
int n,rank,size,i;
int mpi;
mpi = MPI_Init(&argc, &argv);/* MPI 初期化 (プロセス数、プロセスのランク(番号)等の取得)*/
mpi = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
mpi = MPI_Comm_size(MPI_COMM_WORLD, &size);
if(rank ==0){
        printf("何回計算するか ¥n");
        scanf_s("%d",&n);
        printf("n=%d ¥n",n);
        kaishi = MPI_Wtime();
}
mpi = MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);/* 0 番のプロセスから他の全プロセスに整数を送信*/
h = 1.0/n;
sum = 0.0;
for (i = rank+1; i<=n; i+= size){/* 各プロセスで(分割された)数値積分を実行*/
        x = h * (i - 0.5);
        sum = sum + 4.0 / (1.0 + x*x);
}
rankpi = h * sum;
MPI_Reduce(&rankpi,&pi,1,MPI_DOUBLE,/
                        MPI_SUM,0,MPI_COMM_WORLD); * 各プロセスの計算結果 rankpi の総和を計算 (MPI_SUM)し、 0 番のプロセスの変数 pi へ送信*/
if(rank == 0){/* 0 番のプロセスは結果を表示*/
        printf(" pi: %0.16lf",pi);
        syuuryou = MPI_Wtime();
        printf("時間 = %f¥n",syuuryou-kaishi);
}
MPI_Finalize();
```

```
}
```

(2) モンテカルロ法による円周率計算

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

typedef struct
{
   double x;
   double y;
}POINT;


int main(int argc, char* argv[])
{

   POINT     p;
   double    r;
   double    pi;
   int       cnt;
   int       total_cnt;
   int       i;
   int       rank;
   int       size;
   int       local_cnt;
   int       last_local_cnt;
   int       start, end;
   int       sendbuf, recvbuf;
   int       errorcode=0;
   double    syuuryou,kaishi;
   double *length;

   MPI_Init(&argc, &argv); /* MPI 初期化 (プロセス数、プロセスのランク(番号)等の取得)*/
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   kaishi=MPI_Wtime();
```

```
  local_cnt=(int)((double)20000000/(double)size);
  last_local_cnt=20000000-local_cnt*(size-1);
  if(rank != size-1){
start=rank*local_cnt;
end=(rank+1)*local_cnt;
  }else{
start=rank*local_cnt;
end=20000000;
  }

  length = (double *)malloc(sizeof(double)*(end- start+1));
  srand((unsigned int)time(NULL)*rank);
  cnt = 0;
  for (i = 0; i < end-start; i++){ /*  ランダムに点をうつ*/
    p.x = (double)rand() / (double)RAND_MAX;
    p.y = (double)rand() / (double)RAND_MAX;
    r = sqrt(p.x * p.x + p.y * p.y);
    length[i]=r;
  }

  for (i = 0; i < end-start; i++){
    r=length[i];
    if ( r < 1.0 )cnt++;
  }
  sendbuf = cnt;
  recvbuf =0;

  MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD); * 各プロセスの計算結果 sendbuf の「四分円内の点の数」を計算 (MPI_SUM)し、 0 番のプロセスの変数 recvbuf へ送信*/

  total_cnt =recvbuf;
  pi = 4.0 * (double)total_cnt / (double)20000000;
  if(rank==0){
printf(" pi: %0.16lf",pi);
syuuryou=MPI_Wtime();
          printf("時間  = %f¥n",syuuryou-kaishi);
   }
```

```
    MPI_Finalize();
    return 0;
}
```