卒業論文

PC クラスタ上での OpenMP 並列プログラミング（Ｉ）

氏名　　：　柿下　裕彰
学籍番号　：　２２１０９９００５３－６
指導教員　：　山崎　勝弘　教授

提出日　　：　２００３年2月２１日

立命館大学　理工学部　情報学科

内容梗概

本論文では、本研究室で構築した SCore 型 PC クラスタ上で OpenMP での並列プログラミングについて述べる。そして、並列処理という技術がより身近に感じ、効果のあるものであること述べている。

本研究は PC クラスタで行うが、近年 PC クラスタは PC の高性能化、低価格化、Myrinet などの高速なネットワーク環境が普及してきたことから高性能な PC クラスタの構築が可能になった。PC クラスタは高速な演算が可能、非常に大規模な計算が可能、信頼性の向上や、レンタルでも、1 ヶ月で数千万～１億円以上かかるスーパーコンピュータに比べると、非常に安価で構築できるというメリットがある。

OpenMP のプログラミングでは、マンデルブロー集合、ヴィジュネル暗号、ソーティングのプログラミングを行った。

マンデルブロー集合の計算はブロック分割、サイクリック分割で実行した。両方とも並列効果を得ることができたが、サイクリック分割は負荷均衡がとれ理想的な速度向上を得ることができた。ヴィジュネル暗号も計算量がふえるにつれ並列効果がより得られるようになった。ソーティングプログラムはバブルソートに準じたソートプログラム、バブルソートとマージソートを用いたプログラム、クイックソートとマージソートを用いたプログラムを作成したが、バブルソートに準じたソートは同期の遅延により予想以上に並列効果が得られなかった。それを改善するために作成したバブルソートとマージソートを用いたプログラムは計算量の大幅な短縮により並列効果を得ることができた。さらにクイックソートとマージソートを用いたプログラムは前 2 つの作成したソートプログラムよりさらに処理時間を短縮することができた。

目次

図目次

表目次

1　はじめに

パソコンの性能向上はすさまじい勢いで進んでいる。毎年数回新製品が発表されるが、その度に CPU の周波数もメモリ容量も拡大し、1 年も経てば一昔前の製品と思うくらいの進歩がある。マイクロプロセッサの性能向上はすさまじい勢いで進み、一昔前では非現実的であったことが、いまや現実的になっている。しかし今なお、地球シュミレーター、気象予測、物理・化学（流体計算、遺伝子情報相合探索など）、環境問題、データベース処理、CG によるリアルな画像生成のように、1 台のプロセッサでは実現的な時間内で解決できない問題が多く存在する。このような大規模な問題を並列マシンを用いて高性能コンピューティング、すなわち、並列マシン、並列ソフトウェア、並列アルゴリズム、並列プログラミング、及び並列処理を活用することで、実用的な時間内で解決することができる。

この並列処理は、いまや欠かせない技術の一つであるといえる。近年では、共有メモリ計算機の普及に伴い、並列プログラミングも分散メモリ環境から、共有メモリ環境へと移行しつつある。その共有メモリ用のプログラミングモデルとして現在注目を集めているのが、OpenMP である。OpenMP は移植性が高く、プログラミングも比較的簡単であるということから、今後並列プログラミングの主流になると期待されています。

ここ何年かの間に、並列計算機はずいぶん身近なものとなってきた。様々な高性能並列計算機が製品化され価格性能比も飛躍的に向上している。並列計算のためのプログラミング言語やプログラミング環境も充実してきた。まだ研究室に 1 台というわけにはいかないが、広域ネットワークの発達によって、手元に並列計算機がなくても、遠隔地の並列計算機を利用することが容易になった。かつては、教科書の中の「概念」に過ぎなかった並列計算が、普通のユーザが手軽にプログラムを組んで、実際に動作を確認できるまでになっている。並列プログラミングは実用化の時代に入っているといえるであろう。

その背景として、PC クラスタの台頭があげられる。PC クラスタの台頭のバックグラウンドとして、PC の市場競争による驚異的なプロセッサ能力の向上や、そのような PC が安価で入手可能であること、またネットワーク技術の進化、ハードウェアに依存しないプログラム環境の普及などがある。PC クラスタにより高性能計算がより身近になり、ますます並列処理の技術の発展が進んでいる。本研究室の PC クラスタは SCore 型クラスタであり、また SCASH 等のソフトウェアにより分散共有メモリを実現している。また、Myrinet や Ethernet を用いた高速通信により通信のオーバーヘッドをより少なくした環境である。

本研究では、OpenMP による並列プログラミングにより小規模並列プログラムを作成することで、OpenMP 並列プログラミング技法を会得し、作成した並列プログラミングを用いて 16 台の PC クラスタによる並列効果の測定を行っている。本研究はマンデルブロー集合、ヴィジュネル暗号、ソーティングの問題を対象としている。

第 2 章では、並列処理について、また並列プログラミングということで、使用する PC クラスタ、OpenMP の説明を示す。第 3 章では、マンデルブロー集合の計算、第 4 章では、ヴ

ィジュネル暗号の計算、第５章ではソーティングの計算を行っている。ソーティングはバブルソートに準じたソートのプログラムに始まり、そのバブルソートに準じたソートでの問題を解決するためにバブルソートとマージソートを用いた並列プログラミングを作成し、さらに速さを追及したクイックソートとマージソートを用いた並列プログラミングを作成し、計算の実行結果と、考察を示した。

２　並列処理と並列プログラミング

２．１　並列処理

２．１．１　並列処理の現状とこれから

並列プログラミングが実用化の時代に入ってきているとはいえ、並列プログラミングは、まだ一般的になっているとはいい難い。せっかく高性能並列計算機を導入しても、一部の熱心な研究者や学生が積極的に使っている一方で、潜在的ユーザの大多数がその高性能計算(high performance computing)のパワーを試してみようともしない、というのが現実のようである。逐次のプログラミングと並列プログラミングとの間に、かなり大きなギャップがあると感じるようである。

また、並列方式の高性能計算機(HPC：High Performance Computer)の製品が多数出現している。並列計算機は先端的な分野、先ほど挙げた地球シュミレータや気象予測などで活用され、大きな成果をあげているものの、科学・工学の多くの研究者、技術者が日常的に用いる計算機になっていないのも事実である。

その原因はとして、並列計算機の完成度、環境、教育について考えてみる。並列計算機の完成度については、並列計算機の方式は多様性に富んでいてプログラムは機種依存性が高く、一方で、完全自動並列化コンパイラは現状では難しいこと。環境については、ワークステーションやパーソナルコンピュータはありふれたもので、手軽に使用できる一方、本格的な並列計算機は高価なもので、現状では利用できる環境にある人の数が限られていること。教育については、並列計算機に関する教育や教科書は専門教育用であるのが現状であるということがあげられる。

しかし、ここで述べた原因は改善されつつある。並列計算機の完成度というところは、現在ではこの並列プログラミング言語の標準化が行われ始め、分散メモリ型（非共有）では HPF（High Performance Fortran)、共有メモリ型（分散共有メモリ型を含む）では OpenMP が標準化活動を始めている。環境というところは、PC クラスタによりより身近なところに並列計算機を構築できるようになってきた。教育というところでは、並列プログラミングの啓蒙活動として、情報処理学会ほか主催の並列処理シンポジウム（JSPP）の並列ソフトウェアコンテスト（PSC：Parallel Software Contest）が 1994 年から行われてきた。これまで、大変優秀なプログラムが作成され、中から情報処理学会の論文もうまれている。このようにして、徐々に普及が進んでいるという現状があるといえる。

２．１．２　並列メモリモデル

冒頭でも述べたが、並列処理が広く一般に浸透していないことの理由の一つに、並列計算機の方式は多様性に富んでいてプログラムは機種依存性が高く、また、完全自動化並列コンパイラは現状では難しく、プログラムインターフェースも機種依存の場合が多いということがあげられる。すなわち、アーキテクチャによって使用できる言語が異なるということである。並列計算機のアーキテクチャがプログラムに影響を与える因子としてはメモリモデルがあり、そのメモリモデルは以下で説明する共有メモリ型、分散メモリ型、分散共有メモリ型の 3 種があげられる。

（1）共有メモリ

共有メモリマシンは、複数のプロセッサがメモリバス／スイッチ経由で、主記憶に接続される形態である。このアーキテクチュアを有するシステムのことを SMP（Symmetrical Multi Processor）と呼び、この形態はメモリモデルが最も汎用で、プログラムが組みやすく、逐次処理だけでなくスループットを重視するサーバマシン（逐次プログラム／プロセスを多数処理するマシン）としても適しているという特徴がある。そのため近年ますます市場が増えてきている。このアーキテクチャに即した並列プログラミングライブラリとして、pthread や OpenMP がある。共有メモリモデルを図１に示す。

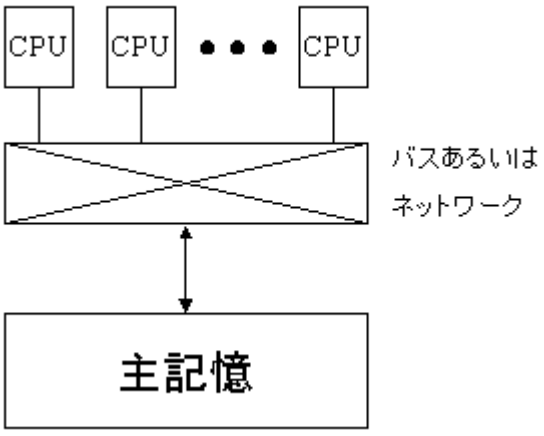

図１：共有メモリモデル

（2） 分散メモリ

プロセッサと主記憶から構成されるシステムが複数個互いに接続された形態であり、大規模なシステムの構築が可能であるという特徴がある。

分散メモリマシンは、プロセッサは他のプロセッサの主記憶の読み書きを行うことはで

きず、必ず相手のプロセッサに介在してもらう必要がある。

この形態のプログラミングモデルは、デットロックさえ気を付ければタイミング依存による嫌なバグが発生することは少ないのだが、通信のプログラミング時にすべてスケジューリングすることはプログラマにとって多大な負担となる。PC や WS を LAN で接続したものもこの形態に属し、代表的な並列プログラミングライブラリとして PVM（Parallel Virtual Machine）や MPI（Massage Passing Interface）などのメッセージ通信ライブラリがある。分散メモリモデルを図２に示す。

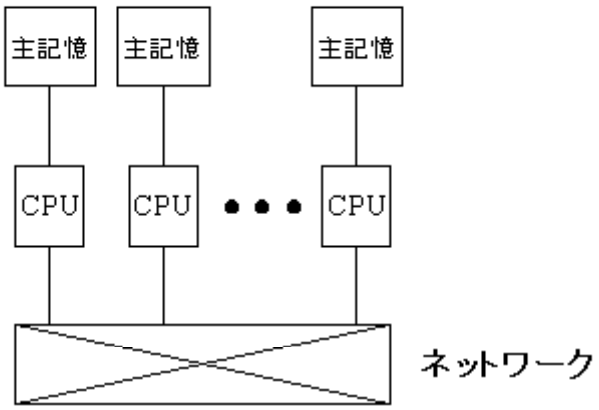

図２：分散メモリモデル

（３） 分散共有メモリ

分散共有メモリマシンと分散メモリマシンのアーキテクチャの差はほとんど減ってきている。とちらも、プロセッサと主記憶から構成されるシステムが複数個互いに接続された形態であり、どちらも大規模なシステムの構築が可能であるという特徴がある。分散共有メモリマシンでは、プロセッサは、他のプロセッサの主記憶を読み書きすることができる。さらにプログラミングモデルにおいては分散共有メモリの方が自由度が高い。しかし、1 個でもバリア同期の場所を間違えるとタイミング依存で非常に解析しにくいバグをつくりこむことになる。最近の PC クラスタは分散共有メモリの形態を有しており、PC クラスタ上でも共有メモリ用並列プログラミングライブラリが利用できるようになっている。
分散共有メモリモデルの図を図３に示す。

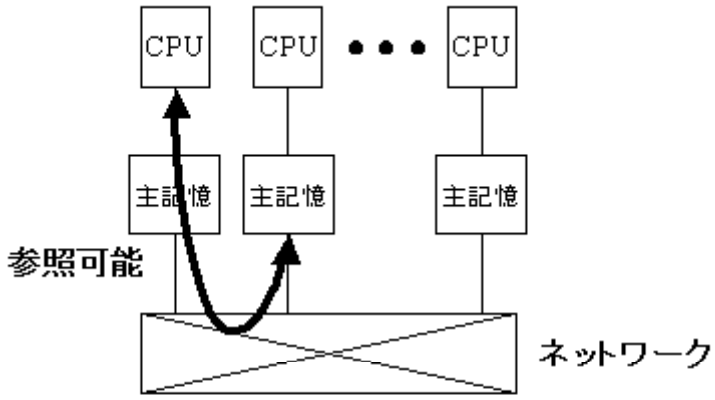

図３：分散共有メモリモデル

２．１．３　並列アルゴリズム

並列アルゴリズムは一般的に大きく４つにわけられる。

（a）プロセッサファーム

まずマスターによって処理を開始する。マスターは与えられた問題に対し複数の独立した計算に分割し、マスターと各スレーブがそれぞれ計算を行い、その結果を再びマスターが回収し結果を得る。このアルゴリズムはマスターとスレッドで独立して計算が行われるので、高い並列効果が得られる。

（b）分割統治

まず問題に対して何らかの計算を行う。そしてその結果をある条件をもとに分割し、またその計算を行う。ある条件が満たされるまで計算と分割を繰り返していき、その部分解を全て統合することによって結果が得られる。

（c）プロセスネットワーク

ある問題に対して計算ステージを複数に分割する。データはあるステージで計算され、それが終わったら次のステージに移り計算される。というふうに各ステージを複数のデータが流れていく。パイプライン処理ともよばれる。

（d）繰り返し変換

まずは与えられた問題をあるオブジェクトに分ける。各オブジェクトは複数の繰り返しの計算により値が変換され、求める値が得られる。その繰り返しはオブジェクトの値があ

る与えられた条件を満たすまで続けられ、また、前のステップで計算された値は自分自身または別のプロセッサ上でランダムに利用される。

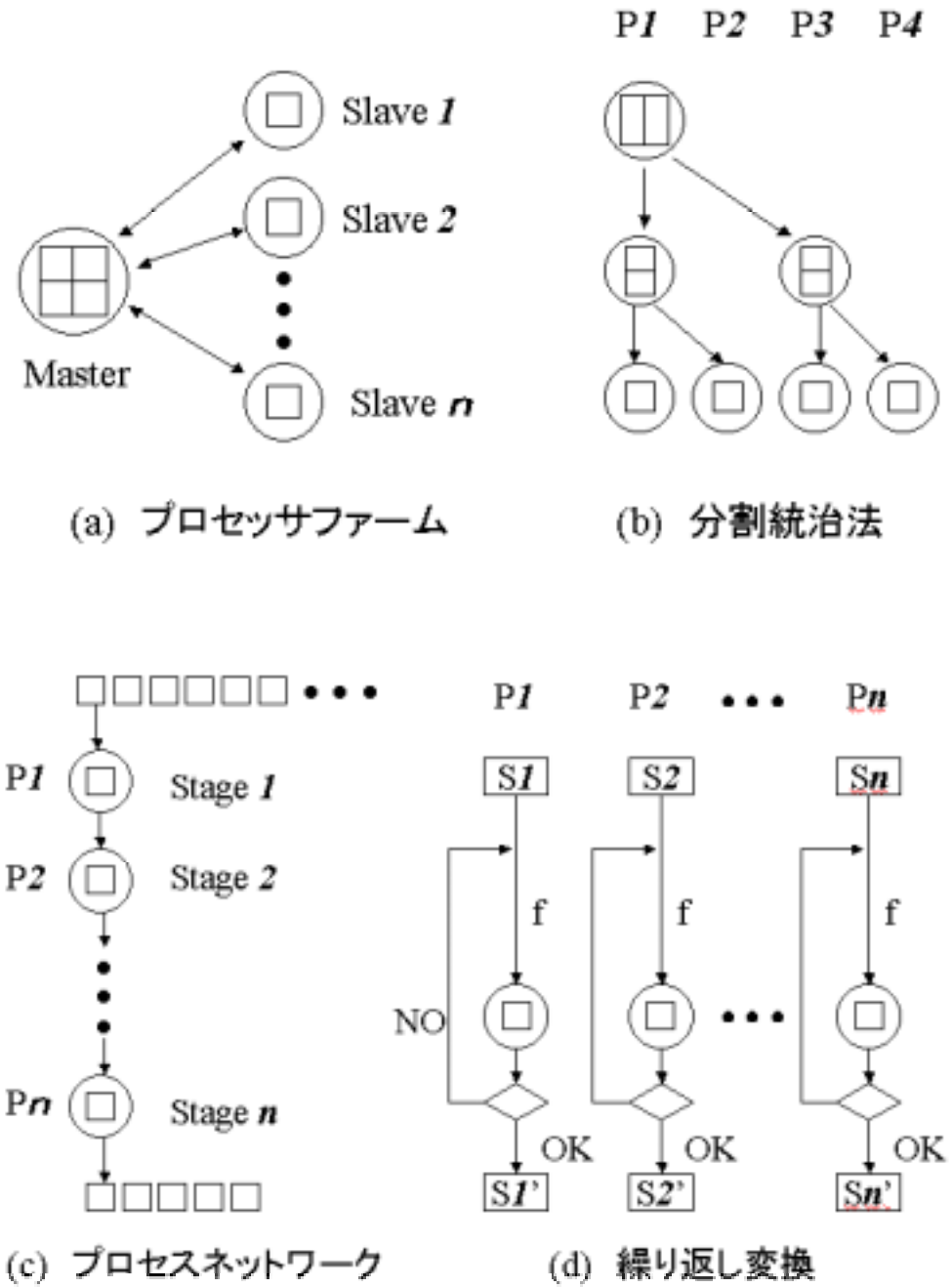

図４：並列アルゴリズムの一般的分類

本論文では、これら 4 つの並列アルゴリズムのうち、高い並列効果が期待できるプロセッサファームと、パイプライン処理のプロセスネットワークを用いて並列プログラムを作

成し実験を行っている。

２．２　PC クラスタ

１　PC クラスタとは

クラスタシステムでは、同じ種類もしくはことなる種類の機能を持つコンピュータを 8 台から 1000 台動作させ、同一作業目的の処理を実行させることで、単体コンピュータでは限界だった信頼性や処理能力の大幅な向上を実現させるシステムを構築することが可能となる。

これは、単体のコンピュータに複数のプロセッサを用意すれば解決するシステムとは異なり、単体のコンピュータがそれぞれネットワークを介して、相互のコンピュータと強調するクラスタリング構成をとることを意味する。これにより、単体のマルチプロセッサシステムよりも高い処理能力や高い可用性を実現することが可能となる。

並列処理でのクラスタは HPC クラスタと呼ばれ、大規模分散処理を主目的としたクラスタとなっている。

PC クラスタは、安価な P.C にフリーの OS を載せ、それを高速のネットワークのネットワークで複数接続し、分散して処理を行うシステムである。PC クラスタは、1990 年前後に、数千から数万台の CPU を搭載する超並列計算機の開発が進む一方で、TCP/IP ベースのネットワークで接続された複数台の計算機を仮想的に一台のマシンとしてとらえて並列プログラミングを走らせるような PVM が開発された。PVM はそれぞれの計算機でメッセージ交換を行うメッセージ通信ライブラリであり、公開されたソフトウェアをインストールするだけで仮想的な並列処理環境が構築できた。これが PC クラスタの幕開けといえる。1995 年前後になると、イーサネットスイッチや 100Mbit Ethernet などの技術も普及し、比較的安価に高性能なネットワークの構築が可能となった。さらに、Myricom 社の Myrinet などクラスタを指向した専用の高速ネットワークが登場した。それにより、専用の並列計算機並みのスループットを持つネットワークの構築が可能となった。一方で、PC 向けのプロセッサの価格低下と急速な性能向上で、コストパフォーマンスに優れた、PC クラスタの実現が可能となった。

２．１　Beowulf 型クラスタ

1994 年に NASA の Earth and space science プロジェクトのために CESDIS（Center of Excellence in Space Data and Information Sciences）で開発された。Linux をベースとした 16 ノードクラスタであり、10Mbps の Ethernet（TCP/IP）を相互結合網として使用した。並列化は PVM と MPI を用いている。Beowulf クラスクラスターの特徴は、市販の PC を使用して、公開されたソフトウェア（Linux）を使うことで、簡単に高性能のクラスタが構築できることである。

２．２　SCore 型クラスタ

SCore は RWC の開発したクラスタリング環境を提供するソフトウェアである。Beowulf のように TCP/IP を用いると、何階層ものプロトコル変換が行われてるために、応用プログラムの多様な資源割り当て要求に対応する事が難しい。そこでオーバーヘッドを減らすために、ユーザレベルから直接ネットワークハードウェアが制御できる zero-copy 通信ソフトウェアの開発が課題となっていた。SCore は、そういった Beowulf 型クラスタの問題点を改善している。本研究室で構築した PC クラスタは SCore 型クラスタである。

SCore のソフトウェア階層を図５に示す。

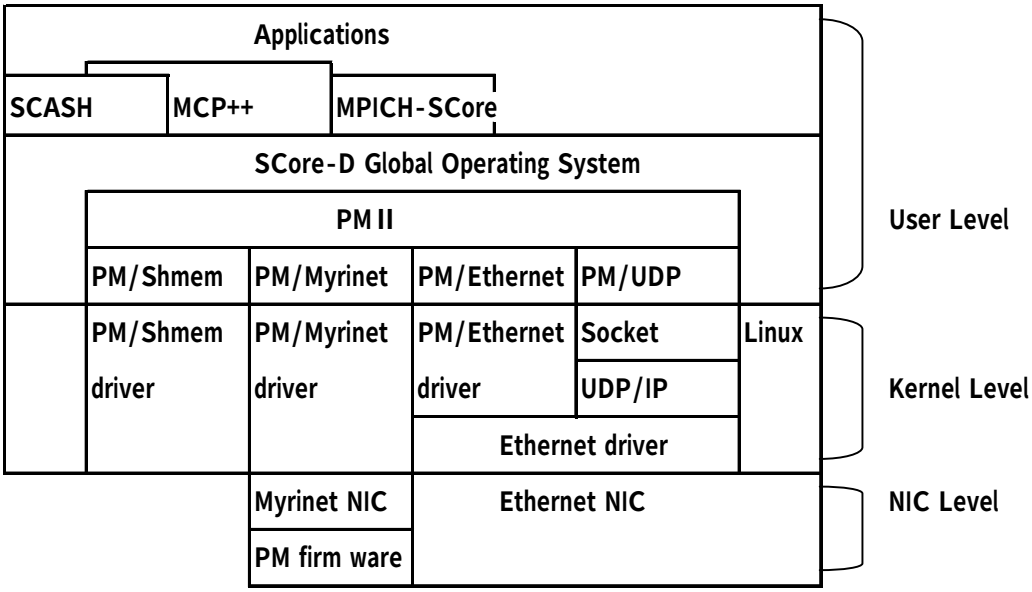

図５：SCore のソフトウェア階層

３　SCore Cluster System Software の特徴

まず、マルチユーザ環境であるというところがある。それは、空きノードでのプロセス実行管理、ギャングスケジュールによる TSS 環境により複数のユーザが共用利用可能であるということである。次に、同じプログラムバイナリで Myrinet でも Ethernet でも利用可能であるというところがある。これは、ユーザの用件に応じて自由なシステムを構築可能であるということでありまた、2 台から 1000 ノードレベルの大規模システムまで構築可能、100Mbps ではバンド幅不足の場合は複数の Ethernet によりバンド幅を上げることも可能である。また、プログラム作成とデバッグは手元の数ノードの Ethernet クラスタ、大規模なプログラム実行はセンターの Myrinet 大規模クラスタで行われる。最後に、長時間ジョブのためのチェックポイントリスタート機構というところがある。これは数週間かかる処理も、ノード故障による被害を最小限に出来るということである。

４　本研究室の PC クラスタ

本研究室の PC クラスタは Compute Host として 16 台、それらを管理する Server Host として 1 台設置する。Ethernet で Server と Cluster16 台(Compute Host)を、Myrinet-2000 で Cluster16 台を接続する。

Server Host は Pentium3 プロセッサの 450MHz、メモリは 512MB の SDRAM であり Compute Host は Pentium3 プロセッサの 500MHz、メモリは 512MB の SDRAM である。それらを接続する Ethernet は最大帯域幅 100Mbps、最小遅延 80μsec であり、クラスタ同士を接続する Myrinet は最大帯域幅２Gbps、最小遅延 9μsec であり、メモリ空間を共有している。

ヴァージョン SCore5.2.0 により SCore 型クラスタを実現し、それに加え SCASH 等のソフトウェアにより分散共有メモリを実現している。OS は RedHat Linux7.3 を使用している。本研究室の PC クラスタ構成図を図６に示す。

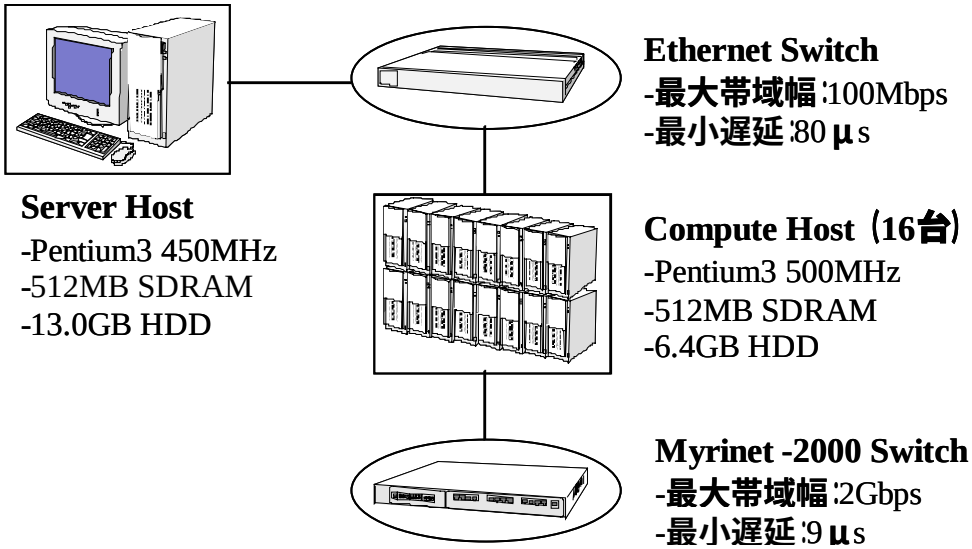

図６：本研究室の PC クラスタの構成

２．３　並列プログラミング

現在、並列プログラミングを記述する方法として以下の方法が挙げられる

（１）コンパイラ指示文を挿入する方法

この方法は既存のプログラムを大きく変更せずに並列化が出来るというメリットがあるが、主にループレベルの並列化に留まり、小規模並列マシン向けの手法となっている。

（２）標準的通信ライブラリ（PVM、MPI など）を用いてプロセッサ間のデータ受け渡し同期を行う方法

この方法は、大規模なメッシュを領域分割して、それぞれの領域をプロセッサに割り当てる並列化手法などで用いられ、分散メモリ型の大規模並列計算機の性能を引き出すことができる。しかしながら、プログラム設計が難しく、また正しく動作することの検証が容易でない。

（３）並列処理向けの様々な言語を用いる方法

これは、並列計算モデルを反映した言語を用いてプログラミングする方法であり、プログラム理論の正しい並列動作が処理系によって保証されているが、並列処理の細かい部分をプログラマが指示できないため、現状では動作するプログラムの性能が不十分である。

２．３．１　分散メモリ並列プログラミング

（１）HPF(High Performance Fortran)

Fortran をベースとした並列言語は、Fortran　D、Vienna　Fortran、PCF-Fortran、CM-Fortran、DAP-Fortran などさまざまなものが提案されてきた。そういったものを統一しようと、Super Computing'91 で HPF の規格原案が生まれた。

HPF2.0 は Fortran95 に基づいており、!HPF$で始まる指示文を挿入することにより並列化が出来るようになっている。HPF は分散メモリ用のプログラム言語であるが、アーキテクチュアに依存しない仕様となっているため、分散メモリでも共有メモリでも仕様可能となっている。

（２）PVM(Parallel Virtual Machine)

PVM はフリーで利用ができ、移植性があるメッセージ通信ライブラリで、大部分はソケットベースで実装してある。

PVM がサポートしているのは、プロセッサが１つのマシンや SMP Linux マシン、ソケットが利用可能なネットワーク（例えば、SLIP、PLIP、イーサネット、ATM）に接続している Linux マシンによるクラスタである。PVM は、プロセッサやシステム構成、使用し

ている物理的なネットワークが異なっているさまざまなマシン構成（異機種間クラスタ）であっても実際に動作する。また、PVM はクラスタ全体に渡って並列に実行しているジョブを制御する機能も持っている。そして、何よりも PVM は長い年月に渡って何の制限も受けずに利用されているため、結果として数多くのプログラミング言語やコンパイラ、アプリケーション・ライブラリ、そしてプログラミングやデバックのためのツール等がある。そしてこれらの成果を「移植性のあるメッセージ通信を開発するためのライブラリ」として使用している。

しかし、PVM のメッセージ通信を呼び出すと標準的なソケット処理の遅延の大きさに加えて、さらにかなりのオーバーヘッドが加わってしまうことに注意が必要である。その上、メッセージを扱う呼び出しそのものがとりわけ「親しみやすい」プログラミング・モデルではないことにも注意が必要である。

（３）MPI(Message Passing Interface)

MPI は規格であり、PVM のようなライブラリではない。現在、分散メモリ型のプログラミングインタフェースとして、最も標準的に用いられている。仕様は、MPIF（The MPI Forum）で検討され、1994 年 6 月に MPI1.0 がまとめられた。その後、並列 I/O やプロセスの動的生成・消滅、1 方向通信（One Sided Communication）などの機能をさらに追加した MPI2.0 の言語仕様が 1997 年 7 月に定められた。

日本では MPICH（MPI CHameleon）が一番普及している。MPICH は MPI1.1 に完全に準拠しており、移植性を考慮して設計してある。LAN と同様に、MPI プログラムがスタンドアローンの Linux システムや UDP や TCP ベースのソケット通信を使った Linux システムで構築したクラスタ上で動作する。しかし MPI が効率的かつ目的に対して柔軟に対応することに重点をおいていることは間違いない。この MPI の実装を移植するには、「チャネル・インターフェース」の５つの関数とパフォーマンス向上のために MPICH ADI(Abstract Device Interface)を実装することになる。

２．３．２　共有メモリ並列プログラミング

（１）スレッド

スレッド並列化記述インタフェースとは、複数の手続きを別々のスレッドで非同期に実行することにより並列処理を実現する枠組みである。並列化制御ライブラリを用いた Fork～Join のプログラミングスタイルとなり、共有メモリマシン用のインタフェースである。ある意味では、データ転送を共有メモリアクセスと同期制御で行うメッセージパッシングととらえることもできる。Unix 上では、pthread ライブラリがこのインタフェースの代表である。また、CRAY や SX などの共有メモリベクトル並列マシンでも「マクロタスキング」として、同様の機能がサポートされている。

（2）OpenMP

OpenMP は、1997 年にアメリカの OpenMP Architecture Review Board が、Fortran をベースとして API の使用で開発したものである。以後、C/C++などにおいても API の仕様で開発された。特徴として、Fortran/C/C++などをベースに、ループ、タスクの並列化部分に指示文を挿入することで、並列プログラムを作成できる。これにより逐次プログラムと並列プログラムを同じソースで管理できる。分散されたメモリを 1 つの共有メモリとして扱えるため、データの受け渡しを考慮しなくて良い。現在、共有メモリ型並列計算機は、広範囲に広がっている。そのため、これらのシステムで容易にプログラムの移植が可能な方法の１つである。

２．３．３ OpenMP

前節で OpenMP について、述べたがここではさらに詳しく説明する。

OpenMP の指示は、データ並列を利用した並列実行を行うループの指定と並列実行を制御する指示から成っている。この他、ユーザが定義したプログラム部分の並列実行の指定も可能である。これらの指示は、Fortran や C の文法の拡張ではなく、コメントあるいはプログマの一部として定義されているため、並列化指示を加えた後でも、逐次実行するようにコンパイル可能である。コンパイラに対する並列化指示以外に、実行時ライブラリと環境変数に関して定義されている。

OpenMP と同様の並列化指示の仕様として、分散メモリマシンを対象にプログラムの並列化を行うために提案された HPF がある。HPF はプログラム中のデータを分散メモリ上に配置する方法を指定し、並列ループの検出などは処理系にまかされている。一方 OpenMP は共有メモリを対象としていることもあり、データ配置に関する指定はなく、並列ループなどの並列実行の指定が中心となっている。HPF は Fortran のみの仕様であり、逐次プログラムをそのまま使えるわけではなく、対象とする並列計算機のアーキテクチャや、プログラムの制御フローなどを考慮しなければならないのに対して、OpenMP は Fortran に加え C/C++にも対応していて、さらに逐次プログラムをそのまま使い、段階的な並列化が可能である。

OpenMP の並列実行は fork-join 型の実行モデルを採用している。このため、プログラム実行中に並列実行部分の指定があった時点で、複数のスレッドを生成し、並列実行部分が終了した時点でマスタ以外のスレッドは消滅して元の１スレッドとなる。

OpenMP では、プログラムの整合性や並列化可能性などは、全てユーザが保証しなければならない。このため、OpenMP コンパイラはループの依存関係やデッドロック等のチェクを行う必要がなく、プログラム中に指示した通りにプログラムの並列化を行うことが可能となっている。OpenMP の指示は、コンパイラに対するヒントとして扱われていた並列指示とは異なり、並列実行を記述するプログラミング言語であるとも言える。

OpenMP が普及してきた背景には共有メモリマルチプロセッサシステムの普及や、共有

メモリマルチプロセッサシステムの並列化指示文の共通化の必要性などが挙げられ、共有メモリマルチプロセッサの並列プログラミングのプログラミングモデルである。OpenMP は新しい言語ではなく、コンパイラ指示文（directives/pragma）、ライブラリ、環境変数によりベース言語である Fortran/C/C++を拡張するのもである。また自動並列化ではなく、並列実行・同期をプログラマが明示することで並列化を行う。したがって、指示文を無視することによって、逐次実行が可能とすることもできる。

OpenMP のアーキテクチャの図７に示す

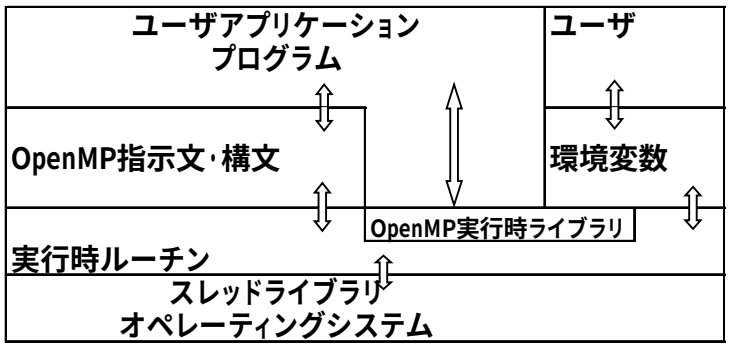

図７：OpenMP のアーキテクチャ

3　マンデルブロー集合の並列プログラミング

3．1　問題定義

マンデルブロー集合とは、以下に示す複素関数で定義される数列 {Zn} が有限であるような複素数 C の集合である。

複素関数 $\left(f(Z): Z_{i+1} = Z_i^2 + C\right)$ に対して、初期値を $Z_0 = (0,0)$ と置き、$Z_1 = f(Z_0), Z_2 = f(Z_1)$, ・・・, $Z_k = f(Z_{k-1})$ のように反復計算を繰り返す。複素平面上の座標値 C の値を変化させ、$Z_k\,(k \to \infty)$ の値が収束か、発散かを求める。
本実験では $(-2.0 \leq$ 実部 $< 0.5), (-1.25 \leq$ 虚部 $< 1.25)$ の範囲の複素平面を $X \times Y$ のメッシュに区切り、$X \times Y$ 個の点について上の反復計算を行う。その結果 $|Z_i| > 2.0$ のとき発散、$i > 200$ の時、収束するとし、後者の結果となる座標点がマンデルブロー集合の要素となる。

複素関数$\left(f(Z): Z_{i+1} = Z_i^2 + C \right)$の$Z$を$x, y$で表すと、

$$Z = x + yi$$

複素平面$\left(f(Z): Z_{i+1} = Z_i^2 + C \right)$の$C$を$x, y$で表すと、

$$C = a + bi$$

以上を複素関数に代入し、複素関数を実部、虚部で分けて表すと

実部：$Z_x = x^2 - y^2 + a$

虚部：$Z_y = 2xy + b$

$i > 200$ でループが抜けた時マンデルブロー集合としてカウントする。

３．２　並列化

並列化は複素平面上のｘ座標の範囲をブロック分割とサイクリック分割で実行する。各プロセッサは自分の受け持つｘ座標の範囲に関して計算を行っていく。分割は以下の図８のようになる。

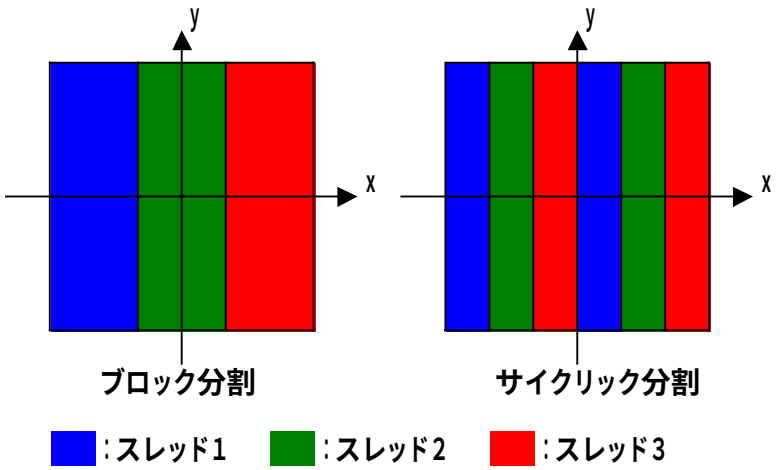

図８：ブロック分割とサイクリック分割

３．３　実行結果

スレッド数をかえたときのブロック分割とサイクリック分割の実行時間を解像度２５０×２５０のときのものを表１に、解像度２５００×２５００のときのものを表２に、解像度２５０×２５０のときの速度向上比を図９に、解像度２５００×２５００のときの速度向上比を図 10 に示す。

表１：マンデルブロー集合の実行時間（解像度２５０×２５０）単位：秒

スレッド数	1 台	2 台	４台	8 台	16 台
ブロック分割	0.43 (1.00)	0.36 (1.20)	0.20 (2.15)	0.12 (3.58)	0.072 (5.97)
サイクリック分割	0.32 (1.00)	0.16 (1.97)	0.082 (3.37)	0.043 (7.37)	0.026 (12.28)

（　）内は速度向上比

表２：マンデルブロー集合の実行時間（解像度２５００×２５００）単位：秒

スレッド数	1 台	2 台	4 台	8 台	16 台
ブロック分割	45.2 (1.00)	36.2 (1.25)	19.7 (2.29)	12.2 (3.7)	6.35 (7.12)
サイクリック分割	31.9 (1.00)	16.0 (1.99)	8.01 (3.98)	4.01 (7.96)	2.01 (15.9)

（　）内は速度向上比

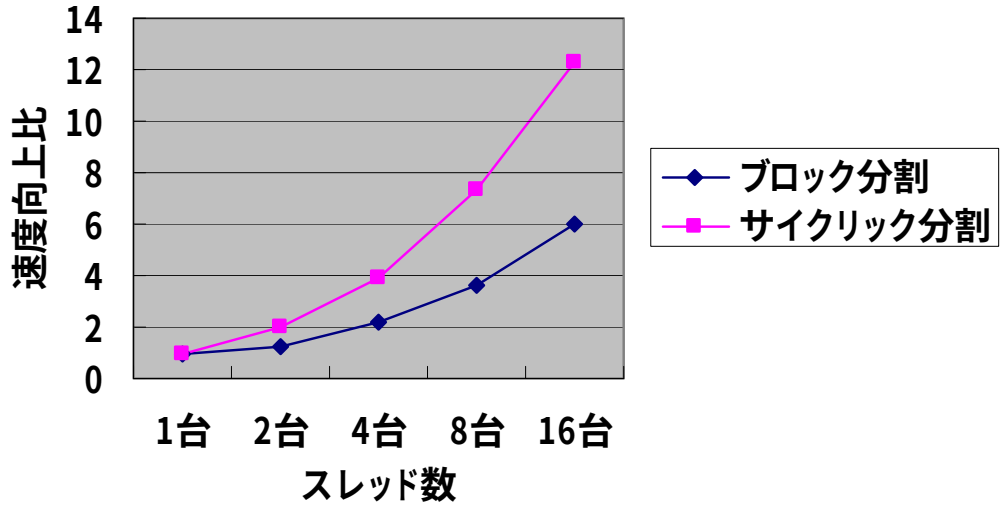

図９：マンデルブロー集合の速度向上比（解像度２５０×２５０）

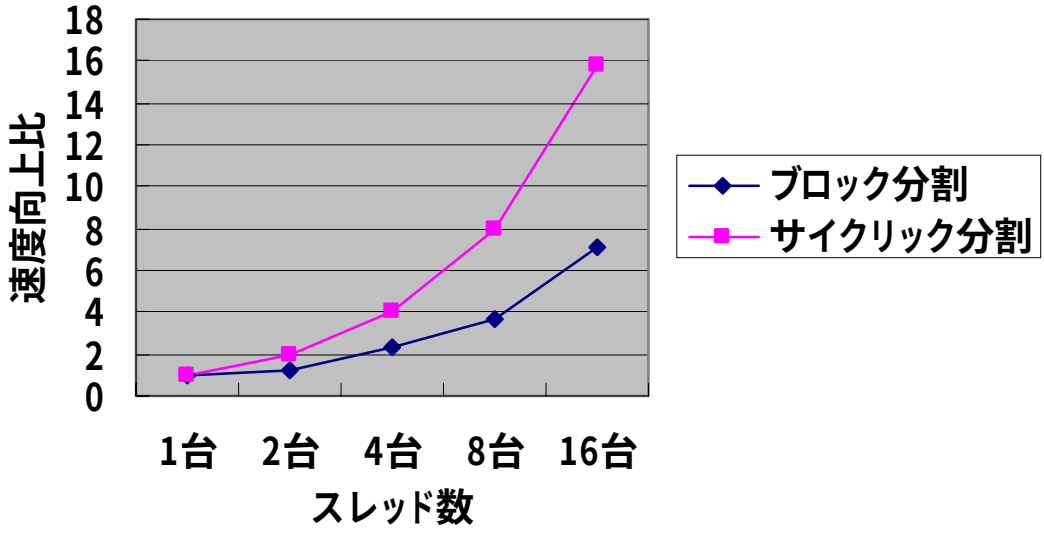

図 10：マンデルブロー集合の速度向上比（解像度２５００×２５００）

３．４　考察

ブロック分割もサイクリック分割も並列効果を得ることができた。特に、サイクリック分割では理想的な並列効果を得ることができました。各スレッドでの計算量が大きく異なれば、プログラム全体の処理時間も、もっとも遅いスレッドに引っ張られることになる。図８のようにサイクリック分割をすることで負荷が大きい部分を各スレッドに分配し、各スレッドが担当する仕事の量をうまく分配できたことにより、全スレッドほぼ同時に仕事を終えることができたからこのような結果を得ることができた。つまり負荷均衡がとれているため理想的な速度向上を得ることができた。故に、ブロック分割とサイクリック分割が大きく違うことからマンデルブロー集合は偏った集合だということになる。

４　ヴィジュネル暗号の並列プログラミング

４．１　問題定義

暗号化は、任意の情報を、ある規則に従って全くの別の情報に変換し、他者に内容を知られないようにする技術である。ヴィジュネル暗号（Vigenere cipher）はその手法の１つで、短いキーワードを繰り返し用いて各ステップごとに、鍵の英字の番号と平文の英字の番号の和を暗号文の英字の番号とする。
例を図 11 に示す。

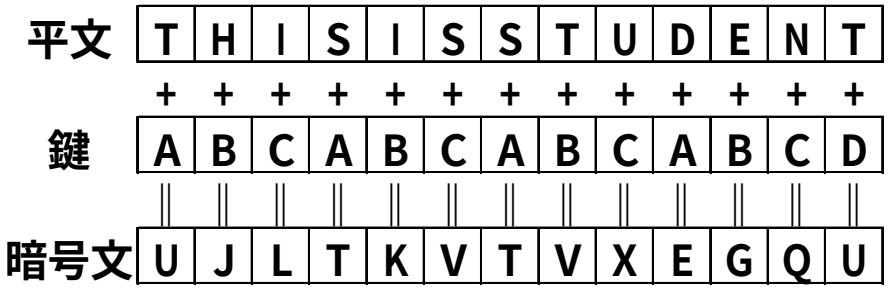

図 11：ヴィジュネル暗号

４．２　並列化

ヴィジュネル暗号は、暗号化する文字列の各文字ごとに、キーワードの対応する文字を用いて、独立に処理するので、分割対象はソースデータである平文（もとの文）とし、各文字の暗号化に要する計算量は、どの文字も同じなのでブロックによる分割を適用する。図 12 にヴィジュネル暗号の分割を示す。また、キーワードを繰り返して用いるとき、スレッド間にキーワードがまたがることがあるので、各スレッドでは暗号化の処理を始める前に、そのスレッドでキーワードの何文字目から使用するのかを計算する必要がある。

しかしこの問題は各文字を配列に格納してから処理することでクリアできる。各スレッドが担当する範囲を指定し、各文字が何番目の文字かを明示することで解決できる。

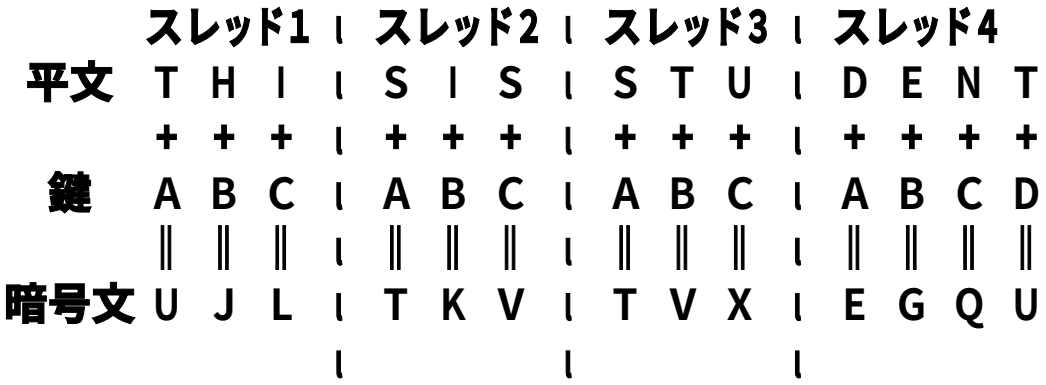

図 12：ヴィジュネル暗号の並列化

４．３　実行結果

今回の実験では１０万文字、１００万文字、５００万文字の平文を並列処理で暗号化した。文字が増えるたび、つまり負荷が多くなるにつれ並列効果を出すことができた。

スレッド数を変えたときの実行時間を表３に、速度向上比を図 13 に示す。

表３：ヴィジュネル暗号の実行時間　　単位：秒

スレッド数	1 台	2 台	4 台	8 台	16 台
10 万文字	11.2 (1.00)	10.8 (1.04)	7.6 (1.48)	6.3 (1.78)	6.1 (1.83)
100 万文字	112 (1.00)	102 (1.10)	66.6 (1.69)	47.4 (2.37)	31.4 (3.58)
500 万文字	578 (1.00)	501 (1.15)	232 (2.49)	160 (3.60)	138 (4.20)

（　）内は速度向上比

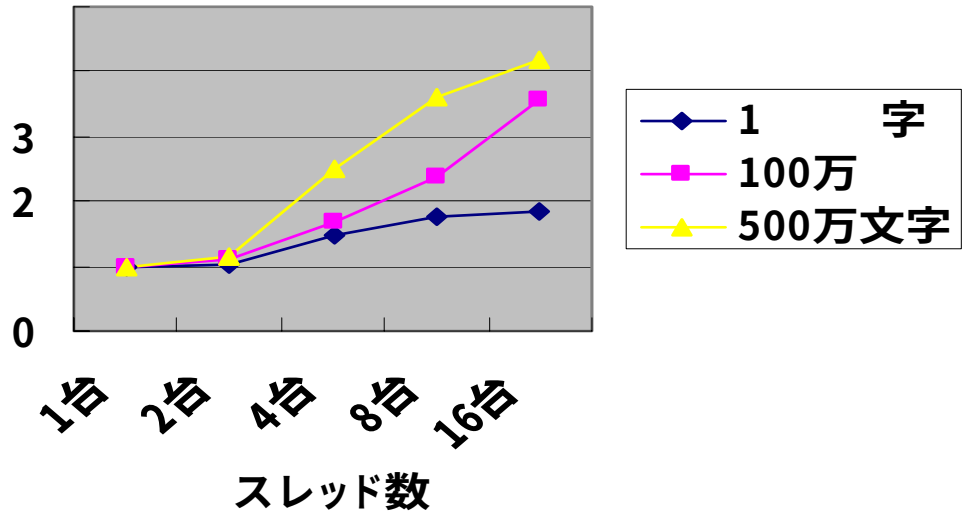

図 13：ヴィジュネル暗号の速度向上比

４．４　考察

ヴィジュネル暗号は、10 万文字、100 万文字、500 万文字と計算量を増加させ、各スレッドにかかる負荷が増えるにつれより並列効果を得ることができた。各文字を配列に格納してから暗号化するためメモリの容量が多くする必要があった。文字定数を使うことで、配列の使用を最小限にし、より多くの文字数でできるようにプログラムを作成した。作成したプログラムでは 1000 万文字まで並列化処理できることは確認済である。

５　ソーティングの並列プログラミング

５．１　バブルソートに準じたソートの並列プログラミング

（１）問題定義

このプログラムは基本的にバブルソートの考えを用いてる。バブルソートは、ｎ個のデータを昇順にソートする場合、まず最初にｎ番目のデータと n-1 番目のデータを比較し、前者が後者より小さければ入れ換えを行い、そうでなければ何もしない。次は n-1 番目のデータと n-2 番目のデータを比較し…と続き、最後に、2 番目と 1 番目の比較・交換を行う。このように、最小の値を持つデータ（バブル）が、泡が水中を浮きあがるように移動していく。次のステップは前と同じように n 番目と n-1 番目のデータを比較を行い、前回入れ換えを行った最後の位置の直前、つまり前回の最後の入れ換えが、K 番目と K-1 番目の間で行われていたら、次回は K+1 番目と K 番目のところまで検索を繰り返す。これを入れ換えが起こらなくなるまで繰り返す。

（２）並列化

ｎ個のデータをいくつかのブロックに分割し、それを各スレッドに割り振る。それぞれに与えられたブロックの中で比較・交換を行う。まず１番後のスレッドで最小の値を後ろから 2 番目のスレッドの最後の値と比較し、1 番後ろのスレッドでの最小の値の方が小さければ交換する。その比較・交換後から 2 番目のスレッドで比較・交換を行い、最小の値を出す。そして、その値を後から 3 番目のスレッドの最後の値と比較・交換する。この動作を繰り返す。このように、バブル（データ）の流れと共にスレッドは処理を行う。そしてスレッド間で比較・交換するときは同期を計る必要がある。図 14 にバブルソートに準じたソートの並列化を示す。

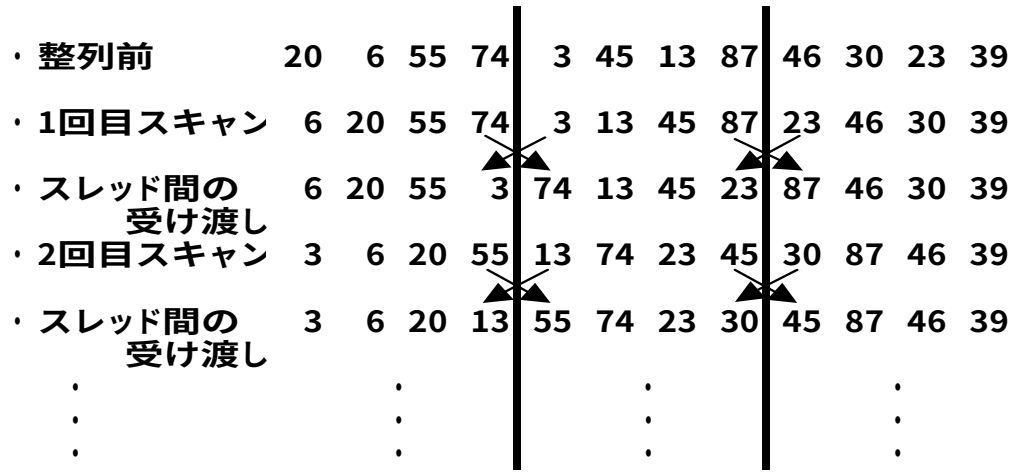

図 14：バブルソートに準じたソートの並列化

（３）実行結果

今回の実験はデータが 5 万個、10 万個、15 万個のソートの並列化を行った。スレッド数を変えたときの実行時間を表４に、速度向上比を図 15 に示す。

表４：バブルソートに準じたソートの実行時間　　単位：秒

スレッド数	1 台	2 台	4 台	8 台	16 台
5 万個	101.8 (1.00)	148.1 (0.69)	131.2 (0.78)	95.41 (1.07)	79.7 (1.28)
10 万個	466.2 (1.00)	1303 (0.36)	725.8 (0.65)	288.6 (1.62)	180.3 (2.59)
15 万個	1227 (1.00)	2520 (0.49)	1700 (0.72)	959.2 (1.28)	574.0 (2.14)

（　）内は速度向上比

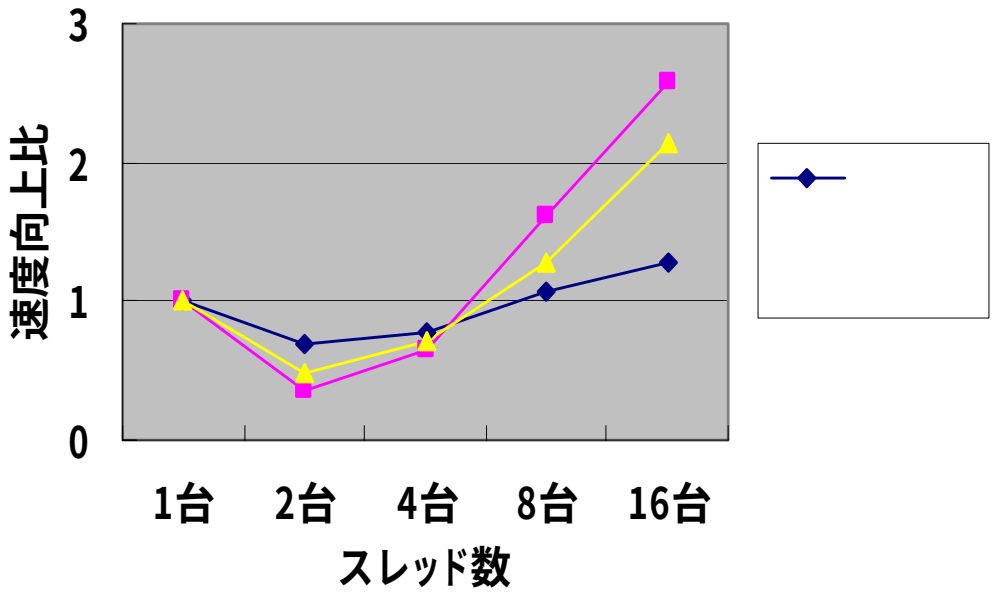

図 15：バブルソートに準じたソートの速度向上

（4）考察

速度向上は 2 台目、4 台目のときは並列化したことで遅くなってしまうが、8 台目、16 台目へとスレッドを増やしていくと並列効果を得ることができた。スレッド 2 台、4 台のときに並列処理することでかえって遅くなる理由は、毎回スキャン後にスレッド間の受け渡しをするための同期を計らなければならないからである。その同期のために遅延するが、この遅延時間が実行時間が短縮する以上にかかってしまうので、かえって遅くなってしまった。さらにこのプログラムはバブルソートの並列化プログラムとは違い、比較・交換が終了して、ソートし終えたところもさらに比較・交換するため、無駄に処理時間をかけている。次節では、この受け渡し毎に発生してしまう同期の遅延を解決するためのプログラムを作成した。

５．２　バブルソートとマージソートを用いた並列プログラミング

（1）問題定義

前節のソートの結果より、処理時間が短くなったといえど、プロセスネットワークというアルゴリズムで考えているため、前節のソートにおいてはスキャンする毎に同期をとるため、同期による遅延が無駄になってしまう。そこで、一番並列効果が出やすいプロセッサファームというアルゴリズムを使ったアルゴリズムを考えた。

バブルソートをただ分割し、各スレッドでソートして値を返すだけではソートされた部分がスレッド数分あるだけで、全体としてソートできない。

そこで整列された数列をソートすることに適しているマージソートを使うことにより並列効果が得られる可能性が高いのでバブルソートとマージソートを用いた並列プログラミングを作成した。

（2）並列化

ｎ個のデータをいくつかのブロックに分割し、それを各スレッドに割り振る。それぞれに与えられたブロックの中で比較・交換を行う。つまり、いくつかのブロックに分割し、各ブロックでバブルソートを行う。そして、いくつかの整列されたデータが作成される。作成されたデータを用いマージソートを行う。マージソートも４つ以上のデータがあるときは並列にソートを行う。バブルソートとマージソートを用いたプログラムの並列化を図 16 に示す。

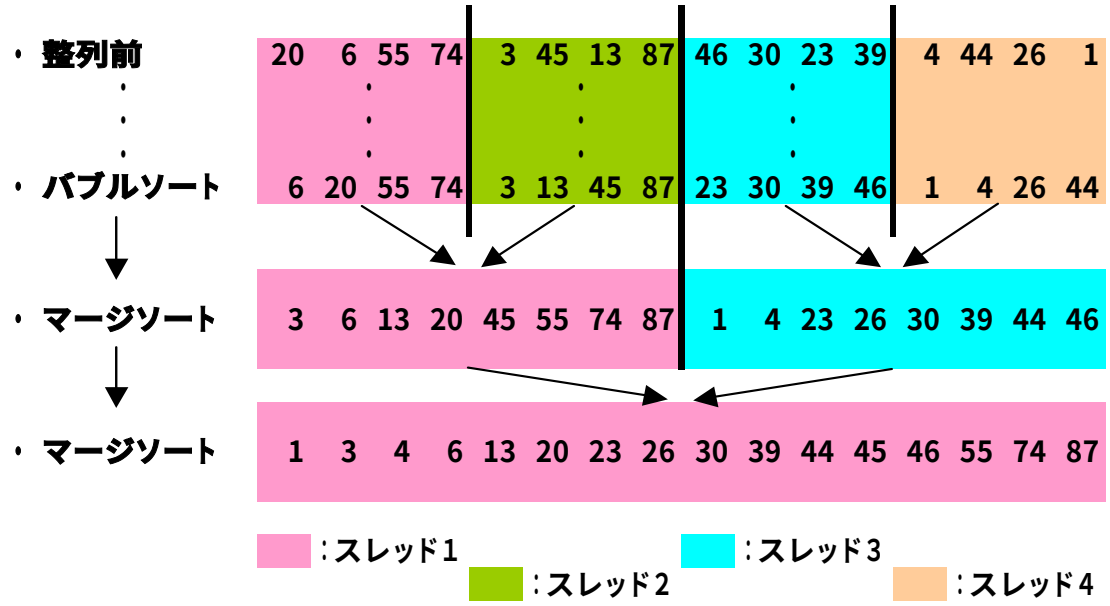

図 16：バブルソートとマージソートを用いた並列化

（3）実行結果

今回の実験はデータが 5 万個、10 万個、15 万個のとき、バブルソートとマージソートを用いたソートの実行を行った。スレッド数を変えたときの実行時間を表 5 に、速度向上比を図 17 に示す。

表 5：バブルソートとマージソートを用いたソートの実行時間　　単位：秒

スレッド数	1 台	2 台	4 台	8 台	16 台
5 万個	65.48 (1.00)	16.37 (4.00)	4.106 (15.9)	1.037 (63.1)	0.2968 (220.6)
10 万個	275 (1.00)	65.6 (4.19)	16.41 (16.8)	4.156 (66.2)	1.098 (250.4)
15 万個	654.3 (1.00)	154.7 (4.23)	37.7 (17.4)	9.34 (70.1)	2.45 (267.1)

（　）内は速度向上比

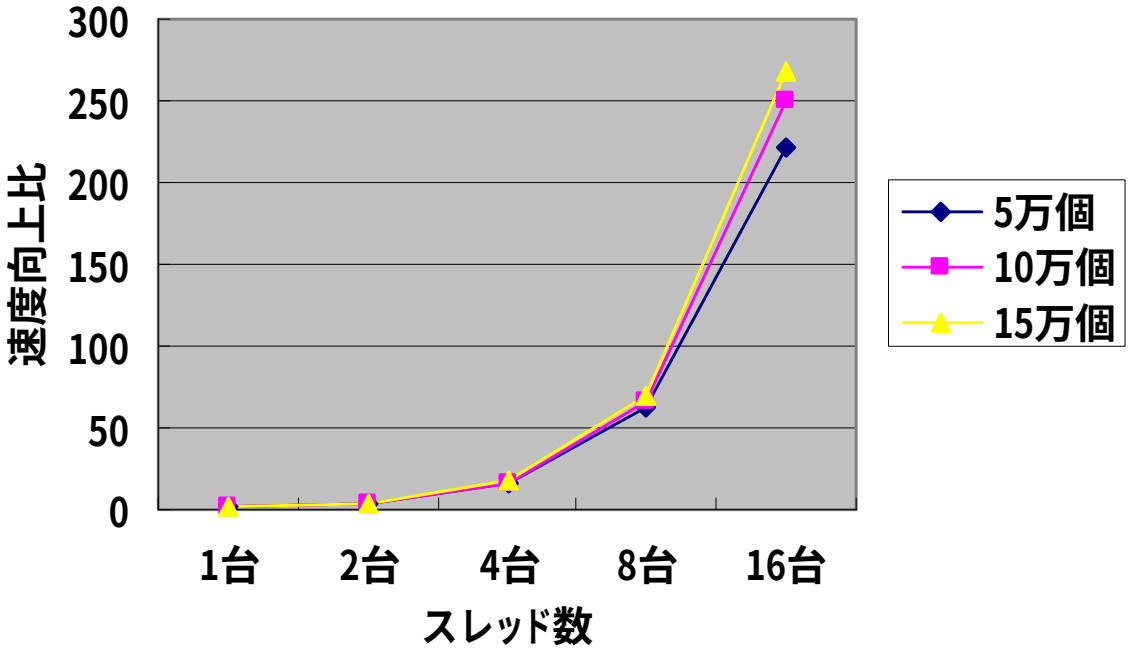

図 17：バブルソートとマージソートを用いたソートの速度向上比

（4）考察

このプログラムはスレッドが 1 台の場合なら普通の逐次のバブルソートとおなじである。スレッドが 2 台以上になると、バブルソートを行った後、マージソートをすることになる。

データが 5 万個、10 万個、15 万個と計算量が増えるに従って並列効果もより得ることができるようになった。5 万個、10 万個、15 万個のどの場合においてもスレッド数を大幅に越える速度向上を得ることができた。このような大幅な速度向上を得ることができた理由はバブルソートとマージソートの計算量に起因するものである。

逐次処理のバブルソート計算量は$O(n^2)$、マージソートの計算量は$O(n\log n)$である。並列に処理する場合、各スレッド数のときのバブルソートの計算量は以下のようになる。

スレッド数 1 台の時の計算量＝$O(n^2)$

スレッド数 2 台の時の計算量＝$O\left(\left(\frac{n}{2}\right)^2\right)=O\left(\frac{n^2}{4}\right)$

スレッド数 4 台の時の計算量＝$O\left(\left(\frac{n}{4}\right)^2\right)=O\left(\frac{n^2}{16}\right)$

スレッド数 8 台の時の計算量＝$O\left(\left(\frac{n}{8}\right)^2\right)=O\left(\frac{n^2}{64}\right)$

スレッド数 16 台の時の計算量＝$O\left(\left(\frac{n}{16}\right)^2\right)=O\left(\frac{n^2}{256}\right)$

このことより、まず最初に行われるバブルソートにより各スレッドが担当する範囲だけを並列にソート処理することでスレッド数分の処理時間の短縮ではなく、スレッド数の二乗分の計算量の短縮が行われることになる。さらにその後行われるマージソートは$O(n\log n)$の計算量がかかるのではなく、逐次のマージソートならまず行われる分割していく作業がすでに行われているので分割作業をする必要がない上、スレッド 2 台の場合なら最後の 2 つの整列された配列をマージソートするだけの状態からのソートになるので大幅な処理時間の短縮ができていることになる。スレッド 4 台なら最後の 4 つの整列された配列をマージソートするのだが、逐次なら 2 回のソートを行って、2 つ整列された配列をつくるのだが、それも並列で行うため、処理時間は 1 回分の時間しかかからない。このようにスレッドを増やせば、バブルソートで時間の大幅な短縮ができ、その後のマージソートもソート部分の後半を並列に処理できるので処理時間の短縮を行うことができる。

このようにあるプログラムを並列化し、速度向上を得るだけでなく、目的に応じて複数のプログラムを融合させることや、また逐次とは全く違うプログラムを作成することでより速度向上を得ることができるということを実感することで理解できた。

さらにソートの高速化を考えると前半の、バブルソートをクイックソートにするとより高速にソートできるのではないかと思い、次節ではクイックソートとマージソートを用いたプログラムを作成した。

５．３　クイックソートとマージソートを用いた並列プログラミング

（１）問題定義

バブルソートとマージソートを組み合わせた並列プログラムによりバブルソートの計算量をうまく減らすことによって大きく並列効果をあげることができた。しかし、各スレッドでバブルソートを行うより、クイックソートを使うほうがより処理時間を減らすことができる。各スレッドで個々にクイックソートを行えばいいだけなので、アルゴリズムも容易にすることができる。

（２）並列化

クイックソートとマージソートを用いた並列プログラムはバブルソートとマージソートを用いた並列プログラムと同様の考え方で、ｎ個のデータをいくつかのブロックに分割し、それを各スレッドに割り振る。それぞれに与えられたブロックの中で整列を行う。つまり、いくつかのブロックに分割し、各ブロックでクイックソートを行う。そして、いくつかの

整列されたデータが作成される。作成されたデータを用いマージソートを行う。マージソートも４つ以上のデータがあるときは並列にソートを行う。クイックソートとマージソートを用いたプログラムの並列化を図 18 に示す。

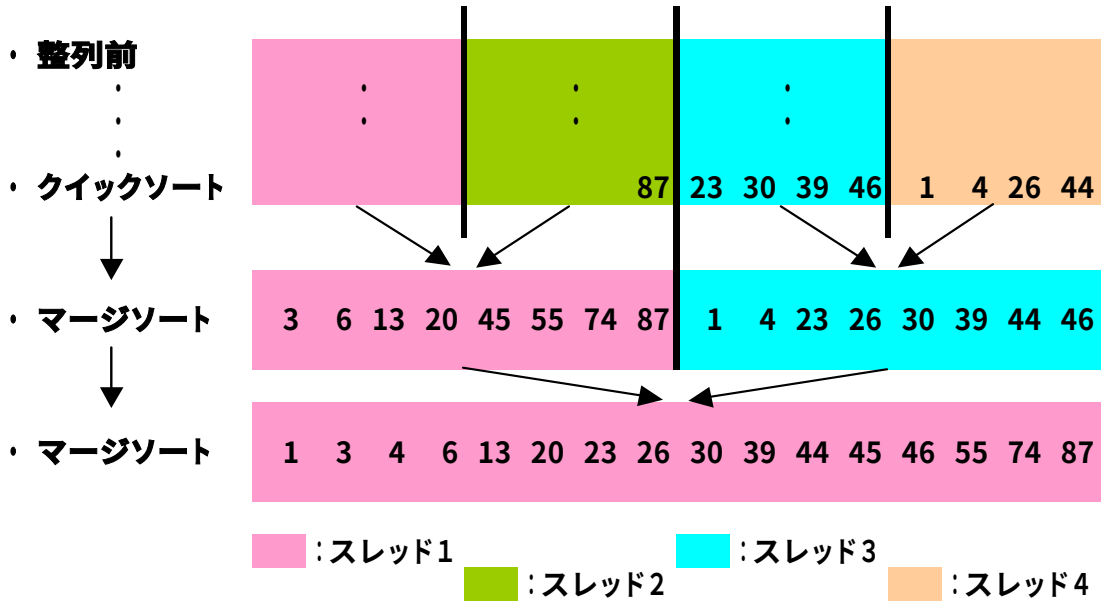

図 18：クイックソートとマージソートを用いた並列化

（３）実行結果

今回の実験は要素が 10 万個、100 個、300 万個、500 万個のとき、クイックソートとマージソートを用いたソートの実行を行った。スレッド数を変えたときの実行時間を表６に、速度向上比を図 19 に示す。

表６：クイックソートとマージソートを用いたソートの実行時間　　単位：秒

スレッド数	1 台	2 台	4 台	8 台	16 台
10 万個	0.0811 (1.00)	0.0754 (1.08)	0.0860 (0.94)	0.0984 (0.82)	0.1105 (0.73)
100 万個	1.101 (1.00)	0.8898 (1.24)	0.9088 (1.21)	0.9947 (1.11)	1.092 (1.01)
300 万個	3.582 (1.00)	2.846 (1.26)	3.156 (1.13)	3.414 (1.05)	3.620 (0.99)
500 万個	6.171 (1.00)	4.909 (1.26)	4.755 (1.3)	5.083 (1.21)	5.461 (1.13)

（　）内は速度向上比

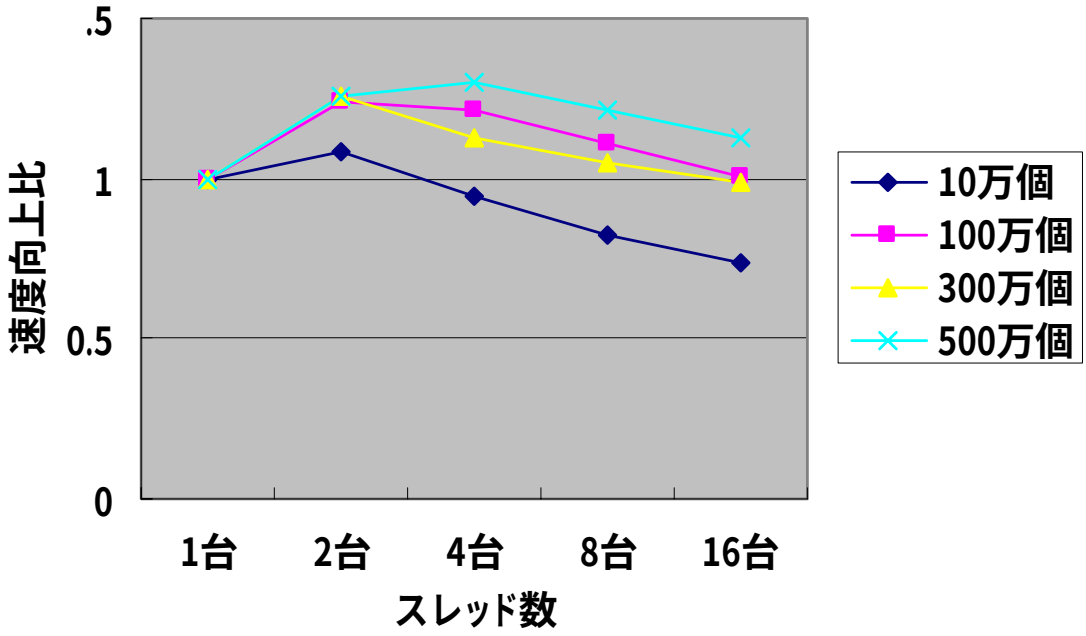

図 19：クイックソートとマージソートを用いたソートの速度向上比

（4）考察

スレッド数が 1 台のときは、マージソートを行わず、クイックソートだけ行われるようになっている。このプログラムは要素の数が 10 万個、100 万個、300 万個のときはスレッド数が 2 台のときが最速になり、要素が 500 万個のときはスレッド数が 4 台のとき最速となった。これはまず行われるクイックソートで、各スレッドが担当する範囲だけを並列にソート処理することでスレッド数分速度が速くなる。そしてバブルソートとマージソートを用いたプログラムと同様に、その後マージソートを行う。しかしスレッド数が増えるとクイックソートで処理時間を短縮した分に対して、マージソートにかかる処理時間があまり小さくなく、スレッドを増やすと逆にクイックソートで短縮した時間よりもマージソートにかかる時間の方が多いので処理時間が大きくなってしまうということになってしまった。

５．４　３つのソートプログラムに対しての考察

バブルソートに準じたソートの並列プログラムもバブルソートとマージソートを用いた並列プログラムもクイックソートとマージソートを用いた並列プログラムも最高処理速度を出せるスレッドの数は違うが、並列効果を得ることはできた。特にクイックソートとマージソートを用いたプログラムはスレッド数 1 台のときは逐次のクイックソートをしているので、スレッドを増やし処理時間を短縮できたことで、並列処理によって、逐次で最速と言われるクイックソートよりも速くソートすることができることを実験を通して実現できた。

６　おわりに

本研究では、OpenMP による並列プログラミングを行い、OpenMP 並列プログラミング技法を増やし、また本研究室で構築した SCore 型 PC クラスタ上で作成した並列プログラムを実行し、PC クラスタの動作テスト及び、測定を行った。
PC クラスタは SCore 等のソフトウェアを利用することにより分散共有メモリ型並列計算機を実現しているが、動作テストによりソフトウェア分散共有メモリ SCASH の制約があることがわかり、プログラミングに戸惑ったところもあった。ソフトウェアで共有メモリを実現するために、とれだけメモリの領域を必要としているのか、また、どのような制約があるのかということをよく理解してプログラミングをする必要があると思われる。

今後の課題としては、クイックソートの並列化、マージソートの並列化、バブルソートの並列化がまず挙げられる。今回の実験では並列アルゴリズムであるプロセッサファームを中心とした並列プログラミングがほとんどであったので、他の並列アルゴリズムでの並列プログラミングをし、並列効果をあげることでさらなる OpenMP 並列プログラミング技法を習得し、より大規模な並列プログラミングに挑戦していくことも課題であると考えている。

謝辞

本研究の機会を与えてくださり、数々の助言を頂きました山崎勝弘教授、小柳滋教授に心より感謝をいたします。また、本研究にあたり、励ましの言葉や様々な貴重なご意見をいただき、また質疑に答えていただくなどした本研究室の皆様に心より感謝いたします。

参考文献

[1] 湯浅太一、安村道晃、中田登志之：はじめての並列プログラミング,bit 別冊,、共立出版、1998

[2] 佐藤三久：JSPP'99 OpenMP チュートリアル資料、RWPC、2000

[3] OpenMP C/C++ API 日本語版、RWPC、2000

[4] R.Chandra, L.Dagum, D.Kohr, D.Maydan, J.McDonald, R.Menon：“Parallel Programming in OpenMP”、MORGAN KAUFMANN PUBLISHERS、2000

[5] 三木光範 他：“PC クラスタ超入門 2000～PC クラスタ型並列計算機の構築と利用～”、超並列計算研究会、2000

[6] NetLaboratory：“RWCP での PC クラスタへの取り組みと構築のポイント、http://venus.netlaboratory.com/pcc/score/rwcp.html

[7] 近藤嘉雪：“C プログラマのためのアルゴリズムとデータ構造”、ソフトバンク、1999

[8] 林晴比古：改訂新 C 言語入門（ビギナー編、シニア編）、ソフトバンク、2001

[9] 山崎勝弘：“コンピュータは進化する”、1999

[10] 黒川耕平：“PC クラスタ上での OpenMP 並列プログラミング（Ⅱ）、立命館大学理工学部情報学科卒業論文、2003

[11] 三浦誉大：“PC クラスタ上での並列プログラミング環境の構築”、立命館大学理工学部情報学科卒業論文、2002

[12] 大村浩文：“PC クラスタの動作テストと OpenMP 並列プログラミング”、立命館大学理工学部情報学科卒業論文、2002

[13] 内田大介：“OpenMP による並列プログラミング1”、立命館大学理工学部情報学科卒業論文、2000

[14] 土屋悠輝：“OpenMP による並列プログラミング2”、立命館大学理工学部情報学科卒業論文、2000

[15] 安藤彰一：“事例ベース並列プログラミングシステムの構築と評価”、立命館大学理工学部情報学科修士論文、1997

[16] 古川智之、松田浩一、安藤彰一：“並列プログラム事例集”、/common/jirei/all/caseset、1996

[17] 米田健治、徳山美香、青地剛宙：“並列プログラム事例集2”、/common/jirei/caseset2、1999