卒業論文

PCクラスタ上でのOpenMPによるマンデルブロ集合の並列化

氏名　　：　多田　修司
学籍番号　：　２７３１０００００１３－１
指導教員　：　山崎　勝弘　教授
提出日　　：　２００５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クラスタ上でのOpenMPによるマンデルブロ集合の並列化について述べる。この研究により得た結果から並列処理の性質やもたらされる効果、またOpenMPの移植性について説明し述べている。

本研究では、OpenMPを使った並列処理でマンデルブロ集合のプログラミング行った。マンデルブロ集合の計算はブロック分割、サイクリック分割で実行する。その後、マンデルブロ集合の近似条件を変え、条件前の結果と比較し、その考察として負荷均衡やオーバーヘッドの問題を取り上げて説明した。全ての条件から並列効果を得ることができた。この実験結果で得られたのは、ブロック分割よりもサイクリック分割のほうが各プロセッサに対して負荷均衡を図ることができる。つまり、多く細かく分割すればより負荷均衡が図れるということである。しかしながら、この実験では負荷均衡が図られているにもかかわらず、格子状サイクリック分割で並列化した実験では速度向上が得られなかった。これはサイクリック分割よりもさらに細かく分割したことで起こったオーバーヘッドが原因である。各プロセッサの短縮された計算時間より、細かく分割するための計算時間が多くなったことでサイクリック分割より効果が得られなかったということである。この研究では処理するデータによって最適な並列プログラムがあり、予測できる限りデータに合った並列プログラムを考えることが速度向上に必要であり、そのようなソフトウェアが開発することが課題であると考える。

目次

図目次

1. はじめに

並列処理研究の応用として、気象予測、環境問題、流体計算、デジタル画像処理、遺伝子の解明、データベース処理などがある。その中でも画像処理やマルチメディアなどとともに、もっとも並列処理の活用が広がるといわれているのが、データベース処理である。データベース処理は並列性ばかりではなく、並列I/Oの点で計算機クラスタに向いた性質がある。

並列計算機には３つのメモリモデルがあり、それぞれに異なった性質がある。複数のCPUがメモリを共有し、異なるCPUが同じメモリ空間にアクセス可能な共有メモリ並列計算機、メモリは各CPUにローカルに接続され、異なるCPUのメモリに単独でのアクセスが不可能な分散メモリ並列計算機、また分散メモリの状態から異なるCPUのメモリ空間にアクセス可能にした共有分散メモリ並列計算機がある。共有メモリに即したライブラリとしては、移植性や並列性能からOpenMPが主流になっている。[5]

近年のパーソナルコンピュータの普及による劇的なマイクロプロセッサの高速化と低コスト化により、複数台のPCとネットワークを用いたシステム構築が広まっている。このシステムをPCクラスタと呼んでいる。現在、PCクラスタにはその冗長性を利用して高信頼システムの実現をねらったものと、複数のコンピュータを用いて大規模システムの実現をねらったものがある。近年、とくに後者のPCクラスタの発展が目覚しく、その勢いはとどまることを知らない。PCクラスタがほかのスーパーコンピュータと異なる点は、最低2台のPCがあればだれでも構築可能なことである。それも、PCクラスタ用のフリーソフトウェアを利用すれば、ソフトウェアのコストを掛けずに構築できるのである。しかし、安定した性能を実現するためには、様々な知識が必要であるのも事実である。

クラスタシステムの中でも注目されているのが新情報処理開発機構の中のリアルワールドコンピューティング(RWC)プロジェクトで、クラスタコンピューティングのために開発されたソフトウェアであるSCoreを利用したクラスタシステムである。SCoreは、ワークステーションやPC等で稼動しているオペレーティングシステムで、Linux上に構築したトータルソフトウェアであり、高性能通信を可能にする通信ライブラリを持っている。さらにLinuxカーネル上に構築したSCore-Dグローバルオペレーティングシステムはクラスタシステムの構成要素であるコンピュータをすべて制御し、クラスタを単一並列コンピュータのようにすることが可能である。またソフトウェア分散共有メモリシステム(SCASH)により分散メモリのクラスタ上で共有メモリのクラスタ上での共有メモリプログラミングを可能にしている。本研究室のPCクラスタはgooseクラスタとraptorクラスタがあり、SCoreのSCASHによりOpenMPが利用可能になった。[11]

本研究では、複数のPCクラスタを用いて、OpenMPによる並列プログラミングを行った。２章では、並列処理とOpenMPの背景や特性について説明を示す。３章では、マンデルブ

ロ集合の計算をPCクラスタ上でOpenMPを使い、ブロック分割、サイクリック分割で並列化し実行、その結果、考察を示す。4章では、マンデルブロ集合の近似条件を変え、同じくブロック分割、サイクリック分割で並列化し実行、それが並列化計算に変化を与えるかを考察し示す。5章では、これまでの結果を踏まえて細かく分割した格子状サイクリック分割により並列化し、3章の結果と比較し、考察する。

2. OpenMPと並列プログラミング

2.1 並列処理

2.1.1 並列処理の目的

並列処理とは、1つの仕事をいくつかの小さな仕事に分割し、それを複数のタスク（CPU）で並列に実行することである。並列処理の目的は、大きく分けて２種類ある。[10]

- 処理時間の短縮
- 信頼性の向上

があげられる。一つは上で述べたような処理時間の短縮である。複数の計算機やCPUを用いることで、全体の処理時間を短縮させることが目的である。もう一つは信頼性の向上である。複数の計算機で全く同様のプログラムを実行させ、その１つが故障や何らかのトラブルで停止した場合、ほかの計算機がトラブルのあった計算機の処理を代行することができる。処理効率は低下するが、複数の計算機に処理させることによりシステム全体のダウンを未然に防ぐ目的である。通常、並列処理は前者を指すことが多く、本実験でもこちらの目的を重視して実験を進めている。

2.1.2 並列処理の方法

あるプログラムを並列処理する場合、プログラムを分割することにより複数の計算機で実行可能になる。そのためにはそのプログラムのどの部分が並列に実行可能かを解析し、それに応じてプログラムを書き換える必要がある。

並列処理プログラミングでは、プログラムをどのように分割するかによって、そのプログラムの性能が変わってくる。並列処理に適したプログラムであっても、プログラムをうまく分割できなかった場合にほとんど並列化できないプログラムが作成され、うまく処理時間が短縮できない。そのような事態を避けるため、並列化の方法を列挙する。

- データ分割手法
- 機能分割手法

の２つに分けることができる。データ分割手法とは、プログラムで扱われるデータが膨大な量で、なおデータに対する処理が単純な場合、データを小さく分割し各計算機、CPUに

割り当てる方法である。この場合、同じような処理を複数のデータに対して独立に適用することになる。このような並列処理の手法を SIMD（Single Instruction Multiple Datastream）方式とか、 SPMD (Single Program Multiple Datastream）方式という。機能分割手法とは、プログラムで扱うデータが少量で、データに対する処理が複雑な場合、処理をいくつかの工程に分割し、それぞれの計算機やCPUに割り当てる方法である。この場合、複数の処理を複数のデータに対して適用することになる。このような並列処理の手法を MIMD (Multiple Instruction Multiple Datastream）方式という。扱う問題によってどちらの分割手法が適するかは、もちろん異なるため、問題をよく分析し、どちらの分割手法が適しているかを見極める必要がある。誤った分割手法を採用すると、並列処理したのに逆に処理時間が増加する、といった理不尽なことも起こるので注意しなければならない。また、上で説明した両方の手法を同時に採用することも可能であり、データも分割して、なおかつ処理も分割する、という手法も活用できる。最近、並列処理で扱う問題は、規模がより大きく、処理がより複雑になってきているので、データ分割と機能分割の両方の手法を組み合わせた並列処理が一般的になってきている。

2.1.3 並列処理に必要な機能

並列処理を行うためにはプログラムを分割しなければならない。どのような分割方法を採用するにしろ、プログラムを分割するには以下のような機能が必要となる。

- タスク生成機能
- タスク間通信機能
- 同期機能

タスク生成機能とは複数の処理を行うために各計算機やCPUに新たにタスクを生成したり、処理の終了したタスクを削除する機能である。タスク間通信機能とは各処理間で情報を交換するための通信機能のことである。同期機能とは各処理間で情報を円滑に行うために、各処理の待ち処理、つまりタスク間の同期処理機能のことである。これらの機能はライブラリの形式で与えられるか、コンパイラが自動的に付加している。これらは、OpenMPのライブラリに備わっているため、プログラムで適切なライブラリ関数を適切な箇所に使用する必要がある。

2.1.4 並列化からくる問題

並列処理を行い、プロセッサ数に比例した理想の速度向上が得られるとは限らない。その理由としてあげられるのが以下の要因である。

- オーバーヘッドの問題
- 負荷分散の問題

オーバーヘッドの問題とは逐次処理時や通信・同期時に起こるオーバーヘッドのことである。オーバーヘッドとは簡単に言うと余計な手間のことである。具体的には、関数の呼び出し時に、パラメータの引渡し、スタック上へのオブジェクトのアドレス設定などに時間がかかる。また、呼び出し側のスタックフレームの保存や、新しいスタックフレームのセットアップなどの手間などがある。プログラムのすべてを並列処理することは無理であるため、逐次処理が発生する。このときにでるオーバーヘッドが理想的な速度向上の妨げになる。また負荷分散の問題とは、各プロセッサが処理を終えたとしても、すべてのプロセッサが終わらない限り、処理は終了しない。その時間は他のプロセッサは待機するため、各プロセッサには付加の大小が生じているということになる。これが負荷分散の問題である。この負荷を等しくなるよう負荷均衡を図れば理想的な速度向上に近づくのである。

2.1.5 並列計算機のメモリモデル

並列処理を行う上で、プログラミングに影響を与える要因としてメモリモデルが挙げられる。主なモデルとして以下の３つをあげ、それぞれのモデルを図１、図２、図３に示す。

(1) 共有メモリモデル

複数のプロセッサがメモリバス/スイッチでメモリを共有し、異なるプロセッサが同じメモリ空間にアクセス可能であるため排他制御をする必要とする場合がある。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）と呼ぶ。この形態の利点はメモリモデルが最も汎用でプログラムが組みやすいことと、並列処理だけでなく、スループットを重視するサーバマシンとしても適しているということである。

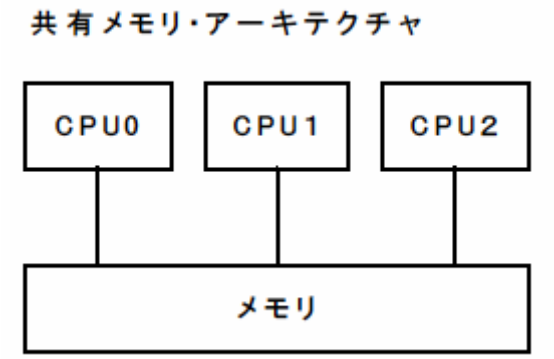


図１：共有メモリアーキテクチャのモデル

(2)　分散共有メモリモデル

プロセッサとメモリから構成されるシステムが、互いに接続された形態である。各プロセッサが全てのメモリとアクセス可能であるため、自由度が高く、大規模なシステムの構築が可能だということである。逆に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わかりにくいバグを作りこむことになるという欠点も持つ。

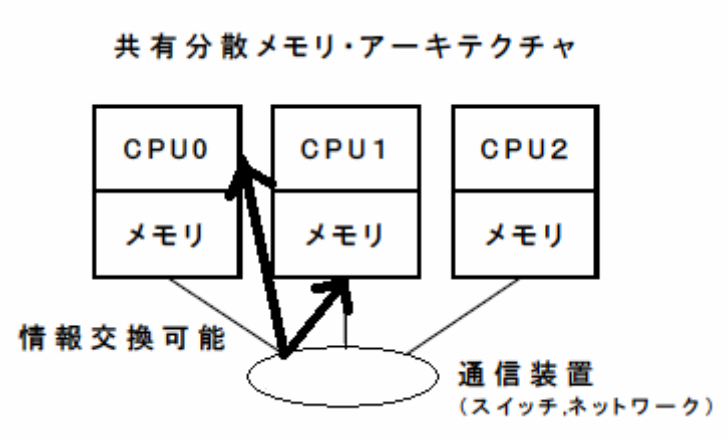


図２：共有分散メモリアーキテクチャのモデル

(3)　分散メモリモデル

メモリは各プロセッサにローカルに接続されているが、アクセスするには相手の許可が必要で、他のプロセッサが持つ情報が必要な場合、プロセッサ間で明示的な情報交換を必要とする。利点は、大規模なシステム構築が可能であることと、デッドロックさえ気をつければ、タイミングに依存する嫌なバグは発生することが少ないことである。

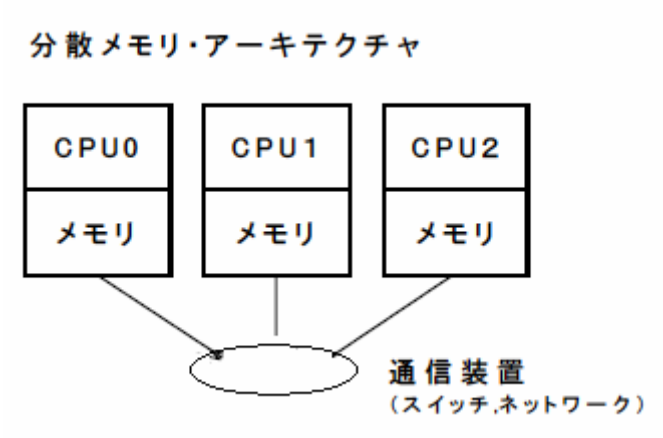


図３：分散メモリアーキテクチャのモデル

2.2　PCクラスタ

2.2.1　PCクラスタの歴史

PCクラスタとしてBeowulfが良く知られている。1990年代半ば，LinuxとSocket上のMPI(メッセージ通信インタフェース)ランタイムシステムを用いてPCクラスタを構築し広めていった。とくにLinuxを含めフリーソフトウェアだけでPCクラスタが構築可能であったため，PCのハードウェアの性能向上とともにPCクラスタは急速に広まった。Beowulfで提供されている機能は，MPIランタイムシステムであり，MPIランタイムシステムから進んで，ジョブスケジューリング機能などPCクラスタをより活用する機能を統合したものをクラスタシステムソフトウェアという。[11]

2.2.2　SCore型PCクラスタ

SCore型PCクラスタは、複数種類のネットワークをサポートした高い通信性能を実現しているだけでなく、スーパーコンピュータに匹敵する運用管理機構を備えている。SCoreは、PMv2通信機構、SCore-Dグローバルオペレーティングシステム、MPIライブラリ（MPICH-SCore）、バッチシステム（PBS/SCore）などから構成される。SCoreの構成を図４に示す。[11]

PMv2は複数種類のネットワークハードウェアをサポートし、ネットワークの種類に依存しない共通のAPIを上位のSCore-DやMPICH-SCoreに対して提供している。このため、上位のアプリケーションはネットワークの違いを意識することなく実行することができる。現在、PMv2がサポートするネットワークは、Myrinet、Ethernet、UDP/IP、共有メモリである。この中でEthernetについては複数のEthernetのカードを用いてバンド幅の向上を実現するNetwork Trunking機構が実現されている。

SCore-DはPCクラスタ上でのランタイムシステムやジョブスケジューリングの機能のほか、ギャングスケジューラによる複数ジョブでのTSS実行機能を実現している。また、信頼性を高める機構としてチェックポイント機構をも実現している。SCore-Dの提供するチェックポイント機構ではユーザプログラムへの変更は一切不要で、プログラム起動時にコマンドオプションでチェックポイントを採取する時間間隔を指定する。さらに、1台のノードPCのディスクが故障するとチェックポイントからのリスタートができなくなるため、複数のノードPCにデータを冗長に分割して格納している。

MPIはMessage-Passing Interfaceの略である。MPIは並列プログラミングの規格であり、実装はCまたはFortran 77の並列プログラミング用ライブラリである。MPIは簡単にクラスタ間でメッセージを授受できるライブラリインタフェースになっており、これによってプログラマは並列計算アルゴリズムの実装に専念することができる。

PBS（ポータブルバッチシステム）とは、ネットワーク接続された複数のPCをひとつのリソース群として、TSS（タイムシェアリングシステム）のような使い方で有効利用するために開発されたシステムである。

また、SCASHはPMv2を用い、ユーザレベルで実現したソフトウェア分散共有メモリシステムである。

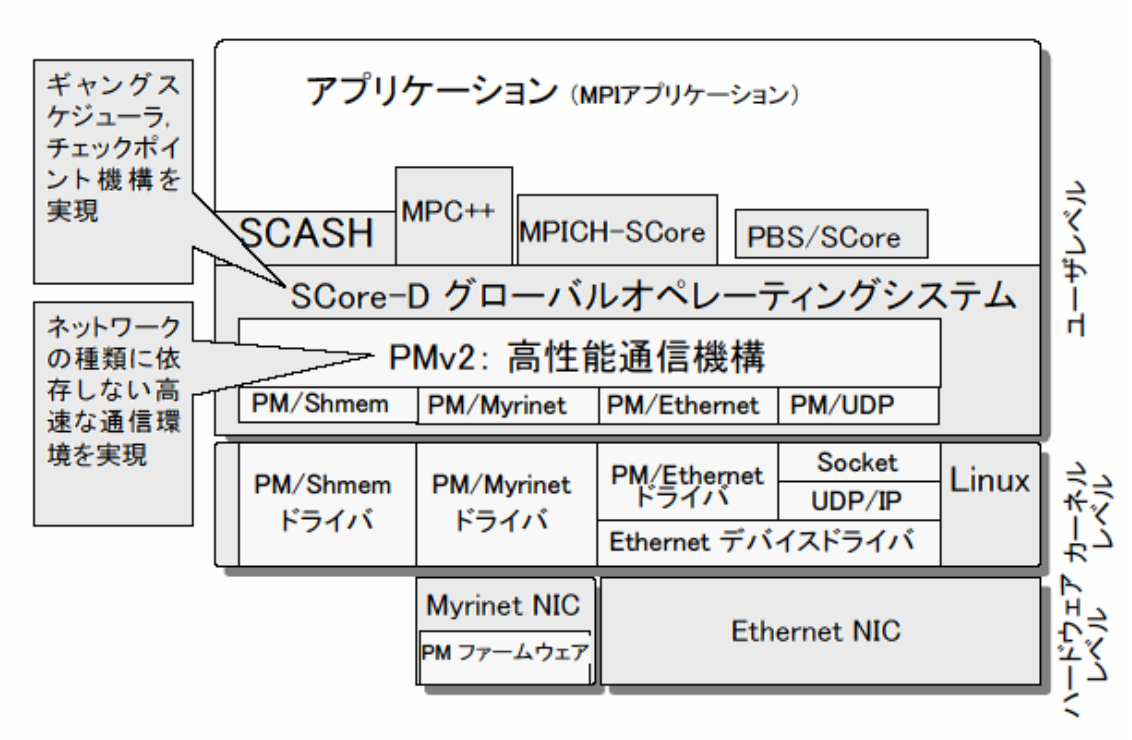


図４：SCoreの構成

2.2.3　本研究室のPCクラスタ

本研究室では、Score型のGooseとRaptorというPCクラスタを使用している。それぞれのクラスタの構成図を図５、図６に示す。

(1)　Goose

Goose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とクラスタ16台(Compute Host)を、Myrinetでクラスタ16台を接続する。

Server HostはPentiumⅢプロセッサの500MHｚ、メモリは512MBのSDRAMであり、Compute HostはPentiumⅢプロセッサの500MHｚ、メモリは512MBのSDRAMである。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4Gbps、最小遅延9μsecであり、メモリ空間を共有する。

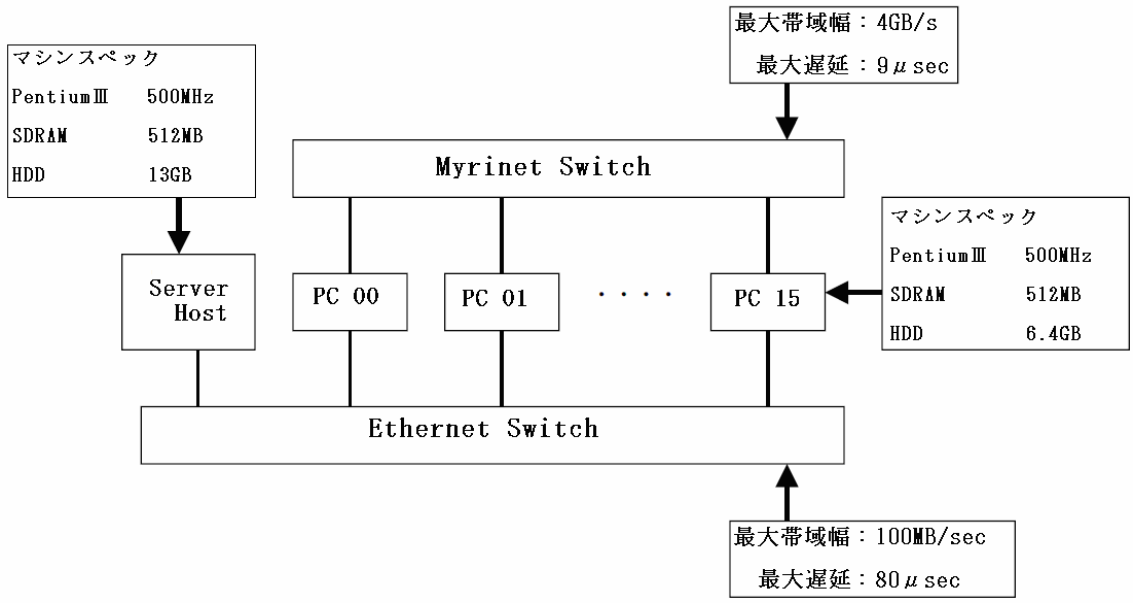


図５：Gooseの構成図

(2)　Raptor

RaptorはCompute Hostとして8台、それらを管理するServer Hostとして1台設置する。EthernetでServerとクラスタ8台(Compute Host)を、Myrinetでクラスタ8台を接続する。

Server HostはPentiumⅢプロセッサの500MHｚ、メモリは256MBのSDRAMであり、Compute HostはPentiumⅣプロセッサの3.2G Hｚ、メモリは2GBのSDRAMである。Ethernetは最大帯域幅1Gbps、最小遅延60μsecであり、クラスタ同士を接続する。

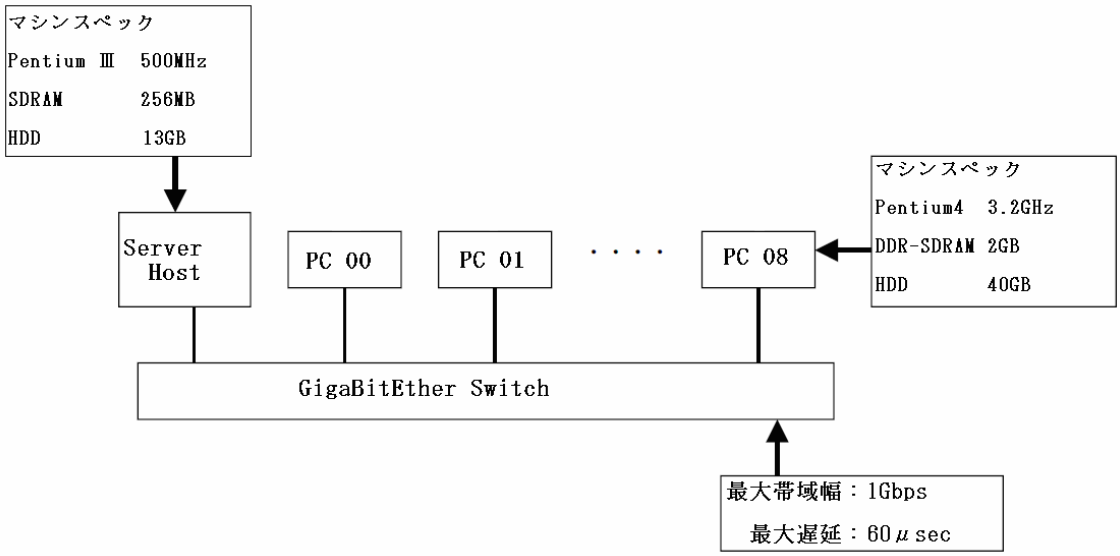


図６：Raptorの構成図

2.3 OpenMP

OpenMPが誕生するまでは、共有メモリ型並列計算機には問題があった。それは、共有メモリ型並列計算機のメーカーや機種に依存しない共通のプログラミングモデルもツールも無く、それぞれの計算機ごとに個別のプログラムを開発する必要があった。このため、あるメーカーの並列計算機のための並列プログラムが用意できても、それは他のメーカーの計算機では利用不可能だった。そこで、主に計算機メーカーに所属する技術者・研究者が共同で開発にしたのが、共有メモリモデルによる並列プログラムの標準規格OpenMPである。その間に、計算機メーカーからも研究機関からも独立した団体であるOpenMP Architecture Review Board (ARB) が設立され、OpenMP規格は、OpenMP ARBによって管理されるようになった。その結果、現在では、ほとんどの共有メモリ型並列計算機上でOpenMPに準拠したプログラムを利用できる環境が整った。また、2001年6月にはさまざまな計算機の性能を比較するための事実上の国際標準であるSPECベンチマークテストにもOpenMPが追加され、いくつかの共有メモリ型並列計算機の性能測定結果が登録されるようになった。今後は、移植性の高い並列プログラムの開発手段として、OpenMPの重要性はさらに高まると思われる。OpenMPは、決して新しいプログラミング言語ではなく、既存のFortranやC(C++)の文法の中にある、コメント文や pragma文を利用して、コンパイラに対し、「プログラムのここからここまでを並列化せよ」と「指示」する、という一種の並列プログラミングモデルである。OpenMPがある計算処理を行うための逐次型プログラムをすでに持っている場合、OpenMPによる並列化には、以下のような利点がある。移行の容易さ最初からプログラムを書き直す必要のあるメーカー独自の並列言語を用いる場合

に比べて、移行の負担は格段に軽減される。また、実行文の追加やループの変形など、かなりのプログラム改造を要するMPIやPVMによる並列プログラミングと比べても、移行の最初の段階の手間が極めて少なくて済む。また、移植性の高さメーカー独自の言語と異なり、OpenMP規格に準拠している並列計算機・コンパイラならば、どこに持って行っても実行が可能である。そして、単一プロセッサ環境との共存の容易さOpenMPでは、一部のライブラリサブルーチン・関数を除き、実行文の文法には逐次型との違いがない。したがって、並列化を行ったプログラムでも、多くの場合、単一プロセッサ環境で引き続き利用することができる。最後に特定のメーカーに依存しないことメーカー独自の並列言語を用いる場合と異なり、OpenMPの規格は、特定のメーカーに依存しない国際組織（OpenMP ARB）によって管理され、そこに参加する多くのメーカーが、この規格の制定・改良に関与している。また、それぞれのメーカーは、自社の計算機に対し、この規格に準拠し高性能かつ信頼性の高いコンパイラを供給される。したがって、使用している計算機のメーカーが万一並列計算機市場から撤退するようなことが起こっても、その計算機と「無理心中」させられる可能性は極めて低いことが言える。[12]

3.　マンデルブロ集合の並列化

3.1　問題定義

マンデルブロ集合とは、複素関数 $f(Z): Z_{n+1} = Z_n^2 + C$ 、$Z_0 = 0$ で表される数列 Zn が n が無限大まで大きくなったときに発散しないような複素数 C の集合である。マンデルブロ集合を図 7 に示す。この複素平面上の集合の点の数を X×Y のメッシュに区切り、複素関数 $f(Z)$ を反復計算して計測するプログラムを設計する。[13]

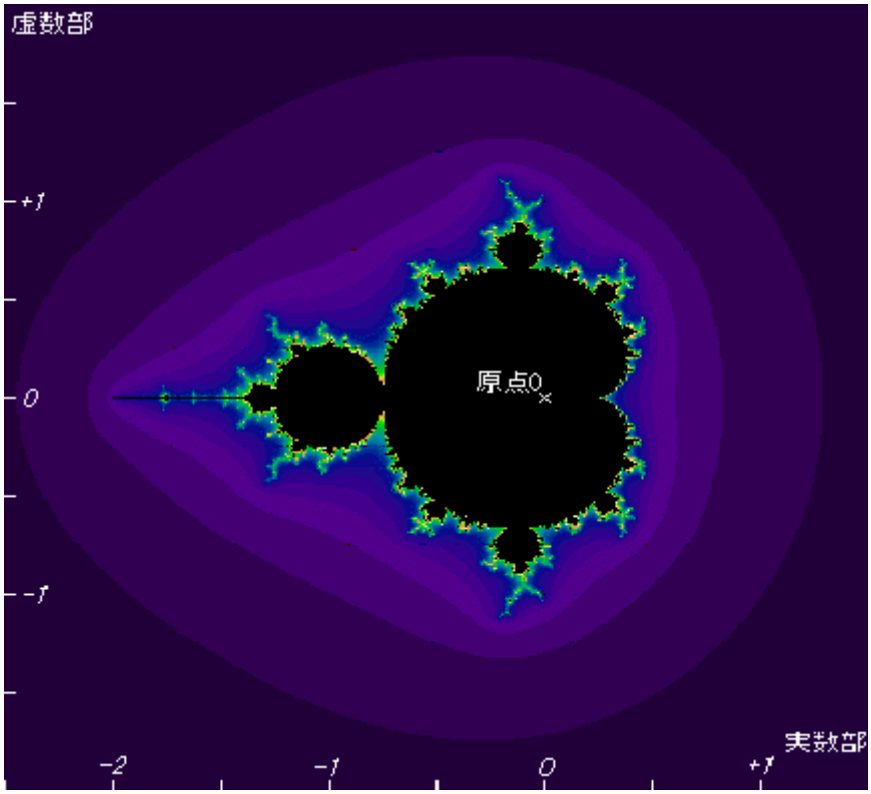


図 7 ：マンデルブロ集合

このとき、数値計算を無限大で扱うことはできない。しかし、条件を付け加えることによって近似することで、計測することができる。ここである k に対して |Zk|>2 ならば Zn →∞となることが証明されているので、これを判定条件にする。これは各 C につき、有限の n に対して 0≤ k ≤n の範囲で順次計算し、ある k において |Zk| ≤2 となった場合 C に属すると判定する。また |Zk|>2 ならば発散するので C に属さないと判定する。また、有限の n で計算を打ち切るため、n>50 のとき収束するとして設定する。この n はこの集合の

点の数を決定するもので n の限界値が大きくなるほど得られるマンデルブロ集合は近似的なものになる。

この条件で得られた図形の計算を複素平面上(-2.0≤実部<0.5)、(-2.2≤虚部<1.5)の範囲で行う。並列化の手法は、複素平面上の X 座標の範囲を以下の方法でブロック分割とサイクリック分割に分けて実行する。

Z を x 、y で表すと、

$$Z = x + yi$$

同じく C を A、B で表すと、

$$C = A + Bi$$

これを複素関数に代入し、複素関数を実部、虚部に分けて表す。

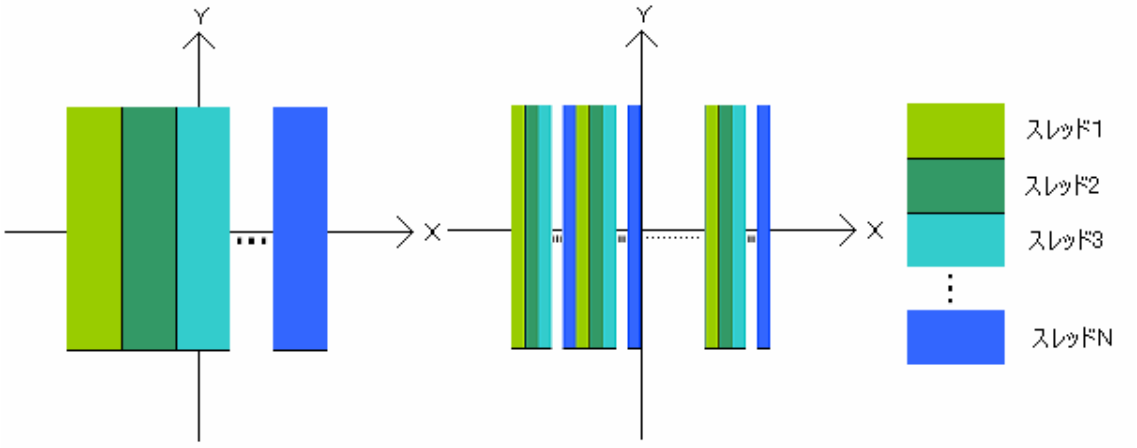
$$Z_x = x^2 - y^2 + A$$

$$Z_y = 2xy + B$$

実部と虚部の始点と終点との距離を計算し、それを定めた解像度によって分割する。
そしてそのメッシュ部分の x 、 y の値をループで上の式に代入する。
上の式の解のうち、|Zn|≤2の条件を満たしたものだけをカウントする。

3.2　並列化

PC クラスタ goose で、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並列化し実行する。
ブロック分割ではスレッド数と同じだけ分割して処理するのに対し、サイクリック分割では1/2500に分割し、それを各スレッドに1つずつ順番に分配して処理する方法をとっている。
ブロック分割とサイクリック分割のモデルを図８に示す。



ブロック分割　　サイクリック分割

図８：ブロック、サイクリックの分割方法

3.3　実行結果

スレッド数と実行時の解像度（2500×3700）を設定し、ブロック分割とサイクリック分割のそれぞれの速度向上の比較を図９に示す。スレッド数16でブロック分割では8.3倍、サイクリック分割では15.8倍の速度向上が得られた。

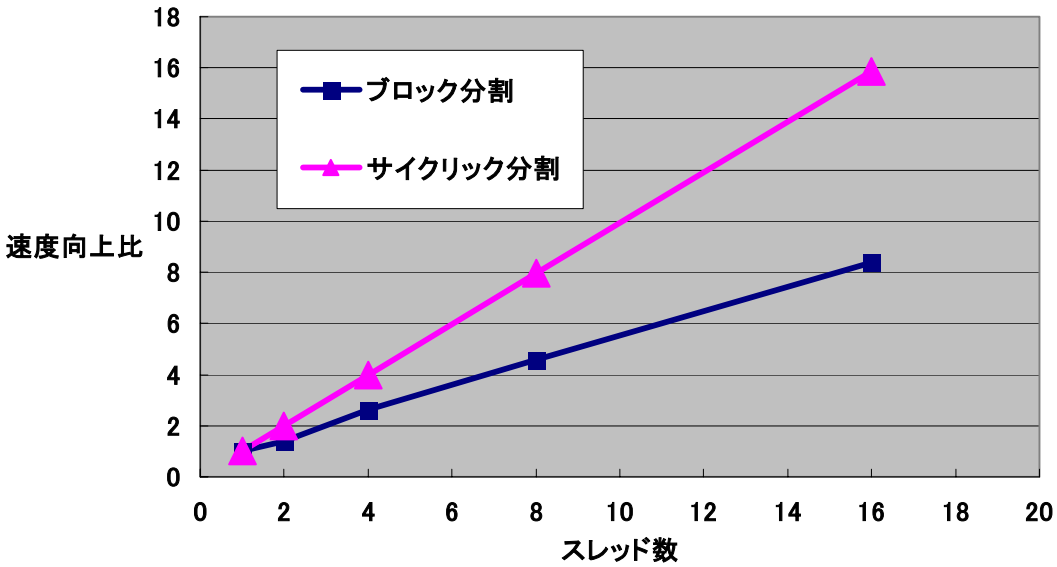


図９：速度向上の比較

3.4　考察

手動分割することによりブロック分割とサイクリック分割のスレッド数が 1 のときの実行時間に差が見られなくなり、比較が容易にできた。ブロック分割とサイクリック分割とでは、スレッド数が増えるにつれ、サイクリック分割のほうが処理時間が短くなり、速度向上も得られた。これはブロック分割よりもサイクリック分割のほうが負荷均衡が取れているということである。

4.　様々な条件での並列化

4.1　問題定義

この章は前章で実験したマンデルブロ集合の計算の条件を変え、PCクラスタraptorで並列化を行い、どのような変化があるのかを調べる。

マンデルブロ集合の複素関数 $f(Z): Z_{n+1} = Z_n^2 + C$ 、$Z_0 = 0$ であらわされる数列を

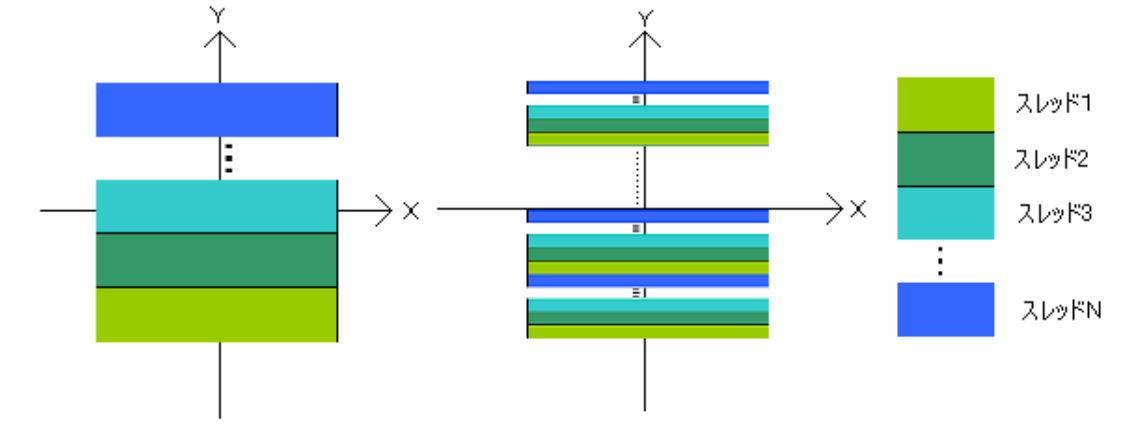
(-2.0≤実部<0.5)、(-2.2≤虚部<1.5)の範囲で複素平面をX×Yのメッシュに区切り、X×Y 個の点について上の複素関数の反復計算を行う。|Zn|>2 で発散し、n>50 のとき収束するとして、収束時の座標点がマンデルブロ集合の要素となる。並列化の手法は、複素平面上のX座標の範囲をブロック分割とサイクリック分割に分けて実行し、マンデルブロ集合に属する座標点の合計数を count に格納する。

今回は以上の条件を以下のように変更する。

(1)　解像度（2500×3700→25000×37000、250×370）
(2)　分割方法（範囲内の実数による分割を虚数による分割へ）
(3)　実部、虚部の範囲（-2.0≤実部<0.5、-2.2≤虚部<1.5→-5.0≤実部<5.5、-5.2≤虚部<5.5）
(4)　発散条件（|Zn|>2→|Zn|>3）
(5)　収束条件（n>50→n>100）

4.2　並列化

今回の並列化は(2)以外は前回と同じ方法で実行できるが、(2)は前回の方法とは異なる。(2)以外は複素平面上のｘ座標の範囲をブロック分割とサイクリック分割で並列化し実行するが、(2)はｙ座標の範囲をそれぞれの方法で分割し並列化する。
ブロック分割はスレッド数と同じだけ分割して処理するのに対し、サイクリック分割は実数（虚数）の距離の解像度によって決定した数だけ分割しそれを各スレッドに1つずつ順番に分配して処理する方法をとっている。スレッドがN個のときにブロック分割とサイクリック分割をｙ軸方向で行った方法を図１０に示す。



ブロック分割　　サイクリック分割

図１０：y 軸方向での分割方法

4.3　実行結果

(1)~(5)のブロック分割とサイクリック分割のそれぞれの速度向上比と前回の実験結果の比較を行った。

(1)　解像度変更

スレッド数と実行時の解像度（2500×3700）、解像度（25000×37000）、（250×370）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１１に示す。スレッド数 8 でブロック分割は 4 倍、2.4 倍、サイクリック分割では 7.3 倍、4.6 倍の速度向上が得られた。この結果では前回の結果と比較すると、解像度（25000×37000）はブロック分割、サイクリック分割と同様にあまり差異は見られなかったが、解像度（250×370）はスレッド数４、８でブロック分割、サイクリック分割両方に差異が見られた。この理由として、解像度（250×370）の情報量が少なかったことによるオーバーヘッドからくる速度の低下だと考えられる。

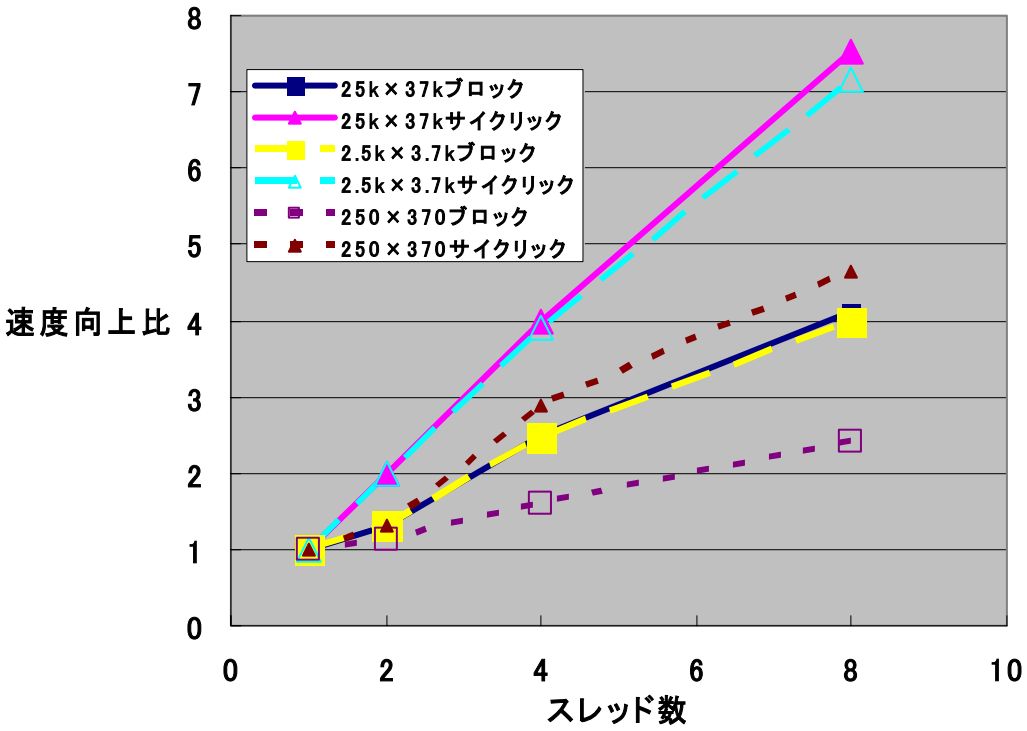


図１１：解像度変更による速度向上の比較

(2)　分割方法の変更

この実験では図２で示すとおり、虚数方向からブロック分割、サイクリック分割したものである。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１２に示す。スレッド数 8 でサイクリック分割では 7.1 倍、ブロック分割では 2.6 倍の速度向上が得られた。この結果は前回の結果と比較するとサイクリック分割に差異は見られなかったが、スレッド数４、８ではブロック分割に差異が見られた。ブロック分割とサイクリック分割の結果に違いが見られるのは、各プロセッサに仕事を与えたときの情報量の大小によって、各プロセッサが処理を終える時間に違いが出るためである。差が出たのはマンデルブロ集合の形が横長だからだと考えられる。

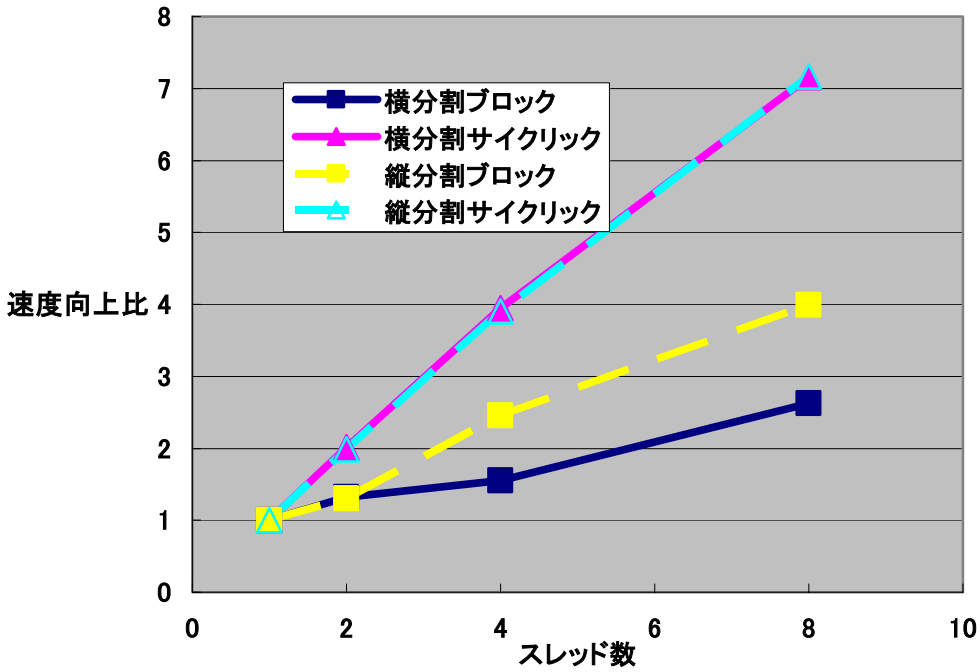


図１２：分割方法による速度向上の比較

(3)　実部、虚部の範囲の変更

ここでは、前回の実験よりも実部と虚部の範囲を拡大して比較した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１３に示す。スレッド数８でサイクリック分割では 7.0 倍、ブロック分割では 2.3 倍の速度向上が得られた。この結果では前回の結果と比較するとサイクリック分割に差異は見られなかったが、スレッド数４、８では②の結果と同様にブロック分割に差異が見られた。理由は広範囲になったため、$-2.0<r<0.5$、$-1.0<i<1.0$ に分布するマンデルブロ集合から考えて、処理が多くなるプロセッサと少なくなるプロセッサに分かれたからと考えられる。

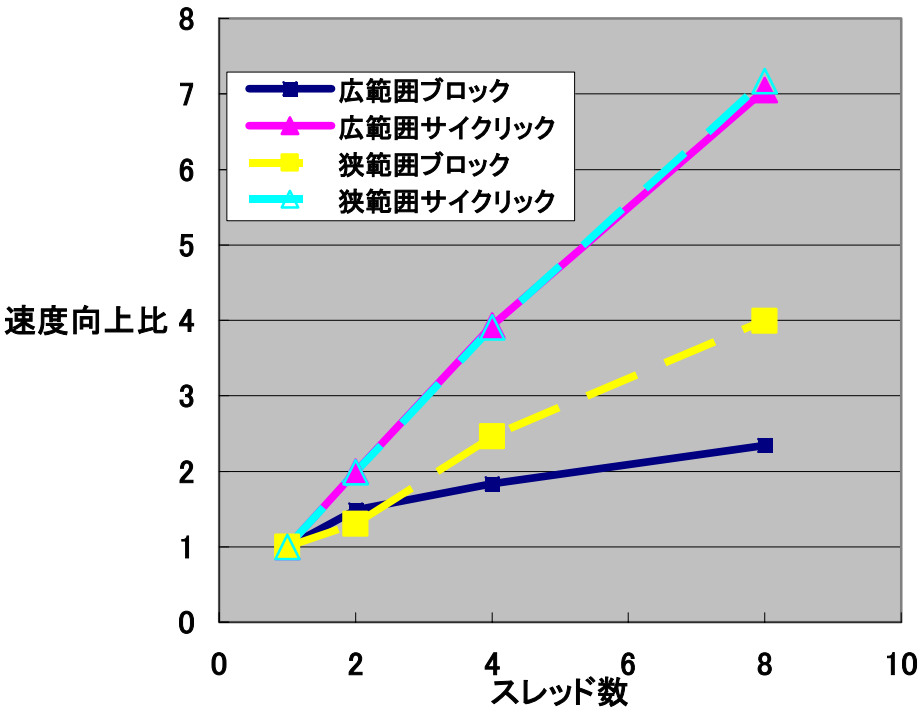


図１３：範囲変更による速度向上の比較

(4)　発散条件の変更

ここでは、複素関数 $f(Z)$ の発散条件を|Zn|>2 から|Zn|>3 に変更して実行した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１４に示す。スレッド数 8 でブロック分割は 4.1 倍、サイクリック分割では 7.2 倍の速度向上が得られた。この結果は前回の結果と比較しても差異は見られなかった。

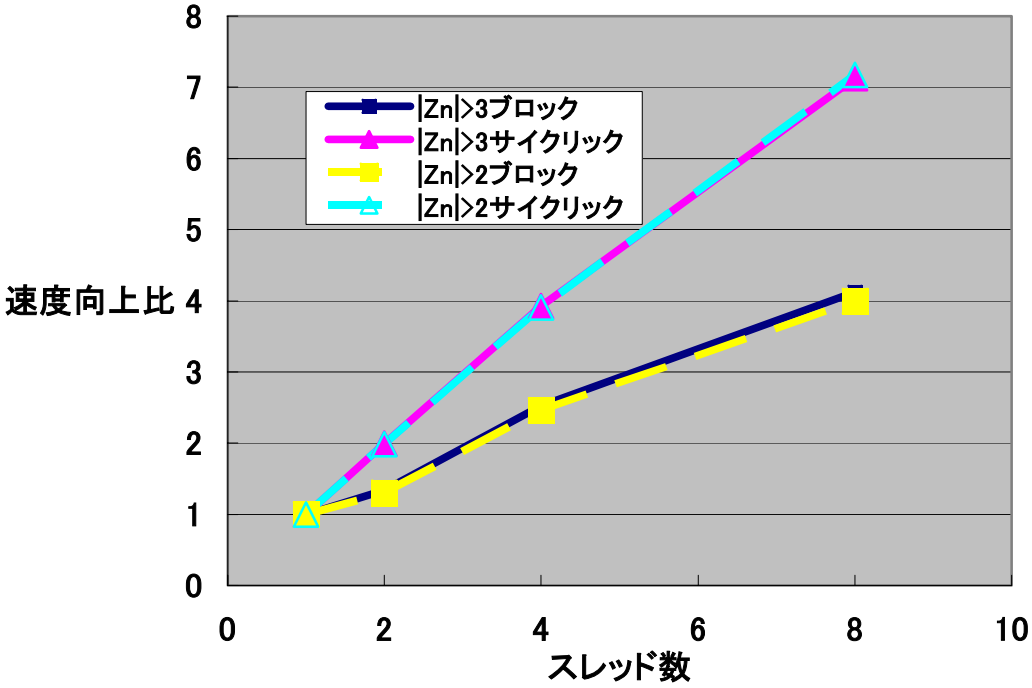


図１４：発散条件変更による速度向上の比較

(5)　収束条件の変更

ここでは、複素関数 の発散条件を n>50 から n>100 に変更して実行した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１５に示す。スレッド数 8 でブロック分割は 4.1 倍、サイクリック分割では 7.2 倍の速度向上が得られた。この結果は前回の結果と比較しても差異は見られなかった。

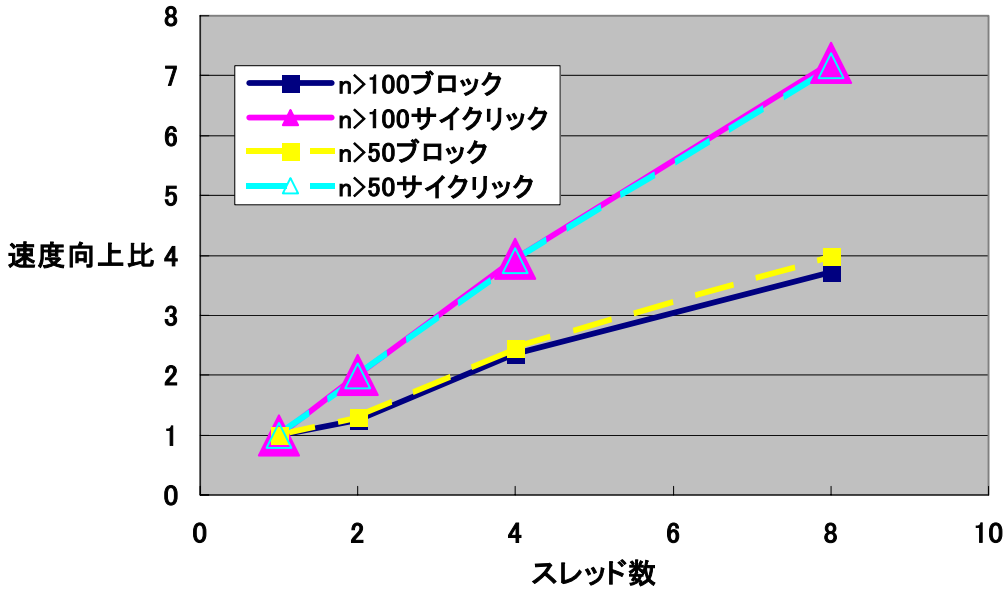


図１３：収束条件変更による速度向上の比較

4.4　考察

(4)、(5)の実験では条件の変更による速度向上の変化はほぼ見られなかったのに対し、(1)はブロック分割、サイクリック分割両方に、また(2)、(3)ではブロック分割にだけ差異が見られた。(1)は計算量が少ないのでプロセッサ1台で十分処理できる計算を無駄に分割し、並列化してことで計算量を増やし、オーバーヘッドが発生したと考えられる。サイクリック分割が(2)、(3)の場合も変化がないのは分割を多くして各プロセッサに割り振るため、情報量の差が少なくなり負荷が小さくなるためである。よって、サイクリック分割はブロック分割よりも理想的な速度向上が見られるのである。

5.　格子状サイクリック分割での並列化

5.1　問題定義

これまでの実験結果から並列化の負荷分散を少なくするためには、より細かく分割する方法が良いと考えられる。そのために３章で行った実験と同条件のマンデルブロ集合を実軸と虚軸の両方面から細かく分割し、各スレッドに割り当てる方法で実行した。以下の問題をブロック分割、サイクリック分割、格子状サイクリック分割で並列化し、結果を比較、考察した。

5.2　並列化

ブロック分割、サイクリック分割の並列化は実部の範囲を1/2500に分割し並列化し実行した。格子状サイクリック分割の並列化は実部の範囲を1/2500に、虚部の範囲を1/3700に分割し、2500×3700個の格子にして並列化を実行した。図１６に格子状サイクリック分割のモデルを示す。

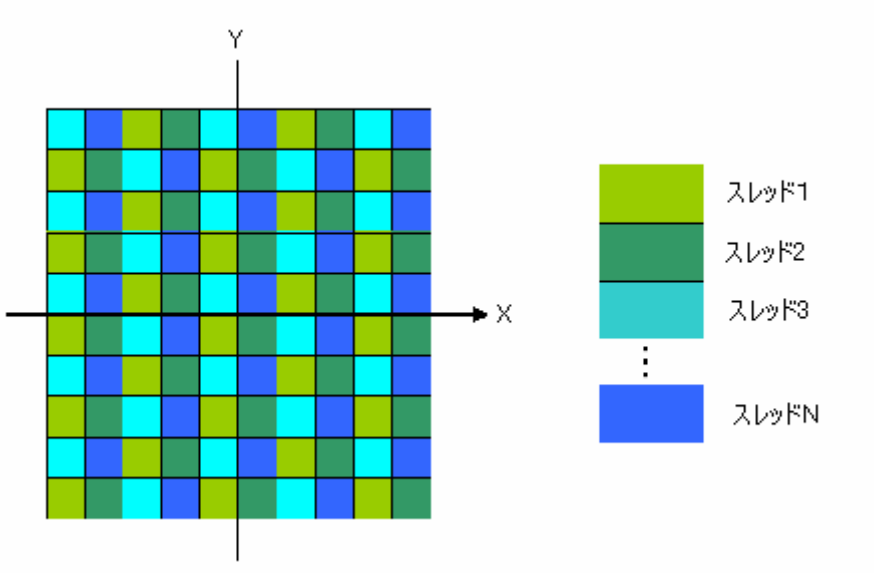


図１６：格子状サイクリック分割のモデル

5.3　実行結果

スレッド数と実行時の解像度（2500×3700）を設定し、ブロック分割とサイクリック分割、格子状サイクリック分割でのそれぞれの速度向上の比較を図１７に示す。スレッド数16でブロック分割では8.3倍、サイクリック分割では15.9倍、格子状サイクリック分割では15.8倍の速度向上が得られた。

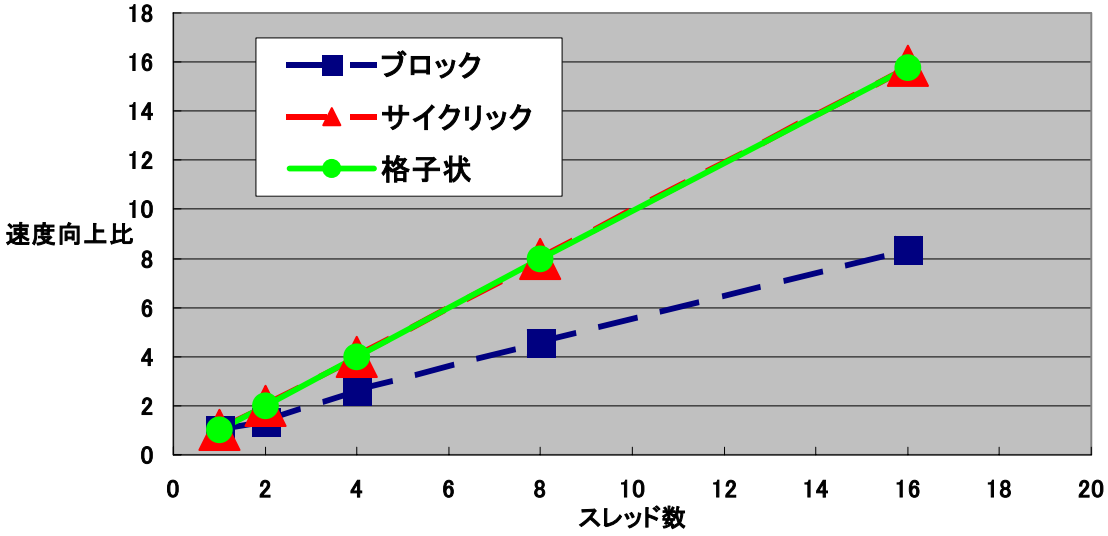


図１７：格子状サイクリック分割による速度向上の比較

5.4　考察

理想では格子状サイクリック分割がサイクリック分割を上回る速度向上を得るとの予想だったが、そうならなかった理由としてオーバーヘッドがある。マンデルブロ集合の図形ではサイクリックから格子状サイクリックへの移行はあまり意味を成さず、多く分割した無駄手間によってオーバーヘッドが生み出され、理想の速度向上が得られなかったと考えられる。各プロセッサの短縮された計算時間より、細かく分割するための計算時間が多くなったことで起きてしまったことである。この実験では処理するデータによって最適な分割方法、スケジューリングがあり、データに合った並列プログラムを考えることが速度向上に必要である。

4. おわりに

本研究では、SCore型PCクラスタでのOpenMPによる並列プログラミングを行ってきた。本研究で行ってきたマンデルブロ集合の様々な角度からの並列化計算を実行することにより、SCoreがPCクラスタの仮想共有メモリシステムを構築し、OpenMPを利用可能にしたことや、OpenMPが幅広いプログラミング言語の種類を網羅していることから、SCoreの仕組みやOpenMPの利用法、移植性の高さを学ぶことができた。また、様々な条件から計算したことで並列化による負荷やオーバーヘッドも実験内容から知ることができた。

今現在、OpenMPという移植性の高い言語ライブラリができたことで容易に並列化が可能になったが、先ほどの章で述べたデータ毎の最適なプログラミングについてはプログラマが工夫しなくてはならない。これからの課題として、プログラムが最適な速度向上が得られるOpenMPに変わる言語ライブラリの開発が必要だと思っている。これから宇宙や遺伝子などの研究が進み、天文学的な数値の計算を扱ったりする際にこのようなライブラリを作ることが、それらの研究の発展に繋がるのではないかと考える。

謝辞

本研究の機会を与えて下さり、数々の助言を頂きました山崎勝弘教授に心より感謝致します。また、本研究にあたり、励ましの言葉や貴重な意見を頂きました本研究室のマスターの林氏、池上氏に心より感謝致します。

参考文献

[1] 三田典玄：改訂新版 入門C言語、アスキー、1990 ．

[2] 三田典玄：改訂新版 応用C言語、アスキー、1990 ．

[3] 合原一幸：カオスーカオス理論の基礎と応用、サイエンス社、1990.

[4]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 ．

[5]黒川耕平：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 ．

[6]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 ．

[7]古川智之、松田浩一、安藤彰一：並列プログラム事例集、立命館大学理工学部情報学科高性能計算研究室、1996 ．

[8]米田健治、徳山美香、青地剛宇：並列プログラム事例集2、立命館大学理工学部情報学科高性能計算研究室、1999.

[9]柿下裕彰：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、2000.

[10]湯浅太一、安村通晃、中田登志之：はじめての並列プログラミング、共立出版、1999.

[11]石川裕、住元真司：Linuxで並列処理をしようーSCoreを用いたクラスタ構築ー、共立出版、2002.

[12]Omni OpenMP Compiler：http://www.hpcc.jp/Omni/home.ja.html.

[13]Ｔ’ｓホーム～フラクタル・プログラム：http://www.mni.ne.jp/~t47889/index.html.