卒業論文

ＭＰＩとＯｐｅｎＭＰを用いたＮクイーン問題の並列化

氏名：杉田 卓司
学籍番号：22600500299
指導教員：山崎 勝弘 教授
提出日：

内容梗概

目次

図目次

表目次

１．　はじめに

並列計算は大量のデータを用いて多くの計算を行う処理を高速化するための手法の一つである。処理時間を削減したり、リアルタイムでの処理をするために複数のプロセッサで１つのタスクを動作させることを並列処理という。並列処理の研究の応用として、気象予測や、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。

並列計算機には３つのモデルがあり、それぞれが異なった特徴を持っている。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（SMP)、メモリが各 CPU に接続され、別の 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、そして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することを可能にした共有分散メモリ並列計算機がある。

PC クラスタを用いた並列処理を行うと、大規模な計算の処理時間を削減することができる。PC クラスタとは複数台の PC を組み合わせ、一つのシステムにしたものである。並列処理を行うには、PC クラスタ専用のソフトウェアを用いて、複数台の PC をネットワークで接続した PC を一つのシステムのように扱えるようにすることと、PC クラスタに計算させるための並列プログラムが必要である。PC クラスタは、同じ性能をもつスーパーコンピュータに比べ、非常に安価でシステムを構築することができる。プロセッサ数に比例して実行速度が向上するという長所を持ち、それに伴って単独のプロセッサではメモリ不足となるような大規模な問題を解くことが可能になる。クラスタシステムの中で最も注目されているのが、リアルワールドコンピューティング（RWC）プロジェクトで開発された、クラスタ用システムソフトウェアである SCore を利用した Score クラスタシステムである。SCore は、Linux 上に構築したトータルソフトウェアであり、高性能通信機能を実現する通信ライブラリを持つ。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ションシステムは、クラスタの構成要素であるコンピュータを全て制御し、クラスタを単一の並列コンピュータのようにすることが可能である。さらに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、共有メモリプログラミングができる。これによって SCore クラスタシステム上で、共有メモリ型並列言語である OpenMP のプログラムが可能になった。

本研究では N クイーン問題を MPI、OpenMP の 2 種類を用いてプログラミングし、条件の異なる３種類のマシンで実行した。MPI は標準的なメッセージパッシングライブラリであり、現在ほとんどの分散メモリ型計算機に実装されている。分散メモリ型計算機のそれぞれのプロセッサ上でプロセスを走らせ、プロセス間でメッセージ交換を行うことで協調して処理を進めるという並列化方式である。OpenMP は共有メモリ型計算機の標準規格として用いられているプログラミングモデルである。本研究では SPMD プログラムでＮクイーン問題をプログラミングした。マシンは、SR8000、Raptor、Nycto の３つを用いて実験し、比較する。N クイーン問題は、ルールは非常に簡単なパズルであるが、規模が大きく

なるにつれて、莫大な計算量が必要になるという特徴がある。更に、法則性が少なく、総当りに近い解き方をせざるを得ない。この特徴はプログラムの特徴、マシンによる並列化性能の違いを比較する実験に適している問題といえる。

第２章では N クイーン問題の定義とアルゴリズム、並列プログラミングをする際に理解しておかなければならない事柄を､特に PC クラスタと MPI と OpenMP の２種類のプログラミング方法について簡単に説明し、第３章、第４章ではそれぞれ、MPI、OpenMP を用いて作成した並列プログラムに関する並列化手法、実験結果、考察を記述する。第５章でこれまでの実験結果、考察を踏まえて MPI とＯｐｅｎＭＰの言語としての比較等の報告をする。全体のまとめは第６章に記述する。

２．　Ｎクイーン問題と並列処理

２．１　　Ｎクイーン問題

Ｎクイーン問題とは、N×N のチェス盤上に N 個のクイーンを置くパズルである。チェスのクイーンは縦、横、斜めに何マスでも進むことができる。ただし、どのクイーンもその移動可能な場所に他のクイーンが置かれていないことが条件である。この局面がいくつあるかを数えるものである。1848 年にチェスプレイヤーのマックス・ベッツェルによって提案された 8 クイーンというパズルを発展させたもので多くの数学者がこの問題に挑戦したが、どうしても総当り的な解き方をせざるを得ない問題であるため、規模が大きくなるに従って計算量が飛躍的に増加するため、並列プログラミングを比較することに適した問題といえる。以下にＮクイーン問題のＮが５から１６についての解の総数の表を示す。

ここからはＮクイーン問題を高速に解く一つの手法を紹介する。
全パターンを試し、得たある１つの解が、回転、反転などで他の解と同じ型になるものが存在する場合、本質的には同じ解であるため、一つの解として数えるとする解の数え方で得られる解を「ユニーク解」という。全ての解の中から、回転、反転などで同じ解になるものをこの数え方でグループ化することができる。１つのユニーク解から回転、反転をすることで、８通りの解を得ることができるため、一見ユニーク解×８が解の総数になると考えがちだが、回転によってユニーク解のままになるものがあるため、そのことを判定する必要があることを、以下の図１のＮが５の場合において説明する。

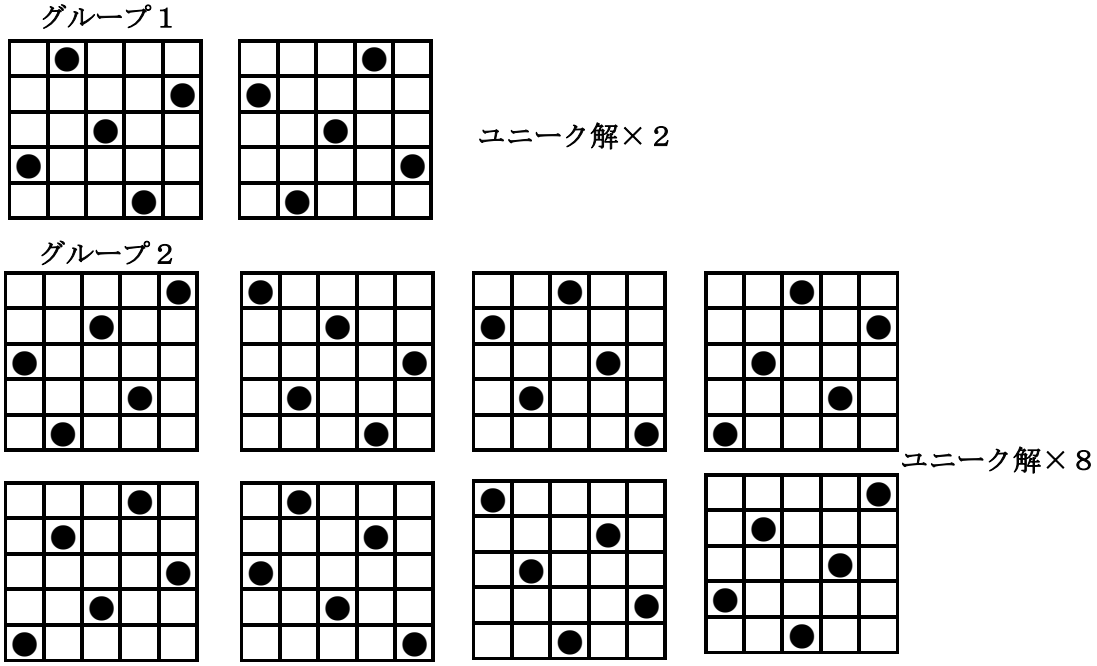


表１：Ｎが５の場合の解のグループ分け

ユニーク解を求めることで、そのユニーク解が何パターンの解になるかを考えることで、解の総数を求める計算量を減らすことができる。

・角にクイーンがある場合

回転によって４つの解になり、写軸で反転することができるため、その２倍の解ができる。このことから右上の角にクイーンがある場合８個の解となる。

・角以外にクイーンがある場合

90 度回転させてユニーク解と同じものになる場合は、180 度、270 度回転させても同じ解となるため、反転させることしかできず、２個の解ができる。

90 度回転させてユニーク解と異なる場合（必ず 270 度回転させても異なる解となる）

→180 度回転させてユニーク解と異なる場合、反転させることができるため８個

→180 度回転させてユニーク解と同じものになる場合、反転させて４個

２．２ アルゴリズム

Ｎ×Ｎのチェス盤上のＮ個のクイーンを図１のように表す。図１は５×５のチェス盤上に５つのクイーンを置いたものである。クイーンは「●」で表している。

・ この配置は図１のような１次元配列で表現することができる。

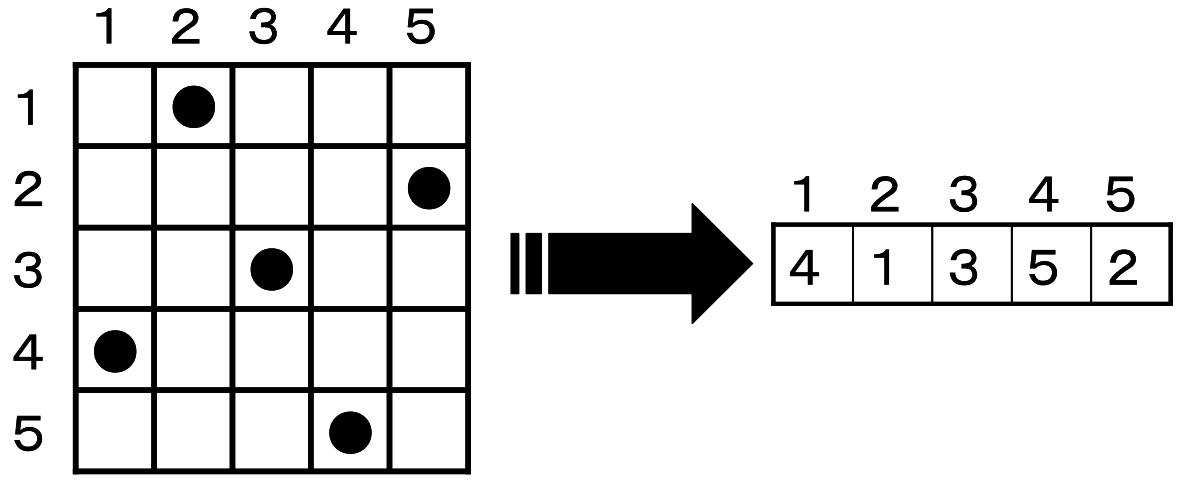
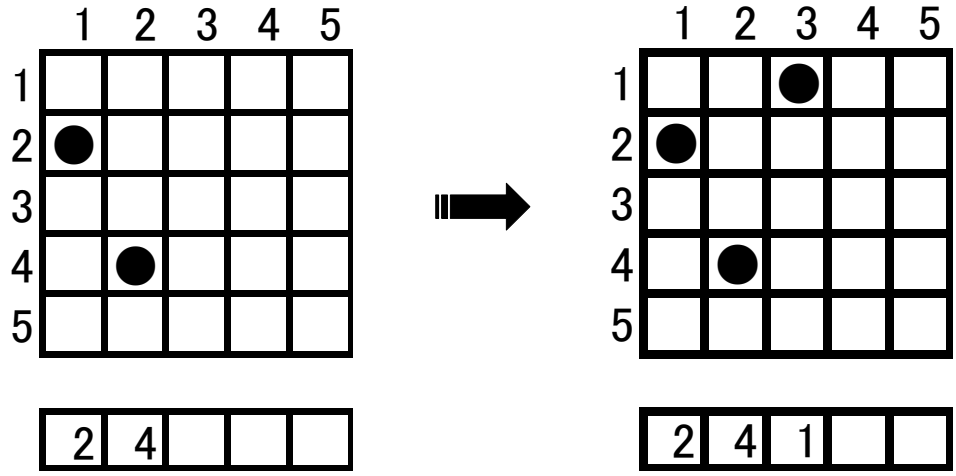


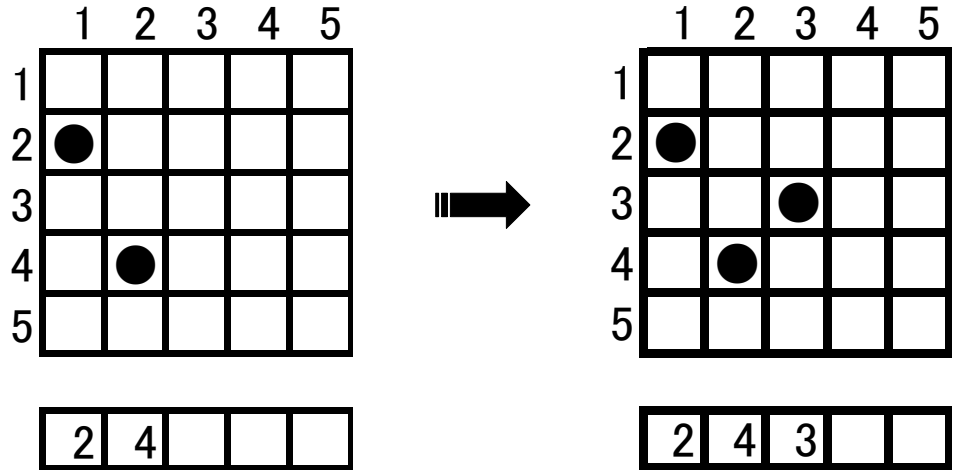
図 1：クイーン配置のデータ構造

配列の要素数はＮで、各要素番号が列番号を表し、要素の値が行の値を表している。各配列要素に１からＮまでの値を入れることでクイーンの配置が決まる。解となる配置の配列内の要素は必ず縦方向と横方向に重複しないため、斜め方向に重複しないことを確認することで解を求めることができる。Ｎ個の順列を満たす数字列が並ぶ度にカウントアップすることで解の総数を求めることができる。

１からＮまでの順列に対して斜め方向に重複していないか確認を行う際に、順列の生成要素を追加する度に斜め方向の確認を行うことで、その先の計算をする必要がなくなり、処理量の削減をすることができる。この作業を行う様子を図２に示す。



斜め方向にクイーンがないため計算続行



斜め方向にクイーンがあるため計算中止

図 2：計算量の削減アルゴリズム

２．３　　並列処理

２．３．１　　ＰＣクラスタ

PC クラスタとは複数台の PC を組み合わせ、一つのシステムにしたものである。並列処理を行うには、複数台の PC をネットワークで接続した PC を専用のソフトウェアを用いて一つのシステムとして扱うことができる。ＰＣクラスタは２～８台のようなごく小規模なものから数千台規模で性能を発揮するものまで、予算や要求によって構成を変えることができるため、特にコストパフォーマンスに優れる。1995 年頃からイーサネットスイッチや

Ｇｉｇａｂｉｔ　Ｅｔｈｅｒなどの技術が普及し、より安価に高性能なネットワークの構築が可能になった。近年では複数のプロセッサを搭載したＳＭＰ型のＷＳやＰＣも容易に入手できるようになり、これらをベースとしたＳＭＰクラスタが主流になっている。高い処理速度をもっているため、計算量が多いものや、リアルタイムに処理しなければならない問題に用いられている。計算量が多いものでは気象予測やＤＮＡ解析、タンパク質構造予測等があり、リアルタイム性を要求されるものではＭＰＥＧ２円コーダやH．２６４などが挙げられる。

２．３．２　並列プログラミング

（1）ＭＰＩ

ＭＰＩはMessage Passing Interfaceの略である。分散メモリ型計算機においてメッセージパッシングを用いて並列プログラミングを行うためのライブラリで、プログラミングとアプリケーションのインターフェースとなるものである。1990年以前はそれぞれのマシンが個々のライブラリを持っていたが、現在はＰＶＭを経てＭＰＩが標準的なメッセージパッシングライブラリとなり、ほとんどの分散メモリ型計算機に実装されている。

ＰＣクラスタを含む分散メモリ型計算機のそれぞれのプロセッサ上でプロセスを走らせ、プロセス間でメッセージ交換を行うことで協調して処理を進めるという並列化方式が一般的である。各プロセスはUNIXの通常のプロセスとほとんど同じである。ＭＰＩではそれぞれのプロセスが何番目のプロセスとして動いているかということを基準として適宜処理を変更しながら全体で矛盾のないようにプログラミングする。ＭＰＩのプロセス間のメッセージ交換では、送信されたメッセージは必ず受信されなければならず、プログラミングする際にはこの点にも留意する必要がある。

（2）ＯｐｅｎＭＰ

ＯｐｅｎＭＰには以下に挙げるメリット、デメリットがある。

メリット

- シームレスに並列化することができる
- 逐次版と並列版を同じソースに管理できる
- ユーザ指示通りに並列化できる
- スレッドプログラミングに比べて、並列性が構造的に記述できる

デメリット

- 並列可能性はユーザが確認する必要がある
- Date Mappingが記述できない
- 配列に対してreduction演算ができない

ＯｐｅｎＭＰは共有メモリ型計算機の標準規格として用いられているプログラミングモデルである。ＯｐｅｎＭＰが開発されるまでは、それぞれの並列計算機ごとに並列プログラミングが開発されている状況で、非常に効率が悪かったため、芋奈計算機メーカーの技術者によって開発された。現在はほとんどの共有メモリ型計算機上でＯｐｅｎＭＰに準拠したプログラムを使用できる環境となった。ＯｐｅｎＭＰは特別なプログラミング言語ではなく、ＣやＣ＋＋、Fortranの文法中にあるpragma文等を用いて、並列化する部分をプログラム中で指示する型の並列プログラミングモデルである。一部のライブラリサブルーチンや関数を除いた実行文の文法は逐次型のプログラムと同じであるため、多くの場合で単一プロセッサ環境でも利用することが可能である。MPI と異なり、ライブラリやリンカ

などが付属されていないため、コンパイラの環境に依存性がある。逐次のプログラムが既にある場合、そのプログラムをほとんど変更することなく並列プログラムに移行することができる点もＯｐｅｎＭＰの特徴である。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。これらの特徴から、ＯｐｅｎＭＰは手軽な並列プログラミングの代表格であるといえる。

ＯｐｅｎＭＰは fork-join による並列実行モデルを用いている。ＯｐｅｎＭＰで記述されたプログラムは、マスタスレッドと呼ばれる単一のスレッドで実行を開始する。マスタスレッドは並列構文が現れるまで逐次リージョンを実行し、並列指示文に達すると複数のスレッドからなるチームを作成し、そのチームのマスタになる。全てのスレッドにより並列に実行される部分を並列リージョンという。また、ワークシェアリング構文に対応するブロックは、全スレッドによって実行されなければならず、構文に no wait 指示節が指定されていない場合、構文の最後で暗黙のバリア同期をチーム内の全スレッドに対して行い、マスタスレッドのみが実行を続ける。図３に fork-join モデルの実行の流れを示す。

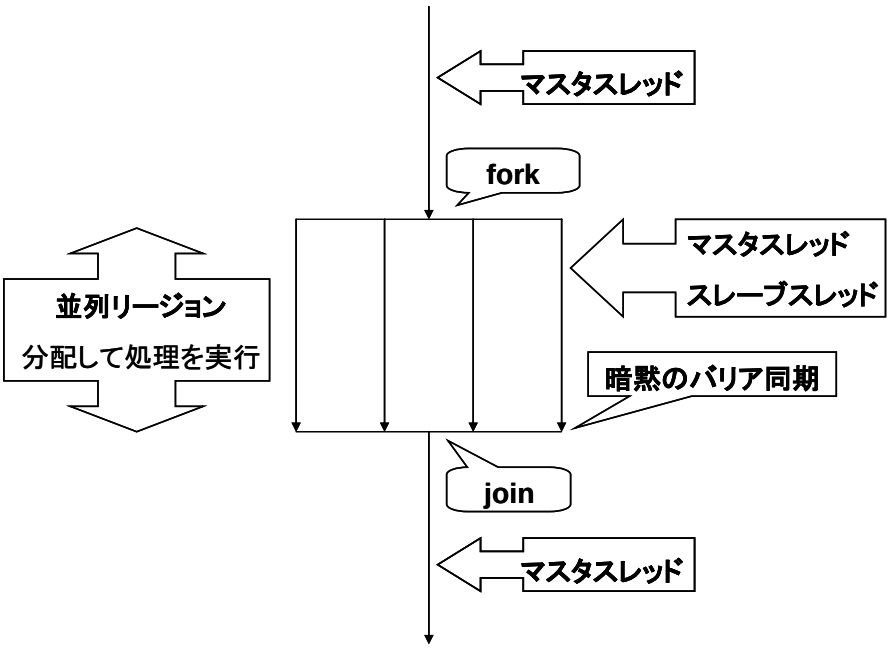


図 3：fork-join モデル

２．３．３　実験環境

（1）　ＳＲ８０００

ベクトル型プロセッサを搭載し、大量の配列処理を得意とするスーパーコンピュータの実行性能と高さと並列コンピュータの高いスケーラビリティを併せ持つスーパーテクニカルサーバである。

ノード数２で、１６台のノードを１次元クロスバネットワークで接続している。ノード間の最高転送速度は１ＧｂｐｓでＩ/Ｏ接続は Ethernet で接続している。

（2）　Ｒａｐｔｏｒ

Raptor は Computer Host として６台、それらを管理する Server Host として１台を設置する。Ethernet で Server とクラスタ６台（Computer Host）を接続する。

Server Host は Xeon の 2.8GHz、メモリは 2GB の SDRAM であり、Computer Host は PentiumⅣの３．２GHz、メモリは２GB の SDRAM である。Ethernet は最大帯域幅１Gbps、最小遅延６０μsec で、クラスタ間を接続する。以下図４に Raptor の構成図を示す。

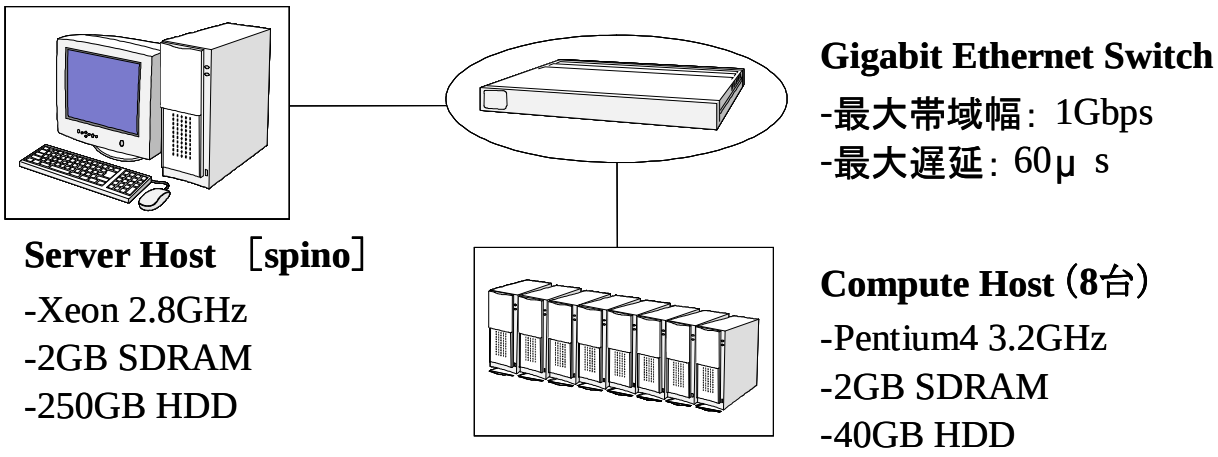


図４：Ｒａｐｔｏｒの構成図

（3）　Ｎｙｃｔｏ

Nycto は Computer Host として１台、それらを管理する Server Host として１台を設置する。Ethernet で Server とクラスタ６台（Computer Host）を接続する。

Server Host は PentiumⅣプロセッサの３GHz、メモリは 2GB の SDRAM であり、Computer Host（8CPU）は Quad Xeon の３GHz が２台、メモリは２GB の SDRAM である。Ethernet は最大帯域幅１Gbps で、クラスタ間を接続する。以下図５に Nycto の構成図を示す。

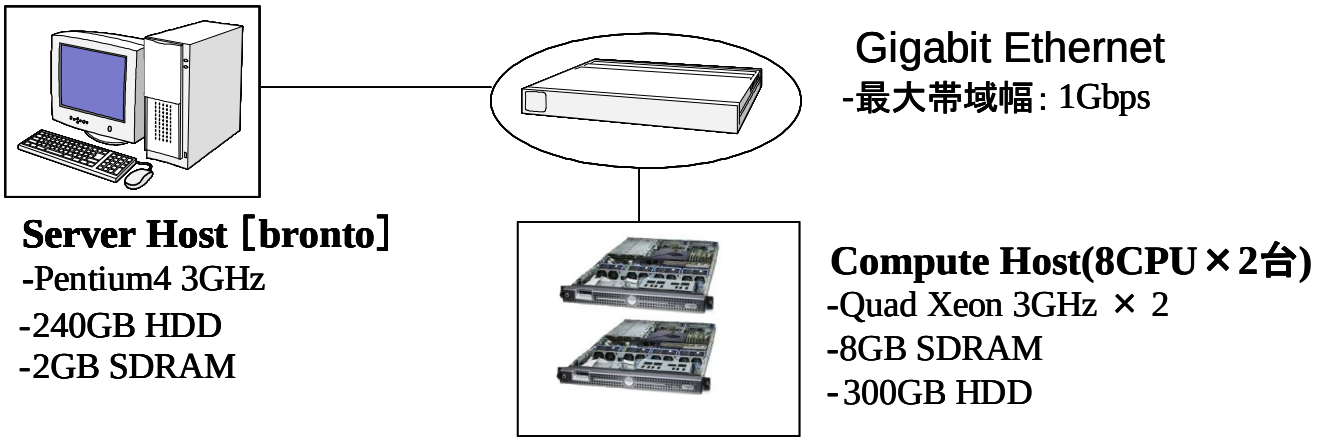


図５：Nycto の構成図

３．　ＭＰＩを用いたＮクイーン問題の並列化

３．１　実験

（１）並列化手法

ＭＰＩでは、マスター・スレーブに分ける必要がなく、各プロセスに１列目のどの位置にクイーンを配置するかで処理を並列に割り当てることができる。各プロセスは部分的な解の個数を求めた後、ＭＰＩのライブラリによって回収され、総和が求められる。以下にＮクイーンの並列化手法のイメージをクイーンの数を１４個、４プロセスでの実行する例を図５に示す。

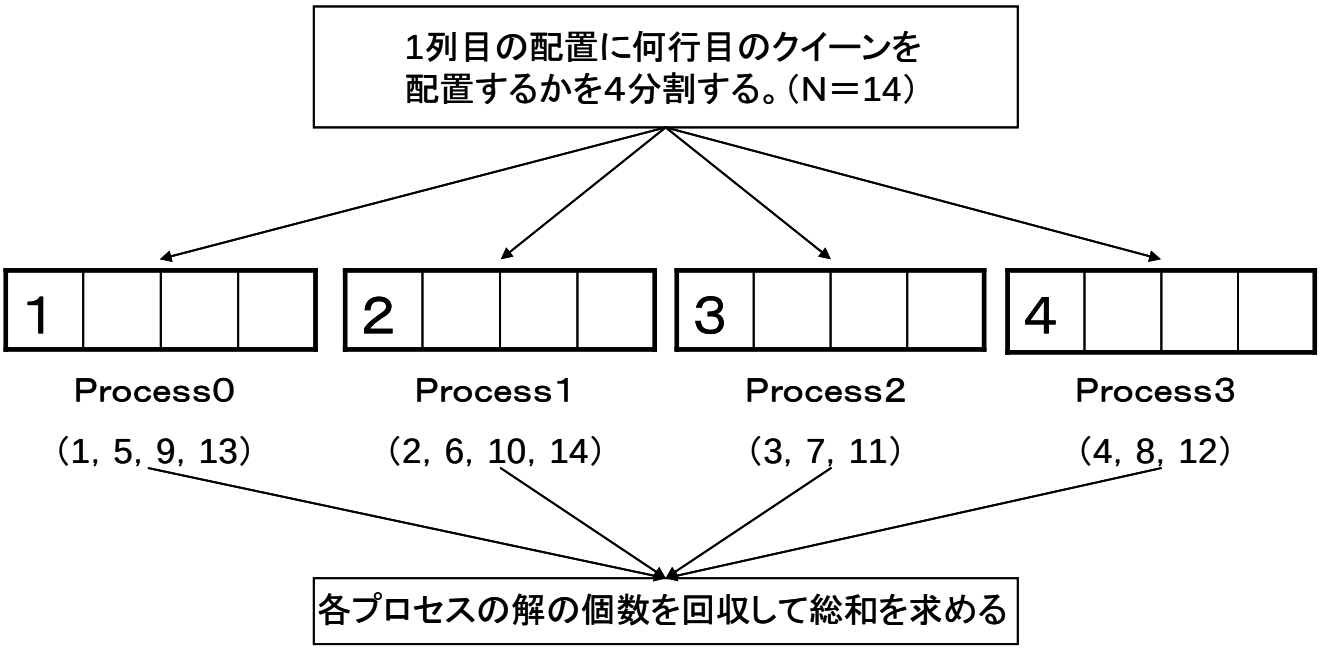


図５：ＭＰＩを用いた並列化のイメージ

（２）実験結果

以下の表１にＳＲ８０００での結果、表２にＲａｐｔｏｒでの結果、表３にＮｙｃｔｏの結果をそれぞれ示す。図６、図７、図８はそれぞれのマシンで、Ｎの数を変化させた速度向上比を示している。

表１：ＳＲ８０００上での実行結果

N=5(解の数＝10)

プロセス数	1	2	4	8	16
実行時間(秒)	0.0003	0.0004	0.0005	0.0005	0.0007
速度向上比	1	0.78	0.63	0.59	0.48

N=6(解の数＝4)

プロセス数	1	2	4	8	16
実行時間(秒)	0.0008	0.0059	0.0006	0.0006	0.0007
速度向上比	1	0.13	1.29	1.26	1.16

N=7(解の数＝40)

プロセス数	1	2	4	8	16
実行時間(秒)	0.0031	0.0019	0.0013	0.001	0.0008
速度向上比	1	1.63	2.32	3.16	3.6

N=8(解の数＝92)

プロセス数	1	2	4	8	16
実行時間(秒)	0.0144	0.0074	0.0041	0.0024	0.0025
速度向上比	1	1.95	3.51	5.98	5.75

N=9(解の数＝352)

プロセス数	1	2	4	8	16
実行時間(秒)	0.0745	0.0409	0.0242	0.0156	0.0096
速度向上比	1	1.82	3.09	4.79	7.73

N=10(解の数＝724)

プロセス数	1	2	4	8	16
実行時間(秒)	0.4043	0.2028	0.1207	0.0789	0.0475
速度向上比	1	1.99	3.35	5.12	8.5

N=11(解の数＝2680)

プロセス数	1	2	4	8	16
実行時間(秒)	2.35	1.29	0.65	0.44	0.24
速度向上比	1	1.82	3.61	5.34	9.79

N=12(解の数＝14200)

プロセス数	1	2	4	8	16
実行時間(秒)	14.87	7.46	3.8	2.49	1.33
速度向上比	1	1.99	3.91	5.97	11.18

N=13(解の数＝73712)

プロセス数	1	2	4	8	16
実行時間(秒)	98.06	52.55	28.76	14.6	8.38
速度向上比	1	1.87	3.41	6.72	11.7

N=14(解の数＝365596)

プロセス数	1	2	4	8	16
実行時間(秒)	685.16	340.86	191.52	101.2	53.03
速度向上比	1	2	3.58	6.77	12.92

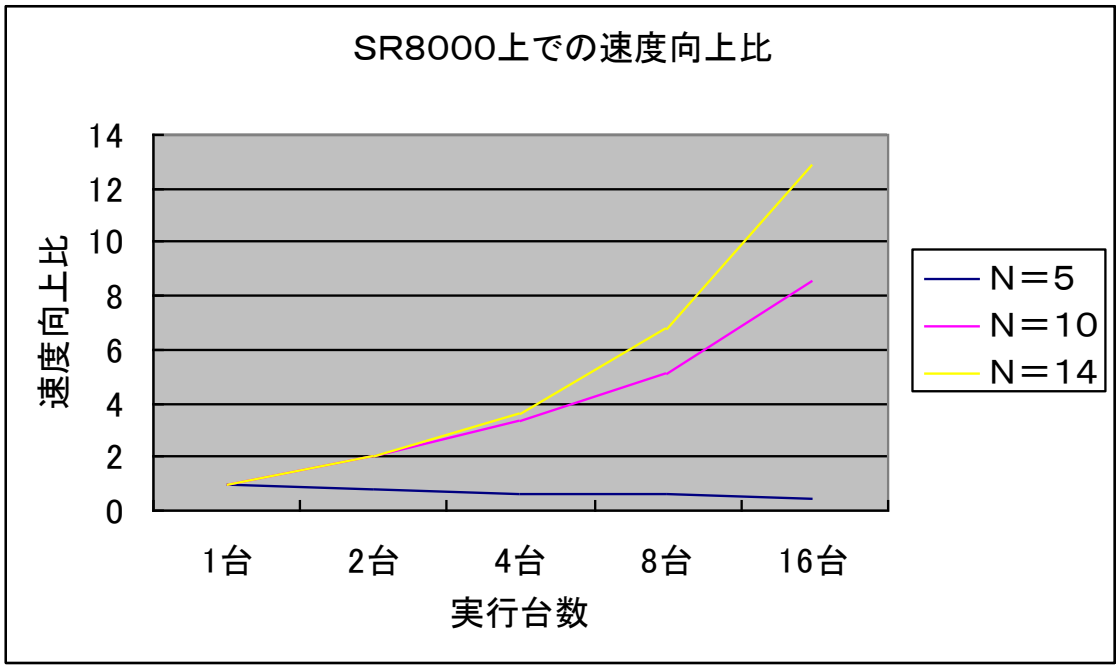


図６：SR800 上での速度向上比

表 2：Ｒａｐｔｏｒ上での実行結果

N=5(解の数＝10)

プロセス数	1	2	4
実行時間(秒)	0.0001	0.0002	0.0003
速度向上比	1	0.49	0.35

N=6(解の数＝4)

プロセス数	1	2	4
実行時間(秒)	0.0001	0.0002	0.0005
速度向上比	1	0.5	0.21

N=7(解の数＝40)

プロセス数	1	2	4
実行時間(秒)	0.0002	0.0002	0.0005
速度向上比	1	0.76	0.34

N=8(解の数＝92)

プロセス数	1	2	4
実行時間(秒)	0.0005	0.0006	0.0007
速度向上比	1	0.79	0.69

N=9(解の数＝352)

プロセス数	1	2	4
実行時間(秒)	0.002	0.0017	0.0015
速度向上比	1	1.14	1.29

N=10(解の数＝724)

プロセス数	1	2	4
実行時間(秒)	0.0104	0.0083	0.0052
速度向上比	1	1.25	1.99

N=11(解の数＝2680)

プロセス数	1	2	4
実行時間(秒)	0.0621	0.0414	0.0321
速度向上比	1	1.5	1.93

N=12(解の数＝14200)

プロセス数	1	2	4
実行時間(秒)	0.363	0.301	0.184
速度向上比	1	1.21	1.97

N=13(解の数＝73712)

プロセス数	1	2	4
実行時間(秒)	2.542	1.841	1.327
速度向上比	1	1.38	1.92

N=14(解の数＝365596)

プロセス数	1	2	4
実行時間(秒)	16.624	11.962	8.709
速度向上比	1	1.39	1.91

N=15(解の数＝2279184)

プロセス数	1	2	4
実行時間(秒)	128.21	97.765	60.28
速度向上比	1	1.31	2.13

N=16(解の数＝14772512)

プロセス数	1	2	4
実行時間(秒)	938.62	716.93	439.3
速度向上比	1	1.31	2.14

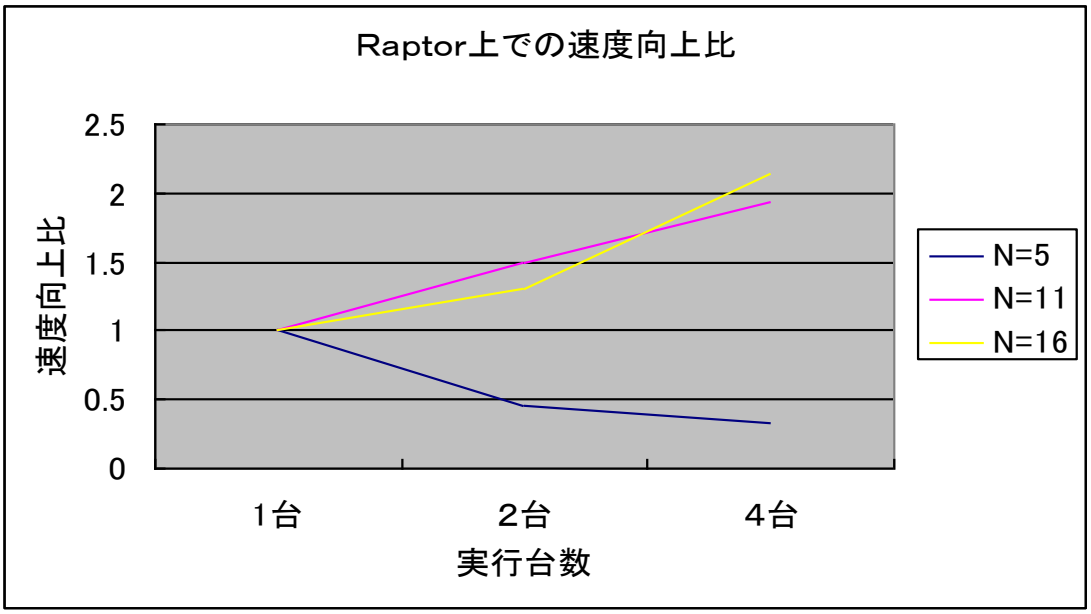


図 7：Raptor 上での速度向上比

表 3：Ｎｙｃｔｏ上での実行結果

N=5(解の数＝10)

プロセス数	1	2	4	8
実行時間(秒)	0.000012	0.000025	0.000038	0.000057
速度向上比	1	0.48	0.32	0.21

N=6(解の数＝4)

プロセス数	1	2	4	8
実行時間(秒)	0.000034	0.000033	0.00004	0.000075
速度向上比	1	1.03	0.85	0.45

N=7(解の数＝40)

プロセス数	1	2	4	8
実行時間(秒)	0.000141	0.000093	0.000066	0.000083
速度向上比	1	1.52	2.14	1.7

N=8(解の数＝92)

プロセス数	1	2	4	8
実行時間(秒)	0.000679	0.000359	0.000208	0.000141
速度向上比	1	1.89	3.26	4.82

N=9(解の数＝352)

プロセス数	1	2	4	8
実行時間(秒)	0.00359	0.001974	0.001163	0.000733
速度向上比	1	1.82	3.09	4.9

N=10(解の数＝724)

プロセス数	1	2	4	8
実行時間(秒)	0.01593	0.009785	0.0058	0.00375
速度向上比	1	1.63	2.75	4.25

N=11(解の数＝2680)

プロセス数	1	2	4	8
実行時間(秒)	0.1145	0.06186	0.03183	0.0208
速度向上比	1	1.85	3.6	5.5

N=12(解の数＝14200)

プロセス数	1	2	4	8
実行時間(秒)	0.7167	0.3585	0.1836	0.1251
速度向上比	1	2	3.9	5.73

N=13(解の数＝73712)

プロセス数	1	2	4	8
実行時間(秒)	4.7183	2.5021	1.3757	0.7407
速度向上比	1	1.89	3.43	6.37

N=14(解の数＝365596)

プロセス数	1	2	4	8
実行時間(秒)	32.7121	16.357	9.1017	4.7515
速度向上比	1	2	3.59	6.88

N=15(解の数＝2279184)

プロセス数	1	2	4	8
実行時間(秒)	240.2952	126.5532	66.7739	32.7931
速度向上比	1	1.9	3.6	7.33

N=16(解の数＝14772512)

プロセス数	1	2	4	8
実行時間(秒)	1863.265	931.7487	471.4101	240.0662
速度向上比	1	2	3.95	7.76

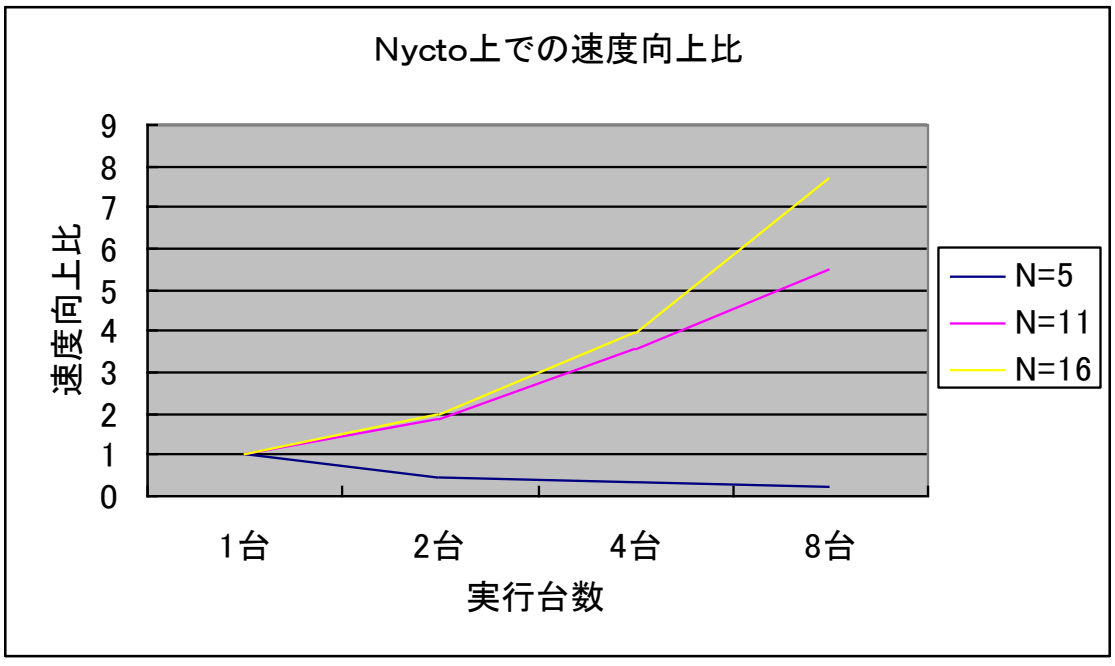


図８：Ｎｙｃｔｏ上での速度向上比

３．２　　考察

３種類のマシンの全てにおいて、計算量が増えるにしたがって理想的な速度向上比が得られるようになってくるということが言える。SR8000 ではNが６、Raptor ではNが９、Nycto ではNが６以上の数字になったところから速度の向上がみられる。Nがこれ以下の値をとる場合は並列処理をしているにも関わらず、速度が遅くなってしまっている。Raptor での実行結果の表を見ると、Nが１０以上の場合では速度向上比に向上がほとんど見られない。こういったことから、ＭＰＩを用いた並列プログラミングをする際には、計算量とマシンのスペックに応じて、最適なシステムを構築することが重要であると言える。そして、速度向上比が得られる計算量でさえあれば、特に１６台実行までは実行台数を増やせば増やすほど、理想的に速度が向上していくと考えることができる。

以下にNが１４の場合のマシン別の速度向上比のグラフを示す。

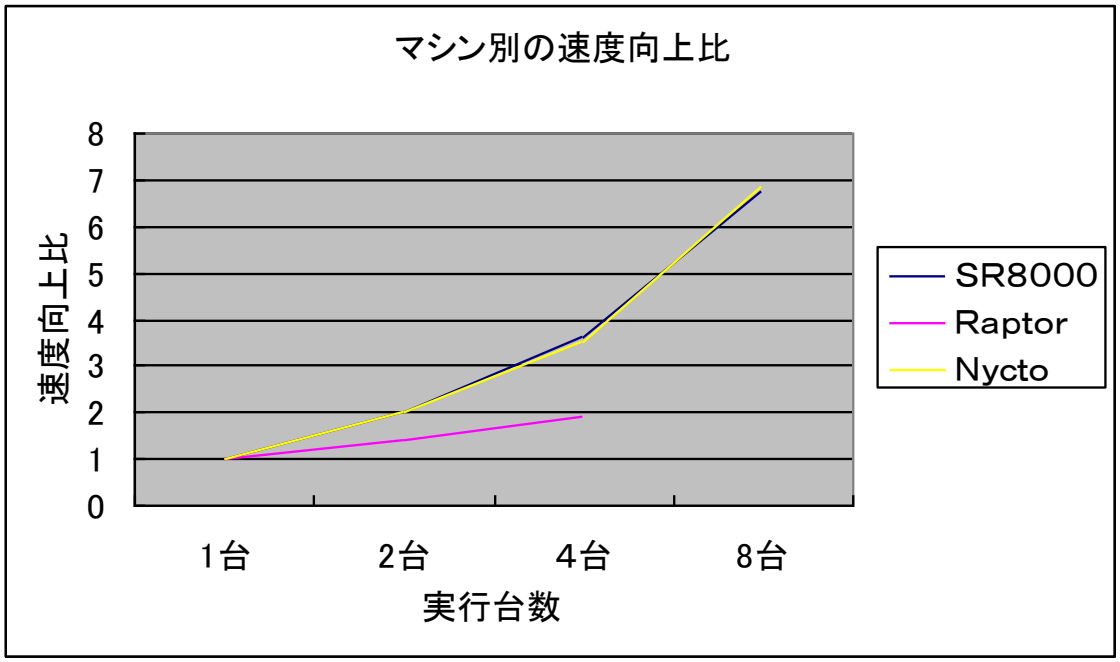


図９：マシン別の速度向上比

前章の速度の表から明らかなように、実行時間は Raptor のみが圧倒的に時間がかかっていることから実行時間はマシンの性能に依存していることが分かる。

４．　ＯｐｅｎＭＰを用いたＮクイーン問題の並列化

４．１　実験

（１）並列化手法

（２）実験結果

以下の表４にＲａｐｔｏｒでの結果、表５にＮｙｃｔｏの結果をそれぞれ示す。

表４：Ｒａｐｔｏｒ上での実行結果

N=5(解の数＝10)

プロセス数	1	2	4
実行時間(秒)	0.000021	0.026	0.0349
速度向上比	1	0.000808	0.000602

N=6(解の数＝4)

プロセス数	1	2	4
実行時間(秒)	0.000047	0.0269	0.0349
速度向上比	1	0.001747	0.001347

N=7(解の数＝40)

プロセス数	1	2	4
実行時間(秒)	0.000174	0.0209	0.0374
速度向上比	1	0.008325	0.004652

N=8(解の数＝92)

プロセス数	1	2	4
実行時間(秒)	0.000874	0.0269	0.0389
速度向上比	1	0.032491	0.022468

N=9(解の数＝352)

プロセス数	1	2	4
実行時間(秒)	0.00452	0.0269	0.0319
速度向上比	1	0.16803	0.141693

N=10(解の数＝724)

プロセス数	1	2	4
実行時間(秒)	0.0241	0.0279	0.0409
速度向上比	1	0.863799	0.589242

N=11(解の数＝2680)

プロセス数	1	2	4
実行時間(秒)	0.1405	0.102	0.0749
速度向上比	1	1.377451	1.875834

N=12(解の数＝14200)

プロセス数	1	2	4
実行時間(秒)	0.883	0.469	0.314
速度向上比	1	1.882729	2.812102

N=13(解の数＝73712)

プロセス数	1	2	4
実行時間(秒)	5.835	3.182	2.256
速度向上比	1	1.833752	2.586436

N=14(解の数＝365596)

プロセス数	1	2	4
実行時間(秒)	40.674	20.364	14.713
速度向上比	1	1.997348	2.764494

N=15(解の数＝2279184)

プロセス数	1	2	4
実行時間(秒)	298.028	160.102	102.014
速度向上比	1	1.861488	2.921442

N=16(解の数＝14772512)

プロセス数	1	2	4
実行時間(秒)	2303.349	1153.691	752.725
速度向上比	1	1.996504	3.060013

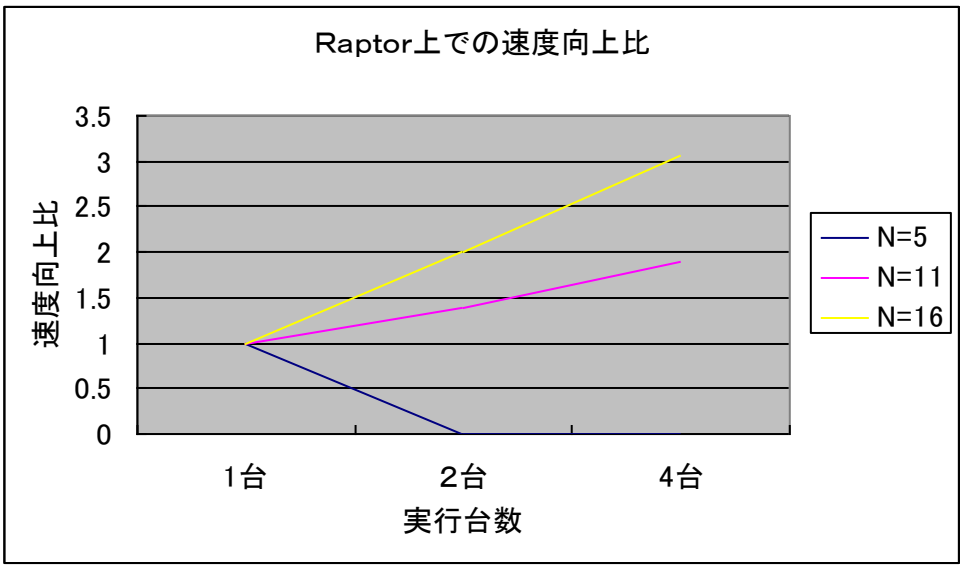


図９：Raptor 上での速度向上比

表５：Ｎｙｃｔｏ上での実行結果

４．２　　考察

５．　ＭＰＩとＯｐｅｎＭＰの比較

６．おわりに

謝辞

参考文献

［1］ 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

［2］ 亀井雄介：ＰＣクラスタ上でのＡＡＣエンコーダの並列化、立命館大学理工学部電子情報デザイン学科卒業論文、２００８

［3］ 多田修司：ＰＣクラスタ上でのＯｐｅｎＭＰによるマンデルブロ集合の並列化、立命館大学理工学部情報学科卒業論文、２００５

［4］ ＰＣクラスタ超入門

［5］ Ｓｔｅｃｖｅ　Ｏｕａｌｌｉｎｅ：C実践プログラミング、オーム社、２００１

http://www.ic-net.or.jp/home/takaken/nt/queen/