卒業論文

# OpenMP による数独パズルの並列化

氏　　名：松岡 毅
学籍番号：2210030362-4
指導教員：山崎 勝弘 教授
提 出 日：2007 年 2 月 19 日

立命館大学 理工学部 情報学科

**内容概要**

並列処理には大規模な問題でも計算時間を大幅に短縮できるというメリットがあり、単一プロセッサの速度限界が近づく中で欠かせない技術の１つである。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なので、今後並列プログラミングの主流になると期待されている。また、高性能な PC が安価で手に入るようになり、Myrinet(米国 Myricom 社により開発・販売されているネットワーク。4Gbps 高速通信が可能であり、クラスタシステムのインターコネクトとして広く利用されている)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。本論文では、並列処理の応用研究として Score 型 PC クラスタ(RWCP が開発した、PC クラスタ向けの並列オペレーティングシステム。クラスタシステム上に SCore をインストールすると、ノード間の効率的な相互通信や、複数ユーザの並列プログラムの時分割実行などが可能となる)上での数独パズルにおける OpenMP による並列化プログラミングについて述べる。数独パズルの並列化は、スレッド数 8 台で実行した。１つの解を初期状態として、３つのパターンの問題について実験する。数独パズルの解の総数をより早く見つけるために PC クラスタを使用した。空欄の条件によって速度向上の比率は違うが、速度向上は得ることができた。

目次

## 図目次



## 表目次

## １．はじめに

複数の計算機を同時に利用して、大規模な計算を高速に処理する並列処理の研究の応用分野として、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。その中でも気象予測、環境問題、流体計算は並列処理の活用が広がると考えられている。

並列計算機には３つのメモリモデルがある。複数のプロセッサがメモリバススイッチ経由で、主記憶に接続された形態を持つ共有メモリモデル(SMP)、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を持ち、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ができる共有分散メモリモデル、及び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を持ち、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ができない分散メモリモデルがある。

近年、汎用の PC を高速のネットワークで結合した PC クラスタが急速に広まっている。PC クラスタの長所として、プロセッサの数に比例して実行速度が向上する。1 台ではメモリ不足で解けないような計算が、並列化によって可能になる。複数の計算機で結果を照合できる。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。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ティングシステムは、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 のプログラミングが可能になっている。[3]

本研究では、データ量が多く、並列処理の性質に向いた数独パズルについて OpenMP を用いて並列化を行う。数独パズルは、2005 年にイギリスでブームが起こった後、あっという間に世界中で大人気のパズルゲームとなった。日本でも根強い人気があり、数独専門の雑誌が何冊も出版されている。

今回の実験で数独パズルを選んだのは、問題をどうやって作っているのか、そしてどのくらい解が存在するのかという点である。数独パズルは、9×9 の方眼状のマスを使い、同じ縦列・横列・３×３のブロック内に１から９までの数字を重複しないように埋めるという数字パズルである。一般の問題としては、１つの解を求めるパズルである。しかし、今回は数独パズルの基となる解はいくつ存在して、どのように作っているのかという発想のもとで、実験として数独パズルの空欄の数を増やし、それに対する解が何通りあるかを調べ、より速く解を求めるために並列処理を用いて実験した。並列処理では、ブロック分割を活用した。実験は、３つのパターンで行なった。１つ目は、１行１列目から右方向に０

を代入していき、０が４５個～４９個の場合についてのパターン。２つ目は、ランダムに０を代入するパターン（０の個数は５９個)。３つ目は、９×９の枠を０にし、中心から５×５に０を代入するパターン(０の個数は５７個)である。

本論文では､2 章では PC クラスタと並列プログラミングについての説明を示す｡3 章で、数独パズルの説明や OpenMP による並列化手法について記述している。4 章においては、実験の結果や考察を示している。全体のまとめは、５章に記した。

## ２．ＰＣクラスタと並列プログラミング

### 2.1　ＰＣクラスタ

(１)PC クラスタについて

PC クラスタとは、PC 単体では性能的にはスーパーコンピュータには及ばないが、複数の PC をネットワークで接続し、仮想的に 1 台の並列コンピュータとして利用することでパフォーマンスを向上させた PC 群のことを言う。

近年では PC の処理能力が高まり、大幅に低価格化が進んでいる。そのため非常にコストパフォーマンスに優れたシステムの構築が可能である。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 1 台のマシンとして、並列プログラムを走らせるような PVM(Parallel Virtual Machine)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言える。

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、比較的安価に高速ネットワーク構築が可能となった。さらに、Myricom 社の Myrinet などの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。これにより、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 クラスタの実行が可能となった。この時期に MPI フォーラムが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」を目指して作られたメッセージ通信の API である MPI(Massage Passing Interface)が広く普及した。現在では、複数のプロセッサを搭載した SMP 型の WS や PC が容易に入手できるようになり、これらをベースにした SMP クラスタが登場している。

(２)本研究室の PC クラスタ

本研究室では、Score 型の Raptor という PC クラスタを使用している。Raptor の構成を図 1 に示す。Score 型 PC クラスタは、RVCP(Real World Compurting Project)が開発した、高性能通信ライブラリを持つ SCore と呼ばれるクラスタシステムソフトウェアを用いて構築された、専用並列マシンと同等の性能を有するクラスタシステムである。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)を接続する。Server Host は Xeon2.8GHz、メモリは 2GB の SDRAM であり、Compute Host は PentiumIV プロセッサの 3.2ＧHz、メモリは 2GB の SDRAM である。Ethernet は最大帯域幅 1Gbps、最大遅延 60$\mu$sec であり、クラスタ同士を接続する。

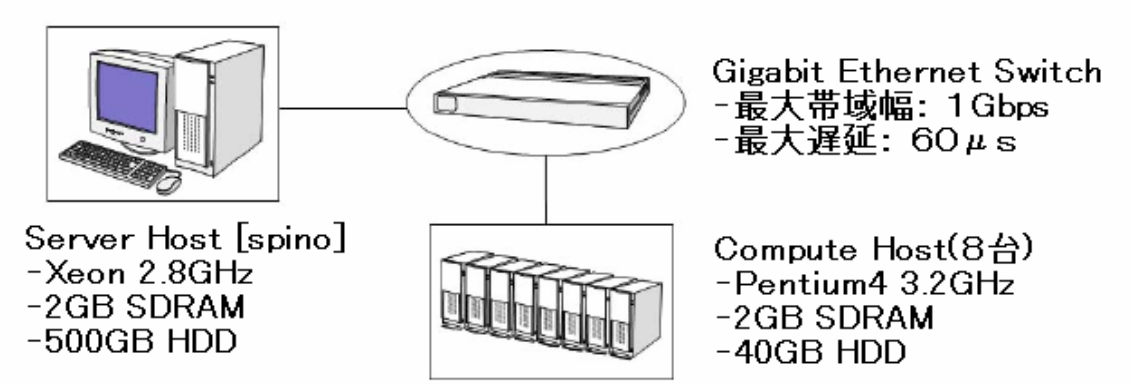


図 1：Raptor の構成

## 2.2 並列プログラミング

並列プログラミングは、大きく２つに分けることができる。１つは分散メモリ並列プログラミング、もう１つは共有メモリ並列プログラミングである。分散メモリ並列プログラミングとして挙げられるのは、PVM と MPI のメッセージパッシング型のプログラミングである。共有メモリ並列プログラミングとして挙げられるのは、OpenMP の従来のプログラミング言語にコンパイラ指示文として並列処理の指示を挿入するプログラミングである。

(１)PVM(Parallel Virtual Machine)

PVM は米国のオークリッジ国立研究所を中心に開発された、メッセージパッシングによる並列計算を行うためのソフトウェアであり、動作するマシンの種類が多く、フリーで手に入るソフトウェアで広く利用されるようになってきている。PVM とは、仮想並列計算機を意味している。

PVM は、ネットワークに接続された異機種 UNIX コンピュータ群を単一の並列コンピュータとして利用することを可能にするソフトウェアである。これにより、多数のコンピュータの持つ計算パワーを１つの大規模計算問題に結集して処理を行うことができる。

PVM は、アプリケーション、マシン及びネットワークレベルでの異機種間利用をサポートする。すなわち、PVM の下では、アプリケーションを構成するタスクは、問題に最も適したアーキテクチャを利用することができる。PVM は、異なるコンピュータ間の整数あるいは浮動小数点の表現の違いを吸収するための、データ変換を扱うことができる。そして、多様なネットワークで接続されたバーチャルマシンを実現する。

PVM はソフトウェアシステムの構成は、大きく分けて２つある。１つはデーモンである。デーモンは、バーチャルマシンを構成するすべてのコンピュータ上に常駐する。ユーザは、ログイン可能でさえあれば、どんなコンピュータにも PVM をインストールすることができる。PVM アプリケーションを実行する場合、最初にユーザはどれか１つのコンピュータで PVM を起動する。次に、ユーザが定義したバーチャルマシンを構成するコンピュータそれ

ぞれにおいて、順次 PVM デーモンを起動する。最後に、どれか１つのコンピュータに表示された UNIX プロンプトに対してコマンドを入力することにより、PVM アプリケーションを実行する。PVM ソフトウェアシステムを構成するもう１つは、PVM インターフェース関数のライブラリである。ライブラリはメッセージパッシング、プロセスの生成、タスクの協調、及びバーチャルマシンの再構成のための関数を提供する。PVM を利用するためには、アプリケーションと、このライブラリを必ずリンクする必要がある。[2]

(２)MPI(Message Passing Interface)

MPI は、MPI フォーラムという任意参加の会議で議論された。PVM とは異なり、MPI はメッセージ通信の API 仕様であって、MPI という特定のソフトウェアがあるわけではない。MPI は２つの理由により事実上の標準としての力を持つことになった。１つは、MPI フォーラムには PVM を始めとする、主要メッセージ通信ライブラリの開発を行った研究者と、主要な並列計算機ベンダのほとんどが参加したこと、もう１つは、出来上がった仕様が数多くの有用な機能を持ち、多くの並列計算機や LAN 環境で高い性能を実現できる、優れたものに仕上がったことである。

MPI は、並列処理用のメッセージパッシングの器楽である。メッセージパッシングとは、メッセージと呼ばれる特定のデータ形式の受け渡しと、これらの一元的な管理に基づく通信手段の１つである。並列計算機の各 CPU 間では数値計算の過程で多くのデータの交換が行われているが、その交換手順を定めたものが MPI である。多くの並列計算機に標準実装され、またフリーウェアとして LAM 等のパッケージが入手可能である。[6]

(３)OpenMP

OpenMP は、1997 年に米国の OpenMP Architecture Review Board が、Fortran をベースとして API の仕様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を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、広範囲に広まっている。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。[7]

次の図２は OpenMP プログラミングの実行モデルの fork-join モデルである。

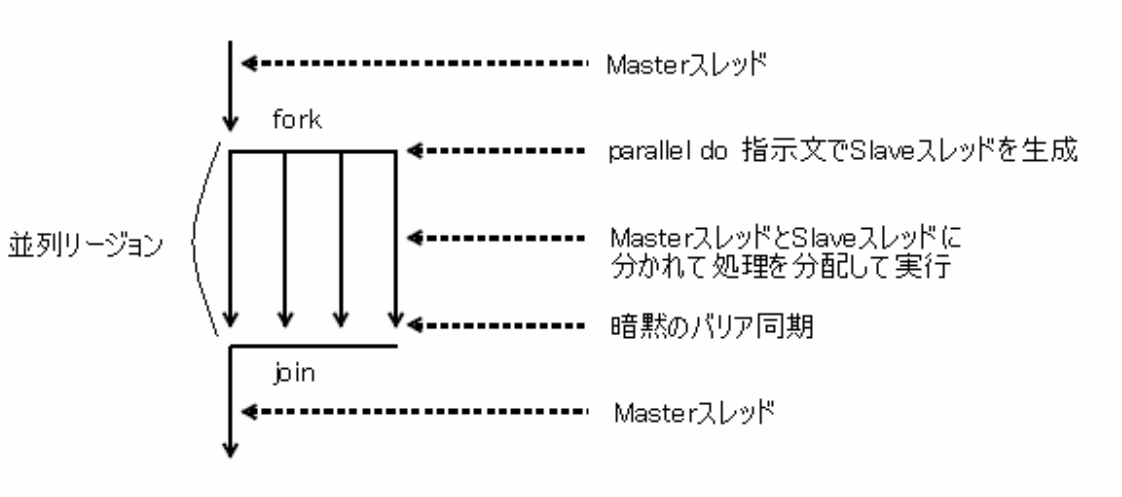


図 2：fork-join モデル

## ３．OpenMPによる数独パズルの並列化

### 3.1　数独パズルの定義

(１)数独パズルの歴史

数独パズルは、アメリカ人建築家であったハワード･ガーンズが匿名で考察したものである。これは１８世紀にスイスの数学者であるレオンハルト･オイラーが考案した、ラテン方陣あるいはオイラー方陣(ｎ行ｎ列の表にｎ個の異なる記号を、各記号が各行および各列に1 回だけ現れるように並べたものである)と呼ばれるものに、３×３のブロックという新たな制限を付け加え、ペンシルパズルとしたものである。これは「Number Place」の名前で1979 年に米国の大手パズル出版社であるデル･マガジン社から初めて出版された。日本には、ニコリの「月間ニコリスト」誌 1984 年４月号で、「数字は独身に限る」の題で初めて紹介された。「数独」という名称はニコリ社の登録商標であり、日本では一般にナンバープレイス（ナンバープレース、ナンプレ）という名称がよく使われている。

数独パズルの世界的な流行は、1997 年にニュージーランド人のヴァイン･グールドが日本の書店で数独の本を手に取ったことに始まる。グールドは数独をコンピュータで自動生成してイギリスの新聞タイムズに売り込み、2004 年 11 月 12 日から「SuDoKu」の名で連載を始めた。2005 年 4 月から５月にかけてブームに火が付き、イギリスの主要日刊紙に軒並み掲載されるという状況になった。その年の７月にはテレビ局スカイ･ワンが、数独をテーマにしたテレビ番組を放映した。この人気は海外にも行渡り、パズルとしては 1980 年ごろのルービック･キューブ以来の大流行となった。[11][12]

(２)数独パズルのルールと解法

ルールは至って簡単である。空欄のマスに１～９までの数値を入れる。その際に、縦列、横列、及び太線で囲まれた３×３のブロックに同じ数字が入ってはいけない。

問題として発表する時は、図３のように最初にヒントとして配置する数字が盤面に掲示されている。掲示されているのは２４～３２個ぐらいまでが多く、これ以上だと解くのが簡単になる。また、それ以下にするのは作成者の技量も必要となってくる。現状では、１７個の数独問題は見付かっているが、１６個は見付かっていない状態である。

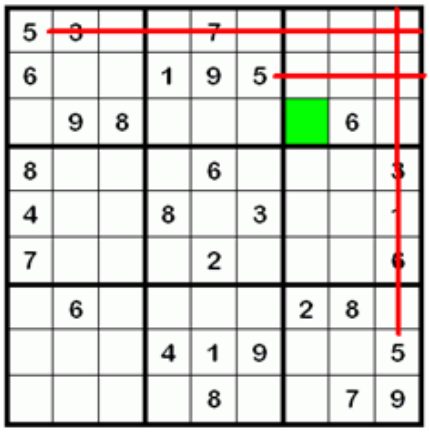


**図 3：数独パズルの例**

解法においては、１つのマスに注目して、そのマスに入る数字を限定する。１つの列または３×３のブロックに注目して、特定の数字が入るマスを探す。この２つで基本的な問題は解くことができる。しかし、中級以上の問題では数字が入るマスは確定しないが、あ

る制約により数字が入る箇所が限定され、決定することがある。具体例として図３を用いる。あるブロックとただ１つの数字に注目し、他のブロックで既に決定された列を候補から排除し、数字が入るマスを限定する。図３の上端横に伸びる赤い線の左端に５があり、上段右のブロックの上段には５は入らない。同じく、その下の短い横に伸びる赤い線の左端にも５があり、上段右のブロックの中段には５は入らない。さらに、下段右のブロックのまだ赤線の引かれていない２マスの内、１つは既に数字が確定しているので、残った部分の緑色のマスに必然的に５が入る。[11][12]

### 3.2　数独パズルのアルゴリズム

・入力データとしてファイルを１つ用意する。
そのファイルに数独のルールに従って、図４のような問題を入力する。
・０の部分は空欄を意味し、そこに１～９までの数値を入れて解を導く。
・０から右方向に順番に数値を入れていき、９行９列まで解が出てきたらカウントする。

```
0 0 0 0 0 0 0 0 0
0 5 8 0 0 0 3 9 0
0 1 7 6 0 3 2 4 0
0 0 2 3 0 1 6 0 0
0 0 0 0 0 0 0 0 0
0 0 3 5 0 4 8 0 0
0 2 4 9 0 6 7 1 0
0 7 1 0 0 0 4 3 0
0 0 0 0 0 0 0 0 0
```

図 4：入力データ例

図５は、図４の０が空欄の場合として表現している。
まず初めに①のように１行１列目に１を代入する。そして、縦列、横列、３×３のブロックを見て同じ数字が入っていないかを確認する。①は３×３のブロック内に１が入っているためにルール違反として１は入らない。次に②のように２を代入する。②はルールに従っているため問題ない。次は③のように１行２列目に１を代入する。これを繰り返して行き９行９列まで解が出たら１つカウントする。

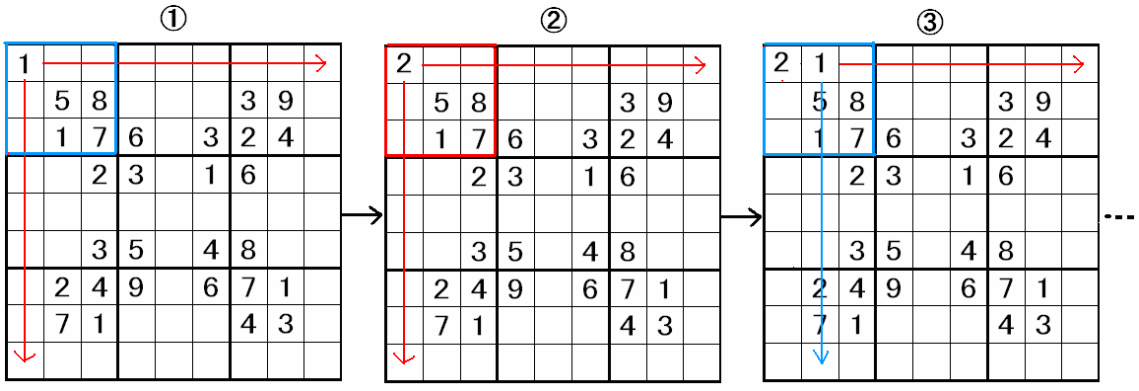


図 5：再帰関数についての例

### 3.3　OpenMPによる並列化手法

OpenMP の特徴として、既存の逐次プログラムをベースに、並列プログラムを作ることができる。指示文を使って、スレッドを生成･制御することができ、スレッドライブラリなどを使うよりも簡単にスレッドプログラミングができる。また、徐々に指示文を加えることにより、段階的に並列化することが可能である。OpenMP の指示文を無視することにより、元の逐次プログラムになり、逐次プログラムと並列プログラムを同じソースで管理することができる。

OpenMP の構文は、parallel 構文、ワークシェアリング構文、同期のための構文の３つに大きく分けることができる。以下に、基本的な構文を説明する。[4]

⑴parallel 構文

parallel 指示文は並列リージョンを定義する。並列リージョンとは、複数のスレッドによって並列実行される部分である。この指示文は、並列実行を開始する基本的な構文である。終了時には、暗黙のバリア同期が取られる。またいくつかの指示文では、ユーザが指示節を用いて、そのリージョンの実行中に変数のスコープ属性を制御することができる。以下は主な指示節である。

・shared･･･構文内で指定された変数がスレッド間で共有される。
・private･･･構文内で指定された変数をスレッドごとに持たせる。
・reduction･･･指定した変数に対し、それぞれのスレッドが保有している部分的な値を、指定された演算子によって同じ演算を行い、１つの結果としてまとめる。

⑵ワークシェアリング構文

ループなどを分割してスレッドに割り当て、それぞれのスレッドで計算する。OpenMP では、以下のようなワークシェアリング構文を定義している。いずれも終了時には、暗黙のバリア同期が取られる。

①for 構文

直後のforループの実行範囲を並列実行するものである。forループはcanonical（正規形）でなくてはならず、ループ制御変数は整数型で強制的にprivate属性になる。for構文は、何も指定しなかったらブロック分割になるが、スケジュール構文を使うことでサイクリック分割を実行できる。for構文の指示節で、scheduleの指定を行なう。Scheduleの種類は以下に示す。

・static･･･schedule(static,chunk_size)が指定された場合、ループはchunk_sizeで指定されたサイズのかたまりに分割される。生成された各チャンクは、チーム内のスレッドにスレッド番号順にラウンドロビン形式で割り当てられる。

・dynamic･･･schedule(dynamic,chunk_size)が指定された場合、ループはchunk_sizeで指定されたサイズのかたまりに分割される。スレッドに割り当てられたチャンクの演算が終了すると、残りのチャンクがなくなるまで動的に別のチャンクをスレッドに割り当てる。

・guided･･･shedule(guided,chunk_size)が指定された場合、最初大きなかたまりをスレッドに割り当て、処理が進むにつれて、かたまりを小さくしながら割り当てる。スレッドに割り当てられたチャンクの演算が終了すると、残りのチャンクがなくなるまで動的に別のチャンクをスレッドに割り当てる。

･runtime･･･schedule(runtime)が指定された場合、スケジューリングは実行時に決定する。

②section構文

それぞれのsectionを並列に実行するというものである。つまり、下のようなプログラムならsection1とsection2の内容が同時に実行されることになる。

```
#pragma omp sections
{
#pragma omp section
{ …section1…}
9
#pragma omp section
{…section2…}
}
```

③single構文

チーム内の１つのスレッド（必ずしもマスタースレッドとは限らない）のみで実行されることを指示するものである。構文は以下の通りである。

```
#pragma omp single
{
…statement…
}
```

（３）同期構文

①master構文

マスタースレッドのみで実行されることを指示するものである。マスター以外のスレッドは、masterが指定されたステートメントを実行しない。マスターセクションの入り口と出口では、暗黙のバリア同期は取られない。

②critical構文

排他的に実行されるcritical sectionを指定する指示文である。大域的な名前を付けることができ、同じ名前のcritical sectionは排他的に実行される。

③barrier指示文

バリア同期を取る指示文である。全てのスレッドが同期点に達するまで待つ。parallel構文とワークシェアリング構文においては、nowait指示節が指定されない限り、終了時に暗黙のバリア同期が取られる。

④atomic構文

複数の同時書き込みを行なう可能性のあるスレッドに対して、指定されたメモリの更新を逐次で行なうものである。

⑤flush 構文

メモリ上にあるオブジェクトに対して、矛盾しないメモリ参照を指示するものである。つまり、指定した変数について、メモリの一貫性を保証する。

今回の実験においては、ワークシェアリング構文のfor構文を用いてブロック分割をしている。図６に例を示す。図６の説明として、今回の数独パズルにおける数値は１～９である。なので、１～９を３分割にする場合であると１つ目のノードに１～３、２つ目のノードに４～６、３つ目のノードは残りの７～９とそれぞれのノード間で解の総数を導く。そして、最後に３つのノードの合計を出力する。

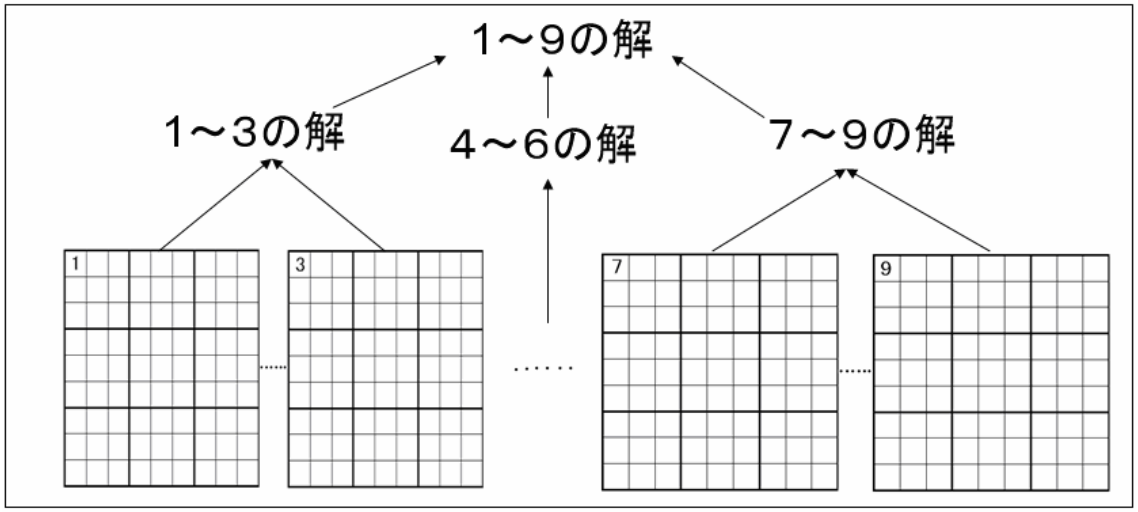


**図 6：３ノードの場合のブロック分割**

## ４．実験

### 4.1　実験条件

初期状態（図７）の状態から３つのパターンを実験した。
１つ目は、１行１列目から右方向に０を代入していき、０が４５個～４９個の場合についてのパターン。
２つ目は、ランダムに０を代入するパターン。
３つ目は、９×９の枠を０にし、中心から５×５に０を代入するパターンである。

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 |
| 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 1 | 4 | 3 | 6 | 5 | 8 | 9 | 7 |
| 3 | 6 | 5 | 8 | 9 | 7 | 2 | 1 | 4 |
| 8 | 9 | 7 | 2 | 1 | 4 | 3 | 6 | 5 |
| 5 | 3 | 1 | 6 | 4 | 2 | 9 | 7 | 8 |
| 6 | 4 | 2 | 9 | 7 | 8 | 5 | 3 | 1 |
| 9 | 7 | 8 | 5 | 3 | 1 | 6 | 4 | 2 |

図 ７：数独パズルの解

### 4.2　実行結果

１つ目のパターンである「０が４５個～４９個の場合の数独パズル」を図８に示す。０が４５個～４９個における実行結果、速度向上比をそれぞれ表１、表２に示す。速度向上比のグラフを図９に示す。

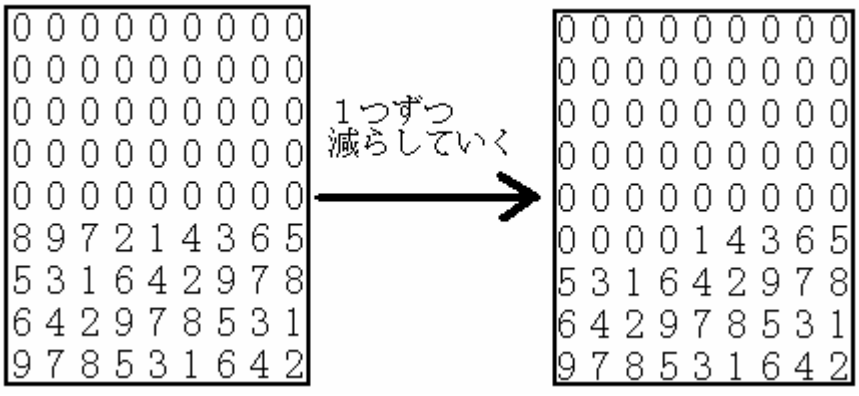


図 ８：０が４５個と４９個の場合の数独パズル

表 １：０が４５個～４９個における実行時間

| 0の個数 | 45個 | 46個 | 47個 | 48個 | 49個 |
|---|---|---|---|---|---|
| 解の総数 | 633312 個 | 633312 個 | 633312 個 | 1266624 個 | 3673248 個 |
| ノード1 | 35.15 | 74 | 201 | 620 | 1377 |
| ノード2 | 23.74 | 42 | 107 | 413 | 925 |
| ノード3 | 18.01 | 36 | 94 | 304 | 678 |
| ノード4 | 11.45 | 32 | 93 | 207 | 464 |
| ノード5 | 11.45 | 32 | 93 | 207 | 464 |
| ノード6 | 11.45 | 24 | 66 | 196 | 438 |
| ノード7 | 11.45 | 24 | 66 | 196 | 438 |
| ノード8 | 11.45 | 24 | 66 | 196 | 437 |

（単位：秒）

表 ２：０が４５個～４９個における速度向上比

| ノード数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 45個 | 1 | 1.48 | 1.95 | 3.07 | 3.07 | 3.07 | 3.07 | 3.07 |
| 46個 | 1 | 1.76 | 2.05 | 2.31 | 2.31 | 3.08 | 3.08 | 3.08 |
| 47個 | 1 | 1.88 | 2.14 | 2.16 | 2.16 | 3.04 | 3.04 | 3.04 |
| 48個 | 1 | 1.5 | 2.04 | 2.99 | 2.99 | 3.16 | 3.16 | 3.16 |
| 49個 | 1 | 1.49 | 2.03 | 2.97 | 2.97 | 3.14 | 3.14 | 3.15 |

（単位：倍）

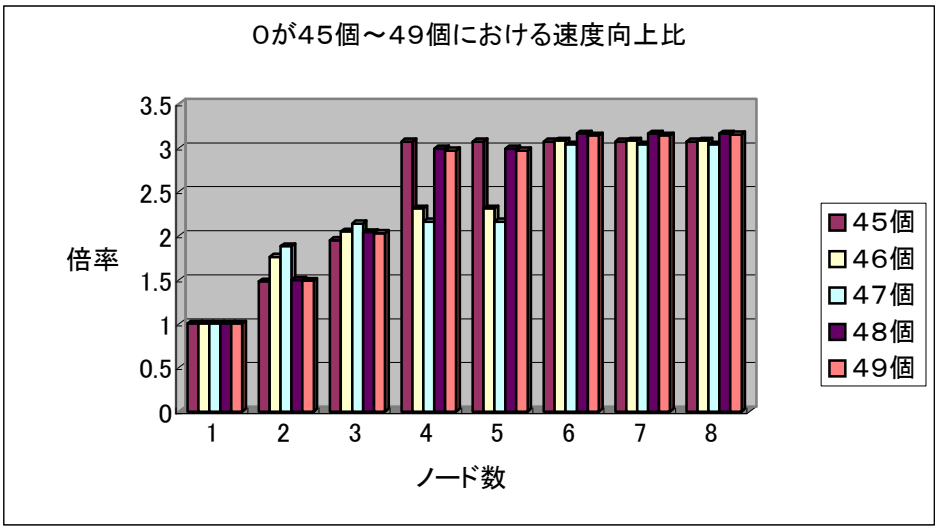


図 ９：０が４５個～４９個における速度向上比

２つ目のパターンである「０をランダムに代入した数独パズル」を図 10 に示す。(０の個数は５９個である)０をランダムに代入した実行結果と速度向上比を表３に示す。速度向上比のグラフを図 11 に示す。０をランダムにした解の総数は４６４０通りある。

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 6 | 7 | 0 | 0 | 0 | 2 | 3 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| 0 | 1 | 4 | 0 | 0 | 0 | 8 | 9 | 7 |
| 0 | 0 | 0 | 8 | 0 | 7 | 0 | 1 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 4 | 0 | 0 | 0 | 8 | 5 | 0 | 1 |
| 0 | 7 | 8 | 5 | 0 | 0 | 0 | 0 | 0 |

図 10：０をランダムに代入した数独パズル

表 3：０をランダムに代入した実行時間と速度向上比

| ノード数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 時間(秒) | 29.81 | 17.59 | 12.98 | 12.45 | 8.26 | 8.25 | 8.25 | 8.24 |
| 倍率(倍) | 1 | 1.69 | 2.29 | 2.39 | 3.6 | 3.61 | 3.61 | 3.61 |

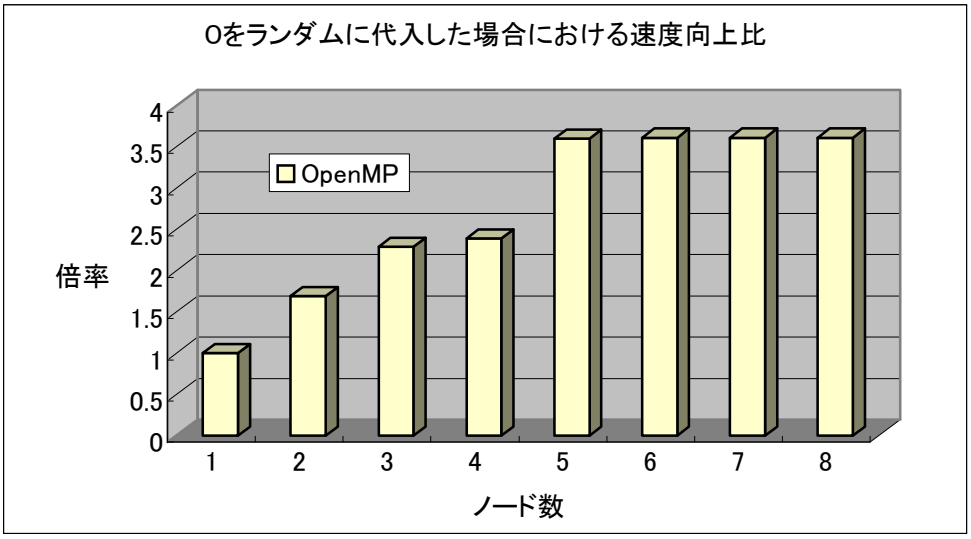


図 11：０をランダムに代入した場合における速度向上比

最後に「９×９の枠を０、中心から５×５に０を代入した数独パズル」を図１２に示す。(０の個数は５７個である)９×９の枠を０、中心から５×５に０を代入した実行結果と速度向上比を表４に示す。速度向上比のグラフを図１３に示す。今回の解の総数は 19,753,544 通りある。

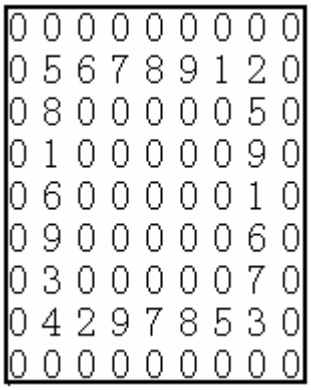


図 12：９×９の枠を０、中心から５×５に０を代入した数独パズル

表 4：9×9 の枠を０、中心から 5×5 に０を代入した実行時間と速度向上比

| ノード数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 時間(秒) | 1023 | 497 | 396 | 257 | 249 | 249 | 249 | 249 |
| 倍率(倍) | 1 | 2.05 | 2.58 | 3.98 | 4.1 | 4.1 | 4.1 | 4.1 |

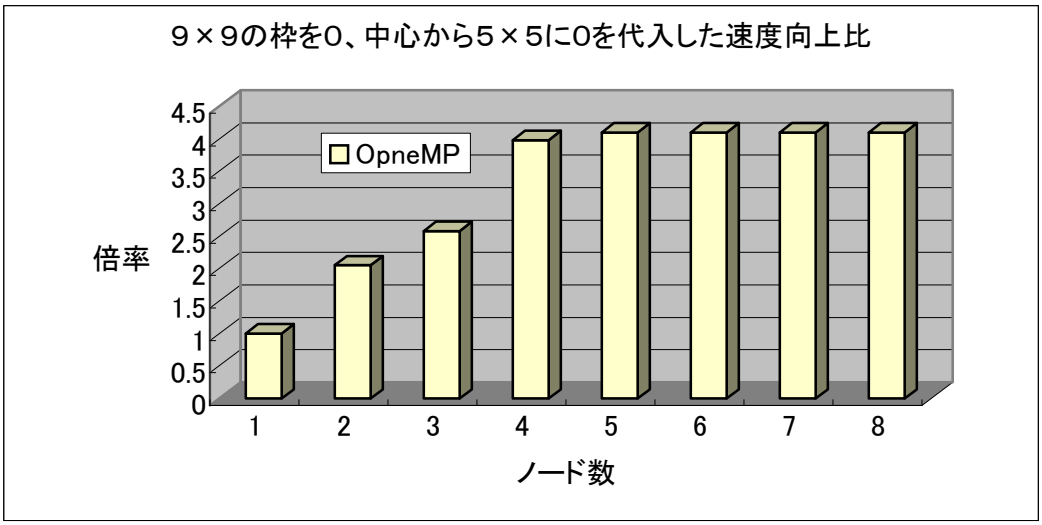


図 13：9×９の枠を０、中心から５×５に０を代入した速度向上比

### 4.3　考察

０を横方向に入れていく１つ目のパターンにおいて、最初の１行１列には、1,2,3,4,7 の５つの数字が入る。これを並列化した結果が表 1 にある。このパターンの実験を行なったのは、どのように解が増えていくかという点である。０が４５個～４７個の場合において、解の総数が同じなのは、6 行１列～７行３列までに入る数字が数独のルールにおいて１つしか不可能だからである。４７個から４８個にかけては２倍も解の総数が増え、４８個から４９個にかけては約３倍増えている。このようにハイペースで解が増えて行くことがわかった。本題の並列化においては、最大で 3.16 倍の速度向上を得ることができた。表 2 を見ると徐々にではあるが、解の総数が増えるにことで並列化のスピードが上がっていることがわかる。

０をランダムに代入していくパターンにおいては、最初の 1 行 1 列には、６以外の全ての数字が入る。これを並列化した結果が表 3 にある。解の総数が４６４０通りと少ないが綺麗な速度向上が得られた。最大で 3.61 倍の速度向上を得ることができた。

０を 9× 9 の枠、中心から 5× 5 に代入したパターンにおいて、最初の 1 行 1 列には、1,2,3,4,7,9 の６つ数字が入る。これを並列化した結果が表 4 にある。この場合は、解の総数が 19,753,544 通りで多いため実行時間がかなりかかった。最大で 4.1 倍と３つのパターンの中で一番速度向上を得ることができた。

今回の実験でわかったことは、解の総数を求めるのに時間がかかる問題は速度向上比が高いということ。ノード数が５～８の場合に、ほぼ同じ結果が返ってくること。ランダムに０を代入して問題を作成する場合、ブロック分割する箇所に入る数字を考えなくてはいけないこと。０の入れ方によって解の総数が桁違いに変わることである。

今回はできる時間内で結果を求めたので、全ての解の総数を確認することはできなかったが、解の総数がどのくらいあるかをインターネットで調べたところ数独パズルの答えとして現れる盤面の数は 6, 670, 903, 752, 021, 072, 936, 960(約 67 垓)通りあるということがわかった。[13]

## ５．おわりに

本研究では、ＰＣクラスタ上における数独パズルの並列化を行なった。理想的な並列効果は得られなかったが、計算時間は速くなった。最大で約 4.1 倍の並列効果を得ることができた。

今後の課題としては、もっと多くのパターンでの実験をすること。そして、より速く解を得るためのプログラミングを考える必要がある。プログラミングにおいては、数独パズルの特徴を考えてより並列化がしやすいように作る必要がある。Score のバージョンがアップされるとともに、OpenMP のコンパイラや、SCASH の性能も良くなっていけば、PC クラスタ上での OpenMP による並列プログラミングはさらに実用化されると考えられる。

**謝辞**

本研究の機械を与えてくださり、数々の助言を頂きました山崎勝弘教授に心より感謝いたします。また、本研究に関して貴重なご意見を頂きました、船附氏、難波氏、そして色々な面で貴重な助言を下さった高性能研究室の皆様に心より感謝いたします。

**参考文献**


[1] Omni OpenMP Compiler Project：http://phase.hpcc.jp/Omni/home.ja.html

[2] 湯浅太一、安村通晃、中田登志之：bit 別冊　はじめての並列プログラミング 共立出版、1998.

[3] 石川裕、堀敦史、原田浩、佐藤三久、住元真司、高橋俊行：Linux で並列処理をしよう、共立出版、2002.

[4] High Performance Programming : http://www.na.cse.nagoya-u.ac.jp/~reiji/lect/hpc02/

[5] 新実治男：OpenMP による並列プログラミング“入門”、京都産業大学ハイテクリサーチプロジェクト、スーパクラスタマシンの効率化とその応用、研究成果中間報告書、2003

[6] PC クラスタ超入門、超並列計算研究会、2000. http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/index.html

[7] OpenMP 入門. http://www.na.cse.nagoya-u.ac.jp/~reiji/lect/hpc02/OpenMPintro.html

[8] 佐藤三久：OpenMP. http://phase.hpcc.jp/Omni/openmp-tutorial/index.htm

[9] 南里 豪志, 渡部 善隆：OpenMP 入門(2)、九州大学情報基盤センター広報、2002. http://spring.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/OpenMP/intro.html

[10] 林雅樹、池上広済：PC クラスタの使用法と OpenMP 並列プログラミング 高性能計算研究室資料、2005.

[11] フリー百科事典『ウィキペディア（Wikipedia）』: http://ja.wikipedia.org/wiki/

[12] 数独パズル：http://assam.cims.hokudai.ac.jp/sample/samplen/numpla.htm

[13] 数独：http://ara.moo.jp/puz/sudoku.htm