卒業論文

SMP クラスタ上でのイラストロジックの並列化

氏　　名：宮本　裕章
学籍番号：2260040083-9
指導教員：山崎 勝弘 教授
提 出 日：2008 年 2 月 20 日

立命館大学 理工学部 電子情報デザイン学科

内容梗概

並列処理を行うことは、大規模な計算の時間短縮が可能にな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、共有メモリを有した SMP クラスタの普及に伴い、並列プログラミングは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言語として現在注目を集めているのが OpenMP である。

本論文では、並列処理の応用研究として SMP クラスタ上でのイラストロジックの並列化について述べる。

イラストロジックの並列化は OpenMP を用いて行い、1 スレッドから 4 スレッドで実行した。1000×1000 のパズルの問題において、4 スレッド実行により、ブロック分割で約 3.5 倍、サイクリック分割で約 3.85 倍の速度向上が得られた。

目次

図目次

表目次

1. はじめに

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、データマイニングなどが挙げられる。その中でも今後、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。

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。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、大規模な計算が可能、コストパフォーマンスが良い点が挙げられる。

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、主記憶に接続される形態の共有メモリや、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、プロセッサが他のプロセッサの主記憶の読み書きができる分散共有メモリ、同じ形態で主記憶の読み書きができない分散メモリがある。分散メモリに即した並列プログラミングライブラリとしては、PVM(Parallel Virtual Mahine)、および、MPI(Message Passing Interface)が有名である。また、共有メモリに即した並列プログラミングモデルとしては、OpenMP が主流になってきている。本研究ではこの OpenMP を用いて実験をする。

共有メモリを有するシステムのことを SMP（Symmetrical Multi Processor）と呼ぶ。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、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（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。そのため、近年ますます SMP 市場が増えており注目を集めている。SMP 構造は、近年普及しているマルチコアやメニーコアなどのプロセッサを搭載した PC に取り入れられているため、SMP クラスタの導入は一般の PC クラスタと同様に、容易である。この SMP クラスタを利用することにより、計算ノード内は共有メモリ型の並列処理、計算ノード間は分散メモリ型の並列処理をそれぞれ行う、ハイブリッド型の並列処理も可能である[1]。

本研究では、高性能計算研究室で研究が進められている SMP クラスタ上での並列処理に注目し、イラストロジックと呼ばれるパズルゲームの解を求めるプログラムを高速で解けるように並列化を行う。イラストロジックは、1988 年から雑誌上で連載され、2007 年には、任天堂が発売している携帯型ゲーム機からピクロス（ピクチャー・クロスワード）という名前でゲームソフトとして発売されている。今回の実験でイラストロジックを選んだのは、解を決定していくアルゴリズムに興味を持ったからである。イラストロジックは、方眼状のマスを使い、その方眼状のマスの外側にある、上と左の鍵と呼ばれる数字をヒントに、各マスを黒マスと白マス（×マス）に埋めていくパズルゲームである。出来上がったときに、フィールドに何かのイラストが浮かび上がる。

今回の実験では、10×10 から 1000×1000 の方眼状のマスを使用し、同じイラストが出力されるイラストロジックの問題の解を求め、その解が求まるまでの時間を計測する。より速く解を求めるために並列処理を行い速度向上を比較する。計算に使用したクラスタは、各計算ノードが SMP 構造を持つ Diplo（16 プロセッサ）クラスタである。

本論文の構成は次の通りである。２章では、イラストロジックの歴史や魅力、またルールとアルゴリズムについて述べ、3 章では OpenMP による並列化手法について記している。４章で、実験結果と考察についてを記している。

2. イラストロジックとその解法

2.1 イラストロジックとは

（1）イラストロジックの歴史

イラストロジックは西尾徹也といしだのん（石田伸子）の二者が独自に発案したパズルゲームである。同時期に別々に発表されたために一時期論争が起こったが、結局は殆ど同じ時期に創案したということが分かり、論争もおさまった。西尾は、ロジックパズルのマトリクスを利用して絵を描くことを考えた。当時の一般的なマトリクス状のロジックパズルにおいては、マトリクスには○か×を埋めていたが、この○と×を黒マスと白マスにすることを考え、ヒントの出し方を考えて現在のルールを完成させた。石田は 1987 年に、ビルの窓を利用して絵を描くという企画で入選した後、窓の絵=格子の上に絵を描くパズルを考えた。この石田の思考の過程の一部は彼女の著書である「ののぐらむ」で見ることができる。

このように二者が独自の過程を経て偶然にも同じ形式の問題に落ち着いた。後に西尾は問題を提供している『パズラー』誌で、石田はパズルの連載をしていた『社会新報』誌で問題を発表することになるが、偶然にも両誌の発行日は共に 1988 年 7 月 2 日であった。西尾は『パズラー』でこのパズルを発表した際に、このパズルを読者からも募集した。これにより高い人気を得、同誌から多くの作家が生まれることになった。一方、石田の作品は、イギリスのダルゲッティにより、ののぐらむと命名されて『サンデー・テレグラフ』紙で連載されることになった。これにより、ののぐらむはイギリスで人気を得ることになる。また、日本においては毎日新聞の日曜版でも連載されていた。任天堂ではピクロス（ピクチャー・クロスワード）の名を冠してパズルゲーム化した。1995 年に携帯用ゲーム機ソフト『マリオのピクロス』を発売し、以降シリーズ化されている[5]。

（2）イラストロジックの魅力

パズルゲームと言えばクロスワードパズルや数独パズルなど様々なものがあるが、それらに共通している魅力は、パズルゲーム特有のマスを埋めていく楽しさである。数あるパズルゲームの中でも、イラストロジックが 1 つの絵を作り出すパズルであるという点が、他のパズルゲームにはない魅力である。

2.2　ルールとアルゴリズム

（1）ルール

問題の例として 5×5 のイラストロジックを図１に示す。数字が書かれていない真ん中のマス群をフィールドとする。以下のルールにしたがって塗りつぶす。

- フィールドの上と左の数字は、その列、行で連続してぬりつぶすマスの数を表す。
- 複数の数字が並んでいる場合は、並んでいる順番どおりに、 書いてある数字の数だけ連続してマスをぬりつぶす。
- 塗らないことが確定したマスは×マスとする。
- 数字と数字の間は、必ず１マス以上の×マスで間をあける。

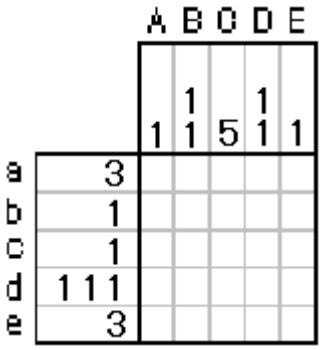


図 1：イラストロジックの例

このルールに基づいて、実際に図１のイラストロジックの例を解いていく。解法の例を図 2 と図 3 に示す。

Phase1：まず、鍵の数字が大きなマスに注目する。C のマスに注目すると、全て黒マスに確定することがわかる。

Phase2：b と c のマスに注目する。鍵の数と黒マスの数が一致しているため、残りのマスは全て×マスとなる。

[Phase1]　　[Phase2]

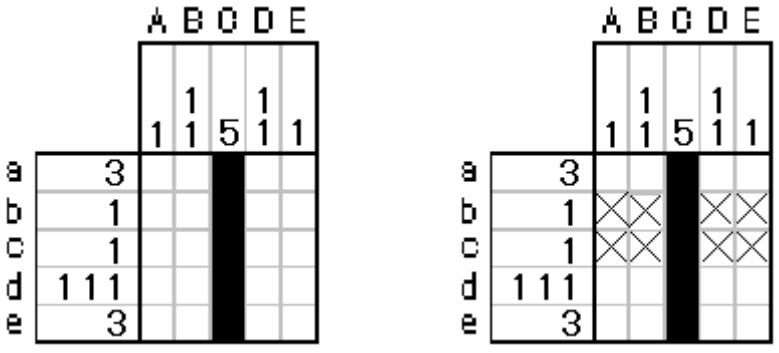


図 2：解の手順１

Phase3：d のマスに注目する。鍵の数字より、マスの埋め方が 1 通りしかないことがわかるので、d のマスが確定する。
Phase4：A と E のマスに注目する。鍵の数と黒マスの数が一致しているので、残りのマスは全て×マスに確定する。
Phase5：a と e のマスが確定し、イカリのイラストが浮かび上がる。

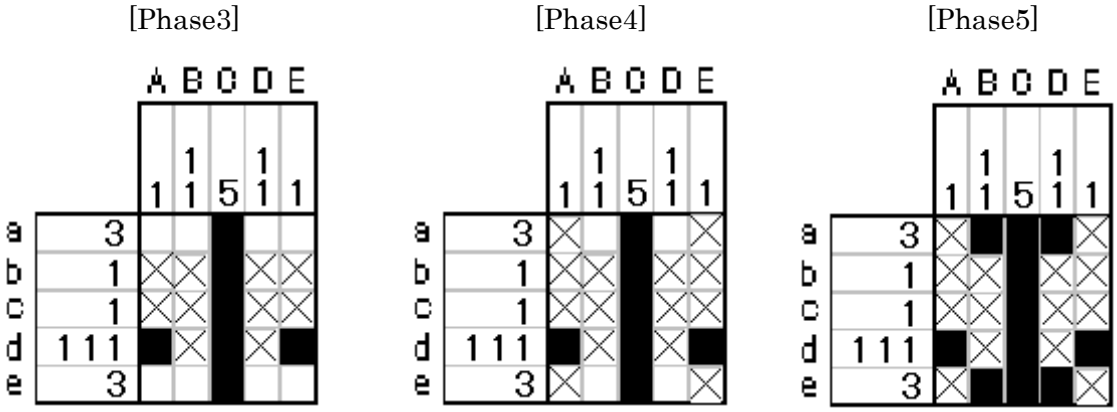


図 3：解の手順 2

（2）アルゴリズム

イラストロジックの問題を解く為のアルゴリズムの例を図 4 から図 18 に示す。また、その説明を以下の①から⑯に示す。なお、この例はフィールドの横の長さが 10 マスの場合である。

①重なりを調べて確定

例を図 4 に示す。鍵の数字がフィールドの長さの半分よりも大きい時、鍵の数字を右に寄せて塗られる黄色のマスと左に寄せて塗られる黄色のマスが重なったマスだけ黒マスに確定される。

図 4：アルゴリズムの説明①

②端から伸びて確定

例を図 5 に示す。端のマスが黒マスに確定している時は、1 つ目の鍵の数字である 3 の分だけ黒マスが右に伸びる。

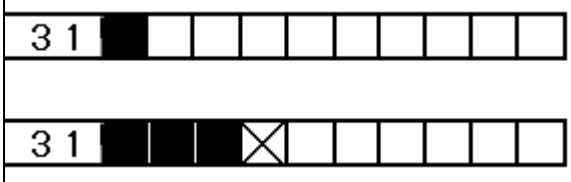


図 5：アルゴリズムの説明②

③×マスが全て確定しているときの黒マスの確定

例を図 6 に示す。(マスの長さ－鍵の数字の合計）＝塗られている×マスの合計が成立する時、残りのマスが黒マスに確定する。例では、10－（2＋3）＝5 が成立し、残りのマスが黒マスに確定する。

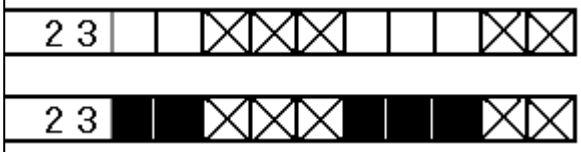


図 6：アルゴリズムの説明③

④鍵と黒マスの数が一致した時の×マスの確定

例を図 7 に示す。鍵の数字の合計が 2 であり、塗られている黒マスの数と等しいので残りのマスは×マスに確定する。

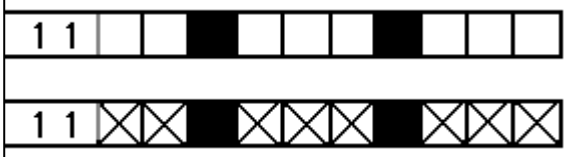


図 7：アルゴリズムの説明④

⑤黒マスと黒マスに挟まれた箇所の黒マスの確定

例を図 8 に示す。鍵が 1 つの時で、黒マスが離れて塗られている時、その間のマスが黒マスに確定する。

図 8：アルゴリズムの説明⑤

⑥黒マスから伸びて確定

例を図 9 に示す。黒マスと端の間が全て×マスの時、そこから初めの鍵の 3 マス分だけ黒マスを塗ることが出来る。

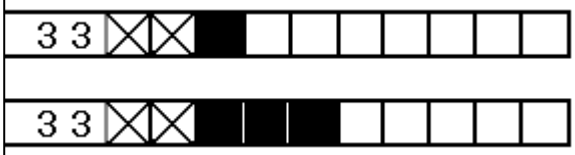


図 9：アルゴリズムの説明⑥

⑦×マスと端に囲まれた箇所での黒マスの確定

例を図 10 に示す。×マスと端の間に黒マスが存在する時、その位置によって黒マスが確定する。

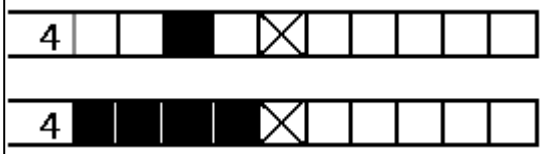


図 10：アルゴリズムの説明⑦

⑧重なりを調べて確定（応用）

例を図 11 に示す。鍵が２つ以上の時、鍵の数字を左に寄せて塗られる黄色のマスと右に寄せて塗られる黄色のマスの重なった箇所を黒く塗る。ただし、その重なったマスが違う鍵の数字同士の場合は黒マスに決定しない。例のように、青のマスは確定できない。

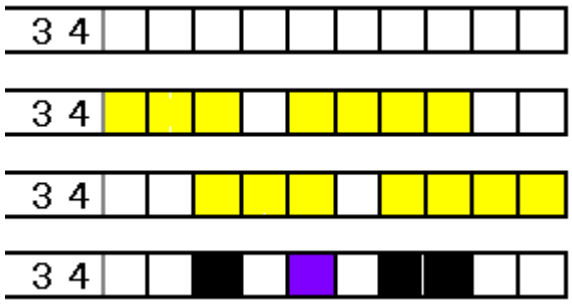


図 11：アルゴリズムの説明⑧

⑨鍵が２つ以上の時、黒マスと鍵の１つが一致した時の×マスの確定

例を図 12 に示す。鍵の数字の中で一番大きい数が確定している時、その両端が×マスに確定する。黒マスの数が一番大きい鍵の数字ではない時、そのマスは一番大きい鍵の数字の一部である可能性があるので確定できない。

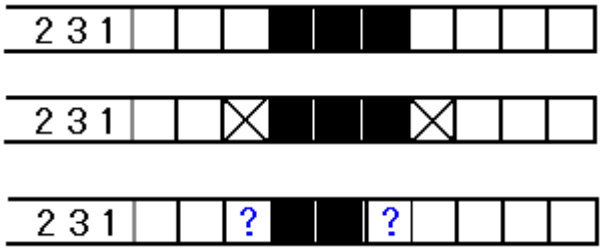


図 12：アルゴリズムの説明⑨

⑩黒マスから鍵の数の分だけ届かないところは×マスに確定

例を図 13 に示す。黒マスから鍵の数だけ伸ばし、届かない箇所は×マスに確定する。例のように、青マスが届かない箇所が×マスとなる。

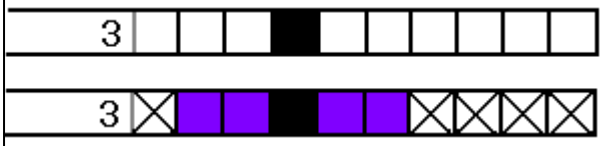


図 13：アルゴリズムの説明⑩

⑪入る余地がないところは×マスに確定

例を図 14 に示す。×マスと端の間や、×マスと×マスに挟まれた間の数が鍵の数字よりも小さい時、そのマスは×マスに確定する。

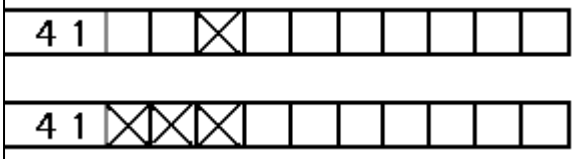


図 14：アルゴリズムの説明⑪

⑫鍵の合計からその列、行が全て確定

例を図 15 に示す。鍵の数字の合計＋鍵の総数－1 = 10 の時、その全てのマスが確定する。

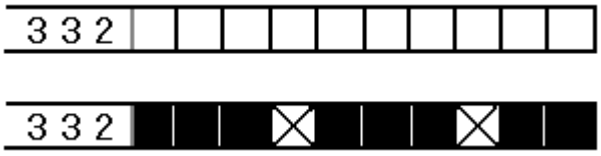


図 15：アルゴリズムの説明⑫

⑬端の隣が黒マスの場合

例を図 16 に示す。端の隣が黒マスに確定している時、1 つ目の鍵－1 の数、つまり 4 マスだけ黒マスが伸びる。

図 16：アルゴリズムの説明⑬

⑭×マスと×マスの間

例を図 17 に示す。端の×マスが確定している時、それを除いた残りのマスで考える。例の場合だと、端の×マスを除いた残りのマスの中で鍵の重なりを考えて、マスを確定している。

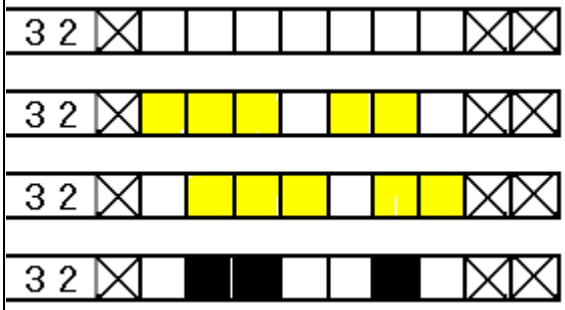


図 17：アルゴリズムの説明⑭

⑮鍵の幾つかが決まっている時、それを省いて考える

例を図 18 に示す。確定している黒マスに対応している鍵を除いて、残りのマスのみで考える。例の場合だと、鍵の数字と一致しているマスを除き、残りのマスの中で重なりを考えている。

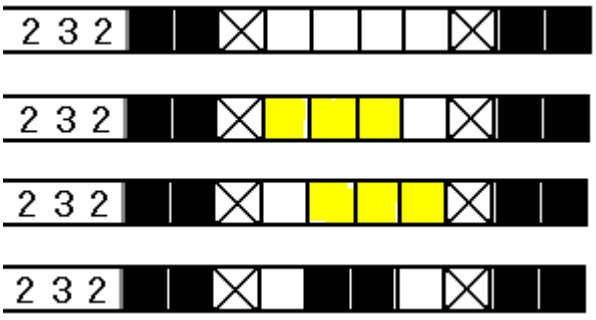


図 18：アルゴリズムの説明⑮

⑯チェック

全ての行と列に、塗られている黒マスと鍵の数字が一致しているかの探索をかける。等しければ、その解は真である。

（3）解けない場合

上記の①から⑮のアルゴリズムだけではイラストロジックの解法プログラムとして不完全である。10×10 の問題で、鍵の数が 1 個から 3 個の基本的な問題は解けるが、鍵の個数を増やしたり、フィールドの大きさを拡大することで、解けない問題も増えてしまう傾向にある。加えて背理法を必要とする問題は解けない。

3.　イラストロジックの並列化手法

3.1　1 画素からの確定探索

解の探索方法としては、行、列ごとにマスを調べていく方法もあるが、より効率の良い速度向上を得るために、粒度が大きいプログラムを組む必要がある。そこで 1 マスごとに解を決定していく方法をとる。探索方法の例を図 19 に示す。x と y はマスの座標を表し、x が縦、y が横の座標である。

main 関数のフローチャートを図 20 に示し、以下に簡単な流れを述べる。length はフィールドの縦の長さを表し、side は横の長さを表している。

まずカウント用の変数を用意し、あらかじめループする回数を決める。次に、問題となる鍵の数字を入力ファイルより参照し、x と y をループさせる。次にユーザ関数を参照し、1 マスごとの解を判断するモジュールを呼び出す。それぞれのモジュールにおいてそのマスの、black か White の値が決定される。black が 1 の時、黒マスが入り、white が 1 の時、白マスが入れられ、どちらの値も 0 の場合は、未確定となる。決められた回数ループを繰り返し、最後にイラストが出力される。

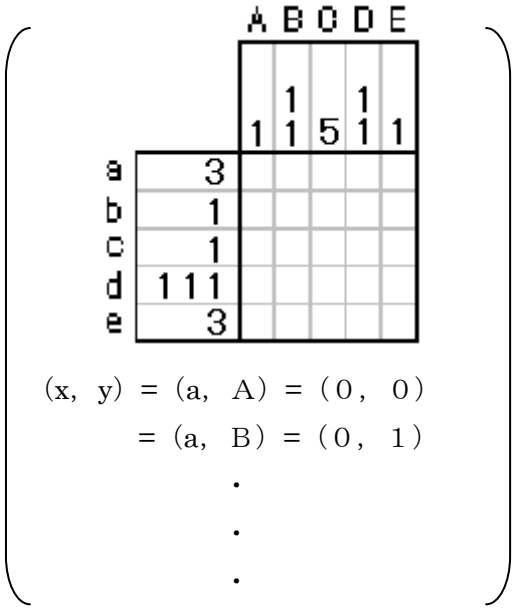


図 19：座標の例

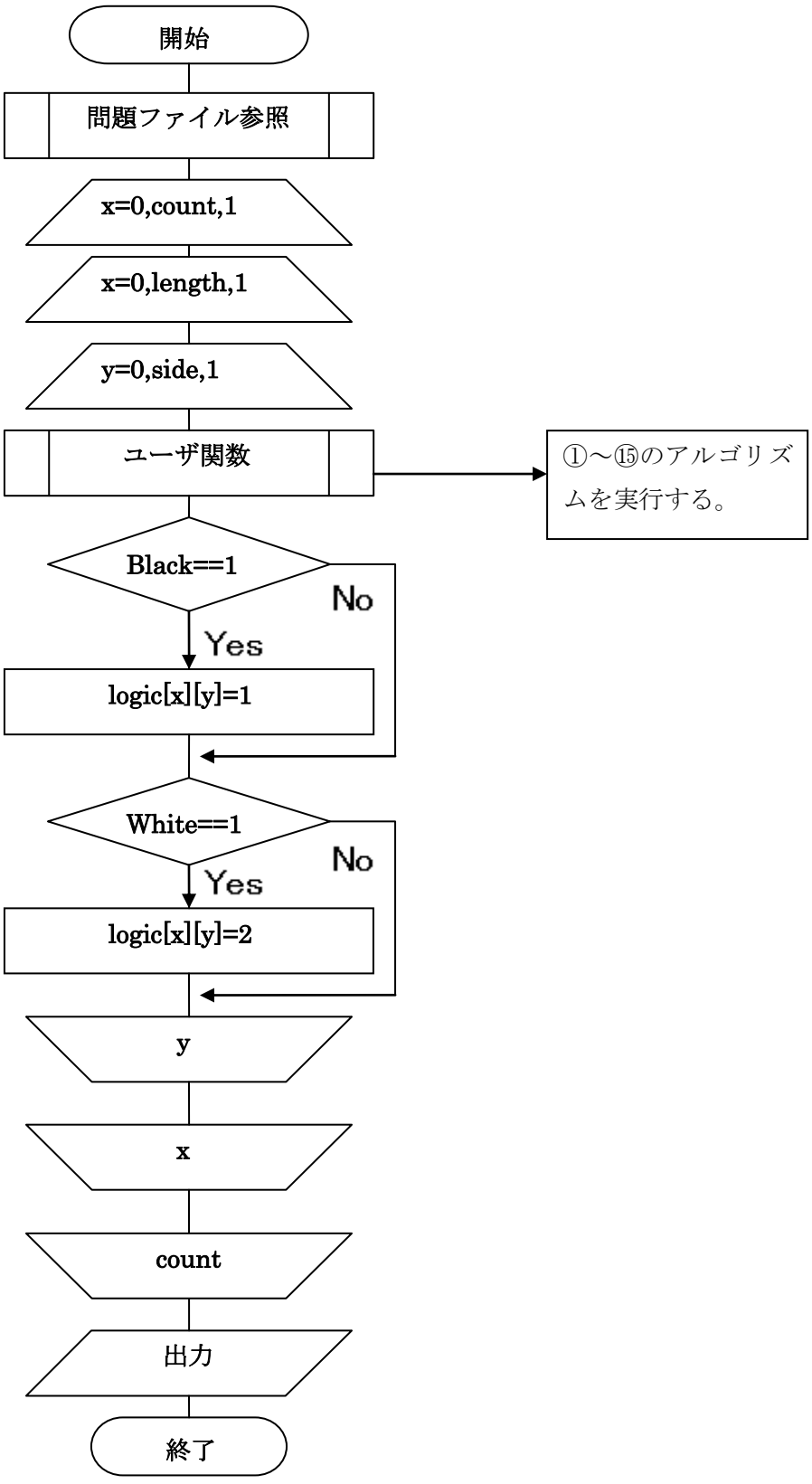


図 20：main 関数のフローチャート

3.2　OpenMP による並列化

（1）OpenMP

OpenMP は、共有メモリ並列処理用の標準化されたプログラミングモデルである。既存の言語の逐次プログラムに指示行を加えることにより、共有メモリシステム上で段階的に並列化することができる。また、すでに広く普及している分散メモリシステム上の MPI によるプログラムと比較して、OpenMP では逐次計算プログラムから並列計算プログラムへの書き換えの手間が少なくてすむというのも１つの特徴である。具体的には、逐次プログラムに#pragma omp で始まる、プラグマ指示文と呼ばれる文を挿入するだけで簡単に並列化が可能である。現在、OpenMP はスレッド並列演算を行うための業界標準的な仕様となっており、この仕様に則したプログラムを書いておくと、異なる計算機上でも OpenMP 対応のコンパイラを通すことで、直ちに並列演算を行うことができる[4]。

４スレッドで実行したときのブロック分割とサイクリック分割の例を図 21 の（a）と（b）に示す。以下にその説明を述べる。

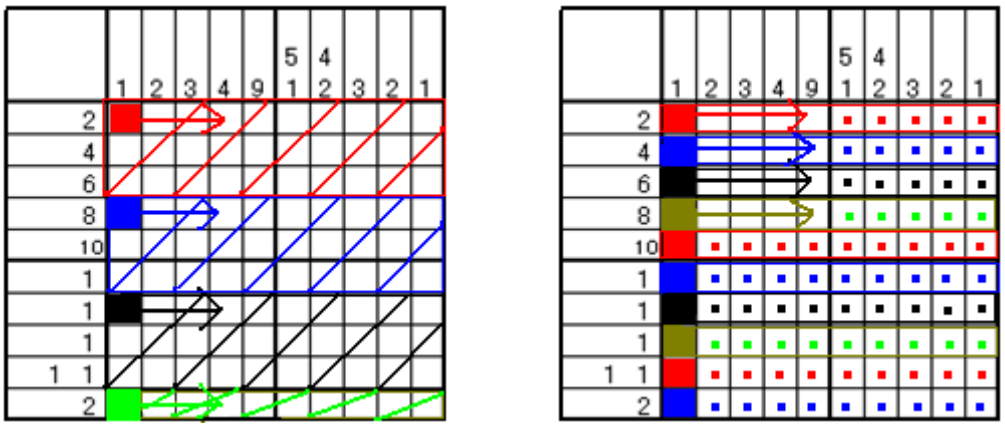
（2）ブロック分割

ブロック分割とは、各スレッドに対して、ある一部分の領域を担当させる方式である。スレッドの数によって自動的にループ文を分割する。シングルスレッドでの動作を想定したプログラムの、マルチスレッド化したいループ文の前に OpenMP の for 構文を挿入するだけで、高速に処理される並列プログラムになる。

（3）サイクリック分割

サイクリック分割とは、各スレッドに対して要素を一つずつ順番に割り当てる方式である。ブロック分割よりも細かく分割することで、それぞれのスレッドに均等に負荷を振り分け、高速化を図る。

赤がスレッド番号１、青がスレッド番号２、黒がスレッド番号３、緑がスレッド番号４に割り当てられている。



（a）ブロック分割　　（b）サイクリック分割

図 21：ブロック分割とサイクリック分割

4. SMP クラスタ上での実験と考察

4.1 実験環境

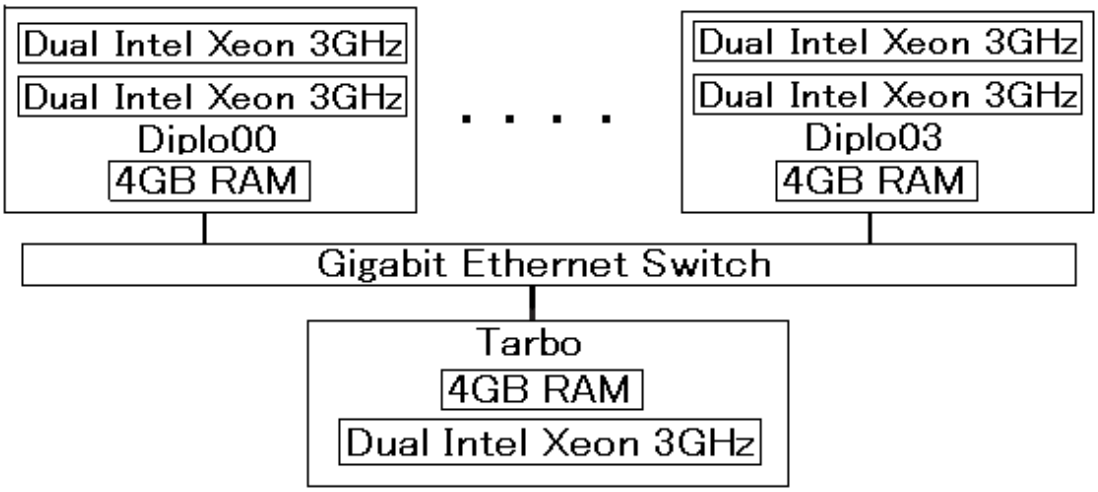


図 22：Diplo クラスタの構成図

本研究室で使用されている Diplo クラスタにより実験を行う。Diplo クラスタの構成図を図 22 に示す。Diplo の計算ノードは Diplo00 から Diplo03 までの計 4 台で、それぞれのノードは Gigabit Ethernet Switch を介して、ホストサーバである Tarbo と繋がっている。Diplo は Intel compiler により、並列プログラミング環境がインストールされている[7]。

なお、実験は Diplo00 内の１ノード４プロセッサで行った。10×10、100×100、1000×1000 の同じイラスト（図 23）が出力される問題を用意し、それぞれを１台、２台、４台で実験した時の実行時間と速度向上比を比較した。また、ブロック分割とサイクリック分割の比較も行った。

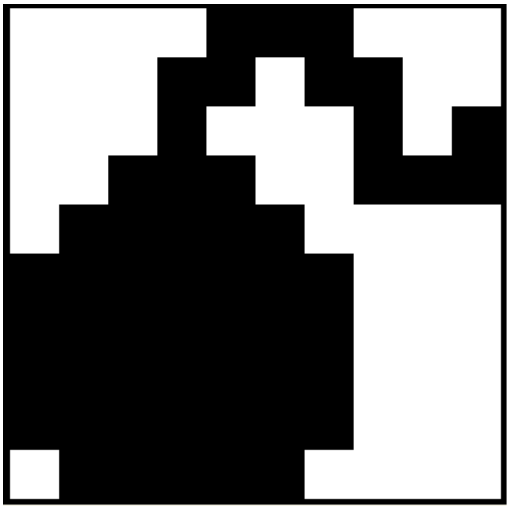


図 23：問題

4.2　実験結果

ブロック分割とサイクリック分割の実験結果を以下に示す。ブロック分割の実行時間と速度向上比を表 1 と図 23 に示し、サイクリック分割の実行時間と速度向上比を表 2 と図 24 に示す。ブロック分割とサイクリック分割の比較は図 25 に示す。縦×横の縦と横は、イラストロジックのフィールドの縦と横の長さである。

表 1：ブロック分割による実行時間（秒）

スレッド数 / 縦×横	1	2	4
10×10	0.0016	0.002	0.0046
100×100	0.48	0.39	0.19
1000×1000	1239.4	660.6	348.07

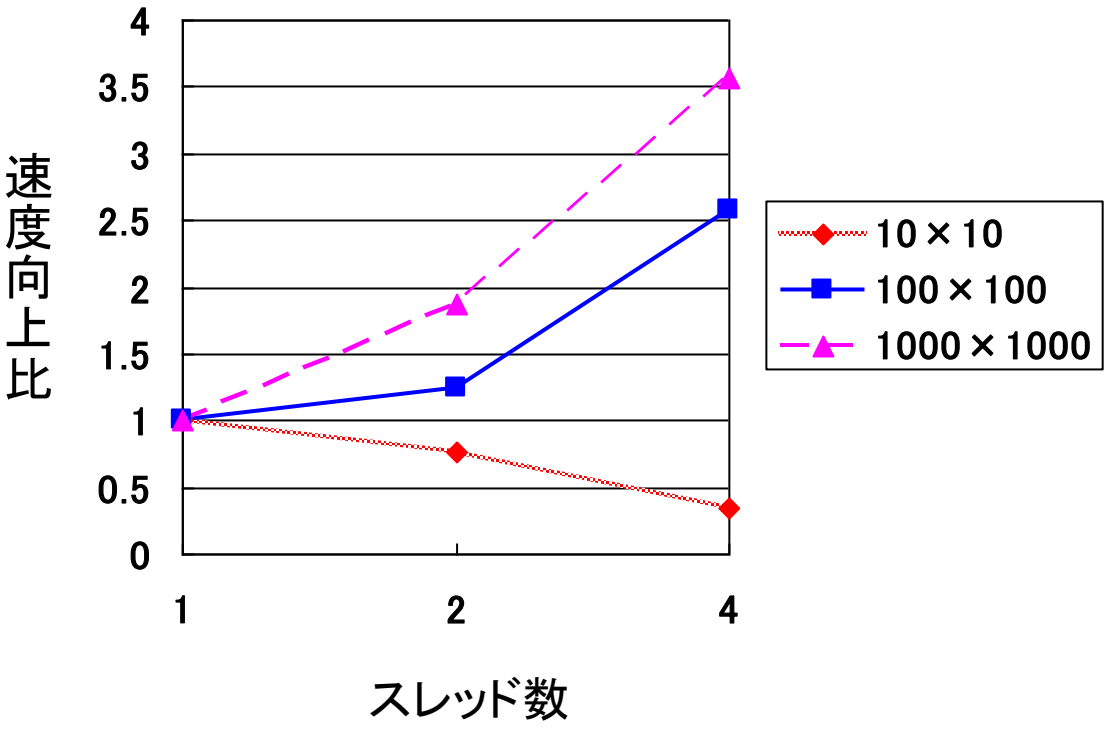


図 24：ブロック分割による速度向上比

表 2：サイクリック分割による実行時間（秒）

スレッド数 / 縦×横	1	2	4
10×10	0.0016	0.003	0.0048
100×100	0.48	0.28	0.15
1000×1000	1240.5	630.6	322.5

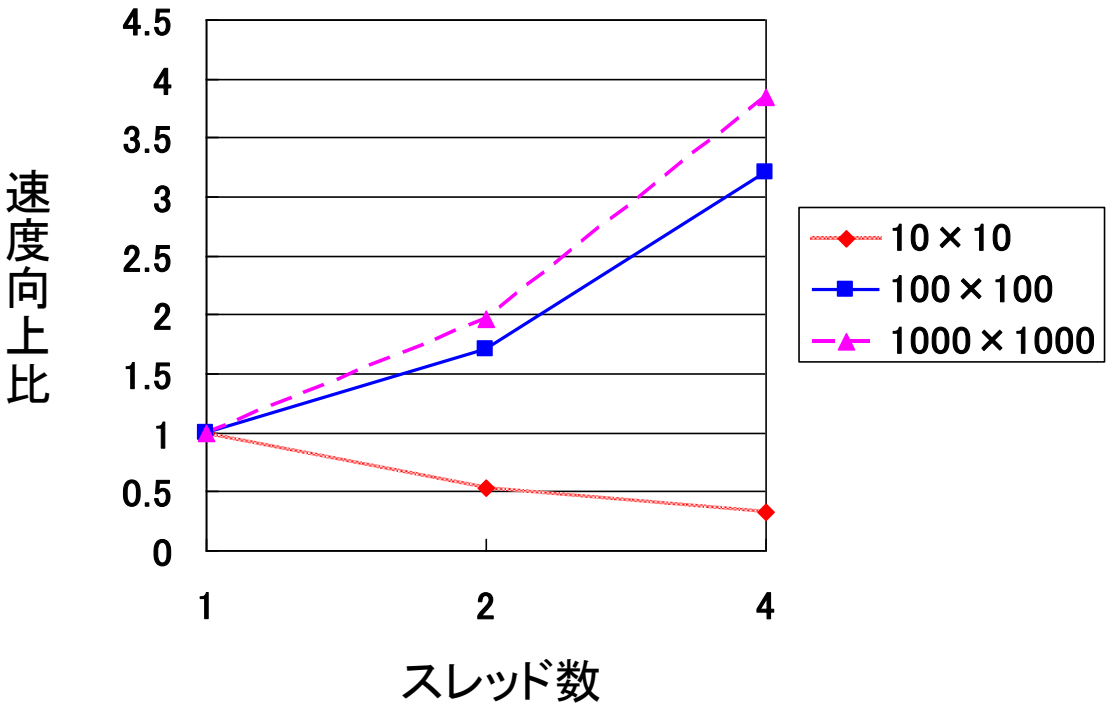


図 25：サイクリック分割による速度向上比

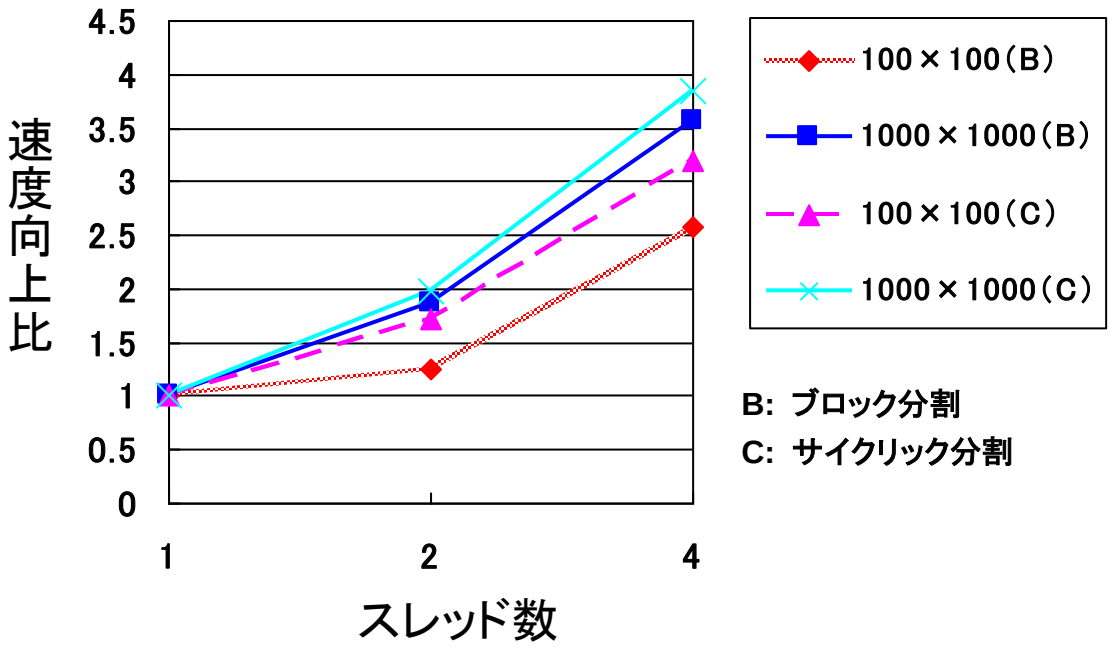


図 26：ブロック分割とサイクリック分割の比較

ブロック分割による実行時間は、1000×1000 の問題において、1 スレッド実行と比べて 2 スレッド実行で約 1.87 倍、4 スレッド実行で約 3.56 倍の速度向上が得られた。また、サイクリック分割による実行時間は 1 スレッド実行と比べて 2 スレッド実行で約 1.97 倍、4 スレッド実行で約 3.85 倍の速度向上であった。ブロック分割とサイクリック分割の実行時間を比較すると、ブロック分割に比べてサイクリック分割の速度は向上した。

4.3　考察

ブロック分割したときのスレッドの割り当て方を図 27 に示す。イラストロジックの解答プログラムを OpenMP により並列化することで、ほぼプロセッサ数に見合う速度向上が得られた。ブロック分割とサイクリック分割の実行時間を比較してみると、4 スレッド実行の 1000×1000 の問題で、サイクリック分割の方が 1.08 倍の速度向上だった。これは、ブロック分割よりもサイクリック分割の方が負荷が均等に分かれたということを示している。

イラストロジックのマスが確定する順序は、第一に、鍵の数字によるものである。次に、鍵の数字のみで確定した黒マスや×マスと、鍵の数字の関係によりマスの確定が拡がっていく。そして、初期段階でのマスの確定では、黒マスは×マスと黒マスの両方から派生して確定する可能性があるが、×マスは、アルゴリズムの性質により×マスから派生して確定することは少ないと推測される。さらに、イラストロジックのルールにより、鍵の数字はその行、列で連続して塗られる数を表しており、1 つの黒マスが確定すると、そこから芋づる式に周りのマスが確定していく可能性が高い。

これらのことから、各スレッドは初めの計算で、鍵の数字から導けるマスの確定を行い、そこから生じる黒マスの周りの計算に集中する。図 27 の例だと、C のスレッドは初めから計算量が多いと考えられるが、逆に A のスレッドは計算量が乏しそうである。このように各スレッドで、実行時間に違いが生じ、大きく分割をするブロック分割ではサイクリック分割ほどの速度向上が得られなかったのではないかと考えられる。

また、10×10 の問題のみ、速度向上が得られていないのは、プロセッサ 1 台で十分処理できるほど計算量が少ない処理を分割し、並列化したことで計算量を増やし、オーバーヘッドが発生したからだと考えられる。

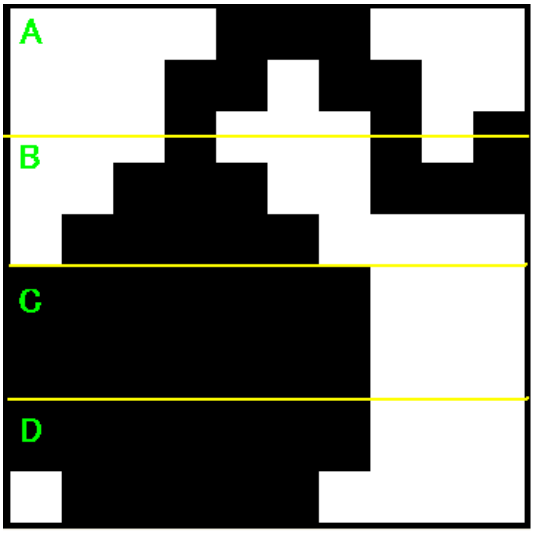


図 27：ブロック分割時のスレッドの割り当て

5. おわりに

本研究では、SMP クラスタ上の１ノード４プロセッサの中で OpenMP によってイラストロジックと呼ばれるパズルゲームの解を求めるプログラムを、高速で解けるように並列化し、実行した。その結果、1000×1000 のパズルの問題において、４スレッド実行により、ブロック分割で約 3.5 倍、サイクリック分割で約 3.85 倍の並列化効果を得られた。

今後の課題としては、問題に対する解の精度を上げ、より多くのパターンの実験をすること。そして、より速く解が得られるようにプログラミングを考える必要がある。また、４ノード１６プロセッサでのハイブリット並列化を行うことにより速度向上が得られるかどうかを確認する必要性がある。OpenMP のコンパイラの性能が良くなっていけば、SMP クラスタ上での OpenMP による並列プログラミングは今後ますます実用化されると思われる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、本研究に関して様々な相談に乗って頂き、貴重な助言を頂きました松崎裕樹氏、志水建太氏、和田智行氏、新井惣一朗氏をはじめ高性能計算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，安村通晃，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] OpenMP 入門：
http://www.na.cse.nagoya-u.ac.jp/~reiji/lect/hpc02/OpenMPintro.html

[3] 松岡毅:OpenMP による数独パズルの並列化，立命館大学理工学部情報学科卒業論文，2007.

[4] 牛島省：OpenMP による並列プログラミングと数値計算法，丸善株式会社，2006.

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

[6] ののぐらむ（お絵かきロジック，イラストロジック）解法教室：
http://www.pro.or.jp/~fuji/java/puzzle/nonogram/knowhow.html

[7] 田中秀宗 : PC クラスタ上での文字列最適周期アルゴリズムの並列化， 立命館大学理工学部情報学科卒業論文，2007.