魔方陣全解出力アルゴリズムの改良と超並列計算環境でのプログラム実行

茨城県立並木中等教育学校 4 年
杉崎 行優

A　研究目的

これまでに学校のコンピュータ室のコンピュータを用いて並列計算システムの構築をした。そのシステム上で、4 次魔方陣の全解を総当たりで探索する並列プログラムを実行したところ、一晩かけても実行が終了しなかった。その後、筑波大学のスーパーコンピュータ T2K-Tsukuba 学際利用プログラムに申し込み、そのプログラムを T2K-Tsukuba の 512CPU コア上で走らせた。しかし、最大実行可能時間である 24 時間をかけても実行は終了しなかった。

そのことから、アルゴリズムを改良し魔方陣の全解を一般のコンピュータ 1 台や T2K-Tsukuba 上で探索することを目的とした。一般のコンピュータ 1 台上で探索することを目的とした。そのために、枝刈り法の改良やより効率のいいアルゴリズムの開発をした。

B　研究方法

この研究は 2012 年 5 月から 2013 年 9 月にかけて行った。今回は魔方陣の全解を探索し出力するプログラムを開発した。総当たりを基本に考え、その上で探索するパターンの候補を減らすための枝刈り法を改良した。

また、並列計算ライブラリには MPI(Message passing interface) を用いた。魔方陣の総数はどれほど多くなるか分からなかったので、オーバーフローしないよう、カウンターには GMP(GNU multiple precision arithmetic library) で提供されている `mpz_t` 型の変数を使用した。

B.1　用語・定数の定義

X を自然数とする。縦 X、横 X の計 X^2 マスに整数が入れられているものを「X 次の陣」と呼ぶ。また、陣の内、1 から X^2 の数が 1 つずつ用いられており、縦の X 列、横の X 列、斜めの 2 列それぞれの合計が全て等しく、L となるものを「X 次の魔方陣」と呼ぶ。ただし L は式 1 によって定義される。

$$L = \frac{1}{X}\sum_{i=1}^{X^2} i = \frac{1}{2}X(X^2+1) \tag{1}$$

B.2　アルゴリズムの改良

この方法では以下の手順を行った。

1. 効率のよい枝刈り法を考案する。

2. 仮計算なども行いながら、そのアルゴリズムの大まかな計算量を、正否判定する陣の数などから求める。

3. そのアルゴリズムをプログラム化し、一般のコンピュータ 1 台上で実行する。

4. 実行のプロファイリング等を用いて、冗長的な処理等を分析し、次のアルゴリズム作成に生かす。

5. 最も効率のよいプログラムよかったプログラムを並列化し、T2K-Tsukuba 上で実行する。

6. 実行時の処理性能低下などを分析し、原因を分析する。

B.3 T2K-Tsukuba の利用

T2K-Tsukuba とは、筑波大学に置かれているスーパーコンピュータの 1 つである。

今回は、この研究のために学際開拓プログラムに申し込み、T2K-Tsukuba を利用させていただいた。今回の申請では、一度に利用できる最大ノード数は 32 台である。また、最大連続実行可能時間は 24 時間となっており、実行時間がこの時間を超えた場合、そのジョブは強制終了される。

並列方式にはマスタ・ワーカー型並列を用いた。実行は全 $16 \times 32 = 512$ プロセスの内、1 プロセスをマスタとしその他をワーカーとした。

T2K-Tsukuba でのプログラム実行の手順は以下の様になっている。

1. ログインノードに SSH(Secure shell) を用いてログインする。

2. プログラムを `mpicc` コマンドでコンパイルし、実行ファイルを作成する。

3. 以下の手順を行うシェルスクリプトを書き、`qsub2` コマンドにてジョブをサブミットする。

   (a) プログラム実行に使用するノードの台数と最大実行時間を指定する。

   (b) 使用するノードをリストアップするスクリプトを実行する。

   (c) `mpirun_rsh` コマンドでプログラム (手順 2 で作成した実行ファイル) を実行する。

4. その後、ジョブは逐次自動実行される。

C　得られた結果

C.1　アルゴリズムの開発・改良と一般のコンピュータ 1 台での実行

C.1.1　利用した一般のコンピュータの性能

今回計算に用いた一般のコンピュータの性能を表 1 に示す。

項目	仕様
CPU	Intel Pentium 4　3.20GHz
CPU コア数	2 cores
CPU ピーク性能	6.40GFLOPS
CPU メモリ	3GB
OS	Linux 3.2.46
コンパイラ	GCC 4.7.2

表 1: 一般のコンピュータの性能概要

なお、実行には 2CPU コアのうち 1CPU コアのみを用いた。

C.1.2　第一次アルゴリズム

第一次アルゴリズムは完全総当たりの方法で、具体的には、1 つ 1 つのマスに 1 から n^2 の数を順に入れていき、全てのマスを埋めたら正否を判定するというものだ。最終的に判定される陣数は $(X^2)^{(X^2)}$ 個である。

このアルゴリズムをプログラム化し、一般のコンピュータで実行したところ、3 次の場合、実行時間が約 20 秒となった。このプログラムで 4 次の場合の計算をすると、式 2 より、実行に約 1.4×10^{12} 秒、つまり約 4.4×10^4 年かかることになる。よって、4 次以上の場合は現実的な時間では実行終了しないので、実際に実行はしなかった。

$$20 \times \frac{(4^2)^{(4^2)}}{(3^2)^{(3^2)}} \approx 1.4 \times 10^{12} \tag{2}$$

また、このアルゴリズムでは 1 から X^2 の全ての候補をそれぞれのマスに代入しているが、この場合、最終的に判定される陣に同じ数字が含まれることもある。これは魔方陣の成立条件に反しており明らかに無駄な処理なので、改善すべきである。

C.1.3　第二次アルゴリズム

第二次アルゴリズムは第一次アルゴリズムで挙げられた「陣の中に同じ数字が代入される場合がある」という点を解決するために、既に代入されている数字を陣に代入しないようにした。最終的に判定される陣数は $X^2!$ 個である。

このアルゴリズムをプログラム化し、一般のコンピュータで実行したところ、3 次の場合、実行時間が約 0.17 秒となった。このプログラムで 4 次の場合の計算をすると、式 3 より、実行に約 9.8×10^6 秒、つまり約 113 日かかることになる。よって、4 次以上の場合は現実的な時間では実行終了しないので、実際に実行はしなかった。

$$0.17 \times \frac{4^2!}{3^2!} \approx 9.8 \times 10^6 \tag{3}$$

C.1.4　第三次アルゴリズム

第三次アルゴリズムでは動的計画法を用いた。具体的には、あらかじめ合計が L である要素 X の数列 (正和列と呼ぶ) を作成しておき、それらの中から X 個選び、それぞれを陣の横列として代入するというものである。正和列は順列で表され、各 X における正和列の個数 E を表 2 に示す。これにより、最終的に判定される陣数は ${}_E\mathrm{P}_X$ 個となる。

X	E
3	48
4	2,064
5	167,280
6	23,136,480

表 2: 各 X における正和列の個数 E

このアルゴリズムをプログラム化し、一般のコンピュータで実行したところ、3 次の場合、実行時間が約 0.05 秒となった。このプログラムで 4 次の場合の計算をすると、式 4 より実行に 8.7×10^6 秒、つまり約 101 日かかることになる。よって、このプログラムも 4 次以上の場合は現実的な時間で実行終了しないので、実際に実行はしなかった。

$$0.05 \times \frac{{}_{2064}\mathrm{P}_4}{{}_{48}\mathrm{P}_3} \approx 8.7 \times 10^6 \tag{4}$$

1	・	3
・	2	・
・	・	・

図 1: $X = 3$ の場合の数字を埋める手順

1	7	・	4
8	2	5	・
・	6	3	・
・	・	・	・

図 2: $X = 4$ の場合の数字を埋める手順

1	8	9	・	5
10	2	11	6	・
12	13	3	14	・
・	7	・	4	・
・	・	・	・	・

図 3: $X = 5$ の場合の数字を埋める手順

1	11	12	13	・	6
14	2	15	16	7	・
17	18	3	8	19	・
20	21	9	4	22	・
・	10	23	・	5	・
・	・	・	・	・	・

図 4: $X = 6$ の場合の数字を埋める手順

C.1.5 第四次アルゴリズム

第四次アルゴリズムは、第一次アルゴリズムのように陣に数字を埋めていく際、列中の $X - 1$ 個の数が埋まれば、残りの 1 マスは L からそれらの数の合計を引くことにより求められることを利用した。

さらに今回は、数字を埋める順番を工夫した。まず初めに斜めのマスを埋め、次にその他のマスを埋めていくというものである。$X = 3$ から $X = 6$ までにおけるその手順を図 1, 2, 3, 4 に示す。数字は総当りするマスの順番、点 (・) は引き算により求められるマスを表している。斜めの列上のマスは縦と横だけでなく斜めの列にも属しているので、それらのマスを優先的に埋めていくことで引き算により求められるマスが増える。

各 X における総当たりするマスの個数 A を表 3 に示す。最終的に判定される陣数は高々 ${}_{X^2}\mathrm{P}_A$ 個である。

X	A	${}_{X^2}\mathrm{P}_A$
3	3	5.0×10^2
4	8	5.2×10^8
5	14	3.9×10^{17}
6	23	6.0×10^{31}

表 3: 各 X における総当たりするマスの個数 A

このアルゴリズムをプログラム化し、一般のコンピュータで実行したところ、

実行時間は 3 次の場合約 0.05 秒、4 次の場合約 0.12 秒、5 次の場合約 217 時間となった。

このプログラムは、今までのプログラムでは不可能だった 4 次や 5 次の魔方陣の全解を探索したという面で優秀である。

このプログラムで 6 次の場合の計算をすると、5 次の実行時間をベースに考えると式 5 より実行に約 1.2×10^{20} 秒、つまり約 3.8×10^{12} 年かかることになる。よって、このプログラムは 6 次以上の場合、現実的な時間で実行終了しないので、実際に実行はしなかった。

$$(217 \times 60 \times 60) \times \frac{{}_{36}\mathrm{P}_{23}}{{}_{25}\mathrm{P}_{14}} \approx 1.2 \times 10^{20} \tag{5}$$

また、このプログラムでは、数字が既に代入されているか否かを判断する際にチェックリストのようなものを使用していた。しかし、実行の際には「その数字が既に代入されているか」ではなく「次にどの数字を代入すればよいか」という情報だけを知ればよい。よって、それに適合している双方向リストをチェックリストの代わりに使用すれば実行時間がさらに短くなると考えられる。

C.2 T2K-Tsukuba 上での実行

C.2.1 T2K-Tsukuba の性能

T2K-Tsukuba の性能を表 4 に示す。

区分	項目	仕様
計算ノード	CPU	quad-core Opteron 2.3GHz × 4 基
	CPU コア数	16 cores
	CPU メモリ	32GB
	OS	Linux 2.6.18
	コンパイラ	GCC 4.1.2
	MPI ソフトウェア	MVAPICH2 1.4.1
並列ネットワーク	ネットワーク	Infiniband 4×DDR
	1 本当たりのピーク性能 (片方向)	2GB/s
	トポロジ	Fat-Tree
全体システム構成	総ノード数	648 台
	CPU ピーク性能	95.4TFLOPS
	ファイルシステム	800TB(Lustre, RAID-6)

表 4: T2K-Tsukuba の性能概要

C.2.2 実行したプログラム

今回は第四次アルゴリズムのプログラムを 5 次の場合で並列実行した。プログラム内並列計算の手順を以下に示す。

1. マスタが N 番目 ($N \in \mathbb{N},\ N \leq A$) のマスまで、数字の総当たりをする。
2. そのうちの 1 つのパターンを 1 つの暇なワーカーに配る。
3. ワーカーがそれを受け取る。
4. そのワーカーが $N+1$ 番目からの数字を総当たりする。マスタは手順 2 から繰り返す。
5. ワーカーの 1 つのパターンに対する処理が終了したら結果をファイルに出力し、マスタからのパターン配布を待ち、手順 3 から繰り返す。
6. マスタの N 番目までの総当たり処理が終了したらマスタが全ワーカーに終了を伝え、実行を終了する。

第四次アルゴリズムの枝刈り法により、ワーカーがマスタから受け取る陣によってワーカーの処理時間が異なる。また、マスタがワーカーに配る陣数は、N の値が小さいほど少なく、N の値が大きいほど多い。つまり、N の値が小さいほど問題を荒く分けているといえ、N の値が大きいほど問題を細かく分けているといえる。

C.2.3 プログラムの実行

プログラムは、N を 1 から 14 まで変化させ並列実行した。各 N における実行時間のグラフを図 5 に示す。24 時間以内に実行が終了したのは N が 3 から 8 のときであった。実行時間は $N = 3$ で最も長く、$N = 6$ で最も短く、$N = 8$ で再び長くなっている。

図 6 に $N = 3,\ 6,\ 8$ における各ワーカーの全体実行時間 (通信時間を含む主要処理時間) を示す。$N = 6,\ 8$ では各ワーカーの全体実行時間がほぼすべて同じであるが、$N = 3$ ではばらつきが出ている。

図 7 に $N = 3,\ 6,\ 8$ における各ワーカーの総通信時間 (通信のみ) を示す。$N = 3,\ 6$ では各ワーカーの総通信時間がほぼ 0 だが、$N = 8$ では 50 分ほどとなっている。

D 考察

D.1 第四次アルゴリズムにおける判定される陣から導かれる予測実行時間の比と実際の実行時間の比の相違

ここで表 5 に、3 次の場合に対する 4 次の場合、4 次の場合に対する 5 次の場合、3 次の場合に対する 5 次の場合それぞれの予測実行時間と実際の実行時間の比を

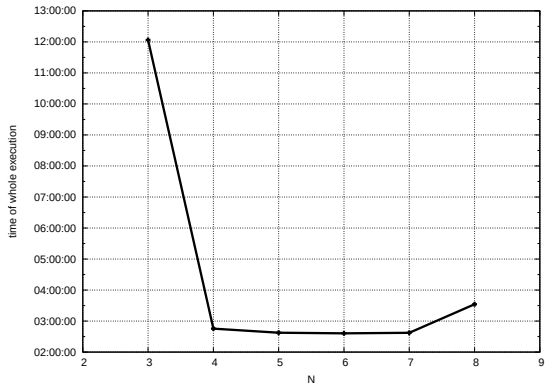

図 5: 各 N における実行時間

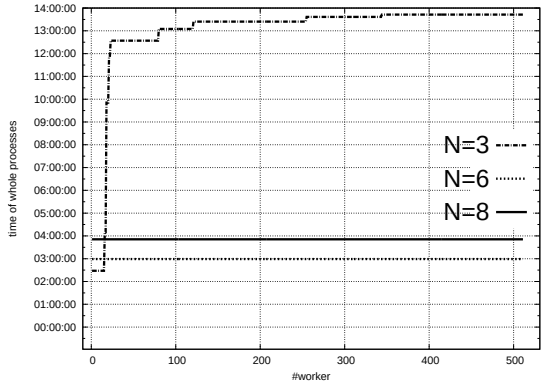

図 6: $N = 3,\ 6,\ 8$ における各ワーカーの全体実行時間

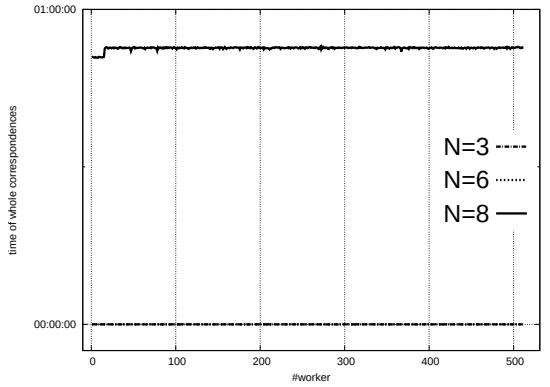

図 7: $N = 3,\ 6,\ 8$ における各ワーカーの総通信時間

示す。

表より、それぞれの予測実行時間の比と実際の実行時間の比は大きく異なっていることが分かる。これは、3 次と 4 次の場合の実行時間がとても短く、実行時間の計測で誤差が生じてしまったことが原因だと考えられる。

	予測実行時間の比	実際の実行時間の比
3 次の場合に対する 4 次の場合	1.0×10^6	2.4×10^0
4 次の場合に対する 5 次の場合	7.5×10^8	6.5×10^6
3 次の場合に対する 5 次の場合	7.7×10^{14}	1.6×10^7

表 5: 理想実行時間の比と実際の実行時間の比

D.2 第四次アルゴリズムのプログラムを T2K-Tsukuba 上で実行した際の、N の値による実行速度の違い

$N = 3$ では、各ノードの全体実行時間にばらつきが出た。これは、$N = 3$ では問題が細かく分けられていなく、マスタから受け取る陣によってワーカーの処理時間が異なることにより、結果として各ワーカーの全体実行時間がつり合うように仕事を分散できていないからだと考えられる。

また、$N = 8$ では、各ワーカーの総通信時間が 50 分ほどであった。これは、$N = 8$ では問題が細かく分けられすぎていて、マスタがワーカーに陣を配る回数が多くなり、結果として通信をする際の固定的処理時間が長くなってしまったからだと考えられる。

上記を考えると、$N = 6$ の場合が、最も問題を分ける細かさと通信回数のバランスが取れており、結果として最も実行時間が速くなったのだと考えられる。

E 結論

- 明らかに成立しない陣の探索を減らすように、アルゴリズムやプログラムを改良することにより、実行時間を短縮することができた。
- アルゴリズムやプログラムの枝刈り法を改良することによって実行時間を短縮することができた。
- 今後もさらにアルゴリズムの最適化を図り、最終的には 6 次魔方陣の全解を探索したい。

F 謝辞

- 筑波大学計算科学研究センター　朴 泰祐 教授
  T2K-Tsukuba の利用申請や並列プログラムなどについて多くのご教示をいただきました。

- 産業技術総合研究所　山本 直孝 様
  コンピュータに関わるお仕事をされている方として、細かい質問などをさせていただきました。
- 茨城県立並木中等教育学校　齋藤 達也 先生
  研究・実験の方向性の指導をしてくださいました。
- 茨城県立並木中等教育学校　粉川 雄一郎 先生
  アルゴリズムの開発や研究のまとめにおいて助言をいただきました。

ここに感謝いたします。

G　参考文献

- “計算設備紹介” http://www.ccs.tsukuba.ac.jp/CCS/research/computer
  著者: 計算科学研究センター
  閲覧日: 2013 年 9 月 24 日