Vol.2011-HPC-130 No.4
2011/7/27

メニーコアプロセッサによる進化計算を用いた数独解法の高速化

佐藤裕二† 長谷川直広† 佐藤未来子†† 並木美太郎††

本稿では，数独問題を用いて，メニーコアプロセッサを用いた進化計算並列化により，これまで処理時間の問題から進化計算の適用が検討されなかった問題でも，実用的な時間内で処理できる可能性を示す．精度向上のために遺伝子座間のリンケージを考慮した遺伝的操作を適用し，性能向上のための並列化のモデルとしては，個体数が制限された条件下での初期値依存性対策として，各ストリーミングマルチプロセッサ内で独立に遺伝的アルゴリズムをマルチスレッドで並列実行するモデルを用いる．特に NVIDIA 社から市販されている GPU の一つである GeForce GTX460 上で進化計算を並列処理することで，超難問でも実行時間は数秒程度であり，かつ正答率 100%も達成できることを示す．

Many-core Processor Acceleration for Sudoku Solution with Genetic Operations

Yuji Sato†, Naohiro Hasegawa†, Mikiko Sato†† and Mitaro Namiki††

In this paper, we use the problem of solving Sudoku puzzles to demonstrate the possibility of achieving practical processing time through the use of many-core processors for parallel processing in the application of genetic computation to problems for which the use of genetic computing has not been investigated before because of the processing time problem. To increase accuracy, we propose a genetic operation that takes building-block linkage into account. As a parallel processing model for higher performance, we use a multiple-population coarse-grained GA model to counter initial value dependence under the condition of a limited number of individuals. Specifically, we show that it is possible to reach a solution in a few seconds of processing time with a correct solution rate of 100%, even for extremely difficult problems by parallel processing of genetic computation on a GeForce GTX 460, a commercial GPU produced by the NVIDIA Corporation.

1. はじめに

進化計算を高速化する一つの手段として並列計算機上での進化計算実装方法の研究[1-5]が 1990 年頃から行われてきたが一般には普及していない．一方，近年，マルチコアプロセッサや Graphical Processing Unit (GPU)などのメニーコアプロセッサが一般の PC にも使われ普及の傾向にある．メニーコアプロセッサの特徴は，超並列計算機と比較して，数～数百の小，中規模の並列性を対象とすること，および安価に利用することができることである．このような背景から，メニーコアプロセッサ上への進化計算の並列化研究が始まっている[6-10]．ただし，現状は代表的な GPU を用いて進化計算向きのベンチマークテストを用いた報告が中心である．

我々の研究目的は，実応用問題を用いて，マルチコアプロセッサや GPU などを用いたメニーコア環境での進化計算並列化により，これまで処理時間の問題から進化計算の適用が検討されなかった問題でも，実用的な時間内で処理できることを示すことである．そのための第一歩として，本稿では実応用問題として数独[11]を取り上げ，GPU を用いた高速化の検討を行う．なぜなら，(1)数独は世界的に普及している実応用問題である，(2)シングルコアを前提とするとバックトラックアルゴリズム[12]と比較して処理時間が大きく劣る一方，バックトラックアルゴリズムが並列化困難なのに対して進化計算は並列化向きであるため，GPU のコア数が増えた場合，処理時間が逆転または同等になる可能性がある，(3)GPU はゲームの CG 処理用として最近のゲーム機には搭載傾向のため，一般的なゲーム問題で GPU を用いた高速化を考えることもできる，(4)問題のサイズを 9 x 9 から 16x16 など大きくしていくと，バックトラックアルゴリズムでは処理時間が指数関数的に増加するのに対して，進化計算は確率的探索アルゴリズムなので処理時間が逆転する可能性がある，などの特徴があるためである．

一方，数独解法に進化計算を適用する研究例は多くはなく，難しい問題を実用的な時間で解くことが出来なかった．この原因の一つとして，9 x 9 の問題サイズは，バックトラックアルゴリズムにとっては探索空間が問題にならない程度の大きさである一方，進化計算には数独問題の性質から交叉で有効な部分解（Building Block (BB)）を壊しやすい難解な問題であることが考えられる．この問題に対応するために，既に我々はリンケージ(linkage)[13]を考慮した遺伝的操作を提案し世代数の観点から進化計算を用いた従来手法に比べて大きく改善できる可能性を示した[14]．一方，処理時間では，バックトラックアルゴリズムにまだ大きく劣る状況にあった．

本稿では，GPU による進化計算の並列化の並列化により，進化計算の処理時間が大

† 法政大学情報科学部
Faculty of Computer and Information Sciences, Hosei University
†† 東京農工大学工学部
Faculty of Engineering, Tokyo University of Agriculture and Technology

きく改善される可能性を示す．以下，第 2 章で，並列計算機やメニーコンプロセッサ上での進化計算並列化に関して示す．第 3 章でリンケージを考慮した遺伝的操作概略と GPU 上での並列 GA の実装方法の提案を行なう．第 4 章で数独の超難問を用いて CPU 上での実行と GPU 上での実行の比較評価を行い，最後に考察とまとめを行う．

2. 進化計算と並列高速化

2.1 超並列計算機による高速化

進化計算を探索問題に適用する場合，複数の個体（個体集団）が異なる初期値から最適解の探索を並行して行う．探索に関しては評価値の高い個体（親個体）を基に遺伝的操作を行い次の探索点（子個体）を求める．現在の個体集団に属する全ての個体の評価を行い，その評価値に基づいて親個体の選択，遺伝的操作を行い，次世代の個体集団を生成する操作を 1 世代とし，世代交代を繰り返しながら最適解の探索を行う．各個体に対して行う評価は全て同じ評価関数を適用するため，SIMD 型のアーキテクチャとの相性が良いと考えられる．1 個体当りの実行時間を T_{exe}，個体数を N，世代数を G とした場合，シングルコアの場合の実行時間 T_{single} と超並列計算機を用いた場合の実行時間 T_{mp} はそれぞれ式(1)，式(2)で表せる．

$$T_{Single} = G \times N \times T_{exe} \tag{1}$$

$$T_{mp} = G \times T_{exe} = \frac{T_{Single}}{N} \tag{2}$$

ただし，式(2)では個体ごとにコアプロセッサを割当て，コアプロセッサ間のデータの転送時間は無視できると仮定した場合の理想的な値である．実際には，進化計算では評価値の高い親個体同士が物理的に遠く離れたコアプロセッサに位置する場合があり，遺伝的操作のためのデータ転送時間が問題となる場合が多い．また，データ転送のコンフリクトを避けるための制御が必要となる．従って，遺伝的操作を物理的に隣接したコア間に限定するなど，超並列計算機上での進化計算の効率的な実行に関する研究が 1990 年頃から行われている．

2.2 メニーコアプロセッサによる高速化

メニーコアプロセッサの概略モデルを図 1 に示す．m 個の Streaming Multi-processor (SM)が Global Memory (GM)を介して通信を行う構成となっている．各 SM は n 個のコアプロセッサと共有メモリから構成されている．SM 内のコアプロセッサ間および共有メモリとのデータの読み書きは高速に行える一方，SM 間および SM と GM 間のデータの読み書きは SM 内の 100 倍程度遅い構成を仮定している．図 1 は GPU を想定した構成でるが，$n = 1$ として共有メモリをローカルメモリに置き換えれば，マルチコアプロセッサにも適用可能である．

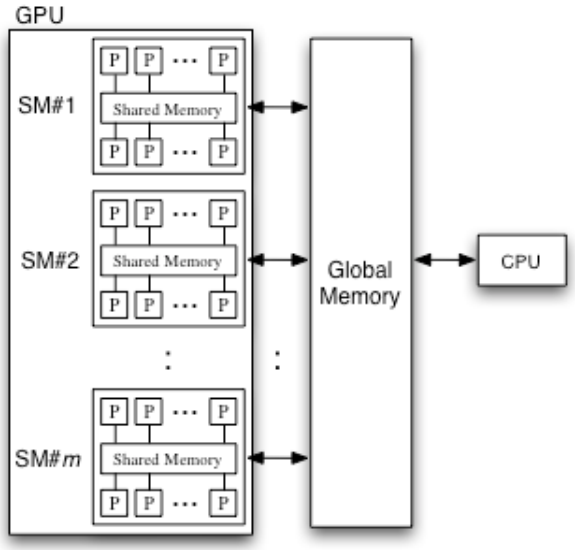

図 1　メニーコアプロセッサの概略モデル

進化計算をメニーコアプロセッサに実装する場合，遺伝的操作などのために SM 間の通信を毎世代行うと性能低下となるため，個体集団を複数のサブ集団に分割して各 SM に割当て，適当なタイミングで SM 間のデータ送受信を行うモデル（島モデル）の実装が妥当と考えられる．SM と GM の間のデータ読み込み時間と書き込み時間がいずれも T_{rw} であるとし，t 世代ごとに GM を介して SM 間でデータ転送を行うとすると，メニーコアプロセッサで島モデルを仮定した場合の実行時間 T_{many1} は式(3)で表せる．

$$T_{many1} = \frac{T_{Single}}{m \times n} + 2T_{rw} \times (m-1)^2 \times \frac{G}{t} \tag{3}$$

島モデルでは，どのようにサブ集団に分割するか，どのようなデータをどのようなタイミングでやり取りするかは設計者が経験に基づき試行錯誤で決める必要がある．ま

た，SM 間の通信速度の性能が低い場合十分な性能が得られないことが考えられる．従って，SM ごとに初期値のみ変えて同一プログラムを実行し，最適解を得た SM が見つかった時点で終了する方式も考えられる．SM ごとに独立して扱った場合の実行時間 T_{many2} は式(4)で表せる．

$$T_{many2} = \frac{T_{Single}}{n} \times \frac{\alpha}{m} \qquad (1 < \alpha < m) \tag{4}$$

式(4)は SM 間で乱数の重なりが少なく分布が一様であるほど効果は大きいと推測される．変数αは乱数の分布に依存する効果を表すための変数である．一方，個体集団サイズが SM 内の共有メモリの容量で制限される．

2.3 遺伝的操作レベルの並列高速化

上記(1)，(2)はいずれも個体レベルの並列高速化である．一方，応用問題や遺伝的操作の設計方法により，個体の評価値を決める計算や遺伝的操作の並列化を考えることができる．1 個体の実行時間 T_{exe} の内，k の割合の処理に対してβ倍の並列化を実現できた場合，T_{exe} は式(5)に示す T_{exe}' に置き換えることができる．

$$T_{exe}' = \left[(1-k) + \frac{k}{\beta} \right] \times T_{exe} \tag{5}$$

以下，ここでは式(4)に示す個体レベルの並列化と式(5)に示す遺伝的操作レベルの並列化を，進化計算を用いた数独問題の解法を GPU に実装した場合の評価を行う．

3. 遺伝的操作を用いた数独解法と GPU への実装方法

3.1 数独解法のためのリンケージを考慮した遺伝的操作

数独の解法に GA を適用する研究例は既に幾つか存在する．例えば，文献[15]ではリージョンごとのサブ染色体をつなぎ合わせた全長 81 ビットの 1 次元の染色体を定義し，リージョン間のつなぎ目のみに交叉位置を限定した一様交叉を行っている．また，文献[16]では同様の染色体の定義を用いて，リージョン間のつなぎ目のみに交叉位置を限定した 1 点交叉，2 点交叉，行または列単位の交叉などに関して交叉法の違いによる有効性の差異を比較調査している．これらの例では，簡単な問題に関しては最適解を容易に見つけ出している一方，初期配置数の少ない難解な問題は現実的な時間内で最適解を求められない場合が多い．その原因として，GA のメインオペレーションである交叉が BB を破壊しやすい設計となっていることが考えられる．この問題を解決する一つの手段として，図 2 に示すリンケージを考慮したリージョンブロックから構成される行または列単位の交叉方法の提案[14]を行った．また，選択の際に，単純なローカルサーチの機能 (Multiple Offspring Sampling: MOS)の追加を行った．選択はトーナメント選択用い，評価関数は「縦・横の各列と行で数字が重複しない」という条件を基に，式(1)に示すように，各列と行の異なる要素数をそれぞれの列と行の評価値 g, h とし，全ての列と行の評価値の総和を個体の評価値 f とした．

$$f(x) = \sum_{i=1}^{9} g_i(x) + \sum_{j=1}^{9} h_j(x)$$
$$g_i(x) = |x_i|, \quad h_j(x) = |x_j| \tag{6}$$

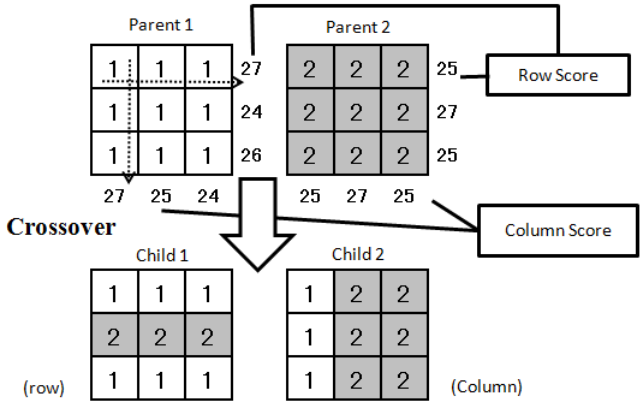

図 2　リージョンブロックから構成される行または列単位の交叉例

表 1 に，数独の初期配置数と最適解を得られた割合および最適解を得るまでにかかった世代数との関係を示す．評価用の数独問題は，問題集[17]から初級（問題番号 1，11）・中級（29，27）・上級（77，106）問題を各 2 問，計 6 問選択した．突然変異のみ適用した場合（一種のランダムサーチ），突然変異と提案する交叉を適用した場合（mut+cross），突然変異と交叉の他に局所探索能力を向上させる工夫を付加した場合（mut+cross+LS）に関して，探索の打ち切り世代数を 100,000 世代に設定し，それぞれ 100 回実行した平均を比較している．100,000 世代で解が得られなかった場合は 100,000 世代として計算して表示している．探索を 100,000 世代で打ち切った場合，難しい問題に対して最適解を得る割合が，突然変異に提案する交叉を加えることにより，さらにローカルサーチの機能を加えることにより明らかに向上していることが分かる．また，解を求めるための平均世代数も削減していることが分かる．

表 1：問題の難易度と解を見つけるまでの平均世代数の関係（打切り世代数: 100,000, 試行回数: 100）

Diffi culty rati	Givens	mut+cross+LS		mut+cross		Swap mutation	
		Count	Average	Count	Average	Count	Average
Easy (No. 1)	38	100	62	100	105	100	223
Easy (No. 11)	34	100	137	100	247	96	6627
Medium (No. 27	30	100	910	100	2274	86	26961
Medium (No. 29	29	100	3193	100	6609	66	42141
Diffi cult (No. 77)	28	100	9482	100	20658	35	77573
Diffi cult (No.	24	96	26825	74	56428	9	94314

3.2 GPU への実装方法

評価のための GPU として SM 内のコアプロセッサ数を考慮して NVIDIA 社の GTX460 を用いる．SM の数に等しい 7 個のブロックを考え，各ブロック内では個体数分のスレッドを用いた並列化を行う．CUDA プログラムを効率良く実行するために，ここでは以下に示す対策を行う．

(1) 初期値依存性対策を考慮した実装

シングルコアを用いた予備実験では，正答率と妥当な実行時間を考慮して個体数を 150 に設定した．一方，個体数 150 とした場合は最適解を見つけるまでの処理時間の分散が大きく，最適解が見つかるまでの世代数は初期値に依存することが分かった．一方，GPU では並列化の効果で個体数を増やしても処理時間に与える影響は少ないと推測される．その反面，並列性を高めるためには個体データなど進化計算に必要な情報は GM ではなく各 SM 内の共有メモリに実装する必要があるが，共有メモリの容量は小さく，必ずしも十分な個体数を確保できない．また，GPU では SM 間のデータ転送速度が，SM 内の通信速度に比べて，極端に遅いため，SM 間の通信が頻繁に必要な実装方式は適さない．従って，ここでは，個体情報などの初期値のみを変えて各 SM 内で同じ進化計算プログラムを実行し，いずれかの SM で解が見つかれば終了する実装方法とする．すなわち，各 SM 内ではスレッドを用いた進化計算プログラムの並列実行を行い，各 SM で初期値を変えて実行することで初期値依存性の対策を考える．各 SM が初期値を変えて同じプログラムを実行することから SIMD 型の並列処理と考えられる．

(2) 高速な共有メモリの節約利用

マルチコアプロセッサ上にベンチマークテストを実装して評価した先の実験[10]では，乱数生成のためのテーブルを各コアに準備することで乱数生成に要する時間の短縮を行った．一方，GTX460 の共有メモリの容量は最大でも 48KB と少量であり，乱数テーブルを各 SM 内の共有メモリに準備すると，必要な個体数を定義できなくなる．そこで，乱数テーブルは準備せず，ライブラリ関数 CURAND を用いることで対応する．CURAND を用いた乱数生成は，各コアに乱数テーブルを準備する場合に比べて処理時間は増えるが，遺伝的操作の度ごとにホストで乱数を生成して各 SM にその値を転送する場合に比べて大きく処理時間を短縮できる．また，読み込み専用のコンスタントメモリ 64KB のうち，8KB がキャッシュとして高速にアクセスできるため，数独の初期配置(4 bytes (int) x 81 = 324 bytes) をこのキャッシュメモリに格納する．共有メモリに割当てるデータは，以下の通りである．

- 個体データ格納用エリア：1 byte (char) x 81 x *N* x 2
- トーナメント選択の作業用エリア：4 bytes (int) x *N*
- 交叉の作業用エリア：4 bytes (float) x *N*/2
- 突然変異の作業用エリア：1 byte (char) x 81 x *N*

(3) サブ染色体単位での並列処理

図 3 に 9 つのリージョンブロック内の突然変異と 3 スレッドへの処理割当の例を示す．3 x 3 のリージョンブロックから構成される数独では，リージョンごとの処理を並列実行するために，1 個体の処理に 9 スレッドを割当てることが考えられる．一方，共有メモリの容量の制限から，ここでは 1 個体の処理に 3 スレッドを割当て，図 2 に示す 3 つのリージョンブロックから構成される行または列ごとに，交叉や突然変異などの処理を 3 スレッドで並列処理することで高速化を図る．

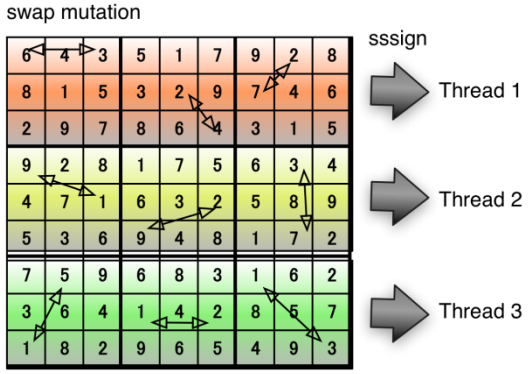

図 3 リージョンブロック内の突然変異とスレッドへの処理割当の例

4. 評価実験

4.1 GPU を用いた高速化

表 2 に，今回の実験で使用した GPU，GTX460 の仕様概略を示す．

表 2 : GTX460 の仕様概略

Board	ELSA GLADIAC GTX 460
#Core	336 (7 SM x 48 Core/SM)
Clock	675MHz
Global memory	1GB (GDDR5 256bits)
Shared memory/SM	48KB
#Register/SM	32768
#Thread/SM	1024

移植性，プログラミングのし易さなどの観点から，Java 言語で作成して先の実験を行っていたため，最初に Java から C へのプログラミング言語の変換を行い，次に GPU 上へ実装して比較評価を行った．先の実験同様，打切り世代数を 100,000 世代とし，個体数を 150 に設定した場合，GPU を用いて高速化した効果[18, 19]を表 3～表 5 に示す．超難問に属する問題として文献[20]で紹介されている，Super Difficult-1 (SD1), Super Difficult-2 (SD2), Super Difficult-3 (SD3)を用いて評価した結果をそれぞれ示している．ただし，打切り世代数を設定しない場合は，全て最適解を見つけることに成功している．尚，GPU の実行環境は以下の通りである．

OS: Ubuntu 10.04
CPU: Phenom X4 945 (3GHz 4 cores)
GPU: GTX 460
GCC: 4.4.3
CUDA Toolkit: 3.2 RC

表 3 : GPU を用いて進化計算を並列高速化した効果 (SD1, Givens: 24)

	Count	Average	CPU time
Java	99	26,333	4m 41s 773
C	91	35,977	1m 10s 218
GTX460 #SM: 7	100	10,014	2s 906

表 4 : GPU を用いて進化計算を並列高速化した効果 (SD2, Givens: 23)

	Count	Average	CPU time
Java	83	45,468	7m 50s 678
C	86	44,250	1m 26s 320
GTX460 #SM: 7	97	22,142	6s 391

表 5 : GPU を用いて進化計算を並列高速化した効果 (SD3, Givens: 22)

	Count	Average	CPU time
Java	74	55,247	9m 28s 888
C	71	57,323	1m 51s 166
GTX460 #SM: 7	95	30,107	8s 727

4.2 スケーラビリティの調査

今回の方式は，SM の数に関してスケーラビリティがあり，SM の数が増えれば初期値依存性対策の効果も向上すると考えられる．そこで，SM の数を 1 から 7 まで変化させて正解が得られるまでの世代数，実行時間と正答率を調査した結果[19]を表 6～表 8 に示す．打切り世代数を 100,000 世代とし，個体数を 150 に設定して 100 回実験した平均を表示している．

表 6 : SM 数と性能の関係 (SD1, Givens: 24)

	Count	Average	CPU time
#SM: 1	62	57,687	16s 728
#SM: 2	80	40,820	11s 845
#SM: 3	89	30,053	8s 733
#SM: 4	98	19,020	5s 527
#SM: 5	97	13,718	3s 985
#SM: 6	99	12,037	3s 495
#SM: 7	100	10,014	2s 906

表 7 : SM 数と性能の関係 (SD2, Givens: 23)

	Count	Average	CPU time
#SM: 1	50	70,067	20s 199
#SM: 2	69	58,786	16s 958
#SM: 3	82	41,757	12s 630
#SM: 4	93	31,254	9s 260
#SM: 5	95	28,709	8s 287
#SM: 6	97	22,065	6s 368
#SM: 7	97	22,142	6s 391

表 8 : SM 数と性能の関係 (SD3, Givens: 22)

	Count	Average	CPU time
#SM: 1	32	82,742	23s 958
#SM: 2	59	68,050	19s 722
#SM: 3	73	57,037	16s 558
#SM: 4	77	47,811	13s 879
#SM: 5	89	37,302	10s 819
#SM: 6	94	33,256	9s 641
#SM: 7	95	30,107	8s 727

4.3 最小世代数

実装技術などの進歩により，GPU を構成する SM 数，SM 内のコアプロセッサ数や共有メモリの容量が増加し，初期値依存性の緩和に有効に働くことが考えられる．初期値依存性の問題が解決された場合の目標性能を見積もるために， SD1 から SD3 を解くために要した最小世代数とその時の実行時間を表 9 に示す.

表 9 : 最小世代数とその実行時間

	Best	CPU time
SD1	83	25 ms
SD2	158	47 ms
SD3	198	76 ms

4.1 考察

表 5〜表 7 から，CPU での実行に比べて GTX460 を用いた進化計算の並列処理により，SD1 では Java 言語で書かれたプログラムに対しては 97.0 倍，C 言語で書かれたプログラムに対しては 24.2 倍高速となっている．SD2 では Java 言語で書かれたプログラムに対しては 73.6 倍, C 言語で書かれたプログラムに対しては 13.5 倍高速となっている．SD3 では Java 言語で書かれたプログラムに対しては 65.2 倍，C 言語で書かれたプログラムに対しては 12.3 倍高速となっている．実行時間が Java 言語で 10 分程度必要だった問題も 10 秒以内で実行可能であり，また正答率も 100%近いことから，GPU などを用いた進化計算並列化により，これまで処理時間の問題から進化計算の適用が検討されなかった数独問題でも, 実用的な時間内で処理できることを示せたと考える．また，表 8〜表 10 から，SM の数の増加に伴い，実行時間は減少し，正答率は確実に増加していることが分かる．特に表 11 から，SM の数が十分増えることで，あるいは何らかの手法で初期値依存性の問題が解決されれば，GPU を用いた進化計算並列化で, 数独の超難問を安定して 1 秒以内に解くことができる可能性があると考える．

一般に，個体数を増やした方が並列化の効果は大きいと考えられる．一方，3.3 (2) で示した共有メモリに割当てられるデータから，個体数を N としてその総量は $249N$ となる．従って，48KB の容量の共有メモリに格納できる個体数の上限が 192 であることが分かった．また，192 が SM 内のプロセッサの数 48 の倍数であり，また CUDA のスレッド処理単位 32 の倍数であることを考慮して, 個体数を 192 に設定した場合の実行時間と正答率を表 10 に示す.

表 10 : SM 当りの個体数の上限値を設定した場合の実行時間と正答率

Sudoku	Count	Average	CPU time
SD1	100	9072	2s 751
SD2	100	13,481	4s 530
SD3	100	22,799	6s 862

個体数を 150 に設定した場合と比較して，SD1 では，正答率が 100%へのまま，処理時間が約 5%短縮している．SD2 では，正答率が 97%から 100%へ向上し，処理時間が約 29%短縮している．SD3 では，正答率が 95%から 100%へ向上し，処理時間が約 21%短縮している. 適切な値に個体数を設定することにより, CPU での実行に比べて，GPU を用いて進化計算を並列化することで，C 言語で書かれたプログラムに対して，SD1 では 25.5 倍，SD2 では 19.1 倍，SD3 では 16.2 倍まで高速化率を向上できることが分かった. また 3 種類全ての超難問に対して 100,000 世代以内で正答率 100%を達成できることが分かった.

5. おわりに

本稿では，数独問題を用いて，マルチコアプロセッサや GPU などを用いたメニーコア環境での進化計算並列化により，これまで処理時間の問題から進化計算の適用が検討されなかった問題でも，実用的な時間内で処理できる可能性を示した．特に NVIDIA 社から市販されている GPU の一つである GTX460 上で進化計算を並列化することで，CPU を用いて C 言語で書かれたプログラムを実行する場合と比較して，実行時間は 16 倍～25 倍高速化し，かつ超難問でも正答率 100%を達成できることを示した．今後はより集積度の高い GPU を用いて遺伝的操作の見直しや島モデルなどの実装を行い，バックトラックアルゴリズムとの比較を行う．

謝辞 本研究の一部は北海道大学情報基盤センター2011 年度公募型共同研究の助成を受けた．

参考文献

1) V. S. Gordon, and D. Whitley, “Serial and parallel genetic algorithms as function optimizers,” in Proc. of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann, 1993, pp. 177–183.
2) H. Mühlenbein, “Parallel genetic algorithms, population genetics and combinatorial optimization,” in Proc. of the 3rd International Conference on Genetic Algorithms, 1989, pp. 416–421.
3) H. Mühlenbein, “Evolution in time and space - the parallel genetic algorithm,” in Foundations of Genetic Algorithms. Morgan Kaufmann, 1991, pp. 316–337.
4) R. Shonkwiler, “Parallel genetic algorithm,” in Proc. of the 5th International Conference on Genetic Algorithms, 1993, pp. 199–205.
5) E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, 2000.
6) J.-H. Byun, K. Datta, A. Ravindran, A. Mukherjee, and B. Joshi,“Performance analysis of coarse-grained parallel genetic algorithms on the multi-core sun UltraSPARC T1,” in Southeastcon, 2009. SOUTHEASTCON’09. IEEE, March 2009, pp. 301–306.
7) R. Serrano, J. Tapia, O. Montiel, R. Sep´ulveda, and P. Melin, “High performance parallel programming of a GA using multi-core technology,” in Soft Computing for Hybrid Intelligent Systems, 2008, pp. 307–314.
8) S. Tsutsui and N. Fujimoto, “Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study,” in GECCO ’09: Proc. 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, 2009, pp. 2523–2530.
9) Asim Munawar, Mohamed Wahib, Masaharu Munetomo, Kiyoshi Akama. Theoretical and Empirical Analysis of a GPU Based Parallel Bayesian Optimization Algorithm. In Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, December 2009, pp. 457-462.
10) Mikiko Sato, Yuji Sato, and Mitaro Namiki. Proposal of a Multi-core Processor from the Viewpoint of Evolutionary Computation. In Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, July 2010. pp. 3868–3875.
11) Wikipedia. Sudoku. Available via WWW: http://en.wikipedia.org/wiki/Sudoku (cited 8.3.2010).
12) Wikipedia. Backtracking. Available via WWW: http://en.wikipedia.org/wiki/Backtracking (cited 1.11.2011)
13) Ying-ping Chen, Tian-Li Yu, Kumara Sastry, and David E. Goldberg. A Survey of Linkage Learning Techniques in Genetic and Evolutionary Algorithms. In IlliGAL Report No. 2007014 April 2007.
14) Yuji Sato and Hazuki Inoue. Solving Sudoku with Genetic Operations that Preserve Building Blocks. In Proceedings of the IEEE Conference on Computational Intelligence in Game. IEEE, August 2010, pp. 23–29.
15) T. Mantere and J. Koljonen, “Solving and Ranking Sudoku Puzzles with Genetic Algorithms,” in Proceedings of the 12th Finnish Artificial Conference STeP 2006, October, 2006, pp. 86-92.
16) T. Mantere and J. Koljonen, “Solving, Rating and Generating Sudoku Puzzles with GA,” in Proceedings of the IEEE Congress on Evolutionary Computation, September, 2007, pp. 1382-1389.
17) Number Place Plaza (eds.), “Number Place Best Selection 110,” vol. 15, ISBN-13: 978-4774752112, Cosmic mook, December, 2008.
18) 佐藤裕二、北咲也、高嶺和也、長谷川直弘．遺伝的操作を用いた数独解法と GPU による高速化に関して．第 3 回進化計算シンポジウム 2010 論文集, pp. 47-52.
19) Yuji Sato, Naohiro Hasegawa and Mikiko Sato. GPU Acceleration for Sudoku Solution with Genetic Operations. In Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, June 2011.
20) Super difficult Sudoku’s. Available via WWW: http://lipas.uwasa.fi/~timan/sudoku/EA_ht_2008.pdf#search='CT20A6300%20Alternative%20Project%20work%202008' (cited 8.3.2010).