

進化計算の並列化への課題： マルチスレッドプログラミングから超多スレッドプログラミングへ

筒井 茂義[†], 藤本 典幸^{††}

阪南大学経営情報学部[†], 大阪府立大学大学院理学系研究科^{††}

1 はじめに

進化計算の高性能化の手法として、大規模な並列計算機やネットワークで結合された多数の計算機を利用する並列進化計算の研究が活発に行われてきた²⁾。今後、進化計算の利用がより多くの分野に広がって行くために、多くのエンドユーザが、安価な計算機環境において実現できる高性能な並列進化計算の実現が重要になると思われる。

近年、マルチコア型の高性能CPUが開発され、一般の安価なPCにも利用できるようになってきた。このタイプの計算機では、2, 4, 6といった数の演算装置を有し、マルチスレッドプログラミングにより、並列進化計算が比較的容易に実現できる。

一方、昨今、いわゆるグラフィックボードとして一般のPCに用いられている画像処理用演算装置 (Graphics Processing Unit, GPU) を用いて流体力学、医療用画像分析、統計データ処理などの科学技術計算の並列計算を実現する並列計算 (GPU Computation, 以下GPU計算) の研究が注目され、CPUに対して数10~100倍の高速化が図られたという有望な結果が報告されている^{6, 4, 10)}。

GPU計算を適用した並列進化計算の研究に関してはGPへの適用を中心に試みられているが、ハードウェアの制限等からまだあまり進んでいないのが現状である (詳しくは文献⁹⁾ 参照)。本年7月に開催されたGECCO-2009では、進化計算へのGPU計算の適用に関するワークショップ

「2009 Computational Intelligence on Consumer Games and Graphics Hardware (CIGPU-2009)」が開催され、2010年には、WCCI (バルセロナ) にあわせてCIGPU-2010が開催される。

本稿では、筆者らが進めている、(1) マルチコアプロセッサを内蔵するPCによる進化計算の並列化の研究⁸⁾、ならびに、(2) GPU計算による進化計算の並列化の研究⁹⁾ について述べ、今後の研究課題について考察する。

2 マルチコア計算機による進化計算のマルチスレッド並列化

先に提案した順序表現向きの分布推定アルゴリズム (EDA) の一種であるEdge Histogram Based Sampling Algorithm (EHBSA)¹¹⁾ の改良型モデル (enhanced EHBSA, 以下eEHBSA) のマルチコアPCによる並列化を、TSPをテスト問題として行った結果について述べる。

2.1 改良型EHBSA (eEHBSA)

2.1.1 エッジの分布

eEHBSAでは、EHBSAと同様、分布推定モデルに、集団全体に含まれる個体のエッジの分布をマトリックス形式で表現したEHM (Edge Histogram Matrix) を用いる。図1は、問題サイズ $L = 5$ のTSPに対する集団サイズ $N = 5$ の集団 $P(t) = \{s_1^t, s_2^t, \dots, s_5^t\}$ のEHMの例である。EHMの各エントリ e_{ij} の整数部分は、集団におけるエッジ $i \rightarrow j$ の総数であり、小数点以下の値 (同図では、0.1としている) は、個体生成時における外乱を与えるバイアスであり、アルゴリズムのパラメータの一つである。ここでは、対称TSPの場合のようにエッジ $i \rightarrow j$ とエッジ $j \rightarrow i$ とが同等である場合を示しているが、スケジューリング問題のようにエッジ $i \rightarrow j$ とエッジ $j \rightarrow i$ とが異なる場合にも同様にEHMは非対称マトリックスとして表現できる。

Parallelization of Evolutionary Computation: From Multi-Thread Programming to Many-Thread Programming

[†] Shigeyoshi Tsutsui (tsutsui@hannan-u.ac.jp)

^{††} Noriyuki Fujimoto (fujimoto@mi.s.osakafu-u.ac.jp)

Department of Management and Information Science, Hannan University (†)

Graduate School of Science, Osaka Prefecture University (††)

$$\begin{array}{l}
s'_1 = (1, 2, 3, 4, 5) \\
s'_2 = (2, 4, 5, 3, 1) \\
s'_3 = (4, 5, 3, 2, 1) \\
s'_4 = (5, 1, 4, 2, 3) \\
s'_5 = (3, 2, 4, 5, 1)
\end{array}
\begin{array}{l}
\begin{pmatrix} 0 & 3.1 & 2.1 & 2.1 & 3.1 \\ 3.1 & 0 & 4.1 & 3.1 & 0.1 \\ 2.1 & 4.1 & 0 & 1.1 & 3.1 \\ 2.1 & 3.1 & 1.1 & 0 & 4.1 \\ 3.1 & 0.1 & 3.1 & 4.1 & 0 \end{pmatrix} \\
\end{array}$$

(a) $P(t)$ (b) EHM^t

Fig. 1 EHMの一例

2.1.2 個体生成法

図2に一つの新個体を生成する方法を示す．新個体は以下のようにして生成される．

- (i) 現在の集団より，テンプレートに用いる個体を一つ選ぶ．
- (ii) EHMを用いて生成するノード数 l_s を式(1)で示す確率密度関数によりサンプリングにより決定し，ノード数 $l_p = L - l_s$ をテンプレートから新個体のメモリ領域にコピーする．コピーの開始位置 p_{top} はランダムに決定する．
- (iii) 残りの l_s 個のノードは，EHMをもとに式(2)に基づいてサンプリングする．

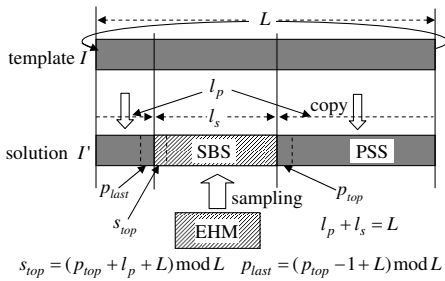


Fig. 2 個体の生成法

先のEHBSAの研究では， l_s の決定に n カットポイント法を用いていたが¹¹⁾，この方法では， l_s の平均値($E(l_s)$)が L/n ($n = 2, 3, \dots$)に制限されるという問題があり，改良型EHBSAでは， $E(l_s)$ が $L \times \gamma$ ($\gamma \in [0, 1]$)となるパラメータ γ を導入した．

$$f(l_s) = \begin{cases} \frac{1-\gamma}{L\gamma} \left(1 - \frac{l_s}{L}\right)^{\frac{1-2\gamma}{\gamma}} & \text{for } \gamma \in (0, 0.5] \\ \frac{\gamma}{L(1-\gamma)} \left(\frac{l_s}{L}\right)^{\frac{2\gamma-1}{1-\gamma}} & \text{for } \gamma \in (0, 1] \end{cases} \quad (1)$$

(iii)のサンプリングでは，現在位置のノードが i であるとき， i の次の位置のノード j は，次式の確率 p_{ij}

$$p_{i,j} = \begin{cases} \frac{e_{i,j}}{\sum_{s \in F(i)} e_{i,s}} & \text{if } j \in F(i) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

で決定される．ただし， $F(i)$ は，現在のノード i からつぎに選べるノードの集合である．この式に基づくサンプリングの計算量は一般に $O(L^2)$ になり，問題サイズが大きくなるとサンプリングに要する時間が問題となる．このような場合，候補リスト¹⁾を使うと計算量を $O(L)$ に近づけることができる．

2.1.3 世代交代モデル

EHBSAでは，1世代に一個体を生成するモデルを用いたが，本研究では，図3に示すように，1世代で N 個体を生成する方式とし，効率的な世代交代モデルに改良している．同図において，新しい個体 I'_i は，集団 $P(t)$ の個体 I_i をテンプレートとして用いる ($i = 1, 2, \dots, N$)．各 i のペア(I_i, I'_i) ($i = 1, 2, \dots, N$)を比較し，良い個体を次の世代 $P(t+1)$ のメンバーとする． I_i と I'_i とを比較するこの方式は，Mahfoudのdeterministic crowding法⁵⁾のように多様性維持に効果がある．また，この方式では，図2で示した γ も重要な設計パラメータとなる．

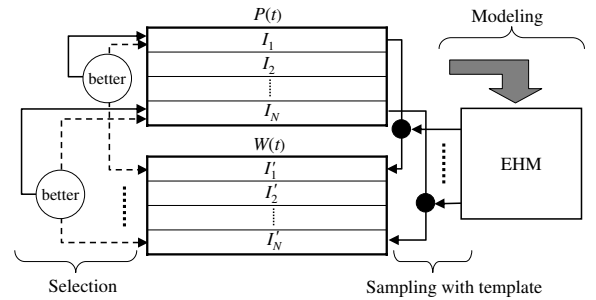


Fig. 3 世代交代モデル

2.1.4 eEHBSAのシングルスレッド性能

eEHBSAの並列化されていない場合の基本性能を見るために，巡回セールスマン問題(TSP)を用いて評価した結果を述べる．ここでは，(1)小規模問題：berlin52, pr76, (2)中規模問題：att532, rat783, (3)大規模問題：fl3795, rl5934の3つの規模の問題を取り上げる．実験条件は以下の通りである．(1)小規模問題に対しては，集団サイズを $2L$ ，最大解生成数を $2L \times 10000$ とし，

ローカルサーチは適用しない。(2)中規模問題に対しては，集団サイズを $L/15$ ，最大解生成数を $L \times 1000$ とし，ローカルサーチには3OPTを適用する。(3)大規模問題に対しては，集団サイズを4とし，最大解生成数を $L/10$ とし，ローカルサーチにはTSPにおける最強のローカルサーチとされているLin-Kernighan (LK) ヒューリスティックを適用する。

プログラミング言語にはJavaを用い，マシンにはIntel® Core™ i7 965 プロセッサを用いる。LKコードにはConcorde³⁾を用い，JNIでEHBSAコードと結合した。それぞれの問題に対して γ を0.1から1.0まで0.1刻みで実験を行った。各実験における試行回数は20回とした。各規模の問題に対して，#OPT (20回の実験中で最適解を発見した回数)および T_{avg} (最適解を発見した実験において，最適解を発見するのに要した時間の平均)による結果をそれぞれ，図4，図5，および図6に示す。

これらの結果からeEHBSAでは， γ の適切な設定が重要であることが分かる。また，いずれの規模の問題においても， $0.2 \leq \gamma \leq 0.4$ において#OPT=20であり，また， T_{avg} も小さい値となっていることが分かる。

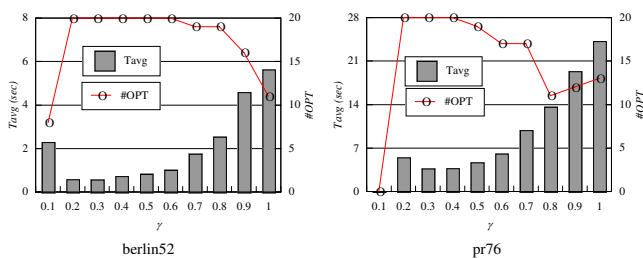


Fig. 4 小規模問題 (berlin52, pr76) の結果 (ローカルサーチなし)

2.2 マルチスレッドプログラミングの実現法

eEHBSAの並列化方式として，同期マルチスレッドモード(synchronous multi-thread mode, SMTM)と非同期マルチスレッドモード(asynchronous multi-thread mode, AMTM)の2つの並列化スレッドプログラミングを考える。使用言語はJavaである。使用プロセッサは，2.1.4項で述べたIntel® Core™ i7 965 である。このプロセッサは4つのコアを内蔵している。BIOSにおいてハイパー

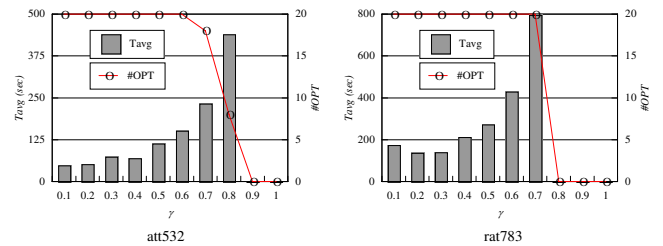


Fig. 5 中規模問題 (att532, rat783) の結果 (3OPT ローカルサーチ)

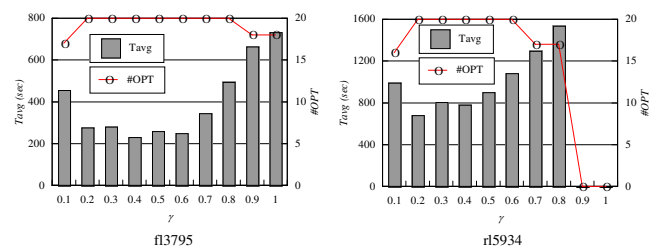


Fig. 6 大規模問題 (fl3795, rl5934) の結果 (LK ローカルサーチ)

スレッディング機能を有効に設定するとOSから見つけ上8個のコアが存在するが，ここではその機能は用いないとした。したがって，本研究では，4つのスレッドを生成してeEHBSAの並列実行を評価する。eEHBSAの繰り返し過程は以下となる。

- (1) $t \leftarrow 0$. 個体 I_i ($i = 1, 2, \dots, N$)からなる初期集団 $P(t)$ 生成。
- (2) $P(t)$ の個体を評価(ローカルサーチを適用する場合は適用後評価し， I_i も変更)。
- (3) $P(t)$ からEHMを作成。
- (4) 新個体 I'_i ($i = 1, 2, \dots, N$)を，EHMおよび I_i の部分解部から生成。
- (5) 各 i ($i = 1, 2, \dots, N$)に対して I_i と I'_i とを比較し， I'_i の方がよければそれを I_i と置き換え，集団 $P(t)$ を更新する。
- (6) $t \leftarrow t + 1$ 。
- (7) 終了条件を満たせば終了，満たさなければ(3)へ。

SMTMでは，各ステップの処理を同期して行う。それに対して，AMTMでは上記(3)まで同期モードで動作し，それ以降は独立に未処理の個体を並列実行する。このとき，ステップ(3)では，全ての個体が更新されてからとはせず， I_i と I'_i の入れ替えとなったときには，現在のEHMから I_i

Table 1 マルチコア計算機による並列進化計算の結果

Instance	Local search	Non-parallelized e EHBSA		Parallelized e EHBSA ($n_{thread}=4$)					p -value	
		T_{avg} (sec)	SE	SMTM		$Speedup$	AMTM			
				T_{avg} (sec)	SE		T_{avg} (sec)	SE		$Speedup$
		$confidence-interval$	$confidence-interval$	$confidence-interval$	$confidence-interval$		$confidence-interval$			
oliver30	no	0.082	0.004	0.05	0.08	1.5	0.02	0.04	3.4	5.93E-07
		[0.074, 0.091]		[0.045, 0.063]			[0.020, 0.004]			
gr48		0.60	0.04	0.29	0.01	2.1	0.16	0.01	3.8	1.31E-09
		[0.53, 0.67]		[0.26, 0.31]			[0.15, 0.17]			
berlin52		0.57	0.03	0.29	0.02	2.0	0.13	0.00	4.3	1.58E-07
	[0.51, 0.64]		[0.25, 0.33]			[0.12, 0.14]				
pr76		3.71	0.14	1.46	0.07	2.5	0.86	0.11	4.3	8.01E-04
	[3.40, 4.01]		[1.31, 1.61]			[0.74, 1.20]				
lin318	3-OPT	2.57	0.64	0.84	0.06	3.1	0.76	0.04	3.4	0.310
		[1.23, 3.91]		[0.70, 0.98]			[0.68, 0.84]			
pcb442		12.49	2.21	3.12	0.28	4.0	2.92	0.26	4.3	0.606
		[7.85, 17.12]		[2.54, 3.70]			[2.38, 3.47]			
att532		73.84	12.29	21.84	3.00	3.4	19.41	2.78	3.8	0.549
	[48.11, 99.56]		[13.57, 28.11]			[13.81, 25.02]				
rat783		139.48	7.57	38.77	2.72	3.6	38.37	2.75	3.6	0.918
	[123.63, 155.32]		[33.07, 44.47]			[32.62, 44.12]				
fl3795	LK	280.88	34.70	105.55	19.08	2.7	85.51	12.28	3.3	0.384
		[208.24]		[65.60, 145.50]			[59.81, 111.21]			
ri5934		807.27	76.04	252.44	26.51	3.2	209.62	16.91	3.9	0.183
	[648.12, 966.43]		[196.96, 307.93]			[174.22, 245.01]				

SE: standard error

の属するエッジを取り出し（それらのエッジの数を -1 する）、 I'_i に属するエッジを加える（それらのエッジの数を $+1$ する）。この仕組みにより、AMTMでは処理の待ち合わせに伴う遅延は発生しない。ただし、処理が非同期に行われるので、各個体に対して各ステップの処理は完全な世代単位には行われなくなる。

2.3 結果と考察

各種サイズのTSPを用いて、2つの並列化方式の結果を表1に示す。実験回数は20とした。この実験では20回の実験で全て最適解を発見している（#OPT=20）。したがって、 T_{avg} は、最適解を見つけるまでに要した時間（秒）の20回の実験の平均である。 $Speedup$ は、並列処理を行わない場合の T_{avg} をマルチスレッド方式による場合の T_{avg} で割ったものである。同表には、各 T_{avg} の95%信頼区間ならびにSMTMとAMTMの $Speedup$ の比較のP-値も示した。

SMTMの結果を見ると、ローカルサーチを用いない小規模問題とローカルサーチを用いる中、大規模問題で大きな違いがあることが分かる。中、大規模問題では、速度比が3.1~3.7であるのに対して、ローカルサーチを用いない小規模問題では速度比が1.5~2.5と高速化の度合いが小さくなっている。

一方、AMTMの結果を見ると、中、大規模問題ではSMTMの結果と大きな違いがなく、同程度の高速化の結果が得られている。大きな違い

は、小規模問題の場合である。この場合にも、SMTMの場合と異なりAMTMでは3.4~4.4と高速化の結果が得られている。ローカルサーチを適用しない場合にはサンプリングに要する時間の割合が大きく（図7参照）、また、 e EHBSAではこの時間は確率的に変動する。したがって、SMTMではスレッド間の待ち時間が長くなる。AMTMではこのような待ち時間はない。

3 GPU計算による進化計算の超多マルチスレッド並列化

ここでは、2次割当て問題（quadratic assignment problem, QAP）の解法にGPU計算を用いて並列進化計算を実行した結果⁹⁾について述べ、その結果の考察を行う。

QAPは、 L 個からなる部門を L 個の場所に次式で定義される関数 $f(\phi)$ が最小になるように割当て、すなわち、 $f(\phi)$ が最小になるような順列 ϕ を求めるという組合せ最適化問題である。

$$f(\phi) = \sum_{r=1}^L \sum_{s=1}^L b_{rs} a_{\phi(r)\phi(s)}. \quad (3)$$

ここで、 $A = (a_{pq})$ および $B = (b_{rs})$ はそれぞれ $L \times L$ のマトリックスであり、 ϕ は $\{1, 2, \dots, L\}$ の順列である。マトリックス A と B は、それぞれ、場所 p, q 間の距離、部門 r, s 間の流量を表して

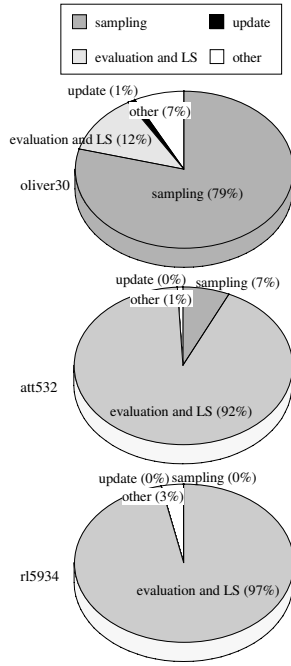


Fig. 7 eEHBSA の処理時間の分布 (oliver30, att532, rl5934)

いる。QAPは評価式が距離と流量の積となっており、両情報が相互に絡み合うため、TSPに比較して格段に困難な問題である。

3.1 GPU計算による超多スレッドプログラミングの一実現法

この研究で用いたGPUは、NVIDIA® GeForce™ GTX285 である。図8にこのアーキテクチャを示す。基本演算のスループットが1命令/クロックである Thread Processor (以下、TP) からなり、8個のTPを単位として1つのMulti-processor (MP) を構成している。MP内のTPは、レジスタ並みの高速アクセスが可能な Shared Memory (SM) を介してデータを共有することができる。SMのサイズは16KBである。MPの数は30である。異なるMPに属するTPのデータ共有は、VRAM経由となり速度が遅くなる。

プログラミング環境としては、CUDA (compute unified device architecture) と呼ぶC言語の統合開発環境があり、コンパイラやライブラリなどから構成されている⁶⁾。Windows環境では、Microsoft Visual Studioに組み込んで使うことができ、本研究では、この方法を用いた。CUDAでは、合計 $8 \times 30 = 240$ 個のTPを動作待ち時間なく有効に動作させる(メモリアクセス遅延を隠す)ために数千という超多スレッドプログラミ

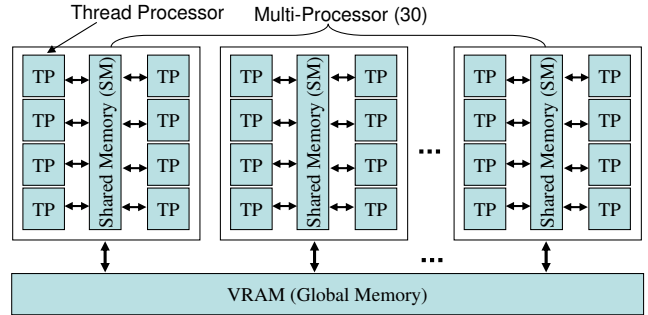


Fig. 8 NVIDIA GeForce GTX285 のアーキテクチャ

ングを前提としている。このため、*grid*, *block*, *thread* という概念を導入し、物理的な構成を直接的に意識せずに超多スレッドプログラミングができる環境を提供している。

GPU計算を進化計算の並列化に応用する際の大きなネックは、SMのサイズにある。個体の集団をVRAMにおくと、各TPからのメモリアクセスの遅延が大きいため、高速化に不利となる。このため、SM (16KB) の活用が重要となる。本研究では、一つのサブ集団 $P(t)$ をSMに置くという分散型GAモデルを考案し、高速化を図ることを考えた。

QAPは大きな問題でも問題サイズが高々150程度である。したがって、順列表現である染色体の遺伝子には *unsigned char* 型を使用することができ、個体表現に要するSMの消費量を節約できる。各サブ集団の世代交代モデルでは、図3で述べたのと同様、一つの個体の処理がCUDA環境で1つのスレッドとなるよう工夫している。サブ集団サイズを N とすると、SMの消費量は、 $(2L \times N)$ バイトとなる。今回は $N = 128$ に固定し、最大問題サイズを $L = 56$ とした。

交叉にはPMXオペレータ、突然変異には2つの任意の遺伝子座の値を入れ替えるSWAPオペレータを使用した。ローカルサーチは適用していない。各サブ集団の世代交代モデルを図9に示す。

各MPにおけるサブ集団の個体は、図10に示すように500世代毎にVRAMを介してシャッフルする。この意味で、GPU内の進化モデルは、分散GAにおける島モデルの一形態といえる。個体数の総数は、 $128 \times 30 \times k (k = 1, 2, \dots)$ であり、これがCUDAプログラミング環境における総スレッド数になる。なお、スレッドのスケジューリ

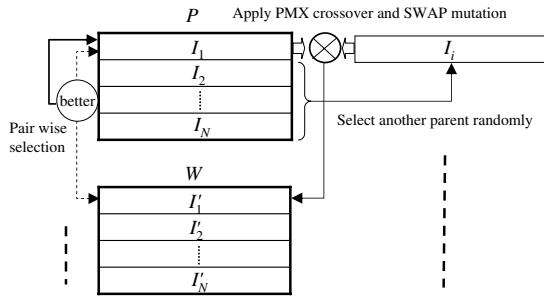


Fig. 9 GPU計算における各サブ集団の世代交代モデル

ングはハードウェアによって行われる。

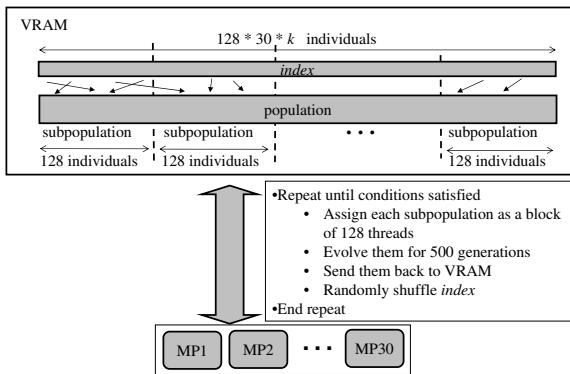


Fig. 10 GPU計算における進化計算の構成

3.2 結果と考察

QAPの各テスト問題に対して、GPU計算によるスレッド並列化の結果と、2章で用いたIntel® Core™ i7 965 プロセッサにおいてシングルスレッドで実行した結果とを、最適解を発見するのに要した時間 (T_{avg}) で比較したものを表2に示す。ここで、CPUにおけるGAのモデルとして、論理的にGPU計算の場合と同じとした場合 (GA-1) と島間の個体の交換をチューニングしたモデル (GA-2) の結果を示した。当然GA-2の結果の方が、GA-1の結果よりも優れている。GPU計算のGA-2に対する *Speedup* の値として3~12という結果が得られた。

画像処理や流体力学などへの応用では、CPUの数10倍~100倍以上の高速化が期待できる¹⁰⁾。今回の結果はこれらの分野と比較すると十分とはいえないが、今後計算モデルの改良により、さらなる速度の向上は十分期待できる。以下では、進化計算においてGPU計算を用いる際に課題となる点について考察し、今後への研究課題

を述べたい。

進化計算にGPUを用いる際に課題となる事項としては以下をあげることができる。

- 進化計算では、処理が確率的に行われ、これにより多くの分岐命令が現れる。GPUではスレッドはブロックごとにMPに割り当てて実行されるが、各ブロック内のスレッドはウォープに分割されて (GTX285の場合、32スレッド単位) SIMD風に処理が行われるので、スレッド間でTrueとFalseの処理がシリアルに行われ、このときMP内のTPにアイドル状態が発生する場合がある。また、分岐がなく同じ処理が行われる場合でも、各スレッド間での処理時間が異なると、全体の処理時間は長い方に合わされ、TPにアイドル状態が発生する。したがって、このような問題を少なくする計算モデルを工夫する必要がある。
- 各MPで共有される高速なメモリであるSMが一般に小さく、大きな問題に適用する場合に問題分割などの十分な工夫が必要となる。
- 各MP間のデータ共有はVRAM経由となる。基本演算命令の実行時間は4クロックであるのに対して、VRAMアクセスの遅延は400から600クロックである (GTX285の場合)⁶⁾。このためMP間でのデータ共有や交換の方法に工夫が必要となる。
- 各スレッドで用いる乱数系列の管理に工夫を要する。
- スレッドで用いるローカル変数にはレジスタが割り当てられ、高速な処理が行える半面、進化計算で多用される配列がローカル変数として使えない。したがって、ローカル変数の効率的な使用には十分な工夫が必要となる。

以上の点に関しては今後のGPUの進歩にも期待できる。事実、NVIDIAは公表している次期GPU (Fermi) では、SMの容量増加やVRAMアクセスに対するキャッシュの導入などが図られている⁷⁾。

しかし、GPUを並列進化計算に用いる研究テーマとしては、数百もの演算ユニットを如何に有効活用するかということが本質であり、進化計算のモデルまでさかのぼった研究を行うことが重要である。たとえば、複数のGPUによる

Table 2 GPU計算による超多マルチスレッドプログラミングのQAPにおける結果

QAP instances	GPU computation				CPU computation with single thread								Speedup ratio to CPU computation	
	Total population	#OPT	T_{avg} (sec)	std	GA-1				GA-2					
					Total population	#OPT	T_{avg} (sec)	std	Total population	#OPT	T_{avg} (sec)	std	GA-1	GA-2
tai20b	128×30×1	10	0.064	0.005	128×30×1	10	0.428	0.039	128×30×1	10	0.422	0.042	6.7	6.6
tai25b	128×30×1	10	0.169	0.015	128×30×1	10	1.386	0.135	128×30×1	10	1.286	0.145	8.2	7.6
kra30a	128×30×5	10	2.002	1.741	128×30×2	9	9.651	4.541	128×30×4	9	11.870	3.115	4.8	5.9
kra30b	128×30×5	9	1.332	0.732	128×30×5	8	23.399	11.492	128×30×4	9	16.745	11.164	17.6	12.6
tai30b	128×30×3	10	0.947	0.576	128×30×3	10	22.649	6.830	128×30×1	10	7.203	6.274	23.9	7.6
tai35b	128×30×4	10	2.510	0.740	128×30×3	10	22.649	6.830	128×30×1	10	7.203	6.274	9.0	2.9
ste36b	128×30×4	10	3.337	1.056	128×30×4	10	33.274	13.062	128×30×2	10	14.675	3.836	10.0	4.4
tai40b	128×30×1	10	1.088	0.087	128×30×1	9	6.016	0.486	128×30×1	10	5.811	0.482	5.5	5.3

#OPT: the number of runs in which the algorithm succeeded in finding the optimal solution

T_{avg} : the average time to find optimal solutions in successful runs in second

std: standard deviation of T_{avg}

負荷分散やマルチコアCPUによる並列計算との機能分担など、新しい並列進化計算のモデルの検討などである。これらにより上述した課題も解決できると考えられる。

4 むすび

本稿では、筆者らが進めている、マルチコアプロセッサを内蔵するPCによる進化計算の並列化ならびにGPU計算による進化計算の並列化の研究について述べ、結果の考察を行った。いずれも、安価なPCで実現できる点に大きな特徴がある。CPUに関しては、一般のPC用では2コアが主流であるが、今後4, 6, ...へと安価なCPUが浸透していくであろう。GPU計算に関しては、進化計算の並列化の新しい手法として大いに期待され、3.2節で述べたような研究課題に対して今後取り組んでいく予定である。

参考文献

- 1) Bentley, J. L.: Fast algorithms for geometric traveling salesman problems, *ORSA Journal on Computing*, Vol. 4, pp. 387–411 (1992)
- 2) Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, Kuwer Academic Publishers (2000)
- 3) Concorde TSP Solver (2006), <http://www.tsp.gatech.edu/concorde.html>
- 4) Fujimoto, N.: Dense Matrix-Vector Multiplication on the CUDA Architecture, *Parallel Processing Letters*, Vol. 18, No. 4, pp. 511–530 (2008)
- 5) Mahfoud, S.: A Comparison of Parallel and Sequential Niching Methods, in *Proceedings of the Six International Conference on Genetic Algorithms*, Morgan Kaufmann (1995)

- 6) NVIDIA, : CUDA Programming Guide 2.3 (2009), www.nvidia.com/object/cuda_develop.html
- 7) NVIDIA, : Next Generation CUDA Architecture (2009), www.nvidia.com/object/fermi_architecture.html
- 8) Tsutsui, S.: Parallelization of an Evolutionary Algorithm on a Platform with Multi-core Processors, *Proceedings of the 9th international conference on Artificial Evolution (EA'09)* (2009)
- 9) Tsutsui, S. and Fujimoto, N.: Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: A Case Study, *Proc. of the GECCO 2009 Workshop on Computational Intelligence on Consumer Games and Graphics Hardware (CIGPU-2009)*, pp. 2523–2530 (2009)
- 10) 青木 尊之: フルGPUによるCFDアプリケーション, *情報処理*, Vol. 50, No. 2, pp. 107–115 (2009)
- 11) 筒井 茂義: エッジヒストグラムを用いる順序表現向き確率モデルGAの提案, *人工知能学会論文誌*, Vol. 18, No. 4, pp. 173–182 (2003)