Vol.2011-HPC-129 No.19
2011/3/16

TSP におけるアントコロニー最適化法の GPU による高速化

小橋一太[†]　藤井昭宏[†]　小柳義夫[†]

本研究では TSP 向けにアントコロニー最適化法(ACO)の GPU 上で高速実装手法を提案し評価する．ACO は確率的でメタヒューリスティックな解法であり，GPU 上の並列実装では複数の確率的な経路選択を効率よくかつ並列に行う必要がある．そこで，GPU 上で並列和と並列二分探索，さらに TSP の近傍探索を組み合わせることで，大きい問題でも高速に品質の高い解に到達する実装手法を実現した．CPU と比較を行ったところ，解の品質を落とさず最大 25 倍程度の高速になることを確認した．

Acceleration of Ant Colony Optimization for the Traveling Salesman Problem on GPU

Kazuta Kobashi[†]　and Akihiro Fujii[†]　and Yoshio Oyanagi[†]

Ant Colony optimization(ACO) is a meta-heuristic and a probabilistic technique for loving NP-hard Problem. The efficiency of the algorithm to solve TSP has already been shown in previous work. However ACO offers good quality of solution, it still needs considerable computational time and resources. Moreover, the structure of the algorithm makes it well suited for Parallelization, so in this paper we propose an implementation of ACO on GPU which has massively parallel architectures. On an implementation, in order to fully utilizing GPU architecture we used parallel reduction and parallel binary search for solution constructions and applied neighbor search for TSP to reduce utilization rate of memory resources. The experimental result shows the proposed method has outstanding performance and efficiency compared to serial CPU for large scale problems.

1. はじめに

巡回セールスマン問題(Traveling Salesman Problem : TSP)とは，組み合わせ最適化問題の一つで，あるセールスマンが N 箇所の都市を一度ずつ訪問し出発点に戻る場合に最短経路を求める問題である．この問題は都市数の規模が大きくなるにつれ探索領域が指数関数的に増大し，厳密解を求めることが困難になることでよく知られている．それに対して実時間内で準最適解を発見するために，焼きなまし法(Simulated Annealing : SA)やタブーサーチ(Tabu Search : TS)，遺伝的アルゴリズム(Genetic Algorithm : GA)といったメタヒューリスティックな手法がよく用いられる．

アントコロニー最適化法(Ant Colony Optimization)は蟻が巣に餌を持ち帰る行動をモデルにした最適化アルゴリズムである．ACO はフェロモンと呼ばれる探索領域評価値を用いた確率的解生成及び，解情報の蓄積が特徴である．

ACO は発見される準最適解の質に注目が集まり数々の研究がなされてきたが，メタヒューリスティック本来の目的は時間的制約のもと解を求めることにあるため，探索時間の短縮は解の質を高めることと同様に重要な課題である．そのため，近年では ACO に備わるマルチエージェント型で高い並列性のあるアルゴリズムから，マルチコアのハードウェアによる多くの高速化手法が提案されている．

OpenMP による ACO の並列化[1][2]では，CPU のコア数に応じた高い効率性が示され，SIMD 型プロセッサにおける ACO の実装[3]においては，0.5%程度の精度向上が示された．また近年注目されている GPU(Graphic Processing Unit)による並列化では，500 都市規模の問題で高い性能向上[4][5]に成功し，また，ACO におけるパラメータの探索[6]にも効果が示されている．

しかし，これら GPU における研究では，GPU の特徴を生かした並列化手法はとられておらず，また数千以上の規模の大きい問対への適用はまだ見られていない．そのため，本研究では ACO の TSP における GPU の効率的な手法として，巡回路生成における並列和と並列二分探索を提案する．また，限られたメモリ資源の下で解探索の効率性を高め大規模問題に適用するために，探索都市を限定した近傍探索用いた．結果として，CPU のシリアル処理による ACO と，GPU を用いた ACO の場合で都市規模を大きくするにつれ最大 25 倍の高速化が見られたことを示す．

[†] 工学院大学
Kogakuin University (Japan)

2. アントコロニー最適化法

自然界において，蟻ははじめランダムに地表を歩き回り餌を探す．そして，一度餌を見つけるとフェロモンと呼ばれる揮発性の物質を残しながら，巣に戻る．また，ほかの蟻は地表に残された．フェロモンを見つけると，ランダムな挙動を止めてフェロモンの後を辿り，餌を見つける．その際同様にフェロモンを分泌しながら巣に戻る．

フェロモンは揮発性であるため，時間とともに蒸発する．よって，多くの蟻が通る経路ほどフェロモンは散布され，また短い経路ほど移動時間が減るためフェロモンの蒸発も少なくなる．蟻が通らない経路にはフェロモンを分泌する機会が減り，最終的にはフェロモンはなくなる．この繰り返しによって，より短い道をすべての蟻が選択するようになる．

TSP における ACO では自然界の蟻と異なり同じ場所を訪問することはない. また，フェロモンは常に更新されるわけではなく，複数のエージェントが独立に巡回路を生成した後に更新する．以下のそのアルゴリズムを示す.

まず，各エージェントは式 2.1 の確率$p_{i,j}$にしたがって巡回路を生成する．$p_{i,j}$は都市 iにいる時，都市 j を選択する確率でありフェロモン値$\tau_{i,j}$より求まる．N_iは各エージェントが都市 i にいる時の未訪問都市の集合である．$\eta_{i,j}$はヒューリスティック値と呼ばれる評価値であり，式 2.2 に示すよう都市 i と都市 j 間の距離$d_{i,j}$の逆数をとる．これにより，距離の短い都市間ほど高い評価が得られる．α及びβはフェロモン値，ヒューリスティック値における強度のパラメータである．

$$p_{i,j} = \begin{cases} \dfrac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{j \in N_i}[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}} & if\ j \in N_i \\ 0 & else \end{cases} \quad \text{（式 2.1）}$$

$$\eta_{i,j} = \frac{1}{d_{i,j}} \quad \text{（式 2.2）}$$

全エージェントが解を生成したら，次にフェロモンの更新を行う．式 2.3，式 2.4 に従い，フェロモンの蒸発と散布を行う．ここでρは蒸発率を決定するパラメータである．フェロモンの散布量$\Delta\tau_{i,j}$は式 2.5 に従い最も短い巡回路T^{best}の巡回路長C^{best}の逆数をとため，良い解ほどその量は多くなる．

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j} \quad \text{（式 2.3）}$$

$$\tau_{i,j} \leftarrow \tau_{i,j} + \Delta\tau_{i,j} \quad \text{（式 2.4）}$$

$$\Delta\tau_{i,j} = \begin{cases} \frac{1}{C^{best}} & if\ (i,j) \in T^{best} \\ 0 & else \end{cases} \quad \text{（式 2.5）}$$

3. GPU における実装

3.1 GPU における開発環境

GPU の計算には NVIDIA 社が提供している C 言語の統合開発環境 CUDA(Compute Unified Device Architecture)を用いた．CUDA ではスレッドのまとまりをブロックと呼ぶ．プログラミングの対象はこのブロックを単位として記述され，ブロックは複数の演算コアを内蔵した Streaming Multiprocessor（SM）に割り当てられ実行される．ブロック間での同期処理はできないため，CUDA では問題をどのようにブロックに分割するかが問題となる．

3.2 CPU と GPU における処理の流れ

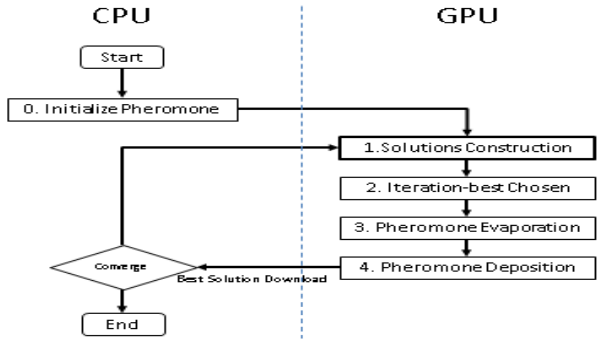


図 1　CPU の GPU での処理

本研究では，図 1 が示すように，ACO における各過程すべてを GPU 上で行った．CPU はイテレーションの制御および，イテレーション毎の最良解の転送のみを行う．ホストマシンのメモリと GPU 上のメモリは独立に存在するが，ホストと GPU 間のバンド幅は GPU 内部と比べると極めて低いため, 可能な限りメモリ転送をなくすことが

好ましい．そのため，たとえ並列性の低い処理でもメモリ転送なくして，GPU 上で演算を行った．

Step 1 では，複数のエージェントをブロックに割り当て巡回路を生成する．同時に，巡回路長も計算する．

Step 2 では，そのイテレーションでもっともコストの良い巡回選択する．最小コストは並列リダクションにより割り出す．

Step 3 では，フェロモン行列全要素に対して，ブロックに分割して蒸発処理を行う．

Step 4 では，step2 で計算したもっともスコアの良かった巡回路上にフェロモンの追加，強化を行う．

3.3 GPU でのエージェントとメモリの割り当て

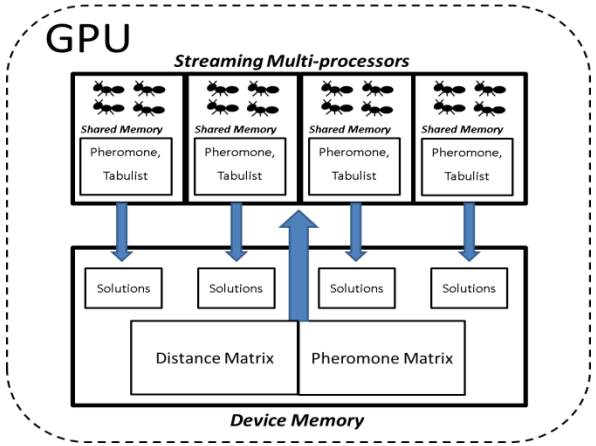


図 2　GPU におけるエージェントとメモリの割り当て

Step1 における巡回路生成時の GPU 上でのメモリの配置を図 2 に示す．

フェロモン情報，都市間の距離情報は全体で共通であり，そのイテレーション内では読み込みのみおこなうため，全体が共有できるグローバルメモリに配置した．各 SM 上では複数のエージェントが巡回路を生成する．SM 上にあるシェアードメモリは，ブロック内でスレッドが共有することができ，グローバルメモリと比較すると 100 倍以上アクセスがはやい．そのため，フェロモン情報のキャッシュとして利用する．TSP では同じ都市を二度訪問することはできないため，蟻毎に各都市の訪問歴を 1bit で TABULIST として保持する．生成された巡回路はグローバルメモリ上に書き込まれる．

3.4 巡回路生成方法

ACO ではフェロモンを重みとした都市選択に Roulette Wheel Selection アルゴリズムが用いられ，それは次の 3 ステップでから成る．

step 1. 重みの累計を求める．

step 2. 累計値を上限とした一様乱数を生成する．

step 3. その値を越えるまで再び重みを足し合わせることで，対応要素を求める．

しかし ACO において GPU で 1 エージェント毎に 1 スレッド割り当てて並列化し，上記手法を実装した場合，効率性が低くまた固有の問題が発生する[7].

step 1.まず，エージェント毎に累計値を求める際，それぞれ 1thread で行ったのでは，GPU の高い並列性が生かしきれない．GPU におけるマルチスレッドの効率化は，多くのスレッドを生成することで，レイテンシを隠すことにある．各蟻において，複数スレッドにより並列和を求めることで，GPU の特徴をいかした設計となる．

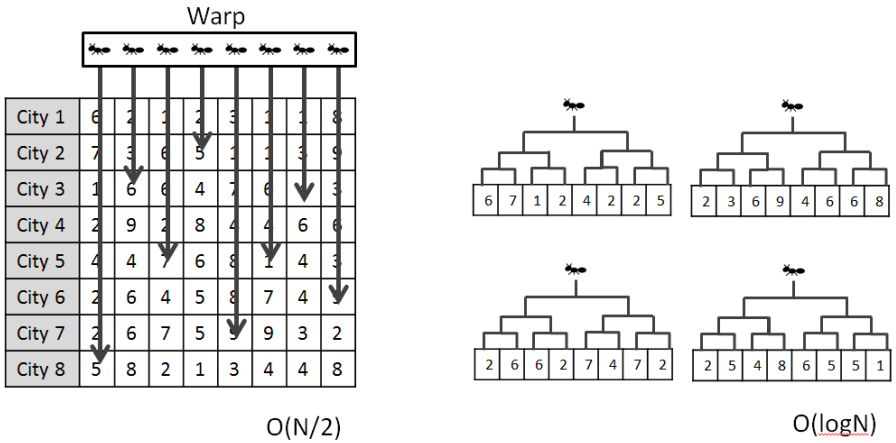


図 3　並列線形探索と並列に二分探索

次に問題となるとは，Step3 において各アリがいつ都市を決定するか不確定であるという点である．NVIDIA の GPU は SIMT(Single Instruction, Multiple Thread)アーキテクチャを採用しており，複数のスレッドに対して同様の命令が発行される．具体的に，

１つの SM 内において，Warp と呼ばれる 32 スレッドは SIMD 風に実行されるのでスレッド間での処理の流れが異なるプログラムはシリアライズされてしまい，結果として多くの待ち合わせ時間が発生してしまう．[8]さらに，あるスレッドが速い段階で選択都市が決定したとしても,次の処理へは 32 スレッドすべてが都市選択を終えるまで移行できないため，スレッドのまとまりはもっとも遅いスレッドに合わせないといけいない．同様のことは，データベースの GPU による検索の研究でもいわれている．[9]よって,このようなランダム性に起因する処理の流れの遅延を避ける方法が GPU では重要となるそこで本研究では（図 3）に示すように都市選択において二分探索を用いることで，上記問題を解決した．特に，Step1 の並列和を行う際に用いたデータをシェアードメモリに残しておくことで，それを利用して二分探索を行える．これを複数の蟻が同ブロック内で行い，並列二分探索を行い，分岐処理を削減した．

3.5 近傍探索

並列和と二分探索は GPU で重みのある要素を選択する場合には有効な手法と考えられる．しかし，この場合，要素の数が 2 ^ n の問題規模にしか適用できない．また，TSP に適用する場合，TSP 独自の性質も問題となる．以下に TSP におけるすでに最適解がわかっている問題の最適解において，各都市から見たとき何番目に近い都市と結びついているかを示した度数分布表を示す．縦軸が問題，横軸が各都市から見たときの近さである．

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	19	20	21	24
att48	39	20	13	11	5	2	2	2												
st70	56	32	21	10	9	6	3	1												
kroA100	86	44	29	21	9	6	1	1	1				1							1
tsp225	195	128	67	34	7	11	3	1	1	1										
pcb442	361	277	143	66	6	13	2	5	4	3	1						1			
pr2392	2056	1561	621	251	140	64	27	11	15	7	9	2	4	4	1	1		3	5	

表 4　最適解における結びついている都市の近傍レベル

（表 4）からわかるように都市数が 2000 以上である問題でさえ，最適解は近傍 20 都市ほどの経路から成っている．このため，TSP において全都市への経路探索は無駄が多く，解の収束速度を遅くしてしまう．本研究では以上の問題から，問題サイズによりある都市からみて距離の近いものを優先的に選択する Candidate Set(CS)を利用した[10].

CS では探索範囲を近傍 M 都市に限定することで，無駄の少ない高速な解探索を可能にする．もし，近傍 M 都市すべてが訪問済みであった場合は，近傍 2M 都市，近傍 3M 都市と段階的に探索範囲を広げる．

また，さらなる利点として大規模な問題への適用があげられる．並列和を行う際フェロモンを一度シェアードメモリにコピーするが，このシェアードメモリは非常に容量が少ない．しかし，CS の利用は，問題規模を事実上 2^N から 2^M に制限するため，必要なメモリは探索する近傍のレベルによって決まり，資源の乏しい環境に適した方法であるといえる．

4. 評価実験

4.1 実行環境

パラメーター	rho 0.5, alpha 1.0, beta 5.0:	
ハードウェア	cpu	: 2x Intel Xeon X5570(2.93GHz)
	memory	: 12GB (6x 2GB DDR3-1333 DIMM)
	GPU	: Tesla C2050(1.15GHz,3GB,14 SMs,ECC ON)
OS		: CentOS 5.5(64bit)
ドライバ		:CUDA DRIVER 3.20
コンパイラ		: nvcc release 3.2, V0.2.1221

4.1 探索する近傍のレベルによる解の比較

まず，探索する近傍の範囲と解の比較を行った．エージェントの数は 32，近傍範囲 10,20,35,100 におけるイテレーション毎（横軸）の巡回路長（縦軸）の推移を図 5 に示す．問題は TSPLIB[11]の都市数 2392 の pr2392 を用い，またこの問題は表 4 が示すように，最適解は近傍 21 都市から成る．

近傍範囲を 10 都市に限定した場合,かなり早いイテレーションの段階で良いスコアを出している．しかし，解の更新頻度がもっともすくなく 10000 イテレーション時のスコアは低い.20 都市の場合は 10 都市と比べると最終スコアが良い.35 都市の場合，速い段階ではあまり良いスコアではない．しかし，解の更新頻度は高く最終スコアも最もよい．近傍範囲を 100 以上に広げた場合は解の収束が遅くなる一方であった．

結果として，近傍範囲を広くするとその分解の多様性は広がり更新頻度は高くなるが収束が遅くなるすぎるが，逆に範囲を狭くしすぎると，すぐ収束して良い解がでるが，多様性は低く最終的な解精度は低いことが分かった．よって数千レベルの問題に対しては以降，近傍レベルは 32 に固定して評価する．

Vol.2011-HPC-129 No.19
2011/3/16

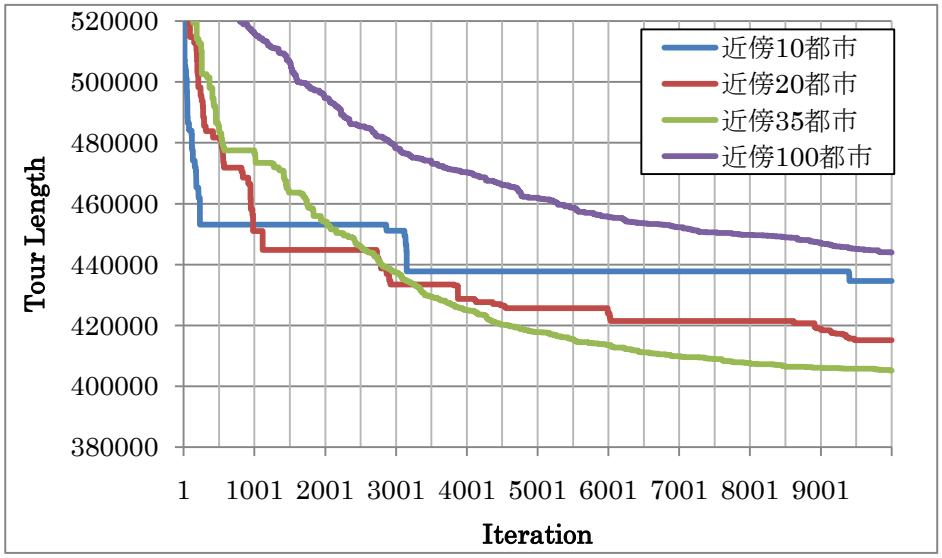


図 5　探索する近傍のレベルによる解の推移

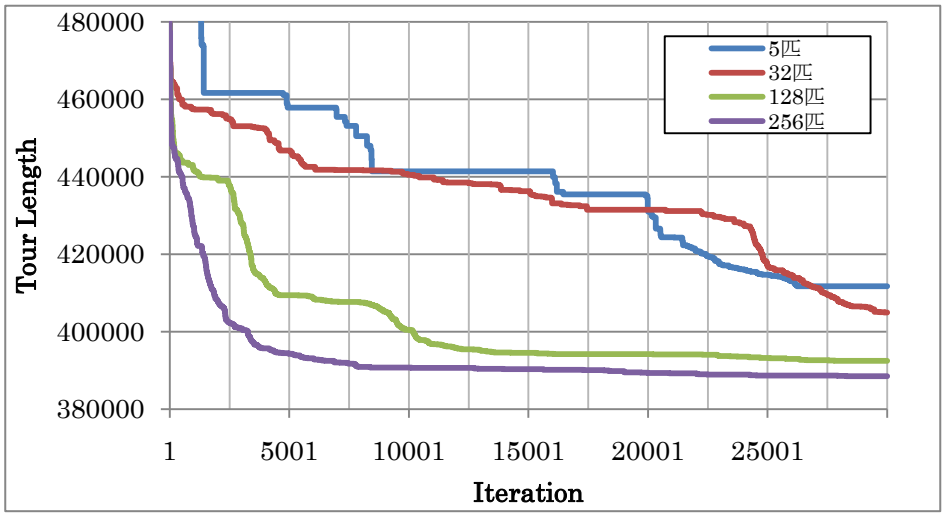


図 6　蟻の数と解の比較

4.2 蟻の数と解の比較

次に，エージェント数と解の比較を行った．適用問題は同様に pr2392 である．

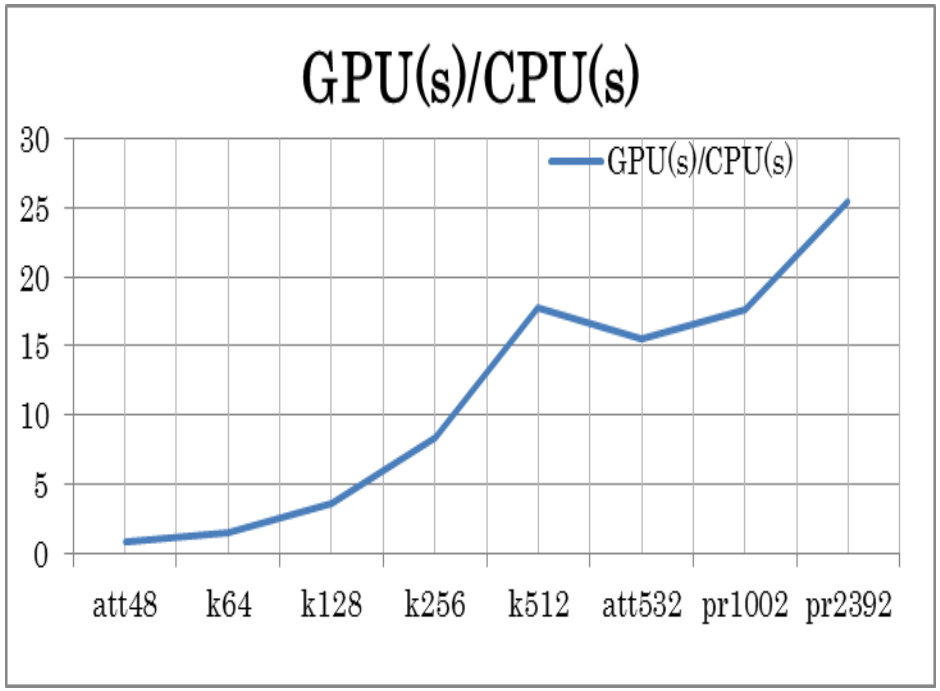
図 6 が示すように，エージェントの数を増やすほど，初期段階の解の質，解の更新頻度，最終解のスコアいずれもよくなることがわかった．特に，蟻の数が 1 匹の場合と比べると 85%のほど経路長の少ない巡回を生成できた．

問題サイズが 500 以下の場合, 蟻の数による解の質に大きな変化は見られなかった．しかし，問題規模を広げるほど，エージェント数による解の質の向上は顕著に見られた．

4.3 CPU との高速化倍率

最後に CPU に対する GPU の速度向上率を比較した．問題は TSPLIB の，att48,att532,pr1002,pr2392 及び, 比較のために乱数で生成した問題, k64,k128k,k256,k512 である．それぞれ数字部分が都市数である．エージェント数を都市数と同様にしたときの 1000 イテレーション時の実行時間を，CPU の場合と GPU を利用した場合で表に示す．また，GPU による速度向上率も図 7 に示す．

表からわかるように問題規模を大きくするほど，より高い速度向上率が見られた．問題規模が小さい時に一部 CPU のほうが高速な場合もあるが, これはエージェントの数少ない時 GPU による並列化度が低く, 性能を引き出しきれていないことが原因にある．



	CPU(s)	GPU(s)	GPU(s)/CPU(s)
att48	0.55	0.67	0.82
k64	1.02	0.68	1.50
k128	4.36	1.20	3.64
k256	19.7	2.36	8.36
k512	91.77	5.16	17.78
att532	90.39	5.80	15.58
pr1002	363.91	20.65	17.62
pr2392	2313.5	91.03	25.41

図 7　CPU に対する GPU を利用した場合の高速化倍率

5. まとめ

本研究では，TSP における ACO の実装において，GPU の特徴を生かした並列和及び並列二分探索を提案した．また，検証により，扱う問題規模が大きくなるほど，エージェント数が解の質に大きな影響をもたらすことがわかった．そして，エージェント数を増やすほど GPU による並列性と効率性は高まり，結果 CPU と比較して最大 25 倍の高速化が見られた．

数＋万規模の問題に対して了解を効率よく生成するにはエージェントの数がより必要であり，GPU の持つ高い並列性はより重要な意味を持つ．よって今後はマルチ GPU による大規模問題への高速な良解探索へ取り組みたい．

参考文献

1)Pierre DelisleKrajecki,Marc Gravel,Caroline GagneMichael. PARALLEL IMPLEMENTATION OF AN ANT COLONY OPTIMIZATION METAHEURISTIC WITH OPENMP.
2) Thang N.buiNguyen,Joseph R. Rizzo Jr.ThanhVu. Parallel Shared Memory Strategies for Ant-Based Optimization Algorithms.
3) 中野 光臣大樹,飯田 全広,末吉 敏則尾崎. (2008). アントコロニー最適化法の並列化手法および超並列 SIMD 型プロセッサへの実装.
4) YouYing-Shiuan. Parallel ant System for Traveling Salesman Problem on GPUs.
5) Wang JieningChunfeng,Dong JiankangZhang. (2009). Inplementation of Ant Colony Algorithm based on GPU.
6) Hongtao BaiOuYang,Ximing Li,Lili He,Hihoug YuDantong. (2009). MAX-MIN Ant System on GPU with CUDA.
7) Harris, M. (2008). Optimizing CUDA. Performance Computing. NVIDIA Corporation
8) 筒井茂義，藤本典幸. GPU 計算を用いた並列深化計算による二次割り当て問題の一解法とその解析.
9) Tim Kaldeway.jeff HagenDi Blas,Eric SedlarAndrea. Prallel Search On Video Cards.
10) 亀田陽介. (2007). 局所最適解をフェロモンの初期配置に利用したアントコロニー最適化法による TPS の解法.
11) TSPLIB : http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/