Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 7, Number 1, pages 20–27, January 2018

GPU-accelerated Ant Colony Optimization for the Traveling Salesman Problem

Ryouhei Murooka, Yasuaki Ito, and Koji Nakano
Department of Information Enginnering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, Hiroshima, 739-8527 Japan
Email: {murooka, yasuaki, nakano}@cs.hiroshima-u.ac.jp

***Abstract*—The main contribution of this paper is to show a GPU implementation of ant colony optimization for the traveling salesman problem. In the ant colony optimization, many ants are deployed to cities and each ant independently visits cities one by one. The trace of an ant corresponds to one of the solution of the traveling salesman problem. The idea of the proposed GPU implementation is to adopt a hybrid technique, stochastic acceptance and tournament selection, stochastically to choose the next visiting cities. Also, to accelerate the computation, we have considered the characteristic of the GPU architecture, such as coalesced access of the global memory and the bank conflict of the shared memory, among others. We have implemented the proposed method on the NVIDIA GeForce GTX 1080. The experimental results show that our proposed GPU implementation for 1002 cities runs in 2065 milliseconds, while the sequential CPU implementation runs 98983 milliseconds. Thus, our GPU implementation attains a speed-up factor of 47.92.**

***Keywords*-GPGPU; CUDA; Traveling Salesman Problem; Ant colony optimization**

I. はじめに

GPU(Graphic Processing Unit) とは，コンピュータのグラフィック処理に特化したデバイスである．内部に存在している複数のコアを使用することにより，大量のタスクを並列に処理することを可能としている．CUDA(Compute Unified Device Architecture) [1] は GPU を汎用的な並列処理のために利用する GPGPU(General-purpose computing on GPU) のための統合開発環境である．CUDA を用いることで，GPU 実装のための並列アルゴリズムの開発を行うことができる [2]–[5].

蟻コロニー最適化とは組み合わせ最適化問題を解くために考案されたメタヒューリスティックな手法となっている．蟻コロニー最適化は実世界の蟻が巣と餌の間の経路を探索するの動きを参考にしている [6]．蟻は食べ物を探す際，最初はランダムに探索を行い，食べ物を見つけたら餌から巣までの経路にフェロモンを落としながら戻る．他の蟻がそのフェロモンを見つけると，ランダムな探索を止めてその経路をたどることにより餌を発見する．そして他の蟻たちがまたフェロモンを補強しながら巣に戻ることによってフェロモンをより強固なものとする．しかしフェロモンは時間とともに蒸発をして跡をたどるのが難しくなる．その時の餌までの距離が長ければ長いほどフェロモンは蒸発をしやすい．したがって，餌までの距離がより短いほどフェロモンが蒸発しづらくなり，蟻が短い距離で餌にたどり着けるようになる．蟻コロニー最適化アルゴリズムは以下の２ステップで構成されている．

Step1 初期化処理
- フェロモン経路の初期化

Step2 繰り返し処理
- それぞれの蟻が一定回数に達するまで繰り返し処理を行う
  - — フェロモン経路を参考に解の構成を行う
  - — フェロモン経路の更新

最初のステップでは，主にフェロモン経路の初期化を行う．次の繰り返し処理のステップでは，確率的状態遷移規則に従って TSP の解を得る．この規則はフェロモン値の量に依存する規則となっている．一度すべての蟻が経路を構成したのち，フェロモン値の量を更新するためにフェロモン値を半減させる蒸発処理と蟻が得た解をもとにフェロモン値を増やす加算処理の２つの処理を行う．これらの処理は指定した回数処理を行うまで繰り返される．

本研究では巡回セールスマン問題 (TSP) に蟻コロニー最適化を適用した最適化アルゴリズムの GPU 実装を提案する．巡回セールスマン問題とは，n 都市が入力として与えられたとき，それぞれの都市を一度だけ訪れる経路の内，最短の経路を見つける問題となっている．提案 GPU 実装では，グローバルメモリのコアレスドアクセスや，シェアードメモリのバンクコンフリクトなどを考慮した実装となっている．本研究の目的は蟻コロニー最適化アルゴリズムの GPU 実装による計算の高速化が主となっており，解精度をの向上ではない．そのため，基本的な巡回セールスマン問題に対する蟻コロニー最適化実装と同等の解を得る．本実装では GPU に NVIDIA GeForce GTX 1080 を使用し，結果として，CPU 実装 [5] と比較して，都市数 1002 のとき，約 48 倍の高速化を達成．既存 GPU 実装と比較して都市数 442 のとき約 2.69 倍を達成した．

本論文は次のような章構成となっている．2 章では巡回セールスマン問題に対する蟻コロニー最適化アルゴリズムについて説明する．3 章では GPU と CUDA アーキテクチャについて説明する．4 章では巡回セールスマン問題に対する蟻コロニー最適化アルゴリズムの GPU 実装について説明する．5 章では実験結果について示している．最後に 6 章で結論を述べている．

II. 巡回セールスマン問題に対する蟻コロニー最適化

本章では蟻コロニー最適化を用いた巡回セールスマン問題のアルゴリズムを説明する．巡回セールスマン問題はセールスマンが n 都市をそれぞれ一度だけ訪れる経路

の内，もっとも短い経路を見つける問題である．巡回セールスマン問題に対する蟻コロニー最適化では，蟻をセールスマンの代わりに見立てて経路の探索を行う．それぞれの蟻がそれぞれの都市を一度だけ訪問し，最後の都市を訪れた後，最初の都市に戻ることで得られる経路が解となる．それぞれの蟻は以下の特性を持っている．

- 蟻が次に訪れる都市を選ぶ際，蟻が現在いる都市と各都市との距離の逆数，各都市間のフェロモン値の 2 つで定められる規則を用いる．
- すでに訪れた都市は再度選ばれないようにするため訪問済みリストに追加しておく．
- 蟻の巡回が終了した後，蟻の巡回ルート上にフェロモンを残す．

蟻コロニー最適化 (ACO) とは最適化問題を解く手法の一つで Colorni，Dorigo，Maniezzo によって提案された手法である [7], [8]．この手法は巡回セールスマン問題 (TSP) で最適解を得るために考えられた手法で，蟻の性質を基に考えられている．

ACO はこの蟻の振る舞いに基づいたアルゴリズムとなっている．ACO は以下の 3 ステップに分かれている．(1) 初期化，(2) 経路構成，(3) フェロモン値の更新．まず始めに初期化処理を行いその後，経路構成とフェロモン値の更新を指定した回数繰り返し行う．入力として n 都市間のそれぞれの距離，m 匹の蟻，が与えられたときの各ステップの詳細について次から説明していく．

A. 初期化処理

初めのステップでは初期化処理を行う．初期化処理では以下のような貪欲法を用いて得た結果を使いフェロモン値の初期値を決定する．

$$\tau(i,j) = \frac{n}{C_g} \quad \forall (i,j) \in L \tag{1}$$

L はすべての都市間の辺，C_g は貪欲法で得た経路の長さとなっている．貪欲法とは任意の都市から次に訪問する都市を決定するとき，まだ訪れていない都市の内，一番近い都市を訪問するといった条件のもとすべての都市を巡回する手法となっている．2 つの都市間のフェロモン値の初期値は C_g の平均の逆数で与えられる．

B. 経路構成

経路構成では m 匹の蟻がそれぞれ独立に都市を訪れる．それぞれの蟻が初めにスタートする都市と次に訪れる都市は毎回ランダムに決定される．蟻が都市 i から都市 j に移動する確率は以下の式で計算される．

$$p_k(i,j) = \begin{cases} \frac{f(i,j)}{\sum_{l \in N_k(i)} f(i,l)} & \text{if } j \in N_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$N_k(i)$ は k 番目の蟻が都市 i にいるとき，まだ訪れていない都市の集合となっている．$f(i,j)$ は都市 i,j 間の適応値となっており，以下の式で計算される．

$$f(i,j) = [\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta} \tag{3}$$

$\tau(i,j)$ は都市 i,j 間のフェロモン値となっている．$\eta(i,j)$ はヒューリスティックな情報である都市 i,j 間の距離の逆数となってる．α と β はフェロモン値と距離のうち，どちらの影響を大きくするかという調整するための重みとなっている．つまり都市 i,j 間のフェロモン値が高い場合や都市間の距離が短い場合は都市 j を訪れる確率が高くなる．この確率を用いてそれぞれの蟻が独立に次に訪問する都市を決定し，最後の都市を訪問したのちに最初の都市に戻ることで巡回するルートを構成する．

C. フェロモン更新

すべての蟻が経路構成を行った後，それぞれの経路情報をもとに各都市間のフェロモン値の更新を行う．フェロモン更新はフェロモン蒸発処理とフェロモン加算処理の 2 つの処理から構成されている．

フェロモン蒸発は蟻の構成する経路が局所的にならないために行う．フェロモン蒸発は以下の式で計算される．

$$\tau(i,j) \leftarrow (1-\rho)\tau(i,j) \quad \forall (i,j) \in L \tag{4}$$

ρ はフェロモンの蒸発率となっている．

フェロモン蒸発処理を行った後，フェロモン加算処理を行う．フェロモン加算処理では各蟻が構成した経路をもとに都市間のフェロモン値に値を加算する．

$$\tau(i,j) \leftarrow \tau(i,j) + \sum_{k=1}^{m} \Delta\tau_k(i,j) \quad \forall (i,j) \in L \tag{5}$$

$\Delta\tau_k(i,j)$ は k 番目の蟻が都市 i,j 間に加算するフェロモン値となっている．この値は次の式で計算される．

$$\Delta\tau_k(i,j) \begin{cases} \frac{1}{C_k} & \text{if } e_{i,j} \in T_k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

C_k は k 番目の蟻が構成した経路の長さ，$e_{i,j}$ は都市 i,j を結ぶ辺，T_k は k 番目の蟻が構成した経路となっている．この式から得られる値は経路の長さが短いほど値が大きくなる．つまり，蟻が構成した経路の長さが短いほど加算されるフェロモン値は大きくなる．

一般的にはフェロモンの初期値はすべて同じ値にされる．そこで，有用と思われるフェロモン値とそれ以外のフェロモン値に偏りを持たせる手法が提案されている [9]–[11]．これらの手法は貪欲法に基づいているため，TSP 以外の組み合わせ最適化問題にも適用することが可能である．

ACO にはさまざまな種類がある．Ant System(AS),Max-Min Ant System(MMAS),Ant Colony System(ACS) が一般的に使われる ACO となっている．AS は最初に提案された ACO である [7], [8]．特徴としては，全ての蟻が経路の構成を終えた後にフェロモン更新を行う [12]．大きな違いとしてはフェロモン値の上限と下限が最初に決まっており，複数の蟻の中で，最も良い解を得た蟻のみがフェロモンの更新を行えるといった点である [13]．ACS では AS で行うフェロモン更新に加えてローカルのフェロモン更新を行っている．ローカルのフェロモン更新とは蟻が経路を更新している間に行われる更新で，AS とは違い，1 匹 1 匹が解を構成するたびにフェロモン値が更新される．本研究では AS を用いて巡回セールスマン問題の解を求める．

III. GPU と CUDA

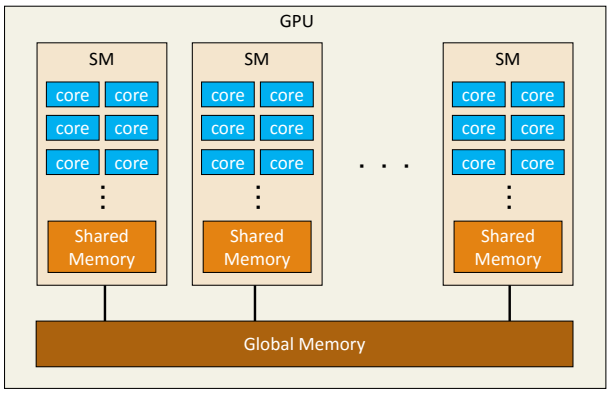

Figure 1. GPU アーキテクチャ

GPU 内には多数の Streaming Multiprocessor(SM) と呼ばれる演算ユニットと複数のメモリから構成されている. 実際にプログラミングを行う際は各メモリの特徴を考慮し使い分ける必要がある. Figure 1 は GPU ハードウェアの概略図である. GPU にはグローバルメモリとシェアードメモリの 2 種類のメモリがある. グローバルメモリは GPU 内で最も大きな容量を持ち, GPU 内にある全ての core からアクセスが可能なメモリである. しかし, レイテンシが非常に大きいためアクセスに時間がかかってしまうことがデメリットである. また, Figure 2 に示すようグローバルメモリへのアクセスした場合, 離散したアドレスへのアクセス (ストライドアクセス) となってしまうため, 大きな遅延が生じてしまう. そこで Figure 3 で示すようにグローバルメモリへのアクセスを行った場合連続したアドレスへのアクセス (コアレスドアクセス) となり同一ワープ内の 32 スレッドが 1 度にデータを取得することができる. したがって, GPU で高速化を行う際にはグローバルメモリへのアクセス回数を抑えたり, アクセスの方法に注意する必要がある.

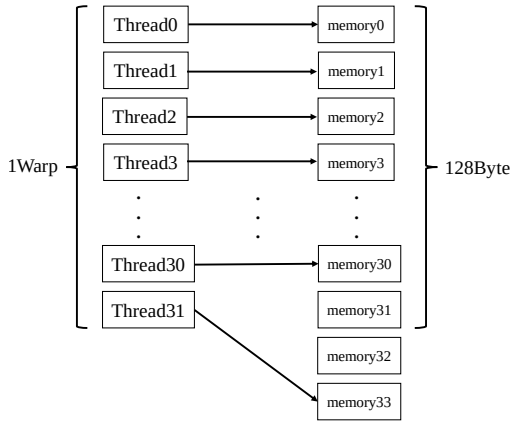

Figure 2. stride access

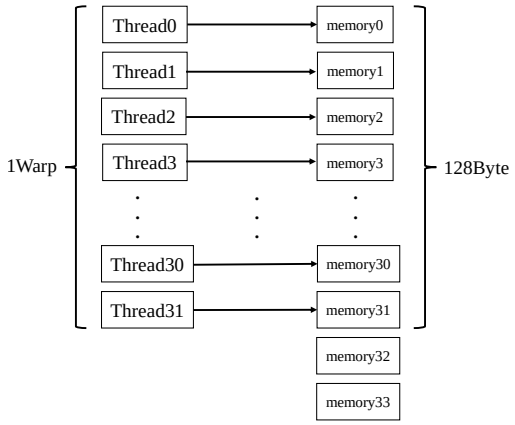

Figure 3. coslesced access

シェアードメモリとはそれぞれの SM 内に独立して存在するメモリで, 容量はグローバルメモリに比べると非常に少なく, 同一 SM 内にある core のみしかアクセスできない. しかし, グローバルメモリよりも高速にアクセスが可能となっている. そのため, アクセス回数の多いデータをシェアードメモリ上に置くことによって更なる高速化を生み出す可能性がある. シェアードメモリは 32 個のバンクに分けられており, シェアードメモリ上に配列が確保される場合, 各要素が各バンクに順に確保される. シェアードメモリにアクセスする際にワープ内の各スレッドのアクセス先が同一バンクの別アドレスとなってしまったときにバンクコンフリクトと呼ばれるアクセス要求の競合が発生する. バンクコンフリクトが発生すると再度アクセス要求を行う必要があるため同一ワープの他のスレッドのアクセス終了を待つ必要がある. Figure 4 と Figure 5 にバンクコンフリクトが発生する場合と発生しない場合を示す.

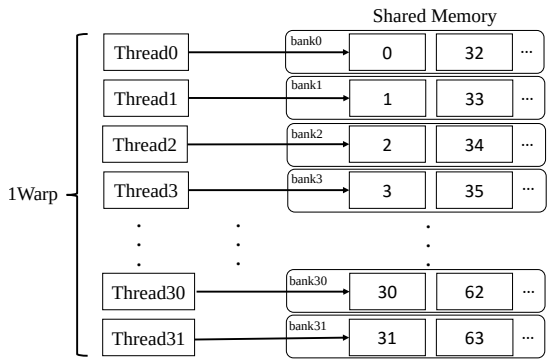

Figure 4. バンクコンフリクトが発生する例

CUDA(Compute United Device Architecture) は 2006 年に NVIDA 社が発表した, GPGPU 向けの統合開発環境である. CUDA 発表以前で GPGPU を実現するにはグラフィック向けの言語である言語を利用するしかなかった. したがって汎用的な演算を行うためにはグラフィックに関する

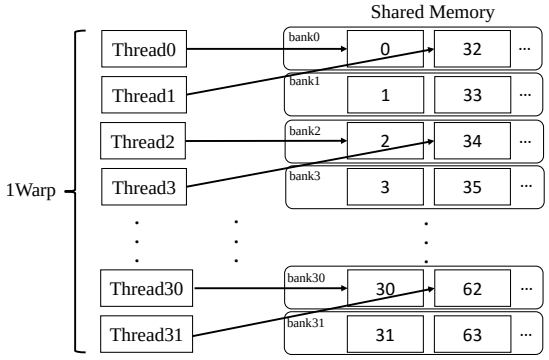

Figure 5. バンクコンフリクトが発生しない例

知識が必要となる. CUDA ではグラフィックの知識が必要なくとも GPGPU を実現できるよう, C 言語や Fortran を拡張したような言語となっているためそれらの言語の知識をもった人なら比較的簡単に扱えるようになった.

CUDA は通常, 1 つのプログラムを, 起動したすべてのスレッドに与えて動作を実行する. スレッドというのはカーネルを動作させる最小の単位となっており, カーネルとは各スレッドが実行するプログラムのことである. CUDA は階層的にこのスレッドを管理しており, スレッドの集合をブロック, ブロックの集合をグリッドと定義している.

GPU の処理は, ワープと呼ばれる 32 スレッド毎の集合が同時に動作している．同一のワープ内では同じ命令が実行されるため, 条件分岐によってワープ内のスレッドが異なる命令を実行した場合，逐次的に処理がされる. よって, ワープダイバージェンスが発生した場合は動かないスレッドが発生するため処理が逐次化されて，実行時間の増加の原因となってしまう．

IV. GPU 実装

本章では前章で説明した蟻コロニーアルゴリズムの GPU 実装の方法を述べる．蟻コロニー最適化アルゴリズムは複数の蟻が独立で解を取得し，取得した解を基にフェロモン更新を行うため，GPU による並列処理に適したアルゴリズムである．経路構成では都市数 n に対してランダムにばら撒かれた m 匹の蟻がそれぞれ独立に経路構成を行い解を取得する．提案実装では 1 ブロック 32 スレッドで起動し，1 ブロックに複数の蟻を割り当てて並列化を行う．1 ブロックに割り当てる蟻の数は入力で与えられる都市数に応じて変更することでメモリを最大限活用する．各蟻のパターン構成は完全に独立しているため以降の説明では 1 匹の蟻に注目して説明を行っていく．ACO では適応度に比例する確率でルーレット選択を行い，次の都市を選んでいき経路を構成する．しかし，ルーレット選択は計算時間のかかる手法となっている．今回の提案実装では厳密な確率選択であるルーレット選択と厳密でない確率選択である Tournament select 法を組み合わせることで従来の手法より高速化を行っている．

A. 初期化

入力として n 都市が与えられたとき，都市 i の座標情報である (x_i, y_i) がそれぞれ得られる．この入力から式 (1) を用いてフェロモン値の初期値を計算する．計算で得られたフェロモン値と各都市間の距離は $n \times n$ の二次元配列でグローバルメモリに保存される．

B. 経路構成

蟻が次に訪れる都市を選ぶ際，式 (4) で得られる適応度に沿った確率をもとに選択を行っている．今回の提案実装では厳密な確率による選択と厳密ではない確率の選択を用いて経路の選択を行っている．厳密な確率による選択は論文 [5] による Stochastic acceptance を基にした手法を実装する．

Stochastic acceptance を基にしたルーレット選択

Stochastic acceptance を基にしたルーレット選択は文献 [5] で提案されている手法となっており，都市 i にいる蟻が Stochastic acceptance を基にしたルーレット選択によって次に訪問する都市を選択するアルゴリズムを示す.

- **Step1** 適応度 $\tau(i,j)(0 \leq i \leq n-1)(0 \leq j \leq n-1)$ のなかからもっとも適応度の高い値を τ_{max} として保存する
- **Step2** [0,$n-1$] の範囲で乱数 r を生成する
- **Step3** $\frac{\tau(i,r)}{\tau_{max}}$ の確率で都市 r を訪問する．訪問する都市が決まらなかった場合，Step2 に戻って訪問する都市が決まるまで繰り返す.
- **Step4** まだ他に未訪問都市が残っている場合，Step1 に戻る．すべての都市を訪問したら経路構成を終了する.

この選択の概要を図 (6) に示す．Step2 では等しく f_{max} で区切られた区間の内で 1 つの区間が選ばれるようになっており，この 1 つの区間が 1 つの頂点に対応している．Step3 では選択した区間の中から $f(i,r)$ に比例した確率で都市が選ばれるようになっている.

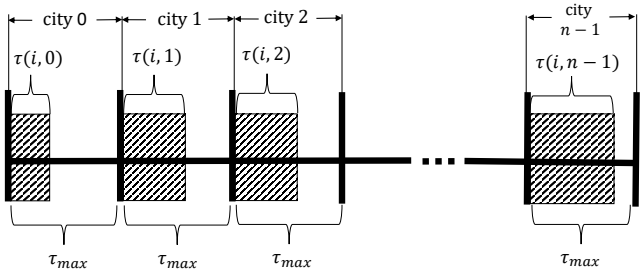

Figure 6. Stochastic acceptance を基にしたルーレット選択の概要

この手法はフェロモン値が最大の値，つまり τ_{max} が既知であった場合，頂都市選択が $O(1)$ で済むという特徴を持っている．また，Step2 以降の処理は複数スレッドで同時に乱数を生成することによって並列に処理を行うことが可能となっている．この手法でフェロモン値に比例し

た確率選択が行えることは文献 [14] によって証明されている．

Tornamet select

この手法は文献 [15] で提案された手法で厳密に適応値に則した都市選択ではない．その代わり厳密な確率選択よりも計算量が少ない手法となる．この手法による都市選択の手順をを次に説明する．

Step1 未訪問都市からランダムに m 個の都市を選択する．

Step2 ランダムに選ばれた m 個の都市の中から適応値 f に則した確率で次に訪問する都市を選択する．

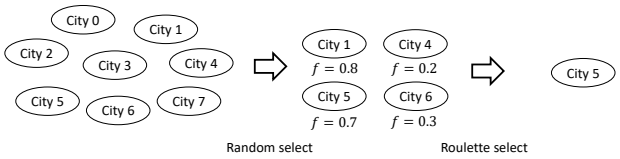

Figure 7. Tournament select

Figure 7 では 8 つの都市に対して Tournament select を行った結果となっている．まず 4 つランダムに都市を選んだ後に適応値に則した確率で訪問する都市を選択する．ルーレット選択によって訪問都市が選ばれるため，適応値が最大な都市が選ばれるわけではなく，4 つのうちすべての都市に訪問する可能性がある．Tournament select を GPU で実装する場合，1 ブロックの起動スレッド数は 1 ワープとする．そしてランダムに都市を選択する数 m でスレッドを割り当て，m スレッドが 1 蟻として経路を構成する．図 Tournament を例にすると $m = 4$ となるため，1 ブロック当たり 8 匹の蟻が経路を構成することとなる．この手法では 1 蟻に対してまず都市選択リストとなるサイズが都市数 n のリストをシェアードメモリに確保する必要がある．蟻はこのリストを用いてランダムに都市を選択する．このリストは初めは都市の番号が保存されたリストとなっている．都市が選択されたらこのリストを更新する必要がある．次に各スレッドがリストを参照し，各蟻の訪問可能な都市の中からランダムに都市を選択する．このとき，ランダムに都市を決定するため選択した都市が被る場合がある．今回は都市が被ってしまって場合もよいものとして選択をおこなっていく．そして各スレッドの選んだ都市の適応値がシェアードメモリに書き込まれ，この値に則した確率で次に訪問する都市が選択される．この時のシェアードメモリは $\frac{32}{m} \times (m+1)$ となっている．これは適応値を基に確率選択を行う際に $m+1$ の倍数番目のメモリには 0 を入れるためである．訪問都市を決定したのち，訪問リストを更新する必要がある．訪問リストは先ほど訪問先に決定した都市の入っているメモリの値ををリストの一番最後の都市番号で上書きすることによって行われる．Figure 8 は蟻が都市 4 に訪れたときのリストの更新を示したものである．この手法を用いることによって一度の処理によってリストの更新ができ，都市を選択する際は残っている都市のリストのサイズで乱数を生成することによって都市を選択することができる．この Tournament select は蟻が構成する経路がよりよいものになりにくいため，より多くの蟻を使用するか，フェロモン更新を増やす必要がある．そのため，厳密な確率選択と比較して大きく高速化できるわけではない．そこで厳密な確率選択と厳密ではない確率選択を組み合わせたハイブリット法を次に提案する．

Index	1	2	3	4	...	91	92	93	94
City number	1	2	3	4	...	91	92	93	94

If ant visit city_4

Index	1	2	3	4	...	91	92	93
City number	1	2	3	94	...	91	92	93

Figure 8. city list

ハイブリット法

Stochastic acceptance を基にしたルーレット選択では訪問済みの都市が多くなってきた場合, 次に訪問する都市を選ぶ確率が非常に低くなってしまう. そのため，経路構成の後半に計算時間がかかる手法となってしまう．対して，Tournament select はすべての未訪問都市のなかから訪問候補の都市をランダムに選ぶため，経路構成の前半では適応度の低い都市が選ばれる可能性が高くなる．この 2 つの手法のデメリットを補う手法がハイブリット手法となっている．経路構成の前半で厳密な確率による都市選択をおこない，適応度の高い都市を選択を行う．経路構成の後半では既に未訪問都市が少ないため，厳密な確率ではない都市選択を行ってもあまり影響はない．このように経路を構成することによって蟻の数やフェロモン更新の回数をあまり増やさずに近似解を得ることができる．ハイブリッド法は Tornement select 同様に 1 ブロックにつき 1 ワープ起動をおこない，1 ブロック内で複数の蟻が経路を構成することとなる．この手法で重要なのは選択手法を切り替えるタイミングである．今回実装では 1 ワープに複数蟻が存在しているためそれぞれが別の手法で経路構成をしてしまうとワープダイバージェンスが発生してしまう．そのためすべての 1 ワープ内の蟻が同時に手法を切り替えることができるようにこちらで事前に切り替えるタイミングを指定する必要がある．本実装では性能評価に用いる各都市に対していくつかの切り替えるパターンで実験を行い，もっともよかったタイミングを用いている．

C. フェロモン更新

次にフェロモン更新の GPU 実装について説明する．前章で説明したように，フェロモン更新はフェロモン蒸発とフェロモン加算から構成される．フェロモン更新では蟻の経路構成で生成した解を基にフェロモンを更新することになる．今回の実装ではフェロモン値 $\tau(i,j)(0 \leq i \leq j \leq n-1)$ がグローバルメモリに $\tau(i,j) = \tau(j,i)$ となるように 2 次元でシンメトリックに保存されている．シンメトリック行列で保存されているため，都市 i に関連するフェロモン値である $\tau(i,0), \tau(i,1), ..., \tau(i,n-1)$ は同一の

行に保存されている．そのため，コアレスドアクセスを行うことができる．本実装はフェロモン更新カーネルと対称化カーネルで構成されている．

1) フェロモン更新カーネル：このカーネルは n ブロックと複数ブロックで構成されている．そのため，1 ブロックが 1 都市のフェロモン更新を担当することになる．例えば，i 番目のブロックがアクセスするフェロモン行列は $\tau(i,0), \tau(i,1), ..., \tau(i,n-1)$ となるため，コアレスドアクセスとなる．初めに，i 番目のブロックがアクセスするメモリをシェアードメモリにコアレスドアクセスで保存する．シェアードメモリに格納する際，もとのフェロモン値に蒸発率を乗算した値を格納することによって蒸発処理を済ませることができる．更に後の対象化カーネルのために値を半分にしておく．次に各スレッドが各蟻の構成した経路の長さをグローバルメモリから読み出し，式 (5) を用いることで並列に加算処理を行う．しかし，この手法だと $\tau(i,j)$ と $\tau(j,i)$ の両方に加算を行うことができない．そのため，対称化カーネルを用いてフェロモン値を正しくする処理を行う．

2) 対称化カーネル：このカーネルではフェロモン更新カーネルで得た結果を正しくする処理である．フェロモン更新カーネルで加算される $\tau(i,j)$ とこのカーネルで加算する必要のある $\tau(j,i)$ は転置の関係にある．そのため，フェロモン更新カーネルの結果にその結果を転置したデータを加算すれば $\tau(j,i)$ に加算を行うことができる．しかしそのまま加算すると更新前のフェロモン値が余分に加算されてしまうため，フェロモン更新カーネルでフェロモン更新カーネルでシェアードメモリから値を読み出す際に値を半分にしている．このカーネルではグローバルメモリアクセスをコアレッシングにするために 2 次元で保存されているフェロモン値を 32×32 の部分配列に分割して保存するこのとき，2 つの対象な部分配列か 1 つの対象な領域を含む部分配列を 1 ブロックに割り当てる．ブロック毎に割り当てられた部分配列をグローバルメモリから読み出してシェアードメモリに転置して保存する．このことを説明したものが Figure 9 となっている部分行列を転置する保存する際，実際には x 方向に 1 要素パディングして大きさが 33×32 のシェアードメモリを用いる．このようにすることによってグローバルメモリに書き戻す際コアレスドアクセスを行いバンクコンフリクトを回避することができる．そして転置したシェアードメモリ領域の値を Figure 10 のようにグローバルメモリ上の片方の部分配列に加算することで対象化カーネルは終了する．

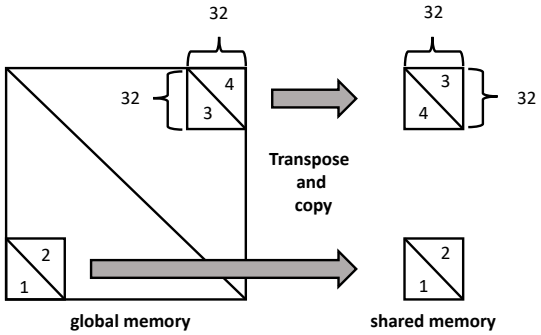

Figure 9. Transepose and copy

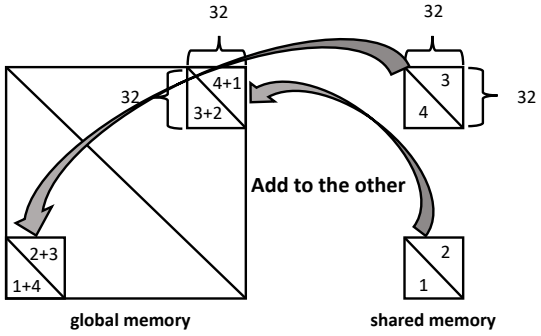

Figure 10. Add to other array

V. 性能評価

この章では巡回セールスマン問題に対する蟻コロニー最適化の提案手法を CUDA　C で実装．提案する GPU 実装を評価するために 1733MHz の動作周波数のコアを 2560 基搭載した NVIDIA GeForce GTX 1080 を使用した．また, CPU 実装には動作周波数 3.66GHz の Intel Core i7-4790 を使用した.．CPU の逐次実装は GPU 実装による都市選択と同じ手法となっている．今回の実装を比較するためのデータとしてベンチマーク [16] を使用している．使用したデータは都市数が 198，280，318，442，783，1002，2392, 3082 の場合となっている．本実装のパラメータである α, β, ρ はそれぞれ 1.0, 2.0, 0.5 を設定している．CUDA で大事な起動スレッド数はすべての実装で 32 となっている．まず初めに提案手法と既存手法に関する精度について言及している．次に経路構成の計測について，次にフェロモン更新の計測について，最後に処理全体の計測について紹介する．

A. 精度

厳密でない確率選択のみで都市選択を行った場合，厳密な確率選択を行った場合と比較して近似解の収束が遅くなってしまう．そのため提案手法では厳密な都市選択とハイブリッドに行うことによって解の収束を速くしている．手法の改善は蟻が近似解を得るまでの計算速度を速くするために行っており，解の精度を向上させるために行っているわけではない．解の精度に関しては提案手法，既存手法の両方とも ACO にを用いた手法となっているため同様の結果が得られるようになっている．手法による違いは解の収束の速さと近似解が得られるまでの時間となっている．なので今回は提案手法と既存手法のどちらも起動する蟻の数を都市数と同数として，既存手法が 100 回のフェロモン更新で得られる近似解に提案手法が到達するまでフェロモン更新を行う．Table I に提案手法が既存手法と同程度の精度が得られるまでにかかるフェロモン更新回数を示す．

Table I
提案手法のフェロモン更新回数

都市数	更新回数
198	112
280	139
318	154
442	161
783	159
1002	203
2392	347
3082	385

B. 経路構成

経路構成について本実装では Stochastic acceptance を基にしたルーレット選択と Tournament slect を基にした選択方法を切り替える実装となっている．切り替えるタイミングは各蟻が Stochastic acceptance を基にしたルーレット選択を失敗し始めるときが理想的だが，本実装は 1 ブロックがに複数蟻を担当しているため事前に切り替えるタイミングを決めなければならない．なので今回の計測結果は，いくつか計測をおこない最も高速であった実装を載せている．本実装の比較対象として，CPU による逐次実装と Stochastic acceptance を基にした実装である [5] を使用している．実装の比較では蟻の数を都市数とし，[5] と同程度の精度を得られるまでフェロモン更新を行った際の計測時間を Table II，Table III に示している．CPU 実装と比較したとき都市数 1002 のとき最大約 47 倍となっており，既存実装と比較して最大約 2.6 倍を達成した．既存実装では起動するブロック数が蟻の数となっており，都市と同じ数となっており，フェロモン更新の回数は 100 回としている．提案実装では 1 ブロック 32 スレッドで起動し，1 ブロックに複数蟻を割り当てている．1 匹の蟻に対して割り当てるスレッド数は Tournament select で選択する都市数 m に依存する都市数 198 から 1002 までは $m = 4$ を割り当て，都市数 2392 のとき $m = 8$，都市数 3082 のとき $m = 16$ としている．提案実装の起動ブロック数は都市数を 1 ブロックあたりの蟻の数で割ったものとなっており，フェロモン更新の回数は提案実装と同程度の精度が得られるまでとなっている．

Table II
経路構成：CPU との実行時間比較

都市数	CPU(ms)	GPU(ms)	CPU/GPU
198	467.522	66.34	7.5237
280	761.671	83.95	9.8726
318	1487.001	145.51	10.7824
442	3008.974	180.04	17.9812
783	17162.44	784.08	22.84123
1002	95541.87	2065.2	47.4813
2392	245173	5931.5	45.4836
3082	319203	8192.4	44.4953

C. フェロモン更新処理の比較

フェロモン更新の計算時間の比較を Table IV に示す．CPU 実装と GPU 実装の計算時間を比較して，都市数 1002 のとき最大約 65 倍の高速化を達成し，経路構成よりも効率化をすることができている．

Table III
経路構成：既存実装との実行時間比較

都市数	既存実装 (ms)	提案実装 (ms)	既存/提案
198	73.25	66.34	1.10416
280	124.81	83.95	1.48671
318	284.51	145.51	1.95526
442	472.51	180.04	2.62669
783	1554.12	784.08	1.98209
1002	4027.3	2065.2	1.950077
2392	9874.86	5931.5	1.66481
3082	14703.46	8192.4	1.79433

Table IV
フェロモン更新：CPU との実行時間比較

都市数	CPU(ms)	GPU(ms)	CPU/GPU
198	86.3	4.2	20.54
280	122.4	6.8	18.01
318	251.2	7.6	33.05
442	435.1	12.7	34.25
783	1614.6	32.7	49.37
1002	3441.6	53.0	64.93
2392	20471.4	541.1	37.83
3082	30154.1	1020.5	29.54

D. アルゴリズム全体の比較

最後に蟻コロニー最適化アルゴリズムの処理すべてに関して様々なインスタンスを用いて比較を行っていく．比較には既存実装と CPU による逐次実装を用いている．既存実装と提案実装の比較は経路構成の比較で説明した通りである．CPU と比較して都市数 1002 のとき最大約 47.92 倍を達成し，既存実装と比較して都市数 442 のとき最大約 2.69 倍を達成した．既存実装と提案実装ではフェロモン更新処理は同様の処理を行っているため高速化は行われていない．しかし経路構成よりも高速化率が上がっているのはフェロモン更新より経路構成の方が時間のかかる処理となっているためである．

Table V
処理全体：CPU との実行時間比較

都市数	CPU(ms)	GPU(ms)	CPU/GPU
198	553.8	66.34	8.348
280	884.0	83.95	10.53
318	1738.2	145.51	11.94
442	3444.0	180.04	19.12
783	18777.0	784.08	23.94
1002	98983.4	2065.2	47.92
2392	265644.4	5931.5	44.78
3082	349357.2	8194.4	42.63

VI. まとめ

本研究では，GPU を用いて巡回セールスマン問題を解くための蟻コロニー最適化による高速化を行い，CPU 実装，及び既存実装との比較を行った．結果として，CPU 実

Table VI
処理全体：CPU との実行時間比較

都市数	既存実装 (ms)	提案実装 (ms)	既存/提案
198	77.25	66.34	1.164
280	131.61	83.95	1.567
318	292.11	145.51	2.007
442	485.61	180.04	2.697
783	1586.61	784.08	2.023
1002	4080.30	2065.2	1.975
2392	10416.27	5931.5	1.756
3082	15724.97	8194.4	1.918

装と比較して，頂点数が 1002 のとき経路構成では頂点数 1002 のとき約 47 倍，フェロモン更新処理では約 65 倍，アルゴリズム全体で約 47.92 倍の高速化を達成した．既存実装と比較して経路構成のみだと都市数 442 のとき約 2.626 倍，アルゴリズム全体では約 2.697 倍の高速化を達成している．

REFERENCES

[1] NVIDIA Corp., *CUDA C Programming Guide*, http://docs.nvidia.com/cuda/cuda-c-programming-guide.

[2] G. H. Dal, W. A. Kosters, and F. W. Takes, "Fast diameter computation of large sparse graphs using GPUs," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE, 2014, pp. 632–639.

[3] T. Fujita, K. Nakano, and Y. Ito, "Bulk execution of Euclidean algorithms on the CUDA-enabled GPU," *International Journal of Networking and Computing*, vol. 6, no. 1, pp. 42–63, 2016.

[4] H. Koge, Y. Ito, and K. Nakano, "A GPU Implementation of Clipping-Free Halftoning Using the Direct Binary Search," in *Algorithms and Architectures for Parallel Processing*, 2014, pp. 57–70.

[5] A. Uchida, Y. Ito, and K. Nakano, "Accelerating ant colony optimisation for the travelling salesman problem on the GPU," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 29, no. 4, pp. 401–420, 2014.

[6] C. Alberto, D. Marco, and M. Vittorio, "Distributed optimization by ant colonies," *Proceedings of the first European conference on artificial life*, pp. 134–142, 1991.

[7] M. Dorigo, "Optimization, learning and natural algorithms," *Dipartimento di Elettronica*, 1992.

[8] ——, "The ant system: Optimization by a colony of cooperating agents," *Transactions on Systems*.

[9] T. Cheng-Fa, T. Chun-wei, and T. Ching-Chang, "A new hybrid heuristic approach for solving large traveling salesman problem," *Information Sciences*, vol. 166, pp. 67–81, 2004.

[10] K. Hitoshi, "Pheromone trail initialization with local optimal solutions in ant colony optimization," *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*, pp. 338–343, 2010.

[11] O. Junichi and K. Hitoshi, "Effective Pheromone Trail Initialization in Max-Min Ant System," in *The 26th Annual Conference of the Japanese Society for Artificial Intelligence, 2012*, 2012.

[12] T. Stuzle and H. Hoos, H, "Max-Min ant system," *Future Generation Computer Systems 16*, pp. 889–914, 2000.

[13] M. Dorigo and M. Gambardella, L, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *Transactions on Evolutionary Computation*.

[14] A. Lipowski and D. Lipowski, "Roulette-wheel selection via stochastic acceptance," *Physica A:Statistical Machanics and its Applications*, vol. 391, no. 6, 2011.

[15] A. Brindle, "Genetic algorithm for function optimization," *Doctoral dissertation and Technical Report TR81-2:University of Alberta,Department of Computer Sciense*, 1981.

[16] G. Reinelt, "TSPlib-a traveling salesman problem library," *ORSA Jounal on Computing*, vol. 3, 1991.