Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

GPU を用いた大規模ネットワーク交通流シミュレーションにおける経路探索計算の高速化

高橋浩司[†1]　花房比佐友[†1]　堀口良太[†1]

概要：道路網整備や混雑課金，流入制御といった，渋滞緩和を目的とした動的な道路運用施策の評価では，大規模ネットワークに適用可能な交通流シミュレーションモデルの活用に期待が集まっている．一般に，道路ネットワークを対象とした交通流シミュレーションでは，出発時及び走行中の車両に対して，所定のタイミングで経路選択モデルを適用し，個々の車両に交通状況に応じた目的地までの経路を動的に選択させている．しかしながら，大規模ネットワークでは経路選択モデルにおける最小費用経路探索計算が処理時間の多くを占めるようになり，施策評価の効率化の妨げになる．本研究では， GPU に搭載される数百～数千の演算コアによる処理能力を，CPU の代わりとして汎用計算に活用する GPGPU 技術をラベル修正法による経路探索計算に適用し，処理時間の高速化を行った．論文では，GPU の各演算コアを経路探索計算に割り当てて処理を並列化する方法を示し，CPU による経路探索計算との処理時間を比較した．

キーワード： 交通流シミュレーション，GPGPU，並列化，最小費用経路探索

1. はじめに

1.1 交通流シミュレーションにおける運用と課題

渋滞による影響は旅行者の時間的損失だけでなく，都市機能の円滑な運行，環境負荷の増加など様々な悪影響を及ぼす．このため，交差点改良や信号制御の見直しといった局所的なボトルネック対策だけでなく，バイパスや環状道路整備，混雑課金等の通行料金施策，あるいは渋滞情報提供による交通需要の空間分散といった，ネットワーク規模での対策が各所で検討・実施されている．これらネットワーク規模での対策の効果評価や，交通管制システムと連携してリアルタイムで交通運用・制御を行うにあたり，大規模ネットワークに適用可能な，交通状況の動的な変化を考慮した交通流シミュレーションに期待が集まっている（例えば[1][2]）．

交通流シミュレーションでは所定の時間間隔で経路探索を行うことで，動的な交通状況変化を考慮した最適な経路を選択させることが可能である．経路探索計算にはダイクストラ法やA*法など様々なアルゴリズム[3][4]が利用されており，目的地までの経路探索に要する時間はネットワークの拡大に伴って対数的に増加することは避けられない．特に，交通流シミュレーションでは，ネットワーク上に多数の車両発着点があり，ネットワーク規模の拡大に応じて，発着点数も増加するため，すべての車両の目的地毎に経路探索を行う必要があり，さらに計算時間が増加する．手元の調べでは，東北地区全域のデジタル道路地図データで作成したアーク数約120万本の広域ネットワークを対象とした場合，計算時間のおよそ6割が経路探索計算で占められるようになっており，経路探索計算の時間短縮が大きな課題となっている．

1.2 グラフィックボードによる高速化

一方，近年ではGPU（graphics processing units）を用いた並列計算が注目されており，科学技術計算の高速化に利用されている．GPUには演算コアが数百～数千個搭載されており，これらを同時に処理させることによって高い処理能力を発揮する．NVIDEA社は，2006年にGeForceシリーズのグラフィックボードで利用可能なC言語の統合開発環境とするCUDAを発表し[5]，GPUによる汎用なプログラムの計算（General Processing GPU : GPGPU）を可能にした．2008年にはApple社が中心として，CPU，GPU，Cell/B.E.およびDSPといったあらゆるアーキテクチャのプロセッサで共通に利用可能なOpenCLを発表した[6]．これらをきっかけとして，GPGPUによる高速化が様々な分野で注目されることになった[7]．2008年には東京工業大学がTesla S1070を搭載したスーパーコンピュータ（TSUBAME）によってTOP500における29位を獲得し[8]，また，2009年には中国の国防科学技術大学がRadeonHD4870X2を搭載した天河一号によってTOP500における5位を獲得した[9]．桑山氏はGPUによって行列計算を並列化し，行列のサイズの変化に対する速度向上の比較を行い，2048×2048のマトリックスにおける計算では575倍もの速度で計算が可能であることを示した[10]．筒井氏は進化計算において2次割り当て問題をアントコロニー最適化とタブーサーチを組み合わせた手法により，GPUによる超並列計算を行った[11]．

GPGPU による高速化は路探索計算においても複数報告されている．大島らはダイクストラ法と A*法の 2 種類の経路探索計算において，次の訪問先を決定する処理を，GPGPU を用いて高速化した[12]．平櫛らは Hybrid-BFS と呼ばれる幅優先探索計算のアルゴリズムで，訪問可能な頂点の列挙における処理に GPU を利用して高速化した[13]．しかしながら，複数の起終点をもつ交通流シミュレーションにおいては１つの経路探索計算が完了する度にデータの転送，初期化が必要であり，これらの時間は並列計算にお

†1 (株)アイ・トランスポート・ラボ
i-Transport Lab. Co., Ltd..

Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

けるオーバーヘッドとなる．

本研究では，交通流シミュレーションを高速化するために，道路ネットワークにおける目的地への最短経路を求める経路探索の計算時間を短縮することを目的とした．そこで，ラベル修正法による幅優先経路探索法において，グラフィックボードの各演算コアの処理能力を経路探索の開始地点毎に割り振り，複数の経路探索計算を同時に行う並列化を行った．

2. GPGPU による経路探索計算

2.1 経路探索アルゴリズムの GPGPU への適性

一般に，経路探索アルゴリズムには，ダイクストラ法に代表されるラベル確定法とラベル修正法の 2 種類がある．ダイクストラ法は次の経路探索を行う候補をコストの低い順にソートし，最低限の評価回数で各終点までの最短経路を求めることが可能である．対して，ラベル修正法は上流のコストが更新された場合に下流のコストを順次して最短経路を計算する．GPU は演算処理を Single Program Multiple Data（SPMD）方式で行うため条件分岐を含む処理ではパフォーマンスが低下する．よって，次のコスト評価を行う対象を決定するためにソートを繰り返すラベル確定法は，GPGPU に不適となる．本研究では，GPGPU において継続的な処理能力を発揮できるラベル修正法による並列化を行った．

本研究では地点をノード，各ノードに接続する経路をアークで表す．図 1 に，複数の経路探索計算を GPGPU によって並列して行うための処理の流れを示す．GPGPU では CPU で処理を行う関数からの命令によって GPU 計算向けの関数を呼び出す．また，GPU による計算ではグラフィックボードに搭載される VRAM（Video RAM）と呼ばれるメモリを使用する．そのため，GPGPU では予め，VRAM 上に配列等のデータ領域を確保し，メインメモリから必要なデータを転送する．GPU による計算の後は，VRAM に保存される結果データを必要に応じてメインメモリに転送する．配列のメモリ再確保には演算処理と比較して長い時間を必要とする．よって，ノードの下流に対してコストを評価するためにノード番号をスタックする検討候補リストは，循環リストの方法によって制御を行った．一度確保したメモリから再度確保し直さないために，検討候補リストは未検討のノードで満たされないだけの十分なバッファ量で確保する．使用可能なメモリ量に上限がない場合，このバッファ量は全ノードの数で確保する．この時，検討候補リストのバッファ量はネットワークの大きさ，ノードの平均隣接数，通過コスト値のばらつきに依存する．NVIDEA 社製のグラフィックボードでは，演算コアは自身の所属する SM（Streaming Multiprocessor）の番号とその中での通し番号を取得可能である．これにより，GPGPU では全演算コアに対する通し番号が設定可能である．この全演算コアに対する通し番号を出発の起点ノード番号に割り当てることによって並列化を行った．

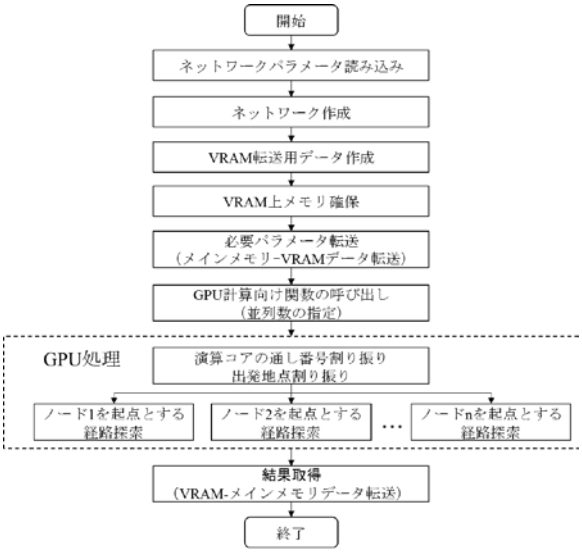

図 1 複数のノードを起点とした経路探索計算の並列化における処理フロー

2.2 ネットワークの作成

道路交通における経路探索においては車両の出入り口の他，交差点や車線数が変更する地点などをノードとし，ノード間の道路をアークで表す．アークは方向性を持つ有向性経路である．歩行者及び車両の運転者は最小コストで移動すると仮定したとき，現在地を起点として目的地に対して，より負担の小さい経路を選択する．コストには，走行距離と各区間の想定速度から得られる所要時間だけでなく，渋滞などの交通状況，通過料金，右左折によるストレスなどが考慮される．本研究では右左折によるストレスを考慮して評価可能にするために，道路構造のネットワークから拡張したネットワークモデルを作成した．図 2 に 1 本の道

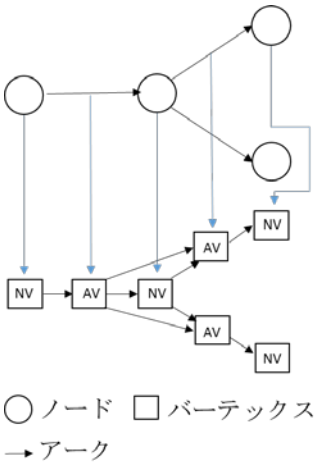

図 2 二方向分岐点における経路探索ネットワーク図
（上図：道路ネットワーク図，下図：右左折コストを考慮可能な経路探索ネットワーク図）

Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

から 2 方向へ分岐する道路における拡張前後のネットワーク図を示す. 車両の進行可能な方向を矢印で示す. 便宜上，拡張後のノードをバーテックスとする．拡張ではアーク上に経路探索の地点候補となるバーテックスを設定し，アーク上のバーテックス同士を接続した．ノードに由来するバーテックスを Node Vertex (以下，NV)，アークに由来するバーテックスを Arc Vertex（以下，AV）とした．

経路探索計算では，それぞれのバーテックスの接続状態を表現する為に隣接行列が用いられる．この行列はバーテックス i とバーテックス j が接続していないときは 0, 接続しているときは 1 で表現され，全バーテックス数 N の二乗の要素数を持つ．本研究では論文[13]を参考に，接続する座標の組み合わせのみを抽出し，1 次元配列で表す接続先バーテックスリストを作成した．また，接続先バーテックスリストと 1:1 で対応する通過コストリスト，それぞれのリストで各地点の接続対象へ適切にアクセスするためのオフセットリストを作成した．それぞれの地点の接続数が 0 から 5 程度である道路ネットワークにおいて，接続先バーテックスリストはメモリ使用量を大きく削減可能にする．

2.3 GPGPU の実装

本研究では NVIDIA 社の提供する GeForce シリーズのグラフィックボードと, 同社で開発をサポートする CUDA によって並列化を行った．GPU は 32 個の演算コアを 1 組とした SM 毎に独立した命令ユニットを持ち，それぞれ個別の命令を実行する．CUDA による並列化計算ではスレッドとブロックと呼ばれる概念が存在する．CUDA では 1 つの SM 内で立ち上げる並列処理をスレッドと呼び，いくつの SM を立ち上げるかをブロックによって制御する．プログラム中ではこの 2 つを定義し，スレッド数×ブロック数の処理を同時に行う．スレッドを多く立ち上げることで，演算コアが新しいデータを取り入れるための遅延時間を隠蔽することが可能とされる．ここで，命令の書き換えのタイミングは SM 内のすべての演算コアが 1 サイクル計算された時となる．よって, 立ち上げるスレッド数は 32 の倍数であることが理想的であるとされる．しかしながら, 同一 SM 内の演算コアは個別の命令を実行することができないため, 同一 SM 内で異なる分岐処理が行われる場合は片方ずつ計算が行われ，計算効率が低下する特徴を持つ．また，GPU は倍精度の値の計算処理は単精度の値の計算処理に比べて 3 倍近く遅くなる．本研究では浮動小数点形式で計算を行う値は単精度で扱うものとした．

CUDA ではグラフィックボードに搭載された VRAM のデータを使用して計算が行われる．GPGPU による並列計算においては，連続したアドレスを取得することで高速にデータを取得することが可能である[14]．CPU による計算では，バーテックス毎に構造体を作成して計算を行うが，CUDA による並列計算では複数の構造体を同時に参照することになるため，データのアクセススピードが低下する．また, VRAM のメモリの確保や VRAM－メインメモリ間の双方向におけるデータの転送は数値計算と比較して処理が遅く，GPGPU におけるオーバーヘッドとなる．これらの処理は 1 度に多くのデータ量を扱うことで時間を短縮することが可能である．以上より，GPU による計算では構造体のメンバー毎に一次元配列を作成して計算を行った．GPU で使用する配列のメモリは CPU で処理される関数内で確保される．

GPU による計算では起点のバーテックス毎に演算処理能力を割り振って計算を行う．以下に示す配列は他の演算コアでの計算結果の保存場所と重複を防ぐためにバーテックス数と並列数の積の数だけデータ量を確保する．

- 地点到達のための最小コスト値
- 最短経路を辿るための上流バーテックス番号
- バーテックスが検討候補リストに保存されているかのフラグ

同様に，検討予定のバーテックス番号を保存する検討候補リストはバッファ量と並列数の積の数だけのデータ量を確保する．

VRAM のデータ量はチップセットに依存しており，バーテックス数と並列数の積でメモリ量を必要とする今回の手法では，広大なネットワークに対してすべてのバーテックスを起点として同時に並列計算することは不可能となる．よって，各要素で確保するメモリ量の合計がグラフィックボードに搭載される使用可能なメモリ量を超えない様に注意を払い並列化を行った．並列化数が 100 である場合，起点の座標番号 1 から 100 までを起点とした経路探索計算を並列化する．計算結果をメインメモリへ転送した後，各バーテックスのコスト値を初期化し，起点の座標番号 101 から 200 までを起点とした計算を並列化する．これを，全バーテックスを起点として計算が完了するまで繰り返し行う．

3. 検証

3.1 環境の比較

本研究では Intel Core i7-5820K の CPU と GeForce GTX 750 Ti の GPU を用いてラベル修正法を行うことにより計算時間の測定を行った．並列化による効果を評価するために，Intel Core i7-5820K ではシングルコアによる計算で要する時間を測定した．使用する CPU 及び GPU の性能を 1 に示す．本研究では CPU と GPU の計算で共通する，メイン

表 1 使用したハードウェアの主な性能

	CPU	GPU
ハード名		GTX750TI-OC-2GD5
搭載チップ	intel i7 5820K	GeForce GTX 750 Ti
コアクロック数	3.30GHz	1.05GHz
コア数	6	640
メモリ量		2G

Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

メモリ内でネットワークの形成ための時間は計測の対象外とした．GPU による計算では，CPU の関数内で VRAM にデータを転送するためのネットワーク情報を作成する処理に要する時間を含めて計測した．

3.2 ネットワーク規模に対する速度比較

本研究では先ず，ネットワーク規模に対する計算速度の違いを評価した．評価するためのネットワークとして，条里制形状のネットワークを作成した．それぞれの格子点にノードを置き，これらの繋ぐアークを設定した．ネットワークの淵は T 時交差点で囲われており，ネットワーク奥部では縦と横のアークが格子状に取り付けられる．全てのアークは起点と終点のノードが逆に取り付けられた対となるアークが存在し，U ターン走行が可能であるものとした．このとき，AV の，NV を含めた平均接続経路数は 4.8~4.9 となる．それぞれのアークの上部と下部には通過コストとして，1 秒から 10 秒のランダムな所要時間コストを与えた．1 辺あたりのノード数を 15，30，45，60，75，90，105，120 に増やしたときの，全バーテックスが各バーテックスまでの最短経路を計算するのに必要な時間を計測した．に CPU と GPU で計算に要した時間を示す．初期化時間は CPU 関数内であらかじめ構成されている情報から GPU を活用するために処理を行う時間である．結果取得は GPU での計算で求めた各バーテックスへの最短経路情報をメインメモリへ転送する処理に要した時間である．データの転送量は起点数と地点数の積となる．ネットワークが小さい時は初期化と結果の取得におけるオーバーヘッドがかさみ，大きな速度向上は見られなかった．バーテックス数が 17,760 以上では 6 倍から 8.1 倍の速度で計算を行うことができた．ここで，結果取得に要する時間は全体のおよそ 3%であり，大きなボトルネックにはなっていないことがわかった．

図 3 にネットワークのバーテックス数と，経路探索の計算に要する時間を全バーテックスで割った値の両対数グラフにしたものを示す．破線は傾きが 2.4 の直線を示し，1 回当たりの経路探索の計算時間がバーテックス数の 2.4 乗に比例して増加していることがわかる．ここで，演算コアの処理能力と並列数を考慮したとき，理論的な性能よりも速度は大きく低下していた．これは，ラベル修正法は，一

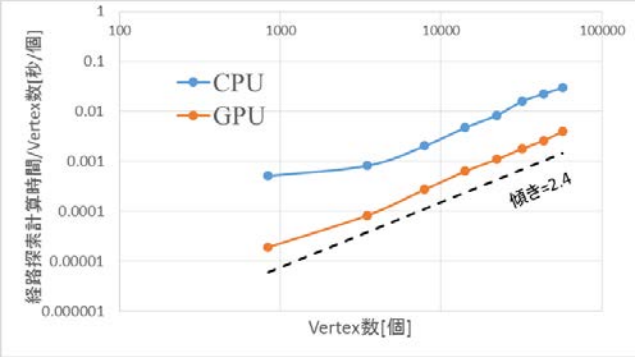

図 3 経路探索計算時間のネットワーク規模依存性

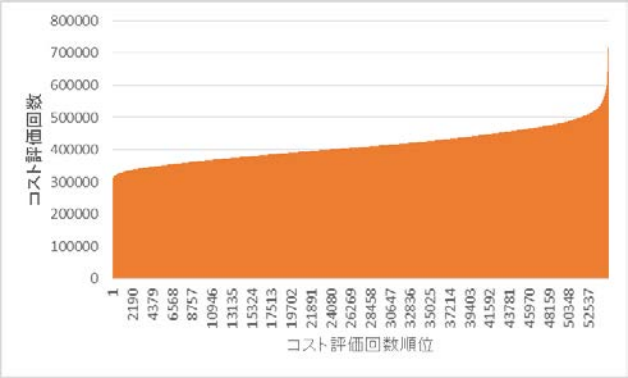

図 4 条理制形状ネットワークにおけるコスト評価回数順位

度求めたノードをその上流のノードの更新によって再度検討される可能性があり，その更新回数が起点よって異なるためであると考えられる．図 4 は 1 辺あたりのノード数が 105 のとときの全バーテックスへの最短経路が確定するまでに，コスト評価を行った回数を起点バーテックス毎に集計し，少ない順に表したものである．このコスト評価回数の平均値は 413,584 であり，分散値は 51,291 であった．GPGPU では並列化した演算コアの全工程が終わるまで，結果の取得といった次の工程を行うことが不可能である．経路探索の並列化によって，コスト評価回数の多い起点と少ない起点が混在することで，演算コアに他の計算の終了待ちをする待機時間が発生する結果となった．また，グラ

表 2 条理制形状ネットワークの規模に対する速度比較

ネットワークサイズ				計算時間[秒]					計算速度倍率（CPU/GPU）
1 辺あたりのノード数[個]	バーテックス数[個]			CPU	GPU				
	NV	AV	合計		初期化	経路探索	結果取得	合計	
15	225	840	1,065	0.425	0.130	0.016	0.003	0.149	2.9
30	900	3,480	4,380	2.84	0.506	0.289	0.025	0.820	3.5
45	2,024	7,920	9,944	16.1	1.05	2.16	0.103	3.313	4.9
60	3,600	14,160	17,760	67.0	1.99	8.94	0.320	11	6.0
75	5,625	22,200	27,825	185	2.84	24.3	0.778	28	6.6
90	8,100	32,040	40,140	509	4.17	57.2	1.60	63	8.1
105	11,025	43,680	54,705	991	5.44	114	2.94	122	8.1
120	14,400	57,120	71,520	1680	6.83	226	4.99	238	7.1

Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

フィックボードは分岐処理において，また，検討中の頂点から接続する頂点の数の違いやコスト値の更新処理の有無によって分岐処理が発生し，速度が低下する要因となる．そして，ラベル修正法では検討対象のバーテックスが演算コア毎に変化するため，メモリアクセスが乱雑となってしまい，個々の演算における処理速度が低下していることが上げられる．

3.3 道路ネットワークにおける検証

実用的なモデルでの検証として，東京 23 区のネットワークに対する経路探索計算を行った．この時，ノード数は 20,580，アーク数は 34,210 であり，全バーテックス数は 54,790 であった. AV の, NV を含めた平均接続経路数は 2.9, 接続経路数の分散は 1.2 となった．アークの上部コスト，下部コストはアーク長と自由速度による所要時間を分割することで設定した．本研究ではスレッド数が 32，ブロック数が 40 の 1,280 並列で計算を行ったとき，最も高いパフォーマンスを得られた. このときの VRAM のメモリ使用量は 1.25G となった．

東京 23 区のネットワークデータにおいて，全て起点バーテックスから各バーテックスまでの最小コストが決定するまでに要した時間を表 3 に示す. 初期化時間と結果取得時間は全体の 3%程度であった．CPU による計算では 2,785 秒，GPU による計算では 211 秒であり，12.6 倍の速度で計算することができた．条理制形状のネットワークでは，1 辺当たりのノード数が 105 のとき, 東京 23 区のネットワークのバーテックス数とほぼ同等となる．ここで，東京 23 区のネットワークでの計算時間は CPU での計算と GPU での計算の両方において，条理制ネットワークでの計算時間よりも遅くなった. これは東京 23 区のネットワークの複雑さが影響すると考えられる．図 5 に東京 23 区ネットワークにおける起点バーテックス毎のコスト値の評価回数を小さい順に並べたものを示す．U ターン走行が不可能で，行き詰まりとなっているネットワーク末端の NV，AV が 451 箇所あり，これらのバーテックスを起点とした経路探索計算ではコスト評価を行った回数が 50 以下となっている. また, 東京 23 区のネットワークでのコスト評価回数の平均値は 698,961 回，分散値は 211,660 であり，コスト評価回数が条理制形状のネットワークより大きくばらついていることがわかる．条理制形状のネットワークではアークの繋がり

表 3 東京 23 区ネットワークにおける速度比較

	計算時間[秒]		所要時間
	CPU	GPU	割合[%]
初期化		5.67	2.6
経路探索	2,785	211	96
結果取得		3.11	1.4
合計	2,785	220	100

図 5 東京 23 区ネットワークにおけるコスト評価回数順位

が近くのノードのみとなっている．ラベルの修正が起点を中心として広がるだけとなり，検討候補リストへ繰り返し保存される回数は少ないと考えられる．対して，東京 23 区のネットワークでは，首都高速道路によって起点位置から遠く離れた地点も早い段階で検討対象になる．これにより，一度求めたバーテックスが別の方向から求められたバーテックスによって検討リストに保存されることになり，多くのバーテックスでコストの再評価がされていると考えられる．

次に CPU による計算での時間増加率と GPU における計算での時間増加率について議論する．先に記述した通り，東京 23 区での計算ではコストの評価回数は起点バーテックス毎に大きく異なる．これにより，GPU による計算では演算コア毎の計算量の差が大きくなり計算効率が大きく下がると考えられる．そこで，1 つの経路探索あたりの計算時間を条理制形状のネットワークと比較する．ネットワーク規模に対する経路探索の時間は AV に依存すると考えられるため，条理制形状のネットワークでは 1 辺当たりのノード数が 90 のときを対象とした．CPU による計算では条理制形状のネットワークでの計算時間は 0.012 秒，東京 23 区での計算時間 0.051 秒となり 4.0 倍の計算時間となった．GPU による計算では条理制形状のネットワークでの計算時間は 0.0014 秒，東京 23 区での計算時間 0.039 秒となり 2.7 倍の計算時間となった．これには作成したプログラムのデータ構造とメモリのキャッシングによる差が現れた可能性が挙げられる．コンピュータによる計算において，データの取得には今後の計算を考慮して，一定の範囲のセグメントをまとめて取得し高速なデータアクセスを可能にするキャッシングの方法がとられる. 本研究では CPU による計算ではバーテックスの構造体毎にデータを作成したためキャッシングによる 1 次キャッシュへの保存が効率的に行われていないと考えられる．対して，GPU による計算ではデータ構造は構造体のメンバー毎に配列を作成した．これにより，1 度に取得されるデータのセグメントは同一の配列のアドレスとなるため，データのアクセス効率が良くな

Vol.2017-DPS-171 No.6
Vol.2017-MBL-83 No.6
Vol.2017-ITS-69 No.6
2017/6/1

ったと考えられる.

4. まとめ

本研究ではラベル修正法の経路探索において，起点毎にGPU の演算コアを割り当てて複数の経路探索を同時に行う並列計算を行った.道路アークを経路探索の頂点として，右左折のコストを考慮可能なネットワークを作成して経路探索を行った．本研究では初めに，U ターン可能な条理制形状のネットワークを作成し，ネットワーク規模の違いに対する計算速度の比較を行った．地点数が少ない場合，VRAM のメモリ確保やデータ管理といった初期化と計算結果の取得に要する時間が多くかかり，高い速度改善効果は得られなかった．しかしながら，バーテックス数が540,140 個を超える大きなネットワークでは 7 倍から 8 倍の速度で計算が可能であることがわかった．GPGPU ではグラフィックボード上の VRAM からメインメモリに計算結果を転送するために時間のロスが発生するものの，1 つの配列でまとめて転送を行うことで処理全体にかかる時間の 3%程度の時間となり大きなオーバーヘッドにはならないことがわかった．次に，実用的なモデルとして，東京 23 区のネットワークを基に経路探索計算を行った．同規模の条理制形状のネットワークと比較すると，高速道路などによるネットワークの複雑さによってコストの更新回数が増加し，CPU による計算と GPU による計算で共に遅くなる結果となった．しかしながら，GPU での計算は CPU での計算に比べて 13.1 倍の速度で計算を行うことができた.

今回の手法では，メモリ使用量はネットワークの大きさと並列数に依存するため，更なる大規模計算を行うためにはメモリ使用量が並列数に依存しないアルゴリズムの開発や，小規模ネットワークの経路探索計算を階層的に行う手法の開発が必要と言える．また，今回の比較対象であるCPU での計算はシングルコアでの計算速度であるので，マルチコアで処理を行った場合の計算速度を測定する必要がある.

参考文献

[1] 宗像恵子，田村勇二，割田博，白石智良．首都高速道路におけるリアルタイム予測シミュレーションの開発．第 29 回交通工学研究発表会講演論文集, 2009. 11.
[2] 三谷卓摩，原祐輔，桑原雅夫, EV 交通シミュレーションを用いたエネルギー・モビリティマネジメントシステムの開発．第 12 回 ITS シンポジウム. 2014. 12.
[3] 天目健二．ナビゲーションシステムと経路探索 特集 次世代道路交通システム—ITS．オペレーションズ・リサーチ, 2000, 7.
[4] 安井雄一郎，高宮安仁，藤澤克樹．大規模経路探索問題に対する高速処理システム ーメモリ階層構造の考慮とクラスタ&クラウド技術による高速化ー．数理解析研究所講究録．第 1676 巻. 2010.
[5] NVIDIA CUDA. http://www.nvidia.com/object/cuda_home_new.html
[6] アップル, Mac OS X Snow Leopard をデベロッパにプレビュー, http://www.apple.com/jp/pr/library/2008/06/09Apple-Previews-Mac-OS-X-Snow-Leopard-to-Developers.html
[7] GPU コンピューティング学会. http://gpu-computing.gsic.titech.ac.jp/node/1
[8] 東京工業大学 学術国際情報センター．東京大学が世界初の大規模 GPGPU コンピューティング基盤を大規模スーパーコンピュータ基盤 TSUBAME 上に実現．プレスリリース, http://www.gsic.titech.ac.jp/node/141
[9] TOP500 Supercomputing Sites. Tianhe-1, November, 2009. https://www.top500.org/lists/2009/11/
[10] 桑山裕樹. GPU を用いた行列演算の高速化．卒業論文. 2010.
[11] 筒井茂義, GPU を用いた高速並列化計算による組み合わせ最適化問題へのアプローチ．オペレーションズ・リサーチ, 2012, 5.
[12] 大森信，谷松宏紀．マルチコアおよび GPU を用いたグラフ形データベースの性能評価．電子情報通信学会技術研究報告. 2014. 1.
[13] 平櫛貴章，高橋大介. GPU クラスタにおける幅優先探索の高速化．研究報告ハイパフォーマンスコンピューティング（HPC）, 2013. 5. 22.
[14] 青木村之，額田彰．はじめての CUDA プログラミング，株式会社 工学社, 2009.