Vol.2013-ICS-172 No.1
2013/11/12

大規模マルチエージェント交通シミュレーション実現にむけたパラメータ最適化についての検討

佐野義仁[1,a)]　福田直樹[1,b)]

概要：エージェントシミュレーションでは，シミュレーション内でエージェントの行動の詳細度が必要な水準で得られることと同時に，シミュレーションの規模を拡大することができるということが，そのシミュレーションの有用性を引き出すために重要となる．しかしながら詳細な動きを行う何百万ものエージェントを扱うことができるシミュレーションを実行するためには，多くの演算処理が必要となる．このような課題に対する解決策の 1 つとして GPU を計算資源として利用する方法が考えられるが，GPU の処理能力を引き出すために，実行環境それぞれに合わせて，実行パラメータを設定しコードの最適化を行う必要がある．本論文では，シミュレーション開発者が GPU ベースのプログラミングを行う時のコーディングおよびパラメータチューニングを支援するためのフレームワークを提案する．提案フレームワークにおけるパラメータチューニングやコーディングの支援方法を示す．

キーワード：エージェントシミュレーション，マルチエージェントシステム，GPU コンピューティング

1. はじめに

マルチエージェントシミュレーションは様々な対象に対して適用されており，例えば道路交通シミュレーション [1]，群衆シミュレーション [2]，空港の避難シミュレーション [3] などへの適用が行われている．

エージェントシミュレーションにおいては，大規模化が 1 つの課題である．現実の世界では，対象を 1 つの都市に注目して考えたとしても，その 1 つの都市の交通が他の都市における交通流入・流出量に影響される場合もあり，たとえ 1 つの都市をシミュレートするためにも，対象とする都市の規模によっては何百万台の車両が存在する状況を再現できることが必要となる．例えば，電気自動車専用レーンを都市に導入した場合の影響を検討するためのマルチエージェントシミュレーションでは，およそ 300 万のエージェントによるシミュレーションが行われた [4]．

大規模化の実現と同時に，複雑シミュレーションの詳細度を確保できることも重要な課題の 1 つである．例えば，空港の避難シミュレーションにエージェントシミュレーションを導入した例 [3] では，従来ではあまり考慮されてこなかった人間の恐怖といった感情や家族といった人間関係をエージェントシミュレーションに導入した避難シミュレーションを行うことで，それまでのシミュレーションでは正常な避難ができると考えられていた災害シナリオにおいて，逃げることができない人間 (エージェント) が存在する可能性を示す結果が得られている．災害やイベントなどの非日常的な状況に対して，車両や人々がどのように行動をするのか分析したり，あるいは行動主体の微細な動きが，シミュレーション対象全体の動きに大きく影響を与える問題に対して綿密に議論を行いたい場合には，エージェントが動的な状況の変化に適切に反応して行動できるようにプログラムされていることが必要となる．すなわち，エージェントが動的な環境変化に対応できるようにコーディングされ，それを合理的な時間内でシミュレーションにより動作させることができる必要がある．例えば，道路交通の分野では，シミュレーション実行途中での細粒度なエージェントのプランニング処理の効果について検証が進められている [5]．

シミュレーションの詳細度を高めつつ大規模化を実現するためには，マルチエージェントシミュレーションの処理の効率化が 1 つの課題である [6]．非常に大規模でかつ複雑な環境 (空港，混み合った駅あるいは都市全体など) で複雑な動きをするエージェントの振る舞いを，何百万ものエージェントに対して再現したシミュレーションを実行す

[1] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University, Johoku, Hamamatsu, Shizuoka, 432-8011, Japan
[a)] gs13017@s.inf.shizuoka.ac.jp
[b)] fukuta@cs.inf.shizuoka.ac.jp

るには，多くの演算処理能力が必要となる．

本研究の背景となる大きな目的は，GPGPU などの技術を用いることによって，汎用的な計算に利用することが想定されていない計算資源を利用し，動的な環境の変化に対応することができるエージェントのプログラムコードを並列実行することによって，マルチエージェントシミュレーションのスケーラビリティの改善を容易にすることである．GPU などの計算資源を利用してハイスケーラブルなシミュレーションを実現をするためには，シミュレーションプログラム内でそれらの計算資源の演算コアを効率的に利用できることが重要となる．本論文では，GPU などの計算資源を利用したプログラムをシミュレーション開発者がコーディングする際の，コーディング作業およびパラメータチューニングプロセスを支援することを可能とするフレームワークについて述べる．具体的には，我々が提案するフレームワークが，様々なシミュレーション実行環境に合わせて，どのようにパラメータチューニングやコーディングを支援できるかを，いくつかの最適化シナリオに基づいて示す．

2. 関連研究

大規模なマルチエージェントシミュレーションを柔軟に実現するために，シミュレーションを利用するユーザのニーズにあわせてエージェントの行動の詳細度を動的に変化させる機構が提案されている [7]．この研究では，シミュレーションの複雑度と規模との間にあるトレードオフに注目し，ユーザが詳細にシミュレーションを行いたい場合では，複雑かつ厳密にシミュレーションを行うようにし，それ以外の場合ではエージェントの行動などを簡易化してシミュレーションを行うことを可能としている．すなはち，シミュレーションのいくつかの処理を必要に応じて動的に削減可能とすることによって大規模化の実現を可能としている．マルチエージェントシミュレーションの大規模化についてのアプローチとして，このほかに文献 [2] の手法がある．文献 [2] の研究では，シミュレーションを行う対象環境である 3 次元空間全体を考慮してシミュレーションを行う代わりに，空間内にリンクとノードの概念を導入してエージェントの行動可能範囲を制限することによって，エージェントの計算処理を劇的に縮小させることを可能としている．これらの方法は、マルチエージェントシミュレーションの詳細度を効果的に単純化することにより大規模なマルチエージェントシミュレーションを実現している．

大規模並列処理コンピュータによる大規模交通シミュレーションの効率的な処理の実現をするための目的でも，多くのアプローチが提案されている．エージェントシミュレーションを行うための基礎研究として，文献 [8] では，スレッドレベルの並列プログラムを効率的に実行することができる IBM Zonal Agent-based Simulation Environment, 通称 ZASE と呼ばれるエージェントサーバを開発した．ZASE は，スレッドレベルによる並列実行によって処理の高速化がされた 2 つ以上のエージェントサーバを組み合わせ，そして，大規模並列処理コンピュータ上で大規模なエージェントシミュレーションの実行を可能とするために，エージェントシミュレーション内の処理を効果的に分解してそれぞれに対してエージェントサーバに適切に割り当てることによって大規模化を図っている．このようなアプローチは，SMP に基づいたスカラプロセッサコンピュータクラスタに適用可能だが，一方で，我々が対象にしている民生用コンピュータ上での GPGPU を利用した処理には，その基本アーキテクチャの違いから，容易には適用できない．

表 1 OpenCL の典型的なコンパイラーオプションパラメータ
Table 1 Typical compiler option parameters used in OpenCL

Compilar option	
Math Intrinsics option	Optimazation option
-cl-single-precision-constant	-cl-opt-disable
-cl-denorms-are-zero	-cl-mad-enable
-cl-fp32-correctly-rounded-divide-sqrt	-cl-no-signed-zeros
	-cl-unsafe-math-optimizations
	-cl-finite-math-only
	-cl-fast-relaxed-math

道路交通シミュレーションで広く利用される一部のグラフ探索アルゴリズムに対しては，GPGPU を利用した大規模グラフの最短経路パスを求めるなどの多くのアプローチ方法が研究されており [9][10]，たとえば最短経路問題では CPU を利用する場合よりも GPGPU を利用した方が高速に処理することができる場合があることが報告されている．一方で，エージェントの振る舞いの詳細な動きを適切に再現するためには，最短経路探索のような単純なグラフアルゴリズムのみでなく，それらより複雑な処理をエージェントに行わせることができるようにする必要が生じる．それらの複雑な処理に対しては，既知の経路探索アルゴリズムに対する GPU 処理への最適化アプローチを単純に適用できない場合があり，個別に最適化を行う必要が出てくる．

GPU の性能特性は，それぞれの GPU 毎に異なり，実行条件や実装コードにも依存する．また，アルゴリズムおよび GPU の種類の組み合わせに対して最適パラメータがあると考えられるが，その最適な組合せは，多くの場合は既知ではない．表 1 は，OpenCL プログラミング用のコンパイラオプションパラメータのリストである．このようなパラメータなどを考慮して，GPU の演算性能を効率的に利用するために，コードやパラメータを適切に最適化することが必要となる．

例えば、CPU-GPU 間通信に関して CPU と GPU の通信パターンの最適化により実行速度を改善することを目指す研究がある [11]．文献 [12] では，様々なプログラムサイズおよび様々なアーキテクチャに合わせて，タスク分割を

自動的に最適化するアプローチが提案され，文献 [13] では，ヘテロジーニアスな分散コンピューティング環境自体の最適化アプローチを提案している．さらに，文献 [14] で提案されているアプローチでは，GPU アーキテクチャにおけるステンシル計算のためのコード生成手法が提案されている．これ以外にも，GPU を効率的に利用するために計算処理の流れを最適化する研究 [15] や，GPU 向け処理の実行時の最適化に関する研究 [16] がある．このように，GPU を利用した処理に対する最適化に関する研究が数多く存在する．一方，例えばコンパイラのパラメータ調節などのいくつかの最適化方法では，より高速に処理をおこなうためのパラメータの指定が計算精度に影響を及ぼす場合がある．この場合には，エージェントシミュレーションの計算結果に重要な影響を与えることが考えられるため，そのような最適化がどのようにシミュレーションに影響するかを調査しながら最適化を行う必要がある．

3. 提案フレームワーク

エージェントシミュレーションの構築を行う時を考えた場合，シミュレーション開発者は GPGPU ベースのプログラミングの専門家であるとは限らない．本論文では，GPGPU を利用して処理を行うことができる特定の規模のエージェントシミュレーションを容易に構築し分析することができ，また，開発者が特定のハードウェア上の実行速度を容易に分析し最適な実行に向けたチューニングを行うことを可能にする機能をもったフレームワークを提案する．

GPU は，一つの命令処理を複数のデータに適用し，スレッドを並列的に実行する SIMD 計算が得意である．このような並列処理を行うコアプログラムをカーネルプログラムと呼ぶ．本研究で作成しているフレームワークでは，プランニング処理用のプログラムをカーネルプログラムとして記述することによって．並列的にプランニング処理を行うことができるようにする．また，各エージェントに適用されるプランニングアルゴリズムがシミュレーションでシミュレートしたい内容によって異なる可能性が考えられる．本論文では，GPU 上でより高速に実行することが可能であるプランニングアルゴリズムの検討ではなく，GPU の使用により並列処理を行うことができるプランニング処理を含めたエージェントシミュレーション全体の処理速度向上についての検討を行う．

図 1 に我々が提案するフレームワークの構成を示す．フレームワークは，Simulation Module, Real-time Rendering Module, Benchmarking Module, User Interface Module, および Parameter Tuning Module の 5 つのモジュールから成る．Simulation Module は，シミュレーション開発者が作成したカーネルプログラムでのプランニング処理などを含んだ簡易的な交通シミュレーションを実現するためのモジュールである．Real-time Rendering Module は，

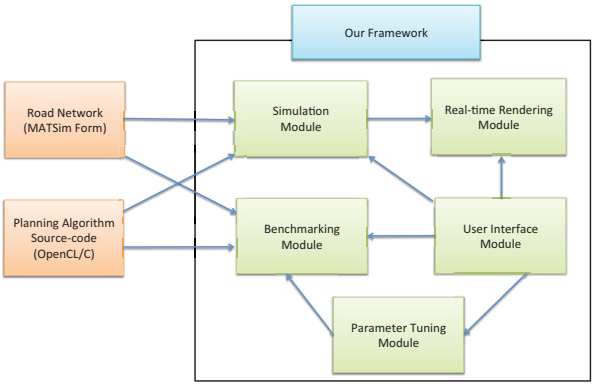

図 1 提案するフレームワークの構成
Fig. 1 The structure of proposed framework

シミュレーション状況をリアルタイムに表示可能とする．Benchmarking Module は，GPGPU を利用してカーネル関数を実行した場合のパフォーマンスとスケーラビリティをテスト，および評価するためのモジュールである．User Interface Module は，フレームワーク上で，様々な関数の実行をコントロールするためのフロントエンドである．そして，Parameter Tuning Module は，開発者が様々な GPU とシミュレーション設定に対するパラメータセットを調節する際の補助機能を実現するモジュールである．

本フレームワークを利用する開発者は，C 言語上で OpenCL プログラミングモデルを用いて，プランニング処理などに対するカーネルプログラムの記述を行う．OpenCL を用いる理由は，習得の容易性と様々なハードウェアプラットフォームをサポートしているからである．この際，カーネルプログラムは道路網のデータやエージェント数といったデータを引数として受け取ることができ，開発者は，それらの引数を利用しつつプログラムの記述を行う．開発者は，作成したファイル名，カーネル関数名，引数に関する情報が記述されたファイル名をフレームワークに与えることによって関数をフレームワークに登録する．開発者はシミュレーション環境上で自身の指定したプランニング処理を OpenCL を用いて実行させることが可能である．

図 2 は，我々が提案するフレームワークでのエージェントのコーディング例である．開発者は，地図データなどのシミュレーション内のデータに対して共通引数を経由してアクセスすることができる．さらに，開発者は，プランニング処理の中で使用される作業領域などのユーザ定義の引数を定義することが可能である．ユーザ定義の引数は，本フレームワークに変数の型，使用するメモリ量の情報を与えることにより利用可能である．引数の記述の後，実行プラットフォームによって割り当てられたエージェント ID を取得するための記述と，アルゴリズムなどの実際の処理の主要部分について記述を行う．

エージェントシミュレーションの開発者が，作成したカーネル関数に対して GPGPU を効率的に動作させたい場

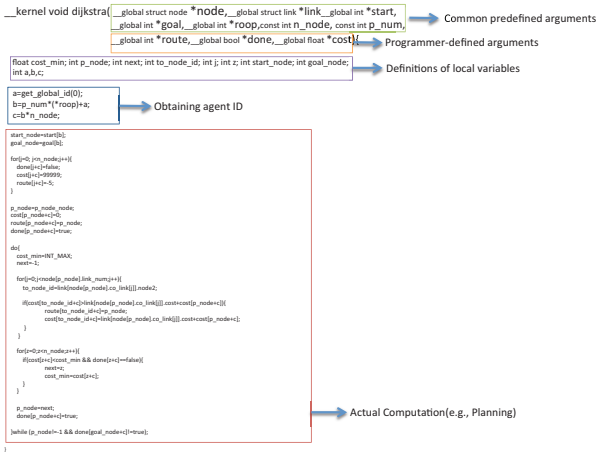

図 2 フレームワークで用いるコード例
Fig. 2 An Example Code using the Framework

合，調節すべきいくつかのパラメータが存在する [17]．本フレームワークでは，開発者がカーネル関数の評価を行う場合，パラレルスレッド数，メモリ配置，コンパイラオプションといったこれらのパラメータを手動で設定をしながらテストを行うことができ，本フレームワークを用いて構成したパラメータでのテスト結果を取得しながら調節が可能である．また，パラメータの組み合わせについてのいくつかの情報をファイルとして開発者が本フレームワークに与えることにより，ファイルで示された組み合わせパターン情報を利用して開発者が利用したいカーネル関数に合わせたパラメータのチューニングを行うことが可能である．具体的には，これらのパラメータ情報に基づいてテストを自動的に行い，そのテスト結果を基により良い性能を発揮することができるパラメータを半自動で選択する．

このように，本フレームワークを用いることによって，様々なハードウェア構成やソフトウェア設定の基でエージェントのプランニング処理に対してテストを行うことが可能である．しかしながら，考えられるすべての実行環境のために様々なパラメータを手動で評価しセットすることは開発者にとって大きな負担がかかる可能性がある．そこで我々は，本フレームワークをネットワーク上に拡張し，ネットワークにつながれた複数台のコンピュータ上で同時にテストなどの計測を行えるように拡張する．テストシナリオを実行環境に送ることによって，フレームワークの管理コンソールがその情報に基づき実行環境のマネージメントを行う．各々の実行環境は，テストシナリオにしたがって，テストを行いテスト結果を評価していくことによってパラメータチューニングを行っていく．テスト結果は，最後にまとめられ，テストシナリオに沿った最適なパラメータを各々の実行環境ごとに取得することが可能となる．

利用するアルゴリズムによっては，1 つの GPU 内コアに行わせる処理が複雑すぎるなどの理由で，正しく処理を行うことができない可能性や，効率的に処理を行うことができない可能性が考えられる．そこで，本研究では，どの程度のシミュレーション規模なら，現在利用する GPU 上で正常に処理を行うことができるかを検討するための機能の 1 つとして，シミュレーション対象とする道路網を一時的に任意の大きさに拡大，縮小する機能を実装する．

表 2 メモリ配置最適化をした場合の計算処理性能
Table 2 Comparison on Performance with memory-assignment optimization

	GeForce 8800GT	GeForce 320M
dijkstra (optimized)	7517.88[msec]	15662.69[msec]
dijkstra (not optimized)	8726.93[msec]	30952.75[msec]
A*(optimized)	5818.73[msec]	12948.14[msec]
A*(not optimized)	6789.76[msec]	22900.87[msec]

4. 実装

我々は，3 章で提案したフレームワークに基づいたランタイムプラットフォームの実装を行った．**図 3** は，実装したランタイムプラットフォームの概観を表す．ランタイムプラットフォームでは，簡易的な交通シミュレーションや，OpenCL に基づいたコーディングが行われたエージェントの処理 (やそのテスト) を行うことが可能である．また，本フレームワークのためのサンプルコードセットとして 4 つの主な経路探索アルゴリズム，Dijkstra，A*，RTA*[18] および LRTA*[19] を実装した．

我々は，本フレームワークの有効性を示すために，経路探索を行うプランニングアルゴリズムを並列実行した際の性能を計測する予備実験を行った．本予備実験では，それぞれエージェントに出発地と目的地を与え，すべてのエージェントが出発地から目的地までの経路を求めるためにかかる処理時間を計測した．実験環境には，MacBook Pro(OS: OS X 10.8.5, CPU: 2.4 GHz Intel Core 2 Duo, compiler: gcc4.2.1 build 5658, GPU: NVIDIA GeForce 320M, memory: 8GB 1067 MHz DDR3) と Mac Pro(CPU:3.0Ghz Quad Core Xeon, GPU:GeForce 8800GT, memory:32GB 800 MHz DDR2) を利用した．

すでに文献 [20] では，Dijkstra，A*，RTA*，LRTA*アルゴリズムを本ランタイムプラットフォーム上で実行した場合について，例えば，MacBook Pro ではエージェント数が 256 以下のとき，エージェント数が減少してもほぼ処理時間が変わらないことを観測している．たとえば，多数ある GPU コアを用いて並列的にプランニング処理を実行できているかどうかの確認ができる．

表 2 は, 本フレームワークを利用してカーネルコード中のメモリ配置の最適化を行った場合と行わない場合との実行結果の比較を示したものである．この比較では，384 のノードを持つ地図データ上で 10000 エージェントを同時に稼働させた．ここで，GeForce 320M は，本フレームワー

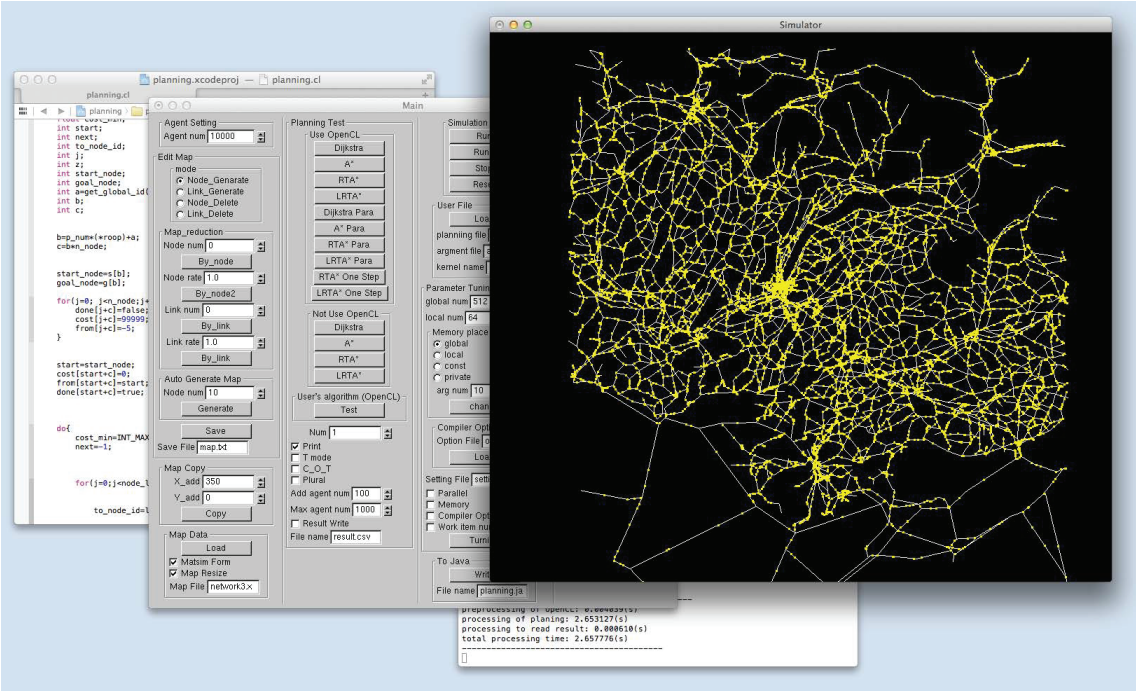

図 3 実装環境の概観
Fig. 3 The overview of execution environment

クによる最適化をした効果が大きく出た．一方，GeFroce 8800GT では，メモリ最適化をした影響をあまり大きく受けていないことがわかる．またこの実験を行うために筆者らが調節したコードを実行した際，GeForce 320M では調節前のコードに比べて調節後のコードを利用して処理を行った方が処理時間が短かったが，GeForce 8800GT では，調節を行う前のコードの方が処理時間が短いという結果となった．このように機種毎に調節した際の効果が異なることから，その影響を容易に観測できることは有益である．

表 3 に，もう 1 つの最適化に関する比較結果を示す．ここでは処理に対するメモリブロックの割り当て (すなわち，OpenCL の中のローカルアイテム数とグローバルアイテム数) の最適化を行った．エージェント数を 10000 として，3 つの地図データを用いて最適化後と最適化前の速度の比較を行ったところ，18.7%から 37.6%の速度の向上が見られた．このように，本フレームワークを用いていくつかのパラメータパターンに従ってテストを行うことによって，OpenCL のパラメータのチューニングを効率的に行うことが可能となる．

本フレームワークを利用すれば，GPU を利用した処理の処理時間の計測などを行うことができるが，チューニングをより効率的に行うためには，GPU による計算のコア単位などでの詳細な動きを分析できることが望ましい．GPU による計算の詳細な動きを分析するために，それぞれの GPU ごとに固有に用意されたプロファイラと連携させる機構の実装は今後の課題である．

5. おわりに

本論文では，シミュレーション開発者がシミュレーションプログラム内で効率的に GPGPU の演算コアを利用するために，GPGPU プログラミングを行う際のコーディングおよびパラメータチューニングを補助するためのフレームワークについて提案した．我々は、コーディングおよびパラメータチューニングを支援するために，ランタイム・プラットフォームを実装した．OpenCL プログラミングモデルに従ってエージェントのプランニング処理などを作成すれば，本フレームワークを利用することによって各 GPU の種類やパラメータの違いによる特性の分析にかかる負担を軽減可能である．

今後の課題としては，本研究で作成したフレームワークによって，どの程度スケーラビリティの改善に寄与することが可能であるかを，具体的なシミュレーション問題における改善事例などを通じて評価することがある．また，処理の高速化として，複数の GPU を同時に組み合わせて使用するような場面への適用を容易にすることも考えられる．このような場面では，あらゆる実行環境の組み合わせを実環境として準備することは容易ではないため，ある程度の典型的な構成の機材とそれらから事前に収集した実行特性を加味して，効果的な環境を実現する機材の組み合わせを，それらを準備することなく事前に検証できるようなフレームワークの実現も必要になると考えられる．

現状では，それぞれの実行環境に対する最適なパラメー

表 3　メモリブロックの割り当てを最適化した場合の計算処理性能
Table 3 Comparison on Performance memory block divisions optimization

	NODE:384	NODE:768	NODE:1536
dijkstra (optimized)	6396.57[msec]	23613.38[msec]	105046.06[msec]
dijkstra (not optimized)	8726.93[msec]	32264.39[msec]	141630.13[msec]
RTA*(optimized)	85.55[msec]	141.20[msec]	302.93[msec]
RTA*(not optimized)	105.25[msec]	184.20[msec]	485.35[msec]

タ設定を調査するために，すべての可能な限りのパラメータ組み合わせパターンでテストを実行する必要がある．パラメータの組み合わせの中から限られた組み合わせを選択し，効率的にテストや評価をするために，Multi-Armed Bandits 問題を解くアルゴリズムの適用などの可能性が考えられる [21][22]．このようなアルゴリズムを利用して，効率的にパラメータ最適化を行うことは，今後の課題である．

参考文献

[1] Balmer, M., Meister, K., Rieser, M., Nagel, K. and Axhausen, K.: Agent-Based Simulation of Travel Demand: Structure and Computational Performance of MATSim-T, *2nd TRB Conference on Innovations in Travel Modeling* (2008).
[2] 山下倫央，岡田　崇，野田五十樹：大規模群集流動の制御に向けたシミュレーション環境の構築，*Joint Agent Workshop and Symposium(JAWS 2012)* (2012).
[3] Tsai, J., Fridman, N., Bowring, E., Brown, M., Epstein, S., Kaminka, G., Marsella, S., Ogden, A., Rika, I., Sheel, A., Taylor, M. E., Wang, X., Zilka, A. and Tambe, M.: ESCAPES - Evacuation Simulation with Children, Authorities, Parents, Emotions, and Social Comparison, *Proc. International Conference on Autonomous Agents and Multiagent Systems(AAMAS 2011)*, pp. 457–464 (2011).
[4] Kanamori, R., Morikawa, T. and Ito, T.: Evaluation of Special Lanes as Incentive Policies for Promoting Electric Vehicles, *Proc. The 1st International Workshop on Multi-Agent Smart Computing(MASmart 2011)*, pp. 45–56 (2011).
[5] de la Hoz, E., Marsá-Maestre, I., López-Carmona, M. A. and Pérez, P.: Extending MATSim to Allow the Simulation of Route Coordination Mechanisms, *Proc. The 1st International Workshop on Multi-Agent Smart Computing(MASmart 2011)*, pp. 1–15 (2011).
[6] Nakajima, Y., Yamane, S. and Hattori, H.: Multi-Model Based Simulation Platform for Urban Traffic Simulation, *13th International Conference on Principles and Practice of Multi-Agent Systems(PRIMA 2010)*, pp. 228–241 (2010).
[7] Navarro, L., Corruble, V., Flacher, F. and Zucker, J.-D.: A Flexible Approach to Multi-level Agent-Based Simulation with the Mesoscopic Representation, *Proc. International Conference on Autonomous Agents and Multiagent Systems(AAMAS 2013)*, pp. 159–166 (2013).
[8] Yamamoto, G., Tai, H. and Mizuta, H.: A Platform for Massive Agent-Based Simulation and its Evaluation, *Proc. International Conference on Autonomous Agents and Multiagent Systems(AAMAS 2007)* (2007).
[9] Caggianese, G. and Erra, U.: GPU Accelerated Multi-Agent Path Planning Based on Grid Space Decomposition, *Proceedings of the International Conference on Computational Science*, pp. 1847–1856 (2012).
[10] Vineet, V., Harish, P., Patidar, S. and Narayanan, P. J.: Fast Minimum Spanning Tree for Large Graphs on the GPU, *Proceedings of the Conference on High Performance Graphics 2009(HPG '09)*, New York, NY, USA, ACM, pp. 167–171 (2009).
[11] AlSaber, N. and Kulkarni, M.: SemCache: Semantics-Aware Caching for Efficient GPU Offloading, *Proceedings of the 27th ACM International Conference on Supercomputing(ICS '13)*, New York, NY, USA, ACM, pp. 421–432 (2013).
[12] Kofler, K., Grasso, I., Cosenza, B. and Fahringer, T.: An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning, *Proceedings of the 27th ACM International Conference on Supercomputing(ICS '13)*, New York, NY, USA, ACM, pp. 149–160 (2013).
[13] Grasso, I., Pellegrini, S., Cosenza, B. and Fahringer, T.: libWater: Heterogeneous Distributed Computing Made Easy, *Proceedings of the 27th ACM International Conference on Supercomputing(ICS '13)*, New York, NY, USA, ACM, pp. 161–172 (2013).
[14] Holewinski, J., Pouchet, L.-N. and Sadayappan, P.: High-Performance Code Generation for Stencil Computations on GPU Architectures, *Proceedings of the 26th ACM international conference on Supercomputing(ICS '12)*, New York, NY, USA, ACM, pp. 311–320 (2012).
[15] Huo, X., Krishnamoorthy, S. and Agrawal, G.: Efficient Cheduling of Recursive Control Flow on GPUs, *Proceedings of the 27th ACM International Conference on Supercomputing(ICS '13)*, New York, NY, USA, ACM, pp. 409–420 (2013).
[16] Vasudevan, R., Vadhiyar, S. S. and Kalé, L. V.: G-Charm: an Adaptive Runtime System for Message-Driven Parallel Applications on Hybrid Systems, *Proceedings of the 27th ACM International Conference on Supercomputing(ICS '13)*, New York, NY, USA, ACM, pp. 349–358 (2013).
[17] Khronos OpenCL Working Group: *The OpenCL Specification Version: 1.2 Document Revision: 19* (2012).
[18] Korf, R. E.: Real-Time Heuristic Search, *Artif. Intell.*, Vol. 42, No. 2-3, pp. 189–211 (1990).
[19] 石田　亨，新保　仁：実時間探索による経路学習，人工知能学会誌，Vol. 11, No. 3, pp. 411–419 (1996).
[20] Sano, Y. and Fukuta, N.: A GPU-Based Framework for Large-Scale Multi-Agent Traffic Simulations, *Proc. 2nd IIAI International Conference on Advanced Applied Informatics (IIAI AAI2013)* (2013).
[21] Tran-Thanh, L., Chapman, A. C., Rogers, A. and Jennings, N. R.: Knapsack Based Optimal Policies for Budget-Limited Multi-Armed Bandits, *AAAI* (2012).
[22] Robbins, H.: Some Aspects of the Sequential Design of Experiments, *Bull. Amer. Math. Soc.*, Vol. 58, No. 5, pp. 527–535 (1952).