

修士論文

題目

混雑時におけるGPU上の  
避難シミュレーションの高速化

指導教員

大野 和彦 講師

2023 年

三重大学大学院 工学研究科 情報工学専攻  
コンピュータアーキテクチャ研究室

林 匠己 (421M521)

# 混雑時における GPU 上の避難シミュレーション の高速化

林 匠己

## 内容梗概

近年、災害対策の強力なツールとしてマルチエージェントを用いた避難シミュレーションの需要が高まっている。しかし、避難シミュレーションはエージェント数が増えると計算コストが大幅に増加する。そのため、実用的な大規模シミュレーションを行うには高速化が必要である。

そこで近年では、高い演算性能を持つ Graphics Processing Unit (GPU) を利用した高速化が注目されている。GPU は、多数の計算コアを持っているため、各エージェントの動きを並列処理により計算でき、高速化が可能となっている。しかし、実用的な大規模シミュレーションを行うためにはさらなる高速化が必要となっている。

本研究では、Node-Link model というグラフマップ上での避難シミュレーションを対象とする。このモデルでは、マップをノード（交差点）とエッジ（道）により表現する。各エッジは距離に比例した、エージェントが通過に要する時間を表すコストを持つ。各エージェントは現在地からゴールまでのコストが最小となる最短経路を計算し、その経路に沿ってノードから次に進むノードを選択する。また、コストはエッジ上のエージェント数によって増減する。これにより、多数の避難者による混雑度の変化が生じ、最短経路の変化が再現できる。このモデルの利点は、経路探索や混雑が起きているかの評価コストが小さいため、大規模シミュレーションに適していることである。

避難シミュレーションを GPU 上で並列化する場合、一般的には各エージェントをそれぞれスレッドに割り当て並列処理を行う。このため、各エージェントの経路探索はそれぞれ独立に行われ、結果的に同一経路の計算が重複して行われる。また、各ステップごとに各エッジが混雑によりコストが増加しているなら、各エージェントの経路が変化する可能性があるため、各エージェントで経路の再計算する必要がある。このような不要な計算は実行時間に大きな影響を与え、エージェント数が増えるにつれて実行時間は大幅に増加する。

そこで本研究では、このような冗長な計算を削減する高速化手法を提案する。避難が開始すると、各ノードをスレッドに割り当て、ゴールまでの

最短経路を並列処理であらかじめ計算する。そして、各エージェントは現在いるノードの経路で次に進むノードを参照し移動する。次のノードに到達すると、そこからさらに次に進むノード参照し移動する。これを繰り返すことでゴールを目指す。また、一定ステップごとに混雑が発生していると判断されたなら、各ノードで経路を再計算する。これにより従来手法ではエージェント数分の並列処理を行い、冗長な計算も含んでいたが、ノード数分で計算できるようになり、冗長な計算も削減できる。

評価方法は、ノード数が64のマップ上で避難者エージェント数を変化させシミュレーションを実行した。エージェントは512から524288の間で変化させた。提案手法による実行時間は従来手法と比較するとエージェント数が32768以下の時は提案手法による改善率が安定せず、従来手法より実行時間が延びるケースもあった。しかし、エージェント数がそれ以上になるとエージェント数の増加に応じて従来手法より実行時間が短縮され、エージェント数が524288の場合に、約1/3となった。したがって、本手法は大規模なシミュレーションで実行速度を大きく改善できるといえる。

# Acceleration of Congested Evacuation Simulation on GPU

Takumi Hayashi

## Abstract

In recent years, the demand for multi-agent evacuation simulations has been increasing as a powerful tool for disaster management. However, the computational cost of evacuation simulation increases significantly with the number of agents. Improving the computation performance is necessary for practical large scale simulations.

Therefore, the use of Graphics Processing Units (GPUs), which have high computational performance, has been attracting attention in recent years. Since GPUs have a large number of computation cores, the behavior of each agent can be computed in parallel, which accelerates the simulation. However, further acceleration is needed for practical large-scale simulations.

In this study, we focus on evacuation simulations on a graph map called the Node-Link model. In this model, the map is represented by nodes (intersections) and edges (roads). Each edge has a cost proportional to its length, which represents the time required for an agent to pass through the edge. Each agent calculates the shortest path from the current location to the goal with the part of the minimum cost, and selects the next edge from the node along the path. The cost increases or decreases depending on the number of agents on the edge. This allows the model to reproduce changes in the shortest path due to changes in congestion caused by a large number of evacuees. The advantage of this model is that it is suitable for large-scale simulations because the cost of route finding and evaluation of congestion is small.

When an evacuation simulation is parallelized on a GPU, each agent is generally assigned to a thread in parallel. For this reason, each agents are executed independently, resulting in redundant computation of the same route. If the cost is increasing due to congestion at each edge at each step, the route for each agent may change. In this case, each agent needs to recalculate the route. Such unnecessary calculation has a large impact on the execution time, and as the number of agents increases, the

execution time The execution time increases significantly as the number of agents increases.

Therefore, we propose a speed-up method to reduce such redundant computations. When the evacuation starts, each node is assigned to a thread and the shortest path to the goal is pre-computed in parallel. Then, each agent moves to the next node by referring to the path of the node where it is currently located. When the agent reaches the next node, it refers to the next node and moves on. This process is repeated to reach the goal. If congestion is detected at a certain step, the route is recalculated at each node. This reduces redundant calculations.

As the evaluation of the proposed method, we performed evacuation simulation on a map with 64 nodes. We changed the number of agent in the range of 512 to 524288. Compared with the conventional method, the improvement rate of the proposed method is not stable with 32768 or less agents, and the execution time of the proposed method is longer than that of the conventional method in some cases. However, with 32768 or larger agents, the execution time is reduced according to the increase in the number of agents, and it is about 1/3 of the conventional method when the number of agents is 524288. Therefore, this method can greatly improve the execution speed in large-scale simulations.

# 目次

1	はじめに	1
2	研究背景	4
2.1	マルチエージェントシミュレーション	4
2.2	GPU と CUDA	5
3	避難シミュレーションのモデル	8
3.1	Node-Link model	8
3.2	混雑によるコストの変化	9
3.3	エージェントの移動処理	10
3.4	エージェントの行動決定処理	11
3.5	従来手法	12
4	提案手法	15
4.1	各ノードで経路探索	15
4.2	提案手法の流れ	16
5	評価	19
5.1	評価環境と評価方法	19
5.2	評価結果	20
5.3	考察	21
6	まとめと今後の課題	24
	謝辞	26
	参考文献	27

## 図 目 次

2.1 エージェントの相互作用 . . . . .	5
2.2 CUDA による並列処理の流れ . . . . .	7
3.3 Node-Link model . . . . .	9
3.4 エージェントの移動処理 . . . . .	11
3.5 従来手法のフローチャート . . . . .	13
3.6 従来手法の問題点の例 . . . . .	14
4.7 提案手法による計算方法の例 . . . . .	16
4.8 提案手法のフローチャート . . . . .	18
5.9 評価結果 . . . . .	21
5.10 グループ化による計算量の削減方法 . . . . .	23

## 表 目 次

5.1 評価環境 . . . . .	19
--------------------	----



# 1 はじめに

近年、災害対策の強力なツールとして様々なモデルを用いた避難シミュレーション [1] が行われており、需要が高まっている。その中で、マルチエージェントシミュレーションはエージェントと呼ばれる個体を相互作用させることで複雑な現象をボトムアップ的に創出している。そして、マルチエージェントシミュレーションを用いた避難シミュレーションでは避難者をエージェントとして表している。しかし、避難シミュレーションはエージェント数が多くなると計算コストが大幅に増加する。そのため、実用的な大規模シミュレーションを行うには高速化が必要である。

この問題に対し、高い演算性能を持つ Graphics Processing Unit (GPU) を利用した高速化が注目されている。CPU は逐次処理能力に優れる一方で数個程度のコアしか持たないのに対し、GPU は、単体性能で劣るが多数の計算コアを持っているため、並列処理能力が優れている。つまり、避難シミュレーションにおいて各エージェントの動きを並列処理により計算でき、高速化が可能となっている。しかし、単純な並列処理では実用的な大規模シミュレーションを行うための実行時間はまだまだ大きく、さらなる高速化が必要となっている。

本研究では、シミュレーションのマップに Node-Link model というグ

ラフ表現を使うことを前提とする。このモデルでは、マップをノード（交差点）とエッジ（道）により表現する。各エッジは距離に比例した、エージェントが通過に要する時間を表すコストを持つ。各エージェントは現在のノードからゴールノードまで移動を行う。このモデルの利点は、経路探索や混雑が起きているかの評価コストが小さいため、大規模シミュレーションに適していることである。また、本研究ではコストはエッジ上のエージェント数によって増加する。これにより、多数の避難者による混雑度の変化が生じ、最短経路の変化が再現できる。エージェントの経路選択として、一般的なマルチエージェントシミュレーションにおいて、避難者の行動は様々なロジックをもとに行う。しかし、ロジックはシミュレーションごとに異なるため、本研究では様々なロジックについては考慮せず、各エージェントは常に変化するコストも踏まえて最短経路を進み続けるとする。上記のように実際のエージェントのロジックを実装したマルチエージェントシミュレーションの場合に本手法が適用できるかについては3章で議論する。

避難シミュレーションを GPU 上で並列化する場合、一般的には各エージェントをそれぞれのスレッドに割り当てて並列処理を行う。このため、各エージェントの経路探索はそれぞれ独立に行われ、結果的に同一経路の計

算が重複して行われる。また、各ステップごとに各エッジのエージェント数を調べ、混雑によるコストの変動を行う。これにより各エージェントの経路が変化する可能性があるため、各エージェントはノードに到達するごとに経路を再計算する必要がある、計算が重複して行われる。このような冗長な計算は実行時間に大きな影響を与え、エージェント数が増えるにつれて実行時間は大幅に増加する。

そこで本研究では、このような冗長な計算を削減する高速化手法を提案する。各ノードをスレッドに割り当て、ゴールまでの最短経路を並列処理であらかじめ計算する。そして、各エージェントは現在のノードから、対応した経路を参照し次に進むノードを決定する。これをノードに到達するごとに行い、ゴールを目指す。また、各ステップごとに各エッジが混雑によりコストが変動した場合、各ノードで経路を再計算する。

本論文は次のように構成されている。まず2章で、マルチエージェントシミュレーションとGPUの特徴について述べる。3章では、本研究で使用するマップとエージェントのモデルについて述べ、従来研究での問題点を挙げる。4章では、提案手法を説明する。5章では、提案手法の評価方法、評価結果、考察を述べる。最後に6章でまとめと今後の課題について述べる。

## 2 研究背景

### 2.1 マルチエージェントシミュレーション

周囲の状況に基づいて一定のルールのもとで自律的に行動する主体をエージェントという。そして、マルチエージェントシミュレーション [2] は個々のエージェントの相互作用をシミュレーションしたものとなっている。例として避難シミュレーションの他に、交通の流れ [3] や鳥の群れの動き [4] といった様々なシミュレーションがある。マルチエージェントシミュレーションでは、エージェントがそれぞれ行動規則を持っており、自身の状態や周囲の状況などによって行動が変化する。このように相互作用させることで複雑な現象をボトムアップ的に創出している。図 2.1 にエージェントの相互作用の例を示しており、赤色のエージェントに注目している。エージェントは視野を持っており、その視野内にいる他のエージェントや周りの環境と相互作用を行っている。注目エージェントが交差点において、エージェント数の多い方に追従する行動規則を持っている場合、視野内でエージェント数が多い方向へ進む。実用的な避難シミュレーションでは、実際の避難者の振る舞いを模倣するような行動モデルやルールが与えられるが、これはシミュレーションごとに異なる。本研究はマルチエージェントシミュレーションの実行効率を改善することが目的であるため、

エージェントの行動については単純化したものを用い, 詳しくは3章で説明する.

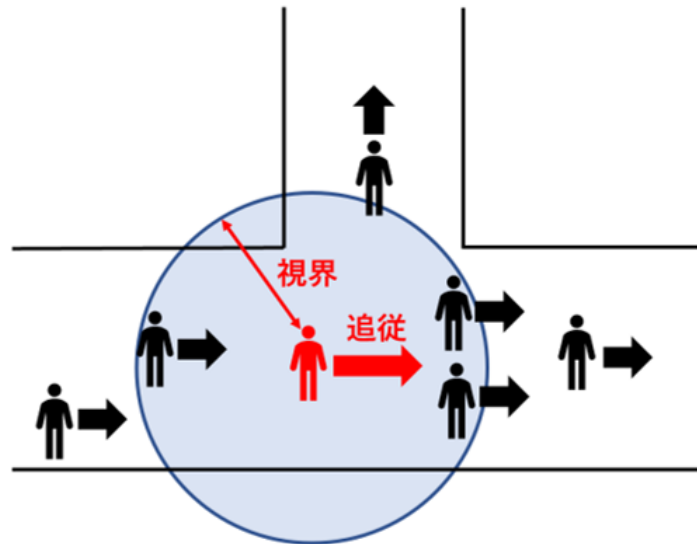


図 2.1: エージェントの相互作用

## 2.2 GPU と CUDA

GPU[5] は多数の計算コアを持ったプロセッサであり, 並列処理を得意としている. また, 多数のコアを GPU 内の Streaming Multiprocessor(SM) によって管理しており, 1 つの SM には最大 1024 スレッドが割り当てられ, 並列に実行される.

CUDA[6] は, NVIDIA 社が提供する GPU 向けの統合開発環境であり, C 言語を拡張した文法とライブラリ関数を用いて開発することが出来る.

図 2.2 に CUDA による並列処理の流れについて示す。CPU 側はホスト、GPU 側はデバイスと呼ばれ、それぞれのメモリ上で処理を行い、必要に応じてデータ転送を行う。処理の流れを、以下に示す。

1. ホスト側からデバイス側に並列処理を行いたいデータを転送する。
2. ホスト側で GPU メモリの確保を行う。
3. デバイス側のメモリを用いて並列処理を行う。
4. 処理を行ったデータをホスト側に転送する。

この一連の流れを行うことで並列処理をすることが出来る。避難シミュレーションを GPU 上で並列化する場合、一般的には各エージェントをそれぞれのスレッドに割り当て、行動決定や移動を並列処理で行う。このため、CPU による逐次的な処理を行うシミュレーションより高速化が可能となっている。

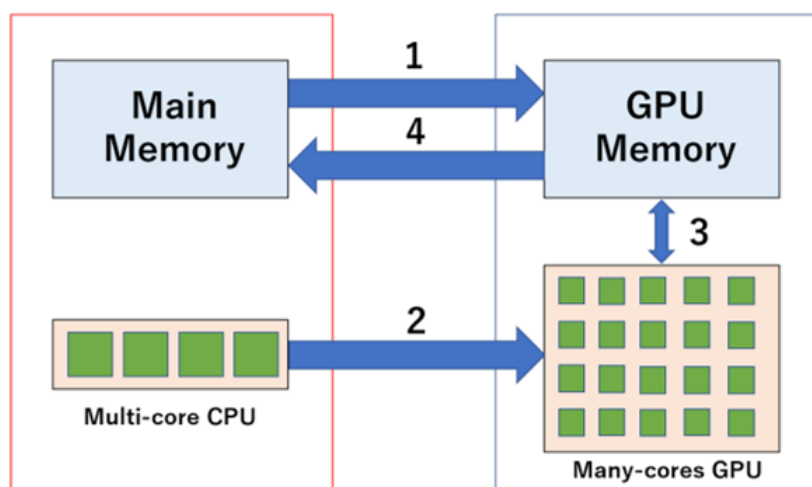


図 2.2: CUDA による並列処理の流れ

### 3 避難シミュレーションのモデル

#### 3.1 Node-Link model

本研究では, Node-Link model[7] というグラフマップを使用し, 図 3.3 に示す. このモデルでは, マップをノード（交差点）とエッジ（道）により表現する. 各エッジは距離に比例した, エージェントが通過に要する時間を表すコストを持つ. また, 各エージェントはいずれかのノード上にのみ存在するが, あるノードからそれに隣接したノードに移動を行う場合, その間のエッジ上を移動しているとする.

各エージェントは現在地から避難ノードであるゴールまでのコストが最小となる最短経路を計算し, 次に進むエッジを選択する. これを次のノードに到達するごとに行う. そして本研究では, 各ステップで各エッジ上を移動しているエージェント数を調べ, エージェント数によって元のコストから増加させる. 具体的なコストの変化方法については 3.2 節で詳しく説明する.

このモデルの利点は, コストが変化することにより, 多数の避難者による混雑度の変化が生じ, 最短経路の変化が再現できることである. これにより, 混雑している道は選択されにくいという現実世界に近いシミュレーションを行うことができる. さらに, 経路探索やコスト変化の評価コスト



が小さいため、大規模シミュレーションに適している。

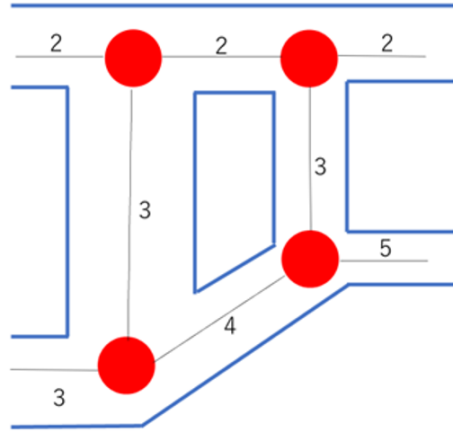


図 3.3: Node-Link model

### 3.2 混雑によるコストの変化

各エッジはコストを持っており、そのコストは各ステップでノード上のエージェント数により変化する。エッジ  $e_i$  のコスト増加量  $C_{add}$  は以下の式で表される。

$$C_{add} = \frac{pop(e_i)}{(len(e_i) * width(e_i)) * C}$$

ただし、 $len$ ,  $width$  はそれぞれエッジ  $e_i$  の長さと幅であり、 $pop$  はエッジ  $e_i$  上のエージェント数である。また、 $C$  は混雑度によるコスト増加の重み係数である。上記の式において  $(len(e_i) * width(e_i)) * C$  の値はユーザが使用するシミュレーションごとに異なるため、本研究ではこの値を 10 と

する. つまり, 各エッジ上においてエージェント数が10 ごとにコストが1 増加する.

### 3.3 エージェントの移動処理

本研究のシミュレーションでは, シミュレーション開始時に, ノードの一つ ( $N_g$ ) が避難先のゴールとして設定される. そして, エージェントは以下のルールで移動する.

- 各エージェントは, 現在位置から  $N_g$  を目指して移動を繰り返し,  $N_g$  に到達すると消滅する.
- シミュレーション開始時, あるエージェントがノード  $N_s$  の上にいるとする. エージェントは次に移動するノード  $N_{next}$  を決定し, 移動を開始する.
- $N_{next}$  に到達すると,  $N_{next}$  を新たに  $N_s$  として, 次に移動するノードである  $N_{next}$  を決定する. これを,  $N_g$  に到達するまで繰り返す.
- $N_{next}$  が決定したエージェントは, 移動するエッジのコスト分のステップが経過した後,  $N_s$  から  $N_{next}$  に移動する.
- 移動先を決定するのは移動中以外の場合で, 移動中のエージェント

が移動先を変更する (たとえば今のエッジを引き返す) ようなことは  
ないものとする.

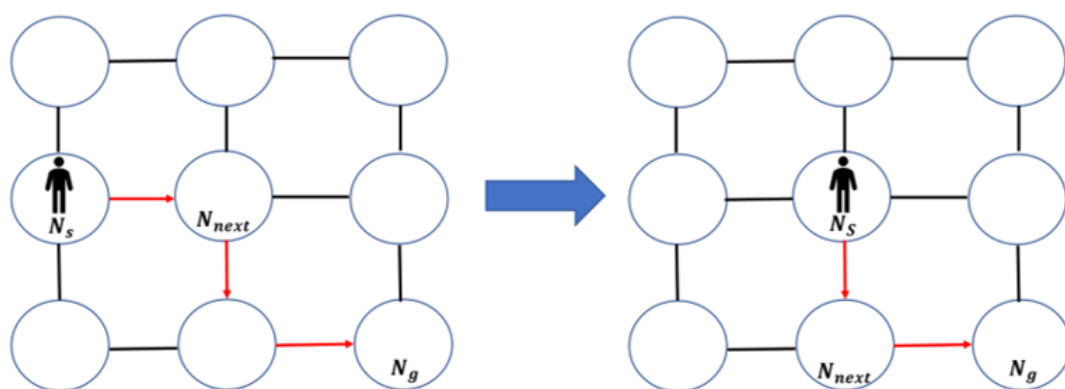


図 3.4: エージェントの移動処理

### 3.4 エージェントの行動決定処理

移動先が決まっていないエージェントは, 次に移動するノード  $N_{next}$  を  
以下の処理 1 と処理 2 により決定する.

#### 処理 1

現在ノードからゴールまでの最短経路をダイクストラ法 [8] で求め,  
隣接ノードの中から, この経路上にあるものを  $N_c$  とする.

#### 処理 2

確率  $p$  で  $N_c$  を  $N_{next}$  とする. それ以外の場合, 隣接ノードのうち直

前にいたノードと  $N_c$  を除いた残りのノードの中から等確率で一つ選び, それを  $N_{next}$  とする。

処理 2 はマルチエージェントシミュレーションの特徴である, 周りの影響を受けてエージェントが最短経路を通らない可能性があることを簡易的であるが表現している。

### 3.5 従来手法

従来の並列計算手法による避難シミュレーションの流れを図 3.5 に示す。シミュレーションが開始すると, 3 章で述べたエージェントの行動決定処理と移動処理において, エージェント  $A_i$  を担当するスレッド  $T_i$  が並列に実行する。その後, コスト変化処理を行う。これの一連の流れを 1 ステップとする。そして, 行動決定処理に戻り, 再度計算が必要なエージェントの処理を行う。全てのエージェントがゴールに到達 (避難完了) するとシミュレーションは終了する。

従来手法の問題点の例を図 3.6 に示す。あるステップにおいて, 各エージェントは行動決定処理により, ノード G までの経路を探索している。各エージェントは, 経路探索をそれぞれ独立でおこなうため, ノード 1 とノード 2 にいるエージェントはそれぞれ同一経路の計算が重複して行われてい

る. さらに各ステップにおいて, 各ノードに到達するごとにこのような同一経路の計算が行われる. このような冗長な計算は, 実行時間に大きな影響を与え, エージェント数が増えるにつれて実行時間は大幅に増加する.



図 3.5: 従来手法のフローチャート

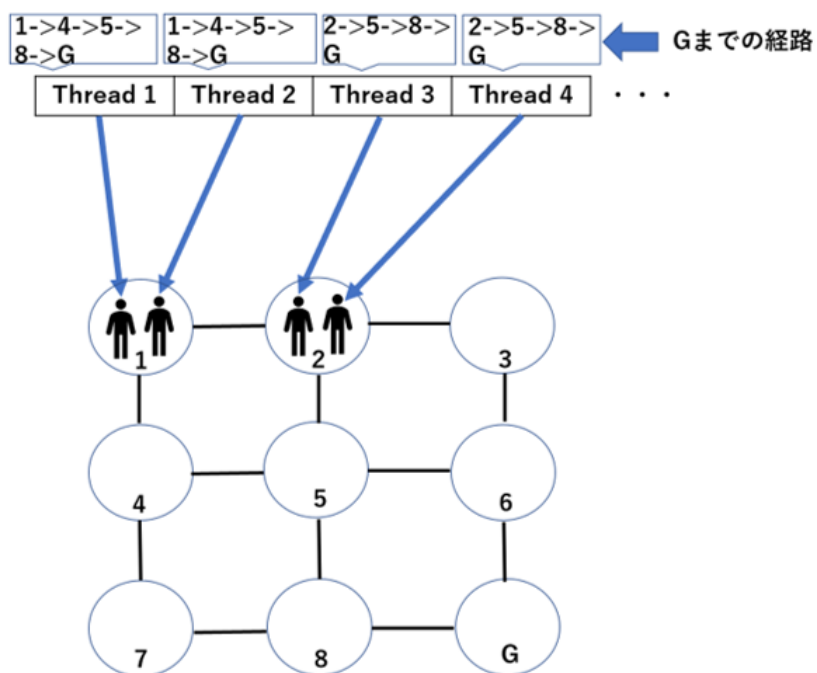


図 3.6: 従来手法の問題点の例

## 4 提案手法

### 4.1 各ノードで経路探索

本研究では、従来手法における冗長な計算を削減する高速化手法を提案する。この問題点に対して、エージェントの行動決定処理で各エージェントの経路探索を行わず、各ノードで経路探索を行う。したがって、ノード  $N_i$  にスレッド  $T_i$  を割り当て並列処理を行うことで計算量を削減する。そして、得られた最短経路から各ノードから次に移動するノード  $N_{next}$  を決定する。

図 4.7 に提案手法による計算方法の例を示す。図 3.6 と同様にノード G までの経路探索を行っている。シミュレーションが開始するとダイクストラ法により、ノード 1 とノード 2 からノード G までの経路探索をそれぞれ行う。最短経路の計算が終わると従来手法の行動決定処理と同様に、隣接ノードの内、この経路上にあるものを  $N_{next}$  とする。つまり、ノード 1 とノード 2 の  $N_{next}$  はそれぞれノード 4 とノード 5 になる。この手法により、従来手法にあった重複した経路探索が無くなるため、冗長な計算を削減することができる。また、経路探索を行う数もエージェント数からノード数に変更されるため、大幅な計算量の削減が可能になる。

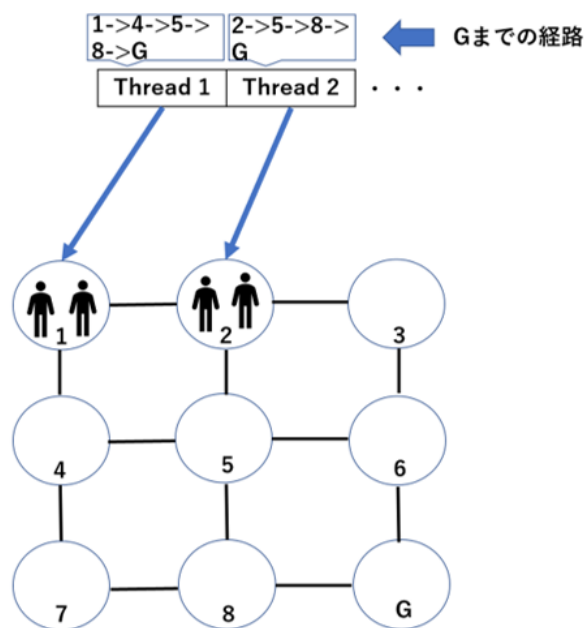


図 4.7: 提案手法による計算方法の例

## 4.2 提案手法の流れ

図 4.8 に提案手法の流れを示す. シミュレーションが開始すると, 4.1 節で述べた各ノードでの経路探索を行う. そして, 行動決定処理と移動処理を行う. ただし, 行動決定処理において, 各エージェントは現在いるノードに対応した  $N_{next}$  を参照して処理を行う. その後, コストの変化処理を行う. これが終わると, コストが変化しない場合は経路の変更が無いいため移動処理に戻るが, 変化した場合は各ノードの  $N_{next}$  が変化する可能性があるため, 再度各ノードで経路探索を行う.

本手法においてノード数とエージェント数が異なるため, 並列処理を行



う際は GPU のスレッドの切り替えが必要である。シミュレーションがスタートすると、各ノードで経路探索を行うためにスレッドを各ノードに割り当てる。そして、経路探索が終わると、各エージェントにスレッドの割り当てを切り替る。この割り当てはコストが変化するまで行い、変化した際は、各ノードに再度割り当ててシミュレーション開始時と同様に経路探索を行う。そして経路探索が終わると、各エージェントに切り替える。これをシミュレーションが終了するまで繰り返し行うことにより、エージェントの経路と移動処理を並列で実行する。

従来手法は同一経路の冗長な計算を行うため冗長な計算が発生するという欠点があるが、シミュレーションが進むごとに避難完了エージェントが増え、経路探索を行うエージェント数が減る。このため、計算する経路探索の数がシミュレーションが進むにつれて減少していくという利点がある。反対に提案手法は全てのノードの経路計算ため、常にノード数分の経路探索を行うという欠点があるが、同一経路による冗長な計算が削減できるという利点がある。ともに欠点と利点があるが、提案手法の方が従来手法と比べ、大幅に経路探索の数が減少すると考えられるため、高速化することができると思われる。

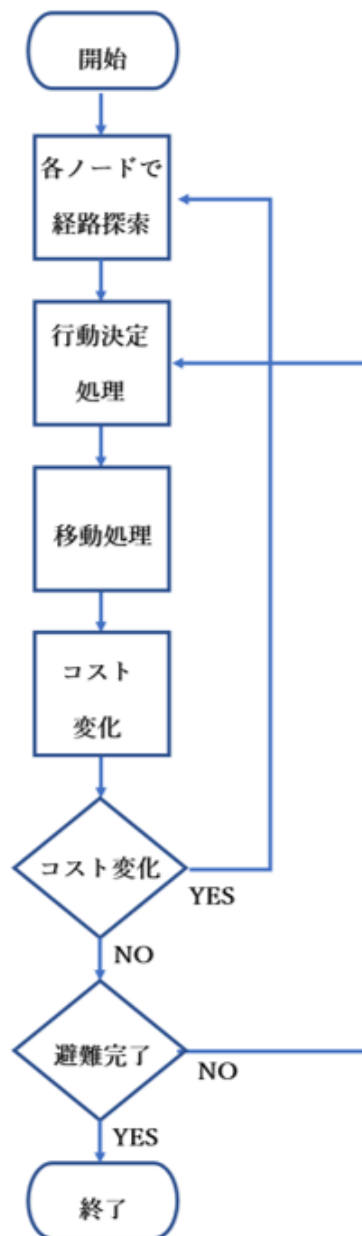


図 4.8: 提案手法のフローチャート

## 5 評価

様々な条件下でシミュレーションを実行し, 従来手法と提案手法による性能向上率を比較した. 5.1 節では, 評価環境と評価方法について述べる. 5.2 節では, 評価結果と考察について述べる.

### 5.1 評価環境と評価方法

避難シミュレーションに提案手法を実装し, シミュレーションの実行時間を測定し評価を行った. 評価環境は表 5.1 の通りである.

表 5.1: 評価環境

	環境
CPU Memory	Intel(R) Core(TM) i9-10900X 32GB
GPU Memory	GeForce RTX 3080 32GB

本実験においてノード数が64のマップ上で, 2000 ステップのシミュレーションを行った. また, エージェントの行動決定処理において, 確率  $p=0.99$  とした.

エージェント数を変更して評価を行った. エージェント数は  $2^9$  から  $2^{19}$  の間で変化させた. エージェントは各ノードにランダムで配置させた. また, すべてのエージェント数においてシミュレーションを 5 回ずつ行い,

その平均を評価結果とした。

## 5.2 評価結果

評価結果を図 5.9 に示す。これは、従来手法の実行時間を 1 としたときの提案手法の実行時間比を表したグラフである。従来手法と比較して、提案手法はエージェント数が 65536 の時に実行時間が小さくなり、524288 の時、最大で約  $1/3$  となった。これは提案手法により冗長な計算が削減されたためだと考えられる。しかし、エージェント数が 32768 以下の場合、提案手法による改善率が安定せず、従来手法より実行時間が大きくなった。これは、エージェント数が少ない場合、シミュレーションの終盤は避難者が少なくなり、各エージェントで最短経路を計算する方が各ノードで最短経路を計算するよりも計算コストが少なくなったと考えられる。さらに、提案手法による高速化よりオーバーヘッドの方が大きくなったことも考えられる。

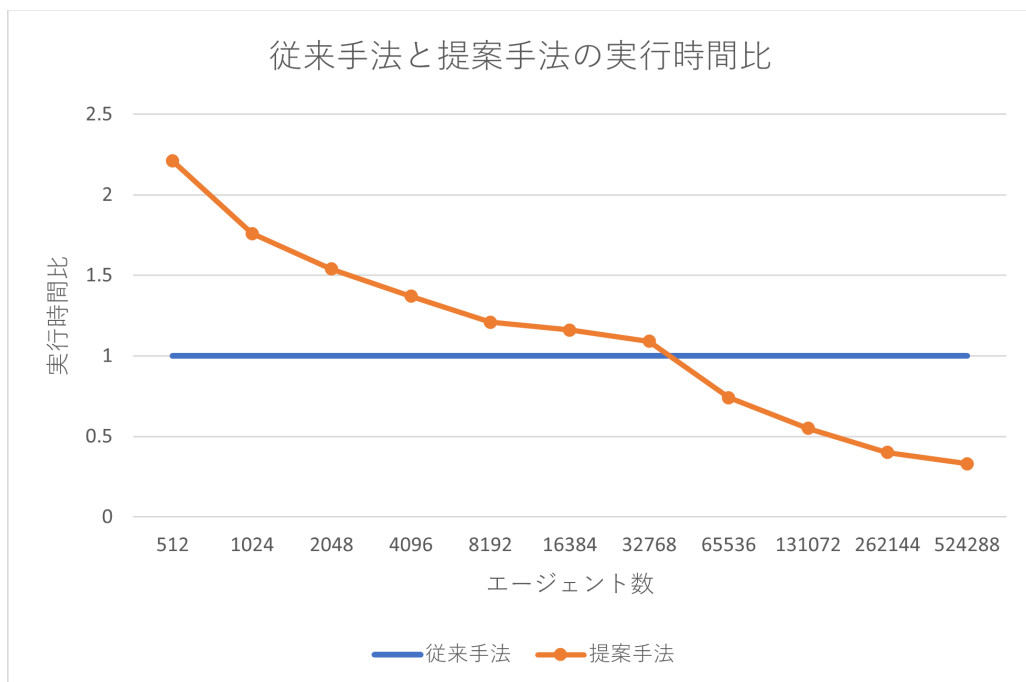


図 5.9: 評価結果

### 5.3 考察

本研究では, 提案手法において常にノード数分の経路探索を行った. しかし, シミュレーション終盤になるほどエージェントの通らないノードが多く存在する. つまり, この通らないノードからの計算を削減することにより, より高速化が可能になり, 少ないエージェント数のシミュレーションでも従来手法より高速化が可能になると考えられる. そして, これを実装した場合は, 有用性を示すためにもエージェントの初期配置を本実験で

はランダムに行ったが、あるノードに大量のエージェントが存在するなど  
の偏った配置のテストケースでの実験が必要だと考える。

本研究において、ランダムで最短経路から外れるなどのマルチエージェントシミュレーションを簡易的に表現したところがあるが、最短経路をあらかじめ計算しておくことでエージェントが多い場合、高速化されることが分かった。つまり、一度計算した経路を再利用することでエージェント数の増加に伴い大幅に実行時間が削減された。これを本来のマルチエージェントシミュレーションに当てはめると、各エージェントは様々な内部パラメータにより異なった経路を通して、ゴールを目指す。そして、同じパラメータの場合は同じ経路を通ることが考えられる。つまり、ある内部パラメータを持つエージェントにおいて一度計算した経路を記憶しておくことで、同じ内部パラメータを持つエージェントの経路探索に使用することで冗長な計算を削減でき、実行時間が短縮できると考えられる。

また、本研究では各ステップでエージェントの移動処理をそれぞれのエージェント行い、次のノードに移動する。これは、あるノードにエージェントが移動する場合、移動処理が同じため冗長となる。つまり、シミュレーションのステップ数が増えるごとに計算量が莫大になり実行時間が大幅に増加する。そこで、エージェントの移動をグループ化することで計算量

を削減する手法が考えられる。図 5.10 にグループ化による計算量の削減方法を示す。同じノード上に存在する同じ内部パラメータを持つエージェントは同じノードに移動すると考えられるため、内部パラメータごとにグループ化してまとめて移動することで移動処理を行う回数が削減される。これにより、オーバーヘッドによる影響が小さくなると考えられる。

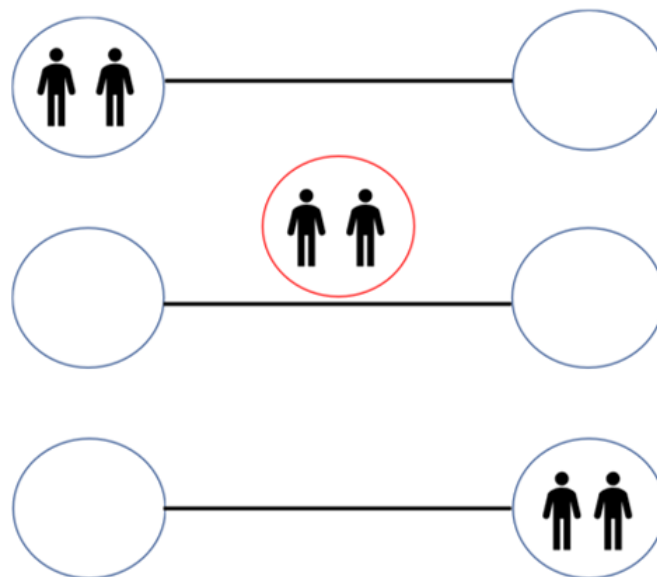


図 5.10: グループ化による計算量の削減方法

## 6 まとめと今後の課題

本論文では GPU 上の避難シミュレーションの高速化について、各ノードで最短経路を事前に計算する手法を提案した。これにより冗長な計算を削減でき、さらに混雑によるコストが変化の際も同様に計算出来るものとなっている。この提案手法において、エージェント数を 512 から 524288 の間で変化させ、従来手法と比較を行った。評価の結果、エージェント数が 32768 以下の時は提案手法による改善率が安定せず、従来手法より実行時間が大きくなった。しかし、エージェント数がそれ以上になるとエージェント数の増加に応じて従来手法より実行時間が短縮され、エージェント数が 524288 の場合に、約 1/3 となった。

今後の課題として、本手法を実際のマルチエージェントシミュレーション上で評価することが挙げられる。本研究では、マルチエージェントシミュレーションにおける経路選択を一定確率で最短経路から外れ、ランダムに次に進むノードを決定するという簡易的な方法で表現した。しかし、実際には周囲の状況により、現在より複雑な経路選択を行うため、モデルの改良が必要である。また、エージェント数が少ない避難シミュレーションにおいて本手法を適用する場合は、本手法の影響よりオーバーヘッドの影響の方が大きくなっているためオーバーヘッドの低減も課題として挙



げられる。

## 謝辞

本研究を行うにあたり、多数のご指導をいただきました大野和彦講師、高木一義教授、並びに深澤祐樹技術職員に深く感謝いたします。また、コンピュータアーキテクチャ研究室の学生には常に刺激的な議論を頂き、精神的にも支えられました。併せて感謝をいたします。

## 参考文献

- [1] 山下倫央, .et.al. “一次元歩行者モデルを用いた高速避難シミュレータの開発とその応用.” 情報処理学会論文誌 53-7(2012) 1732-1744
- [2] マルチエージェントシミュレーション, <https://mas.kke.co.jp/about> (2023-1-31 参照)
- [3] Helbing, Dirk. “Traffic and relate self-driven many-particle systems.” Reviews of modern physics 73. (2001): 1067.
- [4] Olfati-Saber, Reza. “Flocking for multi-agent dynamic systems: Algorithms and theory.” IEEE Transactions on automatic control 51.3(2006): 401-421
- [5] GPU, <https://developer.nvidia.com/cuda-zone> (2023-1-31 参照)
- [6] CUDA, <https://developer.nvidia.com/cuda-zone> (2023-1-31 参照)
- [7] 上田遼. “行動特性に着目した津波避難の分析と対策-人間と ICT の相互作用による安全避難の共創-.” 日本地震工学会論文集 17.4(2017): 4.140-4.169.

[8] ダイクストラ法, <https://nw.tsuda.ac.jp/lec/dijkstra> (2023-1-31 参照)