修士論文

題目

状態を用いたマルチエージェントシミュレーションのスケジューリング最適化

指導教員

大野 和彦　講師

2019 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・ソフトウェア研究室

417M519 廣田裕哉 (417M519)

内容梗概

マルチエージェントシミュレーションは複雑で緊急な振る舞いを研究するためのアプローチである．マルチエージェントシミュレーションは群集シミュレーションや避難シミュレーションなどに用いられる．マルチエージェントシミュレーションにおいて，エージェントは他のエージェントまたは環境と相互作用を行うためエージェントの個体数が増加するほど計算コストが高い．また，シミュレーション中においてエージェントは，感情や性格などの状態を持っており，相互作用の際に特定の状態を持つエージェントを認識する．近年，マルチエージェントシミュレーションフレームワークが開発されており，いくつかはGPUを用いた高速化により大規模なモデルをシミュレートできる．しかし，SIMD型実行を行うGPUでは，エージェントの状態を認識する機能は分岐処理を含むため性能低下の要因となる．本稿では，上記の問題を解決するために，効率的な分岐処理の軽減を提案する．同時に処理される個体の相互作用計算の範囲を同一化し最適化を行うことで従来手法と比較して約24倍ほどの高速化を達成した．

三重大学大学院　工学研究科

Abstract

Multi-agent simulation is an approach to study complex and urgent behavior. Multi-agent simulation is used for crowd simulation, evacuation simulation, etc. In multi-agent simulation, agents are more computationally expensive as the number of agents increases as they interact with other agents or environments. Also, during simulation, an agent has a state such as emotion or personality, and recognizes an agent having a specific state during interaction. However, on GPUs that perform SIMD-type execution, the ability to recognize the agent's state involves branch processing, which causes performance degradation. In this paper, we propose efficient branch processing mitigation to solve the above problems. By optimizing and optimizing the range of interaction calculation of simultaneously processed solids. We achieved about 24 times faster than the conventional method.

三重大学大学院　工学研究科

目 次

図 目 次

表 目 次

1　はじめに

マルチエージェントシミュレーション [1] は複雑で緊急な振る舞いを研究するためのアプローチである．マルチエージェントシミュレーションは群集シミュレーション [2] や避難シミュレーション [3] などに用いられる．マルチエージェントシミュレーション中では一般に障害物などにより認識可能な近傍は複雑な形状をとるが本稿では一定の距離内を近傍とする簡単なモデルを用いる．マルチエージェントシミュレーションにおいてエージェントは個体ごとに近傍のエージェントまたは近傍の環境との相互作用を計算する．そのため，エージェントの個体数が増加するほどシミュレーションの実行時間が増加する．また，エージェントはそれぞれ個体ごとに感情や性格などの状態を持っており，他個体の状態を認識し振る舞いを変化させることがある．

近年，シミュレーション開発を効率化するために様々なエージェントシミュレーション用フレームワークが開発されてきた．いくつかのフレームワークではGPUを用いた並列演算により個体数の多い大規模なシミュレーションの高速化が可能である．GPUで処理する場合はスレッドごとにエージェントを割り当て，各スレッドはワープという32スレッド単位でSIMD型実行を行う．このとき，同時実行するワープ内のスレッドで

分岐が発生する場合，実行パスが異なることで同時実行されるスレッドのうち一部がアイドルとなる時間が発生し, 実行効率が低下してしまう分岐ダイバージェンスという問題がある. エージェントシミュレーションにおいて，SIMD 型実行を行う GPU ではエージェントの状態を認識する機能は各個体ごとに近傍エージェントが異なるため分岐が生じ，性能低下の要因となる．そこで本稿ではエージェントが状態を認識する際にエージェントの属性データをグループ化し，ワープごとに共通範囲を探索することで分岐処理の軽減を提案する．エージェントの属性データをグループ化することの狙いとしては，特定のエージェントの探索に範囲を絞ることであり，ワープごとに共通範囲を探索することの狙いはワープ内での分岐をなくすことである．エージェントの属性データをグループ化する際は状態ごとに定数を与え，それらをキーに属性データをソートすることによってグループ化を行う．ワープごとに共通範囲を探索する際は，シェアードメモリを用いたメモリアクセスの最適化と，探索範囲の削減を行う.

2　背景

2.1　マルチエージェントシミュレーション

マルチエージェントシミュレーションは複数のエージェントを用いた仮想実験のことである．ここでエージェントとは周囲の状況を認識し，得られた情報から思考し，一定の規則の基で自律的に行動する主体のことを指す．仮想空間内に配置されたエージェントが相互作用することで全体の挙動をシミュレートすることができる．そのため，個々の人間や生物が互いに影響し合う複雑なシミュレーションを得意とする．過去には鳥の群れ (boid[4] model)，避難シミュレーションなどの構築に用いられてきた．

エージェントの行動規則が多数存在する場合は自状態を有限オートマトンで管理し，状態に対応した行動を実行する．近年では複数台のコンピュータを用いたクラスターや GPU などの並列処理のプロセッサを用いてより大規模なシミュレーションを行うことができる．エージェントシミュレーションの環境は連続空間，離散空間，ネットワーク空間の 3 種類に分類できる．連続空間では図 2.3 のようにエージェントは任意の方向と距離を移動できる．また，エージェントはそれぞれ実数値の座標で表されるためエージェントの密度などを正確に観察できる．離散空間では図 2.2

のように空間が一定のサイズのセルに分割され，1 セルに 1 エージェントを割り当て処理する．空間をセルに分割するためエージェントの 1 ステップの移動距離は一定となりシミュレーションの妥当性が低下する．ネットワーク空間では図 2.1 のように空間内に存在するノード間をエージェントが移動する．避難経路が決まっているシミュレーションにおいて用いられる．本稿で扱うエージェントシミュレーションは近傍を一定のユークリッド距離とし，連続空間を対象とする．

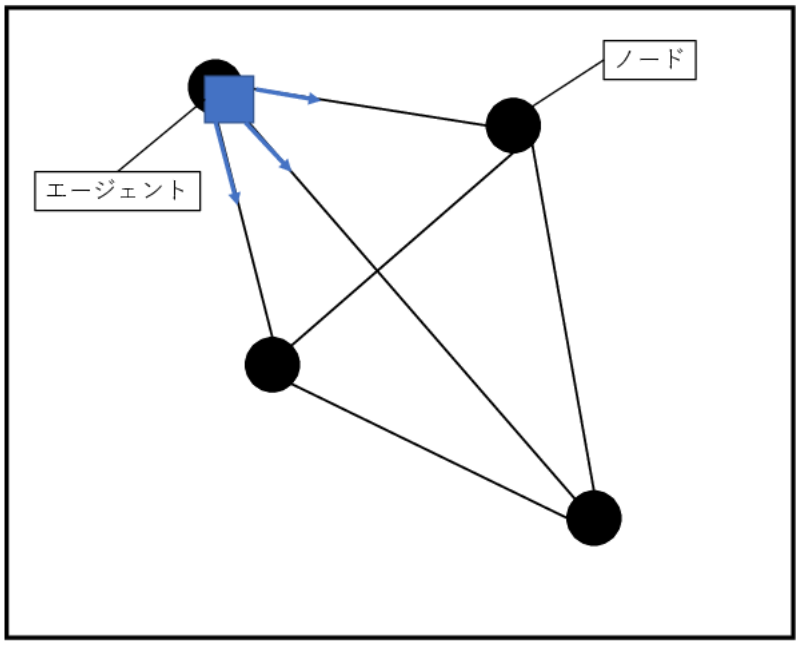

図 2.1: ネットワーク空間

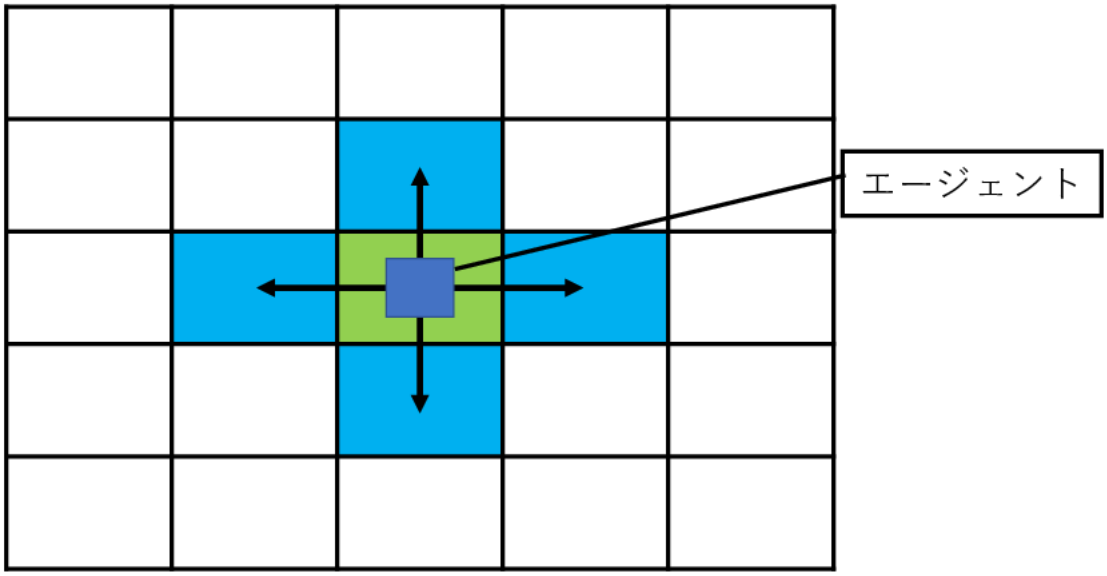

図 2.2: 離散空間

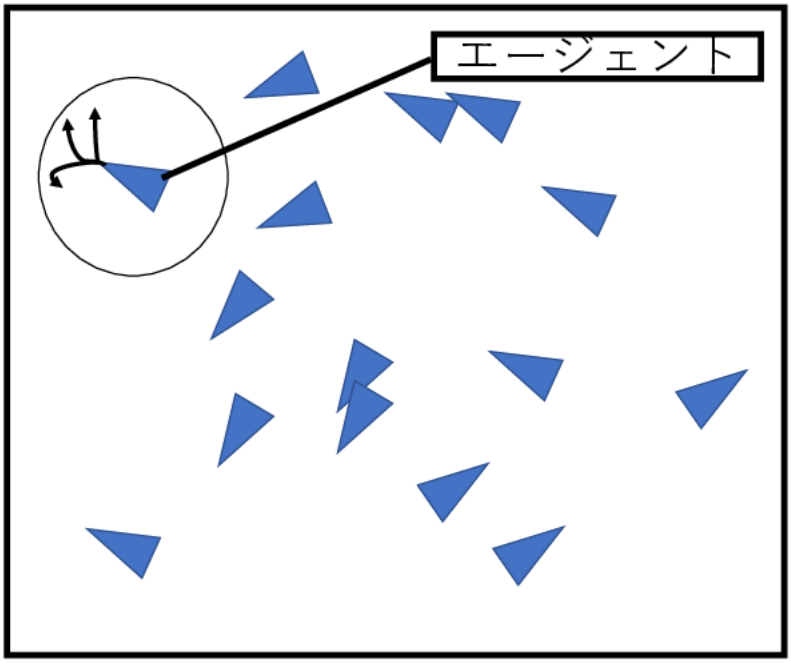

図 2.3: 連続空間

2.2 GPU

GPU[5] は主に画像処理に用いられるプロセッサである．演算を行うコアを大量に搭載し多数のデータを並列に処理できる．GPU では大量にスレッドを生成でき，これらの大量のスレッドは 32 スレッド単位で分割され管理・実行される．この 32 スレッドのグループをワープという．ワープ内の 32 スレッドは同時に同じ命令を実行する SIMD 型の並列処理を行う．

ワープ内の各スレッドがメモリアクセス命令を実行するとき，メモリ上で連続したアドレスへのアクセスは高速になる．GPU はキャッシュを搭載した階層型のメモリアーキテクチャを採用しており，GPU の主記憶であるデバイスメモリへのアクセスは 2 次キャッシュのラインサイズである 128byte 単位で行われる．ワープ内のスレッドが同時に同一キャッシュライン上のデータにアクセスすれば，複数のデータ転送を一度のデバイスメモリへのアクセスで行える．このようなアクセスをコアレッシングアクセスという．また，同一ライン内のデータに対して時間的局所性のあるアクセスを行えば，キャッシュメモリ上にデータが存在するので高速にアクセスできる．GPU にはシェアードメモリと呼ばれるオンチップの高速にアクセスできるメモリを持つ．そこで使用頻度の高いデータをシェ

アードメモリに格納することで高速化が可能である．

2.3 CUDA

CUDA[6] は nVIDIA 社が提供するコンパイラ・ライブラリを含めた GPGPU 統合開発環境であり，ユーザは C/C++ を拡張した文法とライブラリ関数を用いて CUDA プログラムを開発する．CUDA において，CPU 側はホスト，GPU 側はデバイスと呼ぶ．デバイスは PCI-Express を通じてホストにより制御され，ホストから与えられる処理を数千個の CUDA コアで並列実行する．ホスト・デバイスの各 CPU コア・ CUDA コアは図 2.4 に示すように，自身が接続するホストメモリ・デバイスメモリにのみそれぞれアクセスする．ホストメモリ・デバイスメモリ間のデータ転送はユーザ自身が CUDA ライブラリ関数を用いて記述する必要がある．

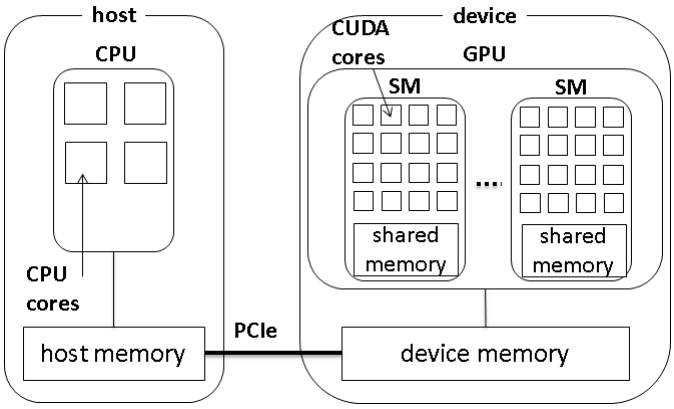

図 2.4: GPU アーキテクチャ

2.4　ブランチダイバージェンス

ワープ内の各スレッドの実行パスが分岐処理により異なる場合，ワープは分岐部分のそれぞれのパスを逐次実行する．例えば，ワープ内のスレッドがif-else文により2通りの実行パスに別れた場合，ワープではif節のパスの各命令を実行した後に，else節のパスの各命令を実行する．分岐内の片側のパスを実行している間，そのパスを実行しないスレッドを担当するコアはアイドル状態となる．この現象をブランチダイバージェンスという．ブランチダイバージェンスによって引き起こされるアイドルコアの増加は性能低下の要因となる．

図2.5にブランチダイバージェンスによる実行効率の低下を示す．同一ワープ内の各スレッドの実行パスが全て同じ場合，実行しないif節またはelse節は処理が省略される．しかしワープ内のスレッドのうち1つでも実行パスが異なる場合，ワープ内ではif-else文全体を実行するため，各スレッドにおいてアイドルとなる時間が発生し，実行効率が低下してしまう．特にネストした分岐やループ文の中に存在する分岐の場合，より実行効率が低下する．

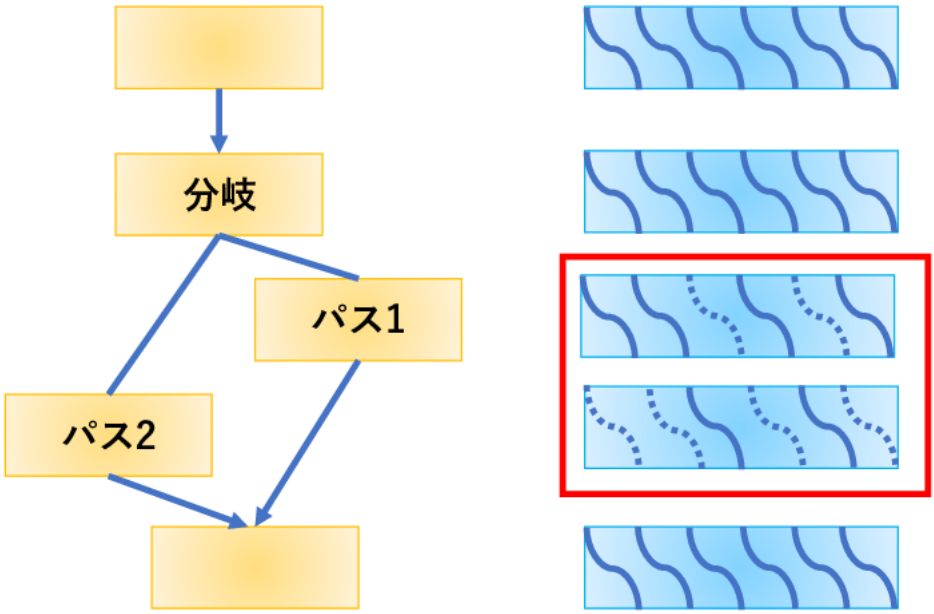

図 2.5: ブランチダイバージェンス

2.5　セルリンクリスト (CL)

セルリンクリスト [7] は，連続体のシミュレーションに使用されるデータ構造である．エージェントシミュレーションにおいては連続空間を使用するモデルで用いられる．空間を同じサイズのセルに分割しておき，各エージェントがどのセル内に存在するかあらかじめ登録する．セルのサイズをエージェントの近傍半径 r にすることで，計算対象のエージェントを自身のセルとその周囲のセルに絞ることができる．図 2.6 にセルリンクリストによる探索範囲の削減の様子を示す．エージェントが所属するセルは，エージェントの x 座標，y 座標をセルの横幅，縦幅で割ることで計算できる．

GPU の実装では，各スレッドの近傍セル内エージェントへのアクセスができるだけ連続となるように Z 階数曲線 [8] を用いる．Z 階数曲線ではデータは図 2.7 のように配置され，近傍セルへのアクセスがラスタ形式のアクセスと比較しコアレッシングアクセスとなるため，メモリアクセスが改善される．

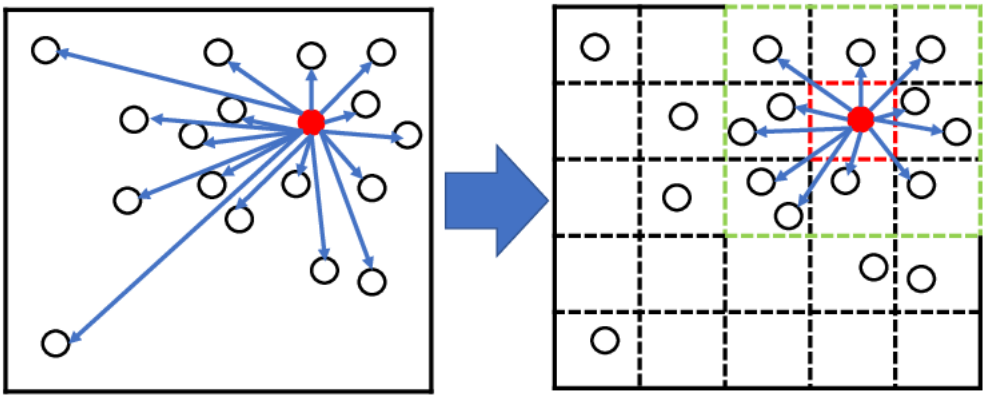

図 2.6: セルリンクリスト

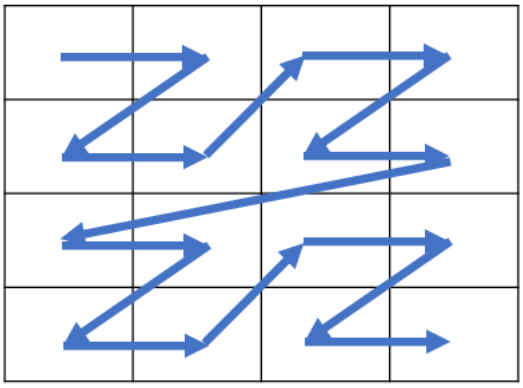

図 2.7: Z 階数曲線

3　提案手法

本稿では，連続体のエージェントシミュレーションにおけるスケジューリングおよび，計算の最適化の実装手法を提案する．しかし，マルチエージェントシミュレーションでは状態によってエージェントは行動を変化させる．例えば，Prey-and-predator モデルでは群れが捕食者を回避しながら餌を探索する．群れを構成するエージェントはそれぞれ近傍の捕食者と仲間を認識して行動する．その際に近傍のエージェントが捕食者かどうかで分岐が発生し，捕食者であった場合は距離をおく，捕食者でない場合は群れを維持するように相互作用計算する．各ワープ内スレッドで担当するエージェントが異なるためブランチダイバージェンスによるアイドルコアが発生する．そのため，ワープごとに探索範囲を共通化することでブランチダイバージェンスの発生を抑える．また，増加した探索範囲を削減することによって計算量の最適化を行う．

3.1　近傍探索領域の共通化

近傍領域の探索では，セルによって分割された各スレッドが担当するエージェントの近傍内の他エージェントとの距離計算を行い，相互作用計算を行う．マルチエージェントシミュレーションのモデルによっては，近傍内のエージェントの状態によって行動が異なる．そのためワープ内で探索範囲が異なると，各スレッドがエージェントを探索する度にブランチダイバージェンスが発生する．そこで，ワープ内スレッドの探索範囲を共通化する．探索範囲の共通化を図 3.8 に示す．

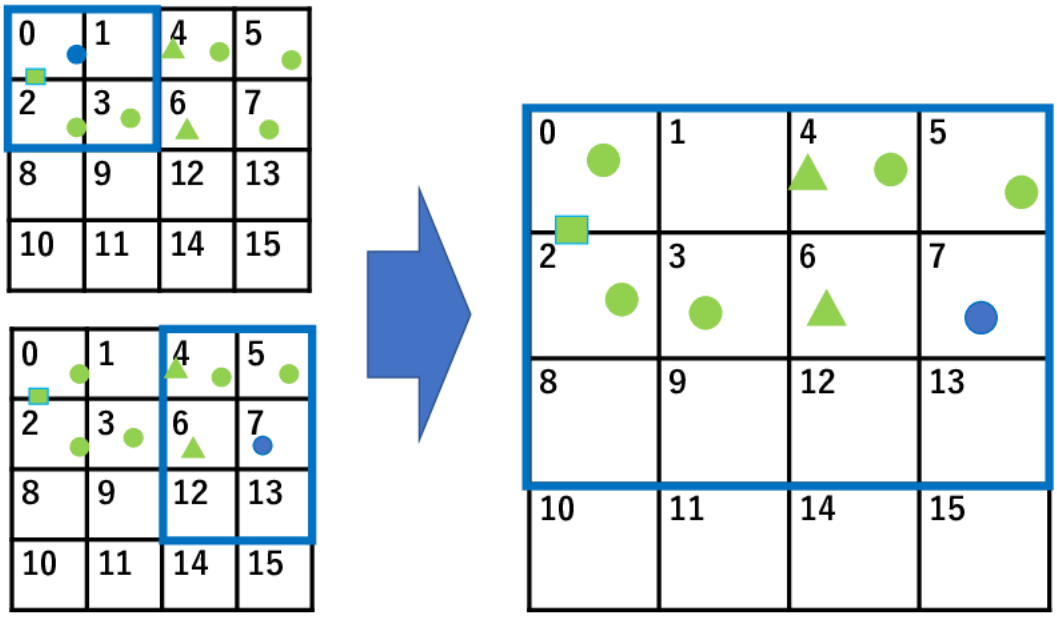

図 3.8: 探索範囲の共通化

ここでエージェントを表す図形は状態の違いを表している．探索範囲を共通化することで，通常よりも探索範囲が多くなるが，ワープ内エージェントが同じエージェントを順に計算していくので状態による分岐が生じない．その様子を図 3.9 に示す．

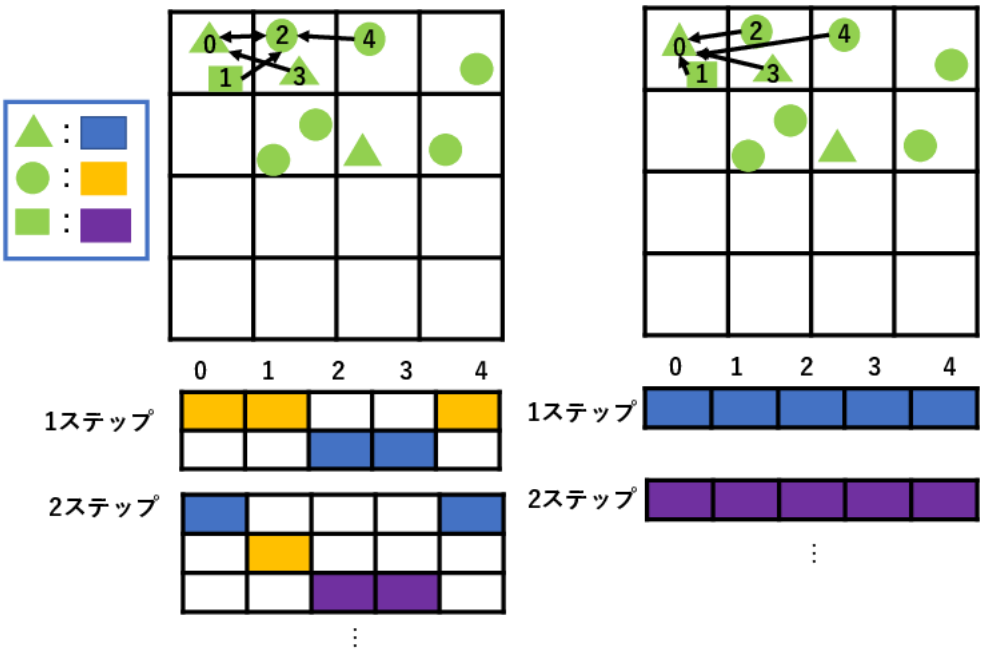

図 3.9: 共通化による分岐の減少

図3.9の左ずより０４のエージェントが１ステップ目で探索するエージェントが異なることでダイバージェンスが発生することがわかるが，探索範囲の共通化を行うことで図3.9右図のように０４のエージェントが毎ステップ探索するエージェントが同一となるためダイバージェンスを回避できる．共通化手順としては，ワープ内の各スレッドのエージェントが所属するセルのｘ座標，ｙ座標を計算する．次に，ワープ内でｘ座標，ｙ座標の最大値を求める．求めたｘ座標，ｙ座標をそのワープ内スレッドの探索範囲とすることで探索範囲を共通化する．

3.2　近傍探索領域の共通化における探索範囲の削減

共通化した探索範囲はワープ内の各スレッドが担当するエージェントの所属しているセルによっては無駄な探索セルが生じる．図3.10に示すように，ワープ内エージェントの共通探索範囲が図3.10の右側の場合，一番右下のセルはどのエージェントの探索範囲でもないので探索が必要のないセルである．そこで，探索が必要のないセルを削減する．探索セルを順に変えながら処理を行っていくが，その際にワープ内の各エージェントが所属するセルの前後左右に次に処理するセルが含まれていないかどうかを判定する．探索範囲内に次に処理するセルが含まれているかの

判定に warp-vote 命令の一つである__any 命令を使用する．__any 命令は引数として bool 値を受け取り，ワープ内の全スレッドで一つでも true となるスレッドが存在すれば true，全スレッドが false ならば false を返す関数である．

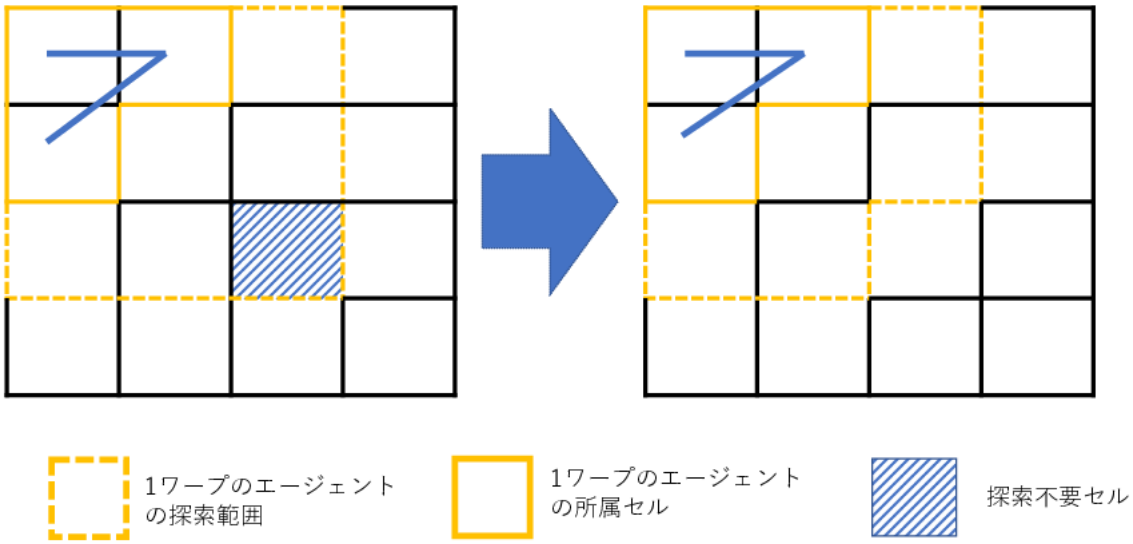

図 3.10: 探索範囲の削減

3.3　状態数とセルでのグルーピング

近傍探索の際にセル内のある特定の状態のエージェントを計算対象とする場合，その状態以外のエージェントは計算対象外となる．そこで，セル内を状態数でさらに分割することで同一セル内のエージェントを同一の状態でグルーピングすることができ，近傍かつ特定の状態のエージェントのみを計算対象とすることができる．グルーピングの結果，近傍探索の際に特定のエージェント以外のエージェントを計算せずに処理できるので高速化に繋がる．図 3.11 に状態とセルによる分割を示す．図 3.11

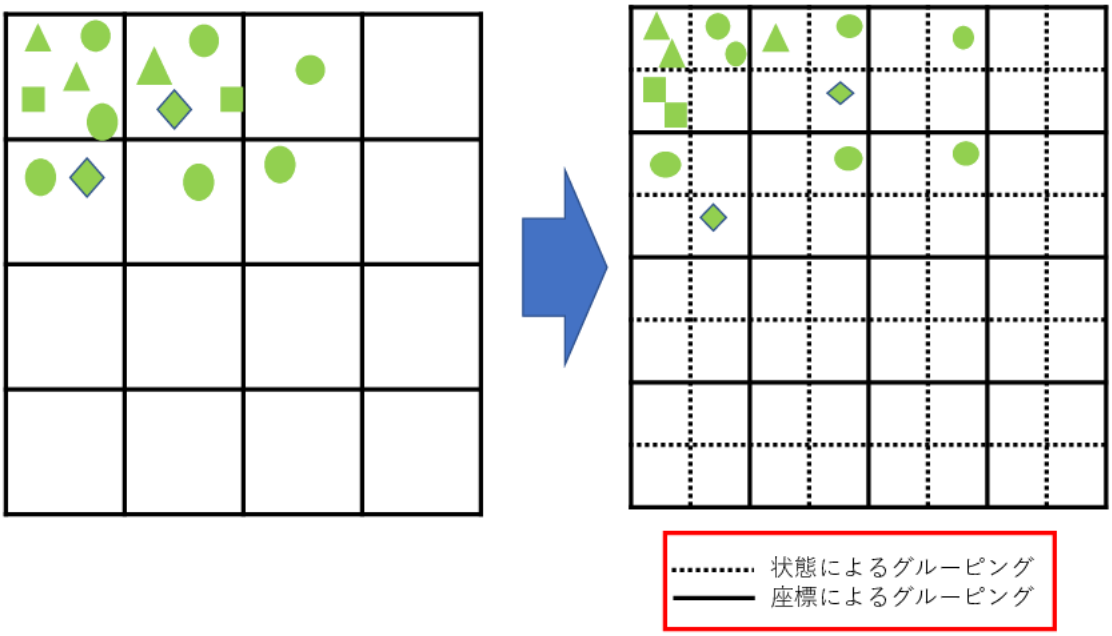

図 3.11: 状態によるグルーピング

において緑のエージェントの図形の違いはエージェントの状態の違いを示す．ここで同じセル内でさらに同一の状態のエージェントをグループ化すると図 3.11 の右図のようになり通常の座標だけでのグルーピングよりも探索対象となるエージェントを削減できる．

4　評価

4.1　評価方法

提案した手法を衝突を回避しながらランダムウォークを行うモデルに状態を与え，特定の状態をカウントする擬似モデルを用いて評価を行った．

特定の状態のエージェントをカウントする機能は ABMS のフレームワークである Artisoc[10] のカウント機能の実装を参考にした．また評価に用いた環境を表 4.1 に示す．評価用プログラムでは連続空間をエージェント

	環境
CPU	Intel(R) Core(TM) i7 930
Memory	6GB
GPU	GForce GTX 1070
Memory	8GB

表 4.1: 評価環境

がランダムに動き，各ステップで一定距離内の他のエージェントとの相互作用を計算し，座標を更新する．その際にエージェントは状態をもち，一定の距離にいるエージェントで特定の状態のエージェントを対象とする近傍探索を実装し，評価をとった．提案手法を未実装のプログラムと探索範囲の共通化を実装したプログラム，共通化した探索範囲の削減を

実装したプログラム，また探索範囲の共通化と状態によるグルーピング
を実装したプログラムの実行時間の比較を行った．

4.2 性能評価

図 4.12 に探索範囲の共通化による速度向上率のグラフを示す．図 4.12 からエージェントの個体数が増加するほど速度が向上していることがわかる．これはエージェントが増加するほどワープ内でのダイバージェンスが起こり，速度が低下していると考えられる．

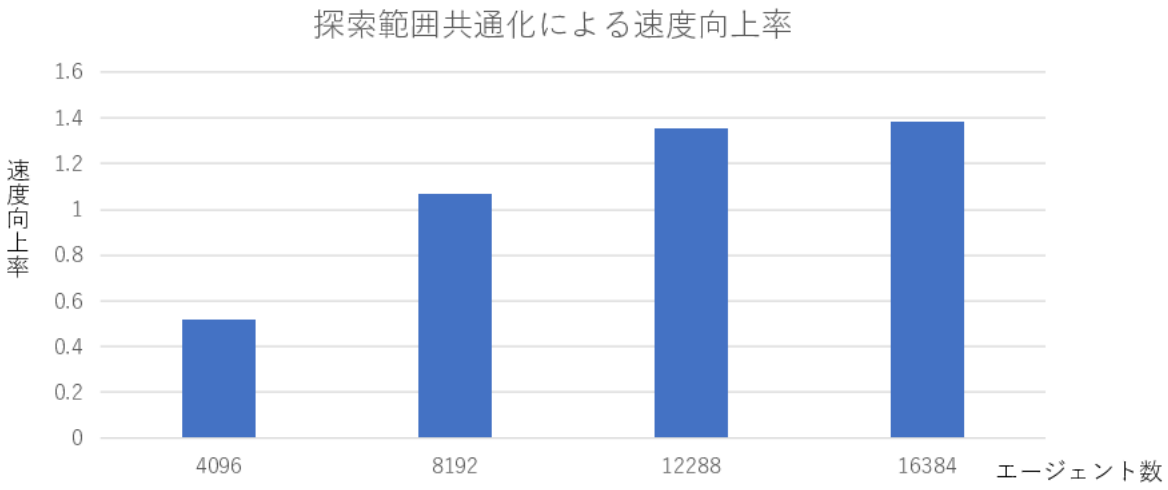

図 4.12: 探索範囲の共通化の速度向上率

続いて，図 4.13 に探索範囲削減による速度向上率を示す．

図 4.13 から全ての個体数で速度向上が見られた．特に個体数が 16384 の時最大 1.5 倍ほど速度向上が見られた．これは，個体数が上がるほど 1 ワープ内で削減できる探索範囲が増加するためだと考えられる．

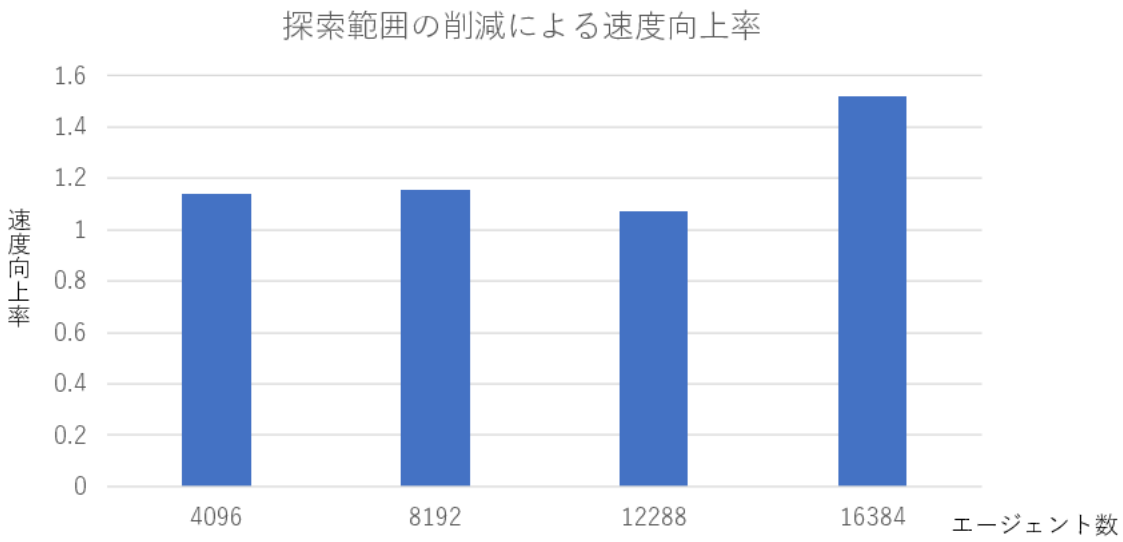

図 4.13: 探索範囲の削減による速度向上率

図 4.14 に状態によるグルーピングによる速度向上率を示す．図 4.14 から全個体数で速度向上が見られた．特に個体数が 8192 の時に最大でおおよそ 1.6 倍ほど速度向上が見られた．これは，個体数が 8192 の時に各ステップにおいて探索対象とする状態の個体をより絞ることができたと考えられる．

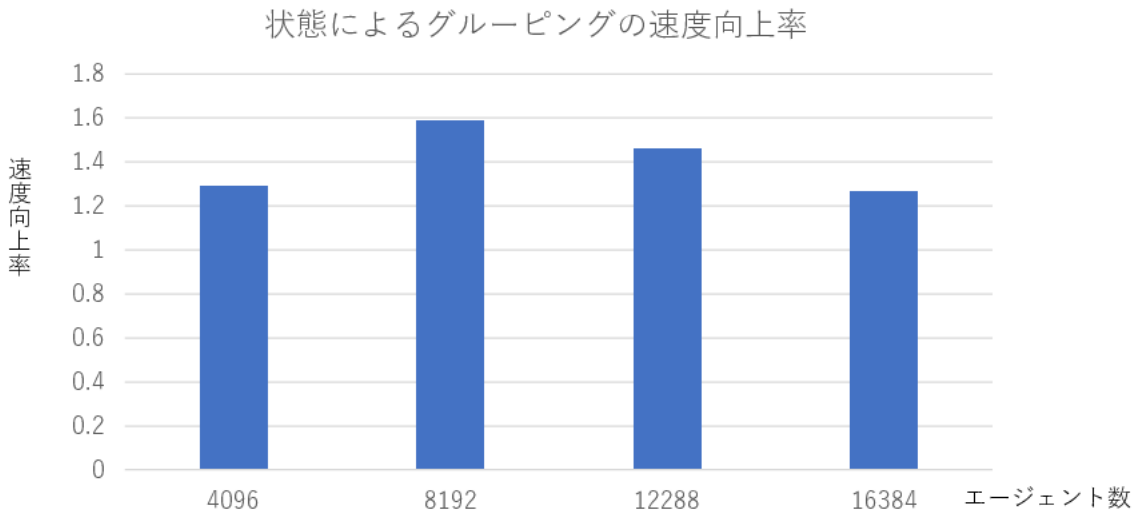

図 4.14: 状態のグルーピングによる速度向上率

最後に図 4.15 に全ての手法を実装したプログラムの速度向上率を示す．図 4.15 から最大で約 24 倍の速度向上が見られた．これは，全ての手法を適用した結果としてそれぞれの実装と比較して探索範囲をより絞ることができるため高い速度向上が得られたと考察できる．個々の手法による速度向上率と比較して高い速度向上が得られたことについては今後の課題とする．

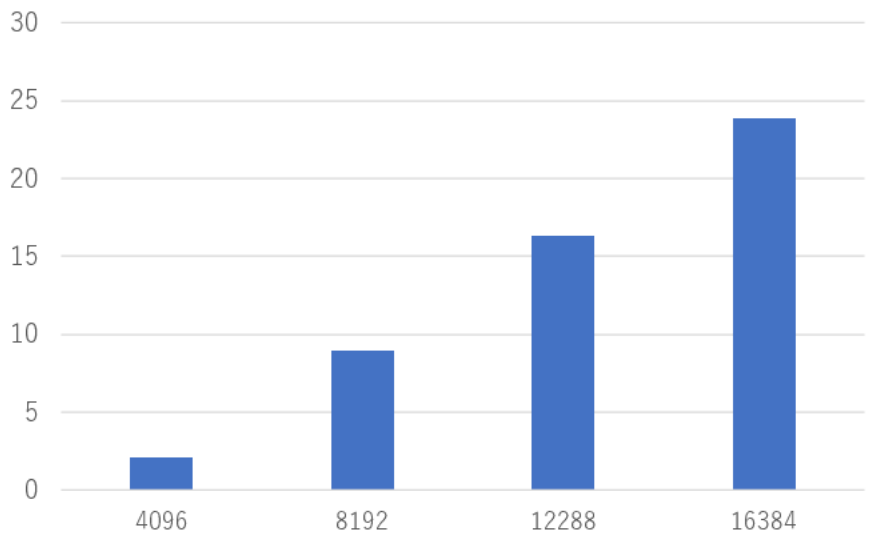

図 4.15: 全手法適用による速度向上率

5　関連研究

エージェントシミュレーションにおけるフレームワークは多数存在する．中でもFLAMEはGPUに対応したエージェントシミュレーションフレームワークとして大規模なシミュレーションを実行可能である．FLAMEGPUでは，状態によるダイバージェンスの発生が考慮されておらず，状態が多数存在するモデルでは性能が低下するという問題があった．Mozhgan[11]らは，FLAMEGPU用のコードを静的解析し，if分の解析を行って分岐ごとに状態を割り当てた．変化する状態を動的にグループ化することでワープ内スレッドの分岐を減らすことができ，ナイーブな実装と比較して最大4倍の速度向上を達成している．この研究では書き込み側のエージェントのみのダイバージェンスを対象としており，近傍探索の際に読み込むエージェントのダイバージェンスは対象とされていない．Xiasong Li[12]らは連続空間のエージェントシミュレーションをシェアードメモリに格納することで高速化を達成している．

6　おわりに

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近傍探索処理の際の状態によるブランチダイバージェンスを回避するためのスケジューリングを提案した．評価の結果，ある状態のエージェントを回避するという単純な擬似モデルにおいて速度向上が見られた．今後の課題としては状態を持つ実問題を扱うシミュレーションでの評価があげられる．

謝辞

本研究を行うにあたり, 御指導, 御助言頂きました大野和彦講師, 並びに多くの助言を頂きました山田俊行講師に深く感謝致します . また, 様々な局面にてお世話になりました研究室の皆様にも心より感謝いたします.

参考文献

[1] M Niazi,A Hussain.Agent-based computing from multi-agent systems to agent-based models: a visual survey.2011

[2] Daniel Thalmann.Crowd Simulation.14 December 2007.

[3] Santos, Gabriel; Aguirre, Benigno E.A critical review of emergency evacuation simulation models.Disaster Research Center

[4] Craig W. Reynolds. Flocks,Herds,and Schools:A Distributed Behavioral Model. Computer Graphics.Volume 21.1987

[5] NVIDIA Corporation, http://www.nvidia.co.jp(参照 2018-02- 05).

[6] NVIDIA Corporation, http://www.nvidia.co.jp(参照 2018-02- 05)

[7] Mattson, W.; B. M. Rice (1999). "Near-neighbor calculations using a modified cell-linked list method". Computer Physics Communications. 119 (23): 135.

[8] Morton, G. M. (1966), A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing, Technical Report, Ottawa, Canada: IBM Ltd.

[9] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[10] http://mas.kke.co.jp

[11] Mozhgan K.Chimeh,Paul Richmond. Simulating heterogeneous bahaviours in complex systems on GPUs.Simulation Modeling Practice and Theory.Volume 83,April 2018,Pages 3-17

[12] Xiaosong Li, Wentong Cai and Stephen John Turner.Supporting efficient execution of continuous space agent-based simulation on GPU.CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE.2016

[13] A high performance agent based modelling framework on graphics card hardware with CUDA Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009), pp. 1125-1126