卒業論文

題目

GPU 上でのセルリンクリスト法の改良によるアイドル時間の削減手法

指導教員

大野　和彦

2019 年

三重大学 工学部 情報工学科
コンピュータソフトウェア研究室

佐々木　良輔 (415826)

内容梗概

近年，避難シミュレーションなどの大規模なマルチエージェントシミュレーション (MAS) の需要が高まっている．しかし，エージェント数の増加などに伴い計算コストが膨大となり，実行時間が大幅に増加する問題がある．そのため，近年性能向上が目覚ましい GPU を用いた並列計算による高速化が行われている．マルチエージェントシミュレーションでは，エージェントは周囲のエージェントや環境から影響を受ける．エージェント数の増加に伴い近傍エージェントの探索コストも増加するため，効率的な探索手法が提案されている．その一つにセルリンクリスト法がある．セルリンクリスト法では，シミュレーション空間を同じサイズのセルに分割し，エージェントとセルの対応付けを行う．これにより，探索時に更新するエージェントが所属しているセルとその周囲のセルに所属するエージェントを近傍候補のエージェントとすることができ, 探索コストを削減できる．一方，GPU は複数のコアを搭載しており，複数のデータに対して同じ命令を実行する SIMD 型実行を行っている．GPU 上で実行するプログラムに条件分岐が含まれている場合，同時実行するスレッドの実行パスが異なる場合がある．実行パスがスレッドごとに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状態となり，実行効率が低下してしまうブランチダイバージェンスという問題がある．GPU 上でセルリンクリスト法を適用したマルチエージェントシミュレーションのプログラムを実行する場合，異なるセルに所属するエージェントを同時実行するスレッドで更新する場合がある．この時，同時実行するスレッドの注目するセルが異なり，注目するセルに所属するエージェント数の差異により，ブランチダイバージェンスが生じてしまう問題がある．

そこで本研究では，各エージェントの重複する探索範囲内のセルを同時に参照することにより，ブランチダイバージェンスの削減を行った．距離計算などの各エージェントの処理を行う時，従来の手法では周囲のセルへの参照はラスタ順に行われるが，本手法では更新するエージェントの所属するセルによって周囲のセルを参照する順序の変更を行った．その結果，シミュレーション空間全体に対してエージェント数が多い場合に実行時間を最大 8%削減できた．

Abstract

Multi-agent simulation(MAS) is used for evacuation simulation, simulation of road congestion and so on. Thus, demand for MAS has been increasing. On the other hand, the calculation cost becomes enormous as the number of agents increases, the execution time greatly increases. Therefore, it exposes a high degree of parallelism and has already been successfully ported to GPU. In MAS, the agents are affected by surrounding agents. The neighboring agents have to be searched every time step. As the number of agents increases, the neighboring agents search cost is increased. Cell-Lined List method is an efficient search method. In this method, the simulation space is divided by cells of the same size, and agents are associated with cell. Neighboring agents search is made efficient by referring only to the agents belonging to the cells surrounding the agent to be updated. The GPU performs SIMD type execution. However, if a condition branch is included in the program which is executed on the GPU, the execution path of the concurrently executed thread may be different. A part of the concurrently executed threads becomes idle due to the execution oath being different. This problem is called branching divergence. When a program to which the cell-lined list method is applied is executed on the GPU, agents belonging to different cells may be updated by a thread which is executed at the same time. Thus, branching divergence occurs due to the difference in the number of agents belonging to the cell referenced by the concurrently executing thread.

In this paper, we improve to refer simultaneously to the cells within the overlap range within the duplicate search range of each agent. As a result, branching divergence are reduced, and the execution time is reduced when the number of agents is large for the entire simulation space.

目 次

図 目 次

表 目 次

1　はじめに

近年，避難シミュレーションや交通シミュレーションなどの大規模なマルチエージェントシミュレーション (MAS)[1] の需要が高まっている．しかし，エージェント数の増加に伴い近傍エージェントの探索コストが膨大になり，実行時間が大幅に増加するという問題がある．一方，グラフィック処理用プロセッサである GPU 上で汎用的な計算を行う GPGPU(General Purpose computing on Graphics Processing Units)[2] は，CPU 以上の計算性能を発揮することから関心が高まっている．これらのことから，大規模な MAS に GPGPU を利用することで高速化が行われている．

GPU は多数のプロセッサを搭載し，SIMD 型実行を行うため，各スレッドは複数のデータに対して同じ命令を実行しながら並列処理を行う．GPU 上で実行するプログラムに条件分岐が含まれている場合，同時実行するスレッドの実行パスが異なる場合がある．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のスレッドがアイドル状態となり，実行効率が低下してしまう問題がある．この問題をブランチダイバージェンスと呼ぶ.

MAS では，各エージェントは周囲のエージェントや環境から影響を受ける．各エージェントの近傍エージェントの探索を毎ステップ行う必要

があるため，エージェント数の増加に伴い探索コストが増加し，実行時間が大幅に増加する．近傍エージェントの探索コストを削減し，高速化する手法にセルリンクリスト法 [3] がある．セルリンクリスト法では，シミュレーション空間を同じサイズのセルで分割し，エージェントとセルの対応付けを行う．これにより，更新するエージェントの周囲のセルに所属するエージェントを近傍候補とすることができ，探索コストを削減できる．従来のセルリンクリスト法では更新するエージェントの探索範囲に対してラスタ順に走査する．同時実行するスレッドが異なるセルに所属するエージェントを同時に更新する場合があり，スレッドによって参照するセルが異なる．各セルに所属するエージェント数は一般的に異なるため、各スレッドのループ回数に差異が生じブランチダイバージェンスを引き起こす．

そこで本研究では，更新するエージェントの探索範囲で重複している範囲に存在するセルを同時に参照することにより，ブランチダイバージェンスの削減を行った．

以下，2 章では背景として GPU，MAS，セルリンクリスト法について説明する．3 章では，本研究で提案するブランチダイバージェンスの削減手法について説明する．4 章では従来のセルリンクリスト法と提案手法を

適用したプログラムの実行時間を比較した結果を示す．最後に 5 章でまとめを行う．

2　背景

2.1　GPU

GPU は演算を行うコアを大量に搭載し多数の処理を並列に実行できる．GPU ではコア数を越える大量のスレッドが生成でき，これらの大量のスレッドは 32 スレッドごとにまとめて管理される．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．

GPU では，主記憶であるデバイスメモリからのデータ転送は L2 キャッシュのラインサイズである 128byte 単位で行われる．そのため，同ワープ内のスレッドのアクセスが同一キャッシュライン上のアドレスであれば，複数のデータ転送を一度のデータ転送で行える．このようなメモリアクセスをコアレスアクセスと呼ぶ．

2.1.1　CUDA

CUDA[4] は NVIDIA 社より提供されている GPGPU 用の SDK であり，C/C++ または Fortran を拡張した文法とライブラリ関数を用いて GPGPU プログラムを開発することができる．CUDA では，CPU 側をホスト，GPU 側をデバイスと呼ぶ．

ホストとデバイスはそれぞれ異なるメモリを持ち，ホストメモリとデバイスメモリの間で通信を行う．GPGPU プログラムでは，デバイスで処理を行う前にホスト側からデバイス側のメモリ確保を行う．そして，ホスト側から処理に必要なデータをデバイス側に転送する．次に，GPU 上で実行する関数であるカーネル関数をホスト側から呼び出すことでデバイス側の処理が開始される．カーネル関数の実行後，結果をホスト側に転送することで処理が終了する．

2.1.2 ブランチダイバージェンス

GPU 上で実行するプログラムに if-else 文のような条件分岐が含まれ，異なる実行パスをもつスレッドが同ワープ内に存在する場合，同ワープ内のスレッドのうち一部のスレッドがアイドル状態となり実行性能が低下してしまう問題がある．この問題をブランチダイバージェンスと呼ぶ．

図 2.1 にブランチダイバージェンスの生じる例を示している．この図は，GPU 上で実行するプログラムに if-else 文が含まれている場合の実行の様子を示している．ワープ内のスレッドがすべて同じ実行パスの場合，条件式が真の時の処理，または偽の時の処理のみ実行する．しかし，同ワープ内に条件式が真となるスレッドと偽となるスレッドが存在する場合，条件式が真の時の処理を行った後，偽の時の処理を行う．この時，条

件式が真の時の処理を行っている間は条件式が偽となるスレッドがアイドル状態となってしまい，実行効率が低下してしまう．

また，ブランチダイバージェンスは if-else 文のような条件分岐だけでなく，while 文のようなループ文でも発生する．図 2.2 にループ文によりブランチダイバージェンスが生じる例を示している．ループ文では，各スレッドの処理量の差異，つまりループ回数の差異によってブランチダイバージェンスが生じる．SIMD 型実行のため，ループ回数の少ないスレッドはループ処理が終わった後，ループ回数の多いスレッドがループ処理を終えるまで次の命令を実行することが出来ず，その間アイドル状態となってしまう．

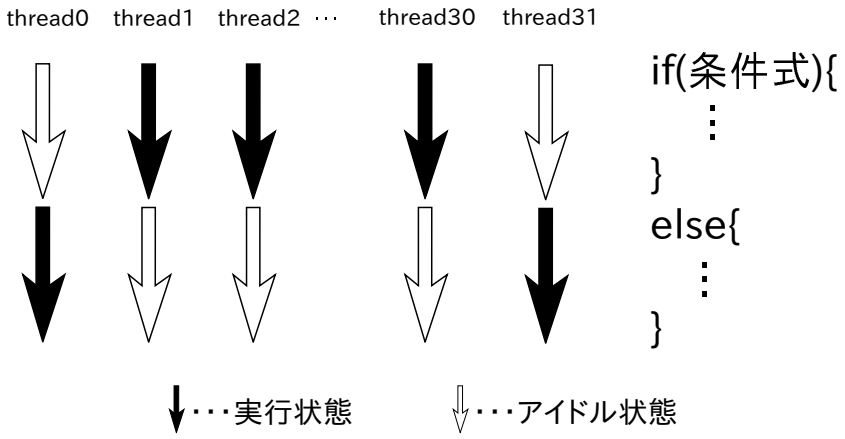

図 2.1: ブランチダイバージェンスの発生例

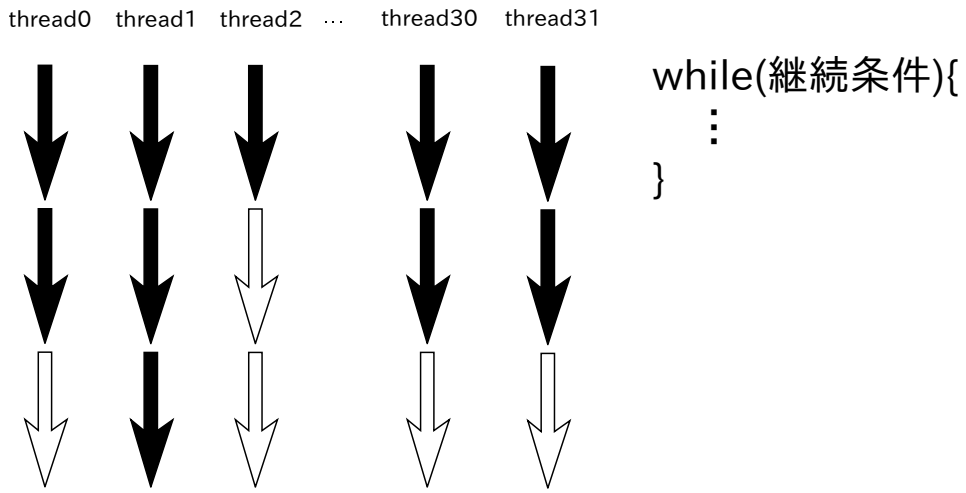

図 2.2: ループ回数によるブランチダイバージェンス

2.2 マルチエージェントシミュレーション

一定のルールのもとで周囲の状況に基づき，自律的に行動するものをエージェントと呼ぶ．複数のエージェントが周囲のエージェントと相互作用を繰り返し，複雑系の現象を再現するシミュレーションをマルチエージェントシミュレーション [1] と呼ぶ．シミュレーションの例として,「高速道路の自然渋滞」や「災害時の避難シミュレーション」などが挙げられる．

マルチエージェントシミュレーションでは，エージェントは周囲のエージェントや環境から影響を受ける．相互作用計算を行うために，周囲のエージェントが相互作用を及ぼす範囲内に存在するかどうかを判定する

必要がある．しかし，ナイーブな実装では自身以外のエージェント全てに対して相互作用範囲内かどうかを判定する必要がある．そのため，エージェント数の増加に伴い，探索コストが膨大となり実行時間が大幅に増加するという問題がある．

そこで，探索コストを削減するため，空間分割法の一種であるセルリンクリスト法 [3] が用いられている．

2.3 セルリンクリスト法

セルリンクリスト法では，図 2.3 に示すようにシミュレーション空間を同じサイズのセルに分割し，各セルに id を割り当てる．そして，エージェントとセルの対応付けを行う．セルサイズを相互作用範囲の半径 r とすることで，更新するエージェントが所属するセルとその周囲のセルに所属するエージェントを近傍エージェントの候補として，距離計算を行う．近傍エージェントの候補との距離が r 以下であれば相互作用計算を行う．このように探索範囲を限定することで探索コストを削減する．また，セルサイズを相互作用範囲の半径 r の半分にすることで，図 2.4 に示すように探索範囲を更に限定でき，近傍候補のエージェント数を更に削減できる．

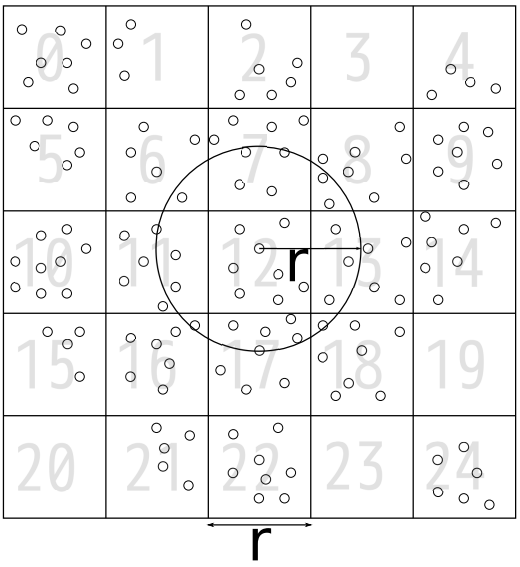

図 2.3: シミュレーション空間の分割と id の割り当て

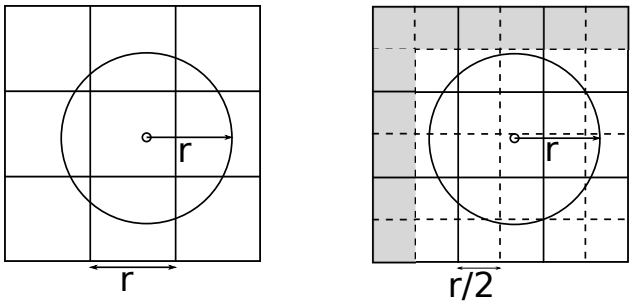

図 2.4: セルサイズの変更による探索範囲の削減

エージェントとセルを対応付ける手法の一種に slide vector 法 [5] がある．slide vector 法では図 2.5 に示すように，エージェント配列をエージェントが所属するセルの id を用いてソートし，エージェント配列上で所属セルの境界となるインデックス値を登録する．境界インデックスを登録

する配列のインデックス値はセル id と対応しているため，セル id からセル内のエージェントを参照できる．

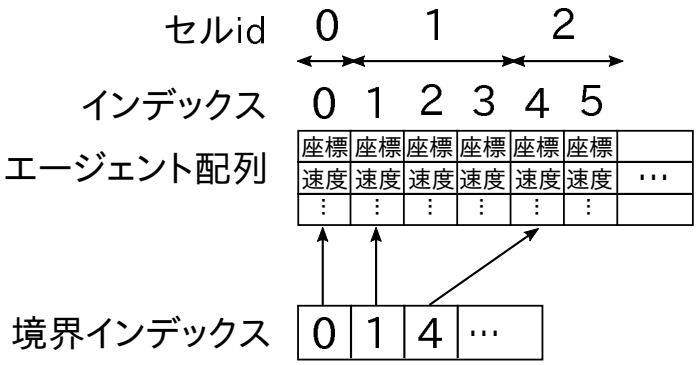

図 2.5: slide vector 法

2.3.1 セルリンクリスト法の問題点

GPU 上で実行するプログラムにセルリンクリスト法を適用すると，ブランチダイバージェンスが生じ，実行性能が低下する場合がある．図 2.6 を用いてブランチダイバージェンスが生じる場合について説明する．図 2.6 の上の図は図 2.3 の一部分を抜き出したものである．下の図は id が 6 のセルと 7 のセルに所属するエージェントがそれぞれ探索する範囲のセルを示し，グレーに色付けされているセルは id が 6 と 7 のセルに所属するエージェントの重複する探索範囲のセルを示している．id が 6 のセルにエージェント 1，id が 7 のセルにエージェント 2 が所属しているとする．この場合，単純な実装のセルリンクリスト法では近傍エージェントの探

索時に探索範囲のセルをラスタ順に走査する．エージェント 1 は最初に id が 0 のセルを参照するが，エージェント 2 は id が 1 のセルを参照するためエージェントごとに参照するセルが異なる．この時，参照しているセルに所属するエージェントの数だけ処理を行うためループ処理を行う．各セルに所属するエージェント数は異なる場合があるため，各セルに所属するエージェント数の差異により，スレッドごとのループ回数が異なりブランチダイバージェンスが生じる．

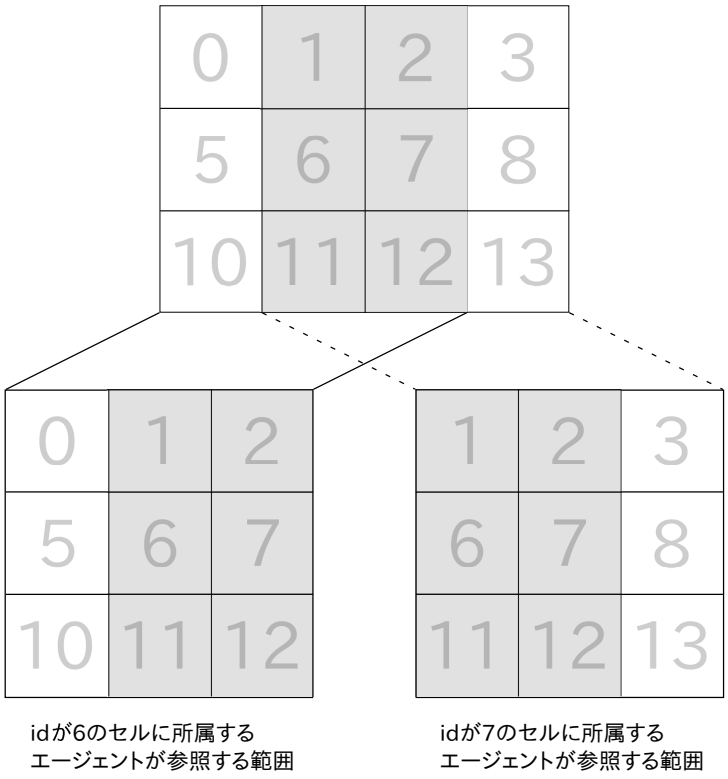

図 2.6: 近傍探索時の探索範囲

本研究では，GPU 上で実行するプログラムにセルリンクリスト法を適用した場合の，各セルに所属するエージェント数の差異によって生じるブランチダイバージェンスの削減を行う．

3　提案手法

前述した問題を解決するため，各セルに所属するエージェント数の差異により生じるブランチダイバージェンスの削減手法を提案する．

3.1　概要

近傍エージェントの探索時に周囲のセルはラスタ順に走査される．これにより，異なるセルに所属するエージェントを同ワープ内で更新する場合に，スレッドごとにループ回数が異なるためブランチダイバージェンスが生じる．この時，近傍エージェントの探索時に探索範囲が重複する場合がある．そこで本研究では，重複している探索範囲のセルを同時に参照することにより，各セルに所属するエージェント数の差異により生じるブランチダイバージェンスの削減を行う．そのため，図 3.7 に示すように更新するエージェントが所属するセルに応じて，探索時に周囲のセルを参照する順序の変更を行う．

本手法を 2.3.1 で挙げた例に適用したものを図 3.8 に示す．図 3.8 の上の図のセルの中の数字は各セルの id を示しているが，下の図の丸で囲われている数字はエージェント 1 とエージェント 2 が近傍探索時に参照するセルの順序を示している．本手法を適用した場合，エージェント 1 は id が

2,0,1,5... の順にセルを参照し，エージェント 2 は id が 2,3,1,5... の順にセルを参照する．このように，重複する探索範囲のセルを同時に参照する．

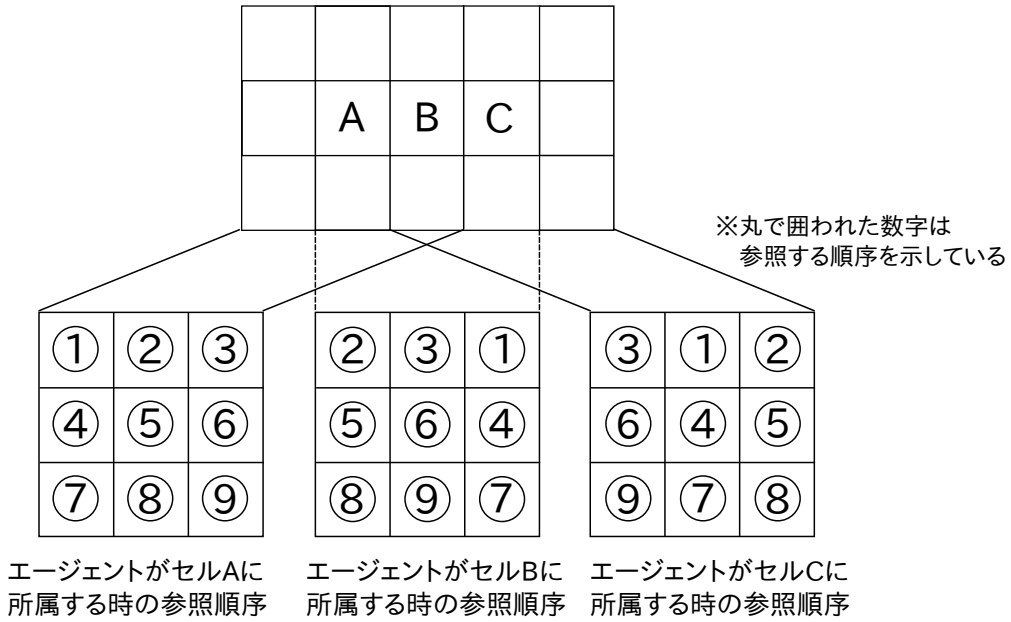

図 3.7: 所属するセルによる参照順序の変更

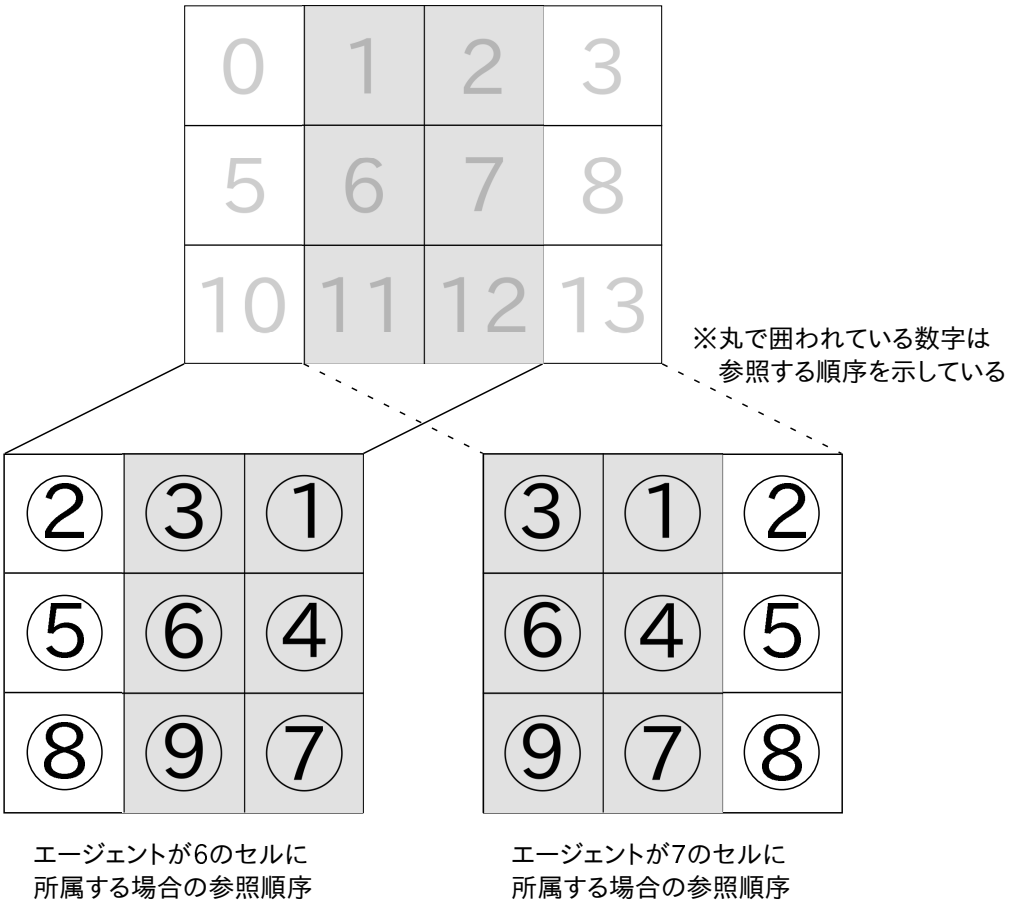

図 3.8: 提案手法を適用した例

3.2 実装

更新するエージェントが所属するセルに応じて探索時に参照するセルの順序を変更するため，図3.9に示しているアクセステーブルを実装する．

```
int table[ ] = {     -1, 0, 1,     // セル A に所属
                      1, -1, 0,    // セル B に所属
                      0, 1, -1};   // セル C に所属
```

図 3.9: 実装したアクセステーブル

このアクセステーブルは，更新するエージェントが所属するセルによって参照する範囲が異なる．例えば，セル B に所属するエージェントであれば 2 行目の 3 つの要素を左から順に参照する．各行で i 番目に参照するセルの列数 C_i は，更新するエージェントが所属するセルの列数を $cellx$ とした時次の式で求められる．

$$C_i = cellx + table[i + 3 * (cellx\%3)]$$

セル B に所属しているエージェントは，初めに $cellx$ とアクセステーブルの参照する範囲の 1 つ目の要素を計算することでセル B の右側の列のセルに参照する．同様に，$cellx$ とアクセステーブルの 2 つ目の値と 3 つ目の値を計算することで，セル B の左の列のセルとセル B と同じ列のセルを参照することができる．これにより，図 3.7 に示すような，更新するエージェントが所属するセルに応じて，探索時に参照するセルの順序を変更することができる．

4　評価

4.1　評価プログラムと評価環境

評価プログラムとしてマルチエージェントシミュレーションの一種である，鳥の群れを再現する Boids モデル [6] を用いる．そして，探索時にラスタ順に走査する従来のセルリンクリスト法と提案手法を適用したセルリンクリスト法の実行時間の比較を行った．表 4.1 に評価環境を示す．

表 4.1: 評価環境

	環境
CPU	Intel(R) Xeon E5-2620
Memory	16GB
GPU	GeForce GTX 950
GPU Memory	2GB

4.2　評価結果

評価プログラムは 5000 ステップ計算し，エージェント数を 65536，セルサイズを固定とし，シミュレーション空間の大きさが 1 辺が 100 から 3200 まで 6 通りに変化させて実行時間を計測した．この結果を表 4.2 に示す．削減率は以下の計算式で求められる．

$$\text{削減率} = \frac{\text{従来手法の実行時間} - \text{提案手法の実行時間}}{\text{従来手法の実行時間}} \times 100$$

表 4.2: 実行結果

空間サイズ	100	200	400	800	1600	3200
従来のセルリンクリスト法 (s)	1122.45	99.91	27.29	15.41	11.57	10.88
提案手法 (s)	1066.08	92.96	25.69	14.16	11.44	11.00
削減率 (%)	5.02	6.95	5.86	8.11	1.12	-1.10

4.3 考察

表 4.2 の結果から分かるように，空間が小さい場合には実行時間の削減ができたが，空間が大きい場合は実行時間の削減ができなかった．このことから，空間が小さい場合に提案手法によりブランチダイバージェンスの削減ができ，実行時間の削減ができたと考えられる．

実行時間の削減ができなかったのは，空間の拡大に伴い各セルに所属するエージェント数が減少したことや，提案手法により参照するセルの順序を変更したことが原因であると考えられる．これにより，ワープ内のスレッドで更新するエージェントの所属するセルの数が増加したことで，同時に参照するセルが増加しブランチダイバージェンスの削減量が減少したことや，コアレスアクセスでなくなったためメモリアクセスにかかる時間が増加したことにより，実行時間の削減ができなかったと考えられる．

5　関連研究

MAS では主にエージェントが複数のタイプに分類され，タイプごとに挙動が異なるモデルに対して研究が行われている．Kunz ら [7] はスレッドの処理ごとにグルーピングする手法を提案している．この手法では，実行する処理の種類によってスレッドをソートする．ソートのみでは処理の境界部分でブランチダイバージェンスが生じる可能性があるため，パディングを導入している．これにより，スレッドによって異なる処理を行う場合に生じるブランチダイバージェンスの削減を行っている．Mozhgan ら [8] は FLAME GPU(Flexible Large Scale Agent Modelling Environment for the GPU)[9][10] を拡張し，エージェントの状態ごとにストリームを生成する手法を提案している．エージェントの状態の数だけリストを作成し，リストごとにエージェントの状態が同じものをまとめる．各状態で行う処理を関数化し，状態の数だけ CUDA ストリーム [13] 生成することにより並列実行を可能にしている．

一方，ブランチダイバージェンスの削減についての研究は，ソフトウェアとハードウェアの両方からの最適化が行われている．ソフトウェア側からの最適化として，Han ら [11] の研究がある．この研究では，ループ文内の if-else 文で生じるブランチダイバージェンスの削減を目的とし，2

つの最適化手法を提案している．1 つ目の手法ではループ内の if-else 文を実行する時に，同ワープ内で過半数を越えるスレッドが実行する分岐方向の命令のみを実行し，ループを 1 つ進める．この時実行されなかった分岐方向のスレッドは，その分岐方向の命令を実行するスレッド数が過半数を越えるまで実行しないことで，アイドル状態となるスレッドの割合の削減を行っている．2 つ目の手法では，if 節と else 節の命令から構造的に類似したコードを分岐の外で実行することにより分岐内の命令数の削減を行っている．ハードウェア側からの最適化としては，Rhu ら [12] のマイクロアーキテクチャの最適化の研究がある．Rhu らは，if 節の命令と else 節の命令を並列に実行できるデュアルパス実行モデルを提案し，これにより if-else 文で生じるブランチダイバージェンスの削減を行った．

6　まとめと今後の課題

本稿では近傍エージェントの探索時に，更新するエージェントの探索範囲内で重複している範囲に存在するセルを同時に参照することで，ブランチダイバージェンスの削減を行い，実行時間を短縮した．そして，二次元空間上での Boids モデルに提案手法を適用し，従来のセルリンクリスト法と提案手法の実行時間の比較を行った．その結果，シミュレーション空間全体に対してエージェント数が多い場合に本手法が有用であることが分かった．

今後の課題として，本研究における実験では相互作用範囲の半径とセルサイズが同じ場合の検証のみ行っている．そのため，セルサイズが相互作用範囲の半径の半分の場合などの検証を行うことが挙げられる．また，本手法はシミュレーション空間全体に対してエージェント数が少ない場合に実行時間の削減ができていないため，本手法の適用の有無を決定する指標の作成や，本手法適用の自動化などが挙げられる．

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] J Ferber, ”Multi-agent systems: an introduction to distributed artificial intelligence”,Harlow: Addison Wesley Longma 199

[2] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell, ”A survey of general-purpose computation on graphics hardware”,Computer Graphics Forum, vol. 26, no. 1, pp. 80-113, 2007

[3] A. J. C. Crespo, et. al., ”GPUs, a new tool of acceleration in CFD: efficiency and reliability on smoothed particle hydrodynamics methods”, PLoS ONE, 6(6), 2011

[4] NVIDIA Developer CUDA Zone, https://developer.nvidia.com/cuda-zone(2019-3-5 参照)

[5] J. M. Dominguez, et. al., ”Neighbour lists in Smoothed Particle Hydrodynamics.”, International Journal for Numerical Methods in Fluids, Vol. 67, issue 12, pp. 2026-2042, 2011

[6] Reynolds, Craig W. "Flocks, herds and schools: A distributed behavioral model." ACM SIGGRAPH computer graphics. Vol. 21. No. 4. ACM, 1987.

[7] Kunz, Georg, et al. "Multi-level parallelism for time-and cost-efficient parallel discrete event simulation on GPUs." Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation. IEEE Computer Society, 2012.

[8] Chimeh, Mozhgan K., and Paul Richmond. "Simulating heterogeneous behaviours in complex systems on GPUs." Simulation Modelling Practice and Theory 83 (2018): 3-17.

[9] FLAME GPU http://www.flamegpu.com/ (2019-3-13 参照)

[10] Richmond, P. "Flame gpu technical report and user guide." University of Sheffield, Department of Computer Science Technical Report, 2011

[11] Han, Tianyi David, and Tarek S. Abdelrahman. "Reducing branch divergence in GPU programs." Proceedings of the Fourth Workshop

on General Purpose Processing on Graphics Processing Units. ACM, 2011.

[12] Rhu, Minsoo, and Mattan Erez. "The dual-path execution model for efficient GPU control flow." 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2013.

[13] NVIDIA Corporation: CUDA C Programming Guide Version 10.1, 2019

[14] NVIDIA Corporation: CUDA C Best Practices Guide Version 10.1, 2019

[15] NVIDIA Corporation: CUDA テクニカルトレーニング Vol 1:CUDA プログラミング入門, 2008