1G-04

GPGPUを用いたMASの高速アルゴリズムライブラリの実装

Implementation of high-speed algorithm library of MAS using GPGPU

吉川　翔子†　　大岩　朗†　　芳賀　博英†

Shoko Yoshikawa　　Akira Oiwa　　Hirohide Haga

1. はじめに

近年，社会学や医療学などのあらゆる分野でマルチエージェントシミュレーション(Multi Agent Simulation，以下 MAS)が注目されている[1]．先行研究では，より現実的な環境を予想してシミュレーションを行うために，GPGPU(General Purpose computing on GPUs)を利用して MAS 用ライブラリ MASCUDA が開発された[2]．しかし現状の MAS の問題点の一つである，エージェント数増加に伴う，エージェント相互間の関係の計算量の増加が解決されていないため，ボイドモデルのような自由空間モデルは多数のエージェントで実行すると膨大な計算量となり，実行時間が大幅に増加してしまう．

本報告では，その問題点を解決するために先行研究に基づいて実装した MASCUDA のライブラリの拡張方法について述べる．

2. 先行研究

2.1. MASCUDA

MASCUDA は，GPU を利用することで MAS を高速に実行できる．MASCUDA では GPGPU のカーネル関数を，開発効率の高い Ruby 言語での記述を可能とする．空間・エージェントの生成，エージェントの移動や周囲のエージェントの取得といった，MAS でよく見られる記述がライブラリ化されている．また，実行時に必要な入出力データの CPU，GPU 間のメモリ管理を自動で行う．そのため，GPGPU の複雑さを隠蔽し，GPGPU を意識しない実装を実現する．

2.2. ボイドモデルの高速化

ボイドモデルとは，近くにいるエージェントと同じ方向に飛ぶなどのルールに従って行動することで，鳥の群れをシミュレーションするモデルである．

Alessandro らはボイドモデルをより高速化するアルゴリズムを提案した[3]．初めに，空間のマッピングをするための配列である GridSpace と LinkedList を用意する．配列の格納の方法を図 1 に示す．マッピングされた空間に対応した GridSpace 上にエージェントを格納し，エージェントが 2 体以上いるときは，LinkedList に連結リストのようにエージェントの ID を格納していく．

周囲のエージェントの取得には，GridSpace と LinkedList を使う．移動させたいエージェントの視野の範囲であれば，対応した GridSpace を見に行きそこにエージェントがいるかどうか確かめる．さらに，GridSpace 上に複数体のエージェントがいる場合は，LinkedList 内の連結リストの最後尾まで見続ける．そして，取得した周囲のエージェントとの関係を計算し，行動する．

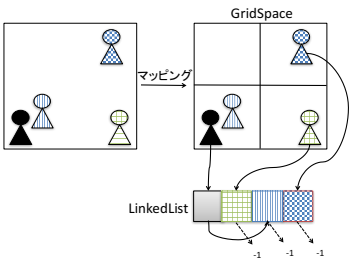

図 1　空間のマッピングとリストへの格納

3. 研究内容

3.1. 高速化アルゴリズムについて

ボイドモデルの高速化アルゴリズムは従来の方法と比べ，1 つ 1 つのエージェントが自分以外のすべてのエージェントとの関係を計算する必要がないので，実行時間を短縮することができる．しかし，GridSpace と LinkedList の作成や更新，それらを用いた周りのエージェントを取得する記述は複雑であるためコード数が増加する．そこで，MASCUDA にこのアルゴリズムをライブラリとして組み込むことで記述のコストを削減する．また，ボイドモデルのみを考慮したアルゴリズムであるため，ボイドモデル以外の MAS のモデルにも適用できるよう一般化する．

3.2. MASCUDA のライブラリの拡張

ライブラリの流れを図 2 に示す．MASCUDA は，空間とエージェントの初期化を行い，その後エージェント行動を繰り返す．

拡張したライブラリでは，エージェント初期化時に GridSpace と LinkedList を作成する．次に配列の更新，GPU への転送を行う．エージェント行動時の周囲のエージェントの取得では，これらの配列を用いる．

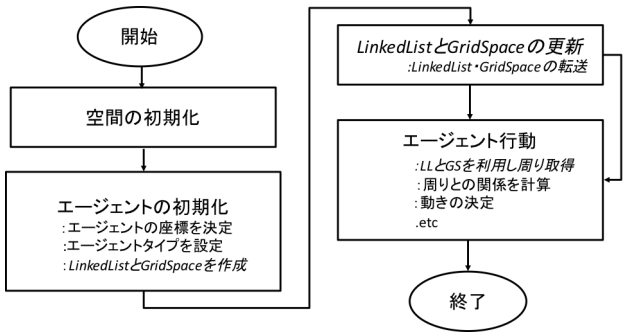

図 2　MAS 実行の流れ

4. 比較

エージェント一つ一つが自分以外のエージェントとの関係を計算するという従来の方法と，高速実行アルゴリズムでの方法を，MASCUDAで実装し，実行速度の比較とコード数の比較を行った．

実装したモデルは，自由空間を移動するモデルの代表となるボイドモデルと，感染症モデル，歩行者モデルの3つである．また，シミュレーションの条件は以下のように設定した．エージェントが見る距離は5，ループ回数は100，範囲は1000×1000とし，エージェント数は10000から1000000で変化させて測定した．また，CUDA Cを用いて実装し，比較した．

†同志社大学理工学部

Faculty of Science and Engineering Department of Intelligent Information Engineering and Science，Doshisha University

4.1. 実行時間とコード数の比較による評価

4.1.1. ボイドモデル

実行時間の結果を図3に示す.

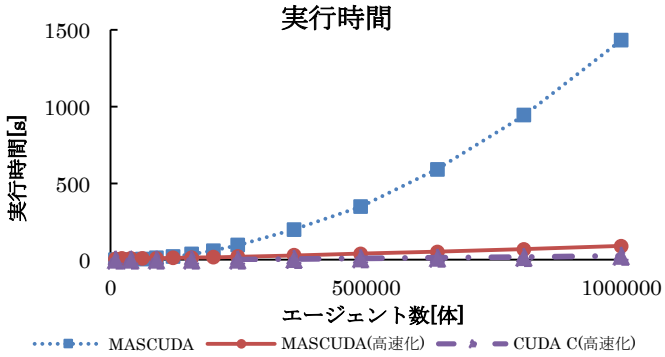

図3 ボイドモデルの実行時間

高速化アルゴリズムは，エージェント数が80000体を超えたあたりから有用になり，実行時間が改善されている．また，MASCUDA(高速化)は，CUDA C(高速化)よりも実行時間が長い.

4.1.2. 感染症モデル

感染症モデルは自分の近くにいるエージェントのうち，感染症にかかっているエージェント数と保菌者となるエージェント数の合計によって感染症にかかる確率が変化するモデルである．実行結果を図4に示す.

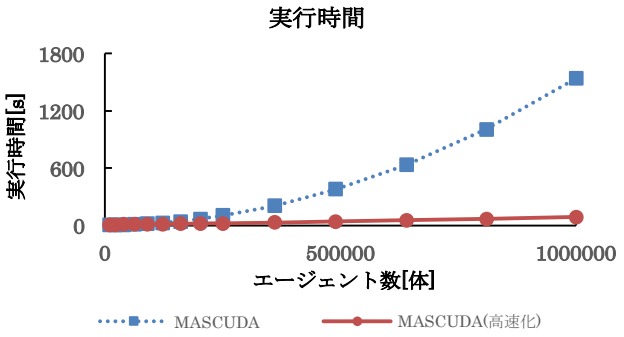

図4 感染モデルの実行時間

4.1.3. 歩行者モデル

歩行者モデルは，エージェントが他のエージェントと衝突しないように左右にある目的地に向かって進むモデルである．目的地に到達すると，反対側にある目的地に向かって進む．実行時間を図 5 に示す.

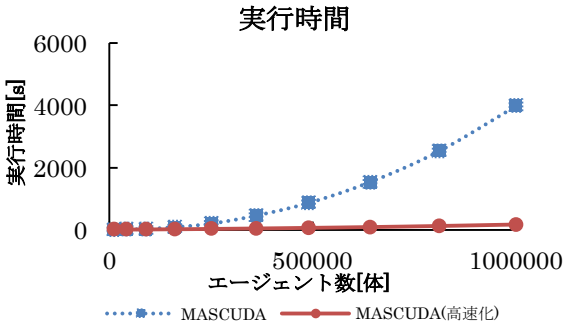

図5 歩行者モデルの実行時間

4.2. コード数

高速化アルゴリズムを用いたライブラリを作成することで，従来の方法と同じ記述方法で実行速度を早くすることができる．表1にそれぞれのコード行数を示す.

表1 ボイドモデルのソースコード行数

種類	コード数[行]
MASCUDA	32
MASCUDA(高速化)	32
CUDA C(高速化)	267

5. 考察

エージェント数が増加するほど，従来の方法と高速化アルゴリズムでの実装での時間の差も大きくなっていき，有用となる．エージェント数で見たとき，従来の方法では，それぞれのエージェントが自分意外のエージェントとの関係を計算するため実行速度が$O(n^2)$となる．一方，高速化アルゴリズムでは，実行速度は$O(n)$となる．ただし，論文のアルゴリズムを使用するとGridSpaceやLinkedListの準備と転送，LinkedListをたどる時間が必要となる.

MASCUDA(高速化)の方がCUDA C(高速化)と比べ実行時間が長いのは，Rubyがインタプリタ型の言語であるため，配列の作成・更新に時間のコストがC言語より大きいからである.

また，ボイドモデルだけでなく歩行者モデル，感染者モデルでも実行時間が短くなり，一般化できていることがわかる.

6. おわりに

拡張したライブラリの有用性を確かめることができた.

今後の課題として，CPU側で行っている配列の処理をGPU側で行えるようにすることでより実行速度を改善する必要がある．GridSpaceとLinkedListのメモリ転送が不要となり，更新がより速く実行できるため，実行時間を短縮することができると考えられる． また，ほかのモデルに適用することでさらに汎用性の高いライブラリができると考えられる.

7. 参考文献

[1]山影進「人工社会構築指南」2006，書籍工房早山
[2]大岩朗，「GPU を利用した MAS 用ライブラリ MASCUDA の開発」2016，電子情報通信学会ソサイエティ大会講演要旨，A-10-12
[3]Alessandro Ribeiro da Silva・Wallace Santos Lages・Luiz Chaimowicz，「Improving Boids Algorithm in GPU using Estimated Self Occlusion」2008，Proceedings of SBGames'08 - VII Brazilian Symposium on Computer Games and Digital Entertainment, Sociedade Brasileira de Computação, SBC, pp.41-46