卒業論文

題目

GPGPUによるエージェントシミュレーションの高速化

指導教員

大野 和彦 講師

2012年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

長谷川　崇晃 (400840)

内容梗概

近年、エージェントシミュレーションが社会学などのさまざまな分野で発展してきている．パソコンの性能向上に伴い、シミュレーションの大規模化が実現できるようになり、多くの可能性が見えてきている．しかし、多量のエージェントを動かすとなるとCPUの処理能力ではまだまだ難しいものがある．そこで注目したのがGPUである．GPUとは一般的に画像処理を専門とするデバイスではあるが、GPUを画像処理以外に活用しようという研究が盛んになってきている．エージェントシミュレーションにGPUを使い並列処理することで、より高速に動かせると期待できる．

エージェントシミュレーションとは、独自性、独立性を持ったエージェントを個々に動かし、その振る舞いを分析するシミュレーション手法である．エージェントシミュレーションは、エージェントが増加すれば、エージェントの単体の処理だけでなくエージェント同士の処理も増加してしまい、CPUに対する負担が多大になるという問題がある．

この問題に対し、エージェントの独立性に着目する．エージェントは独立性が高く、処理において互いに干渉しない所が多くそのような部分に対し並列処理を行う．

本研究において、災害シミュレーションを例に実装、評価を行った．エージェントシミュレーションに使うエージェントをGPUで並列処理をした．その結果、リアルタイムにおいてGPUはCPUの3倍高速であり、またエージェントの数が増加に伴う処理時間の増加はCPUより少ないことが分かった．

Abstract

In late years agent simulation develops in various fields such as the sociology. With the performance enhancement of the PC, Becoming it becomes able to realize a large scale of the simulation and can see much possibility. However, move a large quantity of agents, There is a more serious thing by the processing capacity of the CPU. Therefore it is GPU to have paid attention. GPU is generally a device specialized in the image processing, but a study to say to utilize GPU other than image processing becomes popular. I can expect it when I can move it faster by making parallel computation for agent simulation using GPU.

The agent simulation moves an agent with the independency and is simulation technique to analyze. If agents increase, not only the disposal of simple substances but also the disposal of between agents increases, and the burden for the CPU has a problem that it becomes great with the agent simulation.

For this problem, I pay my attention to the independency of the agent. Because independency is high, the agent performs parallel computation for a part not to interfere in processing each other.

In this study, I implemented disaster simulation for an example and evaluated it. I did parallel computation in GPU in an agent to use for agent simulation. As a result, GPU was an expressway 3 times as large as the CPU and understood that the increase of the processing time with the increase had less number of agents than a CPU again in real time.

目 次

図 目 次

表 目 次

1　はじめに

1.1　背景

近年、エージェントシミュレーションが社会学などのさまざまな分野で発展してきている．パソコンの性能向上に伴い、シミュレーションの大規模化が実現できるようになり、多くの可能性が見えてきている．しかし、多量のエージェントを動かすとなるとCPUの処理能力ではまだまだ難しいものがある．そこで注目したのがGPUである．GPUとは一般的に画像処理を専門とするデバイスではあるが、GPUを画像処理以外に活用しようという研究が盛んになってきている．エージェントシミュレーションにGPUを使い並列処理することで、より高速に動かせると期待できる．

1.2　研究目的

エージェントシミュレーションとは、独自性、独立性を持ったエージェントを個々に動かし、その振る舞いを分析するシミュレーション手法である．エージェントシミュレーションは、エージェントが増加すれば、エージェントの単体の処理だけでなくエージェント同士の処理も増加してしまい、CPUに対する負担が多大になるという問題がある．

この問題に対し、エージェントの独立性に着目する．エージェントは独立性が高く、処理において互いに干渉しない所が多くそのような部分に対し並列処理を行う．GPU には数十数百のコアがありエージェント一つに対し、コアを一つ割り当てるようにして処理をする．これにより、エージェントシミュレーションを高速に処理することができる．

1.3 本文構成

本文の構成は以下のようになっている．第 2 章でまず MegaScript 処理系の概要について述べ，第 3 章にサブマスターホストの実装，第 4 章に性能の評価を行い，最後に第 5 章にてまとめを行う．

2　概要

2.1　災害シミュレーション

2.1.1　基本設計

今回エージェントシミュレーションとして災害シミュレーションを選択した．災害シミュレーションとは災害時において人々がどのように避難するかを、人をエージェントに置き換えて再現するシミュレーションである．人々が避難する様子から適した出口を見付けたり、脱出経路を探したりする．今回の実験として、マップ上をエージェントがノードを経由して動き、一定時間経過後災害が発生したとみなしエージェントが出口に向かう。全てのエージェントが脱出すれば終了とする．

2.1.2　エージェント

エージェントとは、一定のルールを持ちそれに基づいて自立的に行動する知能持ったシステムのことである．エージェントは移動、判断、知覚などを行い、行動する．エージェントシミュレーションを行うために、シミュレーションに必要な行動や判断などをエージェントに実装する．例えば蟻が餌を探すシミュレーションを行う場合、エージェントには蟻が餌を探すように行動や判断などを実装し、蟻が餌を探す動きを再現してシミュレーションをする．

今回は災害シミュレーションを行うため、エージェントには避難を行う人の行動を実装する．エージェントは内部状態や目的地の場所などの情報を保持しており、その値を参照したり変えたりして行動や判断を行う．

2.1.3　マップ

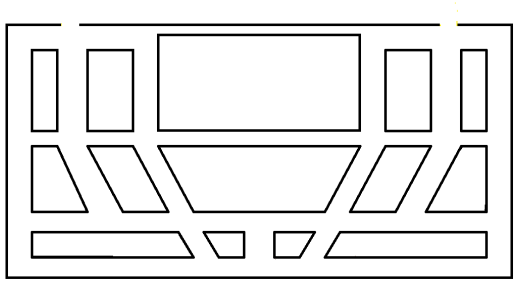

fig . 2.1: マップ

災害シミュレーションを行う場として、図のようなマップを用意する．マップを決める際、現実にある様な物を選ぶのが理想ではあるが今回は実験の行いやすさを優先し、左右対称の幾何学的なマップにした．マップ上には建物などの代わりとして障害物などを配置する．

2.1.4　ノード

エージェントの移動のため、上の図のマップ上にノードを配置する．マップ上の丸い点がノードであり、エージェントが移動する経路の交差点に

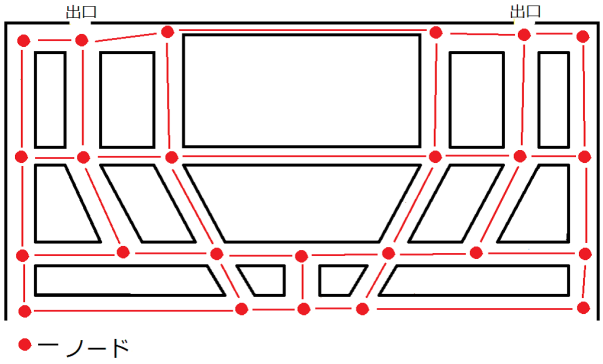

fig . 2.2: ノード

ノードを配置する．ノードとノードの間に辺を置きノード同士を連結させる．エージェントはノードを経由し辺をたどって移動する．マップの上部に 2 つの出口を設置し、災害発生時にエージェントはここから脱出する．

2.2 GPU

GPU とは、一般的に画像処理を専門とするデバイスである．この GPU を画像処理以外で使うことを GPGPU という．GPU 内には簡単な処理ができるコアが数百と並んであり、ここで処理を行う．また、GPU 内には大容量のグローバルメモリがあり、CPU と GPU 両方からメモリへのアクセスと書き換えができる．ほかに、アクセス速度がグローバルメモリより高速で容量が少なく CPU 側からでないと書き換えできないコンスタン

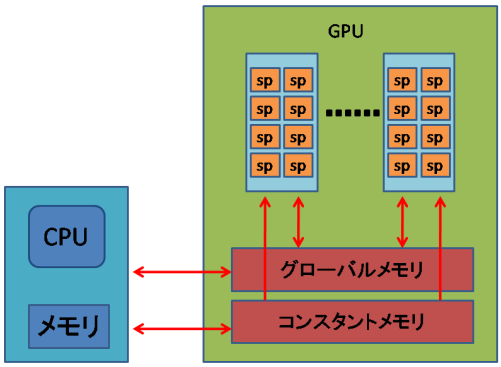

fig . 2.3: GPU

トメモリなどがある．CPU 側から計算に必要なデータを GPU のメモリに転送し計算をおこなう．図の矢印はデータの流れを表したものである．

3　実装

3.1　マップ

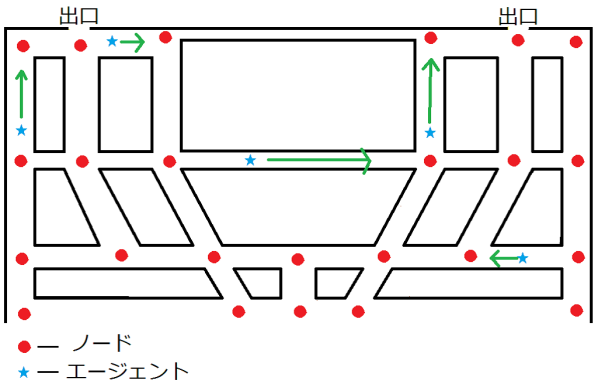

fig . 3.4: マップ

上の図のマップ上にノード、障害物、出口、を設置する．エージェントシミュレーションを行う際、エージェントの動作を可視化する必要がある．今回は OpenGL を利用しエージェントを表示しすることにした．

3.1.1　ノード

ノード間の辺の距離をコストとして、その辺にコストの値を持たせる．エージェントはノードをたどって目的地に向かうが、その経路の探索には if 文を使った条件分岐などの複雑な処理が伴う．GPU では複雑な処理

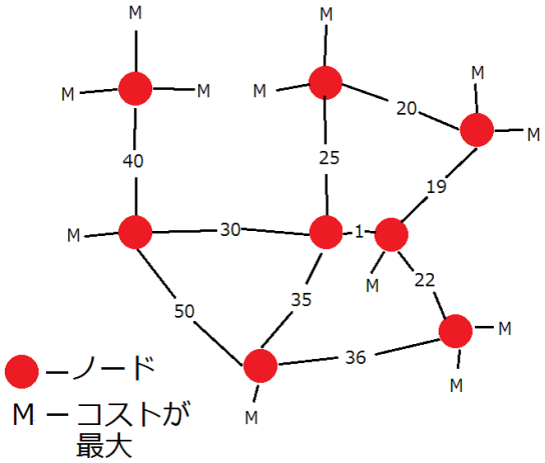

fig . 3.5: ノード

を行うと GPU の性能が大きく低下してしまうので、できる限り複雑な処理を避けたい．その方法として、ノードに他のノードの最短距離を通る場合のコストをあらかじめ持たせておき、そのコストを利用して複雑な処理を避けるようにすることができるが、具体的な方法は後にエージェントの所に記述する．

3.1.2　障害物

障害物は四角形で表現した．エージェントが障害物の内部に入れないようにする．上の図の三角形がエージェントを表している．エージェント 1 が現在のエージェントの位置であり、エージェント 2 が次の移動先の位置である．次の移動先が障害物の内部に入ると判定したときエージェ

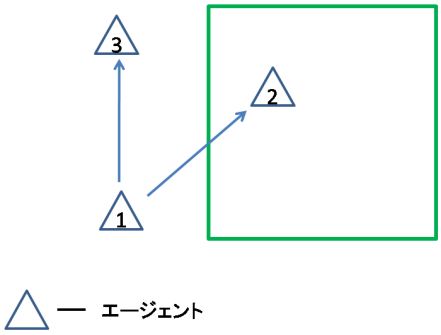

fig . 3.6: 障害物

ントの移動方向を障害物の辺に平行な向きに変えて障害物を避けるようにする．その移動先がエージェント 3 の位置である．

3.1.3 混雑さ

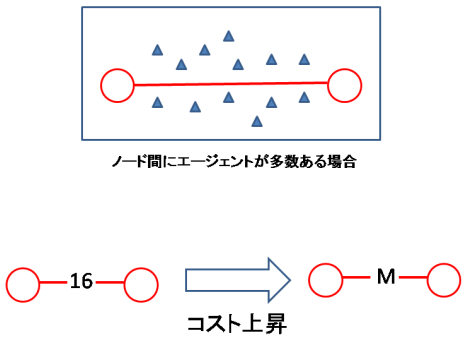

fig . 3.7: 混雑さ

ある経路上にエージェントが密集した場合、その経路は混雑していると判定する．経路上のエージェントの数を計算し一定数を超えていた場合、その経路の移動コストを上げてエージェントがその経路を通過できなくする．細い経路や脱出するときの出口付近ではエージェントが密集しやすい．コストを上げエージェントが避けるようにすると、より自然なシミュレーションが可能になる．

3.1.4　出口

出口はマップの上に2つあり、災害が発生した時にエージェントがここに近づいたらエージェントをマップの外に移動させる．

3.2　エージェント

エージェントには、内部状態、現在地、移動ベクトル、目的地のノードの場所、現在自分のいる場所のノードの場所、目的地に向かうために次に向かうノードの場所がある．エージェントは、最初はマップのノードをゴールとしてランダムに決めて移動する．災害が発生すると内部状態を変更し、2つの出口のどちらかに向かって移動する．

3.2.1　エージェントの移動

エージェントには移動ベクトルがあり、目的地に向かうために次に向かうノードの場所に向いている．このベクトルの向きに従って移動し、ノードに近づいたら次の向かうべきノードの探索を行い移動する．

3.2.2　エージェントとエージェント

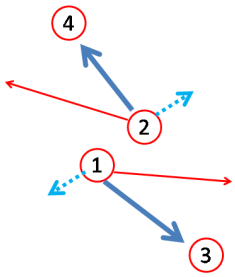

fig . 3.8: エージェントとエージェント

エージェント同士が重なり合わないようにするため、エージェント同士が一定以上近づかないようにする．上の図はエージェントが近づいた場合を表している．エージェント同士に反発力を移動ベクトル加えさせ離れさせる．細い線の矢印がエージェントの移動方向、破線の矢印が斥力であり太い線の矢印がエージェントの移動先である．エージェント１と

エージェント 2 が近づき、3 と 4 に移動する．同士に反発力を移動ベクトル加えさせ離れさせる．

3.2.3　エージェントとノード

エージェントがノードに近づいたら、次の経路の探索を始める．

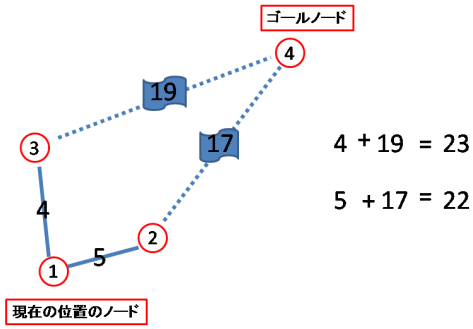

fig . 3.9: エージェントとノード

上の図は経路の探索の 1 例である．ノード 1 は現在のエージェントのいる場所である．ノード 2、ノード 3 はノード 1 に隣接している．ノード 4 はエージェントのゴールのノードであり、ノード 1、2、3 からは離れた位置にある．ノード 1 と 2 の距離のコストが 5 であり、ノード 1 と 3 の距離のコストが 4 である．ノード 2、3 にはノード 4 への最短経路のコストをそれぞれ持たせてあり、その値が 17 と 19 である．ノード間の距離の

コストと最短経路のコストの値を足し合わせた値を、現在のノードの辺でつながっているノードのすべてを比べてみて、最も少ない値のノードを目的地に向かうために次に向かうノードに選ぶ．この場合 4+19=23 と 5+17=22 で、ノード 2 を選択し移動する．目的地のノードに着いたら、別のノードをランダムに一つ選びそのノードに向かって再び移動する．

3.3 並列化

並列化はエージェント単位で行い、GPU のコア一つにエージェント一つを当てるようにする．GPU 側のメモリにデータを転送した後は全ての処理を GPU 側で行い、エージェントの動きの描写に必要なデータ以外は CPU 側には転送しない．これにより転送量、転送回数が減り処理速度は向上する．

3.3.1 メモリ

メモリは大容量で書き換え可能なグローバルメモリと、容量は少なく書き換えができないがグローバルメモリよりアクセスが高速なコンスタントメモリを使用した．CPU 側から一度転送したら変更の必要のない障害物やノードの位置情報などのデータはコンスタントメモリに置く．エージェントの位置情報など値が変化するデータは、グローバルメモリに置く．こ

れによりコアとデータのアクセスが高速になり処理速度が向上する.

4　性能評価

4.1　評価環境

評価を行うための実験環境として、ノートパソコンの ASUS U30S を使用した．ASUS U30S の CPU は intel CORE i5、GPU は GEFORCE GT 520M が搭載しており、この 2 つを使って評価を行う．

Table . 4.1: 評価用 PC 性能

CPU	GPU
intel CORE i5	GEFORCE GT 520M

4.2　評価方法

評価方法について記述します．

本実験における災害シミュレーションの流れは、最初にマップ上にエージェントが動き回り一定時間たつと災害が発生しエージェントが出口に向かうとなります．ただ、災害が発生し出口に向かいエージェントがすべて脱出するまでの時間が、エージェントの動きがランダムであるため脱出完了までの時間がシミュレーションを行うたびに大きく変わってしま．そのため、シミュレーション開始から災害発生までを評価の対象とした．

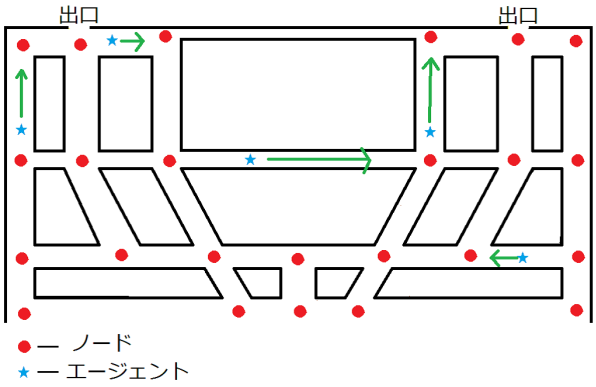

fig . 4.10: 評価用マップ

上の図のマップが評価に使用したマップである．シミュレーション開始すると、このマップ上をエージェントが動き回る．シミュレーションの処理を 5000 回行い、それまでにかかった時間を計測する．エージェントの数を増加していき処理に負荷をかけ、CPU と GPU の違いがどうなるのか調べる．

4.3 評価結果

シミュレーションを 5000 回処理するのにかかった時間をグラフにしたものを図で示す．ただし、エージェントを表示する処理は行っていない．

上のグラフからわかるように、CPU の場合エージェントが 800 個までは処理時間が約 84 秒で一定であり、GPU の場合はエージェントが 2400 個までは処理時間が約 84 秒で一定である．エージェントの個数が 800 個

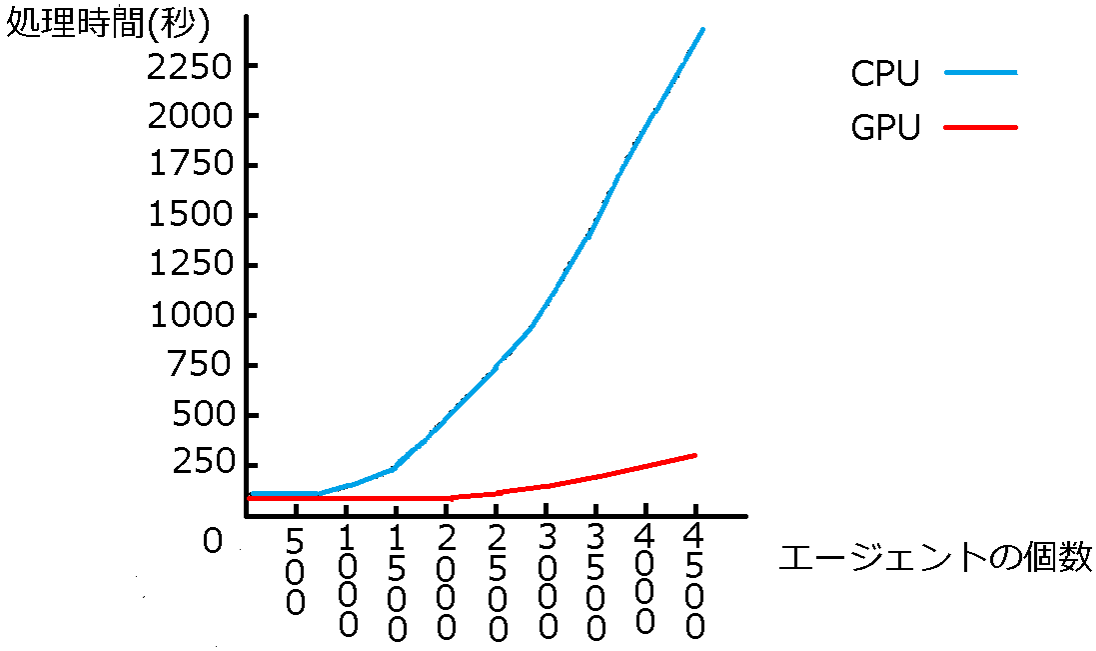

fig . 4.11: 評価結果

以下でも処理時間が約 84 秒で一定である理由は、エージェントを表示するために OpenGL を使用して処理を行っているからである．OpenGL が 1 秒間に 60 回の処理に制限しているため、60 × 84(秒)=5040 となる．したがってリアルタイムシミュレーションとしては CPU はエージェントが 800 個まで、GPU では 2400 個までになり GPU は CPU の 3 倍リアルタイム処理できることがわかった．

また、上の図のグラフを見ると CPU の場合エージェントが 800 個を超えて増加すると処理時間が大きく増加する．GPU の場合は 2400 個を超

えると処理時間は増加するが CPU にくらべて緩やかなのがわかった.

5　あとがき

本研究において、エージェントシミュレーションのエージェントをGPUで並列処理を行うことを、災害シミュレーションを例に実装、評価を行った．その結果、リアルタイムにおいてGPUはCPUの3倍高速であり、またエージェントの数が増加に伴う処理時間の増加はCPUより少ないことが分かった．

今後の課題として、エージェントの知能の向上があげられる．知能が向上することによってより複雑なシミュレーションが可能になる．

また、本研究において、シミュレーションをGPU単体で実現したがCPUがその間何もしていない．このCPUとGPUを連動させたハイブリット型のエージェントシミュレーションについても検討する必要がある．

謝辞

本研究を行うにあたり，ご指導，ご助言いただきました下さいました大野和彦講師，並びに多くの助言をいただきました近藤利夫教授，佐々木敬泰助手に深く感謝いたします．また，様々な局面にてお世話になりました計算機アーキテクチャ研究室の皆様にも心より感謝いたします．

参考文献

[1] 生天目 章 **マルチエージェントと複雑系** *2009* 森北出版

[2] 岡田 賢治 **CUDA 高速 GPU プログラミング入門** *2010* 秀和システム

[3] 青木 尊之, 額田 彰 **はじめての CUDA プログラミング** *2009* 工学社

A　プログラムリスト

B　評価用データ