F-017

群衆流動マルチエージェントシミュレーションのための並列処理実装手法の検討
Examination of parallel processing implementation method for crowd flow multi agent simulation

浅野　俊幸† 廣川 雄一‡ 西川　憲明‡
Toshiyuki Asano Yuichi Hirokawa Noriaki Nishikawa

1. はじめに

近年、大規模な避難誘導や混雑回避を検討する技術として、マルチエージェントシミュレーションが注目されている。多様で複雑な歩行者の集団挙動では、より深い洞察に基づいて歩行者が持つ認知プロセスを考慮した行動を表現できるヒューリスティックモデルにより歩行者行動モデルを構築することが望ましい。歩行者の視覚情報を取り入れたヒューリスティックモデルは、大規模な人流の衝突回避行動等には多くの計算時間が必要となるため計算機資源の抑制と並列化が課題となる。そこで本論文では、我々が開発する群衆避難マルチエージェントシミュレーションモデル（MASH: Multi-Agent based Simulator for Human behavior、以降、MASH モデルという。）の性能・特徴分析を行い、並列化の阻害要因を明らかにしてノード内並列で効果的に実行できる手法について述べる。

2. 群衆流動マルチエージェントシミュレーションモデルの概要

2.1 MASH モデル

本研究で用いるエージェントシミュレーション（MASH モデル）は、既存の歩行者行動モデルの一つであるヒューリスティックモデルをベースとして、複雑な都市空間における災害時の安全な避難誘導計画の検討に資する歩行者流動シミュレーションモデルである[1]。認知科学的視点から構築されるヒューリスティックモデルは、歩行者に内在する意思決定プロセスを明示的に表現するものであり、この意味において直感的に理解しやすいアプローチである。

MASH モデルの大きな特徴の１つは視覚情報の表現である。歩行者は視覚によって得られた情報をヒューリスティックに処理することで、自身の動きを制御していると考える。

対象歩行者 i の振る舞い位置座標 $r_i(t)$ と歩行速度 $v_i(t)$ によって記述されるとする。ここで、t は時を表す。簡易化のため、歩行者の人体は水平面上の半径 $R_i = m_i/220$ の円で表現する。ここで、m_i は歩行者 i の質量である。さらに、歩行者 i は自由歩行速度の大きさ v^0 によって特徴づけられるとする。また、歩行者の視野角は進行方向の左右 ϕ とし、衝突回避行動のために歩行者により観測される視覚距離を d_{max} とする（図１）

歩行者の衝突回避行動を記述する視覚情報としては、適切な角解像度により離散化された視野角度内の全ての方向 α に対し、歩行者 i 方向 α に歩行速度の大きさ v^0 で移動した場合に周辺障害物との衝突が最初に起こるまでの距離 $f(\alpha)$ を考える。方向 α に対して衝突の発生が予測されない場合には、距離 $f(\alpha)$ を最大値 d_{max} に設定する。

本モデルでは、歩行者により計画された目的地までの移動経路を、交差点を表すノード集合と交差点間を表すリンク集合から構成される通路ネットワーク情報を構築して表現し、歩行者の衝突回避行動を組み合わせながら基本的には最短経路探索によって目的経路を決定して進む。

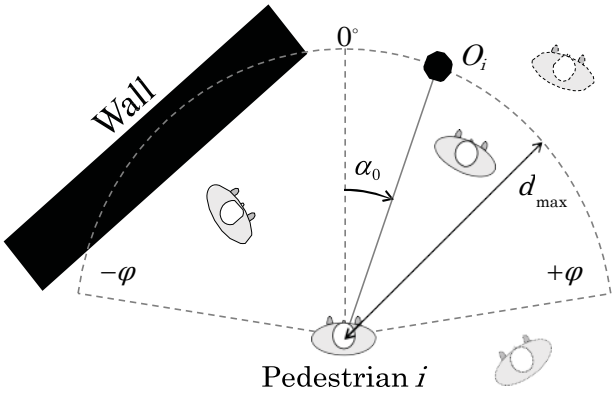

図１　歩行者の視覚情報

2.2 行動モデル

歩行者は、目的地を目指して移動する。その際、群衆の移動だけを考慮すると、歩行者の行動は２つの行動モデルを用いている。

§1　狭域行動（ミクロモデル）

道路や通路のような狭域での避難者の行動は、人間の視覚情報をベースにしたミクロモデルで解析する。避難者は他の避難者や壁面との衝突を回避するように考えて行動するため、人間の行動を精緻に予測することが可能である。

§2　広域行動（経路選択モデル）

市街地や大規模商業施設内といった広域における避難者の行動、特に土地鑑および道路状況を加味した経路選択は 3 節に示す経路選択モデルを用いて解析する。このモデルは道路情報の認識が避難者毎に異なっているのが特徴であり、来街者や地域住民といった避難者の属性を考慮することが可能である。

3. MASH モデルの性能測定と性能分析

3.1 ボトルネック調査

図２には、MASH モデルのメインプログラムの処理フローを示す。

各処理の前後にタイマーを挿入してホットスポットを洗い出した結果を表 1 に示す。評価用シミュレーションとして、ある展示場から 288 名と 1512 名が避難するデータを用いて実行した。Start_Pedestrian_Loop（徒歩避難者ループ）

† 学校法人湘南工科大学　Shonan Institute of Technology
‡ 国立研究開発法人海洋研究開発機構　Japan Agency for Marine-Earth Science and Technology

の処理が最も処理時間を占めており、1512ped（1512 名の避難）のケースでは 99.7%がこの部分である。

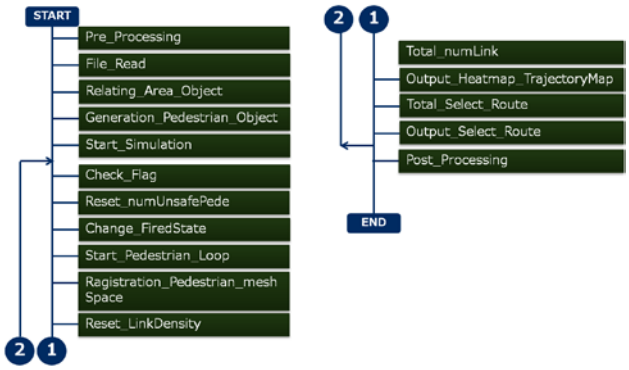

図 2　メインプログラムの処理フロー

表 1　ホットスポット調査

No	pacifico_XXped	72	288	1512
1	Pre_Processing	0.000	0.000	0.000
2	File_Read	0.039	0.038	0.038
3	Relating_Area_Object	0.031	0.031	0.031
4	Generation_Pedestrian_Object	0.008	0.031	0.202
5	Start_Simulation	0.001	0.000	0.001
6	Check_Flag	0.019	0.023	0.047
7	Reset_numUnsafePede	0.001	0.002	0.002
8	Change_FiredState	0.003	0.003	0.005
9	Start_Pedestrian_Loop	**56.952**	**410.461**	**11,193.714**
10	Ragistration_Pedestrian_meshSpace	4.284	5.260	14.631
11	Reset_LinkDensity	0.008	0.016	0.083
12	Total_numLink	0.003	0.003	0.004
13	Output_Heatmap_TrajectoryMap	0.010	0.014	0.022
14	Total_Select_Route	0.020	0.028	0.036
15	Output_Select_Route	0.011	0.036	0.245
16	Post_Processing	0.000	0.000	0.000
	Total	61.390	415.946	11,209.061

3.2 並列化手段の検討

MASH コードの最も処理時間を占める Start_Pedestrian_Loop の OpenMP 並列の検討を実施した。本サブルーチンは避難者数(numPedestrian)で繰り返すループが全体を占めており、このループの並列化を検討した。ループは「初期処理(データのコピー)」「SELECT 文による処理の実行」「後処理」に分かれている。処理時間のほとんどを占める「SELECT 文による処理の実行部分」を並列化するために、図 3 に示すように処理を 3 つに分割した。

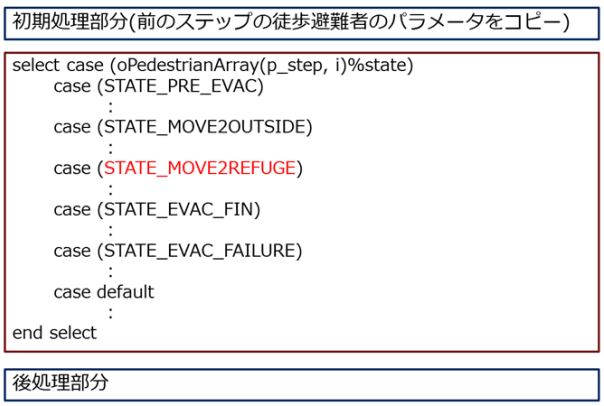

図 3　サブルーチン Start_Pedestrian_Loop の分割イメージ

3.3 並列化を阻害する依存関係

SELECT 文の CASE STATE_MOVE2REFUGE の処理中に、エスカレータにどの避難者が最も近づいているか判定する処理があり、構造体 inout_escArray に情報を格納する。この inout_escArray は避難者ごとに情報を持たず、エスカレータの ID(escIndex) で管理されている。避難者数 numPedestrian で繰り返すループ中、どの避難者が inout_escArray の情報を更新するか分からない。避難者数 numPedestrian でスレッド分割すると、どのスレッドで情報が更新されるか分からないため、並列に実行すると結果に差異が生じることになる。そこで一つ前の繰り返しで更新された情報を参照するように修正を行い、各スレッドで更新された値を並列ループの処理後にマージするように修正を実施する。

並列ループ中に更新される値は構造体 inout_escArray の lastPedeIndex と lastPedeDist の 2 要素である。

3.4 OpenMP 並列化のための修正

図 4 に示すように構造体 inout_escArray の lastPedeIndex と lastPedeDist を一つ前の繰り返しの値を参照するようにし、更新される値は作業用の構造体でスレッド毎に持つように修正を行った。また更新されるタイミング(どの処理で更新されるか)を管理するフラグ(id)を設けた。

```
!$OMP parallel
   num_threads=omp_get_num_threads()
!$OMP end parallel
   allocate(tESCArray(numESC,num_threads))
   do j = 1, num_threads
     do i = 1,numESC
       tESCArray(i,j)%lastPedeIndex = oESCArray(i)%lastPedeIndex
       tESCArray(i,j)%lastPedeDist  = oESCArray(i)%lastPedeDist
       tESCArray(i,j)%id  = 0
     enddo
   enddo
```

図 4　OpenMP 並列化のための初期処理

構造体 inout_escArray の lastPedeIndex と lastPedeDist が更新されるのは、サブルーチン CheckNextNode-ArrivalAndSearchRoute_cPedestrian の中で使用される関数 InitializeMoveOnESC_cPedestrian と関数 MoveOnESC_cPedestrian である。更新する箇所で作業用の構造体に格納するように修正を行った。

図 5 に関数 InitializeMoveOnESC_cPedestrian の修正内容、図 6 に関数 MoveOnESC_cPedestrian の修正内容を示す。

```
!ESC上最後尾の歩行者を自身にする
out_Esc%lastPedeIndex = inout_ped%iIndex

!ESC上最後尾の歩行者とESC入口との距離を0にする
out_Esc%lastPedeDist = 0.d0
out_Esc%id = 1
```

図 5　関数 InitializeMoveOnESC_cPedestrian の修正内容

また、図 7 に示すように並列ループの終了後にスレッド毎に更新した値をマージする処理を追加した。

```
if(th <= 0.d0)then
    reachFlag = .true.
    inout_ped%moveOnESC = -1
    inout_ped%cEscIndex = -1
    inout_ped%frontAgentIndexESC = -1
    if(inout_Esc%lastPedeIndex == inout_ped%iIndex)then
        out_Esc%id = 1
        out_Esc%lastPedeIndex = -1
        out_Esc%lastPedeDist = inout_Esc%length
    end if
else
    if(inout_Esc%lastPedeIndex == inout_ped%iIndex)then
        if(out_Esc%id == 1 ) then
            out_Esc%id = 3
        else
            out_Esc%id = 2
        endif
        out_Esc%lastPedeDist = GetDistance_cVector2D(inout_ped%pos, entPos)
    end if
end if
```

図 6　関数 MoveOnESC_cPedestrian の修正内容

```
do j = 1, num_threads
   do i = 1, numESC
     if(tESCArray(i,j)%id.eq.1) then
       oESCArray(i)%lastPedeIndex = tESCArray(i,j)%lastPedeIndex
       oESCArray(i)%lastPedeDist  = tESCArray(i,j)%lastPedeDist
     else if(tESCArray(i,j)%id.eq.3) then
       oESCArray(i)%lastPedeIndex = tESCArray(i,j)%lastPedeIndex
       oESCArray(i)%lastPedeDist  = tESCArray(i,j)%lastPedeDist
     else if(tESCArray(i,j)%id.eq.2) then
       oESCArray(i)%lastPedeDist  = tESCArray(i,j)%lastPedeDist
     endif
   end do
 end do    deallocate(tESCArray)
```

図 7　OpenMP 化のための値のマージ処理

また、並列ループの中に random_number を使用して乱数を求める処理がある。並列中に各スレッドが異なる乱数値を使用するように、スレッド毎に種を変更する処理を追加した。

3.5 OpenMP 並列化の検証

オリジナルの実行結果と OpenMP 並列化実施版のスレッド数 1 の実行が一致するかどうかの検証を実施するために以下の作業を行った。

I. 並列化実施版での修正と同様に一つ前の繰り返しの値を参照する部分を更新した値を参照するように変更した。
II. 乱数を固定値にして結果を比較し一致することを確認した。コンパイルオプションに-Drnd_chk を指定すると乱数は固定値になる。なおスレッド数 8 以上で足し込みの順序が変わる影響と考えられる差異が確認されたが、影響ない範囲であった。

オリジナル版：　iselect_comm_way　=　30051
OpenMP 並列化版 ：iselect_comm_way　=　30055

検証に用いたシステムは NEC LX サーバを用いた。

CPU：　Intel[R] Xeon(R) CPU Gold6154 @ 3.0GHz
　　2CPU/node, 18cores/CPU
メモリ：　192GByte
コンパイラ：gfortran 4.8.5 20150623

次に、２つの人数が異なる評価データ 288ped と 1512ped を用いて並列効果を検証した。計測した時間は標準出力に出される Total Time を使用した。（図 8）

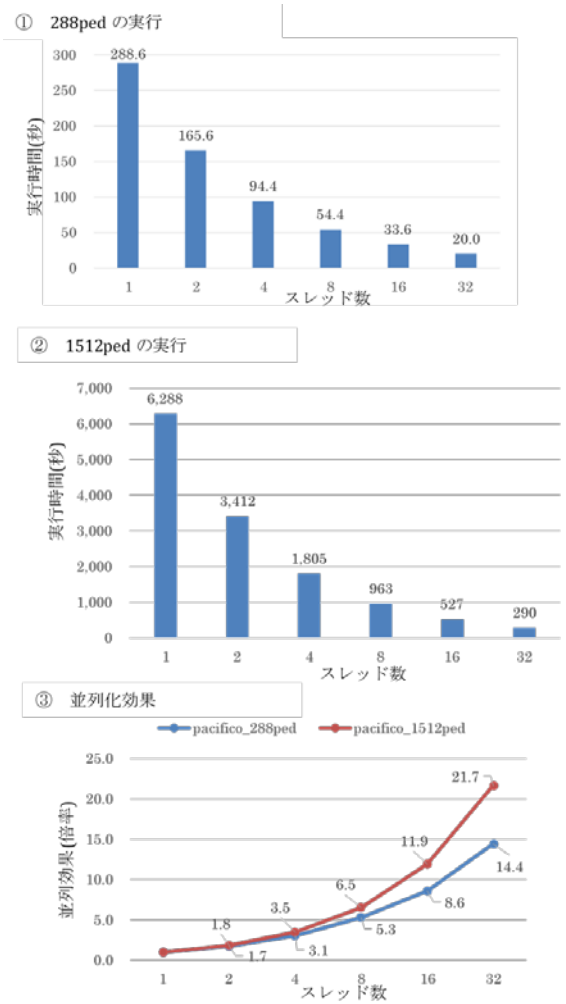

図 8　OpenMP 化の効果

以上より、データの規模が大きい 1512ped の並列効果がより高く 32 スレッド時には 21.7 倍の並列化効果が認められた。

4. MPI 化の検討

4.1 MPI 化の目的

実行時の使用メモリサイズ分割を目的とした MPI 化の検討を実施した。MAS コードは非 MPI の共有メモリプログラムである。そのためプログラムの実行は計算機 1 ノードを用いて実行し、計算機に搭載されたメモリサイズを上限としたデータ量しか扱うことができない。

より大きなデータ量を扱うためには、分散メモリ並列である MPI プログラムに実装する必要がある。MPI 化により、複数の計算機を用いた実行が可能となり、より大きいメモリ空間で実行することが可能となる。

4.2 メモリ分割の対象

プログラム内で用いる配列データを MPI プロセスに分散することで、ノードあたりの使用メモリサイズを削減することができる。分割対象とするデータはエリア情報（配列名 oAreaArray）とし、エリア単位で MPI プロセスに割り当てることを検討した。

分割対象となる配列 oAreaArray はそのエリアに所属する経路、建物、避難所に関する情報を保持している。（図 9）

```
type(area),save,allocatable,dimension(:)    ::
                                    oAreaArray     !エリア配列
type area
   integer(kind=NW) :: id    !エリア id
   integer(kind=NW) :: iIndex!エリア配列における当該エ
                                        リアの格納要素番号
   character(NCHAR) :: name  !エリア名称
   integer(kind=NW) :: floorID    !所属する階 id
   integer(kind=NW) :: floorIndex !階配列における当該所
                                        属階の格納要素番号
   integer(kind=NW) :: numBuilding!建物の数
   integer(kind=NW) :: numRefuge  !避難所数
   type(network)    :: network    !道路網
   type(building),allocatable,dimension(:) ::
                        oBuildingArray   !建物オブジェクト配列
   type(refuge),allocatable,dimension(:)  ::
                        oRefugeArray     !避難所オブジェクト配列
end type area
```

図 9　配列 oAreaArray の宣言部

4.3 MPI 化における問題点

oAreaArray を参照するループ内では、ループ変数および避難者数に依存しているため、現状は避難者を起点としてエリア情報を参照する実装となっている。この実装では、エリア情報を MPI プロセスで分割すると避難者が所属するエリアが変わる度にエリア情報の交換処理（通信処理）が発生する。

この問題点を解消するため、避難者を起点としてエリア情報を参照するのではなく、そのエリアに含まれる避難者の処理をおこなうように修正すれば本課題を解消することが可能であると結論付けた。その際、配列データとして保持する情報を以下のように修正する必要がある。

しかし、大幅なコード修正になることが予想されたため、現状では修正作業に着手できていない。今後、対応したいと考えている。

5. おわりに

本研究では、浅野等が開発したヒューリスティックモデルで 2 次元モデルであることが特徴である MASH モデルを用いて性能・特徴分析を行い、並列化の阻害要因を明らかにしてノード内並列で効果的に実行できる手法について検討した。ノード内並列については、阻害要因であるエスカレータ前の順序付けに対して、解決手法を適用し効果的な並列処理を実現した。一方、MPI 並列化については、阻害要因を突き止めた。現状では避難者に対する MPI プロセス分割が検討されるものの、エリア情報の通信が増加することが見込まれ並列化効果が期待できない。そこで、エリアに対して避難者を紐づけ、エリアで分割することで並列化効果が見込まれるという見解を得ることができた。

人流のマルチエージェントシミュレーションは、パラメータを変えたケーススタディーを数千、数万と実行することが多い。しかし、本研究のような高速化に関する研究は多くなく、有用な知見が得られたと考える。

謝辞

本研究は文部科学省ポスト「京」萌芽的課題 2 「複数の社会経済現象の相互作用のモデル構築とその応用研究」（多層マルチ時空間スケール社会・経済シミュレーション技術の研究・開発）の元で実施したものである．また，研究の一部は JSPS 科研費 JP18K04676，JP17K00328 の助成を受けたものである．

参考文献

[1] 西川憲一，廣川雄一，浅野俊幸，山田武志，印南潤二：ヒューリステックモデルによる歩行者シミュレーション，合同エージェントワークショップ＆シンポジウム 2016 (JAWS2016)、(2016)