GPU サーバのベンチマークと 外乱によって自己組織化するセルオートマトンの実装

岩瀬 雄祐 [1)], 山田 一成 [1)], 大島 聡史 [2)], 片桐 孝洋 [2)]

1) 名古屋大学 情報連携統括本部 情報推進部 情報基盤課
2) 名古屋大学 情報基盤センター 大規模計算支援環境研究部門

iwase@nagoya-u.jp, {a49984a, ohshima, katagiri}@cc.nagoya-u.ac.jp

A Benchmark of GPU Server and The Implementation of Cellular Automata that Exhibit Self-Organizing Properties Induced by Disturbance

Yusuke Iwase[1)], Kazunari Yamada[1)], Satoshi Ohshima[2)], Takahiro Katagiri[2)]

1) Information and Communications Technology Services Infrastructure Division, Nagoya University
2) High Performance Computing Division, Information Technology Center, Nagoya University

概要

名古屋大学情報基盤センターで 2018 年度に導入された GPU サーバのベンチマークを行った．NVIDA 社の CUDA Toolkit に付属する N 体シミュレーションプログラムを用いて GPU サーバの演算性能を測定し，GPU を用いることで CPU 単体よりも大幅な高速化が見込めることが分かった．また，汎用計算に特化した GPU の処理性能と広大なメモリ空間を生かし，非同期セルオートマトンの 1 つである，外乱によって自己組織化するセルオートマトンの実装を紹介する．

1 はじめに

名古屋大学情報基盤センターでは，2018 年度に GPU サーバ「Hewlett Packard Enterprise(HPE) Apollo sx40」(以下 SX40) を導入し，2019 年度に GPU(Graphics Processing Unit) を利用した計算処理である GPGPU(General-purpose computing on graphics processing units) サービスの試験運用を開始した [1]．GPGPU[2] は GPU の演算資源を汎用の計算に応用する技術のことで，GPU の高い演算性能がさまざまな分野で活用されている．

GPGPU の一例としてセルオートマトンの実装が挙げられる．セルオートマトンは空間を格子点で分割し，各点に有限オートマトンであるセルを置くことで構成される抽象システムである [3]．局所演算が中心となるセルオートマトンと GPU の親和性が高いことから，セルオートマトンの 1 つであるライフゲーム [4] において，OpenGL のピクセルシェーダによる実装 [5] や CUDA による実装 [6] 等，GPU を用いた実装が幅広く行われている．

本稿では，GPU サーバ「SX40」のベンチマークを行い，GPU サーバの利用例として，外乱によって自己組織化するセルオートマトンの実装を紹介する．

2 GPU サーバについて

GPU サーバ「SX40」のスペックを表 1 に示す [7]．CPU として Intel Xeon Gold 5122（3.6GHz）を 2 ユニット，ベースメモリを 384GB 搭載している．さらに GPU として NVIDIA Tesla V100 SXM2 を 4 ユニット搭載している．各 GPU は NVLINK[8] で相互接続することで高速な通信が可能で，マルチ GPU 動作時のオーバーヘッドを低減している．

3 GPU サーバのベンチマーク

3.1 概要

GPU サーバのベンチマークには NVIDA 社の CUDA Toolkit[9] に付属する N 体シミュレーション (nbody) プログラム [10] を用いる．このプログラムは N 個の粒子からなる重力多体系の運動方程式を解くもので，マルチ GPU でスケールアウトすることが容易なように作成されている．GPU サーバには 4 ユニットの GPU が搭載されているため，CPU の場合，

表 1　GPU サーバ「SX40」のスペック

HW	プロセッサ	2x Intel Xeon Gold 5122（3.6GHz，4 コア，105W TDP）
	メモリ	384GB（12x 2Rx4 DDR4-2666 32GB）
	ディスク	2x 2.5” 6G SATA 480GB SSD（RAID1）
	ネットワーク	Dual 10GBase-T LAN ports via Intel X550
	GPU	4x NVIDIA Tesla V100 SXM2（16GB，NVLink 対応，300W TDP）
SW	OS	Red Hat Enterprise Linux 7.5 (nbody プログラム検証時)， Red Hat Enterprise Linux 7.7 (セルオートマトン検証時)
	NVIDA ドライバ	396.26 (nbody プログラム検証時)，418.87 (セルオートマトン検証時)
	NVIDA CUDA Toolkit	CUDA 9.2 (nbody プログラム検証時)， CUDA 10.1 (セルオートマトン検証時)

シングル GPU の場合，ならびにマルチ GPU の場合についてベンチマークを行って GPU サーバの性能を測定した．

nbody プログラムのパラメータとして表 2 を用いた．数値精度として単精度と倍精度の 2 種類を設定し，粒子数を 512 個から 2 倍ずつ増やしていき 16,777,216 個まで増やした．GPU 数を 1~4 ユニットに変えて測定を行った．また，比較対象として 1 コアの CPU による測定を行った．CPU による測定は粒子数が 131,072 個で実行時間が 1 時間を超えるようになったため，それより多い粒子数の検証を行っていない．各パラメータについて 10 回の試行を行った．

表 2　ベンチマークに用いたパラメータ

パラメータ	設定値
GPU 数	1，2，3，4
精度	32bit(単精度)，64bit(倍精度)
粒子数	512，1024，2048，4096，8192， 16384，32768，65536，131072， 262144，524288，1048576， 2097152，4194304，8388608， 16777216

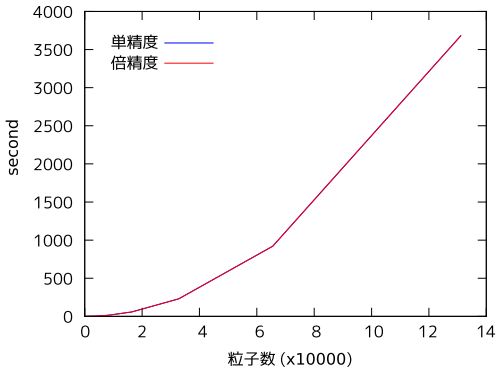

図 1　CPU によるベンチマーク (実行時間)

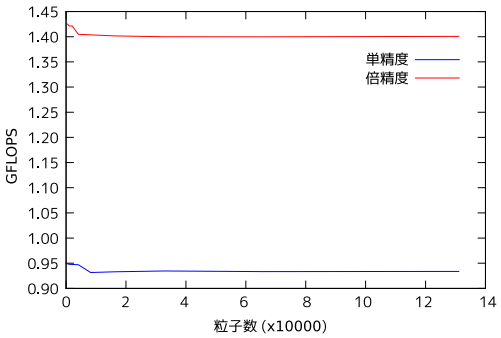

図 2　CPU によるベンチマーク (演算性能)

3.2　CPU によるベンチマーク

シングルコアにおける CPU によるベンチマークにおける実行時間を図 1，演算性能を図 2 に示す．各測定値は nbody プログラムの実行結果から取得し，10 回の試行の平均として求めている．CPU の実行時間は単精度と倍精度で違いはなく，粒子数に応じて単調に増加し，粒子数が 131,072 個の時に 1 時間程度となった．CPU の演算性能は単精度で約 0.93GFLOPS，倍精度で約 1.4GFLOPS となり，倍精度の方が演算性能が高かった．今回用いた GPU サーバは 4 コアの CPU が 2 ユニット搭載されているため，GPU サーバにおける CPU の演算性能の合計は単精度で $(0.93 \times 8 \approx)$ 約 7.4GFLOPS，倍精度で $(1.4 \times 8 \approx)$ 約 11GFLOPS と見積もられる．

3.3　GPU によるベンチマーク

GPU によるベンチマークの実行時間を図 3 (単精度の場合)，図 5 (倍精度の場合) に示す．CPU と比較して，GPU の実行時間は単精度よりも倍精度の方が長くなり，GPU が 1 ユニットの場合，粒子数が 16,777,216

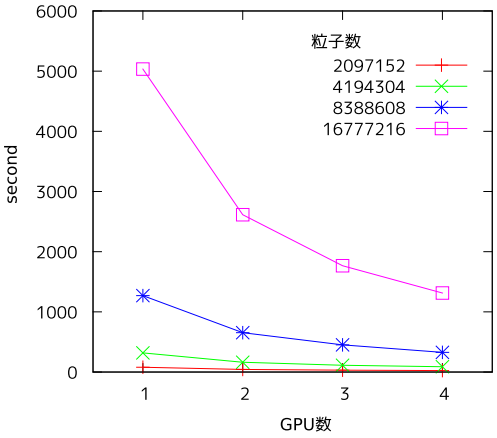

図 3　GPU によるベンチマーク (単精度，実行時間)

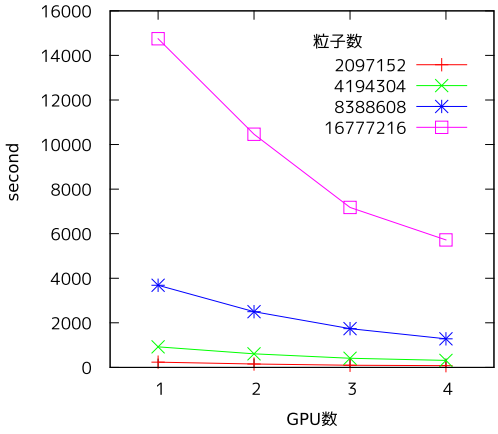

図 5　GPU によるベンチマーク (倍精度，実行時間)

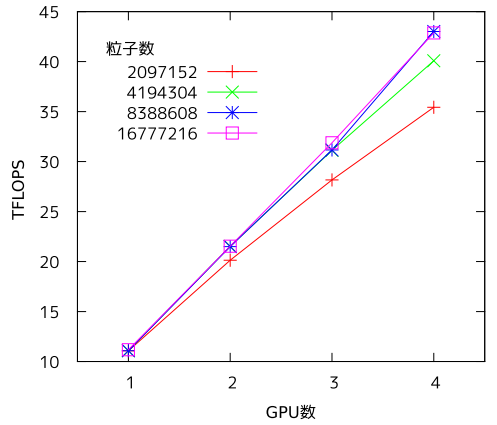

図 4　GPU によるベンチマーク (単精度，演算性能)

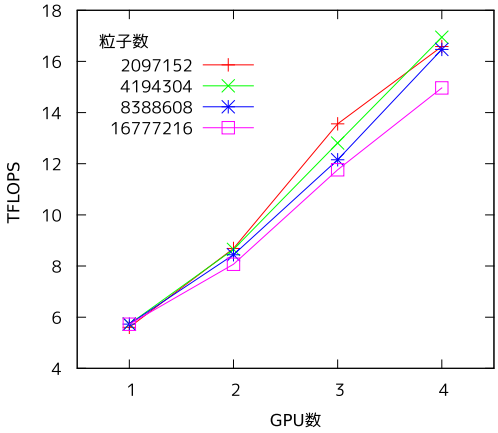

図 6　GPU によるベンチマーク (倍精度，演算性能)

個の時に，単精度で約 1 時間半，倍精度で約 4 時間程度となった．使用する GPU を増やすと実行時間は短くなるが，GPU が増えるにつれて時短の度合いは緩くなった．粒子数が 16,777,216 個において，単精度の場合は 4 ユニットで約 30 分程度，1 ユニットと比較して約 3.8 倍高速となった．また，倍精度の場合，4 ユニットで約 1 時間半程度，1 ユニットと比較して約 2.6 倍高速となった．

同試行における演算性能を図 4 (単精度の場合)，図 6 (倍精度の場合) に示す．GPU の演算性能も CPU と比較して単精度の方が演算性能が高かった．粒子数が 16,777,216 個において，GPU が 1 ユニットの場合，演算性能は単精度で約 11TFLOPS，倍精度で約 5.7TFLOPS となった．演算性能は粒子数が多くなるほど GPU 数に応じて単調に増加するようになった．粒子数が最大の場合において，単精度の場合，4 ユニットで約 43TFLOPS となり，1 ユニットと比較して約 3.9 倍となった．また，倍精度の場合，4 ユニットで約 15TFLOPS となり，1 ユニットと比較して約 2.6 倍となった．

3.4　まとめ

GPU サーバの CPU 性能は単精度で約 7.4GFLOPS，倍精度で約 11GFLOPS と見積もられる．それに対して，GPU 性能は単精度で約 43TFLOPS，倍精度で約 15TFLOPS に達し，CPU と比べて GPU を使用することで大幅な高速化が見込めると考えられる．

4　外乱によって自己組織化するセルオートマトンの実装

4.1　概要

外乱によって自己組織化するセルオートマトンは，外乱というセルの局所的な状態の改変をきっかけに，系の大域的な状態変化を生じるセルオートマトンである [11]．外乱をきっかけに大域的な状態を切り替える問題を設定し，その遷移規則を遺伝的アルゴリズムによって探索した．その結果，外乱の蓄積が一定量を超えるとその影響が系全体に広まる自己組織的な性質に

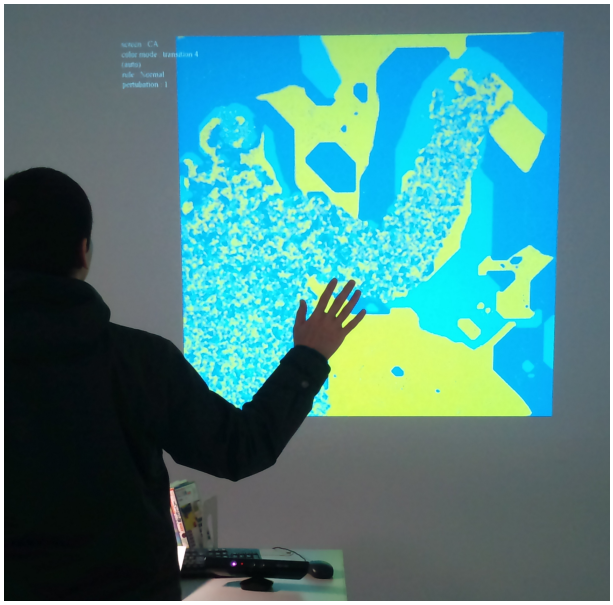

図 7　外界と相互作用するセルオートマトンと人間とのインタラクションの創発

よって，セルの状態数以上の大域的な安定状態を周期的に推移する系が得られた．

セルオートマトンの遷移規則を遺伝的アルゴリズムで探索するためには，セルオートマトンにおいて自己組織的な構造と振舞いを評価・実現するために十分なセル数と状態遷移のステップ数が必要となり，遺伝的アルゴリズムの探索のために十分な個体数，世代数，さらには試行数が必要となり，多くの計算資源が求められる．文献 [11] では，CPU ベースのクラスタ型計算機を用いて，何週間もかけてタスクの設定，パラメータの調整を行って，遷移規則の探索を行っていた．

外乱を人間が加えることによって，セルオートマトンと人間とのインタラクションが創発する [12]．このアートの一例を図 7 に示す．図中のスクリーンにセルオートマトンの大域的な状態が表示されている．また，スクリーンの直下に Kinect センサーを置き，鑑賞者等を含めた距離画像を取得し，その距離に応じてセルオートマトン上で外乱を発生させている．このセルオートマトンは GPU を用いることでスクリーン上に画素数に近い大量のセルをリアルタイムに状態遷移させることが可能となり，鑑賞者の振舞いをきっかけにしてセルオートマトンの大域的な状態変化が発生することで，セルオートマトンと鑑賞者との間にインタラクションのループが生じるものとなった．

そこで，外乱によって自己組織化するセルオートマトンをマルチ GPU で動作するように再実装し，GPU サーバ「SX40」においてベンチマークを行った．

4.2　実装方法

外乱によって自己組織化するセルオートマトンは 2 次元 3 状態 9 近傍の非同期セルオートマトンである．遷移規則は自身の状態とその周囲 8 近傍の状態密度によって遷移後のセルの状態を定める等方則 (outer-totalistic rule) をベースとし，セルの状態構成について正順方向の推移性 (状態 0 → 状態 1 → 状態 2) を制約として加えた対称則を用いる．この遷移規則は特定のセルの状態に依存せず，状態 0 の規則に推移性を加えることで他の状態 1 と状態 2 の規則を定めるもので，遷移規則の圧縮とサイクリックなセルオートマトンの大域的な状態遷移に寄与する．外乱はセルの状態を正順方向にインクリメントする摂動を用いる．

セルオートマトンは NVIDA 社の CUDA を用いて実装する．メモリ転送，カーネル処理の並列実行は cudaSetDevice 命令で実行する GPU を指定することで可能になる．また，各 GPU から別ユニットへのメモリアクセスは cudaDeviceEnablePeerAccess 命令で宣言後，メモリ転送コマンド (cudaMemcpy 等) にて 2 ユニットのメモリを指定することで，NVLink を介して GPU 間で直接通信が可能となる．

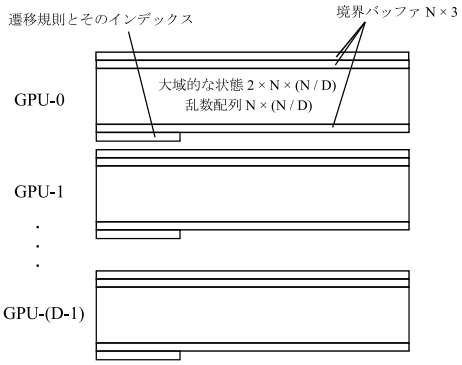

図 8　マルチ GPU を用いる時のメモリ構成

セルオートマトンの大域的な状態 (様相) は $N \times N$ のセルで構成する．GPU 上には，セルオートマトンの大域的な状態をダブルバッファリング方式で更新するため unsigned char を $2 \times (N \times N)$ 個，3 状態の遷移規則とそのインデックスとして unsigned int を 98 個 (遷移規則の 45 個を含む) 確保する．また，乱数生成のため unsigned int を $N \times N$ 個確保する．マルチ GPU を用いる場合，図 8 のように，大域的な状態のダブルバッファと乱数配列を D ユニットで分割し，各 GPU について，大域的な状態の境界通信のための境界バッファに unsigned char を $3 \times N$ 個 (上部境界，下部境界，値交換用のバッファの 3 つ準備)，遷移規則

とそのインデックス用の配列を確保する．

セルオートマトンの大域的な状態の更新処理の流れを以下に示す．

1. 初期状態の設定
   （a）遷移規則とそのインデックスを準備
   （b）大域的な状態をランダムに設定
   （c）乱数配列の初期値をランダムに設定
   （d）各初期状態を CPU から GPU へメモリ転送
2. 大域的な状態の更新 (S ステップの繰返し)
   （a）(外乱発生時) 大域的な状態を GPU から CPU へメモリ転送し，外乱をパターンとして加えて，CPU から GPU へメモリ転送
   （b）参照しているセル空間の境界状態を GPU 間でメモリ転送
   （c）線形合同法で乱数配列の各値を更新
   （d）各セルについて確率 $P_a = 0.2$ で遷移規則により状態遷移
   （e）(画面表示時) 大域的な状態を GPU から CPU へメモリ転送してファイル出力し，可視化プログラム[*1]にて表示

各初期状態は CPU 上で設定を行い，CPU から GPU へメモリ転送することで定める．大域的な状態の更新は，セル空間の境界状態を GPU 間でメモリ転送した後，各 GPU にてカーネルの中で乱数生成とセルの状態遷移を並列処理する．

4.3 GPU によるベンチマーク

$N = 512$，$D = 4$ としたセルオートマトンの大域的な状態の例を図 9 に示す．サンプルとして「NU」の文字を象った外乱を左上から右下へ移動させつつ各セルについて 5% の確率で加えている．ランダムに設定された初期状態は遷移規則[*2]によって同じセルの状態によって構成されるクラスタ構造が生じ，外乱をきっかけにクラスタ構造の輪郭が変化し，大域的な状態変化を生じている．

使用する GPU の数を変化させて GPU サーバのベンチマークを行った．パラメータとして $N = 4096$, 8192, 16384, 32768, $D = 1$, 2, 4, $S = 16$ を用いた．セルオートマトンの処理速度を図 10 に示す．$N = 4046$ (16,777,216 セル ≃17 百万セル) では，1 秒未満で大域的な状態を 1 ステップ更新している．$N = 32768$ (1,073,741,824 セル ≃11 億セル) では，

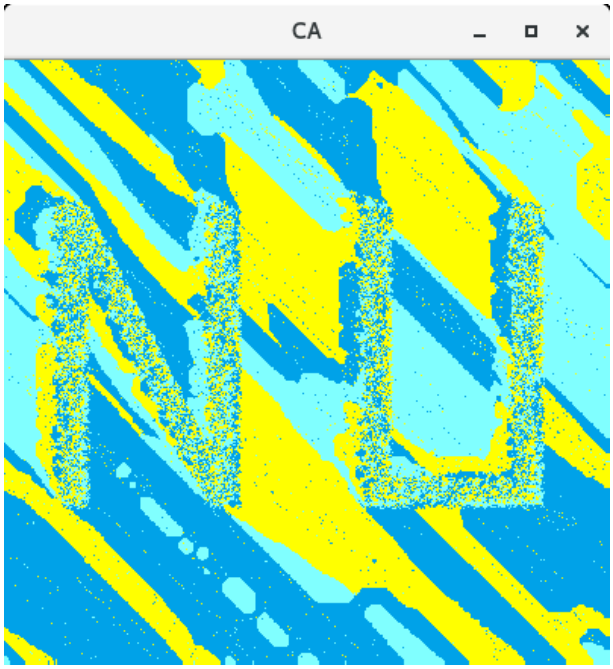

図 9　セルオートマトンの大域的な状態

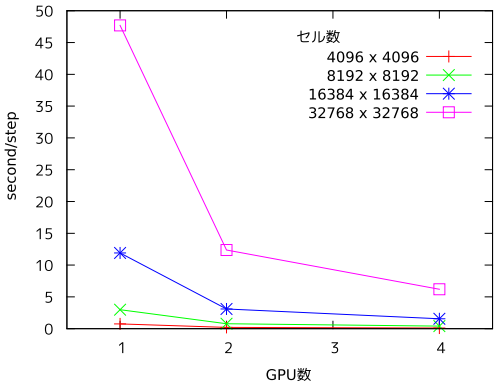

図 10　GPU によるベンチマーク (セルオートマトンの処理速度)

GPU が 1 ユニットにおいて 47.7 秒/ステップ，GPU が 2 ユニットにおいて 12.4 秒/ステップとなり，1 ユニットの 4 倍程度，GPU が 4 ユニットにおいて 6.2 秒/ステップとなり，1 ユニットの 8 倍程度高速化されている．

GPU サーバにおいて 4 ユニットの GPU を全て用いて，セルオートマトンのセル数を増やして実行したところ，$N = 92680$ (8,589,582,400 セル ≃86 億セル) を更新できることを確認した．

4.4 まとめ

GPU サーバを用いることで，広大なセル空間を高速に処理できることが分かった．セルオートマトンはセル同士の局所的な通信で大域的な状態が更新される

[*1] OpenGL にて作成．
[*2] 文献 [11] にて正順サイクル (ascending cycle) を示した規則を用いている．

ため，GPU 同士の通信量が少なく，また，通信に必要となるバッファのメモリを少なくでき，GPU の増加が系の高速化に直接的に寄与すると考えられる．非同期セルオートマトンの実装には遷移規則，ならびに乱数配列が必要となったが，同期セルオートマトンとなるライフゲーム等では，よりシンプルな実装が可能であり，より大規模な系の実行が可能になると考えられる．

5 おわりに

名古屋大学情報基盤センターで 2018 年度に導入された GPU サーバ「SX40」のベンチマークを行い，CPU と比べて GPU を使用することで大幅な高速化が見込めることを示した．また，汎用計算に特化した GPU の処理性能と広大なメモリ空間を生かす例として，外乱によって自己組織化するセルオートマトンの実装例を示した．

汎用計算向けの GPU は，家庭用 GPU に比べて搭載メモリが多いため，並列演算による高速化と共に，大規模な系の展開に適している．情報基盤センターの GPU サーバを活用した研究成果を期待したい．

参考文献

[1] GPU サーバの試験運用について，http://www.icts.nagoya-u.ac.jp/ja/sc/news/maintenance/gpu.html, 名古屋大学情報基盤センター, 2019 年.

[2] J. D. Owens，D. Luebke，N. Govindaraju，M. Harris，J. Krüger，A. E. Lefohn and T. J. Purcell, A Survey of General-Purpose Computation on Graphics Hardware，In Eurographics 2005，State of the Art Reports，pp. 21-51，2005.

[3] S. Wolfram，A New Kind of Science, Wolfram-Media Inc.， 2002.

[4] E. R. Berlekamp, J. H. Conway, R. K. Guy, Winning Ways for Your MathematicalPlays, Academic Press, Vol. 2，1982.

[5] http://download.nvidia.com/developer/SDK/Individual_Samples/samples.html，2003 年.

[6] T. Fujita, D. Nishikori, K. Nakano and Y. Ito，Efficient GPU Implementations for the Conway's Game of Life，2015 Third International Symposium on Computing and Networking (CANDAR)，pp. 11-20，2015.

[7] GPU サーバ（HPE Apollo sx40）利用者マニュアル，名古屋大学情報基盤センター，2019 年.

[8] NVIDIA NVLink, https://www.nvidia.com/ja-jp/design-visualization/nvlink-bridges/，2019 年.

[9] CUDA Toolkit，https://developer.nvidia.com/cuda-toolkit，2019 年.

[10] Lars Nyland, Mark Harris and Jan Prins, (Chapter 31) Fast N-Body Simulation with CUDA, GPU Games 3, Addison-Wesley Professional, pp. 677-695, 2007.

[11] Y. Iwase，R. Suzuki and T. Arita，Evolutionary Search for Cellular Automata with Self-Organizing Properties toward Controlling Decentralized Pervasive Systems and Its Applications, International Journal of Systems Biology and Biomedical Technologies，Vol. 3(1)，pp. 1-19，2015.

[12] 岩瀬 雄祐，鈴木 麗璽，有田 隆也，外界と相互作用するセルオートマトンと人間とのインタラクションの創発，芸術科学会論文誌，Vol. 11(3)，pp. 69-78，2012 年.