情報処理学会論文誌　プログラミング　Vol.11　No.2　1–8 (June 2018)

# レジスタ最適化を用いたCUDAによる格子ボルツマン法の高速化手法

富永 浩文[1,a)]　中村 あすか[2]　前川 仁孝[3]

受付日 2017年12月20日, 採録日 2018年3月17日

**概要**：本論文では, CUDA（Compute Unifide Device Architecture）を用いた格子ボルツマン法（LBM：Lattice Boltzmann Method）を高速化するために，メモリアクセス遅延を削減する手法を提案する．格子ボルツマン法は，解析領域を格子に分割し離散化されたボルツマン方程式を解く手法である．ボルツマン方程式の計算は，周囲の格子点の情報を参照するため，メモリアクセスコストが高いメモリバウンドな計算である．このため，LBM のメモリアクセスコストを削減する手法の 1 つとしてテンポラルブロッキングが用いられている．CUDA によるテンポラルブロッキングを用いた格子ボルツマン法は，ブロックに分割した領域をスレッドブロックに割り当て時間発展方程式を計算する．本計算は，メモリアクセスのコストを抑えるが，シェアードメモリにデータを格納することで，シェアードメモリに対する同期処理やレイテンシによるアクセスコストが処理の大部分を占める．そこで，本論文では，メモリアクセスコストが低いレジスタを用いてテンポラルブロッキングを行うことで処理を高速化する手法を提案する．提案手法は，テンポラルブロッキングにおける複数時間ステップの計算をレジスタ上に保持して行うことで処理を高速化する．

キーワード：格子ボルツマン法，テンポラルブロッキング，レジスタ最適化

# High-speed Method of Lattice Boltzmann Method on the CUDA Using Registers Optimization

Hirobumi Tominaga[1,a)]　Asuka Nakamura[2]　Yoshitaka Maekawa[3]

Received: December 20, 2017, Accepted: March 17, 2018

***Abstract:*** This paper proposes a speed up method of the Lattice Boltzmann Method (LBM) by improving the memory access delay. The LBM is a method of dividing the analysis area to the grid and solving the discretized Boltzmann equation. Calculation of the Boltzmann equation is a memory bound calculations that mean high memory access cost by referring to the surrounding grid points. For this reason, the temporal blocking method is one of the reducing methods of the memory access cost for the LBM. The LBM using temporal blocking by the CUDA calculates the time evolution equation by assigning divided areas to thread blocks. This calculation reduces the access cost to store the data in the shared memory. However, the synchronization and the memory access latency of shared memory occupy the majority of processing. Therefore, this presentation proposes a speed up method of temporal blocking using the registers whose access cost is low. The proposed method accelerates by storing calculation in the registers over multiple time steps of temporal blocking.

***Keywords:*** lattice Boltzmann method, temporal blocking, register optimization

[1] 千葉工業大学大学院情報科学研究科情報科学専攻
Graduate School of Information and Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[2] 千葉工業大学特別研究員
Researcher, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[3] 千葉工業大学情報科学部情報工学科
Department of Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[a)] tominaga@mae.cs.it-chiba.ac.jp

## 1. はじめに

流体を解析するステンシル計算の 1 つとして，格子ボルツマン法（LBM：Lattice Boltzmann Method）が用いられている [1]．LBM は，解析空間を格子状に離散化し，その上を移動する粒子集合の動きをシュミレーションすることで，流体を解析する．本手法は，すべての格子点上の粒子が同時に衝突や並進を行うと見なして計算するため，時間ステップごとの計算を並列化することができる．このため，高い並列処理性能が得られる GPU（Graphics Processor Unit）を用いることで，大規模な問題を高速に解析できることが報告されている [2], [3], [4], [5].

GPU を用いた LBM で高い性能を得るためには，GPU のスレッド階層とメモリ階層を活かすようなプログラミングが必要である [6]．GPU を用いた LBM は，解析領域をブロック形状に分割してスレッドブロックに割り当て，ブロック内の格子点をスレッドに割り当てることで高い性能を得ることができる．一方で，CPU と GPU は別々のアドレス空間を持つため，解析する問題の規模が大きくなるほど，CPU と GPU 間で PCI-Express のように低速なバスを介したデータ転送が頻繁に必要となる．このため，GPU-CPU 間の通信回数を削減する手法の 1 つとしてテンポラルブロッキングが提案されている [5], [7], [8].

テンポラルブロッキングは，解析領域を複数のブロックに分割し，ブロック領域ごとに GPU にデータを転送し計算する手法である．本手法は，割り当てられた領域だけでなく袖領域と呼ばれる冗長な領域に対する解析も行うことで，複数の時間ステップにわたる解析をブロック領域ごとに独立に計算する．本手法を用いることで，ブロック分割による空間的局所性に加えて高い時間的局所性を得られ，データの通信回数を削減することができる．

グローバルメモリやシェアードメモリ，レジスタなどの複数のメモリ階層を持つ CUDA を用いた LBM では，単一 GPU 内でもデータ通信が頻繁に起こる．このため，グローバルメモリ，シェアードメモリ，レジスタ間のデータ通信を削減することで LBM をより高速化できると考えられる．単一 GPU による CUDA のメモリ階層を利用してテンポラルブロッキングを行った報告の 1 つに，ポアソン方程式を用いた文献 [9] があり，高い効果が得られることが報告されている．LBM においてもテンポラルブロッキングを適用することで，単一 GPU における CUDA において高速化が見込めると考えられる．

単一 GPU 内での複数のメモリ階層においてデータ通信を削減するために，LBM の計算における実行メモリバンド幅の効率化を行いワープの単位でデータに効率良くアクセスできるようにデータレイアウトを工夫する手法 [10] や，並進と衝突演算のカーネルをまとめる手法 [11] などが提案されている．これらの手法は，単一ステップにおける効率化を図るものであるため，テンポラルブロッキングを利用することで LBM をさらに高速化できると考えられる．これらをふまえ本論文では，LBM に対し単一 GPU 内のメモリ階層それぞれにテンポラルブロッキングを適用する．シェアードメモリの階層を用いるテンポラルブロッキング手法は，スレッドブロック内で計算に必要なデータをシェアードメモリへ格納する．スレッドブロック内の各スレッドは，シェアードメモリに毎ステップデータアクセスを行い計算する．このため，本手法は，1 格子点あたりの計算に必要なデータ量が多いため，シェアードメモリに格納可能な格子点数と起動可能なスレッド数の制限により，テンポラルブロッキングの段数を増やすことが難しい．一方，テンポラルブロッキングの段数を増やすことで，各スレッドが占有可能なレジスタ数を多くすることができる．このため，シェアードメモリとレジスタを用いて階層的にテンポラルブロッキングを行うことで高速化が期待できる．

レジスタの階層を用いるテンポラルブロッキング手法は，シェアードメモリの階層を利用する手法に加え，各スレッドが，複数段計算するために必要なデータをレジスタに格納し，レジスタのデータのみを用いて複数段計算する．このため，シェアードメモリへのアクセス回数が減少し，スレッド間の同期処理も削減でき，計算の高速化が期待できる．また，CUDA の最小の処理単位である各レジスタは，ワープと呼ばれる 32 スレッドが同時に演算を行うというハードウェア的特性がある．ワープ内のスレッドは，32 のスレッドがつねに同期しながら処理を実行する．このため，ワープ内のスレッドどうしでレジスタの内容を交換や参照ができるワープシャッフル（WS：Warp Shuffle）を用いる．WS を用いることで，スレッド間でのデータのやり取りが可能になり，データの再利用をすることで冗長な演算を削減できる．

## 2. CUDA を用いた格子ボルツマン法

格子ボルツマン法は，解析領域を離散化して格子点として扱う．格子ボルツマン法における各格子点の計算は，離散化方式に BGK モデルが用いられ，2 次元 9 速度モデル（D2Q9）や 3 次元 19 速度モデル（D3Q19）モデル [12] による解析が行われている [13], [14]．BGK モデルは，各格子点の計算に周囲の格子点の情報を用いるため，解析領域のサイズが大きくなるほど解析に時間がかかる．このため，CUDA による高い並列性能を利用した格子ボルツマン法の高速化が行われている [2], [3], [4], [5]．以下では，CUDA のアーキテクチャと CUDA を用いた D2Q9 モデルによる格子ボルツマン法の計算手順について述べる．

### 2.1 CUDA

CUDA は，NVIDIA 社の GPU を効率良く動作させるために開発された並列コンピューティングアーキテクチャで

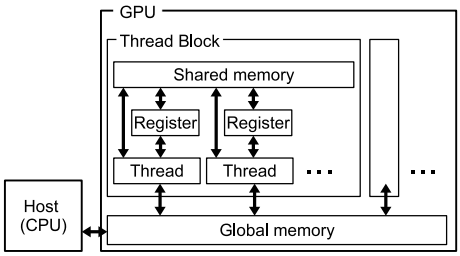


図 1　CUDA プログラミングモデル
**Fig. 1**　CUDA programing model.

ある．**図 1** に，CUDA のプログラミングモデルを示す．図中の矢印はデータ転送を表す．図 1 に示すように，CUDA はスレッド階層とメモリ階層の 2 つの階層から構成される．また，スレッドブロック間でデータを共有するためには，スレッドブロックの外側の階層にあるグローバルメモリを介する必要がある．このため，グローバルメモリのアクセス回数を削減して計算を高速化するには，スレッドブロックに局所性の高い計算を割り当てることでメモリアクセスコストを低くする必要がある [6]．また，GPU は，多くのスレッドを起動するために，Pascal 世代の GPU では約 14 MB 相当の多数のレジスタを搭載する．このレジスタを GPU 内の全スレッドが共有して用いるため，各スレッドが占有するレジスタ数によって同時に起動できるスレッド数が制限される．CUDA プログラミングでは，各スレッドがレジスタを多く用いるか少なく用いるかは，プログラマが決定できる．

### 2.2　CUDA を用いた格子ボルツマン法

格子ボルツマン法は，解析領域を等間隔な格子で離散化し，タイムステップごとに格子上にある粒子の衝突と並進を解析する．本手法では，離散化方法に D2Q9 を用いる場合の各タイムステップの粒子の分布状態 $f_i$ を式 (1) の格子ボルツマン方程式で表す．

$$f_i(\mathbf{x}+\mathbf{c}_i, t+1) = f_i(\mathbf{x}, t) + \hat{\Omega}_i[f_i(\mathbf{x}, t)] (i = 0, 1, \cdots, 8) \tag{1}$$

式中の $t$ はタイムステップ，$\mathbf{x}$ は位置ベクトル，$\mathbf{c}_i$ は 9 方向の方向 $i$ に向かう速度ベクトル，$\hat{\Omega}_i$ は衝突演算子である．衝突演算子 $\hat{\Omega}_i$ には，一般的に BGK 近似が用いられる [14]．BGK 近似を用いると，衝突演算子 $\hat{\Omega}_i$ は，式 (2) で表す．

$$\hat{\Omega}_i[f_i(\mathbf{x}, t)] = -\frac{1}{\tau}[f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)] (i = 0, 1, \cdots, 8) \tag{2}$$

ここで，$\tau$ は緩和時間係数，$f_i^{eq}$ は局所平衡分布関数である．また，密度 $\rho(\mathbf{x}, t)$，流速 $u(\mathbf{x}, t)$ は，それぞれ式 (3)，式 (4) で表す．

$$\rho(\mathbf{x}, t) = \sum_i f_i(\mathbf{x}, t) \tag{3}$$

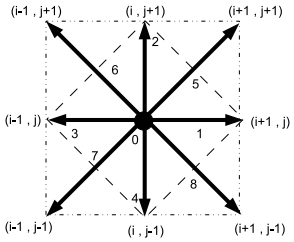


図 2　D2Q9 モデル
**Fig. 2**　D2Q9 model.

$$\mathbf{u}(\mathbf{x}, t) = \frac{1}{\rho(\mathbf{x}, t)} \sum_i \mathbf{c}_i f_i(\mathbf{x}, t) \tag{4}$$

**図 2** に D2Q9 モデルを示す．図中の点線は格子を，格子内の矢印は方向 $i$ に向かう速度ベクトル $\mathbf{c}_i$ を表す．図のように，$\mathbf{c}_i$ は，方向 0 ならば $\mathbf{c}_0 = (0, 0, 0)$，方向 1 ならば $\mathbf{c}_1 = (1, 0, 0)$，方向 2 ならば $\mathbf{c}_2 = (-1, 0, 0)$ となる．D2Q9 モデルの局所平衡分布関数 $f_i^{eq}$ は，式 (5) に示す重み係数 $\omega_i$ を用いて，式 (6) のように表す．

$$\omega_i = \begin{cases} \frac{4}{9} & (i = 0) \\ \frac{1}{9} & (1 \le i \le 4) \\ \frac{1}{36} & (5 \le i \le 8) \end{cases} \tag{5}$$

$$f_i^{eq}(\mathbf{x}, t) = \omega_i \rho(\mathbf{x}, t) \left[1 - 1.5\mathbf{u}^2(\mathbf{x}, t) + 3\mathbf{c}_i\mathbf{u}(\mathbf{x}, t) + 4.5(\mathbf{c}_i\mathbf{u}(\mathbf{x}, t))^2\right] \tag{6}$$

格子ボルツマン法は，これらの計算式を各格子点でそれぞれ計算するため並列性が高い．

CUDA を用いた格子ボルツマン法は，階層的なメモリ構造を効率良く用いるために，解析領域をブロック分割して各スレッドブロックに計算領域を割り当て計算する．スレッドブロックは，割り当てられた計算領域をシェアードメモリに書き込み 1 ステップ分計算し，結果をグローバルメモリに書き戻す．本手法は，シェアードメモリを用いることで空間的局所性が得られるが，毎時間ステップごとにグローバルメモリへアクセスするため，メモリアクセスコストが高い．

## 3.　メモリアクセスコストを削減するテンポラルブロッキング手法

提案手法は，CUDA の各メモリ階層においてテンポラルブロッキングを適用することで LBM を高速化する．CUDA を用いた LBM を高速化するために，シェアードメモリを用いたテンポラルブロッキング（STB），レジスタを用いたテンポラルブロッキング（SRTB），および，SRTB に WS を用いた最適化を行う．提案手法のテンポラルブロッキングのフローチャートを**図 3** に示す．本例は，STB の段数が $ts$ 段，SRTB の段数が $tr$ 段における例を示しており，図中の $timesteps$ はシミュレーション時間，$ts$ はシェ

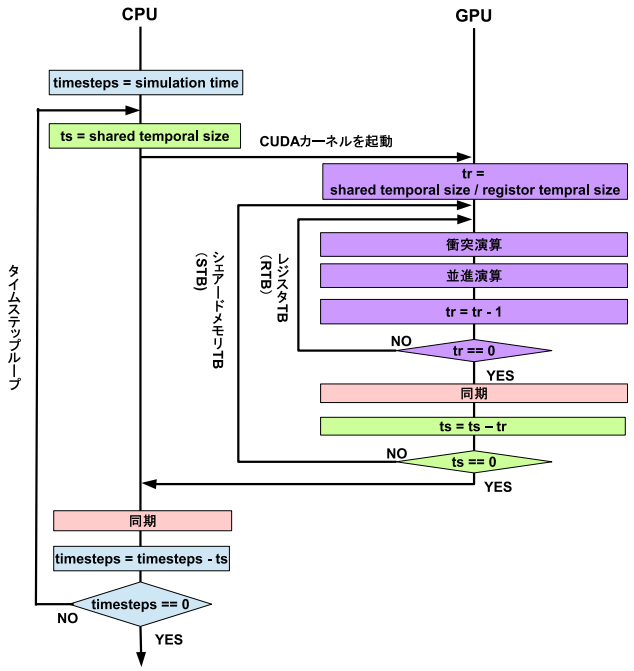


図 3　RTB のループ段数 $ts$，SRTB のループ段数 $tr$ によるテンポラルブロッキングのフローチャート

**Fig. 3**　Flowchart of RTB ts-stages and SRTB tr-stages.

アードメモリのテンポラルブロッキング段数，$tr$ はレジスタのテンポラルブロッキング段数を格納する．提案手法は，CPU 上で解析時間を $timesteps$ で制御し，GPU 上で LBM の衝突並進を計算する．LBM の衝突並進演算を行う CUDA カーネルは，STB ループと RTB ループの 2 重ループで構成される．STB ループは，シェアードメモリのテンポラルブロッキングの段数分ループする．また，RTB ループは，レジスタのテンポラルブロッキングの段数分ループする．このため，本例における $tr = 1$ のときは，STB の動作となる．各スレッドが割り当てられた格子点における衝突並進は，RTB ループ中で計算する．STB ループで各スレッドに割り当てられる計算に必要なデータは，グローバルメモリから読み込むため，STB ループの外で一度のみシェアードメモリへ格納する．これにより，グローバルメモリへのメモリアクセスコストが削減できる．RTB ループで各スレッドが割り当てられた格子点の計算に必要なデータは，シェアードメモリからロードするため，RTB ループの外でレジスタにデータを格納する．これにより，RTB ループ内では STB ループで必要になる同期処理を行うことなく複数段の計算が可能となる．また，各格子点の衝突並進演算を行う RTB ループの段数 $tr$ は，シェアードメモリのテンポラルブロッキングの段数 $ts$ を超える段数を指定すると袖領域のデータがないため指定できない．このため，各テンポラルブロッキングの段数の指定は $tr \leq ts$ となる．

以下では，STB，SRTB について述べ，さらに SRTB による冗長な計算を削減する手法について述べる．

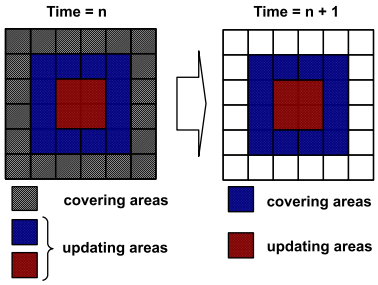


図 4　テンポラルブロッキングの例

**Fig. 4**　An example of temporal brocking.

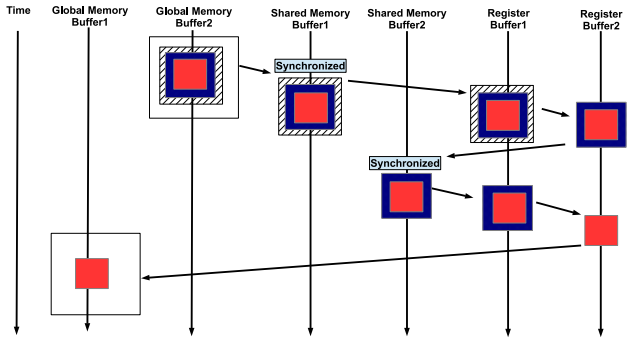


図 5　STB によるテンポラルブロッキング 2 段の計算例

**Fig. 5**　An example of 2-stages calculation by STB.

### 3.1　シェアードメモリを用いたテンポラルブロッキング手法

シェアードメモリを用いたテンポラルブロッキングは，複数時間ステップの計算に必要なデータをシェアードメモリに格納することで空間的局所性を高める手法である [15]．図 4 に，複数のステップを計算するテンポラルブロッキングの例を示す．図中の四角は解析領域を離散化した格子点である．本例は，赤色の領域を 2 ステップ先まで更新する．本論文では，更新するステップ数を段数と呼ぶ．図 4 の赤色の領域を 2 段更新するために，赤色だけでなく青色と黒の領域をメモリに格納する．1 段目は黒色の領域を用いて赤と青色の領域を計算し，2 段目は青色の領域を用いて赤色の領域を計算する．

本手法は，ブロック分割した領域よりも広い領域を低レイテンシでアクセスできるシェアードメモリに格納する．図 5 に，解析領域をシェアードメモリに割り当ててテンポラルブロッキングにより計算する処理の流れを示す．本手法は，まず，ステップごとに結果を格納する領域を切り替える必要があるため，シェアードメモリに 2 個のバッファ領域を確保する．次に，スレッドブロックは，グローバルメモリから計算に必要な袖領域を含む格子点データをシェアードメモリのバッファ領域 1 に格納する．割り当てる際に更新される要素は，グローバルメモリ上でハッチングされているエリアである．次に，シェアードメモリのバッファ領域 1 からレジスタに格子点データをロードし，レジスタ上で格子点データを更新する．このとき，他のスレッ

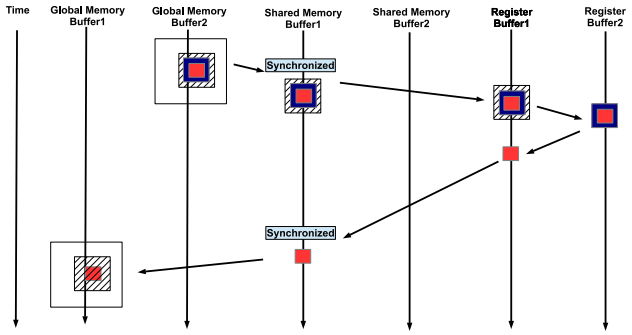


図 6　SRTB によるテンポラルブロッキング 2 段の計算例
**Fig. 6**　An example of 2-stages calculation by SRTB.

ドがシェアードメモリのバッファ領域 1 からデータを読み込むことがあるため，更新が終了するとシェアードメモリのバッファ領域 2 へ更新したデータを格納する．次の時間ステップでは，シェアードメモリのバッファ領域 2 のデータをレジスタにロードして格子点情報を計算し，計算が終了すると更新データをシェアードメモリのバッファ領域 1 へ格納する．本手法は，テンポラルブロッキングの段数が多くなるほどシェアードメモリへのアクセスを繰り返すため，メモリアクセス遅延が発生する．

### 3.2　レジスタを用いたテンポラルブロッキング手法

シェアードメモリのアクセス遅延を最小限にするために，シェアードメモリ上で行われるテンポラルブロッキングをレジスタ上で行う．本手法は，各スレッドブロックで用いるシェアードメモリへのアクセスは計算カーネルの初回のみ行い，各スレッドが計算に必要なデータをそれぞれレジスタに格納する．このため，本手法は，シェアードメモリによる手法よりも多くのレジスタを必要とする．**図 6** に，提案手法で 2 段のテンポラルブロッキングを計算する流れを示す．レジスタを用いたテンポラルブロッキングは，まず，初回のグローバルメモリから分割された領域データをシェアードメモリ上に袖領域を含めてロードする．次に，レジスタ上でテンポラルブロッキングを行うために，レジスタ上にバッファ領域 1 と 2 を確保する．各スレッドが計算するレジスタのデータは，他のスレッド間で参照，格納ができないため，各スレッドは自身が計算する領域のデータを自身が操作できるレジスタにすべて格納する．各スレッドは，レジスタ上のバッファ領域 1 のデータを用いて 1 段目を計算し，1 段目の計算結果をバッファ領域 2 に格納し，バッファ領域 2 を用いて 2 段目を計算する．2 段目の計算が完了したら，グローバルメモリへ自身の計算した領域のデータを格納する．これにより，時間ステップが進む際に他のスレッドの計算結果を参照せずに計算可能であり，同期コストも削減できる．

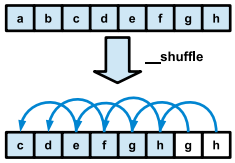


図 7　シャッフル命令の例
**Fig. 7**　An example of shuffle instruction.

### 3.3　Warp Shuffle を用いた SRTB

SRTB は，各レジスタが計算に必要なデータをすべて保持するため，スレッド間でのデータのやり取りが発生せず，シェアードメモリのアクセスや同期処理が起こらないという特徴がある．しかし，各スレッドどうしでは，自身の更新する領域だけでなく袖領域も計算する必要があるため，隣り合う格子点を計算するスレッドどうしが同一の計算を実行する．そこで，スレッドブロック内の各スレッドは，32 スレッドがワープと呼ばれる処理単位で実行され，ワープ内の実行は必ず同期するというハードウェア的特性を利用する．各スレッドは，RTB の各イタレーションで自身が計算した局所平衡分布関数の値をワープ内の他のスレッドにブロードキャストすることで，計算回数が削減できる．他のスレッドが計算した結果を同一ワープ内でブロードキャストできる機能に，WS がある．**図 7** に，シャッフル命令の例を示す．図に示すように，WS は，同一ワープ内の各スレッドがレジスタ上で管理するデータを他のスレッドがシェアードメモリを介さずに参照することで，レジスタ使用量とメモリアクセスコストを削減できる．ただし，WS は 1 次元配列で連続した並びであることが必須であるため，メモリが連続に格納されている方向ベクトルのデータに対して WS を適用する．提案手法による実装は，各方向ベクトルはそれぞれ異なる配列で保持する．このため，WS の適用は，メモリアドレスが連続となる図 2 中の $f_1$，$f_3$ の計算に対して行う．また，32 ワープの壁側にあるデータは，同一ワープ内で使われることはないので，そのまま計算結果を破棄する．

## 4.　評価

GPU を用いた格子ボルツマン法に対するレジスタを用いたテンポラルブロッキングの有効性を確認するために，2 次元ポアズイユ流れ [16] を解析する．評価環境は，CPU が Intel Xeon E5-2687W，GPU が Titan X Pascal である．**表 1** に，評価環境の構成を示す．本評価で解析するポアズイユ流れのモデルは，D2Q9 モデルで一辺 320 格子に離散化し，解析領域の上下の境界条件を bounce-back 条件 [17]，解析領域の左右の境界条件を周期境界条件とする．また，評価に用いたプログラムでは，各格子点のデータを x 座標優先で単精度の 1 次元配列に格納する．

表 1　評価環境
**Table 1**　The evaluation environment.

| | | |
|---|---|---|
| CPU | Processor | Intel Xeon E5-2687W |
| | Memory | 32 GB |
| GPU | Processor | Nvidia Titan X Pascal |
| | Global Memory | 12 GB |
| | Shared Memory | 48 kB |
| | L1 cache | 16 kB |
| | L2 cache | 3 MB |
| | CUDA core | 3584 |
| | CUDA Version | CUDA 9.0 |

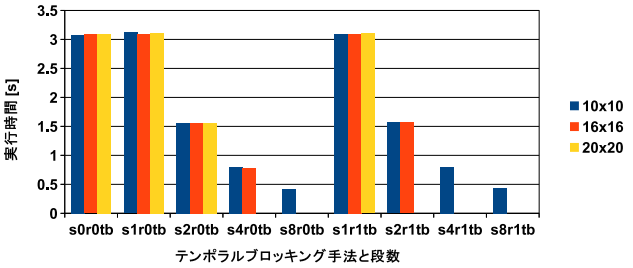


図 8　実行時間
**Fig. 8**　Execute time.

### 4.1　STB と SRTB の実行時間の評価

テンポラルブロッキングに対してレジスタを利用する有効性を確認するために，シェアードメモリを用いたテンポラルブロッキングとレジスタを用いたテンポラルブロッキングの実行時間を測定する．本評価では，レジスタを利用した場合の効果を明確にするために，WS を用いずに実行する．**図 8** に，シェアードメモリとレジスタを用いたテンポラルブロッキング段数とブロック数によるポアズイユ流れの実行時間を示す．図中では，シェアードメモリのテンポラルブロッキングが $n$ 段，レジスタのテンポラルブロッキングが $m$ 段の手法を $snrm$tb と表記する．つまり，s2r1tb は，シェアードメモリ上で 2 段，レジスタ上で 1 段のテンポラルブロッキングを行うことを表す．また，レジスタを用いないシェアードメモリのテンポラルブロッキングの段数での表記は，レジスタ段数をすべて 0 とする．つまり，s0r0tb はテンポラルブロッキングを適用しない手法を表す．

本評価の測定条件では，$n$ の設定を 8 以上にするとシェアードメモリの容量不足により実行不能となる．同様に，s4r0tb，s8r0tb，s2r1tb，s4r1tb，s8r1tb も実行不能であるため，図中では測定結果を空欄とする．

図 8 より，すべてのブロックサイズの条件においてテンポラルブロッキング段数が増加するごとに，処理時間が短縮することが分かる．ブロックサイズ $10 \times 10$ のとき，シェアードメモリを用いた s8r0tb は，テンポラルブロッキングを適用していない s0r0tb に対して，最も高い約 7.36 倍の高速化が得られることが確認できた．これは，グローバルメモリへのアクセスコストが減少したことや，アクティブブロック数を複数起動することによる，グローバルメモリやシェアードメモリのアクセス時間隠蔽の効果であると考えられる．

表 2　メモリアクセスのストールの割合（%）
**Table 2**　Ratio of stall of memory access (%).

| | ストールの割合 |
|---|---|
| s0r0tb | 6.0 |
| s8r0tb | 0.3 |

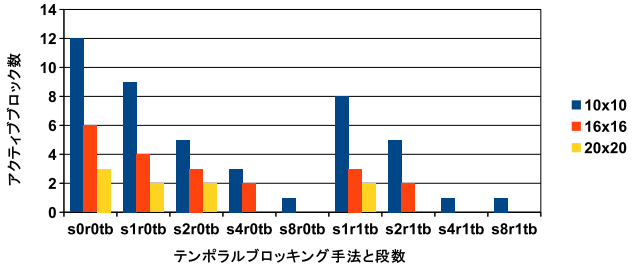


図 9　アクティブブロック数
**Fig. 9**　Number of active blocks.

メモリアクセスの隠蔽による効果を確認するために，NVVP [18] を用いて s8r0tb と s0r0tb において全体の処理時間に占めるメモリ操作に要した割合とアクティブブロック数を測定を調査する．まず，**表 2** に，処理時間に占めるメモリアクセスのストールの割合を示す．表 2 より，s8r0tb と s0r0tb における全体の処理時間に占めるメモリアクセスのストールの割合が s0r0tb が約 6%，s8r0tb が約 0.4%であることが確認できる．これは，メモリアクセスコストの低いシェアードメモリ上で複数ステップの計算を行い，グローバルメモリへのアクセスが削減できたことによる効果であることが分かる．次に，**図 9** に，各手法におけるアクティブブロック数を測定した結果を示す．図 9 より，s0r0tb におけるアクティブブロック数が最も多く，段数が増加するとアクティブブロック数が減少することが分かる．また，図 8 において，高速な手法ほどアクティブブロック数が低いことが分かる．通常，CUDA は，高いメモリアクセスコストを隠蔽して処理を高速化するために，多くのアクティブスレッドブロック数を確保することが重要である．しかし，LBM は，格子計算の中でも特に計算に必要なメモリのコストが高いメモリバウンドな手法である．このため，テンポラルブロッキング段数を増やすとスレッドブロックが計算に必要とするデータ量が多くなり，アクティブブロック数が減少したといえる．アクティブスレッド数が低いにもかかわらず高速化した要因は，メモリアクセスにかかる割合と計算の割合が関係すると考えられる．そこで，s8r0tb と s8r1tb の計算の割合と，メモリアクセスによる処理のストールの割合を NVVP で測定する．**表 3** に，NVVP で測定した結果を示す．表 3 より，s8r1tb の

表 3　計算とメモリアクセスの割合（%）
**Table 3**　Ratio of calclate and memory access (%).

| | 計算の割合 | メモリアクセスの割合 |
|---|---|---|
| s8r0tb | 70 | 3.0 |
| s8r1tb | 80 | 1.0 |

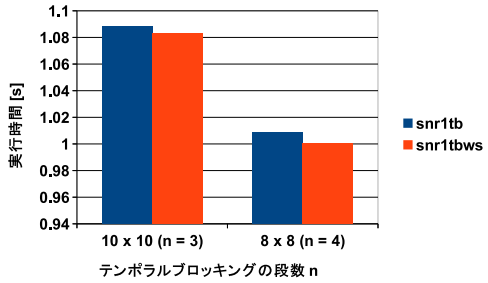


図 10　WS を適用したテンポラルブロッキング $n$ 段の実行時間
**Fig. 10**　Execute time of $n$-stages temporal blocking using WS.

計算の割合約 80%，メモリアクセスの割合が約 1%であるのに対し，s8r0tb の計算の割合は約 70%，メモリアクセスの割合が約 3%である．このため，メモリアクセスにかかる時間を減らし計算の割合が増加したことで，計算が高速化できたと考えられる．

### 4.2　WS を用いたテンポラルブロッキングの実行時間の評価

レジスタを用いたテンポラルブロッキングにおいて WS を用いたテンポラルブロッキングの有効性を評価するために，WS を用いた手法と用いていない手法の実行時間を測定する．WS を用いた SRTB は，スレッドブロックサイズ $10 \times 10$，テンポラルブロッキング 8 段は，レジスタ容量が不足するため実行できなかった．このため，評価は，スレッドブロックサイズ $10 \times 10$，かつ壁を含めて warp サイズである 32 スレッドで割り切れるサイズである段数 3 として条件を設定する．また，本条件より段数が増加したときの効果を明確にするために，スレッドブロックサイズ $8 \times 8$，段数 4 の条件も設定して実行時間を測定する．**図 10** に，WS を用いた手法と用いていない手法の実行時間を示す．図中の横軸は，先に述べた 2 条件であり，縦軸は実行時間である．図 10 より，WS 命令を用いた手法は，レジスタのみを用いた手法と比べ，スレッドブロックサイズ $8 \times 8$，テンポラルブロッキング段数 4 のとき，処理時間が約 1.008 倍となることが確認できた．低い高速化率となったのは，本手法が，並進衝突計算時の局所平衡分布関数のうち，$f_1$ と $f_3$ の値の交換のみ WS を適用したためであると考えられる．また僅かながら高速化したのは，周囲のスレッドが計算した局所平衡分布関数の値を参照することで計算回数が削減できたためであると考えられる．そこで，計算コストの削減を確認するために実行した命令数を NVVP を用いて測定する．**表 4** に，ブロックサイズ $8 \times 8$ のときに各手法が実行した命令数を示す．評価の結果，WS を用いた $snrm$tbws は，WS を用いていない $snrm$tb に比べて実行した命令数が約 12%削減することが確認できた．これは，WS により計算回数が削減できたため，僅かに計算を高速化できたと確認できる．

表 4　演算回数の削減率
**Table 4**　Reduced ratio of number of operations.

| TB サイズ | WS 有 | WS 無 | 削減率 (%) |
|---|---|---|---|
| $8 \times 8$ | 199,738,424 | 227,726,168 | 12.29 |

## 5.　おわりに

本論文では，CUDA を用いた格子ボルツマン法を高速化するために，シェアードメモリ，レジスタ上でテンポラルブロッキングを適用する手法について提案した．評価の結果，シェアードメモリを用いた s8r0tb 手法は，s0r0tb 手法に比べて最大約 7.36 倍の高速化を得ることが確認できた．また，レジスタ使用を最適化し，レジスタの使用量と計算回数を削減するために WS を用いた手法を実装し，評価した．評価の結果，ワープシャッフルを用いた手法は，ワープシャッフルを用いない手法に比べ，計算回数を約 12%削減することが確認できた．今後の課題として，自由空間ではない複雑な形状を有する実問題へ適用や，マルチ GPU 時に CPU-GPU 間，GPU 内のメモリ階層にそれぞれテンポラルブロッキングを適用，3 次元問題への拡張性の検討，他のステンシル計算への応用などがあげられる．

### 参考文献

[1] 福井貴也，越村俊一，松山昌史：格子ボルツマン法による津波氾濫流の 2D-3D ハイブリッド・シミュレーション，土木学会論文集 B2（海岸工学），Vol.66, No.1, pp.61–65 (2010).
[2] Mawson, M. and Revell, A.J.: Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture nVidia GPUs, *CoRR*, Vol.abs/1309.1983 (2013).
[3] Bailey, P., Myre, J., Walsh, S.D.C., Lilja, D.J. and Saar, M.O.: Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors, *2009 International Conference on Parallel Processing*, pp.550–557 (2009).
[4] Calore, E., Marchi, D., Schifano, S.F. and Tripiccione, R.: Optimizing communications in multi-GPU Lattice Boltzmann simulations, *2015 International Conference on High Performance Computing Simulation* (*HPCS*), pp.55–62 (2015).
[5] Jin, G., Lin, J. and Endo, T.: Efficient utilization of memory hierarchy to enable the computation on bigger domains for stencil computation in CPU-GPU based systems, *2014 International Conference on High Performance Computing and Applications* (*ICHPCA*), pp.1–6 (2014).
[6] Cheng, J., Grossman, M. and McKercher, T.: *Professional CUDA C Programming*, Wiley (2014).
[7] Jin, G., Endo, T. and Matsuoka, S.: A Multi-Level Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPU, *2013*

*IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pp.1080–1087 (2013).
[8] 小野寺直幸，青木尊之，下川辺隆史，小林宏充：格子ボルツマン法による 1m 格子を用いた都市部 10km 四方の大規模 LES 気流シミュレーション，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，Vol.2013, pp.123–131 (2013).
[9] Maruyama, N. and Aoki, T.: Optimizing Stencil Computations for NVIDIA Kepler GPUs, *1st International Workshop on High-Performance Stencil Computations 2014* (2013).
[10] Rinaldi, P., Dari, E., Venere, M. and Clausse, A.: A Lattice-Boltzmann solver for 3D fluid simulation on GPU, *Simulation Modelling Practice and Theory*, Vol.25, pp.163–171 (2012).
[11] Suksumlarn, J., Suwannik, W. and Maleewong, M.: Lattice Boltzmann method for two-dimensional shallow water equations with CUDA, *2015 2nd International Conference on Advanced Informatics: Concepts, Theory and Applications* (*ICAICTA*), pp.1–5 (2015).
[12] d'Humières, D.: Multiple–relaxation–time lattice Boltzmann models in three dimensions, *Philosophical Trans. Royal Society of London A: Mathematical, Physical and Engineering Sciences*, Vol.360, No.1792, pp.437–451 (2002).
[13] Flekkøy, E.G.: Lattice Bhatnagar-Gross-Krook models for miscible fluids, *Phys. Rev. E*, Vol.47, pp.4247–4257 (1993).
[14] Qian, Y.H., D'Humieres, D. and Lallemand, P.: Lattice BGK Models for Navier-Stokes Equation, *EPL* (*Europhysics Letters*), Vol.17, No.6, pp.479–484 (1992).
[15] Wittmann, M., Hager, G., Treibig, J. and Wellein, G.: Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters, *CoRR*, Vol.abs/1006.3148 (2010).
[16] 清水圭太，伊澤精一郎，福西　祐，熊　鰲魁：格子ボルツマン法を用いた平面ポアズイユ流れの不安定性の解析，東北支部総会・講演会　講演論文集，Vol.2005, No.40, pp.28–29 (2005).
[17] Yin, X. and Zhang, J.: An improved bounce-back scheme for complex boundary conditions in lattice Boltzmann method, *Journal of Computational Physics*, Vol.231, No.11, pp.4295–4303 (2012).
[18] NVIDIA Visual Profiler — NVIDIA Developer (online), available from ⟨https://developer.nvidia.com/nvidia-visual-profiler⟩ (accessed 2018-04-09).

**富永 浩文**　（学生会員）

1984 年生．2007 年千葉工業大学情報科学部情報工学科学卒業．2009 年同大学大学院情報科学研究科情報科学専攻博士前期課程修了．2010 年同大学大学院情報科学研究科情報科学専攻博士後期課程入学．主として，GPU 等のアクセラレータを用いた各種アプリケーション高速化の研究に従事．

**中村 あすか**　（正会員）

1986 年生．2009 年千葉工業大学情報科学部情報工学科卒業．2011 年同大学大学院情報科学研究科情報科学専攻博士前期課程修了．同大学特別研究員，現在に至る．博士（工学）．主として，並列探索に関する研究に従事．

**前川 仁孝**　（正会員）

1967 年生．1990 年早稲田大学理工学部電気工学科卒業．1992 年同大学大学院理工学研究科電気工学専攻修士課程修了．1993 年日本学術振興会特別研究員．1994 年早稲田大学理工学部助手．1998 年千葉工業大学情報工学科講師．2002 年同大学助教授．2011 年同大学教授．現在に至る．博士（工学）．主として，各種アプリケーションの並列処理の研究に従事．