Vol.2019-HPC-168 No.8
2019/3/6

都市気象LESコードの並列GPU環境における高速化

辻 大亮[1,a)]　多田野 寛人[2,3]　朴 泰祐[2,3]　池田 亮作[2,†1]　佐藤 拓人[4]　日下 博幸[2]

概要：HPC システムでは，GPU 等のアクセラレータを用いて計算性能を向上させるのが主流になっている．GPU は高いメモリバンド幅と並列計算能力を持ち，メモリバンド幅律速な高性能計算のワークロードに適したアクセラレータである．我々は，CPU アプリケーションとして開発されてきた都市気象 LES (Large Eddy Simulation) コードの GPU 化によってシミュレーションの高速化を行っている．LES における主要な計算は，ステンシル計算で構成されており，これまでに支配的な複数の関数を GPU 化することで，NVIDIA Tesla P100 GPU で Broadwell single-socket Xeon CPU に対して 5.0-10.7 倍，Tasla V100 GPU で 7.4-16.6 倍の高速化を達成した．現在，CPU-GPU 間のメモリ転送時間がボトルネックとなっているが，Runge-Kutta 法処理部を全て GPU 関数化することでオーバーヘッドを大きく削減でき，GPU を用いた LES の高速実行が期待できる．

1. はじめに

近年，HPC システムはアクセラレータを用いて性能向上を図ることが増えており，アクセラレータとして広く GPU が用いられている．GPU は高いメモリバンド幅と大量の計算コアによる高い並列計算能力を持っており，多くの HPC 向けアプリケーションとの相性が良いとされている．筑波大学計算科学研究センターでは，次世代スーパーコンピュータとして Cygnus の導入を予定しており，Cygnus はアクセラレータとして GPU だけでなく高性能 FPGA を搭載する [1]．Cygnus では GPU による高性能並列計算を実現しつつ，GPU の苦手とする演算と低レイテンシな通信を FPGA が担う計算モデルが想定されている．そのため，GPU を用いるアプリケーションと FPGA の応用が求められている．

一方で，大規模 HPC システムが求められるアプリケーションの一つに LES (Large Eddy Simulation) がある．LES は流体の数値シミュレーション手法の一つであり，高解像度な計算が可能で，気象分野では NICAM[2] のような全球計算ではなく，都市や街などの小スケールの気象シミュレーションに用いられている．筑波大学計算科学研究センターでは日下・池田らによって LES を用いた気象シミュレーションアプリ City-LES が開発されている [3]．City-LES は都市気象に特化しており，街路樹の一本一本を三次元的に考慮する樹木モデルを採用し，街区内放射計算をラジオシティ法によって高精度で計算し，街区内の熱環境計算を詳細に行えるという特色を持つ．また，City-LES は MPI + OpenMP のハイブリッド並列化が行われている Fortran で記述された CPU アプリケーションであり，スーパーコンピュータによるシミュレーションも行われているが，近年の主流である GPU クラスタを用いたさらなる高速化が望まれている．そこで本研究では City-LES を対象に GPU を用いた高速化を行い，GPU クラスタを活用可能なアプリケーションの開発を目指す．

2. 関連研究

実際に気象シミュレーションを GPU で高速化した例として，東京工業大学と気象庁が同庁の開発する気象モデルである ASUCA を対象に東京工業大学とフル GPU 化を行っている [4]．この研究ではフル GPU 化によって CPU 実行時の 10 倍以上の性能を達成している．

また，理化学研究所計算科学研究センターでは同センターが開発している SCALE (Scalable Computing for Advanced Library and Environment) に含まれている SCALE-LES の GPU 化を行っている [5]．この研究では，GPU での計算領域の分割方法やカーネルの融合やループの入れ替えなどによってデータアクセスの最適化を行い，最終的に CPU 実行の 5 倍以上の性能を達成している．

このように気象シミュレーションや気象 LES を GPU 実行することで高性能化をする例はいくつもあり，本研究が対象とする City-LES も同様に GPU によって高速化可能

[1] 筑波大学 システム情報工学研究科コンピュータサイエンス専攻
[2] 筑波大学 計算科学研究センター
[3] 筑波大学 システム情報工学研究科
[4] 筑波大学 生命環境科学研究科地球環境科学専攻
[†1] 現在，ウェザーニューズ
[a)] dtsuji@hpcs.cs.tsukuba.ac.jp

であると考えられる．

また，これまでにも二星らや高橋らによって City-LES の GPU アプリケーション化が試みられている [6], [7]．[6] にて，二星らは Fortran で記述されたコードを C/CUDA 環境へ書き換えることで GPU 化を行い，単一 GPU における実行で CPU に対して最大 8.4 倍の高速化を達成している．課題として，LES の高解像度化のために MPI 並列化によるマルチ GPU への対応や，大規模 GPU クラスタでの評価などを挙げている．

[7] では，高橋らは CUDA Fortran[9] によって部分的な GPU 化を行い，0.4-25.1 倍の高速化を達成し，時間発展部分をおよそ 2 倍高速化可能であるとした．同研究では，ポアソン方程式の求解部分などの支配的な部分の GPU 化や，性能向上が見られなかった部分の最適化を課題として挙げていた．

いずれも本研究で扱う City-LES を対象としていたが，開発が CPU 実行の Fortran プログラムで行われていたことや，日々アップデートが行われていたため，CPU アプリケーションが最新版としてメンテナンスされている．そこで，本研究では先行研究を受けて最新の City-LES を対象に CUDA Fortran を用いて開発コストの削減をしつつ GPU 化を行うこととした．また，MPI を用いたマルチ GPU 実行への対応と性能評価を行い，Pascal 世代と Volta 世代の GPU での性能評価も行う．

3. City-LES の計算モデル

LES は離散化した格子間隔以上の乱流を直接シミュレーションし，それ以下のスケールの乱流はパラメタライズして計算する数値モデルであり，その多くがステンシル計算で構成される．City-LES の概要を**表 1** と**図 1** に示す．City-LES も主にステンシル計算で構成されており，MPI による X-Y 方向の領域分割と OpenMP による Z 方向のマルチスレッド並列化によってすでに CPU 並列計算機上で高速・高解像度実行が可能になっている．したがって，GPU の持つ高いメモリバンド幅を活かしたより高性能なシミュレーションが達成可能であると考えられる．また，本研究では**表 2** に示した条件に基づいて GPU 化を行う．この条件は City-LES のシミュレーションで頻繁に利用されるもので，GPU による高性能化が達成された際の恩恵が高くなる．

4. City-LES の性能プロファイル

City-LES は多くの関数群から構成されており，その全てを GPU 化することは現実的ではない．そこで，GPU 化の恩恵の高い関数や計算を特定するために City-LES の性能プロファイリングを行った．プロファイリングには筑波大学計算科学研究センターの実験用クラスタ PPX (Pre-PACS-X) のノードを使用した．PPX ノードの構成を

表 1 対象とする City-LES の概要 [3]

基礎方程式	非静力ブジネスク近似方程式系
座標系と離散化	直交座標系，Arakawa-C，有限差分法
時間スキーム	3 段階 Runge-Kutta 法 (Wicker and Skamarock 2002)
空間スキーム	2 次，4 次，6 次精度中央差分 3 次，5 次精度風上差分
SGS モデル	TKE-1 方程式モデル (Deadoroff 1980)，Smagorinsky モデル
数値解法	SMAC 法
ポアソン方程式解法	マルチグリッド前処理付き Orthomin(m) 法
短波放射	近藤 (1994)，Dudhia simple(Dudhia 1989)，放射固定
長波放射	近藤 (1994)，RRTM(Mlawer et al. 1997)，放射固定
街区内放射	ラジオシティ法
地表面モデル	Mascart(1995)，フラックス固定
雲物理	warm rain
コード	Fortran90
並列化	MPI (X-Y 方向) + OpenMP (Z 方向)

表 2 GPU 化を行う City-LES モデルの条件

境界条件	周期境界条件
数値解法	SMAC 法 (Runge-Kutta3 回目のみ圧力補正)
時間スキーム	3 段階 Runge-Kutta 法
空間スキーム	2 次精度中央差分
SGS モデル	Smagorinsky モデル
建物	建物なし

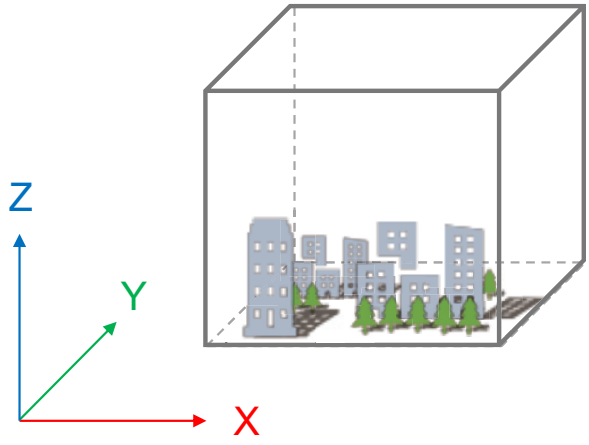

図 1 City-LES の計算領域の概要図

図 2 に，プロファイリング環境を**表 3** に示す．PPX ノードには 2 つの CPU が搭載され，それぞれに PCIe Gen3 で GPU が一台ずつ搭載されている．プロファイリングは問題領域を 2 MPI プロセスで分割し，各 MPI プロセスが 14 スレッドで動作する 2 CPU 実行で行った．City-LES の実行時間は計算関数 solve がほとんどを占めているため，その内訳を**図 3** に示す．図 3 中の MGOrthomin_m は，ポアソン

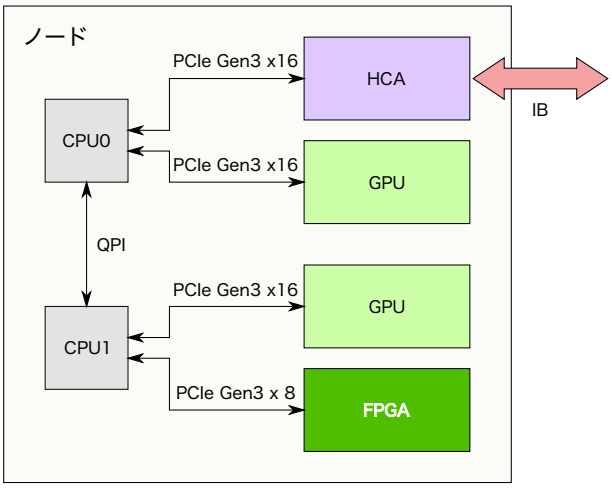

図 2　PPX のノード構成の概要

方程式をマルチグリッド前処理付きの Orthomin(m) 法で解く関数である [8]．sgs_driver や update_rk_u，advection，diffusion，update_rk_scalar は Runge-Kutta 法の求解ループで 3 回実行される関数であり，ステンシル計算で構成されているため GPU 実行に適している関数である．また，check は CFL 条件の判定を行う関数である．

本研究では基本方針として，シミュレーションの実行時間に占める割合の大きな関数を順次 GPU 化する．また，関数のデータ依存性も考慮し，CPU-GPU 間のデータ移動時間を削減できるように GPU 化を進めていく．

現在，最も割合の大きな MGOrthomin_m と sgs_driver について GPU 化を完了した．また，Runge-Kutta ループ内に存在して実行回数の多くなっている update_rk_u，advection，update_rk_scalar の GPU 化を行った．

オリジナルの CPU 版 City-LES コードは，OpenMP + MPI によってマルチスレッド・マルチノード向けに並列化されている．今回の研究では，1 プロセスのマルチスレッド CPU 版を GPU 向けに移植し高速性を確認する．このため，PPX の 1 ノードを利用し，その上で 2 プロセスによる MPI 並列化を行う．GPU を制御するホストプロセスはそれぞれの GPU がバインドされている CPU 上で 1 つずつ起動する．よって，2 プロセス・2GPU での実行であるが MPI は単一ノード上で InfiniBand を用いずに実行している．

5. GPU 実装

本節では GPU 化の手法について述べる．GPU は NVIDIA 社製 GPU を対象とし，CUDA Fortran での開発を行う．また，MPI 処理系には MVAPICH2-GDR[10] を用いた．

5.1 スレッドの構成方法

City-LES の実行時間で多くを占めるのがステンシル計算である．本節ではステンシル計算を GPU で実行する際の方針について述べる．メモリバンド幅律速のステンシル計算を GPU で高速に処理するためには，GPU のメモリ特性に合わせてスレッドを起動する必要がある．

本研究では CUDA Fortran を用いているため，配列要素はメモリ上に列優先で配置される．すなわち City-LES で A (x,y,z) で定義される 3 次元配列に対し，スレッドの構成方法を指定する構造体である dim3(x,y,z) の次元をそのまま対応させることで，連続するスレッドが連続する配列要素にアクセスできる．また，スレッドブロックあたりのスレッド数はワープの倍数である 128 スレッドを最大とし，このスレッドをメモリの連続する方向である dim3 の X 次元に展開することでコアレスアクセスを実現した．さらに

表 3　PPX ノード構成と City-LES 性能プロファイリング環境

CPU	Intel Xeon E5-2690 v4(14 cores) x2
GPU	NVIDIA Tesla P100 (PCIe card version) x2
ホスト OS	CentOS 7.3
コンパイラ	PGI Compiler 17.10
MPI	MVAPICH2-GDR2.3a
CUDA バージョン	9.0.176
問題サイズ	512x256x128 (256x256x128 x2 プロセス (14 スレッド))
時間発展数	200 ステップ

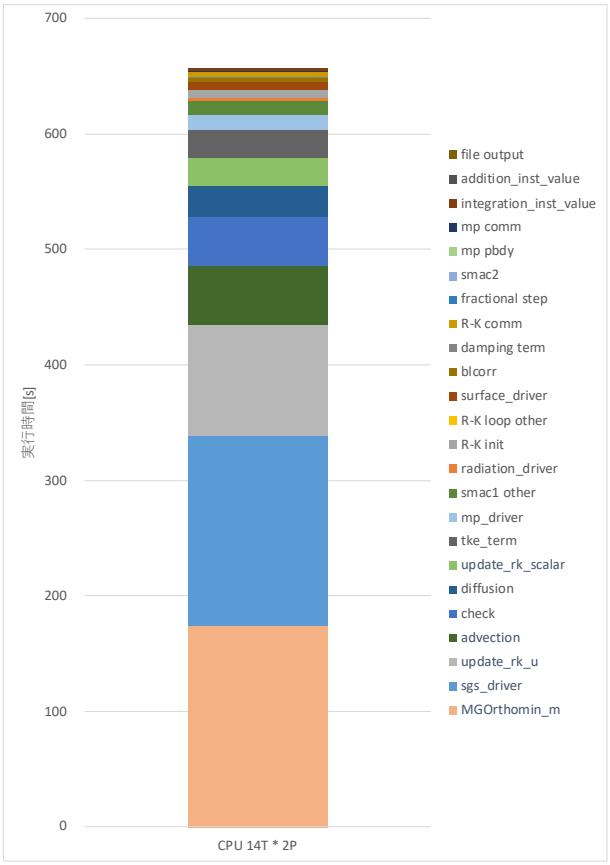

図 3　City-LES solve 関数の性能プロファイル結果

GPU の並列計算能力を活かすために，グリッドの Y 次元にスレッドブロックを展開し，その並列度を City-LES の Y 次元の解像度に設定した．以上の方針を用いて，GPU のメモリバンド幅と高い並列計算性を活かした計算を行う．

5.2 MPI 袖通信のパッキング

City-LES では，MPI を用いた X-Y 方向の 2 次元領域分割を採用し，近傍との袖通信には MPI_Type_vector を用いた派生データ型を用いて行う．本研究で使用する MVAPICH2-GDR ではそのような場合，派生データ型の構成要素である袖領域を自動的なパック・アンパックを通して袖通信を行う．一方でパック・アンパックを CUDA カーネルで明示的に実行して袖通信を実現することも考えられる．そこで MPI ライブラリによる自動パッキングと明示的な手動パッキングのノード内 GPU 間バンド幅を CPU 間バンド幅と共に比較した．

この結果を図 4 に示す．縦軸はメモリバンド幅を表し，横軸は問題サイズと袖通信のデータ量を表す．この結果は 3 次元配列でデータが比較的に不連続になる Y-Z 平面の袖通信において，パッキング時間を含めた ping-pong バンド幅である．また，図中の pack X はスレッドブロックの X 次元にスレッドを配置してグリッドの Y-Z 次元にスレッドブロックを配置する構成方法でパッキングを行った場合であり，pack Y ではスレッドブロックの Y 次元にスレッドを配置してグリッドの X-Z 方向にスレッドブロックを配置する構成方法でパッキングを行った場合である．パック・アンパックを行うカーネルは同一のものとして，pack X ではデータが X 次元に連続している場合 (X-Z 平面) に，pack Y では Y 次元に連続している場合 (Y-Z 平面) にパッキングが高速になることを意図している．

袖通信をライブラリに任せた場合，問題サイズが 128 の 3 乗までではCPU 間通信の性能が高く，それより大きくなると GPU 間のほうがより高速な通信を行えるが，大きな差はない．一方で，CPU 間通信では手動パッキングによりバンド幅が大きく向上していることがわかる．さらに，GPU 間通信に関しては pack Y は問題サイズが大きくなるにつれてバンド幅が向上し，128 の 3 乗以上ではライブラリの自動パッキングよりも性能が良く，256 の 3 乗以上では CPU 間手動パッキングよりも高いメモリバンド幅が得られることがわかった．これは GPU の高いメモリバンド幅がパッキングの作業に向いていることや，手動パッキングによって問題領域に合わせた高速なパッキングが可能になったことが理由と考えられる．

そこで本研究では GPU 実装時における袖通信では手動のパッキングカーネルを使用し，通信コストを削減する．

5.3 MGOrthomin_m の GPU 化

City-LES は，ポアソン方程式の求解に V-Cycle によるマルチグリッド前処理を適用した Orthomin (m) 法 (MGOrthomin (m) 法) を用いる [8]．V-cycle を採用するマルチグリッド前処理では，各 Orthomin(m) 法の反復で問題サイズを数回粗くして，粗い問題で大まかに解いた途中解を用いてより細かい問題サイズの途中解を求める処理を繰り返す．したがって，マルチグリッド前処理では小さな問題サイズを計算する必要がある．この場合，GPU が得意とする大量のデータに対する計算を満たせない上に，袖通信の遅さが問題となる．そこで，本研究では計算は GPU 上で行うものの，袖通信に手動パッキングを使用することでマルチグリッド前処理の性能改善を図った．

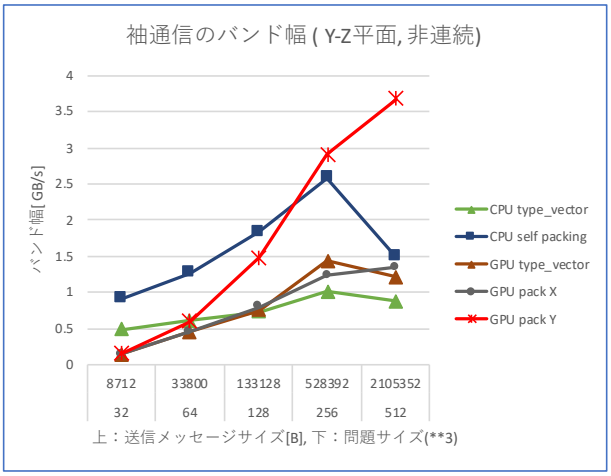

図 4 PPX ノード内における袖領域通信のバンド幅

また，Orthomin(m) 法では収束判定時などに 2 ノルム計算が必要になる．通常ノルム計算はリダクションが必要になる上に，領域分割している全ての空間のデータを用いて算出するため通信を伴った計算が行われる．そこで，今回は各プロセスが 1 つずつ GPU を使用することを想定し，各プロセスごとに GPU 上で 2 乗和を計算したものを，MPI_Allreduce によって CPU メモリ上に集めて CPU 上で 2 ノルムを求めるようにした．これにより，少量のデータの計算や少量のデータ転送が不得意となる GPU の性質を補うことができる．また，CPU 上にノルムの結果が残るのでその後の Orthomin(m) 法の反復の継続の判定も容易になる利点がある．

なお，[8] とは異なり，マルチグリッド法の平滑化には並列化可能な処理であるという観点から，重み付きヤコビ法による反復を採用している．

5.4 sgs_driver の GPU 化

sgs_driver では SGS (Subgrid Scale) モデルを計算する処理を行う，ステンシル計算によって構成される関数群を呼び出す関数である．したがって，5.1 節で述べたように，連続するメモリ領域を連続するスレッドが参照するようにループを書き換えたカーネルを作成した．また，sgs_driver では 1 次元の配列要素に並ぶインデックスによって 3 次元

データを間接参照するコードが一部含まれている．これらの間接参照に用いられる配列要素は City-LES の問題設定に依存して決定されるため，実行時に決定される．そのため，今回はインデックスとなる配列要素にコアレスアクセスできるように，5.1 節と同様の手法で構成したスレッドを 1 次元に展開して ID を割り振ることで対応を行った．

5.5 advection の GPU 化

advection は移流項を計算する関数群である．City-LES は様々な精度での計算を選択する機能を持つが，今回は空間スキームに 2 次精度中央差分を選択しているため，2 次精度の関数のみを GPU 化した．2 次精度の関数は 7 点ステンシル計算で構成されるため，これまでと同様にカーネルを作成して GPU 化を行った．

5.6 update_rk_u と update_rk_scalar の GPU 化

update_rk_u と update_rk_scalar は Runge-Kutta 法の各ステップにおける流速等の値を求めて更新する関数である．そのため，方程式上の多くの配列データを参照するメモリバンド幅律速な計算となっている．したがって，同様の手法でスレッドをマッピングすることでコアレスアクセスをしながらの計算が可能となる．

6. 性能評価

今回 GPU 化を行った各関数について，CPU 実行時と GPU 実行時の実行時間を用いて性能評価を行った．評価は表 3 に示した PPX ノード上で，2 つのプロセスを用いた 2CPU 実行時と 2GPU 実行時の性能の比較とする．評価方法の概要を**表 4** に示す．MPI プロセス数は 2 とし，各プロセスが 256 × 256 × 128 の問題を解くように問題空間と領域分割の設定を行った．CPU 実行時には計算機としての CPU の性能と比較するため，各プロセスが 14 スレッドで動作し，全てのコアを使い切るように実行を行う．

PPX ノードには GPU を NVIDIA Tesla V100 に取り替えたものが存在するため，性能評価では P100 および V100 による GPU 間の性能比較も行う．

6.1 MGOrthomin_m の評価

MGOrthomin_m の測定結果を**図 5** に示す．Orthomin_m 法では問題によって反復回数が異なるため，関数のループ外とループ内の収束判定式前後の 3 箇所を測定した．図 5 における判定式前が first half を，判定式後が latter half にあたる．GPU 実行によりループ外では P100 で 5.0 倍，V100 で 7.4 倍の高速化が達成できた．また，ループ内では判定式前がそれぞれ 6.7 倍と 11.7 倍で，判定式後がそれぞれ 8.0 倍と 11.4 倍であった．それぞれ実行時間の比と Orthomin_m 法の反復から，P100 実行で 5 倍から 8 倍の高速化が可能であると考えられる．

表 4 性能評価環境

MPI プロセス数	2 プロセス
問題サイズ	(2 × 256) × 256 × 128
CPU スレッド数	14 スレッド / プロセス
コンパイルオプション	-O3 -Mcuda=cc60,cc70 -mcmodel=medium -mp -Mextend

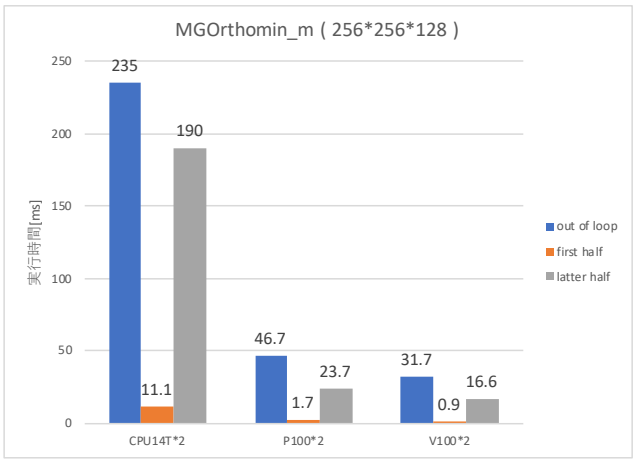

図 5 MGOrthomin_m の測定結果

6.2 その他の関数の評価

他に GPU 化を行った関数の実行時間と高速化率を**表 5** に示す．sgs_driver では GPU 化によって P100 で 10.6 倍，V100 で 16.6 倍の高速化が得られた．また，update_rk_scalar でも P100 では 10.7 倍で V100 では 16.0 倍と，同程度の高速化を達成している．一方，update_rk_scalar は P100 と V100 ともに sgs_driver ほどの高速化が達成できているが，update_rk_u については P100 で 7.5 倍，V100 で 9.0 倍と，他の関数よりも小さな値となった．現在までに原因は把握できていないが，update_rk_u は他の関数に比べて計算に要する変数が多く，GPU のスレッドあたりのレジスタ数などリソースの限界に達しているのではないかと考えている．

advection については P100 で 5.7 倍，V100 で 9.3 倍と高速化は達成できたが，P100 のみにおいて他よりも高速化率は小さいという結果が得られた．これは他の関数は多くの配列変数から計算が構成されており，advection に対して GPU に適していると考えられることと，GPU が Pascal アーキテクチャから Volta アーキテクチャに変更されたことで起きたなんらかの改善が作用して V100 で高速に計算できるようになったのだと考えられる．例えば GPU が非常に得意とする計算以外も含まれる MGOrthomin_m では P100 の高速化率は 5.0 から 8.0 倍であった．

このように，これまで GPU 化を実施した関数について V100 で 9.0 倍から 16.6 倍の高速化を達成でき，City-LES の計算が GPU 実行に適していることがわかった．

表 5　GPU 化した関数の実行時間 [ms] と高速化率

関数	CPUx2	P100x2(高速化率)	V100x2(高速化率)
sgs_driver	272.5	25.8 (10.6)	16.4 (16.6)
advection	83.8	14.7 (5.7)	9.0 (9.3)
update_rk_u	79.2	10.5 (7.5)	8.8 (9.0)
update_rk_scalar	40.3	3.8 (10.7)	2.5 (16.0)

6.3　フル GPU 版 City-LES 実行性能の予測

City-LES は複雑乱流計算を様々な条件で計算するモデルのため，アプリケーションの規模が大きく，現時点では未だ完全な GPU 化は行えていない．本節にてフル GPU 実装における，City-LES の期待実行性能について予測を行う．

これまでに GPU 化した関数を用いて City-LES を実行した際の計算関数の実行時間の測定結果を**図 6** に示す．図 6 は計算関数の実行時間を，CPU 計算時間，GPU 計算時間，MPI 通信時間，CPU-GPU 間メモリ転送時間に分けたもので，CPU -> GPU は CPU から GPU へのメモリ転送時間を，CPU <- GPU は GPU から CPU へのメモリ転送時間を表す．現在のところ，計算データの主に CPU メモリに確保されていることを想定し，GPU カーネルを実行するたびに必要なデータを GPU に転送し，カーネルの終了後に計算結果を CPU に戻す実装をとっている．

これまで GPU 化を行った関数を City-LES に実装することで実行時間が P100 で 1.15 倍，V100 で 1.12 倍になっているが，これは想定の範囲内である．上述の通り，データは毎回 CPU メモリから GPU に読み出し，実行結果を CPU に書き戻しているため，CPU-GPU 間のデータの移動によるオーバーヘッドが非常に大きなボトルネックとなってしまっている．例えば，sgs_driver を GPU で実行する際には 512MB ほど GPU へデータを転送する必要があり，実行完了後には計算結果を用いて GPU 化していない関数で計算を行うため，CPU へ 1GB ほどのデータを戻す必要が生じている．このようなデータ依存によるメモリ転送が Runge-Kutta 法ループ内でも繰り返し行われて非常に大きなオーバヘッドとなっており，GPU 実行時の CPU-GPU 間メモリ転送時間は P100 で 65.1%，V100 で 67.2%を占め，非常に大きなコストとなっている．

一方で，フル GPU 化を行った場合，多くのデータが GPU 上から移動せずとも良くなり，データのメイン位置を GPU 上にすることができる．その場合，CPU-GPU 間メモリ転送も必要なくなり，オーバーヘッドとなっているメモリ転送時間をほとんど考慮せずに済む．そこで，フル GPU 化を完了し，未実装の関数も CPU と同程度の速度で GPU 実行できると仮定した場合の City-LES の実行性能を，メモリ転送時間を無視することで予測する．CPU-GPU 間のメモリ転送時間を無視した場合の GPU での予想される実行時間は P100 で 262 [s]，V100 で 241 [s] と，この場合は CPU に対してそれぞれ 2.52 倍と 2.74 倍の高速化が期待できる．

7.　まとめ

本研究では，筑波大学計算科学研究センターで開発されている都市気象 LES に対し GPU を用いた高速化を行い，GPU クラスタを活用可能なアプリケーションの開発を目的とした．代表的に用いられる条件で性能プロファイリングを行い，計算コストが重い関数郡を順次 GPU で実装し，CPU 性能に対して P100 で 5.0-10.7 倍，V100 で 7.4-16.6 倍の高速化をすることができた．また，GPU 化関数を用いて LES を実行した場合，現段階では実行時間が 1.14 倍に伸びて遅くなるものの，フル GPU 化したアプリケーションで CPU 実行に対して 2.5 倍以上の性能を発揮できることが期待される

今後の課題として，CPU-GPU 間のメモリ転送時間を削減するために GPU 実装が間に合わなかった他の関数群の GPU 化を行わなければならない．Runge-Kutta 法ループ内の関数を全て GPU で計算することで，Runge-Kutta 法の処理の前後のデータ移動を，およそ，512MB と 640MB に削減することができ，CPU-GPU 間のメモリ転送時間のコストを大きく削減可能と考えられる．

また，今回はシングルノードでの性能評価しか行えていない．フル GPU 化が完了した後，筑波大学計算科学研究センターで稼働予定の Cygnus (V100 GPU+FPGA クラスタ) といった大規模 GPU クラスタにおけるマルチノード・GPU での性能評価も必要である．さらに，プロダクションラン相当の問題を解いた場合の性能評価も行いたいと考えている．

謝辞　本研究は，文部科学省「次世代領域研究開発」(高性能汎用計算機高度利用事業費補助金) 次世代演算通信融合型スーパーコンピュータの開発の一環として実施したものである．また，本稿の執筆に当たりご協力頂いた筑波大学計算科学研究センター廣川祐太氏に深く感謝する．

参考文献

[1] 計算科学研究センター:スーパーコンピュータ - 筑波大学 計算科学研究センター Center for Computational Sciences 入手先 ⟨https://www.ccs.tsukuba.ac.jp/supercomputer/#Cygnus⟩ (2019.01.21)
[2] NICAM : NICAM:非静力学正 20 面体格子大気モデル入手先 ⟨http://cesdweb.aori.u-tokyo.ac.jp/ñicam/index.html⟩ (2019.01.30)
[3] Ikeda, R., H. Kusaka, S. Iizuka, and T. Boku.: Development of Urban Meteorological LES model for thermal environment at city scale. 9th International Conference for Urban Climate, Toulouse, France, (July, 2015).
[4] 下川辺隆史, 青木尊之, 石田純一, 河野耕平, 室井ちあし：メソスケール気象モデル ASUCA の TSUBAME 2.0 での実行. 日本流体力学会第 24 回シンポジウム講演予稿集, (December,2010).

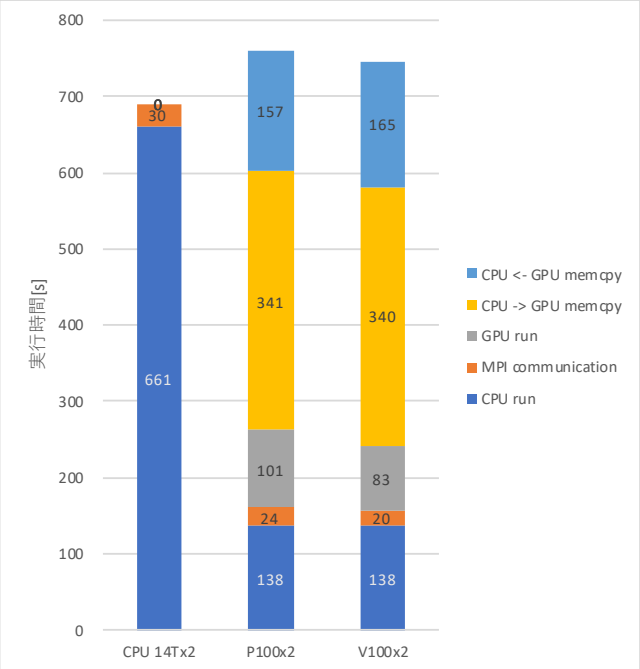

図 6　City-LES の計算関数 solve の実行時間

[5] Mohamed Wahib, Naoya Maruyama: Highly optimized full GPU-acceleration of non-hydrostatic weather model SCALE-LES. IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, USA, (September, 2013).
[6] 二星義裕, 朴泰祐, 池田亮作, 日下博幸 : 高解像度 LES 計算の GPU による計算加速. 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 (January, 2012).
[7] 高橋一成, 朴泰祐 : LES による都市気象モデルの GPU クラスタ上での高速化. 筑波大学情報学群情報科学類卒業研究論文 (February, 2013).
[8] Tadano H., R. Ikeda, H. Kusaka: Speeding up Large Eddy Simulation by Multigrid preconditioned Krylov subspace methods with mixed precision. The 35th JSST Annual Conference International Conference on Simulation Technology (JSST2016), Kyoto, Japan, (October, 2016).
[9] NVIDIA CORPORATION: PGI — Resources — CUDA Fortran 入手先 ⟨https://www.pgroup.com/resources/cudafortran.htm⟩ (2019.01.24)
[10] NBCL.: MVAPICH :: GDR Userguide 入手先 ⟨http://mvapich.cse.ohio-state.edu/userguide/gdr/⟩ (2019.01.24)