Vol.2019-HPC-170 No.39
2019/7/26

都市気象コードCity-LESの並列GPU実装の最適化と性能評価

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：今日の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一つの主流になっている．GPUは高いメモリバンド幅と並列計算能力を持ち，定型的な計算を行うメモリバンド幅律速なHPC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コードCity-LES (Large Eddy Simulation) 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．元コードのLESにおける主要な計算は，3次元ステンシル計算をOpenMP+MPIで並列化しているが，支配的な複数の関数をGPU化することで，GPU実行に適していると思われる関数群では，関数単体実行で2GPU実行では2CPU（28コア）実行時の5.7倍から12.4倍に高速化でき，4GPU実行では4CPU（56コア）実行時の5.5倍から12.5倍の性能を達成することができた．また，全体実行では2GPU実行で2CPU実行に対して2.97倍，4GPU実行では4CPU実行の2.78倍の高速化を達成し，さらにGPU化を進めてフルGPU化を達成できれば2GPUで2CPUに対して3.8倍，4GPUで4CPUに対して3.5倍高速化できると予測している．

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2019年4月より，アクセラレータとしてGPUだけでなく高性能FPGAを搭載するスーパーコンピュータCygnusを導入している[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OpenMP+MPI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．まず本アプリケーションを徹底的にGPU化し，その上でGPUに向かず計算または通信律速となる部分について最終的にFPGA化を行うが，本稿ではGPU化部分について述べる．

これまで，City-LESの主要な計算部分のうち最も割合の大きかった関数を含む5つの関数群をGPU化によって高速化し，City-LESに組み込んで性能評価を行ってきた[4]．しかし，高速化したGPUカーネルを呼び出すためのCPU-GPU間メモリ転送時間が大きなオーバヘッドとなり，City-LES実行の高速化には至っていなかった．そこで本研究ではCity-LES関数の高速化や，CPU-GPU間メモリ転送時間を削減させるためにさらに4つの関数群を

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

ⓒ 2019 Information Processing Society of Japan

GPU 化し，City-LES に組み込んだ上で評価を行った．

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 [5]．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．しかし，ASUCA そのものが非公開コードであるため，具体的な手法や計算律速部分等の詳細は明らかでない．

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 [6]．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 [7], [8]．[7] 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．

[8] では，高橋らは CUDA Fortran[10] 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，最新コードの GPU 化が望まれている．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．

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

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**図 2**

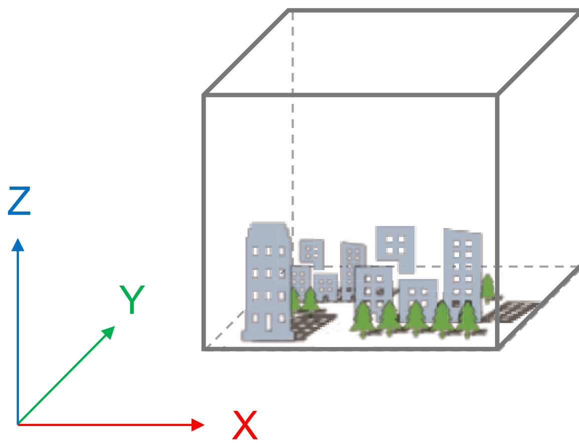

図 1　City-LES の計算領域の概要図

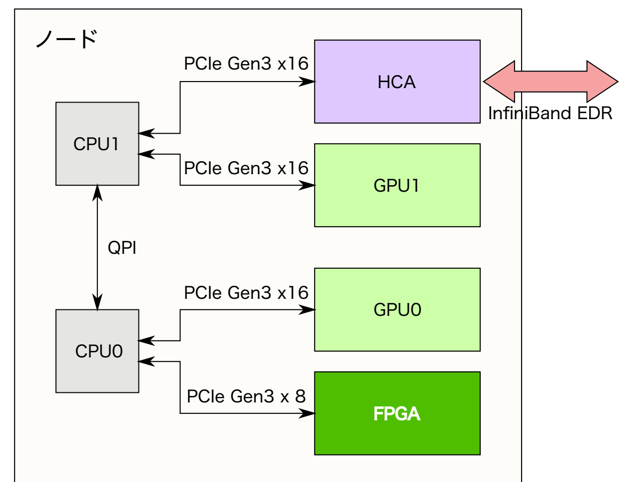

図 2　PPX のノード構成の概要

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 [9]．図中赤字で示された関数群は 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．

図中 sgs_driver から RK comm までの Runge-Kutta 法ループ部分は，solve 関数の実行時間のうちおよそ 57%を占めており，また，MGOrthomin_m 関数は solve 関数の 28%を占めている．

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．

1 節で述べたように，これまでに最も割合の大きな MGOrthomin_m と sgs_driver についてや advection，update_rk_u，update_rk_scalar の 5 つの関数群の GPU 化を完了していた．本研究では新たに surface_driver，tke_term，bl_corr，diffusion といった Runge-Kutta ループ内に存在する 4 つの関数群の GPU 化を行い，solve 関数の 85%に及ぶ部分の GPU 化を完了した．

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．

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[11] を用いた．

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 E5-2660 v4(14 cores) x2
GPU	NVIDIA Tesla P100 (PCIe card version) x2
ネットワーク	infiniBand EDR100
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

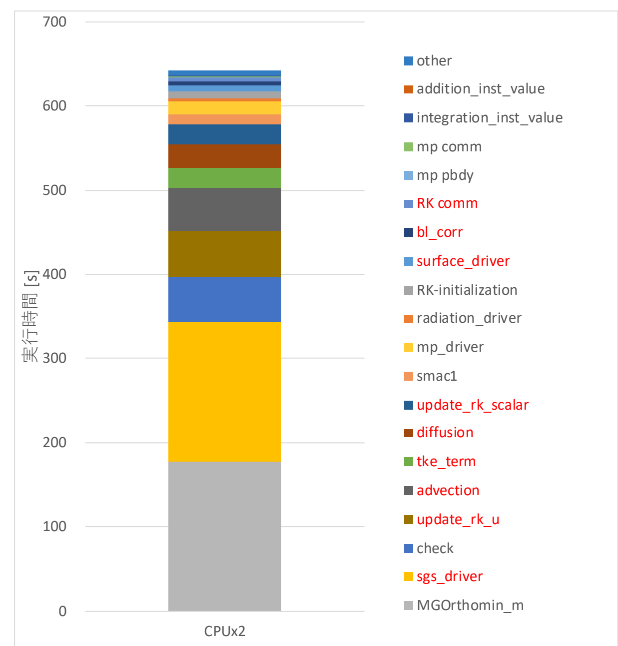

図 3　City-LES solve 関数の性能プロファイル結果

ⓒ 2019 Information Processing Society of Japan

5.1 MPI プロセス数と GPU 数

現在の超高性能 GPU クラスタでは 1 ノードに複数の GPU を搭載することが一般的で，本研究で実験システムとして用いる PPX も，最終的な実装対象である Cygnus も，それぞれ 2 台及び 4 台の GPU を搭載している．CUDA では 1 つの MPI プロセスから複数の GPU にオフローディングすることも可能であるが，プロセス内のスレッド数（非 GPU 化関数は引き続き CPU 内で OpenMP マルチスレッド処理を行う）と GPU 数の関係が複雑になるため，本研究の実装では MPI プロセスは複数の CPU コアをマルチスレッドで利用し，GPU については 1 台のみを対象とする．従って，N 台の GPU を搭載したノードでは N 個の MPI プロセスを立ち上げ，CPU の総コア数を N で除した数を各 MPI プロセスの OpenMP スレッド数とする．従って，複数 GPU を搭載したノードではノード内でも MPI 通信を行う必要がある．一見すると通信オーバヘッドが増えるように見えるが，実際の CUDA におけるマルチ GPU 対応は GPU 間データ移動がそれらを結合するハードウェア（PCI Express なのか NVLINK なのか）に依存するため，性能移植性の高い実装を行うのが難しい．

さらに，Summit[12]，Sierra[13]，TSUBAME3.0[14]，ABCI[15]，Cygnus 等の複数 GPU 搭載ノードを持つクラスタでは並列通信のための相互結合網（InfiniBand や Omni Path Architecture）のホストインタフェースを複数用い，超高速な演算性能を支える高バンド幅な通信網を備えている．これらの複数のホストインタフェースを効率的に用いるためには，それらをバインドして 1 つの MPI 通信で用いるより，ほぼ同期された状態で複数の MPI プロセスがそれぞれ独立に通信チャネルとして分散して利用する方が効率が良いと考えられる．

以上の理由から，本研究での実装では相互結合網を構築するホストインタフェース数，GPU 台数を勘案し，ノード内 MPI プロセス数を決定する．具体的には，実験環境である PPX では GPU 数が 2 台，ホストインタフェース数は 1 台であることから，ノード内 MPI プロセス数は 2 とする．今後の Cygnus での実装は 4MPI プロセス/ノードとなる予定である（Cygnus のノード当たり GPU 数とホストインタフェース数は共に 4 であるため）．

5.2 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 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．

5.3 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にパック／アンパックして袖通信を行う．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．縦軸は袖通信にかかる時間をログスケールで表し，横軸は問題サイズを表す．問題サイズは City-LES の性能評価時に使用したものについて，マルチグリッド法実行時に実際に発生するサイズも含めて測定を行っている．また、袖通信は 2 プロセスか 4 プロセスで行い，周期境界条件を想定して全て InfiniBand を経由するマルチノードで実行した．図中 2 プロセス通信において，near P100x2 は HCA に近い GPU 同士の袖通信を示し，far P100x2 は HCA から QPI を跨いだ GPU 同士の袖通信を示している．Intel Xeon CPU では，ソケット間の QPI パスを跨いだ PCIe デバイス間通信が可能であるが，同じソケットの PCIe デバイス間に比べ通信バンド幅が低下する可能性があるため，このように 2 種類の組み合わせを評価する．

袖通信ではまず非連続な Y-Z 平面から交換し，それが完了したのちに比較的連続な X-Z 平面の交換を行う．今回実装した手動パッキングでは，それぞれの袖部分のパッキングにおいてスレッドの構成方法のみを変更し，全く同じカーネルを用いてパック／アンパックを行なっている．

2 プロセスでの袖通信を Type_vector と手動パッキングで比較した場合，near P100x2 では手動パッキングの方が 2 倍以上高速に通信できている．これは，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．一方で far P100x2 では，32x32x128 では手動パッキングが 2 倍以上高速であるが，それ以上の問題サイズでは手動パッキングが高速である

ⓒ 2019 Information Processing Society of Japan

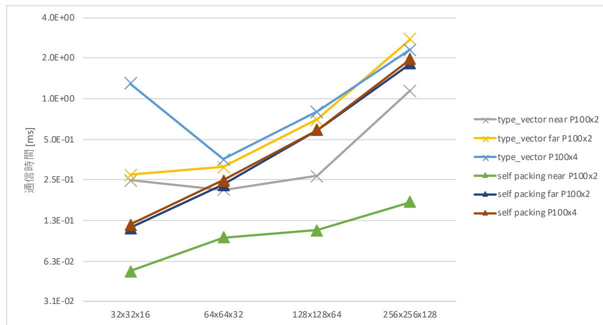

図 4 PPX ノード間における袖領域通信レイテンシ

が，1.2 倍ほどとやや低下している．これは，パッキングにかかる時間よりも通信にかかる時間が多くなっていることや，HCA カードから離れた GPU メモリをノード間に転送させるためのオーバヘッドが大きくなっているためだと考えられる．

4 プロセスの場合では，32x32x128 では Type_vector が非常に遅く，それ以外の問題サイズでも手動パッキングが 1.2 倍ほど高速になっている．32x32x128 において Type_vector が非常に低速になる原因は調査中であるが，MVAPICH2 ライブラリが小さい Type_vector データサイズの転送時に使用する通信手法が 4 プロセス同時に HCA を使用したことで何らかの悪影響を及ぼしたと考えられる．また，手動パッキング時では通信にかかる時間が手動パッキングの far P100x2 とほとんど同じになっていることが確認できる．これは 4 プロセスでの袖通信において通信の高速な near P100 同士の通信に対し，far P100 同士あるいは near と far 相互の袖交換にかかる時間が支配的になっていると考えられる．

また，Type_Vector と手動パッキングを比較すると，全ての問題サイズにおいて手動パッキングの方が高速であり，小さい問題サイズほど比率としての差が大きくなっていることが確認できる．以上より，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．

5.4 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 [9]．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．

また，Orthomin(m) 法では収束判定時などに 2-ノルムの計算が必要になる．通常ノルム計算はベクトルの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，[9] 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．

5.5 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，5.2 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，5.2 節と同様の手法で構成したスレッドを 1 次元に展開してインデックスを割り振ることで対応を行った．

5.6 その他の関数の GPU 化

図 3 に示した関数のうち，Runge-Kutta 法処理部分で呼び出される関数群の GPU 化を行なった．その多くがステンシル計算で構成される関数であり，これまでと同様にカーネルを作成して GPU 化を行った．

一方で，bl_corr は地表面を対象とした関数で計算領域が X-Y 方向の 2 次元分しかないためデータの並列度が低く GPU 化には適していない．また，surface_driver も地表の建物等を考慮する関数群であり問題空間に対する大きさからみてデータの並列度が低く GPU 化には適していない．しかしこれらの関数は Runge-Kutta 法処理部分で呼び出され，CPU-GPU 間のデータ移動の頻度を減らすためにも GPU 化の対象とした．並列化の手法においては地表面に対してはこれまでと同様の方法で並列にスレッドを起動した．また，建物等に関しては建物の存在する格子数を対象に並列化し，並列度は X 次元の格子数とした．

ⓒ 2019 Information Processing Society of Japan

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 ノードを使用して，2 つのプロセスを用いた 2CPU 実行時と 2GPU 実行時と，4 つのプロセスを用いた 4CPU 実行時と 4GPU 実行時を比較する．評価方法の概要を**表 4** に示す．MPI プロセス数は 2 または 4 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．すなわち，並列処理性能は weak scaling を対象とする．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．

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 second half にあたる．P100 実行によってループ外では 2 プロセス実行で 4.8 倍，4 プロセス実行で 4.3 倍の高速化ができた．また，ループ内では判定式前がそれぞれ 6.8 倍と 6.9 倍で，判定式後がそれぞれ 7.1 倍と 6.5 倍であった．それぞれの実行時間の比と Orthomin_m 法の実行における反復回数から，P100 実行で 4.3 倍から 6.5 倍ほどの高速化が達成可能であると考えられる．

一方で，2 プロセス実行と 4 プロセス実行ではどの区間においても実行時間が 4 プロセス実行でわずかに増加している．これはプロセス数が増えたことや PPX ノードのホストインタフェースが 2CPU あるいは 2GPU に対して 1 つしかないことで通信時間が増加しているためであると考えられる．しかし，ループ内前半部に関しては 2 プロセス実行よりも 4 プロセス実行の方が P100 実行時の性能向上がわずかに大きくなっている．これは単純に実行時間から考えると，CPU が 2 プロセスから 4 プロセス実行に移った際の性能が 0.98 倍になったのに対して，P100 が 2 プロセス実行から 4 プロセス実行に移った際の性能が 0.99 倍であったことから求まる値である．また，前半部にはマルチグリッド処理は含まれておらず，袖通信もないため P100 実行時でも性能低下がほとんどなく，実行される MPI_Allreduce では計算結果を CPU 上に集めているためこちらも性能に影響しなかったと考えられる．

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．表の上部分が GPU 化に適すると考えられた関数群であり，下部分が適さないと考えられた関数群である．

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

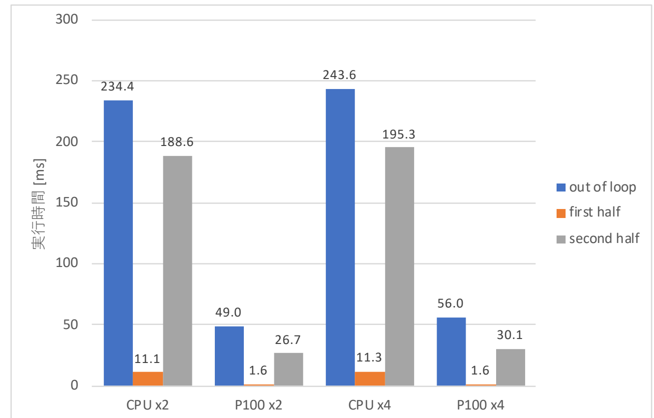

図 5 MGOrthomin_m の測定結果

GPU 化に適すると考えられた関数のうち，sgs_driver では P100 の 2 プロセス実行で 9.4 倍，P100 の 4 プロセス実行で 9.1 倍の高速化が得られた．また，diffusion と update_rk_u ではどちらもそれぞれ 10.8 倍となり，同等の高速化が得られている．update_rk_u は 9.0 倍と 8.2 倍であり，並列度を上げた時の高速化率の伸びが小さくなっているが，これは関数内に MPI_Allreduce を含み，その値を GPU 上で使用しているためであると考えられる．もっとも高速化された関数は tke_term である．この関数は表中の他の関数と比べて単一のステンシル計算から構成されており，通信を含まず，さらに他の関数よりも多くの変数を参照した計算を行うため GPU の高いバンド幅と演算性能を活かせたことでより高速化できたと考えられる．一方でもっとも高速化が小さかった advection 関数では，それぞれ 5.5 倍と 5.7 倍の高速化が得られた．この関数はステンシル計算を行う複数のカーネルからなっており，通信も行わないが，他の関数と比べて演算数に比べて参照する配列の数が少なかったことで，むしろ他の関数が GPU 化に適していたのだと考えられる．

一方で GPU 化に向いていないと思われた関数では，surface_driver では 0.55 倍と 0.72 倍，bl_corr では 0.43 倍と 0.25 倍という結果になった．どちらもほぼ 2 次元空間上の計算となりデータの並列度が小さく GPU 化に適していないため，ほとんど高速化できなかった．しかし，これらの関数の実行時間は CPU 実行 City-LES においてもそれぞれ 1 %に満たないほどであり，これらを GPU 実装することで削減できる CPU-GPU 間メモリコピー時間を考慮すると GPU 化する価値があると考えられる性能である．

ⓒ 2019 Information Processing Society of Japan

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)	CPUx4	P100x4(高速化率)
sgs_driver	274.4	29.1 (9.4)	277.68	30.6 (9.1)
tke_term	40.0	3.2 (12.4)	40.0	3.2 (12.5)
diffusion	45.7	4.2 (10.8)	45.9	4.2 (10.8)
advection	84.4	14.9 (5.7)	84.8	15.3 (5.5)
update_rk_u	88.1	9.8 (9.0)	88.0	10.8 (8.2)
update_rk_scalar	40.3	3.7 (10.8)	40.5	3.8 (10.8)
surface_driver	8.3	15.0 (0.55)	11.6	16.0 (0.72)
bl_corr	0.39	0.9 (0.43)	0.41	1.61 (0.25)

6.3　City-LES 全体実行時の性能評価

これまで GPU 化した関数群を City-LES に移植し，City-LES の CPU 実行時と GPU 実行時の性能評価を行う．性能評価はこれまでと同様に，2 プロセスにおける 2CPU 実行あるいは 2GPU 実行と，4 プロセスにおける 4CPU 実行あるいは 4GPU 実行の実行時間を比較して行う．

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．また，通信時間は全体の実行時間から，CPU 計算時間と GPU 計算時間と CPU-GPU 間メモリ転送時間の和を引いたものとした．これは，通信にはステンシル通信やリダクション通信が含まれ，測定するプロセスによって同期待ちの時間が含まれバラつきが生じるためである．

これまでに GPU 化した Runge-Kutta 法部分の関数を City-LES に実装することで，2 プロセス実行では P100 で CPU の 2.97 倍，4 プロセス実行では P100 で CPU の 2.78 倍の性能を達成することができた．これは City-LES の支配的な部分である Runge-Kutta 法部分やマルチグリッド前処理付きポアソン方程式ソルバ関数を GPU 化したことや，[4] において非常に大きなオーバヘッドとなっていた CPU-GPU 間のデータ転送時間を連続した関数群を GPU 上に実装できたことで抑えられたことが理由である．また，[4] で予測した 2 プロセスフル GPU 化実行時の性能は P100 で 2.52 倍であったが，今回の測定では 2.97 倍とそれ以上の結果が得られている．実行している CPU の性能に差異はあるものの，前回の予測で未 GPU 化 CPU 関数を GPU 実装した場合の性能を CPU と同等と仮定していたのに対し，実際に GPU 化した関数が GPU 実行に適していてより短い時間で実行できるようになったことが理由であると考えられる．

MPI 通信時間に関しては 2 プロセスから 4 プロセスに増やしたことで増加しており，GPU 実行時では 12 秒から 26 秒と 2 倍以上の増加している．これは図 4 で示したように 4 プロセスでの袖通信が 2 プロセスと比較してかなり遅いことが影響しており，その原因も PPX ノード上のホスト

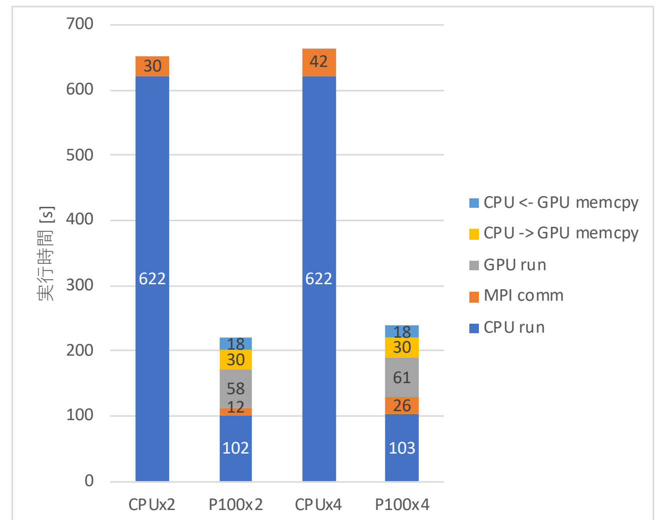

図 6　City-LES の計算関数 solve の実行時間

インタフェースが 2GPU に対して 1 台しかなく，ノードを跨ぐ GPU 同士では GPU メモリ上で RDMA できないことが影響していると考えられる．しかし，現在のところでは MPI 通信時間は全体の 11%ほどであり，袖通信を多用する City-LES においてはさらに深刻な影響はないと考えられる．

また，残りの CPU 実行時間は 40%ほどで，CPU-GPU 間データ転送時間は 21%ほどを占めている．これらの値は City-LES の計算上関連しており，全て GPU 化できればデータ転送時間は限りなく 0 にすることができる．残りの計算関数を全て GPU 実装することを考えると，データ転送時間はほぼ 0 と見積もることができ，残りの未 GPU 化関数を CPU 実行時間と同じ性能で GPU 化できるとすればフル GPU 化時には 2GPU 実行で 2CPU 実行の 3.8 倍，4GPU 実行で 4CPU 実行の 3.5 倍の性能を達成できると予測できる．

7.　まとめ

本研究では [4] に引き続き，筑波大学計算科学研究センターで開発されている City-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，weak scaling において，2 プロセス実行で 2CPU に対して 2GPU 実行で 2.97 倍，4 プロセス実行で 4CPU に対して 4GPU 実行で 2.78 倍の高速化を達成することができた．また，残りの CPU 実行している関数を GPU 実装してフル GPU 化を完了した場合，2GPU で 2CPU の 3.8 倍，4GPU で 4CPU の 3.5 倍の性能を達成できることが期待される．

今後の課題として，今回の実験で 2 MPI プロセス（2GPU）から 4 MPI プロセス（4GPU）と増やした場合，weak scal-

ⓒ 2019 Information Processing Society of Japan

ing であるにも関わらず通信時間が大きく増加したことから，より大規模な並列化を行った場合の通信時間の増大とその詳細を調査する必要がある．今回使用した PPX では HCA が 2GPU に対して 1 台であったことが主な原因でノード間通信性能の悪化が生じたとしたが，本来の実行予定環境である Cygnus ではノードあたり 4GPU に対して 4HCA が搭載されており，今回の結果ほどの通信性能の低下は発生しないと考えられる．また，実際に Cygnus を用いて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．

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⟩ (2019.06.24)
[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/nicam/index.html⟩ (2019.06.24)
[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] 辻 大亮, 多田野 寛人, 朴 泰祐, 池田 亮作, 佐藤 拓人, 日下 博幸 : 都市気象 LES コードの並列 GPU 環境における高速化．研究報告ハイパフォーマンスコンピューティング（HPC），2019-HPC-168, No.8,（February, 2019）．
[5] 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)．
[6] 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).
[7] 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)．
[8] 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)．
[9] 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).
[10] NVIDIA CORPORATION: PGI — Resources — CUDA Fortran 入手先 ⟨https://www.pgroup.com/resources/cudafortran.htm⟩ (2019.06.24)
[11] NBCL.: MVAPICH :: GDR Userguide 入手先 ⟨http://mvapich.cse.ohio-state.edu/userguide/gdr/⟩ (2019.06.24)
[12] Oak Ridge Leadership Computing Facility: Summit - Oak Ridge Leadership Computing Facility 入手先 ⟨https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/⟩ (2019.06.24)
[13] Lawrence Livermore National Laboratory: Sierra — High Performance Computing 入手先 ⟨https://hpc.llnl.gov/hardware/platforms/sierra⟩ (2019.06.24)
[14] GSIC.: TSUBAME3.0 — [GSIC] 東京工業大学学術国際情報センター 入手先 ⟨https://www.gsic.titech.ac.jp/tsubame⟩ (2019.06.24)
[15] National Institute of Advanced Industrial Science and Technology(AIST): ABCI 入手先 ⟨https://abci.ai⟩ (2019.06.24)

ⓒ 2019 Information Processing Society of Japan