Vol.2022-HPC-183 No.6
2022/3/17

都市気象シミュレーション City-LES の OpenACC による完全 GPU 化と並列性能評価

渡邉孔英[1]　菊池航平[1]　朴泰祐[2,1]
佐藤拓人[2]　ドアングアンヴァン[2]　日下博幸[2]

概要： 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シミュレーションコードである City-LES は、都市レベルの気象現象の詳細シミュレーションを行うアプリケーションである。我々はこのコードを高速化するため、これまで主な計算をすべて GPU 実装してきた。これにより計算が著しく高速化した結果、通信にかかる時間が目立つようになったため、ポアソン求解におけるマルチグリッド前処理の袖通信に注目し、その通信オーバーヘッドを削減する手法を実装した。その上で、東京駅周辺の実都市モデルを用いた性能評価を行い、計算科学研究センターで運用されている GPU・FPGA 混載型スーパーコンピュータ Cygnus 上の 4 ノード（16GPU）から 64 ノード（256GPU）での並列実行を行なった。その結果、最大 15.9 倍の性能向上を達成したことを確認した。

キーワード：アプリケーション GPU 化，LES，OpenACC，マルチグリッド前処理

1. はじめに

GPU (Graphics Processing Unit)を使った大規模クラスタによる研究は近年、特に高性能を目指す計算システムにおいて広く行われている。トップレベルの高性能計算システムでは電源供給能力による制限が性能向上を阻むようになりつつあり、そのような状況で計算性能を高めるため、高い消費電力対計算性能をもつ演算加速装置を搭載するシステムが多くなっている。GPU は高性能計算において用いられる演算加速装置としては代表的なもので、数千の小さなコアによる高い SIMD (Single Instruction Multiple Data)並列性と、HBM2 メモリなどによる高いメモリバンド幅が特徴である。現在、GPU を搭載したもので世界最速のスーパーコンピュータである、Oak Ridge National Laboratory の Summit [1]は、6 つの NVIDIA Tesla V100 GPU を搭載したノードを 4608 個有する。

大規模気象シミュレーションでは空間をグリッド分割して空間並列性による並列化を行うのが典型的で、ほとんどの計算が SIMD 処理で行えるため、GPU での実行に向く代表的なアプリケーションである。このシミュレーションに含まれる計算の多くは、Coalescing Access によってメモリバンド幅を活用することも容易である。加えて、近年の GPU は最大 80GB もの大容量のメモリをもち [2]、大規模シミュレーションに対応しやすくなっている。

筑波大計算科学研究センターで開発されている City-LES コード [3]は、従来の流体計算に加え建物による日陰や輻射などを取り込んだ気象シミュレーションを行い、都市区域の気温や気流を解析できる。City-LES コードは都市計画に役立てることを目標に開発されており、本研究では計算をすべて GPU 上へ移植することで、このコードの高速化を試みる。

GPU コンピューティングは高い並列性とメモリバンド幅を提供するものの、実際に CPU コードから GPU コードへ移行する際にはいくつかの問題がある。特に、CPU メモリと GPU メモリの間のデータコピーが必要になることは重大である。この 2 つのメモリではアドレス空間が異なるため、プログラムを部分的に GPU で実行するよう実装した際は、CPU で行うほかの処理との間にデータの同期を行う必要がある。OpenACC2.0 [4]によるコーディングでは、CPU メモリと GPU メモリの間のデータイメージを、BIOS のサポートのもと、仮想的に共有することが可能で、明示的なデータ転送を記述しなくても済むようにプログラムできる。これにより、ユーザーはデータの転送を記述しなくても済むものの、実際には水面下で転送が行われるため、目に見えない性能低下の要因となる。

本研究で対象とする City-LES コードは、都市街区における気温と風速の解析のための Multi-Physics 計算という、従来の流体計算による気象シミュレーションより著しく複雑なモデルを対象とする。そのため City-LES には、流体計算のように GPU での実行に向くもの以外に、並列性が低いため GPU で実行しても大きな高速化が期待できない計算も含まれる。

GPU の 1 コアあたりの計算性能はハイエンドな CPU のコアに比べて著しく低い。このため、GPU を少量の計算、例えば大きなコードの中にある逐次実行が必要な部分に適用することは、計算の低速化を招く。ただそのような小さな計算でも、先述のような CPU と GPU のメモリ間データ転送が発生すれば、いかにその計算部分が CPU で計算され、GPU で実行するよりも部分的に高速実行できたとしても、結局はデータ転送コストの分、マイナスになる可能性がある。従って、データ転送のコストは全体の計算実行コストに含めて考える必要がある。

1 筑波大学 情報理工学位プログラム
2 筑波大学 計算科学研究センター

本研究では、部分的な GPU 化により CPU-GPU 間データ転送が起こっているコードに対し、すべてのコードを完全に GPU へ移植することで、計算の高速化とデータ転送の削減を行う。その後、Strong Scaling 性能の低下要因となっていた袖通信オーバーヘッドの削減を行う。またこれらの実装を、実際の東京駅周辺エリアのモデルに適用し評価する。

City-LES のような気象シミュレーションコードを GPU 化る上での本研究の貢献は以下の通りである。

1. CPU-GPU 間データ転送のオーバーヘッドを伴う CPU での計算を残すより、計算が遅くなるコードも含めた全部分の GPU 移行が、全体としては性能を向上させることを実証する
2. マルチグリッド前処理の改良により袖通信オーバーヘッドを削減して Strong Scaling 性能を向上させる
3. GPU により、Multi-physical なモデルに含まれる、単純な流体計算コードと複雑な気象シミュレーションの両者の性能が改善できることを確認する
4. 高気温に対応できる都市計画に応用可能な、実世界のモデルを対象とした詳細気象シミュレーションを、全 GPU 化されたコードで実行する

2. City-LES

City-LES [3]は非常に高い空間分解能(約 1 から 5m)でマイクロスケールの気象・気候をシミュレーションする、LES (Large-Eddy Simulation)モデルである。このモデルは気象モデルと計算流体シミュレーションモデルの組み合わせで構成されている。このモデルは、以下のような特徴をもつ。

(1) 都市を陽に解像する
(2) 街路樹の熱的・力学的な影響を考慮する
(3) 建物と街路樹による影を解像する
(4) 建物と街路樹による多数回の反射を考慮する
(5) 雲物理・大気放射モデルともカップルする

これらにより特に都市街区の気温や風の環境を、路肩の樹や建物の影を取り込んで高解像度に解析できる。City-LES を利用すれば、ドライミストのような都市のヒートアイランド現象を抑制する方策を、定量的に評価することができる。これらのシミュレーションができることにより、都市構造の研究者や行政における計画策定を助けることができる。

都市街区の熱環境を LES でシミュレーションするためには、建物や道などでの 3 次元輻射過程を計算できるモデルが必要である。City-LES はラジオシティ法 [5]を使った放射モデルをこの過程を計算するのに組み合わせている。ラジオシティ法を用いることで、複雑な都市ブロックにおける光も高精度で計算することができる。表面温度は顕熱・潜熱とともに、短波長・長波長の輻射を元に計算でき、LES モデルと組み合わせた際は、都市の詳細な熱環境が約 1-5m の空間解像度で再現できる。

City-LES コードについては、OpenMP+MPI で並列化されていたコードを元に以前から全体的な GPU 化を行なっている [6]。これにより多くの計算が GPU 化されたものの、地上の放射計算などいくつかの機能は GPU 化が煩雑なため CPU 側へ残された。このような「逆オフローディング」の形になってしまう実装が原因となり、実都市モデルを対象に計算する際、CPU-GPU 間で大きなデータ転送が必要になっていた。本研究では、計算時間上の大きなコストとなり得るデータ転送を除去するため、City-LES に含まれる計算をすべて GPU で実行できるように実装し、このコードを実都市モデルへ適用する。

GPU 化したコードを適用する対象問題として、2020 年東京オリンピックのハイライト地点となる予定であった、東京駅周辺のモデルを使用する。残念ながらオリンピックは 1 年延期され、開催地は北海道の札幌市へ変更になった。しかし、開催地が変更となった主な理由は東京で酷暑が予想されたためであり、このような環境は City-LES でまさに対象としたいものであることから、この地域を引き続き目標としている。図 1 は本研究で扱う東京駅周辺を示したもので、これは CPU 版の City-LES によるシミュレーション結果の例である。

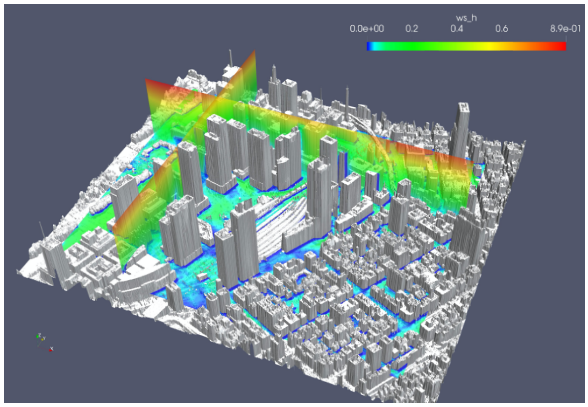

図 1 City-LES によるシミュレーション結果の例

地表面の温度は、特に夏の盛りでは人間生活に大きく影響し、大域的・局所的な過熱は社会的な課題になっている。City-LES のシミュレーションを高速化し、そのような課題を解決する都市計画に役立てることが、本研究の最終的な目標である。

City-LES コードのオリジナル版は、OpenMP によるスレッド並列化と MPI による分散メモリ並列化を組み合わせ、Fortran 言語で汎用 CPU 向けに書かれている。ほとんどの場合、コードの基本的な構造は、物理空間に対応する 3 次元配列を扱うための 3 重ループによる計算の形をしており、3 次元ステンシル計算となる。従って、MPI を使った 3 次元空間上の隣接通信が、ブロックドメイン分割上で必要に

なる。各 MPI プロセスでは、ほとんどの部分が OpenMP によりスレッドレベルで並列化されている。3 次元空間は X 軸と Y 軸について分割されているが、地面に鉛直な方向となる Z 軸では分割しない。気象計算において、Z 軸では重力の関係する演算を扱うことが多く、MPI 分割を単純に行うことが必ずしも簡単でないためである。従って、MPI プロセス間での主な通信は 2 次元での隣接通信となる。

3. GPU 実装

3.1 これまでに行なってきた実装の概要

これ以降、「GPU 化」という言葉を CPU から GPU へのコード移植の意味で使う。

我々は現在までに、City-LES コードの GPU 化を段階的に進めてきた。まず、GPU 化が煩雑であるとみられた地表面関連計算を除き、CUDA Fortran および OpenACC を用いて計算を GPU 化した [6]。この時点で CUDA Fortran で GPU 化したコードと OpenACC で GPU 化したコードの実行時間を比較し、両者の実行時間がほとんど変わらないか、OpenACC の方が高速になることがわかった。また、GPU での実行が CPU よりも低速になるものも含めた GPU 化を行い、これによるデータ転送の削減が全体の実行時間短縮に役立つことと、OpenACC による GPU 化は CUDA Fortran よりも簡単であり、これが小さな計算の GPU 化に効果的であることを示した。

その後、地表面関連計算を CPU 側へ残したことに起因する性能低下が、実都市モデルの計算を行う際に顕著になることがわかった。このため、地表面関連計算も含め全計算が GPU 化された City-LES、すなわち City-LES/GPU を作成し、簡易モデルに適用した際の実行性能について報告している [7]。

3.2 City-LES の基本的な GPU 化

City-LES では MPI レベルの並列化が 2 次元ブロッキングで行われているため、隣接通信で交換する 3 次元領域表面はブロックストライドやストライドパターンでのアクセスとなる。MPI の type-vector 機能が GPU 上で非効率的なのは広く知られており、通信コストを最小化するため、手動で記述した GPU メモリ上での非連続領域の packing/unpacking 処理を適用した。加えて、GDR (GPUDirect RDMA) [8]を、ノードを超えた GPU 間通信のレイテンシを短縮するために利用している。

CityLES の GPU 実装では、CPU 版のオリジナルコードから、OpenACC directive で OpenMP directive を置き換え、1 つの MPI プロセスあたり 1 つの GPU を扱うようにしている。従って、CPU におけるマルチスレッド実行が、GPU でのマルチコア実行に対応する。今後のコード保守性の観点から、元の CPU 版からデータ構造やループ構造の変更は行わず、OpenMP directive を、OpenACC directive と補助的な directive やクローズで置き換えることにとどめている。

図 2 に City-LES の基本的な関数構造を、本研究において測定した 8 ノードでの実行における元の CPU コードの実行時間とともに示す。以下に主に時間を消費している部分の概要を示す。

- smac1

ポアソン方程式をマルチグリッド前処理つき Orthomin(1)法 [9]を使って解く。ステンシル計算と隣接プロセスとの袖通信を行う。

- surface_driver

モデルに含まれる建物の壁面や植生の輻射にかかわる計算を行う。分岐を多く含む、メモリバンド幅律速な計算で構成される。

- sgs_driver

サブグリッドスケールの乱流計算を行う。ほとんどが行列加算とステンシル計算によって構成される。

- update_rk_u

ルンゲ-クッタ法で近似解の計算を行う。行列の平均値計算、Allreduce 通信とステンシル計算を行う。

- check

CFL 条件に合い、計算が成立しているかをチェックする。行列の最大・最小値計算と allreduce 通信を含む。

- advection

移流項の計算を行う、ステンシル計算。

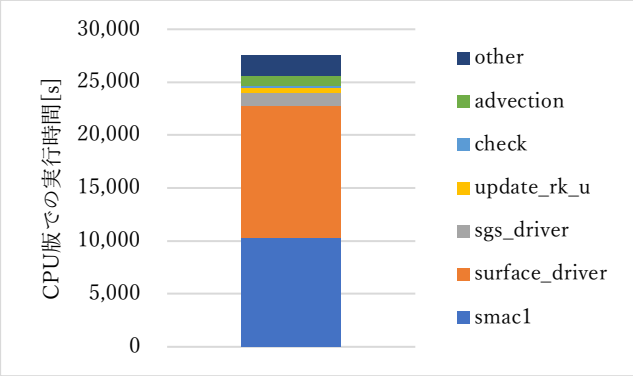

図 2 CPU 版 City-LES において
実行時間に占める割合の多い関数

kernels 指示文を付けたときの自動的な並列化に代表されるように、CUDA に比べ OpenACC は、逐次実行が必要なものを含むどのような部分でも、簡単に GPU カーネルとして記述することができる。一般的に計算速度向上が見込めない部分は GPU 化しないが、本研究では次のようなポリシーをもっている。

たとえ多少の関数(あまり重くない部分)の実行が GPU 上では CPU 上より遅くても、全体の実行時間を短縮するためには、それらを GPU 化して CPU-GPU 間のデータコピーを防ぐ方がよく、また OpenACC は CUDA よりも非常にその実現が容易である。

3.3 マルチグリッド前処理における通信の最適化

City-LES/GPU の実効性能を評価する過程で、ポアソン方程式の求解を行う過程で発生する通信が、Strong Scaling 性能の上でのボトルネックとなっていることがわかった。

ポアソン方程式の求解を行う処理の大部分は、3.2 節で触れた smac1 に含まれる MGOrthomin_m 関数で行われる。この関数ではポアソン方程式の求解を行うためのアルゴリズムとして、マルチグリッド前処理つき Orthomin(1)法を実装している [9]。マルチグリッド法は、グリッドの辺長を半分にしながら計算することで、波長の異なる残差を均一に減衰させる手法である。このため、計算は 1 段進むごとに辺長は半分に、扱うグリッド点数は 8 分の 1 に減少する。この性質上、計算対象となる点の数とそれに直接影響を受ける並列性は段数が進むたびに減少していく。図 3 に例として 512x512x512 の空間グリッドの問題に対し、5 段階粗いサイズまで行った場合のマルチグリッド法計算の概略を示す。City-LES/GPU ではマルチグリッド法で直接求解するのではなく、前処理にすることでマルチグリッド計算を行う段数を減らし、ある程度並列性が確保できる範囲までで計算をするようになっている。それでも、計算を行うグリッドサイズの減少による高次段での並列性低下は避けられない。

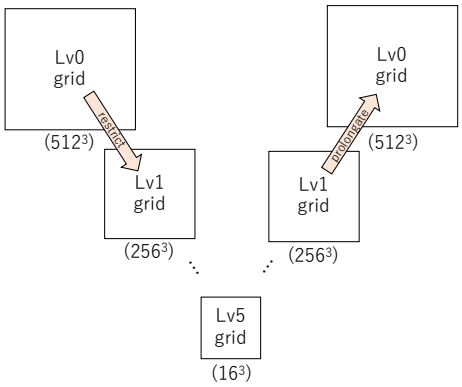

図 3 マルチグリッド計算の略図

マルチグリッド前処理では頻繁な小サイズの通信が行われるため、通信にかかる時間の多くは通信レイテンシに起因する。このため、通信で交換するデータサイズを縮小して転送にかかる時間を減らすよりも、通信を行う回数を削減して通信に伴うオーバーヘッドを削減する方が重要であると考えられる。そこで、通信回数を削減する方法として、本研究では二つの手法を試すことにした。

一つ目は、袖領域を深く交換しておいて、各プロセスが必要な袖領域を数回分計算できるようにする方法である。前節にあげたマルチグリッド前処理の各段階では、主に深さ 1 点のステンシル計算を行う。このため各段で深さ 1 の袖交換を行っていたが、交換する袖領域を深くすることで、必要な袖領域を手元で計算することができるようになる。この方法では直接隣接するプロセスのほか、斜めの方向に位置するプロセスからもデータを受け取る必要があったり、重複した計算を行うぶん計算量が増加したりするものの、通信回数は抑えることができる。

二つ目は、全プロセスに分散している各点のデータを一度の通信でひとつのプロセスへ集約し、そこから先の計算をプロセス内で完結させる方法である。この方法では集約のための転送量の大きな通信や全点を扱うため大きな計算量が発生するが、必要な通信回数を大きく削減できる。

両手法は転送量と計算量の増加があり、扱うグリッドが大きい低次段でこの方法を適用すると、元の方法よりも性能が悪化する可能性がある。このことから、いずれの手法についても、それを適用し始める段数を実行時に指定できるように実装した。また一つ目の方法は二つ目の方法に比べれば転送量と計算量の増加が少ないため、より初期の段から実行時間削減効果が得られるのではないかと考えられた。このことから、一つ目の方法と二つ目の方法を順に組み合わせて使うこともできるよう実装した。なお、コードの都合上、両手法の適用は計算開始段である第 0 段ではなく、一段階粗いグリッドである第 1 段以降で開始できるようになっている。

後述する性能評価では、これらの手法を適用開始段を変えながらそれぞれ試行し、最も実行が高速になった組み合わせでの実行時間を示している。

4. 高解像度実都市シミュレーション

本研究では、City-LES/GPU の実用性能を、東京都にある東京駅周辺を対象に評価した。これはもともと 2020 年東京オリンピックにおけるマラソン競技のハイライト地点であった。東京の街の 8 月の気温(約 35°C)と湿度(80%以上)は非常に高く、多数の観衆の健康を守るためにもこのエリアの気温を予測することは重要であった。残念なことに、オリンピックは 2021 年へ延期され、また結局、マラソンの開催地は北海道の札幌市へと移された。しかしながら、建物や街道、街路樹のあるこの環境でのシミュレーションは気温が各点で変化に富むことがわかっており、有用であると考えている。

本研究では東京駅周辺の 5m のグリッドサイズのシミュレーションを行うにあたり、実際の 3 次元地形を元にした 512x512x512 の空間グリッドのモデルを作成した。

[6]に続く研究で、ラジオシティ法の適用と太陽光の反射・吸収計算のために地表面の物体を扱う surface_driver という重きを占める関数に手を入れた。この部分は芝生や建物、街路樹といった種類のあるものを扱うことから複雑である。

地表面にあたる 2 次元グリッドの各点について、アスファルトや芝生といった材質が都市モデルに定義されている。建物や樹木といった構造物は、材質のほか、建物はその高さ、樹木は葉のついている高さが含まれる。建物と街路樹

は 3 次元構造をしており、隣接グリッドに建つ同じ高さの建物は、合わせてひとつのオブジェクトとして扱われる。

5. 性能評価

5.1 実行環境

本研究では City-LES/GPU コードを、筑波大計算科学研究センターで運用されている複合演算加速スーパーコンピュータ Cygnus [8]で実行した。表 1 に Cygnus の構成を示す。Cygnus には全部で 81 のノードが搭載されているが、本研究ではそのうち最大で 64 ノードを使用した。各ノードは 2 ソケットの CPU、4 つの GPU が標準で装備されている。全ノード中、32 台のノードにはさらに追加で 2 つの FPGA が搭載されているが、本研究では、演算加速器として GPU のみを利用し、FPGA は利用しない。

City-LES/GPU では、MPI プログラム上で GPU の利用を簡単にするため、各 MPI プロセスは 1 つの GPU を利用し，CPU 計算部分では複数コアを想定したマルチスレッド処理を OpenMP で記述している。各 MPI プロセスにおける OpenMP マルチスレッド処理では全てのプロセスに同数の CPU コアが割り当てられることを想定している。Cygnus の 1 ノードは 24 コア(12 コア x2 ソケット)の CPU と 4 つの GPU をもつことから、6 つのコア(スレッド)と 1 つの NVIDIA Tesla V100 GPU が割り当てられた MPI プロセスを、各ノードで 4 つ実行する。並列化の Strong Scaling 性能を測るため、512x512x512 の空間グリッドの問題に対して、使用するノード数を 4 から 64 ノード、GPU 数では 16 から 256 まで変化させた。計算は単精度と倍精度の両方を含んでいる。モデルの時間解像度(Δt)は 0.1 秒であり、今回の評価ではこのモデルで 3000 タイムステップのシミュレーションを行った。

City-LES/GPU の計算結果は CPU 版のものと比較し、温度差が最大約 0.2° C と、無視できるほど小さいことを確認した。このレベルの誤差は GPU 化による計算順序の変化により生じているものだと考えられる。

表 1 Cygnus の構成

CPU	Intel Xeon Gold 6126 (12 cores) x 2
CPU メモリ	DDR4-2666 192GiB (16GiB x 6 channels x 2 sockets)
GPU	NVIDIA Tesla V100 (PCIe) with 32GiB memory x 4
Interconnect	InfiniBand HDR100 x4
OS	CentOS 7.6
コンパイラ	Nvidia HPC SDK 21.2 Intel Compiler 19.1.3
MPI ライブラリ	OpenMPI 3.1.6
CUDA	11.2

5.2 通信最適化手法の比較

本節では、マルチグリッド前処理の通信オーバーヘッド削減手法の試行について述べる。すべての試行で、マルチグリッド段数は 4 段、平滑化に用いる重みつき Jacobi 計算の反復回数は 2 回で固定としている。

図 4 に例として 8 ノードで実行した、袖領域を深く交換する手法を適用した MGOrthomin_m 関数の、計算と通信にかかった時間を示す。また、図 5 には、データを集約して計算する手法を適用した MGOrthomin_m 関数の、計算と通信にかかった時間を示す。この図には、集約を allgather で行い各ノードで冗長計算を行う実装と、gather で 1 ノードに集めて計算し scatter で全ノードへ配る実装の両方の実行時間を示している。

図 4 と図 5 より、最も MGOrthomin_m 関数の実行時間が短くなっているのは、袖領域を深く交換する手法を 1 段目から適用した場合であることがわかる。二つの手法を組み合わせた場合の実行時間も計測したものの、さらなる実行時間の短縮はみられなかった。

また、4 ノード、16 ノード、32 ノード、64 ノードでの実行についても同様の試行を行い実行時間を測定した。その結果、全ての場合で袖領域を深く交換する手法を 1 段目から適用した時に最も実行時間が短くなることがわかったため、次節以降では GPU 版の実行時間として、この設定の場合を示している。

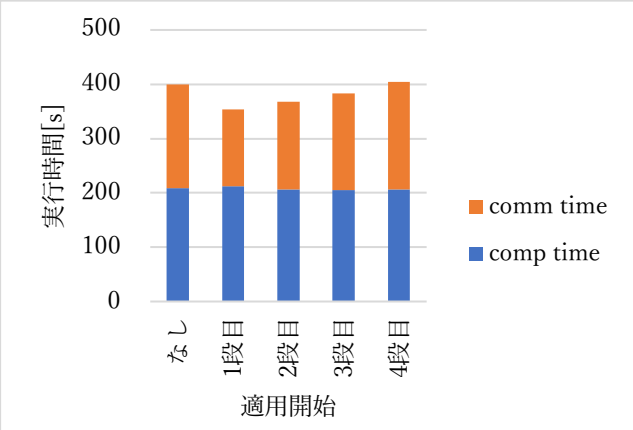

図 4 袖領域を深く交換する方法を適用した際の実行時間 (8 ノード)

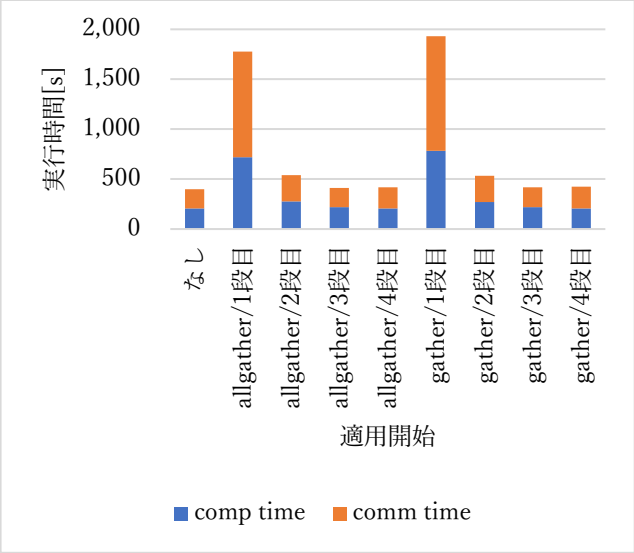

図 5 集約して計算する方法を適用した際の実行時間 (8 ノード)

5.3 全体の実行性能

この節では、計算を CPU で行う City-LES(以降 CPU 版とする)と、全ての計算を GPU 化した City-LES/GPU(以降 GPU 版とする)の間で性能比較を行う。

図 6 に対象問題(東京駅周辺モデル)の CPU 版と GPU 版の実行時間と、CPU 版に比べた GPU 版の高速化率を、4 から 64 ノードを用い、4 ノードを基準とした Strong Scaling で示している。最大の性能向上が得られたのは 8 ノードでの場合で、GPU 版は CPU 版の 15.9 倍の性能向上を達成している。これは、計算性能が下がり得るような複雑さの都市モデルであっても、全 GPU 版が有効であることを顕著に示している。

図 7 は、CPU 版と GPU 版の並列化効率を示している。これは CPU 版と GPU 版それぞれの 4 ノードでの実行時間を基準として、ノード数の増加と速度向上の比を示しており、1 のときリニアに高速化していることを表す。CPU 版の並列化効率は 1.17 から 1.46 の間、GPU 版では 1.49 から 2.92 の間になっている。いずれの場合でも並列化効率は非常に良い。特に GPU 版では顕著であるが、これは GPU 版の 4 ノードでの実行が比較的低速であることに起因する。この原因の詳細は不明だが、実行中にメモリ確保等に関わるオーバーヘッドが発生していると推測される。

8 ノードを超える場合の高速化率はノードが増えるほど減少していくものの、64 ノードの場合でも 1.35 と高い並列化効率を維持している。

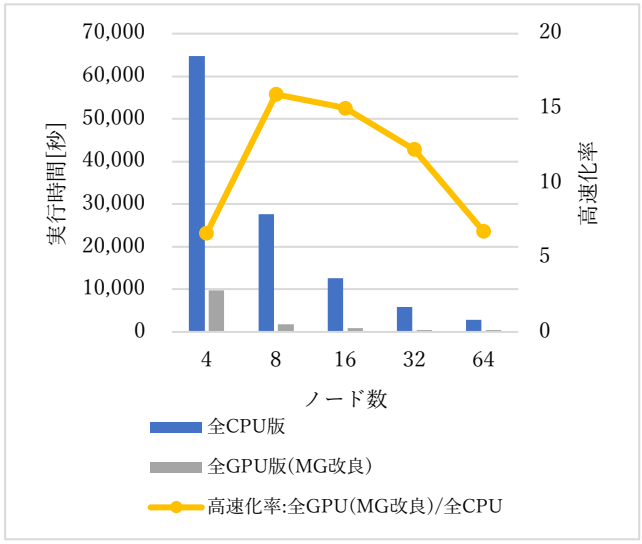

図 6 全 CPU 版と全 GPU 版の実行時間の比較

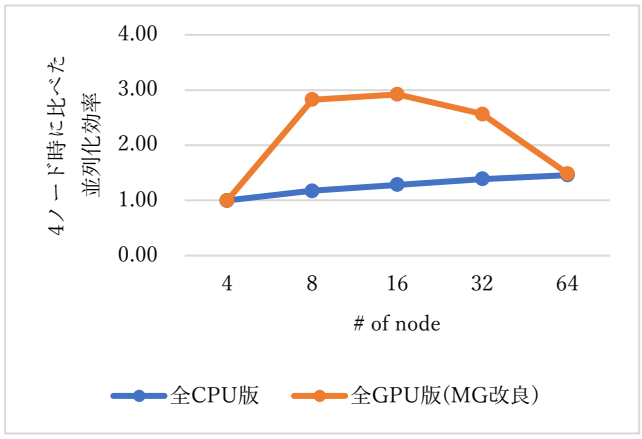

図 7 全 CPU 版と全 GPU 版の並列化効率(4 ノード時基準)

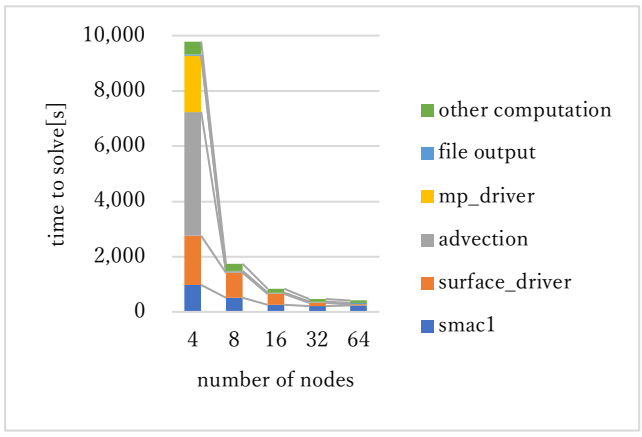

図 8 全 GPU 版の実行時間の内訳

図 8 に、4 から 64 ノードで実行した場合の全 GPU 版の実行時間の簡単な内訳を示す。この図では、smac1(青色)がポアソン求解、surface_driver(オレンジ色)が地表面の構造物に関する計算、advection(灰色)が移流項についてのステンシル計算、mp_driver(黄色)が雲物理にかかわる数値計算、file output(水色)がファイル出力、other computation(緑色)が

これら以外の GPU 化された計算にかかった時間を、それぞれ表している。図 8 をみると、4 ノードの場合においてのみ advection および mp_driver 部分にかかる実行時間が著しく長く、これが高速化率が低くなっている主な原因であることがわかる。また、8 ノード以上では多くの関数の実行時間が短縮していくものの、それに比べると smac1 にかかった時間は変化が小さい。最終的に、64 ノードの場合では smac1 関数が実行時間全体の 54.3%を占めるまでになっており、この部分が Strong Scaling 上のボトルネックとなっていることがわかる。

6. 考察

4 ノードでの実行においては mp_driver と advection にかかる実行時間が 8 ノード以上の場合に比べて著しく長い。プロファイリングの結果、4 ノードの場合であっても、これらの関数の GPU 版に含まれる計算カーネルそのものの実行にかかっている時間は 8 ノードの場合の 3 倍以内に収まっており、実行時間が伸びた主な要因とはいえないことがわかっている。そのため、計算以外に大きなオーバーヘッドがあり、それによって性能低下が起こっているものと考えられるが、原因については究明中である。

高速化率は 8 ノードをピークに、16 ノード以上では低下している。この直接の原因は、smac1 の Strong Scaling 性能が低いことである。しかしこの部分以外の GPU 化された計算にかかる時間が非常に短くなっている。smac1 は他の計算に比べて、多くの通信を含む。Strong Scaling においては、GPU による演算高速化がなされていることから、CPU 版に比べ相対的に通信ボトルネックが顕著になっているものと思われる。

マルチグリッド前処理の通信削減に関して、今回の東京駅周辺の条件では、どのノード数においても、1 段目から袖領域を深く交換する方法を適用した場合で最も実行時間が短縮されることがわかった。マルチグリッド前処理で必要になる通信や計算は、マルチグリッド計算のパラメータの影響を受ける。今回の評価はマルチグリッド段数を 4 段、平滑化に用いる重みつき Jacobi 計算の反復回数を 2 回とした。しかし、例えばマルチグリッド段数を増やせば細かい通信の増加や計算の並列性低下が顕著になる。また重みつき Jacobi 計算の反復回数を増加させると、1 段の計算に必要な袖通信の回数が増え、袖領域を深く交換する方法では転送する領域や重複して行われる計算が増加する。このような厳しいケースでは、通信を大きく減らしたり並列性を高めたりできる、集約を行う方法が有利になる可能性がある。これは別の面から見れば、性能を大きく損ねずにマルチグリッド段数や、平滑化回数を増やせるようになったということでもある。このことを活かし、さらに前処理の効果を高めて高速な求解に繋げられるようなパラメータを検討することは、今後の課題の一つである。

実行性能に影響する要素として、シミュレーション対象のモデルの複雑さが挙げられる。128^3 の小さな問題サイズで、空間内にひとつの建物がある場合、複数の建物が一様に分布している場合、まわりに木のある建物が一様に分布している場合の 3 種類について、実行時間を測定した。ぞれぞれの実行時間は 43.3 秒、54.6 秒、57.4 秒であり、2 つ目と 3 つ目では 1 つ目に比べ、26%および 33%実行時間が増加している。この実行時間増加はポアソン方程式の求解を行う smac1 において、解の収束に時間がかかるようになったことによって起こっている。この結果は、今回の評価で用いた東京駅周辺モデルより複雑な都市モデルでも、ポアソン求解処理の実行時間の増加が無視できるレベルにあることを示している。

7. まとめ

本研究では、筑波大計算科学研究センターで開発されている都市レベル気象シミュレーションコード City-LES の全 GPU 化版である City-LES/GPU にマルチグリッド前処理の通信最適化を実装した上で、東京駅周辺の実都市モデルに適用し、実行性能の評価を行なった。City-LES は地表面の材質や、太陽光による輻射、建物を考慮するように設計されており、全 GPU 実装によって、詳細な都市の気温と気流のシミュレーションをすべて GPU 上で行える。City-LES/GPU における計算の GPU 実装は、OpenACC により行った。

本研究では City-LES/GPU を東京駅周辺をグリッドサイズ 5m という高解像度により X・Y・Z 次元について 512x512x512 というグリッドサイズのモデルに対して実行した。実装と評価には筑波大学計算科学研究センターで運用されている Cygnus スーパーコンピュータを用い、最大 256 台の Tesla V100 GPU をもつ 64 ノードまでスケールさせ、実行時間を測定した。この実験で CPU 版の City-LES に比べ、8 ノード(32 台の GPU)で実行した場合に最大 15.9 倍の高速化を達成し、さらにいずれのノード数においても少なくとも 6.71 倍の高速化が達成されることを確認した。

Strong Scaling 性能の面で見ると、4 ノードでの実行を基準に見て、8 ノードでの実行は 2.82 倍、32 ノードでの実行は 2.57 倍の並列化効率がある。ただし、64 ノードでの並列化効率は 1.49 倍となるため、今回対象とした都市モデルに対してはおおむね 8 から 32 ノードが性能の出る範囲とわかった。気象シミュレーションでは、同じ領域に対していくつもの条件やパラメータを試せることが重要である。このため一度に 64 ノードを使って実行するのではなく、32 ノードでの実行を 2 つ同時に行い、より高い計算スループットを発揮するといった手法が有用だと考えられる。また 64 ノードで高速化率が低下することの原因として、プロセスあたりに割り当てられるグリッド点数が減少することで、計算にかかる時間が短くなりすぎたことが挙げられる。こ

れはさらにグリッド点数が多い、広範囲や高精細なシミュレーションを高速に実行できる余地があることを示唆している。City-LES を用いたシミュレーションでは 1m 間隔の 2000x2000x2000 点でのものも計画されており、それに対する高速化も期待できる。

実際に City-LES が利用される場合、数万タイムステップの計算を行うことが多い。今回評価で行った 3000 ステップの 10 倍の 30000 ステップのシミュレーションを行う場合、Cygnus の 8 ノードを利用すると、CPU 版では 77 時間かかっていたシミュレーションを、本研究において実装した City-LES/GPU により、4.8 時間で実行できる計算になる。一般的にスーパーコンピュータのバッチジョブシステムではジョブ当たりの実行時間が制限されており、例えば Cygnus の運用条件は、1 ジョブ当たり最大 24 時間である。このため連続で行って 77 時間かかるシミュレーションは、実際には途中で中間ファイルへの保存・読み出しを伴いながら、数回に分割して実行する必要がある。本研究の実装により実行にかかる時間が 4.8 時間に短縮され、同じシミュレーション一度の実行で完了できるようになったことで、中間保存が不要になる。ここでは Cygnus で、実行したいシミュレーションを 1 つのジョブで完結できるという望ましい結果となったが、一般的に GPU 化による実行時間の短縮は単に短時間で実行できるというだけでなく、ジョブ前後の I/O 処理も最適化できるという大きな効果を持つ。

City-LES を用いたシミュレーションは東京駅周辺だけではなく、札幌市をはじめとした他の都市も対象に行われている。今後は City-LES/GPU をこうした実都市シミュレーションに適用し、このツールを今後の都市計画、ひいては人間生活の改善に役立てていきたいと考えている。

謝辞

本研究成果は筑波大学計算科学研究センターの 2021 年度学際共同利用プログラム課題「都市を対象とした気象学 LES モデルの開発と応用」における複合型演算加速クラスタ Cygnus の利用により得られたものです。

また、本研究の一部は「高性能汎用計算機高度利用事業」における課題「次世代演算通信融合型スーパーコンピュータの開発」、および JSPS 科研費 21H04869「多重複合演算加速機構を用いた次世代スーパーコンピューティング」の助成を受けたものです。

参考文献

[1] "Summit - Oak Ridge Leadership Computing Facility," https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

[2] "NVIDIA A100|NVIDIA," https://www.nvidia.com/ja-jp/data-center/a100/.

[3] R. Ikeda, H. Kusaka, Iizuka S. and T. Boku, "Development of Urban Meteorological LES model for thermal environment at city scale," 9th International Conference for Urban Climate, Toulouse, France, 2015.

[4] "Homepage | OpenACC," https://www.openacc.org.

[5] M. F. Cohen and D. P. Greenberg, "a radiosity solution for complex environments," ACM SIGGRAPH Computer Graphics, vol. 19, no. 3, pp. 31-40, July 1985.

[6] D. Tsuji, T. Boku, R. Ikeda, T. Sato, H. Tadano and H. Kusaka, "Parallelized GPU Code of City-Level Large Eddy Simulation," Proc. of int. Symposium on Parallel and Distributed Computing(ISPDC), Warsaw, 2020.

[7] 菊池 航平, 渡邉 孔英, 朴 泰祐, 佐藤 拓人, ドアン グアンヴァン, 日下 博幸, “都市気象シミュレーション City-LES における詳細モデル計算の GPU 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）,2021-HPC-179, No.7,1-6.

[8] "Cygnus Supercomputer," https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/2018/12/About-Cygnus.pdf.

[9] H. Tadano, R. Ikeda and 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), Kyoyo, Japan, 2016.