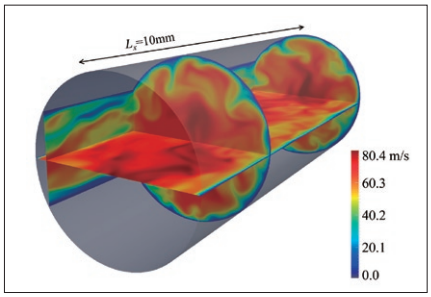
e-Science Journal
東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center
Vol. 15

TSUBAME ESJ.



TTX：反応性乱流の直接数値計算
TTX : A Direct Numerical Simulation Code for Turbulent Reacting Flows

GPUを用いた高性能並列AMRのための ハイレベルフレームワーク
A High-level Framework for Parallel and Efficient AMR on GPUs

クラウド的GPU利用を簡単にするツール：DS-CUDA
DS-CUDA : A Handy Tool to Use GPUs in a Cloud Network

http://www.gsic.titech.ac.jp/TSUBAME_ESJ

TTX：反応性乱流の直接数値計算

源 勇気*　青木 虹造**　店橋 護*
* 東京工業大学・工学院機械系　** 東京工業大学・機械宇宙システム専攻

アメリカエネルギー省によると、今後30年以上にわたり、化石燃料の燃焼によるエネルギー供給量は全世界の消費量の8割以上を担うと予測されており、これによりより厳しい環境規制が各種燃焼機器の設計に課せられる。直接数値計算（DNS）は、産業DFD実行に必要な乱流燃焼モデル開発などの乱流燃焼研究において重要な役割を果たすが、一方で計算コストや物理境界条件の設定において困難を伴う。
本記事では、これらの問題に対する、東京工業大学で開発された直接数値計算コードTTXに組み込まれた数値計算手法について説明する。

はじめに　1

アメリカエネルギー省によると、今後30年以上にわたり、化石燃料の燃焼によるエネルギー供給量は全世界の消費量の8割以上を担うと予測されている[1]。このため今後は、より厳しい環境規制が各種燃焼機器の設計に課せられる。直接数値計算（DNS）は、各種燃焼機器開発設計に関わる産業数値流体力学（CFD）に必要な乱流燃焼モデルの開発などの乱流燃焼研究において重要な役割を果たす。

過去数10年にわたり、DNSは統計的に1次元な自由伝播乱流火炎や周期境界を持つ領域内の燃焼場などの簡単な計算領域形状に対して適用されてきた。しかし、近年の計算機性能の向上により、比較的計算コストが高く、比較的複雑な物理境界を持つ系のDNSが可能になった。これらの境界形状には、例えばV型乱流火炎や噴流火炎、JICFと呼ばれるガスタービンエンジンに関係する燃焼形態や旋回乱流火炎が挙げられる。また、計算機性能の向上により、より複雑で現実的な燃料種の燃焼を詳細に計算できるようになり、これらの計算結果から得られる知見は、今後のモデル開発において有望であると考えられる。これらの複雑な物理境界形状の一つに、計算メッシュ形状に沿わない任意の形状を持つ壁条件や流入条件があるが、過去に行われた比較的複雑な形状を持つDNSでは、これらの物理境界条件に仮定された速度・スカラー分布を適用し、現実的ではない領域形状を適用するなどして、この問題を避けてきた。ポータブルで、高精度な埋込境界条件手法は、この問題を直接的に解決するひとつの手法であり、この計算手法の開発が望まれている。

また、メタンやnヘプタンなどの現実的でより複雑な燃料種を用いた詳細化学反応を考慮に入れたDNSは、乱流燃焼モデル開発に必要ないわゆる火炎と乱流の干渉機構の解明に重要である。しかし、これらの素反応を考慮に入れた計算には膨大な計算コストが必要であり、これの実現には、化学反応速度に関する時間積分の計算速度を飛躍的に向上させる計算手法が必要である。

本記事では、東工大において開発された直接数値計算コードTTXに組み込まれたこれらの数値計算手法について解説し、東京大学Reedbush-Uにおける性能及び精度を評価する。

直接数値計算手法　2

2.1　基礎方程式と計算手法

乱流燃焼直接数値計算の基礎方程式は、質量、運動量、エネルギー、N-1個の化学種の質量分率に関する完全圧縮性の保存方程式からなる。ここで、Nは直接数値計算において考慮する全化学種数である。基礎方程式は、4次精度中心差分法を用いてデカルト直交格子状で離散化され、3次精度Runge-Kutta（RK）法を用いて時間方向に積分される。差分法により起因する非物理的な数値振動は、6次精度陽的フィルターまたは、4次精度コンパクトフィルターを用いて除去される。全ての計算境界条件は、Navier-Stokes characteristic boundary conditions（NSCBC）境界条件を用いて記述される。より詳細な計算手法は、以前の研究において報告されている[2-6]。

2.2　MTS及びCODACT

燃焼のDNSにおいては、化学反応項を含む基礎方程式の時間積分が困難となる。陽解法を用いる場合、その時間刻み（Δt）は流れ場の最小の反応時間スケール以下に設定される必要があるが、その値はCFL条件によって制限される時間刻みよりもはるかに小さなものとなり得る。MTS（multi-timescale）法[7]は、化学反応に関する常微分方程式（ODE）を反応時間スケールよりも大きなΔtによって積分するための時間積分手法である。MTS法において、各化学種（Y_k）はその時間スケール（τ_k）に基づき、一つの積分グループに属する。化学種Y_kが属するグループのグループ番号（M）は$M=\log_{10}(\Delta t/\tau_k)+1$により与えられる。第$M$グループに属する

化学種の保存式は時間刻み $\Delta t_M = \Delta t/10^{(M-1)}$ を用いて時間積分される。本時間積分は最小の Δt_M から順次実施される。Δt_M を用いた時間積分により第 M グループに属する化学種の質量分率が収束すると、そのグループの質量分率はそのまま固定され、続けて時間刻み Δt_{M-1} を用いた時間積分がODE系に対して実施される。以上のように、MTS法を用いることで化学種の最小の時間スケールに制限されない Δt をDNSにおけるグローバルな時間刻みとして利用することが可能となる。

計算コスト削減に対するもう一つのアプローチとして、ODE系のサイズの縮小が挙げられる。CODAC（correlated dynamic adaptive chemistry）[8] は計算領域の各点・各時間ステップにおいて簡略反応機構を動的に生成する手法の一つである。CODACにおいては、化学反応において重要となるいくつかのパラメータを用いて位相空間が構築される。通常このパラメータとしては、温度と燃料、酸化剤、ならびに数種のラジカルの質量分率が選択される。CODACでは、ある点における瞬時の位相パラメータと別の参照点（時空間的に異なる1点）のパラメータを比較することで、構築された位相空間における二点の相関を調べる。相関の有無を判断するための閾値はユーザーによって設定される。もしこの二点における位相パラメータに相関がある場合には、参照点において既に用いられていた簡略反応機構は相関のある別の点において再利用される。ある点と相関を示す参照点が存在しない場合には、新たに簡略反応機構を生成する必要がある。簡略反応機構の生成にはPFA（path flux analysis）[9] が用いられる。

上記手法に加え、TTXではCODACT（CODAC and transport）[10] によりさらなる計算コストの削減がなされる。CODACTでは、CODAC同様の位相パラメータの相関を用いることにより輸送係数（粘性係数、拡散係、熱伝導率）の計算にかかるコストが削減される。輸送係数計算の省略のために用いられる位相パラメータとしては、温度と高い濃度を示す数化学種のモル分率が用いられる。CODACの場合と同様、ある点における位相パラメータが別の点における位相パラメータと相関を示す場合には、その点における輸送係数は計算されず、参照点において計算された値が再利用される。

2.3　埋込境界法

本研究で開発した埋込境界法は、一般的に用いられる仮想領域 [11] の手法に基づくが、ポータブル及び高精度化のためにこの手法を改良した。高精度化に関する重要な手法として、従来の手法では仮想領域として、非流体領域及び流体領域に接するメッシュのみが用いられているが、これに加え、仮想領域にある壁方向厚みを含む領域を含める。壁垂直方向のメッシュの厚みは、例えば、等間隔メッシュで1次元の場合では、max（nd、$nf/2$）と計算される。ここで、nd と nf は、それぞれ中心差分の精度と空間フィルターの精度である。図1に本DNSコードの条件（$nd=4$ and $nf=6$）における計算領域と仮想領域を図示する。

仮想領域のメッシュが決定されれば、それぞれの仮想メッシュ点に対応する参照点が以下の式のように求める。

$$\mathbf{x}_{ref} = 2\mathbf{x}_{wall} - \mathbf{x}_{gs}, \qquad (1)$$

ここで、3つの位置ベクトル $\mathbf{x}$ は、それぞれの仮想メッシュ点に対する参照点位置、壁面位置及びその仮想メッシュ位置を示す。DNS宙で解かれるそれぞれの物理量、q、に関して、参照点における値（参照値）を補完により推定し、この参照値と以下の式を用いて、壁面境界条件に応じ、それぞれの仮想メッシュ位置における値を算出する。

$$q_{gs}(\mathbf{x}_{gs}) = -q(\mathbf{x}_{ref}) + 2q(\mathbf{x}_{wall}), \qquad (2)$$

$$q_{gs}(\mathbf{x}_{gs}) = q(\mathbf{x}_{ref}). \qquad (3)$$

ここで、式（2）はディリクレ条件、式（3）はノイマン条件に対応する。本研究では、仮想領域の値として、速度及び温度には式（2）を、化学種質量分率及び圧力には式（3）を用いた。また、質量保存を満たすため、密度には、式（2）と（3）の両方を用いている。

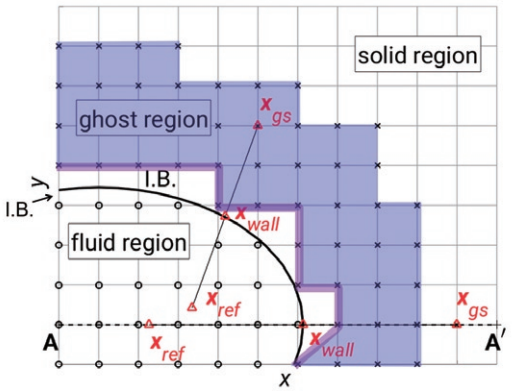


図1　埋込境界法を用いた直接数値計算領域の模式図。青影は仮想領域を示す。

計算速度性能と精度 3

3.1 MTS及びCODACT

TTXを用いて実施した典型的な計算性能を図2に示す。実施した燃焼計算系は、全方向周期境界条件を用いた3次元領域であり、計算においては節2.2のMTS及びCODACT、節2.3の埋込境界法は用いていない。計算領域は予め1000Kに予熱したメタン・空気の予混合気が1気圧で満たされており、計算開始とともに、領域中心部に設ける高温領域により着火開始する。高温領域は、ガウス分布に基づくエネルギーソース項をエネルギー保存方程式に加えることで実現している。1コアあたり、12^3格子点を用い、100時間ステップまでの計算時間の平均速度のweak scalingを示している。図2より、TTXが様々な並列数において優れた並列性能を有していることが分かる。

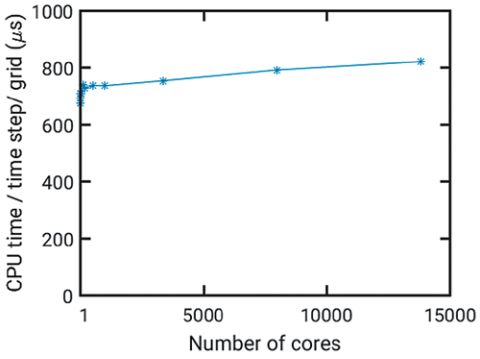


図2 TTXにおける典型的なweak scaling性能。

次に、MTSやCODACTを用いた際の計算性能及び精度について検証した。検証に用いた計算ケースは、ケース（i）：CODACTを用いない陽的RK、ケース（ii）：CODACTを用いず、時間積分はMTSで実施、ケース（iii）：CODACT及びMTSの両方を用いて計算を実施、の3ケースである。この3ケースで行った共通の計算系は、典型的な均一自着火燃焼場であり、温度の時間変化を図3に示す。ここで、ケース（i）については、化学反応時間スケールの制約から、時間刻み幅をΔt =1nsとし、その他のケースについては、Δt=5nsとした。図3に示すように、全てのケースにおいて、同一の温度変化が計算されているが、陽的RKを用いて計算を行ったケース（i）に関しては、圧力が上昇し、化学反応時間スケールがΔtより十分大きくなくなる5ms付近で他のケースの解からずれ始めることが分かる。従って、MTSを用いず陽的RKを用いる計算を正しく行う場合には、さらに小さなΔtを用いることが必要があり、これは計算コストに大きく影響する。

また、今回の系ではCODACTは計算精度には影響を与えないことも図3に示されている。図4は、計算物理時間における累積計算コストを3つのケースついて示したものである。明らかに、一番小さい時間刻み幅をもつケース（i）は、一番多くの計算コストを示し、MTSとCODACTの両方を用いたケースでは一番小さいコストを示す。これら2ケースのコスト差はおよそ7倍から8倍となっており、CODACT及びMTSによる計算速度向上が燃焼直接数値計算において有効であることが示された。

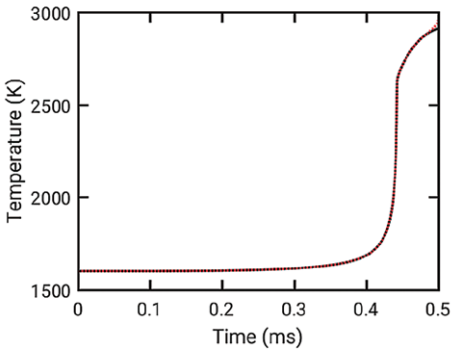


図3 ケース（i）-（iii）における温度時間変化の比較図。赤線：（i）、黒破線：（ii）、黒線：（iii）。

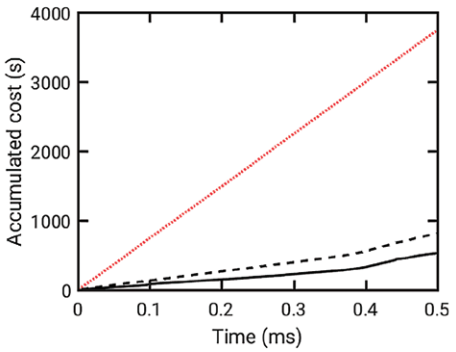


図4 ケース（i）-（iii）における累積計算コスト比較図。赤線：（i）、黒破線：（ii）、黒線：（iii）。

3.2 埋込境界法

開発した埋込境界法の精度を検証するために、非燃焼Taylor-Couette流の2次元計算を実施した。円形の内壁及び外壁は埋込境界を用いて記述し、異なる空間分解能の直接数値計算結果を厳密解と比較することで精度を検証した。図5に最大誤差・誤差の空間2乗平均（L_2）と格子点数の関係を示す。L_2の傾きは3次～4次となっており、従来の直接数値計算と同等の精度を有していることが分かる。従って、精度の面において、従来の2次精度である埋込境界法に比べ、本手法がより直接数値計算に適している。

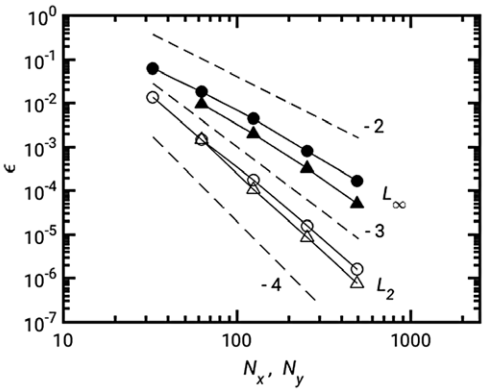


図5 層流Taylor-Couette直接数値計算から得られた誤差収束速度。

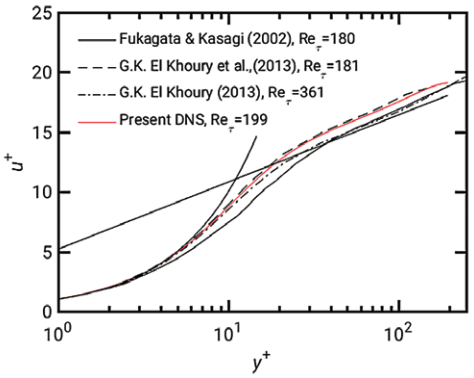


図7 円筒内乱流直接数値計算における平均主流方向速度分布。

非反応円筒管内乱流の直接数値計算は、任意の構造格子を用いて過去に数多く行われており、本研究では埋込境界を用いた非反応円筒管内乱流の直接数値計算をこれらのデータベースと比較し、埋込境界法が壁面乱流を正しく記述できるかについて検証した。初期の摩擦速度と円筒半径から定義されるレイノルズ数Re_τは200であり、計算領域の大きさと計算された流れ方向速度場を図6に示す。直接数値計算は、計算領域流れ方向長さと平均流速から計算されるmean-flow-through(τ_D)に対して40τ_D実施し、各統計量は25τ_D以降からサンプリングした。図7に、平均主流速度分布に関して、本直接数値計算結果と以前の研究において報告されている結果を示す。本計算結果は、同様のレイノルズ数のケースにおいて、過去の計算結果と非常に良い一致を示しており、埋込境界法を用いることで、デカルト格子状においても、円筒のような壁面形状と干渉する乱流場を考慮可能ということが示された。また、本計算系における計算コストは、東京大学Reedbush-Uにおいて1296コアを用いた場合において埋込境界を用いている場合には各ステップ1020msであり、これは埋込境界をオフにすると950msまで減少する。この計算速度差は非燃焼場において5%程度であり、本研究で開発した埋込境界による計算コスト増は、十分小さいと考えられる。

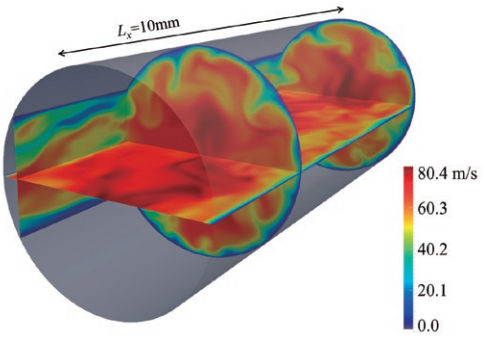


図6 円筒内乱流直接数値計算における計算領域と主流方向速度成分。

3.3 乱流燃焼の3次元直接数値計算

TTXを用いて実施した、典型的な乱流燃焼場の3次元直接数値計算結果を図8に示す。本記事にて述べた数値手法を従来のDNS手法と組み合わせることで、今後、乱流燃焼DNSの適用範囲が、今までのような比較的小さいスケール、シンプルな燃焼場から、より実際の燃焼機器に近い、大きいスケールを持つ複雑流れ中に存在する燃焼場まで広がると考えられる。

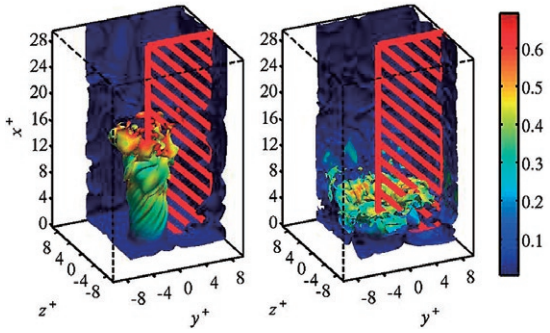


図8 TTXを用いて実施した典型的な旋回乱流中に存在する燃焼DNS結果。
等値面：熱発生率、色：無次元温度。
左：低スワール数（0.6）ケース、
右：高スワール数（1.2）ケース。

まとめ 4

反応速度計算に関する計算性能向上を実現するMTS及びCODACT、より柔軟な境界条件を記述できる埋込境界法を組み込んだ直接数値計算コードTTXについて述べた。MTSとCODACTの利用により、7倍から8倍の計算速度向上が達成できた。また、開発された埋込境界法は、一般的な直接数値計算と同等の精度を有することが示された。

謝辞

本研究の一部は科学研究費補助金・若手研究（B）課題番号16K18026から支援を頂いた。記して謝意を表す。

参考文献

[1] U.S. Energy Information Administration: International energy outlook 2010, DOE/EIA-0484 (2010)

[2] M. Tanahashi, M. Fujimura, and T. Miyauchi: Coherent fine-scale eddies in turbulent premixed flames, Proc. Combust. Inst., Vol. 28, pp. 529–535 (2000)

[3] Y. Minamoto, N. Fukushima, M. Tanahashi, T. Miyauchi, T. D. Dunstan, and N. Swaminathan: Effect of flow-geometry on turbulence-scalar interaction in premixed flames. Phys. Fluids, Vol. 23, 125107 (2011)

[4] M. Shimura, K. Yamawaki, N. Fukushima, Y. S. Shim, Y. Nada, M. Tanahashi, and T. Miyauchi: Flame and eddy structures in hydrogenair turbulent jet premixed flame, J. Turbulence, Vol. 13, pp. 1–17 (2010)

[5] B. Yenerdag, N. Fukushima, M. Shimura, M. Tanahashi, and T. Miyauchi: Turbulence-flame interaction and fractal characteristics of H2-air premixed flame under pressure rising condition. Proc. Combust. Inst., Vol. 35, pp.1277–1285 (2015)

[6] Sussman, P. Smereka and S. Osher: A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow, J. Comp. Phys., Vol. 114, pp.146-159 (1994)

[7] X. Gou, W. Sun, Z. Chen and Y. Ju: A dynamic multi-timescale method for combustion modeling with detailed and reduced chemical kinetic mechanisms, Combust. Flame, Vol. 157, pp. 1111-1121 (2010)

[8] W. Sun, X. Gou, H.A. El-Asrag, Z. Chen and Y. Ju: Multi-timescale and correlated dynamic adaptive chemistry modeling of ignition and flame propagation using a real jet fuel surrogate model, Combust. Flame, Vol. 162, pp. 1530-1539 (2015)

[9] W. Sun, Z. Chen, X. Gou and Y. Ju: A path flux analysis method for the reduction of detailed chemical kinetic mechanisms, Combust. Flame, Vol. 157, pp. 1298-1307 (2010)

[10] W. Sun and Y. Ju: Multi-timescale and Correlated Dynamic Adaptive Chemistry and Transport Modeling of Flames in n-Heptane/Air Mixtures, Proc. 53rd AIAA Aerospace Sciences Meeting (2015).

[11] P. Parnaudeau, E. Lamballais, D. Heitz, J. H. Silverstrini: Combination of the Immersed Boundary Method with Compact Schemes for DNS of Flows in Complex Geometry, ERCOFTAC seriese, Direct and Large-Eddy Simulation V, Vol. 9, pp. 581-590.

GPUを用いた高性能並列AMRのためのハイレベルフレームワーク

モハメド ワヒブ*　丸山 直也*　青木 尊之**
*理化学研究所計算科学研究機構　**東京工業大学学術国際情報センター

適応的格子細分化法（Adaptive Mesh Refinement）は必要な部分のみ格子を細分化することによって計算量を削減可能にするものである。しかしながら粒度の異なる階層的な格子の管理を実行アーキテクチャ向け最適化する必要があり、実際に高効率なAMRを実現することは困難である。この問題は、特にGPUを用いたスーパーコンピュータのような複雑なシステムにおいて顕著である。
本稿では、逐次実行モデル向けに記述された簡潔なAMRコードからGPUスーパーコンピュータ向けに最適化されたコードを自動的に生成するハイレベルフレームワークを紹介する。3つのAMRアプリケーションを本フレームワークによって実装したところ、人手による実装と同程度の性能を達成できることがわかった。また、TSUBAME2.5において最大3640台を用いた規模まで良好なウィークスケーリングを達成できることがわかった。

1 はじめに

科学技術計算では偏微分方程式の解法として一様格子上の差分法が多く用いられている。一様格子では計算対象空間内において必要とされる最も細かな粒度に全体を揃える必要があるが、一部のみ粒度を細かくする必要がある場合には適合格子細分化法（Adaptive Mesh Refinement）が有効である。AMRにおいては、初期状態としては比較的粗い格子とし、部分空間毎に動的に格子を細分化する。必要な箇所以外は粗い格子を用いることにより、理論的には全体の計算量や必要メモリ量を大きく削減可能である。これは将来に向けてより大規模な問題を解くために大変有望な特質であり、今後ますますその重要性が高まることが予想される。

一方で、空間が異なる粒度で表現されるAMRでは一様格子に比べて複雑な処理が必要とされる点が問題であり、特にGPU等のアクセラレータを用いた並列システムにおいて顕著である。AMRでは異なる粒度の格子を管理する必要があり、また実行時に変わりうるため、それらの処理に要する時間がオーバーヘッドとなる。これは特にメモリ空間が分断されたマルチGPUシステムにおいては煩雑な処理となり、データの移動を適切に最適化する必要がある。今日ではGPUの計算科学における有効性は多く示されてきているが、これらの技術的困難さからGPUを用いた並列AMRをサポートしたソフトウェアは希であり、実際に高い性能を達成しているフレームワークは我々の知る限り存在しない。これはCPU向けにはFLASHのような広く用いられているAMRフレームワークが存在することと対照的である[1]。

本稿では、大規模GPUシステムに対応したスケーラブルなAMRフレームワークであるDainoを紹介する[3]。これは高い抽象度を有したプログラミングモデルを実現したものであり、ユーザコードから自動的に並列GPUシステム向けAMRコードを生成する。以下、その概要および性能について述べる。

2 背景

構造AMRでは直交格子を領域内状態に応じてその粒度すなわち格子間隔を変化させる。粗密の格子は通常は入れ子された格子として表現され、全体の問題空間は階層的に管理される。このような階層的な格子の実装手段としては木構造の利用があげられ、木構造AMRでは空間を一定サイズのブロックに区切り、ブロック毎に必要に応じて粒度の調整を行う。

3次元木構造AMRでは八分木を構成し、木の葉は一定要素数の直方体（octant）を持ち、木の根は問題空間全体を表す。それぞれの部分領域毎の必要格子粒度に応じてoctantは再帰的に細分化され、木構造の親子関係が部分領域とその細分化された格子の関係に相当する。また、典型的には近隣の格子間では一定の細分化レベルに対する制約が与えられ、2:1制約では急激な粒度の変化を抑制するために隣接する格子間ではその細分化レベル、すなわち根からの深さの差が高々1であるとされる。八分木では深さ優先順序による葉の走査は空間充填曲線の一つであるモートン曲線に相当する特徴を持つ[4]。これは問題を分散メモリ上で並列に計算する際に有効な特質であり、葉octantをモートンIDによってソートし、PE数に均一に分割することで、問題空間全体を一様に分割することができる。また、空間充填曲線IDによってソートされているため、各PEには近接のoctantが割り当てられ、PE間の通信を削減することにも有効である。図1に空間を3つのPEによって分割する場合の例を示す。

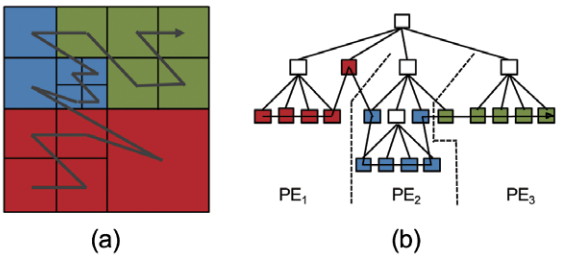


図1 （a）はAMRによる格子細分化例およびそのモートンカーブ。（b）はその木による表現。色の違いは3つのPEに分割した状態を表す。

高性能 AMR のための ハイレベルフレームワーク 3

我々はAMRアプリケーションの開発において高い生産性を実現するフレームワークを設計する。本フレームワークは高い抽象度を有したプログラミングモデルを基本とし、それによりユーザへの負担を最小限にとどめ、同時に高効率かつスケーラブルなAMRコードを自動的に生成する。これは独自コンパイラおよびランタイムソフトウェアから構成され、指示文によって一様格子モデルからの宣言的な拡張によるプログラミングによりAMRの記述が可能である。またこれらの指示文を用いたプログラミングでは特定の機種には依存せず、GPU等の特定機種向けコード生成および最適化はコンパイラによって自動化される。木構造格子の生成および管理などの実行時に必要な定型処理はランタイムソフトウェアによってAPIとして提供される。

3.1 プログラミングモデル

DainoフレームワークではAMRプログラミング用指示文を提供する。ユーザは通常のC言語によって記述された逐次一様格子プログラムにAMR化に必要な格子をあらわす配列の指定やステンシルループの指定を行う指示文を追加することでアプリケーションをAMR化することができる。また、これらの指示文は本フレームワークのコンパイラのみによって解釈されるものであり、通常のコンパイラからは無視され、その場合は一様格子アプリケーションとして正常に実行可能である。図2にステンシルカーネルの指示文の例を示す。指示文の詳細については文献[3]を参照されたい。

```
#pragma dno kernel
void func(float ***a, float ***b, ..) {
#pragma dno data domName(i, j, k)
a, b;
#pragma dno timeloop
for(int t; t< TIME_MAX;t++) {
for(int i; i<NX; i++)
for(int j; i<NY; j++) {
  ... // comput. not related to a and b
  for(int k; k<NZ; k++) {
     a[i][j][k] = c*(b[i-1][j][k] +
     b[i+1][j][k] + b[i][j][k]) +
     b[i][j+1][k] + b[i][j-1][k]);
  } } } }
```

図2 一様格子ステンシルへのDaino指示文の適用例

3.2 最適化

DainoフレームワークではGPUクラスタを用いる際、ステンシルおよび格子の細分化処理をGPU上で実行し、八分木や負荷分散処理はCPU上で実行する。GPU向けには基本的なステンシル向け最適化[5]に加えて、適切なデータ構造の選択、ユーザ管理キャッシュの利用が格子の細分化等の処理に対してコンパイラによって自動的に適用される。また、GPUを効率良く利用することに加えて特に多数のGPUを用いる際はGPU間の通信や負荷分散が重要になる。本フレームワークではコンパイラとランタイムが協調することによって自動的に通信処理の最適化やGPUあたりの割当octant数の均一化が行われ、ユーザは分散メモリ環境で実行する際に必要とされる最適化を意識する必要がない。

3.3 実装

本フレームワークのコンパイラとランタイムソフトウェアの実装について概要を述べる（図3）。コンパイラはLLVMコンパイラの拡張として実装されており、LLVMが提供する機能を用いてGPU向けコード生成も実現している[6]。具体的にはまずLLVMコンパイラのフロントエンドを拡張し、指示文が付加されたユーザプログラムの解析および最適化を行い、LLVMの中間表現（IR）に変換する。次に、IR上の変換処理としてAMRに必要な木構造の管理等のランタイムAPIの呼び出しを適切な箇所に挿入する。最後にLLVMによってIRから実行バイナリコードが生成される。LLVMではIRからNVIDIA GPU向けコード生成が可能であり、我々のフレームワークによる拡張は主にIRを生成する部分までに限定されている。

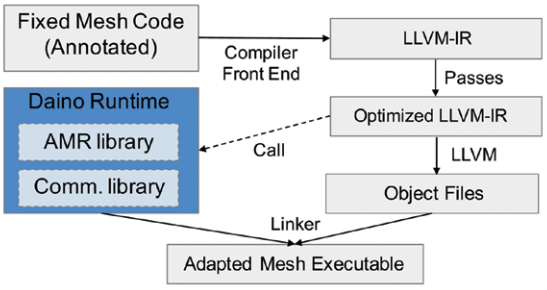


図3　フレームワーク実装概要

ランタイムソフトウェアは2つのライブラリから構成される。まず1つ目はAMRの八分木格子の生成等の基本的な管理をAPIとして提供する。2つ目のライブラリは分散メモリ空間向け通信ライブラリであり、GPU間の袖領域交換等の通信処理をMPIによって実装したライブラリである。このような通信処理は人手によるプログラミングでは煩雑な記述が必要とされる場合が多いが、本ライブラリに必要な処理をまとめることによってコンパイラによるコード生成が大幅に簡略化される。

4　評価

提案フレームワークによるGPUクラスタでの性能を3つのアプリケーションを用いて評価する。評価としては速度およびスケーラビリティを人手によって記述されたバージョンと比較する。

4.1　評価アプリケーション

- Phase field simulation: 2元合金のデンドライト凝固計算[7].
- Hydrodynamics solver: directional splitting法を用いた2次精度双曲線型オイラー方程式の計算[8].
- Shallow-water model: ナビエストークス方程式による浅水モデルの計算[9].

4.1　結果

図4にウィークスケーリング評価結果を示す。図では一様格子、手動記述によるAMR、および本フレームワークによる自動AMRの性能を示した。本結果より、フェーズフィールド法ではTSUBAME2.5の3640台のGPUを用い一様格子に比べて1.7倍超の性能向上を達成できていることがわかる。本アプリケーションはゴードンベル賞を受賞した高い性能を達成しているものであり[7]、それに対してさらなる性能向上を達成できていることは重要な成果と言える。また、手動記述バージョンと比較しても同等のスケーラビリティを達成できていることがわかる。

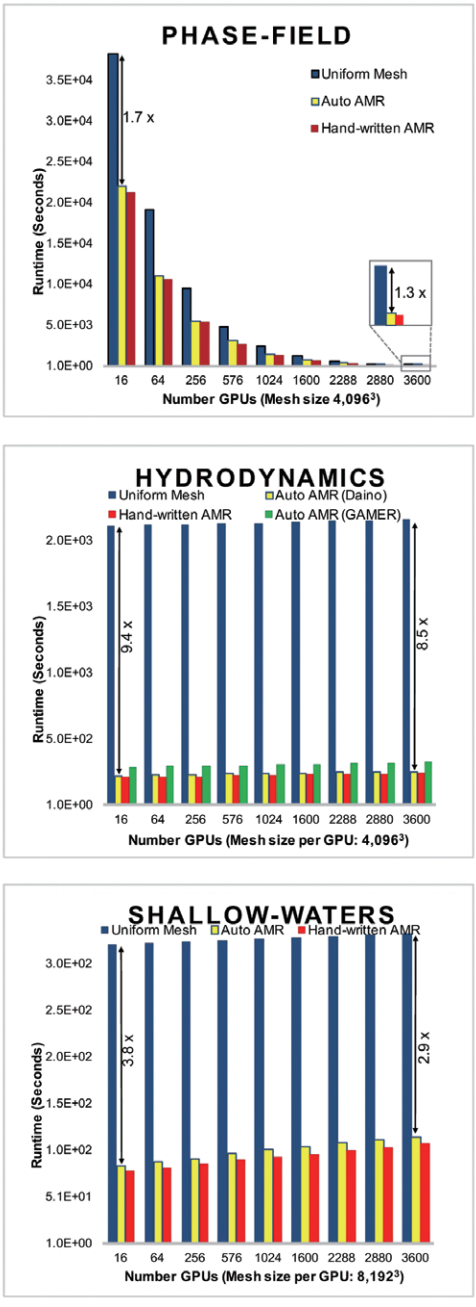


図4　ウィークスケーリング性能

図5にストロングスケーリングさせた場合の一様格子、人手によるAMR、自動AMRの結果を示す。結果からわかるようにDainoによる自動AMRは人手による記述と同程度の性能が達成できている。また、利用GPU数が増えるにつれて一様格子に対する性能向上率が減少しているが、これはoctreeの管理などのオーバーヘッドの影響が相対的に大きくなるためである。

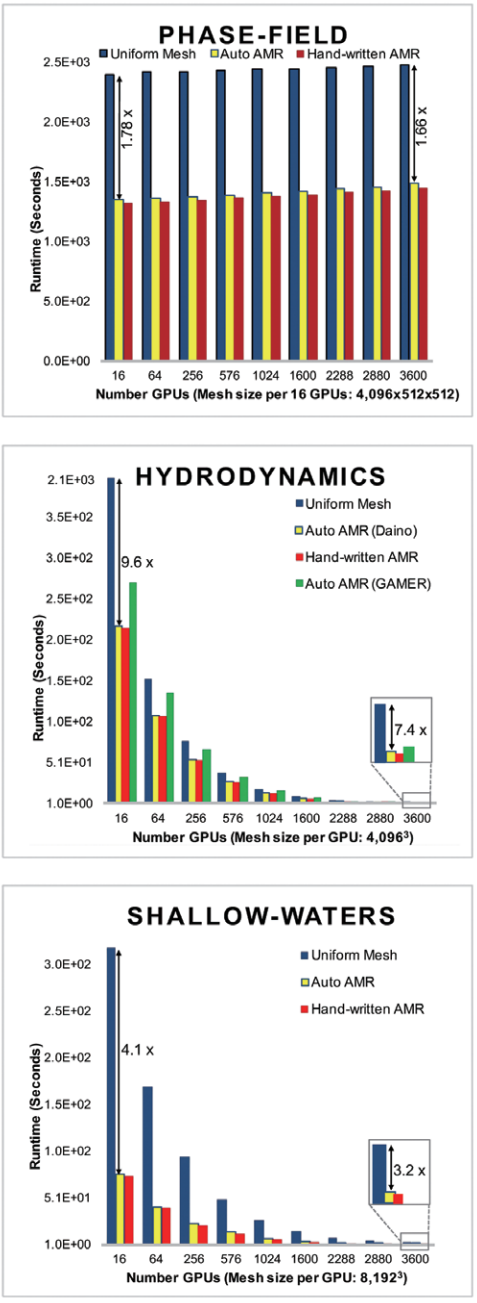


図5 ストロングスケーリング性能

おわりに 5

本稿では構造AMR向けハイレベルフレームワークを紹介した。本フレームワークでは実行アーキテクチャ向けに自動的にコードを生成し、ユーザは逐次実行モデルによるコードからの宣言的な指示文のみよってAMR拡張が可能になる。これによってGPUクラスタ等の高度なプログラミングが必要とされる環境においてもAMRを簡便に利用可能になり、実際にその有効性および性能をTSUBAMEスーパーコンピュータ上で3つのアプリケーションを用いて定量的に示した。GPUを数千台規模の規模まで自動的に利用可能にした研究は我々の知る限り本研究が初めてである。今後は境界条件や格子細分化条件のより柔軟な設定を可能にすること等、本フレームワークの拡張および性能最適化を実施していく予定である。

謝辞

本研究はJST CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」領域、JSPS科研費JP26220002、および学際大規模情報基盤共同利用・共同研究拠点の支援による。

References

[1] A. Dubey, K. Antypas, M. K. Ganapathy, L. B. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide, “Extensible Component-based Architecture for FLASH, a Massively Parallel, Multiphysics Simulation Code,” Parallel Comp., vol. 35, no. 10-11, pp. 512–522, (2009)

[2] M. Wahib, N. Maruyama, Data-centric GPU-based Adaptive Mesh Refinement, IA^3 2015, Workshop on Irregular Applications: Architectures and Algorithms, co-located with SC’15, Austin, TX (2015)

[3] M. Wahib, N. Maruyama, T. Aoki, Daino: A High-level Framework for Parallel and Efficient AMR on GPUs, SC'16, ACM/IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2016)

[4] H. Tropf and H. Herzog, “Multidimensional Range Search in Dynamically Balanced Trees,” Angewandte Informatik, 1981 (1981)

[5] N. Maruyama and T. Aoki, Optimizing Stencil Computations for NVIDIA Kepler GPUs, 1st International Workshop on High-Performance Stencil Computations HiStencils'14 (2014)

[6] http://www.llvm.org

[7] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka, "Peta-scale Phase-field Simulation for Dendritic Solidification on the TSUBAME 2.0 Super- computer," ser. SC '11 (2011)

[8] H.-Y. Schive, U.-H. Zhang, and T. Chiueh, "Directionally Unsplit Hydrodynamic Schemes with Hybrid MPI/OpenMP/GPU Parallelization in AMR," Int. J. High Perform. Comput. Appl., vol. 26, no. 4, pp. 367–377 (2012)

[9] M. Sætra, A. Brodtkorb, K. Lie, "Efficient GPU-implementation of Adaptive Mesh Refinement for the Shallow-Water Equations," J. Sci. Comput., vol. 63, no. 1, pp. 23–48 (2015)

クラウド的GPU利用を簡単にするツール：DS-CUDA

成見 哲
電気通信大学 情報・ネットワーク工学専攻

GPUの利用方法がゲームやHPC領域に広がり、GPUを備えたクラウドサービスも出始めている。
提案するDS-CUDAの場合はGPUでの計算機能（CUDA機能）だけをクラウド側にオフロードする。この方式では、ユーザーアプリケーションをクライアント側で自由に作り替えたりセンサー等の情報が使いやすいだけでなく、耐故障性機能を実現しやすいというメリットがある。
DS-CUDAの機能の概要の他、TSUBAMEの1,024GPUを使ってレプリカ交換分子動力学シミュレーションを880倍加速した例、オーバークロックしたGPUを使って耐故障性機能のオーバーヘッドを測定した例、タブレットを使ってエネルギー効率を測定した例などを報告する。

はじめに 1

高性能なGPU（Graphics Processing Unit）の恩恵は受けたいが、手持ちのコンピュータにGPU環境を整える手間やコストは最小限にしたいというユーザーが増え、様々なサービスやツールが生まれてきている。Amazon EC2[1]では16GPUまでの仮想マシンをクラウド上に構築しGPGPU（General-Purpose computing on GPUs）アプリケーションを手軽に走らせることが出来る。Nvidia GRID[2]では、クラウド側のGPUで計算するだけでなく高精細なレンダリング結果をクライアント側に低遅延で転送することで、貧弱なGPUしか持たない端末でも高機能なGUIアプリケーションを実行することが出来る。

本稿では、GPU上でCUDA（Compute Unified Device Architecture）を使って計算するだけでなく、可視化も同時に行うようなアプリケーションを想定する（例えばInteractive Molecular Dynamics Simulation[3]）。今後タブレットのような多くのセンサーを持つ端末を使ってインタラクティブなシミュレーションと描画を行うことへの要求が高まっていくと考えられるため、クラウド側の高機能GPUをモバイル端末等のクライアント側からシームレスに使う手法が望まれる。

DS-CUDA[4,5]はクライアント側に実GPUを装着しなくてもクラウド側のGPUを見かけ上使えるツールであり、耐故障性機能もあることが特徴である。以下ではこれまでの報告から抜粋してその概要を述べる。

クライアント側とクラウド側の分担 2

何でも一つのコンピュータで実現するのではなく分担する方がコストや電力的に効率が良いことも多いが、クライアント側とクラウド側の分担方法は以下のようないくつかの方式が考えられる（図1）。

A）ほぼ全ての計算やレンダリングをクラウド側で行う。
B）レンダリング部分だけクラウド側で行う。
C）CUDAの計算だけクラウド側で行う。

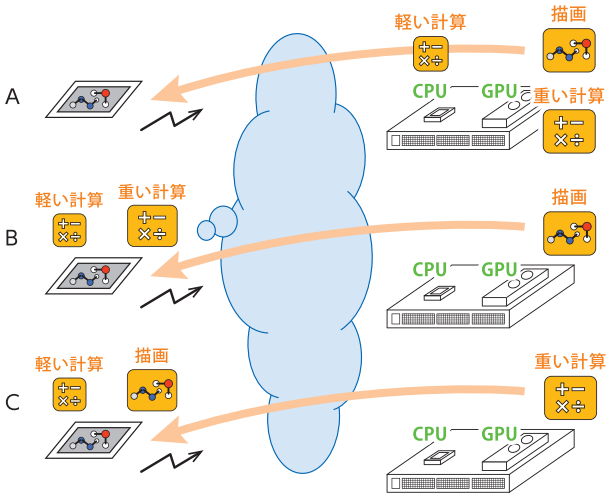


図1 クライアント側とクラウド側の分担

A）の場合、クライアント側はゼロクライアント端末のように振る舞う。つまり、クライアント側のキーボードやタッチなどの入力をクラウド側に送り、逆にクラウド側の画面や音などの出力だけがクライアント側に転送される。しかし、Nvidia GRIDのようなサービスでは通常ユーザーが自分のアプリケーションをコンパイルして実行することは想定しておらず、クラウド側でサポートしないアプリケーションを走らせることは基本的に出

来ない。逆にAmazon EC2のようなサービスの場合、自分で環境構築が出来るためそれ程走らせたいアプリケーションに制約はないが、クラウド側からの画面転送がボトルネックとなって実用的ではないことも多い。また、どちらの場合でも、例えばクライアント側の加速度センサー入力を反映させたいなら何らかの特別な仕組みが必要である。

B）の場合例えばクライアント側のレンダリングAPI（OpenGL等）呼び出しをフックしネットワーク上のクラウド側に仕事を転送する[6]。しかしCUDAが使えないためHPCアプリケーションには向いていない。

C）の場合、画面描画や加速度センサー等の処理はクライアント側で行う。これによりユーザーは自分のアプリケーションをいつもの環境でコンパイルすることが出来、クラウド利用のためにソースコードを変更する必要もない。DS-CUDAやrCUDA[7]はこのような目的での使用に適している。CUDAのAPIだけはツールでフックされクラウド側で処理されるため、高性能なGPUの恩恵を受けられる。

DS-CUDA とその他のツールとの比較 3

表1はいくつかのツールを比較している。C）方式のDS-CUDAとrCUDAはクライアント側でソフトウェアが動くためクライアント独自の機能を実現するのは簡単である。クラウド側にはCUDA APIで呼ばれた引数だけの情報が伝わるため、ファイルなどの可読な情報がクライアントから漏れにくいというメリットもある。しかし、GPUで計算した結果をそのまま使って描画する機能（OpenGL Interoperability機能など）は（今の所）使えない。

rCUDAの場合はダイナミックリンクライブラリの付け替えだけでクラウド側GPUを使えるためリコンパイルが不要であるが、DS-CUDAは必要である。逆にrCUDAはクライアント側でCUDAが必須であるが、DS-CUDAの場合はCUDAをサポートしない環境でCUDAを仮想的に動かすことも個別に対応すれば可能になる[8, 9]。Nvidia GRIDやAmazon EC2ではクライアント側にCUDA環境は不要でありどこからでもすぐに使えるというメリットはあるが、普段自分が手元で開発に使っている環境とは別にクラウド側にも開発環境を持つ必要がある。

また、DS-CUDAの特徴として、GPUが計算間違いをしたときやネットワークトラブルに対する耐故障性機能を有している。GPUの信頼性は、仮にGPGPU専用モデルであってもCPUよりも一般に低く[10]、信頼性のあるシミュレーションを行いたい場合耐故障性機能があると望ましい。特にクラウド環境ではネットワークの不安定さも信頼性に大きく影響する。

A）方式で耐故障性機能を実現するのは難しい。Amazon EC2のようなサービスであればユーザーが個別に耐故障性機能を追加することは可能だが大幅に手間が増える。C）方式であれば、ユーザー側には仮想的に信頼性のあるGPUを見せてCUDA APIを使ってもらい、実際にはツール側で信頼性機能を実装することが出来るため、ユーザーの負担がほぼない。

方式	Nvidia GRID	Amazon EC2	rCUDA	DS-CUDA
画面転送	高速	低速	不必要	不必要
加速度センサー等クライアント側特有の機能	サポートしない	サポートしない	動作する	動作する
CUDA と OpenGL、Direct3D との Interoperability 機能	動作する	動作する	サポートしない	サポートしない
CUDA のリコンパイル	想定していない	不要	不要	必要
クライアントに必要な環境	Nvidia GRID	特になし	CUDA	Ruby, (CUDA)
対故障性機能	なし	なし	なし	あり

表1 クラウド的GPU利用ツールの比較

DS-CUDA の概要 4

図2ではDS-CUDAでのコンパイルの手順、図3ではDS-CUDAシステムの構成例を示す。

まず、クライアント側のCUDAのソースコードはDS-CUDAコンパイラ（実体はRubyのスクリプト）でコンパイルされ、クライアント側とクラウド（サーバー）側の二つの実行ファイルを生成する（図2）。クライアント用には、まずソースコード中のCUDA APIをDS-CUDAラッパー関数に変換し、DS-CUDAライブラリとリンクすることで実行ファイルを生成する。サーバー用には、ソースコードのmain関数を削除し、nvccコンパイラを使って本当のCUDAライブラリとDS-CUDAサーバー本体の入ったライブラリをリンクすることで実行ファイルを生成する。

実際に実行する際は、サーバー側でDS-CUDAデーモンを起動しておく（図3）。クライアント用実行ファイルが起動されると、デーモンを通じてサーバー用実行ファイルがサーバー側に転送されてDS-CUDAサーバーとして起動する。クライアント用実行ファイルがGPU（仮想GPU）にアクセスしようとすると、ネットワークを経由してDS-CUDAサーバーを介してクラウド側の実GPUと通信する。

冗長計算を行う場合、クライアント側からのCUDA APIに対応するラッパー関数内で自動的に二つのサーバーに対し通信する。cudaMemcpy（DeviceToHost）によってGPUでの計算結果が回収される際に二つのサーバーからの内容を比較し、異なる場合は過去のAPI呼び出しを再実行する。マイグレーション機

能を実現するためには、定期的にGPU内でcudaMalloc()により確保したメモリ領域を全てクライアント側にバックアップする。GPUで回避不可能なエラーが起きたり、サーバーとのネットワーク接続が切れたりした場合には、バックアップした内容を新しいGPUに転送することで途中から計算を続けることが出来る。

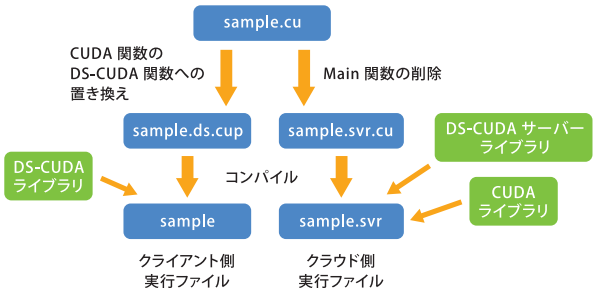


図2 DS-CUDAコンパイラでのコンパイル

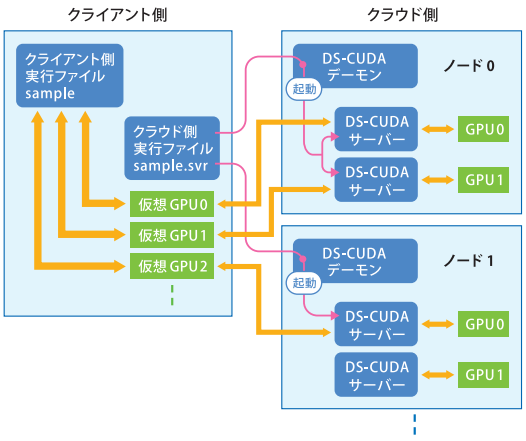


図3 DS-CUDAシステムの構成例

1スレッドからの多数の GPU の利用 5

CUDAで一つのノード内の複数のGPUを使う場合、CPUのコードを並列化しなくても使うことが出来る。しかし大量のGPUを使いたければMPIなどで並列化して各ノードで独立にGPUをコントロールするのが一般的である。ただし分散メモリで考える必要があるため、GPU初心者には難しくなる。DS-CUDAでは仮想的に多くのGPUがクライアント側に見えるため、複数GPUを使用するプログラミングは簡単になる。以下では極端な例としてシングルスレッドのプログラムから1,024GPUを使用した例[11]を紹介する。

この例では、14,366個からなるレプリカ交換分子動力学（MD）シミュレーションをTSUBAME 2.5の最大1,024GPUを使って実行した。レプリカ交換MDシミュレーションは、異なった温度を持つ複数のMDシミュレーションを並行して行い、一定期間（今回は100ステップ）毎に温度条件を交換することで最安定状態を効率よく探索する手法である。一つのレプリカには256または2,048粒子のアルゴン原子が含まれる。各レプリカ間での通信量や通信頻度が少ないためCPUの1スレッドから全GPUを制御してもそれ程負担にならない。

図4は計算速度のStrong Scalingをプロットしたものである。1,024GPU の時、2,048粒子の場合は880倍、256粒子の場合でも340倍の加速を実現している。

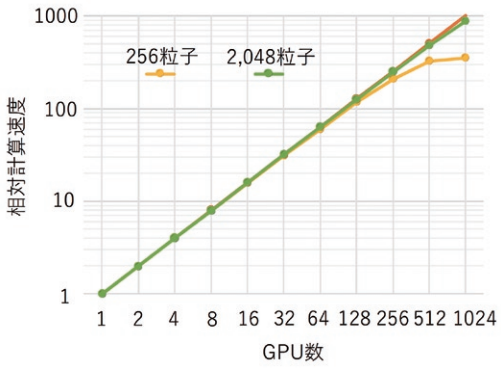


図4 GPU数に対する相対計算速度

図5は2,048粒子の場合についてGPU数を256, 1,024, 4,096に変えた時の計算と通信の割合を示したものである。4,096GPUでは実測していないため、1,024 GPUまでの測定値と合うようにモデル式を作って予測している。

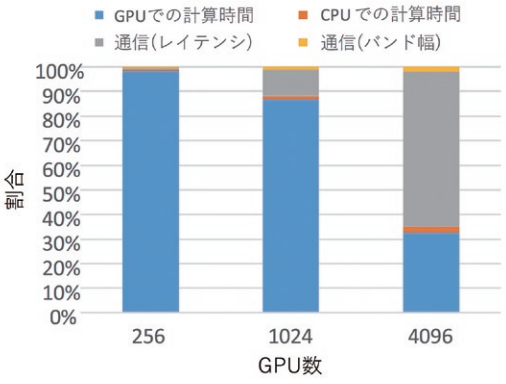


図5 計算時間と通信時間の割合

モデル式において全体の計算時間は、GPUでの計算時間、CPUでの計算時間、通信時間に分け、通信時間は更にレイテンシ成分とバンド幅成分に分けている。一回の通信量は平均する

と1.9 kbyte程度の小ささであるため、例えば1,024GPUの時はレイテンシに起因する通信時間が57.5 msecに対し、バンド幅に起因する成分が6.7 msecとなり、レイテンシが支配的であることが分かる。ただしGPUでの計算時間が471 msecであるため全体としてはまだGPU計算が支配的であることが分かる。しかし4,096 GPUになった場合は、GPUでの計算時間が118 msecに減るのに対しレイテンシ成分が230 msecと支配的になるため効率が出ないと予想される。これを改善したければクライアント側のコードをOpenMP等でマルチスレッド化することである程度回避可能と思われる。

耐故障性機能によるオーバーヘッド 6

耐故障性機能を有効にするには環境変数を設定するだけなのでユーザーには負担は少ないが、性能的にはオーバーヘッドが存在する。特にマイグレーション機能を実現するには定期的にGPU側のメモリ空間をクライアント側にバックアップ（チェックポインティング）するため、適切な頻度に設定しないと性能低下が大きくなる。

図6はチェックポインティングの間隔を変えた時にどの程度計算時間がかかるかを相対的に表したものである[12]。前節と同様のアプリケーションを使っている。GPUはオーバークロックして900秒に一回程度計算間違いを起こした場合について抜き出し（×印）、プログラム的に900秒に一回計算間違いを起こすDS-CUDAサーバー（●印）と合わせてプロットしている。

チェックポインティング間隔が短ければバックアップにかかる時間が増大し、逆に長ければ次のバックアップに行く前に計算間違いをして再計算を繰り返すためやはり時間がかかる。この例だと100秒間隔程度が最適でオーバーヘッドは10%程であると分かった。実際のGPUはもっと信頼性があるため[10]、オーバーヘッドは少なくとも本実験よりは少なくなるはずである。

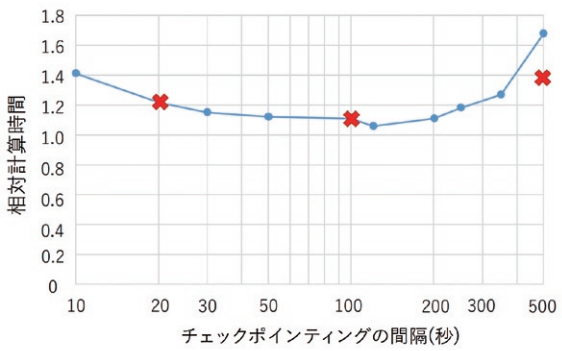


図6 耐故障性機能によるオーバーヘッド

タブレットからの利用 7

図7は、クライアント側にGPUを搭載しないタブレット端末を、クラウド側にGPUを搭載するノートPCを使って、無線接続でGPUを利用した例を示している。表2は、分子動力学に関してシミュレーションしながらレンダリングするアプリケーションを用いて、5,832粒子の時のエネルギー効率（Gflops/Watt）と描画速度（Frame/sec）を測定したものである[13,14]。消費電力測定ではタブレットとノートPCの両方を合計している。

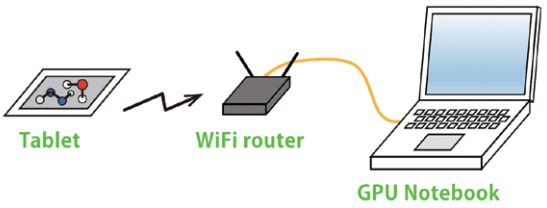


図7 タブレットでDS-CUDAを使う

ノートPC単体では12.8 Gflops/Wなのに対しDS-CUDAを利用することで9.0 Gflops/Wと若干下がっているが、Green500[15]の値と比べても十分によい値となっている。ただしこのflops値の算出では単精度演算をしていることや複雑な関数や割り算などの演算を多めにカウントしているためGreen500の数字と詳細に比較することは出来ない。

タブレット単体での計算速度や描画速度は遅いが、DS-CUDAを使うことで数十倍以上の計算速度、エネルギー効率、描画速度を実現している。特に描画速度が20 Frame/sec程度で十分滑らかであるため、インタラクティブなシミュレーションも可能になる。

また、表には書いていないが比較のためにNativeでCUDAが走るSHIELDタブレットで測定したところ、計算速度、描画速度、エネルギー効率どれもDS-CUDAのシステムの方が上回った。

システム	計算速度 (Gflops)	消費電力 (Watt)	エネルギー効率 (Gflops/Watt)	描画速度 (Frame/sec)
GPU Notebook	1,655	129	12.8	60.2
Tablet	4.4	17.5	0.25	0.16
Tablet + GPU Notebook	701	78	9.0	19.6

表2 計算速度とエネルギー効率

おわりに 8

多数のGPUを手軽に使うためのツールであるDS-CUDAに関して、可視化やクラウド的な観点から有効性を議論した。

DS-CUDAのようなアイデアを使えば、いつも使っている手元のクライアント端末から手軽に大量のGPUを使うことが出来る。タブレットの傾きセンサーやタッチ機能を有効に使いつつ計算＆可視化を高速化することが出来る。また、信頼性のある計算を行いたい時に手軽に耐故障性機能を実現出来る。

エネルギー効率の面から考えると、クライアント側だけでモバイル用のGPUを使ったり、クラウド側だけで本格的なGPUを使って計算するのがよいかもしれないが、それでは十分な計算速度や描画速度を得られないことがある。DS-CUDAのようなアイデアを使えば、計算速度、描画速度、エネルギー効率のどれもバランスよく優れたシステムを構築できる可能性がある。

近年のタブレットの普及状況を見ると、次世代のシミュレーションソフトウェアにおいては計算中に結果を確認出来るインタラクティブ性が求められると考えられるが、今回のようなクラウド的なアイデアは有効な解決手段ではないかと考えている。

謝　辞

本研究の一部は、科学技術振興事業団JSTの戦略的基礎研究推進事業CRESTにおける研究領域「ポストペタスケール高性能計算に資するシステムソフトウェアの創出」の支援により行った。一部の計算は東京工業大学のスーパーコンピュータTSUBAME 2.5を用いて行った。DS-CUDAの開発や評価に関して、株式会社K&F Computing Researchの川井敦氏、千葉大学の老川稔氏、電気通信大学のEdgar Josafat Martinez-Noriega氏、慶應義塾大学の野村昴太郎氏、泰岡顕治氏に深謝致します。

参考文献

[1] Amazon EC2: https://aws.amazon.com/jp/ec2/instance-types/p2/ (Accessed in Jan. 2017)

[2] Nvidia GRID: http://www.nvidia.com/object/grid-technology.html (Accessed in Jan. 2017)

[3] N. Luehr, A. G. Jin, and T. J. Mart´ınez, “Ab Initio Interactive Molecular Dynamics on Graphical Processing Units (GPUs),” Journal of chemical theory and computation, Vol. 11, No. 10, pp. 4536–4544, (2015)

[4] A. Kawai, K. Yasuoka, K. Yoshikawa, and T. Narumi, “Distributed-Shared CUDA: Virtualization of large-scale GPU systems for programmability and reliability,” Future Computing 2012, Nice, France, (2012)

[5] DS-CUDA: http://narumi.cs.uec.ac.jp/dscuda/ (Accessed in Jan. 2017)

[6] S. Shi, C.H. Hsu, “A Survey of Interactive Remote Rendering Systems,” ACM Computing Surveys (CSUR), Vol. 47, No. 4, Article 57, pp. 1-29, (2015)

[7] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti, “Performance of CUDA virtualized remote GPUs in high performance clusters,” International Conference on Parallel Processing, IEEE, pp. 365–374, (2011)

[8] 嶋田貴行，“Windows マシンに仮想CUDA環境を与えるPluginGPU Boxの開発，” 電気通信大学情報・通信工学科卒業研究，(2014)

[9] E. J. Martinez-Noriega, “Running CUDA on Android Through GPU Virtualization,” GPU Technology Conference, San Jose, P4160, (2014)

[10] I. S. Haque, V. S. Pande, “Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rate in GPGPU,” 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp.691-696, (2010)

[11] 老川稔，野村昴太郎，泰岡顕治，成見哲，“ 1,024GPU を使用したレプリカ交換分子動力学シミュレーションの並列化，” 情報処理学会 ACS 論文誌，Vol. 7, No. 4, pp. 1-14, (2014)

[12] 成見哲，老川稔，Martinez-Noriega EDGAR JOSAFAT, 泰岡顕治，“マイグレーション機能を備えたGPU仮想化ツールDS-CUDA，” 第153回ハイパフォーマンスコンピューティング研究発表会，2016-HPC-153(17), pp.1-6, (2016)

[13] E. J. Martinez-Noriega, “High Performance Computing on Mobile Devices through Distributed Shared CUDA,” GPU Technology Conference, San Jose, S5290, (2015)

[14] E. Martinez-Noriega, S. Yazaki, T. Narumi, “Performance Evaluation of Remote CUDA Offloading on Mobile Devices for Energy Efficient Systems,” submitted

[15] Green500 List: https://www.top500.org/green500/ (Accessed in Jan. 2017)

● TSUBAME e-Science Journal vol.15
2017 年 3 月 8 日　東京工業大学 学術国際情報センター発行 ©
ISSN 2185-6028
デザイン・レイアウト：キックアンドパンチ
編集：　TSUBAME e-Science Journal 編集室
　　青木尊之　渡邊寿雄　佐々木淳　板倉有希
住所：　〒 152-8550 東京都目黒区大岡山 2-12-1-E2-6
電話：　03-5734-2085　FAX：03-5734-3198
E-mail：tsubame_j@sim.gsic.titech.ac.jp
URL：　http://www.gsic.titech.ac.jp/

TSUBAME

vol. 15

TSUBAME 共同利用サービス

**『みんなのスパコン』TSUBAME共同利用サービスは、
ピーク性能5.7PFlops、18000CPUコア、4300GPU搭載
世界トップクラスの東京工業大学のスパコンTSUBAME2.5を
東京工業大学以外の皆さまにもご利用いただくための制度です。**

TSUBAME 共同利用サービスの利用区分とカテゴリ

TSUBAME 共同利用サービスには「学術利用」と「産業利用」の利用区分があります。
「学術利用」は、HPCI（革新的ハイパフォーマンスコンピューティングインフラ）、
JHPCN（学際大規模情報基盤共同利用・共同研究拠点）の採択により無償でご利用になれる制度と、
東京工業大学学術国際情報センターが実施する有償の制度がご利用になれます。
「産業利用」は、HPCIの採択により無償でご利用になれる制度と、
東京工業大学学術国際情報センターが実施する有償の制度があり、
有償の制度には利用目的や利用成果を非公開とする「成果非公開」のカテゴリもあります。

TSUBAME 共同利用サービスの利用料金

TSUBAMEにおける計算資源は口数を課金単位としております。1口は3000ノード時間積で、
1計算ノード（12CPUコア、3GPU、58GBメモリ搭載）を3000時間、
あるいは300計算ノードを10時間というように、ご利用の用途に合わせ自由にご利用になれます。
TSUBAME 共同利用サービスの利用区分・カテゴリ・利用料金を下表に示します。

<table>
<tr><th>利用区分</th><th>利用者</th><th colspan="2">制度</th><th>募集期間</th><th>申請および審査</th><th>成果</th><th>料金（税別）</th></tr>
<tr><td rowspan="3">学術利用</td><td rowspan="3">他大学
または
研究機関
等</td><td colspan="2">HPCI</td><td>年1回
10月頃</td><td>HPCI 運用事務局
（高度情報科学技術研究機構）</td><td>公開</td><td>無償</td></tr>
<tr><td colspan="2">JHPCN</td><td>年1回
1月頃</td><td>JHPCN 拠点事務局
（東京大学 情報基盤センター）</td><td>公開</td><td>無償</td></tr>
<tr><td colspan="2">TSUBAME 学術利用</td><td>随時
募集中</td><td>東京工業大学
学術国際情報センター</td><td>公開</td><td>1口 120,000円</td></tr>
<tr><td rowspan="5">産業利用</td><td rowspan="5">民間企業</td><td rowspan="2">HPCI</td><td>実証利用</td><td>年1回
10月頃</td><td rowspan="2">HPCI 運用事務局
（高度情報科学技術研究機構）</td><td rowspan="2">公開</td><td rowspan="2">無償</td></tr>
<tr><td>トライアル・
ユース</td><td>随時
募集中</td></tr>
<tr><td>JHPCN</td><td>企業共同
研究課題</td><td>年1回
1月頃</td><td>JHPCN 拠点事務局
（東京大学 情報基盤センター）</td><td>公開</td><td>無償</td></tr>
<tr><td colspan="2" rowspan="2">TSUBAME 産業利用</td><td rowspan="2">随時
募集中</td><td rowspan="2">東京工業大学
学術国際情報センター</td><td>公開</td><td>1口 120,000円</td></tr>
<tr><td>非公開</td><td>1口 480,000円</td></tr>
</table>

TSUBAME3.0

東京工業大学 学術国際情報センターでは、理論演算性能12.15PFlops 世界トップレベルの性能を有する
TSUBAME3.0を、2017年夏での運用開始に向け現在構築中です。
2017年度にご購入いただいたTSUBAME2.5の計算資源口数で、TSUBAME3.0もご利用いただけます。
詳しくは、東京工業大学 学術国際情報センター 共同利用推進室にお問い合わせください。

お問い合わせ

●東京工業大学　学術国際情報センター　共同利用推進室
●e-mail　kyoyo@gsic.titech.ac.jp　Tel. 03-5734-2085　Fax. 03-5734-3198
詳しくは http://www.gsic.titech.ac.jp/tsubame/をご覧ください。

GSIC
Global Scientific Information and Computing Center