e-Science Journal
東京工業大学 学術国際情報センター
Vol. 5
SC'11 Special Issue

TSUBAME ESJ.

SC11
Seattle,WA
Connecting communities through HPC

SC'11 特集号

ゴードンベル賞
特別賞
奨励賞

Graph 500 ランキング　3位

テクニカル・ペーパー　3件

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

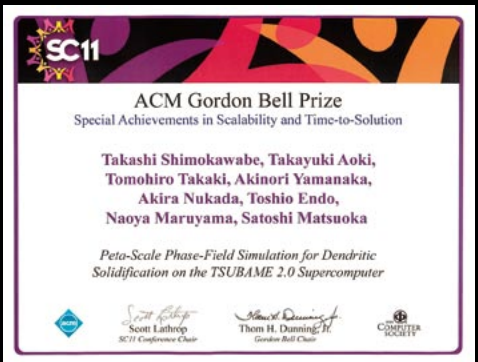
SC'11 特集号

03

ACM ゴードンベル賞・特別賞 & SC'11テクニカル・ペーパー

TSUBAME 2.0スパコンにおける樹枝状凝固成長のフェーズフィールド法を用いたペタスケール・シミュレーション

下川辺隆史　青木尊之　高木知弘　山中晃徳　額田彰
遠藤敏夫　丸山直也　松岡聡



09

ACM ゴードンベル賞・奨励賞

TSUBAME2における大規模生体流体力学シミュレーション

Massimo Bernaschi　Mauro Bisson　遠藤 敏夫
Massimiliano Fatica　松岡 聡　Simone Melchionna　Sauro Succi

SC11
Seattle, WA
Connecting communities through HPC

TSUBAME 2.0スパコンにおける樹枝状凝固成長のフェーズフィールド法を用いたペタスケール・シミュレーション

下川辺隆史* 青木尊之** 高木知弘*** 山中晃徳**** 額田彰**
遠藤敏夫** 丸山直也** 松岡聡**
* 東京工業大学大学院・総合理工学研究科 ** 東京工業大学学術国際情報センター
*** 京都工芸繊維大学大学院・工芸科学研究科 **** 東京工業大学大学院・理工学研究科

軽量・高強度な新材料の開発は低炭素社会の実現に向けて非常に重要である。材料強度を左右する材料のミクロな組織は凝固過程で決定されるが、機械的強度を判定するには数mmというマクロなスケールまでの大規模計算を必要とする。フェーズフィールド法というメソスケールのモデルを用い、TSUBAME 2.0で合金の樹枝状凝固の大規模シミュレーションを行った。CUDAを用いて有限差分法で離散化された時間発展方程式を解き、領域分割により並列化することでTSUBAME 2.0の4000 GPUを用いて、2.0 PFLOPS（単精度）という高い実行性能を得た。この成果に対し、2011年のACMゴードンベル賞・特別賞（本賞）が与えられた。

はじめに 1

軽量・高強度な新材料を開発することにより、物資を高効率（低燃費）で輸送することができ、低炭素社会の実現に大きく貢献する。金属材料の強度はミクロな材料組織に強く依存し、そのミクロ組織は凝固過程において形作られる（図1）。一方、現実に機械的強度を判定するには、数mmのマクロなスケールでの解析が必要となる。ミクロな凝固ダイナミクスを解明するために、非平衡統計力学から導出されるフェーズフィールド法[1]が近年注目されている。導出される方程式は時間空間の偏微分方程式になっていて、有限差分法や有限要素法などの格子法で解かれることが多い。フェーズフィールド法は複雑な非線形項を多く含み、ステンシル計算で解く場合には1格子点あたりの演算量が多くなる。さらに、非常に狭い界面領域に複数の格子を含む必要があるため、格子サイズを小さくする必要があり、格子数が膨大になり時間ステップを小さく取らなければならない。このため、CPUで計算すると時間がかかり過ぎるため、これまでは主に2次元計算による解析が行われてきた。

本研究では、フェーズフィールド法に基づいて二元合金の一方向凝固における樹枝状（デンドライト）組織の成長を複数GPUにより大規模計算する。CUDAを用いて有限差分法で離散化された時間発展方程式をプログラミングし、MPIを用い領域分割法で並列化することでTSUBAME 2.0のほぼ全ての計算資源を使った大規模計算を行う。フェーズフィールド法による超大規模計算を行った例は国内外で報告されておらず、本計算結果が材料科学分野に与えるインパクトは大きい。

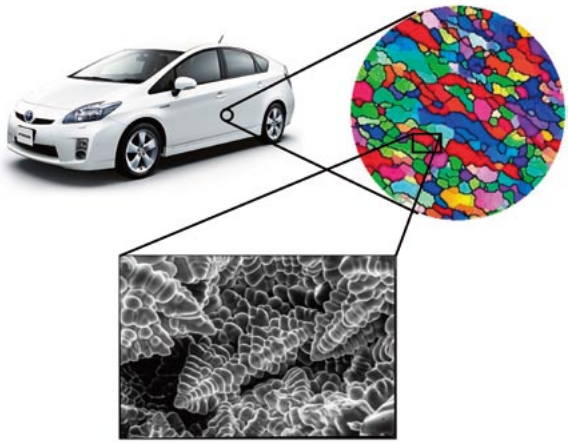


図1 材料組織のミクロな組織とデンドライト

フェーズフィールド法 2

フェーズフィールド法は非平衡統計物理学から導出され、分子スケールとマクロなスケールの中間のメソスケールの現象を記述することができる。秩序変数 ϕ（フェーズフィールド変数）を導入し、固相部分に $\phi=1$ を、液相部分に $\phi=0$ を設定する。界面を含む領域は ϕ が0から1へと急峻かつ滑らかに変化する拡散界面として扱われ、$\phi=0.5$ が界面位置となる。フェーズフィールド法では従来から使われている界面追跡法等の手法が不要となり、領域全体で同一の計算を行うことができる。

本研究で対象とする二元合金の樹枝状凝固成長では、フェーズフィールド方程式と溶質の拡散方程式を解く。フェーズフィールドに対しては、界面エネルギーの異方性を考慮した次の方程式

$$\frac{\partial\phi}{\partial t}=M_\phi\left[\nabla\cdot(a^2\nabla\phi)+\frac{\partial}{\partial x}\left(a\frac{\partial a}{\partial\phi_x}|\nabla\phi|^2\right)+\frac{\partial}{\partial y}\left(a\frac{\partial a}{\partial\phi_y}|\nabla\phi|^2\right)+\frac{\partial}{\partial z}\left(a\frac{\partial a}{\partial\phi_z}|\nabla\phi|^2\right)-\Delta S\Delta T\frac{dp(\phi)}{d\phi}-W\frac{dq(\phi)}{d\phi}\right] \quad (1)$$

を解く。ここで a は勾配係数、M_ϕ はモビリティーであり、W はエネルギー障壁、ΔS は融解エントロピー、ΔT は過冷却度を表している。また、関数 $p(\phi)$ と $q(\phi)$ はそれぞれ $p(\phi)=\phi^3(10-15\phi+6\phi^2)$ と $q(\phi)=\phi^2(1-\phi)^2$ を用いている。

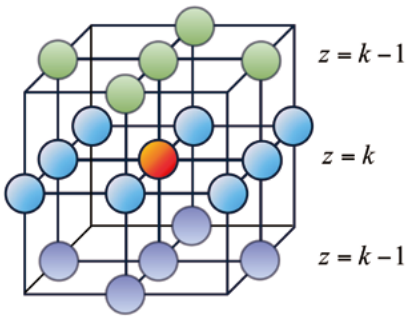
一方、溶質の拡散方程式は、次の方程式を用いる。

$$\frac{\partial c}{\partial t} = \nabla \cdot \left[D_S \phi \nabla c_S + D_L (1-\phi) \nabla c_L \right] \qquad (2)$$

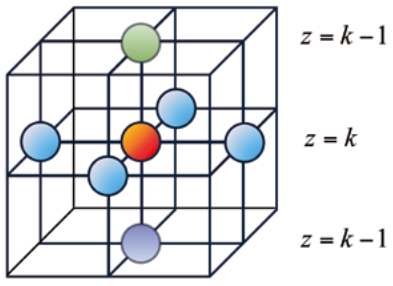
ここで、D_S と D_L はそれぞれ固相、液相の拡散係数を表す。c_S、c_L は $c = (1-\phi)c_L + \phi c_S$ を満たしている。

3 単一 GPU への実装

3次元直交格子上で（1）式と（2）式を2次精度有限差分法で離散化し1次精度の時間積分（オイラー法）を行う。CUDAを用いてGPUコンピューティングのプログラミングを行い、フェーズフィールド変数、濃度変数は全てビデオ・メモリ（CUDAではグローバルメモリと呼ばれる）上に確保し、計算の毎時間ステップでのPCI-Express Bus を介したGPUとCPU間のデータ通信を排除している。3次元空間の格子点（i, j, k）上のフェーズフィールド変数 ϕ（図2（a））と濃度変数 c（図2（b））を時間発展するために必要な近傍格子点上の変数の空間配置を示す。1タイムステップでは、それぞれの格子点での ϕ および c を計算するために、隣接19格子点のフェーズフィールド変数 ϕ と隣接7格子点の濃度変数 c をメモリから読み込み、計算結果をメモリへ2回書き込む。



（a） フェーズフィールド変数のステンシル



（b） 濃度変数のステンシル

図 2 時間発展に必要な隣接格子点アクセス

フェーズフィールド・モデルの計算は多くのメモリアクセスを伴うため、効率よく計算するためにはグローバルメモリへのアクセス回数を低減することが重要である。GPUコンピューティングでは、スレッドおよびブロックをどのように実際の計算に割り当てるかが高速化のキーポイントである。1つのGPUが担当する計算領域サイズをnx × ny × nzとし、図3のように64 × 4 × 32の複数のピースに分割する。GPUで実行されるカーネル関数を1つのスレッドブロックが1つのピースを担当するよう設定し、各ブロックで64 × 4 × 1のスレッドを実行させる。ある点（i, j, k）を割り当てられたスレッドは、z方向へマーチング（ループカウンタk）しながら（i, j, k0）から（i, j, k0+31）の32格子点を計算する。ここでk0は32の倍数で0、32、64…、nz - 32である。あるスレッドがk + 1番目の面を計算するのに必要なデータの一部はk番目の面を計算した際にそのスレッド自身によってすでに使われている。このようなデータはレジスタに一時的に保持し再利用することで、再びグローバルメモリへアクセスすることを回避することができる。本研究で用いるTSUBAME 2.0に搭載されたM2050 GPU はFermi コアのGPU であり、L1/L2キャッシュが利用できるため、これまでのような複数のスレッド間でデータを共有する目的（ソフトウェア・マネージド・キャッシュ）でシェアードメモリを使用する必要がなくなった。

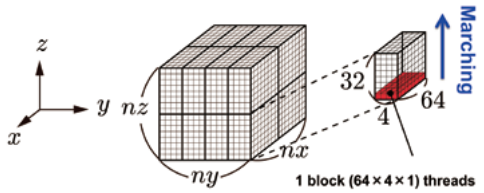


図 3 CUDA スレッドとブロックの割り当て

4 複数ノードに搭載された GPU による計算

大規模問題を高速に計算するには、複数のGPUを用いて計算することが必須となる。GPU内部での並列計算に加えて、上位でのGPU単位での並列化が必要となる。大規模並列計算では、3次元領域分割を行うことにより領域間のデータ通信量を最小にすることができるが、GPU計算の場合はx方向で分割した場合のy-z 断面のメモリアドレスが不連続となり、メモリアクセス性能の低下のオーバーヘッドの方が通信量の低減より大きくなる。ここでは、y方向、z方向に分割する2次元の領域分割法を用い、それぞれの分割領域の計算を各GPUへ割り当てる。複数CPU計算と同様に隣接するGPU間での境界領域のデータ交換が必要になるが、現状ではGPUは他のGPUのグローバルメモリ上のデータに直接アクセスすることができない。そこでGPU間のデータ転送はホスト側のメモリを経由し、次の3段階で構成される。（1）CUDAランタイムライブラリによるGPUからCPUへ

TSUBAME 2.0スパコンにおける樹枝状凝固成長のフェーズフィールド法を用いたペタスケール・シミュレーション

の転送、(2) MPIライブラリによるCPU間のデータ転送、(3) CUDAランタイムライブラリによるCPUからGPUへの転送を行う。

従来の複数ノードのCPU計算の場合と比較すると、GPUの演算性能が高いためGPU内での計算時間が短い。GPU間のデータ通信の時間が無視できず、大規模計算では大きなオーバーヘッドになる。GPU間のデータ通信の時間をGPU計算と如何にオーバーラップすることで隠ぺいするかが非常に重要となる。ここでは、(a) GPU-only method、(b) Hybrid-YZ method、(c) Hybrid-Y methodの3つの実装を行う。

(a)GPU-only method：比較のために計算と通信のオーバーラップを行わず、上記の通信の3ステップを順に行う。

(b) Hybrid-YZ method：1つのGPUが担当するサブ領域を、y方向の境界領域、z方向の境界領域、残りの中心領域に分割する。y、z方向の境界領域をCPUで計算し、中心領域をGPUで計算する。先行研究[2]では境界も中心領域もGPUで計算したが、ここでは境界領域をCPUで計算しGPU関数を分割することなく通信を隠蔽する。ただし、全計算領域サイズによってCPUによる通信と境界領域の計算にかかる時間が中心領域のGPU計算時間よりも長いことがあり、CPUがボトルネックとなることが考えられる。

(c) Hybrid-Y method：(b)と同じようにCPU計算も利用するが、z方向の境界領域(x-y断面)はカーネル関数を分割してGPUで計算する。境界領域を担当するGPUカーネル関数は、グローバルメモリの連続アクセスであるため計算効率がよく、また通信バッファへデータを詰め替える必要もない。

TSUBAME 2.0 の各ノードには3 GPUと2 CPU sockets (12 CPU cores) が搭載されている。このためCPUによる境界領域の計算に1 GPUあたり4CPU cores を割り当て、OpenMPを用いた並列計算をおこなっている。

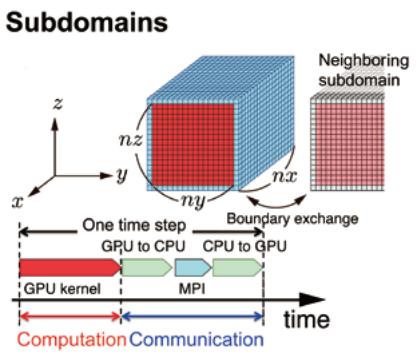


図 4 GPU-only method のダイアグラム

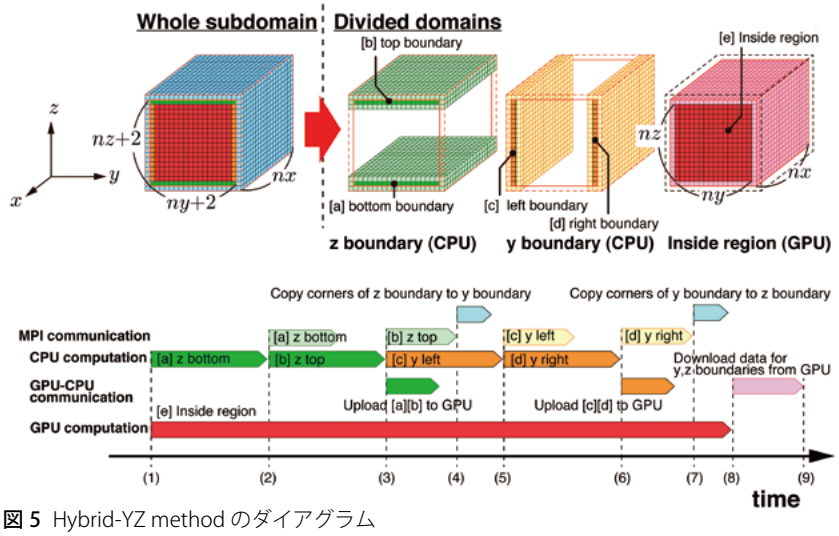


図 5 Hybrid-YZ method のダイアグラム

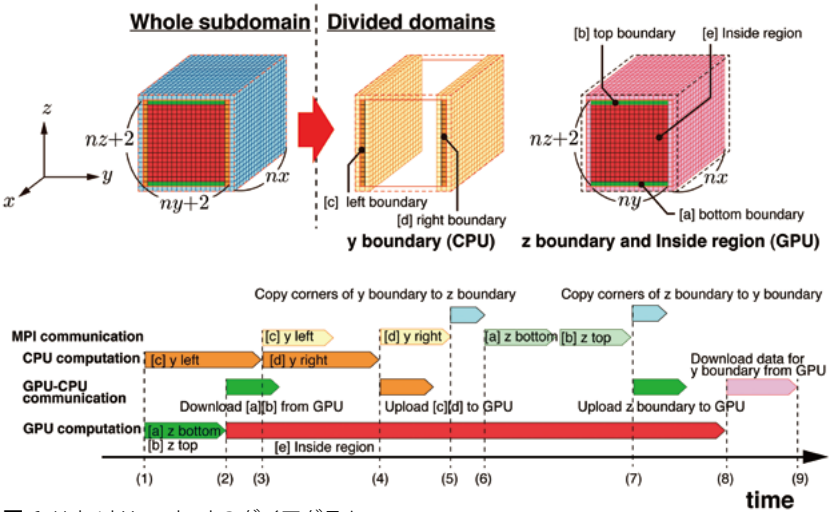


図 6 Hybrid-Y method のダイアグラム

大規模 GPU 計算の実行性能 5

東京工業大学・学術国際情報センターのGPUスパコンであるTSUBAME 2.0の複数GPU(M2050)を用い(a)GPU-only method、(b) Hybrid-YZ method、(c) Hybrid-Y methodの3つの実装の実行性能について述べる。ここでは、Al-Si合金の凝固成長の計算を行った。図7(a)は、大阪大学の安田秀幸教授らが大型放射光施設(SPring-8)を用いて撮影した合金の凝固過程の画像である。図7 (b)は、本研究のフェーズフィールド法により、同じような2次元的な形状で、4096×128×4096 格子を使ってシミュレーションを行った結果である。合金系は異なるが、成長過程が非常によく一致していることが分かる。

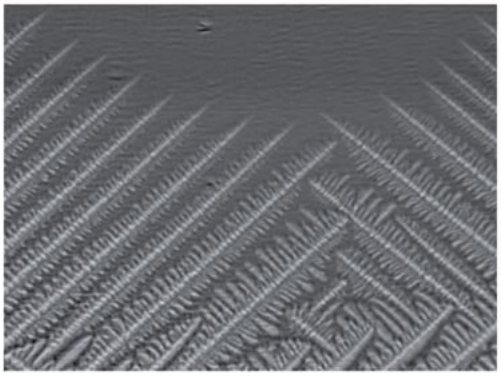


図7 (a) SPring-8を用いて撮影された合金の凝固過程の実験画像(安田教授のご好意による)

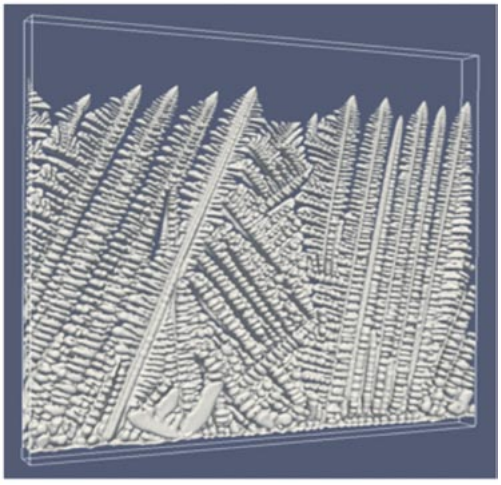


図7 (b) フェーズフィールド法を用いてGPUで計算した凝固成長

GPUコードの実行性能の評価において、GPUコードの浮動小数点演算数を直接測定することは困難である。ここでは全く同じ結果を出すCPUコードに対してPAPI(Performance API)を用いて実測した浮動小数点演算数を元にGPUの実行時間から実行性能を評価している。図8の強スケーリング(単精度計算)は、問題を固定してGPU数を増やして行くときの実行性能の向上を示している。前述の(a)、(b)、(c)の3つの実装の違いによる特徴が表れている。全体の計算領域を512^3、1024^3、2048^3とした。図5、図6のように計算と通信のオーバーラップを導入した(b)Hybrid-YZ method、(c) Hybrid-Y methodではGPU数が少ないときには通信を隠蔽できている。GPU数の増加とともにGPUの計算時間が短くなるため、ある段階で計算時間より通信時間の方が長くなり、もはや通信を隠ぺいすることができなくなる。(c)Hybrid-Y methodでは、GPU数を増やした時にCPUの計算時間が隠ぺいの足を引っ張る部分が大幅に改善され、広い範囲のGPU数で最適化を導入していない(a) GPU-only methodと比較して実行性能が大幅に改善されている。

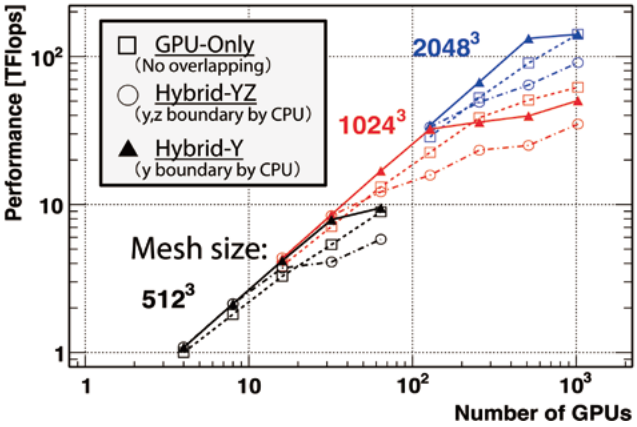


図8 複数GPU計算の実行性能(強スケーリング:単精度計算)

弱スケーリングは1GPUあたりで実行する問題規模を一定にし、GPU数の増加とともに問題サイズも大きくする性能評価である。この測定では、1GPUあたりの計算格子サイズを4096×160×128として、単精度で計算した。図9のように非常に良い弱スケーリングの結果が得られ、(c)Hybrid-Y methodでは、4000 GPUを利用して4096×6480×13000格子に対して行った計算で、2.0 PFLOPS(GPU:1.975 PFLOPS, CPU: 0.025 PFLOPS)という極めて高い実行性能を達成した。これまでにステンシル計算の実アプリケーションとしてPFLOPSを超えた計算例は報告されていない。

TSUBAME 2.0スパコンにおける樹枝状凝固成長のフェーズフィールド法を用いたペタスケール・シミュレーション

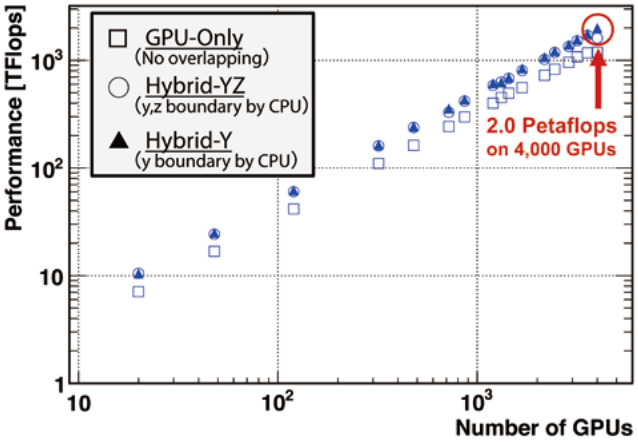


図 9 複数 GPU 計算の弱スケーリング（単精度計算）

TSUBAME 2.0 では消費電力を詳細にモニタリングしている。4000 GPUと16000 CPU を使って計算し、2.0 PFLOPS（ピーク性能の44.5％）の実行性能を達成したときの使用した計算ノードとネットワークの消費電力は約1.36 MWであった（図10）。電力性能は1468 MFlops/Watt であり、少ないエネルギー消費で目的とする計算結果が得られたことになる。

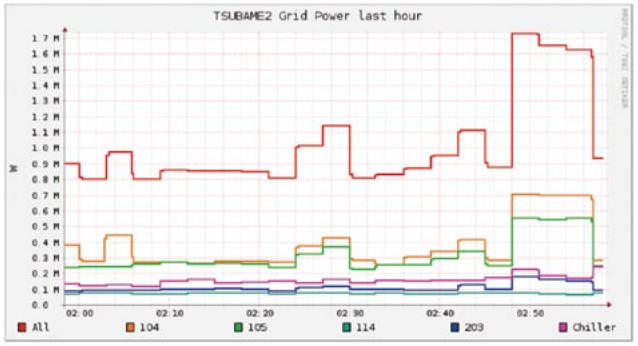


図 10 TSUBAME 2.0 の消費電力モニター

樹枝状凝固の3次元成長を調べるために、4096×1024×4096格子での計算を行った。初期に床面に固相のシードを置き、柱状晶が形成される際の相互干渉などによる空間選択のメカニズムを明らかにすることができる（図11）。

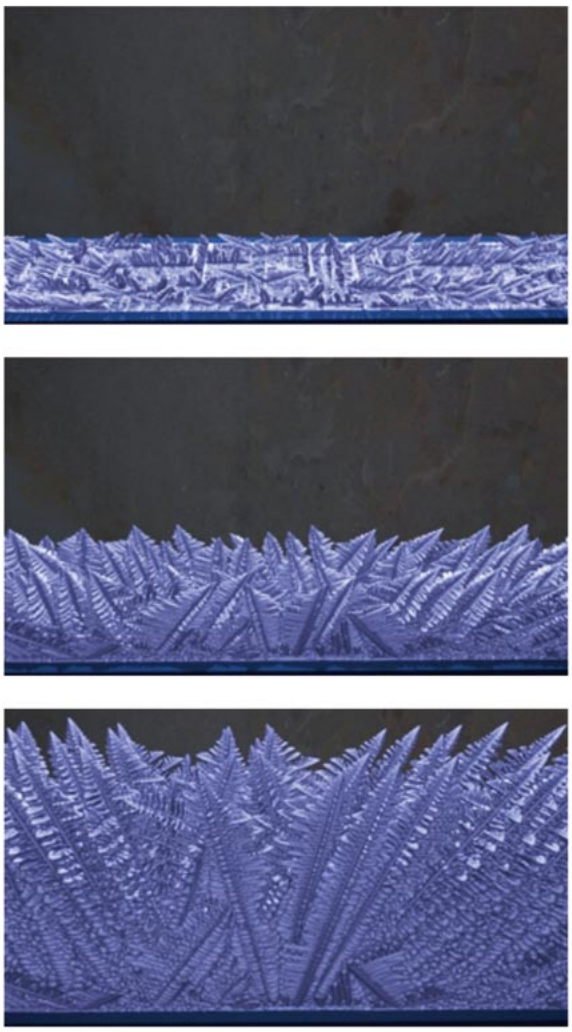


図 11 Al-Si 合金の樹枝状凝固過程の大規模シミュレーション

おわりに 6

フェーズフィールド法により二元合金の樹枝状凝固成長のプロセスをGPUスパコンであるTSUBAME 2.0を用いて大規模計算を行い、格子法に基づいたステンシル計算でありながら単精度で 2.0 PFLOPS という極めて高い実行性能を達成した。これはピーク性能に対して44.5% であり、GPUスパコンが実用アプリケーションに対して十分有効であることを示すことができた。同じような計算でもGPUスパコンを使うことにより大規模計算が可能になり、さまざまな分野での研究の発展や開発の進展が期待できる。

謝辞

本研究は TSUBAME グランドチャレンジ大規模計算制度を利用して実施させて頂き、一部は科学研究費補助金・基盤研究 (B) 課題番号 23360046「GPU スパコンによる気液二相流と物体の相互作用の超大規模シミュレーション」、科学技術振興機構 CREST「次世代テクノロジーのモデル化・最適化による低消費電力ハイパフォーマンス」および「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」、日本学術振興会 (JSPS) グローバル COE プログラム「計算世界観の深化と展開」(CompView) から支援を頂いた。記して謝意を表す。

参考文献

[1] R. Kobayashi: Modeling and numerical simulations of dendritic crystal growth. Physica D, Nonlinear Phenomena, 63(3-4), 410 - 423 (1993)

[2] 小川慧, 青木尊之, 山中晃徳: マルチGPUによるフェーズフィールド相転移計算のスケーラビリティー — 40GPUで5 TFLOPSの実効性能, 情報処理学会論文誌コンピューティングシステム Vol. 3 No. 2 67-75 (2010 June)

[3] A. Yamanaka, T. Aoki, S. Ogawa, and T. Takaki: GPU-accelerated phase-field simulation of dendritic solidification in a binary alloy. Journal of Crystal Growth, 318(1):40 - 45 (2011). The 16th International Conference on Crystal Growth (ICCG16)/The 14th International Conference on Vapor Growth and Epitaxy (ICVGE14)

[4] T. Shimokawabe, T. Aoki, T. Takaki, A. Yamanaka, A. Nukada, T. Endo, N. Maruyama, and S. Matsuoka: Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer, in Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, IEEE Computer Society, Seattle, WA, USA, Nov. 2011.

TSUBAME2 における大規模生体流体力学シミュレーション

Massimo Bernaschi*　Mauro Bisson*　遠藤 敏夫 **
Massimiliano Fatica***　松岡 聡 **　Simone Melchionna*　Sauro Succi*
* CNR-IAC, Istituto Applicazioni Calcolo, Consiglio Nazionale delle Ricerche, Rome, Italy　** 東京工業大学学術国際情報センター　*** Nvidia Corp. Santa Clara, CA, USA

生体中の流体のマルチスケールシミュレーションのためには、複雑形状を流動する粒子の運動を計算する必要があり、数億個オーダーの粒子の相互作用、粒子と流体との相互作用を計算する必要がある。一つの例として、人間の冠動脈の血流を赤血球のサイズの空間解像度を用いて、生理学的なヘマトクリット値（赤血球の体積率）の条件で計算する。本稿ではTSUBAME2スーパーコンピュータ上で高いスケーラビリティを持つ血行力学シミュレーションの手法を提案し、600TFlops以上の高性能を実現した。この結果は、新規数学モデル、計算アルゴリズム、コンピュータアーキテクチャ、チューニング技術の統合により、臨床的に意義のある生体内流れの予測が可能であることを示している。

はじめに　1

輸送現象は生体における基本現象であり、筋肉の収縮、消化現象、細胞への栄養物の輸送、血液循環はその例である。血液は代表的な生体流体であり、代謝、免疫反応、組織修復などの基本的な生理的機能に必要な、生物学的物質を供給する。

生体内の現実的な血液と血管を計算することは困難な課題である。なぜならその計算モデルは、複雑な形状の血管の中を流れ、心拍によって不規則に流速や圧力が変化する流体の動きと、赤血球、白血球や他の粒子の挙動を統合する必要があるからである。

この数年で、大規模血行力学シミュレーションは大きな進歩を遂げた[1-3]が、現実的な形状やサイズの血管に対して、流体と赤血球などの粒子挙動を連立するシミュレーションには至っていなかった。流体の圧力による非局所的相関のために、大域的な形状が、特に動脈の壁面せん断応力に大きな影響を与える。壁面せん断応力は、動脈硬化につながる複雑な生体力学的変化を誘発するとされている。正確で信頼できる血行力学（壁面せん断応力）のシミュレーションは、循環器系疾患の進行を予測するための、新しい非侵襲的な手法となることが期待される。

本稿では、CT血管造影から再構築された人間の冠動脈の、初のマルチスケールシミュレーションについて述べる。冠動脈は心筋に血液を供給し心臓全体に渡るネットワークを成す。シミュレーション規模は5cm程度であり、解像度は赤血球の直径の約8 μ mに対し10 μ m程度である。シミュレーション対象は図1に示すような、約10億ノードから成る流体と1000万～4億5000万の粒子である。シミュレーションは、流体のためのLattice Boltzmann（LB）法と粒子のための特化されたMolecular Dynacmics（MD）法を統合した、MUPHYソフトウェア（Multi PHYsics/multiscale）[4,5]により行う。このシミュレーションはTSUBAME2スーパーコンピュータの4000GPUを用いて500TFlops以上の平均性能、90%以上の並列化効率を達成した。

本研究は、高性能計算技術と物理/計算モデリングの双方の側面において特徴を持つ。非常に複雑な不規則形状の領域をTSUBAME2の4000GPUに均等に割り当てる必要がある。この領域分割問題は流体計算部分だけであっても困難であるが，さらにこのマルチスケール法においてはMD部分の負荷のバランスも良好に保つ必要がある。

このような複雑かつ大規模形状のシミュレーションは既存研究では稀である。本研究では、大規模並列アーキテクチャの利用によって、過去最大規模のシミュレーションを、形状の規則性に依存せずに可能であることを示す。

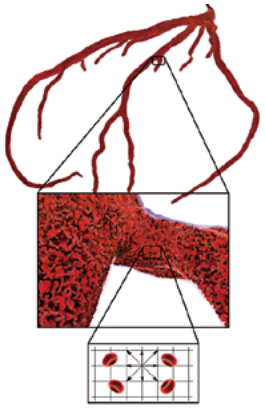


図1　シミュレートされる冠動脈の形状、
Lattice Boltzmannメッシュの中に赤血球（RBC）が存在する。

マルチスケール生体流体力学　2

生物は主に二つの要素から成り立つ：水溶性の溶媒、血漿、サイトソルなどの液体と、溶媒の中を流れる生体粒子、細胞、タンパク質、DNAなどの懸濁物質である。このような条件をシミュレートするために、MUPHY[4,5]ソフトウェアを開発してきた。

以前のバージョンのMUPHYと比べ、現在のフレームワークでは流体と粒子の挙動の数値的解法を扱うことができる。これによりMUPHYは、溶媒と溶質のライブラリを基にして、生体内流れ向けの計算基盤として利用することができる。機能の一部は以下の通りで

ある：ニュートン / 非ニュートンレオロジー特性の選択、分子流体力学から確率的摩擦動力学の再現のためのカーネルの選択、確率的変動の効果、分離した粒子または重合体と分子の形状の溶質の選択、多様な流体力学形状の懸濁物質、異なる粒子間力の選択、規模に対応する粒子・流体間結合機構、不規則境界の物質と組織への対応。このような選択肢により、生体流体は様々な手法でモデル化でき、生物学的・生理学的システムのマルチスケール・マルチフィジックスの挙動が理解できる。

MUPHY は 2 つの計算エンジンを持つ。一つは流体運動学上の流体の挙動を Lattice Boltzmann（LB）法により扱う：衝突による手法により連続体中の流体の挙動を再現する（Navier-Stokes 方程式に基づく直接的巨視的な流体の挙動と対象的に）。二つ目のエンジンは Molecular Dynamics（MD）と技術的側面が共通する手法に基づき、Lagrangian 体の挙動を扱う。しかしながら懸濁物質の性質のためには通常の MD では対応できず、本質的な拡張が必要である。最後に、流体と粒子の結合のために運動モデルに基づいて設計されたカーネルが存在する。これは巨視的流体力学に基づく stresslet や境界積分法などと大きく異なる。これらの計算環境により、現実的な条件における生体流体の研究に利点をもたらす、時間に陽なシミュレーション技術を提供する。特に、1) 冠動脈のような不規則な境界形状に対応する、2) 粒子の動力学を用いるため、複雑な流動学を描写できる、3) メッシュ以下の空間解像度の流体学上の相互作用を可能とする、4) 流体と粒子間の複雑な界面の記述を避ける。
後の二つはシミュレーション性能の加速と、特に物体が流体に及ぼす剛体力に関連した数値安定性の保証のために必須である。フレームワークは、剛体力についてシングルもしくはマルチ時間ステップアルゴリズムを採用する。ここで強調したいことは、LB 法が、力の急激な変化に耐性があり、1000 のオーダーのレイノルズ数のほふく流による濃厚サスペンジョンをシミュレート可能なことである。この利点により現実的な生物学的条件下において広範な物理現象に対応し、構造的関係に支配される連続体に同化できない、非局所的な流動学的反応の再現が可能となる。

2.1 Lattice Boltzmann 法と Molecular Dynamics 法

LB 法 [10] は、離散速度で動く「流体粒子」の格子点 x、時間 t における存在確率の singlet 分散の発展に基づく。流体粒子は物理的粒子の一団 (population と呼ばれる) の集団的動作を表現する。流体と個体の結合は回転並進カーネルに基づき、それらは流体中を動く剛体を、貫通できない物質、柔軟な小胞、またはその組み合わせとして扱う。
流体と固体の流体力学的作用は、i 番目の粒子を中心とした伝達関数に従い形成される。関数は球もしくは楕円体対称であり compact support である。流体力学的形状はメッシュサイズより小さくなりうる [6]。流体と粒子の結合は N 個の懸濁物質ごとに格子点 {x} にまたがる畳み込み演算を必要とする。

基本的に流体力学的作用の計算量は系のサイズの三乗で増加するが、流体と固体が並行で時間発展するため、アルゴリズム上の利点が生じる。実際、LB 法の計算コストは格子点数 M について線形である。ここで O（M）=O（N/c）であり、c = N/M は溶質の密度の定数である。粒子間の直接力の計算のために link-cell 法を用いることにより、粒子力学の計算量は O（N）となる。この性質のために、溶媒を介した粒子間作用は局所的で陽的となり、LB-MD 結合のコストは格子数と懸濁物質数に比例する。しかしながらこの計算は、O（100）の格子点と不連続なメモリアクセスの高コストのために、最も時間を要する部分である。

3 MUPHY

MUPHY（MUlti PHYsics/multiscale）ソフトウェアは元々 Fortran 90 で記述され、並列化のために MPI を用いていた [4]。不規則形状を柔軟でかつ効率的に扱うために、MUPHY は間接アドレッシング法を用いる。これによりメモリ必要量を最小に抑えるだけでなく、良好な負荷分散と、プラットフォームに応じた最良な通信パターンの選択が可能となる。当初 MUPHY は、比較的低クロックの数千の PowerPC プロセッサを高速専用ネットワークで結合した IBM BlueGene アーキテクチャ [7] 上で開発された。BlueGene/P 上で、我々が利用可能な最大構成（294,912 コア）で良好なスケーラビリティをすでに示している [11]。この時数十 TFlops の性能を達成したが、PowerPC アーキテクチャの SIMD 的命令を利用できないなどの制限があった（これらの命令は連続アクセスを必要とするが、MUPHY ではデータアクセスパターンが不連続である）。一方で近年の GPU の計算性能の向上を受け、我々は GPU クラスタを対象とした MUPHY を開発した [11]。

複数 GPU を用いるソフトウェアは、GPU 内と GPU 間の、二レベルの並列性の記述を必要とする。通常のマルチコアシステム上ではハイブリッド方式（OpenMP+MPI など）もしくは単純な分散メモリ方式（MPI ライブラリが共有メモリでも効率的に動作することを期待して）にて実装する一方、複数 GPU システムにおいてはハイブリッド方式が唯一の選択肢である。考慮すべき点はそれだけではない：通常のマルチコア環境では並列度は数スレッド、ハイエンドのものでも数十スレッドである。GPU ではハードウェアを十分に稼働させるためには数百スレッドが必要であり、はるかに細粒度の並列性が必要である。さらに、GPU 上のグローバルメモリのデータを同マシンの別 GPU とやりとりする際には、CPU を介する必要がある（ただし最新の NVIDIA GPU と CUDA ドライバにより、条件によっては可能となる）。

GPU 間の通信のために CPU を介することによるオーバヘッドは発生するものの、CUDA の stream の概念と非同期メモリコピーにより、GPU と CPU 間のデータ転送とカーネル（GPU 上で実行される関数) の実行をオーバラップすることが可能である）。さらに CPU 上の関数（MPI 関数など）の実行は GPU とは並行に行われる。結果として、CPU は MPI 用のコプロセッサのような役割を果たす。

TSUBAME2 における 大規模生体流体力学シミュレーション

3.1 領域分割

本研究のシミュレーション対象の領域は、図 1 に示すように非常に不規則であるため、計算資源間で領域を分割すること自身が大きな課題となる。以前の実験では、計算負荷を均等に分散させるために、SCOTCH グラフ / メッシュ分割ツール [12] の並列版である。PT-SCOTCH を用いた。このツールはグラフバイセクションアルゴリズムに基づき、計算領域の形状にかかわらず分割を行うものである。しかしながら形状の知識が無いと、分割数が増え、部分領域がより不規則になると、分割の性質が悪くなると分かった。境界面積が増え通信オーバヘッドが増加する。より適した解は、グラフに基づく分割と flooding に基づく手法（graph-growing 法とも呼ばれる）を、以下のように統合した場合に得られると分かった：格子はまず PT-SCOTCH により固定数（256）の部分領域に分割される。そして各部分領域は flooding アルゴリズムによりさらに分割される。

MUPHY では通信パターンは実行時に判明する。各タスクは、そのタスクがシミュレーション中にアクセスする必要のある格子点を持った、隣接タスクを決定する。シミュレーション中に、LB アルゴリズムにおいては非局所的な population の流れが存在し、分子のシミュレーションにおいては粒子の移送やドメイン間力の計算があることに注意が必要である。前処理段階においては主に MPI 集団通信が用いられ、それ以降の実行においてはほとんどの通信が以下のような形の一対一通信である：ノンブロッキング通信関数が用いられ、受信処理が先に起動されてから送信処理が起動される。その後各タスクは MPI_wait 関数により、ノンブロッキング通信の終了を待つ。

Lattice Boltzmann 部分においては大域数値（x, y, z 方向運動量など）の計算にのみ MPI 集団通信（reduction）が用いられる。Molecular Dynamics 部分においてもほとんどの通信が一対一である。ただし不規則的形状であることにより領域分割が例外となり、その点は次節で述べる。

3.2 並列 Molecular Dynamics

多くの並列 Molecular Dynamics アプリケーションにおいてはシミュレーション形状は規則的であり、各タスクがほぼ同数の粒子を持つように単純な Cartesian 分割が用いられる。不規則的形状の場合にこのような方法を用いると、LB のための分割と MD のための分割という、二つの異なる分割が生じる。その結果一つの部分領域が二つ以上のプロセッサに所属することとなり、粒子と流体の相互作用が複雑な通信パターンを持つ非局所的処理となってしまう。そのため、我々は MD と LB の間で並列分割を一致させることにした。これにより各タスクは LB の演算、MD の演算、さらに粒子 - 流体間作用の演算をほぼ局所的に実行できる。本手法では LB 格子は、各粒子が所属する部分領域を特定するためにも用いられる：場所 R にある粒子は、そのベクトルにもっとも近い整数ベクトルに対応する格子を持つ部分領域に所属する。粒子は領域間にほぼ均一に存在すると期待できるため、MD 部分の負荷分散も良好に保たれる。

不規則形状の領域に適した新規並列化戦略を開発した。なかでも課題となったのは、部分領域の境界部分の隣に位置し、領域内粒子とも領域間粒子とも作用するような粒子の特定である。我々はセルの概念 [8] によりこれを解決する。ここでのセルは、作用のカットオフ距離以上の辺の長さを持つ立方体であり、計算される不規則形状を埋め尽くすものである（図 2）。この概念により、プロセッサが領域内 / 領域間粒子ペアを効率的に探索可能であり、領域境界に存在する粒子と領域間を移動する粒子の上位集合を交換することにより通信量を削減可能である。

MUPHY のもう一つの構成要素は流体 - 粒子間相互作用の計算部分である。各懸濁粒子は、流体力学的な力と、流体の巨視的な速度と、4 × 4 × 4 サイズの格子に渡る渦度の影響を受ける。同様に格子点は、粒子の運動量輸送の影響を受ける。これらの非局所的な処理は、ある粒子を持つ各プロセッサが流体力学的数量を周りの領域と交換するために、複数の通信ステップを必要とする。

流体 - 粒子間結合は、粒子と隣接領域の境界セル中の格子ノードの間の作用を含む非局所的な処理である。このため結合計算においても領域間力の計算と同様、セルの概念を用いる。流体が粒子に及ぼす力やトルクの計算のために、各プロセッサは境界セル中の粒子の情報を近隣と交換する。最後に外部の粒子に関連する力やトルクは隣接プロセッサと交換され、受信側では外部からの影響が境界粒子に伝えられる。この手法は他の手法より効率的である。たとえば粒子の代わりに境界セル中の格子点を交換した場合、一ステップにおける通信回数を削減することができる。しかしこの手法では界面に近いすべての格子ノードの交換のために通信オーバヘッドは大きくなる。

最後に、粒子が流体へ及ぼす運動量輸送については、境界上の粒子の情報が交換される。この点は流体から粒子への影響の場合と同様だが、二回目のデータ交換は不要である。

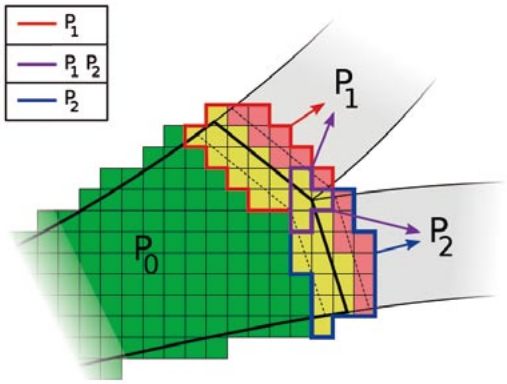


図 2 境界セルと内部セルを不規則領域に敷き詰めた様子。隣接領域とのデータ交換を限定することができる

4 実験結果

これまでに述べたシミュレーションについて、全体実行時間と、計算と通信の内訳について測定する。シミュレーション対象は以下の通りである：流体については約 10 億の格子ノードから成る領域であり、約 3000 億ノードの bounding box に収まる。粒子は 4 億 5000 万の赤血球（RBC）である。流体 - 粒子結合のパラメータについては、以下のように設定した：各赤血球は小球の最小・最大主要方向について 4 μ m および 8 μ m の流体力学的形状を持つとし、ヘマトクリットは 58%とした。

多くの演算は単精度浮動小数で行われ、一部の reduction 処理は倍精度で行われた。LB 計算においては各 GPU スレッドは複数格子ノードの更新処理を行い、そのノード数は利用した GPU 数に依存する（スレッド設定は各 GPU 固定とした）。たとえば 512GPU の場合各スレッドは 8 格子ノードを担当する。MD 計算においては、粒子間作用の演算は粒子ごとに行われる。ここでスレッドグリッドは粒子の配列にそのまま対応するスレッドはそのグローバル ID に応じて粒子に対応し、作用ペアの探索は各スレッド独立に行われる。各スレッドは隣接セルを調べ、各作用ペアについて力への寄与を計算する。この実験により、生理学的なレベルのヘマトクリットにおけるコードの信頼性の基礎試験を行う。

図 3（上部）では、シミュレーションステップごとの実行時間と、LB 部分、MD 部分の内訳時間を示す。いくつかの知見を述べる：MUPHY の Lattice Boltzmann 部分の 1GPU 上の性能は、高度に最適化された他の CUDA LB カーネルと一致する。GPU 数を増やした場合には実行時間は大幅に削減されており、4000GPU の場合には 256GPU の 12.5 倍の性能向上が見られ、並列化効率は約 80%に相当する。MD における直接力の計算は効率 95%に相当する。LB 部分は非常に効率的であり、負荷は常に 4%以下である。

またこの結果は非同期通信と通信・計算のオーバラップの組み合わせにより GPU 間の直接データ交換ができない問題を緩和できていることを示す。

図 3（下部）は全体の並列化効率を示す。その基準は、利用可能なメモリの範囲でシミュレート可能なヘマトクリットレベルの場合には 256GPU であり、それより高レベルの場合は 512GPU である。1024GPU までは線形を超えた性能向上が起こっており、また利用した最大 GPU 数の場合の並列化効率は 80%程度であると分かる。この高いスケーラビリティは、複数の枝を持つ冠動脈向けに提案したグラフ分割と flooding 法のハイブリッド方式により実現できたと考えられる。1200 領域を用いるテストケースでは赤血球の分布には広い分散があったが、MD/LB 部分の実行時間は小さかった。これは部分領域間の境界が小さく計算と通信の共有が最適化されたためである。

図 4 は 4000GPU を用いた場合の通信時間の分散を示す。平均的にはほとんどのタスク（4000 中約 3000）が、全体時間の 50%程度を通信に費やしている。この結果は非同期通信方式が良好に動作し、GPU が計算にほとんどの時間を費やす一方で、CPU がその MPI コプロセッサとして働いていることを示している。

TSUBAME2 の 1334 ノード（4000MPI タスク）を用いた場合の、（重みつき）平均性能は、600TFlops 弱となる。図 5 にその内訳を示す。この成果を実用にもたらすとすると、この TSUBAME2 全体を用いることにより、完全な心拍の、マイクロ秒の時間解像度における、赤血球を全て含んだシミュレーションを、48 時間で行うことが可能である。

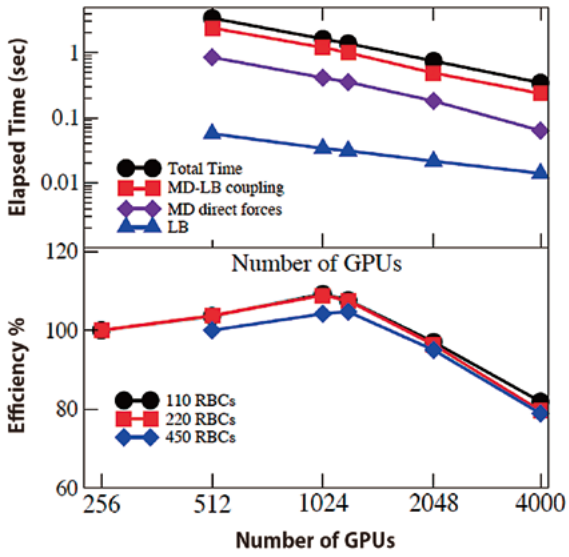


図3 上部: 時間ステップあたりの経過時間。下部: 並列化効率。

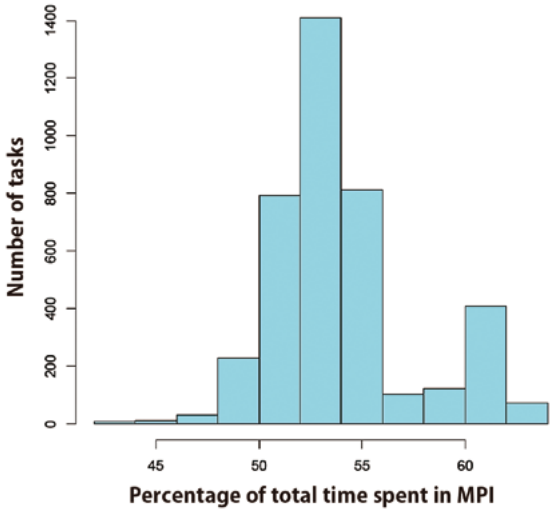


図 4 4000の各タスクが通信に費やした時間の割合の分布

TSUBAME2 における 大規模生体流体力学シミュレーション

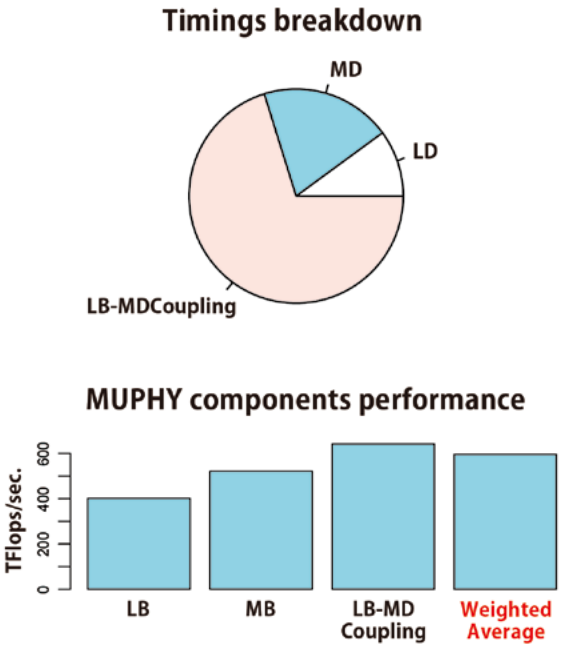


図5 各計算コンポーネントの計算時間の割合とそれぞれの性能

おわりに 5

世界トップクラスの GPU クラスタを用いた、心臓規模の冠動脈の計算生体流体力学シミュレーションを初めて行った。シミュレートされた血行力学系は赤血球レベルの空間解像度を持ち、人間の冠動脈の複雑な形状から成る。計算された環境は 10億の流体ノードと、並行に流れる数億の赤血球を含む。TSUBAME2 スーパーコンピュータの持つ 4000GPU を用いて、約 600TFlops の性能と 90%以上の並列化効率を実現し、4 億 5000 万の赤血球の存在下で約 2 兆回 / 秒の lattice 更新を行った。

この成果は、高性能計算技術と物理的 / 計算モデルの双方における新規開発によるものであり、我々の知る限り、理想化されない形状に対応した初の実装である。この研究は、現実の生体流体力学の研究に対するコンピュータシミュレーションの可能性を大幅に進歩させるものであり、心臓血管の臨床における応用を可能にするものである。

謝辞

本研究はTSUBAME グランドチャレンジ大規模計算制度で実施された。E. Kaxiras, C.L. Feldman, A.U. Coskun, F.J. Rybicki, A.G. Whitmore, G. Amati, F. Pozzati, F. Schifano の各氏との議論に感謝する。

参考文献

[1] D.A. Vorp, D.A. Steinman, C.R. Ethier, Comput. Sci. Eng., pp. 51 (2001).
[2] A. Quarteroni, A. Veneziani, P. Zunino, SIAM J. Num. Analysis, 39, 1488 (2002).
[3] L. Grinberg, T. Anor, E. Cheever, et al., Phil. Trans. Royal Soc. A, 367 1896 2371 (2009).
[4] M. Bernaschi, S. Melchionna, S. Succi et al., Comp.Phys. Comm., 180, 1495, (2009).
[5] S. Melchionna, M. Bernaschi, S. Succi et al, Comp.Phys. Comm., 181, 462, (2010).
[6] S. Melchionna, Macromol. Theory Sim., DOI: 10.1002/mats.201100012 (2011).
[7] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras, Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.1466 (2009).
[8] M. Bisson, M.Bernaschi, S.Melchionna, Commun. Comput. Phys., 10, 1071 (2011).
[9] S. Melchionna, J. Comput. Phys. 230, 3966 (2011).
[10] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Oxford University Press, USA (2001).
[11] A. Peters et al., Proceedings of Supercomputing 2010, New Orleans, 2010.
[12] http://www.labri.fr/perso/pelegrin/scotch

大規模グラフ処理ベンチマークGraph500のTSUBAME 2.0 における挑戦

鈴村 豊太郎 */** **上野 晃司** *
*東京工業大学大学院・情報理工学研究科 **IBM Research – Tokyo

Graph 500とは、スーパーコンピュータのグラフ処理性能を測定する新しいベンチマークである。
スパコンのベンチマークでは、数値計算性能を測るLinpackによるTop 500が有名だが、近年、大規模グラフ処理が、重要性を増しており、Graph500ベンチマークが広がりを見せている。Graph500のリファレンス実装は、使用されているアルゴリズムの問題により、分散メモリ環境で大規模にスケールさせることができなかった。そこで、大規模にスケール可能な2次元分割に注目した。本論文では、2次元分割をTSUBAME2.0上に実装し、1366ノードで頂点数2^36(68.7 billion)、エッジ数2^40(1.1 trillion)のグラフ(Graph500のScale 36)のBFS(幅優先探索)を10.955秒で計算した。
TEPS値は100.366 GE/sであり、2011年11月に発表されたランキングでは世界3位を獲得した。

1 はじめに

大規模グラフ処理はWebページのリンク解析、タンパク質間の相互作用解析、サイバーセキュリティ、VLSIのレイアウトや道路網、送電網の最適化など様々な応用分野あり、近年盛んに研究されている。従来、スーパーコンピュータは物理シミュレーションなどの数値計算に、主に使われてきたが、大規模グラフ処理も重要なアプリケーションとなりつつあり、そのような中、スパコンのグラフ処理性能を測る、Graph500[1]という新しいベンチマーク登場し、注目を集めている。Graph500は、スパコンの通信性能や、グラフデータを格納するメモリの大きさや、メモリへのランダムアクセス性能を測るという、データインテンシブなベンチマークであり、数値計算性能を測るTop 500ベンチマークとは計測する性能が全く異なる。本論文では、Graph500の概要、我々が提案するスケーラブルな最適化実装とTSUBAME2.0 における性能評価について述べる。

2 Graph500の概要

本章では，Graph500ベンチマークの概要と、分散BFSアルゴリズムについて述べる。

2-1 Graph500ベンチマークの概要

Graph500は、大規模なグラフに対してBFSによる探索を実行するベンチマークである。単位時間に処理できたエッジ数と、扱える最大問題サイズが評価指標となる。計算インテンシブなTop500ベンチマークと違い、Graph500ベンチマークは、データインテンシブなベンチマークである。扱える問題サイズは、グラフの頂点数=2SCALEであるようなSCALE値で表す。単位時間に処理できたエッジ数は、TEPS (Traversed Edges Per Second) 値で表す。例えば、100万TEPSとは、100万個の枝を持つ連結グラフのBFSが1秒で完了した場合の性能である。

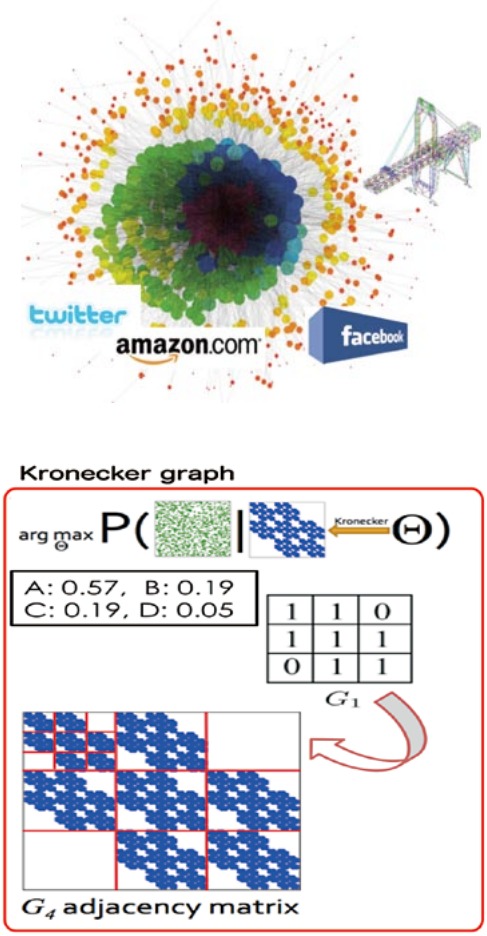


図1 クロネッカーグラフ[4]

ベンチマークを実行するプログラムは、(a)グラフデータの生成、(b)計算するのに最適なデータ構造への変換、(c)BFSによる探索、(d)計算結果の検証の4つの部分から成る。ベンチマークの実行順は次のようになっている、最初に(a)、(b)によりグラフデータを構築し、グラフから始点を64個選ぶ。次に、64個の始点それぞれに対して順番に、(c)BFSによる探索と、(d)計算結果の検証を行う。複数の始点からの探索を同時に行うことはできない。時間を計測しベンチマークとする部分は、(b)のグラフデータ構造への変換(Kernel 1) と、(c)

大規模グラフ処理ベンチマークGraph500のTSUBAME 2.0 における挑戦

のBFSによる探索(Kernel 2) のみである。(a)では、枝数が頂点数の16倍となるようなクロネッカーグラフ[4]を生成する。枝はすべて重みなし、無向辺である。ここで生成されるデータは規則性のない順番で並んだ、枝のリストである。(b)では(a)で生成された枝リストから、隣接行列のCSR (Compressed Sparse Row)や、CSC (Compressed SparseColumn)などのグラフデータ構造に変換する。(c)のBFSは、BFSで辿った頂点の軌跡であるBFS木を出力する。(d)では、このBFS木が正しいかどうかチェックする。このチェックでは、BFS木にループがないこと、枝の張っている頂点同士の深さの差が1以下であること、などの5つのルールを満たしていることをチェックする。

2-2 分散 BFS アルゴリズム

Graph500のリファレンス実装には、OpenMPで書かれた実装や、MPIで書かれた実装、Crayの共有メモリ型プログラミング環境用の実装、など複数の種類が用意されている。TSUBAME2.0で分散実行するには，MPIで書かれた実装を使用する。MPIで書かれた実装には、さらにアルゴリズムや実装方法の異なる4種類の実装が用意されている。これらの実装は、対象としているプログラミング環境や分散方法などは異なるが、全てベースとなるアルゴリズムとしてLevel-synchronized BFSを使っている。このアルゴリズムは、各レベル(深さ)について、そのレベルの頂点をすべて処理してから、次のレベルに進むというアルゴリズムである。

Algorithm I: Level-synchronized BFS

```
1   for all vertex v in parallel do
2   | pred[v]← -1;
3   pred[r] ← 0
4   Enqueue(CQ r)
5   While CQ != Empty do
6   |   NQ ← empty
7   |   for all u in NQ in parallel do
8   |   |   u ← Dequeue(CQ)
9   |   |   for each v adjacent to u in parallel do
10  |   |   |   if pred[v] = -1 then
11  |   |   |   |   pred[v] ← u;
12  |   |   |   |   Enqueue(NQ, v)
13  |   swap(CQ, NQ);
```

アルゴリズム1はLevel-Synchronized BFSの擬似コードである。まず、BFS木を格納するPREDと、頂点が訪問済みかどうかを格納するVISITEDを初期化する。PRED[v]は頂点vのBFS木における親頂点を表す、初期値-1はBFS木にまだ入っていないことを表している。VISITED[v]は頂点vが訪問済みかどうかを表す。初期値0はまだ訪問していないことを表している。次に、BFSの始点となる頂点をCQ(Current Queue)に入れ、探索を開始する。

探索においては、7~16行の1ループが1レベルに相当する。このループ中で、CQは現在のレベルで訪問する頂点、NQ (Next Queue)は次のレベルで訪問する頂点が格納されている。例えば、レベル1でCQに頂点γが入っていたとすると、11、12行目でγの隣接頂点が訪問済みかどうかチェックされ、まだ訪問していない頂点はNQに格納される。次のレベルでCQにはこれらの頂点が格納されていることになる。9行目のforと、11行目のforは並列化が可能なループである。リファレンス実装の4つのMPI実装は、基本的にはLevel-synchronized BFSを実装しているが、グラフデータの分散方法などに違いがある。4つのリファレンスMPI実装の処理の仕方の違いやTSUBAME 2.0 上における性能特性の結果は我々の先行研究[2]を参照して頂きたい。

3 2次元分割によるスケーラブルな実装

リファレンス実装は全て1次元分割を使っているが、1次元分割はスケールさせることが難しい[2]。そこで、隣接行列を2次元に分割するアルゴリズム(2次元分割)[3]を実装したプロセッサをP = R x Cの2次元メッシュ(mesh)に配置する。このメッシュの行を「プロセッサ行」、列を「プロセッサ列」と呼ぶことにする。隣接行列を図4のようにR*C個の行とC個の列に分割し、プロセッサ(i, j)は、隣接行列のA_(i,j)^((1))~A_(i,j)^((C))のCブロックを担当する。頂点は、R x C個のブロックに分割し、プロセッサ(I, j)は、j*R+I 番目のブロックを担当する。1レベルにつき、expandとfoldの2段階の通信を行う。各プロセッサは自分の担当する頂点ブロックのCQを同じプロセッサ列の他のプロセッサに送信する。これをExpandという。Expandは1次元分割の縦分割と同じように、CQをコピーする通信であるが、隣接行列は横にもC個に分割されているので、通信は、同じプロセッサ列の他のプロセッサとだけ行う。次に、各プロセッサはCQと各プロセッサが持っている部分隣接行列から、CQの隣接頂点を探す。PREDやNQを更新するため、CQの隣接頂点を、その頂点の担当プロセッサに送信する。この通信をFoldという。PREDを更新するのに、親の頂点が必要なので、Foldでは、CQの隣接頂点と、親頂点(CQの頂点)の組みを送信することになる。Foldは1次元分割の横分割と同じように、CQの隣接頂点を担当プロセッサに送信する通信だ。しかし、2次元分割では、隣接行列の分割方法から、Foldの通信を行う必要のある相手は、同じプロセッサ行の他のプロセッサのみとなる。

2次元分割の利点は、通信で絡むプロセッサ数が少ないことである。1次元分割では、2種類の分割方法のどちらも、全対全の通信が必要だったのに対し、2次元分割の場合、Expandでは同じ列のノード(R-1)プロセッサと、foldでは同じ行のノード(C-1)プロセッサとしか通信を行わない。よって、通信するプロセッサ数を少なくすることができ、大規模に分散可能になる。

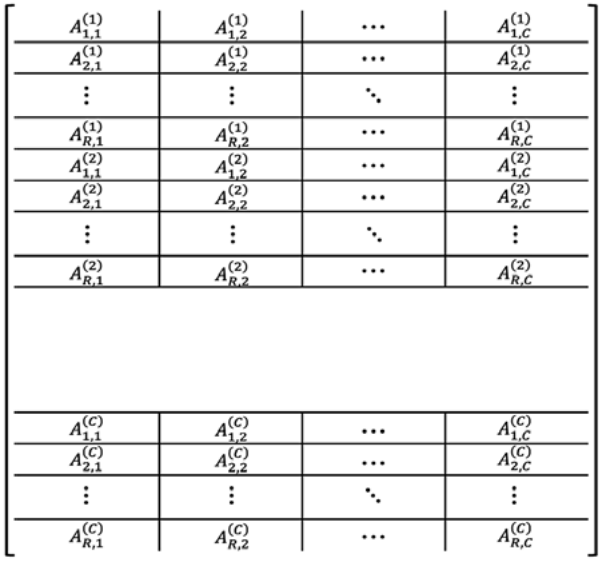


図2 隣接行列の2次元分割

4 性能評価

TSUBAME2.0上での性能評価の結果について述べる。TSUBAME2.0は、1400以上のノードがFat-Treeによるフルバイセクションのインフィニバンドネットワークで接続されている。各ノードには、Intel CPU Xeon 5670 2.93GHz（Westmere EP、6コア、256-KB L2 キャッシュ、12-MB L3）が2つ、NVIDIA M2050 GPU（Fermi）が3つ、48GBのメモリが搭載されている。通信は、各ノードはInfiniband QDRが2リンク使用可能で、合計80Gbpsの通信バンド幅を備えている。

最大1024ノードまで使用して実験した。なお、TSUBAME2.0はGPUメインのスパコンだが。GPUは使用していない、TSUBAME2.0は1ノードあたり物理コア12個だが、SMTを有効にすると仮想的に24コアになる。1ノード24コアとして、各プロセスに均等に割り振った。gcc 4.3.4（OpenMP 2.5）、MVAPICH2 1.6[4]。比較するリファレンス実装は、執筆時点で最新のversion 2.1.4である。

図3は2次元分割とリファレンス実装の比較である。横軸はノード数で縦軸はTEPS（GE/s）である。リファレンス実装のreplicated-csr、replicated-cscと最適化実装は、1ノードあたり2MPIプロセスで実行し、Simpleは1ノードあたり16MPIプロセスで実行した。図7は図6の性能をノード数で割り、1ノードあたりの性能を算出したグラフである。リファレンス実装のsimpleを参考に掲載した。リファレンス実装で、データがない部分はエラーなどで計測できなかったところである。

2次元分割の実装は、リファレンス実装のsimpleの2倍程度の速度が出ている。これは、送信処理と受信処理の並列化や、OpenMPによるプロセス内の並列化の効果によるものである。2次元分割の実装は、リファレンス実装のreplicatedと比べると、性能が低い。これはreplicatedのアルゴリズムはノード数が小さい場合には通信データ量を小さくすることができ、有利だからである。図3から分かるように、replicatedの優位性もノード数が増えるにしたがって急激に低下し、通信データ量は512ノードで2次元分割と逆転する。実際、図3からreplicated-cscはノード数128で既に性能の限界が見え始めている。また、図4は1ノードあたりのWeak Scalingによるスケーラビリティの評価だが、1024ノードまでノード数を増加させても性能が向上し、十分なスケーラビリティが得られていることがわかる。

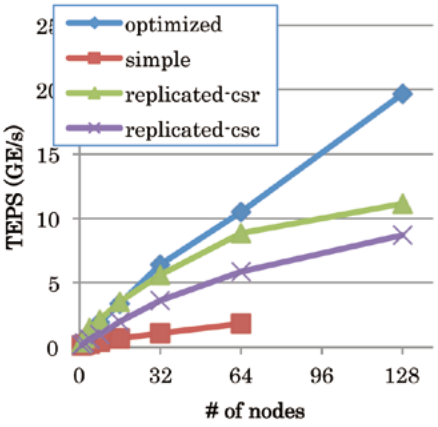


図3 2次元分割と参照実装の比較

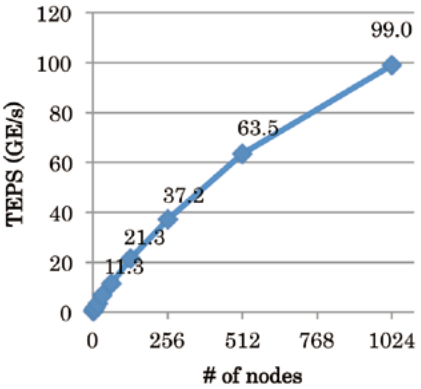


図4 1024ノードまでのスケーラビリティ

まとめと今後の展望 5

本論文では大規模分散環境でスケールさせるため2次元分割によるBFSを実装した。2011年11月のGraph500におけるスコアは、1366ノードで頂点数2^36(68.7 billion)、エッジ数2^40(1.1 trillion) のグラフ(Graph500のScale 36)のBFS(幅優先探索)を10.955 秒で計算した。TEPS値は100.366 GE/sであり、2011年11月に発表されたランキングでは世界3位を獲得した。我々は、2次元分割の他に、通信データの圧縮や頂点の並び替えなどによる最適化も行なっている。
それらの成果は、また別の機会に発表する。

謝辞

本研究の成果は、TSUBAME2.0グランドチャレンジ制度、科学技術振興機構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」から支援を頂いた。

参考文献

[1] Graph500 : http://www.graph500.org/.
[2] Toyotaro Suzumura, Koji Ueno, Hitoshi Sato, Katsuki Fujisawa and Satoshi Matsuoka, "Performance Evaluation of Graph500 on Large-Scale Distributed Environment", IEEE IISWC 2011 (IEEE International Symposium on Workload Characterization) , 2011/11, Austin, TX, US
[3] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L. SC 2005.
[4] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in Conf. on Principles and Practice of Knowledge Discovery in Databases, 2005.

Physis: ヘテロジニアススパコン向けステンシル計算フレームワーク

丸山 直也* 野村 達男** 佐藤 賢斗*** 松岡 聡*
*東京工業大学・学術国際情報センター **Google, Inc. ***東京工業大学・情報理工学研究科

ステンシル計算を対象とし、TSUBAME2.0のような大規模GPUスーパーコンピュータを簡便に利用可能とするフレームワークを提案する。ステンシルを表現する関数を記述し、それを基にフレームワークが自動的にGPU実行コードに変換する。また通信と計算の最適化などの多数の計算ノード上の複数GPUを効率良く使うための種々の最適化を自動的に施す。
本稿では同フレームワークの概要を報告し、TSUBAME2.0を用いた評価結果より良好な性能を達成できることを示す。

1 はじめに

通常のCPUに加えてGPUアクセラレータを共用した計算がその高い性能および電力効率より注目をあびている。Tsubameに搭載されているGPUは515GFLOPSの性能を持ち、同じくTsubameで用いられているCPUの性能の数枚倍高速である。また計算速度だけでなくメモリバンド幅にも優れており、カードあたり150GB/sの速度を達成している。これにより計算速度律速なアプリケーションだけでなくメモリバンド幅律速なアプリケーションにおいても大幅な性能向上が可能であり、実際にTsubameにおいて実証されている[2]。

しかしながらそのような異種プロセッサから構成されるシステム上におけるプログラミングは均質システムに比べて困難である。既存のプログラミングモデルは低レベルかつ個々の機種固有な場合が多く、現状では複数のプログラミングAPI、言語などを共用したハイブリッドプログラミングが必要とされている。例えば、複数のノード上のGPUを利用するためにはGPU用プログラミングAPIおよび複数ノード利用のためのメッセージパッシングAPIなどを協調させたハイブリッドプログラミングが一般的である。これによって個々のシステムコンポーネントの並列性を生かした高効率なアプリケーションの開発が可能になるが、その一方でプログラミングの複雑さが大幅に増加し、プログラム開発行程の増加につながる。また、正しく動作するハイブリッドプログラムを開発させることに加えて、実際に性能向上を達成するためには種々の高度な最適化を施す必要がある。キャッシュブロッキングなどの個々の最適化技術の多くはこれまでによく知られた技術であるが、汎用プログラミング言語のコンパイラ等によって自動的に適用される場合は限定的であり、プログラムを手動で変更する必要がある。GPUクラスタなどの高性能計算環境において高い性能を達成するためには、複数のプログラミングモデルを共用し、さらにそのような複雑なプログラム上で種々の最適化を適用する必要があり、生産性が大幅に低下する。

この問題を解決するためにはより抽象度の高い統合されたプログラミングモデルが必要である。高い抽象度により生産性を向上させ、かつ実行環境によらない可搬性のあるプログラミングモデルの実現が可能である。一般に抽象度を上げることによって性能上のコストが発生するが、それによって生産性が大幅に改善されることも多くの場合重要である。

本稿ではそのような高い抽象度を有したプログラミングモデルの例として、ステンシル計算に特化したアプリケーションフレームワークPhysisを提案する。格子上のステンシル計算では各格子点についてその隣接点を参照することで値を更新する。このような計算パターンは数値シミュレーション、特に偏微分方程式に基づいた計算において頻繁に出現する。ステンシル計算は典型的には性能はシステムのメモリバンド幅に律速される。これはステンシル計算のB/F値がGPU等を含む今日の高性能計算システムのB/F値を上回る場合が多いためであり、従ってメモリアクセスの最適化が重要になる。例えばGPUにおいてはメモリアクセスレイテンシを隠蔽するために非常に多くのスレッドの発行やデータのアライメント、キャッシュブロッキングなどが知られている[3]。これらに加えて複数ノード上のGPUを高効率に用いるためには通信と計算のオーバーラップなどの最適化が重要になる。

Physisフレームワークは実行環境独立な可搬性を有したフレームワークであり、かつ上述の環境固有な最適化を透過的に実現する。直交格子上のステンシル計算をプログラムするために多次元格子の生成、データ移動、ステンシルの適用等を宣言的に表現可能な構文を提供する。直交格子は大域的メモリ空間上に生成され、その操作は実際のメモリシステムの構成によらず単一のメモリ空間上操作としてプログラム可能である。これにより高い生産性および実行環境独立性を実現する。また高い抽象度の宣言的なモデルにより分散メモリ環境上の自動並列化などの高度なコンパイル技術の適用を可能し、さらに自動チューニングや自動チェックポイントなどのソフトウェア技術の適用も可能な設計となっている。

本稿ではPhysisフレームワークのC言語をベースにした実装について報告する。本実装では標準的なC言語を基盤とし、それに対してフレームワークを実現するための小規模な拡張構文を導入したドメイン特化型言語（DSL）を定義する[4]。拡張構文を用いたプログラムはPhysisフレームワークによって実行環境向けのプログラムに自動変換される。具体的にはPhysisプログラムをGPUクラスタ上で実行する場合はフレームワークによってMPIおよびCUDAを用いたソースコードに自動変換される。またこの際に通信と計算のオーバーラップ

Physis: ヘテロジニアススパコン向けステンシル計算フレームワーク

などの最適化も自動的に適用される。

本フレームワーク実装の有効性を評価するためにステンシル計算をフレームワーク上に実装し、その性能をTSUBAME2.0の256GPUを用いた評価を行う。本稿ではその結果の一部について報告し、良好な性能およびスケーラビリティが達成可能であることを示す。より詳細な結果については文献 [1] を参照されたい。

ステンシル計算フレームワーク 2

ステンシル計算において高い生産性の実現を狙ったフレームワークを設計する。同フレームワークはドメイン特化型プログラミング言語（DSL）およびアーキテクチャ固有ランタイムから構成される。DSLは宣言的かつ機種独立にステンシル計算を記述可能であり、ソースコード変換器によって実行アーキテクチャ向けコードに変換される。ランタイムは多次元格子データを簡便かつ機種独立に操作するための抽象化を提供するものである。本節ではフレームワーク設計における主な目標を述べる。

自動並列化： Physis DSLは分散メモリ環境を含む様々な環境において自動並列化が可能なものとして設計する。一般には自動並列化、特にデータの局所性を考慮した並列化は非常に困難であり、汎用言語では分散メモリ環境上などでは現実的ではないとみなされている。我々はDSLを特定の計算パターンに限定し、必要な制約を導入することで本DSLで記述されたプログラムの自動並列化が可能となるようにDSLの設計を行う。

組み込み型DSL設計の採用：全く新しい言語を設計することで、対象の問題に対して高度に最適化されたプログラミングモデルを提供することが可能である。しかしながら、現実的には既存プログラミング言語と大きく乖離することは普及の妨げにもなるため、我々はPhysisにおけるDSLを広く使われている既存汎用言語に対してステンシル計算用拡張を施した組み込み型DSLとして設計する。本稿ではそのような広く普及した言語としてC言語を基盤とする。

宣言的かつ高い記述性を持った言語：高い生産性を実現するために高い抽象度を持った宣言的プログラミングモデルを採用する。宣言的とすることで命令的モデルと比較して記述量を削減可能である。例えばPhysis DSLではプログラマは各格子点の計算について記述するが、格子全体がどのように計算されるかはフレームワークによって実行環境毎に最適となるように決定される。高い抽象度のプログラミング言語はその実装一方で抽象度を必要以上に高めると複雑な現実のシミュレーションの記述が困難になりうる。我々は生産性と性能を両立するために対象ドメインのアプリケーションの記述に必要な構文を吟味した設計を行う。

プログラミングモデル 3

Physis DSLは標準的なC言語にステンシル計算用のデータ型とイントリンシックスを追加したものである。本拡張を用いて記述されたプログラムはソースコード変換器によって実行環境向けのソースコードに変換される。

Physisでは浮動小数点値を持った直交のカルテシアン多次元格子を利用可能である。多次元格子を表現するために格子の型と次元に基づいたデータ型として PSSGrid3DFloatやPSGrid2DDoubleなどを導入する。前者はfloat型を格子データとして持つ3次元格子であり、後者はdouble型の2次元格子である。これらの型の内部構造はユーザプログラマからは隠蔽されており、実態へのハンドルとして機能する。

Physisでは次元および格子点の型によって異なる複数の格子型が提供されているが、以下では説明の簡便化のために格子の型の名前としてPSGridを用いる。

PSGridFloat3D 型の格子は PSGridFloat3DNewおよびPSGridFreeにより作成、破棄される:

```
PSGrid3DFloat PSGrid3DFloatNew(
 size_t dimx, size_t dimy,
 size_t dimz,
 enum PS_GRID_ATTRIBUTE attr)
void PSGridFree(PSGrid g)
```

格子の各要素には以下のイントリンシックスにより一括および各点毎にアクセス可能である:

```
void PSGridCopyin(PSGrid g,
                  const void *src)
void PSGridCopyout(PSGrid g, void *dst)
PSGridGet(PSGrid g, size_t i,
          size_t j, size_t k)
void PSGridSet(PSGrid g,
 size_t i, size_t j, size_t k, T v)
void PSGridEmit(PSGrid g, T v)
```

PSGridGet およびPSGridSetのsize_t 型のパラメータは同じくパラメータとして与えられる格子における位置を表し、格子の次元と同数の個数のパラメータを持つ。PSGridGetの返り値やPSGridSet, PSGridEmitのパラメータvは格子点の型と同一である。

3-1 例

図1は2次元格子における9点ステンシルを実装したステンシル関数である。ステンシル関数の実際の格子への適用はPSStencilMapおよびPSStencilRunを用いて図2のようにプログラム可能である

```
void diffusion(const int x, const int y,
 PSGrid2DFloat g1, PSGrid2DFloat g2, float t) {
 float v=
  PSGridGet(g1,x,y)+PSGridGet(g1,x+1,y)
  +PSGridGet(g1,x-1,y)+PSGridGet(g1,x,y+1)
  +PSGridGet(g1,x,y-1)+PSGridGet(g1,x+1,y+1)
  +PSGridGet(g1,x+1,y-1)+PSGridGet(g1,x-1,y+1)
  +PSGridGet(g1, x-1, y-1);
 PSGridEmit(g2, v / 9.0 * t);
}
```

図1 9点ステンシルの例

```
PSGrid2DFloat g1 = PSGrid2DFloatNew(NX, NY);
PSGrid2DFloat g2 = PSGrid2DFloatNew(NX, NY);
// initial_data is a pointer to input data
PSGridCopyin(g1, initial_data);
PSDomain2D d = PSDomain2DNew(0, NX, 0, NY);
PSStencilRun(
   PSStencilMap(diffusion, d, g1, g2, 0.5),
   PSStencilMap(diffusion, d, g2, g1, 0.5));
```

図2 ステンシル関数の格子への適用例

4 性能評価

提案フレームワークの有効性を評価するためにプロトタイプ実装を行った。実装にはROSEコンパイラフレームワーク[5]を用い、ソースコード変換器およびランタイムから構成される。変換はCPU実行向けコードの場合はCコードを生成し、GPUの場合はCUDAコードを生成する。また分散メモリ環境の場合はMPIにより並列化を行う。

性能評価のために以下の2つのステンシルをPhysis DSL上で実装した。

・Diffusion: 3次元7点ステンシル
・Seismic: 3次元地震波伝播シミュレーション

Diffusionは単一のステンシル関数から構成される比較的小規模なプログラムとなっているが、seismicは27個のステンシル関数およびスタッガード格子を用いて構成される。27個の内、6個は3次元格子の2次元境界平面の計算であり、PSDomainオブジェクトによって格子全体に対して部分的に適用される。プログラムは2つとも単精度浮動小数点を用いる。

評価環境としてTSUBAME2.0を用いる。同マシンは1408台の計算ノードから構成され、各ノードはIntel Xenn Westmere-EP 2.9GHzのCPUを2基、NVIDIA M2050 GPUを3基搭載する。メモリはシステムメモリとして 52GB、GPUメモリとして各GPUあたり3GB搭載している。計算ノードは2本のQDR Infinibandによって接続され、フルバイクセションバンド幅ネットワークを実現するファットツリートポロジを構成する。ソフトウェアとしてはOSとしてSUSE Linux Enterprise Server 11 SP1を用い、CUDA v3.2、gcc/g++ v4.1.2を用いた。

4-1 ウィークスケーリング

図3はdiffusionのウィークスケーリングにおける性能を示したものである。赤線および青線がそれぞれGPUあたり256x128x128、512x256x256の計算を行った場合であり、Y軸およびZ軸の2次元分割を行った。結果よりGPUあたりのデータサイズが大きい512x256x256の場合はほぼ256GPUまで性能がスケールすることがわかる。一方で、256x128x128の場合でも256GPUまで良好に性能向上していることがわかる。

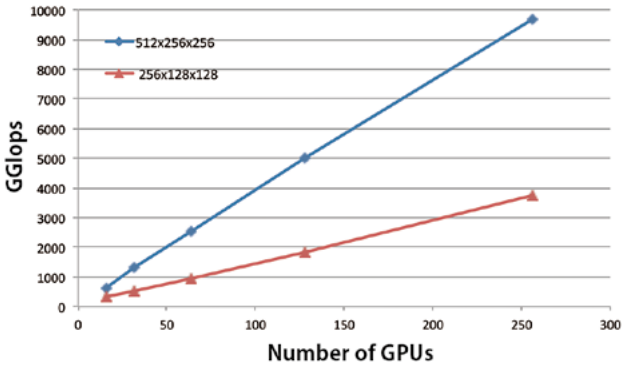


図3 Diffusionのウィークスケーリング性能

図4はSeismicのウィークスケーリング評価結果である。各GPUは2563の領域を計算する。Diffusionとは異なりX軸およびY軸にて領域分割を行った。そのためストライドアクセスをともなう境界領域交換が発生し、その結果としてDiffusionと比較してスケーラビリティが劣ることがわかる。また64GPUを用いた場合に性能低下が見られるがその原因についてはより詳細な解析が必要である。

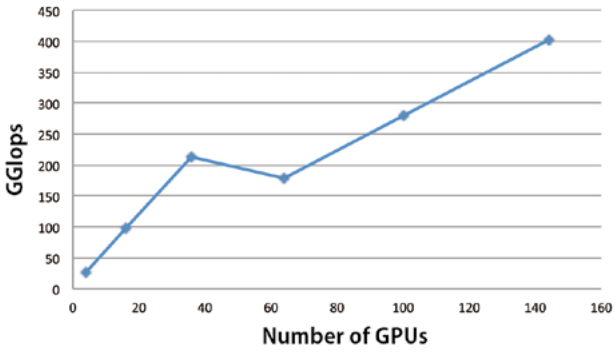


図4 Seismicのウィークスケーリング性能

4-2 ストロングスケーリング

図5は問題サイズ512x512x4096のDiffusionの性能を示したものである。128GPUまで用い、1次元、2次元および3次元分割の場合を比較した。

1次元分割ではZ軸方向に均等に分割し、2次元分割ではY軸およびZ軸、3次元ではすべての軸で均等分割を行った。その結果、利用GPU数が増えるにつれて多次元分割が性能上優れていることがわかる。このような多次元分割の優位性自体はすでに広く知られたものであるが、本フレームワークを用いた場合は明示的に分割を行うプログラミングをせずにフレームワークによって自動的に処理される。

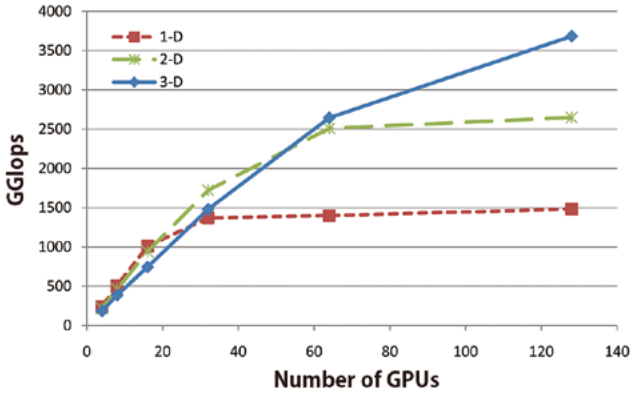


図5 512x512x4096のDiffusionの性能

5 関連研究

GPUアクセラレータを使った科学技術計算に関する研究が多く行われており、高速化だけでなく電力効率の向上が可能であることが示されている。しかしながらアクセラレータを使うことによりプログラミングの複雑さが増し、生産性の大幅な低下が大きな問題となっている。この問題に対して生産性向上を目的とした取り組みが行われている。

Mintは指示文によるステンシル計算フレームワークである[6]。ステンシルのループに対して指示文を追加することでループをGPU上で実行するコードに変換する。YpnosはHaskell言語に基づいたステンシル用DSLであり、GPU用に自動並列化を行う[7]。MintおよびYpnosともに我々と同じく自動並列化等の高生産性のための機能を実現しているが、双方とも単一GPUのみを対象としている点が我々と異なる

Listzは非構造格子計算のためのDSLである[8]。我々のフレームワークと同様に高い抽象度を持ったDSLであり、実際の実行環境上での実装方式はユーザプログラマからは隠蔽されている。これによって対象ドメインに特化した最適化を適用可能にしている。我々のフレームワークが対象とする構造格子に対してListzを適用することも可能である。しかしながらListzは非構造格子に最適化されているため構造格子において高い効率を達成するための最適化が可能かどうかは明らかではない。

6 おわりに

TSUBAME2.0のような大規模GPUスーパーコンピュータにおけるプログラミングの生産性を向上させるためにステンシル計算向けフレームワークPhysisを提案した。C言語に基づいたPhysis DSLによりステンシル計算を簡潔かつ実行環境独立に記述することを実現した。DSL変換器によって対象実行環境向けにコード生成を行い、通信と計算のオーバーラップなどの最適化を自動的に適用する。本稿ではプロトタイプ実装を行い、TSUBAME2.0の256GPUを用いた評価では良好な性能を達成できることを示した。今後はさらなる性能向上を目的としたフレームワークの拡張を施し、またアプリケーションケーススタディを通じて本フレームワークの有効性の実証を行う予定である。

謝辞

本研究の一部は科学技術振興機構CREST「高生産性・高性能アプリケーションフレームワークよるポストペタスケール高性能計算の実現」、科学研究費補助金(22700047)、JST-ANR FP3C、NVIDIA CUDA Center of Excellenceから支援を受けた。

参考文献

[1] N. Maruyama, T. Nomura, K. Sato, and S. Matsuoka, "Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers," In Proceedings of 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), Seattle, WA, USA (2011)

[2] T.Shimokawabe, T.Aoki, C.Muroi, J.Ishida, K.Kawano, T.Endo, A.Nukada, N.Maruyama, S.Matsuoka, "An80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code" in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, (SC'10), New Orleans, LA, USA (2010)

[3] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. M. W. Hwu, "Optimization principles and application performance evaluation of a multi- threaded GPU using CUDA," In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, PPoPP '08, zpages 73–82, (2008)

[4] M. Fowler. Domain-Specific Languages. Addison- Wesley Professional (2010)

[5] M. Schordan and D. Quinlan, "A source-to-source architecture for user-defined optimizations," In Modular Programming Languages, volume 2789 of Lecture Notes in Computer Science, pages 214–223. Springer Berlin / Heidelberg (2003)

[6] D. Unat, X. Cai, and S. B. Baden. "Mint: realizing CUDA performance in 3D stencil methods with annotated c," In Proceedings of the International Conference on Supercomputing (ICS'11), ICS '11, pages 214–224 (2011)

[7] D. A. Orchard, M. Bolingbroke, and A. Mycroft. "Ypnos: declarative, parallel structured grid programming," In DAMP '10: Proceedings of the 5th ACM SIGPLAN workshop on Declarative aspects of multicore programming, pages 15–24 (2010)

[8] H. Chafi, Z. DeVito, A. Moors, T. Rompf, A. K. Su- jeeth, P. Hanrahan, M. Odersky, and K. Olukotun, "Language virtualization for heterogeneous parallel computing," In Proceedings of the ACM international conference on Object oriented programming systems languages and applications, OOPSLA '10, pages 835–847 (2010)

FTI: ヘテロジニアススパコン向け耐障害インタフェース

～100TFlops超　東北地方太平洋沖地震シミュレーション～

Leonardo Bautista-Gomez*　Dimitri Komatitch**　丸山 直也*　坪井 誠司***
Franck Cappello†　松岡 聡‡　中村 武***
* 東京工業大学・情報理工学研究科　** University of Toulouse, Observatoire Midi-Pyrénées
*** JAMSTEC　† University of Illinois, INRIA　‡ 東京工業大学・学術国際情報センター

TSUBAME2.0のようなペタフロップ級の巨大なシステムは、計算資源の障害も無視することはできない。
長時間にも及ぶ大規模科学シミュレーションを行うためには、たとえ障害が発生したとしても計算を続ける必要がある。
我々は、地震波シミュレーションにおいて、リード・ソロモン・エンコーディングを利用することにより、システムの信頼性を保障しながら100 TFlops以上の性能をTSUBAME2.0で実現した。

1 はじめに

スーパーコンピュータなどのHPCシステムは、本来高信頼に設計されるが、近年、システムを構成する機器の増大や複雑化により、障害発生率が増えてきており、現在、ペタスケールシステムでは、その平均故障間隔(MTBF: Mean Time Between Failures)が数日となっている[11]。そのため、長時間に及ぶ大規模科学計算を行えるよう、耐障害性の高いシステムの構築がより一層重要となっている。チェックポイント/リスタート(CR)は、耐障害手法として、広く利用されている。これは、実行中のアプリケーションの状態をチェックポイントと呼ばれるファイルとして並列ファイルシステムなどの高信頼ストレージに保存しておき、障害が発生したときに、最新のチェックポイントから計算を再開させる手法である。

大規模科学アプリケーションを対象とした場合、その状態を保存するためには、数万オーダーに及ぶ数のプロセスが、それぞれ数GBのチェックポイントデータを書き込むため、合計数十TBのデータが書き込まれることとなる。しかし、並列ファイルシステムの低いI/Oバンド幅を考えるとチェックポインティングに要する時間が爆発的に増える。実際に、現在のペタスケールシステムにおいて、大規模科学アプリケーションを持続的に実行できる頻度でチェックポイントを取った場合、その25%の時間をチェックポイントに費やしてしまう事例もある[9]。

さらに、将来のポストペタ・エクサスケールシステムでは、構成する機器の数がさらに増加すると予見される。それに伴い、システム全体の障害発生率が上昇し、MTBFが数分足らずになると予想されている。このため、より高い頻度でチェックポイントを取る必要があるが、チェックポインティングに費やす時間が膨大となってしまうため、ポストペタ・エクサスケールシステムではアプリケーションの計算が実質的に進まなくなると危惧されている。

2 リード・ソロモン符号を利用したチェックポイント

このため、以前よりマルチレベル・チェックポイント/リスートという手法が提案されている。これは、チェックポイント先として並列ファイルシステムだけでなく、計算ノード上のメモリ領域やローカルストレージを活用する手法である。実際に、チェックポイントを保存する際には、並列ファイルシステムより高いI/Oバンド幅をもつメモリ領域やローカルストレージに対して、高い頻度でチェックポイントを保存し、低いバンド幅ではあるが、信頼性の高い並列ファイルシステムに対しては、それより低い頻度でチェックポイントを保存することにより、信頼性と性能を両立させる手法である。実際、年々増加するプロセッサの高い集積度に起因するソフトエラー率(SER: Soft Error Rate) が増えると報告されている[4,10]が、このような、軽度な障害は計算ノード上のメモリ領域やローカルストレージに保存されたチェックポイントから計算をリスタートさせることが可能である。不揮発性メモリである相変化メモリ(PRAM: Phase-change RAM)[2] やTSUBAME2.0にも搭載されているSSD (Solid State Drive)[6]が、このような用途に最適な保存領域となる。 一方、チェックポイントの複製やパリティを他の計算ノードに分散させて保存することにより、計算ノード単体の故障が起きた時に、故障ノードのチェックポイントを新しく割り当てられた計算ノード上へ転送することにより、その計算ノード上で計算をリスタートすることが可能である[1, 7]。しかし、一般にマルチレベルチェックポイント・リスタートでは、複数の計算ノードが一度に故障する場合に備えて、やはりある程度の頻度でチェックポイントを並列ファイルシステムに保存する必要がある。しかし、先に述べたようにそれに起因するオーバーヘッドは大きいという問題が残る。

そこで我々は、トポロジー情報から計算ノードをグルーピングし、各々のグループ内で、リード・ソロモン・エンコーディング (RS Encoding: Reed-Solomon Encoding)を用いたチェックポイントの符号化を適用することにより、並列ファイルシステムへチェックポイントを保存することなく、複数の計算ノードの障害にも耐え得る耐障害インタフェース(FTI: Fault Tolerance Interface)を提案する。しかし、RS Encodingの複雑性により、XORによるパリティ符号などの既存の

手法よりエンコーディングに多くの時間を要する。一方、多くのGPUアプリケーションでは、ほとんどの計算をGPU上で行うため、いくつかのCPUコアは使われていない。そこで、この使われていないCPUコアをエンコーディング専用のコアとして使用することにより、エンコーディングの時間を隠蔽する。我々は、性能モデルに基づき、チェックポイントサイズ、グループサイズ違いによる、エンコーディング/デコーディング時間への影響やスケーラビリティの検証を行い、このFTIを2011年3月11日東北地方太平洋沖地震のシミュレーションに適用した。この計算において1000個のGPUを使用し100 TFlops 以上の性能を達成し、実アプリケーションにおいて、低オーバーヘッドかつ高信頼なシミュレーションを実現した。

100TFlops 超 高信頼 SPECFEM3D 3

スケーラビリティ及び効率性を検証するために、FTIをSPECFEM3Dに組み込み、評価を行った。SPECFEM3Dは地震波の伝搬をシミュレートするアプリケーションであり、世界300以上の研究グループで使用されている。このアプリケーションは、2002年時点で、世界1位であった地球シミュレータ上でCPU 1944個を使用した計算で5 TFlops を出し、その後、2003年Gordon Bell Super Computing Award for Best Performance[8]を受賞したアプリケーションである。さらに、SPECFEM3DはTSUBAME2.0のようなCPU/GPUハイブリット型システムの性能を十分発揮できるように、CUDA[12, 13]による、GPUへ移植がなされた。一般に、このような地震波伝搬シミュレーションでは、単精度浮動小数点が用いられており、また、このGPU版SPECFEM3Dは、有限差分法や有限要素法の計算のようにメモリバウンドな計算特性を持つため、性能がメモリバンド幅律速になることを強調しておく。

TSUBAME2.0を使用した評価を行い、ストロング/ウィークスケールさせたときの、SPECFEM3D の性能を検証する。TSUBAME2.0のアーキテクチャ及び性能を表1に記す。各評価では、SPECFEM3Dの30~40分実行時間を要した。

まず、ストロングスケーリングの結果を図1に示す。ストロングスケールでは、固定された問題サイズに対し、GPUの数を増加させるため、1GPUあたりのチェックポイントサイズは減少する。このため、チェックポイントサイズに応じて、チェックポインティングの間隔を短くし、障害が発生後にリスタートする際、障害発生時点までの復旧時間を短くする。この評価では、チェックポイントを行わない場合(No ckpt.)、FTIを用いてチェックポイントを行う場合 (FTI ckpt.)、並列ファイルシステム Lustre 上にチェックポイントを保存する場合 (Lustre ckpt)の3つの場合を比較した。また、FTIはアプリケーションレベルでチェックポイントを行うため、リスタートに必要なデータのみ保存する。そのデータ量はアプリケーションで使用するメモリ量の20%に相当する。図1より、SPECFEM3D では、5 TFlops (48 GPUs) から 23 TFlops (384 GPUs) へストロングスケールしており、FTIを使用した場合でも、わずか4%ほどのオーバーヘッドでストロングスケールすることを確認した。一方、Lustre へチェックポインティングでは、I/Oバンド幅がボトルネックとなり、スケールしない。Flops 値は5回の計測で得られた実行時間の平均とPAPI[14]から得られた浮動小数点演算数から求めた。

次に、1000 GPU までウェークスケールさせ、より大規模な問題サイズで評価を行った。Fermi GPU[5]では12.5%のメモリ容量は ECCに使用されるため、1GPU で使用可能な2.6GBのオンボードメモリのうち、2.1GB をアプリケーションの実行に使用した。また、チェックポイントの間隔は、Youngの性能モデル から計算される最適な時間間隔 6分に固定した。この評価では、チェックポイントを行わない場合 (No ckpt.)、ローカルの SSD にチェックポイントを保存する場合 (L1)、L1 に加え、2回に1度FTIを使用しRS Encodingを行う場合 (FTI-L1,L2)、FTI-L1,L2に加え、6回に1度、チェックポイントをLustreへ保存する場合 (FTI-L1,L2,L3)、最後にBLCRを用いてLustreへ保存する場合 (BLCR+Lustre)の5つの場合を比較した。BLCRでは、GPUアプリケーションのチェックポインティングはサポートしていないため、各々のプロセスが2.1GBのデータを書き込むようにし、擬似的にチェックポイントを行わせた。また、BLCRはカーネルレベルのチェックポイントを取るため、アプリケーションが使用するメモリ領域全てがチェックポイントとして保存される。SPECFEM3Dにおいて、これは、アプリケーションレベルで保存されるチェックポイントデータの5倍のサイズにあたる。図2にその結果を示す。Flops値は3回の計測で得られた結果を平均した値である。SPECFEM3Dでは、43 TFlops (384GPUs)から117TFlops (1152GPUs) へウィークスケールしており、L1、FTI-L1、L2 の場合でも、およそ8%のオーバーヘッドでウィークスケールすることを確認した。FTIでは、アプリケーションで使用されていないCPUコアをエンコーディングに使用しているので、エンコーディングによるオーバーヘッドを低く抑えることができる。FTI-L1、L2、L3では、さらに3% のオーバーヘドが生じた。これは、Lustreへのチェックポインティングに起因する。一方、BLCR+Lustreではそのチェックポイントサイズと Lustreへのチェックポインティングが原因でウェークスケールしていない。また、この評価では、6分間隔でチェックポイントを行ったが、このような短い時間間隔でチェックポインティングを行った場合でも、実アプリケーションにおいて、100 TFlops 以上の性能を達成している。

FTI:ヘテロジニアススパコン向け耐障害インタフェース ～100TFlops超 東北地方太平洋沖地震シミュレーション～

Nodes	1408 High BW Compute Nodes
CPU	2 Intel Westmere-EP 2.93GHz 12Cores/node
Mem	55.8GB or 103GB (Total: 80.55TB)
GPU	NVIDIA M2050 515GFlops, 3GPUs/node (Total: 4224 NVIDIA Fermi GPUs)
SSD	60GB x 2 = 120GB (55.8GB node) Write speed : 360MB/s (RAID0) 120GB x 2 = 240GB (103GB node) (Total : 173.88TB)
Network	Dual rail QDR IB (4GB/s x 2)
File system	5 DDN DFA10000 units (3 Lustre and 2 GPFS) with 600 2TB HDDs each Measured Lustre write troughput (10GB/s)
OS	Suse Linux Enterprise + Windows HPC

表1 TSUBAME2.0アーキテクチャ

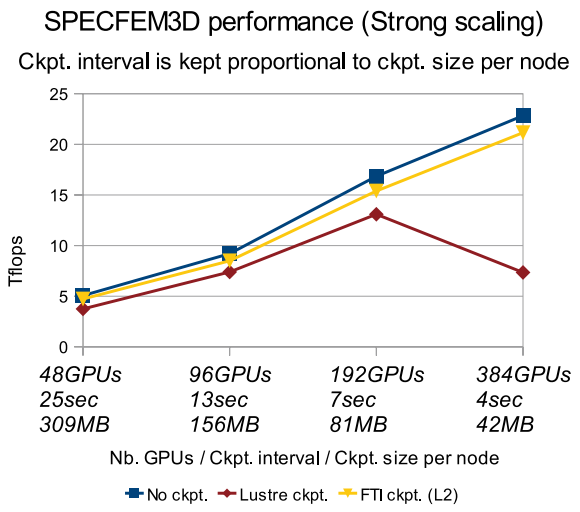


図1 SPECFEM3D performance (Strong scaling)

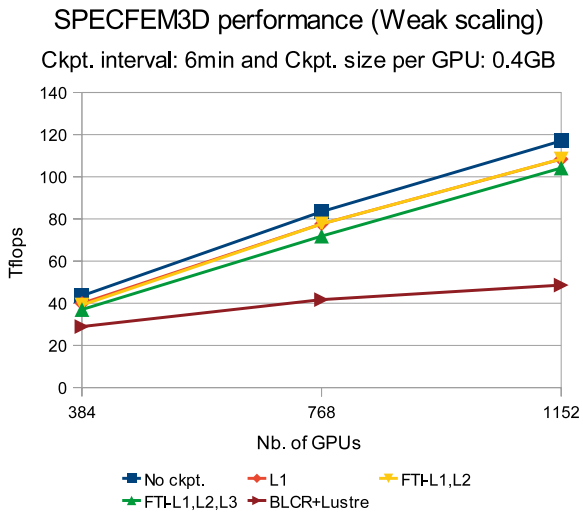


図2 SPECFEM3D performance (Weak scaling)

4 東北地方太平洋沖地震シミュレーションへの適用

我々は、FTIを組み込んだSPECFEM3Dを用いて、東北地方太平洋沖地震のシミュレーションを試みた。このシミュレーションでもTSUBAME2.0を活用する。地震記象(地震計に記録された地震動の波形記録）は、震央距離が30° ~100°の範囲のIRIS GSN及びIFREE OHP 広域地震計で記録された波形データを使用し、その地震記象に対しバンドパスフィルターをかけ0.002Hz~1Hzの周波数帯の波形を取り出す、その後、中村ら[16]と同様の方法で、震源における滑り分布を得るために、非負の拘束条件下で波形インバージョンを適用した。震源や観測点付近の波動場の計算には、IASP91の伝達速度構造モデルを使った。また、CMT（Centroid Moment Tensor）モデルに基づき、走向をN201° E 、断層角を9°と設定し、断層長を余震の発生分布から440kmと仮定する。

上記の条件下で、波形インバージョンを行った結果、震源から北東及び南西100km付近に40mほどの逆断層によるずれあり、また北東と南西に向かって、バイラテラル破壊（中央から外側へ進む断層破壊）が発生したことがわかった。また、同様の条件下で、スペクトル要素法[15, 16]を用いて福島県の広野町の地震計における、理論地震記象を計算した。その理論地震記象を図3、4、5に示す。これらの図から、今回の地震により、この地震計付近では、南へ2m、東へ2m、鉛直方向に1m 静的変位したことがわかる。これは観測された地殻変動の結果とも概ね一致している。

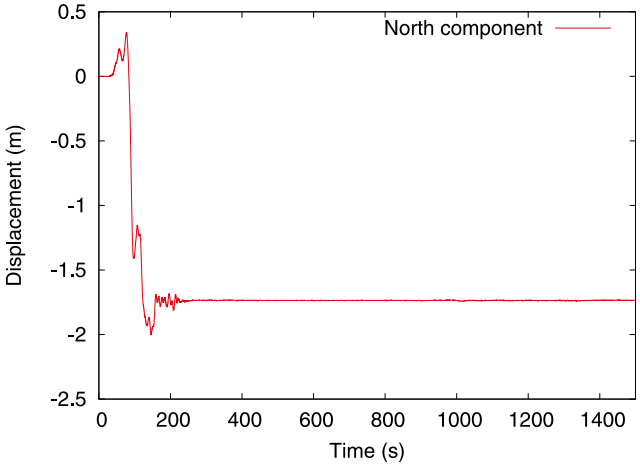


図3 地殻変動（北方向）

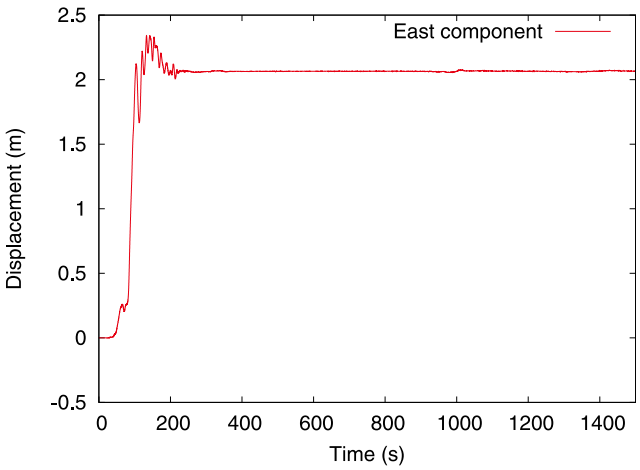


図4 地殻変動（東方向）

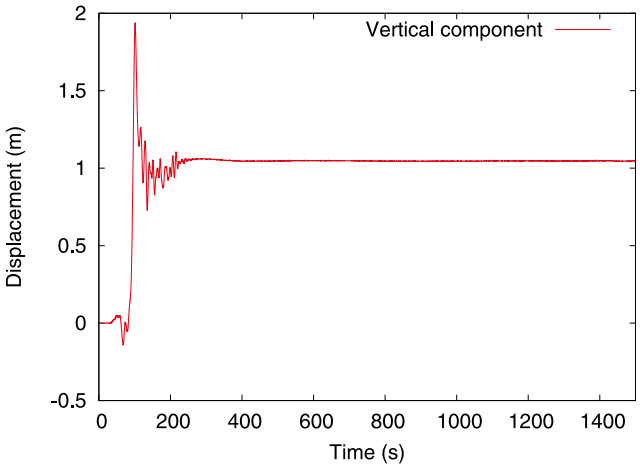


図5 地殻変動（鉛直方向）

おわりに　5

リード・ソロモン・エンコーディングを用いた耐障害インタフェースFTIを紹介した。エンコーディング専用のスレッドを割り当て、エンコーディング時間を隠蔽することによりSPECFEM3Dにおいて100 TFlops以上の性能を達成した。また、FTIを組み込んだSPECFEM3Dを用いて、東北地方太平洋沖地震のシミュレーションを行い、実アプリケーションの高信頼計算を実現した。

謝辞

本研究の一部はJSPS、ANR/JST FP3Cプロジェクト、INRIA-Illinois Joing Laboratory for Petascale Computing から支援を頂いた。記して謝意を表す。

参考文献

[1] A. Moody, G. Bronevetsky, K. Mohror, B. R. de Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, 2010

[2] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, Y. Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Portland, 2009.

[3] Z. Cheng, J. Dongarra, A scalable Checkpoint Encoding Algorithm for Diskless Checkpointing. Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium, HASE 2008, Nanjing, China, December, 2008.

[4] B. Schroeder, E. Pinheiro, W. Weber. DRAM errors in the wild: A Large-Scale Field Study. In Proceedings of the 11th international joint conference on Measurement and modeling of computer systems (SIGMETRICS), ACM, New York, NY, USA, 2009.

[5] NVIDIA Corporation http://www.nvidia.com/object/fermi_architecture.html

[6] A. Moody, G. Bronevetsky, Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O. DOE technical report, 2009

[7] W. D. Gropp, R. Ross, and N. Miller. Providing efficient I/O redundancy in MPI environments. Lecture Notes in Computer Science, 3241:7786, September 2004

[8] D. Komatitsch, S. Tsuboi, C. Ji and J. Tromp, A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator, Proceedings of the ACM / IEEE Supercomputing SC'2003 conference, November 2003.

[9] G. Grider, J. Loncaric, and D. Limpart, Roadrunner System Management Report, Los Alamos National Laboratory, Tech. Rep. LA-UR-07-7405, 2007.

[10] S. Y. Borkar, Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, IEEE Micro, vol. 25, no. 6, pp. 10-16, 2005.

[11] D. Reed, High-End Computing: The Challenge of Scale, Director's Colloquium, LANL, May 2004.

[12] D. Komatitsch, D. Michéa, G. Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, Journal of Parallel and Distributed Computing, vol. 69(5), p. 451-460, doi: 10.1016/j.jpdc.2009.01.006, 2009.

[13] D. Komatitsch, G. Erlebacher, D. Göddeke, D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, Journal of Computational Physics, vol. 229(20), p. 7692-7714, doi:10.1016/j.jcp.2010.06.024, 2010.

[14] PAPI: Performance Application Programming Interface, http://icl.cs.utk.edu/papi/

[15] D. Komatitsch, J. Ritsema, J. Tromp, The spectral-element method, Beowulf computing, and global seismology, Science 298, 1737-1742, 2002.

[16] T. Nakamura, S. Tsuboi, Y. Kaneda, Y. Yamanaka, Rupture process of the 2008 Wenchuan, China earthquake inferred from teleseismic waveform inversion and forward modeling of broadband seismic waves, Tectonophysics, vol. 491, 72-84, 2010.

[17] S. Tsuboi, D. Komatitsch, C. Ji, J. Tromp, Broadband modelling of the 2002 Denali fault earthquake on the Earth Simulator, Phys. Earth Planet. Inter. 139, 305-312, 2003.

● TSUBAME e-Science Journal No.5
2012 年 2 月 28 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　ピパットポンサー・ティラポン
　　渡邊寿雄　佐々木淳　仲川愛理
住所：　〒 152-8550 東京都目黒区大岡山 2-12-1-E2-1
電話：　03-5734-2087　FAX：03-5734-3198
E-mail：tsubame_j@sim.gsic.titech.ac.jp
URL：　http://www.gsic.titech.ac.jp/

TSUBAME

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 2.4PFlops、18000CPUコア、4300GPU搭載
世界トップクラスの東工大のスパコンTSUBAME2.0を
東工大以外の皆さまにご利用いただくための枠組みです。**

課題公募する利用区分とカテゴリ

共同利用サービスには、「学術利用」、「産業利用」、「社会貢献利用」の3つの利用区分があり、さらに「成果公開」と「成果非公開」のカテゴリがあります。
ご利用をご検討の際には、下記までお問い合わせください。

TSUBAME 共同利用とは…

他大学や公的研究機関の研究者の**学術利用**[有償利用]

民間企業の方の**産業利用**[有償・無償利用]

その他の組織による社会的貢献のための**社会貢献利用**[有償利用]

共同利用にて提供する計算資源

共同利用サービスの利用区分・カテゴリ別の利用課金表を下記に示します。TSUBAME 2.0における計算機資源の割振りは口数を単位としており、1口は標準1ノード(12 CPUコア, 3GPU, 55.82GBメモリ搭載)の3000時間分(≒約4ヵ月)相当の計算機資源です。
1000 CPUコアを1.5日利用する使い方や、100 GPUを3.75日利用する使い方も可能です。

利用区分	利用者	制度や利用規定等	カテゴリ	利用課金
学術利用	他大学または研究機関等	共同利用の利用規定に基づく	成果公開	1口：100,000円
産業利用	民間企業を中心としたグループ	「先端研究施設共用促進事業」に基づく	成果公開	トライアルユース(無償利用)
				1口：100,000円
			成果非公開	1口：400,000円
社会貢献利用	非営利団体、公共団体等	共同利用の利用規定に基づく	成果公開	1口：100,000円
			成果非公開	1口：400,000円

産業利用トライアルユース制度(先端研究施設共用促進事業)

東工大のスパコンTSUBAMEを、より多くの企業の皆さまにご利用いただくため、初めてTSUBAMEをご利用いただく際に、無償にてご試用いただける制度です。

(文部科学省 先端研究施設共用促進事業による助成)

詳しくは、下記まで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　kyodo@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