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 (June 2010)

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——40 GPUで5 TFLOPSの実効性能

小川　慧[†1]　青木尊之[†2]　山中晃徳[†3]

純金属のデンドライト凝固成長に対してフェーズフィールド法に基づいたAllen-Cahn方程式と熱伝導方程式を連立させてGPUで計算し実行性能を検証した．離散化された式をCUDAでプログラミングすることにより時間発展の計算を行い，NVIDIAのGPUであるTesla S1070の単一GPUで171 GFLOPS（単精度計算）を達成した．さらに複数GPUに対して領域分割による並列化を行い，東京工業大学のTSUBAMEにおいて実行性能のスケーラビリティーを調べた．各GPUに直結したビデオメモリ間のデータ通信が必要となり，CPU側にバッファ・メモリを用意してGPUとバッファ・メモリの間のデータ転送，複数ノードに分散したバッファ・メモリ間のデータ交換の2段階で通信を行った．GPUの演算性能に対して通信性能が低く，データ転送に必要な部分を先に計算することでデータ転送を開始し，同時に内部領域の計算を行うオーバーラップ計算を行った．これによりストロング・スケーリングの大幅な向上を図ることができ，$1920 \times 1920 \times 1920$格子に対し40 GPUを用いて5 TFLOPSを達成した．計算時間と通信時間の内訳について各部分にかかる時間を詳細に測定し，通信時間の短縮方法を明らかにした．本研究により，大規模なHPCアプリケーションを複数GPUで実行するための方向性を示すことができた．

Multi-GPU Scalability of Phase-Field Simulation for Phase Transition——5 Tera Flop/s Performance on 40 GPUs

Satoi Ogawa,[†1] Takayuki Aoki[†2] and Akinori Yamanaka[†3]

Multi-GPU computing is carried out for the dendritic solidification of a pure metal. The simulation code for the Allen-Cahn equation coupled with the thermal conduction equation is implemented to the CUDA code and runs on the NVIDIA Tesla GPU of TSUBAME grid cluster. The performance of 171 GFLOPS (Single precision) for single GPU has been achieved and 40 GPUs have shown 5 TFLOPS for the computational domain of $1920 \times 1920 \times 1920$ mesh. For a large-scale multi-GPU computing, GPU-to-GPU communications become major time consuming and the overlapping technique between the communication and computation is introduced to hide the communication time. The strong scaling of the GPU number has been improved very much in the case that the computational time is longer than communication time. The breakdown of execution time is examined carefully and the critical issues for HPC application on multi-GPU platform become clear.

1. 緒　　言

金属材料の機械的強度や特性はミクロの組織的構造に基づくため，より高性能な材料を得るためには組織制御方法の開発が求められている．近年，材料の相転移や相分離などのミクロなダイナミクスの解明にフェーズフィールド法が注目されている．フェーズフィールド法から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・有限要素法などで解かれることが多い．しかし，複雑な非線形項を多く含むために1格子点あたりの演算量が非常に多く，また数値計算の安定性から時間ステップを小さくとらなければならず，計算負荷が大きいという欠点があった．これに対し，陰解法で時間ステップを大きくとることや，界面で格子を細分化させるAMR（Adaptive Mesh Refinement）法などが提案され有効であることが示されている[1]が，適用できる問題が限られているという問題点を持っている．

HPC分野において，アクセラレータ技術の適用が注目されている．すでにClear Speedを用いた東京工業大学のスーパーコンピュータTSUBAMEの例[2]や，Cellを導入してTOP500ランキングで世界最速のロス・アラモス国立研究所のRoadrunnerの例[3]がある．近年，GPU（Graphics Processing Unit）を汎用計算に使用する（GPGPU）研究が行われはじめ，2006年にNVIDIA社からGPGPU用のSDKであるCUDA[4]がリリースされて以降，さまざまな分野でGPUを使った研究[5)–7)]がよりいっそう進められてきた．2008年

†1 東京工業大学大学院理工学研究科原子核工学専攻
Graduate School of Science and Engineering, Tokyo Institute of Technology
†2 東京工業大学学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
†3 東京工業大学大学院理工学研究科機械制御システム専攻
Graduate School of Science and Engineering, Tokyo Institute of Technology

には TSUBAME に 680 個の GPU が追加され，LINPACK 性能の向上が報告された例[8]，理研ベンチマークテストでの高い演算性能が報告された例[9),10)] や非圧縮性流体計算の高速化の研究[11)] があり，次世代の計算資源として期待されている．

本論文ではフェーズフィールド法に基づいた Allen-Cahn 方程式と熱伝導方程式を連立させて解く[12)] ことにより，純金属の過冷却凝固デンドライト成長の計算を行う．CUDA を用いて有限差分法で離散化された式をプログラミングし，TSUBAME の GPU 上で計算することにより，CPU と比較して圧倒的に高速に計算できることを示す．従来の GPU 計算の研究が単一 GPU を利用するものが多かったのに対し，本研究では 1 つの GPU 上のメモリには載らないような大きな規模の計算に対し，領域分割法で並列化するとともに，複数ノードに搭載された GPU を使う場合のスケーラビリティーと，GPU 間のデータ通信の計算とのオーバーラップの効果を調べた．

2. デンドライト成長の支配方程式と計算条件

2.1 フェーズフィールド法

フェーズフィールド法では，秩序変数として ϕ を導入し，固相部分に $\phi = 1$，液相部分に $\phi = 0$ と設定する．界面領域は 0 から 1 へと急峻かつ滑らかに変化させた拡散界面として扱い，$\phi = 0.5$ を界面として扱う．フェーズフィールド法では従来使われていた界面追跡法など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．

2.2 純金属のデンドライト成長のモデル

本研究で対象とする純金属のデンドライト凝固成長ではフェーズフィールド法から導出される Allen-Cahn 方程式と熱伝導方程式を解く．界面エネルギーの異方性を考慮した ϕ に対する方程式として式 (1) を用いる．

$$\begin{aligned}\frac{\partial\phi}{\partial t} = M\Bigg[& \frac{\partial}{\partial x}\left(\epsilon^2\frac{\partial\phi}{\partial x} + \epsilon\frac{\partial\epsilon}{\partial\phi_x}|\nabla\phi|^2\right) \\ & + \frac{\partial}{\partial y}\left(\epsilon^2\frac{\partial\phi}{\partial y} + \epsilon\frac{\partial\epsilon}{\partial\phi_y}|\nabla\phi|^2\right) \\ & + \frac{\partial}{\partial z}\left(\epsilon^2\frac{\partial\phi}{\partial z} + \epsilon\frac{\partial\epsilon}{\partial\phi_z}|\nabla\phi|^2\right) \\ & + 4W\phi(1-\phi)\left\{\phi - \frac{1}{2} + \beta + a\chi\right\}\Bigg] \end{aligned} \tag{1}$$

式 (1) 中，β は式 (2)，ϵ は式 (3) である．

$$\beta = -\frac{15L}{2W}\frac{T - T_m}{T_m}\phi(1-\phi) \tag{2}$$

$$\epsilon = \bar{\epsilon}\left(1 - 3\gamma + 4\gamma\frac{\phi_x^4 + \phi_y^4 + \phi_z^4}{|\nabla\phi^4|}\right) \tag{3}$$

ただし，L は潜熱，χ は $[-1.0, 1.0]$ の乱数，a は乱数の振幅，γ は異方性強度，T_m は融点，δ は界面厚さ，σ は界面エネルギー，λ は界面幅制御パラメータ，他定数は $b = \tanh^{-1}(1-2\lambda)$，$W = 6\sigma b/\delta$，$M = bT_m\mu/3\delta L$ である．

一方，凝固過程に最も関係する温度 T について，界面からの潜熱の発生を考慮した熱伝導方程式式 (4) を用いる．

$$\begin{aligned}\frac{\partial T}{\partial t} &= \kappa\nabla^2 T + \frac{L}{C}\frac{\partial h(\phi)}{\partial t} \\ &= \kappa\nabla^2 T + \frac{L}{C}\frac{\partial h(\phi)}{\partial\phi}\frac{\partial\phi}{\partial t} \\ &= \kappa\nabla^2 T + 30\phi^2(1-\phi)^2\frac{L}{C}\frac{\partial\phi}{\partial t}\end{aligned} \tag{4}$$

ここで，κ は熱伝導率 K，比熱 c の比であり $\kappa = K/c$ で表される熱拡散係数である．また，L は潜熱である．$h(\phi)$ は，界面が移動した際（Phase Field ϕ が 0 から 1 に変化した際）に，そこで潜熱が発生するため導入する関数であり，式 (5) を満たす必要がある．

$$\int_0^1 \frac{\partial h(\phi)}{\partial\phi}d\phi = 1 \tag{5}$$

$h(\phi)$ の関数系は任意であるがここでは式 (6) とした．

$$\begin{aligned}h(\phi) &= p(\phi) = \phi^3\left(10 - 15\phi + 6\phi^2\right) \\ \frac{\partial p(\phi)}{\partial\phi} &= 30\phi^2(1-\phi)^2\end{aligned} \tag{6}$$

計算領域に対して等間隔直交格子を用い，式 (1) および式 (4) とも空間微分項が数値流束の発散形式に書け，有限差分法を用いて数値流束を格子点間で求めることにより空間 2 次精度の離散化を行う．時間積分には空間微分の階数が 2 次で時間微分の階数が 1 次であるため，1 次精度の前進オイラー法を用いる．格子点 (i, j, k) において，n 時間ステップの値から $n+1$ 時間ステップの値を計算するために参照する隣接格子点（ステンシル）は**図 1** となる．(i, j, k) に対して斜め方向の参照（$(i+1, j+1, k)$ など）が発生する．

初期条件として計算領域中央の小さな領域に初期生成核を設定し，時間発展により生成核を中心に凝固して結晶が生成する様子をシミュレーションすることができる．各定数は以下を用

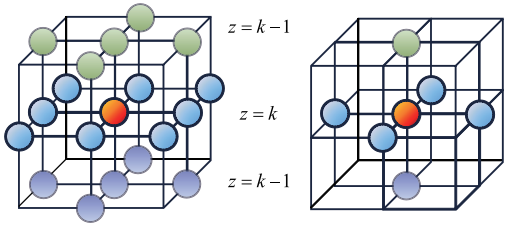
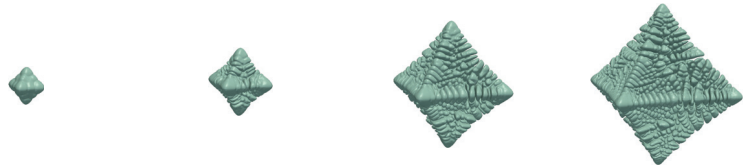


図 1　離散化式の隣接格子点参照（フェーズフィールド方程式（左図），熱伝導方程式（右図））
Fig. 1　The patterns of the neighbor grid access in the discretized Phase-Field equation (Left) and thermal conduction equation (Right).



(a) $t = 3.44 \times 10^{-8}$　(b) $t = 6.88 \times 10^{-8}$　(c) $t = 1.38 \times 10^{-7}$　(d) $t = 2.06 \times 10^{-7}$

図 2　純金属のデンドライト成長シミュレーションの結果
Fig. 2　Snapshots of the dendrite solidification solved by the Phase-Field equation for mesh.

いた．$\epsilon = 0.37$，$\mu = 2.0$，$T_m = 1728$，$L = 2.35 \times 10^9$，$C = 5.42 \times 10^6$，$\kappa = 1.55 \times 10^{-5}$．初期条件として以下を用いた．$T_i = 1511$，$\gamma = 0.05$，$\delta = 0.08$，$\gamma = 0.1$，$a = 0.025$ とする．格子点数 $1024 \times 1024 \times 1024$ に対するデンドライト（樹枝）凝固成長の計算結果を**図 2** に示す．図は $\phi = 0.5$ の等値面を表している．

3. 単一 GPU による計算

3.1 利用する計算機環境の概要

本研究では東京工業大学学術国際情報センタースーパーコンピュータ TSUBAME Grid Cluster を利用する．各ノードは Sun Fire X4600（AMD Opteron 2.4 GHz 16 cores, 32 GByte）であり，10 Gbps の SDR Inifinband でインターコネクトされている．GPU 計算のために使用するノードには，NVIDIA Tesla S1070（動作周波数：1.44 GHz．VRAM 4 GByte）のうち 2 つの GPU が PCI-Express Gen1.0 × 8 で接続されている．S1070 GPU のコアである T10P にはストリーミングマルチプロセッサ（SM）が 30 個含まれる．SM には 8 個のスカラープロセッサ（SP）と，SP から高速なアクセスが可能な 16 Kbyte のシェアードメモリがある．GPU のピーク浮動小数点演算性能は 1036 GFLOPS に達する．一方，TSUBAME の各ノードの Opteron CPU は，1 コアあたり 4.8 GFLOPS である．各 GPU からビデオメモリへのアクセスは SM から 512 bit のメモリインターフェースを介して行われ，ピークメモリバンド幅は 102 GByte/sec となる．Opteron CPU（6.4 GByte/sec. DDR-400）に比べると非常に高速である．実行した環境では CUDA 2.2 対応 Runtime, NVIDIA Kernel Module 185.18.14 がインストールされ，OS は SUSE Enterprise Linux 10 であった．CUDA では多数の軽量なスレッドが起動される．スレッドを束ねる 1 つめのまとまりはブロックと呼ばれ，束ねるスレッドの数を 3 次元的に指定する．たとえば，(tnx, tny, tnz) などのように指定すると，$tnx \times tny \times tnz$ 個のスレッドが指定される．各スレッドは 3 次元的な ID を持つ．ブロックはさらに束ねられてグリッドを構成する．GPU で実行するプログラムのことを CUDA ではカーネル関数と呼ぶが，カーネル関数の 1 回の呼び出しに対して 1 つのグリッドが対応する．グリッド内のブロックの構成は $(bnx, bny, 1)$ などのように 2 次元的に与え，$bnx \times bny \times 1$ 個のブロックが構成される．各スレッドは属しているブロックの 2 次元的な ID も持つ．スレッドやブロックの ID などはビルトイン変数としてスレッドの起動時にセットされる．

3.2 単一 GPU による計算と高速化技術

GPU 計算に対して CUDA を用いてプログラミングを行った．CUDA は NVIDIA 社が提供している GPU プログラミング環境（主に C/C++言語）である．

時間発展に必要な n ステップと $n+1$ ステップの従属変数に対するすべての配列を CUDA でグローバルメモリと呼ばれる VRAM 上に確保する．時間発展の過程で，必要なときだけデータを CPU 側に転送するようにし，PCI-Express Bus を介した通信は可能な限り頻度を減らしている．

1 GPU が担当する計算領域の格子点数を $nx \times ny \times nz$ とする．それらを x 方向に L 分割，y 方向に M 分割，z 方向に N 分割すると，分割された小領域の格子点数は $MX \times MY \times MZ$ となる．ただし，$MX = nx/L$，$MY = ny/M$，$MZ = nz/N$ である．各々の小領域に対して CUDA のブロック内のスレッドを $(MX, MY, 1)$ として割り当てる．各スレッドでは，z 方向に MZ 個の格子点をループで計算する．高い実行性能を得るためには，問題サイズに合わせ適切な分割数を選ぶ必要があり，ここでは $MX = 64$，$MY = 4$ となるように起動した．

ϕ に対する式 (1) の右辺を計算するために，ブロック内のシェアードメモリ上に $(MX +$

2) × (MY + 2) の大きさの 3 つの配列 PR0，PR1，PR2 を確保する．MX と MY に 2 を加えているのは，小領域の境界スレッドが隣接格子点データにアクセスするための袖領域である．$\phi_{i,j,k}$ をスレッドの番号と格子点の番号を対応づけ，以下のように計算を行う．まず，グローバルメモリ上の配列から $\phi_{i,j,k-1}$，$\phi_{i,j,k}$ を PR0，PR1 に読み込む．次に $\phi_{i,j,k+1}$ を PR2 に読み込むと，隣接格子のデータは隣接スレッドがすでに読み込んでいるため，$\phi_{i,j,k}$ の計算に必要な 19 点がすべてシェアードメモリに格納される．ただし，境界スレッド用に，袖領域のデータはグローバルメモリから別途読み込んでおく必要がある．PR0，PR1，PR2 にアクセスすることで式 (1) の右辺の計算をすべて行うことができるようになる．z 方向のループによる計算に従い $\phi_{i,j,k+1}$ を計算するためには $\phi_{i,j,k+2}$ を PR3 に入れる代わりに，すでに計算が終了して不要となった PR0 に入れることができる．3 つのシェアードメモリの配列を使い回すことにより，小さなサイズのシェアードメモリで計算を行うことができる．

温度 T^n に関する計算でも同じようにシェアードメモリをキャッシュ的に使い計算を行う．ただし，式 (4) の右辺に ϕ^n の時間微分 $\partial\phi/\partial t|^n_{i,j,k}$ が現れるため，$\phi^n_{i,j,k} \to \phi^{n+1}_{i,j,k}$ の時間発展と $T^n_{i,j,k} \to T^{n+1}_{i,j,k}$ の時間発展のカーネル関数をフューズさせている．スレッド内で $\partial\phi/\partial t|^n_{i,j,k}$ を保持できるため，グローバルメモリを介することなく，効率良く温度場の計算に用いることができる．

3.3　単一 GPU による計算性能

GPU での計算に対する参照実装として CPU のコードも用意する．計算結果の比較，計算性能の比較以外に，1 格子点あたりの浮動小数点演算を PAPI（Performance API）[13] のハードウェア・カウンタで測定する目的もある．

Tesla S1070 の 1 つの GPU からアクセスできるメモリサイズが 4 GByte であるため，計算できる最大の問題サイズが 640 × 640 × 640 格子に制限される．1 GPU による計算において格子点数を変えながら，実行性能を測定すると，64 × 64 × 64 格子で 116.8 GFLOPS，128 × 128 × 128 格子で 161.6 GFLOPS，256 × 256 × 256 格子で 169.8 GFLOPS，512 × 512 × 512 格子で 168.5 GFLOPS，640 × 640 × 640 格子で 171.4 GFLOPS となる．CPU（Opteron 1 コア）での実行性能が 0.898 GFLOPS であるのに対し，640 × 640 × 640 格子の計算では 171 GFLOPS の性能が得られ，190 倍高速化されていることになる．GPU では整数演算も SP が処理を行うため，CPU コードで数えた浮動小数点演算数を基に経過時間を測定して実行性能を評価した．

本研究のフェーズフィールド法の計算においては，1 格子点あたりの浮動小数点演算が 373 回である．一方，シェアードメモリを使わないと 26 回のグローバルメモリへの読み込みと 2 回の書き込みの計 28 word のメモリアクセスがある．本研究では単精度計算を行っていて，すべての格子点で同じ計算を行うので，3.33 FLOP/Byte の演算密度がある．これに対し，ソフトウェア・マネージド・キャッシュとしてシェアードメモリを用いることで，袖領域を含まない格子点においては読み出しを 2 回に減らすことができ，メモリアクセスを 4 回に低減させることができる．その結果，演算密度を 23.31 FLOP/Byte にまで高めることができる．この値は一般的なステンシル計算である流体計算などと比較すると非常に大きく，計算インテンシブになっているため，GPU の高い演算性能を引き出すことができた．

3.4　単精度計算の妥当性

フェーズフィールド変数は 0 から 1 までの範囲でしか変化せず，フェーズフィールド変数および温度場ともに滑らかに変化するような空間プロファイルを持つ．本研究の計算はすべて単精度浮動小数点で行っているが，数値計算の観点から安定であり，倍精度浮動小数点で計算した結果と比較して最後の有効桁数の 1 つ前まで一致することを確認している．

4.　複数 GPU ノードによる高速化

4.1　複数ノードによる高速化

単一 GPU からアクセスできるグローバルメモリの量は現在，NVIDIA Tesla S1070 の 4 GByte が最大である．メモリに入りきらないような大規模計算および，ある格子点数に対して 1 GPU の高速化率よりもさらに高い高速化率を得るには複数ノードで複数 GPU を用いる GPU 単位での並列化が必要となる．3 章で述べたように CUDA による GPU 計算では，単一 GPU 内でもスレッドのブロックがあるので並列化の階層が 1 つ増えた多階層の並列化になる．本研究はフェーズフィールド方程式を直交格子上で計算する問題を対象としているため，GPU 計算でも領域分割による並列化が有効であることは自明である．複数 GPU による計算を行うには，隣接 GPU のメモリ上にある格子点へのアクセスをともなうため，GPU に搭載されたグローバルメモリ間のデータ転送が必要となる．現在の CUDA の仕様では異なる GPU のメモリ内の値を GPU 間で直接の通信ができない．そこで，**図 3** のように GPU に対応したバッファ・メモリを CPU 側に確保し，「GPU とバッファ・メモリ間のデータ転送」と「バッファ・メモリ間のデータ交換」を行う．

ノード内では OpenMP で通信し，ノード間では MPI（Message-Passing Interface）を使うハイブリッド通信も考えられるが，ここでは flat に MPI 通信を行い，使用する GPU の数と MPI のプロセス数は等しくなる．領域間のデータ転送量を減らすためには，計算全体の格子に対して 3 次元的な領域分割（サイの目型の分割）をする方が有利である．本論文

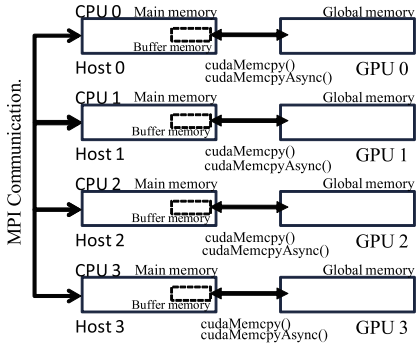


図 3　GPU 間通信の模式図
Fig. 3　GPU-to-GPU communication processes.

では，計算とのバランスを明らかにすることを目的とし，分割数により通信時間の変わらない z 軸方向に沿った 1 次元的な領域分割（短冊型分割）を行う．

4.2　通信と計算のオーバーラップ

複数 GPU を用いた計算では，各 GPU が割り当てられた小領域内のすべての格子の計算（時間発展）を行ってから，袖領域のデータ通信（「GPU とバッファ・メモリ間のデータ転送」と「バッファ・メモリ間のデータ交換」）を行うことが多い．本方法を「非オーバーラップ計算（Non-Overlapping）」と呼ぶことにする．一方，CPU による並列計算においても行われる技法である「計算と通信のオーバーラップ（Overlapping communication time with computation time）」を複数 GPU の計算にも適用することができる．オーバーラップ計算では，**図 4** に示すようにまず袖領域のデータとして交換される格子を先に計算する．計算後，CUDA の非同期実行機構（Asynchronous Concurrent Execution）を用い，stream を 2 つ生成し同時実行する．stream 0 では袖領域を除いた部分の格子点に対する計算を行い，stream 1 においては袖領域の通信を行う．GPU のグローバルメモリ上に確保したバッファ・メモリと，CPU のメインメモリ上に GPU との通信用に確保した pinned されたバッファ・メモリ間の通信には CUDA の非同期通信の API である cudaMemcpyAsync() を用いる．GPU から CPU への通信が完了後，CPU 側では GPU との通信バッファ・メモリ上から MPI 通信用の pinned されたバッファ・メモリへとコピーし，MPI を用いてプロセス間でデータを交換する．MPI 通信の完了後，逆の手順で CPU から GPU への通信を行う．stream 0 と stream 1 の通信と計算の両方の完了を待ち，最後に MPI_Barrier() によりプロセス間の大域的な同期をとることで 1 ステップの計算が完了する．

```
Comp_on_GPU<<<… >>>(…,OUTTER)

Comp_on_GPU<<<…, stream[0]>>>(…, INNER)
cudaMemcpyAsync(CUDA_buf0, f, …, stream[1])

cudaStreamSynchronize(stream[1])
copy_CUDAbuf_to_MPIbuf(MPI_buf0, CUDA_buf0, …)
MPI_Isend(MPI_buf0, …, &req[0])
MPI_Irecv(MPI_buf1, …, &req[1])
MPI_Waitall(req)
copy_MPIbuf_to_CUDAbuf(MPI_buf1, CUDAbuf1, …)
cudaMemcpyAsync(f, CUDAbuf1, …, stream[1])

cudaThreadSynchronize()
MPI_Barrier(…)
```

図 4　オーバーラップ計算の擬似コード
Fig. 4　Flow in Overlapping communication with computation by C-like pseudo code.

5. 性能評価

3 つの解像度での計算（512 × 512 × 512 格子，960 × 960 × 960 格子，1920 × 1920 × 1920 格子）の各々について GPU 間通信と GPU 計算をオーバーラップさせる計算と，非オーバーラップ計算に対して GPU 数を変えて得られた実行性能を**図 5** に示す．ただし，各解像度での実行に関しては GPU 数が少ない領域では 1 GPU カードあたりに搭載されているメモリが 4 GByte であるため実行できない．一方，z 軸方向のみの分割のため，より多用な GPU 数で分割できるように 960 × 960 × 960 や 1920 × 1920 × 1920 という格子点数を選んだ．

いずれの解像度においてもオーバーラップ計算は非オーバーラップ計算に対して性能が大幅に改善されていることが分かる．オーバーラップ計算では 512 × 512 × 512 格子について 1〜8 GPU までの範囲，960 × 960 × 960 格子について 4〜16 GPU まで範囲，1920 × 1920 × 1920 格子において 30〜32 GPU までの範囲で理想的な Strong Scaling を示している．

また，問題サイズを大きくすることによる Weak Scaling の性能については，512×512×512 格子の 8 GPU，960 × 960 × 960 格子の 24 GPU，1920 × 1920 × 1920 格子の 32 GPU で理想的な性能を示している．

オーバーラップ計算の特徴として，Strong Scaling については GPU 数が少ないときは

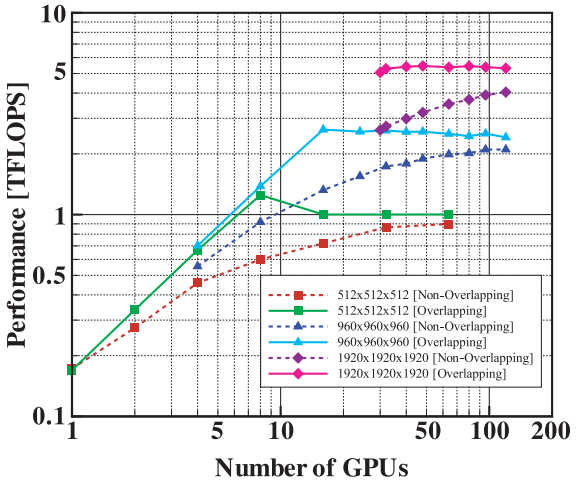
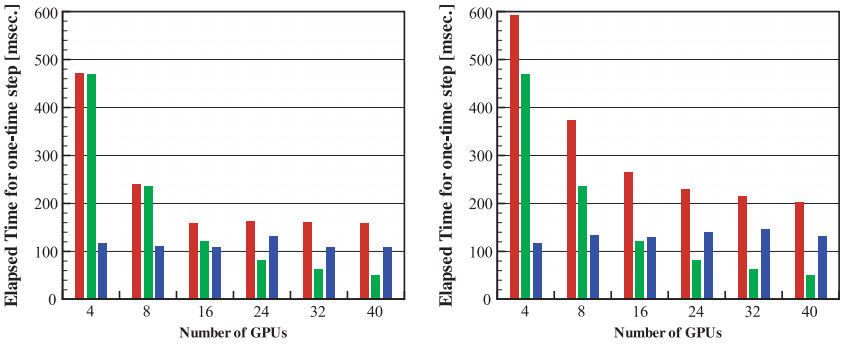


図 5　複数 GPU を用いたオーバーラップ計算・非オーバーラップ計算のスケーラビリティー
Fig. 5　Multi-GPU scalability for overlapping computing and non-overlapping computing.

理想的な直線に近づくが，ある GPU 数より多くなると急激に性能が頭打ちになり，非オーバーラップ計算と変わらない性能に近づく．

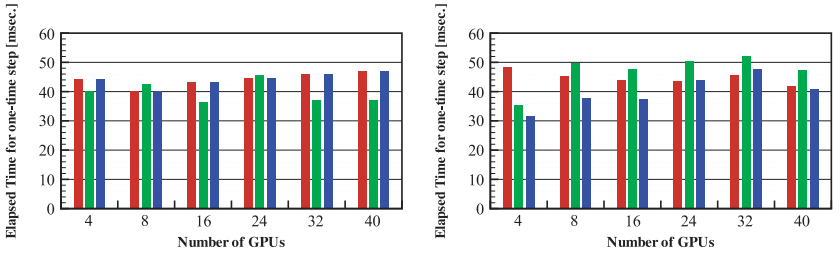
特筆すべきことは 1920 × 1920 × 1920 格子の計算に対して，40 GPU を用いてオーバーラップ計算を行った場合の実行性能が 5 TFLOPS に達したことである．実用的な格子系 HPC アプリケーションとしては，これまでに発表されている中では最高性能を示していると思われる．

オーバーラップ計算と非オーバーラップ計算の違いを明らかにするために計算・通信の各パートにおける所要時間を用いて各通信パートに対して詳細に測定する．960 × 960 × 960 格子における計算を使用する GPU 数を 8，16，24，32，40 GPU のように増加させて調べ，**図 6** に示す．時間は 1 タイムステップの演算に要する時間をミリ秒単位で表していて，各 GPU 数において棒グラフは左側から演算に要した時間（Total），MPI 通信の時間（MPI），計算時間（Computation），通信時間の合計時間（Communication Total）の項目となっている．測定の都合上，オーバーラップ計算における計算時間（Computation）は非オーバーラップ計算と同様と仮定し，グラフに示している．また，通信時間の内訳を調べ，**図 7** に示す．図 6 同様，1 タイムステップの演算に要する時間をミリ秒単位で表しており，棒グラフは左側から CPU–GPU 間のデータ転送（cudaMemcpy），MPI による通信時間（MPI），



(a) Overlapping computation.　(b) Non-Overlapping computation

図 6　オーバーラップ/非オーバーラップ計算の通信時間と計算時間
Fig. 6　Changes of the computational time and communication time in the overlapping/non-overlapping computing (Left: Total, Center: Computation, Right: Communication respectively).



(a) Overlapping computation.　(b) Non-overlapping computation.

図 7　オーバーラップ/非オーバーラップ計算の通信時間の内訳
Fig. 7　Breakdown of communication time (Left: cudaMemcpy or cudaMemcpyAsync, Center: MPI, Right: Mem.Cpy respectively).

CUDA の Pinned されたメモリと MPI 用の送受信用配列とのデータの移し替えに要する時間（Mem.Cpy）である．

GPU 数を増やすにつれて 1 GPU の計算する格子数は減るので，GPU の演算時間は短縮されていく．一方，1 次元的な分割を行っているので，分割された領域どうしが交換する袖領域の大きさは同じであり，cudaMemcpy() にかかる時間や MPI 通信にかかる時間には変

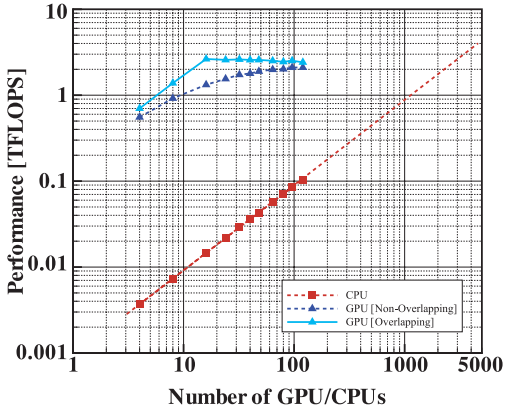


図 8　CPU と GPU の実行性能（格子点数 960 × 960 × 960）
Fig. 8　Performance comparison between CPU and GPU for the mesh of 960 × 960 × 960.

化がほとんどない．オーバーラップ計算による通信時間の隠ぺいについて，960 × 960 × 960 格子の計算を例に説明する．図 6 から 16 GPU 以下では演算時間が通信時間よりも長く，演算時間が計算の実行時間を決めていることが分かる．24 GPU 以上になると通信時間の方が演算時間より長くなり，通信時間が実行時間を決めるようになる．通信時間は GPU 数によらないので，実行性能は横ばいになる．また，MPI の送受信用バッファと cudaMemcpy() の送受信バッファが共有できないために，それぞれに pinned メモリのバッファを確保している．このメモリ間コピーに要する時間が MPI や cudaMemcpy() と同程度であることが図 7 から分かる．今後，MPI と CUDA のドライバ側の改善によりコピーが不要となれば通信時間は 3 分の 2 に短縮されることになり，各解像度の計算に対して理想的な Strong Scaling の範囲をさらに広げることができる．

非オーバーラップ計算では，演算時間と通信時間の合計が計算の実行時間になる．少ない GPU 数でも通信時間がオーバーヘッドとして現れ，Strong Scaling を悪化させている．GPU 数を多くすると演算時間が短縮されていくだけなので，実行時間は次第に通信時間に近づいていく．

TSUBAME の CPU（Opteron 1 コア）との実行性能の比較を行うために，まったく同じ 960 × 960 × 960 格子で両者の計算を行った．この格子サイズでの GPU 計算の最大性能は，オーバーラップ計算を行った場合の 16 GPU で 2 TFLOPS である．CPU の実行性能を GPU と比較しながら**図 8** に示す．実行した 128 CPU コアまでの範囲でほぼ理想的な Strong Scaling を示しているが，GPU 計算に対する実行性能はかなり低い．GPU の最高性能である 2 TFLOPS に追いつくには理想的な Strong Scaling を仮定しても 2200 CPU コア以上が必要となる．

6. 結　　言

純金属のデンドライト凝固成長に対して，フェーズフィールド法に基づいた Allen-Cahn 方程式と熱伝導方程式を連立させたモデルを NVIDIA の Tesla GPU を搭載した TSUBAME を用いて計算した．CUDA によりプログラミングを行い，単一 GPU で 171 GFLOPS（単精度計算）を達成した．さらに複数 GPU に対して領域分割に基づく並列化を行い，40 GPU で 5 TFLOPS の実行性能を達成した．本研究では，以下の新しい知見を得ることができた．

- 領域間通信と GPU 計算のオーバーラップを行うことにより複数 GPU 計算によるスケーラビリティーを大幅に改善し，性能が Strong Scaling する範囲を明らかにした．
- オーバーラップ計算では，通信時間が演算時間より長くなり通信時間を隠ぺいできなくなると急激に実行性能が頭打ちになる．一方，非オーバーラップ計算では少ない GPU 数でも通信時間のオーバーヘッドが実行性能の低下をもたらす．GPU 数を増やすにつれてオーバーラップ計算との性能差は少なくなる．
- 複数 GPU を用いた計算における計算時間と演算時間（CPU–GPU 通信，MPI 通信など）の内訳を明らかにした．
- 複数 GPU 計算に対する実行性能を CPU 計算と直接比較し，GPU 計算の有効性を確認した．

複数 GPU を用いた計算では通信と演算のオーバーラップを行うことが重要であり，TSUBAME の環境を改善することができるならば，

(1)　GPU を PCI Express × 16（Gen 2.0）に接続する，
(2)　ノード間のインターコネクトを Infiniband QDR など，より高速なものにする，
(3)　Infiniband のドライバや CUDA のドライバの改良により pinned メモリの共有が可能になり，コピーを省くことができる，

を行うことにより通信時間を大幅に短縮し，理想的な Strong Scaling の範囲を拡大することができる．さらに，プログラミング上の観点からは，

- 3 次元的な（サイの目状の）領域分割を行うことにより，分割数が増えると領域間通信のデータサイズが減少するため通信時間を短縮することができる．

これらは，流体計算などの他の格子系ステンシル計算についても適用でき，本研究は格子系の陽的手法に対して複数 GPU を用いる計算手法に指針を与えたといえる．

謝辞　本研究の一部は日本学術振興会（JSPS）グローバル COE プログラム「計算世界観の深化と展開」（Comp View），科学研究費補助金・基盤研究（B）課題番号 19360043「多モーメント手法による多目的 CFD コアの開発」および科学技術振興機構（JST）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コンピューティング」（ULP-HPC）から支援を受けている．本研究を遂行するにあたり，スーパーコンピュータ TSUBAME Grid Cluster での大規模並列計算キューの利用に対して東京工業大学学術国際情報センターよりご協力をいただいた．記して謝意を表す．

参 考 文 献

1) Takaki, T., Fukuoka, T. and Tomita, Y.: Phase-field simulation during directional solidification of a binary alloy using adaptive finite element method, *J. Crystal Growth*, Vol.283, No.1-2, pp.263–278 (2005).
2) 遠藤敏夫，松岡　聡，橋爪信明，長坂真路：ヘテロ型スーパコンピュータ TSUBAME の Linpack による性能評価，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.8, pp.62–70 (2007).
3) Barker, K. J., Davis, K., Hoisie, A., Kerbyson, D. J., Lang, M., Pakin, S. and Sancho, J.C.: Entering the petaflop era: the architecture and performance of Roadrunner, *SC '08: Proc. 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp.1–11, IEEE Press (2008).
4) NVIDIA: *NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 2.0*, NVIDIA Corporation (2008).
5) Phillips, J.C., Stone, J.E. and Schulten, K.: Adapting a message-driven parallel application to GPU-accelerated clusters, *SC '08: Proc. 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp.1–9, IEEE Press (2008).
6) Nyland, L., Hariss, M. and Prins, J.: *Fast N-Body Simulation with CUDA*, GPU Gems, Vol.3, chapter31, pp.677–695, Addison-Wesley (2007).
7) Stone, J.E., Phillips, J., Freddolino, P.L., Hardy, D.J., Trabuco, L.G. and Schulten, K.: Accelerating molecular modeling applications with graphics processors, *J. Computational Chemistry*, Vol.28, No.16, pp.2618–2640 (2007).
8) 遠藤敏夫，額田　彰，松岡　聡，丸山直也：異種アクセラレータを持つヘテロ型スーパーコンピュータ上の Linpack の性能向上手法，Vol.2009-HPC-121, No.24, p.8, 情報処理学会 (2009).
9) 小川　慧，青木尊之：CUDA による定常反復 Poisson ベンチマークの高速化，情報処理学会研究報告 2008-HPC-115，Vol.2008, No.43, pp.19–23, 情報処理学会 (2008).
10) 成瀬　彰，住元真司，久門耕一：GPGPU 上での流体アプリケーションの高速化手法，2009 年ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，Vol.2009, No.2, pp.115–122, 情報処理学会 (2009).
11) 小川　慧，青木尊之：GPU によるマルチグリッド法を用いた 2 次元非圧縮性流体解析の高速計算，日本計算工学会論文集，Vol.2009, No.20090021 (2009).
12) Kobayashi, R.: Modeling and numerical simulations of dendritic crystal growth, *Physica D*, Vol.63, No.3-4, pp.410–423 (1993).
13) PAPI. http://icl.cs.utk.edu/papi/

(平成 21 年 10 月 2 日受付)
(平成 22 年 2 月 16 日採録)

小川　慧（学生会員）

昭和 57 年生．平成 19 年東京工業大学大学院理工学研究科原子核工学専攻修士課程修了．同年同博士課程進学，在学中．現在，博士課程修了．博士（工学）．マルチモーメントを用いた高精度数値流体力学，GPU を用いた計算力学に関する研究に従事．平成 20 年日本機械学会フェロー賞（若手優秀講演）受賞，平成 22 年 IEEE Computer Society Japan Chapter 優秀若手研究賞受賞．日本機械学会，日本計算工学会，電子情報通信学会，電気学会，IEEE Computer Society 各会員．

青木　尊之（正会員）

昭和 60 年東京工業大学大学院総合理工学研究科エネルギー科学専攻修了．昭和 61 年富士通研究所を経て東京工業大学大学院総合理工学研究科助手．平成 9 年東京工業大学原子炉工学研究所助教授．平成 13 年〜東京工業大学学術国際情報センター教授（平成 21 年から副センター長）．高精度・数値計算手法の開発および多相流，爆風，乱流，気象，津波，土石流等，スパコンによるさまざまな大規模シミュレーションの研究に従事．博士（理学）．Scientific Visualization にも興味を持ち，Graphics Award 等を多数受賞．日本機械学会・計算力学部門業績賞受賞．JACM 日本計算力学連合副会長．日本機械学会，日本計算工学会，日本流体力学会，可視化情報学会各会員．

山中　晃徳

昭和 56 年生．平成 20 年神戸大学大学院自然科学研究科機械システム科学専攻博士課程修了．平成 20 年より東京工業大学大学院理工学研究科機械制御システム専攻助教．Phase-Field 法を援用した鉄鋼材料の組織形成シミュレーション法，材質予測手法に関する研究に従事．博士（工学）．平成 18 年日本材料学会第 55 期優秀講演発表賞受賞．日本機械学会，日本材料学会，日本鉄鋼協会，日本塑性加工学会各会員．