Vol.2014-HPC-145 No.40
2014/7/30

GPU仮想化による耐故障性を考慮した分子動力学シミュレーション

老川 稔[1,a)]　野村 昴太郎[1]　泰岡 顕治[1]　成見 哲[2,b)]

概要：GPU はその高い並列計算性能から多くの高性能計算機で利用されている．近年では高性能計算機を構成する装置の複雑化と大規模化が進み，その平均故障時間は短くなる傾向にあることから，計算機システムの信頼性を確保するための耐故障性機能が求められている．本稿では，GPU 仮想化ソフトウェアの機能を拡張することにより，GPU を含んだ計算機環境の動作信頼性を向上させるシステムを開発した．GPU を使用した分子動力学シミュレーションを対象とした動作テストを実施し，想定した障害から自動的に復旧されることを確認した．想定した障害は次の２つ，(1)GPU カーネル関数実行中の計算誤り (2) ホスト–GPU 間通信の切断，である．具体的な手法は，GPU の多重化による計算の冗長化，代替 GPU への動的なプロセスのマイグレーション，GPU メモリ上のデータと CUDA 関数の実行履歴のチェックポインティングとロールバック実行である．

キーワード：GPU, 仮想化，耐故障性，分子動力学シミュレーション

Fault-Tolerant Molecular Dynamics Simulation using Virtualized GPUs

Abstract: GPU is widely used with many high-performance computing systems as a calculation accelerator. In recent years, the increasing scale and complexity of high-performance computing systems consists of many components causes the shorter MTBF(Mean Time Between Failure) period, so that the fault tolerant methodology are required to achieve the high reliability of the computing systems. We modified the GPU virtualization software to enhance the reliability of GPU computing systems, and tested the fault tolerant functions we implemented by executing molecular dynamics simulation using GPU devices. We supposed two types system fault, (1)calculation error during GPU kernel function, (2)unexpected disconnection between host and GPUs. We achieved the auto recovery functions from these faults, utilizing redundant GPUs, checkpointing and roll-back recovery techniques.

Keywords: GPU, virtualization, fault tolerant, molecular dynamics simulation

1. はじめに

ディスプレイ表示のための画像データ処理を効率的に処理するために開発された Graphics Processing Unit (GPU) は，その並列処理性能の高さが注目され画像処理以外に科学技術計算の分野でも広く利用されるようになった [1]．特に高速演算処理向けに特化された GPU は General-Purpose computing on GPU(GPGPU) と呼ばれる．デバイスの動作信頼性の観点から見た両者の違いは，GPGPU にのみ ECC(Error Check and Correction) とよばれるビットエラーを訂正する機能付きのメモリが使用されていること，より厳しい信頼性試験が実施されていることなどがある [2]．

一般に電子デバイスには，故障することなく連続稼働することを期待できる平均期間を表す “平均故障間隔”(Mean Time Between Failure, MTBF) と呼ばれる信頼性を表す指標が設定されている．メモリの ECC 機能は MTBF を改善する方向へ働く．メモリに限らず外部ストレージ，通信デバイス装置等にも同様なデータエラーの訂正機能が組み込まれている．大規模化の進む高性能計算機はこれらのデバイスから構成され, その 1 つに GPGPU をアクセラレータ

[1] 慶應義塾大学
Keio University
[2] 電気通信大学
The University of Electro-Communications
[a)] m_oikawa@z2.keio.jp
[b)] narumi@cs.uec.ac.jp

として採用しているスーパーコンピュータ TSUBAME[3] がある．TSUBAME2.0 の実運用環境下で発生した様々な要因によるシステム障害情報は公開されており，1 計算ノード当たり 3 台の GPGPU デバイスが接続された 1 ノードにおける MTBF が約 2,500 日であり，そのノードのおよそ 1430 台から構成される TSUBAME システム全体の MTBF が 1.7 日程度であったことが報告されている [4]．将来的なエクサスケール規模の高性能計算機システムを，現在のクラスタ型スパコンの単純な拡張であるとするならば MTBF がさらに短期間になることが予測され，大規模な計算を行った場合に実用的な運用に耐えられなくなる懸念がある．

本研究では，ソフトウェア的なサポートによるアプリケーションレベルでの動作信頼性の向上を目的としている．関連する研究として，本稿の直接の先行研究に当たる，GPGPU にくらべて安価なコンシューマ向けの GPU を使用して信頼性を向上させる提案 [5] がある．同様なアプローチでソフトウェア的に GPU の動作信頼性を向上させた研究として，GPU 上の DRAM ビットフリップエラーを検出し自動的に GPU カーネルを再実行するソフトウェアフレームワークについて述べた研究 [6]，画像処理向けの GPU にソフトウェア的に ECC 機能を付加することで信頼性を向上させそのオーバヘッドを調査した [7] がある．GPU メモリのビットエラーに焦点を当てた提案であり，より実用的な耐故障性機能に対しては充分といえない．

本稿では，耐故障性機能の対象アプリケーションを GPU を使用する分子動力学シミュレーションに特化した．シミュレーションの実行中に仮想化した GPU との通信が途切れてしまった場合に，同一 GPU クラスタ内に存在する代替 GPU サーバに自動的に接続し，シミュレーションを継続する機能を実装し，評価を行った結果について述べる．第 2 章で GPU 仮想化ソフトウェアの内容と拡張した耐故障性機能について述べた．第 3 章で拡張した機能の動作テストとして実施した分子動力学シミュレーションについて説明した．第 4 章でまとめと課題について述べた．

2. GPU の仮想化と耐故障性機能の拡張

本稿では，GPU を仮想化するフレームワークとして既存のソフトウェアである DS-CUDA を利用した [8][9]．そのソースコードは公開されており，ユーザが自由に機能を追加し試すことが可能である [9]．DS-CUDA は，ネットワークに接続されている他の計算ノードにインストールされた NVIDIA 社製 GPU デバイスを，あたかも自分のローカルノードにインストールされているかのように GPU を仮想化するフレームワークである．仮想化はユーザアプリケーションソースコードのレベルで行われる．その特徴として，NVIDIA 社が提供する CUDA C/C++言語で記述されたソースコードで制御される GPU デバイスを仮想化するために，独自の言語拡張またはソースコードの変更を必要としない特徴をもつ．

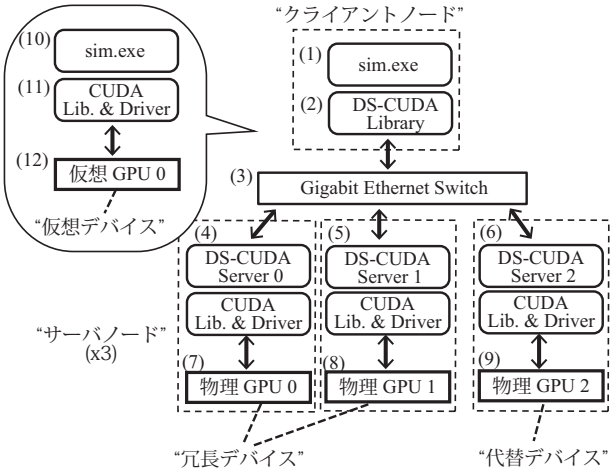

図 1 耐故障性機能を利用するための仮想化 GPU の構成
Fig. 1 Virtualized GPU configuration used for testing fault tolerant mechanism

複数の GPU デバイスを多重化して 1 台の仮想化 GPU として扱えることで，GPU のカーネル関数の実行を多重化し，GPU で発生した計算誤りを検出して自動的に再計算を行う機能が先行研究により実装された [5]．ただし任意のアプリケーションプログラムでの動作はまだ確認されておらず，計算誤りの検出に必要とされるデータは，アプリケーションソース上で毎ステップホスト–GPU 間で転送が行われる必要がある，との制限があった．

耐故障性機能を拡張するにあたり，これらの特徴を踏襲してチェックポインティングやロールバックリスタート機能はユーザアプリケーションソースコードから隠蔽されて自動的に実行されることを目指した．

2.1 DS-CUDA の使用した耐故障性機能をもつ仮想 GPU の構成

GPU 仮想化ソフトウェア DS-CUDA を使用して耐故障性機能をもつ仮想 GPU デバイスの構成する方法を**図 1** で説明する．この図は物理的に存在する 3 台の GPU(7)〜(9) を DS-CUDA が仮想化し，1 台の仮想 GPU(12) としてユーザアプリケーション (10) に見せている様子を表している．sim.exe(1) および (10) のソースコード上では，GPU が 1 台だけクライアントノードに存在しているかのように記述される．図中の四角枠はハードウェアを表し，丸枠はソフトウェア構成を表す．(1)〜(9) で示した構成部は物理的な GPU の接続構成を，吹き出し内の (10)〜(12) は，DS-CUDA によって仮想化された GPU 構成を表す．

sim.exe(1)(10) は，DS-CUDA 環境を使用してコンパイルされたアプリケーションの実行ファイルであり，第 3 章で後述する分子動力学シミュレーションの実行ファイルに相当する．DS-CUDA Library(2) は，DS-CUDA 環境か

ら提供される主にネットワーク通信機能を担当するライブラリである．本稿で拡張した耐故障性機能は主にこの Library に追加実装した．(4)〜(6) は，DS-CUDA 環境が提供する Server プログラムであり各サーバノード上で実行される．Gigabit Ethernet Switch(3) を介して DS-CUDA Library との通信を行い，クライアントノードから要求に従って (7)〜(9) の物理 GPU{0,1,2} の制御を行う．以上のように複数の GPU デバイスを仮想的な 1 台の GPU として扱うことで，ユーザアプリケーション (1) と物理 GPU の間に DS-CUDA Library(2) および DS-CUDA Server が介在してユーザアプリケーションから隠蔽された状態で次の耐故障性機能を実行する．

2.2 仮想化 GPU におけるチェックポイントの作成

チェックポイントの作成は，GPU 仮想化ソフトウェアが全ての仮想 GPU デバイスに割り当てられた全てのメモリ領域をクライアントノード側のアドレス空間上に自動的にミラーリングすることで行なわれる．そのために，シミュレーション中の計算に使用されるデータが GPU デバイスメモリ上以外 (ホストメモリやストレージ等) に存在する場合はデータのミラーリング対象に含まれない．その場合には耐故障性機能として正しく機能しなくなるため，計算に使用するデータをすべて GPU メモリ上に配置する必要がある．本稿で古典 MD シミュレーションを選択した理由のひとつには，GPU で処理するデータの種類が原子の座標，速度，原子間相互作用力のように限定されており，GPU メモリ上に配置しやすいという理由がある．

仮想 GPU に対して発行される CUDA API 関数（カーネル関数実行命令やメモリデータ転送命令）のすべてが DS-CUDA を介して行われることを利用して，図 1 中の DS-CUDA Library(2) が GPU を対象とした制御関数の実行履歴を記憶し，必要に応じて過去の実行履歴に遡ってやり直すことを可能にしている．これらの機能を利用して，冗長計算機能と GPU プロセスマイグレーション機能を実現している．

2.3 GPU の多重化による自動冗長計算

図 1 中に記載した “冗長デバイス” について説明する．冗長デバイスは，ユーザアプリケーションソースコードからは仮想的に 1 台の GPU デバイス (12) として扱われるが，DS-CUDA の仮想化機能により物理的には複数台の GPU が割り当てられる．冗長デバイスは，クライアントノードの環境変数としてホスト名またが IP アドレスにより明示的に登録する必要がある．用途は GPU で実行されるカーネル関数の実行を多重化し，信頼性を向上させることである．

sim.exe(1) から 1 台の仮想化 GPU(12) に対してカーネル関数の実行が要求されると，DS-CUDA Library がその命令を多重化 (本図では 2 重化) し，それぞれを冗長デバイスである物理 GPU0(7) および GPU1(8) で同じ処理内容の計算を実行させる．2 つ得られた実行結果を DS-CUDA Library(2) が自動的に照合し一致した場合はそのままアプリケーションの実行を継続し，不一致が検出された場合には計算誤りが発生したと判定する．計算誤り判定がなされた後は，GPU(7)(8) のメモリ状態を誤りが発生しなかった状態までロールバックさせたのち，不一致が検出された計算を再実行する．この対処法は，GPU 上での計算誤りは発生確率が低く再現性のない過渡的な障害であることを前提としている．定常的な障害であった場合には，次に説明する代替デバイスとの交換が行われる．

冗長デバイスは最大 4 台まで冗長化することができるが，DS-CUDA ソースコード上で定義されている定数の変更により増減することができる．

2.4 代替デバイスを使用した GPU プロセスマイグレーション

図 1 中の “代替デバイス” とは，障害が発生しない限りはユーザアプリケーションから使用されることはなく，アプリケーションの実行中に再計算では回避されない（再計算を一定回数繰り替えしても一致しない）障害，または通信障害などで物理 GPU と通信が切断されてしまった場合に交換される GPU デバイスのスペアを指す．GPU デバイスの交換が行われる際には，故障した GPU 上のメモリ状態が代替デバイスのメモリにリストアされる．

冗長デバイスとは異なり，代替デバイスは明示的にクライアントノードに予め登録しておく必要はない．ユーザアプリケーションの起動時に同一ネットワーク上で DS-CUDA Server の親プロセスであるデーモンプログラムが起動している計算ノードを DS-CUDA library が探索し，見つかった未登録のデバイスを自動的に代替デバイスとして認識する．シミュレーション実行中に，代替デバイスと交換する必要が生じたが登録されている代替デバイスが存在しない場合には，そこでアプリケーションを中断する．

2.5 GPU 仮想化ソフトウェアを利用した擬似的な障害の挿入

大規模な GPU クラスタの実運用上での障害発生率が課題とはいえ，障害発生時にしか機能しない耐故障性機能の動作試験やデバッグ時には意図したタイミングと発生箇所で期待した障害を起こることが望まれる．吉川ら [5] は，仮想 GPU からクライアントへのデータ転送時に確率的に 1Byte が誤るように設定された DS-CUDA サーバを準備し，自動計算機能が正しく機能することを確認した．GPU 上で擬似的にではなく，より自然な意味での障害を発生させたいならば，GPU 内のシステムクロックやメモリクロックの周波数を動作保証範囲外まで上昇させることである程

表 1　GPU クラスタ中の計算ノード主要諸元
Table 1　The specification of GPU cluster used for evaluation

ハードウェア環境	
CPU:	Intel Core i7, 2.67GHz
GPU:	NVIDIA 社 GeForce GTX 580
	16 Streaming Multiprocessors
	512 cuda cores
	1.5GB メモリ
ソフトウェア環境	
オペレーティングシステム:	Linux 64bit 版
	kernel version 2.6.35.6
GPU ドライバ:	NVIDIA 社 version 331.79
C/C++コンパイラ:	gcc version 4.5.1
GPU ソフトウェア開発環境:	NVIDIA 社 CUDA Toolkit version 4.2
GPU 仮想化ソフトウェア:	DS-CUDA version 1.4.2[9]

度高い確率で障害発生を強制させることも有意義ではあるが，障害発生のタイミングと種類等までをコントロールすることができない．

本稿では，GPU カーネル関数内に計算誤り条件を直接記述し，図 1 の特定の DS-CUDA Server だけにその記述を実行させることで，後述する**図 4** に示したように，特定のタイミングでの計算誤り擬似的に発生させている．

2.6　評価に使用したハードウェア環境

NVIDIA 社製のコンシューマ向け GPU である GeForce GTX 580 を使用した．1 ノード当たりに GPU1 台を接続した計算ノード全 16 台を Gibabit イーサネット LAN で接続された GPU クラスタ環境で動作テストを行った．**表 1** に，各計算ノードの主な仕様を示す．

3.　実施した分子動力学シミュレーションの内容

レプリカ交換分子動力学シミュレーション (Replica-Exchange Molecular Dynamics Simulation, REMD) [10][11] の実行中に GPU 上で擬似的に計算エラーまたは通信の切断を発生させ，耐故障性機能の動作テストを行った．本稿では仮想 GPU1 台を対象としたシミュレーションのみを行ったが，将来的に複数台の仮想 GPU で動作テストを行うことを考慮し計算の並列化を行いやすい対象として REMD を選択した．

3.1　REMD シミュレーションの詳細

実施した REMD シミュレーションの詳細を，**図 2** および**図 3** を使って説明する．図 2 はシミュレーション系を表す．レプリカ交換法では，複数の分子動力学系 (a)〜(d) を同時に扱いそれぞれの系をレプリカと呼ぶ．図では実際に使用した 4 つのレプリカ (a)〜(d) を示した．各レプリカにはそれぞれ同数 $N_{\mathrm{atom}} = 32$ のアルゴン原子モデルが含まれる．アルゴン原子モデル間には，Lennard-Jones ポテンシャルモデル式の勾配から計算される相互作用力が働く [10]．各レプリカは互いに独立しており，異なるレプリカ内に存在する粒子間には相互作用は発生しない．シミュレーション開始時における初期条件として，レプリカ (a)〜(d) には互いに異なる温度条件を設定する．開始時刻におけるレプリカの初期設定温度を，(a)>(b)>(c)>(d) とした．この初期状態は，図 3 中の時刻 t_0 における (a)〜(d) に相当する．

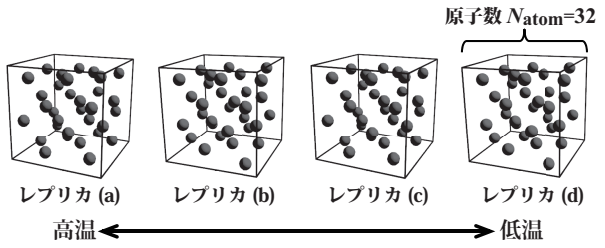

図 2　実施したレプリカ交換分子動力学シミュレーション系
Fig. 2　The System of Replica-Exchange Molecular Dynamics simulation

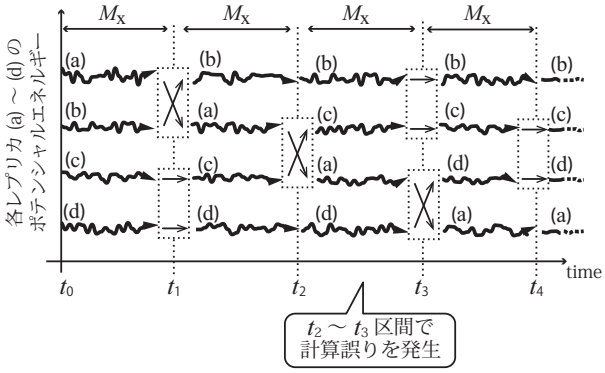

図 3　実施したレプリカ交換分子動力学シミュレーションの内容
Fig. 3　The process of Replica-Exchange Molecular Dynamics simulation

図 3 は実施した REMD シミュレーションの計算処理内容を表している．横軸がシミュレーション内の経過時間であり，縦軸は各レプリカ (a)〜(d) のポテンシャルエネルギーを表す．レプリカに設定された温度が高いほど，時間平均的に高いポテンシャルエネルギーをもつ．図中の M_{x} で示した一定の期間は，各レプリカ内の原子間に作用する力を使った数値的に解かれた運動方程式に従い各原子座標の計算が行われる．時間 M_{x} が経過する毎に，隣接する温度が設定されたレプリカ間で確率的に温度条件が交換される (図中の t_1〜t_4 で示した 4 ポイント)[10]．この温度条件の交換を，図中の破線四角に囲われた 2 つの矢印で表す．交差した矢印は交換が行われたことを示し，平行な矢印は交換が行われなかったことを示す．各レプリカの温度が以前の温度より低く設定された場合には，ポテンシャルエネルギーが低い方へ遷移する．逆に高く設定された場合には，エネルギーが高い方遷移する．

このシミュレーション実行中の t_2〜t_3 の計算を GPU で実行している間に，図 1 に示した冗長デバイスの物理 GPU1(8) において擬似的な計算誤りを発生させた．時刻 t_3 に仮想 GPU0(12) からポテンシャルエネルギーを回収したときに，冗長デバイス GPU0(7) と GPU1(8) から回収された 2 つの冗長データが，DS-CUDA Library(2) で比較され計算誤りとして検出される．時刻 t_3 の時点で GPU のメモリ上にあるデータが正常でないことから時刻 t_2 の時点での GPU メモリに存在していたデータのスナップショットが DS-CUDA Library(2) が管理するメモリ領域から物理 GPU0 および GPU1 にリストアされる．シミュレーション時間が t_3 から t_2 に戻されて実行が継続される．

3.2 REMD シミュレーションの実行結果

実際に REMD シミュレーションを実行した結果を図 4 に示す．横軸は時間を表し，時刻 t_0〜t_4 は図 3 で説明したそれぞれの時刻に対応する．縦軸は各レプリカ (a)〜(d) のもつポテンシャルエネルギーを表す．上段のグラフ (i) は GPU でのエネルギー値の算出過程で計算誤りが発生せず正常にシミュレーションが終了した結果として得られる波形である．下段のグラフ (ii) は，擬似的に発生させた計算誤りを時刻 t_3 の辺りで挿入しポテンシャルエネルギーの計算結果を誤らせた場合に得られた結果である．

灰色のハッチで示した領域内が計算誤りにより (i) と (ii) で異なった波形を示している．それ以前は同一の波形である．ハッチの領域以降では，点線で囲ったレプリカ (c) および (d) の波形も異なった波形を示している．これは，誤ったポテンシャルエネルギーの算出結果に基づいてレプリカ間の温度交換判定が行われてしまいその後のエネルギー計算にまでエラーが伝搬したことを示している．レプリカ (a) と (b) に関しては，エネルギー算出は誤ったが温度交換判定に影響しなかったために，ハッチ領域以降の時刻で差分は発生していない．

冗長計算機能を無効にした場合に，図 4 の下段 (ii) のシミュレーション波形が得られ，有効にした場合には上段 (i) の波形が得られることを確認した．計算誤りが検出され自動再計算が行われたという情報は，DS-CUDA のログメッセージとして出力される．

クライアントノード–GPU 間の通信が切断される擬似エラーを挿入した場合には図 1 の物理 GPU1(8) からの応答がなくなるため，“RPC:Timed out” の通信エラーメッセージが出力されてシミュレーションが異常終了した．耐故障性機能を有効にした場合は，DS-CUDA のログメッセージとして “Timed out” エラーが検出されたこと，冗長デバイスの 1 つである物理 GPU1(8) が無効なデバイスとして除外され代替デバイス (9) が有効なデバイスとして登録され接続されたこと，物理 GPU1(8) のメモリスナップショットが代替デバイス上にリストアされたこと，を表すメッセージが出力され，図 4 の (i) の波形が得られたことを確認することができた．

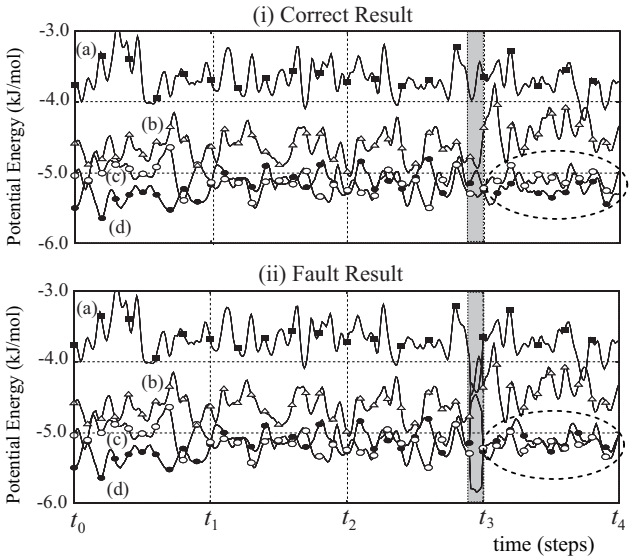

図 4 シミュレーションが正常に終了した場合 (i) および計算誤りが発生した場合 (ii) に得られる波形の比較

Fig. 4 Comparison of Correct and Fault Results of Simulations

4. まとめと課題

GPU 仮想化ソフトウェア DS-CUDA の機能を拡張し，障害発生時における GPU プロセスマイグレーションと自動ロールバック機能を含む代替 GPU デバイスへの動的な切り替え機能を実装した．それらの耐故障性機能が追加された仮想化 GPU を使い，レプリカ交換分子動力学シミュレーションを実施し，GPU からの応答がなくなる擬似的な通信障害を発生させた場合に，自動的に代替 GPU デバイスとの差替えが行われて正常にシミュレーションの実行が継続されることを確認した．

今後の課題は，本稿で対象としなかった他の要因による障害からの復旧機能を充実させることが挙げられる．また，擬似的に発生させた障害からの復旧ではなく，実際に発生した通信障害等に対しても有効に機能することを確認する．そのためには大規模な環境ほど障害の発生する可能性が高くなることが期待されるため，多数の GPU を使った大規模なシミュレーションでの動作検証も行っていく予定である．

謝辞 本研究は, 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．

参考文献

[1] TOP500 Supercomputer Site (online), 入手先 ⟨http://www.top500.org/⟩ (2014.06)
[2] NVIDIA 社 Site (online), 入手先

⟨http://www.nvidia.com/object/why-choose-tesla.html⟩(2014.06).
[3] 東京工業大学 学術国際情報センター, TSUBAME 2.5 HARDWARE SOFTWARE SPECIFICATIONS, 入手先 ⟨http://tsubame.gsic.titech.ac.jp/documents⟩(2014.03)
[4] 松岡聡, 佐藤賢斗, 遠藤敏夫, “エクサスケールスパコンに向けた耐故障性の評価–TSUBAME2.0 を例にして–”, 情報処理学会研究報告, Vol.2013-HPC-141, No.22, pp.1–8(2013).
[5] 吉川和幸, 川井敦, 泰岡顕治, 成見哲, “GPU 仮想化による自動冗長計算システム”, 情報処理学会研究報告, Vol.2012-HPC-134, No.6, pp.1–5(2012)
[6] 丸山直也, 額田彰, 松岡聡, “GPU 向け耐メモリエラーソフトウェアフレームワーク”, 情報処理学会研究報告, Vol.2009-HPC-123, No.8, pp.1–6(2009)
[7] 丸山直也, 額田彰, 松岡聡, “GPU 向けソフトウェア ECC の性能評価”, 情報処理学会研究報告, Vol.2009-HPC-119, No.5, pp.1–6(2009)
[8] Atsushi Kawai, Kazuyuki Yoshikawa, Kenji Yasuoka and Tetsu Narumi: “Distributed-Shared CUDA: Virtualization of Large-Scale GPU Systems for Programmability and Reliability”, Future Computing 2012, Nice, France (2012.07)
[9] 株式会社 K&F Computing Research: DS-CUDA ソフトウェアパッケージ　ユーザガイド (online), 入手先 ⟨https://github.com/Daweek/Original_SC/tree/master/dscudapkg1.4.2⟩ (2014.06.20)
[10] C.Muguruma, Y.Okamoto and M.Mikami: New approach to the first-order phase transition of Lennard-Jones fluids, *The Journal of Chemical Physics*, Vol.120, pp.7557–7563 (2004)
[11] C.Muguruma, Y.Okamoto and M.Mikami: Multicanonical Monte Carlo Calculation of the First-order Phase Transition of Lennard-Jones Fluids, *Croatica Chemica Acta*, Vol.80, No.2, pp.203–209 (2007)