情報処理学会論文誌　コンピューティングシステム　Vol.7　No.4　1–14 (Dec. 2014)

1,024 GPUを使用したレプリカ交換分子動力学シミュレーションの並列化

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

受付日 2014年4月4日, 採録日 2014年8月19日

概要：CUDA C/C++言語で記述された，シングルノード向けレプリカ交換分子動力学シミュレーションのソースコードにGPU仮想化ソフトウェアを適用し，マルチノード上のGPUに対応した計算並列化を行った．レプリカ交換法は，分子/原子の最安定状態を効率的に求めるために用いられるシミュレーション手法の1つである．計算の粒度が大きく，計算ノード間の通信が少ないという特徴から，分散計算機環境での並列化に向く．本研究では，GPU仮想化ソフトウェアの機能を利用することで，シングルノード向けのソースコードを改変することなくマルチノード用に再利用した．最大1,024台のGPUを使用して計算速度の並列化効率を計測し，1,024並列時において効率87%の良好な結果を得た．また，効率を低下させる要因について定量的に解析し，特に多並列時においてノード間通信のレイテンシが支配的になることが分かった．

キーワード：Graphics Processing Unit，CUDA，分子動力学シミュレーション，並列化，レプリカ交換法

Parallel Molecular Dynamics Simulation with Replica-exchange Method Using 1,024 GPUs

Minoru Oikawa[1,a)]　Kentaro Nomura[1,b)]　Kenji Yasuoka[1,c)]　Tetsu Narumi[2,d)]

Received: April 4, 2014, Accepted: August 19, 2014

Abstract: We parallelized a molecular dynamics simulation software using single-node GPUs to support multi-node GPUs by applying GPU virtualization tool to CUDA C/C++ program. To obtain the global minimum state of molecules efficiently, we used replica-exchange method, which is suitable for parallel computation since it has large granularity and small communication between nodes. The GPU virtualization software enabled us to use a single-node program to support multi-node calculation without any modification. Parallel efficiency with up to 1,024 GPUs are measured and 87% of efficiency was obtained even with 1,024 GPUs. The main reason of losing efficiency for higher parallelism was quantitatively analyzed and we found it mostly the latency between nodes.

Keywords: Graphics Processing Unit, CUDA, molecular dynamics simulation, parallelization, replica-exchange method

1. はじめに

画像処理専用プロセッサとして開発されたGraphics Processing Unit（GPU）は，その並列演算処理性能の高さが注目され，高性能計算機の分野に特化されたGeneral-Purpose computing on GPU（GPGPU）向けの製品が開発されるに至り，近年では数値演算アクセラレータとして広く利用されるようになった[1], [2]．その演算性能をユー

[1] 慶應義塾大学
Keio University, Yokohama, Kanagawa 223–8522, Japan
[2] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] m_oikawa@z2.keio.jp
[b)] kentan@a3.keio.jp
[c)] yasuoka@mech.keio.ac.jp
[d)] narumi@cs.uec.ac.jp

© 2014 Information Processing Society of Japan

ザが利用するため，Compute Unified Device Architecture（CUDA）[3]，OpenACC [4]，OpenCL [5] といったプログラム開発環境が提供されており，分子動力学（Molecular Dynamics, MD）シミュレーション分野においても活用されている [6], [7], [8], [9].

近年の高性能計算機は，マルチ（メニー）コアプロセッサを搭載した数千から数万の計算ノードが高速なネットワークで相互に接続されているものが多く，このような多数の計算ノードから構成される計算機クラスタを利用して計算性能を予測する，あるいは引き出すことは容易でない [10]．MD シミュレーションには，粒子間に存在する複数種の相互作用計算を多くのタイムステップごとに繰り返す必要があることから膨大な計算量が必要とされる．レプリカ交換分子動力学（Replica-Exchange MD, REMD）シミュレーションでは，解析対象となる MD 系を複数同時に扱う．それぞれの MD 系（これを今後「レプリカ」と呼ぶ）に，パラメータ（たとえば温度）の異なる設定値を割り当てて各レプリカの統計情報の推移をモニタしながら，あらかじめ設定した条件によってそのパラメータを交換する手法がとられる [11], [12]．各レプリカでの分子間相互作用の計算はそれぞれで独立しているために，シミュレーション全体では計算の粒度を大きく設定することができ，レプリカを扱う各計算ノード間の通信量は少ない．したがって近年主流となっている分散計算機環境での計算に向いている [14], [15]．また，2 つ以上のパラメータ（たとえば温度と圧力の両方）を変化させ多次元的に割り当てる研究も行われている [16], [17]．複数のパラメータを扱う場合には，同時に扱うレプリカ個数がそのパラメータの組合せ数に従って必要になるために急速に増加する．多くのレプリカを同時にシミュレーションすることは分子動力学の応用上意味を持つ．

REMD 同様各計算ノード間の通信が少ないアプリケーションとして Genetic Algorithm があり，Jaros は複数の GPU を用いて計算を加速させている [18]．この際クラスタに存在する 14 台の GPU を使用するために MPI を用いて並列化しているが，CPU での負荷を分散するためではなく多くの GPU を使用するために使用している．本研究では GPU 仮想化ソフトウェアを使うことでシリアルなコードから多数の GPU の使用を可能にしている．このような簡便な方法でどこまで計算速度を稼ぐことができるかを知ることも本研究の目的である．

本研究では，14,366 レプリカを扱ったレプリカ交換 MD シミュレーションを最大で 1,024 個までの GPU を使って並列化した．また実行速度の並列化効率を見積もるモデル式を提示し，実測結果とよく一致することを示す．

2 章ではレプリカ交換分子動力学シミュレーションの概要と実施したシミュレーションの条件を述べる．3 章ではそれを GPU 上で並列化した手法について述べる．4 章で並列化効率のモデル式を構築し，5 章で実測結果との比較を行い，6 章でまとめと今後の課題について述べる．

2. レプリカ交換分子動力学シミュレーション

アルゴン原子群から構成されるバルクを対象とした固体–液体間の相転移温度を見積もるためのシミュレーションを対象とした．1 つの MD 系の温度を上下させることで相転移温度を見積もると，特に低温領域で原子が局所的なエネルギー極小状態にトラップされてしまい，そこから抜け出して最安定状態に緩和するまでに多くのシミュレーションタイムステップが必要になり計算量が膨大になることがある．レプリカ交換法は，最安定状態を効率良く求めることができる手法である．

2.1 レプリカ交換分子動力学シミュレーションの概要

レプリカ交換法は，互いに異なった温度を持つ複数の MD 系を，互いの温度条件を交換しながら並行してシミュレーションすることで局所的な準安定状態にとどまり続けることを避け，効率的に MD 系の最安定状態を探索することができる．並行して扱う複数の MD 系のそれぞれを「レプリカ」と呼び，特定のレプリカを指す場合は添字 $\{a, b, c, ...\}$ を付加することで「レプリカ (a)」等と表記する．レプリカの総数を「レプリカ数」と呼び記号 N_{rep} で表す．

図 1 にレプリカ交換法 MD シミュレーションの手法を示す．説明の便宜上，レプリカ数 $N_{\mathrm{rep}} = 4$ として各レプリカを図中の太線矢印 (a)～(d) で表した．探索する温度範囲を 50 K 以上 200 K 以下として，この温度範囲で互いに異なる設定温度 {200 K, 150 K, 100 K, 50 K} を，時刻 t_0 における初期温度として各レプリカ (a)，(b)，(c) および (d) に割り振る．あらかじめ設定したタイムステップ M_{exch} が経過するごとに，各レプリカが持つポテンシャルエネルギーの状態から，設定した温度条件をレプリカ間で交換を行うかを式 (1) に示す確率 P_{exch} により判定する．

$$P_{\mathrm{exch}} = \min\{1, \exp(-(U_i - U_j)(\beta_j - \beta_i))\}, \qquad (1)$$

ここで，U_i および U_j は隣接する温度条件を持つレプリカ

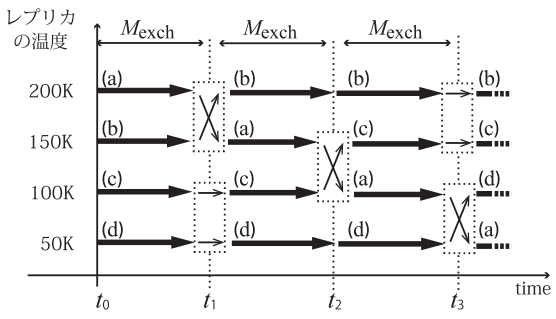

図 1　レプリカ交換法 MD シミュレーションの手法
Fig. 1　The method of replica-exchange MD simuluation.

© 2014 Information Processing Society of Japan

(i) および (j) が持つポテンシャルエネルギー，β_i，β_j はレプリカの温度条件とボルツマン定数の積の逆数を表す．$\min\{X, Y\}$ は X と Y のいずれか小さい方の値を表す．

時刻 t_1 では，隣接する設定温度 200 K と 150 K を持つレプリカ (a)(b) 間で式 (1) が計算され交換が成立し，別の隣接設定温度 100 K と 50 K を持つレプリカ (c)(d) 間でも判定が行われ交換が棄却されたことを表した．図中の破線枠が式 (1) による交換判定を表し，交差した矢印が交換の成立を，並行な矢印が交換の棄却を表す．隣接した温度を持つ 150 K と 100 K のレプリカは，次の交換判定時刻 t_2 で行われる．時刻 t_3 以降は，時刻 t_1 から t_2 で行われた温度交換判定が繰り返し行われる．

交換の判定が行われる M_{exch} ステップの間は，各温度を持つレプリカ内で独立して MD のシミュレーションが実施される．そもそも異なるレプリカ間で原子座標等を参照する必要がないうえに，レプリカ間で温度やポテンシャルエネルギーをやりとりするのは M_{exch} ステップごとであることから，レプリカ間での通信量は非常に少ない．

本研究で対象としたアルゴン原子群のシミュレーションでは，130 K から 150 K 付近で固体–液体相転移温度が得られることが分かっていることから [12], [13]，シミュレーションで探索する温度範囲を 150 K の上下にある程度のマージンを考慮して，50 K 以上かつ 250 K 以下に設定した．レプリカ数 $N_{\mathrm{rep}} = 14{,}336$ とし，温度交換判定ステップ数 $M_{\mathrm{exch}} = 100$ とした．

2.2　アルゴン原子群の分子動力学シミュレーション

前節で述べたように，各レプリカはアルゴン原子群から構成される．本節では，アルゴン原子群の MD シミュレーション条件について説明する．

バルク状態にある原子を対象とするため，立方体状の基本セルの空間的な繰返しからなる周期境界条件を適用した [12]．固体状態にあるアルゴン原子は面心立方格子（Face-Centerd Cubic, FCC）上に配置される．基本セル内の FCC 格子に原子が充填される条件は，基本格子に含まれる原子個数が 4 であることから，基本セル内に含む原子の個数 N_{atom} は $4a^3$（a は任意の整数）の場合とした．固体アルゴンの密度 1.65 (g/cm^3) およびアルゴン原子の質量 39.948 (g/mol) から，基本セルの物理サイズを決めた．本稿では，$a = \{4, 8\}$ のときのそれぞれ $N_{\mathrm{atom}} = \{256, 2048\}$ (個) になる 2 通りに場合についてシミュレーションを実施した．そのときの使用した基本セルの 1 辺の長さはそれぞれ $\{2.18, 4.35\}$ (nm) である．

MD シミュレーションを行うため必要な原子モデル間に作用する力の計算には式 (2) に示す Lennard-Jones ポテンシャル U を使用した [12]．

$$U_{ij} = \begin{cases} 4\,\varepsilon \left[\left(\dfrac{\sigma}{|\boldsymbol{r}_{ij}|} \right)^{12} - \left(\dfrac{\sigma}{|\boldsymbol{r}_{ij}|} \right)^{6} \right], & (|\boldsymbol{r}_{ij}| \le r_{\mathrm{c}}) \\ 0, & (\text{otherwise}) \end{cases} \tag{2}$$

ここで $|\boldsymbol{r}_{ij}|$ は粒子 i と粒子 j 間の距離，r_{c} はカットオフ距離と呼ばれ力場が作用する打ち切り距離を表す．ε と σ はそれぞれポテンシャル U_{ij} の高さと距離の特性を決めるパラメータであり，アルゴン原子にフィッティングされた値である $\varepsilon = 0.9661$ (kJ/mol)，$\sigma = 0.3405$ (nm) を適用した [12]．式 (2) は，$\lim_{|r_{ij}| \to \infty} U_{ij} = 0$ に急速に収束するため，カットオフ距離 $r_{\mathrm{c}} = 3.0\,\sigma$ 以上離れている粒子間に作用する力を 0 として計算を行った．2 粒子間に作用する力 $\boldsymbol{F}_{ij}$ は，式 (2) を $\boldsymbol{r}$ による微分 $-\nabla U$ から計算した．

3.　レプリカ交換分子動力学シミュレーションの GPU による並列化

REMD シミュレーションを GPU クラスタで並列化する方法を説明する．使用した GPU 仮想化ソフトウェアならびに GPU クラスタについて述べ，特に GPU コアの割り振りやメモリ配置，ノード間の通信量について詳しく述べる．

3.1　GPU 仮想化ソフトウェア DS-CUDA の利用

本研究では，REMD シミュレーションを GPU クラスタ上で簡便に実行するために GPU 仮想化ソフトウェア DS-CUDA [19], [20] を利用した．DS-CUDA の概要を**図 2** に示す．DS-CUDA は，CUDA API を仮想化することでネットワーク上に分散配置された GPU を透過的に扱うこ

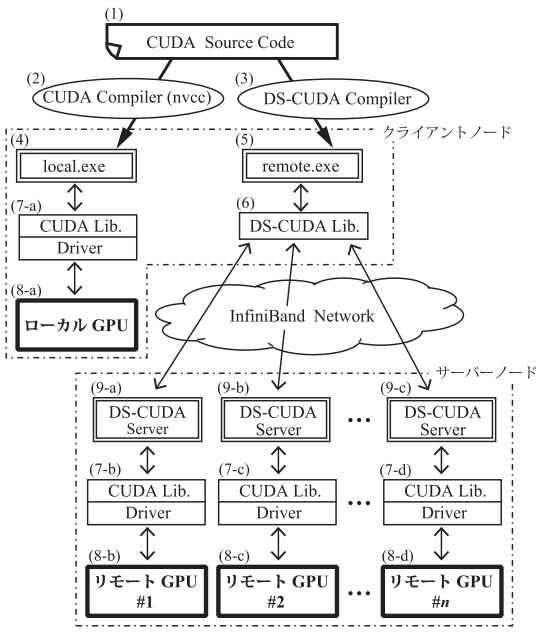

図 2　GPU 仮想化ソフトウェア DS-CUDA の概要
Fig. 2　Overview of DS-CUDA: GPU virtualization software.

© 2014 Information Processing Society of Japan

とができる．

図 2 の左側に位置する “(1)→(2)→(4)↔(7-a)↔(8-a)” で示した部分は，標準の CUDA 環境を使った開発・実行環境を表す．CUDA Source Code(1) を CUDA Compiler(2) を使ってコンパイルし，実行ファイル local.exe(4) を生成する．local.exe(4) は，CUDA API から構成される CUDA Library とデバイスドライバ (7-a) を介して，local.exe(4) が実行されているノードに接続されている GPU(8-a) を制御する．local.exe(4) を実行するノードを「クライアントノード」(図中の上半分にある鎖線内)，クライアントノードに接続されている GPU(8-a) を「ローカル GPU」と呼称する．

次に DS-CUDA を使った開発・実行環境について説明する．DS-CUDA 環境で生成される実行ファイル remote.exe(5) はクライアント–サーバ方式をとり，図中の InfiniBand Network を介した先のノードに接続された GPU(8-b,c,d) を制御する．これらの GPU を「リモート GPU」と呼ぶ．remote.exe(5) は，CUDA 環境で使用したのと同一の CUDA Source Code(1) から DS-CUDA コンパイラ (3) を使用して生成される．DS-CUDA の機能により，CUDA コンパイラ (2) を使用した場合に呼び出される CUDA API 関数は，DS-CUDA Library(6) に含まれる DS-CUDA の API 関数に置き換えられる．DS-CUDA Library(6) は，リモート GPU が接続されたノードで起動されている DS-CUDA Server プログラム (9-a,b,c) と通信を行う．リモート GPU が接続され，DS-CUDA Server(9) が起動されているノードを特に「サーバノード」(図中の下半分部の鎖線内) と呼ぶ．DS-CUDA Server(9-a,b,c) は，GPU と 1 対 1 で実行される．最大 1,024 台の GPU を使えるように DS-CUDA ソースコード上の通信用バッファサイズ定義部を修正した．また，4.2.2 項で述べる通信レイテンシ増加を抑制する機能を追加した．

3.2　使用した GPU クラスタ環境

図 3 に，使用した GPU クラスタの構成図を示す．本稿ではスーパコンピュータ TSUBAME2.5 [2] に含まれる計算ノードの一部を GPU クラスタとして使用した．各計算ノードの諸元を**表 1** に示す．InfiniBand インターコネクトにより接続された最大 343 台の計算ノード上で，前述の DS-CUDA を動作させて使用した．各計算ノードには 6 コアの CPU が 2 基搭載され，3 台の GPU デバイスが PCI Express インターコネクト（Gen2 x16，理論最大通信スループット 8 GByte/sec）に接続されている．使用したすべての計算ノードは，一様な計算性能と通信性能を有する．計算ノード中の 1 台（Node#0）を図 2 に記載したクライアントノードとして使用した．その他のノード（Node#1～#342）をサーバノードとして使用した．

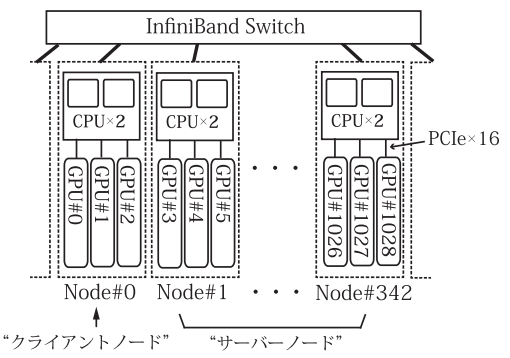

図 3　使用した GPU クラスタの構成
Fig. 3　Structure of the GPU cluster used for our simulation.

各計算ノードのオペレーティングシステム（OS）は，Linux 64 bit 版である．C/C++コンパイラとして GNU コンパイラを使用した．GPU を利用した実行プログラムを作成するために，NVIDIA 社から提供される GPU アプリケーション開発環境である CUDA を使用した．GPU デバイスドライバのバージョンは 304.54 である．

3.3　計算手順と計算機リソースの割り振り

リスト 1 に，REMD シミュレーション全体の処理内容を表す擬似コードを示す．並列化のために MPI や OpenMP は使っておらず，クライアントノードでシリアルに実行される．この擬似コードを使い，GPU コアへの並列計算部分の割り振り，クライアントノードとリモート GPU 間データ通信量の見積りを説明する．

リスト 1 の 2～6 行目でシミュレーションの条件を設定する．n はリモート GPU の数である．本稿で扱った n の範囲は，1 から 1,024 までの範囲にある 2 の整数べきで表した整数である．N_{atom} は周期境界条件における基本セル内のアルゴン原子の個数であり，2.2 節に書いたように 256 個または 2,048 個の 2 通りを設定した．M_{iter} は，M_{exch} タイムステップの繰返し回数であり，計算時間の計測の便宜

表 1　GPU クラスタ中の計算ノード主要諸元 [2]
Table 1　The specification of GPU cluster used for evaluation [2].

ハードウェア環境（全計算ノード共通）	
CPU:	Intel Xeon X5670(6 cores)×2 ソケット
GPU:	NVIDIA 社 Tesla K20X ×3 デバイス
	14 Streaming Multiprocessors
	2,688 cuda cores
	6 GB GDDR5 メモリ
ネットワーク接続	InfiniBand
	32 Gbit/sec（理論最大スループット）
ソフトウェア環境（全計算ノード共通）	
オペレーティングシステム：	Linux 64 bit 版
	kernel version 2.6.32.12
GPU ドライバ：	NVIDIA 社 version 304.54
C/C++コンパイラ：	gcc version 4.3.4
	最適化オプション “-O2”
GPU ソフトウェア開発環境：	NVIDIA 社 CUDA Toolkit version 5.0
GPU 仮想化ソフトウェア：	DS-CUDA version 1.4.2 [20]

© 2014 Information Processing Society of Japan

リスト 1. 複数 GPU を用いた REMD シミュレーションの擬似コード

```
1:  /* シミュレーション条件の設定 */
2:  n ← GPU 並列数
3:  N_rep ← レプリカ個数
4:  N_atom ← レプリカあたりの原子個数
5:  M_exch ← 交換判定を行うタイムステップ数
6:  M_iter ← 交換判定を行う回数
7:  /* 初期状態の生成 */
8:  temp[N_rep] ← 各レプリカの初期温度条件
9:  pos[N_rep][N_atom][3] ← 各レプリカ内の原子の初期座標
10: vel[N_rep][N_atom][3] ← 各レプリカ内の原子の初期速度
11: flag[N_rep] ← 各レプリカの交換判定フラグ
12: U[N_rep][M_exch] ← 各レプリカのポテンシャルエネルギー
13: for d ← 0 to (n-1) do /* 各 GPU への初期データ転送 */
14:     SetDevice(d);
15:     MemcpyH2D(temp[N_rep/n · d], N_rep/n);
16:     MemcpyH2D(flag[N_rep/n · d], N_rep/n);
17:     MemcpyH2D(pos[N_rep/n · d][0][0], 3N_rep N_atom/n);
18:     MemcpyH2D(vel[N_rep/n · d][0][0], 3N_rep N_atom/n);
19: 各レプリカの速度スケーリングと初期緩和;
20: /* 本シミュレーション */
21: for 1 to M_iter do
22:     for d ← 0 to (n-1) do /* GPU へデータ送信#1 */
23:         SetDevice(d);
24:         MemcpyH2D(temp[N_rep/n · d], N_rep/n);
25:     for d ← 0 to (n-1) do /* GPU へデータ送信#2 */
26:         SetDevice(d);
27:         MemcpyH2D(flag[N_rep/n · d], N_rep/n);
28:     for d ← 0 to (n-1) do /* 複数の GPU で並列処理 */
29:         SetDevice(d);
30:         Kernel<<< N_rep/n, スレッド数 >>>(...);
31:     for d ← 0 to (n-1) do /* GPU からデータ回収 */
32:         SetDevice(d);
33:         MemcpyD2H(U[N_rep/n · d][0], N_rep M_exch/n);
34:     for all Replicas do /* クライアントノードで逐次処理 */
35:         P_exch(式 (1)) からレプリカ交換の成立と棄却を判定;
36:         変数 temp[N_rep] と flag[N_rep] を更新;
37: /* 後処理 */
38: シミュレーション結果をファイルに保存;
```

上 $N_{\mathrm{atom}} = 256$ の場合と 2,048 の場合でそれぞれ 1,800, 100 (回) とした．また 2.1 節で述べたように，レプリカ数 $N_{\mathrm{rep}} = 14{,}336$，レプリカ交換を行うまでのタイムステップ数 $M_{\mathrm{exch}} = 100$ である．

8〜12 行目でシミュレーションの初期状態を生成する．変数 $temp[N_{\mathrm{rep}}]$ は，各レプリカが持つ初期温度を表す．変数 $pos[N_{\mathrm{rep}}][N_{\mathrm{atom}}][3]$ および $vel[N_{\mathrm{rep}}][N_{\mathrm{atom}}][3]$ は，アルゴン原子の座標と速度のベクトルである．多次元配列の [3] は 3 次元ベクトルを表す．変数 $flag[N_{\mathrm{rep}}]$ は，式 (1) で計算された各レプリカでの交換の棄却と成立を表すフラグである．変数 $U[N_{\mathrm{rep}}][M_{\mathrm{exch}}]$ は，各レプリカのポテンシャルエネルギー状態のデータ保持に使用する．以上で述べた変数 *temp*，*pos* および *vel* の配列は，1 要素あたり 4 Byte 長の単精度浮動小数点数である．*flag* 配列は 1 要素あたり 4 Byte 長の整数である．

13〜18 行目で，生成した初期状態データを n 台のリモート GPU へ分配する．制御対象とするリモート GPU を選択する擬似命令を「SetDevice(“リモート GPU 番号”)」で表記した．SetDevice(d) 命令が実行された後は，その次に SetDevice() 命令が実行されるまでの間は，d 番目の GPU に対する制御が行われることを意味する．またクライアントノードからリモート GPU へのデータ転送を行う擬似命令を「MemcpyH2D(“転送される配列変数の最初の要素”, “転送される配列変数の要素の個数”)」で表記した．実際のソースコード上では，それぞれの擬似命令は CUDA API 関数である “cudaSetDevice()” および “cudaMemcpy(GPU メモリのアドレス, ホストメモリのアドレス, 送信サイズ (Byte),cudaMemcpyHostToDevice)” に対応する．19 行目で GPU カーネルを呼び出し，短いステップ数だけシミュレーションして原子の初期緩和を行う．

21〜36 行目までが，MD の時間発展を計算する本シミュレーションの計算部を表し，並列化効率の計測対象とした．1 台の GPU を使用した場合では，REMD シミュレーションの総計算時間の 99%以上がこの部分で消費される．22〜27 行目で，温度交換が行われて更新された温度情報と更新フラグをリモート GPU へ転送する．28〜30 行目までが，複数の GPU を使用して計算並列化を行ったブロックである．

GPU 上のプロセッサコアの割当てについて説明する．GPU の 1 つのブロックに 1 つのレプリカ（N_{atom} 個の原子）を担当させた．使用した GPU 1 台には 14 個の Streaming Multi Processor（SMX）が搭載されているので，1 台の GPU に割り当てられるレプリカ数（N_{rep}/n）が 14 の倍数になるように総レプリカ数を設定した．1 粒子が自身以外の $(N_{\mathrm{atom}} - 1)$ 個の粒子から受ける力（Lennard-Jones ポテンシャル，式 (2) 参照）の計算を，SMX 内に含まれる Scaler Proccessor（SP）の 1 つのスレッドが行う．使用した GPU では 1 つの SMX で同時に起動可能なスレッド数上限が 1,024 であったため，N_{atom} が 256 の場合には 1 粒子に 1 スレッドを割り当て，$N_{\mathrm{atom}} = 2{,}048$ の場合には 2 粒子分を 1 スレッドに割り当てた．GPU カーネルでは力やポテンシャルエネルギーの計算の後，原子座標と速度を更新し，それらを M_{exch} ステップ分繰り返す．

GPU 上のグローバルメモリの割当てについて説明する．シミュレーション実行中に定常的に割り当てておく必要のある主要なデータとして，全アルゴン原子（$N_{\mathrm{ALL}} = N_{\mathrm{rep}} \times N_{\mathrm{atom}}$ 個）の座標（リスト (1) の 9 行目）のおよび速度データ（リスト (1) の 10 行目），各レプリ

© 2014 Information Processing Society of Japan

カ系のポテンシャルエネルギー情報（リスト (1) の 12 行目）がある．これらのデータはリモート GPU のグローバルメモリ上にのみ配置し，クライアントノード上のメインメモリにはレプリカ 1 つ分の転送用バッファ以外には配置しない．全原子の座標および速度データとエネルギー情報の占める合計データサイズは，およそ 711 MByte $(=\{(N_{\mathrm{ALL}}\cdot 3\cdot 2)+N_{\mathrm{rep}}\cdot M_{\mathrm{exch}}\}\cdot 4$(単精度浮動小数点数バイト数)) である．このサイズは，使用した GPU 1 台に搭載されているグローバルメモリ容量 6 GByte より十分に小さいので，1 台の GPU 上のグローバルメモリ上でも全データを保持できるサイズである．n 台の GPU を並列化した場合は，GPU 1 台あたりに要求されるメモリサイズが $1/n$ になるのでメモリ不足は生じない．

31〜33 行目で，GPU で行った計算結果のうちポテンシャルエネルギーを回収する．「MemcpyD2H()」は，MemcpyH2D() 命令に比べて反対向きの転送（リモート GPU からクライアントノードへ）を行う擬似命令である．

34〜36 行目では，各レプリカのポテンシャルエネルギーをもとに，クライアントノードの CPU で式 (1) を計算して交換の棄却・成立 $flag$ と温度 $temp$ を更新する．

38 行目ではシミュレーション結果をファイルに保存する．

3.4 クライアントノードとリモート GPU 間の通信データ量

リスト 1 の 24，27 行目にある “MemcpyH2D()” の合計転送サイズを S_{H}，33 行目の “MemcpyD2H()” の転送サイズを S_{D} で表す．$temp$ と $flag$ の配列の大きさから，$S_{\mathrm{H}}=N_{\mathrm{rep}}\cdot 2\cdot 4=114{,}688$ Byte である．同様に，U の配列の大きさから $S_{\mathrm{D}}=N_{\mathrm{rep}}\cdot M_{\mathrm{exch}}\cdot 4=5{,}734{,}400$ Byte である．

4. 並列化効率のモデル式

本稿で扱った並列化効率 R の定義を以下に示す．

$$R=\frac{T_{\mathrm{local}}(1)}{n\cdot T(n)}. \tag{3}$$

R は，GPU 並列数 n のときに線形にストロングスケールした場合の理想的な計算速度に対する実効速度比を表す．分子にある $T_{\mathrm{local}}(1)$ は，図 2 の “local.exe(4)” で示したシミュレーション実行ファイルから，ローカル GPU 1 台を使った場合に要した実行時間を表す．$T(n)$ は，図 2 の “remote.exe(5)” の実行ファイルから，n 台のリモート GPU で並列化した場合の実行時間を表す．$T(n)$ および $T_{\mathrm{local}}(1)$ の計算時間は，リスト 1 に示した中の 21〜36 行目の処理に要した時間とし，シミュレーションの初期化部（19 行目以前）ならびに後処理部（38 行目）は R の計算に含めない．この R の算出に使用する 21〜36 行目の計算内容は，図 1 で示した M_{exch} ステップ分の計算時間と，t_1〜t_4 のタイミングで行われる温度交換判定（式 (1)）ならびに温度交換に要した時間の合計に相当する．$T_{\mathrm{local}}(1)$ の計測結果は，測定ばらつきを考慮して順に N_{atom} が 256 および 2,048 の場合について $M_{\mathrm{iter}}=\{1{,}800,\ 100\}$ の場合を計測した結果，$\{66706.951\,\mathrm{sec},\ 48288.290\,\mathrm{sec}\}$ を得た．これらの値から $M_{\mathrm{iter}}1$ 回に相当する値を $T_{\mathrm{local}}(1)$ とし，それぞれ 37.059 sec および 482.883 sec とした．

式 (3) に含まれるリスト 1 の処理内容から，$T(n)$ を見積もるために以下に示すような 3 つの部分からなるモデル式を考える．

$$T(n)=t_{\mathrm{GPU}}(n)+t_{\mathrm{COM}}(n)+t_{\mathrm{CLT}}, \tag{4}$$

ここで $t_{\mathrm{GPU}}(n)$ はリモート GPU における計算処理時間であり，リスト 1 の 28〜30 行目の部分のうち実際に GPU で計算が行われた時間に相当する．図 1 では M_{exch} で示したタイムステップ分の処理時間に相当する．$t_{\mathrm{COM}}(n)$ はリモート GPU とクライアントノード間に発生する通信時間である．これは，リスト 1 の 22〜27 行目の 2 回の SetDevice() と MemcpyH2D() の処理時間，31〜33 行目の SetDevice() と MemcpyD2H() の処理時間，29 行目および 30 行目の GPU カーネル呼び出しのオーバヘッド時間の和とした．t_{CLT} はクライアントノードにおける逐次計算処理時間を表し，式 (1) の判定時間である．リスト 1 の 34〜36 行目までの処理時間に相当する．

本章中の以降で，式 (4) の右辺の詳細について説明し必要なパラメータを決定する．

4.1 リモート GPU における計算処理時間：$t_{\mathrm{GPU}}(n)$

$t_{\mathrm{GPU}}(n)$ を次式のようにモデル化する．

$$t_{\mathrm{GPU}}(n)=\frac{t_{\mathrm{GPU0}}\cdot N_{\mathrm{rep}}}{n}. \tag{5}$$

右辺の t_{GPU0} は，1 レプリカの 1 温度交換インターバル期間 M_{exch} のシミュレーションを GPU 1 台で実行した場合に要する計算時間を表す．図 2 の “(4)local.exe” で示した実行ファイルを使用し，CUDA API に含まれる “cudaEventRecord()” 関数を利用して，ローカル GPU でリスト 1 の 28〜30 行目までの計測結果から求めた．

t_{GPU0} は，粒子間相互作用の計算を行うことから 1 レプリカあたりの粒子数 N_{atom} に依存する．$N_{\mathrm{atom}}=256$ 粒子，2,048 粒子の場合で，測定誤差を考慮しそれぞれの M_{iter} 回数分を計測した平均値として 37.034 sec および 482.367 sec を得た．これらの値から t_{GPU0} の値としてそれぞれ 2.58 msec，33.6 msec を使用した．本稿では各 GPU に割り当てられたレプリカ数は N_{rep}/n で等しくし，GPU 間の計算負荷の不均衡が生じないようにした．

4.2 クライアントノードとリモート GPU 間の通信時間：$t_{\mathrm{COM}}(n)$

$t_{\mathrm{COM}}(n)$ を次式のようにモデル化する．

© 2014 Information Processing Society of Japan

$$t_{\mathrm{COM}}(n) = \left(\frac{S_{\mathrm{H}}}{B_{\mathrm{H}}} + 2n \cdot L_{\mathrm{H}}\right) + \left(\frac{S_{\mathrm{D}}}{B_{\mathrm{D}}} + n \cdot L_{\mathrm{D}}\right) + n \cdot L_{\mathrm{K}}. \quad (6)$$

右辺の第1項はリスト1の22～27行目に示したクライアントノードからリモートGPU方向へのデータ転送MemcpyH2Dにかかる時間を表す．$2n \cdot L_{\mathrm{H}}$ の項に含まれる係数2は，GPU 1台あたり2回の転送命令（MemcopyH2D）を行うことに対応する．第2項は第1項と反対方向への通信時間を表し，31～33行目に示したリモートGPUからクライアントノードへの通信時間を表す．

L_{H} および L_{D} は，クライアントノードと1台のリモートGPUとの間のデータ転送1回あたりにかかる通信のオーバヘッド時間を表し「レイテンシ」と呼ぶ（単位はsec）．B_{H} および B_{D} は，クライアントノードと n 台のリモートGPUとの間のデータ転送のピークスループットを表す（単位はByte/sec）．それぞれに付加した添字 “$_{\mathrm{H}}$” でクライアントノードからリモートGPUへの転送方向を表し，“$_{\mathrm{D}}$” でリモートGPUからクライアントノードへの転送方向を表す．S_{H} および S_{D} は3.4節で述べた通信データ量である．

右辺3項目に含まれる L_{K} は，t_{GPU} の処理（28～30行目）を，リモートGPUで1回実行するのにかかるレイテンシを表す．L_{K} に含まれる内容は次の3つ，(1) リモートGPUでのcudaSetDevice()の処理時間，(2) リモートGPUへのカーネル関数の引数の送信時間，(3) リモートGPUでのカーネル関数のロード時間，である．第1項と第2項の S/B に相当する $S_{\mathrm{K}}/B_{\mathrm{K}}$ を式(6)に含めないのは，DS-CUDAの機能によりサーバノード側でカーネル関数の本体である.ptxファイルをキャッシュしており，REMDシミュレーションの実行中にInfiniBandネットワーク上で一度しか通信されないことによる．以降で各パラメータを計測または導出した．

4.2.1 通信ピークスループット B_{H}，B_{D} の算出

通信ピークスループットの定義を「クライアントノードからリモートGPUに対して，GPU仮想化ソフトウェアとCUDA API関数cudaSetDevice()とcudaMemcpy()を用いてデータ転送を行ったときの最大転送速度」とした．

図4に，本稿で扱った通信スループット B およびレイテンシ L の計測結果からの算出方法を示した．横軸は転送データサイズ S を示し，縦軸は転送にかかった時間 T_{CPY} を表す．式(6)で示したのと同様に $T_{\mathrm{CPY}} = S/B + L$ としてモデル化を行った．L および B の2つを未知数として，T_{CPY} を測定した結果（図4の白角シンボル）に最小2乗法を使ったフィッティングを行うことで B と L を算出した．グラフ中の破線がフィッティング後の近似式 $f_{\mathrm{FIT}}(S)$ を表す．レイテンシ L は $f_{\mathrm{FIT}}(0)$，スループット B は $f_{\mathrm{FIT}}(S)$ の傾きの逆数からそれぞれ算出した．以上に述べた算出手順から，データの転送方向別に行って得られた結果をそれぞれ B_{D} および B_{H} とした．図4のグラフは，クライアントノードからリモートGPU（1台）へデータ転送を行った場合の B_{H} および L_{H} を算出した例である．

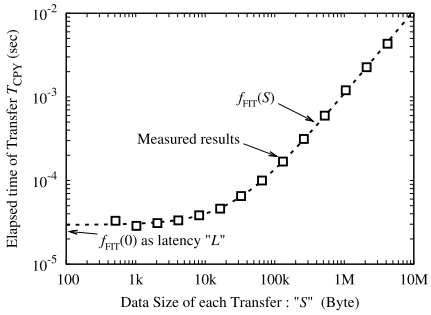

図4 転送データサイズと転送時間の関係，ならびにレイテンシ L とピークスループット B の算出方法

Fig. 4 Relation between the data size of each transfer and its elapsed time, and the calculation method of latency L and the peak throughput B.

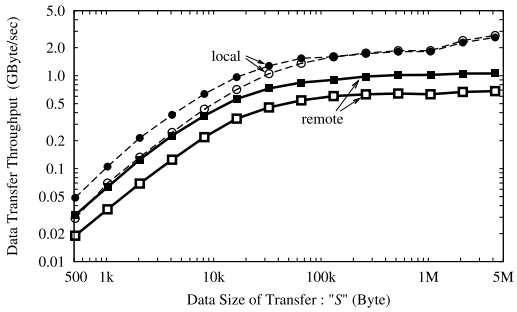

図5 クライアントノードと，ローカルGPUおよびリモートGPU間の実効転送スループットの測定結果

Fig. 5 Throughput between a client node and a remote GPU, or a local GPU.

図5に示したグラフは，T_{CPY} の実測結果から得られた転送データサイズと実効転送速度の関係を表す．このグラフはクライアントノードからリモートGPU 1台に対してデータ転送を行ったときの結果である．転送を実行するAPI関数としてcudaMemcpy()関数を使用し，その引数として指定するデータサイズを512 Byte～4 MByteまで変えて計測した．グラフ “remote” で示した黒四角シンボルがクライアントノードからリモートGPUへの転送方向を，白四角シンボルがその反対方向への転送方向を示す．比較参考データとして，ローカルGPUへの同様な実効転送速度を測定した結果も同図中グラフ “local” に併記した．黒丸シンボルがローカルGPUへの転送方向を，白丸シンボルがクライアントノードへの転送方向を示す．リモートGPUに対する転送スループットは100 k～5 MByteの転送サイズで飽和する傾向がある．ローカルGPUに対しては，100 k～1 MByteの転送サイズでいったん飽和するように見えるが1 MByte以降でスループットが増加する傾向がある．本稿ではリモートGPUの性能の挙動に主眼を置いていることを考慮し，1 GPUを使用する場合でもローカルGPUは使わずリモートGPUを使うこととした．

© 2014 Information Processing Society of Japan

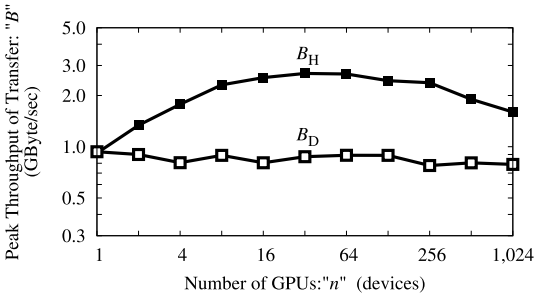

図 6　クライアントノードとリモート GPU 間の通信速度のピークスループットの測定結果

Fig. 6　Measurement result of the peak thoughput between a client node and remote GPUs.

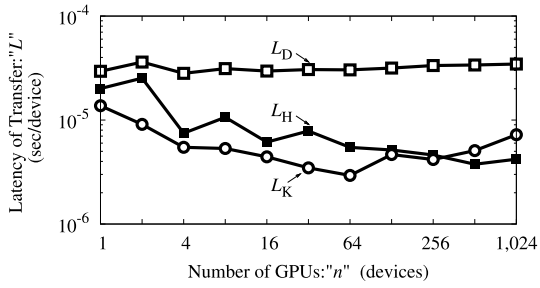

図 7　クライアントノードとリモート GPU 間の各通信レイテンシの測定結果．式 (6) 中の 3 つのパラメータ，L_D，L_H および L_K の算出に使用した測定結果をそれぞれ順に白角，黒角および白丸で表す

Fig. 7　Measurement results of the each latencies L_D, L_H and L_K in t_{COM} (6) between a client node and remote GPUs.

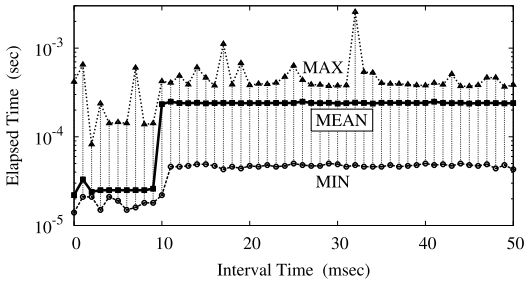

図 8　ローカル GPU への cudaMemcpy(DeviceToHost) 実行時間の実行間隔への依存性

Fig. 8　Interval time dependancy of the elapsed time of “cudaMemcpy(DeviceToHost)” from local GPU.

B_{D} の場合と B_{H} の場合とで，算出時に行った測定手順が異なるのでそれぞれを順にリスト (1) を使って説明する．B_{D} の場合は，リスト (1) の 22〜30 行目および 34 行目の for ループ部をコメントアウトし，コメントアウトされた部分の処理時間の代わりとして 20 msec のウェイト時間を 31 行目の for ループの直後に挿入する．この状態で，21〜36 行目までの実行時間をウェイト時間を除外して計測した．計測結果はネットワークの使用状況や OS ノイズ等の要因により誤差をともなうため，M_{iter} で表す複数回の測定の平均値を T_{CPY} とした．以上の計測を 33 行目で指定する転送サイズを 512 Byte〜4 MByte まで変えて行った．

B_{H} の算出に必要な T_{CPY} を測定する場合は，リスト (1) の 25〜36 行目までをコメントアウトし，その部分に「すべての GPU のバリア同期処理」および B_{D} の場合と同様に「20 msec のウェイト時間」を挿入する．B_{D} の場合と異なりバリア同期処理を入れるのは，クライアントノードからリモート GPU への転送が非同期に実行されるためである．この状態で，21〜36 行目の実行時間をウェイト時間およびバリア同期時間は除外して計測する．B_{D} のときと同じく複数回の測定の平均値を T_{CPY} とした．

以上に述べた方法で，データ転送対象の GPU 台数を 1〜1,024 台の範囲で変えながら，クライアントノードとリモート GPU 間のピークスループット値を計測した結果を**図 6** に示す．転送対象の GPU は $n = 1$ の場合を含みすべて仮想化されたリモート GPU を対象とした．B_{H} を黒角シンボルのグラフ，B_{D} を白角シンボルのグラフで記載した．図 6 の測定結果から，並列化効率モデル式 (6) の通信スループットパラメータ B_{H} および B_{D} に，測定ポイント 11 点の平均値を適用し，それぞれ 1,960 MB/sec，850 MB/sec とした．

4.2.2　通信レイテンシ L_{H}，L_{D}，L_{K} の算出

本稿ではレイテンシを「GPU 仮想化ソフトウェアを利用して GPU へのデータ転送やカーネル実行を行う際の，データ転送やカーネル実行が始まるまでのオーバヘッド時間」として扱っている．レイテンシには，物理的な信号遅延時間・通信用バッファへのデータコピー時間・通信ライブラリの処理時間・CUDA 関数である cudaMemcpy() の実行時間のオーバヘッドまたはカーネル呼び出しのオーバヘッドが含まれる．前節でのスループットの算出時に求めた L の値をグラフ化したものを**図 7** に示す．測定方法は，4.2.1 項で述べた B_{H} および B_{D} の計測方法と同様である．測定結果から，並列化効率モデル式 (6) の通信レイテンシパラメータ L_{H} および L_{D} に，測定ポイント 11 点の平均値を適用し，それぞれ 9.23 μs，31.7 μs とした．

レイテンシの測定にあたり，クライアントノードとローカル GPU 間の転送時間が不連続に変化する現象が見られた．この現象を**図 8** のグラフを使用して説明する．このグラフは，ローカル GPU への cudaMemcpy(DeviceToHost) を繰り返し実行したときの実行時間の平均値（“MEAN” のグラフ），最大値（“MAX” のグラフ）および最小値（“MIN” のグラフ）を繰り返す間隔を横軸にとり計測した結果を示す．ある繰返し間隔それぞれにつき 10,000 回の計測を行った結果を記載した．転送サイズは 1 回あたり 1,024 Byte の固定サイズである．横軸で示した実行間隔が，10 msec を超えたあたりから不連続に増加することを表している（この現象について NVIDIA 社のサポートに問い合わせ，NVIDIA 側で再現することまで確認したが原因は調査中）．この現象によりレイテンシが増加することを避けるため，

© 2014 Information Processing Society of Japan

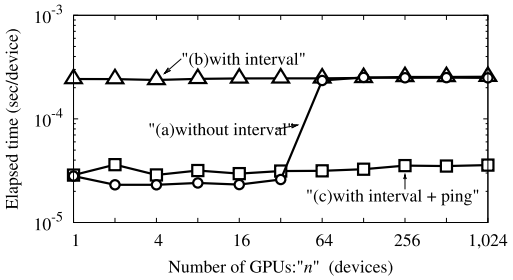

図 9　リモート GPU からクライアントノードへのデータ転送に要した時間．ダミー転送ルーチンの追加のよる効果

Fig. 9　Elapsed time of “cudaMemcpy(DeviceToHost)” from remote GPUs to client node. The effect by the addition of dummy transfer routine.

DS-CUDA サーバプログラムに対して「リモート GPU で REMD 計算とデータ転送を行っていないときに，ダミーデータ転送を 2 msec 周期で発行する」機能を追加したうえで，通信パラメータと並列化効率を計測した．実行時間の増加する境界が 10 msec 付近であることに対して，機能上ではそれよりも短い 2 msec に設定した理由は，ある程度のタイミングマージンを見込んだことによる．

図 9 に，上述のダミー転送ルーチンの有無によって n 台のリモート GPU からクライアントノードへのデータ転送時間への影響を比較した結果を示す．転送対象の GPU は $n = 1$ の場合でも仮想化されたリモート GPU を対象とした．縦軸は 1 GPU あたりの実行時間を示す．計測条件として，データサイズは GPU 1 台につき 1,024 Byte とした．図 8 で示した現象を考慮して，次の 3 つの条件を設定して計測した．グラフ (a) は interval 期間を設けない場合，グラフ (b) は interval 期間を設けダミー転送ルーチンなしの場合，グラフ (c) は，interval 期間を設けかつダミー転送ルーチンありの場合，をそれぞれ表す．interval 時間とは，リスト (1) において実際のリモート GPU への転送時間以外にかかる時間の代替として仮に設定した 20 msec 期間のアイドル時間である．リスト (1) の 31 行目のように for 文によって各 GPU に対し順番に MemcpyD2H() が発行されるため，特定の GPU にとっては次の転送までにある程度の時間がかかる．その時間によって図 8 のようにレイテンシが大きく変わることになる．

グラフ (b) の場合は，interval 期間があるために必ず 20 msec 以上の転送実行間隔があき，図 8 の Interval Time が 10 msec 以上の場合の MEAN 値である 200～300 μsec 付近で一様な結果となる．グラフ (a) は，GPU 並列数が 32 台より多くかつ 64 台以下の範囲で，データ転送時間が不連続に増加した結果となった．これは次の理由によると考えられる．GPU の台数が少ない場合は，より短いインターバルでデータ転送が繰り返されるためレイテンシは 20～30 μsec となる．しかし台数が増加するに従って GPU 1 台に対する転送インターバル時間は長くなり，ある台数（32 台より多くかつ 64 台以下）を境界としてレイテンシが急増し，いったん増加すると短いインターバルには戻らなくなってしまうことから 200～300 μsec の時間がかかる．グラフ (c) では，図 8 の Interval Time が 10 msec 以下の場合の MEAN 値である 20～30 μsec 付近で一様な結果となった．この結果よりダミー転送ルーチンが有効に機能していると判断した．

L_{K} を算出した方法を説明する．リスト (1) の 30 行目の “Kernel” 関数と同じ引数を持ち負荷の軽いダミー処理（グローバルメモリを 4 Byte 書き換える）行う “DKernel” 関数を準備し，Kernel 関数と差し替える．22～27 行目，および 31～36 行目をコメントアウトする．その部分に「すべての GPU のバリア同期処理」を挿入する．この状態で，21～36 行目までの実行時間を計測した．計測結果はネットワークの使用状況や OS ノイズ等の要因により誤差をともなうため，M_{iter} で表す複数回の測定の平均値を L_{K} とした．

GPU 並列数 1～1,024 台の範囲で，L_{H}，L_{D} と同様に測定した結果を図 7 中のグラフ L_{K} に示す．測定結果から，並列化効率モデル式 (6) の GPU カーネル関数実行オーバヘッド L_{K} の値として，測定ポイント 11 点の平均値 5.96 (μsec) を使用した．

4.2.3　通信時間パラメータ計測結果に対する考察

図 5 に示したスループットを測定したグラフは，同様の測定を TSUBAME2.0 の環境で行った結果が参考文献 [19] の Figure 4 に示されている．TSUBAME2.0 から本稿で使用した同 2.5 へのアップデートにより高性能な GPU に差し替えられた点で環境の違いはあるが両者を比較した．全体的な傾向として，リモート GPU 間への転送ピークスループットが 1.0 GByte/sec で飽和すること，ローカル GPU に対しては最大で 3.0 GByte/sec のピークスループットが計測されたこと，という点で一致している．

図 6 に示したピークスループットを表す 2 つのグラフ B_{H} と B_{D} において，B_{H} のグラフにのみ GPU 並列数 n が 64 までピークスループットの上昇する傾向が見られた．これは，DS-CUDA 内で RDMA 機能を利用しており通信処理がハードウェア的にサポートされることに起因する．送受信処理が CPU とネットワーク装置で並列処理されることで，結果として 1：1 通信より 1：多通信の方が転送効率が改善されていると考えられる．B_{H} の最大値付近ではノード間接続の InfiniBand の理論最大スループット 3.2 GByte/sec の 84%の通信効率を示した．

図 7 のレイテンシを表すグラフにおいて，L_{H} と L_{K} が，L_{D} と比べて N_{GPU} が 64 並列以下の範囲で減少する傾向が見られるのは図 6 の B_{H} の場合と同じ理由によると考えれる．一方，L_{D} が GPU 並列数に対してほぼ一様な特性を示すのは，MemcpyD2H の転送処理はその後の命令をブロックして逐次的に実行されるために並列化がなされない

© 2014 Information Processing Society of Japan

ことによる．

4.3　クライアント計算ノードにおける逐次処理時間：t_{CLT}

式 (4) におけるクライアントノードの処理時間 t_{CLT} で行う逐次処理内容は式 (1) に示した確率 P_{exch} の計算であり，具体的には次の処理を行う．(1) GPU から回収したレプリカごとのポテンシャルエネルギー値から温度交換条件式の判定．(2) 温度交換情報と交換フラグの更新．したがって t_{CLT} は，各レプリカが含む粒子数 N_{atom} および GPU 並列数 n に依存せず，レプリカ個数 N_{rep} と温度交換判定周期 M_{exch} に比例する．本稿ではそれぞれ $N_{\mathrm{rep}} = 14{,}366$ (個) および $M_{\mathrm{exch}} = 100$ (steps) に固定値として設定していることから t_{CLT} も固定値になる．

REMD シミュレーションのソースコードから GPU に関係する処理部を除いた実行ファイルを作成して処理対象部を計測するために，リスト (1) の 22〜33 行目をコメントアウトし，21〜36 間の実行時間を計測した．測定ばらつきを考慮してループ回数を決める M_{iter} を 200 回に設定し $(200 \cdot t_{\mathrm{CLT}})$ の実行時間を計測した結果 1.591 (sec) が得られたことから，t_{CLT} を 7.96 (msec) とした．

5.　並列化効率の実測結果

5.1　モデル式との比較

4 章で求めた REMD シミュレーションにおける並列化効率 R のモデル式 (3) から算出した期待値と，実際に GPU クラスタ（図 3）上で DS-CUDA（図 2）を利用して実行時間を計測した結果とを**図 10** に示した．モデル式を実線および破線グラフで表し，実測結果を三角および四角のシンボルで表している．縦軸が並列化効率 R であり，リニアにストロングスケールする計算速度に対する実効計算速度比を表す．横軸は GPU 並列数 n であり，16 から 4,096 台の範囲を記載した（n が 1 から 8 台の場合には，いずれの場合もモデル式と実測結果がともに 0.98 以上 1.0 以下であり，理想に近いリニアスケールを示したため省略）．実測結果は，リスト (1) に対応するシミュレーションソースコード内に時間計測関数を挿入し，モデル式の $T(n)$ に相当する処理部の実行時間の計測した結果である．

図 10 内の左右のグラフ (i)(ii) は，(i) が原子個数 N_{atom} が 256 粒子の結果を，(ii) が N_{atom} が 2,048 粒子のときの結果をそれぞれ表す．N_{atom} の大小は並列計算の粒度の違いを意味し，式 (5) に含まれるローカル GPU での計算時間 t_{GPU0} のそれぞれの値から，(i) より (ii) の方がより粗粒度な計算条件である．したがって (i) よりも計算粒度の大きい (ii) の条件の方がシミュレーション時間全体に占める GPU での計算時間が長いことから，全体的により高い効率が得られている．

またグラフ (i)(ii) それぞれに，“modified” および “not modified” ラベルで示した 2 つのモデル式と実測結果のペアの結果を記載した．“modified” グラフの方は，4.2 節で述べたレイテンシの増加を抑制するためのダミー転送ルーチンを DS-CUDA へ追加した場合の結果である．もう一方の “not modified” グラフはその処理を追加しなかった場合の結果であり，レイテンシ低減処理の効果確認のための比較用データとして記載した．“not modified” グラフが “modified” よりも全体的に低い効率を示していることにより，レイテンシ低減処理の効果を確認した．

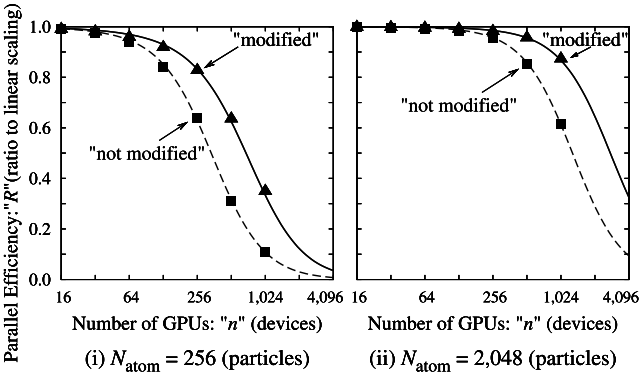

図 10　並列化効率の実測結果とモデル式の比較
Fig. 10　Mesurement result of parallel efficiency compared with its estimation model.

並列化効率モデル式は，実測された結果と誤差 3%の範囲で一致した．1,024 GPU で並列化した場合に，$N_{\mathrm{atom}} = 256$ のとき 35%，$N_{\mathrm{atom}} = 2{,}048$ のとき 87%の効率を達成した．

5.2　消費時間の内訳に関する考察

式 (4) に示したシミュレーション実行時間 $T(n)$ 中の各要因 $\{t_{\mathrm{GPU}}(n),\ t_{\mathrm{COM}}(n),\ t_{\mathrm{CLT}}\}$ の消費時間，およびその $T(n)$ 中に占める割合を算出した結果を**表 2** に示す．通信時間を表す t_{COM} については，式 (7) に示すように，ピークスループットに起因する要因 $t_{\mathrm{COM:B}}$ とレイテンシに起因する要因 $t_{\mathrm{COM:L}}$ に特に分けて記載した．

$$t_{\mathrm{COM}} = t_{\mathrm{COM:B}} + t_{\mathrm{COM:L}}, \tag{7}$$

式 (7) の右辺に含まれる 2 項は，式 (6) で使用している記号を用いてそれぞれ $t_{\mathrm{COM:B}} = (\frac{S_{\mathrm{H}}}{B_{\mathrm{H}}} + \frac{S_{\mathrm{D}}}{B_{\mathrm{D}}})$，$t_{\mathrm{COM:L}} = (2n \cdot L_{\mathrm{H}} + n \cdot L_{\mathrm{D}} + n \cdot L_{\mathrm{K}})$ で表される．n が小さい場合には，t_{GPU} が 100%に近い割合を占めるために省略し GPU 並列数 n が $\{1,\ 64,\ 256,\ 1{,}024,\ 4{,}096\}$ の場合を記載した．$n = 1$ の場合は式 (3) に示したように並列化効率 R の定義式の分子で利用するため特に記載している．また $n = 1$ の場合のみ，DS-CUDA を使用した場合，ならびに CUDA からローカル GPU を使った場合の 2 通りを参考値として併記し，それぞれ表中の $n = 1$，$n = (\mathrm{local})1$ の行に示した．並列化効率 R と実行時間 $T(n)$ は，モデル式算出値を R_0 および $T_0(n)$，実測値を R_1 および $T_1(n)$ とそれぞれに下添字を付加して区別し，表の右端部 4 列に示した．$n = (\mathrm{local})1$ の場合の “実測効率 R_1” の値には，効率の基準値という意

© 2014 Information Processing Society of Japan

表 2　並列化効率モデル式における各要因の消費時間とその割合，ならびに実測値との比較
Table 2　Elapsed time and ratio of each factor in parallel efficiency model, and the measurement result.

GPU 並列数 n	GPU 計算時間:[ms] $t_{\mathrm{GPU}}(n)$	通信時間: $t_{\mathrm{COM}}(n)$ [ms] スループット:$t_{\mathrm{COM:B}}$ $S_{\mathrm{H}}/B_{\mathrm{H}}$	$S_{\mathrm{D}}/B_{\mathrm{D}}$	レイテンシ:$t_{\mathrm{COM:L}}$ $2n\cdot L_{\mathrm{H}}$	$n\cdot L_{\mathrm{D}}$	$n\cdot L_{\mathrm{K}}$	逐次処理時間:[ms] t_{CLT}	合計時間 [ms] $T_0(n)$	並列化効率: R_0	実測時間 [ms] $T_1(n)$	実測効率 R_1
(i) $N_{\mathrm{atom}}=256$ の場合											
(local)1	37,034(–%)	–	–	–	–	–	8(–%)	–	–	37,059	1.00
1	37,034(>99%)	0.06(<1%)	6.7(<1%)	0.02(<1%)	0.03(<1%)	0.006(<1%)	8(<1%)	37,049(100%)	>0.99	37,064	0.99
64	579(97%)	0.06(<1%)	6.7(1%)	1.2(<1%)	2.0(<1%)	0.4(<1%)	8(1%)	597(100%)	0.96	604	0.95
256	145(83%)	0.06(<1%)	6.7(4%)	4.7(3%)	8.1(5%)	1.5(1%)	8(5%)	174(100%)	0.83	175	0.82
1,024	36(33%)	0.06(<1%)	6.7(6%)	19(17%)	32(30%)	6.1(6%)	8(7%)	108(100%)	0.33	104	0.35
4,096	9.0(4%)	0.06(<1%)	6.7(3%)	76(30%)	130(51%)	24(10%)	8(3%)	254(100%)	0.03	–	–
(ii) $N_{\mathrm{atom}}=2{,}048$ の場合											
(local)1	482,367(–%)	–	–	–	–	–	8(–%)	–	–	482,883	1.00
1	482,367(>99%)	0.06(<1%)	6.7(<1%)	0.02(<1%)	0.03(<1%)	0.006(<1%)	8(<1%)	482,383(100%)	>0.99	482,884	0.99
64	7,536(99%)	0.06(<1%)	6.7(<1%)	1.2(<1%)	2.0(<1%)	0.4(<1%)	8(<1%)	7,555(100%)	0.99	7,589	0.99
256	1,884(98%)	0.06(<1%)	6.7(<1%)	4.7(<1%)	8.1(<1%)	1.5(<1%)	8(<1%)	1,913(100%)	0.98	1,922	0.98
1,024	471(86%)	0.06(<1%)	6.7(1%)	19(3%)	32(5%)	6.1(1%)	8(1%)	543(100%)	0.86	540	0.87
4,096	118(32%)	0.06(<1%)	6.7(1%)	76(20%)	130(35%)	24(6%)	8(6%)	362(100%)	0.32	–	–

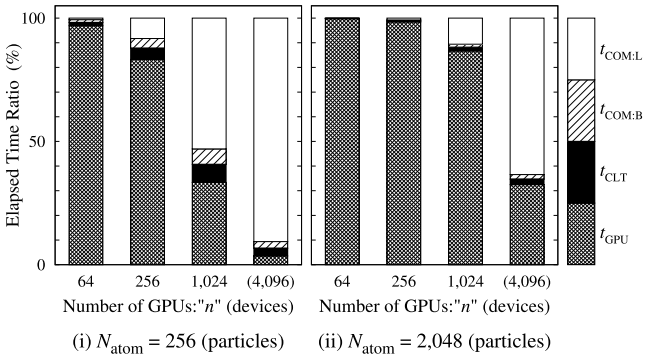

図 11　GPU 並列数 n が 64〜4,096 台のときのシミュレーション消費時間の割合
Fig. 11　The ratio of elapsed time of each factor using 64, 256, 1,024 and 4,096 GPUs.

味から 1.00 を記載している．$N_{\mathrm{atom}}=256$ および 2,048 の両方の場合で，$n=1$ のときのモデル式の合計時間 T_0 の算出値が $n=$ (local)1 の実測結果 T_1 よりも短くなっているが，これは t_{GPU0} の測定誤差によるものと考えられる．

N_{atom} が (i) 256 の場合と (ii) 2,048 の場合でそれぞれ上下 2 段から構成される表は，図 10 の並列化効率を示した (i)(ii) 中の “modified”（実線）で示したグラフに対応する．並列数 $n=64$ のときは，(i)(ii) ともに原子数にかかわらず消費時間の 97%以上が GPU での計算時間 t_{GPU} で占められている．これは GPU コアで行われた計算以外の通信オーバヘッド時間および並列化しなかったクライアントノードでの逐次処理時間 t_{CLT} の $T(n)$ に占める割合がいずれも小さく（3%以下）おおむね理想的な並列化効率が得られることを示す．並列数 n が増加するに従って，t_{GPU} は反比例して減少するが，通信時間は増加する（t_{CLT} は n によらずに一定）．したがって表 2 の括弧内に示した $T_0(n)$ を占める割合をグラフで表した**図 11** から分かるように，全体で消費される時間 $T(n)$ において通信時間 t_{COM} の割合が増加する（図中のバーグラフは，白抜きが $t_{\mathrm{COM:L}}$，斜めハッチが $t_{\mathrm{COM:B}}$，黒ベタが t_{CLT}，クロスハッチが t_{GPU} の $T_0(n)$ に占めるそれぞれの割合を表す）．

図 11 中の $t_{\mathrm{COM:L}}$ の占める割合から，実施した REMD シミュレーションの条件において高並列時の効率 R を低下させるボトルネックは，特に通信レイテンシが支配的である．その傾向が顕著に見られる並列数 n が 1,024 台での通信時間中での通信レイテンシの割合について考える．(i) $N_{\mathrm{atom}}=256$ 粒子の場合では，通信時間の 89%が通信レイテンシで消費されている．(ii) $N_{\mathrm{atom}}=2{,}048$ 粒子の場合では，GPU での計算時間 t_{GPU} の割合は増加するが，通信時間内でのレイテンシの割合は (i) と同じである．$n=4{,}096$ 台で並列計算を行った場合は，計測を行っていないが (i) $N_{\mathrm{atom}}=256$ 粒子の場合で全シミュレーション時間の 90%がレイテンシに起因する通信時間で消費されるだろうことを示している．

実施した REMD シミュレーション条件において，ノード間通信の高速化によって並列化効率を向上を試みた場合に予測される効果を，表 2 の上段 (i) およびそれに対応するグラフ図 11 の (i) における $n=1{,}024$ の場合で述べる．通信スループット $t_{\mathrm{COM:B}}$ が占める通信時間の割合は消費時間全体 $T(n)$ の 6%であるので，仮にスループット性能のみを倍速にできた場合は，実行時間を 3%短縮することができる．通信レイテンシ $t_{\mathrm{COM:L}}$ が占める通信時間の割合は 53%であるので，仮にレイテンシのみを半分にした場合は，実行時間を 26.5%短縮することができる．より高い並列化効率を得るためには，通信スループットの高速化よりはレイテンシの低減化の方が有効である．

© 2014 Information Processing Society of Japan

表 3　リスト 1 を実行したときの消費時間とその割合
Table 3　The total elapsed time of application and its ratio.

GPU 並列数 n	初期化部 1〜19 行目 [s]	メインループ部 20〜36 行目 $T(n)$ [s]	後処理部 37〜38 行目 [s]	合計 [s]
(i) $N_{\mathrm{atom}} = 256$ の場合 ($M_{\mathrm{iter}} = 1{,}800$)				
(local)1	0.8 (<0.1%)	$37.059 \times M_{\mathrm{iter}}$ (99.7%)	171.4 (0.3%)	66,878.4 (100%)
1	1.5 (<0.1%)	$37.064 \times M_{\mathrm{iter}}$ (99.8%)	148.6 (0.2%)	66,865.3 (100%)
(ii) $N_{\mathrm{atom}} = 2{,}048$ の場合 ($M_{\mathrm{iter}} = 100$)				
(local)1	1.8 (<0.1%)	$482.883 \times M_{\mathrm{iter}}$ (99.7%)	164.9 (0.3%)	48,455.0 (100%)
1	3.0 (<0.1%)	$482.884 \times M_{\mathrm{iter}}$ (99.4%)	280.7 (0.6%)	48,572.1 (100%)

5.3　REMD シミュレーション全体の実行時間

並列化効率 R（式 (3)）については，3.3 節で述べたようにリスト 1 の擬似コード中のシミュレーションの時間発展に関するメインループ部のみを対象とした．本節ではリスト 1 中のすべての処理の実行時間の実測結果について述べる．**表 3** に並列化効率 R の定義に含めなかったシミュレーションの初期化部および後処理部を含めた実行時間の $n = 1$ の場合の実測結果を示した．初期化部はリスト 1 の 1 行目から 19 行目まで，後処理部は 37 行目から最終行までである．

表 3 に示したように，初期化部の処理では数 sec 程度の時間が消費される．後処理部では，148〜280 sec の実行時間がかかっている．本稿で実施した $n = 1$ の場合のメインループ回数 M_{iter} では，$N_{\mathrm{atom}} = 256$ の場合で総実行時間の 0.3%，$N_{\mathrm{atom}} = 2{,}048$ の場合で 0.6%に相当し無視できる程度であるといえる．実際に固体–液体相転移温度を求めるシミュレーションを行うときは $M_{\mathrm{iter}} = 10^5$ 以上になるため，仮に $n = 1{,}024$ の並列度になってメインループ部分が 1,000 倍加速されてもメインループ部分以外の部分にかかる時間は 1%未満であり，ほぼ無視できると考えられる．

6.　まとめと課題

シングルノード上のマルチ GPU 向けに記述されたレプリカ交換法分子動力学（REMD）シミュレータのソースコードに対して，GPU 仮想化ソフトウェア DS-CUDA を適用し，そのソースコードを改変することなくマルチノード上の GPU を使用した並列化計算に対応させた．最大 1,024 台までの GPU を使用して並列化効率を計測した結果，1,024 台並列時に 87%と良好な結果を得た．

REMD シミュレータのソースコードとそのアルゴリズムから並列化効率のモデル式を構築し，GPU クラスタ上で計測した通信速度パラメータ（ピークスループットおよびレイテンシ）とシングルノード使用時の計算時間を併用し，並列化効率の期待値を算出した．その期待値は実測結果と誤差 3%以内で一致した．特に GPU 多並列時における効率の低下要因として，ノード間通信のスループットよりはレイテンシの占める割合が支配的であることを述べた．またレイテンシ削減のために，あえてダミーの cudaMemcpy() を発行することが有効であることも示した．

本稿では，計算に使用したすべての GPU および計算ノードの計算速度性能と通信性能が一様であるホモジニアスな GPU クラスタを使用した．また計算実行中には，通信/計算のエラーや装置の故障等低い確率で発生する障害は起きないことを前提としている．しかし今後も装置の規模が拡大し続けるだろう高性能計算機環境において，これらの前提は現実的でない．本稿中で扱った，たかだか 343 ノード（GPU 1,024 台を含むサーバノードとクライアントノード）を使用した並列計算の場合ですら，まれにクライアントからの通信に正常に応答しない GPU デバイスが見られた．そのために，あらかじめ GPU クラスタから数台多めのノードを確保しておき，異常な GPU デバイスが見つかった場合には DS-CUDA の設定を変更することで予備の GPU デバイスと差し替えて使うことで対応した．今後の課題として，部分的に異常な計算ノードまたは GPU デバイスが確認された場合には自動的にそれらを計算リソースから除外するような耐故障性機能を実装し，そのオーバヘッドを考慮した性能評価を行うことがあげられる．

また，通信エラー等の検出しやすい異常に比べて検出の難しい計算エラーが発生した場合も考慮する必要がある．DS-CUDA には，複数の GPU に同じ内容の計算を実行させて結果を比較し，異なる結果が得られた場合を計算エラーと判定して自動的に再計算を行う機能がある（本稿では利用していない）．今後はこのような機能を活用し，必要に応じて機能追加・拡張することで，高性能計算機環境で確率的に異常が発生する，ということを前提にした計算性能の評価することが課題である．

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．すべての計算は東京工業大学のスパコン TSUBAME 2.5 を用いて行った．使用した GPU 仮想化ソフトウェア “DS-CUDA” の主開発者であり，利用にあたりアドバイスをくださった株式会社 K&F Computing Research の川井敦氏に深謝します．

参考文献

[1] TOP500 Supercomputer Site (online), available from ⟨http://www.top500.org/⟩ (accessed 2013-12-01).
[2] 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⟩（参照 2014-03）.
[3] NVIDIA: CUDA C Programming Guide (online), available from ⟨http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html/⟩ (accessed 2013-12-01).

© 2014 Information Processing Society of Japan

[4] OpenACC (online), available from ⟨http://www.openacc-standard.org/⟩ (accessed 2013-12-01).
[5] OpenCL (online), available from ⟨http://www.khronos.org/opencl/⟩ (accessed 2013-12-01).
[6] Yang, J., Wang, Y. and Chen, Y.: GPU accelarated molecular dynamics simulation of thermal conductivities, *Journal of Computational Physics*, Vol.221, pp.799–804 (2007).
[7] Rodrigues, C.I., Hardy, D.J., Stone, J.E., Schulten, K. and Hwu, W.-M.W.: GPU Acceleration of Cutoff Pair Potentials for Molecular Modeling Applications, *Proc. 5th conference on Computing frontiers*, pp.273–282 (2008).
[8] 坂牧隆司，成見　哲，泰岡顕治：グラフィックカードを用いた水表面張力の高速分子動力学シミュレーション，情報処理学会論文誌コンピューティングシステム（ACS），Vol.2, No.2, pp.89–97 (2009).
[9] 宇田川拓郎，関嶋政和：GPU を用いた分子動力学法の高速化と省電力化，情報処理学会研究報告，Vol.2010-HPC-127, No.5, pp.1–5 (2010).
[10] 井上弘士，安藤壽茂，薄田竜太郎，山村周史，柴村英智，三輪英樹，本田宏明，稲富雄一，眞木　淳，平尾智也，青柳睦，村上和彰，石附　茂，小松秀実，安島雄一郎，三吉郁夫，清水俊幸，黒川原佳：大規模スーパコンピュータ向けシステム性能評価環境の構築，情報処理学会論文誌コンピューティングシステム（ACS），Vol.4, No.1, pp.1–23 (2011).
[11] Sugita, Y. and Okamoto, Y.: Replica-Exchange molecular dynamics method for protain folding, *The Journal of Chemical Physics*, Vol.65, pp.1604–1608 (1999).
[12] Muguruma, C., Okamoto, Y. and Mikami, M.: New approach to the first-order phase transition of Lennard-Jones fluids, *The Journal of Chemical Physics*, Vol.120, pp.7557–7563 (2004).
[13] Muguruma, C., Okamoto, Y. and Mikami, M.: 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).
[14] 佐藤　仁，伊藤正勝，中田秀基，松岡　聡：レプリカ交換分子動力学シミュレーター REMD Toolkit のグリッド上での実行，情報処理学会報告 2003-HPC-95, pp.41–46 (2003).
[15] 伊藤正勝，長嶋雲兵：レプリカ交換分子動力学法を用いたシミュレーション並列化フレームワーク，情報処理学会論文誌：コンピューティングシステム（ACS），Vol.45, No.6, pp.234–243 (2004).
[16] Okabe, T., Kawata, M., Okamoto, Y. and Mikami, M.: Replica-exchange Monte Carlo method for the isobaric-isothermal ensemble, *Chemical Phisics Letters*, Vol.335, pp.435–439 (2001).
[17] Sugita, Y., Kitao, A. and Okamoto, Y.: Multidimensional replica-exchange method for free-energy calculations, *The Journal of Chemical Phisics*, Vol.113, No.15, pp.6042–6051 (2000).
[18] Jaros, J.: Multi-GPU Island-Based Generic Algorithm for Solving the Knapsack Problem, *World Congress on Computational Intelligence 2012*, Brisbane, Australia (2012.06).
[19] Kawai, A., Yoshikawa, K., Yasuoka, K. and Narumi, T.: Distributed-Shared CUDA: Virtualization of Large-Scale GPU Systems for Programmability and Reliability, *Future Computing 2012*, Nice, France (2012.07).
[20] 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⟩（参照 2014-06-20）.

老川 稔　（正会員）

1974 年生．1998 年群馬大学大学院工学研究科電気電子工学専攻修士課程修了．同年富士通ディジタルテクノロジ（株）入社，半導体集積回路設計に従事．2013 年千葉大学大学院工学研究科人工システム科学専攻博士課程修了．分子動力学シミュレーションの高速化に従事．

野村 昂太郎

1989 年生．2012 年慶應義塾大学理工学部機械工学科卒業．同年から現在まで同大学大学院理工学研究科に所属．GPU を用いた分子動力学シミュレーションの高速化に従事．

泰岡 顕治

1968 年生．1997 年名古屋大学大学院工学研究科応用物理学専攻博士課程修了．同年理化学研究所基礎科学特別研究員．1998 年慶應義塾大学理工学部助手．2000 年同大学専任講師．2004 年同大学助教授．2007 年同大学准教授．2010 年同大学教授．分子動力学シミュレーションを用いた界面現象，核生成現象等の研究に従事．博士（工学）．2000 年 IEEE Gordon Bell Prizes，2006 年に分子シミュレーション研究会学術賞各受賞．日本機械学会，日本伝熱学会，日本熱物性学会，分子シミュレーション研究会各会員．

© 2014 Information Processing Society of Japan

成見 哲　（正会員）

1970 年生．1998 年東京大学大学院総合文化研究科広域科学専攻博士課程修了．2001 年理化学研究所ゲノム科学総合研究センター研究員．2007 年慶應義塾大学大学院理工学研究科特別研究講師．2009 年電気通信大学准教授．2013 年同大学教授．分子動力学シミュレーション専用計算機の開発，GPU や PS3 による高速化等の研究に従事．学術博士．1996 年日経サイエンス社主催第 2 回 Computer Visualization Contest 最優秀賞，2000 年，2006 年，2009 年 IEEE Gordon Bell Prize 各受賞．人工知能学会，IEEE Computer Society 各会員．

© 2014 Information Processing Society of Japan