Vol.2013-HPC-140 No.31
2013/8/2

GPUクラスタを用いた大規模生体シミュレーションにおけるCPU・GPU間データ転送の効率化

大越 雄一[1]　置田 真生[1]　安部 武志[2]　浅井 義之[2]　北野 宏明[2]　萩原 兼一[1]

概要：Flint は，多様な生体モデルを GPU プログラムに自動変換できる汎用生体シミュレータである．ただし，GPU メモリの容量は高々数 GB であるため，扱えるモデルの規模に制限がある．そこで我々は，より大規模なモデルのシミュレーションのために GPU クラスタを用いる．GPU クラスタでは，ノード間通信のために CPU・GPU 間のデータ転送が必要になる．しかし，データは計算順に従って配置されるため，通信すべきデータはメモリ上に分散している．さらに，CPU・GPU 間データ転送はオーバヘッドが大きいため，データ転送の効率が悪い．提案手法は，あらかじめ通信データをメモリ上の連続領域に配置したプログラムを自動生成する．実験の結果，16 台の GPU クラスタを用いた場合の速度向上率は 2.6 倍であった．

キーワード：生体シミュレーション，自動並列化，CUDA，GPU クラスタ

Efficient data transfer between CPU and GPU for large-scale biophysical simulation on GPU cluster

Ogoshi Yuichi[1]　Okita Masao[1]　Abe Takeshi[2]　Asai Yoshiyuki[2]　Kitano Hiroaki[2]
Hagihara Kenichi[1]

Abstract: *Flint* is a general biophysical simulator, which automatically translates a heterogeneous biophysical model into a simulation code to run on a GPU. The limit of memory space on a GPU bounds the scale of a simulation. We therefore develop an implementation of Flint on a GPU cluster for large-scale biophysical models. With using a GPU cluster, a communication among compute nodes requires data transfer between the node and its own GPU. However, the data to be transferred are ditributed on the memory of the GPU, because existing implementation arranges data according to the order of calculation. The overhead to transfer the distributed data between CPU and GPU degrades the performance of a simulation. Our proposed method can generate a simulation code that arranges the data to be transferred seaquentially on the memory of the GPU. Experimental results show that the simulation code accomplishes 2.6 times speedup using 16 GPUs.

Keywords: Biological simulation, automatic parallelization, CUDA, GPU cluster

1. はじめに

近年，生体機能の解析を目的とした生体シミュレータの研究開発が盛んである．これらのシミュレータは，生体機能の働きを表現した生体モデルについてユーザが指定した時刻における状態を数値計算により求める．生体モデル

[1] 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
[2] 沖縄科学技術研究基盤整備機構 オープンバイオロジーユニット
Open Biology Unit, Okinawa Institute of Science and Technology

は医学分野の発展にともない大規模化し，シミュレーションに要する計算量が増加している．その結果としてシミュレータの実行時間は増大し，計算の高速化が必要となっている．

並列計算によりシミュレーションの高速化を図る生体シミュレータとして，Flint[1]（旧称 *insilico*Sim[2]）がある．Flint は PHML（Physiological Hierarchy Markup Language）[3] を用いて記述された生体モデルを入力とする．PHML は特定の器官に特化しない汎用生体モデル記述言語であり，細胞の電位やイオン濃度のような生体機能を表す物理量を数式で記述する．Flint は，入力モデルに含まれる数式の依存関係を考慮して計算順序を決定し，時間発展型のシミュレーションを実行するプログラムのソースコードを出力する．

Flint が出力できるソースコードは 2 種類あり，それぞれ異なる方法で並列計算を実行する．1 つは OpenMP[4] によってマルチコア CPU を用いて並列計算する CPU 実装であり，もう 1 つは CUDA（Compute Unified Device Architecture）[5] により GPU（Graphics Processing Unit）上で並列計算する GPU 実装 [6], [7] である．CPU 実装については，OpenMP に加えて MPI（Message Passing Interface）[8] も用いることで分散メモリ環境で動作する CPU クラスタ実装も開発されている．

モデルに含まれる数式のうち，依存関係がなく同時に計算できるものが並列計算の対象となる．Flint は，実行時にこれらの数式の計算を自動的に並列化し，計算順序をスケジューリングしてからソースコードを生成する．ユーザは出力されるソースコードをコンパイル・実行することで，モデルに含まれる変数の時間経過による変動を結果として得る．

CUDA は，GPU と呼ばれるグラフィクス処理用のアクセラレータを，汎用計算に応用するための統合開発環境である．CUDA は GPU を，単一命令で複数のスレッドを操作する SIMT（Single Instruction Multiple Thread）型の演算装置として扱う．GPU は多数の演算コアおよび広いメモリ帯域幅を有しており，メモリアクセスがボトルネックとなるような並列計算に適する．以降では，CPU 用のメモリをメインメモリ，GPU が持つメモリをビデオメモリと呼ぶ．Flint は計算ではなくメモリ参照がボトルネックになるため，メモリ帯域幅の広い GPU を利用する実装が有用である．

モデルが十分に大きければ GPU 実装 [6] は CPU 実装よりも高速であるが，ビデオメモリの容量は高々数 GB であるため，扱えるモデルの規模に制限がある．

そこで，複数の GPU にデータを分散させることで 1 台あたりのビデオメモリ使用量を削減できる GPU クラスタ実装を開発する．GPU クラスタ実装は CPU クラスタ実装をもとに開発し，ノード間通信には MPI を用いる．

CPU クラスタ実装および GPU クラスタ実装では，プロセス間で計算結果を共有するための通信が定期的に発生する．特に GPU クラスタ実装は計算結果をビデオメモリに格納するため，通信の前後にメインメモリとの同期を必要とする．このとき発生する CPU・GPU 間データ転送はオーバヘッドが大きく，転送の回数が増加するとシミュレーション全体の性能ボトルネックとなりうる．また，転送の回数を削減しても転送するデータ量が増加すれば効率は低下するため，転送回数と転送量の両方を削減するような方法が不可欠である．

そこで本研究では，全ての通信データをビデオメモリ上の連続領域に配置することで CPU・GPU 間データ転送の転送の効率化を図る．全通信データをまとめて転送することにより，転送を 1 回で完了する．同時に，不要なデータの転送を防ぎ転送に要する時間を削減する．入力モデルのスケジューリング結果から通信データを自動的に抽出し，効率的な通信を行う MPI・CUDA プログラムを生成する．

本論文の構成は以下の通りである．2 章で関連研究を紹介する．3 章で Flint によるシミュレーションの概要を述べる．4 章では GPU クラスタ実装および提案手法について説明し，5 章で評価実験の結果および考察を示す．最後に，6 章で本論文をまとめ，今後の課題を述べる．

2. 関連研究

生体モデル記述言語には，PHML 以外にも SBML（Systems Biology Markup Language）[9] や CellML[10] 等がある．山下らは，ODE のみで構成される CellML モデルからシミュレーションプログラムを自動生成する手法を提案し，実際に C 言語や Java のシミュレーションコードを生成するシステムを実装している [11]．特に LuoRudy1991 モデル [12] については試験的に CUDA 版のコードも生成し，単一 CPU 版より約 50 倍高速なシミュレーションを実現した．ただし，このシステムは単一 GPU を対象としており，ノード間の通信を必要とするクラスタ環境での運用は想定されていない．

生体シミュレーションに限らず，GPU クラスタにおいてノード間をまたがる GPU 同士の通信は CPU を介する必要があり，一般的に CPU・GPU 間のデータ転送によるオーバヘッドがアプリケーションの性能低下につながる．筑波大学計算科学研究センターでは，この問題を解決するために独自の機構 TCA（Tightly Coupled Accelerators）を開発しており，塙らはそのハードウェア実装である PEACH2（PCI Express Adaptive Communication Hub version 2）チップの予備評価について論じている [13]．Flint は特定の環境に依存しない汎用的なシミュレータを目標とするため，現状では TCA のような特殊な機構を前提としない開発を優先する．

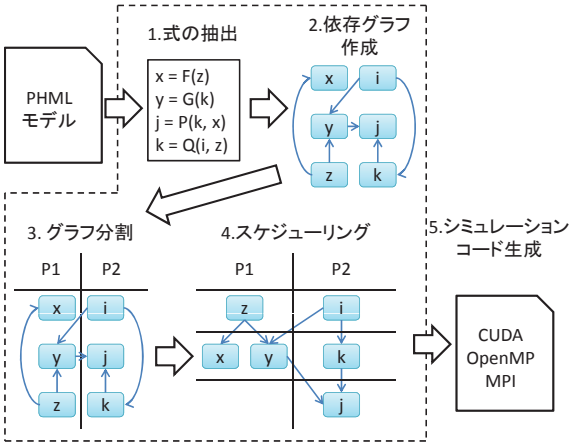


図 1 Flint の概要

3. Flint

既存の CPU クラスタ実装の動作概要を**図 1** に示す．Flint は，入力した PHML モデルから，後述するシミュレーションコードを出力する．PHML は代数関数，常微分方程式（ODE：Ordinary Differential Equation）および遅延微分方程式（DDE：Delay Differential Equation）により生体機能を表現する．Flint はこれらの微分方程式の初期値問題を，オイラー法またはルンゲ=クッタ法を用いて時間発展問題として解く．以降では，代数関数，ODE および DDE をまとめて数式と記述する．

まず，入力された PHML モデルから数式および変数の情報を抽出し，数式を頂点，数式間の依存関係を有向辺とする依存グラフを作成する．以降では，モデルに含まれる変数を PQ（Physical Quantity）と呼ぶ．MPI を用いてシミュレーションを並列実行する場合は，この段階で依存グラフを並列数に合わせて分割する．依存グラフの分割には，外部ライブラリである ParMETIS[14] を用いる．このとき，分割後の部分グラフは各プロセスで計算する数式の集合を，切断辺の本数はプロセス間の通信量を表す．分割において，部分グラフの重みは均等に，切断辺は小数になるよう調節する．これにより，プロセス間の計算負荷を分散するとともに通信量を削減する．

次に，分割後の依存グラフのトポロジカルソートにより数式の計算順序をスケジューリングし，各フェーズで実行する処理を決定する．ここで，フェーズは数式の評価処理の集合または通信処理の集合を表し，計算フェーズおよび通信フェーズを交互に実行することでシミュレーションが進行する．依存関係がなく同時に評価できる数式は同じ計算フェーズに含まれ，並列計算の対象となる．依存グラフの分割時に切断辺が生じている場合，数式の評価に必要な PQ の値を他のプロセスが保持しているため，直前の通信フェーズで最新の値を取得する．

最後に，シミュレーション全体の処理内容をプログラミング言語で記述したシミュレーションコードを生成する．CPU クラスタ実装のシミュレーションコードの概要を**図 2** に示す．シミュレーション中は，PQ 配列 var を用いて PQ の値を管理する．シミュレーション全体は 20 行目から始まるループで表され，シミュレーションの終了時刻 length，ステップ幅 step および計算結果の出力頻度 granularity はユーザが実行時に指定できる．このループは AdvanceStep() で計算を 1 ステップ分進め，時刻 time が length に達するまで step ずつ時刻を進めながら同じパターンの計算および通信を繰り返す．また，時刻 granularity 分の計算が完了するごとに PrintValues() で PQ 配列の内容をファイルに出力する．

```
 1: void calc_phase0(double *vals){
 2:   vals[29] = vals[37] + vals[41];
 3:   ..............
 4: }
 5:
 6: void AdvanceStep(double *vals) {
 7:   calc_phase0(vals); // 数式の評価
 8:   comm_phase1(vals); // 次のフェーズに必要なデータの通信
 9:   calc_phase2(vals); // 通信したデータを用いて数式を評価
10:   comm_phase3(vals);
11:   calc_phase4(vals);
12: }
13:
14: int main(int argc, char *argv[]) {
15:   double time;
16:   double *vals; // PQ 配列
17:
18:   Initialize(argc, argv);
19:   vals = InitVals(); // PQ 配列の初期化
20:   for(time = step; time < length; time += step) {
21:     AdvanceStep(time, vals); // 1 ステップ分の処理
22:     if(time % granularity == 0){
23:       PrintValues(vals); // 結果の出力
24:     }
25:   }
26: }
```

図 2 シミュレーションコードの概要

AdvanceStep() の処理は複数のフェーズに分かれている．計算フェーズ calc_phase$N$() は依存関係のない数式を並列に評価し，通信フェーズ comm_phase$N$() は次の計算フェーズで必要な PQ の値を MPI により通信する．CPU 実装および CPU クラスタ実装では数式の評価を for ループで記述し OpenMP により並列化する．ユーザはこのシミュレーションコードをコンパイルおよび実行することでシミュレーションの結果を得る．

4. GPU クラスタ実装

4.1 概要

GPU 実装および GPU クラスタ実装において，数式の評価は GPU 上で実行する．そのため，計算フェーズはカーネル [5] と呼ばれる GPU 用の関数で記述する．また，GPU

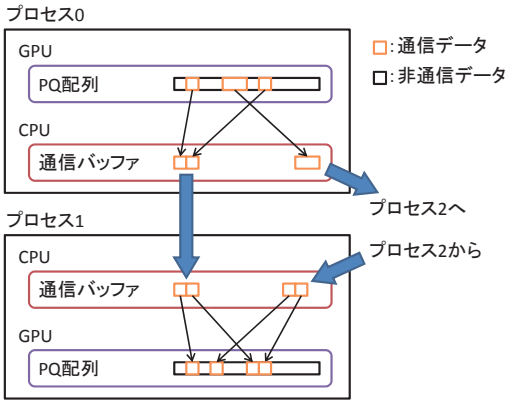


図 3 GPU クラスタ実装における通信

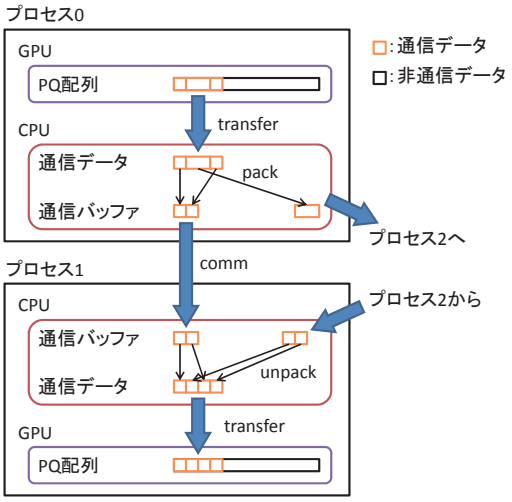


図 4 提案するデータ配置

クラスタ実装では GPU 実装と同様の計算に加えてプロセス間の通信が必要になる．これについては，通信に MPI を用いる CPU クラスタ実装を移植する．

CPU クラスタ実装と異なる点として，GPU クラスタ実装では PQ の値をビデオメモリに保持するため，通信時に CPU・GPU 間で PQ を転送する必要がある．プロセス 0 が送信，プロセス 1 が受信を行う通信フェーズにおける GPU クラスタ実装の動作を**図 3** に示す．

まず，送信側のプロセス 0 は，送信する PQ（送信データ）をビデオメモリからメインメモリへ転送する．次に，メインメモリ上で送信対象のプロセスごとに個別の通信バッファを設け，送信データを格納する．そして，各プロセスへ通信バッファの内容を送信する．最後に，受信側のプロセス 1 は各プロセスから受け取ったデータをビデオメモリへ転送し，PQ 配列の対応する場所へ格納する．

4.2 単純な実装における問題

Flint は，数式の計算速度を重視して PQ の配置を決定し，シミュレーションコードの生成時に PQ 配列に並べる．つまり，実行時のメモリ参照効率を高めるために，スケジュールに沿って数式が参照する順に PQ を配置する．この配置方法において，GPU クラスタ実装では通信データがビデオメモリ上の様々な場所に分散する（図 3）．

通信データの CPU・GPU 間転送には，CUDA の標準関数である cudaMemcpy() を用いる．このとき，cudaMemcpy() の使用方法として以下の 3 つが挙げられる．

(1) 連続領域にある部分ごとに複数回に分けて CPU へ転送する
(2) 全ての PQ を一度に CPU へ転送する
(3) cudaMemcpy() を用いて複製をビデオメモリ上のバッファに集めてから一度に CPU へ転送する

cudaMemcpy() は 1 回あたりのオーバヘッドが大きいため，転送回数が増加する方法（1）は好ましくない．方法（3）も，転送データ量が増加すると cudaMemcpy() の実行回数が増加し，そのオーバヘッドから性能が低下する．また，通信データの容量は全データの 1%にも満たない場合があり，方法（2）では転送するデータの 99%以上が不要になるため効率が悪い．

4.3 提案するデータ配置

提案するデータ配置では，通信時に発生する CPU・GPU 間データ転送の効率を重視する．提案手法においてプロセス 0 が送信側，プロセス 1 が受信側のときの通信の様子を**図 4** に示す．

まず，シミュレーションコードの生成時に，全ての通信データが PQ 配列上で連続するよう並べ替える．PQ は各通信フェーズで使用する順に先頭から並べ，各フェーズごとに送信するもの，受信するものの順に配置する．その後に非通信データを従来通りの方法で並べる．次に，プロセス 0 は通信データのみを一括してビデオメモリからメインメモリへ転送する．これにより，CPU・GPU 間データ転送を一度で完了する．そして，メインメモリ上で通信データを通信バッファへ複製してから各プロセスへ送信する．最後に，プロセス 1 は通信バッファにデータを受け取り，メインメモリ上で別のバッファに複製してからビデオメモリ上の PQ 配列へ転送する．

以降では，ビデオメモリ・メインメモリ間の転送を transfer，プロセス間の通信を comm と呼び，送信時の通信バッファへのデータ複製 pack および受信時の通信バッファからのデータ複製 unpack をまとめて copy と呼ぶ．

5. 評価

まず，提案するデータ配置による CPU・GPU 間の転送時間および計算時間の変化を確認する．次に，GPU クラスタ実装全体の台数効果を評価する．

5.1 実験環境および使用モデル

実験には，16 台の計算ノードからなる GPU クラスタを

表 1　計算ノードの構成

CPU	Intel Xeon Processor X5560 2.80 GHz
メインメモリ	24GB
GPU	Tesla C1060
（ビデオメモリ）	4 GB
CUDA	5.0

表 2　CPU・GPU 間のデータ転送量

分割数	転送量（MB）		削減率
	naive	proposed	（%）
2	92.06	0.28	99.7
4	46.40	0.21	99.6
8	23.43	0.12	99.5
16	12.14	0.07	99.4

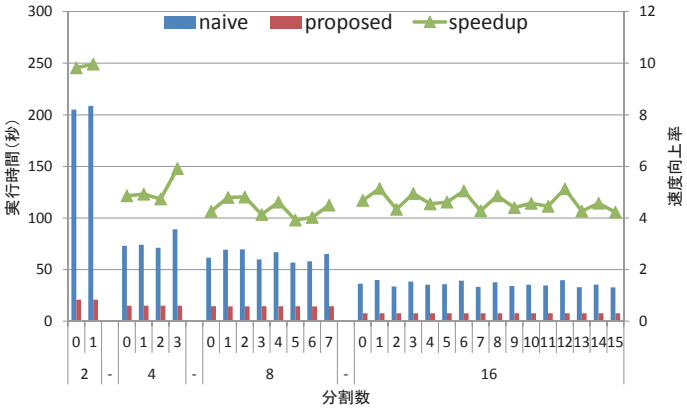


図 5　全体実行時間における単純実装と提案手法の比較

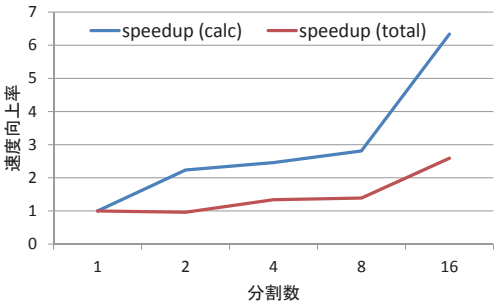


図 7　台数効果

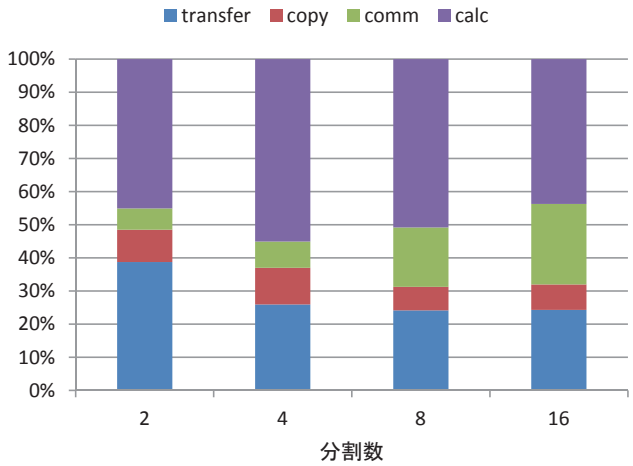


図 8　各処理が全体実行時間に占める割合

使用した．計算ノードの構成を**表 1** に示す．計算ノードは 1 台あたり 1 つの GPU を搭載しており，最大 16 台の GPU を同時に利用できる．また，計算ノード同士は InfiniBand で接続している．

実験には，心室筋細胞における膜電位のダイナミクスを再現する fsk モデルを用いた．fsk モデルは約 12 万のモジュール [3] で構成され，各モジュールは約 460 万次元の ODE を含む．

実験時のパラメータは，length を 10，step を 0.01 に設定し，1000 ステップのシミュレーションを実行した．また，数式の評価および通信前後の処理時間のみを計測するため，GPU・CPU 間の通信を要する計算結果の出力は実行しない．

5.2　データ配置の比較

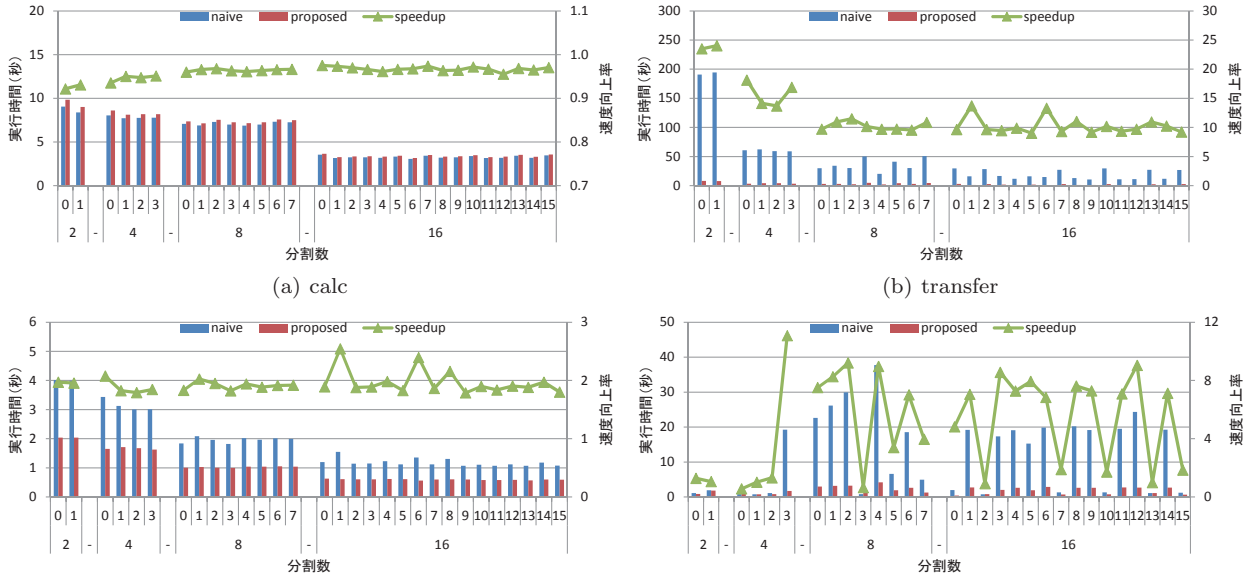
2 種類のデータ配置それぞれを適用した GPU クラスタ実装における，fsk モデルの全体実行時間を**図 5** に示す．naive は計算速度優先の既存手法を GPU クラスタに単純実装したもの，proposed は転送速度優先の提案手法を適用したときの実行時間であり，speedup は naive に対する proposed の速度向上率を表す．naive の転送には，4.2 節で述べた 3 つの方法（1）から（3）のうち，最も実行時間が短かった方法（2）を用いた．なお，計算を分割しないときは通信データが存在せず，2 つのデータ配置に差は生じないため図では省略している．全体として proposed は常に naive よりも高速で，その速度向上率はプロセス数が 2 のとき約 10 倍で最大となる．

実行時間の内訳を**図 6** に示す．transfer，copy，comm は 4.3 節で説明した各処理を，calc は数式の評価を表す．

proposed は，naive と比較して calc が 2.5%から 7.8%遅くなっている．PQ 配列内の PQ は，naive では直近の計算に使用するものが連続しているのに対して，proposed では通信データが連続するよう並んでいる．そのため，naive ではメモリ上の連続領域にあるデータが proposed では離れた位置にあり，メモリ参照の効率が低下していることが原因だと考えられる．

一方で，transfer は 9 倍から 24 倍高速化している．各分割数において，全プロセス中で最大の転送量を**表 2** に示す．naive および proposed で CPU・GPU 間の転送回数は増減しないが，一回当たりの転送量を 99%以上削減している．このことから，提案手法による転送時間の削減を確認できた．

提案手法では，PQ を通信フェーズで利用する順に並べている．各フェーズで通信データを複製する copy は約 2 倍高速化しているが，これはメモリ参照が効率化されたた



(a) calc　(b) transfer　(c) copy　(d) comm

図 6　各処理の実行時間における単純実装と提案手法の比較

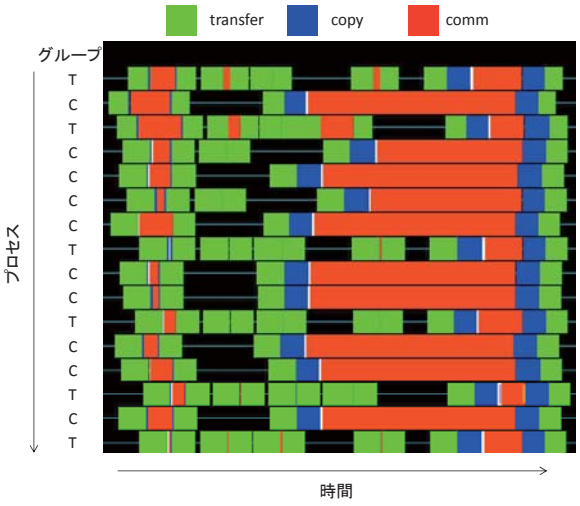


図 9　16 分割実行時の 1 ステップの実行状況

めだと考えられる．

また，comm は最大で 11.1 倍高速化している．後述するように，GPU クラスタ実装において comm のほとんどは他のプロセスからの通信を待つ時間である．transfer および copy が高速化したことで各プロセスが通信を開始する時刻が早まり，待ち時間が減少したことが高速化の要因である．

5.3　スケーラビリティの評価

提案手法において，使用する計算ノード数を変更したときの fsk モデルの速度向上率を**図 7** に示す．speedup (calc) は数式の評価，speedup (total) はシミュレーション全体の速度向上率を表す．速度向上率は calc が最大 6.3 倍，total は最大 2.6 倍となっている．どちらも台数が増加するにつれて単調増加しているが，全体的に total は calc よりも速度向上率が低い．

全体実行時間に占める各処理の割合を**図 8** に示す．値は各分割における全プロセスの平均である．calc 以外に transfer および comm の割合が大きく，これらが全体の性能ボトルネックとなっている．

16 分割の fsk モデルにおいて，1 ステップ中に transfer，copy および comm が実行されるタイミングをガントチャート形式で**図 9** に示す．横軸が時刻を表し，16 プロセス分の情報が縦方向に並んでいる．copy の実行時間は通信データの容量に依存するため，通信前の copy が長い通信は送信するデータの容量が大きく，通信後の copy が長い通信は受信するデータの容量が大きいことを表す．

copy が短い場合でも，その前後に実行する transfer の長さは一定である．これは，そのフェーズで実際に通信するデータ量に関わらず，常に全ての通信データを CPU・GPU 間で転送しているためである．改善のためには，各通信フェーズで必要な PQ のみを転送し，transfer の時間を削減する必要がある．

16 個のプロセスは，通信が 5 回のグループ T および通信が 2 回または 3 回のグループ C に分類できる．transfer は通信のたびに一定時間ずつ生じるため，通信回数が多いグループ T は transfer の合計時間がグループ C よりも長く，通信開始時刻が全体的に遅延する．グループ C の comm のうち実際に通信している時間はわずかだが，この遅延によりグループ T の通信開始を待つ時間が増大する．

6.　まとめ

本研究では，GPU クラスタにおいてノード間通信にともない発生する CPU・GPU 間データ転送の効率化手法を提案した．全ての通信データを自動的にビデオメモリ上の

連続領域に配置し，ノード間通信の際にこれらをまとめてCPU・GPU 間で転送する．これにより，通信データの転送を 1 回で完了するとともに，全体の 99%以上を占める非転送データの転送を防ぎ，データ転送に要する時間の削減を図る．結果，数式の評価は単純実装と比較して 2.5%から 7.8%低速化した．一方で，CPU・GPU 間データ転送は最大 24 倍高速化し，全体として提案手法は単純実装より最大で約 10 倍の高速化を達成した．16 台の GP うクラスタを利用した場合の速度向上率は 2.6 倍と低く，改善が必要である．今後の課題は，CPU・GPU 間の転送時間をさらに削減するために，冗長計算を用いてノード間の通信量を削減することである．

謝辞 本研究の一部は科学研究費補助金（基盤研究（B）23300007，若手研究（B）23700036，新学術領域研究（研究領域提案型）25136711）および文部科学省「医・工・情報連携によるハイブリッド医工学産学連携拠点整備事業」の支援による．本研究に際し，多くの御助言を賜ったサイボウズ株式会社の奥山倫弘氏に深謝する．

参考文献

[1] Asai, Y., Abe, T., Oka, H., Okita, M., Okuyama, T., Hagihara, K., Ghosh, S., Matsuoka, Y., and Kitano, H.: A versatile platform for multilevel modeling of physiological systems: Template/instance framework for large-scale modeling and simulation, *Proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'13)*, Osaka, Japan (2013).

[2] Heien, E., Okita, M., Asai, Y., Nomura, T. and Hagihara, K.: insilicoSim: an Extendable Engine for Parallel Heterogeneous Biophysical Simulations, *Proceedings of the 3rd International Conference on Simulation Tools and Techniques*, Torremolinos, Spain, pp. 78:1–78:10 (2010).

[3] PhysioDesigner.org: About Physiological Hierarchy ML (PHML), PhysioDesigner.org (online), available from ⟨http://physiodesigner.org/phml/index.html⟩ (accessed 2013-6-12).

[4] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. and Menon, R.: *Parallel Programming in OpenMP*, Morgan Kaufmann, San Mateo, CA (2000).

[5] NVIDIA Corporation: CUDA C Programming Guide Version 5.0, NVIDIA Corporation (online), available from ⟨http://docs.nvidia.com/cuda/⟩ (accessed 2013-7-4).

[6] 奥山倫弘, 置田真生, 安部武志, 浅井義之, 野村泰伸, 萩原兼一：インタプリタ型汎用生体シミュレータ insilicoSim の GPU による高速化，情報処理学会研究報告, 2010-HPC-130，Vol. 2011, No. 9, pp. 1–8 (2011).

[7] Okuyama, T., Okita, M., Abe, T., Asai, Y., Nomura, T., Kitano, H., and Hagihara, K.: Accelerating General and Heterogeneous Biophysical Simulations Using the GPU, *Poster in the 4th GPU Technology Conference (GTC 2013)*, San Jose, CA, USA (2013).

[8] Message Passing Interface Forum: MPI Documents, Message Passing Interface Forum (online), available from ⟨http://www.mpi-forum.org/docs/docs.html⟩ (accessed 2013-6-13).

[9] Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., Kitano, H., Arkin, A., Bornstein, B., Bray, D., Cornish-Bowden, A., Cuellar, A., Dronov, S., Gilles, E., Ginkel, M., Gor, V., Goryanin, I., Hedley, W., Hodgman, T., Hofmeyr, J., Hunter, P., Juty, N., Kasberger, J., Kremling, A., Kummer, U., Le Novere, N., Loew, L., Lucio, D., Mendes, P., Minch, E., Mjolsness, E., Nakayama, Y., Nelson, M., Nielsen, P., Sakurada, T., Schaff, J., Shapiro, B., Shimizu, T., Spence, H., Stelling, J., Takahashi, K., Tomita, M., Wagner, J. and Wang, J.: The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics*, Vol. 19, No. 4, pp. 524–531 (2003).

[10] Cuellar, A., Lloyd, C., Nielsen, P., Bullivant, D., Nickerson, D. and Hunter, P.: An overview of CellML 1.1, a biological model description language, *Simulation*, Vol. 79, No. 12, pp. 740–747 (2003).

[11] 山下義陽，副島直樹，川端真成，プンザラン フロレンシオラスティ，嶋吉隆夫，桑原寛明，國枝義敏，天野　晃：形式的に記述された ODE 解法スキームに基づく CellML シミュレーションコード生成システム，生体医工学，Vol. 50, No. 1, pp. 68–77 (2012).

[12] Luo, C. and Rudy, Y.: A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction., *Circulation research*, Vol. 68, No. 6, pp. 1501–1526 (1991).

[13] 塙敏博児, 玉祐悦, 朴泰祐, 佐藤三久：Tightly Coupled Accelerators アーキテクチャに基づく GPU クラスタの構築，先進的計算基盤システムシンポジウム論文集，Vol. 2013, pp. 150–157 (2013).

[14] Karypis, G.: ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering, Karypis Lab (online), available from ⟨http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview⟩ (accessed 2013-6-14).