Vol.2010-HPC-124 No.17
2010/2/23

# GPU クラスタにおける省電力タスクスケジューリング

浜 野 智 明[†1,†2] 額 田 彰[†1,†2]
遠 藤 敏 夫[†1,†2] 松 岡 聡[†1,†2,†3]

GPU を搭載したヘテロ型クラスタシステムの効率的な運用のために，システムのモデリング手法とモデルに基づいたタスクスケジューリング手法を提案する．提案モデルでは複数 CPU(GPU) アプリケーションの競合による性能変化を，アプリケーションの単独実行時間とメモリアクセス量，PCI 通信量から予測する．提案スケジューリング手法はタスクの競合による性能変化を考慮することで，CPU リソースと GPU リソースの両者を有効活用し，システム全体のエネルギー効率の向上を図る．評価はシミュレーションと実測によって行い，タスクの競合による性能変化予測がほぼ正しく行えていることを確認し，また，提案スケジューリング手法により従来手法より高いエネルギー効率を実現できていることを確認した．

# Power-Aware Task Scheduling on GPU Accelerated Clusters

Tomoaki Hamano,[†1,†2] Akira Nukada,[†1,†2]
Toshio Endo [†1,†2] and Satoshi Matsuoka[†1,†2,†3]

We propose a performance model and a task scheduling scheme based on the model to improve the efficiency of GPU accelerated clusters. Our model is aware of performance decline of the applications running on the same node concurrently, and predicts it from elapsed time, amount of memory access and PCI transfer which are measured in stand-alone execution. Our scheduling algorithm aims to reduce not only the makespan but also the energy consumption. In the experiments, we confirmed our model almost predicts the performance decline and our scheduling algorithm improves energy efficiency.

## 1. はじめに

近年，HPC システムにアクセラレータとして GPU(Graphics Processing Unit) を搭載するという手法が関心を集めている．GPU とはグラフィックス処理に特化し設計されたアーキテクチャを持つデバイスであり，大量のデータに単純な命令を繰り返し行うような処理を得意としており，このような処理に関しては汎用 CPU に比べ高いピーク演算性能と電力性能を実現する．実際に GPU を搭載したシステムの例としては，東京工業大学のスパコンである TSUBAME[1] や長崎大学の GPU クラスタ[2]，中国 NUDT の天河スパコンが挙げられる．東工大の TSUBAME は 2006 年に世界のスパコン性能ランキングである Top500[3] において 7 位にランクインし，以降，アクセラレータとして Clear Speed や NVIDIA 社の GPU である Tesla を導入することで，消費電力の増加を最小限に抑えつつ，現在でもスパコンの性能ランキングで国内大学トップレベルの性能を実現している．

一方，HPC システムの運用では現在，ジョブ管理システムとして Condor や OpenPBS,Torque 等が広く用いられている．これらは一般的に，HPC システム全体の性能を事前に集めることで，計算リソースを有効活用するようにジョブをノードまたはコア単位で割り当てていく．また，グリッドのような並列・分散環境においても用いることができるという利点も存在する．しかし，CPU ジョブを CPU リソースに効率よくスケジューリングすることのみを対象としているため，近年登場したような GPU を搭載した新しいシステムにおいてそのまま運用することはできない．

GPU を搭載したシステムの運用において，従来の CPU リソースのみのシステムの運用方法に加え，新たに考慮しなければならない要素として，CPU ジョブと GPU ジョブによるリソースの競合が考えられる．例えば，CPU ジョブは GPU ジョブの host-device メモリ間のデータ転送によるメモリバスなどの競合の影響を受けることが考えられる．実際，我々は実験により CPU ジョブと GPU ジョブの競合により，CPU ジョブが最大 2.5 倍の性能悪化したことを観測した．

そこで，本研究は GPU を搭載したヘテロ型クラスタシステムにおけるタスクスケジュー

†1 東京工業大学
Tokyo Institute of Technology
†2 独立行政法人科学技術振興機構
Japan Science and Technology Agency
†3 国立情報学研究所
National Institute of Informatics



© 2010 Information Processing Society of Japan

リングを対象とする．特に，CPU ジョブと GPU ジョブの競合による性能変化に着目することで，リソースを有効活用しシステム全体のエネルギー効率の向上を目指す．そのため，まずメモリバス等の資源競合を考慮した GPU クラスタのモデリングを行う．次に，スケジューリング長・消費エネルギーを見積もるためタスクスケジューリングのモデリングを行い，そのモデルに基づき CPU リソースと GPU リソースを有効活用することでシステム全体のエネルギー効率を向上を図るタスクスケジューリング手法を提案する．

提案モデルの評価は実機を用い行い，アプリケーションの競合による性能変化をほぼ正しく予測できていることを確認した．また，提案スケジューリング手法の評価はシミュレーションと実測を行い，従来手法より高いエネルギー効率を実現できていることを確認した．

## 2. 関 連 研 究

CPU と GPU によって構成されたヘテロなアーキテクチャ構成のシステムの挙動を探るため，Shuai ら[4] は RodiniaBench というベンチマークプログラムを提案している．RodiniaBench はマルチコア CPU と GPU から構成されるシステムを対象としており，同様の役割を果たすアプリケーションの OpenMP 版と CUDA 版を提供している．含まれるアプリケーションの選択には，カリフォルニアのバークレイ大学が提案した dwarf taxonomy[5] を参考としている．Dwarf taxonomy とは計算のアルゴリズムやデータの動きが似たアプリケーションをひとくくりにし， 全てのアプリケーションを 13 種類に分類するアプリケーションの分類法である．本研究では評価において RodiniaBench を利用している．また，山際ら[6] は GPU を搭載したシステムにおいて，CPU アプリケーションと GPU アプリケーションの競合による性能変化の分析を行っている．本研究では性能変化の分析に加えモデリングを行うことで，性能変化の予測を行う．

ヘテロ環境におけるタスクスケジューリングの研究は 1970 年代から数多く行われているが，特に初期においてはスケジューリング長を対象とし，理論的な最悪性能の解析の研究が多かった．比較的新しい研究としては，Casanova ら[7] はヘテロなグリッド環境において，ファイル I/O の影響を考慮したスケジューリングヒューリスティクスを考案し，シミュレーションで評価している．また Topcuoglu ら[8] は DAG アプリケーションのスケジューリング手法である HEFT と CPOP を提案している．両者とも CPU のみによって構成されたシステムを対象としているのに対し，本研究は近年見られるようになった CPU と GPU を搭載した新しいシステムを対象としている．また，本研究はスケジューリング長の短縮だけではなく，システム全体のエネルギー効率の向上も目的としている点で他研究とは異なる．

また，以前我々はアクセラレータを搭載したシステム上でのタスクスケジューリングについて研究を行い[9]，システムのモデリング手法とスケジューリングアルゴリズムを提案した．しかし，モデル化の際にタスクの競合を考慮できていない等の問題があった．本研究は，システムを詳細に分析することで，より実環境に即したモデリング手法とスケジューリングアルゴリズムを提案する．

## 3. 対象スケジューリング環境の概要

本研究は，汎用 CPU と GPU というヘテロな計算資源を持ったクラスタシステムにおける，オフラインタスクスケジューリングを対象としている．したがって，タスクについての情報（後述）は全て既知であるとする．対象とするスケジューリング環境の概要は図 1 の通りである．

まず対象とするシステムであるが，各ノードには複数 CPU・GPU が搭載されているものとする．そして，それらは各々同一種類のものであるとする．また，各ノードの構成はシステム内であれば全て同一であるとする．

次に対象とするタスクであるが，CPU でのみ実行することができる CPU タスクと，GPU でのみ実行することができる GPU タスクが存在すると仮定する．後者については，一般に，GPU アプリケーションは全ての処理を GPU 上で行うのではなく，複雑な処理やデータの転送などの処理において，CPU リソースを用いている．実際に GPU を搭載した HPC システムである東京工業大学の TSUBAME においては，アクセラレータリソースを用いる際は，それを管理するためのリソースとして CPU コアも 1 コア占有するという資源割り当て方針がとられている．本研究においては，CPU タスクは CPU1 コアを占有し，GPU タスクは GPU だけでなく CPU コアも 1 コア占有するものとする．

## 4. 実 験 環 境

本研究では，以下のマシンをモデル構築の対象及び，実験環境とする．このマシンは AMD Phenom X4 9850 Quad-core を搭載し，HyperTransport(HT)3.0 と PCI-express2.0 × 8 によって 4 台の GPU(NVIDIA GeForce 8800GTS 512) と接続している．マシンの詳細は表 1 の通りである．

また，本研究ではアプリケーションの競合による性能低下モデルにおいて，マシンを図 2 のようにモデル化し，マシンの各部位の帯域の値を用いる．用いた情報は以下である．

- B_core：1 コアが利用できる最大帯域


© 2010 Information Processing Society of Japan

Vol.2010-HPC-124 No.17
2010/2/23

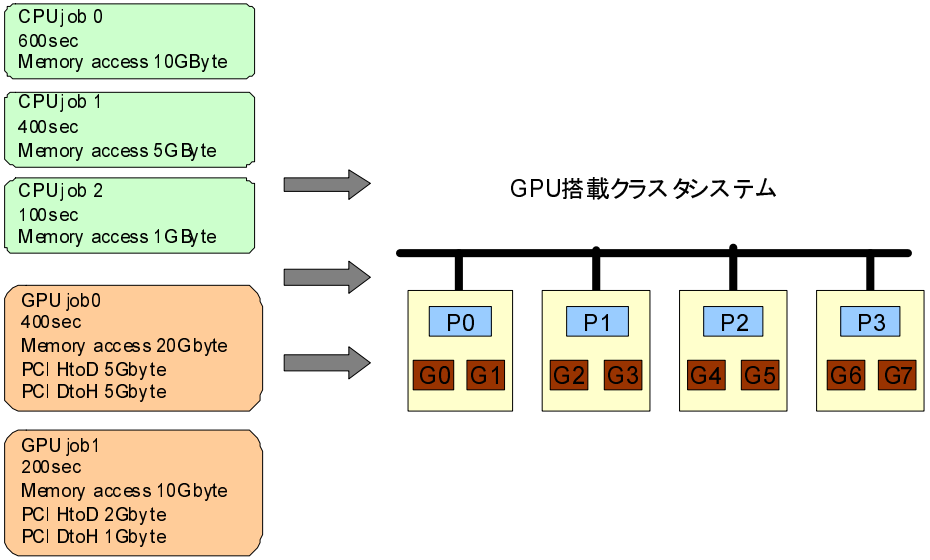


**図 1** 対象スケジューリング環境の概要

- B_cpu：4 コアが合計で利用できる最大帯域
- B_mem：メモリバスの帯域
- B_HtoD(DtoH)：cudaMemcpy のホスト→デバイス (デバイス→ホスト) の帯域

なお，B_core，B_cpu，B_mem は双方向の合計とし，B_HtoD(DtoH) は片方向の値とする．

本実験環境においては，上記の各値は図 2 のようになった．この値は次のように求めた．各コアごとの帯域は Stream Bench を 1 スレッド走らせた時の値であり，コア全体の帯域は 4 スレッド走らせたときの値である．また，メモリバスの帯域は cudaMemcpy の HtoD と DtoH をそれぞれ pinned 領域を用いて 2 スレッド実行したときの合計値である．pinned 領域とは，cudaMemcpy において GPU による DMA 転送が可能な領域のことである．pinned 領域を用いない場合，データは図 2 の矢印のように，データが一端 pinned 領域にコピーされてから PCI 通信が行われる．ここで HT の帯域 (B_HtoD，B_DtoH) であるが，cudaMemcpy の HtoD(DtoH) を pinned 領域を用いないで実行したとき $\frac{B_core}{2} = 2.6$ となることを確認した．これは，pinned 領域を用いない場合 cudaMemcpy の HtoD は図 2 の矢印のようにデータが移動するため，各コアの帯域によって制限されるためであると考えられる．

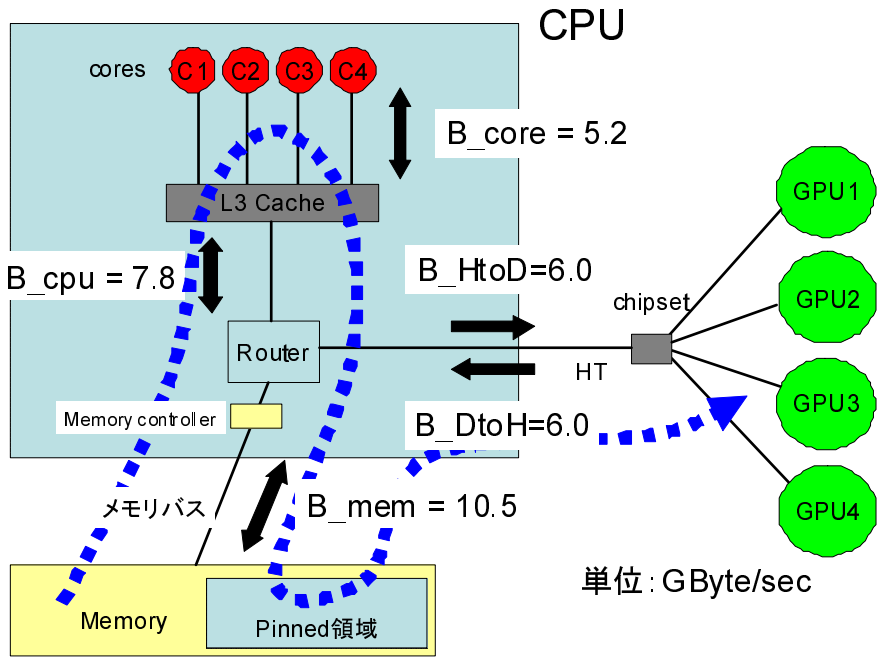


**図 2** 実験環境 (点線矢印：Pinned なし PCI 通信 (HtoD) におけるデータ経路)

## 5. アプリケーションの競合による性能低下の予測手法

### 5.1 予測を行う上での仮定

競合による性能低下の予測を行う上で，図 1 に示すようなアプリケーションを単独で（ノード占有状態で）実行した場合の情報を既知なものとして用いる．



© 2010 Information Processing Society of Japan

**表 1** 実験環境

| | |
|---|---|
| CPU | AMD Phenom X4 9850 Quad-Core |
| System Memory | DDR2-6400 1GB × 4(12.8GB/s) |
| PCI-express bandwidth | 4GB/sec |
| GPU | GeForce 8800GTS 512 × 4 |
| Global Memory Size | 512MB |
| CUDA Version | 2.3 |

- t:実行時間
- m:CPU によるホストメモリアクセス量
- h:ホスト→デバイス PCI 通信量（CPU アプリケーションの場合はゼロ）
- d:デバイス→ホスト PCI 通信量（CPU アプリケーションの場合はゼロ）

また，マシンについては 4 章の情報を用いる．

### 5.2 予測手法

基本的な手法としては，アプリケーションの競合により各通信にかかる時間がどれだけ増加するかを調べることで性能低下を予測する．各通信とは CPU コア⇔ Router，Router ⇔ メインメモリ，Router ⇔ GPU 間の 3 つの通信のことである．以下では，詳細について述べていく．

まず，アプリケーションを S={Memory,HtoD,DtoH,other} と 4 つのステージに分割して考える．Memory とはアプリケーションが CPU からホストメモリにアクセスを行っているステージであり，HtoD はホスト→デバイス通信，DtoH はデバイス→ホスト通信を行っているステージを表す．$T(s)(s \in S)$ をステージ s の実行時間とすると，以下のようになる．

- $T(Memory) + T(HtoD) + T(DtoH) + T(other) = t$

各ステージの実行時間は以下のように求める．

- $T(Memory) = \frac{m}{B_core}$
- $T(HtoD) = \frac{h}{B_HtoD}$
- $T(DtoH) = \frac{d}{B_DtoH}$
- $T(other) = t - T(Memory) - T(HtoD) - T(DtoH)$

次に，フォアグラウンドタスクとバックグラウンドタスクが同時に実行中で，それぞれあるステージのときに，フォアグラウンドタスクが競合によって受ける性能低下の量を見積もる．ここでは説明のため，以下のような状況を考える．

- タスク 0（フォアグラウンド）は Memory のステージ
- タスク 1（バックグラウンド）は HtoD ステージ
- タスク 2（バックグラウンド）は DtoH ステージ
- タスク 3（バックグラウンド）は Memory ステージ

この場合，仮に $B_cpu \geq B_core + 2B_HtoD + 2B_DtoH + B_core$ が成り立てば，CPU コア全体⇔ Router 間で通信の干渉は起こらない．しかし，図 2 のマシンモデルでは，それは成り立たないので，干渉を考える必要がある．ここで，B_HtoD と B_DtoH に 2 を掛けている理由は，cudaMemcpy によって転送されるデータは図 2 の矢印ような経路を通るためである．以下のように各通信帯域を絞る．提案手法では，CPU コア⇔ Router 間の帯域超過による帯域の制限は，各コアについて平等に行うと仮定している．つまりこの場合，帯域を絞ったあとのタスク 0 のメモリアクセス頻度は $\frac{B_cpu}{4}$ となる．次に，同様に Router ⇔メインメモリ間の帯域に超過があるかを調べ，あるのであれば帯域を絞っていく．ここでは，比例関係が保たれるように各帯域を絞るという手法を採用している．以上により，タスク 0 のメモリアクセス頻度が p_core となったとすると，タスク 0 の Memory ステージは $\frac{B_core}{p_core}$ 倍性能悪化したということになる．上記の状況を考える場合はここで操作を終了するが，フォアグラウンドのタスクが HtoD(DtoH) のステージの場合，Router ⇔ GPU 間についても同様の操作を行う．

以上の性能低下の計算をステージの全ての組み合わせについて行う．つまり，バックグラウンドで 3 プロセスが動作していた場合は，(ステージ数)$\hat{}$ { バックグラウンドプロセス数+1}=256 通り行う．以下では，タスク 0（フォアグラウンド）のステージを s0，タスク 1〜タスク 3（バックグラウンド）のステージを s1〜s3 と表し，このとき上記の操作によって求めた s0 の性能低下を $\alpha(s0, \{s1, s2, s3\})$ と表現する．

最後に，競合によるアプリケーション全体への影響を求める．タスク 0〜タスク 3 が各々あるステージにいる時に，タスク 0 が受ける性能低下量を計算し，それを全ての場合の数行い合計すればよい．まず，タスク 0 におけるステージ s0 の時間の割合を $r_0(s0)$ と表すことにする．同様に，タスク 1 におけるステージ s1 の時間の割合を $r_1(s1)$ と表現する．$t_0$ をタスク 0 の単独実行時間とすると，$r_0(s0) = T(s0)/t_0(s0 \in S)$ のように求めることができる．以上を用いると，タスク 0 のタスク 1〜タスク 3 との競合による性能低下は以下の式で求める．これは，単独実行時と比べ何倍性能が悪化したかを表す．

$$\sum_{s0}\sum_{s1}\sum_{s2}\sum_{s3}\alpha(s0, \{s1, s2, s3\})r_0(s0)r_1(s1)r_2(s2)r_3(s3)$$


© 2010 Information Processing Society of Japan

Vol.2010-HPC-124 No.17
2010/2/23

## 6. アプリケーションの競合による性能低下予測モデルの評価

### 6.1 評価方法の概要

4 章で述べたマシン上において，複数アプリケーションを競合実行させ，単独実行させた場合との性能の変化を計測し，予測モデルとの比較を行う．競合実行の基本的な手法としては，アプリケーションをバックグラウンドで複数プロセス繰り返し実行させた状態で，フォアグラウンドで性能低下を求めたいアプリケーションを実行するという手法をとる．ただし，バックグラウンドで実行するプロセスの数は，複数プロセスが同じコアに割り当てられることによる計算性能の競合を回避するために最大 3 プロセスとし，taskset コマンドを用いることで各コアに明示的に割り当てた．

実験に用いたアプリケーションとしては，CPU アプリケーションとして NAS Parallel Benchmark から BT，CG，EP，FT，LU，MG，SP，UA（全てクラス W）．GPU アプリケーションとして RodiniaBench から HotSpot，Kmeans，SRAD，Gaussian．CUDA サンプルコードから alignedTypes，transposeNew．また，行列積の演算を行う Cbmat を用いた．

### 6.2 アプリケーション特性の解析

本実験の前提として，予測を行う上で必要な値の取得のため各アプリケーションについて解析を行った．CPU アプリケーションについては，PAPI(Performance Application Programming Interface)[10] を用い，アプリケーションのメモリアクセス量を取得した．GPU アプリケーションについては，CUDA プロファイラを用い，PCI 通信量を HtoD と DtoH について別々に取得した．また，メモリアクセスサイズは CPU アプリケーションと同様に取得したが，PAPI ではメモリコントローラを通過する全てのデータを取得してしまうため，cudaMemcpy による PCI 通信も含んでしまう．したがって，図 2 のデータ転送経路を考慮し，PAPI で取得した値から，CUDA プロファイラで得られた PCI 通信量の 3 倍を引くことで，CPU からのメモリアクセス量を得た．解析結果は表 2 と表 3 である．

### 6.3 アプリケーションの競合による性能変化

ここでは紙面の都合上，バックグラウンドでアプリケーションを走らせた場合の GPU アプリケーションの性能低下予測についての評価のみを示す．

図 3 は，同じ GPU アプリケーションをバックグラウンドで 3 プロセス連続実行させた状態で，GPU アプリケーションをフォアグラウンドで実行したときの実測による性能低下である．x 軸はバックグラウンドで実行させたアプリケーションの種類を表し，右へ行くほどメモリアクセス頻度及び PCI 通信頻度の小さなアプリケーションとなるようにソートしてある．また，y 軸は単独実行した時と比べ何倍性能低下が起こったかを表す．図 4 は同様の条件における，提案性能低下モデルによる予測性能低下を表す．

**表 2** CPU アプリケーション特性

| | 単独実行時間 (sec) | メモリアクセス量 (GByte) |
|---|---|---|
| BT | 4.87 | 0.3049 |
| CG | 0.95 | 2.8039 |
| EP | 4.14 | 0.0014 |
| FT | 0.51 | 0.6793 |
| LU | 13.70 | 8.6737 |
| MG | 0.66 | 1.9154 |
| SP | 14.13 | 27.4682 |
| UA | 10.43 | 15.3178 |

**表 3** GPU アプリケーション特性

| | 単独実行時間 (sec) | メモリアクセス量 (GByte) | HtoD 通信量 (GByte) | DtoH 通信量 (GByte) |
|---|---|---|---|---|
| alignedTypes | 32.61 | 10.8858 | 0.1666 | 1.9999 |
| gaussian | 16.10 | 0.6292 | 0.0737 | 0.07372 |
| Hotspot | 8.89 | 0.0239 | 0.0018 | 0.000929 |
| Kmeans | 8.44 | 3.6788 | 0.1403 | 0.0731 |
| cbmat | 12.75 | 68.7290 | 11.1977 | 10.4532 |
| SRAD | 15.70 | 27.4051 | 4.5 | 4.5 |
| transposeNew | 30.59 | 10.6715 | 0.0943 | 1.5099 |

図 3 と図 4 を比べると，ほぼ全ての組み合わせにおいて予測モデルによっておおよそ正しく性能低下が予測出来ている事がわかる．これにより，GPU アプリケーション同士の競合による性能低下は，提案モデルによってほぼ正しく把握することができることがわかった．

次に，バックグラウンドで CPU アプリケーションを 3 プロセス連続実行させた状態で，GPU アプリケーションをフォアグラウンドで実行したときの実測性能低下が図 5 である．そして，予測モデルによる性能低下予測が図 6．グラフの見方は図 3 と同様となっている．2 つのグラフを比べると，実測で性能低下がほとんど起こらない部分は予測モデルも正しく予測出来ている事がわかる．しかし，実測で性能低下が大きくなっている部分においては，予測値は実測値よりも最大 20.0 %大きいというずれが生じてしまうことがわかった．しかしながら，予測モデルによって性能低下のおおよその傾向は掴めることができることがわかった．



© 2010 Information Processing Society of Japan

Vol.2010-HPC-124 No.17
2010/2/23

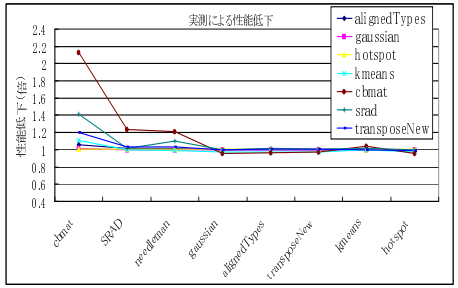


**図 3** 実測による GPU アプリケーションの性能低下（バックグラウンドに GPU アプリ）

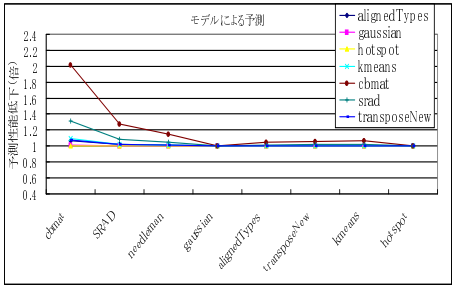


**図 4** 予測モデルによる予測性能低下（左のグラフと同様の条件）

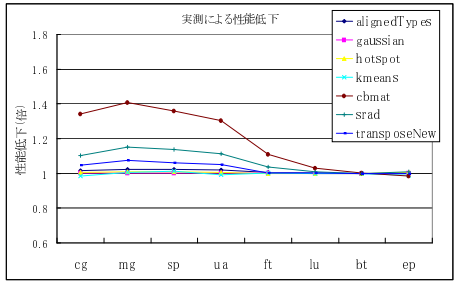


**図 5** 実測による GPU アプリケーションの性能低下（バックグラウンドに CPU アプリ）

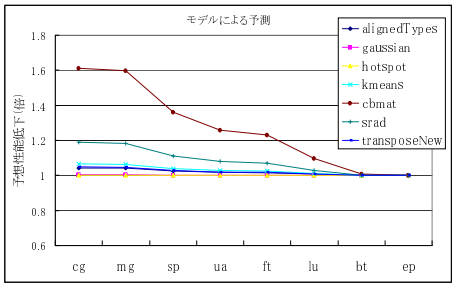


**図 6** 予測モデルによる予測性能低下（左のグラフと同様の条件）

## 7. 提案スケジューリング手法

本提案スケジューリング手法は，タスクの競合による性能変化を考慮することで，CPU と GPU の両リソースの有効活用を図っている．具体的には，本手法は ECT(Earliest Completion Time) をベースとしている．ECT とは，各タスクを，そのタスクの処理を完了する時刻が最も早いプロセッサに割り当てるスケジューリング手法である（図 7）．本提案スケジューリング手法では，タスクを割り当てるプロセッサを決める際，提案性能低下モデルによって競合による性能低下を考慮した上で，タスクを処理を完了する時刻が最も早いプロセッサに割り当てるという手法をとっている．

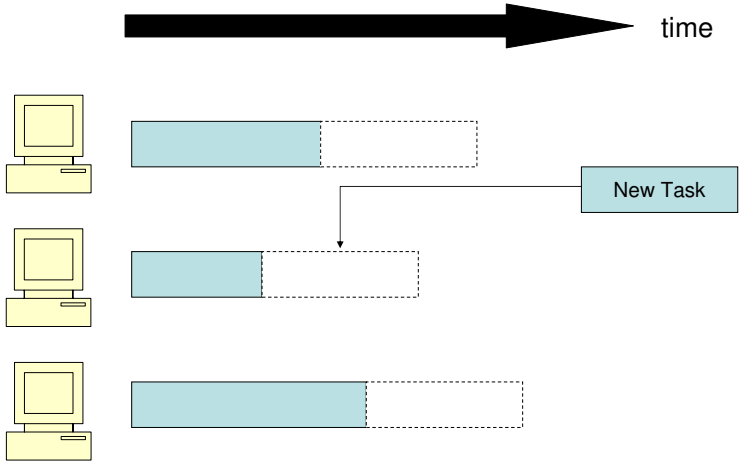


**図 7** ECT によるスケジューリング

## 8. 提案スケジューリング手法の評価

### 8.1 評価方法の概要

提案スケジューリング手法の評価は，まずシミュレーションによって行い，競合による性能低下を考慮していない ECT との比較を行った．そして次に，シミュレーションによって求めたスケジューリング通りに実機でアプリケーションを実行し，シミュレーションと実環境で得られた結果との比較を行った．

### 8.2 評 価 環 境

まず評価に用いるタスクであるが，競合による性能変化予測モデルの評価に用いたアプリケーション群を使用した．したがって，シミュレーションを行う際には，各タスクの性質として表 2 と表 3 を用いた．タスクセット中の各タスクの割合は表 4 のようになっている．



© 2010 Information Processing Society of Japan

**表 4** タスクセット

| | タスクセット 1 | タスクセット 2 | タスクセット 3 |
|---|---|---|---|
| cbmat | 8.33 % | 0 % | 31.25 % |
| その他 11 タスク | 各 8.33 % | 各 9.1 % | 各 6．25 % |
| 合計 | 100 % | 100 % | 100 % |

タスクセットを複数用意した理由は，タスクの偏りによって提案スケジューリング手法が受ける影響を見るためである．タスクセット 1 は全てのタスクの数が等しく，タスクセット 2 は GPU アプリケーションである cbmat を含まないタスクセットとした．また，タスクセット 3 は cbmat を多く含むタスクセットとした．タスクによって cbmat の数を変化させた理由は，cbmat がメモリアクセス頻度及び PCI 通信頻度が他のアプリケーションと比べかなり大きいアプリケーションであるためである．また，タスクセット中のタスクの順序はランダムに決定した．

クラスタシステムのノード数は 10 台とし，各ノードに搭載されている CPU と GPU の数は，CPU4 コア・アクセラレータ 4 つの場合について評価した．

実機で評価する際は，ノード数回スケジューリングを実行することで 1 台のマシンでクラスタシステムを再現した．また，エネルギーの取得はワットチェッカーの積算機能を用いて行った．

### 8.3 評 価 結 果

#### 8.3.1 シミュレーションによる評価結果

図 8 は各タスクセットにおける，提案手法と ECT によるスケジュール長の結果を表している．これから，全てのタスクセットにおいて提案手法により既存手法よりスケジュール長の短縮が実現できた事がわかる．また，タスクセット 3 において提案手法により 3.3 %のスケジュール長短縮という他のタスクセットよりも高い効果が得られた．これは，提案手法はタスクの競合が出来るだけ起こらないようにリソースを有効活用するようなスケジューリング手法であるため，より競合し易いタスクが多いタスクセットほど高い効果が実現できたことを表している．

#### 8.3.2 実測による評価結果

次に，シミュレーションによって得られたスケジューリングを実機に適用し，得られたスケジュール長の結果が図 9 である．提案手法により，実機においても全ての場合において既存手法よりスケジュール長の短縮を実現出来ていることがわかる．しかし，シミュレーション

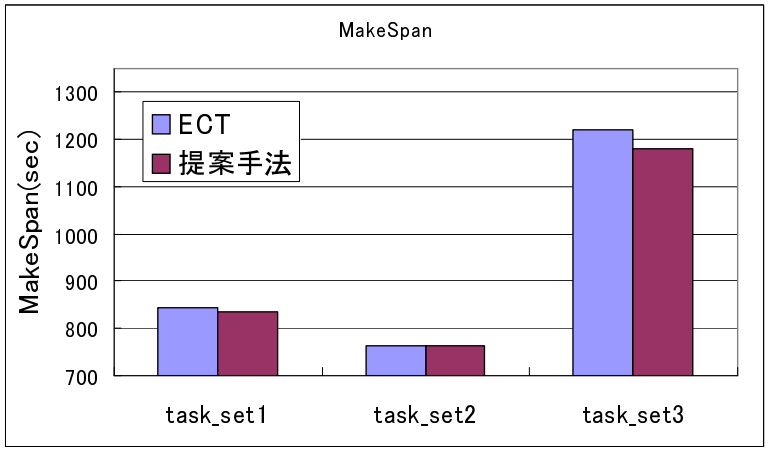


**図 8** シミュレーションによって得られたスケジュール長（原点はゼロとなっていない）

によって得られた値とは誤差が生じてしまっている．これは，提案性能低下モデルの評価の章で述べたように，提案性能低下モデルにも若干の誤差があるため，このような結果になったと考えられる．しかしながら，全てのタスクセットにおいて既存手法よりも良い結果が得られたことから，提案手法の有効性が確認できた．また，各プロセッサの合計実行時間の結果が図 10 である．ここから，提案手法によって最大 2.6 %の競合による影響の削減が実現できたことがわかった．また同時に，タスクセット 3 においては既存手法では 1.77kWh のエネルギーを消費したが，提案手法により 1.74kWh のエネルギー消費となり，1.9 %の消費エネルギー削減も実現できた．しかしながら，タスクセット 1 とタスクセット 2 については，用いたワットチェッカーで取得出来るエネルギーの最小単位に制限があったため，違いは確認できなかった．

## 9. ま と め

GPU を搭載したヘテロ型クラスタシステムの効率的な運用のために，システムのモデリング手法とモデルに基づいたタスクスケジューリング手法を提案した．提案モデルでは複数 CPU(GPU) アプリケーションの競合による性能変化を考慮するため，アプリケーションの単独実行時間とメモリアクセス量，PCI 通信量から性能変化を予測した．アプリケーションの競合による性能変化の予測モデルにより得られた結果と，実測により得られた結果の比


© 2010 Information Processing Society of Japan

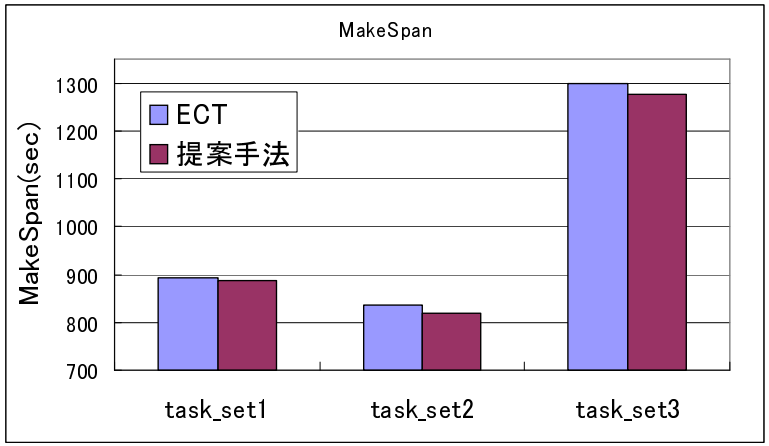


**図 9** 実測によって得られたスケジュール長（原点はゼロとなっていない）

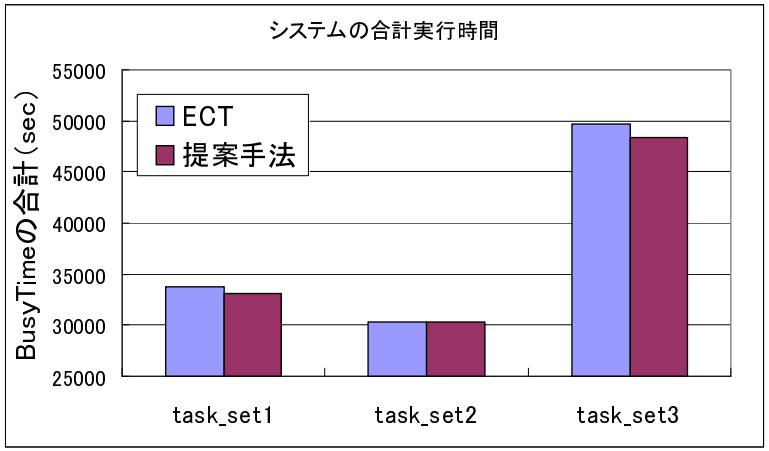


**図 10** 全プロッサの合計実行時間（原点はゼロとなっていない）

較を行ったところ，モデルによってほぼ正しい値が得られていることを確認した．提案スケジューリング手法は，モデルによって見積もったジョブの競合による性能変化を用い，CPU リソースと GPU リソースの両者を有効活用することによって，システム全体のエネルギー効率の向上を図っている．提案手法をシミュレーションと実機によって評価したところ，競合を起こしやすいタスクが多く含まれるタスクセットにおいては，既存手法よりも高いエネルギー効率が得られることを確認した．また，その他のタスクセットにおいても，消費エネルギーに違いは確認できなかったが，提案手法によりスケジュール長の短縮は実現できた．

## 謝　　辞

本研究の一部は，JST-CREST「ULP-HPC:次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」および，科学研究費補助金特定領域研究 (18049028) の補助による

## 参 考 文 献

1) 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 (高性能計算). 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. SIG8(ACS 18), pp. 62–70, 2007.
2) Tsuyoshi Hamada, Rio Yokota, Keigo Nitadori, Tetsu Narumi, Kenji Yasuoka, and Makoto Taiji. 42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence. In *Conference on High Performance Computing Networking, Storage and Analysis*, 2009.
3) TOP500 SUPER COMPUTER SITE. http://www.top500.org/.
4) Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, JeremyW. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization(IISWC), Oct.*, pp. 44–54, 2009.
5) Krste Asanovic, Ras Bodik, and so on. The Landscape of Parallel Computing Research: A View from Berkeley. Technical report, UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec, 2006.
6) Shinichi Yamagiwa and Koichi Wada. Performance Study of Interference on Sharing GPU and CPU Resources with Multiple Applications. In *the 11th Workshop on Advances on Parallel and Distributed Processing Symposium (APDCM09) / IPDPS09*, pp. 1–8, 2009.
7) Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *9th Heterogeneous Computing Workshop*, pp. 349–364, 2000.
8) Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task Scheduling Algorithms for


© 2010 Information Processing Society of Japan

Vol.2010-HPC-124 No.17
2010/2/23

Heterogeneous Processors. In *8th Heterogeneous Computing Workshop*, pp. 3–14, 1999.

9) Tomoaki Hamano, Toshio Endo, and Satoshi Matsuoka. Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In *The Fifth Workshop on High-Performance, Power-Aware Computing (HPPAC)*, pp. 1–8, 2009.

10) Browne. S, Dongarra.JGarner. N, London. K, and Mucci. P. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *Supercomputing, ACM/IEEE 2000 Conference*, pp. 42–42, 2000.



© 2010 Information Processing Society of Japan