Vol.2015-HPC-152 No.2
2015/12/16

CPU・GPU混載ノードにおける電力・性能モデルを用いたオンラインパワーキャッピング手法

都筑 一希[1]　遠藤 敏夫[1]

概要：近年，HPC システムやスーパーコンピュータの大規模化に伴い，限られた電力制約下でシステムを運用していくことが重要となる．特に GPU などのアクセラレータを搭載するシステムにおいては電力変動が大きく，パワーキャッピングがより重要となる．また，パワーキャッピングを CPU と GPU に対する DVFS を用いて行う場合，電力制約を満たす CPU と GPU の組み合わせの中でも性能・エネルギーはそれぞれ異なるため，性能・エネルギー最適な組み合わせを選択したい．そこで，本研究では，ノード電力測定を元に電力・性能モデルを構築することで事前実行なしでノード電力を制約下に抑えつつ，性能・エネルギー最適化を行う電力制御手法を提案する．モデルを使わない単純な制御手法に比べ最大 14.7%の性能向上および 12.6%のエネルギー削減を達成した．

1. はじめに

近年の HPC システムやスーパーコンピュータにおいて，消費電力の問題は大きな課題となっている．例えば，2015 年 11 月現在世界最高性能である Tianhe-2 は，約 34 ペタフロップスの性能が約 18MW の消費電力で実現されている [1]．また，2020 年ごろにはエクサスケールのスパコンの実現を目指しており，それを現実的に可能とされる 20MW 程度の消費電力で実現するには 50GFlops/Watt と，現在の世界トップクラスの技術の 10〜20 倍程度の性能電力比の向上が要求される．また，特に日本においては，東日本大震災に伴う電力危機以降，スパコンの省電力化は強く求められており，このような目標を達成するには，ハードウェアからアプリケーション分野にまたがった技術開発が必要である．

本論文の目的はスパコン上でアプリケーション実行中の消費電力の最大値抑制 (パワーキャッピング) と省エネルギー化の双方を実現することである．そのために主に Dynamic Voltage Frequency Scaling (DVFS) によって電力制御を行う．対象ハードウェアとしては，電力性能比が有利であるとされるアクセラレータを，汎用 CPU に加えて利用するシステムとする．本論文では GPU アクセラレータを持つシステムを対象に実装・評価を行う．DVFS を用いたパワーキャッピングについては，近年の Windows PC などにも機能が搭載されているが，上記のような前提のもとでは，より困難な課題がある．

- 近年の Intel プロセッサでは，RAPL 機能により，CPU とホストメモリ部分のパワーキャッピングを行うことができる．しかしその機能のみでは GPU アクセラレータまで含めた計算機全体のパワーキャッピングを行うことはできない．
- 単純なパワーキャッピングの手法として，電力制約と各時点の実消費電力を比較し，それによって周波数を上下させる方法が考えられる．クライアント PC ではこれで十分な場合が多いが，HPC システム・スパコンで求められるアプリケーション実行中の総エネルギー削減が可能であるかは明らかではない．
- 実行されるソフトウェアの性質の違いによってシステムの消費電力は異なる．特にアクセラレータを持ったシステムではその変動は顕著であり，評価の章で述べるように，ビジー時どうしでも消費電力が 3 倍以上異なる場合がある．そのような前提でパワーキャッピングを行う必要がある．

そこで，本論文では，パワーキャッピングと性能向上，エネルギー削減の両立という目的のために，ノード電力・性能測定をすることによりオンラインで電力・性能モデルを構築することにより事前実行なしで電力制御を行う手法を提案する．現在は NVIDIA GPU アクセラレータを搭載した一台のマシンを対象としている．

本稿の構成は以下のようになっている，２章では研究背景および電力制御の必要性についてより詳細に述べる．３章では提案手法について述べる．４章では評価を行う．５

[1] 東京工業大学
IPSJ, Chiyoda, Tokyo 101–0062, Japan

ⓒ 2015 Information Processing Society of Japan

章では関連研究について紹介し，6章でまとめと今後の課題を述べる．

2. 背景

2.1 電力制御の必要性

消費電力がメガワット級に達するスーパーコンピュータにおける電力制御の必要性は，電力危機への対応や環境保護，電力費用削減のために重要な課題となっている．筆者らも東京工業大学 TSUBAME スーパーコンピュータ上で，ピーク消費電力削減のために，夏季ピークシフト運転を行ってきた [2]．

近年はさらに，次世代のスパコンの事実上の電力性能比を向上させる，「オーバープロビジョニング方式」[3] のためにも電力制御が注目されている．従来のスーパーコンピュータ設計法では，システム全体が稼働した際に消費する理論ピーク消費電力が制約を超えない範囲でハードウェア構成を決定していた．しかし，近年では，まずピーク消費電力が制約を超過することを許容して設計する（オーバープロビジョニング）．そして実際にアプリケーションを実行する際に，そのアプリケーションの特性に応じてシステム稼働時の実際に消費する電力が電力制約値を超えないように制御する（パワーキャップ）．これにより，保守的な設計法よりも高性能なシステムを実現しつつ，実アプリケーションの実行中に電力制約を超えないような運用が可能となる．

以上のような電力制御は，特に電力のダイナミックレンジの大きくなりやすい，アクセラレータ搭載型システムで重要となる．本論文では，そのようなシステムでのパワーキャッピングおよび総エネルギー削減に向けた基礎技術として，CPU・GPU を混載した計算ノードにおけるパワーキャッピング手法に注目する．

2.2 HPC システムにおける電力取得の一般化

従来の HPC システムにおいては，各ノードの電力などを詳細に取得することは困難であった．そのため，アプリケーション毎に異なる消費電力を推定するために，パフォーマンスカウンタ値などを用いた統計的な電力モデルを構築し，アーキテクチャパラメータの決定のときのみ，高性能な電力計を用いるなどの手段が取られた [5]．

しかし近年では，Intel プロセッサの Running Average Power Limit (RAPL) 機能や，NVIDIA 社の NVIDIA Management Livrary (NVML) などにより，HPC システムの全ノードにおいて詳細な電力の取得が容易になっている．またノード電力については標準的なデータ取得方法はないものの，安価なスマートメーターなどの一般化により現実的なものとなっている．たとえば東京工業大学の省電力スパコンプロトタイプである TSUBAME-KFC[4] では，40 ノード全てについて，CPU・GPU だけでなく，各計算ノードの電力を 1 秒間隔で取得することができる．本論文でも，電力監視機能を持ったリモート電源制御装置である OMRON 社 RC3008 を用いている．

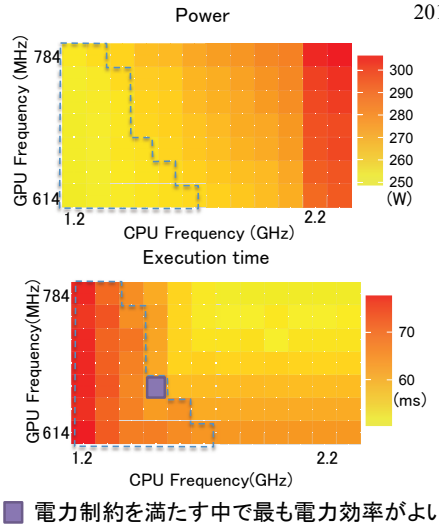

図 1 あるアプリケーションにおける CPU・GPU 周波数に関する電力・性能のヒートマップ

このように近年一般化されている電力取得を用い，本論文では，アプリケーション実行中に電力をリアルタイムに取得し，制約モデルパラメータ決定のために利用する．

2.3 CPU・GPU 同時に電力制御を行う困難さ

また，CPU・GPU 両方の動作周波数を変更させて電力制御を行う際には片方のみの動作周波数を変更させて行う場合に比べより困難になる．図 1 ではあるアプリケーションのそれぞれの CPU・GPU 周波数の組み合わせで実行した際の消費電力と実行時間となっている．このアプリケーションを 260W 以下に抑えようとした場合，青の破線で囲まれた周波数の組み合わせを選択することになる．単に消費電力を 260W の制約を満たしたいだけであれば周波数を下げれば良い．ただ，性能を最適化しようと考えた場合，紫の矩形で示された組み合わせに設定すれば良いが，この組み合わせを見つけるためにはこの性能のヒートマップが必要となる．そのため，我々が提案する手法においては性能・電力モデルを用いることで，電力制御を満たしつつ性能・エネルギー最適化を達成した．

3. 提案手法

提案手法では, CPU, GPU の動作周波数を変更することによりノードの消費電力を制御している．しかし, 単純に現在のノード電力と制約電力を比較し CPU, GPU の動作周波数を変更していく方法では設定までの時間がかかってしまい制約を超過する時間が長くなってしまったり, 性能に関して考慮をしていないため電力制約を満たしたとしても, 性能, エネルギーが最適化されない場合がある．そこで, 提案手法では測定値から消費電力, 性能モデルを構築し, そのモデルを元に CPU, GPU の動作周波数を変更してパワー

ⓒ 2015 Information Processing Society of Japan

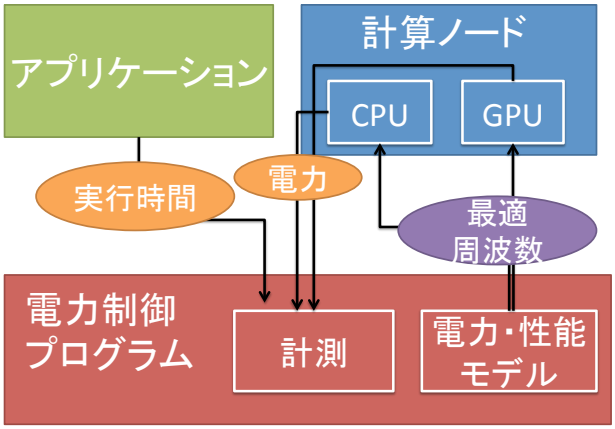

図 2 電力制御のオーバービュー

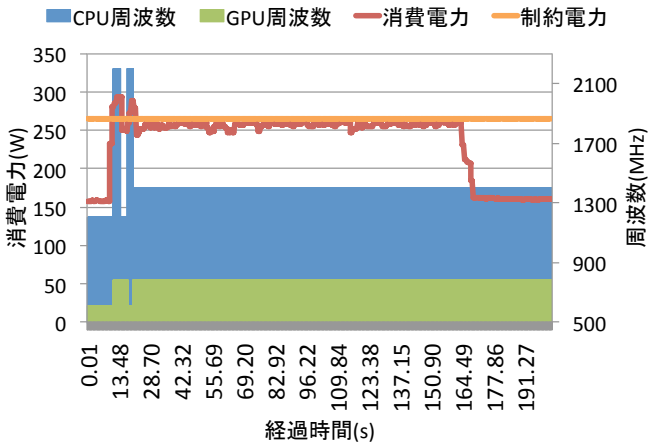

図 3 実際の制御

キャッピングを行う．本章では提案手法の測定方法とモデルについて説明する．また，対象とするアプリケーションは実行中の振る舞いは一定であると仮定している．実際にはアプリケーションのメインとなる演算を行っている部分を繰り返し行うようなものとなっており，その各イテレーションの中での演算の実行時間をファイルに書き込み，それを電力制御プログラムが読み込むことによってアプリケーションの性能データを計測している．将来的には，一定時間あたりのパフォーマンスカウンタなどを取得することにより，対象アプリケーションには何も手を加えない状態で制御を行えるようにすることが目標である．

3.1 電力制御の流れ

電力制御の流れは図 2 のようになっている． まず， アプリケーションが実行されると CPU, GPU の周波数の設定を 3 回変えて電力，性能の測定を行う．これは消費電力，性能モデルのパラメータを求めるために 3 回の測定が必要であるためである． 具体的には CPU 周波数最高と GPU 周波数最高，CPU 周波数最低と GPU 周波数最高，CPU 周波数最高と GPU 周波数最低の 3 回である．その後測定が終わると取得した電力，性能のデータを用いて消費電力，性能モデルのパラメータを計算し，ノード消費電力，性能を推定する．そしてその推定値を元に制約を超えていない CPU，GPU の動作周波数の組み合わせの中から最も実行時間が短いものをエネルギー最適なものとみなし選択し，CPU， GPU の動作周波数をその値に設定する．

3.2 性能モデル

まずは性能モデルを説明する．性能モデルは GPU アプリケーション, CPU アプリケーションを分けてモデルを構築する．ここで f_{CPU},f_{GPU} はそれぞれ CPU，GPU の周波数を表しており，f_{CPU-L},f_{GPU-H} などの表記はそれぞれ CPU の設定可能最低周波数, GPU の設定可能最高周波数を表している．また, T_m は実測した実行時間を表しており，3 回の測定を行うため，実行時間に関しては $T_m(f_{CPU-L}, f_{GPU_H})$, $T_m(f_{CPU-L}, f_{GPU-L})$, $T_m(f_{CPU-H}, f_{GPU-L})$, $T_m(f_{CPU-H}, f_{GPU-H})$ の 3 つの情報を持っていることとなる．式 1 は CPU 周波数に関する実行時間モデル，式 2 は GPU 周波数に関する実行時間モデルとなっている．これらは，実測した実行時間を元に構築される．

$$T_{CPU}(f_{CPU}) = max\ (\ T_m(f_{CPU-L}, f_{GPU-H})\frac{f_{CPU-L}}{f_{CPU}},\ T_m(f_{CPU-H}, f_{GPU-H}))\quad (1)$$

$$T_{GPU}(f_{GPU}) = max\ (\ T_m(f_{CPU-H}, f_{GPU-L})\frac{f_{GPU-L}}{f_{GPU}},\ T_m(f_{CPU-H}, f_{GPU-H}))\quad (2)$$

メモリバウンドなアプリケーションなどはメモリアクセスに律速され周波数を高くしても実行時間が短くならない場合がある．これを再現するために周波数が設定可能最大値の実行時間を実行時間の下限としている．さらにこれら CPU，GPU の実行時間が互いに同期をとり実行時間の長い方に律速される場合を想定し，全体の実行時間は式 3 のようにモデルを構築した．

$$T_{all}(f_{CPU}, f_{GPU}) = max\ (\ T_{CPU}(f_{CPU}),\ T_{GPU}(f_{GPU}))\quad (3)$$

3.3 電力モデル

次に電力モデルについて説明する．電力のモデルは CPU・GPU それぞれにおいて取得できる単位でそれぞれモデルを構築し，それらを足し合わせることによってノード電力を推定している．今回のモデルにおいてディスク，ネットワーク，ファンなど CPU，GPU 以外の電力はベースの電力として CPU・GPU それぞれの周波数に影響されることはないと仮定している．

まずは，GPU の電力モデルについて説明する．GPU の消費電力モデルに関しては式 4 のように表され，アプリケー

ションに関するパラメータ α，β，GPU に関するパラメータ A，B，C によってアプリケーション，GPU ごとの電力を推定することができる．また，f_{GPU} は動作周波数，$V_{GPU}(f)$ は f における GPU の電圧としている．GPU に関するパラメータは事前に測定を行うことによってあらかじめ求めておくこととする．これにより GPU の周波数を設定可能最高値，最低値において測定した電力を用いてアプリケーションに関するパラメータ α，β を求めることができる．

$$\begin{aligned} P_{GPU,app}(f_{GPU}) &= \alpha_{app} A_{GPU} V_{GPU}(f_{GPU})^2 f_{GPU} \\ &+ \beta_{app} B_{GPU} \\ &+ C_{GPU} V_{GPU}(f_{GPU}) \end{aligned} \tag{4}$$

CPU の電力モデルに関して説明する．今回 CPU の電力取得には RAPL を用いているが，RAPL では図に示すように消費電力を観測できる単位が 3 種類定義されており，PKG（チップ全体），PP0（チップ上のコア部分），DRAM（メモリ）となっている．よって取得できる CPU の電力の合計は PP0 と DRAM の和で表されるが，本研究のモデルにおいては PKG-PP0，PP0，DRAM の 3 種類に分け，それぞれに対し式 5，6，7 のようにモデルを構築し，これらの和を CPU 電力の合計とした．

$$P_{PP0}(f_{CPU}) = a_{app1} f_{CPU} V(f_{CPU})^2 + b_{app1} \tag{5}$$

$$P_{PKG-PP0}(f_{CPU}) = a_{app2} f_{CPU} + b_{app2} \tag{6}$$

$$P_{DRAM}(f_{CPU}) = a_{app3} f_{CPU} + b_{app3} \tag{7}$$

$$\begin{aligned} P_{CPU}(f_{GPU}) &= P_{PP0}(f_{CPU}) \\ &+ P_{PKG-PP0}(f_{CPU}) \\ &+ P_{DRAM}(f_{CPU}) \end{aligned} \tag{8}$$

最後に上記の CPU・GPU の電力を足し合わせノード電力とするが，それぞれ AC/DC の変換ロスを考慮し係数として e_{CPU}，e_{GPU} をそれぞれの電力に掛け合わせている．

$$\begin{aligned} P_{node}(f_{CPU}, f_{GPU}) &= e_{CPU} P_{GPU}(f_{GPU}) \\ &+ e_{CPU} P_{CPU}(f_{CPU}) \\ &+ P_{BASE} \end{aligned} \tag{9}$$

4. 評価

4.1 実験方法

ここでは, 提案手法の効果を評価するため，複数の CPU アプリケーション，GPU アプリケーション，CPU と GPU 両方使ったアプリケーションを与えられた電力制約下に制御した際の性能，平均消費電力，消費エネルギー，電力制約を超過した時間について評価した．対象としたアプリケーションは表 1 に示しており，上から 3 つは CPU のみで実行するアプリケーション，真ん中の 3 つは GPU を主に使用したアプリケーション，下の 3 つは CPU と GPU を同時に使用したアプリケーションとなっている．CPU と GPU を同時に使用したアプリケーションは，1 イテレーション内に置いて CPU，GPU でそれぞれ処理を行っており，それらの同期を取った後次のイテレーションへと移行する．つまり，イテレーションの実行時間は CPU，GPU における処理の遅い方の実行時間に律速されるようなアプリケーションを対象とした．

名称	説明
cpuDgemm	dgemm ルーチン
cpuStream	stream ベンチマーク
cpuMatmul	CPU での行列積計算
gpuStream	gpustream ベンチマーク
gpuMatmul	GPU での行列積計算
gpuDgemm	cublas の dgemm ルーチン
cpuMatmul-gpuMatmul	cpuMatmul と gpuMatmul の同時実行
cpuStream-gpuStream	cpuStream と gpuStream の同時実行

表 1 使用したアプリケーション

実験環境としては表 2 に示した構成の計算ノードを用いた．また，ノード電力は OMRON 社の RC3008, GPU 電力取得には NVIDIA 社の nvml(NVIDIA MAnagement Library)，CPU 電力取得には Intel 社の RAPL(Running Average Power Library) を用いた．

	名称	設定可能周波数（MHz）
CPU	Intel Xeon CPU E5-2660	1200-2201 全 1 2 段階
GPU	NVIDIA K20Xm	614,640,666 705,732,758,784

表 2 実験環境

比較する提案手法は表 4 に示したようになっている．提案手法以外の制御手法はモデルを用いずに現在のノード電力のみを考慮して制御をおこなっており，ノード電力が制約を上回っていれば周波数を下げ，ノード電力がある程度制約を下回った場合は周波数を上げる．このとき周波数をどのように変更していくか，また初期周波数が比較手法ごとに異なっている．例えば maxCPU は初期周波数を設定最大値として，ノード電力が制約を超過している場合は CPU 周波数から下げていき，CPU 周波数を下げ切ってもなお制約を満たしていない場合は，GPU の周波数を下げていく．maxLinear，minLinear は CPU・GPU 周波数を同時に調整していくが，その組み合わせは以下のようにおお

ⓒ 2015 Information Processing Society of Japan

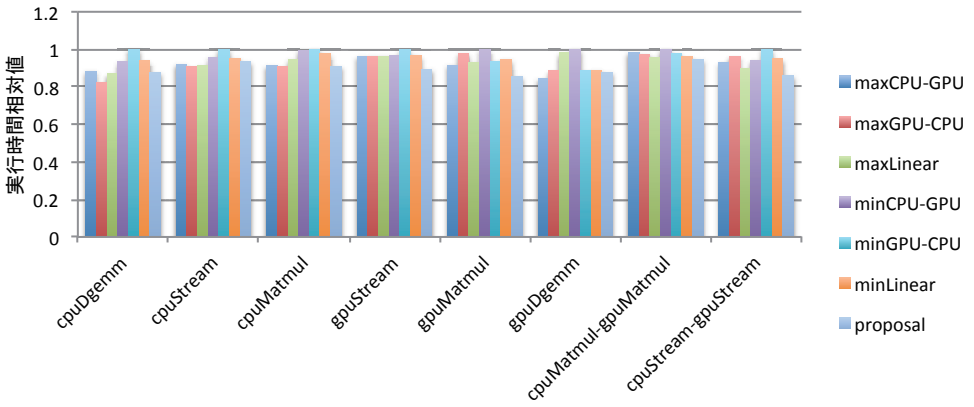

図 4 各アプリケーションにおいて各制御手法で制御した際の実行時間

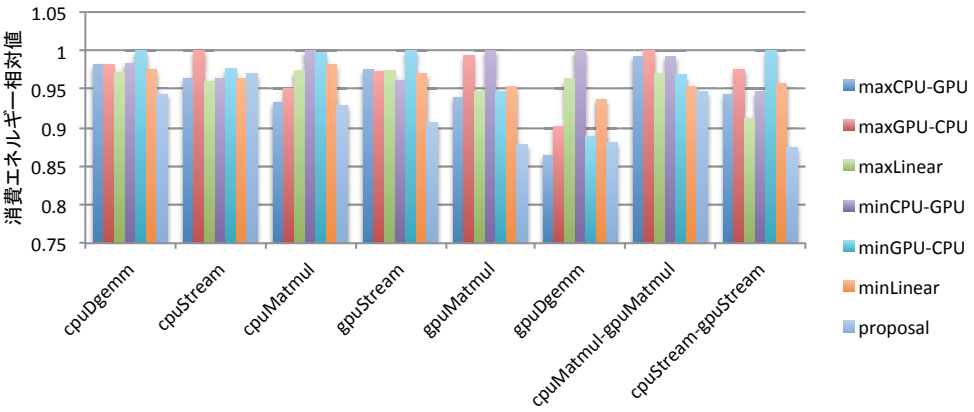

図 5 各アプリケーションにおいて各制御手法で制御した際の消費エネルギー

よそ線形の形で変更していくものとした．アプリケーションごとに消費電力は異なるため，アプリケーションごとに電力制約を設けそれを超過しないよう各制御手法でパワーキャッピングを行った．

名称	制御方式
maxCPU-GPU	初期周波数は最高で CPU-GPU の順で周波数から変更
maxGPU-CPU	初期周波数は最高で GPU-CPU の順で周波数から変更
maxLinear	初期周波数は最高で CPU・GPU 周波数を共に変更
minCPU-GPU	初期周波数は最低で CPU-GPU の順で周波数から変更
minGPU-CPU	初期周波数は最低で GPU-CPU の順で周波数から変更
minLinear	初期周波数は最低で CPU・GPU 周波数を共に変更
proposal	提案手法

表 3 制御手法の説明

minLinear，maxLinear における周波数の組み合わせ (CPU 周波数 (GHz),GPU 周波数 (MHz))
(2.2,784),(2.2,758),(2.1,758),(2.0,732) (1.9,732),(1.8,705),(1.7,705),(1.6,666) (1.5,666),(1.4,640),(1.3,640),(1.2,614)

表 4 制御手法の説明

4.2 実行時間・消費エネルギーに関する比較

図 4 に，各アプリケーションにおいてそれぞれの制御手法で電力制御をした際の実行時間を示す．実行時間はそれぞれのアプリケーションにおいてパワーキャッピングを行った際に実行時間が最大となった制御手法における実行時間を 1 として相対値を用いて評価をする．左 3 つの CPU アプリケーションを見ていくと，全てにおいて maxGPU-CPU が実行時間が最小，minGPU-CPU が最大となっている．これは maxGPU-CPU の場合は初期値が最大周波数でかつ GPU 周波数から設定値を下げていくため，CPU 周波数が最大である時間が比較的長くなるためであり，逆に minGPU-CPU は CPU 周波数が最小である時間が比較的長くなるため実行時間が最大となった．真ん中 3 つの GPU アプリケーションに関しては，上記のような原因で maxCPU-GPU が実行時間最小，minCPU-GPU が実行時間最大となっている．しかし，提案手法が実行時間最大となっているアプリケーションがあるが，これはモデルの誤差で設定周波数を高めに設定してしまったため実行時間は最小となっているが，制約を満たすことはできていない．右 2 つの CPU・GPU アプリケーションの場合，全てのアプリケーションにおいて提案手法による制御での実行時間が最小 となっている．提案手法では各アプリケー

ⓒ 2015 Information Processing Society of Japan

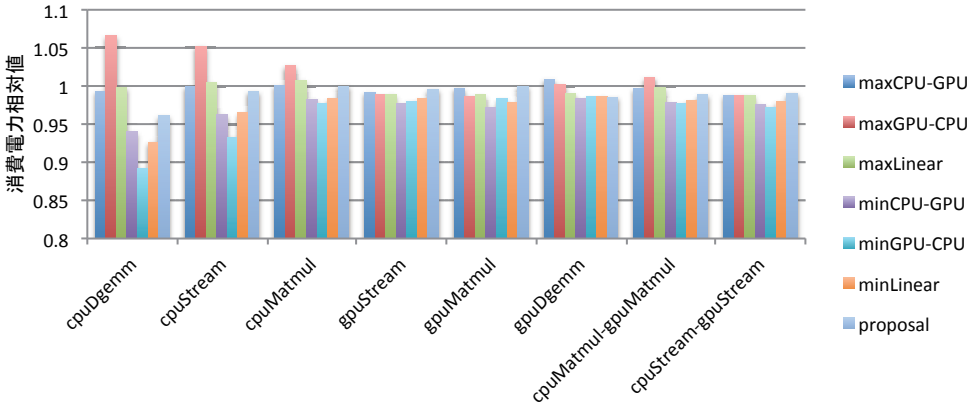

図 6 各アプリケーションにおいて各制御手法で制御した際の平均消費電力

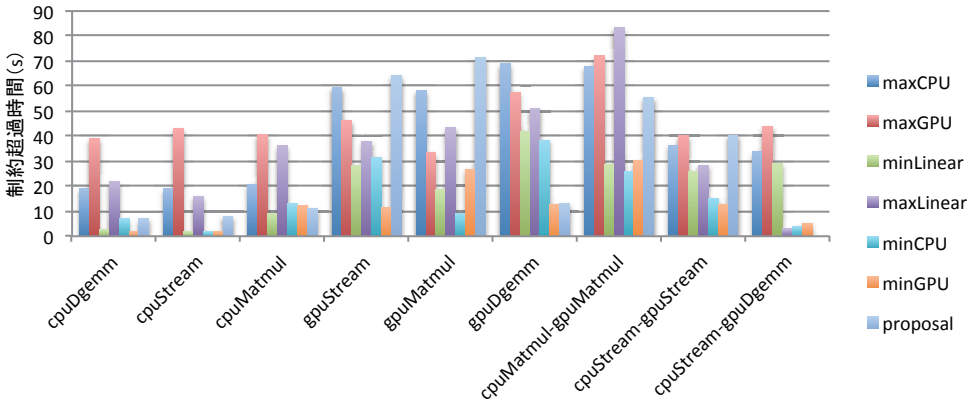

図 7 各アプリケーションにおいて各制御手法で制御した際の制約超過時間

ションにおける実行時間最大となった提案手法と比較して平均 10.7%，最大 14.7%の性能向上を達成した．

次に，アプリケーション実行時の消費エネルギーに関する比較を行う．図 5 に，各アプリケーションにおいてそれぞれの制御手法で電力制御をした際の消費エネルギーを示す．図 6 では各アプリケーションにおいてパワーキャッピングを行った際に消費エネルギーが最大となった制御手法における消費エネルギーを 1 として相対値を用いて評価をする．同一のアプリケーションにおいては同一の電力制約を設けているため，おおよそ実行時間の大小が消費エネルギーの大小と対応している．提案手法では各アプリケーションにおける実行時間最大となった提案手法と比較して平均 8.4%，最大 12.6%の性能向上を達成した．

4.3 平均消費電力に関する比較

次に，平均消費電力に関する比較を行う．図 6 に，各アプリケーションにおいてそれぞれの制御手法で電力制御をした際の平均消費電力を示す．図 6 では各アプリケーションにおける制約電力を 1 として相対値を用いている．従って，1 より小さければノード電力を制約以下に抑えられたということとなる．初期設定値が最大の手法では周波数高い状態が比較的長いため，全体の平均消費電力は制約を超過することが多くなっている．提案手法では全てのアプリケーションにおいて平均消費電力を制約以下に抑えることができている．

4.4 制約超過時間に関する比較

最後に，アプリケーション実行中に制約を超過した時間に関する比較を行う．図 7 に，各アプリケーションにおいてそれぞれの制御手法で制御をした際に制約を超過した時間を示す．提案手法では，測定を行うためアプリケーション実行時の序盤では制約を超過してしまうが，測定はおおよそ 10 数秒程度で終わるため，それ以降超過をしなかった場合は cpuDgemm，cpuStream，cpuMatmul，gpuDgemm のように超過時間も 10 数秒程度となっている．しかし，測定後に設定した値がモデルの誤差により制約を超過するような設定になってしまった場合，現在ではそれを修正する機能が実装されていないため，gpuStream，gpuMatmul のように大幅に制約超過をする時間が増加してしまっている．今後の課題としては，モデルの誤差による制約電力超過を避けるため，モデルの更新などを実装する予定である．

5. 関連研究

従来より，電力制約を行うために，パフォーマンスカ

ウンタ等を用いて電力を推定する技術が研究されてきた．GPU の消費電力に関するモデルを構築し，パフォーマンスカウンタの値からカーネル実効値の消費電力を見積もる手法 [5][6] が提案されてきた．今回用いたモデルでは，パフォーマンスカウンタ値は用いず，取得した電力を元に動作周波数の変更に伴う電力・性能の変化を再現するようなモデルを構築し，静的電力制御のために利用している．

また，GPU 上の各種アプリケーションの性能および電力へ対しての DVFS の影響を調査した研究もある [7][8]．これらの研究は直接には電力制御を行うものではないが，これらの知見を我々のモデルの改善にフィードバックしたい．また 4.1 章で触れた電力取得のタイムラグの問題に関して，NVML におけるタイムラグが Burtscher らにより指摘されている [10]．彼らはさらに NVML によって取得した電力値の推移を基に，より正しいリアルタイム電力値を再現する手法も提案しており，本研究でも今後利用を試みたい．

RAPL を用いてノード全体の電力のモデル化を行う研究もなされている [11]．この研究では，ノード電力を RAPL を用いて推定し，制御を行っている．しかしその手法のみでは，アクセラレータの搭載されたマシンの適切な制御は困難と考えられる．

GPU 搭載マシンのパワーキャッピングのために、DVFS とタスクマッピングの両方を用いる手法 [9] が提案されている．この手法では，アプリケーションがすでに CPU・GPU 間で自由にロードバランスを変更可能な形で記述されている必要がある．これに対し本研究では，任意の GPU アプリケーションを対象とすることを目標としている．

6. まとめと今後の課題

本研究では GPU を搭載した計算ノードにおいてパワーキャッピングを行いつつ，アプリケーションの性能・エネルギーへの影響の軽減を目的とする，そのために，リアルタイムに電力を取得しそれを元にモデル構築し，そのモデルを元に設定周波数を決定するような制御手法を提案した．これにより，モデルを使わない単純な制御手法と比較し最大 14.7%の性能向上および 12.6%のエネルギー削減を達成した．

一方，現在の方式においては，評価の章でも述べた通り一度モデルを構築した後にそのモデルに誤差があった場合にそれを修正する機能がないため，今後モデルを更新していくような改良を行いたい．また，本研究ではアプリケーションは実行中一定の状態であると仮定していたが，電力フェーズが複数あるアプリケーションなど，様々なアプリケーションに対して電力制御ができるような制御方式の構築などが挙げられる．

謝辞

本研究は科学研究費助成事業（基盤研究（S）23220003）と JST-CREST の研究課題「ポストペタスケール時代のメモリ階層の深化に対応するソフトウェア技術」の支援によります．

参考文献

[1] Top500. http://www.top500.org
[2] 遠藤 敏夫, 松岡 聡, 額田 彰, 長坂真路, 四津匡康: グリーンスパコン TSUBAME2.0 における電力危機対応運用．ハイパフォーマンスコンピューティングとアーキテクチャの評価に 関する北海道ワークショップ (HOKKE-19), 情報処理学会研究報告, 2011-ARC-189/HPC-132, 9 pages (2011)
[3] Patki T., Lowenthal D., Rountree B., Schulz M., Supinski B.: Exploring hardware overprovisioning in power-constrained, high performance computing, Proceedings of Proceedings of the 27th international ACM conference on International conference on supercomputing (ICS'13), pp. 173-182 (2013)
[4] 遠藤 敏夫, 額田 彰, 松岡 聡: TSUBAME-KFC: 液浸冷却を用いたウルトラグリーンスパコン研究設備．ハイパフォーマンスコンピューティングとアーキテクチャの評価に 関する北海道ワークショップ (HOKKE), 情報処理学会研究報告, 2013-ARC-199/HPC-142, 6pages (2013)
[5] Nagasaka, H., Maruyama, N., Nukada, A., Endo, T. and Matsuoka, S.: Statistical Power Modeling of GPU Kernels Using Performance Counters, In Proceedings of Interna- tional Green Computing Conference (2010).
[6] Shuaiwen, S., Chun-Yi, S., Barry, R., Cameron and W, K.: A Simplified and Accurate Model of Power- Performance Efficiency on Emergent GPU Architectures., IPDPS, IEEE Computer Society, pp. 673-686 (2013).
[7] Jiao, Y., Lin, H., Balaji, P. and Feng, W.: Power and Performance Characterization of Computational Kernels on the GPU, Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Phisical and Social Computing, IEEE Computer Society, Washington,DC,USA, pp. 221-228 (2010).
[8] Mei, X., Yung, L. S., Zhao, K. and Chu, X.: A Measurement Study of GPU DVFS on Energy Conservation, Proceedings of the Workshop on Power-Aware Computing and Systems, New York, NY, USA, ACM, pp. 10:1-10:5 (2013).
[9] Komoda, T., Hayashi, S., Nakada, T., Miwa, S. and Naka- mura, H.: Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping, iccd13 (2013).
[10] Burtscher, M., Zecena, I. and Zong, Z.: Measuring GPU Power with the K20 Built-in Sensor, Proceeding of Workshop on General Purpose Processing Using GPUs, New York, NY, USA, ACM, pp. 28:28-28:36 (2014).
[11] カオタン, 和田康孝, 近藤正章, 本多弘樹.:RAPL インタフェースを用いた HPC システムの消費電力モデリングと電力評価, 情報処理学会研究報告 2014-HPC-141(20), pp.1-8 (2013).

ⓒ 2015 Information Processing Society of Japan