Vol.2014-HPC-143 No.25
2014/3/4

電力制約下における蓄電池を用いたHPCシステムの性能向上

酒井 崇至[1,a)]　薦田 登志矢[1]　三輪 忍[1]　中村 宏[1]

概要：HPC システムの大規模化に伴い, 電力供給系, 冷却系の建設コストの増大, 運用コストの増大が深刻になっている. こうしたコストを削減する目的で, 電力供給系をオーバープロビジョニングすることで電力供給系の使用効率を高める設計手法が提案されている. このような設計手法を用いるためには, 運用時に計算機資源に電力制約を課す電力管理手法が必要となる. しかし, 従来の電力管理手法を適用した場合, 性能低下が大きく問題があった. そこで本論文では, 蓄電池を用いて時間方向に電力を融通し, 電力制約下における性能向上を実現するパワーシフティング手法を提案する. 提案手法では, 停電時のために設置されている UPS（無停電電源装置）内の蓄電池と周波数制御を併用し, 電力を投入しても性能が上がりにくいフェーズから, 電力を投入することで性能が大きく上がるフェーズへ電力を融通することで高い性能を達成する. 評価の結果, 提案手法を用いることで, 従来の周波数制御を用いた電力制約制御手法に対して, CPU アプリケーションで平均 4.5%, GPU アプリケーションで平均 17.1%の性能向上が実現できることを示した.

キーワード：オーバープロビジョニング, 電力制約, UPS, DVFS

1. はじめに

計算機システムの大規模化に伴い, システム全体の消費電力も急速に増大している. これにより, センタの電力供給系, 冷却系の大規模化による建設コストの増大, および総消費電力増大による運用コストの増大が深刻になっている. 電力供給系の大規模化を防ぐためには, 計算機の電力効率を向上させる手法の研究 [11], [12] とともに, 効率的な電力供給系を設計し運用する手法の確立が重要な課題である.

近年, 多数のサーバを要するデータセンタの分野において, オーバープロビジョニングを用いた電力供給系の設計手法が提案されている [4], [9]. 電力供給系をオーバープロビジョニングするとは, 従来電力供給系の設計に用いられてきたシステムフル稼働時の電力（ピーク電力）ではなく, 実運用時の平均的な消費電力に合わせて, 電力供給系を設計することである. 実運用時の平均的な消費電力は仕様上のピーク電力よりも大幅に低いことが多いため, このような設計手法を用いることで, 不必要に大規模な電力供給系を用意する必要がなくなることが知られている.

電力供給系をオーバープロビジョニングする場合, 実行時における計算機の電力管理が必要である. 処理の実行状況によっては, 瞬間的にピーク電力に近い電力が要求される可能性があり, 電力供給系に過剰な負荷が発生する可能性があるためである. 従来, このような電力超過の可能性を除去するため, プロセッサの周波数やタスク量を調節することで, 計算資源に電力制約を課すパワーキャッピング手法が研究されてきた [4], [9]. しかし, これらの手法は単純に電力制約を守ることを目的として設計されており無視できない性能低下を引き起こすため, 性能制約の厳しい HPC システムに適用する際に問題になっていた.

このような状況の中, パワーキャッピング手法を適用した電力制約下において, UPS（無停電電源装置）内の蓄電池を用いて, 時間方向に電力を融通することで性能を向上するパワーシフティング手法が提案されている [5], [6], [8]. これらの手法では, 停電等非常時用に設置されている UPS 内の蓄電池を用い, 電力要求の少ないフェーズで蓄電池を充電し, 電力要求が電力制約を越えるフェーズで蓄電池を放電することで高いプロセッサ周波数を維持しつつ, 電力制約を守ることが可能になる. しかし, 先行研究での評価では, UPS を用いることが現実的に可能であることの検証に主眼が置かれており, 性能に関しては均一な特性を持つアプリケーションを用いた性能評価に留まっていた.

そこで本論文では, 電力制約下における性能向上の最大化を目的とし, アプリケーション特性を考慮した蓄電池の充放電制御手法を提案する. 蓄電池の容量が有限であるこ

[1] 東京大学 The University of Tokyo, Bunkyo, Tokyo, Japan
[a)] sakai@hal.ipc.i.u-tokyo.ac.jp

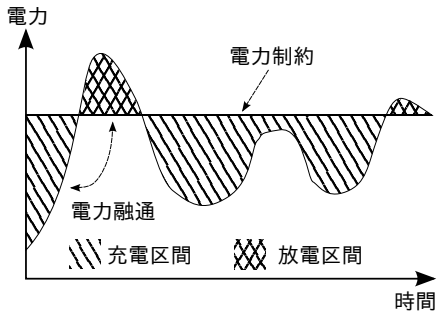


図 1 蓄電池を用いたパワーシフティング手法

とから, 電力制約下において最大の性能を達成するためには, アプリケーションの特性に合わせて充放電タイミングを調整するきめ細かな充放電制御が必要である. HPC アプリケーションは実行時間が長く, 実行が進むにつれて処理の特性が大きく変化するものも多い. これら特性の異なる処理は, それぞれ異なる性能-電力特性を有している. 提案手法では, これらフェーズ間の性能-電力特性の違いを考慮してプロセッサの周波数と蓄電池の充放電量とタイミングを最適化し, 電力制約下における性能向上を達成できる. 電力制約下にある単一ノードを用いた評価実験の結果, 複数のフェーズを有するアプリケーションにおいて, CPU 上で実行されるアプリケーションで平均 4.5%, GPU 上で実行される並列アプリケーションで平均 17.1%の実行時間の短縮がなされるということが分かった.

本論文の構成を示す. 第 2 章で電力供給系と UPS について背景知識を説明する. 第 3 章で提案手法を説明し, 蓄電池の充放電計画を離散最適化問題として定式化する. 第 4 章では, 評価実験により提案手法の有効性を評価する. 第 5 章で関連研究をまとめ, 第 6 章で本稿の結論を述べる.

2. 背景

計算機システムの電力供給系, UPS, および UPS を用いた電力融通手法の先行研究について述べる.

2.1 電力供給系のオーバープロビジョニング

計算機の電力効率を高め, 平均消費電力を削減することが重要である一方, 計算機システムのピーク電力を削減することもまた重要であることが知られている. これは, システムのピーク電力がセンタの電力供給系・冷却系の建設コストに大きく影響するためである. 文献 [7] によれば, これらセンタ施設の建造コストは, センタが 10 年間で償却されるとして一年間に 1W につき 1 ドルから 2.2 ドルである一方, 電力それ自体のコストは一年間に 1W につき 1.2 ドルであり, ピーク電力を削減することが平均消費電力を削減することと同様に重要であることを示している.

このような電力関連コストの構造から, 特にデータセンタ設計・運用の分野において, システムのピーク電力を抑える手法が盛んに研究されている [4], [9], [10]. これらの先行研究では, 供給可能な電力を越える量のピーク電力を持つ計算機資源を運用する, 電力供給系のオーバープロビジョニング手法が提案されている. これにより, 電力供給系の最大供給電力と実運用時における平均消費電力のギャップを埋めることができ, 電力供給系の使用率を改善できる. オーバープロビジョニングを実現するためには, 計算機資源の消費する電力が一定値以下であることを保証するパワーキャッピング手法を適用することが必要となる. 以降, 本論文ではパワーキャッピングが適用された計算機を想定し, 電力制約下において性能を最大化する問題を考える.

2.2 UPS

大規模計算機システムは, 停電等の非常事態における可用性を確保する目的で UPS（無停電電源装置）を備えていることが一般的である. 電力会社からの電力供給が停止した場合, 一時的に UPS が電力供給を行い, その間に自家発電設備が起動する. 数分後, 自家発電設備が完全に起動して電力供給が可能になると, 自家発電設備から電力供給が行われるようになる. UPS 単体がシステム全体に電力を供給し続けられる時間は数分〜30 分程度である場合が多い.

また, 近年のデータセンタでは複数回の A/D 変換による電力ロスを削減する目的で, ラックやサーバごとに UPS が分散して搭載されるようになっている [8]. 本稿では, このような分散型の UPS が搭載されている計算機システムを想定し, これらを用いて計算ノードごとにパワーキャッピングを行う状況を想定している.

2.3 UPS を用いたパワーシフティング

パワーキャッピングに伴う性能低下の問題を解決する目的で, UPS を用いたパワーシフティング手法が提案されている [5], [6], [8]. **図 1** のように, 電力をあまり使用しない時間帯に UPS に含まれる蓄電池に充電を行い, ときおり発生する電力制約を越える高負荷フェーズにおける電力需要を賄うというものである. これにより, 理想的には性能低下を伴わないパワーキャッピングが実現できる. これらの研究では, 蓄電池の電力を使用することによる停電時の信頼性低下や, 充放電頻度に対するバッテリーの寿命低下の影響は, 適切に蓄電池の充放電を制限することで無視でき, UPS を計算機システム内部における電力バッファとして用いることが現実的に可能であることが示されている.

3. 提案手法

アプリケーション中に異なる特性をもつ複数のフェーズがある場合, 従来のプロセッサの周波数制御と蓄電池の充放電制御を組み合わせることで, 電力制約下においてフェーズ間で電力（制約）を融通しあい, 性能を向上させることが可能になる. このとき, 電力制約を守るだけでなく, 全体

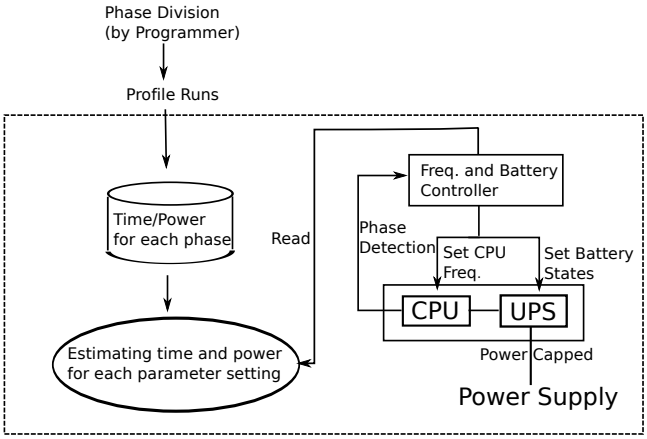


図 2 提案手法の全体像

を通しての充電量が放電量より大きくなければならないなどいくつかの制約を満たしつつ, プロセッサの周波数と蓄電池の充放電を制御しなければならない.

3.1 概要

図 2 に提案手法の全体像を示す. 提案手法では, まずプログラマが後述する指示文を用いてアプリケーションを複数のフェーズに分割する. ここでプログラマにより定義されたフェーズの間で, 蓄電池を用いた電力融通が行われることになる. プログラマが指定しなければならないのは, フェーズがどこで始まりどこで終わるかだけであり, 各フェーズ内でのプロセッサの実行周波数および蓄電池の充放電状況は, 提案システムが自動的に決定する. 次に, フェーズ分割されたアプリケーションをありうる全てのプロセッサ周波数においてテスト実行し, そのときの実行時間および消費電力を各フェーズごとに計測しておく. 提案システムでは, ここで取得したプロファイル情報をもとに, 与えられた電力制約下において最大の性能を達成するようなプロセッサ周波数制御および蓄電池の充放電制御の計画を作成する. 実行時には, フェーズの開始コードにあらかじめ付け加えられたコールバック関数が呼び出され, この中で前述の計画に従ってプロセッサの周波数および蓄電池の充放電状態を設定し, アプリケーション実行が行われる. 提案システムが制御するシステムパラメータおよび, 制御のために用いるプロファイル情報を表にまとめたものを **表 1** に示す.

3.2 フェーズの指定

本手法では, OpenMP のように指示文を用いて, 蓄電池を用いた充放電制御の対象となるアプリケーションフェーズを, プログラマに指定してもらう必要がある. このために, *phase* 指示文を定義している. *phase* 指示文は, *#pragma phase* （*start—end*） *LABEL* という構文により同一の *LABEL* で区切られたコード区間を, 一つのアプリケーションフェーズと見なす. すなわち, 同一のラベルで囲まれたコード区間は, 同一のプロセッサ周波数で実行され, 蓄電池の充電・放電も同様に制御されることになる. この *phase* 指示文は, 同一ラベルに対する *start* と *end* が必ず対になっていなければならない. また, 同一のラベルに対する *start*, *end* の指示文が挿入されたプログラムポイントが必ず対になって実行される保証があれば, 関数呼び出しや条件分岐をまたいで電力フェーズを定義してもかまわない.

制御するパラメータ	粒度
プロセッサの周波数	フェーズごと
蓄電池の充放電状況	フェーズごと
プロファイル情報	
実行時間	フェーズごと, 周波数ごと
プロセッサの消費電力	フェーズごと, 周波数ごと

表 1 実行時に制御するパラメータとプロファイル情報

```
1 #pragma phase start A
2 call () functionA
3 #pragma phase end A
4
5 #pragma phase start B
6 call () functionB
7 #pragma phase end B
```

図 3 電力フェーズを指定する指示文の使用例.

具体的な使用例を**図 3** に示す. この例では, 関数 A の処理と関数 B の処理は別々のフェーズとして定義され, それぞれの関数は異なるプロセッサ周波数で実行される可能性がある. また, 蓄電池の充放電も, 関数 A と B の実行中では異なる可能性があり, 例えば関数 A 実行中に蓄電池を充電して, 関数 B 中の実行中に蓄電池が放電される, といった制御がなされうる. 具体的なプロセッサ周波数の決定, 蓄電池の充放電制御は提案システムがアプリケーションのプロファイル情報をもとに自動的に決定するため, ユーザーがこれらの具体的な設定をする必要はない.

3.3 離散最適化問題を用いた周波数・充放電計画の決定

ここでは, 提案手法がフェーズごとのプロセッサ周波数および蓄電池の充放電状態を決定するために用いる離散最適化問題について説明する.

3.2 節の手法によってアプリケーションが n 個のフェーズに区切られていて, それぞれのフェーズが 1 から n までの番号を一意に割り振られている状況を考える. フェーズ i （ただし $1 \leq i \leq n$） における電力－実行時間曲線を $T_i(p)$ と定義する. $T_i(p)$ はフェーズ i にかける電力 p に対して, フェーズ i を終えるのにかかる実行時間を返す関数である. $T_i(p)$ はフェーズ分割が行われていれば, それぞれのプロセッサ周波数についてテスト実行を行うことで得ることができる. 提案手法は, 与えられた電力制約下 p_{max} においてアプリケーション全体の実行時間を最小化することを目指している. そして本手法ではフェーズごとに蓄電池を用いて電力を融通するため, フェーズ i において蓄電池から供

給される電力を Δp_i とする. Δp_i はマイナスのときは蓄電池に充電することを意味する. 以上の変数を用いて最適化問題として定式化すると, 以下のようになる.

$$\min \quad \sum_{i=1}^{n} T_i(p_{max} + \Delta p_i) \tag{1}$$

$$\text{s.t.} \ \sum_{i=1}^{n} \Delta p_i T_i(p_{max} + \Delta p_i) \leq 0 \tag{2}$$

式（1）は実行時間の最小化を意味する. 式（2）の左辺は, アプリケーション実行の全体を通して, 蓄電池から供給されるエネルギを意味している. 蓄電池はあくまで電力を時間方向に融通しているだけであり, エネルギを増やすことはできない. そのため, 式（2）はエネルギ保存制約式となる. ただし, 定式化で仮定している蓄電池は, 電池容量無限, 充放電速度無限, 充放電によるエネルギ損失がない, という理想的な特性を有する蓄電池である.

現実のプロセッサは有限の数の周波数でしか動作することはできないので, 式（1）, （2）中の Δp_i の組み合わせは有限である. 提案処方では各フェーズに対する可能な全ての Δp_i を代入して全体の実行時間を計算し, 全探索によって最適な $\Delta p_i (1 \leq i \leq n)$ を求める.

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, シミュレーションにより電力制約下において提案手法を適用した場合の性能を評価する. シミュレーションは, 充電容量無限大かつ電力変換ロスのない蓄電池を用いた場合を仮定し, 提案手法を適用していない実機において, 実際に計測される実行時間, 消費電力のプロファイル情報をもとに行った. 具体的には, アプリケーションの各フェーズごと, 取りうる周波数ごとに実行時間と消費電力を測定し, 各フェーズごとの電力-実行時間グラフを取得する. この電力-実行時間グラフをもとに, 提案手法で選択した各フェーズごとの周波数と充放電計画から計算される式（1）中の目的関数の値を, 提案手法を用いた場合の実行時間とした. なお, 実行時間および消費電力のプロファイル情報を取得するために, **表 2** に示した構成の計算ノードを用いた. 実行時間の計測には, Linux の *gettimeofday* 関数を, CPU の電力計測には Intel 社の RAPL ライブラリを, GPU の電力計測には NVIDIA 社の nvml ライブラリを用いている.

評価に用いたアプリケーションを **表 3** に示す. PARSEC [1], SHOC [3] および Rodinia [2] ベンチマークスイートから, 実行時間が長く複数の異なるフェーズを持つ並列アプリケーションとして 4 つのアプリケーションを選んでいる. なお, CPU を用いるアプリケーションでは CPU の電力のみ, GPU を用いるアプリケーションであれば GPU の電力のみを測定し, これらに対する電力制約を変化させた場合に, 提案手法を適用した場合の性能を評価した.

4.2 評価結果

図 4 に, 電力制約下において提案手法を用いた場合, 周波数制御のみを用いて電力制約を課した場合に対する性能向上率を示す. グラフ中に示した電力制約の範囲における性能向上率の算術平均はそれぞれ, *Canneal* で 4.54%, *Stream Cluster* で 0.56%, *mummergpu* で 18.23%, *QTC* で 15.89%である. 提案手法により, 蓄電池を用いたフェーズ間での電力融通を行うことで, 周波数制御だけを用いるパワーキャッピング手法に比べて高い性能を達成するパワーキャッピングが可能になることが分かる. グラフの形を見てみると, 電力制約によって性能向上率が複雑に変化することが分かる. この現象は, プロセッサ周波数を離散的にしか選択できないため, 蓄電池を用いた場合の最適な周波数設定および, 周波数制御のみを用いた場合の周波数設定ともに電力制約の変化に対して, 不連続的に変化するために発生する。

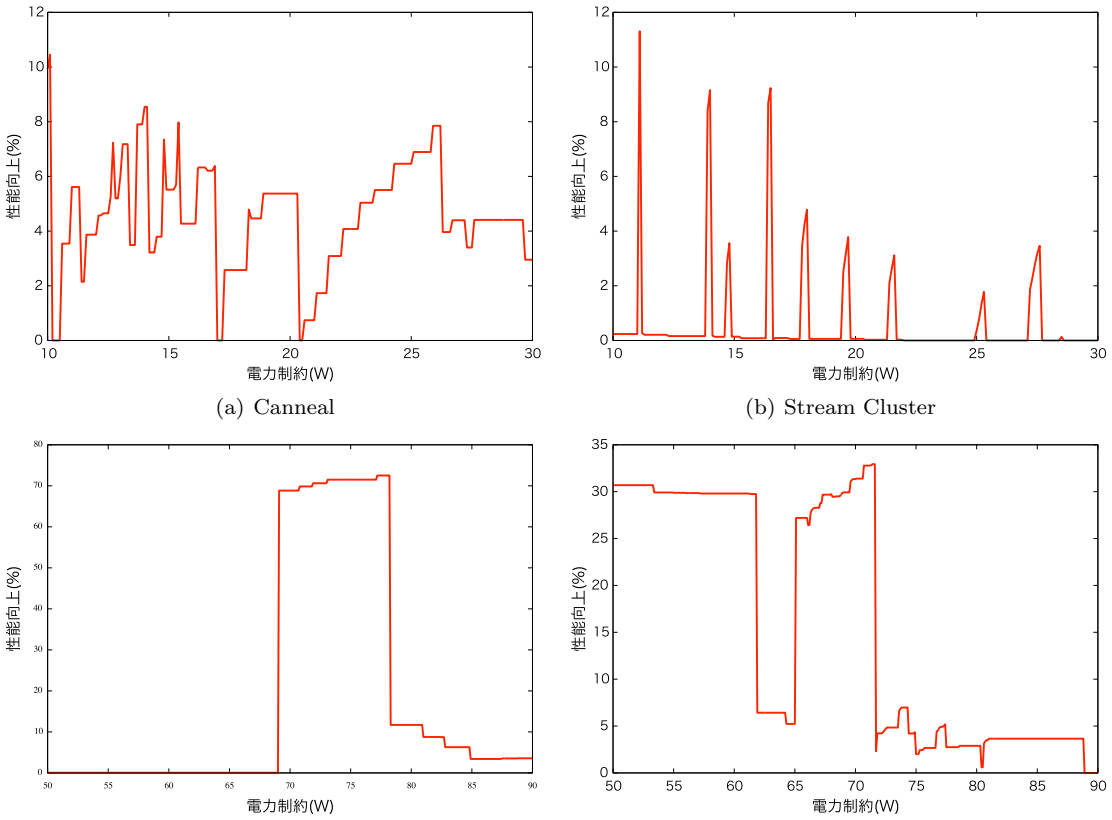
図 5 に, 電力制約を変えたときに, 周波数制御のみを用いた場合および提案手法を適用した場合に各フェーズのプロセッサ周波数がどのように変化したかを示す. 各アプリケーションの各フェーズごと, 各電力制約ごとに選択された周波数および蓄電池の充放電量を詳しく解析すると, 提案手法による性能向上は以下の三つのケースに大別できることが分かる. 一つ目のケースは, 電力を増やしたときに実行時間があまり短縮されないフェーズから実行時間が大きく短縮されるフェーズへと電力融通を行うケースである. 二つ目のケースは, 電力消費が小さく（例えば, 並列度が低く計算ノード内のコアを使い切れない場合）, 電力制約分の電力を使い切れないフェーズで生じた余剰電力を他のフェーズへ融通するケースである. *Canneal* の 20W から 30W ではこの効果が支配的である. 三つ目のケースは, 周波数が離散的であるために生じる余剰電力を融通するケー

プロファイル取得用マシン	名称	周波数 （MHz）
CPU	Intel Core i7 4770 4 コア	800 - 3200 全 15 段階
GPUs	NVIDIA K20c 2496 コア	324, 614, 640 666, 705, 758
MotherBoards	ASRock H87M	–
OS	Ubuntu 13.04	–

表 2 実験機の構成とベンチマーク

ワークロード	プロセッサ	説明
Canneal (PARSEC)	CPU	焼きなまし法
Stream Cluster (PARSEC)	CPU	オンライン クラスタリング
mummergpu (Rodinia)	GPU	DNA 塩基配列の検索
QT Clustering (SHOC)	GPU	精度保証付き クラスタリング

表 3 実験機の構成とベンチマーク



(a) Canneal (b) Stream Cluster

(c) mummergpu (d) QTC

図 4 提案手法による性能向上率: 横軸-電力制約, 縦軸-性能向上率

スである. *Canneal* の 10W から 20W までの電力制約下ではこの効果による性能向上が支配的である. アプリケーションや電力制約によってこれらのケースのどれが性能向上に最も寄与しているかは異なっている.

5. 結論

本研究では, 電力制約下にある計算機システムを想定し, 電力制約下における性能最大化を実現する UPS を用いたパワーシフティング手法を提案した. 提案手法では, ユーザ指示文で分割されたアプリケーションフェーズの間で蓄電池を用いて電力を融通する. このとき, エネルギー保存則や電力制約を満たしつつ, 最大の性能向上を実現するプロセッサの周波数設定および蓄電池の充放電状態を決定しなければならないが, 提案手法ではプロファイルに基づき離散最適化問題を解くことで, 最適な周波数および充放電状態を決定できる. シミュレーションによる評価実験の結果, 複数のフェーズを有するアプリケーションにおいて, 提案手法によって電力制約下における性能向上が達成できることが分かった. 具体的には, CPU 上で実行されるアプリケーションで平均 4.5%, GPU 上で実行される並列アプリケーションで平均 17.1%の実行時間の短縮がなされるということが分かった.

本論文の評価では, 電池容量, 充放電速度が無限大の理想的な蓄電池を想定した評価を行ったが, より現実的な蓄電池を想定したシミュレーション, および実 UPS を用いた評価は今後の課題である. また, 提案手法では充放電計画を決定するために離散最適化問題を全探索を用いて解いている. 電力融通候補のフェーズの数が大きくなった場合でも計算量が爆発しない, 効率の良い求解アルゴリズムを設計することも今後の課題である.

謝辞

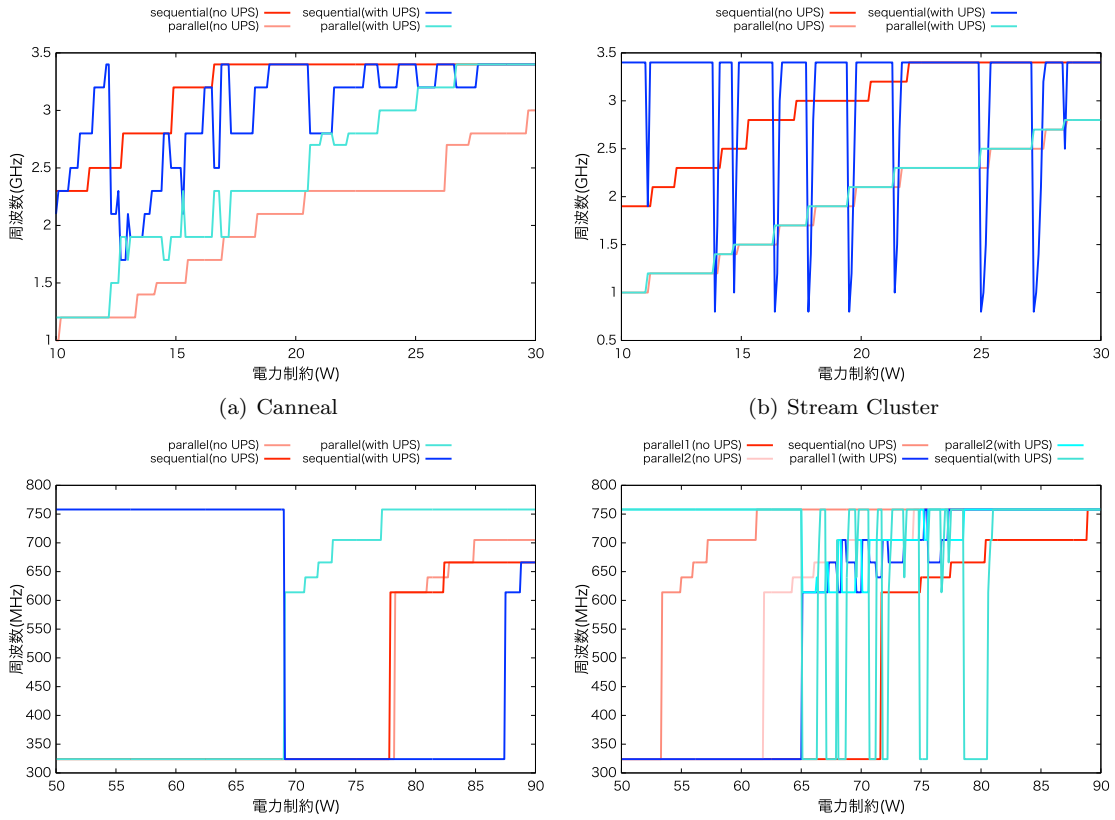
本研究の一部は文部科学省「将来の HPCI システムのあり方の調査研究」事業, JST CREST および科学研究費補助金（課題番号 25540018）による.

参考文献

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, 2008.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 44–54, 2009.

[3] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The



(a) Canneal (b) Stream Cluster

(c) mummergpu (d) QTC

図 5 フェーズごとの電力-実行時間特性: 横軸-電力, 縦軸-実行時間

Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 63–74, 2010.

[4] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News*, 35(2):13–23, June 2007.

[5] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 341–352, 2011.

[6] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar. Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 75–86, 2012.

[7] U. Hoelzle and L. A. Barroso. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[8] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing Distributed Ups Energy for Effective Power Capping in Data Centers. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 488–499, 2012.

[9] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[10] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, 2013.

[11] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, 2009.

[12] J. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili. Keeneland: Bringing Heterogeneous GPU Computing to the Computational Science Community. *Computing in Science Engineering*, 13(5):90–95, Sept.-Oct.