

卒業論文

電力制約下における蓄電池を用いた 高性能計算システムの性能向上

03-120601 酒井 崇至

指導教員 中村 宏 教授

2014 年 2 月

東京大学工学部計数工学科システム情報工学コース

概要

近年、コンピュータの消費電力の増大が大きな問題となっている。そのためコンピュータの性能の指標として単なる実行速度だけではなく、消費電力あたりの実行速度が重要視されるようになってきている。特にスーパーコンピュータのような今日の高性能計算システムでは数メガワットもの電力を消費しており、物理的制約からこれ以上の電力供給力の向上は困難である。このような背景により、予め決められた消費電力の制約下での実行速度の最大化が、今後の高性能計算システムの性能向上の鍵となっている。

そこで、本論文では蓄電池を用いた高性能計算システムの電力制約下での性能向上手法を提案する。現在の高性能計算システムには、停電時にもシステムへの電力供給を続けられるように UPS（無停電電源装置）が搭載されている。それを非停電時にも積極的に充放電を行い、アプリケーションの中の電力をかけても性能が上がりにくい部分から上がりやすい部分へ、電力を時間方向に融通することによって、電力制約下における性能を向上させることができる。本研究では CPU-GPU ハイブリッド構成の計算ノードを用いて、CPU 上での並列アプリケーション及び GPU 上での並列アプリケーションから 2 種類ずつのベンチマークを選び、本手法を用いた場合の性能評価実験を行った。その結果、本手法を上手く適用できるアプリケーションに関しては、本手法を用いない場合に比べて CPU 上での並列アプリケーションでは平均 4.5%、GPU 上での並列アプリケーションでは平均 17.1% の性能向上が実現できることを示し、その有用性を確認した。

目次

第 1 章	序論	1
第 2 章	研究の背景	3
2.1	DVFS	3
2.2	蓄電池を含む電力供給システム	4
2.3	データセンタにおける蓄電池を用いたピーク電力削減手法	6
第 3 章	蓄電池を用いた高速化手法	9
3.1	フェーズ間の電力融通手法の提案	9
3.2	フェーズの要件	12
3.3	フェーズの求め方	12
3.4	電力融通問題の定式化	14
3.5	電力融通問題の解法	15
第 4 章	実験	16
4.1	実験の目的	16
4.2	実験方法	16
4.3	結果	18
4.4	考察	19
第 5 章	結論	29
	謝辞	30
	参考文献	31
	発表文献	34

第 1 章

序論

現代社会においてコンピュータの担う役割はかつてないほど大きくなっており、コンピュータシステムへのますますの高性能化が強く要求されている。より高性能なコンピュータを作るべく、これまで多くの研究者がコンピュータ技術の発展に貢献し、Moore の法則 [1] の示す通りチップの集積度が指数関数的に増すと共にコンピュータの性能も向上し続けてきた。

しかし近年では、消費電力の増大によってコンピュータの性能向上が妨げられてきている。これは一般に、より高速な演算を行うためにはより大きな電力を消費しなければならない、性能向上と共に消費電力も増加し続けてきたためである。その結果として、スーパーコンピュータなどの高性能計算機システムにおいては既に供給できる限界に近い電力を消費しており、それによってコンピュータの性能が制限されている。この様に、与えられた電力制約の中でいかに処理能力を向上させるかが現在の大きな課題となっている。

コンピュータの消費電力を削減するため、DVFS(Dynamic Voltage and Frequency Scaling)と呼ばれる技術が提案され、現在では広く利用されている。DVFS とはプロセッサやメモリの動作速度を、負荷に応じて動的に変動させる技術である。一般に、プロセッサやメモリの周波数を下げると処理能力は下がるが、同時に消費電力も大きく削減される。そのため、性能のクリティカルパス上にないモジュールの動作周波数を下げることにより、システム全体としての性能低下を防ぎつつ消費電力を下げるができる。また、それによって生じた余剰電力をクリティカルパス上のモジュールに融通することで、電力制約下での性能向上を実現できる。

一方で、蓄電池のような電力資源を利用した電力制約下での性能向上手法も考えられる。現在のデータセンタやスーパーコンピュータなどの大規模高性能計算システムにおいては、BCM(事業継続マネジメント)の観点から、地震や火事などの災害による停電時に備えて、自家発電設備や蓄電池が搭載されているケースが多くなってきた。ただし、それらの設備はあくまで緊急時のための予備電源という扱いであり、平常時に使用されることはない。そこで、平常時においては、電力要求が高い場面において蓄電池から電力を供給することができ、電力制約下での性能向上に役立てることができる。この方法をデータセンタに適用する例は文献 [2] に示されている。しかし、スーパーコンピュータなどの高性能計算システムにおいて適用された例は未だ存在しない。データセンタと高性能計算システムでは、動作アプリケーションやハードウェア構成も大きく異なるため、必要となる制御手法も大きく異なると考えられる。

2 第1章 序論

そこで本論文では、高性能計算システムを対象とし、動作周波数と蓄電池の充放電をアプリケーションの動作状況に応じて適切に制御することで、電力制約下で性能を向上させる手法を提案する。本手法では、Tapasya Patki らの研究 [3] の対象となっているような、厳しい電力制約のために全てのモジュールを常に最高動作速度で動作させることはできないようなシステムを対象とする。具体的な手法として、まずアプリケーションのテスト実行時のプロファイルデータからアプリケーションの電力-実行時間曲線の時間推移を予測する。そして、消費電力を減らしても性能が下がりにくい部分を見つけて充電し、逆に消費電力を増やすと大きく性能が上がる部分で放電する。

以降、2 章では本論文に関する技術や研究を紹介し、3 章では解くべき問題の定義と、提案手法の核となる論理を説明する。4 章では 3 章での手法の有用性を確認するための実験方法について説明した後、実験結果を示し、それについての考察を述べる。5 章では結論と今後の課題を述べる。

第 2 章

研究の背景

本章ではまず提案手法の核となる技術である DVFS、及び DVFS を用いた既存の電力削減手法について説明する。そして、対象とする蓄電池を含んだシステムの電力供給システムについて説明した後、蓄電池と DVFS の両方を用いた電力削減手法の関連研究を紹介する。

2.1 DVFS

2.1.1 DVFS とは

プロセッサやメモリを省電力化する最も単純な手法の一つとして、DVFS(Dynamic Voltage and Frequency Scaling) というものがある。これは、プロセッサやメモリの動作周波数を、負荷状況に応じて動的に変更するというものである。ただし、動作周波数が高いほど処理能力も高くなるが、同様に消費電力も大きくなる。かつてプロセッサやメモリは設計時に決められた一つの動作周波数でしか動作することはできなかった。しかし、省電力化の要求が高まってきたため、現在では一つのプロセッサやメモリが複数の動作周波数をサポートしており、負荷状況に応じて適切に動作周波数を選択している。

図 2.1 に、あるサーバプロセッサにおける計算負荷 (Compute load) と消費電力の関係を動作周波数ごとに示す [4]。横軸は計算負荷を、縦軸は消費電力を示している。動作周波数を低くすることにより処理できる最大負荷は下がるが、電力を削減することもできている。つまり、処理できる負荷であれば低い動作周波数の方が消費電力を少なくすることができる。図 2.1 で 1GHz の場合と 2.4GHz の場合を比較すると、動作周波数を下げることにより最大で 20% ほどの電力削減が行えることになる。

2.1.2 DVFS を用いたコンピュータの既存の省電力化手法

これまで、DVFS に関する研究は多く行われてきた。本節ではこれら既存研究や、具体的な DVFS の利用シナリオの例について述べる。

プロセッサに対して DVFS を適用する研究はこれまでに多くなされており、現行のプロセッサにも実装される程普及している。特に、近年では主流となっている 1 チップ上に複数の

4 第2章 研究の背景

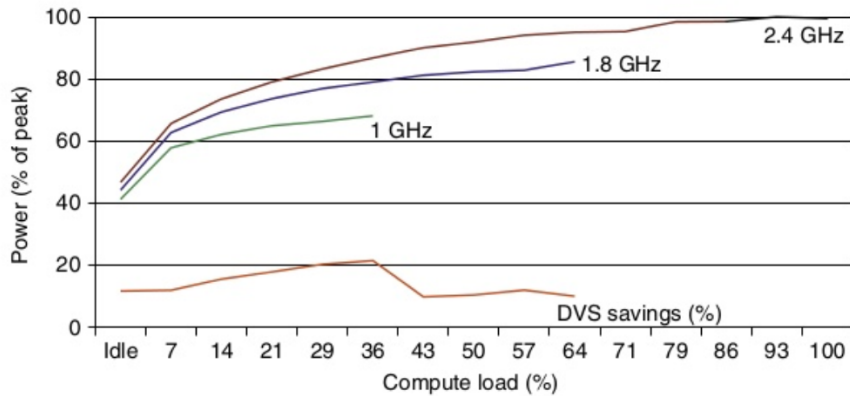


図 2.1. DVFS による電力削減 (AMD Opetron microprocessor) 文献 [4] Figure 1.12 より

コアを搭載した CMP (Chip Multiprocessor) 上では効果的である。例えば、CMP 上で並列アプリケーションを実行した場合、同期待ちによって処理を行わないコアが生じる。そのような状況では、処理を行っているひとつのコアのみを高い周波数で動作させ、その他のプロセッサの動作周波数を落とす手法が有効である。さらに、文献 [5] では、複数プロセッサの組み込みシステムにおいて、ナノ秒単位で DVFS 制御を行うことにより既存の DVFS 制御からさらに 20% もの電力削減が行えるとされている。

また、メモリに対する DVFS についても、近年いくつかの研究がなされている。例えば、文献 [6] では、メモリのバンド幅の使用率を用いてメモリの DVFS 制御を行うことにより、システム全体のエネルギーの 2.4% を削減できることが示されている。さらに、文献 [7] ではプロセッサとメモリの DVFS を同時に用いることによって、それぞれの DVFS を別々に行う場合よりもさらに電力あたりの性能の向上を目指している。この手法では、5 ミリ秒おきにプロセッサとメモリの処理能力の両方を監視して、一方のモジュールの処理能力が足りないときにはそのモジュールに電力を融通することによって処理能力の偏りをなくし、与えられた性能制約を満たしつつ省電力化を行っている。

HPC (High Performance Computing) 領域においても DVFS を用いて、消費電力に対する性能を高めようと研究も多く行われている [8]。HPC では処理能力が高いことが強く求められるため、省電力化を行う場合でも、性能低下ができる限り小さくなるような手法を確立することが強く要求されている。

2.2 蓄電池を含む電力供給システム

スーパーコンピュータやデータセンタなどの大規模高性能計算システムにおいては、高い信頼性や可用性が要求されるため、一瞬たりとも電圧低下や電力供給停止は許されない。そのため、停電や機器の故障によって電力会社からの電力供給が受けられない時にも、コンピュータへの電力供給を継続するためにいくつかの冗長電源設備が用意されている。図 2.2 は本論文で

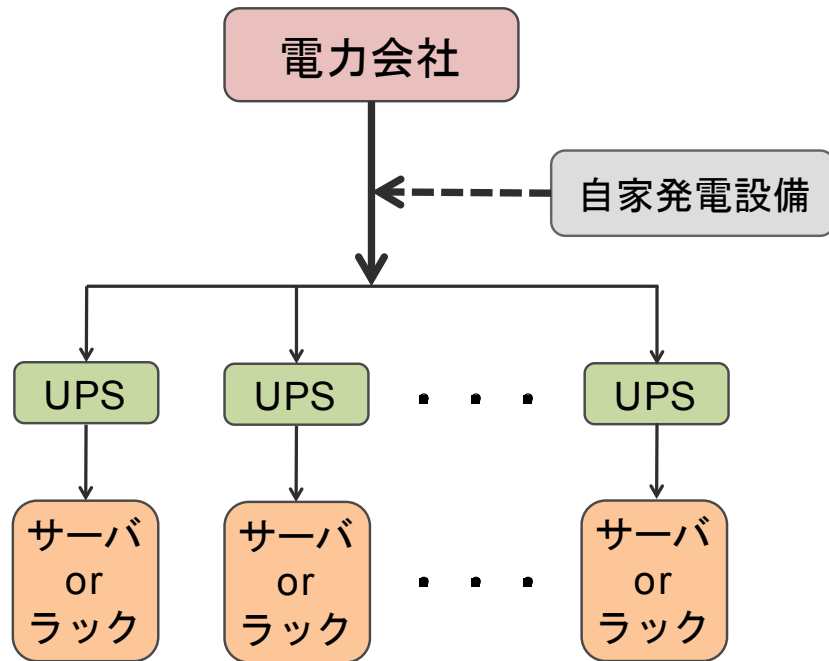


図 2.2. 本論文で対象とする高性能計算システムの電源設備

対象とする高性能計算システムの電源設備である。

一般的な高性能計算システムでは、システム全体にひとつの UPS(無停電電源装置) が搭載されているが、Google や Facebook のデータセンタにおいては図 2.2 一つのラックやサーバごとに UPS が搭載されている [9]。これは電力架線からサーバまでの電力供給の間に行われる D/A 変換の回数を減らして、全体での電力変換効率を向上させるためである。このように、ラックごとやサーバごとなどの細かい単位での UPS を配置する方法は、電力効率を重視する上ではこれからのスタンダードになっていくと考えられている。

UPS を搭載したデータセンタにおいて電力会社からの電力供給が停止した場合には、UPS が電力供給を行い、同時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。電力会社からの電力供給が再開すると自家発電設備は停止し、電力会社からの電力を使用するようになる。

ここで UPS は 3 つの役割を担っている。一つ目は、コンピュータへの供給電圧を安定させること。二つ目は、停電時に自家発電設備からの電力供給が始まるまでの間、電力を供給すること。三つ目は、停電復帰後に自家発電設備から電力会社に電力供給元を切り替えるとき、一時的に電力供給を行うことである。

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。現在の多くの UPS では電源として蓄電池が使用されているが、充放電が行われるのは基本的に停電時のみであり、今のところ平常時に積極的に充放電を行うような使い方はなされていない。

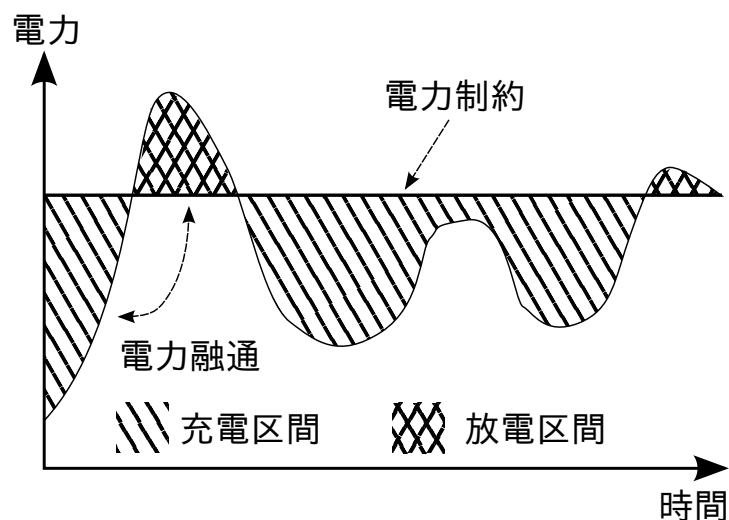


図 2.3. 参考文献 [2] における蓄電池を用いた電力ピークカット手法

2.3 データセンタにおける蓄電池を用いたピーク電力削減手法

前 2.2 節で述べたように、今までは平常時に積極的に UPS の蓄電池から充放電を行うことはなかったが、2011 年に発表された論文 [2] において、UPS からの充放電を用いたデータセンタの電力ピークカット手法が提案された。本稿の提案手法と大きく関わる内容であるので、ここで詳しく紹介する。

データセンタにおいてはコンピュータでの消費電力や冷却にかかる電力コストは全体の運用コストの 10～30% に上り、サービス向上のために電力コストの削減が必要とされている。データセンタを建設するときの初期投資、及び電力会社との契約料金はピーク時の電力に大きく影響される。そのためピーク電力を削減すべく、この論文では UPS 中の蓄電池を用いた電力ピークカット手法を提案している。

データセンタの 1 日の電力需要の推移は、統計や過去の研究によってある程度予測ができるようになっている。その電力需要曲線から最適な蓄電池の充放電計画を立て、電力会社から引き込む電力の最大値を低く抑えることがこの紹介論文の主旨である (図 2.3)。

紹介論文において解くべき対象としている問題を言葉で表現すると以下のようにまとめられる。

- 目的
 - minimize (一日の最大消費電力)
- 与えられる情報
 - 一日の電力推移グラフ
- 制御対象

- バッテリーをいつ、どれだけ充放電するか
- 制約条件
 - 一日の放電時間・回数
 - バッテリーの残量

この紹介論文の研究以前にも、電力会社からのピーク電力を削減するためにプロセッサの動作速度を変更する手法 [10, 11, 12, 13, 14] や、負荷を時間的もしくは空間的に分散させる手法 [15, 16] が提案されてきた。しかし、これらの手法を適用すると処理速度の低下が必ず起こってしまうことが問題であった。紹介論文における提案手法は、UPS に含まれるバッテリーという既存設備を用いることで、この性能低下を起こさずに電力ピークカットを実現できることを示している。

この手法で実際に用いられているアルゴリズムは図で表現すると図 2.4 のようになり、言葉で表現すると以下ようになる。

1. 一番高いピークが、二番目に高いピークと同じ高さになるように放電を計画（制約条件を満たせば次のステップへ）
2. 二番目のピークより高いピーク全てが、三番目に高いピークと同じ高さになるように放電を計画（制約条件を満たせば次のステップへ）
3. . . . (制約条件を満たさなくなるまで繰り返し)

この紹介論文は他にもバッテリー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も考慮に入れて充放電計画を立てることによって、データセンタの事業継続性を保ちつつ電力コストを削減できるとしている。

この紹介論文では性能制約を守った上でどれだけ省電力化を行えるのかという問題であった。一方で、決められた電力制約を超えないように制御を行う Power Capping という手法についての先行研究も存在する [17]。Power Capping を実現するためには様々なアプローチがあるが、定期的に使用電力を監視して電力制約に近づくとプロセッサの周波数を下げる・実行するタスクを減らす、といったような手法が提案されている。

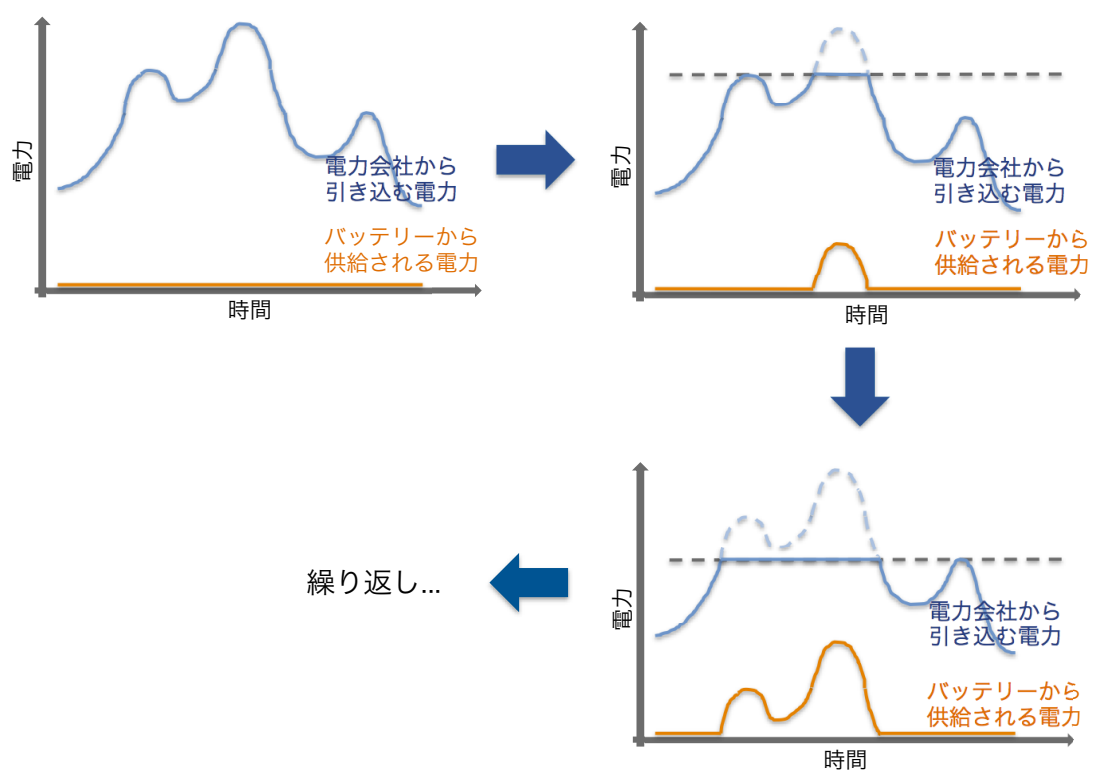


図 2.4. 紹介論文 [2] における UPS を用いた電力ピークカットアルゴリズム

第3章

蓄電池を用いた高速化手法

前章でデータセンタにおける電力コスト削減要請についての背景および関連研究を紹介した。実は電力削減はデータセンタだけに限らず、コンピュータアーキテクチャ全体の共通の課題である。その中でも HPC (High Performance Computing) 領域においては消費される電力は物理的制約による供給可能電力に達しつつあり、近い将来スーパーコンピュータの性能は電力供給能力によって頭打ちになると予想されている。そのため与えられた電力制約下において性能の高いシステムの構築が必要とされている。本章ではその実現手法のひとつとして、既存設備に含まれる蓄電池を用いた電力制約下での性能向上手法を提案する。

3.1 フェーズ間の電力融通手法の提案

図 3.1 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 3.3 節で述べる、フェーズの開始と終了を示す文をアプリケーションのソースコードに埋め込む。次に、プロセッサが取りうる全ての周波数を設定し、アプリケーションのテスト実行を行う。このとき、提案システムはプログラマによって埋め込まれたフェーズ区切り文を読み取り、各フェーズでの平均電力・実行時間を計測し、プロファイル情報を作成する。プロファイル情報をもとに、提案システムは 3.5 節で述べるアルゴリズムを用いて最適な充放電計画を決定する。最適な充放電計画とは、与えられた電力制約下において実行時間を最小とするような、フェーズごとのプロセッサ周波数及び充放電量である。本番実行時にはそれぞれのフェーズが実行される直前に、提案システムによって前述した充放電計画に従ってプロセッサの周波数が設定されることにより、最適な充放電制御が行われることになる。ここで、プログラマが行わなければならない作業はソースコードにフェーズ区切り文を埋め込むことのみであり、最適な充放電計画の決定及び実行は提案システムによって自動的に行われることに注意されたい。

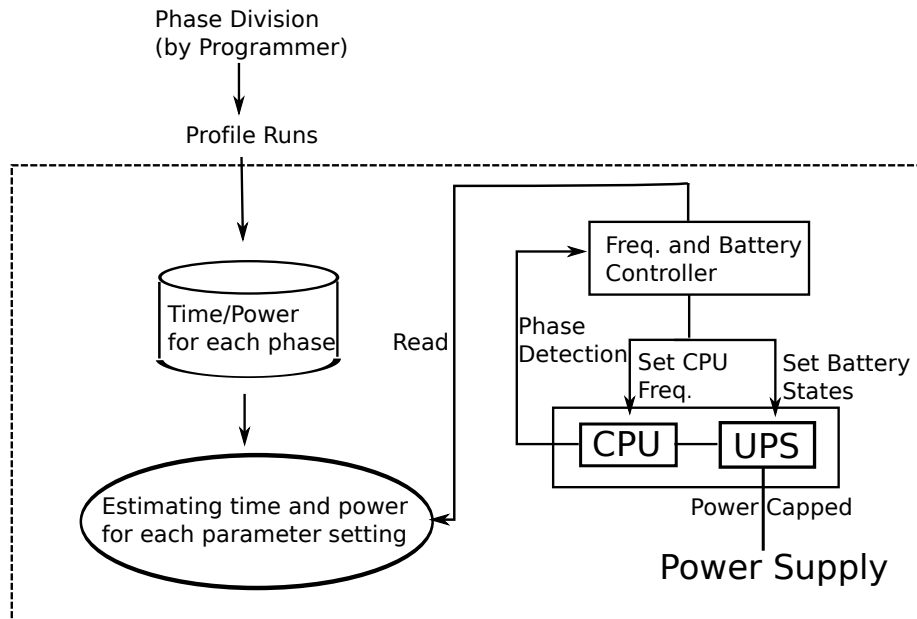


図 3.1. 提案手法の全体像

3.1.1 蓄電池を用いた電力融通によって実現される 実行時間短縮のメカニズム

スーパーコンピュータ上で走る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。CPU による演算中心の処理やメモリアクセスなどの I/O が中心の処理、プロセス間の通信が中心の処理など、異なる処理は基本的に異なる特性を持っている。また同じ処理であっても演算の並列度などの他の多くの要因に処理の特性は影響される。一般に処理の特性が異なると、その処理にかかる電力と処理を終えるまでの実行時間の関係を表した電力－実行時間曲線は異なったものになる。

本手法では処理の特性の違いによる電力－実行時間曲線の違いに着目する。アプリケーションの中に異なる電力－実行時間曲線をもつ区間（以下、それぞれの区間をフェーズと呼ぶ。）が複数ある場合には蓄電池を用いた電力融通を行うことで、以下の 3 つの効果により電力制約下での性能向上を行うことができると考える。

ケース 1

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ことによる性能向上

ケース 2

電力消費が小さく、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ことによる性能向上

ケース 3

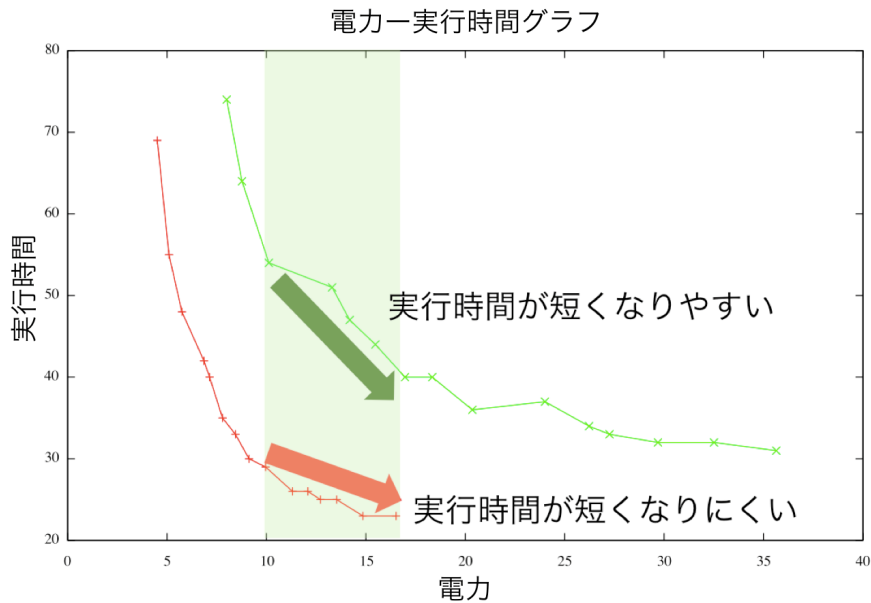


図 3.2. 電力－実行時間曲線の違いによって、かける電力の増加による短縮される実行時間が異なる様子（ケース 1 の効果）

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ことによる性能向上

まず、ケース 1 の効果について説明する。異なる電力－実行時間曲線において、かける電力を変化させたときの実行時間の変動の大きさは異なる（図 3.2）。かける電力を減らしても実行時間があまり短くならないフェーズから、電力を多くかけると実行時間が大きく短くなるフェーズへ蓄電池を用いて時間方向へ電力を融通することにより実行時間を短縮することができる。

次に、ケース 2 の効果について説明する。軽い処理を行っているフェーズでは、プロセッサを最大周波数で動かしても電力制約分の電力を使い切ることができない。従来ではその余剰電力は無駄になっていたが、蓄電池を用いれば電力制約以上の電力を必要としている他のフェーズへその電力を融通することができ、性能向上が見込める。2.3 節で紹介したデータセンタにおける蓄電池を活用した電力融通手法では、この効果により電力ピークカットを実現している。

最後にケース 3 の効果についてであるが、現在のプロセッサは離散的な有限の数の周波数でしか動作することはできない。そのため、電力制約によって最高周波数で動作できない場合であっても、使い切れていない電力というものが存在する。例えば、消費電力 80W および 60W の 2 種類の動作周波数をサポートしているプロセッサに対して 70W の電力制約をかけた場合、プロセッサは消費電力 60W の周波数でしか動作することができないので、10W の電力を使い切れないことになる。蓄電池を用いればこの余った電力も他のフェーズへ融通することができるので、その分実行時間の短縮に繋がると考えられる。

以上の 3 つの効果による電力制約下での性能向上が本論文の提案手法である。次節以降で

は、アプリケーションの区切り方および電力融通をどう行うかについての各論を述べる。

3.2 フェーズの要件

3.1 節で述べたように、本手法では処理ごとの電力ー実行時間曲線の違いを利用して性能を向上させる。そのため、電力ー実行時間曲線の異なる区間をそれぞれ別のフェーズとして定義する。フェーズの区切り方が細かいほど電力融通の機会は増えることになるので、理想的なバッテリーを用いる場合には、アプリケーション内の電力ー実行時間曲線が異なる区間全てを別のフェーズとして区切る場合が理想的なフェーズの区切り方となる。

しかし、実際のバッテリーはあまり高頻度に充放電を行えない・充放電の速度に限界があるなど様々な物理的制約がある。また、フェーズの数が多くなるほど 3.4 節で述べる電力融通問題を解くことが困難になる。さらに、入力データが異なる場合には処理の順序が異なるので、どの時刻にどの処理が行われているかが分かりにくく、アプリケーション内の電力ー実行時間曲線が異なる部分を全て求めること自体も難しい問題である。そのため、現実の問題を扱う場合には電力ー実行時間曲線が異なる全ての部分ではなく、もっと粗い粒度でフェーズを区切ることになる。

3.3 フェーズの求め方

HPC 領域において、アプリケーションの性質や演算装置の特徴に応じてソースコードに手を加えることは珍しいことではない。ソースコードに手を加えるときに手間となるのはソースコードを書き直すことである。以下のコードは CPU 並列化プラットフォームの OpenMP のコードであるが、このようにいくつかのコードを書き足す程度であればプログラマの大きな負担にはならない。

— OpenMP のソースコード —

```
int main(int argc, char *argv[])
{
    int i;
    #pragma omp parallel for          //性能向上のために追加される唯一の行
    for(i = 0; i <= 10000; i++)
    {
        // (並列処理させたいプログラム)
    }
}
```

本手法では、上の例のようにユーザプログラマにいくつかのコードを足してもらうことによりフェーズを区切る。具体的には以下のようなになる。

—— 本手法で想定するフェーズの指定方法 ——

```
int main(int argc, char *argv[])
{
    #pragma phase start A    //フェーズ A の始まりを示す
    //フェーズ A の処理
    #pragma phase end A      //フェーズ A の終わりを示す
    //何らかの処理 (ない場合もある)
    #pragma phase start B    //フェーズ B の始まりを示す
    //フェーズ B の処理
    #pragma phase end B      //フェーズ B の終わりを示す
}
```

この区切りを示すコードは、時系列的に異なる処理の部分に配置さえされていれば、以下のように繰り返し文の中に入っていたり、他の関数にまたがっていたりしても構わない。

—— 繰り返し文における例 ——

```
int main(int argc, char *argv[])
{
    for(i = 0; i <= 10000; i++)
    {
        #pragma phase start A    //フェーズ A の始まりを示す
        //フェーズ A の処理
        #pragma phase end A      //フェーズ A の終わりを示す
        #pragma phase start B    //フェーズ B の始まりを示す
        //フェーズ B の処理
        #pragma phase end B      //フェーズ B の終わりを示す
    }
}
```

関数にまたがっている場合の例

```

functionA()
{
#pragma phase start A    //フェーズ A の始まりを示す
    //フェーズ A の処理
#pragma phase end A      //フェーズ A の終わりを示す
}

functionB()
{
#pragma phase start B    //フェーズ B の始まりを示す
    //フェーズ B の処理
#pragma phase end B      //フェーズ B の終わりを示す
}

int main(int argc, char *argv[])
{
    functionA();
    functionB();
}

```

3.4 電力融通問題の定式化

3.3 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 にかかる電力 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 DVFS パターンについてテスト実行を行うことで得ることができる。この点については 4.2 節で詳しく述べる。

本論文での目的は、与えられた電力制約下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である。そこで、与えられる電力制約を p_{max} とする。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。

$$\min \sum_{i=1}^n T_i(p_{max} + \Delta p_i) \quad (3.1)$$

$$\text{s.t. } \sum_{i=1}^n \Delta p_i T_i(p_{max} + \Delta p_i) \leq 0 \quad (3.2)$$

式 (3.1) は実行時間の最小化を意味する。式 (3.2) 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ーを意味している。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、エネルギーを増やすことはできない。そのため、式 (3.2) はエネルギー保存制約式となる。

また、 $T_i(p)$ はおおまかに近似をとると式 (3.3) で表される直角双曲線になるため、式 (3.1)、(3.2) で定式化される最適化問題は非線形計画問題となる。

$$T(p) = a_0 + \frac{a_1}{p - a_2} \quad (3.3)$$

ただし、ここで対象としている蓄電池は以下のような特徴をもつ理想的な蓄電池である。

1. 電池容量無限
2. 充放電速度無限
3. 充放電によるエネルギー損失はない

想定している蓄電池では電池容量が無限であるので、アプリケーションの実行前後でのエネルギーが保存されていれば、アプリケーションの実行途中で電池容量がゼロになることはない。同様に、充放電速度が無限でエネルギー損失はないので、充放電することにより失われる電力はゼロである。

3.5 電力融通問題の解法

非線形計画問題を解くアルゴリズムはいくつも研究されている。しかし、3.1 節でも述べたように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。そのため、式 (3.1)、(3.2) の中の Δp_i は有限のパターンしか存在しない。本論文では Δp_i が有限個の値しか取れないことを利用して、全てのフェーズにおいて取りうる全ての Δp_i を実際に代入してアプリケーション全体の実行時間を計算することにより、最適な $\Delta p_i (1 \leq i \leq n)$ を求める。また、フェーズ i での充放電量 Δp_i と動作周波数 f_i は 1 対 1 に対応づけられるので、それぞれのフェーズにおける動作周波数 $f_i (1 \leq i \leq n)$ も求められることになる。

第 4 章

実験

4.1 実験の目的

本実験を行うことで、蓄電池を用いた電力融通手法による電力制約下での性能向上効果を見積もる。蓄電池の特性としては 3.4 節で述べたような理想的な特性を仮定する。また 3.1 節で述べたように、蓄電池を用いた電力融通による電力制約下での性能向上が起こる場合には 3 つのケースがある。そこで、それぞれのケースがどの程度の効果をもたらしているかを評価・考察する。

4.2 実験方法

本手法を用いた性能向上を実現するためには以下の 3 つの段階を踏む必要がある。

1. ユーザがアプリケーションのソースコードに埋め込んだフェーズ区切り文を読み取り、アプリケーションを分割する。
2. 分割されたフェーズそれぞれについて電力－実行時間曲線を求める
3. 分割されたフェーズに対する電力融通問題を解く

まず、1 段階目について述べる。実際に 3.3 節のように書かれたソースコードからフェーズ区切り文を読み取るのはコンパイラレベルでの実装が必要となり困難である。そのため、本実験ではそのフェーズに入ったもしくは出た時刻をログとして出力する自作関数をソースコードに埋め込むことによって、その代わりとした。また、今回実験に利用したアプリケーションは自作のものではないので、ソースコードを読んでどの部分がフェーズの区切りであるかを知ることが困難であった。そこで簡単のため、並列処理部分と逐次処理部分のみをソースコードから判別してフェーズ分割を行った。そして、プロセッサの取りうる全ての DVFS パターンについてアプリケーションを実行し、ログファイルを得た。ただし表 4.1 に示したアプリケーションのうち、QT Clustering のみは並列処理の中に 2 つのフェーズがあったため、全体で並列処理 1・並列処理 2・逐次処理の 3 フェーズに分割した。

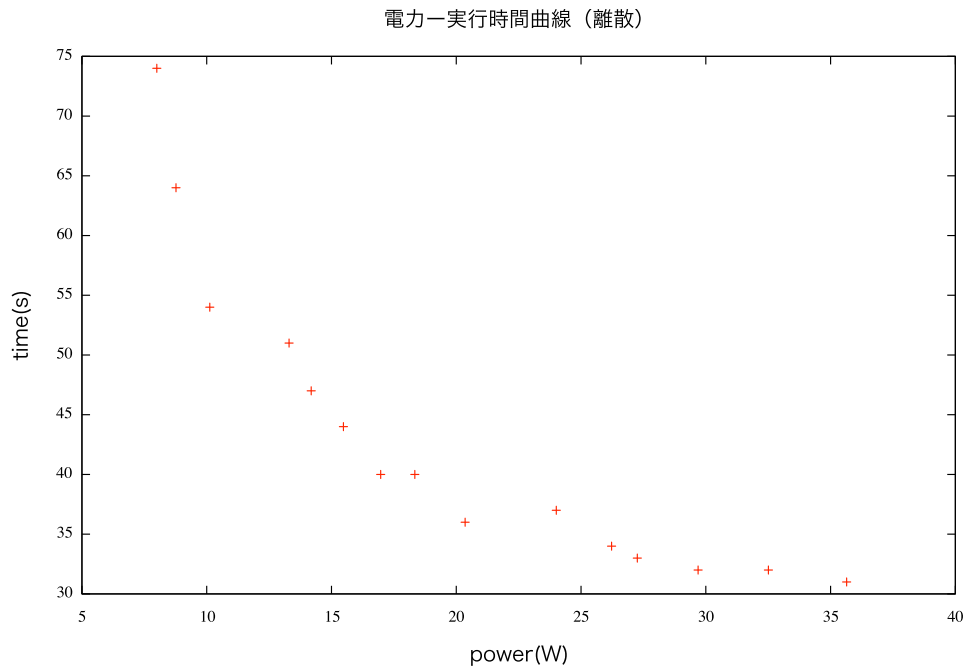


図 4.1. 離散的な周波数しか取れない場合の電力－実行時間曲線

次に 2 段階目である。ここでは 1 段階目で得たログファイルから、それぞれのフェーズについて全ての DVFS パターンにおける平均電力と実行時間を得ることができる。それを用いてそれぞれのフェーズの電力－実行時間曲線を計算した。ただし実際には DVFS パターンは有限であるので、電力－実行時間曲線は図 4.1 のようになった。

最後に 3 段階目は、3.5 節で述べたように全探索により最適な DVFS 設定値を見つけた。

また 4.1 節で述べたように、本実験の目的には電力制約下での性能向上をもたらす 3 つのケースのそれぞれの効果を評価することも含まれる。しかし、ここまでの実験方法ではそれぞれの影響が入り交じった結果しか得られないため、それぞれのケースの効果を個別に見積もることは困難である。ここで、ケース 3 の効果は離散的な周波数しか取ることができない現実のプロセッサでのみ生じることに着目した。そこで、離散的な電力－実行時間曲線を式 3.3 で連続補間することによって、上限周波数と下限周波数の間の全ての周波数で動作することができるよう理想的なプロセッサ特性を求めた（図 4.2）。この理想プロセッサ特性においても同様に実験を行うことにより、ケース 3 の影響のみを切り分けて評価できるようにした。ただし完全に連続であると電力融通問題を全探索で解くことができなくなるため、実際には十分に細かい間隔で補間することで連続的な曲線への近似とした。この結果を周波数が離散的な場合の結果と比較するにより、ケース 3 の効果のみを切り分けて見積もることができる。

用いたベンチマークアプリケーションは表 4.1 の通りである。PARSEC[18]、Rodinia[19]、SHOC[20] 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を選んでいる。CPU での並列アプリケーションでは CPU の電力のみ、GPU での並列アプリケーションであれば GPU の電力のみを測定した。

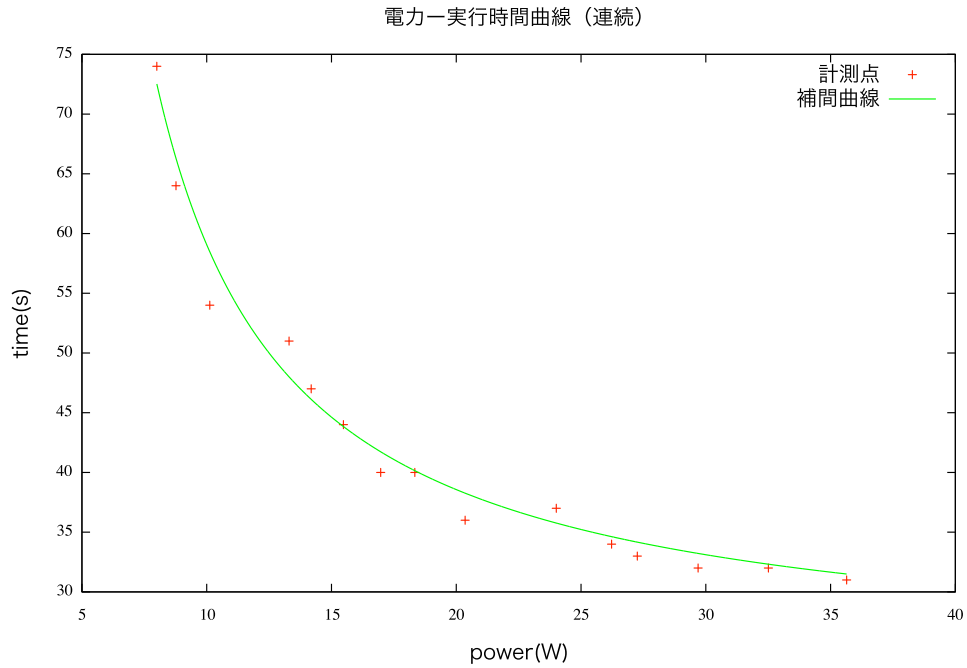


図 4.2. 連続補間した場合の電力－実行時間曲線

ワークロード名		説明
PARSEC Canneal	CPU 並列	SA アルゴリズムを用いてルーティングコストが最小となる chip を設計する.
PARSEC Stream Cluster	CPU 並列	ストリーミングされる点列のオンラインクラスタリングを行う.
Rodinia mummergpu	GPU 並列	深さ優先探索により DNA 塩基配列を求める
SHOC QT Clustering	GPU 並列	クラスタメンバ間の相関が指定されたカットオフ値より高いことを保証するクラスタリング

表 4.1. 使用したベンチマークアプリケーションの詳細

	名称	周波数 (MHz)
CPU	Intel Core i7 4770 4 コア	800 - 3200
GPU _s	NVIDIA K20c 2496 コア	324 - 758
MotherBoards	ASRock H87M	—
OS	Ubuntu 13.04	—

表 4.2. 実験機の構成

4.3 結果

それぞれのアプリケーションの電力－実行時間曲線を図 4.3、4.5、4.7、4.9 に、電力制約ごとの性能向上の様子を図 4.4、4.6、4.8、4.10 に示す。性能向上率は以下のように定義される。

ワークロード名	性能向上率平均
PARSEC Canneal	4.54%
PARSEC Stream Cluster	0.56%
Rodinia mummergpu	18.23%
SHOC QT Clustering	15.89%

表 4.3. 性能向上率の算術平均（周波数が離散の場合）

ワークロード名	性能向上率平均
PARSEC Canneal	2.22%
PARSEC Stream Cluster	0.02%
Rodinia mummergpu	10.44%
SHOC QT Clustering	2.50%

表 4.4. 性能向上率の算術平均（周波数が連続の場合）

$$\text{性能向上率} = 1 - \frac{\text{UPS ありの場合のアプリケーション全体の実行時間}}{\text{UPS なしの場合のアプリケーション全体の実行時間}} \quad (4.1)$$

離散の性能向上率とは、離散的な周波数でしか動作できないプロセッサで本手法を適用した場合の性能向上率である。連続の性能向上率とは、上限周波数と下限周波数の間の全ての周波数で動作することができる理想的なプロセッサで本手法を適用した場合の性能向上率である。ただし、理想的なプロセッサの上限・下限周波数は現実のプロセッサと同じであるとしている。

Canneal の電力－実行時間曲線は並列処理と逐次処理の電力消費の違いをよく表しており、並列処理の方が逐次処理より大きい電力を消費していることが分かる。Stream Cluster も同様の傾向は見えて取れるが、Canneal とは違って逐次処理部分の実行時間が非常に短いことが特徴である。mummergpu のグラフを見ると、どの DVFS 設定においても逐次処理部分の実行時間が変わらないことが分かる。これは 4.2 節で述べたように GPU 並列アプリケーションは GPU の電圧のみを測定しているため、CPU で演算をしている逐次処理部分の実行時間は GPU の周波数に影響を受けないためである。QT Clustering では 3 つのフェーズがあり、mummergpu 同様に逐次処理部分は DVFS の設定によらずほぼ同じ実行時間になっている。

4.4 考察

4 つのアプリケーションの中で、Stream Cluster のみは性能向上がとても小さかった。これは、逐次処理部分の実行時間が並列処理部分の実行時間と比べて非常に短かったため、蓄電池が充放電する時間がほとんどなく、電力融通を上手く行うことができなかったためであると考えられる。

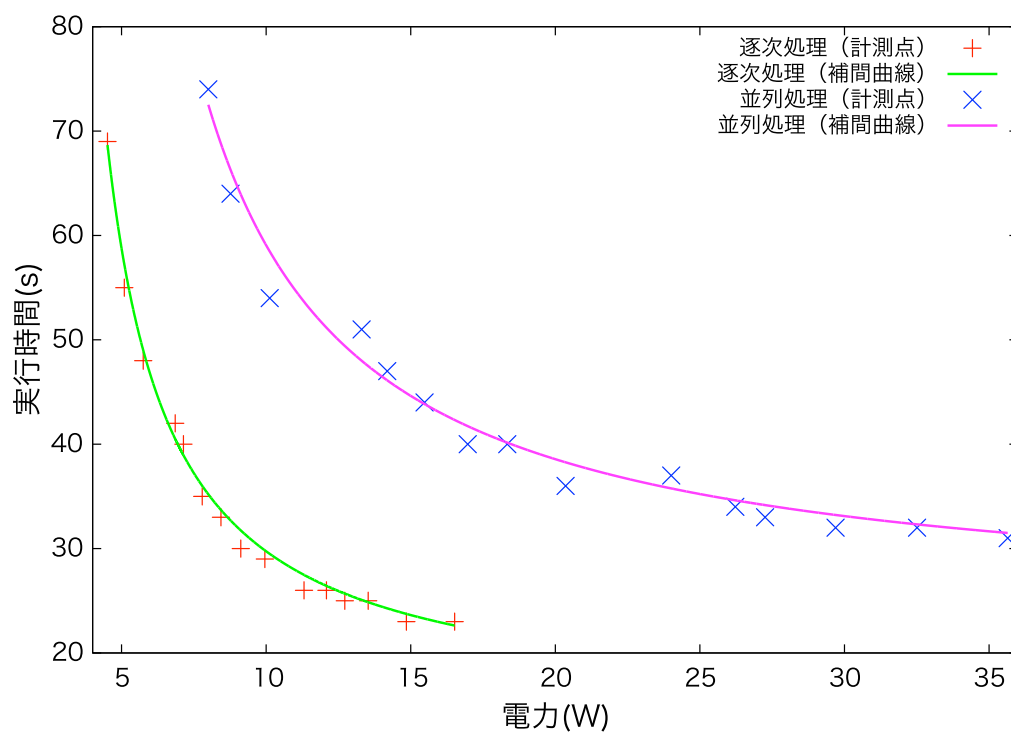


図 4.3. Canneal 電力－実行時間曲線

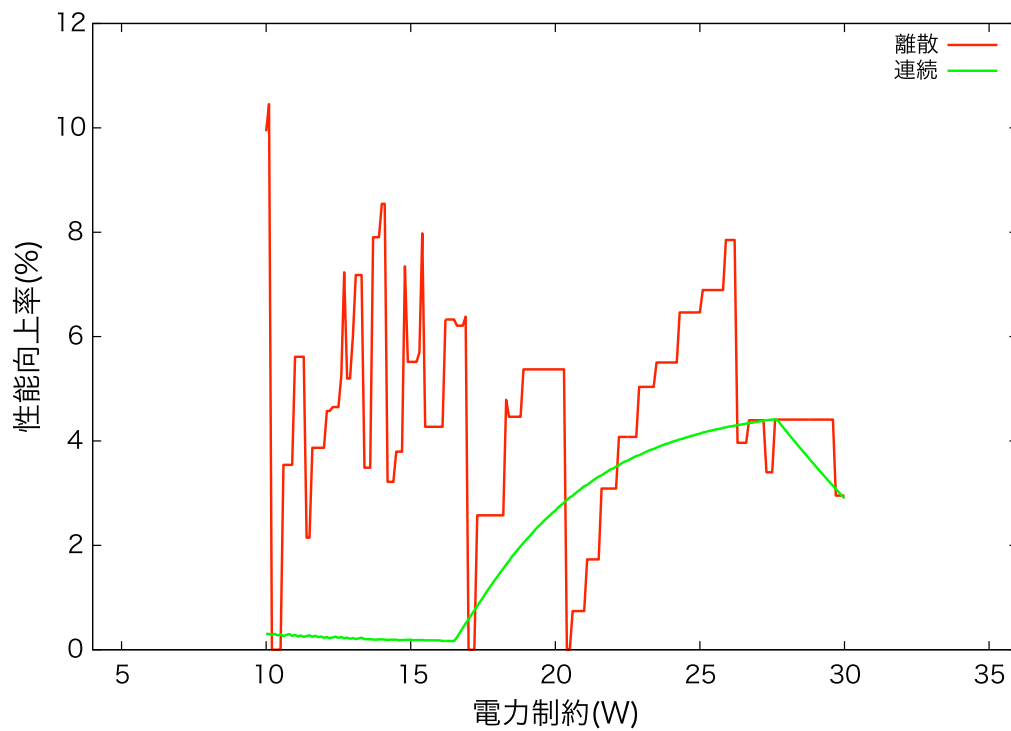


図 4.4. Canneal 電力融通を行わない場合と比べた性能向上

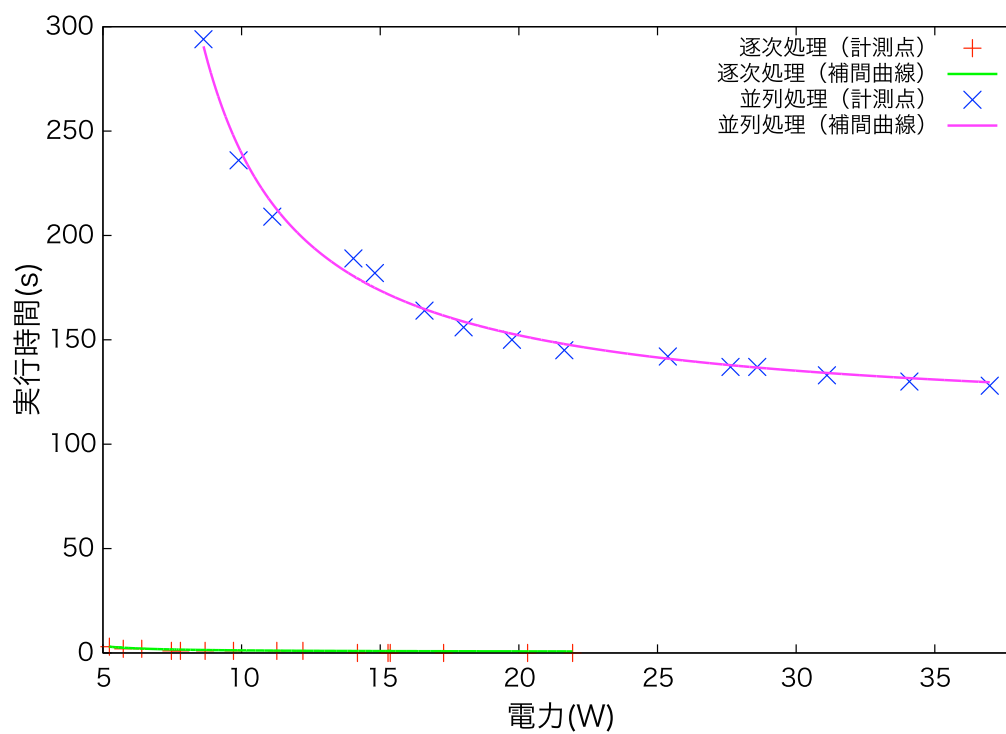


図 4.5. Stream Cluster 電力－実行時間曲線

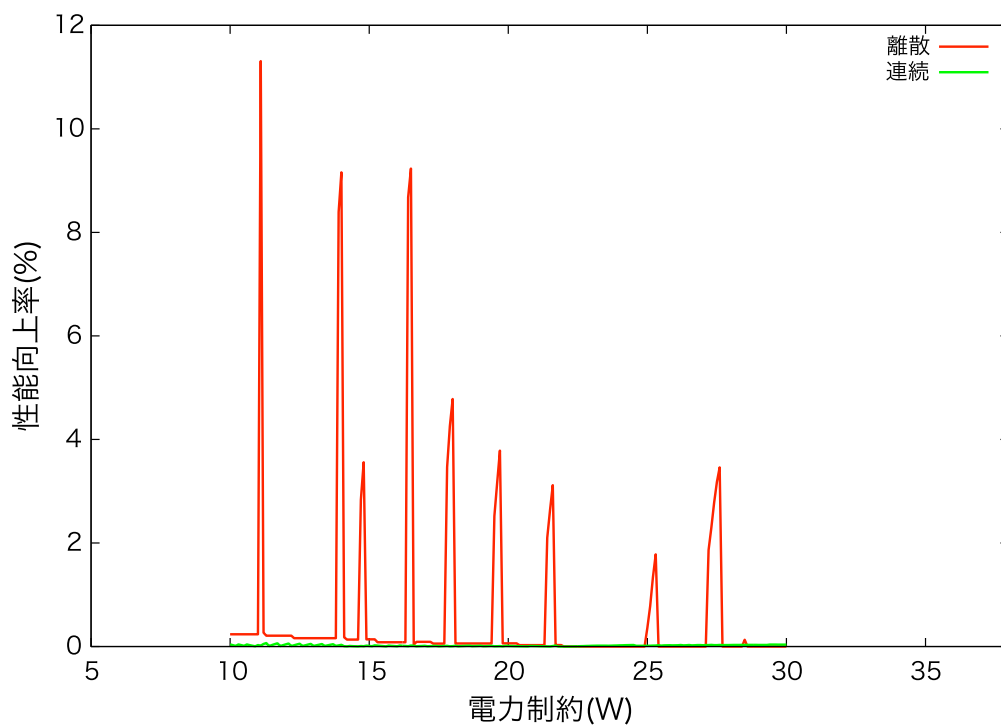


図 4.6. Stream Cluster 電力融通を行わない場合と比べた性能向上

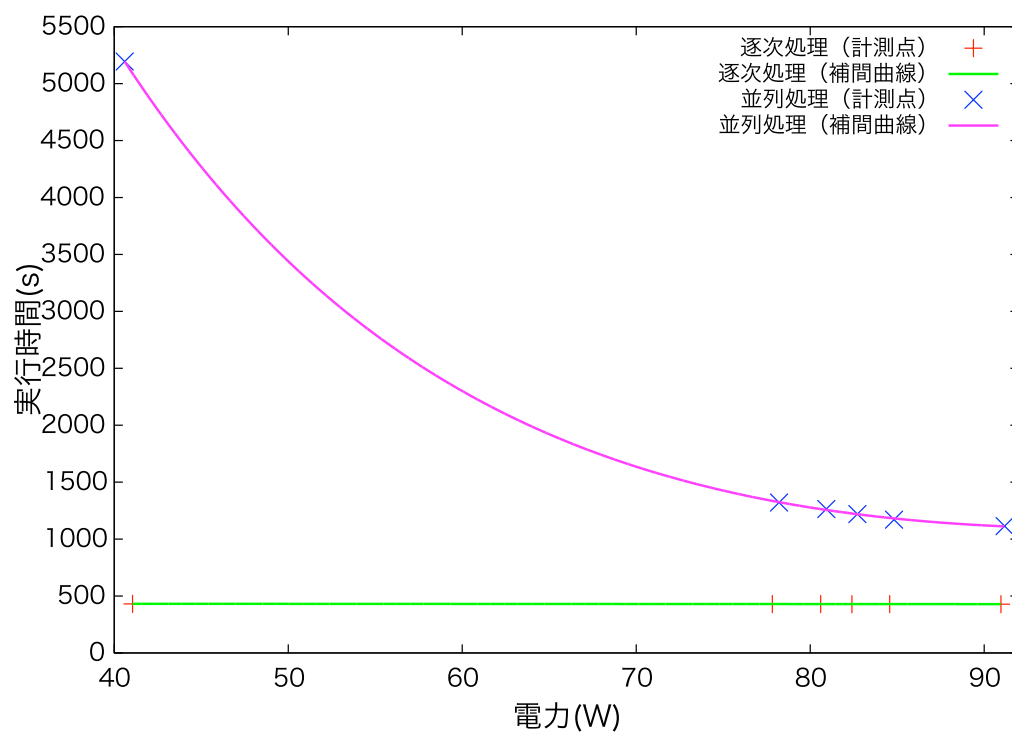


図 4.7. mummergpu 電力－実行時間曲線

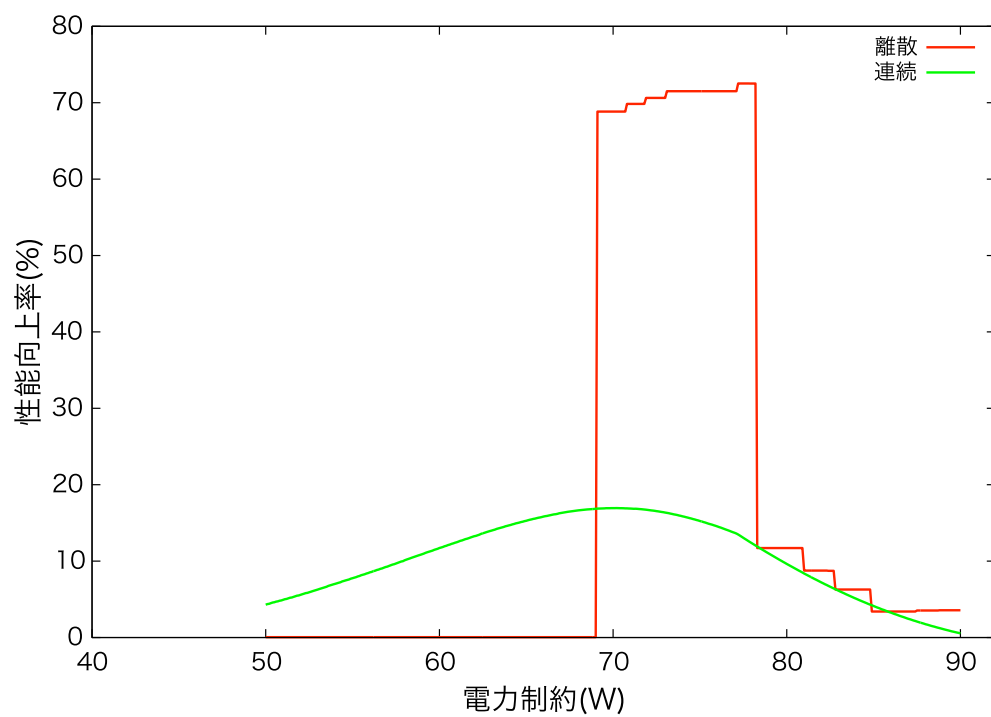


図 4.8. mummergpu 電力融通を行わない場合と比べた性能向上

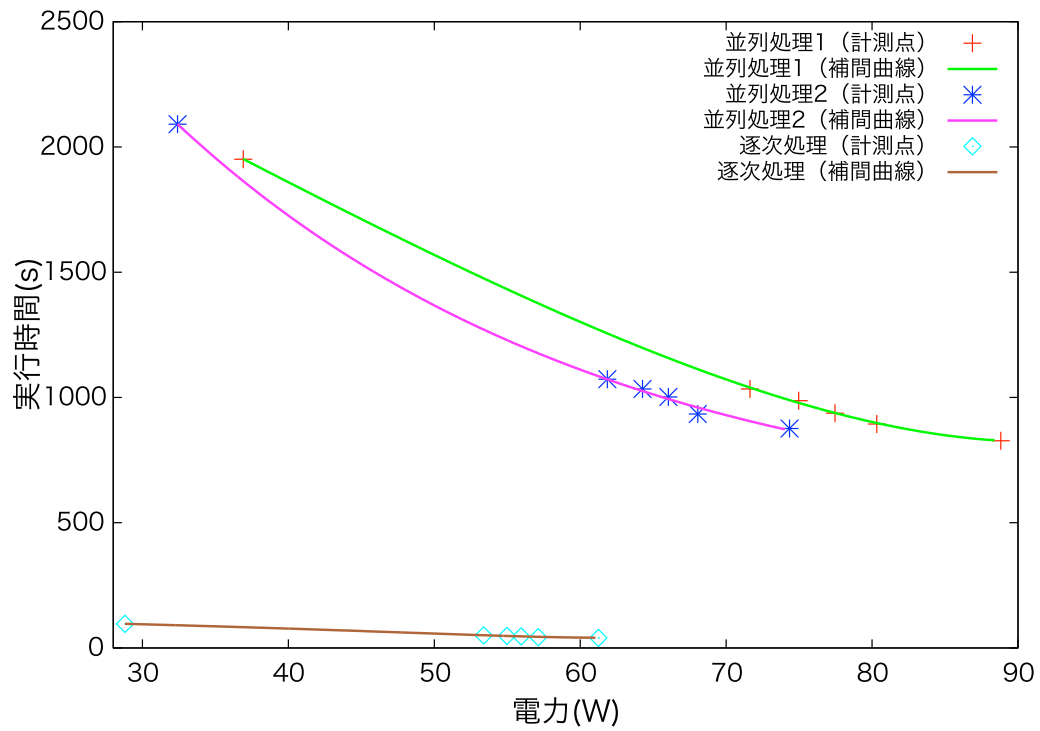


図 4.9. QT Clustering 電力－実行時間曲線

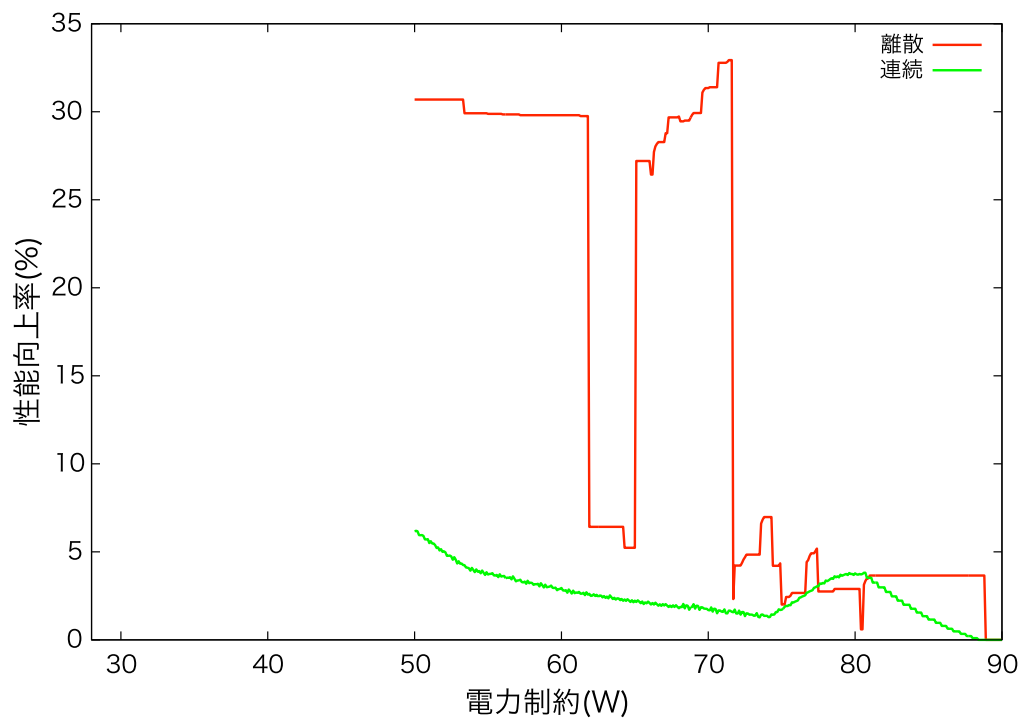


図 4.10. QT Clustering 電力融通を行わない場合と比べた性能向上

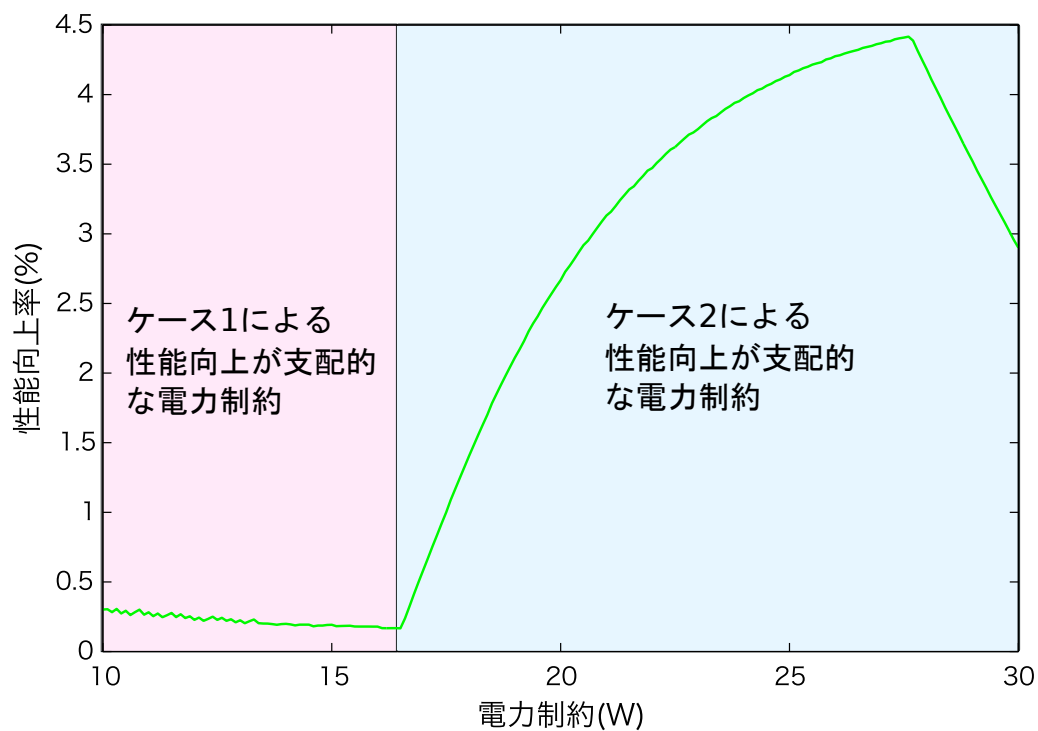


図 4.11. Canneal 電力融通を行わない場合と比べた性能向上（連続の場合のみ）

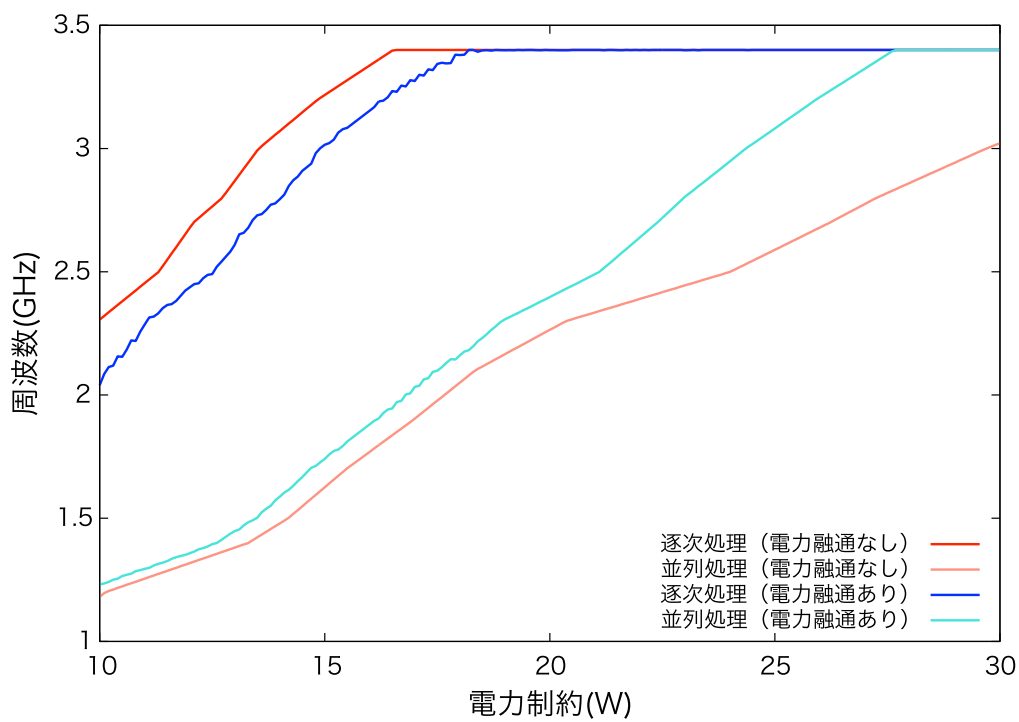


図 4.12. Canneal 周波数の推移の様子（連続の場合）

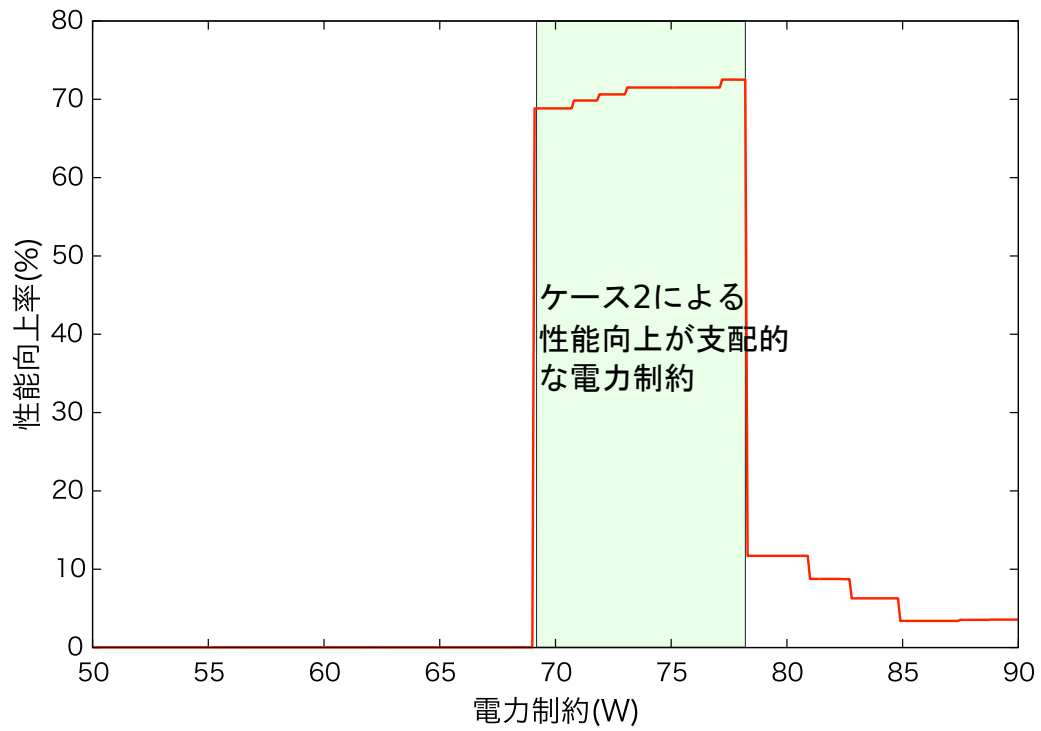


図 4.13. mummergpu 電力融通を行わない場合と比べた性能向上（分散の場合のみ）

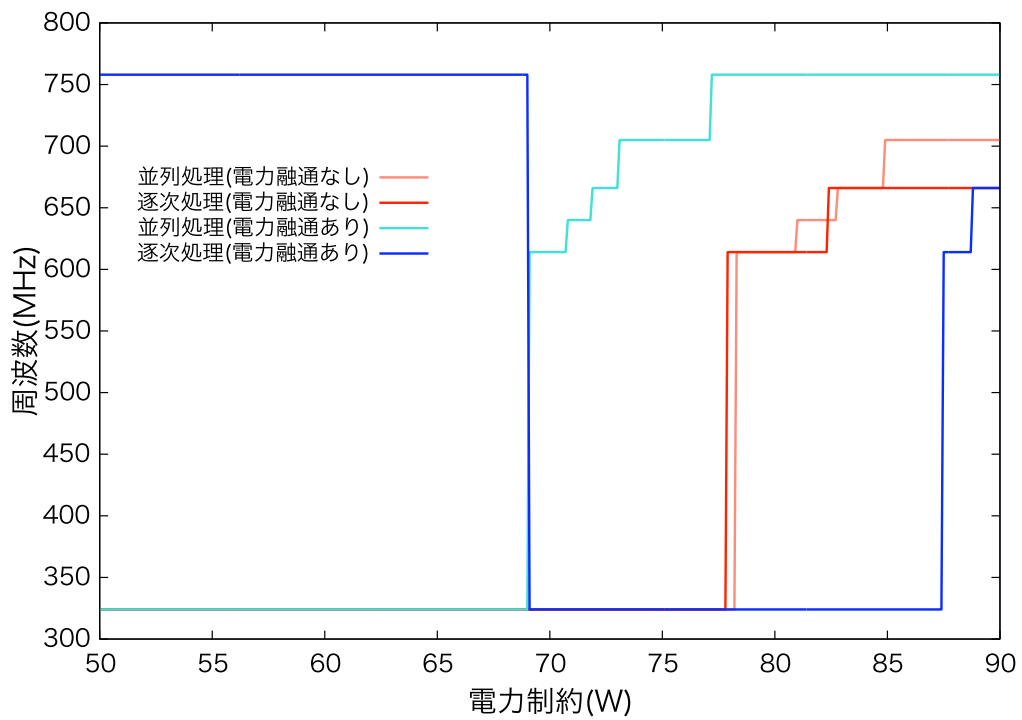


図 4.14. mummergpu 周波数の推移の様子（分散の場合）

4.4.1 周波数が連続的な値を取れる理想的なプロセッサの場合の考察

連続的に周波数を変更できる場合にどのような周波数が最適解となっているかについて考察する。ここでは比較的説明のしやすい、Canneal の連続補間した場合の結果について説明する。

図 4.11、4.12 はそれぞれ、周波数が連続的に変えられる場合の Canneal の電力-実行時間曲線と性能向上の様子である。図 4.11 を見ると、電力制約が 17W の部分から性能向上率が増加し始め、28W 付近でピークに達しその後は減少していくのが分かる。

まず、電力制約が 17W より小さいときの最適周波数について考える。図 4.11 に示したようにこの電力制約においては、ケース 1 による効果が支配的である。これは図 4.12 を見ると分かるように、この電力制約においては、どちらも周波数の上限・下限には達しておらず、ケース 2 の効果はないためである。また、逐次処理の周波数を下げて並列処理の周波数を上げることから、逐次処理部分は電力を上げても実行時間があまり短縮されないフェーズであり、逆に並列処理部分は電力を上げることで実行時間が大きく短縮されるフェーズであったということが分かる。

ここで、ケース 1 におけるそれぞれのフェーズの周波数はどのような法則によって決められているのかを考察する。注意すべきは、蓄電池によって融通しているものは電力ではなくエネルギーであるという点である。フェーズ A で 10W で充電できたからといって、フェーズ B で 10W で放電できるとは限らない。これはそれぞれのフェーズの実行時間が異なるため、充電電されるエネルギーに差が生じるためである。しかし、3.4 節で述べたような理想的な蓄電池の場合にはフェーズ A で 10J 充電できた場合はフェーズ B で必ず 10J 放電することができる。つまり、それぞれのフェーズにかかるエネルギーを微小変化させたときの、実行時間の短縮の大きさを見てやればよいことになる。これは、実行時間を実行エネルギーで微分したものを正負逆転させたものに他ならない。これをエネルギー実行時間短縮率と呼ぶこととする。エネルギー実行時間短縮率は式 3.3 を用いて以下のように求められる。

$$E = p \cdot T(p) \quad (4.2)$$

$$-\frac{dT}{dE} = -\frac{dT}{dp} \bigg/ \frac{dE}{dp} \quad (4.3)$$

$$= -\frac{1}{a_2 - \frac{a_0}{a_1}(p - a_2)^2} \quad (4.4)$$

実際に 17W 以下の部分については、式 4.4 の値がほぼ一致していた。つまりこの部分については、それぞれのフェーズのエネルギー実行時間短縮率の一致する部分が最適解となっている。また、この部分における性能向上率は 0.5% 未満とかなり小さい。

次に 17-28W の部分について考える。図 4.11 に示したようにこの電力制約においてはケース 2 の効果が支配的である。図 4.12 を見ると、17W のときに電力融通をしない場合の逐次処理部分での動作周波数が頭打ちになり、実行時間が短くなくなることが分かる。一方で電

力融通を行う場合には頭打ちになるのがもう少し遅く、頭打ちになってもその部分で余った電力を並列部分へ融通して周波数を上げることができるので、実行時間は短くなり続ける。そのため実行時間の差は開き続け、性能向上率が大きくなり続ける。

この電力制約における最適周波数での式 4.4 の値を見ると、逐次処理の値の方が小さい。これは逐次処理の方がエネルギー実行時間短縮率が大きいことを意味しているため、もし逐次処理の周波数をさらに上げることができれば、並列処理部分での動作周波数を下げ、代わりに逐次処理部分では動作周波数を上げていたと予想される。

最後に 28W 以上の部分について考える。この部分についての説明は容易である。図 4.12 から、28W のときに電力融通を行っている場合には逐次処理・並列処理のどちらのフェーズも動作周波数が頭打ちになる。従って、電力融通をしている場合にはこれ以上実行時間を短縮する余地がない。一方で電力融通していない場合には、並列処理部分の動作周波数をまだ上げることができるため、実行時間が短縮されていく。結果として性能向上率は減少していく。

以上それぞれのフェーズにおける考察より、プロセッサが連続的な周波数で動作できる場合にはエネルギー実行時間短縮率が高いフェーズから低いフェーズへできる限り電力を融通できるように充放電計画が最適となることが分かる。また、17W 以下の部分の性能向上率を見れば分かるように、電力-実行時間曲線の特性の違いによって生じるエネルギー実行時間短縮率の違いを利用した電力融通だけでは効果は小さい。一方で 17W 以上の部分のように、一方のフェーズにおけるプロセッサの性能が頭打ちになることによる余剰電力などがあれば大きな性能向上が見込める。

4.4.2 周波数が離散的な値のみしかとれない現実のプロセッサの場合の考察

離散の場合には電力制約によってはところどころ針が立ったように大きく性能向上している部分がある。これはケース 3 の効果に分類されるもので、もう少し電力制約が大きくなればもう一つ上の周波数に上げることができる、という状況で起こるものである。逐次処理でわずかに蓄えたエネルギーを並列処理部分に融通することで、周波数を一段階上げることができたのである。周波数が連続的に変えられる場合には、周波数を数段階上げる程度では大した時間短縮にはならないため、このようなことは起こらない。

全てのアプリケーションを通して、離散的な周波数しか取れない場合の方が連続的に周波数を変化させられる場合に比べて効果は大きかった。これは 3.1 節で述べたように、周波数が離散的であるために生じる余剰電力を蓄電池を用いて融通することによる性能向上効果のためであると考えられる。

全体的に、離散的な周波数しか取れない場合には電力制約ごとに性能向上率の値が大きく異なる。これまでに述べたように離散的であると余剰電力は多く生じるのだが、周波数を一段階上げるために必要なエネルギー融通量も大きくなる。また、周波数を一段階上げたときに短縮される実行時間も大きい。そのため、電力融通により上手く周波数を上げられるような電力制約の場合は大きく性能向上する一方で、逆に周波数を上手く上げられないような電力制約ではほとんど性能向上させることができないのである。

また、電力制約ごとにケース 1、2、3 のどの効果がどれほど性能向上率に影響しているかを定量的に評価することはできなかった。それぞれの効果がお互いに影響し合っており、分離することが実質的に困難なためである。

以上のことをまとめると、周波数が離散的である場合に効果があるのは、電力制約がサポートされているいずれかの動作周波数よりわずかに小さいような場合である。この場合には他のフェーズの余剰電力を蓄電池で融通することで周波数を上げることができる可能性が高いため、実行時間が大きく短縮される。逆に効果が薄いのは電力制約がいずれかの動作周波数と同じか、わずかに大きい場合である。この場合には次の周波数まで大きなギャップがある上、余剰電力もあまり生じないため電力融通を上手く行うことができない。

4.4.3 全体の考察

今回の実験の結果、離散的な周波数しか取れないプロセッサ上において、同程度の実行時間のフェーズを複数有するアプリケーションに本手法を適用することで、CPU 上で実行されるアプリケーションでは平均 4.5%、GPU 上で実行されるアプリケーションでは平均 17.1% の実行時間の短縮がなされることが分かった。3.1 節で述べたケース 1、2、3 のそれぞれの効果がどれくらい実行時間の短縮に寄与したかを定量的に評価するには至らなかった。ただし、図 4.11、4.13 におけるそれぞれの効果を見る限りでは、ケース 2、3 の実行時間短縮効果はケース 1 に比べて大きいようであった。

今回の実験では電力融通問題を解くために全探索で最適解を求めた。 n 個のフェーズがあるアプリケーションを、 m 個の動作周波数をサポートしているプロセッサ上で走らせた場合の最適解を全探索で求めると、その計算量は $O(m^n)$ となる。これは非常に大きな計算量であるため、新たなアルゴリズムの構築が求められる。そのため、今後は 4.4.1 節で述べたようなエネルギー実行時間短縮率を指標とした計算量の小さなアルゴリズムの考案が課題となるであろう。また今回は理想的な蓄電池を想定したが、蓄電池の電池容量や充放電速度などの物理的制約を考慮に入れたアルゴリズムの構築も課題である。

第 5 章

結論

本研究では、蓄電池を用いた、電力制約下における性能向上手法を提案した。具体的には、まずユーザーに処理区間の境界を示す指示文をソースコードに埋め込ませる。次に、アプリケーションごとに電力-実行時間曲線を取得し、埋め込んだ情報を元にこの曲線を複数区間に分割する。そして、電力制約下で実行時間を最小化する様に、区間ごとの蓄電池の充放電計画を立てる。本提案手法の効果を実機上で検証した結果、実行時間の長さが同程度のフェーズを複数持つ CPU 用の並列アプリケーションでは平均 4.5%、GPU 用の並列アプリケーションでは平均 17.1% の実行時間の短縮がなされることを確認した。

ただし、本提案手法では、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ている。現実の蓄電池では、電池容量・充放電速度・最小放電間隔や寿命などに物理的制約があるため、これを考慮する必要がある。また、本研究では充放電計画の決定アルゴリズムとして全探索を用いている。しかし、上記の様に制約条件を増やすと計算量も増大するため、全探索よりも計算量の小さなアルゴリズムの構築が必要となる。そのためには、4 章で定義したようなエネルギー実行時間短縮率などの評価関数を用いることが有用であると考えられる。

また、今後の更なる展望として CPU・GPU の両者を用いて並列処理を行う場合への、本手法の応用が考えられる。この場合、蓄電池を用いた時間方向の電力融通と、CPU・GPU 間の空間方向の電力融通の相乗効果により、更なる電力対性能の向上が期待できる。

謝辞

本研究を進めるにあたり、ご指導を頂きました中村研究室のスタッフ・先輩方に深く感謝致します。お忙しい中、ミーティングの中で日々の卒業研究へのご指導を頂きました中村宏教授に心から感謝致します。また、理論的な考察や手法についてや研究の方向性への様々なアドバイスを下さった三輪忍助教、些細な質問にも丁寧に答えて頂き、日々の相談にもものって下さった中田尚特任助教兩名に深くお礼申し上げます。

薦田先輩には博士論文や審査でお忙しい中、毎日のように私の研究にとっても多くの時間を割いてお世話頂き本当に感謝しております。薦田先輩なしでは今の私の卒業論文はなかったでしょう。有間先輩には数学的な手法や理論についてアドバイス頂きました上に卒論本文にもついてもたくさんの助言を頂き、大きな助けになりました。會田先輩にはプログラム関連で色々助けて頂きまして、本当に助かりました。皆様に心から感謝致します。

参考文献

- [1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, pages 114–117, April 1965.
- [2] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. *SIGARCH Comput. Archit. News*, 39(3):341–352, June 2011.
- [3] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis 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, New York, NY, USA, 2013. ACM.
- [4] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.
- [5] Wonyoung Kim, M.S. Gupta, Gu-Yeon Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134, 2008.
- [6] Howard David, Chris Fallin, Eugene Gorbatoov, Ulf R. Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.
- [7] Qingyuan Deng, D. Meisner, A. Bhattacharjee, T.F. Wenisch, and R. Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 143–154, 2012.
- [8] Barry Rountree, David K. Lowenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

- [9] Efficient Data Center Summit 2009. <https://www.google.com/about/datacenters/efficiency/external/2009-summit.html>.
- [10] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 303–314, New York, NY, USA, 2005. ACM.
- [11] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.
- [12] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: Coordinated multi-level power management for the data center. *SIGARCH Comput. Archit. News*, 36(1):48–59, March 2008.
- [13] L. Ramos and R. Bianchini. C-oracle: Predictive thermal management for data centers. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 111–122, Feb 2008.
- [14] Xiaorui Wang and Ming Chen. Cluster-level feedback power control for performance optimization. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 101–110, Feb 2008.
- [15] L. Ganesh, J. Liu, S. Nath, G. Reeves, and F. Zhao. Unleash stranded power in data centers with rackpacker. In *Proceedings of the Workshop on Energy-Efficient Design (WEED)*, 2009.
- [16] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.
- [17] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News*, 35(2):13–23, June 2007.
- [18] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai 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, New York, NY, USA, 2008. ACM.
- [19] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin 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, Washington, DC, USA, 2009. IEEE Computer Society.

- [20] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The 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, New York, NY, USA, 2010. ACM.

発表文献

- [1] 酒井崇至, 薦田登志矢, 三輪忍, 中村宏. 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. 情報処理学会研究報告, Vol. 2013-HPC-143, No. 25, pp 1–7, 2014 (予定)