

メモリ消費電力に基づく CPU 周波数動的制御手法の評価

三輪 真弘^{1,a)} 中島 耕太¹ 平井 聡¹ 風間 哲¹ 原 靖¹ 成瀬 彰¹

受付日 2012年3月28日, 採録日 2012年6月7日

概要: 本論文では, メモリ消費電力に基づく CPU 周波数動的制御手法について述べる. 既存の研究事例では, Memory-bound であるアプリケーションの実行時に CPU 周波数を低く設定する省電力化手法を Performance Monitoring Counter (PMC) に基づき行う. しかし, PMC はマシン異常監視 (nmi watchdog) やプロファイラといった様々な用途で利用されるため, PMC をアプリ特性判定に利用するとこれらの用途への利用が制限される. また, PMC によるアプリ特性判定はソフトウェア環境に依存するため, 運用管理面で手間を要することがある. さらに, アプリ特性を PMC によって判定すると, アプリ特性判定のソフトウェア処理がターゲットホスト上で走行し CPU リソースを消費するという問題がある. 本論文では, Memory-bound であるかどうかのアプリ特性をメモリ消費電力に基づき判定する. Memory-bound であるアプリケーションの実行時にメモリ消費電力が高く, CPU-bound であるアプリケーションの実行時にメモリ消費電力が低いという特性を利用して, メモリ消費電力によりアプリ特性を判定する. そして, メモリ消費電力が高い場合に CPU 周波数を低く設定し, 省電力化する. 提案手法は, メモリ消費電力を用いるため PMC の用途を制限しない. また, 提案手法はセンサ情報に基づくため, 外部サーバコントローラから利用可能であり, ソフトウェア環境に依存しない実装が可能である. さらにこの場合, アプリ特性判定に CPU リソースを消費しない. NPB ベンチマーク 8 種による評価では, 平均 3% の性能低下に対し, 平均 5% の CPU 消費電力量を削減できた. 特に, is.C ベンチマークでは 5% の性能低下に対し, 15% の CPU 消費電力量の削減を確認できた. また提案手法は, Last Level Cache Miss イベントを利用した PMC の手法に対し同程度の効果を確認でき, 本手法の有効性が確認できた.

キーワード: Dynamic Voltage and Frequency Scaling (DVFS), 省電力, メモリ消費電力, Performance Monitoring Counter (PMC)

Evaluation of Dynamic Voltage and Frequency Scaling Adaptation Based on Memory Power

MASAHIRO MIWA^{1,a)} KOHTA NAKASHIMA¹ AKIRA HIRAI¹ SATOSHI KAZAMA¹
YASUSHI HARA¹ AKIRA NARUSE¹

Received: March 28, 2012, Accepted: June 7, 2012

Abstract: This paper presents a novel approach of DVFS adaptation based on memory power. Low power system that set a lower frequency to CPU when a Memory-bound application is running is studied. The judgment of application type whether an application running is Memory-bound or not is done based on Performance Monitoring Counter (PMC). PMC, however, is used for various usages such as NMI watchdog and Profiler. So, if PMC is used for the judgment of application type, the other usage of PMC is restricted. And, the usage of PMC is dependent of software environments. Also, there is software overhead of PMC reading. In this paper, we use memory power for the judgment of application type. We found out that when a Memory-bound application is running, high memory power is consumed, and when a CPU-bound application is running, low memory power is consumed. So, our method uses memory power for the judgment of application type. Our method uses only memory power and could be implemented in baseboard management controller, for memory power is captured by a sensor. In this case, no software overhead is needed in target host for the judgment. Experiment results using the 8 benchmarks from NPB benchmark set show that 5% (average) CPU energy reduction under 3% (average) performance degradation is achieved. Especially for is.C benchmark, 15% CPU energy reduction is achieved under 5% performance degradation. Our method has almost the same effect as the method using the count of Last Level Cache Miss event of PMC. So the validity of our method is proved.

Keywords: Dynamic Voltage and Frequency Scaling (DVFS), energy saving, memory power, Performance Monitoring Counter (PMC)

1. はじめに

省電力への要求はますます高まっている。モバイル機器に限らず、PC・サーバにも省電力は求められており、PC・サーバにとって「省電力」は「高性能」に並び、重要な要件になっている。近年のプロセッサには、DVFS (Dynamic Voltage and Frequency Scaling) と呼ばれる CPU の周波数および電圧を変更する機構がある。CPU の消費電力は、周波数と電圧の二乗の積となるため、低 CPU 周波数、低電圧にすることで CPU 消費電力を小さくすることができる [1]。DVFS を活用した例に、Linux の Ondemand Governor [2] がある。Ondemand Governor は、CPU 使用率に応じて CPU 周波数を制御する手法であり、低 CPU 使用率時には低 CPU 周波数、高 CPU 使用率時には高 CPU 周波数に設定する。低 CPU 使用率時に CPU 周波数を低く設定してもシステム性能に対する影響は小さいため、性能を維持し、省電力化できる。

また、CPU 使用率が高い場合にシステム性能低下を抑えつつ、CPU 周波数を制御し、省電力化する研究がある。この研究では、実行中のアプリケーションが CPU-bound か Memory-bound かどうかのアプリケーション特性 (アプリ特性) を調査し、アプリ特性が Memory-bound である場合に CPU 周波数を低く設定することで省電力化を実現している。Memory-bound であるアプリケーションの実行性能はメモリ処理速度に決定付けられるため、Memory-bound であるアプリケーションの実行時に CPU 周波数を低く設定しても性能低下は小さい。この手法は広く研究されており [3], [4], [5], [6], [7], [8], 多くの研究事例では、実行中のアプリ特性を PMC (Performance Monitoring Counter) によって判定している [3], [4], [5], [6], [7]。しかし、PMC はマシン異常監視 (nmi watchdog) や Vtune などの性能解析ツールといった様々な用途に用いられるため、CPU 周波数制御のためのアプリ特性判定に PMC を用いるとこれらと同時に利用できない場合がある。また、PMC を用いるとアプリ特性判定がソフトウェア環境に依存するため、省電力の運用管理に手間を要する。さらに、PMC を用いるとアプリ特性判定のためにソフトウェア処理が走行し、CPU リソースを消費するが、理想的にはアプリ特性判定に CPU リソースは消費しないことが望ましい。

本論文では、メモリ消費電力に基づく CPU 周波数制御手法を提案する。Memory-bound であるアプリケーションの実行時は、メモリアクセスが多いためメモリ消費電力は高いと考えられる。一方、CPU-bound であるアプリケーションの実行時は、メモリアクセスが少ないためメモリ消

費電力は低いと考えられる。そこでメモリ消費電力に基づき、実行中のアプリ特性を判別する。メモリ消費電力が高い場合には当該アプリケーションは Memory-bound であると判別できると考えられる。そこでこの場合、CPU 周波数を低く設定することで、性能低下を抑え、省電力化を実現する。提案手法はメモリ消費電力のみを指標として利用するため、PMC の用途を制限しない。また、メモリ消費電力はセンサ情報であるため、外部サーバコントローラからメモリ消費電力の取得およびそれに基づく CPU 周波数の制御を行えば、ソフトウェア環境に依存しない実装が可能である。さらにこの場合、アプリ特性判定に CPU リソースを消費しない。

メモリ消費電力に基づくアプリ特性判別が可能かどうかの検証のため、様々なアプリケーションの集合である SPEC CPU2006 ベンチマークを用いた調査を行った。その結果、本論文で定義するアプリ特性の定量的な指標である CPU 周波数依存度と実行時の平均メモリ消費電力の相関係数は -0.93 と高く、メモリ消費電力がアプリ特性判定に有効であることを確認できた。

評価では、NPB ベンチマーク 8 種を用い、5% の性能低下制限のもとに CPU 周波数制御を行い、CPU 消費電力量削減の効果を調査した。評価の結果、平均 3% の性能低下に対し、平均 5% の CPU 消費電力量を削減できた。特に、is.C ベンチマークでは 5% の性能低下に対し、15% の CPU 消費電力量の削減を確認できた。また提案手法は、Last Level Cache Miss イベントを利用した PMC の手法に対し同程度の効果を確認でき、本手法の有効性が確認できた。

本論文の構成は以下のとおり。2 章では、本論文の対象とする課題の説明と従来手法について述べる。3 章では、メモリ消費電力がアプリ特性の判定に有効であることを示す。4 章では、メモリ消費電力に基づく CPU 周波数制御手法を評価する。5 章では、関連研究について述べる。6 章では、まとめと今後の課題について述べる。

2. CPU 周波数制御による消費電力量の削減

本章では、本論文の扱う課題と、従来技術について説明する。

2.1 サーバに占める CPU 消費電力

近年の PC・サーバは、CPU の idle、低負荷時には CPU の省電力技術により、省電力化が達成されている。図 1 は、CPU やメモリなどの電力を個別に取得可能な電力測定ボードを利用し、サーバ消費電力の内訳を調査した結果である。図 1 の左側が idle 時、右側が CPU 使用率 100% の高負荷時の結果である。idle 時と高負荷時を比較すると、idle 時は CPU の消費電力が小さい。一方、高負荷時は CPU 消費電力が大きい。高負荷時に CPU 消費電力が大きくなるのは、高負荷 (CPU 使用率 100%) 時は、CPU 周波数が高

¹ 株式会社富士通研究所
Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211-8588, Japan

a) masahiro.miwa@labs.fujitsu.com

CPU 周波数に設定されるためである。高負荷時には、サーバ全体の消費電力に占める CPU の割合が大きく、サーバ消費電力を占める第 1 要因となっていることが分かる。したがって、CPU 使用率が高い場合の PC・サーバの省電力化を実現するには、CPU の省電力化が必要である。そこで、本論文では高負荷時の CPU 省電力化を対象とする。

2.2 アプリ特性を利用した CPU 周波数制御手法

2.2.1 アプリ特性と CPU 周波数依存度

CPU 使用率が高い場合のアプリ特性は大きく 2 つに分けることができる。それらはそれぞれ CPU-bound, Memory-bound と呼ばれる。CPU-bound であるアプリケーションは、キャッシュミスが少なく、メモリアクセスが少ない。一方、Memory-bound であるアプリケーションは、キャッシュミスが多く、メモリアクセスが多い。Memory-bound であるアプリケーションの実行時は、CPU 周波数を低く設定しても、その性能に与える影響は小さい。この理由は、Memory-bound であるアプリケーションは、メモリ処理速度によってアプリケーションの実行性能は決定され、CPU 処理速度は大きな影響を与えないためである。図 2 は、CPU-bound および Memory-bound であるアプリケーション実行時の CPU 周波数を変更したときのそれぞれの性能である。表 1 に調査に用いた環境を示す。

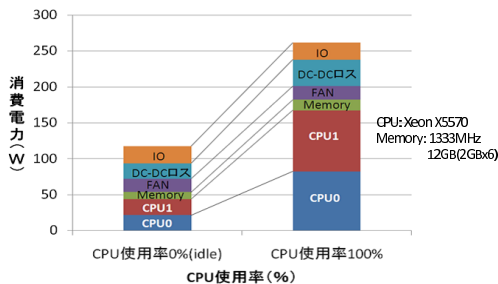


図 1 サーバコンポーネントごとの電力
Fig. 1 Server component power.

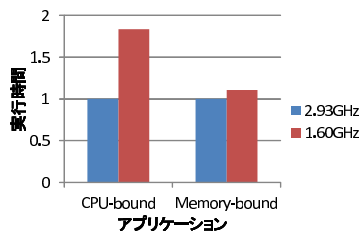


図 2 アプリ特性の違いによる CPU 周波数の性能への影響
Fig. 2 Application performance by different CPU frequency.

実際に、CPU-bound であるアプリケーションは、CPU 周波数の違いによる性能差が大きく、Memory-bound であるアプリケーションは、CPU 周波数の違いによる性能差が小さいことが確認できる。

これまでアプリ特性を CPU-bound, Memory-bound と分類したが、たとえば Memory-bound といっても、その程度は様々である。そこで、アプリ特性を定量的に議論するために、CPU 周波数依存度を導入する。CPU 周波数依存度は、CPU 周波数の変動に対する実行時間の変化の割合であり、式 (1) によって算出できる。具体的には、CPU 周波数依存度は、CPU 周波数を変更したときのクロック周期の増加率に対する、アプリケーションの実行時間の増加率から算出することができる。式 (1) によると、図 2 のアプリケーションでは、CPU-bound アプリケーションでは CPU 周波数依存度 100%、Memory-bound アプリケーションでは CPU 周波数依存度 13%となる (表 2, 表 3)。

$$(CPU \text{ 周波数依存度}) = \frac{(\text{アプリの実行時間の増加率})}{(\text{クロック周期の増加率})} \quad (1)$$

2.2.2 CPU 周波数の制御

一般に CPU 周波数を低く設定すると CPU 消費電力は削減される。特に、Memory-bound であるアプリケーションの実行時は、CPU 周波数を低く設定することによる性能低下が小さく、実行時間はほとんど増加しないため、CPU 消費電力量も削減される。一方、CPU-bound であるアプリケーションの実行時に CPU 周波数を低く設定すると実行時間が増加するため、CPU 消費電力量は削減できるとは限らない。さらに、CPU 以外のコンポーネントの消費電力量は、実行時間の増加に比例する。したがって、消費電力量を削減するには、性能低下率を一定以内に抑制するよう CPU 周波数の制御を行う必要がある。

ここでは、アプリケーションの CPU 周波数依存度より、性能低下率を一定以内に抑制しつつ、下限となる CPU 周波数を設定する制御方法を文献 [3] をもとに説明する。

最高周波数 f_{max} におけるアプリケーションの実行時間

表 1 評価環境

Table 1 Evaluation environment.

CPU	Intel Xeon X5570 @2.93 GHz
Memory	DDR3 1333 MHz 12 GB (4 GBx3)
OS	CentOS 5.7

表 2 実行時間

Table 2 Execution time.

	クロック周期 (ns)	CPU-bound (sec)	Memory-bound (sec)
2.93 GHz	0.3417	33.36	85.67
1.60 GHz	0.6265	61.19	94.98
実行時間増加率 (%)	83.35	83.42	10.87

表 3 CPU 周波数依存度
Table 3 CPU frequency dependency.

	CPU-bound	Memory-bound
CPU 周波数依存度 (%)	100	13

は, CPU 周波数に性能が依存する時間 ($T_{f_{max}cpu}$) と CPU 周波数に性能が依存しない時間 ($T_{f_{max}mem}$) に分けられる (式 (2)).

$$T_{f_{max}} = T_{f_{max}cpu} + T_{f_{max}mem} \quad (2)$$

CPU 周波数を f_{max} から f に変更したときの実行時間は式 (3) のとおり表される.

$$T_f = \frac{f_{max}}{f} T_{f_{max}cpu} + T_{f_{max}mem} \quad (3)$$

したがって, CPU 周波数を f_{max} から f に変更したときの実行時間の増分は, 式 (5) のとおり表される.

$$T_f - T_{f_{max}} = \frac{f_{max}}{f} T_{f_{max}cpu} - T_{f_{max}cpu} \quad (4)$$

$$= \left(\frac{f_{max}}{f} - 1 \right) T_{f_{max}cpu} \quad (5)$$

CPU 周波数依存の時間 $T_{f_{max}cpu}$ は, CPU 周波数が f_{max} の場合の全実行時間 $T_{f_{max}}$ に CPU 周波数依存度 (d) を掛けることで求められ, 実行時間の増分は式 (6) で表すことができる.

$$T_f - T_{f_{max}} = \left(\frac{f_{max}}{f} - 1 \right) T_{f_{max}} * d \quad (6)$$

一方, 性能の低下率 (PF_{Loss}) は, 式 (7) で表される.

$$PF_{Loss} = \frac{T_f - T_{f_{max}}}{T_{f_{max}}} \quad (7)$$

式 (6), (7) より, CPU 周波数 (f) は式 (8) で表される.

$$f = \frac{f_{max}}{PF_{Loss}/d + 1} \quad (8)$$

ここで, PF_{Loss} を性能低下率の上限と考えると, CPU 周波数 f は, CPU 周波数 f_{max} の場合に対し性能低下率の上限を満たす CPU 周波数となる.

2.3 従来技術とその課題

2つの異なる CPU 周波数でアプリケーションを事前に実行し, アプリケーションの CPU 周波数依存度を得る手法が考えられる. その場合, アプリケーションごとの個別分析を事前に必要とするため, 新たなアプリケーションへの対応に手間を要する. 事前解析なしにアプリ特性を判定する従来手法としては PMC を利用するものが多く [3], [4], [5], [6], [7], その多くは PMC の Last Level Cache (LLC) Miss の回数に基づきアプリ特性の判定を行う [4], [5], [7]. LLC Miss の発生は, メモリアクセスの発

生へつながる. したがって, LLC Miss の回数が多ければ Memory-bound, 反対に少なければ CPU-bound であると判定することができる. しかし, PMC を用いる従来技術には以下の課題がある.

- PMC を占有する

PMC はマシン異常監視 (nmi watchdog), プロファイラなど多様な用途に利用される. PMC を CPU 周波数制御に利用するとこれらの用途に使用できない.

- ソフトウェア環境に依存する

PMC による手法では, OS・ハイパーバイザといった環境ごとの設定を要するため運用管理面で手間を要する.

- CPU 周波数制御のオーバーヘッドが発生する

アプリ特性の判定は極力オーバーヘッドなしに実現できることが望ましい. しかし, PMC の手法では, PMC の取得およびアプリ特性の判別に CPU リソースが必要となる.

3. メモリ消費電力に基づく CPU 周波数依存度の算出

メモリ消費電力に基づく CPU 周波数制御の考え方について述べる. 実行中のアプリケーションが Memory-bound である場合, メモリアクセスが多いため, メモリ消費電力は高くなると考えられる. 一方, CPU-bound であるアプリケーションではメモリアクセスが少ないため, メモリ消費電力は低くなると考えられる. つまり, 反対にメモリ消費電力から, 実行中のアプリケーションが Memory-bound か CPU-bound かの特性を判定できる可能性が高い.

メモリ消費電力に基づくアプリ特性判定手法の実現可能性を調査するために, CPU 周波数依存度とメモリ消費電力の相関を調査する. 調査を行った環境を表 1 に示す. 調査用ベンチマークとして, SPEC CPU2006 ベンチマーク [9] を用いる. SPEC CPU2006 は, 整数演算系アプリケーション (INT), 浮動小数点演算系アプリケーション (FP) からなるベンチマークで, 実際的なアプリケーションの集合である. コンパイラは Intel Compiler 12 を使い, SPEC CPU2006 のワーキングセットは, train サイズを選択した. また, メモリアクセスの負荷を様々に変えるため, 各ベンチマークの多重度は 1 から 4 まで変更している. メモリ消費電力はアプリケーション実行時の平均消費電力である.

メモリ消費電力と CPU 周波数依存度の調査結果を図 3 に示す. 横軸はメモリ消費電力であり, 縦軸は CPU 周波数依存度である. メモリ消費電力と CPU 周波数依存度の相関をとると, -0.93 と高い負の相関係数を持つ. この場合, メモリ消費電力が高いほど, CPU 周波数依存度は低く, メモリ消費電力が小さいほど, CPU 周波数依存度が高いことを意味する. 相関係数が高いため, メモリ消費電力から高い精度で CPU 周波数依存度が推定できる. メモリ

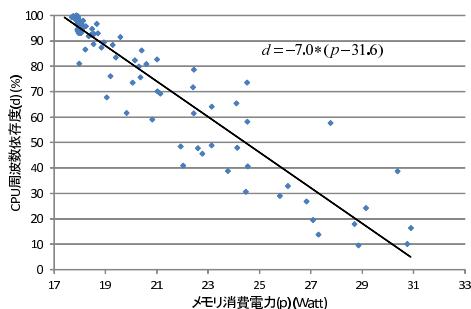


図 3 メモリ消費電力 (p) と CPU 周波数依存度 (d) の相関
Fig. 3 Correlation of memory power (p) and CPU frequency dependency (d).

消費電力から CPU 周波数依存度を得る式は、回帰分析により容易に求めることができる (図 3 中の式). そこで本手法では、CPU 周波数依存度を求めるためのモデル式を SPEC CPU2006 ベンチマークの調査結果により、機械的に生成した. このように本手法ではメモリ消費電力に基づいてアプリ特性を判別する.

メモリ消費電力に基づく CPU 周波数制御は、以下の特徴を持つ.

- PMC を利用しない
 PMC をマシン異常監視 (nmi watchdog) やプロファイラといった用途に用いつつ、省電力制御を実現できる.
- ソフトウェア環境に依存しない
 メモリ消費電力というセンサデータを利用すれば、BMC (Baseboard Management Controller) といった外部サーバコントローラによりアプリ特性判定が実現可能である. この場合、OS・ハイパーバイザといったソフトウェア環境に依存しないため、省電力のための運用管理コストを削減することができる.
- オーバヘッドなくアプリ特性を判定できる
 外部サーバコントローラを利用して実現すれば、ターゲットホスト上で、CPU 周波数制御のためのアプリ特性判定のソフトウェア処理をなくすことができる.

4. 提案手法の評価

本章では、推定した CPU 周波数依存度に基づいて CPU 周波数を決定する方法と、これを実装し、NPB ベンチマークで評価した結果を述べる.

4.1 実装

4.1.1 設定する CPU 周波数の算出

図 3 中の回帰式を利用し、メモリ消費電力から CPU 周波数依存度を得、得られた CPU 周波数依存度および性能低下率の上限を式 (8) に与えることで、性能低下率の上限を満たす CPU 周波数を得ることができる.

4.1.2 CPU 周波数制御のパラメータ

CPU 周波数の制御は、1 秒および 100 ミリ秒間隔で行

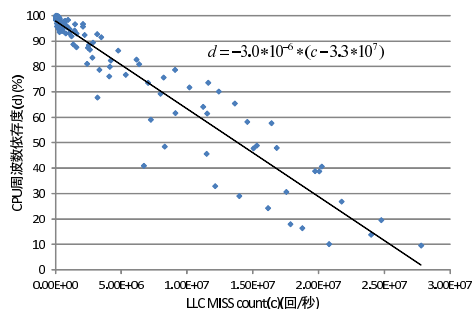


図 4 LLC Miss 回数 (c) と CPU 周波数依存度 (d) の相関
Fig. 4 Correlation of LLC Miss counts (c) and CPU frequency dependency (d).

う. これにより、制御間隔の違いによる CPU 周波数制御の効果の確認を行う. また、評価環境で設定可能な CPU 周波数は、ベース周波数 133 MHz の通倍である 1.60 から 2.93 GHz の間の 11 通りであるため、設定する CPU 周波数は、式 (8) より算出した CPU 周波数 f より大きい設定可能な周波数のうち、最小の CPU 周波数を選択する.

4.2 評価方法

4.2.1 評価項目

性能低下率の上限は 5 (%) とし、性能の低下率および CPU 消費電力量の削減を調査する. 評価には、NAS Parallel Benchmark (NPB) (version 3.2, OpenMP 版) を用いた [10]. NPB から 8 種類のベンチマークを評価に利用し、問題サイズは C サイズ、実行スレッド数は 4 スレッドとしている. コンパイラは、Intel Compiler 12 を用いた.

4.2.2 PMC 手法との比較

従来手法である PMC による手法に対する比較評価を行う. 従来手法の多くは、LLC Miss の回数に基づき、アプリ特性判定を行う [4], [5], [7]. そこで、Nehalem アーキテクチャが有する Uncore の PMC を利用し、UNC_L3.MISS.ANY (EventNum 09H, Umask Value: 03H) イベントを採用した [11]. モデル生成のために、PMC のカウンタ値と CPU 周波数依存度の相関調査を行った. 調査には、メモリ消費電力モデル作成時と同様、SPEC CPU2006 の train セットを用いる.

調査の結果を図 4 に示す. 横軸が LLC Miss の回数であり、縦軸が CPU 周波数依存度である. LLC Miss の回数が多いほど、CPU 周波数依存度が低い. LLC Miss の回数と CPU 周波数依存度との相関係数は -0.95 である. LLC Miss の回数を利用したモデルも回帰分析により容易に得られ、得られた式 (図 4 中の式) に基づき、アプリ特性判定を行う.

CPU 周波数の決定方法および制御アルゴリズムは、メモリ消費電力を利用する場合と同様であり、CPU 周波数依存度を算出するモデル式のみが異なる.



図 5 電力測定ボード

Fig. 5 Power measurable board.

4.3 評価環境

評価環境を表 1 に示す。表 1 の評価環境は、微小抵抗が各コンポーネントの電源ラインに挿入されている電力測定ボードである。このボードを利用し、メモリ消費電力の測定を行う。メモリ消費電力は、3 枚の DIMM モジュールおよび VTT の消費電力の和である。図 5 に電力測定ボードを示す。メモリモジュールの右隣など、ボード上に接続されているまだらの細いケーブルが電力測定に利用されている。電力測定器は、NI USB-6210 (National Instrument 社) を用い、制御には nidaqmxbase-3.4.5 を利用している。電力取得の間隔は 10 msec ごとである。本論文の評価では、実装上の都合、メモリ消費電力はターゲットホスト上から取得しているため、提案手法においても CPU リソースを消費している。

現在はメモリ消費電力の測定が可能な特別な環境を利用しているが、将来的には、CPU やメモリなどの主要コンポーネントは標準的な環境でも個別に消費電力測定が可能になると我々は考えている。

4.4 評価結果

各種制御手法のデフォルト設定 (2.93 GHz) 時に対する CPU 消費電力量の削減率と実行時間の増加率を図 6, 図 7, 図 8, 図 9 に示す。結果は、各制御手法と CPU 周波数の制御間隔ごとに分けている。グラフの横軸は CPU 消費電力量の増加率で、縦軸が実行時間の増加率 (性能低下率) である。実行時間の増加率は、各々のベンチマークにおいて、メモリ消費電力に基づく CPU 周波数制御, PMC に基づく CPU 周波数制御ともにおおむね 5% の性能低下以下に抑えられている。メモリ消費電力および PMC に基づく手法のどちらにおいても、制御間隔による大きな違いはほとんどみられなかった。メモリ消費電力に基づく CPU 周波数制御の結果、CPU 周波数の制御間隔 1 秒, 100 ミリ秒のともに同程度の CPU 消費電力量削減効果であり、平均 5% CPU 消費電力量を削減できた。特に、is.C ベンチマークでは 5% の性能低下で、15% の CPU 消費電力量の削減を確認できた。LLC Miss イベントを利用した PMC に基づく CPU 周波数制御の結果は、おおむねメモリ消費電力に基づく CPU 周波数制御結果と傾向が似ている。ただし、PMC に基づく CPU 周波数制御では、is.C ベンチマークで

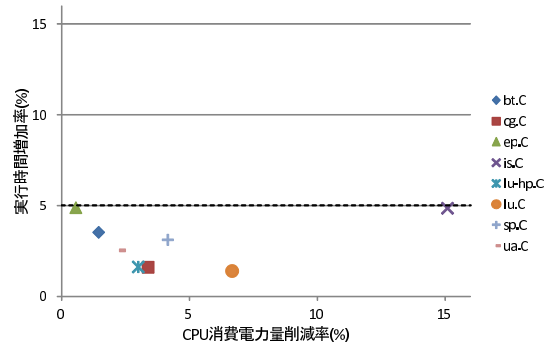


図 6 メモリ消費電力に基づく CPU 周波数制御 (1 sec)

Fig. 6 CPU frequency control based on memory power (1 sec).

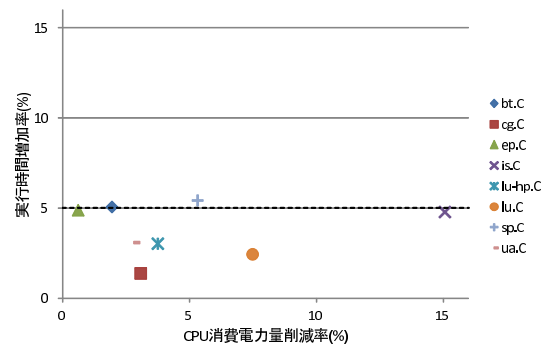


図 7 メモリ消費電力に基づく CPU 周波数制御 (100 msec)

Fig. 7 CPU frequency control based on memory power (100 msec).

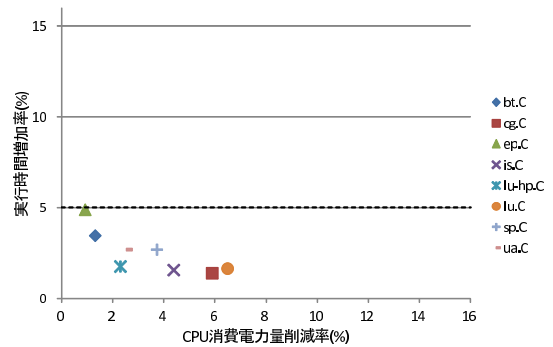


図 8 PMC に基づく CPU 周波数制御 (1 sec)

Fig. 8 CPU frequency control based on PMC (1 sec).

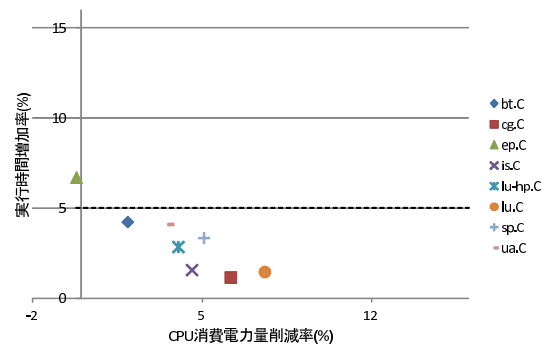


図 9 PMC に基づく CPU 周波数制御 (100 msec)

Fig. 9 CPU frequency control based on PMC (100 msec).

表 4 目標の性能低下率と実際の性能低下率 (メモリ消費電力)

Table 4 Performance degradation (target and real) (memory power).

	memory 1 sec			memory 100 msec		
	目標 (%)	実際 (%)	(目標-実際)	目標 (%)	実際 (%)	(目標-実際)
bt.C	3.8	3.5	0.3	4.4	5.1	-0.6
cg.C	3.3	1.6	1.7	3.3	1.4	1.9
ep.C	4.8	4.9	-0.1	4.7	4.9	-0.2
is.C	2.0	4.9	-2.9	1.9	4.8	-2.9
lu-hp.C	4.3	1.6	2.7	3.8	3.0	0.7
lu.C	3.9	1.4	2.5	4.0	2.4	1.6
sp.C	4.1	3.1	0.9	3.9	5.4	-1.5
ua.C	2.7	2.5	0.2	3.7	3.1	0.6

表 5 目標の性能低下率と実際の性能低下率 (PMC)

Table 5 Performance degradation (target and real) (PMC).

	pmc 1 sec			pmc 100 msec		
	目標 (%)	実際 (%)	(目標-実際)	目標 (%)	実際 (%)	(目標-実際)
bt.C	3.8	3.5	0.3	4.3	4.2	0.1
cg.C	4.8	1.4	3.4	4.9	1.2	3.7
ep.C	4.7	4.9	-0.2	4.5	6.7	-2.2
is.C	4.7	1.6	3.2	4.6	1.6	3.0
lu-hp.C	2.5	1.8	0.8	4.1	2.8	1.3
lu.C	4.1	1.7	2.4	4.3	1.5	2.8
sp.C	3.7	2.7	1.0	4.0	3.3	0.6
ua.C	2.8	2.7	0.0	4.1	4.1	0.0

性能低下が小さい分, CPU 消費電力量削減効果も小さい. 以上の結果から, メモリ消費電力に基づくアプリ特性判定手法の有効性を確認した.

4.5 考察

4.5.1 性能低下率のばらつき

評価では性能低下率の上限を 5% に設定したが, 実際の性能低下率は, 制御手法やベンチマークによってばらつきがある. これには以下の 2 つの要因が考えられる.

(1) 設定可能な CPU 周波数の制限

CPU 周波数は, ベース周波数 (133 MHz) を通倍したものである. したがって, 式 (8) より算出した CPU 周波数ちょうどには, CPU 周波数を設定できない場合がある. この場合, 式 (8) より算出された CPU 周波数より高い設定可能な CPU 周波数のうち, 最も低い CPU 周波数を設定している. このようにすると, 選択した CPU 周波数による性能低下の割合は, 指定したものより低くなる.

(2) CPU 周波数依存度の推定ミス

図 3 中の式や図 4 中の式のモデルから外れるために, 性能低下が大きくまたは小さくなってしまふことがある. たとえば CPU 周波数依存度が高い (CPU-bound) と判定し, 性能低下を制限内に抑えるために CPU 周波数はあまり低くしなかったが, 実際は Memory-bound であり, 性能がほとんど低下しなかった, などのケースが考えられる. 以下では, 後者の CPU 周波数依存度の判定ミスを明らかにす

るため, 各種制御手法で, 実際に設定する CPU 周波数をもとに決まるアプリケーションの性能低下率 (目標) と実際の性能低下率 (実際) を調査し, アプリ特性判定の精度を評価する. 表 4, 表 5 にその結果を示す. 目標とする性能低下率と実際の性能低下率が各列に対応しており, 各行はベンチマーク名である. 調査の結果, メモリ消費電力に基づく CPU 周波数制御, PMC に基づく CPU 周波数制御のともに, おおむね半数のベンチマークで, ほぼ目標どおりに制御できたといえる. ただし, 一部目標の性能低下率と実際の性能低下率の差が大きいものがあるため, それらの改善に取り組む必要がある.

4.5.2 PMC 取得のオーバーヘッド

提案手法を外部サーバコントローラにより実現すれば, 既存手法である PMC の手法での PMC 取得のためのオーバーヘッドをなくすことが可能である. そこで, PMC 取得にかかるオーバーヘッドの調査を表 1 の環境で行った. その結果, PMC の取得 1 回あたり, 約 0.4 μ sec のコストであることを確認した. このコストは単体ノードでのオーバーヘッドとして考えるとほとんど無視できるほど小さい. たとえば, 仮に 1 秒間隔でアプリ特性を判定のため PMC 取得を行う場合, 単体ノードでみるとオーバーヘッドはほとんど影響しない. しかし, 数 1,000 ノードを超える大規模 PC クラスタ環境では, このオーバーヘッドによりいわゆる OS ノイズ問題が生じ, 性能に影響を与える可能性がある [12]. 各ノードの PMC 取得間隔が 1 秒間隔と低頻度であっても,

ある時間帯において全ノードのうちいずれかのノードで PMC 取得を行っている確率はノード数にほぼ比例して高くなる。複数ノードが協調して動作する MPI プログラムでは、ノード間でバリア同期をとる際に 1 ノードでの遅れが全体に伝搬する。特に大規模並列環境では、アプリケーションの並列度増加に従って、同期処理の周期がきわめて短くなる。仮に同期周期が $10 \mu\text{sec}$ であれば、 $0.4 \mu\text{sec}$ のオーバーヘッドにより、約 4% の性能低下が生じる。このような fine-grained なアプリケーションでは、計算と通信の時間粒度が短く高頻度にノード間で同期をとるため、オーバーヘッドのシステム性能に対する影響は増大する。したがって、オーバーヘッドはできる限りなくすことが望ましく、提案手法のハードウェア実装時のオーバーヘッド削減効果は大きいと考えている。

5. 関連研究

DVFS を利用した省電力化手法は広く研究されている。

Ondemand Governor [2] は、CPU 使用率に応じて CPU 周波数を制御する。システムの使用率が低い場合には CPU 周波数を低く設定することで、システムパフォーマンスへの影響を抑え、省電力化する。本論文では、CPU 使用率が高い場合を対象に省電力化を狙う。

本論文と同様、CPU 使用率が高い場合に CPU 周波数を低く設定することで、アプリケーションの実行に必要な消費電力量の削減を行う研究は多く行われている。文献 [4] では、実行中のアプリ特性の判定に PMC の LLC Miss 回数イベントを利用し、消費電力量を削減する CPU 周波数制御を実現している。また、PMC のその他イベントを利用し、アプリ特性の判定を行うものもある [3], [6]。しかし、PMC はマシン異常監視 (nmi watchdog) や Vtune などのプロファイラツールで用いられるため、PMC をアプリ特性判定に用いるとその用途が限定されてしまう。本論文は、アプリ特性判定にメモリ消費電力を利用する。メモリ消費電力はセンサ情報であるため、外部サーバコントローラによる制御が可能であり、この場合、ターゲットホストはアプリ特性判定に CPU リソースを消費しない。他には、組込みプロセッサ向けにアプリ特性判定のための追加のハードウェアを用いる手法が存在する [8]。この方式は、プロセッサアーキテクチャの変更が必要となるが、提案手法は、メモリ消費電力を利用するため、プロセッサアーキテクチャの変更を必要としない。さらに、文献 [5] では PMC による手法に加え、未解決のキャッシュミス要求を格納する MSHR (Miss Status Holding Register) を用いる手法を評価している。しかし、MSHR の利用はプロセッサの実装依存となる所が大きい。

上記は、実行時情報に基づく手法であり、オンラインの手法と呼ばれる。一方、事前解析を行うオフラインの手法も存在する。オフラインの手法は、事前にアプリケーシ

ョンの特徴を分析し、あらかじめ実行時の CPU 周波数を決定する手法である。佐々木ら [13] は、ソースコードを複数領域に分割し、PMC を利用し領域ごとに設定する CPU 周波数を決定する。続いて、決定した CPU 周波数を設定するための命令を各領域の先頭に挿入することで、アプリケーション実行時の CPU 周波数を制御する。オフラインの手法は、アプリケーションごとの事前解析の手間を要する。本手法はオンラインの手法であり、1 度モデルを作成すれば、アプリケーションごとの事前解析は必要としない。

6. おわりに

本論文では、メモリ消費電力に基づく CPU 周波数動的制御手法の評価を行った。提案手法では、メモリ消費電力が CPU 周波数依存度と高い相関係数を持つことから、メモリ消費電力に基づきアプリ特性を判定し、CPU 周波数を制御した。この手法により、CPU 使用率が高い場合に性能低下を抑えつつ、低 CPU 周波数に設定する制御をすることで CPU 消費電力量を削減した。従来の PMC に基づく手法における PMC の用途を制限してしまう問題や、アプリ特性判定のためにオーバーヘッドが発生するという課題を本手法により解決することができる。

評価では、NPB ベンチマーク 8 種を用い、5% の性能低下制限のもとにメモリ消費電力に基づく CPU 周波数制御を行い、CPU 消費電力量削減の効果を調査した。評価の結果、平均 3% の性能低下に対し、平均 5% の CPU 消費電力量を削減できた。特に、is.C ベンチマークでは 5% の性能低下に対し、15% の CPU 消費電力量の削減を確認できた。また提案手法は、Last Level Cache Miss イベントを利用した PMC の手法に対し同程度の効果を確認でき、本手法の有効性が確認できた。

今後の課題には、一部のベンチマークで目標の性能低下率と実際の性能低下率に差が大きくなった原因の調査と改善がある。また、提案手法をハードウェア実装した際の設計コストに関する検討がある。

参考文献

- [1] Intel White Paper, Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (2004).
- [2] Pallipadi, V. and Starikovskiy, A.: The Ondemand Governor: Past, Present, and Future, *The Linux Symposium* (2006).
- [3] Huang, S. and Feng, W.: Energy-Efficient Cluster Computing via Accurate Workload Characterization, *Proc. 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, ser. CCGRID'09*, pp.68-75, Washington, DC, USA, IEEE Computer Society (2009).
- [4] Schone, R. and Hackenberg, D.: On-line analysis of hardware performance events for workload characterization and processor frequency scaling decisions, *Proc. 2nd joint WOSP/SIPEW international conference on Performance engineering, ICPE'11*, pp.481-486, ACM

- (2011).
- [5] 近藤正章, 中村 宏: 主記憶アクセスの負荷情報を利用した動的周波数変更による低消費電力化, 情報処理学会論文誌: コンピューティングシステム, Vol.45, No.SIG 6 (ACS 6), pp.1-11 (2004).
 - [6] Kotla, R., Ghiasi, S., Keller, T. and Rawson, F.: Scheduling Processor Voltage and Frequency in Server and Cluster Systems, *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, pp.234-241 (Apr. 2005).
 - [7] Amur, H., Schwan, K. and Prvulovic, M.: Towards Optimal Power Management: Estimation of Performance Degradation due to DVFS on Modern Processors, Tech. Report GIT-CERCS-10-02.
 - [8] Stanley-Marbell, P., Hsiao, M. and Kremer, U.: A Hardware Architecture for Dynamic Performance and Energy Adaptation, *Power-Aware Computer Systems*, Lecture Notes in Computer Science 2325, Springer-Verlag (2002).
 - [9] SPEC CPU2006, available from (<http://www.spec.org/cpu2006/>).
 - [10] Nas Parallel Benchmark, available from (<http://www.nas.nasa.gov/>).
 - [11] Intel: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part2 (2010).
 - [12] Tsafrir, D., Etsion, Y., Feitelson, D.G. and Kirkpatrick, S.: System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications, *ACM International Conference on Supercomputing*, Cambridge, Massachusetts (June 2005).
 - [13] 佐々木広, 浅井雅司, 池田佳路, 近藤正章, 中村 宏: 統計情報に基づく動的電源電圧制御手法, 情報処理学会論文誌: コンピューティングシステム, Vol.47, No.Sig18 (ACS 16), pp.80-91 (2006).



三輪 真弘 (正会員)

2009年早稲田大学大学院情報理工学専攻修了。同年富士通(株)入社。現在,(株)富士通研究所にて省電力,HPCに関する研究・開発に従事。



中島 耕太 (正会員)

2000年九州大学工学部電気情報工学科卒業。2002年同大学大学院システム情報科学府情報工学専攻修士課程修了。同年富士通(株)入社。博士(工学)。現在,(株)富士通研究所勤務。高速通信機構に関する研究開発に従事。HPCシステム,システムソフトウェアに興味を持つ。SACSI2010 最優秀論文賞受賞。



平井 聡

(株)富士通研究所勤務。1997年より同研究所にてIAサーバシステムの性能向上技術,運用管理技術の研究に従事。



風間 哲

1997年東北大学大学院情報科学研究科情報基礎科学専攻修士課程修了。同年富士通(株)に入社。現在,(株)富士通研究所にてサーバ・ストレージの研究開発に従事。



原 靖

1988年新潟大学工学部情報工学科卒業。同年富士通(株)に入社,現在,(株)富士通研究所にて,端末の省電力技術の研究開発に従事。



成瀬 彰 (正会員)

1996年名古屋大学大学院工学研究科修了(情報工学専攻)。同年富士通(株)入社。(株)富士通研究所にてIAサーバ,PCクラスタシステムに関わる研究開発に従事。計算機アーキテクチャ,並列処理,性能最適化に興味を持つ。平成21年度情報処理学会山下記念研究賞受賞。SACSI2010 最優秀論文賞受賞。