Vol.2011-HPC-130 No.24
2011/7/28

# メモリ消費電力に基づく CPU 周波数の動的制御

三輪 真弘[†]　中島 耕太[†]　平井 聡[†]　成瀬 彰[†]

本稿では、メモリ消費電力に基づく動的な CPU 周波数制御手法について述べる。近年のプロセッサには DVFS と呼ばれる CPU 周波数および電圧を変更可能な仕組みが実装されている。CPU 使用率 100%の場合に DVFS を利用し省電力化する先行研究では、アプリケーションの性能が CPU 周波数に依存する CPU 依存型か CPU 周波数に依存しないメモリ依存型かを、パフォーマンスカウンタ(PMC)により判定し、メモリ依存型のアプリケーションの実行時に CPU 周波数を低く設定する。メモリ依存型では CPU 周波数を低く設定しても性能低下が小さいため、高性能と省電力が両立する。しかし、PMC は性能解析など様々な用途で利用されるため、CPU 周波数制御に PMC を利用すると、PMC の用途が限定される問題がある。

本稿では、CPU 依存型のアプリケーションの実行時にメモリ消費電力が小さく、メモリ依存型のアプリケーションの実行時にメモリ消費電力が大きいことを実験により明らかにし、メモリの消費電力に基づき CPU 周波数を制御する手法を提案する。提案手法の NPB による実機上での評価では、8 個中 6 個のベンチマークにおいて概ね目標通り性能の制御を行うことができた。また、5%の性能低下を許容する条件で CPU 周波数を制御した場合に、lu ベンチマークでは、3%の性能低下に対し、最大 9%消費電力量を削減できた。本手法は、PMC を利用せず、センサーから取得するデータを利用しており、BIOS レベルの実現も可能である。

# Dynamic Voltage and Frequency Scaling Adaptation based on Memory Power

Masahiro Miwa[†]　Kohta Nakashima[†]　Akira Hirai[†]
Akira Naruse[†]

This paper presents a novel approach of DVFS adaptation based on memory power consumption. We find that memory power strongly correlates with CPU Frequency Dpendency (the degree of application performance affected by CPU Frequency). CPU-bound application consumes low memory power and Memory-bound application consumes high memory power. Our method does not need to use the Performance Monitoring Counter that most of prior works use it to decide CPU frequency scaling and our method use data acquired by sensor and could be implemented on BIOS. Experiments result using NPB shows that for lu benchmark (memory-bound), 9% energy saving was achieved under 3% performance loss.

## 1. はじめに

省電力化に関する取り組みは積極的に行われており、モバイル機器のみならず、デスクトップ PC、サーバにおいても省電力化の研究開発はさかんである。近年のデスクトップ PC、サーバ向けのプロセッサには DVFS (Dynamic Voltage and Frequency Scaling)と呼ばれる CPU の電圧および周波数を変更する機構がある。DVFS 機能は、Intel 社製プロセッサでは SpeedStep、AMD 社製プロセッサでは、PowerNow!として利用できる。

DVFS を活用した例として有名なものは、Linux システムの ondemand governor[10]がある。ondemand governor は、CPU 使用率に応じて、CPU 周波数を設定する仕組みである。システムの CPU 使用率が低い場合には CPU 周波数が低く設定されることで、システムの性能への影響は与えず、省電力化する。一方、CPU 使用率 100%の場合にも DVFS を活用して消費電力量を削減する先行研究が存在する。先行研究では、実行中のアプリケーションを CPU 周波数に性能が依存する CPU 依存型と、CPU 周波数に性能が依存しないメモリ依存型に分類し、アプリケーションがメモリ依存型の場合には、CPU 周波数を低く設定する。メモリ依存型のアプリケーションには、CPU 周波数を低く設定しても、アプリケーション性能がそれほど低下しないという特徴があり、これにより高性能と省電力を両立している。先行研究は、パフォーマンスモニタリングカウンタ（PMC）を利用してアプリケーションを CPU 依存型、メモリ依存型に分類するものが多い。しかし、PMC は、VTune 等の性能解析ツールなど様々な用途で用いられており、CPU 周波数制御に PMC を利用すると、PMC の用途が限定されるという問題がある。

本稿では、メモリ消費電力に基づく CPU 周波数制御手法を提案する。CPU やメモリの消費電力を個別に測定可能な環境を用いた調査実験により、CPU 依存型のアプリケーションはメモリ消費電力が小さく、メモリ依存型のアプリケーションは、メモリの消費電力が大きいことを明らかにする。メモリ消費電力が小さいときは CPU 周波数を高く、メモリ消費電力が大きいときは CPU 周波数を低く設定することで、システムの消費電力量を削減する。本手法は、センサーより取得するデータを利用しており、原理的には BIOS レベルの実装も可能であり、その場合 OS に非依存に本手法は適用可能である。

提案手法を NAS Parallel Benchmark(NPB) を用い、実機上で評価をした。評価の結果、8 個中 6 個のベンチマークにおいて概ね目標通りの性能の制御を行うことができた。また、5%の性能低下を許容する条件で CPU 周波数を制御した場合に、lu ベンチ

[†] （株）富士通研究所
Fujitsu Laboratories ltd.

マークでは、3%の性能低下に対し、9%の消費電力量削減を確認できた。

## 2. 関連研究

DVFS 機能を利用した省電力化手法は数多く存在する。ここではアプリケーションの負荷に応じて CPU 周波数を制御する関連研究を説明する。

### 2.1 オフラインの手法

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[8]の手法では、まずソースコードを領域単位に分割し、PMC を利用して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 周波数を制御し、消費電力量の削減を実現している。ただし、アプリケーション毎の事前解析が必要であり、手間を要する。

### 2.2 オンラインの手法

オンラインの手法は、実行時情報に基づく CPU 周波数制御手法であり事前解析を必要としない。

Linux の ondemand governor[10]は、CPU 使用率に応じて、CPU 周波数を設定する仕組みである。システムの CPU 使用率が低い場合には、CPU 周波数が低く設定されることで、システムの性能に影響を与えず、省電力化する。

CPU 使用率が高いときに、CPU 周波数を制御して電力を削減する手法も多く存在する。近藤ら[5]は、PMC を利用して演算処理と主記憶アクセスの負荷バランスを明らかにし、主記憶アクセスの多い場合には CPU 周波数を低く設定することで、性能へのペナルティを抑え CPU 消費電力量の削減が可能であることをシミュレーションにより明らかにしている。K.Choi ら[1]は、組み込み向けプロセッサを対象に、オンチップアクセス時間とオフチップアクセス時間をそれぞれ PMC により予測し、周波数を低く設定するかを判断する。また、[1]を基に汎用向けプロセッサを対象にアウトオブオーダプロセッサにおけるキャッシュミスのオーバーラップの効果を考慮したもの[4]も存在する。他には、電力供給が不足する場合に備え、電力をパワーバジェットの制限内に抑えるように CPU 周波数の制御を行う取り組み[2]もある。また、プログラムのフェーズを考慮した細粒度の手法[7]も存在する。これらは、PMC を必要とするが、PMC は性能解析など他の用途で利用されることが多いため、PMC を CPU 周波数制御に用いると用途が限定されるという問題がある。さらに、専用ハードウェアによる DVFS の制御手法[6]も存在する。[6]では、メモリの挙動を分析する専用のハードウェアを利用し、電力の削減を実現している。しかし、CPU 周波数制御のために専用のハードウェアを準備しなくてはならない。

本稿で提案する手法は、オンライン手法の一種であり、メモリ消費電力に基づいて CPU 周波数を制御するので、PMC を必要としない。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。

## 3. アプリケーションのCPU周波数依存度

アプリケーションは CPU 周波数の変化によって性能が大きく変化するものと変化しないものがある。それらはそれぞれ CPU 依存型のアプリケーションとメモリ依存型のアプリケーションと分類することができる。CPU 依存型の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、メモリ依存型の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。メモリアクセスの多いメモリ依存型のアプリケーションの実行時には、CPU 周波数を低く設定しても、アプリケーション性能への影響が小さいことが知られている[1]。

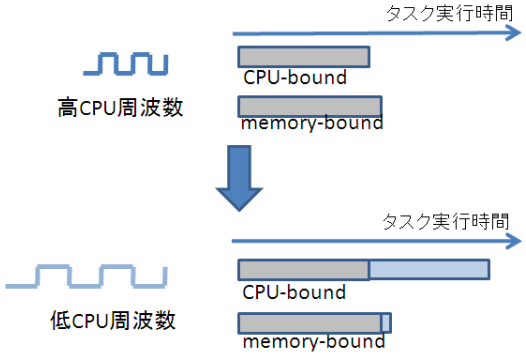


図 1. アプリケーション特性の違いによる DVFS 適用の影響

図 1 は、CPU 依存型、メモリ依存型のアプリケーションの実行時の CPU 周波数を低く設定したときの、実行時間の変化の様子をあらわしている。一律の CPU 周波数の低下に対し、CPU 依存型では大幅に性能が低下するのに対して、メモリ依存型では性能低下が小さい。この場合、一律の CPU 周波数の低下に対し、アプリケーション毎に性能という観点では公平でない。性能の公平性を保つためには、アプリケーションの特性に応じて、CPU 周波数を設定の仕方を変更する必要がある。そのために、実行中のアプリケーションがどの程度 CPU 周波数に依存するかを明らかにする手法が必要である。

本稿では、アプリケーションが CPU 周波数にどの程度依存するかを CPU 周波数依存度として定義する。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（図 2）。

$$\text{CPU周波数依存度(d)} = \frac{(\text{アプリケーションの実行時間の増加率})}{(\text{クロック周期の増加率})}$$

図 2. CPU 周波数依存度算出式

アプリケーションの CPU 周波数依存度の違いによって、一定の性能低下率に対してそれぞれ CPU 周波数をどこまで低く設定してよいかが異なることを説明する。図 3 は、CPU 周波数依存度の異なるアプリケーションについて、CPU 周波数を変更したときの CPU 周波数依存度の異なるアプリケーションの実行時間を示している。1GHz 時（クロック周期は 1ns）のときのアプリケーションの実行時間を基準にすると、CPU 周波数を 500MHz（クロック周期は 2ns）に設定した場合、CPU 周波数依存度 100%のアプリケーションでは、アプリケーションの実行時間は 2 倍になる。また、CPU 周波数依存度 0%のアプリケーションでは、クロック周期によらず一定である。CPU 周波数依存度が 50%であれば、クロック周期が 2 倍になるとアプリケーションの実行時間は 1.5 倍になる。ここで、10%までの性能低下を許容する場合を考える。このとき、CPU 周波数依存度が異なるアプリケーションのそれぞれの実行時、どこまでクロック周期を落としてもよいかは、実行時間が 1.1 のときのクロック周期を見ればよい。すると、CPU 周波数依存度が 100%の場合は 1.1ns、CPU 周波数依存度が 50%の場合は 1.2ns、CPU 周波数依存度が 0%場合は、設定可能な最低の CPU 周波数でよく、2ns まで CPU 周波数を低くしてよい。

このように、CPU 周波数依存度が分かればアプリケーションの性能低下率の制御のもと、アプリケーション毎に最適な CPU 周波数を設定できる。2 つの異なる CPU 周波数でアプリケーションの実行を行えば、アプリケーションの CPU 周波数依存度を得ることができるが、その場合アプリケーション毎の個別分析を事前に必要とするため、事前の解析なしに、また、PMC を利用せず CPU 周波数依存度を得る手法が必要である。

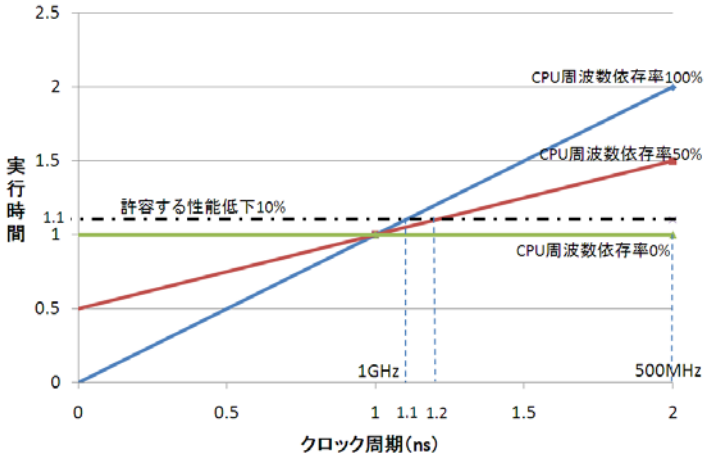


図 3. CPU 周波数依存度の異なる場合の CPU 周波数の設定

## 4. CPU周波数依存度とメモリ消費電力の相関調査

CPU 周波数依存度を明らかにするのに、我々はメモリ消費電力に着目する。メモリ依存型ではメモリアクセスが多いため、メモリの消費電力は高くなり、CPU 依存型ではメモリアクセスが少ないため、メモリの消費電力は低くなると考えられる。つまり、メモリの消費電力から、実行中のアプリケーションの CPU 周波数依存度を推定できると考えられる。

CPU 周波数依存度およびメモリの消費電力の測定は表 1 の環境で行った。

表 1. 評価マシン

| | |
|---|---|
| CPU | Intel Xeon X5570 |
| Memory | 12GB, DDR3 1333MHz Kingston (KVR1333D3D4R9S/4G) |
| OS | CentOS 5.5 |

CPU 周波数依存度とメモリの消費電力のデータ収集には、SPEC CPU2006 を用いた。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体である。実験には以下の 21 個のベンチマークを用いた。様々なメモリアクセスの負荷を取得する

ため、表 2 の各ベンチマークの実行の多重度を 1～4 と変更している。多重度が 2 以上の場合、アプリケーションの終了は、多重実行しているアプリケーションの全てが完了した時点とする。

表 2. 実験に利用した SPEC CPU2006 ベンチマーク

| | | | |
|---|---|---|---|
| 401.bzip2 | 429.mcf | 445.gobmk | 458.sjeng |
| 462.libquantum | 464.h264ref | 471.omnetpp | 473.astar |
| 483.xalancbmk | 433.milc | 434.zeusmp | 435.gromacs |
| 436.cactusADM | 437.leslie3d | 444.namd | 447.dealII |
| 450.soplex | 453.povray | 459.Gems.FDTD | 470.lbm |
| 482.sphinx3 | | | |

### 4.1 メモリ消費電力の測定方法

メモリ消費電力の測定方法の説明をする。表 1 の評価マシンのボードは、電力測定が可能なボードである。ボード上には CPU やメモリなどの消費電力を個別に測定するためのプローブが用意されており、これによりメモリ消費電力の取得を可能としている。現在は、特別な環境の利用を行っているが、将来的には、標準的な環境でも主要コンポーネントの電力測定は可能になると考えている。

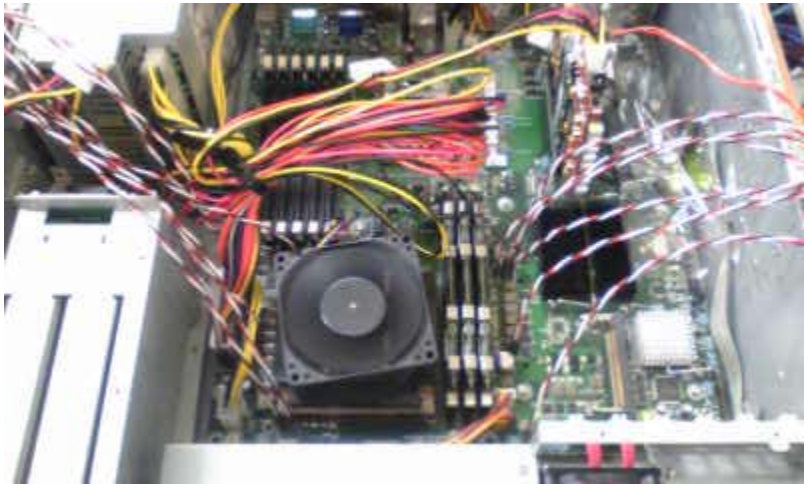

図 4.電力測定ボード

電力の測定は電力測定器 NetDAQ（Fluke 社）により行い、NetDAQ の制御ソフトは Intel Power Analyst を利用している。電力取得の間隔は、100msec としている。図 4 に実験に利用している消費電力測定が可能な環境を図 4 に示す。メモリモジュールの右隣など、ボード上に繋がっ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。

### 4.2 CPU周波数依存度とメモリ消費電力の関係

SPECCPU2006 の各アプリケーションについて、CPU 周波数依存度とメモリの消費電力測定結果を図 5 に示す。横軸はメモリ消費電力(p)(W)であり、縦軸は CPU 周波数依存度(d)(%)である。メモリの消費電力は、アプリケーションの実行時の平均値としている。図 6 より、メモリ消費電力が高いほど CPU 周波数依存度が低く、メモリ消費電力が低いほど CPU 周波数依存度が高い傾向があることが確認できる。実際、CPU 周波数依存度とメモリの消費電力との相関を取ると-0.93 と高い相関関係がある。この調査により、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ことを確認した。メモリ消費電力から 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、回帰式は図 6 に示す通りである。本手法によれば、CPU 周波数依存度を求めるためのモデル式は CPU 周波数依存度の異なる各アプリケーションのメモリ消費電力データより、機械的に生成可能である。

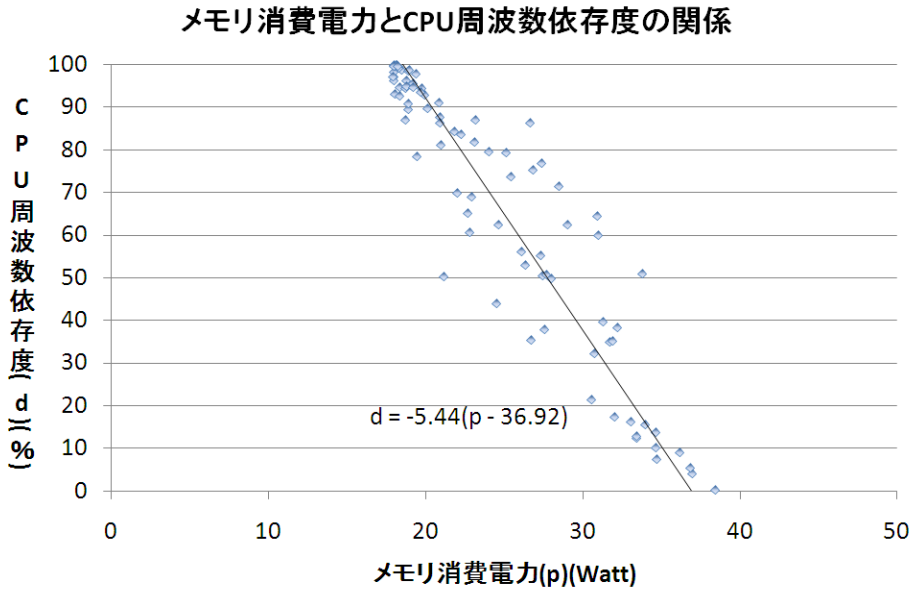


図 5. CPU 周波数依存度とメモリ消費電力の関係

$$d = -5.44 \times (p - 36.92)$$

図 6. CPU 周波数依存度の算出式

## 5. 提案手法のCPU周波数の決定方法

許容する性能低下率の上限と CPU 周波数依存度が与えられたときに、設定すべき CPU 周波数の決定方法を述べる。許容する性能低下率に対し、設定すべき周波数を求め方は関連研究と同様の手法を用いる[3]。最高周波数 fmax におけるアプリケーションの実行時間は、CPU 周波数依存の時間（Tfmax_cpu）と CPU 周波数に非依存の時間（Tfmax_mem）に分けられる（(1)式）。

$$\text{周波数} f_{max} : T_{f_{max}} = T_{f_{max}\_cpu} + T_{f_{max}\_mem} \quad (1)$$

CPU 周波数を fmax から f に変更した場合の実行時間は以下の(2)式の通り表される。

$$\text{周波数} f : T_f = \frac{f_{max}}{f} T_{f_{max}\_cpu} + T_{f_{max}\_mem} \quad (2)$$

したがって、CPU 周波数を fmax から f に変更したときの実行時間の増分は、(3)式の通り表される。

$$T_f - T_{f_{max}} = \frac{f_{max}}{f} T_{f_{max}\_cpu} - T_{f_{max}\_cpu} \quad (3)$$

$$= \left( \frac{f_{max}}{f} - 1 \right) \times T_{f_{max}\_cpu} \quad (4)$$

CPU 周波数依存の時間 Tfmax_cpu は、CPU 周波数 fmax の場合の全実行時間 Tfmax に CPU 周波数依存度（d）を掛けた値と同じであり、実行時間の増分は(5)式で表すことができる。

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

一方、性能の低下率（PFLoss）は、(6)式であらわされる。

$$PF_{LOSS} = \frac{T_f - T_{f_{max}}}{T_{f_{max}}} \quad (6)$$

(5)、(6)より、CPU 周波数（f）は(7)式であらわされる。

$$f = \frac{f_{max}}{PF_{LOSS} / d + 1} \quad (7)$$

ここで、PFLOSS を許容する性能低下率と考えると、CPU 周波数 f は、CPU 周波数 fmax の場合に対し許容する性能低下率を満たす CPU 周波数となる。CPU 周波数依存度（d）は得られたメモリ消費電力と図 6 式より推定し、許容する性能の低下率を与えると性能低下率を満たすための CPU 周波数 f を得ることができる。

## 6. 提案手法の評価

提案手法による性能低下率の制御、および、性能低下を許容する場合の消費電力量の削減の評価を行う。

### 6.1 環境

評価に用いたマシンは表 1 と同環境である。

評価用ベンチマークには、NAS Parallel Benchmark(NPB)(version 3.2, OpenMP 版)を用いた。NPB から 8 種類のベンチマークを実験に利用し、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。

目標とする性能低下率は 5, 10, 15, 20 (%)と変更し、性能の低下および消費電力量の削減を評価する。消費電力量はシステムのアプリケーション実行により増大する電力の主要コンポーネントである CPU とメモリに対して求める。

実験環境において設定可能な周波数は、1.60～2.93GHz の間の以下の 12 段階であるため、提案手法で求めた CPU 周波数 f より大きい設定可能な周波数のうち、最小の周波数を選択する。

CPU 周波数の変更は、3 秒毎に行い、過去 3 秒間のメモリ消費電力の平均値により次の 3 秒間の CPU 周波数を決定する。

### 6.2 性能低下率の評価結果

NPB の実験に用いた 8 種類の各ベンチマークについて、目標とした性能低下率に対する実際の性能低下率を調査した結果を図 7 に示す。横軸が目標とする性能低下率、縦軸が実際の性能低下率である。実験に用いたベンチマークのうち、cg ベンチマーク、is ベンチマークを除き、目標の性能低下率と実際の性能低下率は概ねリニアな関係であることを確認できる。目標とする性能低下率からの誤差の大きい cg ベンチマークについて、2 つの異なる CPU 周波数で CPU 周波数依存度を調べたところ 13%であった。一方、メモリ消費電力より推定した CPU 周波数依存度は 49%であった。実際の CPU 周波数依存度より高い CPU 周波数依存度と判定し、大幅に性能低下しないよう高めの CPU 周波数を設定したため、目標とする性能低下に届かない結果となった。is ベンチマークは、推定した CPU 周波数依存度が 17%であったのに対し、実際の CPU 周波数依存度は 41%であり、許容する性能低下を超えることが考えられるが、低い性能低下となっている。この原因はまだ明らかになっていないが、原因の 1 つとして is のアプリケーションの実行時間は 15 秒と短い実行時間であったのに対し、CPU 周波数の制

御のタイミングを 3 秒と粗いタイミングで制御を行ったことが影響したことが考えられる。平均の誤差率は、目標の性能低下率 5%, 10%, 15%, 20%に対し、それぞれ 2%, 3%, 3%, 3%（median）であり、概ね目標とする性能低下率となっている。

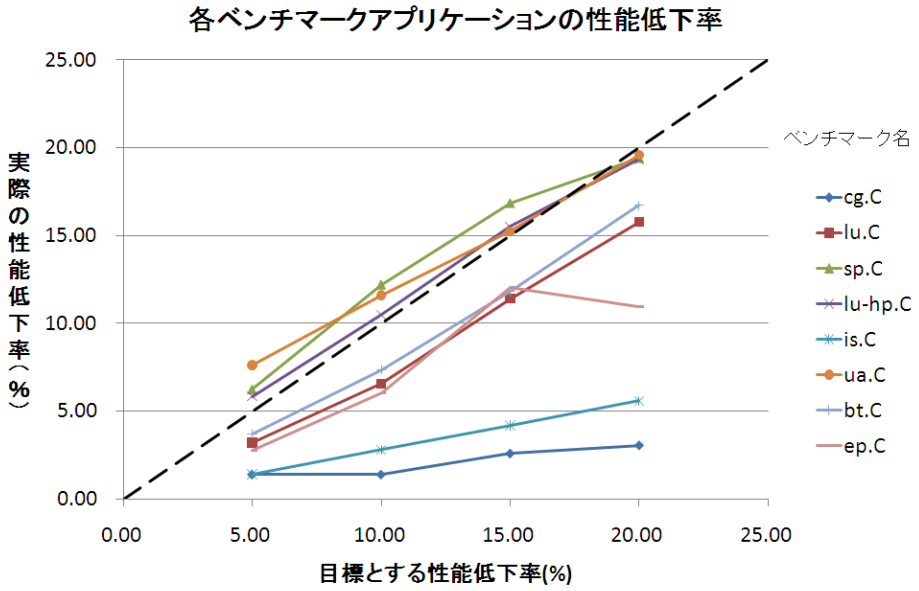


図 7.性能低下率評価結果

### 6.1 消費電力量の評価結果

図 8 は、各ベンチマークの実際の性能低下率に対する消費電力量の削減割合の結果である。横軸は各ベンチマークで、目標とした性能低下率(5%, 10%, 15%, 20%)に対する実際の性能低下率で項目分けしている。0%とは、最高周波数で一律に固定して実行した場合を表しており、このときの消費電力量を基準にしている。また、ベンチマークは左から右にいくにつれ CPU 依存度の大きくなる順番で並べている。図 8 より、メモリ依存型であるほど、消費電力量の削減効果が大きいことがわかる。削減できたのは、メモリ依存型では CPU の消費電力量が大きく削減できたためである。また、CPU 依存型では CPU とメモリの消費電力量の割合をみると、CPU の消費電力量の割合が大きいのに対し、メモリ依存型ではメモリの消費電力の割合が大きくなることも確認できる。実験に用いたベンチマークのうち、lu ベンチマークでは、5%の性能低下を許容する条件で CPU 周波数を制御した場合に、3%の性能低下に対し、9%の消費電力量削減を達成している。

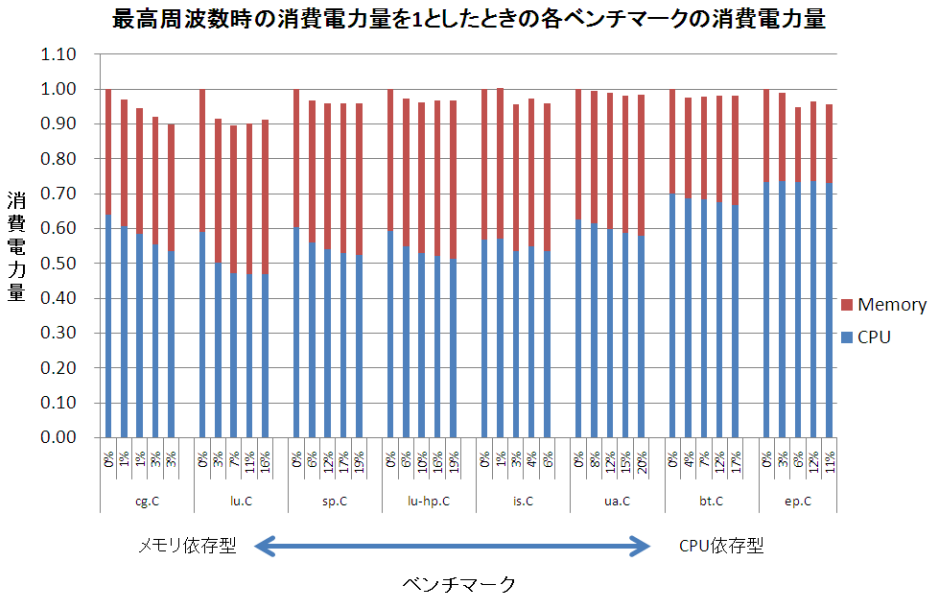


図 8. 消費電力量結果

## 7. おわりに

メモリの消費電力がアプリケーションの CPU 周波数依存度と高い相関を持つことに注目し、メモリ電力に基づき動的に CPU 周波数の制御を行う手法を示した。提案手法の NPB による評価の結果、8 個中 6 個のベンチマークにおいて概ね目標通り性能の制御を行うことができた。また、5%の性能低下を許容する条件で CPU 周波数を制御した場合に、メモリ依存型の lu ベンチマークでは、3%の性能低下率に対し、9%の消費電力量の削減が達成できた。本手法は、PMC を利用する従来手法に代わる高性能と省電力の両立のための手段として活用できる。また、本手法はメモリの消費電力というセンサーにより取得可能な情報を利用しているため、原理上 BIOS レベルでの実装も可能である。

今後は、今回性能低下率の制御がうまく行われなかった is ベンチマークにおいて原因の詳細な分析を行う予定である。また、異なるメモリモジュールや OS を用いたときの本手法の評価を予定している。異なるメモリモジュールや OS にすることで、どの程度のキャリブレーションが必要になるかの確認をする。さらに、他のハードウェ

ア上の取得可能な情報を合わせて利用することで、本手法の精度の向上に取り組む。

## 参考文献

1) K. Choi, R. Soma, and M. Pedram. Fine-grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times. In DATE04, 2004.
2) R. Kolta, S. Ghiasi, T. Keller and F. Rawson, “Scheduling Processor Voltage and Frequency in Server and Cluster Systems”, 2005.
3) S. Huang, W.Feng,“Energy-Efficient Cluster Computing via Accurate Workload Characterization”,.
4) H. Amur, K. Schwan, M. Prvulovic, “Towards Optimal Power Management: Estimation of Performance Degradation due to DVFS on Modern Processors”, 2010.
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”, 2004.
6) P. Stanley-Marbell, M. S. Hsiao, U. Kremer, “A Hardware Architecture for Dynamic Performance and Energy Adaptation”, 2002.
7) J. Kim, S. Yoo, C. Kyung, “Program Phase-Aware Dynamic Voltage Scaling Under Variable Computational Workload and Memory Stall Environment”, 2010.
8) 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”, 2006.
9) Intel White Paper, “Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor”, 2004.
10) V.Pallipadi, A. Starikovskiy, “The Ondemand Governor”, 2006.
11) SPEC CPU2006, http://www.spec.org/cpu2006/.
12) NAS Parallel Benchmark, www.nas.nasa.gov/Software/NPB/.