2005－ARC－163（5）
2005／5／31

CMP 向け動的電源電圧・周波数制御手法の提案

近　藤　正　章†　　中　村　　宏†

チップマルチプロセッサ (CMP) は，複数のプロセス (スレッド) が L2 キャッシュやチップ・メモリ間バスを共有するために，各プロセスの性能は他のプロセッサコア上で同時に実行するプログラムの性質に大きく依存する．共有リソースにおいて競合が生じると，チップ全体の性能が大きく低下したり，各プロセスの性能低下の影響の公平さ (Fariness) が失われるといった問題が生じる．本稿では，CMP において，各プロセッサコアの周波数・電源電圧を個別に制御し，性能，Fairness，および消費エネルギー効率を向上させる手法を提案する．提案手法を評価した結果，多くのプログラムにおいて性能，および Fairness を向上することができ，また命令あたりの消費エネルギーを大きく削減できることがわかった．

A Dynamic Voltage and Frequency Control Technique for a Chip Multiprocessor Architecutre

Masaaki Kondo† and Hiroshi Nakamura†

In a Single Chip Multiprocessor (CMP), threads on different cores share hardware resources such as cache memory and memory bus. Resource contention significantly degrades performance of each thread and also loses fairness between threads. This paper proposes a Dynamic Frequency and Voltage Scaling (DVFS) technique to improve total instruction throughput, fairness, and energy efficiency of CMPs. The proposed technique controls the frequency and the voltage of each processor core individually to balance the utilization ratio of shared resources among threads. We evaluate our technique and the evaluation results show that total instruction throughput and fairness between threads are greatly improved with reducing energy consumption.

1. はじめに

近年，消費電力や発熱量の限界から，クロック周波数向上による高性能化が望めなくなりつつあり，VLSI チップにおける性能向上手段として，Chip Multiprocessor (CMP) が今後の有効なアーキテクチャとして注目されている．複数のプロセッサコアを (PU) を 1 チップに搭載する CMP により，シングルタスクの並列処理，あるいは複数タスクの並行処理を行なうことで，高い処理能力を得ることができる．したがって，周波数の向上に頼ることなく高性能が望めるため，性能あたりの消費電力効率が向上すると期待されている．また，Simultaneous Multi-Threading (SMT) と比較した場合でも，CMP はエネルギー効率に優れるアーキテクチャであるという報告もある[1]．

CMP ではリソース有効活用の観点から，複数の PU があるメモリ階層以下 (たとえば L2 キャッシュ以下) にあるキャッシュやバスを共有するのが一般的である．この共有リソースの使用率はプログラムの性質に大きく依存し，同時に実行するプログラムの組み合わせによってはリソース競合が発生する．したがって，ある PU で実行されているプログラムの性能は，他の PU 上で実行されているプログラムに大きく影響を受ける．

従来より，CMP における共有キャッシュに着目し，PU 間でのキャッシュ競合によるミスの増加を防ぐことにより，アプリケーション実行のスループットを向上させる手法が提案されている[2]．また，プログラムを単独で実行した場合，すなわち競合がない場合に比べて，CMP 上で複数のプログラムを実行した際に，各プログラムの性能がどの程度低下するかの公平さを示す指標である *Fairness* を向上させる手法も提案されている[3]．これら従来手法は，キャッシュを論理的なパーティションに分割し，それらを各プログラム専用の領域として割り当てることで，キャッシュ上での競合を防ぎつつ効率的な実行を提供するものである．

本稿ではキャッシュを分割するのではなく，各コアの動作周波数，および電源電圧を Dynamic Voltage/Frequency Scaling (DVFS) 手法により制御することで，各プロセスの命令処理の進行をコントロールし，各プロセスの共有リソースへのアクセスを調整することで，競合を削減する手法を提案する．その結果として，Fairness および命令処理のスループットを向上させることが可能となる．さらに，低い電源電圧で動作させることで，消費電力が削減できるため，エネルギー効率が向上する．この点は，従来のキャッシュ分割手法に比べて本提案手法の大きな利点である．

本稿では，提案手法について述べ，複数の独立のタスクの並行処理を行なう場合を対象に，Fairness，性

† 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

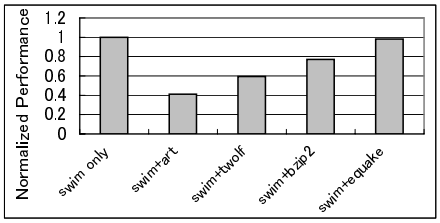

図 1 171.swim の性能

能，および消費エネルギーについて評価を行なう．

2. CMP におけるリソース共有の影響

2.1 リソース共有による性能低下

CMP では，複数のプロセッサコアがキャッシュの一部やバスを共有するのが一般的であり，L2 キャッシュやメモリバスといったリソースを共有することが多い．そのため，各 PU 上で動作するプロセスの性能は，共有リソースへのアクセス頻度，すなわち共有リソースの使用率に大きく依存する．図 1 は，2 つのプロセッサコアを持ち，L2 キャッシュおよびメモリバスを共有する CMP において，片方の PU で SPEC ベンチマークの swim を，他の PU で異なるプログラムを実行した場合の swim の性能を表している．なお，図中の *swim-only* は swim のみを 1 つの PU で実行した場合を示す．評価環境については 4 章で述べる．

図 1 より，プログラムの組み合わせに応じて，swim の性能が大きく異なることがわかる．*swim+equake* では swim-only と比べ，ほぼ同じ性能であるのに対し，*swim+art* では性能低下が非常に大きい．equake は L2 キャッシュミス率が小さく，L2 キャッシュやバス上での競合がほとんど発生しないため，swim を単独で実行するのとほとんど同じ環境となる．逆に，art は L2 キャッシュミスが多く，バスアクセスが頻発するため，swim でキャッシュミスが生じた場合にバス競合が発生し，メモリからのデータ待ち時間が長くなる．これが，性能低下率が大きい理由である．

このように，CMP ではリソース共有の影響で，実際に各 PU で実行しているプロセスの性質により，その性能が大きく変動する．次節では，チップ全体で見た場合の性能，および Fairness について議論する．

2.2 チップ全体の性能および Fairness

2.2.1 トータルスループット

複数の PU を 1 チップに搭載する CMP では，個々のプロセスの性能だけでなく，複数の PU 上で実行される複数のプロセスがどれだけ高速に実行できるかも重要な指標である．本稿では，チップ全体の性能を，単位時間あたりに実行できる全 PU トータルの命令数 (トータルスループット) として，Instruction Per Second (IPS) により表し議論する☆．

☆ DVFS により周波数を変化させることを考慮し，Instruction Per Cycle (IPC) の代りに IPS を用いる．

ある時間 t[s] 中に，コア i 上で実行されるプロセス $Proc_i$ の実行命令数が $Inst_i$ であった時，$Proc_i$ の性能を $IPS_i = \frac{Inst_i}{t}$ とすると，n 個の PU を持つ CMP のチップ全体のトータルスループット IPS_{total} は，

$$IPS_{total} = \sum_{i=0}^{N-1} IPS_i \quad (1)$$

で表される．

ある PU 上で実行しているプロセスの L2 キャッシュミス率が高く，共有リソースへのアクセス率が高い場合，他のプロセス性能は大きく低下する．もともと高い IPS を達成できるプロセスの性能低下が大きいと，トータルスループットで考えた場合，チップ全体の性能は大きく低下することになる．

2.2.2 Fairness

Fairness は，各 PU 上で実行するプロセス間の，リソース共有の影響による性能低下の公平さを示す評価尺度である．文献[3] では，各プロセスの実行時間における Fairness が次のように定義されている．$Tded_i$ をリソース共有の影響がない場合，例えばチップ上でプロセス i のみを実行した場合の実行時間，$Tshr_i$ を複数プロセスが動作している状況下でのプロセス i の実行時間として，n プロセスを実行した場合，以下の式が満される時，Fairness は満されると考える．

$$\frac{Tshr_1}{Tded_1} = \frac{Tshr_2}{Tded_2} = ... = \frac{Tshr_n}{Tded_n} \quad (2)$$

本稿では，実行時間の Fairness と同様に，性能に関する Fairness を以下のように定義する．$IPSded_i$ をリソース共有の影響がない場合の性能，$IPSshr_i$ を複数のプロセスが動作している状況下でのプロセス i の性能として，n プロセスを実行した場合とし，

$$\frac{IPSshr_1}{IPSded_1} = \frac{IPSshr_2}{IPSded_2} = ... = \frac{IPSshr_n}{IPSded_n} \quad (3)$$

が満される時，Fairness は満される．Fairness を定量化するため，以下の式を Fairness の評価尺度として導入する．

$$Fair_{ij} = |X_i - X_j|, where X_i = \frac{IPSshr_i}{IPSded_i} \quad (4)$$

この，$Fair_{ij}$ が小さい値であるほど，Fairness は良いことになる．

Fainess は OS のタスクスケジューリングにとって重要である．OS は priority に従い各タスクに対してタイムスライスを割り当てるが，その際同時に実行されているプロセスは同じようにリソース競合による性能への影響を受ける，すなわち Fairness が保たれていることが前提とされる．そのため，OS のタスクスケジューラの効率は Fainess に大きく依存する[3]．したがって，Fairness を最適化することは重要な課題である．

2.3 競合による性能低下の分類

L2 キャッシュが共有されることによる競合性ミスの増加と，バス競合によるメモリからのデータ転送待時間の増加がどの程度性能に影響するかを調べるため，図 2 に，2PU で L2 キャッシュ，およびメモリバスを共有する CMP において，各 PU で異なる SPEC ベンチ

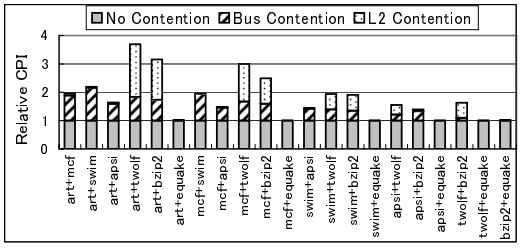

図 2 競合による CPI の悪化

マークのプログラムを実行させた場合における Cycle Per Instruction (CPI) を示す．CPI は全く競合がない場合のサイクル (No Contention)，バスの競合によるサイクル数の増加 (Bus Contention)，L2 キャッシュの競合によるサイクルの増加 (L2 Contention) に分類している．なお，ベンチマーク毎に，競合がない場合の CPI に対する相対的な CPI を示している．この結果は，L2 キャッシュとメモリバスを共有する CMP，L2 キャッシュがそれぞれコア毎に独立にある場合 (各コアの L2 キャッシュサイズはもとの CMP と同じ)，L2 キャッシュ，バス共に全く競合が発生しない場合 (コア 1 つのみを用いて，1 つのプログラムを実行した場合) という条件のもとで各 CPI を評価し，それらの差により求めた．

図 2 より，競合によりほとんどのプログラムで CPI が増加していることがわかる．また，多くのプログラムの場合で，バス競合による CPI の増加が大きいことがわかる．キャッシュ競合によるミスの増加は，キャッシュ容量がプロセスのデータセットに近く，他のプロセスによるアクセスにより再利用性のあるデータが追い出されてしまう時に生じる．しかし，多くの場合ではデータセットがキャッシュ容量よりもはるかに大きい，または小さく，L2 キャッシュ競合が性能に影響しない．一方，バスの競合は，必ず性能低下を引き起こしてしまう．したがって，性能や Fairness の向上のためには，キャッシュ領域の分割により L2 キャッシュ競合を削減するだけでは不十分であり，バス競合に着目した最適化を行なう必要があると考えられる．

3. DVFS による Fairness およびスループットの向上

本節では，各コアの動作周波数，および電源電圧を Dynamic Voltage/Frequency Scaling(DVFS) 手法により制御することで，各プロセスの命令処理の進行を調整し，その結果としてプロセス間の共有リソースの使用率を調整することで，Fairness およびトータルスループットを向上させる手法を提案する．なお，Fairness を向上させることで，多くの場合でトータルスループットも向上するという報告もあり[3]，本稿では主に Fairness を向上させる手法を提案する．

3.1 CMP における DVFS

動的電源電圧・周波数制御 (*Dynamic Voltage / Frequency Scaling: DVFS*) は，プロセッサの処理の負荷などに応じて，動的にプロセッサのクロック周波数と電源電圧を調節する手法であり，低消費電力化の目的で近年広く用いられている手法である．VLSI システムの消費電力は電源電圧の 2 乗に比例するため，電源電圧を下げることで大きな電力削減が期待できる．

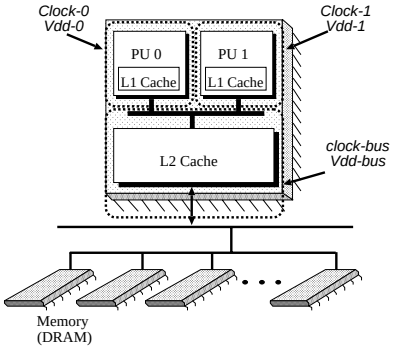

図 3 CMP の例

近年，クロック伝搬における遅延や，消費電力増大への対処を目的として，1 つのチップ内を複数の領域に分割し，各領域が異なるクロックで動作する *Globally Asynchronous Locally Synchronous (GALS)* 手法が注目されている．クロックを伝搬すべき領域を局所に閉じ込めることで，遅延が短くなりクロックのタイミングの調整が容易になるほか，クロック信号自体の消費電力が少なくなるという利点を持つ．しかし，異なるクロック領域間でのデータの通信を行なう際には，安全にデータのやりとりを行なうための非同期通信のインタフェースが必要となる．非同期インタフェースでは，同じクロック領域内で通信を行なうのに比べ，性能のオーバヘッドがある．

この GALS 技術の応用により，シングルコアのマイクロプロセッサ内部を複数の周波数・電源電圧領域に分割し，各領域の負荷に合わせ DVFS 手法を適用することで消費電力を削減する研究もある[4),5)]．チップ内に複数のプロセッサコアを持つ CMP では，図 3 に示すように，各コアに独立のクロックと電源電圧を供給することで，コア毎で独立に電源電圧・周波数制御を行なうことが可能となる．また，System on Chip (SoC) において，チップ内の各モジュールに独立の周波数・電源電圧を供給するチップも開発されており[6),7)]，すでに実現可能な LSI 技術であると考えられる．

3.2 DVFS による Fairness の向上手法

3.2.1 概　　要

バス競合は，複数の PU からのキャッシュミスによるデータ転送リクエストが重なることで生じる☆．主記憶へのデータアクセスレーテンシは非常に長く，non-blocking キャッシュを用いたとしても，プロセッサは

☆ メモリバスは，キャッシュミスリクエスト以外にも，I/O アクセスのために用いられるが，ここではキャッシュミスによるデータ転送リクエストに対象を絞り議論する．

ストールすることが多い．ここで，バス競合によりデータ転送リクエストが待たされると，キャッシュミス解決までのレーテンシが更に長くなり，大きな性能低下が生じる．

特に，あるプロセスのキャッシュミス率が高く，バスに対するアクセスが頻発する場合，他のプロセスの性能低下率が相対的に大きくなる．この場合，Fairnessは保たれない．そこで，頻繁にバスにアクセスするプロセスが動作するPUの周波数を下げることでバスアクセスを抑制し，他のプロセスのバスアクセスを円滑にさせることでFairnessを向上させることができる．

DVFSを利用し，同時に実行されている複数プロセスのFairnessを向上させるためには，1) 競合がない場合に比べ各プロセスの性能がどの程度低下するかを予測し，2) 性能低下率が等しくなるよう，各PUの周波数を制御する必要がある．性能低下率の予測については，他のPU上で同時に実行しているプロセスの性質や，そのデータセットなどにも依存するため，コンパイル時などに行なうのは不可能であり，実行時に動的に予測する必要がる．そのため，本稿ではインターバルベースの手法を用い，動的に性能低下の予測，および周波数・電源電圧制御を行なう．

3.2.2 性能低下の予測

まず，PU毎に未解決のバスリクエスト数をカウントするカウンタ $Creq_i$ を導入する．$Creq_i$ は，PU番号 i のリクエスト数を保持するカウンタである．また，バスで転送中のリクエストがどのPUからのものかを保持するレジスタ $Rreq$ を設ける．もし，PUi のカウンタ $Creq_i$ が1以上であり，$Rreq$ が i でない場合，その時間はバス競合のためにPUi のバスリクエストが待たされていることになる．そこで，リクエストが待たされていたサイクル数をカウントするため，バスサイクル毎に，$Creq_i$ と $Rreq$ を監視し，$Creq_i > 0$ かつ $Rreq \neq i$ の場合にインクリメントするPU毎のカウンタ $Cwait_i$ を導入する．

$Cwait_i$ は，バス競合によるストール時間と考えられるため，$Cwait_i$ を比較することで各プロセスの性能低下率が予測できる．$Cwait_i$ が相対的に大きい場合性能低下が大きく，逆に $Cwait_i$ が小さい場合は性能低下が小さいことになる．

3.2.3 周波数・電源電圧の制御

Fairnessを保つためには，$Cwait_i$ が等しくなるように各PUの周波数・電源電圧を制御すればよい．周波数・電源電圧の変更は，タイムインターバル T_{itvl} 毎に行ない，T_{itvl} 中の $Cwait_i$ を基に，次のインターバルの周波数・電源電圧を決定する．具体的には，$Cwait_i$ が相対的に大きいコアの周波数を上げ，小さいコアの周波数を下げるように制御する．

まず，$Cwait_i$ の平均 $Cwait_{avg}$ と $Cwait_i$ との差分 $diff_i$ をPU毎に求める．$diff_i$ に対して，上限の閾値 Th_u と下限の閾値 Th_l を設け，$diff_i$ が Th_u 以上であった場合はPUi の周波数・電源電圧を1レベル下げる．逆に，$diff_i$ が Th_l 以下であった場合は，PUi の周波数・電源電圧を1レベル上げる．それ以外であった場合は変更は行なわない．なお，$Cwait_i$ の値は T_{itvl} 毎にリセットする．

```
N = number of PUs;
for each bus-cycle {
  for (i = 0; i < N; i++){
    if (Creq_i > 0 && Rreq ≠ i)
      Cwait_i++;
  }
  /* for every T_itvl */
  if ((BusCycleCount % T_itvl) == 0){
    Cwait_avg = (Σ_{i=0}^{N} Cwait_i) / numPU
    for (i = 0; i < N; i++){
      diff_i = Cwait_avg − Cwait_i
      if (diff_i > Th_u && C_lev > MinClockLev)
        DownClockLev(C_lev);
      elseif (diff_i < Th_l && C_lev < MaxClockLev)
        UpClockLev(C_lev);
      else
        /* UnchangingClockLevel */;
      Cwait_i = 0;
    }
  }
  BusCycleCount++;
}
```

図 4 周波数・電源電圧制御のアルゴリズム

上記のアルゴリズムをまとめたものを，図4に示す．

4. 評　　価

4.1 評価環境

提案手法の評価のため，SimpleScalar Tool Set[8]を用いてシミュレーションにより評価を行なう．なお，SimpleScalarは，図3のCMP構成が評価できるように拡張し，また消費電力評価のためにWattch[9]も統合している．評価プログラムは，SPEC CPU2000ベンチマークからキャッシュミス率の異なるいくつかのプログラムを用いる．コンパイラは，Alpha用の命令セットを生成するDEC Cコンパイラを用い，オプションは“-arch ev6 -fast -O4 - non_shared”である．なお，*ref* インプットセットを用い，最初の10億命令実行後に各コア上で動作するプログラムのどちらかが100万命令を実行するまで評価する．

4.2 評価の仮定

表1に評価における各PU，および共有L2キャッシュなどの仮定を示す．また，プロセッサコア数は2個の場合を評価した．また，周波数・電源電圧の仮定はIntel Pentium Mプロセッサの設定をベースに，表2に示す7通りのレベルを仮定する．なお，電源電圧の低下には限界があることをふまえ，400MHz以下の電源電圧には同じ値を用いる．

なお，3.2節で述べた，提案手法のアルゴリズムにおけるパラメータは以下の値を用いて評価を行なう．

- T_{itvl}: 250000 bus-cycle = 625 [μs]
- Th_u: 20000, Th_l: 10000

上記の仮定のもと，評価では提案手法と常に最高周波数で動作する通常のCMP (Original) を比較する．

表 2 周波数・電源電圧の仮定

Clock	1.6GHz	1.2GHz	800MHz	400MHz	200MHz	100MHz	50MHz
Vdd	1.484V	1.276V	1.036V	0.956V	0.956V	0.956V	0.956V

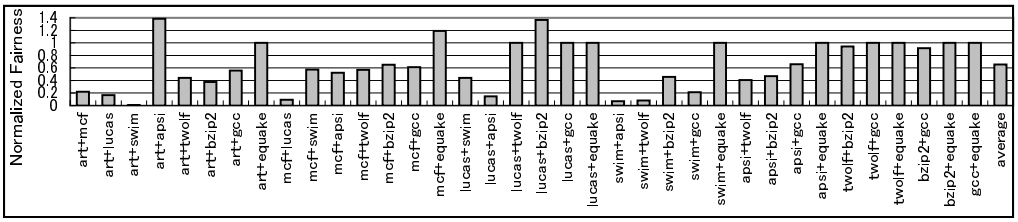

図 5 Original に対する相対的な Fairness

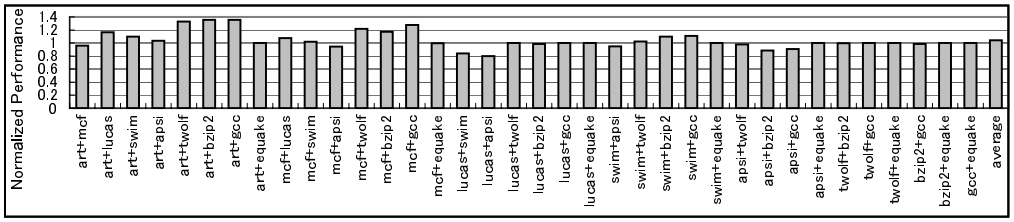

図 6 Original に対する相対的な性能

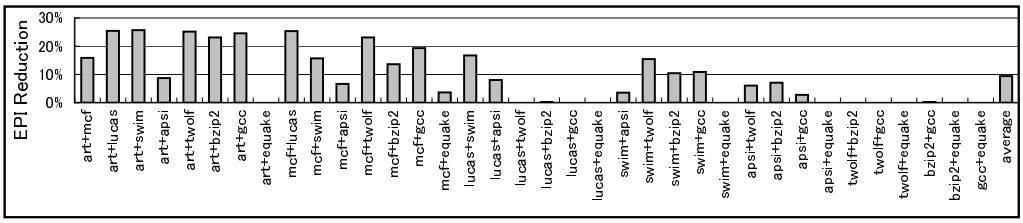

図 7 消費エネルギー効率の向上率

表 1 評価における仮定

Fetch/Issue/Commit	4 instruction / cycle
Branch prediction	Combined bimodal (4K-entry) gshare (4K-entry) selector (4K-entry)
BTB	1024 sets, 4way
Branch Mis-penalty	7 cycles
RUU size	64
LSQ size	32
Functional units	Int: 6 ALU, 2-mult/div FP: 6 ALU 4 mult/div Load/Store: 2 ports
L1 I-Cache	32KB, 32B line, 2way 1 cycle latency
L1 D-Cache	32KB, 32B line, 4way 2 cycle latency
Shared L2 Cache	1MB, 128B line, 8way 10 cycle latency in 1.6GHz
Main memory latency	100 cycle in 1.6GHz
Bus width	8B
Bus clock	400MHz

5. 評価結果

5.1 Fairness

図 5 に Original の Fairness を 1 とした場合に対する，提案手法の相対的な Fairness の値を示す．Fairness は式 (4) の $Fair_{ij}$ の値であり，小さいほど Fairness は良いことになる．

図 5 より，ほとんどのプログラムにおいて，Fairness が向上していることがわかる．特に，キャッシュミス率が比較的高いプログラムの組み合わせの場合に Fairness の向上が大きい．キャッシュミス率が高い場合，バスへのアクセスが頻繁に生じ競合も発生しやすいため，Original では Fairness が悪くなることが多い．提案手法により，バス競合による性能への影響を各プロセスで公平になるように周波数を制御することで，実際に Fairness を大きく向上させることができる．

一方，*art+apsi* および *mcf+equake*，*lucas+bzip2* では，Fairness が悪化している．これらのプログラムでは，Original の場合においても，非常に良い Fairness を達成しており，提案手法により周波数を制御した結果，性能低下率の調整が行き過ぎてしまったためである．しかし，提案手法の場合でも，両プログラムの性能低下率の差，すなわち $Fair_{ij}$ の絶対値はそれぞれ 0.05，0.02，0.06 と非常に小さく，提案手法での

Fairness の悪化はほとんど問題とならない．

5.2 性　　能

図 6 に，各ベンチマークペアの Original に対する提案手法の性能 (IPS_{total}) を示す．評価したプログラムの組み合わせ 36 通り中，性能向上したものが 14 個，性能が悪化したものが 12 個，Original と同じ性能が 10 個という結果となっており，平均では 4%の性能向上を達成している．ここで，性能が低下したプログラムにおいても，Fairness は向上している．この結果は，Fairness の向上が必ずしもトータルスループットの向上には結び付かないことを示している．ただし，*lucas+swim* および *lucas+apsi* を除いて性能低下の割合はそれほど大きくない．

5.3 消費エネルギー

図 7 に，各ベンチマークペアの Original に対する Energy Per committed Instruction (EPI) の削減率を示す．DVFS により，周波数と同時に電源電圧を低下させることで，提案手法では多くのプログラムで消費エネルギー効率が向上している．性能およびエネルギー効率の両者が改善されているプログラムも多い．したがって，提案手法は CMP において非常に有効な手法であると考えられる．

6. 関 連 研 究

従来より，複数のプロセス (スレッド) を実行するプロセッサにおいて，共有リソースの使用率を考慮しつつ，性能や Fairness を向上させる研究が行なわれている．特に，メモリ階層だけでなく，演算器などのリソースも共有する Simultaneous Multi-Threading (SMT) プロセッサでは，リソースの使用率を考慮した最適化の研究が活発に行なわれている[10),11)]．また，CMP において共有 L2 キャッシュを分割し，競合ミスを削減することで，性能や Fairness を向上させる手法も提案されている[2),3)]．また，文献[12)] では，キャッシュ競合の影響を予測するモデルが提案されている．

また，PU 毎に DVFS を適用することで，CMP における消費電力削減を目指す手法も提案されている[13)]．この手法は，並列処理プログラムにおけるバリア同期に達した際のストール時間を利用し，DVFS による消費電力削減を狙うものである．

本稿で提案する手法は，PU 毎に個別に DVFS を行ない，プロセスの実行の進行を調整することで，性能，および Fairness だけでなく，命令あたりの消費エネルギーを削減することができる．この点は，従来の手法に比べて提案手法の大きな利点である．

7. まとめと今後の課題

本稿では，CMP 上の各プロセッサコアの周波数・電源電圧を個別に制御し，Fairness およびトータルスループットを向上させつつ，命令あたりの消費エネルギーを削減する手法を提案した．提案手法は，メモリバス競合による性能への影響を予測し，性能低下率が各プロセスで公平になるように周波数・電源電圧を制御するものである．

評価の結果，提案手法によりほとんどのプログラムで Fariness が向上し，また，トータルスループットが向上するプログラムも多いことがわかった．さらに，消費エネルギー効率も改善させることができるため，提案手法は，CMP において効率的なアプリケーションの実行のために，非常に有効な手法であると結論付けることができる．

今後，さらに消費エネルギー効率を改善するために，アルゴリズムを改良する他，多くのベンチマークプログラムで提案手法の評価を行なう予定である．

謝辞　本研究の一部は，(株) 半導体理工学研究センターとの共同研究によるものである．

参 考 文 献

1) R. Sasanka, S. Adve, Y.-K. Chen, and E.Debes, “Energy Efficiency of CMP vs. SMT for Multimedia Workloads”, *In Proc. the 18th ICS*, pp.196-206, June 2004.
2) G.E. Suh, S. Devadas, and L. Rudolph, “A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning”, *In Proc. 8th High Performacne Computer Architecture*, pp.117–128, Feb. 2002.
3) S.Kim, D.Chandra, and Y.Solihin, “Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture”, *In Proc. 13th PACT*, pp.111–122, Oct. 2004.
4) A. Iyer and D. Marculescu, “Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors”, *In Proc. 30th ISCA*, pp.158–168, May 2002.
5) G.Magklis, et al. “Profile-based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor”, In Proc. 30th ISCA, pp.14–27, June 2003.
6) T. Fujiyoshi, et al., “An H.264/MPEG-4 Audio/Visual Codec LSI with Module-Wise Dynamic Voltage/Frequency Scaling”, In proc. 2005 ISSCC, pp.132–133, Feb. 2005.
7) K. Nose, et al., “Deterministic Inter-Core Synchronization with Periodically All-in-Phase Clocking for Low-Power Multi-Core SoCs”, In proc. 2005 ISSCC, pp.296–297, Feb. 2005.
8) T. Austin, et al., “SimpleScalar: An Infrastructure for Computer System Modeling”, *IEEE Computer*, Vol. 35, No. 2, pp.59–67, Feb. 2002.
9) D. Brooks, et al., “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations”, *In Proc. 27th ISCA*, pp.83–94, June 2000.
10) A. Snavely and D.M. Tullsen, “Symbiotic Jobscheduling for a Simultaneous Multithreading Processor”, *In Proc. ASPLOS IX*, pp.234–244, Nov. 2000.
11) K. Luo, J. Gummaraju, and M. Franklin, “Balancing Throughput and Fairness in SMT Processors”, *In Proc. ISPASS 2001*, pp.164–171, Nov. 2001.
12) D. Chandra, et al., “Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture” *In Proc. 11th HPCA*, pp.340–351, Feb. 2005.
13) C. Liu, et al., “Exploiting Barriers to Optimize Power Consumption of CMPs”, *In Proc. IPDPS 2005*, pp.5, April 2005.