情報処理学会論文誌　コンピューティングシステム　Vol.5　No.4　24–35 (Aug. 2012)

# コア数と動作周波数の動的変更による メニーコア・プロセッサ性能向上手法の提案

今村 智史[1,a)]　佐々木 広[2]　福本 尚人[1]　井上 弘士[2]　村上 和彰[2]

受付日 2012年1月17日, 採録日 2012年4月22日

**概要**：本論文では，消費電力制約下において並列アプリケーション実行時の性能を最大化する Dynamic Core-count and Frequency Scaling（DCFS）手法を提案する．DCFS はアプリケーションの特性に応じてコア数と動作周波数を動的に変更させる手法である．本手法は，“トレーニングフェイズ” と “実行フェイズ” の 2 つのフェイズから構成され，トレーニングフェイズで最適な構成を選択し，実行フェイズではその最適な構成を用いてプログラムが実行される．この 2 つのフェイズの実行を繰り返すことにより，時々刻々と変化する特性に対応することが可能となる．PARSEC ベンチマークを用いた評価の結果，チップ上の全コアを用いた実行に対して，`dedup` において最大で 35%の性能向上を達成し，評価に用いた全 10 個のベンチマークにおいて平均 6%，性能向上が得られた 4 つのプログラムにおいて平均 20%性能が向上した．

**キーワード**：メニーコア・プロセッサ，並列プログラム，DVFS，コアスロットリング，ランタイムシステム

# Improving Performance of Many-core Processors through Dynamic Core-count and Frequency Scaling

Satoshi Imamura[1,a)]　Hiroshi Sasaki[2]　Naoto Fukumoto[1]
Koji Inoue[2]　Kazuaki Murakami[2]

Received: January 17, 2012, Accepted: April 22, 2012

***Abstract:*** We propose dynamic core-count and frequency scaling (DCFS) technique to optimize the power-performance trade-offs for multi-threaded applications. Our proposed technique adjusts core counts and CPU frequency depending on the parallelism of applications under the power consumption constraint. DCFS dynamically controls the settings to optimize against the phases within programs by having two phases: Training and Execution. Additionaly, there is no need for static analysis and modification of applications. We can achieve a performance improvement of 35% for `dedup` and 20% on average among four applications from PARSEC benchmarks compared to the execution with all cores equipped on a chip.

***Keywords:*** many-core processors, multi-threaded applications, DVFS, core throttling, runtime system

## 1. はじめに

シングルコア・プロセッサの性能向上は，動作周波数の上昇や複雑なアウトオブオーダ実行型のスーパスカラ・アーキテクチャを実装することで達成してきた．しかしながら，消費電力の問題によりその性能向上は限界を迎えている．そのため近年のプロセッサでは，1 つのチップに複数のプロセッサ・コア（以下，コアと略称）を搭載したマルチコア・プロセッサ（以下，マルチコアと略称）が主流である．また，数十のコアを搭載したメニーコア・プロセッサ（以下，メニーコアと呼称）が登場しており，微細化技術の発展により 1 つのチップに搭載されるコアの数は今後

[1] 九州大学大学院システム情報科学府情報知能工学専攻
Department of Advanced Information Technology, Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819–0395, Japan
[2] 九州大学大学院システム情報科学研究院情報知能工学部門
Department of Advanced Information Technology, Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819–0395, Japan
[a)] s-imamura@soc.ait.kyusyu-u.ac.jp

さらに増加すると予想される [5], [7], [8]．マルチコアの高性能化と低消費電力化を達成するには効率的な並列処理が重要であり，メニーコアにおいてはその重要性がさらに増す．大量のコンピュータを扱う大規模なデータセンタやスーパコンピュータ等の高性能計算機では消費電力削減に対する要求が高まっており，厳しい消費電力制約下においていかにプロセッサの性能を最大化するかが課題となる．これに加え，今後のメニーコアでは様々な特性や要件を持つアプリケーションを扱うことが予想されるため，それらに応じてエネルギー効率の良い実行を実現できる環境が求められる．

消費電力制約下においてエネルギー効率の良いアプリケーション実行を実現するための既存手法として，Dynamic Voltage and Frequency Scaling（DVFS）があげられる [9]．DVFS によりプロセッサの電力効率を向上させることができるため，マルチコアを対象とした様々な DVFS 適用技術に関する研究がなされている [4]．メニーコアにおいても，消費電力バジェットに応じて適したコアの動作周波数と供給電圧を選択することで電力効率の向上が期待できる．しかしながら，そのような場合には，性能を決定する要因として動作周波数だけでなく並列性も同時に考慮しなければならない．なぜなら，チップ上の全コアを用いた並列処理が最高性能やエネルギー効率の良い実行を実現できるとは限らないためである．

本論文では，消費電力制約下においてメニーコアにおける並列プログラム実行の性能を最大化する Dynamic Core-count and Frequency Scaling（DCFS）手法を提案する．提案手法では，並列プログラムを実行する際にコア数と動作周波数を動的に制御する．具体的には，限られた消費電力バジェットをプログラムの特性に応じてコア数の増加と動作周波数の上昇に適切に配分することで性能向上を狙う．コア数の増加や動作周波数上昇に対するスケーラビリティはプログラムの種類やその実行箇所に応じて異なるため，プログラムの実行中にそれらを動的に変更する必要がある．従来の DVFS では動作周波数と供給電圧のみを動的に変更するのに対し，DCFS では稼働させるコア数も制御することで消費電力バジェットをより効率的に利用する．一定の消費電力制約下において提案手法を評価した結果，従来の全コア実行に比べ最大で 35%の性能向上を達成した．

本論文の構成は以下のとおりである．2 章では，実験環境の説明を行い，実行するプログラムの種類やその実行箇所に応じて性能特性（コア数や動作周波数に対するスケーラビリティ）が異なることを示す．3 章では，提案手法の概要と実装について説明する．4 章では提案手法の評価について述べ，5 章で評価に関する考察を行う．6 章では関連研究を紹介し，最後に 7 章で本論文をまとめる．

## 2. コア数および動作周波数に対するアプリケーションの性能分析

本章では，消費電力制約下においてコア数および動作周波数に対するプロセッサの性能特性がプログラムの種類やその実行箇所に応じて異なることを示す．なお，本実験には PARSEC 2.1 [2] から選択したベンチマーク・プログラムと実機の AMD Opteron を用いた．

### 2.1 実験環境

まず初めに，本実験を行った環境について述べる．実験に用いたシステムの構成は**表 1** のとおりである．本システムは 4 プロセッサによる SMP（Symmetric Multi-Processor）構成で，各プロセッサが 8 コアを搭載したマルチコアであり合計で 32 コアの構成となっている．ベンチマークは `blackscholes`，`dedup`，`x264` の 3 つを選択し，入力サイズはすべて “native” を用いた．

### 2.2 消費電力制約の仮定

本論文では消費電力制約を設定し，プロセッサの消費電力がそれを超えないよう，最大動作周波数がコア数によって決定されると仮定する．消費電力制約としては，式 (1) に示すように全コア（32 コア）が最低動作周波数で稼働する際の動的消費電力とする．ここで，$a$ はスイッチング確率，$N_{allcores}$ はチップ上の全コア数，$C$ は 1 コアあたりの負荷容量，$f_{min}$ は最低動作周波数，$V_{min}$ は最低供給電圧を表す．なお，プロセッサの負荷容量はコア数に比例すると仮定する．

$$P_{constraint} = a \cdot N_{allcores} \cdot C \cdot f_{min} \cdot V_{min}^2 \qquad (1)$$

そして，コア数が $N_{cores}$ の場合の消費電力（式 (2)）がこの制約を超えないよう最大の動作周波数 $f$ と供給電圧 $V$ を選ぶ．つまり，不等式 (3) をつねに満たすようコア数に応じて動作周波数と供給電圧を設定する[*1]．

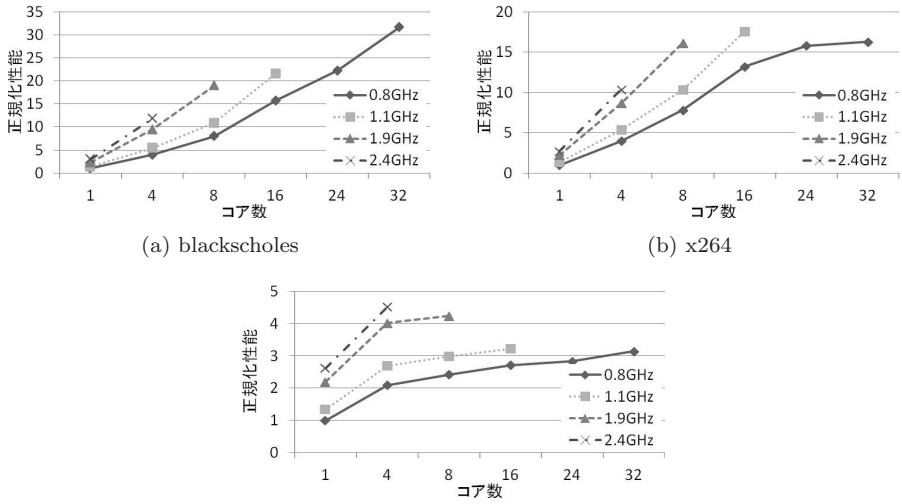
**表 1**　プロセッサの構成
**Table 1**　Configuration of the evaluation system.

| | |
|---|---|
| プロセッサ | AMD Opteron 6136 |
| プロセッサ数 | 4 |
| 1 プロセッサあたりの搭載コア数 | 8 |
| 利用可能な全コア数 | 32（4 × 8） |
| L1 I/D キャッシュ | 128 KB |
| L2 キャッシュ | 512 KB |
| 共有 L3 キャッシュ | 12 MB |
| 主記憶 | 16 GB（DDR3-1333） |
| バススピード | 6.4 GT/s |
| テクノロジーサイズ | 45 nm |

[*1] プロセッサの消費電力が $P_{constraint}$ をけっして超えてはならないと仮定しているため，この最大動作周波数の仮定は保守的なものである．

表 2　消費電力制約下におけるコア数に応じた最大動作周波数と供給電圧，消費電力比
**Table 2**　Maximum CPU frequency, supply voltage and ratio of power consumption under power constraint for each core count.

| コア数 | 動作周波数 [GHz] | 供給電圧 [V] | 制約に対する消費電力比（最悪の場合） |
|---|---|---|---|
| 1 – 5 | 2.4 | 1.3 | 0.88 |
| 6 – 8 | 1.9 | 1.2 | 0.97 |
| 9 – 12 | 1.5 | 1.13 | 0.99 |
| 13 – 19 | 1.1 | 1.04 | 0.97 |
| 20 – 32 | 0.8 | 0.95 | 1 |




(a) blackscholes　(b) x264

(c) dedup

図 1　コア数と動作周波数に応じた性能
**Fig. 1**　Prallelism of three programs from PARSEC with different CPU frequency.

$$P_{N_{cores}} = a \cdot N_{cores} \cdot C \cdot f \cdot V^2 \tag{2}$$

$$\frac{P_{N_{cores}}}{P_{constraint}} = \frac{N_{cores} \cdot f \cdot V^2}{N_{allcores} \cdot f_{min} \cdot V_{min}^2} < 1 \tag{3}$$

**表 2** にコア数に応じた動作周波数と供給電圧，消費電力制約に対する最悪の場合の消費電力比を示す．この消費電力比の値が 1 を下回っていることが消費電力制約を満たしていることを意味する．なお，動作周波数と供給電圧のペアは表 2 に示す 5 通りであり，スイッチング確率 $a$ はコア数によらず一定であると仮定する．

### 2.3　プログラムごとの性能特性

3 つのプログラムにおいて，コア数および動作周波数を変更した場合の性能を**図 1** に示す．グラフの横軸はそれぞれのプログラムに割り当てられるコア数，縦軸は各プログラムの性能（実行時間の逆数）を表している．すべての値は，最低動作周波数（0.8 GHz）で 1 コアにより実行した場合を 1 として正規化したものである[*2]．また，4 種類の線はそれぞれ異なる動作周波数（0.8，1.1，1.9，2.4 GHz）で実行した場合を表しており，各動作周波数で実行可能な最大コア数は消費電力制約により決定される．

図 1 (a) の `blackscholes` では，コア数と動作周波数にほぼ比例した性能向上が得られている．このようなプログラムでは，動作周波数の上昇よりもコア数の増加のほうが性能を効率的に，すなわち少ない消費電力の増加で向上させることができる．これはプロセッサの消費電力が供給電圧の 2 乗と動作周波数に比例するためである．たとえば，性能を 2 倍にするためにコア数もしくは動作周波数を 2 倍にする場合をそれぞれ考える．コア数を 2 倍に増加させると負荷容量の増加により消費電力は 2 倍になる．これに対し，動作周波数を 2 倍に上昇させると供給電圧の上昇をともなうため，消費電力は 2 倍より大きくなる．よって，消費電力制約下において高い並列性を持つプログラムを実行する場合には，低い動作周波数ではあるものの，使用コア数を可能な限り多くすることが得策となる．

一方，コア数増加にともない性能向上が頭打ちになるようなプログラム（図 1 (b) の `x264` や図 1 (c) の `dedup`）を実行する際には，コア数を制限し消費電力バジェットを動作周波数の上昇に用いることで性能を最大化できる．たとえば，`x264` の場合，16 コアを用いた 1.1 GHz での実行により最大性能を達成できる．`dedup` の場合，コア数増加に比べて動作周波数上昇による性能向上が大きいため，最大

[*2] これらすべてのグラフは，評価環境における全コア数と等しい 32 スレッドを生成し，それらを横軸が示す数のコアにバインドして（スレッドパッキング [3]）実行した結果を示す．以降の実験においても，この手法を用いる．

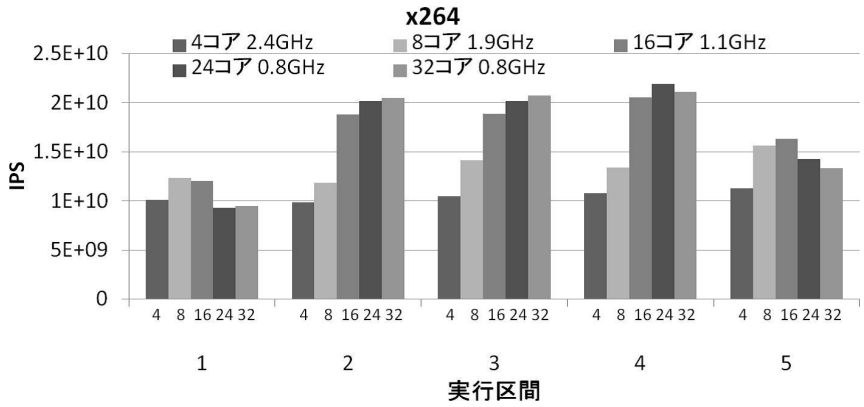



図 2　一定区間ごとの特性（x264）
**Fig. 2**　Performance characteristics of x264 for different phases.

動作周波数である 2.4 GHz で 4 コアにおいて実行した際に性能が最大となる．

### 2.4　プログラム内の性能特性

x264 の実行において一定区間ごとの性能特性を図 **2** に示す．横軸は実行中の連続した 5 つの区間を，縦軸は Instructions Per Second（IPS）を表している．5 種類のバーは，それぞれ異なるコア数（4，8，16，24，32）とそのときの最大動作周波数の組合せで実行した場合の性能である．スレッドパッキングによる実行では，バインドするコア数にかかわらず命令の総数がほぼ一定であるため，IPS が性能を示すのに適した指標となる．

区間 1 では 1.9 GHz の 8 コアによる実行が 5 種類の組合せの中で最大の IPS を達成している．これに対し，2 番目と 3 番目の区間では，最高性能を達成する組合せが 0.8 GHz の 32 コアによる実行となる．また，4 番目と 5 番目の区間では，0.8 GHz の 24 コアと 1.1 GHz の 16 コアによる実行がそれぞれ性能を最大化している．この結果から，それぞれの区間によって適したコア数と動作周波数が異なることが分かる．

図 1 と図 2 の結果から，消費電力制約下において性能を最大化するためには，最適なコア数と動作周波数を動的に制御する手法が必要であるといえる．そこで本論文では，アプリケーションの特性（コア数と動作周波数に対する性能特性）をプログラムの実行中に検知し，その特性に応じてコア数と動作周波数の適した組合せを選択する手法を提案する．次の章で，その詳細を説明する．

## 3.　提案手法：Dynamic Core-count and Frequency Scaling（DCFS）

### 3.1　概要

提案手法の目的は，消費電力制約下でのメニーコアにおける並列プログラム実行を高速化することである．従来の並列プログラム実行では，利用可能なすべてのコアを使用するために，必要な数（全コア数と同じかそれ以上）のスレッドを生成し並列処理を行う．しかしながら，前の章で示したように，全コアによる実行が必ずしも最大性能を達成できるとは限らない．

そこで本論文では，実行するプログラムの特性に応じて，コア数と動作周波数を動的に変更する Dynamic Core-count and Frequency Scaling（DCFS）手法を提案する．提案手法では図 1 (a) の `blackscholes` のような高い並列性を持つプログラムを実行する場合，可能な限り多くのコアを用いて並列処理を行う．それに対して，`x264` や `dedup` のような並列性の低いプログラムを実行する場合には，使用するコア数を減少させ，その分の消費電力バジェットを動作周波数上昇に利用することで消費電力制約下における性能の最大化を狙う．

### 3.2　コア数・動作周波数決定法

提案する DCFS 手法は，“トレーニングフェイズ” と “実行フェイズ” と呼ばれる 2 種類のフェイズから成り立つ．トレーニングフェイズでは，ある短い時間ごとに構成（コア数と動作周波数の組合せ）を変更しつつプログラムを実行し，特性を調べるための指標として IPS を測定・記録する．実行フェイズでは，測定した IPS から性能を最大化する構成を推測し，その構成によりプログラムが実行される．また，一定時間ごとに IPS を再計測することにより，プログラムの特性の変化を検知する．このトレーニングフェイズと実行フェイズはプログラムの実行終了まで繰り返される．本研究では，最適な構成を探索するアルゴリズムとして “全探索法” と “ヒルクライム法” の 2 種類を実装した．なお，この手法はすべて動的なものであり，プログラムの静的な解析やプログラム自体の修正はいっさい必要ないことに注意されたい．この手法の概要を図 **3** に示し，以降で詳細な説明を行う．

- トレーニングフェイズ
  トレーニングフェイズでは，最適な構成を選択するた

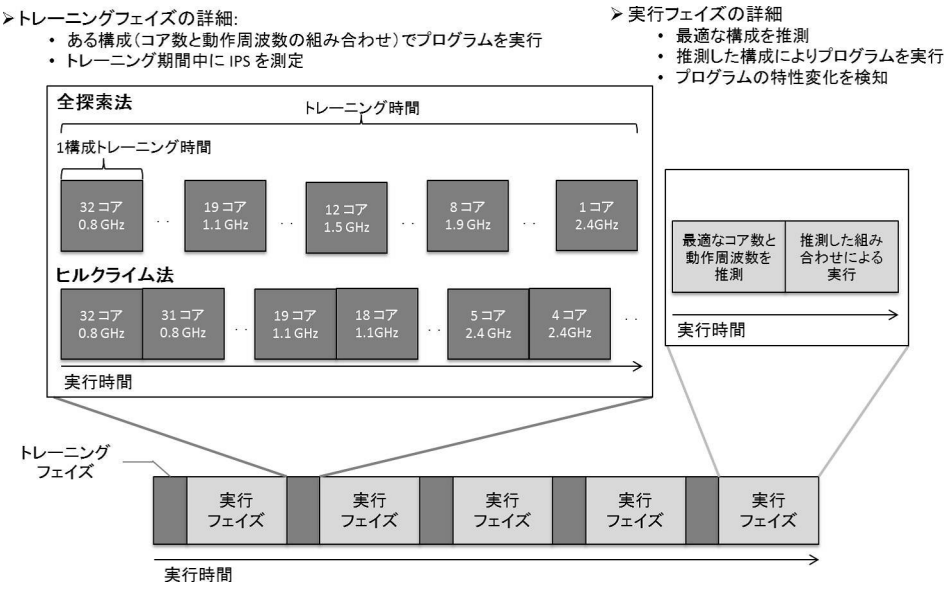



図 3　提案手法の概要
**Fig. 3**　Overview of proposal technique.

めに，構成を動的に変更しつつ IPS を測定する．全探索法では，コア数を全コア数から 1 まで変化させ，全通りの性能を計測する．なお，各コア数において，そのコア数に応じた最大動作周波数のみで計測を行う．また，ヒルクライム法では，利用可能な動作周波数ごとにコア数を全コア数から減少させつつ IPS を計測し，IPS が最大になるコア数を探索する．以下，各アルゴリズムの詳細な説明を行う．

– **全探索法**
まず初めに，全コア数とそのときの最大動作周波数（図 1 の例では，32 コアと 0.8 GHz の組合せであり，以降の説明においても図 1 の例を用いる）によりプログラムを実行し，ある一定時間（“1 構成トレーニング時間” と呼称）IPS を計測・記録する．次に，コア数を 1 減少させ，動作周波数をそのときの最大動作周波数に上昇させる．そして，1 構成トレーニング時間中に IPS を再び計測・記録する．これをコア数が 1 になるまで繰り返す（コア数が 1 のときの動作周波数は 2.4 GHz）．

– **ヒルクライム法**
全探索法と同様に，まず，全コア数とそのときの最大動作周波数（32 コアと 0.8 GHz）によりプログラムを実行し，1 構成トレーニング時間中に IPS を計測・記録する．次に，動作周波数は一定のまま，使用するコア数を減少させ（24 コアと 0.8 GHz），IPS を得る．そして，IPS が低下するまでコア数を減少させ，現在の動作周波数（0.8 GHz）において最大性能を達成する構成を探索する．なお，この手法は IPS をコア数の関数とした場合に，この関数が単峰性関数となっていることを仮定しており，IPS が低下する直前のコア数をその周波数で最大の性能を達成するコア数と見なす．続いて動作周波数を上昇させ（1.1 GHz），そのときに最大限利用できるコア数から IPS が減少するまでコア数を減少させつつ，IPS を得る．これを利用可能なすべての動作周波数（0.8，1.1，1.9，2.4 GHz）に関して繰り返す．

- **実行フェイズ**
トレーニングフェイズの後，記録したそれぞれの IPS を比較し，最大の IPS を達成した構成を現時点の最適な構成として選択し，この構成でプログラムを実行する．また，プログラムの性質が図 2 で見たようにプログラムの実行中に変化することが考えられる．このため，一定時間ごと（今回は 1 秒ごと）に IPS を計測し，現在の IPS が前回の IPS と比較して一定範囲以上変化すれば，プログラムの特性が変化したと見なし再びトレーニングフェイズに移行する．

トレーニングフェイズでは，最適でない構成においてもプログラムを実行するため，探索時にオーバヘッドが発生する．よって，探索する構成の数が多いほど 1 度のトレーニングに要する時間（“トレーニング時間” と呼称）が長くなりオーバヘッドが大きくなる．提案手法では，最適な構成を選択する精度を保ちつつ，トレーニングフェイズにおいて探索する構成の数を減らすことでトレーニング時間を短縮し，さらなる性能向上を得ることができる．

全探索法では全コアから 1 コアまでの構成において性能を計測するため，今回の実験環境（2.1 節参照）では，1 度のトレーニングフェイズにおいて探索する構成の数は 32 である．一方，ヒルクライム法では利用できる動作周波数の数が 5 通りであり，各動作周波数において 2 から 4 構成

を探索するため，1 度のトレーニングフェイズにおいて探索する構成数は 10 から 20 程度である．そのため，ヒルクライム法の適用により全探索法に比べオーバヘッドを削減できる．将来のメニーコアでは，選択可能な動作周波数の数の増加に対し，チップ上のコア数が大幅に増加することが予想される．そのため，今後ますますヒルクライム法の有用性が高くなる．

精度に関しては，コア数を全通り探索する全探索法が最も優れているが，ヒルクライム法でも全探索法と同等の精度が得られる．なぜならヒルクライム法では，IPS をコア数の関数とした場合に単峰性関数となることを仮定しており，図 1 から分かるように実際のベンチマークにおいてもその仮定が正しいといえるためである．

本手法を適用する際，プログラムの振舞いが変化した後にその振舞いが 1 度のトレーニングフェイズに要する時間に比べ十分長い時間継続する場合にトレーニングフェイズで予測した最適な構成での実行が可能となる．しかしながら，プログラムの振舞いが短い時間間隔で変化することもありうる．このような場合，トレーニングフェイズの回数が多くなり，最適でない構成での実行により性能が低下する．また，トレーニングフェイズに続く実行フェイズにおいてプログラムの振舞いが変化する場合，最適な構成で実行フェイズを実行することができなくなるため，性能が低下するといった問題が発生する．

### 3.3 実装

本論文では，実装を簡単化するために，本手法をユーザレベルのランタイムシステムとして実装した．具体的には，ハードウェアカウンタの値を読み出しプロファイリングを行うための Linux の標準ソフトウェアである `perf tool` を改良し，タイマにより定期的に実行命令数を計測し，またコア数と動作周波数を制御するハンドラに制御を移す機能を実装した．コア数を指定するためには，生成したスレッドを特定のコアに割り当てるための Linux 標準 API である `sched_setaffinity(2)` を用いた．また，動作周波数の変更に関しては，/sys/devices/system/cpu/cpuX/cpufreq/scaling_setspeed（X は CPUID）への動作周波数の値の書き込みで実装した．

提案手法では，1 構成トレーニング時間が性能を決定する重要なパラメータとなる．なぜなら，この時間が長すぎる場合には最適でない構成での実行が性能へ与える影響が大きくなり，短すぎる場合には動作周波数とコア数を変更した後にプロセッサの挙動が不安定になり正確な IPS を測定できないことがあるためである．動作周波数とコア数の変更後，プロセッサの挙動が安定するまで IPS の測定を待機しなければならない．ここでのプロセッサの挙動が不安定になるとは動作周波数とコア数を変更した直後に IPS の値が変動することであり，プロセッサの挙動が安定すると は IPS の値がある一定の値（構成の変更が完了したことを意味する値）に収束することである．動作周波数を変更する際には供給電圧の変更が行われるため，供給電圧が安定するまでの間 IPS が変動する可能性がある．しかしながら，供給電圧の変更は数十マイクロ秒で完了するため，ミリ秒単位で計測する IPS にはほとんど影響しない．これに対し，コア数の変更時に行われるスレッドのマイグレーションには数十ミリ秒ほどの時間を要するため，計測する IPS に対し影響を及ぼすと考えられる．本論文の実験では，4 プロセッサ（1 プロセッサあたり 8 コアを搭載）から構成されるプラットフォームを用いている．共有 L3 キャッシュはプロセッサごとに搭載されているため，使用するコア数を 8 から 32 に変更する場合にスレッドのスケジューリングやキャッシュミス率の増加といった理由からスレッドのマイグレーションに最も長い時間を要すると予想できる．そこで，この場合において動作周波数・コア数変更後に IPS の値がある一定の値に収束する時間を計測したところ，最悪の場合で 30 ms であった．そのため，コア数と動作周波数を変更した後の 30 ms 間はデータを収集しないこととした．ただし，スレッドのマイグレーション後のキャッシュミス率増加が計測する IPS に与える影響はプログラムによって異なるため，この 30 ms という値は今回用いたベンチマーク一式特有の値である．そのため，他のベンチマーク一式を用いて評価を行う際，構成変更後に IPS が収束する時間を同様に計測する必要がある．

また，探索時の性能低下を最小化するために，IPS の値が安定した後に現在の構成でプログラムを実行しつつ IPS を計測する時間も可能な限り短くしなければならない．本研究では提案手法をユーザレベルのソフトウェアで実装しており，定期的な IPS の計測や構成変更を実現するために nanosleep(2) システムコールを使用している．そこで，本実験で用いたプラットフォームにおいて nanosleep を用いて妥当な IPS を計測できる最短の時間を計測した結果，30 ms となった．そのため，構成変更後に IPS の値が安定するまで 30 ms 待機し，その後の 30 ms において現在の構成での IPS を計測することとした．つまり，1 構成トレーニング時間は 60 ms である．実行フェイズでは，現在の IPS が前回の IPS と比較し，10%以上増減すればプログラムの特性が変化したと見なし，トレーニングフェイズに移行する．

## 4. 性能評価

### 4.1 評価環境

本章では，実機による提案手法（DCFS）の評価について述べる．評価に用いたシステムの構成は 2.1 節に示したものと同様であり，消費電力制約とコア数に応じた動作周波数は 2.2 節で説明したものと同様である．プログラムは PARSEC 2.1 [2] から `facesim`，`fluidanimate`，`raytrace`

表 3　各ベンチマークの分類
**Table 3**　Classification of the evaluated benchmarks.

| 並列性 | ベンチマーク | 32 コア実行時の<br>1 コア実行時に対する性能比 | 2.4 GHz 実行時の<br>0.8 GHz 実行時に対する性能比 |
|---|---|---|---|
| 高 | blackscholes | 31.6x | 2.98x |
| | swaptions | 31.6x | 2.96x |
| | vips | 29.7x | 2.94x |
| 中 | ferret | 21.4x | 2.98x |
| | freqmine | 18.4x | 2.96x |
| | x264 | 16.3x | 2.78x |
| | canneal | 13.0x | 1.61x |
| | bodytrack | 12.4x | 2.99x |
| 低 | dedup | 3.1x | 2.86x |
| | streamcluster | 2.9x | 1.95x |

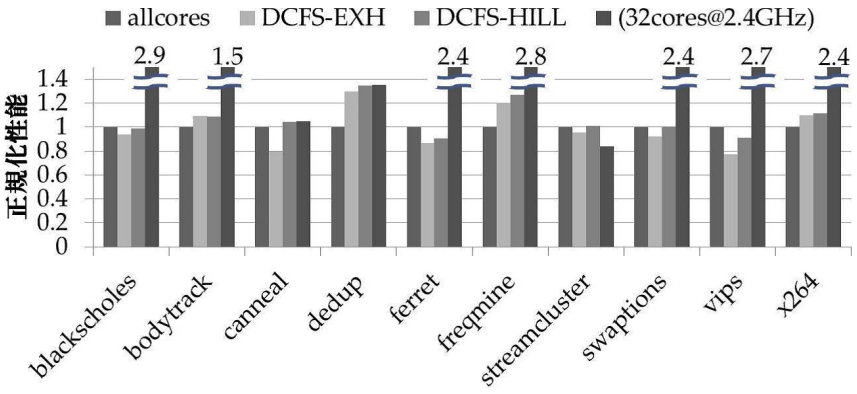



図 4　提案手法の評価結果
**Fig. 4**　Performance normalized to the minimum frequency execution with all cores.

を除いた 10 個を用いる．`facesim` と `fluidanimate` を実行する際，コア数が 2 のべき乗である場合しか正しく動作しない．本評価に用いるプラットフォームでは全コア数が 32 であり提案手法適用時の選択可能なコア数が 6 通りに限定されるため，これら 2 つのプログラムは除外した．また，`raytrace` はコンパイルできなかったため評価に用いていない．なお，すべての評価において “native” の入力サイズを用いた．

これらのプログラムを並列性に応じて分類するために，各プログラムを評価プラットフォームにおいて 32 コアと 1 コアで実行し，1 コア実行に対する 32 コア実行時の性能比を算出した．また，動作周波数が 0.8 GHz と 2.4 GHz のそれぞれの場合に各プログラムを 1 コアで実行し，0.8 GHz での実行に対する 2.4 GHz での実行の性能比を求めた．その結果を**表 3** に示す．`blackscholes`，`swaptions`，`vips` の並列性が高いプログラムにおいては従来型の実行方法である全コアでの実行によって高い性能が予想されるため，提案手法では性能を改善する余地が少ないと考えられる．一方で，並列性が中または低いプログラムにおいては，最適な構成が全コアでの実行ではない場合があると予想されるため，提案手法による性能改善が期待される．また，提案手法ではコア数増加に加えて動作周波数上昇により性能向上を狙うため，`canneal` や `streamcluster` のような（動作周波数上昇による性能向上が小さい）プログラムでは提案手法による性能向上が小さいことが考えられる．これに対し，それ以外のプログラムでは動作周波数上昇に比例した性能向上が得られると予想できる．

### 4.2　評価結果

提案手法を従来の全コア実行（動作周波数は最低動作周波数の 0.8 GHz）と比較して評価を行う．提案手法については，全探索法とヒルクライム法（それぞれ，DCFS-EXH と DCFS-HILL）の 2 種類の手法の性能を評価する．評価結果を**図 4** に示す．横軸はベンチマーク，縦軸は従来の全コア実行時（0.8 GHz の 32 コア）の性能（実行時間の逆数）で正規化した性能を表している．また，最も右側のバーは全コアを最高周波数で実行した場合（2.4 GHz の 32 コア）の正規化性能を示しており，バーの上の数字は 1.4 倍以上の性能の値である．ただし，この構成で実行する際の消費電力は本論文で仮定した消費電力制約を超えるため，あくまでも参考値となる．

高い並列性を持つプログラムである `blackscholes`，`swaptions`，`vips` については，提案手法のどちらの手法でも性能向上が得られていないか，もしくは性能が悪化

している．また，動作周波数上昇による性能向上が小さいプログラムである canneal と streamcluster についても，性能向上はほとんど得られていない．これらについては 4.1 節で理由を述べたとおりであるが，canneal と streamcluster に関して次の章で詳しい解析を行う．さらに，ferret では，並列性の低さと動作周波数上昇による性能向上の大きさにもかかわらず提案手法適用により性能は向上しなかった．この理由についても次章で述べる．

一方で bodytrack，dedup，freqmine，x264 では，提案手法により性能が向上した．表 3 から分かるように，これらは低・中程度の並列性を持つプログラムである．特に並列性の低い dedup では，最大で 35%の性能向上が得られた．次節で，提案手法を適用した dedup の実行について詳しく解析する．評価に用いた全 10 個のプログラムにおいて，全探索法では平均 2%性能が悪化し，ヒルクライム法では平均 6%性能が向上した．また，性能向上が得られた 4 つのプログラムでの性能向上比の幾何平均は，全探索法で 17%，ヒルクライム法で 20%である．

bodytrack を除く全プログラムにおいて，全探索法に比べヒルクライム法のほうがより大きな性能向上を達成している．これは 3.2 節で述べたように，ヒルクライム法は全探索法に比べ 1 度のトレーニングフェイズにおいて探索する構成数が少なく，オーバヘッドが小さいためである．**表 4** にそれぞれの手法においてトレーニングに要した合計時間の実行時間に対する割合を示す．全探索法に比べヒルクライム法のほうがその割合を大幅に削減できていることが分かる．しかしながら，ferret や x264 では依然として実行時間の約 15%近くをトレーニングに費やしている．より高い性能を達成するためには，さらにトレーニング時間を短縮できる探索アルゴリズムを考案する必要がある．また，図 4 の結果から，ヒルクライム法では最適なコア数と動作周波数決定の精度を維持できていることが分かる．

## 5. 評価に関する考察

### 5.1 性能向上が得られなかったプログラムに関する考察

提案手法の特性を理解するために，ferret，canneal，streamcluster について解析を行う．ferret の並列性は中程度であるにもかかわらず，提案手法による性能向上は得られなかった．コア数と動作周波数に応じた ferret 実行時の性能を**図 5** に示す．図の見方は，図 1 と同様である．この結果から，コア数増加にともない性能向上が得られるため，全コア実行により性能が最大となることが分かる．このようなプログラムの実行に提案手法を適用すると，トレーニングフェイズのオーバヘッドにより性能が悪化する．よって，提案手法のトレーニングフェイズに要する時間をさらに短縮することで性能悪化を緩和できるといえる．本論文における実装では，ランタイムシステムはユーザランドで動作するプロセスであり，プロセッサ・アフィニティやコアの動作周波数を変更するためのシステムコールによりオーバヘッドが発生する．そこで今後の課題として，このシステムを OS のカーネルに実装することでオーバヘッドを短縮する．

また，canneal と streamcluster も低い並列性を持つプログラムであるにもかかわらず，提案手法による性能向上は得られていない．Bienia らの研究によると，canneal と streamcluster は評価に用いたプログラムの中で，最

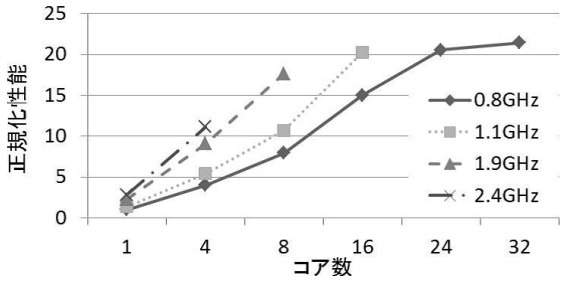
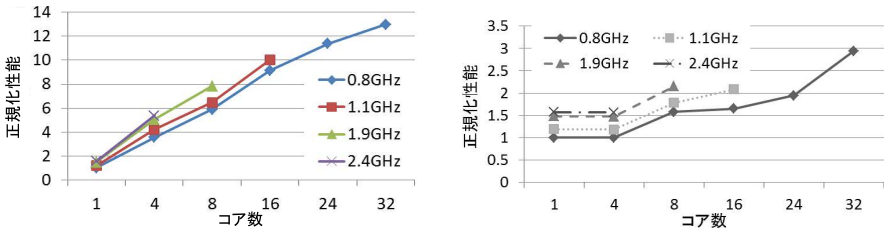



図 5　ferret におけるコア数と動作周波数に応じた性能
**Fig. 5**　Prallelism of ferret with different CPU frequency.

**表 4**　各ベンチマークのトレーニングに要した合計時間の実行時間に対する割合
**Table 4**　Fractions of total trainging time to execution time for each benchmark.

| ベンチマーク | トレーニングに要した合計時間の実行時間に対する割合（%） | |
|---|---|---|
| | DCFS-EXH | DCFS-HILL |
| blackscholes | 11.5 | 4.0 |
| bodytrack | 10.2 | 2.2 |
| canneal | 9.4 | 6.9 |
| dedup | 15.9 | 6.9 |
| ferret | 33.3 | 14.8 |
| freqmine | 13.4 | 6.0 |
| streamcluster | 8.1 | 5.4 |
| swaptions | 6.7 | 1.3 |
| vips | 8.0 | 2.8 |
| x264 | 38.4 | 18.8 |




(a) canneal　(b) streamcluster

図 6　メモリバウンドなプログラムにおけるコア数と動作周波数に応じた性能特性

**Fig. 6**　Prallelism of Memory-bound applications with different CPU frequency.

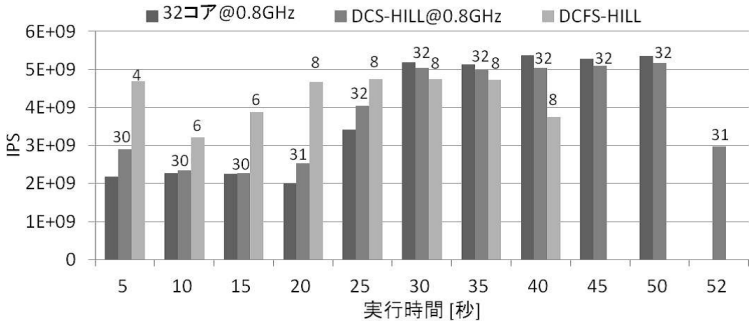



図 7　dedup における全コア実行と提案手法の比較

**Fig. 7**　Comparison between execution with all 32 cores and proposal technique.

もメモリバウンドな 2 つのプログラムである [1]．DCFS は，コア数と動作周波数の制御により性能向上を狙う手法である．しかしながら，メモリバウンドなプログラムでは，動作周波数の変化による性能への影響が CPU バウンドなプログラムと比較して小さいため，動作周波数上昇による性能向上は期待できない．これは表 3 の 0.8 GHz での実行に対する 2.4 GHz の実行の性能比からも分かる．**図 6** は，図 1 と同様に，`canneal` と `streamcluster` の正規化性能を示したものである．この結果から，図 1 に示したプログラムに比べ，動作周波数上昇による性能向上が小さいことが分かる．このような場合，命令数だけでなくラストレベルキャッシュのミス数を監視することで，メモリバウンドなプログラムの特性を検知できる．その情報を利用すれば，コア数を動的に制御することで消費電力の削減が可能となる．

### 5.2　DCFS による性能向上に関する考察

DCFS により最大の性能向上が得られた `dedup` において，DCFS の効果を詳細に分析する．**図 7** は，0.8 GHz の全コア実行（32 コア @ 0.8 GHz）とコア数のみをヒルクライム法で制御する手法（DCS-HILL @ 0.8 GHz），提案手法（DCFS-HILL）の 3 種類の実行方式で `dedup` を実行した際の実行時間ごとの IPS を示している．また，それぞれのバーの上の数字はコア数を表す．DCS-HILL @ 0.8 GHz の実行時間はトレーニングフェイズのオーバヘッドにより全コア実行と比べわずかに延長しているが，5〜20 秒の区間においてコア数を制限することで IPS の向上を達成している．提案手法である DCFS-HILL では，使用しないコアの消費電力バジェットを動作周波数上昇に再割当てすることで IPS をさらに向上させることができている．この結果から，DCFS の適用によりトレーニングフェイズのオーバヘッドを補うのに十分な性能向上が得られるといえる．

## 6.　関連研究

この章では，本研究に関連する 3 つの文献を紹介し，本研究と比較する．

### 6.1　Feedback-Driven Threading [10]

Suleman らによる Feedback-Driven Threading（FDT）は，プログラム実行中に収集した情報を基にスレッド数を動的に変更することで，並列アプリケーション実行時の性能向上や消費電力削減を狙う手法である．FDT は，同期処理に要する時間やオフチップ・バスのバンド幅に対する要求からプログラムの特性を解析し，ループ処理に適したスレッド数を予測する．並列アプリケーション実行時の性能は，スレッド数増加にともない必ずしも向上するとは限らない．たとえば，共有データを頻繁に扱うアプリケーションの場合，同期処理によりスレッド数増加にともなう性能向上が頭打ちになったり性能が悪化したりする．また，共有データをほとんど扱わないプログラムでも，オフチップ・バスのバンド幅の制限によりスレッド数増加にともない性能向上が頭打ちとなる場合がある．

FDT では，コンパイラがループ処理を 2 つの部分に分割する．1 つは，クリティカルセクションの実行に要する

時間やオフチップ・バスの使用率を測定し，その結果を基にそのループ処理に適したスレッド数を推測する部分である．もう 1 つの部分では，推測したスレッド数でプログラムを並列実行する．スレッド数増加にともない性能が悪化する場合，FDT によって性能向上と消費電力削減の両方を達成できる．しかしながら，スレッド数増加にともない性能向上が頭打ちになる場合には，消費電力を削減できるが，性能を向上させることはできない．本論文の提案手法である DCFS では，性能向上に貢献しないコアに元々割り当てられていた消費電力バジェットを残りのコアに再割当てすることでコアの動作周波数を上昇させることができ，性能向上を達成できる．

### 6.2 Intel Turbo Boost 技術 [6]

Turbo Boost（TB）技術は，コアの動作周波数を定格周波数以上に上昇させ性能向上を達成する手法である．プロセッサの消費電力が TDP（Thermal Design Power）未満であるときに，コアの動作周波数を動的かつ自動的に定格周波数以上に上昇させる．消費電力をつねに監視することで，プロセッサの状態に応じて動作周波数を変更できる．

TB により得られる性能向上の度合いは，稼働しているコアの数に強く依存する．たとえば，1 つのコアのみが稼働している場合，消費電力は TDP を大きく下回るため動作周波数を大幅に上昇させることができる．一方，全コアが稼働している場合，消費電力が TDP に近い値となるため動作周波数を大幅に上昇させることができない．そのため，TB は，いくつかのコアが休止状態である場合により大きな効果を発揮する．

メニーコアにおいて並列アプリケーションを実行する際，1 コアのみが処理を行う逐次処理部分の実行は TB により高速化できる．しかしながら，並列処理部分では，OS がチップ上の全コアにスレッドを割り当てるため，TB による性能向上はあまり期待できない．提案手法である DCFS では，アプリケーションの特性に応じて稼働させるコアの数を動的に変更できる．よって，並列性の低いプログラムを実行する場合，いくつかのコアを休止させることでその他のコアの動作周波数を上昇させることができ，従来の TB 以上の性能向上を得ることができる．

### 6.3 Pack & Cap [3]

Cochran らによる Pack & Cap と呼ばれる手法では，本研究と同様にコア数と動作周波数の制御を行っている．彼らの手法の目的は，実行時間中に変化する消費電力制約に対し性能を最大化することである．

彼らの研究と本研究の主な違いは 2 つある．1 つ目は，彼らの研究では消費電力制約下において最適なコア数と動作周波数を推測するために，静的な解析を行っていることである．その静的解析で得られた情報とプログラム実行中に収集した情報を基に，最適な構成を動的に決定する．一方，本研究の提案手法では，最適な構成を決定するために静的な情報はいっさい必要ない．多少のオーバヘッドはともなうが，動的に収集した情報のみを用いてコア数と動作周波数を制御できる．2 つ目の違いは，彼らは 4 コアのプロセッサを用いて評価を行っており，本論文の評価で見られるようなコア数が数十に増加した際のスケーラビリティの問題を考慮していないことである．これは，将来メニーコアが一般化した際に重要となる問題であり，コア数を考慮してメニーコアの性能や消費電力を制御する本手法は有用であるといえる．

## 7. おわりに

現在，微細化技術の発達にともない，1 つのチップに数十のコアを搭載したメニーコアが登場している．一方で，消費電力削減が求められているため，今後のプロセッサでは厳しい消費電力制約下においていかに性能を最大化するかが課題となる．この課題に対して有効な手法として，Dynamic Voltage and Frequency Scaling（DVFS）があげられる．DVFS を適用することで逐次プログラムの実行だけでなく，メニーコアにおける並列プログラムの実行も効率化できる．ただし，そのような場合，メニーコアの性能を決定する要因として動作周波数だけでなくプログラムの並列性も考慮しなければならない．なぜなら，チップ上の全コアを用いた実行が最大性能を達成できない場合が存在するためである．

そこで，本論文では並列プログラム実行時に消費電力制約下において性能の最大化を狙う Dynamic Core-count and Frequency Scaling（DCFS）手法を提案した．提案手法では，消費電力制約下において，プログラムの特性に応じてコア数と動作周波数を動的に制御する．また，本 DCFS 手法は “トレーニングフェイズ” と “実行フェイズ” の 2 つのフェイズから構成され，プログラム内で変化する特性に対応できる．なお，対象プログラムの静的解析や修正はいっさい必要としない．PARSEC ベンチマーク・プログラムを用いた実機による評価では，チップ上の全コアによる実行に比べ，最大で 35%の性能向上を達成した．また，評価に用いた全 10 個のプログラムにおける性能向上の平均は 6%であり，性能向上が得られた 4 つのプログラムの実行においては，平均 20%であった．今後の課題として，5.1 節で述べたように，提案手法を実現するためのランタイムシステムを OS のカーネルに実装することでトレーニングフェイズのオーバヘッドを削減する．また，本論文ではある一定の消費電力制約を仮定し提案手法の評価を行ったが，異なる消費電力制約においても評価を行う．さらに，提案手法適用時の消費電力および消費エネルギーも評価する．

**謝辞**　本研究は，一部，独立行政法人新エネルギー・産

業技術総合開発機構（NEDO）ならびに科学研究費補助金（課題番号：21680005）の支援による．

## 参考文献


[1] Bienia, C., Kumar, S. and Li, K.: PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors, *IEEE International Symposium on Workload Characterization, 2008, IISWC 2008*, pp.47–56, IEEE (2008).

[2] Bienia, C., Kumar, S., Singh, J. and Li, K.: The PARSEC benchmark suite: Characterization and architectural implications, *Proc. 17th international conference on Parallel architectures and compilation techniques*, pp.72–81, ACM (2008).

[3] Cochran, R., Hankendi, C., Coskun, A. and Reda, S.: Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps, *Proc. 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '44*, Washington, pp.175–185, DC, USA, IEEE Computer Society (2011).

[4] Herbert, S. and Marculescu, D.: Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, *ACM/IEEE International Symposium on Low Power Electronics and Design* (*ISLPED*) *2007*, pp.38–43, IEEE (2007).

[5] Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G., et al.: A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS, *Solid-State Circuits Conference Digest of Technical Papers* (*ISSCC*), *2010 IEEE International*, pp.108–109, IEEE (2010).

[6] Intel® Corporation: Intel® Turbo Boost Technology in Intel® Core Microarchitecture (Nehalem) Based Processors. Whitepaper, Intel® Corporation (November 2008).

[7] Ramey, C.: TILE-Gx100 ManyCore Processor: Acceleration Interfaces and Architecture, *Hot Chips 23* (2011).

[8] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T. and Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing, *SIGGRAPH '08* (2008).

[9] Semeraro, G., Magklis, G., Balasubramonian, R., Albonesi, D., Dwarkadas, S. and Scott, M.: Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling, *Proc. 8th International Symposium on High-Performance Computer Architecture, 2002*, pp.29–40, IEEE (2002).

[10] Suleman, M., Qureshi, M. and Patt, Y.: Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on CMPs, *ACM SIGPLAN Notices*, Vol.43, No.3, pp.277–286 (2008).



**今村 智史**

昭和 63 年生．平成 23 年九州大学工学部電気情報工学科卒業．平成 23 年同大学大学院修士課程に進学，現在に至る．マルチコア/メニーコア・プロセッサに関する研究に従事．



**佐々木 広**　（正会員）

2003 年東京大学工学部計数工学科卒業．2005 年同大学大学院情報理工学系研究科修士課程修了．2008 年同大学院工学系研究科博士課程修了．博士（工学）．同年東京大学先端科学技術研究センター特任助教，2010 年より東京大学大学院情報理工学系研究科特任助教を経て，現在，九州大学大学院システム情報科学研究院特任准教授．計算機アーキテクチャ，オペレーティングシステムの研究に従事．IEEE，ACM，USENIX 各会員．



**福本 尚人**　（正会員）

昭和 59 年生．平成 19 年九州大学工学部電気情報工学科卒業．平成 21 年同大学大学院システム情報科学府修士課程修了．平成 24 年同大学院システム情報科学府博士後期課程修了．博士（工学）．同年富士通（株）に入社，現在に至る．マルチコア・プロセッサに関する研究に従事．



**井上 弘士**　（正会員）

昭和 46 年生．平成 8 年九州工業大学大学院情報工学研究科修士課程修了．同年横河電機（株）入社．平成 9 年より（財）九州システム情報技術研究所研究助手．平成 11 年の 1 年間 Halo LSI Design & Device Technology, Inc. において訪問研究員としてフラッシュ・メモリの開発に従事．平成 13 年九州大学において工学博士を取得．同年福岡大学工学部電子情報工学科助手．平成 16 年より九州大学大学院システム情報科学研究院助教授．平成 19 年 4 月より同大学准教授，現在に至る．高性能/低消費電力プロセッサ/メモリ・アーキテクチャ，ディペンダブル・アーキテクチャ，3 次元積層アーキテクチャ，性能評価等に関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．

**村上 和彰**　（正会員）

昭和 35 年生．昭和 59 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年富士通（株）入社．汎用大型計算機の研究開発に従事．昭和 62 年九州大学助手．平成 6 年九州大学助教授，情報基盤研究開発センター長，情報統括本部長．現在，九州大学大学院システム情報科学研究院情報知能工学部門教授．計算機アーキテクチャ，並列処理，システム LSI 設計技術，等に関する研究に従事．工学博士．平成 3 年情報処理学会研究賞，平成 4 年情報処理学会論文賞，平成 9 年坂井記念特別賞，平成 12 年日経 BP 社 IP アワード，平成 12 年情報処理学会創立 40 周年記念論文賞，平成 14 年電子情報通信学会業績賞をそれぞれ受賞．