Vol.2012-ARC-200 No.17
Vol.2012-OS-121 No.17
2012/5/8

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]　佐々木 広[2]　福本 尚人[1]　井上 弘士[2]　村上 和彰[2]

概要：本論文では，消費電力制約下において並列プログラム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コア数と動作周波数の最適な組み合わせを予測し，実行フェイズではその最適な組み合わせによっ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．計 12 個の PARSEC ベンチマークを用いた評価の結果，従来の全コア実行に対し平均で 22%，最適な組み合わせを静的に選択した場合の実行に対し平均で 9%の性能向上を達成した．

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，スレッドパッキング

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ースカラ・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，メニーコアと略称）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更に増加すると予想される [7], [12], [13]．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ー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如何にプロセッサの性能を最大化するかが課題となる [5], [8]．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プログラム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プログラム実行を実現するための既存手法として，Dynamic Voltage and Frequency Scaling (DVFS) が挙げられる [2], [14]．DVFS によりプロセッサの電力効率を向上できるため，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 [6], [7]．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，プログラムの並列性を考慮しつつ性能を決定する要因として動作周波数のみならずコア数も同時に制御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 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

[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
[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

表 1 実験に用いるプラットフォームの構成
Table 1 Configuration of the experimental system

プロセッサ	AMD Opteron 6282 SE
プロセッサ数	4
1 プロセッサ当たりの搭載コア数	16
利用可能な全コア数	64（4 × 16）
L1 I/D キャッシュ	48 KB
L2 キャッシュ	1 MB
共有 L3 キャッシュ	16 MB
主記憶	32 GB (DDR3-1333)
バススピード	6.4 GT/s
テクノロジーサイズ	32 nm

実行に比べ平均で 22%，コア数と動作周波数の最適な組み合わせを静的に選択する実行に比べ平均で 9%の性能向上を達成した．

本論文の構成は以下の通りである．第 2 章では，実験環境と消費電力制約の説明を行い，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．第 3 章では，提案手法の概要と実装について説明する．第 4 章では提案手法の評価について述べ，評価に関する考察を行う．第 5 章では関連研究を紹介し，最後に第 6 章にて本論文をまとめる．

2. コア数および動作周波数に対するプログラムの性能分析

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並列プログラムの性能特性がそ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 [1] 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

実験に用いたプラットフォームの構成は**表 1** の通りである．本プラットフォームは 4 プロセッサによる SMP（Symmetric Multi-Processor）構成で，各プロセッサが 16 コアを搭載したマルチコアであり合計で 64 コアの構成となっている．ベンチマークは `blackscholes`, `x264`, `dedup`, `freqmine` の 4 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 64 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 コア当たりの負荷容量，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し，使用しないコアはパワーゲーティングにより電源が遮断されると仮定する．

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 for each core count under power constraint

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 – 18	2.6	1.30	0.98
19 – 24	2.3	1.21	0.99
25 – 32	2.0	1.12	0.99
33 – 48	1.7	1.00	0.99
49 – 64	1.4	0.95	1

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

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) を常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]．

$$P_{N_{cores}} = a \cdot N_{cores} \cdot C \cdot f \cdot V^2 \qquad (2)$$

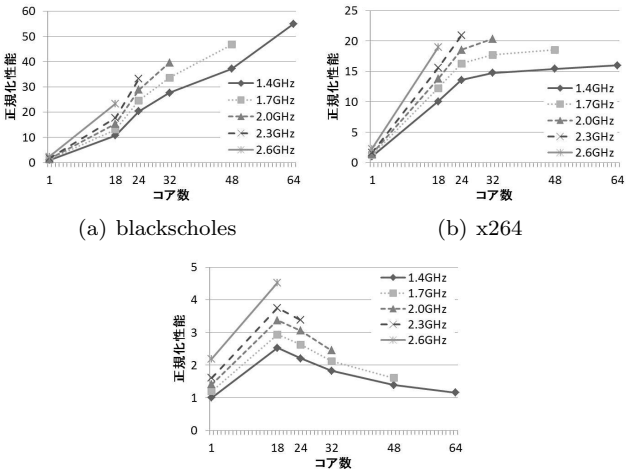
$$\frac{P_{N_{cores}}}{P_{constraint}} = \frac{N_{cores} \cdot f \cdot V^2}{N_{allcores} \cdot f_{min} \cdot V_{min}^2} < 1 \qquad (3)$$

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 はコア数に依らず一定であると仮定する．

2.3 プログラム毎の性能特性

`blackscholes`, `x264`, `dedup` 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．全ての値は，最低動作周波数（1.4 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，5 種類の線はそれぞれ異なる動作周波数（1.4，1.7，2.0，2.3，2.6GHz）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．

[*1] プロセッサの消費電力が $P_{constraint}$ を決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．
[*2] これら全てのグラフは，評価環境における全コア数と等しい 64 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．



(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

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できる．なぜなら，コア数を増加させるとその数に比例して消費電力が増加するのに対し，動作周波数を上昇させるとその値だけでなく供給電圧の 2 乗にも比例して消費電力が増加するためである．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，コア数増加に伴い性能向上が頭打ちになるプログラム（**図 1(b)** の `x264`）や性能が悪化するプログラム（**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．例えば，`x264` の場合，24 コアを用いた 2.3 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，最大動作周波数である 2.6 GHz で 18 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

`freqmine` の実行における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 つのループを，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（18，24，32，48，64）とその時の最大動作周波数の組み合わ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関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．

1，3，5 番目のループでは，2.6 GHz の 18 コアによる実行が 5 種類の組み合わせの中で最大の IPS を達成している．これに対し，2 番目と 4 番目のループでは，最高性能を達成する組み合わせがそれぞれ 2.0 GHz の 32 コアによる実行と 1.7 GHz の 48 コアによる実行となる．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．

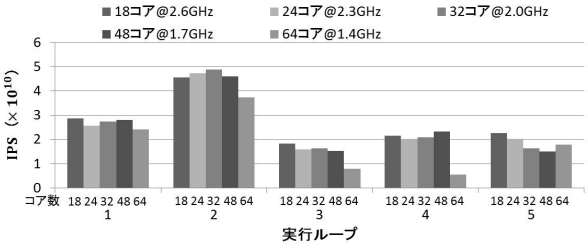


図 2 一定区間ごとの性能特性 (`freqmine`)

Fig. 2 Performance characteristics of `freqmine` for different phases

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言える．そこで本論文では，プログラムの特性（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み合わせを選択する手法を提案する．次の章にて，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，利用可能な全てのコアを使用するために，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

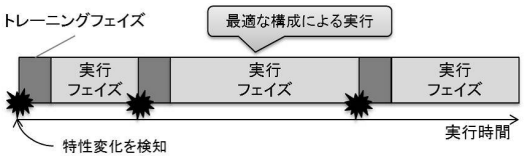


図 3 提案手法の概要
Fig. 3 Overview of proposed technique

行フェイズ” と呼ばれる 2 種類のフェイズから成り立つ．**図 3** に提案手法の概要を示す．トレーニングフェイズでは，ある短い時間毎に構成（コア数と動作周波数の組み合わせ）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．そして，記録した IPS の値から性能を最大化する構成が推測される．実行フェイズでは，推測した構成によりプログラムを実行しつつ，一定時間ごとに IPS を計測することでプログラムの特性変化を検知する．このトレーニングフェイズと実行フェイズをプログラムの実行終了まで繰り返すことにより実行中に変化するプログラムの特性に追従することが可能となる．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，この手法は全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一切必要ないことに注意されたい．

3.2.1 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めに，構成を動的に変更しつつ IPS を測定する．なお，あるコア数での最高性能はその際に利用可能な最大動作周波数での実行により得られるため，それぞれのコア数において最大動作周波数で実行する場合のみ IPS を計測する．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．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．**図 4** にトレーニングフェイズの概要を示す．この図は両アルゴリズムにおいてどのように最適な構成を探索するかを示しており，右下の点線部分は消費電力制約により利用できない構成を表す．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，図 4 左図の ① に示すように全コア数とその時の最大動作周波数（64 コアと 1.4GHz の組み合わせ）によりプログラムを実行し，ある一定時間（“トレーニング期間” と呼称）IPS を計測・記録する．次に，動作周波数を上昇できるポイント ② までコア数を 1 ずつ減少させつつ，トレーニング期間中に IPS を計測・記録する．そして，動作周波数を上昇させ（図中 ③），コア数が 1 になるまで図中の矢印が示すように IPS の計測を同様に繰り返す．最後に，計測した全通りの IPS を比較し，最も高い IPS を達成した構成を最適な構成として選択する．
- **ヒルクライム法**
  まず全探索法と同様に，全コア数とその時の最大動作周波数（64 コアと 1.4GHz）によりプログラムを実行し，トレーニング期間中に IPS を計測・記録する．次に，動作周波数は一定のまま IPS が低下するポイント（図 4 右図の ①）まで，コア数を 1 ずつ減少させつつ各 IPS を得る．そして，現在の動作周波数（1.4GHz）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みなす．続いて動作周波数を上昇させ（1.7GHz），その時に最大限利用できるコア数から IPS が低下するまでコア数を減少させつつ，IPS を得る．ただし，動作周波数を上昇できるポイントに到達するまで IPS が低下しない場合（図中 ②），その地点で動作周波数を上昇させる．これらの探索を利用可能なすべての動作周波数（1.4, 1.7, 2.0, 2.3, 2.6GHz）に関して繰り返す．最後に，各動作周波数において最高性能を達成した構成の IPS を比較し，最適な構成を決定する．

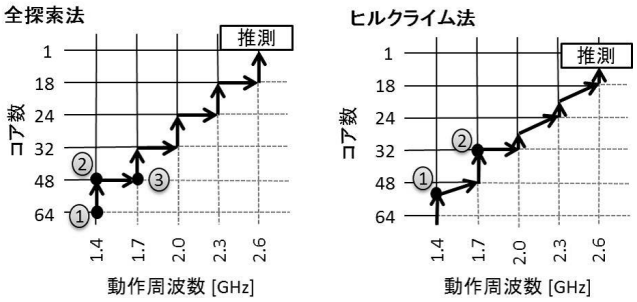


図 4 トレーニングフェイズの概要
Fig. 4 Overview of the training phase

トレーニングフェイズでは様々な構成でプログラムを実行するため，探索時には最適な構成での実行に比べ性能が低下する可能性がある．そのため，1 度のトレーニングフェイズに要する時間を可能な限り短縮する必要がある．この時間は探索する構成の数とトレーニング期間の積によって算出され，探索する構成数を減少させることで短縮できる[*3]．

全探索法では全コアから 1 コアまでの構成において IPS を計測するため，今回の実験環境（第 2.1 節参照）では，一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 64 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 5 構成

[*3] トレーニング期間は変更する前後の構成によって異なる．これについては，第 3.3 節にて詳しく説明する．

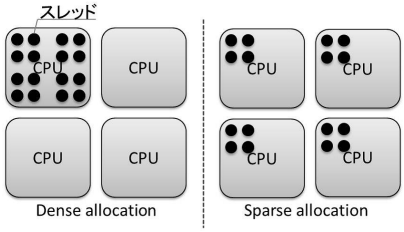


図 5 スレッドパッキング手法の概要
Fig. 5 The overview of two “thread packing” allocation

を探索するため，一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 25 程度である．そのため，ヒルクライム法のほうが全探索法に比べ 1 度のトレーニングフェイズに要する時間を短縮できる．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ため，今後ますますヒルクライム法の有用性が高くなる．

3.2.2 実行フェイズ

実行フェイズでは，トレーニングフェイズにおいて推測した最適な構成によりプログラムを実行する．また，プログラムの特性が図 2 で示す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 がトレーニングフェイズで計測した最大の IPS と比較してあるしきい値以上変化すれば，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．

3.3 実装

本論文では実装を簡単化するために，提案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．本論文では，全実行において全コア数と等しい 64 スレッドを生成し，それらを指定した数のコアに割り当てるスレッドパッキング手法を採用しているが，この手法の実装方法として**図 5** に示す 2 通りが考えられる．図 5 は 16 コアを使用する場合の例を示しており，できるだけ近いコアにスレッドを割り当てる手法を “Dense allocation”，各 CPU にスレッドを均等に割り当てる手法を “Sparse allocation” と呼称する．本論文では，これら 2 通りの手法を実装した．また，動作周波数の変更に関しては，/sys/devices/system/cpu/cpu[CPUID]/cpufreq/scaling_setspeed への動作周波数の値の書き込みで実装した．

3.3.1 トレーニングフェイズ

提案手法では，トレーニング期間が性能を決定する上で重要なパラメタとなる．様々な構成での実行による性能低下を最小限に抑えるために，トレーニング期間を可能な限り短くしなければならない．しかしながら，コア数変更後にスレッドのマイグレーションが完了するまで数百ミリ秒の時間を要するため，トレーニング期間が短すぎる場合には構成変更後に IPS の値が変動し正確な IPS を測定できない可能性がある[*4]．そのため，IPS の値が一定の値に収束するまで IPS の計測を待機する必要がある．

表 3 コア数増加後の待機時間
Table 3 Time to complete thread migration when increasing the core counts

Dense allocation

待機時間 [ms]		増加後のコア数 19-24	25-32	33-48	49-64
変更後のコア数	1-18	150	260	360	600
	19-24	150	360	380	580
	25-32		270	370	590
	33-48			400	480

Sparse allocation

待機時間 [ms]		増加後のコア数 19-24	25-32	33-48	49-64
変更後のコア数	1-18	130	200	200	270
	19-24	140	120	240	270
	25-32		210	340	370
	33-48			300	270

構成を変化した後に IPS が収束する時間を計測した結果，コア数を減少させる場合には常に 1 ms 以内であった．しかしながら，コア数を増加させる場合，実行するプログラムと増加させる前後のコア数によってこれらの時間は異なる値となった．特性の異なる 4 つのベンチマーク（`blackscholes`, `canneal`, `dedup`, `streamcluster`）において，コア数を増加させる各場合に IPS が収束する最長の時間を Dense allocation と Sparse allocation の各手法で計測した結果を**表 3** に示す．提案手法では，増加させる前のコア数と後のコア数に応じて表 3 に示した時間 IPS の計測を待機する．

また，様々な構成での実行による性能低下を最小限にするために，構成を変更し IPS の値が収束した後に IPS を計測する時間も可能な限り短く設定する必要がある．本論文の実装では，IPS 計測のために nanosleep(2) システムコールを用いて一定時間待機する．そこで，本研究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，1 ms となった．すなわち，ある構成におけるトレーニング期間は，IPS が収束するまでの待機時間（コア数を増加させる場合には表 3 に示した値であり，コア数を減少させる場合には 1 ms）と

[*4] 動作周波数と供給電圧の変更は数十マイクロ秒で完了するため，IPS への影響は無視できる [10].

IPS を計測するための 1 ms の和によって算出できる．

3.3.2 実行フェイズ

実行フェイズでは 1 秒ごとに IPS を計測し，現在の IPS がトレーニングフェイズで計測した最大の IPS と比較してあるしきい値以上に増減した場合に性能特性が変化したとみなし，トレーニングフェイズへと移行する．提案手法を適用する際，このしきい値が性能を決定する重要なパラメタとなる．本研究では，全探索法・ヒルクライム法の 2 種類の構成探索アルゴリズムと Dense allocation・Sparse allocation の 2 種類のスレッドパッキング手法の組み合わせにより計 4 種類の提案手法を実装できる（それぞれを DCFS-EXH-DENSE, DCFS-HILL-DENSE, DCFS-EXH-SPARSE, DCFS-HILL-SPARSE と表記する）．そこで，全プログラムの平均実行時間を最短にできるしきい値を各手法において計測したところ，それぞれトレーニングフェイズで計測した最大の IPS の 90, 80, 80, 40 %となった．よって，適用する手法に応じてこれらの値をしきい値として設定する．

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第 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第 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 [1] から `raytrace` を除いた 12 個を用いる．本研究の実験環境では，`raytrace` を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 1.4GHz の 64 コアと 1 コアで実行し，1 コア実行に対する 64 コア実行時の性能比を算出した．その結果を**表 4** に示し，後の考察で用いる．

表 4 各ベンチマークの分類
Table 4 Classification of the evaluated benchmarks

並列性	ベンチマーク	64 コア実行時の 1 コア実行時に対する性能比
高	blackscholes	54.9x
	swaptions	51.7x
	vips	44.9x
中	ferret	29.0x
	fluidanimate	24.0x
	freqmine	20.1x
	x264	16.0x
	canneal	12.7x
低	bodytrack	8.20x
	streamcluster	4.30x
	facesim	3.13x
	dedup	1.16x

表 5 トレーニングフェイズに要した合計時間の実行時間に対する割合
Table 5 Fractions of total trainging time to execution time

ベンチマーク	トレーニングフェイズに要した合計時間の実行時間に対する割合（%）			
	Dense allocation		Sparse allocation	
	EXH	HILL	EXH	HILL
blackscholes	15.6	2.5	16.0	4.0
swaptions	8.3	1.0	7.4	1.7
vips	16.8	2.0	16.2	1.9
bodytrack	4.0	0.4	12.5	11.8
dedup	2.1	0.3	1.7	0.3
x264	7.3	0.9	17.6	10.6
fluidanimate	1.3	0.3	34.2	12.8
streamcluster	0.2	0.6	0.9	1.2
canneal	4.6	0.5	4.2	0.6
ferret	5.5	0.6	20.0	2.4
freqmine	6.3	2.6	4.6	4.2
facesim	0.3	0.1	2.8	14.1

4.2 評価結果

本節では，以下に示す計 12 種類の実行と従来の全コア実行（1.4GHz の 64 コアによる実行）を比較し提案手法の評価を行う．

- 静的に決定した構成による実行（プログラム実行中の構成変更は行わない）
  Dense allocation と Sparse allocation それぞれにおいて以下の 4 種類の実行
  - 2.6GHz の 18 コアによる実行（18 コア@2.6GHz）
  - 2.3GHz の 24 コアによる実行（24 コア@2.3GHz）
  - 2.0GHz の 32 コアによる実行（32 コア@2.0GHz）
  - 1.7GHz の 48 コアによる実行（48 コア@1.7GHz）
- 提案手法
  第 3.3.2 節で述べた 4 種類の手法

これらの各手法によってベンチマークを実行し性能を測定した結果を**図 6** に示す．横軸はベンチマーク，縦軸は従来の全コア実行時の性能（実行時間の逆数）で正規化した性能を表している．各ベンチマークにおいて，一番左から 4 本のバーとその右の 4 本のバーはそれぞれ Dense allocation と Sparse allocation を適用し構成を静的に決定した場合の実行であり，さらにその右側の 5 本のバーはそれぞれ従来の全コア実行と提案手法 4 種類の実行である．また，各提案手法を適用した際の実行においてトレーニングフェイズに要した合計時間の実行時間に対する割合を**表 5** に示す．

高い並列性を持つプログラムである `blackscholes`, `swaptions`, `vips` については，従来の全コアを用いた実行によって最大性能を達成することができる．そのため，提案手法のトレーニングフェイズにおける様々な構成での実

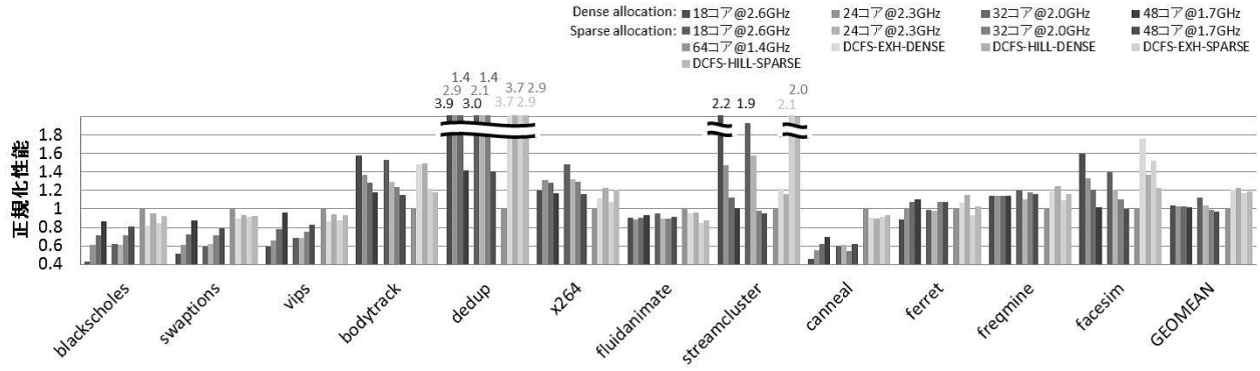


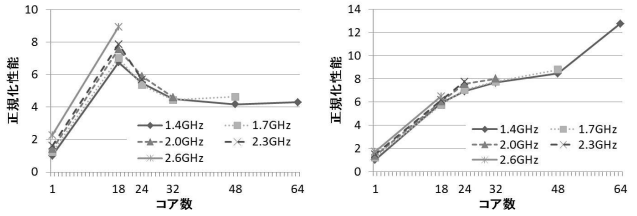
図 6 提案手法の評価結果
Fig. 6 Performance normalized to the minimum frequency execution with all cores

行が性能低下を招く．全探索法では，これら 3 つのベンチマークにおいて実行時間の 7.4%から 16.8%の時間をトレーニングフェイズに費やしており，全コア実行に対し最悪の場合で約 17%性能が低下した．これに対し，ヒルクライム法ではトレーニングフェイズに要する時間の割合を大幅に削減し，最悪の場合でも全コア実行に対し約 92%の性能を達成している．

中・低程度の並列性を持つ `bodytrack`，`dedup`，`x264` に関しては，提案手法を適用することで全コア実行に対し性能を向上できた．特に `dedup` においては，Dense allocation を採用した提案手法により 3.7 倍の性能向上を達成した．しかしながら，これら 3 つのベンチマークでは，提案手法による性能向上が構成を静的に決定した場合の実行の内最大性能を達成しているものより小さい．なぜなら，これらのベンチマークの性能特性は実行中にほとんど変化せず，構成を動的に変更する必要がないためである．このような場合，高並列なプログラムの場合と同様トレーニングフェイズでの様々な構成での実行により最適な構成での実行に比べ性能が低下する．そのため，最適な構成を静的に決定し実行中に構成を変化させない実行により最大性能を達成できる．しかしながら，プログラムの実行前に最適な構成を決定するためには，プログラムの事前解析やモデルによる予測が必要となる．これに対し本論文の提案手法では，それらを一切必要とせず，プログラムの実行開始直後に最適な構成を予測し選択することが可能である．また，`bodytrack` と `dedup` において Dense allocation を採用した提案手法と Sparse allocation を採用した提案手法の性能を比較すると，Dense allocation を採用した手法の性能が大幅に高いことが分かる．本実験で用いたプラットフォームは第 2.1 節で述べたように 4 つのプロセッサから構成されており，それぞれのプロセッサに共有 L3 キャッシュが搭載されている．`bodytrack` と `dedup` では，スレッド間で共有するデータが多数使用されており，可能な限り多くのスレッドが同一キャッシュを用いることで共有データへのアクセス時間を短縮できると予想される．そのため，できるだけ近くのコアを使用する Dense allocation のほうがより高い性能を達成できた．

`fluidanimate` の並列性は中程度であるにも関わらず，全コアによる実行が最大性能を達成している．なぜなら，`fluidanimate` の実行では，図 1(b) に示した `x264` とは異なり，コア数増加に伴い性能が向上するためである．よって，高並列なプログラムと同様にトレーニングフェイズにおける様々な構成での実行により性能が低下する．しかしながら，Dense allocation とヒルクライム法を採用した提案手法では，トレーニングフェイズに要する合計時間の割合がわずか 0.3%であり，全コア実行に対し約 96%の性能を達成した．

`streamcluster` と `canneal` では，前に述べた `bodytrack` と `dedup` の結果とは異なり Sparse allocation を採用した提案手法が Dense allocation を採用したものと比べより高い性能を達成している．特に `streamcluster` においては，Dense allocation を採用した手法の 2 倍近くの性能向上が得られた．Bienia らの研究によると，これら 2 つのベンチマークは評価に用いたベンチマークの中で，最もメモリバウンドなプログラムである [1]．つまり，このようなプログラムを実行する場合，より大きな容量のキャッシュを利用できる手法が有効であると言える．Dense allocation では最大限多くのスレッドが同一の共有 L3 キャッシュを使用するの対し，Sparse allocation では 4 つのプロセッサに搭載された共有 L3 キャッシュをすべて使用できる．そのため，Sparse allocation を採用することで L3 キャッシュミス率を削減でき，高い性能を達成したことが予想できる．ただし，提案手法の 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ため，メモリバウンドなプログラムの実行においては動作周波数上昇による性能向上があまり期待できない．なぜなら，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である．**図 7** は，図 1 と同様に `streamcluster`



(a) streamcluster (b) canneal

図 7 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. 7 Parallelism of Memory-bound applications with different CPU frequency

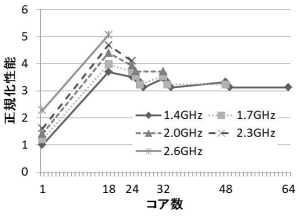


図 8 `facesim` におけるコア数と動作周波数に応じた性能特性

Fig. 8 Parallelism of `facesim` with different CPU frequency

と `canneal` 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．`streamcluster` においてはコア数を制限することで全コア実行に対し性能向上を達成できるが，`canneal` においてはコア数制限による性能向上も得られない．しかしながら，提案手法によって実行開始直後に 64 コアを選択することで従来の全コア実行の約 94%の性能を達成している．

中・低程度の並列性を持つ `ferret`，`freqmine`，`facesim` の 3 つのベンチマークでは，提案手法によって最大性能を達成できた．この結果から，これらのプログラムの性能特性が実行中に変化し，提案手法を適用することでその特性変化に応じて最適な構成を動的に選択できたと言える．しかしながら，`facesim` に関しては第 3.2.1 節で述べた予想に反して，全探索法を採用した提案手法がヒルクライム法を採用した手法に比べより高い性能を達成している．**図 8** に図 1 と同様 `facesim` の正規化性能を示す．第 3.2.1 節のヒルクライム法の説明では，性能を測定するための指標である IPS をコア数の関数とした場合に，その関数が単峰性関数となることを仮定した．しかしながら，図 8 から分かるように `facesim` の場合には単峰性関数となっていない．そのため，ヒルクライム法では最適な構成を探索できず，高い性能を達成できなかった．

構成を静的に決定する場合の実行では，Sparse allocation を採用した 2.6GHz の 18 コアによる実行が評価に用いた全 12 個のベンチマークにおいて従来の全コア実行に対し最大で 3.9 倍，平均で 13%の性能向上を達成した．これに対し，Dense allocation とヒルクライム法を採用した提案手法では全コア実行に対し最大で 3.7 倍，平均で 22%の性能向上が得られた．よって，その提案手法と構成を静的に決定する実行の性能を比較すると，提案手法による性能向上が平均で 9%となる．

5. 関連研究

プログラムの特性に応じて並列度（スレッド数やコア数）を制御し，マルチコアにおいてエネルギー効率の良い並列実行を実現する研究が多くなされている．Suleman らによる Feedback-driven threading と呼ばれる手法は，ループイタレーションにおいてスレッド数を動的に制御する [15]．また，Curtis らはモデルと実行中のプログラム分析から得た情報を基に最もエネルギー効率の良い実行を実現するコア数を選択する手法を提案している [4]．本論文で提案する手法もプログラムの特性に応じて効率の良い実行を実現するための手法であるが，本手法では並列度（コア数）だけでなくコアの動作周波数も同時に制御することでエネルギー効率をより向上できる．

マルチコアの性能を向上するためにコアの動作周波数を動的に制御する手法として，Intel の Turbo Boost テクノロジー（TB）が挙げられる [9]．TB はプロセッサの消費電力が Thermal design power を下回る場合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する手法である．プロセッサの消費電力は稼働するコアの数に大きく依存するため，コア数が少ない場合には動作周波数を大幅に上昇できるが，コア数が多い場合にはその上昇幅が小さくなる．メニーコアにおける並列プログラムの実行では，チップ上の全コアを利用し最大性能を達成しようと全コア数もしくはそれ以上の数のスレッドを生成する傾向にある．しかしながら，様々な特性を持つ並列プログラムが存在するため，全コアを用いた実行が必ずしも最大性能を達成するとは限らない．提案手法の DCFS では，プログラムの並列性が低い場合にコア数をあえて制限し，休止させたコアの分の消費電力を動作周波数上昇に利用できる．

また，本研究での提案手法と同様に，並列プログラムの特性に応じてコア数と動作周波数の両方を動的に制御する手法がいくつか提案されている．Cochran らによる Pack & cap では，プログラムの実行中に時々刻々と変化する消費電力制約下において，コア数と動作周波数を制御し性能の最大化を狙う [3]．しかしながら，この手法ではピーク消費電力の関数として最適なコア数と動作周波数を推測するために静的な回帰分析を行う．これに対し，DCFS はプログラムの静的解析を一切必要としない手法である．また，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

法は有用であると言える．一方，Lee らは GPU においてコアの供給電圧と動作周波数，コア数を動的に制御することの有用性を評価している．彼らは，コア数に加えコアとインターコネクトの供給電圧・動作周波数を動的に制御することでスループットの向上を達成できることを示している [11]．ただし，評価は GPU シミュレータを用いて行われており，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具体的なアルゴリズムは提案されていない．

6. おわりに

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一切必要としない．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，従来の全コア実行と比較すると最大で 3.7 倍，平均で 22%の性能向上を達成した．また，コア数と動作周波数の最適な組み合わせを静的に選択する実行に比べ平均で 9 %の性能向上を達成した．今後の課題として，まず，スレッドパッキング手法の Dense allocation と Sparse allocation それぞれを適用した際にいくつかのベンチマークにおいて性能が異なる原因を詳しく解析する．また，トレーニングフェイズにおいて最適な構成を探索するアルゴリズムを改良し，図 8 に示した `facesim` のようなプログラムにも対応できるようにする．さらに，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い，性能だけでなく消費電力及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., Singh, J. P. and Li, K.: The PARSEC benchmark suite: characterization and architectural implications, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT ’08, ACM, pp. 72–81 (2008).

[2] Choi, K., Soma, R. and Pedram, M.: Dynamic voltage and frequency scaling based on workload decomposition, *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ISLPED ’04, ACM, pp. 174–179 (2004).

[3] Cochran, R., Hankendi, C., Coskun, A. K. and Reda, S.: Pack & Cap: adaptive DVFS and thread packing under power caps, *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 ’11, ACM, pp. 175–185 (2011).

[4] Curtis-Maury, M., Blagojevic, F., Antonopoulos, C. D. and Nikolopoulos, D. S.: Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes, *IEEE Transactions on Parallel and Distributed System*, Vol. 19, No. 10, pp. 1396–1410 (2008).

[5] Fan, X., Weber, W.-D. and Barroso, L. A.: Power provisioning for a warehouse-sized computer, *Proceedings of the 34th annual International Symposium on Computer architecture*, ISCA ’07, ACM, pp. 13–23 (2007).

[6] Herbert, S. and Marculescu, D.: Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED ’07, ACM, pp. 38–43 (2007).

[7] 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, *2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, ISSCC ’10, IEEE, pp. 108–109 (2010).

[8] Hsu, C. and Feng, W.: A Power-Aware Run-Time System for High-Performance Computing, *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC ’05, Washington, DC, USA, IEEE Computer Society, p. 1 (2005).

[9] Intel: Intel Turbo Boost Technology in IntelCore Microarchitecture (Nehalem) Based Processors, *Whitepaper, Intel Corporation* (2008).

[10] Kim, W., Gupta, M., Wei, G. and Brooks, D.: System level analysis of fast, per-core DVFS using on-chip switching regulators, *Proceedings of the 14th annual International Symposium on High Performance Computer Architecture*, HPCA ’08, Ieee, pp. 123–134 (2008).

[11] Lee, J., Sathisha, V., Schulte, M., Compton, K. and Kim, N. S.: Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling, *Proceedings of the 20th annual International Conference on Parallel Architectures and Compilation Techniques*, PACT ’11, IEEE Computer Society, pp. 111–120 (2011).

[12] S, B., B, E. and J, A.: Tile64-processor: A 64-core SoC with Mesh Interconnect, *Solid-State Circuits*, pp. 88–598 (2008).

[13] 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, *ACM SIGGRAPH 2008 papers*, SIGGRAPH ’08 (2008).

[14] Semeraro, G., Magklis, G., Balasubramonian, R., Albonesi, D. H., Dwarkadas, S. and Scott, M. L.: Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling, *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA ’02, IEEE Computer Society, pp. 29–40 (2002).

[15] Suleman, M. A., Qureshi, M. K. and Patt, Y. N.: Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs, *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, ACM, pp. 277–286 (2008).