

稼働コア数制限に基づく マルチコア・プロセッサ性能向上手法の提案

今 村 智 史^{†1} 福 本 尚 人^{†1}
井 上 弘 士^{†2} 村 上 和 彰^{†2}

本稿では、実行プログラムに応じた稼働コア数制限と動作周波数上昇によりマルチコア・プロセッサにおける並列プログラム実行を高速化する手法を提案し、評価を行う。稼働コア数増加に伴い性能向上が得られない場合、稼働コア数をあえて制限することで消費電力を削減する。そして、本来は動作を停止したコアに割り当てられていた消費電力バジェットを稼働コアに再割当てし、稼働コアの動作周波数を上昇させる。つまり、実行プログラムの特性に応じて、活用すべき並列性とコアの動作周波数の間に存在する適切なトレードオフポイントを選択することで性能向上を達成する。適切な稼働コア数に関しては、実行時間モデルと事前実行により取得したプロファイル情報を用いて決定する。ベンチマーク・プログラムを用いて定量的な評価を行った結果、従来の全コア実行に対して最大で約 73%性能が向上し、最大約 44%消費エネルギーを削減した。

High-Performance Multicore Execution with Active Core Throttling

SATOSHI IMAMURA,^{†1} NAOTO FUKUMOTO,^{†1} KOUJI INOUE^{†2}
and KAZUAKI MURAKAMI^{†2}

This paper proposes and evaluates a technique to enhance the performance of multi-core processors with active core throttling. If the performance of parallel execution doesn't increase as the number of cores increases, the power of processors is reduced by throttling active cores. At the same time, active core's operating frequencies are raised by reallocating power budgets which were allocated to idle cores to active core. That is, proposal technique achieves high-performance by choosing the optimal trade-off point between parallelism and active core's operating frequencies. The optimal number of active cores is determined with an execution time model and a profile which is acquired by a pre-execution. Our evaluation shows that the proposed technique achieved 73% of speedup and 44% reduction of energy consumption compared to a conventional parallel execution with all cores.

1. はじめに

複数のプロセッサコア（以下、コアと呼称）を 1 つのチップに搭載したマルチコア・プロセッサ（以下、マルチコアと呼称）が主流になっている。複数のコアにより並列処理を行うことで高性能かつ低消費電力を実現できる。微細化技術の進歩に伴い、チップに搭載されるコア数は増加傾向にある。たとえば、Intel が 2008 年にデスクトップパソコン向けプロセッサとして発売した core i7¹⁾ には 4 つのコア、2009 年に Sun が発売した Rainbow falls²⁾ には 16 個のコアが搭載されている。マルチコアでは、並列プログラムを複数スレッドに分割し各コアに割り当てることで並列処理が可能である。以下、スレッドを割り当てられたコアを稼働コアと呼ぶ。マルチコアにおいて並列プログラムを実行する際、理想的には稼働コア数に比例した性能向上を期待できる。

しかしながら、稼働コア数増加に伴う性能向上が得られない場合がある。これは、逐次処理部分の存在、同期処理における排他制御、負荷の不均一などの要因により、並列化効率が低下するためである。たとえば、逐次処理部分実行時や排他制御が行われる区間では、1 コアのみでしかプログラムを実行できない。また、コア毎の処理時間が異なる場合、多くのコアにおいてバリア同期が完了するまでの待ち時間が長くなる。これらが性能へ与える影響は稼働コア数が増加するにつれて大きくなる傾向にある。そのため、プログラムによっては、稼働コア数を増加させても、それに見合っただけの性能向上を実現できない、もしくは、むしろ性能が低下するといった問題が生じる。

そこで本稿では、消費電力制約下において各コアの電源電圧と動作周波数を最適化することで、並列化効率が低いプログラムにおいても実行を高速化する手法を提案する。従来方式では、プロセッサに搭載された全コアを用いて並列処理を行う。それに対して提案手法では、稼働コア数の増加による性能向上が得られない場合にはあえて稼働コア数を制限する。そして、本来は動作を停止したコアに割り当てられていた消費電力バジェットを稼働コアに再割当てし、稼働コアの動作周波数を上昇させる。すなわち、プロセッサに搭載された全コア稼働時の消費電力を上限値とし、消費電力がその上限値を超えないよう稼働コア数に応じて動作周波数を設定する。これにより、活用すべきスレッドレベル並列度と動作周波

^{†1} 九州大学大学院システム情報科学府

Department of Information Science and Electrical Engineering, Kyushu University

^{†2} 九州大学大学院システム情報科学府

Faculty of Information Science and Electrical Engineering, Kyushu University

数の間に存在する適切なトレードオフ・ポイントを選択可能とし、高い並列処理性能を実現する。なお、このような消費バジェットの借り貸しによる動作周波数の変更は、Intel 社の TurboBoost 技術を用いて実現できる³⁾。

本稿の構成は以下の通りである。第 2 節では、マルチコアにおける並列処理の問題点を述べる。第 3 節では、提案手法の概要を説明し、提案手法と従来の全コア実行方式の性能をモデルを用いて比較する。また、提案手法の実行時間モデルを用いた稼働コア数決定法について述べる。第 4 節では、ベンチマーク・プログラムを用いたシミュレーションにより提案手法の定量的評価を行う。第 5 節では、関連研究を紹介する。最後に第 6 節にて、まとめと今後の課題について述べる。

2. マルチコアにおける並列処理の問題

マルチコアの性能向上を阻害する主な要因として、逐次処理部分の存在、同期処理、負荷の不均一が挙げられる。これらの要因が性能に対して与える影響をそれぞれ説明する。

2.1 逐次処理部分の存在

並列プログラムには、逐次処理部分と並列処理可能部分が存在する。したがって、1 コアに対する並列処理の性能向上比は以下に示す式で表わされる⁴⁾。

$$speedup = \frac{1}{P_{seq} + \frac{1-P_{seq}}{N}} \quad (1)$$

ここで P_{seq} ($0 \leq P_{seq} \leq 1$) はプログラム実行における逐次処理部分の割合、 $1 - P_{seq}$ は並列処理可能部分の割合である。また、 N は稼働コア数であり、理想的な並列処理（つまり、並列処理可能部分の実行時間が $1/N$ ）を仮定している。この式を基に、逐次処理部分の割合に応じた性能向上を図 1 に示す。横軸は稼働コア数、縦軸は 1 コア実行時の性能に対する性能比である。各線は上から、逐次処理部分を全く含まない場合、1%含む場合、5%含む場合、10%含む場合を表わす。

プログラムにおける逐次処理部分の割合が 1%である場合、稼働コア数を 32 まで増加させると約 24 倍の性能向上が得られる。これに対し、10%である場合には稼働コア数増加に伴い性能向上が頭打ちになり、稼働コア数が 32 の場合でも性能向上は約 8 倍に留まる。このように、逐次処理部分の割合が大きい並列プログラムを実行する場合、稼働コア数増加に伴う性能向上が逐次処理部分により制限される。

2.2 同期処理

並列プログラムを複数コアで並列実行する際、コア間で同期をとらなければならない場面

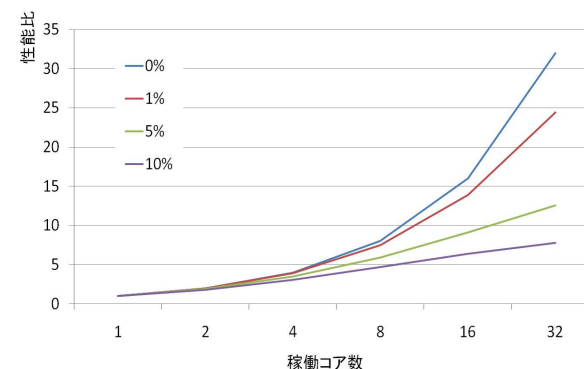


図 1 アムダールの法則

がある。たとえば、クリティカルセクションにおける排他制御や、複数コアによるバリア同期が挙げられる。

複数コアが同一のデータに対して同時に書き込みを行うと、プログラムの整合性が保てない。そのため、共有データに書き込みを行う際にはある 1 つのコアのみが処理できるよう排他制御を行う。この排他制御を行う区間をクリティカルセクションと呼び、この区間内の処理は 1 コアでしか実行できない。そのため、並列処理性能向上が阻害される。

また、あるコアが書込んだデータを他のコアが読出して使用する場合がある。このような場合、書き込みと読み込みの間でバリア同期を行う。バリア同期とは、全コアがバリアの地点に到達するまで各コアを待機させ、全コアが到達したら処理を同時に開始するという同期処理である。バリア同期では、自分がバリアに到達した最後のコアであるか否かを各コアが共有変数を用いて排他的に確認する⁵⁾。そのため、稼働コア数が増加するにつれてバリア同期の処理時間が長くなり、並列処理性能に対する影響も大きくなる。

2.3 負荷の不均一

複数コアにより並列処理を行う際にコア毎の処理時間が異なる場合、バリア同期を行う度に、処理時間が最も長いスレッドのバリアへの到達を他の全スレッドが待たなければならない。そのため、コア毎の処理時間が不均一である場合、効率的な並列処理を行えない。

2.4 性能向上阻害要因による性能への影響

マルチコアにおいてベンチマーク・プログラムを実行した際の性能を図 2 に示す。横軸は稼働コア数、縦軸は 1 コア実行時の性能に対する性能比である。実験環境の詳細は第 4.1

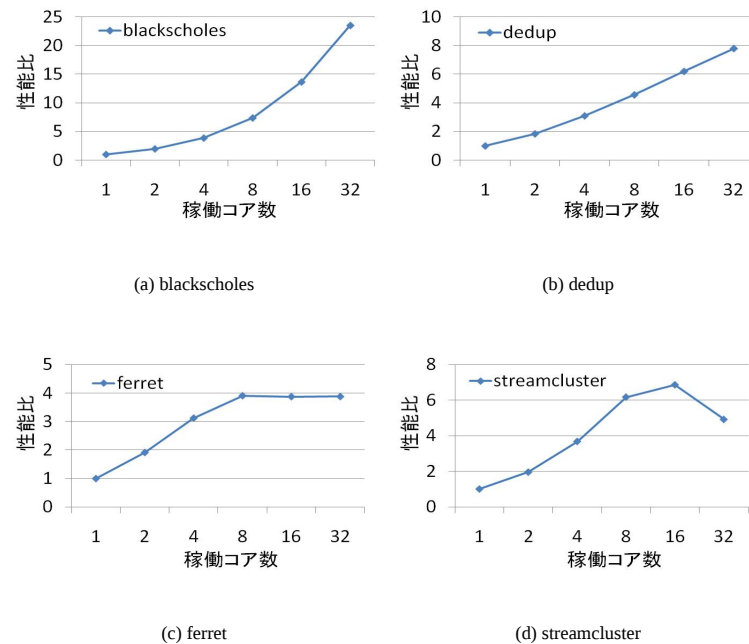


図 2 性能に対する性能向上阻害要因の影響

節で説明する。

blackscholes では、稼働コア数増加に伴い性能向上が得られており、性能向上阻害要因の影響がほとんどないと考えられる。これに対し、dedup では稼働コア数が 32 の場合でも性能向上が約 8 倍に留まり、ferret では約 4 倍で頭打ちになっている。また、streamcluster では稼働コア数を 32 まで増加させると性能が悪化している。そのため、これら 3 つのプログラムの実行には性能向上阻害要因が影響しており、従来の単純な並列実行による高性能化は難しいことが分かる。

3. 稼働コア数制限に基づくプロセッサの高性能化手法

3.1 基本 概念

本稿では、消費電力制約下において各コアの電源電圧と動作周波数を最適化することで、並列化効率が低いプログラムにおいても実行を高速化する手法を提案する。従来手法では、全コアを用いた並列処理を行う。しかしながら、プログラムによっては稼働コア数増加に伴う性能向上が得られないものがある。そのような場合、稼働コア数を増加させても消費電力の増加や性能低下を招く。そこで、稼働コア数をあえて制限することでプロセッサ性能を維持しつつ消費電力を削減する。さらに、本来は動作を停止したコアに割り当てられていた消費電力バジェットを稼働コアに再割当てし、稼働コアの動作周波数を上昇させることにより性能向上を狙う。

提案手法では、稼働コアの動作周波数を稼働コア数に応じて変化させる。本研究では、プロセッサに搭載されている全コアが稼働する時の消費電力を上限値とし、消費電力がその上限値を超えないよう稼働コアの動作周波数を決定する。そのため、全コア実行時に比べ消費電力を維持したままプロセッサ性能を向上できる。したがって、提案手法により性能向上が得られる場合、消費エネルギー削減も可能である。この提案手法では、並列性（つまり稼働コア数）と稼働コアの動作周波数上昇の適切なトレードオフポイントを選択することが重要である。

本研究では、性能向上阻害要因のうちクリティカルセクションにおける排他制御に注目する。なぜなら、クリティカルセクション内の処理は 1 コアでしか実行できず、性能向上阻害要因の中でも性能に与える影響が大きいと判断したためである。

3.2 性能モデリング

本節では、まず従来の並列処理の実行時間をモデル化する。従来手法では、動作周波数は上昇せず、並列処理可能部分では全コアを稼働させて並列処理を行う。従来手法の実行時間モデル T_{all_core} を式 (2) で表す。

$$T_{all_core} = \frac{CC \times P_{seq}}{f} + \frac{CC \times \left(\frac{1 - P_{seq} - CS}{N} + CS \right)}{f} \quad (2)$$

各パラメタの定義は以下の通りである。

- CC : 1 コアでのプログラム実行に要する総クロックサイクル数
- P_{seq} : 全プログラム実行時間に対する逐次処理部分の実行時間の割合
- N : プロセッサに搭載されている全コア数

- CS : 全プログラム実行時間に対するクリティカルセクション実行時間の割合
- f : 定格動作周波数 [GHz]

式(2)の第1項は逐次処理部分の実行時間, 第2項は並列処理可能部分の実行時間を表す.

次に, 提案手法の実行時間をモデル化する. 提案手法では, クリティカルセクションにおける排他制御の影響が大きい場合, 稼働コア数を制限し, 稼働コアの動作周波数を上昇させる. また, 逐次処理部分においても稼働コアの動作周波数上昇を行う. 提案手法の実行時間モデル T_{prop} を式(3)で表す.

$$T_{prop} = \frac{CC \times P_{seq}}{f'_1} + \frac{CC \times \left(\frac{1-P_{seq}-CS}{n} + CS \right)}{f'_n} \quad (3)$$

各パラメータは式(2)とほぼ同様だが, 従来手法と提案手法の異なる点は並列処理可能部分を実行する際の稼働コア数と動作周波数である. 提案手法において, 稼働コア数を n , 稼働コア数に応じて上昇する動作周波数を f'_n と表わす.

以上より, 従来手法に対する提案手法の性能向上比は式(4)となる.

$$speedup = \frac{T_{all_core}}{T_{prop}} \quad (4)$$

ここで, 稼働コア数に応じて上昇する動作周波数 f'_n について説明する. 本研究では, プロセッサに搭載された全コア数が 32 であるとし, 全 32 コア稼働時の消費電力を上限値に設定する. 上限となる消費電力を以下の式(5)から算出する.

$$P_{dyn} = a \times 32C \times f \times V^2 \quad (5)$$

a はスイッチング確率, C はコア 1 個当たりの負荷容量, f は動作周波数, V は供給電圧を表わす. ただし, プロセッサ全体の負荷容量はコア数に比例すると仮定する. そして, 消費電力がこの上限値を超えないよう, 稼働コア数に応じて動作周波数 f'_n を決める. なお, 供給電圧 ± 38 [mV] に対してコアの動作周波数は ± 100 [MHz] 変更可能であるとする. 稼働コア数に応じた消費電力と動作周波数を図 3 に示す. 横軸は稼働コア数, パーの縦軸は全コア稼働時の消費電力に対する消費電力比, 折れ線の縦軸は動作周波数 [GHz] を表わす. なお, 各コアの定格動作周波数は 2 [GHz], 各コアへの定格供給電圧は 1.3 [V] である. この値は Intel 社の PentiumM を基にしたものである⁶⁾. また, IBM 社の Power6 を基に動作周波数の最大値は 5 [GHz] とした⁷⁾. ただし, 本研究ではコアへの供給電圧の上限を設定していない.

3.3 従来手法との比較

本節では, 式(4)を用いて, 従来手法に対する提案手法の有効性を評価する. 稼働コア

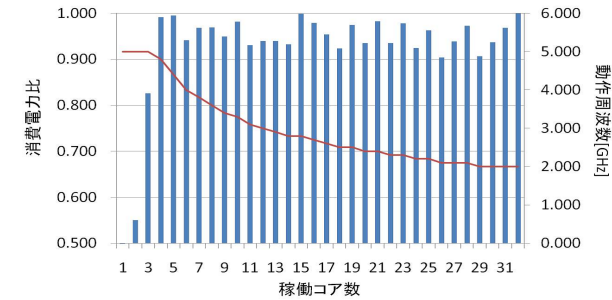


図 3 稼働コア数に応じた消費電力と動作周波数

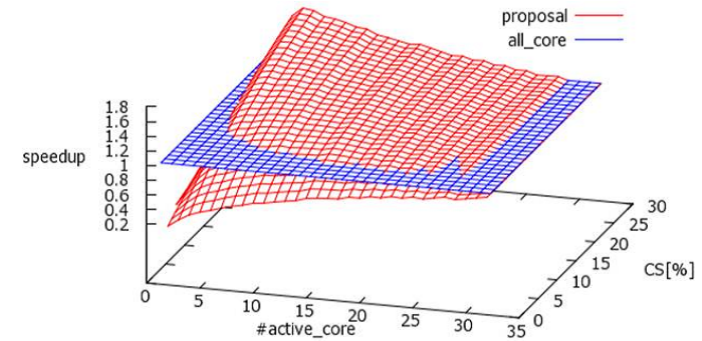


図 4 従来手法に対する提案手法の性能向上比

数 n とクリティカルセクションの割合 CS を変数とする性能向上比を図 4 に示す. x 軸は稼働コア数, y 軸はプログラムにおけるクリティカルセクションの割合 [%], z 軸は従来手法の性能で正規化した提案手法による性能向上比である. 青線の all_core が従来の全コア実行手法の性能, 赤線の $proposal$ が提案手法を示す. ここでは, 並列処理可能部分の占める割合が非常に大きい並列プログラムにおいて提案手法の有効性を示すために, 逐次処理部分の割合を 1% とする.

実行プログラムにおけるクリティカルセクション内の処理の割合が大きくなるほど, 性能向上を最大にする稼働コア数が少なくなり, 性能向上が大きくなる. たとえば, クリティカルセクションの割合が 10% であるとき, 性能向上を最大にするコア数は 10 であり, 性能向上は約 14% である. これに対し, クリティカルセクションの割合が 30% であるとき, 性能

向上を最大にするコア数は4であり、性能向上は約65%である。以上より、プログラムの実行時間に対するクリティカルセクションの実行時間の割合が大きいほど、提案手法により大きな性能向上が得られることが分かる。

3.4 実行時間モデルを用いた稼働コア数決定法

本節では、提案手法における稼働コア数決定法について説明する。提案手法では、前節で示した実行時間モデル式(3)を用いて稼働コア数をプログラム実行前に決定する。その手順を以下に示す。

- (1) 対象プログラムを事前実行し、そのプログラムのプロファイル情報を取得する。
- (2) (1) で得られたプロファイル情報から以下3つの情報を抽出する。
 - プログラムを1コアで実行した場合の所要総クロックサイクル数(CC)
 - 全プログラム実行時間に対する逐次処理部分の実行時間の割合(P_{seq})
 - 全プログラム実行時間に対するクリティカルセクション実行時間の割合(CS)
- (3) (2) から得られた3つの情報を式(3)に代入する。
- (4) 稼働コア数が1コアから32コアまでの全通りにおける実行時間を計算する。
- (5) 実行時間が最短となる場合のコア数をプログラム実行時の稼働コア数として決定する。

4. ベンチマークプログラムを用いた定量的評価

4.1 実験環境

本実験の目的は、従来の全コア実行に比べ提案手法により性能向上と消費エネルギー削減が達成できることを示すことである。以下の4つの評価対象モデルに関する性能を比較する。以下、性能向上を最大とする稼働コア数を最適コア数と呼ぶ。

- all_core:全コア(32コア)による従来実行。
- throttling:最適コア数による実行(動作周波数上昇なし)。最適コア数は既知と仮定。
- proposal_opt:提案手法。最適コア数を既知と仮定。
- proposal_model:提案手法。実行時間モデルを用いて稼働コア数を決定。

稼働コア数に応じて上昇する動作周波数は図3を基にプログラム実行前に決定し、プログラム全体を通して一定であると仮定する。提案手法による実行時間には、事前実行にかかる時間を含んでいない。なお、逐次処理部分の割合は1コア実行時の性能と2コア実行時の性能の比から算出し、クリティカルセクション実行に要する時間は事前実行時に計測した。シミュレータは、マルチコアシミュレータ M5⁸⁾を用いた。対象としたベンチマーク・プログラムは PARSEC⁹⁾のうち、blackscholes, dedup, swaptions, ferret, streamcluster の5つ

表1 シミュレータの設定	
コアの構成	32 コア, イン・オーダ
L1 命令キャッシュ	32KB, 2-way, 64B lines, 0.5 ns, MSHR 8
L1 データキャッシュ	32KB, 2-way, 64B lines, 0.5 ns, MSHR 32
L2 キャッシュ	16MB, 8-way, 64B lines, 4 ns, MSHR 92
L1-L2 間共有バス幅	64B
L2-主記憶間バス幅	16B
主記憶アクセス時間	150 ns
Miss Status Buffer	エントリ数: 20, レイテンシ: 1 clock cycle

表2 各プログラム実行における逐次処理部分・クリティカルセクションの割合

プログラム	逐次部分の割合 [%]	CS の割合 [%]
blackscholes	1.030	0.089
swaptions	0.090	0.212
dedup	7.565	3.824
ferret	4.151	13.746
streamcluster	2.975	0.029

である。表1にシミュレータの設定を示す。

4.2 結果

図5に、プログラムごとの各手法における性能向上比を示す。横軸はベンチマーク・プログラム、縦軸は全32コア実行時(従来手法)に対する性能向上比を表わす。各プログラムのそれぞれのバーは評価対象を表わし、左から all_core, throttling, proposal_opt, proposal_model である。また、バーの上の数字は稼働コア数を示す。さらに、提案手法における稼働コア数決定に必要なデータをプログラムごとに表2に示す。

図6に、全32コア実行時に対する消費エネルギー比を示す。横軸は各プログラム、縦軸は消費得エネルギー比を表わす。なお、図の見方は図5と同様である。

blackscholes と swaptions では、クリティカルセクションにおける排他制御の影響が小さいため、稼働コア数増加に伴い性能が向上する。そのため、全手法において全32コアで実行する場合の性能向上が最大となった。

dedup では、動作周波数を上昇させない場合、全コア実行時の性能が最大となった。提案手法では、性能を維持したまま最適コア数が28となり、消費エネルギーを約3%削減した。

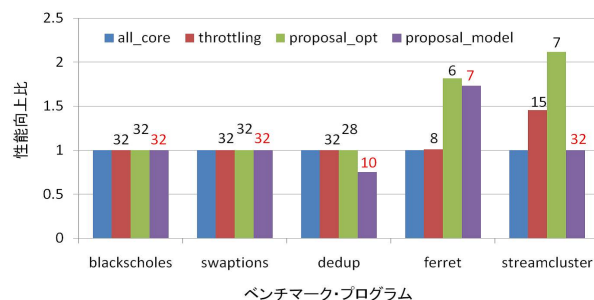


図 5 各プログラムにおける性能向上比

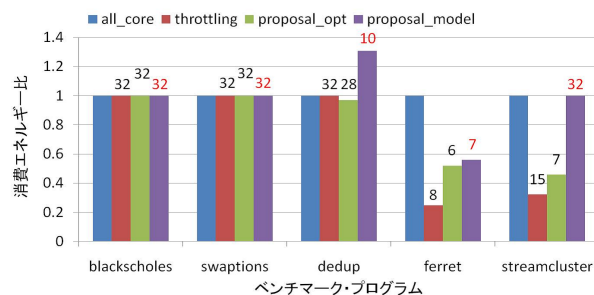


図 6 各プログラムにおける消費エネルギー比

しかしながら、モデルによる稼働コア数決定法では最適コア数と大きく異なる 10 コアを選択したため、従来手法に比べ性能が約 25%低下し、消費エネルギーが約 31%増加した。

ferret はクリティカルセクション内の処理を約 14%含むプログラムである。そのため、稼働コア数を 8 に制限することで性能を維持しつつ、消費エネルギーを約 75%削減した。提案手法では、最適コア数が 6 の場合に、性能が約 82%向上し、消費エネルギーも約 48%削減できた。モデルによる稼働コア数決定法では最適コア数と異なる 7 コアを選択したため、最適コア数で実行する場合ほどの性能向上は得られなかった。しかしながら、約 72%の性能向上と約 44%の消費エネルギー削減が達成できた。

streamcluster を実行する場合、稼働コア数を 15 に制限することで約 46%性能が向上し、消費エネルギーを約 68%削減できた。提案手法では、最適コア数 7 の場合に性能が 2 倍以上向上し、消費エネルギーも約 54%削減した。しかしながら、streamcluster のクリティカル

セクション内の処理の割合は極めて小さいため、モデルにより最適コア数を決定できず、性能向上と消費エネルギー削減は達成できなかった。

以上の結果から、稼働コア数をプログラム実行前に最適な数に決定し、稼働コアの動作周波数を上昇させて並列プログラムを実行する手法により従来の全コア実行に比べ性能向上と消費エネルギー削減が達成できると言える。しかしながら、実行時間モデルを用いた稼働コア数決定法により最適コア数を選択できていないことから、この稼働コア数決定法の精度は十分であるとは言えない。

4.3 提案手法の実行時間モデルによる性能予測の精度

blackscholes, dedup, ferret, streamcluster において、シミュレーションにより実測した性能と式 (3) のモデルにより計算した性能を比較し、実行時間モデルの妥当性を検証する。比較結果を図 7 に示す。グラフの横軸は稼働コア数、縦軸は動作周波数を上昇しない場合の 1 コア実行時の性能に対する性能比である。青線がシミュレーションにより実測した性能、赤線がモデルにより計算した性能を表わしている。

blackscholes では、シミュレーションにより実測した性能と実行時間モデルにより計算した性能がほぼ等しくなっている。そのため、実行時間モデルを用い稼働コア数決定法により最適コア数を選択できた。しかしながら、残りのプログラムでは実測した性能と計算した性能の差が大きい。このことから、実行時間モデルが妥当でないと言える。その理由として 2 つ考えられる。

1 つ目の理由は、あるクリティカルセクション内の処理を実行する際に、全稼働コアがそのクリティカルセクション内の処理を実行すると想定していることである。そのため、同一クリティカルセクションの処理を 1 コアずつ実行することになる。しかしながら、実際には全稼働コアが同一のクリティカルセクション内の処理を実行するとは限らない。そのクリティカルセクション以外の処理であれば他のコアにより並列実行できる。

2 つ目の理由は、性能向上阻害要因としてクリティカルセクションにおける排他制御のみを考慮していることである。文献⁹⁾によると、dedup と ferret はパイプライン並列化されたプログラムである。dedup と ferret はそれぞれ 5 ステージと 6 ステージに分割されており、どちらのプログラムでも最初のステージと最後のステージでは入出力を行うため 1 コアでしか実行できない。それ以外の中間ステージでは複数コアによる並列処理が可能である。しかしながら、両プログラムにおいて、各ステージ毎の実行時間が異なる。特に、ferret では 1 つのステージの実行時間が他ステージに比べ極端に長い。つまり、これらのプログラムを実行する際、負荷の不均一により並列処理性能向上が阻害されていると考えられる。

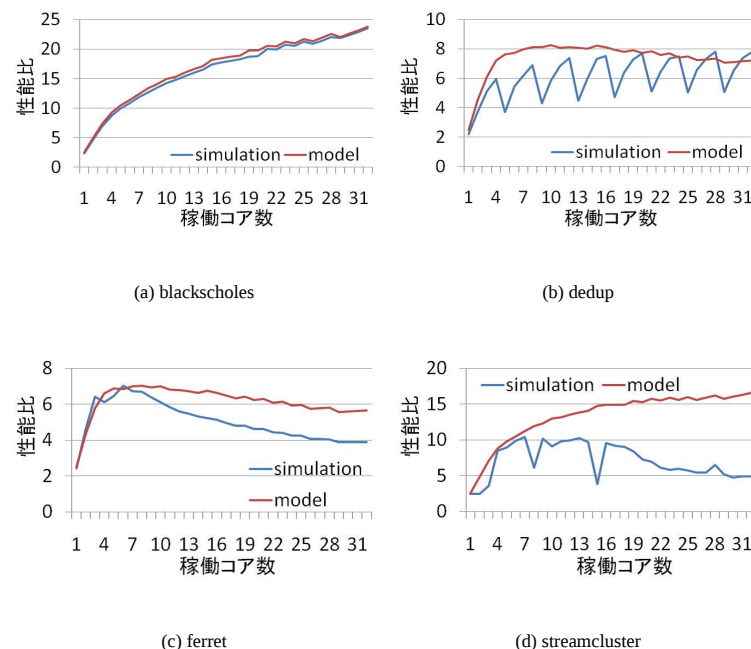


図7 シミュレーションによる実測結果とモデルによる計算結果の比較

また、streamcluster はバリア同期の数が多いプログラムである⁹⁾。バリア同期命令数が blackscholes では 8 であり、swaptions, dedup, ferret では 0 である。これに対し、streamcluster では、129600 である。streamcluster では頻繁にバリア同期が行われ、その度に処理完了が遅いコアを他の全コアが待たなければならない。つまり、streamcluster の実行においても負荷の不均一が性能に対し大きな影響を与える。

5. 関連研究

5.1 稼働コア数制限に基づく性能向上・消費電力削減手法¹⁰⁾

この手法では、稼働コア数増加に伴い性能向上が頭打ちになる場合や性能が悪化する場合に、稼働コア数を制限することで性能向上や消費電力削減を達成する。性能向上阻害要因と

してクリティカルセクションにおける排他制御とオフチップメモリバンド幅の制限に注目している。簡単に手法の説明を行う。

まず、コンパイルの際にコンパイラがループを 2 つの部分に分ける。その 1 つは、プログラムの実行中にクリティカルセクション内の処理にかかる時間やオフチップバスの使用率を計測し、そのループの実行に最適なコア数を推測するためのものである。そして、残りの部分では推測した稼働コア数により並列処理を行う。この手法では、プログラム実行中に最適な稼働コア数を推測し、稼働コア数を動的に変化させる。そのため、プログラムの実行状況に応じた並列処理が可能である。

稼働コア数増加に伴い性能が悪化する場合、この手法により性能向上と消費電力削減が達成できる。しかしながら、稼働コア数増加に伴い性能向上が頭打ちになる場合には、性能を維持したまま消費電力を削減するのみである。本稿で提案した手法では、動作を停止したコアに割り当てられていた消費電力バジェットを稼働コアに再割当てし、動作周波数を上昇させることでさらなる性能向上を狙える。

5.2 Intel 社 TurboBoost³⁾

TurboBoost とは、プロセッサが仕様で定められた温度や消費電力の上限値未満で稼働している時、プロセッサの動作周波数を定格動作周波数以上に上昇させる技術である。Intel 社製プロセッサでは、電力管理用コントローラで各コアの消費電力や温度を常に監視し、プロセッサの状態に応じて動作周波数を動的かつ自動的に変化させることができる。

TurboBoost はもともと逐次処理部分の実行を高速化する技術として開発されたものである。逐次処理部分を実行する際、1 コアしか稼働しないため全コア実行に比べ消費電力と温度が低くなり、その 1 コアの動作周波数を大幅に上昇できる。そのため、本稿で提案した手法により逐次処理部分の実行を高速化することに新規性はない。

ただし、TurboBoost では並列処理中にあえて稼働コア数を制限することはない。つまり、並列処理可能部分を実行する際、全コアにより並列処理を行うため、消費電力と温度が高くなり TurboBoost による動作周波数上昇は難しい。それに対し本稿の提案手法では、並列処理を行う際に稼働コア数を制限することで消費電力と温度を低減する。そして、稼働コア数に応じて稼働コアの動作周波数をあらかじめ設定した値まで上昇させる。そのため、提案手法に TurboBoost を応用することで、実行プログラムに適した数の稼働コア数を決定し、稼働コアの動作周波数を自動的に変化させることができる。

6. おわりに

本稿では、稼働コア数制限に基づくプロセッサ性能向上手法を提案した。性能向上阻害要因の影響により稼働コア数増加に伴う性能向上が得られない場合、稼働コア数をあえて制限し稼働コアの動作周波数を上昇させることで並列プログラムの実行を高速化する。本研究では、提案手法の実行時間モデルを用いて稼働コア数を静的に決定し、稼働コア数を固定してプログラムを実行する手法を評価した。実行時間モデルでは性能向上阻害要因としてクリティカルセクションにおける排他制御しか考慮していないため、最適コア数を正確に決定できなかった。しかしながら、最適コア数を既知とする場合、提案手法により従来の全コア実行に比べ最大で2倍以上の性能向上が得られ、消費エネルギーを最大約54%削減できた。よって、静的に稼働コア数を決定し、稼働コア数を固定してプログラムを実行する手法は有効であると考えられる。

今後の課題として、提案手法の実行時間モデルを再検討する。クリティカルセクション内の処理にかかる時間を見直し、バリア同期や処理量の不均一も考慮した実行時間モデルを考案する。そして、実行時間モデルを用いた稼働コア数決定法の精度を向上させる。

謝辞 日頃から御討論頂いております九州大学安浦・村上・松永・井上・アシル・杉原研究室ならびにシステムLSI究センターの諸氏に感謝いたします。なお、本研究は一部、半導体理工学研究センター（STARC）との共同研究ならびに科学研究費補助金（課題番号：21680005）による。

参考文献

- 1) Kurd, N., Mosalikanti, P., Neidengard, M., Douglas, J. and Kumar, R.: Next Generation Intel Core Micro-Architecture(Nehalem) Clockin, *ISSCC,IEEE*, pp.98–99 (2010).
- 2) Shin, J., Tam, K., Huang, D., Petrick, B., Pham, H., Hwang, C., Li, H., Smith, A., Johnson, T., Schumacher, F., Greenhill, D., Leon, A. and Strong, A.: A 40nm 16-core 128-thread CMT SPARC SoC processor, *IEEE*, Vol.44, No.4, pp.1121–1129 (2009).
- 3) Intel®Corporation: Intel® Turbo Boost Technology in Intel® Core Microarchitecture (Nehalem)Based Processors. Whitepaper, Intel® Corporation, November 2008.
- 4) Amdahl, G.: Validity of the single processor approach to achieving large scale computing capabilities, *Proceedings of the April 18-20, 1967, spring joint computer conference*, ACM, pp.483–485 (1967).
- 5) 福田晃：並列オペレーティングシステム，コロナ社 (1997).
- 6) Grochowski, E. and Annavaram, M.: Energy per instruction trends in Intel microprocessors,

Technology@ Intel Magazine, Vol.4, No.3, pp.1–8 (2006).

- 7) IBM: New IBM POWER6 processors for the Power 570 enable you to get more processing power with fewer processors. IBM United States Hardware Announcement, pp.108-722, dated October 7, 2008.
- 8) Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A. and Reinhardt, S.: The M5 simulator: Modeling networked systems, *Micro, IEEE*, Vol.26, No.4, pp.52–60 (2006).
- 9) Bienia, C., Kumar, S., Singh, J. and Li, K.: The PARSEC benchmark suite: Characterization and architectural implications, *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, pp.72–81 (2008).
- 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).