Vol.2009-ARC-184 No.11
2009/8/4

ワーキングセット評価に基づくスレッドスケジューリング

佐藤雅之[†1] 小寺功[*1] 江川隆輔[†2]
滝沢寛之[†1] 小林広明[†2]

現代のマルチコアプロセッサはメモリとの性能差を隠蔽するために大容量の共有キャッシュメモリを搭載している．しかし，共有キャッシュにおいて資源競合が発生する条件下では，共有キャッシュのエネルギ消費に見合うだけの性能向上が期待できない．本報告では，よりエネルギ効率の高いキャッシュメモリを実現するために，ワーキングセット評価に基づくスレッドスケジューリングを提案する．これは，適切なウェイ数をスレッドに割り当てる動的キャッシュ分割機構と，割り当てられなかったウェイへの電力供給を停止する省電力キャッシュ機構を導入し，ウェイ数の不足によって動的キャッシュ分割機構で発生する性能低下を抑制するスレッドスケジューリングを行う手法である．この手法により，キャッシュを共有するコア間でキャッシュ競合を最小化する．評価により，提案手法は最大 9.8%，平均 1.5%の性能向上を達成すると同時に，最大 46%，平均 13%のエネルギ消費を削減可能であることを明らかにする．

Thread Scheduling based on Assessment of Working Sets

Masayuki Sato ,[†1] Isao Kotera ,[*1]
Ryusuke Egawa ,[†2] Hiroyuki Takizawa [†1]
and Hiroaki Kobayashi[†2]

Modern high-performance multi-core processors have large shared cache memories. However, enlarging a shared cache memory does not always lead to the performance imrpovement due to inter-thread cache conflicts, while it makes major power-consuming resource. To realize an energy-efficient shared cache memory, this paper proposes a thread scheduling method based on assessment of working set size. Assuming that the dynamic cache partitioning mechanism with the power-aware way-adaptable cache mechanism are applied, the proposed thread scheduling method minimizes inter-thread cache conflicts among cores. The proposed method can achieve 9.8% better performance than the worst case scheduling and 1.5% better than the average, while the proposed method can reduce energy consumption by up to 46%, and 13% on average.

1. はじめに

現代のマルチコアプロセッサは，メモリとの性能差を埋めるために大容量の共有キャッシュメモリを搭載している．しかし，キャッシュを共有する複数のスレッドがそれぞれ異なるアプリケーションに由来する場合，スレッド間のデータ置換により，キャッシュ資源競合が発生する．スレッド間のデータ置換は，片方のスレッドのデータがキャッシュメモリに保存される時に，他方のスレッドのデータがライトバックされることである．この時，他方のスレッドが追い出されたデータに再びアクセスすると，シングルスレッド実行環境では発生しないミスが発生する．スレッド間のデータ置換が多く発生すると，各スレッドのミスが増加するためマルチコアプロセッサの性能が低下する[1)]．一方で，半導体製造プロセスの微細化により，マイクロプロセッサの消費電力に占める静的電力の割合が著しく増加している．各ユニットの静的電力は面積に比例して増加するため，巨大な共有キャッシュメモリはマイクロプロセッサの主要な電力消費源である[2)]．このため，マルチコアプロセッサでは巨大な共有キャッシュメモリを搭載しているにもかかわらず，キャッシュ資源競合によってキャッシュメモリが有効に利用されないまま電力のみを消費し続ける状況が発生する．

これらの問題を解決する，エネルギ効率の良い共有キャッシュメモリを実現することを目的として，本報告ではワーキングセット評価に基づくスレッドスケジューリングを提案する．本提案手法は，マルチコアプロセッサに複数の共有キャッシュが搭載されている場合に適用されることを想定している．本提案手法はワーキングセット評価に基づく省電力キャッシュ分割機構[3)] を前提として，スレッドスケジューリングを行う手法である．省電力キャッシュ分割機構は，キャッシュメモリをウェイ単位で領域分割し，各領域を各スレッドに対して専用領域として割り当てることによって，スレッド間のデータ置換を防止する．また，各スレッドの必要とするキャッシュ容量 (ワーキングセットサイズ) を把握し，各スレッドの性能向上に貢献しないウェイがある場合，ウェイへの電力供給を停止することによって省電力化を図る．スレッドスケジューリングは，各コアに対するスレッドの割り当てを決定する際

†1 東北大学 大学院情報科学研究科
Graduate School of Information Sciences, Tohoku Unviersity
†2 東北大学サイバーサイエンスセンター
Cyberscience Center, Tohoku University
∗1 現在，株式会社 ルネサス テクノロジ
Presently with Renesas Technology Corp.

に，キャッシュを共有するスレッドの総ワーキングセットサイズがキャッシュ容量をなるべく超えないようにする．提案手法により，スレッド間のデータ置換を防止しつつ，適切な容量をスレッドに対して割り当てることが可能になると共に，不必要なウェイへの電力供給を停止することができる．この結果，マルチコアプロセッサの高性能・低消費電力化を実現することができる．

本報告の構成について以下に述べる．第 2 節では関連研究について述べる，第 3 節では，ワーキングセット評価に基づくスレッドスケジューリングを提案し，その具体的な動作について述べる．第 4 節で提案手法について性能とエネルギ消費の観点から評価を行う．第 5 節で本報告をまとめる．

2. 関 連 研 究

2.1 キャッシュ分割機構

マルチスレッド処理でキャッシュを共有する場合，一方のスレッドのデータが他方のスレッドのデータによってキャッシュから追い出されると，キャッシュ資源競合が発生する．もし，一方のスレッドが他方のスレッドのデータに追い出されたデータに再びアクセスしようとすると，キャッシュミスが発生する．Kihm ら[1] はこのような置換を inter-thread kickouts(ITKO) と呼び，ITKO の増加とスレッドの実行速度の増加に相関関係があることを示している．

これに対し，ITKO を防ぐことを目的として動的キャッシュ分割機構が提案されている．動的キャッシュ分割機構ではキャッシュをウェイ単位で分割し，ウェイを各スレッドの専用領域として割り当てる．この割り当てにより，同じ領域に複数のスレッドのデータが混在することを防ぎ，ITKO を未然に防ぐことができる．また，動的キャッシュ分割機構はキャッシュを共有するスレッドのうち，どちらのスレッドがより多くのウェイを必要としているかを判断し，割り当てを変更する機能を有する．動的キャッシュ分割機構の研究として，共有キャッシュメモリ全体のミスを最小化するようにウェイの割当を行う Suh ら[4] や Qureshi ら[5] の手法が挙げられる．また，Lin ら[6] はオペレーティングシステムにおけるメモリアドレスマッピング手法を変更することによってキャッシュ分割を実機上で実現している．性能評価によって，過去のシミュレーションベースの研究結果以上に実機上でのキャッシュ分割機構の効果が高いことを明らかにしている．

しかし，動的キャッシュ分割機構の問題点として，特に大きなキャッシュ容量を必要とするスレッドの実行時に，ITKO を防止できるにもかかわらず，性能低下する場合があることが挙げられる．同時実行されている 2 つのスレッドのワーキングセットサイズの合計が共有キャッシュの容量を超える場合，動的キャッシュ分割機構は 2 つのスレッドが必要とするキャッシュ容量を割り当てることができない．このため，ITKO による性能低下を防ぐことができたとしても，キャッシュ容量の不足による性能低下が発生する．Moreto ら[7] は，各スレッドが単独実行時の 90%の性能を維持するために必要とするウェイ数の合計が，共有キャッシュの全ウェイ数を超える場合，動的キャッシュ分割機構による性能低下が大きくなることを報告してる．

2.2 キャッシュの省電力化

大量のオンチップ資源による消費電力の増加は，高性能マイクロプロセッサ設計において大きな制約になりつつある．特に，リーク電流による静的電力はトランジスタのゲート幅が小さくなるにつれて増加し，全体の消費電力の中で最も割合が大きくなっている．このことから，低消費電力なマイクロプロセッサを達成するためには静的電力を考慮することが不可欠である．

これまで，キャッシュメモリの電力を削減するための手法はいくつか提案されている．Albonesi ら[8] は selective cache ways を提案している．これは，キャッシュをデータアレイ単位で分割し，一部のアレイへのプリチャージを無効化することによって，動的電力を削減する．一方，静的電力を削減する手法として，Kaxiras ら[9] が cache decay を提案している．これは無効化されたキャッシュラインの電力供給を停止にすることによって，リーク電力を削減する．また，Flautner ら[10] は同様の目的で drowsy cache を提案している．Cache decay では電力供給の停止によりデータの保持ができないため性能低下が発生するという問題があったが，drowsy cache ではデータの保持に最低限必要な電力のみを供給することで性能劣化を回避している．しかし，これらの関連研究は単一コアのプロセッサのキャッシュメモリに関する研究であり，マルチコアプロセッサを前提とした省電力機能については検討されていない．

2.3 本研究の位置づけ

本研究はマルチコアプロセッサの共有キャッシュメモリにおける資源競合と電力消費増大の問題を解決し，エネルギ効率の高い共有キャッシュメモリを実現することを目標としている．

この目標を実現するため，これまでに我々は省電力キャッシュ分割機構[3] を提案している．この機構は，従来の動的キャッシュ分割機構と同様に，スレッドのワーキングセットのサイズに応じて必要なウェイ数を割り当てる．さらに，割り当てる必要のないウェイが生じた場

Vol.2009-ARC-184 No.11
2009/8/4

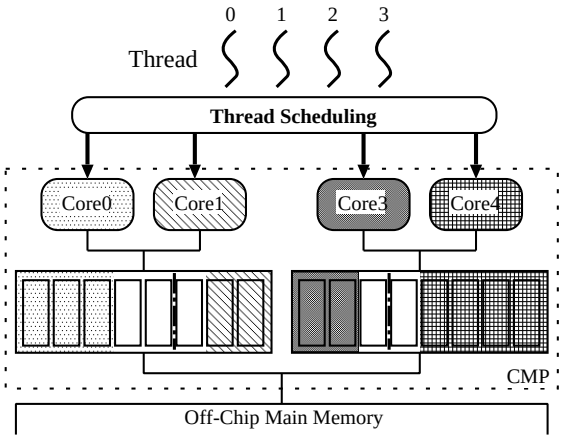

図 1 提案手法の概要

合，スレッドに割り当てを行わずに電力供給を停止し，消費電力を削減することができる．これにより，性能向上と低消費電力を同時に実現できる．しかし，これだけでは動的キャッシュ分割機構における性能低下の問題は解決することはできない．

動的キャッシュ分割機構における性能低下は，スレッドスケジューリングによって改善できると考えられる．プロセッサのマルチコア化によって，オンチップ・オフチップを問わず計算機に大量のコアとそれに付随する共有キャッシュが搭載されることになる．このため，キャッシュを共有するスレッドの組み合わせの決定の自由度が増し，スケジューリングの柔軟性が高まる．キャッシュを共有するスレッドの組のワーキングセットサイズが，共有キャッシュの容量を超えないように，スケジューリングすることもできる．このようなスレッドスケジューリングを行うことにより，動的キャッシュ分割機構による性能低下を抑制できる．本報告では，省電力キャッシュ分割機構による性能低下を回避し，よりエネルギ効率の向上を図るためのスレッドスケジューリングを提案する．

3. ワーキングセット評価に基づくスレッドスケジューリング

3.1 提案手法の概要

本節では，第 2 節の議論に基づき，ワーキングセット評価に基づくスレッドスケジューリング手法を提案する．図 1 に提案手法の概要を示す．複数の共有キャッシュメモリを持つマイクロプロセッサにおいて，省電力キャッシュ分割機構を各共有キャッシュメモリに採用する．さらに，提案手法ではキャッシュメモリを考慮したスレッドスケジューリングを行う．省電力キャッシュ分割機構はスレッドに対するウェイの適切な割り当てと，性能に貢献しないウェイの電力制御を行い，ITKO を阻止しつつ消費電力の増加を抑制する．キャッシュを考慮したスレッドスケジューリングは，ワーキングセットサイズが各共有キャッシュをなるべく超えないようにスケジューリングを行い，キャッシュ容量不足による性能低下を防ぐ．

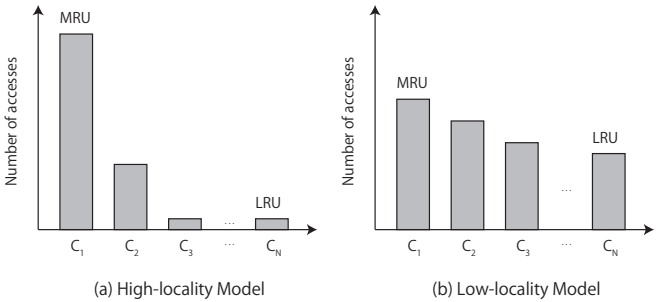

図 2 stack distance profiling の概念

3.2 局所性評価量

提案手法ではスレッドのワーキングセットをカバーするウェイ数を適切に評価する必要があるため，stack distance profiling[11] による局所性評価量[12] を用いたワーキングセット評価を行う．以下に局所性評価量について述べる．Stack distance profiling は，セットアソシアティブキャッシュにおけるスレッドの参照局所性を，同一データへの参照間隔の分布として表すことができる．図 2(a),(b) それぞれに stack distance profiling によって得られる分布を示す．図 2 の C_i $(1 \leq i \leq N)$ はデータ置換ポリシが LRU アルゴリズムで N ウェイのセットアソシアティブキャッシュにおいて，セットにおける LRU の値が i のデータに対するアクセス数を示す．この分布は，スレッド毎のキャッシュメモリに対する参照局所性の違いによって形状が変化する．図 2(a) は参照局所性が高い場合の分布である．参照局所性が高いと MRU データへのアクセスが多く，LRU データへのアクセス数が少ない．この場合，LRU データにヒットする確率は小さく，ウェイ数を減らした場合の性能低下は小さいと推測できる．一方，図 2(b) は参照局所性が低い場合の分布である．参照局所性が低い場合，アクセス数が MRU データから LRU データまで広く分散する．LRU データにヒットする割合から，ウェイ数を増加させた場合のアクセス増加による性能向上が期待できる．

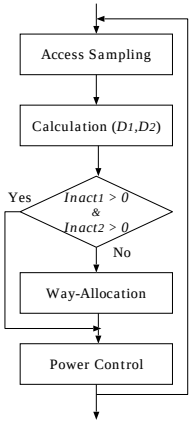

図 3 省電力キャッシュ分割機構の制御フロー（2 コア CMP）

この分布の違いを考慮し，ウェイの増減の必要性を評価するために以下の局所性評価量 D を用いる．

$$D = \frac{LRUcount}{MRUcount}. \tag{1}$$

式 (1) において，$LRUcount$ は LRU データに対するアクセス数，$MRUcount$ は MRU データに対するアクセス数である．

3.3 省電力キャッシュ分割機構

図 3 に，2 つのコアで 1 つのキャッシュメモリを共有する場合の省電力キャッシュ分割機構の制御フローを示す．この機構は，図 3 の制御フローに従ってワーキングセット評価のためのアクセスサンプリングと D 値の計算を行った後，2 種類の制御を行う．1 つ目のウェイ割当制御は，各コアが実行性能を維持するために必要なキャッシュ容量を推定し，全コアが性能を発揮できるようにキャッシュ容量を公平に割り当てる制御である．2 つ目の電力制御は，各コアで実行されているスレッドが割り当てられた全容量を必要としていない場合に，消費電力削減のために一部のウェイを不活性化する電力制御である．これら 2 つの制御は図 3 の制御フローを一定の期間毎に行うことによって行われる．また，両方のコアで不活性となるウェイがある場合，ウェイの割り当てを変更しても性能に影響を及ぼさない．このため，コア i $(i = 0, 1)$ に割り当てられている不活性ウェイ数を $Inact_i$ とおくとき，$Inact_i > 0 (i = 0, 1)$ であれば，ウェイ割当の変更は行わない．

3.3.1 ウェイ割当制御

省電力キャッシュ分割機構の 1 段階目であるウェイの割当制御について述べる．一般的なキャッシュメモリはウェイと呼ばれる単位で構成される．本機構ではウェイ単位で各コアにキャッシュを割り当てる．

制御には局所性評価量 D を用いて，キャッシュ参照の局所性を考慮に入れたウェイの割当を行う．まず，コア i $(i = 0, 1)$ からのアクセスについて D_i を求める．求めた D_i について

$$\text{If } D_j > D_k \quad \begin{cases} Alloc_j & +=1 \\ Alloc_k & -=1 \end{cases} \tag{2}$$

となる割当を行う．ここで $Alloc_i$ はコア i に対する割当ウェイ数である．また，共有キャッシュの総ウェイ数以上のウェイを割り当てることはできないので，全ウェイ数を $Alloc_{all}$ とするとき，

$$Alloc_{all} = \sum Alloc_i \tag{3}$$

である．これにより，参照局所性の程度に応じたウェイ割当を行うことができる．

3.3.2 電 力 制 御

電力制御では way-adaptable cache の制御手法[12] を利用し，D_i を絶対値と比較することによって D_i に応じてウェイを活性化・不活性化し，性能低下を抑制しつつ消費電力の削減を行う．

効果的で適切なウェイ数の制御を行うためには，スレッドが実行中の時点で必要としているワーキングセットに見合ったウェイ数をキャッシュの利用状況から予測する局所的判断と，過渡的な要求の変化に過剰反応しないように，時系列から全体的な傾向を把握する大域的判断の 2 つが必要である．省電力キャッシュ分割機構では局所的判断のために局所性評価量，大域的判断のためにステートマシンをそれぞれ用いる．

3.3.2.1 局所的判断

局所的なキャッシュの要求量を求めるために，式 (1) で定義された局所性評価量 D を再び用いる．閾値 T_1, T_2 $(T_1 < T_2)$ を導入し，D が閾値 T_1 以下の場合は，MRU データ側に著しくアクセスが集中していると判断し，ウェイを 1 つ不活性化するべきと判断する．また，D が別の閾値 T_2 を超えている場合は，LRU データ側にもアクセスが分散していると判断し，不活性化されているウェイを 1 つ活性化すると判断する．T_1, T_2 の値は大きくなるほど不活性化判断する範囲が大きくなり，より少ないウェイ数で稼働するようになる．一

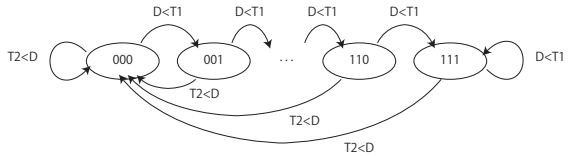

図 4 3bit ステートマシン

方，キャッシュでのヒット数の削減量が大きくなるため，性能低下が発生しやすくなる．したがって，T_1, T_2 の値の設定によって性能と消費電力のトレードオフを調整することができる[3]．

3.3.2.2 大域的判断

前節で，D を閾値と比較することで，増加，減少，維持の 3 つの判定を行うことを示した．しかし，このような判定はアクセスの分布が急激に変化するアプリケーションの場合，ウェイの活性化と不活性化を短期間で繰り返す恐れがある．ウェイを不活性化する場合，ウェイに保存されているダーティなデータをライトバックしなければならないため，ウェイの活性化と不活性化を短期間で繰り返すとライトバックの回数が増加する．この結果，キャッシュの増減制御に伴うオーバーヘッドが著しく増加する．これに対して，より長い期間で安定した状況を見つけ出し，ウェイ数の増減によって引き起こされるオーバーヘッドを抑制し，キャッシュを安定して制御することが必要である．本機構では制御の安定のために x ビットステートマシンを用いる．

例として図 4 に非対称の 3 ビットステートマシンの状態遷移図を示す．3 ビットカウンタで表すことのできる 8 個の状態を考える．カウンタが 000 の状態になるとウェイを増加させる判断を行う．また，111 の状態になるとウェイを減少させる判断を行う．これ以外の状態ではウェイ数を維持する．このステートマシンに対する入力は局所的判断により得られた結果である．局所的判断によりウェイ数を増加するべきと判定された場合は状態 000 に遷移し，逆に減少するべきと判定した場合，カウンタを+1 した状態に遷移する．局所的判断による減少判断が続いた場合，最終的に 111 の状態に到達する．このように，ステートマシンによってフィルタをかけることで，高い周期で増加減少を繰り返すことを防ぎ，オーバーヘッドの少ない安定した制御が可能となる．また，増加判定しやすい非対称のステートマシンにすることによって，ウェイの増加要求に対しては早急に対応し，性能低下を抑制することができる．

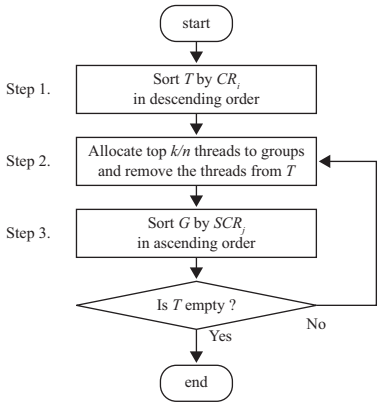

図 5 スケジューリングアルゴリズムのフローチャート

3.4 スレッドスケジューリング

3.4.1 プロファイリング

スレッドスケジューリングを行う前にワーキングセットサイズの評価を行う．本提案手法ではワーキングセットサイズ評価に省電力キャッシュ分割機構の電力制御部分に相当する way-adaptable cache[12] を用い，シングルスレッド実行でプロファイリングを行うことによりウェイ数単位でワーキングセットサイズを算出する．スレッドのワーキングセットサイズ CR_{thread} は way-adaptable cache が割り当てたウェイ数の時間平均として以下に定義する．

$$CR_{thread} = \frac{\sum_{t=t_0}^{t_1} W(t)}{t_1 - t_0}. \quad (4)$$

式 4 において，$W(t)$ は時間 t $(t_0 \leq t \leq t_1)$ において割り当てられたウェイ数である．

3.4.2 スケジューリングアルゴリズム

プロファイリングによって得られたワーキングセット評価の結果から，ワーキングセットサイズの合計が共有キャッシュの容量を超えないようにスレッドスケジューリングを行う．このようなスケジューリングを実現するためには，ワーキングセット評価の結果に基づき，どのコアでどのスレッドを実行するかを決定するアルゴリズムが必要である．以下，本提案手法で用いるスケジューリングのアルゴリズムについて述べる．

本アルゴリズムの目的は．各共有キャッシュメモリを共有するスレッドのワーキングセッ

トサイズの和が，可能な限り等しくなるようにスレッドの組み合わせを作成することである．本アルゴリズムでは，k コアのマルチコアプロセッサに k 個のスレッドをスケジューリングする場合を想定する．また，k/n 個の共有キャッシュメモリがあり，共有キャッシュはそれぞれ n 個のコアによって共有される．提案手法のスケジューリングアルゴリズムを図 5 に示す．初期状態では，k 個のスレッドからなるリスト T と k/n 個のグループからなるリスト G を用意する．ここで，各グループはそれぞれ 1 つの共有キャッシュを表しており，アルゴリズムの終了時に 1 つのグループに割り当てられたスレッドが，同じキャッシュを共有するべきスレッドの組み合わせとなる．

Step1 では，ワーキングセットサイズに基づいてリスト T 中のスレッドを降順にソートする．図 5 の CR_i は i 番目のスレッドのワーキングセットサイズを示す．Step2 ではリスト T の先頭にある k/n のスレッドがそれぞれのグループに割り当てられ，リスト T から削除される．この結果，リスト T の先頭から i 番目のスレッドが i 番目のグループに割り当てられる．Step3 ではグループのリスト G が，グループ内に割当済みのスレッドのワーキングセットサイズの合計によって昇順にソートされる．図 5 の SCR_j はグループ j に割り当てられたスレッドのワーキングセットサイズの合計を表す．このステップは次のイテレーションの Step2 において，最も SCR_j の小さいグループに対して，最も CR_i の大きいスレッドが割り当てられることを保証するものである．最後に，もしリスト T が空でなければ Step2 に戻る．リスト T が空であれば，最後の Step3 は行わずにスケジューリングアルゴリズムは終了する．

4. 性能評価

4.1 評価環境

シミュレーションにより提案手法の有効性を評価する．プロセッサシミュレータ M5 version 1.1[13] とキャッシュの電力モデル CACTI version 4.2[14] を用いてシミュレータを開発した．評価したマルチコアプロセッサのシミュレーションパラメータを表 1 に示す．このマルチコアプロセッサは，全体で 4 つのコアと 2 つの L2 共有キャッシュを持ち，2 つのコアで 1 つの L2 キャッシュを共有する．各 L2 共有キャッシュには省電力キャッシュ分割機構が適用されている．性能と電力のトレードオフを決定するしきい値は，性能低下の少ない $(T_1, T_2) = (0.001, 0.005)$ とする[3]．プロセッサ上で使用するスレッドは SPEC CPU2000 ベンチマークから 4 つのベンチマークを選択し，各ベンチマークをスレッドとして実行する．本評価では，スレッドは各コアに 1 スレッドずつ割り当てられるものとし，シミュレーション開始時に一斉に実行開始する．シミュレーションは実行開始から 10 億サイクル分実行するものとする．どのコアにどのスレッドを割り当てるかは，提案手法のスレッドスケジューリングアルゴリズムを基に決定する．

評価では 6 個のベンチマークを SPEC CPU2000 ベンチマーク集から選択し，さらに 4 個のベンチマークからなるスレッドの組を作成してシミュレータ上で実行する．以下にベンチマークの選択方法を述べる．SPEC CPU2000 ベンチマーク集の各ベンチマークを，キャッシュメモリの利用特徴を表す utility graph[5] によって 3 種類のクラスに分類する．それぞれのクラスは high-utility(H), saturating-utility(S), low-utility(L) である．各クラスから代表的な 2 つのベンチマークを選出し，合計 6 つのベンチマークを選出する．選出した 6 つの

表 1 シミュレーションパラメータ

Parameter	Value
fetch width	8 instructions
decode width	8 instructions
issue width	8 instructions
commit width	8 instructions
inst. queue	64 instructions
L1 I-cache	32kB, 2-way, 32B-line 1 cycle latency
L1 D-cache	32kB, 2-way, 32B-line 1 cycle latency
L2 cache	1MB, 32-way, 64B-line LRU policy, 14 cycle latency
main memory	100 cycle latency
frequency	1 GHz
Technology	70 nm
Vdd	0.9V

表 2 実験に用いたベンチマーク

ベンチマーク	内容	utility	Working set (way)
vpr_place	FPGA Circuit Placement	H	32
twolf	Place and Route Simulator	H	32
equake	Wave Propagation Simulation	S	18
mesa	3-D Graphics Library	S	11
wupwise	Quantum Chromodynamics	L	9
applu	Partial Differential Equation	L	7

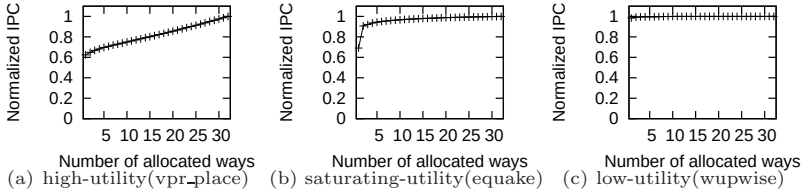

図 6 Utility graphs

表 3 実験に用いたベンチマークの組

組	ベンチマーク			
HHSS	vpr_place	twolf	equake	mesa
HHSL	vpr_place	twolf	equake	wupwise
HHLL	vpr_place	twolf	wupwise	qpplu
HSSL	vpr_place	equake	mesa	wupwise
HSLL	vpr_place	equake	wupwise	applu
SSLL	equake	mesa	wupwise	applu

ベンチマークを表 2 に示す．また，それぞれのクラスの代表的な utility graph を図 6 に示す．vpr_place, twolf に代表される high-utility はウェイ数の増加に従って性能が向上するスレッドである．このことから，より多くのウェイ数を必要とするスレッドである．equake, mesa に代表される saturating-utility はウェイ数が少ない段階ではウェイ数の増加によって性能が向上するが，ウェイ数が多い段階では性能が変化しにくくなるスレッドである．このことから，中程度のウェイ数を必要とするスレッドであるといえる．wupwise, applu に代表される low-utility はウェイ数の変化によって性能がほとんど変化しないスレッドである．このことから，low-utility のスレッドはほとんどウェイ数を必要としない．また，表 2 にプロファイリングで求められたワーキングセットサイズを示す．この傾向は表 2 で示した utility の傾向に一致する．よって，提案手法のプロファイリングによって得られたワーキングセットサイズは，ほぼ正しい傾向を示しているといえる．

これらのベンチマークから評価に用いる 6 つのスレッドの組を作成する．6 つのスレッドから 4 スレッドで構成される組を作成する場合，全部で 15 組 ($_6C_4 = 15$) になるが，utility で同じ特徴を持つ組については，同じキャッシュの利用特徴を持つ組であると判断し，評価では省略する．例えば [vpr_place, equake, wupwise, applu] の組と [twolf, mesa, wupwise, applu] の組はどちらも 1 つの high-utility，1 つの saturating-utility，2 つの low-utility のスレッドからなるため，キャッシュの利用特徴が同じであると考えられる．このようにして選抜したスレッドの組を表 3 に示す．

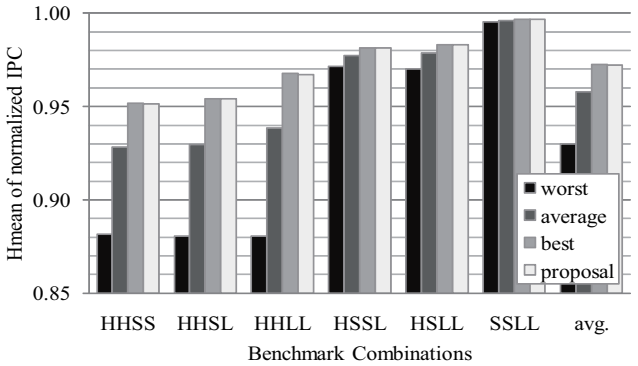

図 7 最悪・平均・最良の性能を示すスケジューリングポリシと提案手法の性能

性能評価指標として正規化 IPC の調和平均を用いる[15]．この指標は以下のように定義される．

$$(HarmonicMean) = \frac{N}{\sum_{i=0}^{N-1} \frac{IPC_i(single)}{IPC_i(multi)}}. \quad (5)$$

ここで N はスレッド数，$IPC_i(single)$ は i 番目のスレッドを単独で実行した場合の IPC，$IPC_i(multi)$ は同時実行した場合の i 番目のスレッドの IPC である．この指標によって，シングルスレッド実行時の性能に対するマルチスレッド実行時の性能を比較可能である．この比較により，提案手法やその他のスケジューリングポリシでキャッシュ資源競合による性能低下の大きさを知ることができる．また，単純な算術平均と比較してスレッド間の平等性を含めて評価することが可能である[16]．

4.2 評価結果

4.2.1 性能評価

図 7 は提案手法とその他のスケジューリングによる性能を示すグラフである．図 7 において，最悪の性能を示したスケジューリングの性能が worst，全スケジューリングの平均性能を average，最良の性能を示したスケジューリングの性能が best である．proposal は提案手法のスケジューリングによる性能である．図 7 より，worst と比較して，proposal では平均で 4.8%，最大で 9.8%の性能向上が得られていることがわかる．よって，提案手法は

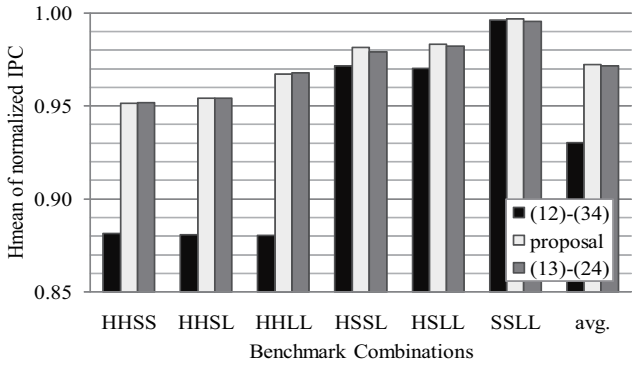

図 8 各スケジューリングポリシの性能

最悪の性能を示すスケジューリングを回避し，性能向上を実現可能である．また，average と比較した場合，proposal によって平均 1.5%の性能向上が得られている．さらに，best との比較では，実験を行った 6 つのスレッドの組のうち，4 組で proposal が最良の性能を示すスケジューリングとなっている．HHSS と HHLL の 2 組では proposal は最良の性能を示すスケジューリングではなかったが，その性能差は微小であることがわかる．以上から，提案手法によってほぼ最良の性能が得られるスケジューリングを選択可能であることが明らかになった．

次に，それぞれのスケジューリングの効果を詳細に解析するため，図 8 に各スレッドの組で考えられる全てのスケジューリングの性能を示す．ここで，(12)-(34) はワーキングセットサイズが第 1 位と第 2 位のスレッドが同じキャッシュを共有し，第 3 位と第 4 位のスレッドが同じキャッシュを共有するスケジューリングを表す．これは，第 1 位と第 2 位のスレッドが同じキャッシュを共有するため，キャッシュで容量不足による性能低下が発生しやすいスケジューリングである．Proposal は提案手法であり，第 1 位と第 4 位のスレッドがキャッシュを共有し，第 2 位と第 3 位のスレッドがキャッシュを共有するスケジューリングである．(13)-(24) は第 1 位と第 3 位のスレッドがキャッシュを共有し，第 2 位と第 4 位のスレッドがキャッシュを共有するスケジューリングである．

図 8 から，ほとんどの場合において競合が発生しやすいスケジューリングである (12)-(34) が最悪の性能を示しており，性能低下も大きいことが明らかである．このため，(12)-(34) スケジューリングを避けることは必須であることがわかる．HHLL の組では，競合が発生しやすい (12)-(34) スケジューリングに対して，提案手法による性能向上が最大となっている．この組はワーキングセットサイズが最大となる 2 つのスレッドと最小となる 2 つのスレッドで構成されている．この場合，(12)-(34) スケジューリングで発生する容量不足の程度が大きくなるため，スレッドの性能低下が最も大きくなる．したがって，提案手法によってこの組で発生するキャッシュ容量の不足を解消することにより，得られる性能向上が最大になる．一方で SSLL の組ではほとんど性能向上が得られていない．これは，SSLL の組に含まれるスレッドが元来キャッシュをあまり必要としないスレッドであるためである．このため，スケジューリングの有無にかかわらず省電力キャッシュ分割機構が十分なウェイの割当を行うことができ，シングルスレッド実行時と比べて性能低下はほとんど発生しない．

また，図 8 において proposal と (13)-(24) を比較した場合，性能は拮抗している．このことからも，最悪の性能を示す (12)-(34) スケジューリングを避けることが最も重要であることがわかる．また，HSSL の組は提案手法のスケジューリングによる性能向上が，他の組より比較的大きい．この組では，(13)-(24) スケジューリングによって片方の共有キャッシュを utility が H と S のスレッドが共有するため，このときのワーキングセットサイズが提案手法のスケジューリングと比較して共有キャッシュのウェイ数より大きくなる．この結果，その共有キャッシュにおける競合が大きくなることによって (13)-(24) スケジューリングによる性能向上が得られにくくなり，提案手法のスケジューリングが優位になると考えられる．但し，今回実験に用いたスレッドの組における proposal スケジューリングと (13)-(24) スケジューリングの性能差がほとんどないことから，両者の有効性とその要因の検討については今後の課題とする．

4.2.2 電力評価

図 9 に電力制御を行わないキャッシュを用いた全てのスケジューリングの平均のキャッシュの消費エネルギ (conventional) と提案手法を適用した場合のキャッシュの消費エネルギ (proposal) を示す．ここで，エネルギ消費は 1 億命令あたりの実行に必要なエネルギ消費で表している．電力制御を行わなかった場合と提案手法を適用した場合を比較すると，平均で約 13%の消費エネルギを削減できることが明らかになった．これにより，提案手法はエネルギ消費の削減に有効であることが示された．

HHSS と HHSL，HHLL の組では消費電力の削減効果は比較的小さい．これらのスレッドの組ではワーキングセットサイズの大きいスレッドが多くのウェイ数を必要とする．このため，提案手法によって不活性にできるウェイ数が減少する．しかし，このような場合であっても提案手法によって 2%程度のエネルギ消費を削減できている．一方で SSLL の組で

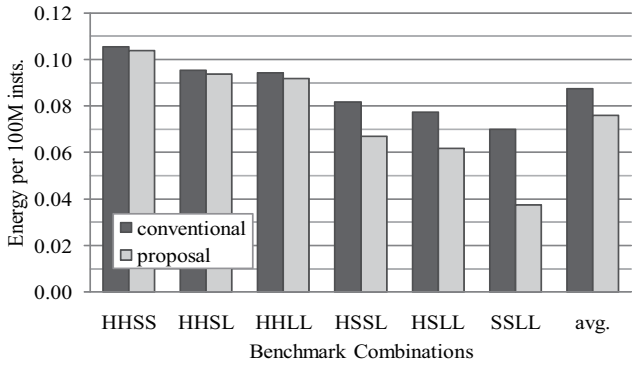

図 9 電力制御を行わなかった場合と提案手法の消費エネルギ

は提案手法によって最大となる 46%のエネルギ消費を削減できている．この組では，ワーキングセットサイズがあまり大きくないスレッドが多いため，提案手法によって性能に貢献しない多くのウェイを不活性化することが可能なためである．

5. おわりに

本報告では，マルチコアプロセッサのキャッシュメモリのエネルギ効率を向上させることを目的とし，ワーキングセット評価に基づくスレッドスケジューリングを提案した．この手法はキャッシュを考慮したスレッドスケジューリングと省電力キャッシュ分割機構を協調させ，性能向上を達成しつつエネルギ消費を削減することができる．評価実験では平均で 1.5%の性能向上，18.5%のエネルギ消費削減を達成し，提案手法が共有キャッシュメモリのエネルギ効率の改善に有効であることが示された．

今後の課題は，動的なスレッドスケジューリングとマイグレーションの実現によって，スレッドで実行されるアプリケーションのフェーズ変化によるキャッシュ利用特徴の変化に対応可能とすることが挙げられる．これにより，さらにエネルギ効率の高いスレッドスケジューリングを達成することができる．また，キャッシュを共有するスレッド数を柔軟に変更可能なスケジューリングの検討を行う．

参 考 文 献

1) Kihm, J., Settle, A., Janiszewski, A. and Connors, D.: Understanding the Impact of Inter-Thread Cache Interference on ILP in Modern SMT Processors, *The Journal of Instruction-Level Parallelism*, Vol.7 (2005).
2) Wong, W.-F., Koh, C.-K., Chen, Y. and Li, H.: VOSCH: Voltage Scaled Cache Hierarchies, *25th International Conference on Computer Design*, pp.496–503 (2007).
3) Kotera, I., Abe, K., Egawa, R., Takizawa, H. and Kobayashi, H.: Power-Aware Dynamic Cache Partitioning for CMPs, *Transaction on High-Performance Embedded Architectures and Compilers*, Vol.3, No.2, pp.149–167 (2008).
4) Suh, G., Rudolph, L. and Devadas, S.: Dynamic Partitioning of Shared Cache Memory, *Journal of Supercomputing*, Vol.28, No.1, pp.7–26 (2004).
5) Qureshi, M.K. and Patt, Y.N.: Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches, *Proceedings of 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.423–432 (2006).
6) Lin, J., Lu, Q., Ding, X., Zhang, Z. and Sadayappan, P.: Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems, *Proceedings of IEEE 14th International Symposium on High-Performance Computer Architecture*, pp.367–378 (2008).
7) Moreto, M., Cazorla, F.J., Ramirez, A. and Valero, M.: Explaining Dynamic Cache Partitioning Speed Ups, *IEEE Computer Architecture Letters*, Vol.6, No.1, pp.1–4 (Jan. 2007).
8) Albonesi, D.H.: Selective Cache Ways: On-Demand Cache Resource Allocation, *Proceedings of 32nd Annual International Symposium on Microarchitecture*, pp. 248–259 (1999).
9) Kaxiras, S., Hu, Z. and Martonosi, M.: Cache Decay: Exploiting Generational Behavier to Reduce Cache Leakage Power, *ACM SIGARCH Computer Architecture News*, Vol.29, No.2, pp.240–251 (2001).
10) Flautner, K., Kim, N.S., Martin, S., Blaauw, D. and Mudge, T.: Drowsy Caches: Simple Techniques for Reducing Leakage Power, *Proceedings of 29th Annual International Symposium on Computer Architecture*, pp.148–157 (2002).
11) Chandra, D., Guo, F., Kim, S. and Solihin, Y.: Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture, *Proc. the 11th International Symposium on High-Performance Computer Architecture*, pp.340–351 (2005).
12) Kobayashi, H., Kotera, I. and Takizawa, H.: Locality analysis to control dynamically way-adaptable caches, *ACM SIGARCH Computer Architecture News*, Vol.33, No.3, pp.25–32 (2005).

Vol.2009-ARC-184 No.11
2009/8/4

13) Binkert, N.L., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G. and Reinhardt, S.K.: The M5 Simulator: Modeling Networked Systems, *IEEE Micro*, Vol.26, No.4, pp.52–60 (2006).
14) Willton, S. J.E. and Jouppi, N.P.: CACTI: an enhanced cache access and cycle time model, *IEEE Journal of Solid-State Circuits*, Vol.31, No.5, pp.677–688 (1996).
15) Luo, K., Gummaraju, J. and Franklin, M.: Balancing throughput and fairness in SMT processors, *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, pp.164–171 (2001).
16) Chang, J. and Sohi, G.S.: Cooperative cache partitioning for chip multiprocessors, *Proc. the 21st annual ACM International Conference on Supercomputing*, pp. 242–252 (2007).