社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8（2）
2008／3／27

チップマルチプロセッサ用の優先度付きNon-Uniformキャッシュアーキテクチャ

坂本 伸昭† 山崎 信行†

† 慶應義塾大学大学院 理工学研究科
〒223-8522 神奈川県横浜市港北区日吉3-14-1
E-mail: †{sakamoto,yamasaki}@ny.ics.keio.ac.jp

あらまし 近年では，組み込みリアルタイムシステムにおいてもチップマルチプロセッサ (CMP) のように処理能力の高いプロセッサが要求されている．CMPでは高次キャッシュの構成方法が重要となる．近年提案されているNon-Uniform Cache Architecture (NUCA) は，コア間のラインマイグレーションを活用することで，アクセスレイテンシを短く保ちつつ，ヒット率を高めることができる．しかしながら，従来のNUCAにおけるラインマイグレーションは，ラインの状態やアクセス頻度に基づくものであるため，リアルタイムシステムにおいては高優先度スレッドのラインが低優先度スレッドの実行によって追い出され性能が低下する優先度逆転問題が生じる可能性がある．本論文ではこの問題を解決するため，優先度に基づくラインマイグレーションを提案する．また単純な優先度による制御だけでなく，再利用されないライン (dead block) を予測し，高優先度であってもチップ外へ追い出すことで，高優先度スレッドの性能を保証しつつ全体性能を向上させる．

キーワード チップマルチプロセッサ，Non-Uniformキャッシュアーキテクチャ，リアルタイムシステム

Prioritized Non-Uniform Cache Architecture for Chip Multiprocessors

Nobuaki SAKAMOTO† and Nobuyuki YAMASAKI†

† Graduate School of Science and Technology, Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522 Japan
E-mail: †{sakamoto,yamasaki}@ny.ics.keio.ac.jp

Abstract In recent years, high performance processors like Chip Multiprocessors (CMPs) have been required even in embedded real-time systems. The performance of CMPs often depends on the configuration of the last-level caches. The traditional Non-Uniform Cache Architecture (NUCA) can achive both short latency and high hit ratio by utilizing the migration of cache blocks. However, the exisiting migration policies for NUCA are based on the state of the cache blocks or the access frequency, and they may cause priority inversion problems in real-time systems. In order to solve the problem, we propose the migration policies based on the priority. In addition, we replace the blocks if they are predicted as dead blocks to improve the total performance.

Key words Chip Multiprocessors, Non-Uniform Cache Architecure, Real-Time Systems

1. 序 論

近年では，ヒューマノイドロボットやユビキタスアプリケーション等の出現により，組み込みリアルタイムシステムにおいても，処理能力の高いプロセッサが必要となってきている．しかしながら，ハードウェア資源や消費電力に制限のある組み込みシステムでは，単純に動作周波数を上げることによって性能を向上させることは難しい．このような背景から，チップマルチプロセッサ (CMP) のようにスレッドレベルの並列性を利用することで性能を向上させるプロセッサが注目されている．

コアあたりのメモリバンド幅がユニプロセッサに比べて不足するCMPでは，高次キャッシュ(注1)のヒット率を高める必要が

(注1)：本論文では二次キャッシュに関する議論を行う．なお三次以降の高次キャッ

ある．このためCMPでは一般的にコア間で高次キャッシュを共有する構成方法がとられる．しかしながら，共有キャッシュ構成はコアとキャッシュとの間に結合網が入るため，半導体プロセスの微細化によって年々増加傾向にある配線遅延の影響を受けてしまう．そこで配線遅延の影響を緩和させるためにNon-Uniform Cache Architecture (NUCA) [1] が提案された．CMPにおけるNUCAは，コア間のラインマイグレーションを活用することで，ヒット率を高く保ちつつ平均のアクセスレイテンシを短くすることができる．

NUCAは性能向上を目的とした場合には有効であると考えられる．しかしながら，リアルタイム処理を目的とした場合には，複数のコアで同時実行しているスレッド間でキャッシュメモリの競合が発生し，高優先度スレッドのラインが低優先度スレッドの実行によってチップ外へ追い出され性能が低下する優先度逆転問題が生じる可能性がある．本論文ではこの問題を解決するためにスレッドの優先度に基づくラインマイグレーションを提案する．また単純な優先度による制御だけでなく，再利用されないライン (dead block) を予測し，高優先度であってもチップ外へ追い出すことでキャッシュメモリの利用効率を向上させる．本論文では以上の手法により，高優先度スレッドの性能を保証しつつ，全体性能を向上させることを目的とする．

2. リアルタイム性

リアルタイム性とは，処理の真偽が時間にも依存するという性質である．狭義には与えられた時間制約 (デッドライン) を守るということを意味する．単純なシステムでは，プログラマがすべての起こり得る状況を想定して時間制約を満たすようにプログラムを作成することが可能である．しかしながら，ネットワークで相互に接続されたような高度で複雑なシステムでは，すべての起こり得る状況を想定することが不可能であるため，リアルタイムスケジューリングが必要となる．古典的なリアルタイムスケジューラにはEDF(Earliest Deadline First) やRM(Rate Monotonic) 等があるが，大抵のリアルタイムスケジューラはデッドラインや周期を優先度に変換し，ある一定時間間隔毎に優先度に従ってプリエンプションを行いながら実行する．例えば，EDFではデッドラインが近いタスクに対して高い優先度を与えて動的にスケジューリングを行う．

3. Non-Uniform Cache Architecture

ユニプロセッサにおけるNUCA [1] では，キャッシュメモリを複数のバンクに分割し，アクセス頻度の高いラインをコアの近くのバンクに配置することで，平均のアクセスレイテンシを短くすることができる．CMPにおけるNUCAも同様の考えに基づく構成方法である．具体的な構成方法は図1に示すように，各バンクを物理的に近い位置にあるコアへ直接接続する．これにより，各コアは自身に接続されているバンクへは高速にアクセス可能となる．また，あるラインがリプレースの対象となった場合，チップ外へ追い出すのではなく，可能であれば他

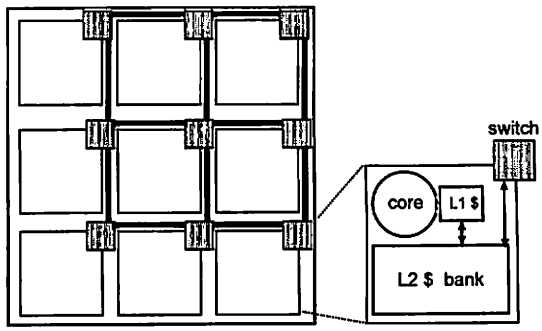

図1 CMPにおけるNUCAの構成例

のバンクへマイグレーションを行う．そして再度そのラインが必要になった場合にはそのバンクからフェッチを行う．複数のコアを単一チップ上に集積するCMPでは，他のバンクに対するアクセスも高速に行うことが可能であるため，ラインマイグレーションを活用することで，ヒット率を高く保ちつつ平均のアクセスレイテンシを短くすることができる．ただしマイグレーション先のキャッシュヒット率に悪影響を及ぼさないように，ターゲットは適切に選択されなければならない．

3.1 既存のマイグレーションポリシ

Changら [2] はチップ上に複数存在するライン (replicate) よりも，ユニークなライン (singlet) を多くチップ上に残すようにマイグレーションを行うcooperative cacheを提案した．replicateの内の1つがチップ外へ追い出されたとしても，再度そのラインが必要となった場合には，他のコアからフェッチできるため，この手法によりキャッシュメモリの利用効率を向上させることができる．

Speightら [3] の手法では，他のキャッシュのヒット率に悪影響を及ぼさないように，他のコアでも利用されると予測される場合にのみマイグレーションを行う．この手法は主に並列プログラムの性能を向上させるのに有効である．

塩谷らは [4] チップ上の全ラインのLRU情報を基にマイグレーションを行う手法を提案した．この手法はLRU情報を保持する機構へのアクセスがボトルネックとなる可能性があるが，頻繁にアクセスされるラインをチップ上に多く残すことができる．

3.2 リアルタイム処理に対する問題点

上述した既存のマイグレーションポリシは，ラインの状態やアクセス頻度に基づくものであり，全体性能の向上を目的とした場合には有効であると考えられる．しかしながらリアルタイム処理を目的とした場合には，複数のコアで同時実行しているスレッド間でキャッシュメモリの競合が発生し，高優先度スレッドのラインが低優先度スレッドの実行によって追い出され性能が低下する優先度逆転問題が生じる可能性がある．

この問題は従来の共有キャッシュ構成においても生じる．佐藤 [5] らはリアルタイム処理用マルチスレッドプロセッサの共有キャッシュにおいて優先度に基づくリプレースメントポリシを提案した．佐藤らの手法ではライン毎にスレッドの優先度を

シュに関しても同様の議論が成り立つ．

付加し，優先度に従ってリプレースを行うことで，高優先度スレッドのキャッシュミスを減少させ性能を保証する．しかしながら，単純な優先度制御では高優先度スレッドの性能は保証できたとしても，低優先度スレッドのヒット率に悪影響を及ぼし，全体性能が大きく低下してしまうという問題がある．

4. 優先度に基づくラインマイグレーション

上述したように NUCA における既存のマイグレーションポリシは，ラインの状態やアクセス頻度に基づくものであるため，高優先度スレッドのラインが低優先度の実行によって追い出され性能が低下する優先度逆転問題が生じる可能性がある．そこで本論文では佐藤らの考えを基に，スレッド毎の優先度に基づくラインマイグレーションを提案する．ラインのリプレース時に低優先度ラインが存在するバンクに対してマイグレーションし，高優先度スレッドが保持するラインをできるだけオンチップに残すことで性能低下を防ぐ．

以下に優先度に基づくマイグレーションの動作を示す．

（1） キャッシュミスが発生した場合，コア毎のリプレースメントポリシに従ってリプレースするラインを選択する．このとき shared 状態のラインは，他のバンクに存在する可能性があるため，マイグレーションは行わずに破棄する．

（2） 他のバンクの同一セットに属するラインと優先度を比較し，低優先度のラインが存在するか，もしくは invalid 状態のラインが存在するバンクに対してマイグレーションを行う．

（3） マイグレーション先が見つからなかった場合，必要であればメインメモリへの書き戻しを行う．

スレッドに与えられる優先度が動的に変化することに対応するため，ライン毎に優先度を保持することはしない．その代わりにコア毎の優先度を保持するテーブルを設け，コア ID から優先度を得るようにする．このテーブルのコストは極めて低いものであり，参照の際のレイテンシはほとんど無視できるものと考えられる．

4.1 優先度の比較方法

ラインのリプレース時に他のバンクの対応するラインと優先度を比較するには，優先度やタグメモリを集中管理する機構を用いる方法 [2] と，ブロードキャストによる方法 [3] が考えられる．集中管理型の機構はタグメモリのコストが高くなるため，本論文ではブロードキャストを用いる方法を用いることにする．ラインのリプレースが発生した場合，データと優先度情報を結合網を介してブロードキャストする．他のコアは自身の対応するラインの方が低ければそれを受け取ることができるが，複数のコアが受け取った場合には，同じラインがチップ上に複数存在することになり，キャッシュメモリの利用効率が低下してしまう．また受け取り側のコアではラインのリプレースが発生するため，結合網の混雑を招いてしまう．これらの理由からマイグレーション先は 1 つに選択するのが適切であると考えられる．そこでスヌープコレクタと呼ばれるマイグレーション先を 1 つに決定するための機構を設ける．各コアは優先度をスヌープコレクタに送信し，スヌープコレクタは最低優先度ラインが存在するバンクに対して受け取り許可を意味する信号を送る．この信号を受け取ったコアは，データをキャッシュメモリに書き込むことでマイグレーションを完了する．

5. dead block 予測機構

単純に優先度基づくラインマイグレーションを行った場合，高優先度スレッドの性能は保証できたとても，低優先度スレッドのキャッシュヒット率に悪影響を及ぼし，全体性能が低下してしまう可能性がある．そこで Hu らの提案した timekeeping metrics [6] を用いて再利用されないライン (dead block) を予測し，dead block であれば高優先度であってもマイグレーションせずにチップ外へ追い出すことで，キャッシュメモリの利用効率を向上させる．また低優先度であっても，dead block でなければ dead block が存在するバンクへマイグレーションを行う．優先度に基づくラインマイグレーションと dead block の予測により，高優先度スレッドの性能を保証しつつ全体の性能を向上させる．

5.1 Timekeeping Metrics

過去のメモリ参照のタイミング情報を利用する timekeeping technique の研究では，timekeeping metrics を用いることで，コンフリクトミスによって追い出されたラインのみをヴィクティムキャッシュに格納し，dead block をフィルタリングすることでヴィクティムキャッシュ内のラインの再利用性を高めることができる．提案手法ではこれを応用し，不要なラインマイグレーションをフィルタリングする．timekeeping metrics には以下に示す 4 つがある．また図 2 に，例としてライン A，B，A の順でキャッシュにロードされた場合の各 timekeeping metrics を示す．

- access interval: ラインがアクセスされる間隔
- live time: ラインが最初にアクセスされてから最後にアクセスされるまでの時間
- dead time: ラインが最後にアクセスされてから追い出されるまでの時間
- reload interval: ラインが追い出されてから次にリロードされるまでの時間

予備評価として，表 1 に示すパラメータを持つ 4 コア CMP 上で，MP3 エンコーダである lame を実行し二次キャッシュの dead time を計測した．なお access interval，live time，reload interval は今回は利用しないため省略する．図 3 に dead time の分布を示す．縦軸は分布率 (%)，横軸はサイクル数である．図 3 より，dead time は 100 万サイクル以下と 200 万サイクル以上に 2 極化していることがわかる．Hu らの議論より，100 万サイクル以下の分布はコンフリクトミスにより追い出されたラインであり，それ以上の分布はキャパシティミスにより追い出されたラインだと考えられる．dead block は基本的にキャパシティミスにより追い出されたラインであるため，最低でも 100 万サイクル以上に設定する必要があると考えられる．ただし値を大きくし過ぎると予測精度が低下してしまう．

5.2 dead time を用いた dead block の予測

本論文では dead time を利用した dead block の予測方法について述べる．提案手法ではライン毎にカウンタを設けること

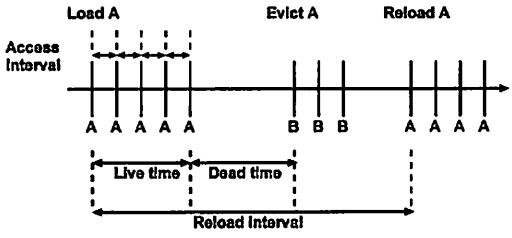

図 2 Timekeeping metrics

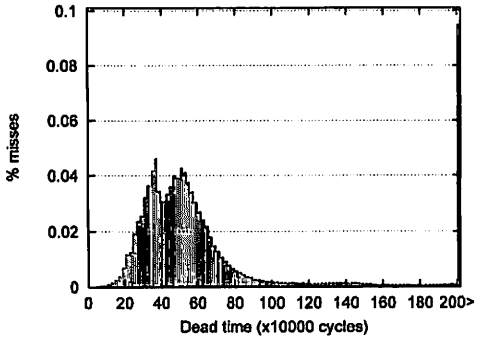

図 3 Dead time の分布

で dead time を計測する．このカウンタは時間の経過と共にインクリメントされ，対応するラインがアクセスされる度にクリアされる．ラインのリプレース時には対応するカウンタを読み，カウンタの値が dead time の閾値として設定された値 (dead time threshold) より大きい場合には，そのラインを dead block と予測し，高優先度であってもマイグレーションを行わない．しかしながら，このカウンタをシステムクロックを基準として実装した場合，カウンタのビット長が長くなり面積が増加してしまう．また，すべてのエントリを同時にインクリメントすることは SRAM では実装不可能であり，代わりに組み合わせ回路で実装しようとするとさらに面積が増加する．そこで文献 [7] で提案されているように，dead time threshold だけ時間が経過する度にカウンタをインクリメントし，カウンタが 2 回インクリメントされた場合に dead block と予測するようにする．この方法により少なくとも dead time threshold 以上の時間が経過したことが保証されるため，対応するラインが dead time だけ経過したと予測できる．この方法は dead time の計測粒度は粗くなるが，各ラインに 2 ビットカウンタを設けるだけで済むため実装面積を小さくできる．また各エントリを同時にインクリメントする必要がなくなるため，SRAM での実装も可能である．

2 ビットカウンタの状態を以下のように定める．

- START:初期状態であり，対応するラインが一度もヒットしていない状態
- COUNT1:対応するラインがヒットした状態
- COUNT2:dead time threshold が 1 回経過した状態
- DEAD:dead time threshold が 2 回経過し，dead block であると予測される状態

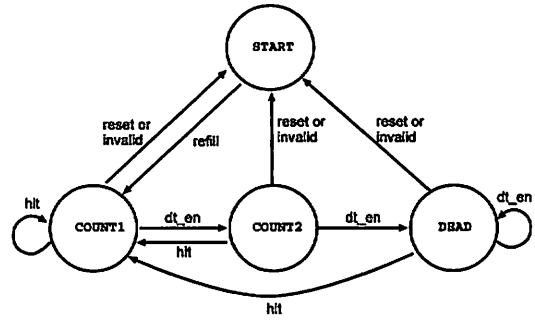

図 4 カウンタの状態遷移

図 4 にカウンタの状態遷移を示す．図 4 に示すように，カウンタは dead time threshold が経過した場合に有効になる信号 (dt_en) と，対応するラインへのアクセスにより状態遷移を行う．カウンタの初期状態は START 状態であり，最初のアクセスでラインがリフィルされると COUNT1 状態へ遷移する．COUNT1 状態で dt_en を計測すると COUNT2 状態へ遷移し，COUNT2 状態で dt_en を計測すると DEAD 状態に遷移する．DEAD 状態に遷移したということは dead time threshold 以上の時間，つまり dead time が経過したことを示しているので，対応するラインは dead block であると予測される．また各状態で対応するラインにヒットすると，その度に COUNT1 状態への遷移を繰り返し dead time の再計測を行う．

5.3 コア内のリプレースメントポリシ

ラインマイグレーションだけでなく，各バンクにおけるリプレースにおいても，優先度と dead block の予測を併用するすることで，さらに提案手法の効果を高めることができると考えられる．リプレースの対象となるラインの選択は

（1） dead block であるかどうか
（2） そのラインを持つスレッドの優先度
（3） そのラインの参照履歴

の順によって決定する．

5.4 共有キャッシュ構成に対する優位性

上述した dead block の予測方法は，一次キャッシュのヒット時に，同じラインが二次キャッシュにも存在するならば，正確な予測を行うために，二次キャッシュコントローラ上に存在するカウンタをリセットする必要がある．これは二次キャッシュコントローラが直接コアに接続されている NUCA だからこそ可能であるといえる．これを共有キャッシュ構成で実現しようとした場合，一次キャッシュのヒット時に，常に結合網を介して二次キャッシュコントローラにアクセスしなければならない．これは結合網の混雑を招いてしまい現実的ではない．専用パスを設けることも考えられるが，コア数が増えたときにカウンタを構成するメモリのポート数が増加し，ハードウェアコストが高価になってしまう．一次キャッシュと二次キャッシュ間を exclusive 構成にすればこの問題はなくなるが，一般的にマルチプロセッサにおいて exclusive 構成をとった場合，コヒーレン

表 1　メモリ階層のパラメータ

パラメータ	値
ラインサイズ	64B
一次キャッシュ(命令，データ)	16KB, 2way, 1 cycle
二次キャッシュ	256KB, 8way，10-24cycle
メインメモリ	150cycle

表 2　二次キャッシュのパラメータ

タグ	2cycle
データ	6cycle
結合網	5cyle

表 3　ベンチマーク

アプリケーション	内容
lame	MP3 エンコーダ
tiffmedian	メディアンカットアルゴリズム
typeset	汎用タイプセッティングツール

スプロトコルが複雑になってしまう．これらのことから提案手法は NUCA にこそ適した手法であるといえる．

6. 評　　価

6.1 評価方法

GEMS SLICC[8] により提案手法を実装し，フルシステムシミュレータ Simics [9] を用いて評価を行った．Simics により OS や各種デバイスを含めた詳細なシミュレーションが可能となる．提案手法に関しては，優先度のみを利用するもの，及び dead block の予測を併用してラインマイグレーションを行う手法を実装した．また比較対象として LRU 情報を用いてラインマイグレーションを行う手法も実装した．

6.1.1 構　　成

SPARC 命令セットを持つ CMP(4 コア) のシミュレーションを行った．各コアはインオーダ実行のプロセッサであり，クロスバスイッチにより相互接続されている．またメモリ階層のパラメータは表 1 のように設定した．二次キャッシュのアクセスレイテンシに関しては，表 2 に示すパラメータとアービトレーションの際のレイテンシを基に決定した．

6.1.2 ベンチマーク

評価には組み込みプロセッサ用ベンチマークである MiBench [10] を用いた．表 3 に使用したプログラムを示す．各プログラムは gcc3.4.6 を用いてコンパイルした．

6.2 シミュレーション結果

6.2.1 優先度特性の評価

始めに dead block の予測を行わずに優先度のみを用いた場合の，優先度特性に関して評価を行う．各コアで同じベンチマークを独立なスレッドとして実行させ，スレッド毎の IPC を計測した．同じベンチマークを実行した場合には，スレッド間の競合ミスが増加するため性能の低下が起こる最悪のケースであると考えられる．また優先度を用いる手法ではコア 0 のスレッドの優先度を最も高くし，段階的に優先度が低くなるように設定した．

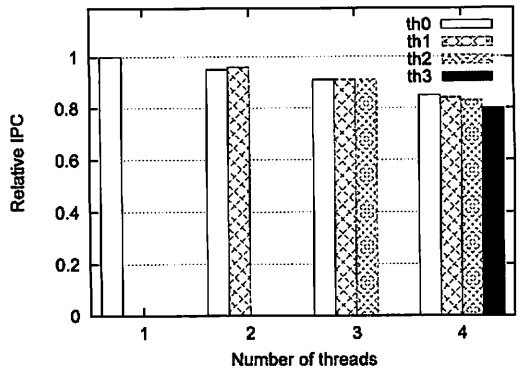

図 5　各スレッドの相対 IPC (LRU)

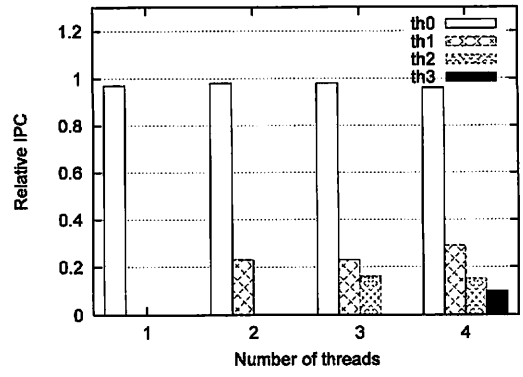

図 6　各スレッドの相対 IPC (優先度付き)

図 5 に LRU 方式，図 6 に優先度を用いる手法で各ベンチマークを実行したときのコア毎の平均 IPC を示す．IPC は LRU 方式で 1 スレッド実行したときもので正規化した．図 5 より，LRU 方式では全体の IPC は高いが，スレッド数の増加に伴いキャッシュメモリの競合が激しくなり各スレッドの IPC が低下していることがわかる．特に 4 スレッド同時実行の場合には IPC が 15%から 20%低下した．このため複数のスレッドを同時に実行した場合，高優先度スレッドの性能が低下しデッドラインミスを起こす可能性がある．

一方で，優先度を用いる手法では最高優先度スレッドの IPC の低下は同時実行スレッド数によらず 1%に抑えられており，最高優先度スレッドの性能を維持できているといえる．しかしながら，優先度が 2 番目以降のスレッドはキャッシュメモリを十分に与えられないため，4 スレッド同時実行の場合には IPC が 74%から 90%と大きく低下してしまった．特に最低優先度スレッドは自身のバンクのほとんどを他のスレッドに利用されてしまうため，大きく性能が低下していることがわかる．このように優先度のみの制御では，全体性能が大きく低下してしまうため，データ量の多いソフトリアルタイムタスクが複数あるような場合には向いているとはいいがたい．

6.2.2 dead block 予測機構の評価

dead block の予測を行った場合の最高優先度スレッドの性能

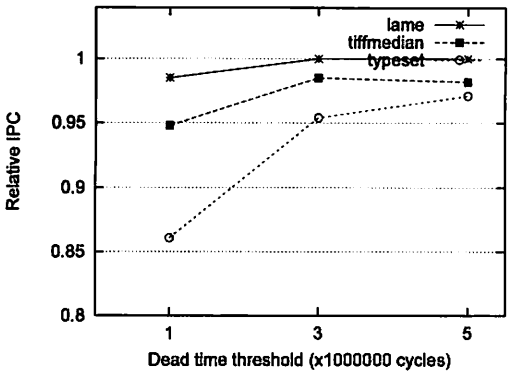

図 7 dead time threshold と最高優先度スレッドの IPC

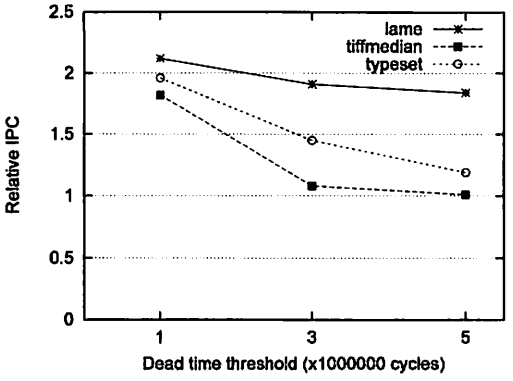

図 8 dead time threshold と全体の IPC

低下の割合と，優先度のみ用いる手法と比べた場合の全体性能の増加割合を評価する．図 7 に，dead time threshold を変化させた場合の最高優先度スレッドの IPC を，図 8 に全体の IPC を示す．IPC は優先度のみの手法のもので正規化した．

図 7，8 より threshold が 300 万サイクルのときにはすべてのベンチマークで最高優先度スレッドの IPC の低下を 5%未満に抑えつつ，優先度のみ用いる手法に比べて全体の IPC を 8%から 91%向上していることがわかる．つまり最高優先度スレッドの性能を保証しつつ，全体性能を向上することができているといえる．しかしながら，提案手法では threshold を大きく設定すると dead block と予測されるラインが少なくなるため最高優先度スレッドの性能低下は抑えられるが，不要なマイグレーションが増加するため全体性能は低下してしまう．逆に threshold を小さく設定すると全体性能は向上するが，最高優先度スレッドの性能が保証できなくなる．つまり提案手法は事前に最適な threshold を求めておく必要があるが，メモリ階層の構成や同時に実行するスレッド等によって最適な値は変化する可能性があるため，静的に求めるのは困難になる．このためソフトウェアやハードウェアの制御により動的に値を求める機構が必要になると考えられる．

7. 結　　論

本論文では CMP の NUCA において，高優先度スレッドの性能を保証しつつ全体の性能を向上させるためのラインマイグレーションを提案した．シミュレーションによる評価によると，従来の LRU 手法では 4 スレッド同時に実行した場合，各スレッドの IPC が 15%から 20%低下したのに対し，優先度を用いる手法では最高優先度スレッドの性能低下を 1%に抑えることができた．また dead block を予測することで最高優先度スレッドの性能低下を 5%未満に抑えつつ，優先度のみの手法に比べて全体性能を 8%から 91%向上させることができた．しかしながら，提案手法は dead time threshold の設定値によって，性能が大きく変化することがわかったため，今後の課題としてソフトウェアやハードウェアの制御により動的に最適な値を求める機構が必要である．

謝辞 本研究は独立行政法人新エネルギー・産業技術総合開発機構 (NEDO) の支援による．

文　　献

[1] C. Kim, D. Burger and S. Keckler: "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches", In Proc of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 211–222 (2002).

[2] J. Change and G. Sohi: "Cooperative Caching for Chip Multiprocessors", In Proc of the 33rd Annual International Symposium on Computer Architecture, pp. 264–276 (2006).

[3] E. Speight, H. Shafi, L. Zhang and R. Rajamony: "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors", In Proc of the 32nd Annual International Symposium on Computer Architecture, pp. 346–356 (2005).

[4] 塩谷, ルォン ディン, 入江, 五島, 坂井："マルチコア・プロセッサの不均質共有キャッシュにおける LRU 大域置き換えアルゴリズム", 情報処理学会論文誌コンピューティングシステム, **48**, SIG 3, pp. 59–74 (2007 年).

[5] 佐藤, 内山, 伊藤, 山崎, 安西："リアルタイム処理用マルチスレッディングプロセッサの優先度に基づくキャッシュサブシステム", 情報処理学会研究会 計算機アーキテクチャ研究会, pp. 37–42 (2001 年).

[6] Z. Hu, S. Kaxiras and M. Martonosi: "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior", In Proc of the 29th Annual International Symposium on Computer Architecture, pp. 209–220 (2002).

[7] 新井："Responsive Multithreaded Processor のキャッシュシステムにおける低消費電力化機構", Master's Thesis, 慶應義塾大学大学院理工学研究科 (2006 年).

[8] M. M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill and D. A. Wood: "Multifacets General Execution-Driven Multiprocessor Simulator (GEMS) Toolset", Computer Architecture News (CAN), **33**, 4, pp. 92–99 (2005).

[9] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt and B. Werner: "Simics: A Full System Simulation Platform", IEEE Computer, **35**, 2, pp. 50–58 (2002).

[10] M. Guthaus, J. Ringenberg, D.Ernst, T. Austin, T. Mudge and R. Brown: "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", IEEE 4th Annual Workshop on Workload Characterization (2001).