Vol.2015-ARC-214 No.7
2015/1/30

TLBミスペナルティ削減のための大容量LLCの利用法に関する初期検討

有間 英志[1,a)]　三輪 忍[1]　中田 尚[1]　中村 宏[1]

概要：近年，不揮発性メモリや3次元積層技術等デバイス技術の進歩によって，これまで以上に大容量のメモリをオンチップに実装することが可能となりつつある．また，この様な大容量メモリをラスト・レベル・キャッシュ (LLC) として用いる利用法が提案され，大幅な性能向上が可能であることが示されてきた．しかし，これまでの大容量 LLC に関する先行研究では，TLB ミスペナルティの影響については，十分な考慮がなされてこなかった．LLC の大容量化に伴い，LLC 上に格納されたデータの内，当該ページアドレスが TLB 上に存在しないものの割合は増大する．その様なデータがアクセスされると TLB ミスが発生し，キャッシュもしくはメインメモリ上に存在する当該ページテーブルエントリへのアクセスが発生する．この TLB ミスペナルティの影響を削減することは，今後 LLC の大容量化がさらに進むにつれて極めて重要となる．そこで本研究では，大容量 LLC 上において，ページテーブルエントリを保持するラインの存在割合を最適化し，ページテーブルへのアクセスの殆どを LLC 上でヒットさせることによって，TLB ミスペナルティの削減を目指す．本稿では，これを行うためのキャッシュリプレイスメントアルゴリズムを検討し評価を行った．

1. はじめに

これまでの商用 CPU においては，メモリウォール問題 [1] の解消のため，オンチップキャッシュの大容量化が推し進められてきた．これによって，アクセスレイテンシの長いメインメモリへのアクセスが隠蔽され，CPU の大幅な性能向上が達成されてきた．キャッシュ，特にラスト・レベル・キャッシュ (LLC) の大容量化の要求は，ワーキングセットサイズやコア数の増大に伴って，今後も継続されると考えられている．

近年，不揮発性メモリや3次元積層技術等デバイス技術の進歩により，従来よりも格段に大容量のメモリを LLC として利用することが可能となりつつあり，その有効性が示されている．文献 [2], [3], [4], [5] では，STT-MRAM や PRAM といった従来の SRAM よりも高集積度かつ低リーク電力の不揮発性メモリを LLC に適用することで，CPU のさらなる性能の向上と低消費電力化が可能であることが示されている．また，文献 [6], [7] では，数十 MB〜1GB 程度の DRAM チップを CPU チップ上に積層し，これを大容量 LLC として利用することで大幅な性能の向上が可能であることが示されている．

しかし，これまでの大容量 LLC に関する先行研究では，TLB ミスペナルティの影響については，十分に考慮されてこなかった．TLB サイズの大幅な増加は今後期待できないとされている．これは，TLB はコンテクストスイッチの度にフラッシュする必要があり，巨大な TLB を用意してもそれを実行中のプロセスが使い切ることが難しいためである．そのため，LLC の大容量化が進むにつれて，LLC 上のデータで，当該ページアドレスが TLB 上に存在しないものの割合は増大する．その様なデータがアクセスされることで TLB ミスが起き，キャッシュもしくはメインメモリ上に存在する，当該ページテーブルエントリへのアクセスが発生する．この TLB ミスペナルティの影響は，当該ページテーブルエントリがキャッシュ上に存在しない場合には，特に重大となる．文献 [8] によれば，現行の CPU を利用した場合でも，実行時間の 40%以上を TLB ミスペナルティが占める様なアプリケーションも存在する．

TLB ミス自体は，スーパーページと呼ばれる 2MB, 1GB 等の巨大サイズのページを利用することで削減できる [9]．これは，TLB の 1 エントリが管理できるメモリの容量が増えることによって，TLB ミスの頻度を減らすことができるためである．しかし，この様なページを利用した場合には，メインメモリ上においてフラグメンテーションが起き易くなるという問題がある．また，OS は一般的にハードコーディングされており，従来の 4KB のページ向けに最

[1] 東京大学大学院
[a)] arima@hal.ipc.i.u-tokyo.ac.jp

適化されているため，スーパーページ向けの最適化が困難であるという側面も存在する．

そこで本研究では，大容量 LLC を利用した場合に問題となる，TLB ミスペナルティを削減するための，LLC の利用法に関する検討を行う．具体的には，大容量 LLC 上においてページテーブルエントリを保持するラインの存在割合を最適化し，ページテーブルへのアクセスの殆どを LLC 上でヒットさせることにより，TLB ミスペナルティの削減を目指す．本稿では，これを行うためのキャッシュリプレイスメントポリシーを検討し，シミュレーションによって評価を行った．

本論文の構成は以下の様になる．まず，2 章にて本研究の背景と動機について述べる．次に 3 章において本研究の提案を述べる．4 章では評価環境及び評価結果を述べる．5 章では関連研究について述べる．最後，6 章では本稿のまとめを述べる．

2. 背景と研究動機

本章では，本研究の背景と動機付けについて述べる．具体的には，2.1 にて本研究が想定するハードウェア構成について述べ，2.2 にてデバイス技術の進歩による LLC の大容量化と，それに伴う TLB ミスペナルティの影響の増大について述べ，2.3 にて TLB ミスペナルティに関する分析を行う．

2.1 想定するハードウェア構成

図 1 に本研究で想定する，CPU のハードウェア構成を示す．CPU は複数コアからなり，これらは LLC を共有している．これは，現行の CPU と同等の構成であり，LLC を積層メモリチップや不揮発性メモリ等によって構成することで，数十 MB を超える様な LLC を搭載することができる．

TLB ミス時のページテーブルウォークは，x86 や ARM と同様でハードウェアによって制御されるものと仮定する．具体的には各コアごとに存在する MMU(Memory Management Unit) によってページテーブルウォークは制御され，当該ページテーブルエントリを LLC もしくはメインメモリから取得する．取得したページテーブルエントリは MMU 内に存在するキャッシュ (MMUC) に格納される．また，MMU は取得したページテーブルエントリの情報を使って仮想/物理アドレスの対応を計算し，TLB 上にこれを格納する．

2.2 LLC の大容量化と TLB ミスペナルティの増大

ワーキングセットサイズやコア数の増大に伴い，LLC の大容量化の要求は今後も継続すると考えられている．特に，コア数の増大に伴って，メインメモリのバンド幅不足が指摘されており，レイテンシの削減だけでなく，バンド

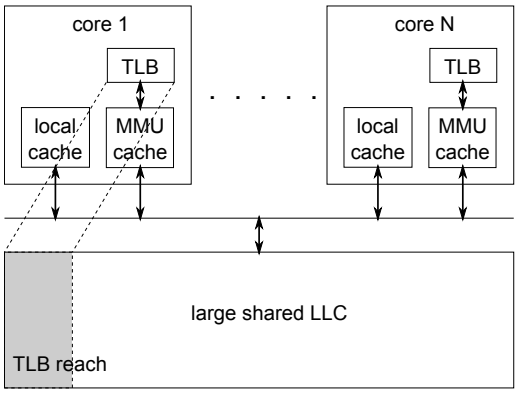

図 1 想定するハードウェア構成

幅消費の削減という観点からも，さらなる大容量の LLC が必要となる [10]．そこで，CPU チップ上にメモリチップを積層する 3 次元積層技術が有力視されており，また，STT-MRAM や PRAM といった不揮発性メモリを利用することによる LLC の大容量化も期待されている．

しかし，LLC の大容量化が進むにつれて，TLB ミスペナルティの影響が大きくなるという問題がある．これは，図 1 に示す様な TLB reach の範囲が相対的に小さくなるためである．ここで TLB reach とは，TLB 上の全てのエントリによってカバーできる物理アドレスの範囲を指し，この範囲にあるデータであれば，TLB ミスなしにアクセスすることが可能である．TLB サイズは LLC サイズと比較して緩やかに増加してきたことが指摘されている [9]．これは TLB はコンテクストスイッチの度にフラッシュする必要があり，巨大な TLB を用意してもそれを実行中のプロセスが使い切ることが難しいためである．そのため，TLB サイズは殆ど固定のまま，LLC サイズのみが増大することになる．その結果相対的に TLB reach の範囲が小さくなり，LCC 上で TLB ミスなしにアクセスできるデータの割合は少なくなる．従って，大容量 LLC において TLB ミスペナルティを削減することは重要である．

2.3 LLC 上のデータ配置と TLB ミスペナルティ

必要なページテーブルエントリをより上位の階層に配置することで，TLB ミスペナルティを削減することができる．特に，メインメモリアクセスのオーバーヘッドは大きいため，LLC におけるページテーブルエントリのヒット率を向上させることは重要である．

図 2 に，各種データごとに LLC 上でのヒット率を分類したものを示す．ただし，横軸は実行したベンチマークプログラムを示しており，縦軸は各種データごとの LLC 上でのヒット率を示す．page table はページテーブルエントリを表し，inst. はプログラムの命令を表し，data はプログラムの扱うデータを表す．評価環境は 4 章で示すとおりである．

LLC 上でのページテーブルエントリへのアクセスの hit

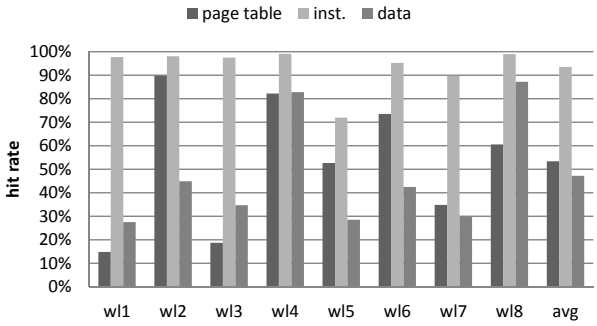

図 2 各データの LLC 上でのヒット率

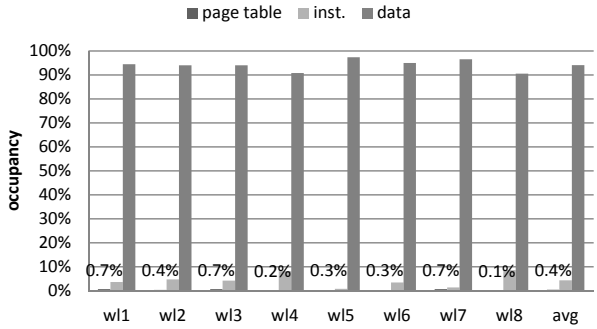

図 3 各データの LLC 上の占有率

率は平均で 53.4%であり，最悪の場合には 14.9%にまで低下する．この様に，ページテーブルエントリへのアクセスは LLC においてミスを起こしやすい．すなわち，TLB ミスペナルティを削減できる余地は大きい．

一方で，ページテーブルエントリを保持するラインが LLC 上で占める容量は小さい．**図 3** に，データの種類ごとの LLC 上での占有率を示す．ただし，横軸は実行したベンチマークを表し，縦軸は LLC 上での各種データの占有率を表す．page table はページテーブルエントリを表し，inst. はプログラムの命令を表し，data はプログラムの扱うデータを表す．評価環境は 4 章で示すとおりである．

図 3 に示す様に，LLC 上においてページテーブルエントリを保持するラインの占有率は平均で 0.412%であり，最低では 0.110%となっている．そのため，より多くのページテーブルエントリを LLC に割り当て，TLB ミスペナルティを削減するという方法が有効であると言える．

3. TLB ミスペナルティ削減のための LLC のデータ配置最適化

本章では，まず 3.1 にて本研究が扱う問題の定義について述べ，次に 3.2 にてそれを解決するためのキャッシュリプレイスメントポリシーについて述べる．

3.1 問題設定

本研究では，大容量 LLC 上においてページテーブルエントリを保持するラインの存在割合を最適化することによって，TLB ミスペナルティを削減し性能向上を目指す．本節では，この制御を適切に行うための問題の定式化について述べる．

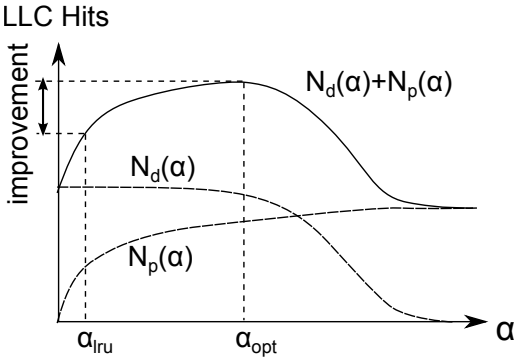

図 4 データ配置最適化の概要

本研究で扱う具体的な目的関数を式 (1) に示す．ただし，α, $N_p(\alpha)$, $N_d(\alpha)$ はそれぞれ，LLC におけるページテーブルエントリを保持するラインの存在割合，ページテーブルエントリの LLC におけるヒット回数，ページテーブルエントリ以外の通常のデータの LLC におけるヒット回数を表わす．すなわち，この目的関数は，これら 2 種類のデータを包括する LLC 上の全てのデータのヒット回数の最大化を目的としている．これによって，通常のデータのヒット率低下の抑制と，TLB ミスペナルティの削減の両立が可能となる．

$$\max(N_d(\alpha) + N_p(\alpha)) \tag{1}$$

この最適化の様子を**図 4** のグラフに示す．ただし図 4 では，横軸はページテーブルエントリを保持するラインの存在割合 α を示し，縦軸は LLC におけるヒット回数を表わす．ページテーブルエントリの LLC ヒット回数 $N_p(\alpha)$ は α の増加関数となり，それ以外の通常のデータの LLC ヒット回数 $N_d(\alpha)$ は α の減少関数となる．本研究では，これらの総和を最大化する $\alpha = \alpha_{opt}$ を求め，α をこの値に近づける様に LLC のデータ配置の制御を行う．特にリプレイスメントアルゴリズムが一般的な LRU である場合 (図 4 中 $\alpha = \alpha_{lru}$) からの改善を図る．

図 5 に LLC におけるデータ配置最適化の様子を示す．ただし，図中 Proc. *で記された領域は，プロセス*に属する LLC 上のデータを便宜的にまとめて表わしたものであり，その中でページテーブルエントリを page で，それ以外を the others で表わしている．図 5 に示す様に，ページテーブルエントリに対して，余分に多くの領域を割り当てておくことで，実行状態のプロセスだけでなく，実行可能状態や割り込み待ち状態のプロセスに属するページテーブルの多くを LLC 上に残すことができるという効果も期待できる．それによって，これらのプロセスがコンテクストスイッチ時に実行状態に遷移した場合，直後に頻発する

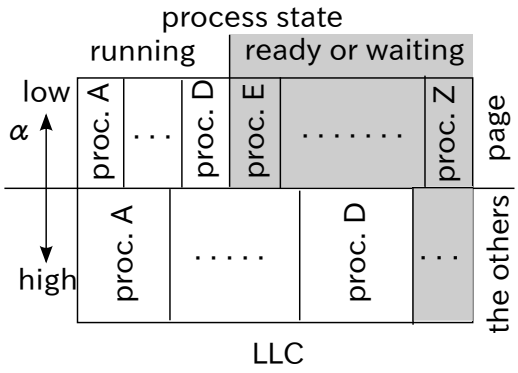

図 5 データ配置最適化の模式図

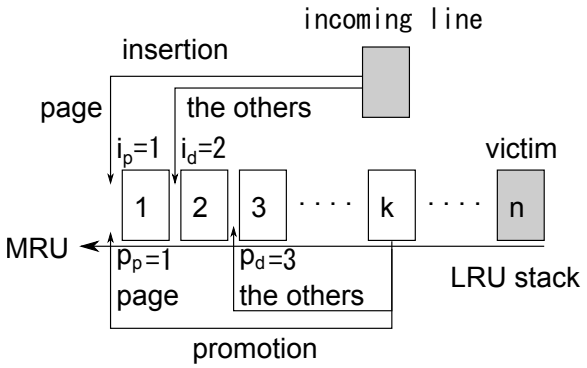

図 6 検討したリプレイスメントアルゴリズム

ページテーブルアクセスのほとんどを，LLC 上でヒットさせることができる．また，LLC が十分に大きければ，稼動プロセスのページテーブルを全て LLC に格納することができ，それによって，ページテーブルアクセスを 100%LLC 上でヒットさせることができる．

3.2 データ配置最適化のためのリプレイスメントアルゴリズム

ページテーブルエントリを保持するラインの割合を最適化するため，本稿では，リプレイスメントアルゴリズムの最適化を行う．**図 6** に本稿で検討するリプレイスメントポリシーを示す．これは一般的な LRU に変更を加えたものであり，あるラインの LLC への挿入直後 (図中 insertion) 及びアクセス直後 (図中 promotion) の LRU ポジションを，そのラインがページテーブルエントリを保持しているかどうかで変更するというものである．図ではページテーブルエントリを保持するラインの挿入直後，アクセス直後の LRU ポジションをそれぞれ i_p, p_p とし，それ以外の通常のラインについては挿入直後，アクセス直後の LRU ポジションを i_d，p_d としている．挿入時に LRU ポジションが最下位のもの，すなわちラインに割り振られた番号が最大のものが victim となる．LRU($i_p = p_p = i_d = p_d = 1$) と比較した場合に，ページテーブルエントリを保持するラインの存在割合 α を高くすべきと判断されれば，$i_p \leq i_d, p_p \leq p_d$ となるようにリプレイスメントポリシーを設定することでこれを達成でき，その逆であれば $i_p \geq i_d, p_p \geq p_d$ と設定することでこれを達成できる．これらの値を上手く設定することで α を最適値 α_{opt} に近づけることができる．

4. 評価

本研究が取り組む問題及び検討した手法に関する予備的な評価を行った．以下 4.1 にて評価環境の詳細について述べ，4.2 にて評価結果について述べる．

4.1 評価環境

提案手法の効果を評価するため，フルシステムシミュレータ Gem5[11] を用いたシミュレーションを行った．具体的には，OS 及び複数のアプリケーションが動作するシステムの機能シミュレーションを行いつつ，同時にそこで得られたメモリアクセスシーケンスを用いて，キャッシュシミュレーションを行った．本シミュレーションにおいては，メインメモリ上のデータのマッピングはシミュレーション中に動作する OS が管理しており，TLB 及びページテーブルへのアクセスはシミュレータが管理している．

シミュレーションで用いたシステム構成を表 1 に示す．2 階層キャッシュを想定し L2 キャッシュは共有キャッシュを想定している．L1 キャッシュ，TLB，MMU キャッシュはコアごとに固有に存在しており，これらはいずれもデータ，命令で分離されている．本稿では，簡単のため LLC のサイズは現行の CPU と同等の 2MB を仮定した．今後の研究では，より大きなサイズの LLC に対して同等の評価を行う予定である．

ベンチマークプログラムは SPEC CPU 2006 [12] からいくつか選定し，重複を含む 8 つのプログラムを同時に稼働させて評価を行った．これは，前述のコンテクストスイッチ時の影響も含んだ評価を行うためである．ただし，シミュレータ上で動作する OS がコアへのスレッド割り当てを管理している．同時に稼働させたベンチマークプログラムの組み合わせは 2 の通りである．ただし，`ast`，`cac`，`omn`，`gem`，`xla`，`sop` はそれぞれ `astar`，`cactusADM`，`omnetpp`，`gemsFDTD`，`xalancbmk`，`soplex` を表す．

表 1 システム構成

Name	Remarks
OS	Linux 2.6.28, 4KB page
CPU	2 core, x86
L1 D/I cache	32KB, 4-way set assoc., 64B line
D/I TLB	64 entry
D/I MMUC	64KB, 4-way set assoc.
L2 (LLC)	2MB, 8-way set assoc., 64B line

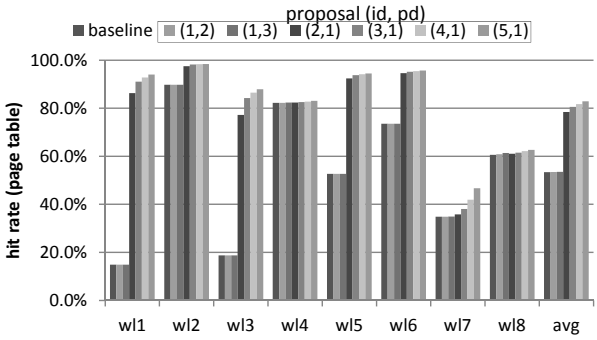

図 7 LLC におけるページテーブルデータのヒット率

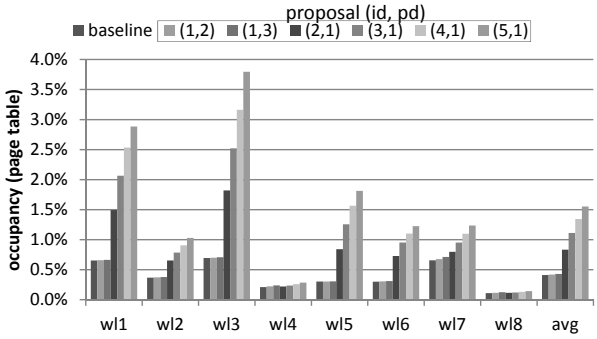

図 8 LLC におけるページテーブルデータの占有率

4.2 評価結果

図 7，**図 8** に，LLC におけるページテーブルエントリを保持するラインのヒット率，占有率をそれぞれ示す．ただし，いずれのグラフも横軸は実行したベンチマークプログラムを示している．縦軸については図 7 の場合は LLC 上でのページテーブルを保持するラインのヒット率を示しており，図 8 では LLC 上でのページテーブルエントリの占有率を表している．baseline は提案手法を適応せず，リプレイスメントポリシーを LRU とした場合を表している．一方で proposal は提案手法を適応した場合を示しており，各々の凡例は $i_p = p_p = 1$ として，(i_d, p_d) を変化させた場合を表している．

$i_d = 1$ で固定のまま p_d に 1 以外の値に割り振った場合，ページテーブルエントリの占有率が増えておらず，ページテーブルエントリを保持するラインのヒット率も向上していないことが分かる．p_d に高い値を割り振ることで，ページテーブル以外の通常のデータが，再利用されても LLC 上に残りにくくなる．一方でページテーブルエントリを保持するラインが再利用された場合には，MRU に配置されるので追い出されにくくなる．しかし，ページテーブルエントリを保持するラインの再利用間隔が長い場合には，競合によって LLC から追い出され，ヒット率は改善しない．今回の評価でははページテーブルエントリを保持するラインの再利用間隔が十分に長かったものと考えられる．

表 2 実行したベンチマークプログラムの組み合わせ

Name	Combination
WL1	ast,ast,cac,cac,omn,omn,gem,gem
WL2	mcf,cac,ast,omn,omn,gem,xla,sop
WL3	mcf,mcf,cac,cac,ast,ast,omn,omn
WL4	mcf,mcf,gem,gem,omn,omn,xla,xla
WL5	mcf,mcf,mcf,mcf,cac,cac,ast,ast
WL6	mcf,mcf,mcf,mcf,cac,ast,xla,sop
WL7	mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
WL8	omn,omn,gem,gem,xla,xla,sop,sop

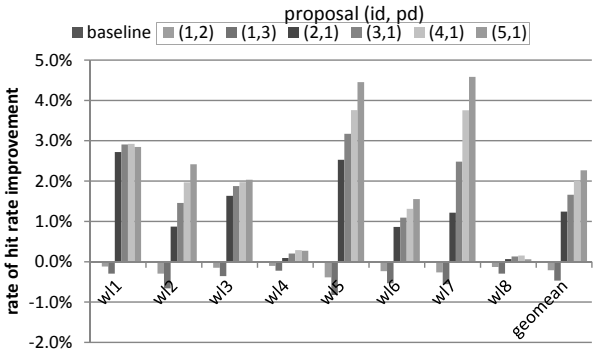

図 9 LLC ヒット率の向上率

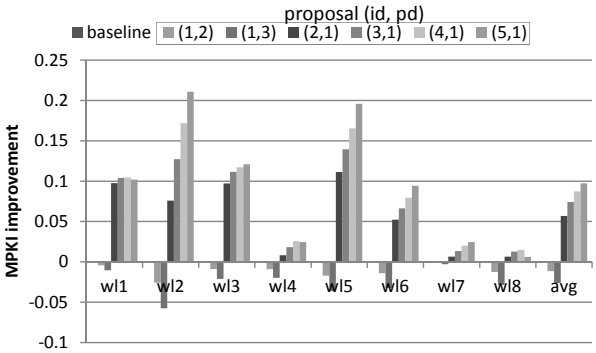

図 10 提案手法による MPKI の削減量

一方で $p_d = 1$ で固定のまま i_d に高い値を割り与えれば割り与えるほど，ページテーブルエントリの占有率が増え，ページテーブルエントリを保持するラインのヒット率も向上することが分かる．例えば $i_d = 5$ とした場合，最大で 42%，平均で 29%ページテーブルエントリのヒット率が向上している．これは，ページテーブル以外の通常のデータについては，データ配置時の LRU ポジションが下がることで，再利用率の低いものが追い出されやすくなり，一方で，ページテーブルエントリはデータ配置時に MRU に配置されるので，追い出されにくいためである．

次に，**図 9**，**図 10** に，LLC ヒット率の向上率，LLC における MPKI の削減分をそれぞれ示す．ただし，いずれのグラフも横軸は実行したベンチマークプログラムを示している．縦軸については図 7 の場合は LLC ヒット率の向上率を示しており baseline で正規化されている．図 8 では縦軸は LLC における MPKI の削減分を表している．凡例は前述のとおりである．$i_d = 5, p_d = 1$ の場合が最適であり，最大で 4.6%のヒット率向上を達成し，MPKI を 0.22 削減

できている．これは，$i_d = 5, p_d = 1$ という設定では，ページテーブルエントリを保持するラインは LLC 上に残りやすく，それ以外の通常のラインは再利用率の高いもののみ LLC に残りやすくする様な制御ができているためである．

5. 関連研究

TLB ミス自体や TLB ミスペナルティの削減に関する研究はこれまでにも行われている．本章では，これらの関連研究について整理する．

文献 [9] では，TLB ミス自体を削減する目的で，スーパーページと呼ばれる 2MB，1GB 等の巨大サイズのページを利用する方式が提案されている．これによって，TLB の 1 エントリが管理できるメモリの容量が増え，TLB ミスの頻度を減らすことができる．しかし，これらのページを利用した場合には，メインメモリ上においてフラグメンテーションが起き易くなるという問題がある．また，OS は一般的にハードコーディングされており，従来の 4KB のページ向けに最適化されているため，スーパーページ向けの最適化が困難である．

ページサイズを従来の 4KB とし，TLB のヒット率を向上させる研究も存在する．文献 [14] では，並列アプリケーション実行時に，全コア間で共有された TLB ミスの履歴を利用して，各々のコアに対して TLB エントリのプリフェッチを行うという方式が提案されている．同様に文献 [15] では，各コア間で共有する L2TLB を導入することで，並列アプリケーション実行時において大幅に TLB ヒット率が向上することが示されている．ただし，これらは並列アプリケーションが動作する場合のみに有効な手法である．

TLB ミスの削減ではなく，本研究と同様に TLB ミス時のペナルティを削減する研究も行われている．文献 [8] では，MMUC のヒット率を向上させる技術を提案しており，また MMUC を全コアで共有することで，同一面積下において，ヒット率が向上することが示されている．

6. まとめ

本稿では，今後 LLC の大容量化が進むにつれてますます重要となる，TLB ミスペナルティの削減を目指した．具体的には LLC 上でのページテーブルエントリを保持するラインの存在割合を最適化することで，この TLB ミスペナルティを削除する方式を提案した．さらに，存在割合を制御するためのリプレイスメントポリシーを検討し，シミュレーションによる評価を行った．

今後は，提案手法のさらなる改良及び，より現実的な環境における評価を行う予定である．提案手法については，insertion 及び promotion の最適位置を動的に求めるための方法について検討する予定である．評価に関しては，サイクルアキュレートなシミュレーションを実施することで，提案手法の効果をより正確に求める予定である．また，より大容量の LLC での評価や，データセンターで用いられるアプリケーションを用いた評価等も予定している．

謝辞

本研究の一部は,NEDO「ノーマリーオフコンピューティング基盤技術開発」事業及び JSPS 科研費「25540018」の助成による．

参考文献

[1] Wulf, W. A. and McKee, S. A.: Hitting the Memory Wall: Implications of the Obvious, *SIGARCH Comp. Arch. News*, Vol. 23, No. 1, pp. 20–24 (1995).
[2] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R. and Xie, Y.: Hybrid Cache Architecture with Disparate Memory Technologies, *ISCA*, pp. 34–45 (2009).
[3] Zhang, W. and Li, T.: Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures, *PACT*, pp. 101–112 (2009).
[4] Arima, E., Noguchi, H., Nakada, T., Miwa, S., Takeda, S., Fujita, S. and Nakamura, H.: Fine-Grain Power-Gating on STT-MRAM Peripheral Circuits with Locality-aware Access Control, *The Memory Forum* (2014).
[5] 有間英志, 薦田登志矢, 中田　尚, 三輪　忍, 野口紘希, 野村久美子, 安部恵子, 藤田　忍, 中村　宏：低 CPU 負荷を考慮した STT-MRAM ラスト・レベル・キャッシュの要求性能の解析, 電子情報通信学会論文誌 A, Vol. J97-A, No. 10, pp. 629–647 (2014).
[6] Loh, G. H.: 3D-Stacked Memory Architectures for Multi-core Processors, *ISCA*, pp. 453–464 (2008).
[7] Qureshi, M. K. and Loh, G. H.: Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design, *MICRO*, pp. 235–246 (2012).
[8] Bhattacharjee, A.: Large-reach Memory Management Unit Caches, *MICRO*, pp. 383–394 (2013).
[9] Talluri, M. and Hill, M. D.: Surpassing the TLB Performance of Superpages with Less Operating System Support, *ASPLOS*, New York, NY, USA, pp. 171–182 (1994).
[10] Rogers, B. M., Krishna, A., Bell, G. B., Vu, K., Jiang, X. and Solihin, Y.: Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling, *ISCA*, pp. 371–382 (2009).
[11] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Computer Architure News*, Vol. 39, No. 2, pp. 1–7 (2011).
[12] Henning, J. L.: SPEC CPU2006 Benchmark Descriptions, *SIGARCH Computer Architure News*, Vol. 34, No. 4, pp. 1–17 (2006).
[13] Bienia, C.: Benchmarking Modern Multiprocessors, PhD Thesis, Princeton University (2011).
[14] Bhattacharjee, A. and Martonosi, M.: Inter-core Cooperative TLB for Chip Multiprocessors, *ASPLOS*, pp. 359–370 (2010).
[15] Bhattacharjee, A., Lustig, D. and Martonosi, M.: Shared last-level TLBs for chip multiprocessors, *HPCA*, pp. 62–63 (2011).