C-003

128 コア CPU エミュレータによる共有キャッシュの挙動について
A study on behavior of shared cache memory using 128 core CPU emulator

今井 謙太郎† 岡本 秀輔†
Kentaro Imai Shusuke Okamoto

1. はじめに

現在の PC の多くには、マルチコアと呼ばれる CPU が組み込まれている。参考文献[1]によると、現時点ではスーパーコンピュータなどの最先端のもので 50 コア程度の CPU まで使用されており、自動車などの組込み型 CPU では 2010 年までにコアの数が 100 超になると予想されている。

PC の方でもこのコアの数は年々増加しており、参考文献[2]によると近年 Intel 社が 80 個のコアを内蔵した CPU をデモするなど、その数が 100 超になるまでは時間の問題というところまで来ている。従って、数年後にはコアの数が 100 超である CPU が市場に多く出回ると予想される。

現在のマルチコア CPU は、コアの数が少ないものでも共有キャッシュを使用している。コアの数が増加するに伴い、1 つの共有キャッシュへ接続されるコアの数は増加し、CPU の性能に対する共有キャッシュの重要度はより増すと考えられる。

そこで本研究では、コアの数が 128 個の場合に共有キャッシュではどのようなことが起こるのかを、ラインの置き換えという観点に着目した。これを調べることで、コア数が 100 超のマルチコア CPU における共有キャッシュの位置づけが見えてくると思われる。

マルチコア CPU の共有キャッシュを調べるにあたって、まず高級言語にてマルチコア CPU エミュレータを実装した(仕様については次章以降で説明をする)。その後、C 言語にて作成した評価用プログラムを実装したエミュレータ上で実行し、L2、L3 の各共有キャッシュでのラインの置き換え回数や L1 でのラインの無効化回数を調べ、その結果に対して考察をしていく。

2. 命令セット

命令セットは MIPS アーキテクチャのものを採用した。その理由としては、現在の世の中で広く使用されている命令セットの 1 つであるということ。さらに実装にあたって資料やコンパイラが多く存在するためである。

エンディアンはリトルエンディアンを使用し、命令長は 32bit となっている。また、処理は整数演算のみに対応している。

3. CPU エミュレータの仕様

図 1 は本研究で実装したエミュレータ設計図である。図の各数字は各レベルのキャッシュ番号を表し、各線は各キャッシュを接続するバスを表している。また L1 はデータキャッシュと命令キャッシュの 2 種類がある。

L2 共有キャッシュは、コア 4 つで 1 つ使用する。さらに L3 共有キャッシュは、L2 共有キャッシュ 4 つにつき 1 つ、つまりコア 16 個ごとで 1 つの L3 共有キャッシュを使用することになる。また、各レベルのキャッシュはバスによって接続されており、接続されたキャッシュ同士でデータの転送が行えるようになっている。

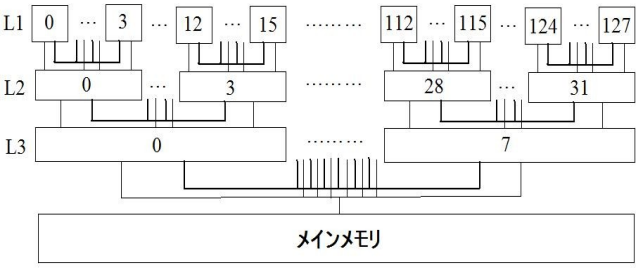

図 1 CPU エミュレータの概略図

4. キャッシュエミュレータの仕様

まずキャッシュ容量について。L1 ではデータキャッシュ、命令キャッシュ共に 2KB となっている。L2 共有キャッシュでは 512KB、L3 共有キャッシュでは 2MB といった容量にそれぞれ設定した。

マッピング方式は L1、L2、L3 の各キャッシュともにセット連想マッピングを採用しており、そのセット数は 4way と設定した。またラインサイズは 4word つまり 16Byte と設定している。

キャッシュには図 2(A,B はキャッシュラインを表している)のような下位のレベルのキャッシュが上位のレベルのキャッシュのラインを必ず保持している inclusive キャッシュと、図 3(A～F はキャッシュラインを表している)のような下位レベルのキャッシュが上位レベルのデータを必ずしも待たないという、exclusive キャッシュの 2 種類の仕組みが存在する。今回実装したキャッシュエミュレータは、後者である exclusive の方を採用した。

exclusive の方を採用した背景には、まず参考文献[3]より、現在存在する CPU の多くが、この exclusive という状態を有する”MESI プロトコル”を使用しているためである。そのため現実の CPU により近い条件で研究するということが望ましいと考えたので、この exclusive 方式を採用した。

また参考文献[4]のよると inclusive の場合では下位の層に必ず上位の層のラインを持つことから、下位の層での置き換えが発生した際に、上位の層が保持するラインをライトバックまたは無効化させる必要ある。そのためキャッシュヒット率が低くなり無駄が多くなってしまうという問題点がある。対して exclusive では、下位の層のラインが置き換えられたとしても、上位の層では該当ラインが残るので、ラインの利用率が上がることになる。

さらに、参考文献[5]によると Intel 社は「マルチコアの性能は、キャッシュとメモリの構成が性能を左右する」として、1 章でも触れた”80 コア CPU”の共有キャッシュ

† 成蹊大学大学院 理工学研究科 Graduate school of Science and Technology, Seikei University

部分に exclusive の方式を採用したと発表している。このことからこれからのマルチコアでは、この exclusive の方式が多く使われていくと考え、本研究ではこの方式を採用した。

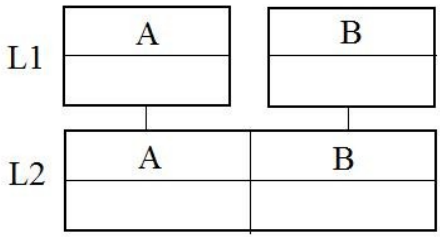

図２　inclusive キャッシュの概念図

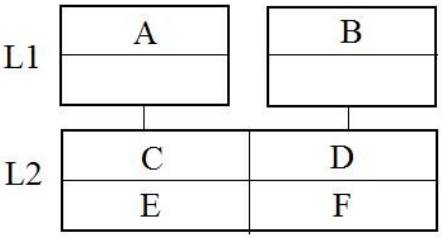

図３　exclusive キャッシュの概念図

しかし、exclusive の方式にも問題点はある。この方式では上位層にあるラインが下位層に存在しない場合がある。この場合、下位層へライトバックする際に書き込み先でのライトアロケートが必要となる。この時、書き込み先となった場所に元々存在したラインが今度は置き換え対象となる。従って、1 度の置き換えが複数の置き換えを引き起こすこともある。図 4 はその一例である。ライン A が置き換わり L2 のライン E の場所へ移る。今度は E が置き換わり L3 のライン F の場所へと移る。そしてライン F はメモリの該当場所へと書き込まれる。このように 1 度の置き換えで他に 2 回の置き換えが発生することもあるのが exclusive の問題点である。

このような形の置き換えが多く起これば、exclusive 方式の利点であるヒット率を上げることが、逆に下げることになりかねない。本研究では、この置き換え回数を調べ、その影響について考察をしていこうと考えている。

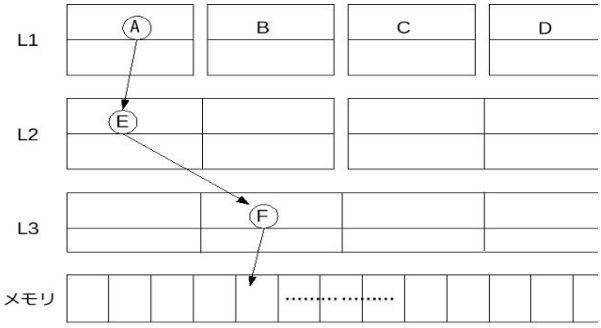

図４　ラインの置き換えの流れ

キャッシュ内でのデータのアドレスは、メモリの物理アドレスによって決まるようになっており、L1 ではメモリのアドレスの下位 5bit 目～9bit 目の 5bit で決まり、L2 では下位 5bit 目～17bit 目の 13bit で決まり、L3 では下位 5bit 目～20bit 目の 16bit で決まるようになっている。

また、キャッシュ内にあるラインの置き換えのアルゴリズムとして、以下の 3 種類を用意した。

FIFO 方式：同一インデックスのラインの内、最も古いラインから置き換えの対象としていく。

使用頻度方式：同一インデックスのラインの内、アクセス回数が最も少ないものを置き換えの対象としていく。

乱数方式：乱数を発生させ、同一インデックスのラインの内、置き換えの対象となるラインを無作為に選択していく。

4.2 で説明したように各キャッシュは、同一レベルの各キャッシュ同士がバスにて接続されている。各キャッシュは、共有データをロード・ストアする際にこのバスを介して共有データを保持する同一レベルのキャッシュを検索し、保持しているキャッシュからデータを転送してもらえるようになっている。また、ストアの際にはデータの転送だけでなく、共有データを保持する同一レベルのすべてのキャッシュのラインを無効化を行うようになっている。

5. おわりに

本報告では 128 コア CPU における共有キャッシュの挙動を調査するための CPU エミュレータの仕様についてまとめた。

実験では、キャッシュ内のラインの置き換えの回数を評価基準として、データ数の多い(最低でも L1 キャッシュ内には収まりきらない)プログラムを評価用として使用することを考えている。また、評価用のプログラムで使用するデータは、共有データ(大域変数)が多く存在するもの、非共有データ(局所変数)が多く存在するもの、そして両方が多く存在するものの 3 種類で評価していきたいと考えている。

さらに 4 章で説明したように、キャッシュ内のライン置き換えアルゴリズムを 3 種類用意したので、これらの違いがラインの置き換え回数にどのような影響を与えるのかも、調べていきたいと考えている。

参考文献

[1]”NEDO 電子情報ロードマップ 2007(コンピュータ分野)”
http://www.nedo.go.jp/denshi/roadmap/2007/index.html
[2]“IBTimes”
http://jp.ibtimes.com/article/biznews/070212/4312.html?category=technology&date=070212&id=4312
[3]“Wikipedia(MESI プロトコル)”
http://ja.wikipedia.org/wiki/MESI プロトコル
[4]“マイコミジャーナル”
http://journal.mycom.co.jp/column/architecture/019/index.html
[5]“Enterprise Platform”
http://itpro.nikkeibp.co.jp/article/COLUMN/20070524/272198/?ST=ep_infrastructure