計算機アーキテクチャ 144−29
（2001. 7. 27）

SCIMAにおけるメモリアクセス機構の検討

大根田 拓† 近藤正章† 中村 宏†

プロセッサとメモリの性能格差の問題への対処を目的として、主にハイパフォーマンスコンピューティング分野をターゲットとしたプロセッサアーキテクチャSCIMAが提案されている。SCIMAではチップ上のメモリとして従来のキャッシュに加えオンチップメモリを搭載し、オンチップメモリへのデータ転送命令として新たにpage-load/page-store命令を備える。本稿では、このSCIMAのメモリアクセス機構について検討を行なった。従来のプロセッサに簡単な拡張を行なうことで実現でき、サイクルタイムに影響を与えないことを目的とした機構を提案し、本構成で実現可能なメモリアクセスの動的スケジューリングが性能に与える影響を評価する。本構成は実現可能性、及び性能面から妥当な構成であるとの結論を得た。

A Study of Memory Access Organization on SCIMA

TAKU OHNEDA,† MASAAKI KONDO† and HIROSHI NAKAMURA †

The performance gap between processor and main memory is serious problem especially in high performance computing. In order to overcome this problem, we have proposed a new processor architecture called SCIMA, which integrates software-controllable addressable memory into processor chip as a part of main memory in addition to ordinary cache. It is necessary to extend load/store unit of conventional out-of-order processor for handling the data accesses to the on-chip memory in SCIMA. This paper shows the mechanisms and operations of load/store unit of SCIMA and their performance evaluations.

1. はじめに

近年、プロセッサの性能向上は著しいが、それに対しメモリの性能はあまり向上していない。このため、プロセッサの性能がメモリアクセスによって制限されてしまうことが問題となっている。この問題に対処するため、従来のプロセッサではキャッシュを用いるのが一般的である。通常、キャッシュ上へのデータ配置・データの置き換えはハードウェアによる制御で行われており、必ずしも全てのデータアクセスで高い効果を得られるとは限らない。

特に、ハイパフォーマンスコンピューティング(HPC)分野のアプリケーションでは、キャッシュが有効に機能しない場合が多い事が指摘されている[1]。キャッシュ容量に対しデータセットが非常に大きく、またデータの時間的局所性がほとんどないためである。結果としてキャッシュミスが頻発し、性能が大きく低下してしまう。そこで、このHPC分野のアプリケーションを対象とした新しいアーキテクチャSCIMA (Software Controlled Integrated Memory Architecture)が提案されている[2]。

SCIMAは、従来のキャッシュに加え、ソフトウェアで制御可能なチップ上のメモリ(オンチップメモリ)を持つ。オンチップメモリは、新たに追加されたpage-load/page-storeと呼ばれる命令を用いることで、オフチップメモリとのデータ転送が行なわれる。従って、ユーザによる明示的なデータ配置、データの置き換えが可能となり、個々のプログラムに対し最適なデータ転送の制御を行わせることができる。

SCIMAでは、オンチップメモリへのデータアクセスを制御するために、従来のプロセッサのメモリアクセス機構に対する拡張を考える必要がある。これまで、SCIMAのメモリアクセス機構としては、ほぼ理想的に近いメモリアクセスが可能、すなわち依存さえなければメモリアクセス命令の動的スケジューリングに制限はないという構成を仮定して検討、及び評価が行われてきた[2]。しかし、実際にSCIMAプロセッサを設計するにあたり、これまでのメモリアクセス機構では構造が複雑で、プロセッサのサイクルタイムに影響を与えると考えられる。

そこで本稿では、これまで考えられてきたSCIMAのメモリアクセス機構をより簡略化し、従来のプロ

† 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

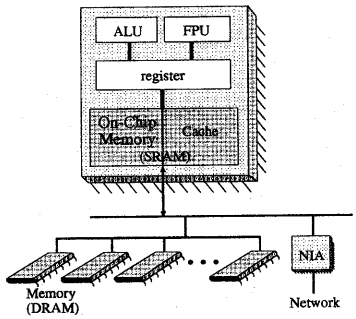


図1　SCIMAの構成

セッサに簡単な拡張を施すことで実現できる構成を検討する。また、その構成が性能に与える影響を評価することで、検討するメモリアクセス機構が妥当であることを示す。

2. SCIMAアーキテクチャの概要

図1に、SCIMAの構成を示す。

SCIMAでは、チップ上のメモリとして、キャッシュに加えオンチップメモリを搭載する。キャッシュはハードウェア制御によりデータ配置・置き換えが行われるのに対し、オンチップメモリは、ソフトウェアでデータ配置・置き換えの指定が可能である。

2.1 アドレス空間

SCIMAでは、論理アドレス空間上にオンチップメモリ領域をマップする。オンチップメモリは大きな連続ブロック領域であるため、この管理をTLBではなく専用レジスタで行ない、TLBミスの頻発を防ぐ。導入されるレジスタは、オンチップメモリの開始アドレスを保持するASR (On-Chip Address Start Register) とオンチップメモリ容量を保持するAMR (On-Chip Address Mask Register) である。

2.2 拡張命令

SCIMAでは、page-load/page-storeと呼ぶオンチップメモリ–主記憶間の転送命令をISA (命令セットアーキテクチャ) 上に備える。この命令により、オンチップメモリのデータ配置・置き換えをソフトウェアで行うことが可能となる。本命令は、データ転送元の開始番地、データ転送先の開始番地、転送サイズ、ブロック幅、ストライド幅、の5オペランドをとる。また、ブロックストライド転送機能を備え、この機能により、不連続なデータをオンチップメモリ上の連続領域に転送させることができるため、無駄なデータ転送を省き、チップ内の記憶領域を有効に利用可能である。

オンチップメモリ領域は*page*と呼ぶ複数のブロックに分割され、この*page*を単位として管理する。*page*のサイズは2のべき乗である。page-load/page-store命令で転送できる最大データサイズはこの*page*のサイズであり、*page*を跨いでの転送はできない。

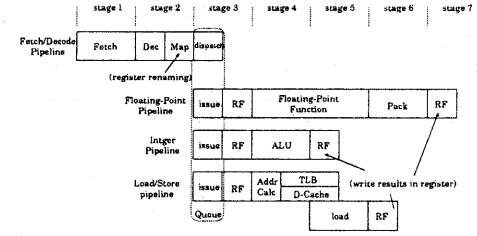


図2　MIPS R10000の実行パイプライン

2.3 キャッシュ・オンチップメモリ統合機構

SCIMAでは、キャッシュとオンチップメモリをハードウェア的に統合し、それらに割り振られる容量比を、対象とするアプリケーションの性質に合わせて実行時に再構成できる機構が提案されている[2]。本稿で検討するメモリアクセス機構においても、この構成を前提とする。

3. MIPS R10000のメモリアクセス機構

SCIMAにおけるメモリアクセス機構を検討するにあたり、MIPS R10000プロセッサの構成[3]をベースとする。本章では、そのR10000の構成について述べる。

3.1 パイプライン構成

MIPS R10000におけるパイプラインステージ構成を図2に示す。まず、stage 1で命令キャッシュから命令がフェッチされ、stage 2でデコード及びレジスタリネーミング処理が行なわれる。その後、各命令は整数演算、浮動小数点演算、load/storeという命令の種類に応じて各命令queueに格納される (図3参照)。本稿ではこの動作を*dispatch*と呼ぶ。

stage 3では、各queueにdispatchされた命令から実行可能なものが選択され、演算器へ発行される。本稿ではこの動作をissueと呼ぶ。stage 4以降では、演算やキャッシュアクセスが行なわれ、それぞれの命令のレーテンシ分のサイクルが経過した後にレジスタに値が書き込まれる。

3.2 out-of-order実行方式

レジスタリネーミング処理後、各queueにdispatchされた命令は全てのオペランドが揃い次第発行可能となり、発行可能な命令は随時issueされる。issueされた命令は続けて演算処理 (キャッシュアクセス) が行われる。このようにR10000では、dispatch後の各命令はout-of-orderにissueされ、実行される。

一方、out-of-order実行を行なうプロセッサでは、例外回復や投機実行ミスからの回復のため、実行中の命令の順番を保持していなければならない。このため、R10000では現在実行中の全命令をプログラム順に保持する*Active list*と呼ばれるqueueを持つ。Active listはFIFO構造をしており、stage 2のリネーム時に各命令の実行順が保持される。

演算を完了した命令は、Active listの当該エントリ

に完了フラグを立てる。完了フラグが立ち、かつActive listの当該エントリがFIFOの先頭に達した時点でその命令のエントリが解放される。この動作を本稿ではcommitと呼ぶ。

3.3 メモリアクセス方式

リネームステージ後、load/store命令は*Address queue*と呼ばれるqueueにdispatchされる。Address queueはcircular FIFO構造をしており、命令の格納および削除はin-orderに行なわれる。

load/store命令のうち、load命令はオペランドが揃い次第out-of-orderに実行される。ここで、先行するstore命令が同じアドレスに対するものであった場合、RAWハザードが生じる。しかしR10000では、先行するstore命令との依存関係の判定はせずに投機的にload命令は実行される。

一方、store命令はprecise interruptの保証のためにcommit時に初めてissueされる。この時、後続のload命令とのアドレスの比較を行なう。アドレスの一致したload命令が既に実行されていた場合は、そのloadは不正な値を読み出していることになるため、当該load命令以降の全命令を破棄して実行をやり直す。なお、store命令はcommit時にissueされる、つまり命令順にissueされることから、WAWハザード、WARハザードは生じない。

4. SCIMAのメモリアクセス機構

SCIMAでは、オンチップメモリを制御するために、ハードウェアの追加・拡張が必要となる。メモリアクセス以外の整数演算や浮動小数点演算の実行については、SCIMAと従来のプロセッサにおいて相違はなく、SCIMAでも従来の構成をそのまま用いる。従って、本章では特に、前述のR10000の構成を基にしたSCIMAのメモリアクセス機構について検討する。

4.1 概　　要

まず、SCIMAにおけるメモリアクセス機構を設計する上で、従来のプロセッサに対し新たに検討するべき事項を以下に示す。

- オンチップメモリに対するload/store命令の処理
- page-load/page-store命令の処理
- メモリアクセス順序保証

以下、これらについて詳述する。

4.1.1 load/store命令

オンチップメモリへのload/store命令は従来のキャッシュへのload/store命令とISA上では同じ命令であり、アクセス先のアドレスがオンチップメモリ領域であるか否かを実行時に判定することで、それぞれのメモリアクセス動作が行われる。キャッシュに対するアクセスであると判断された場合には、前節で述べたR10000と同様のメモリアクセスとなる。一方、オンチップメモリに対するアクセスの場合は、後述するpage-load/page-store命令とのメモリ依存を判定する必要がある。

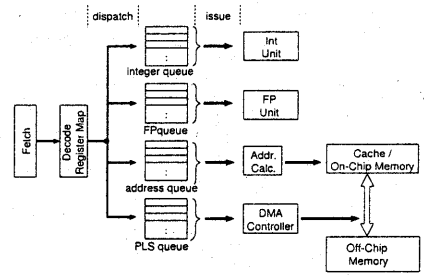


図3　各命令queueへのdispatch

4.1.2 page-load/page-store命令

page-load/page-store命令はISA上に新たに追加される命令であり、この命令を処理するためのハードウェア機構の拡張が必要となる。具体的には、*PLS queue (Page-Load/page-Store queue)*と呼ぶqueueを新たに設け、パイプラインstage 2でデコード・リネームされたpage-load/page-store命令は、このqueueにdispatchされる(図3参照)。PLS queueはaddress queueと同様にFIFO構造をしており、in-orderにエントリの追加・削除が行われる。

page-load/page-store命令はメモリの内容を書き換える命令であるため、precise interruptの保証を考えた場合、store命令同様commit時に転送を開始する必要がある。そのため、page-load/page-store命令はcommit時にin-orderにissueされ、その後DMAコントローラにより、オンチップメモリとオフチップメモリ間でDMA転送が行われる。

4.1.3 メモリアクセス順序保証

R10000におけるload/store命令間のメモリ依存の解決法、すなわちメモリアクセスの順序保証については3.3節で述べた。SCIMAでのオンチップメモリアクセスとなるload/store命令は、従来のキャッシュアクセスのload/store命令とISA上で同じ命令なので、メモリアクセス順序保証についても同様に解決することができる。また、page-load/page-store命令間のメモリアクセス順序保証は、これらの命令がin-orderに実行されることで問題とならない。しかし、page-load/page-store命令とload/store命令間のメモリアクセス順序保証については新たに検討する必要がある。

page-load/page-store命令とload/store命令のメモリ依存は*page*を単位として判定を行なう。これはpage-load/page-store命令はDMAにより転送が行なわれ、個々のアドレスで依存判定をするのが難しいためである。

page-load/page-store命令は1命令で多くのデータを転送する命令であり、page-load/page-store命令とload/store命令をin-orderに行う、すなわちload/store命令は常に先行するpage-load/page-store命令の完了

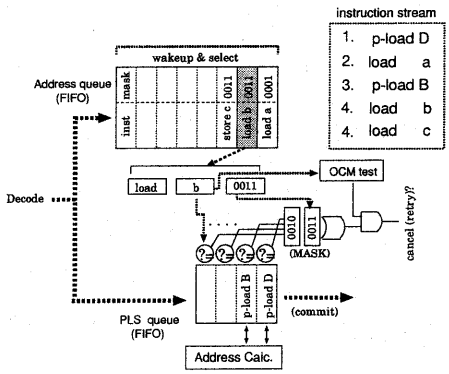


図4　メモリアクセス機構

を待たなければならないとすると、性能が大きく低下すると予想される。従って、page-load/page-store命令に対してメモリ依存のないload命令*を先行してissueできる機構を検討する必要がある。

次節では、このpage-load/page-store命令に対するload命令の動的スケジューリングを実現する機構を提案する。

4.2　メモリアクセス機構

SCIMAのメモリアクセス機構を図4に示す。前節で述べたSCIMAでのメモリアクセス制御を行なうため、R10000に対し、以下の拡張を行なう。

- PLS queue
- 先行するpage-load/page-store命令を識別するためのマスクビット
- load/store命令とpage-load/page-store命令のアドレス比較を行なうための比較器
- page-load/page-store命令のアドレス計算を行なうためのアドレス計算ユニット
- load/store命令がオンチップメモリアクセスかどうかを判定する機構(OCM test)

4.2.1　メモリアクセス機構の動作

マスクビット

load/store命令が自身に先行するpage-load/page-store命令の識別を行なうためのビットであり、load/store命令がdispatchされる時に、先行するpage-load/page-store命令が格納されているPLS queueのエントリに対応するビットがセットされる。page-load/page-store命令が実行を完了した時点で対応するビットがクリアされる。このマスクビットは、load命令実行時に先行するpage-load/page-store命令との依存判定を行なうために用いられる。

アドレス比較器

load/store命令とpage-load/page-store命令との依存

* store命令はprecise interrupt保証のため、in-orderにissueする必要があるが、store命令のissueが遅れることはそれほど性能に影響しないと思われる。

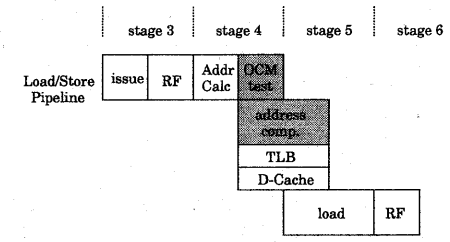


図5　load命令のパイプライン動作

判定を行なうため、PLS queueのエントリ数に対応する比較器を用意する。この比較器を用い、load/store命令の実行時にPLS queueに格納されている全page-load/page-store命令とload/store命令とのアドレスの比較を行なう。なお、*page*のサイズを2^nとすると、load/store命令とpage-load/page-store命令の下位n bitを除いた上位ビット同士を比較することになる。

アドレス計算ユニット

page-load/page-store命令がアクセスするオンチップメモリの*page*アドレス計算のため、新たにアドレス計算ユニットを追加する。計算されたアドレスはpage-load/page-store命令とともにエントリに保持される。

OCM test

OCM testでは、各load/store命令のアドレスによりオンチップメモリアクセスかどうかの判定がなされる。この動作は2.1節で述べたASRとAMRを用いて行なわれる。詳細は文献[2]を参照。

4.2.2　load/store命令のパイプライン動作

図5に、load命令のパイプライン動作を示す。R10000のload/store命令動作に対して、stage 4(後半)からの*OCM test*、*address comp.*(アドレス比較)動作が追加されている。OCM testステージは、オンチップメモリアクセスかどうかの判定である(前述のOCM testでの動作)。Address comp.は、OCM testと合わせ、前述のマスクビットとアドレス比較器を用いたアドレス比較によりpage-load/page-store命令の依存判定を行なう。依存していた場合、load命令は実行を中断し、後で再度実行を行なう。

4.2.3　page-load/page-store命令の動作

dispatchされたpage-load/page-store命令は、転送先のアドレスについてアドレス計算を行ない、PLS queueに計算したアドレスを保持する。その後、commit時点にissueされ、転送が開始される。転送が終了した後、PLS queueからエントリを削除する。

4.3　本構成における議論

3.3節で述べたR10000でのメモリアクセス順序の保証方法では、load命令が命令実行時にstore命令に対する依存を検出する必要がないために構成が簡単になる反面、依存が生じたときのペナルティが大きい。page-load/page-store命令は*page*単位でメモリ依存判定されるため、後続のload命令と依存する確率が高

```
do i = 1,N
  do j = 1, N
    p-load C                  /* block Cの転送 */
      do k = 1,N
        p-load A              /* block Aの転送 */
        p-load B              /* block Bの転送 */

        [C = A × B ]          /* 現在の計算   */
        enddo
    enddo
enddo
```

図6　no-spコード

いと考えられ、投機的にload命令を実行するのは効率が悪い。本稿で提案する機構ではload命令実行時にpage-load/page-store命令との依存を判定し、投機的にloadを実行しないためペナルティが生じない。

また、本稿で提案する構成はわずかの追加拡張で実現でき、拡張対象のR10000プロセッサのサイクルタイムに影響を及ぼさないと思われる。さらに、図5より、SCIMAでのload/store命令に対する追加動作はキャッシュアクセスに並行に行うことができパイプライン段数が増加しないため、load/store命令のレーテンシは増えないと考えている。

5. 性能評価

本性能評価では、SCIMAにおけるオンチップメモリへのload/store命令とpage-load/page-store命令間の動的スケジューリングの自由度の変更が性能に与える影響を評価し、前節で提案したメモリアクセス機構の性能について議論する。

5.1 評価モデル

評価では、動的スケジューリングの自由度が異なる以下の3種類のモデルを用いる。

- in-order: オンチップメモリアクセスのload/store命令は先行する全てのpage-load/page-store命令を追い越して実行できない。page-load/page-store命令は他の命令を追い越して実行できない。
- out-of-order(本稿で提案するモデル): load命令のうち先行する全てのpage-load/page-store命令にメモリ依存がないload命令は追い越して実行を行なう。page-load/page-store命令は他の命令を追い越して実行できない。
- ideal: out-of-orderモデルに加え、page-load/page-store命令がload/store命令を追い越して実行可能とする。

5.2 評価コード

性能評価には、行列積とNPB CG[4]のプログラムを用いる。行列積における行列サイズは200 × 200(倍精度)であり、CGのデータセットはclass Wを使用する。どちらもブロッキングを施し、page-load/page-store命令によりブロック単位での転送を行なう。また、それぞれについて、ソースコード上でpage-load/page-store命令と演算とのソフトウェアパイプライニングを行ない、後続の命令列で参照されるデータを演算に

```
p-load A                      /* 一番最初のblock A */
p-load B                      /* 一番最初のblock B */
p-load C                      /* 一番最初のblock C */
do i = 1,N
    do j = 1, N
        p-load Cnext          /* 次の計算で使われる  block Cの転送 */
        do k = 1,N
            p-load Anext      /* 次の計算で使われる block Aの転送 */
            p-load Bnext      /* 次の計算で使われる block Bの転送 */

            [C = A × B ]      /* 現在の計算 */
        enddo
    enddo
enddo
```

図7　spコード

表1　評価に用いるパラメータ

同時発行命令数	
- 整数演算	2
- 浮動小数点演算(積和)	1
- 浮動小数点演算(除算/平方)	1
- ロード・ストア	1
命令queueサイズ	
- integer・FP・load/store	各 32
- PLS	8
Active list エントリ	104
キャッシュラインサイズ	128B
pageサイズ	4KB
オフチップメモリスループット	2B/cycle
オフチップメモリレーテンシ	80cycle
メモリサイズ 合計	64kB
- no-sp コード (no software pipelining)	キャッシュ 32kB(2-way) オンチップメモリ 32kB
- sp コード (software pipelining)	キャッシュ16kB(1-way) オンチップメモリ 48kB

インターリーブして転送する。以降では、ソフトウェアパイプライニングを行なったコードをsp (software pipelining)、ソフトウェアパイプライニングを行なわないコードをno-sp (no software pipelining) とする。

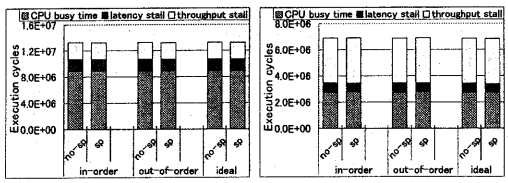
行列積について、no-spコードを図6に、spコードを図7にそれぞれ示す。図中のA、B、C、Anext、Bnext、Cnextはいずれもブロック(小行列)を表し、[C = A × B]はブロック内演算(小行列の積)を表す。また、図7における A、B、C と Anext、Bnext、Cnext のデータはそれぞれ別の*page*への転送を行なう。転送単位である*page*が異なればその pageへの load/store命令は依存とはならず、Anext、Bnext、Cnextの転送とA、B、Cの演算がオーバーラップできる。

5.3 評価環境

評価にはクロックレベルシミュレータを用いる。また、評価におけるパラメータを表1に示す。

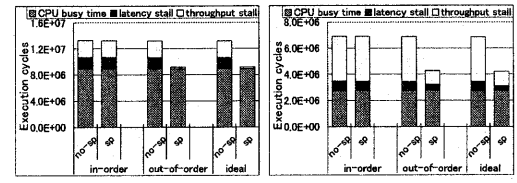
評価では、プロセッサの実行時間をCPU-busy time (T_b)、latency-stall(T_l)、throughput-stall (T_t)の3つに分類する。CPU-busy timeとはプロセッサが実際に計算処理を行っている時間であり、latency-stallは主記憶のアクセスレーテンシがもたらすストール時間を、またthroughput-stallはオフチップメモリのスループット不足に起因するストール時間を指す。

ここで、T_b、T_l、T_tを、プロセッサの総実行時間T、オフチップメモリスループットが無限大とした場合の実行時間T_∞、オフチップメモリスループットが無限



(a) 行列積　　(b) CG

図8　評価結果



(a) 行列積　　(b) CG

図9　評価結果(Active listエントリ解放モデル)

大かつオフチップメモリレーテンシが0とした場合の実行時間T_pを用い、以下のように定義する。

$$T_b = T_p$$
$$T_l = T_\infty - T_p$$
$$T_t = T - T_\infty$$

6. 評価結果、考察

行列積及びCGを用いて評価を行なった結果をそれぞれ図8-(a)、図8-(b)に示す。図8の結果では、3種類の評価モデルで性能差が見られない。従って、図8では動的スケジューリング及びソフトウェアパイプライニングの効果が見られない結果となった。

原因はActive listの影響と考えられる。本評価ではpage-load/page-store命令は転送終了までActive listの先頭にとどまるとしたため、他の命令がcommitできずにActive listに残り、Active listが溢れてしまう。Active listが溢れた時点でデコードが止められ、page-load/page-store命令の実行が終了するまでストールしてしまい、演算と転送がインターリーブされない。

これに対し、page-load/page-store命令がcommit後に転送を開始した時点でActive listのエントリを解放するモデルを新たに仮定し、評価を行なった。行列積、CGの結果をそれぞれ図9-(a)、図9-(b)に示す。図9では、out-of-orderモデル、idealモデルのspコードを用いたもので性能向上が見られる。これは、Active listが溢れることなく、期待通りに演算と転送のインターリーブが行なえるためである。

図9において、全ての評価モデルでno-spコードでの性能向上が見られない。page-load/page-store命令の転送データを直後の演算で利用するために依存関係が存在し、転送終了後まで演算開始ができないためである。

一方、in-orderモデルはspコードであっても性能向上しない。in-orderモデルではload/store命令が直接依存のないpage-load/page-store命令を追い越して実行できないため、演算に用いるload/store命令が現在転送中の(直接依存のない) page-load/page-store命令により止められてしまい、転送の終了を待たされてしまう。

out-of-orderモデルとidealモデルを比較すると、わずかにidealモデルの性能が優っているが、その性能差はほとんど見られない。よって、idealモデルでのみ可能な動的スケジューリングは、性能にあまり大きな効果はないと言える。

以上の結果より、本稿で提案するout-of-orderモデルは、commit後page-load/page-store命令の転送中にActive listのエントリを解放できれば十分高い性能を得られると言える。これを実現するためには、page-load/page-store命令の転送中にページフォールトを発生させないなど、いくつか考慮すべき点があり、今後検討する必要がある。

7. ま と め

本稿では、SCIMAのメモリアクセス機構についての検討を行った。これまで考えられてきたSCIMAのメモリアクセス機構を簡略化し、従来のプロセッサに簡単な拡張を施すことで実現でき、サイクルタイムに影響を与えない構成を意識したメモリアクセス機構を目標に検討を行った。

動的スケジューリングの自由度が異なるいくつかのモデルで性能評価を行った結果、本稿で提案するメモリアクセス機構は十分な性能が得られることがわかった。以上より、提案する構成は、実現可能性及び性能面から妥当な構成であると考えられる。今後は、提案したメモリアクセス機構について、サイクルタイムへの影響をRTLで設計し調べる予定である。

謝辞　本研究を行なうにあたり，御助言，御討論頂いた筑波大学計算物理学研究センターの関係者各位に感謝致します．なお，本研究の一部は日本学術振興会未来開拓学術研究推進事業「計算科学」(Project No. JSPS-RFTF 97P01102)によるものである．

参 考 文 献

1) D.Callahan et al., "Data Cache Performance of SupercomputerApplications", Proc. of Supercomputing '91, pp.564–572, 1990.
2) 中村宏, 近藤正章, 大河原英喜, 朴泰祐, "ハイパフォーマンスコンピューティング向けアーキテクチャSCIMA", 情報処理学会論文誌, Vol 41, No. SIG 5(HPS 1),pp. 15-27, 8 2000.
3) K.C.Yeager, "The MIPS R10000 superscalar processor", IEEE Micro, April, 1996.
4) D. Bailey, et al. "The NAS Parallel Benchmarks 2.0", NASA Ames Research Center Report,NAS-05-020, 1995. http://www.nas.nasa.gov/Software/NPB/