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

佐藤 純一　内山 真郷　伊藤 務　山崎 信行　安西 祐一郎

慶應義塾大学大学院 理工学研究科 開放環境科学専攻

E-mail: {satojun,utiyama,itomu,yamasaki,anzai}@ayu.ics.keio.ac.jp

本論文では、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ためのキャッシュサブシステムの構成について述べる。マルチスレッディングプロセッサは複数のスレッドでキャッシュを共有するため、キャッシュのミス率が 1 スレッドのプロセッサよりも高くなる。キャッシュミスによる下位メモリへのアクセスには大きな遅延が必要となるため、それによってデッドラインミスが起こる可能性がある。そのためリアルタイム処理用のプロセッサでは、優先度の高いスレッドのキャッシュヒット率を高くする必要がある。そこでスレッドの優先度を用いて下位メモリからロードしてきたメモリブロックと入れ替えるキャッシュブロックを選択することで、優先度の高いスレッドのデータをキャッシュ内に保持し、キャッシュミスを減少させる。またキャッシュミス時には、下位メモリへのアクセスを優先度の高いスレッドが先に行うようにする。このようにすることで優先度の高いスレッドのキャッシュアクセス時間を減少させ、リアルタイム処理を支援するキャッシュサブシステムを構築することができる。

A Priority Based Cache Subsystem for Real-Time Multithreading Processors

Jun-ichi Sato, Masato Uchiyama, Tsutomu Ito, Nobuyuki Yamasaki, Yuichirou Anzai

School of Science for Open and Environmental Systems
Graduate School of Science and Technology, Keio University

This paper describes a design of a cache subsystem for multithreading processors to support real-time operations. Multithreading processors tend to make cache miss rate higher than single thread processors because of sharing cache memory among multiple threads. But processors to support real-time operations should hold the rate of cache hit of high prior threads higher than other threads, because their cache miss needs a long access latency to lower memory and it may cause dead-line miss. In order to realize a real-time mechanism, this paper proposes a new method to replace cache blocks with lower memory blocks based on priority. To select blocks based on priority, a thread with higher priority remains on cache memory as long as possible. And if a cache miss happens, a high prior thread has access to lower memory, so that a latency of high prior thread to lower memory will reduce. These methods will be useful to keep dead-line because a cache access time of such a thread will reduce.

1. はじめに

近年、コンピュータ処理に求められる高品質化の一つとしてリアルタイム処理が重要となってきている。特にロボット技術の発達やネットワークの高速化において、リアルタイム処理の果たす役割は大きなものになると考えられる。

Responsive Processor[1] は **Responsive Link** と呼ばれるリアルタイム通信機構をプロセッサコアや様々な I/O ポートと同一のチップ上に備えたシステムオンチップ型の並列分散処理用のプロセッサである。**Responsive Link** を用いることで分散システムにおけるモジュール間リアルタイム通信が可能であり、例えばモジュール化されたロボットの制御やマルチメディア通信の制御等に利用できる。

μ-Pulser[2] は主にパーソナルロボットの制御用に

開発されたOSで、アルタイムスレッドや割り込みスレッドを必要に応じてスケジューリングし、リアルタイム処理を行う。

一方、より低いレベルでスレッドの制御を可能にすることでより高品質のリアルタイム処理を行なうことができる。コンピュータの処理で最も低レベルな処理はプロセッサにおける処理であるが、プロセッサ単独でリアルタイム処理を行なうことは難しい。しかし**μ-Pulser**のようなリアルタイムOSと協調し、その処理を支援する機能を備えることでリアルタイム処理用のプロセッサを実現することができる。

リアルタイム処理用のプロセッサでは、OSによって設定される優先度が高いスレッドの合計の処理時間を少なくするために、そのキャッシュミス率を減少させる必要がある。

以下本論文では、リアルタイム処理用プロセッサのキャッシュにおける問題を捉え、それを解決するためのキャッシュサブシステムを提示する。

2. 背　　景

リアルタイム処理では、定められたデッドラインまでにその処理を終了させなければならない。リアルタイムOSは、アクティブなスレッドの実行の順番をスケジューリングすることでその制約を満たさせる。リアルタイム処理用のプロセッサは、通常の処理を高速に行うことに加えOSによるスレッドの実行の制御を支援する必要がある。

リアルタイム処理はデッドラインを持つため、プロセッサに対して割り込みをかけ、それまで実行されていたスレッドの実行を中断させて優先的に処理を開始する必要がある。現在多くのプロセッサが採用している高速化・高性能化アーキテクチャは、superscalerかVLIWであるが、VLIWはコンパイラによって最適化されたコードを実行することで性能を発揮するため、割り込みの処理には適さない。また**μ-Pulser**のような複数のスレッドを使うOSの使用を考えた場合、複数のスレッドをプロセッサ上に保持するマルチスレッディングを用いると、コンテキストの切替えを高速に行なうことができるため処理の高速化が図れる。さらにリアルタイム処理では各スレッドに優先度が与えられるため、マルチスレッディングを用いることでプロセッサ上に保持されるスレッドの中から優先度の高いスレッドを選び、その実行を優先させることもできる。

複数のスレッドの命令をsuperscalar方式で実行するアーキテクチャとして、Simultaneous Multithreading(SMT)[3]方式がある。リアルタイム処理用のプロセッサアーキテクチャとしては、このSMTが適していると考えられる。

ところがSMTのようなマルチスレッディングプロセッサでは、プロセッサ上で保持・実行するスレッド間でプロセッサ資源の競合が起こる。実行ユニットにおいては、優先度の高いスレッドを先に実行したり競合が起こることが予想される実行ユニットを予め複数用意するなどの対処方法が考えられる。一方レジスタセットやキャッシュといったデータを保持する資源では、プロセッサ上に保持されるスレッド数と同数の資源を用意することが考えられる。高速なアクセスとスレッド間でのデータ保護のために、レジスタセットの場合はできるだけそのようにすることが望ましい。しかしキャッシュの場合、1スレッドのプロセッサと同じ容量のキャッシュを用意するとハードウェア量が大きくなりすぎてしまう。ところがキャッシュの全容量を1スレッドのプロセッサと同じ大きさとすれば、今度はそれぞれのスレッドが使用できるキャッシュ容量が減少してしまう。そこでチップ面積の許す限りのキャッシュを用意することになるが、それでも使用できるキャッシュ容量が小さくなることに変わりはない。

また、あるスレッドが下位メモリからロードしてきたデータブロックを他のスレッドが入れ換えてしまい、それによるキャッシュミスでキャッシュアクセス時間が増加することも考えられる。

つまりマルチスレッディングプロセッサではキャッシュ性能が低下することが予想される。

もしキャッシュミスが生じると、当該アドレスのデータブロックを下位メモリからロードしなければならない。特に外部下位メモリはプロセッサコアよりも低速で動作するため、そのアクセスには大きなレイテンシが必要となり、プロセッサの性能向上を妨げる大きな要因となってしまう。

そこで通常のマルチスレッディングプロセッサは、キャッシュミスのような大きなレイテンシが必要となる場合には実行するスレッドを切り替える。これによってキャッシュミスによるレイテンシを隠蔽し、性能の低下を防ぐ。しかし先に述べたように、リアルタイム処理ではデッドラインまでに処理を終了させる必要がある。そのため優先度の高いスレッドの実行が、プロセッサによるスレッド切り替えのために中断してしまったり、下位メモリへのアクセスにより長い時間待たされるようなことがあってはならない。

よってリアルタイム処理用マルチスレッディングプロセッサでは、優先度の高いスレッドのキャッシュヒット率を高くする必要がある。

そこでキャッシュヒット率の向上を考えるためにキャッシュミスを分類してみると、**初期参照ミス**、**容量ミス**、**競合ミス**の三つに分けることができる。

初期参照ミスは、参照するアドレスが初めて参照されるアドレスであるため必ず起きてしまう。容量ミスは参照対象のアドレス領域がキャッシュの物理量よりも大きいために生じる。そこでキャッシュ容量を大きくしたり、コンパイラによってキャッシュ内のデータ配置を最適化することである程度は回避できる。三つ目の競合ミスは、既にキャッシュにあるデータブロックを下位メモリからロードしてきたデータブロックで置き換えてしまうために生じる。

この競合ミスは、複数のスレッドが同じキャッシュブロックを使用することが考えられるマルチスレッディングプロセッサでは、特に多く発生することが考えられる。そこでこの競合ミスを減少させるために、入れ換えるキャッシュブロックの選択方法を考えてみると、既存の多くのプロセッサでは LRU(Least Recently Used) を用いている。これは入れ換えの対象とするキャッシュブロックセットの中で、最も古くに参照されたブロックを入れ換える方法である。この LRU を用いた方法は、キャッシュの参照に局所性がある場合には同じキャッシュブロックが連続して参照されるため十分な性能を発揮する。しかし異なる複数のスレッドによってアクセスがなされると、それぞれのアクセスには局所性があっても全体としては局所性が失われ、その結果競合ミスが生じることになる。これを防ぐためにキャッシュの連想度 [4] を高める方法があるが、連想度をあまり高めるとキャッシュブロックの選択のために長い時間がかかり、チップ面積も大きくなってしまう。むしろ将来必要になると思われるデータブロックを入れ替えないことで、競合ミスによるキャッシュミスを減少させることができる。

本論文では、この競合ミスを減少させることで高い優先度のスレッドの実行を支援するキャッシュサブシステムを設計する。

3. 関 連 研 究

リアルタイム処理用プロセッサのキャッシュにおける問題点は、優先度の高いスレッドのキャッシュブロックが、他のスレッドによって入れ替えられてしまうことである。

そこで解決策の一つとして、優先度の高いスレッドに固定のキャッシュ領域を割り当てることが考えられる。他のスレッドによる当該キャッシュ領域の使用を禁止し、キャッシュブロックが入れ替えられてしまうことを防ぐ。

[Tullsen *et al.* 95][3] ではプロセッサ上の各スレッドに、[Lioupis 96][5] ではリアルタイム処理が必要なスレッドに、それぞれ固定のキャッシュ領域を割り当てその性能評価を行なっている。[Tullsen *et al.* 95] は SMT プロセッサのキャッシュの構成方法を考えるための評価であるため特にリアルタイム性を考慮しているわけではないが、[Lioupis 96] はリアルタイム処理が必要なスレッドのキャッシュ性能の向上を目指している。しかしいずれも固定されたキャッシュ領域内で容量ミスが発生するため、[Tullsen *et al.* 95] では固定のキャッシュを割り当てない方が良い性能が得られ、[Lioupis 96] でも性能向上はほとんど見られない。

一方 [Lioupis 96] では、下位メモリへのアクセスをパイプライン化することでキャッシュ性能を大幅に向上させることが示されている。つまりプロセッサコアに対して低速となる外部下位メモリへのアクセスを改善することで、キャッシュミスによる性能低下を軽減することができる。

4. 設　　計

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キャッシュサブシステムの設計をするにあたり、本研究においては次の二点に焦点を絞る。一つ目は競合ミスを減らすことで優先度の高いスレッドのキャッシュヒット率を向上させることである。二つ目はキャッシュミスを起こした場合の下位メモリへのアクセス方法を改善し、優先度の高いスレッドの下位メモリのアクセス待ちの時間を減少させることである。

4.1 優先度によるキャッシュ入れ替え機構

優先度の高いスレッドのキャッシュヒット率を向上させるために、下位メモリからロードしてきたデータブロックを入れ換えるキャッシュブロックの選択方法を、LRU を用いた方法からスレッドの優先度を用いた方法に変更する。これは入れ替えを行うブロックを選択する時に、優先度の最も低いスレッドが使用しているブロックを入れ替えの対象とする。このようにすることで優先度の高いスレッドのブロックが入れ替えられることが少なくなるため、競合ミスを減少させることができる。

もし最も低い優先度のスレッドが使用するブロックが複数あった場合には、LRU を用いてそれらの中から入れ替えの対象とするブロックを選択する。また有効なデータを保持していないブロックは最優先で入れ替えの対象とする。よって入れ替え対象のブロック選択は、

(1)　そのブロックを使用するスレッドの有効・無効
(2)　そのブロックを使用するスレッドの優先度
(3)　そのブロックの参照履歴
の順によって決定する。

この機構を実現するために、キャッシュタグは図 1 に示すような構成をとる。

24:6 upper address(31:13)	5:4 LRU	3:1 thread number	0 valid

図 1　Tag の構成

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、タグとして優先度を保持することはしない。その代わりにプロセッサコアにある各スレッドの優先度を保持するテーブルをタグに保持されるスレッド番号で参照することで、そのスレッドの優先度を得る。

4.2　優先度による SDRAM アクセスの追い越し機構

キャッシュミスが生じると、下位メモリから当該データをロードしなければならない。本研究においてはキャッシュの次のメモリ階層は外部の SDRAM とするため、プロセッサコアとの速度差は非常に大きなものとなる。

しかしキャッシュミスは複数のスレッドで生じたり、あるいは命令キャッシュとデータキャッシュの双方で生じることもある。SDRAM へのアクセスはバースト転送ができなければそれぞれの要求毎のシングル転送となり、また SDRAM へのアクセスをその要求が生じた順に処理していくとなると、時間的に後方で生じたアクセス要求は長い間待たされることになる。そこでそのような場合に、アクセスを待っているスレッドのうち優先度が高いものから先に SDRAM へのアクセスを行うようにする。これによりアクセス待ちが生じている場合に、後から生じた優先度の高いスレッドのアクセス要求が先に処理されるようになるため、その待ち時間を削減することができる。

また外部の SDRAM として、通常の SDRAM よりも高速なアクセスが可能な DDR (Double Data Rate) SDRAM を使用する。この DDR SDRAM はクロックの立上りと立ち下がりに同期してデータの入出力を行うため、通常の Single Data Rate の SDRAM よりも 2 倍の速度でデータ転送が可能である。

4.3　キャッシュの構成

図 2 にキャッシュサブシステムのブロック図を示す。キャッシュの入れ換え機構は命令・データそれぞれのキャッシュコントローラに実装し、SDRAM アクセスの追い越し機構は図 2 中の Arbitor 部に実装する。

本研究室で設計しているプロセッサでは、スレッド間でのメモリ保護のために MMU をキャッシュとプロセッサコアの間に配置する。そのためキャッシュのアクセスには物理アドレスが用いられる。

キャッシュの構成には 4 way set-associative 方式を採用する。これはマルチスレッディングプロセッサのため連想度が 2 では競合ミスが増加することが考えられる一方、連想度を 8 まで大きくすると今度はビット幅が広がるためデータの選択に要する時間が大きくなり、キャッシュアクセスが遅くなってしまうことが考えられるからである。

またマルチスレッディングプロセッサであるためノンブロッキングキャッシュとする。もしあるスレッドがキャッシュミスを起こした場合、キャッシュコントローラに用意されたバッファにアドレスとスレッド番号を保持し、下位メモリからのロードを待つ。

キャッシュ容量については仮に 32KB と定める。キャッシュ容量は大きければ大きいほど容量ミスを減らすことができるため、プロセッサコアなど他の部分が占める大きさによっては更に容量を増加させることも検討する。

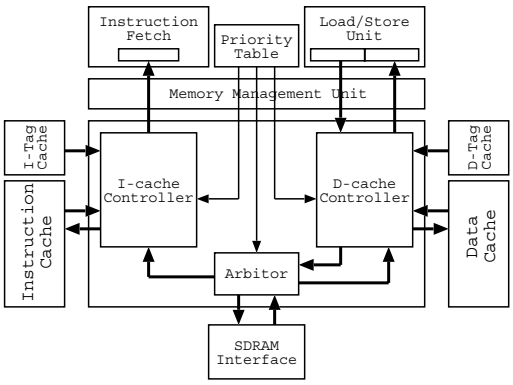

図 2　キャッシュブロック図

5.　評　　価

本論文が提案するキャッシュの構成方法を評価するためには、リアルタイム処理が必要なスレッドとそうではないスレッドを同時に実行させ、リアルタイム処理が必要なスレッドの実行時間を計るべきである。しかしプロセッサコアが未だ設計中であるため、本論文においては次のような方法で評価を行う。

複数個のスレッドを仮定し、それぞれが異なるアド

レス空間を用いてラウンドロビン方式でキャッシュをアクセスする。ただしブロックの入れ替えや SDRAM アクセスの待ち状態が発生しやすいように、同じキャッシュブロックを使用するようにアドレスを設定する。各スレッドには **μ-Pulser** に合わせた 4 段階の優先度を与え、特定数のデータを読み終えるまでの待ち時間を計測する。

図 3 に、最も優先度の高いスレッドのキャッシュアクセス待ちの時間と、並行してキャッシュアクセスを行うスレッド数の関係を、図 4 には全スレッドの平均キャッシュアクセス待ち時間とスレッド数の関係を示す。

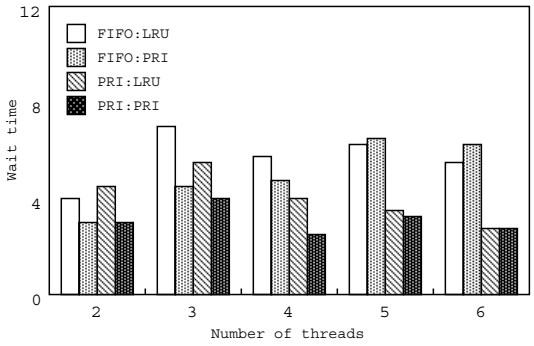

図 3 最も高い優先度のスレッドの待ち時間

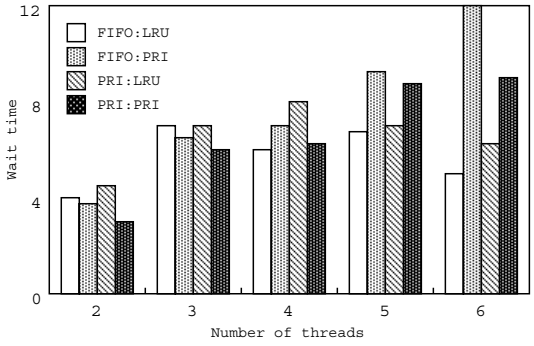

図 4 全スレッドの平均待ち時間

それぞれの図において、横軸にはスレッド数を、縦軸には 1 スレッド時の待ち時間を 1 としてそれぞれのスレッド数の時の待ち時間を正規化したものをとる。またグラフの説明には「SDRAM のアクセス機構：キャッシュの入れ換え機構」という表記を用いる。SDRAM アクセス機構では、「FIFO」がアクセスを要求順に処理することを示し、「PRI」は優先度の高いものを先に処理することを示す。キャッシュの入れ換え機構では、「LRU」が LRU のみを用いてキャッシュブロックを選択することを示し、「PRI」が優先度を用いてブロックを選択することを示す。

5.1 キャッシュ入れ換え機構の評価

図 3 の SDRAM アクセスが FIFO 形式のものを見てみると、スレッド数が 4 までは優先度を用いてキャッシュを入れ換える方式の方が LRU を用いてキャッシュを入れ換える方式よりも少ない待ち時間となっており、LRU を用いた方式よりもキャッシュミスを減少させていることが分かる。しかし図 4 が示すように、平均のアクセス待ち時間は非常に増加している。これは優先度の低いスレッドのキャッシュミスが増加していることを示している。またスレッド数が増えるにつれ優先度を用いた方式は待ち時間が増加し、4 スレッドを越えるとその性能は LRU よりも悪くなっている。これは優先度の最も高いスレッドがキャッシュを占有してしまうためにその他のスレッドによる SDRAM へのアクセスが増加し、その結果最も優先度の高いスレッドがキャッシュミスを起こした場合に SDRAM アクセスが待たされてしまうためであると考えられる。

5.2 SDRAM アクセスの追い越し機構の評価

次に、SDRAM アクセスを高い優先度のスレッドが先に行う場合とそうではない場合を比べてみる。

図 3 を見ると、ほぼ全てにおいて SDRAM アクセスを高い優先度のスレッドが先に行う方が、そうでない場合よりも少ない待ち時間となっている。特にスレッド数が多くなるほど、その待ち時間の削減を果している。また図 4 より、キャッシュの入れ換えに優先度を用いたために増加していた全体的な待ち時間を減少させることにも成功している。

これらの結果は、優先度の高いスレッドがキャッシュの使用と SDRAM アクセスの両方を優先的に行なうためにキャッシュミスとそれに伴う待ち時間が減少し、その実行が早く終了しているためにキャッシュ領域の解放も早く行なわれ、次レベル以下の優先度のスレッドの実行終了も早まるためだと考えられる。

6. 結　　論

本論文では、キャッシュの入れ換えブロックの選択、及びキャッシュミス時の SDRAM アクセスの順番の変更にスレッドの優先度を用いることで、優先度の高いスレッドのキャッシュアクセス時間を減少させることに成功した。この結果より、本論文の提案するキャッシュの構成方法が優先度の高いスレッドのキャッシュミスを減少させ、またキャッシュミスが生じた場合にも優先度の高いスレッドの下位メモリのアクセス時間を削減することが示された。よってこのキャッシュの

構成方法を導入することで、リアルタイム処理用マルチスレッディングプロセッサにおけるリアルタイム処理の性能を向上させることができると考えられる。

今後の課題としては、プロセッサコアの完成を待って実際のプログラムを実行させ、評価を行うことが挙げられる。

参 考 文 献

1) 山崎信行, 松井俊浩：並列分散リアルタイム制御用レスポンシブプロセッサ, 日本ロボット学会誌, Vol. 19, No. 2, pp. 1001–1010 (2001).
2) 矢向高弘, 菅原智義, 安西祐一郎：μ-pulser: パーソナルロボットを構築するためのオペレーティングシステム, 電子情報通信学会論文誌, pp. 207–214 (1994).
3) D.M.Tullsen, , S.J.Eggers, and H.M.Levy., : Simultaneous multithreading: Maximizing on-chip parallelism., in *The 22nd Annual International Symposium on Computer Architecture*, pp. 392–403 (1995).
4) L.Hennessy, J. and A.Patterson, D.: *Computer Architecture A Quantitative Approach*, Morgan Kaufmann, second edition (1995).
5) Lioupis, D.: Real Time Behaviour of Multithreaded Processor, *Real-Time Magazine*, No. 4, pp. 98–102 (1996).