計算機アーキテクチャ　146－2
（２００２．　２．　１）

単一チップマルチプロセッサ・アーキテクチャ SKY におけるメモリ同期機構の評価

市村和人　　小林良太郎　　安藤秀樹　　島田俊夫

名古屋大学大学院 工学研究科

近年のマイクロプロセッサは、スーパースカラ・アーキテクチャにより、より多くの命令レベル並列をプログラムより引き出し高性能化を図ってきた。しかし、この方法は、スーパースカラ・プロセッサが引き出すことのできる命令レベルの並列性の限界や、ハードウェアの複雑さの増加により、限界が見え始めてきた。これを解決する１つの方法として、スレッドレベル並列を利用したマルチスレッド実行がある。マルチスレッド実行では、ロード命令を投機的に実行し、スレッド間でのメモリ同期を効率よく行う必要がある。我々はマルチスレッド実行により性能を向上させる単一チップ・マルチプロセッサ SKY を提案してきた。本論文では、SKY におけるスレッド間のメモリデータ依存を詳細に調査し、性能への影響を調査した。また、簡単な同期機構を実装し評価した結果、データ依存違反によるスレッド破棄を最大で２プロセッサ構成で 72%、４プロセッサ構成で 47%削減し、そのときの性能低下もそれぞれ 2%ポイント、12%ポイントに抑えることができた。

Evaluation of Memory Synchronization Mechanisms in a Single-Chip Multiprocessor Architecture SKY

Kazuhito Ichimura, Ryotaro Kobayashi, Hideki Ando, and Toshio Shimada

Graduate School of Engineering, Nagoya University

Current microprocessors have improved performance by exploiting more amount of instruction-level parallelism (ILP) from a program through superscalar architectures. This approach, however, is reaching its limit beacause of the limited ILP available to superscalar processors and the growth of their hardware complexity. One of approaches that solves those problems is multithreaded execution to exploit thread-level parallelism (TLP). In multithreaded execution, processors must execute load instructions speculatively, and synchronize inter-thread memory access efficiently. We have proposed a multi-processor architecture, called SKY, which efficiently executes multiple threads in parallel. In this paper, we examine the inter-thread memory access dependencies in the SKY architechture, then evaluate the impact on performance. Then we propose a simple memory syncronazation mechanism and evaluate it. Our evaluation results show that the proposed mechanism reduces the number of squashed threads die to memory dependence violation by 72% and 47% in 2 and 4 processors organizat ion, respectively. In this case, the proposed mechanism can suppress performance degradation by 2% points and 4% points, respectively.

1　はじめに

スーパースカラプロセッサに変わるアプローチとして、最近、複数のプロセッサを単一チップに集積するマルチプロセッサの研究が行われている [1, 2, 3, 6, 7, 8, 9, 10, 13]。その背景には、単一制御流において利用可能な命令レベル並列性 (**ILP**: Instruction-Level Parallelism) が限界に近づきつつあるなど、スーパースカラプロセッサに対する悲観的観測に加え、半導体回路技術の進歩にともない、単一チップへ複数のプロセッサを集積化が可能となっていることがある。

単一チップマルチプロセッサでは、複数のプロセッサが集積化されているので、通信レイテンシを減少させることができるという利点がある。これにより粒度の小さなスレッドレベル並列性 (**TLP**: Thread-Level Parallelism) が利用可能となる。中でも、レジスタ値を直接通信し、同期を行う機構 [6, 7] は、メモリを介して同期／通信を行う機構に比べてオーバーヘッドを大幅に減少させることができる [6]。しかし、この機構にはまだ改善の余地がある。なぜならば、同期が成立せず受信待ちとなった命令が後続命令の実行を停止させ、ILP の利用を妨げる問題があるからである。

そこで我々は、**SKY** と呼ぶ単一チップ・マルチプロセッサのアーキテクチャを提案した [3]。SKY は、同期通信を行う必要がある命令が現れても、その命令とデータ依存関係がない後続命令の実行を妨げることはない。このため、スレッド内の ILP 利用を阻害することなく、細粒度の TLP を十分に引き出すことができる。

マルチプロセッサでは、スレッドの並列実行によりプログラムの意味が変わらないようにするため、レジスタとメモリに関するスレッド間のデータ依存関係を見つけだし、同期／通信を行う必要がある。このうち、レジス

タに関してはコンパイラでの静的な解析が比較的容易に行える。

一方、メモリに関しては、静的な解析が容易であるとは限らない。特に、非数値計算プログラムはデータ依存関係が複雑なため解析が困難である。この理由としては、ポインタが多用された複雑なデータ構造を扱ったり、ポインタエイリアシング問題が発生したりすることが挙げられる。この問題に対し、ハードウェアで動的に解析を行い、データ依存違反を効率よく検出する手法に関する研究が行われている (ARB[11]、SVC[12]、Hydra[13] や Multiscalar[15] の機構など)。これらの方式では、この手法においてメモリ依存違反の発生により大きなペナルティを被る可能性があり、違反による性能低下を防ぐためにメモリ同期機構が重要となると考えられる。そこで、本論文ではメモリ同期機構についての議論を行う。

従来の SKY では、理想的なメモリ同期機構を仮定し、メモリに関するデータ依存違反は発生しないものとしてきた。本論文では、まず第一に、SKY におけるスレッド間におけるメモリ依存が性能に与える影響について詳細に評価を行った。次に、メモリ依存違反が性能に与える影響を緩和するために、メモリ同期機構を提案し評価した。その結果、メモリ依存違反によるスレッド破棄を 2 プロセッサ構成で 72%、4 プロセッサ構成で 47%削減できた。この削減により性能低下もそれぞれ 2%ポイント、12%ポイントに抑えることができた。

本論文は 2 章で SKY の概要について説明した上で、3 章でメモリに関するデータ依存違反について述べる。4 章ではメモリ同期機構について議論する。その後、5 章で提案した同期機構について理想的な同期機構と比較評価を行う。

2 SKY の概要

本章では SKY のアーキテクチャ [3] について述べる。SKY は、図 1 に示すようにリング・バスで結合された複数のスーパスカラ・プロセッサからなる。細粒度の TLP 利用するため、プロセッサ間でレジスタ値を直接送受信し、同期／通信のオーバーヘッドを 2 サイクルまで減少させている。また、**命令ウィンドウ・ベース**の同期と呼ぶ同期機構を導入している。この機構は、命令ウィンドウで受信値に対応するタグを用いてレジスタに関する同期をとる機構である。この機構により、同期により後続命令の実行がブロッキングされることはなく、プロセッサは ILP を効率良く利用することができる。

スレッド並列実行のためのオーバヘッドを小さくするため、SKY のマルチスレッド・モデルは通常のマルチスレッド・モデルと比べて次に示す制約を課している。これは、**マルチスカラ・プロセッサ**[7] や **MUSCAT**[8] と同様のモデルである。

- 各スレッドは、逐次実行における動的に連続する部分で構成される。
- 各スレッドは、逐次実行の順において自分の直後のスレッドを生成する。

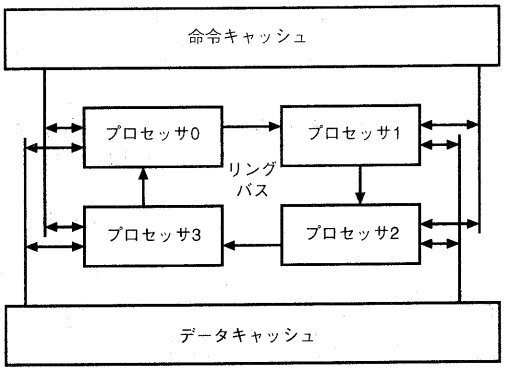

図 1: SKY のハードウェア構成

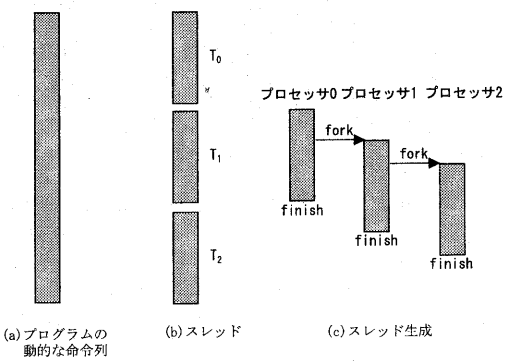

図 2: SKY のマルチスレッド・モデル

図 2(a) に逐次実行命令列を、図 2(b) にこれに対する SKY におけるスレッド分割の様子を示す。同図に示すように、SKY における各スレッドは、動的な命令列における単一の連続した部分からなり、異なる複数の部分からは構成されない。したがって、スレッドに結合はなく、制御に関する同期は必要ない。

図 2(b) に示したスレッドを並列に実行する様子を、図 2(c) に示す。図 2(b) において、各スレッド T_0、T_1、T_2 と逐次実行の順に名前をつける。図 2(c) に示すように、スレッド T_i は実行途中で、スレッド T_{i+1} を生成するという逐次生成を繰り返す。各スレッドは実行中には高々 1 回しか新しいスレッドを生成しない。実行のできるだけ早い時期に新しいスレッドを生成することにより、スレッドの並列実行を実現する。スレッドの生成は、`fork` と呼ぶ専用の命令を用い、終了は `finish` と呼ぶ専用の命令を用いて行う。以下、T_{i+1} を T_i の**子スレッド**と呼び、T_i を T_{i+1} の**親スレッド**と呼ぶ。

SKY のレジスタに関する同期／通信は命令レベルで行う。通信には `send` 命令と呼ぶ専用の命令を用いることで、親スレッドから子スレッドへレジスタ値を送信する。

従来の SKY では、理想的なメモリ同期機構を仮定し、メモリに関するデータ依存違反は発生しないものとし

て、性能向上の上限を示していた。本論文では現実的な方式について次章以降で検討し評価を行う。

3 メモリ依存違反

一般にスレッド間のメモリ依存に対して、ソフトウェアで解決する方法とハードウェアで解決する方法の2種類がある。ソフトウェアで解決する方法とは、コンパイラがプログラムの解析を行い、専用命令やアノテーションなどでハードウェアに指示を与え、ハードウェアはコンパイラの指示に従い同期待ちを行う、というものである。

このようなソフトウェアによる方法は数値計算プログラムと比べ、データ依存関係が複雑な非数値計算プログラムに対して適用することは困難である。そこでSKYでは、スレッド間のメモリ依存ついてはハードウェアで解決を行う。

ハードウェアでスレッド間のメモリ依存の解決は不完全でありスレッド間メモリ依存違反が発生する可能性がある。このため、メモリ依存違反検出機構とメモリ依存違反からの回復機構が必要となる。これらの機構の動作について以降の節で説明を行う。

3.1 メモリ依存違反検出機構

メモリ依存違反とはあるアドレスに対してR(子スレッドでの load) とW (親スレッドでの store) のアクセス順序違反であり、**RAW**(Read-After-Write) **違反**と呼ばれる。この違反の検出方法は次の手順で行う。子スレッドがロード命令を実行する際、ロードアドレスをロードアドレスバッファに保持する。親スレッドがストア命令を実行する際には、ストアアドレスをインデックスに子スレッドのロードアドレスバッファを検索する。エントリが見つかれば、子スレッドが過去にストアアドレスを参照していることを示すので、メモリ依存違反を引き起こしていることが判明する。

この他のメモリ依存違反として **WAW**(Write-After-Write) **違反**が考えられる。しかしSKYでは、各プロセッサが実行したストア命令はプロセッサ内のストアバッファに保持され、ストアバッファからキャッシュ・メモリへの書き込みはインオーダで実行されるため、WAW違反は発生しない。このため本論文では、メモリ依存違反としてRAW違反のみを扱うこととする。

3.2 メモリ依存違反からの回復機構

次にメモリ依存違反からの回復機構について述べる。親スレッド T_0 の実行したストア命令によって子スレッド T_1 のロード命令がメモリ依存違反を引き起こしたとする。この時、メモリ依存違反からの回復方法として3つの方法が考えられる。

1. ロード命令に依存する命令のみを選択的に再実行する方法
   T_1 のロード命令及び、ロード命令に依存する命令について再実行を行うことで違反からの回復を行う。このような回復方法についてはスーパースカラプロセッサにおいても選択的再実行として研究が行われているが非常に複雑な機構が必要となるため現実的でない。さらに、T_1 の子スレッドについてはどの命令が違反を引き起こしたロード命令に依存しているのか検出できない。そのため、この手法は現実的でない。
2. スレッドの先頭から再実行する方法
   T_1 の実行結果を全て破棄して T_1 の先頭から実行を再開することで違反からの回復を行う。この方法を実現するために必要となる情報はスレッドの先頭アドレスと親スレッド T_0 から送信されてきたレジスタ値だけであり、動作も現実的である。T_1 の生成する子スレッドについては破棄する必要がある。
3. 子スレッドを破棄する方法
   T_1 の実行結果を全て破棄して T_1 を実行していたプロセッサを解放することで違反からの回復を行う。この場合、SKYのマルチスレッドモデルでは T_1 については親スレッド T_0 が継続して実行するためプログラムの意味は保たれる。2の方法と似ているが、T_1 を破棄する点が異なる。プロセッサを解放するだけでよいため、2よりも現実的である。

本研究では、最も現実的な方法であると考えられる3の手法を採用する。この手法では、メモリ依存違反が発生した場合にスレッド破棄という大きなペナルティを被る。このペナルティを回避するためのメモリ同期機構について4章で議論する。

4 メモリ同期機構

この章では、スレッド間メモリ依存について詳細な評価を行う。そして評価結果を基に、メモリ同期機構を提案する。

4.1 評価環境

まず、評価環境について説明を行う。ベンチマークプログラムとして、SPECint95の全8種類を使用した。実行駆動型シミュレータ SimpleScalar 3.0 をベースにSKYシミュレータを作成し、評価した。ベースラインとなるスーパースカラプロセッサ及びSKYを構成する1プロセッサの構成を表1に示す。

表1: プロセッサ構成

フェッチ幅	8命令
発行幅	8命令
命令ウィンドウ	64エントリ
リオーダバッファ	128エントリ
機能ユニット	Int ALU × 8、Load/Store unit × 4、FP ALU × 8、Send unit × 4
命令キャッシュ	全てヒット
データキャッシュ	全てヒット
分岐予測機構	1024エントリ連想度2のBTB、1024エントリ履歴長4のPAp
分岐予測ミスペナルティ	4サイクル

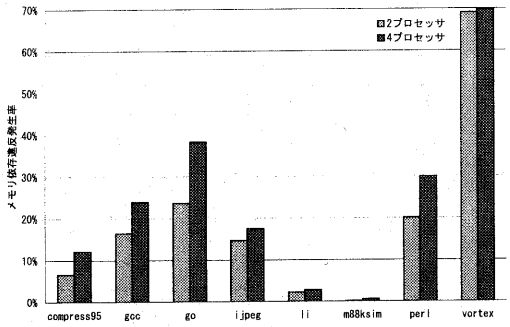

図 3: メモリ依存違反の発生割合

表 2: メモリ依存違反を起こす静的ロード命令の統計

ベンチマーク	メモリ依存違反		
	可能性	違反	割合
compress95	154	4	2.6%
gcc	16444	286	1.7%
go	9714	286	2.9%
ijpeg	2236	35	1.6%
li	101	4	4.0%
m88ksim	388	3	0.8%
perl	1381	28	2.0%
vortex	3126	85	2.7%

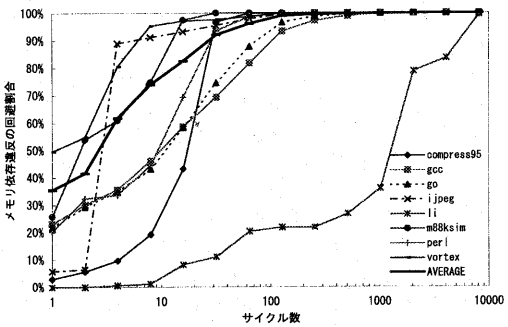

図 4: 同期待ち時間

4.2 メモリ依存違反の発生割合

SKY コンパイラ[4, 5] はスレッドの候補を作成、選択する際にプロファイルを参照し、スレッド間でのメモリ依存距離についての計算を行っている。ただし、マルチスレッド実行した際にスレッド間でメモリ依存違反が発生するかについては見積もりができないため、メモリ依存違反の発生割合については考慮されていない。

本節では、メモリ依存違反が発生するスレッドの割合について評価した。図 3 に評価結果を示す。各ベンチマークごとに 2 本の棒グラフがあり、左側がプロセッサ数 2 の場合、右側がプロセッサ数 4 の場合である。縦軸は全動的スレッドに対しメモリ依存違反により破棄されたスレッドの割合である。

vortex はスレッドが破棄される割合が 70%と最も高いが、スレッド別に詳しく調べたところ 2 つのスレッドがスレッド破棄率の 60%を占めていることが判明した。このことからメモリ依存違反が発生する個所は動的には多いが静的には少ないと言える。

他のベンチマークでは 2 プロセッサでは最大 24%と比較的少ないことがわかる。4 プロセッサでは vortex(70%)、go(38%) とスレッド破棄率の高いベンチマークもあるが、やはり全体的にはスレッド破棄率は低い。

これら一部のスレッドでのみ発生するメモリ依存違反を回避することで性能低下を抑えることができると考えられる。

4.3 メモリ依存違反を起こす静的ロード命令

メモリ依存違反を引き起こしスレッドが破棄される原因となったロード命令について調査した。表 2 にメモリ依存違反を起こす静的ロード命令の統計値を示す。表において、「可能性」はスレッドが同時に実行されているためにメモリ依存違反を引き起こす可能性のあるロード命令の静的な数である。スレッドが同時に実行されていない場合は依存違反を引き起こす可能性がないので含まれない。また、「違反」はメモリ依存違反を実際に引き起こしたロード命令の静的な数である。

ロード命令がメモリ依存違反を引き起こす割合は最大でもわずかに 4%であった。また、メモリ依存違反を引き起こすロード命令は最大でも 286 命令であり、違反を引き起こす可能性のあるロード命令の数と比較すると非常に少ないことが分かる。gcc、go、vortex を除けば、このようなロード命令は 35 以下とさらに少なくなる。これらより、非常に少数のロード命令に着目してメモリに関する同期を行うことでメモリ依存違反を回避できると考えられる。

4.4 メモリ依存違反を回避するのに必要なサイクル数

前節で述べたメモリ依存違反を引き起こすロード命令は何サイクル同期待ちを行えば違反を回避できたかについての調査を行った。図 4 は、横軸に同期待ちを行うサイクル数、縦軸はそのときメモリ依存違反が回避できるスレッドの割合を累積分布で示す。

平均で見ると 30 サイクル以内に 90%のメモリ依存違反が解消される。特に違反の多かった vortex は 1 サイクル同期待ちを行うことで 50%のメモリ依存違反を回避できることが分かる。

4.5 同期機構

前節までの議論から、実際にメモリ依存違反を引き起こすロード命令の静的な数は少なく、ほとんどのロード命令は 30 サイクル程度待つことでこの違反を回避できることが分かった。

このようなメモリ依存違反の特徴を考慮にいれてメモリ同期機構を提案する。この機構は、ロード命令アドレスと同期待ちサイクル数を保持するテーブルを用いる。このテーブルにはメモリ依存違反を引き起こしたロード命令のアドレス (PC) とメモリ依存違反に必要なサイク

ル数を書き込む。ロード命令を実行する際には、ロード命令のPCをインデックスとしてテーブルを参照し出力される同期待ちサイクル数分だけ命令の実行を遅らせることで同期を行う。エントリがなければ、同期待ちは不要と見なしすぐにロード命令を実行する。

この機構で必要とされるテーブルのエントリサイズは表2から300エントリ程度でよいことがわかる。

5 評価

本章では4.5節で提案した機構についての評価を行う。

5.1 評価モデル

評価したモデルを以下に示す。

- none モデル
  同期を行わないモデルである。
- val モデル
  同期を行わないモデルである。ただし、メモリ依存違反の検出の際にストアした値とロードした値を比較する。本来はメモリ依存違反と見なすべき場合でも値が一致した場合には、違反としない。
- wait モデル
  Alpha[14] における **load wait table** をマルチプロセッサ用に拡張したモデルである。この機構の動作としては、メモリ依存違反を検出したら違反したロード命令のPCをload wait tableに記録する。このテーブルに記録されているロード命令は親プロセッサが実行を完了するまで同期待ちを行う。このテーブルは、100kサイクルごとにフラッシュを行う。
- time モデル
  4.5節で提案した機構を備えたモデルである。
- ideal モデル
  理想的にメモリ同期を行うモデルである。

本論文では評価を行っていないが、**MDPT**[15] のような複雑な機構についても現在研究を行っている。このような複雑な機構を用いてSKYアーキテクチャ上で同期を行った場合の性能への影響の調査については今後の課題である。

5.2 メモリ依存違反削減量

メモリ同期機構によってメモリ依存違反がどの程度減少するかを図5、6に示す。4本で組になっている棒グラフはidealを除く4つのモデルであり、縦軸はメモリ依存違反削減率である。図5は2プロセッサ構成のSKYについて、図6は4プロセッサ構成のSKYについての結果である。idealモデルは理想的なハードウェアを仮定しているので、メモリ依存違反は発生しないため図から省いた。ベンチマークごとに見ていくとm88ksim、liはメモリ依存違反が少ないため、同期機構による変化は少ない。

他のベンチマークについては同期機構別にみていくと、valモデルの結果からgcc、vortexでは同一アドレスに対して同一の値の書き込みが多いことがわかる。

waitモデルではすべてのベンチマークで大幅にメモリ依存違反を回避していることからカバレッジは高いといえる。

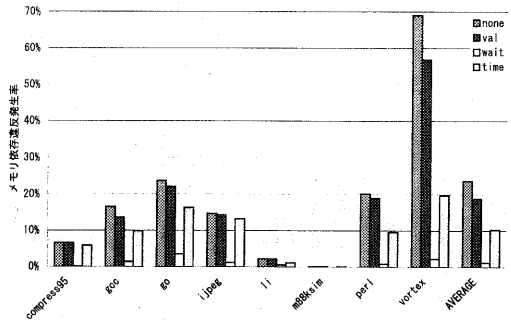

図5: 2プロセッサにおけるメモリ依存違反割合

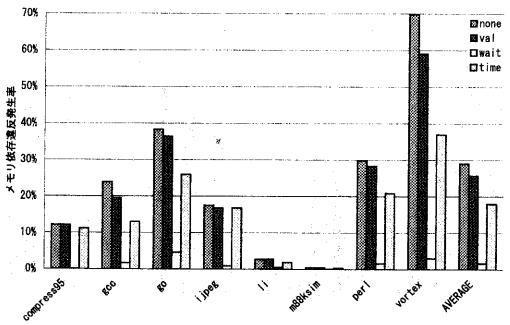

図6: 4プロセッサにおけるメモリ依存違反割合

timeモデルではgcc、perl、vortexでメモリ依存違反を回避でき、特にvortexでは2プロセッサの場合で72%、4プロセッサの場合で47%のメモリ依存違反を回避できた。しかし、compress95、ijpegではほとんどの違反を削減できていない。

5.3 性能低下率

メモリ依存違反が削減されたことによりプロセッサの性能低下がどの程度抑えられたかを調査した。図7、8は各ベンチマークごとに5本の組になっており5種類のモデルで、縦軸は1プロセッサからの性能向上率となっている。図7は2プロセッサ構成のSKYについて、図8は4プロセッサ構成のSKYについての結果である。

idealが従来の理想的にメモリ同期を行う機構を仮定したときの性能である。ベンチマークごとに見ていくとm88ksimはメモリ依存違反がほとんど発生しないため、同期機構による影響はほとんどない。liはTLPを引き出せていないため、同期機構による影響はほとんどない。

waitモデルでは確実に同期を行うため、メモリ依存違反を大幅に削減できているが、同期待ちによるペナルティのために性能低下が激しい。特にvortexでは1プロセッサの性能を下回るほど性能が低下した。

timeモデルではgcc、perl、vortexでメモリ依存違反を削減できた。特にメモリ依存違反の多かったvortexに

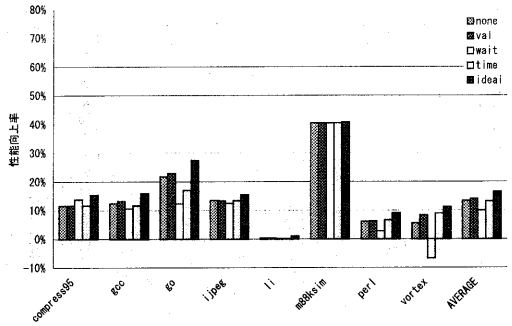

図 7: 2 プロセッサにおける性能向上率

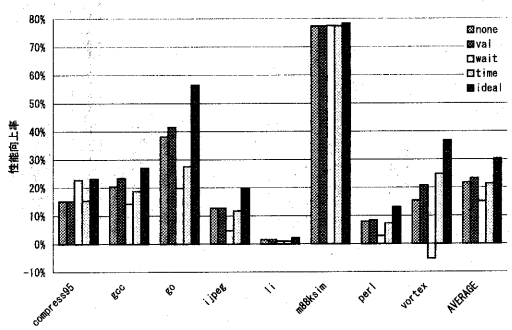

図 8: 4 プロセッサにおける性能向上率

ついては 2 プロセッサで 72%、4 プロセッサでも 47% 削減できた。これにより性能低下も 2 プロセッサで 2%ポイント、4 プロセッサでも 12%ポイントに抑えることができた。

しかし、vortex 以外では性能低下を抑えることができていない。これは短い待ち時間でよいロード命令に対して長く同期待ちをしてしまったために、性能が低下してしまったものと思われる。

6 まとめ

マルチスレッドアーキテクチャではメモリ依存違反が発生した場合のペナルティが大きいため、メモリアクセス違反を防ぐための同期機構が必要であることを示した。同期が行われなかった場合のスレッドが破棄される状況について詳細な調査を行った結果、スレッド破棄を引き起こすロード命令は 300 命令程度で、30 サイクル程度同期待ちを行うことで、スレッド破棄が回避できることが分かった。しかし、通常のスーパースカラプロセッサと違い簡単なハードウェアでは効率よく同期を行うことは難しい。提案した time モデルでは最大で 72%(2 プロセッサ)、47%(4 プロセッサ) のメモリ依存違反を削減し、性能低下はそれぞれ 2%ポイント、12%ポイントに抑えることができた。

謝辞

本研究の一部は、文部省科学研究費補助金基盤研究 (C)(課題番号 11680351) 及び財団法人大川情報通信基金の支援により行った。

参考文献

[1] H. Akkray and M. A. Driscoll, "A Dynamic Multithreading Processor," In *Proc. MICRO-31*, pp.226-236, Nov. 1998.

[2] 木村 啓二ほか, "近細粒度並列処理用シングルチップマルチプロセッサにおけるプロセッサコアの評価," 情報処理学会論文誌, Vol.42, No.4, pp.692-703, 2001 年 4 月.

[3] 小林良太郎ほか, "非数値計算応用向けスレッド・レベル並列処理マルチプロセッサ・アーキテクチャ SKY," 情報処理学会論文誌, Vol.42, No.2, pp.349-366, 2001 年 9 月.

[4] 岩田充晃ほか, "制御等価を利用したスレッド分割技法," 情報処理学会研究報告 98-ARC-128, pp.127-132, 1998 年 3 月.

[5] 川梅慶紀ほか, "単一チップマルチプロセッサ・アーキテクチャ SKY におけるスレッド分割技法の評価," 情報処理学会研究報告 2002-ARC-146, 2002 年 2 月.

[6] S. W. Keckler, et al., "Exploiting Fine-Grain Thread Level Parallelism on the MIT Multi-ALU Processor," In *Proc. ISCA-25*, pp.306-317, June 1998.

[7] G. S. Sohi, et al., "Multiscalar Processor," In *Proc. ISCA-20*, pp.414-425, June 1995.

[8] 鳥居淳ほか, "オンチップ制御並列プロセッサ MUSCAT の提案," 情報処理学会論文誌, Vol.39, No.6, pp.1622-1631, 1998 年 6 月.

[9] K. Olukotun, et al., "The Case for a Single-Chip Multiprocessor," In *Proc. ASPLOS-VII*, pp.2-11, Oct. 1996.

[10] K. Sundaramoorthy, et al., "Slipstream Processors: Improving both Performance and Fault Tolerance," In *Proc. ASPLOS-IX*, Nov. 2000.

[11] M. Franklin, et al., "ARB: A hardware mechanism for dynamic reordering of memory references," *IEEE Transactions on Computers*, vol.45, no.5, pp.552-571, May 1996.

[12] S. Gopal, et al., "Speculative Versioning Cache," In *Proc. Fourth Int. Symp. on High-Performancce Computer Architecture*, Feb. 1998.

[13] L. Hammond, et al., "Data Speculation Support for a Chip Multiprocessor," In *Proc. ASPLOS-VIII*, pp.58-69, Oct. 1998.

[14] R. E. Kessler, et al., "The Alpha 21264 Microprocessor Architecture," *IEEE Micro.*, Vol.19, No.2, pp.24-36, Mar. 1999.

[15] A. Moshovos, et al., "Dynamic Speculation and Synchronization of Data Dependences," In *Proc. ISCA-24*, pp.181-193, May 1997.