複数パスプロセッサにおける複製法を用いたキャッシュシステムの提案

高野直樹[†] 坂井修一[†] 田中英彦[†]

プロセッサの処理能力の向上と比較して、主記憶の DRAM はデータの供給能力が不十分であることをメモリウォールと呼ぶ。このメモリウォールを改善するためにプロセッサと DRAM の間に Cache を設けることで性能の改善を行って来た。本稿では次世代プロセッサとして研究されている複数パスプロセッサに適した Cache システムを考察し、複製法を用いた手法を提案する。また本研究室で研究されている複数パスプロセッサである VLDP に対して、実装する Cache システムについても考案する。

Cache System Using Replicated Memory for Multi-path Processor

Naoki Takano,[†] Shuichi Sakai[†] and Hidehiko Tanaka[†]

It is called “Memory Wall” that DRAM used in main memory does not supply enough data to processor compared with its processing improvement. Cache memory which is implemented betwean processor and DRAM has improved memory throughput. In this paper we consider suitable cache system for multi-path processor studied and propose new cache system applied with replicated cache. We also design cache system for VLDP(Very Large Data Path) processor studied in our lab.

1. はじめに

プロセッサの処理性能向上と比較して、主記憶に使われている DRAM のアクセススピードは大きな性能差が現れる様になった。このプロセッサと DRAM の性能差をメモリーウォールと呼ぶ。今後このメモリーウォールの問題は更に大きくなると考えられている。

近年新しいアーキテクチャを使った Direct Rambus DRAM[7] や、メモリのアクセスを DRAM に適した順番に並べ変える Memory Scheduling[6] と言った主記憶のアクセス性能の向上を計るための開発や研究が盛んになされている。一方でデータの空間的時間的局所性に着目し、DRAM とプロセッサの中間に小量だが高速な SRAM を Cache メモリとして用いてメモリウォールを解決する手法の研究も盛んになされてきた。

今後プロセッサの 1cycle あたりに実行された命令数 (IPC☆) が向上するに従って、今まで以上に十分なデータを供給できる Cache の研究が不可欠である。

本稿では次世代プロセッサとして現在研究されている複数パスプロセッサのための Cache システムについての提案を行う。まず始めに複数パス実行プロセッサを説明し、その 1 つの例として本研究室で研究されている VLDP アーキテクチャの説明を行う。次に従来の Cache システムの手法の説明を行い、複数パス実行プロセッサにおけるポート数の必要性について考察する。同時に予備評価を行い定量的な検証も行う。これらを踏まえて複製法の Cache を使った時の Lazy Store 法を用いた方法を提案する。具体例として VLDP を使った実装例をあげる。

† 東京大学大学院 工学系研究科
Graduate school of Engineering, The University of Tokyo

☆ Instrucion Per Cycle の略

2. 複数パス実行プロセッサ

現在使われている Super Scalar プロセッサでは、分岐予測を行い単一パスの投機的実行を行っているが、3〜4 IPC が性能の限界と言われている。更に IPC の向上をはかるため、分岐予測ミス時のペナルティーを削減を行い分岐成立と不成立の投機的な両パスの実行を行う必要がある。これが複数パス実行プロセッサである。

複数パスプロセッサとしてここでは本研究室で研究されている VLDP(Very Large Data Path) プロセッサを例として取り上げる。

2.1 VLDP の概要

VLDP は CS(Control Section), EU(Execution Section), MS(Memory Section) の 3 つに分割されている [4]。

CS では命令のフェッチを行とともに複数パス実行時

のパスの無効か無効かの管理を行う [4]。VLDP においては命令は IB というブロック単位で処理される。IB は BB(Basic Block) を 4 つ繋げたものであり、1BB につき最大で 8 命令をつめるか、もしくは最後が分岐命令でなければならない。すなわち IB は最大で 32 命令を詰めることが可能である。CS はこの IB 単位で ES に対して命令を供給する。

ES は CS から IB を受け取り実行する [5]。ES 内は 16 個の EU が存在し、1EU に対して 1IB が割り当てられ実行される。レジスタは各 EU に分散して共有しており、高速なネットワークによってデータの受渡しを行う。ES は CS に対して命令の実行の完了を通知する。また MS に対してデータの Load/Store 要求を行い、複数パス実行により生じた依存性の解析を MS に要求する。

MS においては ES から来たデータの要求の依存性解析を行う [3]。Load の要求は各パスごとに依存性を解析し、データを ES に返す。Store の要求は MS に投げられた後、その Store が In order 状態になり有効だと判断された場合は Cache に対しての書き込みが行われる。Cache へはレジスタの通信に使われたネットワークを通して行う。

2.1.1 メモリの依存性の問題

VLDP ではメモリの依存性解析は MS で行われる。メモリの依存性解析の例を図 1 にあげる。

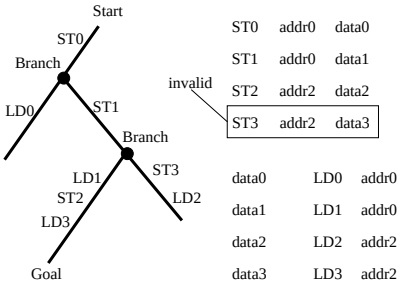

図 1 複数パス実行時のメモリの依存性

ST0 と LD0 は同じアドレスであり、同じパス上にあるので、ST0 での書き込みがそのまま適用される。一方 LD1 と ST1 は違うパスであり、VLDP においては別々の EU で実行される。しかし、LD1 に置いては ST1 で書き込まれた値がそのまま使われなければならない。また ST2 と ST3 は同じアドレスに別々のパスにおいて書き込みが行われているが、LD3 には ST2 で書き込まれた値を、LD2 には ST3 で書き込まれた値を EU に返す必要がある。

VLDP ではメモリの依存性の解析を行うために、各 EU 毎にロードストアキューを持たせ仮想的に上記のツリー構造の条件を満たすものになっている。(図 2)

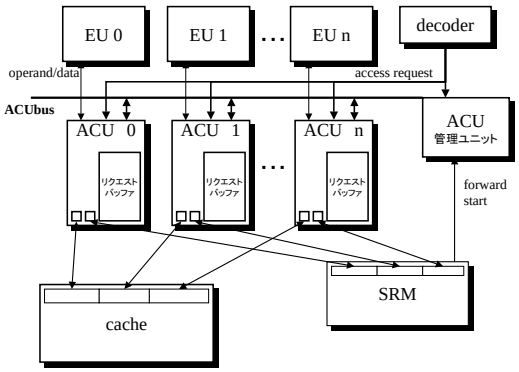

図 2 VLDP におけるロードストアユニットの構成

各 EU 内のメモリの依存性の解析は ACU 内で行われるようになっており、ACU 間をまたがった Load/Store の順序解析は ACU 管理ユニットで行う。また SRM(Store Reservation Unit) では RAW(Read After Write) の解析を行い、ACU 間のデータのフォワーディングを行えるようになっている。このため Cache へは余分なメモリのリクエストが飛ばないようになっている。[3]

以上のように VLDP の MS において正しい Load/Store 命令の実行を保証されており、Cache では要求された順番にメモリに対してデータを反映する必要がある。

3. 複数パス実行プロセッサでの Cache のポート数の考察

Cache が 1cycle で同時にデータの読み書きが行える数をポート数と呼ぶ。ここでは複数パス実行プロセッサにおける Cache メモリをポート数の観点から考える。

3.1 ポート数増加の必要性

複数パス実行は単一パス実行と比較して、無駄なパスの命令も投機的に実行されるので、実行される命令数は多くなる。当然 1cycle あたりの Load/Store の数もマルチパス実行によって増えることが考えられる。

Cache へのアクセスを考えると Store は実行のパスが確定しない限り、すなわち Store 命令が In order 状態にならない限り Cache への書き込みは起らない。プログラムはそれぞれ一定の割合で In Order になる Store が含まれているので、同じプログラムを実行すれば、実効 IPC に比例した Cache への Store による書き込みが起ると考えられる。

一方 Load は投機的に実行されたパスからも要求されるので、Load/Store キュー内にそのデータが存在すれば良いが、見つからないと Cache へ読み込みの

要求が送られる。Load/Store キューの大きさや複数パス実行のアルゴリズムによるが、実行 IPC に比例した数以上に Cache への読み込みの要求が増える可能性がある。つまり従来の Super Scalar プロセッサよりも 1cycle で同時に Cache への読み込み要求が増えることが考えられる。

以上のように複数パス実行プロセッサにおいては Load/Store ともに従来の Super Scalar で用いられてきた Cache よりもポート数を増やす必要性が考えられる。

3.2 従来のポート数の増加手法

SRAM を理想的に多ポート化するとポート数の自乗分の実装コストがかかる。そのため従来の Cache の多ポート化には 3 つの手法が取られてきた。1 つは時分割法、1 つは複製法、そして Bank 分割法である。

時分割法は Cache に使われている SRAM のクロックをプロセッサの複数倍率で駆動することによりポート数を増やす手法である。しかしこの方法で増やせるのはせいぜい 2〜3 ポートが限界であり、スケーラビリティに欠けるので、多ポート化には向かない。

複製法は同じ内容の Cache をポート数だけ用意し、読み込みのポート数を増やす手法である。これは実装コストがポート数倍かかるのと、書き込みの際一貫性を保つため書き込みのポート数は 1 のままであるのが欠点である。(図 3) Cache への書き込みは Store するときにも生じるが、Cache ミスが生じたとき、Cache より下層のメモリからデータ読み込み、それを書き込む必要が生じる。これを Cache Fill と呼ぶ。Cache Fill についても同様に Cache メモリの一乾性を保必要があるので、大きなコストになる。

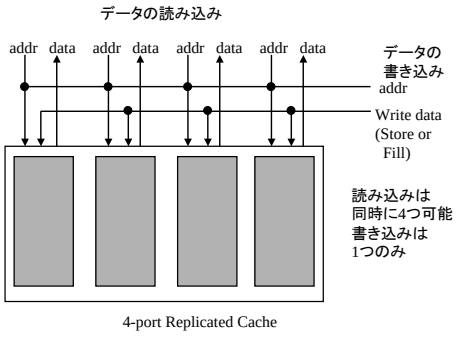

図 3 複製法を使った Cache システム

Bank 分割法は SRAM を複数 Bank に分割し、アドレスによって分散配置を行うことでポート数を増やす方法である。しかしこの方法は Bank 化された Cache と Load/Store ユニットを接続するため Crossbar 等の相互結合網が必要である。また同じ Bank への衝突が起る可能性がある。これを Bank conflict と呼ぶ。(図 4)

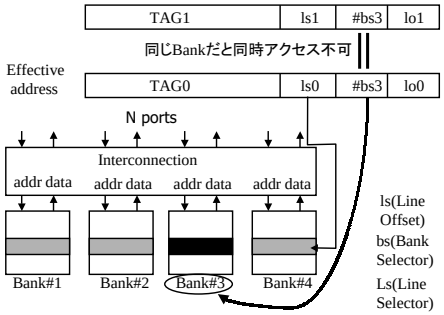

図 4 バンク分割法を使った Cache システム

現在の半導体の集積度の問題から考えると、複製法よりも Bank 分割法の方がトランジスタを有効に利用できるため、利点が大きいように思われる。しかし今後 10 年間の半導体テクノロジー予測 [2] を参照すると、将来 Cache に使えるトランジスタ数は数億個オーダーになり、Bank 分割が必ずしも有利な手法とは言えなくなる。

3.3 予 備 評 価

複数パス実行を行った時に、Cache へのメモリアクセスは同時にいくつのアクセスが起こるかについて、予備評価を行った。複数パス実行を行うためのシミュレーションは、命令のフェッチの手法が異なるだけでも大きくメモリ要求のパターンについても異なると考えられる。ここでは疑似的に複数パス実行のシミュレーションを行い、データのアクセスパターンの統計的な予測をたてることにする。

3.4 評価モデル

予備評価のモデルではいくつかの理想化された条件を用いることにする。理想化される点は以下のようになっている。

- 命令レベルで 100%ヒットのメインパスを常に展開
- サブパスをすべての Branch で展開
- 理想的な Load/Store Queue を持つ
- Cache への Store はメインパスからのみ
- 間違ったアドレスを参照した場合はサブパスはそれ以上展開しない

メインパスは Alpha のプログラムを用いてトレースを取ったものをそのまま実行するようにする。評価においては SPEC98 の整数演算系のアプリケーションを用いた。

サブパスを展開は現在展開されているパスの数が 16 以内に収まるようにする。これは VLDP では EU が 16 個実装されることを考えての値である。また各パスは 1 サイクルで命令を終えることにする。各サブパスは 4 つの Branch が現われた時点で終了するように

する。

各パスにおいては理想的な Load/Store Queue ではデータの依存性の解析が行われるが、Queue で吸収された Load は Cache にはアクセスが起こらないようにする。評価モデルを図 5 に示す。

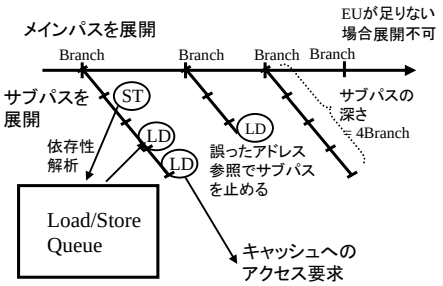

図 5 予備評価モデル

3.5 評価結果

図 6 に Cache に対してのデータの要求の数を示す。また Cache への Load/Store の要求の数と、Queue からの読み込みの数の比較を図 7 に示す。

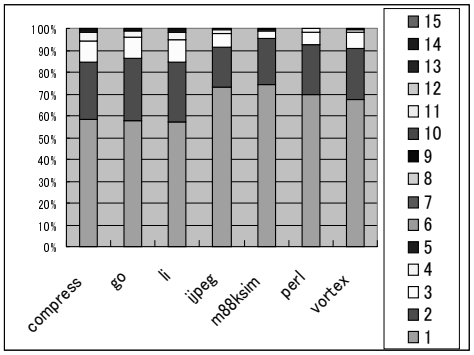

図 6 Cache へのデータの要求

図 7 からわかるように、Store と比較した場合 Load の数は 1 桁ほど多いことがわかる。これは複数パス実行では失敗したパスからであっても Load の要求が Cache に対して行われる可能性があり、一方で Store はインオーダー状態にならないと Cache への書き込みが起こらないためである。また Load 要求の 2/3 が Load/Store Queue で吸収されることがわかる。Load/Store Queue は Cache への同時アクセスの増大を防ぐ役割があることがわかる。一方で図 7 では同時に Cache に対してのデータ要求は 4 以下で、9 割以上を占めていることがわかる。

以上のことから分かるように複数パス実行プロセッサにおいては Store に比較して Load の要求は非常に増えることがわかる。また Cache のポート数は最低でも 4 ポート以上必要であることがわかる。

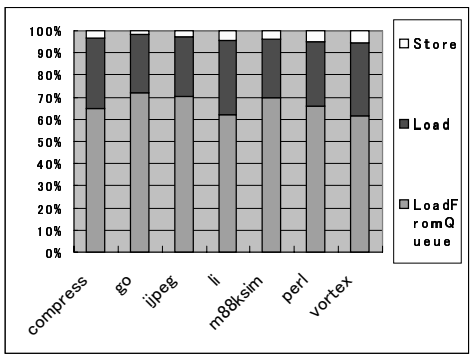

図 7 Cache への Load/Store と Queue からの読み込み

4. 複数パス実行プロセッサ向けの Cache の提案

予備評価を踏まえた上で複数パス実行プロセッサ向けの Cache 機構の提案を行う。

4.1 複製法の Cache を使った Lazy Store

ここで、複製法の Cache に対して新しく Store 専用の Buffer を設けることで、遅延書き込みを行えるようにした Cache を新たに提案する。この Cache を図 8 に示す。

Bank 分割法より複製法を選んだのは Bank 衝突を起こさず効率良く Load 要求に答えるためである。過去の研究 [1] においては Bank 分割法をベースにした手法が提案されていたが、この手法では Load/Store ともに Latency が増加する可能性があり、異なったラインの Bank 衝突が起るのは回避できない。また複数パス実行プロセッサにおいては、Super Scalar プロセッサと比較して Store より Load の方が数が増えることを想定して、Load と親和性が高い複製法を選択する。

本手法は複製法における Cache への書き込みは 1 ポートしか行えない欠点を補うため、Store の際起る書き込みを Lazy に行うことで、Store の書き込みのポート数を仮想的に増やす。

Store データは Cache に書き込みの要求を行った後でもしばらくの間 Load/Store キューに留まっており、その間は Cache にアクセスする必要はない。そこでポートが塞がっている時は、Store 要求が出されても一度に Cache に書き込みを行わずに、一時的に Lazy Store Buffer に保存する。Cache は常に全てのポートを占有されている訳ではなく、一時的に Load/Store ユニットからの要求が来ない時があるので、空いた時間を使って Lazy Store Buffer にある古いデータから

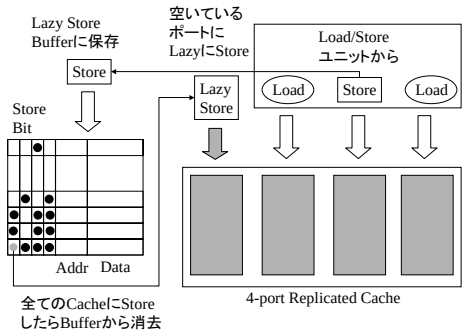

図 8 Lazy Store を使った際の多ポート Cache

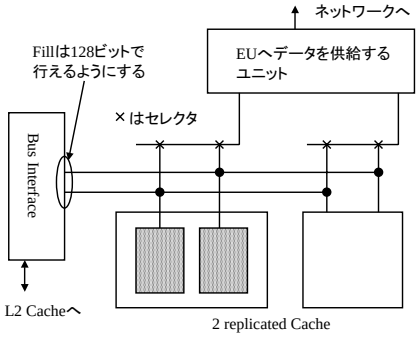

図 9 fill を効率良く行うための複製法 Cache

Store を行う。Store の実行後その Cache の Stored Bit をセットし、どの Cache にすでに書き込まれたかを保存する。Stored Bit 全てがセットされたらそのエントリは Lazy Store Buffer から消去する。Load/Store キューから Store データのエントリが削除される時にまだ Lazy Store Buffer に残っている時は、強制的に Cache に対して書き込みを行うことで一貫性を保つ。

本手法は Load/Store キューからデータが追い出されるまでの時間を使うので、Load/Store キューのエントリ数が小さい時には Lazy に Store を行う時間が少なくなってしまい効果は低下する。

4.2 Cache Fill の問題の改善

Cache Fill が発生すると Cache の 1 ライン分の書き込みが必要なため、データバスと Cache のポートが完全に専有される。一般的に Cache の 1 ラインは 32 バイト〜64 バイト程度とされている。たとえば Alpha 21264 の場合は 2 ポートを同時に使って Cache Fill を行うが、4 サイクル☆かかる。Cache ミスが増加すると Cache Fill により Cache 全体の性能低下も大きくなることが考えられる。

複製法においては Cache メモリに対しての書き込みは、それぞれの複製されたメモリは一貫性を保つ必要があるので、Cache Fill は大きなコストになる。

ここでとる解決手法としては、Cache Fill においてはデータを一度に 1 ライン単位で書き込めるようにすれば良い。具体的にはプロセッサに対しては Load/Store のバスは 64 ビットで構成されていても、下位のメモリからはその倍の 128 ビットでつながっているような構成を考える。(図 9)

ブロック分割と同様にスイッチによってデータの供給ユニットで接続されることになるが、Load 時の Bank Conflict が起こらない。図 9 の例では 2 つの複製された Cache をつかい、読み込みは 2 ポート、書き込みに関しては 1 ポートだが一度に 128 ビット分の書き込みが一度に行えるようになっている。このような手法を使うことで Cache Filll にかかるサイクル数を削減することができる。

4.3 VLDP における Cache モデルの提案

VLDP に置いてはメモリへのアクセスはネットワークを通る設計になっている。このネットワークを使うことで Bank 分割された Cache に対してのアクセスが行えるようになる。EU からキャッシュに対してデータの要求が起こる時、アドレスからどのブロックに要求を行えば良いかがわかるので、VLDP においてはバンク分割をする手法も採り入れる利点がある。(図 10

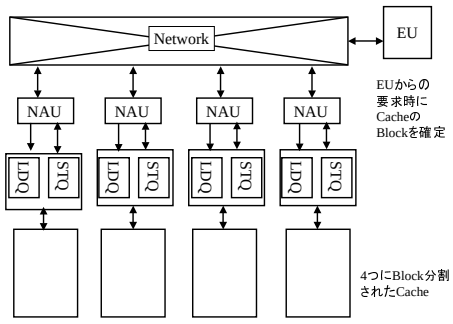

図 10 Block 分割された Cache

各ブロックにわけられた Cache はメモリ NAU(Network Access Unit) を通して Load/Store Buffer に送られて、Cache への読み書きが行われる。Load/Store Buffer に保存されるデータのアクセスのエントリはキャッシュへのアクセスが終了するまで保持される。

VLDP において EU から見た Cache への書き込みは 1 サイクルで終ることになっているので、Store の終了の完了の通知は EU に対して行う必要はない。

一方で Cache は各ブロックごとに分割されているので Conflict ミスの起こる原因になることも考えられる。しかし 1Block あたりのポート数を複製法を用

☆ 正確には 2 サイクルのあとの 1 サイクルのウェイトが入り、完了までには 5 サイクルかかる

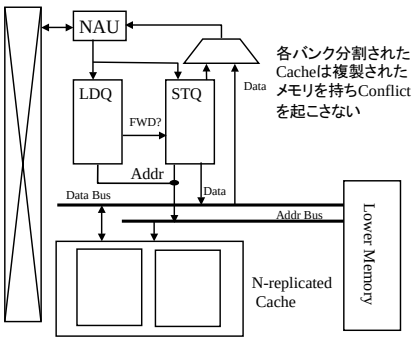

図 11 1Block の拡大図

いることで、Load のポート数を増やすことは可能である。

また Load/Store Buffer が Cache の前段にあるため、ここで Store Buffer からのフォワーディングを行う。また Store Buffer は先に紹介した Lazy Store の手法を用いて複製された Cache メモリへの書き込みのデータの一慣性を保つ (図 11)。

5. まとめと今後の課題

本稿ではメモリーウォールの問題を解決するために、Cache システムの重要性について述べた。また次世代プロセッサとして本研究室で研究されている VLDP について述べ、複数パス実行プロセッサに適した Cache システムについての検討を行った。

予備実験により複数パスプロセッサにおいては Store よりも Load の要求が急激に増加することを示した。これを参考にし複製法を用いた Lazy Store の手法を紹介した。また実際に VLDP に付加するための Cache System についても述べた。

今後は VLDP のシミュレータを用いて Lazy Store や複数法の Cache を用いた場合、具体的な性能の測定を行いたいと考えている。

謝　　辞

本研究の一部は、文部省科学研究費補助金 (基盤研究 (B) 課題番号 11480066) および、(株) 半導体理工学研究センターとの共同研究によるものである。

参 考 文 献

1) Jude A. Rivers, Gary S. Tyson, Edward S. Davidson and Todd M. Austin, “On high-bandwidth data cache design for multi-issue processors”, *International Symposium on Microarchitecture*, Research Triangle Pk United States. pp.46–56, December 1-3, 1997

2) Semiconductor Industry Association, International Technology Roadmap for Semiconductors 1999 Edition. `http://www.itrs.net/ntrs/publntrs.nsf`

3) 入江英嗣、安島雄一朗、辻 秀典、「大規模データパス・アーキテクチャに適したロードストアユニット構成」、 情報処理学会研究報告 計算機アーキテクチャ研究会 、於 ラフォーレ琵琶湖、*Vol2000,No.110, pp.43–48, November 2000*

4) 辻秀典、安島雄一郎、坂井修一、田中英彦、「大規模データパス・アーキテクチャの提案」、 情報処理学会研究報告、*2000–ARC–139* 、於 松山市総合コミュニティーセンター、*Vol.2000, No.74, pp. 55–60, August 2000*

5) 安島雄一郎、辻 秀典、坂井修一、田中英彦、「大規模データパス・アーキテクチャの実行機構」、 情報処理学会研究報告、*2000–ARC–139* 、於 松山市総合コミュニティーセンター、*Vol.2000, No.74, pp. 55–60, August 2000*

6) Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens, “Memory Access Scheduling”, The 27th Annual International Symposium on Computer architecture, Vancouver, Canada, pp.128–138, June 10–14, 2000

7) CRISP, RICHARD, “Direct Rambus Technology: The New Main Memory Standard.”, IEEE Micro (November/December 1997),pp.18–28.