修 士 論 文

マルチスレッド型プロセッサ向きの キャッシュ機構のパイプライン化に関する研究

指導教官　日比野靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

相原孝一

1997 年 2 月 14 日

要 旨

本論文では、マルチスレッド型パイプラインプロセッサの下で、その効果を有効に発揮させるべくキャッシュ機構のパイプライン化を提案する。まず、キャッシュ機構のパイ プライン化によりクロックサイクルを短縮する方法について検討する。これらの方策によってパイプライン化したキャッシュのスループットが保たれることを示す。次に、キャッシュの大容量化についてメモリ・アレイの分割とパイプライン化を検討する。

次に、マルチスレッド型プロセッサのキャッシュの問題であるヒット率に関し、スレッド数に見合うキャッシュ容量を与えればヒット率は十分に維持できることを示す。

最後に、極めて高いスループットのキャッシュからレイテンシの大きな主メモリへのアクセス要求に対する、キャッシュ-主メモリインタフェースの構成について調べ、キャッシュからのメモリ要求頻度に見合う主メモリスループットを与えることで、システム全体としてスループットの低下が低く抑えられることを示す。

目 次

第 1 章

はじめに

集積回路の微細加工技術の進歩により、マイクロプロセッサは高性能かつ低価格で市場に出回っており、今後さらなるプロセッサの高性能化の要求が高まっている。マシンサイクルの短縮をねらうため、RISC(Reduced Instruction Set Computer) が発展しているが、配線遅延の影響で速度向上が飽和してきている [2]。それを克服するため、プロセッサにおいて高度のパイプライン処理を行いスループットの向上を図ることが盛んに行われている [6] [8]。

しかし、スーパーパイプラインのようにプロセッサのパイプラインの段数がある程度にまでなると、プロセッサのクロックサイクルはキャッシュの操作に依存することになる。また、パイプラインの段数が増すと命令間の依存関係によるハザードの発生により、命令スループットは向上しなくなる。

これに対して、独立な命令ストリームを処理するマルチスレッド型パイプラインプロセッサが提案されている [9]。

マルチスレッド型パイプラインプロセッサでは独立な命令ストリームを処理するので、命令間の依存関係はなく、原理的にはいくらでも深いパイプラインにすることができる。しかし、メモリで構成されるキャッシュは、アクセスに時間がかかるため、パイプラインのボトルネックとなる。

しかし、キャッシュのアクセス時間を短くすることはできなくても、キャッシュのスループットの向上、すなわち、キャッシュのサイクル時間を短縮できれば、深いパイプラインを実現することができる。

よって、まずキャッシュ機構のパイプライン化を提案する。同時にメモリアクセス速度

の向上を図るためにキャッシュのメモリアレイの分割を提案する。また、マルチスレッド型プロセッサに適したメモリ構成の方式、およびキャッシュへの書き込み方式も示す。

第 2 章

キャッシュ機構のパイプライン化

本章では、キャッシュの動作を示し、キャッシュ機構にパイプライン化を適用することがキャッシュメモリのスループット向上に有効であることを示す。まず、キャッシュメモリ部の構成を示し、その中で性能上ネックとなっている部分を指摘する。次にネックの 1 つとなっているデコーダ部分に着目し、この部分をパイプライン化することを提案し、スループットの向上をねらう。また、サイクルタイムを制限しているセルアレイの読み出しに関して、セルアレイ分割によるアクセス時間の短縮が有効であることを示す。この両者を組み合わせることにより、高性能化に対するキャッシュ機構の設計方針を示す。

2.1 キャッシュ機構のパイプライン化

プロセッサの高速化の要求によって、プロセッサのパイプライン処理が行われるようになった。パイプライン処理は複数の命令を数段の処理に分割し少しずつずらして同時並行的に実行する実現方式である。これにより命令のスループットを向上させている。本研究ではこのパイプライン処理をキャッシュ機構に適用することによってキャッシュのスループットの向上をねらう。

ここで、具体的にキャッシュの機構を考えてみる (図 2.1)。

キャッシュの操作は大まかに次のような順で行われる。

1. アドレスデコード : アドレスのインデックス部に基づき対応するキャッシュのブロックを選択する。

2. メモリセルアレイの読みだし : デコードされた結果から実際にメモリセルアレイの読み出しを開始する。

3. 判定 : アドレスのタグ部とキャッシュのタグを比較し、そのキャッシュブロック中の語が要求されたアドレスのものであるかどうかを判定する。

4. データ排出 : 対応する語をキャッシュから出力する。

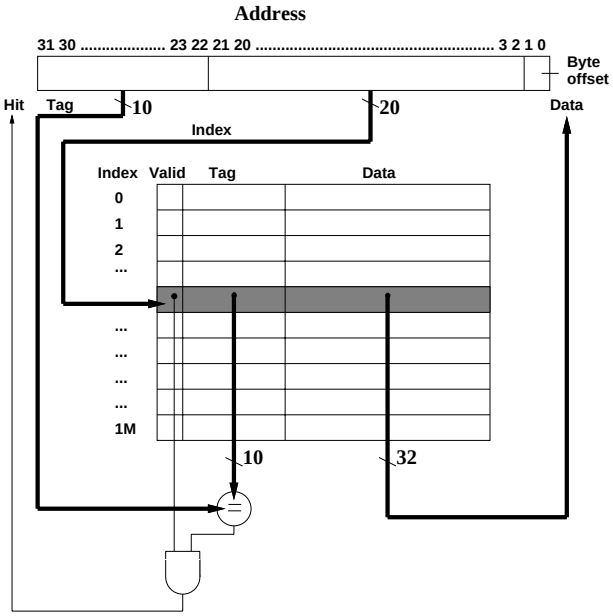


図 2.1 キャッシュの機構

例えば、上の 4 つの操作の動作時間を次のように仮定する。

- アドレスのデコード : 6ns
- メモリセルアレイの読み出し : 4ns
- 判定 : 2ns
- データ排出 : 2ns

上の 4 つの操作に必要な時間は 14ns となる。

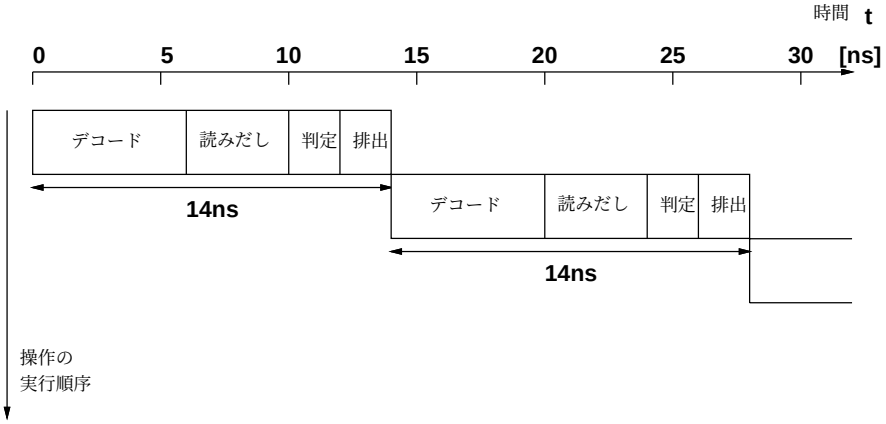


図 2.2 キャッシュの操作を直列的に実行した場合

次に 4 ステージからなるパイプラインを検討してみる。各パイプラインステージでは 1 クロックサイクルで処理を行う。パイプライン方式で連続して実行する状況を図 2.3 に図示する。

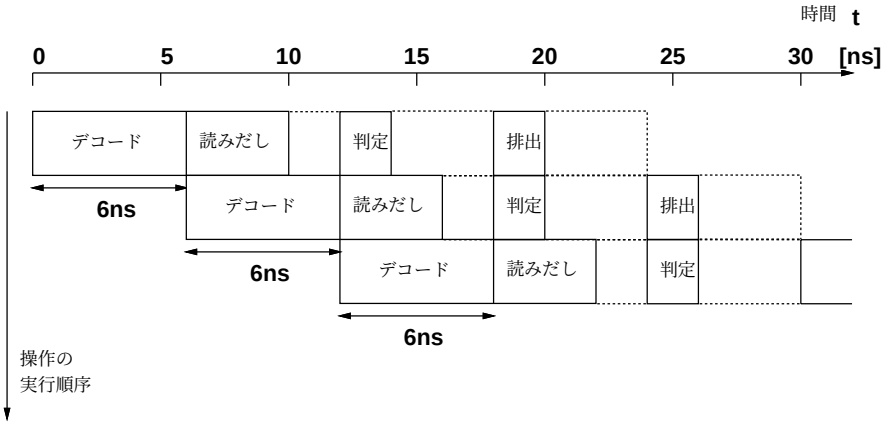


図 2.3 キャッシュの操作をパイプライン方式で実行した場合

クロックサイクルはもっとも遅い操作に合わせなければならないので、キャッシュの操作を直列的に実行した場合のクロックサイクルは 14ns、パイプライン方式で実行した場合のクロックサイクルは 6ns となる。この例での速度向上比は約 2.3 倍程度となる。

これらを考慮すると、上の例ではキャッシュ操作の判定とデータ排出が 2ns であるにもかかわらず、6ns ともっとも遅いアドレスのでコードにクロックサイクルを合わせなければならない。アドレスデコードのステージをさらに分割することができるならばクロックサイクルを短縮することができる。ここで、アドレスデコードのステージが 2 つに分割できると仮定して、ステージにかかる動作時間がどちらも 3ns であるとすると、キャッシュ操作の進行は図 2.4 のようになる。

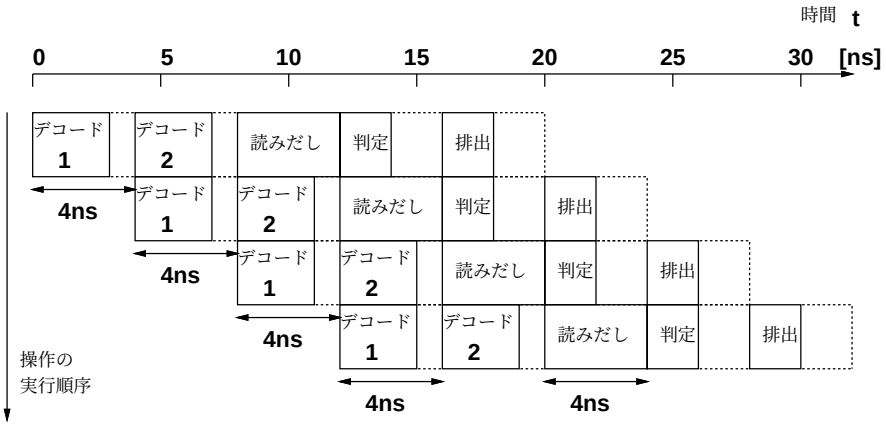


図 2.4 アドレスデコードを 2 ステージに分割した場合

この場合、パイプライン化していないキャッシュ機構との速度向上比は 3.5 倍に向上する。アドレスデコードの 2 ステージ化を行ったにもかかわらず、クロックサイクルは 3ns に短縮することができない。これはメモリセルアレイの読みだし時間が 4ns なのでクロックサイクルは 1 ステージでもっとも遅い読み出しの 4ns に合わせなければならない。このようにもっとも遅いステージの部分をさらに小さなステージに分割し、他のステージ間との処理時間のバランスをうまくとることによってクロックサイクルの短縮、すなわちスループットの向上をねらうことが可能である。

2.2　キャッシュメモリ部の構成

図 2.5 にキャッシュメモリ部の構成を示す。

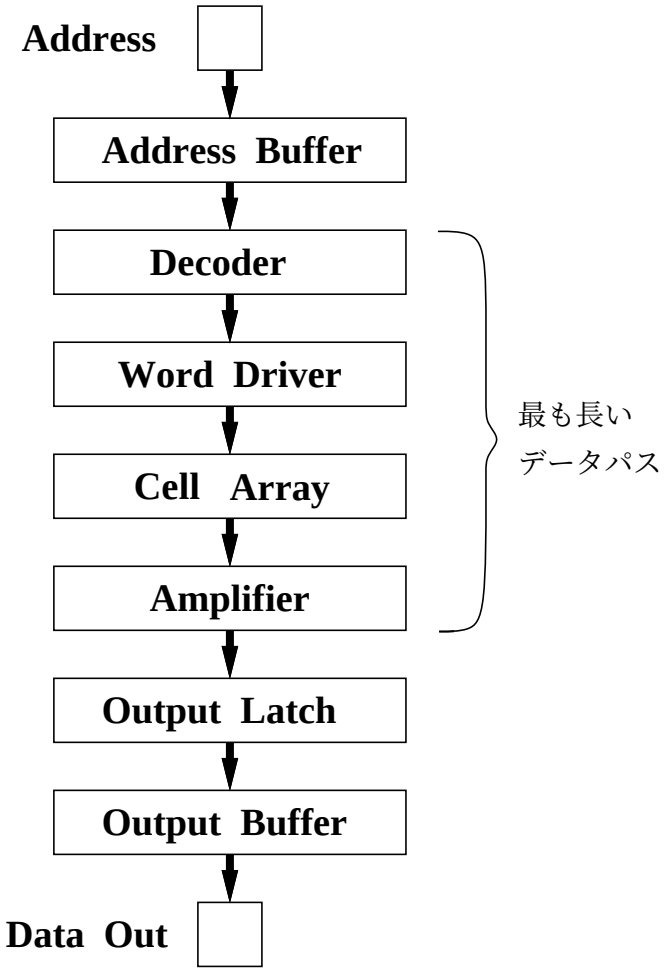


図 2.5 キャッシュメモリ部の構成

上図の中でのクリティカルパスはデコーダからセンスアンプの出口までである。図 2.5 の中で本研究で注目するところはデコーダ部とメモリセルアレイ部である。デコーダ部においてはデコーダ部で行う処理を数段の処理に分割しパイプライン化を適用することを提案する。これによりデコーダ部でのスループットの向上をねらう。またメモリセルアレイ部においてはメモリセルアレイを分割することを提案する。これによりメモリアクセス速度の向上を図る。

2.3 階層的デコード法

プロセッサのスーパーパイプライン化によりクロックサイクルがキャッシュの操作に依存することになる。図2.5でも示したようにキャッシュ中の操作でも特に遅いものにデコーダがある。そこで、デコーダを数段の処理に分割しパイプライン化することによって階層的にデコードを行うことを検討する [1]。

具体的に検討するために、1Mb(1k×1k) のメモリセルアレイを 8 つのブロックに分割し語線 (行選択線) を 3 つのレベルに分けて行い、各レベルの論理段数を見積りパイプラインとの関係を検討した。図 2.6 に階層的デコードの流れを示す。表 2.1 に階層的デコードを行なわない場合の論理段数 (16k×64) を示し、表 2.2 に階層的デコードを行なった場合のパイプラインステージ間の論理段数を示す。

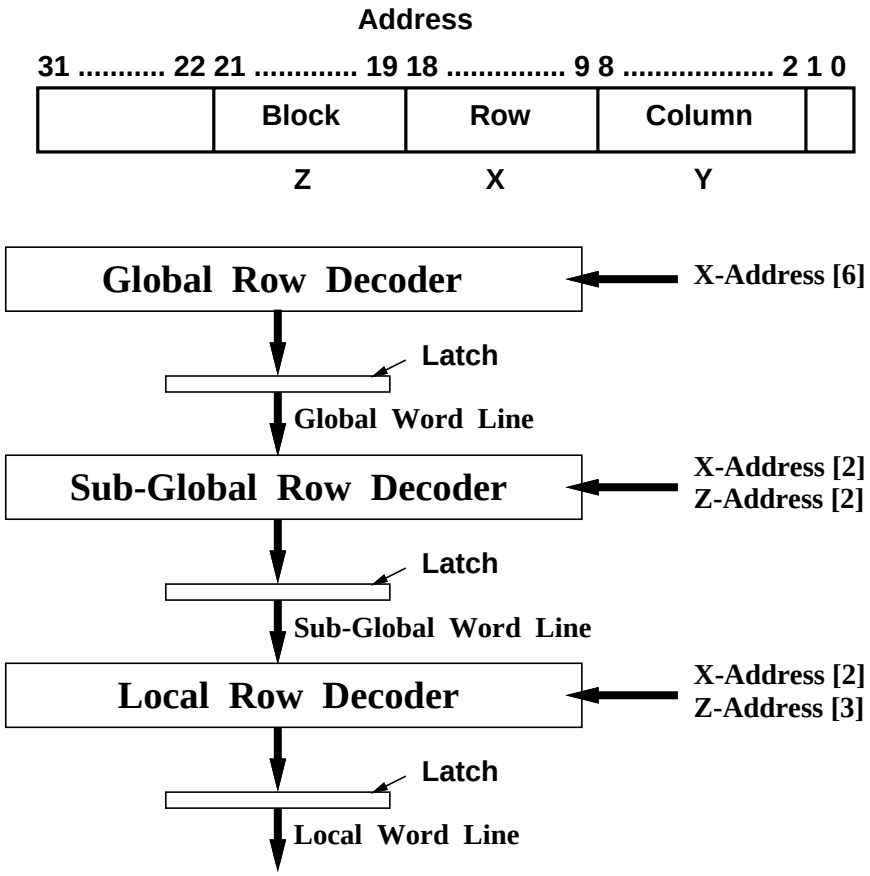


図 2.6 階層的デコードの流れ

表 2.1 パイプラインステージ間の論理段数

	Decoder	判定	データ排出
段数	11	7	4

表 2.2 階層的デコードを行なった場合のパイプラインステージ間の論理段数

	Global Row Decoder	Sub-Global Row Decoder	Local Row Decoder	判定	データ排出
段数	7	6	6	7	4

表 2.1 では、1Mb のメモリセルアレイにデコードを行う際、論理段数が 11 段かかってしまい、他のステージに比べてボトルネックとなっていることが分かる。ここで、デコーダを 3 ステージに分割したところ、キャッシュの判定部分やデータ排出のステージのディレイのバランスをうまくとることができた。クロックサイクルのさらなる短縮が要求された場合、メモリセルアレイの読みだし時間との兼ね合いからさらにデコーダの分割や判定部の分割を進めて行けばよい。

2.4 メモリセルアレイの分割

キャッシュメモリの容量が大きくなるにつれキャッシュのヒット率が向上するので、キャッシュメモリの大容量化は魅力である。しかし、それとは逆にメモリの大容量化が進むにつれメモリセルアレイの配線遅延が問題となってくる。ここで、集積回路の性質であるスケーリング則に基づき配線遅延について検討する。スケーリング係数を K とすると、MOS FET の回路パラメータのスケーリングは次のようになる。

表 2.3 MOS FET の回路パラメータのスケーリング

パラメータ	スケーリング比
容量	**1 / K**
回路あたりの遅延時間	**1 / K**
デバイス面積	**$1 / K^2$**
線抵抗	**K**
配線遅延時間	**1**

上のパラメータを見ても分かるように、デバイスのスケーリング K が大きくなるのに比例して回路あたりの遅延時間も短縮することが分かる。しかし配線遅延時間は容量と線抵抗の CR 積となるためスケーリングにかかわらず 1 である。

ここでメモリのアクセスを考えてみる。メモリのアクセスはデコードを行った後メモリセルアレイのアクセスを行う。デコード時間はゲートの遅延時間に支配されるので 1/K になる。一方、メモリセルアレイのアクセス時間は配線遅延時間に支配されるので、スケーリング比を小さくしてもアクセス時間の短縮は期待できない。

そこでメモリセルアレイを分割することを提案する。

各ブロックにデコーダを配置し階層的にデコードを行うことによって、メモリセルアレイの部分ではブロックを N 分割することによって配線容量は $1/\sqrt{N}$、配線抵抗は $1/\sqrt{N}$ となるので、配線遅延時間は 1/N に短縮することになる。

またデコードの部分においてはデコードとメモリセルのブロック選択が並列に行えるため、両者により大容量化したキャッシュへの高速なメモリアクセスが期待できる。クロックサイクルのさらなる短縮が要求された場合、デコーダをさらに階層的に行いメモリセルアレイもさらに分割していくことで対応できる。

2.5　2 章のまとめ

このように、本研究ではキャッシュメモリ部でのクルティカルパスであるデコーダおよびセルアレイの部分について検討を行ってきた。デコーダの部分ではパイプライン化のためにラッチが入るためレイテンシは増加するが、パイプライン化によってクロック周波数の短縮およびスループットの向上を図ることができる。またそれに応じてメモリセルアレイの分割をしていくことで、両者の調和をとりながら設計を行うことが可能である。

さらにメモリセルアレイの分割を行えば、外形寸法の縮小比率の二乗の割合で配線遅延時間が短縮されるので、キャッシュメモリの大容量化も可能になってくる。

キャッシュの大容量化はそのままヒット率の向上につながるので、ペナルティの大きいキャッシュミスを少なくすることができる。よってキャッシュ機構のパイプライン化はメモリアレイの分割と組み合わすことによって、クロックサイクルの短縮やスループットの向上とともに、メモリアクセスのペナルティが大きくなると考えられ、マルチスレッド型プロセッサに適した手法であるといえる。

第 3 章

キャッシュメモリの方式とメモリ構成

マルチスレッド型プロセッサからはキャッシュへのアクセスが各クロックサイクル毎に行われる。このため、各スレッド間でキャッシュブロックの競合が起こるのではないかという問題が懸念される。

本章では、まず、各スレッドから発生するキャッシュメモリへのアクセスに対してキャッシュのマッピング方式を具体的に検討する。

次に、マルチスレッド型プロセッサで発生するメモリアクセス要求での問題点を指摘して、具体的な解決手法を述べる。

3.1 キャッシュのマッピング方式とキャッシュ容量

3.1.1 スレッド専用キャッシュとスレッド共用キャッシュ

ここで、各スレッド毎に専用のキャッシュを持つ場合とすべてのスレッドが 1 つのキャッシュを共用する場合とを考えてみる。

図 3.1 に各スレッド毎に専用のキャッシュをもつ場合のキャッシュのマッピングの例を示す。キャッシュのブロックが各色で塗りつぶされたところは各スレッドによってマッピングが行われた場所である。当然のことながら各スレッドのマッピングは自身に用意されたキャッシュ内でのみ許される。スレッドのキャッシュへのマッピングにおいて、スレッド 2 のように、キャッシュメモリ空間を少ししか必要としない場合もあれば、逆にスレッド 1 のように、用意されたキャッシュメモリ空間では足りない場合もあり得る。後者の場

合は、同一スレッド内でのブロック競合を起こすことが考えられる。

次に図 3.2 にすべてのスレッドが 1 つのキャッシュを共用する場合のキャッシュのマッピングの例を示す。この場合、キャッシュの容量は各スレッドが専用に持っていたキャッシュサイズをスレッド数分持つ 1 つのキャッシュを用意し、各スレッドのマッピングはスレッド毎にオフセットを持たせる。こうすることによって、各スレッド間でキャッシュメモリ空間を有効的に利用することが可能となる。すると、図 3.1 に見られたような、スレッド 1 での容量性ミス (ブロック競合ミス) が、スレッド共用キャッシュとすることによって解消される可能性がある。

このようにキャッシュへのマッピングは各スレッド毎によっても異なるし、また時間によっても異なってくるので、各スレッド毎に専用のキャッシュを各々持つよりは、すべてのスレッドが 1 つのキャッシュを共用する方が有利であると考えられる。

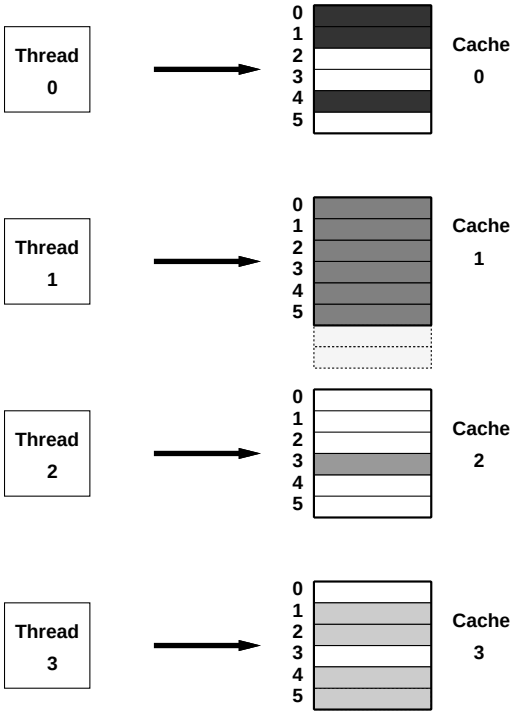


図 3.1 各スレッド毎に専用のキャッシュを持つ場合のマッピング状態

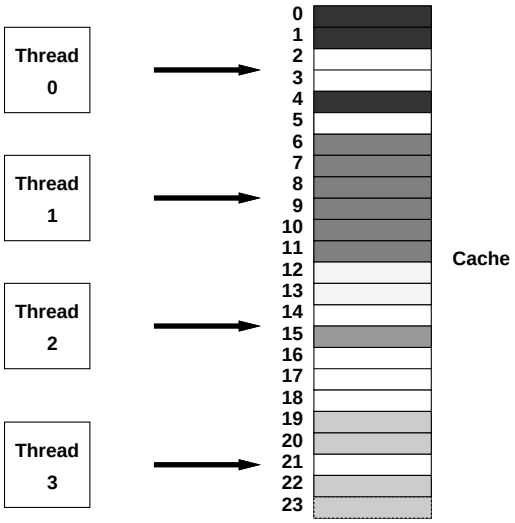


図 3.2 すべてのスレッドが 1 つのキャッシュを共用する場合のマッピング状態

3.1.2 ダイレクトマップとセットアソシアティブ

しかし、図 3.2 のようにいつもうまくマッピングできるとは限らない。A と B で示されたブロックの部分が他のスレッドも同じくそのブロック上にマッピングを必要とする場

合も考えられる。この場合キャッシュのマッピング方式をセットアソシアティブ化することによって、ブロック競合を回避することが考えられる。

キャッシュの容量が一定の場合、セットアソシアティブ化することによってライン数は減少してしまう。これは各スレッド間でのブロックの競合をより増加させてしまうのではないかと考えられるが、プログラムには局所性があるので各スレッドがすべて広いキャッシュ空間を必要としている場合は少ないと考えられる。

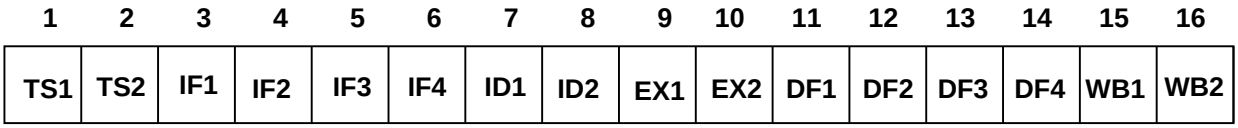
マルチスレッドを扱うキャッシュのマッピング方式について、セットアソシアティブ化を採用するとした場合、連想度をどの程度にするのが適当かを調べる必要がある。しかし連想度をむやみに増加させるとコストの増加だけでなく容量一定の場合セット数を減少させることになるので、かえってミスが増大してしまう恐れがあるので注意が必要である。

3.2　ミスペナルティ

各スレッドから発生するキャッシュミスによるメインメモリへのロードおよびメインメモリへのストアがどのようなペナルティを受けるかについて考察する。

ミスペナルティは、メインメモリへのアクセス時間、また後述するメインメモリアクセス要求待ち列での待ち時間、さらに、キャッシュの更新時間の和である。以下、各々について考察する。

図 3.3 に本研究で想定しているプロセッサのパイプライン構成を示す。また、本研究で検討したメモリ構成を図 3.4 に示す。



(**命令キャッシュおよびデータキャッシュに関するステージ**)

・命令キャッシュ

IF1：アドレスのデコード

IF2：キャッシュの読みだし

IF3：キャッシュの判定

IF4：データ出力

・データキャッシュ

DF1：アドレスのデコード

DF2：キャッシュの読みだしおよび書き込み

DF3：キャッシュの判定

DF4：データ出力

図 3.3　パイプライン構成

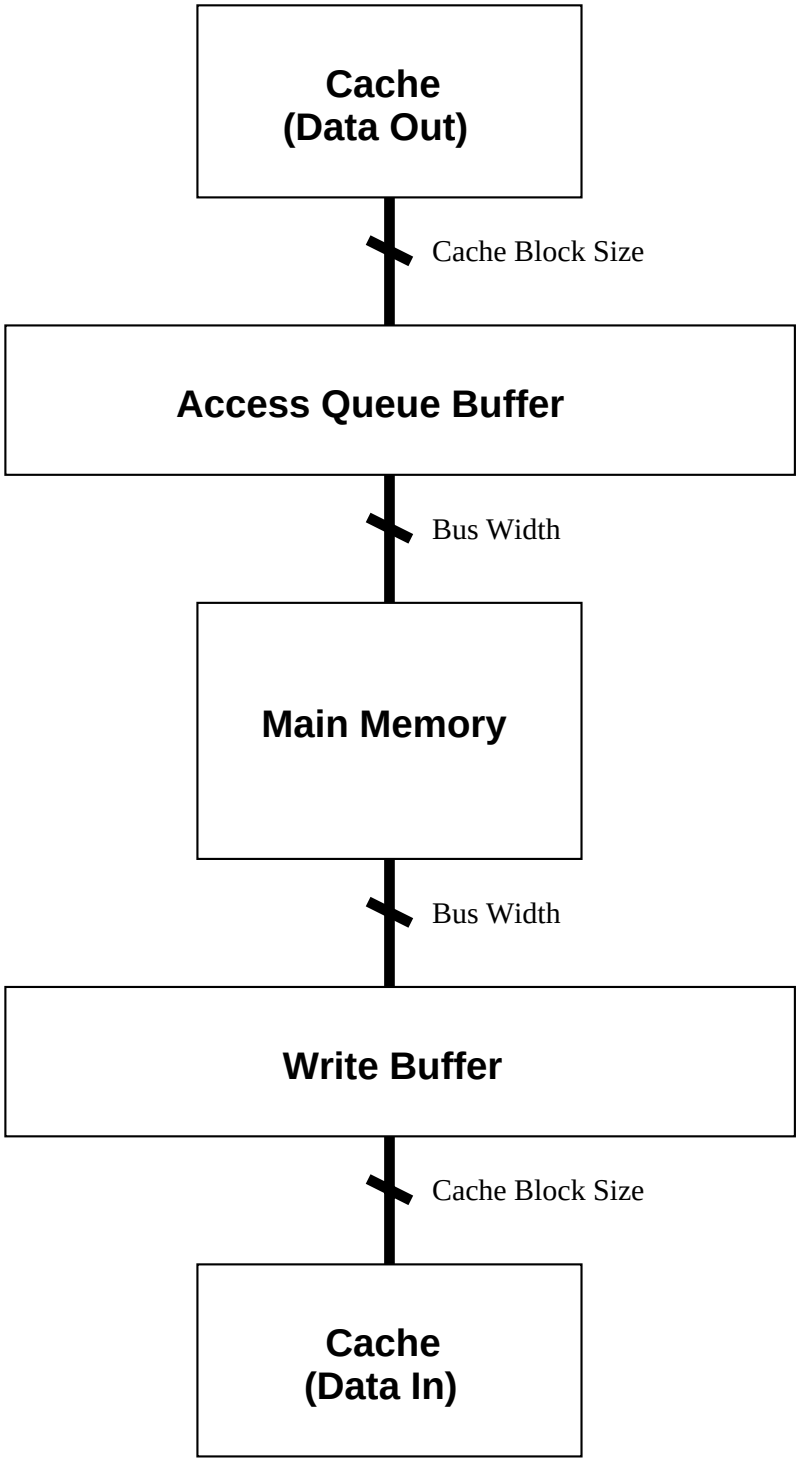

Cache
(Data Out)
Cache Block Size
Access Queue Buffer
Bus Width
Main Memory
Bus Width
Write Buffer
Cache Block Size
Cache
(Data In)

図 3.4　メモリ構成

3.2.1 ブロックサイズとミスペナルティ

メモリのアクセス速度は年々向上を続けているが、プロセッサの速度の向上の方がメモリよりも急速に向上を続けている。また本研究ではマルチスレッド型プロセッサを前提としているので、メモリへのアクセス要求は各スレッドが生起させたキャッシュミスから次々とやってくる。このため、マルチスレッド型プロセッサではメモリの構成が重要となってくる。

プロセッサとメインメモリの速度差が大きくなるにつれ、メインメモリへのアクセスペナルティは大きくなる。メインメモリへのアクセス要因の 1 つであるキャッシュミスを低下させるために、キャッシュのブロックサイズを大きくするという手法がよく行われている。ここで、キャッシュのブロックサイズに対するいくつかのメモリ構成

1. メモリのすべての構成要素の幅が 1 ワード
2. メインメモリ・バス・キャッシュ各々の幅が 4 ワード
3. バス・キャッシュ各々の幅が 1 ワードでインターリーブ方式のメインメモリを適用
4. メインメモリ・バス・キャッシュ各々の幅が 4 ワードでインターリーブ方式のメインメモリを適用

によるペナルティを算出した。今回考慮したメインメモリのアクセス時間は 57 ナノ秒 (18M ヘルツ) とした。その結果を図 3.5 に示す。

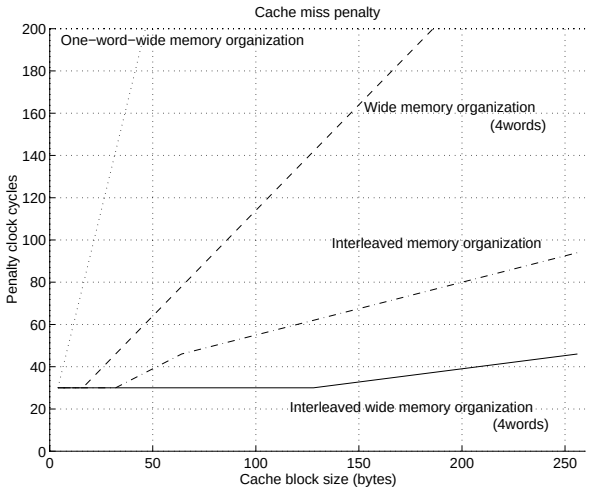


図 3.5 キャッシュブロックサイズに対する各メモリ構成のアクセスペナルティ
(クロック周波数=220M ヘルツ)

図 3.5 からも分かるように、メモリのすべての構成要素の幅が 1 ワードの場合、ブロックサイズが 16 バイト (4 ワード)、データ幅を 4 ワードの広げたメモリ構成では 64 バイト (16 ワード) を各々越えるとペナルティが 100 クロックサイクルを越えてしまう。これに対してインターリーブ方式のメインメモリ構成を適用することによって、キャッシュのブロックサイズの増加に対してもペナルティを十分に低く抑えていることが分かる。

次にメインメモリの動作速度が一定の場合、プロセッサの動作周波数が向上するにつれペナルティがどう変化するかを見るために、プロセッサの動作周波数に対するペナルティをキャッシュのブロックサイズをパラメータとして、各メモリシステムの構成方式について算出した。メインメモリのアクセス時間は前と同様である。その結果を図 3.6 に示す。

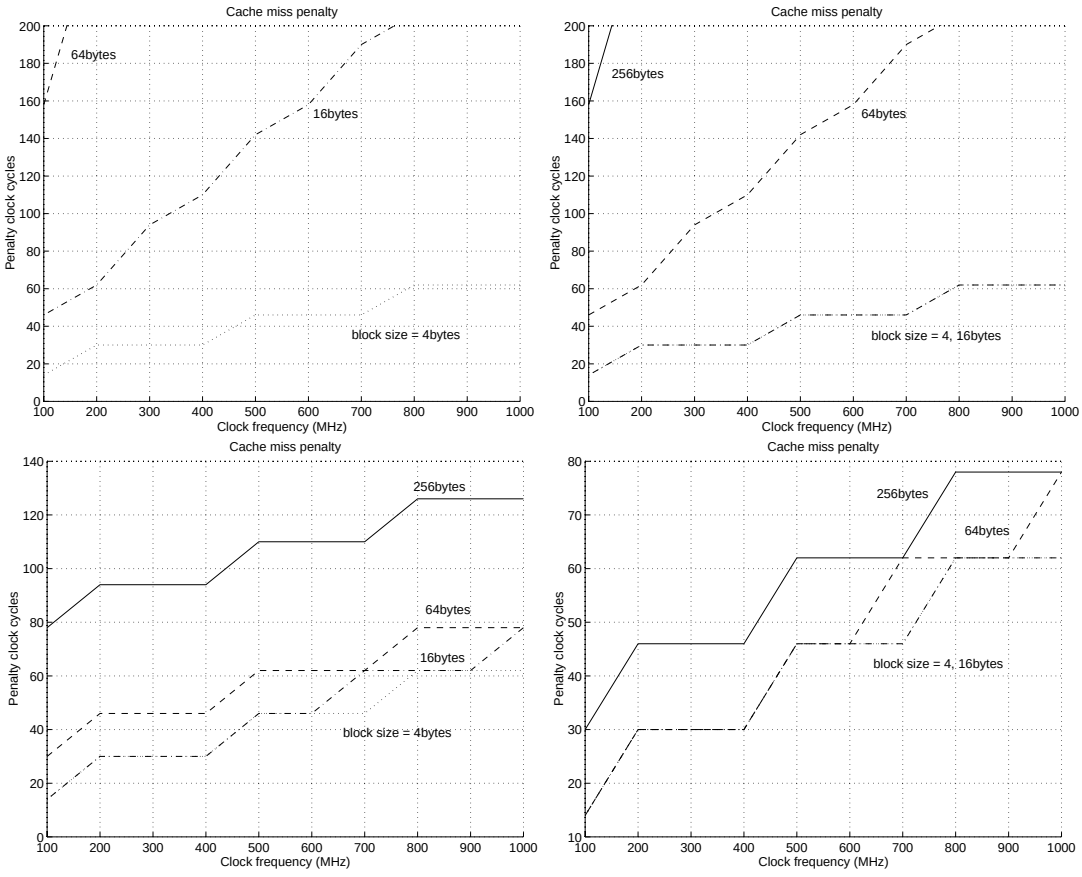


図 3.6 キャッシュミスペナルティ

左上：One-Word-wide organization

右上：Wide memory organization(4words)

左下：Interleaved memory organization

右下：Interleaved wide memory organization(4words)

図3.6を見ると、ブロックサイズが16バイト(4ワード)程度までならデータ幅を4ワードに広げたメモリ構成でも高いクロック周波数に対応できるが、それ以上のブロックサイズを望むにはインターリーブ方式を用いないと対応できないものと思われる。データ幅1ワードでのインターリーブ方式ではブロックサイズは64バイト程度までが妥当であると思われる。データ幅4ワードのインターリーブ方式ではブロックサイズを256バイト(64ワード)としてクロック周波数を1Gヘルツとしてもペナルティは80クロックサイクルにも満たない結果となった。

3.2.2 メモリ要求待ち列

シングルスレッドの場合、例えば命令キャッシュでミスが発生すると、その実行はメインメモリからそのデータがキャッシュ上に書き込まれるまで中断されるのでメインメモリへのアクセスは 1 つしか起こらない。

しかしマルチスレッド型プロセッサでのキャッシュへのアクセス権はクロックサイクル毎に異なるスレッドに切替えられるため、1 つのスレッドがキャッシュミスを起こし、そのスレッドの実行を停止しても、次のクロックサイクルで次のキャッシュへのアクセス権をもつスレッドがキャッシュにアクセスする。このときこのスレッドが引続きキャッシュミスを起こす可能性もある。データキャッシュも同様に考えることができる。ただし、キャッシュミスで生じたスレッドは停止するので、キャッシュミスによるメインメモリへの最大アクセス数はスレッド数ということになる。

メインメモリへのアクセスはキャッシュミスの時のみで起こるのではない。キャッシュからメインメモリへストアする場合もメインメモリへのアクセスが発生する。

しかし、プロセッサとメインメモリの速度差は非常に大きい。メインメモリのスループットよりもその時間あたりにキャッシュから発生するメインメモリへの参照数が大きければメインメモリへのアクセスに待ち列が発生することになる。待ち列の長さは下記のメモリサイクル利用率に依存する。この値は 1 より十分小さくなければならない、

$$\text{メモリサイクル利用率} = \frac{\text{メインメモリアクセスサイクル時間} \times \text{メインメモリアクセス確率}}{\text{マシンサイクル時間}} \tag{3.1}$$

例えば、メインメモリアクセスサイクルが 50 サイクルの場合メインメモリアクセスの確率が 2 %を越えると待ち列長はスレッド数と等しくなり、すべてのスレッドの実行は停止する。待ち列が発生すると、時間的に前に発生したメインメモリへのアクセスが終了するまで次のメインメモリへのアクセスは実行できないので、このスレッドによるメインメモリへのアクセスペナルティはその前のアクセスペナルティをも被ることになる。よって待ち列が及ぼすプロセッサのスループットへの影響は大きい。したがって、メインメモリへのメモリサイクル利用率を 1 より十分小さくすることが性能上重要となってくる。

また、メインメモリのマルチユニット化と、バンク毎に要求待ち列を設けることが有効である。この方策のねらいは、メモリユニット毎のメモリサイクル利用率を低下させることである。

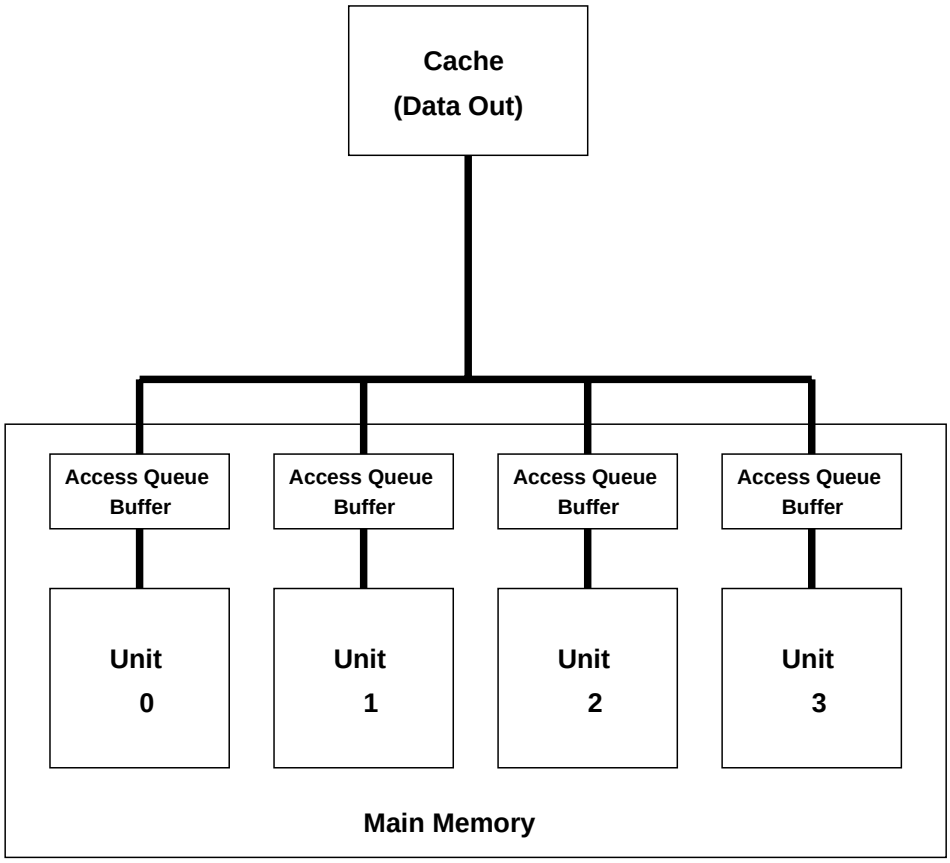


図 3.7 メインメモリのマルチユニット化

メインメモリを複数ユニットで構成し、ユニット毎に待ち列を設けた場合、待ち列にはアクセス種別 (リード / ライト)、アドレスの他に、スレッド番号を保持する必要がある。スレッド番号は、アクセス動作完了時にプロセッサにスレッド番号を告知するために用いる。待ち列にスレッド番号を保持するのは、アクセス動作完了がどのユニットで行なわれるか非同期的であり、アクセス完了がプロセッサからのアクセス要求順とはならないからである。

3.2.3 メインメモリ要求の削減

メインメモリへのアクセスはキャッシュミスの際のリード要求、あるいはストア命令生起時のライト要求である。このうち、ライト要求が発生した時にキャッシュの書き込み方式によってメインメモリへのトラヒックが異なる。

書き込み方式には基本的な手法として、ライトスルー方式がある。この方式ではライトがキャッシュとメインメモリの両方[1]のブロックに対して行われるため、ストア命令によるライトが発生するたびにメインメモリへのアクセスが行われる。

これに対して、メインメモリへのトラヒックを低く抑える手法として、ライトバック方式がある。この方式ではストア命令によるライトはキャッシュのブロックに対してだけ行われる。データが変更されたキャッシュ中のブロックは、置き換えの対象になった時にメインメモリへライトバックされる。そのため、キャッシュの同じブロックに何度ストアを行ってもメインメモリへの書き込みはそのブロックが置き換えられる 1 回だけである。

メインメモリへのトラヒックを減少させることは待ち列の長さを抑えることになるので、プロセッサのスループット向上にそのままつながる。

このため、マルチスレッド型プロセッサでのキャッシュの書き込み方式にはライトバック方式が適切であると考えられる。

[1] これはライトアロケート方式の場合。ノーライトアロケート方式では、ブロックへの書き込みは主メモリに対してだけ行ない、キャッシュへのロードは行なわない。

3.2.4 ブロック更新ペナルティの最小化

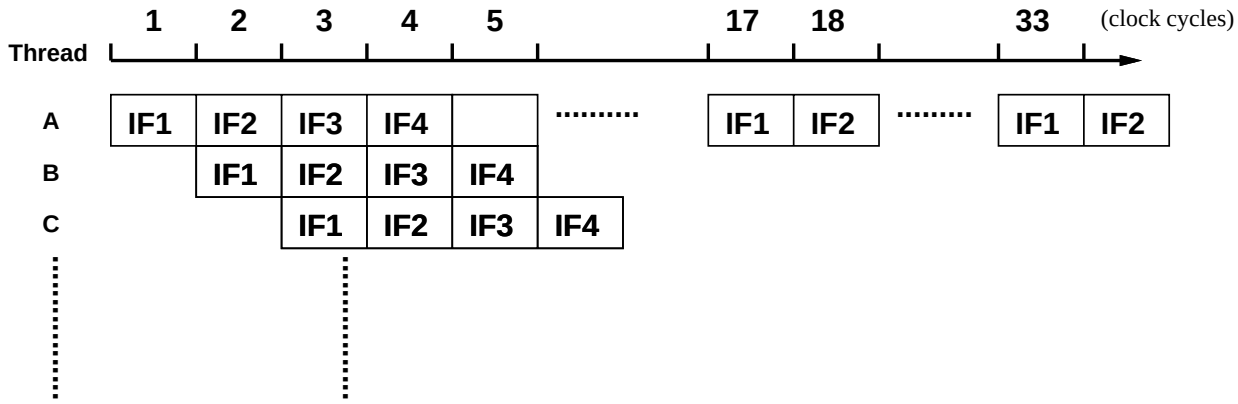


図 3.8 各スレッドのキャッシュへのアクセス (命令キャッシュ)

図 3.8 に、各スレッドがキャッシュへアクセスする様子を示す。この図からも分かるように、キャッシュへのアクセスはクロックサイクル毎に異なるスレッドからやってくる。そのため、メインメモリからあがってきた命令/データをキャッシュへ書き込むためには、そのスレッドがキャッシュアクセスステージに現われた時にしか書き込むことができない。キャッシュに書き込まれたデータはそのスレッドが再びそのステージに現われて読まれることになるので結果的にキャッシュミスペナルティは大きくなる。

このことからメインメモリから命令/データがあがってきた時のキャッシュへの書き込みに対してどのような対策を施すかによってもプロセッサのスループットに大きな影響を及ぼす。この解決策をいくつか検討してみる。

A. キャッシュの完全なマルチポート化

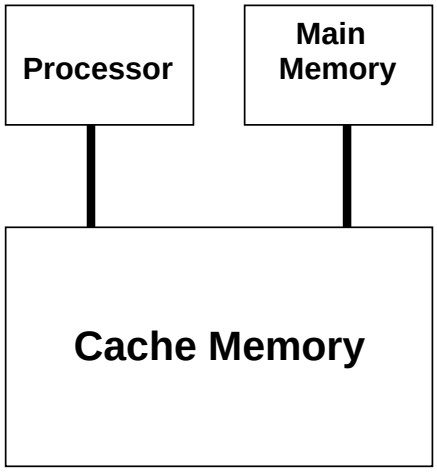


図 3.9 キャッシュポートの完全なマルチポート化

この方式では、各々専用のポートが用意されているので、あるスレッドがキャッシュへアクセスを行なっている時でも、別のスレッドのブロックを更新することができる。しかし、ハード的に実現するには 2 つのポートのために、専用のデコーダとビット線を用意しなければならなくコストが高くつく。

B. キャッシュのマルチバンク化

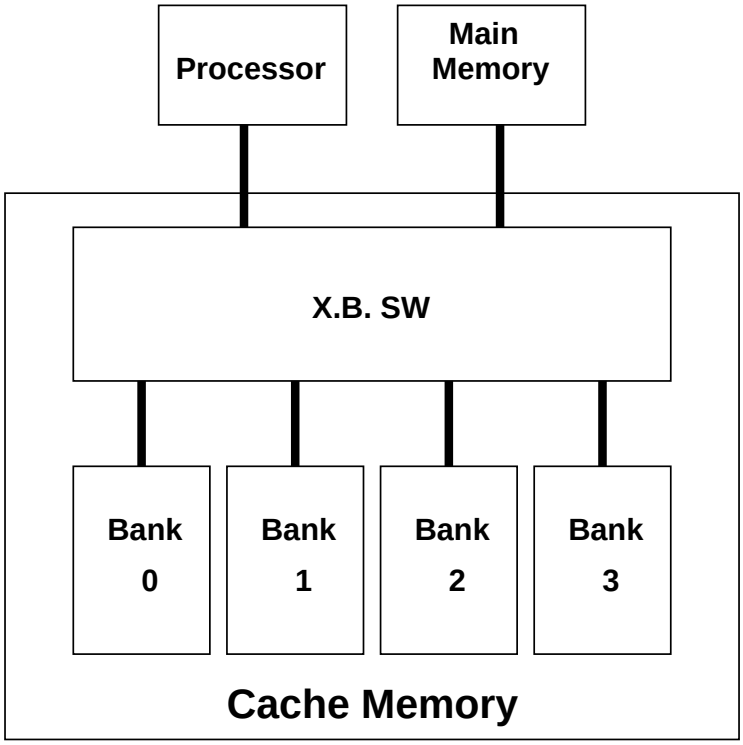


図 3.10 キャッシュのマルチバンク化

キャッシュを複数バンクで構成し、プロセッサとメインメモリからのアクセス要求はクロスバースイッチにより選択する。プロセッサからのアクセス要求は毎サイクルあるが、そのアクセスバンクはサイクル毎に分散するであろうから、空いているバンクにメインメモリからの書き込みが可能となる。構造的には簡単であるが、バンク競合が起こる可能性があり、ブロック更新ペナルティをゼロにはできない。

C. 各スレッドに対応したバッファの設置

前述の 2 つの手法は、キャッシュミスを発生したスレッドが再びキャッシュを参照することになっている。これに対して、図 3.11 に示すように、各スレッドに対応したバッファを各々用意し、ブロックの更新によるペナルティをなくす手法が考えられる。

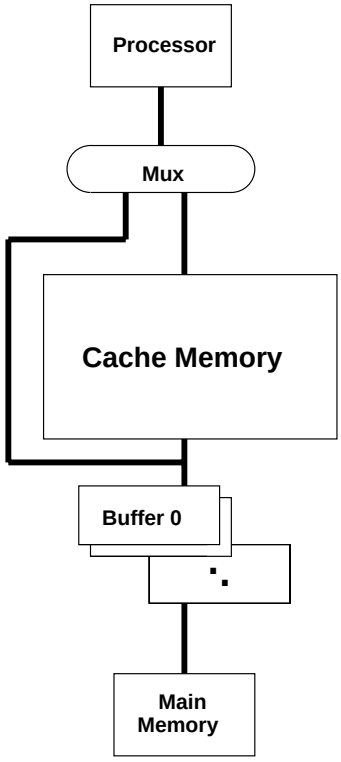


図 3.11 スレッド専用バッファの設置

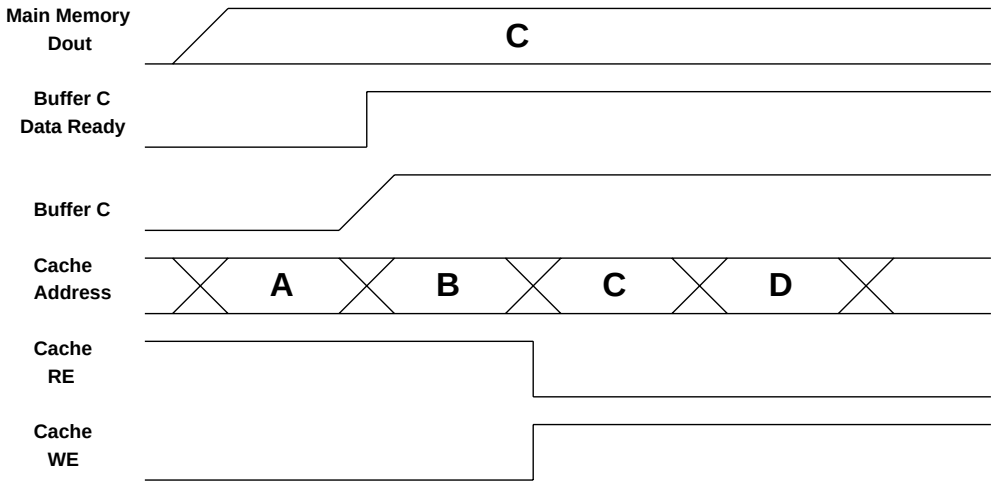


図 3.12 キャッシュのブロック更新のタイミング関係

図 3.12 に、スレッド C がミスを起こした場合のキャッシュのブロック更新のタイミング関係を示す。更新されるブロックデータがメインメモリからあがってきたら、それに該当するスレッドのバッファに書き込む。ミスを起こしたスレッドはキャッシュにアクセスせず、自分のバッファに命令/データを読みに行く。その時キャッシュにはスレッドのアクセスが起きないので、キャッシュへの書き込みを行うことができる。これによってキャッシュミスを起こしたスレッドはバッファに命令/データがあがっていれば、キャッシュへの書き込みを待たずに処理を開始することができる。同時にキャッシュへの書き込みも行うことができる。

この手法はコスト的に見ても、またブロックの更新によるペナルティを解消する点でももっとも好ましい。

第 4 章

シミュレーション実験と評価

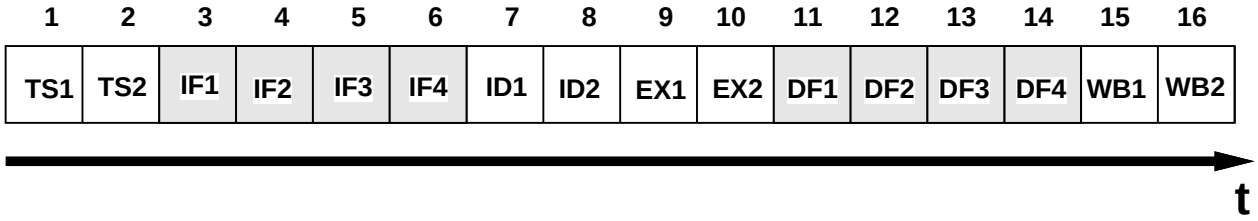
本章では、3 章で述べた方策の有効性を示すために、キャッシュのヒット率、およびスループットについて計算機上でシミュレーション実験を行なう。そして各パラメータを変化させた時の影響を見る。

4.1 実験の概要

4.1.1 システム構成

まず、図 4.1 に示すようなパイプライン構成の下でシミュレーション実験を行なう。図中で網掛けしてある部分がキャッシュに関するステージである。

スレッドは時間が進むにしたがって、図のパイプラインを左から右へながれて行くことになる。



(命令キャッシュおよびデータキャッシュに関するステージ)

・命令キャッシュ

IF1：アドレスのデコード

IF2：キャッシュの読みだし

IF3：キャッシュの判定

IF4：データ出力

・データキャッシュ

DF1：アドレスのデコード

DF2：キャッシュの読みだしおよび書き込み

DF3：キャッシュの判定

DF4：データ出力

図 4.1　パイプライン構成

キャッシュのデータパスとパイプラインのステージとの関係を図 4.2 に示す。

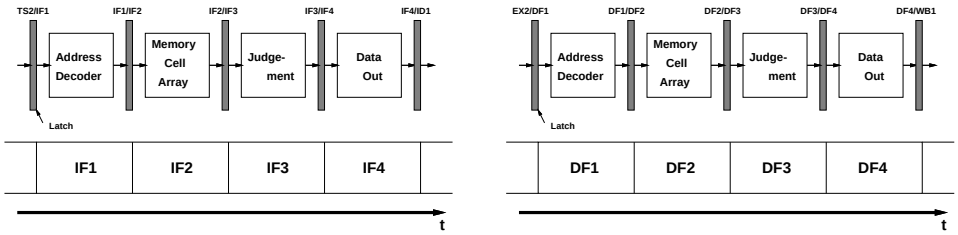


図 4.2 キャッシュのデータパスとパイプラインのステージとの対応関係

(左図：命令キャッシュ　右図：データキャッシュ)

4.1.2 トレースデータと主メモリ領域の配置

各スレッドのトレースデータとして、ベンチマーク[1]の cc、spice、tex を使用した。このトレースデータの形式は以下のようになっている。

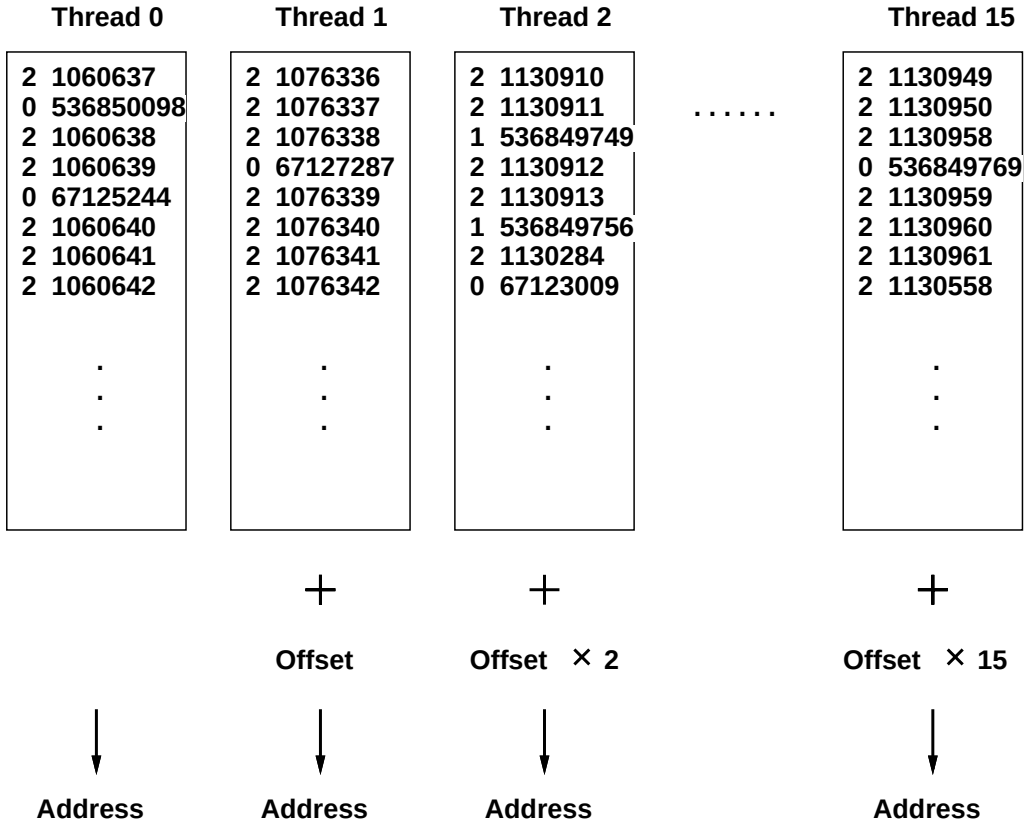


図 4.3 各スレッドのトレースデータ

各ベンチマークのトレースデータは、アクセス種別とアドレスで構成されている。

図 4.3 の先頭の数字がアクセス種別を表すもので、0 はデータキャッシュの読み出し、1 はデータキャッシュの書き込み、2 は命令キャッシュの読み出しとなっている。

その次の数字がアドレスである。本シミュレーションでは、MIPS アーキテクチャを想定している。MIPS アーキテクチャでは、各アドレスの最下位 2 ビットはワード内のバイトオフセット指定に使用される。このため、アドレスは用意されていたトレースデータを 4 で除算したワードアドレスを使用した (図中のアドレスは既に 4 で除算されたものである)。

[1]Stanford Univ. の Hennessy and Petterson から引用した

図 4.3 の中で、スレッド 0 のトレースデータが基準のトレースデータである。その他のスレッドのトレースデータに、これと同じトレースデータ (開始アドレスが同じ) を用いてしまうと、トレースデータの振舞がすべてのスレッドで同じようになってしまうので、オリジナルデータをランダムにある長さで分割し、その分割された先頭アドレスからトレースデータとして使用している。

また、各スレッドのトレースデータにはオフセット (プログラムサイズ以上) を付加させ、各々のアドレスとしている。これにより、各スレッドのトレースデータは主メモリ上に重なることなく配置される。

トレースデータに用いる cc、spice、tex の性質を見るために、ブロック数 (256 ワード / ブロック) に対する命令の累積分布[2]およびデータの累積分布を各々図 4.4、4.5 に示す。

図 4.4、4.5 から、命令およびデータの累積分布は同じような性質を持っていることが分かる。tex はデータの累積分布において、32 ブロック数 (8k ワード) 程度で、すでに 100%近い累積分布を示していることが分かる。つまりプログラムの局所性が高いということが言える。逆に、cc はプログラムの局所性が低いということが言える。

[2]ブロック単位でアクセス頻度を調べ、最大のものからブロック順をソートして、結果を累積したもの

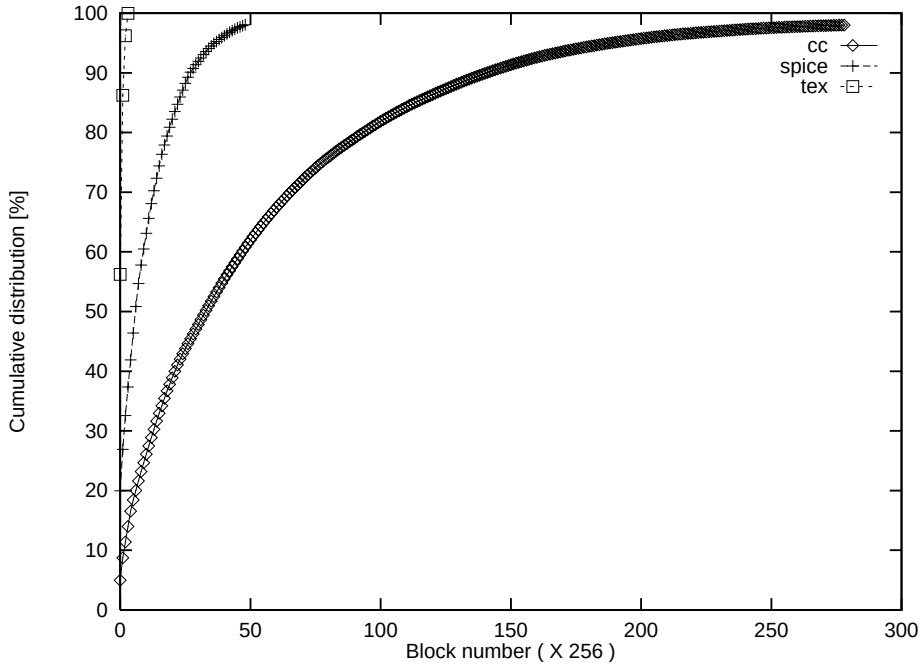


図 4.4 各トレースデータのブロック数に対する命令の累積分布

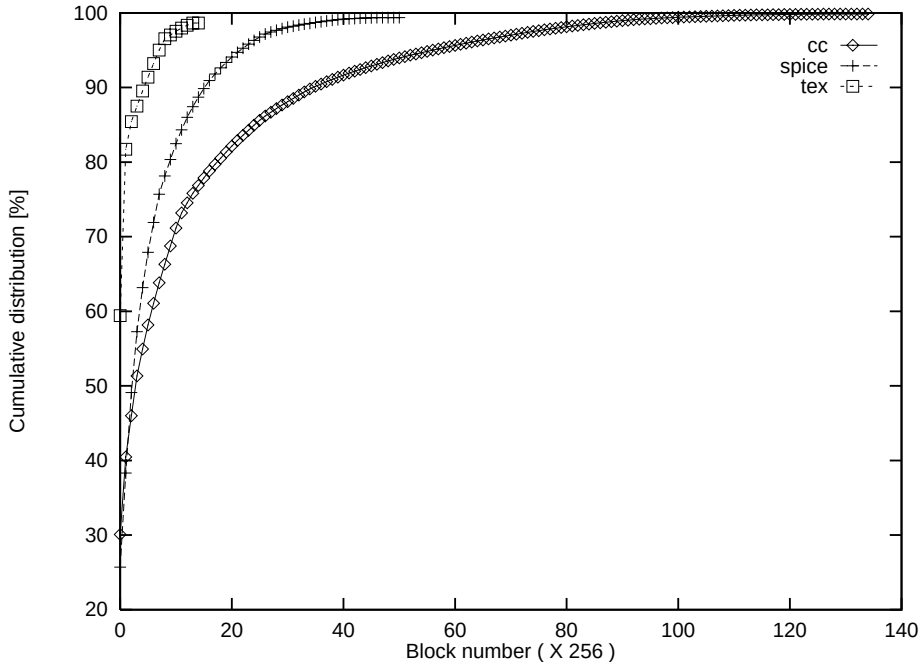


図 4.5 各トレースデータのブロック数に対するデータの累積分布

つぎに、各ベンチマークの命令のアドレス分布範囲 (プログラムサイズ) を示す。

表 4.1 各ベンチマークのアドレス分布範囲

	cc	spice	tex
アドレス分布範囲 [word]	143k	107k	32k
９０%範囲 [word]	35.3k	6.7k	0.3k

表 4.1 を見ると、累積分布が 90 パーセントを占める範囲は、アドレスが分布している範囲の数パーセントにすぎない。つまり、分布が局所的であることが分かる。

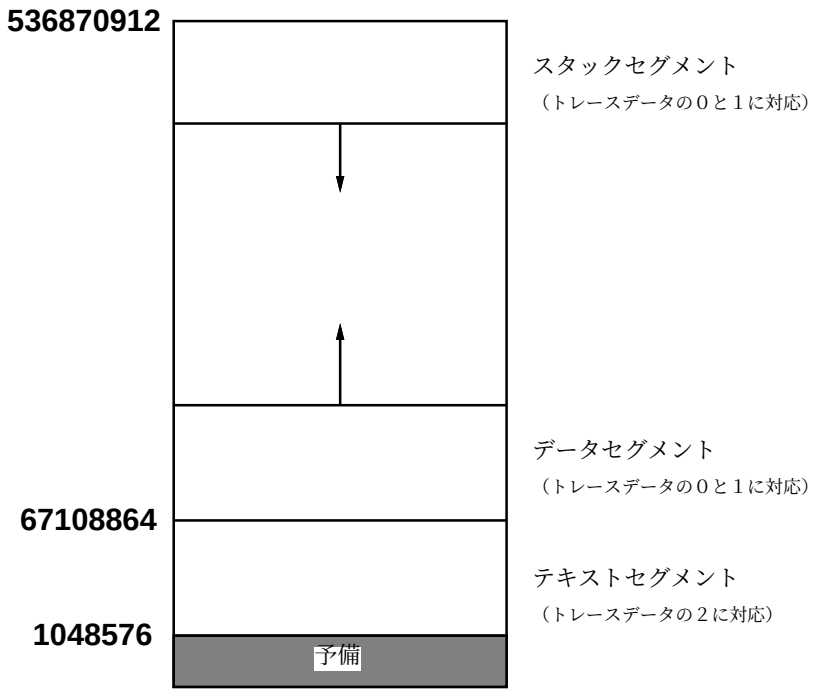


図 4.6 主メモリ領域の配置

MIPS プロセッサベースのシステムは、一般に主メモリを 3 つの領域に分割する (図 4.6)。図中の数字は対応するトレースデータのアドレス (4 で除算したもの) である。最初の部分はアドレス空間のボトムの近く (開始アドレス 1048576) にあるテキストセグメントである。

2 番目の部分はテキストセグメントの上に位置するデータセグメント (開始アドレス 67108864) である。このセグメントは上方に拡張される。

3 番目の部分であるスタックセグメントは、仮想アドレス空間の最上部 (開始アドレス 536870912) に配置される。このセグメントは下方に拡張される。

なお、各スレッドをこの主メモリ領域へ配置するには、スレッド毎にオフセットを付加して各セグメントに配置する。

4.1.3 キャッシュ容量

ベンチマークの中で中間的な局所性を持つ spice を基準として考え、spice のトレースデータの 90%程度を収められるよう、キャッシュ容量は 1 スレッドにつき、8k ワードとする。

シミュレーションで複数のスレッド (16 個) を扱う時は、キャッシュ容量もスレッド数分用意するものとする。

なお、キャッシュの容量 (タグ部などを除く) はマッピングの方式にかかわらず一定とする。

4.1.4 メモリ構成

キャッシュメモリのブロックサイズは 4 ワードとする。キャッシュメモリには各ブロックにバリッドビット (そのブロック内のデータが有効な情報を持っているかどうかを表す) が設けられている。キャッシュがセットアソシアティブ化された時は、各セットのブロック毎にバリッドビットが設けられる。

主メモリは 4 バンク[3]構成としている。バンクへのアクセスはキャッシュの書き込み方式に応じて異なる振舞いを示す。。

[3]前章で述べたユニットとは異なる、ユニット内のバンクである。ここではユニット数を 1 としている。

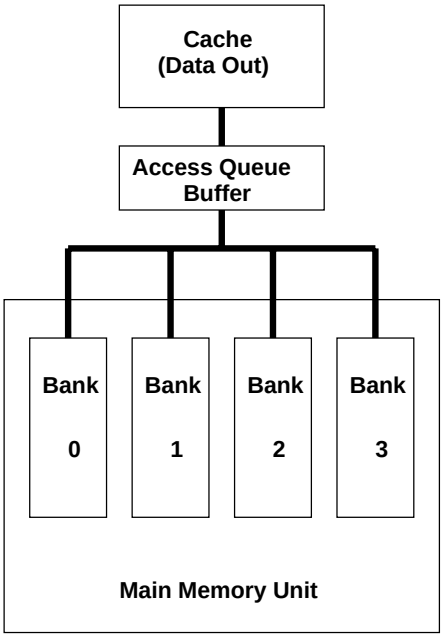


図 4.7 メモリユニットのバンク化

- ライトバック方式の場合
  キャッシュブロックの読み出しは、ブロックサイズ分のデータ幅で主メモリから転送する。キャッシュブロックの書き込みは、そのブロックが置き換え対象になった時に行なわれるので、読み出しと同様に、ブロックサイズ分のデータ幅で主メモリへ転送する。つまり、主メモリのすべてのバンクは同一にアクセスされる。

- ライトスルー方式の場合
  キャッシュブロックの読み出しは、ライトバック時と同様に、ブロックサイズ分のデータ幅で主メモリから転送する。しかし、キャッシュブロックへの書き込みは 1 ワード単位で行なわれるため、主メモリへの書き込みも同様に 1 ワード単位で行なわれる。ライトスルー方式では、ストア命令が発生するたびにメモリユニットへのアクセスが発生するため、メモリユニットは図 4.6 のように 4 バンク化している。各バンクは、アドレスの下位 2 ビットに対応づけている。これにより、書き込みに対するアクセスが各バンクに分散されるため、書き込みにおけるメモリアクセス効率の向上が期待できる。しかし、キャッシュブロックの読み出しは、ブロックサイズ分のデータ幅で行なうため、すべてのバンクがアクセス可能状態にならないと行なうことはできない。

4.1.5 スループットの定義

1 命令の実行は各パイプラインステージで処理が施され、パイプラインを完全に抜けた時、その実行は終了する。本シミュレーション実験では、図 4.8 に示すように、1 命令の実行は 16 段のパイプラインを完全に抜けた時に完了するとしている。

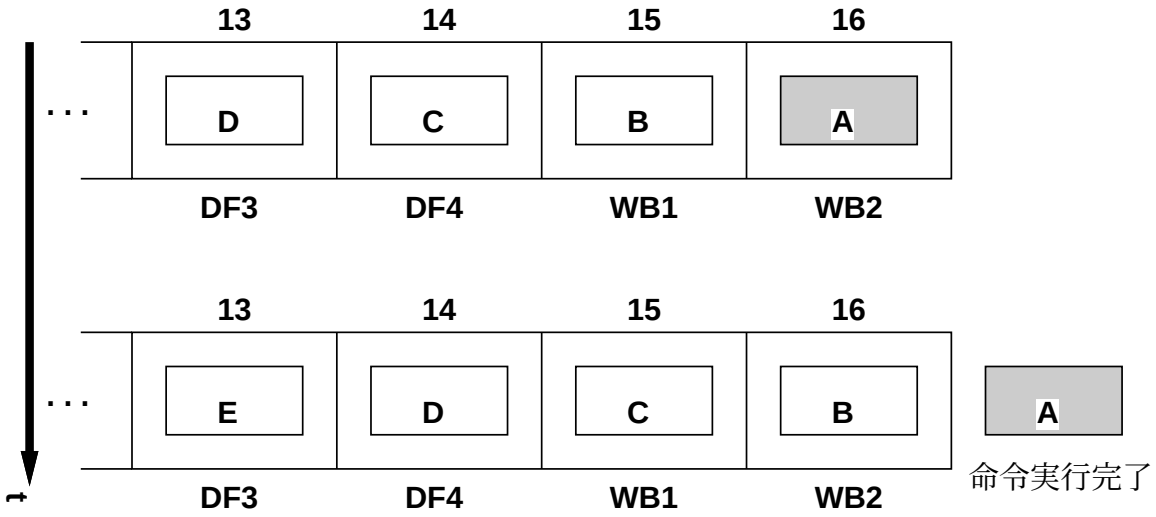


図 4.8 命令実行の完了

スループットは、実行されたクロックサイクルに対して、命令実行が完了した比率を表すものである。よって、このスループットがシステム本来の性能を計る尺度になる。

4.1.6 シミュレーションの流れ

本研究で想定している、プロセッサのパイプライン構成は、図 4.1 に示すものである。実際は、各ステージで各々の処理が行なわれるが、シミュレーション上では図 4.9 の網かけをした部分で、キャッシュに関する処理をまとめて行なっている。

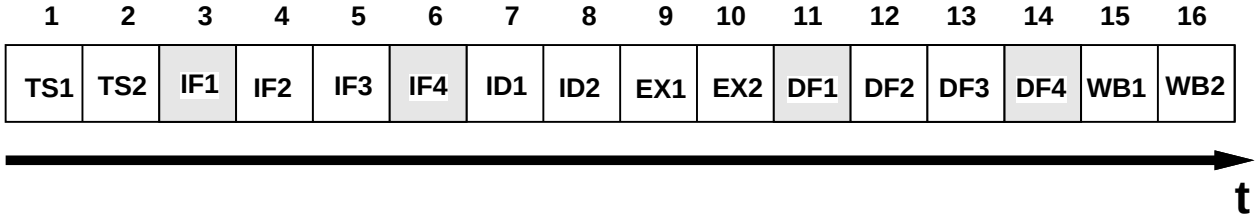


図 4.9 シミュレーション上でキャッシュ処理を行なっているステージ

シミュレーションで、図 4.9 で網かけされていない部分では、何も処理を行なう必要がないので、スレッドがこのステージに現われた場合は、そのスレッドの時刻が進むだけである。

はじめに、スレッドの状態に関して定義を行なう。

- アクティブ状態

  スレッドの処理が行なわれている状態を示す。スレッドが初めてパイプラインに投入される時、アクティブ状態の形で投入される。スレッドがキャッシュヒットを起こすと、この状態になる。スレッドがアクティブ状態のままパイプラインを抜けた時、そのスレッドの実行が完了したとみなしている (**命令実行処理量が算出**)。

- ノンアクティブ状態

  スレッドの処理が停止している状態を示す。スレッドがキャッシュミスを起こすと、この状態になる。スレッドがノンアクティブ状態のままパイプラインを抜けた時、そのスレッドの実行は未完了とみなされ、スレッドをノンアクティブ状態にしたまま、再びパイプラインに投入される。

図 4.10 にシミュレーションの流れを示す。以下、シミュレーションステージの説明を行なう。

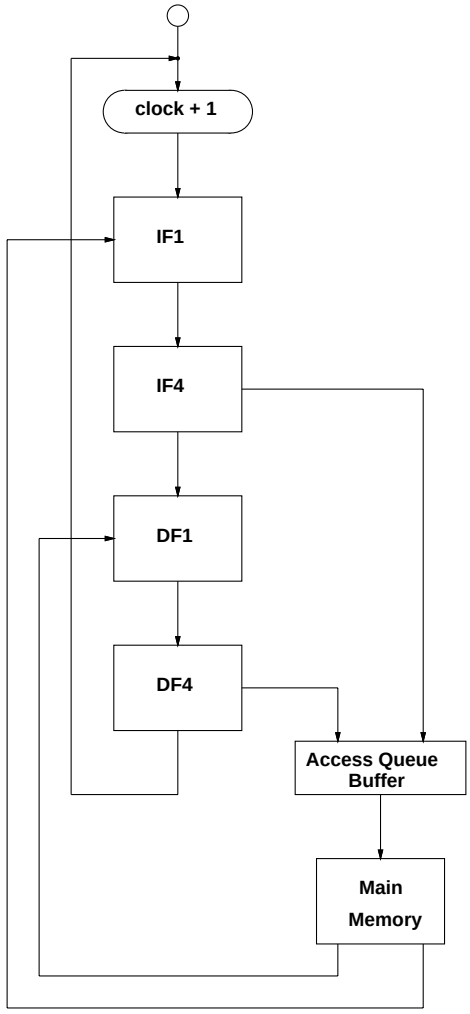


図 4.10 シミュレーションの流れ

- **IF1 および DF1 ステージ (キャッシュブロック更新ステージ)**

  ノンアクティブ状態のスレッドが、このステージに現われた時、スレッド専用バッファに命令やデータが主メモリからあがってきていたら、キャッシュブロックの更新を行ない、スレッドはアクティブ状態となる。

- **IF4 ステージ (キャッシュブロック更新を除く命令キャッシュ処理ステージ)**

  アクティブ状態のスレッドがこのステージに現われると、スレッドに対応したトレースデータが取り込まれる。ノンアクティブ状態のスレッドがこのステージに現われると、処理は行なわれず、スレッドの時刻が進行するだけである。

  トレースデータに、スレッド番号に対応したオフセットが付加され、そのスレッドのアドレスが決定する。

決定したアドレスを、キャッシュメモリへ投入する。ヒットが返されれば、そのスレッドはアクティブ状態となる。

もしミスが返されれば、そのスレッドはノンアクティブ状態となる (**命令キャッシュミスによる主メモリへのアクセス回数が算出**)。そして、主メモリのアクセス待ち列に、スレッド番号が投入される。実際は、アクセス待ち列にアドレスも投入されるが、シミュレーションでは、ミスを起こしたアドレスを格納する専用バッファを設けており、そこに投入するようになっている。キャッシュブロックを更新する際のアドレスはここから読み出される。

- **DF4 ステージ (キャッシュブロック更新を除くデータキャッシュ処理ステージ)**

  実際は、命令デコードを行なった結果、データキャッシュへアクセスを行なう必要がある場合のみ、データキャッシュアクセスが発生する。シミュレーションでは、トレースデータで命令キャッシュの読み出しの次に、データキャッシュのアクセスに関するデータが用意されている場合、データキャッシュへのアクセスが行なわれている。

  アクティブ状態のスレッドがこのステージに現われると、スレッドに対応したトレースデータが取り込まれる。取り込まれたトレースデータがデータキャッシュに関するものであれば、データキャッシュの処理が行なわれる。もし、取り込まれたデータが命令キャッシュに関するものであれば、このトレースデータは保持され、スレッドはアクティブ状態を保ったまま、このステージの処理は行なわない。ノンアクティブ状態のスレッドがこのステージに現われると、処理は行なわれず、スレッドの時刻が進行するだけである。

  各々のトレースデータに、スレッド番号、およびセグメント (データ、スタック) に対応したオフセットが付加され、そのスレッドのアドレスが決定する。

  決定したアドレスを、キャッシュメモリへ投入する。ヒットが返されれば、そのスレッドはアクティブ状態となる。

  もしミスが返されれば、そのスレッドはノンアクティブ状態となる (**データキャッシュミスによる主メモリへのアクセス回数が算出**)。そして、主メモリのアクセス待ち列に、スレッド番号とアクセス種別が投入される (**主メモリへのストア回数が算出**)。実際は、アクセス待ち列にアドレスも投入されるが、シミュレーションでは、

ミスを起こしたアドレスを格納する専用バッファを設けており、そこに投入するようになっている。キャッシュブロックを更新する際のアドレスはここから読み出される。

パイプラインステージに現われない、主メモリの処理は次のように行なわれている。

- 主メモリの処理

  クロックサイクル毎に主メモリのアクセス状態を調べ、もしアクセス可能なら、メモリ待ち列先頭のスレッド番号、およびアクセス種別が、主メモリへ投入される (**この時のメモリ待ち列長が算出**)。主メモリへ投入された要求は、メモリアクセスペナルティを経て、完了とみなされる。主メモリへのアクセスが読み出しの場合、アクセスが完了したら、命令/データのスレッド専用のメモリアクセス完了フラグがたつ (スレッド専用バッファに命令/データがあがってきたことに相当)。

4.2 実験と結果

4.2.1 キャッシュヒット率に関する実験 (実験 1)

A. スレッド専用キャッシュとスレッド共用キャッシュのヒット率

スレッドが専用キャッシュを持つ場合と、スレッドが共用キャッシュを持つ場合でのキャッシュのヒット率を比較する。

スレッド専用キャッシュの場合、スレッド数は 1 とする。また、キャッシュ容量は 1 スレッド分の容量 (8k ワード) を用意する。

スレッド共用キャッシュの場合、スレッド数は 16 とする。また、キャッシュ容量は 16 スレッド分の容量 (128k ワード) を用意する。連想度は 1,2,4,8,16 について行なう。

この実験より、スレッドのキャッシュのマッピングを行なう際に、スレッド専用キャッシュを設けるよりもスレッド共用キャッシュとする方が有利なことを示す。

表 4.2 スレッド専用キャッシュとスレッド共用キャッシュ(cc)

	専用	共用				
		1-way	2-way	4-way	8-way	16-way
命令	0.973	0.967	0.976	0.979	0.979	0.977
データ	0.980	0.970	0.978	0.980	0.980	0.981

表 4.3 スレッド専用キャッシュとスレッド共用キャッシュ(spice)

	専用	共用				
		1-way	2-way	4-way	8-way	16-way
命令	0.992	0.992	0.995	0.995	0.996	0.996
データ	0.991	0.988	0.991	0.991	0.991	0.991

表 4.4 スレッド専用キャッシュとスレッド共用キャッシュ(tex)

	専用	共用				
		1-way	2-way	4-way	8-way	16-way
命令	0.948	0.999	0.999	0.999	0.999	0.999
データ	0.980	0.996	0.996	0.996	0.995	0.996

結果から、キャッシュをスレッド共用キャッシュとしてもヒット率の低下はほとんど見られなかった。スレッド共用キャッシュで連想度を 2 以上持たせることで、スレッド専用キャッシュよりも高いヒット率を示している。

cc よりもアドレスの局所性の高い、spice や tex にいたっては、連想度にかかわらず、スレッド共用キャッシュの方がスレッド専用キャッシュと同等以上のヒット率を示していることが分かる。

B. 連想度によるキャッシュヒット率の変化

キャッシュの連想度に対するキャッシュのヒット率を、各性質を持ったトレースデータに対して行なう。

この実験より、連想度を持たせることにより、ブロックの競合を低く抑えられることを示す。また、セットアソシアティブ化を採用する場合に、プログラムの局所性に対する連想度の適当性についてみる。

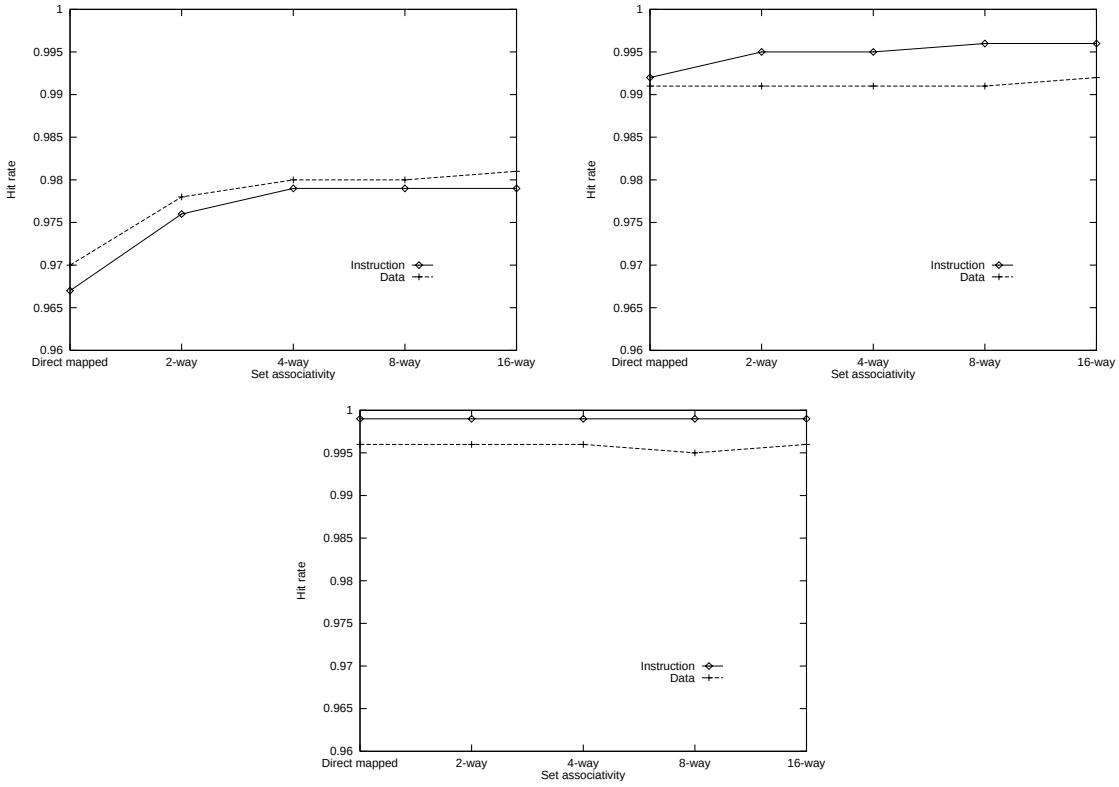


図 4.11 連想度によるキャッシュヒット率の変化
(左上：cc、右上：spice、下：tex)

結果から、高いヒット率を確保するためには、局所性の低いプログラムほど連想度が必要になることが分かる。

また、キャッシュのヒット率は、局所性の高いプログラムほど高い値を示していることが分かる。

4.2.2 スループットに関する実験 (実験 2)

A. メモリアクセスペナルティとスループット

メモリのアクセスペナルティに対するスループットの変化を見る。用いるトレースデータは、アドレス参照の局所性が 3 者の中で中間的な性質を持つ spice で行なう。

まず、キャッシュの書き込み方式をライトスルー方式とし、メモリユニット数を 1 とした時の実験結果を示す。

この実験より、メモリアクセスペナルティがシステム全体のスループットに強く影響することを示す。

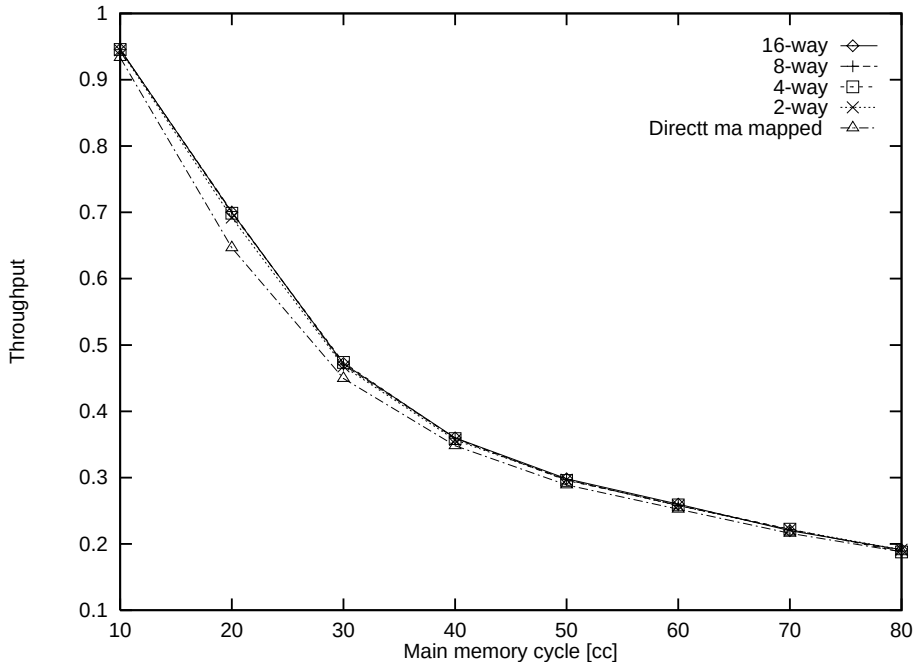


図 4.12 メモリアクセスペナルティとスループット (spice)

結果から、主メモリサイクルが 30 クロックサイクル程度まで増加する時に、スループットは急激に下降し、0.4 程度まで低下している。主メモリサイクルが 50 クロックサイクルの時に、スループットは 0.3 にも満たない結果となった。

B. キャッシュの書き込み方式とスループット

メモリのアクセスペナルティに対するスループットの変化を、キャッシュの書き込みをライトスルー方式の場合とライトバック方式について行なう。

また、キャッシュの連想度は、図 4.20 から 2 程度持てば spice でも高いヒット率を示しているので、2 としている。

この実験より、メインメモリ要求を削減するライトバック方式を採用することによって、メモリアクセスペナルティが増えた時のシステムのスループットの低下を低く抑えることができることを示す。

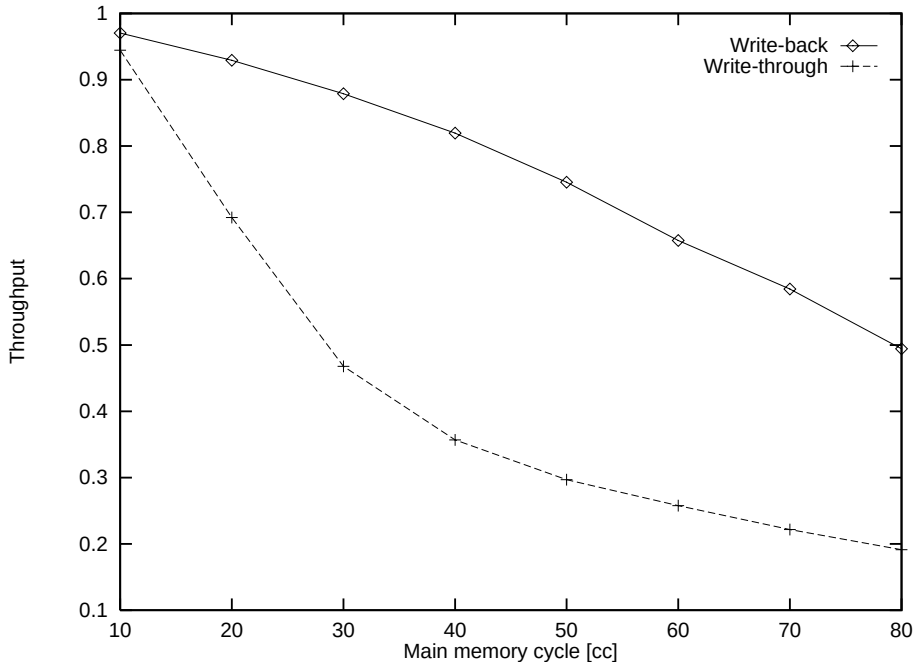


図 4.13 キャッシュの書き込み方式とスループット (spice)

結果から、キャッシュの書き込み方式によって、主メモリアクセスペナルティが増加した時のスループットの低下は明らかに違う変化を見せた。

主メモリサイクルの 30 クロック程度までに対するスループットの低下の割合が、ライトスルー方式では決定的に大きくなっていることが分かる。

ライトバック方式では、主メモリサイクルの増加に伴うスループットの低下が、全般的に緩やかであることが分かる。主メモリサイクルが 80 クロックサイクルと大きくなっても、スループットは 0.5 程度保たれており、この時のライトスルー方式とのスループットの差は 0.3 程度と、大きな値になっていることが分かる。

つぎに、キャッシュの書き込み方式に対する主メモリへの書き出し確率を示す。ライトスルーの場合、主メモリへの書き出しはストア命令が発生する毎に行なわれる。ライトバックの場合、主メモリへの書き出しは、キャッシュミス時にそのブロックが置き換え対象になっている時に行なわれる。

表 4.5 主メモリへのアクセス確率

	キャッシュ書き込み方式	
	ライトスルー	ライトバック
平均メインメモリアクセス確率	**10.78%**	**2.14%**
平均 書き出し確率	**8.38%**	**0.03%**

結果から、キャッシュの書き込み方式をライトバックにすることで、主メモリへの書き出し確率が約 1/25 にも減少していることが分かる。

つぎに、キャッシュの書き込み方式に対するメモリ待ち列の平均長の変化を示す。

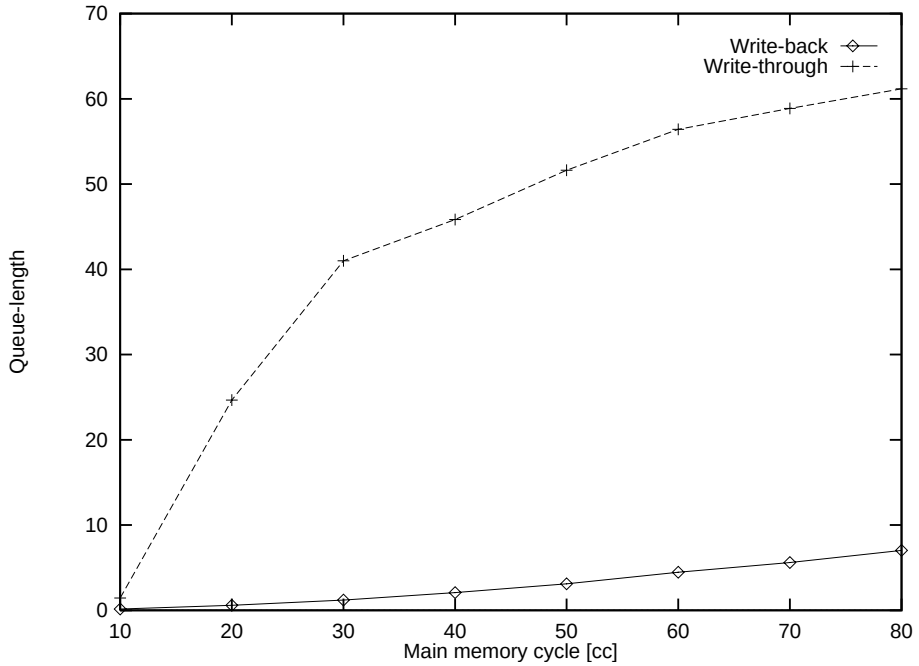


図 4.14 キャッシュの書き込み方式とメモリ待ち列の長さ (spice)

図 4.13 と図 4.14 を見比べると、スループットの低下に伴い、メモリ待ち列の増加が見られることが分かる。メモリ待ち列のグラフは、スループットのグラフに対称的である。

ライトスルー方式の場合、主メモリサイクルが 30 クロックサイクル程度までのメモリ待ち列に、急激な増加が見られることが分かる。それ以上の主メモリサイクルになると、メモリ待ち列の増加度合は緩くなっているのが分かる。

ライトバック方式では、主メモリサイクルの増加によるメモリ待ち列の増加度合は、ライトスルー方式に比べて全体的に緩やかになっている。

さらに、主メモリアクセスペナルティに対するスレッドのスループットの分散を両者の書き込み方式について示す。

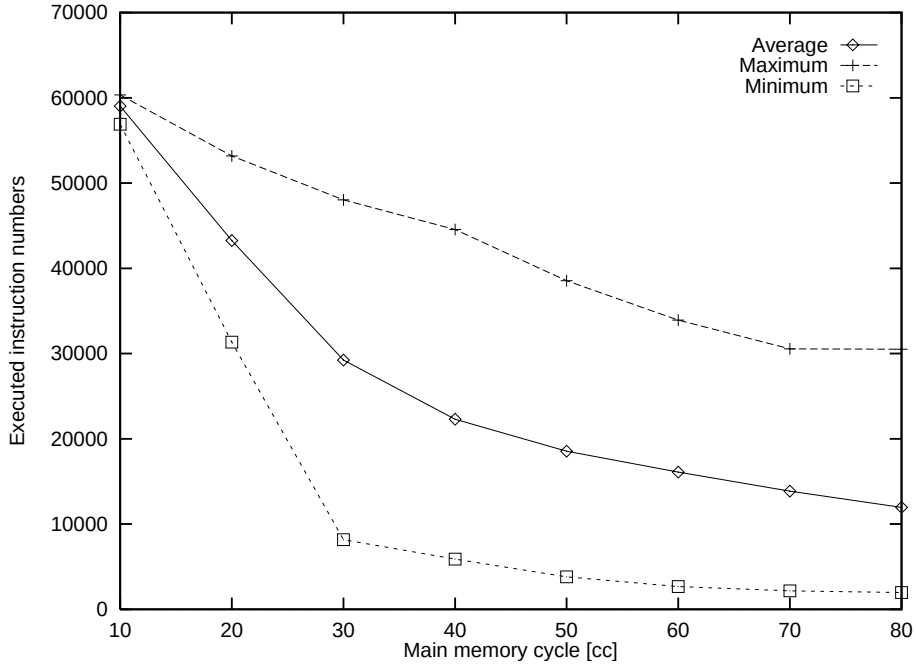


図 4.15 主メモリアクセスペナルティに対するスレッドのスループットの分散
(ライトスルー方式、総ステップ数=1,000,000)

ライトスルー方式の場合、主メモリサイクルが 30 クロックサイクル程度まで、スレッドの平均スループットの低下の度合が大きいが、スレッドのスループットの分散度合も 30 クロックサイクルをピークに大きくなっている。主メモリサイクルが 40 クロックサイクルを越え平均スループットの低下が緩やかになると、分散度合もほぼ一定となっていることが分かる。

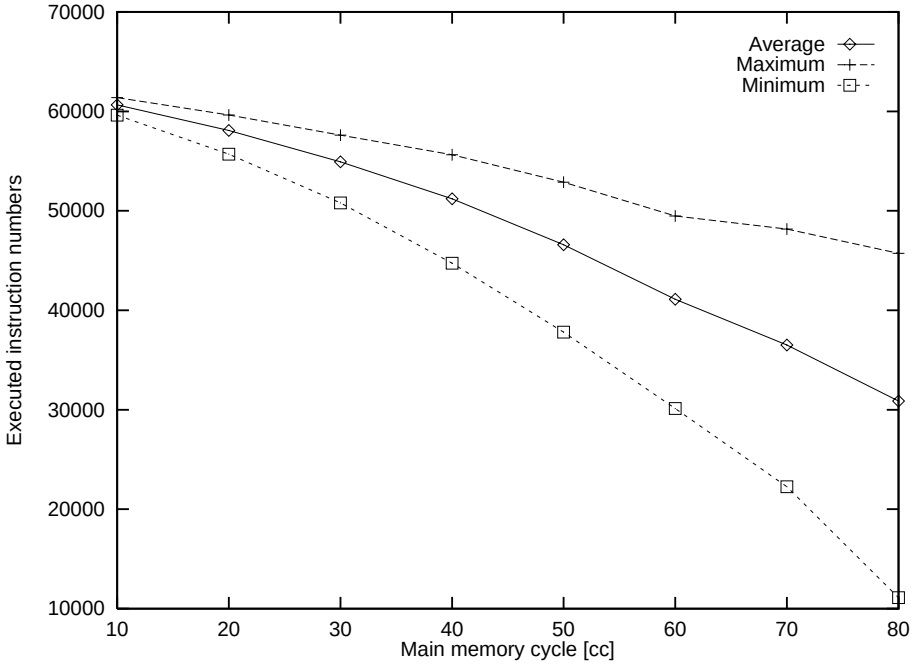


図 4.16 主メモリアクセスペナルティに対するスレッドのスループットの分散
(ライトバック方式、総ステップ数=1,000,000)

ライトバック方式の場合、主メモリサイクルが 40 クロックサイクル程度からスレッドの平均スループットの低下の度合が大きくなっていくが、これに応じて分散の度合も大きくなっていくことが分かる。

C. マルチユニット構成の効果

メモリアクセスペナルティに対するスループットの変化を、メモリのマルチユニット化の度合に対して行なう。

この実験より、メモリ構成をマルチユニット化することで、待ち列の発生を抑えることができ、メモリアクセスペナルティが大きくなってもスループットの低下を低く抑えることができることを示す。

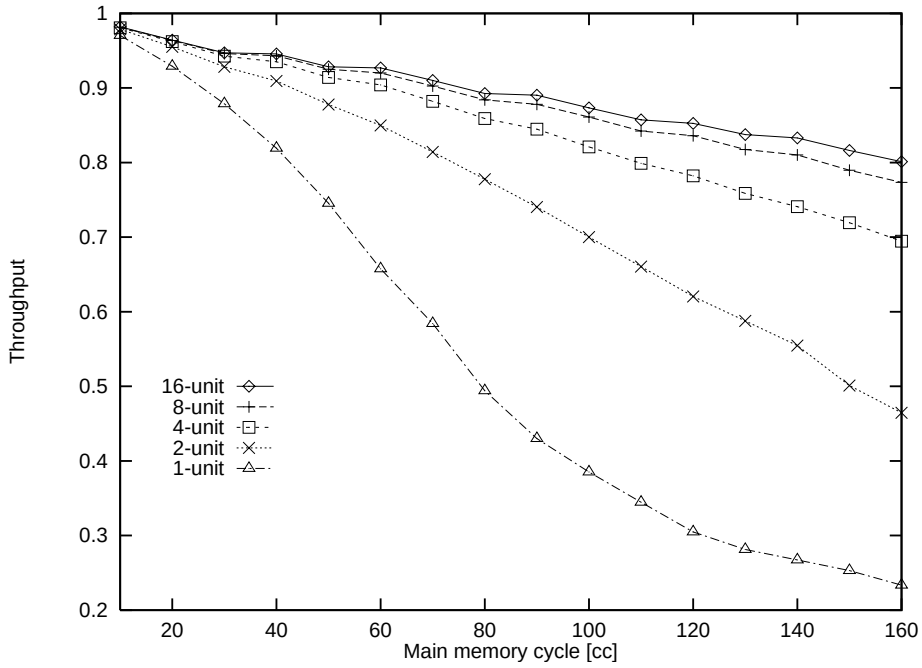


図 4.17 マルチユニット構成の効果 (spice)

結果から、メモリユニット数が少ないほど、スループットの低下の度合が低い主メモリサイクルから進行することが分かる。

ユニット数が 1 の時は、主メモリクロックサイクルが 30 クロックサイクル程度からスループット低下の度合が増加しているが、ユニット数が 2 の場合は、主メモリサイクルが 60 クロックサイクル程度からスループットの低下が増加していることからも分かる。

主メモリサイクルが 160 クロックサイクル程度では、ユニット数が 4 以上の場合ならそれほど大きなスループットの低下は見られなかった。主メモリサイクルが 160 クロックサイクルでも、スループットは 0.7〜0.8 程度と高い値を示していることが分かった。

ユニット数が 1 の場合、主メモリサイクルが増加するにつれスループットが他のものより大きく低下していくが、このカーブは図 4.13 のライトスルー方式のスループットの低

下のカーブによく似ていることが分かる。

つぎに、主メモリサイクルの増加に対するメモリ待ち列の変化を、メモリユニット数をパラメータとしてシミュレーションを行なう。

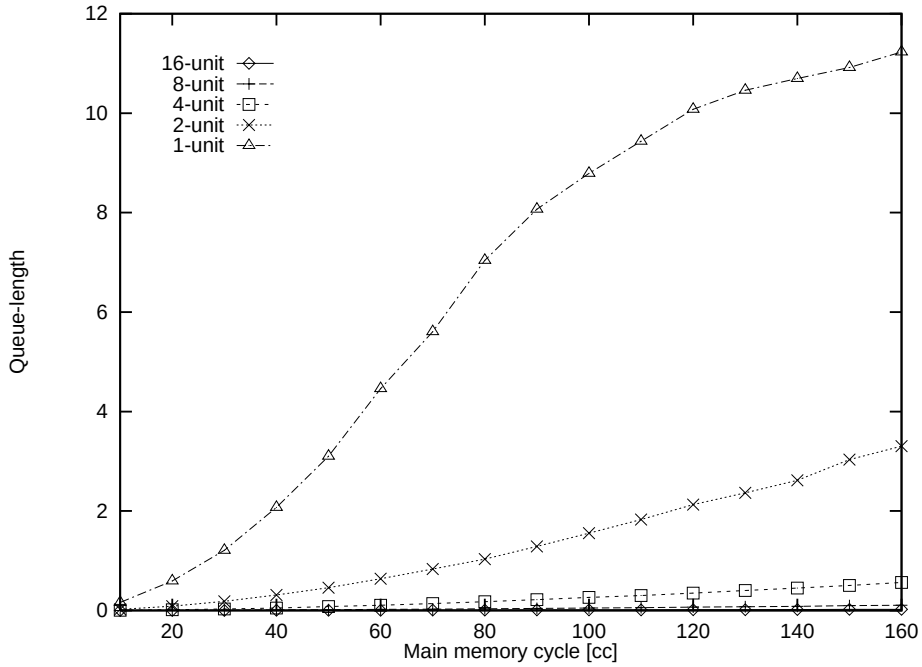


図 4.18 メモリのマルチユニット化とメモリ待ち列の長さ (spice)

図 4.17 と見比べると、スループットの低下に伴い、メモリ待ち列の長さが増加していくことが分かる。スループットのグラフとメモリ待ち列のグラフは対称的になっている。

結果から、主メモリサイクルの増加にともなって、メモリ待ち列が加速度的に増加していくことが分かる。メモリユニット数が増加していくにしたがって、メモリ待ち列の増加の立上りが鈍くなっている。メモリユニット数が 4 程度確保されている場合、主メモリサイクルが 160 クロックサイクルでも、待ち列の長さにほとんど変化は見られないことが分かる。

さらに、主メモリサイクルに対する、スレッドのスループットの分散を、メモリユニット数 1 の場合と 16 の場合について示す。

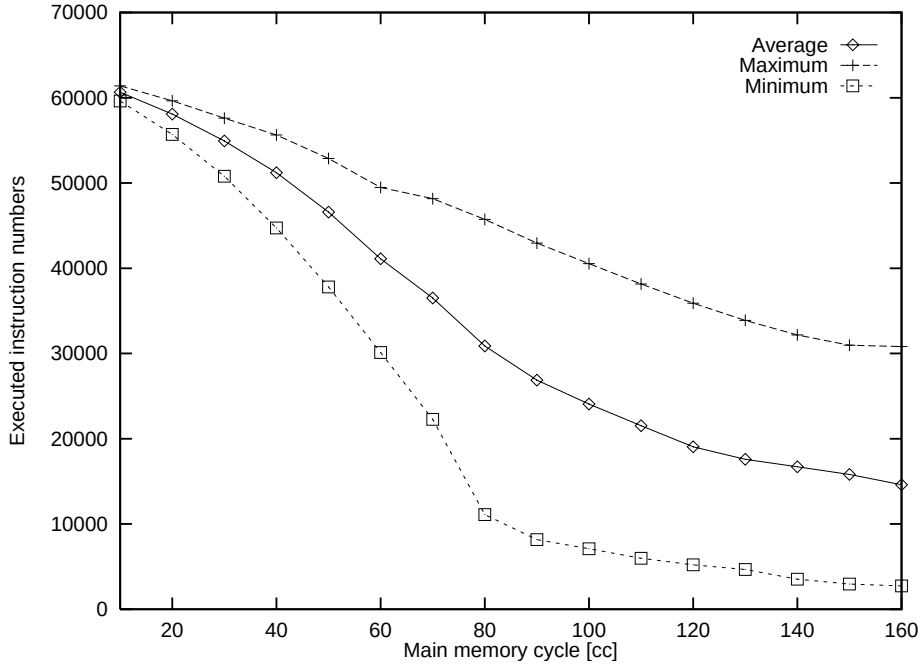


図 4.19 主メモリサイクルに対するスレッドのスループットの分散
(メモリユニット数：1、総ステップ数=1,000,000)

メモリユニット数 1 の場合、主メモリサイクルが 80 クロックサイクルをピークにして、分散の度合が大きくなっている。主メモリサイクルがそれ以上大きくなると、スレッドの平均スループットの低下は緩やかになり、それに応じて、分散の度合の増大も見られなくなっている。

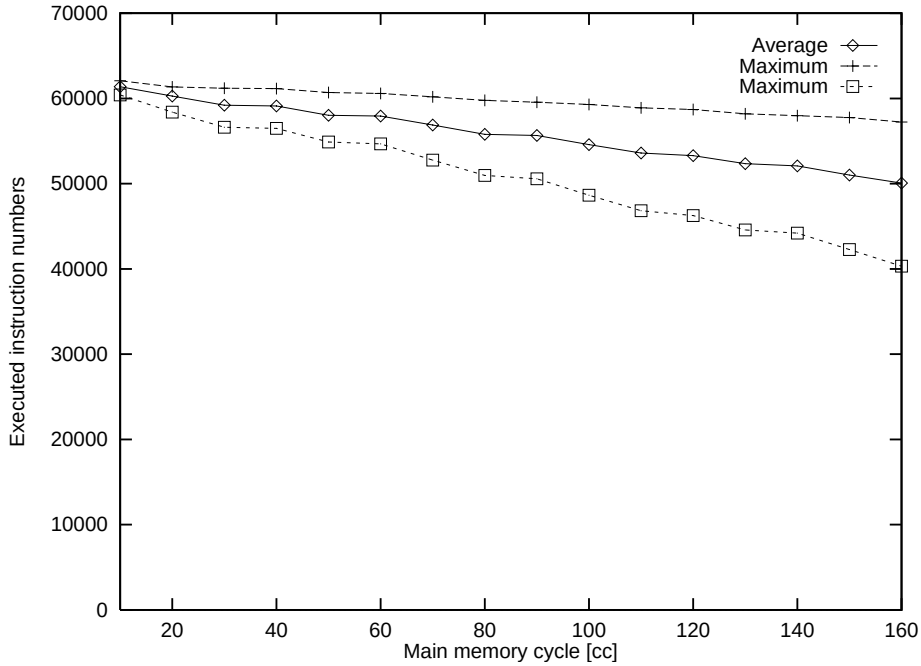


図 4.20 主メモリサイクルに対するスレッドのスループットの分散
(メモリユニット数：16、総ステップ数=1,000,000)

メモリユニット数が 16 の場合、主メモリサイクルが増加しても、スレッドの平均スループットの低下はほとんどなく、分散の度合が徐々に大きくなりつつあることが分かる。

第 5 章

考察

5.1 キャッシュ機構のパイプライン化に関する考察

5.1.1 キャッシュメモリの動作サイクルの短縮

キャッシュ機構のパイプライン化はレイテンシを増加させるが、クロックサイクルの短縮が実現され、プロセッサのスーパーパイプライン化のボトルネックになると考えられるキャッシュのスループットを稼ぐことができる。

またキャッシュメモリアレイを分割して物理寸法を小さくし配線遅延を解消することによって、キャッシュ機構のパイプライン化を現実的なものにすると同時に、キャッシュの大容量化も可能になると考えられ、それに伴うヒット率の向上も期待できる。ヒット率の向上はキャッシュミスペナルティの低下にそのままつながり、スループットの向上につながるものと考えられる。

5.1.2 ライトバック方式の採用とパイプライン化

3.2.3 節でも示されたようにメモリの書き込み方式をライトバック方式とすることでメモリトラヒックを低く抑えることができるが、問題点はキャッシュ機構が複雑化することでありレイテンシが増加する。しかし、キャッシュ機構のパイプライン化をさらにすすめることにより、ライトバック方式を採用してもキャッシュのスループットの低下にはつながらない。

このため、プロセッサ全体のスループットを向上させる上でキャッシュ機構のパイプラ

イン化は大きな役割を果たしているといえる。

5.2 キャッシュのヒット率に関する考察

5.2.1 スレッド共用キャッシュの効果

本研究で仮定しているプロセッサは16のスレッドを処理することを仮定しているので、キャッシュ容量も1つのスレッドが要求する容量の16倍のものを用意することを前提としている。

通常、1つのスレッドが独立したキャッシュを持っていたとすると、そのスレッドの要求するデータはそれ自身のキャッシュのみにマッピングされる。しかし、本研究では16のスレッドが16倍の容量を持ったキャッシュを共用するため、その広い空間を利用してマッピングを行うことになる。例えば1つ1つのスレッドが必ずしも同じアドレス空間を必要としていない。それと同時に、あるスレッドは1つあたりに用意されたスレッドの容量では不足することも場合によっては考えられる。そのため、互いが独立のキャッシュを持つシステムより、本研究で採用しているようなキャッシュ機構の方がキャッシュメモリ空間をうまく利用することができると考えられる。

結果からも分かるようにスレッド間どうしでのブロックの競合は影響していないどころか、局所性の高いspiceやtexにいたっては、キャッシュを各スレッドが共用する場合ヒット率が向上している。これは、同一スレッド間でもキャッシュのブロック競合が頻繁に発生しており、また、プログラムの局所性が高いために、競合によって追い出されたブロックが再び参照される確率が高くなっているためである。

このように、プログラムの局所性が高い場合は共用キャッシュが有利であるということがいえる。これは、キャッシュをスレッド間で共用することによって独立でキャッシュをもつ場合の総和に対して小さな容量を持つことが可能であると考えられる。

5.2.2 マルチスレッドとセットアソシアティブの連想度

次に命令キャッシュにおいて、プログラムの局所性が低くなるにつれて、キャッシュをセットアソシアティブ化した方がダイレクトマップ方式よりもヒット率が高くなる傾向がある。プログラムの局所性が低い場合は、アドレス参照範囲が広くなるため、キャッシュの

メモリ空間を多く必要とする。このため、キャッシュ容量を一定としてスレッド共用キャッシュとした場合、プログラムの局所性が低い場合は、局所性が高い場合に比べて各スレッド間でのブロック競合が発生しやすい状況になる。この時、ある程度の連想度を持たせることにより、同一ラインでのキャッシュ競合を防いでいるためである。局所性の低いプログラムは、逆にいえばある程度局所性のあるブロックを多数発生するわけで、それらのブロックは当然近いうちに参照される。そのブロックが追い出されていて再び参照された時はまたそのブロックにおいてミスが発生することになる。その競合が互いに行われていたとしたら、ヒット率が悪化すると考えられる。図 4.11 での cc の場合、ダイレクトマップでのヒット率がセットアソシアティブに比較して悪い結果を示しているのはまさにこのことが現実に起こっている。

プログラムの局所性が高くなると、連想度を高くしてもヒット率をほとんど向上させることはできなくなっている。これは、各スレッドのアドレス参照範囲が広くないため、各スレッド間でのブロック競合が少ないため連想度をあげる意味がないためである。このブロック競合の少なさはキャッシュのヒット率を全体的に押し上げていることが結果からも分かる。

データキャッシュにおいては全般的に連想度を 2 以上持たせることによって良好な結果が得られている。データアドレスはデータセグメントとスタックセグメントに分かれているためアドレス参照の範囲は広いが、個々の参照に関しては命令アドレスと比較して局所性が高くなっているので、全般的に連想度をある程度持っているとヒット率が良好な値を示しているものと思われる。

両者のヒット率で見ていくと、プログラムの局所性が異なっていても連想度 2 の場合が全体的に良好な結果をあげている。これは連想度から起こる競合とセット数から起こる競合との間で連想度 2 の場合両者のバランスがうまくとれているためであると考えられる。

5.3 キャッシュのスループットに関する考察

5.3.1 ミスペナルティとスループット

メモリアクセスペナルティが増大するにつれてスループットが急激に落ち込んでくる。これは、3.1 式に示したメモリサイクル利用率が増加するためである。ライトスルー方式の場合の主メモリへのアクセス確率は、表 4.5 から 11% である。この場合、メモリユニッ

ト数は 1 なので、約 9.1 クロックを越えるとメモリサイクル利用率は 1 を越えることになる。また、ミスを起こしてスレッドが実行を停止する割合がこの付近で急激に増加する。このため、図 4.12 に示されているように、主メモリサイクルが 10 クロックサイクル以降のスループットの急激な低下になって現われている。

主メモリサイクルが 40 クロックサイクル程度でスループットの低下が緩んでいるのは、このあたりからスレッドの実行が停止する割合が減少するためである。

図 4.13 と図 4.14 からスループットの低下に伴うグラフの変化と、待ち列の長さがの変化が対称的になっている。これは、待ち列の長さを抑えることが、高いスループットを維持することになることを確実に表している。

図 4.14 でライトスルーの場合、待ち列の長さがスレッド数 16 を越えて長くなっているのは、ストア命令による主メモリへの書き出しが行なわれているためである。この書き出しはキャッシュがヒットしてから行なわれるので、この時スレッドは停止しない。そのため、主メモリへのアクセス待ち列がスレッド数以上になる。

スループットの分散は図 4.15 では 30 クロックサイクルでもっとも大きくなっている。これは、主メモリサイクルが 30 クロックサイクル程度までは、キャッシュミスを起こしてメモリ待ち列にスレッドのアクセス要求が並ぶ割合が、急激に増加するためである。しかし、30 クロックサイクルを越えると、ミスを頻繁に起こすスレッドの要求がほとんどメモリ待ち列に並んでしまうので、キャッシュミスペナルティを多く受けるスレッドと、あまり受けないスレッドとのスループットの差はほぼ一定に落ち着く。

5.3.2 キャッシュの書き込み方式とスループット

またキャッシュの書き込み方式についてライトバック方式を採用することによりメモリアクセスペナルティが大きくなった場合のスループットの低下がそれほど見られなくなった。これは、ライトバック方式を採用することにより、主メモリへのアクセス確率が劇的に減少するためである。

表 4.5 から、ライトバックを採用した場合、ライトスルーの 10.8%から 2.1%に主メモリアクセス確率が大きく減少していることが分かる。ライトバック方式では、キャッシュミスを起こしブロックを読み出す時にのみ、主メモリへのアクセスが行なわれる。ライトスルー方式とライトバック方式の間で大きく異なる点は、主メモリへの書き出しの処理である。ライトスルーの場合は、主メモリへの書き出しはストア命令が行なわれるたびに発

生していたが、ライトバックの場合は、キャッシュの置き換え対象ブロックを主メモリへ書き出す時のみである。この差は表 4.5 に示したように、ライトスルーの 8.4%に対して、ライトバックの 0.1%未満と大変な差が生まれてくる。

ライトバック方式を採用することによって、主メモリへのアクセス確率が飛躍的に減少する。このため、メモリ待ち列の長さも大幅に短縮されている。メモリ待ち列には、ライトスルー方式のように 1 ワード単位の主メモリへの書き出し要求が並ばないため、ブロック読み出しなどのキャッシュミスペナルティは小さくなる。それが高いスループット維持に反映されている。

5.3.3 マルチユニット構成の効果

ライトバック方式では、主メモリへのアクセス確率が 2.1%であった。そのため、主メモリをマルチユニット化した場合、一様に各ユニットにアクセスが行なわれると仮定すると、ユニット数が 2 の場合 1.07%、ユニット数が 4 の場合 0.53%、という具合に各ユニットのアクセス確率が減少する。このため、各ユニットのメモリ待ち列の長さもそれに伴い短縮される。図 4.18 から、メモリユニット数を 2 倍にすることによって、メモリ待ち列の長さが 1/2 以下になっていることが分かる。この結果から、メモリ待ち列に並んでいる時のペナルティが、メモリユニット数を増加した割合以上に削減することができることが分かる。

また、メモリユニット数を 4 以上にすることによって、主メモリサイクルが 160 クロックサイクルになっても、0.7 以上のスループットを確保できている。これは、図 4.18 からも分かるように、メモリユニット数を 4 以上にすることによって、メモリ待ち列の長さが 1 以下に抑えることができている。これは、自分の前に主メモリへアクセスしているスレッドによるメモリ待ち列での影響がないことを意味する。このため、この場合の主メモリサイクルの増加に伴うスループットの低下は、単に主メモリサイクルの増加によるものである。

5.3.4 主メモリアクセスの呼源

主メモリアクセスの呼源は、各スレッドによる主メモリへのアクセス要求である。このアクセス要求は、スレッドがキャッシュミスをした時と、主メモリへのライト時に発生す

る。キャッシュミスを起こしたスレッドは実行を停止する。一方、主メモリへのライトによってスレッドの実行は停止しない。

キャッシュ書き込み方式がライトスルーの場合、表 4.5 からも分かるように、主メモリへのアクセスは約 80%が主メモリへのライトによるものである。主メモリへのアクセスが頻繁に発生しても、呼源は減少しないので、ライトスルー方式の呼源モデルは、無限呼源に近いモデルとなる。これにより、メモリ待ち列が急激に増加し、それにともないスループットが急激に低下している。

それに対して、キャッシュの書き込み方式がライトバック方式の場合、主メモリへのアクセスは、ほぼキャッシュミスによって発生している。主メモリへのアクセスが発生するほど、呼源は減少するので、ライトバック方式の呼源モデルは、有限呼源に近いモデルとなる。これにより、メモリの待ち列の増加が緩やかであり、それにともないスループットの低下も緩やかなものとなっている。

ライトバック方式のスループットの変化は、1 次的に有限呼源モデルに近似できるが、主メモリへのアクセスはキャッシュミスだけでなく主メモリへのライトによっても発生しているので、厳密に近似はできない。

第 6 章

結論

高度なパイプライン化によってますます高性能化するプロセッサに対して、本研究ではマルチスレッド型プロセッサ向きのキャッシュ機構のパイプライン化を提案した。

キャッシュ機構のパイプライン化はレイテンシを増加させるがスループットを向上させることができ、またキャッシュメモリアレイを分割して物理寸法を小さくし配線遅延を解消することによって、キャッシュのパイプライン化およびプロセッサのさらなるパイプライン化を現実的なものにしている。また、キャッシュ機構のパイプライン化の実現によって、本研究のシミュレーションでも有効な結果を示した、キャッシュのセットアソシアティブ化やライトバック方式といった、高度なキャッシュの構成にも対応することが可能である。

また、スレッド共用キャッシュにすることによる、時間的ローカリティの問題もほとんどなかったことが分かった。プログラムの局所性が高い spice や tex でのスレッド共用キャッシュのヒット率は、スレッド専用キャッシュに比較して高い値を示したほどである。マルチスレッド型プロセッサ上でのキャッシュは、スレッド数分の容量を用意する必要があるので、スレッド共用キャッシュにすることによってキャッシュ容量を小さくできる可能性があることは、魅力的である。

本研究でのキャッシュメモリの構成、実験を通じて、キャッシュのヒット率のみで正しい性能の評価はできないということができる。同じキャッシュヒット率でも、主メモリサイクルが異なればスループットに大きく影響することが本研究でも明らかに示されている。

キャッシュのヒット率はある限られたメモリ空間のどこにデータをマッピングするかというところのみに着目したパラメータである。しかし、本研究で使用を前提としているマルチスレッド型プロセッサからは各クロックサイクル毎にキャッシュメモリへの要求があ

るため、前にミスを起こしたスレッドのペナルティが終了する前に別のスレッドがミスを起こせば前のミスペナルティをも被ることになる。そのミスペナルティは、主メモリサイクルだけではなく、メモリ要求待ち列の長さによっても大きくなる。

すなわち、マルチスレッド型プロセッサの高性能化のためには待ち列の発生を抑えることが重要であり、それにはメインメモリの性能が重要であるということがいえる。しかし、メインメモリの速度向上の比はプロセッサの速度向上の比には追い付けていない。そのため、メインメモリをマルチバンク化などメモリのバンド幅を広げることが全体のシステムの性能を向上させる鍵となる。

謝辞

本研究を進めるにあたり、終始熱心かつ寛容な御指導を賜わりました、日比野 靖教授に心から感謝いたします。

また、適切な御助言をしていただきました横田 治夫 助教授、丹 康雄 助手、堀口 進 教授に深く感謝いたします。

さらに、マルチスレッド型プロセッサに関する御助言をいただいた伊藤 英治君に感謝いたします。

その他、貴重な御意見、御討論をいただきました日比野・横田研究室の皆様をはじめ、多くの方々の御助言に対し厚く御礼申し上げます。

本研究に関する論文

I. 相原 孝一, 伊藤 英治, 丹 康雄, 日比野 靖 “マルチスレッド型プロセッサ向きのキャッシュメモリの構成と評価”, 情報処理学会研究報告, 97-ARC-123, Vol.97, No.22, pp.37-42, 1997

II. 伊藤 英治, 相原 孝一, 丹 康雄, 日比野 靖, “関数型プログラムの実行に適したマルチスレッド型プロセッサ・アーキテクチャの提案”, 情報処理学会研究報告, 96-ARC-121, Vol.96, No.121, pp.81-88, 1997

参考文献

[1] Toshihiko Hirose, et al., "A 20ns 4Mb CMOS SRAM with Hierarchical Word Decording Architecture", *ISSCC DIGEST OF TECHNICAL PAPERS*, pp.132-133, 1990.

[2] 井口秀之, "次世代集積技術による RISC アーキテクチャプロセッサの設計と評価", 北陸先端科学技術大学院大学修士論文, 1995

[3] Koichiro Ishibashi, et al., "A 300MHz 4-Mb Wave-Pipeline CMOS SRAM using a Multi-Phase PLL", *ISSCC DIGEST OF TECHNICAL PAPERS*, pp.308-309, 1995.

[4] Daniel C. McCrackin, "Eliminating Interlocks in deeply Pipelined Processor by DelayEnforced Multistreaming", *IEEE Trans. on Computer,* Vol.40, No.10, pp.1125-1132, Oct 1991.

[5] Won Woo Park, Donald S. Fusell and Roy M. Jenevein, "Performance Advantages of Multithreaded Processors", sl Proc. of the Third Int'l Conf. on Parallel Processing, Vol.1, pp.97-101, 1991.

[6] David A. Patterson and John L. Hennessy, "COMPUTER ORGANIZATION & DESIGN THE HARDWARE/SOFTWARE INTERFACE", Morgan Kaufmann Publishers Inc, 1994.

[7] R. Guru Prasadh and Chuan-lin Wu, "A Benchmark Evaluation of a Multi-threaded RISC Processor Architecture", sl Proc. of the 20th Int'l Conf. on Parallel Processing, Vol.1 pp.84-91, 1991.

[8] MIPS Technology Inc. "R4400 MICROPROCESSOR PRODUCT INFORMATION", 1996.

[9] B.Wilkinson, “Computer Architecture : Design and Performance”, Prentice Hall, 1991.

[10] 牧野 都治, “待ち行列の応用”, 森北出版, 1969.

付録 A

実装

スレッドがキャッシュ上に現われた時に行なわれる処理を、プログラム上でどのように実装しているか説明する。説明が行なわれているパイプライン中の注目すべきスレッド (M) に網かけを施す。また、そのパイプラインステージで動作するユニットにも網かけを施す。

1. **命令キャッシュ**

- アドレスのデコード (IF1)

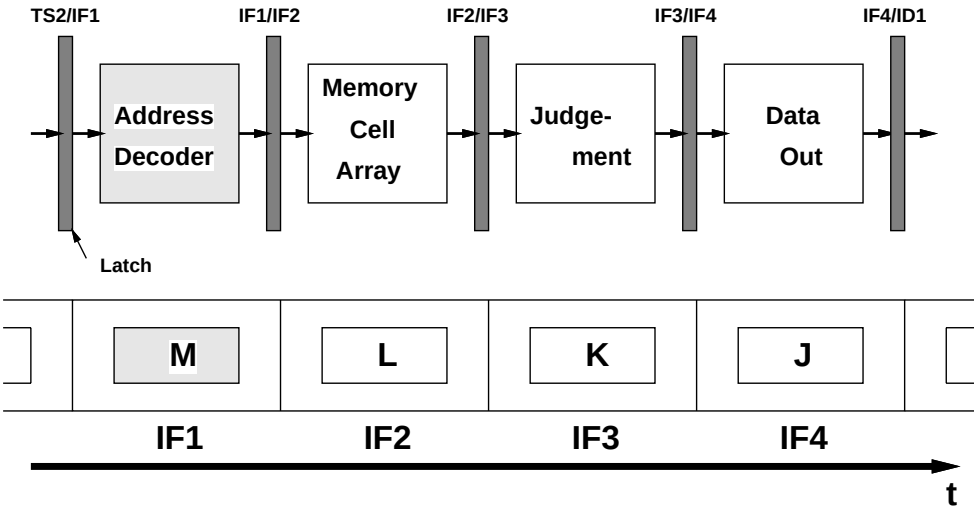


図 A.1 アドレスデコードのパイプラインステージ

スレッド M が命令フェッチを要求してこのステージに現われた時、読みだしアドレスが本来デコードされる。本シミュレーションでは、図 4.3 に示すように、元々デ

コード済のトレースデータが用意されているので、シミュレーション中ではこの場合何も行なわれていない。

それに対し、スレッド M が命令キャッシュでキャッシュミスを起こした場合、図3.11に示したようにバッファを参照しに行く。このバッファにアクセス許可フラグ (後述) がたっていたら、ミスを起こしたアドレスにオフセットを加え、命令キャッシュの該当するブロックを更新する。

キャッシュをセットアソシアティブ化している時は、どちらのセットを更新するかという問題がある。本シミュレーションでは、このセットの選択をランダムに決定している。

また、ブロックの更新については、キャッシュミスを起こしたアドレスが該当するブロック全体を更新するようにしている。

- メモリセルアレイの読み出し (IF2)、および判定 (IF3)

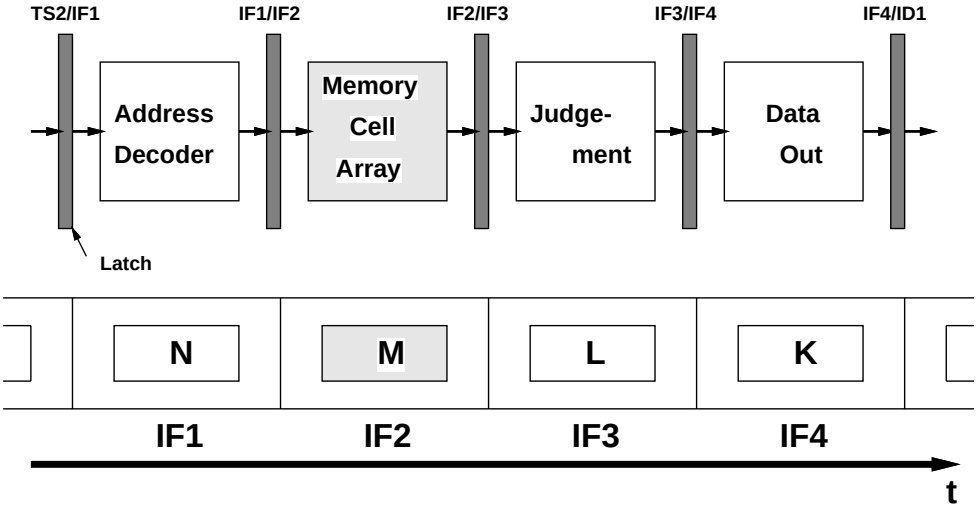


図 A.2 メモリセルアレイ読み出しのパイプラインステージ

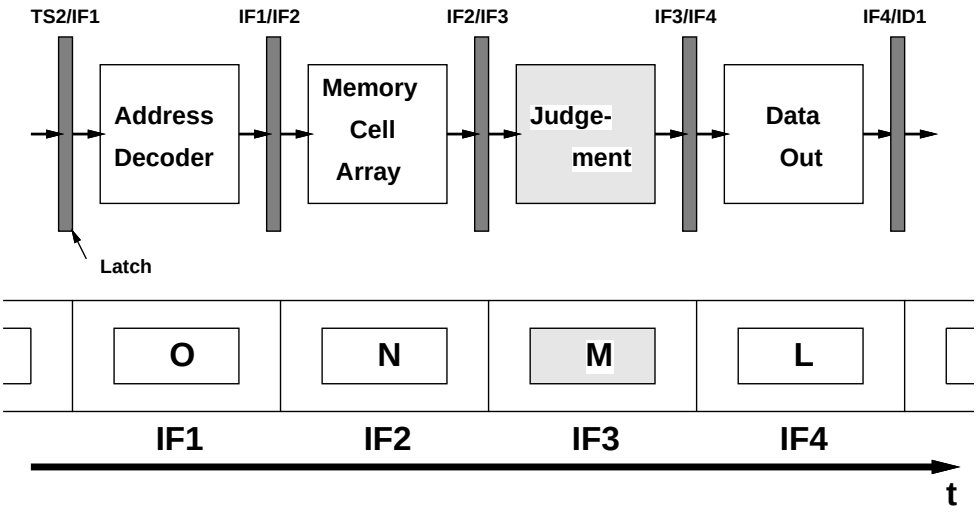


図 A.3 判定のパイプラインステージ

本シミュレーションでは、メモリセルアレイの読み出しと判定ステージに 1 クロックサイクル要すると仮定している。

シミュレーションでは、判定ステージで行なわれる処理を次のデータ排出ステージ (IF4) でまとめて行なっている。このため、プログラム上ではスレッドがこの両ステージに現われた時は、クロックカウンタを繰り上げることのみ行なっている。

- データ排出 (IF4)

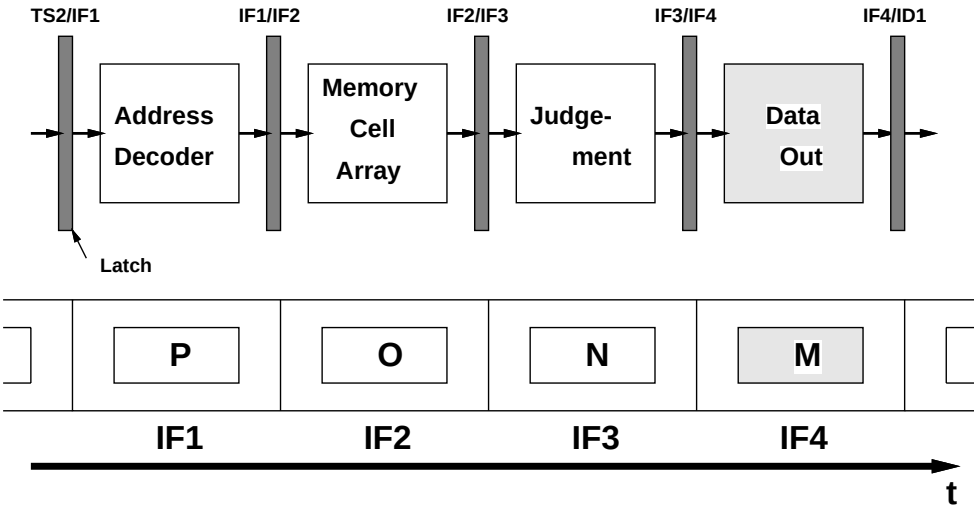


図 A.4 データ排出のパイプラインステージ

本来はデータを排出するためのアンプから構成されているが、本シミュレーションでは、このデータ排出ステージでキャッシュに関するほとんどの処理を行なっている。

まず、各スレッドのアドレスの系列は図 4.3 に示すようになっている。図 4.3 を見ると、命令キャッシュの読み出しが連続して起こっているものもあれば、命令キャッシュの読み出しの後にデータキャッシュの読みだし、あるいはデータキャッシュの書き込みが起こっているものもある。命令キャッシュの読み出しの後に連続して命令キャッシュの読み出しが起こっているのは、前の読み出しがデータキャッシュへのアクセスを必要としていないためである。このため、命令キャッシュで命令フェッチが成功して、データキャッシュステージにこのスレッドが現われた時に、何も方策を講じないとデータキャッシュステージで命令キャッシュの読み出しを行なってしまうという不都合が生じてしまう。このため、本シミュレーションでは、データキャッシュステージで命令キャッシュの読み出しが現われたら、このスレッドはデータキャッシュへのアクセスを必要としていないと認識し、このアドレスをスレッド専用のバッファ(ブロック更新用のバッファとは別のもの) に格納するようになっている。

このスレッドがこのパイプラインステージに現われた時、このバッファにアドレスを読みに行くようにしている。

前回このステージでキャッシュミスを起こしたスレッドが現われた時は、その時に格納されたアドレスを再び読み込んでキャッシュへアクセスするようにしている。キャッシュがヒットするまでこの手順は繰り返され、トレースデータから新たにアドレスを読み込むことはない。すなわち、キャッシュミス中はスレッドが停止することになる。

命令アドレスをデータステージで取り出した場合とキャッシュミス中の場合を除いて、スレッドはトレースデータを読み込む。

スレッドのアドレスにはオフセットが設定され、そのデータがキャッシュ中に存在しているか判定が行なわれる。もし、キャッシュがヒットならばスレッドはそのまま 2 段の命令デコードステージ、および 2 段の演算ステージを経て、データキャッシュに向かう。

キャッシュがミスだった場合は、命令キャッシュミスフラグをたて、バッファにミスを起こしたアドレスを格納する。次回、このスレッドがこのステージに現われた時は、バッファにこのアドレスを読み込むことになる。また、キャッシュミスの際は主メモリへのアクセスを要するので、メモリ待ち列にスレッド番号とアクセス種別を格納する。

- **メモリがマルチユニット化されている場合**

  メモリユニットへの振り分けは、アドレスの下位3ビットから5ビットの合計3ビットによって行なわれる。

2. データキャッシュ

- アドレスのデコード (DF1)

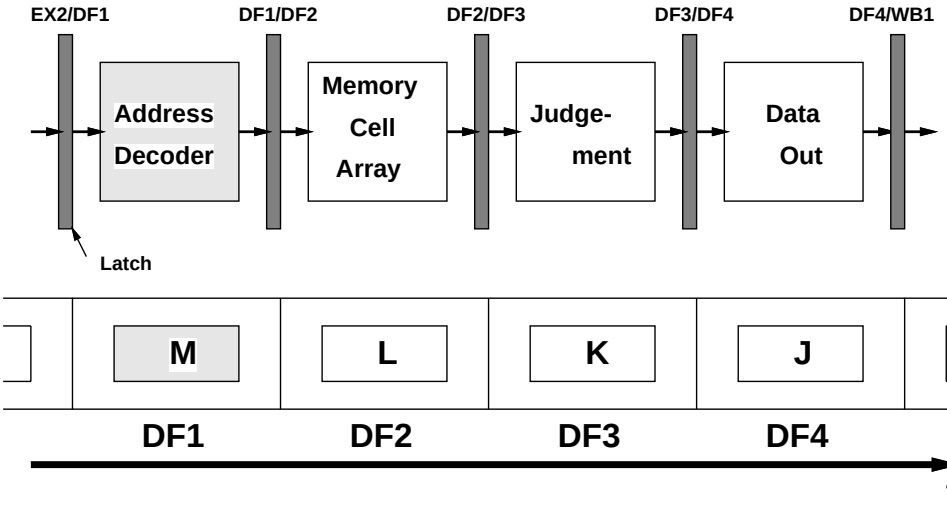


図 A.5 アドレスデコードのパイプラインステージ

スレッド M が命令フェッチを無事済ませると、本来ならば命令をデコードした結果データキャッシュにアクセスが行なわれるかどうかが分かる。しかし、本シミュレーションでは、命令キャッシュの時も述べたように、トレースデータの命令キャッシュの読み出しの後に、データキャッシュの読み出し、あるいは書き込みが続いていたら、データキャッシュのアクセスが行なわれることが分かる。

スレッド M が命令キャッシュでキャッシュミスを起こしてこのステージに現われた場合、当然データキャッシュでの処理は行なわれないようになっている。

スレッド M がデータフェッチを要求してこのステージに現われた時、読み出しアドレスまたは書き込みアドレスが本来デコードされる。本シミュレーションでは、図 4.8 に示すように、元々デコード済のトレースデータが用意されているので、シミュレーション中ではこの場合何も行なわれていない。

それに対し、スレッド M がデータキャッシュでキャッシュミスを起こした場合、図 3.10 に示したようにバッファを参照しに行く。バッファにアクセス許可フラグがたっていたら、ミスを起こしたアドレスにオフセットを加え、キャッシュブロックを更新することになる。

ここで、データへのアクセスには、データセグメントへのアクセスと、スタックセグメントへのアクセスの 2 種類がある。トレースデータにはこれらのアクセス種別が用意されていない。このため、図 4.6 の仮想アドレス空間の 300000000 番地から下をデータセグメント空間、上をスタックセグメント空間とみなした。これをわける理由は、オフセットの与え方が異なるためである。データセグメントは上方へ拡張されるため、プラスのオフセットを加え、スタックセグメントは下方へ拡張されるため、マイナスのオフセットを加えている。

- メモリセルアレイの読み出し (DF2)、および判定 (DF3)

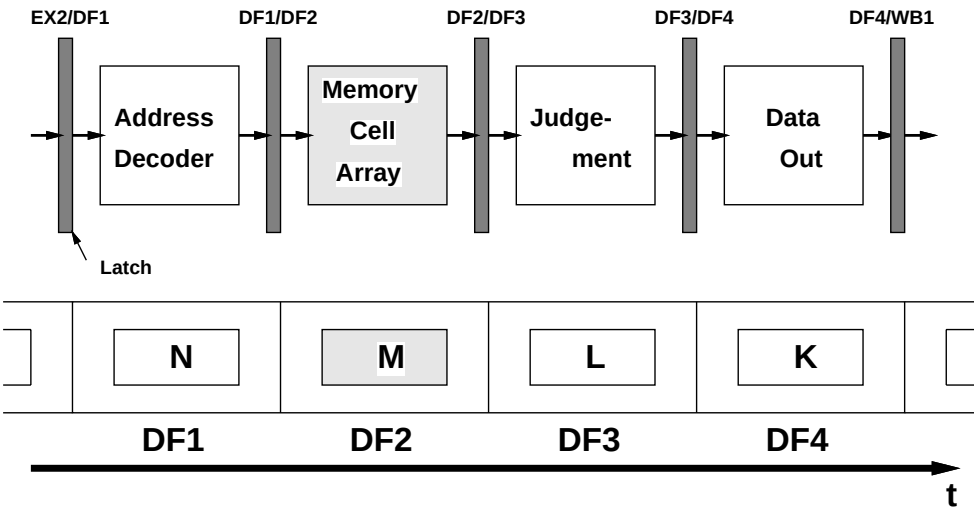


図 A.6 メモリセルアレイ読み出しのパイプラインステージ

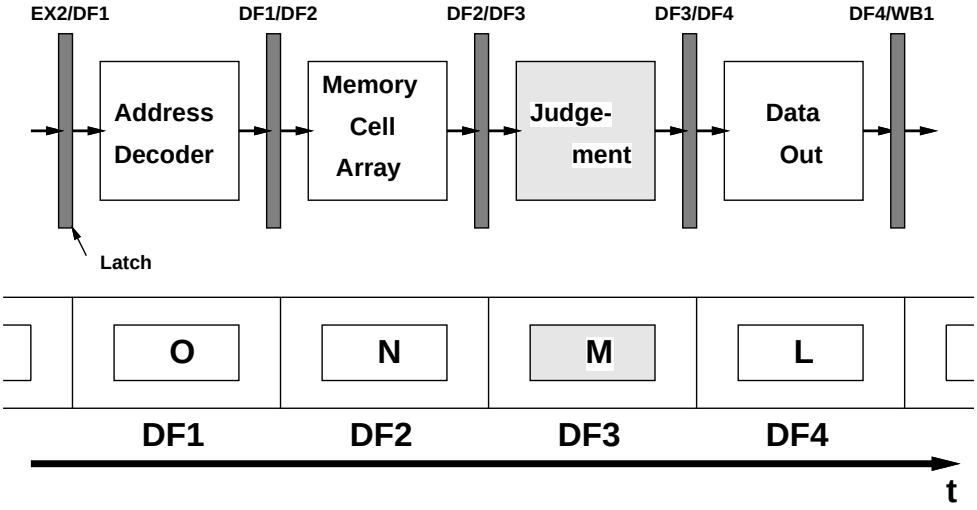


図 A.7 判定のパイプラインステージ

メモリセルアレイの読み出しと判定ステージは、命令キャッシュの場合と同様に 1 クロックサイクル要すると仮定している。

シミュレーションでは、データキャッシュも命令キャッシュと同じように、判定ステージで行なわれる処理を次のデータ排出ステージ (DF4) でまとめて行なっている。このため、プログラム上ではスレッドがこの両ステージに現われた時は、クロックカウンタを繰り上げることのみ行なっている。

- データ排出 (DF4)

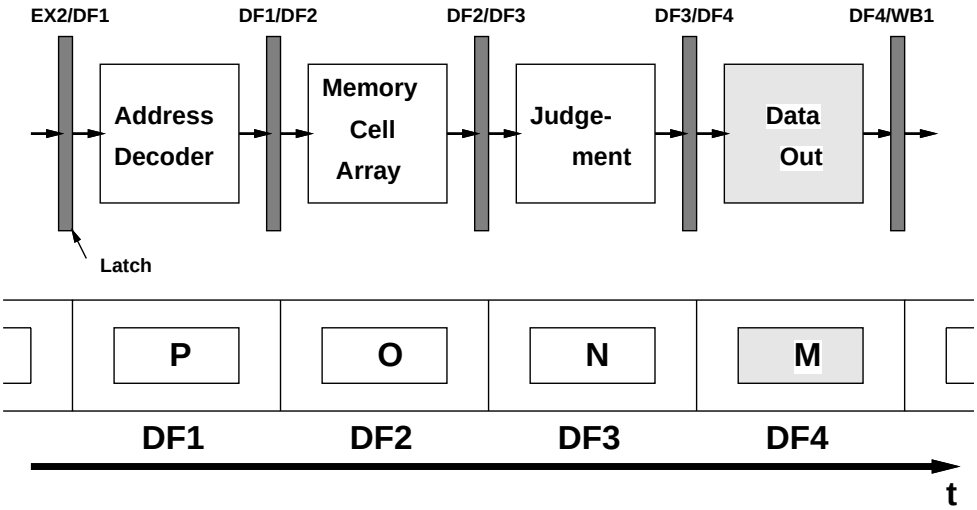


図 A.8 データ排出のパイプラインステージ

スレッド M が命令キャッシュでミスをして、このステージに現われたら、データキャッシュへのアクセスは当然行なわれない。

スレッド M がこのステージに前回現われてキャッシュミスを起こしていたら、その際に格納したアドレスを読み出してキャッシュに再アクセスする。

上記以外の場合は、トレースデータからアドレスを読み出す。

もし、その読み出したアドレスが命令アドレスの場合は、データキャッシュへのアクセスは行なわれないことになるので、そのアドレスを格納した後、スレッドは2段のライトバックステージを経て1命令を実行完了したとみなされる。

もし、その読み出したアドレスがストア命令だった場合は、それがデータセグメントへのアクセスか、スタックセグメントへのアクセスかを判断し、オフセットが加えられる。その後、データキャッシュに判定を求める。

ヒットが返された場合は、そのアドレスに該当するブロックがキャッシュに存在しているので、キャッシュにはデータの代わりにアドレスを書き込む。

キャッシュがセットアソシアティブ化されている場合は、ヒットした方のセットにアドレスを書き込む。

書き込みが終えたら、このスレッドは2段のライトバックステージを経て、1命令を実行完了したとみなされる。

- **ライトスルー方式の場合**

  書き込みは主メモリへも行なう必要がある。書き込みは1ワード単位で行なうので、書き込みアドレスの下位2ビットで、4バンクに振り分ける。メモリ待ち列には、スレッド番号とアクセス種別(この場合書き込み)、さらに、書き込みが行なわれるバンク番号が格納される。

本シミュレーションでは、書き込みミスの処理をライトアロケート方式としている。このため、書き込みミスが発生したら、主メモリからブロックを読み出して、書き込みを行なわなければならない。

もしキャッシュからミスが返されたら、データキャッシュミスフラグをたてる。このフラグがたてられたスレッドは、トレースデータからのアドレスの読み出しが禁止

されるので、停止することになる。そしてそのアドレスを格納する。さらにメモリ待ち列にスレッド番号と読み出し要求を格納する。

- **ライトバック方式の場合**

  書き込みヒット時は主メモリへの書き込みは行なわれない。キャッシュへ書き込みが行なわれたら、そのブロックは置き換え対象となる。これは、このブロックに対して、キャッシュと主メモリとの一貫性が保たれていないことを意味するものである。これを表すために、キャッシュにダーティビットが用意されている。ブロックに書き込みが行なわれると、ダーティビットがセットされる。

  書き込みミス時には、そのブロックが置き換え対象になっているかどうかを調べる必要がある。もし、置き換え対象になっていないなら、そのブロックとそれに該当する主メモリのブロックは一貫性が保たれているので、主メモリへのブロックの書き出しは行なわれない。しかし、置き換え対象になっている場合、主メモリとの一貫性が保たれていないことになるので、メモリ待ち列に書き出し要求を格納する。

  キャッシュがセットアソシアティブ化されている場合は、ランダムにセットを選択する。そして選択されたセットのブロックが置き換え対象になっているかどうか調べる。

  なお、本シミュレーションでは、キャッシュミス時のキャッシュへの書き込みペナルティを小さくするために、先にブロック読みだし要求を出して、その後に置き換え対象のブロックの書き出し要求を出している。順序をこのようにすることで、ブロック読みだし要求は、ブロック書き出し要求のメモリアクセスペナルティを被ることがなくなるためである。また、この手法が可能な理由は、メモリアクセスペナルティが 1 以上である限り、ブロックの更新は最低パイプライン数分のクロックサイクルがかかるため、このブロックは次のサイクルまで保持されるためである。

- **メモリがマルチユニット化されている場合**

  アクセスのユニットへの振り分けは、命令キャッシュの場合と同様に、アドレスの下位 3 ビットから 5 ビットの計 3 ビットによって行なう。

もし、トレースデータから読み出したアドレスがロード命令だった場合は、データ、およびスタックに対応したオフセットを加え、キャッシュの判定を行なう。

ヒットが返された場合は、データ読みだし完了とみなし、このスレッドは2段のライトバックステージを経て1命令の実行完了とみなされる。

ミスが返された場合は、データキャッシュミスフラグがたてられる。そして、そのアドレスを格納する。さらに、その該当するブロックを読み出すため、メモリ待ち列にスレッド番号と、ブロック読みだし要求を格納する。

- **ライトバック方式の場合**

  もし、そのブロックが置き換え対象になっていたら、メモリ待ち列にスレッド番号とブロック書き出し要求を格納する。

  キャッシュがセットアソシアティブ化されている時は、ランダムにセットを選択して、そのブロックが置き換え対象になっているかどうかを調べる。

  なお、本シミュレーションでは、キャッシュミス時のキャッシュへの読み出しペナルティを小さくするために、ストア命令の場合と同様、先にブロック読み出し要求を出して、その後に置き換え対象のブロックの書き出し要求を出している。これが可能な理由は、ストア命令の場合と同様である。

- **メモリがマルチユニット化されている場合**

  アクセスのユニットへの振り分けは、アドレスの下位3ビットから5ビットの計3ビットによって行なわれる。

3. メモリ待ち列から主メモリへの受渡し

以下に述べる処理はクロックサイクル毎に行なわれている。

まず、メモリ待ち列に格納されている、先頭のスレッド番号が調べられる。メモリがマルチユニット化されている場合は、すべてのメモリ待ち列の先頭のスレッド番号が調べられる。

つぎに、メモリのアクセス許可が調べられる。

ライトスルーの場合は、各メモリバンクのアクセス許可が調べられる。

メモリをマルチユニット化している場合は、各ユニットのアクセス許可が調べられる。

- **ライト スルー方式の場合**

  メモリ待ち列に格納されている先頭のアクセス種別が読み出しの場合、読み出しはブロック単位で行なわれるので、各バンクがアクセス許可状態でないと主メモリへのアクセスは行なうことができない。もし、すべてのバンクでアクセスが可能なら、各メモリバンクにスレッド番号とアクセス種別を格納する。この要求はメモリアクセスペナルティを経て、主メモリからあがってくる。

  メモリ待ち列に格納されている先頭のアクセス種別が書き込みの場合、バンク番号が格納されている。もし、これに対応するメモリバンクがアクセス可能なら、そのメモリバンクに、スレッド番号とアクセス種別を格納する。この要求は、メモリアクセスペナルティを経て終了する。

- **ライト バック方式の場合**

  主メモリへの読みだし、および書き込みは、ブロック単位で行なわれるので、各バンクのアクセスはすべて同じタイミングで発生することになる。

要求が受け入れられると、メモリ待ち列からその要求が削除される。そして次の要求が待ち列の先頭に現われるように、待ち列の中で組み替えが行なわれる。

4. 主メモリからスレッド専用バッファへの受渡し

メモリユニットのアクセスが終了すると、そのアクセス種別が調べられる。アクセス種別が書き出しの場合は、その時点でアクセス完了となる。アクセス種別が読み出しの場合は、そのアクセスがどのスレッドにより起こったかを調べ、そのスレッド専用のバッファのアクセス許可フラグをたてる。許可フラグがたてば、キャッシュのブロック更新が可能となる。

付録 B

シミュレーションの結果出力

次ページ以降に、主メモリをユニット構成としたシミュレーションの出力結果を示す。
各結果のファイル名は以下のようになっている。