# 1 はじめに

マルチプロセッサ構成のシステムにおいて、ユーザが容易にシステムの性能を最大限に引き出すためには、逐次的に記述されたプログラムから並列性を抽出し、各プロセッサにタスクを自動的にスケジューリングする並列化コンパイラが必要である。これまでも、これらコンパイラ、スケジューラの研究が行なわれてきたが、いずれも既存のマルチプロセッサシステムに合わせたスケジューリングを行なうため、性能の向上には限界があった。そこで、我々は発想を逆転し、コンパイラやスタティックスケジューリングに合わせてマルチプロッサの構成を基本から設計することを考えた。これにより、コンパイラの性能を十分に引き出す事が可能になり、マルチプロセッサの性能がさらに向上すると考える。

このようなアーキテクチャとして、われわれはマルチプロセッサ ASCA (Advanced Scheduling oriented Computer Architecture) を提案している。このマルチプロセッサシステムは、現在の自動並列化コンパイラが主に行なっている並列化手法であるループレベルの並列化だけでなく、粗粒度、近細粒度並列処理と呼ばれる新しい並列処理手法を組み合わせたマルチグレイン並列処理 [1][2] を効果的に行なえるようなアーキテクチャ構成を取る。このマルチグレイン並列処理手法により、逐次プログラムの並列性を様々な粒度で抽出することが可能となる。

マルチプロセッサシステム ASCA が効率良くマルチグレイン並列処理を行なうためには、コンパイル時にデータの転送時間や命令の実行時間を出来る限り予測可能である事が望ましい。更に近細粒度並列処理を無同期で行なわせるためには、これらの時間がコンパイル時に完全に予測可能でなければならない。このため、動作予測を完全に行なうことのできない商用のプロセッサ、ネットワークを利用することは難しい。そこで、これらの各要素を独自に設計する必要がある。

本研究では、この中からプロセッサの動作予測を困難にしている原因のひとつであるキャッシュについて検討を行なう。

# 2 マルチグレイン並列処理手法

マルチグレイン並列処理手法 [1][2] とは、基本ブロック (BB) やループ、サブルーチン間の並列処理手法である粗粒度並列処理手法（マクロデータフロー処理手法）[3][4]、ループレベルの並列処理である中粒度並列処理手法、BB 内部の並列性を利用する近細粒度並列処理手法 [5] とを階層的に組み合わせて効率良く並列処理をする手法である。

## 2.1 マクロデータフロー処理 (粗粒度並列処理)

マクロデータフロー処理では、プログラムはマクロタスク (MT) に分割される。マクロタスクはブロック途中への飛び込みが無いステートメントの集合からなるブロックである。その形状から疑似代入文ブロック (BPA)、繰り返しブロック (RB)、サブルーチンブロック (SB) に分けられる (図 1)。BPA は本質的には基本ブロックであるが、並列性抽出のために単一の基本ブロックを複数に分割し複数の BPA を生成したり、逆に単一の基本ブロックの処理時間が 1 回のダイナミックスケジューリングに要する時間に比べ短い場合には、複数の基本ブロックを融合し 1 つの BPA を生成する。RB は最外側のナチュラルループである。サブルーチンブロックは、基本的にインライン展開を行なうが、コード長を考慮し効果的にインライン展開できない場合は SB として定義する。

プログラムをマクロタスクに分割した後、マクロタスク間のコントロールフロー、データ依存を解析し、それらを表すマクロフローグラフ (MFG) を生成する [1]。さらに MFG からマクロタスク間の並列性を最大限に引き出すために、各マクロタスクの最早実行可能条件を解析する。この各マクロタスクの最早実行可能条件は、マクロタスクグラフ (MTG) で表すことができる [1]。

そして、マクロデータフロー処理では全プロセッサを分割して得られるプロセッサクラスタ (PC) に MT を割当てるが、条件分岐のような実行時まで確定できない問題に対処するため、マクロタスクを実行時に PC に割り当てるダイナミックスケジューリングを行なう。

```
      INTEGER I,J,K,L,M,N
      REAL A(100),S
      DO 10 I = 1,100
       A(I) = 0.0                  MT1
10    CONTINUE
      K = 2
      L = 3
      M = K * L                    MT2
      N = M ** 2
      IF(N.EQ.0)THEN
       S = 1.0E - 10               MT3
      ELSE
       S = N                       MT4
      ENDIF
      DO 20 J = 2,100
       A(J) = A(J-1) + S           MT5
20    CONTINUE
30    S = S - 1.0                  MT6
      IF(S.GE.0.0) GOTO 30
      WRITE(*,*)S                  MT7
      STOP
      END
```

図 1: MT 分割の例

## 2.2 ループ並列化 (中粒度並列処理)

PC に割り当てられたマクロタスクが Do-all 可能な RB である場合、この RB は PC 内のプロセッシングエレメン

ト (PE) によって中粒度すなわちイタレーションレベルで並列化される (図 2)。

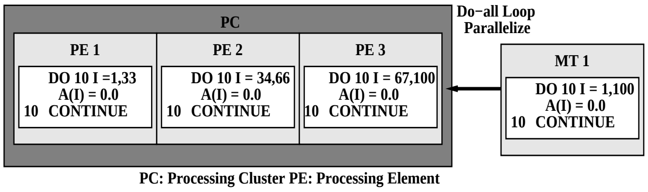


図 2: ループ並列化

### 2.3　近細粒度並列処理

マクロタスクが BPA やループ並列化できないループであった場合、マクロタスク内部はステートメントレベルで並列化され、PC 内部の PE によって並列処理される (図 3)。

このような近細粒度タスク集合の PC 内 PE への割り当てには、データ転送時間を考慮したヒューリスティックアルゴリズムである CP/DT/MISF 法あるいは DT/CP 法 [6] を用いる。これはコンパイル時にスケジューリングを行なうスタティックスケジューリングである。近細粒度並列処理においてはこのスタティックスケジューリングをマシンクロックレベルで厳密に行なう事により、全ての同期を除去した無同期近細粒度並列処理 [5] の適用が可能である。

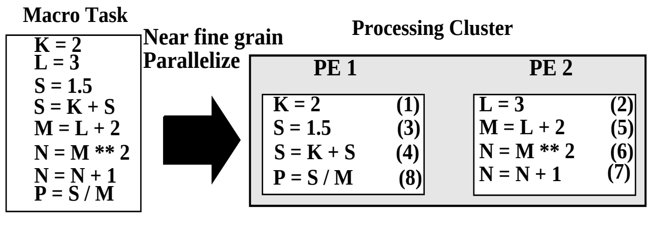


図 3: 近細粒度並列化

## 3　ASCA の全体構成

マルチグレイン並列処理を効果的に実現するための並列計算機として、われわれはマルチプロセッサシステム ASCA を開発している。ASCA の全体構成を図 4に示す。ASCA は、4 または 8 個のプロセッシングエレメント (PE) から構成されるプロセッシングクラスタ (PC) と共有変数などの格納に使用する集中共有メモリ (CSM:Centralized Shared Memory) を仮想的な階層型多重バスで結合したマルチプロセッサシステムである。

### 3.1　ASCA の結合網

共有バスは最も簡単な結合方式であり、どのプロセッサがバスの所有権を得るかのみを考慮すればバスの混雑を防止することができるため、スケジューラからの制御が最も容易である。また、早稲田大学で開発された OSCAR[7] は

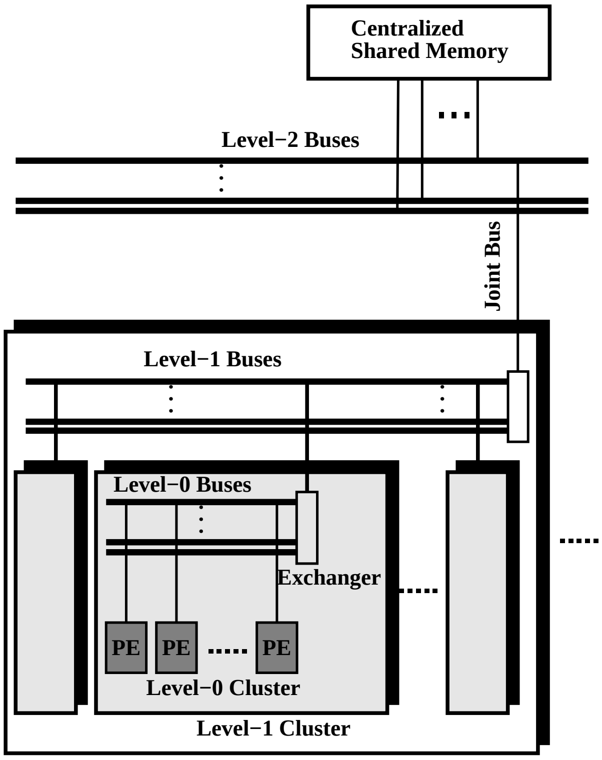


図 4: ASCA の構成

3 本の共有バスを持ち、これらのバス上での転送順序のスケジューリングをクロックレベルで行ない、データ転送を含む基本ブロック内の同期コードを全て除去しても正しい計算結果の得られる無同期近細粒度並列処理を行なうことができることが確認されている [5]。しかし、実際の共有バスは、多数の PE を結合する方法としては実装上の問題が大き過ぎる。

そこで、ASCA では、コンパイラ/スケジューラから見た結合モデルと実際の結合網を分離する。コンパイラ/スケジューラ向きのモデルとしては、階層型多重バスモデルを採用する。そして実際の ASCA の結合網には階層型多重バスをエミュレート可能な多段結合網 R-Clos 網 [8] を用いることにした。

### 3.2　プロセッシングエレメント

プロセッシングエレメント (PE) は、図 5のようにネットワークインタフェース、分散共有メモリ (DSM:Distributed Shared Memory)、ローカルメモリ (LM:Local Memory) およびこれらのメモリを制御するメモリコントローラ、PE 外部とのデータ通信を行なうダイレクト・メモリ・アクセス・コントローラ (DMA) そして ASCA 専用プロセッサである MAPLE[9] から構成される。

MAPLE は、コンパイル時に動作予測を容易にし、実行時不確定性をできる限り排除する事を目的としている。又、

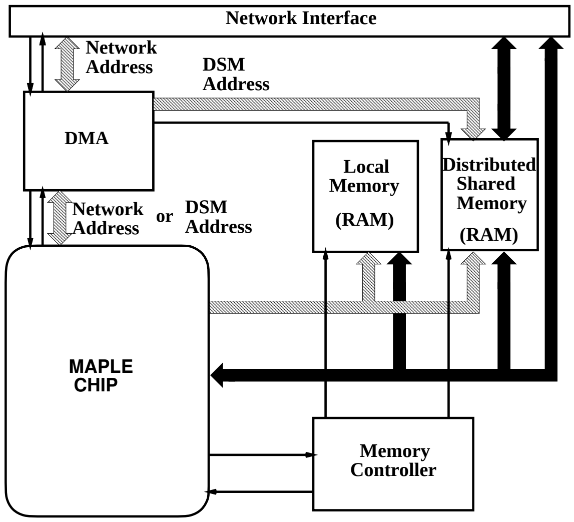


図 5: プロセッシングエレメントの構成

マルチプロセッサとしてプログラムの並列性を利用することも目的としている。従って、スーパスカラや VLIW といったプロセッサ内で命令を並列処理し高速化する技法は、プロセッサ内の処理を複雑にし、本来の目的を妨げる恐れがある。そこで、MAPLE は、教育用の 5 段パイプラインの RISC アーキテクチャDLX [10] を基に以下の必要な機能を追加又は変更して設計を行なう。

- アクセス時間を一定にし、動作予測を可能にしたキャッシュを内蔵
- 全ての演算サイクルを固定
- マルチプロセッサ対応命令 (PE 間データ転送命令、同期命令) を追加

# 4　MAPLE 内蔵キャッシュシステムの目的

## 4.1　商用プロセッサ内蔵キャッシュの問題点

無同期近細粒度並列処理は商用のプロセッサを利用して行なう事ができない。この理由として商用プロセッサに搭載されるキャッシュの動作不確定性があげられる。キャッシュの動作不確定性とは、キャッシュ上にプロセッサコアが必要とするデータが存在する場合とデータが存在しない場合で、データを読み出す速度が異なる為、データ転送速度の予測が不可能であることを意味している。これにより、自動並列化コンパイラは、プロセッサコアのメモリアクセス (ロード命令又はストア命令) 命令について静的にスケジューリングを行なう事ができない。従って、無同期近細粒度並列処理を行なう事ができない。

## 4.2　ソフトウェア制御キャッシュの提案

4.1節では、商用プロセッサ内蔵キャッシュ(従来のキャッシュ) を利用して無同期近細粒度並列処理を行なうことができないことを説明した。しかし、従来のキャッシュでは不可能でも、キャッシュ上にプロセッサが要求するデータが常に存在する新しいキャッシュがあれば無同期近細粒度並列処理をキャッシュを利用して行なうことが可能である。

そこで、近細粒度並列処理では、プロセッサの処理とデータ転送をコンパイラが予測可能なことを利用し、プロセッサコアと独立してキャッシュと PE 内メモリ (LM/DSM) 間のデータ転送を制御することにした。これにより、プロセッサがキャッシュに要求する前にプロセッサの必要とするデータを PE 内メモリからキャッシュへ読み込んでおくことが可能となり、無同期近細粒度並列処理でもキャッシュを使用することが可能になる。

ここで説明したキャッシュは、コンパイラが作成するプログラムによりデータ転送を制御することから、以後、ソフトウェア制御キャッシュと呼ぶことにする。又、これに対し、従来からある、プロセッサからの要求による PE 内メモリへのアクセスをハードウェアで解決するキャッシュをハードウェア制御キャッシュと呼ぶことにする。

# 5　MAPLE 内蔵キャッシュシステム

MAPLE チップ (図 5) に内蔵するキャッシュユニットは、命令キャッシュ、データキャッシュ、ライトバッファ、アウトプットコントローラで構成される (図 6)。

命令キャッシュ：　命令キャッシュは、プロセッサコアが実行する命令を格納する為のキャッシュである。このキャッシュは、ハードウェア制御キャッシュの機能を持ち、その制御を常にハードウェアにより行なう。

データキャッシュ：　データキャッシュは、プロセッサコアが命令の実行に必要とするデータを格納する為のキャッシュである。このキャッシュは、ソフトウェア制御キャッシュとハードウェア制御キャッシュの機能を持ち、処理を行なう時は、一方の機能を利用する。このキャッシュの動作状態をキャッシュ制御モード (ソフトウェア制御モード　又は、ハードウェア制御モード) と呼ぶことにする。キャッシュ制御モードは、マクロタスクの処理内容によりコンパイラが静的に決定する。

ここで、データキャッシュにハードウェア制御キャッシュの機能を設けたのは、以下の理由による。

- マクロタスク中に演算命令よりメモリアクセス命令 (ロード命令又は、ストア命令) が多い場合などに、ソフトウェア制御キャッシュを使用して処理を行なうよ

りハードウェア制御キャッシュで処理を行なった方が速い場合もある

- マクロタスクの途中で、ソフトウェア制御モードでの処理実行が不可能となる場合がある

ライトバッファ：　ライトバッファとは、データキャッシュから PE 内メモリへの書き込みを高速に行なう為の機能である。この機能は、データキャッシュがハードウェア制御モードで動作している場合のみに使われ、ソフトウェア制御モードで動作している時は機能しない。

アウトプットコントローラ：　アウトプットコントローラは、命令キャッシュとデータキャッシュが同時に外部バスへの入出力を行なった場合の優先度を決める。この機能は、データキャッシュがハードウェア制御モードで動作している場合のみに使われ、ソフトウェア制御モードで動作している時は、外部バスはデータキャッシュのみと接続するので機能しない。

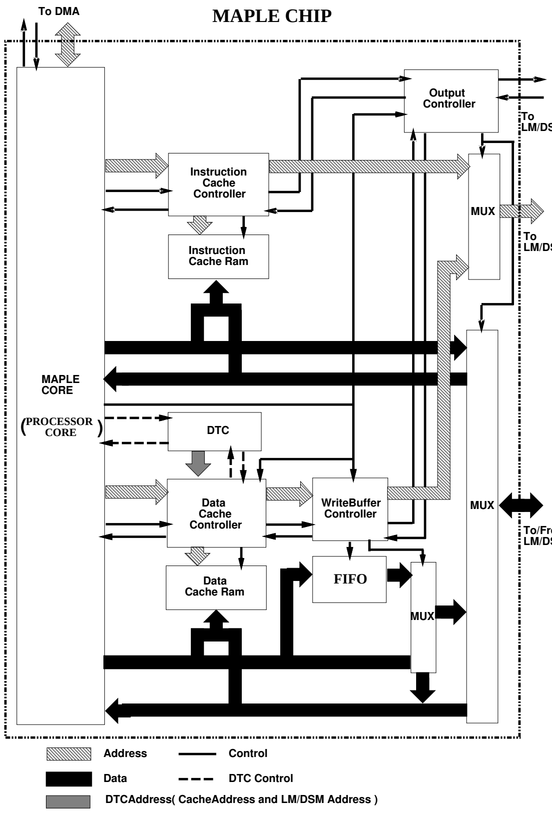


図 6: キャッシュモジュール全体構成

# 6　ソフトウェア制御キャッシュの設計

ソフトウェア制御キャッシュは、データ転送専用のプロセッサであるデータ転送コントローラ (DTC) と、DTC と MAPLE コアの命令に基づきメモリからの転送を実際に行なうソフトウェアキャッシュコントローラで構成する (図 7)。このコントローラは、図 6ではデータキャッシュコントローラ内部に実装されている。

ハードウェア制御キャッシュと比較し、ソフトウェア制御キャッシュを用いる利点は、これまで説明してきたように、キャッシュミスによる動作時不確定性を取り除くことができ、近細粒度並列処理を無同期で実行することが可能になることであるが、さらに、以下の利点がある。

- データ転送をプロセッサの処理と独立して行なうことによりプロセッサが高速に動作可能
- マルチプロセッサ構成において、1 次キャッシュのキャッシュラインの無効化が不要

以下に、データ転送コントローラ (DTC) とソフトウェアキャッシュコントローラについて説明を行なう。

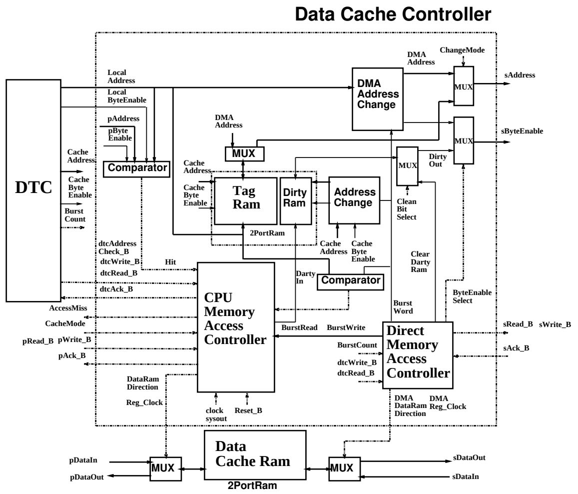


図 7: ソフトウェア制御キャッシュの構成

## 6.1　データ転送コントローラ (DTC)

データ転送コントローラ (DTC) は、図 8に示す 3 段のパイプラインになっており、メモリから命令を取り出す IF ステージ、命令のデコード、アドレス計算を行なう ID/EXE ステージ、そして、PE 内メモリやキャッシュに対してアクセスを行なう ACC ステージで構成する。

IF ステージは、DTC 命令を蓄える為のメモリ、プログラムカウンタとその加算器、MAPLE コアから送られてくるアドレス計算用の値を蓄えるレジスタ、コントローラで構成する。DTC 命令を蓄える為のメモリは命令キャッシュの領域を使用する。

ID/EXE ステージは、動的にキャッシュと PE 内メモリのアドレス計算を行なう為の加算器、命令をデコードする

為のデコーダ、DTC 命令の発行を遅らせる為のカウンタ、コントローラで構成する。

ACC ステージは、メモリアクセスの制御を行なうメモリアクセスコントローラで構成する。

又、パイプラインのストールを管理をストール回路により行なう。

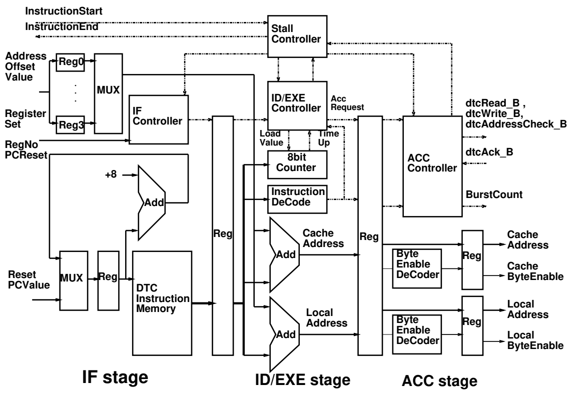


図 8: DTC の構成

次に DTC が実行する命令について説明する。

**キャッシュ/メモリ間データ転送命令：** この命令は、キャッシュと PE 内メモリ (LM/DSM) 間のデータ転送をソフトウェアキャッシュコントローラに指示する命令である。命令には、PL(PreLoad)、PS(PostStore) がある。PL 命令は、PE 内メモリからキャッシュにデータをプレロードする。PS 命令は、キャッシュから PE 内メモリへデータをポストストアする。ソフトウェアキャッシュコントローラは、この指令により指定されたキャッシュ(PE 内メモリ) のアドレス上のデータを指定された PE 内メモリ (キャッシュ) のアドレスへ転送する。転送の最小単位はワードである。ソフトウェアキャッシュコントローラは、DTC からの指令に含まれる転送サイズ分だけバースト転送可能である。このバースト転送は、DTC と独立に行なわれる。

**アドレス変換命令：** この命令は、MAPLE コアがキャッシュに発行したメモリアクセス命令 (ロード命令又は、ストア命令) のアドレスをそれに対応するキャッシュメモリ内のアドレスに変換する命令である。この命令には、LAC(LoadAddressConvert)、SAC(StoreAddressConvert) がある。LAC 命令は、MAPLE が実行したロード命令のアドレスをキャッシュ内のアドレスに変換する。SAC 命令は、MAPLE コアが実行したストア命令のアドレスをキャッシュ内のアドレスに変換する。このアドレス変換命令は、MAPLE コアがメモリアクセス命令を発行する前に必ず発行されている必要があり、MAPLE コアがこのアドレス変換命令に対応するメモリアクセス命令を実行するまで DTC は待機状態となる。

この命令で使用するアドレス変換の情報は、コンパイル時にできる限り生成する。しかし、ループ内の配列参照のようにコンパイル時にアドレス変換情報を作成できないものに関しては、MAPLE コアがプログラム実行中に提供する情報をもとに DTC が簡単な計算を行ない動的にアドレス変換情報を作成する。

この命令により、ハードウェア制御キャッシュとソフトウェア制御キャッシュの間で MAPLE コアが実行する命令の互換性を保つことができるようになる。又、コンパイラが行なう動作予測が失敗した場合を検出する為に利用する。

**処理停止命令：** PE 内メモリ階層へのアクセスのタイミングを遅らせる命令である。この命令は、コンパイル時に何サイクル処理を停止するかを指定することが可能である。1 つの処理停止命令で指定できる最大処理停止サイクル数は、128 サイクルである。これ以上処理を停止したい場合は、この命令を複数実行する必要がある。

次に DTC の処理時間について述べる。DTC は、各ステージ (IF ID/EXE ACC) を、基本的に 1 クロックで処理する。但し、以下の場合は 1 ステージが複数サイクルに渡る。

- アドレス変換命令は、ソフトウェアキャッシュからの応答信号があるまで停止する
- 処理停止命令は、停止クロック数だけ DTC の処理を停止する
- キャッシュ/メモリ間データ転送命令が続けて発行される場合、先に発行されたキャッシュ/メモリ間データ転送命令が終了するまで DTC は処理を停止する

以上説明を行なった命令は、DTC で実行されるが、実際のデータ転送の処理は次に説明するソフトウェアキャッシュコントローラが DTC の指示に従い行なう。

## 6.2 ソフトウェアキャッシュコントローラ

ソフトウェアキャッシュコントローラは、アドレス比較回路、ダーティビットメモリ、タグメモリ、MAPLE とキャッシュ間の状態管理を行なうコントローラ (CPU Memory Access Controller:CMAC)、キャッシュと PE 内メモリの間の状態を管理するコントローラ (Direct Memory Access Controller:DMAC)、データを格納する為のメモリで構成する (図 7)。

ソフトウェアキャッシュコントローラの基本的な機能は、DTC と MAPLE コアからの制御に従いデータ転送を行なうことである。これにより、ハードウェアによるキャッシュライン (ブロック) の管理は不要になると考えられる。しかしながら、コンパイル時の動作予測に失敗した時や、MAPLE

コアとキャッシュ間の書き込みを高速化する為に、ハードウェアによる管理機構も必要になる。

そこで、ソフトウェアキャッシュコントローラでは、キャッシュラインの管理をフルアソシアティブ方式 (タグメモリによりメモリブロックをキャッシュの任意のブロックに配置できるように管理) とし、さらにダーティビットメモリにより、キャッシュラインとメモリブロックのコンシステンシをバイト単位で管理することにする。

ライン管理によりキャッシュと PE 内メモリの配置の対応がとれるため動作予測に失敗した場合、データをキャッシュから PE 内メモリへ書き戻すことが可能である。又、ダーティビットメモリを使用すると、MAPLE コアからバイト又はハーフワードのストア命令が発生してもプレロードによるデータの読み出しを行なわなくてもよくなり、MAPLE からキャッシュへの書き込みが高速に行なえるようになる。

次に DTC からの指令によるソフトウェアキャッシュコントローラの動作について説明する。

**プレロードに対する処理：** DTC からプレロードの要求が発行されると、DMAC と CMAC は同時に処理を開始する。CMAC は、プレロードの要求が発行されると DTC に対して応答信号を返す。但し、DMAC が処理中にキャッシュ/メモリ間データ転送命令が DTC 命令より発行された場合、DMAC が処理を終了するまで応答信号を返すことはない。DMAC は、PE 内メモリ、キャッシュメモリのアドレス、転送サイズを保持し、DMAC が独自に PE 内メモリからの連続的な読み出しを開始する。この時、DMAC はダーティビットメモリをクリーンに初期化し、タグメモリにその読み込んだデータのアドレスを記憶する。

**ポストストアに対する処理：** DTC からポストストアの要求が発行されると、DMAC と CMAC は同時に処理を開始する。CMAC は、ポストストアの要求が発行されると DTC に対して応答信号を返す。但し、DMAC が処理中にキャッシュ/メモリ間データ転送命令が DTC 命令より発行された場合、DMAC が処理を終了するまで応答信号を返すことはない。DMAC は、PE 内メモリ、キャッシュメモリのアドレス、転送サイズを保持し、DMAC が独自に PE 内メモリへの連続的な書き込みを開始する。この時、DMAC は、ダーティビットを調べ、書き戻すワード内のバイトが有効であるかという情報を取得する。この情報を利用してデータを PE 内メモリへ書き込む。その後、ダーティビットメモリはクリーンに初期化される。

**ロードのアドレス変換に対する処理：** DTC からロード命令に対するアドレス変換命令が発行されると、CMAC は、MAPLE コアからのロード命令に対する待ち状態となる。この状態のときに MAPLE コアよりロード命令が発行されると、対応するキャッシュメモリよりデータの読み出しを行なう。この命令ではタグメモリ、ダーティビットメモリは変化しない。

**ストアのアドレス変換に対する処理：** DTC からストア命令に対するアドレス変換命令が発行されると、CMAC は、MAPLE コアからのストア命令に対する待ち状態となる。この状態のときに MAPLE コアよりストア命令が発行されると、対応するキャッシュメモリにデータの書き込みを行なう。この時、CMAC は、タグメモリにその PE 内メモリのアドレスを、ダーティビットメモリに有効なバイトの部分の情報を記憶する。

## 7 評価

ソフトウェア制御キャッシュとハードウェア制御キャッシュの性能について検討した。ここでは例として図 9のループ処理の 1 イタレーションの実行時間を以下の条件で評価した。

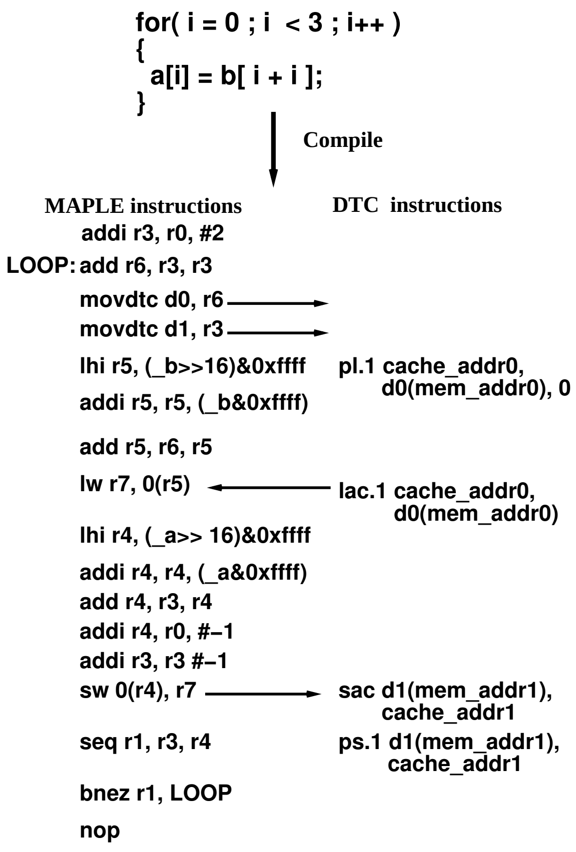


図 9: ソフトウェア制御キャッシュの評価例

- 全ての命令は、命令キャッシュの中に存在する
- データはデータキャッシュの中に存在しない
- 1 ラインは、1 ワードで管理する

- ハードウェア制御キャッシュでは、ミスの原因となったデータを読み込む為に 6 クロックサイクル、ミスの原因となったデータを書き込む為に 2 クロックサイクル必要とする (書き込みはライトバッファを使用)
- ソフトウェア制御キャッシュでは、データの読み込み、書き込み共に 1 クロックサイクル必要とする

この結果を表1に示す。この例では、ソフトウェア制御キャッシュの方がハードウェア制御キャッシュに比べ約 21%速い。

表 1: キャッシュに対する性能評価

| 制御モード | 終了クロック数 |
|---|---|
| ソフトウェア制御キャッシュ | 19 |
| ハードウェア制御キャッシュ | 23 |

次に、同じ例で命令を格納する為に必要なメモリ量について評価した (MAPLE 命令は 1 命令が 4 バイト、DTC 命令は 1 命令が 8 バイト)。この結果を表 2に示す。この例では、ソフトウェア制御キャッシュの方がハードウェア制御キャッシュに比べ約 67%多くメモリを使用する。

表 2: 命令を格納する為に必要なメモリ量の評価

| 制御モード | **MAPLE** 命令 | | **DTC** 命令 | |
|---|---|---|---|---|
| | 命令数 | バイト数 | 命令数 | バイト数 |
| ソフトウェア制御キャッシュ | 17 | 68 | 4 | 32 |
| ハードウェア制御キャッシュ | 15 | 60 | 0 | 0 |

従って、ソフトウェア制御キャッシュでは、性能が向上する利点の一方で命令を格納する為にメモリを大量に消費するという問題も発生する。

これを解決する為には、DTC 命令の命令数を減らすことが必要である。このためには、キャッシュと PE 内メモリ間のデータ転送を連続的に行なう事によりデータ転送の命令数を減らすことが考えられる。

# 8 まとめ

MAPLE 内蔵キャッシュの設計、並びにソフトウェア制御キャッシュの性能解析を行なった。

その結果、現在検討を行なっているソフトウェア制御キャッシュは、ハードウェア制御キャッシュに比べ、性能が向上することを確認できた。又、この評価は、データ転送をプロセッサの処理と独立して行なった場合だけの評価であるため、マルチプロセッサ構成での 1 次キャッシュの無効化を行なわなくてよい場合と、無同期で近細粒度並列処理を行なった場合について考えるとさらに性能が向上すると予想できる。

しかし、このソフトウェア制御キャッシュには、キャッシュ制御用メモリと DTC 命令を格納する為のメモリを大量に必要とするという問題が残されている。従って、この点に関しては、今後さらに検討する必要があると考える。

# 参考文献


[1] 岡本 雅巳, 合田 憲人, 吉田 明正, 笠原 博徳, 成田 誠之助, “実用レベルのマルチグレイン FORTRAN コンパイラの開発”, 情報処理学会研究報告, 96ARC, Oct. 1996

[2] Kasahara H, Honda H, and Narita S,“A Multi-Grain Parallelizing Compilation Scheme for OSCAR”,Proc. 4th Workshop on Languages and Compilers for Parallel Computing , 1991

[3] 合田 憲人, 岩崎 清, 岡本 雅巳, 笠原 博徳, 成田 誠之助, “共有メモリ型マルチプロセッサシステム上での Fortran 粗粒度タスク並列処理の性能評価”, 情報処理学会論文誌, Vol.37, No.3, pp.418-429, Mar. 1996

[4] 本多 弘樹, 合田 憲人, 岡本 雅巳, 笠原 博徳, “Fortran プログラム粗粒度タスクの OSCAR における並列実行方式”, 電子情報通信学会論文誌, D-1 Vol.75, No.8 pp.526-535, Aug. 1992

[5] 尾形 航, 吉田 明正, 合田 憲人, 岡本 雅巳, 笠原 博徳, “スタティックスケジューリングを用いたマルチプロセッサシステム上の無同期細粒度並列処理”, 並列処理シンポジウム JSPP’93, 1993

[6] 笠原 博徳,“並列処理技術”, コロナ社,1991

[7] 笠原 博徳, 成田 誠之助, 橋本 親,“OSCAR(Optimally SCheduled Advanced multiprocessoR) のアーキテクチャ”, 電子情報通信学会論文誌 D, Vol. J71-D No. 8, Aug. 1988

[8] 森村 知弘,“多重バスをエミュレートする多段結合網 R-Clos 網”, 電子情報通信学会, CPSY 97, Aug. 1997

[9] 藤原 崇,“マルチプロセッサシステム ASCA の構成要素プロセッサ MAPLE の設計と実装”, 慶應義塾大学理工学研究科修士論文, 1998

[10] John L.Hennessy , David A.Patterson,“ヘネシー & パターソン コンピュータアーキテクチャ”, 日経ＢＰ社

[11] John L. Hennessy, and David A. Patterson, “COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH SECOND EDITION”, Morgan Kaufmann Publishers, 1996