C-013

タイルアーキテクチャに対するマルチスレッドの空間分割
Space sharing for multithreaded tiled architecture

矢頭 岳人† 米澤 直記† 内田 啓一郎†
Taketo Yato Naoki Yonezawa Keiichiro Uchida

1. はじめに

LSI 微細化による配線遅延の割合増加に対応するためのアーキテクチャとして、複数のプロセッサコアをタイル状に配置したタイルアーキテクチャがある。タイルアーキテクチャでは、1 スレッドに対して複数のコアを割り当てることで命令レベル並列性(ILP : Instruction Level Parallelism)を引き出すことができる。しかし、アプリケーションの処理段階により ILP は変化するため、スレッドに割り当てるコア数を最大需要に合わせてあらかじめ固定しておく方法では、プロセッサの使用が非効率的になる可能性がある。

そこで、本研究では、STA (Space-sharing Tiled Architecture)を提案する。STA は、1 スレッドに割り当てるコア数を可変として、複数スレッドを時分割だけでなく空間分割も行い、同時実行させることでスループットを向上させることを目的とする。

2. 空間分割

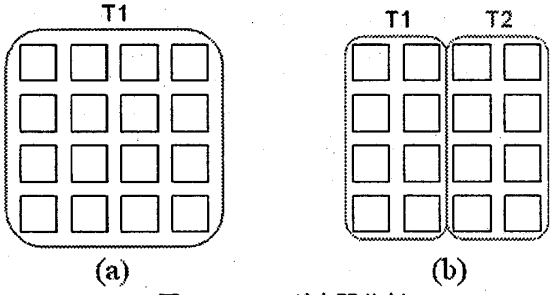

図1 スレッド空間分割

時分割は、スレッドに割り当てるプロセッサ使用時間を分ける方法である。それに対して、空間分割は、スレッドに割り当てるプロセッサ使用領域を分ける方法である。図 1(a)は、1 スレッドを全部に割り当てたものである。また、図 1(b)は、2 スレッドをそれぞれ半分の領域に割り当てたものである。

3. 研究事例

3.1 RAW

RAW[1]は MIT で研究しているアーキテクチャである。各演算タイルが RISC プロセッサであり、個別のプログラムカウンタを持つ。スレッドの空間分割が可能であるが、時分割は困難である。

3.2 TRIPS

TRIPS[2]はテキサス大学で研究しているアーキテクチャである。1 スレッドをすべての演算タイルを使用してデータフロー実行を行う。スレッドの時分割が可能であるが、空間分割は不可能である。

†神奈川大学大学院 理学研究科

4. STA

4.1 アーキテクチャ概要

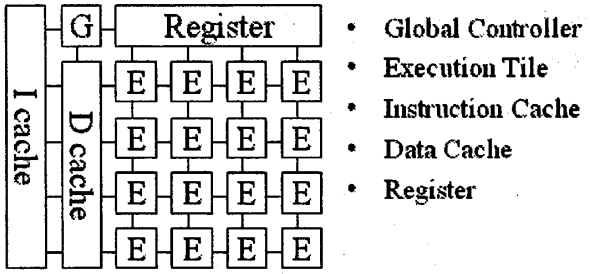

図2 アーキテクチャ

- 16 タイル

4 x 4 の 16 個の演算タイルを持つ。

- ブロック単位実行

ILP を引き出すためにまとめた命令群ごとに実行する。なお、分岐はブロック単位となる。

- ハードウェアマルチスレッディング

4 つのスレッドをハードウェアで管理して、高速にスレッドを切り替える。

- 空間分割マッピング

スレッドの ILP 変化に対応させるため、割り当てるコア数を変える。コンパイラで ILP を引き出す命令ブロックを生成するときに、ブロックのサイズを決める。そして、プロセッサで実行中に、それらブロックの割り当てを行う。

4.2 Global Controller

アーキテクチャ全体の制御を行う。主な機能は、スレッドスケジューリング、フェッチ制御、コミット制御である。

4.3 Execution Tile

4 x 4 の 16 個の演算タイルで計算を行う。1 つの演算タイルは ALU、ルーター、命令バッファを持つ。

4.4 コンパイル方式

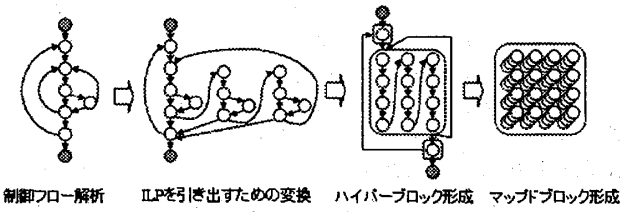

図3 コンパイル（バックエンド）

コンパイラのフロントエンドの部分である構文解析や意味解析は通常の RISC プロセッサ用コンパイラと共通であ

る。バックエンドの部分である依存関係解析やコード生成処理は、本アーキテクチャ特有となる。

バックエンドでは、ベーシックブロックを大きくしたハイパーブロックを形成する。ハイパーブロックとは、ILP を引き出すために、ループ展開、関数のインライン化、条件分岐をプリディケート実行に変換などを行ったものである。

そして、実際にプロセッサに物理的にマッピングできる形式のマップドブロックを形成する。マップドブロックのサイズは、ILP に応じて変える。今回使用するサイズは、Half (4 x 2)、Full (4 x 4)の 2 種類のみとした。

4.5 実行パイプライン

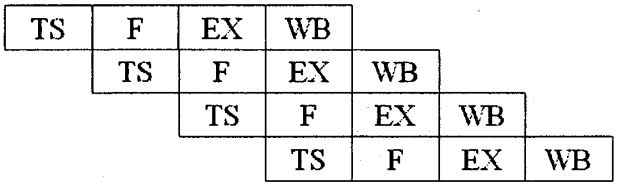

図4　実行パイプライン

TS (Thread Selection)ステージでは、次に実行するブロックの予測、スレッドの選択、割り当てを行う。次の F (Fetch)ステージでは、命令フェッチ、データキャッシュからロードを行う。そして、EX (Execution)ステージで演算処理をブロック単位で行う。最後に、WB (Write Back)ステージで、データキャッシュへのストアとコミット処理を行う。

4.6 スレッドスケジューリング

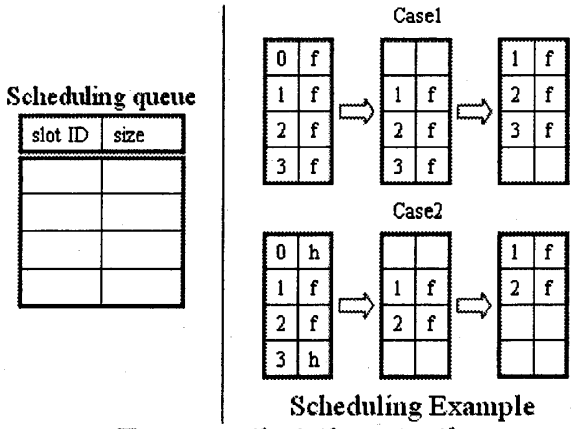

図5　スレッドスケジューリング

スレッドをハードウェアで管理するためのスレッドスロットを 4 つ持たせた。そして、スレッドをプロセッサに割り当てるスケジューリングを行うためにスケジューリングキューを用意した。キューには、スレッドスロット ID と次に実行するブロックのサイズを登録する。サイズは f (full)と h (half)の 2 種類である。図 5 Case1 は、すべてのスレッドが full の場合であり、キューの先頭から、順々に実行していく。Case2 は、full と half のスレッドがそれぞれ 2 つずつある場合であり、実行するスレッドが half の時は、もう 1 つの half のスレッドを同時に実行するため、ID0 と ID3 のスレッドを同時に実行する。

5. 評価

STA のコンパイル方式、スケジューリングアルゴリズムに基づいたテストベクトルを手動で用意した。そして、VHDL で記述した 16 個の演算タイルに、テストベクトルを送りシミュレーションを行った。

ILP が徐々に減少するリダクション計算を行い、空間分割を行わない場合 (Full only)と、行う場合 (Full & Half)の両方についてサイクル数の測定をした。今回はパイプラインの 1 ステージを 4 サイクルと設定してシミュレーションを行った。また、リダクション計算の例として、総和を求めるプログラムと最大値を求めるプログラムを用意した。

表 1　シミュレーション結果

App ＼ Exe Type	Full only	Full & Half
Sum	24 cycle	20 cycle
Max	40 cycle	24 cycle

Sum : 64 個の数値の総和 x 2 スレッド
Max : 64 個の数値の最大値 x 2 スレッド

リダクション計算では計算を進めると並列度が減少する。そのため、処理が進んでいくと待機状態のタイルが現れることになる。その待機状態のタイルに対して、別のスレッドを割り当てることで、スループットを向上させることができる。シミュレーション結果より、空間分割をした方が処理に掛かる時間が短くなっていることが分かる。Max では、空間分割をすることで、サイクル数が 40%減少した。しかし、Sum では、サイクル数の減少量が少なかった。それは、空間分割の粒度が粗いため、急激な ILP の減少に十分対応ができなかったためである。

6. おわりに

空間分割を行うハードウェアマルチスレッディング機能を持つ STA について検討をした。

リダクション計算のシミュレーションを行い、ILP 減少により余った領域に対して別の処理を実行させ、スループットを向上させることが確認できた。

今回は空間分割の粒度を全体の半分とした。今後は、空間分割の粒度を 1/4、1/8 と細かくした場合についてシミュレーションを行い、最適な粒度について検証していきたい。また、シミュレーションをより正確にするため、スレッドスケジューリング、命令フェッチ、コミット制御の回路を作成していくことが課題である。

参考文献

[1]M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe and Anant Agarwal, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE Micro(Mar/Apr 2002).

[2] K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S.W. Keckler, and D.C. Burger, "The Distributed Microarchitecture of the TRIPS Prototype Processor", 39th International Symposium on Microarchitecture (MICRO) (Dec 2006).