計算機アーキテクチャ 135－7
（1999. 11. 27）

命令インタリーブ発行機構を有する マルチスレッド向けプロセッサの提案

小林 真輔　武内 良典　北嶋 暁　今井 正治

大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3
TEL: 06-6850-6626
FAX: 06-6850-6627
E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

あらまし　インストラクション・レベルの並列性のみでなく，スレッドレベルの並列性を利用した「命令インタリーブ機構を有するマルチスレッド向けプロセッサ」を提案する．従来の Superscalar プロセッサや VLIW (Very Long Instruction Word) プロセッサはインストラクション・レベルの並列性を利用することで性能を向上させていたが，提案アーキテクチャではインストラクション・レベルの並列性，スレッドレベルの並列性の 2 種類の並列性を用いることで従来のプロセッサよりも性能が向上すると期待できる．提案アーキテクチャと VLIW プロセッサを比較した結果，JPEG Decode アプリケーションに関して VLIW プロセッサよりも提案アーキテクチャの方が最大 30% 程度実行サイクル数の改善が見られることを示した．

キーワード　マルチスレッド・プロセッサ，VLIW(Very Long Instruction Word)，命令レベルの並列性，スレッド

A proposal of a processor for multi-threading using interleaving threads mechanism

Shinsuke Kobayashi　Yoshinori Takeuchi　Akira Kitajima　Masaharu Imai

Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka
Osaka, 560-8531 Japan
TEL: +81 6 6850 6626
FAX: +81 6 6850 6627
E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

Abstract　The purpose of this study is to propose a processor architecture suitable for multi-threading which achieves better performance than multiple instruction issue architectures, such as Superscalar processor or VLIW processor, not only by using instruction-level parallelism but also by using intra-thread level parallelism. From experimental results, our proposed multi-threaded processor executes JPEG Decoder application 30 % fewer cycles than a VLIW processor which has almost some hardware costs.

Key words　Multi-threading, VLIW(Very Long Instructon Word), Instruction level parallelism, Thread

1 はじめに

Superscalar プロセッサや VLIW プロセッサなどのインストラクション・レベルの並列性を利用したプロセッサが提案され，その結果アプリケーション・プログラムの実行速度は飛躍的に向上した．しかし，これらのプロセッサはインストラクション・レベルの並列性が高いアプリケーションに対しては有効であるが，低いアプリケーションに対しては有効ではない．

本稿では「命令インタリーブ発行機構を有するマルチスレッド向けプロセッサ」を提案する．本アーキテクチャでは，インストラクション・レベルの並列性だけでなく，スレッドレベルの並列性を利用することができる．従来の Superscalar や VLIW は，1 スレッド内の複数の命令の中から依存関係のない命令を発行する方式，すなわちインストラクション・レベルの並列性を利用する方式を採用していた．今回提案するアーキテクチャでは，複数のスレッド内の複数の命令の中から依存関係のない命令を発行する方式，すなわちスレッドレベルの並列性とインストラクション・レベルの並列性の二つを利用する方式を採用する．本方式で，インストラクション・レベルで並列性を抽出する場合と比較して，多くの候補の中から依存関係のない命令を捜し出すことができるために多くの命令を同時に発行できるようになる．Superscalar や VLIW プロセッサにおける命令発行機構 (a) と提案アーキテクチャにおける命令発行機構 (b) との違いを図 1に示す．例としてプログラムの実行が 3 工程 A，B，C の複数回の繰り返しから成る場合を考える．従来アーキテクチャにおける命令発行機構では，工程 A，B，C を逐次的に処理できる．各工程を実行するときに，インストラクション・レベルで命令の並列性を抽出し発行するが，命令間にデータ依存関係等があるために同時に発行できる命令数が制限される．提案アーキテクチャにおける命令発行機構では，工程 A，B，C を並列に処理できる．各工程を実行するときに，インストラクション・レベルで命令の並列性を抽出し発行するのは従来のものと同様であるが，各工程を並列に処理しているために 1 マシン・サイクルで同時に発行できる命令数は従来よりも増加する．また POSIX スレッド [9] ライブラリで用意されている関数の一部を命令とし実現し，専用ユニットによる実現で更なる実行速度向上が期待できる．

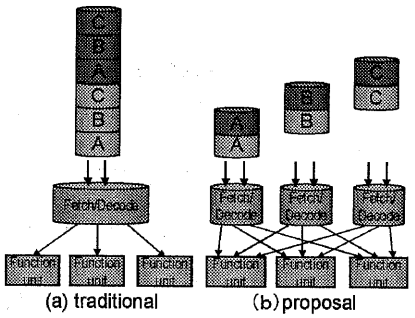

図 1: 従来アーキテクチャにおける命令発行機構と提案アーキテクチャにおける命令発行機構との違い

これまでに提案されているマルチスレッド向けのプロセッサは，インタリーブの粒度から，命令毎にインタリーブする方式と命令ブロック毎にインタリーブする方式の 2 種類に分類できる．命令毎にインタリーブする方式は，平田らによる要素プロセッサアーキテクチャ [1, 2] や，Superscalar をベースにしたアーキテクチャ [3]，VLIW をベースにしたアーキテクチャ [4]，スレッド間通信をサポートしたアーキテクチャ [5] などが提案されている．命令ブロック毎にインタリーブする方式は文献 [6] で提案されている．本稿で提案しているアーキテクチャは命令インタリーブ方式の範疇に入るが，スレッドの優先度を動的に変更できる点やスレッドをスケジューリングするユニットを備えている点で，今までに提案されたアーキテクチャと異なる．

本稿では，第 2 節にスレッドについて述べ，第 3 節に提案アーキテクチャの詳細について述べる．第 4 節では性能評価を行い，第 5 節にまとめと今後の課題を述べる．

2 スレッド

提案アーキテクチャは，スレッド用スケジューラの実装に POSIX スレッド・ライブラリ [9] の関数群を用いる．特に効率を優先させるために関数の中で次の 3 種類の関数を命令として実装した．

pthread_create スレッドを生成する関数

pthread_exit スレッドを終了する関数

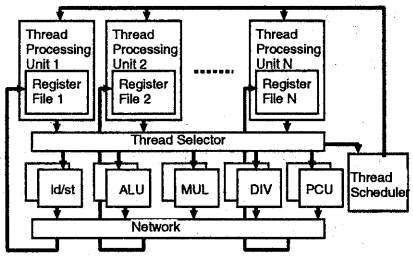

図 2: Structure of CPU core

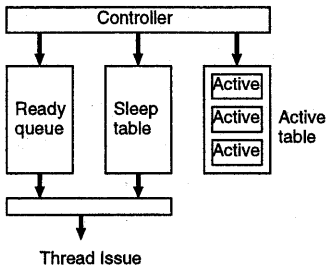

図 3: Structure of Thread Scheduler

pthread_join　スレッドを合流させる関数

POSIX スレッドの仕様にしたがい，起動時に動作するスレッド以外は，スレッドは必ず pthread_create によって生成され，pthread_exit によって終了する．また，スレッドはスレッド ID，優先度，プログラム・カウンタ，スタック，レジスタセットのコンテキストを持つ．

3　提案アーキテクチャ

提案するアーキテクチャの構成図を図 2に示す．提案アーキテクチャでは，独立して命令フェッチ，デコードを行うスレッド処理ユニット，オペレーションの競合を回避するためのスレッド・セレクタ，スレッドのスケジューリングを行うスレッド・スケジューラ，そして機能ユニットから構成されている．以下，スレッド処理ユニット，スレッド・セレクタ，スレッド・スケジューラについて説明する．

3.1　スレッド処理ユニット

スレッド処理ユニットはデコーダ，レジスタファイル と制御用ユニットの組である．複数のスレッド処理ユニットが独立して命令フェッチ，デコードを行うことでスレッドの並列処理を実現している．各スレッド処理ユニットでデコードされる命令には，インストラクション・レベルの並列性を利用するために VLIW 命令を用いる．ここで VLIW 命令とは，1 個の命令内に複数のオペレーションを指定する命令形式のことである．VLIW 命令を用いることで 1 命令に複数のオペレーションを指定することができる．コンパイラによって静的に VLIW 命令用にスケジューリングしたコードを使用することで，インストラクション・レベルの並列性を利用する．また，コンパイラでデータ依存関係を解析し命令を並べ替えるとし，ハードウェアによる負担を軽減する．

3.2　スレッド・セレクタ

スレッド・セレクタは，各スレッド処理ユニットにおいて発行可能な複数のオペレーションが機能ユニットに対して発行されるときに，スレッド処理ユニット間のオペレーションが競合しないように調整を行うユニットである．提案アーキテクチャでは各スレッドに付加された優先度を用いる方法でオペレーションの競合を回避する．優先度は各スレッド処理ユニット中の優先度レジスタに格納されている値を参照することで得られる．各機能ユニットにおいてオペレーションが競合した場合は，スレッドの優先度を比較し，優先度が最も大きいオペレーションを発行する．スレッドの優先度が等しい場合は，スレッド処理ユニットの優先順にしたがい決定する．スレッド・セレクタによって選択されなかったオペレーションは次回発行時に再度調整が行われる．再発行時にはスレッドの優先度をあげる．

3.3　スレッド・スケジューラ

スレッド・スケジューラの構成図を図 3に示す．スレッドは active, ready, sleep のいづれかの状態となっているが，各スレッドの状態を管理するためにスレッド・スケジューラは次の機構を備えている．

Active table　スレッドの状態が Active の時，すなわち CPU 上で実際に動作している状態のスレッドを管理する．Active table に登録されているスレッドの状態は Active である．

Sleep table　スレッドの状態が Sleep の時，すなわちあるスレッドが終了するまで実行不可能である状態にあるスレッドのコンテキストを保存す

る．Sleep table に登録されているスレッドの状態は Sleep である．

Ready queue スレッドの状態が Ready の時，すなわち実行する準備が整っている状態にあるスレッドを保存する．Ready queue に登録されているスレッドの状態は Ready である．

スレッド・スケジューラは POSIX スレッド・ライブラリの pthread_create, pthread_exit, pthread_join の 3 種類の関数をハードウェアで実装したものである．スレッド・スケジューラでは，スレッド生成，スレッド合流，スレッド終了を行うときのスレッドの状態遷移を管理している．スレッド・スケジューラの動作を以下に示す．

- スレッド生成 (thread_create)
  - 空きスレッド処理ユニットが存在するならば，そのスレッド処理ユニットに対してスレッドを発行し，Active table に登録する．このとき発行されるスレッドは Active 状態となる．
  - 空きスレッド・スロットが存在しないならば Ready queue に登録する．このとき Ready queue に登録したスレッドは，Ready 状態となる．
- スレッド合流 (thread_join)
  - 合流するスレッドが終了していない場合，Sleep table に登録する．このとき Sleep table に登録したスレッドは，Sleep 状態となる．
  - 合流するスレッドが既に終了している場合は何もしない．
- スレッド終了 (thread_exit)
  - 合流できるスレッドが Sleep table に存在する場合は Sleep table の登録を抹消し，Active table に登録する．このとき合流したスレッドは，Active 状態となる．
  - 発行可能スレッドが Ready queue に存在する場合は，そのスレッドを発行し，Active table に登録する．このとき発行されたスレッドは，Active 状態となる．

4 性能評価

4.1 目的

提案アーキテクチャの性能を評価するため，実行サイクル数と面積について提案アーキテクチャと従来アーキテクチャとで比較し，提案アーキテクチャの有効性を確認する．

4.2 ターゲット・アーキテクチャ

今回実験に使用した提案プロセッサの構成を示す．

基本アーキテクチャ・タイプ ロード/ストア・アーキテクチャ，ハーバード・アーキテクチャ，3 段パイプライン．

命令セット DLX (32 bit RISC プロセッサ) [7] 命令セットのサブセットにスレッド制御用命令 (thread_create, thread_join, thread_exit) を付加した命令セット

レジスタファイルの構成 汎用レジスタ方式で各スレッドスロット毎に 32 個の 32 ビットレジスタを有する構成．

機能ユニット 機能ユニットの種類は，ロード・ストアユニット，ALU，乗算器，除算器，PCU (Program counter Control Unit)，シフタ，比較器．各ユニットとも結果遅延 1 サイクル，発行遅延 1 サイクルの機能ユニットを採用．機能ユニットの個数は PCU 以外は全て各種類 1 ユニットずつとし，PCU のみスレッド・スロット毎に 1 ユニット有．

メモリモデル オンチップ高速 SRAM．

比較に用いる VLIW プロセッサの構成を次に示す．基本アーキテクチャ・タイプとメモリモデルは提案アーキテクチャと同様である．

命令セット 提案プロセッサの命令セットからスレッド制御用命令を除いた命令セット．

レジスタファイルの構成 汎用レジスタ方式で 32 個の 32 ビットレジスタを有する構成．

機能ユニット 提案プロセッサと同様の機能ユニットを使用するが，PCU は 1 ユニットとした構成．

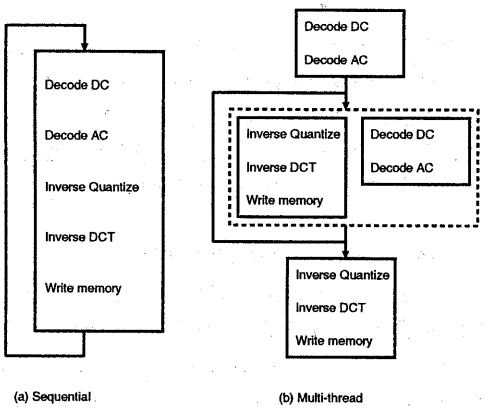

図 4: JPEG Decoder のスレッド分割 (同時実行スレッド数 2)

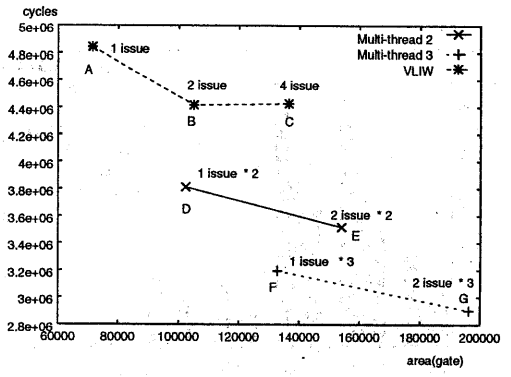

図 5: JPEG Decoder を実行した際の面積と実行サイクル数の関係

4.3 サンプル・プログラム

実行サイクル数を測定する実験では C 言語で記述された JPEG Decoder をサンプル・プログラムとして用いた．JPEG Decoder は主に，DC 成分デコード，AC 成分デコード，逆量子化，逆 DCT，メモリ書き出しの 5 工程をマクロブロック単位で処理している．マルチスレッド・プロセッサで JPEG プログラムを動作させるために，同時実行スレッド数に着目してプログラムのマルチスレッド化を行う．ここで同時実行スレッド数とは，並列に実行可能な最大スレッド数のことである．同時実行スレッド数 2 の場合は，DC 成分デコード，AC 成分デコードをスレッド処理ユニット 1 に，逆量子化，逆 DCT，メモリ書き出しをスレッド処理ユニット 2 に割り当てる．同時実行スレッド数 3 の場合は，DC 成分デコード，AC 成分デコードをスレッド処理ユニット 1 に割り当て，逆量子化，逆 DCT をスレッド処理ユニット 2 に，メモリ書き出しにスレッド処理ユニット 3 を割り当てる．同時実行スレッド数 2 の場合のスレッド分割を図 4に示す．全てのフェーズを逐次的に実行していた動作 (a) がスレッドを分割する事で並列に動作する (b)．同時実行スレッド数 2 の場合はスレッド処理ユニットを 2 個有するプロセッサで実行し，同時実行スレッド数 3 の場合はスレッド処理ユニットを 3 個有するプロセッサで実行した．VLIW プロセッサで実行する場合はスレッド分割は行わず，全て逐次的に処理した．

4.4 実験環境

C プログラム用のコンパイラとして gcc-2.7.2.3 (GNU C Compiler)[8] を基に独自のスケジューリング・パスを追加した VLIW 用コンパイラを使用する．提案プロセッサの面積は VHDL 記述より，論理合成ツール Synopsys 社 Design Compiler を用いて求めた．合成用ライブラリは VLSI テクノロジー社の VSC753d (0.5μm CMOS) ライブラリを用いた．

4.5 実験結果と考察

JPEG Decoder を実行するための実行サイクル数とプロセッサの面積の関係を図 5に示す．横軸は面積を示し，縦軸は実行サイクル数を示す．各点は同時発行可能な命令数 1 である VLIW プロセッサ (A)，同時発行可能な命令数が 2 である VLIW プロセッサ (B)，同時発行可能な命令数が 4 である VLIW プロセッサ (C)，同時発行可能な命令数が 1 であるスレッド処理ユニットを 2 個有する提案プロセッサ (D)，同時発行可能な命令数が 2 であるスレッド処理ユニットを 2 個有する提案プロセッサ (E)，同時発行可能な命令数が 1 であるスレッド処理ユニットを 3 個有する提案プロセッサ (F)，同時発行可能な命令数が 2 であるスレッド処理ユニットを 3 個有する提案プロセッサ (G) である．

2 命令同時発行可能な VLIW プロセッサ (B) とスレッド処理ユニットを 3 個有する提案プロセッサ (F) とを比較すると，実行サイクル数が約 30 % 改善されている．さらに，2 命令同時発行可能な VLIW プ

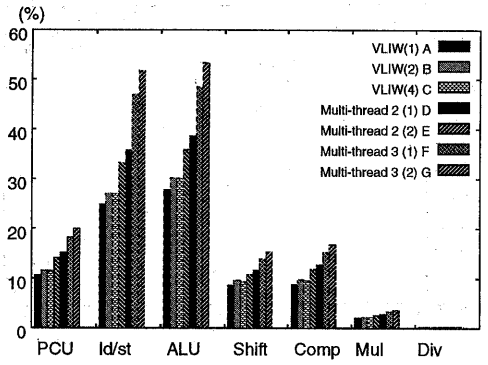

図 6: JPEG Decoder を実行するための機能ユニットの使用率

ロセッサを用いる場合 (B) とスレッド処理ユニットを 2 個有する提案プロセッサ (D) を比較した場合，面積，実行サイクル数ともに提案プロセッサの場合が優れている．スレッド処理ユニットを 3 個有する提案プロセッサ (F, G) はスレッド処理ユニットを 2 個有する提案プロセッサ (D, E) よりも実行サイクル数が 15 % 減少している．しかし，面積を比較すると 3 万ゲート以上面積が大きくなっているために設計制約に合わせた構成を選択が重要となる．

各プロセッサ構成で JPEG Decoder を実行するときの機能ユニットの使用率を表したものを図 6に示す．プロセッサ上でアプリケーション・プログラムを実行した時の機能ユニットの使用率を式

$$\text{機能ユニットの使用率} = \frac{\text{機能ユニットで実行した命令数}}{\text{総実行クロック・サイクル数}}$$

で求め，図6に示した．グラフ中の VLIW は，VLIW プロセッサでの使用率を表し，Multi-thread 2, Multi-thread 3 はそれぞれスレッド処理ユニット数が 2，3 の構成での使用率を表している．括弧内の数字は同時命令発行数を示している．図6より ロード・ストアユニット，ALU に関しては提案プロセッサでは，VLIW プロセッサの最大約 2 倍の使用率になっており，一般に機能ユニットの使用率が向上している．また機能ユニットの使用率を考慮すると，ロード・ストアユニット，ALU の個数を増やすことで JPEG Decoder をより速く実行することができると考えられる．

5 おわりに

命令インタリーブ発行機構を有するマルチスレッド向けプロセッサの提案を行い，試行実験により通常の VLIW プロセッサと比較して提案プロセッサの方が面積，実行サイクル数ともに優れた構成になる場合が存在し，設計制約次第で選択され得ることを示した．

今後の課題は，演算器の数や種類，レジスタファイルに含まれるレジスタの個数を変化させることで最適な構成を検討することである．また今回評価用アプリケーションとして JPEG Decoder を用いたが，他のアプリケーションについても評価することが必要である．

謝辞

本研究を進めるにあたり貴重なコメントを頂いた，大阪大学 VLSI システム設計研究室の諸兄に深謝する．なお，本研究の一部は (株) 半導体理工学研究センターとの共同研究による．

参考文献

[1] 平田博章，奥村晃生，柴田幸茂，新實治男，柴山潔，“マルチスレッドプロセッサおよび 1 チップマルチプロセッサのための命令キャッシュ構成・命令フェッチ方式の性能比較”，電子情報通信学会論文誌 D-I，Vol.J81-D-I，No.6，pp.718-727，June 1998.

[2] 平田博章，木村浩三，永峰聡，西沢貞次，鷲島敬之，“多重スレッド・多重命令発行を用いる要素プロセッサ・アーキテクチャ”，情報処理学会論文誌，Vol.34，No.4，pp.595-695，April 1993.

[3] George E. DADDIS Jr and H.C. Torng, “The Concurrent Execution of Multipule Instruction Streams On Superscalar Processors”, International Conference on Parallel Processing Vol.1 Architecture, pp.76-83, August 1991.

[4] R.Guru Prasadh and Chuan-lin Wu, “A Benchmark Evaluation of a Multi-Threaded RISC Processor Architecture”, International Conference on Parallel Processing Vol.1 Architecture, pp.84-91, August 1991.

[5] Stephen W. Keckler and William J. Dally, “Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism”, In Proceedings of the 19th Annual Synposium on Computer Architecture, IEEE, pp.202-213, May 1992.

[6] Anant Agarwal, Beng-Hong Lim, David Kranz, and John Kubiatowicz, “APRIL: A Processor Architecture for Multiprocessing”, In Proceedings of the 17th Annual Synposium on Computer Architecture, IEEE, pp.104-114, May 1990.

[7] David A. Patterson, John L. Hennessy, *Computer Architecture A Quantitative Approach Second Edition*, MORGAN KAUFMANN PUBLISHERS, INC., 1996.

[8] R. M. Stallman, *Using Porting GNU CC*, Free Software Foundation, Inc., 1995.

[9] Bil Lewis, and Daniel J. Berg, 岩本信一 訳，“P スレッドプログラミング”，ピアソン・エデゥケーション，1999.