レジスタウィンドウを用いた マルチスレッドプロセッサの設計

高瀬 亮, 日高 康雄, 小池 汎平, 田中 英彦
{takase, hidaka, koike, tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部*

1 はじめに

プロセッサの処理速度を向上させる際, クロックサイクルのような物理的な要因とともに, プログラムの並列度を引き出すことも重要である. この手法の一つにマルチスレッドがあるが, レジスタウィンドウを持つ (シーケンシャルなプログラムに対して最適化してある) プロセッサ上でのマルチスレッド処理は, マルチスレッド, レジスタウィンドウという両者の利点を相殺していた.

本稿では, レジスタウィンドウ上に複数のスレッドを実装することにより, 高速なマルチスレッド処理が実現できると考え, このような処理を支援するハードウェア機構について設計した.

2 レジスタウィンドウ上のマルチスレッド

ここでは, UNIX などでいう 「プロセス」の概念からアドレス空間管理などを取り除き, 「命令実行のフロー」のみを管理する単位をスレッドと呼ぶことにする.

マルチスレッドを効率良く実行するためには, スレッドのスイッチにかかるコストを低減することが重要である. レジスタウィンドウを持ったプロセッサ上でのマルチスレッド処理は, スレッドのスイッチの際に使用していた全てのレジスタウィンドウを退避/復元してしまっていた. これがオーバーヘッドとなり, 結果としてレジスタウィンドウを持つプロセッサでマルチスレッド処理を行うのは不利であった.

そこで, 本研究では, 図 1 のように, レジスタウィンドウ上に 「同時に」複数のスレッドを実装する. この結果, レジスタウィンドウ上に存在するスレッド間のスイッチを高速に行うことが可能となる. また, 各スレッドが使用するレジスタウィンドウの数は可変であり, プロシジャコールの深さによって動的に変化する. したがって, プロシジャコールの速度に重点を置いた, レジスタウィンドウの利点も維持することができる.

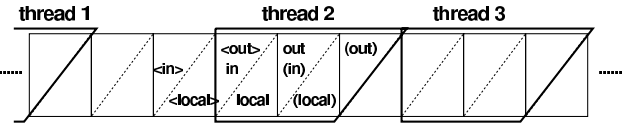

図 1: レジスタウィンドウ上のマルチスレッド

3 拡張するハードウェア機構の概要

レジスタウィンドウを持つプロセッサをベースにして, このプロセッサに一部マルチスレッド向きのハードウェア機構を付加することで, マルチスレッド処理性能を向上させることを考える.

3.1 スレッドのスケジューリング方式

プロセッサ上でのマルチスレッド実現方式として,

1. インターリーブ方式
   一定のクロックサイクル毎にハードウェアで自動的にスレッドのスイッチを行う
2. ブロッキング方式
   スレッドがノンプリエンプティブにスイッチする

の二通りが考えられる. 本研究では, インターリーブ方式を採用する. この結果, レジスタウィンドウ上にあるスレッドへのスイッチにおいて, オーバーヘッドを解消できる.

3.2 WIM の拡張

従来の WIM (Window Invalid Mask) は, 各ウィンドウに 1 ビットが割り当てられたビットマスクであった.

今回は, 各ウィンドウに対応して, そのウィンドウが使用しているスレッドの識別子 (スレッド ID) を持たせるように拡張する. これは, あるスレッドがプロシジャコールを起こし隣のスレッドをメモリ上に退避させる時, そのスレッドの退避場所を判定する必要性が生じることによる.

3.3 スレッドスケジューリングテーブルの導入

また, 図 2 に示すようなスレッドスケジューリングテーブル (以下 TST) を用意する. TST は, 「レジスタウィンドウ上にある」スレッドの管理を行うテーブルであり, TST の 1 エントリが一つのスレッドに対応する.

TST には, 各スレッドが実行可能であるかどうかを表すスレッドステータスフィールド, 各スレッドが使用しているウィンドウトップの位置を指すポインタが存在する. また, TST のエントリ数に対応して, これと同数のプログラムカウンタ (PC), スレッド状態レジスタ (含コンディションコード, CWP), 退避先アドレスなどを計算

---

* *"Design of Multithreading Processor using Register Windows"*
Ryo TAKASE, Yasuo HIDAKA, Hanpei KOIKE and Hidehiko TANAKA
Faculty of Engineering, the University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo, 113, Japan

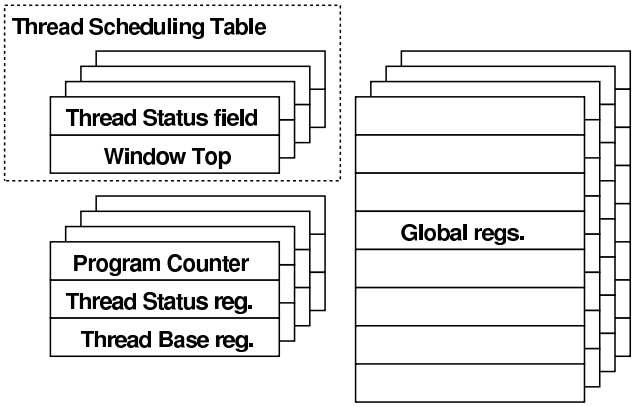

図 2: スレッドスケジューリングテーブル

する際に利用するスレッドベースレジスタ, グローバルレジスタなどが存在する.

3.4 TST によるスケジューリング

TST を用いたスケジューリング動作は図 3 のようになる. プロセッサは, TST のスレッドステータスフィールドを参照し, 実行可能なスレッドを順次スイッチすることにより, マルチスレッド処理を実現する. このスイッチ動作は 3.1 節で述べたように各クロック毎にハードウェアによって行われる.

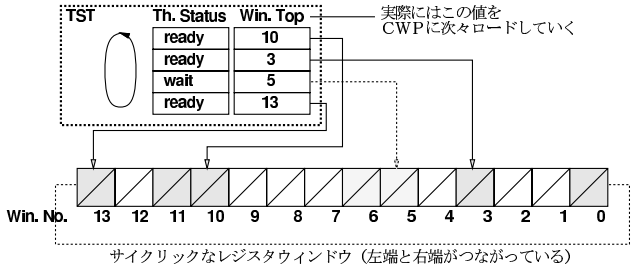

図 3: スケジューリングの様子

4 支援スレッドによるスレッドの動作

4.1 マルチスレッド処理の問題点と支援スレッド

図 3 のようなインターリーブ方式によるスレッドスイッチで, TST に存在するスレッドのみを実行するスケジューリング方針では,

1. レジスタ退避/復元の際のスレッド間の干渉が起こり得る (退避/復元中も実行するスレッドがスイッチしていくので)
2. TST 上にないスレッドの起動が行えない
3. 新たなスレッドの生成が行えない

という問題が残る.

そこで, 支援スレッドというスレッドを導入する. 支援スレッドは, 次に示すような特徴を持つ.

- 自分自身は実体を持たない (レジスタウィンドウを占有しない) スレッドである
- 一般のスレッドがレジスタウィンドウ ↔ メモリ間の転送を行う時, スレッドの起動 (生成) を行う時に, その処理対象となるレジスタウィンドウを一時的に横取りする
- 支援スレッドの優先度[1]は全スレッド中で最も高く, 他のスレッドから干渉を受けることなく動作する

また, 新たなスレッドの起動, 生成については, 空きウィンドウの監視をハードウェアで行い, 空きウィンドウで支援スレッドを実行させることによって行うことを考えている.

4.2 支援スレッドの動作例

支援スレッドを導入して, スレッドがプロシジャコールをした際の退避動作を示したものが図 4 である. この

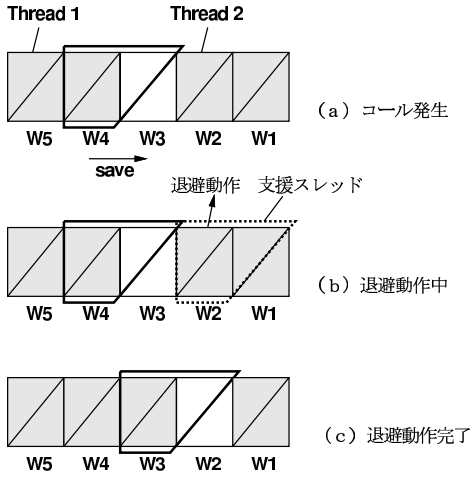

図 4: 退避動作における支援スレッド

図において, (b) の退避動作中はスレッド 1 からの save 命令, スレッド 2 からの restore 命令をブロックすることによって, 他のスレッドから干渉を受けることなく退避を行うことができる.

5 おわりに

マルチスレッド向きのレジスタウィンドウ管理と, それを支援するハードウェア機構について述べた. 現在, SPARC プロセッサをベースにしたシミュレータを実装中である.

参考文献

[1] SPARC International, Inc., "The SPARC Architecture Manual Version 8", Prentice-Hall, Inc., 1992.

[2] 高瀬亮, 日高康雄, 小池汎平, 田中英彦, "レジスタウィンドウを用いた高速マルチスレッドアーキテクチャの検討", 情報処理学会第 51 回全国大会講演論文集 (6), pp. 7–8, Sep. 1995.

[1] スケジューリングの優先度という意味合いではなく, レジスタウィンドウという資源を取り合う時の強さを表す.