1Q−10 PIE64の並列処理管理カーネルの実装 −レジスタウィンドウにおける高速タスクスイッチング−

日高 康雄　山本 敬　小池 汎平　田中 英彦
{hidaka,yamamoto,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部

1 はじめに

我々は，高並列処理においては，通信・同期と並列処理管理が根本的なオーバヘッドであると考え，並列推論エンジン PIE64[5] の要素プロセッサに，推論・通信・管理プロセッサからなる，複合アーキテクチャを採用した [6]. 管理プロセッサが実行するファームウェアは並列処理管理カーネル [7] と呼ばれ，スケジューリングや負荷分散などの並列処理管理機能を担っている. これは，推論プロセッサからの要求，他から到着した負荷の処理，ホスト計算機や周辺装置からの要求などに従った，イベント駆動型のマルチタスク制御を必要とする.

管理プロセッサには，様々な実験ができる汎用性を重視して RISC 型の高速マイクロプロセッサである SPARC を採用したが，SPARC はレジスタウィンドウを持つために，タスクスイッチの際に全てのレジスタを入れ換えると大きなオーバヘッドが生じる. 並列処理管理カーネルの実装では，イベントにできるだけ早く応答する必要があるため，タスクスイッチの高速化が必須である.

本稿では，並列処理管理カーネルの実装上の工夫の中で，レジスタウィンドウにおける高速タスクスイッチングの実現手法について述べる. 本手法の特徴は，手続き呼び出しの高速化というレジスタウィンドウの特長を生かしつつ，タスクスイッチの高速化を達成している点である.

2 SPARC のレジスタウィンドウ

SPARC[3] は，Berkeley RISC I[1], RISC II と同様に，重複のあるレジスタウィンドウによって，手続き呼び出しが高速化されている. すなわち，手続き呼び出しの度に新たな local レジスタが確保され，引数や戻り値は重複部分のレジスタ (呼び出し側の out レジスタ / 手続き側の in レジスタ) で渡される.

現在のウィンドウは，ステータスレジスタ内の Current Window Pointer (CWP) によって示され，ウィンドウの移動は，手続きの入口の save 命令 (CWP をデクリメント) と出口の restore 命令 (CWP をインクリメント) で行なわれる. また，トラップの発生 / トラップハンドラからの復帰の時にもウインドウが移動する.

ウィンドウはサイクリックにつながっており，手続き呼び出しが深くなってウィンドウが一周すると，save 命令実行時にオーバフロートラップが発生する. そのトラップハンドラは，以前のウィンドウ内容をメモリに退避し，空きのウィンドウを確保する. 逆に，restore 命令実行時に有効なウィンドウが残っていない時は，アンダフロートラップが発生し，そのトラップハンドラはメモリに退避しておいたウィンドウ内容を復元する. オーバ / アンダフロー時に退避 / 復帰するウィンドウ数は，[2] によれば固定的に 1 個 (ウィンドウが 8 個以上の場合は，2 個) が良いとされている.

ウィンドウが一周した時にも，現在の out と以前の in が重ならないようにするために，1 個のウィンドウは必ず予約されている. このため通常の処理に使える使えるウィンドウ数は，CPU 内のウィンドウ総数より 1 個少ない.

ウィンドウのオーバフロー，アンダフローは，Window Invalid Mask (WIM) レジスタによって検出される. save 時に，WIM のビット CWP-1 が 1 であるとオーバフローが発生し，restore 時に，WIM のビット CWP+1 が 1 であるとアンダフローが発生する.

高速なタスクスイッチが要求される用途には，各タスクを別のウィンドウに割り付け，レジスタウィンドウのないプロセッサと同様の，save/restore を使わないプログラムを走らせて，CWP の変更でタスクスイッチを行なう手法が考えられている [3]. この手法は，タスクスイッチは非常に高速になるが，本来は 1 個の予約ウィンドウがあれば済んでいたトラップ処理用のウィンドウを，タスクに割り付けるウィンドウの間に一つずつはさまなければならないため，半分程のレジスタが有効に使われなくなる. また，奇数個のウィンドウを持つ SPARC(富士通製 S20 など) は，ウィンドウ一つが完全に無駄になる. さらにこの手法では，手続き呼び出しの高速化という，レジスタウィンドウの本来の特長を完全に失ってしまっている.

3 単純なレジスタウィンドウ管理アルゴリズム

レジスタウィンドウの単純な使い方 [4] では，WIM を常に一つのビットだけが 1 になるように管理する.

オーバフロートラップハンドラのアルゴリズムを以下に示す.

1. save 命令で，退避すべきウィンドウに移動.
2. WIM を右 1 ビット rotate.
3. in レジスタと local レジスタをスタックに退避.
4. restore 命令で，トラップ直後のウィンドウに戻る.
5. rett 命令で，トラップ処理から復帰.

次に，アンダフロートラップハンドラのアルゴリズムを以下に示す.

1. WIM を左 1 ビット rotate.
2. restore 命令 2 回で，復元すべきウィンドウに移動.
3. in レジスタと local レジスタをスタックから復元.
4. save 命令 2 回で，トラップ直後のウィンドウに戻る.
5. rett 命令で，トラップ処理から復帰.

上記両方のステップ 3 のスタックには，out レジスタの一つをスタックポインタ (sp) に使い，in/local レジスタの退避 / 復元は sp の指す領域に対して行なう. このスタックには，レジスタに割り付けられない引数や自動変数も配置される.

sp の値は，そのウィンドウを使用する手続きの最初で，呼び出し側の sp の値から確保すべきバイト数を減じて求める. ここで確保された領域のうち，自動変数などの領域は直ちに使われるが，in/local レジスタの退避領域は，この後手続き呼び出しを繰り返して，ウィンドウが一周した時にはじめて使われる.

sp の値の計算には，save 命令のもう一つの機能である加算機能を用いる. すなわち，手続きの入口の save 命令では，ウィンドウの移動と同時に，移動前の sp の値と負の即値を加算して，移動後の sp の値を計算する.

単純なタスクスイッチのアルゴリズムを以下に示す.

1. stack pointer, return address を退避.
2. ウィンドウ総数 -2 個の save 命令で，アクティブなウィンドウをフラッシュ.
3. 直ちにアンダフロートラップが起こるように，WIM を設定.
4. 新しいタスクの stack pointer, return address を復元.
5. restore 命令 (+ アンダフロートラップ) で一つのウィンドウを復元.
6. return 命令で，タスクの中断箇所から実行を再開.

ステップ 2 の save 命令がウィンドウ総数より 2 だけ少ないのは，予約されているウィンドウと，タスクスイッチ手続きで使っているウィンドウは退避する必要がないためである. またこのステップで実際にウィンドウが退避されるのは，オーバフローが発生した場合だけであり，平均して総数の半分程度のウィンドウが退避されるが，最悪時には，総数 -2 個のウィンドウが退避される.

The implementation of the parallel processing management kernel of PIE64 – fast task switching with register windows
Yasuo HIDAKA, Hanpei KOIKE, Hidehiko TANAKA
University of Tokyo, Faculty of Engineering

タスクスイッチに仮想アドレス空間の切替が伴う場合は，ステップ2の後でその切替を行なう．また，仮想アドレスキャッシュを用いている場合は，アドレス空間の切替とともにキャッシュのフラッシュが必要であり，これがタスクスイッチ時間の大きな部分を占めるため，アクティブウィンドウのフラッシュ時間の割合はそれほど大きくない．

4 タスクスイッチの高速化

タスクスイッチに仮想アドレス空間の切替が伴う場合は，アクティブウィンドウのフラッシュは必然的である．しかし，組み込み用途で物理アドレスを直接用いている場合や，仮想アドレス空間でもスイッチするタスク間でアドレス空間を共有している場合は，各タスクのスタックは必ずアドレスが異なっており，ウィンドウ毎にspを持っているため，アクティブウィンドウのフラッシュは必ずしも必要ではない．また，仮想アドレスキャッシュのフラッシュも必要ないため，タスクスイッチ時間中のアクティブウィンドウのフラッシュ時間の割合が大きくなり，問題となってくる．

4.1 異なるタスクのウィンドウの混在化

そこで，レジスタウィンドウ上に複数のタスクのウィンドウを混在させることを考える．別のタスクのウィンドウに対しても，WIM内のビットを1にすることによって，タスクスイッチの際のアクティブウィンドウのフラッシュは必要なくなり，後で退避の必要が生じた時に，退避を行なうようにすることができる．

しかし，これを単純に実装すると，以下のような問題が発生する．

1. 単純な管理アルゴリズムでは，ウィンドウの退避が必要になるのはオーバフロー時だけであったが，アンダフロー時にも他のタスクのウィンドウが残っていれば，それを退避してから，目的のウィンドウを復元する必要がある．
2. 他のタスクの複数のウィンドウが残っている時，スタックトップに相当するウィンドウを最後まで残しておいて，スタックトップから最も遠いウィンドウから先に退避するのが望ましい．しかし，前項のアンダフロー時のウィンドウの退避では，最も残しておきたいスタックトップのウィンドウから先に退避されてしまう．
3. スタックトップのウィンドウから先に退避されると，あるタスクがスタックトップからすこし離れたウィンドウだけが残っているという状況が発生する．さらにそれが繰り返されると，分断されたウィンドウが残っているという状況が発生する．これらはウィンドウ管理の複雑化を招き，オーバフロー，アンダフロー，タスクスイッチ時のオーバヘッドを不要に増加させる．

これらは全て，アンダフロー時のウィンドウの退避が原因である．

4.2 restore命令の特殊性とそれを応用したウィンドウ管理アルゴリズム

アンダフロートラップの唯一の原因であるrestore命令には，以下のような特殊性がある．

すなわち，restore命令は必ず手続きの出口にあるため，restore実行後のoutレジスタ(実行前のinレジスタ)の値は，戻り値とspだけが必要であり，他のレジスタの値は破壊されても構わない．なぜなら，呼び出されていた手続き側では，もはや終了しているために使うことはなく，また呼び出し側では，それらのレジスタは手続き呼び出しによって揮発するレジスタと考えているために，利用していないからである．呼び出し箇所へのリターンアドレスも，このレジスタの一つに格納されているが，restore命令は必ずreturn命令の遅延スロットで実行されるため，リターンアドレスももはや参照されることはない．

故に，アンダフロートラップの処理においても，戻り値とspのレジスタだけをinからoutにコピーした後，今まで使用していたウィンドウ内に，localレジスタとinレジスタを復元すればよい．つまり，アンダフローが発生した時は，restore命令の実行前後でCWPの値を変えなくても，仮想的にウィンドウを一つ戻すことができる．

このようにすると，アンダフロートラップの発生時に，他のタスクのウィンドウを退避する必要がなくなるため，上記の問題は全て避けられる．そして，あるタスクのウィンドウが残っている場合は，最も残しておきたいスタックトップ近辺のウィンドウが必ず残っているはずで，分断されることは決してない．

以上の変更を行なったアンダフロートラップハンドラのアルゴリズムを以下に示す．

1. restore命令で，使用していたウィンドウに移動．
2. 戻り値とspの値を，inからoutにコピー．
3. inレジスタとlocalレジスタをスタックから復元．
4. トラップを発生したrestore命令の次の命令(NPCで示されるアドレス)にジャンプし，その遅延スロットでステータスレジスタをトラップ発生前の状態に戻す．

最後のステップ4は，save命令2回とrett命令によるトラップからの復帰，そしてトラップを起こしたrestore命令の再実行に相当する処理である．そのようにしないのは，WIM上の1のビットの状態によっては，save命令やrett命令でダブルトラップが起きる可能性があるためである．

オーバフロートラップハンドラ，タスクスイッチのアルゴリズムには，前出の単純なアルゴリズムと本質的な違いはない．主な変更点は，WIMに関する処理が増えることと，アクティブウィンドウのフラッシュがなくなることである．

4.3 制限事項

上記のようにウィンドウ管理アルゴリズムを変更する場合，以下のような制限が生じる．

1. restore命令による加算を行なわない．
   save命令と同様に，restore命令にもウィンドウの移動と同時に加算を実行する機能があるが，この機能を使うことができない．この機能は，コンパイラの局所最適化によって使われているため，その最適化を行なわないように，コンパイラに手を加えなければならない．(gccでは簡単にこの変更を行なうことができる．)このために，場合によっては手続きからの復帰の処理が1命令増えることがあるが，それよりも，レジスタウィンドウを使うことによる手続き呼び出しの高速化の効果の方が大きいと考えられる．
2. 戻り値を返すレジスタが制限される．
   現在利用可能なSPARC用Cコンパイラは，戻り値を返すのに一つのレジスタしか使用しない．もし，構造体などを複数のレジスタを使って返すようなコンパイラがあれば，コンパイラかアンダフロートラップハンドラを変更する必要がある．

5 おわりに

本稿では，PIE64の並列処理管理カーネルの実装において工夫した，レジスタウィンドウにおける高速タスクスイッチング手法について述べた．本手法では，ウィンドウを単なるバンクとは考えずに，手続き呼び出しの高速化というレジスタウィンドウの特長を生かしつつ，タスクスイッチの高速化を実現した．本手法はPIE64に限らず，高速タスクスイッチングが要求される用途に，レジスタウィンドウを持つプロセッサを使う場合に，適用可能である．

今後の課題は，並列処理管理カーネルの実装を終了させ，タスクスイッチング時間や全実行時間などを定量的に評価することである．

なお，本研究の一部は，文部省特別推進研究 No.62065002 の一環として行なわれた．

参考文献

[1] Patterson, D.A. and Sequin, C.H.: "RISC I: A Reduced Instruction Set VLSI Computer", *Proc. of 8th Annual Intl. Symp. on Comp. Arch.*, pp.443-457 (1981).

[2] Tamir, Y. and Sequin, C.H.: "Strategies for Managing the Register File in RISC", *Trans. on Comp.*, Vol.C-32, No.11, pp.977-988 (1983).

[3] Garner, R.B., et. al.: "The Scalable Processor Architecture (SPARC)", *Proc. of COMPCON88*, pp.278-283 (1988).

[4] Kleiman, S.R. and Williams, D.: "SunOS on SPARC", *Proc. of COMPCON88*, pp.289-293 (1988).

[5] 小池, 田中: 並列推論エンジン PIE64, bit 臨時増刊 並列コンピュータアーキテクチャ, 共立出版, Vol.21, No.4, pp.488-497 (1989).

[6] 日高, 小池, 田中: 並列推論エンジン PIE64 の推論ユニットのアーキテクチャ, コンピュータシステム研究会 CPSY90-44, 電子情報通信学会技術研究報告, Vol.90, No.144, pp.37-42 (1990).

[7] 日高, 小池, 田中: 並列処理シンポジウム JSPP'91 論文集, 情報処理学会, pp.69-76 (1991).