2L−10　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の実現

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

1　はじめに

SPARCなどで採用されているレジスタウィンドウという機構は、手続き呼び出しの際、レジスタを介して引数や戻り値の受け渡しを行い、レジスタ内容の退避・復元の手間を省くことによって、手続き呼び出しを高速化するものである。しかしながら、この上でマルチスレッドの処理を行おうとした場合、コンテクストスイッチの際に全てのレジスタの退避・復元を行うと、非常に大きなオーバーヘッドとなり、性能が低下してしまう。

本稿では、重複を持つレジスタウィンドウにおけるタスクスイッチングの高速化手法について述べる。本手法の特長は、手続き呼び出しの高速化という本来の特長を残したまま、タスクスイッチの高速化を実現しているという点である。

2　レジスタウィンドウ

SPARC[1]は、図1に示すような重複のあるレジスタウィンドウを持っており、これにより手続き呼び出しが高速化されている。

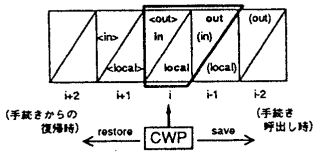

図1: SPARCのレジスタウィンドウ

つまり、呼び出し側のinレジスタと手続き側のoutレジスタが重複しており、ここで引数や戻り値を受け渡すことにより高速な手続き呼び出しを実現している。現在のウィンドウは、Current Window Pointer(CWP)によって示され、手続き呼び出しの際のsave命令、手続きからの復帰の際のrestore命令によってウィンドウが移動する。

ウィンドウはサイクリックにつながっているため、手続き呼び出しが深くなってウィンドウが1周すると、オーバーフロートラップが生じ、1周前のウィンドウの内容をメモリに退避して空きを作る。逆に、restore命令が重なって、以前にオーバーフローでメモリに退避したウィンドウが必要になった時には、アンダーフロートラップが生じ、メモリからの復元を行う。これらは、Window Invarid Mask(WIM)を1にしておくことにより検知する。

3　マルチスレッドの実装

3.1　従来の手法

このレジスタウィンドウ上でマルチスレッドを実装するため、幾つかの手法が用いられてきた。

もっとも単純な方法は、コンテクストスイッチが起こるたびに全ウィンドウをフラッシュすることである[2]。しかし、ウィンドウの退避・復元にはかなりの時間を要し、スイッチのオーバーヘッドは大きなものとなる。

また、高速なスイッチが要求される場合には、各スレッドを別のウィンドウに割り付けて、ウィンドウをそれぞれ個別に使う方法が考えられている[1]。これは、CWPの変更だけでスイッチが行えるため非常に高速であるが、手続き呼び出しの高速化というレジスタウィンドウ本来の特長を完全に失っている。

3.2　異なるスレッドのウィンドウの混在化

そこで、レジスタウィンドウとしての特長を生かしつつ、複数のスレッドをレジスタウィンドウ上に混在させることを考えてみる。図2の上図に示すように、アクティブでないスレッドのWIMを1にすることにより、うまくウィンドウの切り分けができるように見える。しかし、実際には次のような問題が生ずる。

図2を見ると、アンダーフローの処理において、Tb-1をメモリに退避してからTa-2を復元しなければいけない。つまり、アンダーフローの際に、他のスレッドの退避という余分な処理が必要となる。しかも、ここで退避されるウィンドウは、次にそのスレッド(Tb)に処理が移った時に真っ先に必要となるスタックトップのウィンドウである。

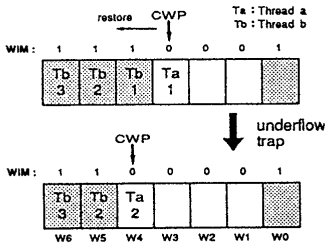

図2: 混在化による問題点(1)

さらに図3のように、スタックトップから少し離れたウィンドウしか残っていない状態でコンテクストスイッチが起こると、各タスクのウィンドウが分断されてしまうという状況が起こりうる。

3.3　問題点を解決した新手法

上記の問題点は、アンダーフローの際にウインドウを退避することに起因している。そこで、restore命令の性質に着目してみる。

restore命令は、必ず手続き呼び出しの出口で実行される。つまり、この時点で呼び出し側に戻るために必要なものは、戻り値とスタックポインタの値、リターンアドレスの3つだけで、それ以外に手続き中で使われたレジスタの値はすべて必要ない。さらに、restore命令は必ずreturn命令の遅延スロットで実行されるため、

Fast task switching with register windows
Kei YAMAMOTO, Yasuo HIDAKA, Hanpei KOIKE, Hidehiko TANAKA
The University of Tokyo

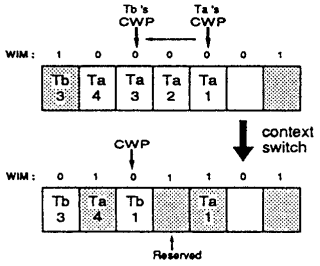

図 3: 混在化による問題点 (2)

リターンアドレスももはや必要なくなっている。したがって、呼び出し側では戻り値と sp が (呼び出し側の)out レジスタに入っていることのみを期待しているのであり、また手続き側ではその 2 つ以外のレジスタはすべて破壊しても構わないということになる。

そこで、アンダーフロートラップの処理としては、戻り値と sp のレジスタだけを in から out にコピーして、今まで使用していたウインドウ内に呼び出し側の in と local レジスタを復元することにすれば、他のスレッドのウィンドウを侵食することなく仮想的にウィンドウを一つ戻すことができる (図 4)。

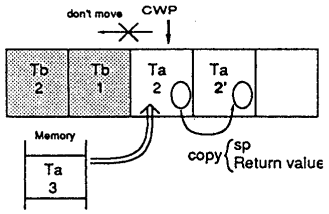

図 4: ウィンドウの退避を行わないアンダーフロートラップ処理

このようにすると、アンダーフローの発生時に他のスレッドのウィンドウを退避する必要がなくなるため、上記の問題点はすべて避けられる。

4 評価

上記手法の有効性を検証するため、評価実験を行った。

まず、実験環境としては、SPARC チップ上にレジスタウィンドウの数を可変にできるようなエミュレータを載せ、その上でテストプログラムを走らせた。

また、テストプログラムとしては、LaTeX のソースファイルのスペルチェックをするプログラムを作成した。これは、3 つのスレッドから成っており、それぞれがバッファを通してデータをやりとりする。このバッファが full または empty になるとコンテクストスイッチが起こる。

そして、レジスタウィンドウの使い方として、次の 3 つの戦略を比較した。

戦略 1 複数のスレッドが混在し、それぞれのスレッド間に空きのウィンドウを挟む。

戦略 2 複数のスレッドが混在するが、それぞれのスレッドは隣接しており、現在アクティブなスレッド用に 1 つだけ空きのウィンドウを用意する (図 5)。

戦略 3 ウィンドウ上に存在するスレッドは 1 つのみで、スイッチのたびに全ウィンドウをフラッシュする。

戦略 1 と 2 の違いは、1 に比べて 2 の方がウィンドウを無駄なく使える反面、各スレッドのスタックトップのウィンドウの out レジスタが隣のスレッドの in レジスタとして使われてしまっているので、スイッチのたびにその重複部分を退避・復元しなければならないことである。

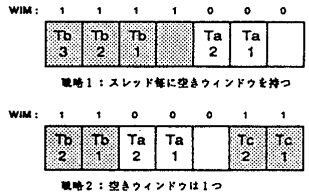

図 5: 戦略 1 と 2

測定の結果を図 6 に示す。

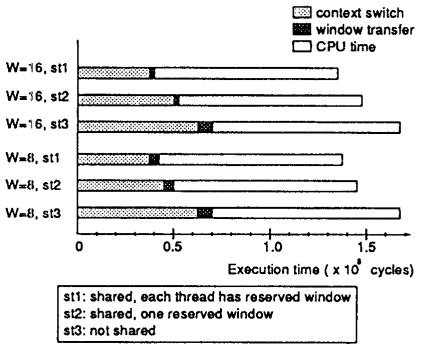

図 6: 実行時間の比較

これを見ると、

- 従来の手法 (戦略 3) よりも、今回提案した手法 (戦略 1,2) の方がスイッチにかかる時間が短くて済み、実行時間にも有意な差が見られる。
- 戦略 2 を 1 と比べると、ウィンドウが有効に利用できるという利点よりも、スイッチのたびに転送が起こるという欠点が効いているようである。
- 戦略 1,2 では、ウィンドウを増やすとウィンドウの転送にかかる時間が減っている。つまり、ウィンドウを増やした分だけ有効に使われて、オーバーフロー、アンダーフローが減ったということである。一方戦略 3 では、W=8 ですでにウィンドウが十分足りているようである。

などということが分かる。

5 おわりに

本稿では、手続き呼び出しの高速化という利点を損なわずに、レジスタウィンドウ上でコンテクストスイッチを素早く行わせる手法について提案し、その評価をおこなった。

今後の課題としては、スレッドのスケジューリングとの関係の考察、処理機構のハードウェア化の検討などがあげられる。

参考文献

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

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