2D−4

浮動小数点レジスタウィンドウを用いた擬似ベクトル処理

位守弘充　伊藤元久　中村宏　中澤喜三郎

筑波大学電子情報工学系

1. はじめに

近年、スーパースカラ方式などにより複数の演算器などをパイプライン的に並行動作することで、高速化を図るなど、スカラプロセッサの性能向上は著しいが、科学技術分野に適用させた場合、実効性能が著しく低下することが多い。

これは、それらの分野ではデータ領域が非常に大きく、データの時間的局所性がない等の理由によって、プロセッサの高い処理能力に貢献する筈のキャッシュメモリが有効に働かないことが多いためである。

一方、科学技術計算用のベクトルプロセッサはキャッシュに依存せず、主記憶をマルチバンクに分割し、アドレス付けをインターリービングすることで擬似的にパイプライン化し、主記憶とのデータ転送をベクトル命令によりパイプライン的に処理している。また大容量のベクトルレジスタを設け、ベクトルロード/ストア命令のベクトル長を長くすることで、主記憶へのaccess latencyが性能に与える影響を軽減している。

そこで我々は、通常のスーパースカラプロセッサに対して主記憶のパイプライン化、プリフェッチ機能［4］、キャッシュに代る必要十分なレジスタの設定、という機能追加を行なうことにより、ベクトル命令の処理をスカラ命令を垂直マイクロプログラム的に用いて処理することを考えた。すなわち上記のような機能追加により演算用データを充分に供給でき、浮動小数点演算パイプラインを切れ目なく稼働させることができることになる。これによりスーパースカラ方式により並列に実行される複数のスカラ命令で、1つのベクトル命令の処理を擬似的に実行する擬似ベクトル処理［1］が可能となる。

一般にレジスタ数を増やすためには、命令のレジスタ指定フィールド長など命令セットアーキテクチャの変更が必要であるが、我々は浮動小数点レジスタのレジスタウィンドウ化［3］を行うことによりこの問題を解決した。これにより提案するアーキテクチャは既存のアーキテクチャとの上位互換性を保つことができる。

本稿では擬似ベクトル処理の原理、及びその評価結果を示し、提案方式の有効性を示す。

2. 擬似ベクトル処理の原理

2.1 擬似ベクトル処理

図1にウィンドウ構成例を示す。図1では論理的なウィンドウ2のregister30はウィンドウ3のregister10と同じレジスタを指し、物理レジスタとしては70となる。この構成では総レジスタ数は88である。

図1を用いた擬似ベクトル処理の原理を図2に示す。今、i番目のループはウィンドウ1を利用して演算を実行しているものとする。これと並行してi+1番目のループを実行するのに必要なデータをキャッシュを介さずに、現在用いていないウィンドウ（裏ウィンドウ、この場合はウィンドウ2）のレジスタ（プリフェッチ部）に直接供給を行う。次にi+1番目のループ実行時には、現在どのウィンドウを使用しているか示すCurrent Floating Register Window Pointer(CFRWP:PSW内にウィンドウ番号を示し得る数ビットを設ける)を1つインクリメントし、ウィンドウ2を使用ウィンドウとする。そして、すでに必要なデータが格納されているプリフェッチ部のデータを利用して演算を実行し、これと並行してi+2番目のループで必要なデータをウィンドウ3のプリフェッチ部にロードを行う。

このようにループごとにウィンドウを切り換えて、使用していないウィンドウのプリフェッチ部にデータの供給を行うような命令（後述）を用意し、かつ主記憶のパイプライン化によってレジスタへのデータ供給のための十分なメモリスループットを実現し、主記憶へのaccess latencyと、演算実行がうまくバランスしていれば、2命令並列のスーパースカラ方式のプロセッサでは、スカラ命令のみを用いてデータ供給と演算を並列に実行することが可能となる。これはスカラ命令によって、ベクトルプロセッサにおけるベクトル処理と等価な処理ができること（擬似ベクトル処理）を示している。

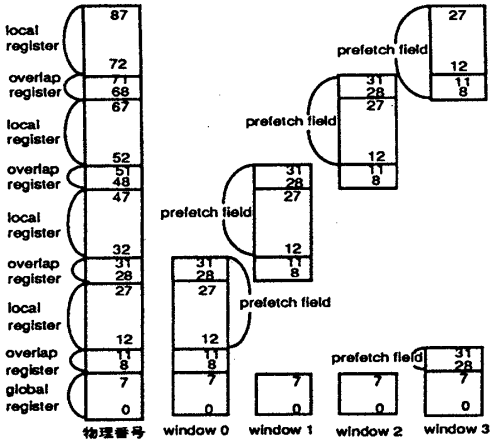

図1　レジスタウィンドウ構成

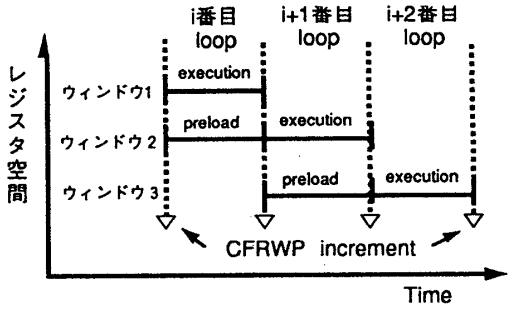

図2 擬似ベクトル処理の原理

2.2 命令セットアーキテクチャ上の拡張

上記のような処理を行う場合、命令セットアーキテクチャ上、追加する命令は以下のようなものでよい。

CFRWPenable:　レジスタウィンドウ機構を用いるか否かを示す、特権命令。もし用いないのであれば、ウィンドウ0のみを使用することになり、完全に拡張前のアーキテクチャと互換性を保つことができる。この時、CFRWPは0に設定する。

CFRWPinc:　CFRWPの値をこの命令によってmodulo n（nはウィンドウ構成数）でインクリメントする。

FRPreLoad:　データを主記憶からCFRWPが指すウィンドウの次のウィンドウの指定レジスタに直接ロードを行う。この命令の動作は通常のロード命令と同様であるが、キャッシュミス時でもデータキャッシュのブロック転送は行わない。従ってキャッシュ内のデータは全く変化しない。

FRPostStore:　現在のウィンドウの1つ前のウィンドウのレジ

Pseudo Vector Processing based on Floating-point Register Window

Hiromitsu IMORI,Motohisa ITO,Hiroshi NAKAMURA,Kisaburo NAKAZAWA

Institute of Information Science and Electronics, University of Tsukuba

スタから主記憶へデータのストアを行う。この命令の動作も通常のストア命令と同様であるが、キャッシュミス時にはキャッシュは無関係でありキャッシュ内のデータは全く変化しない。

このように浮動小数点レジスタウィンドウによる擬似ベクトル処理は、数種類の命令を追加するだけで実現可能であり、また既存のアーキテクチャとも上位互換性を保つことができる。

2.3 擬似ベクトル処理の一例

図3に図1のレジスタ構成における内積計算の例を示す。ここでは解りやすくするため、単純な2命令並列スーパースカラ方式とした。また、各命令は1マシンサイクルで実行完了するとする。

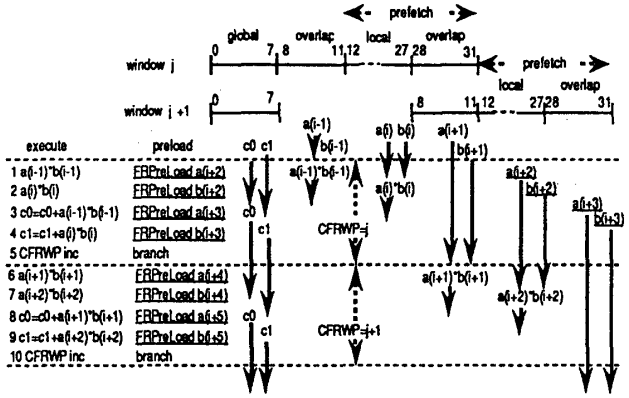

図3 ウィンドウ構成における内積計算の例

図3の場合例えば、2サイクル目でb(i+2)のプリフェッチ命令が発行され、7サイクル目の演算（a(i+2)*b(i+2)）で用いられている。よってstallなしに演算を実行するには、主記憶へのアクセス遅延は4サイクル以下である必要がある。しかし実際にはcurrent windowで使用可能なレジスタを十分活用し、ループアンローリングを行なうなどして1ループの実行時間を長くすることで、プリフェッチ命令を発行してから実際にそのデータを利用する間隔を長くすることができ、ある程度主記憶へのアクセス遅延が大きくても隠すことができる。

3. 性能評価

キャッシュメモリが有効に働かないようなデータ領域が大きく、また局所性がないような問題に対する浮動小数点レジスタウィンドウ化の効果の評価を行う。そのためデータキャッシュは必要なデータは予め格納されていないcold cacheの状態であるとしている。

今回アーキテクチャとしてHewlett-Packard社のPA-RISC1.1アーキテクチャ［2］を用いたが、本稿で提案する手法は、他のスーパースカラ方式を採用したアーキテクチャにおいても当然適用できる。

評価するあたって、どの程度の性能改善が図られたかを示すために、以下に述べる3つのタイプを想定する。

（1）normal: PA-RISC1.1 アーキテクチャ

（2）pipelined: （1）に対して主記憶のパイプライン化のみを行ない、プリフェッチ機能は追加していない

（3）Window: （2）に対してプリフェッチ機能の追加、浮動小数点レジスタウィンドウ構成に拡張を行なったもの

3.1 評価環境

表1のような仮定に基づいて評価を行った。また制御方式に関する詳細な仮定は以下のように置いた。

・2命令同時発行のスーパースカラ方式：stallが発生すると後続命令はインタロックされる。発行される2命令はロード系（FRPreLoad等）、浮動小数点演算、整数演算（CFRWPincを含む）、その他（分岐等）のいずれか異なるグループに属する。

・データ依存関係：依存関係のある命令間で先行命令が浮動小数点演算なら5マシンサイクル以上、ロード命令ならば2マシンサイクル以上、FRPerLoad命令なら20マシンサイクル以上後続命令をインタロックする。

表1仮定項目

アーキテクチャ	Hewlett-Packard社 PA-RISC1.1
制御方式	2命令並列スーパースカラ方式
データサイズ	64ビット
ウィンドウ構成	4構成（図1）
主記憶へのアクセス遅延	20サイクル
命令キャッシュ	warm cache状態：キャッシュミスは考慮しない
データキャッシュ	ブロックサイズ：16byte(（1）、（2）)
Memory allocation	最適に配置されている：バンク衝突なし

3.2 評価結果

ベンチマークとして、Livermore Kernel Loopの中から抜粋して、その調和平均を評価に用いた。また各ベンチマークに対し、可能な限りのループアンローリングを含むハンドコーディングを行い、前節で述べた3種類のプロセッサに対し最適化コードを生成し1回のループあたりの処理サイクルの評価を行っている。評価結果を図3に記す。

図からも主記憶のパイプライン化による効果は大きいといえる。またWindowではnormalのcold cacheと比較して7倍程度の性能改善がみられ、ほとんどnormalのall hit時と遜色がない性能が得られている。

このことからもプリフェッチ機能を追加し、レジスタウィンドウ構成にする効果は大変大きいといえる。

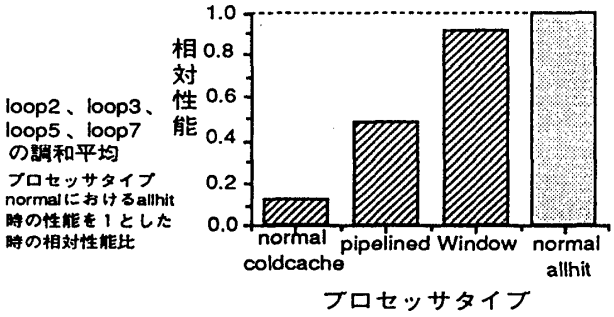

図3 各プロセッサタイプによる性能向上比

4. おわりに

スーパースカラプロセッサに対して、主記憶のパイプライン化、浮動小数点レジスタウィンドウ機能を追加することによって、スカラ命令によって擬似的にベクトル処理を可能とするアーキテクチャを提案した。この擬似ベクトルプロセッサアーキテクチャは数種類の命令を新たに命令セットアーキテクチャに追加するだけで実現が可能であり、また既存のアーキテクチャとの上位互換性も保つことができる。

この手法によって、キャッシュメモリが有効に働かないような科学技術計算分野のアプリケーションにおいて生じる実効性能の低下を改善でき、キャッシュメモリの有効性に性能が左右されず、高い実効性能が得られることがわかった。

参考文献

［1］位守弘充,伊藤元久,中村宏,中澤喜三郎,"疑似ベクトル処理向きメモリアーキテクチャの一提案",情処研報,ARC-91-8,1991

［2］Hewllet-Packard Company,"PA-RISC1.1 Architecture and Instruction Set Reference Manual",Manual Part Number 09740-90039,1990

［3］Sun Microsystems,"The SPARC Architectural Manual,Version 8",Part No.800-1399-09,1989

［4］Jean-Loup Baer,Tien-Fu Chen,"An Effective On-Chip Preloading Scheme To Reduce Data Access Penalty",Supercomputing'91,1991