スライドウィンドウを用いた擬似ベクトル処理

6L−4

李航　位守弘充　中村宏　朴泰祐　中澤喜三郎
筑波大学 電子・情報工学系

1. 序論

大規模科学技術計算には、データ領域が非常に大きくデータ参照に時間的局所性が少ないという特徴がある。従って、これらの計算ではデータキャッシュが有効に働かず、通常のスカラプロセッサの性能はキャッシュミス時の主記憶アクセスペナルティにより著しく低下する[4]。

この問題に対し、我々は浮動小数点レジスタウィンドウを用いた擬似ベクトルプロセッサ PVP-RW (Pseudo Vector Processor based on Register Window) を提案し、その有効性を確認している[1][2][3]。しかし、ウィンドウ構成に自由度が無いためアプリケーションによってはレジスタを有効に活用できない、という問題点があった。

そこで、我々は浮動小数点レジスタにスライドウィンドウを用いる新しい擬似ベクトルプロセッサ PVP-SW (Pseudo Vector Processor based on Slide Window) を提案する。本稿では PVP-SW のアーキテクチャとその処理方式、並びに評価結果を示す。

2. PVP-SW のアーキテクチャ

PVP-SW のアーキテクチャは既存のスカラアーキテクチャの拡張として定義でき、拡張前のアーキテクチャに対し上位互換性を持つ。PVP-SWでは通常のスカラプロセッサに対し以下の機能追加がなされ、主記憶アクセスペナルティによって性能が低下することなく、スカラ命令により高速にベクトル処理が行われる（擬似ベクトル処理）。これらの機能追加のうち、最初の2点はアーキテクチャの拡張を必要とする。

・浮動小数点レジスタのスライドウィンドウ化
・命令の追加
・主記憶のパイプライン化

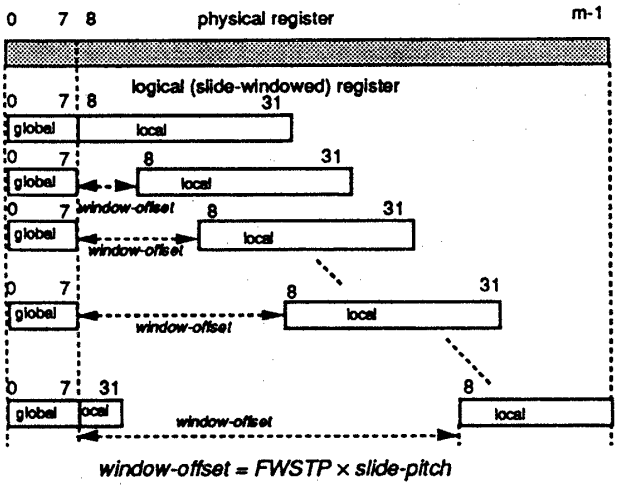

図1 浮動小数点レジスタのスライドウィンドウ構成

2.1. 浮動小数点レジスタのスライドウィンドウ構成

図1に浮動小数点レジスタのスライドウィンドウ構成を示す。1つの論理ウィンドウ内のレジスタ数は、拡張前のアーキテクチャで指定できるレジスタ数と一致しなければならない（ここでは32とする）。従って、1つのウィンドウ内のレジスタは拡張前のアーキテクチャと同様に指定することができる。レジスタは、全てのウィンドウに共通である global register と、ウィンドウ毎に異なる local register から成る。ここでは global register の数を8とした。各ウィンドウの local register の位置は *window-offset* によって与えられる。window-offset の指定単位を *slide-pitch* と呼ぶ。総レジスタ数と slide-pitch はアーキテクチャとして定義されなければならないが、ここでは一般的に各々をm, slide-pitch とおいて説明する。

物理レジスタ番号#R と論理レジスタ番号#r の関係：

$0 \le r \le 7$ の時：#R = #r

$8 \le r \le 31$ の時：

$$\#R = \{(\text{window-offset} + \#r - 8) \bmod (m-8)\} + 8$$

アクティブウィンドウ：

次節で示す追加命令以外の命令は、アクティブウィンドウ内のレジスタを用いて実行される。

FWSTP (FR Window STart Pointer)：

アクティブウィンドウを指すポインタとして n bit の FWSTP を浮動小数点 status register に割り当てる。FWSTP のビット数 n は、以下の式で与えられる。

$$n = \left\lceil \log_2 \left(\frac{m - 8\ (\#\ \text{of global registers})}{\text{slide-pitch}} \right) \right\rceil$$

FWSTPE (FR Window STart Pointer Enable)：

スライドウィンドウの特徴を用いるか否かを表す 1 bit のフラグで、PSW内に割り当てられる。

アクティブウィンドウのwindow-offset：

アクティブウィンドウの window-offset は、FWSTP, FWSTPE を用いて以下のように与えられる。

if FWSTPE = 0 then window-offset = 0
if FWSTPE = 1 then window-offset = FWSTP × slide-pitch

2.2. 追加命令

FWSTPenable：特権命令。FWSTPE の設定を行う。

FWSTPset：非特権命令。s/i (1 bit) と window-stride (n bit) field を持ち、以下のように FWSTP を設定する。

if s/i = 0 (set) then FWSTP := window-stride
if s/i = 1 (increment) then

$$\text{FWSTP} := (\text{FWSTP} + \text{window-stride}) \bmod \left(\frac{m - 8}{\text{slide-pitch}} \right)$$

Pseudo Vector Processing based on Slide-Windowed Registers
Hang LI, Hiromitsu IMORI, Hiroshi NAKAMURA, Taisuke BOKU, and Kisaburo NAKAZAWA
Institute of Information Sciences and Electronics, University of TSUKUBA

FRPreload：n bit の window-stride field を持つ非特権命令。メモリからのデータを、アクティブウィンドウより window-stride 分 先のウィンドウ内指定レジスタに転送する。キャッシュミス時には主記憶からレジスタに直接データが転送されキャッシュの更新は行われない。

FRPoststore：n bit の window-stride field を持つ非特権命令。アクティブウィンドウより window-stride 分前のウィンドウ内指定レジスタからのデータをメモリに転送。キャッシュミス時にはキャッシュの更新は行われない。

2.3. PVP-RW方式との比較

我々が以前提案していた PVP-RW のアーキテクチャは、以下の制約を持つ PVP-SW と等価であり、PVP-SW のアーキテクチャに完全に包含される。

・slide-pitch が 20 である。

・FWSTPset 命令中のwindow-strideが常に 1 である。

PVP-RW と比較すると PVP-SW は slide-pitch が小さくアクティブウィンドウの切り換え距離（どれだけスライドさせるか）を FWSTPset 命令中の window-stride として指定できる、という特徴を持つ。そのためPVP-SWではアプリケーションの性質に応じて柔軟にレジスタを利用することができ、コンパイラによるサポートも容易になる。

3. 擬似ベクトル処理の原理

図 2 に、slide-pitch が 2 である PVP-SW 上での DAXPY ループのオブジェクトコードを示す。また、図 2 のコードを実行した時のレジスタ使用状況を図 3 に示す。図 3 において影つきの部分は aX(i+2)+Y(i+2) の処理を表す。図よりわかるように 3 つのループに渡ってソフトウェアパイプライン的に処理が行われる。

あるデータに対するロード要求から実際にそのデータが利用されるまでの時間間隔を *permitted latency* と呼ぶ。permitted latency が主記憶アクセスレーテンシーより短いと性能は低下する。図 2 のコードでは配列 X, Y いずれに対しても permitted latency は 7 サイクルである。しかし、FRPreload 命令の window-stride の値（図 2 では <+1>）をさらに大きくすることにより、ロード要求をさらに前のループで行えるため、permitted latency を長くすることができる。この手法により主記憶アクセスレーテンシーによる性能低下を防ぐことができ、スカラ命令による高速な擬似ベクトル処理が可能となる。

```
Loop:  FRPreload    Y(i+1) -> r30 <+1>
       ADD          r29 + r30 -> r29        % aX(i) + Y(i)
       FRPreload    X(i+2) -> r31 <+1>
       MULT         r7*r31  -> r31          % a * X(i+1)
       FRPoststore  r29 <-0> -> newY(i)
       FWSTPset     +1 (FWSTP <- FWSTP + 1)
       % branch is omitted
```

図 2　PVP-SWにおけるオブジェクトコード

4. 評価

Livermore Fortran Kernel #1 ～#14 を用いて、以下の 4 つのプロセッサモデルに対し評価を行った。

<original> Hewlett-Packard 社の PA-RISC 1.1 Architecture に基づくモデル。データキャッシュは cold cache。

<PVP-RW 68> PA-RISC 1.1 Architecture を拡張したPVP-RWモデル。ウィンドウ総数 3 、レジスタ総数68。

<PVP-RW 88> PA-RISC 1.1 Architecture を拡張したPVP-RWモデル。ウィンドウ総数 4 、レジスタ総数88。

<PVP-SW 64/2> 本稿で新たに提案する PVP-SW方式モデル。PA-RISC 1.1 Architecture の拡張であり、レジスタ総数64、slide-pitchとして 2 を仮定したもの。

評価においては、2 命令同時発行superscalar 制御方式、in-order 命令実行、主記憶アクセスレーテンシー 20MC (マシンサイクル)を全てのモデルに対し仮定した。また <PVP-RW> と <PVP-SW> においては、主記憶はパイプライン化されスループットは 8Byte/MC であるとした。

評価結果を図 4 に示す。図 4 よりわかるように、提案する <PVP-SW> は <original> に比べ非常に高速である。また <PVP-RW 68> <PVP-RW 88> と比較しても、総レジスタ数が 4 ないし 24 も少ないにもかかわらず、全てのカーネルにおいて性能は良い。

5. 結論

本稿では、PVP-SWのアーキテクチャと処理原理、並びに評価結果を示した。その結果、新たに提案するPVP-SW は以前より提案していた PVP-RW方式と比べて、少ないレジスタで高い処理能力を達成することがわかり、PVP-SW の有効性を確認できた。

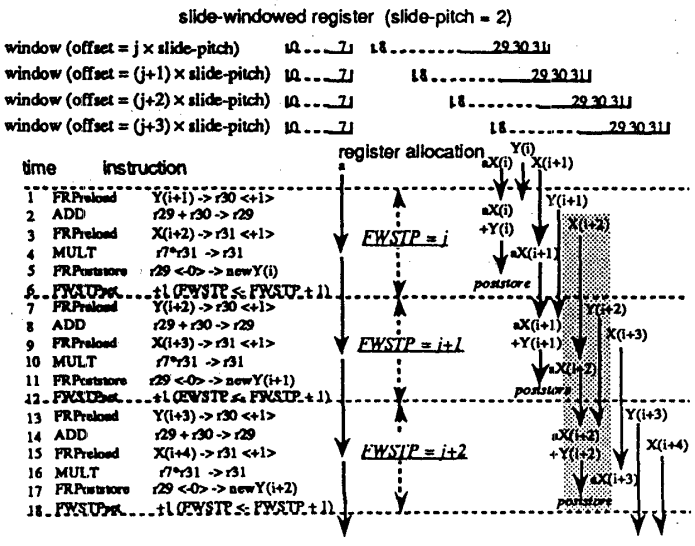

図 3　PVP-SW におけるレジスタ割り付け

参考文献

[1] 位守他, "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", 情報処理学会第 44 回全国大会, 2D-4

[2] 中村他, "レジスタウィンドウとスーパースカラ方式による擬似ベクトルプロセッサの提案", JSPP '92, pp367-374

[3] NAKAZAWA et al., "Pseudo Vector Processor based on Register Windowed Superscalar Pipeline", Supercomputing '92, pp.642-651

[4] M.L.Simmons et al., "Performance Evaluation of the IBM RISC System/6000", Supercomputing '90, pp132-141

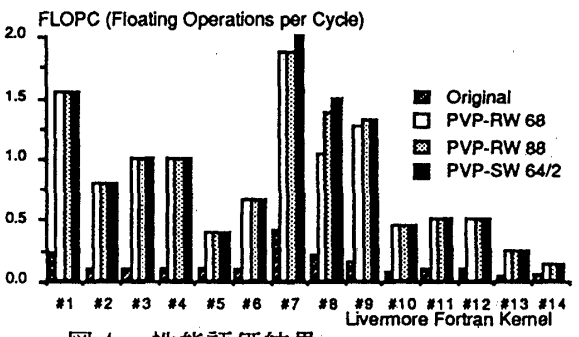

図 4　性能評価結果