ジオメトリプロセサ Procyon − コンパイラ −

4F−11

西崎慎一郎† 新井正樹‡ 小沢年弘‡

† (株) 富士通ソーシアルサイエンスラボラトリ ‡ (株) 富士通研究所

1 はじめに

ジオメトリプロセサ Procyon[1] は、3DCG 用に設計された VLIW アーキテクチャであり、二つのデータ空間と浮動小数点レジスタファイルを持つ。Procyon プロセサで浮動小数点レジスタを扱う場合、演算は同じレジスタファイル同士だけでしか行なえず、load,store は片方のデータ空間を対象にしか行なえない。異なるデータ空間や浮動小数点レジスタファイルへのデータの移動にはコストがかかるため、この移動をなるべく少なくするような工夫が必要になる。

本論文では、これらの特徴に対応するために行なった C 言語の拡張、それに対応した C コンパイラの特徴について述べる。

2 バンク指定 pragma

二つのデータ空間と浮動小数点レジスタファイルを C 言語で制御するために、二つのバンクという概念を定義する (図 1)。

- data0
  データ空間 0、浮動小数点レジスタファイル 0、整数レジスタファイル。
- data1
  データ空間 1、浮動小数点レジスタファイル 1、整数レジスタファイル。

プロセサの特徴を活かし実行速度を向上させるためには、アセンブラとの連携を強めるために、C のソースレベルでバンクを細かく制御できるような仕組みが必要である。

Procyon 用 C 言語では、変数、関数の戻り値、関数引数の受渡しに対して、使用するバンクを pragma で指定

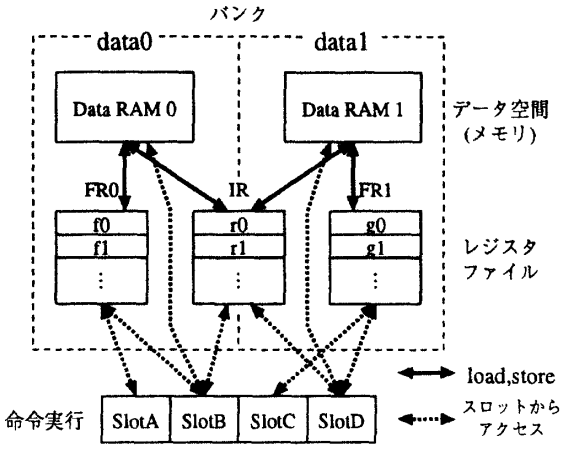

図 1: データ空間・レジスタファイルとバンクの対応

するよう拡張する。pragma の有効範囲は、C 言語の宣言文と同様に pragma が書かれた位置以降のブロック内、およびその内部ブロックとなる。

pragma には以下の三種類がある。

- default のバンクを指定する pragma
  pragma で指定されていない変数や関数に対する default バンクを指定する。この指定がない場合は data0 のバンクが指定されたことになる。
- 変数に対する pragma
  変数のバンクを指定する。ポインタ変数のバンクと、ポインタが指すデータのバンクは同じである。整数変数がレジスタに割り付けられた場合にはバンクの指定は意味がなくなる。
- 関数に対する pragma
  関数の戻り値と各引数のバンクを指定する。引数のバンクを省略した場合は default のバンクが指定されたことになる。

異なるバンク間でのデータの演算や代入がある場合、コンパイラは、それらの前にデータを同一のバンクへ移動するコードを生成する。ポインタ自体を異なるバンクへ移動することは通常意味がないので、警告を出す。

図 2 の pragma の記述例では、9 行目の比較、10 行目と 12 行目の演算、13 行目の引数設定、そして 14 行目の戻り値を返す箇所で、バンク間のデータの移動が起こる。

Geometry processor Procyon − Compiler −

Shin'ichirou NISHIZAKI†, Masaki ARAI‡
and Toshihiro OZAWA‡

† FUJITSU SOCIAL SCIENCE LABORATORY LTD.

‡ FUJITSU LABORATORIES LTD.

```
1: #pragma Procyon defaultbank data0
   /* default のバンク指定 */
2: float f0; /* data0 */
3: #pragma Procyon data0 func0(data0,data1)
4: #pragma Procyon data1 func1(data1,data0)
   /* 関数の戻り値と引数のバンク指定 */
5: float func0(float a0, float a1)
                /* a0 は data0、a1 は data1 */
6: {
7:   #pragma Procyon data1 f1
     /* 変数f1 のバンク指定 */
8:   float f1; /* data1 */
9:   if(a0 > a1) /* data0 > data1 */
10:    f0 = a0 - a1; /* data0 = data0 - data1 */
11:  else
12:    f0 = a1 - a0; /* data0 = data1 - data0 */
13:  f1 = func1(a0,a1);
       /* func1() の第 1 引数 data1 = data0 */
       /* func1() の第 2 引数 data0 = data1 */
14:  return f1;
     /* func0() の戻り値 data0 = data1 */
15: }
```

図 2: pragma の記述例

3　asm 文と専用組み込み命令

C 言語から Procyon のアセンブラを利用できる asm 文を用意する。

(例) a,b,c,d のうち最小の値を min に返す。

```
asm ("fsmall %f14,%0,%1; nop; nop; nop;
      fsmall %f15,%2,%3; nop; nop; nop;
      nop; nop; nop; nop;
      fsmall %4,%f14@W,%f15@E2; nop; nop; nop;"
     : a,b,c,d
     : min
     : "%f14","%f15"
     : MEM_NONE
     : 3);
```

また、Procyon プロセサが持つ浮動小数点数用の特殊演算命令などを有効に利用するため、専用の組み込み命令を用意する。C 言語上では関数の形で使用し、コンパイラが対応するコードに変換する。

(例) f,g の小さい方の値を dst に返す。

```
C 言語      dst = procyon_fsmall_data0(f,g)
アセンブラ  fsmall %f14,%f0,%f1
```

4　スタックと関数呼び出し

原則として、C の各関数が呼び出されるたびに各バンクのメモリ上にフレームを割り付け、それらをリンクしてスタックを構成する。コンパイラが生成するフレームの構造を、図 3に示す。

関数呼び出し時の処理で特徴的な事柄を以下に示す。

- バンク間のデータの移動を少なくするために、関数の引数の受渡しは、指定されたバンクのレジスタとフレーム内の所定の領域を使用して行なう。
- 呼び出された関数側でスタック・ポインタ、フレーム・ポインタを操作し、フレームの割り付けと復帰時の解放を行なう。

  また関数呼び出しのオーバーヘッドを減らすために、このポインタ操作中に、関数呼び出し前に行なわれたレジスタ書き込みの同期をとるように命令スケジューリング [2] を行なう。

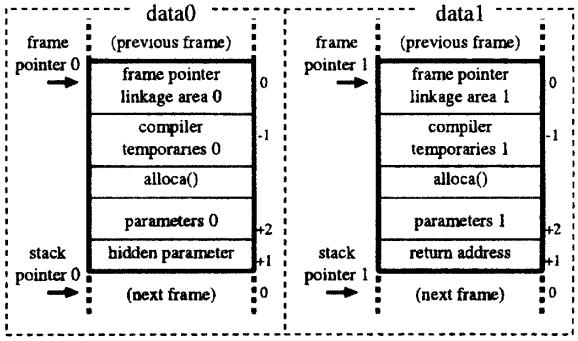

図 3: フレームの構造

5　おわりに

Procyon プロセサの特徴に対応するための C 言語の拡張と Procyon 用の C コンパイラの特徴について述べた。このコンパイラは、主要部を UtiLisp[3] で記述した。SunOS および Solaris 上で開発し、運用している。

謝辞

日頃ご指導いただく東京大学名誉教授の和田英一先生、富士通研究所の木村康則主任研究員に感謝致します。

参考文献

[1] 安里彰他, ジオメトリプロセサ *Procyon* -概要-, 情報処理学会第 55 回全国大会,1997.

[2] 新井正樹他, ジオメトリプロセサ *Procyon* - ソフトウェア・バイパス制御方式 -, 情報処理学会第 55 回全国大会,1997.

[3] Wada Laboratory. *UtiLisp Manual revision 2.0*, Department of Mathematical Engineering, University of Tokyo, January 1988.