計算機アーキテクチャ 151－7
（２００３．１．３１）

曖昧再利用によるステレオ画像処理の高速化

津邑公暁[†] 清水雄歩[†] 中島康彦[††]
五島正裕[†] 森眞一郎[†]
北村俊明[†††] 富田眞治[†]

カメラ画像を用いたステレオ画像処理において，最も計算量を要する視差測定部に対し関数値再利用を適用することにより，距離画像生成を高速化する手法を提案する。関数あたりに計算するピクセル値差の多重度を増やす一方，入力のマッチングに寛容さを持たせる曖昧再利用を適用することで，再利用による効果を上げつつ再利用のヒット率も保つことができることを示す。多重度が 1 のオリジナルプログラムと比較した場合において，メディア演算命令による実装のサイクル数削減率は，多重度を上げた場合でも最大 79%であったのに対し，曖昧再利用を適用した実装では最大 86%のサイクル数を削減することができた。

High Speed Stereo-Image Processing with Tolerant Function-Level Value Reuse

Tomoaki Tsumura,[†] Yuho Shimizu,[†] Yasuhiko Nakashima,[††]
Masahiro Goshima,[†] Shin-ichiro Mori,[†] Toshiaki Kitamura[†††]
and Shinji Tomita [†]

This paper describes the effectiveness of tolerant function-level value reuse against the disparity detecting function in stereo-image processing. The more pixel sets a function processes, the more effective the function-level value reuse is. But the less frequently the function will be reused, and the performance will hit its peak. We propose a way using tolerant reuse. It can increase the effectiveness of function-level value reuse and keep the frequency of reuse high. We show the maximum eliminated cycles with our method reaches to 86% while the maximum eliminated cycles with media processing instructions goes no further than 79%.

1. はじめに

画像処理は膨大な計算量を必要とすることが多く，計算機の性能向上がめざましい近年においても，更なる速度向上が必要とされている分野である。特に自動監視システムや移動ロボットのためのステレオビジョンといった，実時間処理が必要となるシステムにおいては，その要求は大きなものとなる。

ステレオ画像処理において最も計算時間を要するのは距離計算であり，更に細かく見るとその距離計算の際に必要となる視差測定である。SONY は，レーザ光を用いた距離測定のためのチップ Entertainment Vision Sensor を開発した。しかしアクティブセンサシステムはセンサ自身の消費電力を抑えにくいなどの欠点がある。これに対して TYZX などは，パッシブセンサを用いた視差測定専用プロセッサを使用したシステムを開発している。

このように，現在多くのステレオ画像処理システムでは専用プロセサを用いて距離計算を行っている。しかし，将来的には高速な汎用プロセサを用いてより安価に実現されることが予想されることから，プロセサに搭載されたメディア演算命令を用いることで距離画像生成を高速化しようという試みも行われている[1)]。

我々は，カメラ画像を入力として用いた距離画像生成プログラムに対し，最も計算量が必要となる視差検出時の，ピクセル値差分を求める関数に対して関数値再利用を適用し，ステレオ画像処理の高速化を図った。また，メディア演算命令を用いた場合の速度向上と，これを比較した。

以下 2 章ではステレオ画像処理の概要について述べ，3 章で関数値再利用のしくみについて述べる。4 章では距離画像生成への関数値再利用の適用手法について述べる。5 章で評価結果を示し，6 章でまとめを行う。

† 京都大学
Kyoto University
†† 京都大学 / 科学技術振興事業団さきがけ研究 21
Kyoto University / PRESTO, JST
††† 広島市立大学
Hiroshima City University

2. ステレオ画像処理

ステレオ画像処理とは，ある対象を 2 つの異なるカメラから観測して得られる 2 枚の画像から，その対象の距離画像を得る手法である。距離画像とは，対象までの距離情報を濃淡により可視化した画像で，距離画像内では距離が近いものは明るく，遠いものは暗く表示される。

一般にステレオ画像処理は，画像のノイズ除去/ぼかしなどの前処理を行ったのち，視差計算を行って距離画像を生成し，距離画像に再びノイズ除去/ぼかしなどの後処理を施す，という手順で行われる。

本章では，ステレオ画像処理の中心となる距離画像生成について述べる。

2.1 距 離 計 測

対象までの距離情報は，2 つのカメラの**視差**から求めることができる。視差とは一般には，ある計測対象となる基準点において，2 台のカメラの視線がなす角度 (輻輳角) の変化量として定義される。

カメラを平行に並べた 平行ステレオの場合，無限遠点を基準点とする。このため，基準となる一方のカメラ画像における，計測点の投影点に対し，もう一方のカメラ画像における同じ計測点の投影点が，カメラ画像内で何ピクセルぶんずれているかが，視差ということになる。

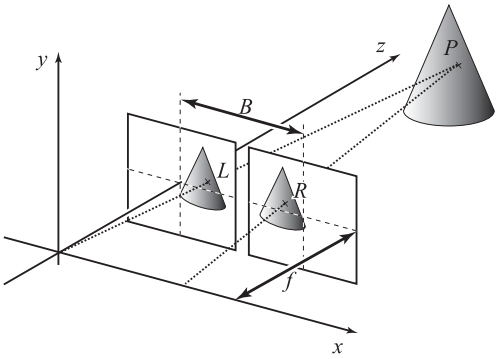

図 1 距離計測

いま，左カメラを基準とする。2 つのカメラを平行に並べた場合，左カメラの中心が z 軸 を通るように，また，左右カメラの並びが x 軸 に平行となるように 3 次元空間上に座標系をとる (図 1)。

x 軸とカメラとの距離を f，左右カメラ間の距離を B とする。また，計測点 P の左右画像上の投影点の x 座標をそれぞれ L_x, R_x とする☆。このとき，計測点 P の x 座標 P_x および z 座標 P_z に関して，

$$\begin{cases} P_x f = P_z L_x \\ (B - P_x) f = P_z (B - R_x) \end{cases}$$

が成りたつ。よって計測点 P までの距離 P_z は，視差 $d = L_x - (R_x - B)$ を用いると，

$$P_z = \frac{Bf}{d}$$

として計算できる。

2.2 対応点探索による視差検出

このように，視差を求めることができれば，対象までの距離情報を得ることができる。ただし，視差を測定するためには，左右の画像中の各ピクセルの対応を正しく調べる必要がある。これは対応点探索と呼ばれ，ステレオ画像処理において最も重要な問題のひとつである。

一般に，対応点探索でよく用いられるアルゴリズムに SSD (Sum of Squared Difference) がある。左右の画像の一方を基準とし，その画像内の，ある一つの点の周囲に小さなウィンドウを仮定する。他方の画像内のさまざまな点において，周囲に同じ大きさのウィンドウをとる。ここで，左右画像のウィンドウ内の各ピクセル値の差の総和が最小となるとき，それが左右の画像の対応点となる (図 2)。

仮定するウィンドウの大きさを大きくとるほど，正確に視差検出が行えることになるが，計算量は二次関数的に増加してゆく。

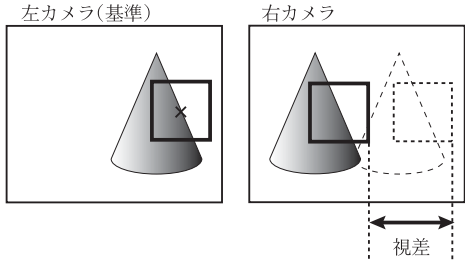

図 2 対応点と視差

距離計算の大部分は，このウィンドウ内のピクセル値比較が占めるため，この部分の高速化が図れれば，距離画像生成のための時間を大幅に短縮できることとなる。次章では，今回高速化手法として採用する再利用について述べる。

3. 区間再利用

区間再利用 (以下，再利用と略す) とは，関数呼出やループなどの命令区間において，その入力と出力のペアを記憶しておき，再び同じ入力によりその命令区間が実行されようとした場合に，過去の記憶された出力を利用することで命令区間の実行自体を省略し，高速化を図る方法である。

現在，数多くの研究が行われている値予測および投

☆ 平行ステレオの場合，投影点の y 座標は常に同一となる。つまり視差は x 軸方向のずれとして検出され，y 座標は計測点までの距離情報には関与しない。

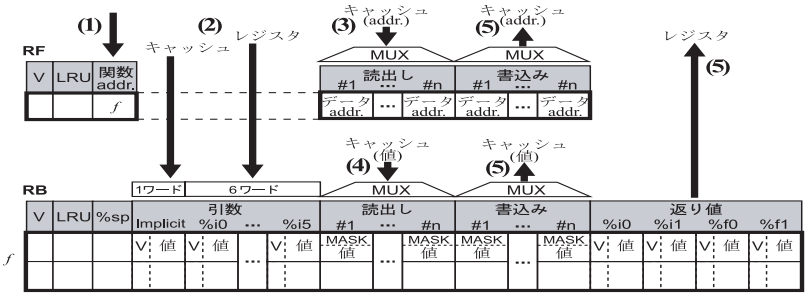

図 3 関数値再利用のための表構造

機的実行は，数多くの命令の投機あるいは実行結果の破棄が必要であるのに対し，再利用は実行する必要のある命令列そのものを削減できるという点で，従来とは発想の異なる高速化技術である。

以下本章では，関数値再利用とそのしくみについて説明する。

3.1 関数値再利用

関数値再利用を行うためには，関数の入出力値のペアを表に登録しておく必要がある。再び同じ関数を実行する必要が起こったとき，すでに同じ入力によりその関数が実行されている場合は，表から読みだすことで正しい出力値を直ちに求めることができる。入力のセットが完全に一致していれば，実行結果は必ず同じとなり，読みだした結果を検証する必要はない。また，副次的な効果として，冗長なロード/ストア命令や消費電力を削減できることも報告されている。

以下，関数値再利用の具体的な流れを述べる。ある関数 f を再利用するためには，f の実行時に f の入出力 f_{in}, f_{out} のみを表に登録する必要がある。

関数 f を呼びだす関数を f_p とすると，関数 f の入力となりうるのは，大域変数および関数 f_p の局所変数である。よって f の入出力をメモリマップ上で識別するためには，

- 大域変数と，f の局所変数との境界
- f の局所変数と，f_p の局所変数との境界

を確定する必要がある。つまり，与えられた主記憶アドレスが，大域変数であるか，または，どの関数の局所変数であるかを，何らかの方法で識別しなければならない。

我々は，SPARC ABI (Application Binary Interface)[2] に従って記述されたプログラムに対し，SPARC ABI の規定に基づく条件を仮定することで，これら変数識別の問題を解決している。詳細は文献[3] を参照されたい。

3.2 再利用機構

関数値再利用を実現するためには，**関数管理表 (RF)** および**入出力記録表 (RB)** が必要となる。ひとつの関数を再利用するために必要なハードウェア構成を図 3 に示す。複数の関数を再利用する場合には，この構成が複数組必要となる。

各表の V は有効エントリか否かのフラグである。また LRU はそのエントリが参照された頻度を表わすカウンタ値のためのフィールドで，エントリ入れ換えの際に参照される。読み出しアドレスは RF が一括管理し，マスクおよび値は RB が管理することにより，読み出しアドレスの内容と RB の複数エントリを CAM により一度に比較する構成が可能となる。

RF は関数の先頭アドレスおよび読み出し/書き込みで参照される主記憶アドレスを保持している。また RB は，関数が呼び出されたときのスタックポインタの値 (%sp)，その関数に渡された引数，主記憶の読み出し/書き込みデータ，および関数の返り値を保持する。なお，各エントリ先頭の V は，そのエントリの有効性を表すフラグ，MASK はそのエントリの有効バイトを表すマスク値である。

返り値は，%i0〜1 (リーフ関数の場合 %o0〜1) または %f0〜1 に格納され，%f2〜3 を使用する返り値 (拡張倍精度浮動小数点数) は対象プログラムには存在しないものと仮定している。

3.3 再利用機構の動作

関数 f を再利用するためには，まず f の実行時に，局所変数を除外しながら，引数/返り値/大域変数，および f_p の局所変数に関する入出力情報を表に登録する。

引数レジスタのうち，読み出しが先行したものに関しては f_{in} として，また，返り値レジスタへ書き込まれたものは f_{out} として登録する。その他のレジスタ参照に関しては，登録する必要はない。主記憶参照も同様に，読み出しが先行したアドレスについては f_{in}，書き込みは f_{out} として登録する。

なお，3.1 で述べたように我々は SPARC ABI の規定に基づき再利用を行っている。いま，フレームポインタを %fp とする。SPARC ABI では，呼び出された関数 f から見た場合，第 6 word までの引数は %i0〜5 に，第 7 word 以降の引数は%fp+92 以降に入ることになる。しかしこの際，第 7 word 以降の引数の参照は %fp 相対で行われるとは限らず，f_p の局所変数との区別がつかない。よって簡単のため，本稿では引数が 7 word 以上の場合は再利用を行わないこととしている。

その後，基本的には復帰命令を実行した時点で，登録中のエントリの V フィールドに有効フラグを書き

込む。ただし復帰までに，他の関数の呼出，入出力数の容量オーバー，引数の第 7 word の検出，システムコールや割り込みの発生，などの擾乱が生じた場合はその時点で登録を打ち切る。

以後は f を呼び出す前に

(1) 関数先頭アドレスを検索し，RF に登録済かどうか調べる
(2) 引数が完全に一致する RB エントリを探す
(3) 主記憶読み出しデータを参照する
(4) 主記憶読み出しデータの一致比較

を行い，全ての入力が一致した場合，(5) 登録されている出力，すなわち返り値，大域変数および局所変数を書き戻すことにより f の実行を省略する。

4. 関数値再利用の適用

今回は，ステレオ画像処理において最も計算時間を要するピクセル値の差を求める関数に対して，再利用を適用することを考える。

本章では，今回評価を行った，視差測定に対する関数値再利用のいくつかの適用方法，および評価の比較対象とする，メディア演算命令を利用した場合の視差測定の実装について述べる。

4.1 ピクセル値差の計算

今回の実装では，各ピクセル値として 32bit 整数を用いた。上位 24bit を RGB の各 8bit に割りあて，下位 8bit は使用しない。

視差検出のためには，ピクセル値の差を多数求める必要があることはすでに述べた。この，2 つ (1 セット) のピクセル値を引数にとり，それらピクセル値の差を返す関数を，pixdiff(L,R) として定義する。今回ウィンドウの大きさは 9×9 としているため，1 度のウィンドウ比較で 81 回 pixdiff が呼ばれることになる。

```
pixdiff(L, R){
  return(   abs(L>>24&0xff, R>>24&0xff)
          + abs(L>>16&0xff, R>>16&0xff)
          + abs(L>> 8&0xff, R>> 8&0xff) );
}
```

図 4 ピクセル値の差を計算する関数

4.2 メディア演算命令の利用

従来手法としてメディア演算命令を利用した場合の効果を調べ，関数値再利用の効果との比較を行う。SPARC には V9 アーキテクチャ以降，マルチメディア拡張命令セットである VIS[4] が実装されており，これを利用することにする。

4.2.1 PDIST 命令

VIS に含まれる pdist 命令は，ふたつの 64bit 変数を引数にとり，それぞれを 8 つ の 8bit ピクセル値とみなして，その 8 つ のピクセルペアの絶対差の総和を算出する命令である。また，その結果はデスティネーションとして指定した freg (浮動小数点レジスタ) にアキュムレートされるため加算命令が省略でき，ピクセル値の絶対差の総和計算を高速に行うことができる。

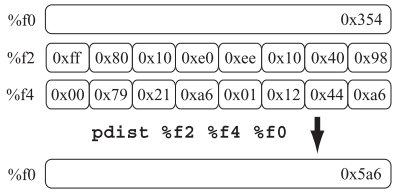

図 5 pdist 命令

4.2.2 PDIST によるピクセル値差計算の実装

今回は pixdiff の中身を，アセンブリレベルで pdist 命令を使用するように書き換えることで実装を行う。

関数 pixdiff の引数であるピクセル値は，reg (汎用レジスタ) %i0〜1 に格納されているが，pdist は入出力に freg を使用するため，pdist 実行前に引数を reg から freg へ，実行後に結果を freg から reg %i0 へ転送してやる必要がある。SPARC では，[%fp−16]〜[%fp] の範囲をこの転送の目的で使用することができる。pixdiff を pdist を用いて記述した例を図 6 に示す。

```
save    %sp, -112, %sp
st      %g0, [%fp-16]
st      %i0, [%fp-12]  /* 第 1 引数の転送 */
ldd     [%fp-16], %f2
st      %i1, [%fp-12]  /* 第 2 引数の転送 */
ldd     [%fp-16], %f4
fzero   %f0            /* accumulator 初期化 */
pdist   %f2, %f4, %f0  /* pdist */
std     %f0, [%fp-16]  /* 結果を転送 */
ld      [%fp-12], %i0
ret
restore
```

図 6 pdist を用いた pixdiff

pdist は引数に double word をとるため，この例では上位 1 word には 0 を詰めているが，最終的に結果はウィンドウ単位で総和をとるため，2 セットのピクセル値差計算を一命令で行うことができる。

一般に n セットのピクセル値差の総和を計算する関数は，pdist で書く場合，前項で述べたようにストア命令や加算命令を必要とせず，単純に freg への 4 つの ld と 1 つの pdist の，$n/2$ 回繰り返しで書くことができる。

4.3 再利用の適用

背景やオブジェクトといった同じ対象の像を構成するピクセルは同じピクセル値を持つものが多く，pixdiff は同じ組み合わせの引数で呼ばれることが多いことが

予想される。すなわち，再利用を適用した場合，そのままでも効果が得られると考えられるが，以下ではこの効果を更に向上させる方法について考える。

4.3.1 関数あたりのピクセル値差計算の多重度

視差計算の際に必要となるピクセル値の比較は，ウィンドウ単位で行われる。つまり，個々のピクセル値の差は最終的にウィンドウ単位で足しあわされたあとに比較されるので，必ずしも 1 ピクセルづつ差を計算する必要はない。

複数セットのピクセル値差の総和を計算するような関数を使用すれば，再利用表にヒットした場合に大きい効果が得られる。ただし，当然比較すべき入力が多くなるぶん再利用される確率は低くなる。ひとつの関数で計算するピクセル値セットの数を，以下本稿では**多重度**と呼ぶことにする。

今回は，多重度 2, 3 の関数 pixdiff2(L_0, L_1, R_0, R_1) および pixdiff3(L_0, L_1, L_2, R_0, R_1, R_2) を定義し，それを利用した場合の効果を調べることにした。なお，3.3 で述べたように再利用される関数の引数は 6 word までに限定しているため，1 関数でまとめて計算することができるピクセル値差は 3 セットまでである。

4.3.2 ループオーバヘッドの削減

前項で述べたように再利用できる関数の引数には制限があるため，再利用関数におけるピクセル値差計算の多重度は 3 より上げることはできない。しかしピクセル値差比較はウィンドウ単位で行われるため，この制限がなければ最大でウィンドウ単位の 81 セットまでをひとつの関数で計算することができる。

多重度を 9 に上げた場合のメディア演算命令を用いた実装との比較を行うため，これに対応する再利用の実装としてループオーバヘッドを削減した場合の評価を行う。

今回ウィンドウの大きさを 9×9 としたため，1 つのウィンドウのピクセル値総和計算には深さ 2 の 9 回ループが存在する。この内側ループを 3 つの pixdiff3 にアンローリングしたものと，pdist の $\lceil 9/2 \rceil = 5$ 回実行に書換えたものとの比較を行う。

4.3.3 曖昧再利用

一関数で計算するピクセル値差の多重度を上げた場合，再利用による効果は大きくなるが，引数が多くなるぶん再利用のヒット率が低くなってしまう。

一方，JPEG 圧縮のアプリケーションにおいて，再利用部に対する入力のマッチングを厳密にしないことによって，結果の精度をほぼ保ちながら再利用率を上げることができると報告されている[5]。

そこで今回はこの方法を適用して，ピクセル値計算の多重化による再利用率の低減を抑える。本稿では以下，この方法を**曖昧再利用**と呼ぶ。

具体的にはピクセル値において，各 RGB 表現部の下位数 bit をマスクすることでピクセル値の一致判別に寛容さを持たせる，という方法をとった (図 7)。マスク値としては fcfcfc00 および f0f0f000 を用い，再利用の効果の変化および距離画像への影響を調べた。

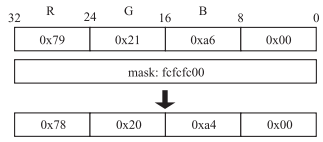

図 7 ピクセル値のマスク

5. 評　　価

評価には，再利用機構を実装した単命令発行の SPARC-V8 シミュレータに，VIS 命令 pdist を追加実装したものを用いた。各パラメータを表 1 に示す。キャッシュ構成や命令レイテンシは SPARC64-III[6]，pdist のレイテンシは UltraSPARC III[7] を参考にした。

D-Cache 容量	64 KByte	
ラインサイズ	64 Byte	
ウェイ数	4	
Cache ミスペナルティ	20 cycles	
Register Window 数	4 set	
Window ミスペナルティ	20 cycles/set	
ロードレイテンシ	2 cycle	
整数乗算 〃	8 cycle	
整数除算 〃	70 cycle	
浮動小数点加減乗算 〃	4 cycle	
単精度浮動小数点除算 〃	16 cycle	
倍精度浮動小数点除算 〃	19 cycle	
pdist 〃	4 cycle	
RB(引数) ⇔ レジスタ 比較	1 cycle	} test
RB(Read) ⇔ Cache 比較	4 byte/cycle	
RB(Write) ⇒ Cache 書込	4 byte/cycle	} write
RB(返り値) ⇒ レジスタ 書込	1 cycle	

表 1 シミュレーション時のパラメータ

ロードモジュールとしては，左右カメラの画像を入力とし距離画像を出力する距離画像生成プログラムを，gcc version 2.95.4 (-O2 -msupersparc) によりコンパイルし，スタティックリンクにより生成したものを用いた。今回入力として用いたカメラ画像および生成された距離画像を図 8 に示す。曖昧再利用時でも，出力画像に大きな崩れはないことが分かる。

表 2 に示したのは，それぞれの多重度において，再利用を適用しなかった場合に対する，pdist および関数値再利用の適用によるサイクル数削減率である。また図 9 には，多重度 1・再利用なし，の場合に要する実行サイクル数を 1 としたときの，各実装が距離画像生成に要したサイクル数の比率を示す。

グラフ中は左から順に，pixdiff，pixdiff2，pixdiff3，pixdiff3+アンローリング，の各実装のサイクル数である。また，各実装のグラフはそれぞれ左から順に，

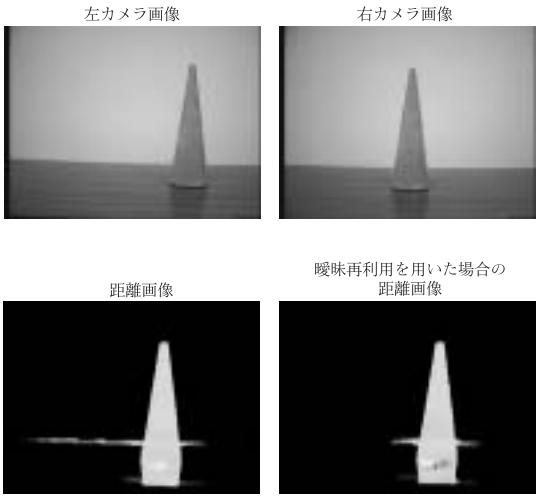

図 8 カメラ画像と，生成された距離画像

	pdist	再利用	曖昧 (fcfcfc)	曖昧 (f0f0f0)
pixdiff	17%	50%	52%	56%
pixdiff2	34%	47%	51%	62%
pixdiff3	38%	45%	46%	62%
pixdiff3+unroll	44%	46%	47%	63%

表 2 サイクル数削減率 (対 original)

original (再利用なし), `pdist` による実装, 再利用, 曖昧再利用 (mask: fcfcfc00), 曖昧再利用 (mask: f0f0f000) の結果を示している。なお，`pixdiff3`+アンローリングの `pdist` による実装は 4.3.2 で述べた通りである。

図中の凡例はサイクル数の内訳であり，exec は命令サイクル数，cache および window はキャッシュミスおよびレジスタウィンドウミスによるペナルティである。また test，write は表 1 で示した，再利用表の操作に要したサイクル数である。

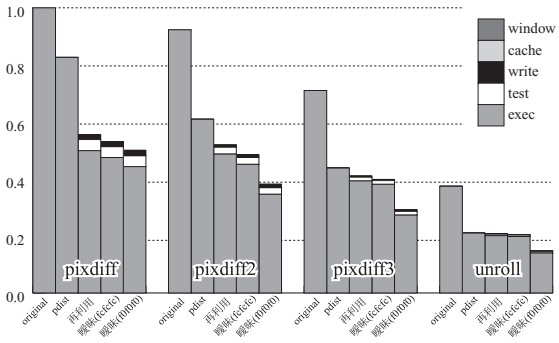

図 9 実行サイクル数の比較

多重度を上げていった場合，通常の再利用では original に対するサイクル数削減率が低下していってしまい，`pdist` による実装に較べると速度向上が少ない。しかし曖昧再利用を適用することで，多重度を上げていった場合でも 60%以上という，高いサイクル数削減率を維持できていることが分かる。

曖昧再利用にループアンローリングを組合せた場合では，再利用率を維持しながら再利用の効果を向上させることが可能となり，多重度 1・再利用なしの実装に対して最大で 86%のサイクル数を削減することができた。また，関数値再利用を適用した実装の全てが，性能上 `pdist` の実装よりも上回るという結果が得られた。

6. おわりに

本稿では，ステレオ画像処理において最も計算時間を要する視差測定に対し，関数値再利用を適用する高速化手法を提案し，性能評価を行った。

まず，いずれの場合においても，メディア演算命令を用いてハンドコーディングした実装よりも関数値再利用を用いたほうが良好な結果を得ることができ，ステレオ画像処理に対する再利用の有効性を示した。

ひとつの関数で計算するピクセル値差の多重度を上げることで，再利用しない場合に比したサイクル数削減率は僅かに低下したが，同時に曖昧再利用を適用することで，サイクル数の削減率を最大 63%まで引き上げることができた。結果，多重度を 3 まで上げ曖昧再利用とループアンローリングを適用したものでは，オリジナルプログラムに対して 86%のサイクル数を削減することができた。

参考文献

1) 清水雄歩, 中島康彦, 五島正裕, 森眞一郎, 北村俊明, 富田眞治: VLIW 型メディアプロセッサを用いたステレオ画像処理の評価, 情報処理学会関西支部 新・支部大会 (2002).
2) Paul, R.: *SPARC Architecture, Assembly Language Programming and C*, Prentice-Hall (1999).
3) 中島康彦, 緒方勝也, 正西申悟, 五島正裕, 森眞一郎, 北村俊明, 富田眞治: 関数値再利用および並列事前実行による高速化技術, 情報処理学会論文誌: ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG 6 (HPS 5), pp. 1–12 (2002).
4) Sun Microsystems: *The VIS™ Instruction Set*, 1.0 edition (2002).
5) Álvarez, C., Corbal, J., Salamí, E. and Valero, M.: On the Potential of Tolerant Region Reuse for Multimedia Applications, *Proceedings of the 15th international conference on Supercomputing*, ACM Press, pp. 218–228 (2001).
6) HAL Computer Systems / Fujitsu: *SPARC64-III User's Guide* (1998).
7) Sun Microsystems: *UltraSPARC® III Cu User's Manual* (2002).