Vol.2013-ARC-206 No.1
2013/7/31

リング型アレイアクセレータ向け 演算ライブラリの実装と性能評価

稲垣 慶和[†1†2]　原 祐子[†2]　姚 駿[†2]　中島 康彦[†2]

我々は，科学技術演算や画像処理，3 次元シミュレーションなどの演算速度向上，消費電力低減の両立を目的として，各演算器とシングルポートメモリを組にした構造をリング接続した構成のリング型アレイアクセラレータを提案してきた．しかし，アプリケーション開発者が，リング型アレイアクセラレータを利用し所望の性能を引き出すためには，ハードウェア構造や専用命令セットの理解と相当のチューニングコストが必要となる．開発者がアクセラレータを利用する際の負担を軽減し，その性能を最大限に利用可能な環境を提供するために，我々はリング型アレイアクセラレータと共に，自動で専用命令を出力するコンパイラや，専用ライブラリを開発する必要がある．本論文では，リング型アレイアクセラレータ，専用ライブラリの構成方式，およびリング型アレイアクセラレータの性能を引き出すためのライブラリ実装におけるチューニング方式について述べる．性能測定シミュレータによりライブラリの実行時間を測定し，リング型アレイアクセラレータの演算性能とデータ転送性能が全体の実行性能に及ぼす影響を確認した．また，汎用 CPU によるプログラムの実行時間を測定し，ライブラリ実行時間と比較したところ，リング型アレイアクセラレータを利用する場合，実行時間を最大約 50%削減可能であることを確認した．

1. はじめに

科学技術演算，画像処理，3 次元シミュレーションなどの高速化に向けて，ベクトル演算器を内蔵した高性能プロセッサや多数の演算ユニットを備える GPGPU の高性能化が進み，これらを使用したコンピュータシステムの研究・開発が進められている．高性能化のためには，演算性能の向上だけでなく，データ配置の効率化やデータ移動の高速化が必須であり，高性能インターコネクトの開発やソフトウェアによるデータ配置，通信に関する最適化の研究も進められている．我々は，演算速度向上と消費電力低減の両立を目的として，メモリ上のデータ移動を最小限に抑える為に各演算器とシングルポートメモリを組にした構造をリング接続し，命令写像をローテーションするリング型アレイアクセラレータ（Energy-aware Multimode Accelerator eXtension：EMAX）を提案してきた[1], [2]．しかし，命令写像によりデータ移動を最小限に抑え，多数の演算器とシングルポートメモリを効率的に使用する高効率な動作を実現するプログラムを作成するためには，ハードウェア構造の理解と高いチューニングコストが必要であり，アーキテクチャに特化したコンパイラや，専用ライブラリの開発が必要となる．そこで画像処理や 3 次元シミュレーションで使用されるステンシル演算処理を，リング型アレイアクセラレータ（EMAX）を動作させるライブラリとして実装し，その性能をシミュレータにより評価した．

†1 (株)富士通コンピュータテクノロジーズ
Fujitsu Computer Technologies Ltd.
†2 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

本論文では，2 章にてリング型アレイアクセラレータ（EMAX）の構成方式，およびライブラリの構成方式について述べる．3 章にて，ライブラリ内での 2 次元ステンシル演算の命令マッピング方式を述べ，4 章にてシミュレータによる性能評価と，汎用 CPU との実行時間の比較結果を提示する．

2. EMAX システム構成

本項では，まず EMAX の構造，システム構成，命令セットについて詳述し，ライブラリの構成について説明する．

2.1 EMAX アクセラレータの構成

図 1 のシステム構成図に示すように．EMAX は，マトリクス状に配置した複数の基本ユニットから構成される．各行には 4 つの基本ユニットが配置され，各ユニットは，図 2 に示すように 2 つの演算器（EX1 および EX2）と 1 つのローカルメモリ（アドレス生成用の EAG を含む），そしてローカルメモリから読みだしたデータをバス経由で一時的に保持する FIFO を 2 つ備えている．FIFO へのデータバスは同一行全てのユニットが接続されており，1 つのローカルメモリのデータを最大 8 つの FIFO へ供給することが可能である．また上端と下端のユニットが接続され，全体としてリング構造を構成している．EMAX とホスト PC は，USB3.0 により SDRAM 経由で接続されている．

2.2 EMAX アクセラレータのインタフェース

EMAX を動作させるには，まずホスト PC にて EMAX の各ユニットの演算種別，セレクタ設定情報，バス切り替え情報，実行回数などの情報を含んだ制御情報を生成し，処理対象データと共に SDRAM へ転送する．EMAX は，次回実行時に必要なデータの受信および出力結果の格納を行う．

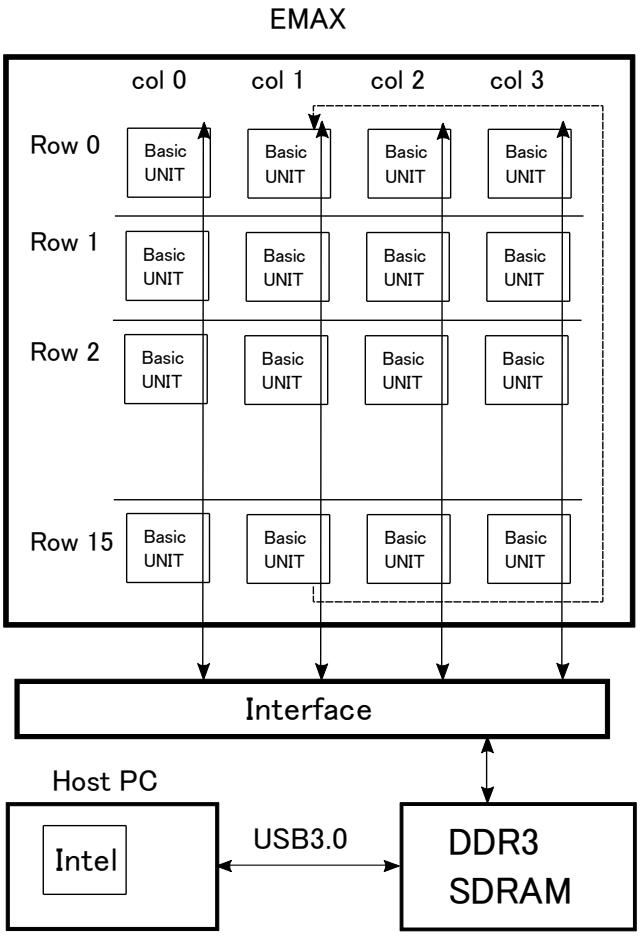


図 1　EMAX システム構成

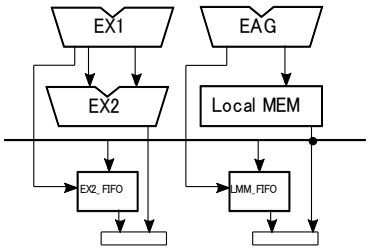


図 2　EMAX 基本ユニット

2.3 EMAX の命令

EMAX では，図 3(a)に示すように基本命令フォーマットが定義されている．各命令先頭部分の row と col は，命令がマップされる論理的なユニット位置を示す．そして dist は次回EMAX起動時に命令が写像されるrow方向への物理的な相対距離を示し，LMM のデータを効率的に再利用するために用いられる．ALU_OP には，図 3(b),(c)に示す EX1 と EX2 の演算命令を指定する．EX1 には EX2_FIFO からの load 命令を指定可能であり，同一行ユニットにおける狭い範囲のデータ再利用を可能とする．図 3(d)に示す MEM_OP は LMM または LMM_FIFO からの load 命令や store 命令が記述可能である．RGI は ALU_OP, MEM_OP が使用するレジスタの初期値を指定する．

```
Case1:@row#, col#, dist [count]  ALU_OP  RGI & MEM_OP RGI LMM_CONTROL
Case2:@row#, col#, dist [count]  ALU_OP  RGI
Case3:@row#, col#, dist [count]                  & MEM_OP RGI LMM_CONTROL
```

(a) Instruction Format

32bit　operation s 16bit[2] operations misc operations load from FIFO floating-point operations	add/add3/sub/sub3 mauh/mauh3/msuh3 mluh/mmrg3/msad/minl/minl3/mh2bw/mcas/ mmid3/mmax/mmax3/mmin/mmin3 ldb/ldub/ldh/lduh/ld fmul/fma3/fadd

(b) EX1 operations

32bit　operation s 16bit[2] operations	and/or/xor mauh/mauh3/msuh3

(c) EX2 operations

load from LMM or LMM_FIFO store to LMM	ldb/ldub/ldh/lduh/ld stb/sth/st/cst

(d) Memory operations

図 3　EMAX 命令

2.4 EMAX のライブラリ

プログラマが EMAX を効率的に使用するには，アプリケーションの処理を EMAX 命令にて記述する必要がある．しかし目的の動作を実現するには，ハードウェア構造と専用命令の理解が必須となる．プログラマの負担を軽減し，効率的にアクセラレータを利用させるため，EMAX 命令を生成するコンパイラの開発や，アプリケーションのカーネルをあらかじめ EMAX 命令を用いて実装したライブラリの開発が必要となる．理想的にはコンパイラにより命令を自動生成できるべきである．しかし，画像処理や 3 次元シミュレーションなどのカーネルを EMAX で実行し，高い実行効率や低消費電力化を実現するには，ハンドチューニングにより命令記述したライブラリ群の利用に分がある．

2.5 ライブラリの構成

図 4 に提案するライブラリ構成を示す．ライブラリ内に EMAX の制御情報を保持する．制御情報は EMAX 命令により記述する．アプリケーションは通常のライブラリ関数と同様に，ライブラリを呼び出すことで，EMAX を利用し演算結果を得る．図 3 の例では，まずアプリケーションが演算に必要となる入出力データの領域を獲得，初期化（input_0, input_1, output）する．そして各領域のアドレスを与えてライブラリを呼び出す．ライブラリ内では予め用意された制御情報に入出力データのアドレスを追加し，最終的に EMAX に転送する制御情報を生成する．SDRAM へは制御情報と共に，演算に使用される入力データを転送する．この時，入力データは直接アプリケーションが確保したメインメモリ領域から転送する．EMAX は制御情報に従い，指定された演算を指定回数繰り返し，演算結果を SDRAM へストアする．SDRAM へ格納された演算結果は随時ホスト PC 側のメインメモリ領域（output）へ転送される．

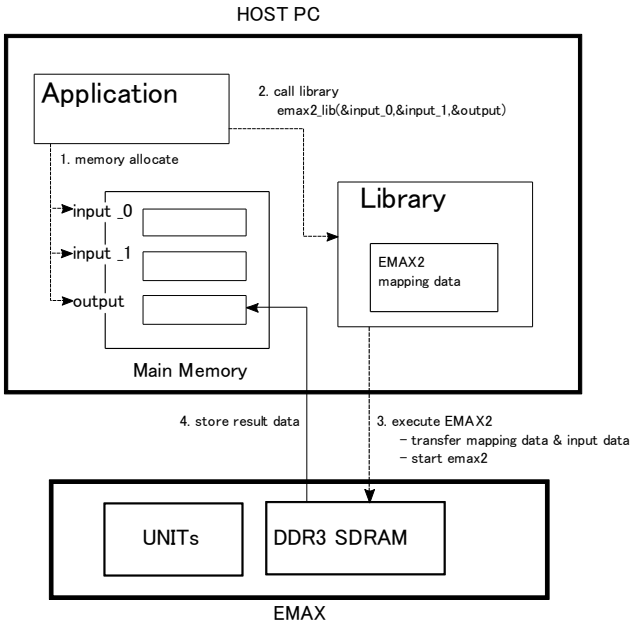
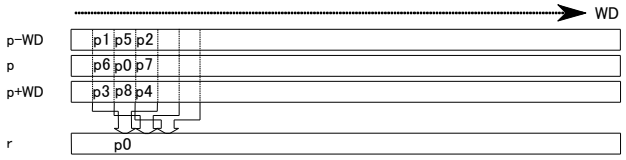


図 4　EMAX ライブラリ構成




```
void unsharp(unsigned char *p, unsigne d char *r)
{
    int t0,t1,t2,  j, k;                for (j=0; j<WD; j++) {
    int p0 = ((0  )*WD+(1   ))*4;           r[p0+0] = 0;
    int p1 = ((0-1)*WD+(1-1) )*4;           t0 = p[p0+1];
    int p2 = ((0-1)*WD+(1+1))*4;            t1 = p[p1+1] + p[p2+1] + p[p3+1] + p[p4+1];
    int p3 = ((0+1)*WD+(1 -1))*4;           t2 = p[p5+1] + p[p6+1] + p[p7+1] + p[p8+1];
    int p4 = ((0+1)*WD+(1 +1))*4;           r[p0+1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
    int p5 = ((0-1)*WD+(1   ))*4;
    int p6 = ((0  )*WD+(1-1 ))*4;           t0 = p[p0+2];
    int p7 = ((0  )*WD+(1+1))*4;            t1 = p[p1+2] + p[p2+2] + p[p3+2] + p[p4+2];
    int p8 = ((0+1)*WD+(1   ))*4;           t2 = p[p5+2] + p[p6+2] + p[p7+2] + p[p8+2];
                                            r[p0+2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

                                            t0 = p[p0+3];
                                            t1 = p[p1+3] + p[p2+3] + p[p3+3] + p[p4+3];
                                            t2 = p[p5+3] + p[p6+3] + p[p7+3] + p[p8+3];
                                            r[p0+3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

                                            p0+=4; p1+=4; p2+=4; p3+=4; p4+=4   ; p5+=4; p6+=4; p7+=4; p8+=4;
                                        }
                                    }
unsigne d char limitRGB(int c) {
    if (c<0x00) retur n 0x00;
    if (c>0xff) return 0xff;
    return c;
}
```

図 5　プログラム例（最内ループ）

3. 命令マッピング方式

本項では，画像処理に使用する鮮鋭化（unsharp）のカーネル（図 5）を例に，ライブラリ内での命令マッピング方式を示す.

3.1 基本マッピング

図 6 の EMAX 命令は，図 5 に示したプログラムの関数 unsharp()を実装した結果である．まず row#0, row#1, row#2 各行の col#0 ユニットの LMM に，入力となる 3 つの配列をプリフェッチする．例のような 2 次元ステンシル演算では，各配列の 3 つの連続したデータを使用するため，各行で 3 つの load 命令を発行している．この時，各行の col#1,col#2 の load 命令は，col#0 の LMM から各ユニットの LMM-FIFO に格納したデータを load できる．Load された 9 つのデータはレジスタを伝搬し，各行のユニットにおいて演算され，最終的な演算結果が row#8 のユニットにより LMM へ store される．各ユニットは，count で指定された要素数（WD）と同じサイクル数動作する．各ユニットは並列動作し，初回の結果出力以降は 9 回の load 命令，21 の演算命令，1 回の store 命令によって得られる結果が毎サイクル出力される．演算に使用される mauh/mauh3 は Xr, Yr，Zr の上位 16bit，下位 16bit をそれぞれ加算する命令，mluh は Xr の上位 16bit と下位 16bit を 8bit データとしてそれぞれ Yr と乗算する命令，mh2bw は Xr，Yr の上位 16bit と下位 16bit データを 8bit データとしてマージする命令である.

3.2 命令写像によるデータ転送削減

2 次元ステンシル演算では，図 7 に示す関数 unsharp()を HT 方向へ入力配列をインクリメントしながら複数回呼び出し，最終的な結果を得ることができる．このとき，

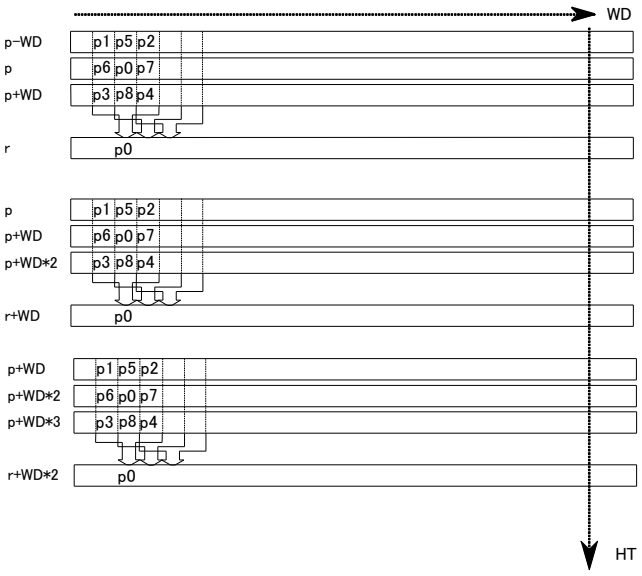
```
//EMAX2 @0,0,0 [WD]                                   & ld (r10+=,4),r1 rg i[p1]  lmr[,p-WD:,WD,0]
//EMAX2 @0,1,0 [WD]                                   & ld (r10+=,4),r2 rg i[p2]
//EMAX2 @0,2,0 [WD]                                   & ld (r10+=,4),r5 rg i[p5]

//EMAX2 @1,0,0 [WD] mauh  (r1 .l,r2.l),r11          & ld (r10+=,4),r 6  rgi[p6] lmr [,p:,WD,0]
//EMAX2 @1,1,0 [WD]                                   & ld (r10+=,4),r7  rgi [p7]
//EMAX2 @1,2,0 [WD] mauh  (r1 .h,r2.h),r1 2         & ld (r10+=,4) ,r0  rgi[p0 ]

//EMAX2 @2,0,0 [WD] mluh  (r0 .l,ri),r20   rgi[,.i239c0:]& ld ( r10+=, 4),r 3 rgi[p3] lm r[,p+WD:,WD,0]
//EMAX2 @2,1,0 [WD] mluh  (r0 .h,ri),r21  rgi[,.i239c1:]& ld  (r10+=, 4) ,r4 rgi[p4 ]
//EMAX2 @2,2,0 [WD] mauh  (r5 .l,r6.l),r15          & ld (r10+=, 4),r8 r gi[p8]
//EMAX2 @2,3,0 [WD] mauh  (r5 .h,r6.h),r1 6

//EMAX2 @3,0,0 [320] mauh 3 (r11 ,r3.l,r4.l),r11
//EMAX2 @3,1,0 [320] mauh 3 (r12 ,r3.h,r4.h) ,r12

//EMAX2 @4,0,0 [320] mluh   (r11,ri ),r13   rgi [,.i13c0:]
//EMAX2 @4,1,0 [320] mluh   (r12,ri ),r14   rgi [,.i13c1:]
//EMAX2 @4,2,0 [320] mauh 3 (r15 ,r7.l,r8.l),r15
//EMAX2 @4,3,0 [320] mauh 3 (r16 ,r7.h,r8.h) ,r16

//EMAX2 @5,0,0 [320] | or  (r1 5,0)>M2,r7
//EMAX2 @5,1,0 [320] mluh   (r15,ri ),r17   rgi [,.i15c0:]
//EMAX2 @5,2,0 [320] | or  (r1 6,0)>M2,r8
//EMAX2 @5,3,0 [320] mluh   (r16,ri ),r18   rgi [,.i15c1:]

//EMAX2 @6,0,0 [320] msuh3  (r20,r 7,r17) ,r20
//EMAX2 @6,2,0 [320] msuh3  (r21,r 8,r18) ,r21

//EMAX2 @7,0,0 [320] msuh ( r20,r1 3) | or (-,0)>M7,r20
//EMAX2 @7,2,0 [320] msuh ( r21,r1 4) | or (-,0)>M7,r21

//EMAX2 @8,0,0 [320] mh2b  w (r21 ,r20)                & st -,(ri+=,4)   rgi[,p0_out:,] lmw[,r:,320,0]
```

```
mauh ( Xr.[fhl], Yr.[fhl] )
  16bit[2] Xr + Yr

mauh3 ( Xr.[fhl], Yr.[fhl] )
  16bit[2] Xr + Yr + Zr

mluh  ( Xr.[fhl], Yr.[fhl], Zr.[fhl]  )
  8bit[2] * 9bit    16bit[2]

mh2bw ( Xr, Yr )
  merge sat(Xr.H16bit).sat(Xr.L16bit)
        .sat(Yr.H16bit).sat(Yr.L16bit)

{fhl}  f:fullword  h:byte3,byte2  H16bit,L16bit
       l: byte1,byte0 H16bit,L16biy
```

図 6　EMAX 命令による実装

unsharp()呼び出し毎に EMAX が起動され，row#1,row#2 のユニットの LMM にプリフェッチされた 2 つの入力配列データは, 次回 EMAX 起動時に使用されるデータと同じ配列となる．このデータを再利用し，データ転送回数を削減するために EMAX 命令の dist を使用した命令を利用する.

図 6 の各ユニットの dist に 1 を設定した場合, 次回 EMAX 起動時には各ユニットの命令が row 方向に 1 ずれた位置に写像される．図 8 は，図 6 の EMAX 命令が，各ユニットの EX1/EX2，EAG にマッピングされた様子である．dist に 1 が設定されていることにより，図 8 に示す row#1 の行の各ユニットで実行された load 命令は，次回実行時には row#2 の行の各ユニットにて実行される．この時 load すべき配列




```
for (i=1; i<HT-1; i++) {
    unsharp(&input[i*WD], &output[i*WD]);
}
```

図 7　プログラム例(最外ループ)

データは, 前回 EMAX を起動した際に, row#2 col#0 の LMM にプリフェッチされている．このデータを再度 load 可能となるため，ホスト PC から再度同じ配列データを EMAX へ転送する必要がなくなる．結果，dist を設定することにより，2 回目以降の EMAX 起動時には前回使用した 2 つの入力配列データが再利用可能となり, データ転送サイズを 1/3 に削減することができる．

3.3 命令マッピングの並列化

EMAX の資源を最大限使用し性能向上を図るため，最外ループを分割し，並列にマッピングすることを考える．図 7 に示す最外ループを 2 分割してマッピングする場合，ループ前半用と後半用の処理に分割し，図 8 と同様の命令配置ブロックを 2 つ用意し, それぞれマッピングすれば良い．しかし例題プログラムでは, 9 行のユニットが必要であり，上下に 2 つの命令配置ブロックをマッピングした場合，18 行分必要となるためマッピング不可能となる．

最内ループをマッピングした結果を参照すると，3 行分のユニットで入力データを load し，その後 2 回または 4 回の演算を繰り返して最終的な結果を得る．この時，load に使用するユニットでは演算器 EX1/EX2 が, 演算に使用するユニットでは EAG が空いている状態が多くなる．並列に命令マップする場合は，この空いた資源を効率的に利用することを考える．

並列マッピングの手順としては，まず図 8 に示す命令マッピングを 1 行置きに命令配置する．そして奇数行のユニットにも同様に 1 行置きにループ後半用の命令を配置する．そして load と各演算命令の依存関係を考慮し, 演算命令を，上方向で空きのある行の演算器 EX1/EX2 にマッピングしていく．結果を図 9 に示す．背景が白の演算器 EX1/EX2

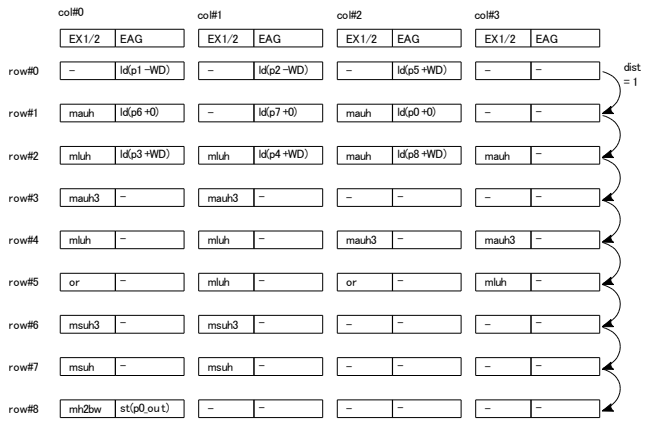


図 8　unsharp の命令配置

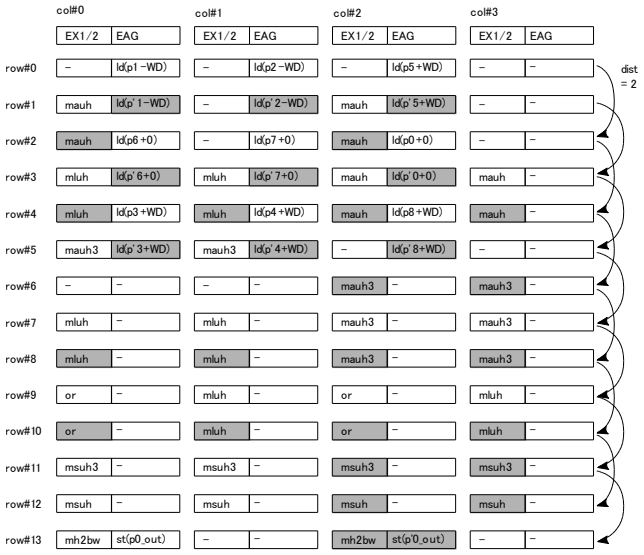


図 9　命令の並列配置

および EAG がループ前半用に使用される資源，灰色が後半用であり, 最終的に 14 行のユニットに命令マッピング可能となる．

また性能向上のためには，LMM のデータ再利用によるデータ転送時間削減が必要である．再利用可能な load 命令は 1 行置きに配置されているため，命令で指定する dist を 2 に設定する．起動毎に 1 行置きに命令が写像され，データの再利用によるデータ転送時間削減が可能となる．

4. 性能評価と考察

本章では，例示した unsharp のような 2 次元ステンシル演算と 3 次元シミュレーションに用いられる 3 次元ステンシル演算を複数の命令マッピングパターンを用いて実装し, EMAX の処理時間をシミュレートした結果を示し，汎用 CPU による実処理時間測定結果と比較する．

4.1 性能測定シミュレートモデル

EMAX システムのハードウェア仕様と，データ転送，実行制御方式を元に，動作時間を測定するシミュレータを開発し，動作性能の見積もりを行った．

表 1　ハードウェアの仮定

EMAX	動作周波数：200Mhz / 800Mhz
HOST-DDR3 転送性能 (USB3.0)	転送レート：400Mbyte/sec
DDR3-LocalMEM 転送性能	動作周波数：100Mhz バス幅　　：64bit
DDR3 SDRAM	容量　　：256Mbyte
Local MEM	容量　　：8Kbyte

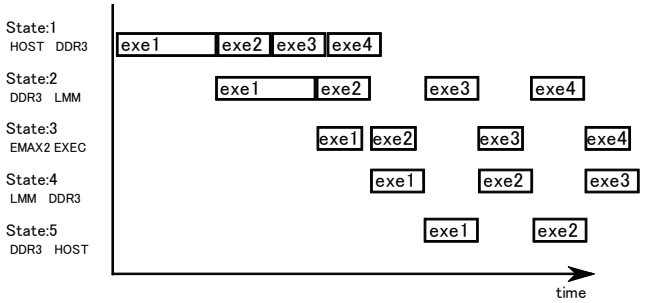


図 10　EMAX 動作ステート

表 2　ライブラリ実行時間

ライブラリ	EMAX 動作周波数 (Mhz)	実行時間 (usec)
unsharp 320x320 (dist=0)	200	4105
	800	4105
unsharp 320x320 (dist=1)	200	1958
	800	1958
unsharp 320x320 (parallel ,dist=2)	200	1965
	800	1965

表 1 に EMAX と各種メモリ，ホスト PC との転送性能の仮定を示す．EMAX の動作周波数は ASIC 化を考慮し，200Mhz/800Mhz を仮定した．

図 10 に EMAX 実行時の動作ステートを示す．EMAX の起動から終了は，5 つの動作ステートから構成される．ステート 1 は HOST PC から DDR3 SDRAM への入力データの転送．ステート 2 は DDR3 SDRAM から Local MEM へのプリフェッチ．ステート 3 は EMAX の各ユニットによる演算処理．ステート 4 は Local MEM から DDR3 SDRAM への出力データ排出，ステート 5 は DDR3 から HOST PC への結果データの転送となる．また各ステートはパイプライン的に動作することが可能である．また性能評価にあたり，ステート 2 とステート 4，ステート 1 とステート 5 の通信パスが衝突するためデータ転送速度への影響を考慮する必要がある．性能測定シミュレータ内で通信パスが衝突した場合，現在使用しているステート動作が完了することを待ち，待機中のステート動作を開始することとする．

4.2 ライブラリ性能の評価

3 章に述べた unsharp について，ライブラリ実装方法毎の性能評価結果を表 2 に示す．実装方法は，命令ローテーションなし（dist=0），命令ローテーションあり（dist=1），並列マップ（parallel, dist=2）の 3 種類である．動作周波数は 200Mhz,800Mhz の 2 パタンを仮定し，総実行時間を測定した．まず全ての結果において，動作周波数を問わず同じ結果が得られた．そこで unsharp（dist=1）について周波数を振ってシミュレートしたところ，図 11 の結果が得られた．周波数が 55Mhz を下回ると実行時間への影響が出はじめる結果となった．このことから EMAX の性能はデータ転送性能に左右される部分が大きく，EMAX 自体の動作周波数は 100Mhz 程度で十分であると考えられる．

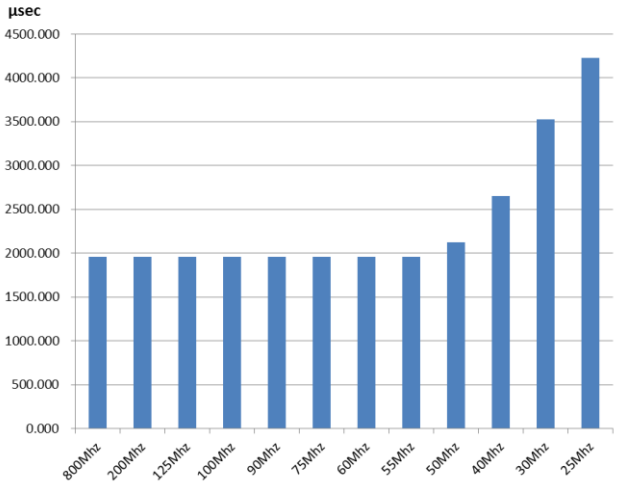


図 11　周波数別 EMAX 実行時間(unsharp)

次にライブラリ実装パタン別の評価を行った．命令ローテーションにより Local MEM のデータを再利用するパタンと，命令ローテーションなしでデータを再利用しないパタンでは，約 2 倍の実行時間の差となり，期待通りの結果が得られた．並列マッピングしたパタンにおいては，EMAX の起動回数が半分になり，演算レイテンシが 2 倍となるものの，総データ転送サイズは命令写像あり（dist=1）の実装パタンと同じである．そのため実行時間の向上は望めないと考えられる．シミュレート結果もほぼ同様であり，並列化パタンでは数 μ sec 実行時間が増えている．これは EMAX 実行毎のデータ転送量が倍に増えたため，データ転送の動作ステートの時間が倍になり，実行待ちステートの待ち時間が増えたことが原因と考えられる．

4.3 汎用 CPU との比較評価

EMAX と汎用 CPU の比較を行った．使用した汎用 CPU のリストとコンパイラ，コンパイルオプションを表 3 に示す．評価プログラムは前述した unsharp に加え，unsharp の配列数を 320x320 から 2048x2048 へ増加したプログラム（unsharp8k）と，3 次元ステンシル演算である．汎用 CPU によるプログラム実行時間の測定は，EMAX ライブラリ部分を C 言語で実装した関数に置き換えて実行時間を測定した．実行時間はデータ配列獲得処理や初期化部分は含まず，EMAX ライブラリ実行相当部分を置き換えた処理時間である．

表 3　比較に使用した汎用 CPU

CPU	コンパイラ	コンパイル オプション
Intel(R) Xeon(R) CPU E5405 2.00GHz	gcc 4.1.2	-msse2 -ffast-math
Intel(R) Pentium(R) 4 CPU 3.06GHz	gcc 4.6.3	-msse2 -ffast-math
SPARC64-VII+ 2860Mhz	solstudio12.3	-m64 –fast –DSPARC –DALLOW_SIMD -DM9000

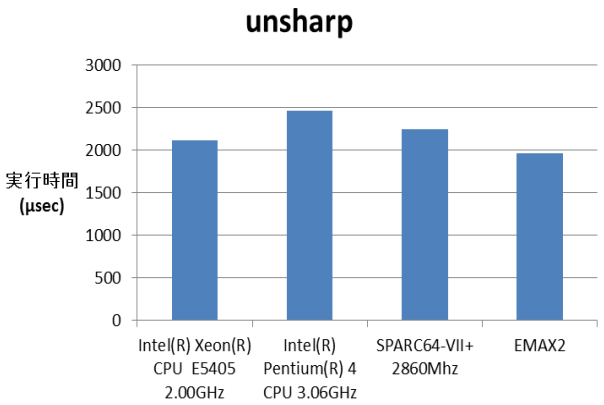


図 12　unsharp の実行時間

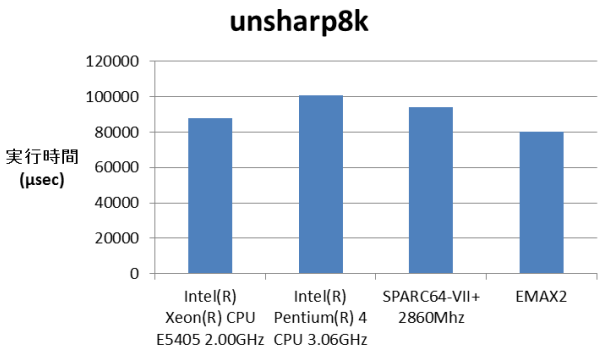


図 13　unsharp8k の実行時間

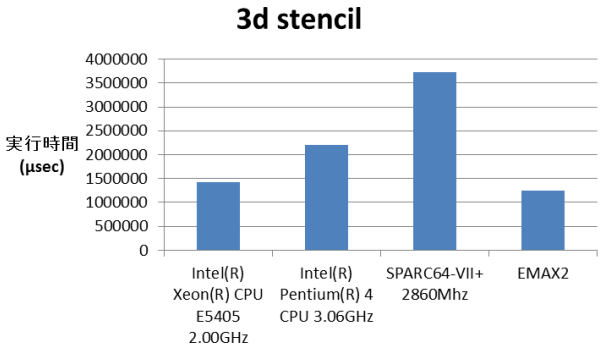


図 14　3d stencil の実行時間

図 12~14 に各汎用 CPU と EMAX による各プログラムの実行時間を示す. まず全てのプログラムにおいて EMAX が最短の実行時間となった．性能差は最大で約 2 倍である．また電力測定は未実施だが，EMAX の動作周波数は 100Mhz 程度を想定しており，対電力性能比は汎用 CPU と比較して, 実行速度以上に EMAX が優れていることが期待できる. また EMAX の複数接続にライブラリが対応することでデータ転送量増加による性能遅延の影響を受けず，さらなる処理速度の向上が見込める．

5. おわりに

本稿では, リング型アレイアクセラレータ EMAX 向けライブラリ構成方式およびマッピング方式を提案した．ライブラリ実装方式毎に周波数を振って実行時間をシミュレートしたところ，EMAX 内での演算処理時間はデータ転送時間に隠蔽される結果となり，ライブラリ実行時間を削減するためには，EMAX 内でデータを再利用させ，データ転送量を削減することが効果的ということがわかった。また汎用 CPU と実行時間比較を行ったところ，EMAX を利用する場合実行時間を最大約 50%削減できる結果が得られ，EMAX を利用することでアプリケーションの処理速度の高速化，低消費電力化が実現できる見通しが立った．しかしライブラリの実装・評価コストは依然高いため、今後はライブラリ生成手法を体系化し，ライブラリ自動生成フレームワークを確立させていく予定である．

謝辞　本研究の一部は科学研究費補助金（基盤(A)24240005，萌芽 24650020，若手（B）23700060），半導体理工学研究センター（超低電圧で稼働できる耐エラープロセッサの製造性を向上させる手法），および，JST-ASTEP 探索タイプ（課題番号 AS242Z02732H）による．

参考文献

1) 王昊，姚駿，中島康彦: "GCC の vectorizer を利用した演算器アレイ向け命令変換手法", 研究報告計算機アーキテクチャ(ARC), 2013-ARC-203 No.9, Feb. (2013)
2) 関賀，姚駿，中島康彦: "リング接続を利用しデータ移動を最小限にするアクセラレータの提案", 研究報告システム LSI 設計技術（SLDM）SIG Technical Reports, 2013-SLDM-159, Vol.17, pp.1-6, Jan. (2013)