EMAX7SPC-0001
Ver.0.1: Sep. 1 2023
Ver.0.2: Apr. 1 2024

EMAX7/ACAP (IMAX3) Architecture Handbook
– In-Memory Accelerator eXtension –

Nara Institute of Science and Technology
Computing Architecture Laboratory
Accelerator Group

Table of contents

List of Figures

List of Tables

Chapter 1

IMAX3 Hardware

1.1 Chiplet and Vector Length

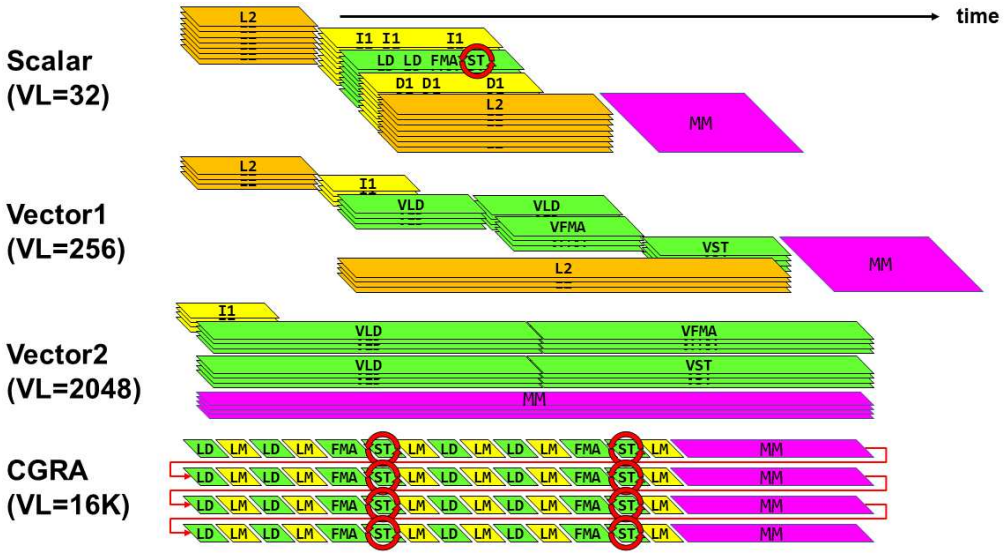

Figure.1.1: Variation of Vector Length

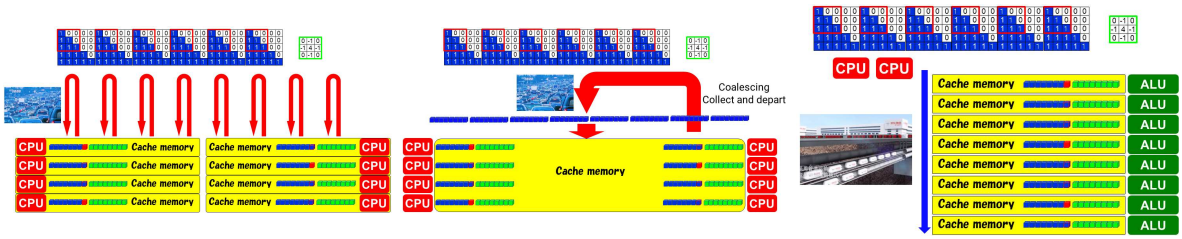

Figure.1.2: Multicore system, GPGPU and CGRA

チップレット化には，チップ間データ転送オーバヘッドを最小化するために，十分なベクトル長が必要である．そして，IMAX3 は，チップレット指向アーキテクチャである．図 1.1 は，ハードウェア構成とベクトル長の関係である．32 要素程度の SIMD は，スカラプロセッサに搭載できる．256 要素以上の SIMD は，ベクトルと呼ばれる．このうち，ベクトル 1 は，キャッシュメモリに接続されている．ベクトル 2 は，メインメモリに直結することで，ベクトル長を 2048 程度まで拡張できる．そして，CGRA は，ALU と，64KB のキャッシュメモリのサンドイッチ構造にすることで，ベクトル長を 16K 程度まで拡張できる．不規則なアクセスパターンをキャッシュメモリにカプセル化することで、メインメモリは規則的なアクセスパターンによる高速動作を維持できる．

図 1.2 に，典型的な実行モデルを示す．1 台の車が，1 つの CPU であると仮定する．青色のデータが入力画像，緑色のデータが重みとする．すべての CPU は，たとえ同じデータであっても，キャッシュ メモリに不足しているデータを取得しようとする．しかし，データがいつ到着するかを予測することはできない．GPGPU も多くのコアを搭載している．そして，車両は，出発前に可能な限り整列して待機する．出発と到着を統合することで，渋滞を緩和することができる． ただし，目的地が分散している場合，対処は困難である．統合できるかどうかは，プログラミングスキルに依存する．そして，右側が IMAX である．上部に，いくつかの CPU がある．数多くのキャッシュメモリは，自らデータを取得しに行くことはなく，CPU から提供されるデータを待つ．CPU は緑色の重量データを一度に送信でき，データ通信量と消費エネルギーを削減できる．

1.2 Independent network for memory and execution

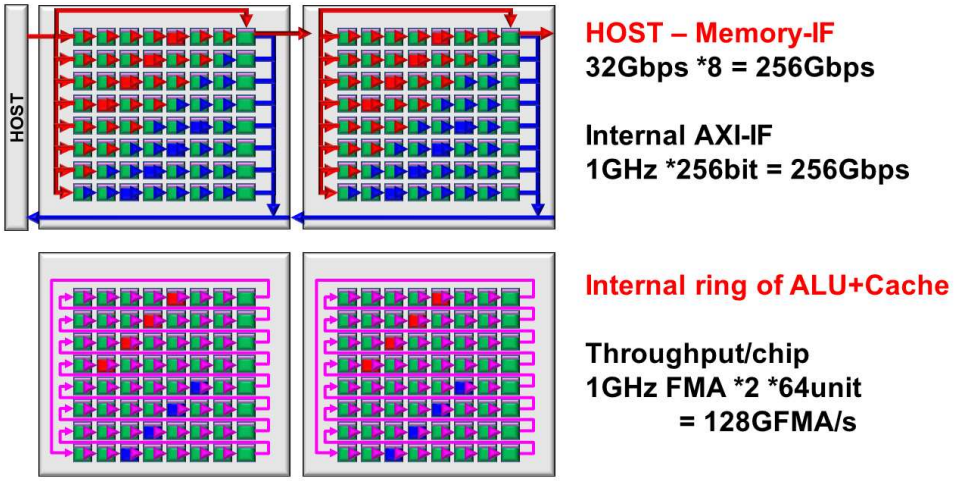

Figure.1.3: Execution network and Memory network

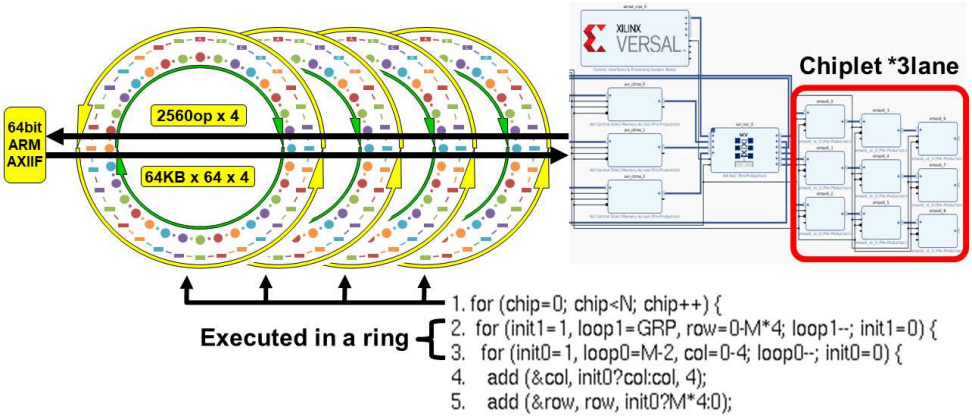

Figure.1.4: IMAX2 multichip structure

図 1.3 に示すように，IMAX2 には 2 種類のネットワークがある．8 個のユニットをグループ化し，メモリインターフェイスに並列接続することにより，レイテンシを短縮している．演算には，各チップ内で 64 個のユニットがリング状に接続され，ダイヤル錠のように，演算とローカルメモリの組合せを変更できる．リング構造はステンシル計算に役立つ．マッピングされた操作をスライドさせることで，ALU とキャッシュメモリのペアを変更でき，キャッシュメモリ内のデータの多くが再利用できる．また，図 1.4 に示すように，起動オーバーヘッドを削減するために，トリプルループを IMAX2 に一度にマッピングできる．最外ループは複数チップにマッピングされ，内側の 2 重ループは各チップにマッピングされる．図 1.5 は，トリプル

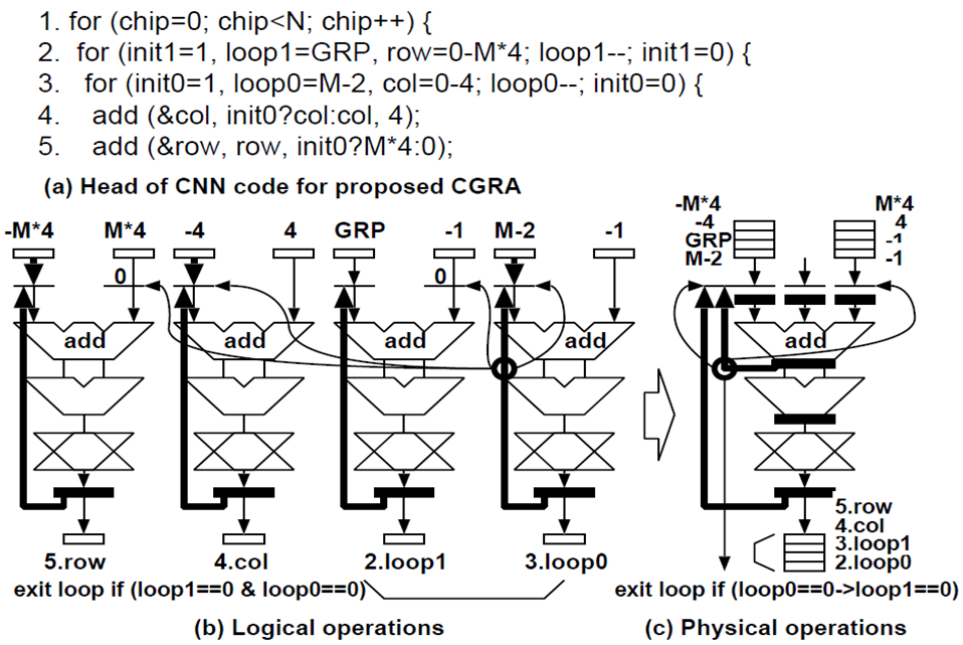

Figure.1.5: Burst execution of triple loops

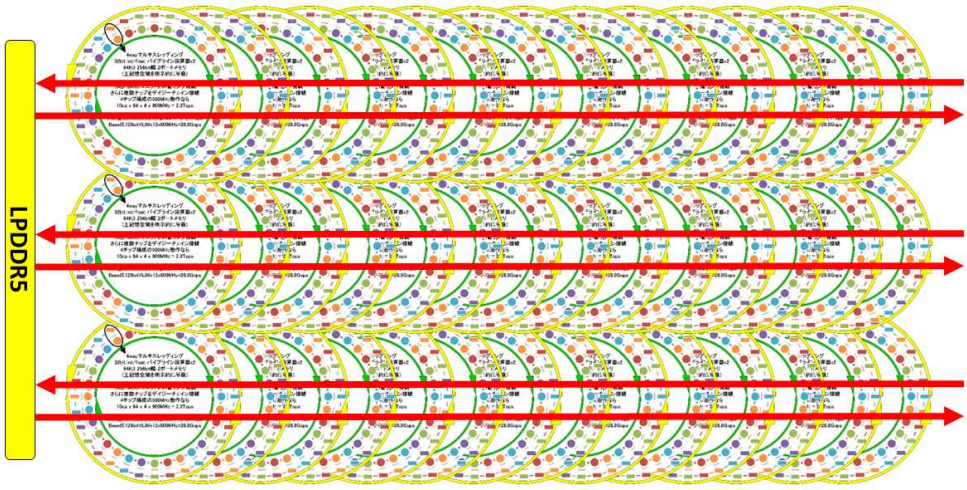

Figure.1.6: IMAX3 multilane structure

ループ制御を 4 つの論理ユニット (1 つの物理ユニット) が実行する仕組みである．そして，IMAX3 は，図 1.6 に示すように，IMAX2 を複数レーン接続したものである．

1.3 Multilevel pipelining

図 1.7 は，LPDDR5 に接続された複数レーンの IMAX2 と，各レーンのマイクロパイプライン動作を示している．マイクロパイプラインは，CGRA の基本モードである．各レーンには複数の IMAX2 チップをカスケード接続できる．このモードは，従来の逐次プログラムに適用でき，コンパイルも高速である．次に，Figure1.8 は，複数レーンの IMAX2 と，各レーンのミディアムパイプライン動作をを示している．ミディアムパイプラインは，各レーンにおいて，キャッシュメモリのダブルバッファリングにより実装されており，ソート，ハッシュ関数，および，FFT など，ステージ間の分離が必要な処理に対応することができる．Figure1.9 は，IMAX3 のマクロ パイプラインを示す．複数レーンは，LPDDR5 を介して 1 つの長大なパイプラインとして連結される．各レーンには，マイクロおよびミディアムパイプラインが含まれる．Figure1.10 は，IMAX3 の最終形態を示している．各レーンに複数の IMAX2 チップが搭載されている．図 1.11 のように，多数のレーンと CPU を投入して，多くの種類のカーネルを同時にマッピングすることがで

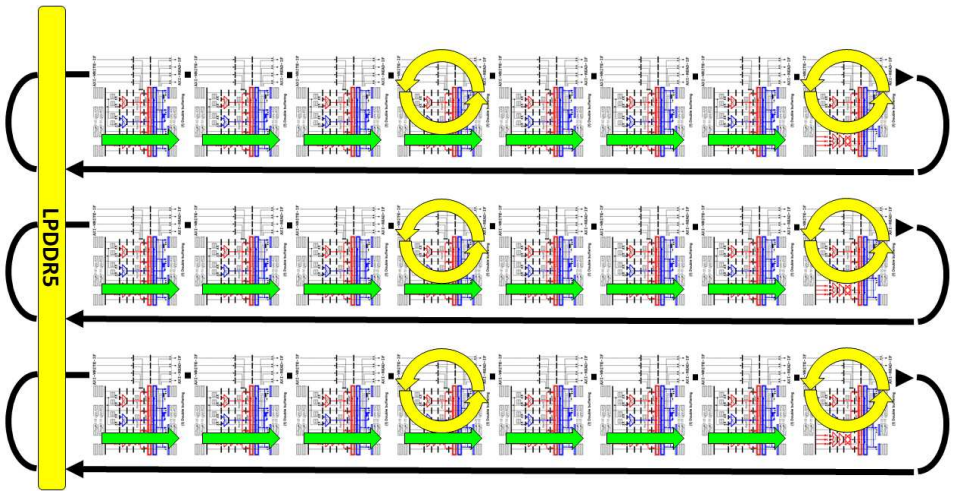

Figure.1.7: Micro pipelining

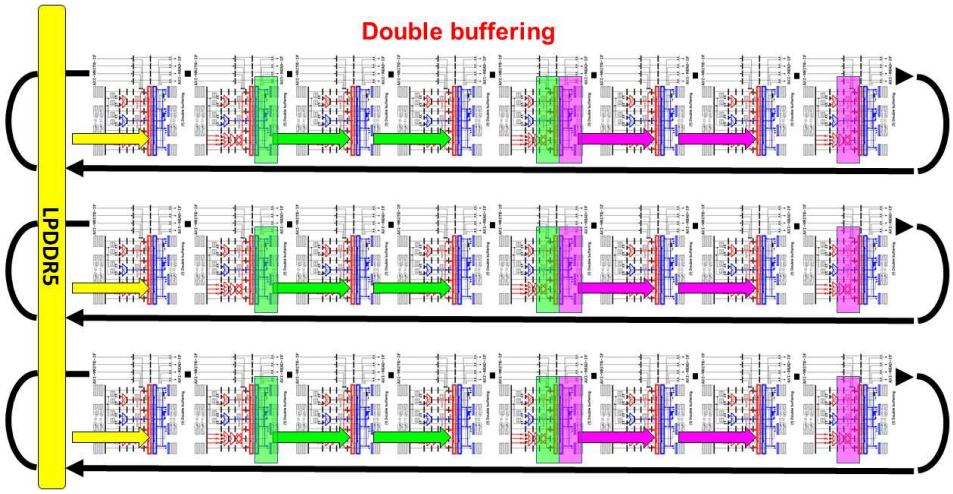

Figure.1.8: Medium pipelining

きる.

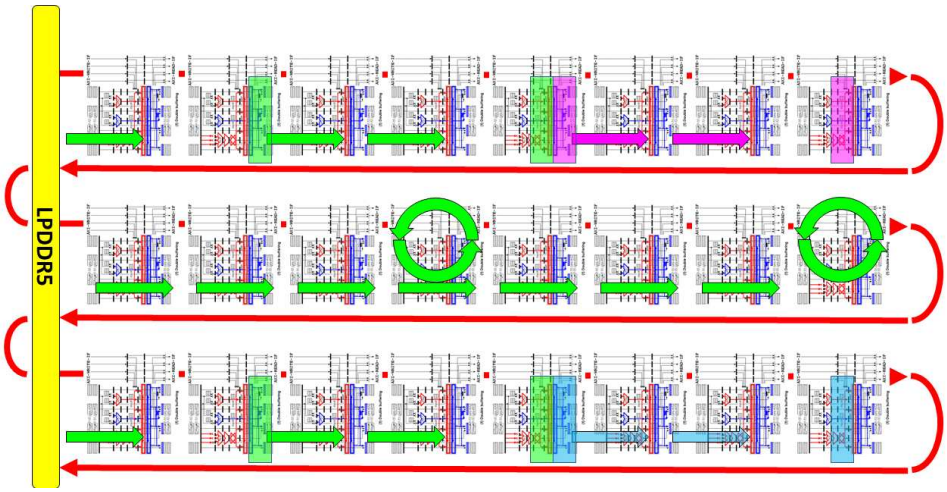

Figure.1.9: Macro pipelining

Figure.1.10: Multilevel pipelining

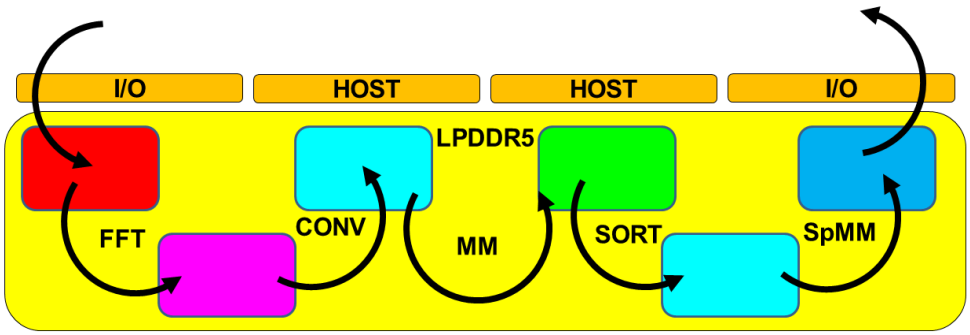

Figure.1.11: Mapping of application kernels

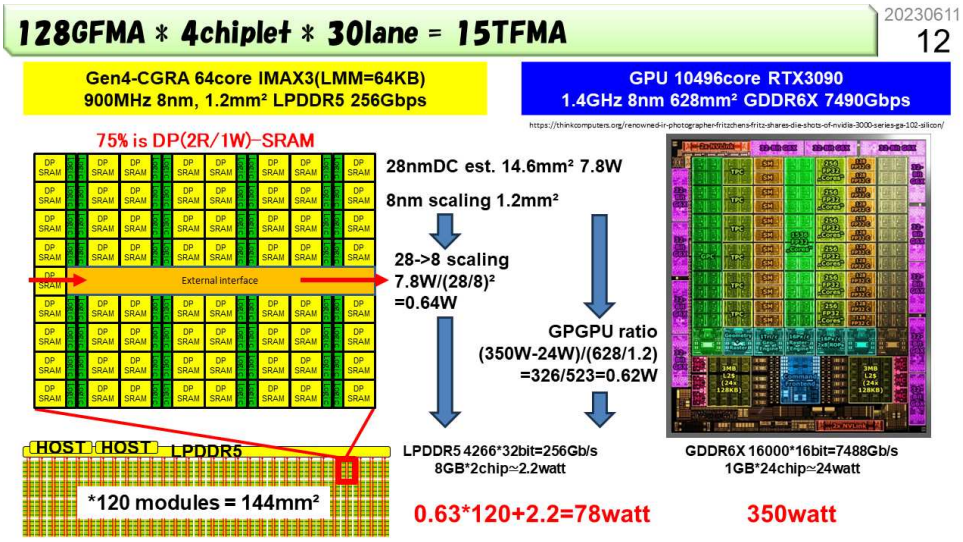

Figure.1.12: Area estimation

1.4 Prototype

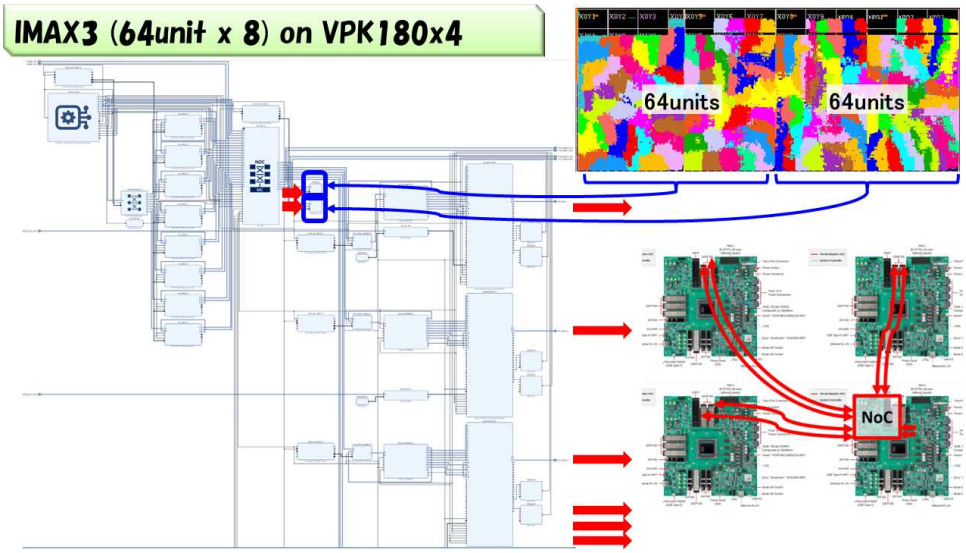

Figure.1.13: Prototype of IMAX3

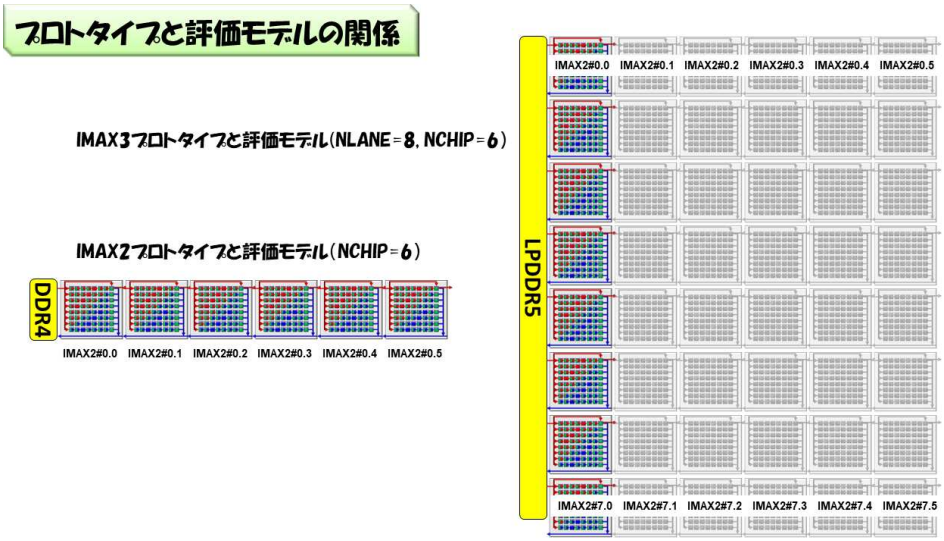

Figure.1.14: Evaluation models

図 1.12 に示すように，IMAX3 の 75%の面積はキャッシュメモリである．LPDDR5 の各ポートに，64 ユニット構成の IMAX2 を 4 基カスケード接続した場合，10240 オペレーションを一度にマッピングできる．LPDDR5 において 30 ポートが使用可能な場合，307,200 オペレーションをマッピングできる．8nm 技術で製造した場合，144mm 平方に，IMAX2 を 120 基搭載できる．Figure1.13 は，IMAX2 を IMAX3 にスケールアップする進行中のプロジェクトである．VPK180 には，64 ユニット構成の IMAX2 を 2 基搭載でき，NoC を介して合計 8 基の IMAX2 を接続できる．図 1.14 は，プロトタイプを用いた性能評価モデルである．実装されるのは，IMAX2#0.0，IMAX2#1.0，…，IMAX2#7.0 の 8 基である．各 IMAX2 は，IMAX2 アプリケーションプログラムでは，NLANE=8,NCHIP=1 の構成に対応する．一方，IMAX2 アプリケーションプログラムにおいて，NLANE=8,NCHIP=6 と記述して走行させることも可能である．この走行方法は，カスケード構成の IMAX2 を複数レーン装備する構成に対応する．ただし，実装されているのは先頭の IMAX2#*.0 のみであるため，カスケード接続される後続 IMAX の実行結果は 0 となり，また，カスケード接続に伴うオーバヘッドも測定されない．実行速度は，あくまで理想値である．

Chapter 2

IMAX3 Software

2.1 IMAX3 interface mapped on CPU memory space

IMAX3 は，大容量外部メモリに，IMAX2 を複数レーン接続した構成であり，制御のためのハードソフトインタフェースは，IMAX2 ごとの制御インタフェースの集合である．

2.2 Dataflow example

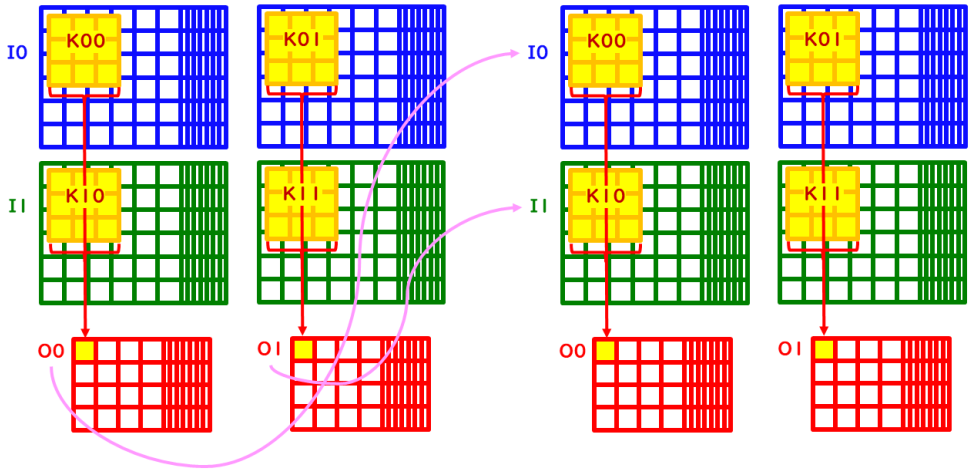

Figure.2.1: Tycal series of 4D-array convolution

図 2.1 に，典型的な畳み込み演算を示す．入力データ (I) の一部と，対応するカーネル (K) の乗算を行い，総和を求める．6x6 の 2 次元 I0 が，奥行方向に重なっているのは，6x6 の画像が 1 つのバッチに複数枚含まれることに対応する．I1 も同様の構造をしており，例えば，I0 が青の成分，I1 が緑の成分に対応する．入力データとカーネルの組が多数あり，縦方向の総和が出力 O0 となる．入力データが 2 次元であり，カーネル K を X 方向と Y 方向にずらして計算を繰り返すため，出力 O0 も 2 次元である．また，入力データ (I) はそのまま，カーネル (K) のみを差し替え，同様の計算をすると，別の出力 O1 が求まる．すなわち，入力データ (I)，カーネル (K)，出力データ (O) は，いずれも 4 次元配列である．さらに，多層畳み込み演算では，出力データ (O) を次の入力データ (I) として，同様の計算を繰り返す．

以上の畳み込み演算を IMAX3 により実行する方法は，いくつかある．図 2.2 は，2 次元畳み込みの基本形である．IMAX の各ユニットには，ホストのドライバが制御するキャッシュメモリと演算器が入っている．主記憶の青色入力データは，キャッシュメモリの青い部分にブロードキャストされ，主記憶の黄色のカーネルは，黄色の部分にブロードキャストされる．ユニット 0 番は，カーネルの左上に対応するデータを取り出し，ユニット 1 番に送る．ユニット 1 番は，乗算を行い，結果をユニット 2 番に送る．以上を繋ぎ，最後に，ユニット 10 番が 9 個の乗算結果の総和を赤色出力に足し込み，3x3 の畳み込み演算を 1 つ完了する．全て

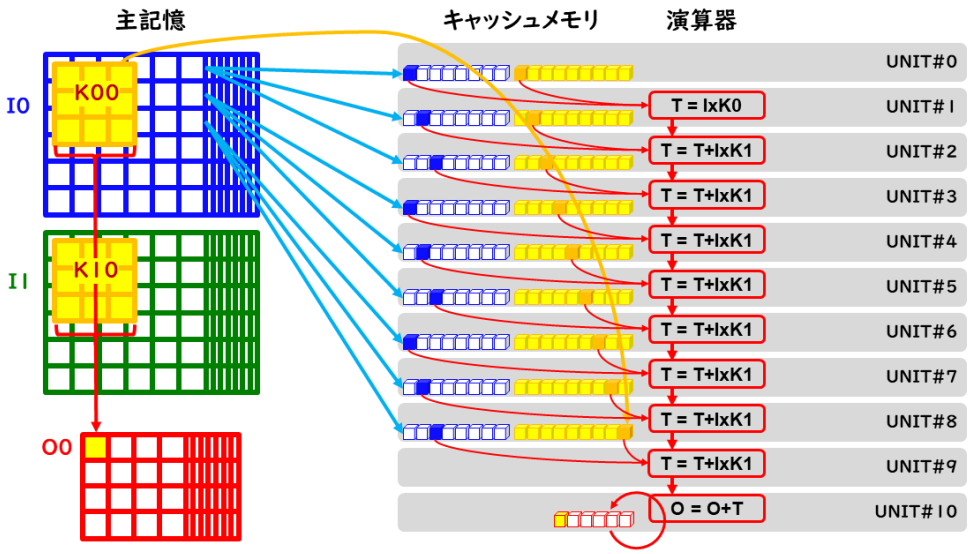

Figure.2.2: Typical mappting of convolution

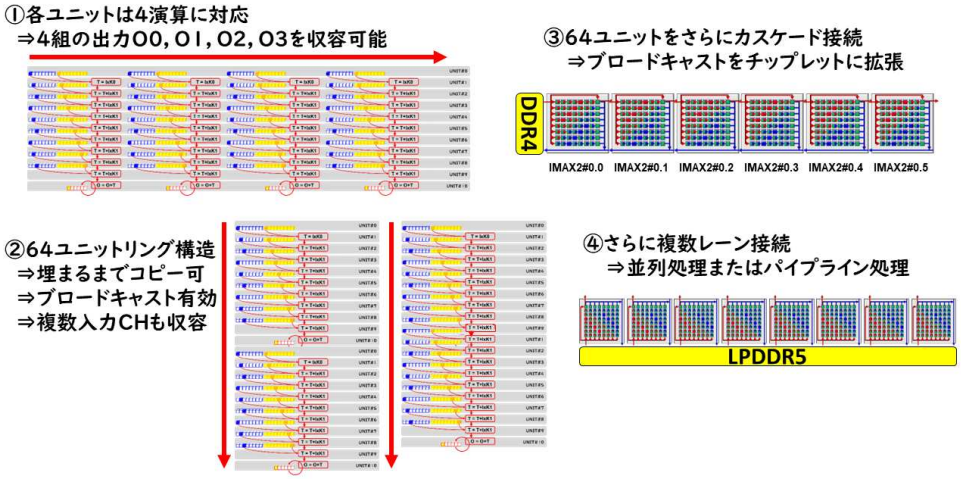

Figure.2.3: Several options to speedup convolution

のユニットが入力データを右にシフトしながら次々と計算し，出力データが連続して更新される．以上が，典型的な，CGRA の計算方法である．

IMAX の特徴は，以上の基本形を 4 次元配列の大きさに合わせて組み合わせる自由度の多さにある (図 2.3)．最適化の目標は，実行時間を短縮することであり，手段は，ブロードキャストと，キャッシュメモリの最大限再利用である．IMAX2 の 1 チップは，2 重ループを一度に連続実行でき，4 組の畳み込み演算を論理的な 4 列構造に写像できる．残るは，4 次元データのどの軸を 2 重ループに写像するかである．入力 I のどの軸を選ぶかを決めると，他は自動的に決まるため，まず，青色入力データ (I) の軸を選ぶ．候補は，図 2.4 に示す，X 軸，Y 軸，バッチ軸，チャネル軸の 4 つである．ただし，X 軸は，アドレスが連続するため，必ず選ぶべきである．同様に，チャネル軸は，64 ユニットを埋めるために利用すれば効率が良いため，残る軸は，Y 軸またはバッチ軸の 2 択となる．

図 2.5 に，2 つの写像方法を示す．X 軸と Y 軸を使う場合，黄色のデータは，連続アドレスであるため，並べ替えが不要である．X の長さと Y の長さを乗じた 1 枚のデータサイズが，ユニット内のキャッシュメモリに入る場合，最も効率が良い．例えば，キャッシュメモリが 16Kword 分あれば，128x128 まで収容できる．256x256 の場合も，この形のまま，Y 方向に 4 分割して実行すればよい．一方，多層畳み込みの後方では，2x2 等の小さな正方形になるため，IMAX の連続実行時間が短くなり，起動オーバヘッドが大きくなる．このような場合，Y 軸の代わりにバッチ軸を使う．並べ替えに伴うオーバヘッドは増加するが，バッチ数が 100 の場合，2x100 となり，データサイズを大きくでき，起動オーバヘッドを削減できる．3 次元畳み

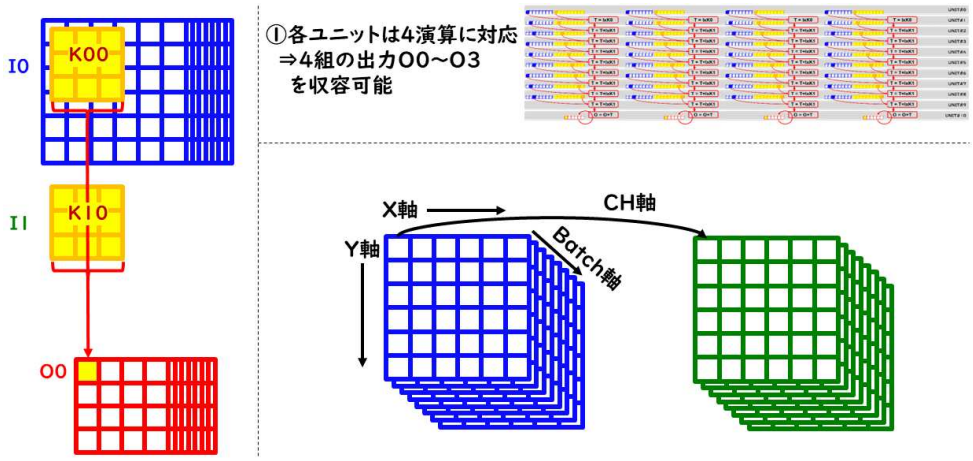

Figure.2.4: Selecting two axis from 4D array

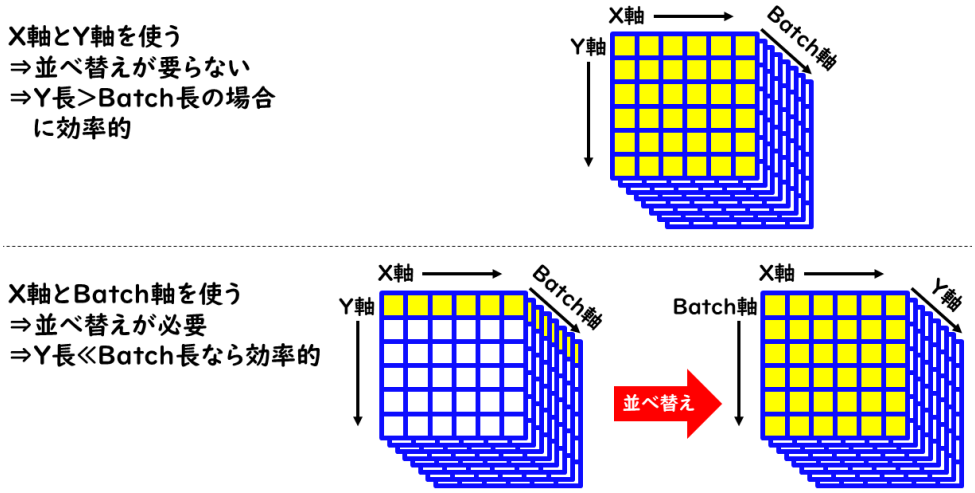

Figure.2.5: Two options to increase the length of the burst execution

込みの場合も，同様に，IMAX の連続動作時間を長くし，起動回数を減らせばよい．

2.3 Programming model of Macro pipelining

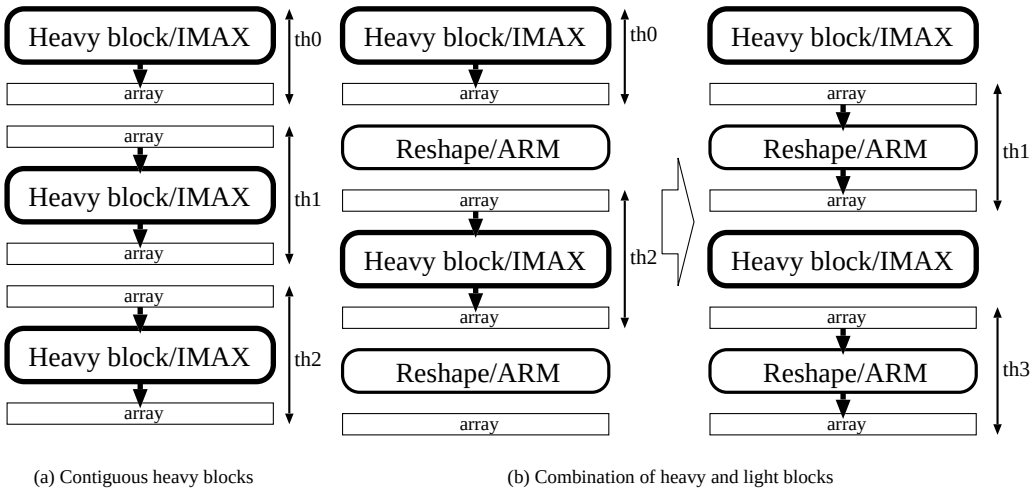

Figure.2.6: Programming model

図 2.6 は，マクロパイプライニングに関する，プログラミングモデルである．(a) は，処理時間の長い区間が 3 つあり，3 つのスレッドを起動して，各々を IMAX により高速化するマクロパイプライニングである．各スレッドの入力と出力が干渉しないよう，スレッド間のデータ構造にダブルバッファが必要となる．一方，(b) は，処理時間の長い区間を IMAX により実行するものの，処理間に，配列添字の入れ替え等，ホストによる短時間処理が介在するケースである．短時間処理をダブルバッファとして利用することにより，全てのスレッド間にダブルバッファを適用する場合に比べ，メモリ使用量を抑制できる．

2.4 Programming style of Macro pipelining

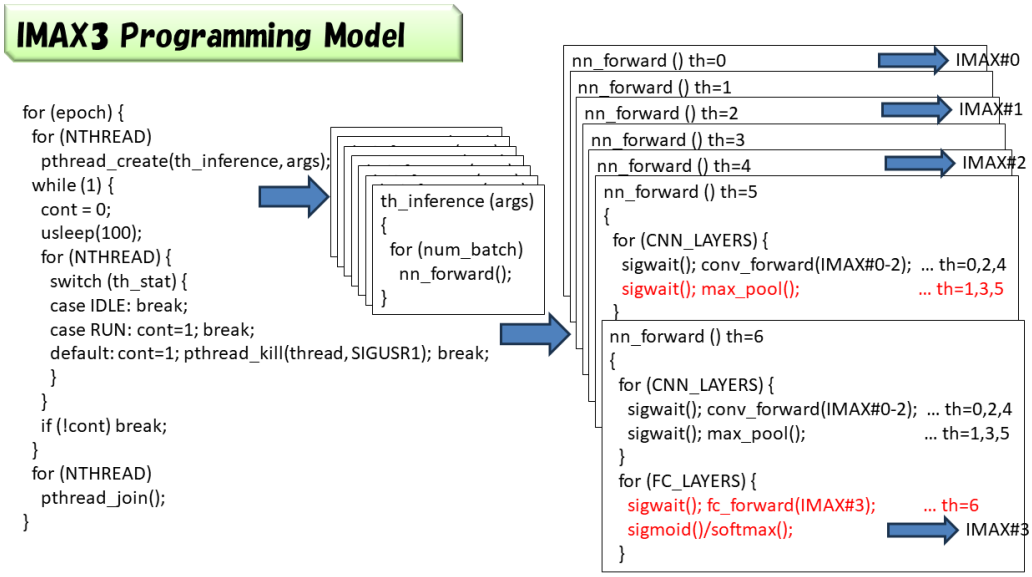

Figure.2.7: Programming stype

図 2.7 は，プログラミングスタイルである．マクロパイプライニングには，複数のスレッドを起動し，ループ構造を内包する関数 (th_inference) を同時に実行開始する．th_inference() は，多層 CNN をシミュレートする nn_forward() を繰り返し実行する．nn_forward() は，複数スレッドにより同時に実行されるものの，各スレッドは，担当箇所のみを実行し，前段層と後段層の処理完了との待ち合わせを行いつつ，パイプライン処理を行う．各層のうち，LANE を引数とする関数呼び出しが，IMAX2 を使用する．マクロパイプライニングには，HOST と IMAX2 が各々複数参加している，Busy loop ではなく，sigwait() を使うことにより，HOST のコア数が十分にない場合にも，スレッド間の待ち合わせが HOST の能力を超えない仕組みとしている．

Chapter 3

Examples

3.1 Image recognition (tsim)

MNIST

```
cent% make -f Makefile-cent.emax7nc all clean
cent% cd ../; tsim/tsim-cent.emax7nc -x -t -I0 -C1 -F1
```

```
acap% make -f Makefile-acap.emax7+dma all clean
acap% cd ../; tsim/tsim-acap.emax7+dma -x -t -I0 -C1 -F1
```

CIFAR10

```
cent% make -f Makefile-cent.emax7nc all clean
cent% cd ../; tsim/tsim-cent.emax7nc -x -t -I1 -C6 -F2
```

```
acap% make -f Makefile-acap.emax7+dma all clean
acap% cd ../; tsim/tsim-acap.emax7+dma -x -t -I1 -C6 -F2
```

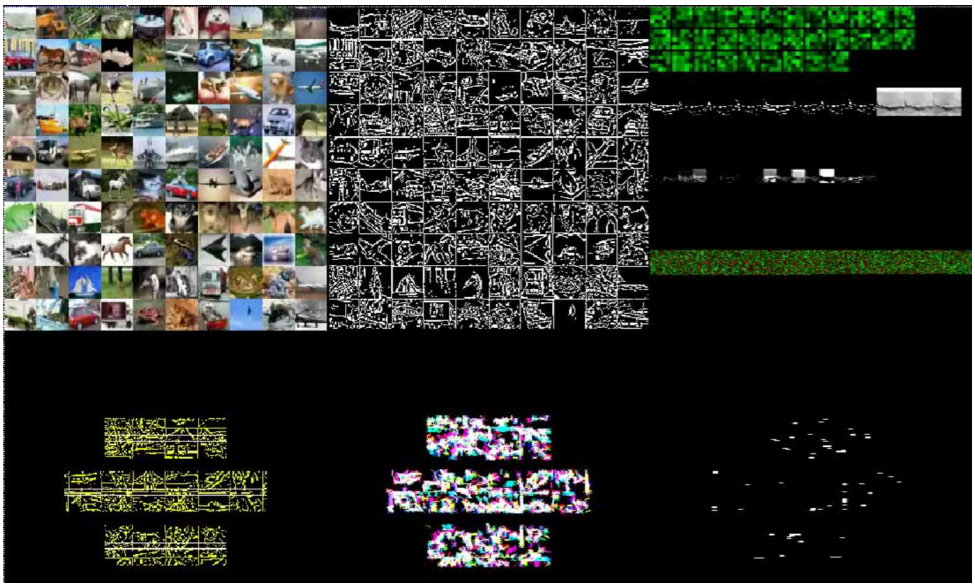

Figure.3.1: Image recognition (training + inference)

3.1.1 Header

IMAX3は，複数のIMAX2レーンを制御するために，ARMのマルチスレッディングを利用する．NTHREADは，複数IMAX2レーンの起動に際し，ARMにおいて起動しておくスレッド数である．EMAX_LANEは，同時に起動するIMAX2レーンを制御するための，制御変数セットの上限である．また，NLANEは，実機において検出されたIMAX2のレーン数である．EMAX_LANEを超えて，実機において検出されたIMAX2は，利用されない．すなわち，NLANEにセットされる値は，必ず，EMAX_LANE以下である．なお，NTHREADは，NLANE以上でなければならない．

```
#define MAX_NTHREAD 16
volatile struct th_inference_args {
  int      thid;
  int      stat;     /* 0:idle, 1:run, 2:wait (enq/deq, DMA, EXEC) */
  sigset_t sigset;   /* sys/_sigset.h 2B/4B */
  int      deq;
  int      enq;
  float4D *slice;
  CNNet   *net;
  int      batch_size;
  int      nchan;
  int      insize;
} th_inference_args[MAX_NTHREAD];
volatile int th_inference_retv[MAX_NTHREAD];
pthread_t    th_inference_t[MAX_NTHREAD];
void         th_inference(struct th_inference_args *);
```

3.1.2 IMAX3 thread driver

複数のIMAX2を起動して，マクロパイプラインを構成する際には，IMAX2カーネルを含む関数ラッパー（以下の例ではth_inference）を用意し，ARMのpthread_createを用いて，NTHREAD個のスレッドを起動する．スレッドの引数は，thread番号，パイプライン同期用のenq/deqを含む．

```
int THREAD;
for (THREAD=0; THREAD<NTHREAD; THREAD++) {
  th_inference_args[THREAD].thid       = THREAD;
  th_inference_args[THREAD].stat       = 1; /* run */
  sigemptyset(&th_inference_args[THREAD].sigset);
  sigaddset(&th_inference_args[THREAD].sigset, SIGUSR1);
  pthread_sigmask(SIG_BLOCK, &th_inference_args[THREAD].sigset, NULL);
  th_inference_args[THREAD].enq        = 0;
  th_inference_args[THREAD].deq        = 0;
  th_inference_args[THREAD].slice      = &slice;
  th_inference_args[THREAD].net        = net;
  th_inference_args[THREAD].batch_size = batch_size;
  th_inference_args[THREAD].nchan      = nchan;
  th_inference_args[THREAD].insize     = insize;
}
if (NTHREAD > 1) {
  for (THREAD=0; THREAD<NTHREAD; THREAD++)
    pthread_create(&th_inference_t[THREAD], NULL, (void*)th_inference, &th_inference_args[THREAD]); /* 0-(NTHREAD-1) */
  while (1) {
    int cont = 0;
    usleep(4000); /* 4msec 2024/01/24 Nakashima */
    for (THREAD=0; THREAD<NTHREAD; THREAD++) {
      switch (th_inference_args[THREAD].stat) {
      case 0:            break;/* idle */
      case 1:  cont = 1; break;/* run */
      default: cont = 1; pthread_kill(th_inference_t[THREAD], SIGUSR1); break; /* wait */
      }
    }
    if (!cont) break;
  }
  for (THREAD=0; THREAD<NTHREAD; THREAD++)
    pthread_join(th_inference_t[THREAD], NULL);
}
else {
  for (THREAD=0; THREAD<NTHREAD; THREAD++)
    th_inference(&th_inference_args[THREAD]);
}
```

3.1.3 IMAX3 thread wrapper

ラッパーは，複数同時に起動される．内部で複数スレッドがパイプライン動作するよう，引数で与えられるスレッド番号に基づき，担当箇所を実行するよう記述する．

```
/* IMAX3 MACROPIPELIING for EVALUATION */
void th_inference(struct th_inference_args *args)
{
  int THREAD     = args->thid;
  float4D *slice = args->slice;
  CNNet *net     = args->net;
  int batch_size = args->batch_size;
  int nchan      = args->nchan;
  int insize     = args->insize;
  int j, k;
  /************************************/
  /* TARGET of MACRO-PIPELINING/IMAX3 */
  /************************************/
  slice->nstrides = batch_size;
  slice->nchannel = xtest.nchannel;
  slice->kstrides = xtest.kstrides;
  slice->stride_size = xtest.stride_size;
  for (j=0; j+batch_size<=xtest.nstrides; j+=batch_size) {
    if (THREAD == 0) {
      slice->data = &(xtest.data[j*xtest.stride_size*xtest.kstrides*xtest.nchannel]);
      if (cnn_mode) {
        if (enable_x11) {
          F4i2Ipl(batch_size, nchan, insize, insize, I, slice); /* 100batch x 28x28 x 1chan */
          copy_I_to_BGR(D, batch_size, insize, insize, I);
          BGR_to_X(0, D);
        }
        // copy data to input layer
        copy4D(&(net->ninput), slice);
        if (attn_mode) {
          attention(&(net->ninput), &(net->work), &(net->attention)); /* batch,RGB,H,W */
          if (enable_x11) {
            F4i2Ipl(batch_size, nchan, insize, insize, I, &(net->work)); /* 100batch x 28x28 x 1chan */
            copy_I_to_BGR(D, batch_size, insize, insize, I);
            BGR_to_X(3, D);
            F4i2Ipl(batch_size, nchan, insize, insize, I, &(net->ninput)); /* 100batch x 28x28 x 1chan */
            copy_I_to_BGR(D, batch_size, insize, insize, I);
            BGR_to_X(1, D);
          }
        }
      }
      if (eye_mode) {
        F4i2Ipl(batch_size, nchan, insize, insize, I, slice); /* 100batch x 28x28 x 1chan */
        copy_I_to_BGR(R, batch_size, insize, insize, I); /* R <- I */
        copy_I_to_BGR(L, batch_size, insize, insize, I); /* L <- I */
        if (enable_x11)
          BGR_to_X(0, R);
        /* pre-processing by eye-model */
        eyemodel(enable_x11, slit_type); /* L+R -> Sl+Sr */
        /* import Sr to hidden_layer */
        Ipl2F4h(10, WD, HT, Sr, Cr, R, &net->nhidden[0]); /* 100batch x 24x24 x 9chan -> hidden */
      }
    }
    nn_forward(/*MACROPIPE*/1, THREAD, net, c[input_type], f[input_type], &pred, spike_mode);
    if ((NTHREAD==1 || eye_mode)
      ||(CNN_DEPTH==1 && THREAD==2)
      ||(CNN_DEPTH==3 && THREAD==6)
      ||(CNN_DEPTH==4 && THREAD==8)
      ||(CNN_DEPTH==6 && THREAD==12)) {

      for (k=0;k<batch_size;k++) {
        float *A = &(pred.data[k*pred.stride_size]);
        nerr += (MaxIndex(A, pred.stride_size) != ytest[j+k]);
      }
      if (enable_x11) {
        clear_BGR(D);
        copy_H_to_BGR(D+WD*(HT*1/4), &net->nhidden[0]);
        copy_H_to_BGR(D+WD*(HT*2/4), &net->nhidden[CNN_DEPTH-1]);
        BGR_to_X(2, D);
        while (x11_checkevent());
      }
    }
  }
  /*********************************/
  /* END of MACRO-PIPELINING/IMAX3 */
  /*********************************/
  args->stat = 0; /* idle */
}
```

3.1.4 IMAX3 region

以下の nn_forward は，インファレンスを行う最上位関数である．現在は，手作業により，引数 THREAD と，使用する IMAX2 レーン番号を関連付けている．また，enq/deq を用いて，マクロパイプラインを同期している．将来，自動化ツールが完成すれば，手作業は不要となる．

```
void nn_forward(int MACROPIPE, int THREAD, CNNet *net, struct c *c, struct f *f, float2D *oubatch, int spike_mode)
{ /* NTHREAD 1:MACRO_PIPE_OFF 2-:MACRO_PIPE_ON */
  /*                            EMAX7:NTHREAD=8 other:NTHREAD=1 */
  /*                             ZYNQ:NLANE=X                   */
  /*                             othr:NLANE=4                   */
  /* train:     nn_forward(0, 1)                                */
  /* inference: nn_forward(1, T)                                */
  /* camera:    nn_forward(0, 1)                                */
  /* th#>0 の場合, 前段 enq==deq なら待機 最終 th#未満の場合, 自段 enq!=deq なら待機 */
  /* 待機状態になければ, 前段 deq=1-deq に更新                                    */
  /* 自段 enq で自身の dbuf 選択                                                  */
  /* 最後に,            自段 enq=1-enq に更新                                     */
  /* DBUF の場合                                                                  */
  /* th#0      ****0* ****1* ****2*  CNN0:ninput->nhidden[0]                     */
  /*      enq 0     11     00     11                                              */
  /*      deq 0     01     10     01                                              */
  /* th#1     -------*0*    *1*    *2*  nhidden[0]->npool[0]                     */
  /*      enq 0        11     00     11                                           */
  /*      deq 0        01     10     01                                           */
  /* th#2     ----------****0* ****1* ****2*  CNN1:npool[0]->nhidden[1]          */
  /*      enq 0              11     00     11                                    */
  /*      deq 0              01     10     01                                    */
  /* th#3     ----------------*0*    *1*    *2*  nhidden[1]->npool[1]            */

  /* NHIDDEN/NPOOL を DBUF として使う場合                                          */
  /* th#0      ****0*  ****1*  ****2*  CNN0:ninput->nhidden[0]                   */
  /*      enq 0     1 1     0 0     1 1                                          */
  /*      deq 0     0 1     1 0     0 1                                          */
  /* th#1     ------*0*     *1*     *2*  nhidden[0]->npool[0]                    */
  /*      enq 0       1     1 0     0 1                                          */
  /*      deq 0       0     1 1     0 0     1                                    */
  /* th#2     ----------****0*  ****1*  ****2*  CNN1:npool[0]->nhidden[1]        */
  /*      enq 0             1 1     0 0     1 1                                  */
  /*      deq 0             0 1     1 0     0 1                                  */
  /* th#3     --------------*0*     *1*     *2*  nhidden[1]->npool[1]            */

  int batch_size = net->ninput.nstrides;
  int i, j, k, l;
  float *temp;

  if (cnn_mode && !(CNN_DEPTH==1 && FC_DEPTH==1) && !(CNN_DEPTH==3 && FC_DEPTH==1)
              && !(CNN_DEPTH==4 && FC_DEPTH==1) && !(CNN_DEPTH==6 && FC_DEPTH==2)) { /* IGNORE MACRO_PIPE in cnn_mode */
    if (THREAD > 0) exit(0);
  }
  if (eye_mode) { /* IGNORE MACRO_PIPE in eye_mode */
    if (THREAD > 0) exit(0);
  }
  if (spike_mode) { /* IGNORE MACRO_PIPE in spike_mode */
    if (THREAD > 0) exit(0);
  }
  if (spike_mode) {
    /* SMAX1 assumes -V* -C1 */
    // add bias broadcast<2>(hbias, hidden.shape);
    temp = net->nhidden[0].data;
    for (i=0;i<net->nhidden[0].nstrides;i++) {
      for (j=0;j<net->nhidden[0].nchannel;j++) {
        for (k=0;k<net->nhidden[0].kstrides*net->nhidden[0].stride_size;k++,temp++)
          *temp += net->hbias[0].data[j];
      }
    }
    relu4(&(net->nhidden[0]));                                          /* V1->c.nhidden[0].data */
    max_pooling(&(net->npool[0]), &(net->nhidden[0]), c[0].psize, c[0].psize); /*   ->c.npool[0].data */
    flat4Dto2D(&(net->nflat[0]), &(net->npool[0]));                     /*   ->f.nflat[0].data */
    smax_trial(0, net, c, f); /* IGNORE MACRO_PIPE */
  }
```

```
  else {
    int LANE = 0;
    for (l=0; l<CNN_DEPTH; l++) {
      /*************************************************************************/
      if (!MACROPIPE || NTHREAD==1 || eye_mode) {}
      else if (THREAD == l*2) {
        if (THREAD>0) while (th_inference_args[THREAD-1].enq==th_inference_args[THREAD-1].deq) { inference_sigwait; }
                      while (th_inference_args[THREAD  ].enq!=th_inference_args[THREAD  ].deq) { inference_sigwait; }
        //th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq; /* DBUF の場合 */
#if defined(EMAX7)
        if (l >= NLANE) {
          printf("nn_forward_CNN: LANE(%d) >= NLANE(%d)\n", l, NLANE);
          exit(1);
        }
        LANE = l;
        emax7[LANE].sigwait = 1; /* ON */
        emax7[LANE].sigstat = &th_inference_args[THREAD].stat;
        emax7[LANE].sigset  = &th_inference_args[THREAD].sigset;
#endif
      }
      else
        goto end_of_cnn; /* skip other task */
      /*************************************************************************/
      if (l>0 || cnn_mode) {
        // first layer, conv, use stride=2
        /**********************ninput*** V CNN    */
#ifndef CUDA
/*★ IMAX3 ★*/conv_forward(THREAD, LANE, l==0?&(net->ninput):&(net->npool[l-1]), &(net->Ki2h[l]),
                          &(net->nhidden[l]), c[l].ksize, &(net->tmp_col[l]), &(net->tmp_dst[l]));
#else
        conv_forward_cuda(l==0?&(net->ninput):&(net->npool[l-1]), &(net->Ki2h[l]),
                          &(net->nhidden[l]), c[l].ksize, &(net->tmp_col[l]), &(net->tmp_dst[l]));
#endif
      }

      // add bias broadcast<2>(hbias, hidden.shape);
      temp = net->nhidden[l].data;
      for (i=0;i<net->nhidden[l].nstrides;i++) {
        for (j=0;j<net->nhidden[l].nchannel;j++) {
          for (k=0;k<net->nhidden[l].kstrides*net->nhidden[l].stride_size;k++,temp++)
            *temp += net->hbias[l].data[j];
        }
      }
      // Activation, relu, backup activation in nhidden
      // nhidden = F<relu>(nhidden);
      relu4(&(net->nhidden[l]));
      copy4D(&(net->nhiddenbak[l]), &(net->nhidden[l]));

      /*************************************************************************/
      if (!MACROPIPE || NTHREAD==1 || eye_mode) {
      }
      else if (THREAD == l*2) {
        if (THREAD>0) th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq;
                      th_inference_args[THREAD  ].enq = 1-th_inference_args[THREAD  ].enq;
      }
      /*************************************************************************/
    end_of_cnn:
```

```
      /*************************************************************************/
      if (!MACROPIPE || NTHREAD==1 || eye_mode) {
      }
      else if (THREAD == l*2+1) {
        while (th_inference_args[THREAD-1].enq==th_inference_args[THREAD-1].deq) { inference_sigwait; }
        while (th_inference_args[THREAD  ].enq!=th_inference_args[THREAD  ].deq) { inference_sigwait; }
        //th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq; /* DBUF の場合 */
      }
      else
        goto end_of_maxpool; /* skip other task */
      /*************************************************************************/

      // max pooling /*後段 nhidden が空いたら開始*/
      max_pooling(&(net->npool[l]), &(net->nhidden[l]), c[l].psize, c[l].psize);
      copy4D(&(net->npoolbak[l]), &(net->npool[l]));

      /*************************************************************************/
      if (!MACROPIPE || NTHREAD==1 || eye_mode) {
      }
      else if (THREAD == l*2+1) {
        th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq; /* NHIDDEN/NPOOL を使った DBUF の場合 */
        th_inference_args[THREAD  ].enq = 1-th_inference_args[THREAD  ].enq;
      }
      /*************************************************************************/
    end_of_maxpool:
      continue;
    }
```

```
    /*************************************************************************/
    if (!MACROPIPE || NTHREAD==1 || eye_mode)
      LANE = 0;
    else if (THREAD == CNN_DEPTH*2) {
      while (th_inference_args[THREAD-1].enq==th_inference_args[THREAD-1].deq) { inference_sigwait; }
      //th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq; /* DBUF の場合 */
#if defined(EMAX7)
      if (CNN_DEPTH >= NLANE) {
        printf("nn_forward_FC: CNN_DEPTH(%d) >= NLANE(%d)\n", CNN_DEPTH, NLANE);
        exit(1);
      }
      LANE = CNN_DEPTH;
      emax7[LANE].sigwait = 1; /* ON */
      emax7[LANE].sigstat = &th_inference_args[THREAD].stat;
      emax7[LANE].sigset  = &th_inference_args[THREAD].sigset;
#endif
    }
    else
      goto end_of_fc; /* skip other task */
    /*************************************************************************/
    for (l=0; l<FC_DEPTH; l++) {
      if (l==0)                                                /************************npool**** A CNN    */
        flat4Dto2D(&(net->nflat[0]), &(net->npool[CNN_DEPTH-1]));/*************************|****** BOUNDARY */
      else                                                     /************************nflat**** V FC     */
        copy2D(&(net->nflat[l]), &(net->nout[l-1]));

      // second layer full-connection
/*★ IMAX3 ★*/multiply_float2D(THREAD, LANE, &(net->nout[l]), &(net->nflat[l]), 0, &(net->Wh2o[l]), 0);
      repmat_add(&(net->nout[l]), &(net->obias[l]), batch_size);

      if (l < FC_DEPTH-1) {
        // activation, sigmloid, backup activation in fhidden
#if 1
        sigmoid(&(net->nout[l]));
#else
        relu2(&(net->nout[l]));
#endif
        copy2D(&(net->noutbak[l]), &(net->nout[l]));
      }
      else { /* l == FC_DEPTH-1 */
        // softmax calculation
        softmax2D(&(net->nout[FC_DEPTH-1]), &(net->nout[FC_DEPTH-1]));
      }
    }
    /*************************************************************************/
    if (!MACROPIPE || NTHREAD==1 || eye_mode)   {
    }
    else if (THREAD == CNN_DEPTH*2)
      th_inference_args[THREAD-1].deq = 1-th_inference_args[THREAD-1].deq; /* NHIDDEN/NPOOL を使った DBUF の場合 */
    /*************************************************************************/
    end_of_fc:;
  }

  if ((!MACROPIPE || NTHREAD==1 || eye_mode)
    ||(CNN_DEPTH==1 && THREAD==2)
    ||(CNN_DEPTH==3 && THREAD==6)
    ||(CNN_DEPTH==4 && THREAD==8)
    ||(CNN_DEPTH==6 && THREAD==12)
      ) {
    // copy result out
    copy2D( oubatch, &(net->nout[FC_DEPTH-1]));
  }
}
```

Chapter 4

Appendix

4.1 References

本章では，関連仕様書・規格，参考文献，関連ソースプログラム，および，ツールチェインを列挙する．

- EMAX5 基本特許 proj-arm64/doc/pat35.tgz
- IMAX 基本特許 proj-arm64/doc/pat36.tgz
- ARMv8 アーキテクチャ仕様書 ..... proj-arm64/doc/arm/DDI0487A_f_armv8_arm.pdf
- ARM Cortex-A53 MPCore Processor Technical Reference Manual
  ..... proj-arm64/doc/arm/ARM-CORTEX-A53_R0P4.pdf
- ZYNQ Ultrascale+ SoC Technical Reference Manual
  ..... proj-board/zcu102/doc/ug1085-zynq-ultrascale-trm.pdf
- AMBA AXI4 and ACE Protocol Specification ..... proj-arm64/doc/arm/AXI4_specification.pdf
- IMAX3 仕様書 proj-arm64/doc/emax7/emax7j.pdf
- IMAX3 C 言語ディレクティブ変換 proj-arm64/src/conv-c2d/conv-c2d
- IMAX2 仕様書 proj-arm64/doc/emax6/emax6j.pdf
- IMAX2 C 言語ディレクティブ変換 proj-arm64/src/conv-c2c/conv-c2c
- IMAX2 シミュレータ proj-arm64/src/csim/csim
- プログラム例（3x3-2D 畳み込み） ..... proj-arm64/sample/mm_cnn_if/cnn+rmm.c
- プログラム例（3x3-3D 畳み込み） ..... proj-arm64/sample/mm_cnn_if/cnn3d+rmm.c
- プログラム例（行列積） ..... proj-arm64/sample/mm_cnn_if/mm+rmm.c
- プログラム例（逆行列） ..... proj-arm64/sample/mm_cnn_if/inv+rmm.c
- プログラム例（Lightfield レンダリング） ..... proj-arm64/sample/mm_cnn_if/gather+rmm.c
- プログラム例（Lightfield 距離画像生成） ..... proj-arm64/sample/mm_cnn_if/gdepth+rmm.c
- プログラム例（画像認識+Stochastic 計算） proj-arm64/sample/tsim/imax.c