

EMAX8/amd64 (IMAX4) Architecture Handbook  
– In-Memory Accelerator eXtension –

Nara Institute of Science and Technology  
Computing Architecture Laboratory  
Accelerator Group

# Table of contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>IMAX4 Hardware</b>                                 | <b>4</b>  |
| <b>2</b> | <b>IMAX4 Software</b>                                 | <b>6</b>  |
| 2.1      | IMAX4 interface mapped on CPU memory space . . . . .  | 6         |
| <b>3</b> | <b>Examples</b>                                       | <b>9</b>  |
| 3.1      | Image recognition training/inference (tsim) . . . . . | 9         |
| 3.1.1    | Header . . . . .                                      | 10        |
| 3.1.2    | IMAX4 thread driver . . . . .                         | 10        |
| 3.1.3    | IMAX4 thread wrapper . . . . .                        | 11        |
| 3.1.4    | IMAX4 region . . . . .                                | 12        |
| 3.2      | LLM GPT (vsim) . . . . .                              | 15        |
| 3.3      | LLM GPT (llama) . . . . .                             | 16        |
| <b>4</b> | <b>Appendix</b>                                       | <b>18</b> |
| 4.1      | Prototype systems . . . . .                           | 18        |
| 4.2      | References . . . . .                                  | 18        |

# List of Figures

|     |  |    |
|-----|--|----|
| 1.1 | Prototype of IMAX3 (VPK120+VPK180x4) . . . . .                       | 5  |
| 2.1 | Physical memory space (IMAX3 compatible model: DMAbuf=4GB) . . . . . | 6  |
| 2.2 | Physical memory space (IMAX4 native model: DMAbuf=64GB) . . . . .    | 7  |
| 3.1 | Image recognition (training + inference) . . . . .                   | 9  |
| 3.2 | Llama . . . . .  | 16 |
| 4.1 | Prototype systems . . . . .  | 18 |

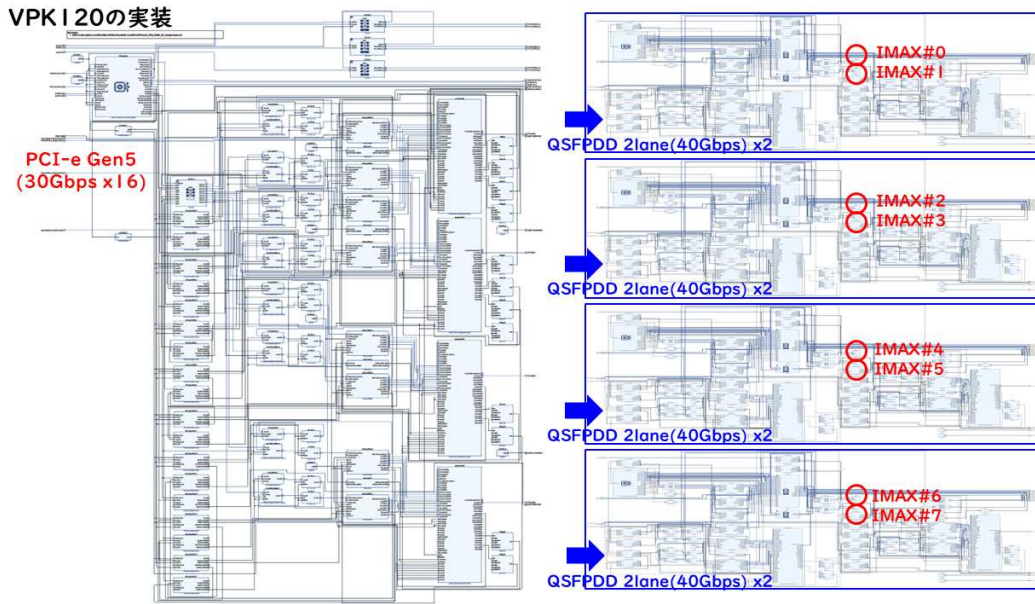
# List of Tables

# Chapter 1

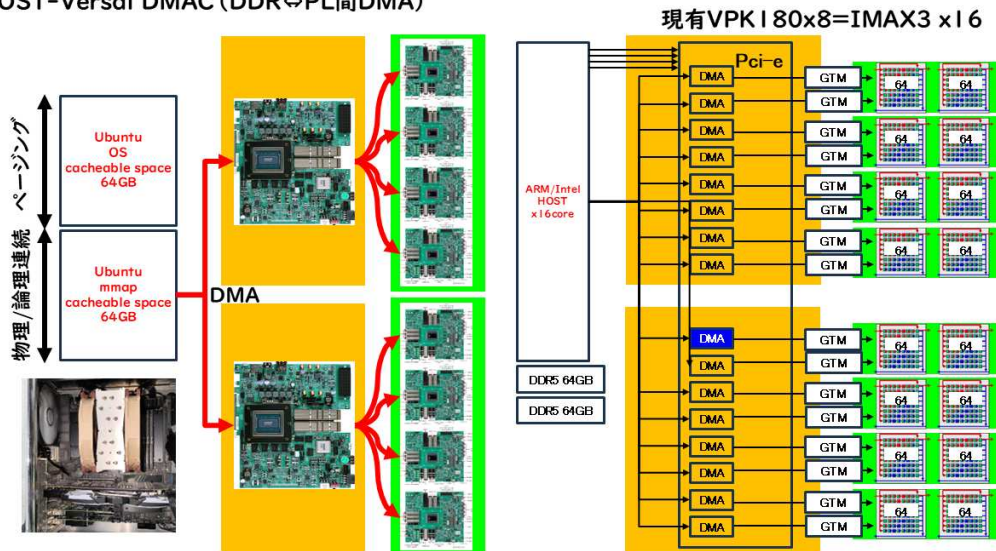
## IMAX4 Hardware

IMAX is a CGLA (Coarse Grained Reconfigurable Array), and like systolic arrays, data flows in one direction. However, because it has a ring structure rather than a two-dimensional structure, there are no positional constraints on instruction mapping. This gives it the advantage of fast compilation. There is a large-capacity cache memory next to the arithmetic unit, and broadcast, scatter, and gather are also possible. In this way, IMAX combines the predictability of performance of long vectors, the versatility of CGRAs, the high-speed compilation of systolic arrays, and the high efficiency of near-memory computing.

Figure1.1 shows the configuration of the IMAX4 prototype, in which four VPK180s are connected to a VPK120 connected to PCI-e. This will be used to evaluate the performance of servers.



1. GTM(4lane)x8組(QSPFDDx8)でIMAXx16
2. HOST-Versal DMAC (DDR⇄PL間DMA)



**現状**

- IMAX数を増やしGPGPUに対抗するにはスケラビリティが重要
- IMAX増設にはARMコアが少ない
- IMAX増設に対しDDRバンド幅小
- Cbuf(OS用),Ibuf(DMA用)とも cacheableだが小さい

**GPGPU**



**PCI-eの場合**

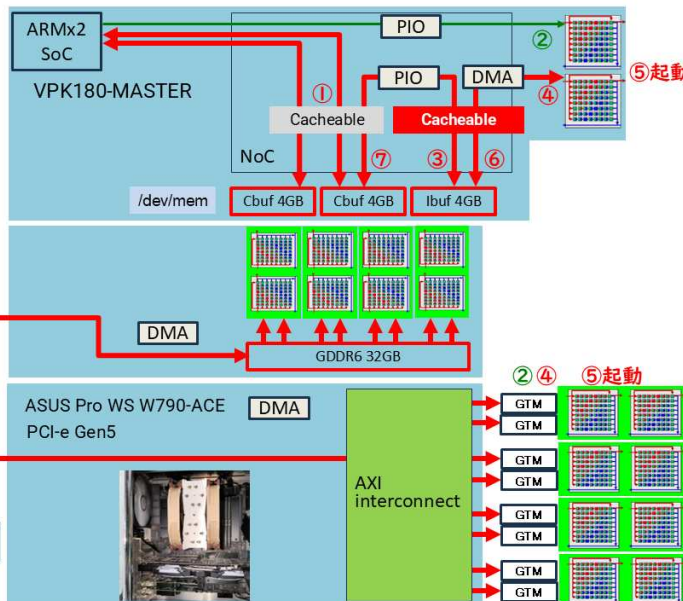
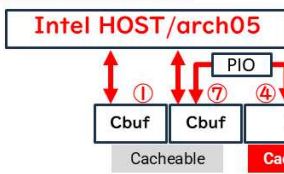


Figure.1.1: Prototype of IMAX3 (VPK120+VPK180x4)

# Chapter 2

## IMAX4 Software

### 2.1 IMAX4 interface mapped on CPU memory space

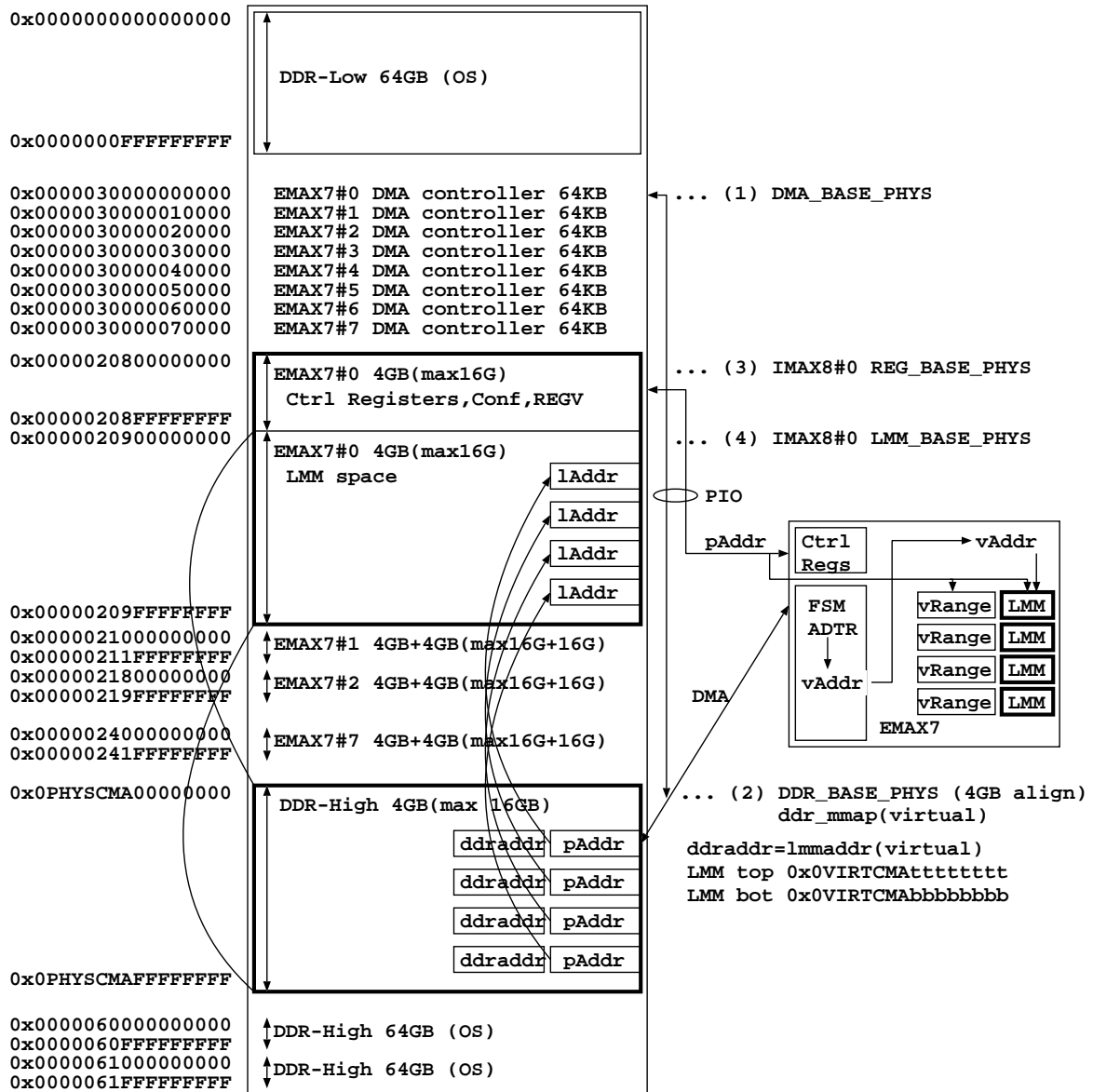


Figure.2.1: Physical memory space (IMAX3 compatible model: DMAbuf=4GB)

IMAX4 is configured by connecting multiple lanes of IMAX2 to a large-capacity external memory,

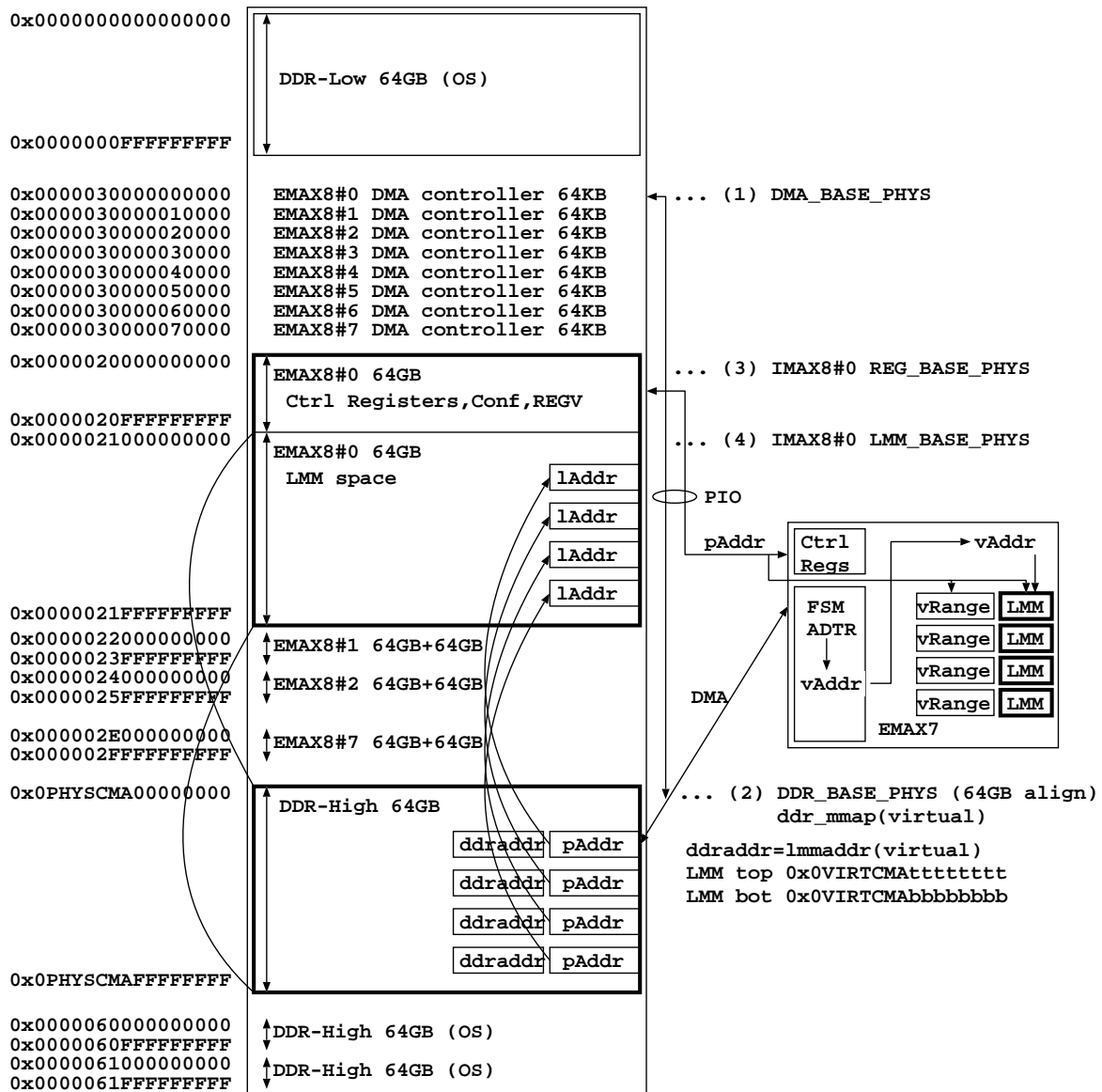


Figure.2.2: Physical memory space (IMAX4 native model: DMAbuf=64GB)

and the hardware/software interface for control is a collection of control interfaces for each IMAX2. Figures 2.1 and Fig.2.2 show the physical memory space of the CPU. The physical memory space related to IMAX consists of: (1) DMA control register space (64KB/lane starting from DMA\_BASE\_PHYS): Controls DMA between the CPU side cacheable DMA area and LMM in IMAX; (2) CPU side cacheable DMA area (4GB/64GB starting from DDR\_BASE\_PHYS): Space where DMA with LMM is possible; (3) IMAX control register space (4GB/64GB/lane starting from REG\_BASE\_PHYS, lane distance is 32GB/128GB): Mapped to CPU side space and referenced by PIO; (4) LMM space (4GB/64GB/lane starting from LMM\_BASE\_PHYS, lane distance is 32GB/128GB): Mapped to CPU side space and referenced by PIO/DMA. (3) and (4) are continuous spaces for each lane, and are mmap()ed in units of 8GB/128GB in the CPU side space. IMAX distinguishes between (3) and (4) by bit32 of the address for 8GB-mmap() and bit36 of the address for 128GB-mmap().

(4) performs address conversion using the following procedure.

1. When each lane of IMAX is started, an offset is set in the IMAX address conversion register (ADTR) by subtracting the starting physical address (lmm\_phys) of each LMM space from the starting virtual address (ddr\_mmap) of the CPU-side cacheable DMA area.



2. Each LMM is assigned up to four sets (overlapping possible between units) of address ranges within the CPU-side cacheable DMA area (the lower 4GB/64GB of virtual addresses per application), and each LMM is assigned to a part of the CPU-side cacheable DMA area. In other words, the LMM in IMAX functions as a software-controlled cache memory (total capacity is 512KB\*64unit=32MB) for the CPU-side cacheable DMA area.
3. For the CPU side address when starting DMA, specify a physical address in the CPU side cacheable DMA area calculated using `ddraddr (=lmmddr)` (application virtual address) - `ddr_mmap` (starting virtual address of the CPU side DMA area) + `ddr_phys` (`DDR_BASE_PHYS`). For the LMM side address, specify an LMM space physical address calculated using `ddraddr (=lmmaddr)` (application virtual address) - `ddr_mmap` (starting virtual address of the CPU side DMA area) + `lmm_phys` (starting physical address of each LMM starting from `LMM_BASE_PHYS`). The IMAX side interface recognizes that the reference target is (4), and uses the address (application virtual address) obtained by adding `ADTR` (`ddr_mmap - lmm_phys`) to the LMM space physical address for LMM reference. Each LMM compares this with the address ranges mentioned above, and if it is within its assigned range, performs R/W to its own LMM.

Note that for a correct comparison with the address range, `DDR_BASE_PHYS` and `ddr_mmap` must be aligned to the 4GB/64GB boundary. The steps for this are as follows (finally, `ADTR=0`).

```
/* 8GB temporally */
ddr_mmap = mmap(NULL, DDR_MMAP_SIZE*2, PROT_READ|PROT_WRITE, MAP_SHARED, fd_ddr, DDR_BASE_PHYS);
munmap(ddr_mmap, DDR_MMAP_SIZE*2); /* release and reset to aligned */
/* 4GB */
ddr_mmap = mmap((ddr_mmap+(DDR_MMAP_SIZE-1)) &~(DDR_MMAP_SIZE-1),
                DDR_MMAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_FIXED, fd_ddr, DDR_BASE_PHYS);
```

# Chapter 3

## Examples

### 3.1 Image recognition training/inference (tsim)

#### MNIST

```
cent% make -f Makefile-cent.emax8nc all clean
cent% cd ../; tsim/tsim-cent.emax8nc -x -t -I0 -C1 -F1
```

```
acap% make -f Makefile-acap.emax8+dma all clean
acap% cd ../; tsim/tsim-acap.emax8+dma -x -t -I0 -C1 -F1
```

#### CIFAR10

```
cent% make -f Makefile-cent.emax8nc all clean
cent% cd ../; tsim/tsim-cent.emax8nc -x -t -I1 -C6 -F2
```

```
acap% make -f Makefile-acap.emax8+dma all clean
acap% cd ../; tsim/tsim-acap.emax8+dma -x -t -I1 -C6 -F2
```

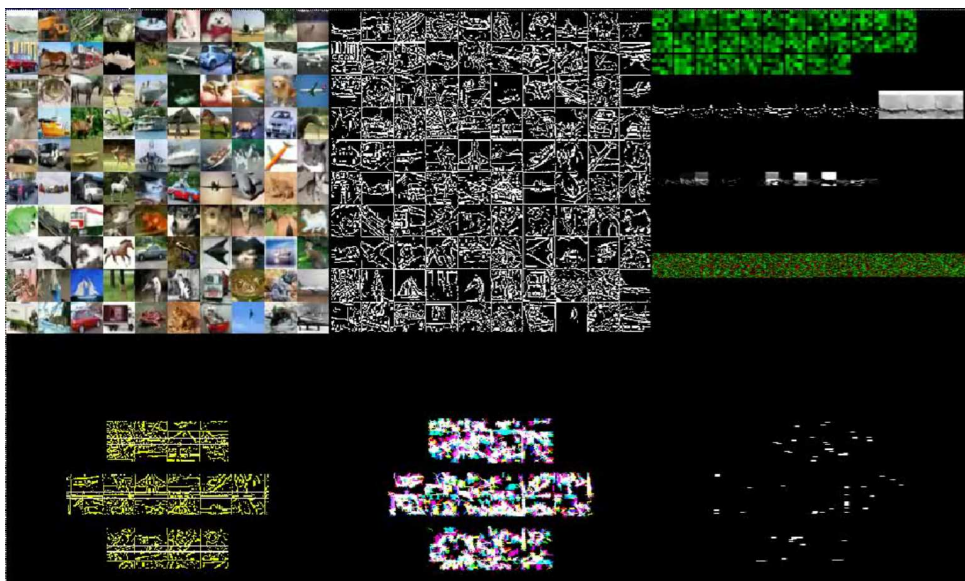


Figure.3.1: Image recognition (training + inference)

### 3.1.1 Header

IMAX4 uses ARM's multithreading to control multiple IMAX2 lanes. NTHREAD is the number of threads to be activated in ARM when multiple IMAX2 lanes are activated. EMAX\_LANE is the upper limit of the control variable set to control the IMAX2 lanes activated simultaneously. Also, NLANE is the number of IMAX2 lanes detected in the actual machine. IMAX2 detected in the real machine over EMAX\_LANE is not used. In other words, the value set in NLANE is always less than or equal to EMAX\_LANE. Note that NTHREAD must be greater than or equal to NLANE.

```
#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 IMAX4 thread driver

When launching multiple IMAX2 and configuring a macro pipeline, prepare a function wrapper (th\_inference in the example below) that includes the IMAX2 kernel, and use ARM's pthread\_create to launch NTHREAD threads. Thread arguments include thread number and enq/deq for pipeline synchronization.

```
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 IMAX4 thread wrapper

Multiple wrappers are started at the same time. In order for multiple threads to work in the pipeline internally, the code should be written to execute the part in charge based on the thread number given as an argument.

```

/* IMAX4 MACROPIPELINING for EVALUATION */
void th_inference(struct th_inference_args *args)
{
    int THREAD      = args->thid;
    float4D *slice  = args->slice;
    CNNNet *net     = args->net;
    int batch_size  = args->batch_size;
    int nchan       = args->nchan;
    int insize      = args->insize;
    int j, k;
    /******
    /* TARGET of MACRO-PIPELINING/IMAX4 */
    /******
    slice->nstrides = batch_size;
    slice->nchannel = xttest.nchannel;
    slice->kstrides = xttest.kstrides;
    slice->stride_size = xttest.stride_size;
    for (j=0; j+batch_size<=xttest.nstrides; j+=batch_size) {
        if (THREAD == 0) {
            slice->data = &(xttest.data[j*xttest.stride_size*xttest.kstrides*xttest.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 -> S1+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/IMAX4 */
    /******
    args->stat = 0; /* idle */
}

```

### 3.1.4 IMAX4 region

The `nn_forward` below is the top-level function that performs the inference. Currently, the argument `THREAD` is manually associated with the IMAX2 lane number (`LANE`) to be used. In addition, `enq/deq` is used to synchronize the macro pipeline. In the future, when automation tools are completed, manual work will be unnecessary.

```

void nn_forward(int MACROPIPE, int THREAD, CNet *net, struct c *c, struct f *f, float2D *oubatch, int spike_mode)
{ /* NTHREAD 1:MACRO_PIPE_OFF 2--:MACRO_PIPE_ON */
  /*
   *          EMAX8: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]));
    max_pooling(&(net->npool[0]), &(net->nhidden[0]), c[0].psize, c[0].psize); /* V1->c.nhidden[0].data */
    flat4Dto2D(&(net->nflat[0]), &(net->npool[0])); /* ->c.npool[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 == 1*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(EMAX8)
      if (l >= NLANE) {
        printf("nn_forward_CNN: LANE(%d) >= NLANE(%d)\n", l, NLANE);
        exit(1);
      }
      LANE = l;
      emax8[LANE].sigwait = 1; /* ON */
      emax8[LANE].sigstat = &th_inference_args[THREAD].stat;
      emax8[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
    /******input*** V CNN */
  #ifndef CUDA
  /*★ IMAx4 ★*/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 == 1*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 == 1*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 == 1*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].deq==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 */
}
#endif defined(EMAX8)
if (CNN_DEPTH >= NLANE) {
    printf("nn_forward_FC: CNN_DEPTH(%d) >= NLANE(%d)\n", CNN_DEPTH, NLANE);
    exit(1);
}
LANE = CNN_DEPTH;
emax8[LANE].sigwait = 1; /* ON */
emax8[LANE].sigstat = &th_inference_args[THREAD].stat;
emax8[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)
        flat4Dto2D(&(net->nflat[0]), &(net->npool[CNN_DEPTH-1])); /******npool**** A CNN */
    else
        copy2D(&(net->nflat[l]), &(net->nout[l-1])); /******nflat**** V FC */

    // second layer full-connection
    /*★ IMAX4 ★*/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, sigmoid, 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]));
}
}
}

```

## 3.2 LLM GPT (vsim)

### CHAT

```
ubuntu% make -f Makefile-ubuntu.emax8nc all clean
ubuntu% (cd cformers; python3 chat.py) or
ubuntu% ./vsim-ubuntu.emax8nc gptneox
-m /home/nakashim/.cformers/models/OpenAssistant/oasst-sft-1-pythia-12b/int4_fixed_zero
-prompt "50278 12092 2 0 50281" -seed 42 -threads 4 -n_predict 100 -top_k 20 -top_p 0.95
-temp 0.85 -repeat_last_n 64 -repeat_penalty 1.3
```

```
acap% make -f Makefile-acap.emax8+dma all clean
acap% ./vsim-acap.emax8+dma gptneox
-m /home/nakashim/.cformers/models/OpenAssistant/oasst-sft-1-pythia-12b/int4_fixed_zero
-prompt "50278 12092 2 0 50281" -seed 42 -threads 4 -n_predict 100 -top_k 20 -top_p 0.95
-temp 0.85 -repeat_last_n 64 -repeat_penalty 1.3
```

```
static void ggml_compute_forward_mul_mat_q4_0_f32( /* ggml.c */
    const struct ggml_compute_params * params,
    const struct ggml_tensor * src0,
    const struct ggml_tensor * src1,
    struct ggml_tensor * dst) {

    if (params->type == GGML_TASK_INIT) {
#ifdef EMAX8
        if (!IMAX_READY) {
            IMAX_READY = 1;
            NTHREAD = nth; /* max 8 for monitor.c */
            init_xmax(nth); /* Nakashima */
        }
#endif
        :
        memset(params->wdata, 0, params->wsize);
        return;
    }

    if (params->type == GGML_TASK_FINALIZE) {
        if (nb01 >= nb00) {
            return;
        }
        :
    }

    if (nb01 >= nb00) {
        const int nr = ne01*ne02*ne03;
        const int dr = (nr + nth - 1)/nth;
        const int ir0 = dr*ith;
        const int ir1 = MIN(ir0 + dr, nr);
        void * wdata = params->wdata;
        th_inference_args[ith].thid = ith;
        th_inference_args[ith].nth = nth;
        th_inference_args[ith].ith = ith;
        th_inference_args[ith].nr = nr;
        th_inference_args[ith].ir0 = ir0;
        th_inference_args[ith].ir1 = ir1;
        th_inference_args[ith].src0 = src0;
        th_inference_args[ith].src1 = src1;
        th_inference_args[ith].wdata = wdata;
        th_inference_args[ith].dst = dst;
        imax_ggml_compute_forward_mul_mat_q4_0_f32(&th_inference_args[ith]); /* See vsim in EMAX6/ZYNQ64 Handbook */
    } else {
        :
    }
}
}
```



### 3.3 LLM GPT (llama)

#### CHAT

```
ubuntu% make -f Makefile-ubuntu.emax8nc all clean
ubuntu% sudo ./llama-cli-ubuntu.emax8nc -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q8_0.gguf
-p "Hello" -n 32
ubuntu% sudo ./llama-cli-ubuntu.emax8nc -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q2_K.gguf
-p "Hello" -n 32
ubuntu% sudo ./llama-cli-ubuntu.emax8nc -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q8_0.gguf
-p "Prime numbers smaller than ten" -n 32
ubuntu% sudo ./llama-cli-ubuntu.emax8nc -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q2_K.gguf
-p "Prime numbers smaller than ten" -n 32
```

```
acap% make -f Makefile-acap.emax8+dma all clean
acap% sudo ./llama-cli-acap.emax8+dma -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q8_0.gguf
-p "Hello" -n 32
acap% sudo ./llama-cli-acap.emax8+dma -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q2_K.gguf
-p "Hello" -n 32
acap% sudo ./llama-cli-acap.emax8+dma -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q8_0.gguf
-p "Prime numbers smaller than ten" -n 32
acap% sudo ./llama-cli-acap.emax8+dma -t 4 -s 8 -fa
-m ~/llama/model/rinna-yourri-7b-instruction-gguf/rinna-yourri-7b-instruction-q2_K.gguf
-p "Prime numbers smaller than ten" -n 32
```

```
arch06:1
system info: n_threads = 2 / 2 | AVX = 0 | AVX_VNNI = 0 | AVX2 = 0 | AVX512 = 0 | AVX512_VBMI = 1 | SVE = 0 | ARM_FMA = 1 | F16C = 0 | FP16_VA = 0 | WASM_SIMD = 0 | BLAS = 0 | SSE3 = 0 |
sampling:
  repeat_last_n = 64, repeat_penalty = 1.000, frequency_penalty = 0.000, presence_penalty = 0.000, top_k = 40, tfs_z = 1.000, top_p = 0.950, min_p = 0.050, typical_p = 1.000, temp = 0.000, mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temperature
generate: n_ctx = 4096, n_batch = 2048, n_predict = 32, n_keep = 1

Prime numbers smaller than ten== 8 == SRC1 ggml_backend_cpu_graph_compute : 0000aaaafb184f5
are 2, 3, 5, 7, 11, 13 and 17
arch06:1/usr/home/nakashim

== 6 == SRC1 ggml_backend_cpu_graph_compute : 000055d50daa78c0 size=0000000000006218 0 dmabuf
system info: n_threads = 1 / 32 | AVX = 1 | AVX_VNNI = 1 | AVX2 = 1 | AVX512 = 1 | AVX512_VBMI = 0 | SVE = 0 | ARM_FMA = 0 | F16C = 1 | FP16_VA = 0 | WASM_SIMD = 0 | BLAS = 0 | SSE3 = 1 |
sampling:
  repeat_last_n = 64, repeat_penalty = 1.000, frequency_penalty = 0.000, presence_penalty = 0.000, top_k = 40, tfs_z = 1.000, top_p = 0.950, min_p = 0.050, typical_p = 1.000, temp = 0.000, mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temperature
generate: n_ctx = 4096, n_batch = 2048, n_predict = 32, n_keep = 1

Prime numbers smaller than ten== 7 == SRC1 ggml_backend_cpu_graph_compute : 000055d50db6002
are 2, 3, 5, 7, 11, 13, 17, 19
arch06:1/usr/
```

Figure.3.2: Llama

```

monitor_time_start(ith, T_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK); /* Nakashima */
#if !defined(EMAX8)
// attempt to reduce false-sharing (does not seem to make a difference)
// 16 * 2, accounting for mmla kernels
float tmp[32];
for (int64_t iir1 = ir1_start; iir1 < ir1_end; iir1 += blk1) {
    for (int64_t iir0 = ir0_start; iir0 < ir0_end; iir0 += blk0) {
        for (int64_t ir1 = iir1; ir1 < iir1 + blk1 && ir1 < ir1_end; ir1 += num_rows_per_vec_dot) {
            :
        }
    }
}
#else
th_inference_args[ith].thid      = ith;
th_inference_args[ith].dst      = dst;
th_inference_args[ith].num_rows_per_vec_dot = num_rows_per_vec_dot;
th_inference_args[ith].ir0_start = ir0_start;
th_inference_args[ith].ir0_end   = ir0_end;
th_inference_args[ith].ir1_start = ir1_start;
th_inference_args[ith].ir1_end   = ir1_end;
th_inference_args[ith].src0      = src0;
th_inference_args[ith].src1      = src1;
th_inference_args[ith].type      = type;
th_inference_args[ith].src1_cont = src1_cont;
th_inference_args[ith].vec_dot   = vec_dot;
th_inference_args[ith].vec_dot_type = vec_dot_type;
th_inference_args[ith].wdata     = wdata;
th_inference_args[ith].row_size  = row_size;
th_inference_args[ith].r2        = r2;
th_inference_args[ith].r3        = r3;
th_inference_args[ith].blk0      = blk0;
th_inference_args[ith].blk1      = blk1;
th_inference_args[ith].src1_col_stride = src1_col_stride;
switch (type) {
case GGML_TYPE_F16:
    monitor_time_start(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_F16);
    imax_compute_forward_mul_mat_one_chunk_f16(&th_inference_args[ith]); /* ggml_vec_dot_f16() Nakashima */
    monitor_time_end(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_F16);
    break;
case GGML_TYPE_Q2_K:
    monitor_time_start(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q2K);
    imax_compute_forward_mul_mat_one_chunk_q2_k(&th_inference_args[ith]); /* ggml_vec_dot_q2_K_q8_K() Nakashima */
    monitor_time_end(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q2K);
    break;
case GGML_TYPE_Q3_K:
    monitor_time_start(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q3K);
    imax_compute_forward_mul_mat_one_chunk_q3_k(&th_inference_args[ith]); /* ggml_vec_dot_q3_K_q8_K() Nakashima */
    monitor_time_end(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q3K);
    break;
case GGML_TYPE_Q6_K:
    monitor_time_start(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q6K);
    imax_compute_forward_mul_mat_one_chunk_q6_k(&th_inference_args[ith]); /* ggml_vec_dot_q6_K_q8_K() Nakashima */
    monitor_time_end(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q6K);
    break;
case GGML_TYPE_Q8_0:
    monitor_time_start(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q80);
    imax_compute_forward_mul_mat_one_chunk_q8_0(&th_inference_args[ith]); /* ggml_vec_dot_q8_0_q8_0() Nakashima */
    monitor_time_end(ith, IMAX_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK_Q80);
    break;
default:
    printf("ERROR in ggml_EMAX8: unknown ggml_type=%d\n", type);
    break;
}
#endif
monitor_time_end(ith, T_COMPUTE_FORWARD_MUL_MAT_ONE_CHUNK); /* Nakashima */

```

# Chapter 4

## Appendix

### 4.1 Prototype systems

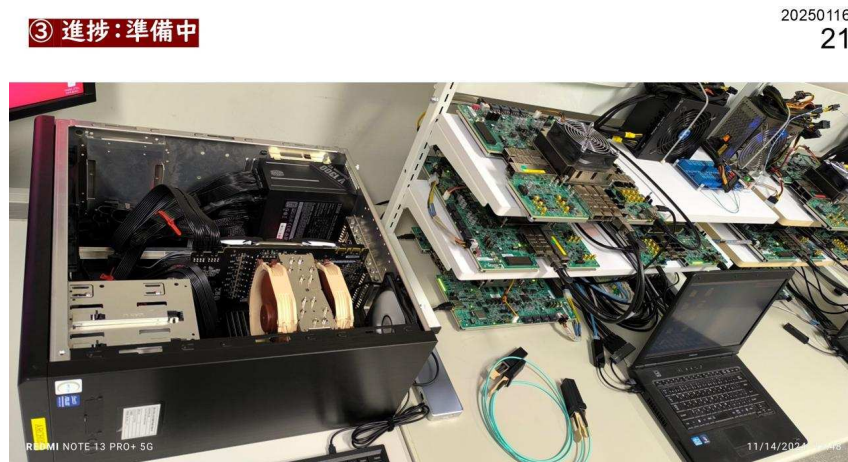


Figure.4.1: Prototype systems

### 4.2 References

This chapter lists related specifications, standards, references, related source programs, and tool chains.

- EMAX5 Basic patent ..... proj-arm64/doc/pat35.tgz
- IMAX Basic patent ..... proj-arm64/doc/pat36.tgz
- IMAX4 Handbook ..... proj-arm64/doc/emax8/emax8e.pdf
- IMAX3 Handbook ..... proj-arm64/doc/emax7/emax7e.pdf
- IMAX3 Preprocessor ..... proj-arm64/src/conv-c2d/conv-c2d
- IMAX2 Handbook ..... proj-arm64/doc/emax6/emax6e.pdf
- IMAX2 Preprocessor ..... proj-arm64/src/conv-c2c/conv-c2c
- IMAX2 Simulator ..... proj-arm64/src/csim/csim
- Example(3x3-2D cnn) ..... proj-arm64/sample/mm\_cnn\_if/cnn+rmm.c
- Example(3x3-3D cnn) ..... proj-arm64/sample/mm\_cnn\_if/cnn3d+rmm.c
- Example(MM) ..... proj-arm64/sample/mm\_cnn\_if/mm+rmm.c
- Example(Inverse matrix) ..... proj-arm64/sample/mm\_cnn\_if/inv+rmm.c
- Example(Lightfield rendering) ..... proj-arm64/sample/mm\_cnn\_if/gather+rmm.c

- Example(Lightfield depth map) .....proj-arm64/sample/mm\_cnn\_if/gdepth+rmm.c
- Example(Image recognition+stochastic ALU) ..... proj-arm64/sample/tsim/imax.c
- Example(Chat GPT (ggml)) .....proj-arm64/sample/vsim/imax.c
- Example(Chat GPT (llama)) ..... proj-arm64/sample/llama/imax.c