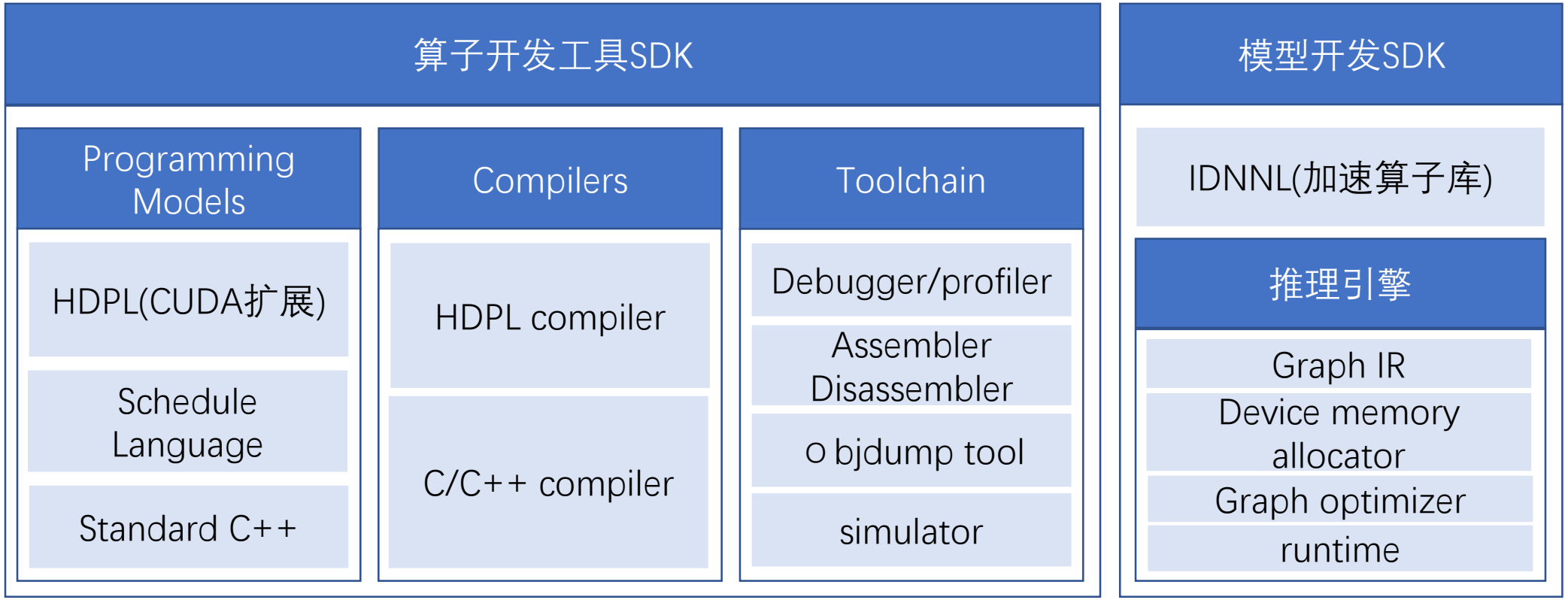
AI Core软件栈设计

星汉工具链

- 可编程性：使用C/C++编程，兼容CUDA前端语法
- 开发者套件是个系统工程，基于LLVM，包括编译器，链接器，调试器，profiler，算子库
- 面向用户：内部算子开发者，外部定制化开发者



编程模型

- **需要解决的主要问题**
  1. 针对AI Core的特殊架构，提供编程模型，比如CIM的CONV的实现
  2. 卷积之外的新算子的开发，提供给用户高效开发新算子的能力,算法在不断的变化过程中，我们很难预测未来新的算法会带来的算子的需求。
  3. 提供给用户适合机器学习算子开发的并发模式(前期主要针对视觉模型，分类和目标检测)
  4. 异构编程(heterogeneous programming)
- **主要的并行模型**
  1．消息转递：MPI
  2． 数据并行：OpenMP，OpenCL，OpenAcc，CUDA
  3．共享内存多线程: java synchronized，pthread

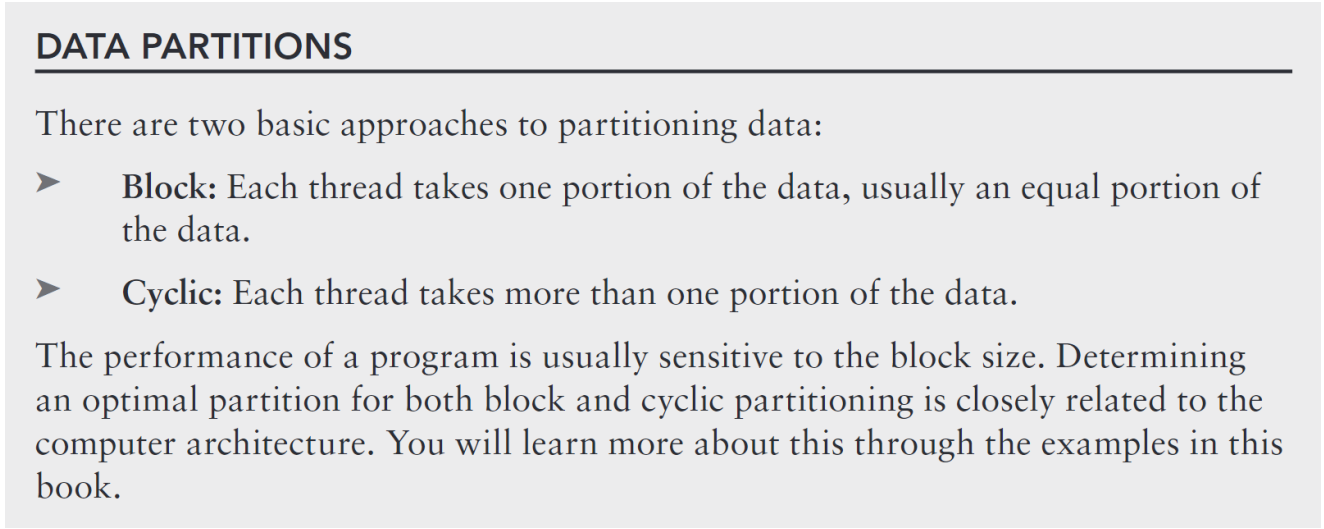
Parallelism

- Task parallelism

  - There are many tasks or functions that can be operated **independently** and largely in parallel. Task parallelism focuses on distributing functions across multiple cores. (eg. http server)

- Data parallelism

  - There are many data items that can be operated on at the same time. Data parallelism focuses on distributing the data across multiple cores.



DATA PARTITIONS

There are two basic approaches to partitioning data:

- **Block:** Each thread takes one portion of the data, usually an equal portion of the data.
- **Cyclic:** Each thread takes more than one portion of the data.

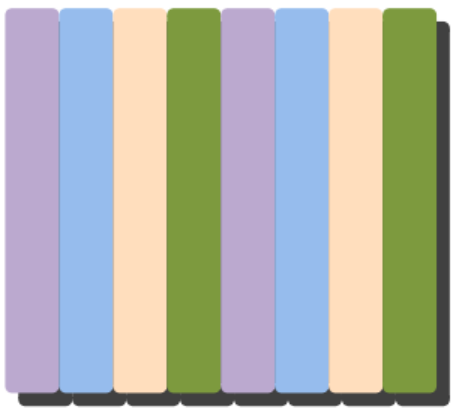
The performance of a program is usually sensitive to the block size. Determining an optimal partition for both block and cyclic partitioning is closely related to the computer architecture. You will learn more about this through the examples in this book.

Block partition: each thread takes one data block

Cyclic partition: each thread takes two data blocks

Block partition on one dimension

Block partition on both dimensions



Cyclic partition on one dimension

HDPL

- Houmo/Heterogeneous Data Parallel Language
- 星汉体系架构的CUDA语言扩展
- 高效解决数据并行问题
- 分析“.hu”为后缀的文件，做语法扩展
- Host和Device异构编程

HDPL

- **C/C++ with Minimal Extensions**
- **Declaration specifiers to indicate where things live**

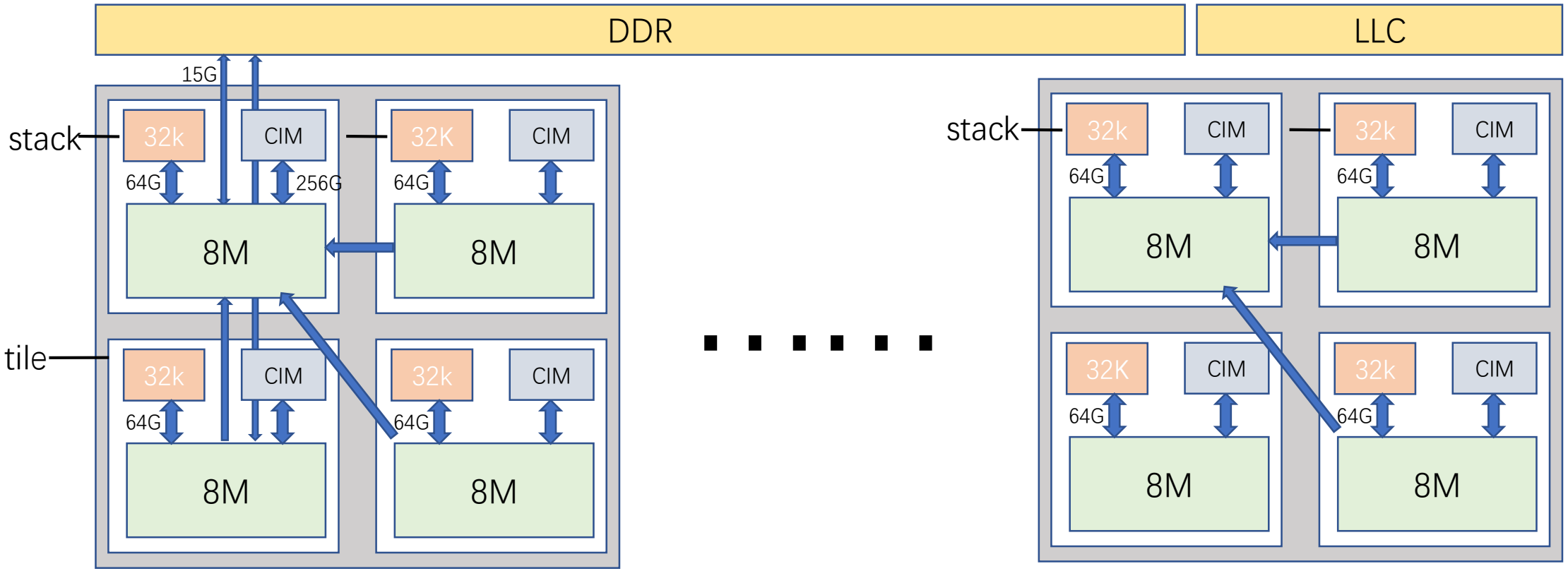
  __global__ void KernelFunc(...); // kernel function, runs on device
  __device__ int GlobalVar; // variable in device memory
  __slocal__ int LocalVar; // local variables
- **Extend function invocation syntax for parallel kernel launch**

  KernelFunc<<<m, n>>>(...); // launch m cores n tiles
- **Tile and Core info**

  get_tile_id(), get_core_id()

存储器层次结构

HDPL语言关键字	名称	容量
__global__	Global memory	8G
__slocal__	local memory	8M * 16 = 128M
	register	32个64bit per core



设备内存分配和拷贝

- **分配设备内存**

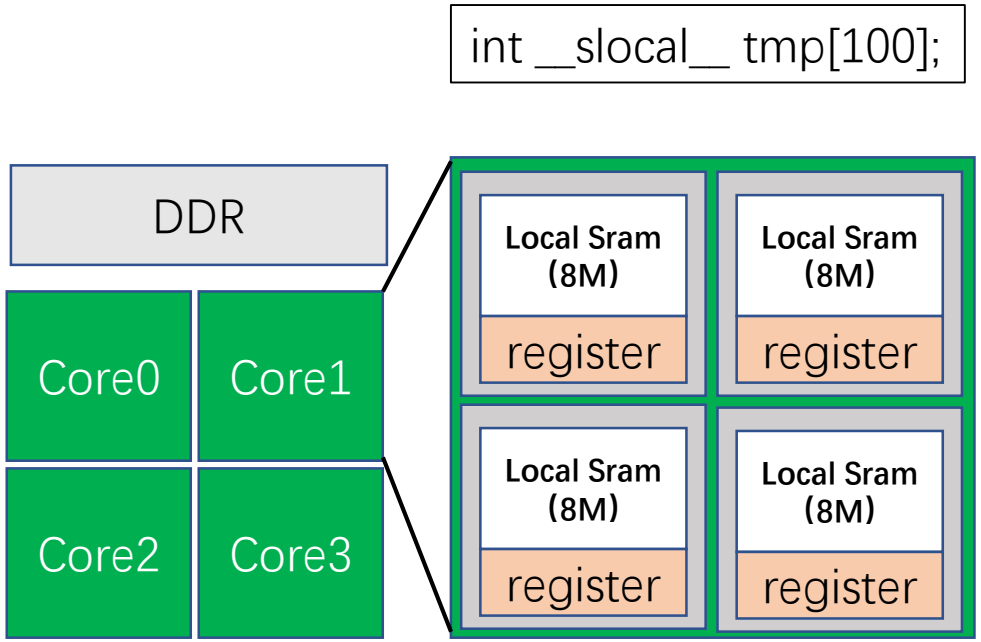
```
int HmMalloc(void* pdevptr, uint32_t size, DEVICE mem_kind);
```

- **设备和Host的拷贝**

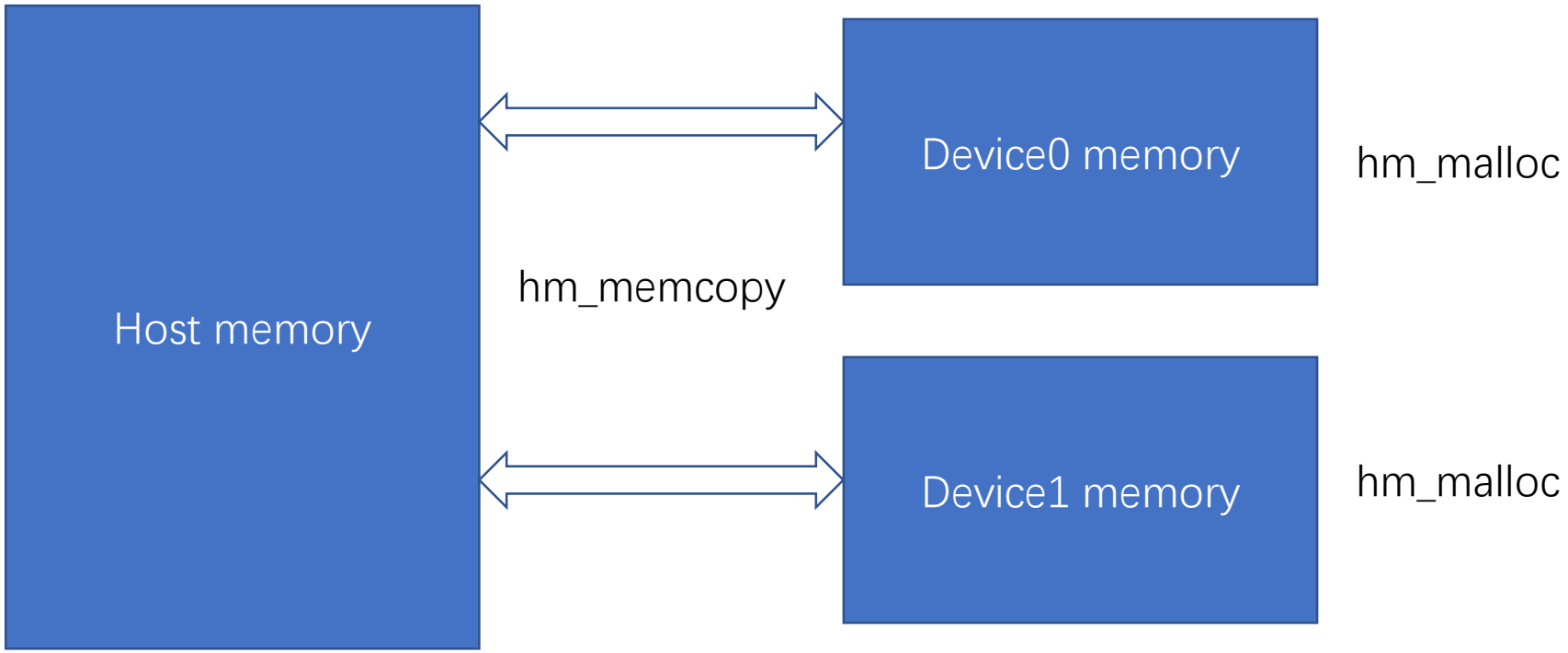
```
int HmMemcpy(void* src, void* des, uint32_t, size, DEVICE mem_kind);
```

- **Local Sram**

  __slocal__关键字

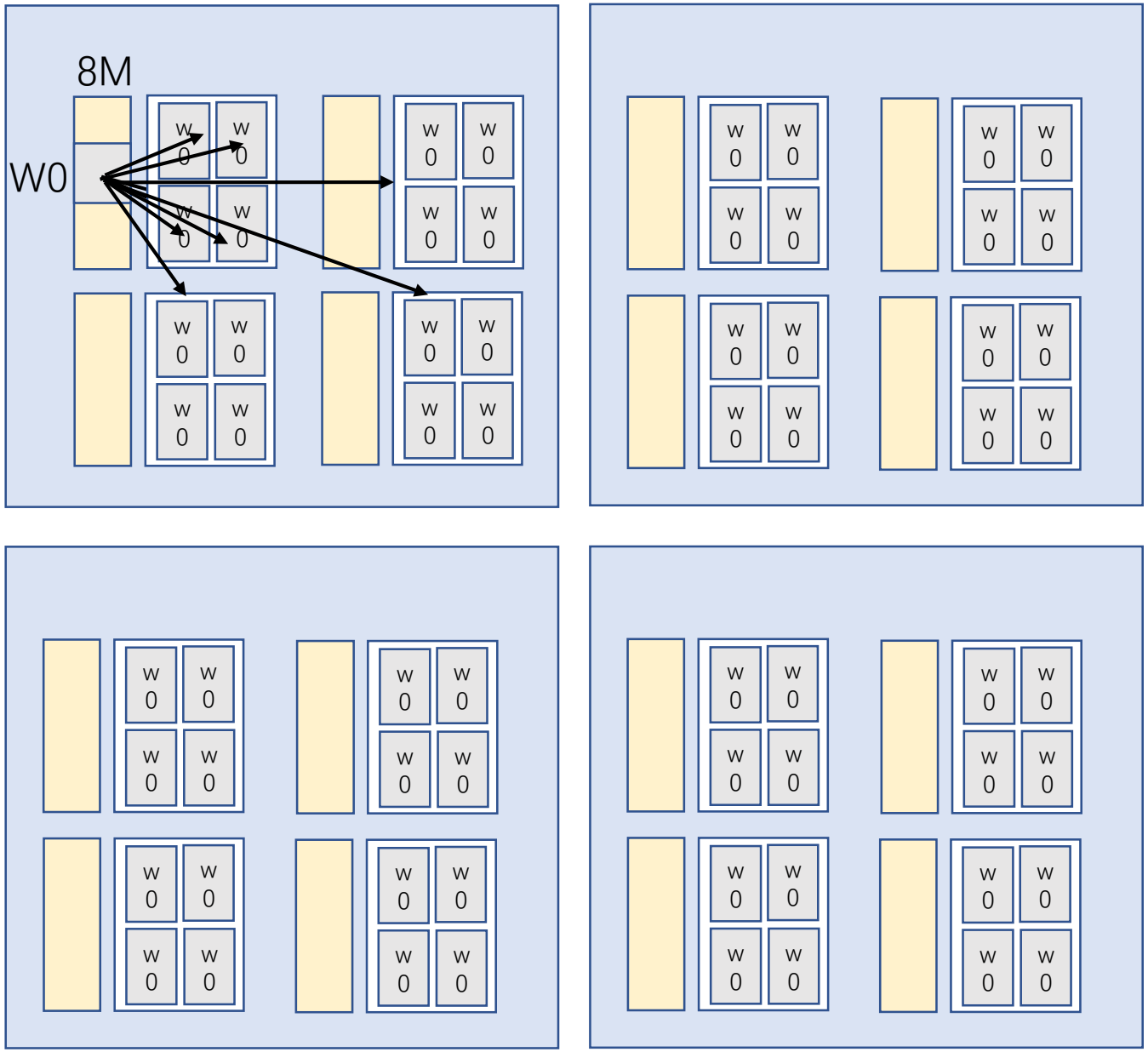


Heterogeneous Memory Model



```
static inline void __asm__ __cim_conv(const void* fm_in, const void* fm_out,
                                       const int fm_in_w, const int fm_in_h,
                                       const int fm_in_c, const int fm_out_w,
                                       const int fm_out_h, const int fm_out_c,
                                       const int kernel_w, const int kernel_h,
                                       const bool stride, const bool padding_top,
                                       const bool padding_left, const int padding_value) {
  CmdInfo cmd;

  cmd.config0 = FEATURE_VAL |
                ((fm_in & 0x1ffff) << FM_IN_OFFSET) |
                ((fm_out & 0x1ffff) << FM_OUT_OFFSET) |
                ((kernel_w & 0xf) << KERNEL_W_OFFSET) |
                ((kernel_h & 0xf) << KERNEL_H_OFFSET) |
                ((fm_in_w & 0xfff) << FM_IN_W_OFFSET);

  cmd.config1 = ((fm_in_h & 0xfff) << FM_IN_H_OFFSET)          |
                ((fm_in_c & 0x3f) << FM_IN_C_OFFSET)           |
                ((fm_out_w & 0xfff) << FM_OUT_W_OFFSET)        |
                ((fm_out_h & 0xfff) << FM_OUT_H_OFFSET)        |
                ((fm_out_c & 0xfff) << FM_OUT_C_OFFSET)        |
                ((stride & 0x1) << STRIDE_OFFSET)              |
                ((padding_top & 0x1) << PADDING_TOP_OFFSET)    |
                ((padding_left & 0x1) << PADDING_LEFT_OFFSET)  |
                ((padding_value & 0xff) << PADDING_VALUE_OFFSET);

  __asm__ __volatile__ ("wr128_cscpr %0, %1, cspr_0, 0\n"
                        :
                        : "r" (cmd.config0), "r" (cmd.config1)
                        : "memory"
                       );
  __asm__ __volatile__ ("tesnor_config cpcsr_0\n");
} « end   cim conv »
```

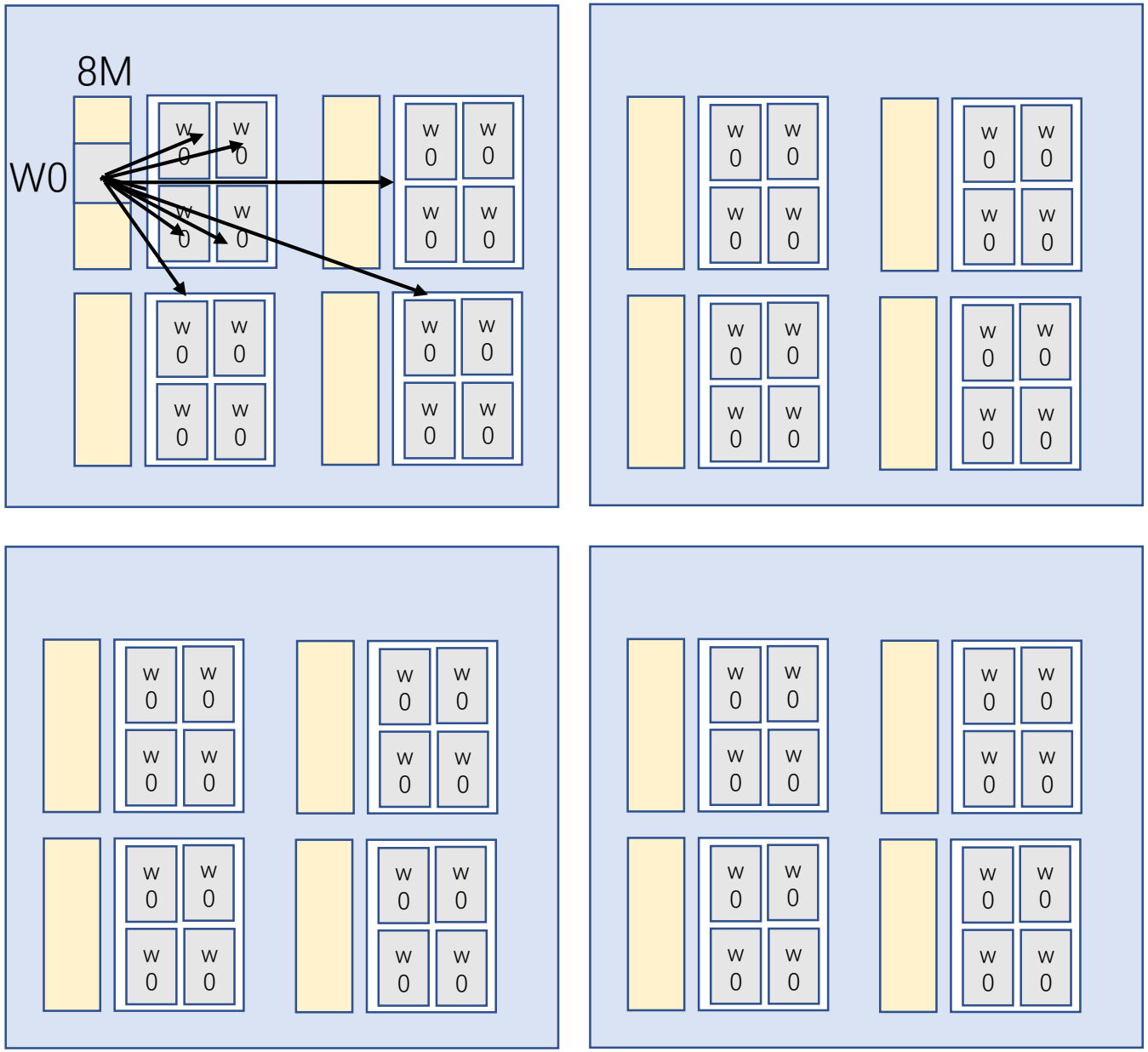
Convolution example: weight shape [64 x 64 x 3 x 3]



[1 x 64 x 152 x 152] x [64 x 64 x 4 x 3]

```
__global__ void conv2d_1macro(const int8_t* f1,
    const int8_t* pout, int c, int h, int w,
    int out_c, int padding, int stride,
    const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  if (tile_id == 0 && core_id == 0) {
    __sram_broadcast2_cim_2d(weight0, 0, 0, 0, size);
    __sram_broadcast2_cim_2d(weight0, 0, 1, 0, size);
    __sram_broadcast2_cim_2d(weight0, 0, 2, 0, size);
    __sram_broadcast2_cim_2d(weight0, 0, 3, 0, size);

    __sram_broadcast2_cim_2d(weight0, 1, 0, 0, size);
    __sram_broadcast2_cim_2d(weight0, 1, 1, 0, size);
    __sram_broadcast2_cim_2d(weight0, 1, 2, 0, size);
    __sram_broadcast2_cim_2d(weight0, 1, 3, 0, size);

    __sram_broadcast2_cim_2d(weight0, 2, 0, 0, size);
    __sram_broadcast2_cim_2d(weight0, 2, 1, 0, size);
    __sram_broadcast2_cim_2d(weight0, 2, 2, 0, size);
    __sram_broadcast2_cim_2d(weight0, 2, 3, 0, size);

    __sram_broadcast2_cim_2d(weight0, 3, 0, 0, size);
    __sram_broadcast2_cim_2d(weight0, 3, 1, 0, size);
    __sram_broadcast2_cim_2d(weight0, 3, 2, 0, size);
    __sram_broadcast2_cim_2d(weight0, 3, 3, 0, size);
  }

  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
} « end conv2d_1macro »
```

Weight shape [64 x 64 x 3 x 3]



每个core执行相同的代码

```
__global__ void conv2d_1macro(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  if (tile_id == 0 && core_id == 0) {
    __sram_broadcast2_all_cim_2d(weight0, 0, 0, 0, size);
  }

  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

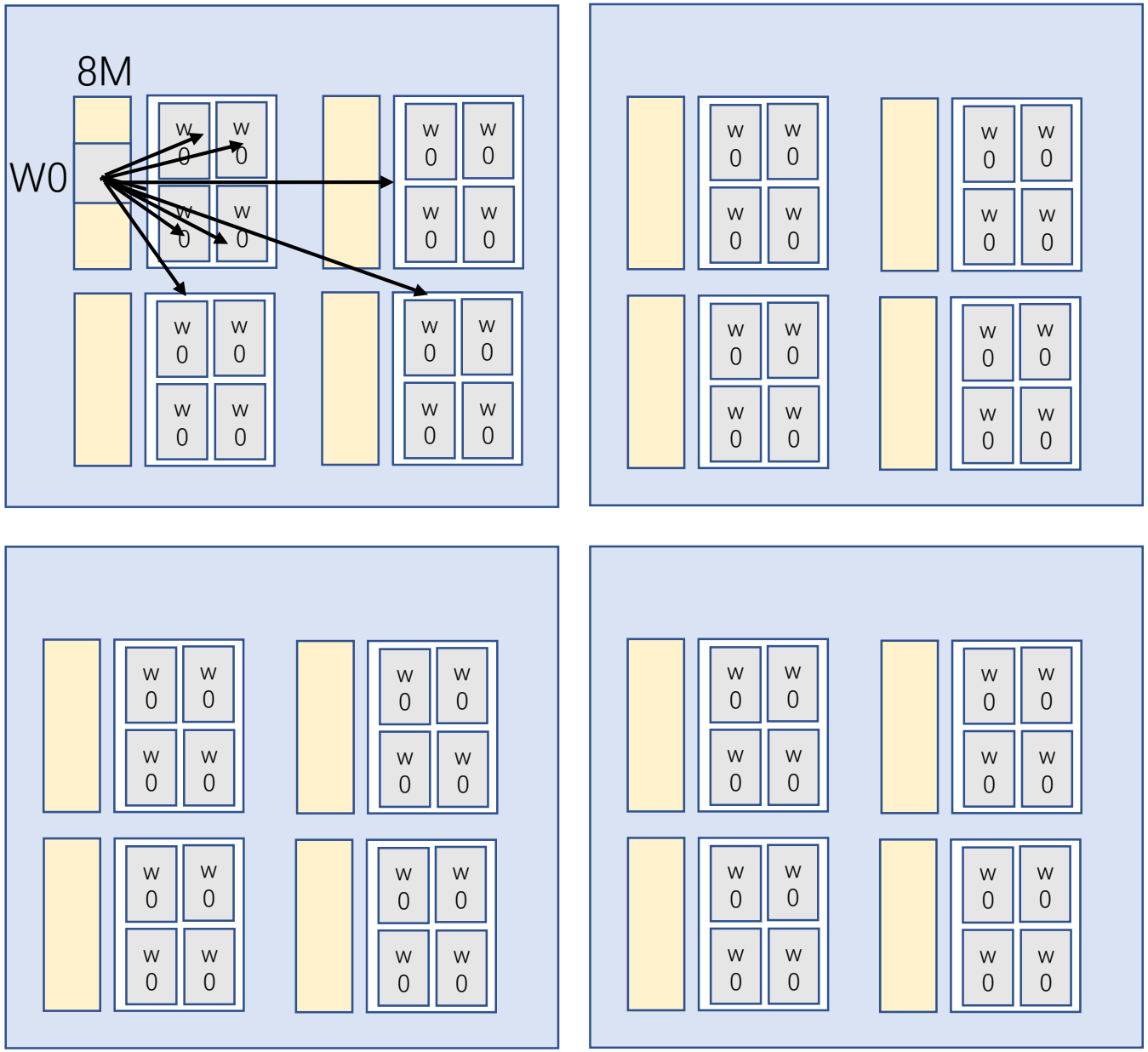
每个core执行不同的代码　Code of core0, tile0

```
__global__ void conv2d_1macro_t0(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  __sram_broadcast2_all_cim_2d(weight0, 0, 0, 0, size);
  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

Code of others

```
__global__ void conv2d_1macro_tn0(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

Weight shape [64 x 64 x 3 x 3]



每个core执行相同的代码

```
__global__ void conv2d_1macro(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  if (tile_id == 0 && core_id == 0) {
    __sram_broadcast2_all_cim_2d(weight0, 0, 0, 0, size);
  }

  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

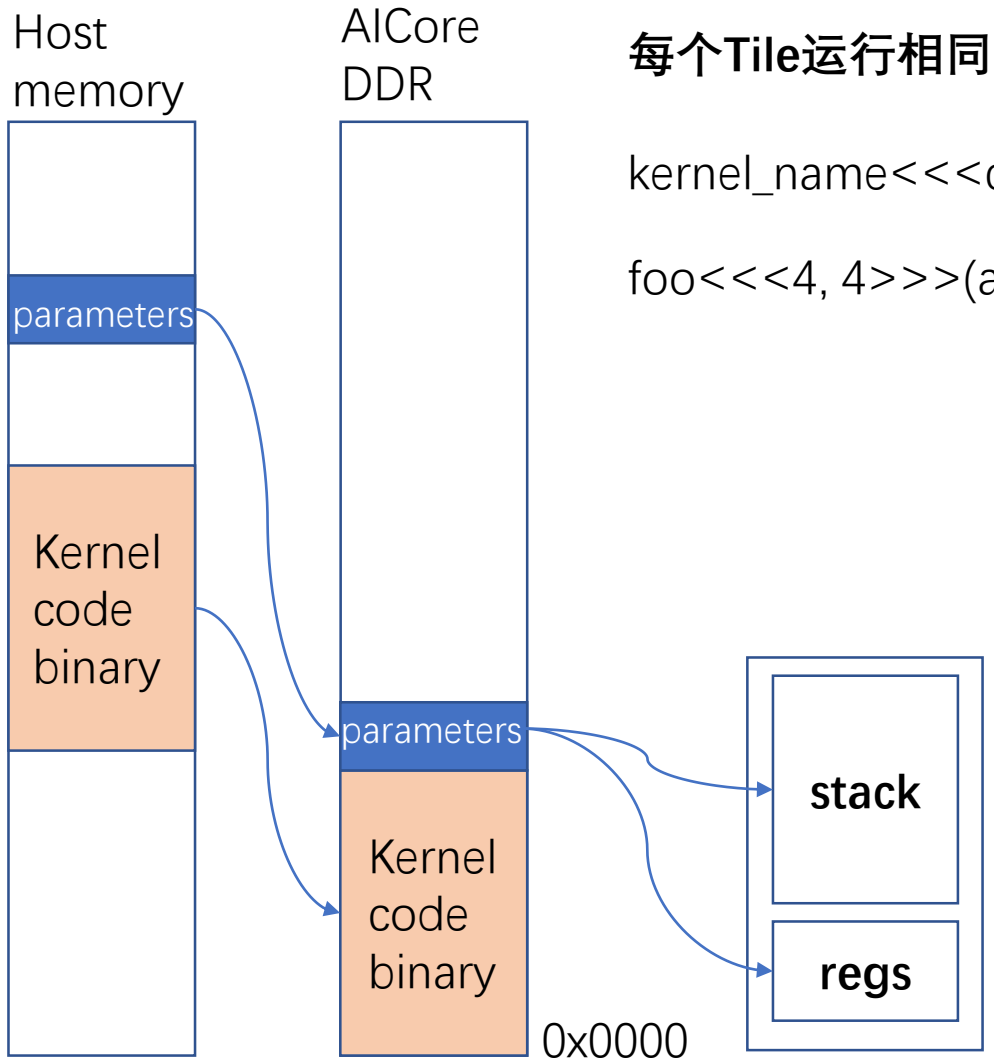
每个core执行不同的代码　Code of core0, tile0

```
__global__ void conv2d_1macro_t0(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  __sram_broadcast2_all_cim_2d(weight0, 0, 0, 0, size);
  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

Code of others

```
__global__ void conv2d_1macro_tn0(const int8_t* f1,
const int8_t* pout, int c, int h, int w, int out_c, int padding, int stride,
const int8_t* weight0) {
  int tile_id = get_tile_id();
  int core_id = get_core_id();
  int size = 64 * 9;
  __sync(cim); // (lock0-15);
  __conv2d(f1, pout c, h / 2 + 1, w / 2 + 1, padding, out_c, stride);
}
```

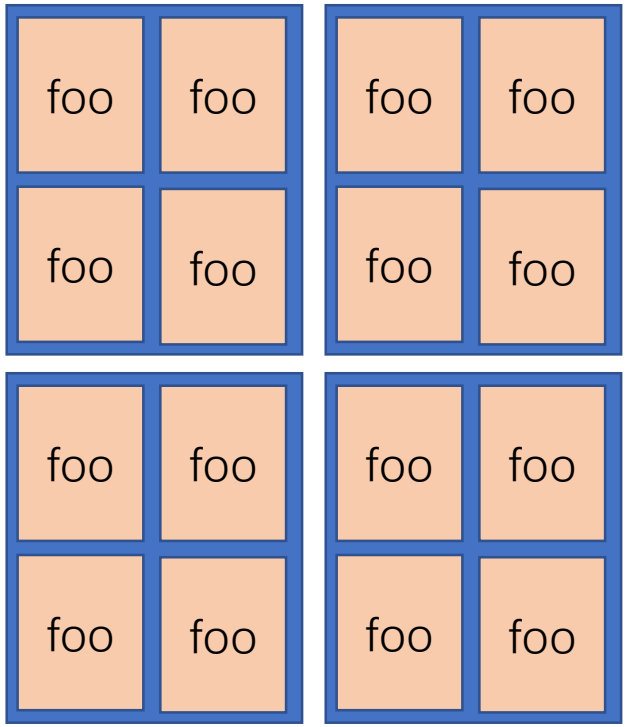
Launch Kernel流程

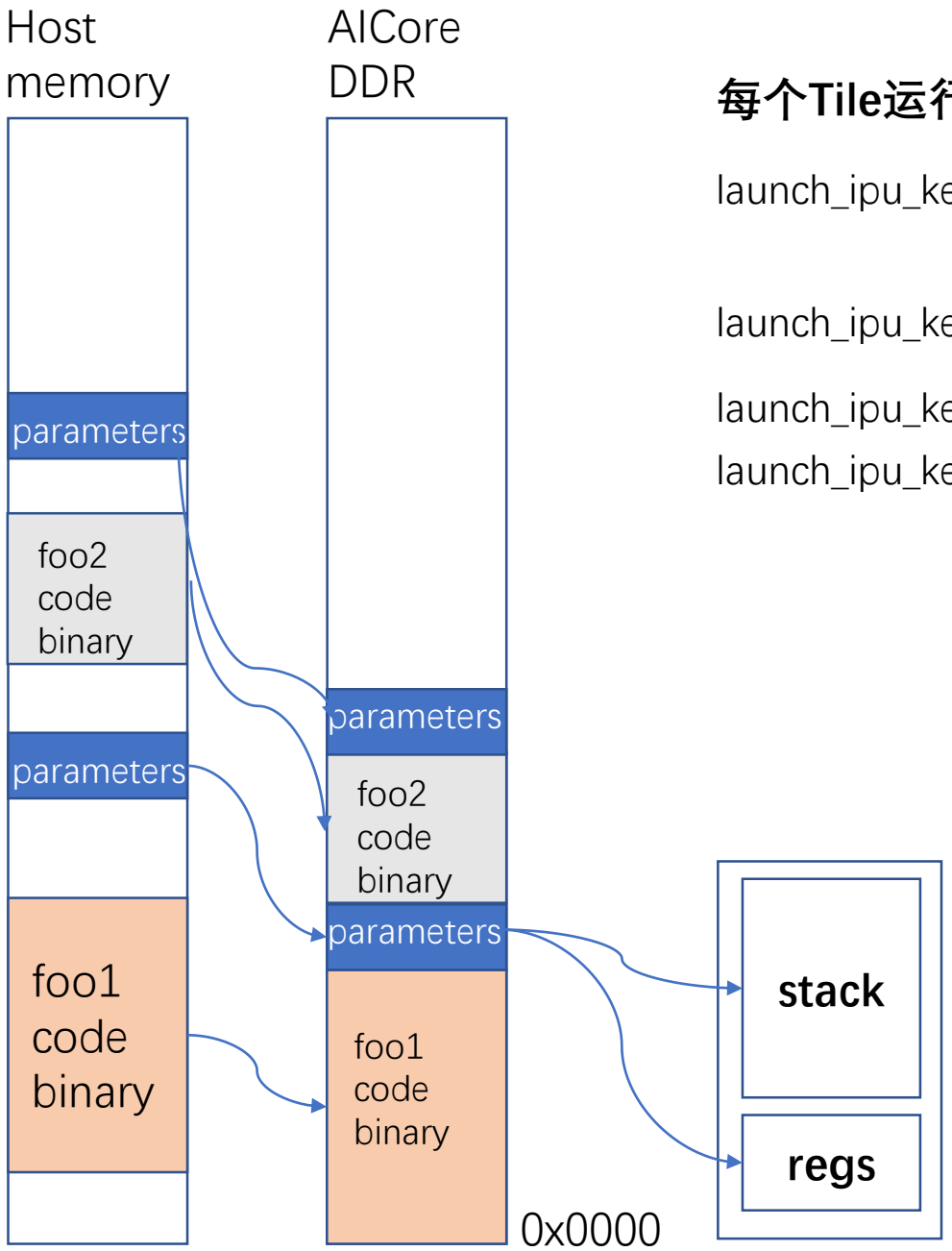


每个Tile运行相同的code的模式

kernel_name<<<core_num, tile_num, sharedMemSize, stream>>>(argument list);

foo<<<4, 4>>>(a,b,c,d);





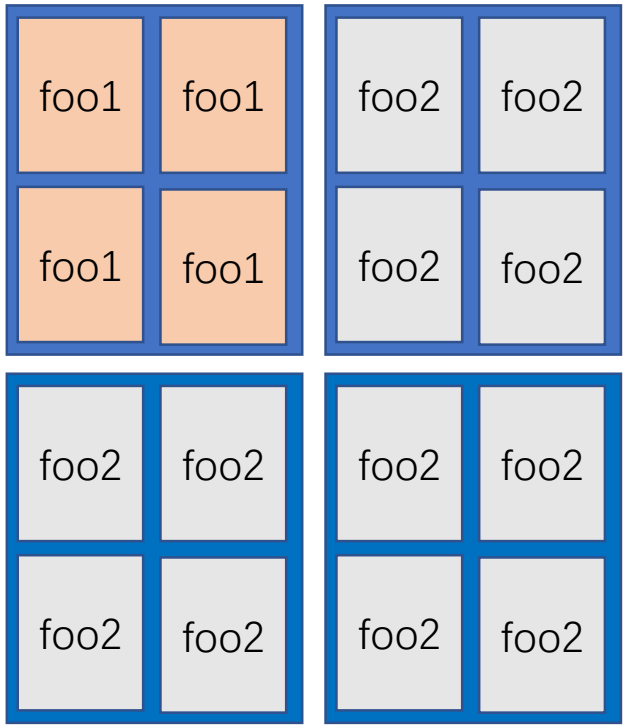
每个Tile运行不同的code的模式

launch_ipu_kernel(tid, func_name1, num_param1s, param1s, …
tid, func_name1, num_param2s, param2s, ….)

launch_ipu_kernel(tid_mask, func_name, params, …)

launch_ipu_kernel(0xF, foo1, a, b, 1)

launch_ipu_kernel(0xFFF0, foo2, a, b, 1)



Example: Host Code

- **Allocate host memory**
  - unsigned int numBytes = N * sizeof(float);
  - float* h_A = (float*) malloc(numBytes);
- **Allocate device memory**
  - float* d_A = 0;
  - HmMalloc((void**)&d_A, numbytes);
- **Copy data from host to device**
  - HmMemcpy(d_A, h_A, numBytes, cudaMemcpyHostToDevice);
- **Execute the kernel**
  - function<<< core_id, tile_id>>>(d_A, b);
- **Copy data from device back to host**
  - HmMemcpy(h_A, d_A, numBytes, MemcpyDeviceToHost); // free device memory
  - HmFree(d_A);

Example: axpy

HOST CODE

```
#include "hdpl/hm_compile_module.h"
#include <math.h>
#include <vector>

int main() {
    int n = 1000;
    auto a_host = (float*)malloc(n * sizeof(float));
    auto b_host = (float*)malloc(sizeof(float));
    void* a_dev = NULL;
    hm_malloc(&a_dev, n * sizeof(float), DEVICE_MEM);
    void* b_dev = NULL;
    hm_malloc(&b_dev, sizeof(float), DEVICE_MEM);
    hm_memcpy(a_dev, a_host, n * sizeof(float), HOST_TO_DEVICE);

    // Create program module from kernel source
    auto module = hm_create_module("axpy.hu");
    int cluster_num = 1;

    // Launch cluster module
    int ret = hm_launch_cluster_module(module, 1, 1, a_dev, b_dev, n);
    int ret = hm_launch_module(module, 1, 1, a_dev, b_dev, n);
    hm_memcpy(b_host, b_dev, sizeof(float), DEVICE_TO_HOST);
    return 0;
} « end main »
```

Device CODE

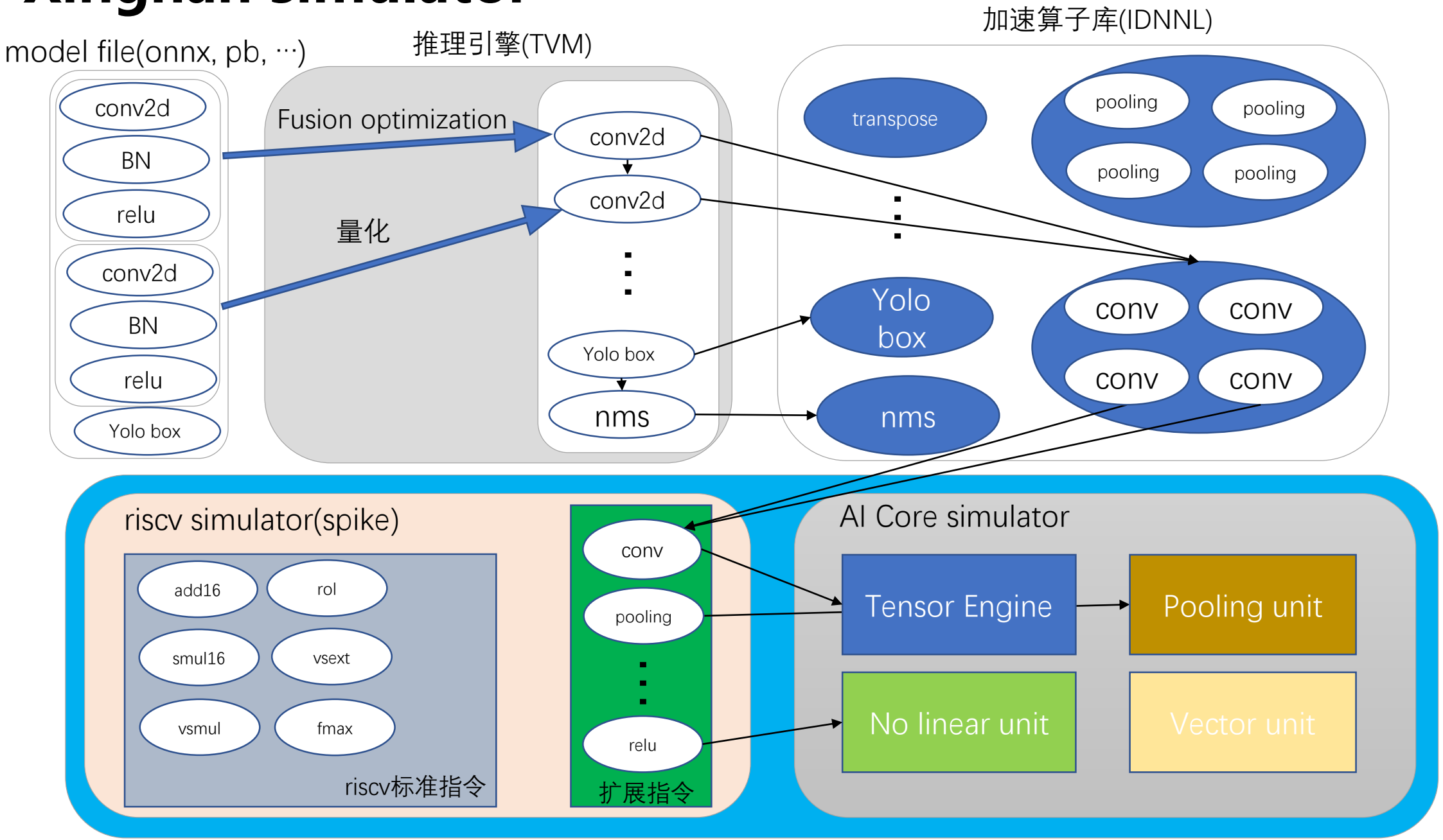
```
// y = a * x + b
__global__ void axpby(float *y, const float *x,
    int loop_count, float a,
    float b) {
  int tid = tile_id();
  int ntiles = tile_num();
  int coreid = core_id();
  int thread_id = coreid * ntiles + tid;
  if (tid >= ntiles) {
    return;
  }
  const int buf_size = BATCH_SIZE;
  int offset = thread_id * buf_size;
  for (int loop = 0; loop < loop_count; loop++) {
    for (int i = 0; i < buf_size / 16; i++) {
      __slocal__ float32x16_t local_x[1];
      __slocal__ float32x16_t local_y[1];
      GM2SLM(x + offset + i * 16, local_x, 16 * sizeof(float));
      local_y[0] = a * local_x[0] + b;
      SLM2GM(local_y, y + offset + i * 16, 16 * sizeof(float));
    }
  }
  return;
} « end axpby »
```

Calling convention: 传参机制的设计

- host device
- device和device

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Xinghan simulator



Models from other framework

TVM framework

TVM Model converter

Relay IR

IPU graph runtime

IPU simulator

conv2d

pooling

HDPL implemented ops

cim

sfu

AI Core simulator

core

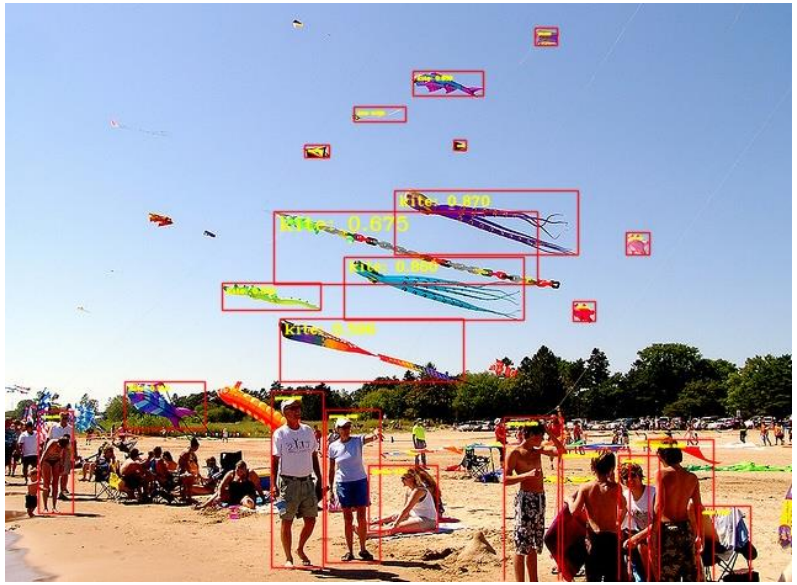
core

core

core

riscv simulator

```
__global__ void multiclass_nms_unmerged_xpu(int class_num, int offset, int background_label, int topk_nms,
        int keep_top_k, bool normalized, float nms_threshold, float nms_eta,
        float score_threshold, const float* bbox, const float* score_nms,
        const int* score_index, int out_dim, float* out, int* out_index) {
    int num_nmsed = 0;
    const int MAX_KEEP_NUM = 100;// max detection num per image
    const int MAX_CLASS_NUM = 80;
    const int MAX_NMS_NUM = 512;// max candidate box to nms, be limited by topk kernel
    if ((keep_top_k > MAX_KEEP_NUM) || (class_num > MAX_CLASS_NUM)
            || (topk_nms > MAX_NMS_NUM) || (out_dim > 6)) {
        return;
    }

    __local__ float lm_scores[MAX_NMS_NUM];
    __local__ int lm_index[MAX_NMS_NUM];
    __local__ float lm_box[MAX_NMS_NUM * 4];//[xmin, ymin, xmax, ymax]
    __local__ float selected[MAX_KEEP_NUM * 6];//[label, score, 4 coordinates]
    // nmsed number for one class with it's indices
    __local__ int selected_index[1 + MAX_KEEP_NUM];
    int cid = core_id();
    int ncores = core_num();
    int num_cluster = cluster_num();
    if (cid >= ncores) {
        return;
    }
    // printf("----------------- multiclass_nms_unmerged_xpu %d %d-----------\n", num_cluster, ncores);
    // printf("------------- __global__ multiclass_nms_unmerged_xpu %d %d--------\n", cid, ncores);
    int total_thread = ncores * num_cluster;
    int thread_id = cid * num_cluster + cluster_id();
    for (int c = thread_id; c < class_num; c += total_thread) {
        if (c == background_label) {
            selected_index[0] = 0;
            LM2GM(selected_index, out_index + c * (1 + keep_top_k), sizeof(int));
            continue;
        }
        GM2LM(score_nms + c * topk_nms, lm_scores, topk_nms * sizeof(float));
        if (lm_scores[0] <= score_threshold) {
            selected_index[0] = 0;
            LM2GM(selected_index, out_index + c * (1 + keep_top_k), sizeof(int));
            continue;
        }
```



Xinghan Simulator

- Spike作为RISCV的功能模拟器
  - 多Core和多Tile的支持
    - Thread safe支持
  - 支持AI Core扩展指令
    - 比如增加vadd_vv，需要 在riscv/insns目录下增加vadd_vv.h

```
// vadd.vv vd, vs1, vs2, vm
VI_VV_LOOP
({
  vd = vs1 + vs2;
})
```

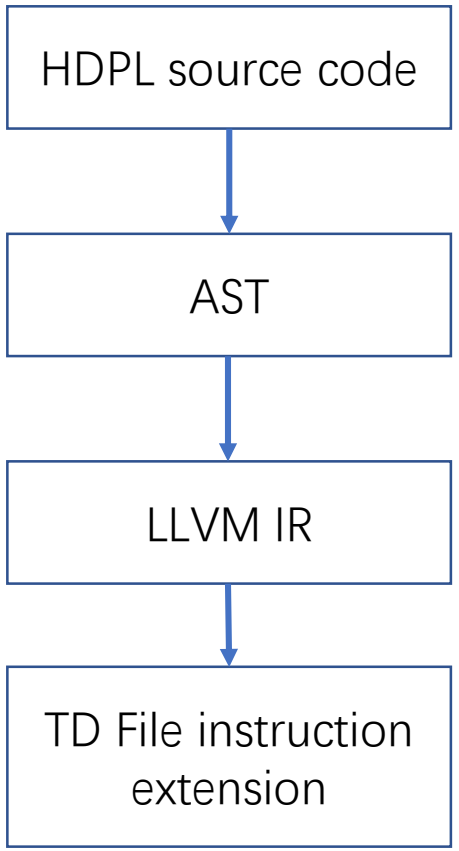
- AI Core功能模拟
- 支持用户开发模型，算子和调试
  - printf的支持

XingHan Runtime

- Device management:
  - hmGetDeviceProperties()
- Device memory management:
  - hmMalloc(), hmFree(), hmMemcpy()

IPU LLVM

- 增加扩展寄存器
- 增加扩展指令



图编译器

Models from other framework

TVM framework

TVM Model converter

Relay IR

framework

HDPL IR

graph optimizer

优化算子库 | code generator

graph runtime

一般编译器解决方案

PaddlePaddle

Framework Frontend

Target dependent optimization

Target codegen

PYTORCH

Framework Frontend

Target dependent optimization

Target codegen

TensorFlow

Framework Frontend

Target dependent optimization

Target codegen

ONNX

Framework Frontend

Target dependent optimization

Target codegen

GIT lab

- IPU LLVM
  git@10.10.1.13:toolchain/illvm.git
- AI Core机器学习加速库(IDNNL)
  git@10.10.1.13:toolchain/idnnl.git
- IPU运行时编译库
  git@10.10.1.13:toolchain/hmjitc.git
- 星汉模拟器
  git@10.10.1.13:toolchain/hm_sim.git
- 星汉加速推理引擎
  git@10.10.1.13:toolchain/itvm.git

工具链SDK1.0项目时间点