中国科学院大学计算机学院专业选修课（硕博通用课程）

GPU架构与编程

第七课：GPU架构（一）

赵地
中科院计算所
2024年秋季学期

讲授内容

- GPU编程补充知识：AMD ROCm编程
- 基础知识
- 编程模型
- GPGPU-Sim简介
- 课程大作业（补充）

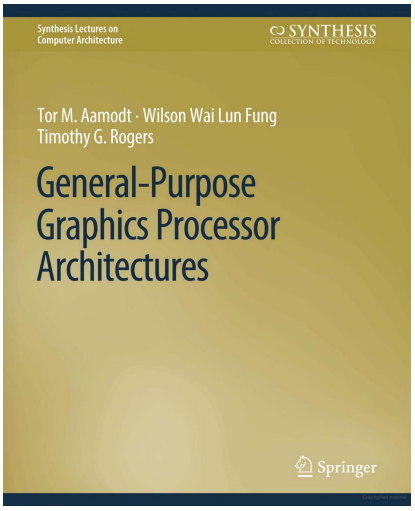
GPU编程补充知识：AMD ROCm编程

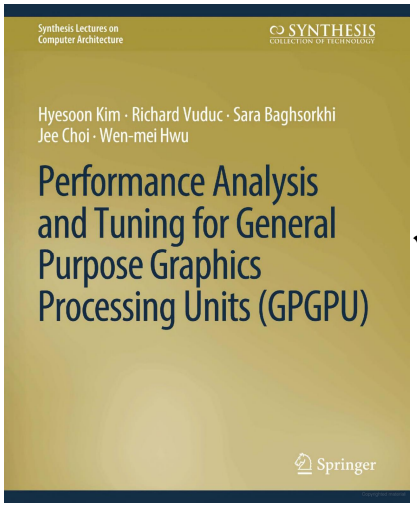
- 2024年“GPU架构与编程”课堂讲座
  - 题目：介绍AMD ROCm
  - 演讲人：陈雯，AMD高级工程师
  - 时间：2024年10月23日 18点10分至19点10分
  - 地点：腾讯会议（国科大在线）

讲授内容

- GPU编程补充知识：AMD ROCm编程
- 基础知识
- 编程模型
- GPGPU-Sim简介
- 课程大作业（补充）

参考书：GPU架构

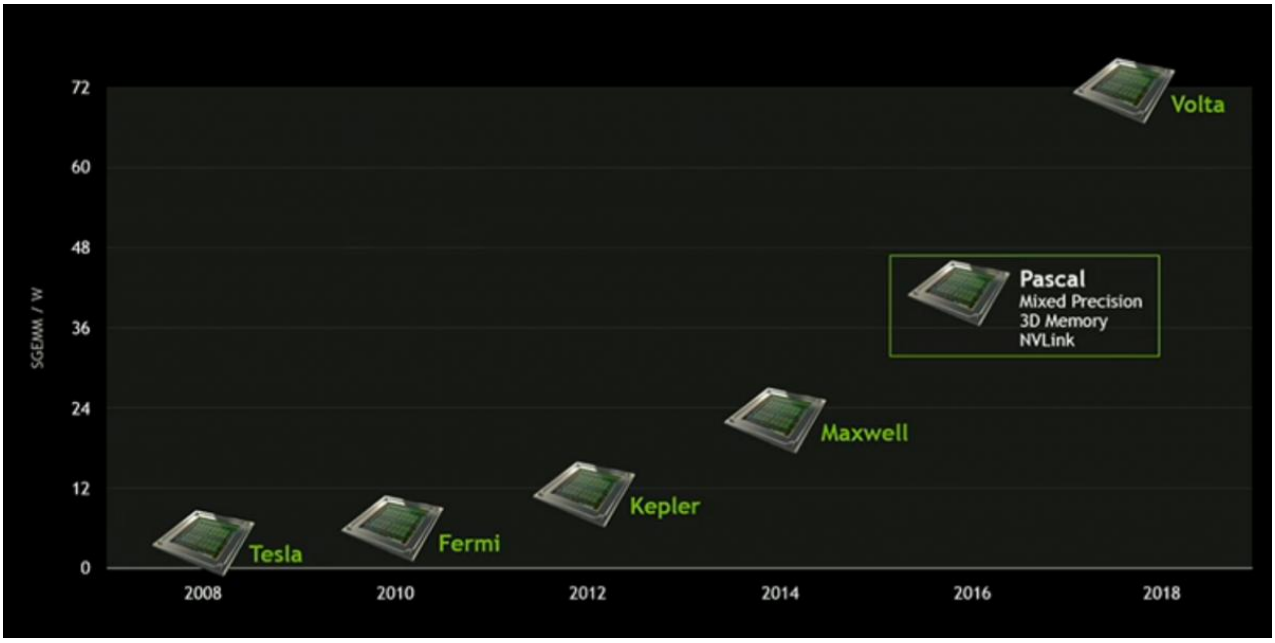




✓**教材：Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.**

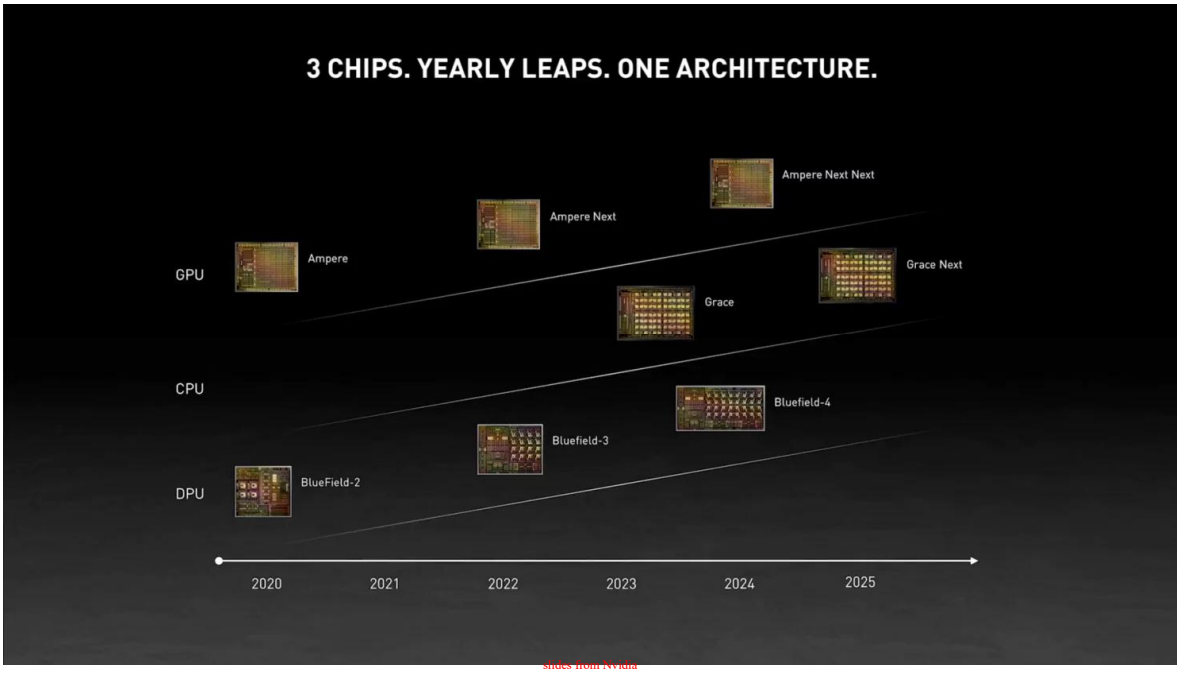
✓参考书：Hyesoon Kim, Richard Vuduc, Sara Baghsorkhi, Jee Choi, and Wen-mei Hwu. 2012. Performance Analysis and Tuning for General Purpose Graphics Processing Units (1st. ed.). Morgan & Claypool Publishers.

GPU的发展史（英伟达GPU为例）



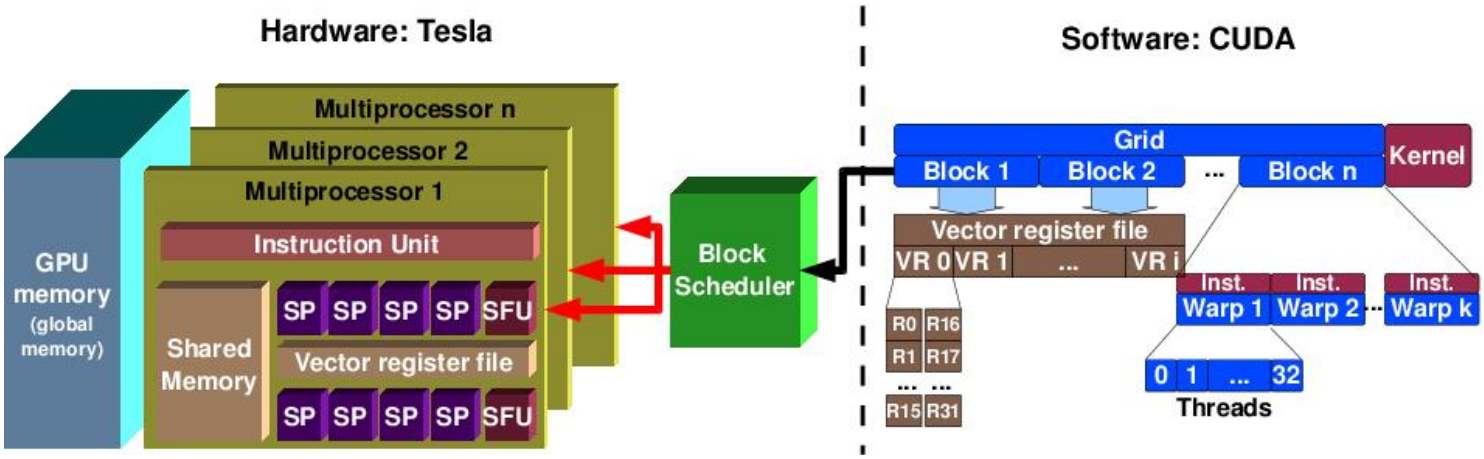
slides from Nvidia

GPU的发展史（英伟达GPU为例）



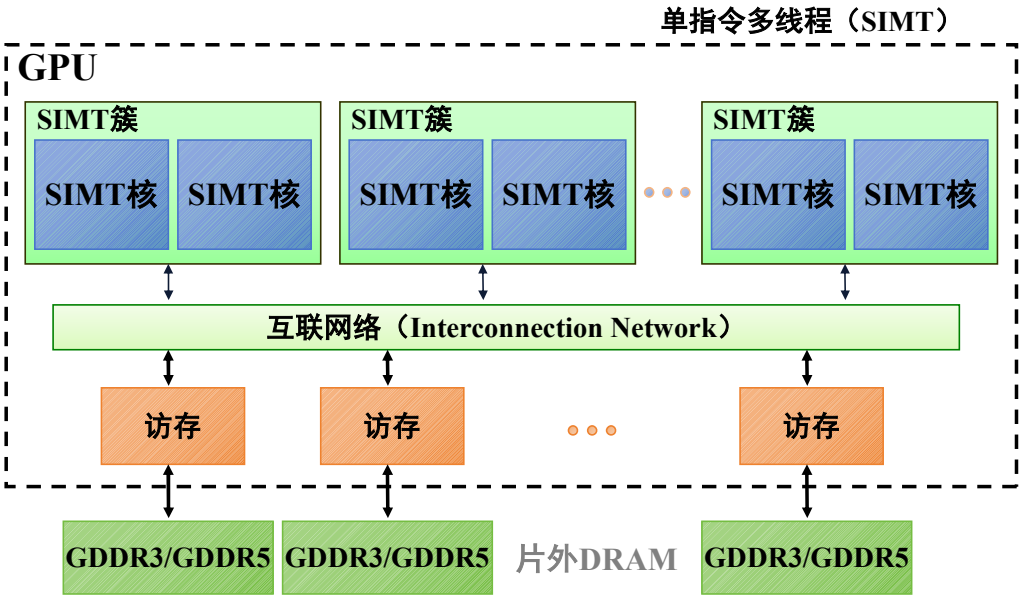
Slides from Nvidia

总体构架：从CUDA到GPU



C. Collange, M. Daumas, D. Defour and D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami Beach, FL, USA, 2010, pp. 351-360, doi: 10.1109/MASCOTS.2010.43.

总体构架：GPU架构（GPGPU-Sim）



http://www.gpgpu-sim.org/

- ✓SIMT核
  - ✓SIMT流水线（SIMT Frontend）
  - ✓缓存
- ✓片上网络（Interconnection Network）
- ✓时钟
- ✓缓存与访存

Performance between Multicore and Multithreaded Architectures

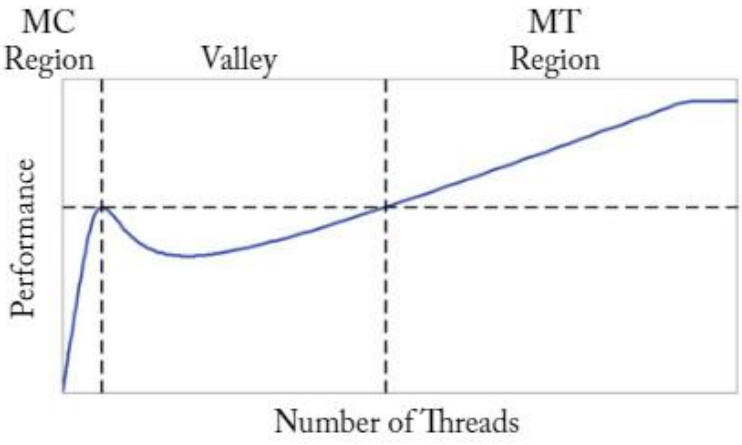


Figure 1.3: An analytical model-based analysis of the performance tradeoff between multicore (MC) CPU architectures and multithreaded (MT) architectures such as GPUs shows a "performance valley" may occur if the number of threads is insufficient to cover off-chip memory access latency (based on Figure 1 from Guz et al. [2009]).

Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.

Energy Consumption

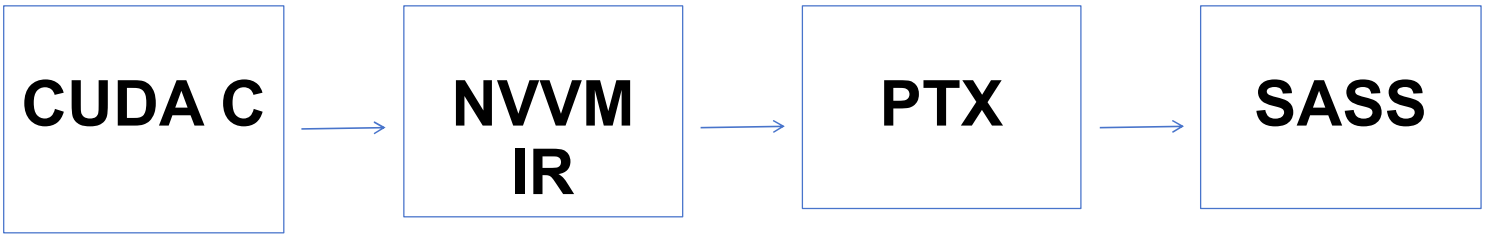
Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit 32KB SRAM	5	50
32 bit DRAM	640	6400

Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.

讲授内容

- GPU编程补充知识：AMD ROCm编程
- 基础知识
- 编程模型
- GPGPU-Sim简介
- 课程大作业（补充）

CUDA Compiling



- CUDA C: Compute Unified Device Architecture
- NVVM IR: NVVM: NVidia LLVM IR
- PTX: Parallel Thread Execution, or NVPTX
- SASS: Streaming ASSembler, Shader ASSembly

NVCC: Nvidia CUDA Compiler

- Nvidia CUDA Compiler (NVCC) is a proprietary compiler by Nvidia intended for use with CUDA.
- NVCC separates these two parts and sends host code (the part of code which will be run on the CPU) to a C compiler like GCC or Intel C++ Compiler (ICC) or Microsoft Visual C++ Compiler.
- NVCC sends the device code (the part which will run on the GPU) to the GPU. The device code is further compiled by NVCC.
- NVCC is based on LLVM. https://en.wikipedia.org/wiki/Nvidia_CUDA_Compiler

NVCC: Nvidia CUDA Compiler

- NVCC is a compiler driver which works by invoking all the necessary tools and compilers like cudacc, g++, cl, etc.
- NVCC can output either C code (CPU Code) that must then be compiled with the rest of the application using another tool or PTX or object code directly.
- An executable with CUDA code requires: the CUDA core library (cuda) and the CUDA runtime library (cudart).

https://en.wikipedia.org/wiki/Nvidia_CUDA_Compiler

PTX：并行线程执行

- ✓并行线程执行（Parallel Thread eXecution，PTX）是英伟达公司使用的指令集；
- ✓PTX是伪汇编指令集：不在硬件上直接执行；
- ✓ptxas是NVIDIA发布的汇编器，将PTX汇编成在硬件运行的指令（SASS）；
- ✓每一代硬件都支持不同版本的SASS，PTX在编译时被编译成多个版本的SASS，对应不同版本的硬件。

PTX：并行线程执行

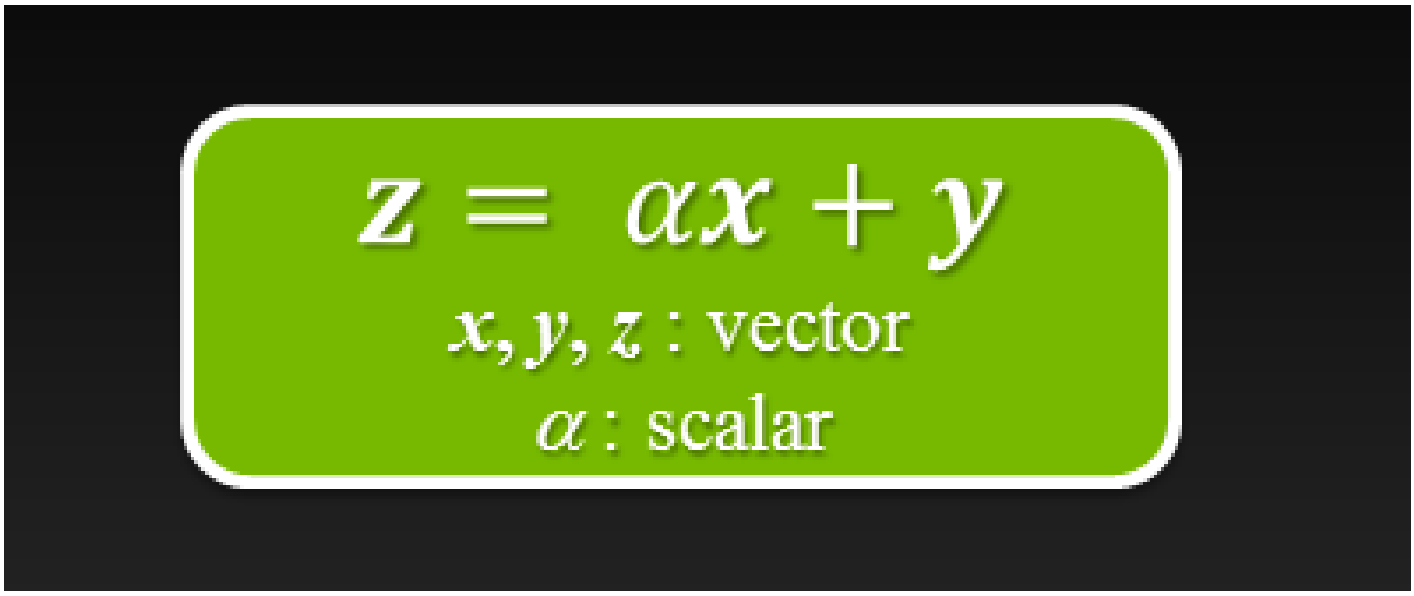
- ✓PTX代码仍然嵌入到二进制文件中，以支持未来的硬件；
- ✓在运行时，运行时系统根据可用硬件选择合适版本的SASS运行；
- ✓如果没有，运行时系统（the runtime system）会调用嵌入式PTX上的即时(just-in-time，JIT) 编译器，将PTX代码编译为与可用硬件相对应的SASS。

SASS

- ✓**The assembly code used for NVIDIA devices is known as ”SASS”, which corresponds exactly to the actual binary code.**
- ✓**Most details of SASS are kept secret.**
- ✓**In addition to the closed-source compiler, nvcc, NVIDIA provides a closed-source disassembler called cuobjdump, which can translate the binary into SASS assembly code.**
- ✓**However, NVIDIA offers no means of translating the SASS to binary, leaving researchers unable to make use of the assembly code.**

Ari B. Hayes, Fei Hua, Jin Huang, Yanhao Chen, Eddy Z. Zhang, Decoding CUDA Binary, CGO 2019, 229-241.

Example: SAXPY computation



https://developer.nvidia.com/blog/six-ways-saxpy/

CPU version of SAXPY

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
            y[i] = a*x[i] + y[i];
}
main() {
    float *x, *y;
    int n;
    // omitted: allocate CPU memory for x and y and
initialize contents
    saxpy_serial(n, 2.0, x, y); // Invoke serial SAXPY
kernel
    // omitted: use y on CPU, free memory pointed to by x
and y
}
```

https://developer.nvidia.com/blog/six-ways-saxpy/

CUBLAS version of SAXPY

```
int N = 1<<20;
cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);
cublasShutdown();
```

https://developer.nvidia.com/blog/six-ways-saxpy/

OpenACC version of SAXPY

```
void saxpy(int n, float a, float *
restrict x, float * restrict y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
      y[i] = a*x[i] + y[i];
}
...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
```

https://developer.nvidia.com/blog/six-ways-saxpy/

Thrust version of SAXPY

```
using thrust::placeholders;

int N = 1<<20;
thrust::host_vector x(N), y(N);
...
thrust::device_vector d_x = x;
// alloc and copy host to device
thrust::device_vector d_y = y;
// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
d_y.begin(), d_y.begin(), 2.0f * _1 + _2);
y = d_y; // copy results to the host vector
```

https://developer.nvidia.com/blog/six-ways-saxpy/

CUDA version of SAXPY

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
      int i = blockIdx.x*blockDim.x + threadIdx.x;
      if(i<n)
           y[i] = a*x[i] + y[i];
}
int main() {
      float *h_x, *h_y;
      int n;
      // omitted: allocate CPU memory for h_x and h_y and initialize contents
      float *d_x, *d_y;
      int nblocks = (n + 255) / 256;
      cudaMalloc( &d_x, n * sizeof(float) );
      cudaMalloc( &d_y, n * sizeof(float) );
      cudaMemcpy( d_x, h_x, n * sizeof(float), cudaMemcpyHostToDevice );
      cudaMemcpy( d_y, h_y, n * sizeof(float), cudaMemcpyHostToDevice );
      saxpy<<<nblocks, 256>>>(n, 2.0, d_x, d_y);
      cudaMemcpy( h_x, d_x, n * sizeof(float), cudaMemcpyDeviceToHost );
      // omitted: use h_y on CPU, free memory pointed to by h_x, h_y, d_x, and d_y
}
```

https://developer.nvidia.com/blog/six-ways-saxpy/

PTX version of SAXPY

```
.visible .entry _Z5saxpyifPfS_(
.param .u32 _Z5saxpyifPfS__param_0,
.param .f32 _Z5saxpyifPfS__param_1,
.param .u64 _Z5saxpyifPfS__param_2,
.param .u64 _Z5saxpyifPfS__param_3
)
{
.reg .pred %p<2>;
.reg .f32 %f<5>;
.reg .b32 %r<6>;
.reg .b64 %rd<8>;

ld.param.u32 %r2,
[_Z5saxpyifPfS__param_0];
ld.param.f32 %f1,
[_Z5saxpyifPfS__param_1];
ld.param.u64 %rd1,
[_Z5saxpyifPfS__param_2];
ld.param.u64 %rd2,
[_Z5saxpyifPfS__param_3];
mov.u32 %r3, %ctaid.x;
mov.u32 %r4, %ntid.x;
mov.u32 %r5, %tid.x;
mad.lo.s32 %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra BB0_2;

cvta.to.global.u64 %rd3, %rd2;
cvta.to.global.u64 %rd4, %rd1;
mul.wide.s32 %rd5, %r1, 4;
add.s64 %rd6, %rd4, %rd5;
ld.global.f32 %f2, [%rd6];
add.s64 %rd7, %rd3, %rd5;
ld.global.f32 %f3, [%rd7];
fma.rn.f32 %f4, %f2, %f1, %f3;
st.global.f32 [%rd7], %f4;

BB0_2:
ret;
}
```

Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.

SASS version of SAXPY (NVIDIA Fermi Architecture)

```
Address     Dissassembly                                    Encoded Instruction
========    ============================================    ======================
/*0000*/         MOV R1, c[0x1][0x100];                    /* 0x2800440400005de4 */
/*0008*/         S2R R0, SR_CTAID.X;                       /* 0x2c00000094001c04 */
/*0010*/         S2R R2, SR_TID.X;                         /* 0x2c00000084009c04 */
/*0018*/         IMAD R0, R0, c[0x0][0x8], R2;             /* 0x2004400020001ca3 */
/*0020*/         ISETP.GE.AND P0, PT, R0, c[0x0][0x20], PT; /* 0x1b0e40008001dc23 */
/*0028*/     @P0 BRA.U 0x78;                               /* 0x40000001200081e7 */
/*0030*/    @!P0 MOV32I R5, 0x4;                           /* 0x18000000100161e2 */
/*0038*/    @!P0 IMAD R2.CC, R0, R5, c[0x0][0x28];         /* 0x200b8000a000a0a3 */
/*0040*/    @!P0 IMAD.HI.X R3, R0, R5, c[0x0][0x2c];       /* 0x208a8000b000e0e3 */
/*0048*/    @!P0 IMAD R4.CC, R0, R5, c[0x0][0x30];         /* 0x200b8000c00120a3 */
/*0050*/    @!P0 LD.E R2, [R2];                            /* 0x840000000020a085 */
/*0058*/    @!P0 IMAD.HI.X R5, R0, R5, c[0x0][0x34];       /* 0x208a8000d00160e3 */
/*0060*/    @!P0 LD.E R0, [R4];                            /* 0x8400000000402085 */
/*0068*/    @!P0 FFMA R0, R2, c[0x0][0x24], R0;            /* 0x3000400090202000 */
/*0070*/    @!P0 ST.E [R4], R0;                            /* 0x9400000000402085 */
/*0078*/         EXIT;                                     /* 0x8000000000001de7 */
```

Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.

SASS version of SAXPY (NVIDIA Pascal Architecture)

```
Address     Dissassembly                                  Encoded Instruction
=======     ============================================  ====================
                                                          /* 0x001c7c00e22007f6 */
/*0008*/       MOV R1, c[0x0][0x20];                      /* 0x4c98078000870001 */
/*0010*/       S2R R0, SR_CTAID.X;                        /* 0xf0c8000002570000 */
/*0018*/       S2R R2, SR_TID.X;                          /* 0xf0c8000002170002 */
                                                          /* 0x001fd840fec20ff1 */
/*0028*/       XMAD.MRG R3, R0.reuse, c[0x0] [0x8].H1, RZ; /* 0x4f107f8000270003 */
/*0030*/       XMAD R2, R0.reuse, c[0x0] [0x8], R2;       /* 0x4e0001000270002 */
/*0038*/       XMAD.PSL.CBCC R0, R0.H1, R3.H1, R2;        /* 0x5b30011800370000 */
                                                          /* 0x081fc400ffa007ed */
/*0048*/       ISETP.GE.AND P0, PT, R0, c[0x0][0x140], PT; /* 0x4b6d038005070007 */
/*0050*/    @P0 EXIT;                                     /* 0xe30000000000000f */
/*0058*/       SHL R2, R0.reuse, 0x2;                     /* 0x3848000000270002 */
                                                          /* 0x081fc440fec007f5 */
/*0068*/       SHR R0, R0, 0x1e;                          /* 0x3829000001e70000 */
/*0070*/       IADD R4.CC, R2.reuse, c[0x0][0x148];       /* 0x4c1088000527020 4 */
/*0078*/       IADD.X R5, R0.reuse, c[0x0][0x14c];        /* 0x4c1008000537000 5 */
                                                          /* 0x0001c800fe0007f6 */
/*0088*/       IADD R2.CC, R2, c[0x0][0x150];             /* 0x4c1080005470202 */
/*0090*/       IADD.X R3, R0, c[0x0][0x154];              /* 0x4c1008005570003 */
/*0098*/       LDG.E R0, [R4];        }                   /* 0xeed4200000070400 */
                                                          /* 0x0007c408fc400172 */
/*00a8*/       LDG.E R6, [R2];                            /* 0xeed4200000070206 */
/*00b0*/       FFMA R0, R0, c[0x0][0x144], R6;            /* 0x4980030005170000 */
/*00b8*/       STG.E [R2], R0;                            /* 0xeedc200000070200 */
                                                          /* 0x001f8000ffe007ff */
/*00c8*/       EXIT;                                      /* 0xe30000000007000f */
/*00d0*/       BRA 0xd0;                                  /* 0xe2400fffff87000f */
/*00d8*/       NOP;                                       /* 0x50b0000000070f00 */
                                                          /* 0x001f8000fc0007e0 */
/*00e8*/       NOP;                                       /* 0x50b0000000070f00 */
/*00f0*/       NOP;                                       /* 0x50b0000000070f00 */
/*00f8*/       NOP;                                       /* 0x50b0000000070f00 */
```

For M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers. 2018. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers.

OpenCL version of SAXPY

```
int i;
// Allocate space for vectors A, B and C
float alpha = 2.0;
float *A =
(float*)malloc(sizeof(float)*VECTOR_SIZE);
float *B =
(float*)malloc(sizeof(float)*VECTOR_SIZE);
float *C =
(float*)malloc(sizeof(float)*VECTOR_SIZE);
for(i = 0; i < VECTOR_SIZE; i++)
{
    A[i] = i;
    B[i] = VECTOR_SIZE - i;
    C[i] = 0;
}
```

```
// Get platform and device information
cl_platform_id * platforms = NULL;
cl_uint     num_platforms;
//Set up the Platform
cl_int clStatus = clGetPlatformIDs(0,
NULL, &num_platforms);
platforms = (cl_platform_id *)

malloc(sizeof(cl_platform_id)*num_platf
orms);
clStatus =
clGetPlatformIDs(num_platforms,
platforms, NULL);
```

https://developer.nvidia.com/blog/six-ways-saxpy/

OpenCL version of SAXPY

```
//Get the devices list and choose the
device you want to run on
cl_device_id     *device_list = NULL;
cl_uint           num_devices;

clStatus =
clGetDeviceIDs( platforms[0],
CL_DEVICE_TYPE_GPU, 0,NULL,
&num_devices);
device_list = (cl_device_id *)
malloc(sizeof(cl_device_id)*num_device
s);
clStatus =
clGetDeviceIDs
( platforms[0],CL_DEVICE_TYPE_GPU,
num_devices, device_list, NULL);
// Create one OpenCL context for each
device in the platform
cl_context context;
context = clCreateContext( NULL,
num_devices, device_list, NULL, NULL,
&clStatus);

// Create a command queue
cl_command_queue command_queue =
clCreateCommandQueue(context,
device_list[0], 0, &clStatus);

// Create memory buffers on the device for
each vector
cl_mem A_clmem = clCreateBuffer(context,
CL_MEM_READ_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);
cl_mem B_clmem = clCreateBuffer(context,
CL_MEM_READ_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);
cl_mem C_clmem = clCreateBuffer(context,
CL_MEM_WRITE_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);
```

https://developer.nvidia.com/blog/six-ways-saxpy/

OpenCL version of SAXPY

```
// Copy the Buffer A and B to the device
clStatus =
clEnqueueWriteBuffer(command_queue, A_clmem,
CL_TRUE, 0, VECTOR_SIZE * sizeof(float), A,
0, NULL, NULL);
clStatus =
clEnqueueWriteBuffer(command_queue, B_clmem,
CL_TRUE, 0, VECTOR_SIZE * sizeof(float), B,
0, NULL, NULL);
// Create a program from the kernel source
cl_program program =
clCreateProgramWithSource(context, 1,(const
char **)&saxpy_kernel, NULL, &clStatus);
// Build the program
clStatus = clBuildProgram(program, 1,
device_list, NULL, NULL, NULL);
// Create the OpenCL kernel
cl_kernel kernel = clCreateKernel(program,
"saxpy_kernel", &clStatus);
// Set the arguments of the kernel
clStatus = clSetKernelArg(kernel, 0,
sizeof(float), (void *)&alpha);
clStatus = clSetKernelArg(kernel, 1,
sizeof(cl_mem), (void *)&A_clmem);
clStatus = clSetKernelArg(kernel, 2,
sizeof(cl_mem), (void *)&B_clmem);
clStatus = clSetKernelArg(kernel, 3,
sizeof(cl_mem), (void *)&C_clmem);
// Execute the OpenCL kernel on the
list
size_t global_size = VECTOR_SIZE; //
Process the entire lists
size_t local_size = 64;           //
Process one item at a time
clStatus =
clEnqueueNDRangeKernel(command_queue,
kernel, 1, NULL, &global_size,
&local_size, 0, NULL, NULL);
```

https://developer.nvidia.com/blog/six-ways-saxpy/

OpenCL version of SAXPY

```
// Read the cl memory C_clmem on device
to the host variable C
clStatus =
clEnqueueReadBuffer(command_queue,
C_clmem, CL_TRUE, 0, VECTOR_SIZE *
sizeof(float), C, 0, NULL, NULL);

// Clean up and wait for all the comands
to complete.
clStatus = clFlush(command_queue);
clStatus = clFinish(command_queue);

// Display the result to the screen
for(i = 0; i < VECTOR_SIZE; i++)
    printf("%f * %f + %f = %f\n", alpha,
A[i], B[i], C[i]);
```

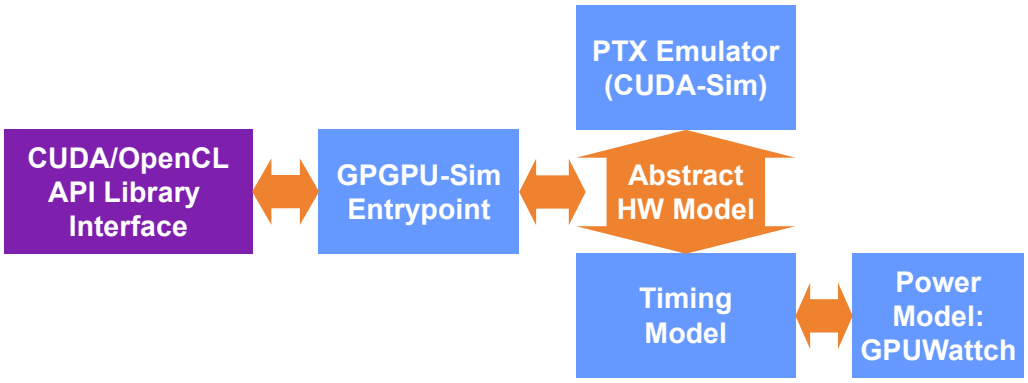
```
// Finally release all OpenCL allocated
objects and host buffers.
clStatus = clReleaseKernel(kernel);
clStatus = clReleaseProgram(program);
clStatus = clReleaseMemObject(A_clmem);
clStatus = clReleaseMemObject(B_clmem);
clStatus = clReleaseMemObject(C_clmem);
clStatus =
clReleaseCommandQueue(command_queue);
clStatus = clReleaseContext(context);
free(A);
free(B);
free(C);
free(platforms);
free(device_list);
return 0;
```

https://developer.nvidia.com/blog/six-ways-saxpy/

讲授内容

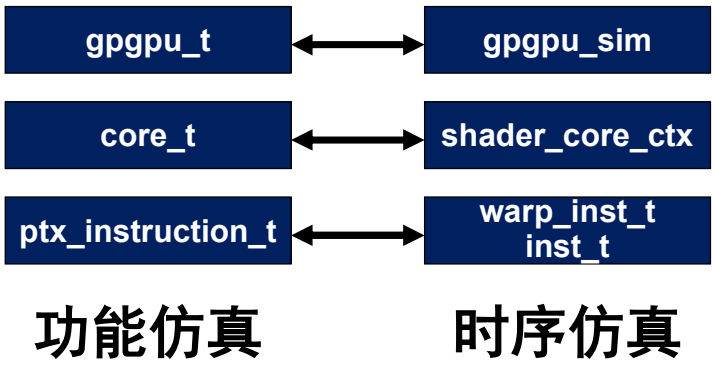
- GPU编程补充知识：AMD ROCm编程
- 基础知识
- 编程模型
- GPGPU-Sim简介
- 课程大作业（补充）

GPGPU-sim的整体结构



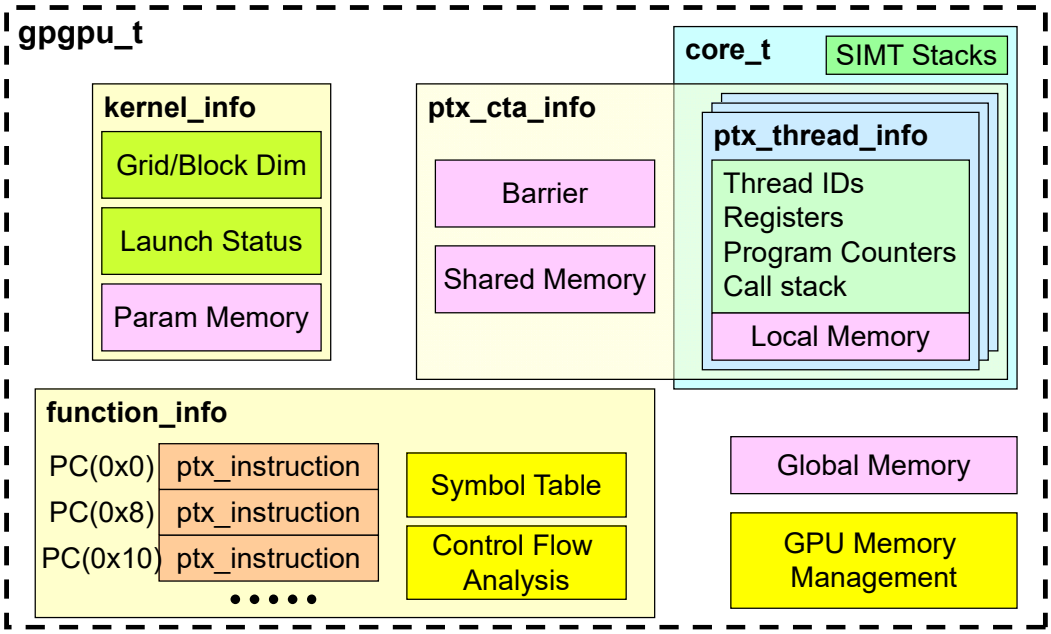
http://www.gpgpu-sim.org/

GPGPU-sim的功能仿真与时序仿真



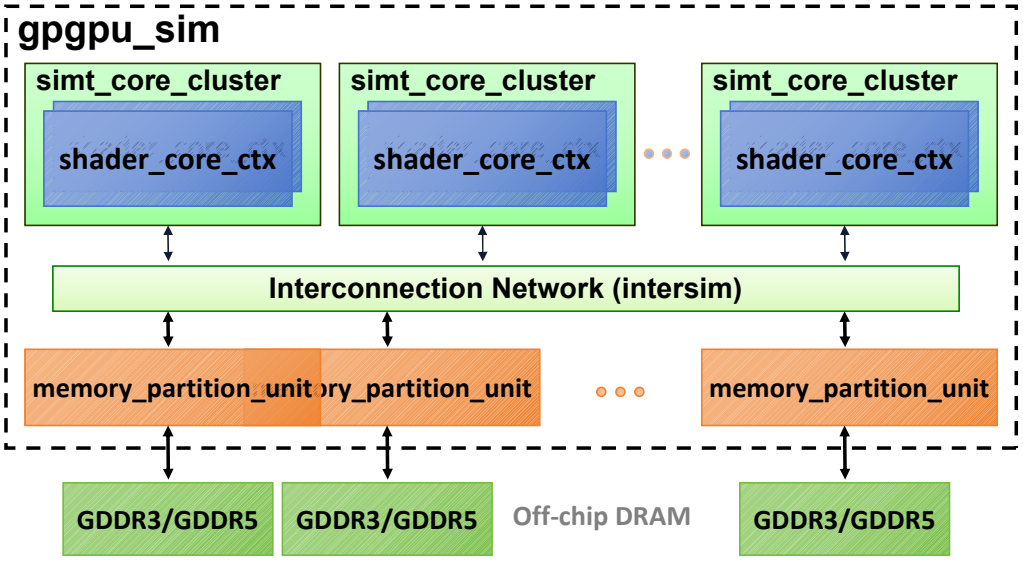
http://www.gpgpu-sim.org/

GPGPU-sim的功能仿真



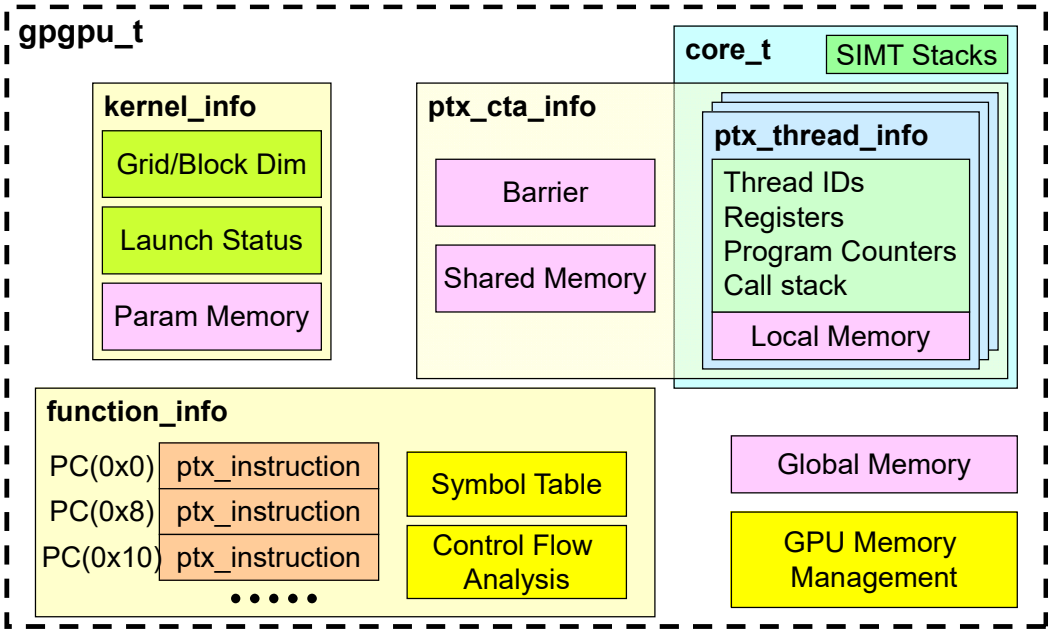
http://www.gpgpu-sim.org/

GPGPU-sim的时序仿真



http://www.gpgpu-sim.org/

功能仿真的主要模块



讲授内容：GPGPU-Sim简介

I. 模拟仿真模式
II. Debugging
III. 配置
IV. 输出信息
V. 可视化

模拟仿真模式

- ✓性能模拟模式（Performance Simulation）：使用 GPGPU-Sim来评估运行应用程序所需的GPU时钟周期数（GPU clock cycle）；
- ✓功能仿真模式（Functional Simulation）：对新的应用程序（application）进行模拟仿真；功能模拟模式将程序利用一组warp来执行，其中每个协作线程阵列（CTA）的所有warp都被执行，直到这些warp全部完成或全部处于barrier等待；在全部处于barrier等待的情况下，一旦所有warp在barrier处相遇，这些warp就会获准穿过barrier。

http://www.gpgpu-sim.org/

Debugger模式

- ✓在性能模拟模式（Performance Simulation）下，交互式调试器模式（Interactive debugger mode）提供了一个类似GDB的接口，用于调试GPGPU-Sim中的功能行为。

http://www.gpgpu-sim.org/

Cuobjdump支持

- ✓cuobjdump是NVidia提供的一款软件，用于从二进制文件中提取 SASS 和 PTX 等信息；
- ✓GPGPU-Sim支持使用cuobjdump提取运行SASS或PTX所需的信息。

http://www.gpgpu-sim.org/

讲授内容：GPGPU-Sim简介

I. 模拟仿真模式
II. Debugging
III. 配置
IV. 输出信息
V. 可视化

Debugging

✓环境变量：通过环境变量，GPGPU-Sim可以配置某些与调试相关的行为；调试时，生成有关模拟器（simulator）运行状态的附加信息，并打印出来；

✓debug tracing。

http://www.gpgpu-sim.org/

讲授内容：GPGPU-Sim简介

I. 模拟仿真模式
II. Debugging
III. 配置
IV. 输出信息
V. 可视化

配置

- ✓Simulation Run Configuration
- ✓Statistics Collection Options
- ✓High-Level Architecture Configuration
- ✓Additional Architecture Configuration
- ✓Scheduler
- ✓Shader Core Pipeline Configuration
- ✓Memory Sub-System Configuration
- ✓Operand Collector Configuration
- ✓DRAM/Memory Controller Configuration
- ✓Interconnection Configuration
- ✓PTX Configurations

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# functional simulator specification
-gpgpu_ptx_instruction_classification 0
-gpgpu_ptx_sim_mode 0
-gpgpu_ptx_force_max_capability 70

# Device Limits
-gpgpu_stack_size_limit 1024
-gpgpu_heap_size_limit 8388608
-gpgpu_runtime_sync_depth_limit 2
-gpgpu_runtime_pending_launch_count_limit 2048
-gpgpu_kernel_launch_latency 5000
-gpgpu_TB_launch_latency 0
```

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# Compute Capability
-gpgpu_compute_capability_major 7
-gpgpu_compute_capability_minor 0

# PTX execution-driven
-gpgpu_ptx_convert_to_ptxplus 0
-gpgpu_ptx_save_converted_ptxplus 0

# high level architecture configuration
-gpgpu_n_clusters 80
-gpgpu_n_cores_per_cluster 1
-gpgpu_n_mem 32
-gpgpu_n_sub_partition_per_mchannel 2
```

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# volta clock domains
#-gpgpu_clock_domains <Core Clock>:<Interconnect
Clock>:<L2 Clock>:<DRAM Clock>
-gpgpu_clock_domains 1132.0:1132.0:1132.0:850.0
# boost mode
# -gpgpu_clock_domains 1628.0:1628.0:1628.0:850.0

# shader core pipeline config
-gpgpu_shader_registers 65536
-gpgpu_registers_per_block 65536
-gpgpu_occupancy_sm_number 70

# This implies a maximum of 64 warps/SM
-gpgpu_shader_core_pipeline 2048:32
-gpgpu_shader_cta 32
```

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# Pipeline widths and number of FUs
#
ID_OC_SP,ID_OC_DP,ID_OC_INT,ID_OC_SFU,ID_OC_MEM,OC_
EX_SP,OC_EX_DP,OC_EX_INT,OC_EX_SFU,OC_EX_MEM,EX_WB,
ID_OC_TENSOR_CORE,OC_EX_TENSOR_CORE
## Volta GV100 has 4 SP SIMD units, 4 SFU units, 4
DP units per core, 4 Tensor core units
## we need to scale the number of pipeline
registers to be equal to the number of SP units
-gpgpu_pipeline_widths 4,4,4,4,4,4,4,4,4,4,8,4,4
-gpgpu_num_sp_units 4
-gpgpu_num_sfu_units 4
-gpgpu_num_dp_units 4
-gpgpu_num_int_units 4
-gpgpu_tensor_core_avail 1
-gpgpu_num_tensor_core_units 4
```

配置：SM7_QV100

```
# Instruction latencies and initiation intervals
# "ADD,MAX,MUL,MAD,DIV"
# All Div operations are executed on SFU unit
-ptx_opcode_latency_int 4,13,4,5,145,21
-ptx_opcode_initiation_int 2,2,2,2,8,4
-ptx_opcode_latency_fp 4,13,4,5,39
-ptx_opcode_initiation_fp 2,2,2,2,4
-ptx_opcode_latency_dp 8,19,8,8,330
-ptx_opcode_initiation_dp 4,4,4,4,130
-ptx_opcode_latency_sfu 100
-ptx_opcode_initiation_sfu 8
-ptx_opcode_latency_tesnor 64
-ptx_opcode_initiation_tensor 64
```

配置：SM7_QV100

```
# Volta has sub core model, in which each scheduler has its own register file and EUs
# i.e. schedulers are isolated
-gpgpu_sub_core_model 1
# disable specialized operand collectors and use generic operand collectors instead
-gpgpu_enable_specialized_operand_collector 0
-gpgpu_operand_collector_num_units_gen 8
-gpgpu_operand_collector_num_in_ports_gen 8
-gpgpu_operand_collector_num_out_ports_gen 8
# volta has 8 banks, 4 schedulers, two banks per scheduler
# we increase #banks to 16 to mitigate the effect of Regisrer File Cache (RFC) which we do not implement in the current version
-gpgpu_num_reg_banks 16
-gpgpu_reg_file_port_throughput 2
```


配置：SM7_QV100

```
# shared memory bankconflict detection
-gpgpu_shmem_num_banks 32
-gpgpu_shmem_limited_broadcast 0
-gpgpu_shmem_warp_parts 1
-gpgpu_coalesce_arch 60

# Volta has four schedulers per core
-gpgpu_num_sched_per_core 4
# Greedy then oldest scheduler
-gpgpu_scheduler gto
## In Volta, a warp scheduler can issue 1 inst per cycle
-gpgpu_max_insn_issue_per_warp 1
-gpgpu_dual_issue_diff_exec_units 1
```

配置：SM7_QV100

```
## L1/shared memory configuration
#
<nsets>:<bsize>:<assoc>,<rep>:<wr>:<alloc>:<wr_alloc>:<set_i
ndex_fn>,<mshr>:<N>:<merge>,<mq>:**<fifo_entry>
# ** Optional parameter - Required when mshr_type==Texture
Fifo
# Defualt config is 32KB DL1 and 96KB shared memory
# In Volta, we assign the remaining shared memory to L1
cache
# if the assigned shd mem = 0, then L1 cache = 128KB
# For more info, see https://docs.nvidia.com/cuda/cuda-c-
programming-guide/index.html#shared-memory-7-x
# disable this mode in case of multi kernels/apps execution
-gpgpu_adaptive_cache_config 1
# Volta unified cache has four banks
```


配置：SM7_QV100

```
-gpgpu_l1_banks 4
-gpgpu_cache:dl1  S:1:128:256,L:L:s:N:L,A:256:8,16:0,32
-gpgpu_shmem_size 98304
-gpgpu_shmem_sizeDefault 98304
-gpgpu_shmem_per_block 65536
-gpgpu_gmem_skip_L1D 0
-gpgpu_n_cluster_ejection_buffer_size 32
-gpgpu_l1_latency 20
-gpgpu_smem_latency 20
-gpgpu_flush_l1_cache 1
```

配置：SM7_QV100

```
# 32 sets, each 128 bytes 24-way for each memory sub
partition (96 KB per memory sub partition). This gives
us 6MB L2 cache
-gpgpu_cache:dl2 S:32:128:24,L:B:m:L:P,A:192:4,32:0,32
-gpgpu_cache:dl2_texture_only 0
-gpgpu_dram_partition_queues 64:64:64:64
-gpgpu_perf_sim_memcpy 1
-gpgpu_memory_partition_indexing 2
```

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# 128 KB Inst.
-gpgpu_cache:il1 N:64:128:16,L:R:f:N:L,S:2:48,4
-gpgpu_inst_fetch_throughput 4
# 128 KB Tex
# Note, TEX is deprected in Volta, It is used for
legacy apps only. Use L1D cache instead with .nc
modifier or __ldg mehtod
-gpgpu_tex_cache:l1 N:4:128:256,L:R:m:N:L,T:512:8,128:2
# 64 KB Const
-gpgpu_const_cache:l1 N:128:64:8,L:R:f:N:L,S:2:64,4
-gpgpu_perfect_inst_const_cache 1
```

http://www.gpgpu-sim.org/

配置：SM7_QV100

```
# interconnection
#-network_mode 1
#-inter_config_file config_volta_islip.icnt
# use built-in local xbar
-network_mode 2
-icnt_in_buffer_limit 512
-icnt_out_buffer_limit 512
-icnt_subnets 2
-icnt_flit_size 40
-icnt_arbiter_algo 1
```


配置：SM7_QV100

```
# memory partition latency config
-gpgpu_l2_rop_latency 160
-dram_latency 100

# dram model config
-gpgpu_dram_scheduler 1
-gpgpu_frfcfs_dram_sched_queue_size 64
-gpgpu_dram_return_queue_size 192
```

配置：SM7_QV100

```
# for HBM, three stacks, 24 channles, each (128 bits)
16 bytes width
-gpgpu_n_mem_per_ctrlr 1
-gpgpu_dram_buswidth 16
-gpgpu_dram_burst_length 2
-dram_data_command_freq_ratio 2  # HBM is DDR
-gpgpu_mem_address_mask 1
-gpgpu_mem_addr_mapping
dramid@8;00000000.00000000.00000000.00000000.0000RRRR.R
RRRRRRR.RBBBCCCB.CCCSSSSS
```


配置：SM7_QV100

```
# HBM timing are adopted from hynix JESD235 standered
and nVidia HPCA 2017 paper
(http://www.cs.utah.edu/~nil/pubs/hpca17.pdf)
# Timing for 1 GHZ
# tRRDl and tWTR are missing, need to be added
#-gpgpu_dram_timing_opt
"nbk=16:CCD=1:RRD=4:RCD=14:RAS=33:RP=14:RC=47:
#
CL=14:WL=2:CDLR=3:WR=12:nbkgrp=4:CCDL=2:RTPL=4"

# Timing for 850 MHZ, V100 HBM runs at 850 MHZ
-gpgpu_dram_timing_opt
"nbk=16:CCD=1:RRD=3:RCD=12:RAS=28:RP=12:RC=40:

CL=12:WL=2:CDLR=3:WR=10:nbkgrp=4:CCDL=2:RTPL=3"
```

配置：SM7_QV100

```
# HBM has dual bus interface, in which it can issue two
col and row commands at a time
-dram_dual_bus_interface 1
# select lower bits for bnkgrp to increase bnkgrp
parallelism
-dram_bnk_indexing_policy 0
-dram_bnkgrp_indexing_policy 1

#-dram_seperate_write_queue_enable 1
#-dram_write_queue_size 64:56:32
```


配置：SM7_QV100

```
# stat collection
-gpgpu_memlatency_stat 14
-gpgpu_runtime_stat 500
-enable_ptx_file_line_stats 1
-visualizer_enabled 0

# power model configs, disable it untill we create a
real energy model for Volta
-power_simulation_enabled 0

# tracing functionality
#-trace_enabled 1
#-trace_components WARP_SCHEDULER,SCOREBOARD
#-trace_sampling_core 0
```

讲授内容：GPGPU-Sim简介

I. 模拟仿真模式
II. Debugging
III. 配置
IV. 输出信息
V. 可视化

输出信息

✓General Simulation Statistics
✓Simple Bottleneck Analysis
✓Memory Access Statistics
✓Memory Sub-System Statistics
✓Control-Flow Statistics
✓DRAM Statistics
✓Cache Statistics
✓Interconnect Statistics

讲授内容：GPGPU-Sim简介

I. 模拟仿真模式
II. Debugging
III. 配置
IV. 输出信息
V. 可视化

可视化

✓高层次微构架行为：AerialVision是一款用于GPGPU-Sim的GPU性能分析工具；
✓逐周期（Cycle by Cycle）微构架行为：GDB macro。

http://www.gpgpu-sim.org/

讲授内容

- GPU编程补充知识：AMD ROCm编程
- 基础知识
- 编程模型
- GPGPU-Sim简介
- 课程大作业（补充）

评分标准

- ✓课程大作业（一）：30%　加分题：20%
- ✓课程大作业（二）：30%　加分题：20%
- ✓期末考试（开卷）：30%
- ✓考勤：10%
- ✓满分：100%（如果总分超过100%，成绩为100%）

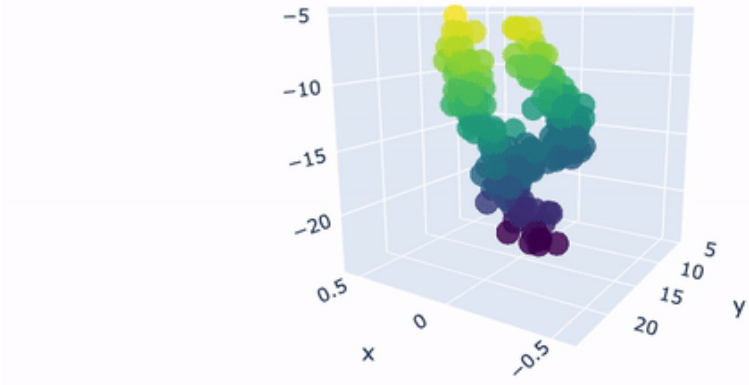
课程大作业：GPU编程

- ✓基于CUDA语言、Triton语言（加分题）、在GPGPU-sim上（加分题），实现CNN推理和训练，并在3D MNIST数据集合上测试，不能调用英伟达公司library；
- ✓调优。与英伟达library相比，你的实现达到了几成功力呢？
- ✓截止日期：作业一、第9周上课前，截止日期之前可多次提交；作业二、第16周上课前，截止日期之前可多次提交。
- ✓评分细节即将发布。

课程大作业：GPU编程

- ✓数据集合：3D MNIST。
- ✓数据描述（下载地址）：

https://www.kaggle.com/datasets/daavoo/3d-mnist



课程大作业：GPU编程

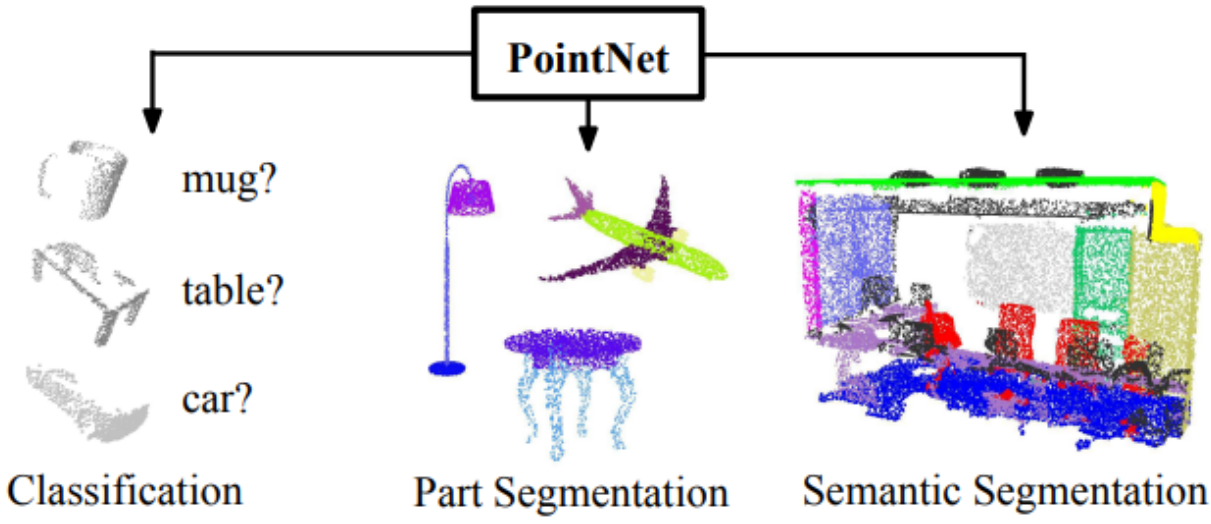
✓算法：PointNet，参考文献：

https://openaccess.thecvf.com/content_cvpr_2017/papers/Qi_PointNet_Deep_Learning_CVPR_2017_paper.pdf，在百度搜到中文介绍。

R. Charles, H. Su, M. Kaichun and L. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017 pp. 77-85.

课程大作业：GPU编程

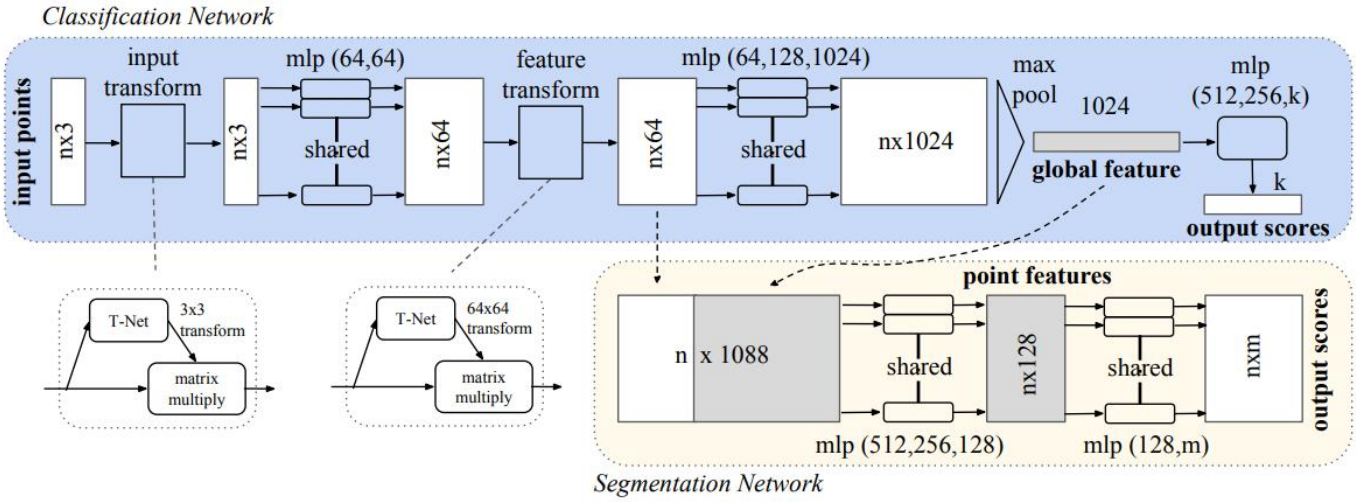
✓原理：



R. Charles, H. Su, M. Kaichun and L. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017 pp. 77-85.

课程大作业：GPU编程

✓算法：



R. Charles, H. Su, M. Kaichun and L. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017 pp. 77-85.

课程大作业：GPU编程

✓参考代码：https://github.com/charlesq34/pointnet

✓MATLAB代码：

https://www.mathworks.com/help/vision/ug/point-cloud-classification-using-pointnet-deep-learning.html

课程大作业：GPU编程

- 课程大作业（一）：用CUDA实现PointNet，并对3D MNIST进行推理（分类），30分；
  - ✓加分题（选择以下一题），20分：
    - ✓用Triton实现PointNet，并对3D MNIST进行推理（分类）；
    - ✓在GPGPU-Sim上加速你的CUDA实现PointNet推理（分类）。
- 课程大作业（二）：用CUDA实现PointNet，并对3D MNIST进行训练（分类），30分；
  - ✓加分题（选择以下一题），20分：
    - ✓用Triton实现PointNet，并对3D MNIST进行训练（分类）；
    - ✓在GPGPU-Sim上加速你的CUDA实现PointNet训练（分类）。