UCLA Samueli School of Engineering

CS-259
GPU Microarchitecture

Blaise Tine
UCLA Computer Science

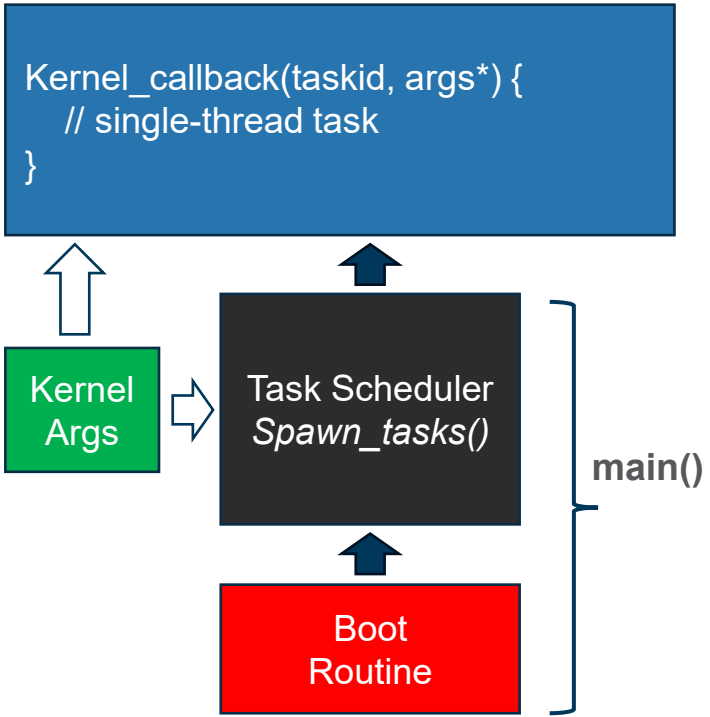
Attendance

Agenda

- Warp Scheduling
- Projects

OpenGPU Kernel launch Architecture

- Software-based scheduler
- One program executes on the GPU per kernel invocation
- The kernel program contains
  - Boot routine
  - Task scheduler
  - kernel callback
- All active core executes the boot routine to initialize RISC-V ABI registers
- After boot, each core executes the kernel scheduler
- The scheduler uses the kernel dimension in kernel argument to dispatch the kernel callback on allocated hardware resources.
  - Depending on the kernel dimension, only a selected number of core will be assigned a task.



OpenGPU Kernel launch Architecture (2)

- Kernel binary and kernel arguments are loaded to memory
  - Using separate device buffers allocated on the host CPU
- Host CPU use GPU's device configuration register to specify the addresses of those buffers
  - VX_DCR_BASE_STARTUP_ADDR
  - VX_DCR_BASE_STARTUP_ARG
- The task scheduler access the kernel argument via ***MSCRATCH*** CSR register.

```
#include <stdint.h>
#include <vx_intrinsics.h>
#include <vx_spawn.h>
#include "common.h"

void kernel_body(int task_id, kernel_arg_t* __UNIFORM__ arg) {
  auto src0_ptr = reinterpret_cast<TYPE*>(arg->src0_addr);
  auto src1_ptr = reinterpret_cast<TYPE*>(arg->src1_addr);
  auto dst_ptr  = reinterpret_cast<TYPE*>(arg->dst_addr);

  dst_ptr[task_id] = src0_ptr[task_id] + src1_ptr[task_id];
}

int main() {
  kernel_arg_t* arg = (kernel_arg_t*)csr_read(VX_CSR_MSCRATCH);
  vx_spawn_tasks(arg->num_points, (vx_spawn_tasks_cb)kernel_body, arg);
  return 0;
}
```

Sample vecadd kernel on OpenGPU

OpenGPU Kernel launch Architecture (3)

- Serial multi-kernels execution
  - Load kernel A with its argument
  - Load kernel B with its argument
  - Launch kernel A
  - Launch kernel B
- Concurrent multi-kernels execution
  - Load kernel A
  - Load kernel B
  - Load wrapper kernel with its argument
    - argument containing A and B arguments
  - Launch kernel wrapper kernel
    - Wrapper kernel scheduler will distribute A and B among the available resources

Hardware vs Software Kernel Scheduler

- Software scheduler advantages
  - Enable dynamic scheduling
  - Not scheduling resource limits
  - Reduced hardware cost
- Software scheduler disadvantages
  - Scheduler latency
    - Needs large workload to hide cost

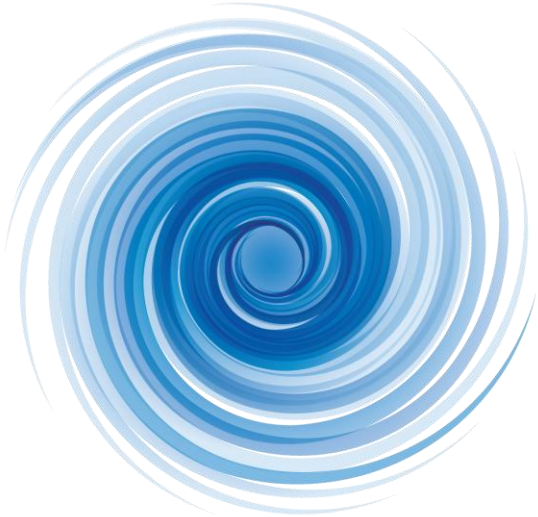
Additional Reading

- CUDA Programming Guide
  - http://docs.nvidia.com/cuda/cuda-c-programming-guide/#abstract
- Documentation for the GPGPUSim simulator
  - Good source of information about the general organization and operation of a stream multiprocessor
  - http://gpgpu-sim.org/manual/index.php/GPGPU-Sim_3.x_Manual
- Operation of a Scoreboard
  - https://en.wikipedia.org/wiki/Scoreboarding
- General Purpose Graphics Architectures, T. Aamodt, W. Fung, and T. Rogers, Chapter 2.2
- J. Wang, A. Sidelink, N Rubin, and S. Yalamanchili, "Dynamic Thread Block Launch: A Lightweight Execution Mechanism to Support Irregular Applications on GPUs", *IEEE/ACM* International Symposium on Computer Architecture (ISCA), June 2015

Class Projects

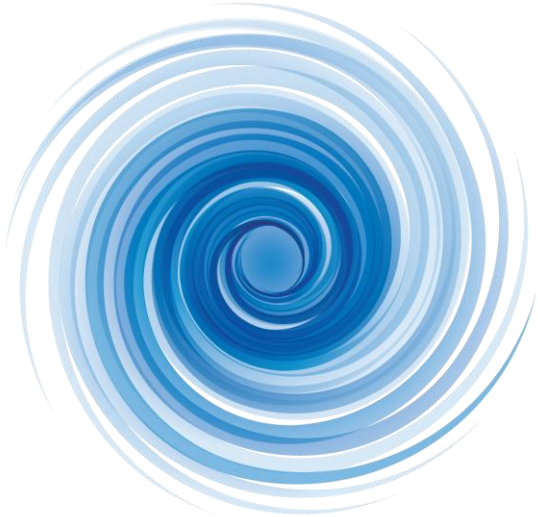
Vortex Projects

1. GPU 16-bit Compressed Extension
2. GPU Virtual Memory Extension
3. GPU Command Processor
4. GPU Atomics ISA Extension
5. GPU Debug ISA Extension
6. GPU HIP Runtime and Compiler Extension
7. GPU Ray-Tracing Unit
8. GPU Compressed Tiling Rasterization
9. GPU Unified Cache Shared Memory
10. GPU Shared Memory DMA
11. Dynamic Kernel Scheduler



Vortex Projects (2)

12. GPU Subwarps Scheduling
13. GPU Preemption for interrupts and exceptions
14. GPU Asynchronous Synchronization
15. GPU Tensor Core Sparsity
16. GPU DSP-based Tensor Unit
17. GPU Transform Architecture
18. GPU Tensor Memory
19. GPU Outer Product Tensor Core
20. GPU Systolic Tensor Core
21. Accelerating CNN on Vortex
22. Running FlashAttention-3 on Vortex



GPU 16-bit Compressed Extension

- Description
  - RISC-V 16-bit ISA reduces code size
  - GPU-port of the implementation
- Details
  - SimX implementation
  - RTL implementation
- Skill Levels
  - Verilog, C++
- Team size: 2
- References
  - https://riscv.org/wp-content/uploads/2015/05/riscv-compressed-spec-v1.7.pdf

GPU Shared Virtual Memory

- Description
  - Implement GPU Virtual memory
  - Based on OpenCL SVM
- Details
  - Page-table management on Host driver
  - Two-level page-table & TLBs
- Skill Levels
  - Verilog & C++
- Team Size: 2-3
- References
  https://www.intel.com/content/www/us/en/developer/articles/technical/opencl-20-shared-virtual-memory-overview.html
- https://www.intel.com/content/dam/develop/external/us/en/documents/tutorial-svm-basic.pdf

GPU Command Processor

- Description
  - Implement deferred GPU commands processing
- Details
  - Based on LibHSA specs
  - Configurable hardware queues
  - Commands synchronization barriers
- Skill Levels
  - Verilog & C++
- Team Size: 2-3
- References
  https://github.com/HSA-on-FPGA/LibHSA
  https://ieeexplore.ieee.org/document/8122108

GPU Atomics ISA extension

- Description
  - Implement the RISC-V standard extension
  - Support GPU non-blocking multi-bank caches
- Details
  - Verilog implementation
  - Cycle-level implementation provided
- Skill Levels
  - Verilog & C++
- Team Size: 2-3
- References
  - https://five-embeddev.com/riscv-user-isa-manual/Priv-v1.12/a.html
  - https://github.com/michaeljclark/riscv-atomics

GPU Debug ISA extension

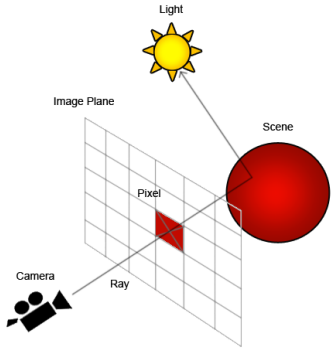
- Description
  - Implement the RISC-V standard extension
  - Support GDB via for OpenCOD
- Details
  - SimX ISA support
  - SimX debug module
  - OpenCOD pluggin
- Skill Levels
  - C++
- Team Size: 3
- References
  - https://riscv.org/technical/specifications/debug-specification/
  - https://github.com/riscv/riscv-isa-sim

Vortex HIP Runtime and Compiler

- Description
  - Implement the AMD HIP API on Vortex
  - Based on OpenCL 3.0 features
- Details
  - Runtime C++
  - Compiler Backend
- Skill Levels
  - C++
- Team Size: 2
- References
  - https://github.com/ROCm/hip
  - https://github.com/ROCm/HIP-CPU
  - https://github.com/CHIP-SPV/chipStar

GPU Extension for RayTracing

- Description
  - At the core of the Ray-tracing hardware there is a component called the T&I unit, Traversal and intersection unit, that traverses the bounding volumes in a 3D scene and performs the visibility testing for each triangle primitive.
- Details
  - Implementing a T&I unit and connect to the Vortex graphics pipeline.
  - Write custom GPU code to exercise the T&I
- Skill Levels
  - Verilog & C++
- Team size: 4
- References
- https://people.ece.ubc.ca/~aamodt/publications/papers/saed.micro2022.pdf
- https://intra.engr.ucr.edu/~htseng/files/2024MICRO-TTA.pdf
- https://people.ece.ubc.ca/aamodt/publications/papers/lumibench.iiswc2023.pdf
- https://dl.acm.org/doi/10.1145/3582016.3582024
- https://people.ece.ubc.ca/~aamodt/publications/papers/chou.asplos2025.pdf



GPU Compressed Rasterizer Tiling

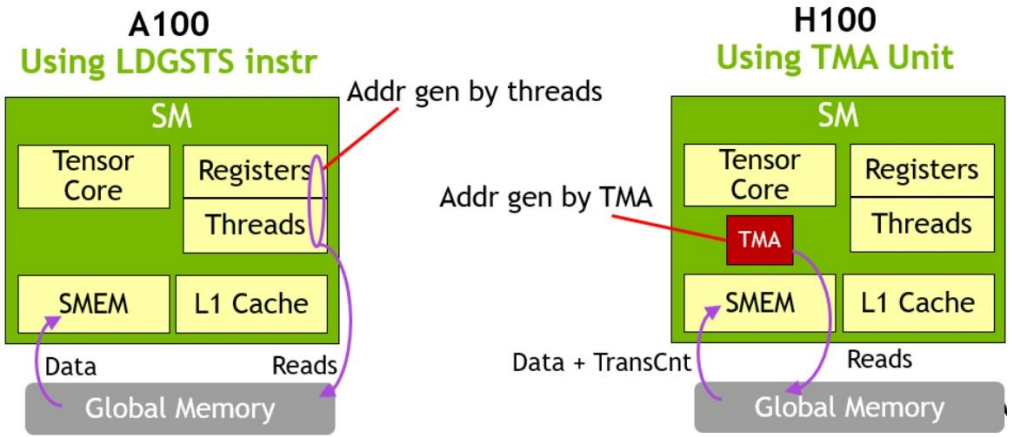
- Description
  - Implement compression on rasterizer tile buffer
  - Tile buffer hold rendering tiles covering segments of render target
  - Use bit compression
- Details
  - RTL implementation
  - SimX implementation
- Skill Levels
  - Verilog, C++
- Team size: 2
- References
  - Skybox: https://github.com/vortexgpgpu/skybox
  - Vulkan-Sim: https://github.com/ubc-aamodt-group/vulkan-sim

Unified Shared Memory Cache

- Description
  - Nvidia GPU caches since Fermi are partitioned with into shared memory storage.
  - You will implement simulation and RTL design.
- Details
  - Extend SimX Cache
  - Extend RTL Cache
- Skill Levels
  - Verilog & C++
- Team size: 2
- References
  - https://research.nvidia.com/sites/default/files/pubs/2012-12_Unifying-Primary-Cache/Gebhart_MICRO_2012.pd

GPU Shared Memory DMA

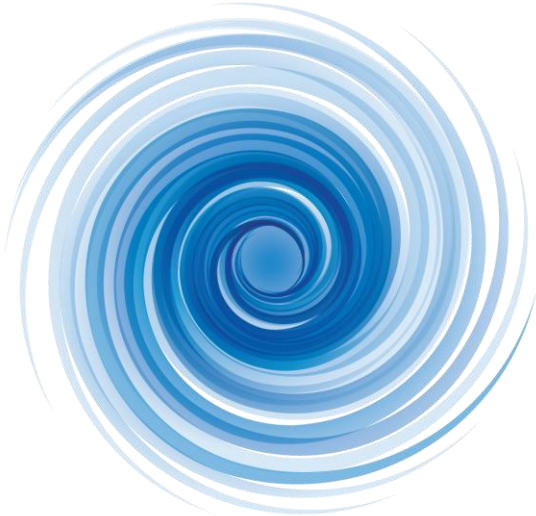
- Description
  - Implement DMA engine for moving memory blocks between global memory and share memory
  - You will implement simulation and RTL design.
- Details
  - Extend SimX
  - Extend RTL
- Skill Levels
  - Verilog & C++
- Team size: 2-3



- References
  - https://research.colfax-intl.com/tutorial-hopper-tma/

Dynamic Kernel Scheduler

- Description
  - Implement deferred command processor
  - Implement a KMU with dynamic scheduling
- Details
  - vx_spawn
  - support core affinity mask
- Skill Levels
  - Verilog & C++
- Team Size: 2
- References
  https://casl.gatech.edu/wp-content/uploads/2015/04/dtbl_isca42_jin.pdf
  https://github.com/THU-DSP-LAB/ventus-gpgpu-verilog/tree/main/src/gpgpu_top/cta_top
- vx_spawn
- **hw/rtl/afu using driver=xrt**



Subwarps Scheduling

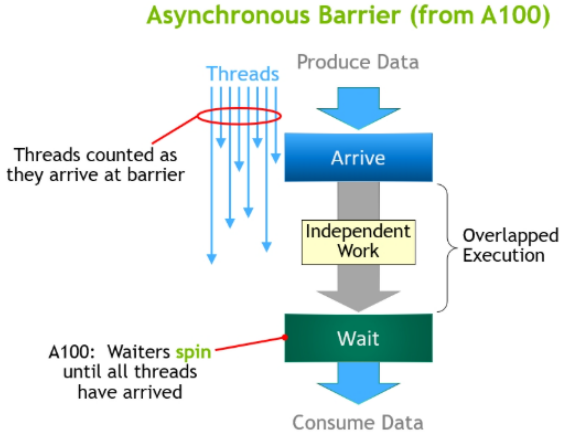
- Description
  - Using large warps (size > 8) affects performance due to divergence
  - Adding Subwarps (e.g. size=4) can amortize the cost
- Details
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team Size: 3
- References
  https://research.nvidia.com/publication/2022-01_gpu-subwarp-interleaving

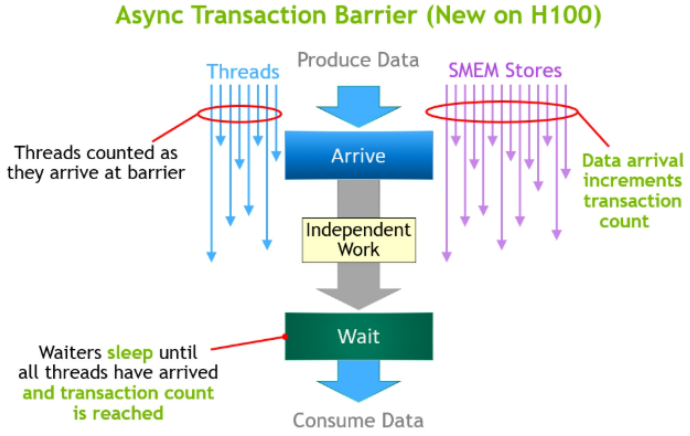
GPU Preemption

- Description
  - Being able to interrupt the GPU execution and resuming execution where it last stopped
  - Useful for handling exceptions and interrupts
- Details
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team Size: 2
- References
  https://hzhou.wordpress.ncsu.edu/files/2022/12/sc16.pdf?utm_source=chatg
- https://dl.acm.org/doi/pdf/10.1145/3123939.3123950

GPU Asynchronous Synchronization

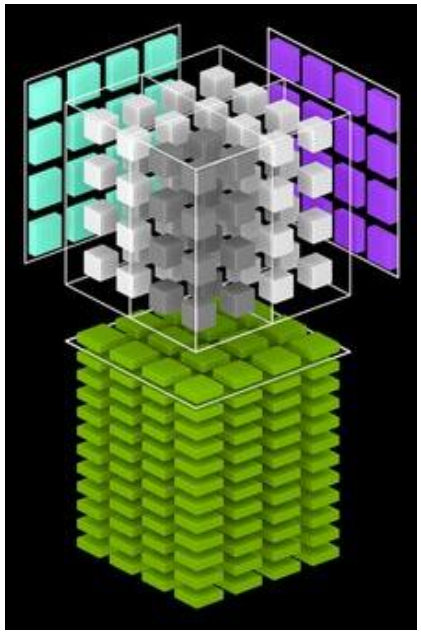
- Description
  - Adding asynchronous transaction barrier
  - Asynchronous barrier will allow warp to do independent work after arrival at the barrier and stall only when accessing the tagged resource – free when all participant arrive
- Details
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team Size: 2
- References





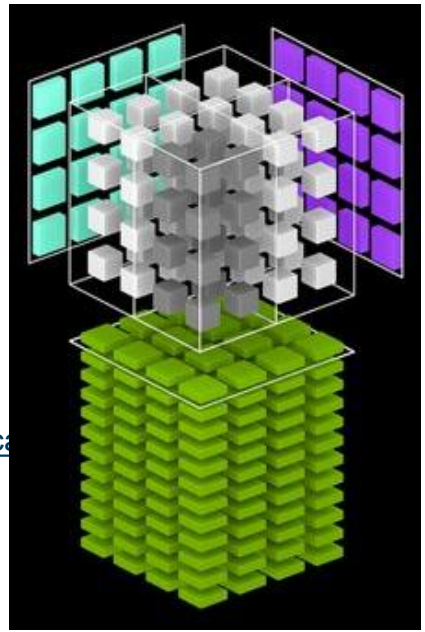
Sparse Tensor Acceleration

- Description
  - A tensor core is a critical component for accelerating fused matrix accumulation on GPU. Implement 4:2 structured sparsity
- Details
  - Support int8, int4, fp16, bf16, f32 formats
  - Write custom GPU code to exercise the Tensor unit
- Skill Levels
  - Verilog & C++
- Team size: 3
- References
- **https://people.eecs.berkeley.edu/~ysshao/assets/papers/virgo-asplos2025.p**
- **https://arxiv.org/pdf/2402.00028**
- **https://github.com/hansungk/vortex/tree/rtl**



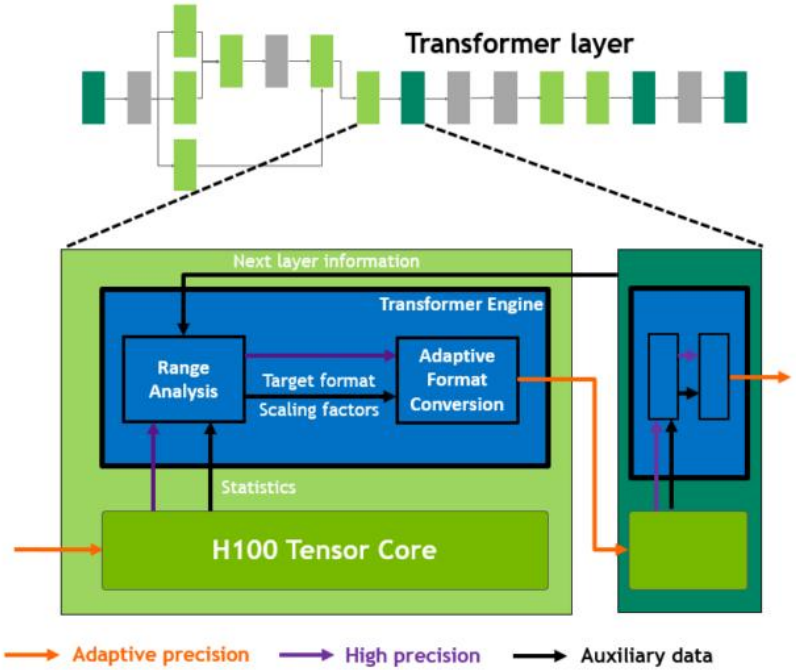
DSP-Based Tensor Unit

- Description
  - Xilinx FPGA support hard block integer DSP48E1
  - Implement Tensor Core Dot Product targeting
- Details
  - Support int8, int4, fp16, bf16, f32 formats
  - Write custom GPU code to exercise the Tensor unit
- Skill Levels
  - Verilog
- Team size: 1
- References
- Iterative floating point computation using FPGA DSP blocks | IEEE Conference Publica | IEEE Xplore



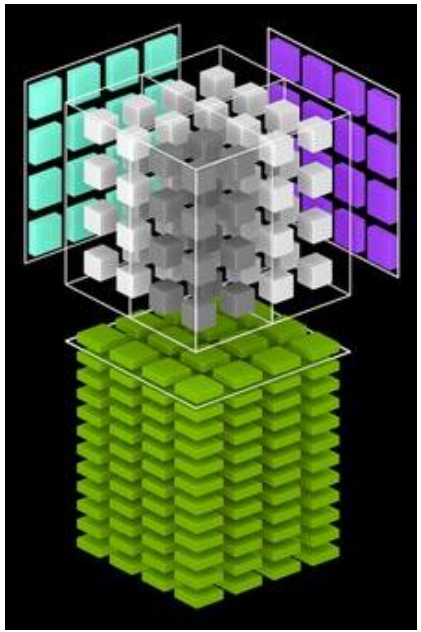
Transformer Architecture

- Description
  - Optimize precision of fp8
  - Hardware-assisted variable-precision matrix multiplication
- Details
  - Software stack
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team size: 2-3
- References
  - https://github.com/NVIDIA/TransformerEngine



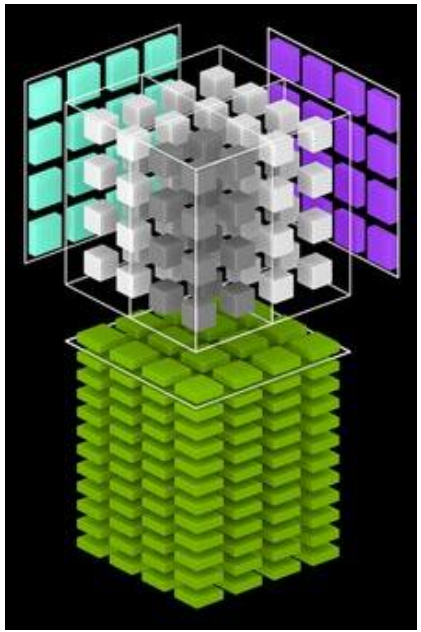
GPU Tensor Memory

- Description
  - Adding new private memory hierarchy to the GPU
  - Fast handling of Tile data transfer
- Details
  - Software
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team size: 3
- References
- https://research.colfax-intl.com/cutlass-tutorial-writing-gemm-kernels-using-tensor-memory-for-nvidia-blackwell-gpus



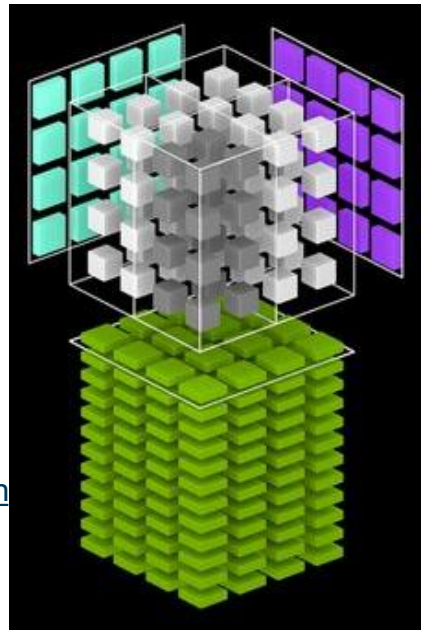
GPU Outer Product Tensor Core

- Description
  - Nvidia tensor core use inner product design
  - Fast handling of Tile data transfer
- Details
  - Software
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team size: 3
- References
- https://research.colfax-intl.com/cutlass-tutorial-writing-gemm-kernels-using-tensor-memory-for-nvidia-blackwell-gpus



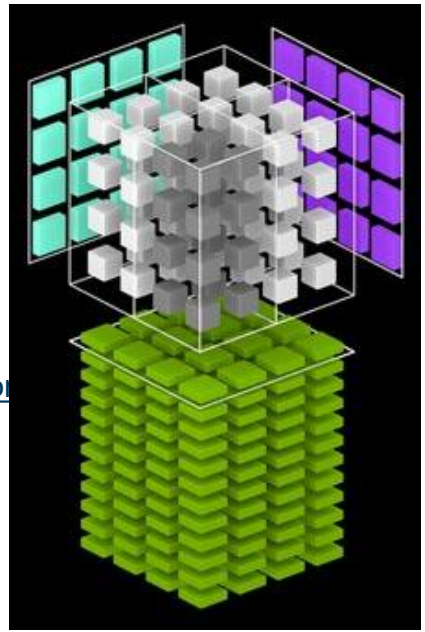
GPU Systolic Tensor Core

- Description
  - Implement Systolic-array based Tensor core on Vortex
  - Based on Virgo architecture
- Details
  - Software
  - SimX
  - RTL
- Skill Levels
  - Verilog & C++
- Team size: 3
- References
- GitHub - ucb-bar/virgo: Cluster-level matrix unit integration into GPUs, implemented in Chipyard SoC



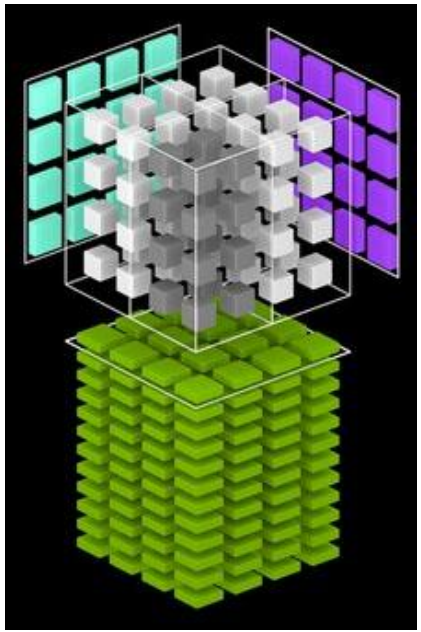
Accelerating CNN on Vortex

- Description
  - Implementing a full CNN demo on Vortex GPU
  - Add extension to execute on FPU vs tensor core
- Details
  - Software
- Skill Levels
  - C++
- Team size: 2
- References
- GitHub - warrenlyr/CUDA-CNN-From-Scratch: Build Convolutional Neural Network fro
  scratch and accelerate with CUDA.



Running FlashAttention-3 on Vortex

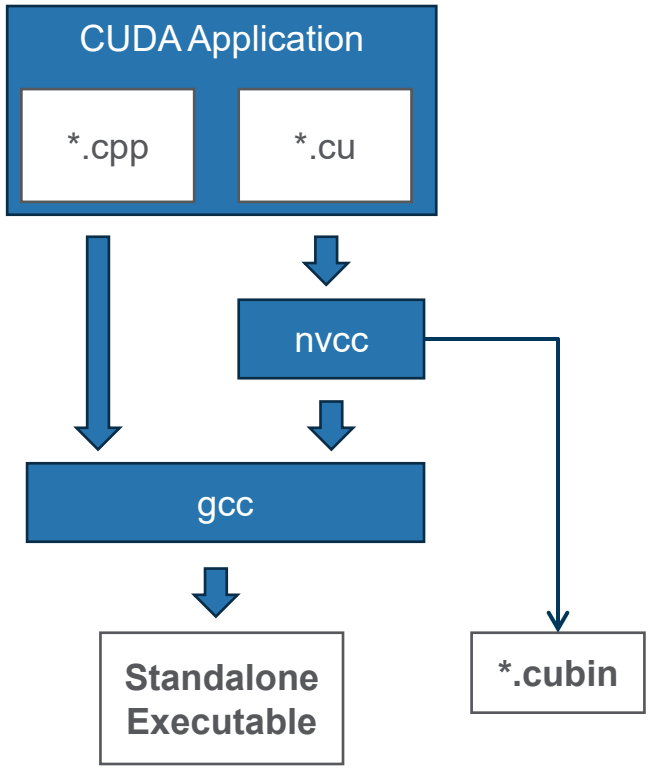
- Description
  - Flash Attention is a revolutionary technique that dramatically accelerates the attention mechanism in transformer-based models, delivering processing speeds
  - Porting a demo on Vortex
- Details
  - Software
- Skill Levels
  - C++
- Team size: 2
- References
  - GitHub - Dao-AILab/flash-attention: Fast and memory-efficient exact attention
  - 2407.08608
  - Understanding Flash Attention: Writing Triton Kernel Code



GPU ISA

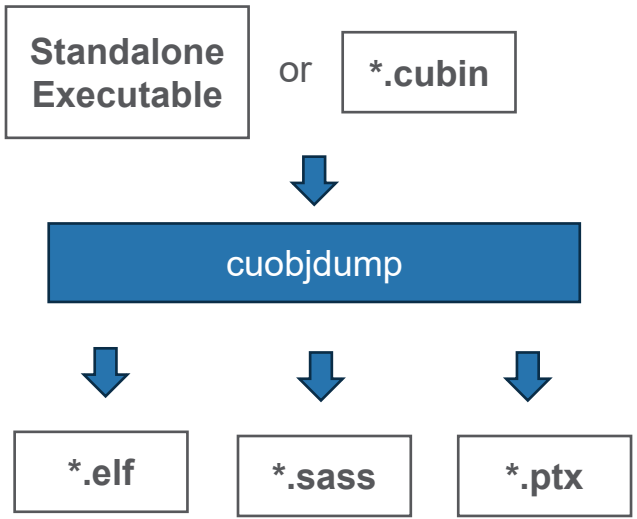
CUDA Program Compilation

- NVCC compiler
  - Generate device binary
- Host compiler
  - Link host + device
  - Generate merged executable
  - How does it work?



CUDA Program Disassembly

- cuobjdump
  - Device object dump utility
  - Extract
    - PTX
    - SASS
    - ELF



CUDA PTX

- PTX := Parallel Thread Execution
- ISA Intermediate Representation
  - Portable assembly language
  - Abstract machine and ABI
- Main abstractions
  - Memory model
    - .const, .shared, .tex, .param
  - Register types
    - .reg, .sreg, .preg
  - Operations
    - Add, Sub, LD, ST, etc..

```
.version 7.0
.target sm_70
.address_size 64

.visible .entry _Z3addPiS_S_(
  .param .u64 _Z3addPiS_S__param_0,
  .param .u64 _Z3addPiS_S__param_1,
  .param .u64 _Z3addPiS_S__param_2
) {
  ld.param.u64 %rd1, [_Z3addPiS_S__param_0];
  ld.param.u64 %rd2, [_Z3addPiS_S__param_1];
  ld.param.u64 %rd3, [_Z3addPiS_S__param_2];
  cvta.to.global.u64 %rd4, %rd3;
  cvta.to.global.u64 %rd5, %rd2;
  cvta.to.global.u64 %rd6, %rd1;
  ld.global.u32 %r1, [%rd6];
  ld.global.u32 %r2, [%rd5];
  add.s32 %r3, %r2, %r1;
  st.global.u32 [%rd4], %r3;
  ret;
}
```

CUDA PTX (2)

- Special Registers
  - %tid: per CTA thread id (x,y,z)
  - %ntid: per CTA num threads (x,y,z)
  - %laneid: warp lane id
  - %warpid: per CTA warp id
  - %nwarpid: per CTA number of warps
  - %ctaid: CTA grid indices (x,y,z)
  - %nctaid: CTA grid sizes (x,y,z)
  - %smid: SM id
  - %nsmid: number of SMs

```
.version 7.0
.target sm_70
.address_size 64

// compute unified thread id for 2D CTA
mov.u32  %r0,%tid.x;
mov.u32  %h1,%tid.y;
mov.u32  %h2,%ntid.x;
mad.u32  %r0,%h1,%h2,%r0;
```

Difference between tid vs laneid?

Which registers are used to compute the kernel's *block* and *grid* variables?

CUDA PTX (3)

- Barrier synchronization
  - bar.cta.sync id [#n]
    - barrier id
    - wait for #n threads
  - bar.warp.sync mask
    - wait for all threads in mask
  - vote.<mode>.pred q, p, mask
    - Select each p into q
    - <mode> = all, any, uni
  - vote.ballot.u32 d, p, mask
    - Select each p into d
    - d contains lane ids

```
.version 7.0
.target sm_70
.address_size 64

// Use bar.sync to arrive at a pre-computed barrier number and
// wait for fixed number of cooperating threads to arrive:
#define CNT1 (8*12) // Number of cooperating threads

st.shared [r0],r1;     // write my result to shared memory
bar.cta.sync  1, CNT1; // arrive, wait for others to arrive
ld.shared r2,[r3];     // use shared results from other threads

st.shared.u32 [r0],r1;         // write my result to shared memory
bar.warp.sync  0xffffffff;     // arrive, wait for others to arrive
ld.shared.u32 r2,[r3];         // read results written by other threads

vote.all.pred    p,q;
vote.uni.pred    p,q;
vote.ballot.b32  r1,p;  // get 'ballot' across warp
```

CUDA PTX (4)

- Control-flow
  - Predication
  - No explicit instruction
  - Uniform Hint
    - .uni

```
 .version 7.0
 .target sm_70
 .address_size 64

 setp.eq.f32  p,y,0;     // is y zero?
@!p div.f32      ratio,x,y  // avoid division by zero
@q  bra L23;                // conditional branch


@p   bra{.uni}  tgt;          // tgt is a label
     bra{.uni}  tgt;          // unconditional branch
```

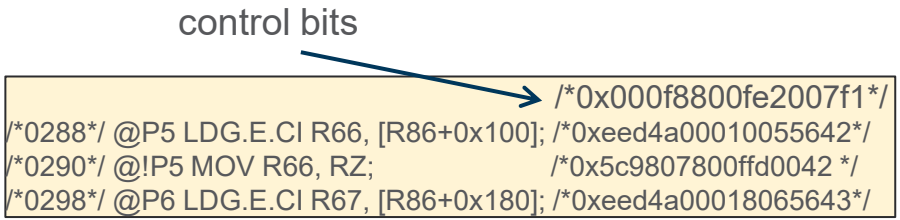
CUDA SASS

- Source and Assembly
  - Minimal documentation
- Low-level ISA
  - Machine-specific instruction set
    - Pre-Volta 256-bit encoding
      - Control bits + 3 instruction words
    - 128-bit encoding (Volta+)
      - Control bits + 1 instruction word
  - Direct hardware registers
    - R1, RZ(zero)
  - Cache control
    - CCTL, CCTLT

```
code for sm_70
        Function : _Z3addPiS_S_
/*0000*/      IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ;
/*0010*/ @!PT SHFL.IDX PT, RZ, RZ, RZ, RZ ;
/*0020*/      IMAD.MOV.U32 R2, RZ, RZ, c[0x0][0x160] ;
/*0030*/      MOV R3, c[0x0][0x164] ;
/*0040*/      IMAD.MOV.U32 R4, RZ, RZ, c[0x0][0x168] ;
/*0050*/      MOV R5, c[0x0][0x16c] ;
/*0060*/      LDG.E.SYS R2, [R2] ;
/*0070*/      LDG.E.SYS R5, [R4] ;
/*0080*/      IMAD.MOV.U32 R6, RZ, RZ, c[0x0][0x170] ;
/*0090*/      MOV R7, c[0x0][0x174] ;
/*00a0*/      IADD3 R9, R2, R5, RZ ;
/*00b0*/      STG.E.SYS [R6], R9 ;
/*00c0*/      EXIT ;
/*00d0*/      BRA 0xd0;
/*00e0*/      NOP;
/*00f0*/      NOP;
```

CUDA SASS (2)

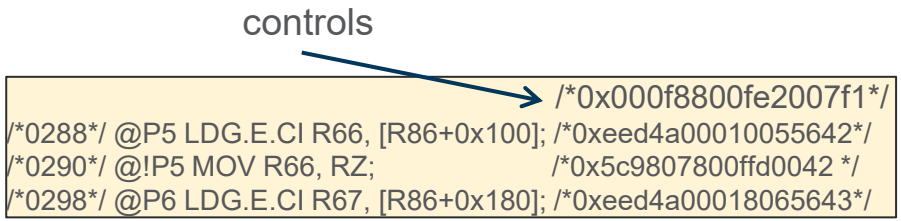
- Instructions with control states
  - Moving scoreboard scheduling to software
  - Reduce scoreboard hardware complexity
  - Enable instruction interleaving



nvdisasm output on Pascal

CUDA SASS (2)

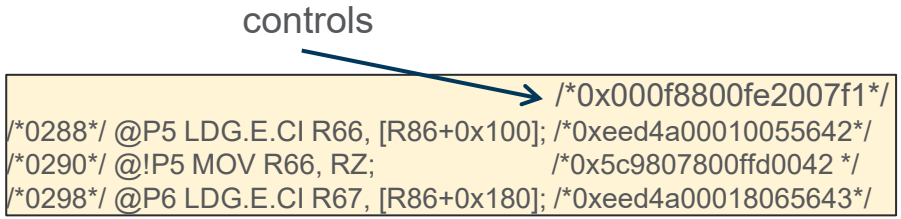
- Control States
  - Reuse flags
    - 4 bits value
    - Use 4 operands cache
    - 2-way associative CAM
    - Reduce register pressure



nvdisasm output on Pascal

CUDA SASS (3)

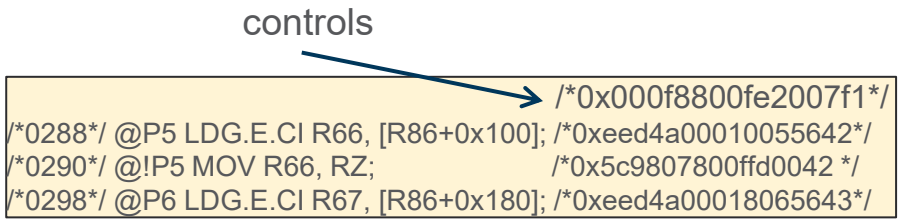
- Control States
  - Wait barrier id
  - Read barrier id
  - Write barrier id, mask
    - Dependency barriers
    - Track instruction completion
    - Variable latency instructions
    - Writers are assigned a barrier id
    - Readers wait on dependent barrier ids
    - Read barrier mask is for WAR dependencies.



nvdisasm output on Pascal

CUDA SASS (4)

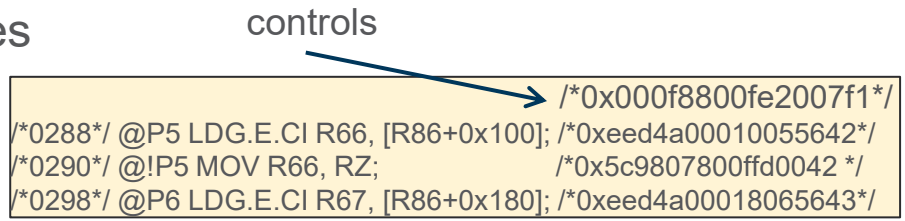
- Control States
  - Yield flag
    - If clear, switch to another warp
    - Will invalidate all reuse flags
    - Switching warps costs an extra cycle



nvdisasm output on Pascal

CUDA SASS (4)

- Control States
  - Stall cycles
    - 4 bits value
    - Stall scheduler for up to 15 cycles



nvdisasm output on Pascal