Introduction to Parallel & Distributed Computing

Introduction to Parallel Programming with ROCm HIP (1)

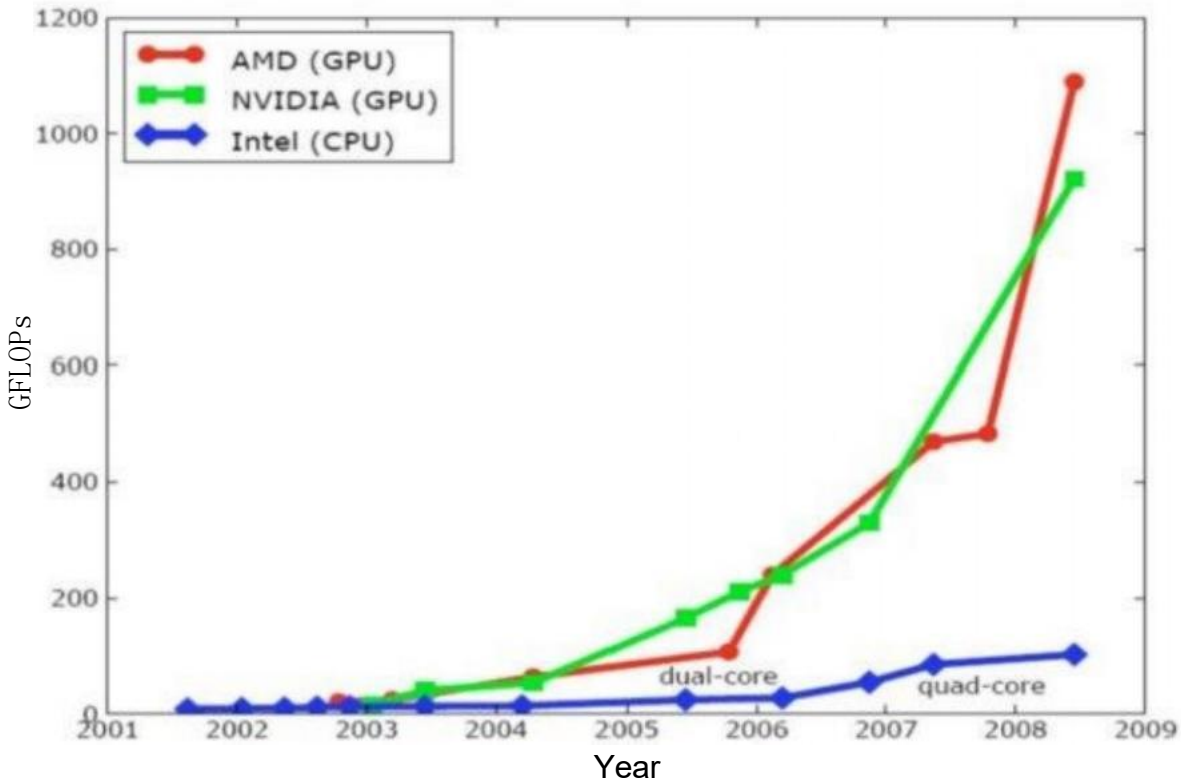
Lecture 8, Spring 2024

Instructor: 罗国杰

gluo@pku.edu.cn

Adapted from UW CSEP 590B, Dr. Hari Sadasivan
https://courses.cs.washington.edu/courses/csep590b/

Silent revolution in Graphics

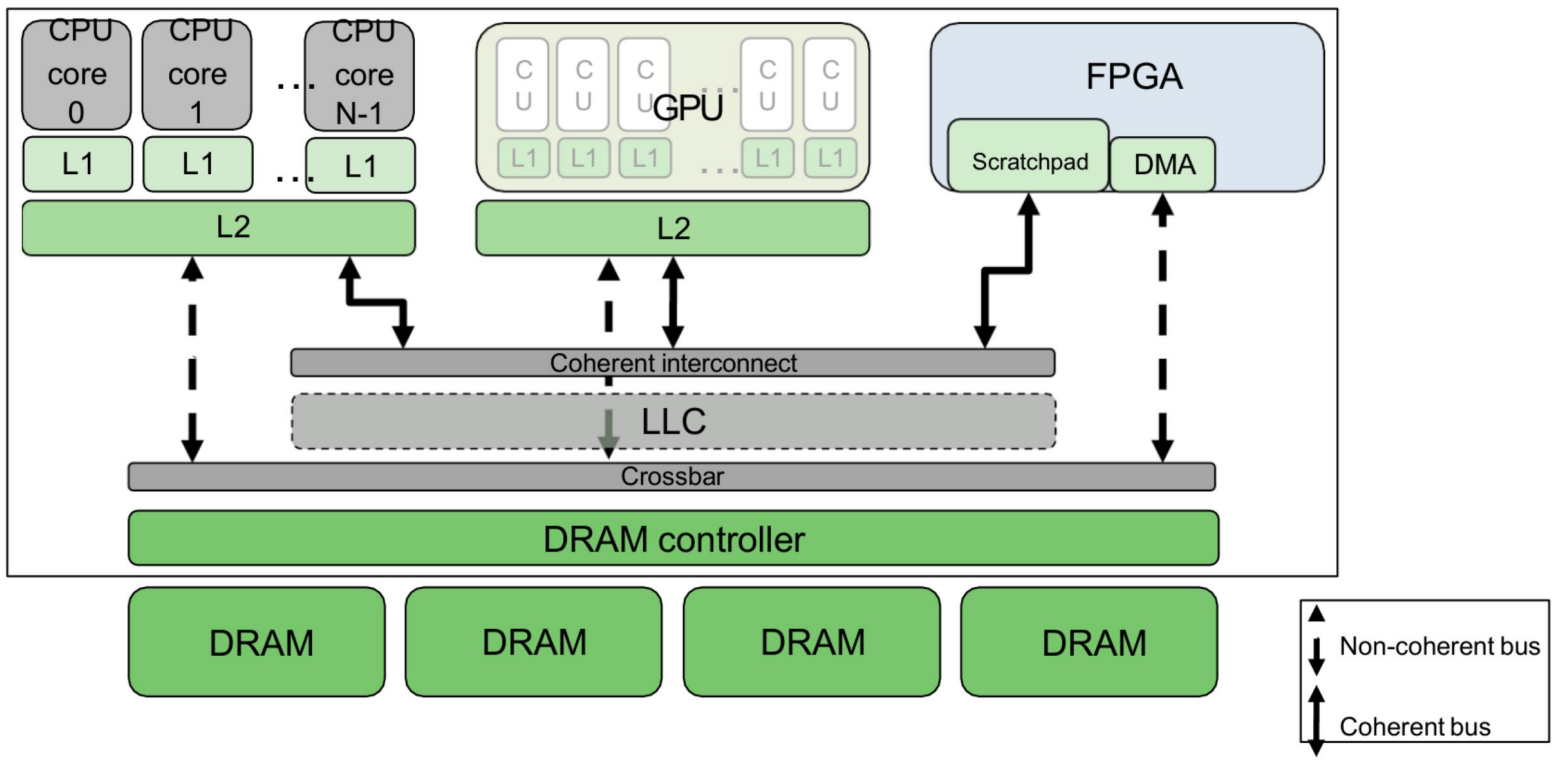


An Effective Dynamic Scheduling Runtime and Tuning System for Heterogeneous Multi and Many-Core Desktop Platforms, Binotto et al.

adapted from UW CSEP 590, Dr. Hari Sadasivan

Heterogeneous Computing Systems

- The end of Moore's law created the need for heterogeneous systems
  - More suitable devices for each type of workload
  - Increased performance and energy efficiency



Credits: Prof.Reetu Das, UM

GPUs in High Performance Computing

adapted from UW CSEP 590, Dr. Hari Sadasivan

Computing at Exascale

El Capitan at Lawrence Livermore National Laboratory (LLNL) uses AMD's MI300A

Performance expected to exceed 2ExaFLOPs, which comes with a $600 million price tag.

adapted from UW CSEP 590, Dr. Hari Sadasivan

Frontier-World's fastest supercomputer

- Oak Ridge National Lab
- Uses MI250X and 3rd Gen AMD Zen CPU for 1.6ExaFLOPs

adapted from UW CSEP 590, Dr. Hari Sadasivan

CPUs:Latency Oriented Design

- High clock frequency
- Large caches

-Convert long latency memory accesses to short latency cache accesses

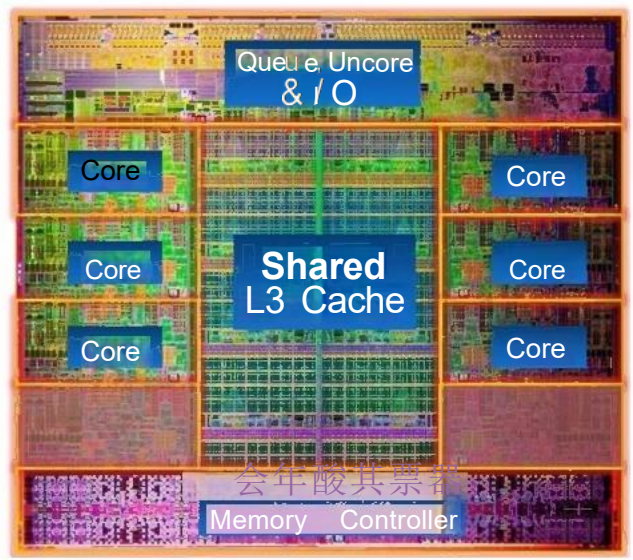
- Sophisticated control

-Branch predictipn for reduced branch latency

- Powerful ALU

-Reduced operation latency

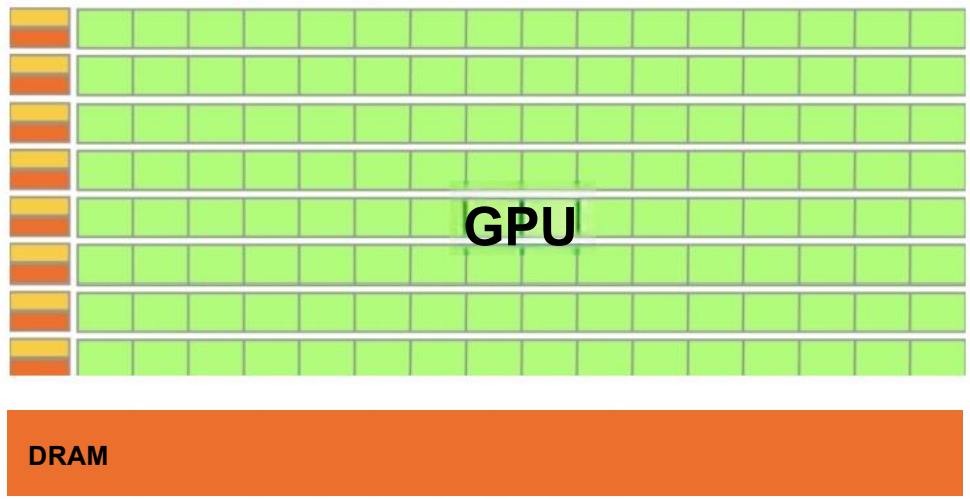
Intel CoreM i7-3960X Processor Die Detail



adapted from UW CSEP 590, Dr. Hari Sadasivan

GPUs:Throughput Oriented Design

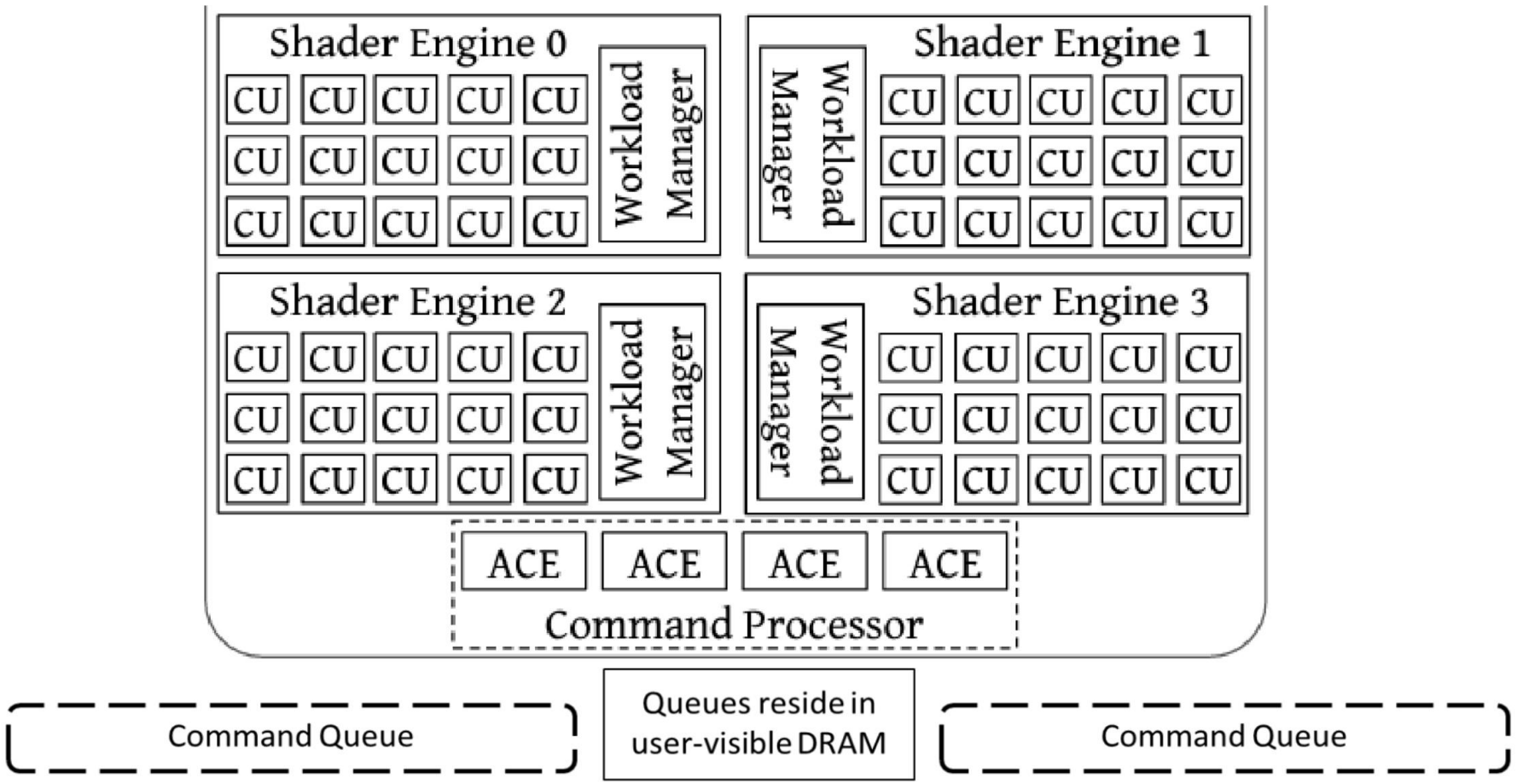
- Moderate clockfrequency
- Small caches
  -To boost memory throughput
- Simple control
  -No branch prediction
- Energy efficient ALUs
  -Many,long latency but heavily pipelined for high throughput



- Require massive number of threads to tolerate latencies

Credits: Prof. Reetu Das, UM

AMD GCN HW Layout

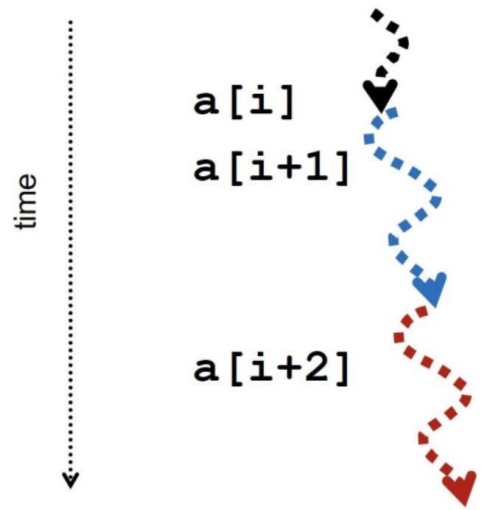


Nathan Otterness and James H.Anderson.2021.Exploring AMD GPU Scheduling Details by Experimenting With "Worst Practices".

adapted from UW CSEP 590, Dr. Hari Sadasivan

Sequential Execution Model

- One instruction at the time
- Optimizations possible at the machine level



int a[N]; // a is image, N is large

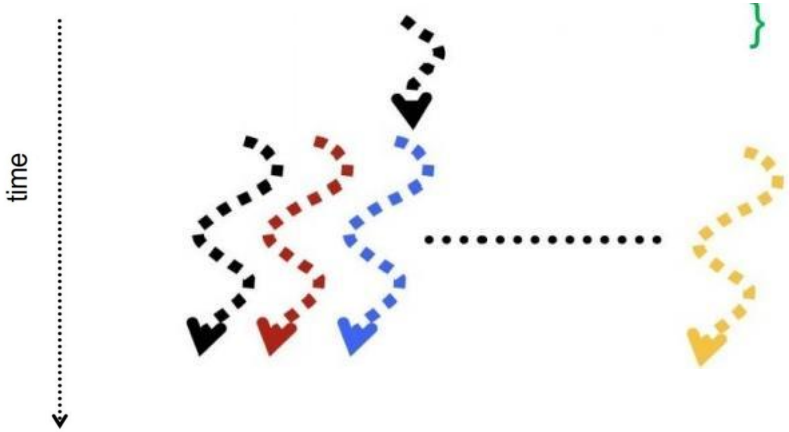
```
for (i =0; i < N; i++){
  a [i] = a[i]* fade;
}
```

Parallel Execution Model:SIMD

- Example:image fading
  - -Some instructions executed concurrently

```
int a[N];//N is large

for all elements do in parallel
{ a[i]=a[i]*fade;
}
```



Most modern CPUs offer some limited support for SIMD

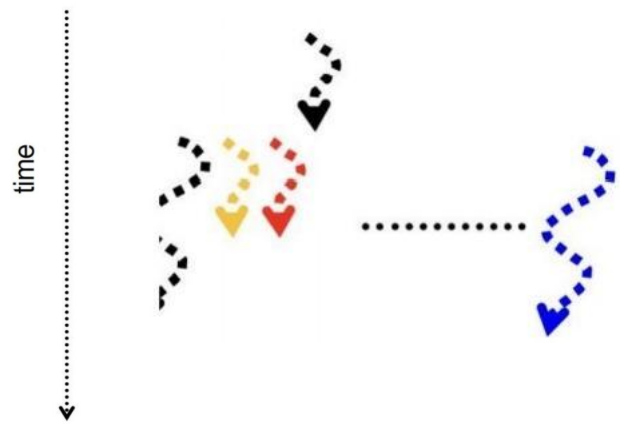
This has been tried before:ILLIAC III,UIUC,1966
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4038028&tag=1
http://ed-thelen.org/comp-hist/vs-illiac-iv.html

Parallel Execution Model:SPMD

- Example:image fading
  -The code is identical across all threads
  -The execution path may differ

```
int a[N]; //N is large

for all elements do in parallel
  If (a[i] >threshold) a[i]*=fade;
```



adapted from UW CSEP 590, Dr. Hari Sadasivan

Data Parallelism vs Task Parallelism

- Data parallelism: reorganization of data such that computations can be performed in parallel
  - Some things are naturally parallel
- Task parallelism: When computational tasks are independent,they may be performed in parallel

SIMD vs.SIMT Execution Model

- SIMD:A single sequential instruction stream of SIMD instructions
  - □ each instruction specifies multiple data inputs
  - □ [VLD,VLD,VADD,VST],VLEN
- SIMT: Multiple instruction streams of scalar instructions
  - □threads grouped dynamically into warps
  - □ [LD,LD,ADD,ST],NumThreads

- Two Maior SIMT Advantages:

□ Can treat each thread separately □ i.e.,can execute each thread independently (on any type of scalar pipeline)□ MIMD processing

o Can group threads into warps flexibly □ i.e.,can group threads that are supposed to truly execute the same instruction □ dynamically obtain and maximize benefits of SIMD processing

---

adapted from UW CSEP 590, Dr. Hari Sadasivan

Sample GPU SIMT Code (Simplified)

CPU code

```
for (i=0; i<100000; ++i) {
    C[i] = A[i] + B[i];
}
```

HIP code

```
// there are  100000 threads
   __global__ void KernelFunction(…) {
      int tid = blockDim.x*blockIdx.x+threadIdx.x;
      int varA = A[tid];
      int varB = B[tid];
      C[tid] = varA+varB;
  }
```

Slide credit: Hyesoon Kim

Warps not Exposed to GPU Programmers

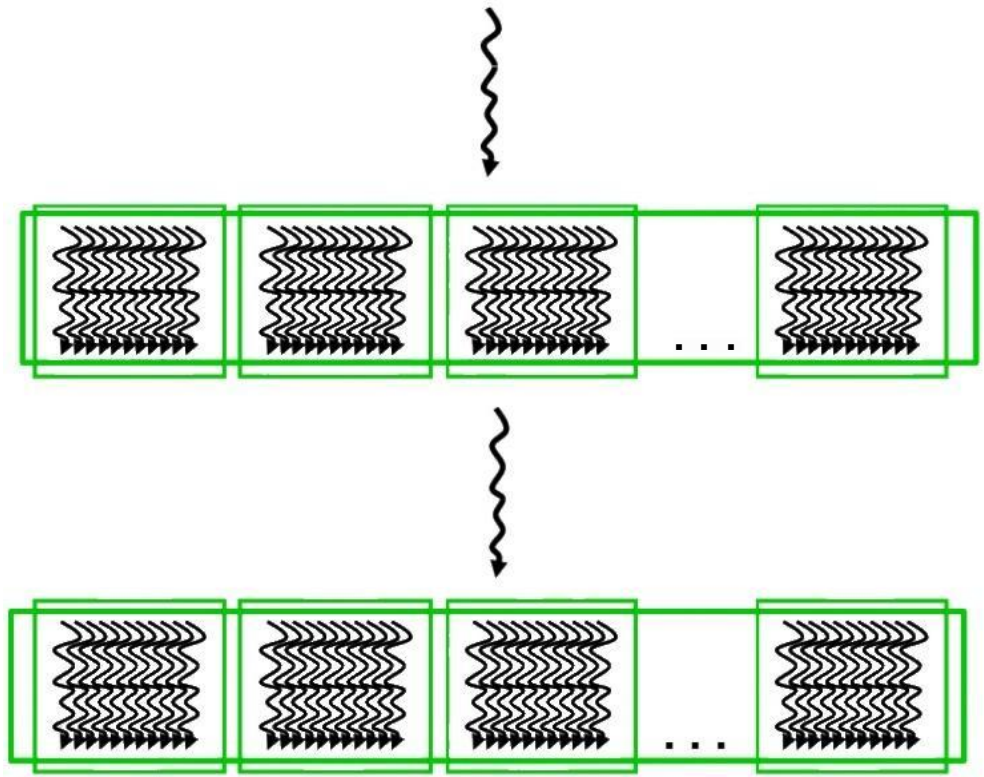
- CPU threads and GPU kernels
  - Sequential or modestly parallel sections on CPU
  - Massively parallel sections on GPU: Blocks of threads

Serial Code (host)

Parallel Kernel (device)
KernelA<<<nBlk,nThr>>>(args);

Serial Code (host)

Parallel Kernel (device)
KernelB<<<nBlk,nThr>>>(args);

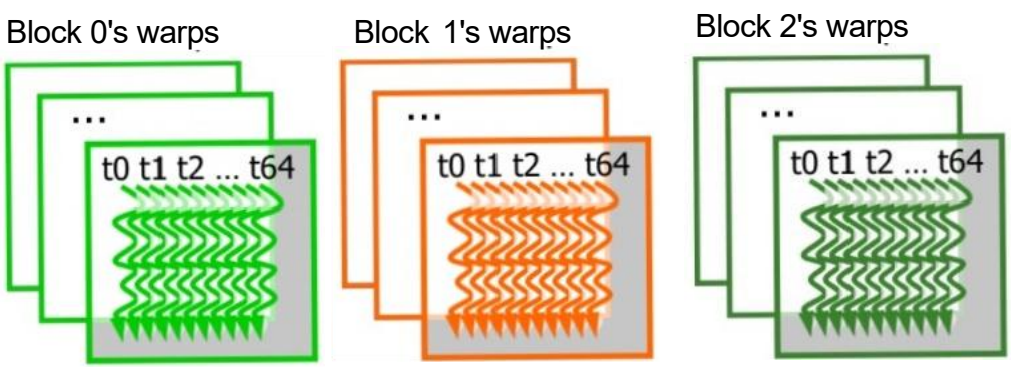


Slide credit:Hwu &Kirk

From Blocks to Warps

- GPU cores:SIMD pipelines
  - 16-lane wide Compute Units

- Workgroups/Blocks are divided into wavefronts/warps
  - SIMD unit (64 threads)



Warp-based SIMD vs.Traditional SIMD

- Traditional SIMD contains a single thread
  - Sequential instruction execution;lock-step operations in a SIMD instruction
  - Programming modelis SIMD (no extra threads)□ SW needs to know vector length
  - ISA contains vector/SIMD instructions

- Warp-based SIMD consists of multiple scalar threads executing in a SIMD manner(i.e.,same instruction executed by all threads)
  - □ Does not have to be lock step
  - □ Each thread can be treated individually (i.e.,placed in a different warp)
    - □ programming model not SIMD
      - SW does not need to know vector length
      - Enables multithreading and flexible dynamic grouping of threads
  - ISA is scalar □ SIMD operations can be formed dynamically
  - Essentially,it is SPMD programming model implemented on SIMD **hardware**

SPMD

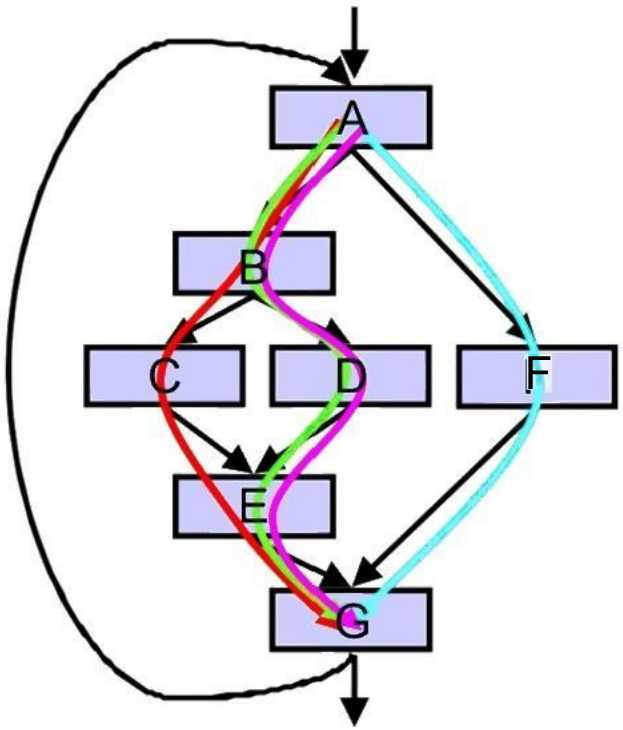
- Single procedure/program,multiple data
  - This is a programming model rather than computer organization
- Each processing element executes the same procedure,except on different data elements
  - Procedures can synchronize at certain points in program,e.g.barriers
- Essentially, multiple instruction streams execute the same program
  - Each program/procedure 1)works on different data,2)can execute a different control-flow path,at run-time
  - Many scientific applications are programmed this way and run on MIMD hardware (multiprocessors)
  - Modern GPUs programmed in a similar way on a SIMD hardware

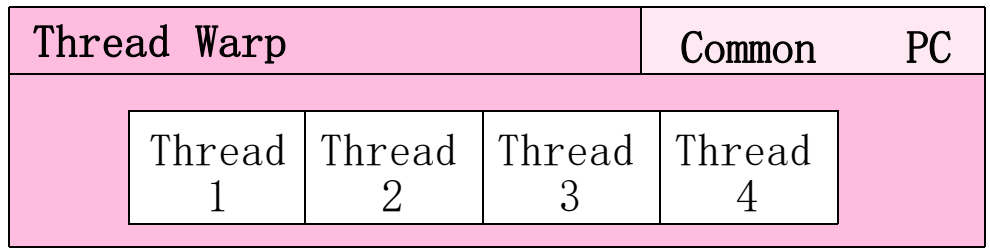
adapted from UW CSEP 590, Dr. Hari Sadasivan

Threads Can Take Different Paths in Warp-based SIMC

Each thread can have conditional control flow instructions

Threads can execute different control flow paths



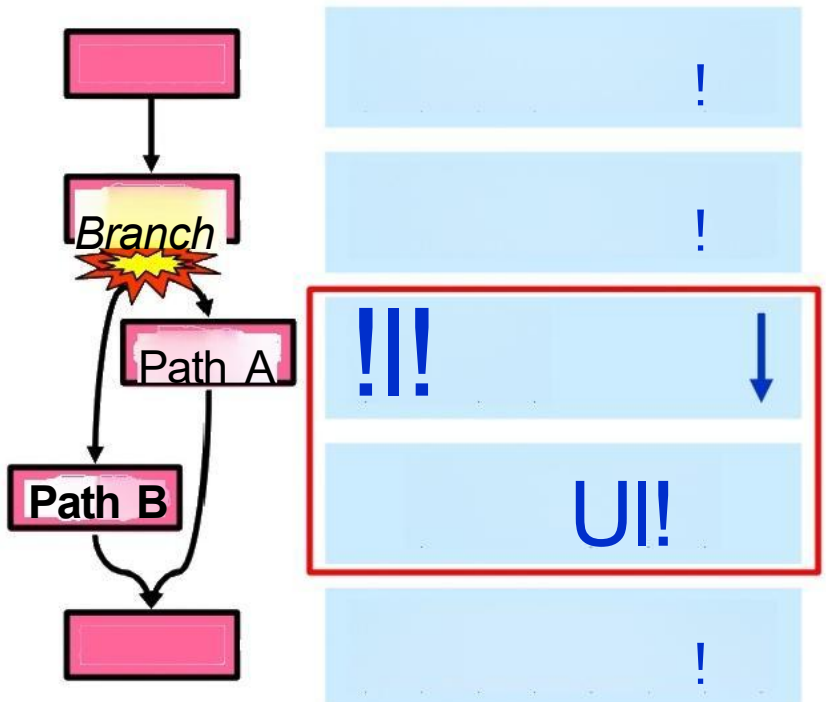


Slide credit:Tor Aamodt

Control Flow Problem in GPUs/SIMT

- A GPU uses a SIMD pipeline to save area on control logic
  - Groups scalar threads into warps

- Branch divergence occurs when threads inside warps branch to different execution paths



Applications optimal for CPU and GPU

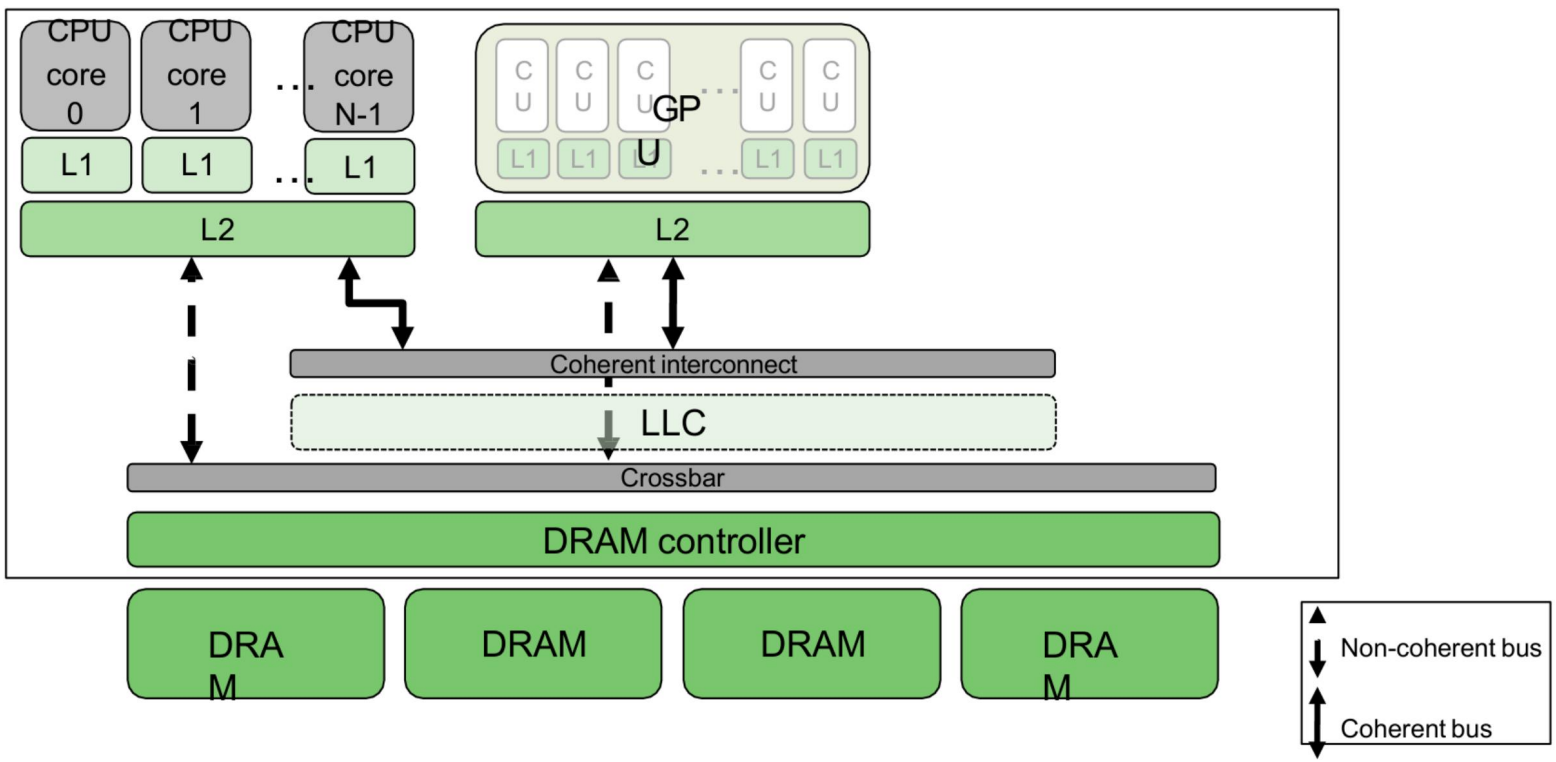
Strategies:Use Both CPU and GPU

- CPUs for sequential parts where latency matters
  -CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  -GPUs can be 10+X faster than CPUs for parallel code

adapted from UW CSEP 590, Dr. Hari Sadasivan

Heterogeneous Computing Systems

- The end of Moore's law created the need for heterogeneous **Systems**
  - More suitable devices for each type of workload
  - Increased performance and energy efficiency



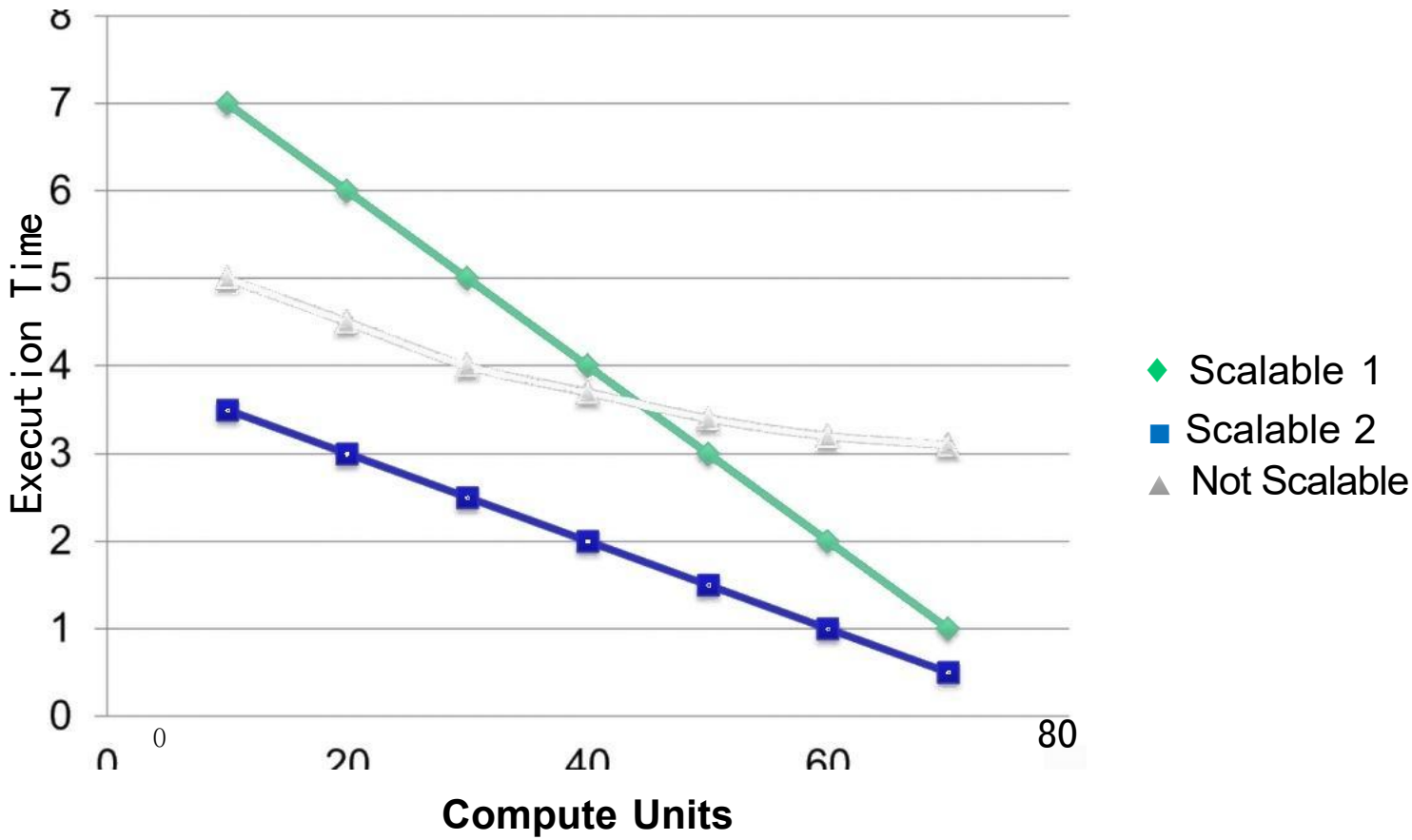
Chang+, "Collaborative Computing for Heterogeneous Integrated Systems," ICPE 2017.

Parallel Programming Workflow

- Identify compute-intensive parts of an application
- Adopt scalable algorithms
- Optimize data arrangements to maximize locality
- Performance Tuning
- Pay attention to code portability, scalability, and maintainability

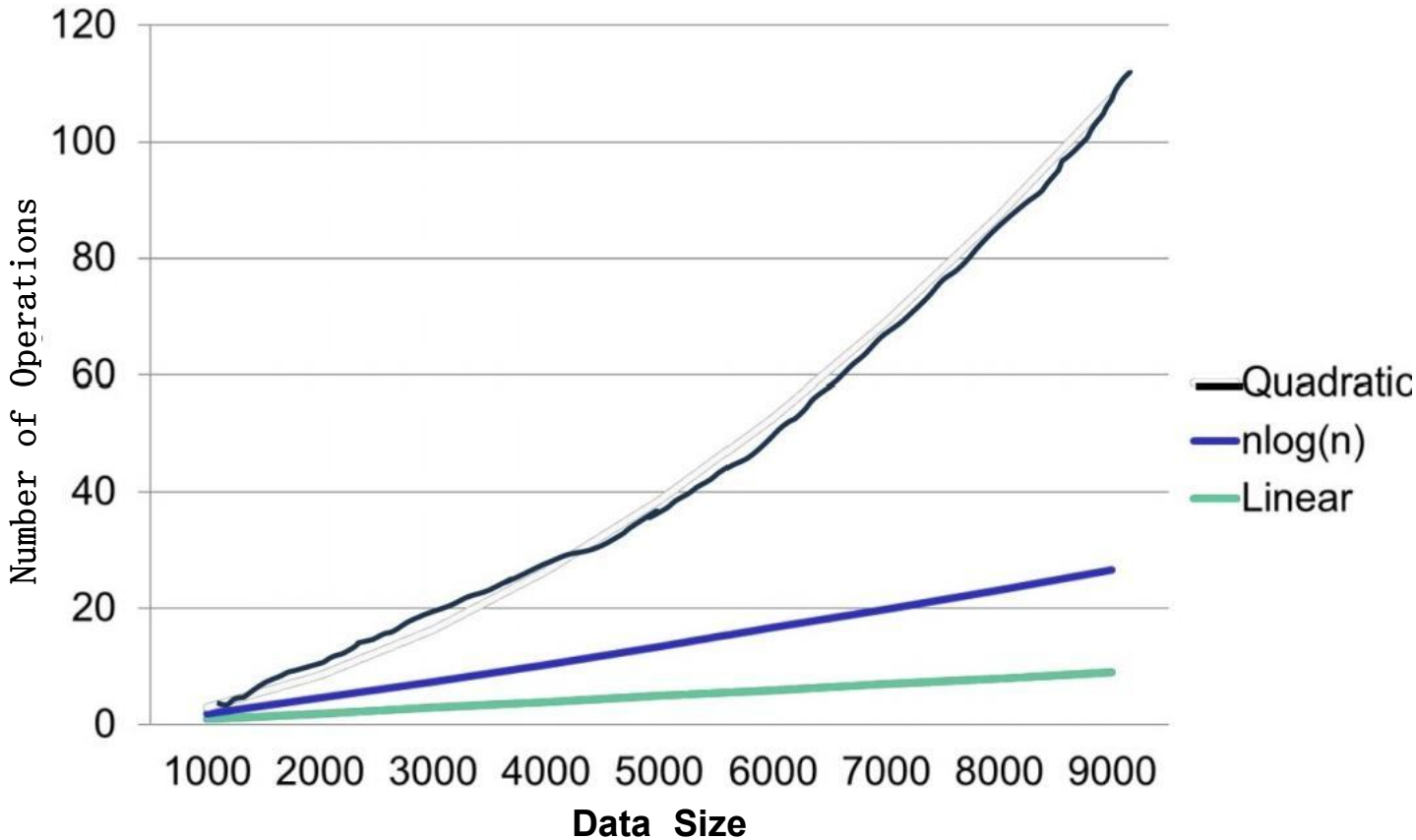
adapted from UW CSEP 590, Dr. Hari Sadasivan

Parallelism Scalability



adapted from UW CSEP 590, Dr. Hari Sadasivan

Algorithm Complexity and Data Scalability



adapted from UW CSEP 590, Dr. Hari Sadasivan

Load Balance

- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish

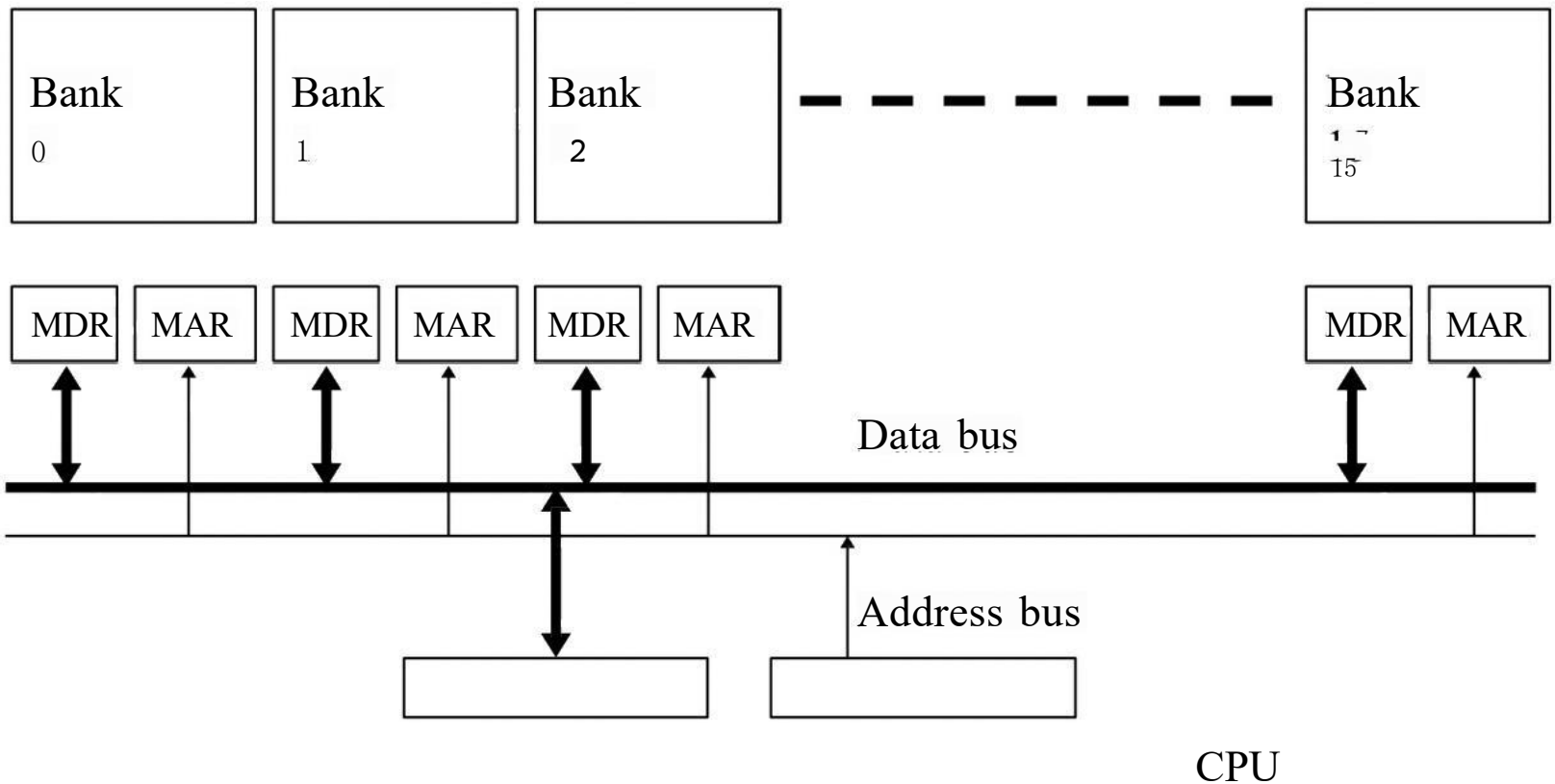
Massive Parallelism: Regularity

Regularity is the key for getting a goodscalability

adapted from UW CSEP 590, Dr. Hari Sadasivan

Memory Banking

Memory is divided into banks that can be accessed independently;
banks share address and data buses (to minimize pin cost)
Can start and complete one bank access per cycle
Can sustain N concurrent accesses if all N go to different banks



Global Memory Bandwidth

The memory bandwidth is limited

Global Memory Bandwidth

Ideal

Reality

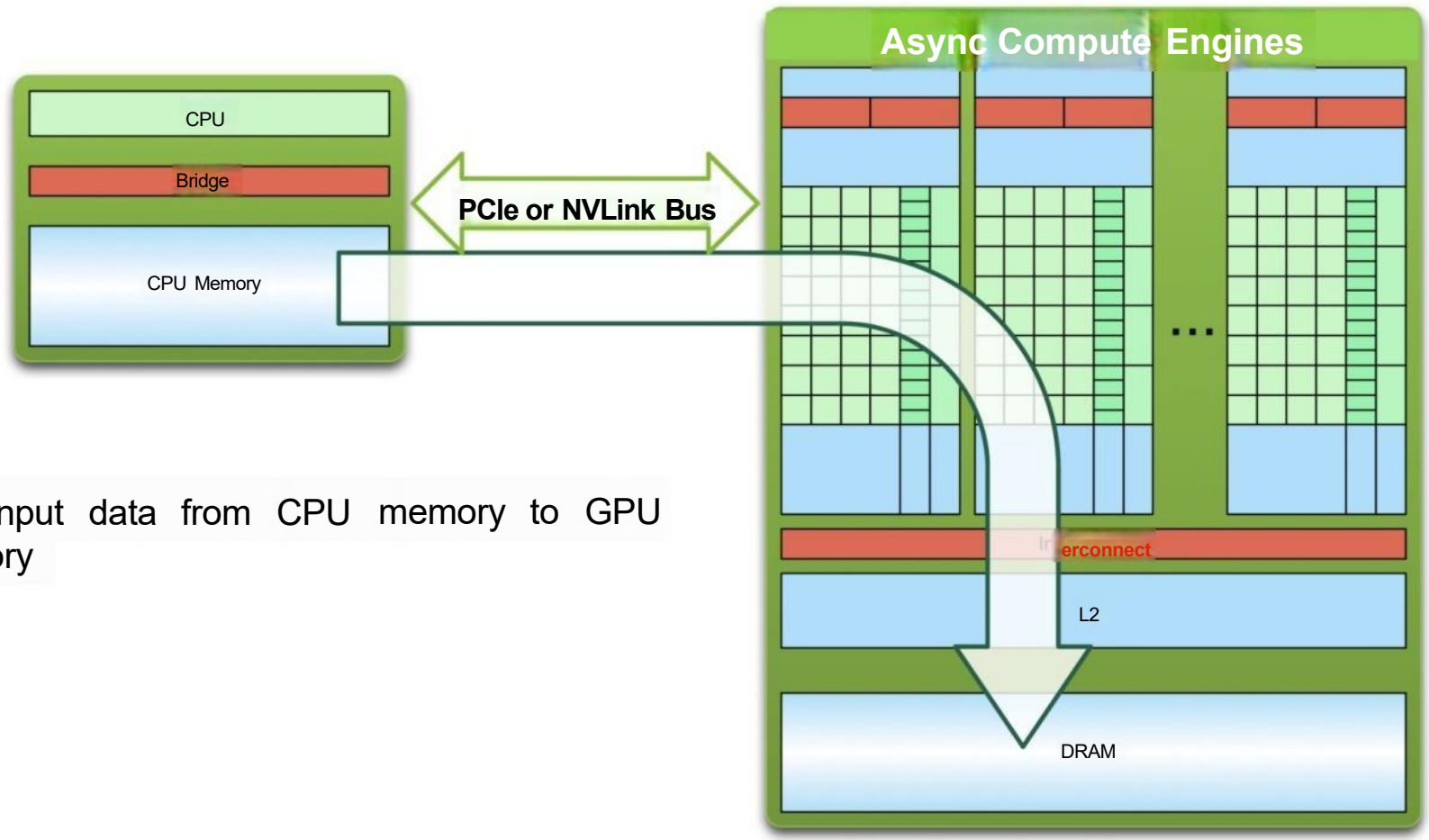
Conflicting Data Accesses

- Conflicting data accesses cause serialization and delays

  - Massively parallel execution cannot afford serialization

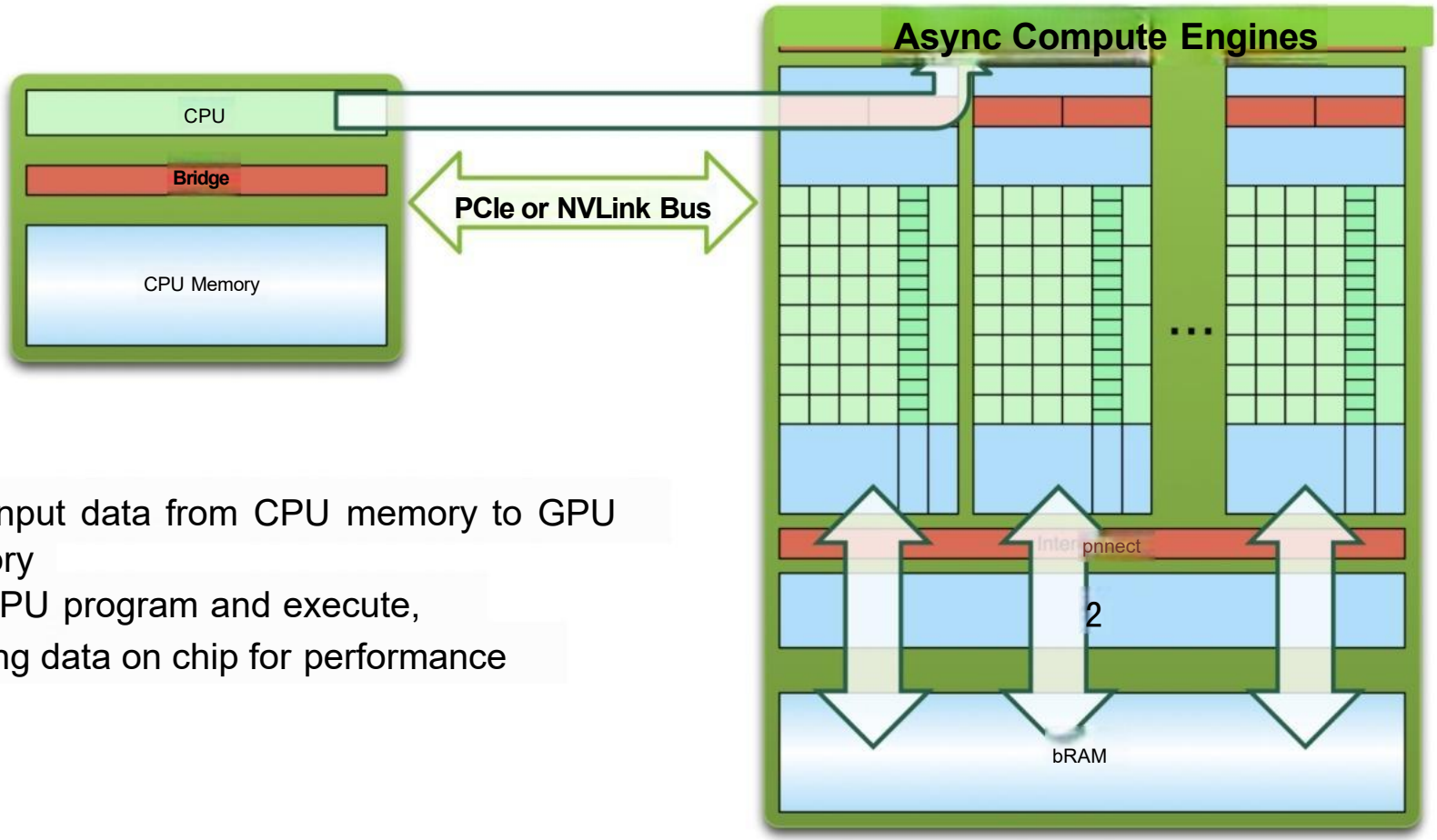
  - Contentions in accessing critical data causes serialization

Simple GPU Processing Workflow Pattern



1.Copy input data from CPU memory to GPU memory

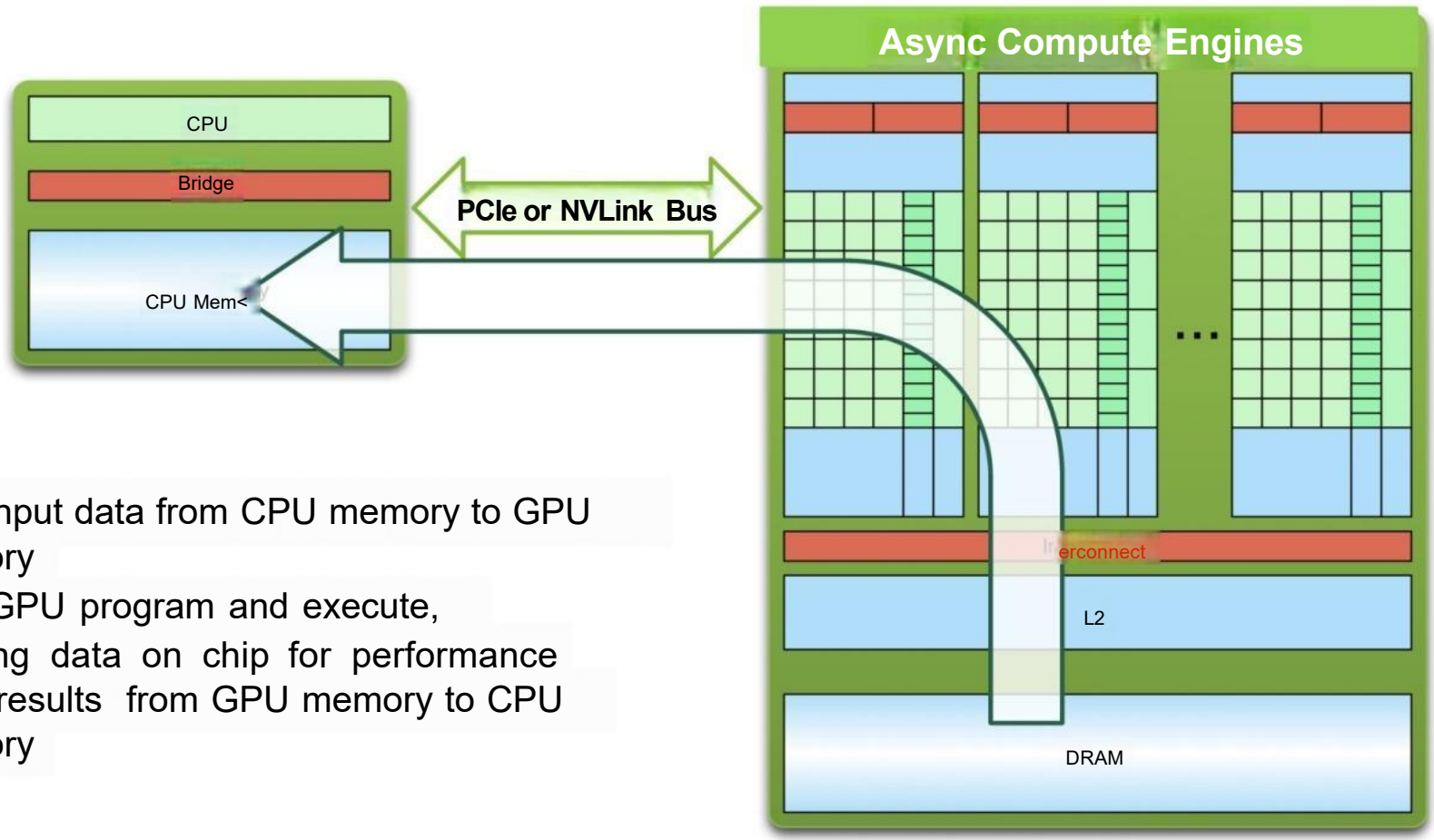
Simple GPU Processing Workflow Pattern



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

adapted from UW CSEP 590, Dr. Hari Sadasivan

Simple GPU Processing Workflow Pattern



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

adapted from UW CSEP 590, Dr. Hari Sadasivan

ROCm Software Ecosystem

Heterogeneous-compute Interface for Portability (HIP) is part of a larger software distribution called ROCm

- The ROCm package provides libraries and programming tools for developing HPC and ML applications on AMD GPUs

- All the ROCm environment and the libraries are provided from the supercomputer

- Heterogeneous System Architecture (HSA) runtime is an API that exposes the necessary interfaces to access and interact with the hardware driven by AMD GPU driver

ROCm SW stack for HPC&AI workloads

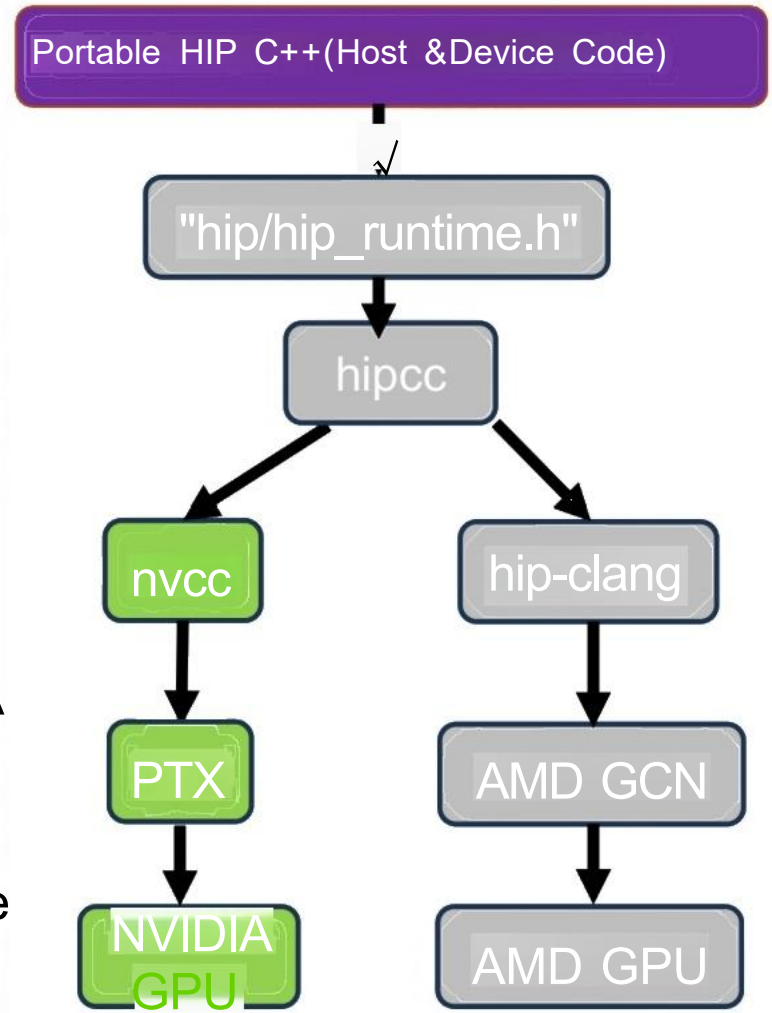
adapted from UW CSEP 590, Dr. Hari Sadasivan

AMD GPU Programming Concepts

Programming with HIP:
Kernels, blocks, threads, and more

What is AMD's HIP?

Heterogeneous-compute Interface for Portability,or HIP,is a C++runtime API &kernel language for AMD &NVIDIA devices.



HIP:

- Open-source.
- Provides an API for an application to leverage GPU acceleration for both AMD and CUDA devices.
- Syntactically similar to CUDA.Most CUDA API call can be converted in place:cuda ->hip
- Supports a strong subset of CUDA runtime functionality.

adapted from UW CSEP 590, Dr. Hari Sadasivan

A Tale of Host and Device

Source code in HIP has two flavors:
Host code and Device code

- The Host is the CPU
  Host code runs here
- Usual C++syntax and features
  - Entry point is the'main'function
  - HIP API can be used to create device buffers,move between host and device,and launch device code.

- The Device is the GPU
- Device code runs here
- C-like syntax
- Device codes are launched via "kernels"
- Instructions from the Host are enqueued into"streams"

adapted from UW CSEP 590, Dr. Hari Sadasivan

HIP API

- Device Management:
  - hipSetDevice(), hipGetDevice(), hipGetDeviceProperties()
- Memory Management
  - hipMalloc(), hipMemcpy(), hipMemcpyAsync(), hipFree()
- Streams
  - hipStreamCreate(), hipSynchronize(), hipStreamSynchronize(), hipStreamFree()
- **Events**
  - hipEventCreate(), hipEventRecord(), hipStreamWaitEvent(), hipEventElapsedTime()
- Device Kernels
  - __global__, __device__ ,hipLaunchKernelGGL()
- Device code
  - threadIdx,blockIdx,blockDim,__shared__
  - 200+math functions covering entire CUDA math library.
- Error handling
  - hipGetLastError(),hipGetErrorString()

adapted from UW CSEP 590, Dr. Hari Sadasivan

Kernels, memory, and structure of host code

adapted from UW CSEP 590, Dr. Hari Sadasivan

Device Kernels: The Grid

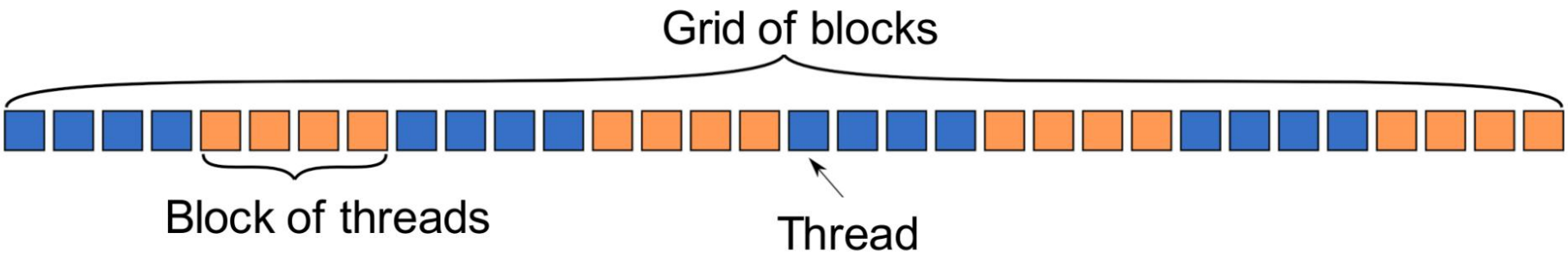
- In HIP, kernels are executed on a 3D "grid"
- The “grid” is what you can map your problem to
  - It's not a physical thing, but it can be useful to think that way
- AMD devices (GPUs) support 1D, 2D, and 3D grids, but most work maps well to 1D
- Each dimension of the grid partitioned into equal sized "blocks"
- Each block is made up of multiple “threads”
- The grid and its associated blocks are just organizational constructs
  - The threads are the things that do the work
- If you're familiar with CUDA already, the grid+block structure is very similar in HIP

Device Kernels: The Grid

Some Terminology:

CUDA	HIP	OpenCL™
grid	grid	NDRange
block	workgroup	work group
thread	work item /thread	work item
warp	wavefront	sub-group

The Grid: blocks of threads in 1D



Threads in grid have access to:

· Their respective block: blockIdx.x

· Their respective thread ID in a block: threadIdx.X

· Their block's dimension: blockDim.x

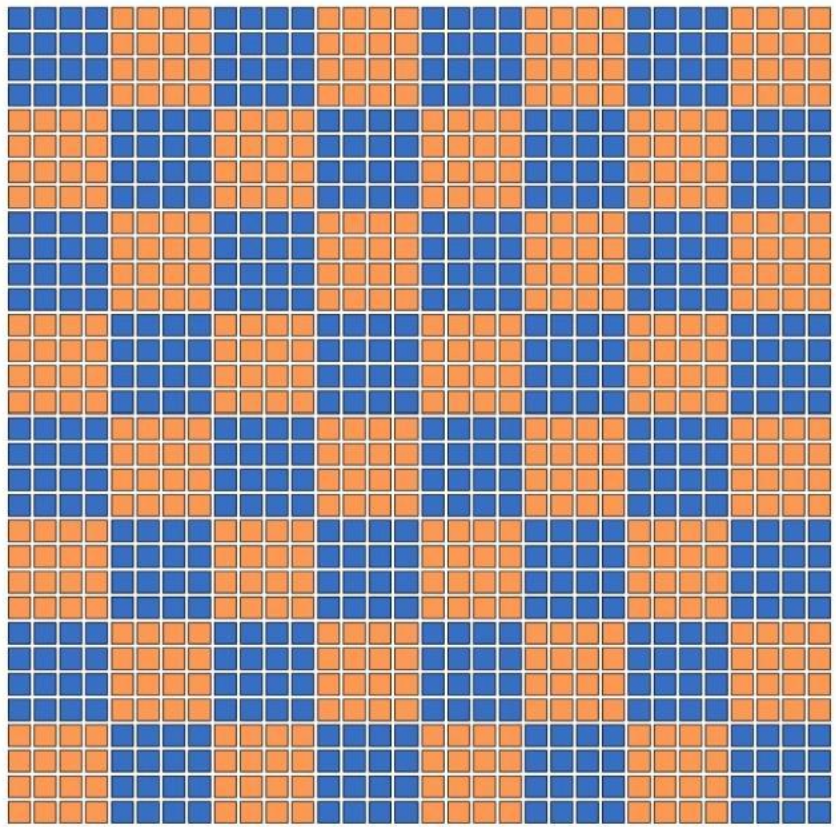
- The number of blocks in the grid: gridDim.x

The Grid: blocks of threads in 2D

- Each color is a block of threads
- Each small square is a thread
- The concept is the same in 1D and 2D
- In 2D each block and thread now has a two-dimensional index

Threads in grid have access to:

- Their respective block IDs: blockIdx.x, blockIdx.y
- Their respective thread IDs in a block: threadIdx.x, threadIdx.y



adapted from UW CSEP 590, Dr. Hari Sadasivan

Kernel Properties

A simple embarrassingly parallel loop:

```
for (int i= 0; i<N; i++)
   h_a[i] *= 2.0;
```

Can be translated into a GPU kernel:

```
__global__ void myKernel(int N, double *d_a)
{
   int i = threadIdx.x + blockIdx.x*blockDim.x;
   if (i<N) d_a[i] *= 2.0;
}
```

- A device function that will be launched from the host program is called a kernel &is declared with the __global__ attribute
- Kernels should be declared void
- All pointers passed to kernels must point to memory on the device
- All threads execute the kernel's body "simultaneously"
- Each thread uses its unique thread and block IDs to compute a global ID
- There could be more than N threads in the grid

adapted from UW CSEP 590, Dr. Hari Sadasivan

Kernel launch

Kernels are launched from the host:

```
dim3 threads(256,1,1);                  //3D dimensions of a block of threads
dim3 blocks((N+256-1)/256,1,1); //3D dimensions the grid of blocks

hipLaunchKernelGGL(myKernel // Kernel name (_global_void function)
                   blocks,          // Grid dimensions
                   threads.         // Block dimensions
                   0,               // Bytes of dynamic LDS space
                   0,               // Stream(O=NULL stream)
                   N, a);           // Kernel arguments
```

Analogous to CUDA kernel launch syntax:

```
myKernel<<<blocks,threads,0,O>>>(N,a);
```

SIMD operations

Why blocks and threads?

Natural mapping of kernels to hardware:

- Blocks are dynamically scheduled onto CUs
- All threads in a block execute on the same CU
- Threads in a block share LDS memory and L1 cache
- Threads in a block are executed in **64-wide** chunks called "wavefronts"
- Wavefronts execute on SIMD units(Single Instruction Multiple Data)
- If a wavefront stalls (e.g.data dependency)CUs can quickly context switch to another wavefront

A good practice is to make the block size a multiple of 64 and have several wavefronts (e.g., 256 threads)

Device Memory

The host instructs the device to allocate device memory and records a pointer to device memory

```
int main(){
  ...
  int N = 1000 ;
  size_t  Nbytes  =N*sizeof(double);
  double *h_a=(double *)malloc (Nbytes);  // Host memory

  double *d_a  = NULL;
  hipMalloc(&d_a, Nbytes);                // Allocate Nbytes on device

  ...

  free(h_a);                              // free host memory
  hipFree(d_a);                           // free device memory
}
```

Device Memory

The host queues memory transfers:

```
// copy data from host to device
hipMemcpy(d_a,h_a,Nbytes,hipMemcpyHostToDevice);

// copy data from device to host
hipMemcpy(h_a,d_a,Nbytes,hipMemcpyDeviceToHost);

// copy data from one device buffer to another
hipMemcpy(d_b,d_a,Nbytes,hipMemcpyDeviceToDevice);
```

Device Memory

Can copy strided sections of arrays:

```
hipMemcpy2D(d_a,          //pointer to destination
            DLDAbytes,    //pitch of destination array
            h_a,          //pointer to source
            LDAbytes      //pitch of source array
            Nbytes        //number of bytes in each row
            Nrows.        //number of rows to copy
            hipMemcpyHostToDevice);
```

Error Checking

·Most HIP API functions return error codes of type

hipError_t hipError_t status1 = hipMalloc(…) ;

hipError_t status2 = hipMemcpy(…) ;

·If API function was error-free,returns hipSuccess,otherwise returns an error code.

·Can also peek/get at last error returned with

hipError_t status3 =hipGetLastError();

hipError_t status4 = hipPeekLastError() ;

·Can get a corresponding error string using hipGetErrorString(status).Helpful for debugging, e.g.

```
#define  HIP_CHECK(command) { \
   hipError_t status = command; \
   if (status!=hipSuccess) { \
      std::cerr<<“Error:HIP reports”<<hipGetErrorString(status)<<std:endl; \
      std:abort();
   }
}
```

adapted from UW CSEP 590, Dr. Hari Sadasivan

Putting it alltogether

```
#include "hip/hip_runtime.h"
int main() {  int N =
    1000;
   size_t Nbytes =N*sizeof(double);
   double *h_a =(double*)malloc(Nbytes);                    // host memory
   double *d_a =NULL;
   HIP_CHECK(hipMalloc(&d_a,Nbytes));

   HIP_CHECK(hipMemcpy(d_a,h_a,Nbytes,hipMemcpyHostToDevice));          // copy data to device

   hipLaunchKernelGGL(myKernel,dim3((N+256-1)/256,1,1),dim3(256,1,1),0,0,N,d_a);//Launch kernel
   HIP_CHECK(hipGetLastError());

   HIP_CHECK(hipMemcpy(h_a,d_a,Nbytes,hipMemcpyDeviceToHost))

   free(h_a);                       //free host memory
   HIP_CHECK(hipFree(d_a));         //free device memory
```

```
__global__ void myKernel(int N,double *d_a) {
   int i = threadIdx.x+blockIdx.x*blockDim.x;
   if (i<N) {
            d_a[]]*=2.0; } }
```

```
#define HIP_CHECK(command){                          \
   hipError_t status =command;if                     \
   (status!=hipSuccess){                             \
     std:cerr<< "Error:HIP  reports"                 \
              <<hipGetErrorString(status)\           \
              <<std::endl                            \
     std:abort();}}
```

adapted from UW CSEP 590, Dr. Hari Sadasivan

Vector Addition

```
__global__ void vecAddkernel(float*A_d, float*B_d, float*C_d, int n) {
    int i=threadIdx.x+blockDim.x*blockIdx.x;
    if (i<n) {
        C_d[i]=A_d[i]+B_d[i];
    }
}
```

Vector Addition

```
void vecAdd(float*A,float*B,float*C,int n) {
    int size =n* sizeof(float) ;
    float*A_d,B_d,C_d:

    // Allocate device memory
    hipMalloc((void **)&A_d,size);
    hipMalloc((void **)&B_d,size);
    hipMalloc((void **)&C_d,size);

    // Transfer A and B to device memory
    hipMemcpy(A_d,A,size,hipMemcpyHostToDevice);
    hipMemcpy(B_d,B,size,hipMemcpyHostToDevice);

    // Kernel invocation code
    vecAddKernel<<<ceil(n/256.0),256>>>(A_d,B_d,C_d,n);
    hipDeviceSynchronize();

    // Transfer C from device to host
    hipMemcpy(C,C_d,size,hipMemcpyDeviceToHost);

    // Free device memory for A,B,C
    hipFree(A_d);hipFree(B_d);hipFree (C_d);
}
```

adapted from UW CSEP 590, Dr. Hari Sadasivan

Another way to launch kernel

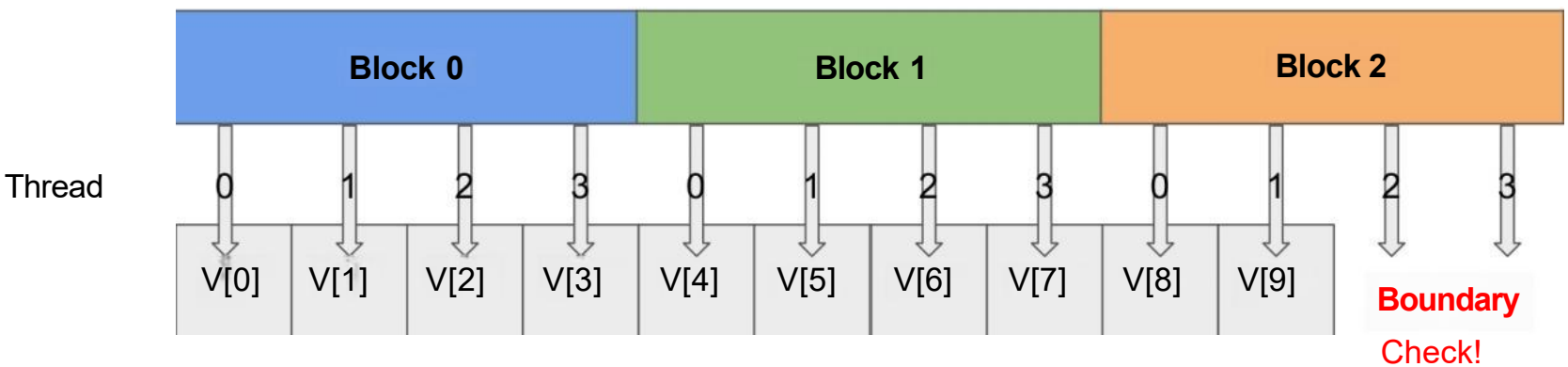
```
int vecAdd(float*A,float*B,float*C,int n){
    ....
    // Kernel invocation code
    dim3  DimGrid(ceil(n/256),1,1);
    dim3 DimBlock(256,1,1);

    vecAddKernel<<<DimGrid,DimBlock>>>(A_d,B_d,C_d,n)
    hipDeviceSynchronize();

    // Transfer C from device to host
    ...
}
```

Mapping to the data

Suppose we use 1d thread blocks of size 4



i=blockIdx.x*blockDim.x+threadIdx.x

Questions

1. How many floating operations are being performed in the vector add kernel? Give your answer in terms of N and explain.

2. How many global memory reads and writes are being performed by the vector add kernel? Give your answer in terms of N.

Questions

1. How many floating operations are being performed in the vector add kernel? Give your answer in terms of N and explain.

   N, one for each pair of input vector elements

2. How many global memory reads and writes are being performed by the vector add kernel? Give your answer in terms of N.

   Reads: 2N, one for each of the two input vectors elements.

   Writes: N, one for each output vector element.

Questions

3.We want to use each thread to calculate 4 output elements of vector addition. Each block processes 4*blockDim.x consecutive elements that form 4 sections.All threads in each block will first process a section with each thread processing one element.They will then all move to the next section with each thread processing one element.For each section,consecutive threads should process consecutive elements.

What would be the expression for mapping the thread/block indices to i,the data indexof the **first** element to be processed by each thread?Suppose we launch 1D blocks and give your answers in terms of blockIdx.x,blockDim.x,and threadIdx.x.

Questions

3.We want to use each thread to calculate 4 output elements of vector addition. Each block processes 4*blockDim.x consecutive elements that form 4 sections.All threads in each block will first process a section with each thread processing one element.They will then all move to the next section with each thread processing one element.For each section,consecutive threads should process consecutive elements.

What would be the expression for mapping the thread/block indices to i,the data indexof the **first** element to be processed by each thread?Suppose we launch 1D blocks and give your answers in terms of blockIdx.x,blockDim.x,and threadIdx.x.

Ans:i=blockDim.x*4*blockIdx.x+threadIdx.x

Function Qualifiers

hipcc makes two compilation passes through source code.One to compile host code, and one to compile device code.

- __global__ functions:
  - These are entry points to device code,called from the host
  - Code in these regions will execute on SIMD units

- __device__ functions:
  - Can be called from __global__ and other __device__ functions.
  - Cannot be called from host code.
  - Not compiled into host code-essentially ignored during host compilation pass

- __host__ __device__ functions:
  - Can be called from __global__ __device__, and host functions.
  - Will execute on SIMD units when called from device code!

Memory declarations in Device Code

· Malloc/free not supported in device code.

· Variables/arrays can be declared on the stack.

· Stack variables declared in device code are allocated in registers and are private to each thread.

· Threads can all access common memory via device pointers,but otherwise do not share memory.

-Important exception: __shared__ memory

·Stack variables declared as __shared__:

- Allocated once per block in LDS memory
- Shared and accessible by all threads in the same block
- Access is faster than device global memory (but slower than register)
- Must have size known at compile time

adapted from UW CSEP 590, Dr. Hari Sadasivan

Thread Synchronization

* __syncthreads() :

  - Blocks a wavefront from continuing execution until all wavefronts have reached __syncthreads()

  - Memory transactions made by a thread before **__syncthreads()** are visible to all other threads in the block after **__syncthreads()**

  - Can have a noticeable overhead if called repeatedly

Shared Memory

```
__global__ void reverse(double *d_a){
    __shared__ double s_a[256]; // array of doubles, shared in this block

    int tid =threadIdx.x;
    s_a[tid] = d_a[tid];                // each thread fills one entry

   // all wavefronts must reach this point before any wavefront is allowed to continue.
                                        // something is missing here…
    _syncthreads();

    d_a[tid] = s_a[255-tid]; // write out array in reverse order
}

int main() {
    ...
    hipLaunchKernelGGL(reverse,dim3(1),dim3(256),0,0,d_a); // Launch    kernel
    ...
}
```

Device Management

Multiple GPUs in system? Multiple host threads/MPI ranks? What device are we running on?

- Host can query number of devices visible to system:

```
int numDevices =0;  hipGetDeviceCount(&numDevices);
```

- Host tells the runtime to issue instructions to a particular device:

```
int deviceID  =0; hipSetDevice(deviceID);
```

- Host can query what device is currently selected:

```
hipGetDevice(&deviceID);
```

- The host can manage several devices by swapping the currently selected device during runtime.
- MPI ranks can set different devices or over-subscribe (share) devices.

Device Properties during run-time

The host can also query a device's properties:

```
hipDeviceProp_t props;
hipGetDeviceProperties(&props,deviceID);
```

hipDeviceProp_t is a struct that contains useful fields like the device's name,warpsize, max workgroup size, max grid size, #CUs, cache, clock speed, and GCN arch architecture.

-See “hip/hip_runtime_api.h” for full list of fields.

Querying System from terminal

- rocminfo:Queries and displays information on the system's hardware
  - -More info at: https://github.com/RadeonOpenCompute/rocminfo
- Querying ROCm version:
  - -If you install ROCm in the standard location (/opt/rocm)version info is at: /opt/rocm/.info/version-dev
  - -Can also run the command'dkms status'and the ROCm version will be displayed
- rocm-smi:Queries and sets AMD GPU frequencies,power usage,and fan speeds
  - -sudo privileges are needed to set frequencies and power limits
  - -sudo privileges are not needed to query information
  - -Get more info by running'rocm-smi -h'or looking at: https://github.com/RadeonOpenCompute/ROC-smi

```
§/opt/rocm/bin/rocm-smi
        ============R0Cm        System        Management        Interface====================

GPU    Temp     AvgPwr    SCLK          MCLK          Fan     Perf        PwrCap        VRAM%   GPU%
1      38.0c    18.0W     1440Mhz   !   945Mhz        0.0%    manual      220.0W        0%      0%

                          ==========End   of   ROCm   SMI   Log   ==========
```

adapted from UW CSEP 590, Dr. Hari Sadasivan

Blocking vs Nonblocking API functions

- The kernel launch function,hipLaunchKernelGGL,is **non-blocking** for the host.
  - -After sending instructions/data,the host continues immediately while the device executes the kernel
  - -If you know the kernel will take some time,this is a good area to do some work (i.e.MPI comms)on the host

- However, hipMemcpy is **blocking.**
  - -The data pointed to in the arguments can be accessed/modified after the function returns.

- The non-blocking version is hipMemcpyAsync

```
hipMemcpyAsync(d_a,h_a,Nbytes,hipMemcpyHostToDevice,stream);
```

- Like hipLaunchKernelGGL,this function takes an argument of type hipStream_t

- It is not safe to access/modify the arguments of hipMemcpyAsync without some sort of synchronization.

Streams

- A stream in HIP is a queue of tasks (e.g.kernels,memcpys,events).
  - -Tasks enqueued in a stream **complete in order on that stream.**
  - -Tasks being executed in different streams are allowed to overlap and share device resources.

Streams are created via:

```
hipStream_t stream;
hipStreamCreate(&stream);
```

- And destroyed via:

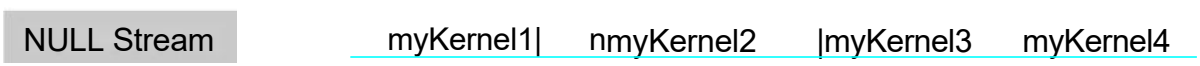
```
hipStreamDestroy(stream);
```

- Passing 0 or NULL as the hipStream_t argument to a function instructs the function to execute on a stream called the 'NULL Stream':
  - -No task on the NULL stream will begin until **all previously enqueued tasks in all other streams have completed.**
  - -Blocking calls like hipMemcpy run on the NULL stream.

Streams

· Suppose we have 4 small kernels to execute:

```
hipLaunchKernelGGL(myKernel1 dim3(1),dim3(256),0,0,256,d_a1);
hipLaunchkernelGGL(myKernel2 dim3(1),dim3(256),0,0,256,d_a2);
hipLaunchKernelGGL(myKernel3 dim3(1),dim3(256),0,0,256,d_a3);
hipLaunchKernelGGL(myKernel4 dim3(1),dim3(256),0,0,256,d_a4);
```



·Even though these kernels use only one block each,they'll execute in serial on the NULL stream:

Streams

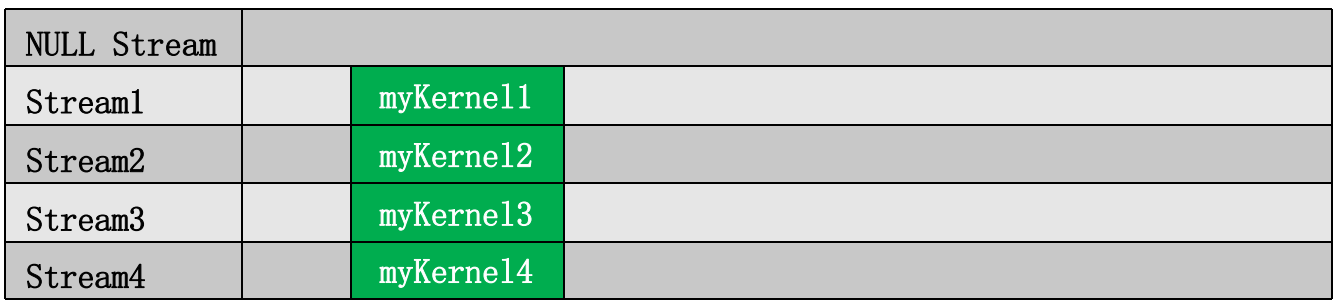
With streams we can effectively share the GPU's compute resources:

```
hipLaunchKernelGGL(myKernel1 dim3(1),dim3(256),0,stream1,256,d_a1);

hipLaunchKernelGGL(myKernel2 dim3(1),dim3(256),0,stream2,256,d_a2);

hipLaunchKernelGGL(myKernel3 dim3(1),dim3(256),0,stream3,256,d_a3);

hipLaunchKernelGGL(myKernel4 dim3(1),dim3(256),0,stream4,256,d_a4);
```



Note 1:Check that the kernels modify different parts of memory to avoid data races.

Note 2:With large kernels,overlapping computations may not help performance.

Streams

· There is another use for streams besides concurrent kernels:
  -Overlapping kernels with data movement.

·AMD GPUs have separate engines for:
  -Host->Device memcpys
  -Device->Host memcpys
  -Compute kernels

·These three different operations can overlap without dividing the GPU's resources.
  -The overlapping operations should be in separate, non-NULL, streams.
  -The host memory should be **pinned.**

Pinned Memory

Host data allocations are pageable by default.The GPU can directly access Host data if it is pinnec instead.

·Allocating pinned host memory:

```
double *h_a = NULL;
hipHostMalloc(&h_a,Nbytes);
```

·Free pinned host memory:

```
hipHostFree(h_a);
```

·Host<->Device memcpy **bandwidth increases significantly when host memory is pinned.**

-It is good practice to allocate host memory that is frequently transferred to/from the device as pinned memory.

Streams

Suppose we have 3 kernels which require moving data to and from the device:

```
hipMemcpy(d_a1,h_a1,Nbytes,hipMemcpyHostToDevice));
hipMemcpy(d_a2,h_a2,Nbytes,hipMemcpyHostToDevice));
hipMemcpy(d_a3,h_a3,Nbytes,hipMemcpyHostToDevice));

hipLaunchKernelGGL (myKernel1,blocks,threads,0,0,N,d_a1);
hipLaunchkernelGGL (myKernel2,blocks,threads,0,0,N,d_a2);
hipLaunchkernelGGL (myKernel3,blocks,threads,0,0,N,d_a3);

hipMemcpy(h_a1,d_a1,Nbytes,hipMemcpyDeviceToHost);
hipMemcpy(h_a2,d_a2,Nbytes,        hipMemcpyDeviceToHost);
hipMemcpy(h_a3,d_a3,Nbytes,hipMemcpyDeviceToHost);
```

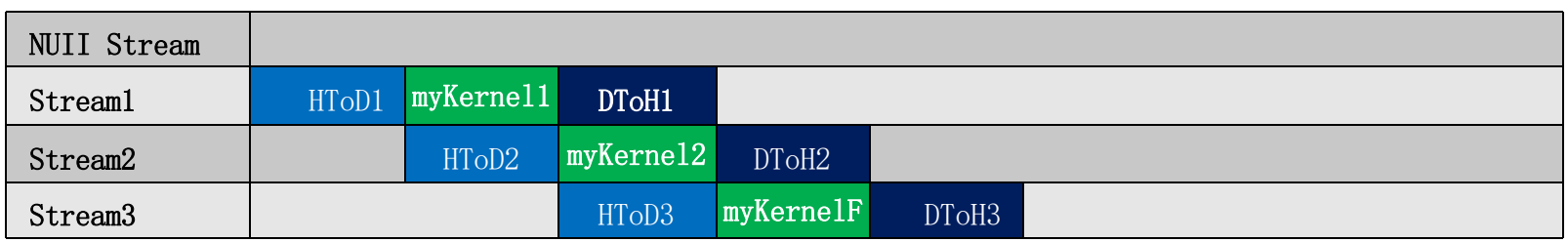
Streams

Changing to asynchronous memcpys and using streams:

```
hipMemcpyAsync(d_a1,h_a1,Nbytes,hipMemcpyHostToDevice, stream1);
hipMemcpyAsync(d_a2,h_a2,Nbytes,hipMemcpyHostToDevice,  stream2);
hipMemcpyAsync(d_a3,h_a3,Nbytes,hipMemcpyHostToDevice, stream3);

hipLaunchKernelGGL (myKernel1,blocks,threads,0,stream1,N,d_a1);
hipLaunchKernelGGL (myKernel2,blocks,threads,0,stream2,N,d_a2);
hipLaunchKernelGGL (myKernel3,blocks,threads,0,stream3,N,d_a3);

hipMemcpyAsync(h_a1,d_a1,Nbytes,hipMemcpyDeviceToHost,stream1);
hipMemcpyAsync(h_a2,d_a2,Nbytes,hipMemcpyDeviceToHost,stream2);
hipMemcpyAsync(h_a3,d_a3,Nbytes,hipMemcpyDeviceToHost, stream3);
```



Synchronization

How do we coordinate execution on device streams with host execution?
Need some synchronization points.

- hipDeviceSynchronize();
  - -Heavy-duty sync point.
  - -Blocks host until all work in all device streams has reported complete.

- hipStreamSynchronize (stream);
  - -Blocks host until all work in stream has reported complete.

Can a stream synchronize with another stream? For that we need 'Events'.

Events

A hipEvent_t object is created on a device via:

```
hipEvent_t event;
hipEventCreate(&event);
```

We queue an event into a stream:

```
hipEventRecord(event,stream);
```

- The event records what work is currently enqueued in the stream.
- When the stream's execution reaches the event,the event is considered'complete'.

At the end of the application,event objects should be destroyed:

```
hipEventDestroy(event);
```

Events

What can we do with queued events?

- hipEventSynchronize(event);
  - -Block host until event reports complete.
  - -Only a synchronization point with respect to the stream where event was enqueued.

- hipEventElapsedTime(&time,startEvent,endEvent);
  - -Returns the time in ms between when two events, startEvent and endEvent, completed
  - -Can be very useful for timing kernels/memcpys

- hipStreamWaitEvent(stream,event);
  - -Non-blocking for host.
  - -Instructs all future work submitted to stream to wait until event reports complete.
  - -Primary way we enforce an ordering'between tasks in separate streams.

Questions

1.How would one declare a variable err that can appropriately receive the returned value of a HIP API call?

Questions

1.How would one declare a variable err that can appropriately receive the returned value of a HIP API call?

Ans:hipError_t err ={stmt};where {stmt}is a HIP API call such as hipMalloc()or a kernel function call

Questions

2.If we want to copy 5000 bytes of data from host array h_A(h_A is a pointer to element 0 of the source array)to device array d_A(d_A is a pointer to element 0 of the destination array),what would be an appropriate API call for the data copy in HIP?

Questions

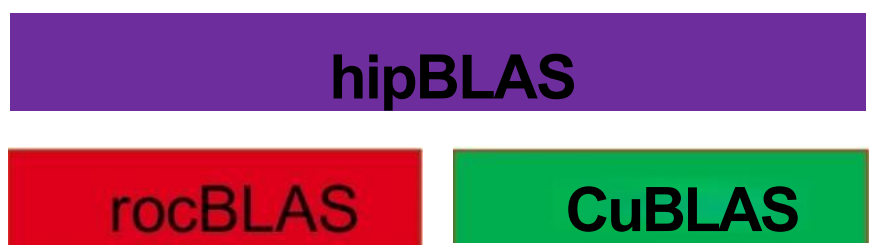
2.If we want to copy 5000 bytes of data from host array h_A(h_A is a pointer to element 0 of the source array)to device array d_A(d_A is a pointer to element 0 of the destination array),what would be an appropriate API call for the data copy in HIP?

Ans:hipMemcpy(d_A,h_A,5000,hipMemcpyHostToDevice);

AMD GPU Libraries

·A note on naming conventions:
  -roc*->AMGCN library usually written in HIP
  -cu*->NVIDIA PTX libraries
  -hip*->usually interface layer on top of roc*/cu*backends

· hip*libraries:
  -Can be compiled by hipcc and can generate a call for the device you have:
    -hipcc->hip-clang->AMD GCN ISA
    -hipcc->nvcc(inlined)->NVPTX
  -Just a thin wrapper that marshals calls off to a "backend"library:
    -corresponding roc*library backend containing optimized GCN
    -corresponding cu*library backend containing NVPTX for NVIDIA devices
  -E.g.,hipBLAS is a marshalling library:



adapted from UW CSEP 590, Dr. Hari Sadasivan

Why libraries?

- Code reuse
- High Performance
  - Maximize compute
  - Maximize memory bandwidth
- No need to deal with low level GPU code

Math library equivalents

CUBLAS	**ROCBLAS**	Basic Linear Algebra Subroutines
CUFFT	**ROCFFT**	Fast Fourier Transforms
THRUST	**ROCTHRUST**	STL Library for parallel algos
CUB	**ROCPRIM**	Optimized Parallel Primitives
EIGEN	**EIGEN**	C++Template Library for Linear Algebra

MORE INFO AT:GITHUB.COM/ROCM-DEVELOPER-TOOLS/HIP □HIP_PORTING_GUIDE.MD

adapted from UW CSEP 590, Dr. Hari Sadasivan

HIP Libraries

- rocBLAS
- rocSparse
- rocFFT
- rOcRAND
- rocSolver
- RCCL
- MIOpen

ROC vs HIP Libraries

- ROC libraries
  - rocBLAS
  - rocSparse
  - rocFFT
  - rocRAND
- Emphasize

Native Performance

- HIP libraries
  - hipBLAS
  - hipSparse
  - hipFFT
  - hipRAND
- Emphasize

Interoperability

adapted from UW CSEP 590, Dr. Hari Sadasivan

BLAS libraries

- **Level 1-scalar-vector or vector-vector operations**
  - SSCAL-x=a*x
  - SAXPY-y=a*x+y
  - SDOT -dot product
- **Level 2**
- **Level 3**

BLAS Libraries

- **Level 1-scalar-vector or vector-vector operations**
- **Level 2-vector-matrix operations**
  - SGEMV-matrix vector multiplication
  - STRMV-triangular matrix vector multiplication
  - SSYR-A=alpha*x*x¹+A
- **Level 3**

adapted from UW CSEP 590, Dr. Hari Sadasivan

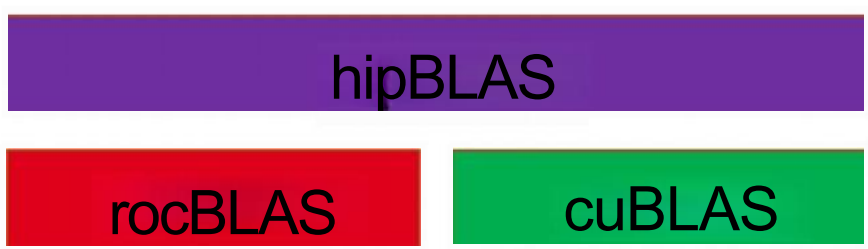
BLAS Libraries

- Level 1 - scalar-vector or vector-vector operations
- Level 2 - vector-matrix operations
- Level 3 - matrix-matrix operations
  - SGEMM - matrix matrix multiplication

adapted from UW CSEP 590, Dr. Hari Sadasivan

AMD GPU Libraries:BLAS

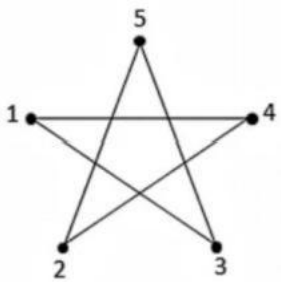
· rocBLAS-'sudo apt install rocblas`
  -Source code: https://github. com/ROCmSoftwarePlatform/rocBLAS -
  Documentation: https://rocblas. readthedocs. io/en/latest/
  -Basic linear algebra functionality
    -axpy,gemv,trsm,etc
  -Use hipBLAS if you need portability between AMD and NVIDIA devices
· hipBLAS -`sudo apt install hipblas`
  -Documentation: https://github. com/ROCmSoftwarePlatform/hipBLAS/wiki/Exported-functions
  -Use this if you need portability between AMD and NVIDIA
  -It is just a thin wrapper:
    -It can dispatch calls to rocBLAS for AMD devices
    -It can dispatch calls to cuBLAS for NVIDIA devices



rocSparse

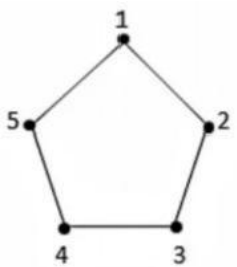
A library for sparse linear algebra operations

Sparse Data



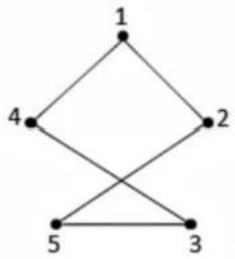
	1	2	3	4	5
1					
2					
3					
4					
5					

(a)



		2	3	4	5
1					
2					
3					
4					
5					

(b)

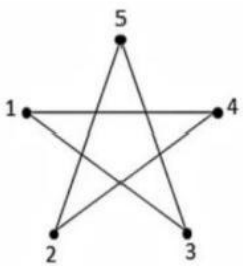


		2	3	4	5
		1		1	
2					1
3					
4					
5					

(c)

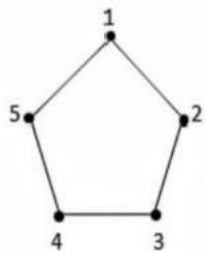
Sparse Data

1



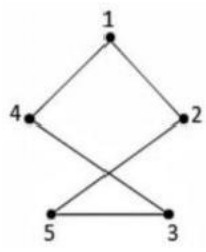
		2	3	4	5
3					
4					
5					

(a)



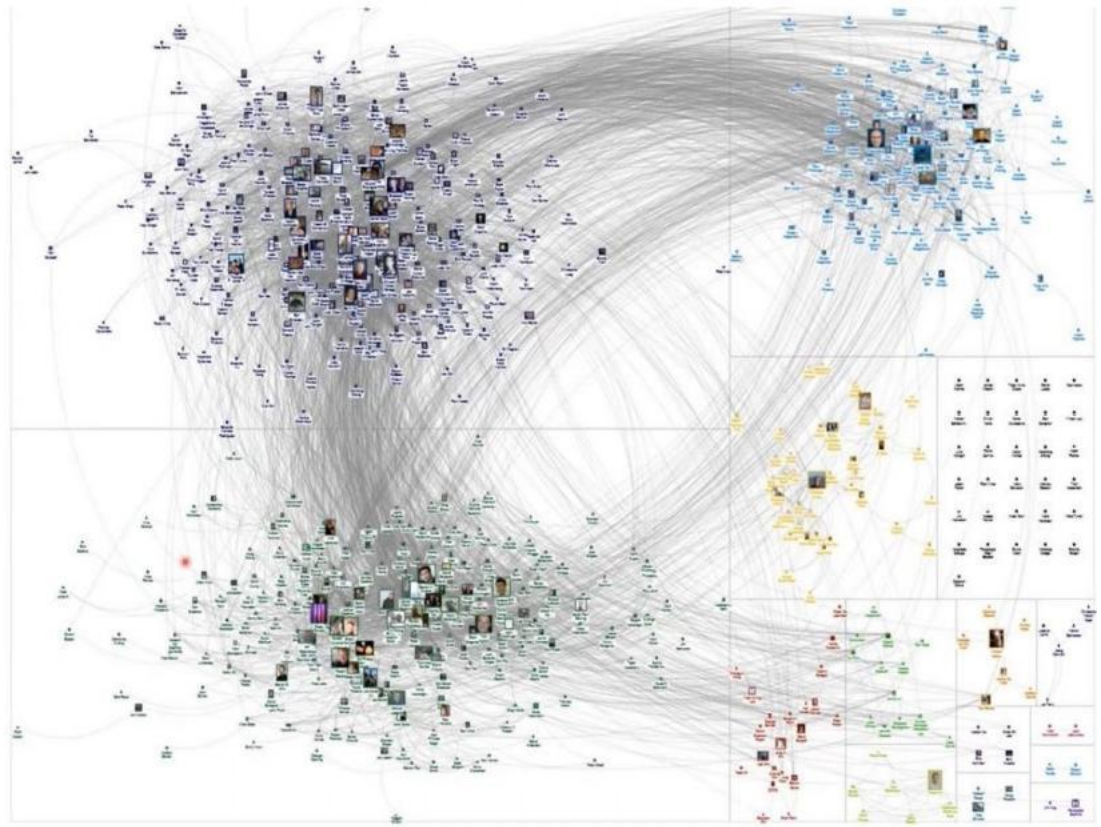
		2	3	4	5
1					
2			1		
3		1			
4			1		
5					

(b)



		2	3	4	5
1				1	
2					
3				1	
4					
5					

(c)



Prof.Yifan Sun,Prof.David R.Kaeli and Dr.Trinayan Baruah

The Curse of Dimensions

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

16, 384×16, 384×4=1GB

Sparse Matrix Compression

- The Coordinate Representation (COO)
- The Compressed Sparse Row (CSR)
- BSR
- GEBSR
- ELL
- HYB

rocSparse supports BLAS-like operations

rocSparse APIs

- **Level 1**
  - rocsparse_saxpyi()
- **Level 2**
  - rocsparse_scoomv()
  - rocsparse_scsrmv()
- **Level 3**
  - rocsparse_sbsrmm()
  - rocsparse_scsrmm()

rocFFT

Fast Fourier Transform

Prof.Yifan Sun,Prof.David R.Kaeli and Dr.Trinayan Baruah

Discrete Fourier Transform (DFT) is complex

$$\tilde{x}_j = \sum_{k=0}^{n-1} x_k exp(\pm i \frac{2\pi jk}{n}) \text{ for } j = 0, 1, ..., n-1$$

$$\tilde{x}_{jk} = \sum_{q=0}^{m-1} \sum_{r=0}^{n-1} x_{rq} exp(\pm i \frac{2\pi jr}{n}) exp(\pm i \frac{2\pi kq}{m})$$

for j=0,1,…,n-1 and k=0,1,…,m-1

$0(N^2)$

FFT is simpler

$$\tilde{x}_j = \sum_{k=0}^{n-1} x_k exp(\pm i \frac{2\pi jk}{n}) \text{ for } j = 0, 1, ..., n-1$$

$$\tilde{x}_{jk} = \sum_{q=0}^{m-1} \sum_{r=0}^{n-1} x_{rq} exp(\pm i \frac{2\pi jr}{n}) exp(\pm i \frac{2\pi kq}{m})$$

for j=0,1,….,n-1 and k=0,1,…,m-1

$$O(N^2) \longrightarrow O(N \log N)$$

FFT for image transformation

Edge detection

Original image

Power spectrum with mask that filters low frequencies

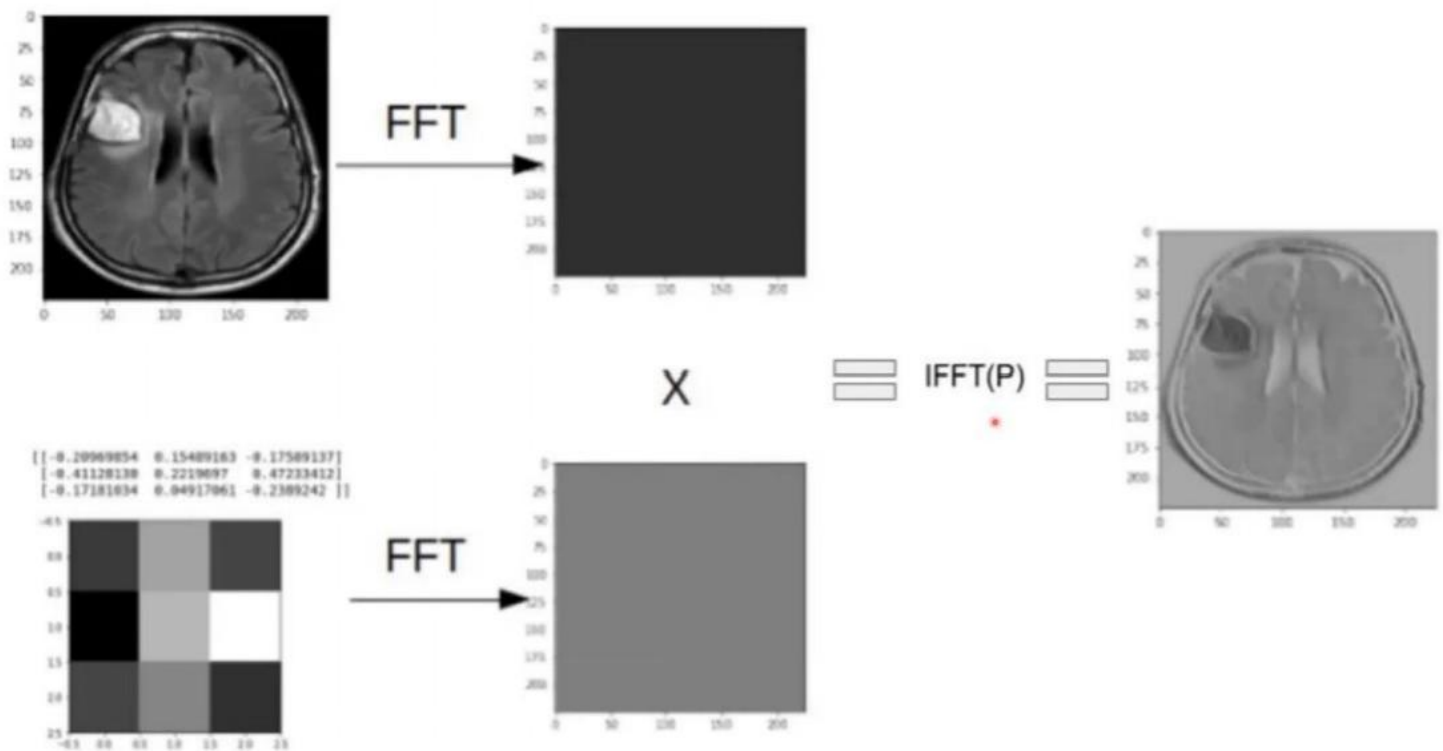
Result of inverse transform

Power spectrum with mask that passes low frequencies

Result of inverse transform

Smoothing

FFT for converting convolution to matrix multiplication

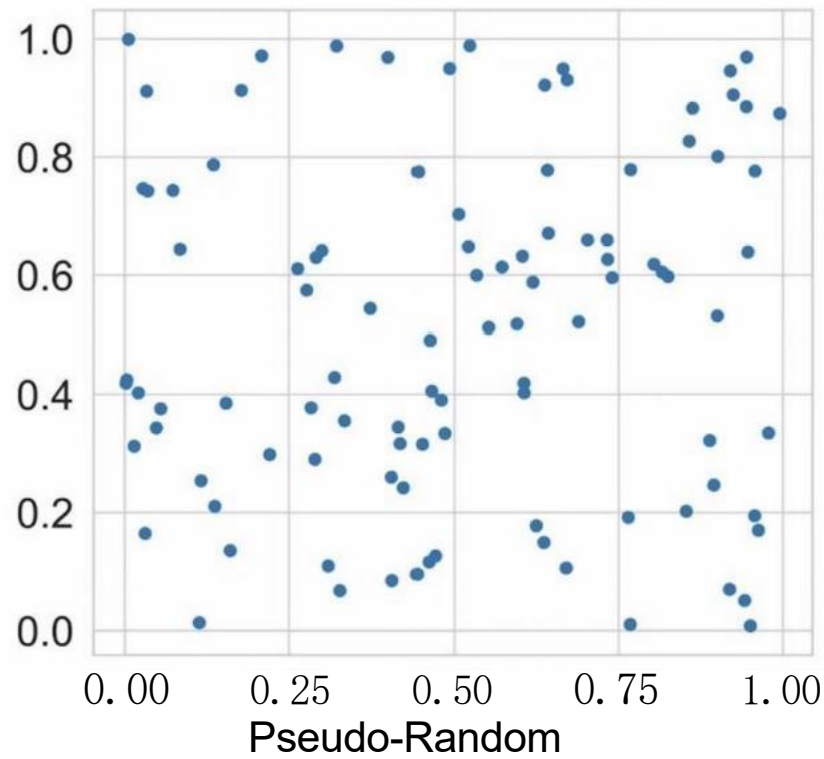


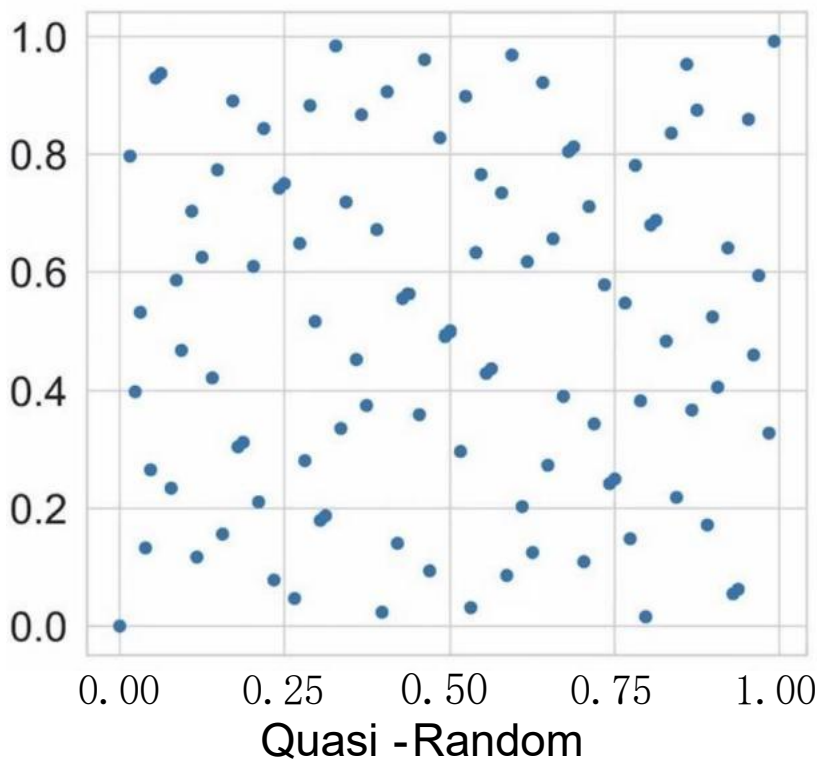
Prof.Yifan Sun,Prof.David R.Kaeli and Dr.Trinayan Baruah

rocRand

Generating Random Numbers

Pseudo vs Quasi-random





Prof.Yifan Sun,Prof.David R.Kaeli and Dr.Trinayan Baruah

rocRAND APIs

API	Data Type	Distribution	Range
rocrand_generate	uint32	Uniform distribution	[0,232]
rocrand generate char	uint8	Uniform distribution	(0,28)
rocrand_generate_short	uint16	Uniform distribution	(0,216)
rocrand_generate_uniform	uint32	Uniform distribution	(0,1)
rocrand_generate_uniform	float	Uniform distribution	(0,1)
rocrand_generate_uniform_double	double	Uniform distribution	(0,1)
rocrand_generate_uniform_half	half	Uniform distribution	(0,1)
rocrand_generate_normal	float	Normal distribution	
rocrand_generate_normal_double	double	Normal distribution	
rocrand_generate_normal_half	half	Normal distribution	
rocrand_generate_log_normal	float	Log-Normal distribution	
rocrand_generate_log_normal_double	double	Log-Normal distribution	
rocrand_generate_log_normal_half	half	Log-Normal distribution	-
rocrand_generate_poisson	uint32	Poisson distribution	

Some Links to Key Libraries

■BLAS

-rocBLAS(https://github.com/ROCmSoftwarePlatform/rocBLAS) -
hipBLAS(https://github.com/ROCmSoftwarePlatform/hipBLAS)

■FFTs

-rocFFT(https://github.com/ROCmSoftwarePlatform/rocFFT)

■ Random number generation

-rocRAND(https://github.com/ROCmSoftwarePlatform/rocRAND)

-hipRAND(https://github.com/ROCmSoftwarePlatform/hipRAND)

■ Sparse linear algebra

-rocSPARSE(https://github.com/ROCmSoftwarePlatform/rocSPARSE)

-hipSPARSE(https://github.com/ROCmSoftwarePlatform/hipSPARSE)

■ Iterative solvers

-rocALUTION（https://github.com/ROCmSoftwarePlatform/rocALUTION ）

■ Parallel primitives

-rocPRIM(https://github.com/ROCmSoftwarePlatform/rocPRIM) -
hipCUB(https://github.com/ROCmSoftwarePlatform/hipCUB)

More links to key libraries

Machine Learning libraries and Frameworks

- -Tensorflow: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
- -Pytorch: https://github.com/ROCmSoftwarePlatform/pytorch
- -MIOpen (similar to cuDNN): https://github.com/ROCmSoftwarePlatform/MIOpen
- – Composable Kernel (similar to CUTLASS): https://github.com/ROCm/composable kernel
- -Tensile (backend to rocBLAS): https://github.com/ROCmSoftwarePlatform/Tensile
- -RCCL(ROCm analogue of NCCL): https://github.com/ROCmSoftwarePlatform/rccl

adapted from UW CSEP 590, Dr. Hari Sadasivan

Porting CUDA Applications to HIP

adapted from UW CSEP 590, Dr. Hari Sadasivan

Getting started with HIP

CUDA VECTOR ADD

```
__global__ void add(int n,
                    double *x,
                    double *y) {
  int index = blockIdx.x * blockDim.x
            + threadId.x;
  int stride = blockDim.x * gridDim.x;

  for (int i=index; i<n; i+=stride) {
    y[i] = x[i] + y[i];
  }
}
```

HIP VECTOR ADD

```
_global__ void add(int n,
                   double *x,
                   double *y) {
int index = blockIdx.x * blockDim.x
          + threadId.x;
int stride = blockDim.x * gridDim.x;

for (int i=index; i<n; i+=stride) {
  y[i] = x[i] + y[i];
}
```

KERNELS ARE SYNTACTICALLY THE SAME

adapted from UW CSEP 590, Dr. Hari Sadasivan

CUDA APIs vs HIP API

CUDA	HIP
cudaMalloc(&d_x,N*sizeof(double));	hipMalloc(&d_x,N*sizeof(double));
cudaMemcpy(d_x,x,N*sizeof(double), cudaMemcpyHostToDevice);	hipMemcpy(d_x,x,N*sizeof(double), hipMemcpyHostToDevice);
cudaDeviceSynchronize();	hipDeviceSynchronize();

adapted from UW CSEP 590, Dr. Hari Sadasivan

Launching a kernel

CUDA KERNEL LAUNCH SYNTAX

```
some_kernel<<<gridsize,blocksize,
                shared_mem_size,stream>>>
                (arg0, arg1, …) ;
```

HIP KERNEL LAUNCH SYNTAX

```
hipLaunchKernelGGL(some_kernel,
                gridsize,blocksize,
                shared_mem_size,stream,
                arg0, arg1, …);
```

adapted from UW CSEP 590, Dr. Hari Sadasivan

Difference between HIP and CUDA

Some things to be aware of writing HIP,or porting from CUDA:

·AMD GCN hardware warp'size =64 (warps are referred to as wavefronts'in AMD documentation)

·Device and host pointers allocated by HIP API use flat addressing

-Unified virtual addressing is enabled by default

-Unified memory is available

·Dynamic parallelism not currently supported

·CUDA 9+thread independent scheduling not supported (e.g.,no__syncwarp)

·Some CUDA library functions do not have AMD equivalents

·Shared memory and registers per thread can differ between AMD and Nvidia hardware

·Inline PTX or AMD GCN assembly is not portable

- **Include“hip/hip_runtime.h”instead of cuda.h**

Despite differences,majority of CUDA code in applications can be simply translated.

adapted from UW CSEP 590, Dr. Hari Sadasivan

Enter Hipify

- AMD provides 'Hipify'tools to automatically convert most CUDA code
  - -Hipify-perl
  - -Hipify-clang

  Good resource to help with porting; https://github.com/ROCm-Developer-Tools/HiP/blob/master/docs/markdown/hip_porting_guide.md

- In practice,large portions of many HPC codes have been automatically Hipified:
  - -~90%of CUDA code in CORAL-2 HACC
  - -~80%of CUDA code in CORAL-2 PENNANT
  - -~80%of CUDA code in CORAL-2 QMCPack
  - -~95%of CUDA code in CORAL-2 Laghos

  The remaining code requires programmer intervention

Hipify tools

·Hipify-perl:

-Easy to use:

point at a directory and it will attempt to hipify CUDA code

-Very simple string replacement technique:may make incorrect translations

-s**ed -e's/cuda/hip/g',(e.g.,cudaMemcpy becomes hipMemcpy**)

-Recommended for quick scans of projects

· Hipify-clang:

-Requires clang compiler

-More robust translation of the code.Uses clang to parse files and perform semantic translation

-Can generate warnings and assistance for code for additional user analysis

-High quality translation,particularly for cases where the user is familiar with the make system

Hipify-perl

- Sits in $HIP/bin/(**export PATH=$PATH:[MYHIP]/bin)**
- Command line tool:hipify-perl foo.cu >new_foo.cpp
- Compile: **hipcc new_foo.cpp**

Pros:

- Does not require CUDA
- Does not require source file to be syntactically correct

Cons:

- May be unable to convert some CUDA code to HIP automatically
- May not be able to expand some macros correctly, parse complicated function argument lists etc.

Hipify-clang

- Build from source
- Converts CUDA to AST and then traverses tree by transformation of matches

https://github.com/ROCm-Developer-Tools/HIP/tree/master/hipify-clang

Pros:

- Can parse &convert complicated constructs
- Supports any new CUDA version
- Supports clang options:-I,-D,--cuda-path

Cons:

- Input code should be correct
- CUDA SDK should be installed
- All includes &defines should be provided to transform code

Hipification requires same headers that would be needed to compile it with clang:

./hipify-clang foo.cu -I /usr/local/cuda-8.0/samples/common/inc

When no GPU, Run HIP programs on CPU

- HIP CPU Runtime
  - https://github.com/ROCm-Developer-Tools/HIP-CPU
  - A header-only library that allows CPUs to execute unmodified HIP code.
  - Generic and does not assume a particular CPU vendor/architecture.