GPU Programming CUDA Nitty Details

Atomics

© Christian Lessig, 2017

 Analogous to atomics in C++ For global and shared memory

 Specific versions for entire GPU and thread block atomicAdd(), atomicSub(), atomicExch(), atomicMin(), atomicMax(), atomicInc(), atomicAnd(), ...

Default precision is IEEE 32-bit (float and int) > Device is designed for it



Default precision is IEEE 32-bit (float and int) > Device is designed for it Many applications require 64-bit precision > e.g. most simulations in scientific computing

 Default precision is IEEE 32-bit (float and int) > Device is designed for it Many applications require 64-bit precision > e.g. most simulations in scientific computing Some applications require less precision > e.g. graphics, deep learning

© Christian Lessig, 2017





Data from https://devblogs.nvidia.com/parallelforall/mixed-precision-programming-cuda-8/

hp

Ο

int $n^2 = n/2;$

© Christian Lessig, 2017

half2 $*x2 = (half2^*)x$, $*y2 = (half2^*)y$; for (int i = start; i < n2; i+= stride) y2[i] = hfma2(halves2half2(a, a), x2[i], y2[i]);

Warp shuffle functions

Communicate values in a warp without using shared memory

© Christian Lessig, 2017

Warp shuffle functions

Communicate values in a warp without using shared memory • E.g.:

© Christian Lessig, 2017

shfl sync(unsigned mask, T var, int srcLane, int width=warpSize);

Warp shuffle functions

Communicate values in a warp without using shared memory • E.g.:

shfl sync(unsigned mask, T var, int srcLane, int width=warpSize);

© Christian Lessig, 2017

value = shfl sync(0xffffffff, value, 0);

Pinned host memory Data transfer from host to device copies from host RAM to device RAM.

© Christian Lessig, 2017

Pinned host memory

Data transfer from hc RAM to device RAM. malloc / new alloca necessarily in RAM

© Christian Lessig, 2017

 Data transfer from host to device copies from host RAM to device RAM.

 malloc / new allocated memory is pageable, i.e. not necessarily in RAM

Pinned host memory

RAM to device RAM. necessarily in RAM

- Data transfer from host to device copies from host
 - > malloc / new allocated memory is pageable, i.e. not
 - Allocate pinned (non-pageable) memory:

Pinned host memory

RAM to device RAM. necessarily in RAM

- Data transfer from host to device copies from host
 - > malloc / new allocated memory is pageable, i.e. not
 - Allocate pinned (non-pageable) memory:
- // float* data = (float*) malloc(8*sizeof(float)); cudaMallocHost((void**)& data pinned, 8*sizeof(float));



Driver API

Driver API for more fine grained control

© Christian Lessig, 2017

Driver AP

Driver API for more fine grained control

// cuMemcpyHtoD(d B, h B, size); cudaMemcpy(d B, d h, size, cudaMemcpyHostToDevice)

// vecAdd<< blocks, threads>>(d A, d B, d C) void* args[] = { &d A, &d B, &d C, &N }; cuLaunchKernel(vecAdd, blocksPerGrid, 1, 1, threadsPerBlock, 1, 1, 0, args, NULL);

© Christian Lessig, 2017

NVRTC On-the-fly (runtime) compilation of code

compileFileToPTX("vecAdd.cu", 0, NULL, &ptx, &ptxSize); CUmodule module = loadPTX(ptx, argc, argv);

CUfunction kernel addr; cuModuleGetFunction(&kernel addr, module, "vecAdd"));

© Christian Lessig, 2017

Asynchronous Execution

- Motivation:

Processing large data sets Smaller tasks that do not utilize full device > Latency critical applications Executes multiple kernels (streams) concurrently You Typically combined with asynchronous data transfer

Asynchronous Execution

for (int i = 0; i < nStreams; ++i) { cudaStreamCreate(&stream[i]); cudaMemcpyAsync(&dd[i], &dh[i], i, cudaMemcpyHostToDevice, stream[i]); kernel<<<block, threads, 0, stream[i]>>>(dd); cudaMemcpyAsync(&dh[i], &dd[i], size, cudaMemcpyDeviceToHost, stream[i]);

Asynchronous Execution

for (int i = 0; i < nStreams; ++i) { cudaStreamCreate(&stream[i]); cudaMemcpyAsync(&dd[i], &dh[i], i, cudaMemcpyHostToDevice, stream[i]); kernel<<<block, threads, 0, stream[i]>>>(dd); cudaMemcpyAsync(&dh[i], &dd[i], size,

© Christian Lessig, 2017

cudaMemcpyDeviceToHost, stream[i]);

Unified memory

Unified address space for data Memory management (host <-> device transfers) are handled by the API / driver

© Christian Lessig, 2017

Memory allocation on the device

free (ptr);

- Memory can be allocated dynamically on the device from a pre-defined "heap" area (that, however, resides in in global memory)
 - global void kernel(int size) { char* ptr = (char*) malloc(size); memset(ptr, 0, size);

Dynamic parallelism

- existing one

© Christian Lessig, 2017

Enables to generate new thread grid from within an

 Avoids significant overhead of host <-> device synchronization when parallelism is data dependent

Warp matrix operations

Core of the deep learning hardware on CUDA devices. Provides dedicated hardware to implement:

D =

FP16 or FP32

from https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/



Warp matrix operations Core of the deep learning hardware on CUDA devices.

// Load the inputs
wmma::load_matrix_sync(a_frag, a, lda);
wmma::load_matrix_sync(b_frag, b, ldb);

// Perform the matrix multiplication
wmma::mma_sync(acc_frag, a_frag, b_frag, acc_frag);

© Christian Lessig, 2017

Independent thread scheduling

Pre-Volta

Program Counter (PC) and Stack (S)

Volta

Convergence Optimizer

from https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

© Christian Lessig, 2017





32 thread warp with independent scheduling



Graphics inter-op

© Christian Lessig, 2017

 Enables to share data with OpenGL / DirectX Useful for example for global illumination or simulation of fluids that are more efficiently computed in Cuda but must be displayed using a graphics API.

Cooperative groups

Extends the concept of a warp / thread block to user specified groups of threads that can interact
Synchronization of sets of blocks or entire grid (without host intervention)
Communication of values within a group of threads

© Christian Lessig, 2017

Cuda libraries

CUFFT cuSparse ° cuBLAS cuDNN CUTLASS thrust Ο

• • •

nvprof: command line profiler

© Christian Lessig, 2017



nvprof: command line profiler

bauhaus:build lessig\$ /Developer/NVIDIA/CUDA-8.0/bin/nvprof ./main ==1714== NVPROF is profiling process 1714, command: ./main Execution time: 381.35 ms. ==1714== Profiling application: ./main ==1714== Profiling result: Time Calls Time (%) Min Avq 88.93% 380.73ms float*, unsigned int) 6.00% 25.692ms 25.692ms 25.692ms 1 25.692ms 5.07% 21.701ms ==1714 == API calls:Time Time (%) Calls Min Avg 66.50% 380.68ms 2 190.34ms 18.900us 38 24.95% 142.84ms 2 71.419ms 194.69us 14

© Christian Lessig, 2017

• • •

Max	Name
80.67ms	cudaDeviceSynchronize
12.64ms	cudaMalloc

- 1 21.701ms 21.701ms 21.701ms [CUDA memcpy DtoH]
- [CUDA memcpy HtoD]
- 99 3.8458ms 3.7730ms 4.4032ms transposeMatrix3(float*,
- Max Name

nvvp: visual profiler



nvvp: visual profiler

		NVIDIA Visual Profiler		
°님님!!! ■ *NewSession1 %				
0 s	0.25 s	0.5 s	0.75 s	1 s
Process "main" (1786)			•,;;	
Thread 1959428096				
- Runtime API				
L Driver API				
Profiling Overhead		1		
[[0] GeForce GTX 980				
Context 1 (CUDA)				
⊢ ▼ MemCov (HtoD)				
L ™ MemCov (DtoH)				
Etroome				
E 🔂 🎧 Results	<u>».</u> ,	transposeMatrix3(float*, float*, unsigned int)		
		Session		1 845 s (1
. CUDA Appli		Kernel		384.069 r
he guided analy		Invocations		99
ou through the		Importance		100%
ptimization opr				
application. Onc				
amiliar with the				
ndividual analys				
inguided mode.				
our application				
ully utilize the c				
o do this you st				
pplication's ove				
well as the perfc				
ndividual kernel				
Examin				
Determine your or				



nvvp: visual profiler

			NVIDIA Vis	NVIDIA Visual Profiler			
* 🖾 🖳 🍱 🖦 💊 🔹 🔍	🔍 F 📉 🔒						
🔍 *NewSession1 🔀							
	0 s	0.25 s	0.5 s 0.75	s 1 _, s			
Process "main" (1786)							
Thread 1959428096							
- Runtime API							
L Driver API							
Profiling Overhead							
I [0] GeForce GTX 980							
Context 1 (CUDA)							
	_						
Compute							
⊢ 🍸 100.0% transpos.							
Streams							
└ Default							
E E 😚 🛄 Results			Start	1 10R c (1			
E 🔂 🎧 Results			Start	1 496 s (1			
1. CUDA Appli			End	1.5 s (1,49			
			Duration	3.804 ms			
i në gulded anal; vou through the			Stream	Default			
stages to help vi			Grid Size	[256,256			
ptimization opr			Block Size	[32,32,1			
application. Onc			Registers/Thread	10			
amiliar with the			Shared Memory/Block	4.125 KiB			
orocess, you car			▼Occupancy	1004			
individual analys			I heoretical	100%			
your application			Shared Memory Configuration				
ully utilize the c			Shared Memory Requested	96 KIB			
novement capal			Shared Memory Executed	96 KIB			
To do this you sł			Shared Memory Bank Size	4 B			
application's ove			Global Cache Configuration				
well as the perfc			Global Cache Requested	OT			
Individual kernel			Global Cache Executed	off			
Examin							
Determine your or							

