GPU Computing Tools

Varun Sampath
University of Pennsylvania
CIS 565 - Spring 2012

Agenda

- CUDA Toolchain
  - APIs
  - Language bindings
  - Libraries
  - Visual Profiler
  - Parallel Nsight
- OpenCL
- C++ AMP

CUDA Documentation

- http://developer.nvidia.com/nvidia-gpu-computing-documentation
- CUDA C Programming Guide
- CUDA C Best Practices Guide
- CUDA API Reference Manual
- Occupancy Calculator
- Much more

CUDA Organization

- Host code: two layers
  - Runtime API
    - `cudart` dynamic library
    - `cuda_runtime_api.h` (C)
    - `cuda_runtime.h` (C++)
  - Driver API
    - `nvcuda` dynamic library
    - `cuda.h`
- Device code
  - Kernel → PTX (parallel thread eXecution)

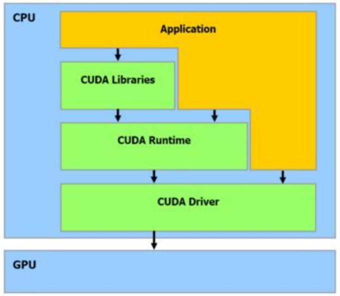


Image from [Stack Overflow]()

CUDA API Comparison

CUDA Runtime API

```
// create CUDA device & context
cudaSetDevice( 0 ); // pick first device
kernel_naive_copy<<<cnBlocks, cnBlockSize>>>
(i_data, o_data,
rows, cols);
```

Differences?

CUDA Driver API

```
cuInit(0);
cuDeviceGet(&hContext, 0);
// pick first device
cuCtxCreate(&hContext, 0,
hDevice));
cuModuleLoad(&hModule,
“copy_kernel.cubin”);
cuModuleGetFunction(&hFunction, hModule,
“kernel_naive_copy");
…
cuLaunchGrid(cuFunction,
cnBlocks, 1);
```

Code from [CUDA Best Practices Guide 4.0]

Some CUDA Language Bindings

- Note: the following are not supported by NVIDIA
- [PyCUDA] (Python)
  - Developed by [Andreas Klöckner]
  - Built on top of CUDA Driver API
  - Also: PyOpenCL
- [JCuda] (Java)
- MATLAB
  - Parallel Computing Toolbox
  - [AccelerEyes] Jacket

Whetting your appetite

```
import pycuda.driver as cuda
import pycuda.autoinit, pycuda.compiler
import numpy

a = numpy.random.randn(4,4).astype(numpy.float32)
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)
```

[This is examples/demo.py in the PyCUDA distribution.]

Slides [URL]

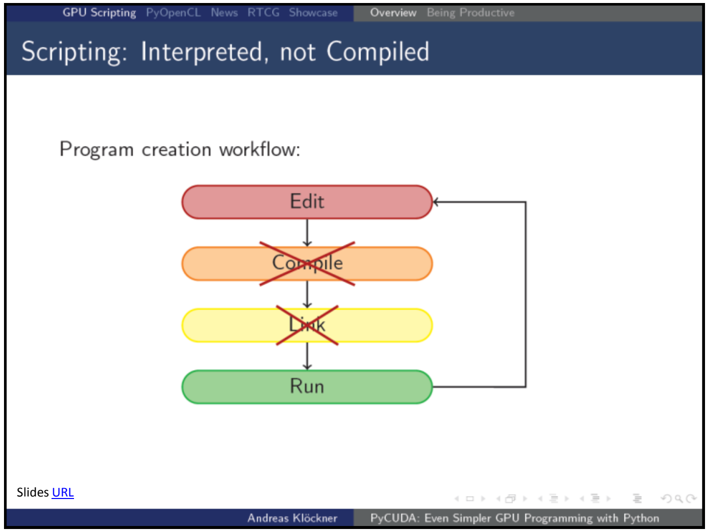
Whetting your appetite

```
mod = pycuda.compiler.SourceModule("""
    __global__ void twice(float *a)
    {
      int idx = threadIdx.x + threadIdx.y*4;
      a[idx] *= 2;
    }
    """)

func = mod.get_function("twice")
func(a_gpu, block=(4,4,1))

a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
print a_doubled
print a
```

Slides [URL]



Scripting: Interpreted, not Compiled

Program creation workflow:

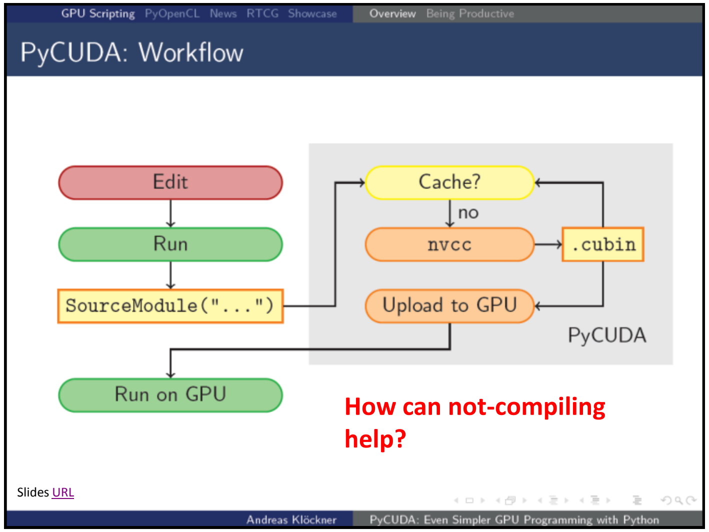
Edit

~~Compile~~

~~Link~~

Run

Slides URL



PyCUDA: Workflow

Edit

Run

`SourceModule("...")`

Cache?

no

`nvcc`

`.cubin`

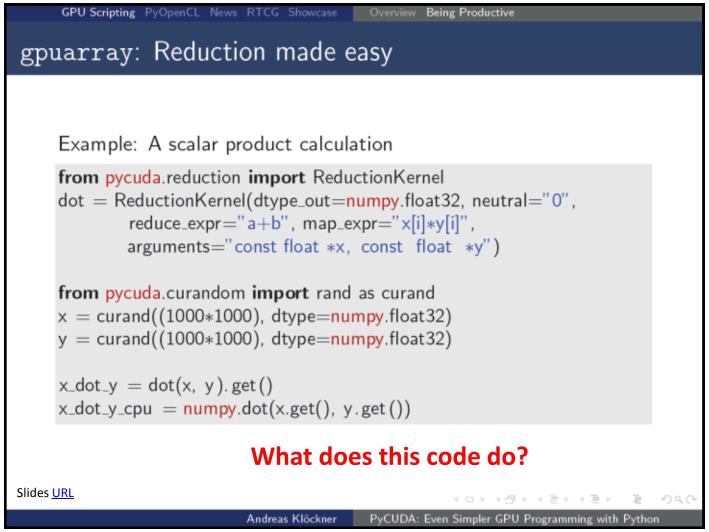
Upload to GPU

PyCUDA

Run on GPU

How can not-compiling help?

Slides URL



`gpuarray`: Reduction made easy

Example: A scalar product calculation

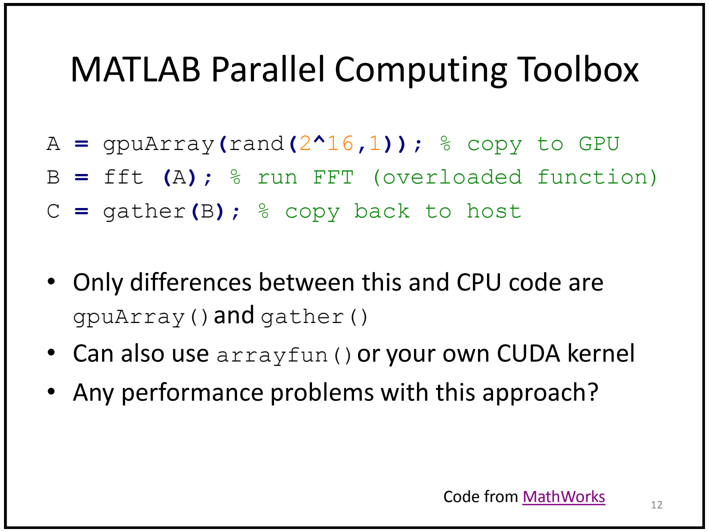
```
from pycuda.reduction import ReductionKernel
dot = ReductionKernel(dtype_out=numpy.float32, neutral="0",
        reduce_expr="a+b", map_expr="x[i]*y[i]",
        arguments="const float *x, const float *y")

from pycuda.curandom import rand as curand
x = curand((1000*1000), dtype=numpy.float32)
y = curand((1000*1000), dtype=numpy.float32)

x_dot_y = dot(x, y).get()
x_dot_y_cpu = numpy.dot(x.get(), y.get())
```

What does this code do?

Slides URL



MATLAB Parallel Computing Toolbox

```
A = gpuArray(rand(2^16,1)); % copy to GPU
B = fft (A); % run FFT (overloaded function)
C = gather(B); % copy back to host
```

- Only differences between this and CPU code are `gpuArray()` and `gather()`
- Can also use `arrayfun()` or your own CUDA kernel
- Any performance problems with this approach?

Code from MathWorks

CUDA Libraries

- Productivity
  - Thrust
- Performance
  - cuBLAS
  - cuFFT
  - Plenty more

Prelude: C++ Templates Primer

```
template <typename T>
T sum(const T a, const T b) {
        return a + b;
}

int main() {
        cout << sum<int>(1, 2) << endl;
        cout << sum<float>(1.21, 2.43) << endl;
        return 0;
}
```

- Make functions and classes generic
- Evaluate at compile-time
- Standard Template Library (STL)
  - Algorithms, iterators, containers

Reference: MIT OCW

Thrust - “Code at the speed of light”

- Developed by Jared Hoberock and Nathan Bell of NVIDIA Research
- Objectives
  - Programmer productivity
    - Leverage parallel primitives
  - Encourage generic programming
    - E.g. one reduction to rule them all
  - High performance
    - With minimal programmer effort
  - Interoperability
    - Integrates with CUDA C/C++ code

Objectives from Intro to Thrust Slides

Thrust - Example

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>
int main(void)
{
        // generate 16M random numbers on the host
        thrust::host_vector<int> h_vec(1 << 24);
        thrust::generate(h_vec.begin(), h_vec.end(), rand);
        // transfer data to the device
        thrust::device_vector<int> d_vec = h_vec;
        // sort data on the device
        thrust::sort(d_vec.begin(), d_vec.end());
        // transfer data back to host
        thrust::copy(d_vec.begin(), d_vec.end(),
                     h_vec.begin());
        return 0;
}
```

Code from GPU Computing Gems

Thrust Design

- Based off STL ideas
  - Algorithms, iterators, containers
  - Generic through C++ templates
- Built on top of CUDA Runtime API
  - Ships with CUDA 4.0+
- Four fundamental parallel algorithms
  - `for_each`
  - `reduce`
  - `scan`
  - `sort`

Thrust-CUDA C Interoperability

```
size_t N = 1024;
// raw pointer to device memory
int* raw ptr;
cudaMalloc(&raw_ptr, N*sizeof(int));
// wrap raw pointer with a device ptr
device ptr<int> dev ptr =
      device_pointer_cast(raw_ptr);
// use device ptr in Thrust algorithms
sort(dev_ptr, dev_ptr + N);
// access device memory through device ptr
dev_ptr[0] = 1;
// free memory
cudaFree(raw ptr);
```

Code from GPU Computing Gems

Thrust with User-Defined Functions

```
struct saxpy_functor {
      const float a;
      saxpy_functor(float a) : a( a) {}
      __host__ __device__
      float operator()(float x, float y){ return a*x+y; }
};
void saxpy(float a, device vector<float>& x, device
vector<float>& y) {
      // setup functor
      saxpy_functor func(a);
      // call transform
      transform(x.begin(), x.end(), y.begin(), y.begin(),
func);
}
```

Code from GPU Computing Gems

Thrust Performance

- Templates allow inlining and type analysis
  - How could knowing types improve global memory performance?

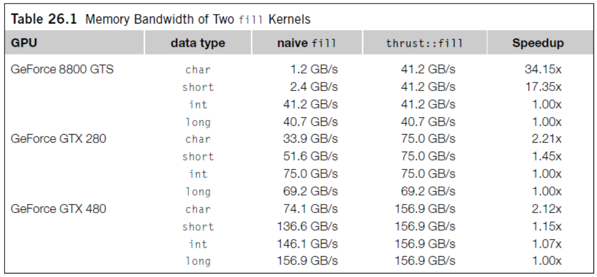


Table 26.1 Memory Bandwidth of Two `fill` Kernels

GPU	data type	naive `fill`	`thrust::fill`	Speedup
GeForce 8800 GTS	`char`	1.2 GB/s	41.2 GB/s	34.15x
	`short`	2.4 GB/s	41.2 GB/s	17.35x
	`int`	41.2 GB/s	41.2 GB/s	1.00x
	`long`	40.7 GB/s	40.7 GB/s	1.00x
GeForce GTX 280	`char`	33.9 GB/s	75.0 GB/s	2.21x
	`short`	51.6 GB/s	75.0 GB/s	1.45x
	`int`	75.0 GB/s	75.0 GB/s	1.00x
	`long`	69.2 GB/s	69.2 GB/s	1.00x
GeForce GTX 480	`char`	74.1 GB/s	156.9 GB/s	2.12x
	`short`	136.6 GB/s	156.9 GB/s	1.15x
	`int`	146.1 GB/s	156.9 GB/s	1.07x
	`long`	156.9 GB/s	156.9 GB/s	1.00x

Image from GPU Computing Gems

Thrust Toy-box

- Kernel fusion with `transform_iterator` and `permutation_iterator`
- Conversion between arrays of structs (AoS) and structure of arrays (SoA) with `zip_iterator`
- Implicit ranges

CUDA Specialized Libraries

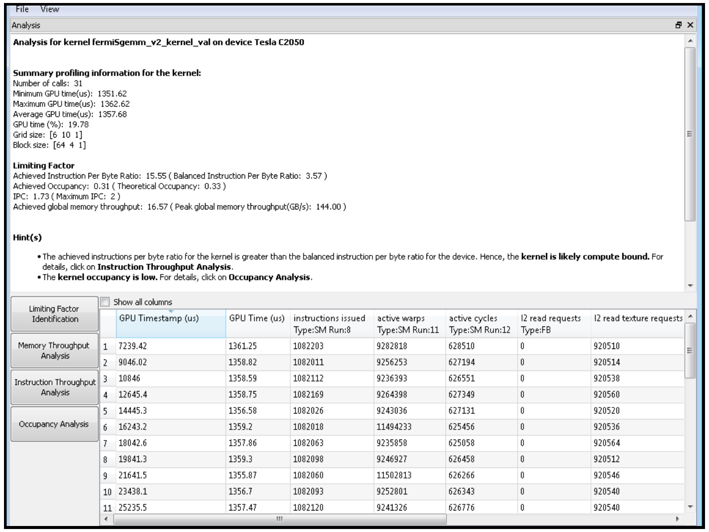
- NVIDIA cuBLAS
  - Basic Linear Algebra Subprograms (BLAS)
- NVIDIA cuFFT
  - Compute Fast Fourier Transforms
- NVIDIA NPP
  - Image and Signal Processing
- See more: http://developer.nvidia.com/gpu-accelerated-libraries

CUDA Profiling and Debugging

- Visual Profiler
- Parallel Nsight
- cuda-gdb

Visual Profiler

- Graphical profiling application
- Collects performance counter data and makes recommendations
  - Global memory throughput
  - IPC
  - Active warps/cycle
  - Cache hit rate
  - Register counts
  - Bank conflicts
  - Branch divergence
  - Many more (Full list in Visual Profiler User Guide)



Analysis for kernel fermiSgemm_v2_kernel_val on device Tesla C2050

Summary profiling information for the kernel:
Number of calls: 31
Minimum GPU time(us): 1351.62
Maximum GPU time(us): 1362.62
Average GPU time(us): 1357.68
GPU time (%): 19.78
Grid size: [6 10 1]
Block size: [64 4 1]

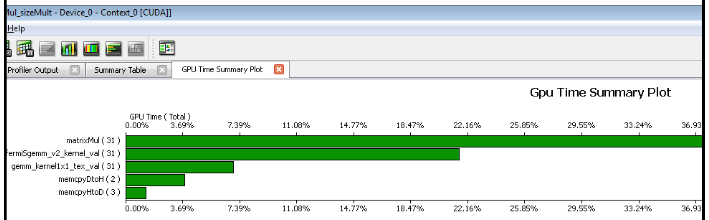
Limiting Factor
Achieved Instruction Per Byte Ratio: 15.55 (Balanced Instruction Per Byte Ratio: 3.57)
Achieved Occupancy: 0.31 (Theoretical Occupancy: 0.33)
IPC: 1.73 (Maximum IPC: 2)
Achieved global memory throughput: 16.57 (Peak global memory throughput(GB/s): 144.00)

Hint(s)

- The achieved instructions per byte ratio for the kernel is greater than the balanced instruction per byte ratio for the device. Hence, the **kernel is likely compute bound.** For details, click on **Instruction Throughput Analysis**.
- The **kernel occupancy is low.** For details, click on **Occupancy Analysis**.

☐ Show all columns

	GPU Timestamp (us)	GPU Time (us)	instructions issued Type:SM Run:8	active warps Type:SM Run:11	active cycles Type:SM Run:12	l2 read requests Type:FB	l2 read texture requests
1	7239.42	1361.25	1082203	9282818	628510	0	920510
2	9046.02	1358.82	1082011	9256253	627194	0	920514
3	10846	1358.59	1082112	9236393	626551	0	920538
4	12645.4	1358.75	1082169	9264398	627349	0	920560
5	14445.3	1356.58	1082026	9243036	627131	0	920520
6	16243.2	1359.2	1082018	11494233	625456	0	920536
7	18042.6	1357.86	1082063	9235858	625058	0	920564
8	19841.3	1359.3	1082098	9246927	626458	0	920512
9	21641.5	1355.87	1082060	11502813	626266	0	920546
10	23438.1	1356.7	1082093	9252801	626343	0	920540
11	25235.5	1357.47	1082120	9241326	626776	0	920540



Visual Profiler

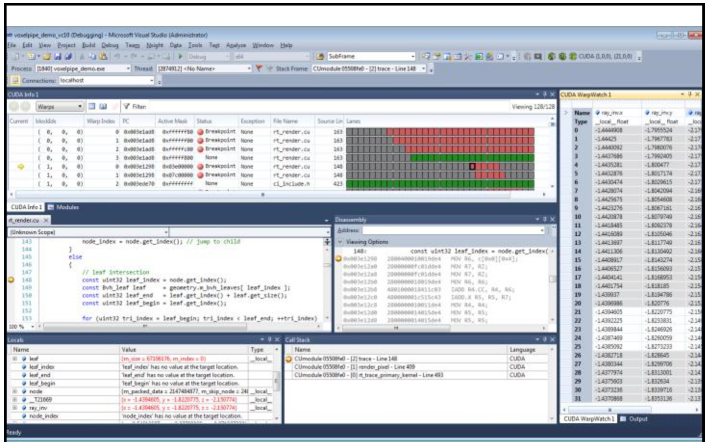
Does plots too!

Parallel Nsight

- Motivation
  - Why didn't breakpoints in Visual Studio work for debugging CUDA?

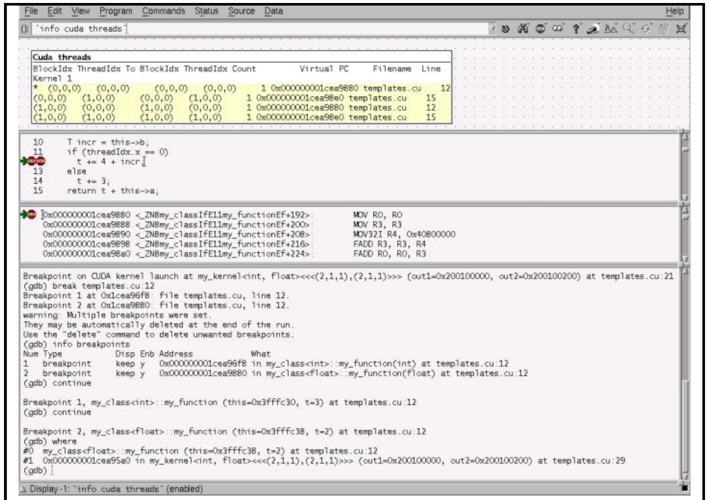
Parallel Nsight

- Debugger and Profiler for:
  - CUDA
  - OpenCL
  - Direct3D Shaders
- Integrated into Visual Studio 2008/2010
- Caveat: requires extra GPU for display while debugging
  - Supports NVIDIA Optimus



Parallel Nsight, showing breakpoints for different warps, disassembled kernel code, local variables, call stack, and register values per warp

Image from [NVIDIA]



```
(): `info cuda threads`
```

```
Cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count        Virtual PC      Filename  Line
Kernel 1
*  (0,0,0)   (0,0,0)     (0,0,0)   (0,0,0)     1 0x000000001cea9880 templates.cu    12
(0,0,0)   (1,0,0)     (0,0,0)   (1,0,0)     1 0x000000001cea98e0 templates.cu    15
(1,0,0)   (0,0,0)     (1,0,0)   (0,0,0)     1 0x000000001cea9880 templates.cu    12
(1,0,0)   (1,0,0)     (1,0,0)   (1,0,0)     1 0x000000001cea98e0 templates.cu    15
```

```
    T incr = this->b;
    if (threadIdx.x == 0)
      t += 4 + incr;
    else
      t += 3;
    return t + this->a;
```

```
0x000000001cea9880 <_ZN8my_classIfE11my_functionEf+192>:        MOV R0, R0
0x000000001cea9888 <_ZN8my_classIfE11my_functionEf+200>:        MOV R3, R3
0x000000001cea9890 <_ZN8my_classIfE11my_functionEf+208>:        MOV32I R4, 0x40800000
0x000000001cea9898 <_ZN8my_classIfE11my_functionEf+216>:        FADD R3, R3, R4
0x000000001cea98a0 <_ZN8my_classIfE11my_functionEf+224>:        FADD R0, R0, R3
```

```
Breakpoint on CUDA kernel launch at my_kernel<int, float><<<(2,1,1),(2,1,1)>>> (out1=0x200100000, out2=0x200100200) at templates.cu:21
(gdb) break templates.cu:12
Breakpoint 1 at 0x1cea96f8: file templates.cu, line 12.
Breakpoint 2 at 0x1cea9880: file templates.cu, line 12.
warning: Multiple breakpoints were set.
They may be automatically deleted at the end of the run.
Use the "delete" command to delete unwanted breakpoints.
(gdb) info breakpoints
Num Type           Disp Enb Address            What
1   breakpoint     keep y   0x000000001cea96f8 in my_class<int>::my_function(int) at templates.cu:12
2   breakpoint     keep y   0x000000001cea9880 in my_class<float>::my_function(float) at templates.cu:12
(gdb) continue

Breakpoint 1, my_class<int>::my_function (this=0x3fffc30, t=3) at templates.cu:12
(gdb) continue

Breakpoint 2, my_class<float>::my_function (this=0x3fffc38, t=2) at templates.cu:12
(gdb) where
#0  my_class<float>::my_function (this=0x3fffc38, t=2) at templates.cu:12
#1  0x000000001cea95a0 in my_kernel<int, float><<<(2,1,1),(2,1,1)>>> (out1=0x200100000, out2=0x200100200) at templates.cu:29
(gdb)
```

```
Display-1: `info cuda threads` (enabled)
```

CUDA-GDB: No *nix User Left Behind

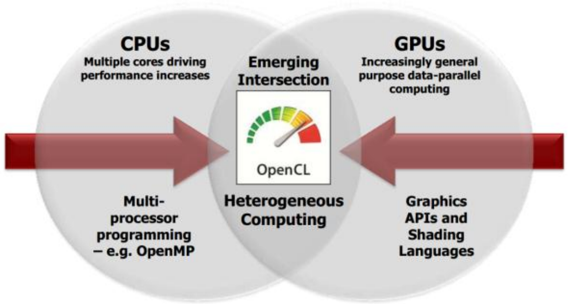
Image from [NVIDIA]

Image from the [Khronos Group]

OpenCL

- Initially developed by Apple with help from AMD, IBM, Intel, and NVIDIA ([Wikipedia])
- Specification defined by the Khronos Group

Processor Parallelism



OpenCL is a programming framework for heterogeneous compute resources

Slide from the Khronos Group

OpenCL Goals

- Parallel Compute Framework for GPUs
  - And CPUs
  - And FPGAs
  - And potentially more
- Some compliant runtimes
  - AMD APP SDK (for AMD CPUs, GPUs, and APUs)
  - Intel OpenCL SDK (for Intel CPUs)
  - NVIDIA OpenCL Runtime (for NVIDIA GPUs)

Do we want CPUs and GPUs executing the same kernels though?

OpenCL Host Code

```
size_t szLocalWorkSize[2];
size_t szGlobalWorkSize[2];

szLocalWorkSize[0] = 8;
szLocalWorkSize[1] = 8;
szGlobalWorkSize[0] = cols;
szGlobalWorkSize[1] = rows;

// setup parameter values
copyCode = oclLoadProgSource(“copy_kernel.cl", "", &copyLen);
hCopyProg = clCreateProgramWithSource(t->hContext,1, (const char
**)&copyCode, &copyLen, &errcode_ret);
clBuildProgram(hCopyProg, 0, NULL, NULL, NULL, NULL);
// create kernel
t->hCopyKernel = clCreateKernel(hCopyProg, "kernel_naive_copy",
&errcode_ret);
clSetKernelArg(t->hCopyKernel, 0, sizeof(cl_mem), (void*)&dev_i_data);
clSetKernelArg(t->hCopyKernel, 1, sizeof(cl_mem), (void*)&dev_o_data);
clSetKernelArg(t->hCopyKernel, 2, sizeof(cl_int), (void*)&rows);
clSetKernelArg(t->hCopyKernel, 3, sizeof(cl_int), (void*)&cols);
clEnqueueNDRangeKernel(t->hCmdQueue, t->hCopyKernel, 2, NULL,
szGlobalWorkSize, szLocalWorkSize, 0, NULL, NULL);
```

OpenCL Host Code

```
size_t szLocalWorkSize[2];
size_t szGlobalWorkSize[2];

szLocalWorkSize[0] = 8;
szLocalWorkSize[1] = 8;
szGlobalWorkSize[0] = cols;
szGlobalWorkSize[1] = rows;

// setup parameter values
copyCode = oclLoadProgSource(“copy_kernel.cl", "", &copyLen);
hCopyProg = clCreateProgramWithSource(t->hContext,1, (const char
**)&copyCode, &copyLen, &errcode_ret);
clBuildProgram(hCopyProg, 0, NULL, NULL, NULL, NULL);
// create kernel
t->hCopyKernel = clCreateKernel(hCopyProg, "kernel_naive_copy",
&errcode_ret);
clSetKernelArg(t->hCopyKernel, 0, sizeof(cl_mem), (void*)&dev_i_data);
clSetKernelArg(t->hCopyKernel, 1, sizeof(cl_mem), (void*)&dev_o_data);
clSetKernelArg(t->hCopyKernel, 2, sizeof(cl_int), (void*)&rows);
clSetKernelArg(t->hCopyKernel, 3, sizeof(cl_int), (void*)&cols);
clEnqueueNDRangeKernel(t->hCmdQueue, t->hCopyKernel, 2, NULL,
szGlobalWorkSize, szLocalWorkSize, 0, NULL, NULL);
```

← What are these for?

Look Familiar?

OpenCL Device Code

```
__kernel void kernel_naive_copy(
       __global const float4 * i_data,
       __global float4 * o_data,
       int rows, int cols)
{
   uint x = get_global_id(0);
   uint y = get_global_id(1);
   o_data[y*rows + x] = i_data[y*rows + x];
}
```

See some similarities?

OpenCL Code

- Very similar to CUDA Driver API and CUDA C
  - NVIDIA has a short guide outlining syntax differences
- C-based API
  - C++ wrappers and bindings to other languages (e.g. PyOpenCL) available

Which should I choose?

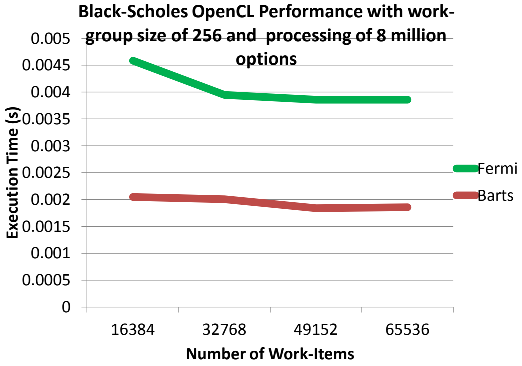
OPENCL OR CUDA?

Compatibility

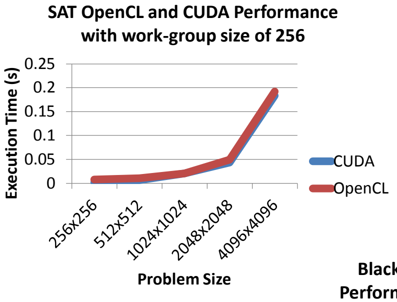
- CUDA runs only on NVIDIA GPUs
  - Not necessarily true…
- OpenCL is supported by a lot of vendors

Image from the Khronos Group

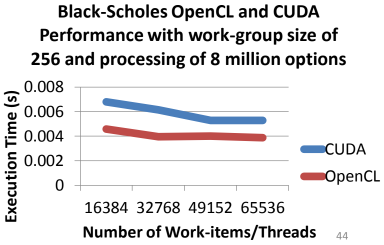
Doesn't everyone just want an NVIDIA GPU?


Black-Scholes OpenCL Performance with work-group size of 256 and processing of 8 million options

Performance Comparison on NVIDIA GPUs


SAT OpenCL and CUDA Performance with work-group size of 256

[This was done with the CUDA 3.2 Toolkit. CUDA 4.1 brought a new LLVM compiler to CUDA (OpenCL compiler was already LLVM-based)]


Black-Scholes OpenCL and CUDA Performance with work-group size of 256 and processing of 8 million options

Programming Framework Comparison

- CUDA 4.0 brought a lot of advancements
  - Unified address space
  - C++ new/delete, virtual functions on device
  - GPUDirect peer-to-peer GPU communication
- OpenCL does not have these features
  - And 18-month release cadence is slower than NVIDIA's

Libraries & Mindshare

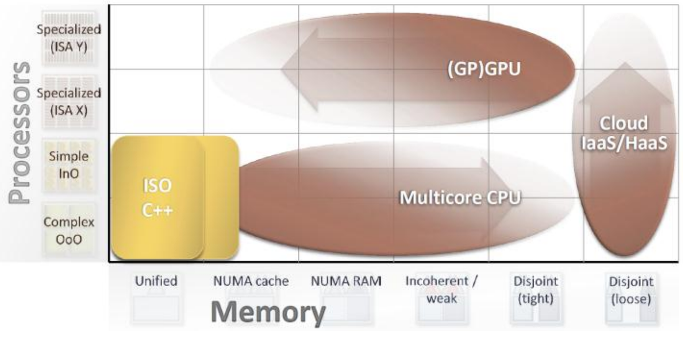
- CUDA has a larger ecosystem
  - Thrust is a particularly important library
- Will OpenCL catch up?
  - Growing in other ways
    - OpenCL Embedded Profiles
    - WebCL

C++ AMP

C++ AMP (Accelerated Massive Parallelism)

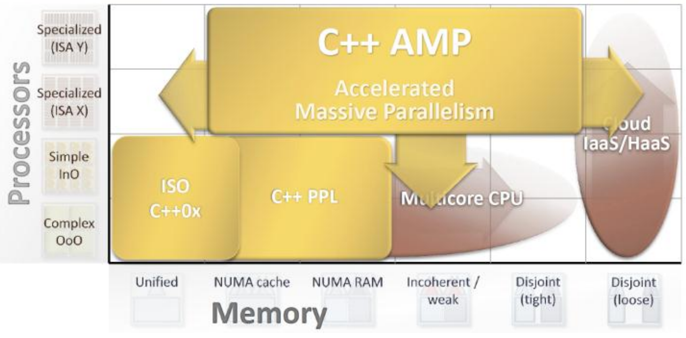
- Announced by Microsoft in June 2011
- Targeting “heterogeneous parallel computing”
  - – Multicore CPUs
  - – GPUs
  - – Cloud Infrastructure-as-a-Service (IaaS)

Programming Models & Languages



Slide from Herb Sutter’s AMD Fusion Keynote

Programming Models & Languages



Slide from Herb Sutter’s AMD Fusion Keynote

C++ AMP Matrix Multiply

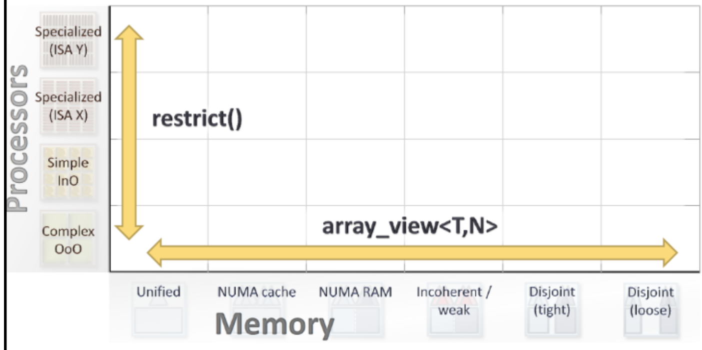
```
void MatrixMult( float* C, const vector<float>& A,
const vector<float>& B,
int M, int N, int W )
{
    array_view<const float,2> a(M,W,A), b(W,N,B);
    array_view<writeonly<float>,2> c(M,N,C);
    parallel for each( c.grid, [=](index<2> idx)
      restrict(direct3d) {
        float sum = 0;
        for(int i = 0; i < a.x; i++)
            sum += a(idx.y, i) * b(i, idx.x);
        c[idx] = sum;
    } );
}
```

C++ AMP Matrix Multiply

```
void MatrixMult( float* C, const vector<float>& A,
const vector<float>& B,
int M, int N, int W )
{
    array_view<const float,2> a(M,W,A), b(W,N,B);
    array_view<writeonly<float>,2> c(M,N,C);
    parallel for each( c.grid, [=](index<2> idx)
      restrict(direct3d) {
        float sum = 0;
        for(int i = 0; i < a.x; i++)
            sum += a(idx.y, i) * b(i, idx.x);
        c[idx] = sum;
    } );
}
```

- `array_view`: abstraction for accessing data (like an “iterator range”)
- Lambda expressions: like functors of thrust but with less syntactic overhead
- `restrict`: ensure only language capabilities supported by device are used

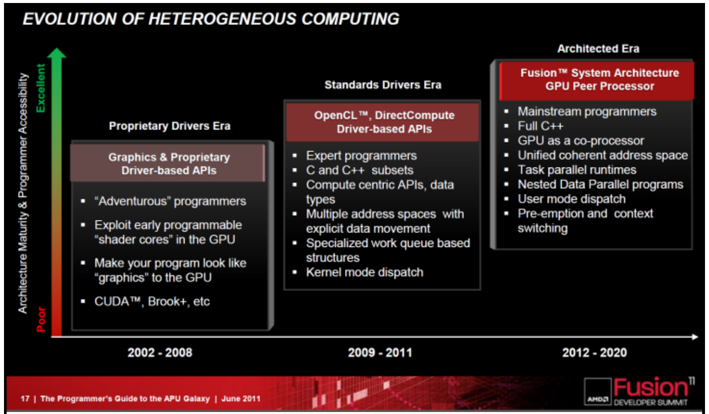
C++ AMP at a Glance



Slide from Herb Sutter’s AMD Fusion Keynote

C++ AMP

- Only 1 new keyword added to C++
  - All other functionality in classes and functions
- Released as open specification 2 weeks ago
- Debugging and Profiling included in Visual Studio 11



EVOLUTION OF HETEROGENEOUS COMPUTING

Proprietary Drivers Era

Graphics & Proprietary Driver-based APIs

- "Adventurous" programmers
- Exploit early programmable "shader cores" in the GPU
- Make your program look like "graphics" to the GPU
- CUDA™, Brook+, etc

2002 - 2008

Standards Drivers Era

OpenCL™, DirectCompute Driver-based APIs

- Expert programmers
- C and C++ subsets
- Compute centric APIs, data types
- Multiple address spaces with explicit data movement
- Specialized work queue based structures
- Kernel mode dispatch

2009 - 2011

Architected Era

Fusion™ System Architecture GPU Peer Processor

- Mainstream programmers
- Full C++
- GPU as a co-processor
- Unified coherent address space
- Task parallel runtimes
- Nested Data Parallel programs
- User mode dispatch
- Pre-emption and context switching

2012 - 2020

17 | The Programmer's Guide to the APU Galaxy | June 2011

Conclusion: What do you think?

Slide from Herb Sutter's AMD
Fusion Keynote

References

- Bell, Nathan and Hoberock, Jared. "Thrust: A Productivity-Oriented Library for CUDA." GPU Computing Gems: Jade Edition. Link
- Klöckner, Andreas. "PyCUDA: Even Simpler
- GPU Programming with Python." Slides
- Reese, Jill and Zaranek, Sarah. "GPU Programming in MATLAB." Link
- Rosenberg, Ofer. "OpenCL Overview." Slides
- Sutter, Herb. "Heterogeneous Parallelism at Microsoft." Link

---

Bibliography

- Klöckner, et al. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation." arXiv
- Moth, Daniel. "Blazing-fast code using GPUs and more, with C++ AMP." Link