Raising the Level of Abstraction of GPU-programming

Ferosh Jacob[1], Ritu Arora[2], Purushotham Bangalore[2],
Marjan Mernik[32], Jeff Gray[1]
jacobf@ua.edu, {ritu, puri, mernik}@cis.uab.edu,
gray@cs.ua.edu

[1]Department of Computer Science
University of Alabama, Tuscaloosa, Alabama

[2]Department of Computer & Info Sciences
University of Alabama at Birmingham, Birmingham, Alabama

[3]Faculty of Electrical Engineering and Computer Science
University of Maribor, Slovenia

July 13, 2010

Why GPU Programming?

- Extensively parallel programs
- Excellent computational platforms for scientific calculations
- Introduction of GPGPU
- Desktops and laptops with GPUs
- OpenCL, CUDA, DirectCompute

Overview: CUDA and OpenCL

Similarities

- C language modified
- Used in GPGPU
- Follows a SPMD

Differences

- CUDA $\Rightarrow$ NVIDIA, OpenCL $\Rightarrow$ Khronos
- OpenCL multi-vendor support
- OpenCL is still evolving
- OpenCL $\Rightarrow$ heterogeneous

Abstraction in GPU

- Allows programmer to focus on essence of parallel computing, rather than language-specific accidental complexities of CUDA or OpenCL
- Higher level similarity in program structure
- Duplicated code
- Need to support heterogeneous architecture
- Higher abstraction allows the programmer to focus on the important issues while technical infrastructure is transparent

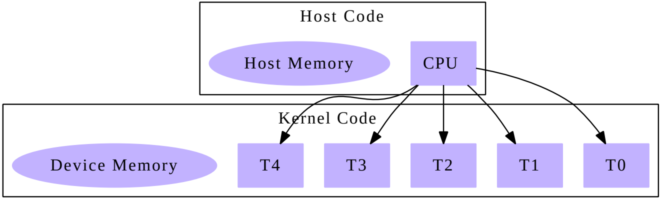
Source code for Vector Addition

```
  /*
   *  SEQUENTIAL CODE
   */
float* h_A = (float*) malloc (mem_size_A);
float* h_B = (float*) malloc (mem_size_B);
float* h_C = (float*) malloc (mem_size_C);

initarray(h_A,h_B);
sequentialAdd(h_A,h_B,h_C);
printArray(h_C);

//Release Memory
free(h_A);
free(h_B);
free(h_C);
```

CUDA and OpenCL

Five Steps

1. Allocate memory in host
2. Copy memory from host to device
3. Execute the kernel
4. Copy memory from device to host
5. Release all memory



API correspondence

Function	CUDA	OpenCL
Allocate Memory	`cudaMalloc`	`clCreateBuffer`
Transfer Memory	`cudaMemcpy`	`clWriteBuffer` `clReadBuffer`
Call Kernel	$<<< x, y >>>$	`clEnqueueNDRange` `clSetKernelArg`
Block Identifier	`blockIdx`	`get_group_id`
Thread Identifier	`threadIdx`	`get_local_id`
Release Memory	`cudaFree`	`clReleaseMemObject`

Source code comparison of parallel to sequential

```
  /*
   *  PARALLEL CODE
   */

//Set GPU execution parameters
_GPUinit(16,16,4,4);
_XPUmalloc(h_A,"float*",mem_size_A);
_XPUmalloc(h_B,"float*",mem_size_B);
_XPUmalloc(h_C,"float*",mem_size_C);

initarray(h_A,h_B);
_GPUcall("arrayAdd", in(h_A,h_B),out(h_C));
printArray(h_C);

//Release Memory
_XPUrelease();
```

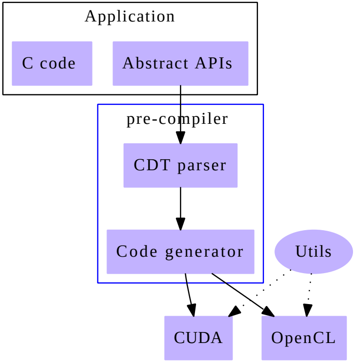
Generated code for CUDA and OpenCL

```
/*
 * CUDA code _XPUmalloc(h_C,"float*",memsize_C)
 */
float* h_C = (float*) malloc (mem_size_C);
float* _GEN_PREFIX17;
cutilSafeCall(cudaMalloc((void**)&_GEN_PREFIX17,
              mem_size_C));
```

```
/*
 * OPenCL code _GPUcall("arrayAdd", in(h_A,h_B),out(h_C));
 */
err = clSetKernelArg(kernel[0],0,sizeof(cl_mem),&h_A);
err |= clSetKernelArg(kernel[0],1,sizeof(cl_mem),&h_B);
err |= clSetKernelArg(kernel[0],2,sizeof(cl_mem),&h_C);

assert(err == CL_SUCCESS);
clFinish(cmd_queue);

//Read the results back to host
err = clEnqueueReadBuffer(cmd_queue,h_C,CL_TRUE,0,mem_size_C,
                          (void*)_GEN_PREFIX20,
```

Overview

1. C Code with additional predefined functions
2. Function calls (Abstract APIs) changed to GPU code
3. Frequently used block of code are saved as functions in Utils

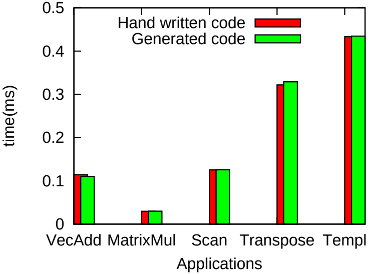


Experimentation

Application	CUDA	CPP	Abstr	# vars,lines	use
Vector Addition	29	15	13	3,16	6
Matrix Multiplication	28	14	12	3,14	6
Scan Test Cuda	82	NA	72	1,10	12
Transpose	39	17	26	2,13	8
Template	25	13	13	2,12	6

Table: Source code analysis of CUDA, CPP and Abstract API

Execution time comparison



- Generated code performs as efficient as hand written for different problems

Related Works

- *hi*CUDA : Use directives in C code, tightly coupled with CUDA
- CUDA-lite : Use annotations in base code, targeted for CUDA programmers
- CGIS : A new GPU programming language with support for many devices, OpenCL not addressed
- CUPP : A tool to integrate CUDA into existing C++ applications
- Sh : Abstract layer for GPU languages but in graphics domain

Conclusion and future work

Conclusion

- An approach to improve GPU programming
- Static code analysis for host code
- No Performance loss(hand written code versus generated code)

Future work

- Need to generate kernel code for complete evaluation

CUDACL: GPU programming in Eclipse IDE