Karlsruhe Institute of Technology

GPGPU Computing with OpenCL

Matthias Vogelgesang (IPE), Daniel Hilk (IEKP)

Institute for Data Processing and Electronics, Institut für Experimentelle Kernphysik

Motivation

- More data is generated, more data has to be processed and analyzed
- Despite Moore’s law, CPUs hit a performance wall
- GPU architectures *can* give a higher throughput and better performance

GPU advantages

Karlsruhe Institute of Technology

Why are GPUs good at what they do?

- GPUs are heavily optimized towards pixelation of 3D data
- GPUs have flexible, programmable pipelines
- Architecture consists of many but rather simple compute cores
- Instruction set is tailored towards math and image operations

Some numbers of NVIDIAs GTX Titan flagship

- 6 GB at 288.4 GB/s
- 4500 (SP) / 1500 (DP) GFLOPs (equivalent of supercomputer in 2000)
- 250 W power consumption

Limitations

There are no silver bullets

- Optimal performance with regular, parallel tasks
- High operations-per-memory-access ratios[1]
- Bus can become a bottleneck[2]
- Limited main memory, thus partitioning might be necessary

Think about your algorithm first

- Cliché quote: “premature optimization is the root of all evil”
- $O(c^n)$ is slow, no matter where you run it

---

[1] 4500 GFLOPS / 288.4 GB/s = 16 FLOP/B

[2] 4500 GFLOPS / 16 GB/s (PCIe 3.0 x16) = 280 FLOP/B

History and Background

Development of GPGPU abstractions

- Early research prototypes (e.g. Brook) used OpenGL shaders
- NVIDIA presented CUDA in 2007
- OpenCL initiated by Apple first released in 2008/09
- High-level pragmas in OpenACC à la OpenMP since 2012

Why OpenCL?

- Open, vendor-neutral standard
- Cross-platform support (Linux, Windows, Mac)
- Multiple hardware platforms (CPUs, GPUs, FPGAs)

OpenCL concepts

Programming model

Karlsruhe Institute of Technology

Platform

- A host controls $\geq$ 1 *platforms* (e.g. vendor SDKs)
- A platform consists of $\geq$ 1 *devices*
- The host manages resources and schedules execution
- The devices execute code assigned to them by the host

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Programming model

Platform

- A host controls ≥ 1 *platforms* (e.g. vendor SDKs)
- A platform consists of ≥ 1 *devices*
- The host manages resources and schedules execution
- The devices execute code assigned to them by the host

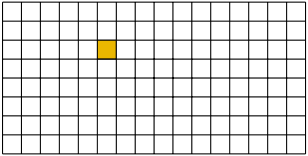
Devices

- A device has ≥ 1 *compute units*
- Each CU has ≥ 1 *processing elements*
- How CUs and PEs are mapped to hardware is *not* specified

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid

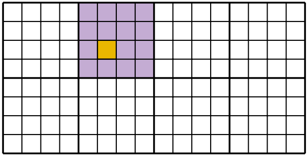


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Execution model

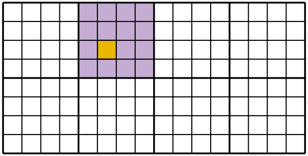
Karlsruhe Institute of Technology

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups



Execution model

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups
- Work groups are scheduled on one or more CUs

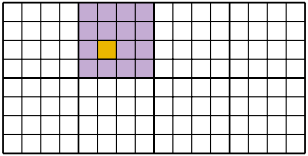


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Execution model

Karlsruhe Institute of Technology

- Work is arranged as work items on a 1D, 2D or 3D grid
- Grid is split into work groups
- Work groups are scheduled on one or more CUs
- Work items are executed on PEs



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Kernel

Karlsruhe Institute of Technology

- A kernel is a piece of code executed by *each* work item
- In most cases it corresponds to the innermost body of a for loop, e.g. from

  ```
  for (int i = 1; i < N-1; i++)
      x[i] = sin(y[i]) + 0.5 * (x[i-1] + x[i+1]);
  ```

  you would extract the kernel

  ```
  x[i] = sin(y[i]) + 0.5 * (x[i-1] + x[i+1]);
  ```

- A kernel has implicit parameters to identify itself
  - Location relative to the work group
  - Location relative to the global grid
  - Number of work groups/items

Memory model

Memory, buffers and images

- Host *cannot* access device memory directly and vice versa
- Buffers to transfer data between host and device memory
- Images are structured buffers

Device memory

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

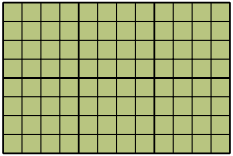
Memory model

Memory, buffers and images

- Host *cannot* access device memory directly and vice versa
- Buffers to transfer data between host and device memory
- Images are structured buffers

Device memory

Global host-accessible, read/write-able by *all* work items



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Memory model

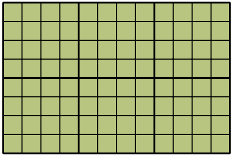
Memory, buffers and images

- Host *cannot* access device memory directly and vice versa
- Buffers to transfer data between host and device memory
- Images are structured buffers

Device memory

Global host-accessible, read/write-able by *all* work items

Constant host-accessible, read-only by *all* work items



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

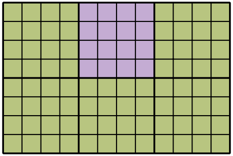
Memory model

Memory, buffers and images

- Host *cannot* access device memory directly and vice versa
- Buffers to transfer data between host and device memory
- Images are structured buffers

Device memory

Global	host-accessible, read/write-able by *all* work items
Constant	host-accessible, read-only by *all* work items
Local	local to a work group



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

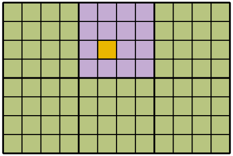
Memory model

Memory, buffers and images

- Host *cannot* access device memory directly and vice versa
- Buffers to transfer data between host and device memory
- Images are structured buffers

Device memory

Global	host-accessible, read/write-able by *all* work items
Constant	host-accessible, read-only by *all* work items
Local	local to a work group
Privat	local to a work item



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

OpenCL API

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Implementations

Vendor	Rev.	GPU	CPU	FPGA	OS
NVIDIA	1.1	✓	✗	✗	Linux, Windows, Apple
AMD	1.2	✓	✓	✗	Linux, Windows, Apple
Intel	1.2	✓	✓	✗	Linux, Windows
Apple	1.1[1]	✓	✓	✗	Apple
Altera	1.0	✗	✗	✓	Linux, Windows

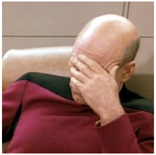
[1] OpenCL 1.2 from OS X 10.9

Prerequisites

- OpenCL is specified as a C API and a kernel language
- Link against `-lOpenCL` — generic driver loads implementation at run-time
- Header location depends on host platform ...

```
/* UNIX and Windows */
#include <CL/cl.h>

/* Apple */
#include <OpenCL/cl.h>
```



Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Kernel syntax

- Written in a C99 superset
- Address space specifiers (`global` and `local`)
- Work item and math related builtins
- Vector types (e.g. `int4`, `float3`, …)

```
kernel void
scale_vector (global float *output,
              global float *input,
              float scale)
{
    int idx = get_global_id (0);    /* global location */
    output[idx] = scale * input[idx];
}
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Querying all platforms

```
cl_uint n_platforms;
cl_platform_id *platforms = NULL;

e = clGetPlatformIDs (0, NULL, &n_platforms);

platforms = malloc (n_platforms * sizeof (cl_platform_id));

e = clGetPlatformIDs (n_platforms, &platforms, NULL);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Querying devices of one platform

```
cl_uint n_devices;
cl_device_id *devices = NULL;

e = clGetDeviceIDs (platforms[0], CL_DEVICE_TYPE_ALL,
                    0, NULL, &n_devices);

devices = malloc (n_devices * sizeof (cl_device_id);

e = clGetDeviceIDs (platforms[0], CL_DEVICE_TYPE_ALL,
                    n_devices, &devices, NULL);

/* If you don't use it anymore, decrement the reference */
e = clReleaseDevice (device);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Device contexts

Resources are shared between devices in the same context, thus contexts model application specific behaviour:

```
cl_context context;

context = clCreateContext (NULL, n_devices, devices,
                           NULL, NULL, &err);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Buffer objects

Buffers are created in a context. At run-time, the OpenCL environment decides when memory is transfered to a specific device.

```
size_t size;
cl_mem dev_input;
cl_mem dev_result;

size = 1024 * 1024 * sizeof (float);
dev_input = clCreateBuffer (context, CL_MEM_READ_ONLY,
                             size, NULL, &err);
dev_result = clCreateBuffer (context, CL_MEM_WRITE_ONLY,
                              size, NULL, &err);
```

Command queues

Device commands (data transfer, kernel launches ...) are enqueued in one command queue per device:

```
cl_command_queue queue;

queue = clCreateCommandQueue (context, devices[0], 0, &err);
```

The third parameter can be used to toggle out of order execution and profiling.

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Transfering data

```
e = clEnqueueWriteBuffer (queue, dev_input,
                          TRUE, /* blocking call? */
                          0, size,
                          host_input,
                          0, NULL, NULL);
```

Building kernel code

Kernel code is compiled at run-time because the target hardware is not necessarily known at compile-time (…and allows cool stunts like run-time code generation)

```
cl_program program;
cl_kernel kernel;

/* Create and build program */
program = clCreateProgramWithSource (context, 1, source,
                                     NULL, &e);
e = clBuildProgram (program, n_devices, devices,
                    NULL, NULL, NULL);

/* Extract kernel */
kernel = clCreateKernel (program, "scale_vector", &e);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Launching kernels

```
size_t global_work_size[] = { 1024 };
size_t global_work_offset[] = { 0 };
cl_event event;

e = clEnqueueNDRangeKernel (queue, kernel,
                            1, /* grid dimensions */
                            global_work_offset,
                            global_work_size,
                            0, NULL, &event);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Events

All commands accept and return `cl_event` objects

```
cl_int clEnqueueXXX (...,
                     cl_uint wait_list_length,
                     const cl_event *wait_list,
                     cl_event *event);
```

that can be used to

```
/* Wait for one or more events */
e = clWaitForEvents (1, &event);

/* Query event information */
e = clGetEventInfo (event, CL_EVENT_COMMAND_EXECUTION_STATUS,
                    sizeof (cl_int), &result, NULL);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Kernel synchronization

Events are also used to ensure correct enqueuing order in out-of-order queues:

```
clEnqueueNDRangeKernel (queue, kernel_foo,
                        ...,
                        NULL, NULL, &foo_event);

clEnqueueNDRangeKernel (queue, kernel_bar,
                        ...,
                        1, &foo_event, &bar_event);

clReleaseEvent (foo_event);
clReleaseEvent (bar_event);
```


Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik

Work item synchronization

Guarantee that all work items are waiting at the same point before proceeding:

```
barrier (mem_fence_flags);
```

Make sure that all the other work items read the same values:

```
mem_fence (mem_fence_flags);
write_mem_fence (mem_fence_flags);
read_mem_fence (mem_fence_flags);
```

`mem_fence_flags` must be a combination of

- `CLK_LOCAL_MEM_FENCE`: for guarantees inside a work group
- `CLK_GLOBAL_MEM_FENCE`: across all work items

Considerations

- All resources are reference-counted $\rightarrow$ release them when not used!
- Every call returns an error code $\rightarrow$ check all of them!
- Using `double` will decrease performance by factor two (if it works at all)

Institute for Data Processing and Electronics,
Institut für Experimentelle Kernphysik