Lecture 20—About OpenCL

ECE 459: Programming for Performance

March 26, 2013

Introduction

OpenCL: coding on a heterogeneous architecture.

- No longer just programming the CPU;
  will also leverage the GPU.

OpenCL = Open Computing Language.
Usable on both NVIDIA and AMD GPUs.

SIMT

Another term you may see vendors using:

- **S**ingle **I**nstruction, **M**ultiple **T**hreads.
- Runs on a vector of data.
- Similar to SIMD instructions (e.g. SSE).
  However, the vector is spread out over the GPU.

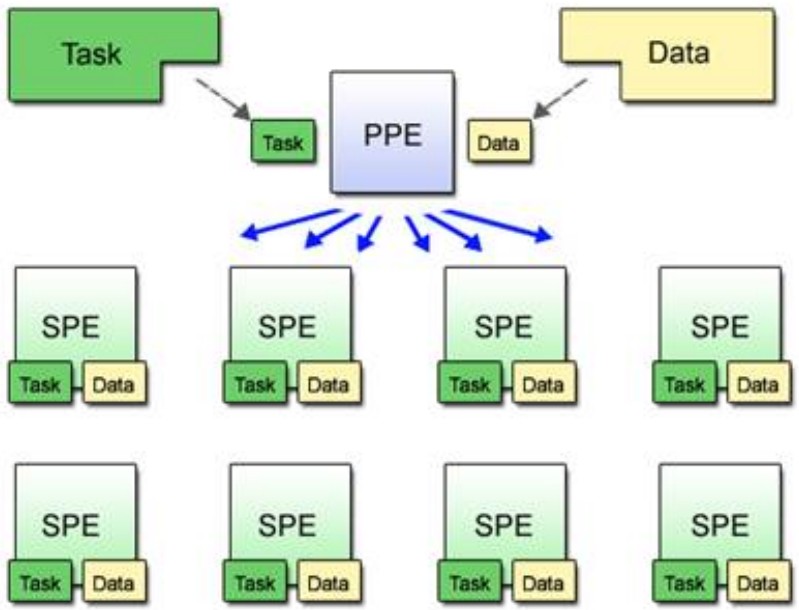
Other Heterogeneous Programming Examples

- PlayStation 3 Cell
- CUDA

(PS3) Cell Overview

Cell consists of:

- a PowerPC core; and
- 8 SIMD co-processors.



(from the Linux Cell documentation)

CUDA Overview

Compute Unified Device Architecture:
NVIDIA's architecture for processing on GPUs.

"C for CUDA" predates OpenCL,
NVIDIA supports it first and foremost.

- May be faster than OpenCL on NVIDIA hardware.
- API allows you to use (most) C++ features in CUDA; OpenCL has more restrictions.

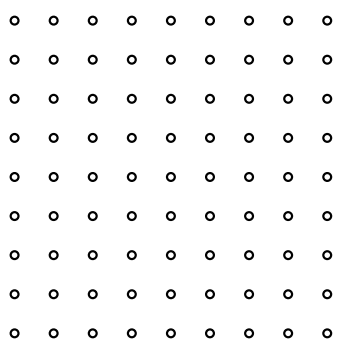
GPU Programming Model

The abstract model is simple:

- Write the code for the parallel computation (*kernel*) separately from main code.
- Transfer the data to the GPU co-processor (or execute it on the CPU).
- Wait ...
- Transfer the results back.

Data Parallelism

- Key idea: evaluate a function (or *kernel*) over a set of points (data).



Another example of data parallelism.

- Another name for the set of points: *index space*.
- Each point corresponds to a **work-item**.

Note: OpenCL also supports *task parallelism* (using different kernels), but documentation is sparse.

Work-Items

Work-item: the fundamental unit of work in OpenCL.
Stored in an *n*-dimensional grid (ND-Range); 2D above.

OpenCL spawns a bunch of threads to handle work-items.
When executing, the range is divided into **work-groups**, which execute on the same compute unit.

The set of compute units (or cores) is called something different depending on the manufacturer.

- NVIDIA - *warp*
- AMD/ATI - *wavefront*

Work-Items: Three more details

One thread per work item, each with a different thread ID.

You can say how to divide the ND-Range into work-groups, or the system can do it for you.

Scheduler assigns work-items to warps/wavefronts until no more left.

Shared Memory

There are many different types of memory available to you:

- private memory: available to a single work-item;
- local memory (aka "shared memory"): shared between work-items belonging to the same work-group; like a user-managed cache;
- global memory: shared between all work-items as well as the host;
- constant memory: resides on the GPU and cached. Does not change.

There is also host memory (normal memory); usually contains app data.

Example Kernel

Here's some traditional code to evaluate $C_i = A_i B_i$:

```
void traditional_mul(int n,
                     const float *a,
                     const float *b,
                     float *c) {
  int i;
  for (i = 0; i < n; i++) c[i] = a[i] * b[i];
}
```

And as a kernel:

```
kernel void opencl_mul(global const float *a,
                       global const float *b,
                       global float *c) {
  int id = get_global_id(0);  // dimension 0
  c[id] = a[id] * b[id];
}
```

Restrictions when writing kernels in OpenCL

It's mostly C, but:

- No function pointers.
- No bit-fields.
- No variable-length arrays.
- No recursion.
- No standard headers.

OpenCL's Additions to C in Kernels

In kernels, you can also use:

- Work-items.
- Work-groups.
- Vectors.
- Synchronization.
- Declarations of memory type.
- Kernel-specific library.

Branches in kernels

Kernels contain code, which can contain branches (if statements). Hence, computation from each work-item can branch arbitrarily. The hardware will execute *all* branches that any thread in a warp executes—can be slow!

In other words: an `if` statement will cause each thread to execute both branches; we keep only the result of the taken branch.

A loop will cause the workgroup to wait for the maximum number of iterations of the loop in any work-item.

Note: when you set up work-groups, best to arrange for all work-items in a workgroup to execute the same branches.

Synchronization

Different workgroups execute independently.
You can only put barriers and memory fences between work-items in the same workgroup.

OpenCL supports:

- Memory fences (load and store).
- Barriers.
- `volatile` (beware!)

Summary

Brief overview of OpenCL and its programming model.

Many concepts are similar to plain parallel programming (more structure).