Distributed GPGPU Computing

Martin Stumpf
Ste||ar Group, LSU

September 25, 2014
Table of Contents

1. GPGPU - Overview
   - GPGPU
   - OpenCL

2. The MPI way

3. The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4. Implementing "Hello, World!" with HPXCL

5. Performance and Scaling
   - The Mandelbrot Renderer
   - Results
Outline

1 GPGPU - Overview
   - GPGPU
   - OpenCL

2 The MPI way

3 The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4 Implementing "Hello, World!" with HPXCL

5 Performance and Scaling
   - The Mandelbrot Renderer
   - Results
The theoretical calculation power of a GPU is much higher than a CPU.

Example

CPU (Intel Xeon E5-2670 v3):
- 12 Cores, 2.3 GHz, 32 FLOPS/cycle
  - 884 GFLOPS
- Prize: $1500

GPU (NVidia Tesla K40):
- 2880 Cores, 745 MHz, 2 FLOPS/cycle
  - 4291 GFLOPS
- Prize: $4000

So, what computational tasks are actually suitable for GPGPU?
Problems suitable for GPGPU

Every problem that fits the **SPMD** programming scheme, can benefit greatly from GPGPU.

Examples:
- Fluid Simulations
- Mathematical Vector Operations
- Image Processing
- Stencil Based Simulations

SPMD based programming languages:
- CUDA (NVidia)
- OpenCL (Vendor independent)
- C++ AMP (Microsoft)
An OpenCL device is split in two components:

- The **Buffer**: Represents memory on the device
- The **Kernel**: A C-style function that modifies one or multiple elements of a buffer

Kernel source code stays plain text and gets compiled at runtime

⇒ OpenCL programs are device independent

Kernel executions on the device run asynchronous to the host program
OpenCL

Host

write

read

execute

Buffer

Kernel

OpenCL Device
Outline

1. GPGPU - Overview
   - GPGPU
   - OpenCL

2. The MPI way

3. The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4. Implementing "Hello, World!" with HPXCL

5. Performance and Scaling
   - The Mandelbrot Renderer
   - Results
Disadvantages:

- MPI and OpenCL are independent from each other
  - Connection between computation and data exchange has to be implemented manually
- Every OpenCL device can only be accessed within its own node
- If no further methodologies are used, the whole cluster will run in lockstep
Distributed OpenCL with MPI

MPI

Node

User Program

OpenCL Device

Node

User Program

OpenCL Device

Node

User Program

OpenCL Device
Outline

1 GPGPU - Overview
   - GPGPU
   - OpenCL

2 The MPI way

3 The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4 Implementing "Hello, World!" with HPXCL

5 Performance and Scaling
   - The Mandelbrot Renderer
   - Results
What is HPX?
- A scaling C++ runtime system for parallel and distributed applications
- Based on the ParalleX model

Advantages for distributed OpenCL:
- Global Namespace
- Cluster as "one large machine" (MPI: every Node is autonomous)
- Data dependencies (futures) (MPI: Send-Wait)
HPXCL

- Is our implementation of a distributed OpenCL runtime
- Uses HPX as distribution mechanism
- Wraps every OpenCL datastructure in an HPX component:

<table>
<thead>
<tr>
<th>OpenCL</th>
<th>HPXCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl_device</td>
<td>hpx::opencl::device</td>
</tr>
<tr>
<td>cl_program</td>
<td>hpx::opencl::program</td>
</tr>
<tr>
<td>cl_kernel</td>
<td>hpx::opencl::kernel</td>
</tr>
<tr>
<td>cl_mem</td>
<td>hpx::opencl::buffer</td>
</tr>
<tr>
<td>cl_event</td>
<td>hpx::opencl::event</td>
</tr>
<tr>
<td></td>
<td>(soon: hpx::future)</td>
</tr>
</tbody>
</table>
Distributed OpenCL with HPXCL

HPX Global Namespace

User Program

Locality

OpenCL Device Component

Locality

OpenCL Device

Component

Locality

OpenCL Device

Component

Locality

OpenCL Device

Component
Effect on distributed GPGPU programming

- Abstracting the whole cluster as one machine
- Simpler, no need to think in a distributed way
- Data dependencies
  - faster due to prevention of lockstep
  - possible to apply standard OpenCL synchronization techniques
- Seamless integration of additional OpenCL nodes into the system
- Possibility to run heterogeneous nodes/devices in one system
- Easy to port non-distributed code to distributed OpenCL whilst maintaining descent scaling
Outline

1. GPGPU - Overview
   - GPGPU
   - OpenCL

2. The MPI way

3. The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4. Implementing "Hello, World!" with HPXCL

5. Performance and Scaling
   - The Mandelbrot Renderer
   - Results
Implementing "Hello, World!" with HPXCL

Retrieving an OpenCL device:

```cpp
// Get list of available OpenCL devices
std::vector<hpx::opencl::device> devices =
    hpx::opencl::get_all_devices( CL_DEVICE_TYPE_ALL,
                                  "OpenCL 1.1" ).get();

// Check whether there are any devices
if(devices.size() < 1)
{
    hpx::cerr << "No OpenCL devices found!" << hpx::endl;
    return hpx::finalize();
}

// Choose the first device found
hpx::opencl::device cldevice = devices[0];
```
Implementing "Hello, World!" with HPXCL

Creating a buffer:

```cpp
// Create a buffer
hpx::opencl::buffer buf =
    cldevice.create_buffer(CL_MEM_READ_WRITE, 14);
```

Writing to the buffer:

```cpp
// Create some data
const char some_data[] = {
    '\x47', '\x64', '\x6b', '\x6b',
    '\x6e', '\x2b', '\x1f', '\x56',
    '\x6e', '\x71', '\x6b', '\x63',
    '\x20', '\xff' };

// Write data to buffer
auto write_done = buf.enqueue_write(0, 14, some_data);
```
Implementing "Hello, World!" with HPXCL

Creating a kernel:

```cpp
const char hello_world_src[] =
  "  __kernel void hello_world(__global char * buf)  
  {  
    size_t tid = get_global_id(0);  
    buf[tid] = buf[tid] + 1;  
  }  
";

// Create the program
hpx::opencl::program prog =
  cldevice.create_program_with_source(hello_world_src);
prog.build();

// Create the kernel
hpx::opencl::kernel hello_world_kernel =
  prog.create_kernel("hello_world");
```
Connecting the buffer to the kernel:

```cpp
// Set the buffer as kernel argument
hello_world_kernel.set_arg(0, buf);
```

Executing a kernel:

```cpp
// Create the work dimensions
hpx::opencl::work_size<1> dim;
dim[0].offset = 0;
dim[0].size = 14;

// Run the kernel
auto kernel_done = hello_world_kernel.enqueue(dim,
                                            write_done);
```
Implementing "Hello, World!" with HPXCL

- Reading the result from the buffer:

```cpp
// Read from the buffer
auto read_result = buf.enqueue_read(0, 14, kernel_done);

// Get the data (blocking call)
hpx::serialize_buffer<char> data_ptr = read_result.get();

// Print the data. This will print "Hello, World!".
hpx::cout << data_ptr.data() << hpx::endl;

// Gracefully shut down HPX
return hpx::finalize();
```
Outline

1. GPGPU - Overview
   - GPGPU
   - OpenCL

2. The MPI way

3. The HPX way
   - Advantages
   - HPXCL
   - Layout
   - Effect on distributed GPGPU

4. Implementing "Hello, World!" with HPXCL

5. Performance and Scaling
   - The Mandelbrot Renderer
   - Results
The Mandelbrot set is based on the complex series:

\[ z_{n+1} = z_n^2 + c, \quad z_0 = 0 \]

The set itself is defined as: \[ \{ c \in \mathbb{C} : \exists s \in \mathbb{R}, \forall n \in \mathbb{N}, |z_n| < s \} \]

Creating Mandelbrot Images

The mandelbrot set can be visualized by associating every pixel with a coordinate \( (x, y) \) and then setting \( c = y \cdot i + x \).

Coloring pixels by how fast the series diverges can create impressive images.
The Mandelbrot Renderer
Stats for Nerds

- Resolution: 2560×1920
- Smoothing: 8×8 Supersampling
- Bailout: 10000
- Maximum iterations: 50000
- GPUs: 32× NVidia Tesla K20
- Render time:
The Mandelbrot Renderer

Stats for Nerds

- Resolution: 2560x1920
- Smoothing: 8x8 Supersampling
- Bailout: 10000
- Maximum iterations: 50000
- GPUs: 32x NVidia Tesla K20
- Render time: 0.6 seconds
Mandelbrot Benchmark - Speedup

- Optimal Speedup
- Measured Speedup
Parallel Efficiency

Mandelbrot Benchmark - Parallel Efficiency

Parallel Efficiency

Number of GPUs

Optimal Efficiency
Measured Efficiency
We combined the renderer with the Google Maps API v3:
Google Maps Live Demo

- **URL:** [http://mstumpf2-2.lsu.edu:8080/](http://mstumpf2-2.lsu.edu:8080/)
- Please use **Firefox**!