

OpenCL-based Actors in C++

Raphael Hiesgen

raphael.hiesgen@haw-hamburg.de

iNET RG, Department of Computer Science
Hamburg University of Applied Sciences

July 25, 2013



Hochschule für Angewandte
Wissenschaften Hamburg

Hamburg University of Applied Sciences

Agenda



- 1** GPU Computing
- 2 Introduction to OpenCL
- 3 Actors with `libcppa`
- 4 OpenCL-enabled Actors
- 5 Conclusion & Outlook

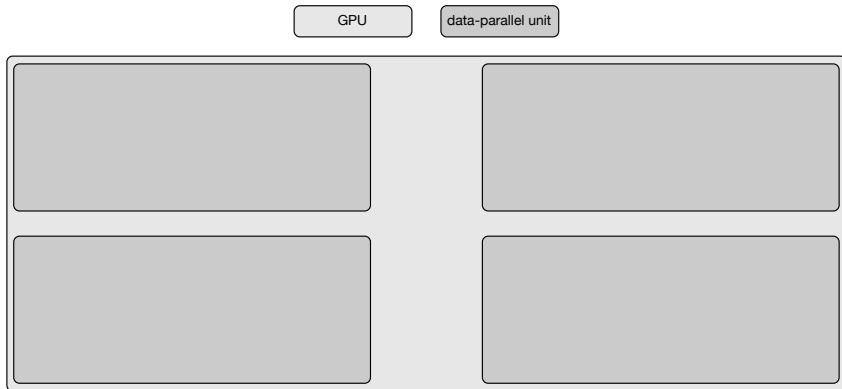
- GPUs evolved to data-parallel programmable units
- Easily outperform CPUs in several use cases
- Heterogenous hardware is widely available, even in mobile devices
- Programmable with frameworks (e.g., CUDA and OpenCL)

GPU Architecture

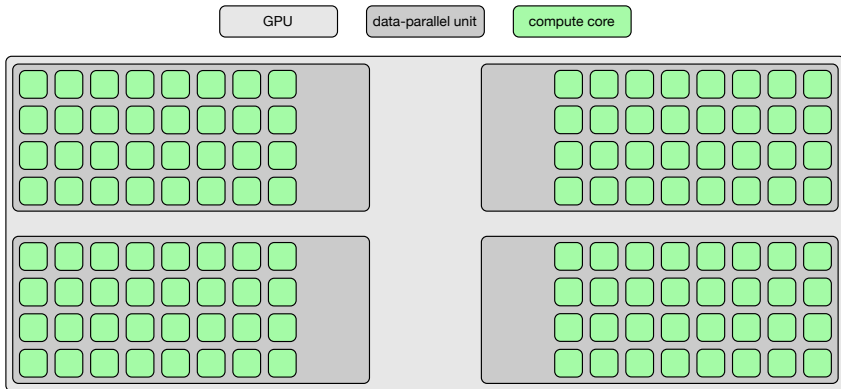
GPU

A diagram showing a small rounded rectangular box labeled "GPU" at the top center, connected to a large, empty rectangular box below it, representing the GPU architecture.

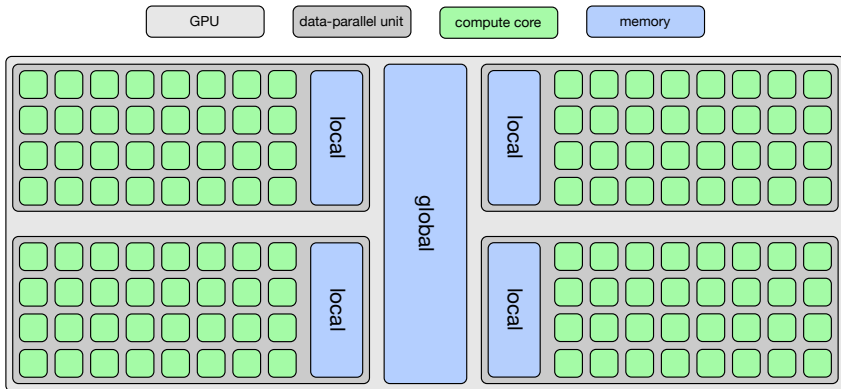
GPU Architecture



GPU Architecture



GPU Architecture



Agenda



- 1 GPU Computing
- 2 Introduction to OpenCL**
- 3 Actors with `libcppa`
- 4 OpenCL-enabled Actors
- 5 Conclusion & Outlook

Introduction to OpenCL



- Open Computing Language
- Standard for parallel general purpose computing
- Platform-independent (CPU & GPU)
- Developed by the Khronos Group
- First released in 2008
- Latest stable release: 1.2 (November 2011)
- Next release: 2.0 (Provisional Specification, July 2013)

Introduction to OpenCL



- Program consists of two parts: host and device
- Communication via command queue
- Code executed on a device is called kernel
 - Written in a C dialect
 - Compiled at runtime
 - Always returns `void`
 - Arrays as input and output parameters

- A kernel is executed in an N-dimensional index space
- The dimensions must be specified by the programmer
 - Up to three dimensions
 - Limited by the hardware
- Each N-tuple is a global ID
 - Each N-tuple refers to a point in the index space
 - Represents a single kernel execution
 - Called a work-item
- Work-items are bundled in work-groups
 - Each work-group has an ID
 - Each work item has local, group-dependent ID

Kernel Execution II

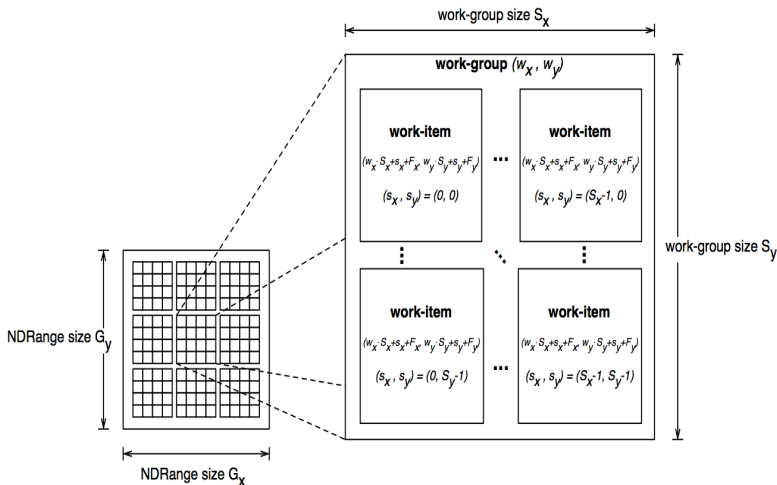


Figure : Index space for a two dimensional kernel. [1]

Memory Areas

- Global, constant, local, and private memory areas
- Different access speed and modifiers

area	host access	ownership	speed
global	read, write	global	slow
constant	read, write	global	slow
local	none	work-group	fast
private	none	work-item	fast

OpenCL Setup Steps

- Find a platform (e.g. Apple, Nvidia CUDA)
- Find the available devices (either GPUs or CPUs)
- Create a context
- Create a command queue
- Create a program object from kernel source
- Compile a kernel
- Create and initialize memory objects on the device
- Set memory objects as arguments for the kernel
- Enqueue kernel for execution

Agenda



- 1 GPU Computing
- 2 Introduction to OpenCL
- 3 Actors with libcppa**
- 4 OpenCL-enabled Actors
- 5 Conclusion & Outlook

The Actor Model



Actors are concurrent entities, that ...

- Communicate via message passing
- Do not share state
- Can create (“spawn”) new actors
- Can monitor other actors

- High-level, explicit communication: no locks, no implicit sharing
- Applies to both concurrency *and* distribution
 - Divide workload by spawning actors
 - Network-transparent messaging
 - Run transparently on heterogenous hardware

Agenda



- 1 GPU Computing
- 2 Introduction to OpenCL
- 3 Actors with `libcppa`
- 4 OpenCL-enabled Actors**
- 5 Conclusion & Outlook

OpenCL-enabled Actors



- Integration of OpenCL actors into `libcppa`
 - Coherent programming model
 - Network transparency
- High-level abstraction
 - OpenCL is handled in the background
 - Messaging handled by `libcppa`'s runtime
- A facade for each OpenCL actor runs on the CPU
 - Handles messages
 - Extracts kernel arguments from messages
 - Initiates the kernel execution on the OpenCL device

Example

- Matrix multiplication of two square matrices
- N-dimensional index space is necessary
 - One OpenCL dimension per matrix dimension
 - One work-item in each dimension per row / column
 - One kernel instance for each index in the matrix
- OpenCL actors are created by using `spawn_cl`
- Calculation is triggered by a message containing the input values
- Result is sent back to the client

The Kernel



```
constexpr const char* kernel_name = "matrix_mult";
constexpr const char* kernel_source = R"__(
    __kernel void matrix_mult(__global float* matrix1,
                               __global float* matrix2,
                               __global float* output) {
        size_t size = get_global_size(0);
        size_t x = get_global_id(0);
        size_t y = get_global_id(1);
        float result = 0;
        for (size_t idx = 0; idx < size; ++idx) {
            result += matrix1[idx + y * size]
                    * matrix2[x + idx * size];
        }
        output[x+y*size] = result;
    }
)__;
```

Usage Example



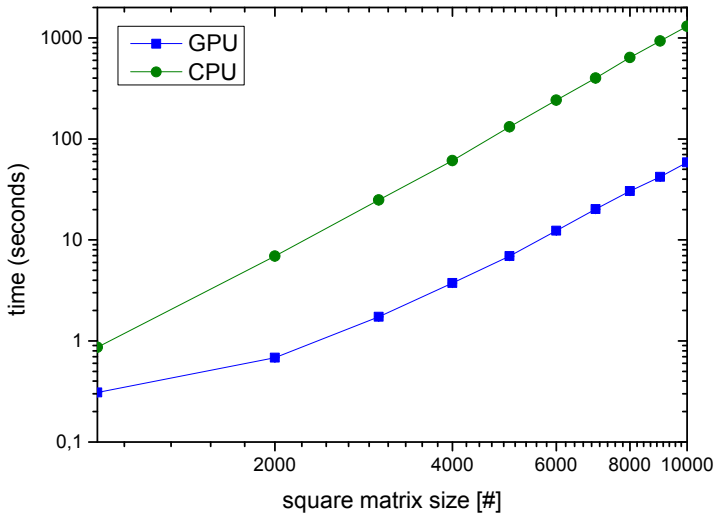
```
size_t matrix_size = 5;

vector<float> m1 = create_matrix(matrix_size);
vector<float> m2 = create_matrix(matrix_size);

auto worker =
    spawn_cl<float*(float*,float*)>(kernel_source,
                                   kernel_name,
                                   {matrix_size,
                                   matrix_size});

sync_send(worker, m1, m2).then(
    [](const vector<float>& result) {
        print_as_matrix(result);
    }
);
```

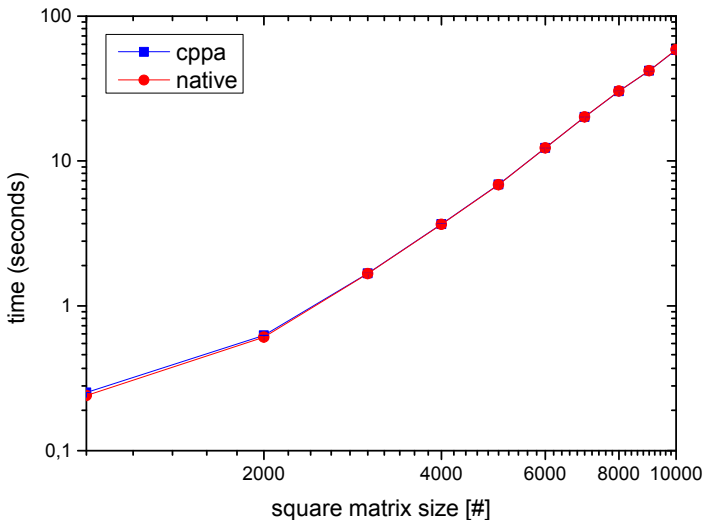
Speed Benchmark



Embedding OpenCL adds overhead:

- Actor creation
- Managing OpenCL resources
- Conversion from messages to kernel arguments
- Sent result messages

Overhead Benchmark



Agenda



- 1 GPU Computing
- 2 Introduction to OpenCL
- 3 Actors with `libcppa`
- 4 OpenCL-enabled Actors
- 5 Conclusion & Outlook

- GPUs
 - More cores than CPUs
 - Data-parallel architecture
- OpenCL-enabled actors
 - Easy setup via `spawn_cl`
 - Integrated into `libcppa`
 - Messages handled by `libcppa`'s runtime
 - Keep benefits, e.g., network transparency
- Performance
 - Outperforms CPUs for several use cases
 - Little overhead compared to native OpenCL

Future & Ongoing Work



- Memory management
- Handling of multiple GPUs
- Improvements in runtime performance
 - Prevent copying in 1:1 communication between OpenCL-enabled actors
- Spawn OpenCL-enabled actors on other machines at runtime

GPUs have several characteristics important to OpenCL.



- Number of parallel compute-cores
- Maximum $\frac{\text{work-items}}{\text{work-group}}$
- Clock frequency
- Global device memory
- Out of order execution

■ GPUs:

	GT 650M	Tesla C2075
compute cores	2	14
$\frac{\text{work-items}}{\text{work-group}}$	1024	1024
clock frequency [MHz]	900	1147
global device memory [MB]	1024	5375
out of order queue	false	true

■ CPU:

- 12 cores
- 2394 MHz
- x86_64 processor

-  A. Munshi, “The OpenCL Specification,” Khronos OpenCL Working Group, 2012.
-  D. Charousset and T. C. Schmidt, “libc++ - Designing an Actor Semantic for C++11,” in *Proc. of C++Now*, May 2013.



Thank you for your attention.
Questions?

iNET: <http://inet.cpt.haw-hamburg.de>