

A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices

Amnon Barak, Tal Ben-Nun, Ely Levy and Amnon Shiloh

Department of Computer Science, Hebrew University of Jerusalem, Israel

Outline

- **Introduction**

- **MGP**

- The Virtual OpenCL (VCL) Layer
- The API Layer
 - C++ API
 - OpenMP-like API
 - Scatter-Gather API

- **Performance**

- **Related Work**

- **Future Work**

Introduction

- **Many GPU programming is complicated**
 - Requires a-priori knowledge of number of devices
 - Or programming a job scheduler
 - Tough to manage and schedule jobs
 - Usually suitable for very specific tasks
 - Statically assigned devices
 - Hard to program
 - Requires many lines-of-code
 - Error checking
 - (Very) hard to debug

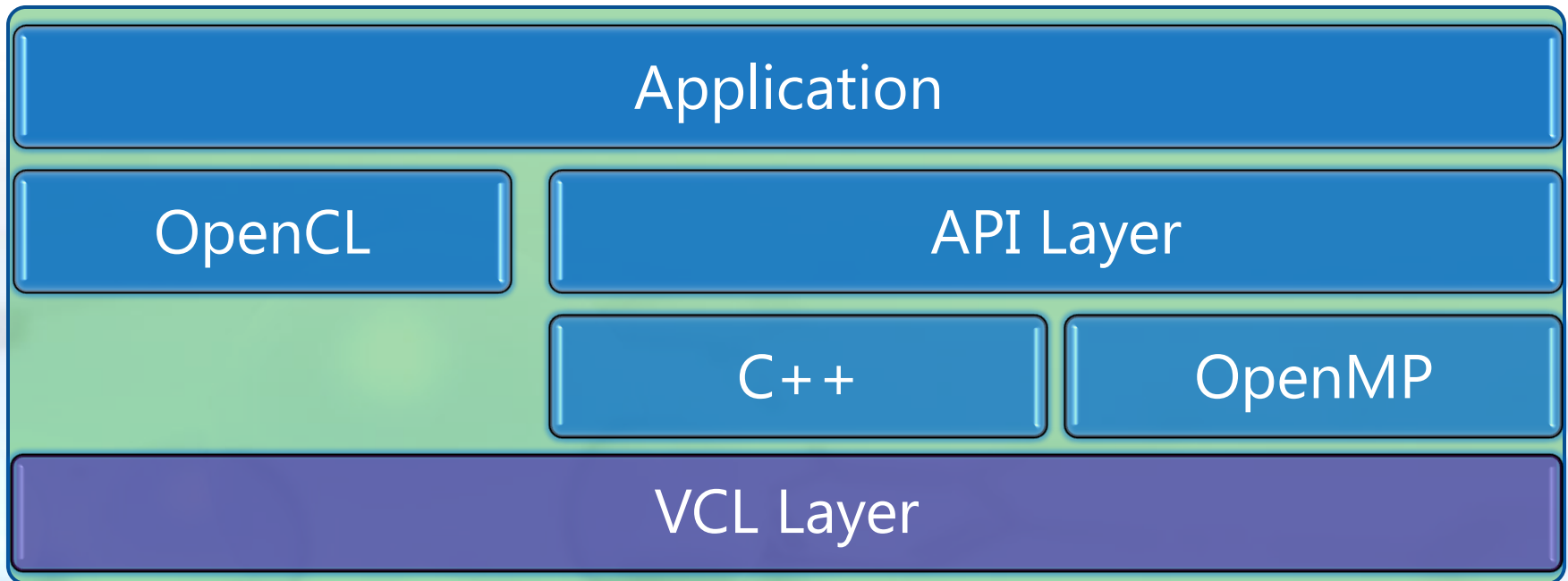
Introduction

- **Current implementations are based on MPI**
 - Examples: SHOC, MITHRA, TSUBAME, etc.
- **Result: It takes hundreds of lines of code for a simple many GPU task that requires an exchange in MPI**
- **The OpenMP approach for parallel programming depends on shared memory and is simpler to use**

Introducing The Many GPU Package (MGP)

- **A package for heterogeneous computing over many GPUs**
- **Consists of two layers:**
 - Virtual OpenCL (VCL) Layer – Runs unmodified OpenCL applications on a GPU cluster
 - API Layer – Easy large-scale heterogeneous programming API
- **Advantages**
 - Geared towards performance
 - Based on requirements of real HPC applications
 - Implements parallel programming paradigms e.g. Scatter-Gather
- **MGP is available at <http://www.mosix.org>**

MGP Layer Scheme



MGP – VCL Layer

- **Cluster-wide implementation of OpenCL**
 - Allows applications to transparently use cluster-wide devices (e.g. CPUs, GPUs)
- **Comprises a run-time environment that simulates a Single System Image (SSI)**
 - An application does not need to worry about device availability and origin
 - Full multiple OpenCL context support
- **Network latency is the main limiting factor**
 - VCL optimizes the amount of data transfers in the network

MGP – API Layer (C++ API)

- **A simple API that transparently runs kernels on many devices**
 - Object-oriented
 - Task-based
- **Computes task buffer-dependencies in real-time and schedules tasks to the cluster accordingly**
- **Designed so that the backend (e.g. CUDA, DirectCompute) is interchangeable**

C++ API Example

```
using namespace MGL;
```

```
// ...
```

```
Context::CreateContext("OpenCL");
```

```
int size = 256;
```

```
Buffer *gBuff = new Buffer(hBuff, sizeof(float) * size, INOUT, MEMORY);
```

```
Task *task = new Task(PROGRAMSTR, "", "sampleKernel", &gBuff, 1, 1, &size, NULL);
```

```
Context::GetContext().Enqueue(task);
```

```
task->wait(STATE_FINISHED);
```

```
delete gBuff; delete task;
```

```
Context::Destroy();
```

MGP – API Layer (OpenMP-like API)

- **An even simpler, but more restrictive API for MGP**
 - Based on OpenMP principles
 - Ease of programming, using “#pragma” annotations
- **Allows limited functionality of MGP**
 - Only basic tasks, no parallel programming paradigms

MGP – C++ vs. OpenMP

API Comparison

C++ API

```
const char *PROGRAM = " \
kernel void mul2(global float *dst) \
{ \
int id = get_global_id(0); \
dst[id] *= 2.0f; \
}";
using namespace MGL;
Context::CreateContext("OpenCL");
int dims = 256;
// Multiplying x by 2
Buffer *gX = new Buffer(x, sizeof(float) * dims,
                       INOUT, MEMORY);
Task *task = new Task(PROGRAM, "", "mul2", &gX,
                     1, 1, &dims, NULL);
Context::GetContext().Enqueue(task);
task->Wait(STATE_FINISHED);
delete gX; delete task;
Context::Destroy();
```

OPENMP-LIKE API

```
#pragma mgl kernel target(OpenCL) implements(mul2)
kernel void mul2(global float *dst)
{
int id = get_global_id(0);
dst[id] *= 2.0f;
}

int dims = 256;
// Multiplying x by 2
#pragma mgl task input(x) output(x) global(dims)
mul2(x);

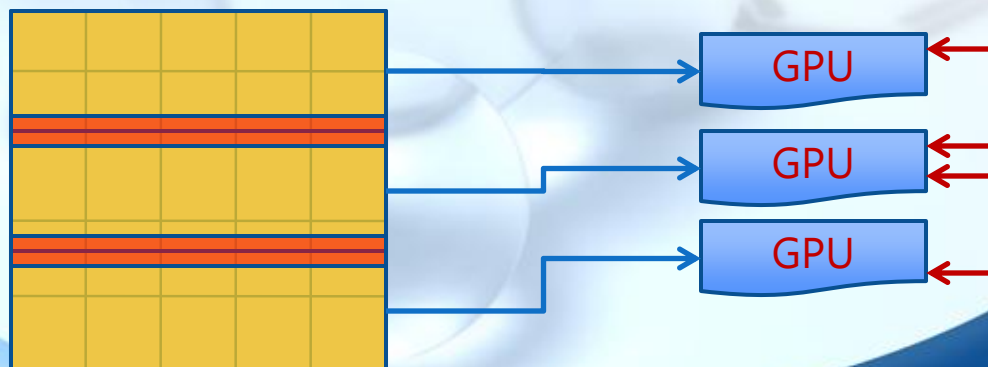
#pragma mgl taskwait
```

MGP – API Layer (Scatter-Gather API)

- **MGP provides a specialized API for Scatter-Gather tasks**
- **A parallel programming paradigm designed for tasks that:**
 - Scatter single/multiple buffers across many devices with the same kernel
 - May exchange data (e.g. boundary conditions) between kernels
 - The scattered buffer(s) are eventually gathered

Scatter-Gather Example

- **The Stencil2D benchmark from the Scalable Heterogeneous Computing (SHOC) benchmark was ported to MGP**
 - The MPI implementation spans ~655 lines-of-code
 - Our port is more scalable than the MPI version (using stripes instead of grid-blocks) and takes ~64 lines-of-code



Scatter-Gather Example

```
// Create context, buffers and scatterhandlers
for (int i = 0; i < iters; i++) {
    stenciltask = new ScatterTask(...);
    Context::Enqueue(stenciltask);
    if (i == iters - 1)
        stenciltask->Gather();
    else
        stenciltask->Exchange();
    delete stenciltask;
    swap(source, dest);
}
// Return the resulting matrix
source->GetData(...);

// Delete context, buffers and scatterhandlers
```

Performance



Performance – VCL Layer (Overhead)

This table shows the overhead of starting 1,000 minimal kernels consecutively, excluding compilation and buffer I/O times.

Note that the overhead of starting a single kernel by the VCL layer on a local device is on average $\sim 33\mu\text{s}$ and $\sim 85\mu\text{s}$ on a remote device.

Buffer Size	Native Time (ms)	MGP Local Diff. (ms)	MGP Remote Diff. (ms)
4 KB	26	30	81
64 KB	27	30	81
1 MB	45	30	81
4 MB	100	31	81
16 MB	321	33	86
64 MB	1207	37	91
256 MB	4762	38	92
512 MB	9498	37	92

Performance – API Layer (Overhead)

This table shows the overhead of the API layer by measuring 10,000 minimal kernels consecutively, both with and without compilation and buffer I/O times.

Note that the overhead of starting a single kernel by the API layer on a local device is only $\sim 3.88\mu\text{s}$ on average on a Linux machine and $\sim 32.70\mu\text{s}$ on a Windows machine.

Platform	Kernel Time (μs)	Total Time (ms)
Linux Native OpenCL	17.37	721.34
Linux MGL	21.25	332.56
Windows Native OpenCL	59.40	399.42
Windows MGL	92.10	308.42

Performance – API Layer

This table shows the performance of one of the SHOC benchmarks – FFT, with a 256 MB buffer.

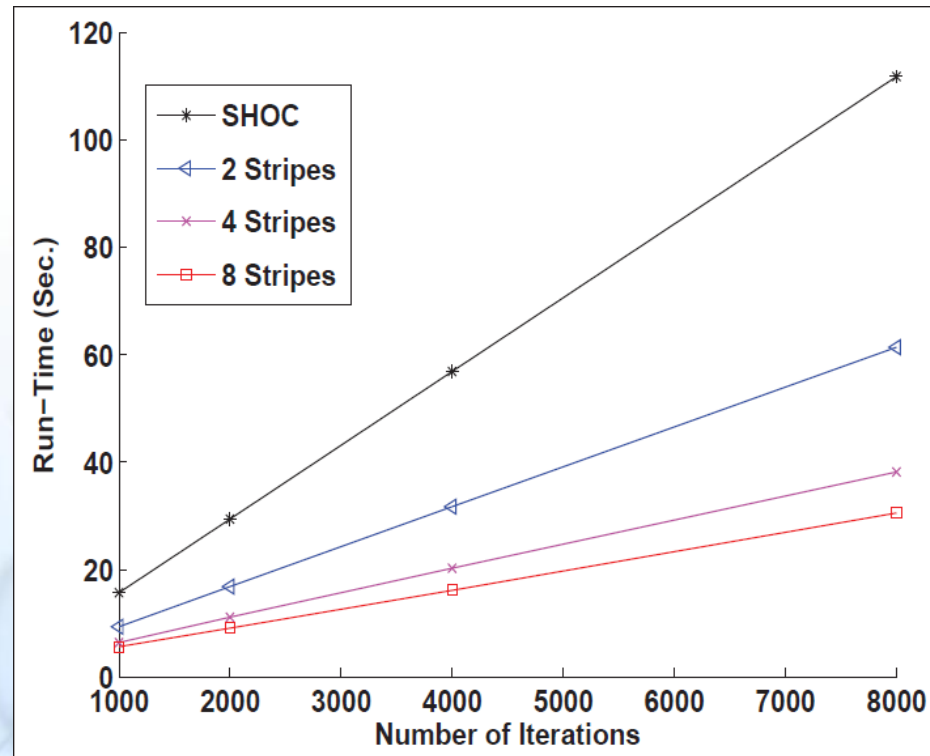
The program was ported to MGP's API layer and then run on 4 and 8 nodes in parallel. The resulting times and speedups are shown in the table.

Number of Iterations	Native Time (Sec.)	4 Nodes		8 Nodes	
		Time (Sec.)	Speedup	Time (Sec.)	Speedup
1000	42.34	19.27	2.19	16.29	2.60
2000	82.25	30.11	2.73	22.03	3.73
4000	162.17	52.58	3.08	33.37	4.86
8000	321.91	97.53	3.29	55.95	5.74

Performance – Scatter-Gather

This graph shows the various run-times of the Stencil2D benchmark as a function of the number of iterations.

The SHOC plot shows the performance of the original benchmark, whereas the 2, 4 and 8 stripes show the run-times of the version ported to MGP.



Related Work

- **Parallel Programming Paradigms**

- “Survey on parallel programming model” (Kasim et al., NPC 2008)

- **CUDA Under OpenMP**

- “A proposal to extend the OpenMP tasking model for heterogeneous architectures” (Aiguade et al., *IWOMP 2009*)

- **Remote CUDA (rCUDA)**

- Available at:
<http://www.hpca.uji.es/?q=node/36>

Future Work

- **More functionality for the OpenMP-like API**
 - For example, scatter-gather over the OpenMP API
- **More parallel programming paradigms**
 - e.g. MapReduce for tasks
- **Dynamic resource management**
 - Load balancing
 - Fair share
- **Best device**
 - Parameterize the best device on the cluster for a given task

THANK YOU FOR LISTENING

Questions?