OpenCL in Scientific High Performance Computing

Matthias Noack
noack@zib.de

Zuse Institute Berlin
Distributed Algorithms and Supercomputing
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ...Why?
  - ...Why not?
  - ...What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ...Why?
  - ...Why not?
  - ...What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ... Why?
  - ... Why not?
  - ... What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ... Why?
  - ... Why not?
  - ... What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ... Why?
  - ... Why not?
  - ... What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ... Why?
  - ... Why not?
  - ... What are you using instead?
Audience Survey

- Who has a rough idea what OpenCL is?
- Who has hands-on experience with OpenCL?
- Who is using OpenCL in a real-world code?
  - ...Why?
  - ...Why not?
  - ...What are you using instead?
Context

ZIB hosts part of the HLRN (Northern German Supercomputing Alliance) facilities.
Context

ZIB hosts part of the HLRN (Northern German Supercomputing Alliance) facilities. Systems:

- 2× Cray XC40 (#118 and #150 in top500, year 4/5 in lifetime) with Xeon CPUs
- 80-node Xeon Phi (KNL) Cray XC40 test-and-development system
- 2× 32-node Infiniband cluster with Nvidia K40
- ZIB-internal test-and-development systems with:
  - Nvidia Pascal, AMD FirePro W8100, Cavium ThunderX (ARM), ...
Context

ZIB hosts part of the HLRN (Northern German Supercomputing Alliance) facilities. Systems:

- 2\times \text{Cray XC40} (#118 and #150 in top500, year 4/5 in lifetime) with \text{Xeon} CPUs
- 80-node \text{Xeon Phi} (KNL) \text{Cray XC40} test-and-development system
- 2\times 32-node Infiniband cluster with \text{Nvidia K40}
- ZIB-internal test-and-development systems with:
  - \text{Nvidia Pascal, AMD FirePro W8100, Cavium ThunderX (ARM)}, \ldots

What I do:

- computer science research (methods)
- development of HPC codes
- evaluation of upcoming technologies
- consulting/training with system users
The Situation with Scientific HPC

- tons of **legacy code** (FORTRAN) authored by domain experts
  - ⇒ rather closed community
  - ⇒ decoupled from computer science (ask a student about FORTRAN)
- highly **conservative** code owners
  - ⇒ modern software engineering advances are picked up very slowly
The Situation with Scientific HPC

- tons of **legacy code** (FORTRAN) authored by domain experts
  - ⇒ rather closed community
  - ⇒ decoupled from computer science (ask a student about FORTRAN)
- highly **conservative** code owners
  - ⇒ modern software engineering advances are picked up very slowly

- intra-node parallelism dominated by **OpenMP** (e.g. Intel) and **CUDA** (Nvidia)
  - ⇒ vendor and tool dependencies ⇒ **limited portability**
  - ⇒ multiple diverging code branches ⇒ **hard to maintain**
- inter-node communication = **MPI**
The Situation with Scientific HPC

- tons of **legacy code** (FORTRAN) authored by domain experts
  - rather closed community
  - decoupled from computer science (ask a student about FORTRAN)
- highly **conservative** code owners
  - modern software engineering advances are picked up very slowly

- intra-node parallelism dominated by **OpenMP** (e.g. Intel) and **CUDA** (Nvidia)
  - vendor and tool dependencies ⇒ **limited portability**
  - multiple diverging code branches ⇒ **hard to maintain**
- inter-node communication = **MPI**

- hardware life time: **5 years**
- software life time: **multiple tens of years**
  - outlives systems by far ⇒ **aim for portability**
Do not contribute to that situation!

What can we do better?

- put **portability** first (≠ performance portability)
  - ⇒ **OpenCL** has the largest **hardware coverage** for intra-node programming
    - CPUs, GPUs, accelerators, FPGAs, DSPs, …
  - ⇒ library only ⇒ **no tool dependencies**
Do not contribute to that situation!

What can we do better?

- put **portability** first (≠ performance portability)
  - ⇒ **OpenCL** has the largest **hardware coverage** for intra-node programming
    - CPUs, GPUs, accelerators, FPGAs, DSPs, . . .
  - ⇒ library only ⇒ **no tool dependencies**

- use modern techniques with a broad community (beyond HPC)
  - ⇒ e.g. **modern C++** for host code
  - ⇒ e.g. **CMake** for building
Do not contribute to that situation!

What can we do better?

- put **portability** first (≠ performance portability)
  - OpenCL has the largest **hardware coverage** for intra-node programming
    - CPUs, GPUs, accelerators, FPGAs, DSPs, ...
  - library only ⇒ **no tool dependencies**

- use modern techniques with a broad community (beyond HPC)
  - e.g. **modern C++** for host code
  - e.g. **CMake** for building

- develop code **interdisciplinary**
  - domain experts design the model . . .
  - . . . computer scientists the software
OpenCL (Open Computing Language) in a Nutshell

- open, royalty-free **standard** for cross-platform, **parallel programming**
- maintained by **Khronos Group**
- personal computers, servers, mobile devices and embedded platforms
- first released: **2009-08-28**
OpenCL (Open Computing Language) in a Nutshell

- open, royalty-free **standard** for cross-platform, **parallel programming**
- maintained by **Khronos Group**

- personal computers, servers, mobile devices and embedded platforms
- first released: 2009-08-28

[Image of various logos and a diagram showing OpenCL, SYCL, SPIR]
OpenCL Platform and Memory Model

- Compute Device (CD)
  - Device memory with global/constant address space
- Compute Unit (CU)
  - Local memory address space
- Processing Element (PE)
  - Private memory address space

⇒ Relaxed consistency model
⇒ Write code for this abstract machine model
⇒ Device-specific OpenCL compiler and runtime maps it to actual hardware
OpenCL Platform and Memory Model

Compute Device (CD)     Compute Unit (CU)

Processing Element (PE)

Memory Model:
- CD has device memory with global/constant addr. space
- CU has local memory addr. space
- PE has private memory add./space
⇒ relaxed consistency model
OpenCL Platform and Memory Model

Compute Device (CD)  Compute Unit (CU)

Processing Element (PE)

Memory Model:
- CD has device memory with global/constant addr. space
- CU has local memory addr. space
- PE has private memory add./space
  ⇒ relaxed consistency model

⇒ write code for this **abstract machine model**
⇒ device-specific OpenCL compiler and runtime maps it to **actual hardware**
## OpenCL Machine Model Mapping

<table>
<thead>
<tr>
<th>OpenCL Platform</th>
<th>CPU Hardware</th>
<th>GPU Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Device</td>
<td>Processor/Board</td>
<td>GPU device</td>
</tr>
<tr>
<td>Compute Unit</td>
<td>Core (thread)</td>
<td>Streaming MP</td>
</tr>
<tr>
<td>Processing Element</td>
<td>SIMD Lane</td>
<td>CUDA Core</td>
</tr>
<tr>
<td>global/const. memory</td>
<td>DRAM</td>
<td>DRAM</td>
</tr>
<tr>
<td>local memory</td>
<td>DRAM</td>
<td>Shared Memory</td>
</tr>
<tr>
<td>private memory</td>
<td>Register/DRAM</td>
<td>Register/DRAM</td>
</tr>
</tbody>
</table>
OpenCL Host Program and Kernel Execution

⇒ a host program uses OpenCL API-calls
⇒ kernels are written in OpenCL C/C++ kernel language
⇒ kernels are compiled at runtime for a specific device
⇒ kernels are executed on a range of work-items as work-groups of cooperating threads
OpenCL Host Program and Kernel Execution

⇒ a host program uses OpenCL API-calls
⇒ kernels are written in OpenCL C/C++ kernel language
⇒ kernels are compiled at runtime for a specific device
⇒ kernels are executed on a range of work-items as work-groups of cooperating threads
OpenCL Host Program and Kernel Execution

⇒ a **host program** uses OpenCL API-calls
⇒ **kernels** are written in OpenCL C/C++ kernel language
⇒ kernels are **compiled at runtime** for a specific device
⇒ kernels are **executed** on a range of **work-items** as **work-groups** of cooperating threads
OpenCL Host Program and Kernel Execution

⇒ a **host program** uses OpenCL API-calls
⇒ **kernels** are written in OpenCL C/C++ kernel language
⇒ kernels are **compiled at runtime** for a specific device
⇒ kernels are **executed** on a **range** of **work-items** as **work-groups** of cooperating threads
OpenCL Host Program and Kernel Execution

⇒ a **host program** uses OpenCL API-calls
⇒ **kernels** are written in OpenCL C/C++ kernel language
⇒ kernels are **compiled at runtime** for a specific device
⇒ kernels are **executed** on a range of **work-items** as **work-groups** of cooperating threads
## Selected Target Hardware:

<table>
<thead>
<tr>
<th>vendor</th>
<th>architecture</th>
<th>device</th>
<th>compute</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>Haswell</td>
<td>2× Xeon E5-2680v3</td>
<td>0.96 TFLOPS</td>
<td>136 GiB/s</td>
</tr>
<tr>
<td>Intel</td>
<td>Knights Landing</td>
<td>Xeon Phi 7250</td>
<td>2.61 TFLOPS*</td>
<td>490/115 GiB/s</td>
</tr>
<tr>
<td>AMD</td>
<td>Hawaii</td>
<td>Firepro W8100</td>
<td>2.1 TFLOPS</td>
<td>320 GiB/s</td>
</tr>
<tr>
<td>Nvidia</td>
<td>Kepler</td>
<td>Tesla K40</td>
<td>1.31 TFLOPS</td>
<td>480 GiB/s</td>
</tr>
</tbody>
</table>

*calculated with max. AVX frequency of 1.2 GHz: 2611.2 GFLOPS = 1.2 GHz × 68 cores × 8 SIMD × 2 VPUs × 2 FMA
COSIM - A Predictive Cometary Coma Simulation

Solve dust dynamics:

\[
\vec{a}_{\text{dust}}(\vec{r}) = \vec{a}_{\text{gas-drag}} + \vec{a}_{\text{grav}} + \vec{a}_{\text{Coriolis}} + \vec{a}_{\text{centrifugal}}
\]

\[
= \frac{1}{2} C_d \alpha N_{\text{gas}}(\vec{r}) m_{\text{gas}} (\vec{v}_{\text{dust}} - \vec{v}_{\text{gas}}) |\vec{v}_{\text{dust}} - \vec{v}_{\text{gas}}| - \nabla \phi(\vec{r}) - 2 \vec{\omega} \times \vec{v}_{\text{dust}} - \vec{\omega} \times (\vec{\omega} \times \vec{r})
\]

Compare with data of 67P/Churyumov–Gerasimenko from Rosetta spacecraft:

Panels 1-2: OSIRIS NAC Image, Panels 3-4: Simulation Results, Right Image: ESA – C. Carreau/ATG medialab, CC BY-SA 3.0-igo
HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

- understand the energy transfer in photo-active molecular complexes
  ⇒ e.g. photosynthesis

**HEOM - The Hierarchical Equations of Motion**

**Model for Open Quantum Systems**

- understand the energy transfer in photo-active molecular complexes
  - e.g. photosynthesis
- millions of coupled ODEs

\[
\frac{d\sigma_u}{dt} = \frac{i}{\hbar} [H, \sigma_u] \\
- \sigma_u \sum_{b=1}^{B} \sum_{k=1}^{K-1} n_{u,(b,k)} \gamma(b,k) \\
- \frac{B}{\beta \hbar^2 \nu_b} - \sum_{k=1}^{K-1} \frac{c(b,k)}{\hbar \gamma(b,k)} V_{s(b)} V_{s(b)} \sigma_u \\
+ \sum_{b=1}^{B} \sum_{k=1}^{K-1} i V_{s(b)}^{\times} \sigma^+_{u,b,k} \\
+ \sum_{b=1}^{B} \sum_{k=1}^{K-1} n_{u,(b,k)} \theta_{MA(b,k)} \sigma^-_{(u,b,k)}
\]

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

- understand the energy transfer in photo-active molecular complexes ⇒ e.g. photosynthesis
- millions of coupled ODEs
- hierarchical graph of matrices

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

- understand the energy transfer in photo-active molecular complexes
  ⇒ e.g. photosynthesis

- millions of coupled ODEs

- hierarchical graph of matrices

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

- understand the energy transfer in photo-active molecular complexes
  ⇒ e.g. photosynthesis
- millions of coupled ODEs
- hierarchical graph of matrices

Interdisciplinary Workflow

Mathematical Model

domain experts

computer scientists
Interdisciplinary Workflow

- ODEs
- PDEs
- Graphs
- ...
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)

domain experts  computer scientists
Interdisciplinary Workflow

Mathematical Model

- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)

- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL

domain experts  computer scientists
Mathematica and OpenCL

(* Load OpenCL support *)
Needs["OpenCLLink"]

(* Create OpenCLFunction from source, kernel name, signature *)
doubleFun = OpenCLFunctionLoad["__kernel void doubleVec(__global mint * in, mint length) {
    int index = get_global_id(0);
    if (index < length)
        in[index] = 2*in[index];
}", "doubleVec", {{_Integer}, _Integer}, 256]

(* Create some input *)
vec = Range[20];

(* Call the function *)
doubleFun[vec, 20] (* NDRange deduced from args and wg-size *)

Mathematica and OpenCL

(* Load OpenCL support *)
Needs["OpenCLLink"]

(* Create OpenCLFunction from source, kernel name, signature *)
doubleFun = OpenCLFunctionLoad[
__kernel void doubleVec(__global mint * in, mint length) {
  int index = get_global_id(0);
  if (index < length)
    in[index] = 2*in[index];
}", "doubleVec", {{_Integer}, _Integer}, 256]

(* Create some input *)
vec = Range[20];

(* Call the function *)
doubleFun[vec, 20] (* NDRange deduced from args and wg-size *)
Mathematica and OpenCL

(* Load OpenCL support *)
Needs["OpenCLLink"]

(* Create OpenCLFunction from source, kernel name, signature *)
doubleFun = OpenCLFunctionLoad[
  "__kernel void doubleVec(__global mint * in, mint length) {
  int index = get_global_id(0);
  if (index < length)
    in[index] = 2*in[index];
  }",
  "doubleVec", {{_Integer}, _Integer}, 256]

(* Create some input *)
vec = Range[20];

(* Call the function *)
doubleFun[vec, 20] (* NDRange deduced from args and wg-size *)

Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM
Interdisciplinary Workflow

- **Mathematical Model**
  - ODEs
  - PDEs
  - Graphs
  - ...

- **High-Level Prototype (Mathematica)**
  - domain scientist’s tool
  - high level
  - symbolic solvers
  - arbitrary precision
  - very limited performance

- **OpenCL kernel within Mathematica**
  - replace some code with OpenCL
  - compare results
  - figure out numerics
  - use accelerators in MM

... → **C++ Host Application**
Interdisciplinary Workflow

- **Mathematical Model**
  - ODEs
  - PDEs
  - Graphs
  - ...

- **High-Level Prototype (Mathematica)**
  - domain scientist’s tool
  - high level
  - symbolic solvers
  - arbitrary precision
  - very limited performance

- **OpenCL kernel within Mathematica**
  - replace some code with OpenCL
  - compare results
  - figure out numerics
  - use accelerators in MM

- **C++ Host Application**
  - start single node
  - OpenCL 1.2 for hotspots
  - modern C++ 11/14/17
  - CMake for building
<table>
<thead>
<tr>
<th>name</th>
<th>version</th>
<th>OpenCL version</th>
<th>supported devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel OpenCL SDK</td>
<td>2017 R1</td>
<td>CPU: 1.2 (exp. 2.1), GPU: 2.1</td>
<td>CPUs (≤ AVX2), Intel GPUs</td>
</tr>
<tr>
<td>Intel OpenCL SDK</td>
<td>14.2</td>
<td>1.2</td>
<td>Xeon Phi (KNC)</td>
</tr>
<tr>
<td>Nvidia OpenCL</td>
<td>CUDA 8</td>
<td>1.2 (exp. 2.0)</td>
<td>Nvidia GPU</td>
</tr>
<tr>
<td>AMD APP SDK</td>
<td>3.0</td>
<td>2.0 (GPU), 1.2 (CPU)</td>
<td>GPU, CPUs (AVX,FMA4,XOP)</td>
</tr>
<tr>
<td>PoCL</td>
<td>0.14</td>
<td>2.0</td>
<td>CPUs (AVX-512), GPUs, ARM</td>
</tr>
</tbody>
</table>

Vendors seem not to be too enthusiastic about OpenCL:
- portable OpenCL still means version 1.2 (released Nov. 2011)
- partial OpenCL 2.0 support by Nvidia introduced rather silently
- Xeon Phi implementation discontinued by Intel, no AVX-512 support (yet?)
## OpenCL SDKs and Versions

<table>
<thead>
<tr>
<th>name</th>
<th>version</th>
<th>OpenCL version</th>
<th>supported devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel OpenCL SDK</td>
<td>2017 R1</td>
<td>CPU: 1.2 (exp. 2.1), GPU: 2.1</td>
<td>CPUs (≤ AVX2), Intel GPUs</td>
</tr>
<tr>
<td>Intel OpenCL SDK</td>
<td>14.2</td>
<td>1.2</td>
<td>Xeon Phi (KNC)</td>
</tr>
<tr>
<td>Nvidia OpenCL</td>
<td>CUDA 8</td>
<td>1.2 (exp. 2.0)</td>
<td>Nvidia GPU</td>
</tr>
<tr>
<td>AMD APP SDK</td>
<td>3.0</td>
<td>2.0 (GPU), 1.2 (CPU)</td>
<td>GPU, CPUs (AVX,FMA4,XOP)</td>
</tr>
<tr>
<td>PoCL</td>
<td>0.14</td>
<td>2.0</td>
<td>CPUs (AVX-512), GPUs, ARM</td>
</tr>
</tbody>
</table>

Vendors seem not to be too enthusiastic about OpenCL:

- portable OpenCL still means version 1.2 (released Nov. 2011)
- partial OpenCL 2.0 support by Nvidia introduced rather silently
- Xeon Phi implementation discontinued by Intel, no AVX-512 support (yet?)
The OpenCL Installable Client Driver (ICD) Loader

- allows multiple OpenCL installations to be installed and used next to each other
The OpenCL Installable Client Driver (ICD) Loader

- allows multiple OpenCL installations to be installed and used next to each other

```
/  
__etc/OpenCL/vendors/  
  __intel64.icd.........textfile with actual library name/path
  __amdocl64.icd .......textfile with actual library name/path

__usr/lib/  
  __libOpenCL.so ........ loader looking into /etc/OpenCL/vendors

__opt/intel_ocl/lib/  
  __libintelocl.so ........ actual Intel implementation

__opt/AMDAPPSDK/lib/  
  __libamdocl64.so .........actual AMD implementation
```
The OpenCL Installable Client Driver (ICD) Loader

- allows multiple OpenCL installations to be installed and used next to each other

```
/  
   __ etc/OpenCL/vendors/  
      __ intel64.icd ........... textfile with actual library name/path
      __ amdocl64.icd ........ textfile with actual library name/path
   __ usr/lib/
      __ libOpenCL.so ........ loader looking into /etc/OpenCL/vendors
   __ opt/intel_ocl/lib/
      __ libintelocl.so ........ actual Intel implementation
   __ opt/AMDAPPSDK/lib/
      __ libamdocl64.so ........ actual AMD implementation
```

- applications typically link to the loader, direct link is also possible
- only some loaders support OPENCL_VENDOR_PATH env. variable
  ⇒ problematic for user-installation (modifying /etc/ requires root)
Platform and Device Selection

- OpenCL API: *lots of device properties* can be queried
- simple and pragmatic: `oclinfo` tool ⇒ *platform/device index*

---

https://github.com/noma/ocl
### Platform and Device Selection

- **OpenCL API**: lots of device properties can be queried
- simple and pragmatic: oclinfo tool ⇒ platform/device index

<table>
<thead>
<tr>
<th>Platform 0:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME: AMD Accelerated Parallel Processing</td>
</tr>
<tr>
<td>VERSION: OpenCL 2.0 AMD-APP (1912.5)</td>
</tr>
<tr>
<td>Device 0:</td>
</tr>
<tr>
<td>NAME: Hawaii</td>
</tr>
<tr>
<td>VENDOR: Advanced Micro Devices, Inc.</td>
</tr>
<tr>
<td>VERSION: OpenCL 2.0 AMD-APP (1912.5)</td>
</tr>
<tr>
<td>Device 1:</td>
</tr>
<tr>
<td>NAME: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz</td>
</tr>
<tr>
<td>VENDOR: GenuineIntel</td>
</tr>
<tr>
<td>VERSION: OpenCL 1.2 AMD-APP (1912.5)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME: Intel(R) OpenCL</td>
</tr>
<tr>
<td>VERSION: OpenCL 1.2 LINUX</td>
</tr>
<tr>
<td>Device 0:</td>
</tr>
<tr>
<td>NAME: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz</td>
</tr>
<tr>
<td>VENDOR: Intel(R) Corporation</td>
</tr>
<tr>
<td>VERSION: OpenCL 1.2 (Build 57)</td>
</tr>
</tbody>
</table>

[https://github.com/noma/ocl]
Compilation

OpenCL Header Files:

⇒ avoid trouble: use reference headers, ship with project
  ⇒ https://github.com/KhronosGroup/OpenCL-Headers

CMake:

find_package(OpenCL REQUIRED)

• OpenCL CMake module only works in some scenarios
  ⇒ the magic line:

  mkdir build .
  cd build .
  cmake -DCMAKE_BUILD_TYPE = Release -DOpenCL_FOUND= True -DOpenCL_INCLUDE_DIR=../../thirdparty/include -DOpenCL_LIBRARY=/opt/intel/opencl_runtime_16.1.1/opt/intel/opencl-1.2-6.4.0.25/lib64/libIntelocl.so

  make -j
Compilation

OpenCL Header Files:

⇒ avoid trouble: use reference headers, ship with project

⇒ https://github.com/KhronosGroup/OpenCL-Headers

CMake: find_package(OpenCL REQUIRED)

• OpenCL CMake module only works in some scenarios

⇒ the magic line:

```bash
mkdir build.intel_16.1.1
cd build.intel_16.1.1

cmake -DCMAKE_BUILD_TYPE=Release -DOpenCL_FOUND=True -DOpenCL_INCLUDE_DIR=../ ../thirdparty/include/ -DOpenCL_LIBRARY=/opt/intel/opencl_runtime_16.1.1/opt/intel/opencl-1.2-6.4.0.25/lib64/libintelocl.so ..

make -j
```
Handling Kernel Source Code

a) loading source files at runtime:
   ✓ no host-code recompilation
   ✓ #include directives

b) embedded source as string constant:
   ✓ self-contained executable for production use
Handling Kernel Source Code

a) loading source files at runtime:
   ✓ no host-code recompilation
   ✓ #include directives

b) embedded source as string constant:
   ✓ self-contained executable for production use

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading **source files** at runtime:
   ✓ no host-code recompilation
   ✓ `#include` directives

b) embedded source as **string constant**:
   ✓ self-contained executable for production use

---

`header.cl`

`#include`

`kernel_source.cl`

`kernel_source.cpp/hpp`

`resolve_includes.sh`

---

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading source files at runtime:
   ✓ no host-code recompilation
   ✓ #include directives

b) embedded source as string constant:
   ✓ self-contained executable for production use

- create raw string literal
  R"str_not_in_src(
   // input
    )str_not_in_src"

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading **source files** at runtime:
   - ✓ no host-code recompilation
   - ✓ `#include` directives

b) embedded source as **string constant**:
   - ✓ self-contained executable for production use

- create raw string literal
  
  ```
  R"str_not_in_src(
  // input
  )str_not_in_src"
  ```

```
cornell_source.cl
```

```
header.cl
```

```
resolve_includes.sh
```

```
cl_to_hpp.sh
```

```
kernel_source.hpp
```

```
kernel_wrapper_class.hpp/cpp
```

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading **source files** at runtime:
   ✓ no host-code recompilation
   ✓ #include directives

   - create raw string literal
     
     ```
     R"str_not_in_src(
       // input
     )str_not_in_src"
     ```

b) embedded source as **string constant**:
   ✓ **self-contained** executable for production use

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading source files at runtime:
   ✓ no host-code recompilation
   ✓ #include directives

b) embedded source as string constant:
   ✓ self-contained executable for production use

- create raw string literal
  \[ R"str_not_in_src( // input )str_not_in_src" \]

CMake dependency

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading source files at runtime:
   - ✓ no host-code recompilation
   - ✓ `#include` directives

b) embedded source as string constant:
   - ✓ self-contained executable for production use

- create raw string literal
  
  ```
  R"str_not_in_src( 
    // input 
  )str_not_in_src"
  ```

- `#include` resolve_includes.sh
- `cl_to_hpp.sh` generates
  
  ```c
  kernel_source_hpp
  ```

- `#include` kernel_wrapper_class_hpp/.cpp

https://github.com/noma/ocl
Handling Kernel Source Code

a) loading **source files** at runtime:
   ✓ no host-code recompilation
   ✓ `#include` directives

b) embedded source as **string constant**:
   ✓ *self-contained* executable for production use

- create raw string literal
  
  ```
  R"str_not_in_src(
    // input
  )str_not_in_src"
  ```

---

**Diagram:**

- **header .cl**
- **kernel_source .cl**
- **#include**
- **resolve_includes.sh**
- **cl_to_hpp.sh**
- **kernel_source.hpp**
- **kernel_wrapper_class.hpp/.cpp**
- **CMake dependency**
- generates
- load file at runtime (via alternative ctor)

**Sources:**

- `https://github.com/noma/ocl`
Example OpenCL Runtime Configuration File

[opencl]
# use first device of second platform
platform_index=1
device_index=0

# enable zero copy buffers for CPU devices
zero_copy_device_types={cpu}

# pass a custom include path to the OpenCL compiler
compile_options=-I../cl

# load kernel source from file at runtime
kernel_file_heom_ode=../cl/heom_ode.cl
kernel_name_heom_ode=heom_ode

# unset option, load embedded source
#kernel_file_rk_weighted_add=
#kernel_name_rk_weighted_add=

https://github.com/noma/ocl
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

C++ Host Application
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

C++ Host Application
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

Distributed Host Application
Interdisciplinary Workflow

**Mathematical Model**
- ODEs
- PDEs
- Graphs
- ...

**High-Level Prototype (Mathematica)**
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

**OpenCL kernel within Mathematica**
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

**C++ Host Application**
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

**Distributed Host Application**
- scale to multiple nodes
- partitioning, balancing,
  neighbour exchange, ...
- wrapped MPI 3.0
OpenCL and Communication/MPI

Design Recommendation:

- keep both aspects as independent as possible
- design code to be agnostic to whether it works on a complete problem instance or on a partition
- implement hooks for communication between kernel calls
- wrap needed part of MPI in a thin, exchangeable abstraction layer

Current trade-offs:

- communication introduces additional logical host-device transfers
  ⇒ scaling starts slowly, e.g. two nodes might be slower than one
- a single process might not be able to saturate the network
  ⇒ multiple processes per node sharing a device (CPU device: set CPU mask)
- pick one: zero-copy buffers or overlapping compute and communication
  ⇒ either host (comm.) or device (comp.) own the memory at any point in time
  ⇒ overlapping requires copies
OpenCL and Communication/MPI

Design Recommendation:

- keep both aspects as independent as possible
- design code to be agnostic to whether it works on a complete problem instance or on a partition
- implement hooks for communication between kernel calls
- wrap needed part of MPI in a thin, exchangeable abstraction layer

Current trade-offs:

- communication introduces additional logical host-device transfers
  ⇒ scaling starts slowly, e.g. two nodes might be slower than one
- a single process might not be able to saturate the network
  ⇒ multiple processes per node sharing a device (CPU device: set CPU mask)
- pick one: zero-copy buffers or overlapping compute and communication
  ⇒ either host (comm.) or device (comp.) own the memory at any point in time
  ⇒ overlapping requires copies again
Data Transfer Paths

device

OpenCL driver

device memory

DMA

pinned device buffer

host

application code

fabric

driver

fabric

driver

application code

OpenCL driver

device memory

host memory

mem cpy

host memory

mem cpy

host memory

mem cpy

pinned device buffer

pinned fabric buffer

RDMA

pinned fabric buffer

pinned device buffer

OpenCL

driver

application code

fabric

driver

device memory

OpenCL

driver

application code

fabric

driver

device memory

OpenCL

driver
Data Transfer Paths

can be avoided in some cases with OpenCL
Data Transfer Paths

- **device memory**
- **pinned device buffer**
- **host memory**
- **fabric buffer**
- **DMA**
- **RDMA**

**CUDA GPU-Direct RDMA**

- can be avoided in some cases with OpenCL

**OpenCL driver**
**application code**
**fabric driver**
**application code**
**OpenCL driver**

**device**
**host**
Benchmark Results: COSIM load imbalance (Xeon)
Benchmark Results: COSIM load imbalance (Xeon)

COSIM Runtime vs. Particle Count (2x Xeon, Haswell)

⇒ every 32 particles
Benchmark Results: COSIM load imbalance (Xeon)

COSIM Runtime vs. Particle Count (2x Xeon, Haswell)

- Intel OpenCL SDK
- AMD APP SDK
- PoCL

⇒ every 32 particles
Benchmark Results: COSIM load imbalance (Xeon)

COSIM Runtime vs. Particle Count (2x Xeon, Haswell)

- Intel OpenCL SDK
- AMD APP SDK
- PoCL

⇒ every 32 particles
⇒ 384 workitems = 16 × 24 cores
Benchmark Results: COSIM load imbalance (Xeon Phi)
Benchmark Results: COSIM load imbalance (Xeon Phi)

No PoCL data for multiples of 32 in this range
Benchmark Results: COSIM load imbalance (Xeon Phi)
Benchmark Results: COSIM node imbalance, all
Benchmark Results: COSIM node imbalance, all

COSIM Runtime vs. Particle Count (all)

- Intel OpenCL SDK, 2x Xeon
- AMD APP SDK, 2x Xeon
- PoCL, 2x Xeon
- Intel OpenCL SDK, Xeon Phi
- AMD APP SDK, Xeon Phi
- PoCL, Xeon Phi

runtime per iteration [s]

particles per compute node
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

C++ Host Application
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

Distributed Host Application
- scale to multiple nodes
- partitioning, balancing, neighbour exchange, ...
- wrapped MPI 3.0
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

C++ Host Application
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

Distributed Host Application
- scale to multiple nodes
- partitioning, balancing,
  neighbour exchange, ...

Optimisation / Production Runs
- wrapped MPI 3.0
Interdisciplinary Workflow

Mathematical Model
- ODEs
- PDEs
- Graphs
- ...

High-Level Prototype (Mathematica)
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

OpenCL kernel within Mathematica
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

C++ Host Application
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

Distributed Host Application
- scale to multiple nodes
- partitioning, balancing, neighbour exchange, ...
- wrapped MPI 3.0

Optimisation / Production Runs
- always collect perf. data
- profile/tune code
- add performance tweaks
- use device-specific kernel variants if needed
HEOM Benchmark Results: CPU SDK comparison

OpenCL CPU SDK Comparison on 2x Xeon (HSW)

- **fmo_22baths_d3.cfg**
  - Intel: 29 ms
  - AMD: 35 ms
  - PoCL: 30 ms

- **lhci_1bath_d8.cfg**
  - Intel: 29 ms
  - AMD: 35 ms
  - PoCL: 30 ms

Average kernel runtime [ms] for OpenCL SDKs by Intel, AMD, and PoCL.
HEOM Benchmark Results: CPU SDK comparison

OpenCL CPU SDK Comparison on 2x Xeon (HSW)

OpenCL CPU SDK Comparison on Xeon Phi (KNL)
HEOM Benchmarks: Workitem Granularity on CPUs

Impact of Work-Item Granularity on 2x Xeon (HSW)

Granularity:
- Matrix
- Element
HEOM Benchmarks: Workitem Granularity on CPUs

Impact of Work-Item Granularity on 2x Xeon (HSW)

Impact of Work-Item Granularity on Xeon Phi (KNL)
HEOM Benchmarks: Workitem Granularity on GPUs

Impact of Work-Item Granularity on Tesla K40

average kernel runtime [ms]

Granularity:
Matrix
Element

Matrix Element

average kernel runtime [ms]

Granularity:
Matrix
Element

Matrix Element

Matrix Element
HEOM Benchmarks: Workitem Granularity on GPUs

Impact of Work-Item Granularity on Tesla K40

Impact of Work-Item Granularity on FirePro W8100
HEOM Benchmarks: Performance Portability

Runtime Comparison on Different Hardware

- fmo_22baths_d3.cfg
- lhci_1bath_d8.cfg

Hardware:
- 2x Xeon (HSW)
- Xeon Phi (KNL)
- Tesla K40
- FirePro W8100

Average kernel runtime [ms]
HEOM Benchmarks: Performance Portability

**Runtime Comparison on Different Hardware**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>fmo_22baths_d3.cfg</th>
<th>lhci_1bath_d8.cfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x Xeon (HSW)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xeon Phi (KNL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tesla K40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FirePro W8100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Performance Portability Relative to Xeon**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>fmo_22baths_d3.cfg</th>
<th>lhci_1bath_d8.cfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x Xeon (HSW)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xeon Phi (KNL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from FLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from GiB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tesla K40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from FLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from GiB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FirePro W8100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from FLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expected from GiB/s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 / 35
Interdisciplinary Workflow

**Mathematical Model**
- ODEs
- PDEs
- Graphs
- ... 

**High-Level Prototype (Mathematica)**
- domain scientist’s tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

**OpenCL kernel within Mathematica**
- replace some code with OpenCL
- compare results
- figure out numerics
- use accelerators in MM

**C++ Host Application**
- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- CMake for building

**Distributed Host Application**
- scale to multiple nodes
- partitioning, balancing, neighbour exchange, ...
- wrapped MPI 3.0

**Optimisation / Production Runs**
- always collect perf. data
- profile/tune code
- add performance tweaks
- use device-specific kernel variants if needed
Conclusion

General

- work interdisciplinary
- put portability first

Wishlist

- @Intel: AVX-512 / Xeon Phi support would be highly appreciated
- @vendors: keep up with the standard
- @Khronos Group: take communication/computation scenarios into account
Conclusion

General
- work interdisciplinary
- put portability first

OpenCL
- **highest portability** of available parallel programming models
- integrates well into interdisciplinary workflow
- **runtime compilation** allows compiler-optimisation with runtime-constants
- performance portability is not for free, but ...
  ⇒ ...**better two kernels than two programming models**
Conclusion

General
- work interdisciplinary
- put portability first

OpenCL
- **highest portability** of available parallel programming models
- integrates will into interdisciplinary workflow
- **runtime compilation** allows compiler-optimisation with runtime-constants
- performance portability is not for free, but ...
  \[ \Rightarrow \ldots \text{better two kernels than two programming models} \]

Wishlist
- **@Intel**: AVX-512 / Xeon Phi support would be highly appreciated
- **@vendors**: keep up with the standard
- **@Khronos Group**: take communication/computation scenarios into account
Thank you.

Feedback? Questions? Ideas?

noack@zib.de

The author would like to thank the domain experts from the HEOM and COSIM teams for the many fruitful discussions on the OpenCL user-experience and reported struggles with the different implementations and target systems. This project was supported by the German Research Foundation (DFG) project RE 1389/8, and the North-German Supercomputing Alliance (HLRN).