OpenCL in Scientific High Performance Computing—The Good, the Bad, and the Ugly

Matthias Noack
noack@zib.de

Zuse Institute Berlin
Distributed Algorithms and Supercomputing
Context

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

- $2 \times$ 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
- $2 \times$ 32-node Infiniband cluster with Nvidia K40
- 2 test-and-development systems with AMD FirePro W8100
Context

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

- $2 \times$ 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
- $2 \times$ 32-node Infiniband cluster with Nvidia K40
- 2 test-and-development systems with AMD FirePro W8100

What I do:

- computer science research (methods)
- development of HPC codes
- evaluation of upcoming technologies
- consulting/training with system users
Why OpenCL? (aka: *The Good*)

Scientific HPC in a Nutshell

- 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
Why OpenCL? (aka: *The Good*)

Scientific HPC in a Nutshell

- 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
Why OpenCL? (aka: The Good)

Scientific HPC in a Nutshell

- 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**
Why OpenCL? (aka: The Good)

Goal for new code: Do not contribute to that situation!

- **portability** first (≠ performance portability)
  - ⇒ **OpenCL** has the largest **hardware coverage** for intra-node programming
  - ⇒ library only ⇒ **no tool dependencies**
Why OpenCL? (aka: *The Good*)

Goal for new code: Do not contribute to that situation!

- **portability** first (≠ performance portability)
  - ⇒ **OpenCL** has the largest **hardware coverage** for intra-node programming
  - ⇒ library only ⇒ **no tool dependencies**

- use modern techniques with a broad community (beyond HPC)
  - ⇒ **modern C++** for host code
Why OpenCL? (aka: *The Good*)

Goal for new code: Do not contribute to that situation!

- **portability** first (≠ performance portability)
  - ⇒ OpenCL has the largest *hardware coverage* for intra-node programming
  - ⇒ library only ⇒ **no tool dependencies**

- use modern techniques with a broad community (beyond HPC)
  - ⇒ **modern C++** for host code

- develop code **interdisciplinary**
  - ⇒ domain experts design the model . . .
  - ⇒ . . . computer scientists the software
## 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

Model for Open Quantum Systems

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

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

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

- understand the energy transfer in photo-active molecular complexes
  \[ \Rightarrow \text{e.g. photosynthesis} \]

- 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

Mathematical Model

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

- ODEs
- PDEs
- Graphs
- ...

Mathematical Model → High Level Prototype (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
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 *)

NDRange can be larger than length

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

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

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
- CMake for building
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>16.1.1</td>
<td>1.2 (CPU), 2.1 (GPU)</td>
<td>CPUs (up to 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 (LLVM, AVX-512)</td>
</tr>
</tbody>
</table>

Vendors seem not to be too enthusiastic about OpenCL:

• portable OpenCL still means version 1.2 (released Nov. 2011)
• Xeon Phi implementation discontinued by Intel, no AVX-512 support (yet?)
• partial OpenCL 2.0 support by Nvidia introduced rather silently
### 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>16.1.1</td>
<td>1.2 (CPU), 2.1 (GPU)</td>
<td>CPUs (up to 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 CUDA 8</td>
<td></td>
<td>1.2 (exp. 2.0)</td>
<td>Nvidia GPU</td>
</tr>
<tr>
<td>AMD APP SDK 3.0</td>
<td></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 (LLVM, AVX-512)</td>
</tr>
</tbody>
</table>

Vendors seem not to be too enthusiastic about OpenCL:

- portable OpenCL still means version 1.2 (released Nov. 2011)
- Xeon Phi implementation discontinued by Intel, no AVX-512 support (yet?)
- partial OpenCL 2.0 support by Nvidia introduced rather silently
Installation/Linking/Running

Platform and Device selection:
⇒ simple, deterministic way: oclinfo tool ⇒ platform/device index

ICD loader mechanism (libOpenCL.so):
• OpenCL typically not pre-installed in HPC environments
• adding ICD files to /etc/OpenCL/ requires root
⇒ not all loaders support OPENCL_VENDOR_PATH environment variable
• different ICDs report different platform/device order
• different order from different API paths (with or without context creation)
• SDK installation order matters
⇒ use reference ICD
⇒ avoid ICD and link directly
• libs in /etc/OpenCL/vendors/*.icd
• libamdocl64.so, libintelocl.so, ...
Installation/Linking/Running

Platform and Device selection:
⇒ simple, deterministic way: oclinfo tool ⇒ platform/device index

ICD loader mechanism (libOpenCL.so):

- OpenCL typically not pre-installed in HPC environments
- adding ICD files to /etc/OpenCL/ requires root
  ⇒ not all loaders support OPENCL_VENDOR_PATH environment variable
- different ICDs report different platform/device order
  - different order from different API paths (with or without context creation)
  - SDK installation order matters
⇒ use reference ICD
⇒ avoid ICD and link directly
  - libs in /etc/OpenCL/vendors/*.icd
  - libamdocl64.so, libintelocl.so, ...
Compilation

OpenCL Header Files:
⇒ avoid trouble: use reference headers, ship with project

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
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

#include resolve_includes.sh

cl_to_hpp.sh
```

```
R"str_not_in_src(  // input
)str_not_in_src"

kernel_source.cl

kernel_source.hpp

kernel_wrapper_class.hpp/.cpp
```
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"
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
  \[
  \text{R"str_not_in_src(}
  \quad // \text{ input}
  \quad \text{)}str_not_in_src"}
  \]
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"

header .cl

#include

kernel_source .cl

resolve_includes.sh

cl_to_hpp.sh

kernel_source .hpp

#include

kernel_wrapper_class .hpp/.cpp
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
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

#include
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`
  - `#include`
- `kernel_source .cl`
  - `resolve_includes.sh`
  - `cl_to_hpp.sh`
  - `kernel_source .hpp`
    - `#include`
    - `kernel_wrapper_class .hpp/.cpp`

**CMake dependency**

- `load file at runtime (via alternative ctor)`
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
- CMake for building
Interdisciplinary Workflow

- domain experts
- computer scientists

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
- 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
- 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

Trade-offs:

- communication introduces local 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
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

Trade-offs:

- communication introduces local **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 Memory**
  - OpenCL driver
  - DMA
  - Pinned device buffer
  - Mem cpy
  - Host memory
  - Mem cpy
  - Pinned fabric buffer
  - RDMA
  - Pinned device buffer
  - Mem cpy
  - Host memory
  - Mem cpy
  - Pinned device buffer

- **Host Memory**
  - Application code
  - Fabric driver
  - Mem cpy
  - Host memory
  - Mem cpy
  - Fabric driver
  - Mem cpy
  - Host memory
  - Mem cpy
  - Pinned device buffer

- **OpenCL Driver**
  - Application code
  - Fabric driver
  - Mem cpy
  - Host memory
  - Mem cpy
  - Fabric driver
  - Mem cpy
  - Host memory
  - Mem cpy
  - Pinned device buffer
Data Transfer Paths

- **Device Memory**
- **Pinned Device Buffer**
- **Host Memory**
- **Pinned Fabric Buffer**
- **RDMA**
- **DMA**

The diagram illustrates data transfer paths between devices and memory spaces, highlighting how data can be transferred more efficiently with OpenCL, potentially avoiding certain transfer paths in some cases.
Data Transfer Paths

CUDA GPU-Direct RDMA can be avoided in some cases with OpenCL
Benchmark Results: COSIM load imbalance (Xeon)
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
Benchmark Results: COSIM load imbalance (Xeon)

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

⇒ 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)

No PoCL data for multiples of 32 in this range
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

particles per compute node vs. runtime per iteration [s]
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

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
- 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
- CMake for building

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

Optimisation / Production Runs
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
- 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)

average kernel runtime [ms]

<table>
<thead>
<tr>
<th>File</th>
<th>Intel</th>
<th>AMD</th>
<th>PoCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmo_22baths_d3.cfg</td>
<td>220</td>
<td>300</td>
<td>280</td>
</tr>
<tr>
<td>lhci1_1bath_d8.cfg</td>
<td>220</td>
<td>300</td>
<td>280</td>
</tr>
</tbody>
</table>
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)
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

- fmo_22baths_d3.cfg
- lhci_1bath_d8.cfg

Granularity:
- Matrix
- Element

average kernel runtime [ms]

Matrix Element Matrix Element
HEOM Benchmarks: Workitem Granularity on GPUs

### Impact of Work-Item Granularity on Tesla K40

<table>
<thead>
<tr>
<th>Granularity:</th>
<th>fmo_22baths_d3.cfg</th>
<th>lhci_1bath_d8.cfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td><img src="chart1.png" alt="Matrix Bar Chart" /></td>
<td><img src="chart2.png" alt="Matrix Bar Chart" /></td>
</tr>
<tr>
<td>Element</td>
<td><img src="chart1.png" alt="Element Bar Chart" /></td>
<td><img src="chart2.png" alt="Element Bar Chart" /></td>
</tr>
</tbody>
</table>

### Impact of Work-Item Granularity on FirePro W8100

<table>
<thead>
<tr>
<th>Granularity:</th>
<th>fmo_22baths_d3.cfg</th>
<th>lhci_1bath_d8.cfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td><img src="chart3.png" alt="Matrix Bar Chart" /></td>
<td><img src="chart4.png" alt="Matrix Bar Chart" /></td>
</tr>
<tr>
<td>Element</td>
<td><img src="chart3.png" alt="Element Bar Chart" /></td>
<td><img src="chart4.png" alt="Element Bar Chart" /></td>
</tr>
</tbody>
</table>
HEOM Benchmarks: Performance Portability

Runtime Comparison on Different Hardware

<table>
<thead>
<tr>
<th>Hardware</th>
<th>fmo_22baths_d3.cfg</th>
<th>lhcii_1bath_d8.cfg</th>
</tr>
</thead>
</table>
| 2x Xeon (HSW) | ![HSW](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIUAAADkCAYAAAAQYq5AAAAglbGQAACGh0dHBzaW9uIHdoeXBlbyBvZiB3aXRoIGFzdG8gYmVzdCBvZiByZWZ0...)
| Xeon Phi (KNL)  | ![KNL](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIUAAADkCAYAAAAQYq5AAAAglbGQAACGh0dHBzaW9uIHdoeXBlbyBvZiB3aXRoIGFzdG8gYmVzdCBvZiByZWZ0...)
| Tesla K40      | ![K40](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIUAAADkCAYAAAAQYq5AAAAglbGQAACGh0dHBzaW9uIHdoeXBlbyBvZiB3aXRoIGFzdG8gYmVzdCBvZiByZWZ0...)
| FirePro W8100  | ![W8100](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIUAAADkCAYAAAAQYq5AAAAglbGQAACGh0dHBzaW9uIHdoeXBlbyBvZiB3aXRoIGFzdG8gYmVzdCBvZiByZWZ0...)

Hardware

- 2x Xeon (HSW)
- Xeon Phi (KNL)
- Tesla K40
- FirePro W8100
HEOM Benchmarks: Performance Portability

Runtime Comparison on Different Hardware

Performance Portability Relative to Xeon
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
- 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

OpenCL

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

OpenCL

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

HPC Wishlist

- zero-copy buffers with shared ownership
- equivalents to CUDA’s GPU-Direct and CUDA-aware MPI
- no way to specify memory alignment beyond data type size of a kernel parameter
- @vendors: please keep up with the standard
- @Intel: AVX-512 / Xeon Phi support would be highly appreciated
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).

https://doi.org/10.1145/3078155.3078170