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

https://doi.org/10.1145/3078155.3078170

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**
- 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
  - $\Rightarrow$ rather closed community
  - $\Rightarrow$ decoupled from computer science (ask a student about FORTRAN)
- highly **conservative** code owners
  - $\Rightarrow$ modern software engineering advances are picked up very slowly
- intra-node parallelism dominated by **OpenMP** (e.g. Intel) and **CUDA** (Nvidia)
  - $\Rightarrow$ vendor and tool dependencies $\Rightarrow$ **limited portability**
  - $\Rightarrow$ multiple diverging code branches $\Rightarrow$ **hard to maintain**
- inter-node communication = **MPI**

Why OpenCL? (aka: *The Good*)

Scientific HPC in a Nutshell

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

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

- hardware life time: **5 years**
- software life time: **multiple tens of years**
  - $\Rightarrow$ outlives systems by far $\Rightarrow$ **aim for portability**

Why OpenCL? (aka: *The Good*)

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

- **portability** first ($\neq$ performance portability)
  - $\Rightarrow$ **OpenCL** has the largest **hardware coverage** for intra-node programming
  - $\Rightarrow$ library only $\Rightarrow$ **no tool dependencies**

Why OpenCL? (aka: *The Good*)

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

- **portability** first ($\neq$ performance portability)
  - $\Rightarrow$ **OpenCL** has the largest **hardware coverage** for intra-node programming
  - $\Rightarrow$ library only $\Rightarrow$ **no tool dependencies**
- use modern techniques with a broad community (beyond HPC)
  - $\Rightarrow$ **modern C++** for host code

Why OpenCL? (aka: *The Good*)

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

- **portability** first ($\neq$ performance portability)
  - $\Rightarrow$ **OpenCL** has the largest **hardware coverage** for intra-node programming
  - $\Rightarrow$ library only $\Rightarrow$ **no tool dependencies**
- use modern techniques with a broad community (beyond HPC)
  - $\Rightarrow$ **modern C++** for host code
- develop code **interdisciplinary**
  - $\Rightarrow$ domain experts design the model . . .
  - $\Rightarrow$ . . . computer scientists the software

Target Hardware

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

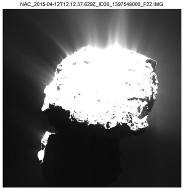
* calculated with max. AVX frequency of 1.2 GHz: 2611.2 GFLOPS = 1.2 GHz × 68 cores × 8 SIMD × 2 VPUs × 2 FMA

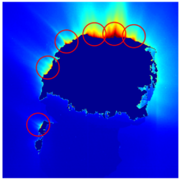
COSIM - A Predictive Cometary Coma Simulation

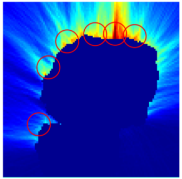
Solve dust dynamics:

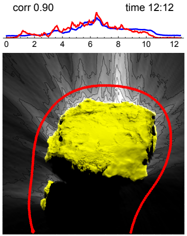
$$
\begin{aligned}
\vec{a}_{\text{dust}}(\vec{r}) &= \vec{a}_{\text{gas-drag}} + \vec{a}_{\text{grav}} + \vec{a}_{\text{Coriolis}} + \vec{a}_{\text{centrifugal}} \\
&= \frac{1}{2} C_d \alpha N_{\text{gas}}(\vec{r}) m_{\text{gas}} (\vec{v}_{\text{dust}} - \vec{v}_{\text{gas}}) |\vec{v}_{\text{dust}} - \vec{v}_{\text{gas}}| - \nabla \phi(\vec{r}) \\
&\quad -2\vec{\omega} \times \vec{v}_{\text{dust}} - \vec{\omega} \times (\vec{\omega} \times \vec{r})
\end{aligned}
$$

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





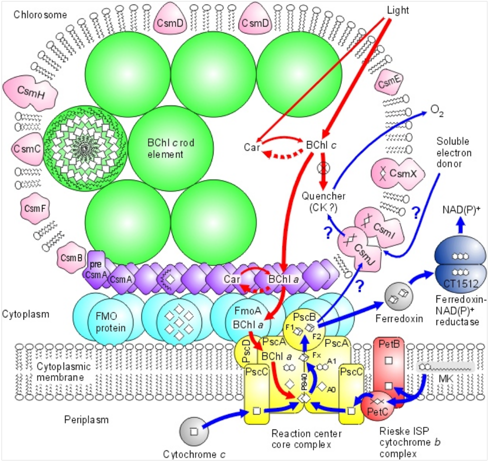




HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

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



[Image by University of Copenhagen Biology Department]

M. Noack, F. Wende, K.-D. Oertel, *OpenCL: There and Back Again*, in High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Vol. 2, 2015

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

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

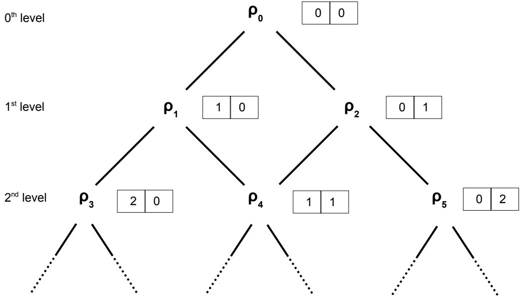
$$
\boxed{
\begin{aligned}
\frac{d\sigma_u}{dt} = {} & \frac{i}{\hbar}[H, \sigma_u] \\
& - \sigma_u \sum_{b=1}^{B} \sum_{k}^{K-1} n_{u,(b,k)} \gamma(b,k) \\
& - \sum_{b=1}^{B} \left[\frac{2\lambda_b}{\beta \hbar^2 \nu_b} - \sum_{k}^{K-1} \frac{c(b,k)}{\hbar \gamma(b,k)} \right] V_{s(b)}^{\times} V_{s(b)}^{\times}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

M. Noack, F. Wende, K.-D. Oertel, *OpenCL: There and Back Again*, in High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Vol. 2, 2015

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

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

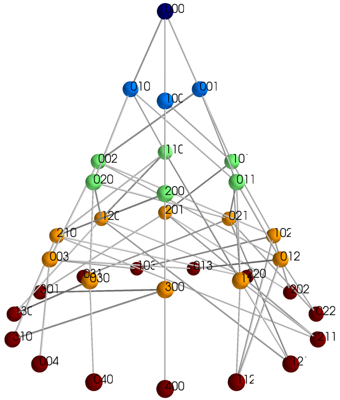


M. Noack, F. Wende, K.-D. Oertel, *OpenCL: There and Back Again*, in High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Vol. 2, 2015

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

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

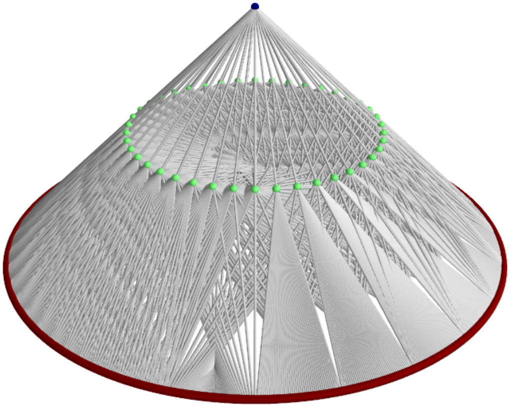


M. Noack, F. Wende, K.-D. Oertel, *OpenCL: There and Back Again*, in High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Vol. 2, 2015

HEOM - The Hierarchical Equations of Motion

Model for Open Quantum Systems

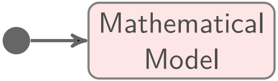
- understand the energy transfer in photo-active molecular complexes
  - $\Rightarrow$ e.g. photosynthesis
- millions of coupled ODEs
- hierarchical graph of matrices



M. Noack, F. Wende, K.-D. Oertel, *OpenCL: There and Back Again*, in High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, Vol. 2, 2015

Interdisciplinary Workflow

domain experts computer scientists

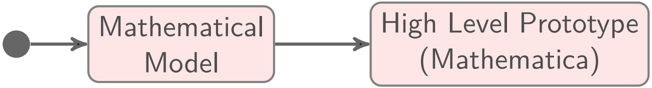


Interdisciplinary Workflow

- ODEs
- PDEs
- Graphs
- ...

Interdisciplinary Workflow

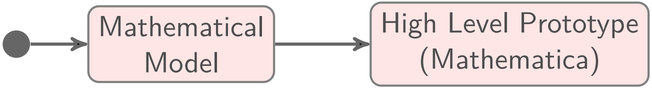
domain experts
computer scientists



- ODEs
- PDEs
- Graphs
- ...

Interdisciplinary Workflow

domain experts computer scientists

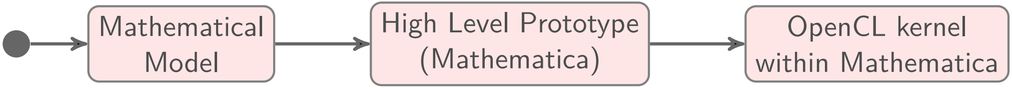


- ODEs
- PDEs
- Graphs
- ...

- domain scientist's tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

Interdisciplinary Workflow

domain experts   computer scientists



- ODEs
- PDEs
- Graphs
- ...

- domain scientist's tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

Interdisciplinary Workflow

domain experts | computer scientists

Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica

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

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 *)
```

https://reference.wolfram.com/language/OpenCLLink/guide/OpenCLLink.html

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 *)
```

special OpenCL typedefs
matching Mathematica types

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

domain experts | computer scientists

● → Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica

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

domain experts | computer scientists

● → Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica

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

domain experts | computer scientists

Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → . . .

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

domain experts | computer scientists

Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → ...

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

C++ Host Application

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

OpenCL SDKs and Versions

name	version	OpenCL version	supported devices
Intel OpenCL SDK	16.1.1	1.2 (CPU), 2.1 (GPU)	CPUs (up to AVX2), Intel GPUs
Intel OpenCL SDK	14.2	1.2	Xeon Phi (KNC)
Nvidia OpenCL	CUDA 8	1.2 (exp. 2.0)	Nvidia GPU
AMD APP SDK	3.0	2.0 (GPU), 1.2 (CPU)	GPU, CPUs (AVX,FMA4,XOP)
PoCL	0.14	2.0	CPUs (LLVM, AVX-512)

OpenCL SDKs and Versions

name	version	OpenCL version	supported devices
Intel OpenCL SDK	16.1.1	1.2 (CPU), 2.1 (GPU)	CPUs (up to AVX2), Intel GPUs
Intel OpenCL SDK	14.2	1.2	Xeon Phi (KNC)
Nvidia OpenCL	CUDA 8	1.2 (exp. 2.0)	Nvidia GPU
AMD APP SDK	3.0	2.0 (GPU), 1.2 (CPU)	GPU, CPUs (AVX,FMA4,XOP)
PoCL	0.14	2.0	CPUs (LLVM, AVX-512)

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:

$\Rightarrow$ simple, deterministic way: `oclinfo` tool $\Rightarrow$ platform/device index

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:

```
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

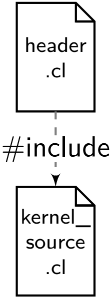
b) embedded source as **string constant**:

- ✓ *self-contained* executable for production use

Handling Kernel Source Code

a) loading **source files** at runtime:

- ✓ no host-code recompilation
- ✓ `#include` directives



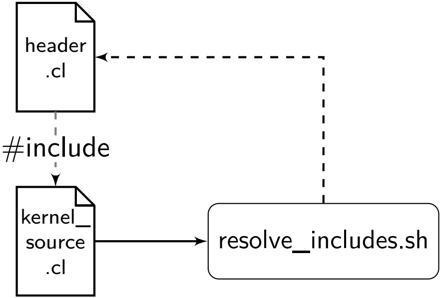
b) embedded source as **string constant**:

- ✓ *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

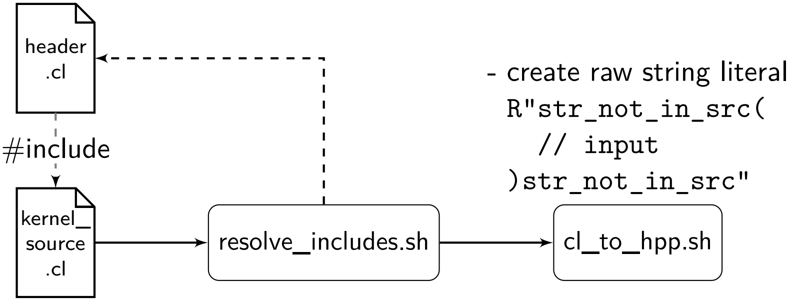
Handling Kernel Source Code

a) loading **source files** at runtime:

- ✓ no host-code recompilation
- ✓ `#include` directives

b) embedded source as **string constant**:

- ✓ *self-contained* executable for production use



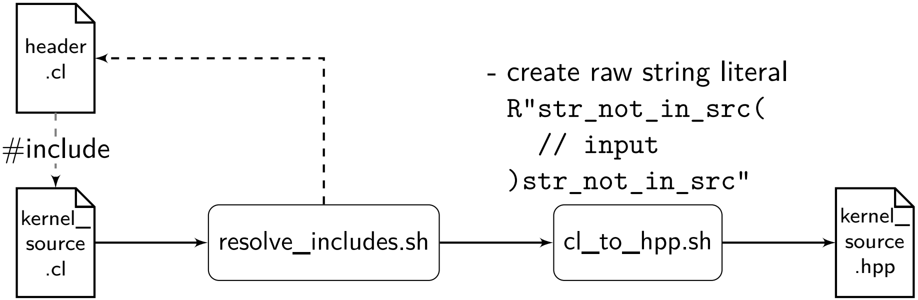
Handling Kernel Source Code

a) loading **source files** at runtime:

- ✓ no host-code recompilation
- ✓ `#include` directives

b) embedded source as **string constant**:

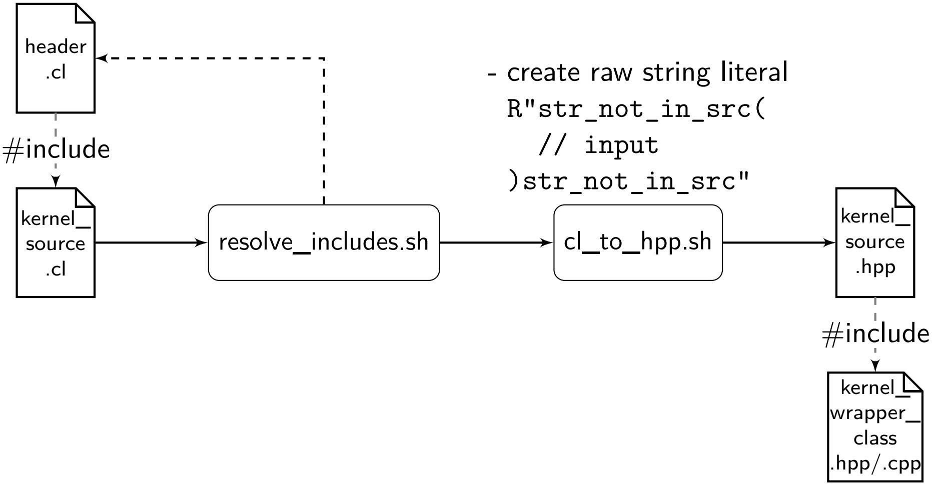
- ✓ *self-contained* executable for production use



Handling Kernel Source Code

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

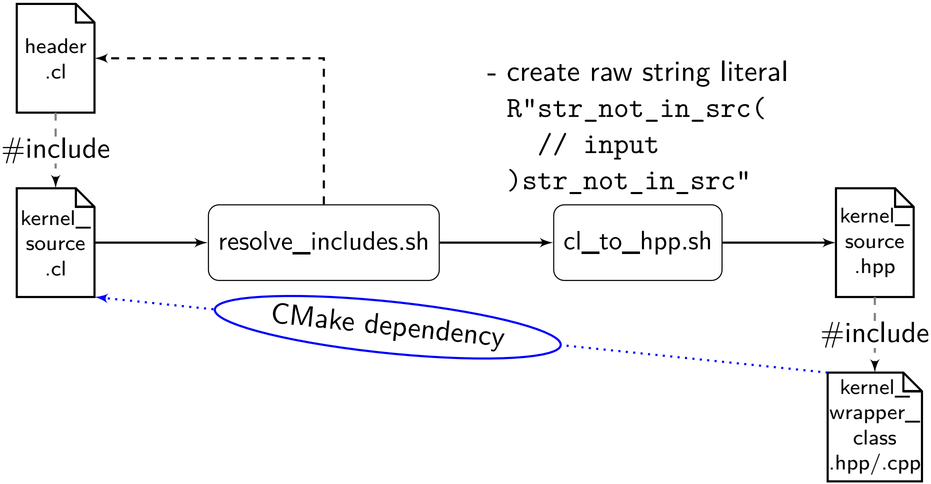
b) embedded source as **string constant**:
- ✓ *self-contained* executable for production use



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



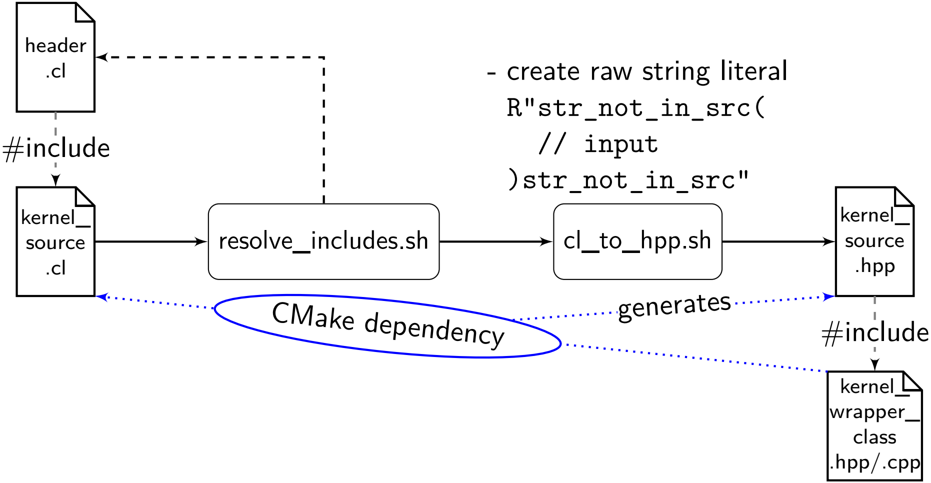
Handling Kernel Source Code

a) loading **source files** at runtime:

- ✓ no host-code recompilation
- ✓ `#include` directives

b) embedded source as **string constant**:

- ✓ *self-contained* executable for production use



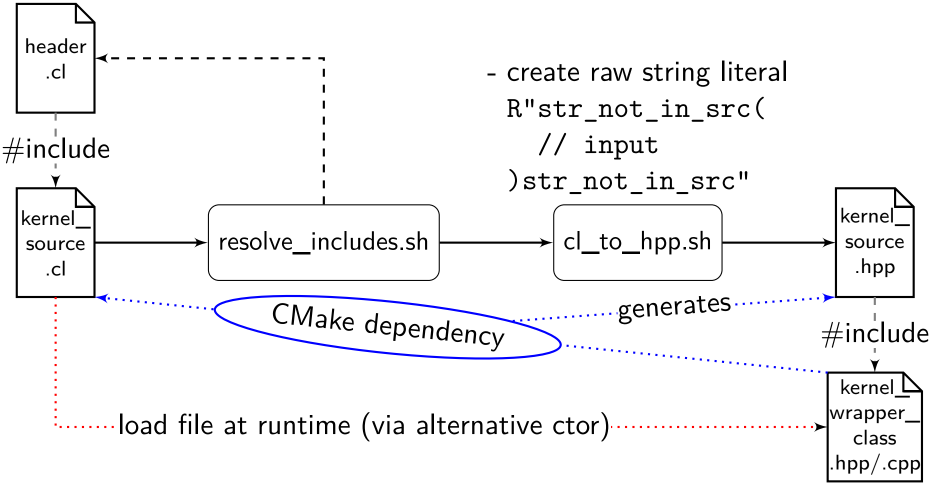
Handling Kernel Source Code

a) loading **source files** at runtime:

- ✓ no host-code recompilation
- ✓ `#include` directives

b) embedded source as **string constant**:

- ✓ *self-contained* executable for production use



Interdisciplinary Workflow

domain experts | computer scientists

● → Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → . . .

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

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 → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → . . .

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 → Distributed Host Application

C++ Host Application

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

Distributed Host Application

Interdisciplinary Workflow

domain experts | computer scientists

● → **Mathematical Model** → **High Level Prototype (Mathematica)** → **OpenCL kernel within Mathematica** → . . .

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** → **Distributed Host Application**

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

OpenCL and Communication/MPI

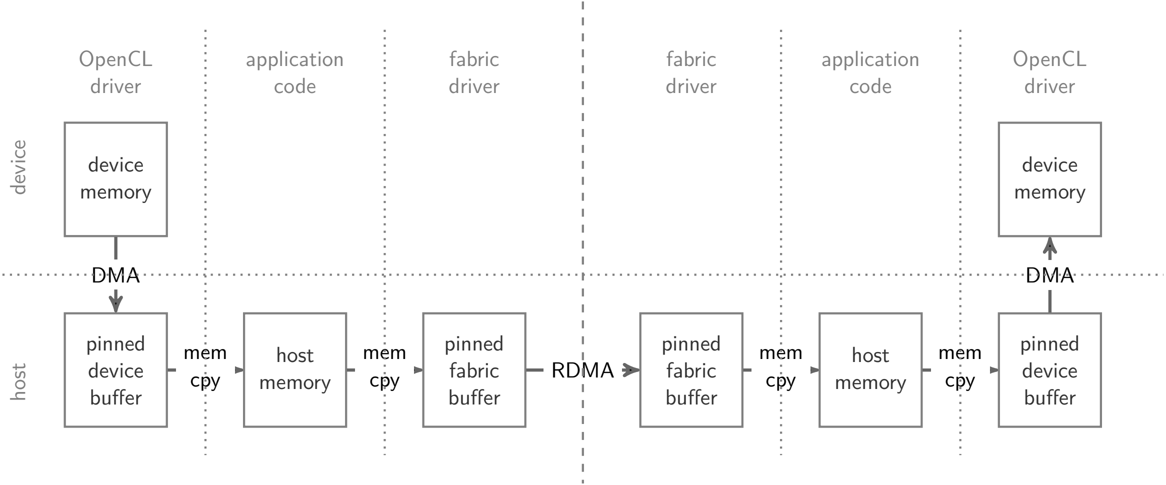
Design Recommendation:

- 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

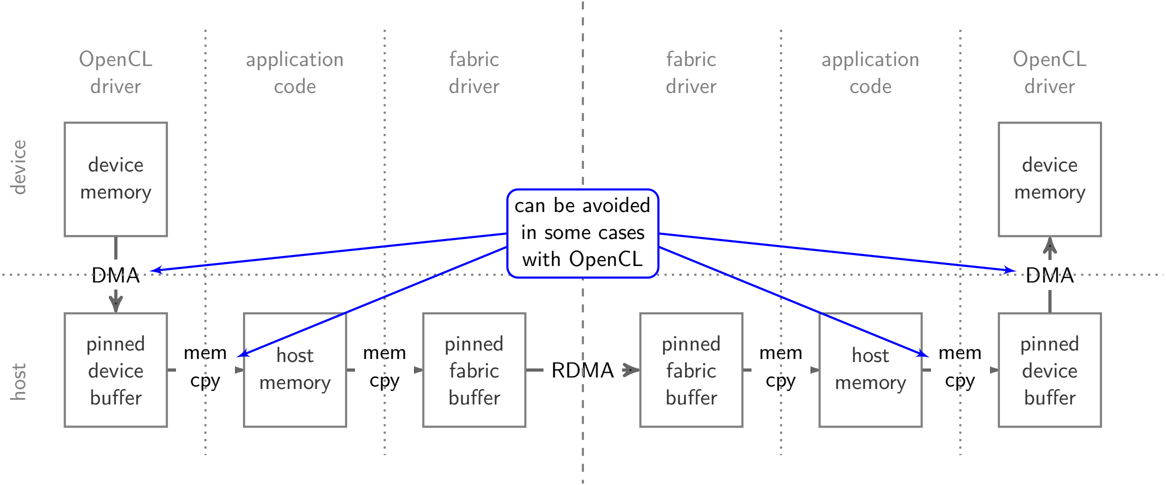
Trade-offs:

- 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

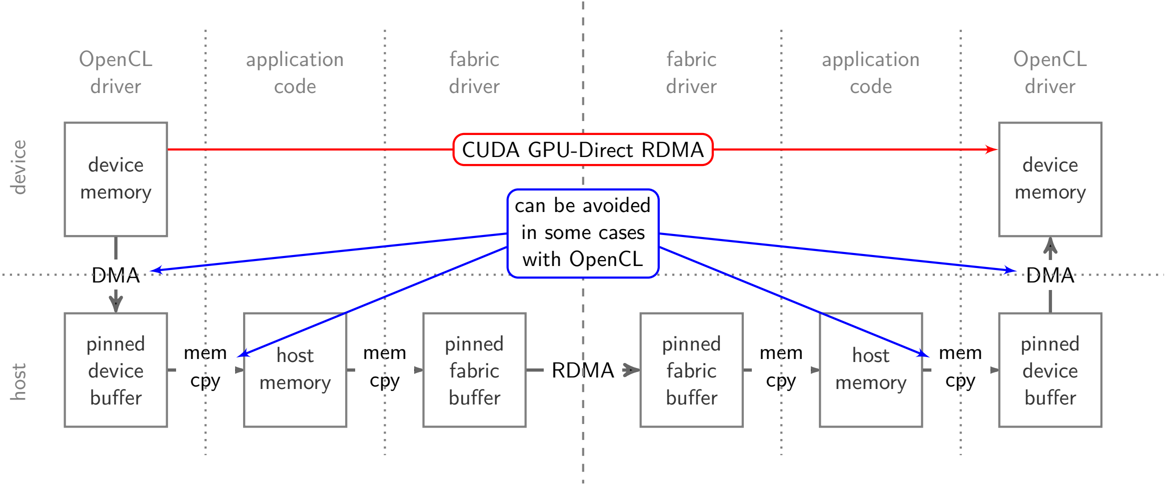
Data Transfer Paths



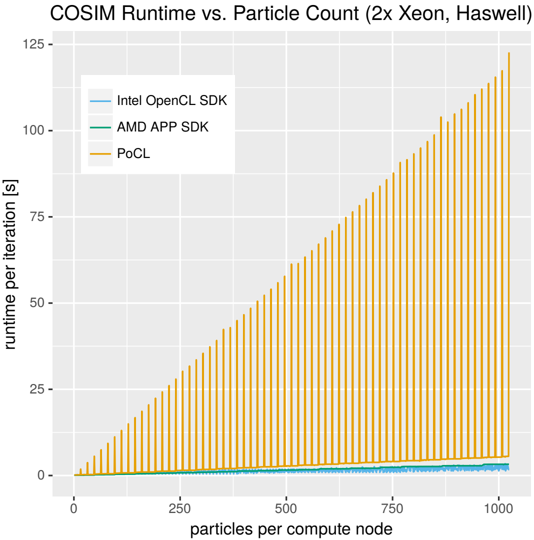
Data Transfer Paths



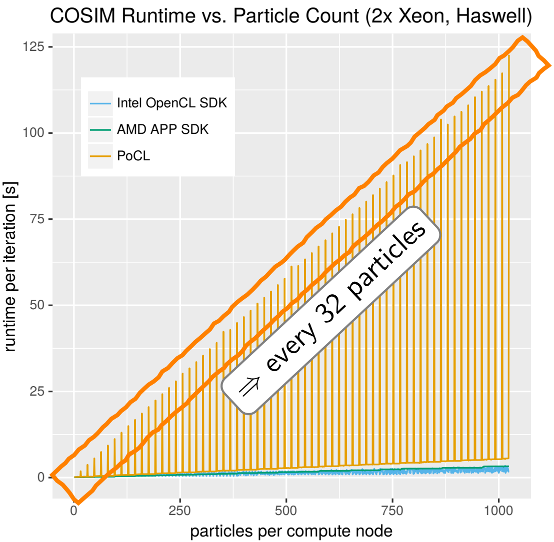
Data Transfer Paths



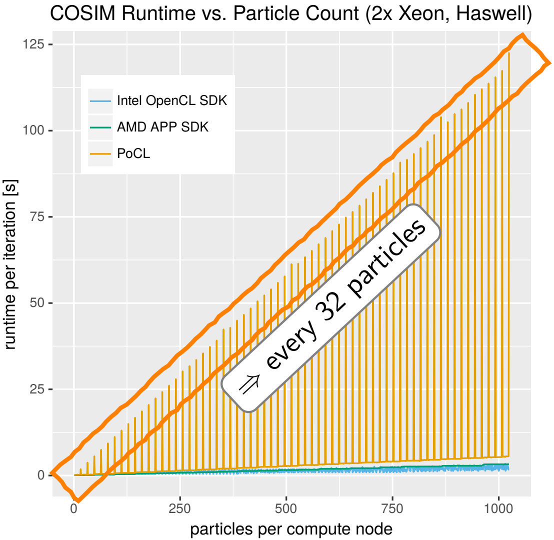
Benchmark Results: COSIM load imbalance (Xeon)

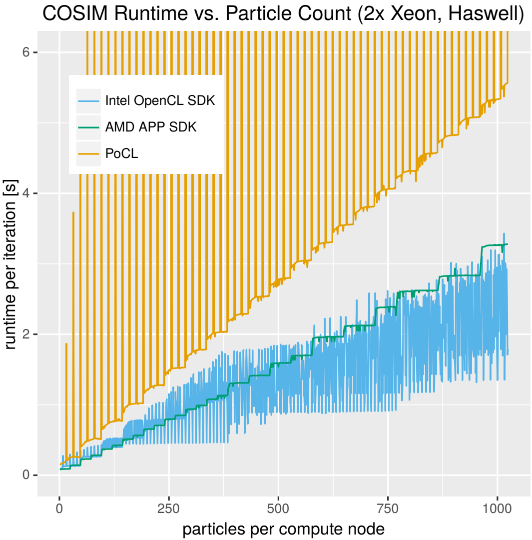


Benchmark Results: COSIM load imbalance (Xeon)

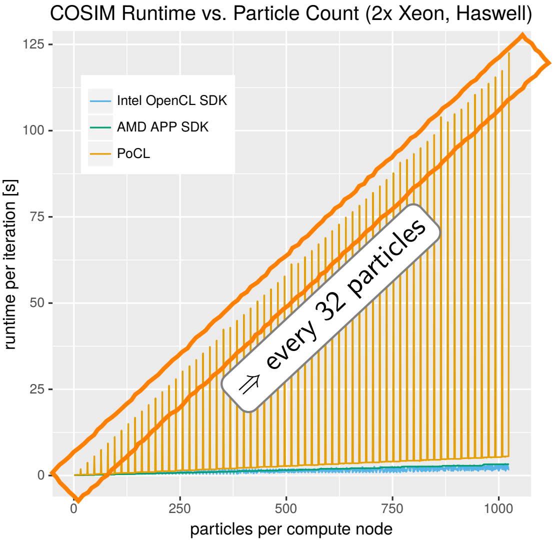


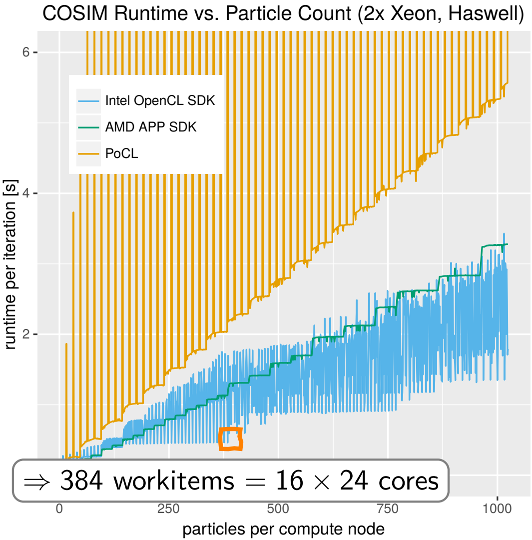
Benchmark Results: COSIM load imbalance (Xeon)



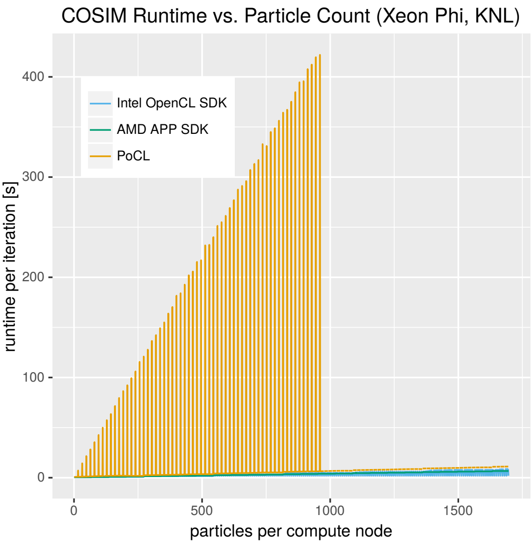


Benchmark Results: COSIM load imbalance (Xeon)

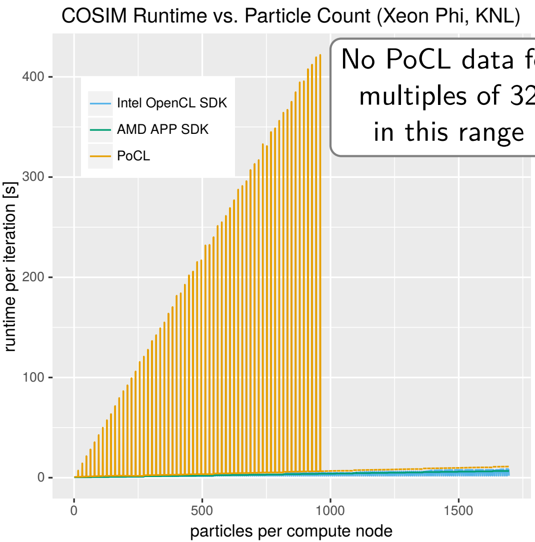




Benchmark Results: COSIM load imbalance (Xeon Phi)

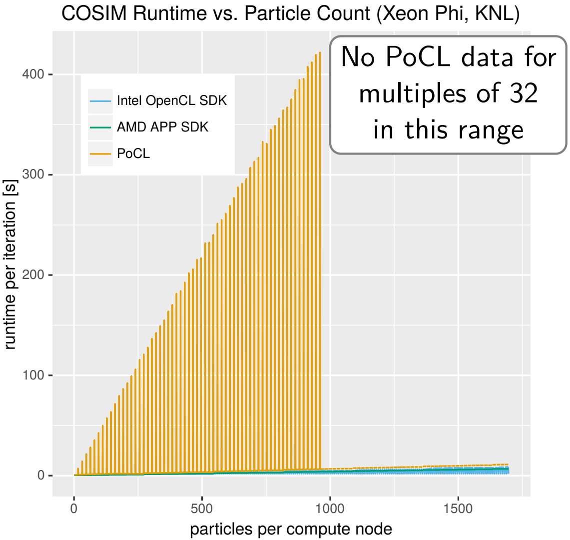


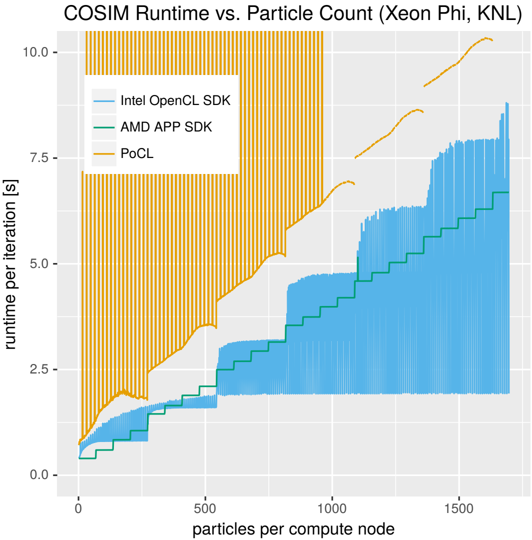
Benchmark Results: COSIM load imbalance (Xeon Phi)



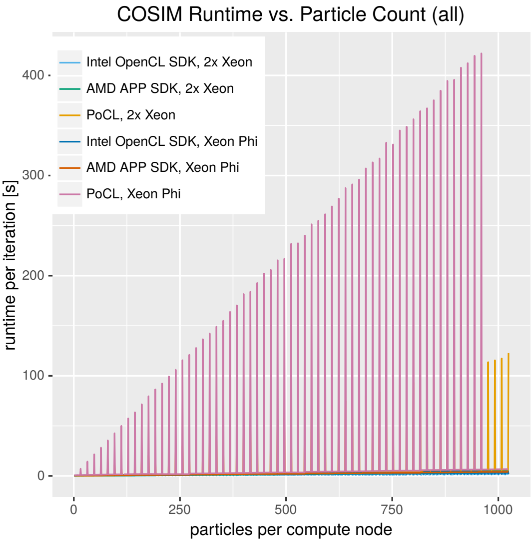
No PoCL data for multiples of 32 in this range

Benchmark Results: COSIM load imbalance (Xeon Phi)

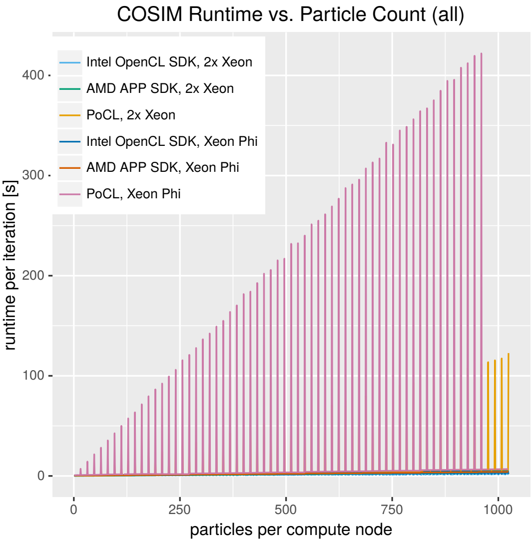


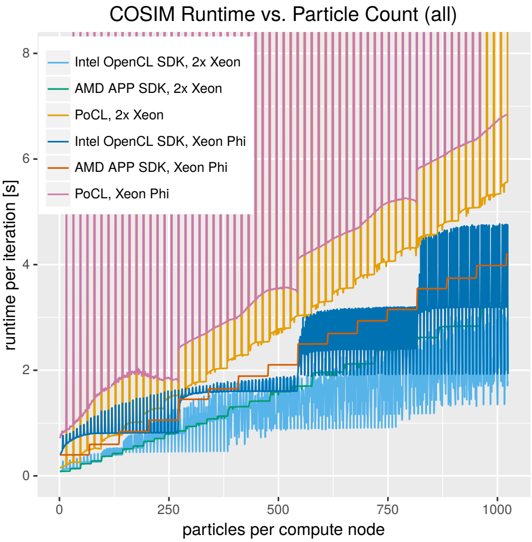


Benchmark Results: COSIM node imbalance, all



Benchmark Results: COSIM node imbalance, all





Interdisciplinary Workflow

domain experts | computer scientists

Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → …

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 → Distributed Host Application

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

domain experts | computer scientists

Mathematical Model → High Level Prototype (Mathematica) → OpenCL kernel within Mathematica → ...

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 → Distributed Host Application → Optimisation / Production Runs

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

domain experts | computer scientists

Mathematical Model → **High Level Prototype (Mathematica)** → **OpenCL kernel within Mathematica** → ...

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

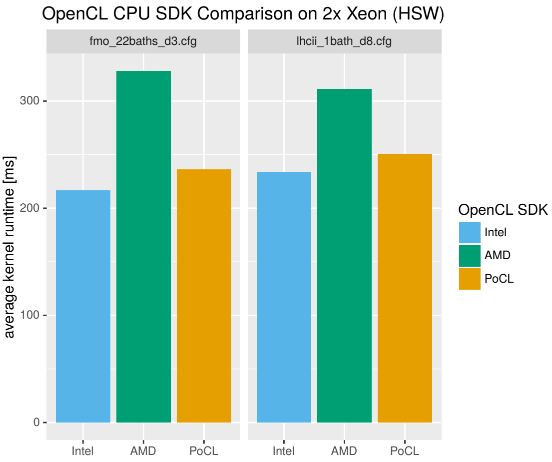
... → **C++ Host Application** → **Distributed Host Application** → **Optimisation / Production Runs**

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

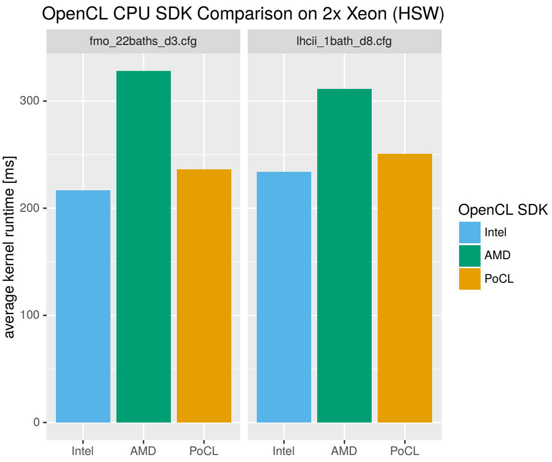
Distributed Host Application
- scale to multiple nodes
- partitioning, balancing, neighbour exchange, ...
- wrapped MPI 3.0

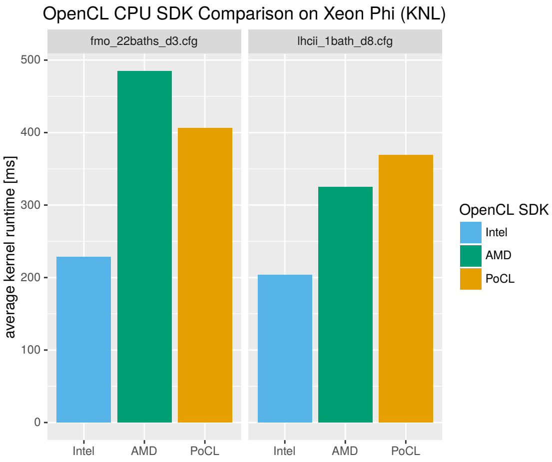
Optimisation / Production Runs
- always collect perf. data
- profile/tune code
- add performance tweaks
- use device-specific kernel variants if needed

HEOM Benchmark Results: CPU SDK comparison

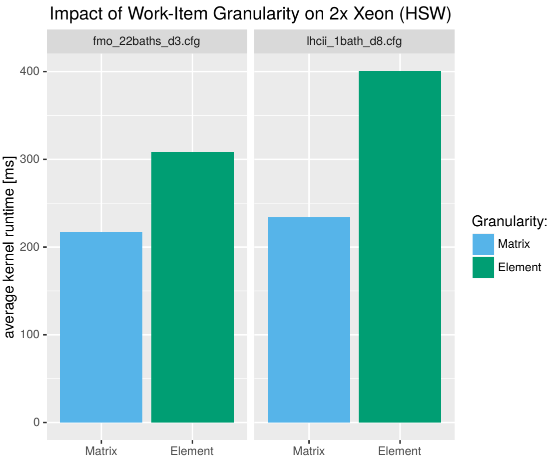


HEOM Benchmark Results: CPU SDK comparison

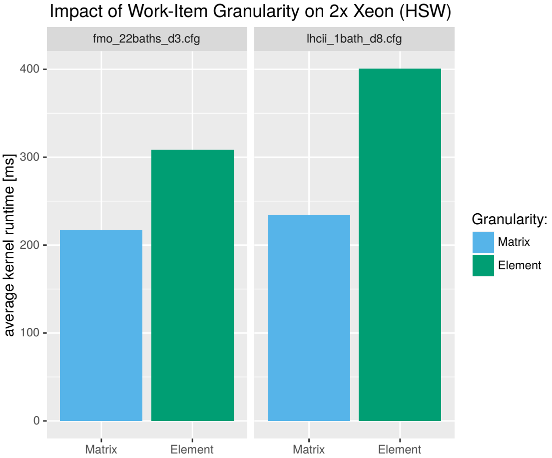


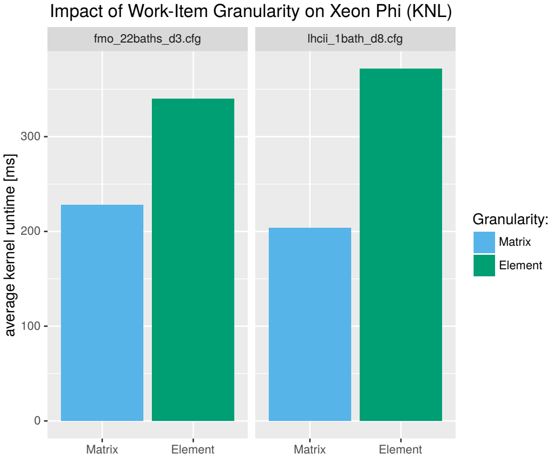


HEOM Benchmarks: Workitem Granularity on CPUs

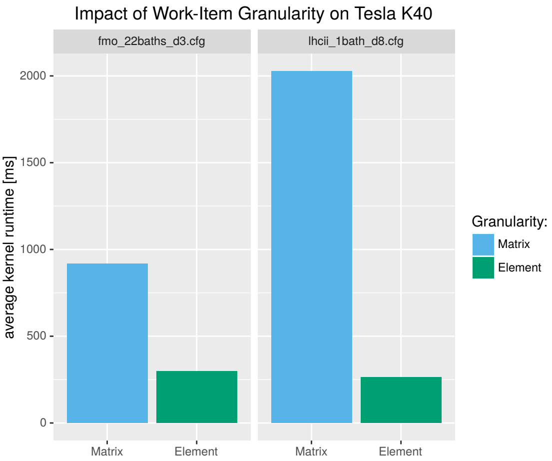


HEOM Benchmarks: Workitem Granularity on CPUs

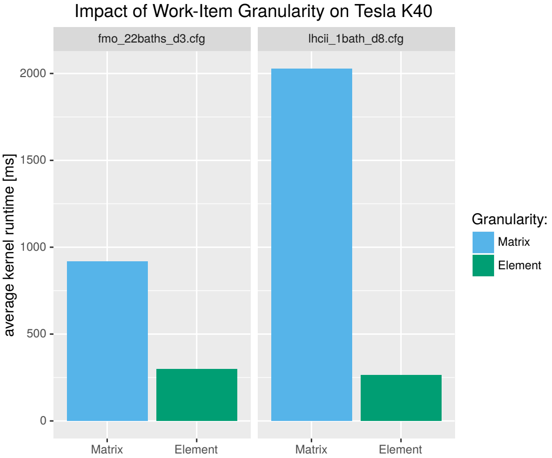


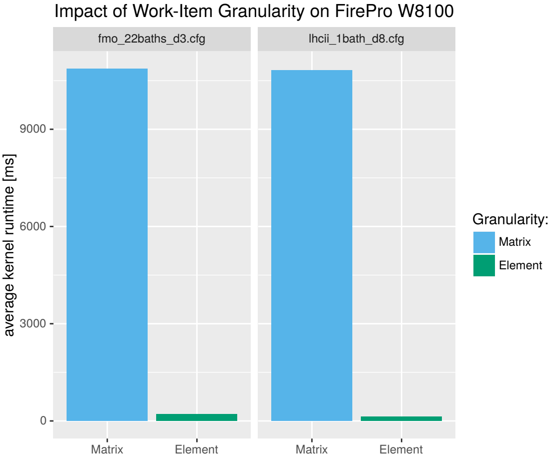


HEOM Benchmarks: Workitem Granularity on GPUs

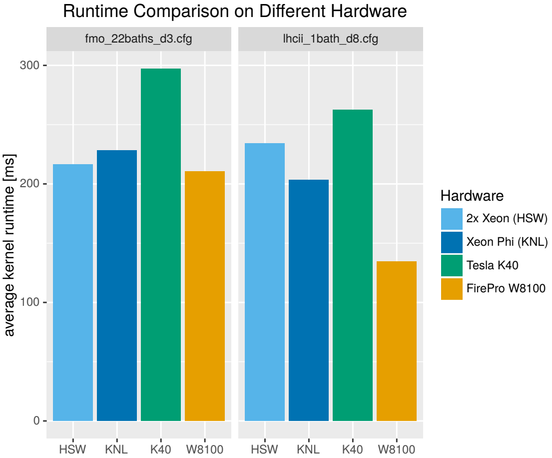


HEOM Benchmarks: Workitem Granularity on GPUs

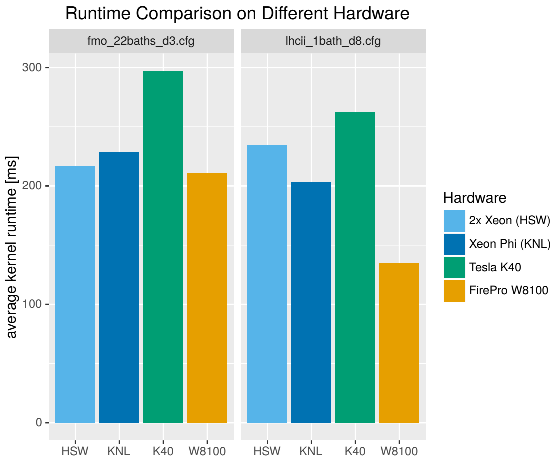


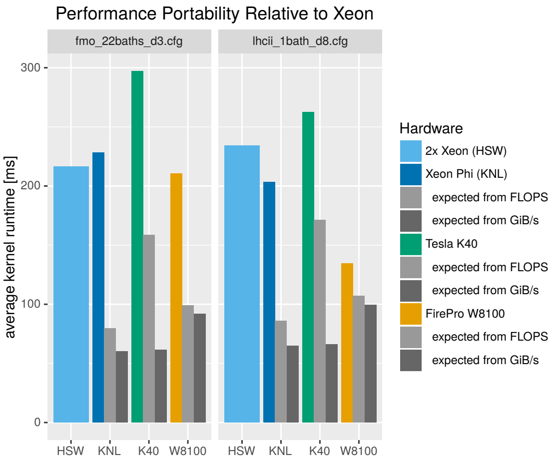


HEOM Benchmarks: Performance Portability



HEOM Benchmarks: Performance Portability





Interdisciplinary Workflow

domain experts | computer scientists

Mathematical Model → **High Level Prototype (Mathematica)** → **OpenCL kernel within Mathematica** → . . .

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** → **Distributed Host Application** → **Optimisation / Production Runs**

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 will into interdisciplinary workflow
- runtime compilation allows compiler-optimisation with runtime-constants
- performance portability is not for free, but ...
  - $\Rightarrow$ ... better to have two kernels than two programming models

Conclusion

OpenCL

- highest portability of available programming models
- integrates will into interdisciplinary workflow
- runtime compilation allows compiler-optimisation with runtime-constants
- performance portability is not for free, but ...
  - $\Rightarrow$ ...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