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?

The Situation with Scientific HPC

- tons of **legacy code** (FORTRAN) authored by domain experts
  - ⇒ rather closed community
  - ⇒ decoupled from computer science (ask a CS 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 CS 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 CS 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 ($\neq$ performance portability)

$\Rightarrow$ **OpenCL** has the largest **hardware coverage** for intra-node programming

  - CPUs, GPUs, AI accelerators, FPGAs, ...

$\Rightarrow$ library only $\Rightarrow$ **no tool dependencies**

Do not contribute to that situation!

What can we do better?

- put **portability** first ($\neq$ performance portability)
  $\Rightarrow$ **OpenCL** has the largest **hardware coverage** for intra-node programming
    - CPUs, GPUs, AI accelerators, FPGAs, ...

  $\Rightarrow$ library only $\Rightarrow$ **no tool dependencies**

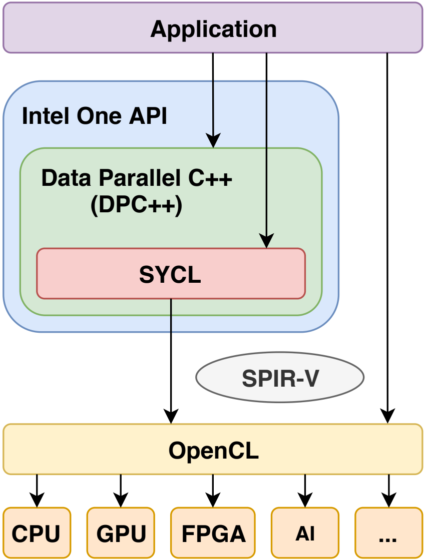
- use modern techniques with a broad community (beyond HPC)
  $\Rightarrow$ e.g. **modern C++** for host code
  $\Rightarrow$ e.g. **CMake** for building

Do not contribute to that situation!

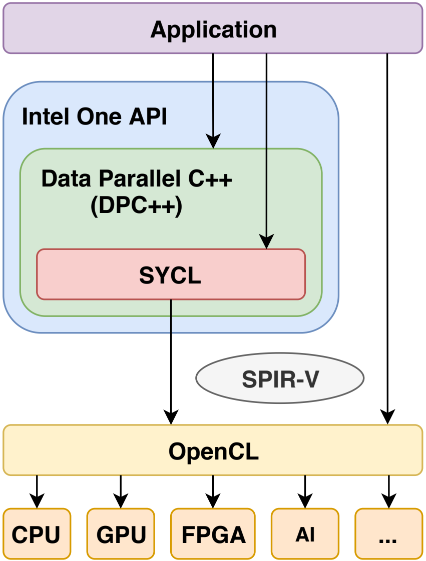
What can we do better?

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

Big Picture: OpenCL, SYCL, SPIR-V and Intel One API

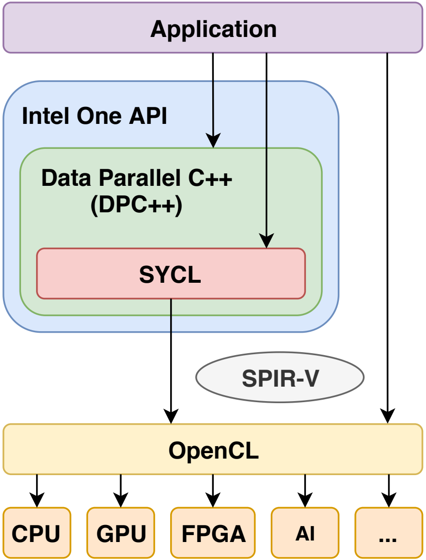


Big Picture: OpenCL, SYCL, SPIR-V and Intel One API



- **higher-level** programming model for OpenCL
- **single source**, standard C++,
  $\Rightarrow$ **SYCL compiler** needed

Big Picture: OpenCL, SYCL, SPIR-V and Intel One API



SYCL

- **higher-level** programming model for OpenCL
- **single source**, standard C++,
  $\Rightarrow$ **SYCL compiler** needed

SPIR

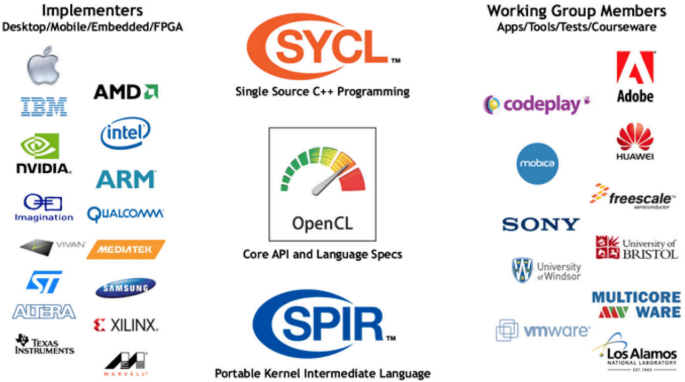
- **S**tandard **P**ortable **I**ntermediate **R**epresentation
- standardised intermediate language
  $\Rightarrow$ based on LLVM-IR
- **device-independent binaries** for OpenCL

OpenCL (Open Computing Language) in a Nutshell

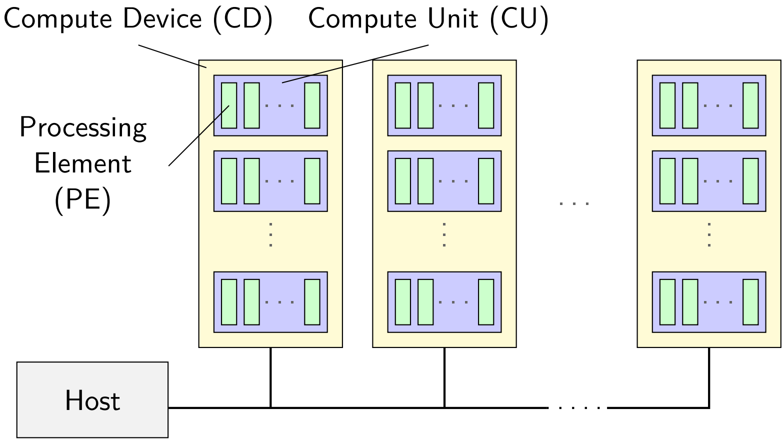
- open, royalty-free **standard** for cross-platform, **parallel programming**
- maintained by **Khronos Group**
- personal computers, servers, mobile devices and embedded platforms
- first **released: 2009-08-28**

OpenCL (Open Computing Language) in a Nutshell

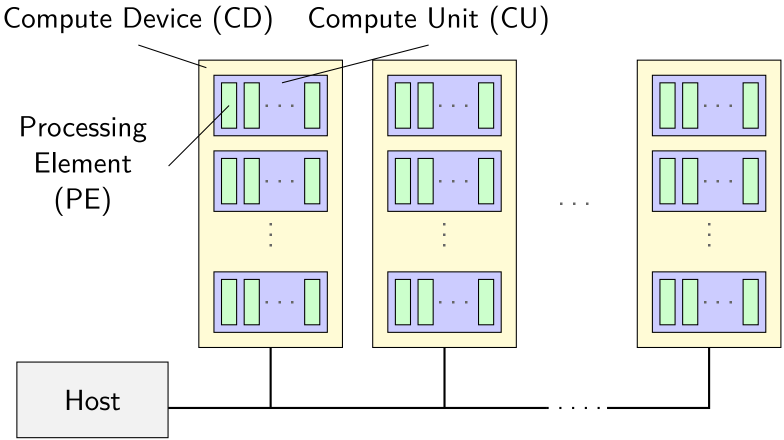
- open, royalty-free **standard** for cross-platform, **parallel programming**
- maintained by **Khronos Group**
- personal computers, servers, mobile devices and embedded platforms
- first **released: 2009-08-28**



OpenCL Platform and Memory Model



OpenCL Platform and Memory Model



Memory Model:

- CD has device memory with **glottal/constant** addr. space
- CU has **local** memory addr. space
- PE has **private** memory addr. space

⇒ relaxed consistency

OpenCL Machine Model Mapping

OpenCL Platform	CPU Hardware	GPU Hardware
Compute Device	Processor/Board	GPU device
Compute Unit	Core (thread)	Streaming MP
Processing Element	SIMD Lane	CUDA Core
global/const. memory	DRAM	DRAM
local memory	DRAM	Shared Memory
private memory	Register/DRAM	Priv. Mem./Register

OpenCL Machine Model Mapping

OpenCL Platform	**CPU Hardware**	**GPU Hardware**
Compute Device	Processor/Board	GPU device
Compute Unit	Core (thread)	Streaming MP
Processing Element	SIMD Lane	CUDA Core
global/const. memory	DRAM	DRAM
local memory	DRAM	Shared Memory
private memory	Register/DRAM	Priv. Mem./Register

$\Rightarrow$ write code for this **abstract machine model**

OpenCL Machine Model Mapping

OpenCL Platform	**CPU Hardware**	**GPU Hardware**
Compute Device	Processor/Board	GPU device
Compute Unit	Core (thread)	Streaming MP
Processing Element	SIMD Lane	CUDA Core
global/const. memory	DRAM	DRAM
local memory	DRAM	Shared Memory
private memory	Register/DRAM	Priv. Mem./Register

$\Rightarrow$ write code for this **abstract machine model**

$\Rightarrow$ device-specific OpenCL compiler and runtime maps it to **actual hardware**

OpenCL Machine Model Mapping

OpenCL Platform	**CPU Hardware**	**GPU Hardware**
Compute Device	Processor/Board	GPU device
Compute Unit	Core (thread)	Streaming MP
Processing Element	SIMD Lane	CUDA Core
global/const. memory	DRAM	DRAM
local memory	DRAM	Shared Memory
private memory	Register/DRAM	Priv. Mem./Register

⇒ write code for this **abstract machine model**

⇒ device-specific OpenCL compiler and runtime maps it to **actual hardware**

⇒ **library-only** implementation: no toolchain, many language bindings

OpenCL Machine Model Mapping

OpenCL Platform	CPU Hardware	GPU Hardware
Compute Device	Processor/Board	GPU device
Compute Unit	Core (thread)	Streaming MP
Processing Element	SIMD Lane	CUDA Core
global/const. memory	DRAM	DRAM
local memory	DRAM	Shared Memory
private memory	Register/DRAM	Priv. Mem./Register

⇒ write code for this **abstract machine model**

⇒ device-specific OpenCL compiler and runtime maps it to **actual hardware**

⇒ **library-only** implementation: no toolchain, many language bindings

⇒ currently: **widest practical portability** of parallel programming models

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

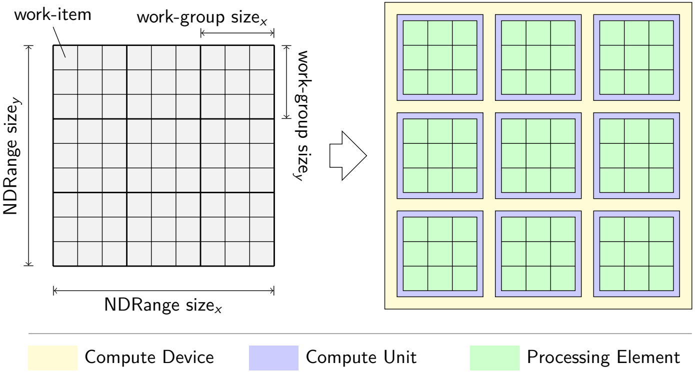
OpenCL Host Program and Kernel Execution

⇒ 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:

Device Name (architecture)	**[TFLOPS]**	**[GiB/s]**	**[FLOP/Byte]**
2× Intel Xeon Gold 6138 (SKL)	2.56	238	**10.8**
2× Intel Xeon E5-2680v3 (HSW)	0.96	136	**7.1**
Intel Xeon Phi 7250 (KNL)	2.61[a]	490/115[b]	**5.3/22.7**[b]
Nvidia Tesla K40 (Kepler)	1.31	480	**2.7**
AMD Firepro W8100 (Hawaii)	2.1	320	**6.6**

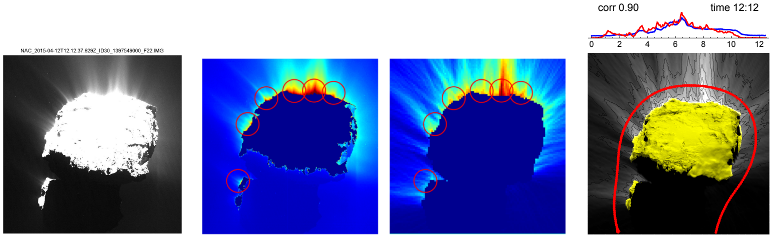
* calculated with max. AVX frequency of 1.2 GHz: 2611.2 GFLOPS = 1.2 GHz × 68 cores × 8 SIMD × 2 VPUs × 2 FMA

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:



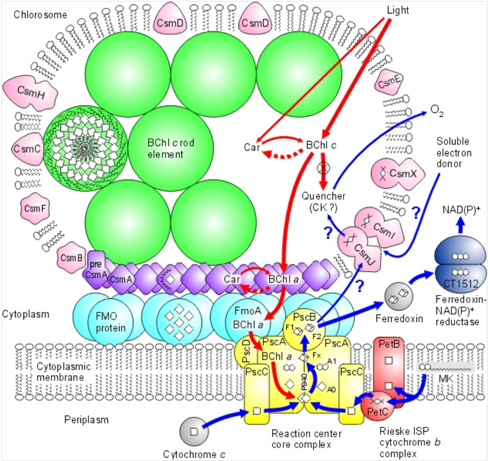
Panels 1-2: OSIRIS NAC Image, Panels 3-4: Simulation Results, Right Image: ESA – C. Carreau/ATG medialab, CC BY-SA 3.0-igo

DM-HEOM: Computing the **H**ierarchical **E**quations **Of** **M**otion

Model for Open Quantum Systems

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

…but also **quantum computing**



[Image by University of Copenhagen Biology Department]

M. Noack, A. Reinefeld, T. Kramer, Th. Steinke, *DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion*, IPDPS/PDSEC'18, 10.1109/IPDPSW.2018.00149

DM-HEOM: Computing the **H**ierarchical **E**quations **O**f **M**otion

Model for Open Quantum Systems

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

  . . . but also **quantum computing**
- millions of coupled ODEs

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

M. Noack, A. Reinefeld, T. Kramer, Th. Steinke, *DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion*, IPDPS/PDSEC'18, 10.1109/IPDPSW.2018.00149

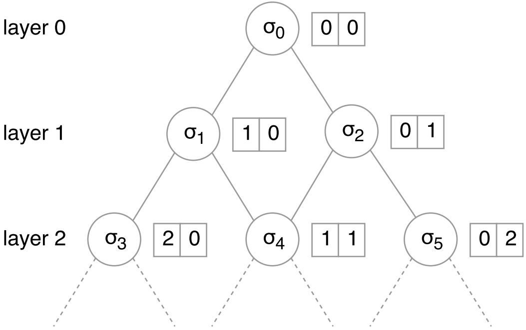
DM-HEOM: Computing the **H**ierarchical **E**quations **O**f **M**otion

Model for Open Quantum Systems

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

  . . . but also **quantum computing**

- millions of coupled ODEs
- hierarchical graph of complex matrices (auxiliary density operators, ADOs)
  - ⇒ dim: $N_{\text{sites}} \times N_{\text{sites}}$
  - ⇒ count: **exp. in hierarchy depth** d



M. Noack, A. Reinefeld, T. Kramer, Th. Steinke, *DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion*, IPDPS/PDSEC'18, 10.1109/IPDPSW.2018.00149

DM-HEOM: Computing the **H**ierarchical **E**quations **O**f **M**otion

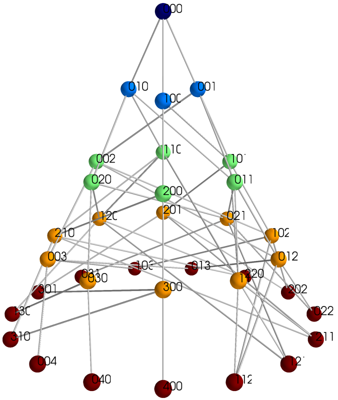
Model for Open Quantum Systems

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

  ... but also **quantum computing**

- millions of coupled ODEs

- hierarchical graph of complex matrices (auxiliary density operators, ADOs)
  - ⇒ dim: $N_{\text{sites}} \times N_{\text{sites}}$
  - ⇒ count: **exp. in hierarchy depth** d



M. Noack, A. Reinefeld, T. Kramer, Th. Steinke, *DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion*, IPDPS/PDSEC'18, 10.1109/IPDPSW.2018.00149

DM-HEOM: Computing the **H**ierarchical **E**quations **O**f **M**otion

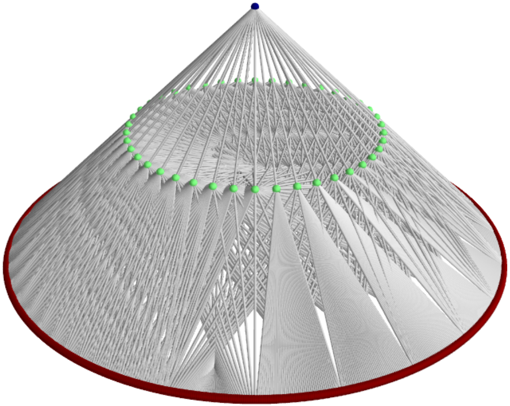
Model for Open Quantum Systems

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

  . . . but also **quantum computing**

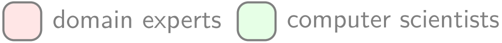
- millions of coupled ODEs

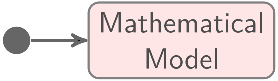
- hierarchical graph of complex matrices (auxiliary density operators, ADOs)
  - ⇒ dim: $N_{\text{sites}} \times N_{\text{sites}}$
  - ⇒ count: **exp. in hierarchy depth** d



M. Noack, A. Reinefeld, T. Kramer, Th. Steinke, *DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion*, IPDPS/PDSEC'18, 10.1109/IPDPSW.2018.00149

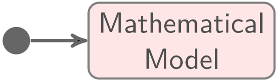
Interdisciplinary Workflow





Interdisciplinary Workflow

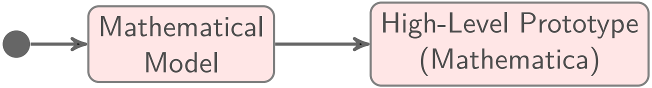
domain experts | computer scientists



- ODEs
- PDEs
- Graphs
- ...

Interdisciplinary Workflow

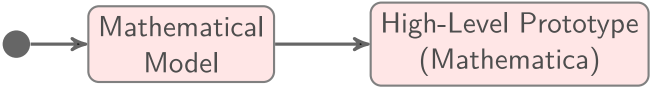
domain experts · computer scientists



- ODEs
- PDEs
- Graphs
- ...

Interdisciplinary Workflow

domain experts | computer scientists

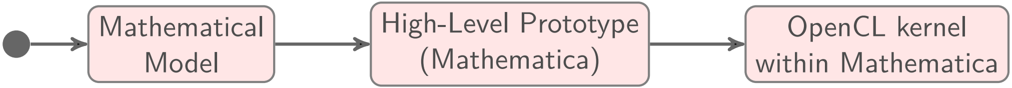


- ODEs
- PDEs
- Graphs
- ...

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

Interdisciplinary Workflow

domain experts | computer scientists

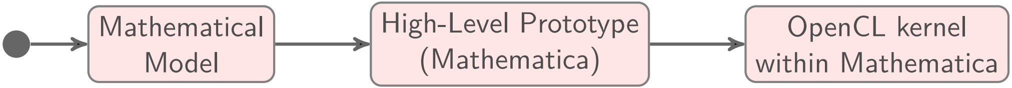


- ODEs
- PDEs
- Graphs
- . . .

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

Interdisciplinary Workflow

domain experts computer scientists



- ODEs
- PDEs
- Graphs
- ...

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

- 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

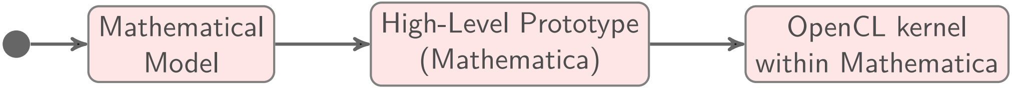
Mathematica and OpenCL

```
(* 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



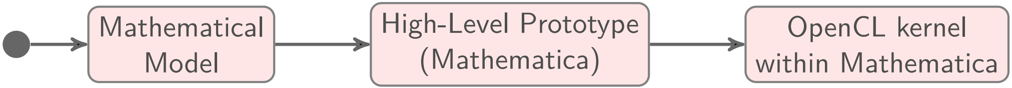
- ODEs
- PDEs
- Graphs
- ...

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

- replace some code with OpenCL

Interdisciplinary Workflow

domain experts | computer scientists



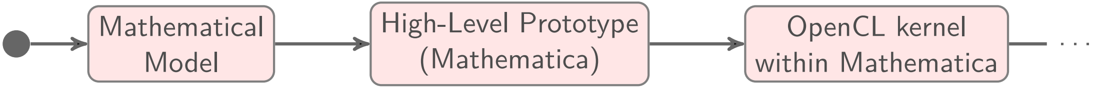
- ODEs
- PDEs
- Graphs
- ...

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

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

Interdisciplinary Workflow

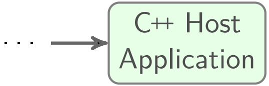
domain experts | computer scientists



- ODEs
- PDEs
- Graphs
- . . .

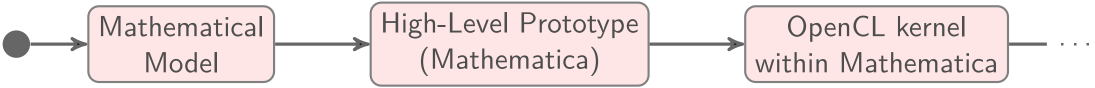
- domain scientist's tool
- high level
- symbolic solvers
- arbitrary precision
- very limited performance

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



Interdisciplinary Workflow

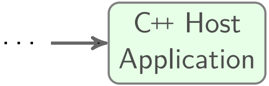
domain experts | computer scientists



- ODEs
- PDEs
- Graphs
- . . .

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

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



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

OpenCL SDKs and Versions

name	version	OpenCL version	supported devices
Intel OpenCL	**18.1**	CPU: 2.1, GPU: 2.1	CPUs (**AVX-512**), Intel GPUs, **no KNL**
Intel OpenCL	...	CPU: 1.2 (exp. 2.1), GPU: 2.1	CPUs (**up to AVX2**), Intel GPUs
Intel OpenCL	**14.2**	1.2	Xeon Phi (**KNC**, IMCI SIMD)
Nvidia OpenCL	CUDA 10.1	1.2 (exp. 2.0)	Nvidia GPU
AMD-APP SDK	$\geq$ 18.8.1	2.0 (GPU), ~~1.2 (CPU)~~	GPU, ~~CPUs (AVX,FMA4,XOP)~~
PoCL	1.3	2.0	CPUs (AVX-512), GPUs, ARM

OpenCL SDKs and Versions

name	version	OpenCL version	supported devices
Intel OpenCL	**18.1**	CPU: 2.1, GPU: 2.1	CPUs (**AVX-512**), Intel GPUs, **no KNL**
Intel OpenCL	...	CPU: 1.2 (exp. 2.1), GPU: 2.1	CPUs (**up to AVX2**), Intel GPUs
Intel OpenCL	**14.2**	1.2	Xeon Phi (**KNC**, IMCI SIMD)
Nvidia OpenCL	CUDA 10.1	1.2 (exp. 2.0)	Nvidia GPU
AMD-APP SDK	$\geq$ 18.8.1	2.0 (GPU), ~~1.2 (CPU)~~	GPU, ~~CPUs (AVX,FMA4,XOP)~~
PoCL	1.3	2.0	CPUs (AVX-512), GPUs, ARM

$\Rightarrow$ Intel rediscovered OpenCL for HPC (**One API**)
$\Rightarrow$ OpenCL 2.x mostly supported now, but 1.2 is still lowest common denominator
$\Rightarrow$ many more, e.g. FPGA SDKs by Intel (Altera), and Xylinx

The OpenCL Installable Client Driver (ICD) Loader

- allows multiple OpenCL installations to be installed and used next to each other
- applications link with a generic `libOpenCL.so`

The OpenCL Installable Client Driver (ICD) Loader

- allows multiple OpenCL installations to be installed and used next to each other
- applications link with a generic `libOpenCL.so`

```
/
├── usr/lib/
│      └── libOpenCL.so ...... runtime loader looking into /etc/OpenCL/vendors
├── etc/OpenCL/vendors/
│      ├── intel64.icd ......... textfile with actual library name/path
│      └── amdocl64.icd ........ textfile with actual library name/path
├── 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
- applications link with a generic `libOpenCL.so`

```
/
├── usr/lib/
│       └── libOpenCL.so ...... runtime loader looking into /etc/OpenCL/vendors
├── etc/OpenCL/vendors/
│                 ├── intel64.icd .......... textfile with actual library name/path
│                 └── amdocl64.icd ........ textfile with actual library name/path
├── 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**)

Compilation

OpenCL Header Files:

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

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 (optional, shown here: bypass `libOpenCL.so`):

```
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
```

Platform and Device Selection

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

Platform and Device Selection

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

```
Platform 0:
  NAME:    AMD Accelerated Parallel Processing
  VERSION: OpenCL 2.0 AMD-APP (1912.5)
  Device 0:
    NAME:    Hawaii
    VENDOR:  Advanced Micro Devices, Inc.
    VERSION: OpenCL 2.0 AMD-APP (1912.5)
  Device 1:
    NAME:    Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
    VENDOR:  GenuineIntel
    VERSION: OpenCL 1.2 AMD-APP (1912.5)
Platform 1:
  NAME:    Intel(R) OpenCL
  VERSION: OpenCL 1.2 LINUX
  Device 0:
    NAME:    Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
    VENDOR:  Intel(R) Corporation
    VERSION: OpenCL 1.2 (Build 57)
```

Handling Kernel Source Code

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

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

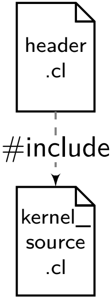
b) embedded source as **string constant**:

- ✓ self-contained executable for production use

Handling Kernel Source Code

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

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



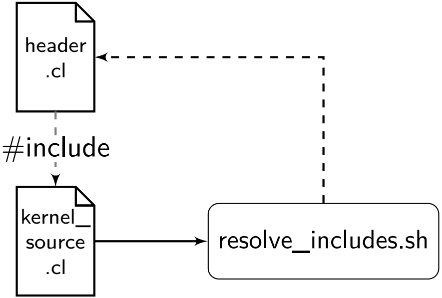
b) embedded source as **string constant**:

- ✓ 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

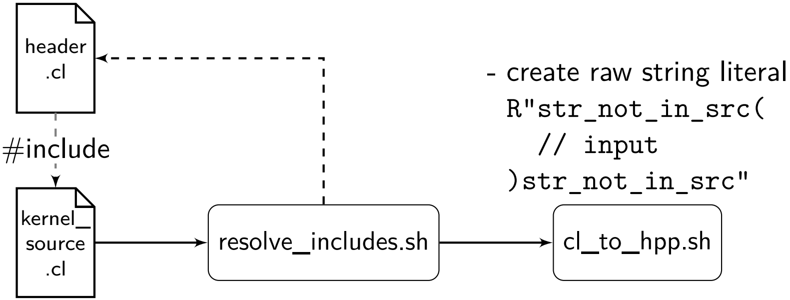
Handling Kernel Source Code

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

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

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

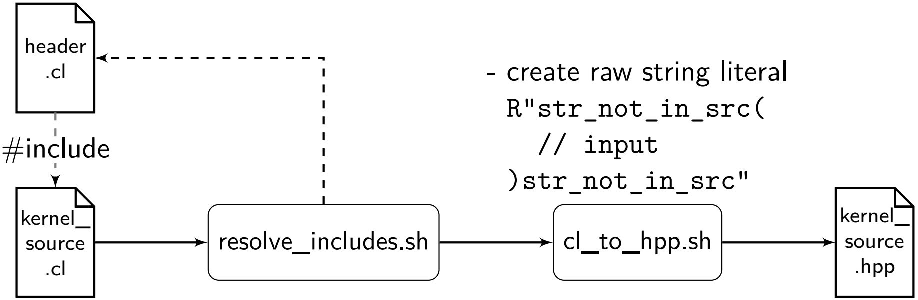
- ✓ self-contained executable for production use



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



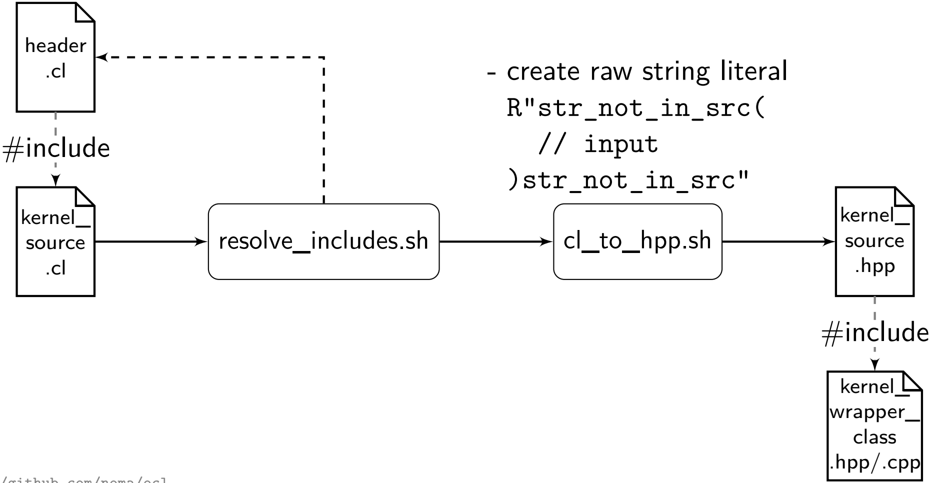
Handling Kernel Source Code

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

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

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

- ✓ self-contained executable for production use



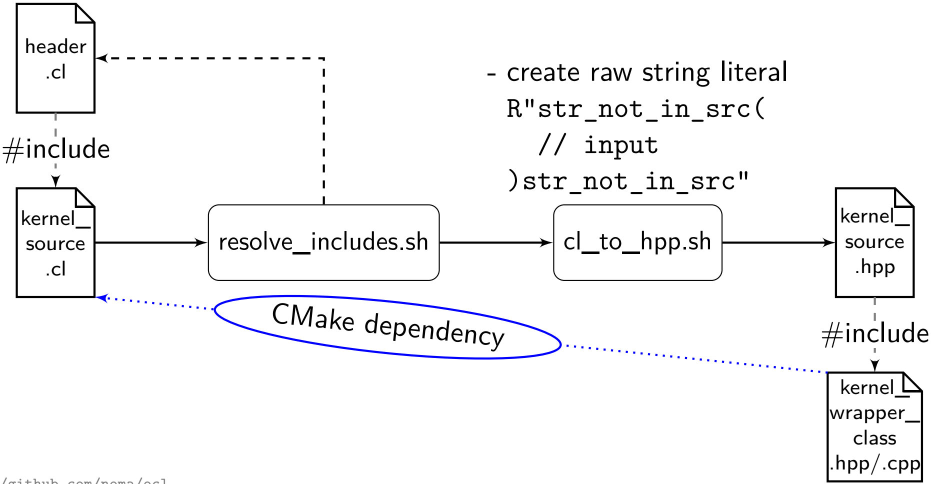
Handling Kernel Source Code

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

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

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

- ✓ self-contained executable for production use



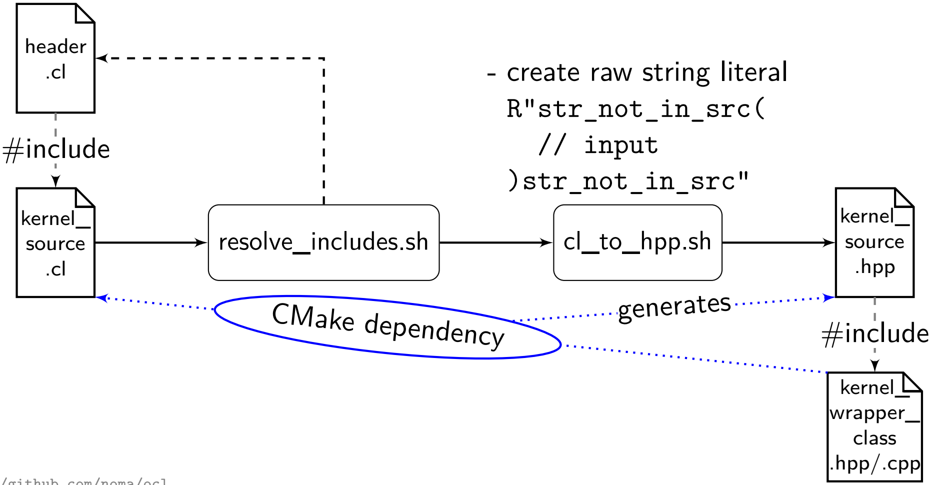
Handling Kernel Source Code

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

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

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

- ✓ self-contained executable for production use



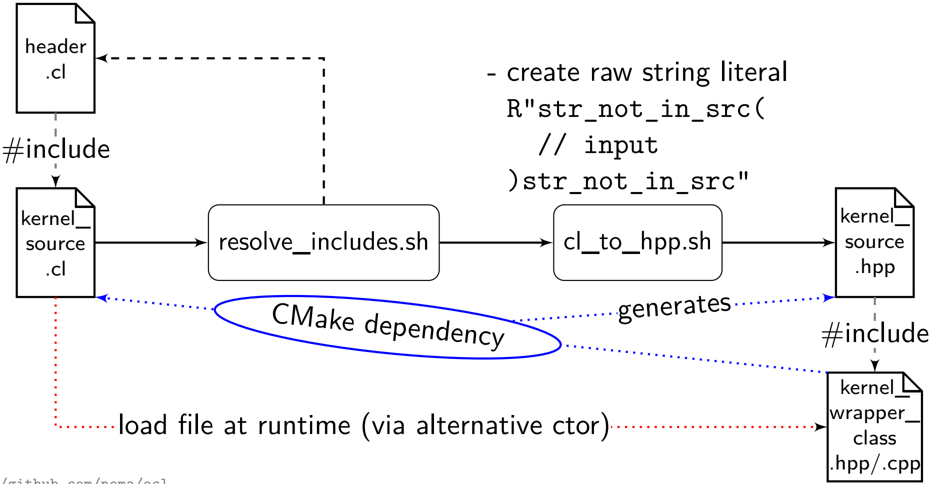
Handling Kernel Source Code

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

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

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

- ✓ self-contained executable for production use



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=
```

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**

- start single node
- OpenCL 1.2 for hotspots
- modern C++ 11/14/17
- 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++ 11/14/17
- 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++ 11/14/17
- 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
- provide **hooks** to trigger communication in-between kernel calls
- **wrap** needed parts of MPI in a thin, exchangeable abstraction layer

OpenCL and Communication/MPI

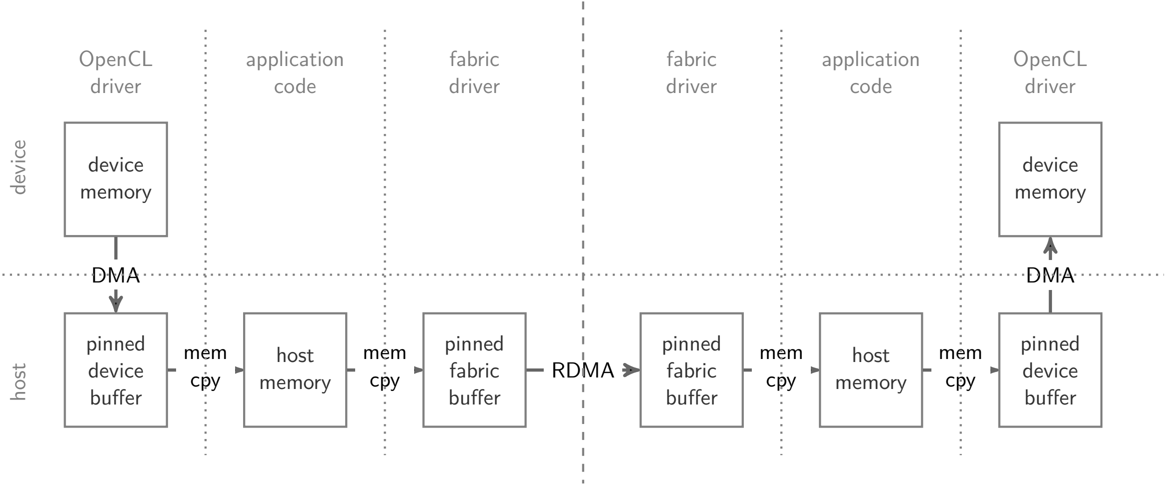
Design Recommendation:

- 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
- provide **hooks** to trigger communication in-between kernel calls
- **wrap** needed parts of MPI in a thin, exchangeable abstraction layer

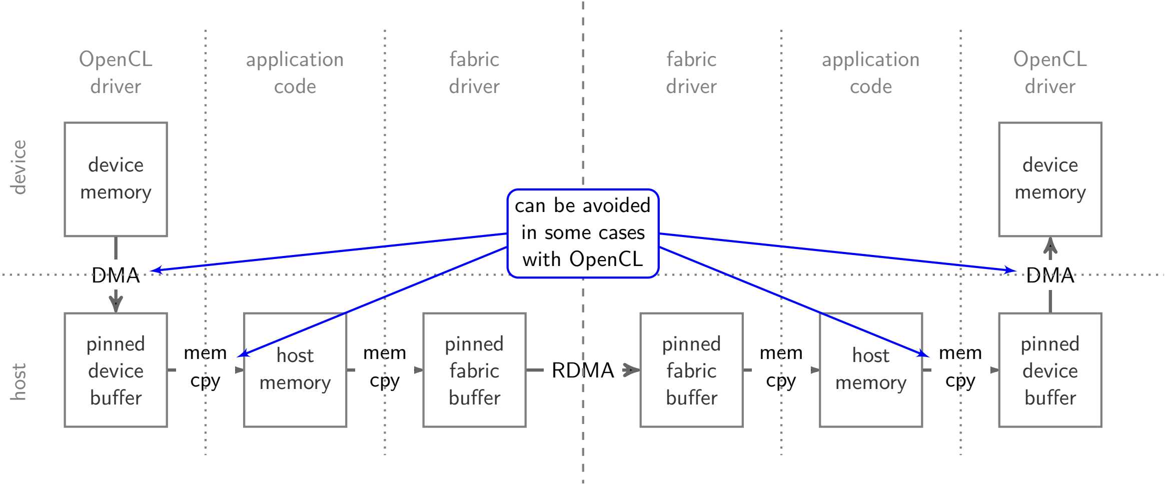
Current trade-offs:

- 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

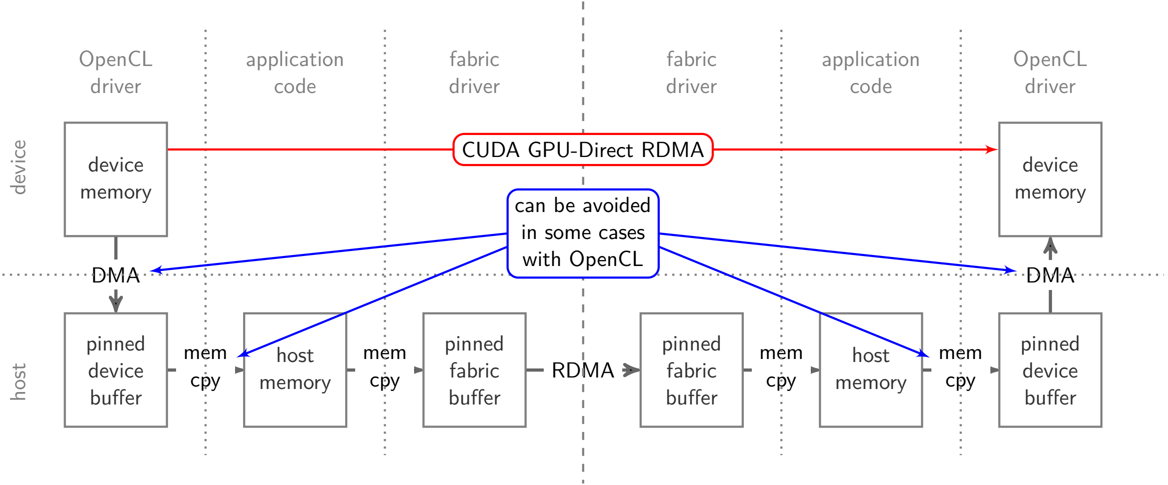
Data Transfer Paths



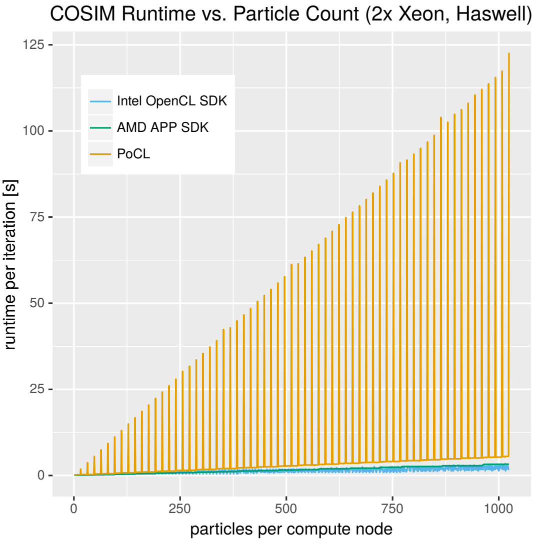
Data Transfer Paths



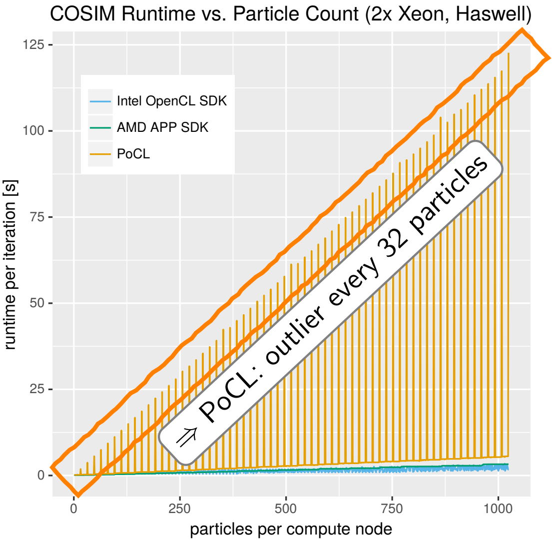
Data Transfer Paths



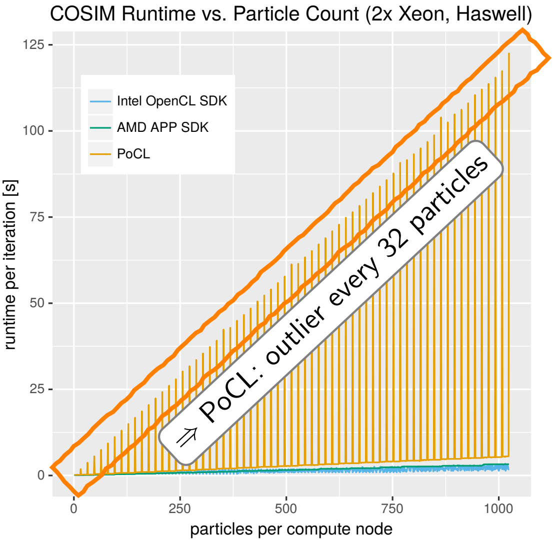
Benchmark Results: COSIM load imbalance (Xeon)

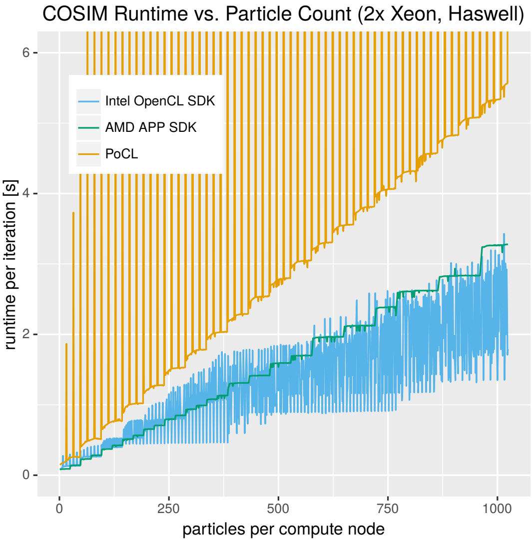


Benchmark Results: COSIM load imbalance (Xeon)

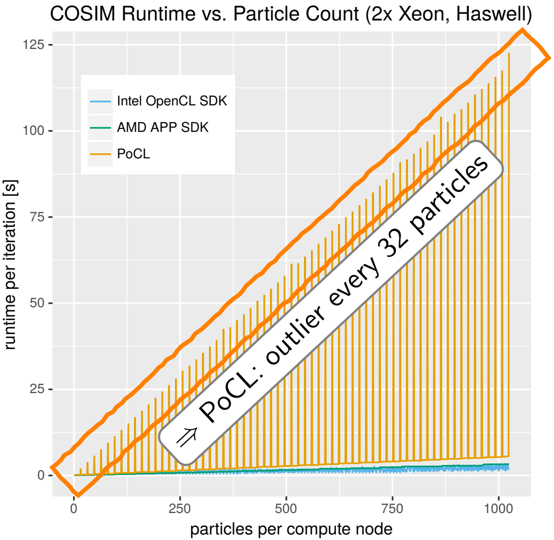


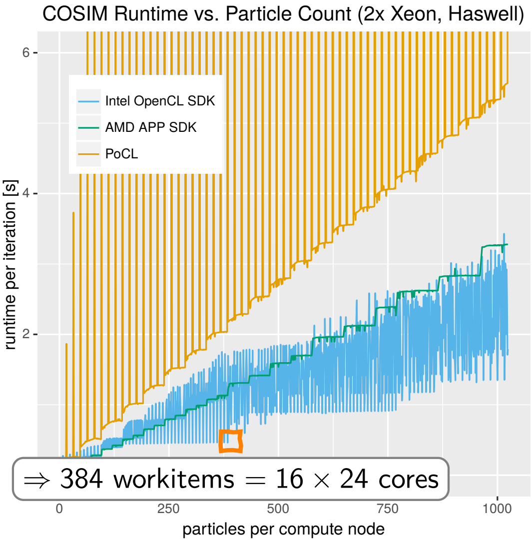
Benchmark Results: COSIM load imbalance (Xeon)





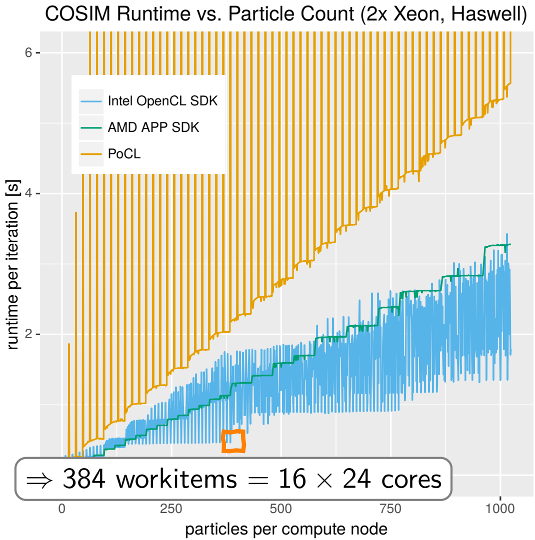
Benchmark Results: COSIM load imbalance (Xeon)





Benchmark Results: COSIM load imbalance (Xeon)

- different performance and characteristics across **OpenCL implementations**

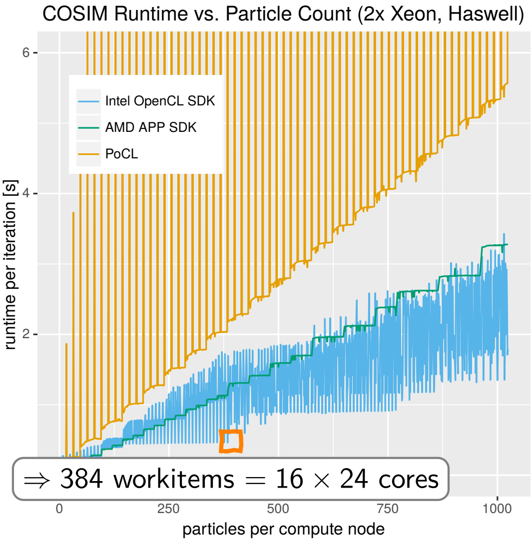


$\Rightarrow$ 384 workitems = 16 $\times$ 24 cores

Benchmark Results: COSIM load imbalance (Xeon)

- different performance and characteristics across **OpenCL implementations**
- highest **per-node-efficiency** with 384 **work-items per node** with Intel SDK

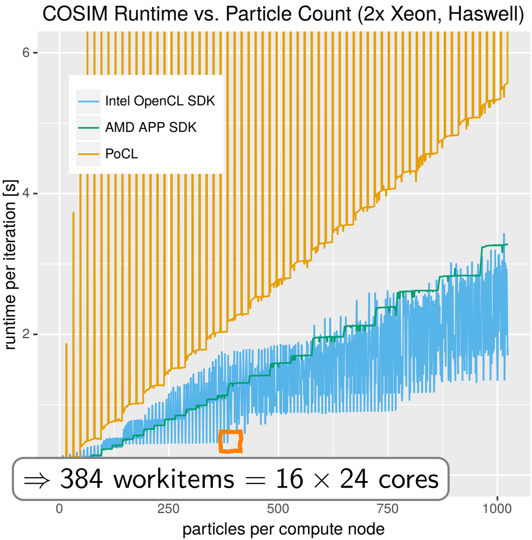
  ⇒ 16 = **logical SIMD-width** required by Intel OpenCL vectoriser



⇒ 384 workitems = 16 × 24 cores

Benchmark Results: COSIM load imbalance (Xeon)

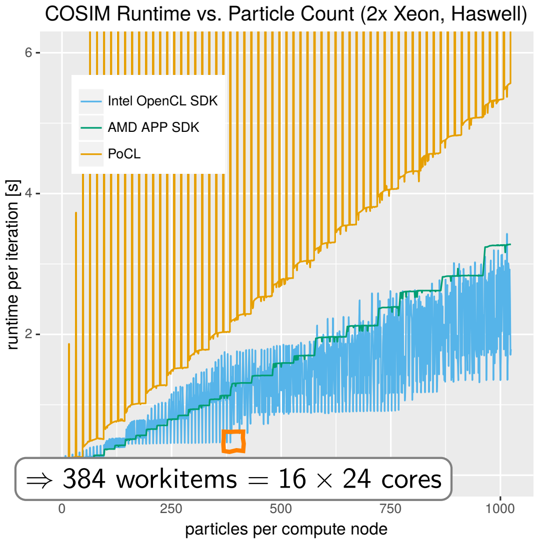
- different performance and characteristics across **OpenCL implementations**
- highest **per-node-efficiency** with 384 **work-items per node** with Intel SDK
  - ⇒ 16 = **logical SIMD-width** required by Intel OpenCL vectoriser
- **work-items per node** can dramatically affect **job runtime**
  - ⇒ ±1 **work-item** on a single node can more than double job runtime



Benchmark Results: COSIM load imbalance (Xeon)

- different performance and characteristics across **OpenCL implementations**
- highest **per-node-efficiency** with 384 **work-items per node** with Intel SDK
  - ⇒ 16 = **logical SIMD-width** required by Intel OpenCL vectoriser
- **work-items per node** can dramatically affect **job runtime**
  - ⇒ ±1 **work-item** on a single node can more than double job runtime

⇒ **benchmark**, adapt **job size**, **pad** work-items to $n \times 16$



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

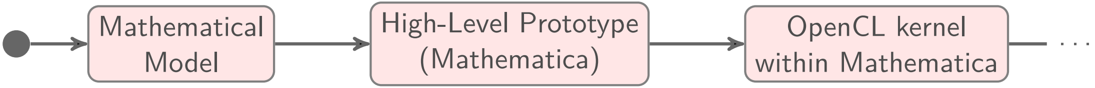
. . . → **C++ Host Application** → **Distributed Host Application**

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

- 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

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

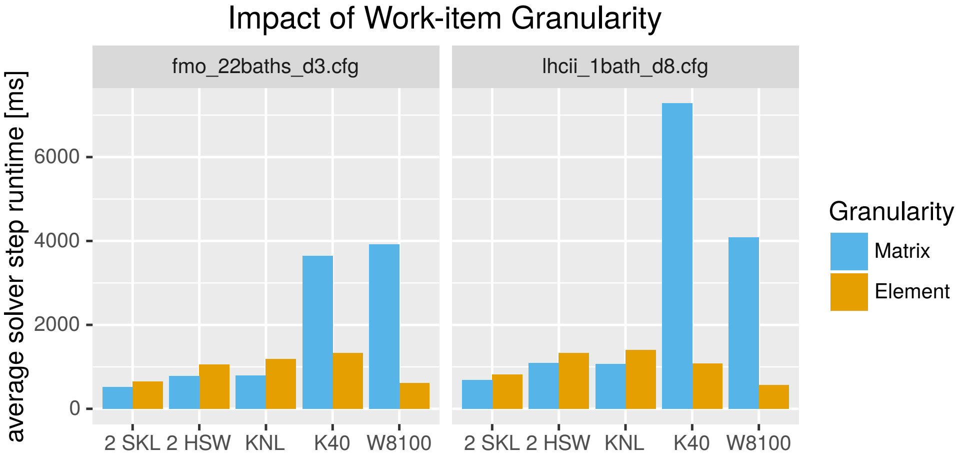
. . . → **C++ Host Application** → **Distributed Host Application** → **Optimisation / Production Runs**

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

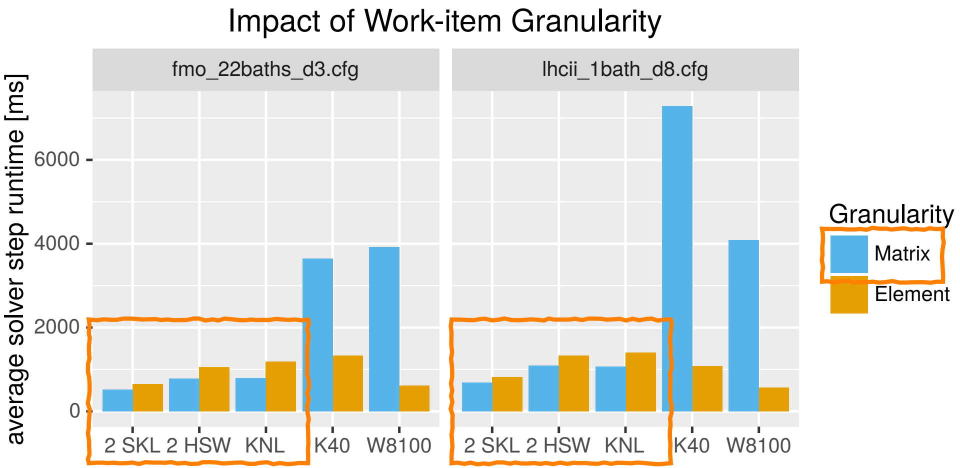
Distributed Host Application
- scale to multiple nodes
- partitioning, balancing, neighbour exchange, . . .
- wrapped MPI 3.0

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

DM-HEOM Benchmarks: Work-item Granularity

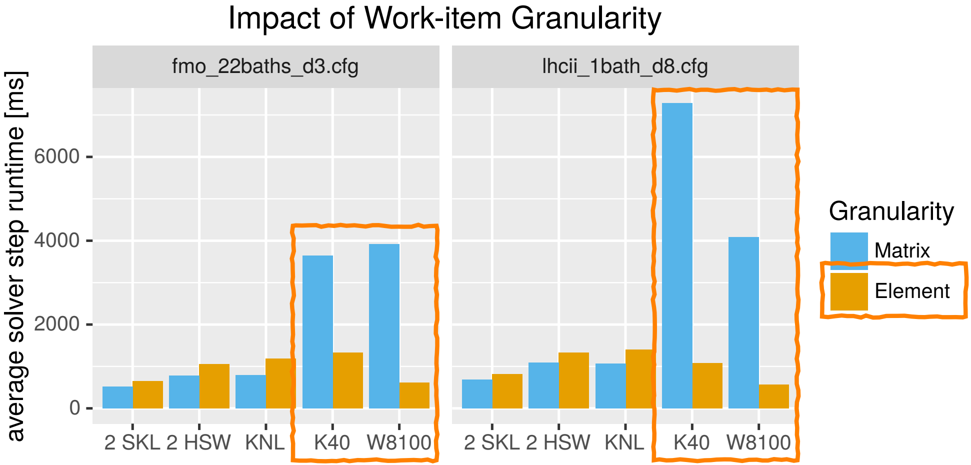


DM-HEOM Benchmarks: Work-item Granularity



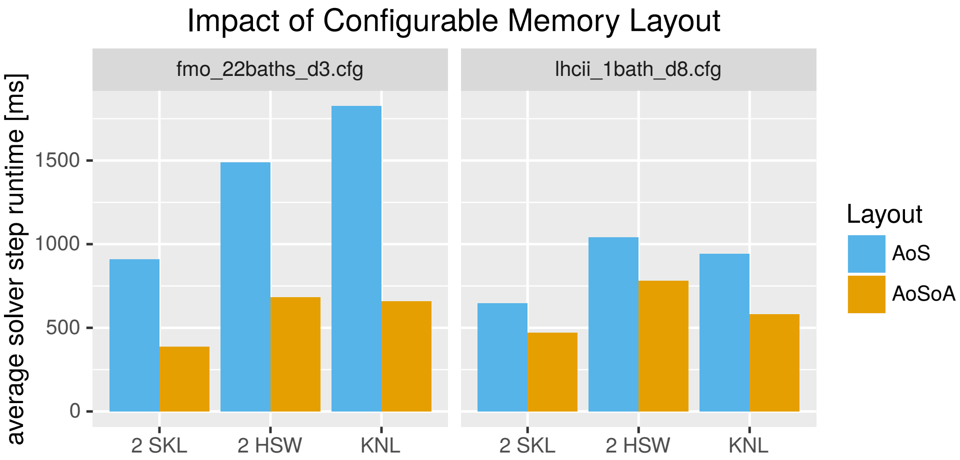
⇒ **CPUs**: 1.2× to 1.35× speedup for Matrix granularity

DM-HEOM Benchmarks: Work-item Granularity

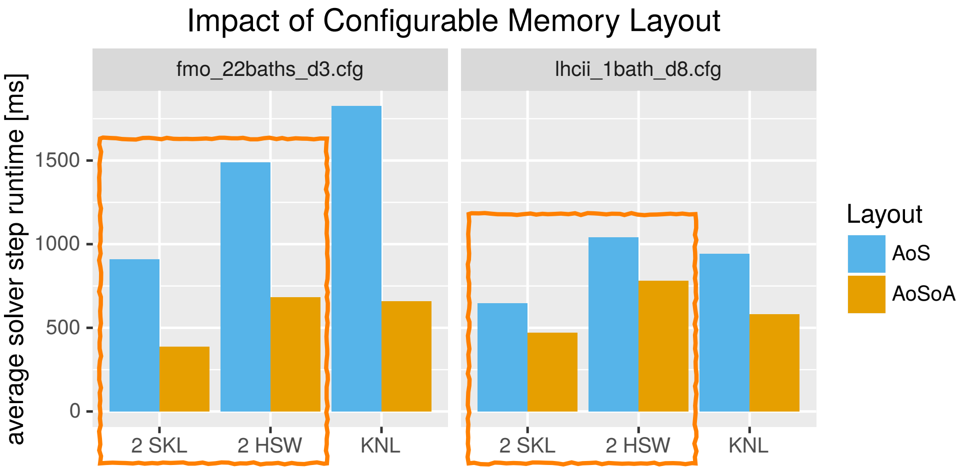


⇒ **GPUs**: up to 6.7× (K40) and 7.2× (W8100) speedup for Element granularity

DM-HEOM Benchmarks: Memory Layout

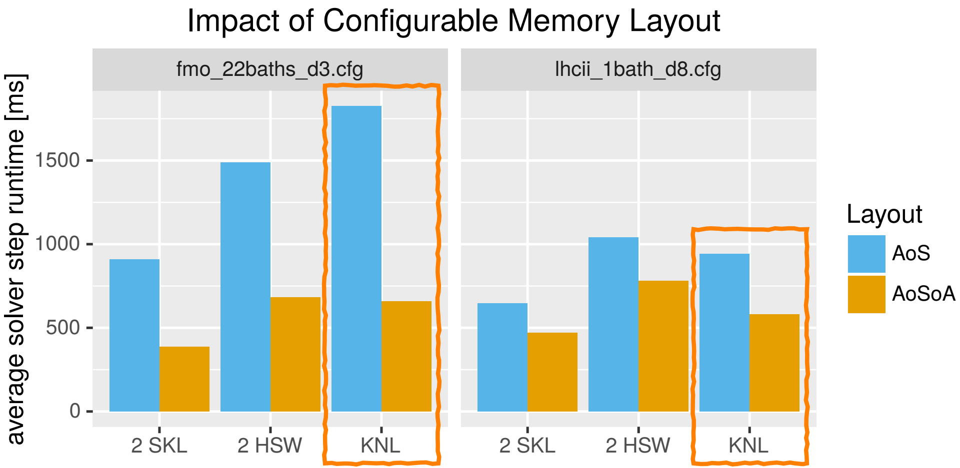


DM-HEOM Benchmarks: Memory Layout



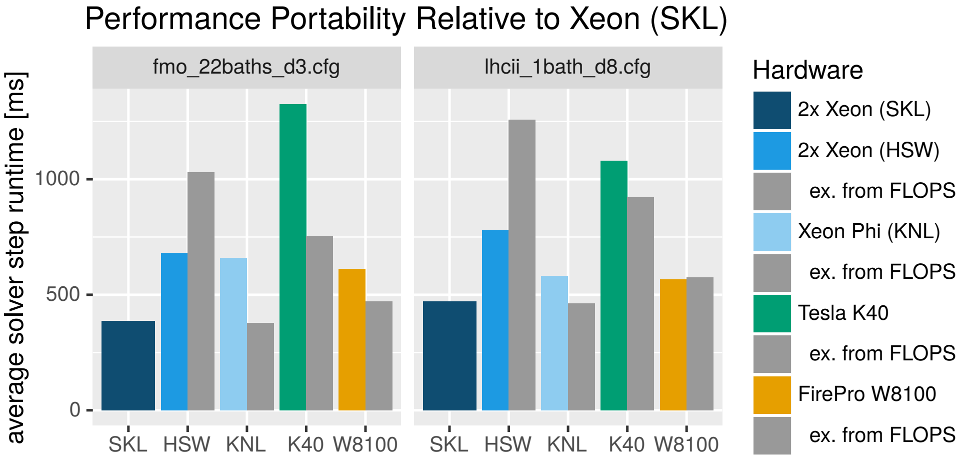
⇒ **SKL** and **HSW**: 1.3× to 2.4× speedup with AoSoA

DM-HEOM Benchmarks: Memory Layout

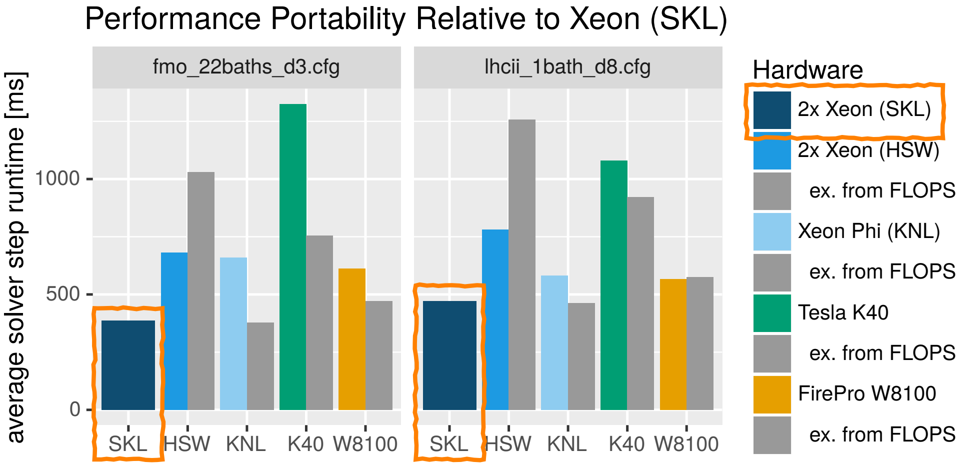


⇒ **KNL**: 1.6× to 2.8× speedup with AoSoA

DM-HEOM Benchmarks: Performance Portability

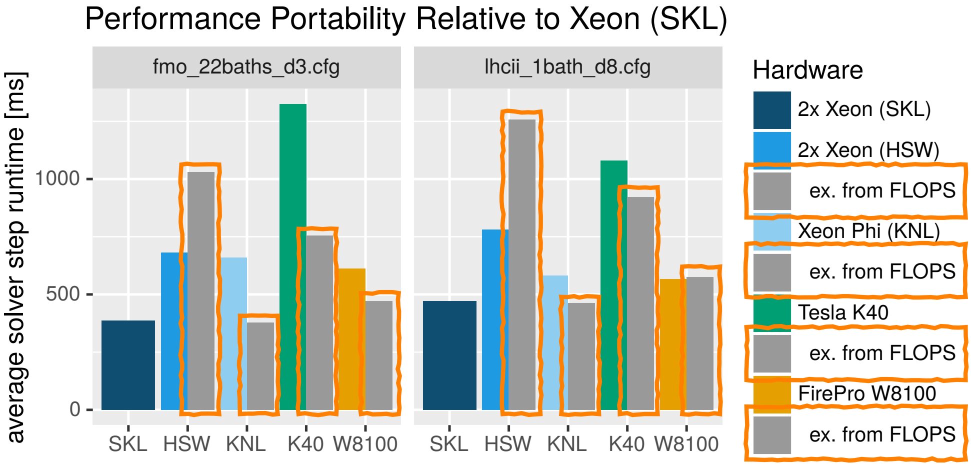


DM-HEOM Benchmarks: Performance Portability



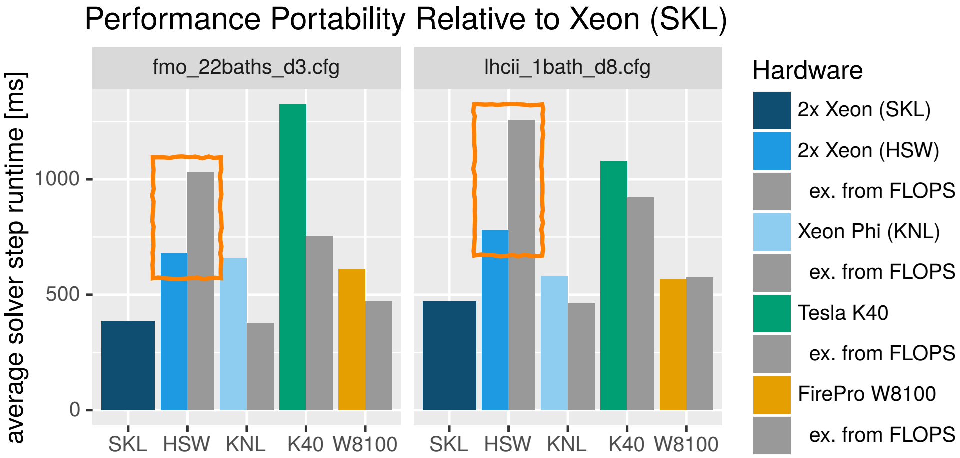
⇒ **SKL** (Xeon) is the **reference**

DM-HEOM Benchmarks: Performance Portability



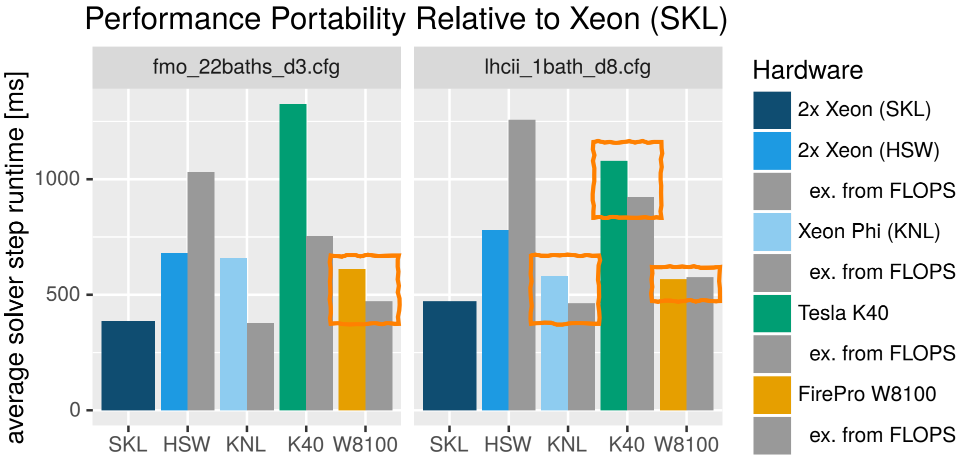
⇒ gray bars are **expected runtimes** extrapolated from peak FLOPS

DM-HEOM Benchmarks: Performance Portability



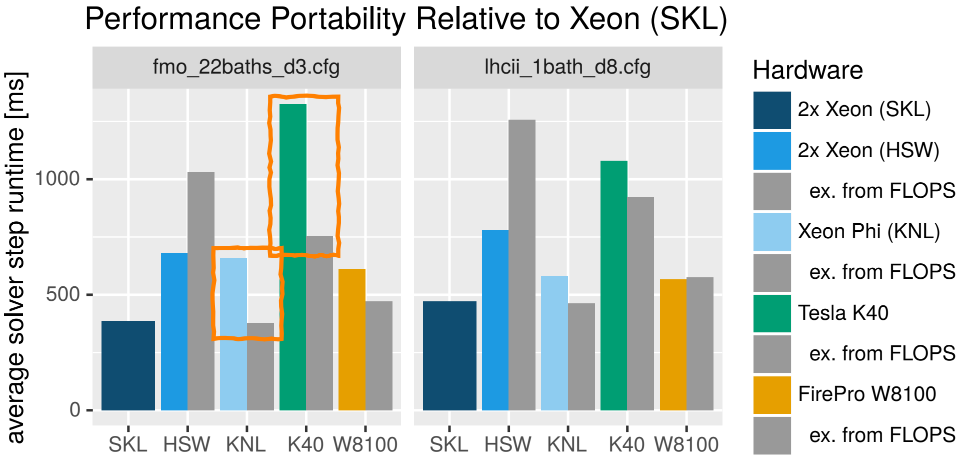
⇒ Older Haswell Xeon exceeds expectations, due to better OpenCL support

DM-HEOM Benchmarks: Performance Portability



⇒ Good: within 30 % of expectation

DM-HEOM Benchmarks: Performance Portability



⇒ KNL and K40 sensitive to **irregular accesses** from extreme coupling in this scenario.

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++ 11/14/17
- 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

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:

  $\Rightarrow$ e.g. via configurability of **work-item granularity** and **memory layout**

  $\Rightarrow$ worst case: **multiple kernels**, still **better than multiple programming models**

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:
  - ⇒ e.g. via configurability of **work-item granularity** and **memory layout**
  - ⇒ worst case: **multiple kernels**, still **better than multiple programming models**

Caveats

- **vendors** are slow in implementing new standards ⇒ use OpenCL 1.2 + **complain**
- interoperability with **communication** APIs not addressed by current standard

Thank you.

Feedback? Questions? Ideas?

noack@zib.de

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) project RE 1389/8-1 and the "Research Center for Many-Core HPC" at ZIB, an Intel Parallel Computing Center. The authors acknowledge the North-German Supercomputing Alliance (HLRN) for providing compute resources.