

# DM-HEOM: A Case Study in Performance Portability

Matthias Noack (noack@zib.de)



Zuse Institute Berlin

Distributed Algorithms and Supercomputing

# Motivation

## Observations:

- many HPC applications are **legacy code**
  - ⇒ written and incrementally improved  
**by domain scientists**
  - ⇒ often without (modern) software engineering



# Motivation

## Observations:

- many HPC applications are **legacy code**
    - ⇒ written and incrementally improved **by domain scientists**
    - ⇒ often without (modern) software engineering
  - hardware life time: **5 years** . . . increasing variety
  - software life time: **multiple tens of years** ⇒ outlives systems by far
- ⇒ need for **code modernisation**



# Motivation

## Observations:

- many HPC applications are **legacy code**
    - ⇒ written and incrementally improved **by domain scientists**
    - ⇒ often without (modern) software engineering
  - hardware life time: **5 years** . . . increasing variety
  - software life time: **multiple tens of years** ⇒ outlives systems by far
- ⇒ need for **code modernisation**



What can we do better?

# Motivation

## Observations:

- many HPC applications are **legacy code**
    - ⇒ written and incrementally improved **by domain scientists**
    - ⇒ often without (modern) software engineering
  - hardware life time: **5 years** . . . increasing variety
  - software life time: **multiple tens of years** ⇒ outlives systems by far
- ⇒ need for **code modernisation**



## What can we do better? – Our Suggestion:

- **portability first**, then portable performance

# Motivation

## Observations:

- many HPC applications are **legacy code**
    - ⇒ written and incrementally improved **by domain scientists**
    - ⇒ often without (modern) software engineering
  - hardware life time: **5 years** . . . increasing variety
  - software life time: **multiple tens of years** ⇒ outlives systems by far
- ⇒ need for **code modernisation**



## What can we do better? – Our Suggestion:

- **portability first**, then portable performance
- **interdisciplinary collaborations** where computer scientists design the software

# Motivation

## Observations:

- many HPC applications are **legacy code**
    - ⇒ written and incrementally improved **by domain scientists**
    - ⇒ often without (modern) software engineering
  - hardware life time: **5 years** . . . increasing variety
  - software life time: **multiple tens of years** ⇒ outlives systems by far
- ⇒ need for **code modernisation**



## What can we do better? – Our Suggestion:

- **portability first**, then portable performance
- **interdisciplinary collaborations** where computer scientists design the software
- **modern methods and technologies** with a community beyond HPC

# Contributions

a) 1st **D**istributed **M**emory implementation of the **HEOM** method: **DM-HEOM**



# Contributions

- a) 1st **D**istributed **M**emory implementation of the **HEOM** method: **DM-HEOM**
- b) **I**nterdisciplinary development workflow

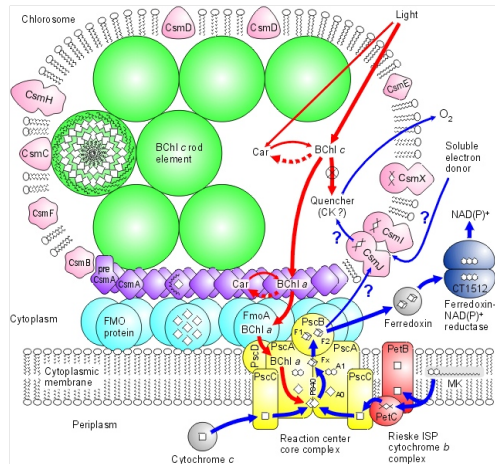
# Contributions

- a) 1st **D**istributed **M**emory implementation of the **HEOM** method: **DM-HEOM**
- b) **I**nterdisciplinary development workflow
- c) Design guidelines/experiences for **performance portable** HPC applications

# HEOM - Hierarchical Equations of Motion

## 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]

# HEOM - Hierarchical Equations of Motion

## 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

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

# HEOM - Hierarchical Equations of Motion

## Model for Open Quantum Systems

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

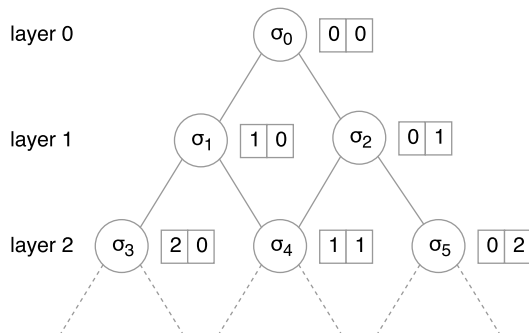
$$\begin{aligned}\frac{d\sigma_u}{dt} = & -\frac{i}{\hbar} [H, \sigma_u] && (\text{LvN commutator}) \\ & + \sum_{\text{baths}} A\sigma_u && (\text{same node}) \\ & + \sum_{\text{baths}} B\sigma_{u+} && (\text{links to layer+1}) \\ & + \sum_{\text{baths}} C\sigma_{u-} && (\text{links to layer-1})\end{aligned}$$

# HEOM - Hierarchical Equations of Motion

## 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  $d$**

$$\begin{aligned}\frac{d\sigma_u}{dt} = & -\frac{i}{\hbar} [H, \sigma_u] && (\text{LvN commutator}) \\ & + \sum_{\text{baths}} A\sigma_u && (\text{same node}) \\ & + \sum_{\text{baths}} B\sigma_{u+} && (\text{links to layer+1}) \\ & + \sum_{\text{baths}} C\sigma_{u-} && (\text{links to layer-1})\end{aligned}$$



## HEOM - Computational Properties

- **ODE:** dominated by commutator term:  $[H, \sigma_u] = H\sigma_u - \sigma_u H$

# HEOM - Computational Properties

- **ODE:** dominated by commutator term:  $[H, \sigma_u] = H\sigma_u - \sigma_u H$ 
    - for each  $\sigma_u$ :  $16N_{\text{sites}}^3$  FLOP per  $2 \cdot 2 \cdot 8 \cdot N_{\text{sites}}^2$  Byte
- ⇒ **arithmetic intensity:**  $\frac{N_{\text{sites}}}{2} \frac{\text{FLOP}}{\text{Byte}}$  ⇒ **compute bound** for larger  $N_{\text{sites}}$



# HEOM - Computational Properties

- **ODE**: dominated by commutator term:  $[H, \sigma_u] = H\sigma_u - \sigma_u H$ 
  - for each  $\sigma_u$ :  $16N_{\text{sites}}^3$  FLOP per  $2 \cdot 2 \cdot 8 \cdot N_{\text{sites}}^2$  Byte

$\Rightarrow$  **arithmetic intensity**:  $\frac{N_{\text{sites}}}{2} \frac{\text{FLOP}}{\text{Byte}} \Rightarrow$  **compute bound** for larger  $N_{\text{sites}}$
- **numerical integration** (weighted add):

# HEOM - Computational Properties

- **ODE**: dominated by commutator term:  $[H, \sigma_u] = H\sigma_u - \sigma_u H$ 
  - for each  $\sigma_u$ :  $16N_{\text{sites}}^3$  FLOP per  $2 \cdot 2 \cdot 8 \cdot N_{\text{sites}}^2$  Byte

⇒ **arithmetic intensity**:  $\frac{N_{\text{sites}}}{2} \frac{\text{FLOP}}{\text{Byte}} \Rightarrow$  **compute bound** for larger  $N_{\text{sites}}$
- **numerical integration** (weighted add):

⇒ **arithmetic intensity**:  $\frac{1}{8} \frac{\text{FLOP}}{\text{Byte}} \Rightarrow$  **memory bound** on relevant hardware

# HEOM - Computational Properties

- **ODE**: dominated by commutator term:  $[H, \sigma_u] = H\sigma_u - \sigma_u H$ 
  - for each  $\sigma_u$ :  $16N_{\text{sites}}^3$  FLOP per  $2 \cdot 2 \cdot 8 \cdot N_{\text{sites}}^2$  Byte
- ⇒ **arithmetic intensity**:  $\frac{N_{\text{sites}}}{2} \frac{\text{FLOP}}{\text{Byte}}$  ⇒ **compute bound** for larger  $N_{\text{sites}}$
- **numerical integration** (weighted add):
- ⇒ **arithmetic intensity**:  $\frac{1}{8} \frac{\text{FLOP}}{\text{Byte}}$  ⇒ **memory bound** on relevant hardware

Device Name (architecture)	compute [TFLOPS]	memory bw. [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 <sup>a</sup>	490/115 <sup>b</sup>	5.3/22.7 <sup>b</sup>
Nvidia Tesla K40 (Kepler)	1.31	480	2.7
AMD Firepro W8100 (Hawaii)	2.1	320	6.6

<sup>a</sup>Assuming 1.2 GHz AVX frequency. <sup>b</sup>On-chip MCDRAM / DRAM

## HEOM - Example Systems

system name	$N_{\text{sites}}$	baths per site	depth $d$	total ADO memory
FMO	7	22	8	2.8 GiB
LHC II monomer	14	1	3	5.6 GiB
PS I	<b>96</b>	1	<b>3</b>	<b>129.2 GiB</b>
PS I	<b>96</b>	1	<b>4</b>	<b>3231.0 GiB</b>

memory consumption assuming an RK4 solver

## HEOM - Example Systems

system name	$N_{\text{sites}}$	baths per site	depth $d$	total ADO memory
FMO	7	22	8	2.8 GiB
LHC II monomer	14	1	3	5.6 GiB
PS I	<b>96</b>	1	<b>3</b>	<b>129.2 GiB</b>
PS I	<b>96</b>	1	<b>4</b>	<b>3231.0 GiB</b>

memory consumption assuming an RK4 solver

⇒ larger systems **cannot** be solved on a single node:

- memory footprint
- time to solution

## HEOM - Example Systems

system name	$N_{\text{sites}}$	baths per site	depth $d$	total ADO memory
FMO	7	22	8	2.8 GiB
LHC II monomer	14	1	3	5.6 GiB
PS I	<b>96</b>	1	<b>3</b>	<b>129.2 GiB</b>
PS I	<b>96</b>	1	<b>4</b>	<b>3231.0 GiB</b>

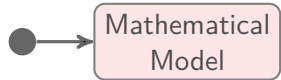
memory consumption assuming an RK4 solver

⇒ larger systems **cannot** be solved on a single node:

- memory footprint
- time to solution

⇒ distributed memory implementation **required**

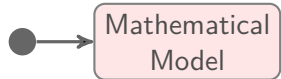
## Interdisciplinary Workflow



 domain experts  computer scientists

# Interdisciplinary Workflow

 domain experts  computer scientists

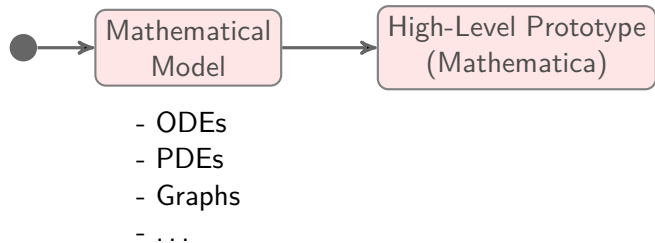


- ODEs
- PDEs
- Graphs
- ...



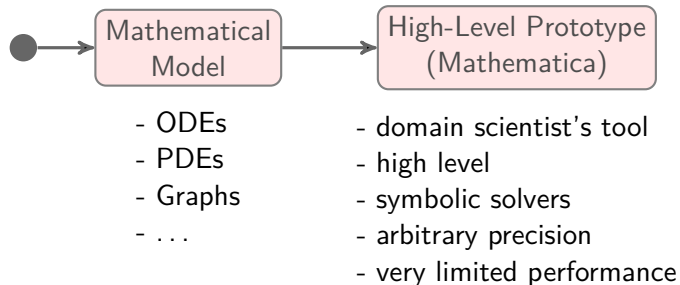
# Interdisciplinary Workflow

 domain experts  computer scientists



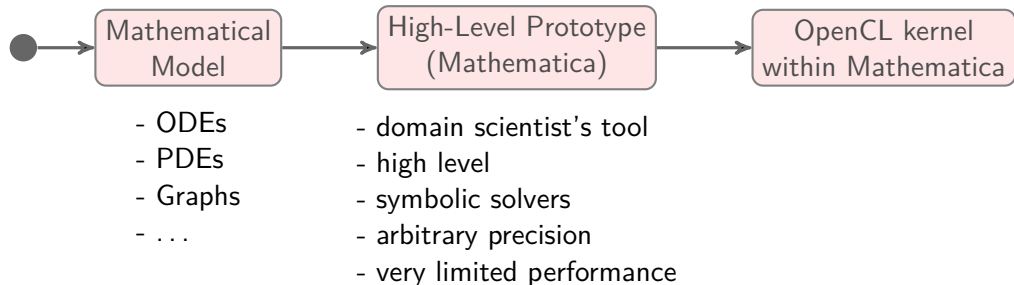
# Interdisciplinary Workflow

 domain experts  computer scientists



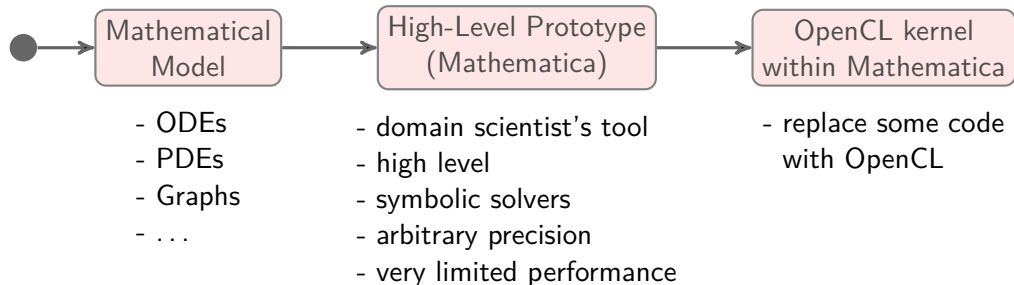
# Interdisciplinary Workflow

 domain experts  computer scientists



# Interdisciplinary Workflow

 domain experts  computer scientists



## 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**

## Implementers Desktop/Mobile/Embedded/FPGA



Single Source C++ Programming



Core API and Language Specs

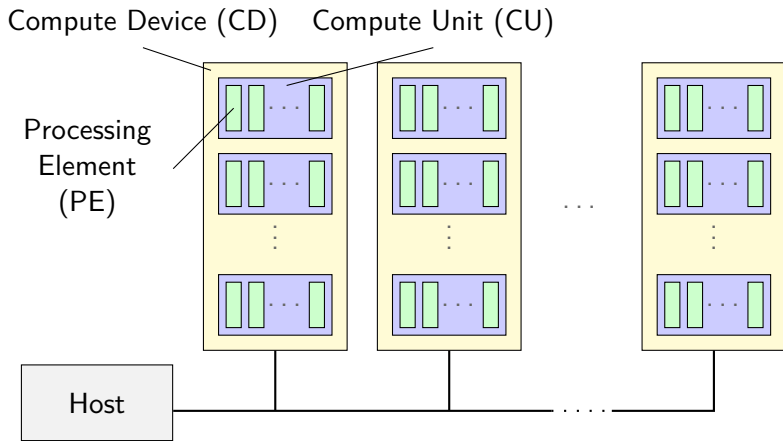


Portable Kernel Intermediate Language

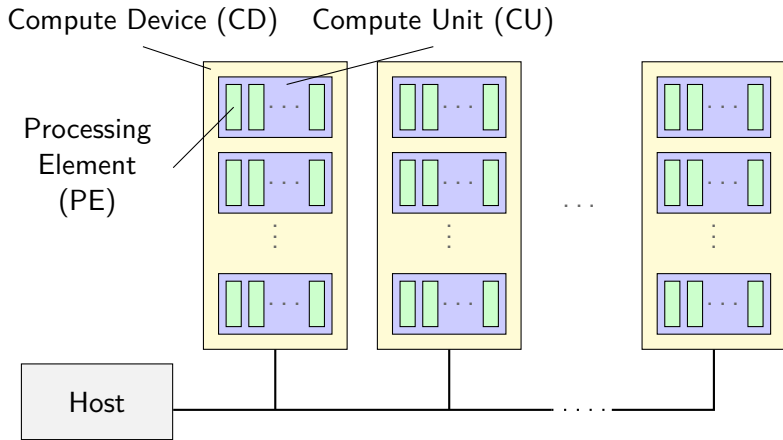
## Working Group Members Apps/Tools/Tests/Courseware



# OpenCL Platform and Memory Model



# OpenCL Platform and Memory Model



## Memory Model:

- CD has device memory with **global/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
<b>Compute Device</b>	Processor/Board	GPU device
<b>Compute Unit</b>	Core (thread)	Streaming MP
<b>Processing Element</b>	SIMD Lane	CUDA Core
<b>global/const. memory</b>	DRAM	DRAM
<b>local memory</b>	DRAM	Shared Memory
<b>private memory</b>	Register/DRAM	Priv. Mem./Register

## OpenCL Machine Model Mapping

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

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

## OpenCL Machine Model Mapping

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

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

## OpenCL Machine Model Mapping

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

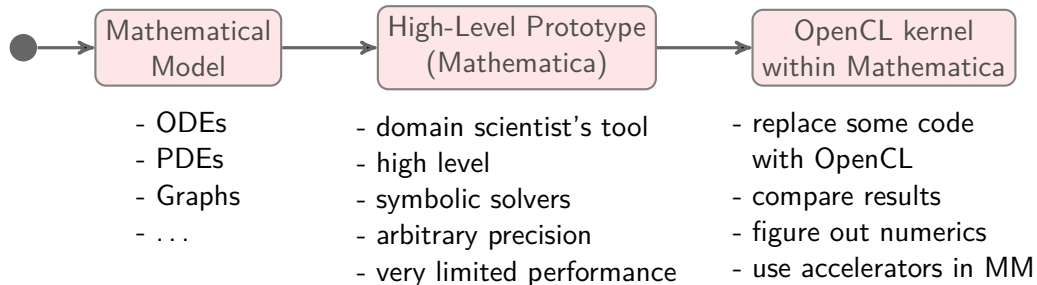
## OpenCL Machine Model Mapping

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

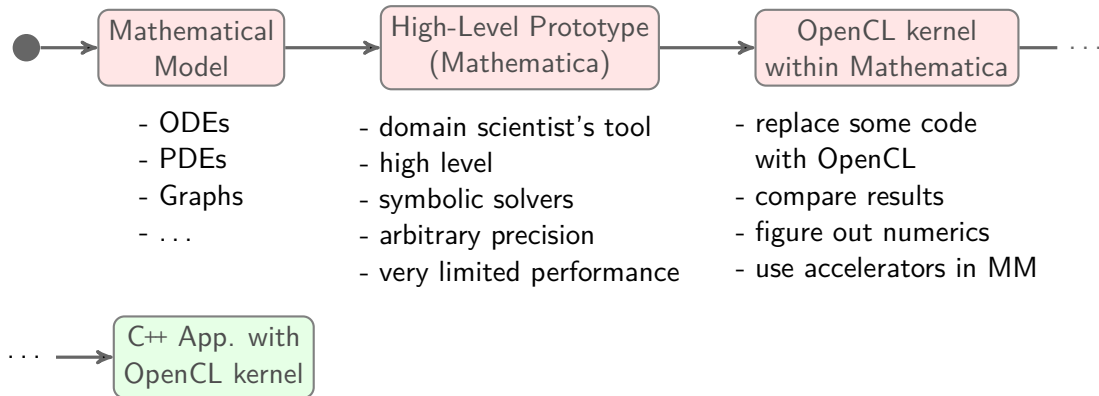
# Interdisciplinary Workflow

 domain experts  computer scientists



# Interdisciplinary Workflow

 domain experts  computer scientists



# DM-HEOM: Design Goals and Principles

- **portability**
  - modern, *vendor-independent* standards: C++11/14/17, OpenCL, MPI-3, CMake, ...



# DM-HEOM: Design Goals and Principles

- **portability**
  - modern, *vendor-independent* standards: C++11/14/17, OpenCL, MPI-3, CMake, ...
- **performance portability**
  - flexibility: make performance-critical aspects configurable
  - avoid device-specific code branches (i.e. redundant code)

# DM-HEOM: Design Goals and Principles

- **portability**
  - modern, *vendor-independent* standards: C++11/14/17, OpenCL, MPI-3, CMake, ...
- **performance portability**
  - flexibility: make performance-critical aspects configurable
  - avoid device-specific code branches (i.e. redundant code)
- **scalability**
  - from single nodes to supercomputers
  - a single code base for small and large problems

# DM-HEOM: Design Goals and Principles

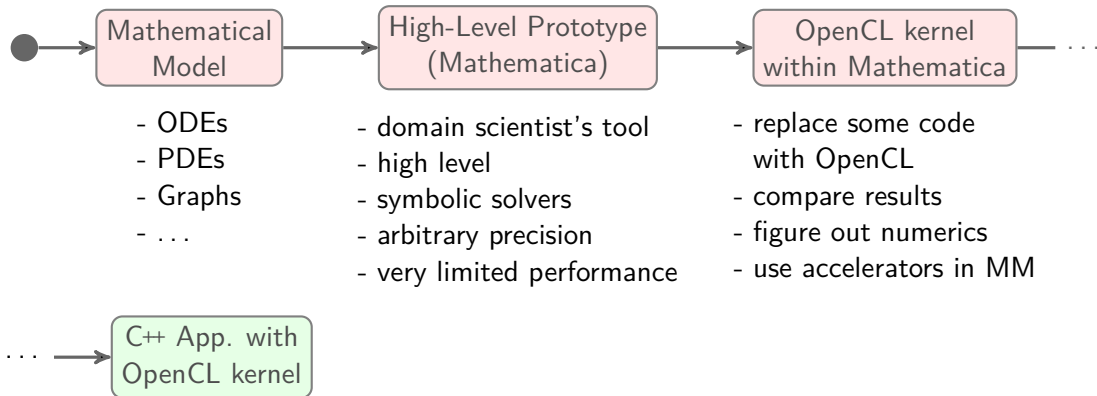
- **portability**
  - modern, *vendor-independent* standards: C++11/14/17, OpenCL, MPI-3, CMake, ...
- **performance portability**
  - flexibility: make performance-critical aspects configurable
  - avoid device-specific code branches (i.e. redundant code)
- **scalability**
  - from single nodes to supercomputers
  - a single code base for small and large problems
- powerful high-level **abstractions**
  - small applications on top
  - easily understood by domain scientists

# DM-HEOM: Design Goals and Principles

- **portability**
  - modern, *vendor-independent* standards: C++11/14/17, OpenCL, MPI-3, CMake, ...
- **performance portability**
  - flexibility: make performance-critical aspects configurable
  - avoid device-specific code branches (i.e. redundant code)
- **scalability**
  - from single nodes to supercomputers
  - a single code base for small and large problems
- powerful high-level **abstractions**
  - small applications on top
  - easily understood by domain scientists
- **separation and exchangeability** of different aspects and strategies
  - partitioning, numerical methods, memory layout, parallelisation, communication, etc.

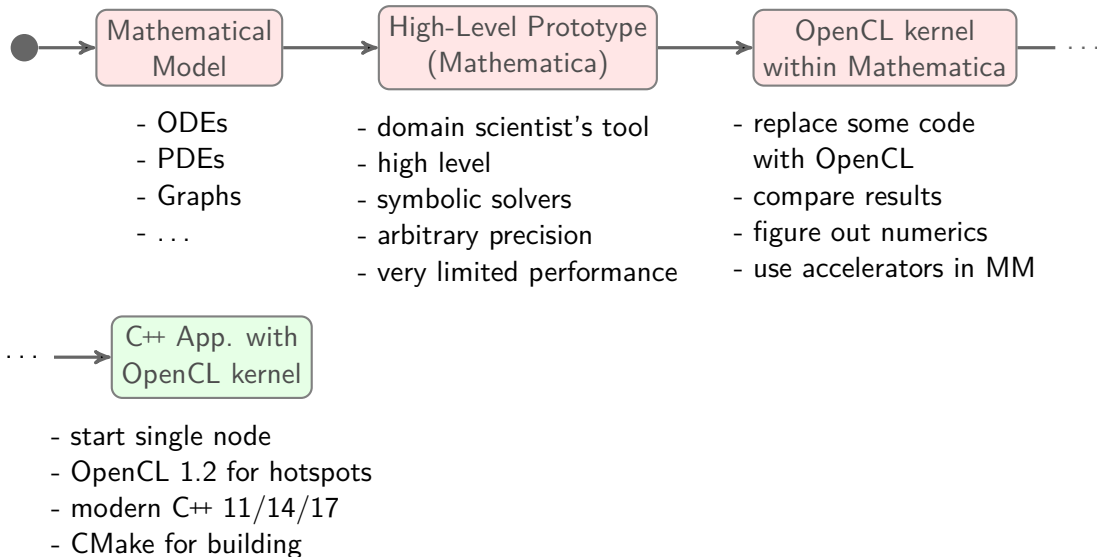
# Interdisciplinary Workflow

 domain experts  computer scientists

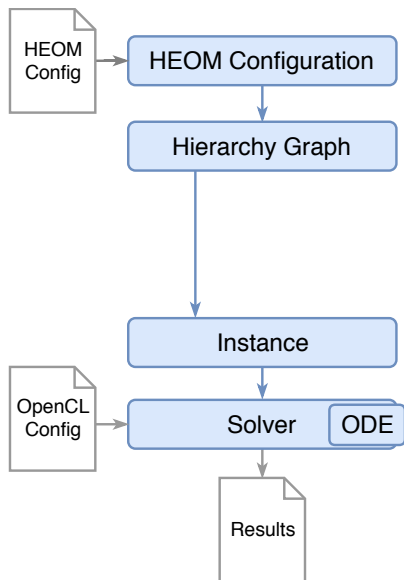


# Interdisciplinary Workflow

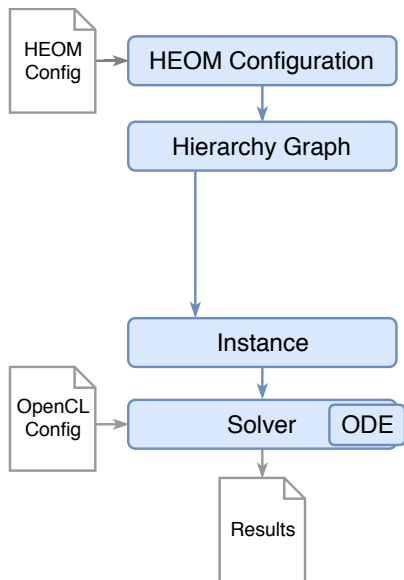
 domain experts  computer scientists



# Distributed Memory HEOM: Single Node C++ Application



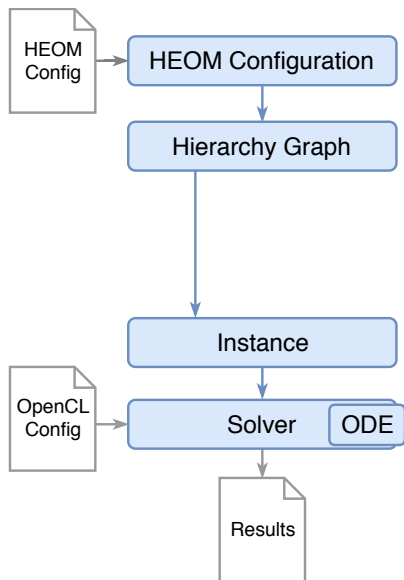
## Distributed Memory HEOM: Single Node C++ Application



- *HEOM Config* describes physics

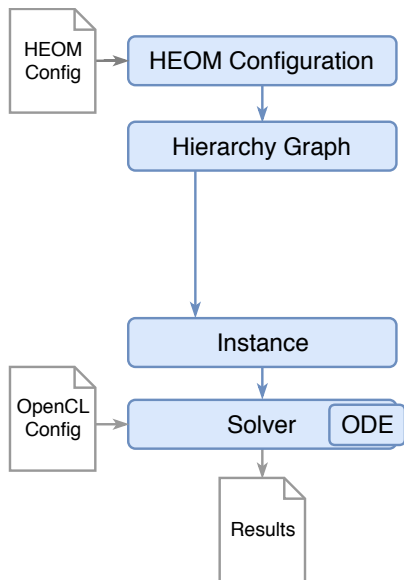


## Distributed Memory HEOM: Single Node C++ Application



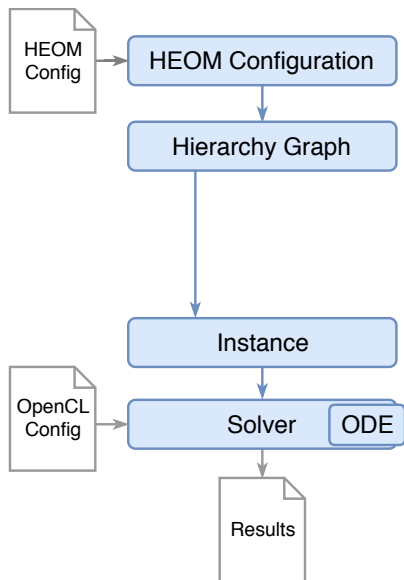
- *HEOM Config* describes physics
- *Hierarchy Graph* is used to initialise a problem *Instance*

## Distributed Memory HEOM: Single Node C++ Application



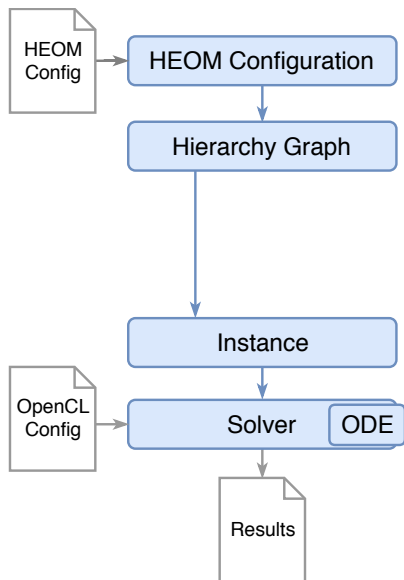
- *HEOM Config* describes physics
- *Hierarchy Graph* is used to initialise a problem *Instance*
- *ODE* (HEOM formula)
  - ⇒ encapsulates OpenCL kernel code

## Distributed Memory HEOM: Single Node C++ Application



- *HEOM Config* describes physics
- *Hierarchy Graph* is used to initialise a problem *Instance*
- *ODE* (HEOM formula)
  - ⇒ encapsulates OpenCL kernel code
- *OpenCL Config* specifies runtime configuration

# Distributed Memory HEOM: Single Node C++ Application



- *HEOM Config* describes physics
- *Hierarchy Graph* is used to initialise a problem *Instance*
- *ODE* (HEOM formula)
  - ⇒ encapsulates OpenCL kernel code
- *OpenCL Config* specifies runtime configuration
- *Solver* works on *ODE*
  - ⇒ encapsulates OpenCL runtime
  - ⇒ encapsulates numerics
  - ⇒ produces *Results*

# From Portability to Performance

## OpenCL:

- guarantees portability ...
- ... but no portable performance

# From Portability to Performance

## OpenCL:

- guarantees portability ...
- ... but no portable performance

## Strategy:

1. identify key optimisations each device requires
2. make the code configurable to the device's needs  
... without writing a version for each device

# Performance Portability: Node-Level

## OpenCL Runtime Kernel Compilation

- necessary for **portability**

# Performance Portability: Node-Level

## OpenCL Runtime Kernel Compilation

- necessary for **portability**
- exploitable for **performance**
  - a) facilitate compiler optimisation
  - b) configure code before compilation



# Performance Portability: Node-Level

## OpenCL Runtime Kernel Compilation

- necessary for **portability**
- exploitable for **performance**
  - a) facilitate compiler optimisation
  - b) configure code before compilation

### a) compiler optimisation:

- use host-code runtime-constants as kernel-code compile-time constants
  - e.g. sizes, loop-counts, ...
- ⇒ resolve index computations, eliminate branches, unroll loops, ...

# Performance Portability: Node-Level

## OpenCL Runtime Kernel Compilation

- necessary for **portability**
- exploitable for **performance**
  - a) facilitate compiler optimisation
  - b) configure code before compilation

### a) compiler optimisation:

- use host-code runtime-constants as kernel-code compile-time constants
  - e.g. sizes, loop-counts, ...
- ⇒ resolve index computations, eliminate branches, unroll loops, ...

### b) configurable kernel code:

- work-item granularity
- memory-layout

## Performance Portability: Work-item Granularity

- amount of work per OpenCL *work-item*
  - processed on the smallest parallel hardware execution unit (PE)
- ⇒ most important for efficient device utilisation

## Performance Portability: Work-item Granularity

- amount of work per OpenCL *work-item*
  - processed on the smallest parallel hardware execution unit (PE)
- ⇒ most important for efficient device utilisation

## GPU Devices

- many SIMT cores
  - thousands of light-weight hardware threads
  - executed in groups of 32 or 64 thread
- ⇒ **one ADO matrix element** per thread

## Performance Portability: Work-item Granularity

- amount of work per OpenCL *work-item*
  - processed on the smallest parallel hardware execution unit (PE)
- ⇒ most important for efficient device utilisation

### GPU Devices

- many SIMT cores
  - thousands of light-weight hardware threads
  - executed in groups of 32 or 64 thread
- ⇒ **one ADO matrix element** per thread

### CPU Devices

- fewer, more complex general-purpose cores
  - SIMD vector units with 4 to 8 lanes
- ⇒ **one ADO matrix** per processing element

## Performance Portability: Work-item Granularity

- amount of work per OpenCL *work-item*
  - processed on the smallest parallel hardware execution unit (PE)
- ⇒ most important for efficient device utilisation

### GPU Devices

- many SIMT cores
  - thousands of light-weight hardware threads
  - executed in groups of 32 or 64 thread
- ⇒ **one ADO matrix element** per thread

### CPU Devices

- fewer, more complex general-purpose cores
  - SIMD vector units with 4 to 8 lanes
- ⇒ **one ADO matrix** per processing element

⇒ requires a **compile-time configurable outer loop-nest** inside the kernel

# Performance Portability: Memory Layout

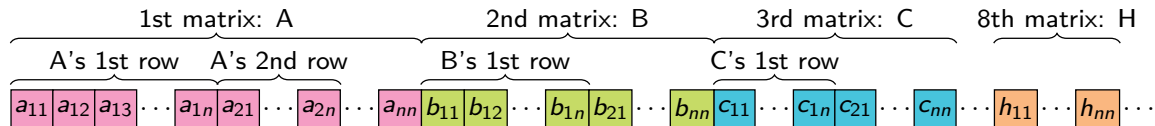
- ⇒ match requirement of **device-specific memory architecture**
- **GPU**: coalesced access from working-groups without bank-conflicts
  - **CPU**: **contiguous SIMD vector load/store** instructions (avoid gather/scatter ops)

# Performance Portability: Memory Layout

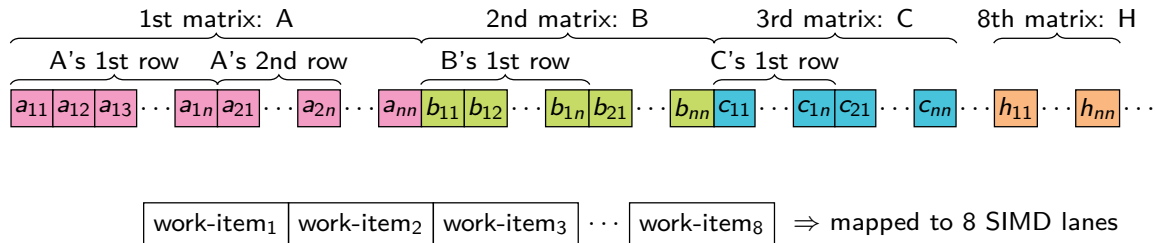
- ⇒ match requirement of **device-specific memory architecture**
  - **GPU**: coalesced access from working-groups without bank-conflicts
  - **CPU**: **contiguous SIMD vector load/store** instructions (avoid gather/scatter ops)
- ⇒ **parallelisation strategy, granularity** and **memory layout** must match
  - e.g. outer loop vectorisation for SIMD architectures



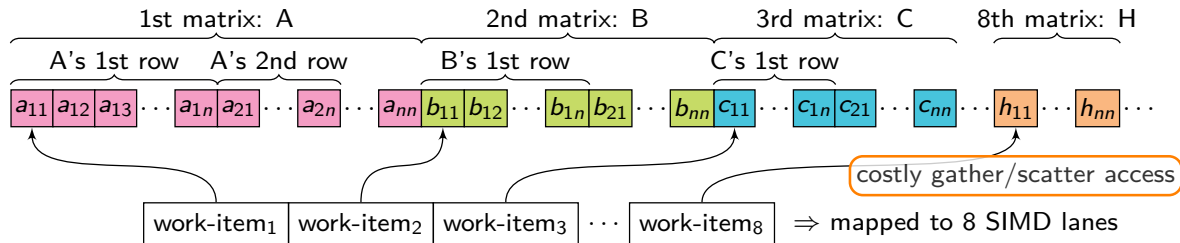
# AoS vs. SIMD-friendly AoSoA Memory Layout



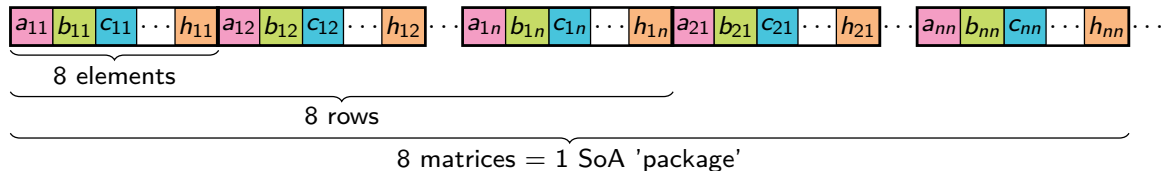
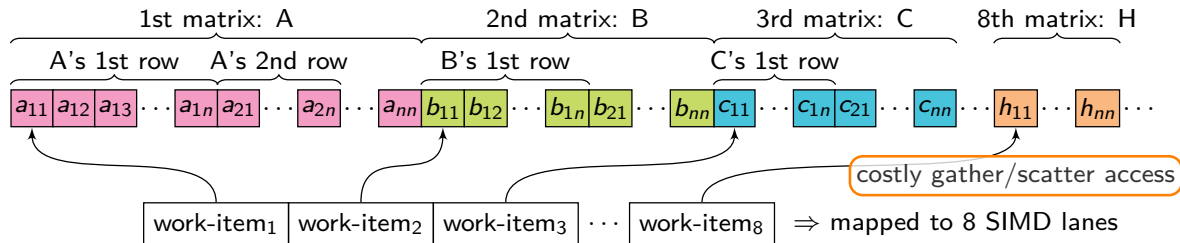
# AoS vs. SIMD-friendly AoSoA Memory Layout



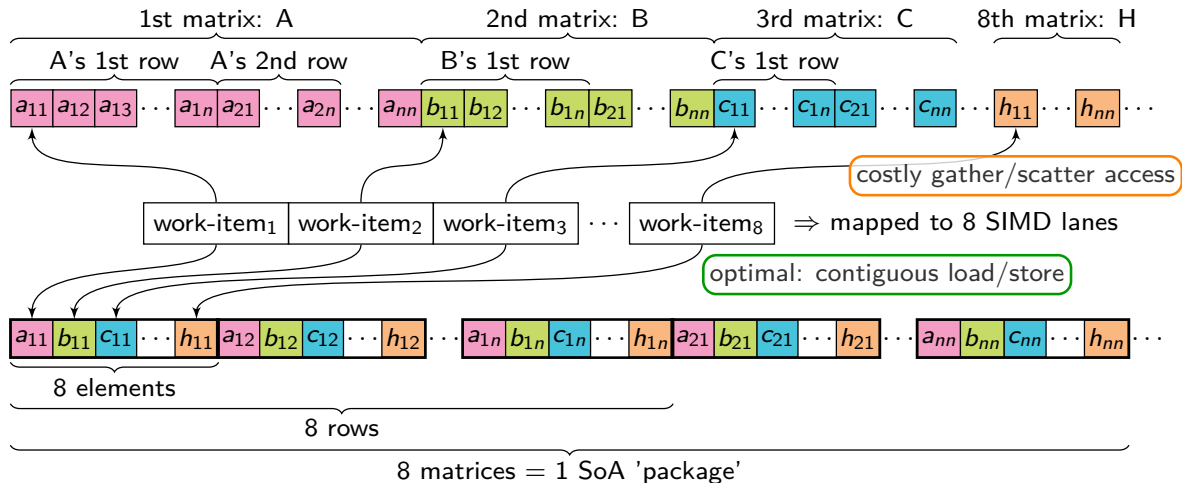
# AoS vs. SIMD-friendly AoSoA Memory Layout



# AoS vs. SIMD-friendly AoSoA Memory Layout

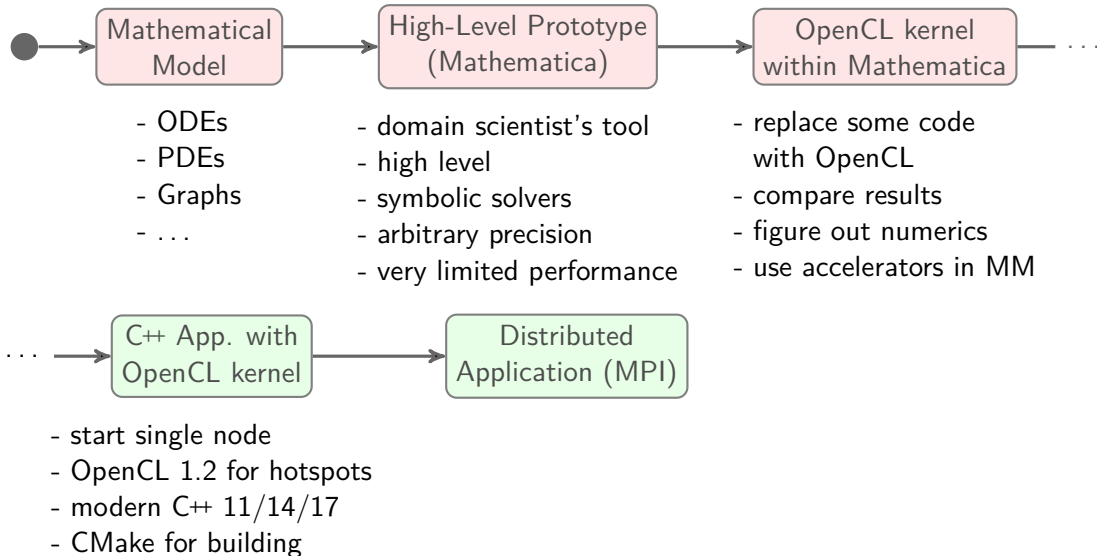


# AoS vs. SIMD-friendly AoSoA Memory Layout



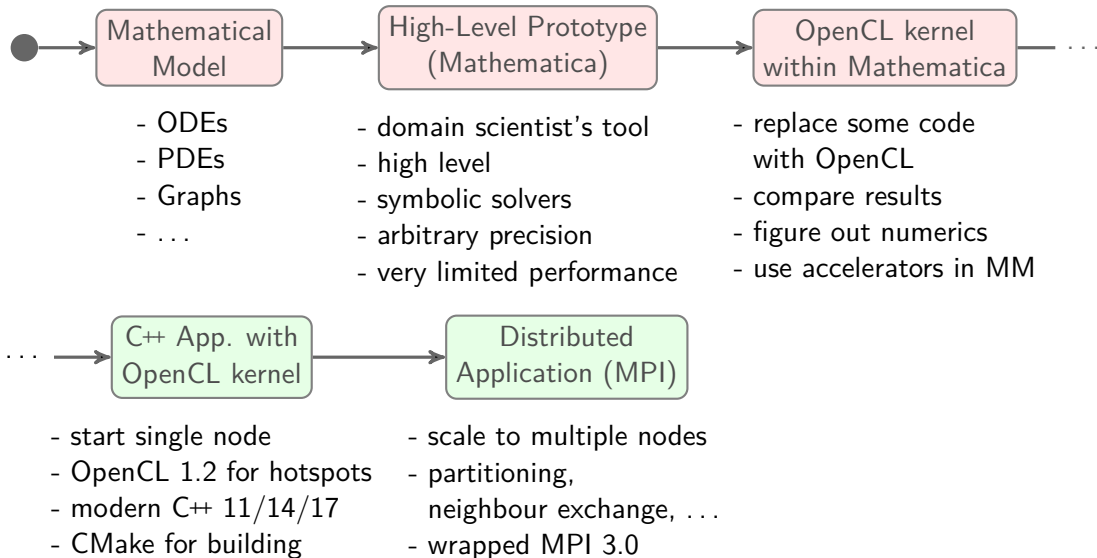
# Interdisciplinary Workflow

 domain experts  computer scientists



# Interdisciplinary Workflow

 domain experts  computer scientists



## Partitioning the Hierarchy

- **Problem:** map hierarchy graph nodes to  $n$  partitions (compute nodes)
  - minimise communication
  - minimise load imbalance

⇒ GP is **NP-hard**

⇒ hierarchy graph is highly connected

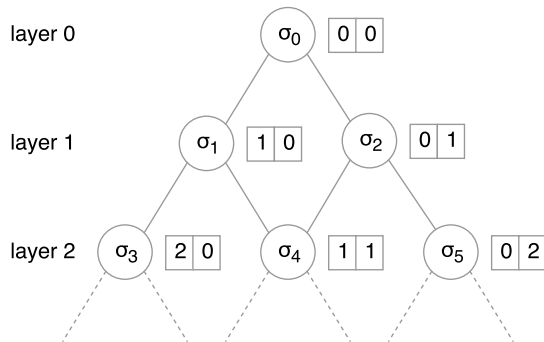


# Partitioning the Hierarchy

- **Problem:** map hierarchy graph nodes to  $n$  partitions (compute nodes)
  - minimise communication
  - minimise load imbalance

⇒ GP is **NP-hard**

⇒ hierarchy graph is highly connected

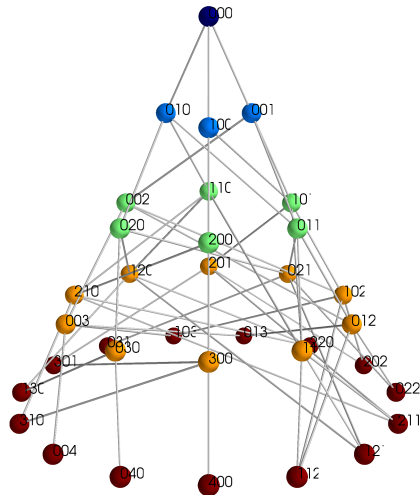


# Partitioning the Hierarchy

- **Problem:** map hierarchy graph nodes to  $n$  partitions (compute nodes)
  - minimise communication
  - minimise load imbalance

⇒ GP is **NP-hard**

⇒ hierarchy graph is highly connected

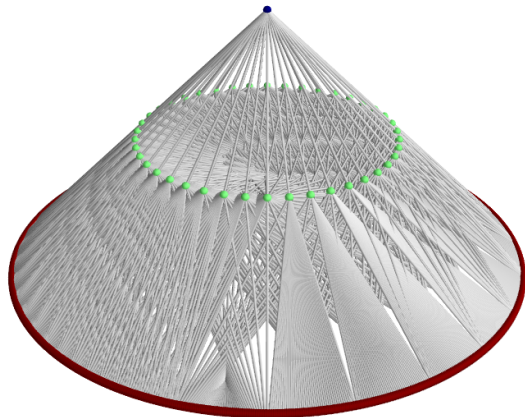


# Partitioning the Hierarchy

- **Problem:** map hierarchy graph nodes to  $n$  partitions (compute nodes)
  - minimise communication
  - minimise load imbalance

⇒ GP is **NP-hard**

⇒ hierarchy graph is highly connected



## Partitioning the Hierarchy (PS-I with $d = 3$ )

- **METIS** generated partitionings
  - partitioning quality mostly resilient to METIS settings

## Partitioning the Hierarchy (PS-I with $d = 3$ )

- **METIS** generated partitionings
  - partitioning quality mostly resilient to METIS settings

part- itions	neighbor parts.			hierarchy nodes per part. (avg)			
	<i>avg</i>	<i>min</i>	<i>max</i>	<i>nodes</i>	<i>shared %</i>	<i>halo</i>	<i>overhead %</i>
16	15	15	15	9803	91.7 %	12508	127.6 %
32	31	31	31	4902	95.0 %	7858	160.3 %
64	62	60	63	2451	96.8 %	4739	193.2 %
128	127	125	127	1225	98.0 %	2958	241.4 %
256	241	212	255	613	98.8 %	1689	274.8 %
512	281	124	414	306	99.0 %	929	303.3 %

## Partitioning the Hierarchy (PS-I with $d = 3$ )

- **METIS** generated partitionings
  - partitioning quality mostly resilient to METIS settings

part- itions	neighbor parts.			hierarchy nodes per part. (avg)			
	<i>avg</i>	<i>min</i>	<i>max</i>	<i>nodes</i>	<i>shared %</i>	<i>halo</i>	<i>overhead %</i>
16	15	15	15	9803	91.7 %	12508	127.6 %
32	31	31	31	4902	95.0 %	7858	160.3 %
64	62	60	63	2451	96.8 %	4739	193.2 %
128	127	125	127	1225	98.0 %	2958	241.4 %
256	241	212	255	613	98.8 %	1689	274.8 %
512	281	124	414	306	99.0 %	929	303.3 %

⇒ highly connected partitioning graph

⇒ almost **all-to-all**

## Partitioning the Hierarchy (PS-I with $d = 3$ )

- **METIS** generated partitionings
  - partitioning quality mostly resilient to METIS settings

part- itions	neighbor parts.			hierarchy nodes per part. (avg)			
	<i>avg</i>	<i>min</i>	<i>max</i>	<i>nodes</i>	<i>shared %</i>	<i>halo</i>	<i>overhead %</i>
16	15	15	15	9803	91.7 %	12508	127.6 %
32	31	31	31	4902	95.0 %	7858	160.3 %
64	62	60	63	2451	96.8 %	4739	193.2 %
128	127	125	127	1225	98.0 %	2958	241.4 %
256	241	212	255	613	98.8 %	1689	274.8 %
512	281	124	414	306	99.0 %	929	303.3 %

⇒ almost all nodes required by other partitions

⇒ prevents inner/outer communication/computation overlap

## Partitioning the Hierarchy (PS-I with $d = 3$ )

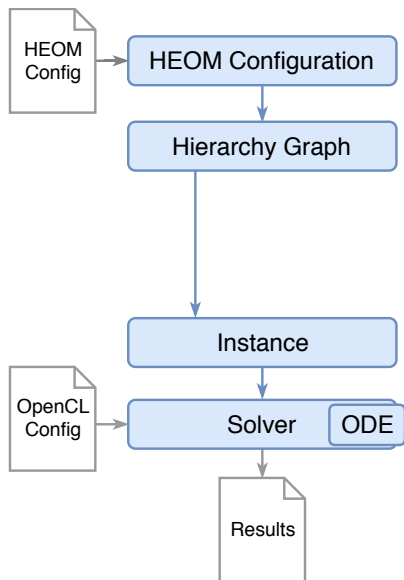
- **METIS** generated partitionings
  - partitioning quality mostly resilient to METIS settings

part- itions	neighbor parts.			hierarchy nodes per part. (avg)			
	<i>avg</i>	<i>min</i>	<i>max</i>	<i>nodes</i>	<i>shared %</i>	<i>halo</i>	<i>overhead %</i>
16	15	15	15	9803	91.7 %	12508	127.6 %
32	31	31	31	4902	95.0 %	7858	160.3 %
64	62	60	63	2451	96.8 %	4739	193.2 %
128	127	125	127	1225	98.0 %	2958	241.4 %
256	241	212	255	613	98.8 %	1689	274.8 %
512	281	124	414	306	99.0 %	929	303.3 %

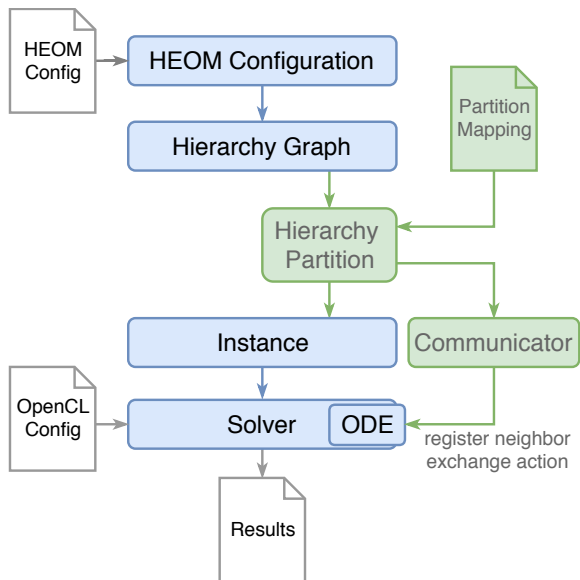
⇒ more halo nodes than local nodes



# Distributed Memory HEOM: Multi-Node C++ Application

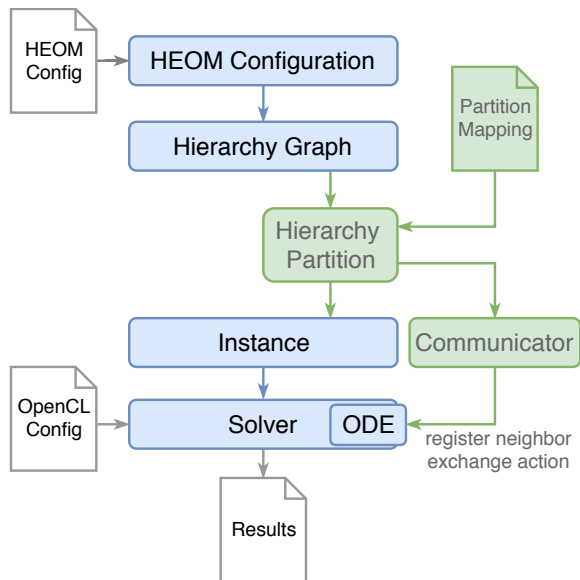


# Distributed Memory HEOM: Multi-Node C++ Application



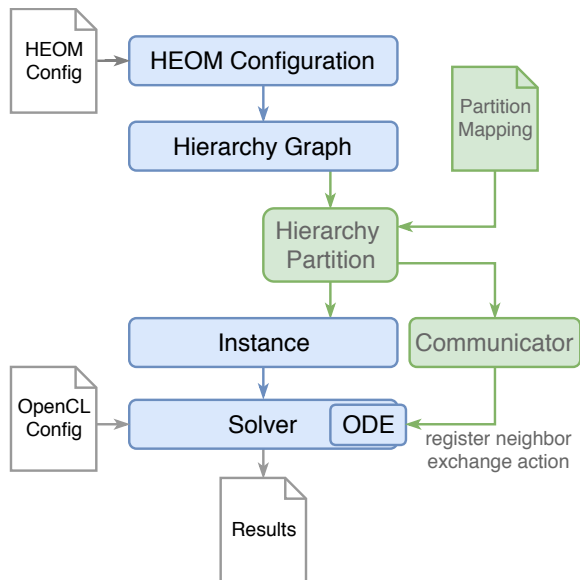
- simple transition to multi-node application

# Distributed Memory HEOM: Multi-Node C++ Application



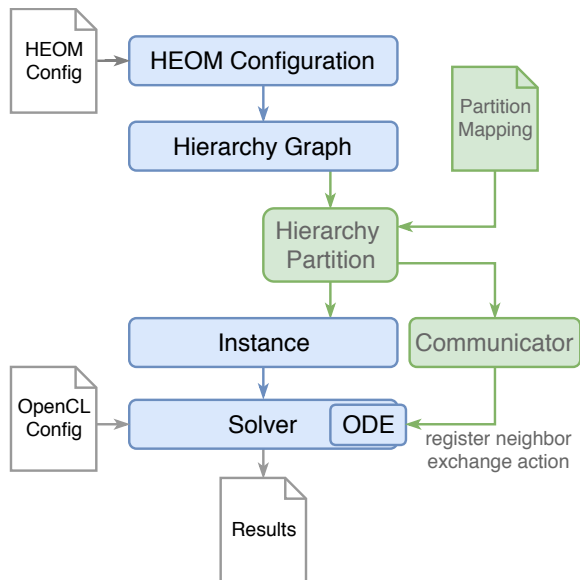
- simple transition to multi-node application
- *Partition Mapping* as additional input (METIS)

# Distributed Memory HEOM: Multi-Node C++ Application



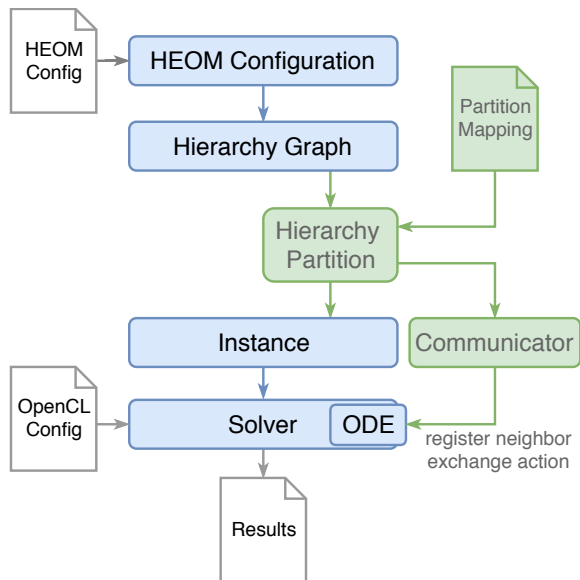
- simple transition to multi-node application
- *Partition Mapping* as additional input (METIS)
- generated *Hierarchy Partition* as new input for *Instance*
  - ⇒ Instance and Solver do not care

# Distributed Memory HEOM: Multi-Node C++ Application



- simple transition to multi-node application
- *Partition Mapping* as additional input (METIS)
- generated *Hierarchy Partition* as new input for *Instance*
  - ⇒ Instance and Solver do not care
- *Communicator* encapsulates MPI-3
  - Neighborhood Collectives
  - Derived Data Types
  - ⇒ fully **declarative** communication API

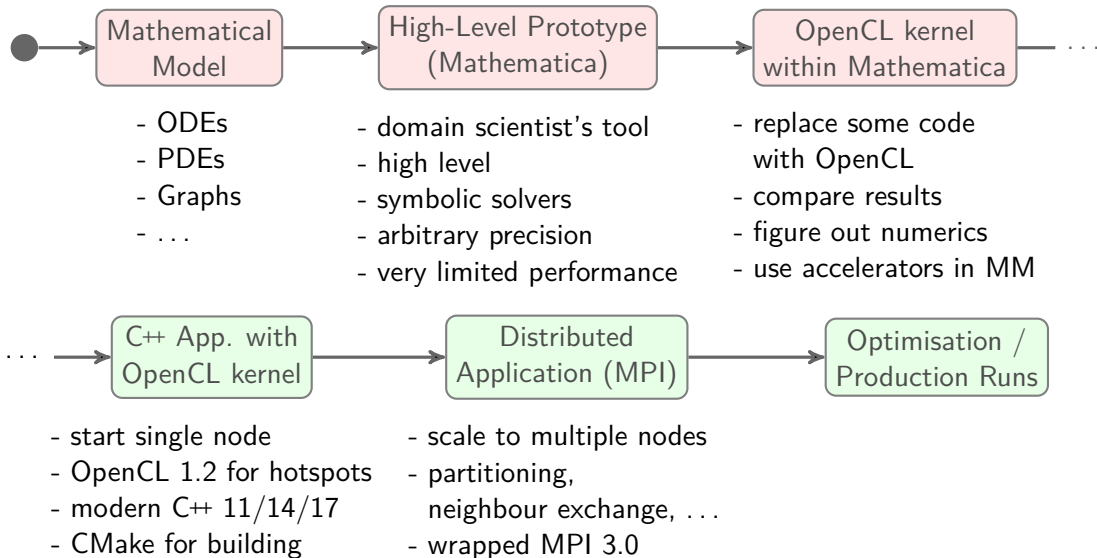
# Distributed Memory HEOM: Multi-Node C++ Application



- simple transition to multi-node application
- *Partition Mapping* as additional input (METIS)
- generated *Hierarchy Partition* as new input for *Instance*
  - ⇒ Instance and Solver do not care
- *Communicator* encapsulates MPI-3
  - Neighborhood Collectives
  - Derived Data Types
  - ⇒ fully **declarative** communication API
- *ODE* can trigger action
  - ⇒ neighbor exchange prior to evaluation

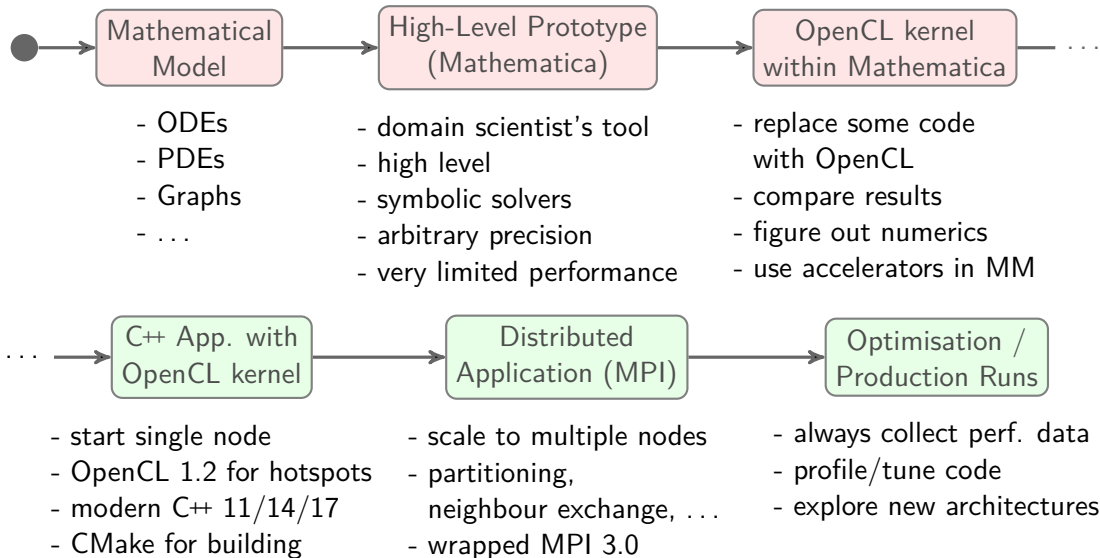
# Interdisciplinary Workflow

 domain experts  computer scientists



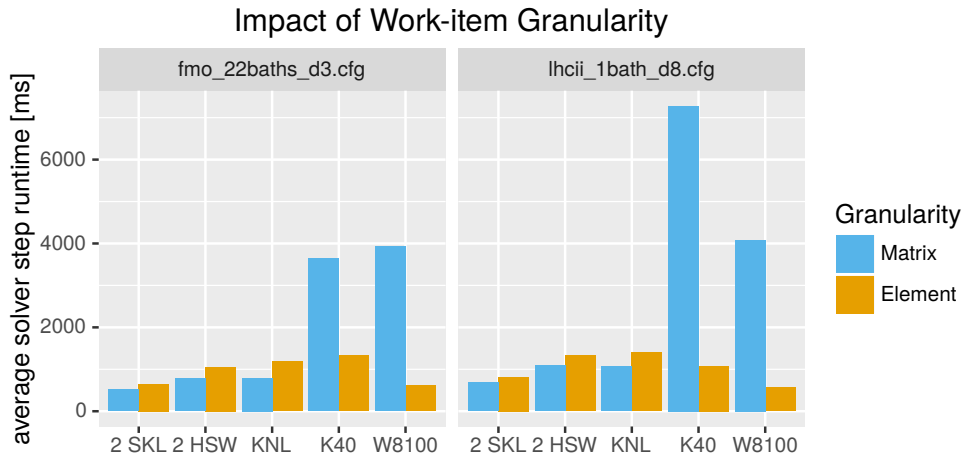
# Interdisciplinary Workflow

 domain experts  computer scientists

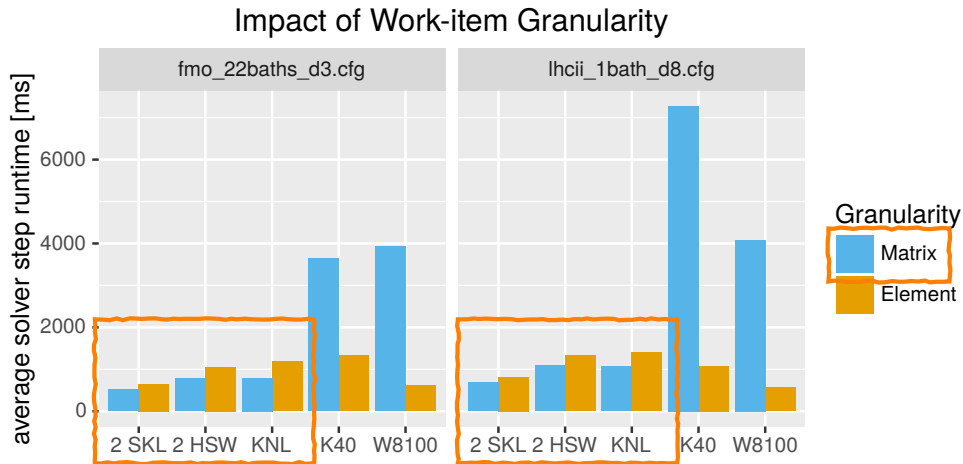




## Benchmarks: Work-item Granularity

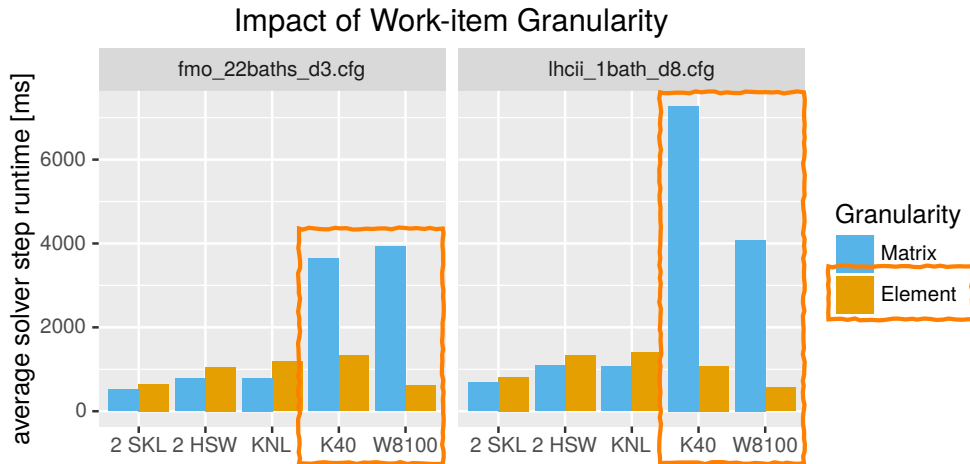


## Benchmarks: Work-item Granularity



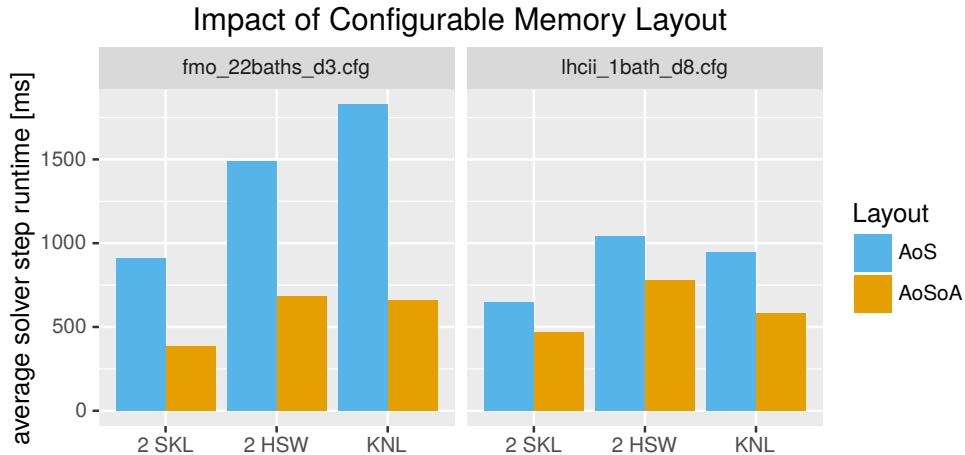
⇒ **CPUs:**  $1.2\times$  to  $1.35\times$  speedup for Matrix granularity

## Benchmarks: Work-item Granularity



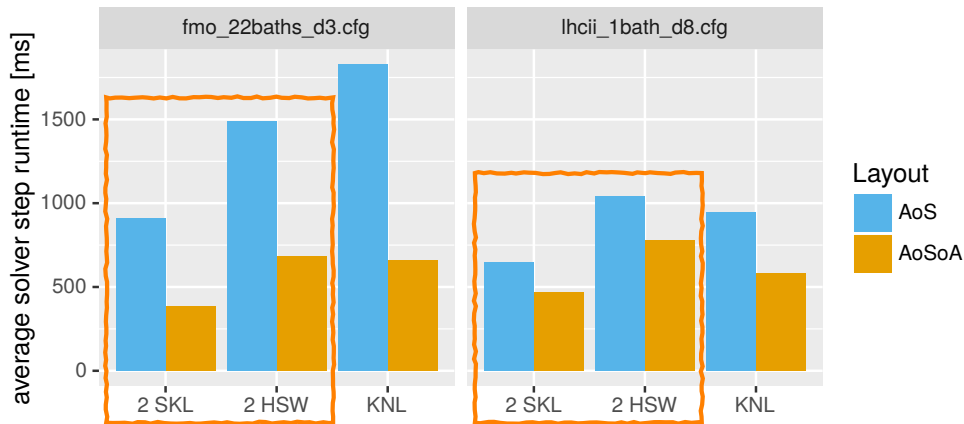
⇒ **GPUs:** up to  $6.7\times$  (K40) and  $7.2\times$  (W8100) speedup for Element granularity

## Benchmarks: Memory Layout



## Benchmarks: Memory Layout

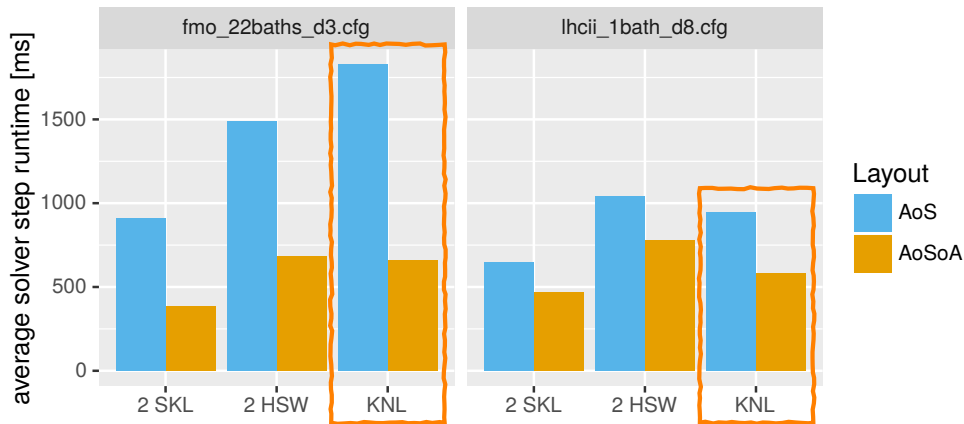
### Impact of Configurable Memory Layout



⇒ **SKL** and **HSW**:  $1.3\times$  to  $2.4\times$  speedup with AoSoA

## Benchmarks: Memory Layout

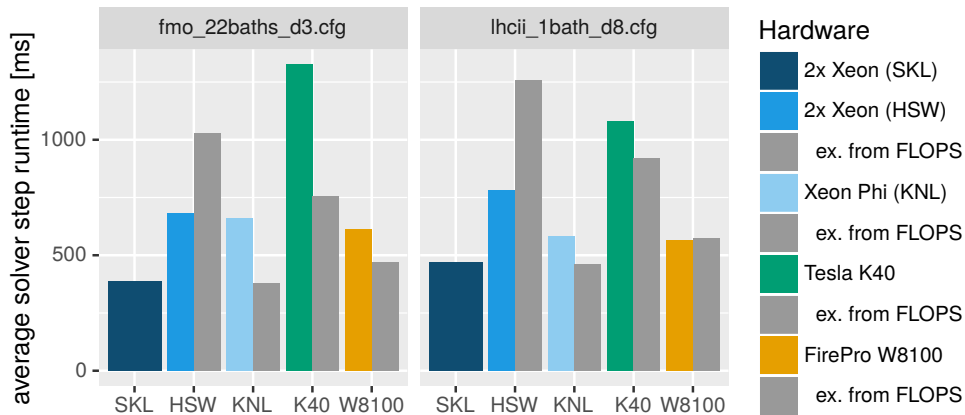
Impact of Configurable Memory Layout



⇒ **KNL**:  $1.6\times$  to  $2.8\times$  speedup with AoSoA

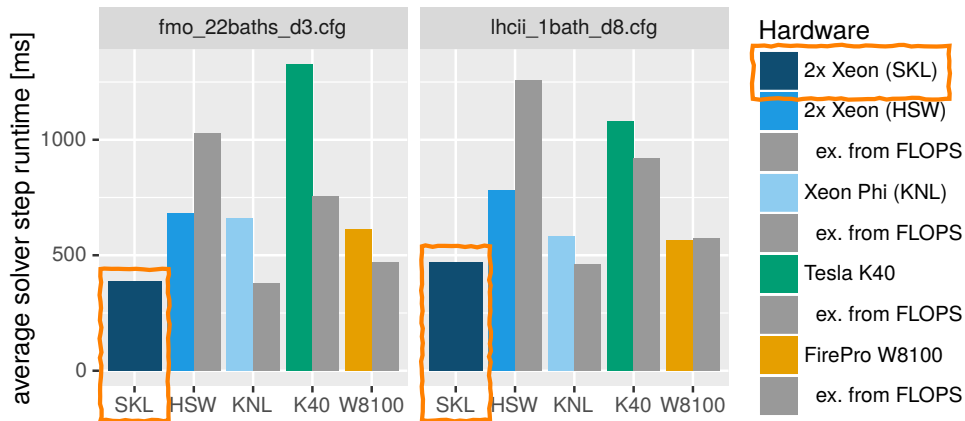
# Benchmarks: Performance Portability

## Performance Portability Relative to Xeon (SKL)



# Benchmarks: Performance Portability

## Performance Portability Relative to Xeon (SKL)

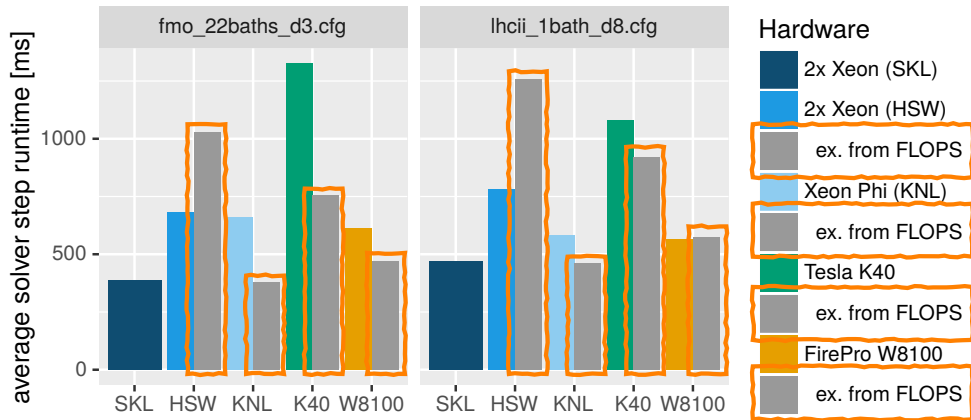


⇒ **SKL** (Xeon) is the **reference**



# Benchmarks: Performance Portability

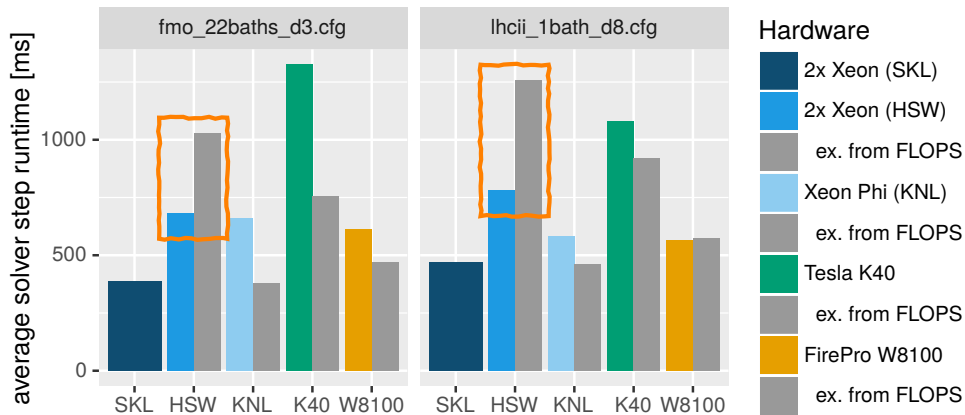
## Performance Portability Relative to Xeon (SKL)



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

## Benchmarks: Performance Portability

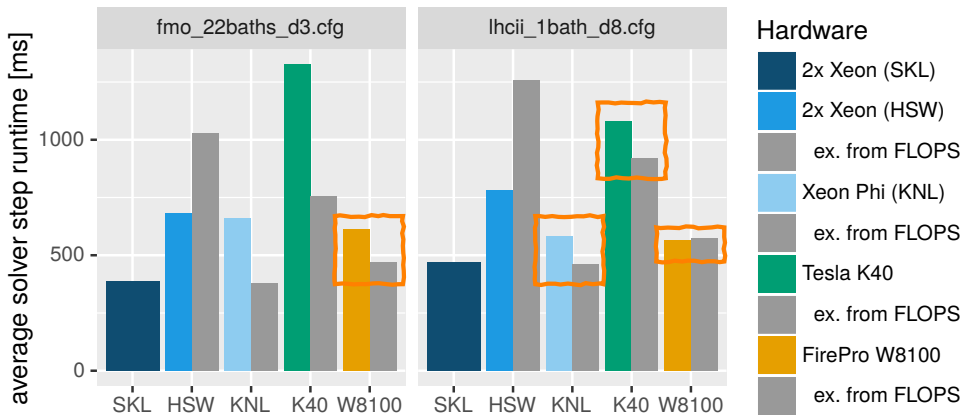
Performance Portability Relative to Xeon (SKL)



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

# Benchmarks: Performance Portability

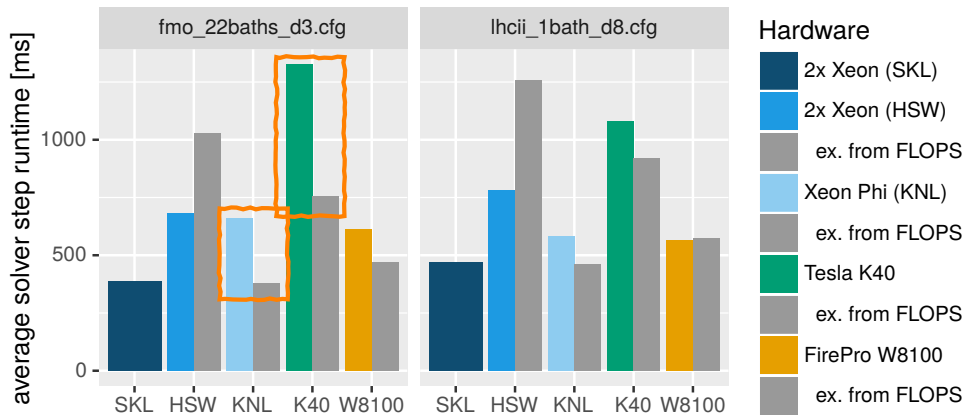
## Performance Portability Relative to Xeon (SKL)



⇒ Good: within 30 % of expectation

## Benchmarks: Performance Portability

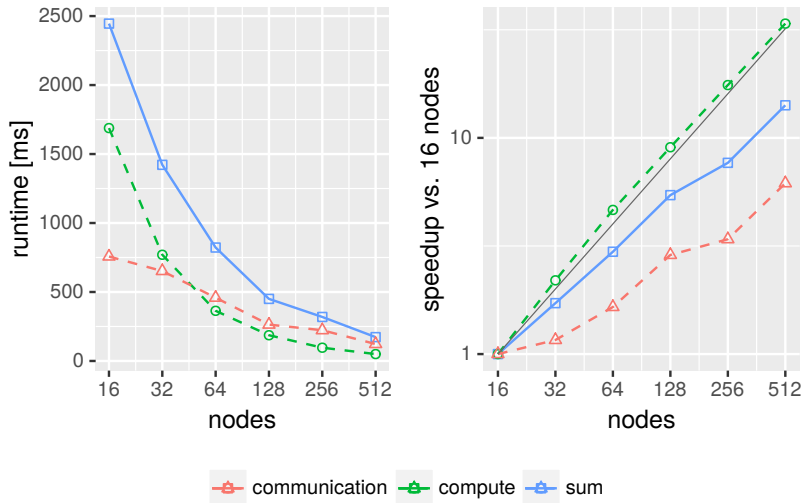
Performance Portability Relative to Xeon (SKL)



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

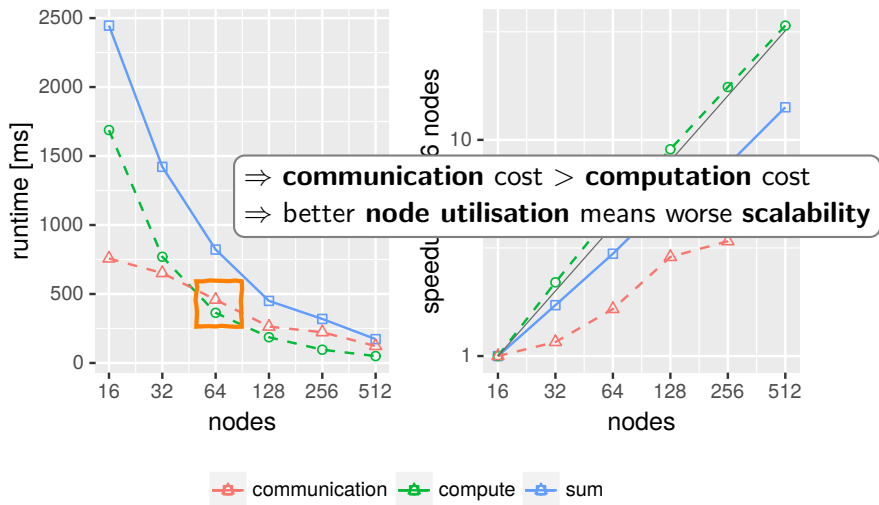
# Benchmarks: Scalability

## Strong Scaling of PS I with 3 Layers



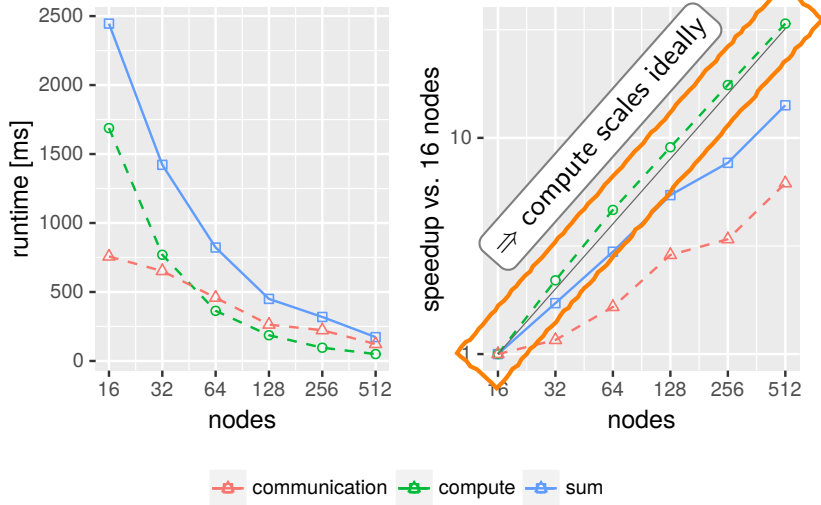
# Benchmarks: Scalability

## Strong Scaling of PS I with 3 Layers



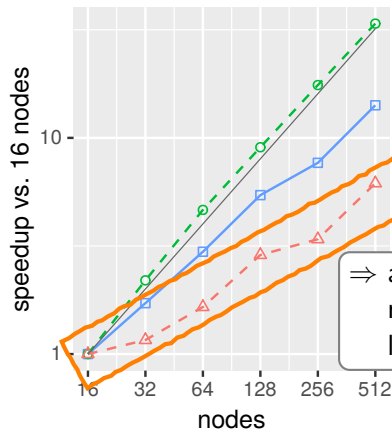
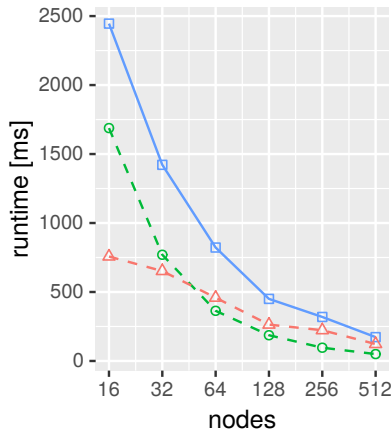
# Benchmarks: Scalability

## Strong Scaling of PS I with 3 Layers



# Benchmarks: Scalability

## Strong Scaling of PS I with 3 Layers



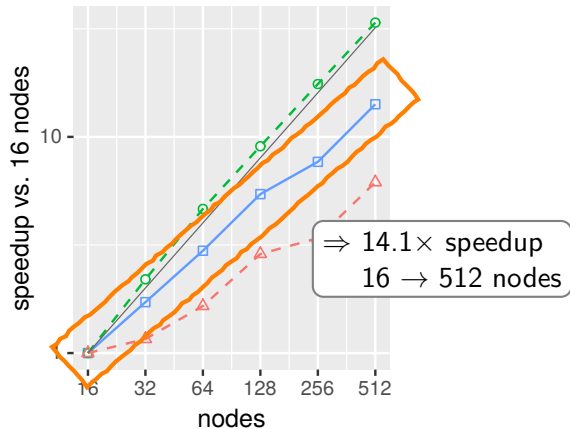
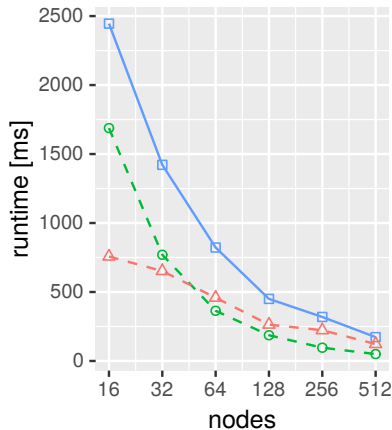
⇒ all-to-all like  
neighbour-exch.  
limits scalability

communication compute sum



# Benchmarks: Scalability

## Strong Scaling of PS I with 3 Layers



communication compute sum

# Summary

## Lessons's learned:

- **interdisciplinary workflow** is key for developing HPC codes
- standards (OpenCL, MPI-3, ...) enable **portability**
- a *flexible* design enables **portable performance**
  - ⇒ leverage **runtime compilation**
  - ⇒ **work-item granularity** and **memory and layout**



# Summary

## Lessons's learned:

- **interdisciplinary workflow** is key for developing HPC codes
- standards (OpenCL, MPI-3, ...) enable **portability**
- a *flexible* design enables **portable performance**
  - ⇒ leverage **runtime compilation**
  - ⇒ **work-item granularity** and **memory and layout**



## DM-HEOM:

- first **Distributed Memory HEOM** implementation
- pushes the boundary of **feasible problem sizes**
- **practical scalability** from laptops to supercomputers

# Thank you.

Feedback? Questions? Ideas?

[noack@zib.de](mailto:noack@zib.de)



This work was funded by the Deutsche Forschungsgemeinschaft (DFG), projects RE 1389/8-1 and KR 2889/7-1. Additional funding came from Intel Corp. within the activities of 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 computing time on the Cray XC40.