Application Performance on Many-Core Platforms

ZIH Colloquium

26.04.2018

Thomas Steinke
Zuse Institute Berlin

ZIB, HLRN and Supercomputing

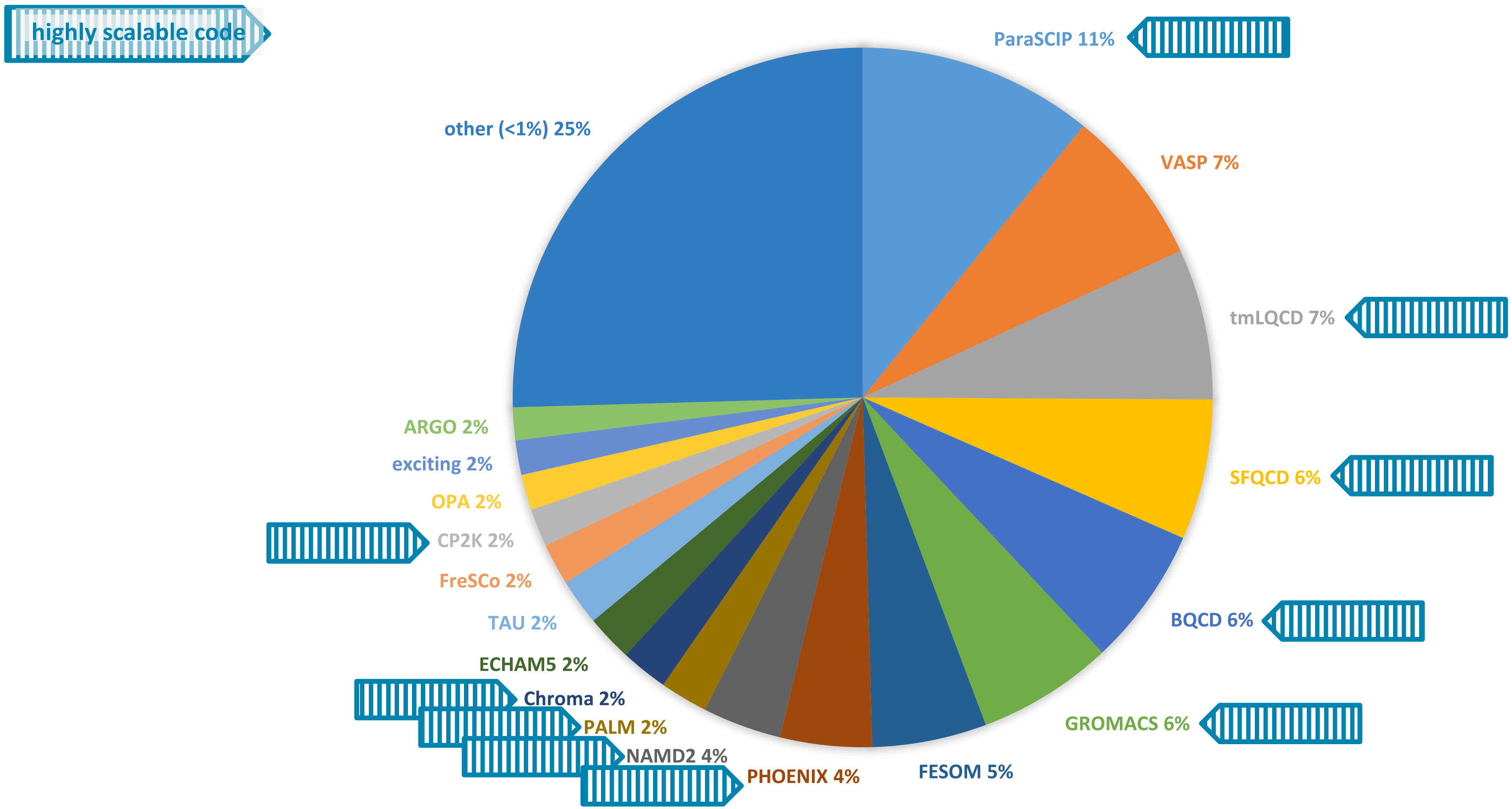
TDS (Berlin, ZIB)
- **80 KNL nodes**
- 10 Cray DataWarp nodes

"Konrad" (Berlin, ZIB)
- 1872 Xeon nodes
- 44928 cores

10 Gbps (243 km linear distance)

"Gottfried" (Hanover, LUIS)
- 1680 Xeon nodes
- 40320 cores + 64 SMP servers, 256/512 GB

HLRN Workloads



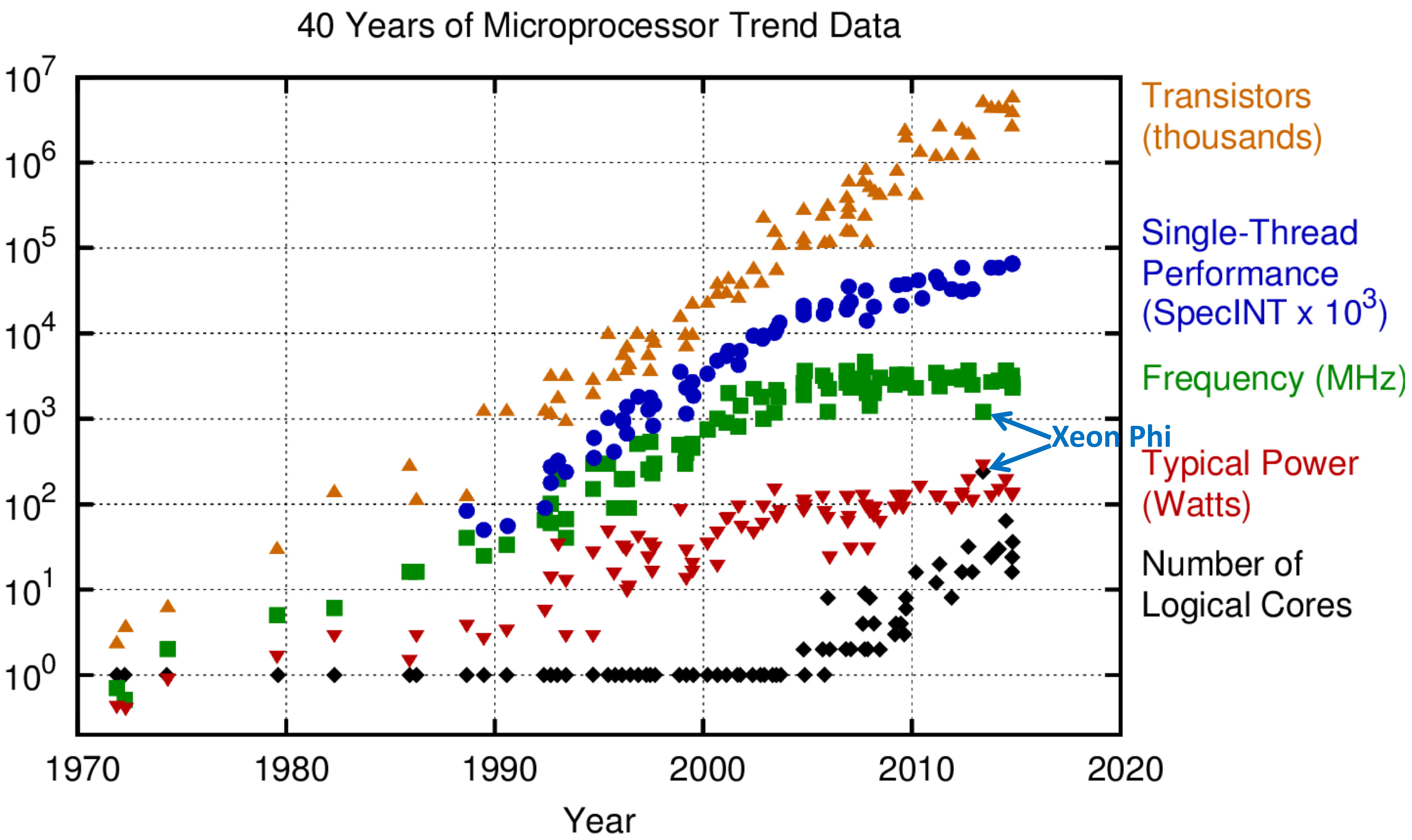
IPCC @ ZIB
Research Center for Many-core HPC

Context and Areas of Work

Intel Parallel Computing Center @ ZIB

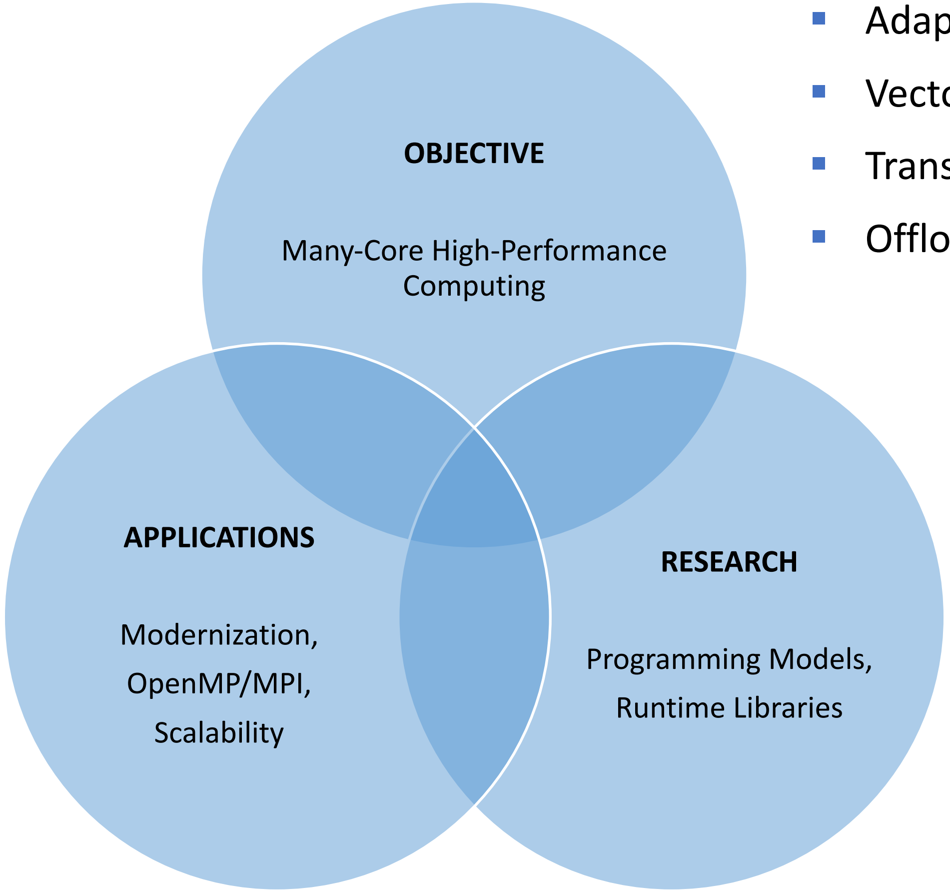
- Founded in 2013
- Part of the **Research Center for Many-Core HPC** at ZIB
- Goal: prepare HLRN user community for (near) future HPC platforms

The Need to Scale



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Areas of Work



Challenges

- Adapting data structures for enabling SIMD
- Vectorizing complex code structures
- Transition to hybrid MPI + OpenMP
- Offload Over Fabric

Selected Applications

- ***VASP*** (electronic structure)
- ***GLAT*** (atomistic thermodynamics)
- ***BQCD*** (high-energy physics)
- ***HEOM*** (photo-active processes)
- ***PALM*** (fluid dynamics)
- ***PHOENIX3D*** (astrophysics)

Benchmark Platforms

Phase I (Intel Knights Corner – KNC):

- 4x nodes with Xeon Phi 7120P nodes, IB network
- Cray XC30 TDS with 16 nodes with Intel Xeon Phi 5120 (60 cores)

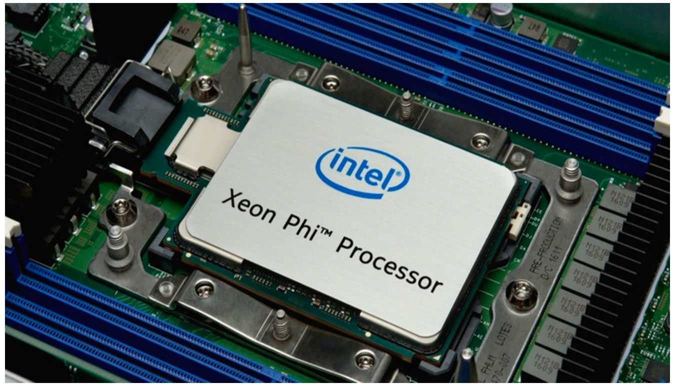
Phase II (Intel Knights Landing – KNL):

- Development server with Intel Xeon Phi 7210
- Cray XC40 TDS with 80 Xeon Phi 7250 nodes (68 cores)

Intel Xeon Phi 7200 Architecture

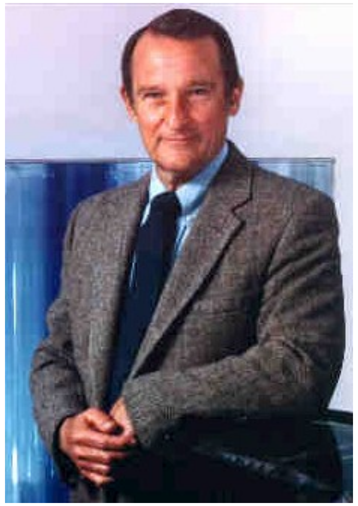
Knights Landing (KNL): Intel's 2. Many Integrated Core (MIC) Architecture1)

- self-booting CPU, optionall with integrated OmniPath fabric
- 64+ core (based on Intel Atom Silvermont architecture, x86-64)
- 4-way hardware-threading
- 512-bit SIMD vector processing (AVX-512)
- On-Chip Multi-Channel (MC) DRAM: 16 GiB
- DDR4 main memory: up to 384 GiB



A. Sodani et al., Knights Landing: Second-Generation Intel Xeon Phi Product, IEEE Micro vol. 6, April 2016

*"As long as we can make them smaller,
we can make them faster."*



Seymour R. Cray

HLRN-I at ZIB, 2002: **2,5 TFlops**

12 IBM p690 cabinets, each 4 x 8 = 32 Power4 cores, 384 cores per site

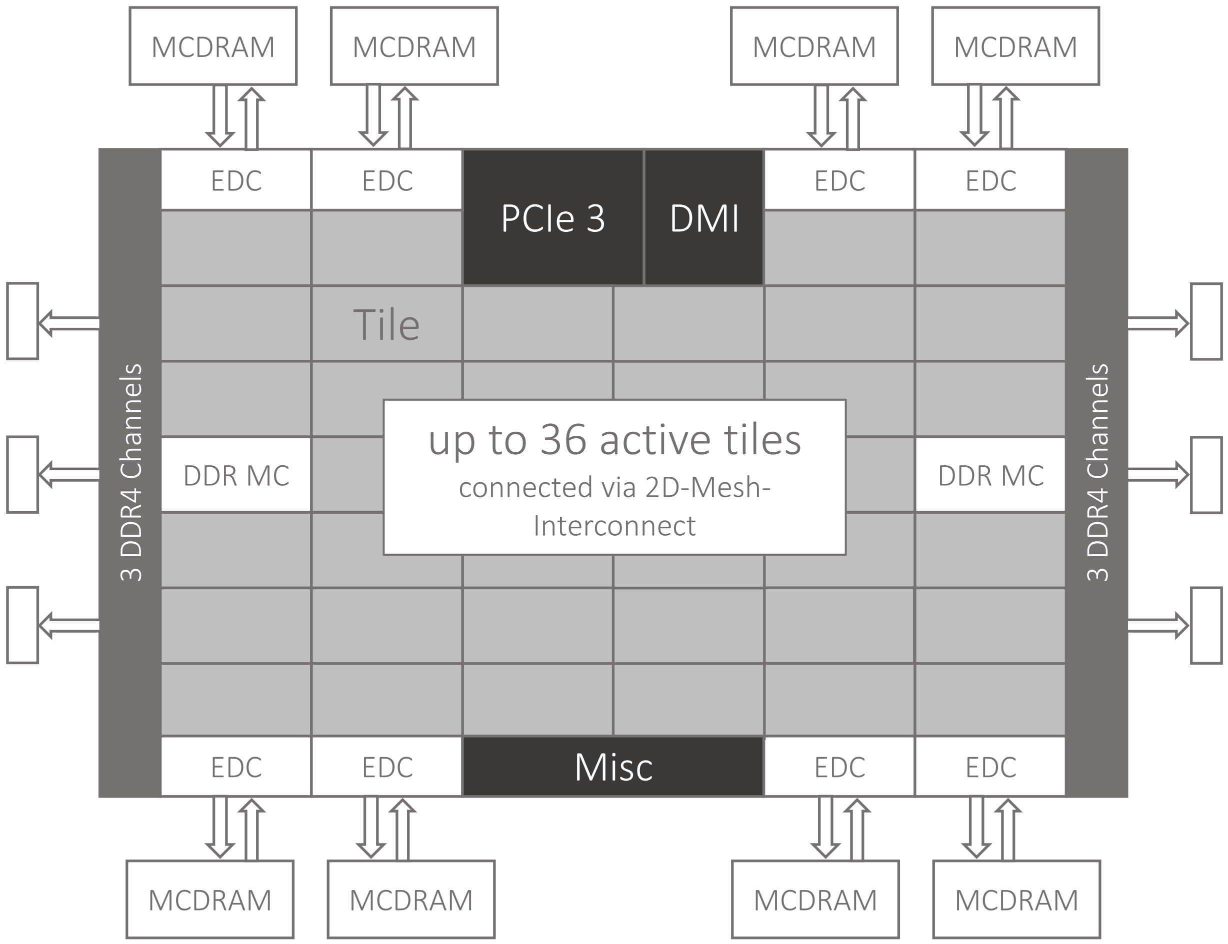
IBM p690
384 cores
2,5 TFLOPS
200 kWatt
10 mio €

Xeon Phi KNL, 2016, **3 TFlops**

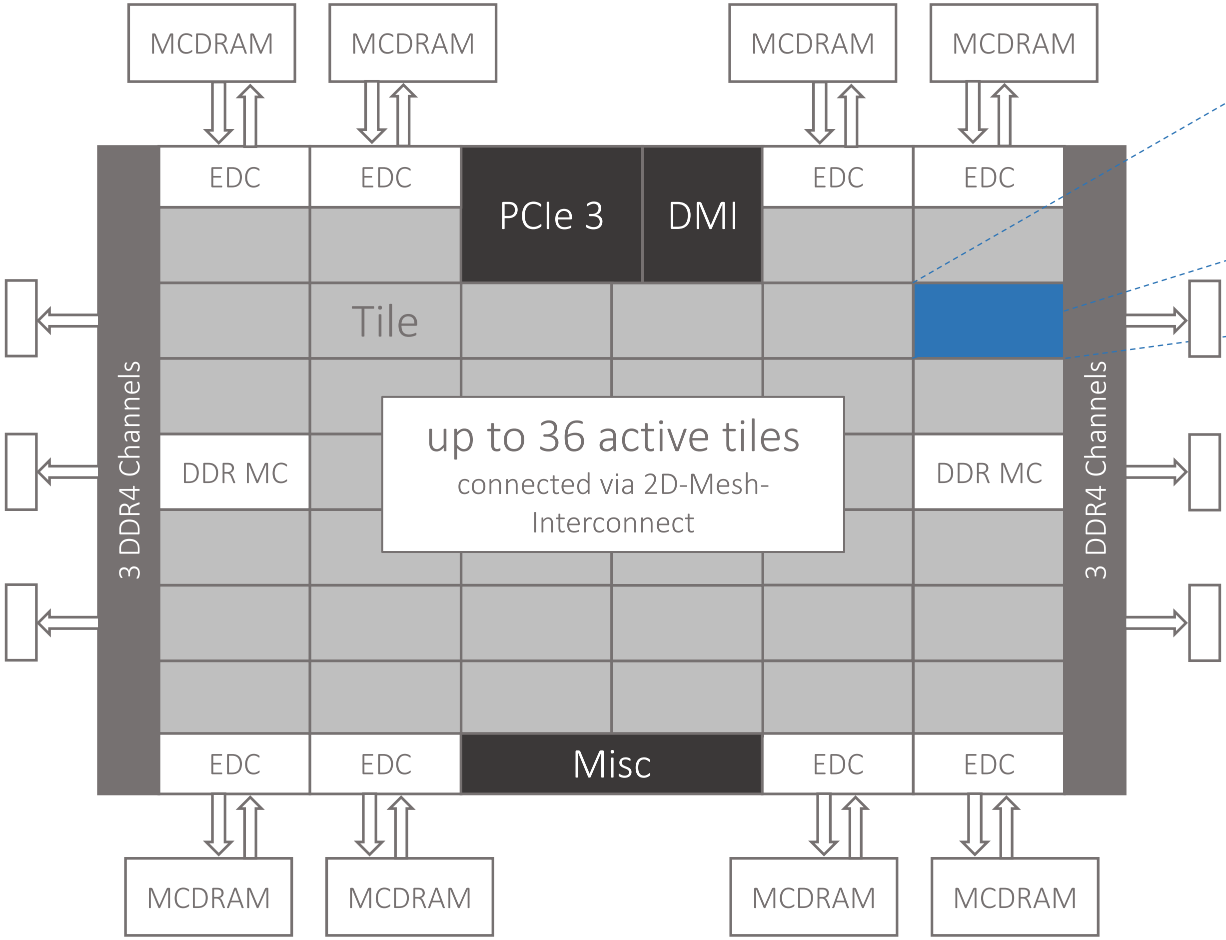
8 billion gates

Intel Xeon Phi 7290
72 cores (288 threads)
3 TFLOPS
245 Watt
6662 € (6/2017)

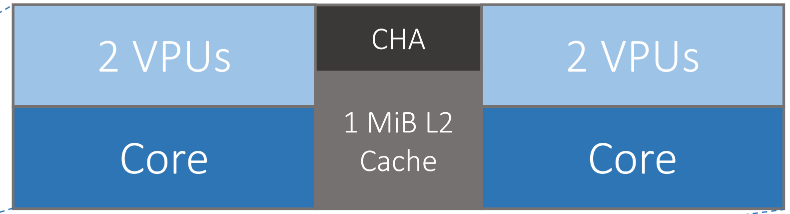
Intel KNL Architecture



Intel KNL Architecture (II)

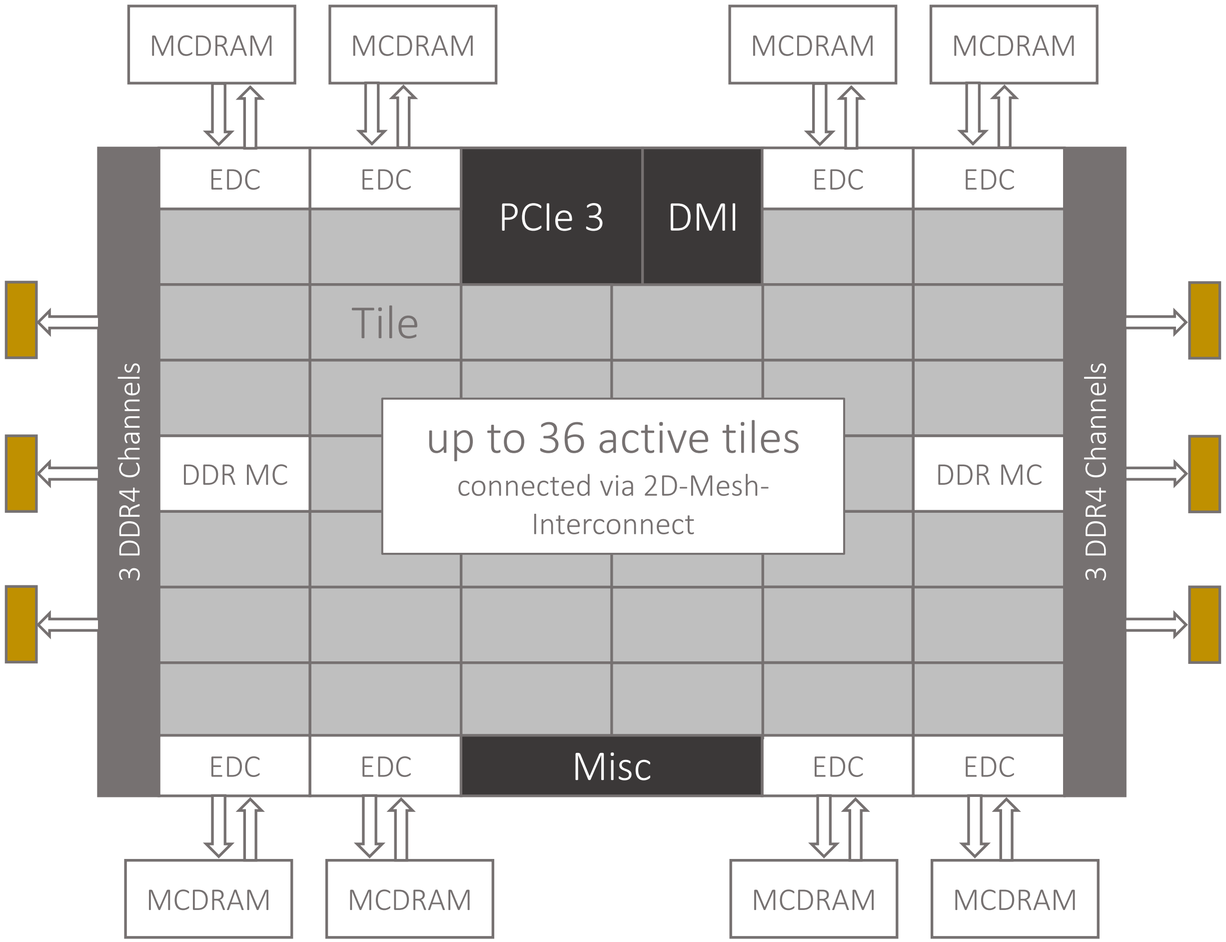


Tile



- 2 out-of-order cores
- 2 VPUs per core (AVX-512)
- 1 MiB shared L2-Cache
- Caching/Home-Agent (CHA)
  - interface to 2D-Mesh-Interconnect
  - distributed tag directory (MESIF cache coherency protocol)

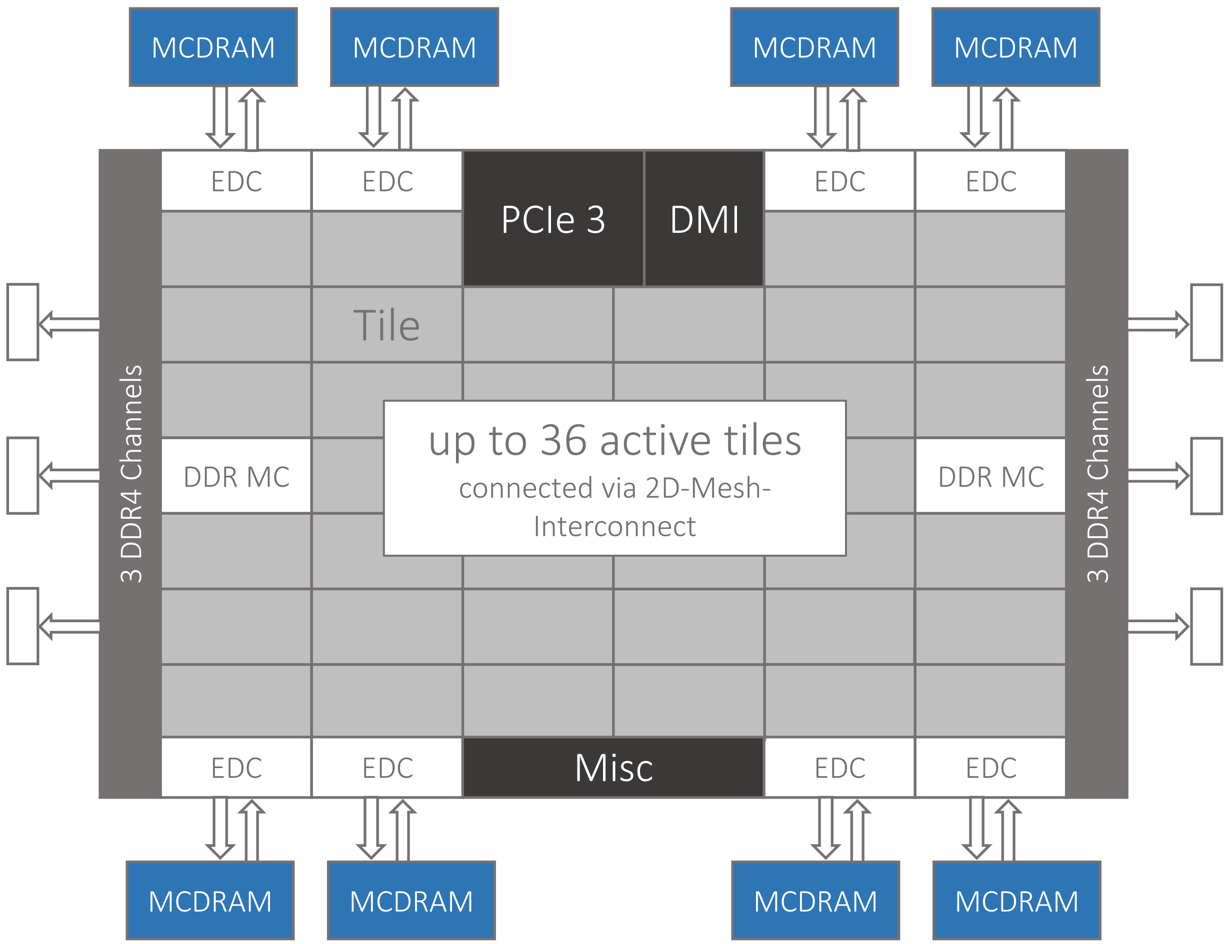
Intel KNL Architecture (III)



DDR4 Memory (up to 384 GiB)

- 2 memory controller
- 6 DDR4 Channels

Intel KNL Architecture (IV)



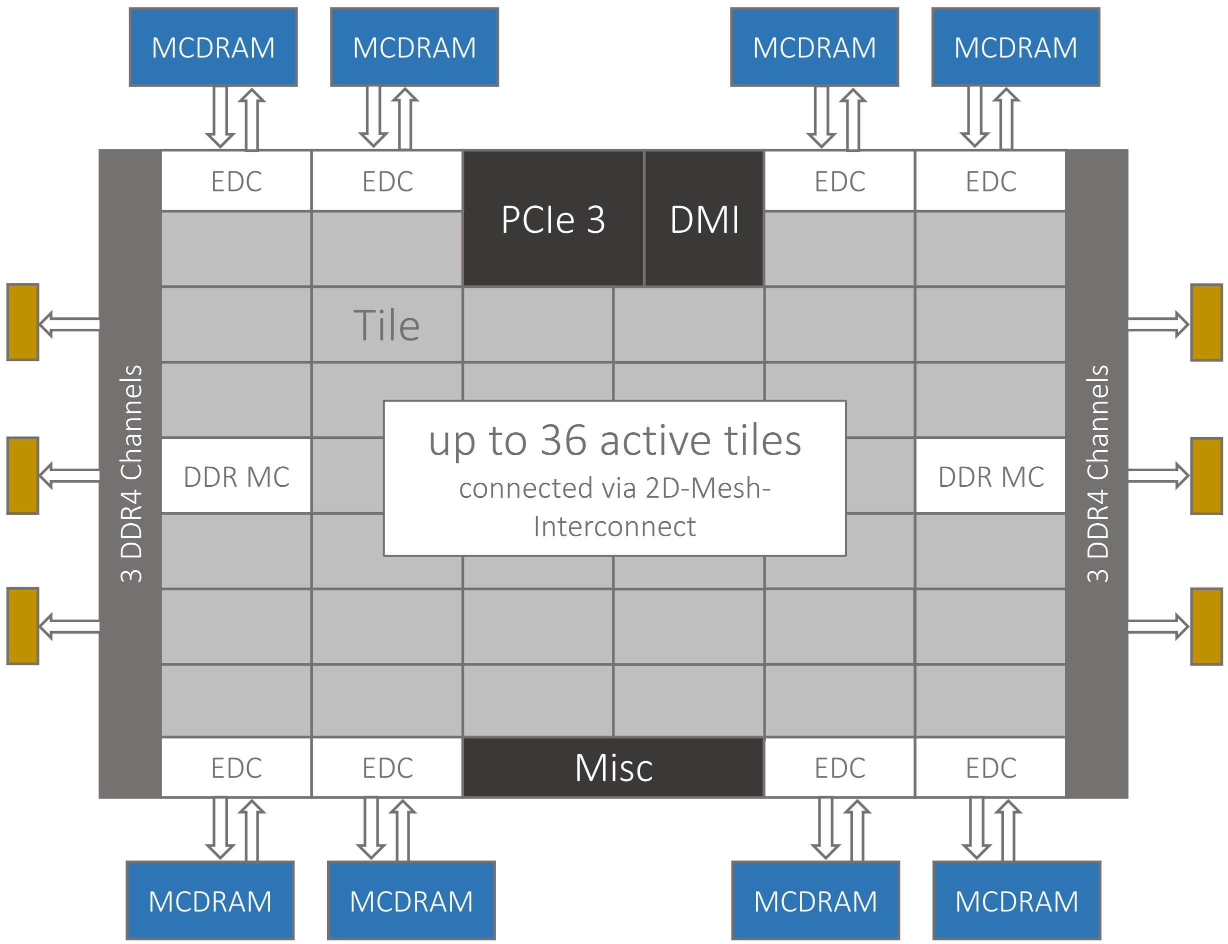
DDR4 Memory (up to 384 GiB)

- 2 memory controller
- 6 DDR4 Channels

MCDRAM (16 GiB)

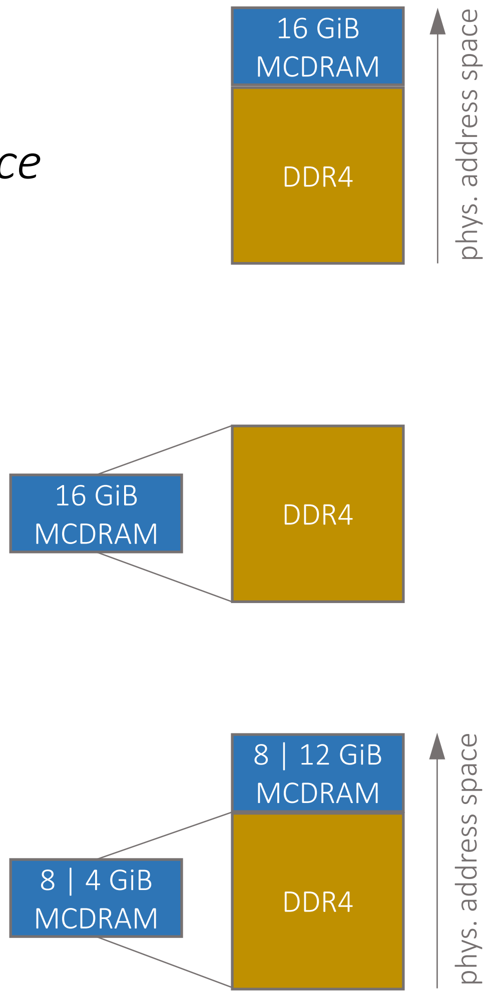
- 8 on-chip units, each 2 GiB

Intel KNL Architecture (V)



Memory Modes:

- ❑ Flat-Mode
  DDR4 and MCDRAM in same address space
- ❑ Cache-Mode
  MCDRAM = direct-mapped Cache for DDR4
- ❑ Hybrid-Mode
  8 | 4 GiB MCDRAM in Cache-Mode, remainder in Flat-Mode



Phi Features in Mainstream CPU

- AVX-512 support in high-end Intel Xeon CPUs
    - 1-2 AVX-512 SIMD units per core

- 2D mesh interconnect replacing (multiple) rings

- A lot of improvements of the Intel software stack (compiler, MKL)
- New features in performance analysis tools (Intel Vector Advisor)

Acknowledgements

- Matthias Noack
- Florian Wende

- Thorsten Schütt
- Tobias Kramer
- Guido Laubender

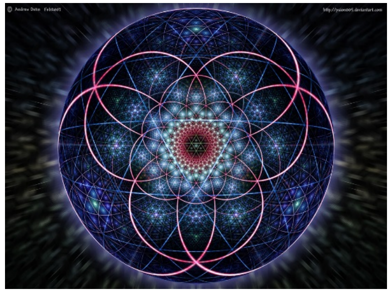
Key Results – Code Modernization

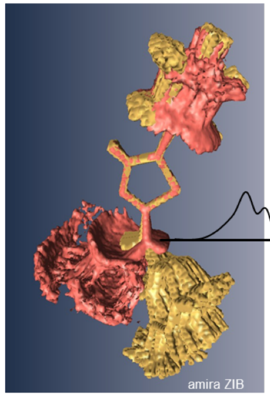
CODE EXAMPLES: VASP & DM-HEOM

Work of Matthias Noack & Florian Wende, and
with contributions from Frank Cordes (getlig&tar) & Thorsten Schütt

Code Modernization – Overview (Q3/2017)

Code	Performance (over 2S HSW)	Comments
VASP	1.1 – 1.3x (SIMD) 1.5 – 1.9x (+ OpenMP)	Single-node Multi-node
PALM	1.3 – 1.4x	Moderately sized multi-node
BQCD	1.4x (over 2S IVB)	Multi-node, over 2S IVB, L3 cache issue
DM-HEOM	*on par,* AVX2 only	Missing OpenCL with AVX-512 support
GLAT	2 – 3x (over 2S IVB)	Multi-node, parallel efficiency > 60%





Work of Frank Cordes (getlig&tar), Matthias Noack, Thorsten Schütt, and Florian Wende

VASP: SIMD and FFT

Work by Florian Wende

Code Modernization: VASP

- Collaboration with Georg Kresse and Martijn Marsman (Univ Vienna), Jeongnim Kim (Intel), and Zhengji Zhao (NERSC)
- VASP is a electronic structure plane-wave code mostly used for atomistic simulations in material science
- #1 code in electronic structure based molecular simulation domain on HLRN, NERSC and other sites

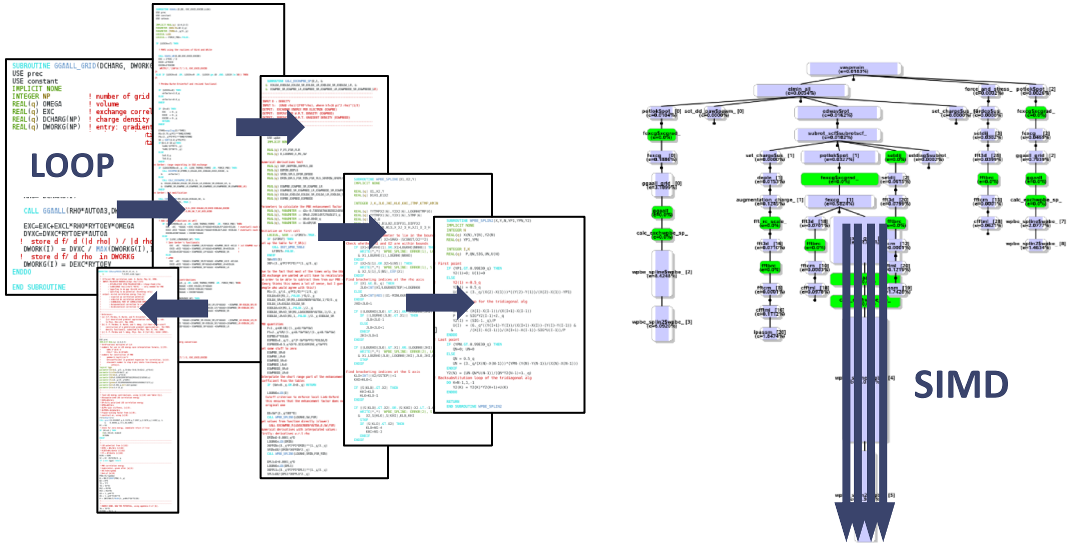
- **SIMD in the complex control flows and how to enhance 3D FFT**

Work of Florian Wende

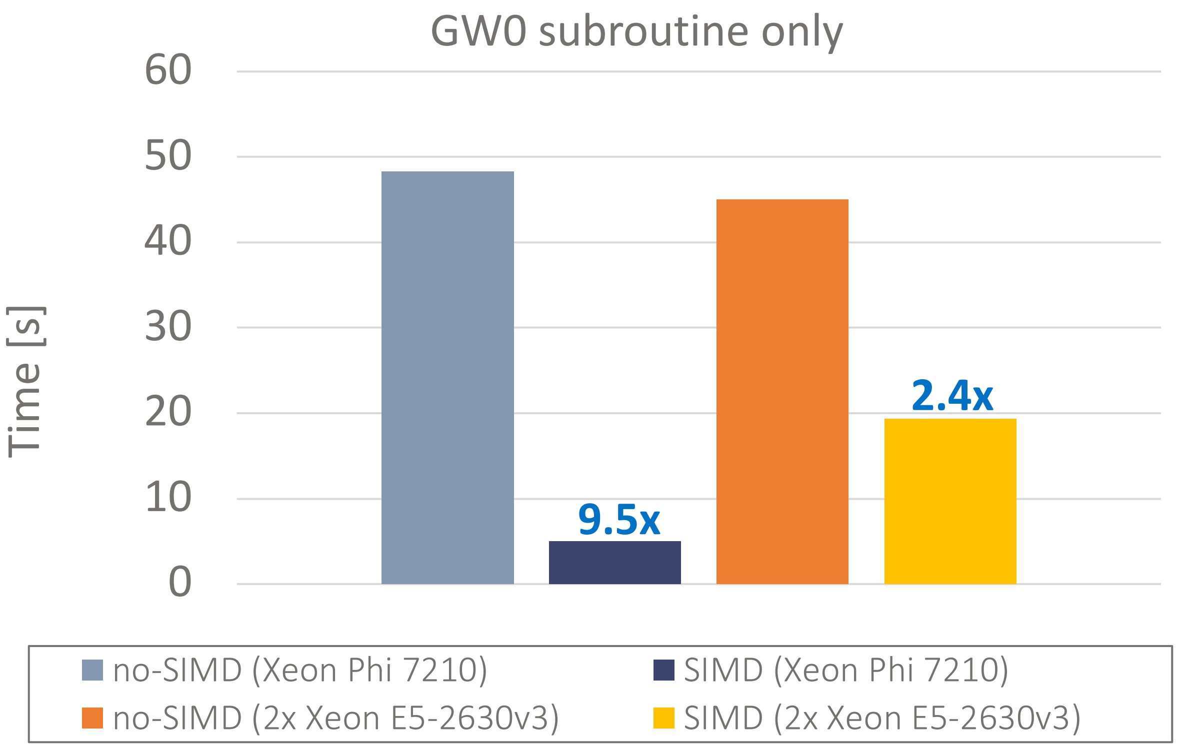
SIMD in Complex Control Flows

SIMD is most challenging for branchy and complex loops

- Change in data layout
- Seperate into pre-conditioning & SIMD loops



Work of Florian Wende



SIMD with OpenMP 4.x

OpenMP 4.x compiler directives

- Portability across compilers
- Low code invasiveness
- No SIMD intrinsics for Fortran

- Idea: combine OpenMP 4.x SIMD with “high-level vectors” (loop chunking) to increase flexibility and expressiveness

Enabling (Partially) SIMD (I)

Non-vectorizable loop split into parts to enable SIMD vectorization

```
idx = 0;
for (i = 0; i < ni; ++i) {
  while ("some condition")
    ++idx;
  d = data[idx];
  for (j = 0; j < nj; ++j)
    res[j] += d * (...);
}
```

C-version of the code
(not optimized)

`nj` rather small: not a candidate for SIMD vectorization

Enabling (Partially) SIMD (II)

Non-vectorizable loop split into parts to enable SIMD vectorization

```
idx = 0;
for (i = 0; i < ni; ++i) {
  while (“some condition”)
    ++idx;
  d = data[idx];
  for (j = 0; j < nj; ++j)
    res[j] += d * (...);
}
```

Loop-Chunking, e.g. CHUNKSIZE=32

```
idx = 0;
for (i = 0; i < ni; i += CHUNKSIZE) {
  ii_max = min(CHUNKSIZE, ni – i);
  for (ii = 0; ii < ii_max; ++ii) {
    while (“some condition”)
      ++idx;
    vidx[ii] = idx;
  }
  ...
}
```

Data Pre-Conditioning:
Compute `idx`-values in advance to enable SIMD vectorization afterwards!

Enabling (Partially) SIMD (III)

Non-vectorizable loop split into parts to enable SIMD vectorization

```
idx = 0;
for (i = 0; i < ni; ++i) {
  while (“some condition”)
    ++idx;
  d = data[idx];
  for (j = 0; j < nj; ++j)
    res[j] += d * (...);
}
```

```
idx = 0;
for (i = 0; i < ni; i += CHUNKSIZE) {
  ii_max = min(CHUNKSIZE, ni – i);
  for (ii = 0; ii < ii_max; ++ii) {
    while (“some condition”)
      ++idx;
    vidx[ii] = idx;
  }
  for (ii = 0; ii < ii_max; ++ii)
    vd[ii] = data[vidx[ii]];
  ...
}
```

Load data in a separate loop: leave it to the compiler to vectorize or not

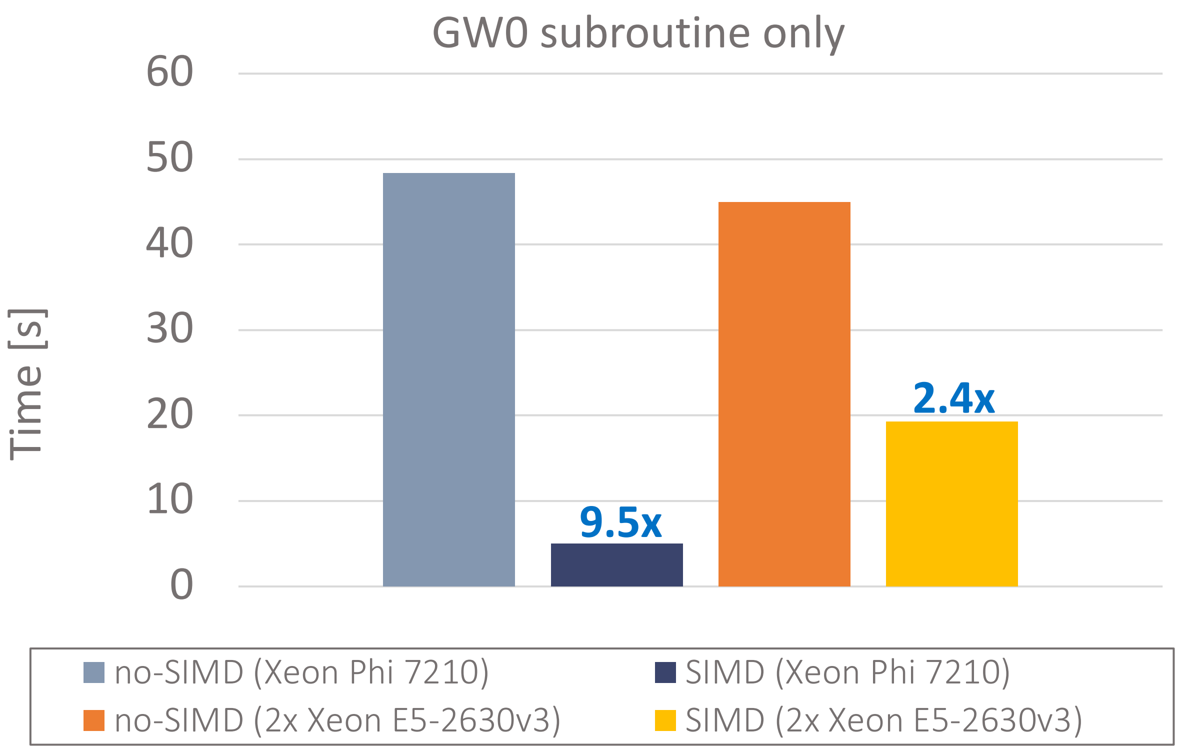
Enabling (Partially) SIMD (IV)

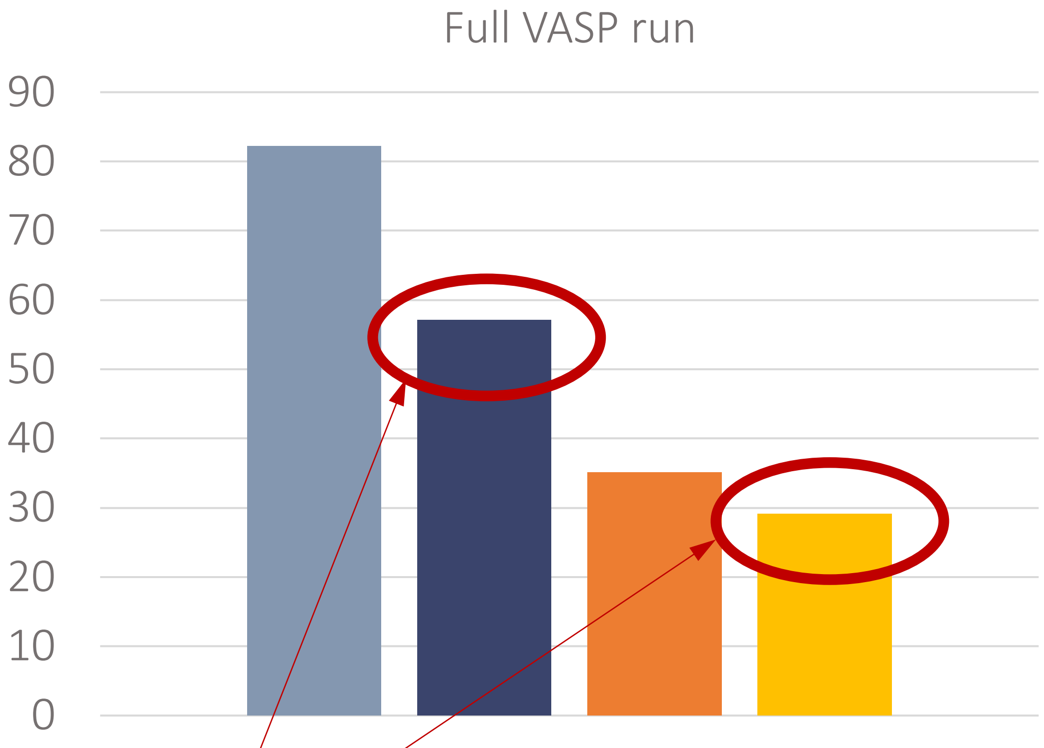
Non-vectorizable loop split into parts to enable SIMD vectorization

```
idx = 0;
for (i = 0; i < ni; ++i) {
  while (“some condition”)
    ++idx;
  d = data[idx];
  for (j = 0; j < nj; ++j)
    res[j] += d * (...);
}
```

```
idx = 0;
for (i = 0; i < ni; i += CHUNKSIZE) {
  ii_max = min(CHUNKSIZE, ni – i);
  for (ii = 0; ii < ii_max; ++ii) {
    while (“some condition”)
      ++idx;
    vidx[ii] = idx;
  }
  for (ii = 0; ii < ii_max; ++ii)
    vd[ii] = data[vidx[ii]];
  for (j = 0; j < nj; ++j)
    #pragma omp simd
    for (ii = 0; ii < ii_max; ++ii)
      res[j] += vd[ii] * (...);
}
```

Performance Gain





further optimization needed!

Xeon Phi nodes: quadrant mode, all data in MCDRAM

VASP – Enhancing 3D FFT Computation

Both **FFTW** and MKL benefit from the following improvements on top of already existing library implementations

- Plan caching
- Composed 3D FFT
- *Ball-cube optimization (wip)*
- Less compiler dependent (e.g. Intel vs. GCC)

hosted on Github: `github.com/flwende/fftlib`

Performance gain für FFT part:

- Plan caching:
  - **24%** (16 threads) – **7%** (4 threads)
- Composed 3D FFT: 2D+1D
  - **27%** (16 threads) – **10%** (4 threads)

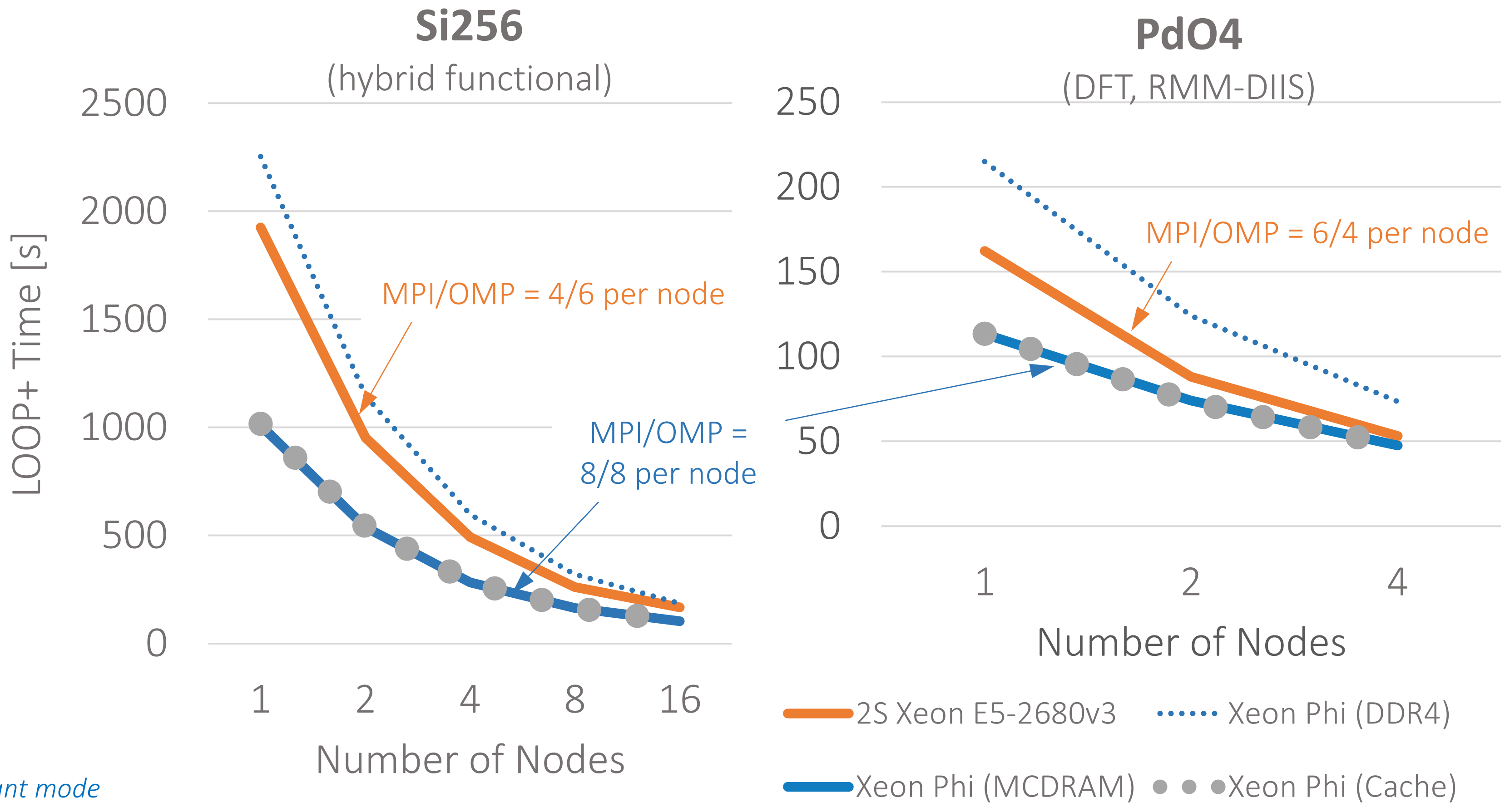
Platform: Intel Xeon Phi 7250, Cray Aries Network, quad-flat, all data in MCDRAM,
64 cores are used per Xeon Phi (one hardware thread per core)
Software: Intel ifort 17, MKL
Input: PdO4, ALGO=VeryFast (RMM-DIIS), {80x120x54, 160x240x108} FFT

VASP – Overall Performance

Joint work of the VASP developers, ZIB and Intel

Xeon Phi 7250 vs. 2S Xeon E5-2680v3:
1.5x – 1.9x gain

MCDRAM vs. DDR4:
1.5x – 2.2x gain



Xeon Phi nodes: Intel Xeon Phi 7250 (Cray XC40), quadrant mode
Haswell nodes: 2S Intel Xeon E5-2680v3 (Cray XC40)
Software: Intel 17 + Cray-MPICH 7.4.3 + MKL + ELPA 2016.05.003

OpenCL in HPC – DM-HEOM

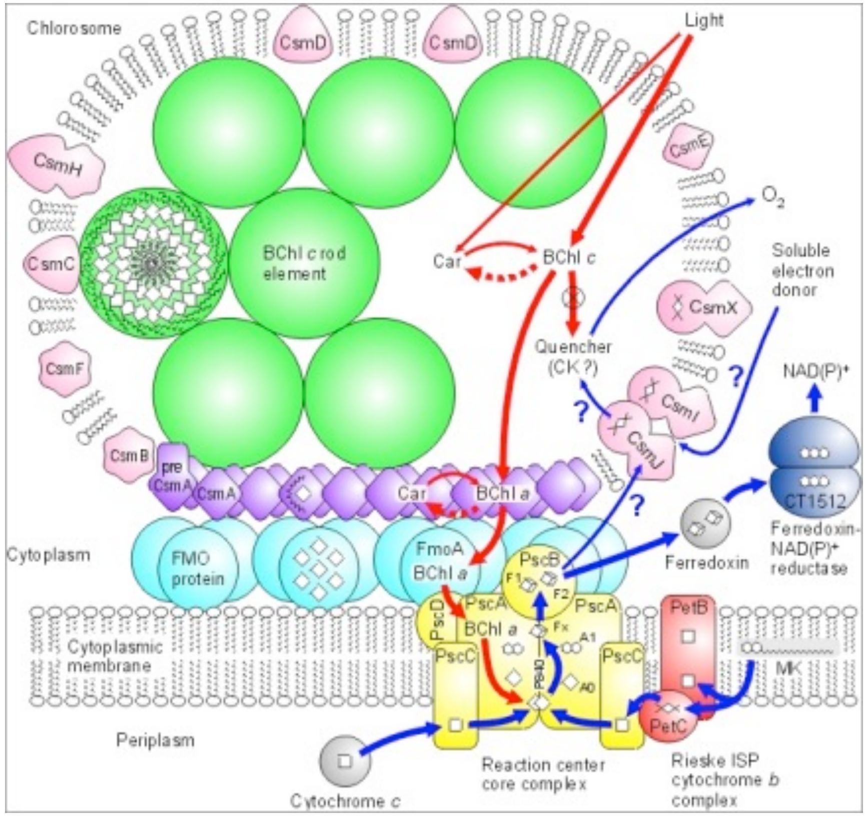
Work by Matthias Noack

Physics algorithms by Tobias Kramer

OpenCL in HPC – The DM-HEOM Showcase

Newly developed physics code solving the HEOM: designed for portability

- C++11/14, OpenCL 1.2, MPI 3.0
- runs on CPU + GPU

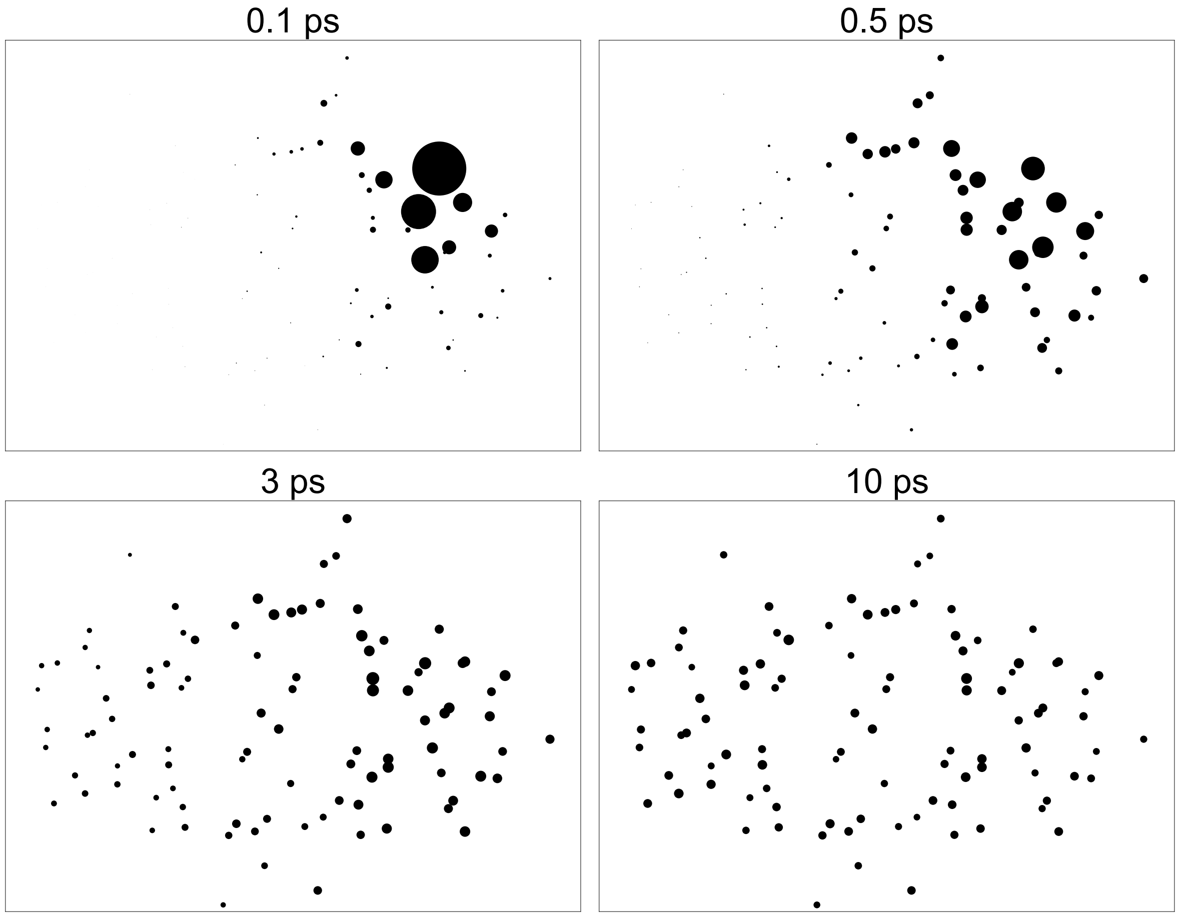


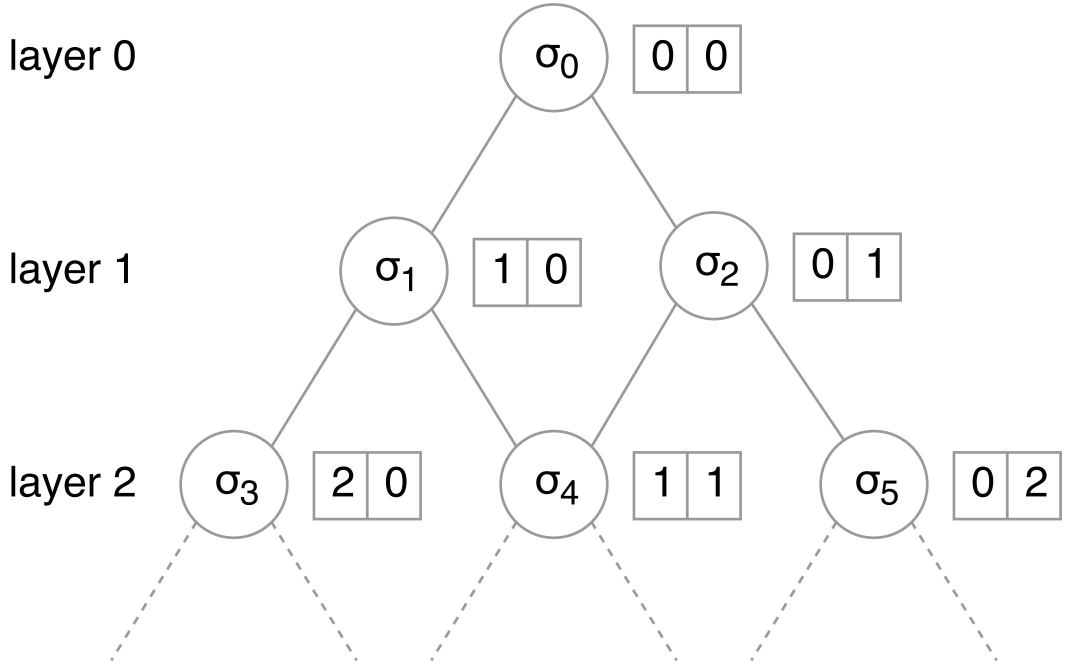
...and **proxy code** (hexciton_benchmark)

- 20 different OpenCL kernel variants
- 17 different **OpenMP** kernel variants

- Extensive OpenMP/OpenCL study → "Pearls" book chapter

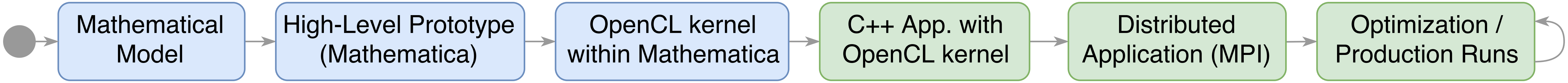
Hierarchy of Bath Interactions



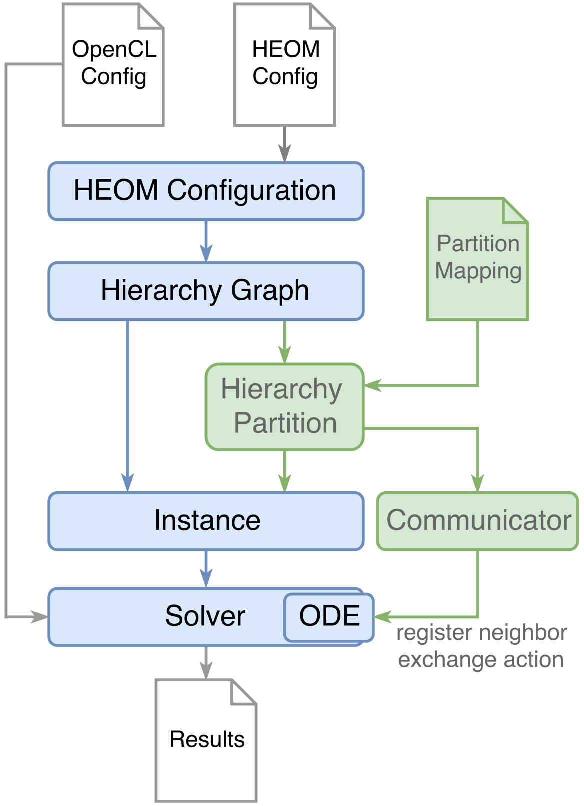


DM-HEOM Implementation & Usage

- Interdisciplinary development workflow:

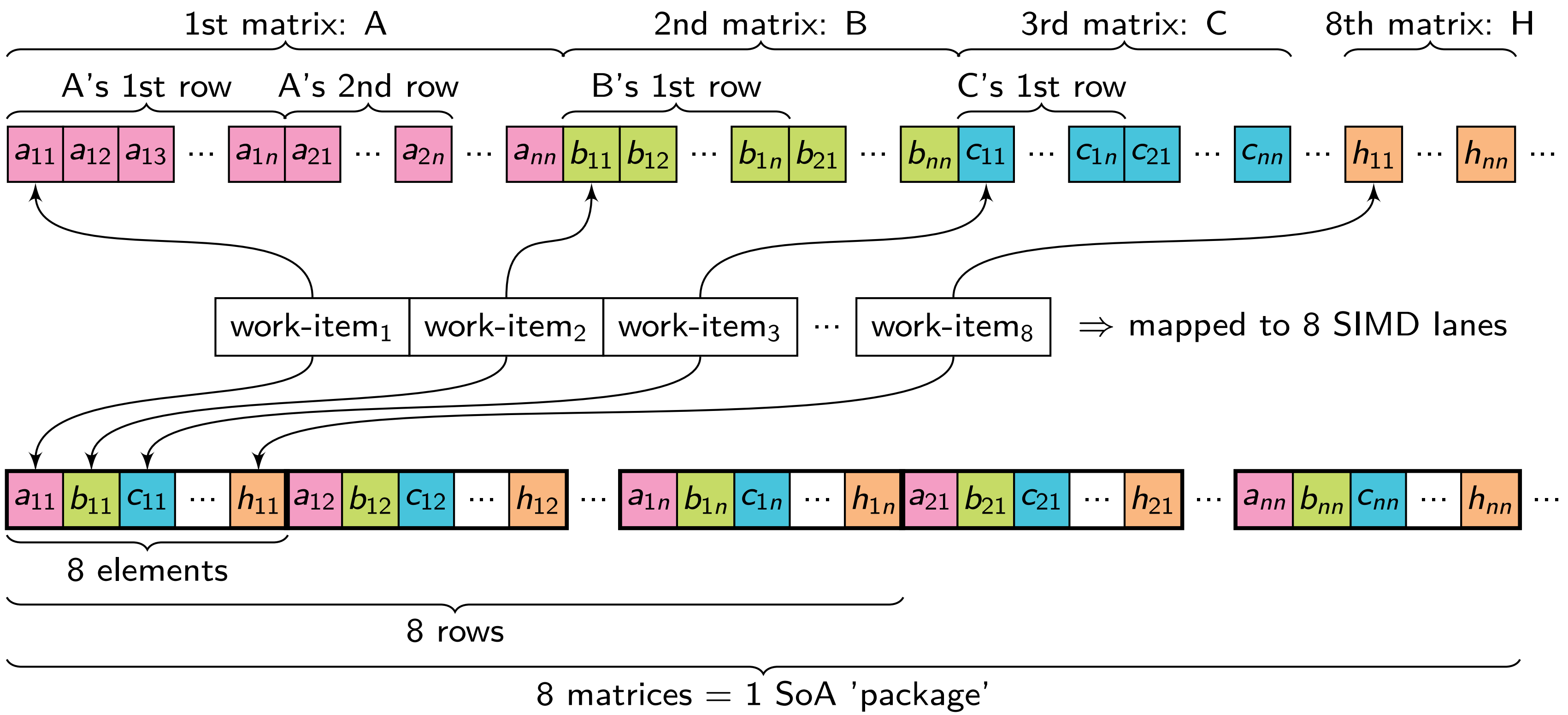


- Runtime config of DM-HEOM:



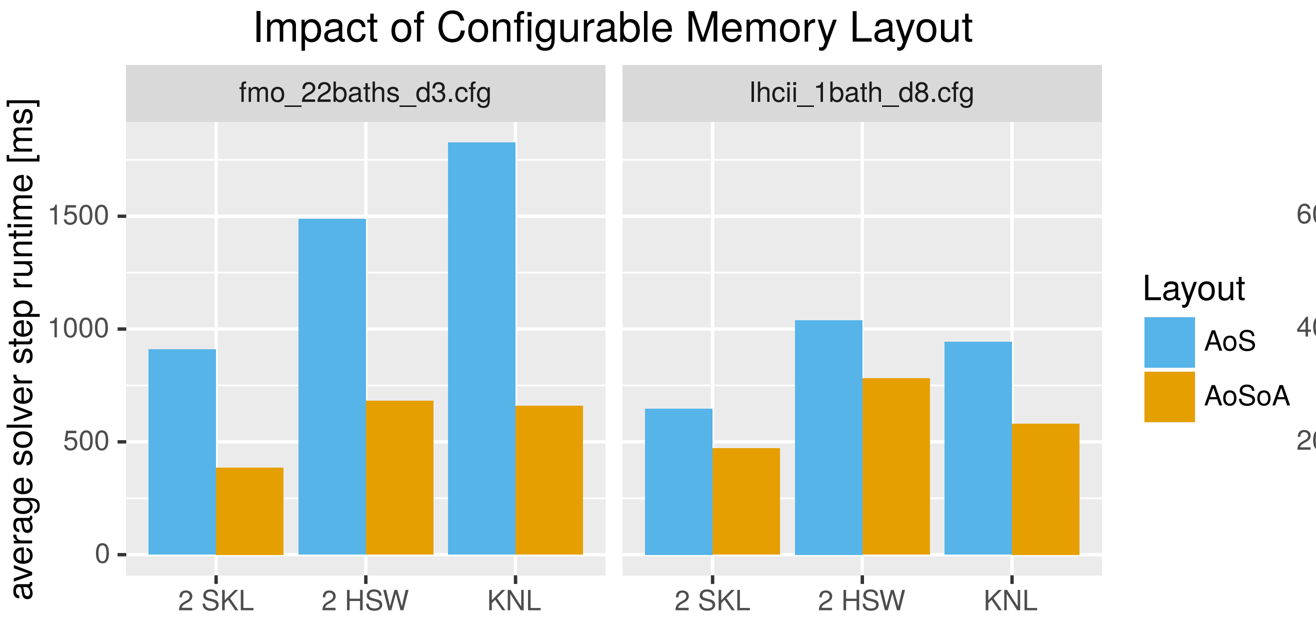
ZUSE INSTITUTE BERLIN

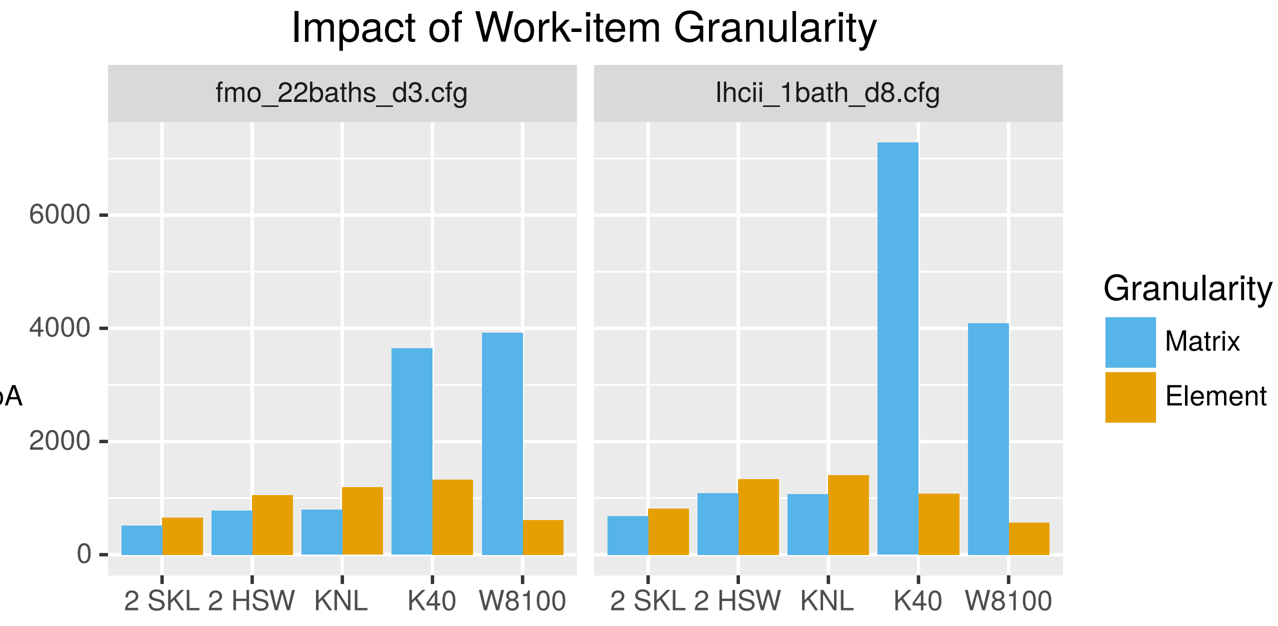
Memory Layout



- Element-wise interleaving of 8 matrices
- Eliminates all gather/scatter ops
- Speedup **3.4**x with auto-vectorizer
  - Contradiction: **manual memory layout vs. automatic vectorization**

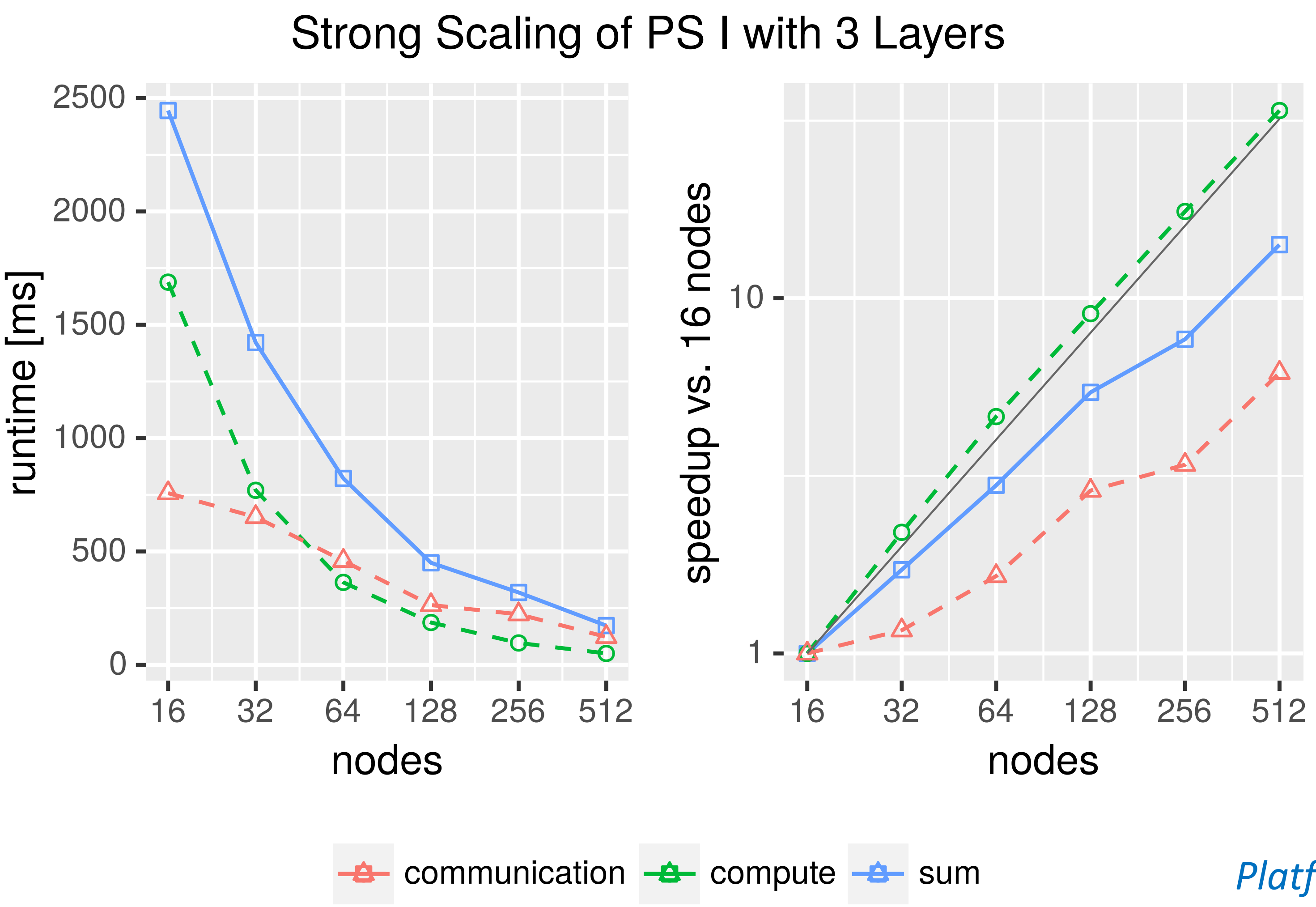
Performance Impact Factors





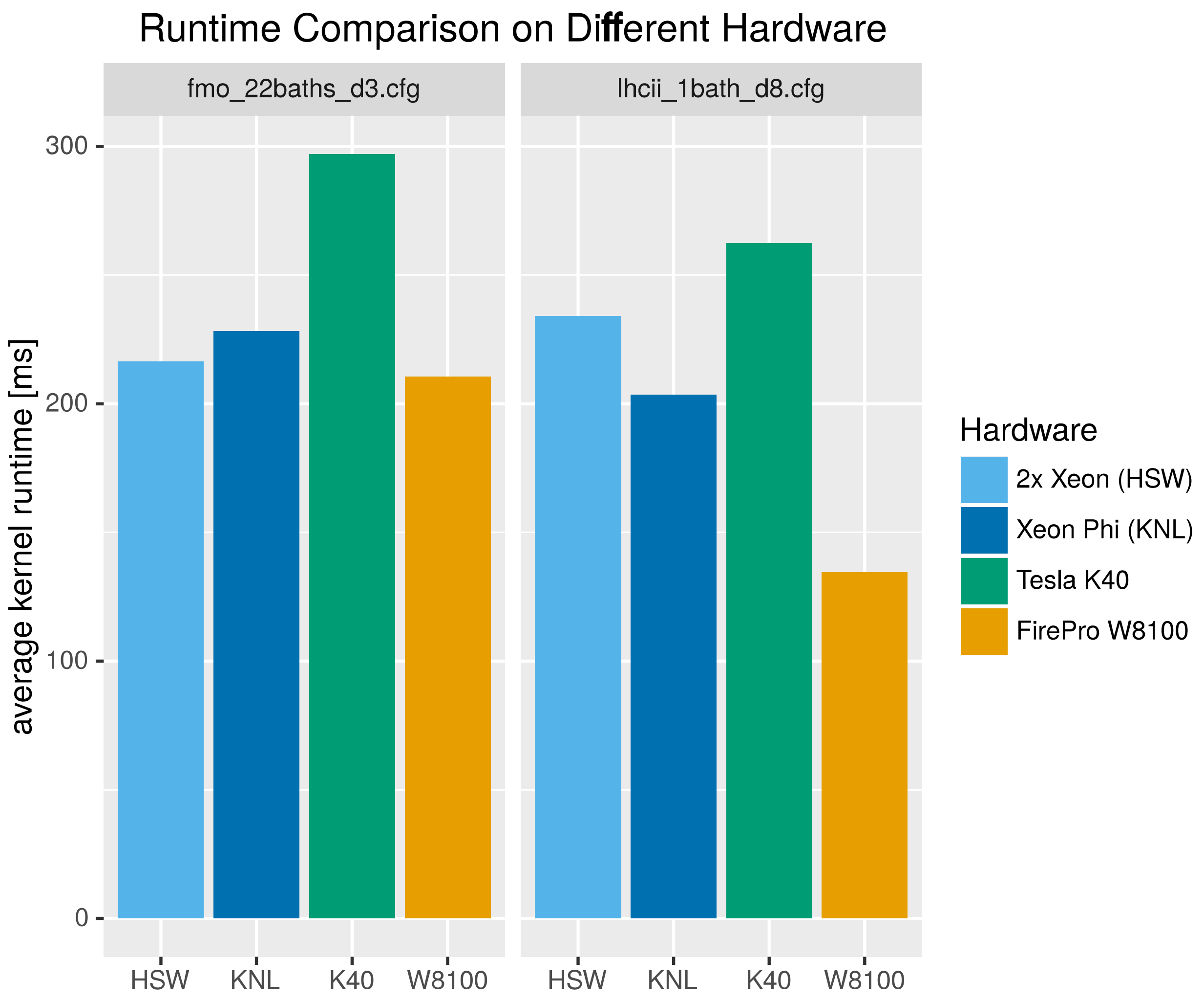
OpenCL SDK Comparison

DM-HEOM Scalability



Platform: Cray XC40 (HLRN)

OpenCL Performance Portability



Lesson's Learned

- Use **AoSoA** memory layout
- Exploit **runtime kernel compilation** of OpenCL
- Performance gap between OpenCL and OpenMP is small on Xeon Phi
  - Simple transition for Phi optimized kernels
  - No replacement for runtime kernel compilation
- OpenCL guarantees portability, not portable performance
  - Takes some extra effort (granularity, memory layout, ...)
- Use explicit/manual vectorization over relying on the compiler
  - Portable non-OpenCL solutions: C++ SIMD type libraries, OpenMP SIMD constructs
- *Haswell performance (surprisingly) insensitive to optimization*

Key Results – Programming Tools

KART (Matthias Noack)

Enhanced Explicit Vectorization (Florian Wende)

HAM-Offload (Matthias Noack)

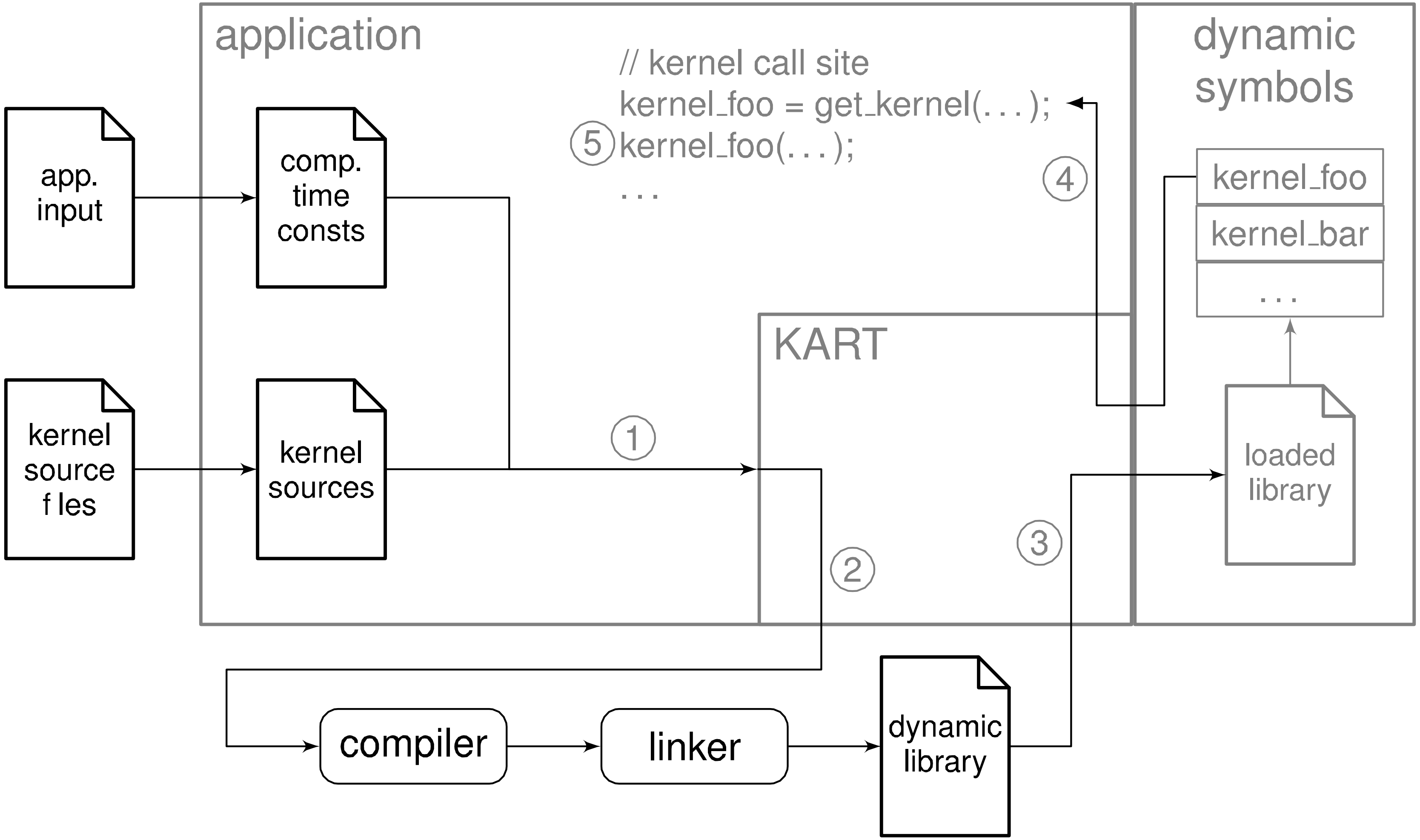
KART - Kernel compilation At RunTime

- Idea: defer compilation of kernels (hotspots) until application time
  - OpenCL does it by design, OpenMP cannot
- Goals:
  - Improving/enable SIMD vectorization, conditinal elimination, memory acess optimization, loop tansformation, ...
- KART: = C++ library
  - API for C++/Fortran
  - Can use any compiler like on the command line
    - LLVM/JIT is not enough

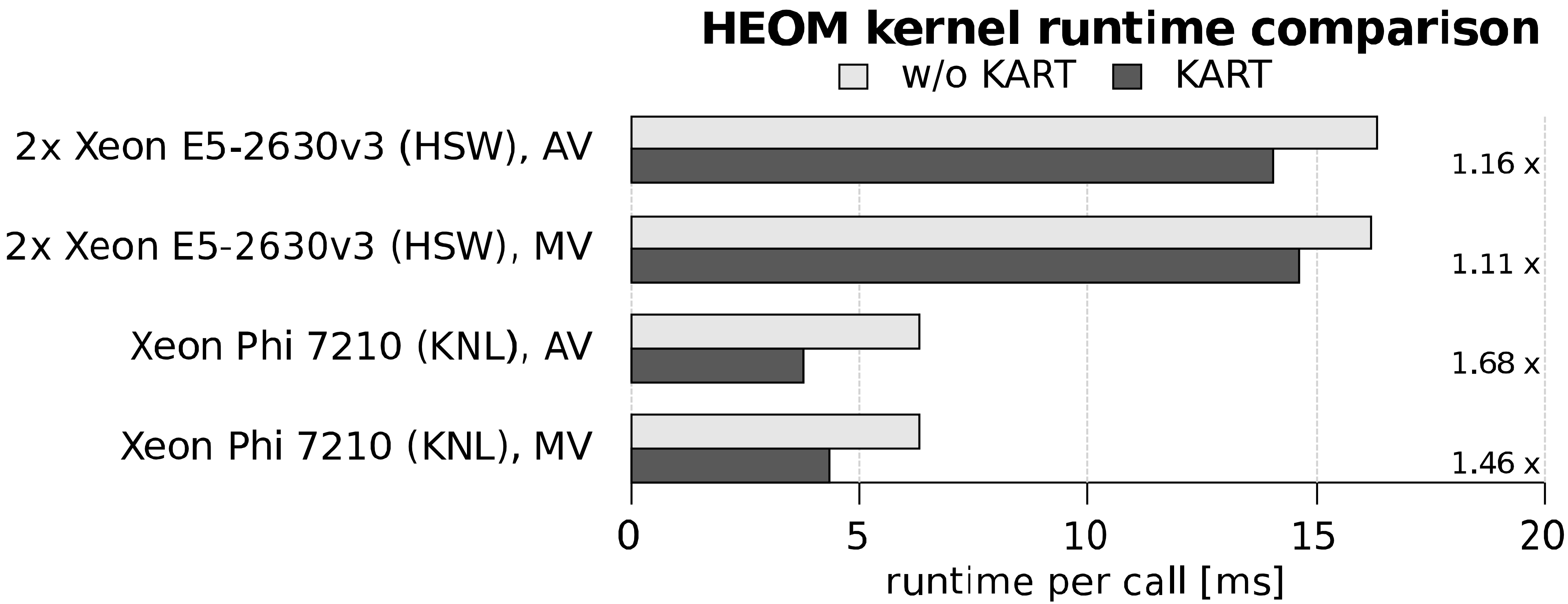
KART on Github: https://github.com/noma/kart
noack@zib.de

M. Noack, F. Wende, G. Zitzlsberger, M. Klemm, Th. Steinke; *KART – A Runtime Compilation Library for Improving HPC Application Performance*, ISC'17 IXPUG Workshop Proceedings

KART Implementation



KART with HEOM Hexiton Benchmark



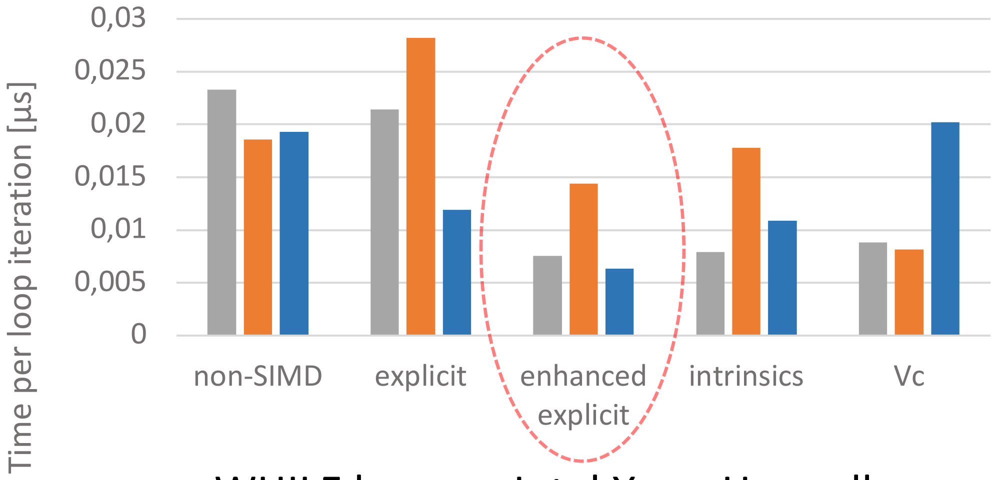
Enhanced Explicit SIMD

- Observation: limitations of directive-based vectorization of complex codes
  - is not effective with all compilers
  - does not allow to mix with scalar code
  - is hard to debug in case of erroneous output

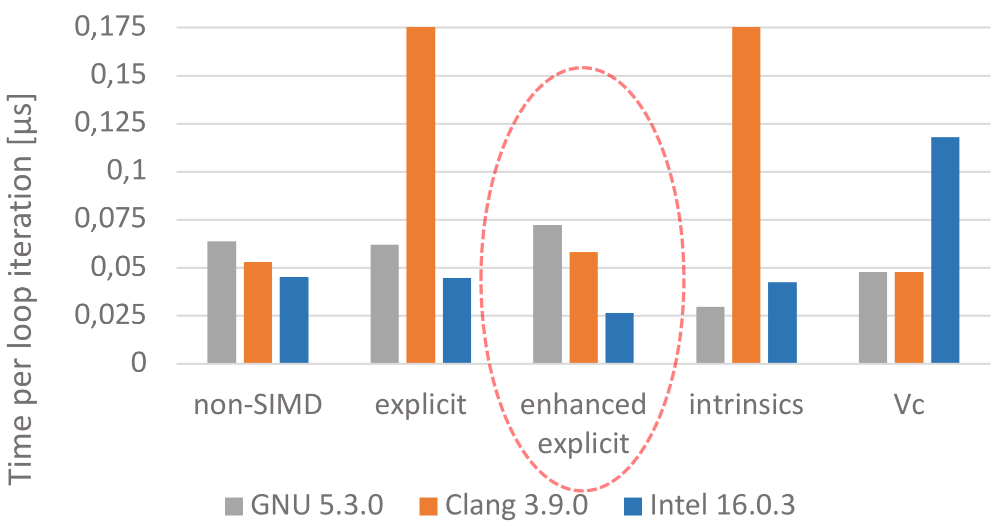
- Approach: combine high-level vector data types with OpenMP 4 SIMD directives to better promote explicit vectorization
  - generic, portable, expressive
  - vector data types as arguments to functions: enables SIMD functions
  - natural mixing of scalar and vector code

Enhanced Explicit SIMD

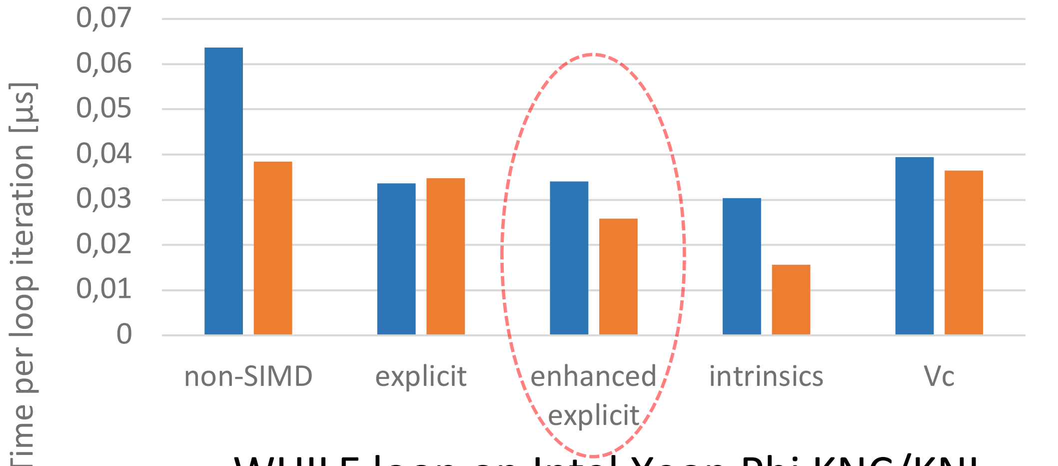
Conditional return on Intel Xeon Haswell



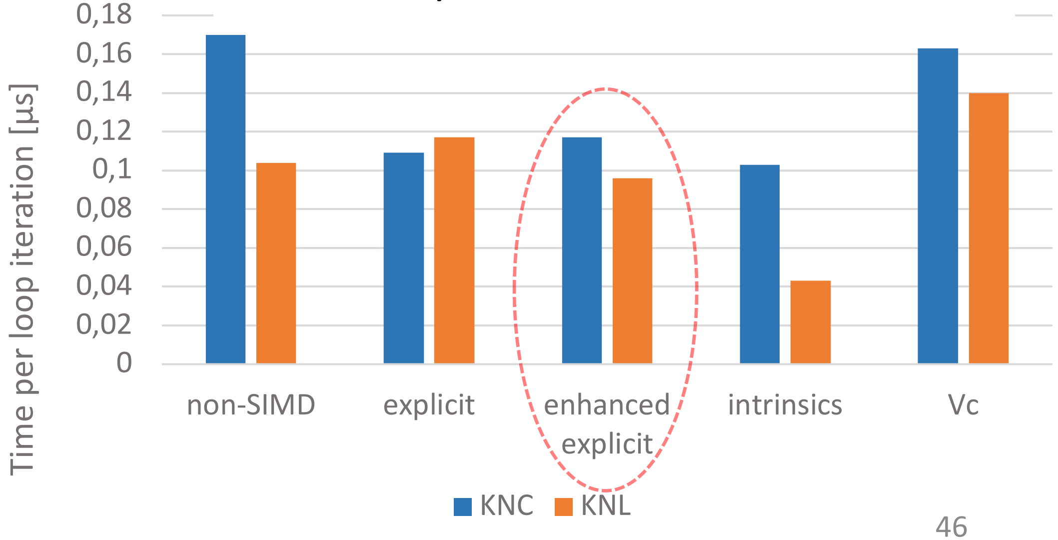
WHILE loop on Intel Xeon Haswell



Conditional return on Intel Xeon Phi KNC/KNL



WHILE loop on Intel Xeon Phi KNC/KNL

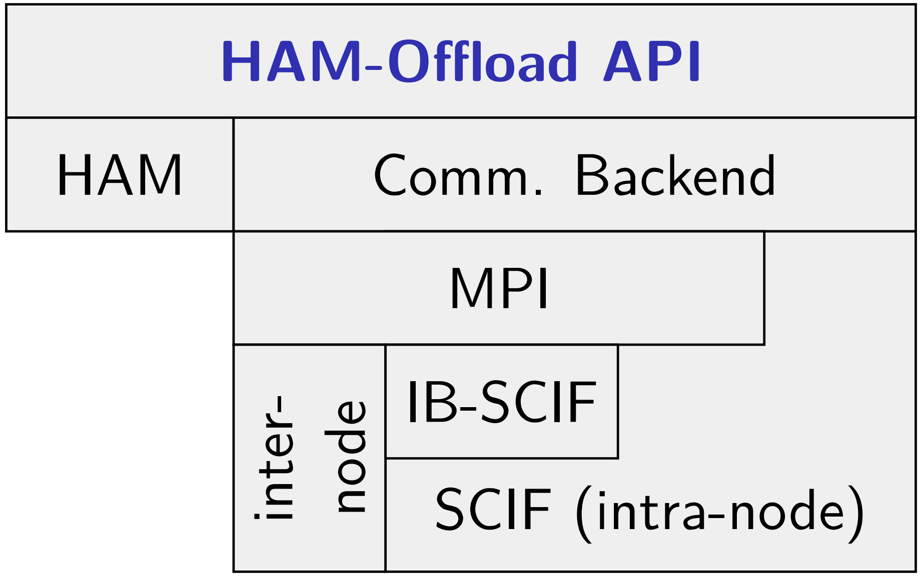


Fast Offload-Over-Fabric: HAM-Offload

- HAM - Heterogeneous Active Messages for efficient offloading
- Motivation:
  - Unify intra- and inter-node offloading
    - Inter-node offloading was/is not supported by Intel LEO and OpenMP 4
  - Avoid hybrid MPI+X programming
    - Arbitrary remote function call over fabric for ABI-conform platforms
- Minimized offload overhead
- No language/compiler extensions
- Principal broad usage area, e.g.
  - Cosmology (WALLS, Stephen Hawking Centre for Theoretical Cosmology, Cambridge, UK)
  - Atomistic thermo-dynamical simulation (GLAT, getlig&tar, ZIB),
  - Genomics: integration into SEQAN library (Free University Berlin) *(work in progress)*

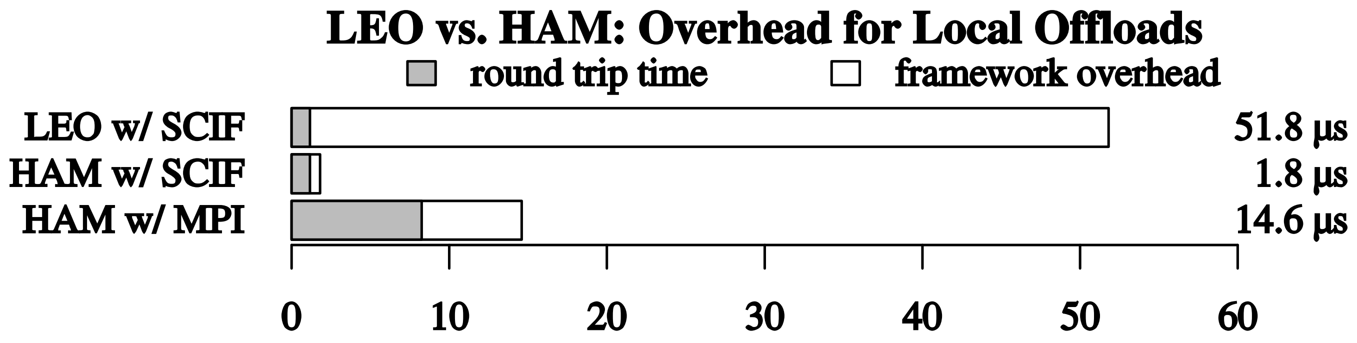
HAM on Github: https://github.com/noma/ham
noack@zib.de

HAM-Offload Architecture

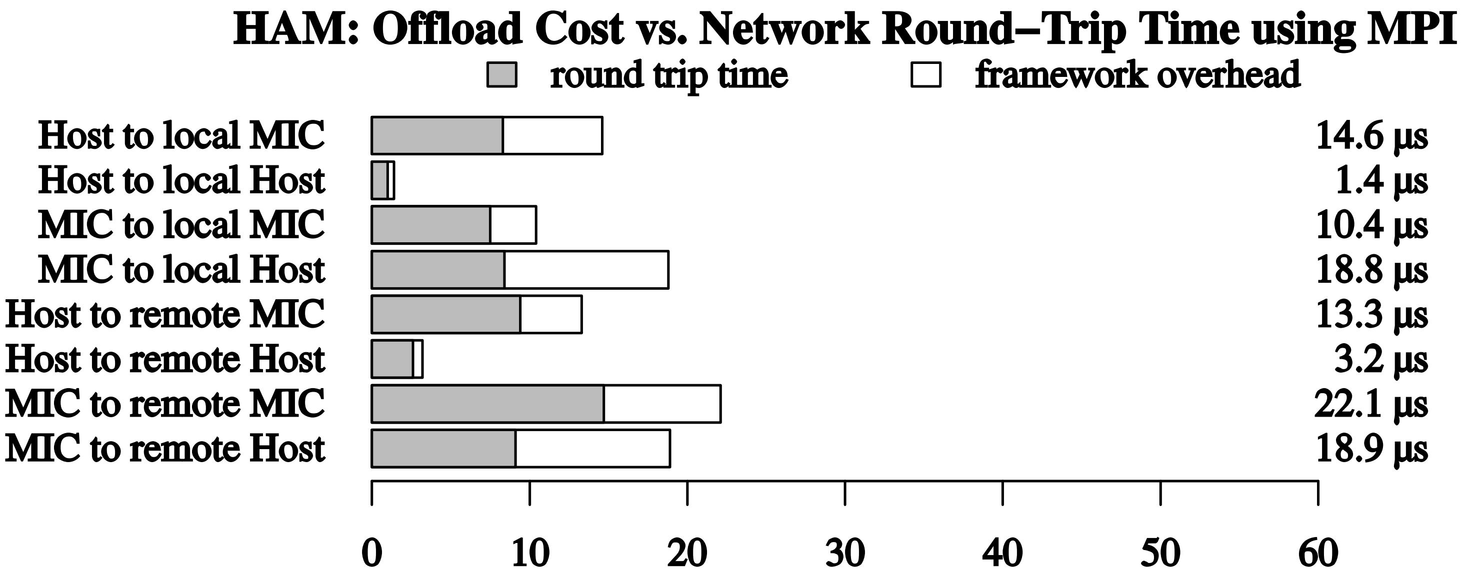


- HAM:
  - Symmetric exec model; different binaries
  - Provide code address translation between heetrogeneous exec‘s in *O(1)*
    - Use C++ type-system to generate message handlers and build translation data structures
- HAM-Offload C++ API

Micro Benchmarks



- 28.7x reduced offload cost compared with LEO
- still 3.5x with MPI

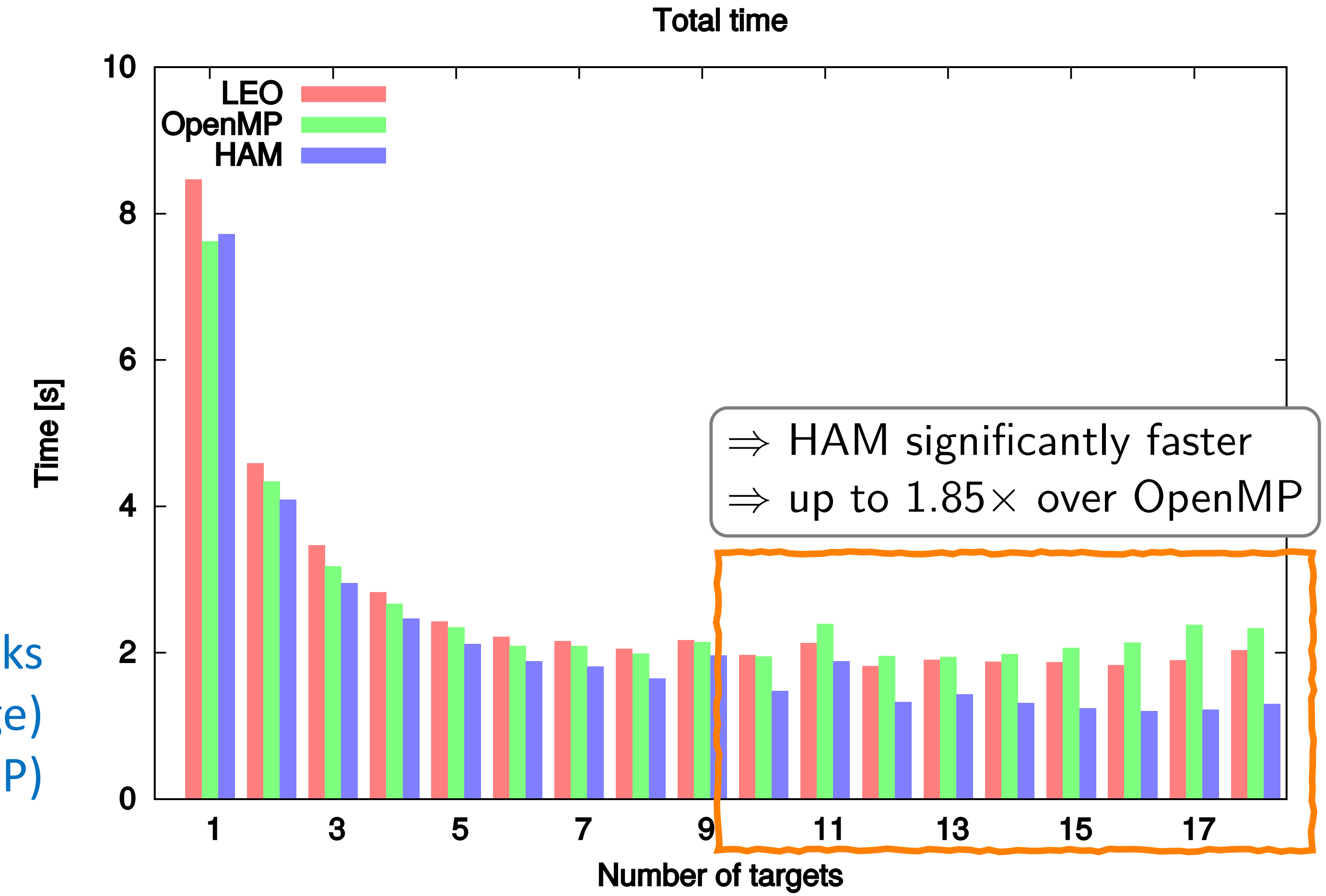


- no MPI communication path is slower than LEO with SCIF
- 19% - 55% framework overhead (0.4 µs – 10.4 µs)

Data obtained on Intel Grizzly Pass system, 2015-03-30

HAM Showcase: WALLS code

- Simulation of the evolution of a network of domain **walls** in the early universe
- Highly optimized for Intel Xeon/Phi (KNC)



Benchmarks of Offload Frameworks
on COSMOS supercomputer (Cambridge)
SGI UV2000 + 24x KNC cards (5110P)

Selected IPCC Publications

1. M. Noack, A. Reinefeld, T. Kramer, Th. Steinke; **DM-HEOM: A Portable and Scalable Solver-Framework for the Hierarchical Equations of Motion**; 19th IEEE Int. Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2018), Vancouver
2. M. Noack, F. Wende, G. Zitzlsberger, M. Klemm, Th. Steinke; **KART – A Runtime Compilation Library for Improving HPC Application Performance**, in IXPUG (ISC'17) Workshop Proceedings, Frankfurt am Main, Germany
3. M. Noack; **OpenCL in Scientific High Performance Computing – The Good, the Bad, and the Ugly**, IWOCL'17, Toronto, Canada
4. F. Wende, M. Marsman, Z. Zhao, J. Kim; **Porting VASP from MPI to MPI + OpenMP [SIMD]**, IWOMP 2017, p. 107, Vol. 8766, LNCS, New York, USA
5. H. Knoop, T. Gronemeier, M. Sühring, P. Steinbach, M. Noack, F. Wende, Th. Steinke, Ch. Knigge, S. Raasch, K. Ketelsen; **Porting the MPI-Parallelized LES Model PALM to Multi-GPU Systems and Many Integrated Core Processors**, Int. J. of Computational Science and Engineering, 2018, Vol. 17 N3, DOI: 10.1504/IJCSE.2017.10011396
6. F. Wende, M. Noack, Th. Steinke, M. Klemm, G. Zitzlsberger, C. J. Newburn; **Portable SIMD Performance with OpenMP 4.x Compiler Directives**, Euro-Par'16 Proceedings (LNCS), Toulouse, France
7. O. Krzikalla, F. Wende, M. Höhnerbach; **Dynamic SIMD Vector Lane Scheduling**, in IXPUG (ISC'16) Workshop Proceedings (LNCS), Frankfurt am Main, Germany
8. F. Wende, F. Cordes, Th. Steinke; **Concurrent Kernel Execution on Xeon Phi within Parallel Heterogeneous Workloads**, Euro-Par'14 Proceedings (LNCS), Porto, Portugal
9. M. Noack, F. Wende, Th. Steinke, F. Cordes; **A Unified Programming Model for Intra- and Inter-Node offloading on Xeon Phi Clusters**, SC'14 Proceedings, New Orleans, USA
10. F. Wende, Th. Steinke, M. Klemm, A. Reinefeld; **Concurrent Kernel Offloading**, in: *High Performance Parallelism Pearls, Vol. 1*, (ed. J. Reinders, J. Jeffers), Morgan Kaufman, Elsevier
11. M. Noack, F. Wende, K. D. Oertel; **OpenCL: There and Back Again**, in: *High Performance Parallelism Pearls, Vol. 2* (ed. J. Reinders, J. Jeffers), Morgan Kaufman, Elsevier