1) Introduction

2) KNL case studies for 4 HPC application codes developed in the Max Planck Society

   NSCouette: a pseudo-spectral DNS code for turbulence research (open source)

   GOEMHD3: a finite-differencing, domain-decomposition code for MHD simulations of the solar plasma

   VLASOV6d: a semi-lagrangian, domain-decomposition code for plasma simulations in 6D phase space (open source)

   ELPA: a popular, high-performance, scalable Eigensolver library (open source)

3) Conclusions
Introduction

About the Max Planck Computing and Data Facility (MPCDF)

- HPC and data center of the Max Planck Society
- formerly known as the RZG, located at the research campus in Garching near Munich, Germany
- MPCDF operates a Petaflop-class supercomputer and numerous mid-range clusters
About the Max Planck Computing and Data Facility (MPCDF)

- HPC and data center of the Max Planck Society
- formerly known as the RZG, located at the research campus in Garching near Munich, Germany
- MPCDF operates a Petaflop-class supercomputer and numerous mid-range clusters
- HPC applications group provides high-level support for the development, optimization, analysis and visualization of HPC applications … + *technology watch*

→ original contributions and long-term support for optimization and porting of HPC codes developed in the Max Planck Society, e.g.

- FHI-aims, OCTOPUS, S/PHI/nX, ESPResSo++ (materials and bio science), ELPA (eigensolver library)
- GENE, SeLaLib/VLASOV6D, SOLPS, GPEC, VMEC (plasma physics/fusion research)
- VERTEX, GOEMHD3, NSCOUETTE (astrophysics, comp. fluid dynamics)

Libinastemp: SIMD Intrinsics as template. A basic vectorization library (SSE, AVX, AVX512, …) in C++ by B. Bramas
https://gitlab.mpcdf.mpg.de/bbramas/inastemp
Methodology for Xeon Phi (KNL) assessment

Basic questions:

- porting effort for typical (“legacy”) HPC simulation codes: Fortran, C++, MPI (+OpenMP)
- performance of community codes (GROMACS, LAMMPS, GENE, ...)
- performance expectations Xeon Phi vs. Xeon
- sustained application performance per TCO: → energy, investment (?)
- extrapolation to Knights Hill, Skylake, ...

General approach:

- select (representative) codes which are promising for KNL
  memory-bound on Xeon, SIMD aware, hybrid MPI/OpenMP, memory scalable (RAM < 16 GB)
- start with a *minimally invasive approach*
  HBM API/memkind lib not (yet) employed → *quadrant/flat mode*
  sophisticated KNL-specific tuning not (yet) employed – except for ELPA
- *node-to-node comparison:*
  Xeon Phi 7210/30 @1.3 GHz, 64 cores (KNL), flat mode (or cache mode), quadrant mode
  Xeon E5-2698v3 @2.3 GHz, 32 cores (HSW)
- *fair comparison*: use optimal setup (e.g. MPI tasks, OpenMP threads) and code for each platform
NSCouette: direct numerical simulations (DNS) of incompressible Taylor-Couette flows

In collaboration with Marc Avila (ZARM, Bremen), B. Hof, J.M. Lopez (IST Austria), L. Shi (Max Planck Institute for Dynamics and Self-Organization, Göttingen)

NSCouette:
- high-performance, *pseudospectral* DNS code for solving the Navier-Stokes equation in axial geometry → turbulence in pipe flows, astrophysical flows, ...
- OpenSource: https://gitlab.mpcdf.mpg.de/mjr/nscouette
- FORTRAN with hybrid MPI-OpenMP parallelization
- memory bound on Xeon CPUs
- highly scalable on Xeon → fits into 16 GB MCDRAM per node


**Strong scalability of NSCouette on Xeon E5(v2)**

![Strongscaling-smalll](image1.png)
![Strongscaling-large](image2.png)

Basic Vtune analysis on Xeon Broadwell

Elapsed Time: 7.088s
GFLOPS: 46.009

CPU Utilization: 62.2%
Average CPU Usage: 24.880 Out of 40 logical CPUs

Memory Bound: 59.6%
Cache Bound: 69.3% of Clockticks
DRAM Bound: 28.8% of Clockticks
NUMA: 1% of Remote Accesses: 0.0%

FPU Utilization: 4.1%
GFLOPS: 46.009
Scalar GFLOPS: 2.664
Packed GFLOPS: 43.346

Top 5 hotspot loops (functions) by FPU usage
This section provides information for the most time consuming loops/functions with floating point operations.

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>FPU Utilization</th>
<th>Loop Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>[MKL FFT]@coDFTCoTad_Compact_Bad_v_12_d</td>
<td>16.077s</td>
<td>6.5%</td>
<td></td>
</tr>
<tr>
<td>[MKL FFT]@coDFTCoBatch_Compact_Bad_v_15_d</td>
<td>9.696s</td>
<td>6.0%</td>
<td></td>
</tr>
<tr>
<td>[MKL BLAS]@axx2,axx2axx</td>
<td>8.124s</td>
<td>2.0%</td>
<td></td>
</tr>
<tr>
<td>[Loop at line 105 in nonlinear]</td>
<td>5.764s</td>
<td>7.4%</td>
<td></td>
</tr>
<tr>
<td>[Others]</td>
<td>3.967s</td>
<td>2.0%</td>
<td>Vectorized (Body)</td>
</tr>
</tbody>
</table>

Collection and Platform Info
KNL experiences I: NSCouette

NSCouette (64,384,64) Xeon Phi (KNL) vs. Xeon (HSW), single node

HSW node: 32 cores, KNL node: 64 cores

<table>
<thead>
<tr>
<th>Runtime per timestep [s]</th>
<th>HSW (32 MPI)</th>
<th>KNL (64 MPIx3 OMP, DRAM)</th>
<th>KNL (64 MPIx1 OMP)</th>
<th>KNL (64 MPIx3 OMP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI transpose</td>
<td>2</td>
<td>5 (2.5)</td>
<td>3 (1.5)</td>
<td>3 (1.5)</td>
</tr>
<tr>
<td>Matrix-vector</td>
<td>6</td>
<td>9 (4.5)</td>
<td>5 (2.5)</td>
<td>5 (2.5)</td>
</tr>
<tr>
<td>Linear eqs.</td>
<td>6</td>
<td>9 (4.5)</td>
<td>5 (2.5)</td>
<td>5 (2.5)</td>
</tr>
<tr>
<td>FFT</td>
<td>4</td>
<td>8 (4)</td>
<td>4 (2)</td>
<td>4 (2)</td>
</tr>
</tbody>
</table>
NSCouette (512,256,512) Xeon Phi (KNL) vs. Xeon (HSW)

HSW node: 32 cores, KNL node: 64 cores
Observations:

- virtually no porting effort: KNL speedup wrt. HSW: 1.35
- main kernels (typical for a pseudospectral method):
  - single threaded 2D-FFTs from MKL perform (surprisingly) well
    → could be OpenMP-nested in NSCouette
  - “linear equations”: large OpenMP/vector loops with serial LAPACK (DGETRS) calls perform moderately well
    → more analysis needed, not a real hotspot in NSCouette
  - global matrix transposition (MPI_alltoall + local transpose) performs well on OPA
  - overlap of communication and computations performs well on KNL platform
    → implemented with an OpenMP thread (requires MPI_THREAD_SERIALIZED)

→ NSCouette is KNL-ready (→ https://gitlab.mpcdf.mpg.de/mjr/nsouette)
GOEMHD3: MHD code for simulating (solar) plasmas with large Reynolds numbers

In collaboration with Jörg Büchner, Jan Skala (Max Planck Institute for Solar System Research, Göttingen)

- 3-dimensional, time-explicit, finite-differencing MHD code
- standard 2-dimensional domain-decomposition (MPI-OpenMP)
- memory bound on Xeon CPUs
- highly scalable (weak and strong) → fits into 16 GB MCDRAM per node

Results on KNL:
- weak downscaling to few KNL node(s)
- virtually no porting effort
- one straightforward loop optimization (inspired by source-code inspection) → speed up on KNL, HSW
- overall KNL speedup wrt. HSW: 1 ... 1.3
- optimal KNL setups:
  16 MPI x 8 OpenMP or 32 MPI x 8 OpenMP
VLASOV6d: kinetic (fusion, astro, …) plasma simulations in 6-dimensional phase space

By Klaus Reuter (MPCDF), in collaboration with Katharina Kormann, Eric Sonnendrücker (Max Planck Institute for Plasma Physics, Garching)

- Vlasov-Poisson solver with semi-Langrangian advection in 6-dimensional (!) phase space
- based on OpenSource Fortran library SeLaLib (https://selalib.gforge.inria.fr)
- high-fidelity kinetic simulations of (magnetized) fusion plasmas
- newly developed 6-dimensional domain decomposition (non-trivial: “curse of dimensions”) shows excellent scalability
- highly optimized (MPI, OpenMP, SIMD) on Xeon CPUs

Strong scalability of VLASOV6d on Xeon E5(v2)
Results on KNL:

- code compilation went smoothly thanks to the familiar toolchain
- on the KNL, the code benefits from optimizations for the host and vice versa
  → no KNL-specific optimizations were introduced
- using the 16 GB MCDRAM is the key to get good performance

Comparing the best setups on each machine:

- HSW: 16 MPI tasks x 4 OpenMP threads
  run_time = 30.0 s
- KNL: 64 MPI tasks x 4 OpenMP threads
  run_time = 22.5 s
  → run on KNL is faster up to a factor of 1.33

→ Vlasov6d is KNL ready
  large-scale tests and applications on Marconi@CINECA

Numerical setup

- grid resolution: $32^6$ points (8 GB in double precision for $f(x,v)$ only)
- 7-pt Lagrange interpolation → halos of width 3 are exchanged
- 5 time steps
ELPA: a high-performance direct Eigensolver library for dense matrices

By Andreas Marek (MPCDF), in collaboration with the ELPA consortium, credits to A. Vazquez-Mayagoitia (ALCF)

- ELPA is a high-performance library for the massively parallel solution of symmetric (or Hermitian), eigenvalue problems for dense matrices
- in widespread use, e.g. in electronic structure codes
- OpenSource → https://elpa.mpcdf.mpg.de
- replacement for ScaLAPACK routines
  - pdsyed, pzheevd (all eigenvectors)
  - pdsyevr pzheevr (subset of eigenvectors)
- relies on optimized kernels (based on intrinsics) for all major HPC architectures:
  - SSE, AVX, AVX2, BlueGene/P/Q, VSX, AMD bulldozer, ..


ELPA on KNL

- AVX-512 kernels developed in advance using Intel emulator software (SDE)
  → development straightforward thanks to SDE and Intel's Intrinsic Guide (list of all intrinsics)
  → AVX-512 on KNL misses some instructions (AVX512DQ) of the Xeon-line (→ more code, higher CPI)
- ELPA kernels run roughly at 25% of peak performance
  • (red line: 1.3 GHz * 32 Flops/Cycle * #cores)

Performance evolution of kernels on x86:
- SSE→AVX: ~1.5x (per core)
- AVX→AVX2: ~1.5x (per core)
- AVX2 → AVX-512: ~1.5x (per core)
ELPA on KNL

- overall node performance wrt. Xeon Haswell dual-socket node: 0.5 ... 0.9, depending on matrix sizes, number of eigenvectors
- performance tuning in progress
  - it seems that ELPA suffers from L2 cache misses on KNL (25% for MCDRAM, 14% for DRAM ???)
  - hybrid MPI-OpenMP code not yet sufficiently efficient
ELPA on KNL

- scalability up to 260k KNL cores (64 MPI ranks per node) demonstrated on Theta@Argonne

In collaboration with Alvaro Vazquez-Mayagoitia (Argonne National Laboratory)

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH1135
KNL experiences IV: ELPA

Observations

General look and feel:

- it was straightforward to compile and run ELPA on KNL
  first results after a few minutes

- some initial surprises:
  compiler performance: GNU (gfortran + gcc) 6.1.0 was faster (up to 20%) than Intel (17.0.098)
  documentation: it was not easy to find details like latency of MCDRAM, DRAM, …

Tools

- Intel SDE: fantastic tool, enabled writing and testing intrinsic AVX-512 kernels, before having access to real hardware

- Intel Advisor is a very useful tool (and has improved a lot recently)
  but only with Intel compiler; some integration in vtune would be nice

- Intel Vtune is a very useful tool (and has improved a lot recently) to get a first overview
  but we perceive some “information gap” between Vtune (high-level code analysis, low-level hardware information) and Advisor (detailed code analysis, most relevant metrics for advanced performance tuning):

  - how much percentage of code is vectorized (total + subroutine level)
  - what is the performance in GFlop/s (total + subroutine level)
  - finding the arithmetic intensity (total + subroutine level)
  - memory bandwidths used and data amount transferred (total + subroutine level)
  - which parts of the code are memory bound or compute bound on KNL

=> this is why we like tools like likwid
General conclusions and remarks

- node-to-node performance ratio KNL:HSW of 1 : 1 … 1.5 : 1 relatively easy to achieve for a certain class of codes: (memory-bound, hybrid, …?) without significant code modifications

- optimizations for KNL speed up (the same!) code on Xeon CPUs
  → we need (even better) profiling/performance tools
  → multidimensional arrays in “legacy” codes: memory alignment is a challenge, HBM API practical?

- KNL can be a very attractive platform depending on its – assumed – cost-efficiency

- computational scientists are usually not impressed by a 30% performance gain
  => how to position KNL’s cost-efficiency to computational scientists (i.e. users and decision makers) ?

  => what about KNL vs. Skylake ?

  => what about popular community codes ?

Acknowledgments

- Intel, Fujitsu/Microstaxx, Argonne National Lab
- V. Blum (Duke U.), A. Vazquez-Mayagoitia (ALCF)
- MPCDF systems and operations team