Porting the ELPA library to the KNL architecture

MIC Programming Workshop @ LRZ

Dr. Andreas Marek

together with

Dr. Hermann Lederer, Dr. Pavel Kus, Dr. Lorenz Hüdepohl
Outline

- What is the ELPA library
- A roofline model for KNL
- Writing AVX-512 kernels
- Going to many nodes
- KNL experiences
- Conclusions
The ELPA library: 

**Eigenvalue solvers for Petaflop applications**

- ELPA is a high-performance library for the massively parallel solution of dense, symmetric (hermetian) eigenvalue problems
- replacement for Scalapack routines (pdsyevd, pzheevd, pdsyevr, pzheevr)
- widespread used, e.g. in electronic structure codes
- opensource (see https://elpa.mpcdf.mpg.de)
- available with many linux distributions (Debian, Fedora, Suse etc..)
- supports a large variety of platforms (X86, X86_64, OpenPower, BG P/Q, GPUs, KNL)
- achieved already 0.3 PFLOPS/s on 294k cores (BG/P FZJ Juelich in 2011)
The ELPA library: Eigenvalue solvers for Petaflop applications

=> used in many HPC centers worldwide (Juelich, Cineca, ORNL, LLNL …)

=> nowadays used in most material science codes (FHI-aims, Quantum Esspresso, VASP, OpenMX…)

=> Intel is interested in integrating ELPA in it’s MKL library


The ELPA library: Eigenvalue solvers for Petaflop applications

=> used in many HPC centers worldwide (Juelich, Cineca, ORNL, LLNL …)

=> nowadays used in most material science codes (FHI-aims, Quantum Espresso, VASP, OpenMX…)

=> Intel is interested in integrating ELPA in it`s MKL library


The development of the ELPA library started under the lead of MPCDF in 2008; it has been funded by a BMBF project 01IH08007 from Dec. 2008 - Nov. 2011.

ELPA is maintained and developed at MPCDF.

Since Feb. 2016 a new BMBF project 01IH15001 is funding the latest developments of ELPA (including the porting to KNL).
The algorithm in a nutshell - I

1. step: - transformation of matrix to banded matrix
- store information for later back-transform
The algorithm in a nutshell - II

2. step: - transformation to tridiagonal matrix
- store information for later back-transform
The algorithm in a nutshell - III

3. step: - solve eigenvalue problem
The algorithm in a nutshell - IV

4. step: - back-transformation to banded matrix
The algorithm in a nutshell - V

5. step: - back-transformation to full matrix
The algorithm in a nutshell - VI

The backtransformation is computationally expensive and relies on optimized kernels.
Prior to porting ELPA (and especially judging the performance) we want to have a feeling on what you can expect/get on a new architecture

=> roofline model is helpful

very nice work of NERSC in the NESAP project

Typical arithmetic intensities in BLAS routines:

BLAS Level 2 (Matrix-Vector): \( \sim O(1) \)
BLAS Level 3 (Matrix-Matrix): \( \sim O(n) \)

Prior to porting ELPA (and especially judging the performance) we wanted to have a feeling on what you can expect/get on a new architecture

➢ roofline model is helpful

very nice work of NERSC in the NESAP project

Typical arithmetic intensities in BLAS routines:

BLAS Level 2 (Matrix-Vector): $\sim O(1)$
BLAS Level 3 (Matrix-Matrix): $\sim O(n)$

Porting the kernels to AVX-512 - using Intel`s SDE

- we have used Intel`s „Software Development Emulator“ (http://software.intel.com/en-us/articles/intel-software-development-emulator) on a standard Haswell node to develop the AVX-512 kernels

Porting the kernels to AVX-512 - using Intel`s SDE

- we have used Intel`s “Software Development Emulator” (http://software.intel.com/en-us/articles/intel-software-development-emulator) on a standard Haswell node to develop the AVX-512 kernels
Porting the kernels to AVX-512 - using Intel`s SDE

- we have used Intel`s „Software Development Emulator“ (http://software.intel.com/en-us/articles/intel-software-development-emulator) on a standard Haswell node to develop the AVX-512 kernels

=> this allowed us to start with the development of the AVX-512 kernels before the hardware was available

=> this lead to ~4000 lines of AVX-512 intrinsics code
Depending on the setup, 30% to 75% of runtime is spend in the back-transformation kernels

Example code:

```c
__mm512d h1, h2, q1, x1, y1, q2, x2, y2;

for(i = 2; i < nb; i++)
{
    h1 = __mm512_set1_pd(hh[i-1]);
    h2 = __mm512_set1_pd(hh[ldh+i]);

    q1 = __mm512_load_pd(&q[i*ldq]);
    x1 = __mm512_FMA_pd(q1, h1, x1);
    y1 = __mm512_FMA_pd(q1, h2, y1);

    q2 = __mm512_load_pd(&q[(i*ldq)+8]);
    x2 = __mm512_FMA_pd(q2, h1, x2);
    y2 = __mm512_FMA_pd(q2, h2, y2);
    ...
```

A problem turned out to be a compiler independent alignment of the data (and removing the effort from the users that build the ELPA library)

=> we had to use „posix_memalign“
   For the data allocation and to ensure correct striding
Porting the kernels to AVX-512 - II

Programming AVX-512 intrinsics showed a little surprise:

On KNL some instructions are missing (compared to "old" Xeons and upcoming SkyLake)

=> more code needed to program around
=> higher CPI

Reminder: AVX-512 – KNL and SKX

- KNL and SkyLake server architecture share a large set of instructions
  - but sets are not identical
- Subsets are represented by individual feature flags (CPUID)
Programming AVX-512 intrinsics showed a little surprise:

On KNL some instructions are missing (AVX-512DQ of normal xeon line)

=> more code needed to program around
=> higher CPI

Reminder:  AVX-512 – KNL and SKX

- KNL and SkyLake server architecture share a large set of instructions
  - but sets are not identical
- Subsets are represented by individual feature flags (CPUID)
Porting the kernels to AVX-512 - IV

- ELPA measurements are ~ ¼ of theoretical value (ELPA is memory bound)
- we still work on improving this value; new version of Intel Advisor allows to measure the roofline directly (not yet done)

Theoretical limit:
1.3 GHz * 32 Flops/cycle * #cores
= 2.6 TFLOPS/s @ 64 cores

Speedup AVX2 → AVX512 ~ 1.5x - 1.6x

double precision values!

ELPA @ 64 cores: ~520 GFLOPS/s
Porting the kernels to AVX-512 - V

Theoretical limit:
1.3 GHz * 32 Flops/cycle * #cores
= 2.6 TFLOPS/s @ 64 cores
Speedup AVX2 → AVX512 ~ 1.5x-1.6x

memory bound
=> we can not expect peak

Applying the roofline performance model to the Intel Xeon Phi Knights Landing processor“, D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. Malas, J.-L. Vay, and H. Vincenti, Lawrence Berkley National Laboratory

- ELPA measurments are ~ ¼ of theoretical value (ELPA is memory bound)
- we still work on improving this value; new version of Intel Advisor allows to measure the roofline directly (not yet done)
Porting the kernels to AVX-512 - VI

Kernel ~ 44% of r.t.

Kernel ~ 20% of r.t.

~ 1.3x

~ 1.5x - 1.6x for kernels
AVX2 → AVX512
Going to many nodes...

- In a collaboration between the ELPA-AEO and the ELSI project (http://wordpress.elsi-interchange.org) it was possible to do first test of the ELPA library on the KNL system „THETA“ of the Argonne National Lab

- All benchmarks have been performed by Vazquez-Mayagoit Alvaro (Argonne National Lab)

  => successful runs on up to 200,000 cores and a matrix size >1,000,000 could be demonstrated

  => we are still trying to improve the scaling on KNL
Going to many nodes...

Preliminary results

ELPA2 double-precision real, KNL AVx-512 optimized

dense diagonalization of 1048k x 1048k matrix (prop. world record)

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

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

Intel MIC Programming Workshop @ LRZ, Dr. Andreas Marek (MPCDF)
KNL experiences - Summary I

General look and feel:

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

- architecture and software feels like a good old friend

- some initial surprises:
  a) compiler performance: in some setups GNU (gfortran + gcc) 6.1.0 was faster (up to 20%) than Intel (17.0.098); We have not checked again with newer versions of Intel 2017 or Intel 2018 beta. Probably this is not the case anymore

  b) documentation: it was not easy to find details like latency of MCDRAM (~190 ns), DRAM (~120 ns), ...

  c) missing instructions on AVX-512 intrinsics
KNL experiences – Summary II

Tools:

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

- **Intel Advisor**: very useful tool (and has improved a lot recently). Latest ability to create roofline plots helps extremely (we could only use that up to now on Haswell systems)

- **Intel Vtune**: is also very useful with interesting metrics (on Xeon processors). We found the advanced metrics on KNL less useful

- We perceive an "information gap" between Vtune (high-level code analysis, low-level hardware information) and Advisor (detailed code analysis, most relevant metrics for advanced performance tuning):
It would have been helpful if we could have easily obtained the following information:

- 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
It would have been helpful if we could have easily obtained the following information:

- 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

Good perspective: with the new version of Intel Advisor (which we sadly do not yet have on our KNL system) a lot of these points are addressed!

=> In the near future we will be able to answer these questions for ELPA on KNL systems
Conclusions

➢ As always: roofline analysis extremely helpful for performance optimisations
➢ We have ported the ELPA library to run on KNL clusters
➢ Writing AVX-512 intrinsic kernels proved to be necessary
➢ Successfull runs on up to ~200,000 cores have been demonstrated
➢ We are still working on improving the performance
  - detailed roofline analysis with new Advisor
  - cache blocking
  - but also trying libxsmm (dgemm sizes get small in ELPA during iteration)
➢ Some of the Intel tools (SDE, Advisor) were really helpful for working on KNL
➢ The work on KNL is not yet finished ...
Questions?

Thank you for your attention