Is Arm ready for HPC?

An evaluation of the HPC software ecosystem

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MONT-BLANC

Fabio Banchelli
fabio.banchelli@bsc.es
/fabio-banchelli

Daniel Ruiz
daniel.ruiz@bsc.es
/daniruizm

Ying Hao Xu Lin
ying.xulin@bsc.es
/yinghao-xu

Filippo Mantovani
filippo.mantovani@bsc.es
/filimanto

Motivation

In recent years, the HPC community has increasingly grown its interest towards the Arm architecture with research projects targeting primarily the installation of Arm-based clusters. State of the art research project examples are the European Mont-Blanc, the Japanese Post-K, and the UK's GW4/EPSRC.

Objective of our study:

- ThunderX, 2x48 Arm cores @ 2.0GHz (designed by Cavium), 32GB, 40GbE
- AMD Opteron A1170 "Seattle", 1x8 cores @ 2.0GHz (Arm Cortex-A57), 16GB, 10GbE
- NVIDIA Jetson-TX1, 1x4 cores, (Arm Cortex-A57), 4GB, 1GbE

Availability of Arm-based HPC platforms | Mature software ecosystem | Acceptance of Arm technology by HPC community

Preliminary evaluation of Arm system software

- Arm HPC Compiler
- Arm Performance Libraries
- Complex HPC production code

QuantumESPRESSO
WRF
FEniCS

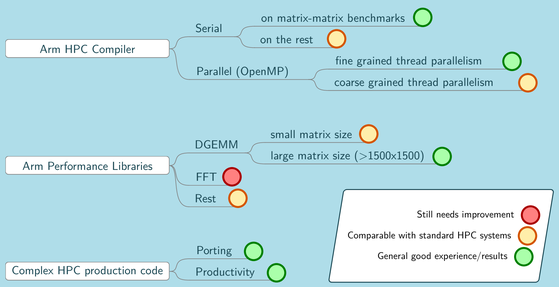
Part of Student Cluster Competition 2016/2017

Conclusions

In general, the Arm software ecosystem has been proven mature enough for running on Arm-based clusters without major restrictions even in an educational project like SCC. Performance wise, the results are in-line with other mainstream comparable softwares (e.g. GCC, ATLAS, OpenBLAS) with two exceptions:

i) the Arm tools allow to reach better performance in matrix-matrix operations
ii) further optimization work should be done for the FFT part.

The Allinea development tools ecosystem looks solid as well, however we did not test it here. Instead we took advantage of BSC performance analysis tools.



Future work

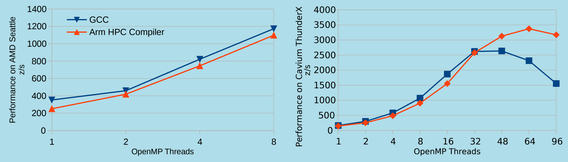
i) Detailed study of # of instructions vs IPC with different compilers
ii) "Time to Solution" vs "Energy to Solution" using different compilers and math libraries
iii) Integrate results with mixed approach Arm PL + FFTW
iv) Extend the evaluation of WRF on different/lower number of cores with JetsonTX1

Compilers

We performed a comparison between GCC 7.1.0 and the Arm HPC Compiler 1.3.0. Both have been evaluated using the default OpenMP runtime: `libgomp` for GCC and `libomp` for the Arm HPC Compiler. C, C++ and Fortran compilers are included in both suites. We use `-O3 -mcpu={cortex-a57,thunderx}` depending on the platforms where we run our tests. Tests are repeated 10 times: variation is below 5% and only average is plotted.

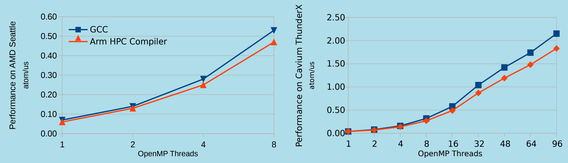
Lulesh

CORAL mini-app for hydrodynamics written in C++. It uses OpenMP for intra-node parallelism, featuring fine-grained parallel regions.



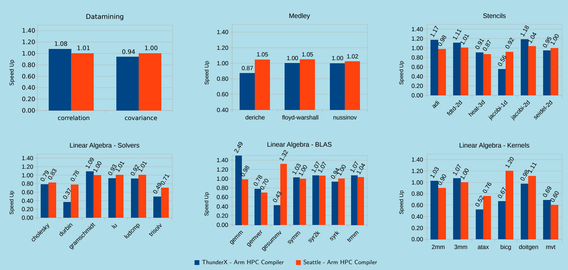
CoMD

ExMatEx mini-app for molecular dynamics written in C. It uses OpenMP for intra-node parallelism, featuring coarse-grained parallel regions.



PolyBench

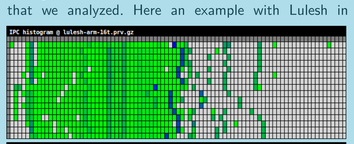
Benchmark suite written in C including four different kinds of kernels: linear algebra, stencil, data mining and medley. We executed all of them on a single core/single thread and compared relative performance of Arm HPC compiler vs GCC on two platforms.



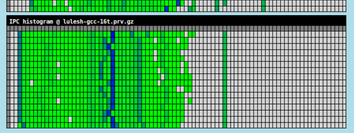
Observations

i) Serial code: better performance using Arm HPC Compiler with matrix-matrix operations. On average we reproduce http://bit.ly/gcc_vs_llvm
ii) Parallel code: better performance at higher number of cores and fine grained parallelism (e.g. Lulesh) using Arm HPC Compiler.
iii) Parallel code: similar tendency with more coarse grain parallelism (e.g. CoMD), but GCC performs 20% better.
iv) Arm HPC Compiler seems to handle more efficiently thread creation/destruction
v) Arm HPC Compiler generates in general more instructions than GCC, that run at higher IPC in all systems that we analyzed. Here an example with Lulesh in ThunderX:

Arm HPC Compiler
Total Instr. 4.36 x 10^9
Total Exec. Time 51.35 s



GCC
Total Instr. 2.99 x 10^9
Total Exec. Time 48.27 s

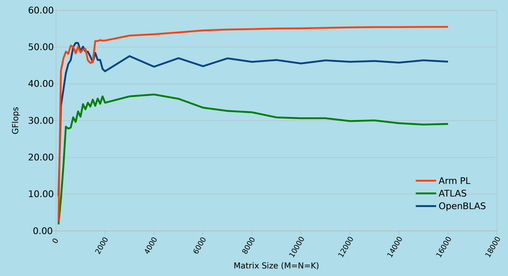


Math libraries

BLAS, LAPACK and FFT are a collection of low-level math routines commonly used by the scientific community. The Arm Performance Libraries implement them and are maintained, supported and tuned by Arm for a wide range of Arm-based SoCs using OpenMP for their parallel version. We compared them with OpenBLAS and ATLAS first in a DGEMM micro-benchmark and then in a large HPC production code, QuantumESPRESSO.

DGEMM micro-kernel

Performance obtained with the Arm Performance Libraries 2.2.0 is comparable with the one offered by other optimized libraries for small matrix sizes. A significant gain in performance can be noticed with matrix sizes bigger than 1500x1500. Here an example in AMD Seattle, where we reach 87% of the peak performance with a matrix 16000x16000.

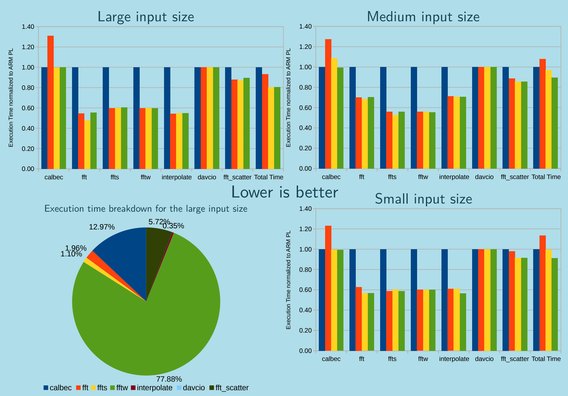


Quantum ESPRESSO

Quantum Espresso is an integrated suite of codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves and pseudopotentials. It is mostly written in Fortran and uses MPI and (optionally) OpenMP to exploit parallelism.

Since its implementation uses BLAS, LAPACK and FFT functions, it is a good candidate for the evaluation of the Arm Performance Libraries in a real HPC application.
We evaluated the execution time of the MPI only code running the `pwscf-small` input set `http://bit.ly/pwscf-small` on 8 cores of our AMD Seattle SoC. As in the previous DGEMM benchmark, we compare executions using:

i) Arm Performance Libraries 2.2.0
ii) ATLAS 3.11.39 + FFTW 3.3.6
iii) OpenBLAS 0.2.20 + FFTW 3.3.6
iv) Arm Performance Libraries 2.2.0 + FFTW 3.3.6

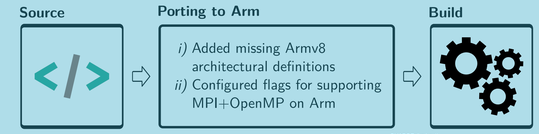


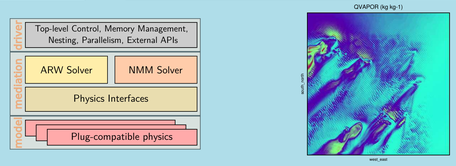
Execution time breakdown for the large input size

We plot execution time for each of the most relevant functions of the code normalized to Arm PL. `calbec` is the only function making heavy use of matrix-matrix multiplication, `davcio` performs mostly IO routines while `fft_scatter` handles communications via MPI. The rest of the routines call FFT functions. As expected in view of the DGEMM micro-kernel results, in the functions with matrix-matrix operations Arm PL performs better. Further optimization work should be done for the FFT part. Combining Arm PL and FFTW allow us to reach the best overall execution time.

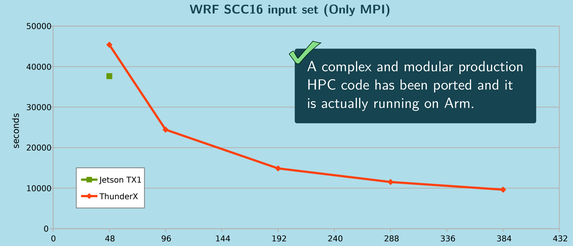
WRF

Weather Research and Forecasting (WRF) is a next-generation mesoscale numerical weather prediction system. It is designed for both atmospheric research and operational forecasting needs. WRF includes codes in Fortran and C and it is implemented for taking advantage of shared memory as well as message passing parallel programming environment.





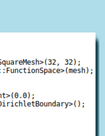
WRF SCC16 input set (Only MPI)

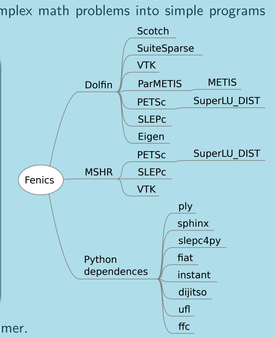


FEniCS

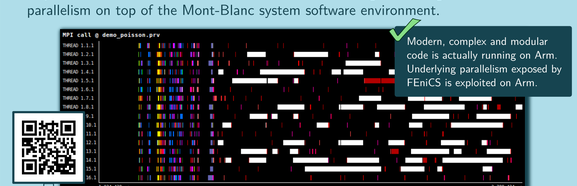
A popular computing platform for partial differential equations. FEniCS supports Python and C++ languages and runs on multiple platforms ranging from laptops to high-performance clusters. It transforms complex math problems into simple programs with few lines of code.

```
int main()
{
  // Create mesh and function space
  auto mesh = std::make_shared<UnitSquareMesh>(32, 32);
  auto V = std::make_shared<Poisson::FunctionSpace>(mesh);

  // Define boundary condition
  auto u0 = std::make_shared<Constant>(0.0);
  auto boundary = std::make_shared<DirichletBoundary>();
  DirichletBC bc(V, u0, boundary);

  // Define variational forms
  Poisson::BilinearForm a(V, V);
  Poisson::LinearForm L(V);
  auto f = std::make_shared<Source>();
  auto g = std::make_shared<dUdN>();
  L.f = f;
  L.g = g;

  // Compute solution
  Function u(V);
  solve(a == L, u, bc);

  // Save solution in VTK format
  File file("poisson.pvd");
  file << u;

  return 0;
}
```





Parallelization is transparent to the programmer.
The distribution package does not support Arm, so rebuilding is required. It depends on packages, but building process on Arm is not supported by the developers. Dependencies have been rebuilt from source taking advantage of all available parallelism on top of the Mont-Blanc system software environment.



Scan it and learn how to study advanced parallel applications with Paraver!

Designed with: INKSCAPE