D2.3
Report and best practices on porting of the mini-applications to the ExaNoDe architecture

Workpackage:	2	Co-Design for Exa-scale HPC systems	
Author(s):	Aditya Kela		JUELICH
	Dirk Pleiter		JUELICH
Authorized by	Petar Radojkovic		BSC
Reviewer	Paul Carpenter		BSC
Reviewer	Emre Ozer		ARM
Reviewer	Antoniu Pop		UOM
Dissemination Level	Public		

Date	Author	Comments	Version	Status
2018-03-20	Dirk Pleiter	Initial draft	V0.0	Draft
2018-03-26	Aditya Kela	Updated draft	V0.1	Draft
2018-04-03	Aditya Kela	Updated draft	V0.2	Draft
2018-04-05	Dirk Pleiter	Consolidated draft	V0.3	Draft
2018-04-06	Dirk Pleiter	Final version	V1.0	Final
2018-04-14	Dirk Pleiter	Final version based on internal reviewers comments	V1.1	Final

Executive Summary

The ExaNoDe project adopted the mini-application approach for managing the challenge of porting real-life applications to a new architecture for performance evaluation purposes. Mini-applications are simplified versions of the full applications, where the relevant performance features of the application are maintained, but the total number of lines-of-code is significantly reduced.

With this deliverable we provide the implementation of the four mini-applications, which are based on the four applications selected in Deliverable D2.1, as well as a report about the implementation and porting of the mini-applications. Furthermore, initial performance numbers have been created on an ExaNoDe prototype system based on Trenz boards.

The following four mini-apps have been implemented and ported:

- BQCD: A massively-parallel application for simulating Quantum Chromodynamics, which is the theory for strong interactions.
- HydroC: An application-based benchmark mimicking a 2-dimensional CFD code based on the Finite Volume Method
- KKRnano: A highly scalable material science application based on the Density Functional Theory (DFT) method.
- MiniFE: A mini-application implementing an Implicit Finite Elements method in 3 dimensions.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

A mini-application [Heroux2009] is a reduced but self-contained application extracted from a real large-scale application with the objective of rapidly exploring the parameter space of the real application by quickly traversing parameter choices for hardware platforms, runtime and compile-time environments. Mini-applications can be thought of as occupying a middle ground between benchmark suites like LINPACK (HPL) [Dongarra1999] and full-scale applications, which are better suited for testing near-production systems. [Heroux2009] provides a list of categories that a mini-application aids with:

- Interaction with external research communities via an open-source requirement for the mini-application.
- Simulator for the study of processor, memory and network architectures
- Early node architecture studies
- Network scaling studies
- Study of new languages and programming models
- Compiler tuning

A mini-application is not just a stripped down version of the large-scale application but is a good representation of the parameter space traversed by the large-scale application. Our efforts to develop these mini-applications have started with the original application and have been to cut out code that was not necessary for the required parameter space performance analysis.

In this deliverable we report on the mini-applications based on the applications selected in deliverable D2.1. This report is organised as follows: We start by describing the benchmarking environment in Section 2. In Section 3 the different mini-applications are described in more detail and references to the corresponding git repositories are provided. Next we present in Section 4 performance results obtained on a single node of the Trenz-based prototype as well as a state-of-the-art supercomputer. Finally, concluding remarks are provided in Section 5.

2 JUBE benchmarking framework

The JUBE benchmarking environment [JUBE] helps performing and analyzing benchmarks in a systematic way. It provides a script-based automated framework to easily create benchmark sets by:

- Choosing platforms
- Configuring for the chosen platform
- Compiling
- Running a benchmark suite
- Data pre- and post-processing
- Storage

In addition, different benchmarks with different parameters are created automatically. For example, if parameter *A* takes values in *{a1, a2}* and parameter *B* takes values in *{b1, b2}*, benchmarking can be done for the combination: *{(a1, b1), (a1, b2), (a2, b1), (a2, b2)}.*

Each mini-application is a combination of the source code plus a JUBE folder containing:

- template makefiles

- template inputfiles
- specs-<mini-app>.xml
- <mini-app>.xml

specs-<mini-app>.xml: This file contains the parameters that get substituted in the template input-files and the template make files.

<mini-app>.xml: This file substitutes the parameter values from the spec-<mini-app>.xml file to the template files and combines the source with these substituted files into a subfolder *inside* the JUBE directory. As a second step, it runs the code. It analyses the results as the third step.

3 Mini-applications

In this section, we describe each of the four mini-applications:

1. BQCD
2. HydroC
3. KKRnano
4. MiniFE

Also, their contents are compared to their corresponding overall large-scale application. We also detail the efforts made to port the applications to the ARMv8 architecture. A link to each of the repository is provided.

3.1 BQCD

BQCD (Berlin Quantum ChromoDynamics program) is a hybrid Monte-Carlo code that simulates Quantum Chromodynamics on a lattice (LQCD) with dynamical Wilson-type fermions [Nakamura2010]. It is written in Fortran 90 and uses MPI and OpenMP for parallelisation. A relatively simple kernel, where mainly sparse matrix-vector multiplications are performed, dominates overall performance. The application is part of the UEABS [UEABS] and one of the PRACE-3IP benchmark applications. It is currently used for large-scale projects on different Tier-1 systems. LQCD is on multiple future research roadmaps and is an application area that is in need for exascale computing resources [Brower2017].

The BQCD mini-app is the complete stand-alone complex arithmetic conjugate-gradient method stripped down from the original complex arithmetic Hybrid Monte-Carlo method from the BQCD code. The mini-application includes a SIMD version of the conjugate gradient method for the ARMv8 architecture making use of the ARM NEON compiler intrinsics. It is important to note that the multiplication of two Fortran complex numbers is not done in SIMD. In order to implement SIMD vectorization, the structure of the complex arrays was changed from the standard sequence of alternating real and imaginary parts of each complex number to a layout that involved a collection of real numbers with the length of the collection corresponding to the SIMD width followed by the corresponding imaginary numbers (also with a length of the SIMD width) with such a continuing alternation.

In the mini-application, a C pre-processor was employed for conditional compilation and for macro processing. All the BQCD source files have the .F90 suffix and the suffix for the pre-processed file is .f90. The .f90 files will be the files that are compiled. This extra step was chosen to check the result of the pre-processing. Macro names are almost always upper-case (mixed case sometimes). The Fortran code is lower-case. The mini-application can be compiled either for single precision or for double precision arithmetic.

Repository: https://gitlab.version.fz-juelich.de/exanode/miniapp-bqcd.git

3.2 HydroC

HydroC[1] is already a mini-application, being a simplified version of the astrophysical code RAMSES. It is considered here as it represents a large class of relevant codes. It is a 2-dimensional CFD using the Finite Volume Method with a Godunov's scheme and a Riemann solver at each interface on a regular 2D mesh. The code basis is O(1,000) lines of code and thus small. Another aspect that is interesting in this context is the support of accelerators through an OpenCL version of the code [Lavallee2012].

The HydroC mini-application is the HydroC99_2DMpi benchmark which is a fine grain OpenMP + MPI version of the CFD that uses C99. It does not include the OpenACC, CUDA and the OpenCL methods of the CFD. This C99 version does a domain decomposition via MPI and then proceeds with a 2D sweep of the domain. A k-D tree was used for the domain decomposion as the code can then use the power of 2 processors while the 2D sweep algorithm makes use of the alterate directions scheme.

OpenMP was used to parallelize the Godunov and the Riemann routines. The Riemann routine is called from within the Godunov routine. Therefore, to parallelize the kernel, a parallel region at the main level in the temporal loop was constructed and the work was shared among threads at the Godunov level.

Repository: https://gitlab.version.fz-juelich.de/exanode/miniapp-hydroc.git

3.3 KKRnano

KKRnano is based on the Density Functional Theory (DFT) method, which is a very popular method in condensed matter physics and material science [Thiess2012]. It is written in Fortran 90 and uses MPI and OpenMP for parallelisation. The overall performance is dominated by dense matrix and other linear algebra tasks. The application is optimized for scaling to a very large number of atoms and thus for execution on massively-parallel HPC systems. It is, e.g., member of Jülich's High-Q Club[2], which is a list of applications that could demonstrate scalability using 28 racks of Blue Gene/Q, i.e. 458,752 cores. Material science is an area that will in future be in need of exascale computing resources.

The KKRnano mini-application represents the core operation of the Density Functional Theory application KKRnano. The core problem of the Green function based DFT calculation is the iterative matrix inversion of a block sparse and short ranged matrix instead of direct inversion solutions. This implies the main calculation in the mini-application is a block-sparse times block-sparse matrix-matrix multiplication. The blocks have double precision complex entries. Block sizes are chosen to be a size of 16x16 when the angular momentum expansion LMAX takes the value of 3 (where s, p, d and f electrons are considered). The block size of 32x32 is chosen with a non-collinear spin treatment. Each block-times-block multiplication is therefore, a task of 32 or 256 kiFlop respectively. The tight-binding KKR leads to a low scattering interaction between nearest neighbour cells.

Repository: https://gitlab.version.fz-juelich.de/exanode/miniapp-minikkr.git

3.4 miniFE

MiniFE[3] is already a mini-application, which is widely used for co-design projects in the USA, e.g. in the context of the Trinity system at NERSC or the CORAL systems at ORNL, LLNL and ANL. MiniFE mimics the finite element generation, assembly and solution for an

[1] https://github.com/HydroBench/Hydro/tree/master/HydroC
[2] http://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/_node.html
[3] https://mantevo.org/packages/

unstructured grid problem. The calculations are performed using a 3-dimensional box of configurable size. While the discretisation is structure, MiniFE treats it as an unstructured grid. The numerical problem is linear and the resulting matrix is symmetric. Therefore conjugate gradient can be applied, which is a popular algorithm for solving sparse linear systems.

The MiniFE mini-application contains several kernels for computing the diffusion matrix and the source vector element-operators, constructing the scattering element-operators into a sparse matrix and vector form, performing a sparse matrix-vector product during the conjugate gradient solve and the axpy, dot, and norm blas vector operations.

MiniFE provides support for computation on multiple cores, including pthreads and Intel Threading Building Blocks, and also a CUDA version for GPU. The OpenMP version was chosen for the mini-application.

Repository: https://gitlab.version.fz-juelich.de/exanode/miniapp-minife.git

4 Performance Results

In this section, we present performance comparisons of each of the mini-applications between experiments done on the Trenz-based prototype and the JURECA supercomputer at the Juelich Forschungszentrum.

The Trenz-based prototype maintained by FORTH consists of 4 nodes, each of which is a TEBFO808 Trenz board which is equipped with a Trenz TE0808 UltraSOM+ module. The module is equipped with a Xilinx ZYNQ UltraScale+ ZU9EG FPGA. This FPGA integrates 4 ARM Cortex-A53 cores plus 2 ARM Cortex-R5 cores. The latter have not been used here.

For the experiments, we have restricted ourselves to a single Trenz node.

In addition, we have made use of UnimemMPI, a library that provides an implementation of the MPI standard that was developed within the scope of the ExaNeST project, which was made available earlier than the optimized MPICH implementation under development in the ExaNoDe project.

The JURECA supercomputer consists of the following:

- 1872 cluster nodes, each with 2× Xeon E5-2680v3 CPUs, out of which 75 nodes are additionally equipped with 2× NVIDIA K80 GPUs
- 1640 booster nodes, each with 1× Xeon Phi 7250-F

For the experiments, we restrict ourselves to a single cluster node.

Table 1: Comparison of the CPUs used for the JURECA supercomputer and the Trenz-based prototype

	JURECA cluster node	**Trenz TE0808 UltraSOM+**
Number of sockets	2	1
Number of cores per socket	12	4
Core clock speed (base)	2.5 GHz	1.5 GHz
L1 instruction cache	32 kiByte	32 kiByte
L1 data cache	32 kiByte	32 kiByte

L2 cache	256 kiByte (private)	1 MiByte (shared)
L3 cache	30 MiByte (shared)	-

In the following sub-sections we perform a comparison between the JURECA processor and the processor used in the Trenz board. For used PAPI to measure the following performance counters:

- PAPI_TOT_INS: total number of instructions
- PAPI_TOT_CYC: total number of clock cycles
- PAPI_L1_DCM: number of L1 data cache misses
- PAPI_L2_DCM: number of L2 data cache misses

The data is collected for runs using 1, 2, 3 and 4 OpenMP threads distributed over different cores. Additionally a profiling overview using callgrind was produced on the Trenz boards using a single thread.

4.1 BQCD

The input parameters for the experiments include a lattice of size 8x8x8x8, with periodic boundary conditions for the fermionic fields in the *x*, *y*, *z* direction, Wilson gauge action and a Clover fermi action. The lattice size was chosen to be small as in real life larger lattices would be distributed over a larger number of nodes. The full input deck is specified in the "specs-bqcd.xml" file in the JUBE folder of the mini-application.

The non-SIMD version of the mini-application was used for these experiments.

The results are shown in Figure 1 to Figure 5 where we observe the following:

- The number of instructions on both platforms is similar.
- Significant differences are observed for the total number of clock cycles. This is expected as BQCD is a memory-bandwidth limited application and the JURECA node provides a significantly larger memory bandwidth (see also Section 4.6).
- The number of cache misses on both platforms differs by up to 10%. As the problem exceeds the size of the L1 and L2 cache on both architectures, this observation is consistent with expectations.

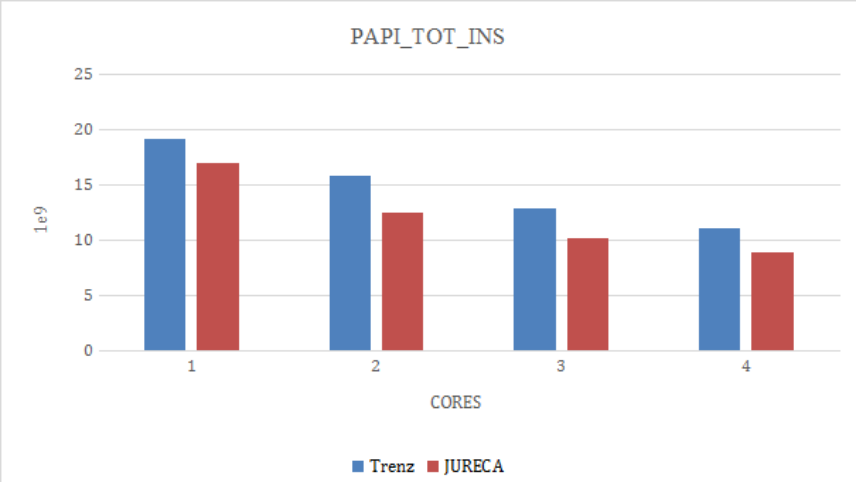


Figure 1: Total number of instruction for BQCD

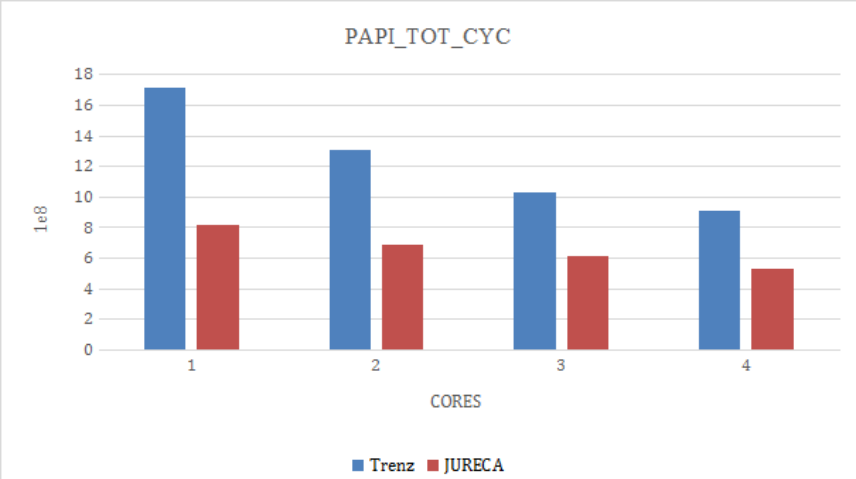


Figure 2: Total number of cycles for BQCD

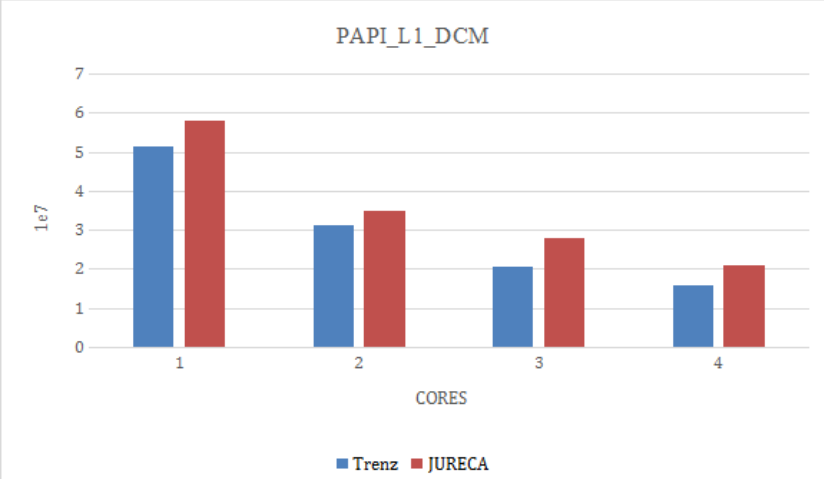


Figure 3: Number of L1 data cache misses for BQCD

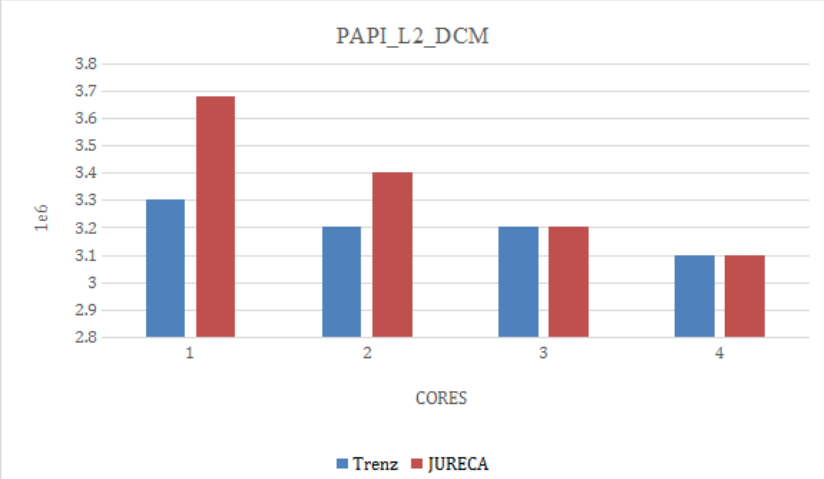


Figure 4: Number of L2 data cache misses for BQCD

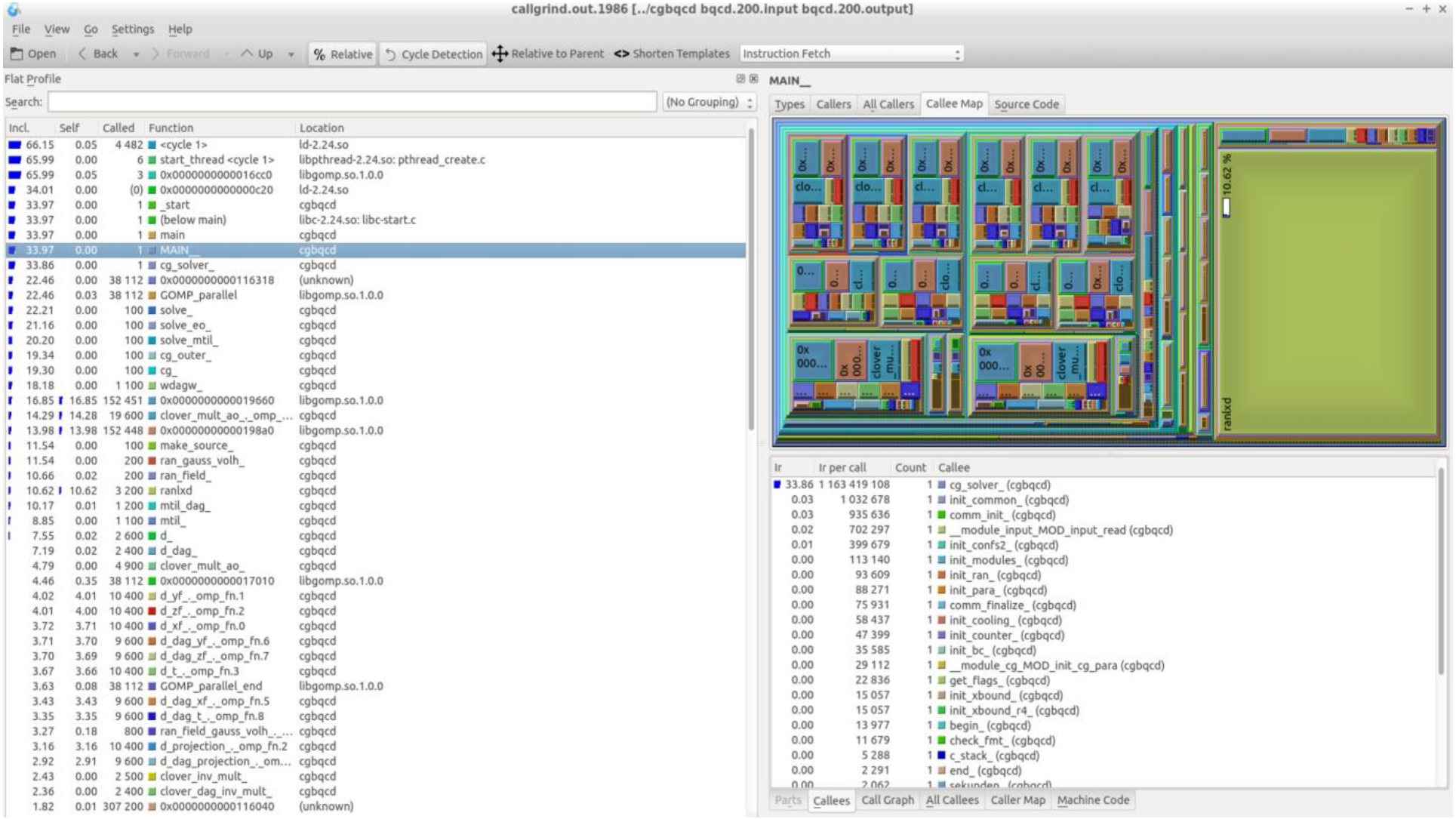


Figure 5: BQCD profile

4.2 HydroC

The input parameters for the experiments include a domain size of 500x500 which had a reflective boundary in the right, left, up and down direction. The courant factor used was 0.8 and the nxy_step parameter was chosen to be 500. Moreover, a single run included 100 iterations. The complete input deck is specified in the "specs-hydro.xml" file in the JUBE folder of the mini-application.

The results are shown in Figure 6 to Figure 9. We make the following observations:

- The number of instructions executed per core on both platforms is similar. However, on Trenz only a relatively mild dependence on the number of cores is observed, which is unexpected and requires further investigation.
- Similar as in case of BQCD (see Figure 2), using the same number of cores, the total number of cycles is about twice larger on the Trenz board compared to the JURECA node.
- While the number of L1 cache misses is similar on both platforms, a significantly smaller number of L2 cache misses is observed for the JURECA processing cores.

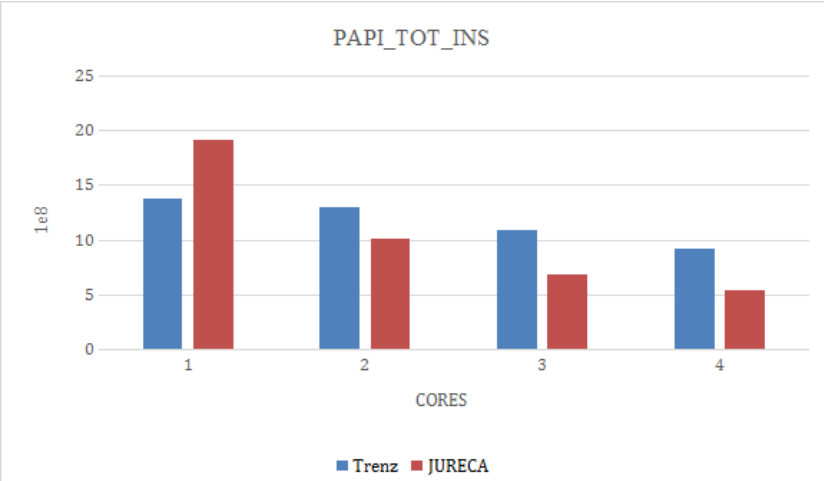


Figure 6: Total number of instructions for HydroC

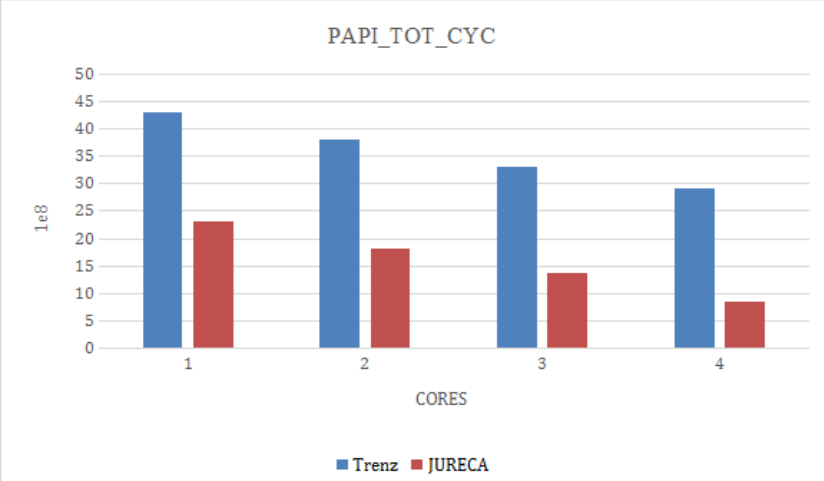
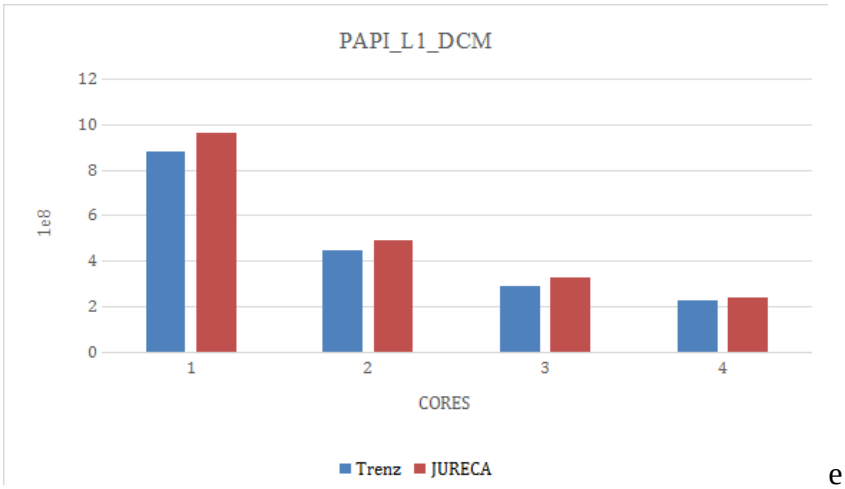


Figure 7: Total number of cycles for HydroC



e

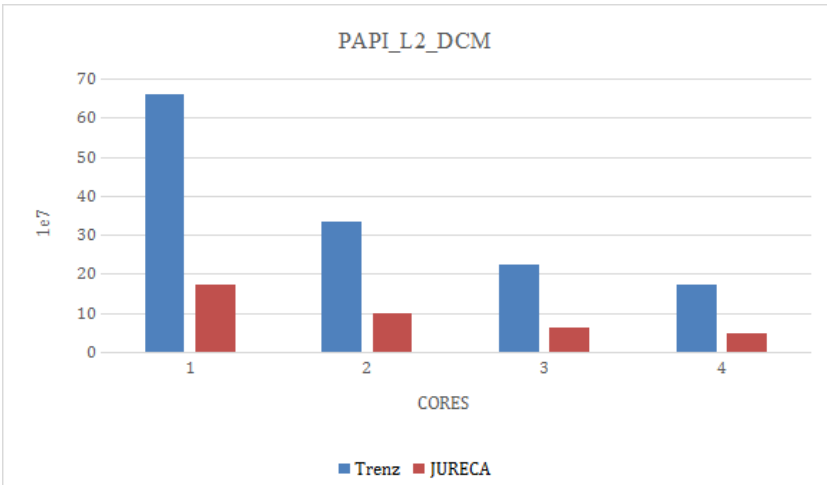


Figure 8: Number of L2 data cache misses for HydroC

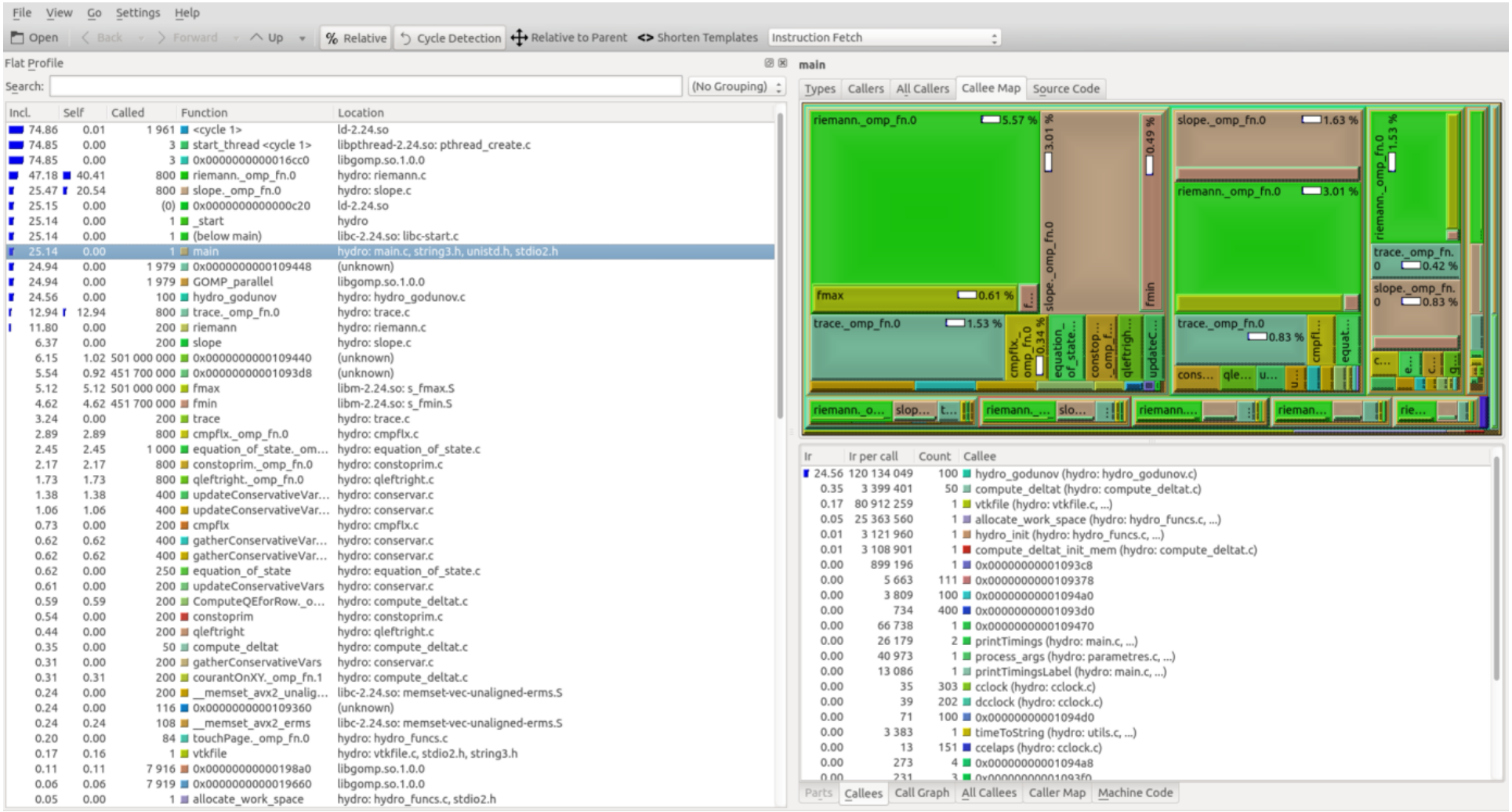


Figure 9: HydroC profile

4.3 KKRnano

The KKRnano input deck was chosen to be the Zinc Oxide problem.

The results are shown in Figure 10 to Figure 14. We highlight the following observations:

- Like for the previously presented applications, BQCD and HydroC, the number of instructions executed on both platforms is similar, although a slightly larger difference is observed.
- Unlike for the other applications the number of cycles is very similar. We expect this to be due to the fact that this application tends to be limited by throughput of floating-point operations, i.e. the very large difference in terms of available memory bandwidth has a much smaller effect.
- This is consistent with the observation that the number of cache misses per instruction is significantly lower for this application. While PAPI_L1_DCM/PAPI_TOT_INS is about 0.03 and 0.5 for BQCD and HydroC, respectively, it is about 0.0005 for KKRnano.

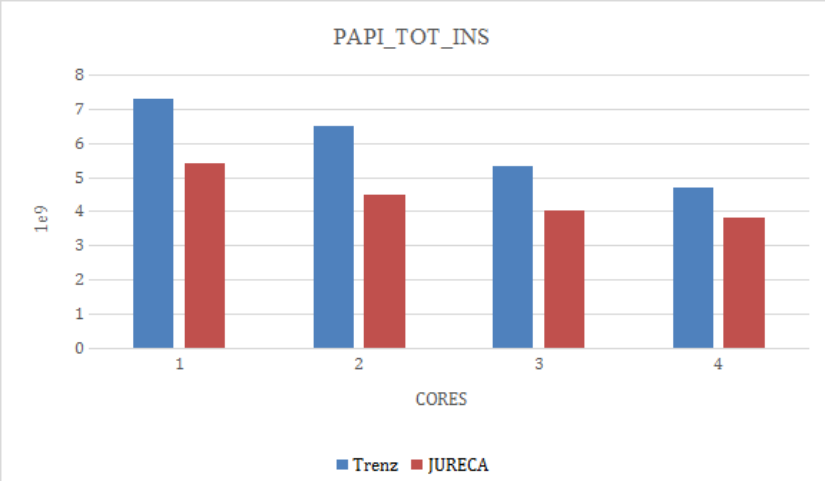


Figure 10: Total number of instructions for KKRnano

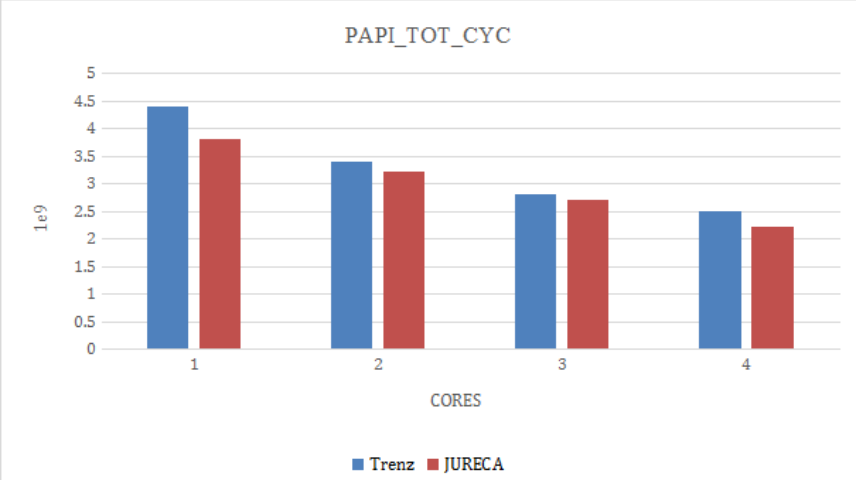


Figure 11: Total number of clock cycles for KKRnano

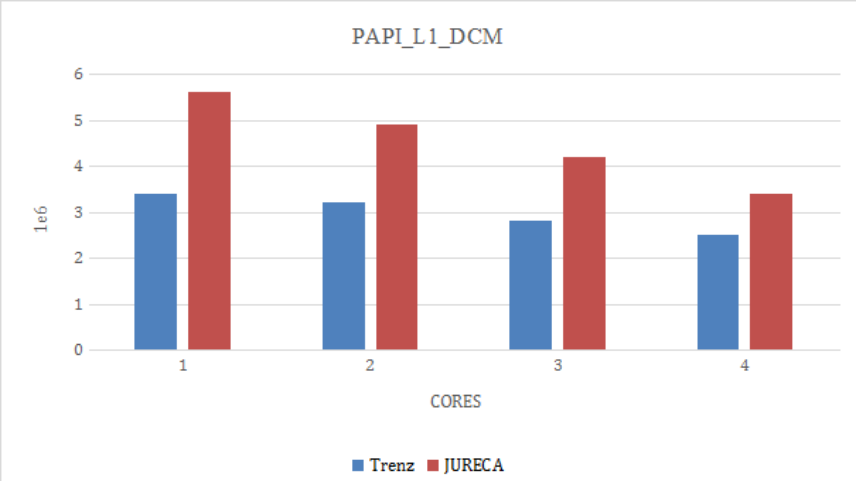


Figure 12: Number of L1 data cache misses for KKRnano

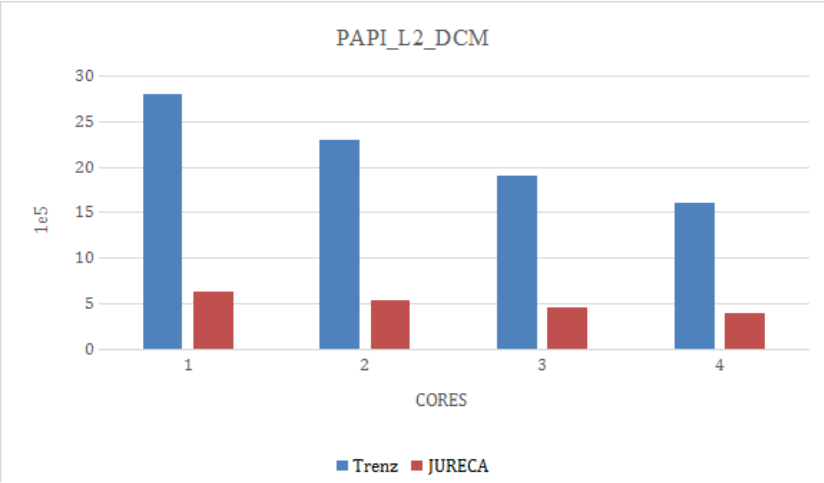


Figure 13: Number of L2 data cache misses for KKRnano

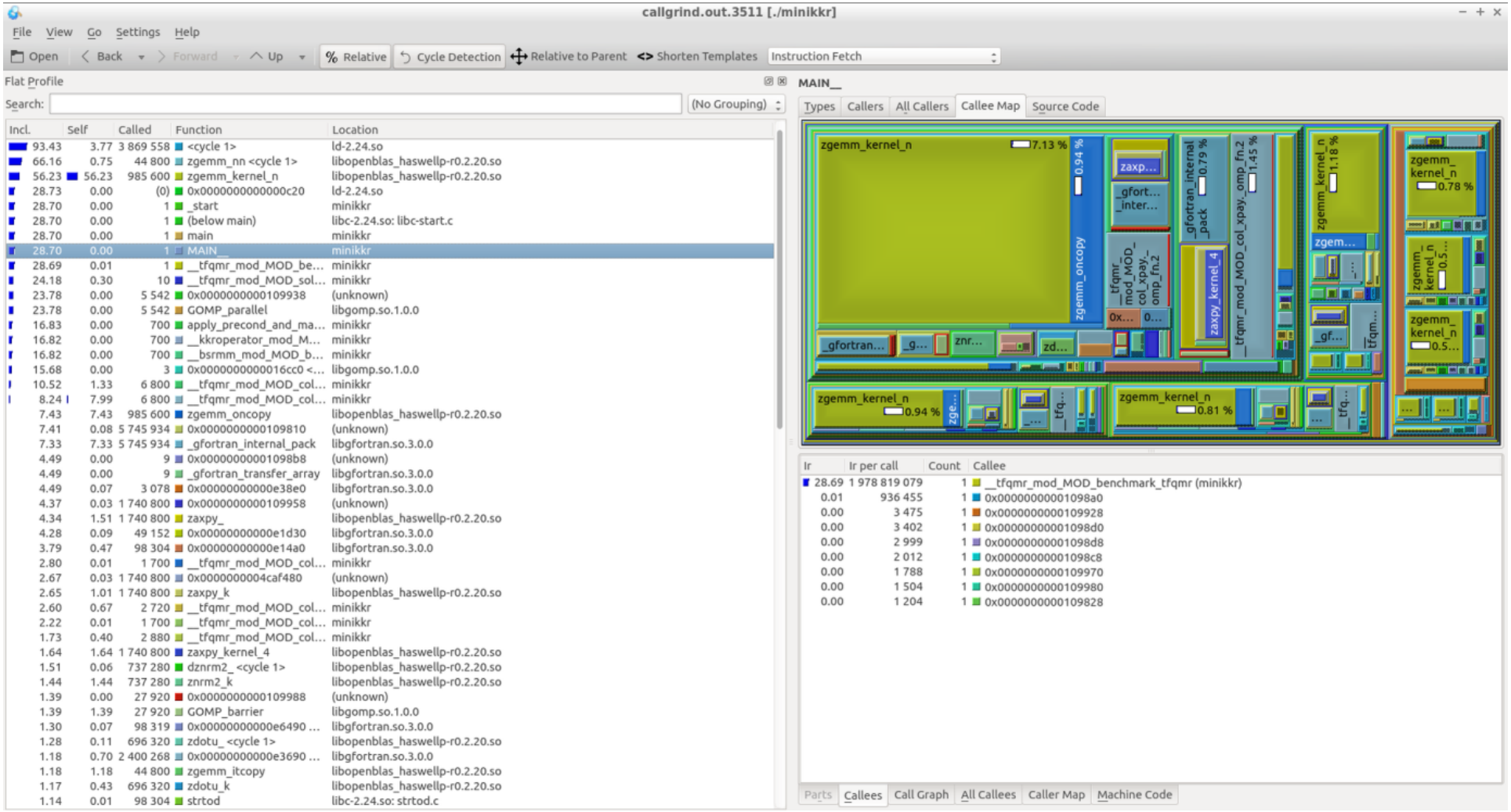


Figure 14: KKRnano profile

4.4 miniFE

The input parameters for the experiment includes nx=10, ny=10, nz=10 without locking and with calculations optimized for NUMA. The complete input deck is specified in the "specs-minife.xml" file in the JUBE folder.

The results are shown in Figure 15 to Figure 19. We make the following observation(s):

- The number of instructions executed on the Trenz board cores exceeds those executed by the JURECA cores by a factor 4-5. Whether this difference is due to code generation inefficiencies or due to inefficiencies of OpenMP on the Trenz nodes requires further investigation.
- Despite the L1 cache size and the aggregate L2 cache size available to 4 cores is the same on both platforms, the number of L1 and L2 cache misses is significantly larger on Trenz. However, the number of L1 and L2 cache misses per instruction is 0.007 and 0.003, respectively, and thus relatively small.
- The number of cycles required to execute miniFE on Trenz is about 8 times larger. Given the relatively small number of L1 and L2 cache misses, this performance difference seems to be mainly due to the difference in number of instructions in combination with the larger throughput of instructions per cycle on the JURECA cores (see Figure 23).

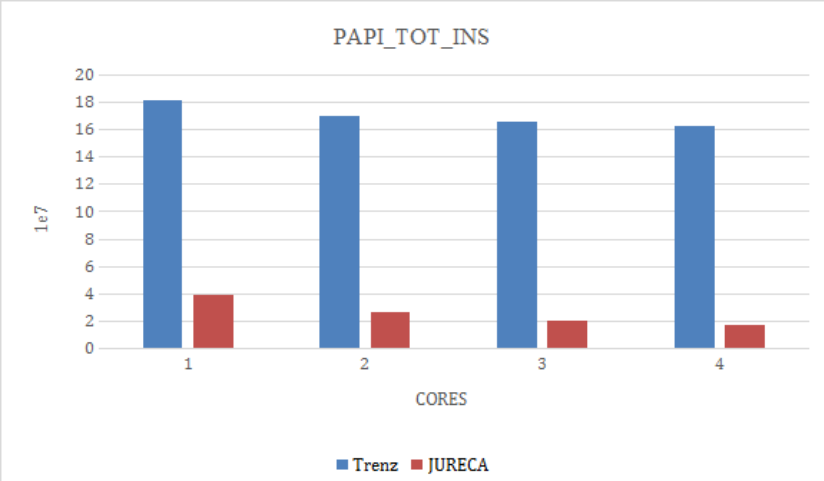


Figure 15: Total number of instructions for miniFE

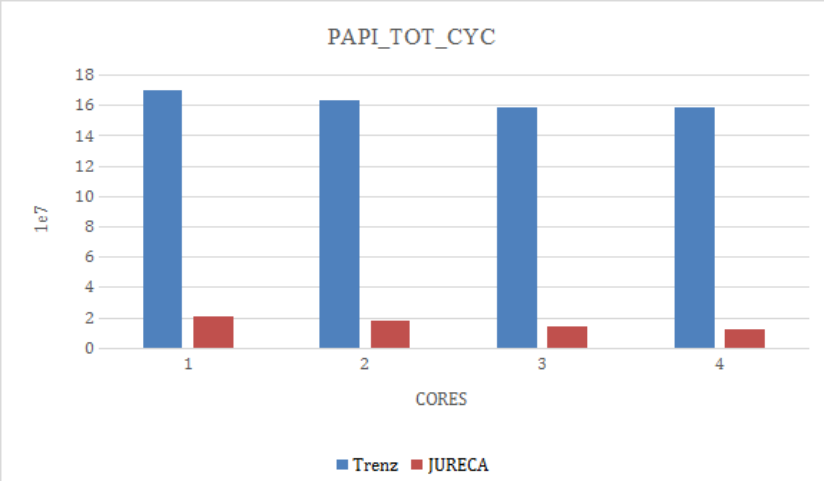


Figure 16: Total number of cycles for miniFE

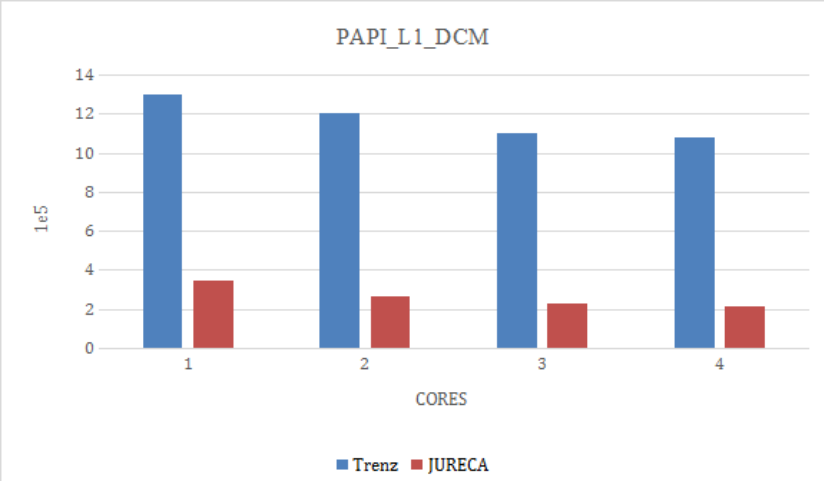


Figure 17: Number of L1 data cache misses for miniFE

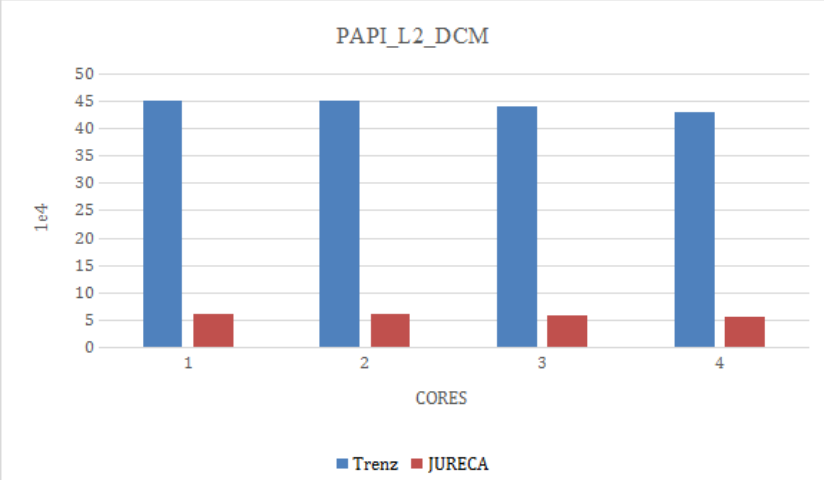


Figure 18: Number of L2 data cache misses for miniFE

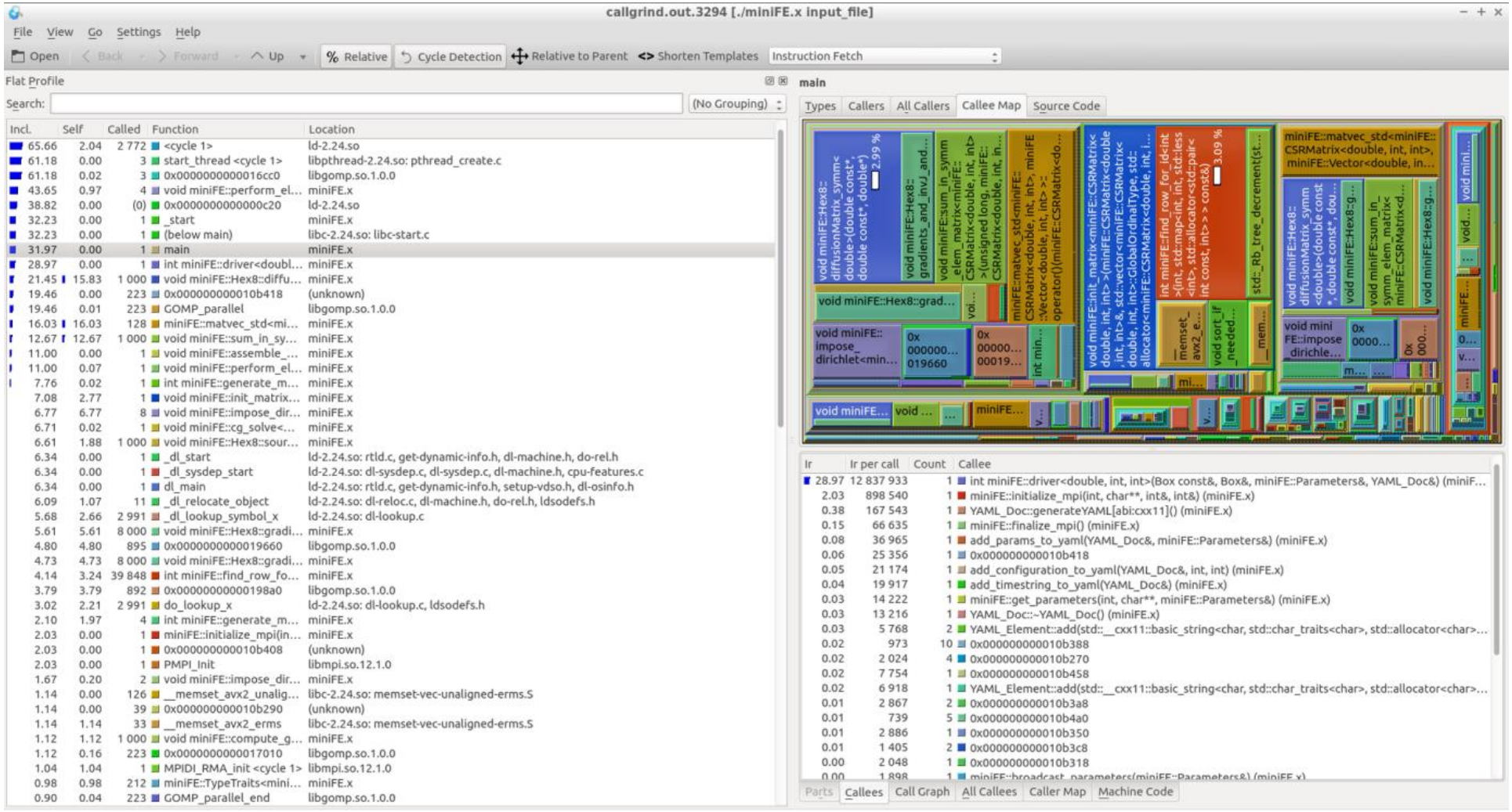


Figure 19: miniFE profile

4.5 Throughput of instruction analysis

We calculate the average Instructions per Cycle (IPC) for the mini-applications as follows:

IPC = PAPI_TOT_INS / PAPI_TOT_CYC

The results are shown in Figure 20 to Figure 23.

Figure 20: IPC for BQCD

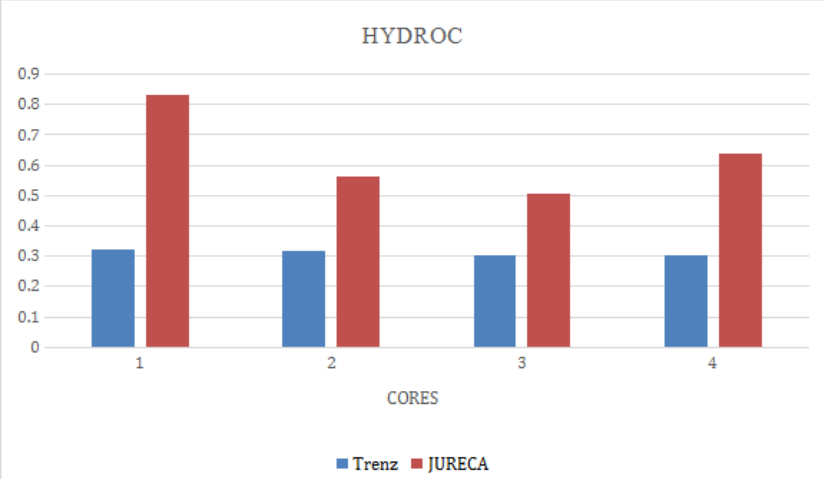


Figure 21: IPC for HydroC

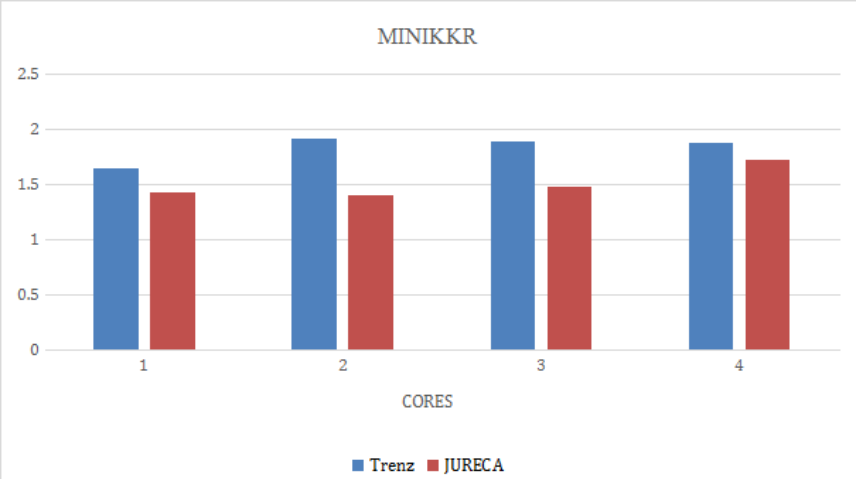


Figure 22: IPC for KKRnano

Figure 23: IPC for miniFE

4.6 Memory bandwidth analysis

We ran experiments using the SAXPY benchmark (single-precision scalar *A* times vector *X* plus vector *Y*), where on output the input vector Y is replaced. SAXPY was executed for vector lengths *L* = (1024, 2048, 4096, …, 2097152) over a single thread. The performance counters PAPI_TOT_INS (total instructions completed), PAPI_TOT_CYC (total cycles), PAPI_L1_DCM (L1 data cache misses) and PAPI_L2_DCM (L2 data cache misses) are compared over varying vector lengths.

The results are shown in Figure 24 to Figure 27. We make the following observations:

- The number of instructions is similar on both platforms.
- The number of cycles on the Trenz prototype is almost an order of magnitude larger compared to the JURECA node, which translates to an even larger difference in wall-clock time taken the difference in clock frequency into account.

- A significant difference in number of cache misses is observed, of half an order of magnitude, which is unexpected as the cache line size is the same on both architectures.[4]

The memory bandwidth on the Trenz board was measured using the SAXPY benchmark to be B_{mem} = 1.6 GByte/s. The kernel loop executes two flops per iteration, while having 2 loads and one store operation (ignoring the load of the scalar variable), i.e. the arithmetic intensity AI = (2/12) Flop/Byte = 0.17 Flop/Byte. Therefore, the maximum attainable performance for scalar-vector multiplication and addition is B_{fp} = 0.27 GFlop/s. The peak performance for the chip is $B_{fp,peak}$ = 1.5 GHz * 8 SP Flop/cycle = 12 GFlop/s. The maximum attainable performance is therefore in this case about 2% of the peak performance.

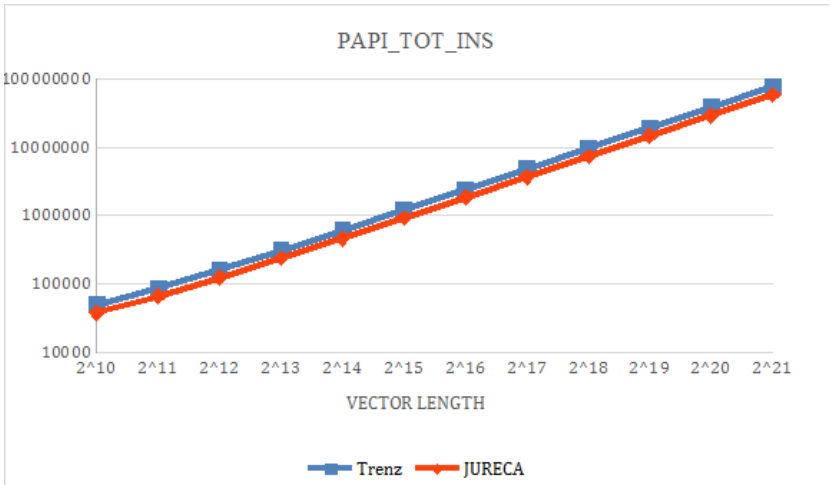


Figure 24: Total number of instructions for SAXPY

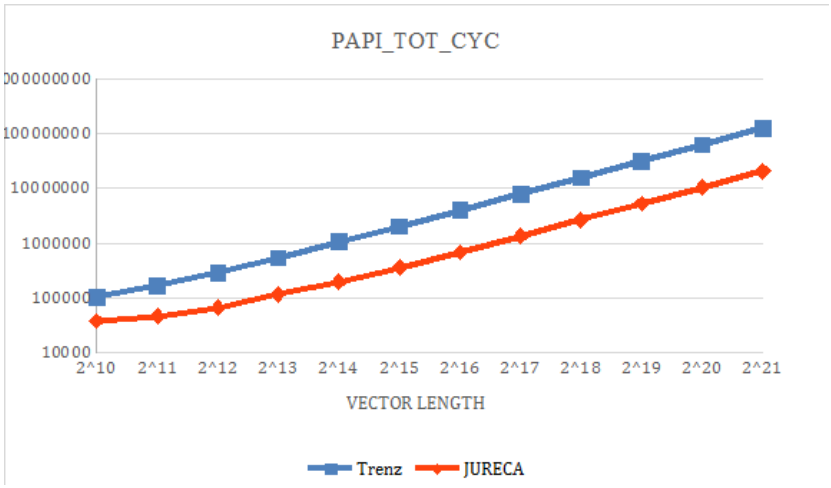


Figure 25: Total number of cycles for SAXPY

[4] For large L we expect the number of cache misses to be about $L/4$. The number of L1 cache misses observed for JURECA matches this expectation.

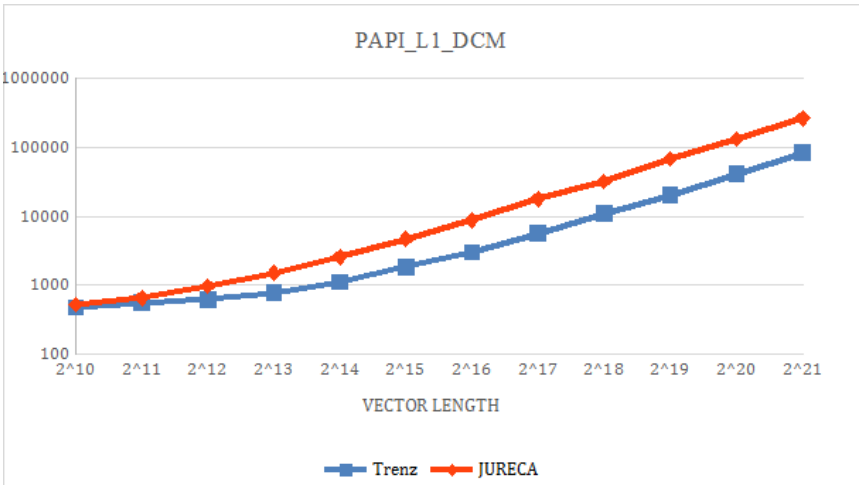


Figure 26: Number of L1 data cache misses for SAXPY

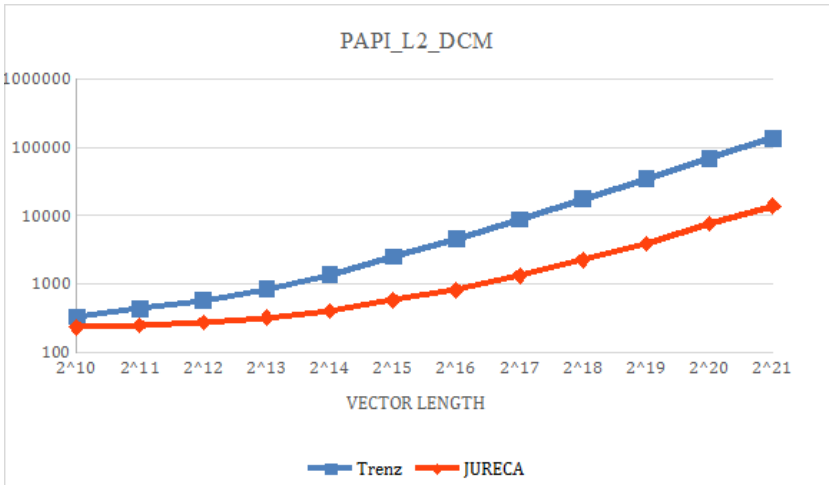


Figure 27: Number of L2 data cache misses for SAXPY

5 Summary and Concluding Remarks

The availability of several mini-applications, all with different computational characteristics, facilitates performance evaluation of current and future ExaNoDe hardware components for real-life applications. While in this report the focus was on results obtained on single nodes, all mini-apps do support parallelisation over multiple nodes using MPI. While a performance analysis for scaling to multiple nodes is still pending, within a single node the UNIMEM-based version of MPI has been successfully tested on the prototype based on Trenz boards.

In units of number of cycles needed to execute the mini-applications on a single core, the Trenz node is 2-8 times slower for all applications except for KKRnano. The difference in performance is expected to be largely due to the memory bandwidth, which is significantly smaller for the Trenz node compared to the JURECA node. However, also significant differences in number of cache misses have been observed despite the similar aggregate size of the caches.

6 References

[Brower2017] R. Brower et al., “Lattice QCD Application Development within the US DOE Exascale Computing Project,” arXiv:1710.11094.

[Dongarra1999] J. Dongarra et al., “Top 500 Supercomputing Sites,” Technical Report, University of Tennessee, Knoxville, USA, 1999

[Gruenwald2012] D. Grünwald, “BQCD With GPI. A Case Study,” HPCS’12, 2012 (doi:10.1109/HPCSim.2012.6266942).

[Heroux2009] M.A. Heroux et al., “Improving Performance via Mini-applications,” Sandia Report SAND2009-5574, 2009 (doi: 10.2172/993908).

[JUBE] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/JUBE/_node.html

[Lavallee2012] Pierre-François Lavallée et al., “Porting and optimizing HYDRO to new platforms and programming paradigms – lessons learnt,” Technical report, PRACE, December 2012.

[Nakamura2010] Y. Nakamura, H. Stüben, „BQCD – Berlin Quantum Chromodynamics program,” PoS LAT2010, 040, 2010 (arXiv:1011.0199).

[Thiess2012] A. Thiess et al., "Massively parallel density functional calculations for thousands of atoms: KKRnano," Physical Review B, Vol. 85, 235103, 2012 (doi:10.1103/PhysRevB.85.235103).

[UEABS] http://www.prace-ri.eu/ueabs/