D5.2 – First-year Report on Proof-of-Concept Activities
Version 1.0

Document Information

Contract Number	676553
Project Website	www.pop-coe.eu
Contractual Deadline	Month 12, September 2016
Dissemination Level	Public
Nature	Report
Authors	José Gracia (HLRS)
Contributors	Stephan Walter (HLRS), Dirk Schmidl (RWTH)
Reviewers	Claudia Rosas (BSC)
Keywords	parallel efficiency, single-core performance, data-locality, collective MPI operations, data-dependencies

Notices: *The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No n°676553.*

Change Log

Version	Author	Description of Change
v0.1	José Gracia	Initial structure of document
v0.2	Contributors	Added individual PoC activity reports
v0.3	José Gracia	High-level overview
v0.4	José Gracia	Ongoing activities
v0.5	José Gracia	Executive summary and conclusions; submitted to internal review
v0.6	José Gracia	Revision due to review; resubmitted for approval
v1.0	José Gracia	Released to the EC

Contents

Executive Summary

In this document we report the current state of Proof-of-Concept activities. This type of activity is the central service of work package 5. The aim of the Proof-of-Concept activity is to assist users in implementing complex code refactoring or applying advanced parallelisation techniques.

Proof-of-Concept activities may be initiated as a result of performance assessments in work package 4, but unlike them, are expected to take significantly longer to conclude.

So far, three Proof-of-Concept activities have been concluded and are thus described in some detail in separate chapters of this document. These three activities have resulted in significant performance improvements. A further two activities are currently under way and their state is only summarised briefly. Both activities show potential for substantial performance increase.

1 High-level Overview of Proof-of-Concept Activities

The project POP offers three distinct service activities. Two of them, Audit and Performance Plan, are performance analysis services hosted in work package 4. The Audit is an initial assessment of an applications performance characteristics and identification of possible bottlenecks, while the Performance Plan does more detailed root cause analysis of bottlenecks. One of the outputs of theses assessments, in particular from the Performance Plan, is concrete recommendations for changes of the applications code, algorithm or data structures.

In some cases, the recommendations can be implemented with simple code refactoring by a person with average parallel programming skills. In general, however, implementing the recommendation will require complex code refactoring and expert knowledge of parallel programming. Such specialised skills are not available in most groups that do parallel programming. Therefore, POP offers a third level of service called Proof-of-Concept (PoC). During this activity, POP staff will exemplify some of the recommended code refactoring, either directly on the customers application or on a simplified skeleton thereof. The aim is to illustrate the necessary changes and to train the customer on their own code.

The procedure of the PoC activities has been defined in D5.1. In short, customer and POP staff prepare a PoC Plan. This document defines the specific use-case and lists the particular recommendations that will be addressed during the activity. It also defines a metric to track progress and optionally sets a target to be reached. At the end of the activity, POP staff prepares a PoC Report which is handed over to the customer for approval, thus concluding the activity. Due to their nature, PoC activities take much more time than analysis activities. We expect PoC activities to last between 3 and 6 months.

PoC activities are expected to yield best-practise guidelines and input for training material which will be taken up in work packages 3 *Community Development*, and 6 *Training and Documentation*, respectively. We also expect some feedback on and suggestions for improvement of practise of performance assessment in work package 4 *Analysis*.

PoC activities contribute to milestones MS3 (expected in project month 24) and MS4 (expected in project month 30) with, respectively, 33 and 42 completed PoC activities. PoC activities, however, depend on the completion of assessments (and on their recommendation of a PoC) and therefore the expected rate of PoC activities cannot be extrapolated linearly from the project start. In addition, users wish to prepare the code before handing it to POP staff, which causes further delays between the recommendation of a PoC and the request by the user. As of September 1st, i.e. end of project month 11, the POP Project has concluded three PoC activities and is working on a further two.

The remainder of this document reports on the PoC activities conducted so far. The already concluded activities on the codes *Ateles* (section 3), *FIRST* (section 4) and *GraGLe2d* (section 5), are reported in different sections, respectively. Section 2 summarises ongoing PoC activities. We make some final observations in section 6.

2 Summary of Ongoing PoC Activities

To date, the POP Project has concluded three PoC activities, which are describe in subsequent sections of this document. In addition there are two further PoC activities under way, specifically on the codes *Quantum Espresso* and *DFTB*. Their current state is briefly reported in the following.

Quantum Espresso is a parallel computational suite to calculate electronic-structures and materials modeling. It is developed as an open initiative of the Quantum Espresso Foundation[1]. It is mainly implemented in Fortran using a combination of the parallel programming models MPI and OpenMP. The code has been analysed by POP staff (see Audit POP_AR_2).

The performance Audit registered that the current hybrid MPI/OpenMP version has low parallel efficiency for 5 threads and beyond. Significant parts of the code are not yet properly parallelised with OpenMP. As a consequence, the master threads ends up doing a large part of the computational work and communication. The Audit recommends to fully parallelise the code using a task-based approach. This approach has two advantages in this particular case: 1) simplicity of code refactoring (computations and communications are already easily encapsulated in tasks in Quantum Espresso), 2) potential to overlap computation and communication through proper data-dependencies between tasks.

The core of the code is independent FFT operations followed by all-to-all MPI communication. For simplicity the PoC is done on this core component (*FFTlib*, extracted and provided by the customer), rather than on the full application. Experiments and evaluation of the PoC are done on MareNostrum III at BSC targeting high parallel efficiency on up to 48'896 cores.

This activity is already well progressed with implementation under way.

DFTB is a code used in Chemistry and Materials research. The code is written in Fortran and parallelised with MPI. It has been analysed by POP staff (see Audit POP_AR_25) and found to exhibit low parallel efficiency which seems to stem from a large number of MPI collective operations. These collective operations are part of an algorithm to calculate sparse-matrix multiplications. The PoC activity aims to increase parallel efficiency by optimising the use of collective operations.

This activity has just started recently.

[1] http://foundation.quantum-espresso.org

3 Report on Activity Ateles PoC

Description of the Application

Ateles is a Finite-Element Computational Fluid Dynamics code, which uses the Discontinous-Galerkin scheme on top of a distributed parallel octree mesh data structure. The code uses an explicit time integration scheme and a static mesh refinement to increase the numerical accuracy in areas of interest.

Previous Assessment and Recommendation

The application has already been analysed during the performance Audit POP_AR_4, with respect to the overall performance of the application, and the Performance Plan POP_PP_01, where the load balancing problem observed during the Audit was addressed.

The PoC focused on the recommendations regarding computational performance on the Cray XC40, which is an x86 system, stemming from the initial Audit. In detail this was a code refactoring through the inlining of excessively called functions, the replacement of divisions with multiplications inside loops, and the improvement of the code vectorization. In addition to this, the applicant also requested a brief evaluation of the code performance on the NEC SX-ACE vector machine, at HLRS, which has a different computer architecture compared to common x86 systems.

We agreed with the user to use two input sets, which solve the same problem but with values 16 and 32 for the `MaxPolyDegree` input parameter, that effects the accuracy of the solution. Since the number of executed instructions inside the analysed iterations correspond to the power of four of `MaxPolyDegree`, a vectorized binary should scale better with higher values of `MaxPolyDegree` than a nonvectorized binary.

Implementation

Inlining Our first task was the generation of a full profile of the executed function calls. Followed by an analysis with respect to the potential speedup through inlining. After the identification of the appropriate function calls inside the source code, the inlining task was performed through the usage of preprocessor scripts. We were able to reduce the number of function calls by nearly 97% leading to an iteration runtime reduction of roughly 6%. As a positive side effect this also improved significantly the analysability of the application through the reduction of the instrumentation overhead.

Division replacement Our second task was the identification of candidates for the replacement of divisions through multiplications. We achieved this through a review of the source code files with the most important functions. The corresponding code modifications lead to an additional execution time reduction of roughly 18%. It is worth noting, that replacement of division with multiplications have an impact on the numerical results of calculations, which is the reason that compiler often shy away from doing such transformation automatically.

We suppose, that we were able to generate an awareness of the problem with divisions inside loops.

Vectorization From the previous Performance Audit it was already known, that the GNU compiler was not able to vectorize the code of important functions because of conditions inside the computation loops. We were able to eliminate this obstructions, but new problems arise that

MaxPolyDegree	CRAY (GNU)	CRAY (Intel)	SX-ACE (sxmpif03)
16	8.26 s	8.64 s	66.78 s
31	94.69 s	95.32 s	771.48 s
32	188.07 s	155.58 s	5985.13 s
33	121.22 s	124.47 s	984.29 s

Table 1: Runtime for the fully optimised source code with different compilers and maxPolyDegree on the x86 System HazelHen and the NEC SX-ACE

prevent that the code vectorize with the preferred GNU compiler. Corresponding to our goals, we tested with the Intel compiler as alternative compiler and were able to confirm from the compiler logs, that the Intel compiler was able to vectorize the critical code parts independent of our code refactoring. Nevertheless, Figure 1 shows, that although the application is heavily compute bound, the vectorized Intel binary is for the lower `MaxPolyDegree` not faster than the optimised GNU binary without vectorization.

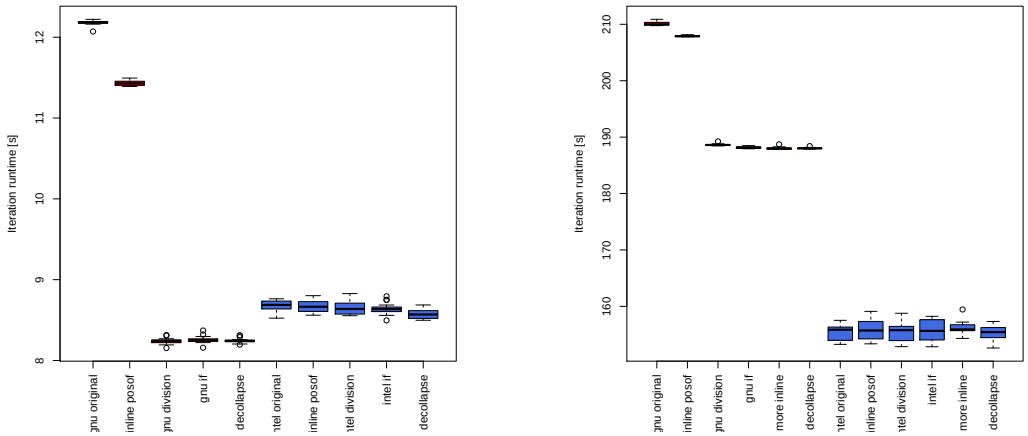


Figure 1: Runtime of the analysed itertions for the different code versions compiled with the GNU and Intel compiler on HazelHen.

NEC SX-ACE After some code modifications we were able to analyse the performance of the application on the NEC SX-ACE vector machine. We were able to detect through the NEC tools and special hardware counters a problem with the vectorization through the NEC Fortran compiler that lead to a bug report. We also discovered a problem with the main memory access pattern, that lead to a high number of main memory bank conflicts. A known source for such a behaviour is a memory access pattern with a large 2^n stride. We tested therefore also the `MaxPolyDegree` values 31 and 33, which should prohibit memory accesses with such a stride.

Since such access patterns can also be problematic on x86 systems, we decided to execute the additional `MaxPolyDegree` settings also on HazelHen. Table 1 shows that on both machines the `MaxPolyDegree` of 33 is significantly faster than the runs with a `MaxPolyDegree` of 32. And this although the higher value of `MaxPolyDegree` implies 13 % more work. So we were able to detect an unknown performance problem on both machines.

Conclusions

We were able to fulfil during the PoC of the Ateles code all goals and increased the application performance up to 50% for the preferred GNU compiler. Figure 1 shows the effect of the optimisation on the application runtime for the chosen input configurations.

We were also able to show the developers that there is a significant problem with the main memory access pattern on the NEC SX-ACE vector machine and very likely also on the x86 system HazelHen. The developers should now be in the position to write specific micro benchmarks to further analyse the vectorization behaviour of the application in combination with the memory access pattern, which is, together with load imbalance, the main performance issue of the application.

4 Report on Activity FIRST PoC

Description of the Application

FIRST is a simulation tool for elasto-hydrodynamic coupled multi-body systems. The application is developed and distributed by the company IST mbH in Aachen. The users of FIRST typically run the application on desktop systems. IST mbH realized that the runtime for large datasets is not satisfying for some users, so a performance analysis was done as part of POP (see Audit POP_AR_20).

Previous Assessment and Recommendation

The initial performance assessment figured out, that two hotspots exist in the application for the chosen test dataset. Solving a Jacobian matrix (fijaco) took 61% of the execution time and the integration of this matrix (simpr) took 36 %. Analyzing the instructions per cycle metric (table 2) of the application revealed that the core utilization was good.

Functions	IPC
total application	2.36
fijaco	2.34
simpr	2.40

Table 2: IPC values for functions in the application

As serial optimization seems to be no option, the recommendation of POP was to parallelize the code with OpenMP. As the code is written in Fortran77 and makes extensive use of common blocks, a first parallel version needs to be done with care, since data races can be introduced easily due to the use of global variables. As the developers are not experts in OpenMP parallel programming, POP was asked for assistance here. This PoC study aims at parallelizing the most time consuming loop to show how such a parallelization can be done for a Fortran77 code.

Implementation

The main loop in `fijaco` was parallelized with an OpenMP `parallel for` construct. Many of the common blocks needed to be privatized using the `threadprivate` clause to avoid data races. Other accesses to shared data needed to be protected by `critical` regions. In cases where sharing intermediate results was not easily possible, functions were executed redundantly. Furthermore, a dynamic loop scheduling was used to avoid load imbalance. All these steps were supported by performance and debugging tools.

Figure 2 shows the speedup on a quad-core desktop CPU as it is typically used be customers. As only a hotspot with 61% of the total time was parallelized, Ahmdah's law states that the speedup is limited by the green line. The measured speedup is close to this line, so the parallelization works efficiently.

Conclusion

As part of this PoC study the most time consuming loop of the FIRST application was parallelized with OpenMP. It was shown how the parallelization can be achieved although the data sharing through common blocks was challenging. The reached speedup was satisfying for IST

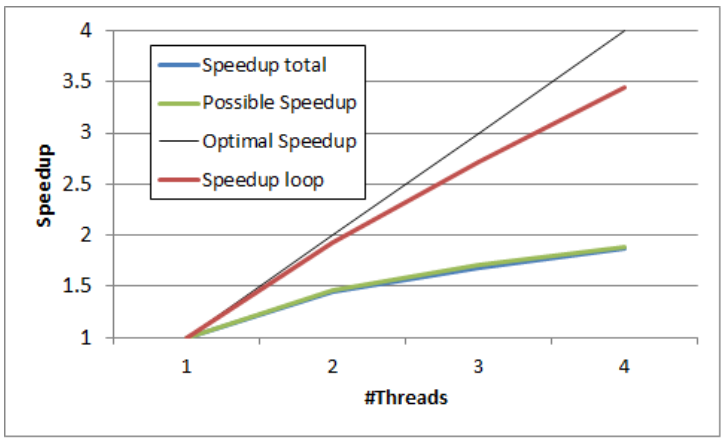


Figure 2: Scalability of the FIRST code

mbH for a first version. To improve the parallelization the next step is to parallelize `simpr`, which was the second large hotspot in the code.

5 Report on Activity GraGLeS2D PoC

Description of the Application

The application GraGLeS2D is a code for the simulation of microstructure evolution in polycrystalline materials. GraGLeS2D is under development at the institute of Physical Metallurgy and Metal Physics of RWTH Aachen University[2]. The code is written in C++ and parallelized with OpenMP. The target platform for the application is a 16-socket BCS machine operated at the IT Center of the University. These machines are hierarchical NUMA machines with 2 levels of non uniform memory accesses. Standard 4-socket Bullx s6010 server are combined with a proprietary cache coherent link, called Bull Coherence Switch (BCS). This leads to a setup where every socket can access local memory, remote memory on the same s6010 board over the Intel QPI link and memory over the BCS link on remote s6010 boards.

Previous assessment and recommendation

A performance audit was done for the GraGLeS2D application as part of POP (see Audit POP_AR_7). In one of the hotspots, the function *ExplicitGrainBoundary::projectToGrainBoundary*, a serial performance bottleneck was detected. A lot of calls to square root and divisions were detected which performed badly. Furthermore the scalability on the BCS machine was not satisfying. As shown in Figure 3 the speedup going from 8 to 128 threads (1 to 16 sockets) is below 6 for many functions.

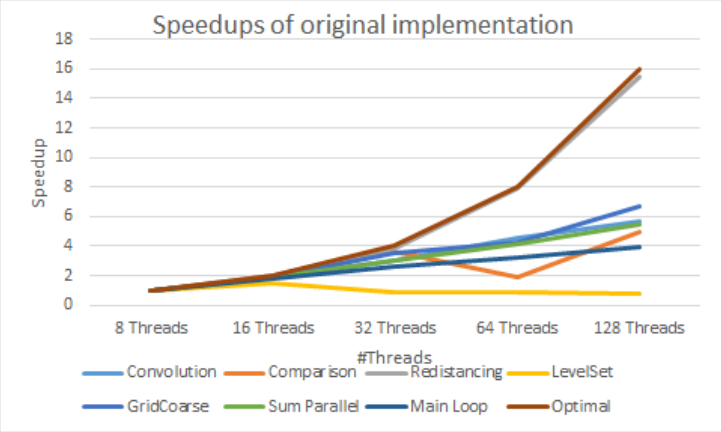


Figure 3: Scalability of the original version of GraGLeS2D.

In the audit two main reasons were identified for the limited scalability:

1. The memory allocation turned out to be a bottleneck at scale.

2. Hardware counter experiments showed a lot of remote memory accesses in the *comparison* region where the comparison of boundary grains is performed.

The recommendation after the audit was to reduce the serial overhead caused by the square root operations and divisions, to reduce the overhead of memory allocation calls by using versions of malloc which are optimized for multithreaded use and to further investigate and optimize

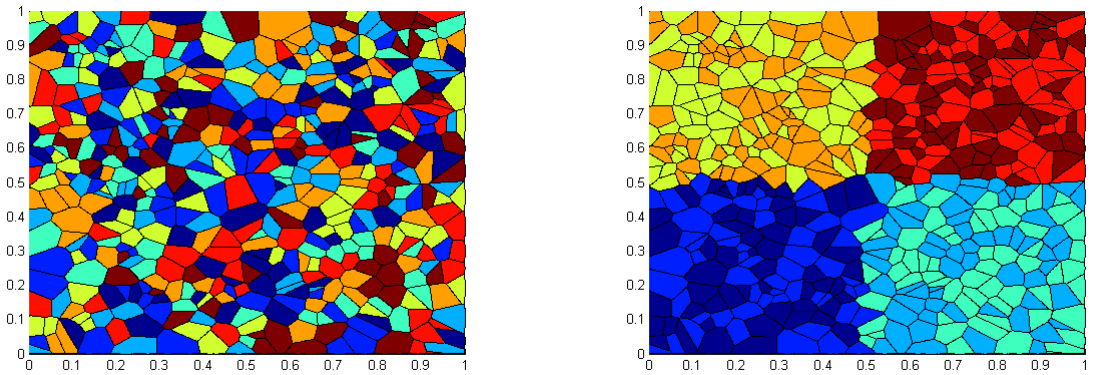
[2] http://www.imm.rwth-aachen.de

the data layout on the system to reduce the number of remote accesses. The goal of this proof-of-concept study is to implement the recommended changes in cooperation with the user to show that a better scalability and performance can be reached on the BCS target system.

Implementation

During the PoC activity the square root operations and division operations were investigated. It turned out, that the square root operations were used in a comparison step, where pairs of distances were compared. As square root is a linearly increasing function, comparing both values without applying the square root operations let to the same ranking and the overhead of square root calls could be avoided. The division operations were also not necessary in many cases. Very often, only the sign of the number was needed and not the exact number, so the division operation was moved into the branch where it was really needed. Overall, this let to a speedup of 3.17 in the convolution region.

As a second step was to test 2 different versions for memory allocation which were optimized for multithreading. The first one is provided with the Intel Compiler, `kmp_malloc`. This version requires to change all `malloc` calls in the program to `kmp_malloc` calls. The second option was to use the `jemalloc` library which replaces all `malloc` calls without the need to do any source code changes. Although `kmp_malloc` performed slightly better in synthetic tests, it would have been to much work to change all kind of memory allocation in the application. Therefore `jemalloc` was used and let to a speedup of nearly 3 in parts of the application.



(a) Initial load/data distribution

(b) Load/data distribution after optimization

Figure 4: Load/data distribution

The last optimization changed the load and data distribution across threads. During the comparison of grains, the data from neighboring grains is needed. The distribution of grains to threads was investigated and illustrated in Figure 4(a). The same color indicates that the data is located on the same socket and similar colors indicate that sockets are on the same board, connected over a QPI link. In this study the work distribution algorithm was changed to reduce the number of remote accesses. Since the QPI links deliver roughly 4 times the bandwidth than the BCS links, the focus was on reducing the traffic over the BCS links. Figure 4(b) shows the data distribution achieved with the new algorithm. Obviously more neighboring grains now have similar colors. These changes reduced the runtime of the `exeuteComparison` region roughly by a factor of 4.

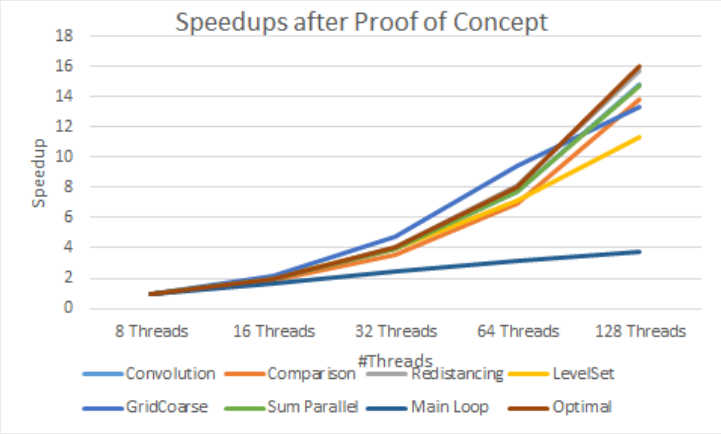


Figure 5: Scalability of the code GraGLeS2D after the PoC activity was performed

Conclusion

During this PoC study the GraGLeS2D code was optimized to better perform on a 16-socket BCS system. The main goal was to show that a good scalability using OpenMP can be achieved on a large hierarchical NUMA system for the application. Serial optimizations were applied leading to a 3.17x improvement in the convolution regions and the overhead of memory allocation calls was reduced by using the `jemalloc` library which leads to nearly 3x improvement in some parts of the application. Finally, the data layout was optimized to reduce the number of remote accesses during execution which improved the runtime of the executeComparison region by 4x. Figure 5 shows the speedup after the PoC activities. It shows, that the speedup was increased significantly for most functions and that now a good scaling can be reached. So, the PoC activity can be seen as successfully accomplished.

6 Conclusions

In this document we have reported the current state of Proof-of-Concept activities, which is the main service activity in work package 5. PoC is the highest level of service of the POP Project aiming at assisting customers to implement recommendations of previous POP Project performance assessments.

So far, the POP Project has concluded 3 PoC activities, with a further 2 in progress. This number is expected, as PoC activities are preceded in general by a Performance Plan activity. Also, PoC activities bind significant more resources on the customer side, which needs time to organize, or can be accommodated only at a later point in time by the customer.

The PoC activities have in all cases led to significant improvement of the respective code's performance under the conditions define at the beginning of the activity. More importantly however, follow-up communication done under the work package 2 shows, that significant improvement are also realised under production conditions on arbitrary data sets.

While the lowish number of completed PoC does not allow to draw any robust conclusion, it is worth to note, that a large part of the codes struggle with single-core performance, not only parallel performance. Also, most application suffer from some kind of load imbalance. However, we have observed various reasons for load imbalance, as for instance data-locality issues which lead to different execution times depending where the task is allocated, but also classical imbalance in the amount of computation or communication volume.

Apart from the immediate advantage for the customers, we have also exemplified, that good OpenMP scalability can be achieved on large hierarchical NUMA systems even for tightly coupled problems (see section 5). This fact is not taken for granted in the HPC user community. We have also shown that clever single-core optimisations can be portable across architectures as disparate as scalar x86 versus vector-processor suc as the NEC SX-ACE (see 3). Finally, our PoC activities have uncovered a bug in a very mature Fortran compiler.

Acronyms and Abbreviations

- BSC: Barcelona Supercomputing Center
- CA: Consortium Agreement
- CAdv: Customer Advocate
- DoA Description of Action (Annex 1 of the Grant Agreement)
- D: deliverable
- EC European Commission
- FFT: Fast-Fourier-Transform
- GA: General Assembly / Grant Agreement
- HLRS: High Performance Computing Centre (University of Stuttgart)
- HPC: High Performance Computing
- IPR Intellectual Property Right
- Juelich: Forschungszentrum Juelich GmbH
- KPI: Key Performance Indicator
- NUMA: non-uniform memory access
- M: Month
- MPI: Message-Passing Interface, a shared-memory programming model
- MS: Milestones
- OpenMP: a shared-memory programming model
- PEB: Project Executive Board
- PM: Person month / Project manager
- PoC: Proof-of-Concept
- POP: Performance Optimization and Productivity
- R: Risk
- RV: Review
- RWTH Aachen: Rheinisch-Westfaelische Technische Hochschule Aachen
- USTUTT (HLRS): University of Stuttgart
- WP: Work Package
- WPL: Work Package Leader