Jülich Supercomputing Centre

Technical Report

JUQUEEN Extreme Scaling Workshop 2015

D. Brömmel, W. Frings, B. J. N. Wylie (Eds.)

FZJ-JSC-IB-2015-01

JÜLICH
FORSCHUNGSZENTRUM

FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. +49 (2461) 61-6402

Technical Report

JUQUEEN Extreme Scaling Workshop 2015

D. Brömmel, W. Frings, B. J. N. Wylie
(Eds.)

FZJ-JSC-IB-2015-01

(last change: 27.02.2015)

Contents

JUQUEEN Extreme Scaling Workshop 2015

Dirk Brömmel, Wolfgang Frings, and Brian J. N. Wylie

Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH

Executive Summary

From 5 to 6 February 2015, Jülich Supercomputing Centre (JSC) organised the latest edition of its series of Blue Gene Extreme Scaling Workshops. These workshops started with the 2008 Blue Gene Porting, Tuning & Scaling Workshop [1] using the JUGENE Blue Gene/P, then followed by dedicated Extreme Scaling Workshops in 2009 [2], 2010 [3] and 2011 [4]. These latter three workshops attracted 28 teams selected from around the world to investigate scalability on the most massively-parallel supercomputer at the time with its 294 912 cores. 26 of their codes were successfully executed at that scale, three became ACM Gordon Bell prize finalists, and one participant was awarded an ACM/IEEE-CS George Michael Memorial HPC fellowship. The Leibniz Supercomputing Centre (LRZ) adopted a similar format for workshops in 2013 [5] and 2014 [6] to scale applications on the SuperMUC IBM iDataPlex system, and from 28 participating code teams three succeeded in running on all 18 "thin node" islands (147 456 cores in total).

The focus for the current workshop was on application codes likely to be able to scale during the workshop to run on the full JUQUEEN system [7]. This 28-rack IBM Blue Gene/Q with 28 672 compute nodes, consisting of 1.6 GHz PowerPC A2 processors each with 16 cores (64 hardware threads) and 16 GB of node memory, has a total of 458 752 cores capable of running 1 835 008 processes or threads. A broad variety of 17 application codes which have demonstrated that they can productively exploit the entire JUQUEEN resources have already been recognised as members of the High-Q Club [8]. The High-Q Club is a collection of the highest scaling codes on JUQUEEN and as such requires the codes to run on all 28 racks. Codes also have to demonstrate that they profit from each additional rack of JUQUEEN in reduced time to solution when strong scaling a fixed problem size or a tolerable increase in runtime when weak scaling progressively larger problems. Furthermore the application configurations should be beyond toy examples and we encourage use of all available hardware threads which is often best achieved via mixed-mode programming. Each code is then individually evaluated based on its weak or strong scaling results with no strict limit on efficiency. The workshop thus provided an opportunity for additional candidates to prove their scalability and qualify for membership, or – as was the case for one of the codes – improve on the scaling and efficiency that they had already achieved.

Seven application teams were invited to stay for two days and work on the scalability of their codes, with dedicated access to the entire JUQUEEN system for a period of 30 hours. Most of the teams' codes had thematic overlap with JSC Simulation Laboratories or were part of an ongoing collaboration with one of the SimLabs. Following earlier tradition, the 2015 Extreme Scaling Workshop was directly preceded by a Porting and Tuning Workshop, offered by JSC as part of the PRACE Advanced Training Centre (PATC) curriculum. Hence most of the application teams were among the 25 new and more experienced users of JUQUEEN who were also present for the prior three days and used the opportunity for initial preparations, performance analyses and tuning tips.

During both workshops the code teams were supported by JSC Cross-sectional teams and Climate Science, Fluids & Solids Engineering and Neuroscience SimLabs, along with IBM and JUQUEEN technical support. Particular thanks are due to Sandra Diaz, Markus Geimer, Klaus Görgen, Sabine Grießbach, Lars Hoffmann, Michael Knobloch, Alex Peyser, Christoph Pospiech, Michael Rambadt, Michael Schlottke, Michael Stephan, Alexandre Strube and Kay Thust, and the workshop participants themselves who openly shared their own knowledge and expertise.

The seven participating code teams[1] were:

- CoreNeuron *electrical activity of neuronal networks with morphologically-detailed neurons*
  Fabien Delalondre, Pramod Kumbhar, **Aleksandr Ovcharenko** (Blue Brain Project, EPFL), and Michael Hines (Yale University)
- ex_nl/FE^2 *scale-bridging approach incorporating micro-mechanics in macroscopic simulations of multi-phase steels*
  Axel Klawonn and **Martin Lanser** (University of Cologne), **Oliver Rheinbach** (TU Freiberg), Jörg Schröder (University Duisburg-Essen), Daniel Balzani (TU Dresden), and Gerhard Wellein (University Erlangen-Nürnberg)
- FEMPAR *massively-parallel finite-element simulation of multi-physics problems governed by PDEs* — High-Q Club member since Dec. 2014
  Santiago Badia, **Alberto F. Martín**, and Javier Principe (Centre Internacional de Mètodes Numèrics a l'Enginyeria (CIMNE), Universitat Politècnica de Catalunya)
- ICON *icosahedral non-hydrostatic atmospheric model*
  Catrin Meyer (Forschungszentrum Jülich GmbH, JSC) and **Thomas Jahns** (Deutsches Klimarechenzentrum GmbH)
- MPAS-A *multi-scale non-hydrostatic atmospheric model for global, convection-resolving climate simulations*
  Dominikus Heinzeller (Karlsruhe Inst. of Technology, Inst. of Meteorology and Climate Research) and Michael Duda (National Center for Atmospheric Research, Earth System Laboratory)
- psOpen *direct numerical simulation of fine-scale turbulence*
  Jens Henrik Goebbert (Jülich Aachen Research Alliance) and **Michael Gauding** (TU Freiberg)
- SHOCK *structured high-order finite-difference computational kernel for direct numerical simulation of compressible flow*
  Manuel Gageik and Igor Klioutchnikov (Shock Wave Lab., RWTH Aachen University)

The workshop surpassed our expectations and completely achieved its goal: all seven teams succeeded in running and validating their codes on 28 racks within the first 24 hours of access to the full JUQUEEN system. They also demonstrated excellent strong and/or weak scaling which qualifies five new members for the High-Q Club: unfortunately, MPAS-A scaling was limited to only 24 racks (393 216 cores). In this case, the dataset used was insufficient to have a performance benefit with 28 racks. A total of 370 'large' jobs were executed (58 on 28 racks, 19 on 24 racks, 6 on 20 racks and 105 on 16 racks) using 12 of the 15 million core-hours reserved for the workshop. Most of the familiar LoadLeveler job scheduling quirks were avoided by deft sysadmin intervention, and a single nodeboard failure requiring a reset resulted in only a short outage when smaller jobs could be executed on the remaining racks.

[1] Workshop participants marked in **bold**

Summary of Results

Selected results from the workshop are summarised in this section, pointing out problems or limitations encountered. Some aspects are compared in relation to the other 16 codes in the High-Q Club where this is insightful.

Languages vs. Models, Memory vs. Threads Since Blue Gene/Q offers lower-level function calls for some hardware-specific features that are sometimes not available for all programming languages, a starting point is looking at the languages used. The left of Figure 1 shows a Venn set diagram of the programming language(s) used, combining workshop and High-Q Club codes. It indicates that all three major programming languages are equally popular (without considering lines of code). Of the workshop codes, three combine Fortran with C, two used C and C++, and the remaining two exclusively used only either Fortran or C.

The four hardware threads per core of the Blue Gene/Q chip in conjunction with the limited amount of memory suggest to make use of multi-threaded programming. It is therefore interesting to see whether this is indeed the preferred programming model and whether the available memory is an issue. The right of Figure 1 shows a Venn set diagram of the programming models used, again combining workshop and High-Q Club codes, and revealing that mixed-mode programming does indeed dominate. Looking at the workshop codes in particular, all seven used MPI, which is almost ubiquitous for portable distributed-memory parallelisation. dynQCD for example is the only High-Q Club application employing lower-level machine-specific SPI for maximum performance. Three of the workshop codes exclusively used MPI for their scaling runs, both between and within compute nodes, accommodating to the restricted per-process memory and even trading higher memory requirements for faster MPI communicator management. The `PAMID_COLLECTIVES_MEMORY_OPTIMIZED` environment variable was critical for FEMPAR and ex_nl/FE^2 to reduce the time of `MPI_Comm_split` from 15 minutes down to under 10 seconds. A memory fragmentation issue in a third-party library currently inhibits the use of OpenMP by FEMPAR. On the other hand MPAS-A just started to include OpenMP multi-threading, whereas an earlier investigation with the SHOCK code found this not to be beneficial. The remaining four workshop codes employ OpenMP multi-threading to exploit compute node shared memory in conjunction with MPI, as is typical of High-Q Club applications in general. Instead of OpenMP, three of the High-Q Club applications prefer POSIX threading for additional control. CoreNeuron has an ongoing effort investigating use of OpenMP-3 tasking and new MPI-3 capabilities (e.g. non-blocking collectives) are under consideration, so these are generally expected to become increasingly important.

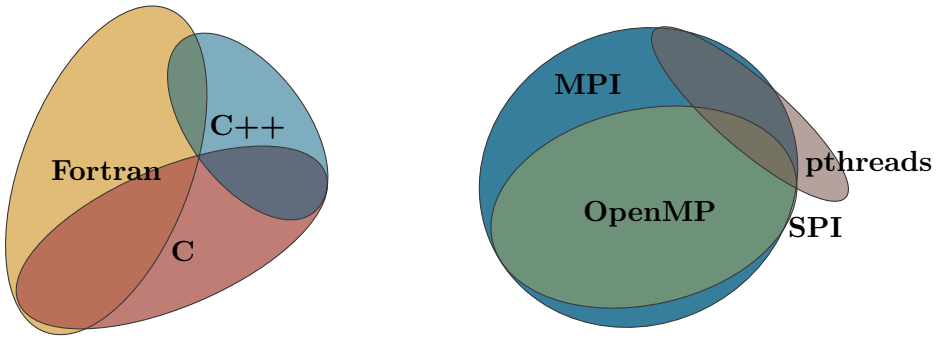

Figure 1: Venn set diagrams of programming languages (left) and parallel programming models (right) used by codes in the High-Q Club and the workshop's participants.

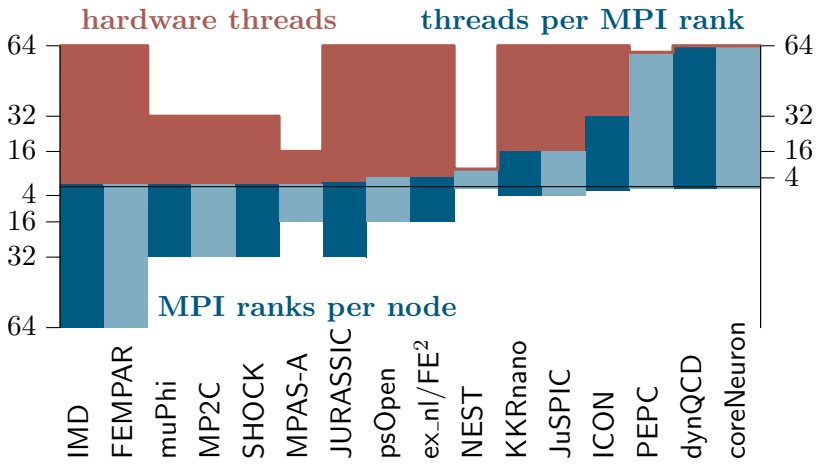

Figure 2: Chart showing the relation between the number of MPI ranks per node and threads per rank used by the mentioned codes. The number of resulting hardware threads used on each compute node is shown in red.

For CoreNeuron available memory is the limiting factor for larger simulations, with the current limit being 155 million neurons using 15.9 GB of RAM. MPAS-A required 1 GB of memory on each process for its regular 3 km mesh simulation (over 65 million grid cells with 41 vertical levels), and could therefore only use a single hardware thread per core, limiting its effective performance. The other six workshop codes benefited from using all four hardware threads of each processor code. In this way FEMPAR was able to increase its efficiency and scalability to 1.75 million processes using $27\frac{1}{2}$ racks of JUQUEEN when employing an additional (fourth-)level of domain decomposition.

Figure 2 shows the relation between the number of MPI ranks and threads per node where this information was available for workshop and High-Q Club codes. On either side of this diagram are the two extremes of using all 64 hardware threads on each CPU by either 64 MPI ranks or 64 threads. Included in red is the resulting number of hardware threads used by the codes, i.e. the concurrency. Clearly, where the information is available, the codes seem to benefit from using more hardware threads than physical cores and favour this configuration.

Weak and strong scaling and performance An overview of the results of a scaling workshop entails some form of comparison of achievements in *strong* (fixed total problem size) and *weak* (fixed problem size per process or thread) scaling, put in context of the scalability results from other codes in the High-Q Club.

Figures 3 and 4 show strong and weak scaling results of the workshop codes, including in grey results from a selection of High-Q Club codes. This indicates the spread in execution results and diverse scaling characteristics of the codes. The figures show that the workshop codes not only managed to run on the full JUQUEEN system, but they also achieved very nice scalability and most therefore qualified for High-Q Club status as an outcome of the workshop. Note that in many cases the graphs do not have a common baseline of one rack since datasets sometimes did not fit available memory or no data was provided for 1024 compute nodes: for strong scaling an execution with a mimimum of seven racks (one quarter of JUQUEEN) is accepted for a baseline, measurement with perfect-scaling assumed from a single rack to the baseline).

In Figure 3 almost ideal strong-scaling speed-up of 27x on 28 racks is achieved by SHOCK, whereas ICON only achieved a modest 12x speed-up, and the other workshop codes in between.

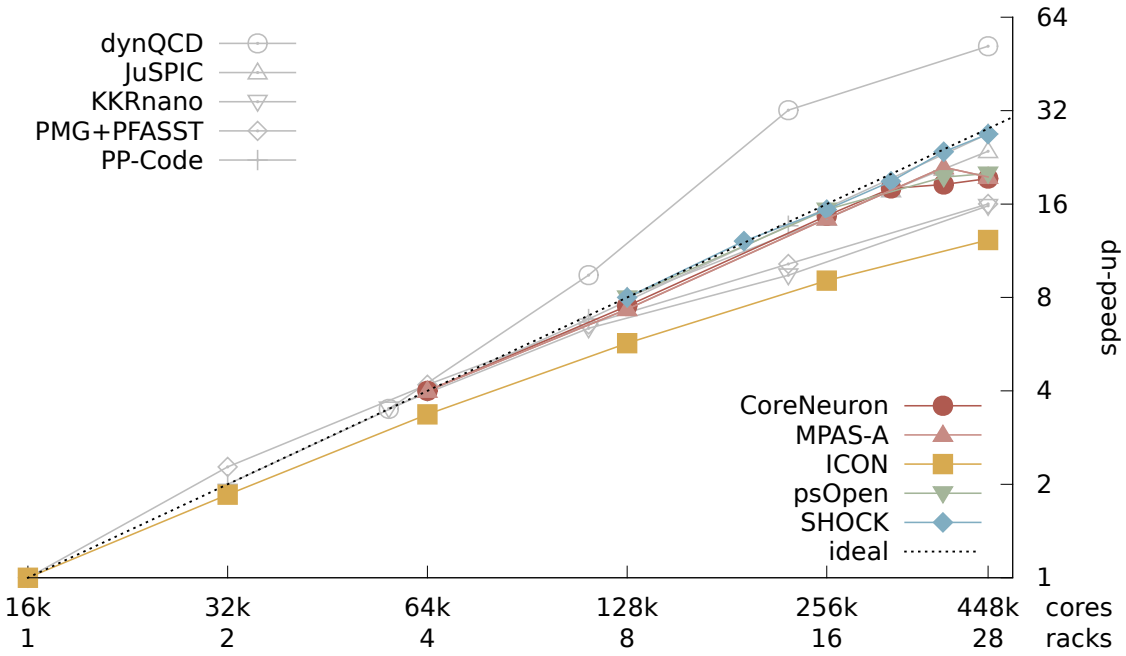

Figure 3: Strong scaling results of the workshop codes with results from existing High-Q Club members included in light grey.

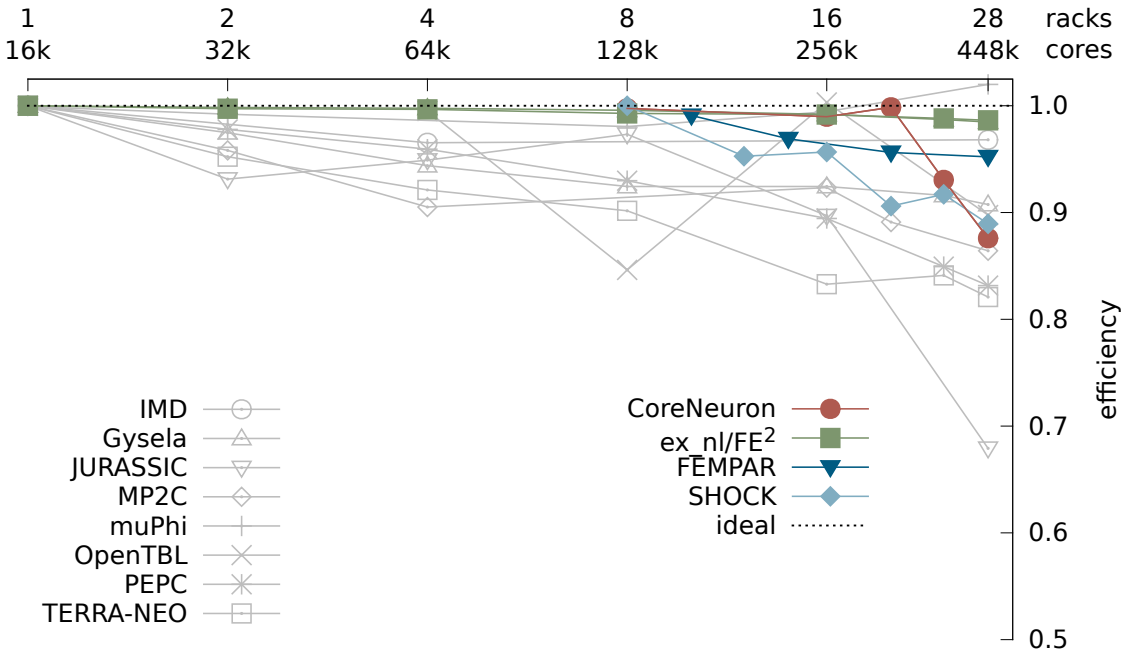

Figure 4: Weak scaling results of the workshop codes with results from existing High-Q Club members included in light grey.

dynQCD stands out with superlinear speed-up of 52x, probably due to its exceptional ability to exploit caches as problem size per thread decreases.

Even with its heroic dataset of over 65 million grid-points, MPAS-A suffered substantial performance breakdown in strong scaling going from 24 to 28 racks due to growing communication costs overwhelming diminishing per-rank computation. A similar breakdown was also found with SHOCK when strong scaling with 64 MPI ranks per compute node (but not evident with only 32 rpn). In both cases, larger datasets might avoid this breakdown.

In Figure 4 ex_nl/FE^2 is able to sustain weak scaling efficiency of 99% with 28 racks, whereas for CoreNeuron efficiency drops to a still respectable 88%. muPhi was able to achieve 102% efficiency on 28 racks compared with a single rack, whereas JURASSIC only managed 68% efficiency due to excessive I/O for the reduced-size test case. Various codes show erratic scaling performance, most likely due to topological effects. SHOCK is characterised by particularly poor configurations with an odd number of racks in one dimension (i.e. 4×3, 4×5 and 4×7). Similarly, OpenTBL shows marked efficiency drops for non-square numbers of racks (8 and 28).

Most optimisations employed by the codes are not specific to Blue Gene (or BG/Q) systems, but can also be exploited on other highly-parallel systems. High-Q Club codes have also run at scale on various Cray supercomputers, K computer, MareNostrum-3, SuperMUC and other x86-based computers, as well as on systems with GPGPUs.

Custom mappings of MPI process ranks to JUQUEEN compute nodes generated by the Rubik [13] tool were investigated with psOpen and found to deliver some benefits, however, for the largest machine partitions these did not provide the expected reduction in communication times yet suffered from increased application launch/initialisation time.

Detailed results for each code are found in the following chapters with reports provided by each of the participating teams. These present and discuss more execution configurations and scaling results achieved by the application codes during the workshop.

Managing I/O Another critical point attracting increasing attention is performance of file I/O, which is often a scalability constraint for codes which need to read and write huge datasets or open a large number of files. MPAS-A needed 20 minutes to load its initial condition data of 1.2 TB using PIO/NetCDF — after taking the better part of a week to transfer this single file from KIT (due to the source institution policy limiting outgoing transfer bandwidth) — and simulation output was disabled for these tests to avoid similar writing inefficiency. Large-scale executions of ICON, psOpen and SHOCK (and various High-Q member codes) using the popular HDF5 and pNetCDF libraries needed to disable file I/O and synthesise initialisation data, whereas CoreNeuron replicated a small dataset to fill memory to 15.9 GB.

SIONlib [9], which was developed to address file I/O scalability limitations, has been used effectively by three High-Q codes (KKRnano, MP2C and muPhi) and several other applications are currently migrating to adopt it (e.g., NEST).

Tools at scale Darshan [10] was engaged with SHOCK and various other codes to investigate I/O performance on JUQUEEN and identify copious reads of small numbers of bytes, but found not to work with applications written in C++ or Fortran. The Score-P instrumentation and measurement infrastructure [11] employed by the latest release of Scalasca [12] was used to profile file I/O performance of MPAS-A, however, only MPI call count and timing is currently supported and not measurement of bytes read or written. While Score-P profiles have been produced for applications with one million threads, the largest trace collection configuration currently handled with OTF2 is approximately 655 360 threads (or processes). SIONlib needs to be employed for such traces to avoid file creation limitations, however, excessive memory requirements have so far prevented Scalasca automated parallel trace analysis of these.

High-Q Club codes

The full description of the High-Q Club codes along with developer and contact information can be found on the web page [8]. The current list has 17 codes, one of which is FEMPAR. The others are:

dynQCD *lattice quantum chromodynamics with dynamical fermions*
JSC SimLab Nuclear and Particle Physics & Bergische Universität Wuppertal

Gysela *gyrokinetic semi-Lagrangian code for plasma turbulence simulations*
CEA-IRFM Cadarache

IMD *classical molecular dynamics simulations*
Ruhr-Universität Bochum & JSC SimLab Molecular Systems

JURASSIC *solver for infrared radiative transfer in the Earth's atmosphere*
JSC SimLab Climate Science

JuSPIC *fully relativistic particle-in-cell code for plasma physics and laser-plasma interaction*
JSC SimLab Plasma Physics

KKRnano *Korringa-Kohn-Rostoker Green function code for quantum description of nanomaterials in all-electron density-functional calculations*
FZJ-IAS

MP2C *massively-parallel multi-particle collision dynamics for soft matter physics and mesoscopic hydrodynamics*
JSC SimLab Molecular Systems

$\mu\phi$ (muPhi) *modelling and simulation of water flow and solute transport in porous media, algebraic multi-grid solver*
Universität Heidelberg

Musubi *multi-component Lattice Boltzmann solver for flow simulations*
Universität Siegen

NEST *large-scale simulations of biological neuronal networks*
FZJ/INM-6 & IAS-6

OpenTBL *direct numerical simulation of turbulent flows*
Universidad Politécnica de Madrid

PEPC *tree code for N-body simulations, beam-plasma interaction, vortex dynamics, gravitational interaction, molecular dynamics simulations*
JSC SimLab Plasma Physics

PMG+PFASST *space-time parallel solver for systems of ODEs with linear stiff terms, e.g. from lines discretisations of PDEs*
LBNL, Universität Wuppertal, Università della Svizzera italiana & JSC

PP-Code *simulations of relativistic and non-relativistic astrophysical plasmas*
University of Copenhagen

TERRA-NEO *modeling and simulation of earth mantle dynamics*
Universität Erlangen-Nürnberg, LMU & TUM

waLBerla *Lattice-Boltzmann method for the simulation of fluid scenarios*
Universität Erlangen-Nürnberg

References

[1] Bernd Mohr & Wolfgang Frings, Jülich Blue Gene/P Porting, Tuning & Scaling Workshop 2008, Innovatives Supercomputing in Deutschland, inSiDE 6(2), 2008.

[2] Bernd Mohr, Wolfgang Frings, Jülich Blue Gene/P Extreme Scaling Workshop 2009, Technical Report FZJ-JSC-IB-2010-02, Forschungszentrum Jülich, Feb. 2010. http://juser.fz-juelich.de/record/8924

[3] Bernd Mohr, Wolfgang Frings, Jülich Blue Gene/P Extreme Scaling Workshop 2010, Technical Report FZJ-JSC-IB-2010-03, Forschungszentrum Jülich, May 2010. http://juser.fz-juelich.de/record/9600

[4] Bernd Mohr, Wolfgang Frings, Jülich Blue Gene/P Extreme Scaling Workshop 2011, Technical Report FZJ-JSC-IB-2011-02, Forschungszentrum Jülich, Apr. 2011. http://juser.fz-juelich.de/record/15866

[5] Helmut Satzger et al, Extreme Scaling of Real World Applications to >130,000 Cores on SuperMUC, Poster, Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC13, Denver, CO, USA), Nov. 2013.

[6] Ferdinand Jamitzky, Helmut Satzger, 2nd Extreme Scaling Workshop on SuperMUC, Innovatives Supercomputing in Deutschland, inSiDE 12(2), 2014.

[7] JUQUEEN – Jülich Blue Gene/Q, Jülich Supercomputing Centre. http://www.fz-juelich.de/ias/jsc/juqueen/

[8] The High-Q Club at JSC. http://www.fz-juelich.de/ias/jsc/high-q-club

[9] SIONlib: Scalable I/O library for parallel access to task-local files. http://www.fz-juelich.de/jsc/sionlib

[10] Darshan: HPC I/O characterisation tool, Argonne National Laboratory. http://www.mcs.anl.gov/research/projects/darshan/

[11] Score-P: Community-developed scalable instrumentation and measurement infrastructure. http://www.score-p.org/

[12] Scalasca: Toolset for scalable performance analysis of large-scale parallel applications. http://www.scalasca.org/

[13] Rubik tool for generating structured Cartesian communication mappings, Lawrence Livermore National Laboratory. https://computation.llnl.gov/project/performance-analysis-through-visualization/software.php

CoreNeuron, the Blue Brain Project

Fabien Delalondre[1], Pramod Kumbhar[1], Aleksandr Ovcharenko[1], and Michael Hines[2]

[1]Blue Brain Project, EPFL
[2]Yale University

CoreNeuron Description

The aim of the Extreme Scaling Workshop for the HPC team of the Blue Brain Project, in collaboration with Michael Hines from Yale University, was to execute an electrical simulation of a neuronal network using the CoreNeuron simulator utilising the full JUQUEEN system, analyse the code performance and address any possible issues arising from operating at such an extreme scale.

CoreNeuron supports a reduced set of the functionalities offered by the open source simulator NEURON [1]. The software aims at supporting an efficient and scalable simulation of the electrical activity of neuronal networks that include morphologically detailed neurons. CoreNeuron has been implemented with the goal of minimising memory footprint and obtaining optimal performance, relying on the use of a single MPI process per node and 64 OpenMP threads on IBM Blue Gene/Q systems. The computational workflow of CoreNeuron includes various computation and communication steps that are presented in Figure 1.

CoreNeuron is written in C/C++ and utilises a hybrid (MPI + OpenMP) parallelisation. The initial circuit utilised for testing contained 3 million neurons. Its visual representation of the neural network being simulated on a single rack of IBM Blue Gene/Q can be found in Figure 2. In order to extend the problem to larger scales, the circuit is replicated in memory by some factor, both for cells and connections, allowing us to conduct scaling studies and still remain within the memory bounds. During the workshop, the data was read from disk in either a "file-per-process" or "file-per-thread" manner. Replication was then performed "in-memory", whereupon all threads would then receive roughly the same amount of compute work. The total I/O size for 28 racks was 28 TB, comprising data in both binary and ASCII formats. Without in-memory replication, the I/O demands for the 24 million neuron circuit would scale linearly, totalling 224 TB. The developers are now in a process of transitioning to a more optimised parallel I/O implementation based on the HDF5 library [2].

Results

As described above, the simulator was configured to run with 1 MPI rank per node with 64 OpenMP threads. Strong scaling results were obtained by running tests on 4, 8, 16, 20, 24, and 28 racks, where the size of the problem remained the same across all runs and consisted of simulating 24 million morphologically detailed neurons for 10 milliseconds of biological time. Weak scaling results were obtained by executing tests on 4, 8, 16, 20, 24, and 28 racks, where the problem size was the same per node, i.e. 2 906 neurons per node, or roughly 45 neurons

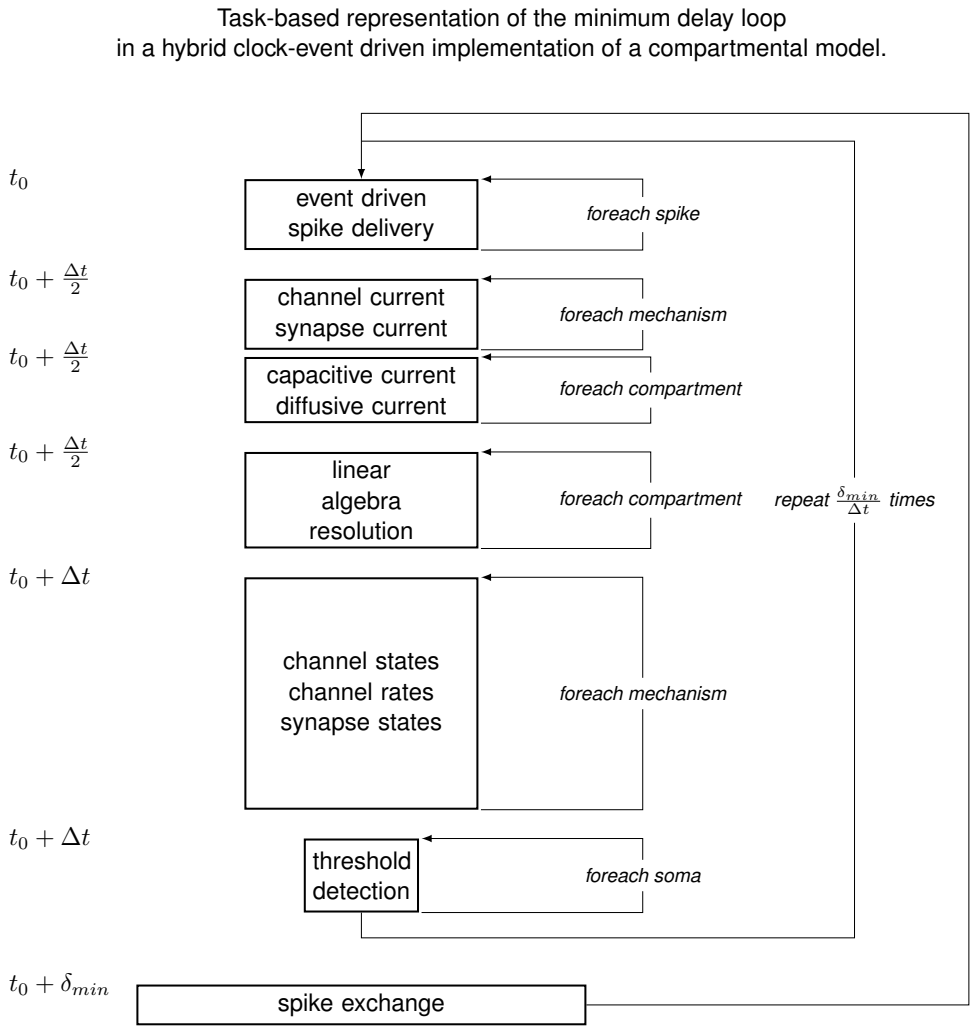

Figure 1: CoreNeuron workflow.

per thread, while simulating 10 milliseconds of biological time. In the weak scaling tests, the replication factor was the same as the number of racks employed in the simulation.

The strong scaling results shown in Table 1, Figure 3 and Figure 4 indicate that while simulation time decreases with the increased number of racks, MPI communication time (`MPI_Allgather` and `MPI_Allgatherv`) increased when using 20 or more racks, which is reflected in the overall simulation time. The load imbalance starts to increase after 16 racks due to the fact that the problem at hand was not exactly divisible by the number of threads being used beyond 16 racks, essentially creating an unbalanced problem. The simulation data available at the time of the workshop had been arranged so as to be distributable across a power of two number of racks, and consequently it was not able to be uniformly distributed across 20, 24 and 28 racks. The authors strongly believe that such an issue will disappear when the next implementation of parallel I/O and corresponding static load balancing functionalities which are under development will be fully in place. It is then worthwhile to analyse the strong scaling data up to 16 racks, the largest number of racks usable without the load imbalance issue arising from the misconfiguration of the problem setup. Up to 16 racks, we see that the

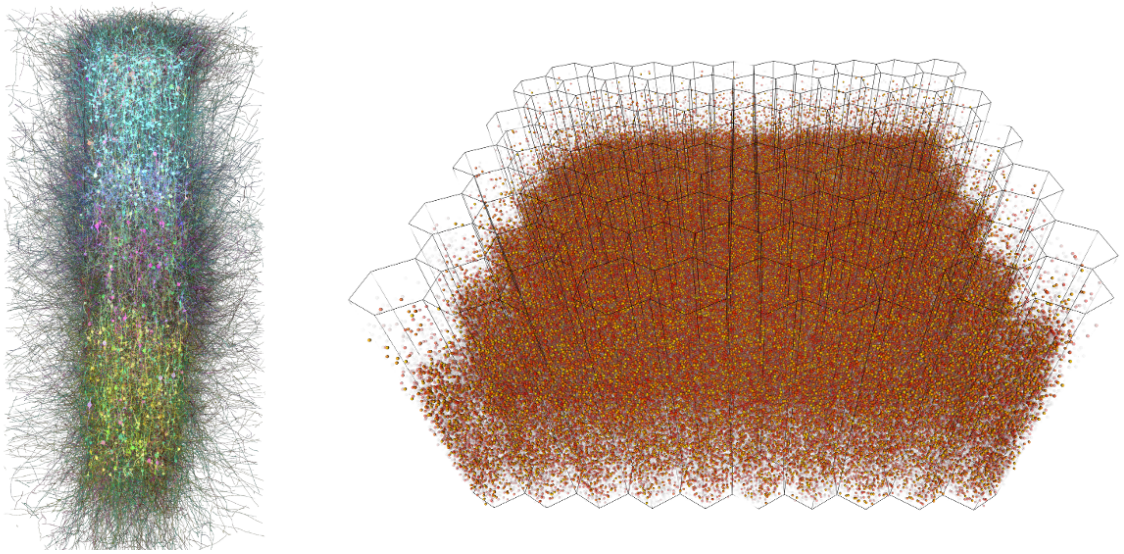

Figure 2: Simulation of 3 million neurons on a single IBM Blue Gene/Q rack. On the left is the front view of a single column from the circuit, together with its spike activity. On the right is a top-side view of the full 3 million circuit, displaying the somas with a membrane voltage above a certain threshold.

Table 1: CoreNeuron strong scaling tests.

racks	neurons/node	DRAM [GB]	simulation [s]	MPI [s]	setup [s]	total [hours]
4	5 813	14.2	412.121	0.441	193.250	7 503
8	2 907	7.94	219.528	0.572	136.569	7 993
16	1 454	4.13	112.281	0.695	81.143	8 177
20	1 090	3.16	91.800	4.749	68.223	8 356
24	909	2.69	89.070	5.748	62.431	9 729
28	818	2.45	85.250	5.985	62.575	10 864

code loses about 10% of strong scaling efficiency and the authors are currently investigating which part of the workflow is responsible for the loss.

The weak scaling results presented in Table 2, Figure 5 and Figure 6 show that the parallel efficiency remains nearly optimal up to 20 racks, and the time needed to complete the simulation is roughly the same. As described earlier, owing to the configuration of our input data available during the workshop, the load imbalance issue presents itself in results of the weak scaling runs on 24 and 28 racks. As the problem is distributed across more compute nodes, the spike exchange processing increases for each MPI task and thus affects the weak scaling behaviour.

Table 2: CoreNeuron weak scaling tests.

racks	total neurons	DRAM [GB]	simulation [s]	MPI [s]	setup [s]	total [hours]
4	11 904 000	7.94	214.951	0.291	219.327	3 914
8	23 808 000	7.94	219.528	0.572	136.569	7 993
16	47 616 000	7.93	221.276	1.003	89.010	16 113
20	55 800 000	7.43	219.318	14.462	85.223	19 963
24	66 960 000	7.43	235.333	22.255	71.386	25 705
28	82 024 264	7.80	250.025	29.268	50.490	31 861

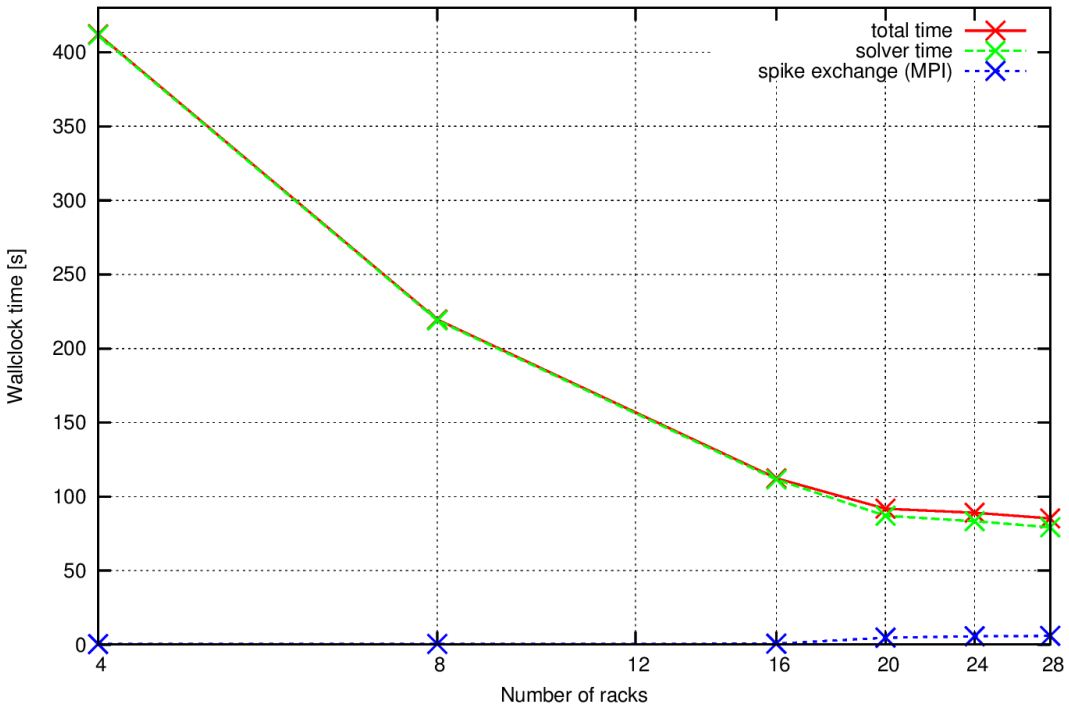

Figure 3: CoreNeuron simulation time for the strong scaling experiments.

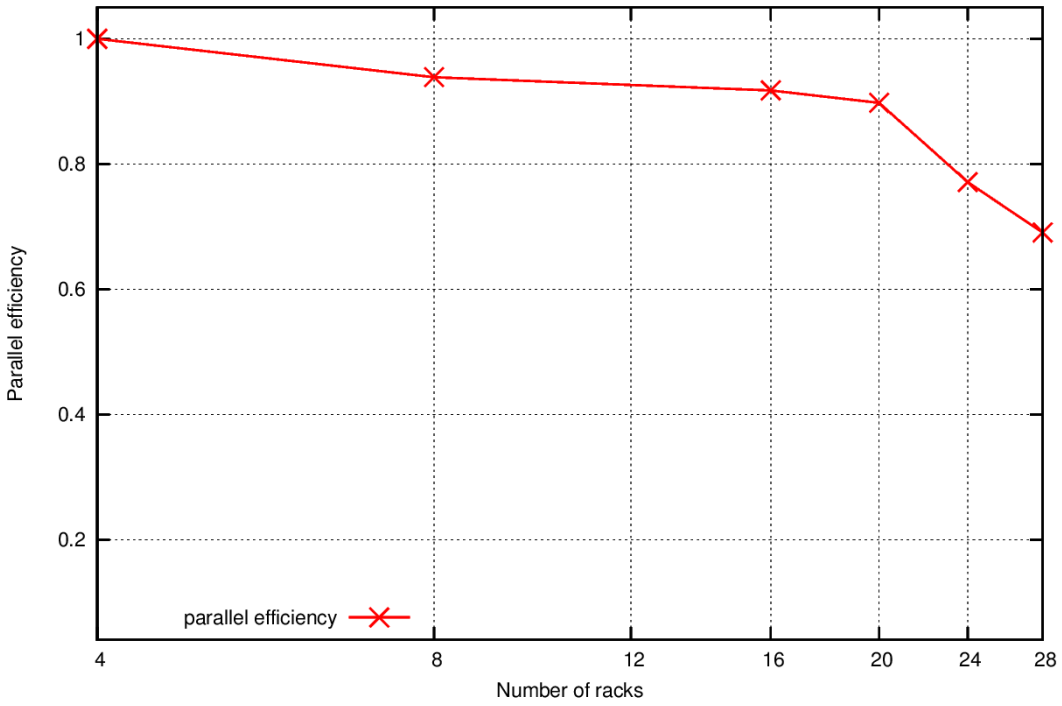

Figure 4: CoreNeuron parallel efficiency for the strong scaling experiments.

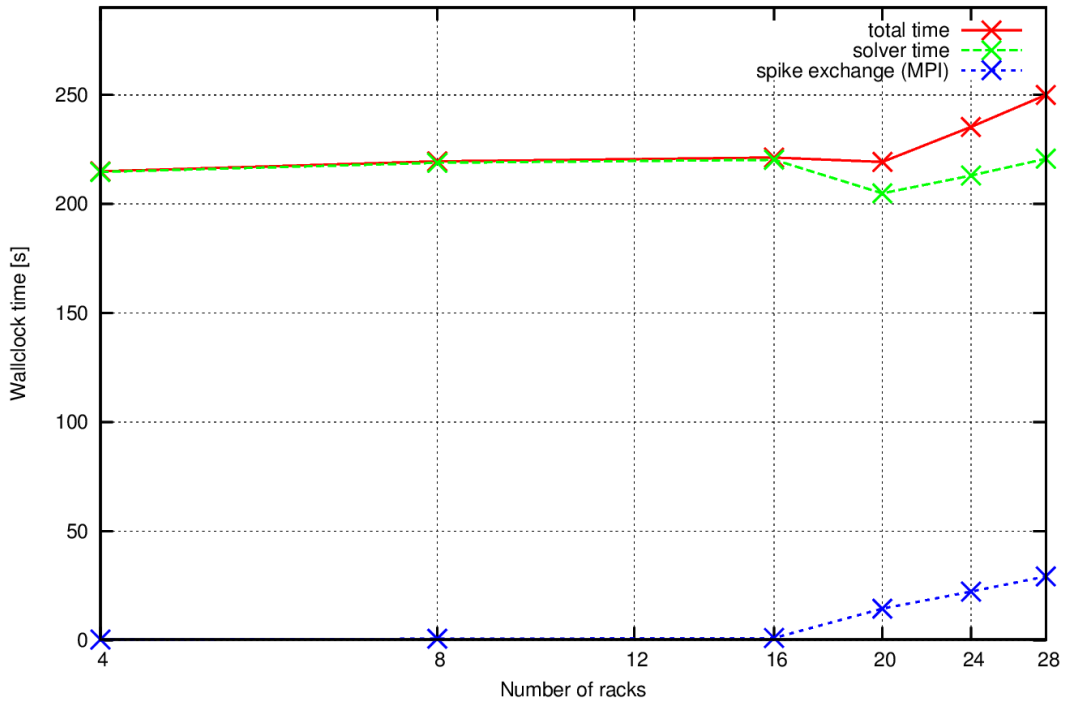

Figure 5: CoreNeuron simulation time for the weak scaling experiments.

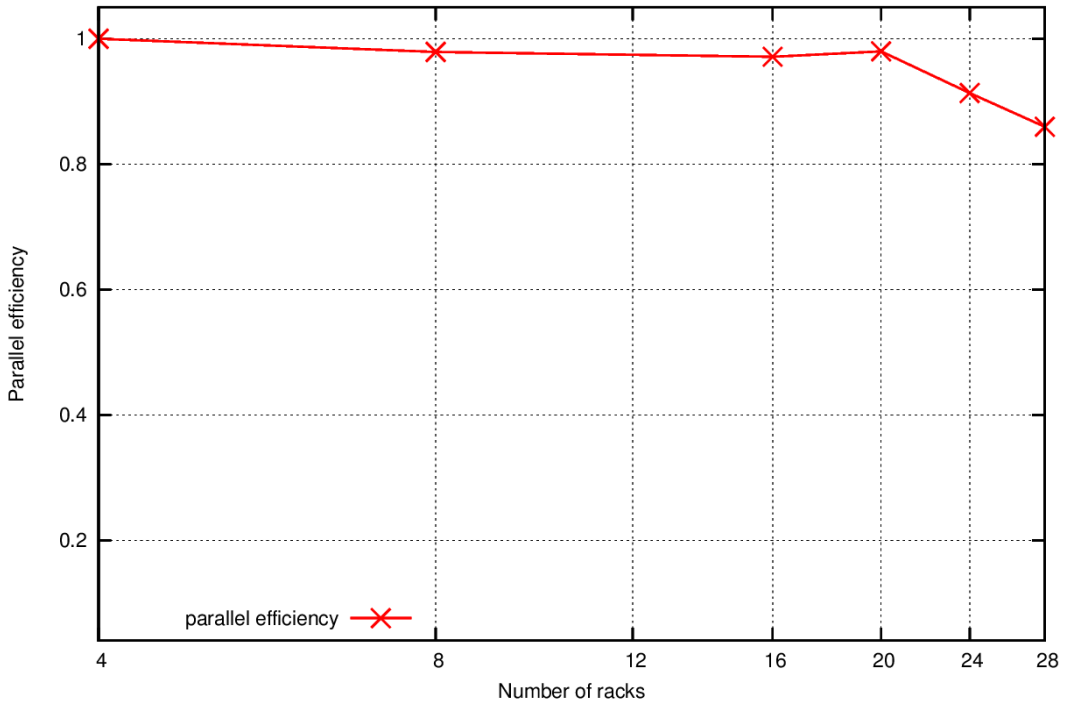

Figure 6: CoreNeuron parallel efficiency for the weak scaling experiments.

To examine scaling behaviour at the limit of available memory, and to see how many neurons could be simulated on the full JUQUEEN BG/Q, a simulation was run with a total of 155 million neurons, utilising a maximum of 15.9 GB of DRAM. The simulation took 491 seconds, 1.96 times longer than the 82 million neuron simulation.

In order to improve the on-node performance of the simulator, the developers of CoreNeuron have been working on transforming the representation of data in memory from an arrays-of-structures (AoS) layout to structures-of-arrays (SoA). Even with an SoA memory layout and corresponding transformation of the compute kernels the XL C compiler is not able to vectorise most of the kernels (note that these kernels are vectorised by Intel and PGI compilers).

Most of the compute kernels have the following form:

```
#pragma ibm independent_loop
for (i = 0; i < cntml; i++)
{
       p_2[i] =  p_5[i] * p_3[i] * p_3[i] * p_3[i] * p_4[i];
       p_1[i] = p_2[i] * ( p_3[i] - p_5[i] );
}
```

Though the iteration of loops are independent and have only simple arithmetic, the loops are non-vectorisable on BG/Q (note the forward dependency of `p_2` in the above loop). This problem has been discussed with the XL C developers. The XL C compiler could not vectorise the kernels because of a limitation in the vector loads and stores on BG/Q hardware. The XL compiler has to perform a specialised procedure in which each store and load must be broken up into a lower part, an upper part, and then merged together. Due to this process, forward dependencies with distance zero like the ones shown above present a data hazard. The XL C developers have suggested that a manual distribution of compute loops could break these dependencies. Considering the large number of auto-generated kernels from the NEURON mechanism DSL, it appears infeasible to manually transform these loops. The authors are investigating other options.

References

[1] http://www.neuron.yale.edu/neuron/

[2] http://www.hdfgroup.org/HDF5/

EXASTEEL - Computational Scale Bridging using a FE^{2}TI approach with ex_nl/FE^2

A. Klawonn[1], M. Lanser[1], and O. Rheinbach[2]

[1]Universität zu Köln
[2]Technische Universität Bergakademie Freiberg

Description of the Code

We are concerned with the computational simulation of advanced high strength steels, incorporating phase transformation phenomena at the microscale. Our research project "EXASTEEL – Bridging Scales for Multiphase Steels" is part of the German priority program (DFG-Schwerpunktprogramm 1648) SPPEXA (Software for Exascale Computing) and a joint effort of mathematicians (A. Klawonn, M. Lanser, U Cologne; O. Rheinbach, TU Freiberg), engineers (J. Schröder, U Duisburg-Essen; D. Balzani, TU Dresden), and computer scientists (G. Wellein, U Erlangen-Nürnberg). Our present goal is to bring computational scale bridging to the complete JUQUEEN machine (458 752 BG/Q cores).

The FE^2 method, see, e.g., [1–5], is a computational micro-macro scale bridging approach directly incorporating micromechanics in macroscopic simulations. In this approach, a microscopic boundary value problem based on the definition of a representative volume element (RVE) is solved at each macroscopic Gauß integration point. Then, volumetric averages of

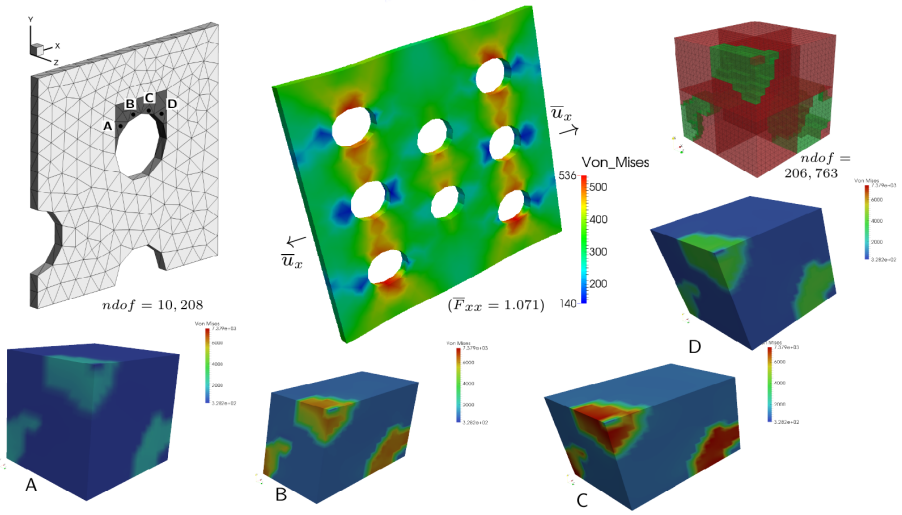

Figure 1: In the FE^2 computational scale bridging method, in each macroscopic Gauß point a microscopic problem is solved.

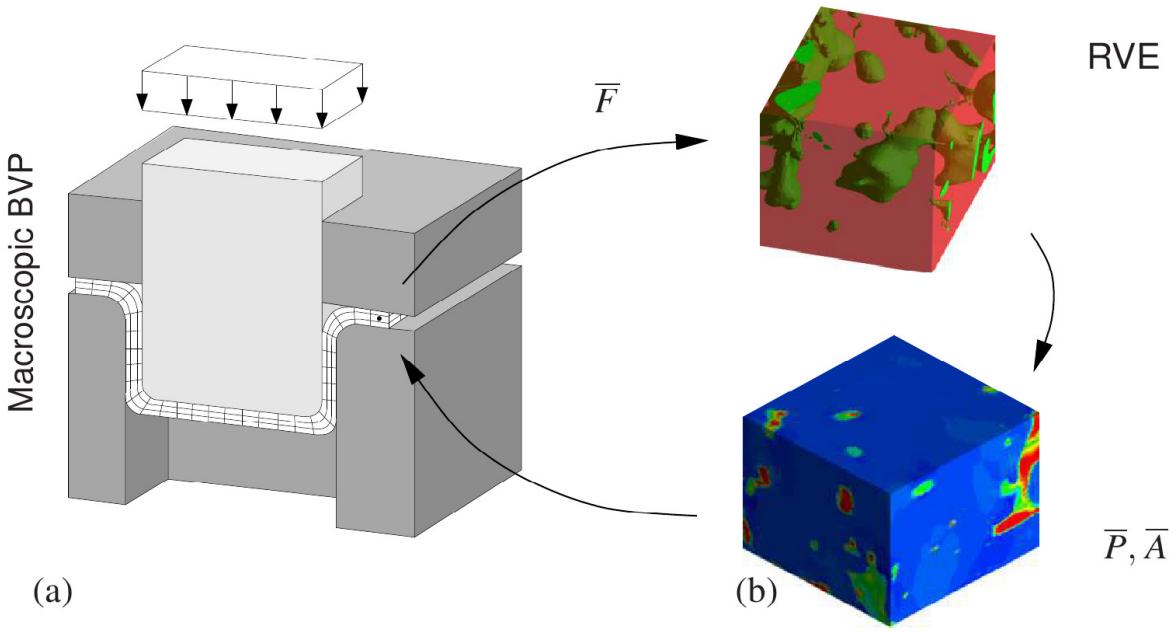

Figure 2: The FE^2 approach with (a) macroscopic boundary value problem (BVP) and (b) microscopic BVP on an RVE.

microscopic stress distributions are returned to the macroscopic level, which replaces a phenomenological material law at the macro scale. The microscopic problems are thus coupled through the macroscopic problem; see Figures 1, 2, and 3.

On the RVEs nonlinear implicit structural mechanics problems have to be solved. We are applying the FETI-DP (Finite Element Tearing and Interconnecting) method as a solver on the RVEs. Nonoverlapping domain decomposition methods of the FETI type [6–15] are well established solution methods in implicit structural mechanics. A structural simulation using a FETI-DP algorithm was awarded an ACM Gordon Bell prize already in 2002 using 4000 processors of the then second fastest supercomputer of the world. Unfortunately, the classical FETI-DP method does not scale well beyond 10K processor cores. Inexact FETI-DP methods [16], have shown a much better parallel scalability, and scalability for 65K cores was shown during the 2008 JUGENE scaling workshop in Jülich [13, 17]. Recently, nonlinear FETI-DP and BDDC methods [18–20] with improved concurrency were introduced. In these methods, the nonlinear problem is decomposed into concurrent subproblems before linearisation. This is opposed to standard Newton-Krylov approaches where the problem is first linearised and then decomposed.

Hybrid parallelisation in our context was discussed in [21]. Nonlinear non-overlapping domain decomposition is not new. It was used, e.g., in multiphysics and fluid-structure-interaction, as a coupling method in the case of a small number of subdomains. Only recently, it has attracted interest as a scalable solver approach [18–20, 22, 23]. The ASPIN method [24] is a related nonlinear overlapping domain decomposition approach as a solver.

We refer to the combination of the FE^2 scale bridging method with a FETI-DP method on each RVE as a FE^2TI method. For our FE^2 method, as a solver on the RVEs, we use a Newton-Krylov-irFETI-DP method using PETSc 3.5.2, hypre 2.9.1a, and UMFPACK 5.6.2 as a direct solver on the subdomains. Our code is written mainly in C/C++. We have used the IBM XL C/C++ compiler for Blue Gene, V12.1.

Repeat until convergence:

1. Apply boundary conditions to RVE based on macroscopic deformation gradient: Enforce $x = \overline{F}X$ on the boundary of the microscopic problem $\partial\mathcal{B}$.
2. Solve microscopic nonlinear boundary value problem using (ir)FETI-DP or related methods.
3. Compute and return macroscopic stresses as volumetric average over microscopic stresses P, i.e., $\overline{P} = \frac{1}{V}\int_{\mathcal{B}} P\,dV$.
4. Compute and return macroscopic tangent moduli as average over microscopic tangent moduli A, i.e., $\overline{A} = \frac{\partial}{\partial\overline{F}}(\frac{1}{V}\int_{\mathcal{B}} P\,dV)$.
5. Set up tangent matrix and rhs of linearised macroscopic boundary value problem using $\overline{P}$ and $\overline{A}$.
6. Solve linearised macroscopic boundary value problem.
7. Update macroscopic deformation gradient $\overline{F}$.

Figure 3: Algorithmic description of the FE^{2}TI approach. Overlined letters denote macroscopic quantities.

Results

We present our scaling results for the computational scale bridging using the FE^2 method in 2D and 3D. For the first time, scalability to 458 752 cores is achieved for our approach. We first scale up the size of the macroscopic problem while keeping the size of the microscopic RVEs fixed. We also keep the number of FETI-DP subdomains for each RVE fixed and use one MPI rank per FETI-DP subdomain. As we increase the number of processor cores in proportion to the problem size (weak scalability), in the best case, for a parallel efficiency of 100%, we would expect a constant time to solution.

In Tables 1 and 2, we see the weak scalability for 2D and 3D; we use one MPI rank for each BG/Q processor core and OpenMP multithreading with 4 threads. The base line for our parallel efficiency is the smallest meaningful macroscopic problem, i.e., with 8 Gauß points in 2D and 16 Gauß points in 3D. A parallel efficiency of approximately 98% is achieved in Tables 1 and 2. In Figures 4 and 5 the data from the tables is depicted.

In our implementation, we use `MPI_Comm_split` to create subcommunicators for the computations on the RVEs. As suggested at the workshop, we used the environment variable `PAMID_COLLECTIVES_MEMORY_OPTIMIZED=1` to keep the time for the communicator split short. In our computations the resulting timings for the communicator split was below 2 seconds.

In Tables 1 and 2, the number of subdomains for each RVE, i.e., 256 in 2D and 512 in 3D, is still small. In Table 3, starting from the largest problem in Table 1, the size of the RVEs is increased by a factor of 4.

Next, we disable threading and consider the effect of an overcommit using pure MPI. In Table 4, we show weak scaling but using an overcommit with up to 4 MPI ranks for each BG/Q processor core. In the latter case, only 256 MB are available for each MPI rank. We use 16, 32, and 64 MPI ranks per node and the RVE size is kept constant, i.e., the total problem size is increased by a factor of 4. We, of course, cannot expect perfect scalability in this situation. But we still see, that acceptable scalability is obtained when scaling from a total of 458 752 MPI ranks to 917 504 MPI ranks, i.e., the total time to solution is 266.47s instead of $2 \cdot 215.41\text{s} = 430.82\text{s}$. Using 1 835 008 MPI ranks only results in small additional savings.

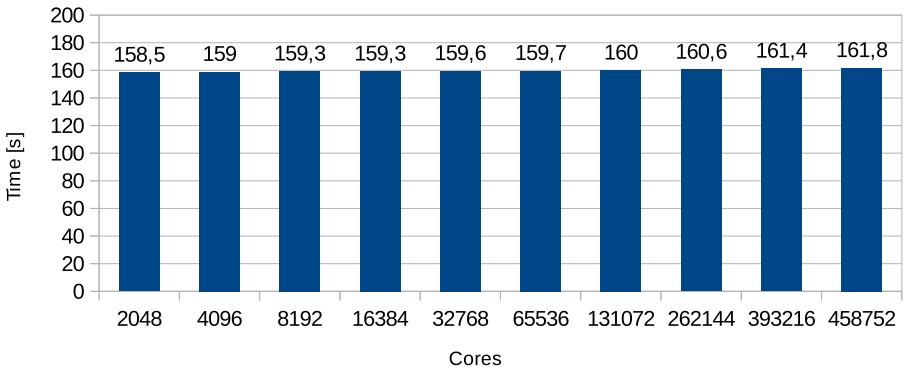

Figure 4: Weak scalability for FE^2 in 2D using FETI-DP on each RVE. Data from Table 1.

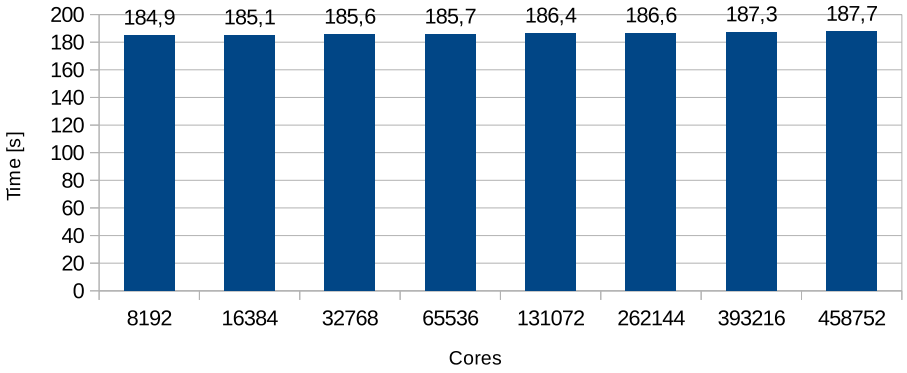

Figure 5: Weak scalability for FE^2 in 3D using FETI-DP on each RVE. Data from Table 2.

The scaling runs presented here are for heterogeneous nonlinear hyperelasticity. In our application, we are also interested in plastification on the RVEs. We will perform corrresponding numerical experiments in the near future.

Although our focus in this scaling workshop was on scale bridging, we also took the opportunity to work on the scalability of a recent implementation of one of our nonlinear FETI-DP methods, which is still work in progress. Here a memory problem was observed for a large number of MPI ranks. Therefore, when scaling from a single BG/Q compute node to 28 672 nodes the total time for a Newton step in phase 2 of the algorithm increased from 3.28s (1 node), 4.21s (16 nodes), 5.31s (256 nodes), 6.98s (4096 nodes) to 15.28s (28 672 nodes). As a result of this workshop, we will analyse the responsible software subpackage and our use of it in detail.

The workshop is valuable to bring together people from different fields with common interest in computing, which would otherwise not meet in person. Especially for young researchers, awareness for the hardware and software ecosystems in HPC on the Tier-0/1 level is created. The workshop encourages and facilitates the use of tools which can in return radically improve productivity. It also allows to work exclusively and cooperatively on the implementation and numerical experiments for a few days.

Table 1: Scaling up the macro problem: FE^2 in 2D using FETI-DP on each RVE; heterogeneous hyperelasticity; P1 finite elements macro, P2 finite elements micro; 5.1 million d.o.f. on each RVE; 256 subdomains for each RVE.

FE^2 in 2D (Weak scaling)				
bg_size	**MPI-ranks**	**#RVEs**	**Time to Solution**	
128	2 048	8	158.47s	100.0%
256	4 096	16	159.03s	99.6%
512	8 192	32	159.27s	99.5%
1 024	16 384	64	159.32s	99.5%
2 048	32 768	128	159.58s	99.3%
4 096	65 536	256	159.68s	99.2%
8 192	131 072	512	159.99s	99.1%
16 384	262 144	1 024	160.62s	98.7%
24 576	393 216	1 536	161.41s	98.2%
28 672	458 752	1 792	161.78s	98.0%

Table 2: FE^2 in 3D using FETI-DP on each RVE; heterogeneous hyperelasticity; Q1 finite elements macro, P2 finite elements micro; 1.6 million d.o.f. on each RVE; 512 subdomains for each RVE.

FE^2 in 3D				
bg_size	**MPI-ranks**	**#RVEs**	**Time to Solution**	
512	8 192	16	184.86s	100.0%
1 024	16 384	32	185.09s	99.9%
2 048	32 768	64	185.61s	99.6%
4 096	65 536	128	185.72s	99.5%
8 192	131 072	256	186.43s	99.2%
16 384	262 144	512	186.61s	99.1%
24 576	393 216	768	187.32s	98.7%
28 672	458 752	896	187.65s	98.5%

Table 3: We increase the RVE sizes starting from the largest problem in Table 1; heterogeneous hyperelasticity; P1 finite elements macro, P2 finite elements micro.

FE^2 in 2D (Increasing RVE sizes)					
bg_size	**MPI-ranks**	**#RVEs**	**RVE-size**	**RVE-size × #RVEs**	**Time to Solution**
28 672	458 752	1 792	5 126 402	9 186 512 384	161.78s
28 672	458 752	1 792	7 380 482	13 225 823 744	248.19s
28 672	458 752	1 792	13 117 442	23 506 456 064	483.68s
28 672	458 752	1 792	20 492 802	36 723 101 184	817.06s

Table 4: Weak scaling efficiency using 16 / 32 / 64 MPI ranks per compute node. FE^2 in 3D using FETI-DP on each RVE. Here, due to the memory constraints, we use 1 594 323 d.o.f. per RVE and 512 subdomains per RVE.

FE^2 in 3D (1x, 2x, 4x MPI overcommit)					
bg_size	**ranks per node**	**MPI-ranks**	**#RVEs**	**Time to Solution**	
28 672	16	458 752	896	215.41s	100%
28 672	32	917 504	1 792	266.47s	81%
28 672	64	1 835 008	3 584	522.10s	41%

Acknowledgements

This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 “Software for Exascale Computing” (SPPEXA) under KL2094/4-1, RH 122/2-1. The authors gratefully acknowledge the use of JUQUEEN during the Workshop on “Extreme Scaling on JUQUEEN” (Jülich, 05.02.2015 - 06.02.2015).

References

[1] R.J.M. Smit, W.A.M. Brekelmans, and H.E.H. Meijer. Prediction of the mechanical behavior of nonlinear heterogeneous systems by multi-level finite element modeling. *Computer Methods in Applied Mechanics and Engineering*, 155:181–192, 1998.

[2] C. Miehe, J. Schröder, and J. Schotte. Computational homogenization analysis in finite plasticity. Simulation of texture development in polycrystalline materials. *Computer Methods in Applied Mechanics and Engineering*, 171:387–418, 1999.

[3] J. Schröder. *Homogenisierungsmethoden der nichtlinearen Kontinuumsmechanik unter Beachtung von Stabilitätsproblemen.* PhD thesis, Bericht aus der Forschungsreihe des Institut für Mechanik (Bauwesen), Lehrstuhl I, Universität Stuttgart, 2000. Habilitationsschrift.

[4] V. Kouznetsova, W.A.M. Brekelmans, and F.P.T. Baaijens. An approach to micro-macro modeling of heterogeneous materials. *Computat. Mechanics*, 27:37–48, 2001.

[5] F. Feyel. Multiscale FE^2 elastoviscoplastic analysis of composite structures. *Computational Materials Science*, 16:344–354, 1999.

[6] Charbel Farhat, Jan Mandel, and Francois-Xavier Roux. Optimal convergence properties of the FETI domain decomposition method. *Comput. Methods Appl. Mech. Engrg.*, 115:367–388, 1994.

[7] Charbel Farhat and Jan Mandel. The two-level FETI method for static and dynamic plate problems – Part I: an optimal iterative solver for biharmonic systems. *Computer Methods in Applied Mechanics and Engineering*, 155:129–152, 1998.

[8] Manoj Bhardwaj, David Day, Charbel Farhat, Michel Lesoinne, Kendall Pierson, and Daniel Rixen. Application of the FETI method to ASCI problems – scalability results on one thousand processors and discussion of highly heterogeneous problems. *Int. J. Numer. Meth. Engrg.*, 47:513–535, 2000.

[9] Charbel Farhat, Kendall Pierson, and Michel Lesoinne. The second generation of FETI methods and their application to the parallel solution of large-scale linear and geometrically nonlinear structural analysis problems. *Computer Methods in Applied Mechanics and Engineering*, 184:333–374, 2000.

[10] Axel Klawonn and Olof B. Widlund. FETI and Neumann-Neumann iterative substructuring methods: connections and new results. *Communications on Pure and Applied Mathematics*, LIV:57–90, 2001.

[11] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. A scalable dual-primal domain decomposition method. *Numer. Lin. Alg. Appl.*, 7:687–714, 2000.

[12] Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. FETI-DP: A dual-primal unified FETI method – Part I: A faster alternative to the two-level FETI method. *Int. J. Numer. Meth. Engrg.*, 50:1523–1544, 2001.

[13] Axel Klawonn and Oliver Rheinbach. Highly scalable parallel domain decomposition methods with an application to biomechanics. *ZAMM Z. Angew. Math. Mech.*, 90(1):5–32, 2010.

[14] Axel Klawonn and Olof B. Widlund. Dual-Primal FETI Methods for Linear Elasticity. *Comm. Pure Appl. Math.*, 59(11):1523–1572, 2006.

[15] Axel Klawonn and Oliver Rheinbach. Robust FETI-DP methods for heterogeneous three dimensional elasticity problems. *Comput. Methods Appl. Mech. Engrg.*, 196(8):1400–1414, 2007.

[16] Axel Klawonn and Oliver Rheinbach. Inexact FETI-DP methods. *Internat. J. Numer. Methods Engrg.*, 69(2):284–307, 2007.

[17] Oliver Rheinbach. Parallel iterative substructuring in structural mechanics. *Arch. Comput. Methods Eng.*, 16(4):425–463, 2009.

[18] Axel Klawonn, Martin Lanser, Patrick Radtke, and Oliver Rheinbach. On an adaptive coarse space and on nonlinear domain decomposition. *Domain Decomposition Methods in Science and Engineering XXI*, volume 98 of *Lect. Notes Comput. Sci. Eng.*, pages 71–83. 2014. `http://dd21.inria.fr/pdf/klawonna_plenary_3.pdf`.

[19] Axel Klawonn, Martin Lanser, and Oliver Rheinbach. Nonlinear FETI-DP and BDDC methods. *SIAM J. Sci. Comput.*, 36(2):A737–A765, 2014. `http://www.mathe.tu-freiberg.de/files/personal/253/rheinbach-nonlinear.pdf`.

[20] Axel Klawonn, Martin Lanser, and Oliver Rheinbach. A nonlinear FETI-DP method with an inexact coarse problem. 2014. In Proceedings of the 22nd International Conference on Domain Decomposition Methods. Preprint: `http://www.mathe.tu-freiberg.de/files/personal/253/rheinbach-plenarytalk-dd22.pdf`.

[21] Axel Klawonn, Martin Lanser, Oliver Rheinbach, Holger Stengel, and Gerhard Wellein. Hybrid MPI/OpenMP parallelization in FETI-DP methods. Technical Report 2015-02, Fakultät für Mathematik und Informatik, Technische Universität Bergakademie Freiberg, 2015. Accepted for publication in Springer Lect. Notes Comput. Sci. Eng., `http://tu-freiberg.de/fakult1/forschung/preprints`.

[22] Julien Pebrel, Christian Rey, and Pierre Gosselet. A nonlinear dual-domain decomposition method: Application to structural problems with damage. *Inter. J. Multiscale Comp. Eng.*, 6(3):251–262, 2008.

[23] Felipe Bordeu, Pierre-Alain Boucard, and Pierre Gosselet. Balancing domain decomposition with nonlinear relocalization: Parallel implementation for laminates. In *Proceedings of the First International Conference on Parallel, Distributed and Grid Computing for Engineering*, Civil-Comp Press, 2009.

[24] Xiao-Chuan Cai and David E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM J. Sci. Comput.*, 24(1):183–200 (electronic), 2002.

FEMPAR: Scaling Multi-Level Domain Decomposition up to the full JUQUEEN supercomputer

Santiago Badia, Alberto F. Martín, and Javier Principe

Centre Internacional de Mètodes Numèrics a l'Enginyeria (CIMNE),
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

Description of the Code

FEMPAR [1], developed by the members of the LSSC team at CIMNE, is a parallel hybrid OpenMP/MPI, object-oriented framework for the massively parallel Finite Element (FE) simulation of multiphysics problems governed by PDEs. It provides tools for the numerical simulation of a wide range of different physical phenomena, including compressible/incompressible flows, magnetics, solid mechanics, fluid-structure interaction, or thermal coupling. FEMPAR has been designed to tackle multiphysics, nonlinear, and multiscale problems. For such problems, it makes use of scalable implicit massively parallel solvers that are based on Balancing Domain Decomposition by Constraints (BDDC) preconditioning ideas [2, 3], combined with fully-coupled block LU preconditioning [4].

In particular, within the domain decomposition kernel, FEMPAR provides a novel, fully-distributed, communicator-aware, recursive, and inter-level overlapped implementation of the MultiLevel BDDC (MLBDDC) preconditioner [5]. Figure 1 depicts the global structure of computation and communication underlying this kernel. This code weakly scales up to 458,752 JUQUEEN cores for coercive three-dimensional problems (the Laplacian and Linear Elasticity problems). The largest problem solved with FEMPAR up to now involved 30 billion unknowns. FEMPAR is released under the GNU GPL v3 license, and is more than 200K lines of Fortran95/2003/2008 code long.

As an application example, Figure 2 illustrates the vorticity isosurfaces for the incompressible Taylor-Green vortex problem at $Re = 1600$ at four different time steps, starting with the initial condition at the top-left corner and evolving in time from top to bottom, and left to right. These simulation results were obtained with FEMPAR by means of a segregated velocity/pressure algorithm, that involves a pressure Poisson MLBDDC solver per time step.

Results

Before the workshop, we could already scale the MLBDDC solver up to the full JUQUEEN supercomputer. In particular, a 3-level BDDC preconditioner, supplied either with corner and edges, or corner, edge and face constraints, was successfully applied to the Laplacian and Linear elasticity discrete problems with excellent weak scalability results. These experiments were performed with 16 MPI tasks/node, and 1 OpenMP thread/MPI task, so that we were not actually taking any profit from the hardware threads of the IBM PowerPC A2 cores. Given such limitation, the main goal during the workshop was to explore approaches that enable the exploitation of hardware multi-threading.

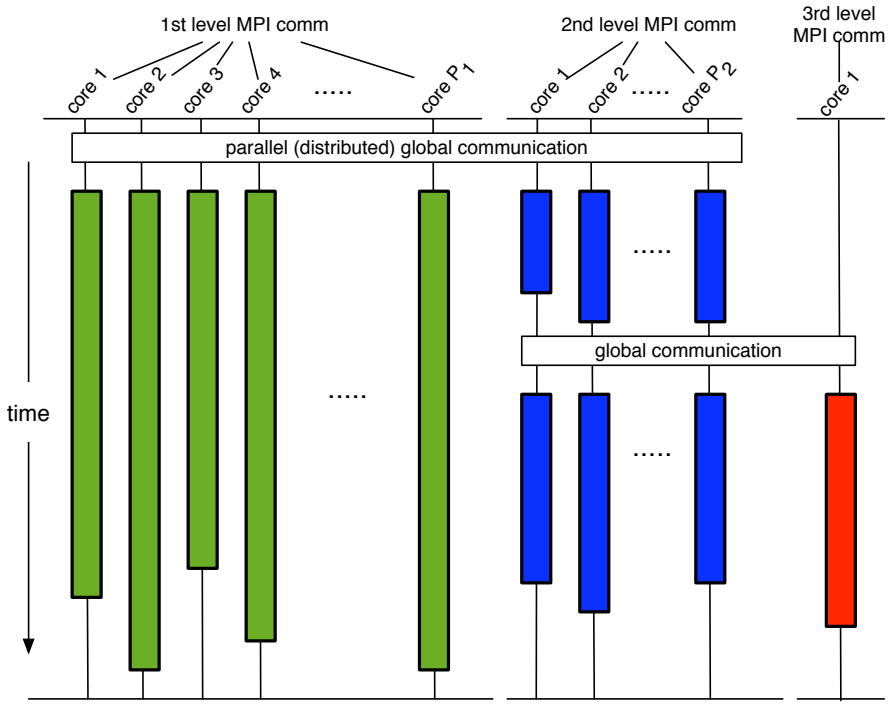

Figure 1: Computation and communication structure of the fully-distributed, communicator-aware, recursive, and inter-level overlapped implementation of the MLBDDC preconditioner.

Although FEMPAR supports hybrid MPI/OpenMP execution, it (currently) only exploits OpenMP for some phases during the computation. In particular, in the solution of *local* Dirichlet/Neumann problems (at each intermediate level), and in the solution of the coarsest-grid problem. For the solution of such problems on JUQUEEN, FEMPAR relies on HSL_MA87 [6], a numerical library for the multi-threaded sparse direct solution of SPD linear systems. Although for "large" load per core these kernels consume the bulk of the computation, there is intrinsically a serial bottleneck for increasing number of threads due to the parts which are not parallelised. On the other hand, arithmetic complexity of sparse direct methods is well known to grow as $O(n^2)$ for 3D problems, with n being the size of the coefficient matrix. These two factors render slower (for same global problem size) those hybrid configurations which use less MPI tasks than physical cores (and more OpenMP threads).

Given such scenario, we have two possible (efficient) approaches for the exploitation of hardware threads. On the one hand, a hybrid MPI/OpenMP approach with 16 MPI tasks/node, and 2/4 threads/MPI task (core). On the other hand, a pure MPI approach with 32 or 64 MPI tasks/node. The first approach, although convenient, could not be explored during the workshop, due to a memory related issue that is still under investigation with the help of JSC staff. In particular, heap and mmap *system memory* consumed by HSL_MA87 with 2/4 OpenMP threads significantly increases with respect to the 1 thread case. Besides, this increase is not reproducible, and may become "large" depending on how the tasks performed by the threads are scheduled/synchronised by the underlying software/hardware stack. This prevents the code from solving problems with a load close to the 1 GB/core limit, precisely those loads for which we expect the largest performance benefit from the exploitation of hardware threads via

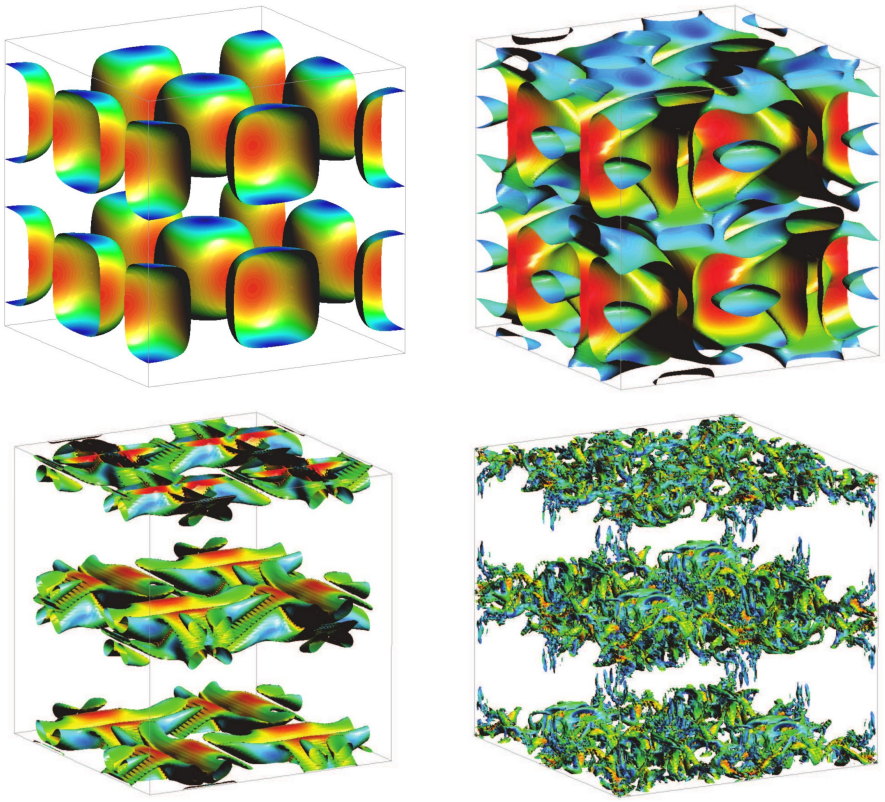

Figure 2: Vorticity isosurfaces for the incompressible Taylor-Green vortex problem at $Re = 1600$.

OpenMP.

In light of these memory issues, we decided to put on hold the hybrid MPI/OpenMP approach, focusing ourselves on the pure MPI approach. Although we also performed experiments with 32 MPI tasks/node, the results with 64 MPI tasks/node confirm a higher profit from the hardware threads in terms of aggregated efficiency. The usage of 64 MPI tasks/node implies a very moderate amount of memory of 256 MB/MPI task, and a 4-fold increase in the coarse-grid problem size to be solved at each level of the hierarchy. In order to cope with a smaller load per core, and larger coarse-grid problems, we decided to add an additional level in the preconditioning hierarchy, and study a 4-level BDDC preconditioner. In particular, Table 1 reports the configuration of the experiment that we performed with 64 MPI tasks/node, with the number of MPI tasks (subdomains) at each level, and the loads per MPI task (subdomain) tested. We applied this algorithm to the Laplacian problem discretised with Q1 FEs, and studied the weak scalability of the code on JUQUEEN with the BDDC space supplied with either corner and edge (ce), or corner, edge, and face (cef) constraints, for 3 different loads per core on the first level. To keep the presentation simpler, we will focus on the results that we obtained with the 4-level MLBDDC(cef) solver, with the largest load of 25^3 FEs/core.

A first bottleneck that we had to face was related to the initialisation stage of the code. In such stage, the MPI tasks in the global communicator are split into two disjoint subcommunicators via a call to `MPI_Comm_split`. One of these two includes the MPI tasks in the first level of the hierarchy, while the other one those which belong to the rest of levels (2nd, 3rd, and 4th in Table 1). With default settings, `MPI_Comm_split` was scaling as $O(P^2)$, with P being the number of MPI tasks in the global communicator. The workaround recommended by JSC staff was to activate the `PAMID_COLLECTIVES_MEMORY_OPTIMIZED` environment variable. This set-up

Table 1: Configuration of the 4-level BDDC preconditioner for the FEMPAR experiments with 64 MPI tasks/node performed during the workshop.

Level	# MPI tasks				FEs/core
1st	592.7K	884.7K	1.26M	1.73M	$10^3/20^3/25^3$
2nd	9.26K	13.8K	19.7K	64K	4^3
3rd	343	512	729	1K	3^3
4th	1	1	1	1	n/a
	615K	918K	1280K	1795K	

switches to a different algorithm within the underlying message-passing stack. As shown in Table 2, it also has a tremendous *positive* impact on performance/scalability, at the price of a moderate increase in memory consumption.

Table 2: Performance/scalability of `MPI_Comm_split` and average memory/1st level MPI task consumed by the 4-level BDDC(cef) solver, with the largest load of 25^3 FEs/core, once the preconditioner is set-up.

		Default settings		`PAMID_..._OPTIMIZED`	
bg_size	P	Time (sec.)	Mem. (MB)	Time (sec.)	Mem. (MB)
9609	615K	n/a	115.3	2.46	127.8
14344	918K	n/a	122.5	3.92	143.3
20002	1280K	365	131.7	5.74	163.2
28047	1795K	862	n/a	8.09	187.8

Once the bottleneck related to `MPI_Comm_split` was overcome, we proceeded with the actual weak scalability test. Table 3 reports the number of PCG solver iterations and total computation time in seconds for the 4-level BDDC(cef) solver, when applied to the discrete Laplacian problem using 64 MPI tasks/node, and the largest load of 25^3 FEs/core. Total computation time includes both preconditioner set-up and the PCG phase. These results confirm remarkable scalability for the approach that we pursue for the extreme scale implementation of the MLBDDC preconditioner. In particular, with a 4-level BDDC(cef) preconditioner, we were already able to strike a balance such that computation/communication related to coarser-grid levels in the hierarchy (i.e., 2nd, 3rd and 4th in Table 1) are completely absorbed (i.e., masked) by the finest-grid level duties (i.e., 1st level in Table 1) due to the effect of inter-level overlapping (see Figure 1). Besides, on smaller scale test cases, we compared the computation times of the codes using 16 and 64 MPI tasks/node (with the same number of MPI tasks/level in both cases), confirming an approximately 50% saving in aggregated efficiency by the exploitation of hardware multi-threading (i.e., the computation time with 64 MPI tasks/node was approximately twice as much as the one with 16 MPI tasks/node).

Finally, we would like to remark that we expect that the achievements resulting from our participation in the workshop will have a high impact on the scientific computing community in general, and in the development of fast parallel solvers tailored for FE analysis in particular [5].

Table 3: Weak scalability for the FEMPAR 4-level BDDC(cef) solver with 64 MPI tasks/node and the largest load of 25^3 FEs/core.

bg_size	P	#PCG iterations	Total time (sec.)
9609	615K	25	22.1
14344	918K	26	22.6
20002	1280K	27	22.9
28047	1795K	27	23.0

Acknowledgments

This work has been funded by the European Research Council under the FP7 Programme Ideas through the Starting Grant No. 258443 – COMFUS: Computational Methods for Fusion Technology. A. F. Martín was also partially funded by the Generalitat de Catalunya under the program "Ajuts per a la incorporació, amb caràcter temporal, de personal investigador júnior a les universitats públiques del sistema universitari català PDJ 2013." We acknowledge GCS for awarding us access to resource JUQUEEN. We gratefully acknowledge JSC's staff in general, and Dirk Brömmel in particular, for their support in porting/debugging FEMPAR and its dependencies to/on JUQUEEN.

References

[1] FEMPAR web page. `https://web.cimne.upc.edu/groups/comfus/fempar.html`

[2] S. Badia, A. F. Martín and J. Principe. Implementation and scalability analysis of balancing domain decomposition methods. *Archives of Computational Methods in Engineering*. Vol. 20(3), pp. 239–262, 2013.

[3] S. Badia, A. F. Martín and J. Principe. A highly scalable parallel implementation of balancing domain decomposition by constraints. *SIAM Journal on Scientific Computing*. Vol. 36(2), pp. C190–C218, 2014.

[4] S. Badia, A. F. Martín and R. Planas. Block recursive LU preconditioners for the thermally coupled incompressible inductionless MHD problem. *Journal of Computational Physics*, Vol. 274, pp. 562–591, 2014.

[5] S. Badia, A. F. Martín and J. Principe. Multilevel balancing domain decomposition at extreme scales. In preparation, 2015.

[6] J. Hogg, J. Reid and J. Scott. Design of a Multicore Sparse Cholesky Factorization Using DAGs. *SIAM Journal on Scientific Computing*. Vol. 32(6), pp. 3627–3649, 2010.

ICON with HD(CP)^2 setup 120 m

Catrin Meyer[1] and Thomas Jahns[2]

[1]Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH
[2]Deutsches Klimarechenzentrum GmbH

Description of the Code

The Max Planck Institute for Meteorology (MPI-M) and the German Weather Service (DWD) developed different icosahedral non-hydrostatic (ICON) dynamical cores of high resolution with the functionality of local zooming [1–4][a]. These new model systems contain a consistent numerical approximation to the adiabatic dynamics of the atmosphere and the tracer transport processes, so that a better basis can be provided for the modelling of chemistry related processes. The possibility of local zooming provides sufficient accuracy in regions of, e.g., complex topography, and at the same time keep the overall computational expense affordable in long-term climate simulations.

The ICON dynamical cores solve the fully compressible non-hydrostatic equations of motion for simulations at very high horizontal resolution. The discretisation of the continuity and tracer transport equations is consistent in a sense that mass of air and its constituents are conserved, which is a fundamental requirement for atmospheric chemistry. Furthermore, the vector invariant form of the momentum equation are used, and thus, vorticity dynamics are emphasised. The new dynamical core solves the system of primitive equations in grid point space on the icosahedral grid, which facilitates the quasi-isotropic horizontal resolution on the sphere and the restriction to regional domains. The discretisation method is defined on a special case of Delaunay triangulation on the sphere, i.e., the icosahedral geodesic grid.

A cloud resolving, or large-eddy simulation (LES) version [6] of the ICON core with ultra-high horizontal grid spacing of approximately 100 m is developed within the BMBF initiative "High definition clouds and precipitation for advancing climate prediction"[b], or HD(CP)^2 for short [5].

The code is mainly written in Fortran, with some library code (model time management, I/O) written in C. It is parallelised with MPI and OpenMP, both of which can optionally be switched off. It requires netCDF as an external library. Large output files can be produced for detailed inspection of model state, but output was switched off for the workshop because very short runs simulating only seconds of model time like those used in the workshop generate no significant insight.

[a]See `http://www.mpimet.mpg.de/en/science/models/icon.html` and `http://www.dwd.de/bvbw/appmanager/bvbw/dwdwwwDesktop?_nfpb=true&_pageLabel=dwdwww_result_page&gsbSearchDocId=749190` for more details.

[b]See the project website for more detailed information: `http://hdcp2.zmaw.de/Mission.2261.0.html`

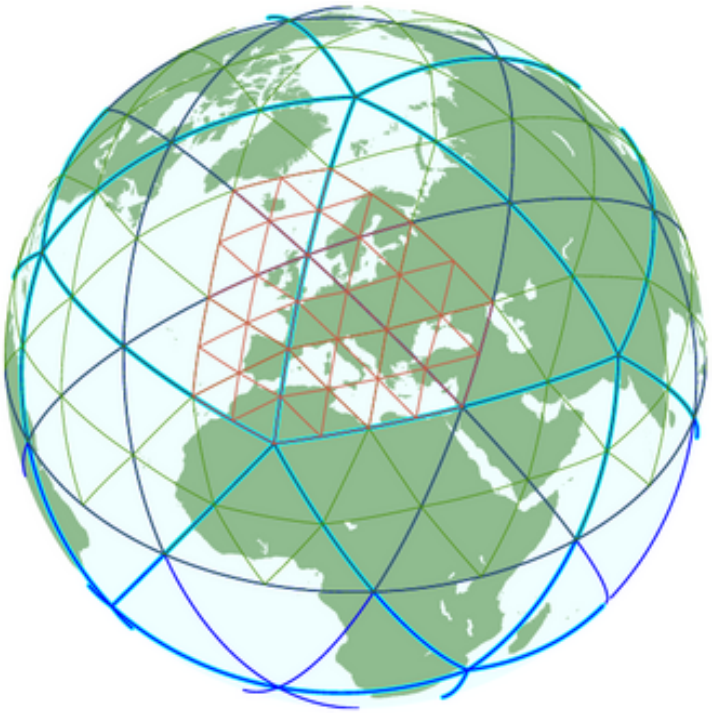

Figure 1: The icosahedral grid used by ICON with a finer refinement for Europe.

Results

During the workshop we performed strong scaling tests for the HD(CP)^2 setup with a horizontal resolution of 120 m. Our test case computed 20 s simulation time. The performance analysis is based on the ICON internal timer for "timeloop total" (`timeloop`). This instrumented region of code includes the recurring physics and the dynamic core of the model but no initialisation and output. During our tests in the workshop the I/O was switched off.

Table 1 and Figure 2 summarise the performance results for 1 MPI rank per node and 64 OpenMP threads per rank. We were able to show that the LES physics and the dynamical core scales well up to the full JUQUEEN machine. We reach a speed-up of 12.25 and still an efficiency of 44 % on 28 racks compared to 1 rack.

Table 1: Scaling tests for the ICON 120 m HD(CP)^2 setup, showing number of compute nodes (bg_size), ranks per node (rpn), total number of processes (MPI ranks), number of OpenMP threads per rank (tpr), total number of threads (Threads), timing from `timeloop`, averaged memory consumption per node (Memory)

bg_size	rpn	MPI ranks	tpr	Threads	Time (s)	Memory [GB]
1 024	1	1 024	64	65 536	154.39	13.19
2 048	1	2 048	64	131 072	83.34	8.25
4 096	1	4 096	64	262 144	45.96	5.76
8 192	1	8 192	64	524 288	27.08	4.51
16 384	1	16 384	64	1 048 576	17.03	3.88
28 672	1	28 672	64	1 835 008	12.60	3.71

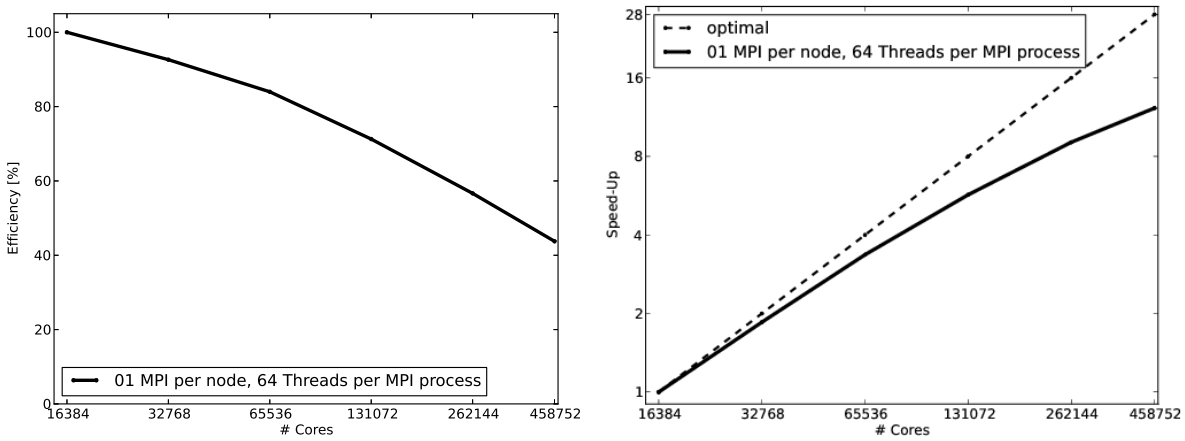

Figure 2: Scaling for the ICON 120 m HD(CP)^2 setup with timing from `timeloop`

Unfortunally for more MPI ranks the internal timer of ICON needed to be switched off, because of impeded scalability The timers use `MPI_MINLOC` and `MPI_MAXLOC` to find ranks with minimal/maximal run-time in portions of the code, but both reductions are unaccelerated on Blue Gene/Q. We did a quick replacement with `MPI_Wtime` and Fortran `WRITE` to the console. With this "new" timer measurement we did a comparison between different MPI/OpenMP combinations (Table 2, Figure 3). Because we placed the "new" timer measurement at slightly different positions in the code, it does not produce the values of the internal timers. But a comparative analysis is still possible: We get a better performance if we use more MPI tasks. For measurments at higher number of MPI ranks we recognized another problem during the initialization process. A pair of reductions (`MPI_Allreduce`) during the domain decomposition phase of model initialization also takes significantly longer on Blue Gene/Q than on other machines, because hardware-accelerated reductions only work for specific, previously undocumented circumstances. Specifically ICON

1. initializes MPI with `MPI_THREAD_MULTIPLE`,
2. passes reduction arguments with `MPI_IN_PLACE`,
3. uses user-defined reductions (`MPI_Op_create`) and
4. employs communicators derived from `MPI_COMM_WORLD` for reductions.

All four aspects need to be changed in code to exploit acceleration. The combination with 4 MPI tasks and 16 OpenMP threads per rank does not fit in a wallclock of 45 minutes on the full machine because of this issue.

In our test case the initialisation phase needs most of the runtime, because of the short simulation time. For a real simulation case the model will run for several days or months and so the initialisation takes place only once at the beginning and does not influence the total runtime as much as in our test case. The performance results of the test case are therefore comparable with a real simulation case. The main parts of the model, i.e., the LES physics and dynamic core, can use the 28 racks of JUQUEEN efficiently.

Conclusions

The above measurements are due to the significant progress made during the last two years in terms of efficient memory use in HD(CP)^2: previous tests could not launch at 120 m resolution on Blue Gene/Q due to memory constraints.

Table 2: Scaling tests for the ICON 120 m HD(CP)^2 setup, showing number of compute nodes (bg_size), ranks per node (rpn), total number of processes (MPI ranks), number of OpenMP threads per rank (tpr), total number of threads (Threads), timing from `MPI_Wtime`, averaged memory consumption per node (Memory)

bg_size	rpn	MPI ranks	tpr	Threads	Time (s)	Memory [GB]
2 048	1	2 048	64	131 072	145.34	8.25
4 096	1	4 096	64	262 144	78.49	5.76
8 192	1	8 192	64	524 288	45.34	4.51
16 384	1	16 384	64	1 048 576	27.37	3.88
28 672	1	28 672	64	1 835 008	21.78	3.71
2 048	2	4 096	32	131 072	113.30	8.38
4 096	2	8 192	32	262 144	62.39	5.88
8 192	2	16 384	32	524 288	36.04	4.61
16 384	2	32 768	32	1 048 576	21.67	4.27
28 672	2	57 344	32	1 835 008	15.86	4.51
2 048	4	8 192	16	131 072	114.73	8.62
4 096	4	16 384	16	262 144	63.51	6.08
8 192	4	32 768	16	524 288	36.33	5.40
16 384	4	65 536	16	1 048 576	21.61	6.12

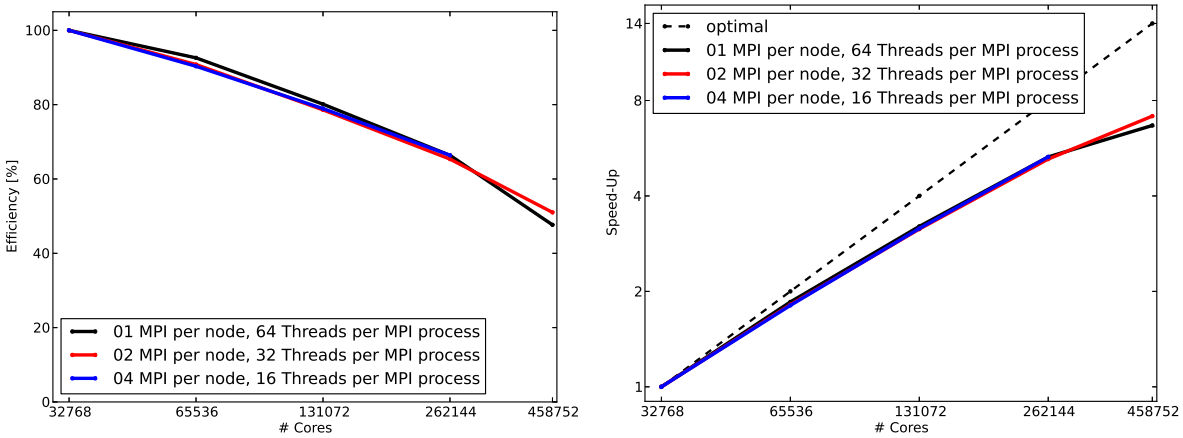

Figure 3: Scaling for the ICON 120 m HD(CP)^2 setup with timing from `MPI_Wtime`

It also shows in the scalability issues associated with initialisation, output and internal time measurements that there is still major efforts in each of these parts of the implementation required to make efficient use of Blue Gene/Q hardware for full-featured production runs. It should also be noted that because the runs can only begin at already quite high numbers of tasks to accumulate the required total memory minimum, some inefficiency due to load-imbalance cannot be deduced from the scaling plots for this model resolution since it is included in the already MPI-parallelized baseline simulation.

References

[1] A. Gassmann and H.-J. Herzog, *Towards a consistent numerical compressible non-hydrostatic model using generalised Hamiltonian tools*, Q. J. R.Meteorol.Soc., **(**134) (2008) 1597–1613.

[2] H. Wan, *Developing and testing a hydrostatic atmospheric dynamical core on triangular grids*, Ph.D. thesis, International Max Planck Research School on Earth System Modelling, (2009).

[3] H. Wan, M. A. Giorgetta, G. Zängl, M. Restelli, D. Majewski, L. Bonaventura, K. Fröhlich, D. Reinert, P. Rípodas, L. Kornblueh and J. Förstner, *The ICON-1.2 hydrostatic atmospheric dynamical core on triangular grids – Part 1: Formulation and performance of the baseline version*, Geoscientific Model Development, **6(3)** (2013) 735–763 [DOI: 10.5194/gmd-6-735-2013]

[4] G. Zängl, D. Reinert, P. Ripodas, and M. Baldauf, *The ICON (ICOsahedral Nonhydrostatic) modelling framework of DWD and MPI-M: Description of the nonhydrostatic dynamical core*, Q. J. R.Meteorol.Soc., (2014) [DOI: 10.1002/qj.2378].

[5] B. Stevens, J. Biercamp, U. Burkhardt, S. Crewell, S. Jones, A. Macke, A. Seifert and C. Simmer, *HD(CP)^2: High definition clouds and precipitation for advancing climate prediction*, Tech. report (2011)

[6] A. Dipankar, R. Heinze, C. Moseley and B. Stevens, *A large eddy simulation version of ICON (ICOsahedral Nonhydrostatic): Model description and validation*, J. of Advances in Modeling Earth Systems (2014) (submitted)

MPAS-A Extreme Scaling Experiment

Dominikus Heinzeller[1] and Michael Duda[2]

[1]Karlsruhe Inst. of Technology, Inst. of Meteorology and Climate Research
[2]National Center for Atmospheric Research, Earth System Laboratory

Description of the Code

The Model for Prediction Across Scales (MPAS) [1] is composed of a set of several earth-system component models built within a shared software framework; at present, MPAS includes an atmospheric model, MPAS-A [2], an ocean model, MPAS-O [3], and a land-ice model, MPAS-LI. The key feature common to all MPAS models is their use of unstructured centroidal Voronoi tesselations (CVTs) as their horizontal meshes. CVT meshes allow global simulations to be performed on a variable-resolution mesh with smooth transitions between regions of low and high resolution. In this scaling experiment, we focus on the atmospheric model, MPAS-A. Based on the Voronoi mesh, the model uses a C-grid staggering for the state variables, i.e., wind components are modelled at the faces of every cell, and the prognosed component of the wind is orthogonal to the cell face. The governing equations can then be cast in a way such that energy, momentum and water vapour content are conserved (Figure 1).

The MPAS code is mainly written in Fortran, with some smaller parts written in C. Parallel I/O is realised using the Parallel I/O library PIO, for which the user can choose serial/parallel NetCDF3 or serial/parallel NetCDF4. Since MPAS-A is a global model, only one initial conditions file (`init.nc`) is read during the initialisation of the model. The frequency of writing output data (diagnostic data, restart data) is entirely up to the user. Parallelisation of the latest released version of the model code employs only MPI for inter-process communication, but work is underway to enable hybrid parallelisation with MPI and OpenMP.

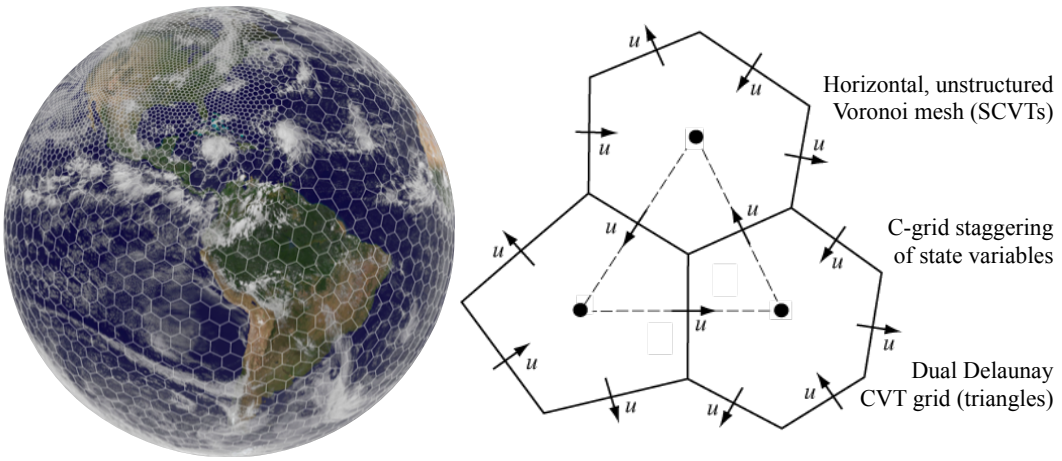

Figure 1: (left) global variable resolution mesh, (right) C-grid staggering

Results

We used a regular, global 3 km mesh with more than 65 million horizontal grid cells and 41 vertical levels to conduct the extreme scaling experiment. Up to now, only a few runs using this mesh have been conducted on the National Center for Atmospheric Research (NCAR) Yellowstone supercomputer [4] using 16 384 MPI tasks. This resolution was chosen to (a) investigate the scalability of MPAS-A on the full JUQUEEN system, and (b) demonstrate the feasibility to conduct global, convection-resolving simulations on massively parallel systems.

In a first step, we analysed the code with Scalasca/Score-P [5]. An example profile from MPAS-A in Figure 3 shows a significant imbalance of MPI Collective File I/O time. Using several smaller model configurations (120 km regular mesh, 120–15 km variable-resolution mesh, 90–12 km variable-resolution mesh), we could show that the parallel efficiency of MPAS-A can be correlated to the number of grid cells owned by each MPI task. For less than approx. 160 grid cells per task, the MPI communication overhead becomes significant and the parallel efficiency drops below 70%. This is in agreement with prior findings on NCAR Yellowstone. For the 3-km mesh with more than 65 million grid points, we therefore expected a breakdown of the parallel efficiency around 400 000 MPI tasks.

Initial runs with a resolution of 3 km revealed previously unknown problems of MPAS-A on JUQUEEN. In the model initialisation, a bootstrapping step is required to set up the grid and instruct individual tasks with whom to share information about neighbouring grid cells ("halo/ghost cells"). This initial bootstrapping takes between 18 and 29 minutes, depending on the number of MPI tasks (Table 1). A second problem comes from reading the initial conditions file `init.nc`, which for the 3 km mesh is 1.2 TB in size. Performance improvements for this step were achieved by introducing two runtime environment variables that were presented during the workshop (`BGLOCKLESSMPIO_F_TYPE`, `ROMIO_HINTS`), and by optimising the number of I/O tasks as a function of total tasks (128 I/O tasks per rack, i. e., per 16 384 tasks). While the parallel reading of the initial conditions file improves slightly with the number of tasks (I/O tasks located in different racks), the bootstrapping takes longer for larger numbers of tasks. Hence, the overall model initialisation is to some degree independent of the number of tasks and takes approximately 45 minutes for the 3 km mesh. One notable exception here is the run on 8 racks, for which the initial I/O is only 50% of that of the other runs. The exact reasons for this behaviour need to be investigated, but it is possible that this combination of file size and I/O tasks are a sweet spot on JUQUEEN.

The substantial memory requirements for the 3 km mesh did not allow to run the model with only 1 or 2 racks. The baseline for our scaling experiment is therefore the run on 4 racks (65 536 MPI tasks, 512 I/O tasks). Performance improvements for the model integration were achieved by using appropriate compiler flags for this system (`-O3 -qstrict -qarch=qp -qtune=qp`). Contrary to the model initialisation, the time integration step scales very well up to around 400 000 tasks, with a parallel efficiency of 87% for 24 racks (393 216 tasks) compared to the baseline with 4 racks. The test run on the full system (28 racks, 458 752 tasks) showed a poor performance compared to the run on 24 racks, since the MPI overhead becomes significant for only 142 owned cells per task. All test runs were conducted with a 18 s model integration time step for 1 hr model time. However, we found that in order to keep the model stable when starting off from a 48 km re-analysis dataset (CFSR) as initial conditions, a more conservative time step is required. This is because MPAS currently lacks a dynamical initialisation system (e. g., digital filters, adaptive timestepping). We repeated runs for 4, 8, and 16 racks with a 12 s time step without detecting any instabilities. The measured real time for the three 12 s runs was very close to 1.5 times the real time for the corresponding 18 s runs.

Table 1 and Figure 2 summarise the required times of the individual steps of the 3 km runs. All runs were conducted with MPI parallelisation only and with 16 MPI tasks per node. Due to

Table 1: MPAS-A 3 km global simulation experiment (strong scaling)

bg_size	Threads (MPI only)	Bootstrapping (s)	Initial read (s)
1024	16384	1240	n/a
2048	32768	1260	n/a
4096	65536	1260	1260
8192	131072	1370	590
16384	262144	1560	1020
24576	393216	1680	1080
28672	458752	1740	1140

bg_size	Time integration for 1-hour model time (s)	Parallel efficiency integration only	Integration in 24h walltime*
1024	n/a	n/a	n/a
2048	n/a	n/a	n/a
4096	1760	100.0%	29h
8192	960	91.2%	53h
16384	490	90.1%	104h
24576	335	87.7%	152h
28672	360	69.5%	141h

*incl. model initialisation (bootstrapping, reading), no writing, 12 s time step

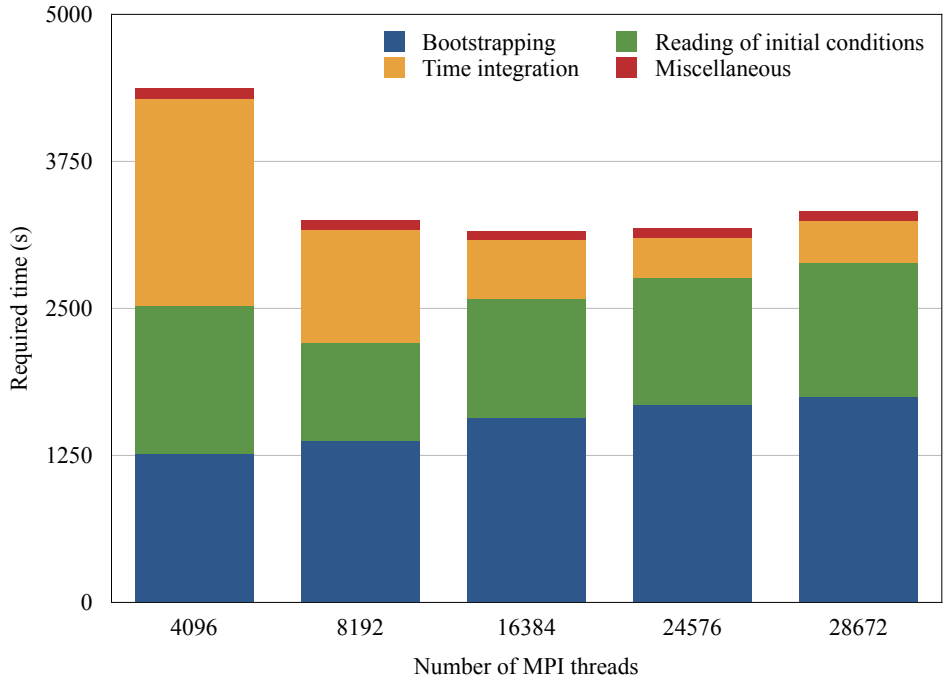

Figure 2: Required times for individual steps of the test runs (18 s time step)

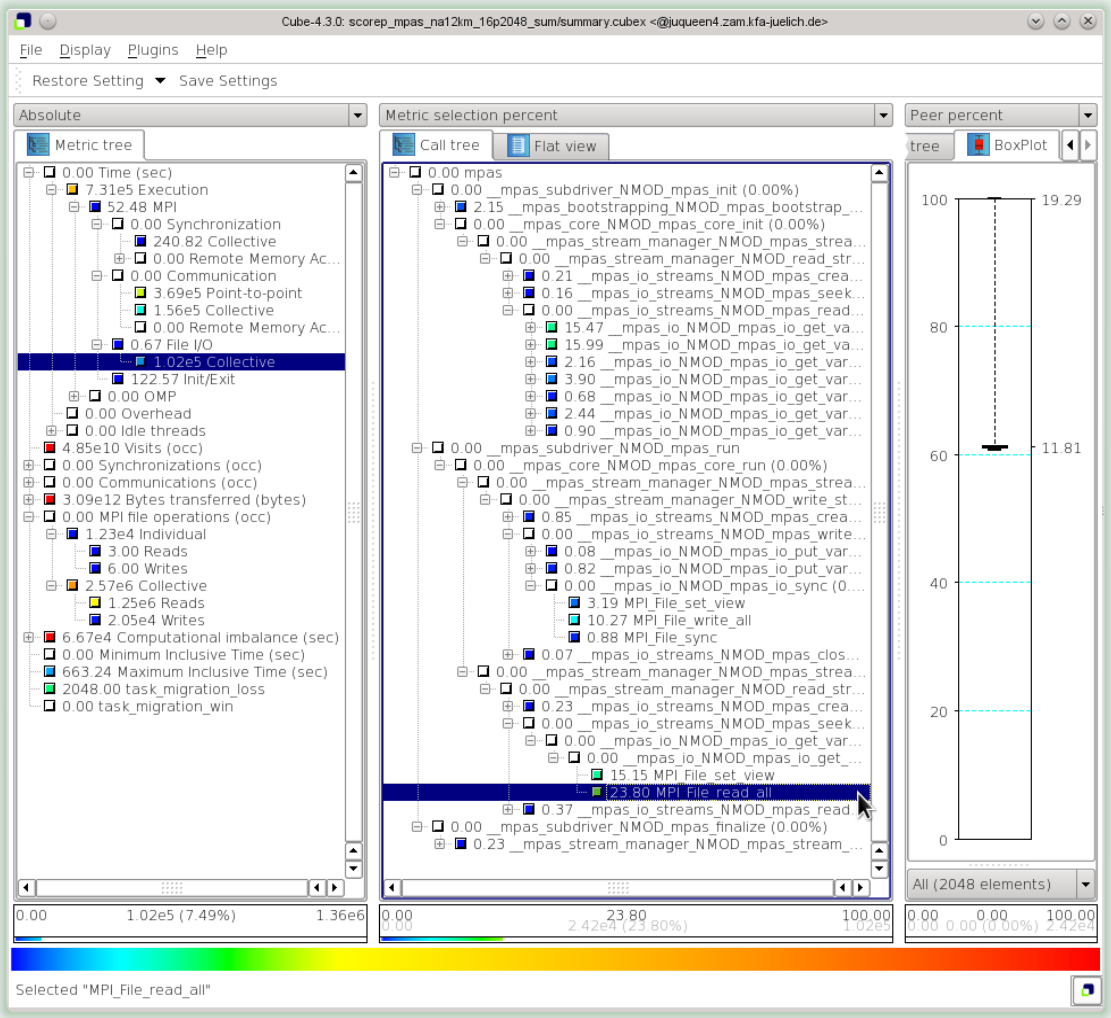

Figure 3: Scalasca/Score-P profile of 2048-process execution of MPAS-A showing time in `MPI_File_read_all`. Rank 2032 takes 19.3s, which is some 7.5s longer than the others which are blocked in the preceding `MPI_File_set_view`.

time constraints, we could only conduct runs without writing output to disk. The last column in Table 1 is therefore an upper limit on how many hours the 3-km model can be advanced within 24h walltime, and is calculated as follows.

A 12 s model integration time step is assumed, and the real time required is scaled from the 18 s runs by 1.5 for 24 and 28 racks (no 12 s runs were conducted). Further, for a typical production run, diagnostic output files of 15 GB in size are written every 3 h model time, while comprehensive output files of approx. 250 GB in size are written every 24 h model time. A restart file of 2.1 TB is written at the end of the model run. For the 16 and 24 rack runs, we take a conservative estimate that roughly two hours of the 24 h walltime will be used up by writing these files to disk. Under this assumption and the requirement that each job advances the model by full days only, we obtain typical job sizes for production runs of 4 days on 16 racks, and 6 days on 24 racks.

Conclusions

During the workshop we could demonstrate that it is possible conduct global, convection-resolving atmospheric simulations on present massively-parallel systems. We identified bottlenecks and pitfalls in the MPAS-A code, such as the initial bootstrapping and the need for optimising the file I/O operations. The lectures and the hands-on sessions of the Porting and Tuning Workshop provided us with several ideas on how to improve the reading and writing on JUQUEEN and presumably also on other massively parallel systems. In the future, we would like to conduct further experiments with output to disk enabled to verify the assumptions made above.

Acknowledgments

We would like to acknowledge the extensive and valuable support of Dirk Brömmel, Wolfgang Frings, Markus Geimer, Klaus Görgen, Sabine Grießbach, Lars Hoffmann, Catrin Meyer, Michael Rambadt, Michael Stephan, Brian Wylie. We also thank and PRACE for funding the earlier PATC workshop.

References

[1] `http://mpas-dev.github.io`

[2] W.C. Skamarock, J.B. Klemp, M.G. Duda, L.D. Fowler, S.-H. Park, T.D. Ringler (2012): A Multi-scale Nonhydrostatic Atmospheric Model Using Centroidal Voronoi Tesselations and C-Grid Staggering. Monthly Weather Review, **140**, 3090–3105.

[3] T.D. Ringler, M. Petersen, R.L. Higdon, D. Jacobsen, P.W. Jones, M. Maltrud (2013): A Multi-Resolution Approach to Global Ocean Modeling. Ocean Modelling, **69**, 211-232.

[4] NCAR Yellowstone. `https://www2.cisl.ucar.edu/resources/yellowstone`

[5] Scalasca scalable performance analysis toolset. `http://www.scalasca.org/`

Direct Numerical Simulations of Fluid Turbulence at Extreme Scale with psOpen

Jens Henrik Goebbert[1] and Michael Gauding[2]

[1]Jülich Aachen Research Alliance, Germany
[2]Chair of Numerical Fluid Dynamics, TU Freiberg, Germany

Description of the Code

The hybrid OpenMP/MPI code psOpen has been developed at the Institute for Combustion Technology, RWTH Aachen University, to study incompressible fluid turbulence by means of direct numerical simulations. Direct numerical simulation (DNS) solves the Navier-Stokes equations for all scales down to the smallest length scale present in turbulent flows and provides a complete description of the flow, where the three-dimensional (3D) flow fields are known as a function of space and time. Because of growing computational capabilities DNS of turbulent flows has become an indispensable tool. Particularly for high Reynolds numbers DNS is computationally very expensive as the number of required grid points N^3 increases tremendously with Reynolds number, i.e.

$$N^3 \propto \mathrm{Re}_\lambda^{9/2}, \tag{1}$$

where Re_λ denotes the Taylor based Reynolds number. DNS of high Reynolds number turbulence are conducted nowadays on up to 12288^3 grid points and reach Taylor based Reynolds numbers of more than 2000. Figure 1 shows the field of a passive scalar and its dissipation rate in a turbulent flow obtained from DNS with $\mathrm{Re}_\lambda = 530$. Very fine fronts arise that have to be resolved on the numerical grid.

For efficiency and accuracy psOpen employs a pseudo-spectral method, where a Fourier transform of the governing equations is solved in spectral space. Here, derivatives in the transport equation turn into multiplications by the wavenumber. All linear terms can be treated in this way. However, the Fourier transformation of the non-linear term turns into a convolution in Fourier space. This operation is computationally very expensive and requires $\mathcal{O}(N^{2\cdot 3})$ operations. Therefore, instead of directly evaluating the convolution operation, the multiplication of the non-linear term is computed in real space. This approach requires only $\mathcal{O}(N^3 \log N^3)$ operations and is called a pseudo-spectral method since only differentiation is performed in Fourier space. A pseudo-spectral method requires frequently transformations between real and spectral space which is particularly challenging for massively-parallel setups.

Implementing the communication for block decomposition in an efficient manner is a very demanding task. For massively-parallel computations on distributed systems a spatial decomposition of the computational domain is mandatory. The frequently invoked Fourier transformation is a non-local operation and requires access to all data along a global grid line for each computational process. psOpen was improved by a new inhouse-developed 3D FFT library optimised to the special needs of pseudo-spectral DNS. Most CPU time is consumed in this

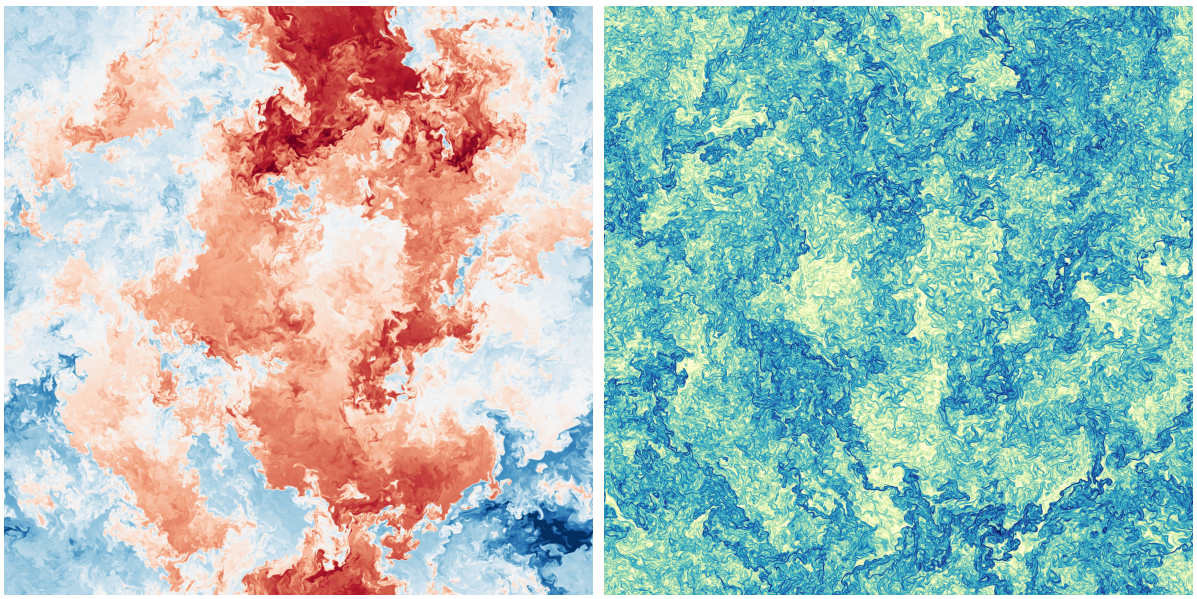

Figure 1: DNS of a passive scalar advected by a turbulent velocity field with 4096^3 grid points and $\mathrm{Re}_\lambda = 530$. Slice of scalar field (left) and scalar dissipation (right).

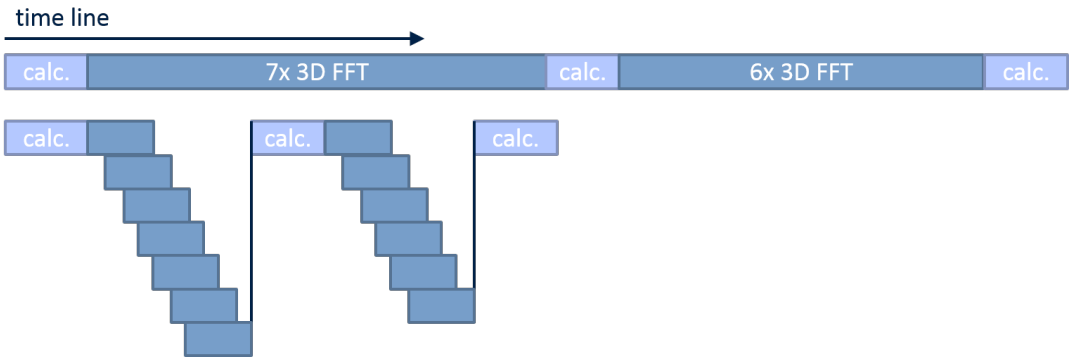

Figure 2: Illustration of blocking communication (top) and non-blocking communication (bottom). For the latter, the processing of several FFTs is overlapped.

library which frequently invokes the 3D-FFT algorithm. It uses the Engineering and Scientific Subroutine Library (ESSL), which is specifically designed for the IBM Blue Gene/Q processor and provides a shared memory FFT function set for best parallelisation on each node. For each iteration the flow field solver calls nine 3D-FFTs and the solver for passive scalar additionally another four 3D-FFTs. Without the improved 3d-FFT library this results in 80% of the compute time.

For an optimal use of the hardware resources three techniques have been combined which reduces the communication time by a factor of more than two.

1. The number of operations and the size of data to be transposed while computing a 3D-FFT has been reduced by integrating the de-aliasing cut-off filter into the 3D-FFT. In total the number of 1D-FFTs is reduced by 29.6% when compared to the non-filtered setup if the filter cuts one-third of the frequencies in each dimension. At the same time the data to be sent is reduced by 33.3% for the first global transpose and by 55.5% for the second transpose.

2. A MPI-parallelised 3D-FFT is based on time-consuming collective communications. The new 3D-FFT library allows to overlap these communications with the computation of multiple FFTs at the same time. This reduces each 3D-FFT by another 15-25%.

3. All 3D-FFTs needed to compute one iteration of the velocity and passive scalar are blocked in two single calls of the non-blocking 3D-FFT library 'nbcFFT', which therefore can overlap one instance of six and another of seven 3D-FFTs at once. This increases the benefit from an overlap communication and computation of 3D-FFTs and reduces the required extra time for a passive scalar by almost 50%. A schematic of the non-blocking communication approach is shown in Figure 2.

Results

The first proof of scalability on IBM Blue Gene architecture was made with the "Jülich Blue Gene/P Porting, Tuning, and Scaling Workshop 2008" and the code has been further improved over the years since then. In 2012 it was ported to JUQUEEN and for the "First JUQUEEN Porting and Tuning Workshop 2013" and "Second JUQUEEN Porting and Tuning Workshop 2014" had switched from the three-dimensional FFT library P3DFFT to the inhouse-developed 3D-FFT library nbcFFT.

The scaling performance of psOpen with three different grid sizes, namely 4096^3, 6144^3 and 8192^3, has been studied in the latest workshop. The chosen grid sizes are of high relevance for production runs. Production runs with up to 6144^3 grid points have been already conducted and those with 8192^3 grid points are planned in the near future. It is customary to compute DNS of homogeneous isotropic turbulent flows in a periodic box in single precision arithmetic. Therefore, all data presented in this report are based on single-precision computations. Compared to double-precision this reduces the communication amount by a factor of two, while keeping constant the computational cost on a Blue Gene/Q architecture.

Figure 3 shows the speedup of psOpen for configurations between 2048^3 and 8192^3 grid points. Full machine runs were performed for 6144^3 and 8192^3 grid points. The memory requirement becomes very demanding for these grid sizes and determines the number of compute nodes that are necessary. psOpen exhibits an almost linear speedup up to 16384 compute nodes for all investigated grid sizes. Runs with 24576 compute nodes exhibit a satisfying efficiency, considering the underlying mathematical operation of solving the Poisson equation which enforces global communication. For the full-machine run with 28672 compute nodes we observe a further decline in efficiency, which might result from the large (size 7) B-dimension of the five-dimensional torus network of JUQUEEN. For reference Table 1 summarises the investigated runs and shows the time per iteration.

Table 1: Strong scaling tests of psOpen for various grid sizes.

					time per iteration in seconds			
bg_size	rpn	ranks	tpp	threads	2048^3	4096^3	6144^3	8192^3
2048	32	65 536	2	131 072	0.6295	3.0785	-	-
4096	32	131 072	2	262 144	0.3281	1.8815	-	-
8192	32	262 144	2	524 288	0.1805	0.9606	2.3276	-
16384	32	524 288	2	1 835 008	-	0.4950	1.2132	-
24576	32	786 432	2	1 572 864	-	-	0.9530	3.7811
28672	32	917 504	2	1 835 008	-	-	0.9278	3.6801

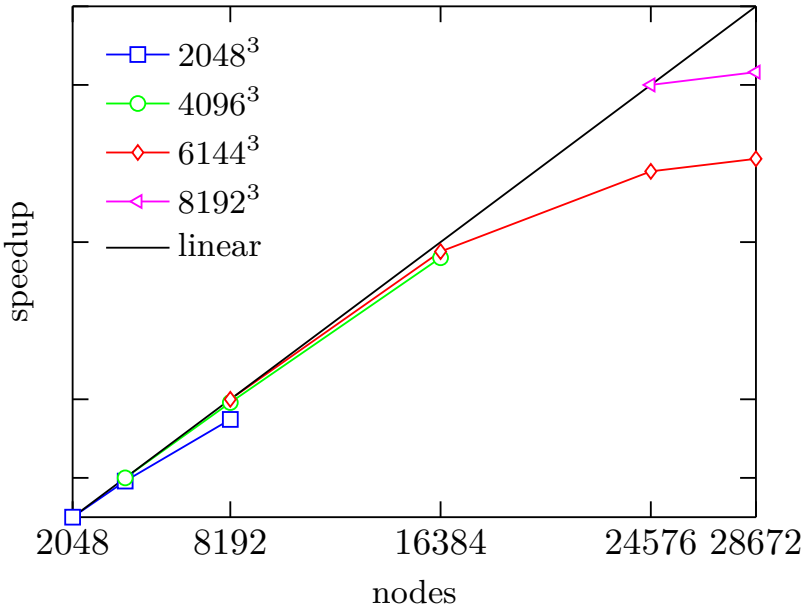

Figure 3: Strong scaling of psOpen for four grid sizes between 2048^3 and 8192^3 grid points. Linear scaling is shown for reference. psOpen exhibits an almost linear speedup for up to 16384 compute nodes.

Currently 32 MPI processes each with two OpenMP tasks are running on each compute node. As the IBM XLF compiler on JUQUEEN does not support nested OpenMP the number of MPI processes per compute node is currently rather large. Using GCC compilers and/or introducing pthreads over OpenMP might reduce this overhead in future.

File I/O is integrated into psOpen by means of parallel HDF5. The I/O performance depends on the number of allocated compute nodes. With 8192 compute nodes up 30.2 GB/s can be reached. For the scaling analysis psOpen was initialised with synthetic turbulence and file I/O was skipped.

For custom process mapping we use the Python tool Rubik [3], which was called from the jobscript and generates the map file on the fly. It provides an intuitive interface to create mappings for structured communication patterns and showed 10% performance increase for psOpen production runs in the past. As psOpen has a two-stage communication pattern with large packages in stage one and smaller packages in stage two, the mapping was chosen such that in the first stage only MPI processes on the same nodeboard communicate. The communication between MPI processes on different nodeboards therefore exchange the small packages of the second stage. Even though this mapping shows good results up to 16 rack runs, psOpen did not seem to benefit in the full-machine run. A satisfying explanation for this could not be found yet. As writing and reading of the mapfile for 28 racks required 15 minutes, further investigations was skipped and the custom mapping was disabled for the full-machine run setup.

Acknowledgements

Funding from the Cluster of Excellence “Tailor-Made Fuels from Biomass”, which is part of the Excellence Initiative of the German federal state governments, is gratefully acknowledged. The authors also gratefully acknowledge the computing time provided on the supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC) within the projects JHPC09 and HFG00.

References

[1] J. H. Goebbert, M. Gauding, M. Gampert, P. Schaefer, N. Peters, L. Wang; *A new View on Geometry and Conditional Statistics in Turbulence* Inside: Innovatives Supercomputing in Deutschland, The German National Supercomputing Centres HLRS, LRZ and NIC, 9(1), 30-37, (2011)

[2] N. Peters, L. Wang, J. P. Mellado, J. H. Goebbert, M. Gauding, P. Schaefer, M. Gampert; *Geometrical Properties of Small Scale Turbulence*, Proceedings of the John von Neumann Institute for Computing, NIC Symposium, 365-371, (2010)

[3] RUBIK, Lawrence Livermore National Laboratory, `https://computation.llnl.gov/project/performance-analysis-through-visualization/software.php`

SHOCK: Structured High-Order Computational Kernel for Direct Numerical Simulation of compressible flow

Manuel Gageik and Igor Klioutchnikov

Shock Wave Laboratory, RWTH Aachen University, 52056 Aachen, Germany

Description of the Code

SHOCK (**S**tructured **H**igh-**O**rder **C**omputational **K**ernel) is an application for Direct Numerical Simulation (DNS) of compressible flow using a shock-capturing high-order finite difference WENO-scheme to calculate the full system of Navier-Stokes equations on curvilinear structured meshes. The three-dimensional, two-dimensional rotational symmetric and two-dimensional unsteady, compressible flow can be simulated. Simulations of inviscid flows (Euler equations) are also possible.

SHOCK is programmed in C and contains various capabilities of the numerical method:

1. fifth and ninth order WENO scheme (spatial discretisation of first order derivatives)

2. sixth and tenth order central differences (spatial discretisation of second order derivatives)

3. third and fourth order Runge-Kutta (temporal discretisation)

WENO uses nonlinear weights preserving monotonicity in the vicinity of strong gradients (e.g. shocks). In order to improve the numerical stability of the WENO scheme, a local Lax-Friedrichs flux vector splitting is implemented. A powerful general mesh topology containing sub-zones with rotated coordinate systems is implemented enabling the use of arbitrary rotated and curved meshes. As a consequence, the introduction of a transformation matrix between the sub-zones and asynchronous MPI communication become necessary. SHOCK uses a pure MPI parallelisation. Further, instead of a global *Cartesian* communicator, now the mesh interfaces provide the information about communication partners that are connected via the interfaces.

SHOCK uses a file format for CFD data (CGNS [CFD General Notation System], using HDF5). Limited to reading only one file at the start and writing only one file at the end of the simulation (size $< 200\,\mathrm{GB}$ for current largest production), the I/O of SHOCK is optimised for ease-of-use and data pre- or post-processing is not needed. The results can be transferred continuously from the production system. Although HDF5 supports parallel I/O its performance is still an issue and further improvements are necessary. An additional small binary file contains the decomposition of the mesh.

Further details of SHOCK are published on our website [4] and publications, e.g. [1–3].

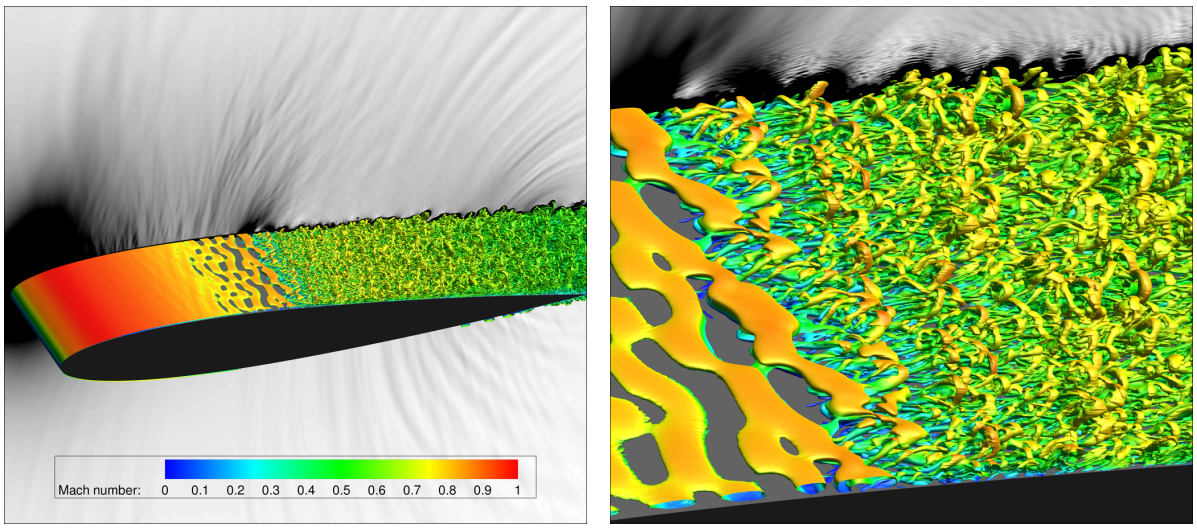

Figure 1: NACA 0012, $M_\infty = 0.65$, $Re_c = 5 \cdot 10^5$, $\alpha = 2°$, 1 billion mesh points. Q-criterion iso-surfaces (Q=100) coloured by local Mach number ($0 < M < 1$).

File I/O (HDF5) problems

Before the "Extreme Scaling Workshop" started, various configuration cases were generated and tested during the preceding "Porting & Tuning Workshop". Advice given by the JSC support team was very useful, since a critical problem of the file I/O was detected. The HDF5-based I/O showed scaling problems for opening of the CFD data file. The initialisation of SHOCK — mainly reading the start information (coordinates, velocity, pressure and density) — would take too long to perform all scaling configurations within the limited time window of the workshop. Within SHOCK, the command `cgp_open` (from the CGNS-library) is called and takes up to hours depending on the job size and file size. An HDF5 expert could reproduce this problem with the same file and confirmed the bad performance of CGNS and HDF5. In his view, the problem is caused by the current file-opening strategy. All processes each read the metadata of the HDF5 file using independent MPI file I/O operations. The I/O analysis tool Darshan showed thousands of small accesses reading only 512 bytes. An improved (not yet released) version of HDF5 replaces the global independent reading by a single read of the file by one process which then broadcasts the information to all. First results show significant reductions of the time for opening a file. Nevertheless, this version of CGNS/HDF5 is still not available, therefore a quick workaround was implemented. Since the binary file which is read in seconds contains all decomposition information, it was possible to generate the required start information within SHOCK. Consequently, the I/O was nearly deactivated and the initialisation phase then took only seconds.

Results

Using the deactivated I/O, 24 test cases were simulated. These consisted of six job sizes (bg_size = 8192, 12288, 16384, 20480, 24576 and 28672 [full machine]) for 4 problem types (weak scaling with 32 rpn [ranks per node], weak scaling with 64 rpn, strong scaling with 32 rpn and strong scaling with 64 rpn). For the strong scaling problem, a fixed mesh size (2688 x 2560 x 3072 = $2 \cdot 10^{10}$ mesh points) is utilised and decomposed for all ranks. The number of mesh points is not a power of 2 since the JUQUEEN bg_sizes contain also factors

of 3, 5 and 7 (which are not usually used). For the weak scaling problem, a fixed mesh size per rank is used. Due to an error within the job creation script, these mesh sizes are not identical for the 32 rpn and 64 rpn type. For all 32 rpn simulations the mesh sizes per rank are $16 \times 16 \times 16$, and for all 64 rpn simulations the mesh sizes are $16 \times 32 \times 32$. This is not really a problem but it prevents a direct comparison of both types. Consequently, a significantly larger simulation time of "weak, 64 rpn" compared to "weak, 32 rpn" is observable in Table 1.

The results of all test cases are illustrated in Figure 2. The lower x-axis represents the bg_size and the upper x-axis the respective number of MPI ranks. The y-axis represents the speedup with respect to the smallest job (bg_size = 8192). The weak scaling curves (on the left side) show good scaling since the performance loss caused by an increasing bg_size is approximately 10% for the worst case (maximal bg_size). A detailed quantitative evaluation of this loss is not possible since the mesh sizes per rank of the production situations and even for "weak, 32 rpn" and "weak, 64 rpn" are different, and therefore also the relation of time for computation to communication is different. However, the general scaling behaviour of the weak scaling problem is more than satisfactory (ideally unity for all bg_sizes) and it is reasonable to assert that SHOCK is capable of effectively simulating even larger problem sizes with the JUQUEEN maximum bg_size of 28672.

Against this background, hints from some other participants of the "Extreme Scaling Workshop" were very insightful. First, we implemented a version using single-precision for all variables (formerly all double-precision). We tested it after the workshop and could reduce the simulation time by 30%. The analysis of the impact on accuracy is not yet done. Second, we obtained a subroutine to obtain the memory utilisation and used it in SHOCK. These improvements can allow enlargement of the problem size per rank (for a constant memory size) since the memory requirement is reduced (from double to single precision) and monitoring of memory usage is possible (using the provided subroutine). Consequently, the ratio of time for computation to communication is improved and efficiency could be increased. Here, further tests are necessary.

The second type of test cases are the strong scaling curves (right side of Figure 2). An additional dashed line is plotted which shows perfect linear scaling behaviour. The blue curve (32 rpn) scales nearly linearly. The orange curve (64 rpn) seems to show super scaling behaviour. This means that SHOCK is able to obtain an extra performance boost from the decomposition of the problem size. A possible explanation might be that the number of cache misses is reduced. When the processor looks in the cache (which is very fast) and does not find the item being looked up, it has to go to memory (which is much slower than cache). For a smaller problem size, the probability is larger that the processor finds the item in the cache. Here, the change from double to single precision could additionally increase this effect since twice as many items can be stored in the cache. The drop of the orange "strong, 64 ppn" curve at bg_size = 28672 is perhaps caused by a very bad ratio of time for computation to communication. This is linked with the relation of ghost points (which are communicated to neighbour processes) and real points which store computation results. For the considered case ("strong, 64 rpn" with bg_size = 28672) the relation of ghost points to real points is 1.03. Benefit gained from the local problem size reduction is smaller than the loss added by the extra communication required for this decomposition. However, the reduced memory effort (double precision to single precision) of SHOCK leads to the possibility to increase the problem size. Then the ratio of time for computation to communication is much better and the drop ("strong, 64rpn" at bg_size = 28672) might disappear.

In Table 1, simulation time measurements in minutes are listed for all test cases. Time for communication was also recorded but is not shown, since it was measured only by rank 0 and with no blocking communication or additional `MPI_Barrier` before and after the communication (SHOCK uses `MPI_Isend`/`MPI_Irecv` and `MPI_Waitall`) these measurements are therefore

Table 1: SHOCK scaling results (weak/strong scaling, 32/64 ranks per node)

rpn	bg_size	MPI ranks	Time (min.) Weak	Strong
32	8192	262 144	0.142871	2.527755
32	12288	393 216	0.144875	1.663479
32	16384	524 288	0.146119	1.317004
32	20480	655 360	0.159132	1.067620
32	24576	786 432	0.159243	0.856886
32	28672	917 504	0.165825	0.751480
64	8192	524 288	0.836828	2.567251
64	12288	786 432	0.878316	1.397425
64	16384	1 048 576	0.874725	1.084076
64	20480	1 310 720	0.923541	0.924936
64	24576	1 572 864	0.912548	0.754874
64	28672	1 835 008	0.940829	0.993653

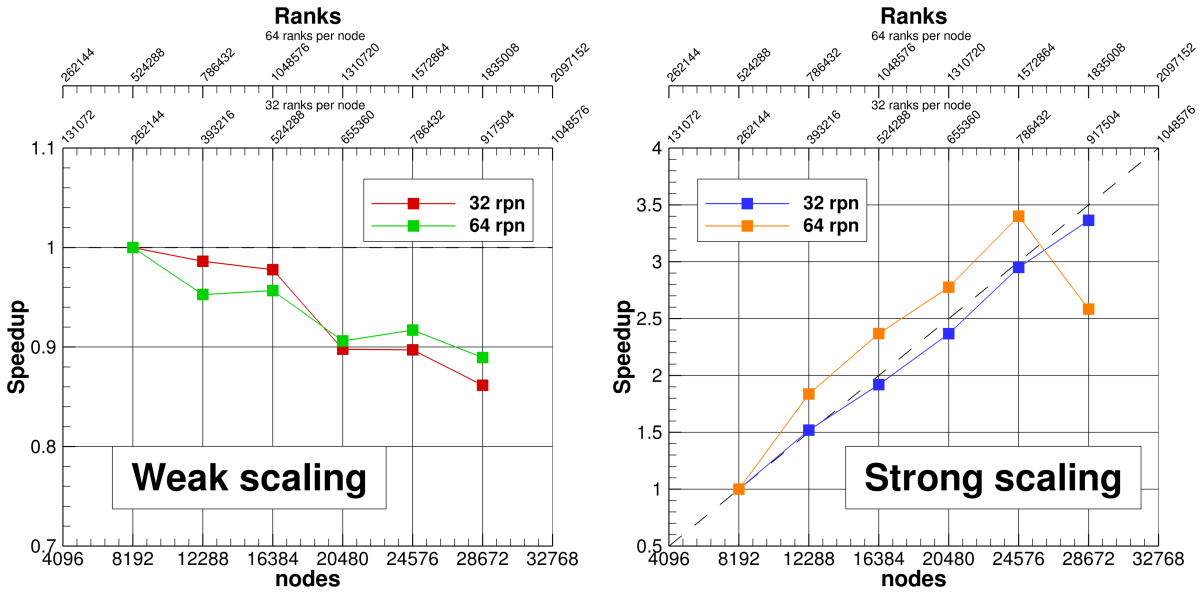

Figure 2: SHOCK weak and strong scaling graphs (32/64 ranks per node)

not representative for the entire collection of processes.

All in all, we are very content with SHOCK's scaling behaviour. It is above our expectations. The only limiting factor (detected so far) is the I/O using HDF5, where we are largely depending on improvements of its parallel I/O.

References

[1] M. Gageik, I. Klioutchnikov, H. Olivier; *Mesh study for a direct numerical simulation of the transonic flow at Re_c = 500,000 around a NACA 0012 airfoil*; DGLR Jahrestagung - Deutscher Luft- und Raumfahrtkongress, Augsburg, September 2014, DGLR-2014-0028

[2] V. Hermes, I. Klioutchnikov, H. Olivier; *Numerical investigation of unsteady wave phenomena for transonic airfoil flow*; Aerospace Science and Technology **25(1)** (2013) 224-233; [doi:10.1016/j.ast.2012.01.009]

[3] V. Hermes, I. Klioutchnikov, H. Olivier; *Linear stability of WENO schemes coupled with explicit Runge-Kutta schemes*; Int. J. Numer. Meth. Fluids **69** (2012) 1065-1095; [doi: 10.1002/fld.2626]

[4] `http://www.swl.rwth-aachen.de/en/numerical-simulation/shock/`