Parallel Computing 90 (2019) 102567

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Review

Student Cluster Competition 2018, Team University of Warsaw, University of Wroclaw, Warsaw University of Technology: Reproducing performance of a multi-physics simulations of the Tsunamigenic 2004 sumatra megathrust earthquake on the intel skylake architecture

Julia Bazińska[a], Maciej Korpalski[b], Maciej Szpindler[c,*]

[a] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
[b] Faculty of Mathematics and Computer Science, University of Wrocław, Poland
[c] Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw, Poland

ARTICLE INFO

Article history:
Received 12 April 2019
Revised 14 September 2019
Accepted 1 October 2019
Available online 14 October 2019

Keywords:
Student cluster competition
Numerical reproducibility

ABSTRACT

In their work, Uphoff et al. (2004) presented optimization of the SeisSol code. For the reproducibility challenge during the Student Cluster Competition at SC18 we reproduce selected results from the mega-scale earthquake simulation by Uphoff et al. on our competition computer cluster. Our cluster configuration is a 4-node system with Intel Xeon Gold processors (96 cores total, Skylake architecture) running under a 3 kW power limit imposed by the competition. We ran the computations during a 48-h marathon along with other competition applications. We successfully reproduced an increase in FLOPS w.r.t. the discretization order. Results we obtained for scaling are not consistent with the original work due to crucial hardware differences.

1. Introduction

SeisSol is a large-scale simulation software for computational earthquake dynamics. In the paper by Uphoff et al. [4] several performance optimizations are introduced, resulting in significant speedup. One of the tasks of the Student Cluster Competition 2018 was to investigate two major code improvements and reproduce these performance results. The Student Cluster Competitions (SCC) at the Supercomputing Conference gathers international teams of undergraduate students from Science, Technology, Engineering and Mathematics (STEM) disciplines to compete using clusters of their own design. Each team demonstrates performance of their architecture having only a 3 kW of power budget available to run real-world High Performance Computing (HPC) workloads. This report presents our approach and final achievements with reproducing given original results of Uphoff et al.

The first optimization approach is the flux matrix decomposition. Instead of multiplying three matrices in each time step, a chain product of five smaller matrices is used in such a way that both the resulting number of FLOPs and memory usage drops to $O(N^5)$ from $O(N^6)$, as described in 3.4.1 of Uphoff et al. [4]. This optimization is particularly suited for the Intel Knights Landing (KNL) many-core architecture. Decomposed flux matrices can fit in the KNL L2 cache, thus overcoming the problem of constant cache eviction. For small matrix multiplication, static code generated by the `libxsmm` library is used.

Secondly, Local Time Stepping (LTS) for dynamic ruptures is introduced as an optimization. It takes advantage of computing time steps tailored for specific elements in dynamic ruptures, thus overcoming a major performance bottleneck and achieving a speed-up of up to 6.8 in comparison to global time stepping.

The following sections include descriptions of our hardware setup, experiment details, results and final conclusions.

2. Experimental setup

The hardware used was designed for the SCC. It consists of 4 identical compute nodes. Despite the 3 kW power limit, we did not have to power cap the CPUs. The Hardware configuration is as follows:

- Dell PowerEdge R740,
- Intel Xeon Gold 6126 2.6 GHz CPU,
- two sockets of 12 cores (24 cores in total),
- 192GB DDR4 RAM,

* Corresponding author.
E-mail address: m.szpindler@icm.edu.pl (M. Szpindler).

https://doi.org/10.1016/j.parco.2019.102567
0167-8191/

- NVIDIA Volta V100 GPU Accelerators (not used for reproducibility task),
- GigaIO FabreX interconnect (not used for reproducibility task),
- 1Gb Ethernet network.

The Software configuration is as follows:

- MooseFS 3.0.101 distributed filesystem,
- CentOS 7.5 1804,
- Linux Kernel 4.16.14,
- Intel Parallel Studio 2018.

Theoretical peak performance of one compute node is 1996 GFLOPs (for double precision operations at 2.6 GHz).

Simulation datasets were provided by the SCC committee – they are size-reduced versions of input data used in [4].

We intended to use the GCC 7 compiler and MPICH 3.3b2 (a version with support for FabreX) to benefit from both fast inter-node MPI communication and on-node performance based on Skylake's AVX instructions. Unfortunately, during the competition marathon, we experienced problems with the interconnection hardware and compute nodes becoming unpredictably unresponsive. Due to SCC rules and other tasks running, we could not reboot the system. Thus we decided not use the FabreX interconnect and roll back to 1Gb Ethernet, also switching to Intel MPI and the Intel compiler suite. 1Gb Ethernet is slower compared to FabreX communication, so we expected a performance drop while running over multiple nodes. However, after conducting the experiments, it turned out that the network was not a significant bottleneck.

3. Compilation and running

Details of the compilation of SeisSol and its dependencies can be found in the Appendix. Parameters used for the runs are also provided in the Appendix.

We invoked the runs with following commands:

```
ulimit -s unlimited
export OMP_NUM_THREADS=24
mpirun -n $NUM_NODES./SeisSol params.par
```

SeisSol uses big static arrays and because Fortran stores such arrays on stack, we have to set stack memory to unlimited in order to launch a run.

To achieve the full potential of our Skylake architecture we used the AVX-512 instruction set (F, CD, VL, DQ, BW). We did that by adding an '`skx`' option to:

```
generated_code/gemmgen/Arch.py,
site_scons/arch.py,
src/{\ldots} /precision.h,
```

as described in [3]. We configured it with 64 byte memory alignment and additional compilation flags:

```
-xCORE-AVX512 -fma.
```

Most of our work compares two versions of SeisSol with the same naming convention referred to in [4]: Baseline (BL) is the first version of SeisSol optimized for KNL architecture and Shaking Corals (SC) is the newer, fine optimized version. Most of features described here refer to the SC version.

SeisSol can use various matrix memory layouts. Matrices can be stored in either dense, sparse, or block partitioned memory layout. The authors provide a proxy application for testing these layouts on random data. We used it for both Baseline and Shaking Corals versions to investigate how memory layout affects actual performance. We ran the auto-tuning script for the discretization order 5 (figuring it is complex enough while still not very long to run) to verify which layout minimizes time to solution. We identified the dense layout as the best choice for our configuration.

4. Experiments

First, we describe and discuss the outcome of our attempt to reproduce the results from Uphoff et al. [4] concerning performance improvement of the Shaking Corals version over the Baseline. In the second experiment we analyze the impact of Local Time Stepping on performance in comparison with Global Time Stepping.

4.1. Single node performance

In the Shaking Corals version, the flux matrix decomposition is described to reduce the number of floating point operations and also minimize memory usage, thus preventing cache eviction. In [4] several experiments are presented. We aimed to reproduce the performance and speed-up figures associated with the whole wave propagation scheme on a single node of our competition cluster. In the original experiments, a speed-up of 1.275 on average (over discretization orders 4–7) is achieved on Intel Haswell (HSW) node architecture and 1.21 on average on KNL.

In order to run the baseline version we use an older proxy application from the `libxsmm` repository [1] (commit `b62a688`) for reprodicibility. Computations were done in double precision in the same configuration as the SC version (single node, 24 OpenMP threads).

To run the Shaking Corals version we used the proxy application from the git repository [2]. In our tests, the matrix storage used by the libxsmm library was set to dense, chosen with the auto-tuning procedure. The tests were conducted in single precision and ran on a single node (two 12-core CPUs, one MPI task, 24 OpenMP threads). We used the default thread affinity.

We measured absolute single node performance on the proxy application. Fig. 1 shows measured GFLOPS as reported by the applications.

The BL experiments were run in double precision and the results can be compared with those obtained by Uphoff et al. Absolute performance achieved for this version was higher than on HSW architecture and lower than on KNL (for all discretization orders). We point to the differences in the architectures as the reason for the performance difference. Skylake supports AVX-512 instructions and Haswell supports only AVX2. In comparison to KNL, Skylake has much smaller peak performance (less cores). The potential of the better cache architecture in Skylake is not exploited, as the BL version does not have cache optimizations.

The SC based measurements were collected in single precision. Due to the difference in precision, hardly any conclusions could be drawn from speed-up numbers between SC and BL. For this reason we focus on absolute performance numbers measured and improvements between different discretization orders.

In comparison to the original study, our SC performance studied for orders from 4 to 7 showed similar trend in GFLOPs improvement: a rapid increase from O4 to O5 and from O5 to O6, but comparable results for O6 and O7. The highest number of GFLOPs was achieved for order 7 and reached 45% of the node's peak performance in single precision, which is less than Uphoff's results in double precision (56% for HSW and 50% KNL). Difference in precision provides no fair direct comparison to original numbers reported by Uphoff et al.

We were able to reproduce performance improvement for increasing discretization orders using the Skylake architecture. We did not achieve as high percentage of machine usage compared

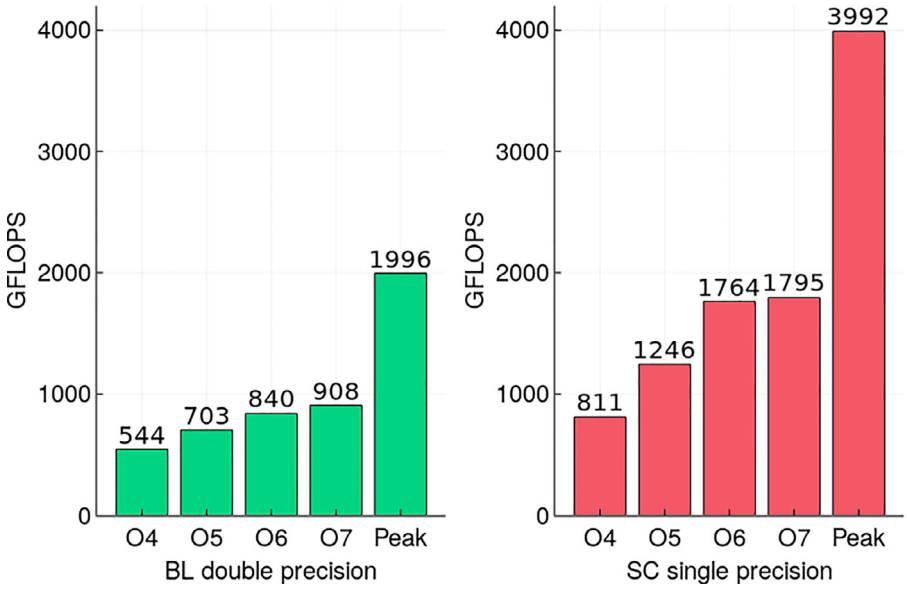

Fig. 1. Absolute performance on single node proxy tests of Shaking Corals (SC) and baseline (BL) on a mesh with 100 000 elements ran for 100 timesteps.

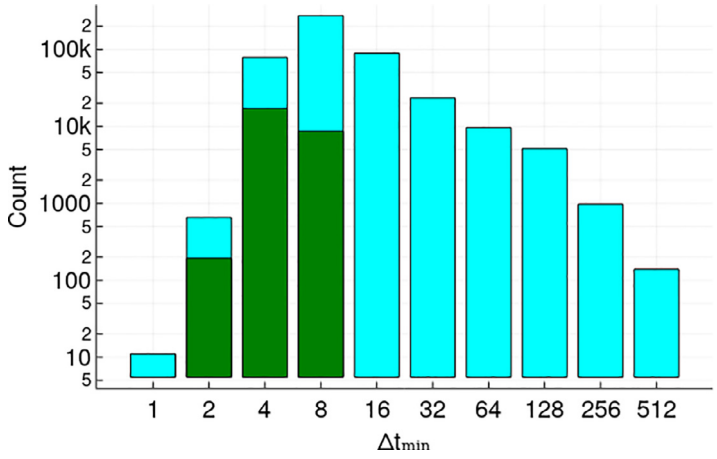

Fig. 2. Elements per time cluster. Cyan – regular elements, green – dynamic ruptures. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

to Uphoff et al. [4]. Explanation for this behavior is unclear and requires further investigation, since we used single precision and a newer node architecture generation compared to the original study.

4.2. Local versus global time stepping

SeisSol typically uses Local Time Stepping (LTS), prior to optimizations introduced in [4] for the regular elements. In the LTS method, the elements are grouped into clusters of time steps of length being powers of 2. By decreasing the time step for more demanding elements and increasing it for the less demanding ones, the simulation achieves better performance. In [4], LTS for dynamic rupture optimization are introduced. Thanks to a reformulation of the numerical scheme, calculations are further optimized by using static code generation for dense matrix multiplication. In Fig. 2 the time cluster assignments are shown. We relied on ParMETIS partitioning to balance the work load for LTS.

As our cluster has 4 nodes with 2 sockets of 12 cores each, we decided to do experiments on 6 cores (half of a socket, one MPI process), 12 cores (one socket, one MPI process), 24 cores (2 sockets, one node, two MPI processes) and 48 cores (2 nodes with 2 sockets each, four MPI processes). We allocated one MPI task per socket to utilize NUMA domains effectively with multiple threads. In all of the experiments the default Intel MPI thread affinity was used. Only the last experiment involved inter-node communication (over 1Gb Ethernet).

Due to the time constraint at the competition, we decided to run all the LTS and GTS experiments in single precision.

Unfortunately, we did not have enough time to run SeisSol on more than 2 nodes during the competition. All runs used discretization order 6, with 481 565 elements, versions using local and global time stepping are named L6 and G6, respectively. Results are shown in Figs. 3 and 4.

Generally, LTS achieved higher performance. The ratio of LTS to GTS performance is the highest on 12 cores and is equal to 99%, while the lowest ratio of 89% is achieved on two nodes. On average the ratio was 94%. This is comparable to the ratio obtained in the original work for the Shaheen cluster (92%).

Scalability results were not consistent with Uphoff et al. because in the Uphoff et al. study, the number of nodes in the experiments was in terms of hundreds. Another reason could be our use of a 1 Gb Ethernet interconnect, while in the original

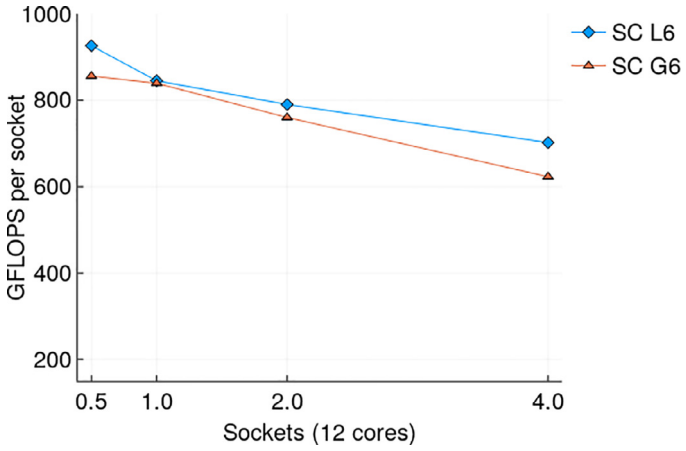

Fig. 3. Scalability with discretization order O6. We ran experiments on 6, 12, 24 and 48 cores.

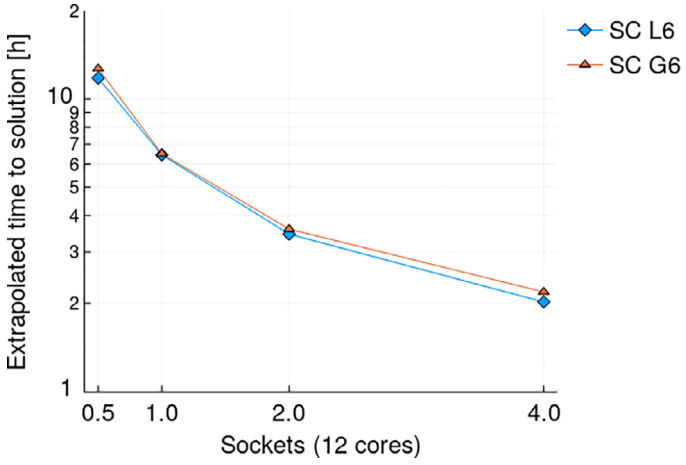

Fig. 4. Extrapolated time to solution with discretization order O6. We ran experiments on 6, 12, 24 and 48 cores.

experiment used Infiniband or Aries. We point out the latter as the reason for the FLOPS drop for two nodes.

5. Geophysical results

The datasets, we were given, was a size-reduced, less accurate version of the earthquake simulation described in [4]. In a short time we managed to run multiple 500 s simulations.

While SeisSol relies on ADER explicit time integration scheme, it enforces the Courant–Friedrichs–Lewy (CFL) condition which allows numerical convergence. With CFL set to a given value of $CFL = 0.5$, the simulation turned out to become unstable after 40s. We did not have time to run the whole experiment again during the competition, so the visualization is reduced to a time window of $t = 50$ s (5 timesteps).

Uphoff et al. in [4] analyzed simulated horizontal and vertical surface displacements and found matching with GPS observations. In the original simulation, rupture lasted about 440 s, which the authors assessed as slightly too fast. Due to unstable conditions of our simulation we were only able to interpret first phase of the simulation. Synthetic horizontal and vertical surface displacements after 50 s of the simulation show development of the earthquake (Figs. 5 and 6).

Our results are not directly comparable with original study. We can only analyze the very beginning of the process. Simulated surface displacements are only visible in a limited geographical region and tend to develop in similar directions as in the original paper. The magnitude is similar for vertical displacement and lower for horizontal displacement probably due to the much shorter simulated time window.

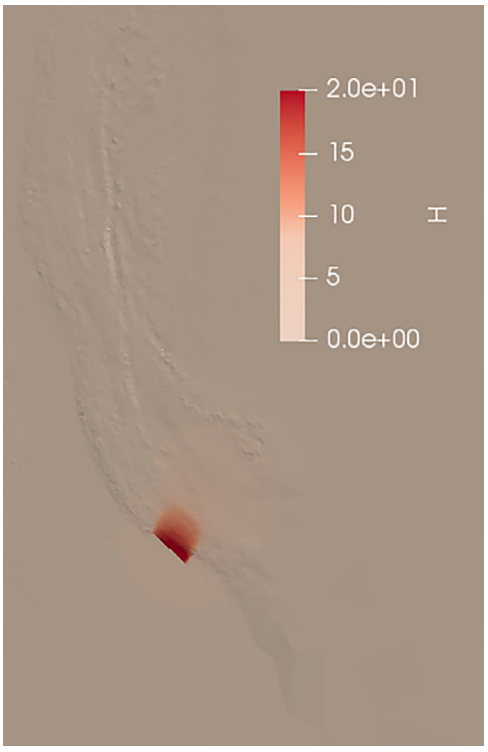

Fig. 5. Synthetic horizontal surface displacement (H).

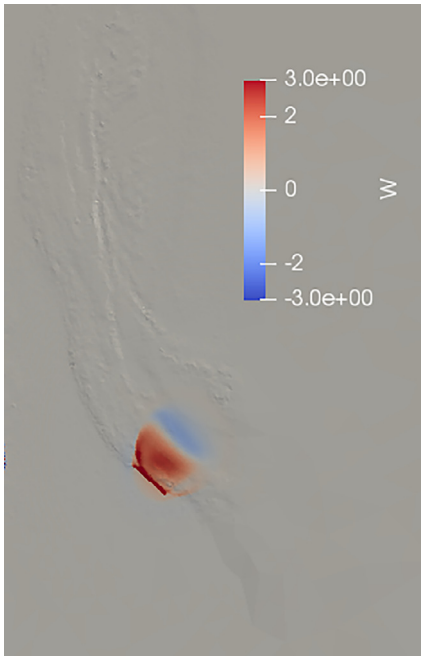

Fig. 6. Synthetic vertical surface displacement (W).

6. Conclusions

In conclusion, our results were not consistent with Uphoff et al. and we were not able to reproduce their results. This could be

due to the profound differences in hardware used between our experiments.

In the first experiment concerning Baseline and Shaking Corals performance, we managed to reproduce the FLOPS increase for increasing discretization orders, while not achieving such high machine usage. We cannot verify the improvement of the Shaking Corals version because of the difference in precision.

In the second experiment investigating Local and Global Time Stepping, it is hard to judge whether the results were reproduced. In [4] the number of machines used was two orders of magnitude greater. It was not possible to conduct such an experiment in the conditions of the Student Cluster Competition. We did not achieve as strong scaling.

Declaration of Competing Interest

None.

Acknowledgments

This work would not be possible without the hardware provided by the Texas Advanced Computing Center (TACC) for the competition.

Participation in the Student Cluster Competition was supported by the University of Warsaw rector with a travel grant.

This research was carried out with the support of the Interdisciplinary Centre for Mathematical and Computational Modelling (ICM) at the University of Warsaw under grant number G69-12.

Appendix A. SeisSol compilation and parameters file

Before compiling SeisSol, we had to set paths to libraries and compilers. It is done by script setup_paths.sh:

```
source $INTEL_HOME/compilers_and_libraries\
/linux/bin/compilervars.sh intel64
export SEISSOL_HOME=.
export PATH=$HDF5_HOME/bin:$NETCDF_HOME/bin:\
$METIS_HOME/bin:$PARMETIS_HOME/bin:\
$SCONS_HOME/bin:$PATH
export LD_LIBRARY_PATH=$HDF5_HOME/lib:\
$NETCDF_HOME/lib:$METIS_HOME/lib:\
$PARMETIS_HOME/lib:$SCONS_HOME/lib:\
$LD_LIBRARY_PATH
export SCONS_LIB_DIR=$SCONS_HOME/lib/scons-2.2.0
export PATH=$LIBXSMM_HOME/bin:$PATH
source $SEISSOL_HOME/venv/bin/activate
export C_INCLUDE_PATH=$HDF5_HOME/include:\
$NETCDF_HOME/include:$METIS_HOME/include:\
$PARMETIS_HOME/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$HDF5_HOME/include:\
$NETCDF_HOME/include:$METIS_HOME/include:\
$PARMETIS_HOME/include:$CPLUS_INCLUDE_PATH
```

Compilation:

- HDF5 library

```
CC=mpicc ./configure \
--enable-parallel --prefix=$HDF5_HOME
make -j 24 && make -j 24 install
```

- NetCDF library

```
CPPFLAGS=-I$HDF5_HOME/include
LDFLAGS=-L$HDF5_HOME/lib CC=mpicc
./configure --enable-shared=no \
--prefix=$NETCDF_HOME
make -j 24 && make -j 24 install
```

- METIS library

```
make config prefix=$METIS_HOME
make -j 24 && make -j 24 install
```

- ParMETIS library

```
make config prefix=$PARMETIS_HOME
make -j 24 &&make -j 24 install
```

- LIBXSMM library

```
make generator
```

- SCons

```
python setup.py install \
--prefix=$SCONS_HOME
```

To properly use scons on our cluster, we use python virtualenv in order to get required packages.

```
virtualenv-3 -p python2 venv
```

Next we have to adjust system paths (setup_paths.sh) and SeisSol compilation parameters. SeisSol compilation:

```
scons -j 24 buildVariablesFile=\
build/options/tatry_cluster.py
```

Compile parameters used:

```
compileMode = 'release'
parallelization = 'hybrid'
generatedKernels = 'yes'
measureNodeLevelPerformance = 'none'
useExecutionEnvironment = 'yes'
logLevel = 'warning'
logLevel0 = 'info'
compiler = 'gcc'
netcdf = 'yes'
netcdfDir = '/mfs/seissol/libs/netcdf/'
hdf5 = 'yes'
hdf5Dir = '/mfs/seissol/libs/hdf5/'
metis = 'yes'
metisDir ='/mfs/seissol/libs/parmetis/'
arch = 'hsw'
order = $ORDER
```

References

[1] libxsmm: Github repository, https://github.com/hfp/libxsmm [Online; accessed 22-May-2019].
[2] SeisSol: Github repository, https://github.com/SeisSol/SeisSol [Online; accessed 22-May-2019].
[3] SeisSol wiki: optimization for non Intel architectures. https://github.com/SeisSol/SeisSol/wiki/Optimization-for-non-Intel-architectures [Online; accessed 14-November-2018].
[4] C. Uphoff, S. Rettenberger, M. Bader, E.H. Madden, T. Ulrich, S. Wollherr, A.-A. Gabriel, Extreme scale multi-physics simulations of the tsunamigenic 2004 sumatra megathrust earthquake, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 21.