



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Report

Student Cluster Competition 2022

written by

Christian Grüneberg, Niclas Schroeter, Lukas Schulte,
Frederic Voigt, Christian Willner and Johannes Wünsche

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Supervision UHH: Jannek Squar
OvGU: Michael Blesel
Co-Supervision UHH: Anna Fuchs
OvGU: Michael Kuhn

Hamburg, October 29, 2022

Contents

1	Introduction	4
2	Team Cluster	6
2.1	Hardware	6
2.2	Operating system	6
2.3	Storage and file system	7
2.4	Problems and difficulties	8
2.4.1	Software and driver problems	8
2.4.2	Power Cables	9
2.4.3	GPU topology	9
2.4.4	CPU	11
3	Online Competition Clusters	13
3.1	Niagara	13
3.2	Thor	13
3.3	Bridges-2	14
4	Microbenchmarks	15
4.1	HPL	15
4.2	HPCG	16
4.3	HPCC	16
5	ICON	18
5.1	Building and configuration	19
5.2	Compilers and dependencies	19
5.3	Runtime parameters	21
5.4	Final configurations for the online competition	23
5.5	Profiling	24
5.6	On-site competition	26
6	NWChem	27
6.1	Tuning	27
6.1.1	Scripts	28
6.1.2	Input Tuning	28
6.2	Results	29
6.3	Profiling	32

7 Xcompact3D	34
7.1 Online Competition	36
7.2 Coding Challenge	43
7.3 On-site competition	44
8 Secret Application - FALL3D	45
9 Learnings	47
Bibliography	49
Appendices	51
List of Figures	52
List of Tables	54

1 Introduction

Authors: Frederic Voigt, Christian Willner

This is the report for SCC 2022 of the Team Elbe. SCC is the abbreviation of the Student Cluster Competition – a friendly comparison of the HPC skills between university teams around the world. Co-organized by the HPC-AI Advisory Council and the ISC group, the competition takes place on the exhibitor floor during the ISC High Performance Conference. The 2022 edition was held in Hamburg, Germany.

As a consequence of the ongoing COVID-19 epidemic, the SCC was restructured into a two-part event. First there was a digital part that was open for applicants of all universities worldwide. The on-site event was restricted to teams in Europe.

As a steady participant in the SCC, the UHH has applied for the 2022 edition with a new twist. This year, the team would be a joint-venture with the OvGU Magdeburg. Due to their former involvement at the SCC and the university of Hamburg, Michael Blesel and his thesis advisor Michael Kuhn in Magdeburg gave this construct more than one good reason to be tried.

With the general teaching and studying experiences during the pandemic, the team was sure that the distance between the universities would not hinder good communication and co-working. After some very populated first meetings with more than 30 interested students, six came forward to become the on-site team.

The first step showed success, as the Team Elbe was selected as a participant in both the digital and the on-site part of the SCC¹. There were 17 teams, from six continents, in total to participate in the online event. The other four on-site teams were from Heidelberg, Edinburgh, Zürich and Barcelona.

To even out the playing field, the clusters of the digital event were provided by the organizers. Every qualified team had to optimize different applications on three different HPC architectures.

A very interesting part of the on-site event is the diversity of the student teams' clusters. Team Elbe is strongly connected to the DKRZ in Hamburg and therefore also to their hardware provider Atos. Atos was kind enough to supply the student team with two powerful nodes for the time of the competition. At the time of the planning, the challenges had not been posted and the student team was sure that GPUs would be a big part

¹<https://www.businesswire.com/news/home/20211209005803/en/HPC-AI-Advisory-Council-and-ISC-Group-Announce-2022-Student-Cluster-Competition-Roster>

of the upcoming event. Due to the renaming of the advisory council of the SCC into HPC-AI, the team was also confident that at least some of the tasks were related to artificial intelligence (AI). For these reasons the 3 kW power budget was invested in four powerful NVIDIA N100 GPU cards (Section 2.1). A bet many other teams in the competition joined in on.

The following text will first discuss the hardware used by our team on-site, as well as the hardware provided for the online competition. After that, the microbenchmarks will be described, followed by the individual applications that were handled in the course of the competition. In the end we will summarize our learnings and discuss problems for future teams to avoid.

2 Team Cluster

Authors: Lukas Schulte

2.1 Hardware

Thanks to our industry partner Atos/Bull¹, which currently sets up the new supercomputer at DKRZ – Levante² –, we will again have access to powerful hardware. During the last years, we have tried many different systems, including CPU-only, GPU-based, Xeon Phi systems and vector machines. This year we got a well balanced system using both powerful GPUs and CPUs, very fast SSDs and lots of RAM split up into only two nodes as shown in Figure 2.1. This has the advantage that it does not need any switches or other network equipment and allows for high speed communication using the 200Gb Infiniband NICs.

2.2 Operating system

For the operating system, we went with Rocky Linux 8.6, which was a first for our team. In previous years the operating system of choice was CentOS, a distribution that's binary compatible with RHEL and maintained by Redhat themselves. Unfortunately, Redhat discontinued the CentOS project and ended all software support for CentOS 8 in 2021. So we had to find a different distribution. We had only a few criteria to be met by the distribution: First of all, we wanted a relatively recent kernel version for compatibility reasons. Then we preferably wanted an RHEL-compatible system because of the great hardware and software support for file systems, drivers, and applications. When Redhat announced the discontinuation of CentOS many new RHEL-compatible distributions emerged, trying to take CentOS's place in the market. Two of the fastest-growing alternatives are Alma Linux and Rocky Linux. We wanted to give one of them a try and choose Rocky Linux 8 as our operating system.

Generally speaking, Rocky Linux did a great job and everything worked well and was reliable. But, as discussed in Section 2.4.1, the release model of Rocky Linux turned out to be not optimal for this kind of system where you don't necessarily want to upgrade the operating system version only to install additional software.

¹<https://atos.net/en/solutions/high-performance-computing-hpc>

²<https://docs.dkrz.de/doc/levante/>

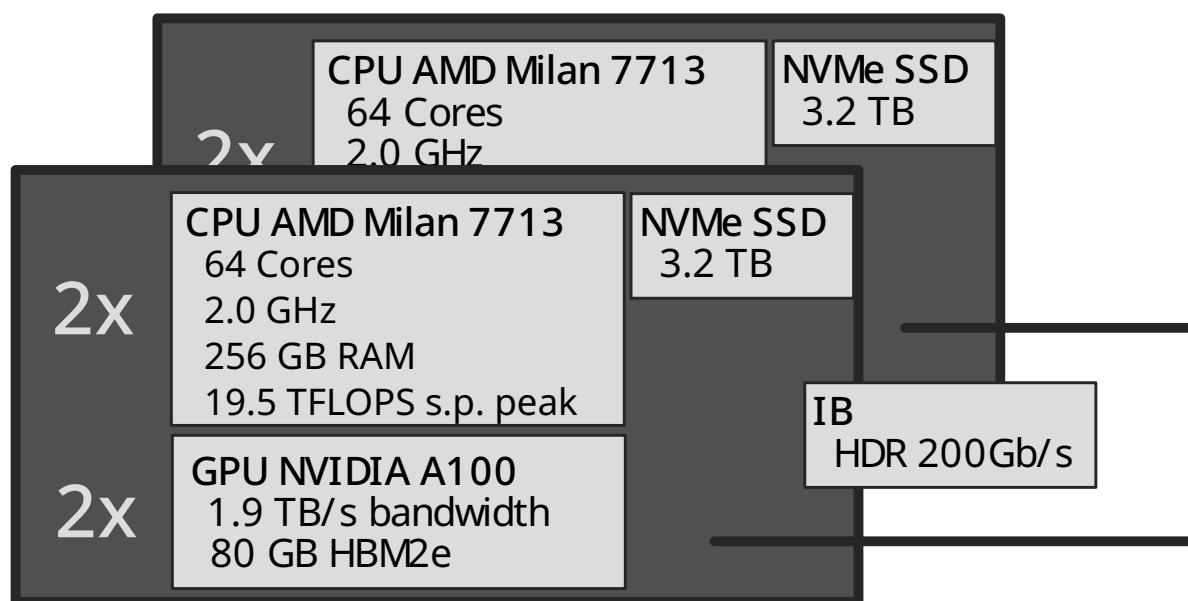


Figure 2.1: Sketch of our cluster.

2.3 Storage and file system

Even though we had only two nodes, the storage architecture wasn't necessarily straightforward. We knew that this kind of software typically produces large amounts of data that is written to the drives, so we expected the storage system to have a great impact on the overall performance. Moreover, since these applications made use of the MPI-IO libraries we needed a file system that supports this I/O standard. We decided to make one of the nodes a storage node that handles all the files and metadata while being a client too. Fortunately, Atos provided us with very fast NVMe SSDs so we were looking forward to getting excellent throughput and latency results even with a single disk. We put all three spare NVMe SSDs into the storage server and created a simple RAID 0 Volume using a software RAID through LVM. As the base file system, we did choose XFS as a fast filesystem with very little overhead. Using a RAID 0 volume comes with a lot of risks, but given that there was no important data stored in this volume, everything was backed up to other places and the entire system had to work only for a few days, we did choose the performance improvement over the better redundancy.

On top of all of that, we did choose BeeGFS as a parallel file system. BeeGFS is a hardware-independent POSIX-compliant parallel file system that is designed with performance and ease of use in mind. In previous competitions our teams already had good experiences with BeeGFS and so did we. Notably, the installation and configuration were very easy and straightforward. Without any previous experience with this software, we were able to install and configure the whole file system and network shares within minutes.

One of the advantages of BeeGFS is its scalability and most importantly its ability to scale it down to a very small cluster. Usually, this kind of file system is designed with

many different nodes with different roles in mind. BeeGFS uses a dedicated metadata server, management server, multiple storage servers, and clients that can then access the files stored on this distributed file system. In our case, one node was just a client and the other node was metadata, management, storage server, and even client all at once, still providing decent performance with all desired features of a parallel file system. Still, we were not able to fully saturate the theoretical performance of our storage system through the BeeGFS clients but we did not invest too much time into tuning the FS. Since we already knew that all of our applications weren't very limited by the I/O system and still achieving multiple Gigabytes per second throughput, we didn't invest much more time into tuning. Luckily, at the on-site competition, we met some BeeGFS engineers who gave us the hint that increasing the worker thread count might improve performance in our case and indeed it did.

Even though our storage system is pretty much overkill for most applications, we think it helped a lot in the secret application. As described in Chapter 8 the parallel IO library gave us a huge performance improvement. With the high throughput and low latencies due to the SSDs and Infiniband connection, this setup turned out to be very capable.

2.4 Problems and difficulties

Unfortunately, this section is a rather long one since we encountered a lot of problems. It all started very well. We got Atos as a sponsor who was willing to give us two very capable nodes for the competition. Even though there was a global shortage of all semiconductor products the hardware arrived very early and in theory we had plenty of time to test and configure our systems. We set up our two machines with the network connection and began the testing of the software. But only a few days in one of our nodes died due to a hardware failure. We don't know what exactly failed, probably a mainboard defect. Therefore we had to proceed using only one node until we received the replacement node which arrived just in time, one week before the competition started. So we could neither test the Infiniband connection, the network file system or scaling of the software beyond a single node.

2.4.1 Software and driver problems

Once we finally received our second node we tried to prepare as much as possible before the competition started. But when installing the necessary drivers for the Infiniband NICs one of the systems was completely bricked and we had to reinstall the whole operating system and redo all configurations. In theory, we had scripts that automated the installation and configuration of most parts. But unfortunately, there were two major changes in the software these weeks which made our scripts pretty much useless. First of all, there was a major version release of Redhat Enterprise Linux (version 8.6). Since we went with Rocky Linux instead of RHEL or other binary-compatible distributions

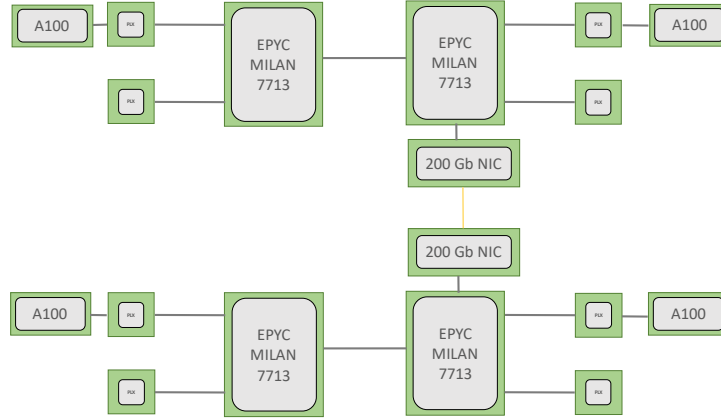


Figure 2.2: Initial system topology.

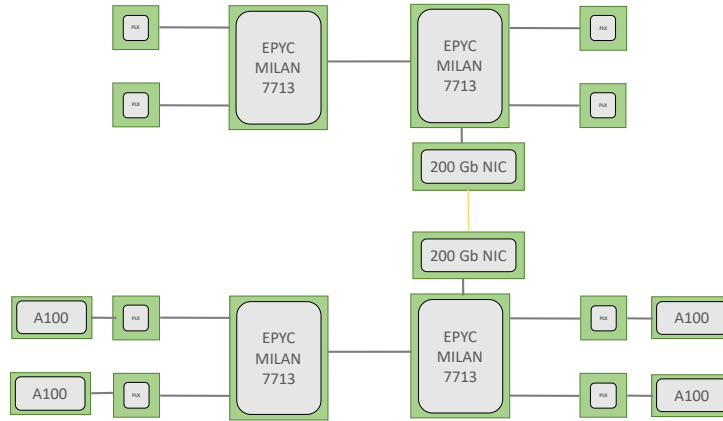
this was a problem for us, because Rocky Linux immediately shuts down all support for a major release version once a new one is released. Therefore we were forced to upgrade both nodes to Rocky Linux 8.6. Moreover, in this exact week, NVIDIA decided to release their GPU kernel modules as open source which changed the whole installation procedure. All of this did cost a lot of time and we had only two days left for testing after the initial setup of both nodes.

2.4.2 Power Cables

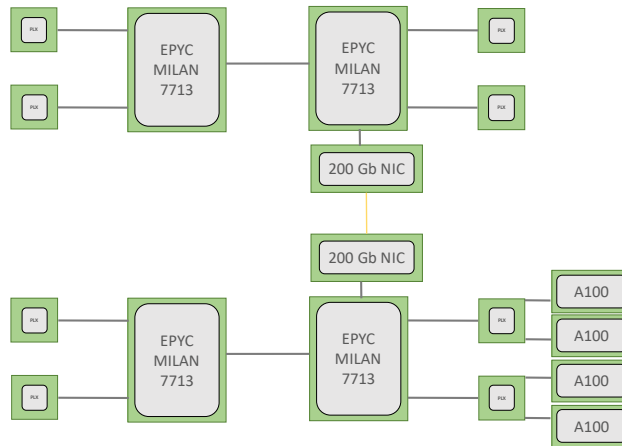
When we arrived at our booth at the ISC we immediately encountered a problem with the power outlets provided by the ISC. We needed four of the IEC 60320 C19 standard couplers but there were only two of them available. Unfortunately, they had no other outlets available at the ISC. Even though our power consumption was way too low to require both power supplies at the same time, we had two 2200W power supplies and never drew more than 1600W from the outlet the system always throttled without both couplers connected. Fortunately, we were able to order two adapter cables which, thanks to express delivery, arrived the very next day early in the morning. So this wasn't a real issue except it didn't allow us to do the necessary testing on the first day.

2.4.3 GPU topology

When we tried bench-marking our GPUs with microbenchmarks we observed very poor scaling across our two nodes. According to our contacts at NVIDIA, this was due to our hardware configuration or rather topology. In our initial setup, each GPU was connected to a different NUMA node as shown in Figure 2.2. Since there was a problem within the UCX library used by the HPL binary that had problems with communication with a NIC across a NUMA node this was most likely the cause of the problem. They recommended



(a) Two NUMA nodes.



(b) Single NUMA node (used).

Figure 2.3: Possible topology configurations.

bringing all the GPUs into one system, preferably into one single NUMA node. Because our mainboards did not allow all GPUs with full bandwidth within one single NUMA node, we had a decision to make. We could either put all four GPUs into one NUMA node or split them up into two NUMA nodes on one node, but both configurations are sub-optimal.

If we used booth NUMA Nodes in one node, as shown in Figure 2.3, then we would still have some communication overhead that would likely be not an issue because the problem was supposedly only apparent with NIC communications.

But putting all four GPUs into one NUMA node wasn't optimal either.

In our servers, the GPUs weren't connected to the CPUs directly but through a PLX Chip that splits the one PCIe Slot into two PCIe slots that then share their bandwidth. Given that these PLX chips only run at PCIe 3.0 speeds we're already halving the effective bandwidth of the GPUs. So the two options were either still having to deal with NUMA Overhead or halving the worst-case PCIe bandwidth of all the GPUs.

All of this happened right at the competition so our time was very limited and we could not try out booth options. On the recommendation of our NVIDIA contact, we decided to put all the GPUs into one single NUMA node which gave us acceptable scaling in the benchmarks.

2.4.4 CPU

Later in the competition, when it came to benchmarking the scientific application that we prepared and optimized, we noticed some problems with scaling and performance across the nodes even sockets on one of the systems. After some investigation, we found out that we had issues with the boosting behavior in one of both nodes. We don't know why but all boosting was disabled by the operating system on one of both nodes and it kept turning off when we tried to re-enable it. This improved the performance by quite a lot on this single node but on both nodes, the performance was still very poor. We found out that this was due to one of the CPUs. For some reason, this one CPU ran a lot slower, about 400 to 900MHz for no apparent reason. According to Atos, it was most likely a hardware defect within the CPU. We disabled all boosting options on both nodes, limiting the clock speeds to 2 GHz. Even though single node performance drastically decreased we were able to scale beyond a single node, albeit with rather poor performance.

Through the rest of the competition, we encountered some more problems with the CPU configurations. As an example, the link speed between both CPU sockets was set to the lowest possible setting which drastically reduced inter-process communication speeds. The bandwidth between both sockets was even lower than the Infiniband connection between both nodes. Generally speaking, there are many options pre-configured in the BIOS that were not optimal for our use case and configured to work with an older CPU generation. Moreover, the BIOS version was outdated and a BIOS update would have solved most of our problems with these systems. At least that's what Gigabyte suggested. Given that time was very limited and we weren't allowed to perform this update, which

in itself comes with some risks, we did not update the BIOS.

Even though we had a lot of problems and spent way more time debugging these issues than actually working on the applications we learned a lot in the few days. If everything went just fine we would have never learned as much about the systems as we did now. Thanks to the help, especially from a technician from Gigabyte, who spent hours of his valuable time at our booth helping us to rebuild the nodes, we were still able to at least produce some results to participate in the competition. Most problems or rather defects were just unlucky which very annoying because our work on the applications worked very well. Most other problems wouldn't be a problem if we had more time for preparation.

Retrospective, our cluster turned out to be rather sub-optimal for the tasks and applications in the competition. First of all the nodes are built very compact and optimized for compute density rather than efficiency. Putting 128 processing cores and up to 8 high-power GPUs into a small 2U chassis is a challenge for the cooling system. Even though we hit the power target of 3kW and surpassed it, this was only possible due to the cooling fans. These fans used about 800W of power when running at 100% meaning that we wasted almost a third of our power budget on the cooling system. Since there is no air conditioning in booths running the fans 100% load was necessary for long-running applications. A physically larger enclosure would have required a lot less power for the cooling fans. Moreover using only two nodes naturally comes with advantages over using more nodes: All the data is close together, and the connection speed between the nodes is as fast as possible with minimal latencies. But in our set of applications more nodes and more CPUs would have improved the overall performance. This year there was not a single application, besides the microbenchmarks, that could even run on a GPU. Therefore the GPUs didn't do anything once the micro benchmarks finished. We even saw super linear speedup across multiple nodes in the online competition. A cluster with four nodes without GPUs would have been better suited for this year's software selection. But that's something we could not have known before we received our cluster.

3 Online Competition Clusters

Authors: Lukas Schulte, Christian Willner

For the competition each team had to build, optimize and run the applications on each of the three provided clusters. Each with a unique architecture. The University of Toronto provided access to their SciNet Niagara supercomputer, an Intel based cluster, described in Section 3.1. The Pittsburgh Supercomputing Center opened up their Bridges-2 (see Section 3.3) nodes, which have AMD processors installed. The third cluster was provided by the HPC-AI Advisory Council itself. Their HPC unit is called Thor (Section 3.2) and has NVIDIA BlueField-2 cards installed, that can be used for the SCC.

3.1 Niagara

Niagara, one of Canada’s most powerful supercomputers, is installed at the University of Toronto and available for the participants of the online-part of the SCC. It was officially launched in March, 2018 and expanded in March of 2020. According to the Top500 list, a list of the most powerful supercomputers worldwide¹, in June 2020, Niagara is at place 140 with 80640 cores and draws 919 *kW* of power at its R_{peak} of 6.25 *PFlops/s*.

It is a homogeneous system based on Intel Xeon Gold 6248 20C Processors at 2.5 *Ghz*, connected with InfiniBand HDR100, build by Lenovo. Each of the 2024 nodes has 202 *GB* of RAM.

The installed operating system is CentOS7.6 and the queue submissions are managed by Slurm. Each job can be scheduled from 15 minutes to 24 hours. Larger jobs are favored over short ones.

3.2 Thor

The HPC-AI Thor system is an Intel based cluster, just like Niagara. It has 36 nodes, each with a Xeon Broadwell CPU and 256 *GB* of RAM. It even uses the same InfiniBand HDR100 adapters.

An interesting aspect of this setup are the NVIDIA BlueField-2 HDR100 cards. It basically works as an autark ARM powered node. These data processing units are running Linux

¹www.top500.org

and have their own InfiniBand and Ethernet interfaces. They are supposed to increase the overlap in communication and computation. Ideally this is done with the proprietary MPI implementation of the vendor to automatically use the potential of the DPU.

The added amount of communication that is necessary for the extra steps, can only be offset at a certain buffer threshold. The applications have hence to be optimized to make use of bigger communication buffers than they might have traditionally implemented. This plays an important role in the coding challenge (see Section 7.2).

3.3 Bridges-2

To bring in some variation, a non-Intel cluster is also used for the digital part of the SCC. The Pittsburgh based Bridges-2 cluster is a heterogeneous system with three different node types, for different purposes. An easy comparison is given in the following Table 3.1. All nodes are connected with HDR InfiniBand 200 *Gb/s* adapters.

# Nodes	Type	RAM	Usage
488	AMD EPYC 7742	256 GB	Data analytics
24	NVIDIA Tesla V100	512 GB	Deep learning
4	Intel Xeon 8260M	4 TB	Genome sequence assembly

Table 3.1: Different types of Bridges-2 nodes.

4 Microbenchmarks

Authors: Lukas Schulte, Niclas Schroeter

4.1 HPL

HPL is a portable and parallel implementation of the High-Performance Linpack benchmark. It is a measure of a computer's double precision (64) floating point rate of execution. The first published version dates back to 1979 which makes it probably the oldest benchmark still in use today. This makes it particularly interesting for performance comparisons across generations of computer systems.

It is widely used and the benchmark was chosen for the Top500 list.

The benchmark used in the HPL package is to solve a dense system of linear equations. The specifications allow for optimizations considering the problem size and exact implementation to achieve the best possible performance for a given system. To meet the specifications for the Top500 list the benchmark must use a LU factorization with partial pivoting. Moreover, the algorithm must be in $2/3n^3 + O(n^2)$ double precision floating point operations.

Since this benchmark is used to determine the rank within the Top500 list, manufacturers of CPUs and GPUs have a huge interest in tuning this benchmark for their own hardware. Because our cluster had a total of four GPUs with theoretical R_{peak} of 9.7 TFLOPs, tuning for the GPUs was way more beneficial than tuning for the CPUs, with a theoretical peak performance of about 2 TFLOPs each.

Fortunately NVIDIA provides precompiled binaries packaged into a docker/enroot container with all optimizations pre-applied. They even provide optimized input parameters for our exact system configuration (with four GPUs in a dual socket system with two EPYC CPUs) that we then used. We tried playing around with some parameters and even compiled our own binary with a CUDA enabled BLAS library, but to no one's surprise, the NVIDIA optimized version turned out to be the best one for our system.

Using the NVIDIA containers (version 21.4) and their sample input configurations we were able to achieve a result of 33.4 TFLOPS. Given the theoretical Peak performance of about 43 TFLOPS of a single node, that's about 78% of the theoretical Peak performance. When testing with only two GPUs in a single node, we achieved about 20 TFLOPS. Therefore we think our topology, as discussed in 2.4.3, might have limited scaling across more GPUs.

4.2 HPCG

The high performance conjugate gradient (HPCG) benchmark was proposed in 2013 as a means of better representing the computation and data access patterns used in actual applications [DH13]. The goal was to introduce a more realistic metric to rank system performance, compared to HPL, which only represents a fraction of the applications that are run in the context of HPC. To this extent, HPCG implements the preconditioned conjugate gradient method with a local symmetric Gauss-Seidel preconditioner. This involves the usage of different computational patterns, such as sparse matrix-vector multiplications, dot products or local triangular solves.

As with HPL, we used the HPCG implementation provided as a container (v21.4) by NVIDIA to make use of the GPUs in our system. After installing all the necessary software on our nodes and moving the hardware around, the only tuning parameter that is left when running the HPCG container is the input file. Unfortunately, due to the problems with the short preparation time before the competition and the hardware problems during the competition, we did not have much time to test different inputs, so the final run was conducted using 256 in all input dimensions, resulting in a final performance of 0.75 TFLOPS, which was also below our expectations.

4.3 HPCC

The HPC Challenge (HPCC) is a benchmark suite, initially proposed to augment the Top500 list by including multiple other benchmarks alongside HPL to better represent the actual applications run on HPC systems [LDK⁺05]. There are seven benchmarks present in the suite: HPL, STREAM, RandomAccess, PTRANS, FFT, DGEMM and b_eff Latency/Bandwidth. These benchmarks cover different areas of performance, such as processor performance in the case of HPL or DGEMM, communication performance in the case of PTRANS and memory system performance in the case of STREAM [Wic05].

Since NVIDIA does not provide a container for HPCC, we opted to utilize the version available in Spack, namely HPCC v1.5.0, having to forgo the utilization of our GPUs in this case. Regarding performance tuning, there are multiple parameters that influence the final results. First, HPCC has MPI and BLAS dependencies, meaning that multiple different implementations should be tested per dependency. Once proper implementations are found, the next tuning parameter is the input file, which is very similar to the HPL input, with four extra lines to supply different parameters to PTRANS, if deemed necessary.

In order to investigate the impact of the different implementations for the necessary dependencies, we created multiple scripts to run HPCC with different build configurations, using a selection of the benchmark results to evaluate the performance over multiple executions of the benchmark, reporting the results as a mean and accompanying standard deviation. A sample output is provided in Table 4.1. Since the reported metrics in

	MVAPICH2	HPC-X	MPICH
hpl_tflops	1.7524 (0.0067)	1.7193 (0.0246)	1.6069 (0.0636)
stardgemm_gflops	21.3268 (0.0706)	22.5267 (0.0249)	22.6012 (0.0545)
ptrans_gbs	13.1767 (0.3478)	17.2075 (0.1914)	2.9819 (0.1008)
mpifft_gflops	54.8982 (1.4831)	61.9185 (2.6011)	44.9419 (0.7795)
avg_ping_pong _latency_usec	1.7572 (0.0055)	0.6021 (0.0022)	1.1762 (0.0063)
avg_ping_pong _bandwidth_gbytes	14.9639 (0.0952)	13.2703 (0.0410)	5.5157 (0.0539)

Table 4.1: Benchmark outputs for HPCC using different MPI implementations.

HPCC are very numerous, we had to restrict our evaluations to a smaller subset. We chose different metrics for the evaluation of the different tuning options in such a way that we should be able to assess the overall impact of the different implementations, as seen in Table 4.1 for MPI. Using these scripts, we made measurements to decide on the best MPI, BLAS and also a potential FFT implementation. It should be noted that we had to conduct these initial evaluations on a single node once again, due to the availability issues before the competition. During these evaluations, we concluded that HPC-X would be the optimal MPI implementation, while OpenBLAS would be the optimal BLAS implementation. Based on the results from these scripts, we also decided to stick to the internal FFT implementation of HPCC instead of using FFTW2 or Intel MKL.

Unfortunately, we were unable to investigate whether or not these observations transfer to the complete cluster setup with two nodes, while also not being able to tune the input file for optimal performance. Due to the hardware problems described in Section 2.4 on the first day of the competition, we were only able to submit runs for HPL and HPCG within the time limit, not leaving any spare time for further examinations or even a single run of HPCC.

5 ICON

Authors: Niclas Schroeter, Frederic Voigt

The name ICON stems from ICOSahedral Nonhydrostatic general circulation model. It's an earth system to model a global numerical weather forecast, as well as a climate model. The icosahedral part of the name hints at the grid type of the application, which offers a nearly homogeneous coverage of the globe and avoids the pole-problem in lat-long grids.

First the sphere is divided in 20 equilateral spherical triangles, that can be further subdivided via bi- or trisections. These subdivisions can be localized to improve the prediction in certain areas. The location of averaged scalar model variables, like temperature, moisture or air density, are located on the circumcenter, while the wind components are modeled onto the midpoints. The different levels of the system allow the differentiation of water, snow or earth surfaces to increase the prediction accuracy.

ICON has many different dependencies and parameters that allow for performance tuning. The following section showcases the general workflow while tuning ICON. The first section will describe the tuning with compilers and dependencies, followed by ICON-specific runtime options in the second section. Since the tasks were near identical for both the online and the on-site competition, the workflow described in the following sections was applied to both competitions.

Tasks

The task for ICON was to simulate a whole year as a coupled atmosphere-ocean-experiment. To do this, we were provided with weather input data of the year 1850. The goal was to run the experiment as fast as possible. There were two additional conditions: The first one was to stay within a 30 minute time limit, while the second one was to not deviate too far from a given reference result. To check the second constraint, the correctness, there were visualization options in the form of post-processing scripts. A secondary task was the tool-based examination of the program run with the profiler IPM. The delivered results should be interpreted afterwards and the main bottlenecks of the application should be described further.

5.1 Building and configuration

To build ICON it was first necessary to log in to DKRZ Git to get access to the code and the submodules. The actual configuration process is based on a prebuilt script. To configure ICON, first the dependencies are loaded and sourced. They are all available in Spack, which was used for these operations. Most of the dependencies will be examined further in the next section as they gave us options for optimization, barring some dependencies like Python, which was used for post-processing, or CDO. The next steps consist of the usual build procedures, i.e. setting the compiler flags and other variables, followed by *make*. The use of OpenMP and vectorization, which will be discussed further below, also had to be enabled in the configuration step. The differences of the configuration step on Bridges-2, Niagara and our on-site cluster are mainly limited to the used compiler and some other dependencies, which needs to be reflected in the utilized flags as well, but the overall usage remained similar.

Throughout the build and configuration process, we repeatedly encountered bugs and problems that could be attributed to peculiarities of ICON. For example, some paths in the scripts are designed for Mistral, on which ICON was originally designed. Other problems, such as the use of MPICH and OpenBLAS, are explained in more detail below.

5.2 Compilers and dependencies

The first tuning option is the choice of compiler. For this purpose, we opted to use either the Intel compiler suite or GCC, since both of these compiler collections are well established. The choice was heavily influenced by the underlying hardware. For example, our tests on the Niagara cluster, which is equipped with Intel CPUs, have shown that the Intel compilers produce a better optimized executable than the GCC compilers when running on Intel hardware. The executable built with Intel compilers required slightly less than 20 minutes to simulate the entire year without further tuning, whereas the executable built with GCC needed more than 30 minutes. However on non-Intel hardware, this difference in performance vanishes, leaving the GCC-built executable ahead of the Intel pendant.

For all following evaluations, the runtimes were compared on a reduced version of the experiment, only simulating 3 months instead of one year, due to limitations on computing resources and time constraints. Due to this limitations, we were only able to run each experiment once, which harms the statistical significance of our results. Though it should be mentioned that whenever we had to repeat experiments due to perceived irregularities, we noticed a rather small variance between the different runs.

After choosing the best compiler for the given hardware, the next major dependency that influences the runtime is the MPI implementation. For all available clusters, we investigated three different MPI implementations: MPICH, MVAPICH2 and NVIDIA HPC-X, which utilizes OpenMPI. We built ICON once for every MPI implementation

and then compared the different runtimes on the reduced simulation. Table 5.1 shows examples of the resulting runtimes during the online competition. Note that MPICH performed far worse than all other implementations and was not even able to compile on Niagara. This considerably longer runtime can most likely be attributed to problems with the collective operations that are necessary after each time step of calculations in ICON. We noticed that some of these collective operations took much longer than others, sometimes requiring multiple minutes instead of less than a second. This was reproducible on multiple runs of this particular experiment, though the exact timing of the longer lasting collectives was irregular.

	MPICH	MVAPICH2	HPC-X
Niagara	*	216s	192s
Bridges-2	605s	310s	280s

Table 5.1: Runtimes of ICON with different MPI implementations.

The next dependency that has a large impact on ICON’s performance is the BLAS library. Once again we opted to evaluate the most established ones, namely Intel MKL and OpenBLAS. As with the compilers before, it appears that Intel MKL produces the best results whenever used in conjunction with Intel hardware. On AMD hardware though, these results were not reproducible. Example measurements are shown in Table 5.2. These measurements were taken while using the Intel compilers on Niagara and GCC on Bridges-2, both for compiling ICON and the dependencies. Note that the usage of OpenBLAS on Niagara led to compilation problems, so in order to still be able to compare them, we built ICON and the dependencies, including OpenBLAS and Intel MKL, with GCC instead and then compared the two different versions on a full simulation. Results indicate that OpenBLAS performs worse (runtime of 43 minutes) than Intel MKL (runtime of 40 minutes) when everything is built with GCC. We suspect that this difference would at least carry over if OpenBLAS could have been built with the Intel compilers as well, though it will probably grow wider, given our observations of the performance of Intel software and hardware in combination.

	MKL	OpenBLAS
Niagara	192s	*
Bridges-2	325s	280s

Table 5.2: Runtimes of ICON with different BLAS implementations.

The last dependency that we experimented with for potential performance gains was NetCDF, particularly the parallel variant PnetCDF. We suspect that building the NetCDF dependency with PnetCDF would have improved our runtime further, since this would have enabled a parallel I/O option that was potentially available during the configuration step for ICON, but we ran into compilation problems whenever we used PnetCDF. One of the ICON submodules would fail to compile on every cluster and we

were unfortunately not able to resolve this problem, which is why we had to refrain from using it for any of our final configurations.

The final tuning options regarding the compiler and dependencies are the compiler options. Per default, the ICON build script already utilized the `-O3` flag for the most part alongside other optimization flags, leaving `-ffast-math` or the Intel pendant `-fp-model fast=2` off. In normal usage scenarios, this would be recommended. However, since we had the option to check our results for correctness, with the goal of producing the fastest execution times possible, we decided to enable the fast math option in all our runs. For the online competition, this further reduced our runtimes on both clusters, without harming the correctness, therefore giving us an important advantage. In the on-site competition, there was also a reduced runtime, unfortunately though, the fast math optimizations distorted the precision of the results so that they were outside the tolerance range for correct results.

5.3 Runtime parameters

After investigating the impact of the different compilers and implementations of dependencies, the next options to increase the performance of ICON are the different runtime parameters, beginning with the distribution of the Ocean processes. Since the task for the competition is a coupled ocean atmosphere experiment, processes need to be assigned to either the Ocean component or the Atmosphere component. The setup that is initially used, without further modifications, assigns three quarters of the processes to the Atmosphere component and the last quarter to the Ocean component, while also noting that this distribution designates too many processes to the Ocean component. In order to find a better distribution, we conducted two experiments per cluster, the first one being a coarse evaluation with large step size, while the second one was fine-grained to find the optimal distribution around the previous optimum from the coarse evaluation. An example is provided in Figure 5.1, where we tested the different distributions on Niagara.

The next runtime parameter is the *nproma* value. This constant specifies the blocking length for array dimensioning and inner loop lengths, the arrays in ICON are also organized in sizes using this constant. A block of *nproma* size can be processed as one vector by vector processors or by one thread in a threaded environment. It is also the length of the innermost loops and it can be used for achieving better cache blocking. Due to these reasons, changing this constant can result in higher performance of the nodes during their calculations. Both the Ocean and the Atmosphere component have their own *nproma* parameter in the ICON scripts, though we decided to set them to the same value, as both components utilize the same hardware. As with the distribution of the Ocean processes, we first conducted a coarse-grained experiment to then limit the following fine-grained experiment to the range of the previous best results. The example provided in Figure 5.2 was also conducted on Niagara.

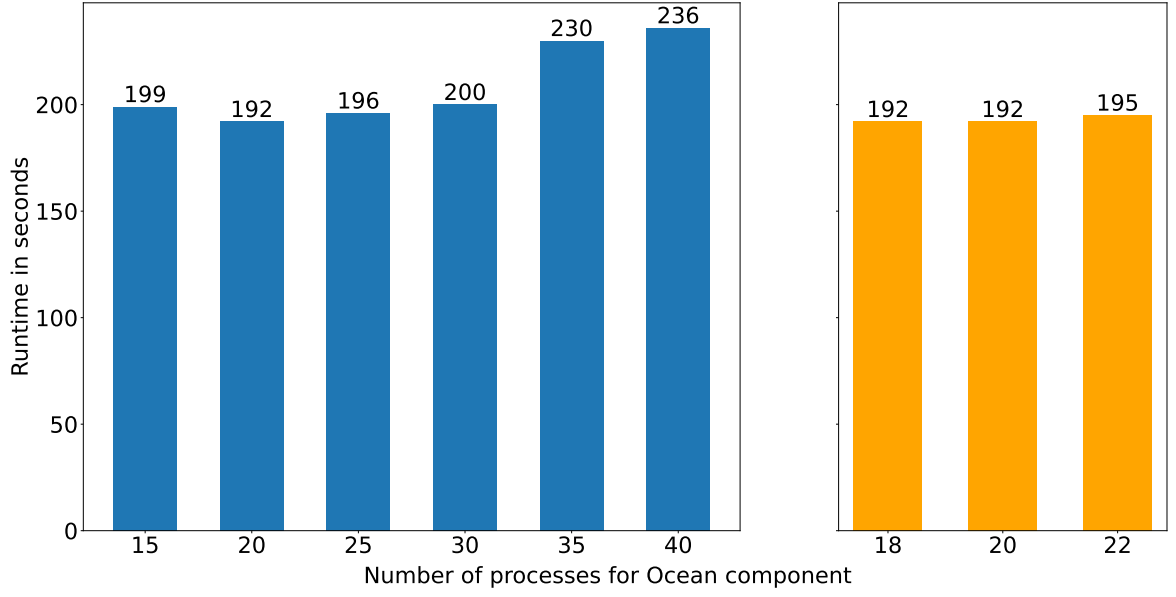


Figure 5.1: Different distributions of the Ocean processes on Niagara. The whole experiment utilized four nodes of 40 processes each. The bars represent one experiment each, with the labels referring to the total number of processes assigned to the Ocean component. The left figure corresponds to the initial coarse evaluation, the right shows the fine-grained one.

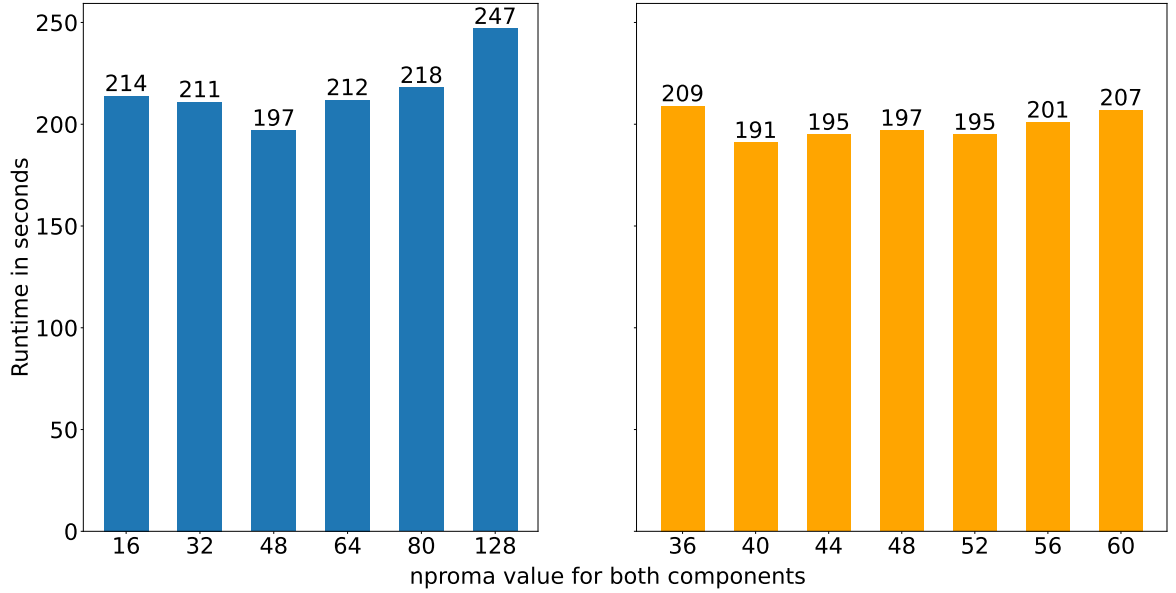


Figure 5.2: Different values for *nproma* on Niagara. The left figure corresponds to the initial coarse evaluation, the right shows the fine-grained one.

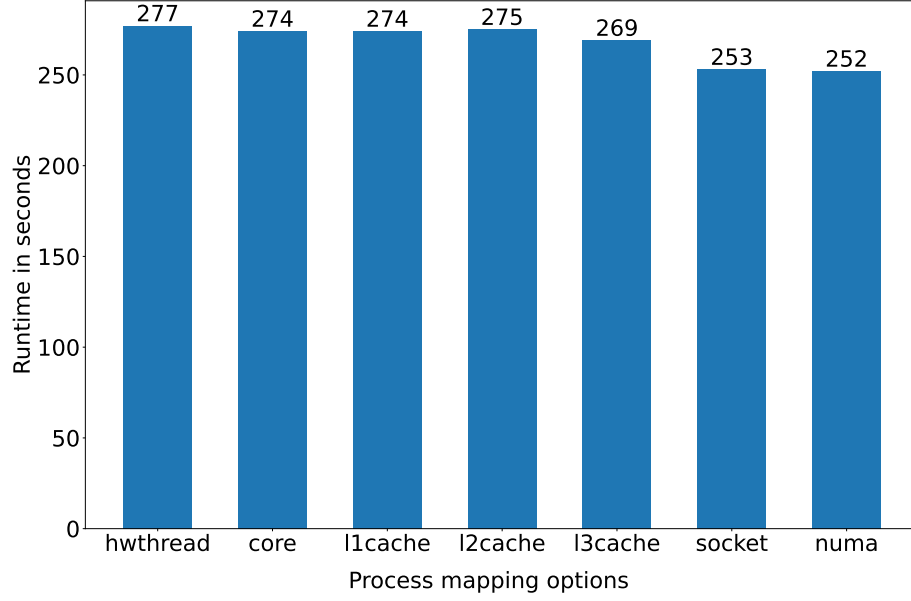


Figure 5.3: Different process mapping options, tested on Bridges-2.

The next option that we investigated was process placement. Given that most processes in ICON communicate mainly with their neighbors or via collective communications, proper process placement can further decrease the runtime. Therefore we used the *map-by* parameter provided by *mpiexec* to test different mappings on our cluster and Bridges-2. On Niagara, the *map-by* parameter did not work properly, so we instead utilized the *KMP_AFFINITY* environment variable to control the mappings for the same effect. The example in Figure 5.3 shows the different mapping options and their effects when tested on Bridges-2.

The last tuning option that we examined was multithreading. ICON is written with hybrid parallelization, enabling the usage of OpenMP alongside MPI. Once enabled during compilation, the number of threads per process can be controlled via the ICON scripts. On all clusters, we tested different configurations, ranging from one thread per process (the default, equal to no multithreading) to single-digit processes per node, while the remaining resources were used as threads. We found that whenever we enabled multithreading, the performance on this particular experiment decreased considerably. Changes to the corresponding environment variables that influence the performance in those scenarios did not alleviate this problem either. Due to this, we decided to forgo the usage of multithreading in all of our final configurations.

5.4 Final configurations for the online competition

After going through the tests described in the previous section, we combined everything to one final configuration per cluster. On Niagara, we utilized the Intel compiler suite, Intel

MKL as the BLAS library of our choice, alongside HPC-X as the MPI implementation. Of the 160 processes in total, 20 were assigned to the Ocean component, with the process mapping option of *tile* passed to *KMP_AFFINITY*. The optimal *nproma* value on this processor architecture appeared to be 40 in this particular case. With the aforementioned fast math options enabled as well, we managed to produce a final runtime of 12 minutes and 8 seconds.

On Bridges-2, the superior option for the compilers turned out to be GCC, while using OpenBLAS as the BLAS library. For the MPI implementation, HPC-X remained the best option. Of the overall 384 processes distributed across the four nodes for the competition, 32 were assigned to the Ocean component, mapping the processes to NUMA domains via *map-by*. As for the *nproma* value, the optimal choice appeared to be 16. Once again enabling the fast math options during compilation, we finished with a runtime of 15 minutes and 15 seconds.

Combining these two results, we finished the ICON portion of the competition in first place, beating out all other teams in this category of the online competition.

5.5 Profiling

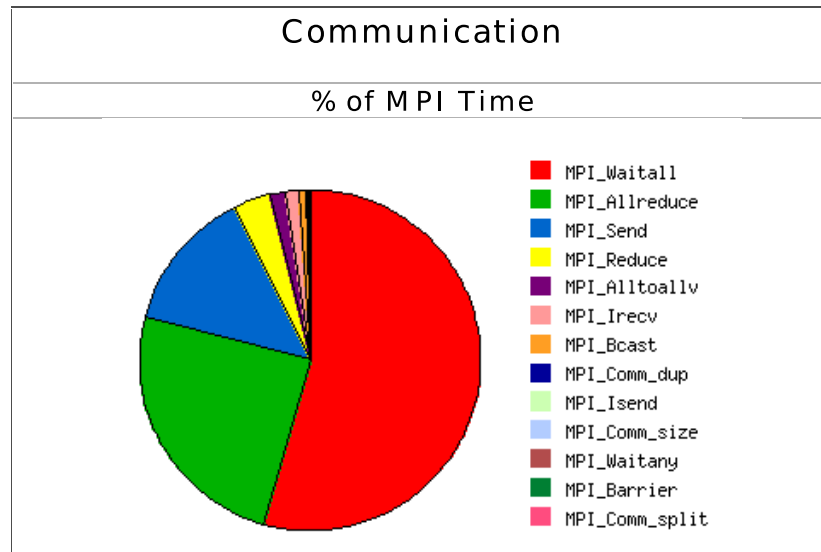


Figure 5.4: Shares of communication time for the different MPI calls within ICON. This figure was generated via IPM on Niagara.

One of the tasks was to create ICON profiles with IPM and interpret them, both on Bridges-2 and on Niagara. Since this analysis revealed an additional insight into the general behavior of ICON, we would like to roughly summarize it here.

On both clusters, the application spends about a quarter of its runtime in MPI communication. The call to *MPI_Wait* accounts for most of this time (over half on Niagara and

two-thirds on Bridges). Next is *MPI_Allreduce*, which is about a quarter of the runtime used for MPI calls. *MPI_Send* is third and is about a tenth of the time. An example of that is displayed in Figure 5.4, visualizing the percentage of MPI communication on Niagara.

The first bottleneck can be quickly determined, as the application spends most of its time waiting. Figure 5.5 provides further insight into potential causes, displaying the communication balance ordered by MPI rank. Ranks 140 to 159 were responsible for the Ocean component, the rest was appointed to the Atmosphere component. Due to the distributed computation for the atmosphere and the ocean, load balancing is an important aspect in ICON. In the atmospheric calculation about half of the communication time is spent waiting. In the ocean group, the waiting time is much more present and takes the lion's share of the communication. We suspect that these long waiting times stem from the synchronization between the two components. Apparently, the processes responsible for the ocean component finish their calculations and accompanying communication calls much faster than the atmosphere processes.

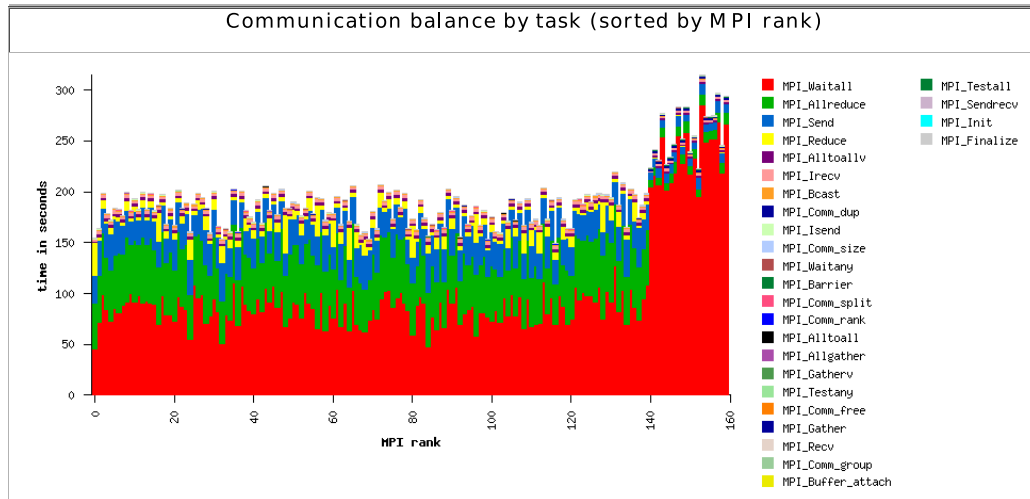


Figure 5.5: Communication balance of the individual processes, sorted by MPI ranks. This figure was also generated via IPM on Niagara.

The second bottleneck of the application is *MPI_Allreduce*. We suspect that it is used for synchronization and result exchange at the end of time steps within the simulation. It should also be noted that the overall time necessary to communicate, excluding any wait calls, is larger within the atmosphere component as a whole. It can be concluded that there is a severe load imbalance between the two components when working on the experiment that was used for this competition. Given that the distribution of Ocean and Atmosphere processes, that was used to generate these profiles, was the best performing one, we suspect that this load imbalance cannot be resolved by simply shifting the process distribution, but that it is rather an inherent problem with ICON itself or at the very least with the setup of the particular experiment for the competition.

5.6 On-site competition

For our final configuration of ICON on our cluster, we chose GCC as the compiler suite due to the AMD CPUs. For the dependencies, OpenBLAS and HPC-X performed the best. The best performing *nproma* value was 20. It should be noted that the tests to determine the best options had to be conducted on a single node of our final cluster setup due to the availability issues prior to the competition. On this setup, the optimal distribution of processes per component was 100 processes for the Atmosphere component and 28 for the Ocean component. On the single node, we achieved runtimes of slightly over 40 minutes, so from these initial tests, we estimated a runtime below 30 minutes for our complete setup with both nodes. However, due to the CPU problems described in Section 2.4, executing ICON on both nodes led to worse performance than having it compute on a single node, however the performance of the single node had also worsened compared to our initial tests. Due to these factors we ended the competition regarding ICON with a total runtime of 1 hour and 11 minutes, far beyond our initial estimations.

6 NWChem

Authors: Frederic Voigt, Johannes Wünsche

NWChem brands itself as an open source high-performance computational chemistry tool. It has a strong emphasis on its scaling capabilities and is hence a good candidate for the SCC.

The software is developed by EMSL at the Pacific Northwest National Laboratory in the state of Washington, USA since the 1990s. Scientists can use it to model kinetics and dynamics of solid-state materials, nanostructures and biomolecules. The source code is primarily written in Fortran and uses MPI as well as OpenMP. This method of parallelization is usually hidden by a global array programming model. These arrays can be scaled to be shared on distributed memory addresses [ABdJ⁺20].

Tasks

The first task was the processing of a total of three different inputs, a CCSD(T)-experiment, a TCE CCSD(T)-experiment and a DFT-experiment. These were to be performed on the basis of CPU nodes. In addition, one experiment should also be GPU based and the input data (except for the DFT experiment) could be selected from the previous ones. Like ICON (see Section 5), an analysis of the program run with the IPM profiler should be performed and a small analysis of the program behavior should be made.

6.1 Tuning

Since NWChem was available as a Spack package, building and compiling the application on both clusters was relatively easy at first. The challenge of tuning was to try out the different dependencies of the packages and to test their interactions. The focus was on the compiler, MPI and the packages scalapack, lapack, blas and FFTW. To test these configurations, we wrote scripts that tried most viable combinations.

Some combinations we were able to discard early on due to overlapping functionality, for example Intel-mkl implements a wide range of the previously named interfaces.

6.1.1 Scripts

The aforementioned scripts can be divided into two main functionalities. First, for the evaluation of the best performing *combination of libraries*, a standard case with a brief runtime, namely DFT, has been chosen to give an overview of estimates how well performing each combination is. From this, the best performing will be further evaluated to measure whether OpenMP (if available) can offer any improvement over the already tested work distribution. Referring to the documentation, one, two, four, and eight threads have been used here, as a greater number of threads is very likely to worsen performance in the long run. This has also been confirmed in our analysis, with a maximum of four threads delivering any performance gain.

6.1.2 Input Tuning

NWChem has shown to be sensitive to the tuning of the actual input it received and therefore we invested some time into assuring a well-tuned input file was available for all three benchmarks which needed to be evaluated.

Here focus was laid on three factors: feature utilization, memory usage and algorithm choice. Starting with feature utilization, input files needed to specify beyond OpenMP environments for which steps OpenMP should be used. For most cases it made sense and brought real performance gain, though some did not benefit from the distribution to threads rather to processes at all. Especially, one case produced a worsened runtime compared to using only processes, therefore OpenMP was disabled for this step. Furthermore, GPU utilization needed to be activated by specifying the host capabilities.

Next to input feature, memory availability and distribution needed to be specified, which turned out to be trial-and-error in many cases as internals of some computations require a certain amount of global/local memory available. We started from common memory bounds and ratios, extracted from multiple discussions in the *NWChem* forum, and modified them to fit to the available systems. Also problematic for this study was that shorter (ergo smaller cases) require less memory and different memory distributions, so a proper testing of runtime had to be performed on the full benchmark, which, depending on the benchmark, could take about two hours. In a less time restraint environment this is probably the point where a greater degree of optimization can be expected.

Lastly, we tested some algorithm variants and parameters to take the best choice runtime-wise. Since these algorithms are implementations of quantum-chemical processes with only numbers as their names, the organizers provided a selection, we were able to choose from, to produce the correct results. We measured all given variants and chose the best performing one. The parameters for them were identical, mostly relating to observed tile domain.

6.2 Results

The effects of the individual tested combinations of library implementations described in Section 6.1 can be seen for Bridges-2 in Figure 6.1 and for the Niagara cluster in Figure 6.2. The importance of the OMP threads, or more precisely the chosen number of threads, for the runtime of NWChem has already been discussed in Section 6.1.1 and can be viewed for the Bridges-2 cluster in Figure 6.3 and for the Niagara cluster in 6.4. Here, the weak performance of the application, in terms of strong-scaling, becomes very apparent.

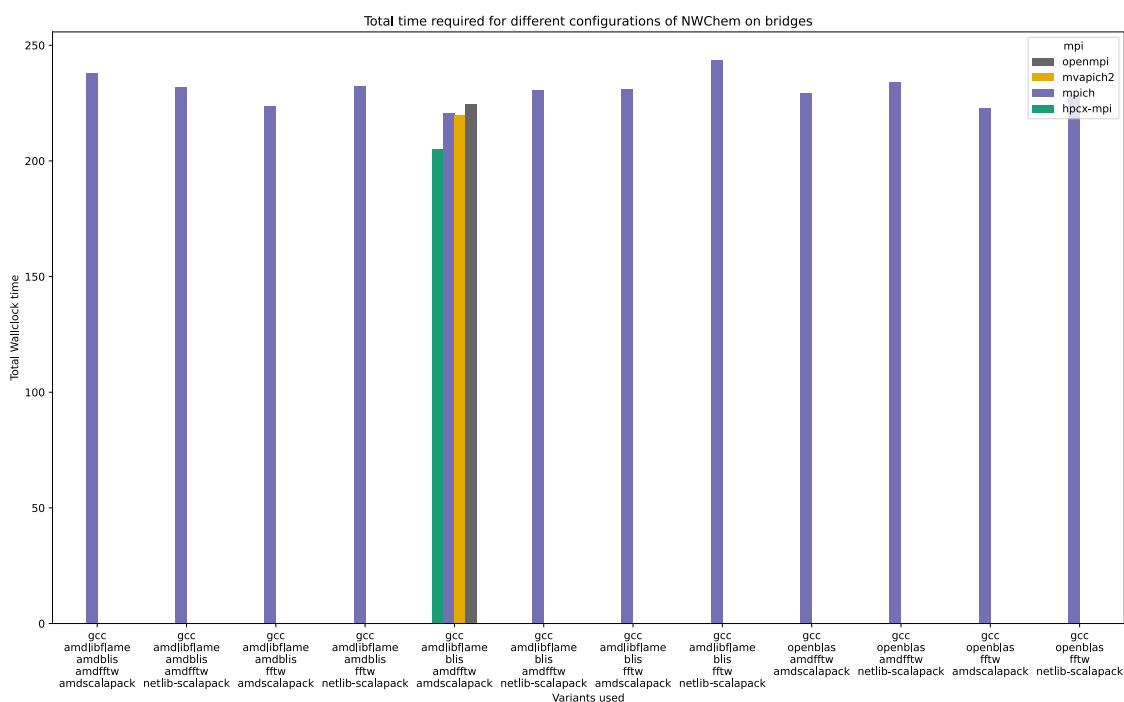


Figure 6.1: Runtime of NWChem for shifted combinations of the implementations of compiler, scalapack, lapack, blas and FFTW on the Bridges-2 cluster.

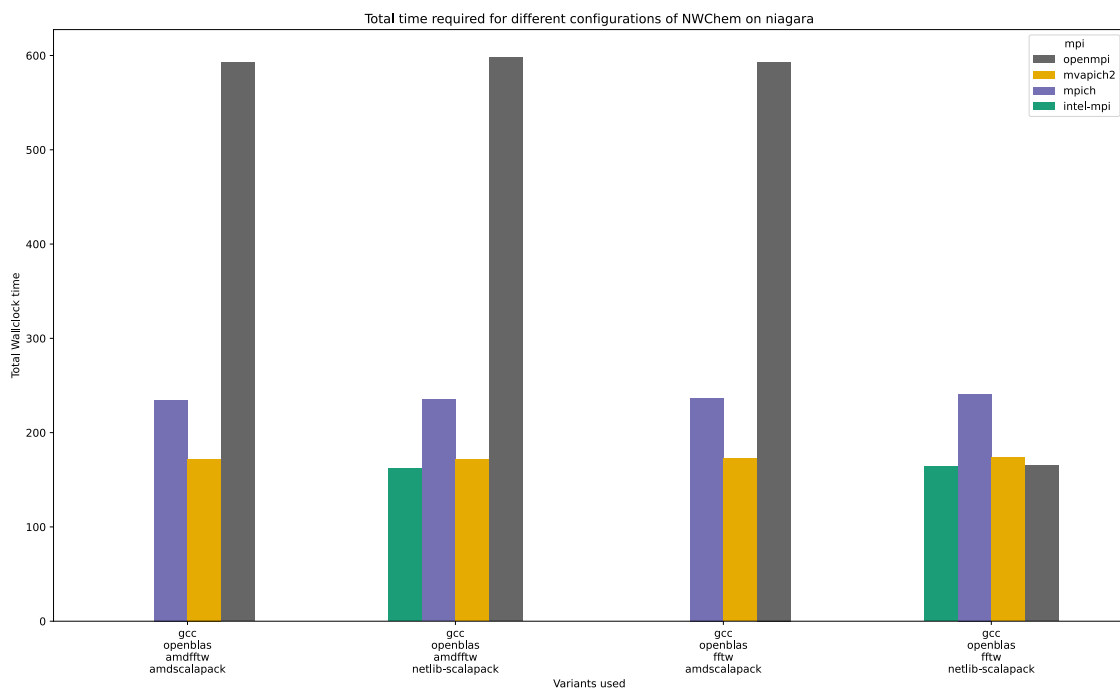


Figure 6.2: Runtime of NWChem for shifted combinations of the implementations of compiler, scalapack, lapack, blas and FFTW on the Niagara cluster.

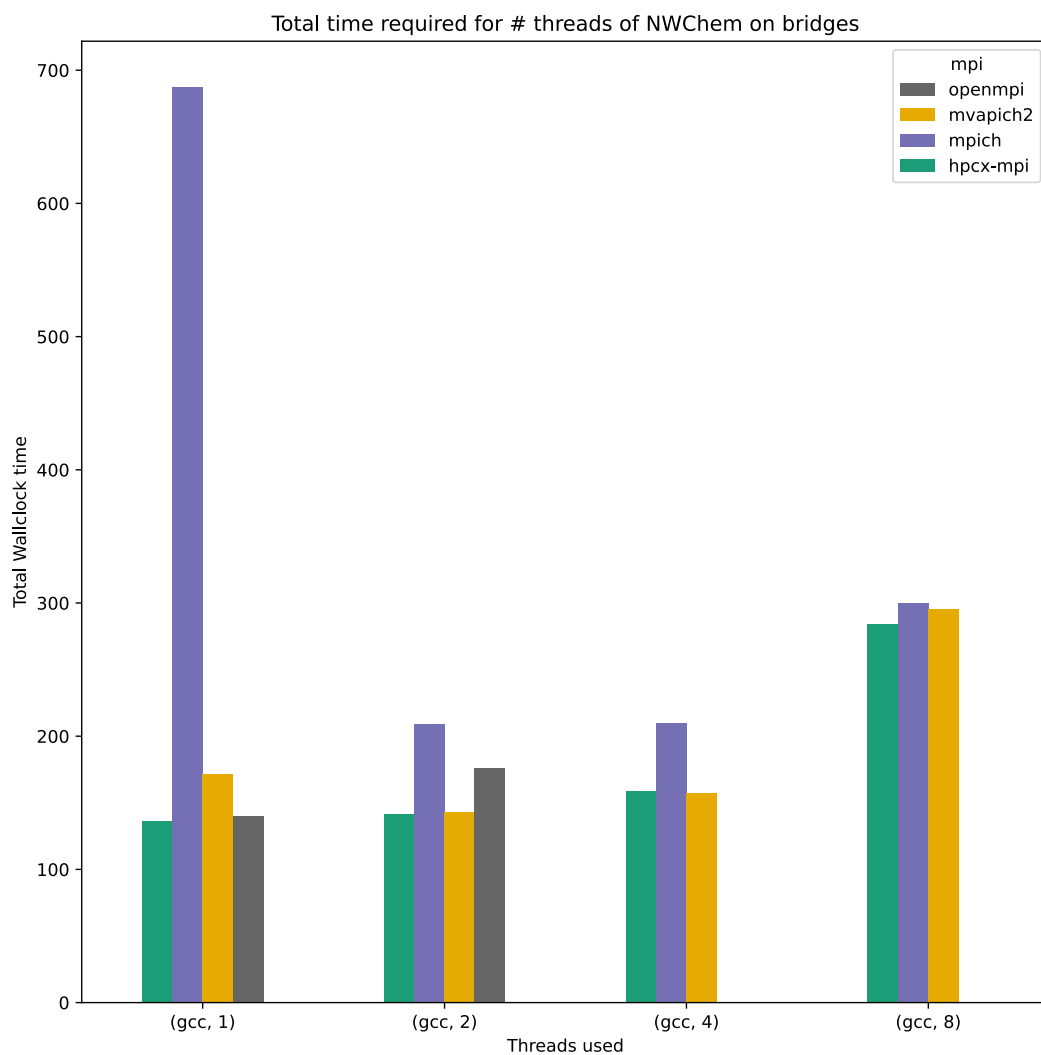


Figure 6.3: Runtime of NWChem for moved OMP thread count on the Bridges-2 cluster.

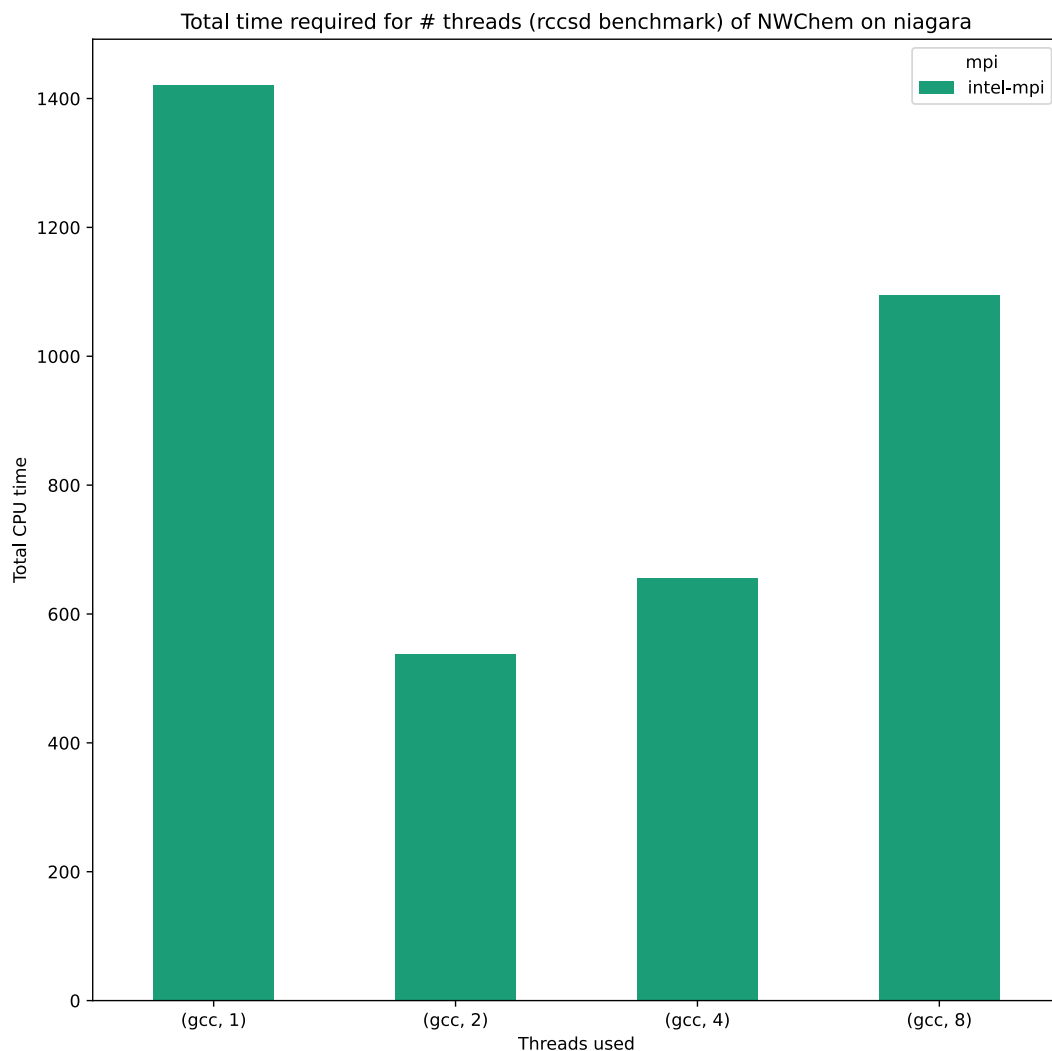


Figure 6.4: Runtime of NWChem for moved OMP thread count on the Niagara cluster.

The final achieved times for each challenge can be found in Table 6.1.

6.3 Profiling

NWChem was to be subjected to further analysis by using IPM. The analysis part was relatively simple and only the three most used MPI calls should be named: These were *MPI_Wait* in first place, *MPI_Recv* in second and *MPI_Isend* in third. A part of the

IPM analysis can be viewed in Figure 6.5.

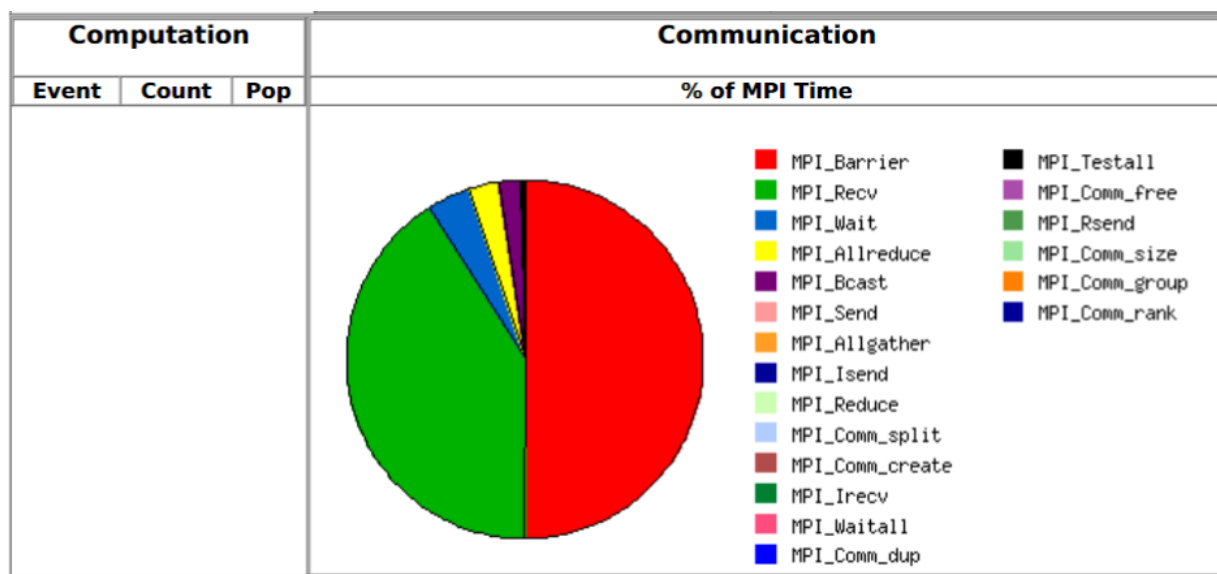


Figure 6.5: Distribution of call frequencies of MPI functions in NWChem according to the IPM profiling.

Challenge	Niagara	Bridges-2
RCCSD-T	3455.5s	2191.4s
TCE CCSD(T)	9729.2s	8856.0s
DFT	135.9s	161.8s
GPU		6002.1s

Table 6.1: Final results of the NWChem runs.

7 Xcompact3D

Authors: Christian Grüneberg, Christian Willner

The Application Xcompact3D is a high-performance framework for solving the Navier-Stokes equations and associated scalar transport equations and is based on Fortran90. It uses Direct and Large Eddy Simulations and combines the versatility of industrial codes with the accuracy of spectral codes. Currently it is able to solve the incompressible and low-Mach number variable density Navier-Stokes equations using sixth-order compact finite-difference schemes with a spectral-like accuracy on a monobloc Cartesian mesh. The development of Xcompact3D started in the mid-90's as Incompact3D for serial processors and only for incompressible flows. Over the time MPI support, simulation of compressible flows in the low Mach number limit and wind farm simulation were added and Incompact3D was renamed to Xcompact3D in 2019. Currently the developer are working on GPU support but the version for the competition only supported CPU's.

Xcompact3D achieves a good scalability due to the use of 2DECOMP library which allows a 2D pencil decomposition of the mesh, which allows to divide the mesh in 2 dimensions as shown in Figure 7.2. The benefit, in contrast to the previously used 1D decomposition as shown in Figure 7.1, is that the mesh can be divided between more processes at the expense of a more complex communication scheme between the processes. The consequence is that the runtime depends on the chosen division of the problem into rows and columns in order to distribute the calculations evenly among the processes. Also the chosen MPI library to build the application affects the runtime, as the actual communication could be performed differently or because of different performance of each MPI library for different message sizes.

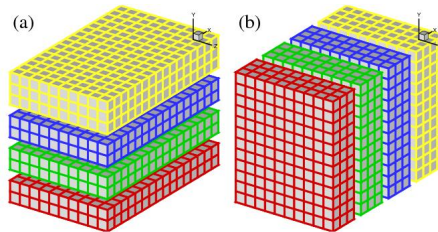


Figure 7.1: 1D domain decomposition example using 4 processors a) decomposed in y direction b) decomposed in x direction. Figure taken from [Li10].

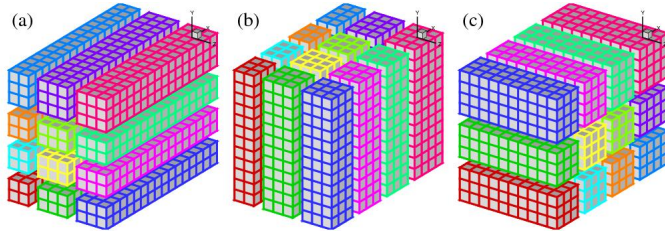


Figure 7.2: 2D domain decomposition example using 4x3 processor grid. Figure taken from [Li10].

Since there was no Spack package for Xcompact3D, we had to build one first. The Xcompact3D makefile supports gfortran or the Intel Fortran compiler. The default library for the Fast Fourier transformation is FFT which is built into 2DECOMP. Alternatively, FFTW or Intel MKL are also supported.

7.1 Online Competition

The tasks for the online competition were based on the simulation of the flow between two wind turbines with uniform incoming wind.

1. Profiling the simulation for 4 nodes on both clusters for 2500 iterations and submit two IPM profiles with comments on potential bottlenecks
2. Identify the best configuration for the fastest wall clock time for 2500 iterations on 4 nodes for both clusters
3. Generate 3D visualizations of the flow for 5000 iterations on a cluster of our choice with ParaView
4. Bonus Task: perform two strong scalability studies on both clusters using 1 to 8 nodes for 2500 iterations

For this we were given an input file where we could change 5 parameters. Firstly, *p_row* and *p_col* for the domain decomposition. Both variables multiplied should give the number of MPI processes for the application. If this is not the case, this leads to an error during the execution. The next parameter was *ilast* for the number of iterations, which is given for the tasks. The two last parameters were *icheckpoint* for writing checkpoints after a number of iterations in a backup file and *ioutput* for the frequency to create visualization data. Both parameters *icheckpoint* and *ioutput* were set to number higher than the number of iterations to not influence the performance. Only exception were the runs for the visualization of task 3.

For the choice of the compiler for each cluster we only considered GCC and the Intel compiler which were already defined in the makefile. To find out which compiler was the best for each cluster we took the input file for the wind turbine case and compared the runtime for versions of Xcompact3d built with different compilers. For the Niagara cluster, the Intel compiler was the better compiler, probably due to better optimization for the Intel CPU's used for the cluster. Instead, for Bridges-2 GCC achieved better results than the Intel compiler.

After we decided which compiler we used for each cluster, we wanted to find out which MPI library performs the best on each cluster. We considered OpenMPI, MPICH, MVAPICH2, HPC-X on both clusters and Intel-MPI only on Niagara. Furthermore, we considered that each MPI library could perform differently based on the domain decomposition, due to different implementations of the communication scheme. So we used the wind turbine case with just 100 iterations with 4 nodes, 512 processes on Bridges-2 and 320 processes on Niagara, with different combinations of *p_row* and *p_col* for each MPI library, as shown in Figure 7.3 and Figure 7.4.

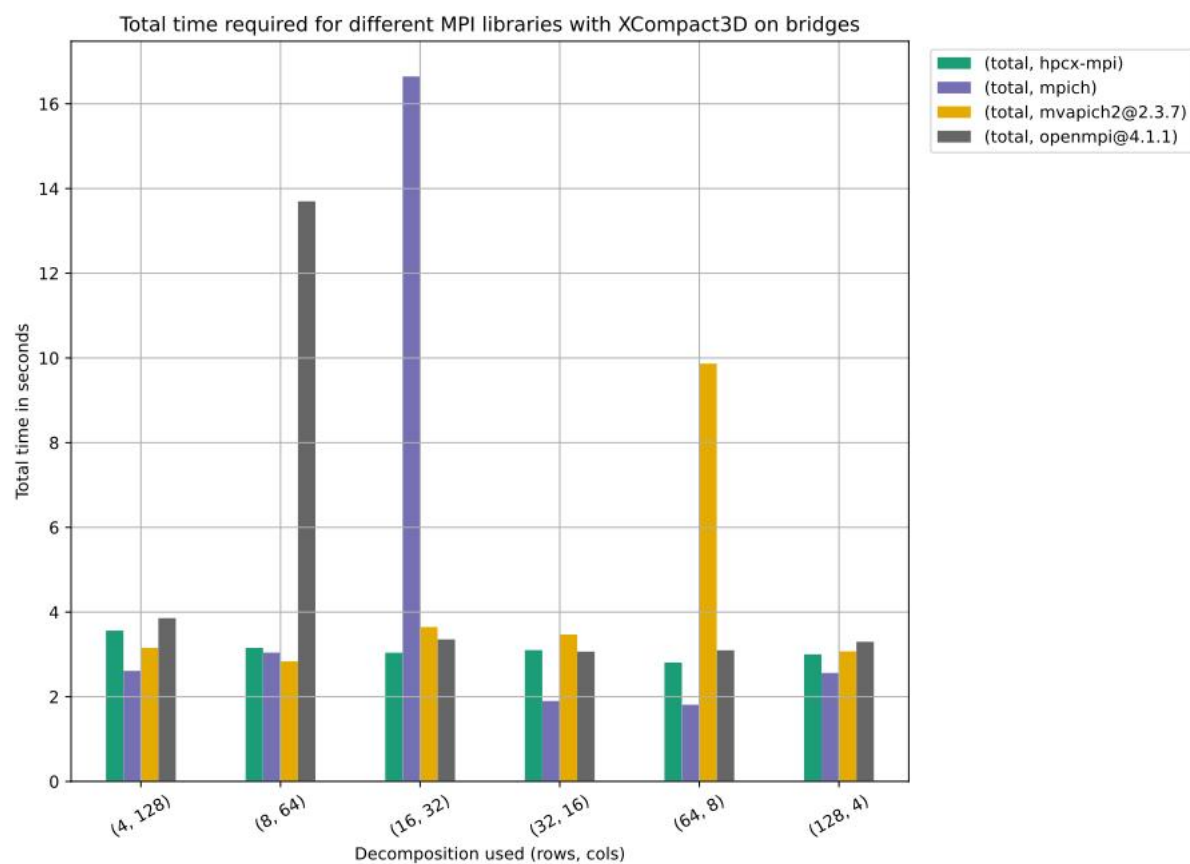


Figure 7.3: MPI comparison for Bridges-2.

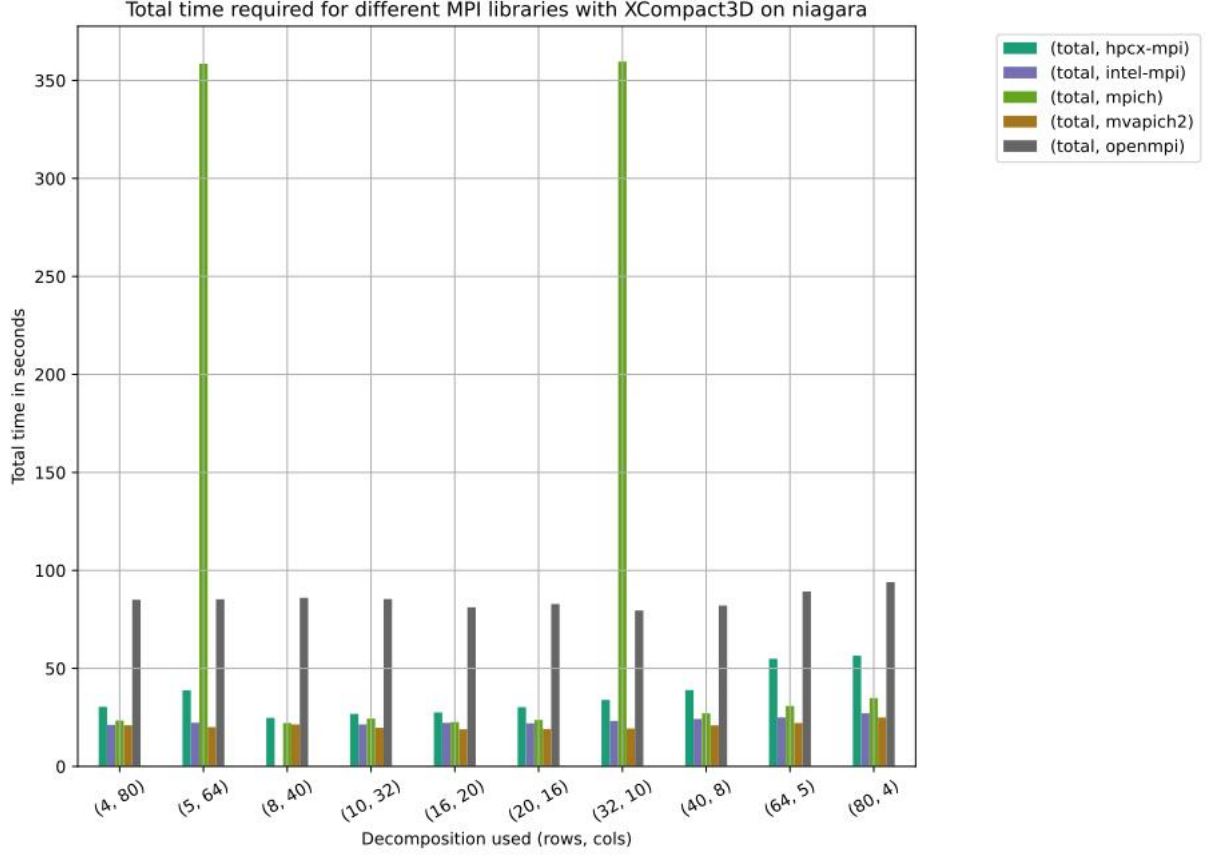


Figure 7.4: MPI comparison for Niagara.

The result was that for Bridges-2 MPICH and for Niagara MVAPICH2 were the best performing MPI libraries. Although both plots illustrate how important a well chosen domain decomposition is for the performance.

The last thing we considered to gain a better performance was the use of different FFT libraries. Instead of only relying on the default FFT implementation we used so far, we also considered FFTW on both cluster and additionally Intel-MKL on Niagara. Then again we used the wind turbine case and tested all the different decompositions with each FFT library and the best MPI task library for each cluster for 2500 iterations on 4 nodes to find the best configuration for task 2 on each cluster.

cluster	runtime in min	compiler	MPI	FFT	p_row	p_col
Bridges-2	16.44	GCC	MPICH	default FFT	8	64
Niagara	15.46	Intel	MVAPICH2	FFTW	16	20

Table 7.1: Best runs on each cluster with the used compiler and libraries.

Remarkably is that Niagara is still faster than Bridges-2 despite the fewer process count

for 4 nodes, 512 to 320. This is probably due to the AVX-512 support of the Intel CPU's used for the Niagara cluster, which the AMD CPU's on Bridges-2 do not support.

To profile Xcompact3D on each cluster for task 1, we had to use IPM. Since HPC-X is already build with IPM support we decided to use Xcompact3D built with HPC-X instead of the best performing MPI library for each cluster. However the HPC-X we used on Niagara for the runs for task 2 were not built with IPM support and so we had to recompile with a different HPC-X version which was only possible with GCC, not with the Intel compiler.

cluster	compiler	MPI	FFT
Bridges-2	GCC	MPICH	default FFT
Niagara	GCC	MPAPICH2	default FFT

Table 7.2: Used compiler and libraries for profiling each cluster.

We created profiles for different decompositions on each cluster. Overall, the time spent in MPI functions and the used memory is greater for Bridges-2 compared to Niagara, probably due to the higher process count.

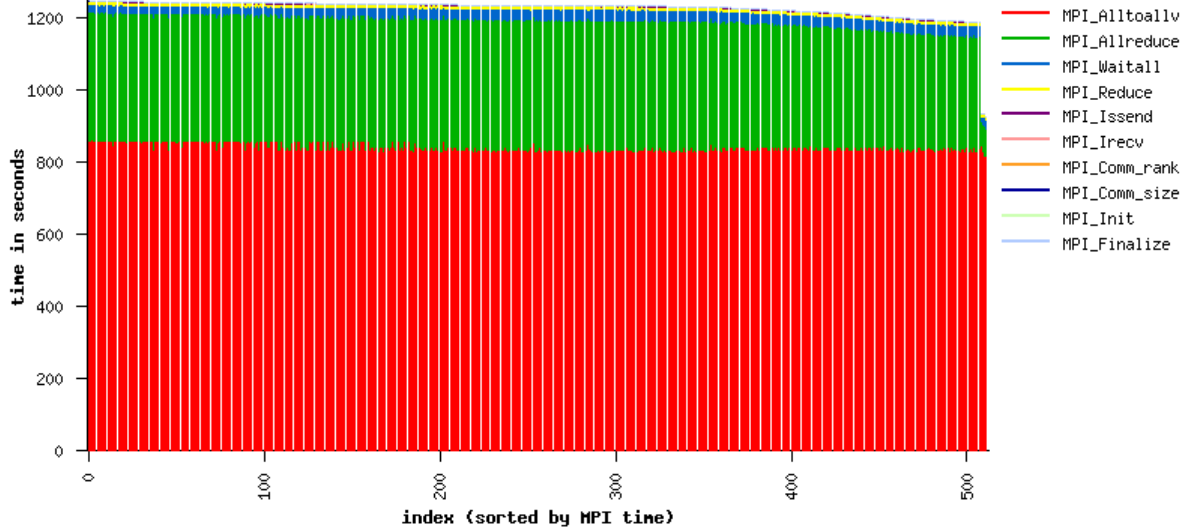


Figure 7.5: Distribution MPI time task wise on Bridges-2.

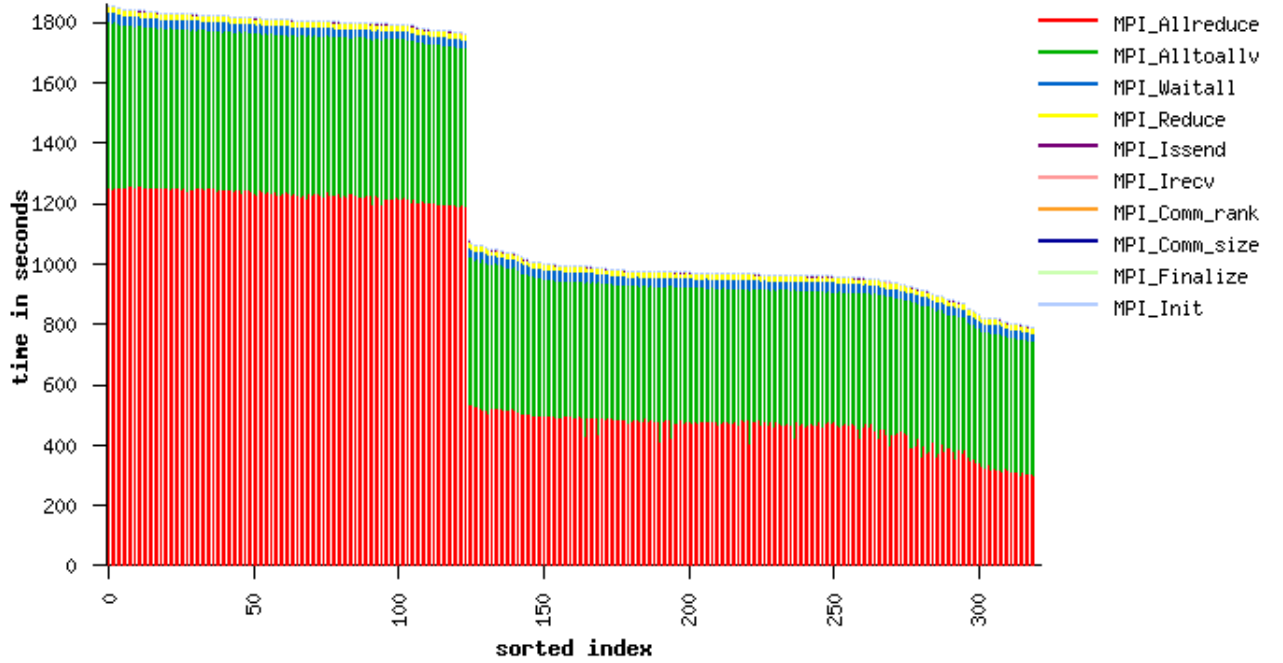


Figure 7.6: Distribution MPI time task wise on Niagara.

The biggest difference between both cluster is seen in Figure 7.5 and Figure 7.6. All processes on Bridges-2 have a nearly equally distributed time per MPI time. For Niagara that was not the case. The approximately first 120 tasks wait longer then the remaining tasks.

For task 3 we used the Bridges-2 cluster to create the visualization of the wind turbine case for 5000 iterations and we set the *ioutput* parameter from the input file to 1000 to create a visualization data every 1000 iterations. The Figure 7.7 shows the visualization at various time steps created with the open source tool ParaView.

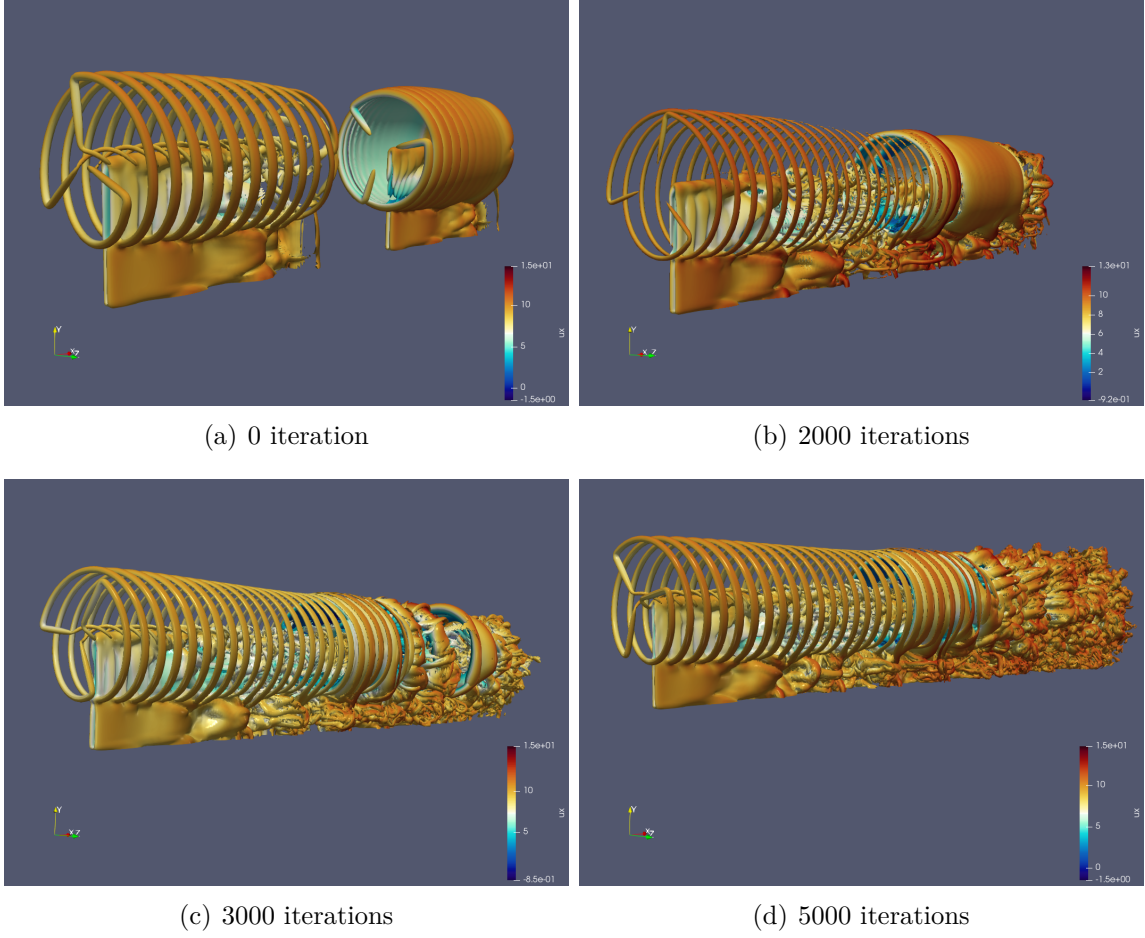


Figure 7.7: Visualization at different time steps.

For the task 4, the bonus task, we should perform a strong scalability studies on each cluster from 1 to 8 nodes with 2500 iterations. Due to restrictions on the Niagara cluster it was only possible to allocate 4 nodes for a job. On Bridges-2 we used MPICH and the default FFT library and on Niagara MVAPICH2 and Intel-MKL. Furthermore, we used different row \times column decompositions to find the fastest configuration for the number of nodes used, as shown in Figure 7.8 and Figure 7.9. In Figure 7.10 we combined all runs to a boxplot, for each cluster, to show that there was a relatively large variance depending on the decomposition. Overall, Xcompact3D shows a good strong scaling behavior, in the measured range up to 8 nodes and an increase in processes shortens the runtime. The only exception is a small peak for the Bridges-2 cluster at 6 nodes, but when the number of processes is increased further, the runtime starts to decrease again.

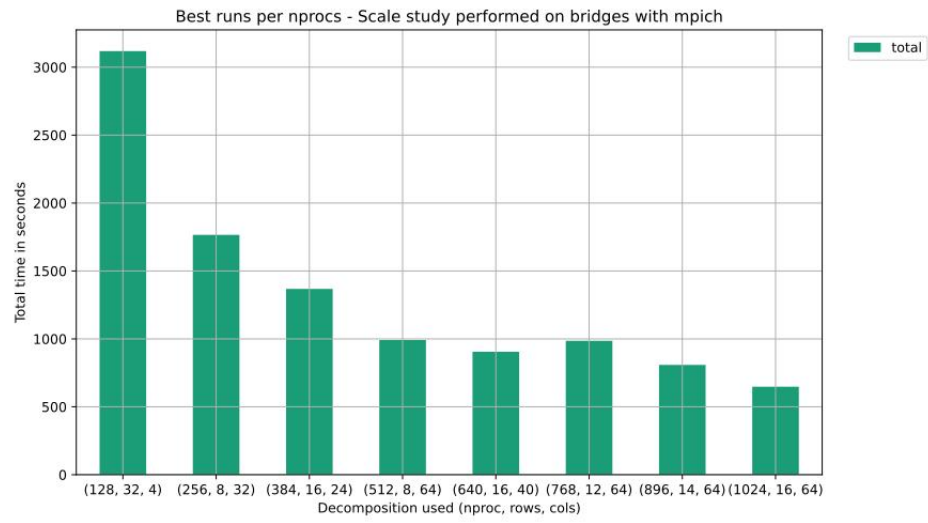


Figure 7.8: Fastest run per number of nodes for Bridges-2.

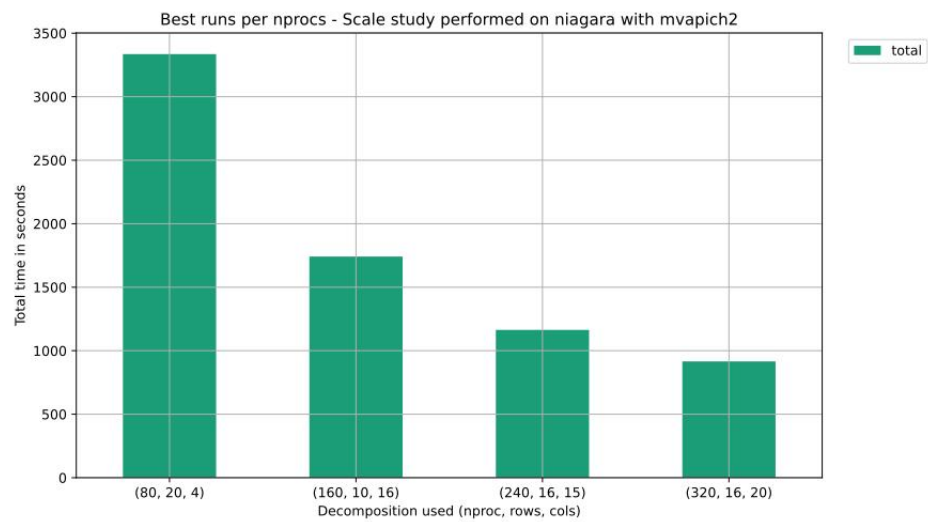


Figure 7.9: Fastest run per number of nodes for Niagara.

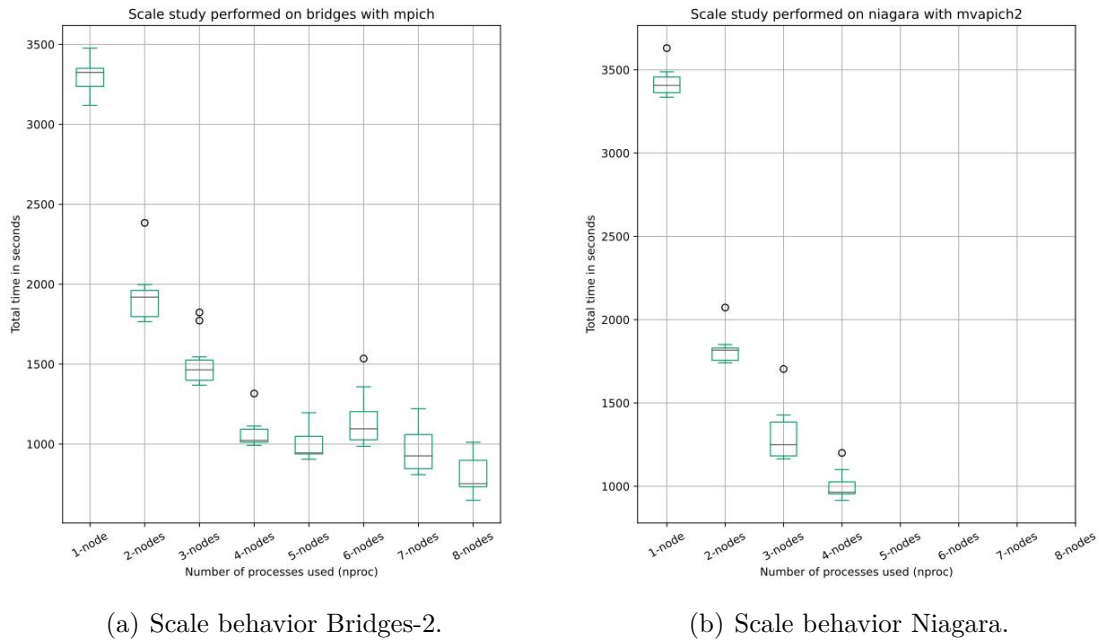


Figure 7.10: Boxplots for scaling behavior for each cluster.

7.2 Coding Challenge

The coding challenge was set up to extend XCompact3D to utilize DPU offloading. The task was divided into 5 parts:

1. Getting familiar with DPUs
2. Modify XCompact3D to utilize asynchronous communication in All-to-All calls
3. Evaluative Benchmarking of the two versions of the program
4. Write a technical report about our findings
5. Bonus: Explorative study to test out different inputs with unmodified and modified XCompact3D

The DPU infrastructure was available on the Thor cluster (see Section 3.2). The challenge had to be performed on a special branch of the code base, that already included some implementation work. The main starting boost was the linking of the asynchronous communication library in the form of libNBC¹.

For the asynchronous communication to work, it was important to change API calls and break them into two parts. `call transpose_x_to_y(in, out, decomp)` had to become

¹<http://www.2decomp.org/occ.html>

call `transpose_x_to_y_start(hande, in, out, send_buffer, receive_buffer, decomp)` and call `transpose_x_to_y_wait(hande, in, out, send_buffer, receive_buffer, \hookrightarrow decomp)`. There are several calls of this type for each combination of x,y and z. The classic overlap pattern (early as possible start of communication, late as possible waiting for completion) should be used to enable a good overlap between communication and computation.

This type of division needed additional buffers to be added and maintained. For each of the dimension combinations there had to be a separate buffer space, to avoid overwriting of existing data.

7.3 On-site competition

For the on-site competition, the same wind turbine case as in the online competition was used with modified parameters for the calculation for 2000 iterations and the only task was to submit the best run including the makefile, job submission file, the input.i3d file and the log file for this run. But due to the hardware problems the time to optimize Xcompact3D for our hardware before and during the competition was very limited. Therefore we used our experience from the online competition for the Bridges-2 cluster, due to similar hardware, as our starting point. To build Xcompact3D we used GCC, HPC-X and tried both the FFT and FFTW library. Initially we reduced the iteration count and tried different row \times column decompositions on 2 nodes, which turned out to be very slow. At that time we already suspected that there was a problem with the communication between the two nodes. Therefore we repeated the whole process to find the best decomposition but this time on just one node, which surprisingly gave slightly better results. So we got the final parameter for the best run, as shown in Table 7.3.

	runtime in min	compiler	MPI	FFT	p_row	p_col
best run	49.84	GCC	HPC-X	default FFT	16	16

Table 7.3: Used compiler, libraries and parameter for the best on-site run.

Overall our achieved runtime was behind our expectations. The average time for one iteration was 1.5 s and in for comparison on Bridges-2 for 2 nodes the average time for one iteration was approximately 0.37 s. That means, without the hardware problems and more time for optimization on our hardware, a much better result would have been possible.

8 Secret Application - FALL3D

Author: Frederic Voigt

As a surprise challenge during the SCC another Fortran application was revealed. FALL3D is a model solver for advection–diffusion–sedimentation equations on a terrain–following grid with a finite differences scheme [FCM18]. The main purpose was the forecast of volcanic ash distribution and passive transport in the atmosphere after an eruption [FCM18]. In the current version FALL3D-7.3.1 parallel execution was possible for the first time (in publicly available open source code), which made it well suited as a secret application in the SCC [FCM18].

Tasks

The input was a *.inp* file, which is FALL3D’s own input format. For the SCC, we looked at data from an eruption of Mount St. Helen on May 18, 1980 [Wes]. The input file could be further processed with a simple command from FALL3D.

Tuning

Fall3D itself was not included in Spack. However, all dependencies - an MPI implementation with Fortran compilation and NetCDF - were available in Spack, so building the application was relatively easy. We already needed both dependencies for the previous applications. As a compiler we used GCC 12. For the Secret Application, we managed to use the library PnetCDF (the parallel variant of NetCDF). We assume that the use of this library significantly improved the final performance, since the I/O could be parallelized.

Tuning consisted, for the building aspect, only of setting the *-enable-parallel* flag in the *configure*-step, as well as using the compiler optimization levels.

Another aspect was the computational grid used by FALL3D. This was divided into 3 dimensions among the processes. Based on our previous experience, we empirically determined that the outermost loop should be kept as large as possible. Also, based mainly on our experience with ICON, we decided to experiment with binding of the processes and tested socket, NUMA and L3-cache bindings. Therefore we tried different combinations which can be seen in Figure 8.1.

We also used the CPU boost mode, which was explained in chapter 2. The use led to significantly improved performance (sometimes with improvements in the range of up to

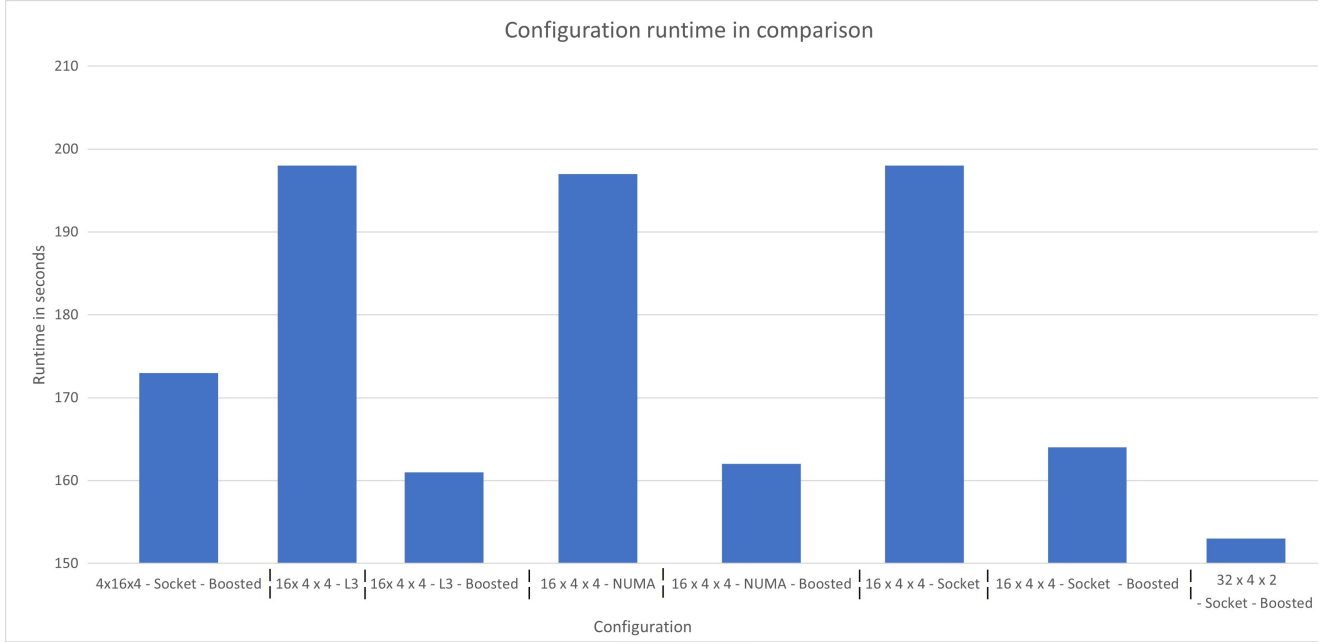


Figure 8.1: Comparison of FALL3D results. For the configurations, first the distribution of the processes on the 3D-Grid is given. After that the binding instance is given and finally if the configurations was boosted or not.

25%, such as with a $16 \times 4 \times 4$ split with binding on L3 cache level).

Results

In the end, a configuration with the largest outer loop, a $32 \times 4 \times 2$ partition, prevailed. It also showed that boosting had a positive effect across all configurations. The effects of binding, on the other hand, were relatively small and there were only differences in the range of a few seconds.

The best runtime was 153 seconds in the end.

9 Learnings

Authors: Frederic Voigt, Christian Willner

We have been able to gain new experience in various aspects of the project work and have also been able to take away a few things on a technical level.

Foremost, we have acquired a much more routine handling of the tools, documentations or the application of theoretical knowledge in basic areas of scientific computing.

For example, most of us had already worked with Spack once or twice, but after the competition, its use became routine. The power of this package manager became clear especially in the applications NWChem and Xcompact. Also the careful use of the possibilities of Spack, like the usage and combination of compilers, dependencies or pre-configuration of the applications for the on-site competition, became easier for all of us.

In the more theory-based courses, the practical aspects of working with scientific applications tend to be less prominent in our experience. For example, working with dependencies of individual applications, linking libraries or instrumenting/sampling applications for profiling. While these aspects presented us with great challenges at the beginning, thanks to the steep learning curve we almost always knew what to do at the end of the SCC.

Working with documentation is part of the daily routine in any form of programming. The scientific, mostly non-commercial, applications sometimes have harsh documentation and sometimes it is very difficult to find them at all. In the course of the SCC work, however, we have become more and more comfortable with this circumstance and think that this knowledge will help us in all parts of software development.

During the competition, we were sensitized to the properties of scientific software. So we were able to find even the most absurd errors more and more quickly. One illustrative example, in the ICON application, is that certain commands in the jobscript must have a space at the end because the strings are concatenated. Such errors can take an enormous amount of time, but with time you get a feeling for how bugs are systematically tracked down and where the sources of errors can be found.

But we were also able to gain some knowledge on a less practical level. We were particularly pleased with the way in which theoretical knowledge from the university courses was applied to the practical side of the competition.

A prime example of this was the modification of our hardware. Although this admittedly led to some complications in the actual implementation, it illustrates one aspect of high-performance computing. The importance of the interconnection network of nodes and the arrangement of the nodes among each other was well demonstrated to us here. For example, simply changing the hardware elements in a way that was more suitable for the underlying software led to an increase in performance.

A similar interesting aspect was already mentioned in ICON. The mapping of processes and threads to the hardware is often only discussed very briefly in lectures due to time constraints. We were all the more pleased that we were able to get a relatively high speedup by changing the mapping, since the initial mapping made such poor use of the hardware.

When it comes to scientific high performance computing, it is mostly the case that Fortran will be mentioned at some point. As a group we were stunned when every single application demanded some degree of Fortran literacy. None of us had any prior knowledge of the language and we had to learn the language to dive deeper into certain problems. The application of Fortran was interesting for two reasons. On the one hand, Fortran still forms the basis for many other programming languages thanks to its efficiency and it was very beneficial for the general understanding of the language. On the other hand, we were able to gain insight into the basics of high performance computing in general. Most problems did not ask to change the source code, but especially the coding challenge (see Section 7.2) needed some experience in modern Fortran. Since the task here was to rebuild the communication and blocking and non-blocking communication is an important aspect of efficient programming, this was all the more beneficial for our understanding of basic HPC concepts.

The last aspect we would like to mention is the profiling of the programs with IPM. This profiler may not be the first choice as a tool, but we have enjoyed evaluating the programs. It is very interesting to see how different changes and tuning affects the overall behavior of the programs. Even without a close look at the source code, weaknesses can be identified and, with a rough understanding of the application functionality, assumptions can be made about where these weaknesses arise. In ICON, for example, it could be seen that the interaction between atmosphere and ocean nodes leaves clear traces in the profiling.

All in all, the SCC remains a very positive memory for us and we were able to learn a lot.

Bibliography

- [ABdJ⁺20] E Aprà, E J Bylaska, W A de Jong, N Govind, K Kowalski, T P Straatsma, M Valiev, H J J van Dam, Y Alexeev, J Anchell, V Anisimov, F W Aquino, R Atta-Fynn, J Autschbach, N P Bauman, J C Becca, D E Bernholdt, K Bhaskaran-Nair, S Bogatko, P Borowski, J Boschen, J Brabec, A Bruner, E Cauët, Y Chen, G N Chuev, C J Cramer, J Daily, M J O Deegan, T H Dunning, Jr, M Dupuis, K G Dyall, G I Fann, S A Fischer, A Fonari, H Früchtl, L Gagliardi, J Garza, N Gawande, S Ghosh, K Glaesemann, A W Götz, J Hammond, V Helms, E D Hermes, K Hirao, S Hirata, M Jacquelin, L Jensen, B G Johnson, H Jónsson, R A Kendall, M Klemm, R Kobayashi, V Konkov, S Krishnamoorthy, M Krishnan, Z Lin, R D Lins, R J Littlefield, A J Logsdail, K Lopata, W Ma, A V Marenich, J Martin Del Campo, D Mejia-Rodriguez, J E Moore, J M Mullin, T Nakajima, D R Nascimento, J A Nichols, P J Nichols, J Nieplocha, A Otero-de-la Roza, B Palmer, A Panyala, T Pirojsirikul, B Peng, R Peverati, J Pittner, L Pollack, R M Richard, P Sadayappan, G C Schatz, W A Shelton, D W Silverstein, D M A Smith, T A Soares, D Song, M Swart, H L Taylor, G S Thomas, V Tipparaju, D G Truhlar, K Tsemekhman, T Van Voorhis, Á Vázquez-Mayagoitia, P Verma, O Villa, A Vishnu, K D Vogiatzis, D Wang, J H Weare, M J Williamson, T L Windus, K Woliński, A T Wong, Q Wu, C Yang, Q Yu, M Zacharias, Z Zhang, Y Zhao, and R J Harrison. NWChem: Past, present, and future. *J. Chem. Phys.*, 152(18):184102, May 2020.
- [DH13] Jack Dongarra and Michael A Heroux. Toward a new metric for ranking high performance computing systems. *Sandia Report, SAND2013-4744*, 312:150, 2013.
- [FCM18] A. Folch, A. Costa, and G. Macedonio. Fall3d-7.3.1 users manual. 2018. <http://datasim.ov.ingv.it/download/fall3d/manual-fall3d-7.3.1.pdf> (visited: 2022-09-07).
- [LDK⁺05] Piotr Luszczek, Jack J Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John McCalpin, David Bailey, and Daisuke Takahashi. Introduction to the hpc challenge benchmark suite. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2005.
- [Li10] Ning Li. 2 decomp and fft user guide. 2010. <http://www.hector.ac.uk>

/cse/distributedcse/reports/incompact3d/UserGuide.html (visited: 2022-08-08).

[Wes] Liz Westby. Mount st. helens in eruption, may 18, 1980. <https://www.usgs.gov/media/videos/mount-st-helens-eruption-may-18-1980> (visited: 2022-09-07).

[Wic05] Nathan Wichmann. Cray and hpcc: Benchmark developments and results from past year. *Proceedings of CUG*, pages 16–19, 2005.

Appendices

List of Figures

2.1	Sketch of our cluster.	7
2.2	Initial system topology.	9
2.3	Possible topology configurations.	10
5.1	Different distributions of the Ocean processes on Niagara. The whole experiment utilized four nodes of 40 processes each. The bars represent one experiment each, with the labels referring to the total number of processes assigned to the Ocean component. The left figure corresponds to the initial coarse evaluation, the right shows the fine-grained one. . . .	22
5.2	Different values for <i>nproma</i> on Niagara. The left figure corresponds to the initial coarse evaluation, the right shows the fine-grained one.	22
5.3	Different process mapping options, tested on Bridges-2.	23
5.4	Shares of communication time for the different MPI calls within ICON. This figure was generated via IPM on Niagara.	24
5.5	Communication balance of the individual processes, sorted by MPI ranks. This figure was also generated via IPM on Niagara.	25
6.1	Runtime of NWChem for shifted combinations of the implementations of compiler, scalapack, lapack, blas and FFTW on the Bridges-2 cluster. . .	29
6.2	Runtime of NWChem for shifted combinations of the implementations of compiler, scalapack, lapack, blas and FFTW on the Niagara cluster. . . .	30
6.3	Runtime of NWChem for moved OMP thread count on the Bridges-2 cluster.	31
6.4	Runtime of NWChem for moved OMP thread count on the Niagara cluster.	32
6.5	Distribution of call frequencies of MPI functions in NWChem according to the IPM profiling.	33
7.1	1D domain decomposition example using 4 processors a) decomposed in y direction b) decomposed in x direction. Figure taken from [Li10].	34
7.2	2D domain decomposition example using 4x3 processor grid. Figure taken from [Li10].	35
7.3	MPI comparison for Bridges-2.	37
7.4	MPI comparison for Niagara.	38
7.5	Distribution MPI time task wise on Bridges-2.	39
7.6	Distribution MPI time task wise on Niagara.	40
7.7	Visualization at different time steps.	41
7.8	Fastest run per number of nodes for Bridges-2.	42
7.9	Fastest run per number of nodes for Niagara.	42

7.10	Boxplots for scaling behavior for each cluster.	43
8.1	Comparison of FALL3D results. For the configurations, first the distribution of the processes on the 3D-Grid is given. After that the binding instance is given and finally if the configurations was boosted or not. . .	46

List of Tables

3.1	Different types of Bridges-2 nodes.	14
4.1	Benchmark outputs for HPCC using different MPI implementations. . . .	17
5.1	Runtimes of ICON with different MPI implementations.	20
5.2	Runtimes of ICON with different BLAS implementations.	20
6.1	Final results of the NWChem runs.	33
7.1	Best runs on each cluster with the used compiler and libraries.	38
7.2	Used compiler and libraries for profiling each cluster.	39
7.3	Used compiler, libraries and parameter for the best on-site run.	44

