

# Supercomputing Frontiers and Innovations

2020, Vol. 7, No. 4

## Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

## Editorial Board

### Editors-in-Chief

- **Jack Dongarra**, University of Tennessee, Knoxville, USA
- **Vladimir Voevodin**, Moscow State University, Russia

### Editorial Director

- **Leonid Sokolinsky**, South Ural State University, Chelyabinsk, Russia

### Associate Editors

- **Pete Beckman**, Argonne National Laboratory, USA
- **Arndt Bode**, Leibniz Supercomputing Centre, Germany
- **Boris Chetverushkin**, Keldysh Institute of Applied Mathematics, RAS, Russia
- **Alok Choudhary**, Northwestern University, Evanston, USA

- **Alexei Khokhlov**, Moscow State University, Russia
- **Thomas Lippert**, Jülich Supercomputing Center, Germany
- **Satoshi Matsuoka**, Tokyo Institute of Technology, Japan
- **Mark Parsons**, EPCC, United Kingdom
- **Thomas Sterling**, CREST, Indiana University, USA
- **Mateo Valero**, Barcelona Supercomputing Center, Spain

## Subject Area Editors

- **Artur Andrzejak**, Heidelberg University, Germany
- **Rosa M. Badia**, Barcelona Supercomputing Center, Spain
- **Franck Cappello**, Argonne National Laboratory, USA
- **Barbara Chapman**, University of Houston, USA
- **Yuefan Deng**, Stony Brook University, USA
- **Ian Foster**, Argonne National Laboratory and University of Chicago, USA
- **Geoffrey Fox**, Indiana University, USA
- **Victor Gergel**, University of Nizhni Novgorod, Russia
- **William Gropp**, University of Illinois at Urbana-Champaign, USA
- **Erik Hagersten**, Uppsala University, Sweden
- **Michael Heroux**, Sandia National Laboratories, USA
- **Torsten Hoefler**, Swiss Federal Institute of Technology, Switzerland
- **Yutaka Ishikawa**, AICS RIKEN, Japan
- **David Keyes**, King Abdullah University of Science and Technology, Saudi Arabia
- **William Kramer**, University of Illinois at Urbana-Champaign, USA
- **Jesus Labarta**, Barcelona Supercomputing Center, Spain
- **Alexey Lastovetsky**, University College Dublin, Ireland
- **Yutong Lu**, National University of Defense Technology, China
- **Bob Lucas**, University of Southern California, USA
- **Thomas Ludwig**, German Climate Computing Center, Germany
- **Daniel Mallmann**, Jülich Supercomputing Centre, Germany
- **Bernd Mohr**, Jülich Supercomputing Centre, Germany
- **Onur Mutlu**, Carnegie Mellon University, USA
- **Wolfgang Nagel**, TU Dresden ZIH, Germany
- **Alexander Nemukhin**, Moscow State University, Russia
- **Edward Seidel**, National Center for Supercomputing Applications, USA
- **John Shalf**, Lawrence Berkeley National Laboratory, USA
- **Rick Stevens**, Argonne National Laboratory, USA
- **Vladimir Sulimov**, Moscow State University, Russia
- **William Tang**, Princeton University, USA
- **Michela Taufer**, University of Delaware, USA
- **Andrei Tchernykh**, CICESE Research Center, Mexico
- **Alexander Tikhonravov**, Moscow State University, Russia
- **Eugene Tyrtshnikov**, Institute of Numerical Mathematics, RAS, Russia
- **Roman Wyrzykowski**, Czestochowa University of Technology, Poland
- **Mikhail Yakobovskiy**, Keldysh Institute of Applied Mathematics, RAS, Russia

## Technical Editors

- **Yana Kraeva**, South Ural State University, Chelyabinsk, Russia
- **Mikhail Zymbler**, South Ural State University, Chelyabinsk, Russia
- **Dmitry Nikitenko**, Moscow State University, Moscow, Russia

## Contents

<b>Effects of Using a Memory Stalled Core for Handling MPI Communication Overlapping in the SOR Solver on SX-ACE and SX-Aurora TSUBASA</b> T. Soga, K. Yamaguchi, R. Mathur, O. Watanabe, A. Musa, R. Egawa, H. Kobayashi .....	4
<b>Enhancing the in Situ Visualization of Performance Data in Parallel CFD Applications</b> R.F.C. Alves, A. Knüpfer .....	16
<b>Improving Quantum Annealing Performance on Embedded Problems</b> M.R. Zielewski, M. Agung, R. Egawa, H. Takizawa .....	32
<b>Developing an Architecture-independent Graph Framework for Modern Vector Processors and NVIDIA GPUs</b> I.V. Afanasyev .....	49
<b>Update on Performance Analysis of Different Computational Architectures: Molecular Dynamics in Application to Protein-Protein Interactions</b> V.A. Fedorov, E.G. Kholina, I.B. Kovalenko, N.B. Gudimchuk, Ph.S. Orekhov, A.A. Zhmurov .....	62
<b>Computer Design of Structure of Molecules of High-Energy Tetrazines. Calculation of Thermochemical Properties</b> V.M. Volokhov, E.S. Amosova, A.V. Volokhov, T.S. Zyubina, D.B. Lempert, L.S. Yanovskiy, I.D. Fateev .....	68



This issue is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

# Effects of Using a Memory Stalled Core for Handling MPI Communication Overlapping in the SOR Solver on SX-ACE and SX-Aurora TSUBASA

*Takashi Soga*<sup>1</sup>, *Kenta Yamaguchi*<sup>1</sup>, *Raghunandan Mathur*<sup>2</sup>,  
*Osamu Watanabe*<sup>3</sup>, *Akihiro Musa*<sup>3,4</sup>, *Ryusuke Egawa*<sup>4</sup>, *Hiroaki Kobayashi*<sup>4</sup>

© The Authors 2020. This paper is published with open access at SuperFri.org

Modern high-performance computing (HPC) systems consist of a large number of nodes featuring multi-core processors. Many computational fluid dynamics (CFD) codes utilize a Message Passing Interface (MPI) to exploit the potential of such systems. In general, the MPI communication costs increase as the number of MPI processes increases. In this paper, we discuss performance of the code in which a core is used as a dedicated communication core when the core cannot contribute to the performance improvement due to memory-bandwidth limitations. By using the dedicated communication core, the communication operations are overlapped with computation operations, thus enabling highly efficient computation by exploiting the limited memory bandwidth and idle cores. The performance evaluation shows that this code can hide the MPI communication times of 90% on the supercomputer SX-ACE system and 80% on the supercomputer SX-Aurora TSUBASA system, and the performance of the successive over-relaxation (SOR) method is improved by 32% on SX-ACE and 20% on SX-Aurora TSUBASA.

*Keywords:* Thermal plasma flows, SOR method, MPI, OpenMP, Performance tuning, SX-ACE, SX-Aurora TSUBASA.

## Introduction

Recently, high-performance computing (HPC) systems have been attaining higher arithmetic operation performance. According to the latest Top500 ranking [1], the highest performance of an HPC system is 513 petaflop/s (Pflop/s). HPC systems consist of a large number of nodes with multi-core processors. An application on these systems has to be divided into parallel computation operations and is executed on the multiple cores in parallel. In these cases, a core cannot directly access the data on other nodes; rather, it has to access other nodes by using the data communication provided by the MPI library, which enables point-to-point communication among cores (processes) and collective communication with all cores (processes). In general, parallelization of programs is expected to enable their faster executions by increasing the number of cores. However, with an increased amount of parallelization performed, the decrease in data communication operation time is often small compared with the decrease in computation operation time. Therefore, decreasing the communication time in large-scale simulations is required.

Research studies have explored the combining MPI and OpenMP models (MPI+OpenMP models) to enable overlapping between computation and communication operations, and new features on OpenMP and MPI have been utilized. Sergent et al. studied task scheduling using the OpenMP Tools interface in OpenMP 5.0 and leveraged idle periods of computational threads to progress MPI communications [13]. Castillo et al. presented a mechanism for exchanging event information between MPI and task-based runtime through the MPI tools interface (MPI.T) in MPI 3.0 and enhanced a task scheduler for improving the performance of overlapping of

---

<sup>1</sup>NEC Solution Innovators, Osaka, Japan

<sup>2</sup>NEC Technologies India, New Delhi, India

<sup>3</sup>NEC Corporation, Tokyo, Japan

<sup>4</sup>Tohoku University, Sendai, Japan

computation with communication [3]. The MPI+OpenMP models have also been evaluated on various HPC systems. Gorobets et al. developed a parallel CFD algorithm of turbulent flows with a multilevel MPI+OpenMP+OpenCL parallelization and evaluated the performance of the CFD code on Intel Xeon, Xeon Phi, and Xeon with NVIDIA K80 GPU systems [5]. Oyarzun et al. also evaluated the performance of a thermo-fluid code with the similar parallelization on ARM-based CPUs and GPUs fused in a System-on-Chip (SoC) architecture [12]. Idomura et al. evaluated plasma turbulence with MPI+OpenMP parallelization on K-computer [6]. However, the performance of the MPI+OpenMP models on vector supercomputers SX-ACE and SX-Aurora TSUBASA has rarely been evaluated. Therefore, in this paper we clarify the potential of the models on SX-ACE and SX-Aurora TSUBASA using only the basic function of OpenMP, which is the schedule clause of the work-sharing constructs.

In Section 1, we present our simulation code and the specifications of the evaluation systems. Section 2 describes evaluated overlapping models using a dedicated communication core. In Section 3, we evaluate the performance of the models on the systems. The last section concludes this paper.

## 1. Modern Vector Supercomputers and Target Application

### 1.1. SX-ACE

The SX-ACE supercomputer system is composed of up to 512 nodes interconnected by a custom network switch. Figure 1 depicts an overview of the SX-ACE processor with four powerful vector-architecture cores. A node of SX-ACE consists of one processor and a local memory. The processor can provide a double-precision floating point operating rate of 256 Gflop/s with a memory bandwidth of 256 GByte/s, and its memory capacity is 64 GBytes. In order to achieve a high sustained performance, the ratio of memory bandwidth to floating-point operation (flop/s) rate (Bytes per Flop, B/F) is a key factor. The system B/F per processor of SX-ACE is 1.0, which represents a good balance between performance and memory bandwidth. Each core is composed of a scalar processing unit (SPU), a vector processing unit (VPU), and a vector on-chip cache called Assignable Data Buffer (ADB) implemented with Miss Status Handling Register (MSHR) [4, 10, 11]. VPU is a fundamental component of the SX-ACE vector architecture with its performance of 64 Gflop/s. SX-ACE can process up to 256 vector elements, eight bytes each, by a single vector instruction. The vector architecture works in a single instruction multiple

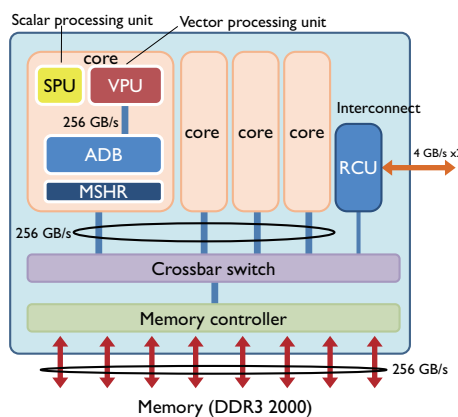


Figure 1. Overview of SX-ACE processor

data (SIMD) manner. The ADB with MSHR avoids redundant data transfers for vector load operations by keeping reusable data on a chip. SPU mainly works as a VPU controller and inherits the architecture of 64-bit RISC processors. The SX-ACE nodes are connected via a custom inter-connect network at a 4 GB/s bandwidth per direction.

The operating system (OS), SUPER-UX, is a production-proven environment based on UNIX System V with several extensions for performance and functionality.

The Fortran and C/C++ compilers support automatic vectorization and parallelization, and parallelization using OpenMP (OpenMP Version 2.5). NEC MPI is implemented in accordance with MPI-3.0 [2].

The Fortran compiler, FORTRAN90/SX, and the C/C++ compiler, C++/SX, support functions of automatic optimization, automatic vectorization, automatic parallelization, and OpenMP (OpenMP Version 2.5). NEC MPI is implemented in accordance with MPI-3.0 [2].

Here, Tab. 1 lists specifications of SX-ACE and SX-Aurora TSUBASA and options of each Fortran compiler.

**Table 1.** Specifications of SX-ACE and SX-Aurora TSUBASA

		SX-ACE	SX-Aurora TSUBASA
Core	Theoretical performance	64 Gflop/s	304 Gflop/s
	Memory bandwidth	256 GB/s	405.5 GB/s
	B/F	4.0	1.33
CPU	Number of cores	4	8
	Theoretical performance	256 Gflop/s	2.43 Tflop/s
	LLC capacity	1 MB (private)	16 MB (shared)
	LLC bandwidth	1000 GB/s	3244 GB/s
	Memory bandwidth	256 GB/s	1.35 TB/s
	B/F	1.0	0.55
Node	Number of CPUs	1	8
	Memory capacity	64 GB	384 GB
	Network bandwidth	8 GB/s	25 GB/s
Fortran compiler	Version	Rev.537	Rev.3.0.6
	Options	-Popenmp, -pi, -EP	-fpp, -finline-functions, -fopenmp

## 1.2. SX-Aurora TSUBASA

SX-Aurora TSUBASA is the newest vector supercomputer released in 2018 [8, 17]. It consists of one or more card-type vector engines (VEs) and a vector host (VH). The VE is the main part of SX-Aurora TSUBASA and contains a vector processor, a 16 MB shared last-level cache (LLC), and six High Bandwidth Memory 2 (HBM2) memory modules, as shown in Fig. 2. The vector processor has eight vector cores. As shown in Tab. 1, the peak performances of a vector core and a vector processor are 304 Gflop/s and 2.43 Tflop/s, respectively. The memory bandwidths are 405.5 GB/s per vector core and 1.35 TB/s per VE. The B/F rates are 1.33 per vector core and 0.55 per vector processor, respectively. The VH is a standard x86 Linux

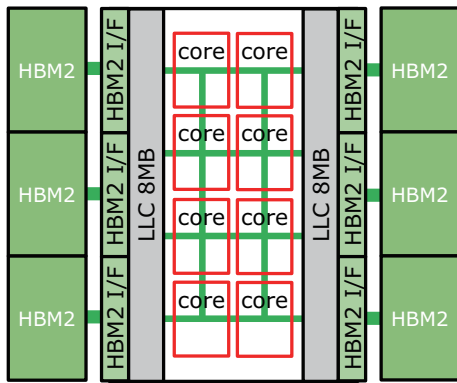


Figure 2. Block diagram of a vector engine

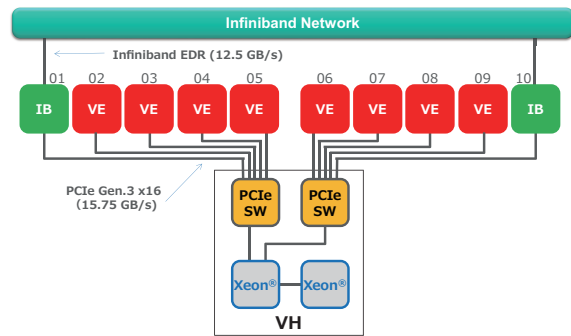


Figure 3. Block diagram of SX-Aurora TSUBASA system

server and executes the OS of the VE. While the conventional SX series runs the OS on a vector processor, SX-Aurora TSUBASA offloads OS-related processing to the VH. The VH consists of up to two Xeon processors and can handle up to eight VEs as shown in Fig. 3. Moreover, SX-Aurora TSUBASA can compose a large system by connecting the VHs via the InfiniBand switch.

The SX-Aurora TSUBASA system supports Red Hat Enterprise Linux and CentOS, and the VH provides OS functions such as process scheduling and handling of system calls invoked by the application running on the VEs. The programming environment includes NEC MPI and NEC Software Development Kit for Vector Engine (NEC SDK). NEC SDK contains an NEC Fortran compiler, NEC C/C++ compiler, NEC Numeric Library Collection, NEC Parallel Debugger, and NEC Ftrace Viewer. The Fortran and C/C++ compilers support functions of automatic optimization, automatic vectorization, automatic parallelization, and OpenMP (OpenMP Version 4.5). NEC MPI is implemented in accordance with MPI-3.1.

### 1.3. Thermal Plasma Flow Code

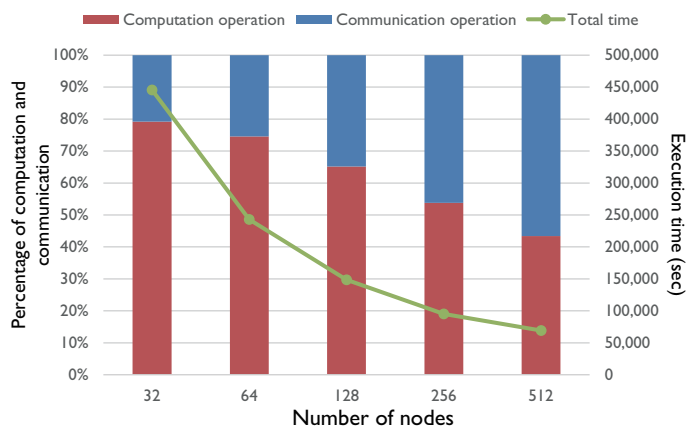
An evaluated code simulates the 3D turbulence on thermal plasma flow. This code solves the conservation that is simulated by solving the conservation equations of mass, momentum, and energy [14]. It uses the Hybrid Upwind K-K scheme and the Adams-Moulton method with third-order accuracy to discretize the convection terms and the time derivative terms, respectively. The discretized equations are numerically solved using the Successive Over-Relaxation (SOR) method. The simulation needs to be magnified in order to accurately reproduce the phenomenon in this code, and our target is the large scale simulation of 2.7 billion grids.

The code uses a Cartesian coordinate system and consists of triple loops. The two outer loops, Y-axis and Z-axis, are parallelized by the domain decomposition method, and the outermost loop, Z-axis, is also parallelized by OpenMP. The sub-routine with the SOR method has an unvectorizable loop structure, and the red-black parallelization method [7] is utilized to vectorize it on the SX-ACE supercomputer [16]. However, the computation cost of the sub-routine with the SOR method is the largest. The data sizes of the evaluated code are listed in Tab. 2.

The SOR method is an iterative method that executes a convergent calculation of the residual error. To perform this process, a collective communication operation for summing up residual values and a point-to-point communication operation for exchanging the boundary data are required. It takes longer to perform the point-to-point communication than the collective

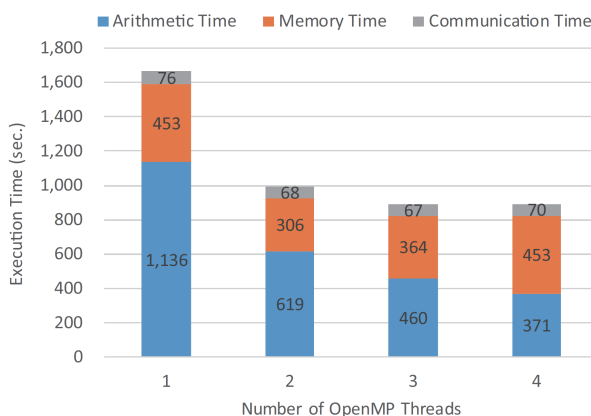
**Table 2.** Data sizes of evaluated code

Model	Data size			Number of grids
	X	Y	Z	
Small data	516	204	204	21,473,856
Medium data	1280	512	512	1,342,177,280
Large data	2560	1024	1024	2,684,354,560

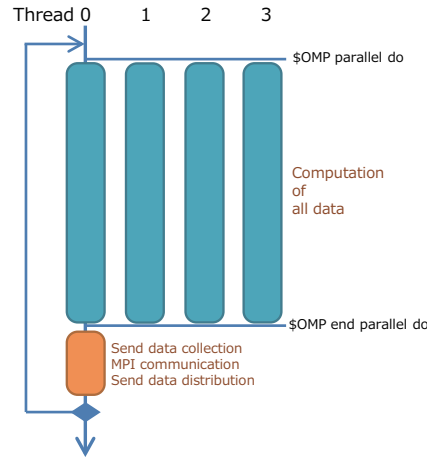

**Figure 4.** Breakdown of execution time

communication in this code. Therefore, the execution time for the point-to-point communication should be shortened.

Figure 4 shows the breakdown of execution time for the large data with 2.7 billion grids on the evaluation code when increasing the number of nodes. The blue and red parts of the bars indicate the execution times for the computation and communication on SX-ACE, respectively. When this code is executed on the 128-node SX-ACE, the computation operation takes up more time than the communication operation. However, when we increase the number of nodes to 512, the communication operation becomes dominant. As a result, the communication operation time becomes dominant in the entire operation time as the number of nodes increases.


**Figure 5.** Relationship between execution time and number of cores on SX-ACE





**Figure 6.** Diagram of the computation and communication operations of the SOR method

We evaluate the execution time (arithmetic time, memory time, and communication time) with changing the number of OpenMP threads per processor from one-thread to four-thread on SX-ACE. Figure 5 shows the relationship between the execution time and the number of OpenMP threads using four SX-ACE nodes with the small data. The memory time indicates the stall time of the core due to data load from the memory. The actual B/F [8] is 5.2 from the hardware counter of instructions on SX-ACE and indicates that the code is memory intensive for SX-ACE. The arithmetic time decreases with increasing the number of threads. However, the memory time increases from the two-thread case to the four-thread case. Therefore, the data transfer between the processor and the main memory becomes a bottleneck.

As mentioned above, even if the memory-intensive program uses all cores for computation operation, it cannot achieve a highly parallelized performance. Therefore, in our overlapping strategy, one core is assigned for the communication operation and the other cores for the computation operation. We examine the effect of the overlapping model from the viewpoint of hiding the communication operation time in memory-intensive applications such as those using the SOR method.

## 2. Evaluated MPI+OpenMP Models

Figure 6 shows the diagram of the parallel execution of the SOR method by MPI using OpenMP. This parallelization cannot overlap the computation operation with the communication one. Specifically, a thread on each core executes the calculation of the computation area that is divided by OpenMP, and then thread 0 starts processing of the communication operations after all the threads for computation have been completed. The communication operation gathers/scatters the boundary data for MPI communications.

The overlap of the computation operation and the communication operation is necessary to decrease the execution time. Figure 7 shows the basic concept of overlapping in the SOR method. First, the computation of the boundary data, which is sent by MPI communications, is executed for the communication operation. In the next step, both the computation of the non-boundary data and the communication of the boundary data are executed in parallel. Eventually, the time for the communication operation can be hidden.

Figure 8 shows the diagram of the overlapping model using the “schedule” clauses on OpenMP. First, all cores execute the computation of the boundary data. It then executes the

communication operation. After that, thread 0 begins to execute the non-boundary data, which is parallelized on four threads. Meanwhile, the other threads calculate the non-boundary data after the calculation of the boundary data. Therefore, the communication operation is overlapped with the calculation of the non-boundary data. When the communication operations are completed, thread 0 begins the execution of the computation operations for the non-boundary data. These operations are controlled automatically by OpenMP. The “schedule (static)” clause is used in this implementation because it has a smaller overhead for OpenMP than that of the “schedule (dynamic)” clause [9]. Although the overlapping can be achieved simply by using the directive statement, there is still an overhead of OpenMP.

In this study, we focus on the memory-intensive code that cannot effectively use all cores in a processor. Figure 9 shows the diagram of our execution model [15]. In the first step, all threads execute the computation operation of the boundary data. Then, thread 0 is used as a dedicated communication thread and executes the communication operation. At the same time, other threads simultaneously execute the computation operation of the non-boundary data. Each thread executes the prearranged DO loop as shown in Fig. 9, where the program calculates the starting and ending point of the DO loop on each thread.

### 3. Performance Evaluation

We evaluate three kinds of execution models: the non-overlapping model (Fig. 6), called “Original”, the overlapping model (Fig. 8), called “Schedule”, and our model (Fig. 9), called “Manual”. This evaluation measures the calculation time of the SOR method in 20 time steps using three data sets in the code. First, the performance on SX-ACE is evaluated using the small and large data sets. The next evaluation uses the small and medium data sets on SX-Aurora TSUBASA. We will evaluate the performance on SX-Aurora TSUBASA using a large data set when a large system for it can be constructed.

#### 3.1. Evaluation Results on SX-ACE

Figure 10 shows the execution times with the small data set using four and 16 nodes of SX-ACE. Here, computation time contains arithmetic time and memory time. Each node executes one MPI process and four OpenMP threads. Figure 10 (a) shows the case of using four nodes. The communication time includes the operation times for gathering and scattering of boundary data. For “Original”, the computation time is 742 seconds and the communication time is 145 seconds, resulting in the total time of 887 seconds. “Manual” hides 121 seconds in the communication

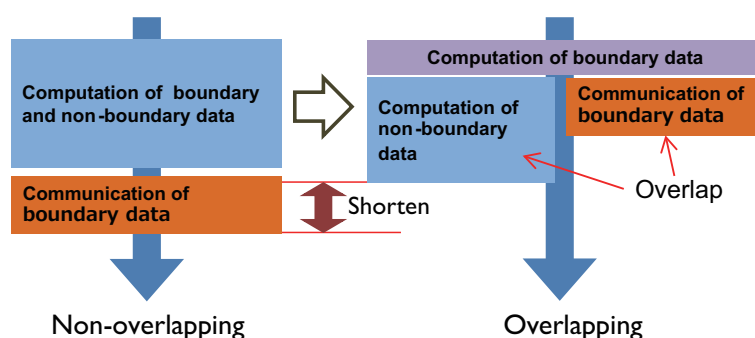


Figure 7. Basic concept of overlapping

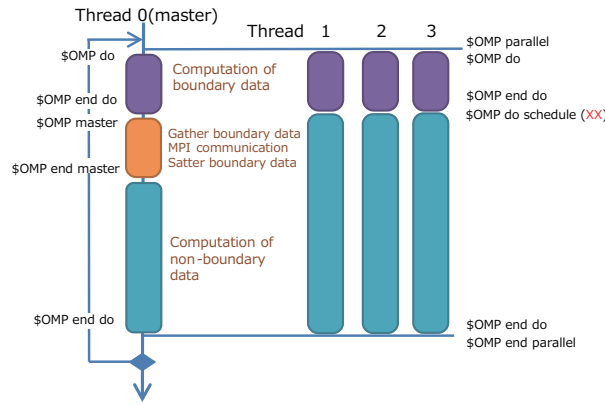


Figure 8. Diagram of implementation using “schedule” clauses on OpenMP

time, and it corresponds to 83% of the communication time. “Schedule” hides 40 seconds. The overlapping effect of “Schedule” is smaller than that of “Manual” because the OpenMP overhead of “Schedule” is larger. Figure 10 (b) shows the result for the 16-node case. The computation time for “Original” decreases from 742 seconds to 208 seconds compared with the four-node case. On the other hand, the communication time for “Original” decreases by only 13 seconds, and the percentage of communication time to the total operation time becomes about 40%. “Manual”

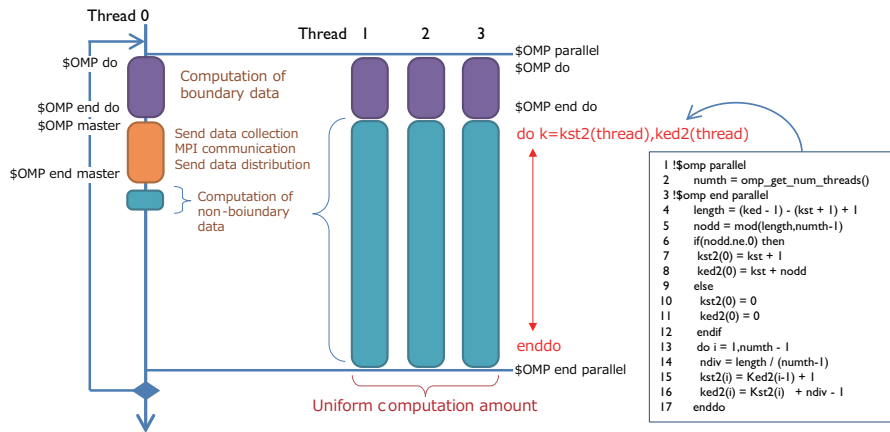


Figure 9. Diagram of our overlapping model

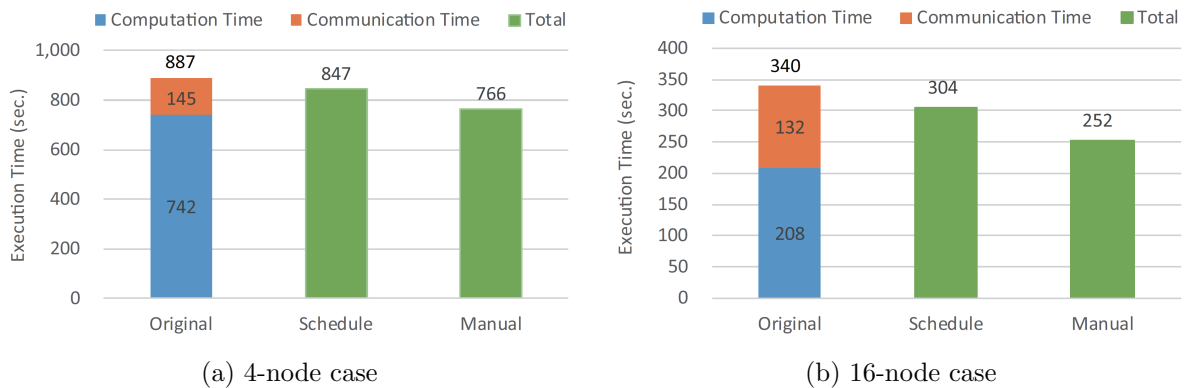


Figure 10. Evaluation results with small data set on SX-ACE

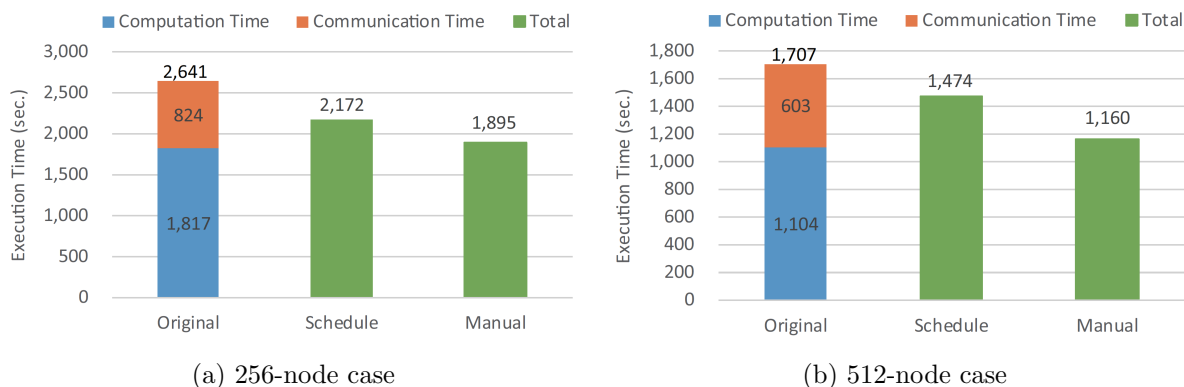


Figure 11. Evaluation results with the large data set on SX-ACE

hides the communication time of 88 seconds, which corresponds to 67% of the communication time of “Original”. “Schedule” hides 36 seconds in the communication time of “Original”.

Figures 11 (a) and (b) show the results of the 256-node and 512-node cases with the large data set, respectively. Each node of SX-ACE uses one MPI process and four OpenMP threads in this evaluation. In Fig. 11 (a), “Manual” hides the communication time of 746 seconds, which corresponds to 90% of the communication time of “Original”, and “Schedule” hides 469 seconds in the communication time of “Original”. When the number of nodes increases from 256 to 512, the percentage of communication time to the total time becomes large. However, “Manual” in Fig. 11 (b) is able to hide 547 seconds in the communication time of “Original”, which represents a 90% decrease in the communication time. Meanwhile, the hidden time on “Schedule” decreases to only 233 seconds. Our overlapping model “Manual” can hide a great part of the communication time, and the execution times for the 512-node decreases by about 32%. On the basis of the above results, we show that SX-ACE has a potential to improve the performance of MPI+OpenMP models for the memory-intensive code.

### 3.2. Evaluation Results on SX-Aurora TSUBASA

Figure 12 shows the execution times with the small data set using two and four VEs of SX-Aurora TSUBASA. Each VE is combined via a PCIe switch and executes one MPI process and eight OpenMP threads. Figure 12 (a) shows the results using two VEs. The communication time of “Original” in Fig. 12 (a) is 43 seconds. “Manual” hides 29 seconds, which represents a

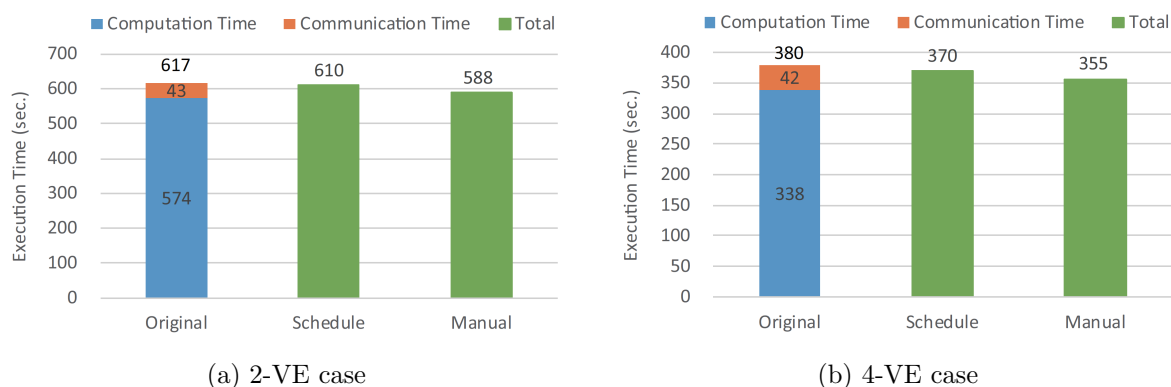
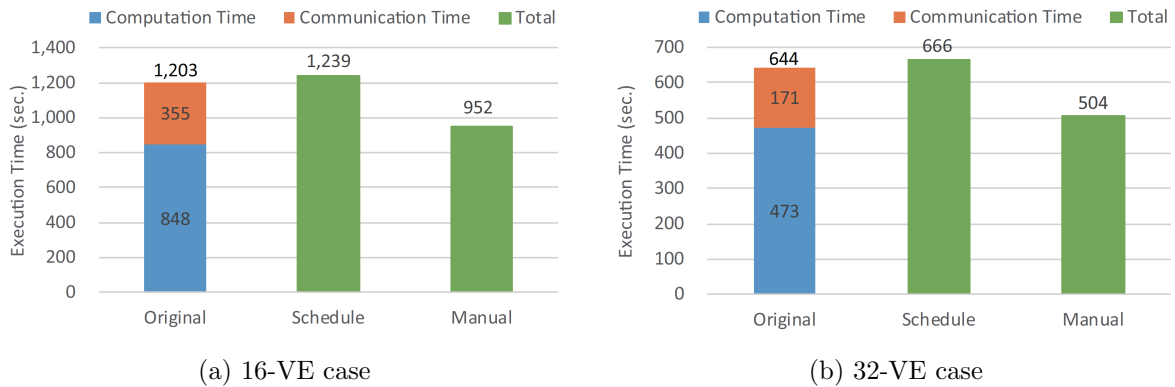


Figure 12. Evaluation results with small data set on SX-Aurora TSUBASA



**Figure 13.** Evaluation results with medium data set on SX-Aurora TSUBASA

roughly 67% decrease in the communication time. Figure 12 (b) shows the results using four VEs. The communication time of “Original” is nearly the same as that in Fig. 12 (a), although increasing the number of MPI processes. “Manual” also hides 25 seconds, which corresponds to roughly 60% of the communication time. However, “Schedule” in the case of two and four VEs has little overlapping effort.

We utilize 16 and 32 VEs of SX-Aurora TSUBASA for evaluation in the medium data set. The 16-VE and 32-VE systems contain two VHs and four VHs, respectively. Each VH is connected via an InfiniBand switch, and executes one MPI process and eight OpenMP threads. Figures 13 (a) and (b) show the evaluation results for 16 VEs (2 VHs) and 32 VEs (4 VHs). The computation time in “Original” on 32 VEs decreases from 848 seconds to 473 seconds compared with the case of 16 VEs, and the ratio of the communication times to the total time on the 16 and 32 VEs are about 30% and 27%, respectively. In both cases, “Manual” can hide a part of the communication time: on 16 VEs, it hides about 251 seconds among the communication time of 355 seconds, which corresponds to roughly 70% of the communication time, and 32 VEs, it hides about 140 seconds among the communication time of 171 seconds, which represents a roughly 80% in the communication time. Then, the overall execution times on each case can decrease by about 20%. Meanwhile, “Schedule” was unable to decrease the execution times. SX-Aurora TSUBASA was released in 2018, and the fortran compiler was newly developed. Therefore, the overhead of OpenMP on SX-Aurora TSUBASA is high. In this evaluation, our overlapping model, “Manual”, can hide a great part of the communication time on SX-Aurora TSUBASA. We expect our overlapping model to produce increasing performance on a larger data set with a larger system.

## Conclusions

In this work, we focused on the overlapping between the computation and MPI communication operations of the SOR method in a thermal plasma flow simulation code, and examined an implementation of MPI+OpenMP models where OpenMP thread 0 assigns the communication operation and a part of the computation operation. Evaluation results demonstrated that SX-ACE and SX-Aurora TSUBASA show a good potential for overlapping computation and MPI communication on memory intensive codes, and our overlapping model, “Manual”, can hide the MPI communication times of 90% on SX-ACE and 80% on SX-Aurora TSUBASA. Our overlapping model with SX-ACE and SX-Aurora TSUBASA is expected to increase performance of memory intensive codes.

In future work, we will evaluate our three models using memory intensive codes from various application fields on SX-Aurora TSUBASA and other systems: such as Intel Xeon and AMD EPYC.

## Acknowledgments

The authors would like to thank Associate Professor Masaya Shigeta of Osaka University for a lot of useful advice. This research uses the SX-ACE systems of Cyberscience Center at Tohoku University.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Top 500 the list. <https://www.top500.org/>
2. Vector supercomputer SX series SX-ACE. [https://de.nec.com/de\\_DE/en/documents/SX-ACE-brochure.pdf](https://de.nec.com/de_DE/en/documents/SX-ACE-brochure.pdf)
3. Castillo, E., Jain, N., Casas, M., et al.: Optimizing computation-communication overlap in asynchronous task-based programs. In: Proceedings of the ACM International Conference on Supercomputing, ICS '19, June 2019, Phoenix, Arizona, USA. pp. 380–391. ACM (2019), DOI: 10.1145/3330345.3330379
4. Egawa, R., Komatsu, K., Takizawa, H., et al.: Early evaluation of the SX-ACE processor. In: Proceedings of the 27th International Conference for High Performance Computing, Networking, Storage and Analysis (2014)
5. Gorobets, A., Soukov, S., Bogdanov, P.: Multilevel parallelization for simulating compressible turbulent flows on most kinds of hybrid supercomputers. *Computers & Fluids* 173, 171–177 (2018), DOI: <https://doi.org/10.1016/j.compfluid.2018.03.011>
6. Idomura, Y., Nakata, M., Yamada, S., et al.: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer. *The International Journal of High Performance Computing Applications* 28(1), 73–86 (2014), DOI: 10.1177/1094342013490973
7. Iwashita, T., Shimasaki, M.: Algebraic block red-black ordering method for parallelized ICCG solver with fast convergence and low communication costs. *IEEE Transactions on Magnetics* 39(3), 1713–1716 (2003), DOI: 10.1109/TMAG.2003.810531
8. Komatsu, K., Momose, S., Isobe, Y., et al.: Performance evaluation of a vector supercomputer SX-Aurora TSUBASA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, 11-16 November 2018, Dallas, Texas, USA. IEEE Press (2018), DOI: 10.5555/3291656.3291728
9. Mattson, T.G., He, Y., Koniges, A.E.: *The OpenMP Common Core*. The MIT Press (2019)

10. Momose, S., Hagiwara, T., Isobe, Y., et al.: The brand-new vector supercomputer, SX-ACE. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) *Supercomputing. Lecture Notes in Computer Science*, vol. 8488, pp. 199–214. Springer, Cham (2014), DOI: 10.1007/978-3-319-07518-1\_13
11. Musa, A., Sato, Y., Soga, T., et al.: Effects of MSHR and prefetch mechanisms on an on-chip cache of the vector architecture. In: *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, 10-12 Dec. 2008, Sydney, NSW, Australia. pp. 335–342. IEEE (2008), DOI: 10.1109/ISPA.2008.100
12. Oyarzun, G., Borrell, R., Gorobets, A., et al.: Efficient CFD code implementation for the ARM-based Mont-Blanc architecture. *Future Generation Computer Systems* 79, 786–796 (2018), DOI: 10.1016/j.future.2017.09.029
13. Sergent, M., Dagrada, M., Carribault, P., et al.: Efficient communication/computation overlap with MPI+OpenMP runtimes collaboration. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) *Euro-Par 2018: Parallel Processing. Lecture Notes in Computer Science*, vol. 11014, pp. 560–572. Springer, Cham (2018), DOI: 10.1007/978-3-319-96983-1\_40
14. Shigeta, M.: Turbulence modelling of thermal plasma flows. *Journal of Physics D: Applied Physics* 49(49), 493001 (2016), DOI: 10.1088/0022-3727/49/49/493001
15. Soga, T., Yamaguchi, K., Mathur, R., et al.: Effects of using a memory-stalled core for handling MPI communication overlapping in the SOR solver. In: *The 29th International Conference on Parallel Computational Fluid Dynamics*, 15-17 May 2017, Glasgow, UK (2017)
16. Soga, T., Musa, A., Okabe, K., et al.: Performance of SOR methods on modern vector and scalar processors. *Computers & Fluids* 45(1), 215–221 (2011), DOI: 10.1016/j.compfluid.2010.12.024
17. Yamada, Y., Momose, S.: Vector Engine Processor of NEC Brand-New supercomputer SX-Aurora TSUBASA. In: *International symposium on High Performance Chips, Hot Chips 2018*, August 2018, Cupertino, USA (2018)

# Enhancing the in Situ Visualization of Performance Data in Parallel CFD Applications

Rigel F. C. Alves<sup>1</sup> , Andreas Knüpfer<sup>1</sup> 

© The Authors 2020. This paper is published with open access at SuperFri.org

This paper continues the work initiated by the authors on the feasibility of using *ParaView* as visualization software for the analysis of parallel CFD codes' performance. Current performance tools are unable to show their data on top of complex simulation geometries (e.g. an aircraft engine). In our previous paper, a plugin for the open-source performance tool *Score-P* has been introduced, which intercepts an arbitrary number of manually selected code *regions* (mostly functions) and send their respective measurements – *amount* of executions and cumulative *time* spent – to *ParaView* (through its in situ library, *Catalyst*), as if they are any other flow-related variable. This paper adds to such plugin the capacity to also show communication data (messages sent between MPI ranks) on top of the CFD mesh. Testing is done again with Rolls-Royce's in-house CFD code, *Hydra*. The plugin's original feature (regions' measurements) is here revisited, in a bigger test-case, which is also used to illustrate the new feature (communication data). The benefits and overhead of the tool are discussed.

*Keywords:* parallel computing, performance analysis, in situ processing, computational fluid dynamics.

## Introduction

Computers have become mandatory resources in solving engineering problems. For the size of today's typical ones (like designing aircraft), one needs to *parallelize* the simulation (e.g. of the air flowing through the airplane's engine) and run it in High Performance Computing (HPC) hardware. Those are expensive infrastructures, both from *time* and *energy* consumption point-of-views. Therefore the application needs to have its parallel performance high-tuned for maximum productivity.

There are many tools for analyzing the performance of parallel applications; one of them is *Score-P*<sup>2</sup> [9], the development of which the *Centre for Information Services and HPC* (ZIH) of the Technische Universität Dresden participates in. It instruments the simulation code and monitors its execution, and can be easily turned on or off by the user at compile time. When applied to a source code, the simulation will produce in the end, apart from its native outputs, also the performance data. This is illustrated in the upper part of Fig. 1 below.

However, all tools currently available to *visualize* the performance data (generated by software like *Score-P*) lack important features, like three-dimensionality, time-step association (i.e. frame playing) and most importantly, matching to the simulation original geometry (where everything happens in terms of computations and therefore where load imbalances lie).

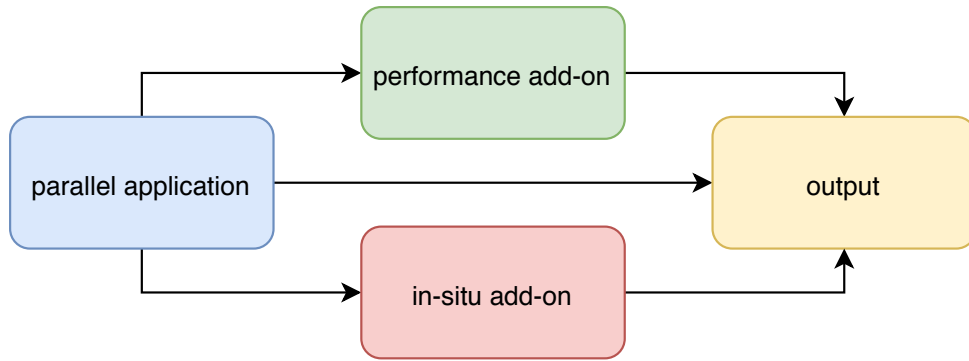
As a separate category of add-ons, tools for enabling *in situ* visualization of applications' output data – like temperature or pressure in a Computational Fluid Dynamics (CFD) simulation – already exist too; one example is *Catalyst*<sup>3</sup> [3]. They also work as an optional layer to the original code and can be activated upon request, by means of preprocessor directives

<sup>1</sup>Technische Universität Dresden, Center for Information Services and High Performance Computing (ZIH), Dresden, Germany

<sup>2</sup>*Scalable Performance Measurement Infrastructure for Parallel Codes* – an open-source “highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications” [tool's website].

<sup>3</sup>An open-source “in situ use case library, with an adaptable application programming interface (API), that orchestrates the delicate alliance between simulation and analysis and/or visualization tasks” [tool's website].



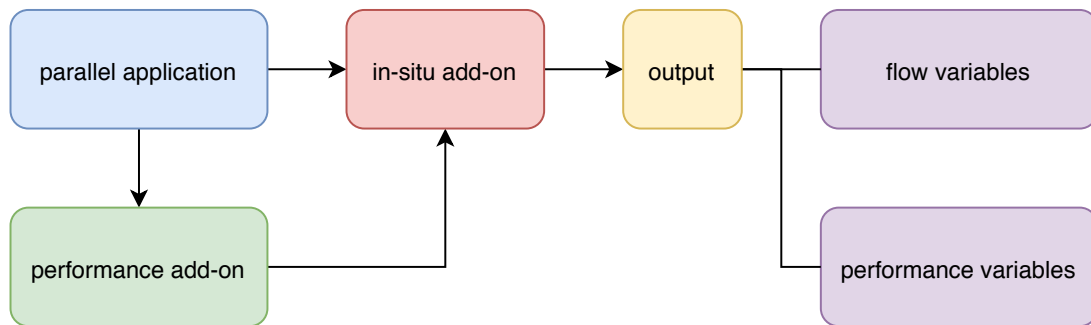


**Figure 1.** Schematic of software components for parallel applications

at compilation stage. The simulation will then produce its native outputs, if any,<sup>4</sup> plus the *coprocessor*'s (a piece of code responsible for allowing the original application to interact with the in situ methods) ones, in separate files. This is also illustrated in Fig. 1. These tools have been developed by visualization specialists for decades long by now and feature abundant visual resources.

Then the question comes: why not use such in situ tools (made to extract data from the simulation by separate side channels, just like the performance instrumenters) for the benefit of the performance analysis of parallel applications (filling by that the lack of visual resources of the performance tools)?

This work continues our investigations on the feasibility of merging the aforementioned approaches. First, by unifying the overlapping functionalities of both kinds of tools, insofar as they augment a parallel application with additional features (which are not strictly required for the application to work in the first place). Second, by using the advanced functionalities of dedicated visualization software for the purpose of performance analysis. Figure 2 illustrates the idea.



**Figure 2.** Schematic of the software components for a combined add-on

In our previous paper [2], we mapped performance measurements of code regions – *amount* of executions and cumulative *time* spent – to the simulation's geometry, just like it is done for flow-related properties. In this paper, we shall add to our tool the capacity of showing communication data (messages sent between MPI ranks) on top of the CFD mesh. We will then see how communication inefficiencies become immediately visible.

HPC analysis tools usually produce either *performance profiles* or *event traces*. In the case

<sup>4</sup>I.e. if the in situ channel does not replace completely the code's original outputs, as it uses to happen.

of Score-P, they are: performance profiles in the Cube4 format, to be visualized with *Cube*<sup>5</sup>; and parallel event traces in the OTF2 format, to be visualized with *Vampir*<sup>6</sup>. But neither of them, nor the other currently available tools, nor the related attempts in the literature (to be summarized in Section 1 below), display their measurements onto complex geometries (like those found in industry-grade CFD problems), what makes our proposal novel.

A design requirement is that the combined solution must be easily applicable on the source code, yet without becoming a permanently required component: it needs to be “activatable” on demand, as it is the case for each of its constitutive parts (*performance measurement* and *in situ processing*). As evaluation case, the Rolls-Royce’s in-house CFD code (*Hydra*) will be used.

This paper is organized as follows: in Section 1 we discuss the efforts made so far at the literature to map performance data to the simulation’s geometry and the limitations of their results. In Section 2 we present the methodology of our approach, which is then evaluated in the test-case in Section 3. Finally, Section 4 discusses the overhead associated with using our tool. We then conclude the article with a summary and point directions for future work.

## 1. Related Work

In order to assist the developer of parallel codes in its optimization tasks, many software tools have been developed. For a comprehensive list of them, including information about their:

- *scope*, whether single or multiple nodes (i.e. shared or distributed memory);
- *focus*, be it performance, debugging, correctness or workflow (productivity);
- *programming models*, ranging through MPI, OpenMP, Pthreads, OmpSs, CUDA, OpenCL, OpenACC, UPC, SHMEM and their combinations;
- *languages*: C, C++, Fortran or Python;
- *processor architectures*: x86, Power, ARM, GPU;
- license types, platforms supported, contact details, output examples etc.

the reader is referred to the *Tools Guide* of the *Virtual Institute – High Productivity Supercomputing* (VI-HPS). However, none of them currently match the generated data back to the simulation original geometry.

The necessity of bringing together the branches of *performance analysis* and *visualization* has already been identified by the scientific community [4, 7] and is being pursued at research level. Vierjahn et al. [12] mapped performance data to the simulation geometry, but developing the visualization environment from scratch (the test-case was indeed simpler than a CFD problem). Going this way, it would take decades for the tool to reach the same capabilities of today’s top graphic programs. Huck et al. [5], on the other hand, did follow our approach: performance tool *TAU* was linked to visualization software *VisIt* to show performance data on top of the Earth’s oceans in a climate simulation; however the linking required writing a new VisIt file format reader. To reproduce those results would require the effort of manually recreating such interface for each different (CFD) simulation code, which was undesired. Husain et al. [6] followed a similar path, also using VisIt as visualization tool, but MemAxes as performance measurer – a software which does not seem to have a website, what impacts on its availability. Their results required modifying the source code of both tools involved, which was again undesired. Finally,

---

<sup>5</sup>A free, but copyrighted “generic tool for displaying a multi-dimensional performance space consisting of the dimensions (i) performance metric, (ii) call path, and (iii) system resource” [tool’s website].

<sup>6</sup>An “easy-to-use framework that enables developers to quickly display and analyze arbitrary program behavior at any level of detail” [tool’s website].

similar hurdles can be encountered in the work of Wood et al. [13]: the application needs to be first enveloped by a multipurpose framework (SOSflow) and then linked with a non widely known in situ infrastructure (Ascent) in order for performance data to be shown on simulation geometries which are comprised of parallelepipedic rectilinear grids.

Not without reason, all attempts (to match performance data to the simulation’s geometry) described above were unable to test their features on more complex (CFD) meshes: the required user effort precluded it... We will then advance the state-of-the-art by aiming for a solution which uses an already established way of extracting data from a simulation, which can be directly (i.e. without wrapping layers) applied to any numerical code, running any type of mesh.

## 2. Methodology

### 2.1. Prerequisites

The goal aimed by this research depends on the combination of two basic, scientifically established methods: *performance measurement* and *in situ processing*.

#### 2.1.1. Performance measurement

When applied to a source file compilation, Score-P automatically inserts probes between each code “region”<sup>7</sup>, which will at run-time measure a) the *number of times* that region has been executed and b) the total *time* spent in those executions, by each process (MPI rank) within the simulation. Its application is done by simply prepending the word `scorep` into the compilation command, e.g.: `scorep [Score-P’s options] mpicc foo.c`. It is possible to exclude regions from the instrumentation (e.g. to keep the related overhead low), by adding the flag `--nocompiler` to the command above. In this case, Score-P sees only user-defined regions (if any) and MPI-related functions, the detection of which can be easily (de)activated at run-time, by means of an environment variable: `export SCOREP_MPI_ENABLE_GROUPS=[comma-separated list]`. Leaving it blank turns off instrumentation of MPI routines. Its default value is set to catch all of them.

Finally, the tool is also equipped with an API, which allows the user to extend its functionalities through plugins [11]. The combined solution proposed by this paper takes indeed the form of such a plugin.

#### 2.1.2. In situ processing

In order for Catalyst to interface with a simulation code, an *adapter* needs to be built, which is responsible for exposing the native data structures (mesh and flow properties) to the *coprocessor* component. Its interaction with the simulation code happens through three function calls (*initialize*, *run* and *finalize*), illustrated in blue at Fig. 3. Once implemented, the adapter provides the generation of post-mortem files (by means of the *VTK*<sup>8</sup> library) and/or the live visualization of the simulation, both through *ParaView*<sup>9</sup> [1].

<sup>7</sup>Every “function” is naturally a “region”, but the latter is a broader concept and includes any user-defined aggregation of code lines, which is then assigned a name. It could be used e.g. to aggregate all instructions pertaining to the main solver (time-step) loop.

<sup>8</sup>An open-source “software for manipulating and displaying scientific data” [tool’s website].

<sup>9</sup>An open-source “multi-platform data analysis and visualization application” [tool’s website].

```

int main(int argc, char **argv)
{
    MPI_Init(& argc, & argv);
    #ifdef USE_CATALYST
        initialize_coprocessor_();
    #endif

    // STARTING PROCEDURES...

    #ifdef CATALYST_SCOREP
        // tell the plugin that the time-step loop is about to start
        cat_sco_initialize_();
    #endif
    // MAIN SOLVER LOOP
    for (int time_step = 0; time_step < num_time_steps; time_step++)
    {
        // COMPUTATIONS...

        #ifdef USE_CATALYST
            run_coprocessor_(time_step, time_value, ...);
        #endif
        #ifdef CATALYST_SCOREP
            // tell the plugin to process the current time step
            cat_sco_run_(time_step, time_value);
        #endif
    }
    #ifdef CATALYST_SCOREP
        // tell the plugin that the time-step loop is over
        cat_sco_finalize_();
    #endif

    // ENDING PROCEDURES...

    #ifdef USE_CATALYST
        finalize_coprocessor_();
    #endif
    MPI_Finalize();
    return 0;
}

```

**Figure 3.** Illustrative example of changes needed in a simulation code due to Catalyst (blue) and then due to the plugin (violet)

## 2.2. Combining both Tools

In our previous paper [2], we have introduced a Score-P plugin, which allows performance measurements for an arbitrary number of manually selected code regions to be mapped to the simulation original geometry. In this paper, we are extending our software to *pipeline* (i.e. send for visualization) also communication data (messages exchanged between ranks) on top of the CFD mesh. The plugin must be activated at run-time through an environment variable (`export SCOREP_SUBSTRATE_PLUGINS=Catalyst`), but works independently of Score-P's *profiling* or *tracing* modes being actually on or off. Like Catalyst, it requires three function calls (*initialize*, *run* and *finalize*) to be inserted in the source code, illustrated in [violet](#) at Fig. 3. Additionally, a call must be placed before each function to be pipelined:

```
#ifdef CATALYST_SCOREP
    ! send the following region's measurements to ParaView
    CALL cat_sco_pipeline_next_()
#endif

    CALL desired_function(argument_1, argument_2...)
```

**Figure 4.** Illustrative example of the call to tell the plugin to show the upcoming function's measurements in ParaView

The above layout ensures that the desired function will be captured when executed at that specific moment and not in others (if the same routine is called multiple times – with different inputs – throughout the code, as it is usual for CFD simulations). The selected functions may be nested. This is not needed for the new feature of our tool (show communication on the simulation geometry), as the instrumentation of MPI routines is done independently at run-time (see Section 2.1.1 above).

Finally, the user needs to add a small piece of code into its simulation Catalyst adapter, in order for the plugin-generated variables to be pipelined, as shown on Fig. 5.<sup>10</sup> The first part is related to the selected regions inside the simulation code (the original feature of our tool); it contains two vectors since for each selected region the plugin will generate two variables (which correspond to the two basic measurements made by Score-P, as explained in Section 2.1.1 above). The second part refers to the tracking of communication between MPI ranks (the new feature of our tool); it also contains two vectors for each of the supported calls (MPI.Send, Isend, Get and Put), one to store the *amount of times* that specific call was made (inside the time-step), another to store the total *amount of bytes* transported through those calls.

## 3. Evaluation

### 3.1. Settings

*Hydra* is Rolls-Royce's in-house CFD code [10], based on a preconditioned time marching of the Reynolds-averaged Navier-Stokes (RANS) equations. They are discretized in space us-

<sup>10</sup>In order for the plugin to work with simulation codes written in C or Fortran, its three main calls have name mangling and no namespaces. However, given VTK requires C++ features, a Catalyst adapter needs to be written in C++, hence the plugin calls shown on Fig. 5 are free from such restrictions.

```

#ifdef CATALYST_SCOREP
    // Related to the selected regions
    std::vector<vtkNew<vtkUnsignedIntArray> >freq(cat_sco::meas::get_size());
    std::vector<vtkNew<vtkDoubleArray > >time(cat_sco::meas::get_size());

    for (std::size_t i = 0; i < cat_sco::meas::get_size(); ++i)
    {
        freq[i] -> SetName( (cat_sco::meas::get_name(i) + " : freq").c_str() );
        time[i] -> SetName( (cat_sco::meas::get_name(i) + " : time").c_str() );
        freq[i] -> SetNumberOfComponents(1);
        time[i] -> SetNumberOfComponents(1);
        freq[i] -> SetNumberOfTuples(vtk_grid->GetNumberOfPoints() );
        time[i] -> SetNumberOfTuples(vtk_grid->GetNumberOfPoints() );
        freq[i] -> FillTypedComponent(0, cat_sco::meas::get_counter(i));
        time[i] -> FillTypedComponent(0, cat_sco::meas::get_time (i));

        vtk_grid -> GetPointData() -> AddArray(freq[i].GetPointer() );
        vtk_grid -> GetPointData() -> AddArray(time[i].GetPointer() );
    }

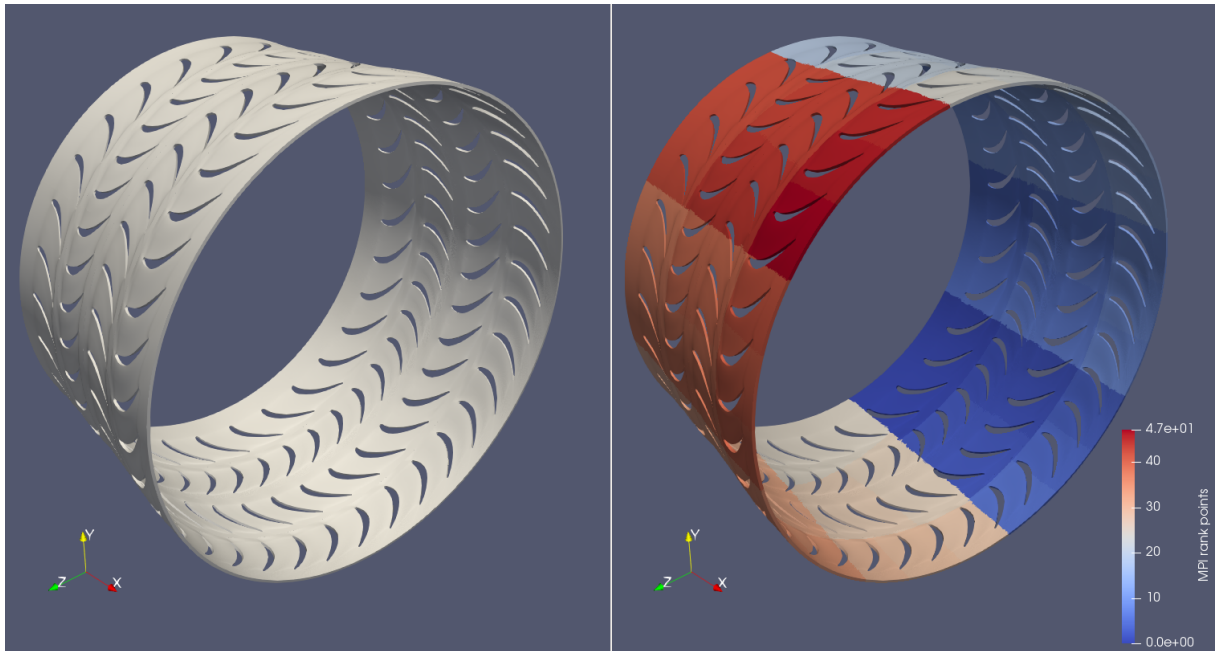
    // Related to communication
    std::vector<vtkNew<vtkUnsignedIntArray > >counter(cat_sco::comm::get_size());
    std::vector<vtkNew<vtkUnsignedLongArray> >bytes (cat_sco::comm::get_size());
    std::stringstream name;

    for (std::size_t i = 0; i < cat_sco::comm::get_size(); ++i)
    {
        name << "MPI_Put to " << i;
        counter[i] -> SetName( (name.str() + " : counter").c_str() );
        bytes [i] -> SetName( (name.str() + " : bytes" ).c_str() );
        counter[i] -> SetNumberOfComponents(1);
        bytes [i] -> SetNumberOfComponents(1);
        counter[i] -> SetNumberOfTuples(vtk_grid->GetNumberOfPoints() );
        bytes [i] -> SetNumberOfTuples(vtk_grid->GetNumberOfPoints() );
        counter[i] -> FillTypedComponent(0, cat_sco::comm::get_counter_put(i));
        bytes [i] -> FillTypedComponent(0, cat_sco::comm::get_bytes_put (i));

        vtk_grid -> GetPointData() -> AddArray(counter[i].GetPointer() );
        vtk_grid -> GetPointData() -> AddArray(bytes [i].GetPointer() );
        name.str(""); name.clear();
    }
#endif

```

**Figure 5.** Illustrative example of the addition needed in the simulation Catalyst adapter due to the plugin



**Figure 6.** Geometry used in the simulations (left) and its partitioning among processes for parallel execution (right)

ing a second-order, edge-based finite volume scheme with a multistage, explicit Runge-Kutta scheme as a steady time marching approach. Multigrid and local time-stepping acceleration techniques are used to improve steady-state convergence [8]. Figure 6 shows the test-case selected for this paper: it represents a simplified (single cell thickness),  $360^\circ$  experimental grid of two turbine stages in an aircraft engine, discretized through roughly 1 million points. Unsteady RANS calculations have been made with second-order, time-accurate dual time-stepping. Turbulence modelling was based on standard 2-equation closures. Preliminary analyses with Score-P and Cube revealed two code functions to be especially time-consuming: *iflux\_edge* and *vflux\_edge* (both mesh-related); they were selected for pipelining.

The simulations have been done using two entire *Haswell* nodes of Dresden University’s HPC cluster (Taurus), each with 24 ranks (i.e. pure MPI, no OpenMP), one per core and with the entire core memory (2583 MB) available. Processes are pinned by default. Figure 6 shows the domain partitioning among the ranks, done in a geometric fashion – which ensures a similar number of grid points between each sub-domain<sup>11</sup> – and not subject to any stochastic variance.

One full engine shaft rotation was simulated, comprised of 200 time-steps (i.e. one per  $1.8^\circ$ ), each internally converged through 40 iteration steps. Catalyst was generating post-mortem files every 20<sup>th</sup> time-step (i.e. every  $36^\circ$ ), what led to 10 stage pictures by the end of the simulation.

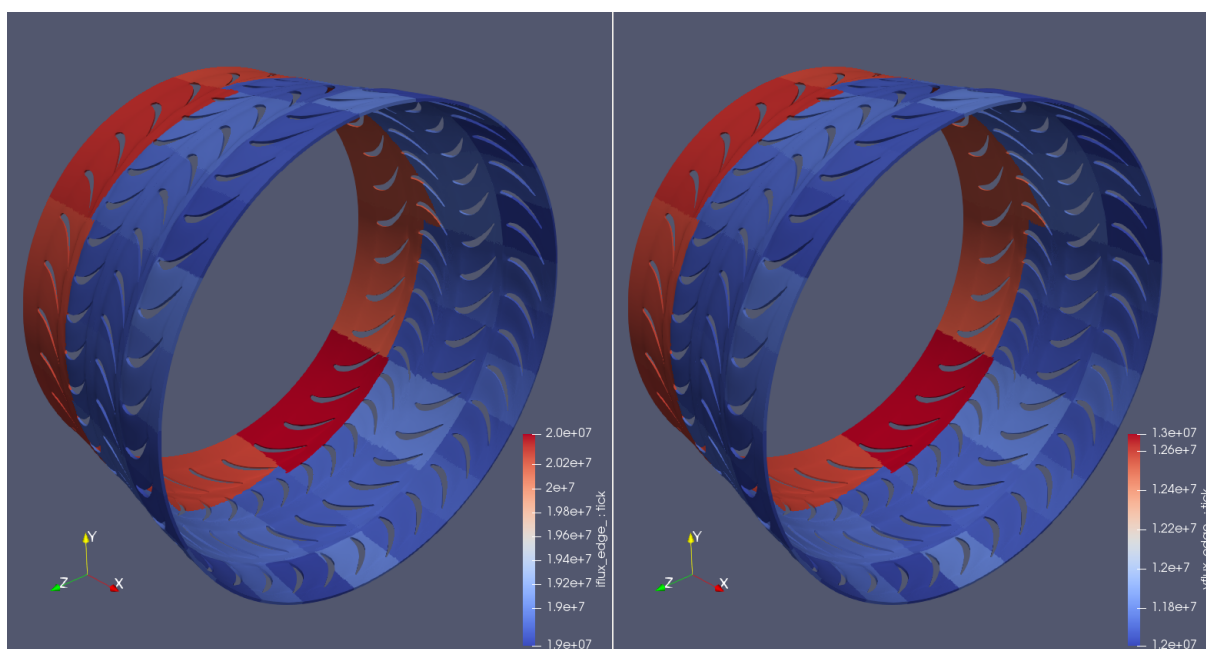
Finally, everything was built / tested with release 2018a of Intel<sup>®</sup> compilers in association with versions 4.0 of Score-P and 5.5.2 of ParaView.

## 3.2. Results

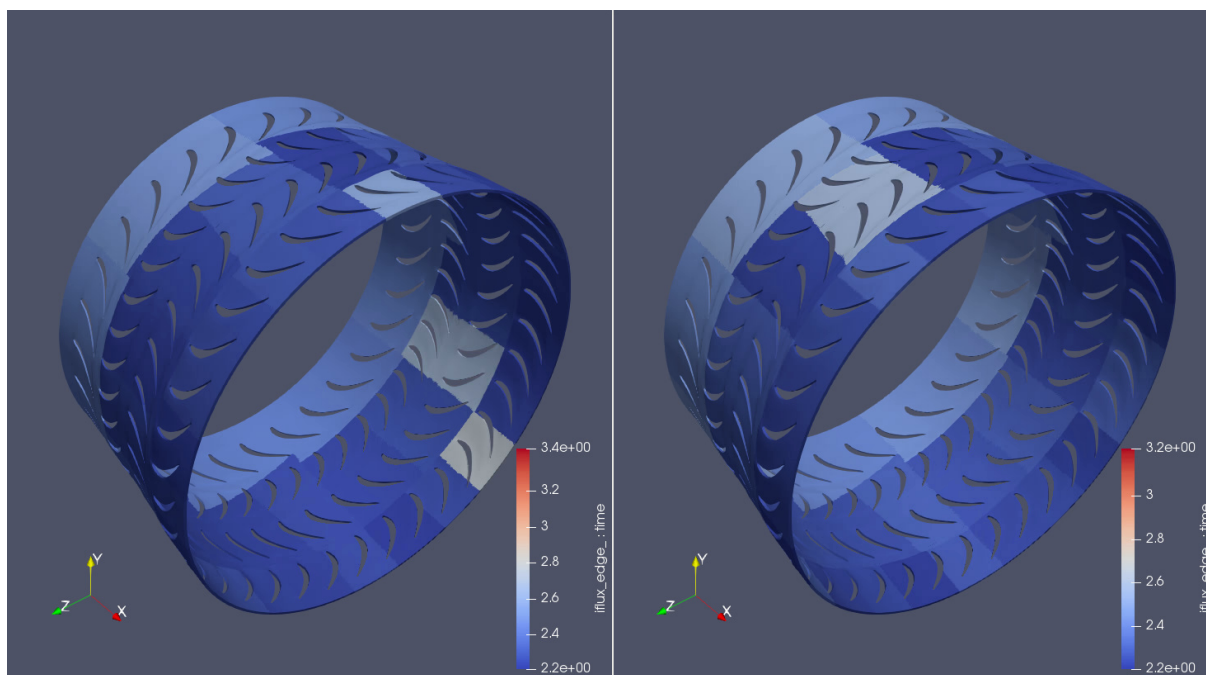
### 3.2.1. Original feature – manually selected code regions

Figure 7 shows the amount of executions, per time-step, of the two selected regions; it is constant in every time-step and not subject to any stochastic variance. From the picture it

<sup>11</sup>It does not look so in the picture because the grid gets finer in the  $x$  direction.



**Figure 7.** Amount of executions, per time-step, of two selected code functions (*iflux\_edge* on the left, *vflux\_edge* on the right) in the test-case

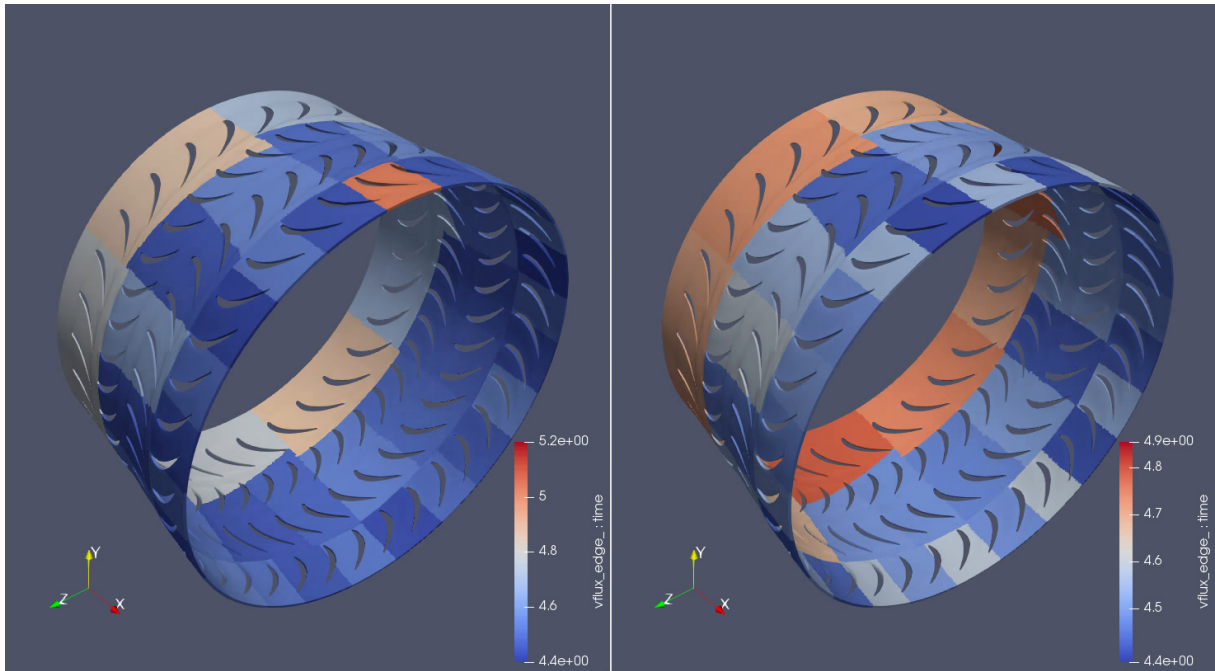


**Figure 8.** Total time spent in function *iflux\_edge*, in an arbitrary time-step, on two consecutive runs of the test-case

is visible that ensuring a similar number of mesh *points* between the sub-domains does not necessarily mean an equally similar number of *edges*,<sup>12</sup> as both functions are applied at the edge level and their amount of executions differ up to 1 million times (every time-step) between maximum and minimum among the ranks. There is a clear bias towards overloading the sub-domains closer to the turbine inlet (the air flows in the positive  $x$  direction), plus the creation of a chess-like pattern (interleaved areas of higher and lower number of executions).

<sup>12</sup>Hydra works with *unstructured* meshes: grid cells do not need to be of uniform type across the entire domain.





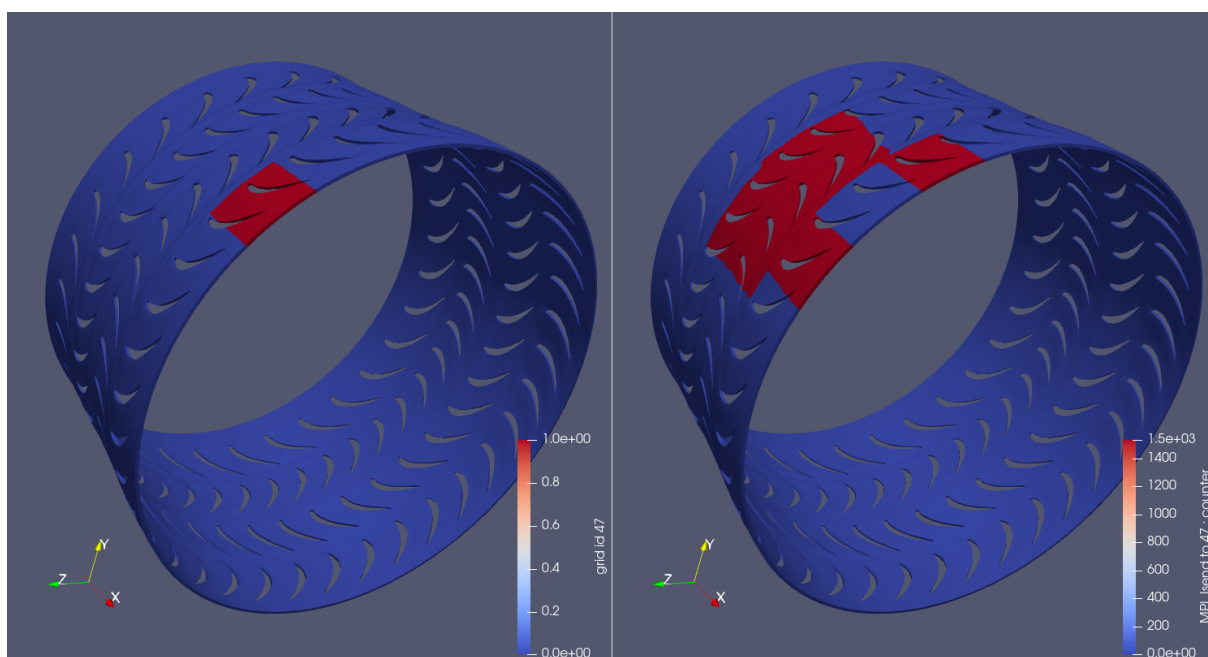
**Figure 9.** Total time spent in function *vflux\_edge*, in an arbitrary time-step, on two consecutive runs of the test-case

The aforementioned distortions indeed reflect on the code performance. Figures 8 and 9 show the *total* – i.e. comprising all executions – time spent (in seconds) in the selected regions in an arbitrary time-step: they change every time-step and are subject to stochastic variance (hence two pictures per function, each referring to the same time-step at consecutive runs of the test-case). Both the bias in the inlet / outlet direction and the chess-like pattern are visible; furthermore, it becomes clear that the load imbalance is stronger in *vflux\_edge*.

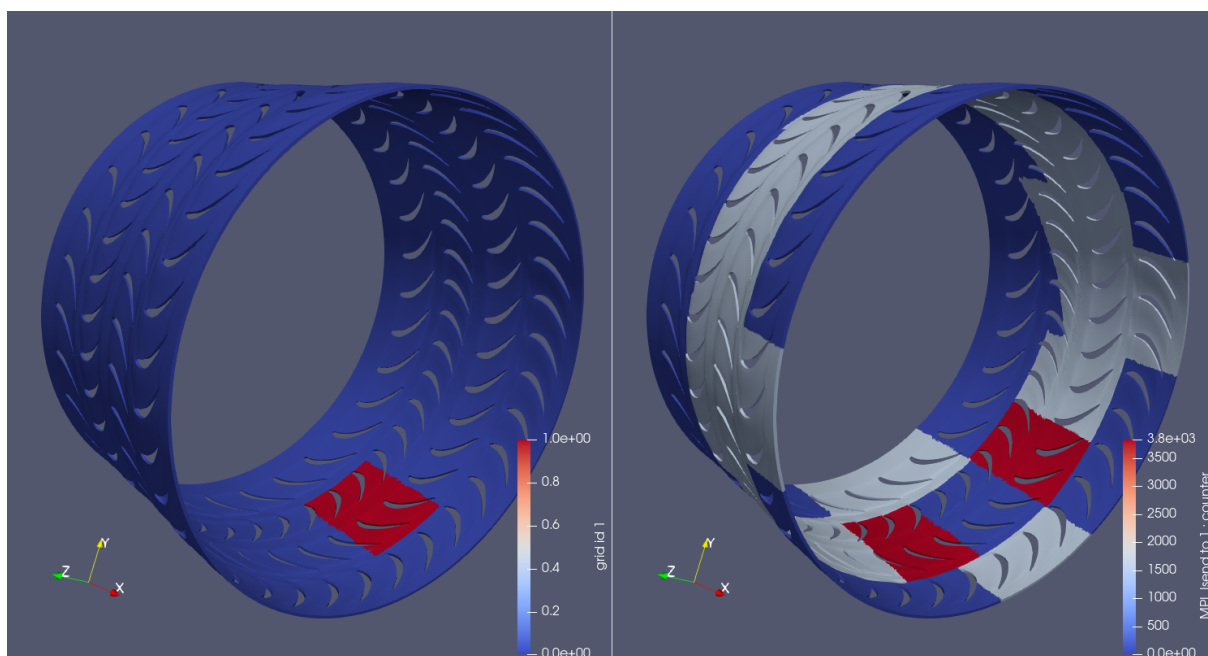
### 3.2.2. New feature – tracking of communications

Mapping communications data to the simulation geometry is a new feature of our plugin and is being presented here for the first time. Figure 10 shows the location of an arbitrary subdomain (left) and those communicating with it (right), colored by the amount of messages sent (in this case, MPI\_Isend calls) in an arbitrary time-step of the test-case. It represents the expected behavior: only the neighbors communicate with the selected rank (the last one, number 47). However, this is not the case for other subdomains within the same simulation: Figure 11 shows the same information as the previous one, but now for rank number 1. Notice how many non-neighbors communicate with it in the selected time-step, and thousand of times indeed. This means an unneeded burden on the simulation run-time and should be avoided. It actually could be avoided, as such non-neighbors are not properly sending any data through those thousands of MPI calls, as revealed by Fig. 12, which corresponds to Fig. 11, but now coloring the sender subdomains by total amount of *bytes* sent at the time-step shown.

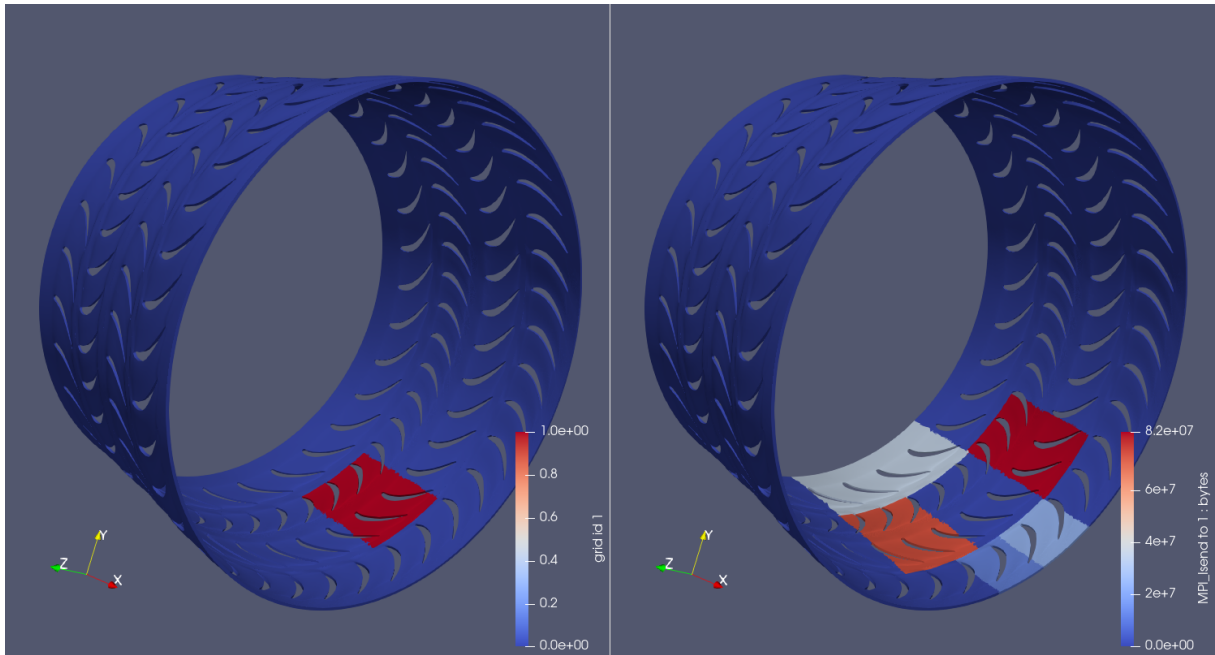
Such conclusions are only made possible thanks to the mapping of communication data back to the simulation geometry; it would indeed be difficult to reach the same insights from the traditional way of showing communication traces (lines crossing each other on a two-dimensional, horizontal bar-chart-like visualization). It has proved the benefits of our tool.



**Figure 10.** Location of an arbitrary subdomain (left) and those communicating with it (right), colored by the amount of messages sent (in this case, MPI\_Isend calls) in an arbitrary time-step of the test-case



**Figure 11.** Location of another arbitrary subdomain (left) and those communicating with it (right), colored by the amount of messages sent (in this case, MPI\_Isend calls) in an arbitrary time-step of the test-case



**Figure 12.** Location of another arbitrary subdomain (left) and those communicating with it (right), colored by the amount of bytes sent (in this case, through MPI\_Isend calls) in an arbitrary time-step of the test-case

## 4. Overhead

Provided we are talking about performance analysis, it is important to analyse the impact of our tool itself on the performance of the instrumented code execution.

### 4.1. Settings

In the following table, the *baseline* results refer to the pure simulation code, running as per the settings presented in Section 3; the numbers given are the average of 5 runs  $\pm 1$  relative standard deviation. The *+ Score-P* results refer to when Score-P is added onto it, running with both profiling and tracing modes deactivated (as neither of them is needed for the plugin to work)<sup>13</sup>. Finally, *++ plugin* refers to when the plugin is also used: running only one *feature* (regions or communication) at a time<sup>14</sup> and on the iterations when there would be generation of output files<sup>15</sup>.

Score-P has been always applied with the `--nocompiler` option. When the plugin is used to show communication between ranks, this option will be enough, as no instrumentation (manual or automatic) is needed when only MPI calls are being tracked. On the other hand, when the goal is to measure code regions, the instrumentation overhead is considerably higher, as every single function inside the simulation code is a potential candidate for the analysis (as opposed to when tracking communications, when only MPI-related calls are intercepted). In this case,

<sup>13</sup>If present, there would be at the end of the simulation, apart from the simulation output files, those generated by Score-P for visualization in Cube (profiling mode) or Vampir (tracing mode). Their generation can co-exist with the plugin execution, but it is not recommended: the overheads sum up.

<sup>14</sup>The plugin can perfectly run in all its features simultaneously. However, this is not recommended: the overheads sum up.

<sup>15</sup>Given the simulation was not being visualized live in ParaView, there was no need to let the plugin work in time-steps when no data would be saved to disk.

```

#ifdef SCOREP_USER
#include "scorep/SCOREP_User.inc"
#endif

! {...}
subroutine IFLUX_EDGE(...)
  implicit none
#ifdef SCOREP_USER
  SCOREP_USER_REGION_DEFINE( iflux_region )
#endif

! {variable declarations}
#ifdef SCOREP_USER
  if(MODULO(time_step, 20) == 0 .OR. time_step == 1) then
    SCOREP_USER_REGION_BEGIN(iflux_region, "iflux_edge",
&    SCOREP_USER_REGION_TYPE_COMMON)
  endif
#endif

! {function body}
#ifdef SCOREP_USER
  if(MODULO(time_step, 20) == 0 .OR. time_step == 1) then
    SCOREP_USER_REGION_END( iflux_region )
  endif
#endif

  return
end

```

**Figure 13.** Example of a manual (user-defined) code instrumentation with Score-P; the optional `if` clauses ensure measurements are collected only at the desired time-steps

it is necessary to add the `--user` Score-P compile flag and manually instrument the simulation code (i.e. only the desired regions were visible to Score-P). This is achieved by means of an intervention as illustrated in Fig. 13: `if MODULO...` additionally ensures measurements are collected solely when there will be generation of output files and at time-step 1 – the reason for it is that Catalyst runs even when there is no post-mortem files being saved to disk (as the user may be visualizing the simulation live) and the first time-step is of special importance, as all data arrays must be defined then (i.e. the (dis)appearance of variables in later time-steps is not allowed)<sup>16</sup>. Finally, when measuring code functions, interception of MPI-related calls has been turned off at run-time<sup>17</sup>.

## 4.2. Results

Table 1 shows the impact of the proposed plugin on the test-case performance. The memory section refers to the *peak* memory consumption per rank, reached *somewhen* during the

---

<sup>16</sup>Hence, in the end there were two narrowing factors for Score-P: the *spacial* (i.e. accompany only the desired functions) and the *temporal* (accompany only at the desired time-steps) ones.

<sup>17</sup>By means of the `SCOREP_MPI_ENABLE_GROUPS` environment variable (see Sec. 2.1.1 above).

**Table 1.** Overhead results of the plugin on the test-case’s performance

	running time			memory (MB)		
	++ plugin	+ Score-P	baseline	++ plugin	+ Score-P	baseline
<b>regions</b>	47m12s (6%)	46m30s (5%)	44m20s $\pm$ 2%	405 (-4%)	410 (-3%)	423 $\pm$ 1%
<b>comm.</b>	49m21s (11%)	46m38s (5%)	44m20s $\pm$ 2%	468 (11%)	410 (-3%)	423 $\pm$ 1%

simulation; it neither means that *all* ranks need that much memory (simultaneously or not), nor that the memory consumption is like that during the *entire* simulation. Score-P’s individual footprint is so small that it lies within the statistical margin of oscillation of the value itself; the same applies to the plugin footprint when measuring the two code regions. Tracking communications, on the other hand, adds many more data arrays to ParaView (two per rank per type of communication call), hence the associated overhead is higher.

The run-time overhead, in its turn, is more critical. But fortunately it lies within acceptable thresholds. The regions feature has again been less burdensome, but that has to do with how many regions are being intercepted within the source code (here, only two of them).

## Conclusions

In this paper, we have extended our software that allows mapping parallel performance data back to the simulation geometry, by means of (combining) the code instrumenter *Score-P* and the in situ library *Catalyst*, resulting on three-dimensional, time-stepped (framed) visualizations in the graphical program *ParaView*. The tool, which takes the form of a Score-P plugin, is capable of matching to the domain mesh (e.g. an aircraft engine):

- measurements for an arbitrary number of manually selected code regions – original feature of the tool, introduced in our previous paper [2] and here revisited (in a bigger test-case);
- communication data (messages exchanged between MPI ranks) – new feature of the tool, presented here for the first time.

All that is based exclusively on open-source dependencies. The tool source code is available at <https://gitlab.hrz.tu-chemnitz.de/alves-tu-dresden.de/catalyst-score-p-plugin>.

The advantage of using ParaView as visualization software comes to all the resources already available in – and experience accumulated by – it after decades of continuous development. Visualization techniques are usually not the specialization field of researchers working with code performance: it is more reasonable to take advantage of the currently available graphic programs than attempting – from scratch – to equip the existing profiling tools with their own GUIs. In this threshold, the developed plugin makes load imbalances and communication inefficiencies easier to identify. It works independently of Score-P’s *profiling* or *tracing* modes and with either *automatic* or *manual* code instrumentation. Finally, like Catalyst itself, its output frequency (when doing post-mortem analyses) is adjustable at run-time (through the plugin input file).

## Future Work

We plan to continue this work in distinct directions:

*More extensive evaluation cases.*

To run the plugin in bigger test-cases, as the difficulty in matching each parallel region id number (the MPI rank) with its respective grid part (hence the benefit of matching performance data back to the simulation mesh) increases with scaling. Concomitantly, to run the plugin in test-cases which comprise regions with distinct flow physics, when the computational load becomes less dependent on the number of points / cells per domain and more dependent on the flow features themselves (given their non-uniform occurrence): chemical reactions in the combustion chamber, shock waves in the inlet / outlet (at the supersonic flow regime), air dissociation in the free-stream / inlet (at the hypersonic flow regime) etc.

*Develop new visualization schemes for performance data.*

To take advantage of the many filters available in ParaView for the benefit of the code optimization branch, e.g. by recreating in it the statistical analysis – display of *average* and *standard deviation* between the threads/ranks measurements – typically available in performance tools.

## Acknowledgments

The authors would like to thank: Rolls-Royce Germany (especially Marcus Meyer, Axel Gerstenberger, Jan Suhrmann & Paolo Adami), for providing both the industrial CFD code and its test-case for this paper, what has been done in the context of BMWi research project *Prestige* (FKZ 20T1716A); Kitware (also part of *Prestige*; especially Mathieu Westphal & Nicolas Vuaille), for the support on implementing the Catalyst adapter; the Score-P support and development team (especially Bert Wesarg), for the assistance with the plugin API; and the Taurus admins (especially Maik Schmidt), for the help with the supercomputer.





*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large data visualization. The visualization handbook 717 (2005)
2. Alves, R.F.C., Knüpfer, A.: In situ visualization of performance-related data in parallel CFD applications. In: Schwardmann, U., Boehme, C., B. Heras, D., et al. (eds.) Euro-Par 2019: Parallel Processing Workshops. pp. 400–412. Springer International Publishing, Cham (2020), DOI: 10.1007/978-3-030-48340-1\_31
3. Ayachit, U., Bauer, A., Geveci, B., et al.: Paraview Catalyst: Enabling in situ data analysis and visualization. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV2015, Austin, TX, USA. pp. 25–29. ACM (2015), DOI: 10.1145/2828612.2828624
4. Bremer, P.T., Mohr, B., Pascucci, V., et al.: Connecting Performance Analysis and Visualization. Dagstuhl Manifestos 5(1), 1–24 (2015), DOI: 10.4230/DagMan.5.1.1

5. Huck, K.A., Potter, K., Jacobsen, D.W., et al.: Linking performance data into scientific visualization tools. In: 2014 First Workshop on Visual Performance Analysis. pp. 50–57. IEEE (2014), DOI: 10.1109/VPA.2014.9
6. Husain, B., Giménez, A., Levine, J.A., et al.: Relating memory performance data to application domain data using an integration API. In: Proceedings of the 2nd Workshop on Visual Performance Analysis, VPA '15, Austin, Texas. ACM (2015), DOI: 10.1145/2835238.2835243
7. Isaacs, K.E., Giménez, A., Jusufi, I., et al.: State of the Art of Performance Visualization. In: Borgo, R., Maciejewski, R., Viola, I. (eds.) EuroVis - STARS. The Eurographics Association (2014), DOI: 10.2312/eurovisstar.20141177
8. Khanal, B., He, L., Northall, J., et al.: Analysis of Radial Migration of Hot-Streak in Swirling Flow Through High-Pressure Turbine Stage. *Journal of Turbomachinery* 135(4) (2013), DOI: 10.1115/1.4007505
9. Knüpfer, A., Rössel, C., Mey, D.a., et al.: Score-P: A joint performance measurement runtime infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Brunst, H., Müller, M.S., Nagel, W.E., et al. (eds.) *Tools for High Performance Computing 2011*. pp. 79–91. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), DOI: 10.1007/978-3-642-31476-6\_7
10. Lapworth, L.: Hydra-CFD: a framework for collaborative CFD development. In: *International Conference on Scientific and Engineering Computation, IC-SEC*, June, Singapore. vol. 30 (2004), [https://www.researchgate.net/publication/316171819\\_HYDRA-CFD\\_A\\_Framework\\_for\\_Collaborative\\_CFD\\_Development](https://www.researchgate.net/publication/316171819_HYDRA-CFD_A_Framework_for_Collaborative_CFD_Development)
11. Schöne, R., Tschüter, R., Ilsche, T., et al.: Extending the functionality of Score-P through plugins: Interfaces and use cases. In: Niethammer, C., Gracia, J., Hilbrich, T., et al. (eds.) *Tools for High Performance Computing 2016*. pp. 59–82. Springer International Publishing, Cham (2017), DOI: 10.1007/978-3-319-56702-0\_4
12. Vierjahn, T., Kuhlen, T.W., Müller, M.S., et al.: Visualizing performance data with respect to the simulated geometry. In: Di Napoli, E., Hermanns, M.A., Iliev, H., et al. (eds.) *High-Performance Scientific Computing*. pp. 212–222. Springer International Publishing, Cham (2017), DOI: 10.1007/978-3-319-53862-4\_18
13. Wood, C., Larsen, M., Gimenez, A., et al.: Projecting performance data over simulation geometry using SOSflow and ALPINE. In: Bhatele, A., Boehme, D., Levine, J.A., et al. (eds.) *Programming and Performance Visualization Tools*. pp. 201–218. Springer International Publishing, Cham (2019), DOI: 10.1007/978-3-030-17872-7\_12

# Improving Quantum Annealing Performance on Embedded Problems

Michael R. Zielewski<sup>1</sup> , Mulya Agung<sup>2</sup> , Ryusuke Egawa<sup>3</sup> ,  
Hiroyuki Takizawa<sup>1,2</sup> 

© The Authors 2020. This paper is published with open access at SuperFri.org

Recently, many researchers have been investigating quantum annealing as a solver for real-world combinatorial optimization problems. However, due to the format of problems that quantum annealing solves and the structure of the physical annealer, these problems often require additional setup prior to solving. We study how these setup steps affect performance and provide insight into the interplay among them using the job-shop scheduling problem for our evaluation. We show that the empirical probability of success is highly sensitive to problem setup, and that excess variables and large embeddings reduce performance. We then show that certain problem instances are unable to be solved without the use of additional post-processing methods. Finally, we investigate the effect of pausing during the anneal. Our results show that pausing within a certain time window can improve the probability of success, which is consistent with other work. However, we also show that the performance improvement due to pausing can be masked depending on properties of the embedding, and thus, special care must be taken for embedded problems.

*Keywords:* quantum annealing, quantum computer, job-shop scheduling, combinatorial optimization.

## Introduction

As quantum computing devices are increasing in size and availability, quantum computing is experiencing a surge of interest as a method to efficiently solve problems that are otherwise impractical for classical computers or for which even small speedups are desirable. This interest is justified by existing quantum algorithms that improve upon classical ones, such as Shor's algorithm for integer factorization [31] and Grover's search algorithm [14]. One type of quantum computing that excels at solving combinatorial optimization problems is quantum annealing (QA). By exploiting fundamental quantum mechanical properties, such as tunneling and superposition, QA has high potential to efficiently find high quality solutions [18]. While early results [15, 20] may not have seemed promising for QA over classical solvers, later results [2, 12, 24] were more promising, and indicated that harder benchmark problems reveal the power of QA over classical solvers. QA achieves computation by preparing a system in an initial quantum configuration, and slowly evolving the system toward a final configuration, which represents the target problem. According to the adiabatic theorem, if the system is evolved slowly enough, it will remain in the ground state and the final qubit values represent a solution to the target problem [13]. However, due to problem properties, thermal excitations, noise, and other factors, the system does not always remain in the ground state; thus, solutions may not be found. Performance and usability are further impacted by preparatory steps required not only by QA, but also by the current physical implementation of QA.

One of the major challenges in using QA is converting problems to quadratic unconstrained binary optimization (QUBO) form, which is the native problem that QA solves. QUBO form requires that all variables take binary values, and that all variable terms are of degree two or less.

<sup>1</sup>Graduate School of Information Sciences, Tohoku University, Sendai, Japan

<sup>2</sup>Cyberscience Center, Tohoku University, Sendai, Japan

<sup>3</sup>Department of Information and Communication Engineering, Tokyo Denki University, Tokyo, Japan



For some types of problems, such as integer programming problems, conversion of non-binary variables to binary variables can be achieved through techniques such as one-hot encoding, yet will result in a significantly higher number of variables in the QUBO. For other types of problems that already use binary variables, such as the satisfiability problem, additional variables may be introduced when reducing terms having a degree greater than two. Therefore, it is clear that conversion to QUBO form often results in a problem containing significantly more variables than the original representation did.

A second challenge associated with current QA arises from the physical quantum processing unit (QPU) that implements QA. Commercially available QPUs have limited connectivity between qubits, and impose similar constraints on qubits in QUBOs. Limited connectivity can be remedied through minor embedding, which is a process that produces a mapping of a QUBO to the QPU, while allowing one logical qubit to be represented by a “chain” of multiple physical qubits. A problem that requires embedding is known as an embedded problem. As many applications, such as the one considered in this work, contain multiple variables that must be represented by qubit chains, the number of qubits in an embedding can be significantly larger than in its corresponding QUBO. This effectively reduces the maximum sizes of problems that can be solved by the QPU, especially for embedded problems with variables that share many constraints and have high connectivity.

The effects that problem formulation and embedding have on performance are not well studied. The purpose of this work is to investigate these effects, determine methods through which performance can be maximized, and investigate the interplay between them. We select scheduling problems for our evaluation. Scheduling is one of the most ubiquitous types of problems in optimization and has applications in many fields. The standard form of these problems is finding an assignment of tasks to resources that satisfies problem constraints [6]; however, real world applications require domain specific variants [16, 29, 32, 35]. In this work, we select the standard  $n \times m$  job-shop scheduling problem (JSP) to evaluate the performance of QA. One characteristic of the JSP that makes it a suitable candidate for evaluating the performance of QA is that it is NP-hard, and often requires the use of heuristic solvers. Furthermore, binarization of the JSP results in extra variables, which can have a significant impact on performance. Additionally, the structure and connectivity of the JSP necessitates embedding, which allows us to evaluate the performance of a multitude of unique embeddings. Seeking methods to improve performance, we also evaluate post-processing techniques and modified anneal schedules that include a pause. Finally, we conclude with a suggestion of how the considered methods can be combined to address the above challenges and achieve high performance with QA on embedded problems. The main contributions of this work can be summarized as follows:

- We investigate the impacts of problem formulation and embedding on performance;
- We provide methods to improve performance and show the interplay between them;
- We show that the effects of anneal pausing can be masked with a poor selection of embedding, and provide an explanation as to why this occurs; and
- We suggest a combined approach to achieve high performance on embedded problems.

The rest of this paper is organized as follows. Section 1 provides background on QA and its current implementation, and reviews a JSP formulation for QA. Then, in Section 2, we describe in detail our methods for variable reduction, embedding, post-processing, and anneal schedule modification. Section 3 contains our evaluation setup, results, and analysis. Related work is discussed in Section 4. The final section contains our conclusions and potential future direction.

## 1. Background

In this section, we first describe QA in detail. We then introduce our target problem, the JSP, and provide details on how it can be formulated for QA.

### 1.1. Quantum Annealing

QA is a quantum mechanical metaheuristic for minimizing combinatorial optimization problems. System evolution follows a time-dependent Hamiltonian of the form

$$H(s) = A(s)H_I + B(s)H_F, \quad (1)$$

where  $A$  and  $B$  are time-dependent coefficients,  $H_I$  is the initial Hamiltonian,  $H_F$  is the final or problem Hamiltonian, and  $s$  is the re-scaled annealing time taking values in the range  $[0, 1]$ . In a system of  $N$  qubits,  $H_I$  and  $H_F$  are given by

$$H_I = \sum_{i=1}^N \sigma_x^i, \quad (2)$$

and

$$H_F = \sum_i^N h_i \sigma_z^i + \sum_i^N \sum_{j=i+1}^N J_{i,j} \sigma_z^i \sigma_z^j, \quad (3)$$

where  $\sigma_z^i$  and  $\sigma_z^j$  are the Pauli matrices operating on qubits  $i$  and  $j$ , and  $h$  and  $J$  are local biases and coupling strengths that encode the problem to be solved.

System evolution starts at  $s = 0$  in the ground state of  $H_I$ , where  $A$  is large and  $B = 0$ . Tunneling strength, which is determined by  $A$ , is initially strong and analogous to the temperature term in the classical metaheuristic simulated annealing [21]. As system evolution progresses,  $A$  monotonically decreases to 0 and  $B$  monotonically increases, introducing  $H_F$ . According to the adiabatic theorem, the system will remain in the ground state provided that evolution is sufficiently slow with respect to the minimum gap, that is, the difference in energy between the two lowest energy states of the system [13].

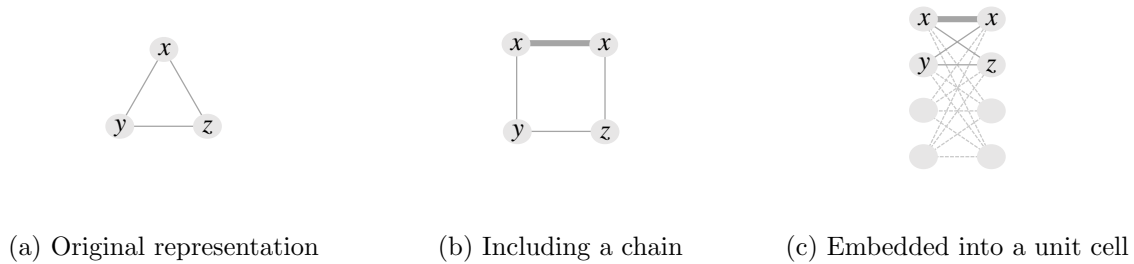
When system evolution ceases at  $s = 1$ , the system is in a classical state described by the Ising model

$$E(s) = \sum_i^N h_i s_i + \sum_i^N \sum_{j=i+1}^N J_{i,j} s_i s_j, \quad (4)$$

where  $s_i \in \{\pm 1\}$  are spin variables. Similar to the Ising model, but having binary variables  $x_i \in \{0, 1\}$ , is the QUBO problem of the form

$$E(x) = \sum_i^N \sum_{j=i}^N Q_{i,j} x_i x_j, \quad (5)$$

where  $Q_{i,j}$  is a matrix containing local and quadratic biases. Note that QUBO variables can be converted to Ising variables with the mapping  $s_i = 2x_i - 1$ . Due to the binary values of the QUBO matching those most frequently used in traditional computing, we use QUBO form throughout the rest of this work.



**Figure 1.** An example embedding for a problem consisting of three variables

In QA, there are no limitations placed on the connectivity of qubits; however, in current QPUs implementing QA, qubit connectivity is limited. In the QPU used in this work, the layout of qubits in the QPU follows the  $C_{16}$  Chimera graph, which is a  $16 \times 16$  grid of  $K_{4,4}$  bipartite graphs, termed unit cells. In this graph, vertices, and thus qubits, have a maximum connectivity of six. To solve QUBOs with variables having higher connectivity, or QUBOs that have a structure that does not match that of the QPU, a technique called minor embedding must first be performed. In minor embedding, connectivity between logical variables in the QUBO is achieved by allowing the variables to be represented by a “chain” of multiple physical qubits in the QPU. To ensure that all qubits of a chain take the same value, a strong negative bias is used for intra-chain couplings. Problems that require chaining often use far more qubits in the QPU than variables in their QUBO, which effectively reduces the maximum sizes of problems that can be solved directly on the QPU.

We provide an example problem that requires embedding, as well as a potential embedding for it, in Fig. 1. The problem contains three fully connected variables, and can be represented graphically by a triangle, as shown in Fig. 1a. Note that in the unit cell in Fig. 1c, no three vertices can be used to represent this system. Only by chaining a variable, as is done in Fig. 1b, is it possible to embed the system into the unit cell as shown in Fig. 1c.

## 1.2. Job-shop Scheduling Problem

In this work, we consider the standard  $n \times m$  JSP in which a set of  $n$  jobs  $J$  is to be scheduled on a set of  $m$  machines  $M$  [4]. Each job  $j \in J$  must be processed exactly once by each machine  $r \in M$ . The processing of a job on a machine is called an operation, denoted by  $o_i \in O = \{o_1, \dots, o_{n \times m}\}$ , where every  $o_i$  corresponds to the machine  $r$  on which the operation is to be executed. Each operation has a corresponding positive integer processing duration  $d_i \in D = \{d_1, \dots, d_{n \times m}\}$ . The  $i$ -th set of operations and processing durations corresponds to job  $j_i$ .

We seek a schedule of  $J$  on  $M$  that respects the following three constraints. First, operations of a job must take place in the order defined in  $O$ , and no operation can start until all preceding operations have completed. Second, machines can only execute one operation having a nonzero processing duration at any given time. Lastly, all operations must be scheduled with exactly one starting time and cannot be interrupted during processing. A schedule is valid if none of these constraints are violated.

We define the shortest time in which all operations complete, the optimal makespan of an instance, with  $T$ . In the optimization version of the JSP, we aim to minimize the makespan of a schedule in such a way that the makespan is nearest to the instance’s optimal makespan. In this work, we consider the decision variant of the JSP, in which we only seek a valid schedule.

Before solving any problem with QA, its representation must be made compatible with QUBO form. Typical problem representations may be composed of non-binary variable types, such as integers, which are not directly compatible with QUBO form and require transformation [23]. One common method of integer variable binarization is to use one-hot encoding [26]. In this method, given an integer variable whose values span  $N$  integers, we define  $P = (p_1, \dots, p_N)$  as the sequence of those integer values. A vector of binary variables  $\mathbf{x} = [x_1, \dots, x_N]$ ,  $x_i \in \{0, 1\}$ , is used to represent  $P$  in such a way that if  $x_i = 1$ , the  $i$ -th element of  $P$  is selected. In order to maintain a coherent representation by ensuring that exactly one value is selected, the following constraint is introduced:

$$\sum_i^N x_i = 1. \tag{6}$$

This constraint can then be modeled as the following penalty

$$\left( \sum_i^N x_i - 1 \right)^2. \tag{7}$$

One method of binarizing the JSP is for a binary variable  $x_{i,t}$  to represent the execution of operation  $i$  at time  $t$  [6]. In this work, we use a formulation that was introduced in [36] that has already been used with QA and in which binarization is achieved through variables  $x_{i,t}$  representing the *starting* of operation  $i$  at time  $t$ . One limitation of these methods is that they require  $\tilde{T}$ , an estimation of  $T$  used for limiting the number of binary variables created. An estimate that is less than  $T$  results in a QUBO that cannot solve the JSP instance. On the other hand, overestimating  $T$  produces an excess of binary variables. The penalties applied to these variables, representing their constraints, that are used to construct the QUBO are:

$$\sum_i \left( \sum_t x_{i,t} - 1 \right)^2, \tag{8}$$

$$\sum_n \left( \sum_{\substack{k_{n-1} < i < k_n \\ t+p_i > t'}} x_{i,t} x_{i+1,t'} \right), \tag{9}$$

$$\sum_m \left( \sum_{(i,t,k,t') \in R_m} x_{i,t} x_{k,t'} \right). \tag{10}$$

These penalties can be summarized as follows: Equation (8) penalizes configurations in which operations do not start exactly once, Equation (9) penalizes out of order execution for operations within a job, and Equation (10) penalizes configurations that violate the capacity of a machine. For additional details, the reader is directed to [36].

## 2. Methods

In this section, we introduce the methods that are required for preparing a problem for QA as well as those that will be used to improve the performance of QA on the JSP.

## 2.1. Variable Reduction

To investigate the impacts of the formulation process on the performance of QA, we employ two commonly used software packages for creating QUBOs. The first of these packages is *D-Wave Binary CSP* (DBC). This tool is used to convert constraint satisfaction problems to QUBO form and is developed by D-Wave Systems [8]. The second package is *PyQUBO* (PYQ), developed by Recruit Communications [28]. While the object of this work is not to directly compare these two tools, we observe that the output QUBOs can differ, despite both being constructed from the same penalties and capable of producing correct schedules. For small size problems, DBC and PYQ often output identical QUBOs; however, as problem size increases, DBC outputs larger QUBOs than PYQ. The source of this discrepancy is auxiliary variables, which are only found in QUBOs constructed by DBC. While auxiliary variables are typically introduced to reduce polynomial terms with a degree higher than two, none of the terms in Equations (8), (9), or (10) contain such terms. Thus, the creation of auxiliary variables can be considered a software error that can be corrected. However, as QUBOs produced by DBC are able to successfully solve JSP instances, a comparison of results from the QUBOs of the two tools will help to emphasize the impact that formulation efficiency has on performance.

A second method of variable reduction is variable pruning, which is a formulation dependent method for removing variables that cannot be set in valid solutions. We adopt the variable pruning method from [36]. This method first examines the duration and order of operations within a job. Then, variables are removed if they allow an operation to execute at a time that would result in an invalid schedule. The specific conditions for which a variable is pruned are if it allows an operation to start earlier than the sum of the processing durations of its predecessors, or if it allows an operation to start later than the sum of the processing durations of itself and its successor operations subtracted from  $\tilde{T}$ . Therefore, by identifying the variables which meet these conditions, a set of variables can be pruned.

## 2.2. Re-Embedding

We generate embeddings for QUBOs with the heuristic minor embedding algorithm described in [7], which is the standard algorithm for problems of arbitrary structure. Owing to its heuristic nature, the minor embedding algorithm generates embeddings over a range of sizes where the largest often require double the amount of qubits than the smallest do. The significance of this becomes apparent when considering the number of qubits available in the QPU; if a QUBO is especially large or its variables have connectivity that is difficult to achieve on the QPU, the embedding algorithm may fail to find any embedding. Additionally, we generally seek embeddings that minimize either the total number of qubits required or maximum chain lengths in order to avoid factors that degrade performance, such as early freezeout [3, 7]. In order to find smaller embeddings for QUBOs and to evaluate performance at different embedding sizes, we repeat the embedding process for each QUBO.

## 2.3. Post-Processing

An additional post-processing step is required for problems that require chains of physical qubits. During post-processing, chains are examined to determine the value of the corresponding logical qubit. This step is necessary because despite the strong coupling bias between qubits in a chain, which compels the qubits to take the same value, the values within a chain do not always

match at the end of an anneal. In the case where all qubits within a chain take the same value, the logical qubit also takes that value. However, if the values within a chain span both binary values, the chain is “broken”, and additional logic is required to determine which value the logical qubit should take. We consider two post-processing methods for resolving broken chains: majority voting (MV), which is provided in [9]; and minimizing energy (ME), which is provided in [10]. In the first method, MV, the logical qubit takes on the value that most frequently occurs in the chain. The second method, ME, implements a greedy algorithm that assigns the value resulting in the lowest energy penalty based on linear and quadratic biases associated with the variable. Related work has shown that MV can reduce the penalty of a broken chain, but leads to higher penalties from constraints of neighboring qubits [29]. We expect that ME will result in higher performance since it accounts for neighboring qubits when selecting the value for a broken chain.

## 2.4. Anneal Schedule Modifications

The D-Wave 2000Q provides annealing controls, which are special features that provide additional control over the anneal schedule, beyond specifying the annealing time. These controls have been used to great effect in improving performance [16, 17, 22, 25, 38]. In this work, we focus on modifying the anneal schedule so that it includes a pause, a period of time during which  $A$  and  $B$  remain constant. Marshall et al. [25] have shown that pausing at the right time allows the system to relax to the ground state, and improves success probability by orders of magnitude. A follow-up study performed by Izquierdo et al. [17] evaluated the performance improvement from pausing on embedded problems. However, there is, as yet, no evaluation into the effect that embedding size has on the performance improvement from pausing. In this work, we seek to extend the understanding of pausing and explain the effect of embedding size on pausing. To accomplish this, we perform additional anneals with modified annealing schedules that include a pause, similar to what is done in [17, 25].

## 3. Evaluation and Discussion

This section presents our evaluation configuration, evaluation results, and ends with a discussion of our findings.

### 3.1. Configuration

We solve JSP instances on the D-Wave 2000Q quantum computer. The instances on which we evaluate QA are randomly generated with  $n = m \in \{3, 4, 5, 6, 7\}$  with operation processing durations,  $d_i \in D$ , selected from a subset of  $\{0, 1, 2\}$ . For each instance,  $T$  is found by solving the instance with a classical solver. The formulation described in Section 1.2 is implemented, with  $\tilde{T} = T$ , and a QUBO is generated with both DBC and PYQ. Initial results showed that variable pruning was a necessary procedure for obtaining any valid schedules, regardless of instance size. Thus, variable pruning is performed on each instance and no results are shown for QUBOs whose variables have not been pruned. The minor embedding algorithm was repeated one million times in order to collect embeddings of different sizes for each QUBO. Starting from 102 random JSP instances, QA was performed on a total of 36,805 embeddings. For all samples taken with the annealer, the anneal time was set to 20  $\mu$ s, and 300 anneals were performed, with all other

parameters set to their default values. In the final experiment, in which we introduce a pause into the anneal schedule, the pause duration is set to 100  $\mu$ s. Pause locations were limited to the range [0.25, 0.75], after initial testing revealed that the optimal pause location consistently occurred between these values, and pause locations were selected at intervals of 0.01. We also applied ten spin-reversal transforms, a procedure that is used to reduce the effect of hardware biases on results, and has been shown to be a critical step in analyzing the effects of pausing [17]. A limited set of instances was selected for evaluation, and embeddings for each instance were selected uniformly, based on the number of qubits required for the embedding.

We evaluate the performance of QA using  $P_{success}$ , the observed probability of successfully finding a valid schedule, that is, one that violates no constraints and is the ground state of  $H_F$ . This is a standard metric [16, 17] for evaluating the performance of QA and is defined as

$$P_{success} = \frac{\text{number of valid schedules}}{\text{number of anneals}}. \quad (11)$$

### 3.2. Results

We first show the effect of variable reduction on QUBO size and the embedding process. Figure 2 shows the number of variables in QUBOs generated by DBC and PYQ for JSP instances where  $n = m = 4$ , and  $T$  ranges from 5 to 13. This figure shows the effect described in Section 2.1; DBC and PYQ output QUBOs of equal size for small JSP instances, yet for larger ones, DBC introduces unnecessary auxiliary variables, resulting in larger QUBOs. For the largest instance in this figure — where  $T = 13$  — auxiliary variables comprise 22% of the variables in the DBC QUBO. The effects that this has on the embedding process are shown in Fig. 3, which compares the frequency distributions of embedding sizes, which are approximately normal, for the JSP instances where  $T = 9$  and  $T = 13$  in Fig. 2. When  $T = 9$ , the small amount of auxiliary variables produced by DBC have a relatively small effect on the embedding size distribution, resulting in the distribution being shifted right and thus embeddings being larger. On the other hand, when  $T = 13$ , the significant number of auxiliary variables has a considerable effect on embedding size. Furthermore, the increased number of variables in the QUBO also affects the failure rate of the embedding algorithm. In this figure, the failure rate of the algorithm is visualized by redistributing the percentage of failed embeddings to each embedding size proportionally, based on the percentage of successful embeddings generated for that size. We see that the embedding failure rate is only 25% for the PYQ QUBO, but is as high as 96% for the DBC QUBO. These results show that an efficient formulation is necessary to minimize failed embeddings and embedding size.

Next, we evaluate the performance implications of embedding size on  $P_{success}$ . In Fig. 4,  $P_{success}$  is shown for JSP instances where  $n = m = 4$ , yet having different operation processing times than the instances used for Fig. 2 and Fig. 3. For this series of data, the default post-processing method MV is used. These figures show the effect that unnecessary variables have on  $P_{success}$ ; annealing using embeddings generated from the DBC QUBO produces no valid schedules. Figure 4a also shows that  $P_{success}$  is moderately negatively correlated with embedding size ( $r = -0.69$ ), and thus, fewer valid schedules are found with larger embeddings. This effect is also visible in Fig. 4b, where only the smallest embeddings of the PYQ QUBO have a nonzero  $P_{success}$ . These results emphasize not only the advantages of using smaller embeddings but also the disadvantages of using larger embeddings.

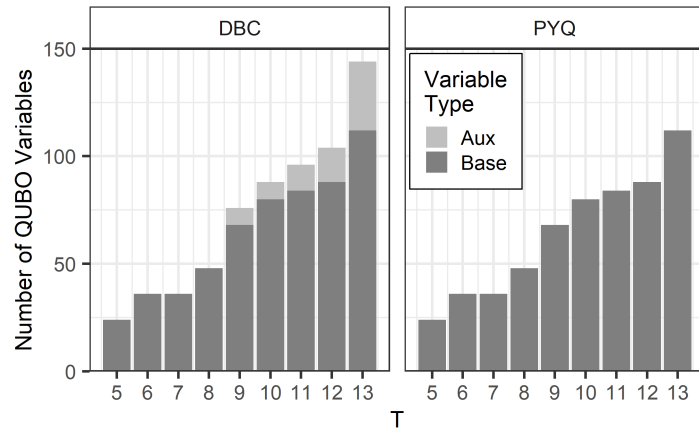


Figure 2. Comparison of the number of variables in QUBOs by tool

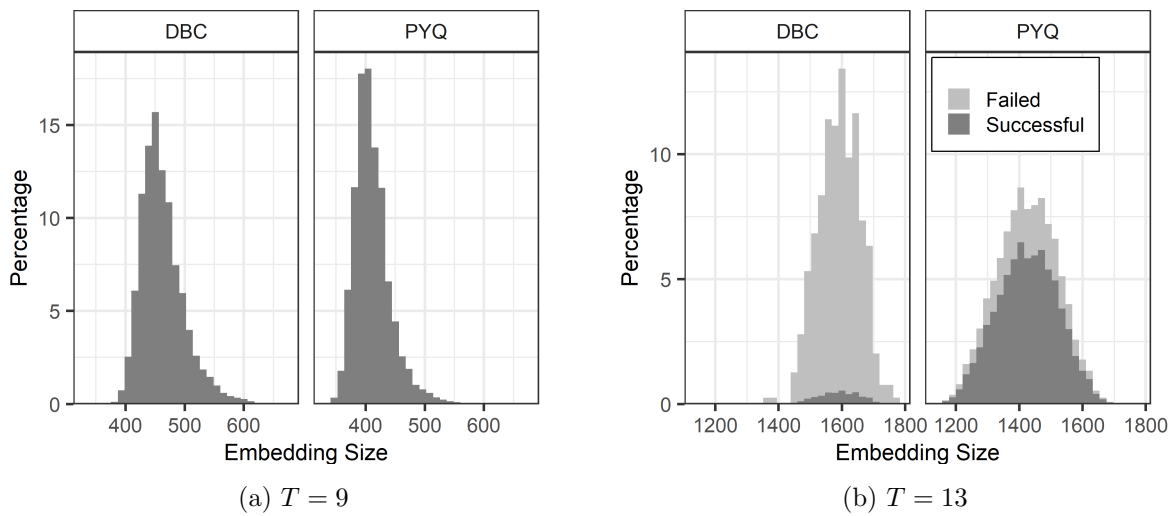


Figure 3. Distributions of embedding sizes for two JSP instances

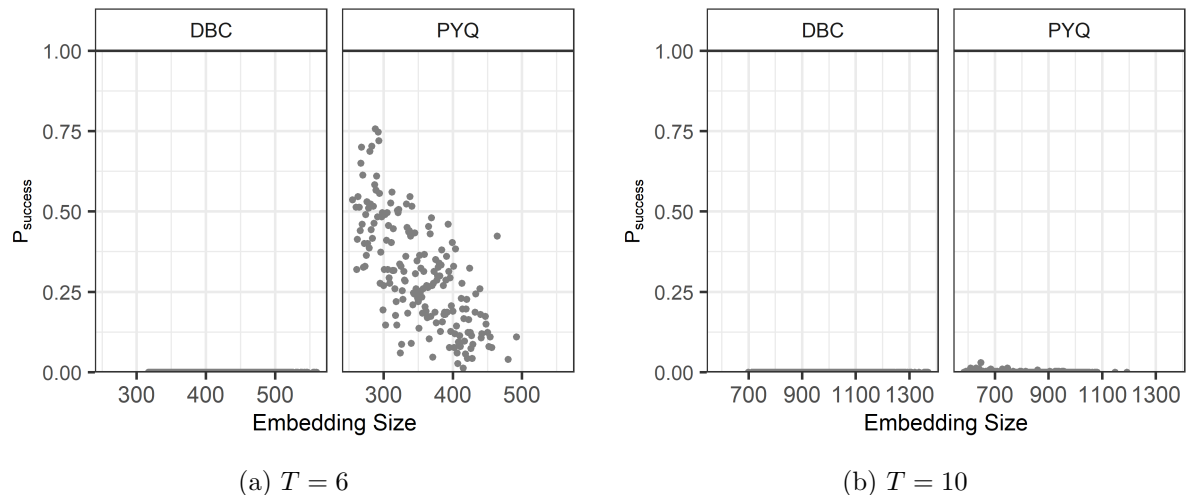
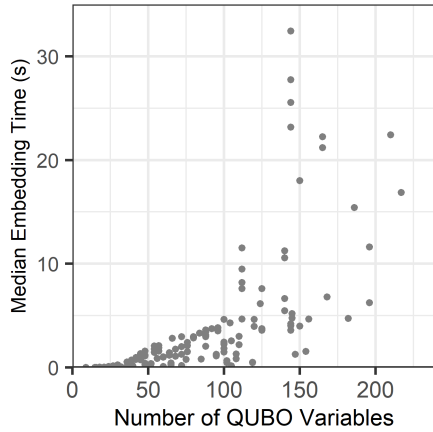
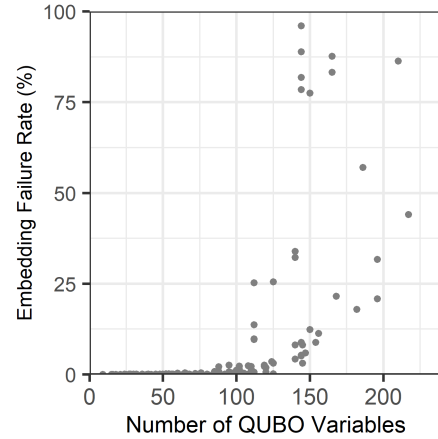


Figure 4. Downward trend in  $P_{success}$

We provide additional statistics on the embedding process in Fig. 5 and Fig. 6, where we show the median embedding time and embedding failure rate, which both increase with QUBO size. We see that for relatively small QUBOs, the cost of re-embedding is low due to short




**Figure 5.** Median embedding times

**Figure 6.** Embedding failure rates

**Table 1.** Summary of post-processing methods

$P_{success}$ comparison	Percentage
MV < ME	33.32
MV > ME	<0.01
MV = ME = 0	56.80
MV = ME $\neq$ 0	9.87

embedding time and low embedding failure rate. On the other hand, for larger QUBOs, the cost of re-embedding is high due to both the long embedding time and high embedding failure rate. Considering these costs alone may suggest that re-embedding is a poor choice for large QUBOs; however, the benefit of re-embedding is highest for large QUBOs. This is shown in the results from the embeddings for PYQ QUBOs in Fig. 4a and Fig. 4b, corresponding to a small and large QUBO, respectively. In the case of the small QUBO, re-embedding found smaller embeddings that resulted in a higher  $P_{success}$ , but any embedding solved the instance as each resulted in a  $P_{success}$  that was greater than zero. For the case of the large QUBO, the smallest embedding found through re-embedding also resulted in a higher  $P_{success}$  than larger embeddings, but most larger embeddings resulted in a  $P_{success}$  of zero. Therefore, despite the increasing cost of re-embedding with QUBO size, the largest QUBOs are the ones that most benefit from re-embedding, as it may be required to solve problem instances.

Next, in Fig. 7, we show a comparison of post-processing methods on PYQ QUBOs only. Here, we see that ME consistently recovers a higher number of valid schedules than MV does. As shown where  $T = 9$ , ME can even recover valid schedules when MV often produces none. Table ?? shows a broader comparison of post-processing methods for all embeddings used in this work. Excluding the cases where neither post-processing method resulted in a positive  $P_{success}$ , which often corresponds to the largest embeddings, ME resulted in a higher  $P_{success}$  77% of the time. For the majority of the remainder of the cases with a positive success rate, ME and MV returned the same number of valid schedules. The percentage of cases in which MV outperformed ME is less than 0.01%. Therefore, to maximize  $P_{success}$  on the JSP, ME should be used during post-processing.

Finally, we evaluate the impacts of embedding size on an altered anneal schedule that includes a pause. Figure 8 shows our results from a set of ten embeddings of a PYQ QUBO where

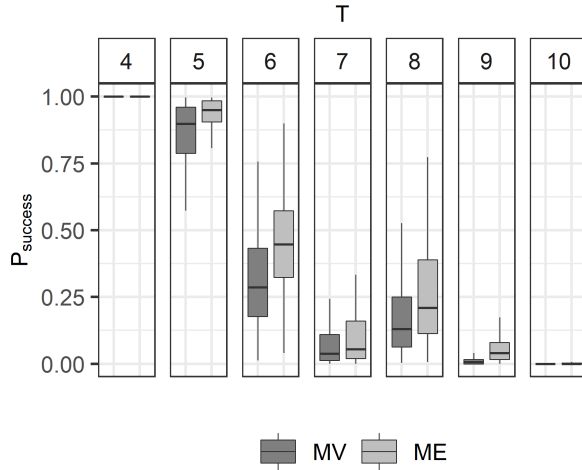


Figure 7. Comparison post-processing methods

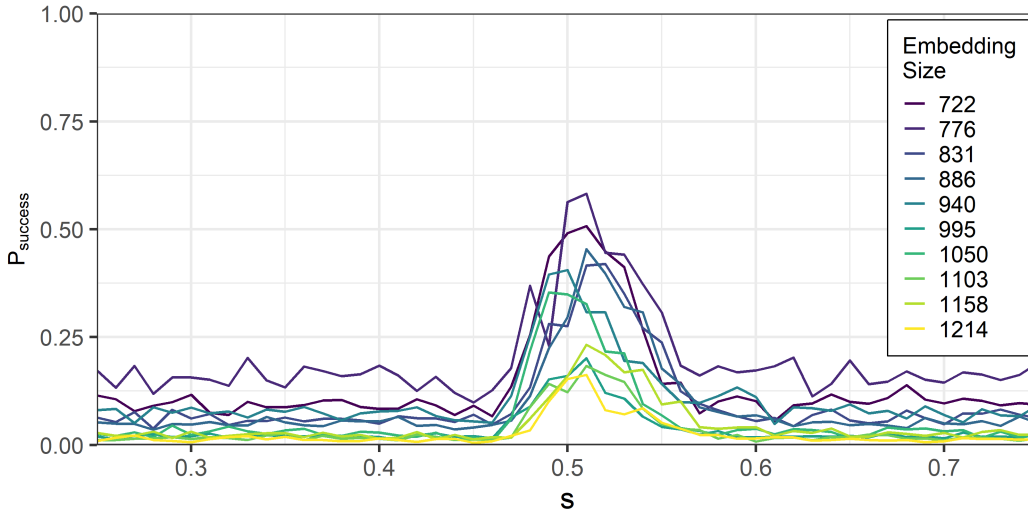


Figure 8. Effect of embedding size on pausing

ME is used during post-processing and is representative of results obtained for embeddings of other instances. These results are consistent with those in [17, 25]; only within a narrow region of  $s$  does pausing result in a higher  $P_{success}$ , and outside this region,  $P_{success}$  remains relatively constant. In this figure, we see that  $P_{success}$  decreases as embedding size increases, which was also observed in Fig. 4a. However, this figure also shows that the performance improvement from pausing near the optimal pause location also decreases as embedding size increases. We attribute this effect to the increased cost associated with flipping larger chains. That is, as the number of qubits within a chain increases, changing the value of all chained qubits comes at a higher energy cost. Thus, for embeddings with longer chains, thermal relaxation will not be as effective as it is for shorter chains. Lastly, we see that while embedding size affects  $P_{success}$ , it does not significantly affect the optimal pause location. In summary, we see that in order to maximize performance when pausing near the optimal pause location, embedding size should be minimized.

### 3.3. Discussion

As previously mentioned in the introduction to this paper, two major challenges need to be addressed to improve QA performance. Our evaluation shows how the four methods in Section 2 can address these challenges and further improve QA performance. We address the first challenge through an efficient problem formulation — which we represent with the PYQ tool — and through variable pruning. Our results show the impacts of this through smaller QUBOs, smaller embedding sizes, and reduced embedding failure rates. While the second challenge is also addressed through our variable reduction techniques, we primarily use re-embedding to find smaller embeddings that result in higher performance. The necessity of this step is shown in Fig. 4b, where the embeddings that are most likely to be found by the embedding algorithm are unable to solve the JSP instance. Despite performing both variable reduction and re-embedding,  $P_{success}$  can still be low for larger QUBOs. We show that this can be remedied through proper selection of a post-processing method and insertion of a pause near the optimal pause location in an anneal schedule. However, to make the most of pausing, our results show that re-embedding needs to be performed to find smaller embeddings with relatively short chains, since the performance benefits from pausing decrease as chain length increases. Lastly, our results show that while embedding size affects the performance improvement from pausing, it does not significantly change the optimal pause location. In summary, to achieve the highest performance with QA, all the considered methods must be performed together.

## 4. Related Work

One of the first steps in QA is problem formulation. While this is a problem dependent step, Lucas [23] provides Ising formulations of many NP-hard problems and shows that some share fundamental components. One of the major considerations in this step is the number of variables used, as these are represented by qubits on the QPU, which are highly limited in number. The desire to minimize the number of variables can be clearly seen in the work presented by Venturelli et al. [36] through the use of variable pruning methods. However, even when using this method, only relatively small problems are able to be solved. Stollenwerk et al. [32] show effects of tunable resolution in discretization of a variable. Fine resolution results in high quality solutions, but comes at the cost of an increased number of variables. Conversely, a lower resolution can reduce the number of variables required, but results in lower quality solutions.

In order to enable solving large scale problems with current QA hardware, various hybrid quantum-classical methods have been proposed. These range from methods that partition and modify problems over many anneals to strategies for decomposing problems into subproblems to be solved by classical and quantum computers. Karimi and Rosenberg [19] present a method to iteratively set the values of variables that nearly always take the same value across many annealing runs. Through the use of this method, the authors note that not only does problem size decrease, but success rate increases, and outperforms QA when the method is not used. Another iterative method that targets problems too large to be fully embedded on the QPU is given by Rosenberg et al. [30]. In this method, a large problem is solved by iteratively setting variables, and solving the remaining subproblem. In contrast to these iterative methods are hybrid methods that employ both classical and quantum computers to solve subproblems of a decomposed problem. Such a method is presented by Tran et al. in [35], in which a battery capacity constraint for a Mars lander is offloaded to classical computers. A similar work applies this method to multiple scheduling problems, and provides the decomposition for each [34]. Yet

another hybrid quantum-classical method is to use classical computation resources as search managers for problems. In this method, a binary search tree is constructed for a QUBO, and is then used to direct the search based on results from the annealer by setting the values of certain variables. Not only does setting variable values make use of previous results, but it also provides means to reduce problem size. This method can be found in many works [34–36]. Similar to the methods in our work, hybrid methods can successfully reduce problem and embedding size, and improve performance. Additionally, hybrid quantum-classical methods can also enable guided searches of the solution space. Therefore, development of hybrid quantum-classical methods is a promising area of research for QA.

Another step required by QA that also has an effect on the size of the problem submitted to the QPU, is embedding. While embedding does not influence the size of the original problem, it does determine the size of the embedded problem, which is what will be solved on the QPU. Thus, embedding algorithms that result in small embeddings are of great interest. Currently, the embedding tool developed by D-Wave, *minorminer*, is commonly used [7]. This method employs a heuristic, resulting in embeddings of different sizes being found for the same QUBO. However, embedding size is not the only metric by which embedding algorithms are judged; two other metrics of interest are the time required to generate an embedding, and the resulting performance of annealing with a specific embedding. Therefore, embedding algorithms that can improve any of these three metrics are useful.

Pudenz et al. [27] show that as chain length increases, performance decreases. This implies that two of the metrics for embedding, embedding size and performance, may be related. However, using chain length alone to measure embedding size may be misleading, as shortening one chain may result in lengthening other chains. Abbott et al. [1] show that embedding time can eliminate any computational speedup resulting from using QA. While they do not present a new embedding algorithm, they note that for their type of problem, the dynamically weighted maximum-weight independent set problem, the embedding cost for any number of weight configurations of a graph can be reduced to the embedding cost for one configuration by reusing the embedding with altered weights. One embedding algorithm that is able to improve all three metrics, by considering the structure of the QPU, is presented by Date et al. [11]. However, their algorithm assumes the working graph of the QPU has a 100% yield, that is, no qubits on the QPU are turned off. Furthermore, the algorithm only applies to Chimera architecture QPUs, and will not apply to Pegasus architecture QPUs. In [37], Yarkoni et al. note that a parameter related to embedding, chain strength, has a strong effect on performance. These works show that the embedding process can have a significant impact on the performance of QA and affect the computational speedup obtained from using QA. These results also show that there is room for improvement in current embedding methods.

Another way of improving performance of QA is through the use of anneal controls. However, as they are only available on the latest Chimera architecture QPU, research implementing them is relatively limited. Lanting et al. [22] present one of the first works using anneal controls to improve performance by applying non-uniform driver Hamiltonians, through the use of anneal offsets, to mitigate perturbative anticrossings. Marshall et al. [25] explore the use of anneal pausing and reverse annealing, and are able to improve performance by orders of magnitude. Izquierdo et al. [17] extend this work to include embedded problems, and study the effect of chain strength. In [38], Ikeda et al. perform forward annealing, and use the results as input to reverse annealing. They show that reverse annealing can improve performance, and that selecting the lowest energy configurations from forward annealing results in higher performance than randomly selecting a configuration. However, they only explore limited schedule configurations

for reversed annealing. In these works, it is clear that anneal controls have a large configuration space, and an exhaustive search would be costly in terms of QPU access time. Yarkoni et al. [38] address this by proposing and successfully implementing an evolutionary algorithm to tune anneal offsets on a per-qubit basis. Overall, these works show that annealing controls enable performance improvements, yet finding the optimal configuration is challenging.

## Conclusions and Future Work

In this work, we evaluated QA on the JSP. We first converted each JSP instance to use binary variables and determined a set of excess variables that could be pruned. We then used two tools to create QUBOs of different sizes that each produced valid schedules for the original JSP instance. Next, for each QUBO, we repeated the embedding process to find embeddings over a large range of embedding sizes. We retrieved samples from the QPU using each embedding, and applied two post-processing techniques to recover incomplete configurations. Finally, we performed annealing a second time, using a modified annealing schedule containing a pause.

Our results show that for the JSP, QA performance is sensitive to problem formulation, embedding, and post-processing. Furthermore, when studying the effects of anneal controls, such as anneal schedule modifications, special care must be taken when preparing a problem for annealing so as not to mask the effect of the control on performance. We have shown that embedding size can have a significant impact on performance, and that the embeddings that are most commonly found by embedding tools do not result in the highest performance. We then proposed re-embedding as a method to improve  $P_{success}$ . While re-embedding introduces a time-performance trade-off, we showed that it can be necessary for solving larger instances. We have also shown that the size of a QUBO affects the sizes of embeddings found, and consequently, QUBO size also impacts performance. Thus, minimization of QUBO size through an efficient formulation and variable pruning techniques are critical to high performance. We then showed that a higher number of valid schedules can be found through use of a post-processing method that considers the penalties associated with different qubit values. Finally, we showed that pausing can improve  $P_{success}$ , yet the magnitude of improvement lowers as embedding size increases, due to the increased energy cost associated with changing the state of an entire qubit chain. In summary, we have shown that due to the sequential nature of and the interplay between the steps used to solve the JSP, they must be considered together in order to achieve high performance. Furthermore, as these steps are not specific to the JSP, they can be applied to other embedded problems to improve performance.

Some main limitations of the current QA hardware are the low number of qubits and low connectivity between qubits. While Pegasus architecture QPUs [5] will increase both the number of qubits and qubit connectivity, embedded problems may still be challenging, as they often require numerous variables due to one-hot encoding of integer variables. We expect that quantum-classical hybrid methods, such as those of the related work, will become increasingly important, as they can decompose problems so that fewer resources are required for solving. Additionally, as many problems require variable chaining, embedding algorithms that find smaller, higher performing embeddings, in less time than current methods, will also be of interest for QA. Lastly, since anneal schedule modifications can significantly improve QA performance, yet many real-world applications will require embedding, which as we have shown limits the effectiveness of anneal schedule modifications, additional work is required to determine which modifications result in the highest performance.

## Acknowledgements

The authors thank Professor Samy Baladram for his comments on the manuscript. We also thank Professor Masayuki Ohzeki and Professor Masamichi Miyama for their useful discussions.

This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program “R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications”, Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Abbott, A.A., Calude, C.S., Dinneen, M.J., et al.: A hybrid quantum-classical paradigm to mitigate embedding costs in quantum annealing. *International Journal of Quantum Information* 17(05), 1950042 (2019), DOI: 10.1142/S0219749919500424
2. Albash, T., Lidar, D.A.: Demonstration of a scaling advantage for a quantum annealer over simulated annealing. *Phys. Rev. X* 8(3), 031016 (2018), DOI: 10.1103/PhysRevX.8.031016
3. Amin, M.H.: Searching for quantum speedup in quasistatic quantum annealers. *Phys. Rev. A* 92(5), 052323 (2015), DOI: 10.1103/PhysRevA.92.052323
4. Applegate, D.L., Cook, W.J.: A computational study of the job-shop scheduling problem. *INFORMS J. Comput.* 3(2), 149–156 (1991), DOI: 10.1287/ijoc.3.2.149
5. Boothby, K., Bunyk, P., Raymond, J., et al.: Next-generation topology of D-Wave quantum processors. [https://www.dwavesys.com/sites/default/files/14-1026A-C\\_Next-Generation-Topology-of-DW-Quantum-Processors.pdf](https://www.dwavesys.com/sites/default/files/14-1026A-C_Next-Generation-Topology-of-DW-Quantum-Processors.pdf) (2019), accessed: 2020-11-18
6. Bowman, E.H.: The schedule-sequencing problem. *Operations Research* 7(5), 621–624 (1959), DOI: 10.1287/opre.7.5.621
7. Cai, J., Macready, W.G., Roy, A.: A practical heuristic for finding graph minors. <https://arxiv.org/abs/1406.2741> (2014), accessed: 2020-11-18
8. D-Wave Systems: D-Wave Binary CSP. <https://github.com/dwavesystems/dwavebinarycsp>, accessed: 2020-07-10
9. D-Wave Systems: dimod. <https://github.com/dwavesystems/dimod>, accessed: 2020-07-10
10. D-Wave Systems: dwave-system. <https://github.com/dwavesystems/dwave-system>, accessed: 2020-07-10
11. Date, P., Patton, R.M., Schuman, C.D., et al.: Efficiently embedding QUBO problems on adiabatic quantum computers. *Quantum Inf. Process.* 18(4), 117 (2019), DOI: 10.1007/s11128-019-2236-3

12. Denchev, V.S., Boixo, S., Isakov, S.V., et al.: What is the computational value of finite-range tunneling? *Phys. Rev. X* 6(3), 031015 (2016), DOI: 10.1103/PhysRevX.6.031015
13. Farhi, E., Goldstone, J., Gutmann, S., et al.: Quantum computation by adiabatic evolution. <https://arxiv.org/abs/quant-ph/0001106> (2000), accessed: 2020-11-18
14. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Miller, G.L. (ed.) *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, 22-24 May 1996, Philadelphia, Pennsylvania, USA. pp. 212–219. ACM (1996), DOI: 10.1145/237814.237866
15. Hen, I., Job, J., Albash, T., et al.: Probing for quantum speedup in spin-glass problems with planted solutions. *Phys. Rev. A* 92(4), 042325 (2015), DOI: 10.1103/PhysRevA.92.042325
16. Ikeda, K., Nakamura, Y., Humble, T.S.: Application of quantum annealing to nurse scheduling problem. *Scientific Reports* 9(1), 12837 (2019), DOI: 10.1038/s41598-019-49172-3
17. Izquierdo, Z.G., Grabbe, S., Hadfield, S., et al.: Ferromagnetically shifting the power of pausing. <https://arxiv.org/abs/2006.08526> (2020), accessed: 2020-11-18
18. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. *Phys. Rev. E* 58(5), 5355–5363 (1998), DOI: 10.1103/PhysRevE.58.5355
19. Karimi, H., Rosenberg, G.: Boosting quantum annealer performance via sample persistence. *Quantum Inf. Process.* 16(7), 166 (2017), DOI: 10.1007/s11128-017-1615-x
20. Katzgraber, H.G., Hamze, F., Zhu, Z., et al.: Seeking quantum speedup through spin glasses: The good, the bad, and the ugly. *Phys. Rev. X* 5(3), 031026 (2015), DOI: 10.1103/PhysRevX.5.031026
21. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220(4598), 671–680 (1983), DOI: 10.1126/science.220.4598.671
22. Lanting, T., King, A.D., Evert, B., et al.: Experimental demonstration of perturbative anticrossing mitigation using nonuniform driver hamiltonians. *Phys. Rev. A* 96(4), 042322 (2017), DOI: 10.1103/PhysRevA.96.042322
23. Lucas, A.: Ising formulations of many NP problems. *Frontiers in Physics* 2, 5 (2014), DOI: 10.3389/fphy.2014.00005
24. Mandrà, S., Zhu, Z., Wang, W., et al.: Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics versus quantum approaches. *Phys. Rev. A* 94(2), 022337 (2016), DOI: 10.1103/PhysRevA.94.022337
25. Marshall, J., Venturelli, D., Hen, I., et al.: Power of pausing: Advancing understanding of thermalization in experimental quantum annealers. *Phys. Rev. Applied* 11(4), 044083 (2019), DOI: 10.1103/PhysRevApplied.11.044083
26. Okada, S., Ohzeki, M., Taguchi, S.: Efficient partition of integer optimization problems with one-hot encoding. *Scientific Reports* 9(1), 13036 (2019), DOI: 10.1038/s41598-019-49539-6
27. Pudenz, K.L., Albash, T., Lidar, D.A.: Error-corrected quantum annealing with hundreds of qubits. *Nature Communications* 5(1), 3243 (2014), DOI: 10.1038/ncomms4243

28. Recruit Communications: PyQUBO. <https://github.com/recruit-communications/pyqubo>, accessed: 2020-07-10
29. Rieffel, E.G., Venturelli, D., OGorman, B., et al.: A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing* 14(1), 1–36 (2015), DOI: 10.1007/s11128-014-0892-x
30. Rosenberg, G., Vazifeh, M., Woods, B., et al.: Building an iterative heuristic solver for a quantum annealer. *Comput. Optim. Appl.* 65(3), 845–869 (2016), DOI: 10.1007/s10589-016-9844-y
31. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* 41(2), 303–332 (1999), DOI: 10.1137/S0036144598347011
32. Stollenwerk, T., OGorman, B., Venturelli, D., et al.: Quantum annealing applied to de-conflicting optimal trajectories for air traffic management. *IEEE Transactions on Intelligent Transportation Systems* 21(1), 285–297 (2020), DOI: 10.1109/TITS.2019.2891235
33. Tanahashi, K., Takayanagi, S., Motohashi, T., et al.: Application of Ising machines and a software development for Ising machines. *Journal of the Physical Society of Japan* 88(6), 061010 (2019), DOI: 10.7566/JPSJ.88.061010
34. Tran, T.T., Do, M., Rieffel, E.G., et al.: A hybrid quantum-classical approach to solving scheduling problems. In: *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, 6-8 July 2016, Tarrytown, NY, USA.* pp. 98–106. AAAI Press (2016), <http://aaai.org/ocs/index.php/SOCS/SOCS16/paper/view/13958>
35. Tran, T.T., Wang, Z., Do, M., et al.: Explorations of quantum-classical approaches to scheduling a Mars lander activity problem. In: *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, 13 February 2016, Phoenix, Arizona, USA.* AAAI Workshops, vol. WS-16-12. AAAI Press (2016), <http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12664>
36. Venturelli, D., Marchand, D.J.J., Rojo, G.: Quantum annealing implementation of job-shop scheduling. In: *Proceedings of the Eleventh Workshop on Constraint Satisfaction Techniques for Planning and Scheduling, COPLAS 2016, 13-14 June 2016, London, UK.* pp. 25–34 (2016)
37. Yarkoni, S., Plaat, A., Bäck, T.: First results solving arbitrarily structured maximum independent set problems using quantum annealing. In: *2018 IEEE Congress on Evolutionary Computation, CEC 2018, 8-13 July 2018, Rio de Janeiro, Brazil.* pp. 1–6. IEEE (2018), DOI: 10.1109/CEC.2018.8477865
38. Yarkoni, S., Wang, H., Plaat, A., et al.: Boosting quantum annealing performance using evolution strategies for annealing offsets tuning. In: *Feld, S., Linnhoff-Popien, C. (eds.) Quantum Technology and Optimization Problems - First International Workshop, QTOP@NetSys 2019, 18 March 2019, Munich, Germany, Proceedings.* *Lecture Notes in Computer Science*, vol. 11413, pp. 157–168. Springer (2017), DOI: 10.1007/978-3-030-14082-3\_14



# Developing an Architecture-independent Graph Framework for Modern Vector Processors and NVIDIA GPUs

Ilya V. Afanasyev<sup>1</sup>

© The Author 2020. This paper is published with open access at SuperFri.org

This paper describes the first-in-the-world attempt to develop an architectural-independent graph framework named VGL, designed for different modern architectures with high-bandwidth memory. Currently VGL supports two classes of architectures: NEC SX-Aurora TSUBASA vector processors and NVIDIA GPUs. However, VGL can be easily extended to other architectures due to its flexible software structure. VGL is designed to provide users with the possibility of selecting the most suitable architecture for solving a specific graph problem on a given input data, which, in return, allows to significantly outperform existing frameworks and libraries, developed for modern multicore CPUs and NVIDIA GPUs. Since VGL uses an identical set of computational and data abstractions for all architectures, its users can easily port graph algorithms between different target architectures without any source code modifications. Additionally, in this paper we show how graph algorithms should be implemented and optimised for NVIDIA GPU and NEC SX-Aurora TSUBASA architectures, demonstrating that both architectures have multiple similar properties and hardware features.

*Keywords: vector computers, NVIDIA GPUs, graph algorithms, graph framework, VGL, CUDA, optimisation.*

## Introduction

Developing efficient implementations of graph algorithms is an extremely important problem of modern computer science, since graphs are heavily used in many applications fields: social networks and web graphs analysis, navigation, solving infrastructural problems, and many others. Supercomputing architectures with high-bandwidth memory (HBM) are able to significantly speed up solving various graph problems, which belong to the data-intensive class and thus potentially benefit from faster memory hierarchy. Nowadays, high-bandwidth memory is installed either into GPUs or systems with vector processing features (vector processors or CPUs with vector extensions). Efficiently implementing graph algorithms on systems with high-bandwidth memory is difficult, since implementation approaches significantly differ from those used for traditional multicore CPUs, mainly because these systems utilize SIMD-processing (Single Instruction Multiple Data) features.

Various graph libraries and frameworks have been developed for various modern architectures, mainly multicore CPUs and NVIDIA GPUs. Graph libraries usually provide highly-optimised implementations of several fundamental graph algorithms, while graph frameworks typically include optimised computational and data abstractions, which can be used to easily express different graph algorithms variations. However, existing graph-libraries and frameworks have the following drawbacks:

1. none of the existing frameworks and libraries support efficient graph processing on vector systems (such as NEC SX-Aurora TSUBASA);
2. existing frameworks typically target only a specific architecture, forcing its users to completely rework the implementation when using a different architecture is required; and
3. existing frameworks in many cases can be further optimized for their target architectures (including NVIDIA GPUs).

---

<sup>1</sup>Moscow Center of Fundamental and Applied Mathematics, Moscow, Russia

To approach the first problem we have previously developed a VGL (Vector Graph Library)<sup>2</sup> [2, 5] framework for the NEC SX-Aurora TSUBASA vector architecture. VGL significantly outperforms many existing graph-processing frameworks, developed for modern multicore CPUs and NVIDIA GPUs.

As shown in [3], NVIDIA GPUs and NEC SX Aurora TSUBASA vector architecture have many common hardware features. This means that **many graph algorithms can be implemented on these architectures with similar optimisation and implementation approaches**. However, at the moment of this writing no research has been carried out to confirm (or refute) this thesis. In order to approach this problem, we ported our VGL framework (originally developed for NEC SX-Aurora TSUBASA vector architecture) to the latest NVIDIA GPUs, what allowed us to compare implementation and optimisation approaches, which should be used for both architectures.

As a result we have developed a first-in-the-world architecture independent framework, which simultaneously targets multiple architectures with high-bandwidth memory: modern NEC SX-Aurora TSUBASA vector system and NVIDIA GPUs. This is achieved by using a unified set of computational and data abstractions, identical for all architectures. Moreover, our framework has flexible software structures, which allow to easily extend it to different architectures (for example multicore CPUs). This property of VGL allows its users to select the most suitable architecture for solving a specific graph problem on a given input data, thus solving it considerably faster compared to the existing frameworks and libraries, developed for a single particular architecture.

The article is organized as follows. Section 1 describes primary target architectures of the VGL framework: NEC SX-Aurora TSUBASA vector processors and modern NVIDIA GPUs. Section 2 describes existing state-of-the-art frameworks, developed for modern NVIDIA GPUs and multicore CPUs. In addition, Section 2 provides the description of computational and data abstractions, used in the VGL framework. Section 3 describes program structure of the VGL framework, which allows it to operate on different architectures, and thus to easily port graph algorithms implementations between them. Section 4 provides a detailed comparison of implementation and optimisation approaches, which are used to implement VGL computational abstractions on NEC SX-Aurora TSUBASA and NVIDIA GPU architectures. In particular, effects of different optimisations is compared for these two architectures. Section 5 evaluates the performance of multiple graph algorithms implemented via VGL framework, as well as VGL performance against existing graph-processing frameworks for other architectures. Conclusion summarizes the study and points directions for further work.

## 1. Target Architectures Overview

### 1.1. NEC SX-Aurora TSUBASA

The NEC SX-Aurora TSUBASA vector architecture [15, 20] consists of multiple vector engines (VE), installed into a vector host (VH), which is a typical x86 node. Vector engines are used as a primary processors for executing vectorised applications, while vector host is used as a secondary processor for executing basic operating system (OS) functions, as well as some scalar computations offloaded from the VE. The VE has eight powerful vector cores, each one operating with vector instructions of 256 length. Each vector core consists of two computational

---

<sup>2</sup>VGL is available for free download at [vgl.parallel.ru](http://vgl.parallel.ru)

components: a scalar processing unit (SPU) and a vector processing unit (VPU). All vector computations are performed by VPUs, while SPUs are designed to provide a relatively high performance on scalar computations, without the need to explicitly offload them to the vector host (thus significantly reducing the amount of transfers through interconnect). In the following subsections the most important hardware characteristics of the NEC SX-Aurora TSUBASA architecture will be provided, related to graph processing.

## 1.2. NVIDIA GPU

NVIDIA GPUs [14] are also installed into the system as coprocessors, similar to SX-Aurora vector engines. Modern NVIDIA GPUs have thousands of CUDA cores, which are grouped into streaming multiprocessors (SM). Streaming multiprocessors execute instructions based on the warp concept: a group of 32 threads running on CUDA cores perform exactly the same instruction at every given moment of time. This computing model has many common features with vector computing, since they both belong to SIMD [9] class. This particular feature determines the fact that various graph algorithms may potentially be implemented similarly on these two classes of architectures. Further in the paper warps and vector instructions will be sometimes referred as “SIMD instructions”. In this paper two most recent GPUs of Tesla family are used: V100 and A100, hardware characteristics of which will be also provided in the following subsection and simultaneously compared with vector engines.

## 1.3. Hardware Characteristics Comparison

The main hardware characteristics of the two latest generations of SX-Aurora Vector Engines and NVIDIA GPUs are listed in Tab. 1. These hardware characteristics most noticeably affect the performance of graph processing. For example, peak memory bandwidth determines how fast information about graph edges can be loaded from memory, while the memory capacity determines graph of which size can be processed using GPU or VE, etc.

**Table 1.** The comparison between main hardware characteristics of modern NVIDIA GPUs and NEC SX-Aurora TSUBASA vector engines

Hardware Characteristic	NEC SX-Aurora TSUBASA (1st generation)	NEC SX-Aurora TSUBASA (2nd generation)	NVIDIA V100 GPU	NVIDIA A100 GPU
Peak memory bandwidth	1200 GB/s	1500 GB/s	900 GB/s	1500 GB/s
Memory capacity	48 GB	48 GB	16-32 GB	40-80 GB
LLC size	16 MB	16 MB	6 MB	40 MB
Prefetching support	yes	yes	no	yes
LLC bandwidth	3000 GB/s	3000 GB/s	N/a	N/a
SIMD size	256	256	32	32
Cores number	8	8	5120	6912
Interconnect bandwidth	up to 32 GB/s	up to 32 GB/s	up to 300 GB/s	up to 600 GB/s

Based on the provided in Tab. 1 information, the following conclusions can be made. The first generation of SX-Aurora vector engines have comparable characteristics to V100 GPUs, while the second generation – to A100 GPUs. However, several differences exist: for example, GPUs typically have interconnect with higher bandwidth, which allows to copy input graphs into GPU memory much faster. At the same time vector engines have significantly less resource of inner parallelism: they use only 8 cores, each one operating with vectors of 256 length, while modern GPU have thousands of cores, which require even more threads running in order to efficiently hide memory latency. This potentially allows vector engines to process small-sized and medium-sized graphs more efficiently compared to GPUs.

## 2. State of the Art

### 2.1. Existing Graph-Processing Frameworks

Several graph libraries and frameworks have been recently developed for modern multicore CPUs and NVIDIA GPUs. Ligra [18], Galois [17] and GAPBS [6] are the most well-known examples of multicore CPUs frameworks and libraries, while Gunrock [19] CuSHA [13], Medusa [22], and Enterprise [16] frameworks and libraries target modern NVIDIA GPUs. However, the following factors determine the relevance of developing VGL framework:

- none of the existing frameworks target modern vector systems, such as NEC SX-Aurora TSUBASA;
- none of the existing frameworks are capable of operating with relatively high performance on different architectures, such as NVIDIA GPUs, multicore CPUs and vector processors; and
- performance of almost all existing frameworks and libraries for NVIDIA GPU architectures can be further improved by applying additional optimisations, discussed in this paper.

### 2.2. VGL Abstractions

In [2, 5] we have proposed a set of four computational abstractions and four data abstractions, which can be efficiently implemented on the NEC SX-Aurora TSUBASA architecture. Further in this paper we will describe how these abstractions can be ported to the NVIDIA GPU architecture. However, first it is necessary to describe the main functionality of each abstraction and discuss why these abstractions are suitable for both classes of architectures.

**Graph.** A graph is the main data-abstraction of the VGL framework. Graphs in the VGL are stored in optimized and preprocessed VectCSR format [4]. The VGL framework provides a convenient interface for working with both directed and undirected graphs. For directed graphs, outgoing and incoming edges are stored for each vertex, while for undirected graphs all edges are stored as outgoing. This allows VGL users to easily implement pull-based and push-based algorithms [7].

**Frontier.** Frontier is a specific subset of graph vertices and the second important data-abstraction of the VGL. Frontiers in the VGL allows to control which vertices and edges need to be processed inside computational abstractions. For example, the advance abstraction processes all vertices from the input frontier, as well as all their adjacent edges. Frontiers in the VGL have different types: sparse, dense, and all-active (last one includes all graph vertices). Sparse

frontiers are represented via lists of indices, while dense – via an array of flags, where each flag corresponds to the presence of vertex inside the frontier.

**Vertices array.** Vertices arrays allow storing information about graph vertices, for example current level of each vertex in the BFS (Breadth-First Search) algorithm, or distances to each vertex in the shortest paths algorithms. Vertices arrays have a straightforward implementation using aligned arrays, allocated either in vector engine or unified memory of GPUs.

**Edges array.** Edges arrays allow storing information about graph edges, which is required when working with weighted graphs. Weighted edges are stored as a structure of arrays, providing better memory access pattern for vector instructions and warps.

**Advance.** The advance abstraction is the main tool of traversing graphs in the VGL. The advance input consists of a graph, an input frontier, and several user-defined handler functions: *vertex preprocess op*, *edge op*, *vertex postprocess op*. The advance applies *vertex preprocess op* to each vertex of its input frontier, *edge op* to each of its adjacent edges, and then *vertex postprocess op* to each vertex again. It is guaranteed that the execution of *vertex preprocess op*, edge-processing, and *vertex postprocess op* operations for each vertex are serialized. However, all *edge op* operations for each adjacent edge are executed in parallel. The computational workflow of the advance abstraction (as well as three others) is illustrated in Fig. 1. The advance abstraction is used in all situations, when processing graph edges is required.

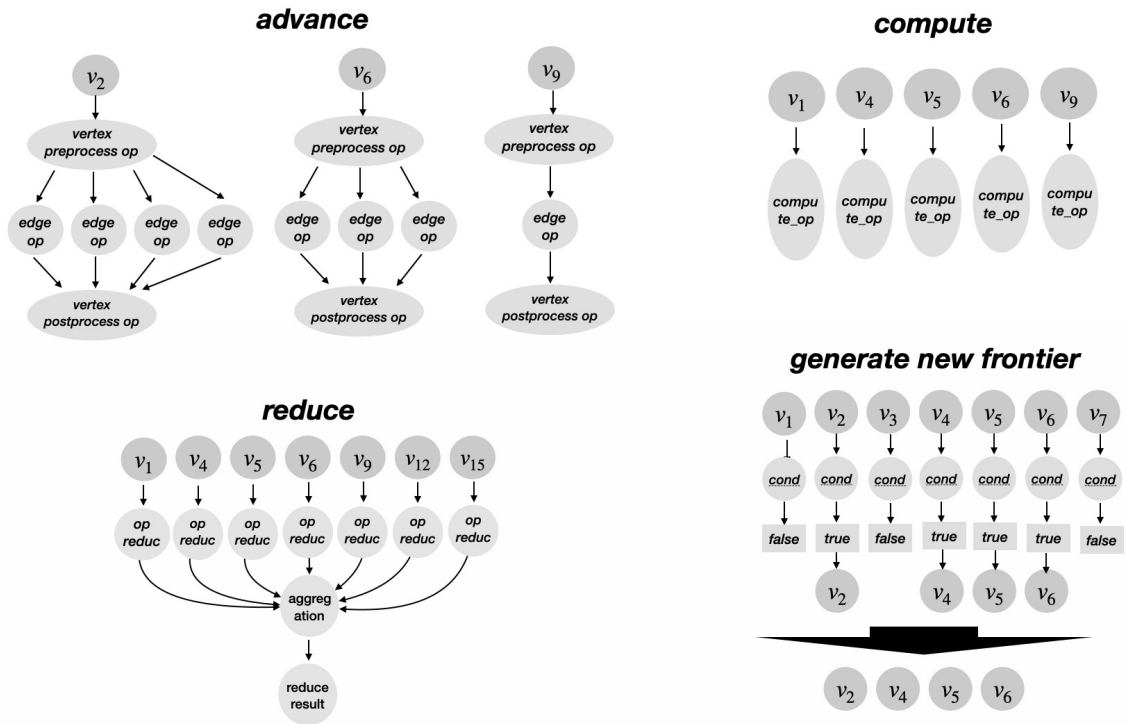


Figure 1. The computational workflow of VGL abstractions

**Compute.** The compute abstraction applies a user-defined operation to each vertex of the given input frontier. Typically, this abstraction is used for a wide range of operations over graph vertices: initializing distances in shortest paths, implementing the “hook” phase in connected component algorithms, and many others.

**Reduce.** The reduce abstraction applies a user-defined operation (which returns some value) to each vertex of a given input frontier. The returned values are reduced using additionally specified reduction operation (SUM, MAX, MIN, AVG). This abstraction can be used for a

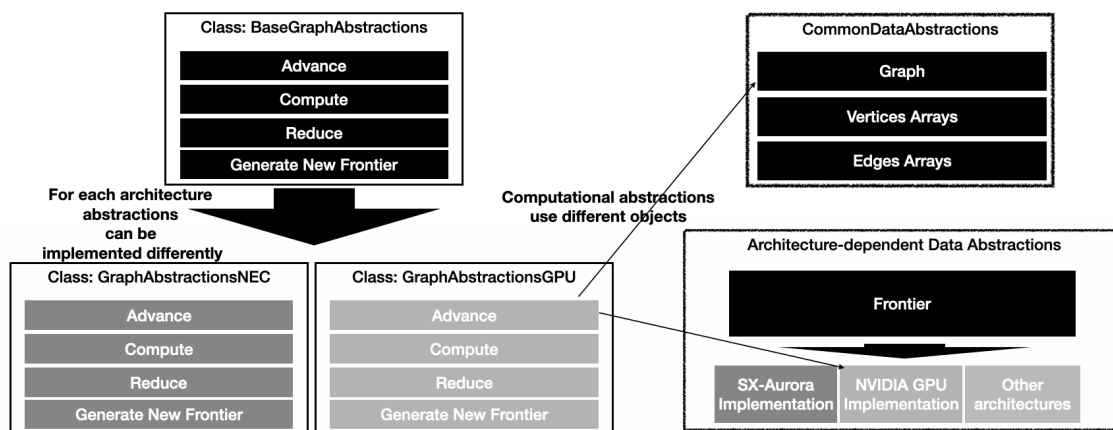
large number of applications: estimating future frontier size in BFS, calculating dangling nodes inputs in page rank, etc.

**Generate New Frontier.** The generate new frontier abstraction allows to create a new frontier of graph vertices, using a specified condition. This condition can be based on vertex id, its degree, or some data from user-defined vertices arrays.

The described computational abstractions are suitable for both NVIDIA GPUs and NEC SX-Aurora TSUBASA architectures for the following two reasons. First, each abstraction has a large resource of so-called data-driven parallelism, since they all execute the same operations over different data (graph vertices and edges). This allows to efficiently use vector instructions and warps, which is crucial for both architectures. In addition, each abstraction has a large resource of inner parallelism, since all graph vertices and edges can be processed in parallel, what is important since both architectures can be classified as massively-parallel. Thus, the described abstractions can be implemented on both architectures with approximately the same level of efficiency, in the case when correct optimisations are applied. These optimisations and implementation approaches will be described in details further in the paper.

### 3. Program Structure of VGL Framework

The software structure of the developed framework is illustrated in Fig. 2. This software structure allows VGL to operate on various target platforms, since all the abstractions have identical interfaces for all platforms. Each computational abstraction is implemented as a method of a base class, and has a basic OpenMP parallel implementation. For each architecture, a separate derived class can be created, where these methods are overloaded to contain architecture-specific implementations. Implementation and optimisation approaches inside overloaded methods are not limited to any extend; for example, abstractions for NVIDIA GPU can be implemented via CUDA, while for NEC SX-Aurora TSUBASA – by using special vector instructions and directives.



**Figure 2.** The software structure of the VGL framework. Third-party users are allowed to extend abstractions for different architecture by implementing derived classes

Data abstractions are split into two categories: architecture-dependent and architecture-independent. Graph, vertices array and edges array belong to architecture-independent category, since they have exactly the same implementation for each architecture currently supported in VGL. The frontier belongs to the architecture dependent category, since our experi-

ments demonstrated that frontiers require significantly different implementation approaches for NVIDIA GPU, SX-Aurora TSUBASA and multicore CPUs architectures.

The described software structure allows third-party users to extend VGL on the new architectures by implementing derived classes and changing several implementations of abstractions if necessary. Since interfaces of all abstractions remain exactly the same, graph algorithms implementations will also remain the same for different architectures, which makes VGL an architectural-independent framework.

## 4. Porting VGL Abstractions to NVIDIA GPU: a Detailed Comparison of Implementation and Optimisation Approaches

Despite the fact that VGL computational and data abstractions for different architectures can be implemented independently, for NVIDIA GPUs and NEC SX-Aurora TSUBASA architectures similar implementation and optimisation approaches can be used in many situations. Further in this section we will discuss which optimisations have been used for both architectures when implementing different computational abstraction. In the most important cases we will also demonstrate which acceleration has been achieved on each platform by applying each optimisation.

**Advance.** Implementing the advance abstraction on NVIDIA GPUs and vector architectures is difficult since it has highly-irregular computational workflow caused by (1) the irregular distribution of vertex degrees and (2) a large number of indirect memory accesses performed during edge traversals. Moreover, the advance abstraction contributes from 50% to 95% of execution time for many graph algorithms. The main optimisations used in the advance abstraction are listed in Tab. 2 together with the acceleration values obtained by implementing each of the optimisations.

Inter-core workload balancing in graph algorithms is typically implemented via splitting graph vertices into groups based on their degrees. Vertices from different groups are processed using different amount of hardware resources (cores). Dividing vertices into groups can be implemented either based on graph preprocessing (preliminary sorting graph vertices based on their degree), or dynamically during graph algorithm execution (using different vertex queues). According to our experiments both these approaches allow to achieve a comparable acceleration on NVIDIA GPUs, while for the NEC SX-Aurora TSUBASA architecture only the first approach (preprocessing) can be efficiently used. In order to provide better hardware utilisation, different groups of vertices can be simultaneously processed on the same GPU or vector engine. This optimisation can be implemented using CUDA-streams for NVIDIA GPUs, while OpenMP nested parallelism can be used for the NEC SX-Aurora TSUBASA architecture. This optimisation allows to achieve higher acceleration on NVIDIA GPUs, since modern GPUs have significantly larger resource of parallelism, and thus require more graph vertices and edges to be processed in parallel.

Low-degree vertices should be accurately processed on both architectures, since it is hard to efficiently use vector instructions or warps when loading information about their adjacent edges. On both architectures loading information about graph edges with load/store instructions must have sequential memory access pattern in order to maximise the sustained bandwidth. To achieve this goal we used two different techniques: constructing graph vector extension [4], or using

**Table 2.** The effect of different optimisations applied to the implementation of the advance abstraction

Optimisation	V100 GPU, acceleration (times)	NEC SX-Aurora, TSUBASA 1st generation, acceleration (times)
Inter-core workload balancing based on graph preprocessing	6.4	3.1
Dynamic inter-core workload balancing	5.5	0.93
Concurrent processing of different group of vertices	1.9	0.8
SIMD-processing of low-degree vertices based on vector extension	1.4–5.2	2.1–8.1
SIMD-processing of low-degree vertices based on virtual warp	4.8	0.5
Graph clusterisation	5	3.8
Prefetching most frequently accessed vertices into LLC	–	1.2
Switching from push to pull-based graph traversal	0.9	1.7
Packing indirectly accessed 4-byte into 8-byte	1.1	1.3

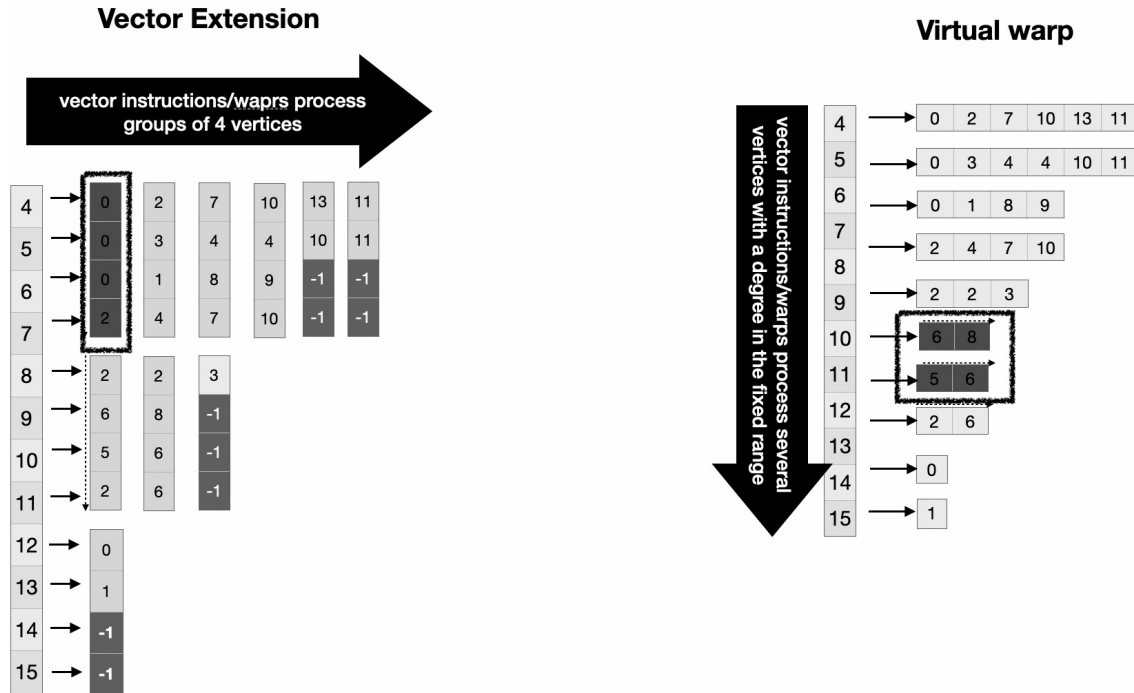
“virtual warp” concept [12]. Vector extension allows to process groups of *VECTOR\_LENGTH* vertices by simultaneously processing first edges of all vertices (using a single SIMD instruction), then second, and etc. Virtual warps concept is based on splitting SIMD instruction in separate parts, with each part processing vertices in a fixed range, as shown in Fig. 3. Vector extension allows to achieve a very significant and comparable acceleration on both architectures, while applying “virtual warp” concept to vector instructions leads to program slowdown on the NEC SX-Aurora TSUBASA architecture.

In order to efficiently load information about indirectly accessed graph vertices, the clusterisation [21] should be used for both architectures. The clusterisation is based on grouping information about most frequently accessed graph vertices in the adjacent regions of memory, which can be later prefetched into LLC cache (which allows to obtain an additional acceleration on the NEC SX-Aurora TSUBASA).

The performance of the advance abstraction also depends on the direction, in which graph edges are traversed. During pull traversal [7] in VGL the incoming edges are processed, while during push – the outgoing. According to our experiments, pull-direction is preferable for NEC SX-Aurora TSUBASA architecture, while push – on GPUs. Finally, for the NEC SX-Aurora TSUBASA multiple indirectly accessed values can be packed into 8-byte values, since gather and scatter instructions to 8-byte values are approximately 2 times faster compared to 4-byte values for this architecture. On NVIDIA GPUs, such optimisation does not provide any significant acceleration.

**Compute.** The implementation of the compute abstraction on both architectures is almost identical and straightforward, since all its operations can be performed independently in parallel.





**Figure 3.** Vector extension against virtual warps comparison. Red edges are simultaneously processed using a single vector instruction of length 4

On NVIDIA GPUs the compute abstraction is implemented via a parallel CUDA kernel, while on NEC SX-Aurora TSUBASA – via a vectorized parallel loop.

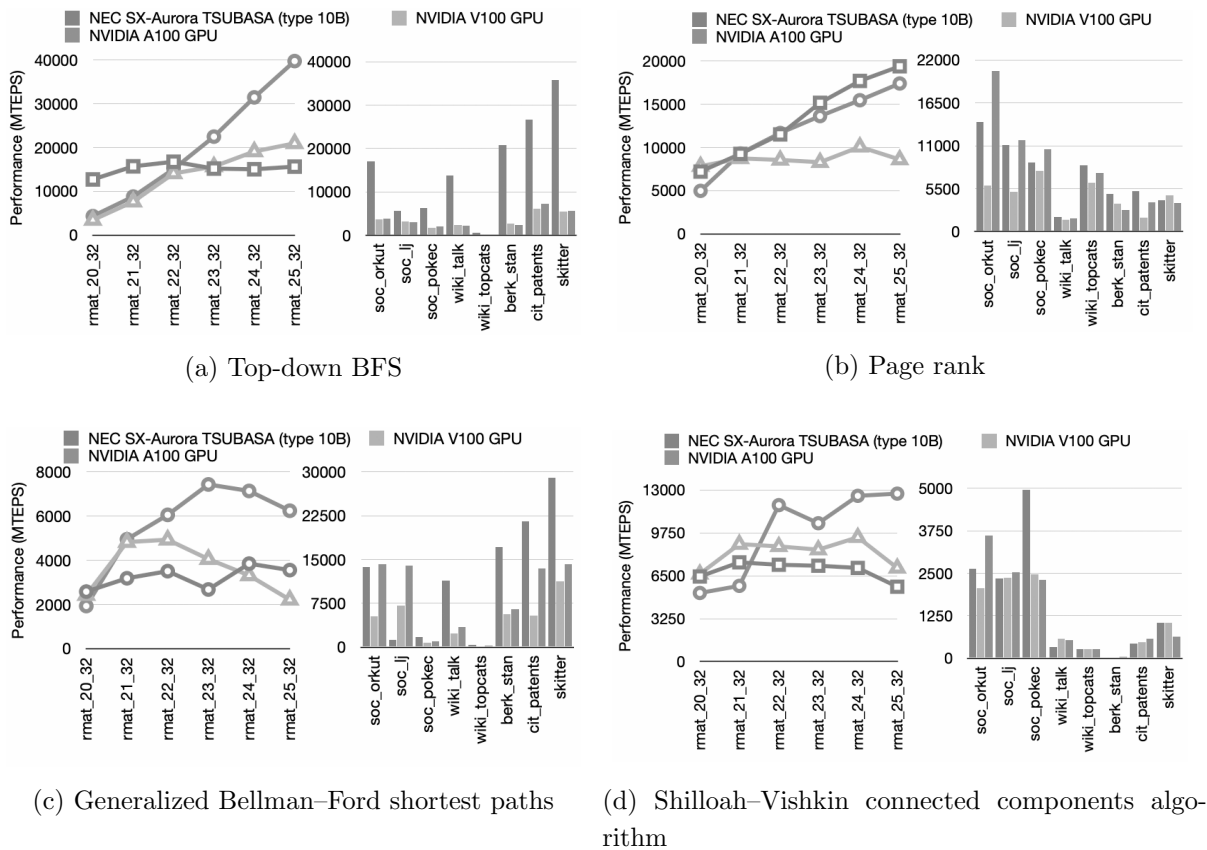
**Reduce.** For the NEC SX-Aurora TSUBASA architecture, the reduce abstraction is implemented via a vectorized and parallelized loop, where each vector core accumulates the reduced values on vector registers. This way the reduction vector instructions are executed only on the last stage of algorithm, when the obtained on vector registers values are reduced into the scalars. On the NVIDIA GPU architecture, the reduction is implemented based on using parallel reduction inside shared memory [10]. However, the reduce is implemented on NVIDIA GPUs less efficiently since shared memory has a higher latency compared to vector registers.

**Generate New Frontier.** The generate new frontier abstraction on GPUs is implemented based on parallel prefix sum algorithm [11], which generates indexes of vertices from the output frontier. On the NEC SX-Aurora TSUBASA a different algorithm is implemented [5], which generates lists of frontier indexes using special vector buffers, later unrolling them into a linear list. Both these approaches demonstrate approximately the same performance.

## 5. Performance Evaluation

The performance of the VGL framework has been evaluated on cluster equipped with (1) 12-core Intel (R) Xeon (R) Gold 6126 processors, (2) NVIDIA V100 and (3) A100 GPUs, and (4) SX-Aurora TSUBASA Type 10B (First Generation) vector engines. Unfortunately, at the moment of this writing we do not have an access to the second generation of SX-Aurora TSUBASA architecture. As input graphs we used synthetic RMat [8] and several real-world graphs from the SNAP [1] collection.

The performance evaluation is split into two stages. On the first stage we compared the performance of VGL-based implementations launched on SX-Aurora TSUBASA, V100 and A100 GPUs. This comparison is demonstrate in Fig. 4 for different graph problems and algorithms.



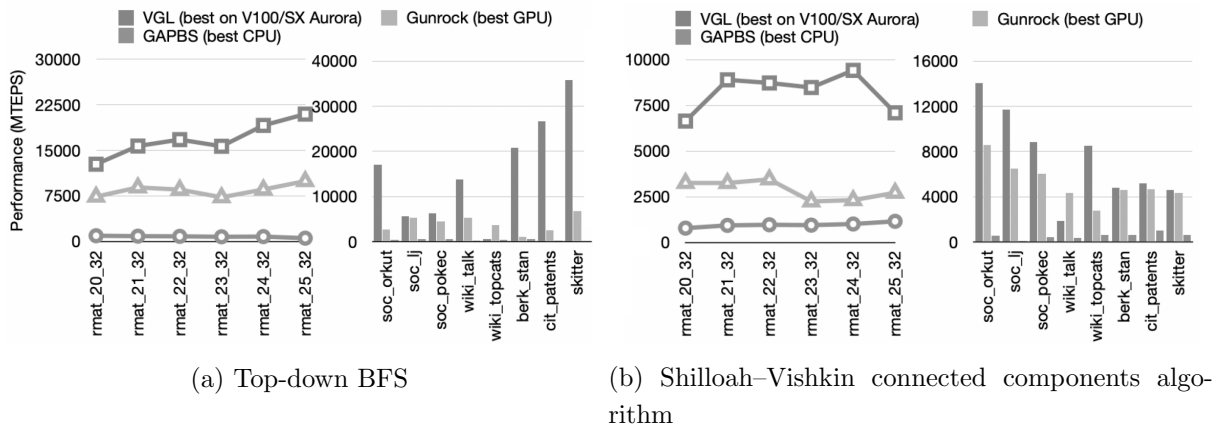
**Figure 4.** The comparison of VGL-based implementations of different graph problems and algorithms

The following conclusions can be made based on the provided performance data. The first generation NEC SX-Aurora TSUBASA vector engines have a comparable with V100 performance on medium-sized and large graphs, which once again confirms the thesis about the similarity of these architectures. At the same time, SX-Aurora implementations are significantly faster on small-sized RMT and most of real-world graphs, which have relatively small size. This can be explained by the fact that SX-Aurora requires significantly less vertices and edges, which have to be processed in parallel in order to efficiently utilize hardware resources.

At the second stage, the importance of developing of an architectural-independent framework is demonstrated in Fig. 5. For a specified graph problem and input graph we selected the fastest VGL-based implementation (among SX-Aurora and V100 GPU architectures), and compared it to the fastest available among CPU-based and GPU-based frameworks and libraries, listed in Section 2. The necessity of selecting different architectures can be explained by the fact that different architectures are faster at processing different input graphs, as shown in Fig. 4. According to our experiments, the Gunrock framework and NVGRAPH library provide the highest performance on NVIDIA GPUs, while GAPBS library has the highest performance among single-socket multicore CPU implementations. As shown in Fig. 5, VGL outperforms these frameworks and libraries from 3 to 15 times on different input graphs.

## Conclusion

In this paper we have described the first-in-the-world attempt to develop an architecture-independent graph framework VGL, which targets multiple modern systems with high-



**Figure 5.** The comparison of VGL-based implementations to best available multicore CPU and NVIDIA GPU frameworks

bandwidth memory: NEC SX-Aurora TSUBASA and NVIDIA GPUs. VGL has from 3 to 15 times better performance compared to the existing frameworks, developed for modern multicore CPUs and NVIDIA GPUs. Moreover, due its flexible software structure, the VLG framework can be easily extended to other massively parallel architectures, such as the A64FX, AMD EPYC Rome and Intel KNL, which is an important direction of future research.

Finally, in this paper we have compared optimisation approaches, which should be used in order to efficiently implement graph algorithms on NEC SX-Aurora TSUBASA vector processors and NVIDIA GPUs. Applying various optimisations, such as graph clusterisation or constructing graph vector extension, allowed to achieve similar acceleration on both these architectures, which emphasizes the similarity of these architectures in the context of implementing various graph algorithms.

## Acknowledgements

The reported study was funded by RFBR and JSPS, project number 21-57-50002.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Stanford Large Network Dataset Collection – SNAP. <https://snap.stanford.edu/data/> (2020), accessed: 2020-12-29
2. Afanasyev, I.V.: Developing a prototype of high-performance graph-processing framework for NEC SX–Aurora TSUBASA vector architecture. Numerical methods and programming 21, 290–305 (2020), DOI: 10.26089/NumMet.v21r325
3. Afanasyev, I.V., Voevodin, V.V., Kobayashi, H., et al.: Analysis of relationship between SIMD-processing features used in NVIDIA GPUs and NEC SX-Aurora TSUBASA vector processors. In: Malyshkin, V. (ed.) International Conference on Parallel Computing Technologies, PaCT 2019. Lecture Notes in Computer Science, vol. 11657, pp. 125–139. Springer (2019), DOI: 10.1007/978-3-030-25636-4\_10

4. Afanasyev, I.V., Voevodin, V.V., Kobayashi, H., et al.: Developing efficient implementations of shortest paths and page rank algorithms for NEC SX-Aurora TSUBASA architecture. *Lobachevskii Journal of Mathematics* 40(11), 1753–1762 (2019), DOI: 10.1134/S1995080219110039
5. Afanasyev, I.V., Voevodin, V.V., Komatsu, K., et al.: VGL: a high-performance graph processing framework for the NEC SX-Aurora TSUBASA vector architecture. *The Journal of Supercomputing* (2021), DOI: 10.1007/s11227-020-03564-9
6. Azad, A., Aznavah, M.M., Beamer, S., et al.: Evaluation of graph analytics frameworks using the GAP benchmark suite. In: *IEEE International Symposium on Workload Characterization, IISWC 2020, 27-30 October 2020, Beijing, China*. pp. 216–227. IEEE (2020), DOI: 10.1109/IISWC50251.2020.00029
7. Besta, M., Podstawski, M., Groner, L., et al.: To push or to pull: On reducing communication and synchronization in graph computations. In: Huang, H.H., Weissman, J.B., Iamnitchi, A., et al. (eds.) *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2017, 26-30 June 2017, Washington, DC, USA*. pp. 93–104. ACM (2017), DOI: 10.1145/3078597.3078616
8. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: Berry, M.W., Dayal, U., Kamath, C., et al. (eds.) *Proceedings of the Fourth SIAM International Conference on Data Mining, 22-24 April 2004, Lake Buena Vista, Florida, USA*. pp. 442–446. SIAM (2004), DOI: 10.1137/1.9781611972740.43
9. Flynn, M.J.: Very high-speed computing systems. *Proceedings of the IEEE* 54(12), 1901–1909 (1966), DOI: 10.1109/PROC.1966.5273
10. Harris, M., et al.: Optimizing parallel reduction in CUDA. *NVIDIA Developer Technology* 2(4), 70 (2007)
11. Hillis, W.D., Steele Jr., G.L.: Data parallel algorithms. *Commun. ACM* 29(12), 1170–1183 (1986), DOI: 10.1145/7902.7903
12. Hong, S., Kim, S.K., Oguntebi, T., et al.: Accelerating CUDA graph algorithms at maximum warp. *SIGPLAN Not.* 46(8), 267–276 (2011), DOI: 10.1145/2038037.1941590
13. Khorasani, F., Vora, K., Gupta, R., et al.: CuSha: vertex-centric graph processing on GPUs. In: Plale, B., Ripeanu, M., Cappello, F., et al. (eds.) *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, 23-27 June 2014, Vancouver, BC, Canada*. pp. 239–252. ACM (2014), DOI: 10.1145/2600212.2600227
14. Kirk, D.: NVIDIA CUDA software and GPU parallel computing architecture. In: Morrisett, G., Sagiv, M. (eds.) *Proceedings of the 6th International Symposium on Memory Management, ISMM 2007, 21-22 October 2007, Montreal, Quebec, Canada*. pp. 103–104. ACM (2007), DOI: 10.1145/1296907.1296909
15. Komatsu, K., Watanabe, O., Musa, A., et al.: Performance evaluation of a vector supercomputer SX-Aurora TSUBASA. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, 11-16 November 2018, Dallas, TX, USA*. pp. 54:1–54:12. IEEE Press (2018)

16. Liu, H., Huang, H.H.: Enterprise: breadth-first graph traversal on GPUs. In: Kern, J., Vetter, J.S. (eds.) Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, 15-20 November 2015, Austin, TX, USA. pp. 68:1–68:12. ACM (2015), DOI: 10.1145/2807591.2807594
17. Nguyen, D., Lenharth, A., Pingali, K.: A lightweight infrastructure for graph analytics. In: Kaminsky, M., Dahlin, M. (eds.) ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, 3-6 November 2013, Farmington, PA, USA. pp. 456–471. ACM (2013), DOI: 10.1145/2517349.2522739
18. Shun, J., Blelloch, G.E.: Ligra: a lightweight graph processing framework for shared memory. SIGPLAN Not. 48(8), 135–146 (2013), DOI: 10.1145/2517327.2442530
19. Wang, Y., Davidson, A.A., Pan, Y., et al.: Gunrock: a high-performance graph processing library on the GPU. In: Asenjo, R., Harris, T. (eds.) Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2016, 12-16 March 2016, Barcelona, Spain. pp. 11:1–11:12. ACM (2016), DOI: 10.1145/2851141.2851145
20. Yamada, Y., Momose, S.: Vector Engine Processor of NEC Brand-New supercomputer SX-Aurora TSUBASA. In: International symposium on High Performance Chips, Hot Chips 2018, August 2018, Cupertino, USA (2018)
21. Zhang, Y., Kiriansky, V., Mendis, C., et al.: Making caches work for graph analytics. In: Nie, J., Obradovic, Z., Suzumura, T., et al. (eds.) 2017 IEEE International Conference on Big Data, BigData 2017, 11-14 December 2017, Boston, MA, USA. pp. 293–302. IEEE Computer Society (2017), DOI: 10.1109/BigData.2017.8257937
22. Zhong, J., He, B.: Medusa: Simplified graph processing on GPUs. IEEE Trans. Parallel Distributed Syst. 25(6), 1543–1552 (2014), DOI: 10.1109/TPDS.2013.111

# Update on Performance Analysis of Different Computational Architectures: Molecular Dynamics in Application to Protein-Protein Interactions

*Vladimir A. Fedorov*<sup>1</sup>, *Ekaterina G. Kholina*<sup>1</sup>, *Ilya B. Kovalenko*<sup>1,2,3,4,5</sup>, *Nikita B. Gudimchuk*<sup>1,5</sup>, *Philipp S. Orekhov*<sup>1,6</sup>, *Artem A. Zhmurov*<sup>7</sup>

© The Authors 2020. This paper is published with open access at SuperFri.org

Molecular dynamics has proved itself as a powerful computer simulation method to study dynamics, conformational changes, and interactions of biological macromolecules and their complexes. In order to achieve the best performance and efficiency, it is crucial to benchmark various hardware platforms for the simulations of realistic biomolecular systems with different size and timescale. Here, we compare performance and scalability of a number of commercially available computing architectures using all-atom and coarse-grained molecular dynamics simulations of water and the Ndc80-microtubule protein complex in the GROMACS-2019.4 package. We report typical single-node performance of various combinations of modern CPUs and GPUs, as well as multiple-node performance of the “Lomonosov-2” supercomputer. These data can be used as the practical guidelines for choosing optimal hardware for molecular dynamics simulations.

*Keywords: molecular dynamics, coarse grain, tubulin, microtubule, Ndc80.*

## Introduction

Over the last decades, molecular dynamics (MD) simulations have become a powerful tool for investigating molecular systems (including protein assemblies) with the exceptionally high temporal and spatial resolutions unattainable so far using experimental techniques. Molecular systems of biological interest typically consist of up to millions of atoms, so MD simulations of biomacromolecules still represent a major computational challenge. The GROMACS package [1] is among the most efficient and popular engines for MD simulations as it runs efficiently on a wide variety of hardware from desktop workstations to supercomputers. Here, we evaluate which hardware combinations show optimal performance. This work continues the systematic comparison of the multiple currently available hardware architectures in terms of their MD simulation performance, which has been initiated in [3] for a number of biologically relevant molecular systems. In the present study, we use two types of molecular systems as computing benchmarks: (i) water boxes (WB) of different size and (ii) a biomolecular system consisting of the Ndc80 protein complex with a microtubule (MT) fragment [4]. Moreover, we extend our benchmark to coarse-grained (CG) MD simulations using MARTINI force field [7], a popular CG model for biomolecular simulations [8], using the same protein complex for testing. The MARTINI model is based on a four-to-one mapping, i.e., on average four heavy atoms plus associated hydrogens are represented by a single interaction center (called *a bead*). The overall aim of the coarse-graining approach is to provide a simple model that is computationally fast and easy to use, yet flexible enough to be applicable to a large range of biomolecular systems.

<sup>1</sup>Lomonosov Moscow State University, Moscow, Russia

<sup>2</sup>Federal Scientific and Clinical Center of Specialized Types of Medical Care and Medical Technologies, Federal Medical-Biological Agency of Russia, Moscow, Russia

<sup>3</sup>Astrakhan State University, Astrakhan, Russia

<sup>4</sup>Scientific and Technological Center of Unique Instrumentation of the RAS, Moscow, Russia

<sup>5</sup>Center for Theoretical Problems of Physicochemical Pharmacology, RAS, Moscow, Russia

<sup>6</sup>Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region, Russia

<sup>7</sup>KTH Royal Institute of Technology, Science for Life Laboratory, Stockholm, Sweden

The paper is organized as follows. In Section 1, we introduce the biomolecular systems used for benchmarking and describe the simulation setups employed for all-atom and coarse-grained simulations. In Section 2, we provide a comprehensive overview of performance achieved on various computational platforms. Finally, in Conclusion, we summarize the results and outline possible directions for further work.

## 1. Methods

All calculations were performed using the GROMACS-2019.4 version, which allows parallel computing on hybrid architectures. All benchmarks were run for 15 minutes. All-atom (AA) simulations were run in the explicit solvent using the TIP3P water model and the CHARMM27 force field for proteins. The production simulation runs were carried out in the NPT ensemble at 300K, using the Parrinello-Rahman algorithm [10] and the V-rescale thermostat for a duration of 1  $\mu$ s each. The structure of Ndc80 in complex with a MT fragment was obtained from the Protein Data Bank (PDB id 2VE7). The size of the virtual cell was chosen in such a way that the distance from the protein surface to the nearest box boundary was not less than two nanometers. For AA simulations the particle mesh Ewald (PME) method was used for the long-range electrostatics. Here, we used the interpolation order of 4 for PME, which equals cubic interpolation and should give electrostatic energies accurate to about  $5 \cdot 10^{-3}$ . The mass rescaling approach (i.e., partial transfer of mass from heavy atoms to the hydrogens bound to them) [5] allowed us to use 4-fs time step for AA MD simulations of the Ndc80 system instead of 1-fs time step used for the WB simulations and thus to accelerate them. Further details about the utilized MD protocol can be found in [4].

**Table 1.** Molecular dynamics systems used in the benchmark. Water box size is shortcutted in the system name, where the number stands for the thousands of particles in the system. Ndc80 is an acronym of a kinetochore protein

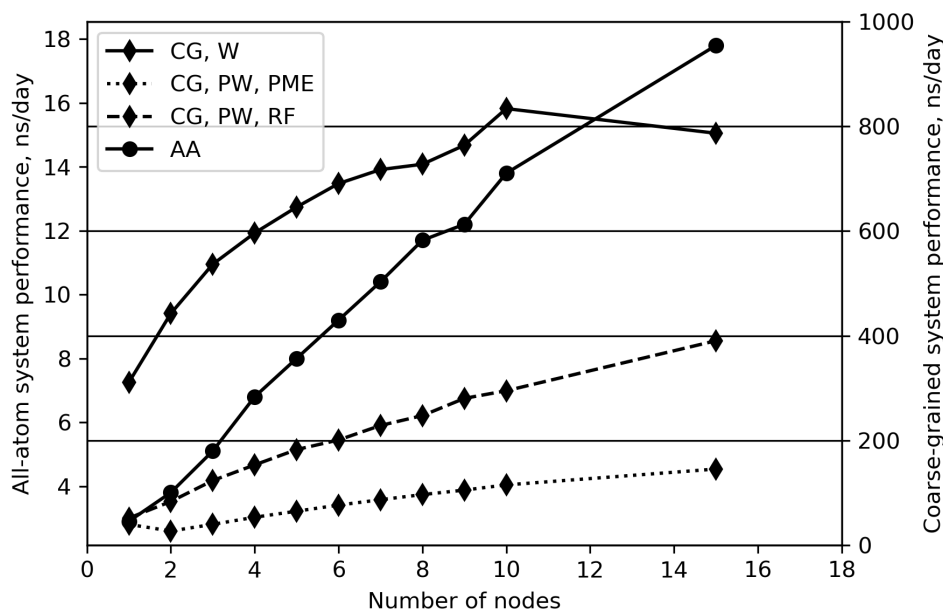
MD systems	MD system name	Number of particles	Box type	System size (nm)	Time step (fs)
Water box (WB)	WB-10	10206	cube	4.7x 4.7x 4.7	1
	WB-80	80232	cube	9.3x9.3x9.3	1
	WB-120	121527	cube	10.7x10.7x10.7	1
	WB-160	159780	cube	11.7x11.7x11.7	1
	WB-200	203415	cube	12.7x12.7x12.7	1
Ndc80 complex with microtubule	Ndc80 AA	750295	cube	22.1x17.2x20.1	4
	Ndc80 CG W	133371	cube	27x22x27	20
	Ndc80 CG PW	386294	cube	27x22x27	20

Coarse-grained simulations were run using the most recent version 2 of the MARTINI force field and the yet unreleased version 3 of this force field. The simulations with MARTINI 2 were run in combination with the polarizable water model (PW) [12], which allows for proper screening of interactions and other polarization effects. For these simulations, we utilized either

the PME or the reaction-field (RF) approach for the long-range electrostatics (with the cut-off  $r_c = 1.1$  nm, the dielectric constant beyond the cut-off was set to infinity [2]). The MARTINI beta version 3 is currently lacking an appropriately parameterized polarizable water model, so the simulations using this version of the CG force field were run with the standard water model (W) corresponding simply to a van der Waals particle. The RF approach was used for the long-range electrostatics in this case. The detailed protocols can be found elsewhere [6]. Specifications of all MD systems used for benchmarking are summarized in Tab. 1.

## 2. Results and Discussion

In order to examine the performance of MD simulations as a function of particle count both in homogeneous systems and in realistic systems, we have carried out MD simulations for a series of water boxes of the increasing size (see Tab. 1) and for the Ndc80-MT complex in the explicit solvent. In contrast to the previously reported benchmark [3], the updated results suggest that an increase of the system size leads to a commensurable decrease of computer performance in the explored range of system size (10,000–200,000 atoms), i.e., 20-fold increase in the number of atoms results in approximately equal decrease of performance for the CPU-only architecture. For GPU-accelerated simulations, we have found out even slower decrease of performance, which, for instance, scales down by the factor of 12.5 with the 20-fold increase of the system size for the RTX3080/Intel Core i9-9940X combination, see Tab. 2. However, for the largest system, which we have tested in the present benchmark, the all-atom Ndc80-MT complex consisting of over 750,000 atoms, the performance drop becomes disproportional. Overall, it implies an extremely high potential of the single-node hybrid architectures for the simulations of molecular systems with up to 100,000–200,000 atoms particularly emanating from the recent adaptations of the MD software for such platforms [9].



**Figure 1.** Performance of “Lomonosov-2” supercomputer (ns/day), depending on the number of computing nodes for all-atom (AA) and coarse-grained (CG) MD simulations of the Ndc80 complex with a fragment of MT



We have also addressed the question of scalability of MD simulations in GROMACS by estimating the dependence of the “Lomonosov-2” supercomputer [11] performance on the number of computer nodes used. For the all-atom Ndc80 system, as well as for the two CG systems with polarizable water (PW), the performance grows almost linearly as a function of the number of supercomputer nodes (Fig. 1) roughly following Amdahls law. However, for the smallest coarse-grained system (with the standard water model, W), the performance reaches the plateau at 10 nodes.

Finally, we have assessed the performance of two alternative schemes commonly used to treat the long-range electrostatics in CG simulations: PME and RF. For all the tested systems, RF outperforms PME suggesting the former as the best option. However, the difference is the strongest for the no-GPU platforms where it can be as large as 7-fold. For the GPU-accelerated platforms, a much lower handicap of 1.2–3.5 is observed.

**Table 2.** Single-node performance (ns/day) for systems of different size depending on various combinations of CPUs and GPUs

GPU	MD system								
	WB-10	WB-80	WB-120	WB-160	WB-200	Ndc80 AA	Ndc80 CG W	Ndc80 CG PW PME	Ndc80 CG PW RF
no GPU*	100.3	14.1	9.0	6.6	5.2	4.2	910.8	25.9	153.8
RTX 2070 Super*	290.8	48.5	32.3	25.8	20.9	15.1	1657.5	152.2	260.9
RTX 2080 Ti*	349.9	62.6	43.8	34.5	26.8	19.5	1750.2	187.5	286.6
RTX 3080*	365.8	67.9	47.5	37.3	29.4	23.5	1817.6	252.4	330.4
no GPU**	113.6	16.5	10.5	8.1	6.3	5.7	1127.8	24.4	181.4
RTX 2080 Ti**	338.5	45.5	29.1	22.2	16.5	14.3	1993.1	207.7	331.4
2 RTX 2080 Ti**	207.4	49.6	30.0	22.6	17.5	14.7	2232.4	88.2	279.3

\* Nodes with the Intel Core i9-7900X CPU.

\*\* Nodes with the Intel Core i9-9940X CPU.

## Conclusion

Our comparative performance analysis of molecular dynamics software (using the popular GROMACS package as an example) provides the guidelines for selection of the best-performing GPU-based architectures for both all-atom and coarse-grained MD simulations of realistic molecular systems. It also outlines certain limitations of the single-node workstations and highlights the importance of the HPC platforms (e.g., “Lomonosov-2” supercomputer) for the simulations of large systems exceeding ca. 200,000 particles on the relevant time scale.

In the case of the AA MD simulations, the presence of a modern graphics accelerator speeds up the calculations by about 5 times both for a 100 thousand atom system, and a million atom system. Moreover, as the size of the system grows, the acceleration increases slightly. However, in the case of CG models, the increase in performance with a modern GPU is not so impressive – only about 2 times. In this case, the multicore and multiprocessor computing architectures are also very important.

When using a parallel supercomputer 15 nodes (or more) are optimal for both AA and CG calculations. However, for CG systems with non-polarizable water, 10 nodes is the optimal choice, since with a further increase in the number of nodes, performance begins to decline.

## Acknowledgments

This study was designed and conducted with support from the Russian Foundation for Basic Research, projects 17-00-00479, 17-00-00482 and 19-04-00999. The research was carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University supported by the project RFMEFI62117X0011.







*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Abraham, M.J., Murtola, T., Schulz, R., et al.: GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2, 19–25 (2015), DOI: 10.1016/j.softx.2015.06.001
2. Abraham, M., van der Spoel, D., Lindahl, E., et al.: The GROMACS development team GROMACS User Manual, version 2019. <http://www.gromacs.org> (2019)
3. Fedorov, V.A., Kholina, E.G., Kovalenko, I.B., et al.: Performance analysis of different computational architectures: Molecular dynamics in application to protein assemblies, illustrated by microtubule and electron transfer proteins. *Supercomputing Frontiers and Innovations* 5(4), 111–114 (2018), DOI: 10.14529/jsfi180414
4. Fedorov, V.A., Orekhov, P.S., Kholina, E.G., et al.: Mechanical properties of tubulin intra- and inter-dimer interfaces and their implications for microtubule dynamic instability. *PLoS computational biology* 15(8), e1007327 (2019), DOI: 10.1371/journal.pcbi.1007327
5. Feenstra, K.A., Hess, B., Berendsen, H.J.C.: Improving efficiency of large time-scale molecular dynamics simulations of hydrogen-rich systems. *Journal of Computational Chemistry* 20(8), 786–798 (1999), DOI: 10.1002/(SICI)1096-987X(199906)20:8<786::AID-JCC5>3.0.CO;2-B
6. Kholina, E.G., Kovalenko, I.B., Bozdaganyan, M.E., et al.: Cationic antiseptics facilitate pore formation in model bacterial membranes. *The Journal of Physical Chemistry B* 124(39), 8593–8600 (2020), DOI: 10.1021/acs.jpcc.0c07212
7. Marrink, S.J., Risselada, H.J., Yefimov, S., et al.: The MARTINI force field: coarse grained model for biomolecular simulations. *The journal of physical chemistry B* 111(27), 7812–7824 (2007), DOI: 10.1021/jp071097f
8. Monticelli, L., Kandasamy, S.K., Periole, X., et al.: The MARTINI coarse-grained force field: extension to proteins. *Journal of chemical theory and computation* 4(5), 819–834 (2008), DOI: 10.1021/ct700324x
9. Páll, S., Zhmurov, A., Bauer, P., et al.: Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS. *The Journal of Chemical Physics* 153(13), 134110 (2020), DOI: 10.1063/5.0018516

10. Parrinello, M., Rahman, A.: Polymorphic transitions in single crystals: A new molecular dynamics method. *Journal of Applied physics* 52(12), 7182–7190 (1981), DOI: 10.1063/1.328693
11. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., et al.: Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community. *Supercomputing Frontiers and Innovations* 6(2), 4–11 (2019), DOI: 10.14529/jsfi190201
12. Yesylevskyy, S.O., Schäfer, L.V., Sengupta, D., et al.: Polarizable water model for the coarse-grained MARTINI force field. *PLoS Comput Biol* 6(6), e1000810 (2010), DOI: 10.1371/journal.pcbi.1000810

# Computer Design of Structure of Molecules of High-Energy Tetrazines. Calculation of Thermochemical Properties

Vadim M. Volokhov<sup>1</sup> , Elena S. Amosova<sup>1</sup> , Alexander V. Volokhov<sup>1</sup> ,  
Tatiana S. Zyubina<sup>1</sup> , David B. Lempert<sup>1</sup> , Leonid S. Yanovskiy<sup>1,2,3</sup> ,  
Ilya D. Fateev<sup>4</sup>

© The Authors 2020. This paper is published with open access at SuperFri.org

The article presents high-performance calculations, using quantum chemical ab initio methods, of thermochemical characteristics of high-energy compounds:  $C_2N_6O_4$ ,  $C_2N_6O_5$ ,  $C_2N_6O_6$ ,  $C_2H_2N_6O_4$ ,  $C_3HN_7O_6$ ,  $C_3HN_7O_4F_2$ ,  $C_4N_{10}O_{12}$ ,  $C_3HN_6O_4F$ ,  $C_4N_{10}O_8F_4$ ,  $C_4N_8O_8F_2$ . The IR absorption spectra, structural parameters and atomic displacements for the most intense vibrations, as well as the enthalpies of formation are provided in the article. The calculations were performed at the B3LYP/6-311+G(2d,p) level and using the combined methods CBS-4M and G4 within the Gaussian 09 application package (Linda parallelization). It is shown that the enthalpy of formation depends on the molecule structure.

*Keywords:* high-performance computing, enthalpy of formation, quantum-chemical calculations, high-enthalpy compounds, IR spectra of gaseous molecules, combined CBS-4M method, combined G4 method.

## Introduction

The rapidly developing new technologies, especially in the field of engines for advanced aircraft, stimulate great interest in the creation of high-energy materials for various purposes. In the recent time, modern computer technologies have played an increasingly important role in the creation of new materials with specific properties [1–3].

The value of the standard enthalpy of formation  $\Delta H_f^\circ$  of a chemical compound is one of the main criteria for its energy intensity, thus the determination of this value, both experimental and calculated, becomes a key task for assessing the efficiency of using a particular substance as a component of high-energy materials. In terms of thermochemistry, it is important to know how the enthalpy of formation of a substance changes with some structural changes. One of the most reliable methods for studying the dependences of the enthalpy of formation on various parameters characterizing molecules (along with the experiment) is quantum-chemical ab initio calculations using high-precision combined methods, such as CBS-4M and G4 within the Gaussian 09 package.

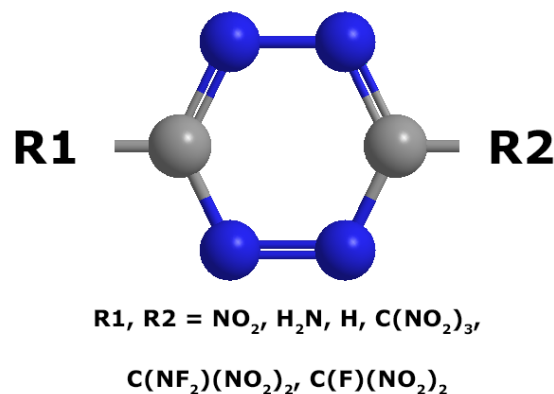
This work is dedicated to determination by quantum chemical methods of  $\Delta H_f^\circ$  of a number of high-enthalpy compounds with the common structure as shown in the Fig. 1. All these compounds were designed by the authors and are promising as components of fuel for various purposes. They have not been synthesized yet, and their thermochemical properties have been studied for the first time in this work. It should be noted that for molecules of such complexity, the spread of experimental data might be larger than that of calculated one, as it has been observed in one of our previous works [4].

<sup>1</sup>Institute of Problems of Chemical Physics of the Russian Academy of Sciences, Chernogolovka, Moscow Region, Russia

<sup>2</sup>Central Institute of Aviation Motors, Moscow, Russia

<sup>3</sup>Moscow Aviation Institute (National Research University), Moscow, Russia

<sup>4</sup>Research Computing Center Lomonosov Moscow State University, Moscow, Russia



**Figure 1.** Structure of the studied molecules

The article is organized as follows. In Section 1 we describe the method we used to determine the enthalpy of formation of the given molecules. In Section 2 we discuss the results of our calculations: enthalpies of formation and IR spectra. Section 3 contains computational details of our calculations. Conclusion summarizes the study.

## 1. Calculation Method

The enthalpy of formation of the investigated gaseous molecules was calculated by the atomization method as was described in our previous works [4, 5]. The simulation was performed within the Gaussian 09 package [6] using the hybrid density functional B3LYP [7, 8] with the basis set 6-311+G(2d,p), which had already proven itself in molecular calculations, and the combined methods CBS-4M [9, 10] and G4 [11, 12].

The CBS series of methods was introduced by Petersson and colleagues [9, 10, 13–17] and includes three main methods listed in ascending order of accuracy and estimated time: CBS-4M, CBS-QB3, CBS-APNO. To obtain an accurate energy value, these methods use both additive and extrapolation schemes, and empirical corrections. We used the CBS-4M method in our work as the least demanding on computational resources. The CBS-4 [9] method which is a base for the CBS-4M one uses low level theory UHF/3-21G(\*) for geometry optimization and zero-point energies and a series of further energy calculations for a given geometry using large basis sets for the SCF method, medium size ones for MP2 calculations, and small basis sets for the higher level method. The modifications introduced in the CBS-4M method are the use of the minimum population localization in the CBS extrapolation, the change in the two-electron empirical parameter and corrections for spin-orbit interactions in atomic energies [10].

The G4 method was introduced in 2007 by Curtiss and colleagues [11]. It uses calculations at the B3LYP/6-31G(2df,p) level for geometry optimization and zero-point energies. The Hartree-Fock energy limit is calculated using a linear two-point extrapolation scheme and Dunning's basis sets [18–22]. The G4 method for approximating the energies of more accurate calculations combines the CCSD (T) method with a sufficiently high level of electron correlation and a medium-sized basis set (6-31G (d)) with the energies from the calculations of a lower level of the theory (MP4 and MP2) with large basis sets. In addition, the remaining errors are taken into account by several empirical corrections that do not depend on the studied molecule.

The main steps of the calculation by the atomization method of the enthalpy of formation of compounds with the common formula  $C_wH_xN_yO_zF_p$  are listed below:

1. Calculation of the atomization energy in the nonrelativistic approximation.

$$\sum D_0 = wE_0(C) + xE_0(H) + yE_0(N) + zE_0(O) + pE_0(F) - E_0(C_wH_xN_yO_zF_p), \quad (1)$$

where  $E_0(C)$ ,  $E_0(H)$ ,  $E_0(N)$ ,  $E_0(O)$ ,  $E_0(F)$  are calculated total energies of atoms. Total energy of the molecule  $E_0(C_wH_xN_yO_zF_p)$  is calculated by the formula  $E_0(C_wH_xN_yO_zF_p) = \varepsilon_0 + ZPE$ , where  $\varepsilon_0$  is a total energy of the molecule, and ZPE is a sum of zero-point energies of all vibrational modes of the molecule.

2. Calculation of the enthalpy of formation at  $0K$

$$\begin{aligned} \Delta H_f^\circ(C_wH_xN_yO_zF_p, 0K) = & w\Delta H_f^\circ(C, 0K) + x\Delta H_f^\circ(H, 0K) + \\ & + y\Delta H_f^\circ(N, 0K) + z\Delta H_f^\circ(O, 0K) + p\Delta H_f^\circ(F, 0K) - \sum D_0, \end{aligned} \quad (2)$$

where the first five summands are the enthalpies of formation of gaseous atomic components known from experiment.

3. Calculation of the enthalpy of formation at  $298.15K$

$$\begin{aligned} \Delta H_f^\circ(C_wH_xN_yO_zF_p, 298K) = & \Delta H_f^\circ(C_wH_xN_yO_zF_p, 0K) + (H^0(C_wH_xN_yO_zF_p, 298K) - \\ & - H^0(C_wH_xN_yO_zF_p, 0K)) - w(H^0(C, 298K) - H^0(C, 0K)) - \\ & - x(H^0(H, 298K) - H^0(H, 0K)) - y(H^0(N, 298K) - H^0(N, 0K)) - \\ & - z(H^0(O, 298K) - H^0(O, 0K)) - p(H^0(F, 298K) - H^0(F, 0K)), \end{aligned} \quad (3)$$

where the second summand is obtained from the calculation of the molecule, the third to seventh summands are known from experiment (or calculated from experimental molecular constants). The values of the enthalpy of formation of gaseous atoms and thermal corrections can be taken from various reference books or literature sources, for example [23–27].

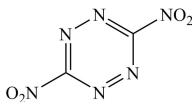
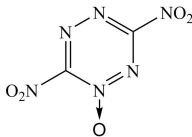
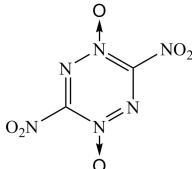
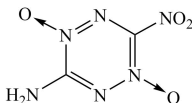
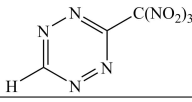
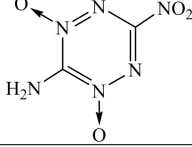
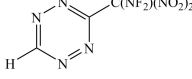
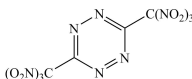
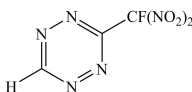
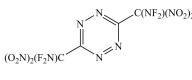
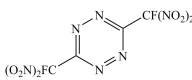
In this work, we used the experimental atomic enthalpies of formation from the NIST-JANAF thermochemical tables [24]. Since the theoretical calculation systematically overestimates the values of the zero-point frequencies, the frequencies are corrected using empirically selected coefficients. To obtain more accurate frequencies, it is necessary to correct the vibration frequencies when calculating the ZPE corrections and corrections ( $H^0(C_wH_xN_yO_zF_p, 298K) - H^0(C_wH_xN_yO_zF_p, 0K)$ ). The values of the scaling factors are used for this, which are recommended in the literature for various calculation methods and various basis sets [6].

## 2. Results and Discussion

### 2.1. Enthalpy of Formation

Table 1 shows the structures of the calculated molecules, molecular weights, enthalpies of formation (in kcal/mol, kJ/mol, kJ/kg) obtained at different calculation levels: B3LYP/6-311+G(2d,p), CBS-4M and G4. Figure 2 and Fig. 3 show the structural parameters and the change in the enthalpies of formation of the calculated gaseous molecules.

**Table 1.** Chemical and structural formulae, molecular weights (Mw, in amu), enthalpies of formation  $\Delta H_f^\circ$  of calculated molecules (in kcal/mol – regular font, *kJ/mol* – italic, **kJ/kg** – bold), obtained at the B3LYP/6-311+G(2d,p), CBS-4M and G4 levels

N M <sub>w</sub>	Structural formula	B3LYP/6-311+G(2d,p)	CBS-4M	G4
<b>1</b> <i>C<sub>2</sub>N<sub>6</sub>O<sub>4</sub></i> 172.06		139.88 <i>585.25</i> <b>3401.42</b>	130.04 <i>544.08</i> <b>3162.16</b>	128.76 <i>538.74</i> <b>3131.12</b>
<b>2</b> <i>C<sub>2</sub>N<sub>6</sub>O<sub>5</sub></i> 188.06		133.51 <i>558.59</i> <b>2970.28</b>	124.31 <i>520.10</i> <b>2765.61</b>	120.84 <i>505.59</i> <b>2688.49</b>
<b>3</b> <i>C<sub>2</sub>N<sub>6</sub>O<sub>6</sub></i> 204.06		136.4 <i>570.72</i> <b>2796.84</b>	128.14 <i>536.14</i> <b>2627.37</b>	122.72 <i>513.47</i> <b>2516.30</b>
<b>4</b> <i>C<sub>2</sub>H<sub>2</sub>N<sub>6</sub>O<sub>4</sub></i> 174.07		105.24 <i>440.35</i> <b>2529.63</b>	104.93 <i>439.03</i> <b>2522.09</b>	97.67 <i>408.66</i> <b>2347.59</b>
<b>5</b> <i>C<sub>3</sub>HN<sub>7</sub>O<sub>6</sub></i> 231.08		152.63 <i>638.6</i> <b>2763.49</b>	125.99 <i>527.16</i> <b>2281.26</b>	124.95 <i>522.76</i> <b>2262.36</b>
<b>6</b> <i>C<sub>2</sub>H<sub>2</sub>N<sub>6</sub>O<sub>4</sub></i> 174.07		99.38 <i>415.79</i> <b>2388.58</b>	94.61 <i>395.83</i> <b>2273.92</b>	91.53 <i>382.95</i> <b>2199.92</b>
<b>7</b> <i>C<sub>3</sub>HN<sub>7</sub>O<sub>4</sub>F<sub>2</sub></i> 237.08		143.87 <i>601.96</i> <b>2539.06</b>	120.94 <i>506.00</i> <b>2134.29</b>	118.04 <i>493.86</i> <b>2083.10</b>
<b>8</b> <i>C<sub>4</sub>N<sub>10</sub>O<sub>12</sub></i> 380.1		192.74 <i>806.43</i> <b>2121.61</b>	142.90 <i>597.91</i> <b>1573.02</b>	138.54 <i>579.64</i> <b>1524.97</b>
<b>9</b> <i>C<sub>3</sub>HN<sub>6</sub>O<sub>4</sub>F</i> 204.06		91.98 <i>384.86</i> <b>1885.86</b>	70.45 <i>294.84</i> <b>1444.75</b>	71.18 <i>297.81</i> <b>1459.29</b>
<b>10</b> <i>C<sub>4</sub>N<sub>10</sub>O<sub>8</sub>F<sub>4</sub></i> 392.1		174.96 <i>732.02</i> <b>1866.94</b>	132.66 <i>555.05</i> <b>1415.58</b>	124.29 <i>520.05</i> <b>1326.33</b>
<b>11</b> <i>C<sub>4</sub>N<sub>8</sub>O<sub>8</sub>F<sub>2</sub></i> 326.1		71.07 <i>297.35</i> <b>911.87</b>	31.74 <i>132.81</i> <b>407.28</b>	30.63 <i>128.14</i> <b>392.97</b>





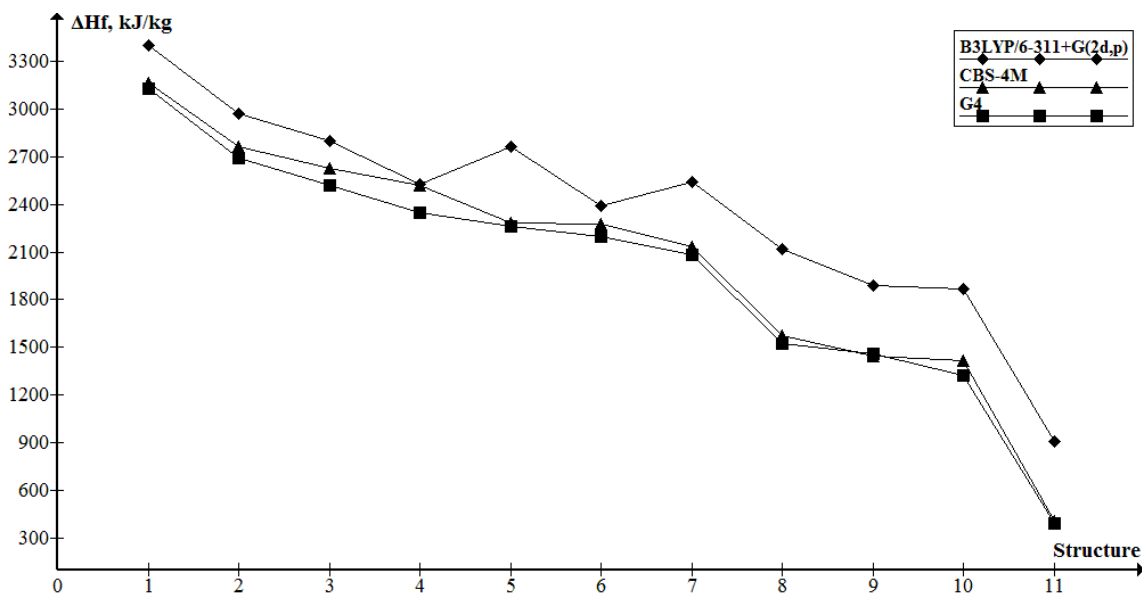


Figure 3. Enthalpies of formation of structures 1–11

method). The addition of the first oxygen atom lowers the enthalpy of formation by 429 kJ/kg, and the effect of the addition of the second one is significantly lower (166 kJ/kg).

The transition from structure **3** to **4** by replacing one of the  $NO_2$  fragments with  $NH_2$  leads to a decrease in the enthalpy of formation by 172 kJ/kg.

In the case when oxygen atoms are located opposite each other (structure **4**), the enthalpy of formation is higher (by 153 kJ/kg) than the one for the isomer with structure **6**, where both oxygen atoms are located near the  $NH_2$  fragment.

$C_3HN_7O_6$ . The molecules in the series from structure **1** to structure **5** are rearranged by replacing one  $NO_2$  fragment with  $H$  and the other with  $C(NO_2)_3$ . This rearrangement is accompanied by a decrease in the enthalpy of formation by 854 kJ/kg. Replacement in structure **5** of one of the  $NO_2$  fragments by  $NF_2$  (structure **7**) or  $F$  (structure **9**) leads to a decrease in the enthalpy of formation by 193 kJ/kg and 825 kJ/kg, respectively. The presence of two oppositely located  $C(NO_2)_3$  groups in structure **8** leads to a decrease in the enthalpy of formation in comparison with structure **5** by 718 kJ/kg.

## 2.2. IR Absorption Spectra

The IR absorption spectra and displacements of atoms for the most intense vibrations are shown in Fig. 4, Fig. 5, Fig. 6 and Fig. 7 correspondingly.

The highest frequencies (from  $1628\text{ cm}^{-1}$  to  $1670\text{ cm}^{-1}$ ) with noticeable intensities are observed in the  $NO_2$  groups, which are part of the  $R1$  and  $R2$  fragments in almost all the structures under consideration. The exceptions are structures **4** and **6**, in which the most intense vibrations are with frequency of  $1457\text{ cm}^{-1}$  for structure **4** and  $1398\text{ cm}^{-1}$  for structure **6**, corresponding to the vibrations of the  $NO$  bond of nitrogen atoms in the ring. Vibrations along the  $C - N$  bond of the nitrogen atom belonging to the  $NO_2$  group are characterized by frequencies with lower intensity:  $1333\text{ cm}^{-1}$  for structure **5**,  $1330\text{ cm}^{-1}$  for structure **8**,  $1350\text{ cm}^{-1}$  for structure **9**,  $1335\text{ cm}^{-1}$  for structure **10**, and  $1348\text{ cm}^{-1}$  for structure **11**. Asynchronous vibrations of carbon and nitrogen atoms included in the ring are also characterized by frequencies with noticeable intensity:  $1355\text{ cm}^{-1}$  for structure **2** and  $1455\text{ cm}^{-1}$  for structure **3**.

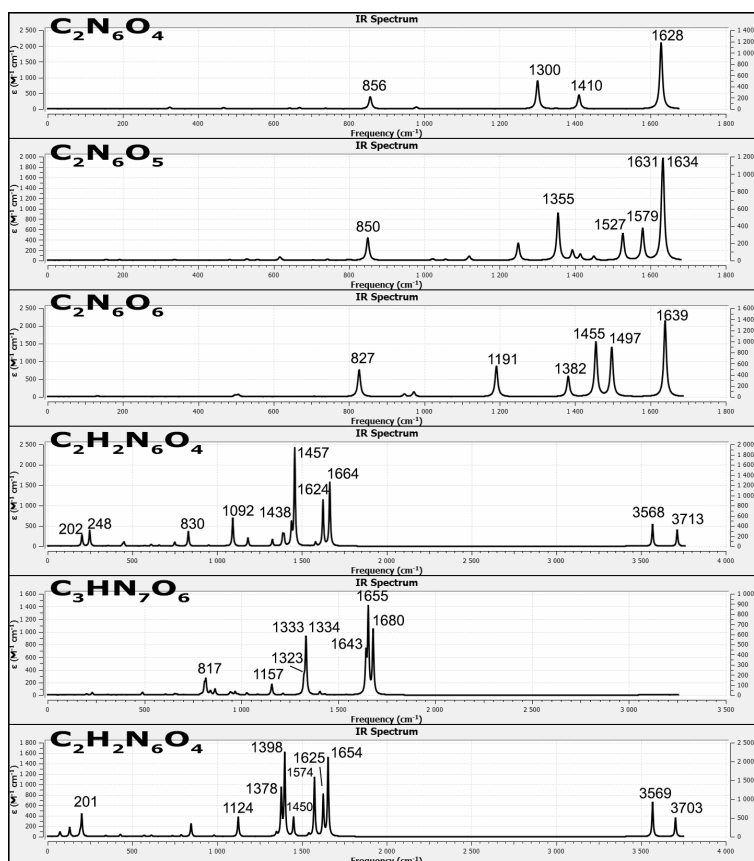


Figure 4. IR absorption spectra for the calculated compounds (structures 1–6)

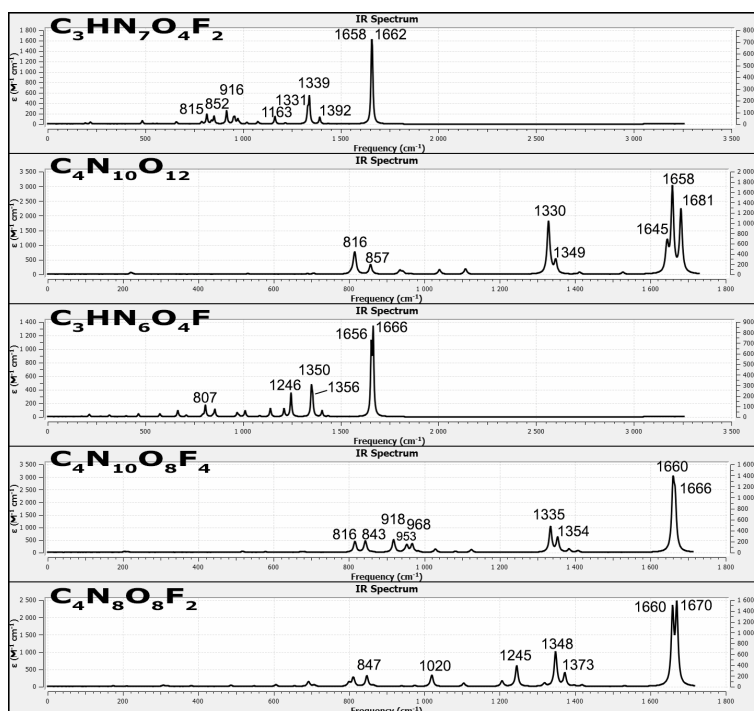


Figure 5. IR absorption spectra for the calculated compounds (structures 7–11)

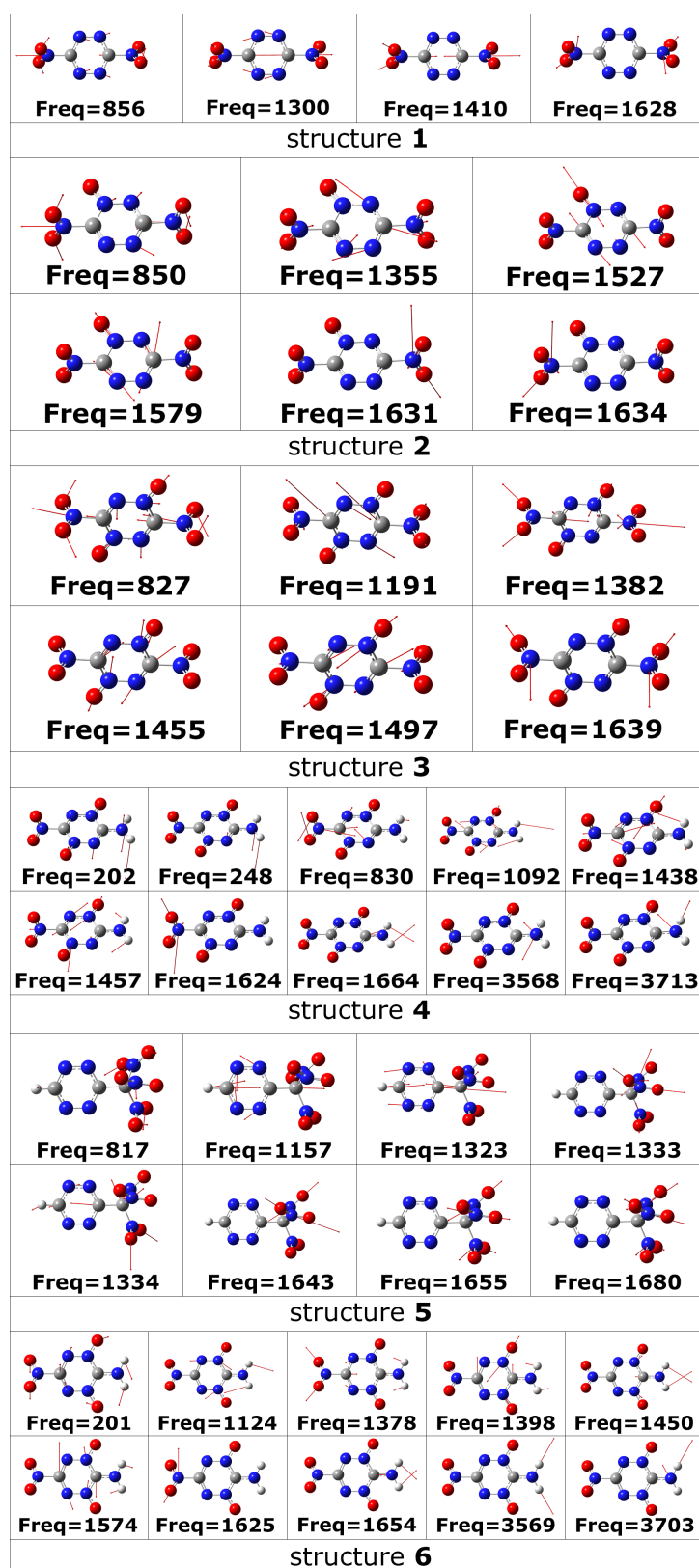


Figure 6. Displacement vectors for the indicated frequencies (structures 1–6)

### 3. Computational Details

Various computational sources have been used for the quantum chemical simulation depending on the complexity of the task. Simulations of the simpler molecules have been performed on

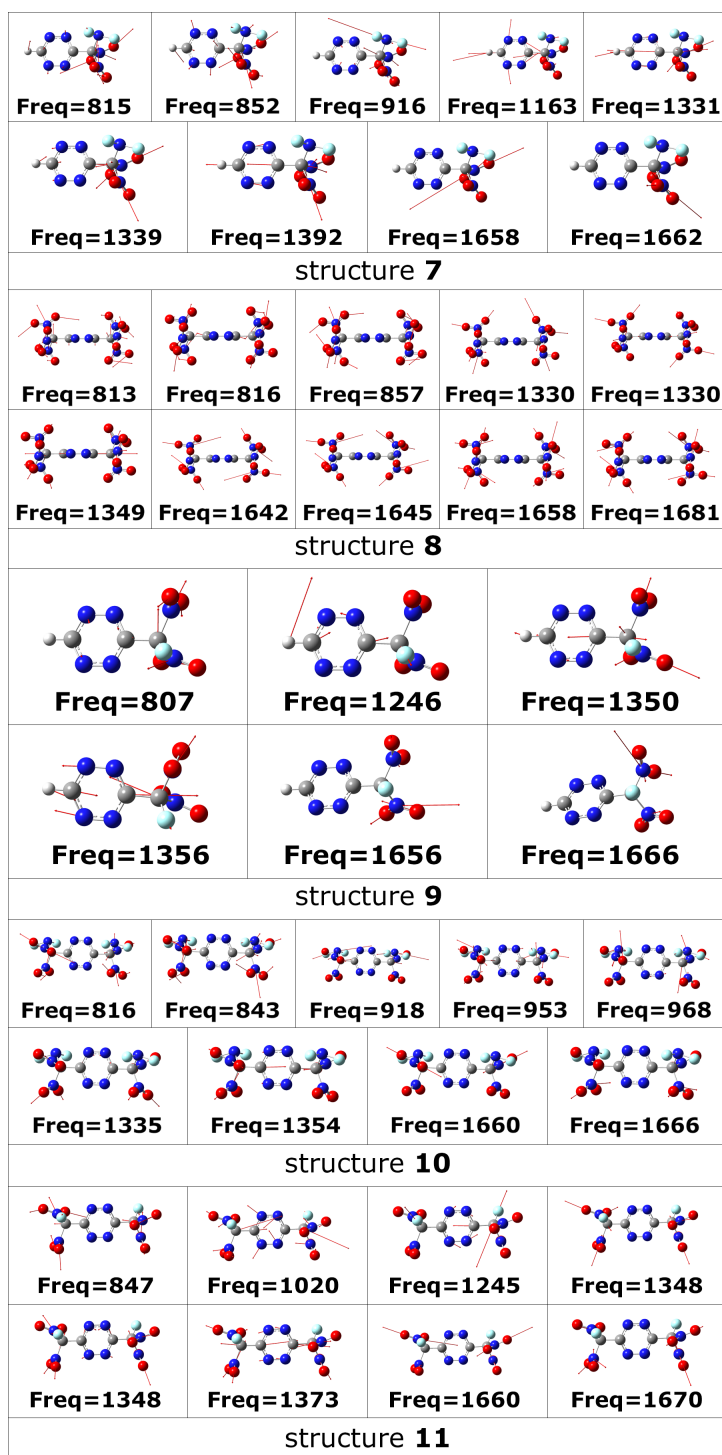


Figure 7. Displacement vectors for the indicated frequencies (structures 7–11)

the local computational resources of the Institute of Problems of Chemical Physics of the Russian Academy of Sciences using the minicluster with computational nodes consisting of two Intel Xeon®X5675@3.46GHz processors, 48 Gb RAM and two GPU Nvidia Tesla C2075 per node, and computational cluster with nodes consisting of Intel Xeon®5450-5670@3GHz processors and 8 and 12 Gb RAM. The computation time for the given tasks have amounted to several hours approximately. For the computationally intensive tasks, we have used the high-performance computing resources of the Lomonosov Moscow State University with computational nodes consisting

of Intel Xeon®Gold 6140 CPU @2.30GHz processors and 256 Gb per node. The computation time on the heaviest structures by the most expensive G4 method amounted to two months.

## Conclusion

Thermochemical data of high-energy compounds:  $C_2N_6O_4$ ,  $C_2N_6O_5$ ,  $C_2N_6O_6$ ,  $C_2H_2N_6O_4$ ,  $C_3HN_7O_6$ ,  $C_3HN_7O_4F_2$ ,  $C_4N_{10}O_{12}$ ,  $C_3HN_6O_4F$ ,  $C_4N_{10}O_8F_4$ ,  $C_4N_8O_8F_2$  has been obtained as a result of high-precision quantum-chemical calculations. It is shown that the enthalpy of formation decreases with increasing complexity of the structure of the molecules of the studied series. The IR absorption spectra, structural parameters, and atomic displacements for the most intense vibrations are also calculated. The calculations have been performed at the B3LYP/6-311+G(2d,p) level and using the combined CBS-4M and G4 methods. It has been demonstrated that the use of the CBS-4M calculation level for the selected class of molecules gives results that are close (within 7%) to those obtained at the G4 level and leads to a significant (up to 10 times) savings in the calculation time, although the values of the enthalpy of formation obtained by this method are either overstated or understated in comparison to the results of the G4 calculation. All molecules have been designed in the work for the first time and their physicochemical properties have been calculated and described for the first time.

## Acknowledgements

The work was performed using the equipment of the Center for Collective Use of Super High-Performance Computing Resources of the Lomonosov Moscow State University [28–30] (project Enthalpy-2065) and computational resources of the Institute of Problems of Chemical Physics of the Russian Academy of Sciences. This work was carried out with the support of the RFBR grant No. 20-07-00319. Atom displacements were calculated in the framework of State Assignment No. AAAA-A19-119120690042-9. The IR spectra were calculated within State Assignment No. AAAAA19-119061890019-5.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Hosseini, S.G., Moeini, K., Abdelbaky, M.S.M., et al.: Synthesis, Characterization, Crystal Structure, and Thermal Behavior of New Triazolium Salt Along with Docking Studies. *J. Struct. Chem.* 61, 366–376 (2020), DOI: 10.1134/S002247662003004X
2. Abdulov, K.S., Mulloev, N.U., Tabarov, S.K., et al.: Quantum Chemical Determination of the Molecular Structure of 1,2,4-Triazole and the Calculation of its Infrared Spectrum. *J. Struct. Chem.* 61, 510–514 (2020), DOI: 10.1134/S0022476620040022
3. Lv, G., Zhang, D.L., Wang, D., et al.: Synthesis, Crystal Structure, Anti-Bone Cancer Activity and Molecular Docking Investigations of the Heterocyclic Compound 1-((2S,3S)-2-(Benzyloxy)Pentan-3-yl)-4-(4-(4-(4-Hydroxyphenyl)Piperazin-1-yl)Phenyl)-1H-1,2,4-Triazol-5(4H)-One. *J. Struct. Chem.* 60, 1173–1179 (2019), DOI: 10.1134/S0022476619070205

4. Volokhov, V.M., Zyubina, T.S., Volokhov, A.V., et al.: Predictive Modeling of Molecules of High-Energy Heterocyclic Compounds. *Russian Journal of Inorganic Chemistry* 66 (2021) 69–80, DOI: 10.1134/S0036023621010113
5. Volokhov, V.M., Zyubina, T.S., Volokhov, A.V., et al.: Quantum Chemical Simulation Of Hydrocarbon Compounds With High Enthalpy Of Formation. *Russian Journal of Physical Chemistry B: Focus on Physic* 40 (2021) 3–15 (in Russian), DOI: 10.31857/S0207401X21010131
6. Frisch, M.J., Trucks, G.W., Schlegel, H.B., et al.: *Gaussian 09, Revision B.01*. Gaussian, Inc., Wallingford CT, 2010
7. Becke, A.D.: Densityfunctional thermochemistry. III. The role of exact exchange. *J. Chem. Phys.* 98, 5648 (1993), DOI: 10.1063/1.464913
8. Johnson, B.J., Gill, P.M.W., Pople, J.A.: The performance of a family of density functional methods. *J. Chem. Phys.* 98, 5612 (1993), DOI: 10.1063/1.464906
9. Ochterski, J.W., Petersson, G.A., Montgomery Jr., J.A.: A complete basis set model chemistry. V. Extensions to six or more heavy atoms. *J. Chem. Phys.* 104, 2598–2619 (1996), DOI: 10.1063/1.470985
10. Montgomery Jr., J.A., Frisch, M.J., Ochterski, J.W., et al.: A complete basis set model chemistry. VII. Use of the minimum population localization method. *J. Chem. Phys.* 112, 6532–6542 (2000), DOI: 10.1063/1.481224
11. Curtiss, L.A., Redfern, P.C., Raghavachari, K.: Gaussian-4 theory. *J. Chem. Phys.* 126, 084108 (2007), DOI: 10.1063/1.2436888
12. Curtiss, L.A., Redfern, P.C., Raghavachari, K.: Gn theory. *WIREs Comput Mol Sci.* 1, 810–825 (2011), DOI: 10.1002/wcms.59
13. Nyden, M.R., Petersson, G.A.: Complete basis set correlation energies. I. The asymptotic convergence of pair natural orbital expansions. *J. Chem. Phys.* 75, 1843–1862 (1981), DOI: 10.1063/1.442208
14. Petersson, G.A., Bennett, A., Tensfeldt, T.G., et al.: A complete basis set model chemistry. I. The total energies of closed-shell atoms and hydrides of the first-row atoms. *J. Chem. Phys.* 89, 2193–2218 (1988), DOI: 10.1063/1.455064
15. Petersson, G.A., Al-Laham, M.A.: A complete basis set model chemistry. II. Open-shell systems and the total energies of the first-row atoms. *J. Chem. Phys.* 94, 6081–6090 (1991), DOI: 10.1063/1.460447
16. Petersson, G.A., Tensfeldt, T.G., Montgomery Jr., J.A.: A complete basis set model chemistry. III. The complete basis set-quadratic configuration interaction family of methods. *J. Chem. Phys.* 94, 6091–6101 (1991), DOI: 10.1063/1.460448
17. Montgomery Jr., J.A., Frisch, M.J., Ochterski, J.W., et al.: A complete basis set model chemistry. VI. Use of density functional geometries and frequencies. *J. Chem. Phys.* 110, 2822–2827 (1999), DOI: 10.1063/1.477924

18. Dunning Jr., T.H.: Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *J. Chem. Phys.* 90, 1007–1023 (1989), DOI: 10.1063/1.456153
19. Kendall, R.A., Dunning Jr., T.H., Harrison, R.J.: Electron affinities of the first-row atoms revisited. Systematic basis sets and wave functions. *J. Chem. Phys.* 96, 6796–6806 (1992), DOI: 10.1063/1.462569
20. Woon, D.E., Dunning Jr., T.H.: Gaussian-basis sets for use in correlated molecular calculations. 3. The atoms aluminum through argon. *J. Chem. Phys.* 98, 1358–1371 (1993), DOI: 10.1063/1.464303
21. Peterson, K.A., Woon, D.E., Dunning Jr., T.H.: Benchmark calculations with correlated molecular wave functions. IV. The classical barrier height of the  $H + H_2 \rightarrow H_2 + H$  reaction. *J. Chem. Phys.* 100, 7410–7415 (1994), DOI: 10.1063/1.466884
22. Wilson, A.K., van Mourik, T., Dunning Jr., T.H.: Gaussian Basis Sets for use in Correlated Molecular Calculations. VI. Sextuple zeta correlation consistent basis sets for boron through neon. *J. Mol. Struct. (Theochem)* 388, 339–349 (1996), DOI: 10.1016/S0166-1280(96)80048-0
23. Curtiss, L.A., Raghavachari, K., Redfern, P.C., et al.: Assessment of Gaussian-2 and density functional theories for the computation of enthalpies of formation. *J. Chem. Phys.* 106(3), 1063–1079 (1997), DOI: 10.1063/1.473182
24. NIST-JANAF Thermochemical tables. <https://janaf.nist.gov/>, accessed: 2020-10-30
25. Computational Chemistry Comparison and Benchmark DataBase. <https://cccbdb.nist.gov/hf0k.asp>, accessed: 2020-10-30
26. Efimov, A.I., Belorukova, L.P., Vasilkova, I.V., et al.: Properties of inorganic compounds. Reference book. Khimiya, Leningrad. 1983 (in Russian)
27. Gurvich, L.V. Bond Dissociation Energies. Nauka, Moscow. 1974 (in Russian)
28. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., et al.: Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community. *Supercomput. Front. Innov.* 6(2), 4–11 (2019), DOI: 10.14529/jsfi190201
29. Voevodin, V.V., Zhumatiy S.A., Sobolev, S.I., et al.: Practice of “Lomonosov” supercomputer. *Open systems* 7, 36–39 (2012). (in Russian)
30. Nikitenko, D.A., Voevodin, V.V., Zhumatiy, S.A.: Deep analysis of job state statistics on “Lomonosov-2” supercomputer. *Supercomput. Front. Innov.* 5(2), 4–10 (2019), DOI: 10.14529/jsfi180201