




# Multiple HPC Environments-Aware Container Image Configuration Workflow for Large-Scale All-to-All Protein–Protein Docking Calculations

Kento Aoyama<sup>1,2</sup>, Hiroki Watanabe<sup>1,2</sup>, Masahito Ohue<sup>1</sup> ,  
and Yutaka Akiyama<sup>1</sup>  

<sup>1</sup> Department of Computer Science, School of Computing,  
Tokyo Institute of Technology, Tokyo, Japan  
{aoyama,h\_watanabe}@bi.c.titech.ac.jp,  
{ohue,akiyama}@c.titech.ac.jp

<sup>2</sup> AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory,  
National Institute of Advanced Industrial Science and Technology,  
Tsukuba, Ibaraki, Japan

**Abstract.** Containers offer considerable portability advantages across different computing environments. These advantages can be realized by isolating processes from the host system whilst ensuring minimum performance overhead. Thus, use of containers is becoming popular in computational science. However, there exist drawbacks associated with container image configuration when operating with different specifications under varying HPC environments. Users need to possess sound knowledge of systems, container runtimes, container image formats, as well as library compatibilities in different HPC environments. The proposed study introduces an HPC container workflow that provides customized container image configurations based on the HPC container maker (HPCCM) framework pertaining to different HPC systems. This can be realized by considering differences between the container runtime, container image, and library compatibility between the host and inside of containers. The authors employed the proposed workflow in a high performance protein–protein docking application—MEGADOCK—that performs massively parallel all-to-all docking calculations using GPU, OpenMP, and MPI hybrid parallelization. The same was subsequently deployed in target HPC environments comprising different GPU devices and system interconnects. Results of the evaluation experiment performed in this study confirm that the parallel performance of the container application configured using the proposed workflow exceeded a strong-scaling value of 0.95 for half the computing nodes in the ABCI system (512 nodes with 2,048 NVIDIA V100 GPUs) and one-third those in the TSUBAME 3.0 system (180 nodes with 720 NVIDIA P100 GPUs).

**Keywords:** Containers · Container image configuration · Singularity · Bioinformatics · Message passing interface



# 1 Introduction

Containers that contribute to application portability through process isolation are now being widely used in computational applications. Today, many researchers run containers in various computing environments such as laptops, clouds, and supercomputers. Container technology is becoming essential for retaining scientific reproducibility and availability beyond system differences [1–3]. However, there remain certain limitations that need to be overcome to facilitate accurate configuration of container images for use in high-performance computing (HPC) applications running in multiple HPC environments. This requires users to understand systems, container runtimes, container image formats, and their compatibility with those used in HPC environments [4]. In addition, when an application establishes a message passing interface (MPI) communication over containers, the MPI library compatibility between the host system and the inside of the container must be ensured. This makes container deployment difficult. Therefore, these problems constitute a major obstacle facing the extensive use of the container technology in HPC environments.

To introduce the container’s techniques and benefits to one of our HPC applications, MEGADOCK [5], the authors, in this study, propose use of a custom HPC container image configuration workflow. The said workflow is based on the HPCCM framework [6] to give users easier way to make containers when considering the specification differences between the hosts and containers in multiple HPC environments. Furthermore, we also showed the performance results of the containers configured using the proposed workflow in the target HPC environments with a large-scale dataset for over a million protein–protein pairs of docking calculations.

Key contributions of this research are listed hereunder.

- A container image configuration workflow for an all-to-all protein–protein docking application (MEGADOCK) for HPC environments is proposed.
- The workflow provides functions to customize container image configurations by considering specification differences between target HPC environments using the HPCCM framework.
- It has been confirmed that the parallel performance of containers configured using the proposed workflow exceeds a strong-scaling value of 0.95. The container was run with more than 2,000 GPUs for docking calculations of over a million protein–protein pairs.

## 2 Background

### 2.1 Containers for HPC Environment

Docker [7] is the most widely used container in general computing environments. Its usage ranges from personal development to large-scale production systems in cloud environments. This has been actively developed and great efforts have been



made to standardize the container specification [8]. This, therefore, becomes a de-facto standard format of the containers.

However, in Docker’s toolset design, there are several concerns about the performance overhead, operational policies, and affinity for traditional HPC software stacks, particularly those related to system privileges [9]. Owing to such concerns in the HPC community, other container environments have been proposed for use in HPC environments. These include Singularity [10], Shifter [11], Chariecloud [12], and Sarus [13]. They are operated on HPC systems, and benchmark performances of HPC containers indicate that they perform nearly at par with the bare-metal environment [14–16]. Those container environments provide similar features, for example, they do not require privileges for users, thereby solving the security concerns of HPC system policies unlike the general Docker environment<sup>1</sup>. In addition, they also support the ‘pull’ function which downloads a container image from general container registry services (e.g. Docker Hub [17]) and convert it to their own container image format.

Presently, the most emerging container environment in the HPC field is Singularity [10], which was originally developed by the Lawrence Berkeley National Lab and subsequently moved to Sylabs Inc. It provides runtime support for host GPU/MPI libraries to use those from the inside of the containers to meet the requirements of HPC applications. It also provides original container building toolsets along with its own registry service. This helps users upload container images for improving the preservability and portability of the application [18]. These functions make it easy for users to use host GPU devices with GPU-enabled container images that are available on Docker Hub, Singularity Hub [19], and NVIDIA GPU Cloud (NGC) [20].

Consequently, the number of HPC systems that provide container environments is constantly increasing. This is due to the widespread use of Singularity and other containers; however, there remain several difficulties in the deployment of containers in HPC environments. Some of these difficulties are described in the next section.

## 2.2 Problems of Container Image Configuration Workflow

Figure 1 describes an example of a typical container deployment workflow for several environments, including HPC systems.

HPC container deployment workflows are generally expected to support both Docker and Singularity to keep application portability in a wide range of computing environments. However, supporting both container environments from the level of container image specification (recipe) requires efforts for its maintenance. To this end, Singularity provides functions to download a container image from general registry services, and this image can be subsequently converted to Singularity’s image format [10]. Therefore, it is possible to use various container images including Docker’s images and run them on HPC systems using Singu-

---

<sup>1</sup> The `rootless-mode` is available from Docker 19.03 (since July 2019).



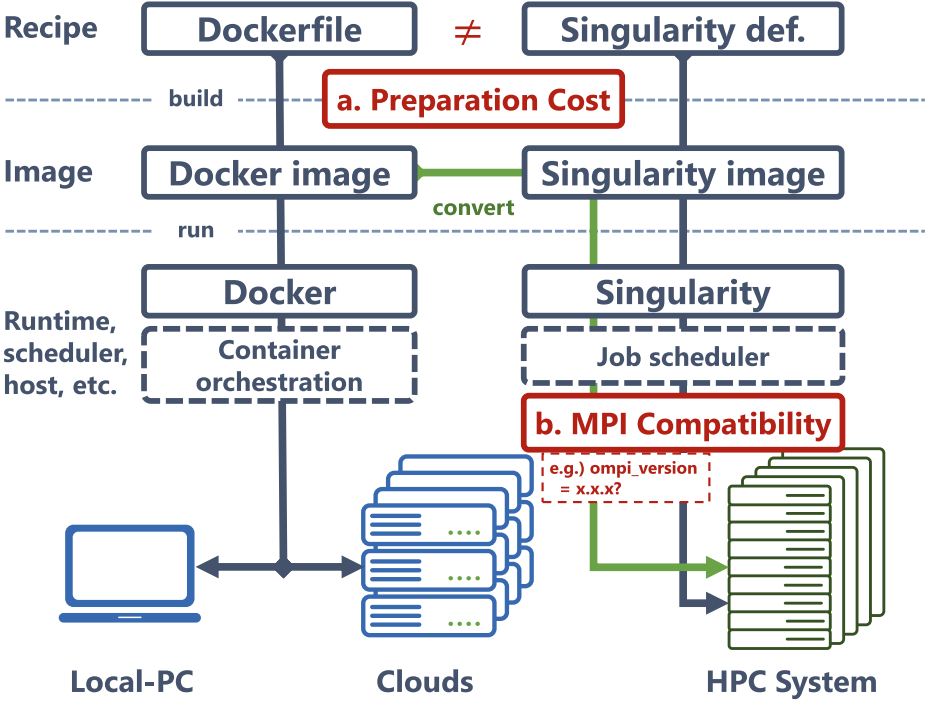


Fig. 1. Example of general container deployment workflow

larity. However, deployment of typical HPC applications nonetheless encounters several problems.

**A. Preparation Cost for Container Image Recipe with Host Dependent Library.** First, there exists a dependent library problem necessitating the availability of local libraries for using high-speed interconnects within target HPC systems. These must be installed within containers. For example, **openib** [25], **ucx** [26] or a similar library needs to be installed in the container if it is running on the system with InfiniBand [27]. On the other hand, the **psm2** [28] library is required when it runs on the system with Intel Omni-Path [29].

Technically, it is possible to install almost all of the libraries in one container; however, it is generally not recommended as a best practice for container image configuration. Because most of the advantages of the containers originated from its light-weightness, the containers must be as simple as possible.

**B. MPI Library Compatibility for Inter-containers Communication.** Second, if the process in a singularity container uses the MPI library to communicate with the process outside of the container, then the Application Binary Interface (ABI) must be compatible between MPI libraries of the host and con-



tainer. For instance, it is necessary to install the same (major and minor) version of the library when OpenMPI [30] older than version 3.0 is used [2].

The problem pertaining to ABI compatibility can be overcome by using latest releases of MPI libraries, such as MPICH [31] v3.1 (or newer) or IntelMPI [32] v5.0 (or newer) given that they officially support compatibility between different library versions. However, users must know what version of MPI libraries are supported in both host systems and container images. Deployment of containerized MPI applications to HPC systems nonetheless involves large expenditures.

The above-mentioned problems are major difficulties to be considered when configuring the container image for the HPC environments.

### 3 HPC Container Maker (HPCCM) Framework

To solve these difficulties and ease the configuration of container specifications, use of the HPC Container Maker (HPCCM) framework was proposed by the NVIDIA corporation [6]. HPCCM is an open source tool to ease generation of container specification files for HPC environments. HPCCM supports both the Docker and Singularity specification formats via use of a highly functional Python recipe. This provides various useful functions to configure container images along with their application and system dependencies.

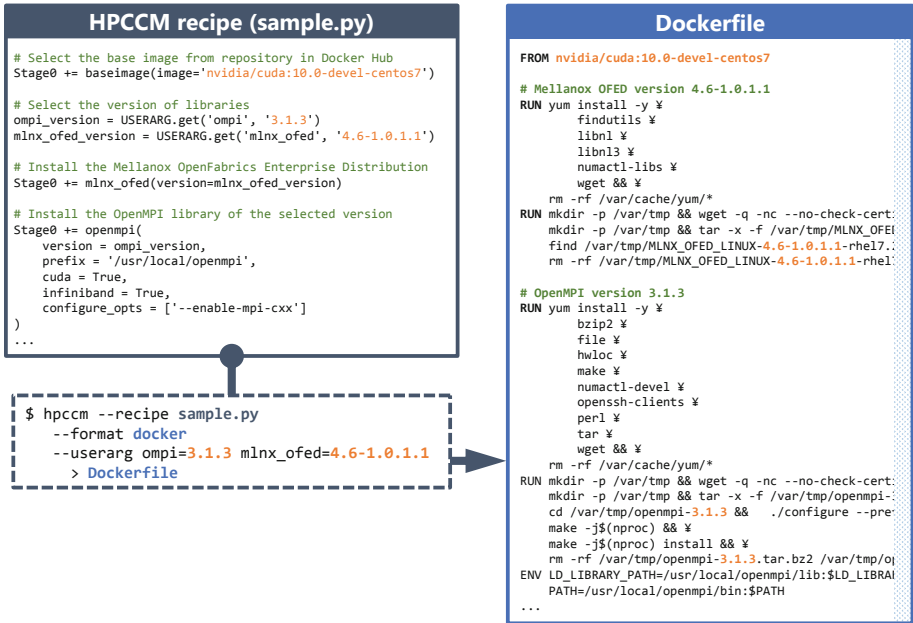


Fig. 2. Sample of HPCCM recipe and generated container specification (Dockerfile)



Figure 2 shows a sample Python recipe of HPCCM and a generated container specification in the ‘Dockerfile’ format. HPCCM contains the ‘building blocks’ feature, which transparently provides simple descriptions to install the specific components commonly used in the HPC community. Additionally, it supports flexible Python-based code generation functions, including recipe branch and validating user arguments; thus, it provides users with an easy method to generate the multiple container specifications from the same Python recipe file.

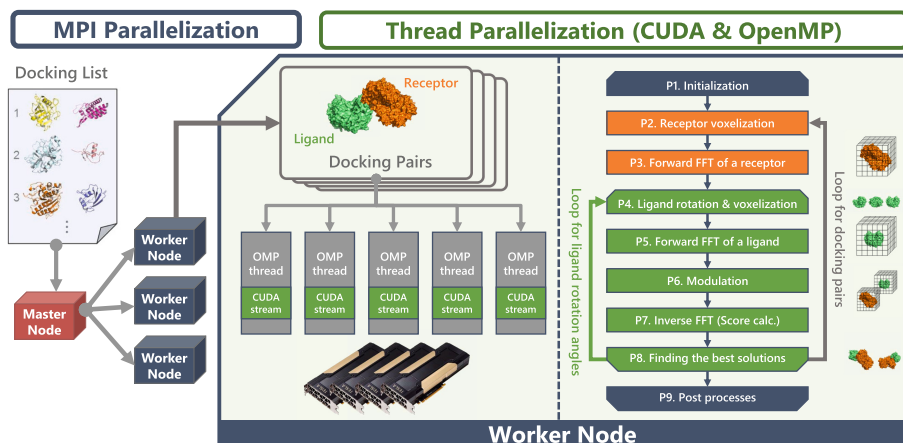
By adopting the HPCCM framework, the cost of container recipe preparation can be reduced by implementing one Python recipe and setting parameters of container specifications for HPC environments.

The authors used this HPCCM framework as a base for the proposed container deployment workflow for target HPC environments. The following section provides an overview of the target application and proposed workflow.

## 4 Container Deployment Workflow for MEGADOCK Application Using HPC Container Maker

### 4.1 MEGADOCK: A High Performance All-to-All Protein–Protein Docking Application

The authors selected MEGADOCK [5] as the proposed container configuration workflow application. MEGADOCK is an all-to-all protein–protein docking application written in C++/CUDA for use in large-scale computing environments. The internal process is based on Fast Fourier Transform (FFT) calculations for grid-based protein–protein docking using FFT libraries (e.g. FFTW [22], CUFFT [24]).



**Fig. 3.** Overview of docking calculations in MEGADOCK 5.0 (under development) and its OpenMP/GPU/MPI hybrid parallelization



In the latest implementation of MEGADOCK 5.0, which is under development, each docking pair calculation is independently assigned to an OpenMP [23] thread with CUDA streams [24]. The set of docking pairs is distributed by the master to workers in a typical master-worker framework implemented in C++ using the MPI library (Fig. 3).

At present, the authors are working toward improving the performance of the application as well as container portability in multiple environments while upgrading to the next MEGADOCK version. Currently, Docker images and their container specifications in the ‘Dockerfile’ format for GPU-enabled environments are provided to users having access to the MEGADOCK public repository on GitHub [33]. The authors reported scalable performance when operating those containers in a cloud environment using Microsoft Azure [34].

However, it is required to solve several container configuration difficulties when we assume the MEGADOCK application with Singularity containers on different HPC systems that are presented in previous sections. Therefore, the authors, in this study, propose use of an HPC container deployment workflow using the HPCCM framework. The said workflow supports a wide variety of computing environments and solves deployment problems in HPC systems for further advancement in this project.

## 4.2 HPC Container Workflow for MEGADOCK with HPCCM

Figure 4 provides an overview of the proposed container configuration workflow for deploying MEGADOCK in different HPC environments while using the HPCCM framework. Introducing the HPCCM framework in combination with the MEGADOCK application workflow offers the following advantages.

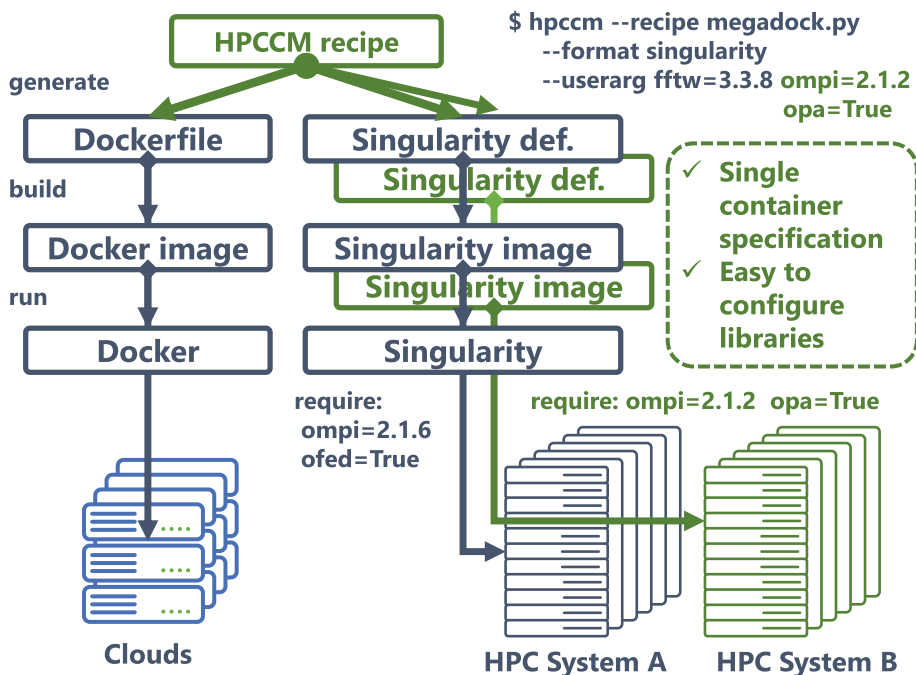
### 1. Decreasing the cost of preparing container images

The workflow based on the HPCCM framework supports the configuration of container specifications in different environments by setting appropriate parameter values. Additionally, it supports both Docker and Singularity specification formats. This results in the reduction of management costs for container specification files, thereby facilitating continuous integration (CI) of container workflow.

### 2. Avoiding library compatibility problems

The workflow provides a clear opportunity to specify the versions of dependent libraries by setting parameter values when container specifications are generated. Explicit and easy specifications of library versions help in overcoming problems associated with library compatibility. This is particularly true in cases where the exact version of the MPI libraries pertaining to the host HPC system and the inside of the container must match to avoid ABI compatibility issues.





**Fig. 4.** Proposed HPC container deployment workflow for different HPC environments

### 4.3 Example of User Workflow

First, a user generates a custom container specification for both the target system and container environment by setting parameter values. Subsequently, the user builds a custom container image by using the container specification file in local environment (e.g. laptop, general cluster, etc.).<sup>2</sup>

Next, the user deploys custom containers to the target system for running the MEGADOCK application. Here, a user selects a compatible host MPI module and loads it while launching containers. The said containers can then communicate with processes over Singularity containers. Finally, custom containers pertaining to the MEGADOCK application distribute docking tasks via MPI communication in the target HPC system.

## 5 Evaluation Experiments

In this section, we evaluate the parallel performance of the custom containers in the target HPC environments. Container images were configured based

<sup>2</sup> This process can be skipped if there already exists a custom container image prepared for the target environment.



on the workflow proposed in the previous section. Additionally, we conducted a large-scale experiment involving over a million protein–protein pair docking calculations requiring a large number of computing nodes of the target HPC environment.

Target HPC environments used in both experiments correspond to ABCI (Table 1), located at the National Institute of Advanced Industrial Science and Technology, Japan, and TSUBAME 3.0 (Table 2), located at the Tokyo Institute of Technology, Japan. Both these environments provide Singularity environments and each computing node equips NVIDIA GPU devices; however, the systems have different hardware and software specifications.

**Table 1.** ABCI system hardware specifications

Item	Description	#
CPU	Intel Xeon Gold 6148, 2.4 [GHz]	×2
GPU	NVIDIA Tesla V100 for NVLink	×4
Memory	384 [GB]	
Local storage	NVMe SSD, 1.6 [TB]	×1
Interconnect	InfiniBand EDR, 100 [Gbps]	×2
Total number of computing nodes		×1,088

**Table 2.** TSUBAME 3.0 system hardware specifications

Item	Description	#
CPU	Intel Xeon E5-2680 v4, 2.4 [GHz]	×2
GPU	NVIDIA Tesla P100 for NVLink	×4
Memory	256 [GB]	
Local storage	NVMe SSD, 2.0 [TB]	×1
Interconnect	Intel Omni-Path HFI, 100 [Gbps]	×4
Total number of computing nodes		×540

### 5.1 Experiment 1. Container Deployment for Target HPC Environment

At first, we prepared custom container images for target environments and tested their execution performance using a small number of computing nodes with a benchmark dataset. The experiment aimed at validating the proposed workflow and ensuring that the custom container functions properly in the target environment.



**System and Container Specifications.** Specifications of the system software and container images used during experimentation are listed in Table 3.

Custom container images were prepared to those that are properly configured with the GPU/MPI libraries so they are compatible with the system modules [21] provided by the host (Table 3). The NVIDIA container image obtained from the Docker Hub (`nvidia/cuda:10.0-devel-centos7`) was selected as a base image because CUDA-10.0 [24] supports both GPU architectures in the target environments.<sup>3</sup>

Additionally, we installed each version of the OpenMPI [30] library by using different parameters to match the version of the host system module. The dependent libraries for the InfiniBand EDR [27] and the Intel Omni-Path HFI [29] were installed when necessary.

**Table 3.** Specifications of system software and container images used in Experiment 1

	ABCI	TSUBAME 3.0
<i>System software specification</i>		
OS	CentOS 7.5.1804	SUSE Linux Enterprise Server 12 SP2
Linux kernel	3.10.0	4.4.121
Singularity [10]	singularity/2.6.1	singularity/3.2.1
CUDA [24]	cuda/10.0/10.0.130	cuda/8.0.61
OpenMPI [30]	openmpi/2.1.6	openmpi/2.1.2-opa10.9
<i>Container image specification</i>		
Base image	nvidia/cuda:10.0-devel-centos7	nvidia/cuda:10.0-devel-centos7
FFTW [22]	fftw-3.3.8	fftw-3.3.8
CUDA [24]	cuda-10.0.130	cuda-10.0.130
OpenMPI [30]	openmpi-2.1.6	openmpi-2.1.2

**Dataset.** The dataset used during the experiment corresponds to the ZLab Docking Benchmark 5.0 [35]. We selected 230 files of the PDB (protein 3-D coordinates) format data labeled `unbound`. This was calculated for the protein–protein docking of the all-to-all ( $230 \times 230 = 52,900$ ) pairs.

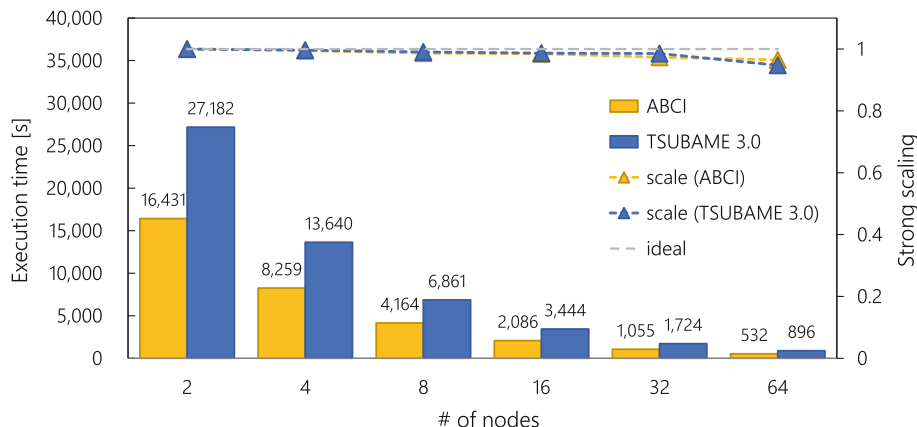
**Computational Details.** The input files are stored in a virtually distributed shared file system, called BeeGFS On Demand (BeeOND) [36], which is temporarily constructed on the set of non-volatile memory express (NVMe) storages in computing nodes. The output files are generated for each local NVMe storage

<sup>3</sup> The version of loaded CUDA modules were different in each environment; however, we confirmed that they did not exhibit any significant performance differences.



upon completion of each protein–protein pair docking calculation. When all calculations are completed, the output files are compressed as a `.tar` archive and moved to the global storage.

The measured execution time is obtained using the task distribution framework in MEGADOCK. This indicates the duration time from the start of task processing to the end of all tasks. The data point in the plot implies that each execution time is chosen from a median of three executions for the same calculations.



**Fig. 5.** Performance results of MEGADOCK docking calculations performed on ZLab Docking Benchmark 5.0 (all-to-all, 52,900 pairs) dataset in ABCI and TSUBAME 3.0 environments.

**Results.** Figure 5 depicts the performance results of the docking calculations using the benchmark dataset in both target environments. In all the docking calculations, no fatal errors were detected. The demonstration of the proposed custom container image configuration workflow was considered successful.

On average, the execution of the docking calculations in the ABCI environment was faster in comparison with that in TSUBAME 3.0 by 1.65 times at each point. The parallel performance in strong-scaling was found to be 0.964 on ABCI and 0.948 on TSUBAME 3.0 in the comparison of the execution time when running on 2 nodes versus 64 nodes. There are no significant differences between the environments in terms of scalability because the dataset used for this experiment was not sufficiently large.

The results obtained in the ABCI environment, which had four NVIDIA Tesla V100 devices, demonstrated better performance in comparison with TSUBAME 3.0 that comprised four NVIDIA Tesla P100 devices. This indicates that the performance of FFT-based docking calculations in MEGADOCK are computationally expensive, which heavily depends on the performance of the CUFFT library



with the NVIDIA GPU, and therefore, the performance is directly affected by the host GPU device architecture.

## 5.2 Experiment 2. Performance Evaluation with Large-Scale Computing Nodes and over a Million Protein–Protein Pairs

Next, we performed a large-scale experiment using a larger number of computing nodes and over a million protein–protein pairs of docking calculations. To understand the principles of biological systems and elucidate the causes of diseases, over a million all-to-all protein pairs of docking calculations were considered in this experiment.

We reserved and used half of the computing nodes of the ABCI system (512 nodes with 2,048 GPUs) and one-third of the TSUBAME 3.0 system (180 nodes with 720 GPUs) for this experiment. The computational resources for calculations were supported by the “Grand Challenge” programs, which are open recruitment programs for researchers, coordinated by AIST and Tokyo Tech, respectively.

**System and Container Specifications.** Environmental specifications of the system hardware were identical to that described for the first experiment. Additionally, system software and container images were nearly identical to those corresponding to the first experiment. Several versions of libraries were modified, but no significant performance impact was observed.

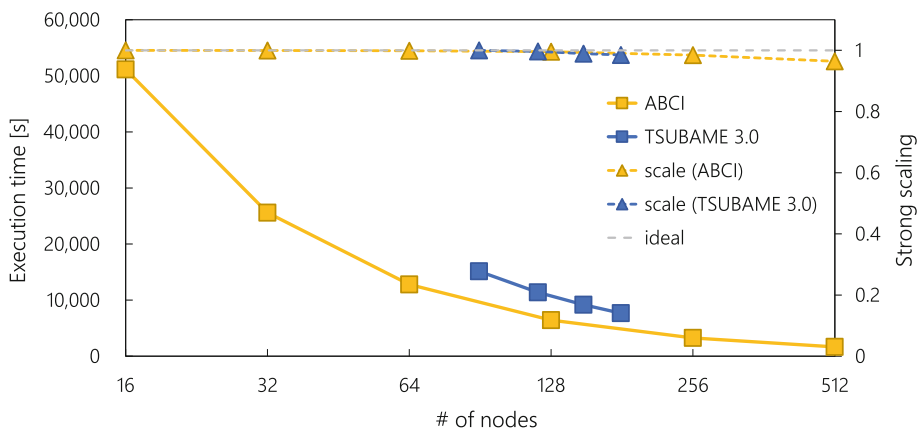
**Dataset.** We used the dataset from the ZLab Benchmark 5.0 [35], which is the same as in the first experiment. To validate the large-scale application performance, we simply amplified the set of docking pairs to 25 times larger than the whole of the original dataset and created a virtual large-scale benchmark dataset. This dataset includes duplicated protein pairs; however, the docking calculations in the MEGADOCK application are completely independent of each other. Therefore, we computed 1,322,500 pairs of protein–protein docking calculations in total.

**Computational Details.** The application deployments, storage usage, and measurement methods are the same as in the first experiment.

As for the number of computing nodes used in each environment, we selected 16, 32, 64, 128, 256, and 512 nodes in the ABCI environment, and 90, 120, 150, and 180 nodes in TSUBAME 3.0. These node counts were set considering the limitation of reserved computational resources as well as performance predictions obtained from the previous experiment.

**Results.** Figure 6 depicts performance results obtained by performing large-scale docking calculations in the ABCI and TSUBAME 3.0 systems. The scale of computational resources and dataset size used in this experiment were larger





**Fig. 6.** Performance results of the MEGADOCK docking calculations with 1,322,500 pairs of proteins on ABCI and TSUBAME 3.0.

compared to the previous experiment; however, the parallel performance in both environments was observed to be similar.

The observed execution time equaled 1,657s when using half the ABCI system (512 nodes with 2,048 NVIDIA V100 GPUs) and 7,682s when using one-third of the TSUBAME 3.0 system (180 nodes with 720 NVIDIA P100 GPUs). A direct comparison of the performance in each environment is not warranted owing to differences between measured data points and computational resources. However, ABCI clearly demonstrated better overall performance.

The parallel performance in strong-scaling was found to be 0.964 on ABCI and 0.985 on TSUBAME 3.0 in the comparison of the execution time when running on each minimum-measured and maximum-measured number of computing nodes. This indicated that our container application workflow is able to achieve good scalability on the target HPC environments.

The older version of MEGADOCK required approximately half a day to run a million protein-protein pairs of docking calculations using the entire TSUBAME 2.5 system [5]. However, the latest MEGADOCK version completes over a million protein-protein docking-pair calculations within 30 min in the latest HPC environment.

## 6 Discussion

The proposed workflow considers ABCI and TSUBAME 3.0 as target HPC environments when deploying Singularity containers because they adopt similar architectural concepts but different specifications pertaining to both hardware and software. Thus, the environments are sufficient as targets for a proof-of-concept of our workflow.

Further, we can easily switch specific dependent libraries in each environment using the proposed workflow to fill gaps caused by differences in specifica-



tions. However, the proposed workflow does not cover other gaps, such as those pertaining to binary optimization of CPU/GPU architectural differences, MPI communication optimization for network architecture, and other performance optimization approaches. These features must be included in future implementations to enhance the utility of the proposed workflow.

## 7 Conclusion

In this study, the authors incorporated the HPCCM framework into a large-scale all-to-all protein–protein docking application called MEGADOCK to integrate the container deployment workflow over multiple HPC systems with different specifications. The proposed workflow provides users an easy means to configure containers for different systems and offers the flexibility to operate on both Docker and Singularity container formats. This helps users avoid container difficulties within HPC systems, such as host-dependent libraries and ABI compatibility of MPI libraries.

Further, we evaluated the parallel performance of container execution in both ABCI and TSUBAME 3.0 systems using a small benchmark dataset and a virtual large-scale datasets containing over a million protein–protein pairs. Result demonstrate that the parallel performance achieved exceeds a strong-scaling value of 0.95 when using half the ABCI system (512 nodes with 2,048 GPUs) and one-third of the TSUBAME 3.0 system (180 nodes with 720 GPUs). This demonstrates that the latest HPC environment can complete over a million protein–protein docking calculations within half an hour.

The authors believe that performance results obtained in this study can contribute to accelerate exhaustive large-scale ‘interactome’ analysis for understanding principles of biological systems. Additionally, the authors believe the proposed workflow would be beneficial for contributing to the portability of scientific achievements.

**Code Availability.** The entire source code of proposed container workflow and manual instructions are available in the following GitHub repository.

[https://github.com/akiyamalab/megadock\\_hpccm](https://github.com/akiyamalab/megadock_hpccm)

**Acknowledgments.** Computational resources of the AI Bridging Cloud Infrastructure (ABCI) were awarded by the ABCI Grand Challenge Program, National Institute of Advanced Industrial Science and Technology (AIST), and resource of the TSUBAME 3.0 was awarded by the TSUBAME Grand Challenge Program, Tokyo Institute of Technology.

This work was partially supported by KAKENHI (Grant No. 17H01814 and 18K18149) from the Japan Society for the Promotion of Science (JSPS), the Program for Building Regional Innovation Ecosystems “Program to Industrialize an Innovative Middle Molecule Drug Discovery Flow through Fusion of Computational Drug Design and Chemical Synthesis Technology” from the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT), the Research Complex Program “Wellbeing Research Campus: Creating new values through technological and



social innovation” from Japan Science and Technology Agency (JST), and conducted as research activities of AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL).

## References

1. Zhang, J., Lu, X., Panda, D.K.: Is singularity-based container technology ready for running MPI applications on HPC clouds? In: Proceedings of the 10th International Conference on Utility and Cloud Computing (UCC 2017), Austin, TX, USA, pp. 151–160. ACM (2017). <https://doi.org/10.1145/3147213.3147231>
2. Veiga, V.S., et al.: Evaluation and benchmarking of Singularity MPI containers on EU research e-infrastructure. In: Proceedings of the 1st International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE HPC), Denver, CO, USA, pp. 1–10. IEEE TCHPC (2019). <https://doi.org/10.1109/CANOPIE-HPC49598.2019.00006>
3. Paolo, D.T., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M.L., Notredame, C.: The impact of Docker containers on the performance of genomic pipelines. *PeerJ* **3**(3), e1273 (2015). <https://doi.org/10.7717/peerj.1273>
4. Canon, R.S., Young, A.J.: A case for portability and reproducibility of HPC containers. In: Proceedings of the 1st International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE HPC), Denver, CO, USA, pp. 49–54. IEEE TCHPC (2019). <https://doi.org/10.1109/CANOPIE-HPC49598.2019.00012>
5. Ohue, M., Shimoda, T., Suzuki, S., Matsuzaki, Y., Ishida, T., Akiyama, Y.: MEGADOCK 4.0: an ultra-high-performance protein-protein docking software for heterogeneous supercomputers. *Bioinformatics* **30**(22), 3281–3283 (2014). <https://doi.org/10.1093/bioinformatics/btu532>
6. McMillan, S.: Making containers easier with HPC container maker. In: Proceedings of the SIGHP Systems Professionals Workshop (HPCSYSPROS 2018), Dallas, TX, USA (2018). <https://doi.org/10.5281/zenodo.3552972>
7. Docker. <https://www.docker.com/>. Accessed 9 Dec 2019
8. Open Container Initiative. <https://www.opencontainers.org/>. Accessed 9 Dec 2019
9. Jacobsen, D.M., Canon, R.S.: Contain this, unleashing Docker for HPC. In: Proceedings of the Cray User Group (2015)
10. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: scientific containers for mobility of compute. *PLoS ONE* **12**(5), 1–20 (2017). <https://doi.org/10.1371/journal.pone.0177459>
11. Gerhardt, L., et al.: Shifter: containers for HPC. *J. Phys. Conf. Ser.* **898**(082021) (2017). <https://doi.org/10.1088/1742-6596/898/8/082021>
12. Priedhorsky, R., Randles, T.: Charliecloud: unprivileged containers for user-defined software stacks in HPC. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2017), Denver, CO, USA, no. 36, pp. 1–10. ACM (2017). <https://doi.org/10.1145/3126908.3126925>
13. Benedicic, L., Cruz, F.A., Madonna, A., Mariotti, K.: Sarus: highly scalable Docker containers for HPC systems. In: Weiland, M., Juckeland, G., Alam, S., Jagode, H. (eds.) *ISC High Performance 2019*. LNCS, vol. 11887, pp. 46–60. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-34356-9\\_5](https://doi.org/10.1007/978-3-030-34356-9_5)



14. Torrez, A., Randles, T., Priedhorsky, R.: HPC container runtimes have minimal or no performance impact. In: Proceedings of the 1st International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE HPC), Denver, CO, USA, pp. 37–42. IEEE TCHPC (2019). <https://doi.org/10.1109/CANOPIE-HPC49598.2019.00010>
15. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: Proceedings of 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2015), Philadelphia, PA, USA, pp. 171–172 (2015). <https://doi.org/10.1109/ISPASS.2015.7095802>
16. Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A.F.: Performance evaluation of container-based virtualization for high performance computing environments. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, pp. 233–240. IEEE (2013). <https://doi.org/10.1109/PDP.2013.41>
17. Docker Hub. <https://hub.docker.com/>. Accessed 9 Dec 2019
18. Sochat, V.: Singularity registry: open source registry for Singularity images. J. Open Source Softw. **2**(18), 426 (2017). <https://doi.org/10.21105/joss.00426>
19. Sochat, V., Prybol, C.J., Kurtzer, G.M.: Enhancing reproducibility in scientific computing: metrics and registry for singularity containers. PLoS ONE **12**(11), 1–24 (2017). <https://doi.org/10.1371/journal.pone.0188511>
20. NGC - GPU-Optimized Software Hub Simplifying DL, ML and HPC workflows. <https://www.nvidia.com/en-us/gpu-cloud/>. Accessed 9 Dec 2019
21. Furlani, J.L., Osel, P.W.: Abstract yourself with modules. In: Proceedings of the Tenth Large Installation Systems Administration Conference (LISA 1996), Chicago, IL, USA, pp. 193–204 (1996)
22. Matteo, F., Steven, G.J.: The design and implementation of FFTW3. Proc. IEEE **93**(2), 216–231 (2005). <https://doi.org/10.1109/JPROC.2004.840301>
23. Leonardo, D., Ramesh, M.: OpenMP: an industry standard API for shared-memory programming. Comput. Sci. Eng. **5**(1), 46–55 (1998)
24. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue GPU Comput. **6**(2), 40–53 (2008). <https://doi.org/10.1145/1401132.1401152>
25. OpenFabrics Alliance. <https://www.openfabrics.org/>. Accessed 11 Dec 2019
26. Unified Communication X. <https://www.openucx.org/>. Accessed 11 Dec 2019
27. InfiniBand Architecture Specification, Release 1.3.1. <https://cw.infinibandta.org/document/dl/8125>. Accessed 11 Dec 2019
28. intel/opa-psm2. <https://github.com/intel/opa-psm2>. Accessed 11 Dec 2019
29. Birrittella, M.S., et al.: Intel Omni-Path architecture: enabling scalable, high performance fabrics. In: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, Santa Clara, CA, USA, pp. 1–9. IEEE (2015). <https://doi.org/10.1109/HOTI.2015.22>
30. Gabriel, E., et al.: Open MPI: goals, concept, and design of a next generation MPI implementation. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30218-6\\_19](https://doi.org/10.1007/978-3-540-30218-6_19)
31. MPICH. <https://www.mpich.org/>. Accessed 11 Dec 2019
32. Intel MPI Library. <https://software.intel.com/mpi-library>. Accessed 11 Dec 2019
33. akiyamalab/MEGADOCK. <https://github.com/akiyamalab/MEGADOCK>. Accessed 11 Dec 2019



34. Aoyama, K., Yamamoto, Y., Ohue, M., Akiyama, Y.: Performance evaluation of MEGADOCK protein-protein interaction prediction system implemented with distributed containers on a cloud computing environment. In: Proceedings of the 25th International Conference on Parallel and Distributed Processing Techniques and Application (PDPTA 2019), Las Vegas, NV, pp. 175–181 (2019)
35. Vreven, T., et al.: Updates to the integrated protein-protein interaction benchmarks: docking benchmark version 5 and affinity benchmark version 2. *J. Mol. Biol.* **427**(19), 3031–3041 (2015). <https://doi.org/10.1016/j.jmb.2015.07.016>
36. BeeGFS. <https://www.beegfs.io/>. Accessed 9 Dec 2019

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

