Portability of Multiphysics Applications on Heterogeneous Modular Supercomputers

Daniel Caviedes-Voullième[1], Seong-Ryong Koh[1], Stefan Poll[1], Estela Suarez[1,5], Takashi Arakawa[2,3], Kengo Nakajima[3,4(✉)], and Shinji Sumimoto[3]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
{d.caviedes.voullieme,s.koh,s.poll,e.suarez}@fz-juelich.de
[2] CliMTech Inc., Chiba, Japan
arakawa@climtech.jp
[3] The University of Tokyo, Kashiwa, Chiba, Japan
{nakajima,sumimoto}@cc.u-tokyo.ac.jp
[4] RIKEN Center for Computational Science (R-CCS), Kobe, Japan
[5] Department of Computer Science, University of Bonn, Bonn, Germany

Abstract. The Modular Supercomputer Architecture (MSA) integrates various processing units, with compute modules tailored for specific algorithms, forming a unified heterogeneous system. Modules operate as parallel clustered systems, interconnected via a common or federated network. Optimized resource management allows flexible node selection, aiming for better performance. Hardware heterogeneity poses challenges for developers, who must port and optimize codes for various configurations. This study evaluated the performance of two applications-TSMP (regional Earth-system simulation) and mAIA (fluid dynamics)-across two MSA platforms (DEEP, Wisteria/BDEC-01), considering both hardware and software configurations.

Keywords: MSA · Heterogeneous Systems · Coupling Library

1 Heterogeneous Computing and MSA

Post-excale systems will be much larger than current ones. Semiconductor performance scaling and increasing CPU operating clock frequency will be limited. Rising demands for computer power are challenged by electricity costs.

High power consumption of CPUs makes large-scale homogeneous CPU clusters costly. Accelerator processing units (APUs) execute specific operations efficiently, offering high performance and lower energy consumption, making them popular in HPC. Recently, GPUs have been widely used, along with new custom accelerators for AI applications [1].

Therefore, the next flagship-level Supercomputer will be a heterogeneous computer with several different hardware configurations for each target application area optimized.

R. Wyrzykowski et al. (Eds.): PPAM 2024, LNCS 15581, pp. 47–62, 2025.
https://doi.org/10.1007/978-3-031-85703-4_4

As the number of systems equipped with GPUs, which have high power-to-performance ratios, increases, the number of GPU-enabled applications and applications using machine learning is also increasing. Recent systems run a variety of applications such as simulations in computational science and engineering, big data analytics, AI, machine learning, etc.

However, not all HPC applications benefit from accelerators, so CPUs remain essential for many users. Thus, HPC systems need heterogeneous designs combining diverse processing units.

The *Modular Supercomputer Architecture* (MSA) is a system design that aims at combining different processing units at system level [2]. Various compute modules, each tailored to suit specific classes of algorithms, are interconnected to form a unified heterogeneous system. Each module operates as a parallel, clustered system of considerable scale. The modules may all be attached to a common interconnecting network, or a federated network can be used to link the module-specific interconnects. On the system software side, an optimized resource manager facilitates the assembly of diverse resource combinations based on the workload requirements of each application. In this way, each user has full freedom in selecting the types and amount of nodes that the application needs. This aims to yield better application performance by leveraging near-optimal resource combinations, enabled by performance-portable implementations in the applications.

The complexity added by hardware heterogeneity poses challenges to application developers, who need to port and optimise their codes to an increasing variety of system configurations and software environments. The community's willingness to port large, complex code to new system architectures depends largely on whether there are enough success stories. It is therefore crucial to carefully analyse the effort required and the performance achieved by those application developers that have already attempted to port their code to new system designs. Equally important is to compare performances across platforms, and to report those in an fair and unbiased manner.

The present work aims at doing both in the context of modular supercomputing: describing the performance obtained by two application use-cases – the TSMP for regional earth-system simulation and the mAIA for fluid dynamics – running on two completely different MSA platforms both in terms of their hardware and software configuration.

Our main new contributions are:

- Porting the applications (TSMP and mAIA) to the DEEP and Wisteria/BDEC-01 modular platforms.
- Deploying the h3-Open-BDEC on the DEEP system
- Reporting on the performance of two applications on the DEEP and Wisteria/BDEC-01 systems, studying separately the contribution from the hardware configuration and the coupling library.

The paper is structured as follows. Section 2 shortly describes the two MSA platforms used in the study and the main elements of their software stack. The key characteristics of the application use cases are given in Sect. 3, with Sect. 4

detailing the modifications included on the original codes to port them to the new architecture. The results of the evaluation are presented in Sect. 5. The paper is wrapped up with a discussion of the related work (Sect. 6) and a summary of the concluding remarks in Sect. 7.

2 Target Systems and Software

The experiments described in Sect. 5 were executed on two modular supercomputers, the DEEP system at the Jülich Supercomputing Centre (JSC) in Germany, and the Wisteria/BDEC-01 system at the University of Tokyo in Japan. Both systems and related software are described in the following subsections.

2.1 DEEP

DEEP Hardware: The DEEP system is a Modular Supercomputer prototype built by Megware and deployed at JSC within the research project *DEEP-EST*. It is made of several modules, from which only two have been used in this work: the general purpose CPU *cluster*, and the GPU-accelerated *booster*. We focus therefore the system description on those, summarizing their main hardware characteristics in Table 1. The DEEP cluster module, composed of 50 nodes with Intel Xeon Skylake Gold CPUs, prioritizes reliable performance and versatile applicability, to handle HPC workloads (or parts thereof) with intricate and dynamic control flow. Emphasizing general-purpose performance and energy efficiency, this module is engineered to meet diverse computational demands. In contrast, the DEEP booster caters to highly scalable applications or workload segments. With 75 nodes, each housing an Intel Xeon Cascade Lake Silver CPU and an NVIDIA Tesla V100 GPU, the booster offers scalability and computational throughput, ideal for tasks with extensive parallelism and suitable control structures. Energy efficiency is a primary focus of the booster design, achieved through GPU-centric computing where the majority of compute operations are handled by the high-end GPU, while the Xeon CPU manages I/O, network communication (including MPI), and GPU device control via a PCIe Gen 3 link with 16 lanes. Both cluster and booster nodes are connected via an InfiniBand EDR fabric with 100 Gbit/s bandwidth.

DEEP Software: The DEEP software stack is designed to fit the requirements of a diverse portfolio of HPC applications and allow them to map their intrinsic scalability patterns onto the hardware: highly parallel code segments execute on the booster, while less scalable parts benefit from the cluster's high single-thread performance.

The DEEP programming environment abstracts the hardware complexity of MSA by dealing with it in the lower layers of the software stack, while offering user-friendly interfaces and standard parallel programming paradigms. For example, low-level interconnect management features were developed and integrated within MPI, but without changing the MPI calls that are directly used

Table 1. Key hardware features of the DEEP and Wisteria/BDEC-01 modules used in this work.

	DEEP system		Wisteria/BDEC-01	
	Cluster	Booster	Odyssey	Aquarius
Deployment	2019	2020	2021	2021
Node count	50	75	7,680	45
CPU type	2× Intel Xeon 6146	1× Intel Xeon 4215	1× Fujitsu A64FX	2× Intel Xeon 8360Y
CPU codename	Skylake	Cascade Lake	-	Ice Lake
Cores @freq.	2×12 @3.2 GHz	1×8 @ 2.5 GHz	1×48 @ 2.2 GHz	2×36 @ 2.4 GHz
GPUs per node	n.a.	1× Nvidia V100	n.a.	8× Nvidia A100
DDR4	192 GB	48 GB	n.a.	512 GB
HBM	n.a.	32 GB (GPU)	32 GB.	40 GB (GPU)
Mem.BW/node	256 GB/s	900 GB/s (GPU)	1,024 GB/s	1,555 GB/s (GPU)
HD	-	-	25.8 PB	25.8 PB
NVMe SSD	25.6 PB	38.4 PB	1.0 PB	+ 1.0 PB
Interconnect	EDR-IB (100 Gb/s)	EDR-IB (100 Gb/s)	Tofu Interconnect-D	HDR-IB (200 Gb/s) (4×)
Topology	fat-tree	tree	6D torus	fat tree
Power per node	500 W	500 W	200 W	2,900 W
Cooling	warm-water	warm-water	cold water	cold water

by the application developers. The DEEP system relies particularly on ParaStation MPI [3], which can manage communication across different modules even across gateway nodes in system configurations where different interconnect technologies are used (not the case on DEEP). ParaStation MPI is also aware of the particular network topology in MSA and uses this knowledge to optimise collective MPI operations on modular systems. Features as CUDA awareness enhance productivity and performance of hybrid MPI codes running on the booster.

The scheduler software used in DEEP is Slurm. While traditional job schedulers excel in optimizing monolithic supercomputers, the MSA demands capabilities to manage diverse resources and enable co-scheduling across modules. Extensions to packages like psslurm (part of ParaStation) and Slurm facilitate optimal utilization of heterogeneous resources, supporting co-scheduling across modules. Enhanced support for the Multiple-Program Multiple-Data (MPMD) paradigm enables heterogeneous jobs with different executables on distinct job allocations. In practical use cases, such as preprocessing data before simulation, data reduction, and final result visualization, executing codes on different modules involves simply specifying the nodes for each step to the Slurm scheduler.

2.2 Wisteria/BDEC-01 and h3-Open-BDEC

Wisteria/BDEC-01 (Hardware). The *Wisteria/BDEC-01* system (Fig. 1), which began operations at the Information Technology Center, the University of Tokyo (ITC/UTokyo) in May 2021, is the first system based on the concept of BDEC (Big Data & Extreme Computing) aiming to integrate *Simulation, Data, and Learning (S+D+L)*. The Simulation Node Group (*Odyssey*) consists

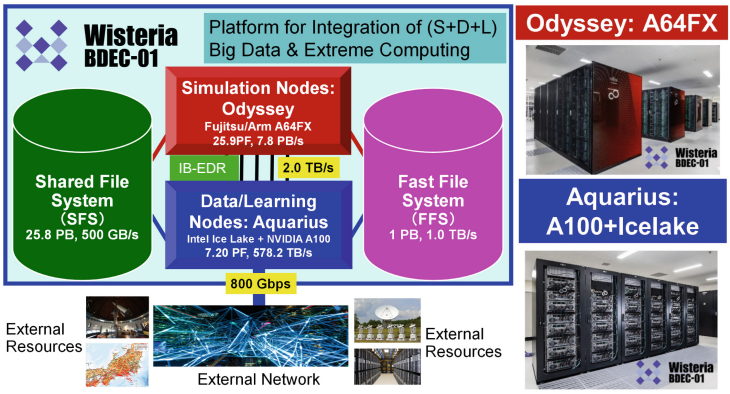

Fig. 1. Wisteria/BDEC-01 [4]

of 20 racks of *FUJITSU PRIMEHPC FX1000* equipped with A64FX processors, each of which consists of 48 cores and 2 or 4 assistant cores, achieving a theoretical peak performance of 3.38 TFLOP/s and a total peak performance of 25.9 PFLOP/s. The total memory capacity in Odyssey is 240 TiB, while its total memory bandwidth is 7.8 PB/s. Each node is interconnected using *Tofu Interconnect D* with a bisection bandwidth of 13.0 TB/s. The Data/Learning Node Group (*Aquarius*) consists of general-purpose CPUs (Intel Xeon Platinum 8360Y (Ice Lake), 36 cores, 2.4 GHz) and 8 GPUs (NVIDIA A100 Tensor Core (SXM4, 40 GB)) per node. The total peak performance of Aquarius is 7.2 PFLOP/s, with a total memory capacity of 36.5 TiB and a total memory bandwidth of 578.2 TB/s. Each of Aquarius node is interconnected using InfiniBand HDR (200 Gbit/s per link), achieving full bisection bandwidth between nodes. Odyssey and Aquarius are connected with each other a network bandwidth of 2.0 TB/s using a total of 160 InfiniBand EDR (IB-EDR) (100 Gbps) links. Additionally, Aquarius can communicate externally at a total network transfer rate of 800 Gbps. All nodes of Aquarius can directly access various external resources including servers, storage, and sensor networks through external networks such as SINET, enabling real-time data acquisition [4]. The storage system of the overall Wisteria/BDEC-01 consists of a shared file system (capacity: 25.8 PB, data transfer rate: 0.504 TB/s), and fast file system (FFS) with SSDs (1.0 PB, 1.00 TB/s), accessible from both of Odyssey and Aquarius.

h3-Open-BDEC (Software). *h3-Open-BDEC* [5,6] is an innovative software infrastructure for integration of (S+D+L) on Wisteria/BDEC-01. Its development has been supported by Japanese Government during FY.2019-2023. It is designed for extracting the maximum performance of the supercomputers with minimum energy consumption, focusing on: (1) Innovative methods for numerical analysis, (2) Hierarchical data driven approach based on machine learning, and (3) Software for heterogeneous systems, such as Wisteria/BDEC-01 with

Odyssey and Aquarius based on different architectures, and no MPI library supporting both simultaneously, a single MPI job over Odyssey-Aquarius is impossible. For this reason we developed a communication procedure based on *h3-Open-SYS/WaitIO*, a library that provides a MPI-like interface to facilitate communication between multiple parallel programs running on Odyssey and Aquarius, via IB-EDR (*WaitIO-Socket*) and via Fast File System (*WaitIO-File*). The *h3-Open-UTIL/MP* is a multi-physics coupler, which was originally developed for a homogeneous supercomputer system using MPI. In this project, we combine *h3-Open-UTIL/MP* with *h3-Open-SYS/WaitIO* to integrate (S+D+L) on a heterogeneous environment like Wisteria/BDEC-01 [7]. To submit such (S+D+L) jobs, currently two separate jobs must be submitted from Odyssey and Aquarius, respectively.

3 Target Applications

To evaluate the portability of real applications, we ported TSMP and mAIA to the DEEP and Wisteria/BDEC-01 modular platforms. Both TSMP and mAIA are realized by coupling multiple modules, but the difference is that TSMP couples modules using a high-level coupling library called OASIS3, while mAIA couples modules at the MPI communication library level. The target applications for each are described below.

3.1 TSMP, ToySMP

The Terrestrial Systems Modelling Platform (TSMP) is a multi-domain and multi-physics framework [8] coupling the atmospheric model *COSMO*, the land surface model *CLM*, and the hydrological model *ParFlow* via the *OASIS3-MCT* coupler. TSMP operates under a highly flexible Multiple-Program-Multiple-Data (MPMD) paradigm in which different combinations of the solvers are possible. Similarly, it is possible to map each of the solvers to different computational resources. Heterogeneous and modular computing are an interesting approach for TSMP because of the inherently complex load balancing of the three solvers. Since ParFlow has been ported to GPUs [9], TSMP is capable of running on heterogeneous and modular supercomputers, running COSMO and CLM on CPUs and (currently only) ParFlow on GPUs. Experiments with TSMP have shown that MSA can increase resource and energy efficiency, and improve the scalability of the coupled system, but time-to-solution improvements only happen under certain conditions. In the future, the upcoming TSMP2 will be able to offload ParFlow and ICON (which will replace COSMO) onto GPUs, leaving only CLM on CPUs (as it is not well suited for GPU computations) and is expected to result in overall speed-up for TSMP. The relative loads of the components will guide decisions concerning onto which hardware the solvers are mapped.

TSMP has been tested on the DEEP experimental modular system, using different CPU and GPU modules as well as on the modular production system

JUWELS. However, porting it to other experimental systems can prove particularly challenging, as substantial porting work is required to enable the complex software environment required by TSMP. This makes experimenting with TSMP on multiple systems rather cumbersome. Additionally, the complex multi-physics implies that physical consistency across domains needs to be guaranteed in test problems, further hindering the flexibility required for experimentation. *ToySMP* was developed to alleviate these issues. As its name implies ToySMP is a toy (dwarf, mini-app) version of TSMP with very minimal software dependencies. It maintains the structure of TSMP in terms of the MPI communicators and the coupling strategy, but abstracts most of the complexities in the implementation, which require physical consistency (pre-processing) and allows further flexibility in terms of computational load. Nevertheless, ToySMP is intended to closely follow the implementation in TSMP in the communicator structure, in the common coupling interfaces and APIs, and in the coupling APIs in the toy components.

3.2 mAIA

The multi-physics framework for the code mAIA [10] has been pursued at the Institute of Aerodynamics of the RWTH Aachen University in Germany since 2004. The code is able to simulate compressible and incompressible flows, aeroacoustics, combustion flames, and multiphase problems. The code mAIA consists of noble numerical schemes for the finite-volume (FV) [11], the lattice-Boltzmann (LB), and the discontinuous Galerkin (DG) methods. The code has been tuned and executed with a mixed MPI/OpenMP parallelism. The mesh generation is automated to obtain hierarchical Cartesian grids with a quadtree/octree structure. Furthermore, the solver supports solution adaptive mesh refinement as well as flows with immersed moving boundaries.

The simulation of a biofluid problem is performed by using a LB method in the quasi-incompressible flow regimes. By solving the microscopic distribution of the fluid particles the LB method can determine flow fields in a simpler manner than the typical finite-difference and finite-volume approaches. The LB algorithm operates in a fully local manner, and its locality ensures a massive parallelism on HPC systems. The LB method based on the BGK model [12] is used to solve a computational domain generated with a local refinement method at the boundaries. The LB method has been validated and applied to generic flow configurations [13] and bio-fluid mechanical problems [14].

A hierarchical Cartesian computational mesh is generated using the massively parallel method presented in [13]. That is, each MPI-process initially generates a cube, in the following referred to as *cell*, with edge length equal to the maximum geometrical size in the three Cartesian coordinate axes. Subsequently, this cube is subdivided into eight child cells constituting an octree with *parent-child* relationships between the initial cell on level l_0 and its descendants on level l_1. Cells outside the geometry are identified and removed from the tree. The subdivision process then recursively continues the refinement up to a user-defined level l_m. On l_m, all levels l_j $(j < m)$ are removed and a Hilbert curve is placed in the

remaining cells. The computational mesh is then equally distributed among the MPI processes by cutting the Hilbert curve with respect to the Hilbert identifier. Thereby, each process keeps only those cells that have been assigned to it. In a parallel algorithm each MPI process subsequently continues the refinement up to a user-defined level l_n. Finally, neighborhood information across MPI ranks is globally restored and a cell ordering is introduced following the Hilbert curve on level l_m and a z-ordering, i.e., a depth-first ordering, for all levels l_k ($m < k \leq n$). It should be noted that although the algorithm is capable of creating boundary- or patch-refined meshes, only uniform meshes are used in the current study.

The issues caused by imbalanced processes are treated by a load-balancing technique. Note that in the current study a generic configuration is chosen to provide a fundamental basis to understand the motions of fluids in a human brain.

4 Porting/Implementations

In this section, details of the implementations of the *h3-Open-BDEC* to TSMP/ToySMP and mAIA are described.

4.1 Overview: Porting h3-Open-BDEC

h3-Open-SYS/WaitIO. The *h3-Open-SYS/WaitIO* (WaitIO) is a heterogeneous communication library among system areas and computer centers. It supports multiple communication devices that are POSIX Socket (*WaitIO-Socket*), POSIX File (*WaitIO-File*), the combination of POSIX Socket and POSIX File (*WaitIO-Hybrid*), ibverbs (*WaitIO-Verb* for InfiniBand), and uTofu (*WaitIO-Tofu* for Tofu interconnect). Application users write coupling communication programs in an MPI-like manner over multiple heterogeneous systems without installing any system-level software, just with the *WaitIO* user level libraries.

h3-Open-UTIL/MP. The *h3-Open-UTIL/MP* (UTIL/MP) is a coupling library developed as part of *h3-Open-BDEC*. This library possesses functionalities equivalent to *OASIS3-MCT* (OASIS3), including process group management, data exchange, and grid remapping. Additionally, WaitIO enables coupling between modules with different architectures becomes possible. In Wisteria/BDEC-01, MPI is employed for communication within the same module, while global communication involves both MPI and WaitIO, and communication between modules solely utilizes WaitIO. Using this capability, coupled computations are actually being performed where an atmospheric model runs on CPU nodes and an Machine Learning (ML) library runs on GPU nodes, allowing for the physical processes of the atmospheric model to be learned by machine learning [15].

4.2 TSMP, ToySMP

As mentioned in Sect. 3.1, in this study a group of toy models (ToySMP) was used to mimic the behavior of TSMP. ToySMP consists of three components: *compA*, *compB*, and *compC*. Each model corresponds to *COSMO*, *CLM* and *ParFlow* respectively. Similar to TSMP, compA and compB are written in Fortran, while compC is written in C language. The three TSMP components are coupled by OASIS3. The long term goal of our research is to replace OASIS3 with UTIL/MP in TSMP to perform coupled calculations across different platforms. However, due to the effects this has on dependencies, as a preliminary step, ToySMP has been designed to allow the use of both OASIS3 and UTIL/MP couplers. To achieve this, we developed a module called `toysmp_interface` (and also `toysmp_c_interface`) that provides a coupling interface for the three toy-models. These toy-models call APIs of `toysmp_interface`, and `toysmp_c_interface` APIs, depending on the configuration, internally invoke the appropriate APIs of either OASIS3 or UTIL/MP, simply via preprocessor branching.

4.3 mAIA

The simulations using mAIA were conducted on the DEEP and the Wisteria/BDEC-01 systems. The task consists of two parts, i.e., the first step to perform benchmarks on a single module and the second to integrate the WaitIO interface into the code mAIA without performance degradation. The main objective is to realize the MSA simulations of mAIA on both systems. In this section the porting and implementation of WaitIO is discussed with the code structure.

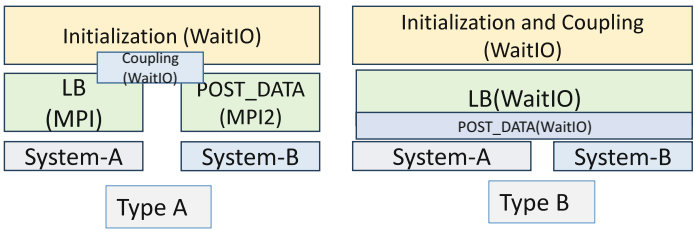

Fig. 2. Strategies towards the heterogeneous computing version of mAIA based on a WaitIO implementation.

The mAIA consists of multiple *solver modules* and *coupler modules*. Users are able to select the solvers and couplers depending on their purpose. The WaitIO implementation to the mAIA modules considers two strategies in developing the heterogeneous computing version of mAIA (see Fig. 2). Type A in Fig. 2 uses multiple solvers, e.g., *LB-POST_DATA* (LB on the System-A and POST_DATA on the System-B), *FV-DG* or *LB-DG*, each of which is executed on multiple

heterogeneous systems. In this case, WaitIO is used for the initialization and the data coupling among different solver modules. Each solver uses MPI communication and exchanges the simulation results via shared memory. Therefore the data exchange parts in the mAIA coupler needs new functions by the WaitIO implementation. In the Type B strategy in Fig. 2, one solver is executed in multiple systems, i.e., the mAIA coupler between the mAIA modules is deactivated and processes *LB-POST_ DATA*. In this case, WaitIO is used to communicate data in a single solver. In the design of a heterogeneous computing version of mAIA using WaitIO, the porting/implementation aims at the seamless interface for both Type A and Type B cases. The implementation procedure uses the MPI conversion library of WaitIO, because it can easily replace the MPI APIs by adding `WAITIO_` prefixes to MPI function names, MPI datatypes, and MPI operation types.

The targeted modules are a lattice Boltzmann solver (`LB`) and the post processing module (`POST_DATA`). For the Type A implementation, the initialization part of mAIA was converted by replacing MPI functions with WaitIO ones. For the Type B implementation, the LB part was also rewritten by the WaitIO functions. The data coupling part between the `LB` and `POST_DATA` solvers is modified to ensure the on-the-fly data post-processing. Furthermore, mAIA supports a GPU offloading via the C++ parallel *STL*. The implementation of the WaitIO version of mAIA includes heterogeneous computing between CPU and GPU systems. The evaluation of CPU and GPU coupling is planned on both the DEEP and the Wisteria/BDEC-01 system.

5 Performance Evaluations

5.1 TSMP, ToySMP

Overview. ToySMP was created to mimic the behavior of each component model of TSMP. The execution pattern of the models and the flow of data are also set to be similar to TSMP. These processes are illustrated in Fig. 3. *compA* and *compB* are executed on CPU nodes (cluster module in DEEP, and Odyssey in Wisteria/BDEC-01), while *compC* is executed on GPU nodes (booster module in DEEP, and Aquarius in Wisteria/BDEC-01). The models *compA* and *compC* perform the first step in the execution and send the results to *compB*. After receiving this data, *compB* performs one-step computation and returns the results to *compA* and *compC*. Since *compA*, *compC*, and *compB* are executed sequentially, the time involved in data exchange includes the waiting time for the calculations of the counterpart model. Consequently, it is not possible to calculate performance metrics like throughput from the measured data. Therefore, in this study, we decided to evaluate the relative scalability when changing the number of processes.

Problem Setting: Since the coupler of TSMP performs grid remapping, each component can compute on its own grid system. However, in this study, to

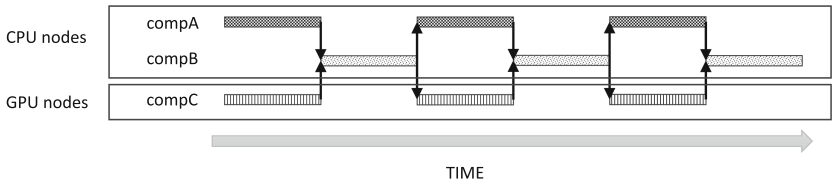

Fig. 3. Time integration and data exchange pattern of ToySMP, following the same sequence as TSMP.

simplify the performance evaluation, the three components were assumed to have the same grid. Additionally, the domain decomposition method was unified with a one-dimensional *east-west* division. Therefore, when the same number of processes is assigned to each component, data exchange will be performed on a one-to-one basis per process. In this experiment, the grid size was set to 1024×1024, and the number of exchange fields from one component to another per one time step was 10. Therefore, the total amount of communication data per step is approximately 335 MB ($1024 \times 1024 \times 8 \times 10 \times 4$). The number of processes assigned to each component was set to the same value and varied from 1 to 128. In this case, the data size of single field per process is shown in Table 2.

Table 2. Data sizes per process (strong scaling).

processes	1	2	4	8	16	32	64	128
data size (byte)	8,388,608	4,194,304	2,097,152	1,048,576	524,288	262,144	131,072	65,536

Results and Discussion: As mentioned in Sect. 4.1, WaitIO has three modes: *Socket*, *File*, and *Hybrid*. The results measured on Wisteria/BDEC-01 for each mode are shown in Fig. 4, along with the results from DEEP in Socket mode. The measurements from DEEP demonstrate shorter execution times and better scalability compared to those from Wisteria/BDEC-01. This difference is likely attributed to the hardware configuration differences between DEEP and Wisteria/BDEC-01. Specifically, DEEP has a smaller heterogeneity between its CPU and GPU modules and a more tightly integrated system compared to Wisteria/BDEC-01. In Hybrid mode, the threshold for switching between *Socket* and *File* can be set via an environment variable, which was set to 524288 for this experiment. This value corresponds to 16 processes. The measurement results indicate that for 32 or more processes, the results of the *Hybrid* mode align with those of the *Socket* mode. Conversely, within the range where the *File* mode is applied, the *Hybrid* mode outperforms the *File* mode. This is because the *Hybrid* mode uses the *Socket* mode for the rendezvous protocol to transfer

larger message communications. UTIL/MP is equipped with the capability to aggregate multiple data transfers into a single transmission. This functionality was originally implemented to reduce the latency of routine calls in MPI communication. Additionally, since WaitIO, particularly *WaitIO-File*, shows better performance with larger data sizes, applying this feature in our experiment was expected to enhance performance. Therefore, we configured UTIL/MP to aggregate 10 field data sets and exchange all at once, and conducted the measurements accordingly. The measurement results for DEEP and Wisteria/BDEC-01 under this configuration are shown in Fig. 5. For DEEP, there is a slight performance improvement up to 32 processes, but performance plateaus at 64 and beyond. In the *Socket* mode on Wisteria/BDEC-01, the performance remains nearly the same. However, significant performance improvements were observed in the *File* and *Hybrid* modes. Notably, in *Hybrid* mode, performance scaled effectively up to 128 processes, this demonstrates the effectiveness of UTIL/MP's functionality.

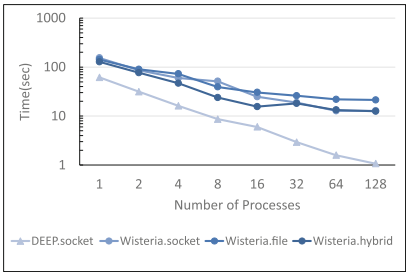

Fig. 4. Data exchange time of *h3-Open-UTIL/MP* on DEEP and Wisteria/BDEC-01

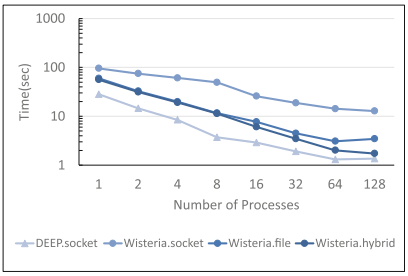

Fig. 5. Data exchange time when 10 data are packed and exchanged together.

5.2 mAIA

Overview: We selected *LB-POST_DATA* solver module coupling. And, the LB module of mAIA possesses three major functions which simulate the lattice gas behaviour, i.e., the collision step, the propagation step, and the prolongation step. These functions calculate the flow field based on the BGK model implemented in the LB module. At each time step the particle collision determines a new distribution function which streams the particle momentum in the propagation direction. The computation time of the three major functions extends to more than 50% of the total computing time. The second largest portion is consumed to apply the physical conditions at the domain boundaries and to update the field values at the interface. The coupling with the LB module is enabled by defining the number of solvers and each solver type. In this study the post-processing module is activated to perform the on-the-fly data processing of LB solutions.

Problem Setting: The evaluation procedure for a WaitIO coupling is defined by following steps using *LB-POST_DATA* solver module coupling. Since both the Type A and Type B implementations of mAIA are in progress, the evaluation is performed as follows: the Type A evaluation measures the initialization time of WaitIO by comparing it with MPI on the DEEP, and with homogeneous processing on Wisteria/BDEC-01. The Type B evaluation measures the parallel performance via scaling tests using a heterogeneous combination of Arm CPUs on Odyssey and Intel CPUs on Aquarius of the Wisteria/BDEC-01 system. In the evaluation of the Type A initialization time, on DEEP ParaStation MPI was used as the MPI library, and *WaitIO-Verbs* were used as the WaitIO library. On Wisteria/BDEC-01, Fujitsu MPI was used as the MPI library, and WaitIO-Tofu was used as the WaitIO. The evaluation time in minutes was measured using `LB` and `POST_DATA` execution with 5000 steps. The `LB` part used an MPI library in each case. In the evaluation of the Type B execution time, *WaitIO-Socket* and *WaitIO-Hybrid* were used in Wisteria/BDEC-01 with `LB` and `POST_DATA` coupling. The `LB` and `POST_DATA` solvers run on both Odyssey and Aquarius at the same time in a heterogeneous computing environment.

Results and Discussion: Figure 6 shows the Type A initialization time. In Fig. 6 the number of processes per node was 4 in Odyssey, and 16 processes per node were used in the DEEP cluster module. In Fig. 6 the processing time of the WaitIO was competitive or faster than that of MPI up to 128 processes. In cases with a small number of processes, WaitIO was faster. However, increasing the number of processes over 128 MPI ranks, the MPI libraries perform faster than the WaitIO. In the comparison of the run-times determined on the DEEP and the Wisteria/BDEC-01(Odyssey), the execution on the Odyssey system was much slower in a small number of processes. With an increasing number of processes, the runtime on the Odyssey system was competitive with that on the DEEP system. Since the problem size is fixed, the results are caused by the increased overhead in communication between processes. Figure‘7 shows the results of Type B execution time. In Fig. 7 the notation 128(O:64+A:64) indicates a job execution with a total of 128 processes, from which 64 processes ran on Odyssey and 64 processes ran on Aquarius. Note that the case uses 4 processes per node in Odyssey and 64 processes per node in Aquarius due to the limitation of the number of nodes. From the results in Fig. 7, one can conclude that the heterogeneous coupling of Arm CPU and Intel CPU functions works correctly, and improves the performance when increasing the number of processes. Comparison between *WaitIO-Socket* and *WaitIO-Hybrid*, in the case with 128 processes, the *WaitIO-Hybrid* setup is faster. However, the setup with *WaitIO-Socket* shows a better performance in the case with 512 processes, where the size of communication messages is reduced by increasing the number of processes. The *WaitIO-Hybrid* performance is degraded when the size of messages exceeds 512 KB.

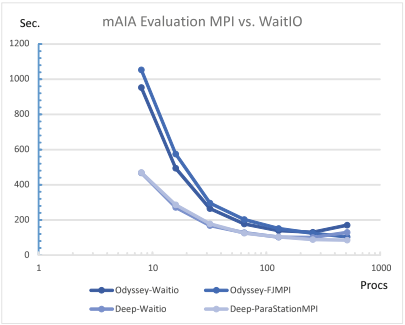

Fig. 6. Comparison of mAIA execution time (lower is better): MPI vs. WaitIO in Type A.

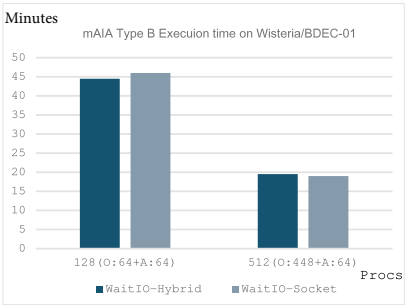

Fig. 7. Comparison of mAIA execution times (lower is better): WaitIO in Type B.

6 Related Work

The Modular Supercomputing Architecture (MSA) has been already described in more detail in [2,16]. Previous work documenting the experience of application developers that ported and benchmarked different use cases on the DEEP system are reported in [17] and [18]. Experiments on other MSA systems in JSC have been published in [19,20]. All those previous experiments were done with different application use-cases to those reported in the current work, and no comparison was done with Wisteria/BDEC-01 or between any two different MSA systems. The present work is unique in that it compares different hardware and software implementations of the MSA concept, using different use cases to those reported on previous publications.

7 Concluding Remarks

This study evaluated the performance of two applications (TSMP, mAIA) on two systems (DEEP, Wisteria/BDEC-01), which are based on the idea of MSA. We introduced h3-Open-BDEC, originally developed for coupled simulations on Wisteria/BEDC-01 with heterogenous architectures, to JSC's DEEP system. We evaluated the performance of two applications on two systems from two perspectives: hardware configuration and coupling libraries. Obtained results show that, like the original TSMP on MSA, performance is highly dependent on configuration settings, but scalability can be achieved with proper tuning.

For this study, we developed ToySMP, which simulates the data transfer and management between applications in the Terrestrial Systems Modelling Platform (TSMP). We introduced UTIL-MP to ToySMP for coupling, and evaluated the performance of the code on DEEP and Wisteria/BDEC-01. Obtained results show that, like the original TSMP on MSA, performance is highly dependent on

configuration settings, but scalability can be achieved with proper tuning. Moreover, UTIL/MP's feature for aggregating data transfers was validated. Future work includes improving ToySMP so that it can more accurately simulate the behavior of the original TSMP, deploying the original TSMP to Wisteria/BDEC-01, and applying h3-Open-BDEC to it.

TSMP has already realized coupled calculations in a heterogeneous environment for the DEEP system, but for mAIA, we started by considering the coupling of multiple components. We evaluated two types of workloads. Type-A is running LB application of mAIA coupled with post-processing across multiple systems, and compares performance of communications by MPI and those by WaitIO library of h3-Open-BDEC on both of DEEP and Wisteria/BDEC-01 (Odyssey). Codes using MPI and WaitIO provide similar performance on both DEEP and Wisteia/BDEC-01 (Odyssey), respectively. Type-B evaluates the performance of a single application on heterogeneous environments of Wisteria/BDEC-01. Results of Type-B show that the `LB` solver in mAIA performs well on Wisteria/BDEC-01 using a heterogeneous combination of Arm CPU and Intel CPU. We continue to develop the coupled mAIA codes in two direction, Type-A and Type-B, and elucidate the effectiveness of such heterogeneous coupling applications. In the direction of Type-A, we are planning integration of simulations and machine learning/AI workloads using h3-Open-BEDC, as we have already done in previous works [7].

Although this study is very preliminary, it demonstrated the effectiveness of coupled simulations on MSA-based systems using h3-Open-BDEC. In the future, we would like to continue to improve the software in terms of both computational efficiency and ease of use, and verify its effectiveness for a wider range of applications using various MSA-based systems.

Acknowledgements. Part of the work reported here has received funding form the DEEP Projects from the European Commission's FP7, H2020, and EuroHPC Programmes, under Grant Agreements n° 287530, 610476, 754304, and 955606. The EuroHPC Joint Undertaking (JU) receives support from the European Union's Horizon 2020 research and innovation programme and Germany, France, Spain, Greece, Belgium, Sweden, and Switzerland. Moreover, part of the work is supported by *JSPS Grant-in-Aid for Scientific Research (S)*(19H05662), and by *Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures* (jh230017, jh240029). DCV and SP also acknowledge that this work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1502/1-2022 - Projektnummer: 450058266.

References

1. Silvano, C. et al.: A survey on deep learning hardware accelerators for heterogeneous HPC platforms (2023). https://arxiv.org/abs/2306.15552
2. Suarez, E., Eicker, N., Lippert, T.: Modular supercomputing architecture: from idea to production. In: Computing: From Petascale toward Exascale. CRC Press, 3rd edn., pp. 223–251. Florida, USA (2019)

3. Pickartz, S.: Virtualization as an enabler for dynamic resource allocation in HPC, Dissertation, RWTH Aachen University, Aachen (2019)
4. Homepage of Wisteria/BDEC-01. https://www.cc.u-tokyo.ac.jp/en/supercomputer/wisteria/system.php
5. Hopmepage of h3-Open-BDEC. https://h3-open-bdec.cc.u-tokyo.ac.jp/
6. Iwashita, T., et al.: h3-Open-BDEC: innovative software platform for scientific computing in the exascale era by integrations of (simulation + data + learning). Project Poster, ISC-HPC (2020)
7. Sumimoto, S., et al.: A system-wide communication to couple multiple MPI programs for heterogeneous computing. In: Proceedings of 23rd PDCAT (Parallel and Distributed Computing, Applications and Technologies) (2023). https://doi.org/10.1007/978-3-031-29927-8_25
8. Shrestha, P., Sulis, M., Masbou, M., Kollet, S., Simmer, C.: A scale-consistent terrestrial systems modeling platform based on COSMO, CLM, and ParFlow. Monthly Weather Rev. **142**, 3466–3483 (2014)
9. Hokkanen, J., Kollet, S., Kraus, J., Herten, A., Hrywniak, M., Pleiter, D.: Leveraging HPC accelerator architectures with modern techniques - hydrologic modeling on GPUs with ParFlow. Computational Geosciences (2021)
10. Homepage of AIA. http://www.aia.rwth-aachen.de/forschung/numerische-stroemungsmechanik/
11. Cetin, M.O., Koh, S.R., Meinke, M., Schröder, W.: Computational analysis of exit conditions on the sound field of turbulent hot jets. Comptes Rendus Mécanique **346**(10), 932–947 (2018). https://doi.org/10.1016/j.crme.2018.07.006
12. Bhatnagar, P.L., Gross, E.P., Krook, M.: A model for collision processes in gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. Phys. Rev. **94**(3), 511–525 (1954)
13. Eitel-Amor, G., Meinke, M., Schröder, W.: A Lattice-Boltzmann method with hierarchically refined meshes. Comput. Fluids **75**, 127–139 (2013). https://doi.org/10.1016/j.compfluid.2013.01.013
14. Koh,S., Kim, J., Lintermann, A.: Numerical analysis of oscillatory flows in the human brain by a lattice-boltzmann method. In: Proceedings of 14th WCCM-ECCOMAS Congress 2020 (2021). https://doi.org/10.23967/wccm-eccomas.2020.226
15. Arakawa, T., Yashiro, H., Nakajima, K.: Development of a coupler h3-Open-UTIL/MP. In: International Conference on High Performance Computing in Asia-Pacific Region, 72–83 (2022). https://doi.org/10.1145/3492805.3492809
16. Suarez, E., et al.: Modular supercomputing architecture. ETP4HPC (2022). https://doi.org/10.5281/zenodo.6508394
17. Kreuzer, A., Suarez E., Eicker N., Lippert, T.: Porting applications to a Modular Supercomputer. IAS Series vol. 48, p. 1–254, Forschungszentrum Juelich GmbH, Juelich (2021). http://hdl.handle.net/2128/30498
18. Suarez, E., Kunkel, S., Kuesters, A., Plesser, H.E., Lippert, Th., Modular Supercomputing for Neuroscience, In: Amunts, E., BrainComp 2019, LNCS, vol 12339, pp 63–80, Springer International Publishing, Cetraro, Italy (2019). https://doi.org/10.1007/978-3-030-82427-3_5, http://hdl.handle.net/2128/28352
19. Kreuzer, A., Eicker, N., Amaya, J., Suarez, E., Application Performance on a Cluster-Booster System. In: IEEE Proceedings of IPDPSW, pp. 68–78, (2018). https://doi.org/10.1109/IPDPSW.2018.00019
20. Bishnoi, A., et al.: Earth system modeling on modular supercomputing architecture: coupled atmosphere-ocean simulations with ICON 2.6.6-rc. Geosci. Model Dev. **17**(1), 261–273 (2024). https://doi.org/10.5194/gmd-17-261-2024