JOIN (Jurnal Online Informatika)
p-ISSN: 2528-1682, e-ISSN: 2527-9165
Volume 10 Number 2 |December 2025: 248-259
DOI: 10.15575/join.v10i2.1557

Performance Evaluation of NAS Parallel and High-Performance Conjugate Gradient Benchmarks in Mahameru

Taufiq Wirahman[1], Arnida L. Latifah[1,2], Furqon H. Muttaqien[1,3], I Wayan Aditya Swardiana[1], Andria Arisal[4], Syam B. Irianto[1], Rifki Sadikin[1]
[1]Research Center for Computing, National Research and Innovation Agency, Bogor, Indonesia
[2]Informatics Study Program, School of Computing, Telkom University, Bandung, Indonesia
[3]Information System Study Program, School of Applied Sciences, Telkom University, Bandung, Indonesia
[4]Research Center for Data and Information Sciences, National Research and Innovation Agency, Bandung, Indonesia

Article Info

Article history:

Received January 15, 2025
Revised June 7, 2025
Accepted July 13, 2025
Published August 17, 2025

Keywords:

Conjugate Gradient Algorithm
High-Performance Computing
MPI vs OpenMP
Supercomputing Performance
Parallel Computing

ABSTRACT

High-Performance Computing (HPC) plays a crucial role in accelerating scientific advancement across numerous fields of research and in effectively implementing various complex scientific applications. Mahameru is one of the largest national HPC systems in Indonesia and has been utilized by many sectors. However, it has not undergone proper benchmarking evaluation, which is vital for identifying issues related to hardware and software configurations and confirming system reliability. Therefore, this study aims to evaluate the performance, efficiency, and capabilities of Mahameru. We present a benchmarking system on Mahameru utilizing two benchmark suites: the NAS Parallel Benchmarks (NPB) and the high-performance conjugate gradient (HPCG) benchmark. Our results indicate that the NPB exhibits a lower speedup in Message Passing Interface (MPI) compared to OpenMP, which can be attributed to the communication overhead and the nature of the computational tasks. Additionally, the HPCG benchmark demonstrates that Mahameru performance can compete with the lower tiers of the Top 500 supercomputers. When operating at full capacity, Mahameru can achieve approximately 2.5% of its theoretical peak performance. While the system generally performs reliably with parallel algorithms, it may not fully leverage hyperthreading with certain algorithms. This benchmark result can serve as a basis for decision-making regarding potential upgrades or changes to a system.

Corresponding Author:

Arnida L. Latifah,
Informatics Study Program, School of Computing, Telkom University, Bandung
Main Campus (Bandung Campus), Jl. Telekomunikasi no. 1, Bandung, West Java, Indonesia 40257
Email: arnidalatifah@telkomuniversity.ac.id, arnida.l.latifah@brin.go.id

1. INTRODUCTION

The development of High-Performance Computing (HPC) facilities in the National Research and Innovation Agency (BRIN) named Mahameru BRIN HPC was started in 2023 by integrating existing computing servers and the establishment of a new HPC system. It has served various high-performance computing requests from many research fields. As one of the national research facilities, the Mahameru BRIN HPC is open for Indonesian researchers. Before the establishment of Mahameru BRIN HPC, the first generation, referred to as Gen 1, was initiated by the Indonesian Institute of Sciences (LIPI) and utilized a Fujitsu system featuring the Sandy Bridge architecture in 2014. Moving forward to the second

generation, known as Gen 2, which was still under the authority of LIPI, it was transitioned to an HP system in 2018 that was built on Broadwell architecture. Subsequently, the third generation (Gen 3) was also a product of LIPI and utilized a Dell system that was implemented in 2019, showcasing the Skylake architecture. The fourth generation, Gen 4, was then built during the BRIN era and known as Mahameru BRIN HPC. It was characterized by the adoption of a Lenovo system that was rolled out in 2023, featuring the Icelake architecture, which has been designed to significantly enhance performance and efficiency. Within many years of HPC development, the HPC system has not been subjected to any benchmarking assessments, leaving awareness of the actual performance metrics and capabilities of these advanced computing systems.

Benchmarking HPC systems is essential for various reasons, primarily to investigate performance evaluation, guide system design, and improve productivity. By establishing standardized metrics and methodologies, benchmarking facilitates the comparison of different systems and identifies performance bottlenecks, ultimately leading to more informed decisions regarding hardware and software configurations. Benchmarks simulate specific workloads, allowing for the assessment of system capabilities and performance under realistic conditions [1]. Benchmarking provides critical insights into system architecture, helping designers optimize configurations for specific workloads, such as those seen in scientific computing and Artificial Intelligence (AI) applications [2]. Effective benchmarking can measure HPC productivity by correlating mission goals with resource utilization, thus enabling more efficient scientific outcomes [3]. A public repository of productivity benchmarks can standardize assessments across the HPC community, fostering collaboration and innovation [3]. Specifically, benchmarking is beneficial for users, administrators, and management. Users can get an overview of the system's performance, so they can adjust the configuration of their applications to make them more optimal. Administrators can find the strengths and weaknesses of the systems, reveal the bottleneck, and get an overview of properties that are related and unrelated to expectations of system performance. For management, benchmarking provides policy consideration of the use of the HPC system and a basis for future system design development. There are various HPC benchmarking frameworks had been conducted in many different HPC facilities, for example Taub and TianHe-2 [4][5], Egyptian National HPC Grid (EN-HPCG) [6], University of Luxembourg HPC [7], Barcelona Supercomputing Center [8], and Fugaku [9][10][11]. Other benchmarks for specific architectures were also investigated previously, for example, Intel Xeon with Xilinx Alveo U280 [12] and Intel x86 server CPU architecture with Broadwell EP and Cascade Lake SP [13].

HPC benchmarks can be categorized into two primary types: system/synthetic and application benchmarks. Synthetic benchmarks evaluate the performance of hardware components and configurations. The benchmark measures the theoretical peak performance of CPU, memory, network, and storage components [14], for example Taub and TianHe-2 supercomputers benchmark presented in [4]. Meanwhile, application benchmarks evaluate the configuration and performance of specific applications on HPC systems, for instance, I/O parallel libraries used in RegCM [15]. Various benchmark suites are accessible to evaluate the efficacy of an entire system and its constituent subsystems, and each of these tools presents its own set of benefits and drawbacks [16].

Addressing the critical gap that no prior benchmark has been applied. This study presents the first-ever application of the synthetic benchmark to evaluate the performance, efficiency, and capabilities of Mahameru BRIN HPC. This provides the first comprehensive insights about the performance baseline, the architectural strengths and weaknesses, the optimum configuration, and the basis for future system development. To do so, this study uses two benchmark suites: NAS Parallel Benchmarks (NPB) and High-Performance Conjugate Gradients (HPCG) Benchmark [21][22].

The subsequent sections of this paper are structured as follows: Section 2 describes the hardware and software of Mahameru HPC BRIN, the benchmark suites, the experiments design, and evaluation metrics used in Mahameru HPC BRIN benchmark; Section 3 presents the results along with a discussion; and finally, Section 4 encapsulates the conclusions and explores potential future work.

2. METHODS

2.1. Mahameru HPC Specification

In this section, we present the specifications of the Mahameru and its management system. Mahameru's architecture is given in Figure 1. A user can connect to Mahameru's login node through a secure connection by providing the user's credentials and the user's machine's credentials. Once the user connects to the login node, the user can hop to the management node to use computing resources

p-ISSN: 2528-1682
e-ISSN: 2527-9165

available at the Mahameru system. The Mahameru HPC uses SLURM as HPC workload management [19]. The Mahameru HPC has several resource partitions to accommodate different computing requirements, such as short, medium, or long time of running times and modes of parallelism for single or multi-node.

Regarding software module management in Mahameru HPC, and since Mahameru HPC is intended to provide computing resources to a broad spectrum of research, module systems are being installed to manage different software module sets for typical computing requirements in various research fields. The Mahameru HPC also served as a computing facility for sequencing and a Cryo-EM laboratorium facility at BRIN. The Mahameru BRIN HPC was deployed with the operating system of Red Hat Enterprise Linux 8.9 and specifications as shown in Table 1. All nodes in Mahameru are connected through a high-speed Infiniband with a bandwidth 2 x 100 Gbps network with a spine-leaf configuration to ensure reduced network latency and hop count, increasing network efficiency. Mahameru also uses an IBM GPFS (SpectrumScale) high-performance storage cluster mounted as a filesystem across master and computing nodes with Journal Storage and High-Density Storage with a total usage capacity of 6 Petabytes. Computing resources based on GPU servers are also integrated into the Mahameru BRIN HPC. We provide GPU cards like DGX A100 and DGX A1.

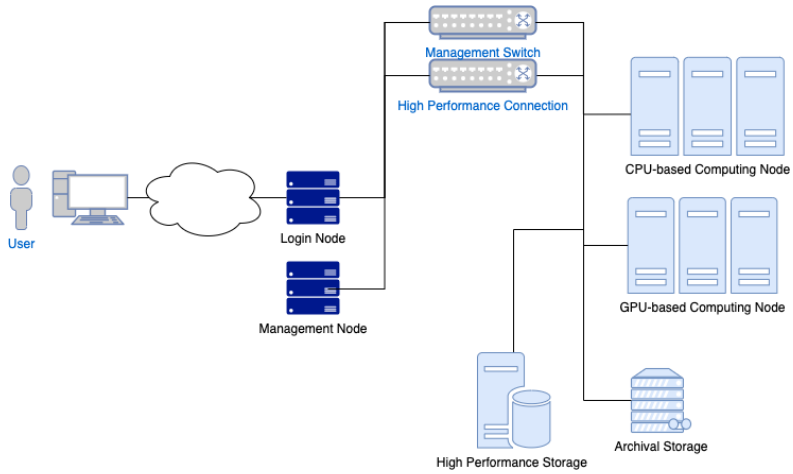

Figure 1. Overall Mahameru HPC System Architecture.

Table 1. Hardware specification of Mahameru High-Performance Computing System

	Computing Node	High-Memory node
Processors	2 x Xeon Gold 6338 (2 GHz, Turbo 3.2/2.6 GHz, 32 cores, 64 threads, L3 48 MB, L2 40MB, L1 1 MB/1.5 MB)	8 x Xeon Gold 8280 (2.7 GHz, Turbo 3.2/2.6 GHz, 28 cores, 64 threads, L3 48 MB, L2 40MB, L1 1 MB/1.5 MB)
Memory	256 GB, DDR4-3200 MHz, 8 channel	1536 GB, DDR4-2933 MHz, 8 channel
Number of nodes	92 nodes	1 node

2.2. NAS Parallel Benchmark (NPB)

The NAS Parallel Benchmarks (NPB) is a collection of programs designed to evaluate the performance of parallel supercomputers [17]. Developed by the NASA Advanced Supercomputing (NAS) Division, the benchmarks are based on computational fluid dynamics (CFD) applications and are widely used in the HPC community to assess system capabilities.

The NPB suite is important for benchmarking HPC infrastructures because of its ability to simulate real-world computational workloads. By running these benchmarks, researchers can assess computational performance that can evaluate the raw computational power of the system, including processor speed and floating-point operation capabilities; measure memory hierarchy efficiency by examining how efficiently the system handles memory operations, including cache utilization and memory bandwidth; and evaluate communication mechanisms by testing the interconnect performance, including latency and bandwidth, which is crucial for parallel applications that require frequent data

exchanges. The original NPB suite, NPB 1, was a "pencil-and-paper" specification that outlined five computational kernels and three pseudo-applications to solve the 3D Navier-Stokes system for incompressible flow [17], as presented in Table 2. They are included in the BT, SP, and LU modules. The Block Tri-diagonal (BT) uses an implicit algorithm based on Alternating Direction Implicit (ADI) factorization; the Scalar Penta-diagonal (SP) uses Beam-Warming approximate factorization to decompose the 3D matrix; and the Lower-Upper Gauss-Seidel (LU) uses the Symmetric Successive Over-Relaxation method. These benchmarks were designed to mimic large-scale CFD applications' computation and data movement characteristics, providing a standardized method for comparing different HPC systems.

As HPC technologies advanced, the NPB suite evolved to reflect these changes. NPB 2 introduced reference implementations in popular parallel programming models such as MPI (Message Passing Interface), making it easier to run the benchmarks on various parallel architectures [20][21]. NPB 3 further expanded the suite by incorporating additional benchmarks for unstructured adaptive meshes, parallel I/O, multizone applications, and computational grids, with the support of OpenMP, Java, and High-Performance Fortran [21]. This evolution ensured that the benchmarks remained relevant for assessing modern HPC systems. The benchmarks are available in different problem sizes, known as classes, to accommodate systems with varying capabilities. These classes range from Class A (smallest) to Class E (largest). The problem sizes and parameters for each class are summarized in Table 3.

Table 2. Computational kernels of the NPB suite.

Kernel	Objective
Integer Sort (IS)	To assess integer computation and data communication ability
Embarassingly Parallel (EP)	To measure the capacity of floating point operations
Conjugate Gradient (CG)	To evaluate data communication mechanisms, memory locality, and caches
Multi Grid (MG)	To stress short and long-distance data communication
Fast Fourier Transform (FT)	To simulate intensive long-distance communication
Block Tri-diagonal (BT)	To evaluate the performance of computational workloads
Scalar Penta-diagonal (SP)	To measure the efficiency of solving structured grid problems
Lower-Upper Gauss Seidel (LU)	To test the performance of solving sparse linear systems

Table 3. Hardware specification of Mahameru High-Performance Computing System

Kernel	Parameter	Class A	Class B	Class C	Class D	Class E
IS	#keys	223	225	227	231	235
	key max. value	219	221	223	227	231
EP	#random-number pairs	228	230	232	236	240
CG	#rows	14000	75000	150000	1500000	9000000
	#nonzeros	11	13	15	21	26
	#iterations	15	75	75	100	100
	eigenvalue shift	20	60	110	500	1500
MG	grid size	256x256x256	256x256x256	512x512x512	1024x1024x1024	2048x2048x2048
	#iterations	4	20	20	50	50
FT	grid size	256x256x128	512x256x256	512x512x512	2048x1024x1024	4096x2048x2048
	#iterations	6	20	20	25	25
BT	grid size	64x64x64	102x102x102	162x162x162	408x408x408	1020x1020x1020
	#iterations	200	200	200	250	250
	time step	0.0008	0.0003	0.0001	0.00002	0.000004
SP	grid size	64x64x64	102x102x102	162x162x162	408x408x408	1020x1020x1020
	#iterations	400	400	400	500	500
	time step	0.0015	0.001	0.00067	0.0003	0.0001
LU	grid size	64x64x64	102x102x102	162x162x162	408x408x408	1020x1020x1020
	#iterations	250	250	250	300	300
	time step	2.0	2.0	2.0	1.0	0.5

2.3. High Performance Conjugate Gradients (HPCG) Benchmark

HPCG benchmark is a benchmark suite software proposed by [18], specifically designed for ranking HPC systems. The HPCG benchmark evaluates HPC system performance in solving large, sparse linear systems through computations and data access patterns typical in scientific applications. This HPCG benchmark, with a memory-bound and data-intensive benchmark, complements the existing High-Performance Linpack (HPL) benchmark used by https://top500.org, a continuously updated ranked list of the most powerful computer systems in the world. The HPCG benchmark has been selected due to its broad representation of various real-world applications from bioinformatics to

environment/weather applications that employ linear solver algorithms. It measures the performance of sparse linear system equation solving, such as in 3D heat diffusion problems, additive Schwarz domain decomposition, multigrid preconditioning, and parallel Conjugate Gradient solver. Additionally, it serves as an easily accessible reference implementation for a conjugate gradient algorithm. HPCG implements a preconditioned CG approach, utilizing a local symmetric Gauss-Seidel preconditioner. During a typical execution, HPCG comprises the following phases: (i) problem setup; (ii) verification and validation testing; (iii) reference sparse matrix-vector (MV) and Gauss-Seidel timing; (iv) reference CG timing and residual reduction; and (v) optimization of CG setup and results reporting.

2.4. Experiments Design

This study consists of two benchmarking experiments, i.e., NPB experiments which utilize NPB suite 3.4.2 without the multizone version, and HPCG experiments which utilize HPCG 3.1. For the setup, both HPCG and NPB are compiled using Intel Compiler 2024.0 with the -O3 optimization option to get the best performance. We use Intel MPI 2021.11.0 to run both benchmark suites. Every node that runs the simulation is reserved exclusively, which means that other jobs are not allowed to run on the same node. Hence, the simulations are not affected by other programs and produce more accurate timing and measurement. This study is not conducted during an exclusive period, such as a maintenance period, so not all the 92 nodes are used for this benchmarking study due to being used by regular users. In NPB experiments, we conduct simulations using a combination of five parameters that have multiple values. All parameters and their possible values can be seen in Table 4. The most distinctive parameter in NPB is the programming model, which provides two different approaches, i.e., MPI programming and threaded programming. Hence, the NPB experiments consist of two sub-experiments.

Table 4. Parameters in NPB experiments.

Parameters	Values
Programming model	MPI, OpenMP
Kernel or Module	IS, EP, CG, MG, FT, BT, SP, LU
Problem class	A, B, C, D, E
Number of nodes	1, 2, 4, 8, 16, 32
Number of cores (MPI only)	1, 2, 4, 8, 16, 32, 64
Number of threads (OpenMP only)	1, 2, 4, 8, 16, 32, 64, 128

For the MPI approach, we run the simulation in a single-node and a multiple-node environment. The number of nodes used to run all modules is 1, 2, 4, and 8 nodes, except for the CG module, which also runs at 16 and 32 nodes. The number of cores in a single-node environment varies from 1 to 64 cores, while the number of cores in multiple-node environments is 64 cores per node. In total, there are 410 simulations with different parameter combinations. Each simulation is run five times. In contrast to the MPI approach, we run the OpenMP simulations only in a single node using a different number of threads. As mentioned in the architecture section, each node has two processors, and each processor has 64 threads. Thus, the maximum number of threads that can be run is 128. In total, there are 240 different parameter combinations. Same as the MPI experiment, each simulation also runs five times.

Unlike NPB experiments, HPCG experiments can only be run using MPI. The problem in HPCG is only specified by the values of the parameters NX, NY, and NZ. In this study, the values of NX, NY, and NZ are the same, i.e., 160. The simulations run from 1 node up to 64 nodes with 64 cores for each node, except for simulations using a single node. All possible values of each parameter are shown in Table 5. Therefore, there are only 12 simulations that run five times each, like NPB experiments.

Table 5. Parameters in HPCG experiments.

Parameters	Values
Number of nodes	1, 2, 4, 8, 16, 32
Number of cores (single node only)	1, 2, 4, 8, 16, 32, 64

2.5. Evaluation Metrics

To evaluate the efficiency of a program on a specific computer, it is beneficial to understand the theoretical peak performance of the machine. By measuring the program's actual performance, we can determine the percentage of the theoretical peak the program is achieving. In the context of numerical programs used in scientific computing, peak performance is quantified in FLOP/s (floating-point operations per second), which refers to the count of floating-point operations (as opposed to integer operations) such as multiplication and addition performed each second. The theoretical peak flop/s of a computer is estimated as a product of (1) number of cores, (2) number of sockets, (3) clock speed, and (4) cycles per instruction (CPI). The theoretical peak of Mahameru is then computed as written in (1).

$$\begin{aligned} R_{peak} &= (\#\ cores) \times\ (\#\ sockets) \times\ (clock\ speed) \times\ CPI \\ &= 32 \times 2 \times 2.0 \times 10^9 \times 32 \\ &= 4.096\ TFLOPS\ (per\ node) \\ &= 376.8\ TFLOPS\ (total\ 92\ nodes) \end{aligned} \tag{1}$$

Aside from R_{peak}, we also use speedup as a metric to estimate Mahameru's performance in solving problems using multiple processors. Speedup is computed by dividing the time a problem is solved in parallel by the time a problem is solved using a single processor. The formula for calculating speedup is as follows:

$$S\ =\ \frac{T_{parallel}}{T_{single}} \tag{2}$$

In an ideal scenario, if we use p processors in parallel, we expect to get p times speedup or the computation time in solving a problem is reduced by a factor of p. However, R_{peak}, we will not get this ideal speedup (perfect linear speedup) when benchmarking an HPC system because of various overhead and latency. In this case, we could use the Fraction of peak metric or the Efficiency metrics to calculate the performance of an HPC system compared to its theoretical peak performance or speedup. The formula for calculating the Fraction of peak and Efficiency is as follows:

$$F_{peak} = \frac{R_{actual}}{R_{peak}} \times\ 100\% \tag{3}$$

$$Eff\ =\ \frac{S}{MPI\ rank} \times\ 100\% \tag{4}$$

In this study, we measure the speedup metric for both benchmarks and add the FLOP/s metric for HPCG benchmarks. Both will be compared with Mahameru's theoretical performance to calculate the fraction of peak and efficiency of the Mahameru HPC system.

3. RESULT AND DISCUSSION

3.1. NPB Result

The average speedup per core/thread from the NPB benchmark is shown in Figure 2 (left), where the blue bars show the average speedup using MPI and the red bars show the average speedup using OpenMP. The result indicates that OpenMP consistently achieves higher average speedups per thread compared to MPI across all benchmarks. This outcome suggests that OpenMP, which is designed for shared memory parallelism, benefits from the low to no communication overhead and closer memory access in MAHAMERU BRIN HPC. On the other hand, MPI, which is optimized for distributed memory systems, demonstrates lower speedups, likely due to communication overhead between processes. Benchmarks such as EP, FT, and BT show significant differences in speedup between the two paradigms, emphasizing the efficiency of OpenMP for workloads with a high degree of shared data and reduced communication demands. The SP kernel cannot be executed in the MPI paradigm in MAHAMERU BRIN HPC due to synchronization overheads and strong data dependencies.

Figure 2 (right) illustrates that all kernels and modules with class D in the NPB-OpenMP exhibit varying degrees of speedup. Generally, the memory capacity (including main memory and cache) is sufficient to support the increasing computational demands. As expected, no single kernel or module attained an ideal speedup or perfect linear speedup, which is the maximum theoretical value for a parallel algorithm [22]. Apart from the Conjugate Gradient (CG) kernel, all other kernels and modules demonstrated linear speedup, although the rate of the increase in speedup was still much lower than the increase in the number of processors. Among the five computational kernels, the EP kernel achieved the

p-ISSN: 2528-1682
e-ISSN: 2527-9165

highest speedup. This result is expected, as EP represents a set of independent tasks that do not require communication between them [23], as shown in Figure 3b for the MPI version. Among the pseudo-application modules, the LU module also demonstrates minimal communication, resulting in the highest speedup, which is slightly greater than that of EP. In contrast, the Multigrid (MG) kernel achieved the least speedup and performed similarly to IS.

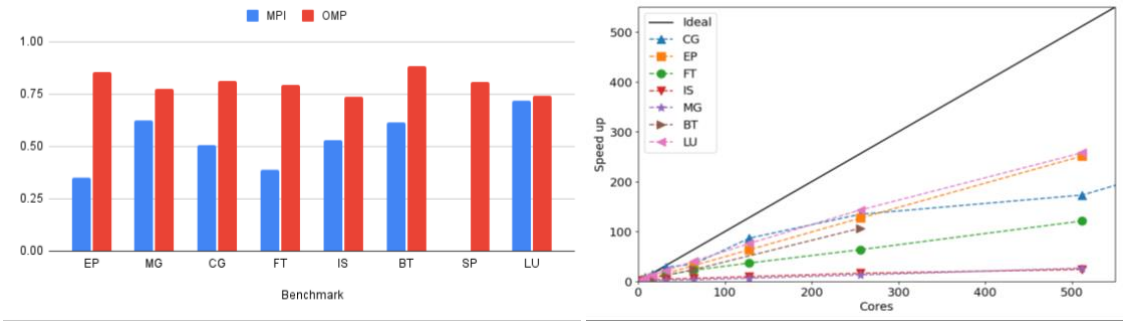

Figure 2. The average speed up per core/thread of NPB benchmark using MPI and OpenMP (left). The speed up of each kernel and module in the NPB-OpenMP benchmark with class D (right).

The execution time composition for each process in the kernels and modules of NPB-MPI is illustrated in Figure 3, showing that in most NPB algorithms, the ratio of computation to communication time decreases as the number of processors increases. For example, in the IS and CG kernels, communication time significantly increases to more than 50% of the total time when using many processors (128 for IS and 512 for CG), see Figure 3a-3c. According to [24], these two kernels are characterized by intensive memory communications and complex data dependencies, which are crucial for evaluating parallel programming frameworks. The CG benchmark involves communication patterns that are prevalent in the numerical solution of partial differential equations. This makes it a representative workload for modern applications that rely heavily on memory and network performance [25]. In contrast, the ratio of communication time in the EP kernel, as shown in Figure 3b appears very minimal compared to other kernels. This is because each task in an EP benchmark can be executed independently, without requiring data from other tasks. This independence reduces the need for communication, making the EP algorithm ideal for parallel execution. As noted in [26], the EP benchmark serves as a prime example of tasks designed to be executed with minimal communication, highlighting the efficiency of such benchmarks in parallel computing environments.

The MG and FT benchmarks require a power-of-two number of processors. In the MG benchmark, the communication time clearly increases with a larger number of processors, as illustrated in Figure 3d. In contrast, the FT benchmark achieves the highest communication efficiency when using the maximum number of processors in a single socket (32 processors), see Figure 3e. Communication falls off with inter-sockets or internodes configurations of processors as the network becomes saturated.

In the pseudo-application modules, the use of a number of processors is prescribed. The BT (Block Tridiagonal solver) and SP (Scalar Pentadiagonal solver) benchmark requires a square number of processors (e.g., 1, 4, 16) to solve three sets of uncoupled systems of equations in three directions (*X*, *Y*, and *Z*). The BT benchmark solves a block tridiagonal system with 5x5 blocks, and the SP benchmark solves a scalar pentadiagonal system. These two kernels are appropriate to benchmark high-bandwidth networks because of their heavy reliance on load balance and communication granularity. Our benchmark shows that the communication time ratio from the overall computation time (*xcomm*, *ycomm*, and *zcomm*) increases significantly when using more than 1 processor for the BT benchmark (Figure 3f) while the SP benchmark (Figure 3g) also increases, but not as significantly as the BT benchmark. The use of 64 or 256 processors in the BT or SP kernel demonstrates a slightly higher percentage of computation/processing time compared to using 16 processors. It is important to note that this percentage reflects computation time as a portion of the total time consumed, but it does not

imply that the actual computation time is longer. As presented in Figure 2 (left), the speed-up for SP and BT is quite significant.

The LU (Lower-Upper Gauss-Seidel solver) benchmark requires processors to count power-of-two numbers (e.g., 1, 2, 4, 8) to solve linear equations using the Symmetric Successive Over-Relaxation (SSOR) procedure. This benchmark is chosen for its sensitivity to the small message communication performance of an MPI implementation. Our benchmark shows that the communication time ratio from overall computation time (*lcomm* and *ucomm*) and the message communication exchange ratio from overall computation time (*exch*) increase according to the number of processors used. The result is as expected from this benchmark because the increase in the number of processors means that the messages sent to communicate also increase.

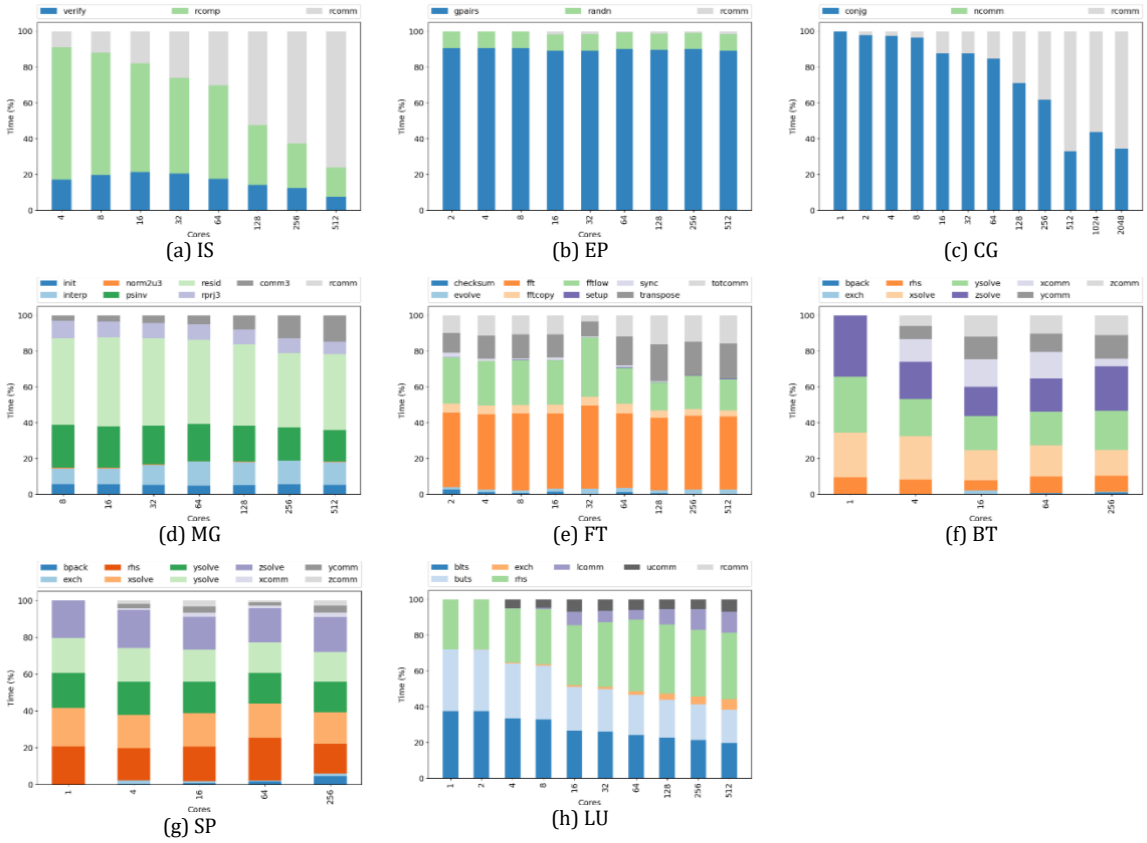

Figure 3. Time percentage of each process in each kernel and module in the NPB-MPI benchmark with class D.

3.2. HPCG Benchmark Result

Table 6 presents the HPCG benchmark results based on experiments using 1 - 32 nodes as shown in Table 4. The HPCG benchmark on the MAHAMERU HPC achieves a performance of TFlop/s reaching up to 1.13882 (when utilizing 32 nodes), which corresponds to a fraction of peak of 0.87%. When compared to other supercomputer clusters, Mahameru appears to be in competition with Ares from Cyfronet, Poland, which has a peak fraction of 0.9%.

Table 6. Results of HPCG benchmark.

Node	MPI Rank	#Eqs (+E3)	Mem(GB)	GFLOP/s	Speed up	Efficiency
1	1	4,096	2.93	1.54503	1	100.00
1	2	8,192	5.86	3.09279	1.9998	99.9
1	4	16,384	11.71	5.9186	3.8283	95.71
1	8	32,768	23.43	9.411	6.1039	76.30
1	16	65,536	46.86	16.814	10.9140	68.21
1	32	131,072	93.73	27.1893	17.6795	55.25
1	64	262,144	187.46	33.4688	21.8248	34.10
2	128	524,288	374.91	69.7274	45.4415	35.50
4	256	1,048,576	749.83	129.11	84.2006	32.89
8	512	2,097,152	1,499.66	286.109	186.4271	36.41

p-ISSN: 2528-1682
e-ISSN: 2527-9165

Node	MPI Rank	#Eqs (+E3)	Mem(GB)	GFLOP/s	Speed up	Efficiency
16	1024	4,194,304	2,999.31	569.669	371.1479	36.24
32	2048	8,388,608	5,998.62	1138.82	741.4653	36.20

A more detailed comparison with some other Top 500 supercomputers can be found in Table 7. Table 6 also indicates that as the number of processors (MPI ranks) increases, the speedup also rises. Consequently, the efficiency of HPCG gradually decreases due to communication within the processors. In the case of a single socket (with MPI ranks up to 32), the efficiency drops to 55%. Meanwhile, the intersocket utilization within a single node can decrease by up to 34%, which is somewhat comparable to the utilization of multiple nodes that achieve an efficiency of 36%. This indicates that the intersocket communication is not significantly different from the internodes communication. Due to Mahameru's status as a shared national resource, experiments were limited to 32 nodes to avoid disrupting active users. However, based on the trend observed from 1 to 32 nodes, Figure 4 demonstrates the estimation of the peak performance utilizing the full capacity of Mahameru (92 nodes) which would likely yield a peak performance of approximately 2.5% of the system's theoretical peak. While this is only a projection, it provides a conservative estimate of Mahameru's capability at scale. Future work will aim to validate this estimation during system maintenance windows or exclusive benchmarking periods.

Table 7. Performance of some Top 500 supercomputer clusters.

No	Top 500 Rank	Computer/Site	Cores	HPCG (PFLOP/s)	Fraction of Peak
1	4	Supercomputer Fugaku, RIKEN Center for Computational Science, Japan	7,630,848	16.000	3.0%
2	1	Frontier, DOE/SC/Oak Ridge National Laboratory, USA	8,699,904	14.050	0.8%
3	2	Aurora, DOSE/SC/Argonne National Laboratory, USA	9,264,128	5.613	0.3%
4	354	SNL/NNSA CTS-1 Attaway, Sandia National Laboratories, USA	52,920	0.039	1.0%
5	471	CEA-HFi, Commissariat a l'Energie Atomique (CEA), France	73,728	0.033	1.1%
6	442	Ares, Cyfronet, Poland	37,824	0.031	0.9%

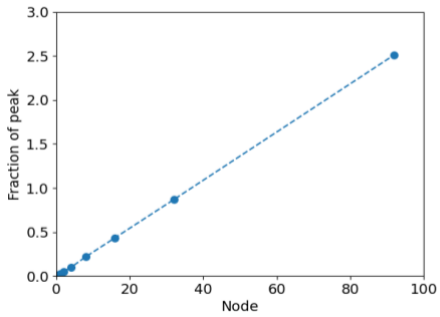

Figure 4. Ratio of peak performance of HPCG with the theoretical peak of Mahameru versus number of nodes.

3.3. Discussion

Deploying the Mahameru HPC system directly into production without conducting benchmarks presents a significant issue. Consequently, inconsistent performance has emerged across various workloads. Certain applications are underperforming compared to the previous generation, and resource allocation seems inefficient. Without benchmarking, system administrators are unable to determine whether the issues related to hardware configuration, interconnect performance, or software inefficiencies. This study contributes to providing benchmarking results for the HPC system based on the NPB and HPCG benchmarks. The NPB evaluates key performance metrics of the processor, memory,

and communication subsystems for various classes of computational workloads, while HPCG simulates performance of real-world scientific applications, especially memory-bound and communication-heavy codes.

The results from the NPB and HPCG benchmarks provide valuable insights into the performance characteristics and scalability of the Mahameru HPC system. The NPB benchmark results demonstrate the different behaviors of two parallelism paradigms, MPI and OpenMP parallelization approaches. The lower speedup of MPI compared to OpenMP can primarily be attributed to the communication overhead associated with message passing, particularly for the tested problem sizes and node counts. This highlights a well-known trade-off in HPC: while OpenMP can efficiently exploit shared-memory parallelism within a single node, MPI is essential for scaling across multiple nodes, where communication latency becomes a limiting factor. Despite these drawbacks, MPI remains critical for large-scale distributed applications, where the ability to coordinate work across many nodes outweighs the overhead costs. So, for the Mahameru cluster configuration, OpenMP emerges as the more efficient parallel programming model for maximizing core and thread utilization, especially for benchmarks with communication-intensive workloads. This also indicates the lack of visibility into communication bottlenecks across nodes.

The HPCG benchmark results further support this understanding. Although empirical testing was limited to 32 nodes due to resources sharing, the observed performance trend suggests consistent scaling characteristics. However, direct experimentation at the full 92-node capacity would provide stronger validation and remain an important next step for system evaluation. The initial result shows that the Mahameru system's performance is competitive with the lower tier of the Top 500 supercomputers. This outcome emphasizes the importance of memory bandwidth and latency as bottlenecks rather than raw computing capability alone. An interesting finding from both benchmarks is that hyperthreading offers limited benefits for algorithms based on conjugate gradient. These algorithms rely heavily on the presence of physical cores to maintain sustained performance. This indicates that future system configurations and scheduling policies on Mahameru should prioritize the utilization of physical cores over hyperthreading for similar workloads to maximize efficiency. This is particularly relevant for applications such as computational fluid dynamics and other scientific simulations that depend on conjugate gradient methods.

4. CONCLUSION

This study was motivated by a critical gap in the deployment of the Mahameru HPC system: the absence of formal benchmarking. This could limit both the effective use of Mahameru's resources and the confidence of its research community in the system's capabilities. To address this, we conducted a detailed performance evaluation of various parallel algorithms using the NAS Parallel Benchmarks (NPB) and the HPCG benchmark suite. The NPB results revealed that OpenMP consistently outperformed MPI in terms of speedup for most benchmarks, largely due to lower communication overhead and more efficient thread-level execution. However, MPI remains essential for large-scale distributed computing, and our results suggest that hybrid parallel programming models (MPI + OpenMP) are best suited to Mahameru's architecture for balancing efficiency and scalability, such as in handling large-scale and communication-intensive tasks. Using the HPCG benchmark, we observed that Mahameru achieves approximately 2.5% of its theoretical peak performance with 32 nodes, placing it within the lower tier of systems on the Top500 list. This demonstrates that, while the system is not yet fully optimized, it has the potential to support complex, real-world scientific applications such as CFD simulations that rely on conjugate gradient methods. Additionally, the study found that hyperthreading offers minimal benefit for conjugate gradient-based workloads, reinforcing the importance of prioritizing physical cores in job scheduling and system tuning.

In summary, this work fills the initial benchmarking gap by delivering a clear performance baseline, identifying architectural strengths and weaknesses, and offering practical configuration strategies. While these findings provide strong guidance for CPU and communication-bound workloads, future research should incorporate I/O performance analysis, potentially using the HPC Challenge benchmark suite. This addition would provide a comprehensive system performance profile, enhancing our understanding of the file system performance, I/O throughput, and latency. In practical applications, benchmarking at application level, such as with OpenFOAM (CFD) and WRF (weather modeling), can guide users in selecting optimal parallelization strategies for their specific workloads.

ACKNOWLEDGEMENTS

This publication was supported by Telkom University. We thank our colleagues from the Research Center for Computing and Mahameru BRIN HPC's administrator team who provided insight and expertise that greatly assisted the research, although they may not agree with all the interpretations/conclusions of this paper.

REFERENCES

[1] G. Xie and Y.-H. Xiao, "How to Benchmark Supercomputers," in *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, Guiyang, China: IEEE, Aug. 2015, pp. 364–367. doi: 10.1109/dcabes.2015.98.

[2] E. Strohmaier and H. Shan, "Apex-Map: A Synthetic Scalable Benchmark Probe to Explore Data Access Performance on Highly Parallel Systems," in *Euro-Par 2005 Parallel Processing*, J. C. Cunha and P. D. Medeiros, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 114–123.

[3] S. Faulk, J. Gustafson, P. Johnson, A. Porter, W. Tichy, and L. Votta, "Measuring High Performance Computing Productivity," *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, pp. 459–473, Nov. 2004, doi: 10.1177/1094342004048539.

[4] M. Hao, W. Zhang, Y. Zhang, M. Snir, and L. T. Yang, "Automatic generation of benchmarks for I/O-intensive parallel applications," *Journal of Parallel and Distributed Computing*, vol. 124, pp. 1–13, Feb. 2019, doi: 10.1016/j.jpdc.2018.10.004.

[5] Y. Liu *et al.*, "623 Tflop/s HPCG run on Tianhe-2: Leveraging millions of hybrid cores," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 39–54, Feb. 2016, doi: 10.1177/1094342015616266.

[6] M. Elshambakey, A. I. Maiyza, M. S. Kashkoush, G. M. Fathy, and H. A. Hassan, "The Egyptian national HPC grid (EN-HPCG): open-source Slurm implementation from cluster to grid approach," *J Supercomput*, vol. 80, no. 12, pp. 16795–16823, Aug. 2024, doi: 10.1007/s11227-024-06041-9.

[7] S. Varrette, H. Cartiaux, S. Peter, E. Kieffer, T. Valette, and A. Olloh, "Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0," in *2022 6th High Performance Computing and Cluster Technologies Conference (HPCCT)*, Fuzhou China: ACM, Jul. 2022. doi: 10.1145/3560442.3560445.

[8] D. Zivanovic *et al.*, "Main Memory in HPC: Do We Need More or Could We Live with Less?," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, pp. 1–26, Mar. 2017, doi: 10.1145/3023362.

[9] M. Sato, Y. Kodama, M. Tsuji, and T. Odajima, "Co-Design and System for the Supercomputer 'Fugaku,'" *IEEE Micro*, vol. 42, no. 2, pp. 26–34, Mar. 2022, doi: 10.1109/mm.2021.3136882.

[10] T. Aoyama, I. Kanamori, K. Kanaya, H. Matsufuru, and Y. Namekawa, "Bridge++ 2.0: Benchmark results on supercomputer Fugaku," 2023, doi: 10.48550/ARXIV.2303.05883.

[11] "Performance of the Supercomputer Fugaku for Breadth-First Search in Graph500 Benchmark," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2021, pp. 372–390. doi: 10.1007/978-3-030-78713-4_20.

[12] "Optimized Implementation of the HPCG Benchmark on Reconfigurable Hardware," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2021, pp. 616–630. doi: 10.1007/978-3-030-85665-6_38.

[13] "Understanding HPC Benchmark Performance on Intel Broadwell and Cascade Lake Processors," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2020, pp. 412–433. doi: 10.1007/978-3-030-50743-5_21.

[14] A. Fuchs, J. Squar, and M. Kuhn, "Ensemble-Based System Benchmarking for HPC," in *2024 23rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Chur, Switzerland: IEEE, Jul. 2024, pp. 1–8. doi: 10.1109/ispdc62236.2024.10705405.

[15] "An I/O Analysis of HPC Workloads on CephFS and Lustre," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2019, pp. 300–316. doi: 10.1007/978-3-030-34356-9_24.

[16] D. G. Chester, S. A. Wright, and S. A. Jarvis, "Understanding Communication Patterns in HPCG," *Electronic Notes in Theoretical Computer Science*, vol. 340, pp. 55–65, Oct. 2018, doi: 10.1016/j.entcs.2018.09.005.

[17] D. H. Bailey *et al.*, "The Nas Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, Sep. 1991, doi: 10.1177/109434209100500306.

[18] J. Dongarra, M. A. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 3–10, Feb. 2016, doi: 10.1177/1094342015593158.

[19] "SLURM: Simple Linux Utility for Resource Management," in *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. doi: 10.1007/10968987_3.

[20] L. V. Kalé *et al.*, "NAS Parallel Benchmarks," in *Encyclopedia of Parallel Computing*, D. Padua, Ed., Boston, MA: Springer US, 2011, pp. 1254–1259. doi: 10.1007/978-0-387-09766-4_133.

[21] D. A. Mallon, G. L. Taboada, J. Tourino, and R. Doallo, "NPB-MPJ: NAS Parallel Benchmarks Implementation for Message-Passing in Java," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Weimar: IEEE, Feb. 2009, pp. 181–190. doi: 10.1109/PDP.2009.59.

[22] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, "A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures," *Commun. comput. phys.*, vol. 15, no. 2, pp. 285–329, Feb. 2014, doi: 10.4208/cicp.110113.010813a.

[23] J.-C. Régin, M. Rezgui, and A. Malapert, "Embarrassingly Parallel Search," in *Principles and Practice of Constraint Programming*, vol. 8124, C. Schulte, Ed., in Lecture Notes in Computer Science, vol. 8124. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 596–610. doi: 10.1007/978-3-642-40627-0_45.
[24] J. Löff *et al.*, "The NAS Parallel Benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures," *Future Generation Computer Systems*, vol. 125, pp. 743–757, Dec. 2021, doi: 10.1016/j.future.2021.07.021.
[25] "A CUDA Implementation of the High Performance Conjugate Gradient Benchmark," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2015, pp. 68–84. doi: 10.1007/978-3-319-17248-4_4.
[26] H. Lu, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel, "Quantifying the Performance Differences between PVM and TreadMarks," *Journal of Parallel and Distributed Computing*, vol. 43, no. 2, pp. 65–78, Jun. 1997, doi: 10.1006/jpdc.1997.1332.