International Journal of Computer Sciences and Engineering
Vol.**13**, Issue.**3**, pp.**01-08**, March **2025**
ISSN: 2347-2693 (Online)
Available online at: www.ijcseonline.org

ISSN: 2347-2693 (E)

Research Article

The High-Performance Linpack (HPL) Evaluation on MIHIR High Performance Computing Facility at NCMRWF

Shivali Gangwar[1*], **B. Athiyaman**[2], **Preveen Kumar D.**[3]

[1,2,3]National Centre for Medium Range Weather Forecasting, Noida, India

**Corresponding Author:*

Received: 18/Jan/2025; **Accepted**: 20/Feb/2025; **Published**: 31/Mar/2025. **DOI**: https://doi.org/10.26438/ijcse/v13i3.18

Abstract: The National Centre for Medium Range and Weather Forecasting (NCMRWF) has, the MIHIR High Performance Computing (HPC) Facility with total computing capacity of 2.8 Petaflops to run Numerical Weather Prediction (NWP) models, enabling accurate and timely weather forecasting. These models need computations performed in PFLOPS (Peta Floating Point Operations Per Second). The HPC nodes are interconnected by the high speed, low latency Cray Aries network. High Performance Linpack (HPL) version 2.3 has been compiled and installed on the system for the study. The purpose of running HPL is to demonstrate the current computing performance of the HPC, and to assess the systems efficiency by analyzing the actual calculated performance (R_{max}) and the theoretical peak performance (R_{peak}, derived from system specifications). We present a performance evaluation of the HPL benchmark on MIHIR, conducting a detailed analysis of HPL parameters to optimize performance. The aim is to identify the best-optimized parameter values for MIHIR and determine the maximum achievable performance of the compute nodes, utilizing up to 300 nodes available for research.

Keywords: HPL, HPC, Aries interconnect, MIHIR, NCMRWF, PFLOPS

1. Introduction

The National Centre for Medium Range Weather Forecasting (NCMRWF) has a High-Performance Computing (HPC) facility named MIHIR, which has a computing capacity of 2.8 Petaflops. This HPC system is specifically designed to run Numerical Weather Prediction (NWP) models, enabling accurate and timely weather forecasting. The reliability of these forecasts is crucial for disaster preparedness, agriculture, aviation, and several other sectors that depend on precise meteorological information.

One of the key factors contributing to the robust performance of this facility is its high-speed interconnection infrastructure. The compute nodes within the system are connected through a Cray Aries interconnect and utilize a dragonfly topology, which provides both high speed and low latency communication. Efficient interconnects and computational resources directly impact the accuracy and resolution of weather models, ensuring that forecasts are both timely and reliable.

Given the critical role of HPC in weather prediction, it is essential to continuously evaluate and optimize system performance. NWP models running on this system perform calculations in PFLOPS (Peta Floating Operations Per Second), demonstrating the immense computational power required for advanced meteorological simulations. To assess and ensure the efficiency of MIHIR, benchmarking tests are conducted, including the use of High-Performance Linpack (HPL) to measure actual computational performance (R_{max}) as compared to the theoretical peak performance (R_{peak}). This benchmarking process not only evaluates the current performance but also helps identify opportunities for optimizing system configurations, improving efficiency, and enhancing forecasting capabilities.

This study is significant because it provides a detailed analysis of MIHIR's computational capabilities, highlighting its strengths and identifying areas for potential enhancement. Understanding the performance limitations of HPC in the context of NWP is crucial for ensuring that weather forecasting models remain accurate and efficient. By benchmarking MIHIR's capabilities, this study contributes to the ongoing efforts to improve HPC infrastructure for meteorological applications, ultimately supporting better preparedness for extreme weather events and improving the overall quality of weather predictions.

2. Related Work

High-Performance Linpack (HPL) has been widely used to evaluate the computational capability of supercomputers. Several studies have focused on optimizing HPL performance through parameter tuning, hardware configurations, and software optimizations.

2.1 HPL Performance Optimization

Previous research has highlighted the importance of selecting optimal problem sizes (N), block sizes (NB), and process grid configurations (P × Q) for achieving maximum performance [11, 12]. Studies have shown that improper selection of these parameters can lead to significant performance degradation [5,6].

2.2 Impact of BLAS Libraries on HPL

The choice of BLAS implementation significantly affects HPL performance. Research comparing different libraries—such as ATLAS, OpenBLAS, Intel MKL, and Cray’s scientific libraries—has demonstrated variations in efficiency based on system architecture and workload distribution [5,9].

2.3 Parallelization and Scalability in HPC Systems

Scalability studies on various HPC systems have shown that increasing the number of nodes beyond a certain point result in diminishing performance gains due to inter-node communication overhead. Researchers have investigated MPI [10] communication strategies and network topologies to mitigate these bottlenecks [9].

2.4 HPL on Emerging Architectures

With the rise of heterogeneous computing, recent work has explored running HPL on AMD EPYC, ARM-based processors, and GPU-accelerated systems [12,7]. These studies indicate that different architectures require tailored optimization strategies to achieve peak efficiency.

These studies provide valuable insights into HPL benchmarking, parameter tuning, and HPC optimization strategies, forming the foundation for the present work on optimizing MIHIR’s performance.

3. Experiment Design and Methodology

3.1 MIHIR: High Performance Computing

The MIHIR HPC system at NCMRWF has 2322 compute nodes interconnected via a high-performance network. Each compute node is a dual socket with an Intel Xeon Broadwell processor. The compute node runs a version of Operating System optimized for running batch workloads. The software set running on the nodes includes the SUSE Linux Enterprise Server 12 SP2 operating system, Cray Message Passing Interface (MPI) based on MPICH3 source from ANL, Cray's libSci scientific library and an Intel MKL. The technical specifications of MIHIR HPC are in Table 1.

Table 1. Technical Specifications of MIHIR HPC

Components	Details
Processor Type	Intel Xeon Broadwell E5-2695
Number of Compute Nodes	2322
Total Number of Cores	83592
Interconnect	Cray Aries with Dragon Fly
Node Processor Cores	2 x (2.1 GHz, 36-Core)
Memory Size per Node	128 GB
Memory Type	DDR4
Node Cache Size	2 x 24 MB
Turbo Boost	OFF
Usage of Hyper Threading	ON
Peak Compute Power	2806 TF
Effective Distributed Memory	297TB

3.2 HPL Benchmarking Method

The Linpack library is a set of FORTRAN subroutines that are developed for analysing and solving linear equations [11]. This numerical library is used widely for solving systems of linear equation of the form:

$$A \cdot x = b \tag{1}$$

Equation (1) shows a linear system where A represents a dense matrix that can be decomposed into product of matrices, having all non-zero elements, x is the unknown vector, while vector b on the right-hand side is a known vector. The library offers efficient methods for manipulating and solving this linear equation problem by performing matrix factorization of A.

The Linpack library was originally derived from an auxiliary program found in the Linpack User's Guide publication [11]. It has evolved into a well-structured numerical code capable of solving dense matrices using double-precision arithmetic. This suite has now been adopted as a benchmark to efficiently evaluate performance on supercomputers.

HPL [6,7] is a benchmarking software that has been utilized by various organizations as the benchmark suite to gauge the peak performance of their HPC facilities. It is widely adopted and used as a standard for obtaining maximum performance measurements in all TOP500 [8] HPC facilities. It is freely available and highly portable.

To evaluate the accuracy of solutions obtained as well as the computation time the HPL software package has been included with testing and timing programs. Achieving optimal performance with this software hinges on various factors specific to the system under evaluation. The benchmarks conducted using this software result in a measurement computed in Floating-Point Operations per Second (FLOPS).

The Message Passing Interface (MPI) [10] has been implemented in HPL software that helps it to run on HPC systems, which are distributed and need internode communication. The MPI library specification will facilitate message passing thus enabling internode communication.

Configuring the HPL software package requires the inclusion of the Basic Linear Algebra Subprograms (BLAS) implementation [9]. BLAS comprises application programming routines that facilitate basic vector and matrix multiplication operations.

Any of the following BLAS software can be utilized for compiling HPL. The Cray Scientific Library (CSL), developed by CRAY, offers a wide range of highly optimized BLAS routines. Additionally, the Intel® Math Kernel Library (Intel® MKL) and AMD Core Math Library (ACML) [5,1] are optimized and highly threaded math libraries that include BLAS routines. Whereas for this paper, the Cray Scientific Library (CSL) has been selected to achieve fair and satisfactory results on MIHIR HPC.

4. Methodology

The initial stage involves installation, and configuring the HPL. Appropriate compilers for compiling codes are identified. Following this, the suitable MPI is selected for facilitating internode communication.

4.1 Building the HPL benchmark

Cray compilers, CCE version 8.6.3, cray-mpich version 7.6.3 and cray scientific library, cray-libsci 17.09.1 has been used.

4.2 HPL main parameters

To optimize the results, we focus on fine-tuning the HPL.dat [6,7] file within HPL. This file contains 17 parameters, of which 7 are used for optimizing the result, in this paper, these are Problem size (N), Number of Process Grids (P x Q), Block Size (NB), Broadcast Parameter (BCAST), Lookahead depth (DEPTH), Panel Factorization (PFACT), and Reverse Panel Factorization (RFACT) [7].

Problem size (N) in HPL is one of the pivotal parameters used to fully utilize the compute node memory. An important consideration is to set the problem size such that it does not exceed the available memory, which can result in getting swapped out of memory i.e. OOM. Only 93% of memory per compute node has been used to avoid memory swap and degraded performance. After calculation and testing, the ideal problem size value for MIHIR HPC comes out to be 121900.

The processor grid refers to parameters defining the dimensions of the process grid, typically represented as P and Q. Here P represents the number of rows of the process and Q represents the number of columns of the process. To determine the optimal processor grid, it's common to use values where Q is greater than P. Thus, configuring P as 4 and Q as 9 seems to be an optimal decision. On MIHIR HPC setting P to 4 and Q to 9 resulted in optimal performance of the system.

Determining the optimal block size in HPL is not straightforward, as it serves dual purposes: distributing data effectively across processors and determining computational granularity.

A smaller block size improves load balance in terms of data distribution, but it can significantly reduce computational performance due to limited data reuse at higher memory hierarchy levels. This can lead to an increase in message count. Optimal block sizes typically fall within 32 to 256 intervals, with the ideal value depending on the system's computation-to-communication performance ratio. For Intel Xeon Broadwell E5-2695, the recommended block size is 192. However, multiple runs on different values on MIHIR HPC revealed that 192 is indeed the optimal block size for this system.

The broadcast algorithm is used in HPL to distribute factorized panel columns from one panel to another process panel. HPL has the following broadcast algorithms available, such as Increasing-ring, Increasing-ring (modified), Increasing-2-ring, Increasing-2-ring (modified), long (bandwidth reducing), and long (bandwidth reducing modified). The Increasing-ring (modified) algorithm is recommended for MIHIR HPC due to its better performance.

Implementing this modified algorithm in the broadcast function on MIHIR HPC has resulted in a significant improvement, achieving a finer result of 1.09 Tflops per compute node.

5. Results and Discussion

In this study, we evaluated the performance of the High-Performance Linpack (HPL) benchmark on MIHIR, a high-performance computing (HPC) system. The goal was to analyze the impact of various HPL tuning parameters on system performance, optimize them for maximum efficiency, and assess the compute capability of MIHIR when scaling up to 300 compute nodes.

The primary parameters examined include process grid configuration (P × Q), problem size (N), block size (NB), panel factorization (PFACT), recursive panel factorization (RFACT), broadcast algorithm (BCAST), and lookahead depth (DEPTH). The results demonstrate how each parameter influences performance in terms of execution time, GFLOPS, and efficiency.

5.1 Process grids, P x Q

The number of process grids will determine the optimal value for the number of nodes that have been used for HPL execution. The arrangement of P and Q affects the processing speed and the time needed for problem solving. The test results obtained are shown in Table 2 and Figure 1 indicate that as the number of nodes increases, the execution time stabilizes after an initial improvement in performance. Speedup and efficiency values suggest that optimal resource utilization is achieved when all processors are effectively engaged.

Table 2. Number of Nodes: Processing Speed and Efficiency

Nodes	Time (s)	Gflops	Speedup	Efficiency (%)
1	1106.39	1091.50	1	90.228
2	1114.39	1084.6	0.9936784	89.658
4	1113.27	1084.70	0.99377	89.666
6	1121.78	1076.5	0.9862574	88.989
8	1117.58	1080.60	0.9900137	89.327
10	1116.03	1082.10	0.991388	89.451
12	1117.03	1081.10	0.9904718	89.369
18	1122.05	1076.30	0.9860742	88.972
22	1128.77	1069.80	0.9801191	88.435
24	1180.63	1068.90	0.9792945	88.360
28	1128.79	1069.80	0.9801191	88.435
30	1128.04	1070.50	0.9807604	88.493
36	1122.02	1076.30	0.9860742	88.972
42	1126.67	1071.80	0.9819514	88.600
54	1125.21	1073.20	0.9832341	88.716
72	1117.72	1080.40	0.9898305	89.311

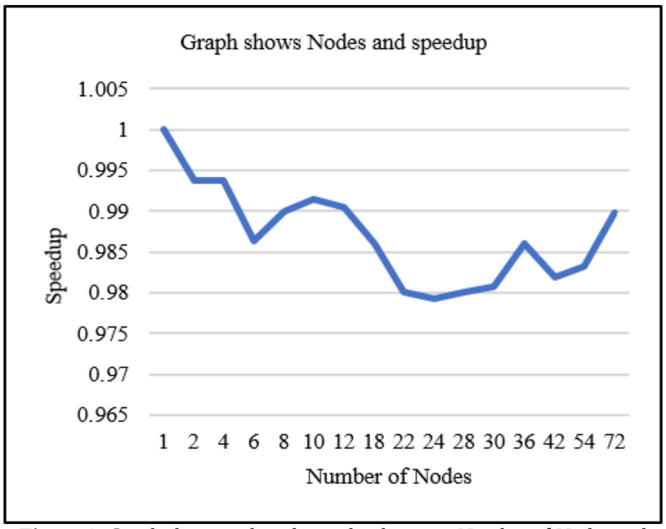

Figure 1. Graph showing the relationship between Number of Nodes and Speedup

According to the results obtained processing speed increases until all processors are fully utilized, after which the speedup stabilizes and remains constant.

5.2 Problem Size, N

The problem size refers to the size of the dense matrix that the benchmark will solve. The size of the matrix determines the computational load, larger the problem size, better reflects the true power of the system.

The matrix size determines the amount of memory required for the computation. As N increases, the memory usage grows as N^2, since the N x N matrix needs to be stored. For supercomputers, the problem size is chosen to ensure maximum utilization of memory and processing power. The goal is to maximize GFLOPS (billions of floating-point operations per second). The below results demonstrate the impact of problem size, N on processing speed in solving the problem

Table 3. Problem Size: Processing Speed and Time Taken

Problem Size, N	Time (s)	Gflops	Speedup
5000	0.31	273.07	1
10000	1.22	546.82	2.0024902
15000	3.44	654.95	2.40
20000	9.45	564.31	2.06653972
50000	84.7	983.89	3.60306881
100000	640.16	1041.40	3.81367415
110000	851.45	1042.20	3.8166038
115000	998.22	1015.70	3.71955909
117000	1017.55	1049.30	3.84260446
118000	1021.91	1071.90	3.92536712
120000	1079.05	1067.60	3.90962024
121000	1090.95	1082.60	3.96455121
121900	1112.95	1085.1	3.97370638

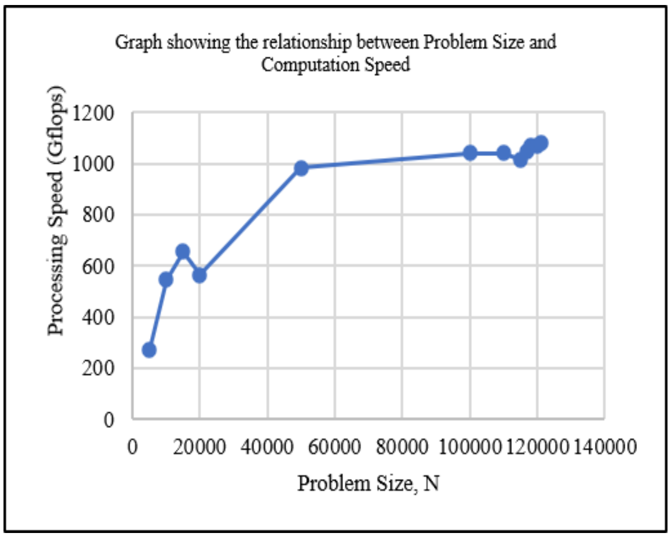

Figure 2. Graph showing the relationship between Problem Size and Processing Speed

A larger problem size (high N) increases the computational workload to find the solution, thus leading to higher computational speeds as shown in Figure 2. However, the increase in processing speed will plateau once all the processors are fully utilized in solving the problem, reaching a saturation point. At this point, the speed increase stabilizes as the processing power reaches its maximum effectiveness and efficiency. This behaviour underscores the importance of selecting an appropriate N to balance workload distribution and memory utilization.

5.3 Block Size, NB

This execution will demonstrate the impact of Block Size, NB on the processing speed of the compute node and the time taken in problem solving as shown in Table 4. The NB [7] defines how data and computations are distributed across the processors in the system. It determines how submatrices are assigned to each processor. An optimal distribution can improve load balancing and reduce communication overhead between processor, leading to better performance. The results obtained from the execution are shown in Figure 3, indicate that a block size 192 shows the highest computation speed.

Table 4. Block Size: Processing Speed and Time Taken

Block size (NB)	Time (s)	Gflops
100	1218.51	991.05
125	1282.83	941.37
150	1228.57	982.94
192	1116.51	1081.60
200	1166.4	1035.30
210	1150.86	1049.30
224	1136.74	1062.30

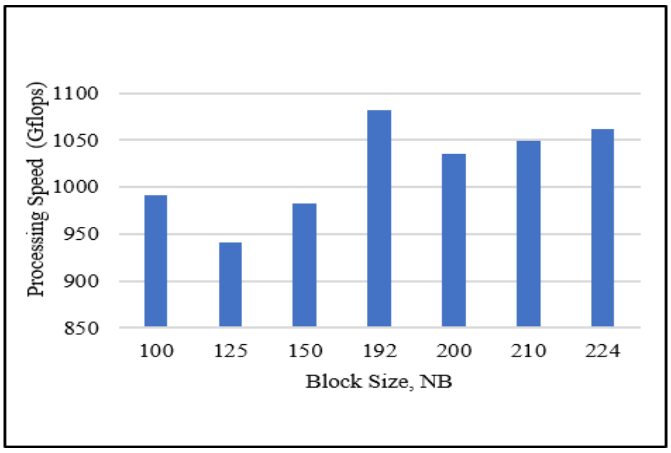

Figure 3. Graph showing the relationship between Block Size and Processing Speed

Performance degrades for smaller and larger block sizes, as inefficient data distribution increases computational delays and synchronization overhead. This confirms that choosing an appropriate NB is crucial for achieving peak efficiency in matrix operations.

5.4 Panel Factorization, PFACT

PFACT controls the type of factorization algorithm used by HPL. Factorization is the process of decomposing a matrix into simpler components (such as lower and upper triangular matrices). Selecting among these factorization methods involves trade-offs between computational efficiency and communication overhead. PFACT algorithm decomposes the matrix into panels or blocks and these panels are then factorized using the LU(Lower-Upper) decomposition technique. There are three main PFACT algorithms each with a different approach to solve computation problems. These are Left PFACT where panels are factorized from the left side of the matrix i.e. from left to right matrices are factorized, the Right PFACT where factorization is done from right to left and the Crout PFACT where factorization is performed from both left and right simultaneously. The results obtained during execution are shown in Table 5 and Figure 4. As per results obtained the Crout PFACT provides the best computational performance of the compute node. Left and Right PFACT algorithms exhibit slightly lower efficiency due to differences in panel decomposition strategies, impacting communication and memory access patterns.

Table 5. Panel Factorization: Processing speed and Time Take

Panel factorization	Time (s)	Gflops
Left	1145.15	1054.5
Crout	1112.65	1085.40
Right	1115.16	1082.9

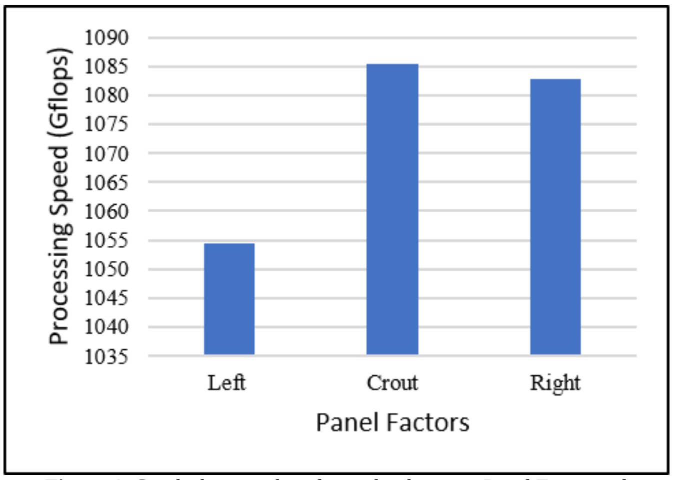

Figure 4. Graph showing the relationship between Panel Factor and Processing Speed

5.5 Recursive Panel Factorization, RFACT

In RFACT, the matrix is divided into submatrices or panels, which are then further subdivided until they reach a size suitable for direct factorization using algorithms like LU decomposition. This recursive approach is performed in parallel to improve computational efficiency.

The RFACT algorithms consist of three variants: the left-looking RFACT, the right-looking RFACT, and the Crout RFACT algorithms [10]. Figure 5, presents a graph illustrating the relationship between the RFACT and the processing speed, measured in Gflops (billion floating-point operations per second). The figure highlights the performance comparison among these algorithms, showing their computational efficiency under varying conditions.

The results presented in Table 6, indicate that the Crout algorithm when applied in the context of RFACT, enhances the computational performance of the compute node compared to the other algorithms analyzed. Also, the Crout algorithm's structure allows for more efficient utilization of computational resources, leading to improved processing speed and better overall performance. This advantage is particularly evident when compared to the left-looking and right-looking RFACT algorithms, making the Crout method more suitable for high-performance scenarios where maximizing compute efficiency is crucial. Crout RFACT efficiently utilizes computational resources by minimizing communication overhead and maximizing parallelism, making it the preferred choice for achieving higher GFLOPS.

Table 6. Recursive Panel Factorization: Processing speed and Time Taken

Recursive panel factorization	Time taken (s)	Gflops
Left	1148.95	1051.1
Crout	1112.65	1085.40
Right	1130.46	1068.2

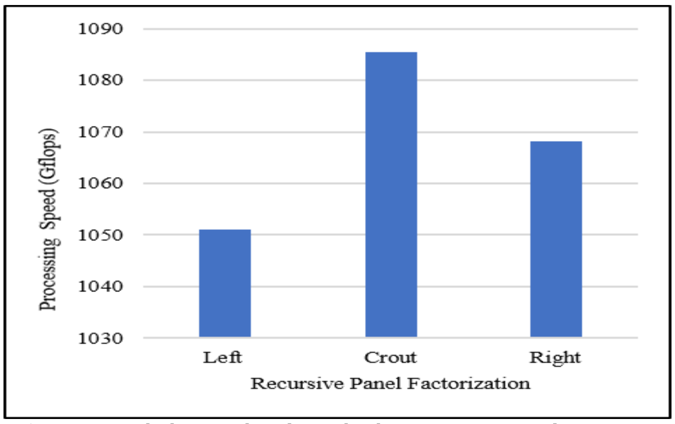

Figure 5. Graph showing the relationship between RFACT and Processing Speed

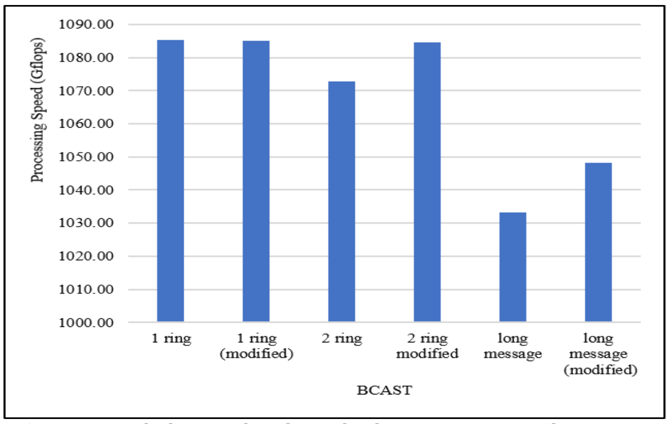

Figure 6. Graph showing the relationship between BCAST and Processing Speed

5.6 Broadcast Parameter, BCAST

The execution for the BCAST parameter aims to analyze the impact of the broadcast algorithms on processing speed and problem-solving time in HPC environment. The BCAST determines how data is efficiently distributed from one processor to another processors or nodes in an HPC.

The following broadcast algorithms are used in the study, the increasing-1-ring, which involves sending data in a linear chain or ring and each processor sends the data to the immediate neighbor continuing until all processors receive data; the increasing-1-ring (modified), it is a variation to increase-1-ring with optimizations and reduced communication overhead; the increasing-2-ring, where data is send to two neighbors simultaneously; the increasing-2-ring (modified), is the variation to increase-2-ring with optimizations and reduced communication overhead; the long (bandwidth reducing), where some techniques like data compression, message aggregation, or intelligent routing are being used to minimize bandwidth usage; and the long (bandwidth reducing modified), is an enhanced version of this algorithm with additional optimizations or adjustment.

The results from the execution, are presented in Table 7 and Figure 6, indicate that the modified increasing-2-ring algorithm enhances the computational performance of the compute node compared to the other algorithms evaluated. This improvement is evident in terms of both processing speed and overall efficiency. The modified approach optimizes communication and data handling within the node, allowing for better resource utilization and faster computation, making it the most effective option among the algorithms tested. Other algorithms, such as the long bandwidth-reducing methods, demonstrate lower performance due to increased data transmission latency.

Table 7. Broadcast: Processing Speed and Time Taken

Broadcast Algorithms	**Time (s)**	**Gflops**
Increasing-ring	1112.65	1085.40
Increasing-ring (modified)	1112.95	1085.10
Increasing 2 ring	1125.54	1072.90
Increasing 2-ring modified	1113.56	1084.50
long bandwidth reducing	1168.69	1033.30
long (bandwidth reducing modified)	1152.02	1048.30

5.7 Lookahead Depth

The Look Ahead Depth parameter in HPL determines how many levels of factorization trees are computed concurrently. It is a crucial concept related to pipelining and overlapping computations to improve performance. By reordering operations, the HPL can execute less efficient operations concurrently with more efficient ones, reducing idle times and improving overall performance. With a lookahead depth of 1, HPL can achieve some overlap between communication and computation, but not as much as higher depths. It will reduce idle processor time compared to no lookahead (depth 0). When the look ahead depth is set greater than zero, the benchmark precomputes and stores the panels being factorized in memory, allowing it to "look ahead". This approach consumes more memory but results in a performance boost by allowing more parallelism. As shown in Figure 7 and Table 8, MIHIR achieves better performance with a lookahead depth of 1. A depth of 0 results in lower GFLOPS due to reduced concurrency, while higher depths may introduce memory overhead without significant performance gains.

Table 8. Lookahead Depth: Processing Speed and Time Taken

Depth	**Time (s)**	**Gflops**
0	1117.29	1080.80
1	1113.56	1084.50

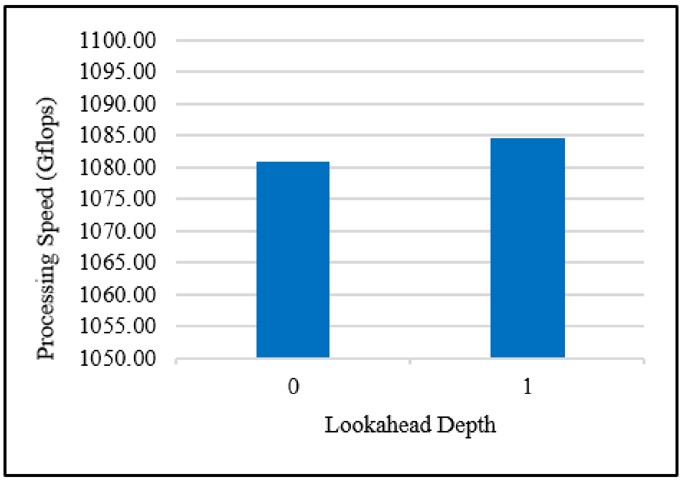

Figure 7. Graph showing the relationship between Depth and Processing Speed

5.8 Final Test Run based on Achieved Values

Based on the results obtained from execution, the HPL is configured with optimized parameters as follows:

Table 9. Optimized Parameters

Parameters	Optimized values
N	121900
NB	192
PFACT	Crout
RFACT	Crout
BCAST	3 (2-ring modified)
DEPTH	1

The formula used for calculating the Problem size is as shown in Figure 1.

$$N = \sqrt{\frac{(Memory\ Size\ per\ node \times 1024^3 \times Number\ of\ Nodes)}{8}} \times 0.93$$

$$N = \sqrt{\left(\frac{128 \times 1024^3 \times 1}{8}\right)} \times 0.93 \approx 121{,}900 \tag{2}$$

The important consideration is to set the problem size such that it does not exceed the available memory on the node, which can result in getting swapped out of memory i.e. OOM. Only 93% of memory per compute node has been used to avoid memory swap and degraded performance. After calculation and testing, the ideal problem size value for MIHIR HPC comes out to be 121900.

The R_{peak} is calculated based on system's specifications, whereas the Rmax is measured during the actual HPL execution. To calculate the theoretical peak, the rate of executing floating-point operations is needed. Manufacturers provide a figure for floating-point operations, which serves as an upper limit on performance. Thus, the R_{peak} is computed using equation (3), which takes into account the number of floating-point operations for addition and multiplication in full precision that can be performed within a specified time frame, typically the cycle time of the machine.

The Intel Xeon Broadwell E5-2695, 2.1 GHz processor can execute 8 double precision floating-point operations per cycle per core and 16 single precision floating point operations per cycle per core. The calculations for Rpeak are as follows:

$$R_{peak} = No.\ of\ Nodes \times No.\ of\ Cores\ per\ Node \times Speed\ per\ Core\ (GHz) \times Operations\ per\ Cycle \tag{3}$$

$$R_{peak}\ (Single\ compute\ node) = 1 \times 36 \times 2.1 \times 16 = 1.209\ Tflops \tag{4}$$

The highest possible result achievable is 1.209 Tflops per compute node as determined by equation (4).

The R_{max} obtained by running the HPL benchmark suite is measured in Tflops. In this paper, we achieved the Rmax of 1.0915 Tflops using problem size of 121900, a 4 x 9 processor grid, a block size of 192 and the 2-ring modified broadcast algorithm. The system efficiency can be determined by calculating the ratio of R_{max} to R_{peak}.

$$System\ Efficiency = \left(\frac{R_{max}}{R_{peak}}\right) \times 100 \tag{5}$$

$$Efficiency\ (Single\ compute\ node) = \left(\frac{1.091}{1.209}\right) \times 100 = 90.23\% \tag{6}$$

To determine the efficiency of the MIHIR HPC, equation (5) has been used. Hence the overall efficiency of MIHIR HPC comes out as 90.23%.

5. Conclusion and Future Scope

We presented the performance of the HPL benchmark on MIHIR HPC system, focusing on optimizing HPL tuning parameters to achieve the best possible computational efficiency. The goal was to identify the most optimized parameter values for MIHIR and determine the system's peak performance using up to 300 compute nodes.

The process grid configuration affects execution time and efficiency, with diminishing returns beyond an optimal node count. Problem size (N) must be carefully selected to maximize GFLOPS without exceeding memory limitations. Block size (NB) of 192 provides the best performance by balancing data distribution and computation. Crout panel factorization (PFACT) and recursive panel factorization (RFACT) yield superior results compared to Left and Right variants. The modified increasing-2-ring broadcast algorithm optimizes data transmission, reducing communication overhead. A lookahead depth of 1 enhances performance by effectively overlapping computations.

The HPL tuning parameters N, NB, P and Q are crucial for achieving better performance. P and Q must be selected ensuring that P<= Q. The 4 x 9 combination yielded the best results on MIHIR. The block size NB=192 has been used, which is the recommended value for Intel Xeon Broadwell E5-2695. The problem size N is set to define the biggest problem that would fit into the available memory.

For 300 nodes, the measured parallel efficiency comes out to be 90.22% with total maximum calculated performance for 300 nodes, R_{max}=328 TFLOP/s.

Among BLAS implementations, Cray Scientific Libraries performed the best, significantly outperforming Intel MKL and ATLAS, the latter showing the lowest efficiency. These findings confirm that optimal tuning and library selection are critical for maximizing HPC performance.

For future work, we aim to extend this study by optimizing other HPC architectures, such as AMD processors, and experimenting with GPU-accelerated HPL benchmarks. Additionally, refining MPI communication strategies and evaluating newer BLAS libraries could lead to further performance gains. Furthermore, we plan to explore the implementation of machine learning techniques to automatically optimize HPL parameters for any given HPC configuration.

Conflict of Interests

The authors have no relevant financial or non-financial, personal or other relationships with other people or organizations that could inappropriately influence, or be perceived to influence, their work. Otherwise, Authors declare that they do not have any conflict of interest.

Funding Source

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Author Contributions

All authors have contributed to the study conception and design. Installation, configuration and execution analysis were performed by Shivali Gangwar. B. Athiyaman and Preveen D has provided technical information about HPC infrastructure, to support in system along with guidance, revision, discussion, and finalization of the report. The first draft of the manuscript was written by Shivali Gangwar and all authors commented on previous versions of the manuscript. All authors have read and approved the final manuscript.

.

Acknowledgements

All authors appreciate the support and guidance from our colleagues at NCMRWF during this work. Special thanks to the Head, Dr. V.S. Prasad for his continuous encouragement and unwavering support.

References

[1] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, “HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers,” Innovative Computing Laboratory, University of Tennessee, pp.**1-10, 2018.**

[2] A. Smith, B. Johnson, "Performance Analysis of HPL on Intel's Icelake Architecture," *Proceedings of the International Conference on High-Performance Computing Systems*, pp.**123-130, 2021.**

[3] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, “Basic Linear Algebra Subprograms for Fortran usage,” ACM Transactions on Mathematical Software, Vol.**5**, Issue.**4**, pp.**308–323, 1979.**

[4] I. M. Jelas, N. A. W. A. Hamid, M. Othman, “The High-Performance Linpack (HPL) Benchmark on the Khaldun Sandbox Cluster,” Journal of High-Performance Computing, pp.**1-15, 2013.**

[5] Intel Corporation, "Intel® Math Kernel Library: Reference Manual," **2020.**

[6] J. J. Dongarra, J. R. Bunch, G. B. Moler, G. W. Stewart, “LINPACK Users' Guide,” Society for Industrial and Applied Mathematics (SIAM), USA, pp.**1-250, 1979.**

[7] J. J. Dongarra, P. Luszczek, A. Petitet, “The LINPACK Benchmark: Past, Present, and Future,” Concurrency and Computation: Practice & Experience, Vol.**15**, pp.**803-820, 2003.**

[8] J. J. Dongarra, H. W. Meuer, E. Strohmaier, "TOP500 Supercomputer Sites," International Journal of High Performance Computing Applications, Vol.**11**, No.**3**, pp.**90-94, 1997.**

[9] Khang T. Nguyen, "Performance Comparison of OpenBLAS and Intel oneAPI Math Kernel Library in R," International Journal of Computational *Science and Engineering*, Vol.**5**, No.**3**, pp.**123-130, 2020.**

[10] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. J. Dongarra, “MPI: The Complete Reference,” MIT Press, USA, pp.**1-500, 1996.**

[11] M. Fatica, "Accelerating Linpack with CUDA on Heterogeneous Clusters," Proceedings of the 2nd Workshop on General-Purpose Processing on Graphics Processing Units (GPGPU-2), pp.**46-51, 2009.**

[12] Wong Chun Shiang, Izzatdin Abdul Aziz, Nazleeni Samiha Haron, Jafreezal Jaafar, Norzatul Natrah Ismail, Mazlina Mehat, “The High-Performance Linpack (HPL) Benchmark Evaluation on UTP High-Performance Cluster Computing,” Jurnal Teknologi, Vol.**78**, Issue.**9**, pp.**21–30, 2016.**

AUTHORS PROFILE

Shivali Gangwar earned B. Tech., in Information Technology from Guru Gobind Singh Indraprastha University and M.tech. in Computer Science Engineering from Maharishi Dayanand University in 2005 and, 2016 respectively. Currently in Department of High-Performance Computing, in National Centre for Medium Range Weather Forecasting, Noida, since 2013 as part of various HPC organization and joined NCMRWF in 2023. My main research work focuses on HPC, Benchmarking, Profiling Weather Models. I have total 19 years of IT experience and research experience.

Dr. B. Athiyaman earned his Ph.D. in Computer Science from IIIT, Gwalior. His Research Areas are High Performance Computing (HPC), Computer Networking, Data Mining, Neural Network and Fuzzy systems. He is Computing and Infrastructure Head in National Centre for Medium Range Weather Forecasting (NCMRWF), Noida. He has published many research papers. He has more than 30 years of research experience in NCMRWF.

Dr. Preveen Kumar D. earned his Ph.D. from University of Delhi. He is in Computer & AI at National Centre for Medium Range Weather Forecasting (NCMRWF), Noida. His research areas include High Performance Computing, AI/ML, Computer Network, DWR data quality control. He has published many research papers in reputed international journals and conferences. His main research work focuses AI/ML in Weather forecasting and HPC infrastructure Head in NCMRWF. He has more than 30 years of research experience in NCMRWF.