https://doi.org/10.26438/ijcse/v6si1.6065

JCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Volume-6, Special Issue-1, Feb 2018 E-ISSN: 2347-2693

MPI performance guidelines for scalability

K.B. Manwade[1*], D.B. Kulkarni[2]

[1]Dept. of CSE, Walchand College of Engineering, Sangli, India
[2]Dept. of IT, Walchand College of Engineering, Sangli, India

[*]Corresponding Author: mkarveer@gmail.com, Tel.: +91-84129-68254

Available online at: www.ijcseonline.org

***Abstract*—** MPI (Message Passing Interface) is most widely used parallel programming paradigm. It is used for application development on small as well as large high-performance computing systems. MPI standard provides a specification for different functions but it does not specify any performance guarantee for implementations. Nowadays, its various implementations from both vendors and research groups are available. Users are expecting consistent performance from all implementations and on all platforms. In literature, performance guidelines are defined for MPI communication, IO functions and derived data types. By using these guidelines as a base we have defined guidelines for scalability of MPI communication functions. Also, we have verified these guidelines by using benchmark application and on different MPI implementations such as MPICH, open MPI. The experimental results show that point to point communication functions are scalable. It is quite obvious as in point to point communication the only pair of processes is involved. Hence these guidelines are defined as performance requirement by considering the semantics of these functions. All processes are involved in collective communication functions; therefore defining performance guidelines for collective communication is difficult. In this paper, we have defined the performance guidelines by considering the amount of data transferred in the function. Also, we have verified our defined guidelines and reasons for violations of these guidelines are elaborated.

***Keywords*—** Performance guidelines for MPI functions, Scalability of MPI functions, High-performance computing

I. Introduction

In [1], authors analyzed the performance of various parallel programming paradigms like UPC, OpenMP, and MPI. They conclude that MPI outperforms over other two paradigms. Various implementations of MPI standards are available. Users should be made aware of performance portability of these implementations i.e. every implementation of MPI standard gives the same performance. Hence implementation-specific optimization in the program is not required. Also MPI standard defines both specialized as well as generalized communication functions. As per requirement user can use them. The user expects good performance from specialized functions than generalized one but MPI standard does not ensure any performance guarantee. In literature, various MPI self-consistent performance guidelines are defined. The guidelines indicate common sense expectation that an MPI function should perform no worse than a combination of other MPI functions that can implement the same functionality.

In [2], authors have introduced the notion of self-consistent performance requirements for MPI implementations. In this paper, they have defined and verified guidelines for MPI communication functions for varying data size. In [3], William D. Gropp et. al have defined performance guidelines for MPI-IO functions. The guidelines for MPI parallel I/O whose performance depends on the semantics of MPI standard and not on underlying hardware are termed as performance requirements. And those whose performance depends on underlying hardware are termed as performance expectations. In [4], performance expectations and guidelines for MPI derived data types are defined and verified.

This paper focuses on formulation and verification of performance guidelines for MPI communication functions with respect to varying number of processes. The rest of this paper is organized as follows; Section II focuses on related work in the domain of MPI performance guideline verification. In section III, new performance guidelines for scalability of MPI communication functions are defined. The details of experimental setup, the benchmark used and results obtained are elaborated in section IV. The conclusion of our experiments and findings are given in section V.

II. Related Work

In [2], authors have introduced, formulated and verified performance guidelines for MPI communication functions. These guidelines are verified with respect to varying data size. However, these guidelines are not verified with respect to varying number of processes in the program. In [3], performance guidelines are defined for MPI-IO functions and are verified for varying data size as well as varying number of processes.

As a parallel I/O in MPI is more subtle its guidelines can be categorized as performance requirements & expectations. If the details of underlined I/O subsystem are known and sure about its working then the guideline is treated as performance requirement otherwise as performance expectation. In [4], performance guidelines are defined for MPI derived types and verified only for varying data size. While defining guidelines five derived data type constructors like contiguous, vector, index blocked, indexed and structure is considered. Also, performance penalties involved in disk movement to read/write derived data types are considered.

Various MPI benchmark frameworks are available for automatic verification of MPI performance guidelines. The PGMPI [5] framework, verify guidelines for all collective communication functions. The guidelines for MPI point to point communication and IO functions are not verified by this framework. Another benchmark framework for MPI communication functions is SKaMPI [6]. It provides benchmark codes for point to point, collective and master-slave communication patterns. It does not provide code for IO functions.

III. PERFORMANCE GUIDELINES FOR MPI COMMUNICATION FUNCTIONS

A. Point to point communications

The performance guidelines for MPI point to point communications for varying data size are defined in [2] as,

MPI_A(N) ≤ MPI_B(N) (1)

to mean that MPI function A is not slower than MPI function B and implement the operation for same data size N. On similar line we have defined guidelines for varying number of processes as

MPI_A(P) ≤ MPI_B(P) (2)

to mean that MPI function A is not slower than MPI function B and implement the same operation for the same number of processes P. As a point to point communication takes place between a pair of processes, a number of processes don't affect on the performance of communication functions. Therefore we have defined guidelines for point to point communication similar to those defined in [2]. By considering the semantics of point to point communication, we have defined these guidelines as performance requirements.

MPI_Isend(P) + MPI_wait() ≤ MPI_Send(P) (3)

In blocking send message sending activity get completed or sender can reuse buffer only when the entire message data is copied into receiver's buffer. Whereas in immediate send message data can be copied in an overlapping manner to system buffer or receiver's buffer as soon as the sender start writing data into the buffer. From this fact we have formulated equation (3) as a performance requirement.

MPI_Rsend(P) ≤ MPI_Send(P) (4)

Unlike blocking send, message data is copied to receiver's buffer directly without buffering in case of ready send. Therefore ready send will give better performance than blocking send (equation 4).

MPI_Send(P) ≤ MPI_Ssend(P) (5)

The synchronous send involves synchronization overhead; therefore, its performance is worse than standard blocking send. Hence the guideline defined as shown in equation 5.

MPI_Sendrecv(P) ≤ MPI_Isend(P) + MPI_
Recv(P) +MPI_wait(P) (6)

MPI_Sendrecv(P) ≤ MPI_Send(P) + MPI_
Irecv(P) +MPI_wait(P) (7)

The sendrecv function is used for bidirectional send and receive between a pair of processes. This operation takes care of cyclic dependency during send and receives the message, but in case of independent send and receive the dependency need to be resolved by the programmer. Because of this factor sendrecv shows better performance than the combination of Isend and Recv or Send and Irecv as shown in equation 6 and 7.

B. Collective communications

The performance of MPI collective functions depends on the amount of data transferred by the function, algorithm used for its implementation and number of processes involved in the communication. Therefore defining firm performance guidelines for collective communications is difficult. By considering amount of data transferred as a parameter, we have defined performance guidelines as,

MPI_A(P) ≤ MPI_B(P) (8)

to mean that functions A transfers fewer data than function B, therefore, it takes less time for communication than its counterpart. The performance guidelines for collective communications functions and the amount of data transferred by these functions are given in Table 1.

Table 1 Performance guidelines for collective functions

Sr. No.	Performance guidelines for MPI collective functions
1	MPI_Gather(P) ≤ MPI_Allgather(P) (N X P) ≤ (N X P) X P
2	MPI_Alltoall(P) ≤ MPI_Allgather(P) (N X P) ≤ (N X P) X P
3	MPI_Scatter(P) ≤ MPI_Bcast(P) N ≤ (N X P)
4	MPI_Gather(P)+MPI_Bcast(P) ≤ MPI_Allgather(P) (N X P)+ N ≤ (N X P) X P
5	MPI_Bcast(P) ≤ MPI_Scatter(P)+MPI_Allgather(P) (N X P) X P ≤ (N)+(N X P)
6	MPI_Reduce(P) ≤ MPI_Allreduce(P) (N X P) ≤ (N X P) X P
7	MPI_Reduce(P)+MPI_Bcast(P) ≤ MPI_Allreduce(P)

	(N X P)+(N X P) ≤ (N X P) X P
8	MPI_Reducescatter(P) ≤ MPI_Reduce(P)+MPI_Scatterv(P) (N X P)+N ≤ (N X P)+N*
9	MPI_Reduce(P) ≤ MPI_Reducescatter(P)+MPI_Gather(P) (N X P) ≤ [(N X P)+N]+N
10	MPI_Allreduce(P)≤ MPI_Reducescatter(P)+MPI_Allgather(P) (N X P) X P ≤ [(N X P)+N]+(N X P)
11	MPI_Reducescatter(P) ≤ MPI_Allreduce(P) (N X P)+N ≤ [(N X P) X P])
12	MPI_Gather(P) ≤ MPI_Reduce(P) (N X P)=(N X P)
13	MPI_Allgather(P) ≤ MPI_Allreduce(P) (N X P)\times P=(N X P) X P
14	MPI_Gather(P) ≤ MPI_Gatherv(P) (N X P)=(*N X P)
15	MPI_Scatter(P) ≤ MPI_Scatterv(P) N ≤ N*
16	MPI_Allgather(P) ≤ MPI_Allgatherv(P) (N X P)\times P ≤ (*N X P) X P
17	MPI_Alltoall(P) ≤ MPI_Alltoallv(P) (N X P) ≤ (*N X P)
* Mark indicate than data size N is different for each process rather than N/P	

IV. RESULTS AND DISCUSSION

A. *Experimental set-up*

To verify the defined guidelines, experiments are carried out on WCE-Rock cluster [7]. This cluster contains three nodes connected using InfiniBand network and total 40 cores. Each core has the processing power of 2.25 GHz. The total main memory in the cluster is 1 TB and physical storage of 1622 GB. Two MPI libraries open MPI and MPICH are used for verification of our defined guidelines.

Different MPI benchmarks are available for performance guideline verification. The OSU [8] micro-benchmark contains code for point to point and collective communications. The latency and bandwidth parameters are used to measure the performance of communication functions. ReproMPI [9], contains benchmark code for MPI collective communication and their equivalent counterparts. The performance is measured in terms of communication latency. Performance of these codes is measured as communication latency in seconds. The NAS [10] benchmark are designed to evaluate the performance of MPI functions. It contains mainly five kernels and three pseudo code applications. We have OSU benchmark code and measured the latency of communication for 1024 and 1048576 bytes data size for a different number of processes. We have used Hockney model to represent communication latency. Following terms are used to express communication latency using Hockney model.

1. The latency for each message: α
2. The transfer time per byte: β
3. Number of processes: P
4. Size of the message: N

B. *Results*

Table 2. Verifications of performance guidelines for collective functions

Performance guidelines for MPI collective functions	Open MPI		MPICH	
	1024 bytes	1048576 bytes	1024 bytes	1048576 bytes
MPI_Rsend(P) ≤ MPI_Send(P)	√	√	√	√
MPI_Send(P) ≤ MPI_Ssend(P)	√	√	√	√
MPI_Sendrecv(P) ≤ MPI_Isend(P) + MPI_Recv(P) + MPI_wait()	√	√	√	√
MPI_Sendrecv(P) ≤ MPI_Send(P) + MPI_Irecv(P) + MPI_wait()	√	√	√	√
MPI_Gather(P) ≤ MPI_Allgather(P)	√	√	√	√
MPI_Alltoall(P) ≤ MPI_Allgather(P)	√	√	√	√
MPI_Scatter(P) ≤ MPI_Bcast(P)	√	×	√	√
MPI_Gather(P)+MPI_Bcast(P) ≤ MPI_Allgather(P)	√	√	√	√
MPI_Bcast(P) ≤ MPI_Scatter(P)+MPI_Allgather(P)	√	√	√	√
MPI_Reduce(P) ≤ MPI_Allreduce(P)	√	√	√	√
MPI_Reduce(P)+MPI_Bcast(P) ≤ MPI_Allreduce(P)	√	√	√	√
MPI_Reducescatter(P) ≤ MPI_Reduce(P)+MPI_Scatterv(P)	√	√	√	√
MPI_Reduce(P) ≤ MPI_Reducescatter(P)+MPI_Gather(P)	√	×	√	√
MPI_Allreduce(P)≤ MPI_Reducescatter(P)+MPI_Allgather(P)	√	√	√	√
MPI_Reducescatter(P) ≤ MPI_Allreduce(P)	√	√	√	√
MPI_Gather(P) ≈ MPI_Reduce(P)	√	×	√	√
MPI_Allgather(P) ≈ MPI_Allreduce(P)	×	×	×	×
MPI_Gather(P) ≤ MPI_Gatherv(P)	√	√	√	√
MPI_Scatter(P) ≤ MPI_Scatterv(P)	√	√	√	√
MPI_Allgather(P) ≤ MPI_Allgatherv(P)	√	√	√	√
MPI_Alltoall(P) ≤ MPI_Alltoallv(P)	√	√	√	√
MPI_Isend(P)+MPI_wait() ≤ MPI_Send(P)	√	√	√	√

The guidelines for point to point communications for both data size: 1024 bytes and 1048576 bytes are verified for both Open MPI and MPICH library. This is quite obvious as the point to point communication takes place between the pair of processes; therefore a number of processes will not make any impact on the performance of communication function. Most of the collective communication functions guidelines are verified but few functions are not verified.

C. *Violation of guidelines*

As shown in table 2, few collective communication guidelines are violated on Open MPI and MPICH. Approximately 5% of defined collective communication guidelines are violated. The details of violated guidelines are given below.

Case study 1: MPI_Scatter ≤ MPI_Bcast

In Open MPI, scatter function is implemented by using binomial tree algorithm $[\log_2 P \times (\alpha + N \times \beta)]$ for a small message and linear algorithm $[(P-1) \times (\alpha + N \times \beta)]$ for a large message. Therefore this guideline is verified for a small message as shown in figure 1. The broadcast algorithm

is implemented by using binomial tree algorithm [$\log_2 P \times (\alpha + N \times \beta)$] for a small message and splitted linear algorithm [$(P-1) \times (\alpha + N \times \beta)$] for a large message.
Therefore broadcast function shows better performance than scatter collective communication as shown in figure 2.

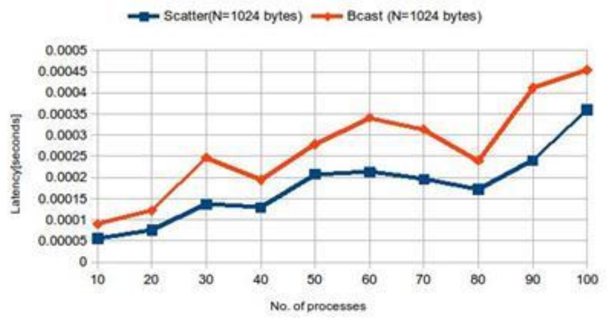

Figure 1. *Verification of guideline MPI_Scatter ≤ MPI_Bcast for small message*

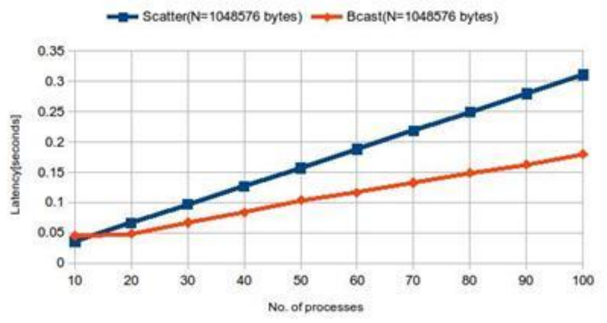

Figure 2. Violation of guideline MPI_Scatter ≤ MPI_Bcast for large message

Case study 2: MPI_Allreduce(P) ≤ MPI_Reducescatter(P) + MPI_Allgather(P)

In Open MPI, to implement Allreduce function recursive doubling algorithm [$\log_2 P \times (\alpha + N \times \beta)$] is used for both small and large messages. To implement Reduce-scatter and Allgather algorithm binomial tree algorithm [$\log_2 P \times (\alpha + N \times \beta)$] for a small message and linear algorithm [$(P-1) \times (\alpha + N \times \beta)$] for a large message is used. As latency of recursive doubling and binomial tree algorithm is same the guideline is verified for a small message as shown in figure 3; but the latency of linear algorithm is more than recursive doubling algorithm, the guideline is violated as shown in figure 4.

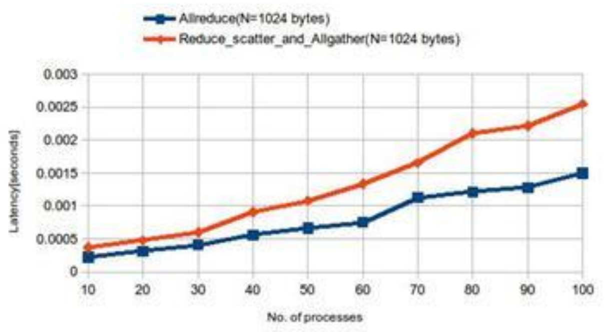

Figure 3. Verification of guideline MPI_Allreduce(P) ≤ MPI_Reducescatter(P) + MPI_Allgather(P) for Small message

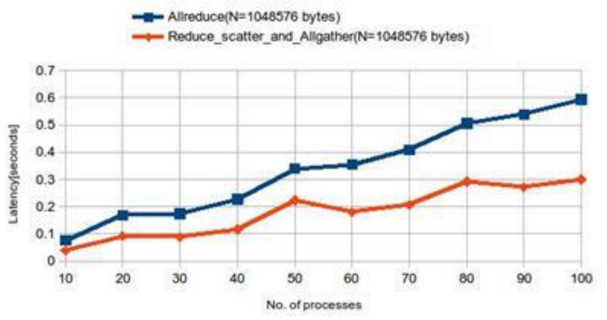

Figure 4. Violation of guideline MPI_Allreduce(P) ≤ MPI_Reducescatter(P) + MPI_Allgather(P) for large message

Case study 3: MPI_Gather(P) ≈ MPI_Reduce(P)

In Open MPI, to implement Gather function binomial tree algorithm [$\log_2 P \times (\alpha + N \times \beta)$] is used for small message and linear algorithm with 32 KB segmentation is used for large messages. To implement Reduce function binomial tree algorithm [$\log_2 P \times (\alpha + N \times \beta)$] is used for small message and a linear algorithm is used for the large message. Because of segmentation overhead, the guideline is violated for a large message (figure 6).

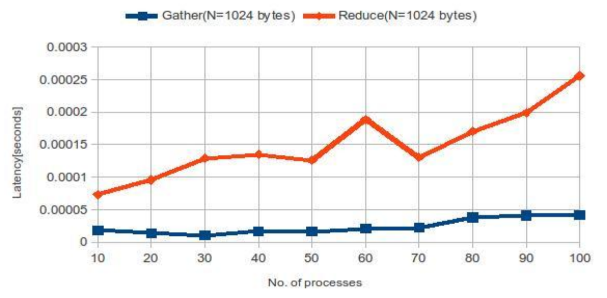

Figure 5. *Verification* of guideline MPI_Gather(P) ≈ MPI_Reduce(P) for small message

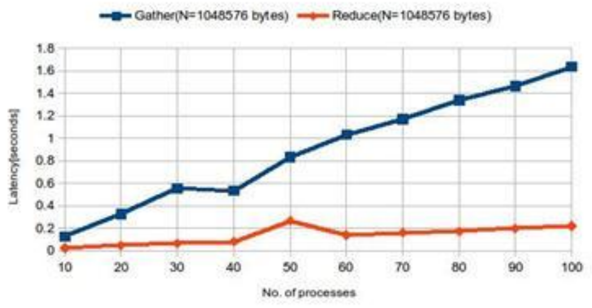

Figure 6. Violation of guideline MPI_Gather(P) ≈ MPI_Reduce(P) for large message

Case study 4: MPI_Allgather(P) ≈ MPI_Allreduce(P)

In open MPI to implement Allgather function linear and binomial tree algorithms are used. For small message linear algorithm [(P-1) × (α + N × β)] is used and for large message, binomial tree algorithm [$\log_2$P × (α + N × β)] is used. Whereas to implement Allreduce function recursive doubling algorithm [$\log_2$P × (α + N × β)] is used for both small and large messages. Therefore, as shown in figure 7 and 8 the defined performance guideline between Allgather and Allreduce are violated for both small and large message.

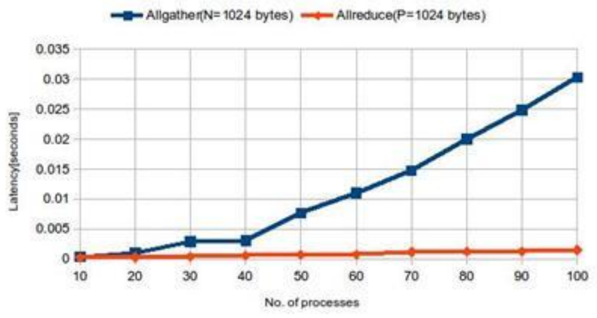

Figure 7. Violation of guideline MPI_Allgather(P) ≈ MPI_Allreduce(P) for small message

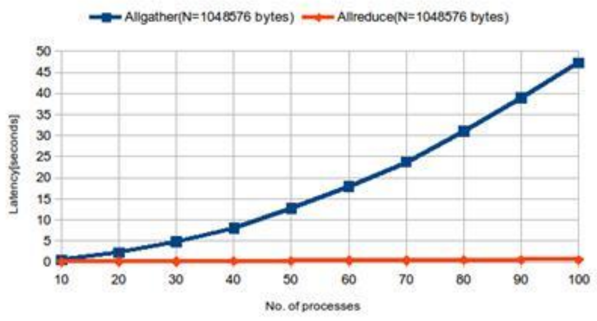

Figure 8. Violation of guideline MPI_Allgather(P) ≈ MPI_Allreduce(P) for small message

V. Conclusion and Future Scope

Though MPI is a standard programming paradigm for parallel programming, it requires improvement in various areas like performance, scalability, fault tolerance, support for debugging and verification, topology-aware process placement, derived data types, collective communication, parallel IO etc. We have defined new guidelines and experimentally verified them.

The experimental result shows that defined guidelines are verified except few guidelines. The scalability guidelines for point to point communication functions are verified as a point to point communication takes place between a pair of processes. Therefore a number of processes in the program will not affect on communication latency. Hence all guidelines for point to point communication are verified. In case of collective communication functions, the communication latency depends on a number of processes in the program and algorithm used for its implementation. Therefore even though both specific and general collective function transfer same amount of data, the guidelines get violated because a different algorithm is used for implementation of collective communication functions. Approximately 5 % of defined guidelines are violated.

In future, we are going to extend our work to verify the guidelines for MPI-IO functions for scalability.

References

[1] A. Mallón, Guillermo L. Taboada, Carlos Teijeiro, Juan Touriño, Basilio B. Fraguela, Andrés Gómez, Ramón Doallo, J. Carlos Mouriño, "Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures", Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2009. Lecture Notes in Computer Science, pp. 174–184, 2009.

[2] William D. Gropp, Rajeev Thakur, "Self-consistent MPI performance guidelines", IEEE Transaction on parallel and distributed systems, 2005.

[3] William D. Gropp, Dries Kimpe, Robert Ross, Rajeev Thakur and Jesper Larsson Traff, "Self-consistent MPI-IO performance requirements and expectations", Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2008. Lecture Notes in Computer Science, 2008.

[4] William D. Gropp, Dries Kimpe, Robert Ross, Rajeev Thakur and Jesper Larsson Traff, "Performance Expectations and Guidelines for MPI Derived Datatypes", Recent Advances in the Message Passing Interface. EuroMPI 2011. Lecture Notes in Computer Science, 2011.

[5] Sascha Hunold, Alexandra Carpen-Amarie, Felix Donatus Lübbe, and Jesper Larsson Träff TU Wien, "Automatic verification of self-consistent MPI performance guidelines", Parallel Processing, Euro-Par 2016. Lecture Notes in Computer Science, 2016.

[6] Ralf Reussner, Peter Sanders, and Jesper Larsson Träff, "SKaMPI: A Comprehensive Benchmark for Public Benchmarking of MPI," Journal of Scientific Programming, vol. 10, issue 1, pp. 55-65, 2002.

[7] WCE Rock Cluster, High performance computing cluster, URL: http://wce.ac.in/it/landing-page.php?id=9.

[8] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. Kini, P. Wyckoff, and D. K. Panda, “Micro-Benchmark Performance Comparison of High-Speed Cluster Interconnects” , Proceedings of 11th Symposium on High Performance Interconnects, 2003.

[9] Hunold, S., Carpen-Amarie, A., “Reproducible MPI benchmarking is still not as easy as you think”, IEEE Transactions on Parallel and Distributed Systems , vol. 27, issue 12, 2016.

[10] Subhash Saini, Robert Ciotti,Brian T. N. Gunney, Thomas E. Spelce, Alice Koniges, Don Dossa, Panagiotis Adamidis, Rolf Rabenseifner, Sunil R. Tiyyagura, Matthias Mueller, “Performance Evaluation of Supercomputers using HPCC and IMB Benchmarks”, Journal of Computer and System Sciences, vol. 74, issue 6, 2008.

Authors Profile

Mr. K. B. Manwade has completed BE in Computer Science & Engineering from Walchand College of Engineering, Sangli in 2004. He has completed MTech degree in Computer Science & Technology from Department of Technology, Shivaji University, Kolhapur in 2008. Currently, he is pursuing Ph.D. degree in Computer Science and Engineering at Walchand College of Engineering, Sangli (A Ph.D. research center of Shivaji University, Kolhapur).

Dr. D. B. Kulkarni received the doctorate degree in Computer Science and Engineering from Shivaji University, Kolhapur in the year 2005. Currently, he is a professor in Information Technology Department of Walchand College of Engineering, where he teaches many courses in the area of Computer Science. During his professional life, he has been involved in several R&D projects funded by AICTE, DRDO and UGC. Recently he received Early Adopter award of National Science Foundation's (NSF) of Technical Committee on Parallel Processing (TCPP) of US$1500 for framing and adapting curriculum on “Parallel Computing” in UG and PG curriculum in the institute. He was Visiting Professor in Institute of Computer Technology (ICT) of Vienna University of Technology, Austria, in 2010. He is a coauthor of tens of scientific papers published in international journals and conference proceedings. His research interests include the area of High-Performance computing and Computer Network.