Asian Journal of Information Technology 13 (11): 700-706, 2014
ISSN: 1682-3915

Optimizing MPI Communication Using Heuristic Algorithms

[1]T. Satish Kumar, [2]S. Sakthivel and [3]M. Manjunatha Swamy
[1]Faculty of Information and Communication Engineering,
Anna University, Chennai, Tamilnadu, India
[2]Department of CSE, Sona College of Technology, Salem, Tamilnadu, India
[3]Department of CSE, RNS Institute of Technology, Bengaluru, India

Abstract: For high performance computing using distributed memory architecture, MPI is the de-facto standard. To achieve high system performance the MPI communication routines have to be optimized. This can be done by tuning the runtime parameters. But, to find the optimal values for the important runtime parameter is a challenging task. Several hundred runs are required and the parameter values found are specific to a particular input. In this study, certain standard benchmarks are used to overcome this problem so that the optimal values found for the parameters can be used for other similar applications. Two heuristic algorithms: Genetic algorithm and Simulated Annealing algorithm are used to find the optimal MPI runtime parameter values. It is proved to have significantly reduced the time and effort in predicting the parameters. A comparison is made among two algorithms and also among variations in Genetic algorithm based on performance gain obtained using optimal runtime parameter values with respect to default MPI parameter values.

Key words: MPI runtime parameters, Genetic algorithm, simulated annealing, parameter optimization, de-fecto

INTRODUCTION

Message Passing Interface (MPI) is the de-facto standard for programming on High Performance Computing (HPC) (Landau, 2013) which runs processes on different processors of distributed memory systems. MPI is a standard library based on the consensus of the MPI forum which has over many participating organizations including vendors, researchers, software library developers and users. MPI allows tuning of parameters to match the underlying cluster architecture. Usually, vendors do the parameter tuning for production clusters (Pellegrini *et al*., 2012) like BlueGene, Tihane and Cray Titan. But as mid-sized clusters are becoming more common, parameter tuning has a non-trivial task.

Parameters may be further classified as compile time and runtime parameters. Compile time parameters are those which are set when MPI library is compiled and whose tuning are more related to functionality such as enabling or disabling the Infiniband support. Runtime parameters on the other hand, allow the customization of the MPI environment to better fit the specific needs of the application, operating system or the hardware. For example, the semantics of point to point communication can be changed by setting a threshold value for the size of message being transmitted. If the message size is above threshold value, MPI library uses Rendezvous protocol requiring acknowledgement or else eager protocol is used which does not require acknowledgment to be sent. Although, MPI library provides default values for the parameters, they may not lead to optimal performance in all cases and is dependent on underlying cluster architecture.

Most popular and widely used implementation of MPI library is Open MPI (Grama *et al*., 2003) which allows tuning of runtime parameters through Modular Component Architecture (MCA) and MVAPICH (MVAPICH Team, 2014) that does it through environment variables. Open MPI implements both MPI-1 (Message Passing Interface Forum, 1995) and MPI-2 (Message Passing Interface Forum, 1997).

MCA is composed of frameworks, components and modules. Framework manages zero or more components and is dedicated to a specific task like providing MPI collective operation functionality. Component is a specific implementation of frameworks' interface. An MCA module is an instance of a component. If a node running MPI application has two Ethernet NICs, then there will be one TCP MPI point to point component but two TCP point to point modules. MCA parameters are "key = value" pairs and are set in a file so that it becomes easy for tuning their values.

Corresponding Author: T. Satish Kumar, Faculty of Information and Communication Engineering, Anna University, Chennai, Tamilnadu, India

Open Tool for Parameter Optimization (OTPO) (Chaarawi *et al*., 2008) is a tool used to determine the optimal parameter combination to minimize the execution time on a target machine. But, it takes lot of time to determine the combination and is specific to input program. Machine Learning algorithms (Pelligrini *et al*., 2009) have been proposed to predict the optimal value for runtime parameters but it works efficiently if the number of parameters is less.

A set of NAS Parallel Benchmarks (NPB) (Bailey *et al*., 1994) have been used in this study that are common in HPC. The idea is that optimal values found for runtime parameters for these benchmarks can be used for other similar applications. Heuristic algorithms have been used to determine the optimal values for runtime parameters. These found an approximate solution close to the best one quickly, when classical methods take a lot of time. Two heuristic algorithms namely Genetic Algorithm (GA) (Beasley *et al*., 1993) and Simulated Annealing (Kirkpatrick *et al*., 1983) have been used in the experiments. In GA, we use various selection methods like Roulette Wheel (RW) selection (Sivanandam and Deepa, 2008), Fitness Proportionate Solution, Stochastic Universal Sampling (SUS) (Pencheva *et al*., 2009) and Elitism and reproduction methods like single and multipoint crossover. GA stops at maximum number generations. In this research, the maximum number of generations are kept to 2000 (Agbele *et al*., 2012). Finally, comparisons are made between the heuristic algorithms and also by changing various operators on GA.

LITRATURE REVIEW

There are few research related to optimization of the MPI runtime parameters till date. OTPO automates the process of tuning the runtime parameters by running the input program several hundred times, considering various possible runtime parameter combinations for each run. It is based on Abstract Data and Communication Library (ADCL) (Gabriel and Huang, 2007) to search for parameter combination with lowest execution time. ADCL adopts two unique methods, one using brute force approach for testing all the available combinations and other using a heuristic approach that relies on parameter values that characterizes the performance. OTPO not only takes lot of time to find the optimal runtime parameter values but also works specific to the input program. In contrast to OTPO, the method used in this research finds optimal parameter values for a set of more generalised benchmarks, so that same values can be used for similar applications. In addition, heuristic algorithms like Genetic algorithm and simulated annealing provide optimal values in lesser number of iterations.

Combined Elimination (CE) (Pan and Eigenmann, 2006) is an iterative algorithm which combines both the Batch Elimination (BE) and Iterative Elimination (IE). BE identifies optimization flags having negative effect and eliminates them in batch whereas in IE, one flag is turned off at a time having maximum negative effect and thus takes interactions among optimizations into consideration. If the interaction among optimizations is less, CE uses BE, else it uses IE. The algorithm takes into account parameters having only binary value whereas MPI runtime parameters can take a range of values.

Machine learning (Pelligrini *et al*., 2009) is used to predict the optimal values for the runtime parameters. First training programs are executed on the targeted architecture and information is collected about the features and the values of runtime parameters that lead to minimum execution time. This collected data is utilized to build a prediction model using Machine Learning algorithms such as decision tree and neural networks. One profiled execution of the new program is required to extract its program features like the amount of data exchanged in point to point and collective communication, communication graph, etc. Trained model is then queried based on program features to predict the values for runtime parameters that may yield optimal performance. It is learnt that this approach reduces the number of runs for the experiments and is used more efficiently for smaller collection of runtime parameters.

In research by Pelligrini *et al*. (2012) optimal values of runtime parameters are determined for NASA Advanced Super Computing (NAS) parallel benchmarks using randomized algorithms in exploration phase. Later, using statistical method called Analysis of Variance (ANOVA) (Montgomery, 2005), those parameter which have highest effect on performance are determined. In contrast, this research focuses on more than one Heuristic Method for finding the optimal values for runtime parameters and comparison among them.

SYSTEM ARCHITECTURE

System architecture is shown in Fig. 1. The User Interface (UI) layer consists of configuration files where the user enters the configuration details like No. of generations, No. of binary valued parameters, population size, etc.

The next layer is the optimization layer. Configuration file is read using file handling functions. Runtime parameters are chosen and selected benchmarks are run by setting values to the chosen parameters. Heuristic algorithm is applied to find out the optimal values for these parameters.

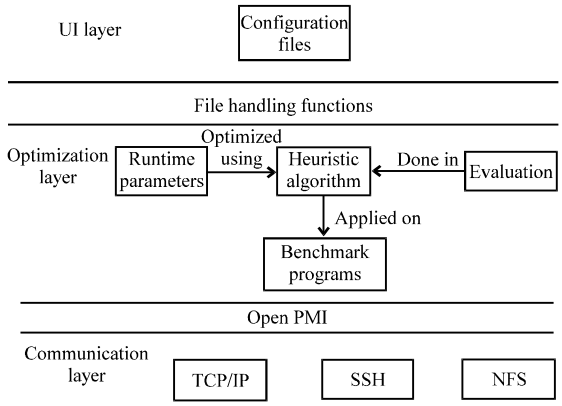

Fig. 1: System architecture

The third layer is the communication layer. A Beowulf cluster is rigged up consisting of 2-8 nodes which can be scaled up as per the requirement. Open MPI is used for communication between the master and the slave nodes. For this to happen, same piece of code has to reside on all the machines. Network File System (NFS) is used to share the common folder containing the source code. Open MPI uses SSH to communicate within nodes. So, open SSH has to be installed and the password authentication has to be removed on all nodes. TCP/IP protocol is used for the communication.

EXPERIMENTAL SET UP

Table 1 shows the configuration of the cluster used for experiments which were conducted on 2, 4 and 8 nodes, respectively.

Open MPI runtime parameters are set in the file 'mca-param.conf' as MCA parameters which are 'key = value' pairs. This is followed by execution of benchmark and recording the execution time.

TESTING PROBLEMS

A set of application benchmarks from NPB Version 3.3 have been selected so that effects of computation on communication can be efficientlycaptured (Table 2).

Six benchmarks are Conjugate Gradient (CG), Data Traffic (DT), Embarrassingly Parallel (EP), Fourier Transform (FT), Integer Sort (IS) and Multi-Grid (MG). CG is an algorithm for finding the numerical solution for particular systems of linear equations, namely those whose matrix is positive-definite and symmetric. Irregular long distance communications are tested by this benchmark. DT tests the data traffic between the nodes. In Embarrassingly Parallel (EP) benchmark, a problem is divided into sub problems with little or no efforts where dependency between the parallel tasks doesn't exist.

Table 1: Target architecture

Parameters	Values
No. of nodes	2-8
No. of cores per node	4
Memory per node	4 GB
Open MPI version	1.4.2
Make	Dell 390 Optiplex
Processor	Intel core i5
OS	Ubuntu 12.04 LTS
Clock frequency	2.5 GHz

Table 2: Runtime parameters

Parameter name	Description
mpi_yield_when_idle	Decides whether a MPI process must yield the core to other processes when idle
mpi_paffinity_alone	Controls binding of process to specific processing resources like cores, sockets, etc.
mpi_preconnect_mpi	Determines whether application has to pre-connect all its processes during MPI_Init
mpi_leave_pinned	Pre-registers user message buffers
btl_sm_eager_limit	Size limit on message to determine if it has to be sent eagerly without waiting for receiver to be ready
btl_sm_max_send_size	Size limit of a fragment of large message
btl_sm_num_fifos	Number of FIFOs per receiving process
btl_sm_fifo_size	Size of the FIFO queue
btl_sm_rndv_eager_limit	Size (bytes) of "phase 1" fragment sent for all large messages

Problem of generating Gaussian random deviates according to a particular scheme is considered in this benchmark. FT solves a Partial Differential Equation (PDE) using fourier transform which converts it to a simple algebraic equation. A solution is got for the algebraic equation on which inverse Fourier transform is applied to get the solution to the original PDE. IS does sorting operation on a large set of integers. This benchmark program tests both communication performance and computation speed. MG is used to find approximate solution to discrete Poisson problem while testing both long and short distance communication.

ALGORITHMS USED

GA (Goldberg, 2013) is search algorithm that mimics the process of biological evolution. It is predominantly used in the optimization problem solving techniques (Kumar *et al*., 2014). Starting with an initial population of candidate solutions, evolution takes place as the fittest individuals contribute the offspring to the next generation. In this research, the chromosome consists of string of binary values representing the value of runtime parameters. Parameters that take a range of values can be represented using more than one bit of the chromosome. A set of chromosomes form a population and initial population is created randomly. The fitness value is associated with each of the chromosome. In this case it is execution time. According to the binary chromosomes, values are set for runtime parameters in the file

'mca-param.conf' and the benchmark programs are run. The execution time is recorded as the fitness value of the chromosome.

Based on this fitness value, parents are selected using various selection methods and are used to reproduce the offspring of next generation. Here, in the implementation, crossover and mutation operations are used for reproduction process. This cycle continues till specified number of generations after which approximately optimal solution is found. General flow chart of a GA is shown in Fig. 2.

Three selection strategies have been used in this work namely Roulette Wheel (RW) selection/fitness proportionate solution, Stochastic Universal Sampling (SUS) and Elitism. In roulette wheel selection, the fitness value of each chromosome is used to determine the probability of getting selected. If f_i is the fitness of individual i in the population, the selection probability is given by Eq. 1:

$$P_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \quad (1)$$

where, N is the number of individuals in the population. Necessary, modification has been made so that the chromosome taking less execution time has high probability of getting selected.

SUS is a slight modification of RW where instead of a single pointer N equally spaced pointers are used. Here, N represents the number of selections required. N pointers are generated starting from pointer1 and equally spaced by 1/N. Individuals are selected whose fitness spans the position of the pointer. SUS does not exhibit any bias and does minimal spread. If an individual occupies 3.5% of the wheel out of 100 individuals, then on an average the individual is expected to be selected 3 times or 4 times.

Elitism is a variation of RW where first half of the population, fittest individuals are carried over to the next generation. Each new generation will be a combination of new and old population.

Reproduction consists of crossover between the selected parents. Two kinds of crossover are used in the experiment namely single point crossover and multipoint crossover. In single point crossover two mating chromosomes are cut at a single position and bits next to the cross-sites are exchanged. Figure 3 illustrates the process.

In multipoint crossover the bits in even or odd positions will be exchanged. This produces two new children with their bits exchanged at multiple positions. Figure 4 shows multipoint crossover at even positions. SA is a heuristic algorithm which finds good approximation to the global optimum. Annealing in

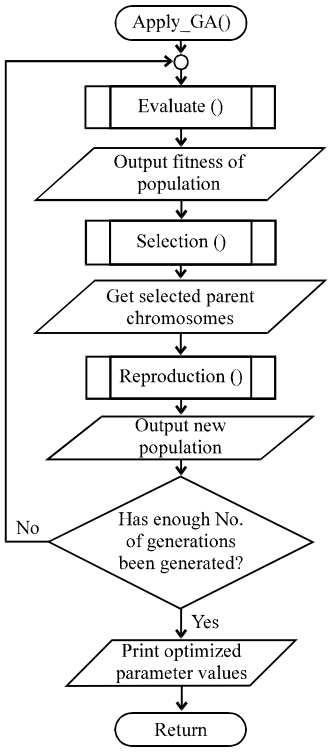

Fig. 2: Flow chart of GA

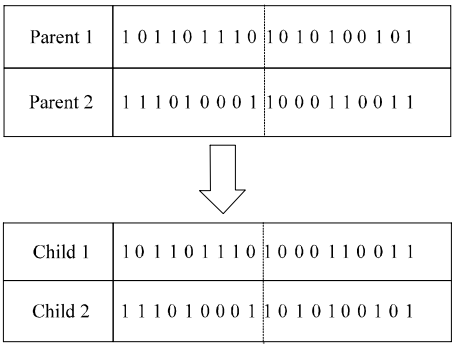

Fig. 3: Single point crossover

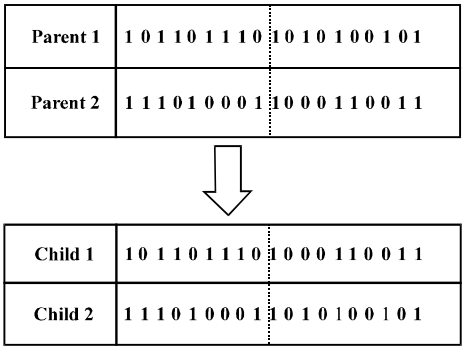

Fig. 4: Multipoint crossover where the bits in even position (bold) are exchanged

metallurgy is a technique of heating a material followed by controlled cooling to increase the size of crystals and reduce their defects. Similarly, in SA algorithm used, there is a slow decrease in the probability of accepting worse solution.

First an initial solution is generated randomly and execution time is recorded. Then, small change is brought in the values of runtime parameters that form the neighbour solution. The acceptance is notified when the neighbour solution takes less time to execute. It is accepted with a probability given in Eq. 2:

$$P(i) = \min(1, e^{(2d/T)}) \quad (2)$$

Where:

d = The difference between the execution times of old and neighbouring solution

T = The control parameter

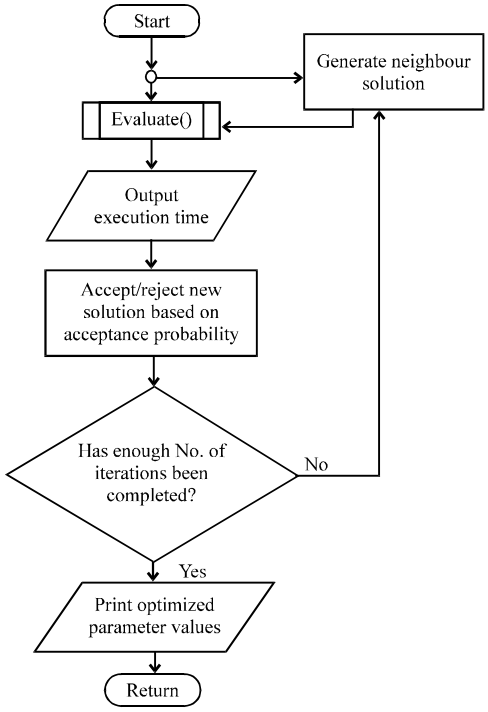

Fig. 5: Flow chart of simulated annealing

On slowly reducing the temperature T, algorithm converges to global minimum. The value of P(i) goes on reducing as 'T' reduces. So initially, probability of accepting worse solution will be more which goes on decreasing as the value of 'T' decreases. Figure 5 shows the flow chart of simulated annealing process.

OBSERVATIONS

Benchmark programs were run by setting the runtime parameter values determined by GA and SA. A sample observation for 8 nodes is shown in Table 3. All the values are in milliseconds (execution time) except for (%) gain in performance. Table 4 shows the optimal runtime parameters for CG benchmark on 2, 4 and 8 nodes. Similarly, values were found for other benchmarks as well.

RESULTS

Based on the observations and calculations, a comparison is made between the two heuristic algorithms GA and SA used for finding optimal runtime parameter values on 2, 4 and 8 systems as shown in Fig. 6-8, respectively.

Figure 9 shows the probability of finding optimal parameter values using GA and SA. It is found that probability of finding solution using GA is 72% while that using SA is 28%.

A comparison was also made among various selection strategies on basis of their ability to find optimal parametervalues. Figure 10 shows that roulette wheel has better probability of finding optimal solution than SUS and Elitism.

When a comparison was made between single point and multi point crossover in finding optimal solution, it was found that multipoint crossover is better than single point crossover. Figure 11 shows the comparison of two crossover methods.

Performance gain up to 18.95% was achieved using optimized runtime parameter values with respect to

Table 3: Execution time of benchmarks run using parameter values found using GA and SA

	8 nodes-100 g											1000 iterations	
	Roulette wheel			Elitism			SUS				Gain performance		Gain performance
Parameters	SP	MP_ODD	MP_EVEN	SP	MP_ODD	MP_EVEN	SP	MP_ODD	MP_EVEN	Default	(%)	SA	(%)
CG	1521.320	1470.860	1478.690	1554.170	1477.420	1478.020	1503.140	1473.950	**1470.300**	1814.140	18.95	1474.930	18.45
DT	678.870	644.124	**644.063**	648.101	649.565	651.491	649.014	647.265	650.263	681.759	5.44	666.767	2.18
EP	867.292	866.530	**854.959**	863.681	866.251	876.989	859.224	871.417	871.389	908.664	5.91	892.330	1.80
FT	1738.890	1761.760	**1738.550**	1740.950	**1733.830**	1775.460	1770.750	1751.390	1759.420	1870.300	7.29	**1687.010**	9.80
IS	619.897	617.262	625.672	627.762	633.565	627.517	627.878	**608.899**	631.299	639.840	4.83	623.077	2.62
MG	1305.560	1303.680	1303.820	**1295.680**	1311.910	1299.560	1302.650	1308.910	1309.670	1353.540	4.27	1297.230	4.16

SP: Single Point crossover, MP_ODD: Multipoint crossover at Odd positions, MP_EVEN: Multipoint crossover at Even positions

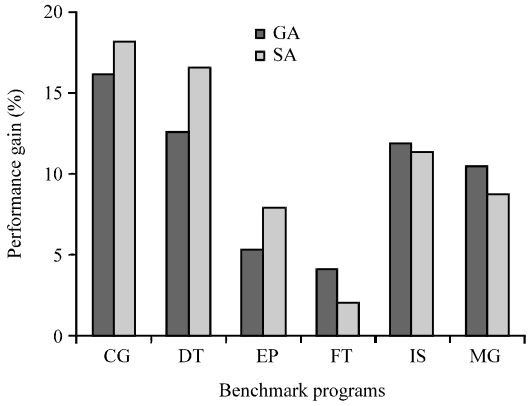

Fig. 6: Comparison of performance gain using runtime parameter values found with SA and GA for 2 nodes as compared with that of default values

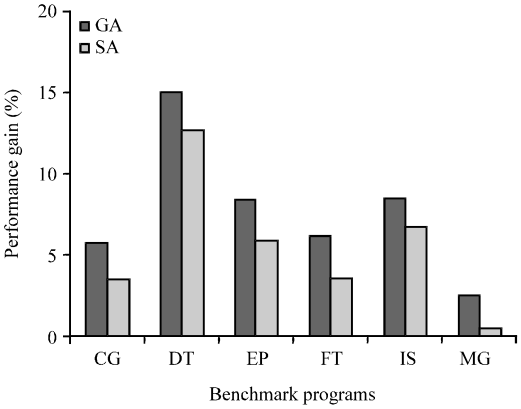

Fig. 7: Comparison of performance gain using runtime parameter values found with SA and GA for 4 nodes as compared with that of default values

Table 4: Optimal runtime parameter values for CG benchmark on 2, 4 and 8 nodes

CG	No. of nodes		
Parameters name	2	4	8
mpi_yield_when_idle	1	1	0
mpi_paffinity_alone	0	1	1
mpi_preconnect_mpi	0	1	1
mpi_leave_pinned	0	1	0
btl_sm_eager_limit	16 kb	8 kb	16 kb
btl_sm_max_send_size	32 kb	8 kb	32 kb
btl_sm_rndv_eager_limit	16 kb	16 kb	16 kb
btl_sm_num_fifos	4	2	8
btl_sm_fifo_size	8 kb	1 kb	8 kb

default open MPI's parameter settings. Average performance gain using GA is 8.16% and that using SA is 7.57%. Probability of finding optimal parameter values is high using GA than SA. Roulette wheel selection proves to be better in finding optimal solution when compared to

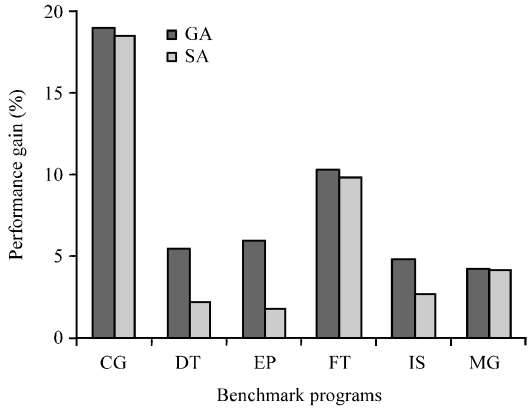

Fig. 8: Comparison of performance gain using runtime parameter values found with SA and GA for 8 nodes as compared with that of default values

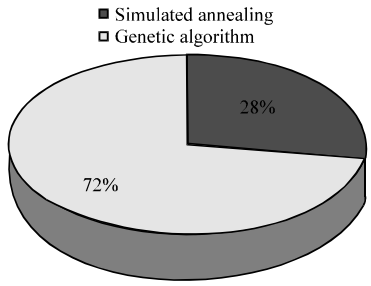

Fig. 9: Probability of finding optimal parameter setting using GA and SA

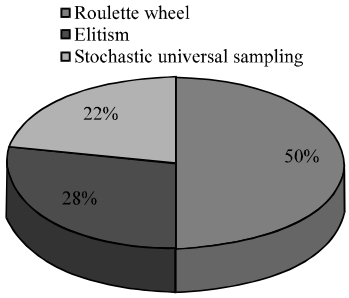

Fig. 10: Probability of finding optimal runtime parameter setting using different selection methods

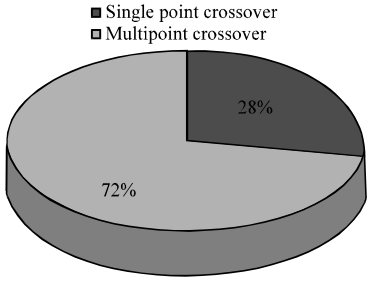

Fig. 11: Probability of finding optimal runtime parameter setting using different crossover methods

SUS and Elitism. It was also found that multipoint crossover is better than single point crossover as a reproduction technique.

CONCLUSION

MPI communication is optimized by finding optimal values for runtime parameters. NAS parallel benchmarks are used so that the optimal values found for the parameters can be used for other similar applications. Two heuristic algorithms GA and SA are used to greatly reduce the time and effort. Up to 18.95% gain is obtained in performance with average percentage gain being 8.16% with respect to MPI's default parameter setting.

Comparison was made between two heuristic algorithms and also among various selection and reproduction strategies in GA. It was found that GA used with Roulette wheel selection and multipoint crossover gives better chances of finding optimal solution quicker than any other algorithms.

REFERENCES

Agbele, K., A. Adesina, D. Ekong and O. Ayangbekun, 2012. State of the art review on relevance of genetic algorithm to internet web search. Asian J. Inform. Technol., 11: 117-124.

Bailey, D., E. Barszcz, J. Barton, D. Browning and R. Carter et al., 1994. The NAS parallel benchmarks. RNR Technical Report RNR-94-007, March 1994. http://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf.

Beasley, D., D.R. Bull and R.R. Martin, 1993. An overview of genetic algorithms: Part 1, Fundamentals. Univ. Comput., 15: 58-69.

Chaarawi, M., J.M. Squyre, E. Gabriel and S. Feki, 2008. A tool for optimizing runtime parameters of open MPI. Proceedings of the 15th European PVM/MPI Users Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS., Volume 5205, September 7-10, 2008, Dublin, Ireland, pp: 210-217.

Gabriel, E. and S. Huang, 2007. Runtime optimization of application level communication patterns. Proceedings of the 12th International Workshop on High-Level Parallel Programming Models and Supportive Environments, March 26, 2007, Long Beach, CA., USA.

Goldberg, D.E., 2013. Genetic Algorithms in Search, Optimization and Machine Learning. Pearson Publication, Singapore.

Grama, A., G. Karypis, V. Kumar and A. Gupta, 2003. Introduction to Parallel Computing. 2nd Edn., Addison Wesley, Redwood City, California.

Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, 1983. Optimization by simulated annealing. Science, 220: 671-680.

Kumar, T.S., S. Sakthivel and S.S. Kumar, 2014. Optimizing code by selecting compiler flags using parallel genetic algorithm on multicore CPUs. Int. J. Eng. Technol., 6: 544-551.

Landau, R.H., 2013. A beginner's guide to high-performance computing. Oregon State University, USA.

MVAPICH Team, 2014. MVAPICH 1.0 user and tuning guide. http://mvapich.cse.ohio-state.edu/.

Message Passing Interface Forum, 1995. MPI: A message passing interface standard. Version 1.1, June 12, 1995. http://www.mpi-forum.org/.

Message Passing Interface Forum, 1997. MPI-2: Extensions to the message passing interface. Version 1.2, July 18, 1997. http://www.mpi-forum.org/.

Montgomery, D.C., 2005. Design and Analysis of Experiments. 6th Edn., John Wiley and Sons Inc., New York.

Pan, Z. and R. Eigenmann, 2006. Fast and effective orchestration of compiler optimizations for automatic performance tuning. Proceedings of the International Symposium on Code Generation and Optimization, March 26-29, 2006, Washington, DC., USA., pp: 319-332.

Pellegrini, S., R. Prodan and T. Fahringer, 2012. Tuning MPI runtime parameter setting for high performance computing. Proceedings of the IEEE International Conference on Cluster Computing Workshops, September 24-28, 2012, Beijing, pp: 213-221.

Pelligrini, S., J. Wang, T. Fahringer and H. Moritsch, 2009. Optimizing mpi runtime parameter settings by using machine learning. Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Volume 5759, September 7-10, 2009, Espoo, Finland, pp: 196-206.

Pencheva, T., K. Atanassov and A. Shannon, 2009. Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. Proceedings of the 10th International Workshop on Generalized Nets, December 5, 2009, Sofia, Bulgaria, pp: 1-7.

Sivanandam, S.N. and S.N. Deepa, 2008. Introduction to Genetic Algorithms. Springer, USA., ISBN: 354073 189X, Pages: 442.