A New Process Placement Algorithm in Multi-core Clusters Aimed to Reducing Network Interface Contention

Ghobad Zarrinchian, Mohsen Soryani, and Morteza Analoui

Iran University of Science and Technology, Tehran, Iran
{Zarrinchian,soryani,analoui}@comp.iust.ac.ir

Abstract. The number of processing cores within computing nodes which are used in current clustered systems, are growing up rapidly. Despite this trend, the number of available network interfaces in such nodes almost has been remained unchanged. This issue can lead to high usage of network interface in many workloads, especially in workloads which have high inter-process communications. As a result, network interface would become a performance bottleneck and can degrade the performance drastically. The goal of this paper is to introduce a new process mapping algorithm in multi-core clusters aimed to reducing network interface contention and improving the performance of running parallel applications. Comparison of the new algorithm with other well-known methods in synthetic and real workloads indicates that the new strategy can gain 5% to 90% performance improvement in heavy communicating workloads.

1 Introduction

Parallel processing is one of the basic approaches to obtain high processing power. This power is necessary to run many scientific and economic applications which are known as Grand Challenge Applications (GCA). In this regard, various architectures have been introduced. Of these architectures, cluster computing has gained more popularity such that based on last published issues in 2011 [1], up to 82% of 500 top supercomputers in the world, used this architecture. Besides, recent advancements in multi-core processor technology have made these processors an excellent choice to use in clustered nodes. Magny-cours series of AMD Opteron and Westmere series of Intel Xeon which have 12 and 10 cores per chip respectively are some examples of multi-core processors which are becoming common in recent computing nodes.

Although multi-core processors, can improve computational capability, but they raise some challenges. The main challenge in this regard, is the contention of various cores for using shared resources like memory and buses. In the presence of such contention, shared resources can be performance bottleneck and can degrade the performance of running parallel applications drastically. Consequently, efficient execution of parallel applications in such systems needs more deliberations of these systems. In doing so, there are lots of studies including [2-6] which provide insights into the conditions in which efficient performance of clustered systems can be gained.

When multi-core computing nodes are used individually, memories and buses are the main shared resources that contention on them can adversely affect the performance. But when these nodes are connected together to form a clustered system, network interfaces are raised as another important resources. This is because various processes of a parallel job (when placed on different nodes) use these interfaces for communications and synchronizations. In spite of considerable growth in the number of processing cores within computing nodes, the number of available network interfaces almost has been remained unchanged and this number is 1 or 2 for most systems. This issue can lead to high usage of network interfaces in many workloads, especially in workloads which have high inter-process communications. Since network interface port can service just one request at a time, other communication requests received from different cores must be queued to service later. The more cores in a node, the more requests for network interface. As a result, waiting time of messages at interface queue will be increased. This issue can finally prolong the execution time of parallel programs. According to these notes, if we could distribute parallel processes in available computing nodes in a way that requests arriving in each network interface be decreased, queuing time will also be decreased and we can expect performance improvement. In doing so, our goal in this paper, is to present a solution for mapping parallel processes to multi-core clusters in order to reduce network interface contention. After presenting our proposed mapping strategy, we will compare it with some well-known methods and it is shown that the new mapping method can obtain 5% to 90% performance improvement based on used scenarios.

2 Related Works

Various methods have been proposed for mapping parallel processes to processing cores. Of these methods, Blocked and Cyclic are two common approaches which are already investigated in [7-8]. In Blocked method, the mapping is started by selecting a computing node and assigning parallel processes to its free cores one-by-one. When there is no free core, another node will be used and this procedure is repeated until the end of assignment. In Cyclic method, parallel processes are distributed among computing nodes as Round Robin. As a result, maximum number of nodes and minimum number of cores in each node is used in this method (in contrast to Blocked method which uses minimum number of nodes and maximum number of cores in each node).

Although Blocked and Cyclic methods are used in many situations as a default method, but these approaches have little intelligence and do not consider the volume of communications between processes. Because of this issue, other techniques have been proposed which are more intelligent than Blocked and Cyclic. Some of these methods are [9-13]. Proposed mapping algorithm in these studies is based on graph partitioning techniques. The main idea in these techniques is to find processes that communicate to each other frequently and to map them near each other (e.g. place them in the same node). By this way, those processes can benefit from higher bandwidth of memory compared to network interface bandwidth. In order to do this, Application Graph (AG) and Cluster Topology Graph (CTG) are established and then, it is tried to find an efficient mapping from AG to CTG. In AG, vertices represent parallel processes and edges represent communications between processes. In CTG, vertices and edges represent processing cores and available bandwidth between them respectively. Since graph mapping problem is known as NP problem, some heuristics have been introduced which are based on graph partitioning approaches. Dual recursive bipartitioning (DRB) and K-way graph partitioning are two common heuristics. In DRB, AG is divided into two subgroups such that processes which communicate to each other frequently will be grouped in the same subgroup, but processes which communicate to each other infrequently, will be placed in different subgroups. By ‘frequently’ we mean the total volume of data exchanged between each pair of processes. The CTG is also divided into two subgroups in the same way as done with AG. Then, each subgroup of AG is assigned to the peer subgroup of CTG. This operation is repeated on each subgroup recursively until one process in AG or one core in CTG remaines. K-way graph partitioning is the same with DRB except that instead of two groups, graphs are divided into K groups.

Although graph partitioning techniques try to improve performance by mapping frequently communicating processes near each other, but when we try to put such processes near themselves, some shared resources can became performance bottlenecks and these methods are oblivious to this issue. Studies that propose a mapping approach to mitigate contention problem are very limited. Of these studies, we can point to [14-16]. [14] Introduces a mapping algorithm to avoid congestion on Torus interconnection networks. But this study does not consider congestion problem on network interface. In [15] the problem of contention on network interface is investigated. This study tries to put a combination of parallel jobs which have high inter-node communications and low inter-node communications in one node. By this way, network interface contention is alleviated while maximum number of processing cores is used in an efficient way. However this study does not provide a systematic algorithm to use in all scenarios and under every condition. [16] uses a scheduling method to mitigate contention and does not benefit from an intelligent mapping.

3 Proposed Mapping Algorithm

In order to reduce network interface contention, the conditions in which contention is raised, must be recognized. By determining such conditions, we can present the solution. If we could accommodate all processes of a parallel job in just one computing node, there will be no usage of network interface and hence, there is no contention. But when the number of processes is high, or the number of free cores in computing nodes is low, parallel processes must be placed in more than one node inevitably. In this case, high volume of inter-process communications can raise the contention on network interface and hence, degradation of performance will be occurred. To tackle this problem, we should determine a threshold on the number of processes which reside in a node and have high inter-node communication demands. This means that we should distribute processes among available nodes in order to reduce network requests arriving to each interface. Consequently, waiting time at interface queue for inter-node messages will be decreased. In this paper, we tried to determine an appropriate value for threshold using the number of adjacent processes (for each process) and the number of available free cores in computing nodes. Fig. 1 shows our mapping algorithm pseudocode. The first step is to separate parallel jobs based on the length of messages they send. Since larger messages need more service time, processes which send larger messages should use intra-node communications to benefit from high bandwidth of memory. We categorized messages into 3 groups: large messages (1MB or higher), medium messages (2KB to 1MB), and small messages (2KB or less). Based on these categories, we separate parallel jobs. First we select parallel jobs which send large messages (step 1), and then, it is the time to select and map jobs which send medium and small messages respectively (steps 4,6). If processes of a job send messages with different lengths, largest message length is considered for action. After partitioning jobs, parallel jobs in each group are sorted (step 2) based on average number of adjacent processes for each process (Adj_{avg}). Jobs which have more average adjacency are mapped earlier. This is because these jobs may need to distribute between the nodes to have efficient performance. As a result, these jobs should be mapped before other jobs to use available capacity of computing nodes. After choosing a job to map, processes of this job are sorted based on their communication demands and processes which have more communications, are mapped earlier. In proposed algorithm, communication demand for process i is calculated by: $\sum_{j=1, j\neq i}^{P} L_{ij}\lambda_{ij}$ in which, L_{ij} is the length of messages sent from process i to process j (largest length when having different lengths), λ_{ij} is the rate of sending messages from i to j, and P represents number of parallel processes for current job. After determining process with most communication demand (given process 'A'), this process is assigned to a node with most free cores. Then, adjacent processes of 'A' are sorted based on the communication demand between 'A' and them, and it is tried to map adjacent processes of 'A' in the same node as 'A'.

Now, it must be noted that if the number of adjacent processes is high, or the number of available free cores in current node is low, some adjacent processes must be mapped to other computing nodes. In such situations, as mentioned earlier, high

```
New_Mapping_Algorithm( )
Input: Workload graph, Cluster architecture
Output: Mapping information
{
 1. job_pool = select_jobs ( high_length );
 2. sort_jobs ( job_pool );
 3. while ( job_pool is not empty )
 {
  crnt_job = select_job ( job_pool );
  If ( $Adj_{avg}$ <= $FreeCores_{avg}$ -1 )
     No threshold is determined;
  Else
     $Threshold = \left\lfloor \frac{\sum_{i=1}^{P} \frac{Adj_{pi}}{Adj_{\max}}}{num\_of\_nodes} \right\rfloor$ ;
  sort_process ( crnt_job );
  crnt_process = select_process ( crnt_job );
  crnt_node = selec_node ( cluster_arch );
  crnt_socket = select_socket ( cluster_arch );
  map_process ( crnt_process, crnt_node, crnt_socket);
  sort_adj ( crnt_process );
  map_adj_processes ( threshold );
 }
 4. job_pool = select_jobs ( medium_length );
 5. repeat steps 2,3;
 6. job_pool = select_jobs ( small_length );
 7. repeat steps 2,3;
}
```

Fig. 1. Pseudocode of the proposed mapping algorithm

volume of inter-process communications can lead to severe contention on network interface and degrade the performance. So before mapping processes of current job, we should determine a threshold on the number of processes which reside in a node and use network interface for their inter-node communications. To determine the threshold, we act as follow: If average adjacency for processes is less than or equal to average number of free cores ($\mathrm{FreeCores}_{avg}$) in computing nodes (except one processing core which is used to place process 'A'), approximately, we can say that 'A' and its adjacent processes can reside in just one node and there is no significant inter-node communications, probably. In such case, there is no need to determine a threshold. In contrast, if average adjacency is higher than the average free cores, some processes must be placed out of current node. In this case, threshold is determined by:

$$Threshold = \left\lfloor \frac{\sum_{i=1}^{p} \frac{Adj_{pi}}{Adj_{\max}}}{num\_of\_nodes} \right\rfloor \tag{1}$$

In eq. 1, a weight ($\frac{Adj_{pi}}{Adj_{\max}}$) is assigned to each process. In this weighted value, Adj_{pi} represents number of adjacent processes for process pi and Adj_{max} represents maximum adjacency between processes. The reason for choosing a weighted threshold is because high amount of adjacency makes us determine a threshold. Consequently, processes which have more adjacency should have more impact (or weight, as a result) on selected threshold than others. The weighted value is then divided by the number of nodes (num_of_nodes) to distribute processes between all computing nodes. It is to be mentioned that although distributing processes between all cluster nodes, does not always lead to optimum results, but our experiments show that in many scenarios, it can result in efficient performance. An important note about eq. 1 is that if number of computing nodes is more than parallel processes, the threshold will be equal to 0 which is meaningless. In this case, we set the threshold value to 1.

4 Evaluation of the New Mapping Algorithm

4.1 Simulation Testbed

In this paper, we used Omnet++ v4.1 simulator to perform our experiments. The system which we considered for simulation, is a multi-core cluster containing 16 computing nodes which are connected through an intermediate switch. Each computing node has 4 sockets and each socket is a 4-core processor, so each node contains 16 processing cores. The architecture of each node is based on the NUMA[1] architecture. This means that each socket can access to its local memory (although it can also access to remote memories but with more latency). In each node, we used a network interface with InfiniBand technology. InfiniBand, is one of the most advanced technologies which is used to establish high performance clusters. Table 1 lists the parameters we used in our simulations.

[1] Non Uniform Memory Access.

Table 1. Simulation parameters

Parameter	Value
Main memory bandwidth	4GB/s
Remote memory access latency	10% more than local memory access latency
Cache bandwidth (for intra-chip communications)	Corresponds to AMD Opteron 2352 chip
Maximum length of common buffer in cache	1MB
Network interface bandwidth	1GB/s (corresponds to InfiniHost MT23108 4x)
Switching latency at intermediate switch	100ns (independent of message length)

4.2 Experimental Results

To evaluate the new mapping method, we used synthetic and real workloads. In synthetic workloads, messages which had different lengths and rates were generated. In these traffics, we used four different communication patterns between parallel processes. These patterns which are based on communication patterns in message passing libraries are: Bcast/Scatter, Gather/Reduce, All-to-All and Linear. In Bcast/Scatter, one process as the root process broadcasts its messages to other processes and other processes are just receiver. In Gather/Reduce, one process as the root process, receives messages from other processes and other processes are just senders. In All-to-All, each process sends messages to all other processes. In Linear, each process receives messages from a previous process and sends its messages to a next process (there is a linear communication pattern between processes). Tables 2 to 5 show the definition of 4 synthetic workloads which each, contains a number of parallel jobs with different communication patterns. Real workloads were extracted from communication behavior of NPB[2] benchmarks. Tables 6 to 9 show the definition of 4 real workloads which each, contains some benchmarks with different number of processes and different benchmark classes.

For performance evaluation, we used sum of the waiting times of messages at server queues (network interface and memory) as our main metric. We compared our results with the results obtained from Blocked, Cyclic and DRB methods. Fig. 2 shows

Table 2. Synt_workload_1

Job	No. of Processes	Pattern	Length	Rate	Message Count
0	64	All-to-All	64KB	100m/s	2000
1	64	Bcast/Scatter	64KB	100m/s	2000
2	64	Gather/Reduce	64KB	100m/s	2000
3	64	Linear	64KB	100m/s	2000

Table 3. Synt_workload_2

Job	No. of Processes	Pattern	Length	Rate	Message Count
0	64	All-to-All	2MB	10m/s	2000
1	64	Bcast/Scatter	2MB	10m/s	2000
2	64	Gather/Reduce	2MB	10m/s	2000
3	64	Linear	2MB	10m/s	2000

[2] NAS Parallel Benchmarks.

Table 4. Synt_workload_3

Job	No. of Processes	Pattern	Length	Rate	Message Count
0	32	All-to-All	2MB	10m/s	2000
1	32	Bcast/Scatter	2MB	10m/s	2000
2	32	Gather/Reduce	2MB	10m/s	2000
3	32	Linear	2MB	10m/s	2000
4	32	All-to-All	64KB	10m/s	2000
5	32	Bcast/Scatter	64KB	10m/s	2000
6	32	Gather/Reduce	64KB	10m/s	2000
7	32	Linear	64KB	10m/s	2000

Table 5. Synt_workload_4

Job	No. of Processes	Pattern	Length	Rate	Message Count
0	24	All-to-All	2MB	10m/s	2000
1	24	Bcast/Scatter	2MB	10m/s	2000
2	24	Gather/Reduce	2MB	10m/s	2000
3	24	Linear	2MB	10m/s	2000
4	24	All-to-All	64KB	10m/s	2000
5	24	Bcast/Scatter	64KB	10m/s	2000
6	24	Gather/Reduce	64KB	10m/s	2000
7	24	Linear	64KB	10m/s	2000

Table 6. Real_workload_1

Job	No. of Processes	Benchmark	Class
0	25	SP	C
1	32	IS	C
2	32	FT	B
3	16	FT	B
4	16	IS	C
5	32	CG	C
6	8	IS	B
7	25	BT	C
8	16	CG	B

Table 7. Real_workload_2

Job	No. of Processes	Benchmark	Class
0	8	IS	B
1	32	FT	B
2	32	IS	C
3	32	MG	C
4	32	CG	C
5	32	IS	B
6	32	MG	B
7	32	CG	B
8	16	BT	C

Table 8. Real_workload_3

Job	No. of Processes	Benchmark	Class
0	25	BT	B
1	32	CG	B
2	32	EP	B
3	32	FT	B
4	32	IS	B
5	25	LU	B
6	32	MG	B
7	25	SP	B

Table 9. Real_workload_4

Job	No. of Processes	Benchmark	Class
0	25	SP	C
1	32	CG	C
2	32	EP	C
3	32	MG	C

the performance results for 4 synthetic workloads and 4 real workloads. In this figure, ‘B’ indicates Blocked, ‘C’ indicates Cyclic, ‘D’ indicates DRB, and ‘N’ indicates our new mapping algorithm.

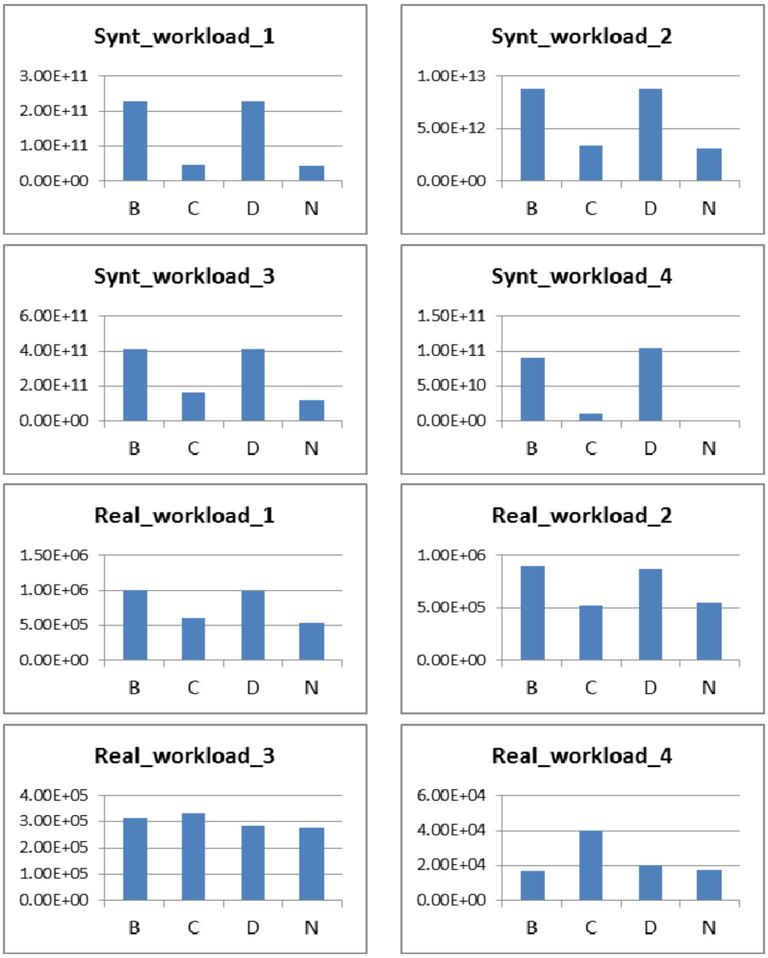

Fig. 2. Waiting time of messages for synthetic and real workloads (in mili-seconds)

According to Fig.2 it can be seen that the new mapping strategy, has produced better results compared to other methods. In synthetic workloads, the number of processes in parallel jobs is more than the number of processing cores within a node. Besides, significant part of communications is due to jobs which have All-to-All patterns. These factors cause synthetic workloads to be heavy communicating workloads. In such workloads, the Blocked technique which tries to accommodate parallel processes in minimum number of nodes, has led to severe contention on network interface and has unacceptable performance, consequently. In contrast, Cyclic has gained better performance by distributing processes among computing nodes. Since in DRB method, processes which are communicating frequently, are mapped near each other, process mapping is done as Blocked and the results are not efficient. The reason that the new method has performed more efficient than Cyclic is that in the new algorithm, efficient mapping conditions is determined for each parallel job independent of other jobs. In other words, if the amount of adjacency and communications between processes, is high, the new method will distribute the processes, otherwise it acts like Blocked. Based on performance results, the new mapping technique has gained performance improvements up to 5%, 8%, 29% and 91% for Synt_workload_1 to Synt_workload_4 respectively (performance gain is calculated compared to the best result from other methods, i.e. Cyclic in here). In Real_workload_1 and Real_workload_2 scenarios, IS and FT benchmarks were used more than other benchmarks. These benchmarks have high communications and their communication pattern is All-to-All entirely. As a result, the above mentioned workloads are heavy and as can be seen in Fig. 2, the Cyclic method has performed better than the Blocked and DRB methods. In these workloads, the new approach has acted as efficient as Cyclic and even better (in Real_workload_1 scenario, 11% performance improvement is observed). In order to show that our approach can perform efficiently not only in heavy workloads, but also in non-heavy workloads, Real_workload_3 and Real_workload_4 were used. Real_workload_3 is a medium workload in term of communications and as can be seen in Fig. 2, there is no significant difference between performance results of different methods for this scenario. Despite this, the new mapping technique has performed a little bit better than others. Real_workload_4 is a scenario which has light communications and as we can expect, Blocked and DRB methods have better results than Cyclic. Performance results for this scenario show that the new mapping method has performed as well as Blocked which indicates that the new approach can have efficient results even in light communicating workloads.

5 Conclusion

In this paper, we proposed a new process mapping algorithm to assign parallel processes to multi-core clusters aimed to reducing network interface contention. Since the number of processing cores within recent computing nodes is growing up rapidly, contention on shared resources is posing itself as a serious challenge and should be considered for optimizing performance. Here, we tackled this problem and proposed a process placement algorithm to alleviate contention on network interface as one of the main shared resources. We compared our technique with other well-known methods and observed that improved performance was gained (5% to 90%) in experimental workloads. Our mapping algorithm is easy to implement and its efficiency makes it usable in recent high performance multi-core clusters.

Acknowledgments. We thankfully appreciate Research Institute for ICT (ITRC) of Iran for supporting us in this project. We hope this work be a valuable research to extend the technological knowledge of this institute.

References

1. http://www.top500.org
2. Hood, R., Jin, H., Mehrotra, P., Chang, J., Djomehri, J., Gavali, S., Jespersen, D., Taylor, K., Biswas, R.: Performance Impact of Resource Contention in Multicore Systems. In: IEEE International Symposium on Parallel and Distributed Processing, Atlanta (2010)
3. Chai, L., Gao, Q., Panda, D.K.: Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In: 7th IEEE International Symposium on Cluster Computing and the Grid, Rio De Janeiro, Brazil (2007)
4. Jokanovic, A., Rodriguez, G., Sancho, J.C., Labarta, J.: Impact of Inter-Application Contention in Current and Future HPC Systems. In: IEEE Annual Symposium on High-Performance Interconnects, Mountain View, U.S.A (2010)
5. Kayi, A., El-Ghazawi, T., Newby, G.B.: Performance issues in emerging homogeneous multi-core architectures. Elsevier Journal of Simulation Modeling Practice and Theory 17(9) (2009)
6. Narayanaswamy, G., Balaji, P., Feng, W.: Impact of Network Sharing in Multi-core Architectures. In: 17th IEEE International Conference on Computer Communications and Networks, Virgin Islands, U.S.A (2008)
7. Dummler, J., Rauber, T., Runger, G.: Mapping Algorithms for Multiprocessor Tasks on Multi-core Clusters. In: 37th IEEE International Conference on Parallel Processing, Portland, U.S.A (2008)
8. Ichikawa, S., Takagi, S.: Estimating the Optimal Configuration of a Multi-Core Cluster: A Preliminary Study. In: IEEE International Conference on Complex, Intelligent and Software Intensive Systems, Fukuoka, Japan (2009)
9. Chen, H., Chen, W., Huang, J., Robert, B., Kuhn, H.: MPIPP: An Automatic Profile-guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters. In: 20th Annual International Conference on Supercomputing, New York, U.S.A (2006)
10. Mercier, G., Clet-Ortega, J.: Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments. In: 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Berlin, Germany (2009)
11. Rodrigues, E.R., Madruga, F.L., Navaux, P.O.A., Panetta, J.: Multi-core Aware Process Mapping and Its Impact on Communication Overhead of Parallel Applications. In: IEEE Symposium on Computers and Communications, Sousse, Tunisia (2009)
12. Khoroshevsky, V.G., Kurnosov, M.G.: Mapping Parallel Programs into Hierarchical Distributed Computer Systems. In: 4th International Conference on Software and Data Technologies, Sofia, Bulgaria (2009)
13. Jeannot, E., Mercier, G.: Near-Optimal Placement of MPI Processes on Hierarchical NUMA Architectures. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Part II. LNCS, vol. 6272, pp. 199–210. Springer, Heidelberg (2010)
14. Agrawal, T., Sharma, A., Kale, L.V.: Topology-Aware Task Mapping for Reducing Communication Contention on Large Parallel Machines. In: 20th IEEE International Symposium on Parallel and Distributed Processing, Rhodes Island, Greece (2006)
15. Koop, M.J., Luo, M., Panda, D.K.: Reducing Network Contention with Mixed Workloads on Modern Multicore Clusters. In: IEEE International Conference on Cluster Computing and Workshops, New Orleans, U.S.A (2009)
16. Koukis, E., Koziris, N.: Memory and Network Bandwidth Aware Scheduling of Multiprogrammed Workloads on Clusters of SMPs. In: 12th International Conference on Parallel and Distributed Systems, Minneapolis, U.S.A (2006)