Improving inter-node communications in multi-core clusters using a contention-free process mapping algorithm

Mohsen Soryani, Morteza Analoui & Ghobad Zarrinchian

Improving inter-node communications in multi-core clusters using a contention-free process mapping algorithm

Mohsen Soryani · Morteza Analoui ·
Ghobad Zarrinchian

Abstract High performance clusters, which are established by connecting many computing nodes together, are known as one of main architectures to obtain extremely high performance. Currently, these systems are moving from multi-core architectures to many-core architectures to enhance their computational capabilities. This trend would eventually cause network interfaces to be a performance bottleneck because these interfaces are few in number and cannot handle multiple network requests at a time. The consequence of such issue would be higher waiting time at the network interface queue and lower performance. In this paper, we tackle this problem by introducing a process mapping algorithm, which attempts to improve inter-node communications in multi-core clusters. Our mapping strategy reduces accesses to the network interface by distributing communication-intensive processes among computing nodes, which leads to lower waiting time at the network interface queue. Performance results for synthetic and real workloads reveal that the proposed strategy improves the performance from 8 % up to 90 % in tested cases compared to other methods.

Keywords High performance clusters · Network interface · Inter-node communications · Process mapping algorithm

1 Introduction

Parallel processing is one of the basic approaches to achieve high computing power required by many scientific and economic applications. For this purpose, various ar-

M. Soryani · M. Analoui · G. Zarrinchian (✉)
Iran University of Science and Technology, Hengaam street, Resaalat square, Naarmak, Tehran, Iran
e-mail: zarrinchian@comp.iust.ac.ir

M. Soryani
e-mail: soryani@iust.ac.ir

M. Analoui
e-mail: analoui@iust.ac.ir

chitectures have been proposed in which cluster computing has gained more popularity. Based on [1], up to 81 % of 500 top supercomputers in the world use these systems as their main architecture. Another trend in this realm is the advances in multi-core processors technology that make them an excellent choice to use in clustered nodes. Westmere series of Intel Xeon and Magny-cours series of AMD Opteron, with 10 and 12 cores per chip, respectively, are examples of multi-core processors, which are becoming common in recent computing nodes.

Although multi-core processors can improve computational capabilities, they also introduce some challenges. The main challenge in this regard is the contention of different processing cores on shared resources, like memory and buses. Under such condition, shared resources can be a performance bottleneck, and can adversely affect the performance of running applications. Consequently, efficient execution of parallel applications in such distributed systems needs more deliberations on the performance of these systems. In doing so, there are a lot of studies including [2–6], which provide insights into the conditions in which efficient performance of clustered systems can be obtained.

In spite of considerable growth in the number of processing cores within the computing nodes, the number of available network interfaces has remained fixed 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 with high volume of interprocess communications. Since a network interface can service just one request at a time, other communication requests must be queued to service later. Higher number of cores in a node results in more accesses to the network interface. Therefore, waiting time of inter-node messages at the network interface queue will be increased, which in turn, prolongs the execution time of parallel programs. Thus, we can distribute parallel processes among the nodes in a way that the queue length in each network interface reduces considerably. This is important for the processes with high communication demand.

This paper presents a solution for mapping parallel processes into multi-core clusters in order to improve inter-node communications and to achieve high performance when running parallel applications with high volume of network requests. For this purpose, we have used queuing network theory to extract an analytical model which represents inter-node communication time in a limited-size cluster. Using this model, we determine affecting factors on communication time, and extract some basic rules to use in the new mapping algorithm. Finally, we evaluate the performance of our approach using the simulation for synthetic and real multi-job workloads. We show that the proposed method can improve the performance in communication-intensive workloads from 8 % up to 90 % compared to the results obtained from other well-known mapping methods.

1.1 Related works

There are a lot of works which have investigated improving communication performance in clustered systems. Some works, as stated in [7], have focused on improving MPI[1] libraries to obtain efficient performance. An example of such works is a study

[1]Message passing interface.

of Chai et al. [8]. Some other works offer communication improvement for special interconnection networks. Rex et al. in [9], for example, have tried to improve the performance of InfiniBand interconnection networks. In this study, the data to be sent is located in huge pages, which requires less page address translation.[2] Consequently, the time needed for memory registration operation,[3] and hence communication time is reduced. Although methods mentioned earlier are worthy of consideration, the main approach to improve communication performance is to use efficient process mapping techniques when assigning parallel processes to processing cores. Such techniques have significant impact on improving performance, and hence they have attracted more attention compared to the other methods.

Blocked and Cyclic are two common methods for mapping processes in clustered systems. These techniques have been investigated in some studies such as [6, 10, 11]. In the Blocked method, process mapping is started by selecting a computing node and assigning parallel processes to its idle cores one-by-one. When there is no idle core to continue, another computing node is used and this procedure is repeated until the end of assignment. In the Cyclic method, parallel processes are distributed among computing nodes in a round-robin fashion. Although Blocked and Cyclic methods are used as the default process placement policy in many situations, these approaches have little inherent intelligence. Moreover, in a multi-job workload, the efficiency of these techniques may be dependent on the order in which parallel jobs fall within the workload.

Beyond the Blocked and Cyclic, there are some other advanced methods in which [7, 12–14] are the most important ones. The proposed mapping technique 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 place them near each other (e.g. to place them in the same node). In this way, those processes can profit from higher bandwidth of main memory compared to network interface bandwidth. In order to do this, two graphs are created: Application Graph (AG) and Cluster Topology Graph (CTG). For the AG, vertices represent parallel processes, and edges represent communications between parallel processes. For the CTG, vertices and edges represent cores and available bandwidth between corresponding cores, respectively. Using these two graphs, the problem of assigning parallel processes into processing cores is converted to a graph mapping problem, i.e. finding an efficient mapping from AG to CTG. Since the graph mapping problem is known as a NP problem [15], some heuristics have been introduced, which are based on graph partitioning concepts. DRB[4] and K-way Graph Partitioning are two common heuristics. While the DRB method divides AG into two sub-graphs, K-way graph partitioning divides it into K sub-graphs. Each sub-graph contains frequently communicating processes. By ‘frequently’, we mean the total volume of data exchanged between each pair of processes. The CTG is also divided into sub-graphs in the same way as for AG. Each sub-graph of AG is then assigned to the peer sub-graph of CTG. This operation

[2]Page address translation is a procedures done in InfiniBand networks before sending data.

[3]Memory registration operation is a procedure done in InfiniBand networks before sending data.

[4]Dual recursive bi-partitioning.

is repeated on each sub-graph recursively until one process in AG or one processing core in CTG remains.

The main issue related with graph partitioning techniques is that they may cause shared resources to be a performance bottleneck, when mapping frequently communicating processes near each other. Currently, there are few studies that propose a mapping approach to remove such bottlenecks. References [16–19] are some of these studies. Agrawal et al. in [16] propose a mapping algorithm to avoid congestion on Torus interconnection networks. This study just pays attention to the problem of contention on the interconnection network, and does not consider the matter on the network interface. Koop et al. in [17] address the network interface contention problem by putting a combination of parallel jobs with high inter-node communications and low inter-node communications within one computing node. In this way, contention is alleviated while the maximum number of processing cores is used in an efficient way. However, this study does not provide a systematic mechanism to use in all scenarios and under every condition. In [18], a scheduling method is used to mitigate contention on memory and network interface. However, this method does not employ an intelligent mapping approach. Because of lack of a systematic approach to address the contention issue, we already proposed a mapping strategy to reach the goal [19]. In that study, we tried to limit the number of network interface accesses by determining a threshold on the number of processes which send inter-node messages. Although we obtained efficient results in our previous paper, the way in which we determined the threshold value was just an approximation, and we had not considered communication characteristics of workloads and important cluster-specific parameters, such as the bandwidth of main memory or network interface. Moreover, the condition in which we determined the threshold value in the previous paper had some inefficiency, which is addressed in this paper. Briefly speaking, this study is a complete revision of our previous work, which tries to present a systematic method to address the contention problem.

1.2 Paper organization

This paper is organized as follows: In Sect. 2, an analytical model for a hypothetical clustered system is extracted by the help of queuing network theory. This model is then used in Sect. 3 to extract some basic rules. In Sect. 4, we propose our new mapping algorithm to improve inter-node communications. Performance evaluation results for synthetic and real workloads as well as simulation results are presented in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Analytical modeling of a clustered system

In this section, an analytical model for a hypothetical clustered system is extracted by the help of queuing network theory. This model is then used to extract some basic rules, which are employed in the proposed mapping strategy. The clustered system for which we present an analytical model is a system containing 16 computing nodes. These nodes are connected to one another using an intermediate switch and form a

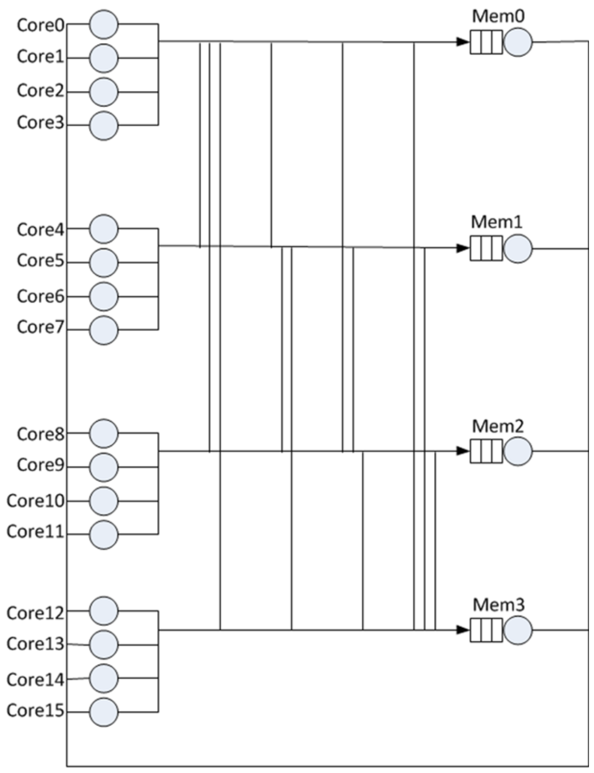


Fig. 1 Queuing network model for processing cores and main memories in a given node

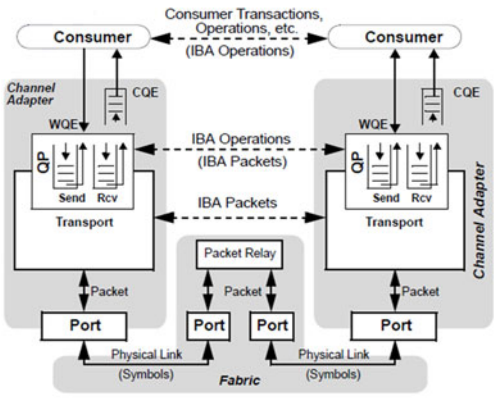
star topology. There are 4 sockets in each computing node and each socket represents a 4-core chip. As a result, each node contains 16 processing cores. We used NUMA[5] as the architecture of each node because it is more efficient than SMP[6] and is used in recent systems. In NUMA architecture, there is a local memory for each socket while remote memories can be accessed with more latency. A queuing network corresponding to internal components of a computing node is shown in Fig. 1.

Network interface is one of the most important components in each computing node. In this paper, we have chosen InfiniBand technology for the interface (for the rest of the paper, we refer to the 'network interface' by just 'interface'). A schematic of internal functionality for this kind of interface is depicted in Fig. 2a. According to this figure, when a process needs to send or receive data, a pair of communication queues (which is called Queue Pair or QP for short) is established for the process in the interface. One of these two queues is used for sending messages (Send Queue) and the other is used for receiving messages (Rcv Queue). When a send request is placed in the Send Queue, a unit called DMA[7] verifies the request and then refers to the memory to fetch the requested data to be sent. The DMA then takes care of the

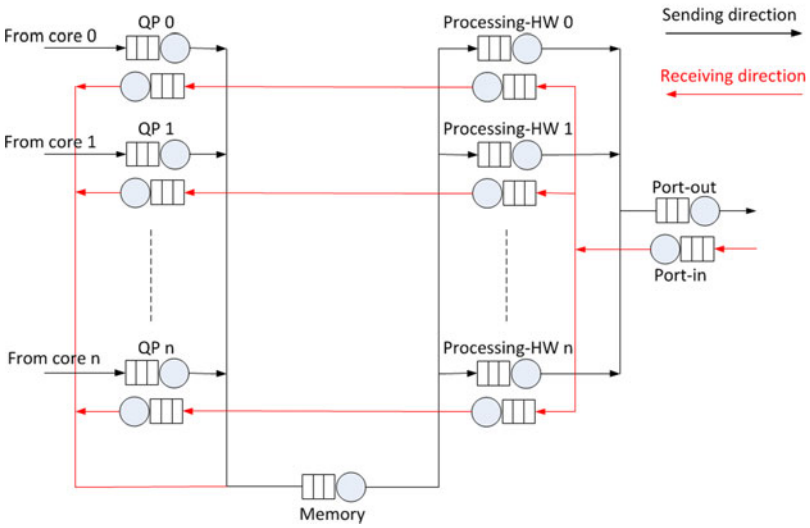
[5]Non-uniform memory access.

[6]Symmetric multi-processors.

[7]Direct memory access.



a) Schematic of internal functionality [20]



b) A queuing network model

Fig. 2 A model for an InfiniBand network interface

request to perform other required processing procedures. When receiving a request, the DMA goes through similar actions (but in reverse order).

For the rest of paper, we consider some assumptions in our modeling. First, hardware processing units, which prepare messages to send as packets are denoted by "Processing-HW". Second, since there are multiple DMA and multiple packet processing units in recent interfaces, we assume that each QP has its own DMA and Processing-HW units. Figure 2b shows a queuing network model we have used to model InfiniBand network interface. In order to model intermediate switch, we considered a simple model in which two servers are used for each port of the switch: one for sending direction and the other for receiving direction.

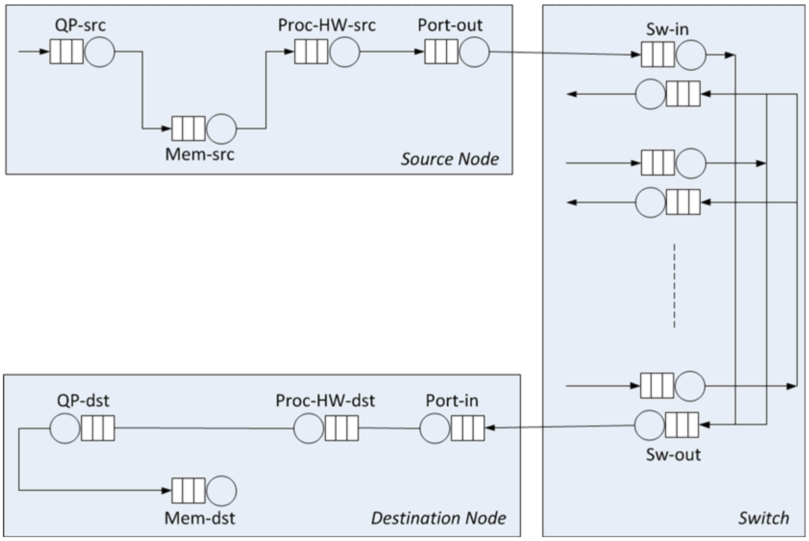


Fig. 3 The path of inter-node communications

Since our main goal is to improve inter-node communications, we should extract an analytical model representing inter-node communication time. For this purpose, the path of transferring inter-node messages and servers in this path must be considered. According to the queuing network models, the path for transferring inter-node messages corresponds to the one shown in Fig. 3. According to the path shown, the time taken to send a message (given a message in class ‘r’) can be calculated as shown below:

$$T_{\mathrm{r}}^{\mathrm{src}\to\mathrm{dst}} = T_{\mathrm{r}}^{\mathrm{src}} + T_{\mathrm{r}}^{\mathrm{sw}} + T_{\mathrm{r}}^{\mathrm{dst}} \tag{1}$$

In Eq. (1), $T_{\mathrm{r}}^{\mathrm{src}}$ is the time spent at the source node for sending message ‘r’, $T_{\mathrm{r}}^{\mathrm{sw}}$ is the time spent at the intermediate switch, and $T_{\mathrm{r}}^{\mathrm{dst}}$ is the time spent at the destination node for receiving message ‘r’. Since the time spent at a given node (or at the intermediate switch) equals to the sum of the elapsed time at different servers of that node, each of the timing parameters in Eq. (1) can be rewritten as Eqs. (2)–(4). In these equations, ‘src’ and ‘dst’ indices are used to imply corresponding servers at the source and destination nodes, respectively.

$$T_{\mathrm{r}}^{\mathrm{src}} = R_{\text{QP-src,r}} + R_{\text{Mem-src,r}} + R_{\text{Proc-hw-src,r}} + R_{\text{Port-out,r}} \tag{2}$$

$$T_{\mathrm{r}}^{\mathrm{sw}} = R_{\text{Sw-in,r}} + R_{\text{Sw-out,r}} \tag{3}$$

$$T_{\mathrm{r}}^{\mathrm{dst}} = R_{\text{Port-in,r}} + R_{\text{Proc-hw-dst,r}} + \mathrm{R}_{\text{QP-dst,r}} + R_{\text{Mem-dst,r}} \tag{4}$$

In Eqs. (2–(4), $R_{X,\mathrm{r}}$ indicates residence time of messages in class ‘r’ at server X. $R_{X,\mathrm{r}}$ is calculated as below [21]:

$$R_{X,\mathrm{r}} = W_{X,\mathrm{r}} + S_{X,\mathrm{r}} \tag{5}$$

In Eq. (5), $W_{X,\mathrm{r}}$ and $S_{X,\mathrm{r}}$ are average queue time and average service time at server X for the message in class 'r', respectively. To compute $W_{X,\mathrm{r}}$, we used the G/M/1 model in which inter-arrival time and service time of messages are considered to have general and exponential distributions, respectively. In the G/M/1 model, there is no exact formulation to calculate $W_{X,\mathrm{r}}$, so we should limit ourselves to an approximating one. One suitable approximation extracted from G/G/1 model is [21]:

$$W_{X,\mathrm{r}} \approx \frac{C_\mathrm{a}^2 + \rho_{X,\mathrm{r}}^2}{1 + \rho_{X,\mathrm{r}}^2} \frac{\rho_{X,\mathrm{r}} S_{X,\mathrm{r}}}{1 - \rho_{X,\mathrm{r}}} \tag{6}$$

In Eq. (6), C_a is the coefficient of variations (CV) of message inter-arrival time, and $\rho_{X,\mathrm{r}}$ is the utilization of corresponding server. Based on utilization law [21], utilization of a server can be obtained by $\rho_{X,\mathrm{r}} = \lambda_{X,\mathrm{r}} S_{X,\mathrm{r}}$ in which $\lambda_{X,\mathrm{r}}$ is the arrival rate of messages of class 'r' to server X. Moreover, total utilization of a server is the sum of its utilizations caused by different workload classes. So, we used Eqs. (7)–(12) to obtain total utilization of different servers. In these equations, we assume that there are entirely CL_msg workload classes for messages and $\mathrm{CL}_\mathrm{norm}$ workload classes for normal memory accesses caused by different processing cores (by 'normal memory accesses' we mean memory accesses done for fetching a word from memory and not for transferring a message). We also assume that messages sent by a given core are entirely of the same class. Based on this assumption and because each core has access to its own QP and Proc-HW servers, corresponding QP and Proc-HW servers for a given core are utilized by just one workload class. This issue is formulated as Eqs. (7) and (8).

$$\rho_\mathrm{QP} = \rho_\mathrm{QP,r} \tag{7}$$

$$\rho_\mathrm{Proc\text{-}hw} = \rho_\mathrm{Proc\text{-}hw,r} \tag{8}$$

Since the main memory is used for fetching data as well as transferring messages, the memory unit is utilized by both workload classes, i.e. message classes and normal memory access classes (Eq. (9)). Message classes are further divided into three subclasses: inter-node messages sent to other nodes, inter-node messages received from other nodes, and messages sent as intra-node messages. So, the utilization of main memory can be considered as Eq. (10). In this equation, $\mathrm{CL}_\mathrm{out\text{-}msg}$, $\mathrm{CL}_\mathrm{in\text{-}msg}$, and $\mathrm{CL}_\mathrm{intra\text{-}msg}$ indicate the number of message classes for outgoing messages, incoming messages, and intra-node messages, respectively.

$$\rho_\mathrm{Mem} = \sum_{i=1}^{\mathrm{CL}_\mathrm{msg}} \rho_{\mathrm{Mem},i} + \sum_{j=1}^{\mathrm{CL}_\mathrm{norm}} \rho_{\mathrm{Mem},j} \tag{9}$$

$$\rho_\mathrm{Mem} = \sum_{i=1}^{\mathrm{CL}_\mathrm{out\text{-}msg}} \rho_{\mathrm{Mem},i} + \sum_{j=1}^{\mathrm{CL}_\mathrm{in\text{-}msg}} \rho_{\mathrm{Mem},j} + \sum_{k=1}^{\mathrm{CL}_\mathrm{intra\text{-}msg}} \rho_{\mathrm{Mem},k} + \sum_{l=1}^{\mathrm{CL}_\mathrm{norm}} \rho_{\mathrm{Mem},l} \tag{10}$$

The utilization of output port and input port of the network interface is simply obtained by Eqs. (11) and (12). It must be noted that the same equations are applicable for 'Sw-in' and 'Sw-out' servers, respectively. This is because messages from 'Port-out' are transmitted directly to 'Sw-in', and messages received at 'Port-in', come directly from 'Sw-out' (Fig. 3). The only difference is that instead of interface's service time, switch's service time must be considered when calculating utilization.

$$\rho_{\text{Port-out}} = \sum_{i=1}^{\text{CL}_{\text{out-msg}}} \rho_{\text{Port-out},i} \tag{11}$$

$$\rho_{\text{Port-in}} = \sum_{i=1}^{\text{CL}_{\text{in-msg}}} \rho_{\text{Port-in},i} \tag{12}$$

At this stage, we have required timing model for inter-node communications. In the next section, we use this model to conduct some experiments and to extract some basic rules, which will be used later in our proposed mapping strategy.

3 Process mapping rules

In this section, we conduct various experiments to extract a few simple and basic rules using the analytical model we previously obtained. These rules, which are used in the proposed mapping strategy, are representative of conditions in which efficient inter-node communications is reachable. In the rest, we consider some assumptions for all of the experiments: memory bandwidth is equal to 4 GB/s, network interface bandwidth is 1 GB/s (corresponds to the performance of InfiniHost MT23108 4X), switching latency at the intermediate switch is 100 ns (independent of message size) and the rate of normal memory accesses caused by each core is 100,000 req/s. We also assume that the data to be sent by each core resides in the local memory of socket containing that core.

The first experiment investigates the impact of message rates and sizes on the communication time. In this experiment, one processing core sends messages out to another node. Figure 4 shows the average time required to send a message for different rates and sizes of messages. In this figure, we assume $C_{\text{a}} = 1$. The figure shows that the message length has more impact on the communication time than the message rate. For example, the time taken to send a 512 kB message at the rate 10 req/s is 2565 µs. When the length is doubled, the sending time becomes 5333 µs, which is as twice as before. But when the rate is increased even to 5 fold, the sending time has negligible growth (just 5 % increase in communication time). According to the figure, we can see that increasing the rate is just significant for large messages (messages larger than 2 MB) and not for medium or small messages. Based on these observations, we can extract the first basic rule:

Rule #1 *In order to have efficient communications, the message length should have more contribution to the mapping solution compared to the message rate.*

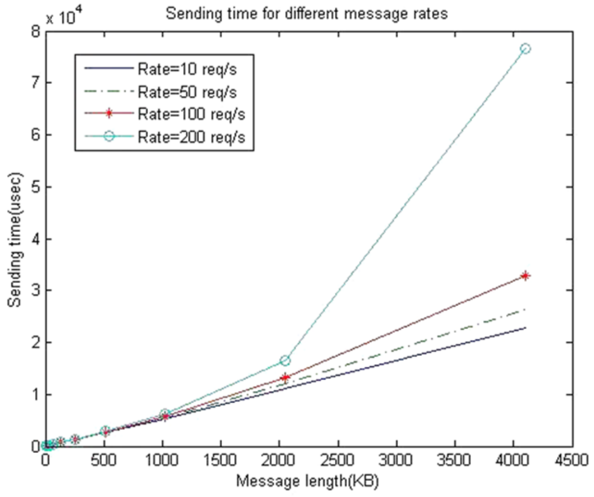


Fig. 4 Time taken to send inter-node messages with different sizes and rates

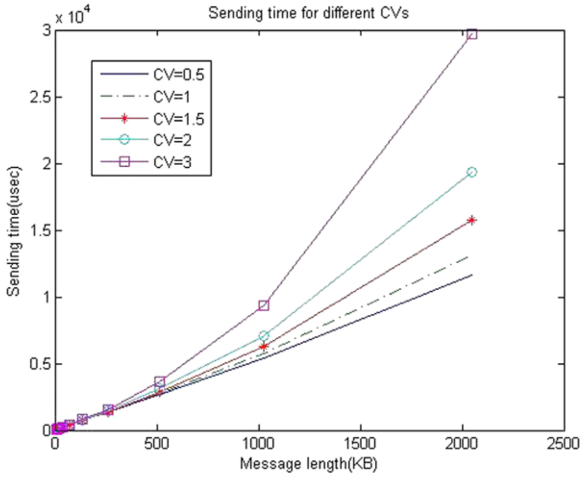


Fig. 5 Time taken to send inter-node messages for different CV values

The second experiment investigates the impact of CV (Coefficient of Variations of message inter-arrival time) on the communication time. It is obvious that higher CV causes higher latency. But in this experiment we want to precisely measure the effect of CV values. Figure 5 shows the impact of CV values on the transferring time for different message sizes (the rate is fixed at 100 req/s). As shown in the figure, increasing the CV has considerable impact on message sending time for large messages. This impact is less for medium messages and negligible for small messages. For example, by increasing the CV value from 0.5 to 3 for 1 MB and 64 kB messages at the rate 100 req/s, sending time will be increased by up to 75 % and 5 %, respectively. This increase is negligible for small messages (0.2 % increase for 1 kB messages). So, the second basic rule can be extracted as follows:

Rule #2 *While the CV of message inter-arrival time for small messages can be ignored, it has drastic impact on the efficiency of communications for medium and large messages, and hence it should be considered in the mapping solution.*

The third experiment investigates the impact of two different inter-node message classes on each other when these messages are sent out from a node. In this experiment, two different processing cores, which are located in the same computing node and in the same socket, send their messages to another node and slowdown in communication time (which is caused by simultaneously sending messages) is measured. The upper values in each cell of Table 1 show the percentage of slowdown, when $C_a = 1$. For example, when the two processing cores send 64 kB messages at the rate 1000 req/s, they prolong their communication time by up to 1.27 % (on average) compared to the case in which just one core send its messages. As another example, when 1 MB messages at the rate 10 req/s are sent beside 64 kB messages at the rate 10 req/s, 64 kB messages experience 0.18 % slowdown while 1 MB messages experience just 0.019 % slowdown. Based on the results, we see that the small and medium messages incur negligible slowdown on each other when sending together. However, when these messages are sent together with large messages, they would experience high slowdown. This issue is also observable for other CV values. Higher CV value causes higher slowdown. Based on these observations, we can extract our third rule:

Rule #3 *Because of drastic effect of large messages on other messages, we should isolate the communication path of large messages from the path of other messages. For performance reasons, large messages should be transferred within the node as intra-node messages.*

Our fourth experiment is similar to the third one, except that one of the two cores sends intra-node messages. The lower values in each cell of Table 1 show the percentage of slowdown in inter-node communication time caused by intra-node communications. In this sense, messages on each column and on each row represent intra-node and inter-node messages, respectively. Clearly, since the two processing cores are just competing to access the main memory, the slowdown values are less than the ones obtained in the previous experiment. In this experiment, just like the previous one, small and medium messages incur less slowdown compared to the large messages. So, the fourth rule is as follows:

Rule #4 *Parallel processes which reside within the same computing node and send large inter-node messages, should be assigned to different sockets. This way, contention on the memory (and consequently on the interface) will be alleviated and the slowdown in communication time caused by large messages will be reduced.*

Our last experiment studies the impact of message length and message rate on the network interface utilization. As Eq. (6) implies, more server utilization causes more waiting time, which indicates the importance of utilization parameter. Figure 6 shows the utilization of interface output port for different messages. According to

Table 1 The percentage of slowdown caused by different messages on each other

	1 kB–10 req/s	1 kB–100 req/s	1 kB–1000 req/s	64 kB–10 req/s	64 kB–100 req/s	64 kB–1000 req/s	1 MB–10 req/s	1 MB–100 req/s	1 MB–500 req/s
1 kB–10 req/s	0.005	0.005	0.012	0.01	0.05	0.48	0.077	0.8	7.12
	0.004	0.004	0.005	0.005	0.007	0.031	0.009	0.047	0.23
1 kB–100 req/s	0.028	0.029	0.035	0.033	0.073	0.503	0.1	0.82	7.15
	0.028	0.028	0.029	0.029	0.031	0.054	0.032	0.07	0.261
1 kB–1000 req/s	0.058	0.058	0.065	0.062	0.103	0.532	0.13	0.85	7.18
	0.057	0.057	0.057	0.057	0.06	0.08	0.062	0.1	0.3
64 kB–10 req/s	0.001	0.002	0.017	0.012	0.112	1.18	0.18	1.97	17.76
	0.0009	0.0009	0.002	0.001	0.007	0.062	0.01	0.1	0.552
64 kB–100 req/s	0.002	0.003	0.018	0.013	0.114	1.19	0.18	1.98	17.97
	0.002	0.002	0.003	0.002	0.008	0.063	0.011	0.1	0.552
64 kB–1000 req/s	0.002	0.004	0.02	0.014	0.122	1.27	0.19	2.12	20.47
	0.002	0.002	0.003	0.003	0.008	0.062	0.011	0.1	0.54
1 MB–10 req/s	0	0	0.019	0.019	0.131	1.4	0.2	2.36	21.43
	0	0	0	0	0	0.075	0.019	0.112	0.66
1 MB–100 req/s	0	0	0.017	0.017	0.14	1.6	0.24	2.66	27
	0	0	0	0	0	0.07	0.017	0.1	0.63
1 MB–500 req/s	0	0.011	0.056	0.033	0.32	3.6	0.51	6.32	#
	0	0	0	0	0.011	0.056	0.011	0.09	0.51

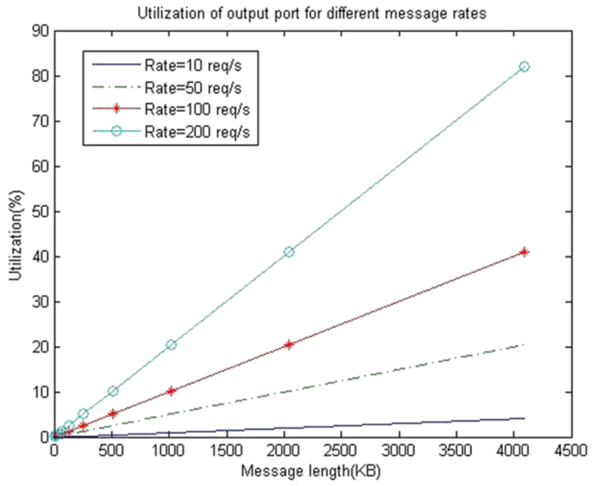


Fig. 6 Utilization of interface output port for different messages

this figure, the size and rate parameters have equal effect on the utilization of interface, which is obvious based on the utilization law ($\rho = \lambda S$). As can be seen from the figure, medium and large messages, especially at high rates, can significantly increase the utilization of interface port, which is led to higher waiting time. As a result, we should limit the utilization of network interface port in order to speed up inter-node communications. This issue forces us to determine a threshold on the number of processes, which send medium or large inter-node messages. Based on these observations, our fifth rule is explained as follows:

Rule #5 *A threshold must be fixed for the number of processes, which send medium or large inter-node messages in a computing node. This can expedite inter-node communications.*

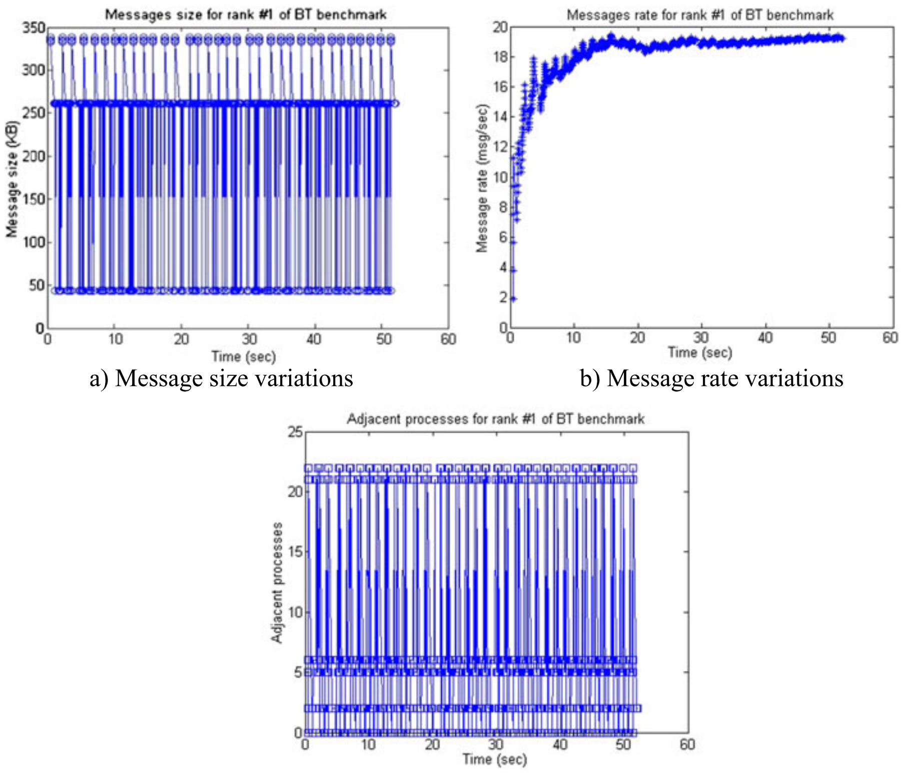
Although distributing processes with high communication demands can improve inter-node communications in a parallel job, when addressing workloads with multiple parallel jobs, this may cause inter-job contention. With the presence of severe inter-job contention, communications of different jobs can affect each other. This is in contrast to Rule #3, which indicates that the communication path of different jobs should be isolated. In such situations, distributing parallel processes has a detrimental effect on the communication performance. So, yet another rule should be considered:

Rule #6 *A trade-off should be considered based on the severity of inter-job contention. This trade-off determines whether a threshold for distributing processes should be considered or not.*

4 Proposed mapping algorithm

4.1 Proposed strategy

In this section, we propose our mapping solution to assign parallel processes into processing cores of a multi-core cluster. The solution is based on the rules obtained in the previous section. It must be noted that while the analytical results are just valid in steady state conditions, they can be used to predict the behavior of many parallel applications. This issue stems from the fact that in most parallel applications, computation and communication behavior of parallel processes are kept unchanged during execution. Evidence to this claim is our analysis on NPB[8] benchmarks. NPB suite is a package containing several benchmarks designed to evaluate the performance of supercomputer systems [22]. Figure 7 shows the communication behavior of BT benchmark. In this figure, communication characteristics are shown for process 1 (process with rank 1) and for the first 1000 communicated messages. In the figure, Sect. ‘a’



a) Message size variations

b) Message rate variations

c) Adjacent processes

Fig. 7 Communication behavior of BT benchmark

[8]Nas Parallel Benchmarks.

shows the variations in the size of communicated messages, Sect. ‘b’ shows the variations in the rate of communicated messages and finally, Sect. ‘c’ shows adjacent processes of current process (process 1) during execution. As can it be seen, communication parameters like message size or adjacent processes are kept unchanged during execution (the rate parameter goes steady after a warm-up phase). Although these results are just for one process, our observations confirm that similar behavior is seen in other processes of the benchmark and also in other benchmarks. This fact indicates the steady state manner of parallel applications and the capability of analytical results to predict the behavior of these applications. In the following, our proposed mapping algorithm is explained.

Based on the first analytical rule that emphasizes the role of message length in communication time, we should separate parallel jobs based on the size of messages they send. Here, we considered three different message sizes to categorize the jobs: large messages (1 MB or higher), medium messages (2 kB to 1 MB) and small messages (2 kB or less). If processes of a job send messages with different sizes, the largest message length is considered for action. According to Rule #3, an efficient mapping strategy should isolate the path of communications for different message sizes. Based on this rule, large messages should be transferred as intra-node messages as much as possible. In doing so, we first start mapping by assigning parallel processes of the first category and try to put adjacent processes near each other and in the same node to profit from higher bandwidth of memory compared to the interface bandwidth. This trend is followed by assigning other jobs in the second and third categories, respectively. In each category, parallel jobs are mapped to the computing nodes in a job-by-job manner. For this purpose, parallel processes of a given job are sorted based on their communication demand, and processes with higher demands are mapped earlier.

The size and the rate of messages sent by each process are the main parameters that define the communication demand of that process. According to Rule #2, however, high values of CV can incur considerable delay in message sending time when sending medium or large messages. More delay in message sending time can be translated to more communication demand. To determine the impact of CV on the communication demand, we've used an approximate method. In this method, the percentage of slowdown in the communication time for different CV values (from $C_a = 0$ to $C_a = 3$) is computed. We use then a fitness function, which fits our slowdown percentages accurately. This fitness function is used to determine the percentage of slowdown for any given CV value. Calculated fitness functions for the first and the second categories are $f(x) = 0.85x^2 + 0.01x$ and $f(x) = 0.12x^2$, respectively. To calculate the percentage of slowdown, we used 1 MB and 64 kB messages at the rate 10 req/s for the first and second category, respectively. By denoting the CV of communication from process ‘i’ to process ‘j’ by CV_{ij} and using the fitness functions we mentioned above, the demand of communication from process ‘i’ to process ‘j’ (CD_{ij}) is computed using the equation below. In this equation, L_{ij} and λ_{ij} are the size and the rate of messages sent from process ‘i’ to process ‘j’, respectively.

$$\mathrm{CD}_{ij} = L_{ij}\lambda_{ij} + \frac{f(\mathrm{CV}_{ij})}{100} L_{ij}\lambda_{ij} \tag{13}$$

After assigning a process with highest communication demand (given process A) to an idle core, adjacent processes of A are assigned in the same node as much as possible so that they profit from high bandwidth of memory (Rule #3). But there is an important issue: If the number of adjacent processes is high, or if the number of available idle cores in current node is low, some of adjacent processes must be assigned to other computing nodes inevitably. Under such conditions, presence of high inter-node communications can lead to high utilization of interface, which can degrade the performance. In this situation, based on Rule #5, a threshold must be determined for the number of processes, which have high inter-node communications. But as mentioned in Rule #6, severe inter-job contention prevents us from fixing a threshold value. The trade-off condition and the way in which we determine the threshold value will be explained in the next sub-section. An important note in the new algorithm is that when assigning multiple processes of the first category to the same node, it should be tried to put these processes in different sockets aiming at alleviating contention on the local memories and improving inter-node communications (Rule #4).

4.2 Fixing a threshold value

As mentioned earlier, in situations in which we cannot accommodate a process together with all of its adjacent processes in just one computing node, high inter-node communications may cause severe contention on the interface, and a threshold should be determined on the number processes accessing the interface. On the other hand, distributing processes among computing nodes can deteriorate the performance when there is high inter-job contention. In this paper, we used a simple but yet efficient method to recognize high inter-job contention. In this method, average adjacency among all jobs (jobs in the third category are ignored because they have negligible impact on the communication performance) in the workload, indicated by $\mathrm{WL\_ADJ}_{\mathrm{avg}}$, is computed using the average adjacency among processes of each job (indicated by $\mathrm{ADJ}_{\mathrm{avg}}$). $\mathrm{WL\_ADJ}_{\mathrm{avg}}$ is then compared with the number of cores in each node (in this paper we assumed 16 processing cores in each node). If $\mathrm{WL\_ADJ}_{\mathrm{avg}}$ is more than the number of cores, it is deduced that each process in the workload has inter-node communications, which implies high amount of inter-job contention. In this situation, all processes are mapped near each other as much as possible, and no threshold is determined. Although may not severe, there could be considerable contention on the interface when $\mathrm{WL\_ADJ}_{\mathrm{avg}}$ is lower than the number of cores. In such conditions, we should again determine whether fixing a threshold value is required or not. In doing so, it is verified that whether a process together with all of its adjacent processes can be lodged in just one computing node or not. We do this by comparing $\mathrm{ADJ}_{\mathrm{avg}}$ with the average idle processing cores in the clustered system (indicated by $\mathrm{IdleCores}_{\mathrm{avg}}$). The volume of inter-node communications would not be considerable If $\mathrm{ADJ}_{\mathrm{avg}}$ is lower than $\mathrm{IdleCores}_{\mathrm{avg}}$. Otherwise, a significant portion of communications would be as inter-node communications. In this case, we should determine a threshold for each job, individually. To determine the threshold value, we can minimize average waiting time of messages at both interface queue and memory queue in a given node. In other words, the goal is to find specific number of nodes that minimizes the value of W which is computed as follows:

$$W = W_{\text{interface}} + W_{\text{memory}} \tag{14}$$

In the equation above, $W_{\text{interface}}$ and W_{memory} are average waiting time of messages at the interface queue and memory queue, respectively. Computing average waiting time of messages at a given server requires information about the rate of requests flowing to that server. Arrival rate of messages to the interface queue is approximately computed as shown below:

$$\lambda_{\text{interface}} = \left\lfloor \frac{P}{n} \right\rfloor M\lambda_{\text{avg}} \tag{15}$$

In the equation above, 'P' is the number of parallel processes, 'n' is the number of selected computing nodes, 'M' is the average number of adjacent processes out of current node, and λ_{avg} is the average message rate between adjacent processes.

Equation (15) says that for 'P' processes distributed among 'n' computing nodes, there are $\lfloor \frac{P}{n} \rfloor$ processes, which reside in each selected node. Each process within a node sends messages to M adjacent processes (on average) out of its node at the rate λ_{avg}. To calculate the parameter 'M', we assume that when a process resides in a node, all of its adjacent processes also reside in the same node, as much as possible. Using this assumption, 'M' is computed as follows:

$$M = \begin{cases} \text{ADJ}_{\text{avg}} - (\lfloor \frac{P}{n} \rfloor - 1) & \text{if } \text{ADJ}_{\text{avg}} > \lfloor \frac{P}{n} \rfloor - 1 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Arrival rate of messages to the memory queue is approximately computed as follows:

$$\lambda_{\text{memory}} = \lambda_{\text{mem}}^{\text{intra}} + \lambda_{\text{mem}}^{\text{out}} + \lambda_{\text{mem}}^{\text{in}} \tag{17}$$

In Eq. (17), $\lambda_{\text{mem}}^{\text{int ra}}$ is the rate of accesses to the memory for intra-node communications, $\lambda_{\text{mem}}^{\text{out}}$ is the rate of accesses to the memory by the messages sent outside, and $\lambda_{\text{mem}}^{\text{in}}$ is the rate of accesses to the memory by the messages received at current node as a destination node. Each of these parameters is calculated as follows:

$$\lambda_{\text{mem}}^{\text{intra}} = \left\lfloor \frac{P}{n} \right\rfloor N\lambda_{\text{avg}}\tau \tag{18}$$

$$\lambda_{\text{mem}}^{\text{out}} = \left\lfloor \frac{P}{n} \right\rfloor M\lambda_{\text{avg}} \tag{19}$$

$$\lambda_{\text{mem}}^{\text{in}} = \left(P - \left\lfloor \frac{P}{n} \right\rfloor\right) \frac{\text{ADJ}_{\text{avg}}}{P-1} \left\lfloor \frac{P}{n} \right\rfloor \lambda_{\text{avg}} \tag{20}$$

In Eq. (18), 'N' indicates average number of adjacent processes in current node. Using the assumption considered in Eq. (16), 'N' can be calculated as below:

$$N = \begin{cases} \lfloor \frac{P}{n} \rfloor - 1 & \text{if } \text{ADJ}_{\text{avg}} > \lfloor \frac{P}{n} \rfloor - 1 \\ \text{ADJ}_{\text{avg}} & \text{otherwise} \end{cases} \tag{21}$$

Parameter 'τ' in Eq. (18) represents the number of memory accesses per each intra-node message transfer. Since in most MPI implementations, two memory copes are done for transferring intra-node messages, we assume $\tau = 2$. Equation (19) is obtained just like Eq. (15), because for each message which must be sent as an inter-node message, one access to the memory is required. In Eq. (20), $(P - \lfloor \frac{P}{n} \rfloor)$ indicates the number of processes placed outside of current node and send messages at the rate λ_{avg} to each of the $\lfloor \frac{P}{n} \rfloor$ processes inside the current node with the probability $\frac{\text{ADJ}_{\text{avg}}}{P-1}$. By denoting the optimal number of nodes, which minimizes W, with n_{opt}, the threshold value is then computed as follows:

$$\text{Threshold} = \left\lceil \frac{P}{n_{\text{opt}}} \right\rceil \tag{22}$$

In some communication-intensive workloads in which processes use extremely high rates or very large messages, utilization of servers may reach to 100 %. In this case, it is not possible to compute the average waiting time. In such situations we determine the threshold using the sum of arrival rates to the servers. In other words, instead of minimizing W by Eq. (14), we try to minimize λ, which is computed as below:

$$\lambda = \lambda_{\text{interface}} + \lambda_{\text{memory}} \tag{23}$$

In the proposed mapping algorithm, parallel jobs in each category are sorted based on the average adjacency among their processes, and jobs which have higher adjacency are mapped earlier. Because of high adjacency in these jobs, they may require to be distributed among the computing nodes to obtain efficient performance. As a result, these jobs should be mapped before other jobs to use available idle cores of computing nodes. Figure 8 shows a brief pseudocode of the proposed mapping algorithm.

5 Experimental results

5.1 Experimental results for synthetic workloads

Our first set of evaluations was performed on synthetic workloads. For this purpose, we used the Omnet++ v4.1 simulator [23]. The system which we have considered for simulation has the same characteristics as for one used in Sect. 3 for conducting simulations. Table 2 lists the parameters we used in our simulations. In synthetic workloads, messages with different sizes and rates were generated. These traffics use four different communication patterns between parallel processes including: Bcast/Scatter, Gather/Reduce, All-to-All, and Linear. In Bcast/Scatter, one process (as a root process) broadcasts its messages to other processes and the other processes are just receiver. In Gather/Reduce, one process receives messages from other processes and the other processes are just sender. 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. We have defined 10 synthetic workloads, some with fixed patterns (which are named based on the pattern used)

```
New_Mapping_Algorithm( )
Input: Workload graph, Cluster architecture
Output: Mapping information
{
1. $WL\_ADJ_{avg}$ =find_total_average_adjacency( workload graph );
2. if($WL\_ADJ_{avg}$ <=num_of_cores )    // inter-job contention is not severe
   {
   2.1. job_pool = select_jobs ( high_ length_messages );
   2.2. sort_jobs ( job_pool );    // sort based on $ADJ_{avg}$
   2.3. while ( job_pool is not empty )
      {
      2.3.1. current_job = select_job( job_ pool );
      2.3.2. If ($ADJ_{avg} <= IdleCores_{avg} - 1$)
               No threshold is determined;
             else
             {
                  $n_{opt}$ =find_optimum_nodes();
                  $Threshold = \left\lceil \frac{P}{n_{opt}} \right\rceil$;
             }
      2.3.3. sort_processes(current_job);    //sort based on comm. demand
      2.3.4. current_process = select_process( current_job );
      2.3.5. map_process( current_process );
      2.3.6. sort_adj_processes( current_process );    // sort based on comm. demand
      2.3.7. map_adj_processes( threshold, cluster architecture );
      } // end while
   2.4. job_pool = select_jobs ( medium_ length_messages );
   2.5. repeat steps 2.2 and 2.3;
   2.6. job_pool = select_jobs ( small_ length_messages );
   2.7. repeat steps 2.2 and 2.3;    // ignore choosing a threshold
   } end if
   else // inter-job contention is severe
      do steps 2.1 to 2.7;    //ignore choosing a threshold for each job
}    // end of New_Mapping_Algorithm
```

Fig. 8 The pseudocode of the proposed mapping algorithm

Table 2 Simulation parameters

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

and some with mixed patterns (which are named “Mixed_1” to “Mixed_ 6”). Table 3 displays the definition of fixed-pattern synthetic workloads, and Tables 4 and 5 show the same matter for mixed-pattern synthetic workloads.

Since our main goal in this paper is to improve performance by reducing waiting time of messages at server queues, we have considered some of the waiting time of

Table 3 Message characteristics for fixed-pattern synthetic workloads

Job	No. of processes	Rate	Length	Message count	
				All-to-All	Bcast/Scatter–Gather/Reduce–Linear
0	32	500 m/s[*]	2 MB	1500	2000
1	32	500 m/s	2 MB	1500	2000
2	32	100 m/s	2 MB	1500	2000
3	32	10 m/s	2 MB	1500	2000
4	32	1000 m/s	64 kB	1500	2000
5	32	100 m/s	64 kB	1500	2000
6	32	10 m/s	64 kB	1500	2000
7	32	1000 m/s	1 kB	1500	2000

[*] m/s means messages per second

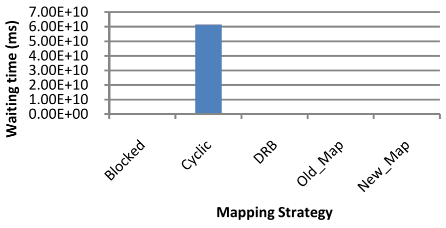
Table 4 Message characteristics for Mixed_1 to Mixed_4 workloads

Job	No. of processes	Pattern	Mixed_1		Mixed_2		Mixed_3		Mixed_4		Count
			Rate	Length	Rate	Length	Rate	Length	Rate	Length	
0	64	All-to-All	100 m/s	64 kB	1000 m/s	64 kB	10 m/s	2 MB	100 m/s	2 MB	2000
1	64	Bcast/Scatter	100 m/s	64 kB	1000 m/s	64 kB	10 m/s	2 MB	100 m/s	2 MB	2000
2	64	Gather/Reduce	100 m/s	64 kB	1000 m/s	64 kB	10 m/s	2 MB	100 m/s	2 MB	2000
3	64	Linear	100 m/s	64 kB	1000 m/s	64 kB	10 m/s	2 MB	100 m/s	2 MB	2000

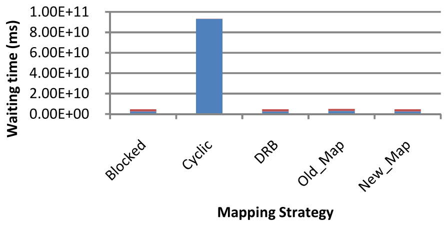
Table 5 Message characteristics for Mixed_5 and Mixed_6 workloads

Job	No. of processes		Pattern	Rate	Length	Message count
	Mixed_5	Mixed_6				
0	32	24	All-to-All	10 m/s	2 MB	2000
1	32	24	Bcast/Scatter	10 m/s	2 MB	2000
2	32	24	Gather/Reduce	10 m/s	2 MB	2000
3	32	24	Linear	10 m/s	2 MB	2000
4	32	24	All-to-All	10 m/s	64 kB	2000
5	32	24	Bcast/Scatter	10 m/s	64 kB	2000
6	32	24	Gather/Reduce	10 m/s	64 kB	2000
7	32	24	Linear	10 m/s	64 kB	2000

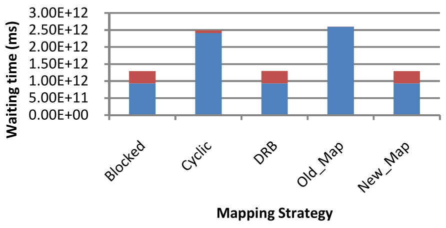
messages at server queues (network interface and main memory) as our main metric to compare the results. We compared our performance results with the results obtained from the Blocked, Cyclic, DRB, and also with the results obtained from our previous proposed mapping method. Figure 9 illustrates the performance results for different workloads. In this figure, ‘Old_Map’ and ‘New_Map’ indicate our previous mapping strategy and the new algorithm, respectively. To extract DRB results, we



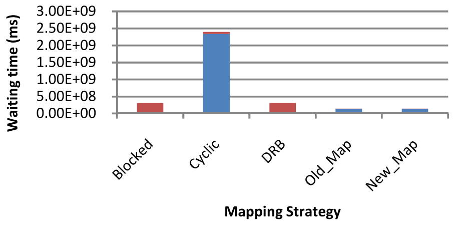
a) Bcast/Scatter



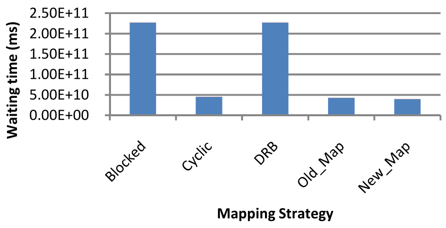
b) Gather/Reduce



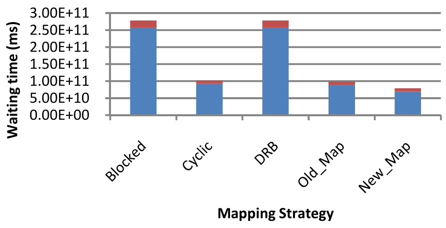
c) All-to-All



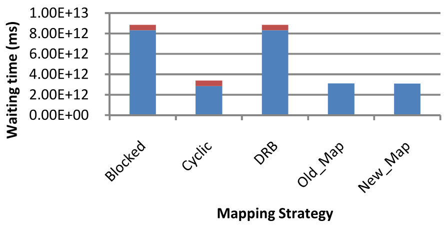
d) Linear



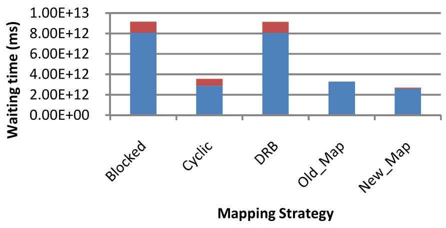
e) Mixed_1



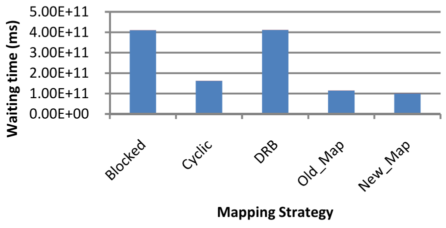
f) Mixed_2



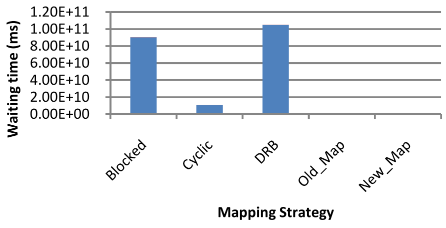
g) Mixed_3



h) Mixed_4



i) Mixed_5



j) Mixed_6

Fig. 9 Waiting time of messages for synthetic workloads

used Scotch v5.1 software [24]. In the figure, the red section of each bar indicates total waiting time at main memories' queues, and the blue section represents total waiting time at network interfaces' queues. Total waiting time of messages shown is in mili-second. As it can be seen from Fig. 9, our new method has performed as well as the Blocked, DRB, and Old_Map in the Bcast/Scatter and Gather/Reduce scenarios. In the Bcast/Scatter scenario, since each parallel job (containing 32 processes) has only one sender process and the rest of processes are just receivers, the best strategy for mapping is to put all receiver processes near the sender. As a result, the Blocked and DRB methods provide efficient results. In this scenario, the Cyclic method has led to inefficient performance. Using this method, dominant part of communications are inter-node communications, which use lower bandwidth of network interface compared to the memory bandwidth, and consequently, the Cyclic has degraded the performance. In the Bcast/Scatter scenario and by using the new mapping strategy, since average adjacency is equal to 1, no threshold is determined for the jobs. In this case, the new method (as well as the Old_Map method) attempts to put parallel processes near each other as much as possible. So, the results are similar to the one obtained in the Blocked and DRB methods.

In the Gather/Reduce scenario, performance results have similar justifications as the previous scenario. In All-to-All scenario, each process sends its messages to all other processes. Since all parallel processes of a job cannot be lodged in just one computing node, the volume of inter-node communications is high. Because of high amount of inter-job contention in this workload, the Cyclic and Old_Map methods (which simply distribute processes among the nodes) have led to unacceptable results. In this scenario, the New_Map method has detected high inter-job contention, which has resulted efficient results as well as the Blocked and DRB. For the Linear workload and by using the Cyclic approach, all communications are of inter-node type. Knowing the lower throughput of network interface compared to the memory, waiting time of messages for this method is the highest. In the Blocked and DRB techniques, dominant part of communications is done as intra-node communications, which has led to high waiting time at memories' queues. The New_Map strategy (as well as the Old_Map method) has performed very well and 53 % improvement in performance is observed, compared to the best result of other methods. The reason is that communications of processes are appropriately divided between intra-node and inter-node communications. Moreover, processes that send large messages are placed in different sockets. This reduces the contention on memories (and also on the network interfaces). All of these issues make the New_Map algorithm a good strategy for this kind of pattern.

By looking to the characteristics of parallel jobs defined in Mixed_1 to Mixed_6 workloads, we find that there are one or more parallel jobs with All-to-All pattern. Because of high number of parallel processes in these scenarios, even by using Blocked or DRB method, a significant part of communications in the All-to-All jobs is done as inter-node communications, and consequently, there is severe contention on the interface. Despite this, other parallel jobs have low average adjacency and their volume of communication is not high. As a result, inter-job contention is not significant and the New_Map method has decided to determine a threshold for just All-toAll jobs. The New_Map technique blends the benefits of both Cyclic and Blocked approaches in these workloads. In other words, when facing to an All-to-All job, it acts

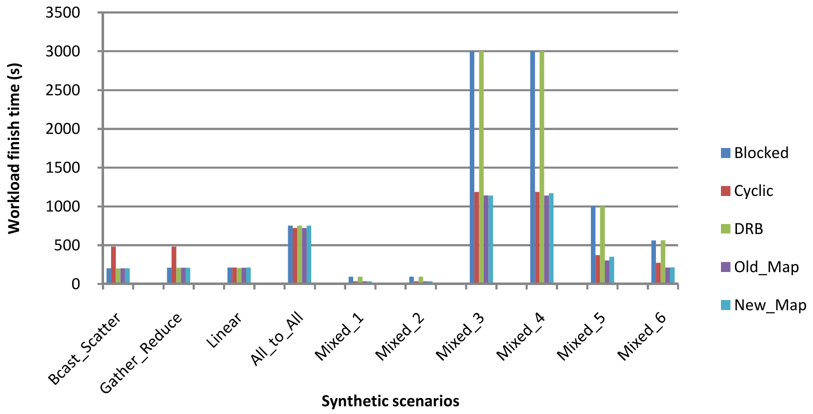


Fig. 10 Workload finish time for different mapping strategies

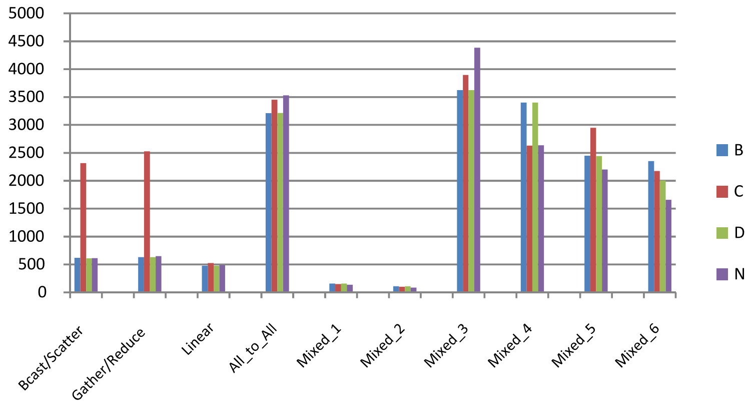


Fig. 11 Total finish time of parallel jobs for different mapping strategies

like the Cyclic and distributes parallel processes between the nodes, and when facing to other jobs (which have light communications) it does not determine any threshold and behaves similar to the Blocked. The New_ Map strategy has gained 12.8 %, 21.9 %, 8.8 %, 23.7 %, 37.6 %, and 91.1 % performance improvement for Mixed_1 to Mixed_6 scenarios, respectively, compared to the Cyclic approach. The new method has also gained 7.7 %, 19.6 %, 0.3 %, 17.4 %, 12.2 %, and −1.5 % performance improvement compared to the Old_Map technique. As we can see from the results, selecting a better threshold has resulted in better performance in most scenarios compared to the Old_Map technique. In addition to the waiting time of messages at server queues, we have also used other metrics for our comparisons. Two other main metrics are workload finish time (the time at which execution of all parallel jobs in the workload is finished) and total finish time of parallel jobs in each workload. Performance results using these two metrics are shown in Figs. 10 and 11. Once again, the New_Map method has generated efficient results in nearly all scenarios, which indicates the robustness of the new strategy.

Table 6 Real workload definitions

Job	Benchmarks				
	HW_1	HW_2	MW	LW_1	LW_2
0	SP (25-C)*	IS (8-B)	BT (25-B)	SP (25-B)	SP (25-C)
1	IS (32-C)	FT (32-B)	CG (32-B)	CG (32-B)	CG (32-C)
2	FT (32-B)	IS (32-C)	EP (32-B)	EP (32-B)	EP (32-C)
3	FT (16-B)	MG (32-C)	FT (32-B)	MG (32-B)	MG (32-C)
4	IS (16-C)	CG (32-C)	IS (32-B)	–	–
5	CG (32-C)	IS (32-B)	LU (25-B)	–	–
6	IS (8-B)	MG (32-B)	MG (32-B)	–	–
7	BT (25-C)	CG (32-B)	SP (25-B)	–	–
8	CG (16-B)	BT (16-C)	–	–	–

*SP (25-C) means SP benchmark with 25 processes in class C

5.2 Experimental results for real workloads

In addition to evaluating performance results for synthetic workloads, we have also used real workloads in our comparisons. Real workloads were extracted from communication behavior of NPB benchmarks. NPB benchmarks have different communication characteristics in terms of message size, message rate and communication pattern. To simulate the behavior of NPB benchmarks, their traces were generated and then fed to the simulator. Here, we defined 5 real workloads. Employed benchmarks in each workload are listed in Table 6. Various kinds of benchmarks are used in each real workload. For example, HW_1 and HW_2 are heavy (communication-intensive) workloads, MW is a medium workload, and LW_1 and LW_2 are light workloads in term of communication demand. In heavy workloads, IS and FT benchmarks are used more than other benchmarks. These benchmarks have high communications and their communication pattern is entirely of All-to-All type. MW, LW_1 and LW_2 scenarios have been defined to show that our proposed algorithm have efficient performance not only in communication-intensive workloads, but also in medium and light communications. The performance results for 5 real workloads are illustrated in Fig. 12. As expected, the Cyclic has performed better than the Blocked and DRB methods in heavy workloads. In these workloads, the New_Map strategy has gained the best results by achieving 24.7 % and 12.1 % performance improvement compared to the Cyclic. It has also obtained 14.8 % and 16.9 % performance improvement compared to the Old_Map method, which is caused by selecting a better threshold value. The performance results for MW scenario show that there are no significant differences between various mapping methods. Despite this, the new method is more efficient than the others. In spite of light communications in LW_1, our strategy (together with the Old_Map technique) has achieved acceptable performance, and its result is even better in LW_2, which is somewhat more communication-intensive than the LW_1. Unfortunately, the way in which we gathered the benchmark traces, limited us to extract performance results using the "workload finish time" and "total finish time of jobs" metrics for real workloads.

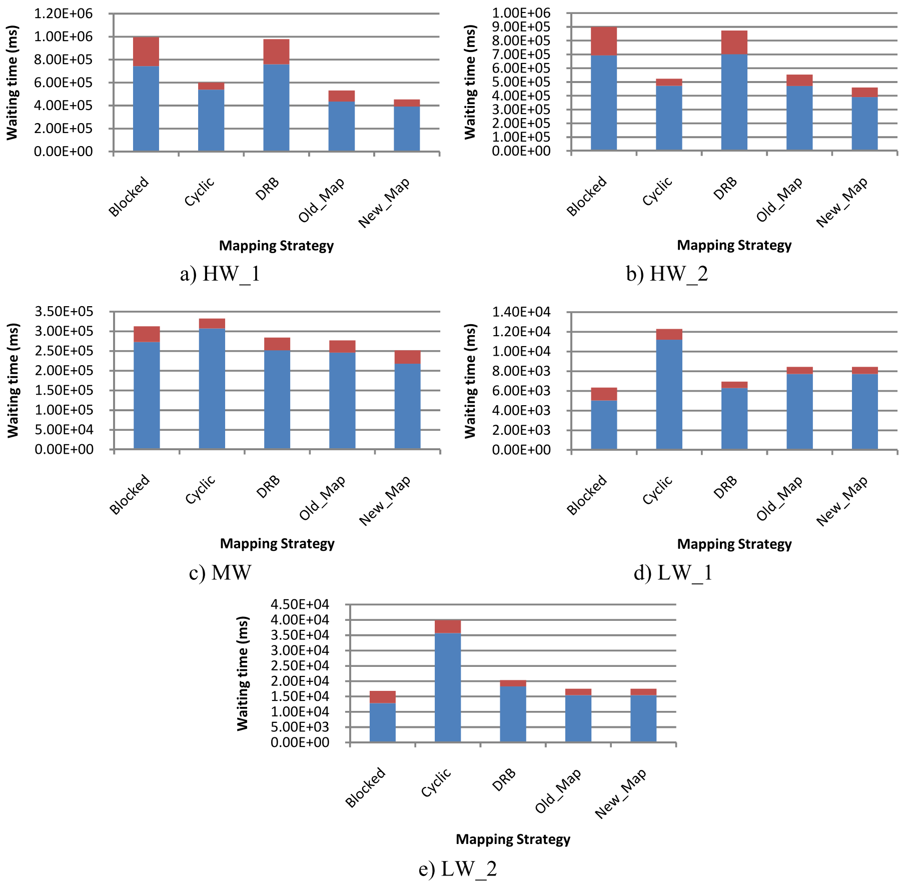


Fig. 12 Waiting time of messages for real workloads

6 Conclusion

In this paper, we proposed a new mapping technique to assign parallel processes into processing cores of a multi-core cluster aimed at reducing network interface contention and expediting inter-node communications. The proposed technique, which is based on queuing network modeling of a limited-size cluster, was compared to some other well-known methods for synthetic and real workloads. Performance results revealed that our new mapping method can gain significant performance in many workloads, especially in communication-intensive workloads which we observed up to 91 % improvement compared to the Cyclic method. We also achieved improvement in performance by up to 19.6 % compared to our previous strategy. Simplicity of implementation of the new technique and its efficient performance for various communication patterns make our algorithm an applicable mapping method to use in recent high performance clusters.

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 (2010) Performance impact of resource contention in multicore systems. In: IEEE international symposium on parallel and distributed processing, Atlanta, USA
3. Chai L, Gao Q, Panda DK (2007) 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
4. Jokanovic A, Rodriguez G, Sancho JC, Labarta J (2010) Impact of inter-application contention in current and future HPC systems. In: IEEE annual symposium on high-performance interconnects, Mountain View, USA
5. Kayi A, El-Ghazawi T, Newby GB (2009) Performance issues in emerging homogeneous multi-core architectures. Simul Model Pract Theory 17(9):1485
6. Narayanaswamy G, Balaji P, Feng W (2008) Impact of network sharing in multi-core architectures. In: 17th IEEE international conference on computer communications and networks, Virgin Islands, USA
7. Chen H, Chen W, Huang J, Robert B, Kuhn H (2006) MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In: Proceedings of the 20th annual international conference on supercomputing, New York, USA
8. Chai L, Hartono A, Panda DK (2006) Designing high performance and scalable MPI intra-node communication support for clusters. In: IEEE international conference on cluster computing, Barcelona, Spain
9. Rex R, Mietke F, Rehm W, Raisch C, Nguyen H (2006) Improving communication performance on InfiniBand by using efficient data placement strategies. In: IEEE international conference on cluster computing, Barcelona, Spain
10. Dummler J, Rauber T, Runger G (2008) Mapping algorithms for multiprocessor tasks on multi-core clusters. In: 37th IEEE international conference on parallel processing, Portland, USA
11. Ichikawa S, Takagi S (2009) 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
12. Mercier G, Clet-Ortega J (2009) 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
13. Rodrigues ER, Madruga FL, Navaux POA, Panetta J (2009) Multi-core aware process mapping and its impact on communication overhead of parallel applications. In: IEEE symposium on computers and communications, Sousse, Tunisia
14. Khoroshevsky VG, Kurnosov MG (2009) Mapping parallel programs into hierarchical distributed computer systems. In: 4th international conference on software and data technologies, Sofia, Bulgaria
15. Diaz J, Petit J, Serna M (2002) A survey of graph layout problems. ACM Comput Surv 34(3):313
16. Agrawal T, Sharma A, Kale LV (2006) Topology-aware task mapping for reducing communication contention on large parallel machines. In: 20th IEEE international symposium on parallel and distributed processing, Rhodes Island, USA
17. Koop MJ, Luo M, Panda DK (2009) Reducing network contention with mixed workloads on modern multicore clusters. In: IEEE international conference on cluster computing and workshops, New Orleans, USA
18. Koukis E, Koziris N (2006) Memory and network bandwidth aware scheduling of multiprogrammed workloads on clusters of SMPs. In: 12th international conference on parallel and distributed systems, Minneapolis, USA
19. Soryani M, Analoui M, Zarrinchian G (2012) A novel process mapping strategy in clustered environments. Int J Grid Comput Appl 3(2):30
20. InfiniBand Trade Association (2007) InfiniBand architecture specification, vol 1, release 1.2.1
21. Menasce D, Almeida VLF, Dowdy LW (2004) Performance by design: computer capacity planning by example Prentice Hall, New York. ISBN: 0-13-090673-5
22. http://www.nas.nasa.gov/publications/npb.html
23. http://www.omnetpp.org
24. http://www.labri.fr/perso/pelegrin/scotch/