Processor Affinity and MPI Performance on SMP-CMP Clusters

Chi Zhang Xin Yuan Ashok Srinivasan
Department of Computer Science, Florida State University, Tallahassee, FL 32306
{czhang,xyuan,asriniva}@cs.fsu.edu

***Abstract*—Clusters of Symmetric MultiProcessing (SMP) nodes with multi-core Chip-Multiprocessors (CMP), also known as SMP-CMP clusters, are becoming ubiquitous today. For Message Passing interface (MPI) programs, such clusters have a multi-layer hierarchical communication structure: the performance of intra-node communication is usually higher than that of inter-node communication; and the performance of intra-node communication is not uniform with communications between cores within a chip offering higher performance than communications between cores in different chips. As a result, the mapping from Message Passing Interface (MPI) processes to cores within each compute node, that is, *processor affinity*, may significantly affect the performance of intra-node communication, which in turn may impact the overall performance of MPI applications. In this work, we study the impacts of processor affinity on MPI performance in SMP-CMP clusters through extensive benchmarking and identify the conditions when processor affinity is (or is not) a major factor that affects performance.**

Keywords-**Processor affinity; MPI; SMP-CMP clusters**

I. Introduction

Clusters of Symmetric MultiProcessing (SMP) nodes with multi-core Chip-Multiprocessors (CMP) are becoming ubiquitous today. We refer to such clusters as SMP-CMP clusters. The majority of the supercomputers in the Top 500 supercomputers list published in June 2009 [1] are SMP-CMP clusters. Due to their high performance to price ratios, SMP-CMP clusters are likely to be the dominating high performance computing platforms for the foreseeable future. Hence, it is important to fully understand the performance issues in such clusters.

We consider Message Passing Interface (MPI) applications on SMP-CMP clusters, with focus on the pure MPI model where each MPI process runs on one core [20]. Due to nodal architecture features and the recent optimizations of MPI intra-node communication using shared memory [7], [9], [11], SMP-CMP clusters have a multi-layer hierarchical communication infrastructure for MPI applications. Intra-node communication usually has higher performance than inter-node communication. In addition, the performance of intra-node communication is non-uniform: communications between cores within a chip can usually achieve lower latency and higher bandwidth than communications between cores in different chips. Contemporary SMP-CMP clusters allow the mapping between processes to cores to be specified through *processor affinity* rules that describe how each MPI process can be associated with the cores in a node [21]. Clearly, with the multi-layer communication hierarchy, processor affinity can influence intra-node communications in MPI applications and affect the overall performance [6], [8].

While it is well known that processor affinity can affect MPI performance [6], [8], [18], [23], the impacts of processor affinity are still not well understood. In particular, the conditions under which processor affinity is or is not critical to the performance are unclear. Due to the lack of clear understanding, there is no well-defined scheme to set processor affinity; and the choice of processor affinity in practice is often ad hoc. This is the issue we try to address in this paper. A good understanding of the impacts of processor affinity is also important in several related areas including the design and evaluation of MPI collective algorithms, the development of performance models, and the understanding of MPI application performance in general.

In this work, we systematically study the impacts of processor affinity on MPI performance, focusing on identifying the situations when processor affinity is important and when processor affinity is only a minor factor. The goal is to develop a guideline to determine whether to consider processor affinity or not when running MPI applications, evaluating MPI communication performance, and modeling MPI performance. We perform extensive experimentation with micro-benchmarks and NAS parallel benchmarks on two representative platforms, the Jaguar cluster at Oak Ridge National Lab (ORNL) with AMD Opteron six-core processors and an Intel cluster with Intel Xeon Quad-core processors. The major conclusions include the following:

- The impact of processor affinity depends heavily on the inter-node to intra-node communication ratio. This applies to both the Jaguar cluster and the Intel cluster in our experiment although the Jaguar cluster has newer generation processors and is less sensitive to processor affinity. The inter-node and intra-node communication ratio is a good indicator for predicting whether processor affinity is critical to the performance for both collective communications and MPI applications.
- Processor affinity has strong impacts on sparse communication patterns. Since MPI collective communication operations are often realized with sparse communication patterns (for example, the all-gather operation can be realized by a logical ring pattern [22]), processor affinity must be taken into consideration when evaluating and developing collective communication routines for SMP-

CMP clusters.

- The impacts of processor affinity in general decrease as the system size increases, which is a good news for people building large scale SMP-CMP clusters.
- For many MPI applications, the inter-node communication dominates the total communication cost, and the impact of processor affinity is negligible. This indicates that in many cases, treating an SMP-CMP cluster as a traditional SMP cluster is a close approximation.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the background of this work. Section 4 details the experimental methodology and Section 5 reports experimental results. Finally, Section 6 concludes the paper.

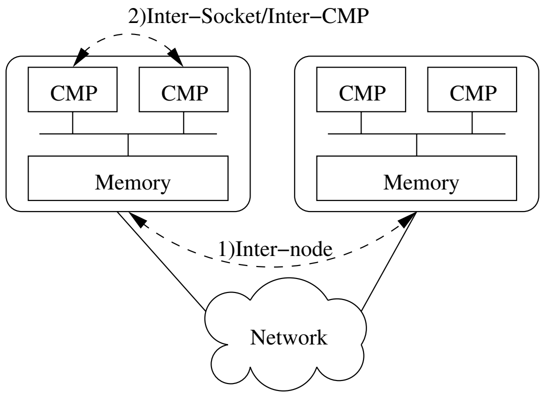
II. Related work

Closely related to this work is the research on characterizing MPI communication on multi-core clusters and MPI communication optimization for SMP-CMP clusters. Chai studied the application communication characteristics on SMP-CMP clusters, pointed out potential bottlenecks and ways to avoid them, and investigated the scalability in multi-core clusters [8]. Alam [6] studied the performance of scientific MPI applications on SMP-CMP clusters and discovered that processor and memory affinity can have significant impacts on the performance. The results are confirmed in [18]. These studies, however, did not try to determine the conditions when processor affinity is (or is not) critical to the performance, which is the focus of our work. Much work has been done to improve MPI performance on SMP-CMP clusters. Techniques were developed to improve both point-to-point communications [7], [9], [11], [13] and collective communications [15], [16], [17], [19]. These optimizations further differentiate the communication performance in the multi-layer communication infrastructure in SMP-CMP clusters and manifest the impacts of processor affinity.

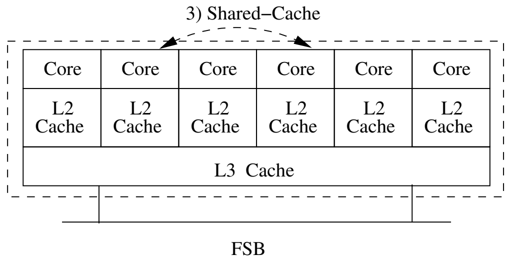
III. MPI communication on SMP-CMP clusters

We will use the architectures of the clusters in our experiments to illustrate the communication issues in common SMP-CMP clusters. We use two clusters in the experiments, one with Intel Xeon processors and the other one with AMD six-core Opteron processors, which will be referred to as the Intel cluster and the AMD cluster, respectively. The compute nodes in both clusters are 2-way SMPs. The overview of the architectures of the two clusters is shown in Figure 1 (a), which resembles a typical SMP-CMP cluster. The Intel cluster uses Xeon quad-core processors (Clovertown), which is depicted in Figure 1 (c) while the AMD cluster uses six-core Opteron processors, depicted in Figure 1 (b). Intel Xeon Clovertown quad-core processors put two dual-core processors with shared L2 cache on the same chip. Every core in an AMD Opteron quad-core processor, on the contrary, owns a private L2 cache and shares a common L3 cache.

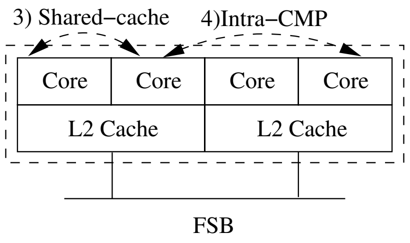
MPI communications on an SMP-CMP cluster can be either inter-node or intra-node. Inter-node communications are between cores in different nodes and the messages must go



(a) Cluster overview



(b) AMD six-core processor



(c) Intel quad-core processor

Fig. 1: A SMP-CMP Cluster

through network interface cards. Intra-node communications are between cores within one SMP node. Contemporary MPI libraries such as MVAPICH[2] and Open MPI [3] realize intra-node communications with shared memory operations. Hence, intra-node communication has in general higher performance than inter-node communication. Moreover, the cache organization in the processors can significantly affect the intra-node communication performance [8], [10]. Depending on the location of the involving cores within a node, MPI intra-node communication can be further classified into three categories: shared-cache, intra-socket, and inter-socket communication [10]. Inter-socket (or inter-CMP) communications, (2) in Figure 1 (a), are between two cores residing on the same node but different sockets. Shared-cache communications, (3) in Figure 1 (c), are between two cores that share cache (e.g. shared L2 cache in Intel Clovertown and shared L3 cache in the AMD processor). Intra-socket communications, (4) in Figure 1 (c), are between two cores within a socket, but not sharing caches. The Intel cluster has all three types of intra-node communications: inter-socket, intra-socket, and shared-cache, while the AMD cluster only has two types of intra-node communications: inter-socket and shared-cache. Among different intra-node communications, shared-cache is much faster than intra-socket and inter-socket since it takes the advantage of cache-to-cache data transfer [10].

Since different types of intra-node communications have different performance, how MPI processes are mapped to cores within a node can make a difference in the performance. Most operating systems by default allow each process to be mapped to any of the cores in a node. Contemporary SMP-CMP clusters provide mechanisms for processor affinity, the capability to specify that a process can run only on a specific core or a set of cores. In Linux, a system call, *set_affinity()*, can restrict the mapping of a process to a subset of all cores. In a large scale cluster such as the ORNL Jaguar cluster, the *numactl* command can be used to specify processor affinity.

IV. Methodology and benchmarks

A. TestBed

Two SMP-CMP clusters are used in our experiments. The first one is the Jaguar cluster at ORNL, which is a Cray XT5-HE with a total of 224162 cores and a proprietary interconnect [4]. Each compute node in Jaguar consists of two AMD Opteron six-core processors at 2.6GHz and 16GB memory. The second one, Draco, is a local cluster, consisting of 16 compute nodes with a total of 128 cores. Each node is a Dell Poweredge 1950 equipped with two 2.33Ghz quad-core Xeon E5345 processors and 8GB memory. Every node runs Linux with 2.6.9-42.ELsmp kernel and mvapich2.0.1.2. All compute nodes are connected by a 20Gbps InfiniBand DDR switch (CISCO SFS 7000D).

B. Benchmarks

To study the impacts of processor affinity, we select and design a set of benchmarks, including a set of communication intensive collective communication algorithms and a benchmark that performs random communications. We also use the NAS parallel benchmarks [5] in the study.

1) Collective communication benchmarks: We use a set of collective communication algorithms with different communication characteristics in the study: the logical ring all-gather algorithm, the binomial tree broadcast algorithm, and the recursive doubling all-gather algorithm. All these algorithms are used in MPICH [22]. Next, we will briefly describe the communication patterns in the algorithms. In the description, we use the following notations. P_k is the process whose rank is k, $0 \leq k \leq p-1$; p is the total number of processes; $P_i \rightarrow P_j$ denotes a communication from P_i to P_j.

The logical ring all-gather algorithm organizes P_0, P_1, ..., P_{p-1} as a logical ring: the pattern is $P_0 \rightarrow P_1 \rightarrow ... \rightarrow P_{p-1} \rightarrow P_0$. The algorithm communicates messages in the logical ring $p-1$ times. When p is a power of 2, the recursive doubling all-gather algorithm has $log(p)$ steps. In each step k, $0 \leq k \leq log(p)-1$, P_i exchange data with $P_{i \bigoplus 2^k}$, where $\bigoplus$ is the exclusive or operation. The message size doubles in every step. Assuming broadcasting from node P_0 to all other nodes and p is a power of 2, the binomial tree algorithm works in $log(p)$ steps. In step k, $0 \leq k \leq log(p)-1$, P_i, $0 \leq i \leq 2^k - 1$, forwards the broadcast message to P_{i+2^k}.

2) Random communication benchmark: The random communication benchmark allows us to experiment with random communication patterns while controlling the inter-node and intra-node communication ratio. This benchmark takes several input parameters including the number of intra-node communications in each node ($intra\_node$), the ratio of inter-node communications and intra-node communications ($ratio$), and the message size ($msize$). It generates a random communication pattern with (1) the number of intra-node communications in each node being $intra\_node$, and (2) the inter-node to intra-node communication ratio being $ratio$. After the random pattern is computed, all communications in the random pattern is posted with MPI_Isend and MPI_Irecv, followed by a MPI_Waitall.

3) MPI applications: We use four NAS benchmarks [5] in the study: CG, LU,MG, and SP. We do not include BT since we were not able to properly execute this program on Jaguar in some cases. Other NAS benchmarks either do not have many communications (EP) or are dominated by all-to-all communications (IS, FT) that processor affinity does not play a role.

C. Experiment method

Communication matrix: In the study, we extract the communication pattern in each benchmark, and represent the pattern using a matrix, m, where $m[i][j]$ denotes either the total amount of data send from P_i to P_j or the number of messages from P_i to P_j. Figure 2 is the pattern matrix for the logical ring pattern on 8-process program (the values in the matrix are the number of communications).

The communication matrices for collective communication benchmarks and random benchmarks are straight-forward to obtain. For the NAS parallel benchmarks, we obtain the matrices as follows. We run the NAS benchmarks and collect communication traces using the PMPI profiling interface. We then analyze the traces and extract the communication pattern matrices from the traces. Since the NAS benchmarks that we use in the experiments are point-to-point benchmarks, we only collect communication patterns for point-to-point communications: collective communications, which account for very small fractions of all communications in these benchmarks, are ignored.

$$\begin{bmatrix} 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 2: Communication matrix for the logical ring all-gather algorithm on 8 processes

Best and worst processor affinity: Once we obtain the communication matrix, best and worst processor affinities for the communication matrix is computed. To do this, we model the nodal architectures of the two clusters with graphs: the nodes in Draco are modeled as the graph in Figure 3; and the nodes in Jaguar are modeled as the graph in Figure 4. These models allow us to capture the three types of intra-node communications: shared-cache communication, intra-socket, and inter-socket communication. In Figures 3 and 4, links marked by 'x' are share-cache links; links marked by 'y' are intra-socket links; and links marks by 'z' are inter-socket links. In Draco, shared-cache communications go through share-cache links; intra-socket communications go through intra-socket links; and inter-socket communications go through the bottom switch and inter-socket links. For nodes in Jaguar, there are only two types of intra-node communications: shared-cache communications and inter-socket communications as shown in Figure 4. It should be clear from the figure that the graphs are trees: from each node (core), there is only one path to another node.

Processor affinity decides which MPI process is mapped to which core. For a given communication pattern, the quality of processor affinity is different: some incur many inter-socket communications while others do not. Intuitively, one would prefer mapping MPI processes such that inter-socket communications are minimized. We define the *best processor affinity* to be a mapping that has the smallest inter-socket traffic among all potential process-to-core mappings. When there are multiple such mappings, the tie is broken by intra-socket traffics. If there are still ties, shared-cache communications are considered. The *worst processor affinity* is defined in a similar but opposite manner.

The intra-node communication pattern is an 8×8 matrix for the 8-core SMP node and 12×12 matrix for the 12-core SMP node. For the 8-core node, the best processor affinity is computed as follows. We enumerate all possible process-to-core mappings, that is, $8! = 40320$ mappings. For each mapping, we compute the loads on each link in the graph model based on the communication matrix. The computation is trivial since each source-destination pair is connected by exactly one path. The mapping that results in the lowest/highest load in the inter-socket link is the best/worst affinity. If multiple mappings result in the same maximum load in inter-socket links, we compare the loads in the intra-socket links, and so on. For the 12-core node, we enumerate all possible mappings while taking advantage of the fact that two processors are exactly the same, so the number of mappings to be considered is $C_{12}^{6} = 924$ cases. For a benchmark that runs on more than one compute node, we compute the best/worst affinity for each individual node: different nodes can have different processor affinity.

V. Experiment results

A. Collective communication benchmarks

Figures 5 to 7 show the results for the collective benchmarks on the two platforms. On Draco with 128 processes (16 nodes), the logical ring algorithm, whose results are shown in Figure 5, has a smallest inter-node communication and intra-node communication ratio in terms of the number of messages in the pattern, $1 : 7 = 0.142$. The binomial tree broadcast algorithm, whose results are shown in Figure 6, has a ratio of 0.65. The recursively doubling all-gather algorithm, whose results are shown in Figure 7, has a ratio of 1.00. For the Jaguar cluster (AMD) with 128 processes (we use 11 nodes with 8 cores in the last node), the ratio is 0.09 for the logical ring all-gather algorithm, 0.66 for the binomial tree broadcast algorithm, 1.0 for the recursive doubling all-gather algorithm.

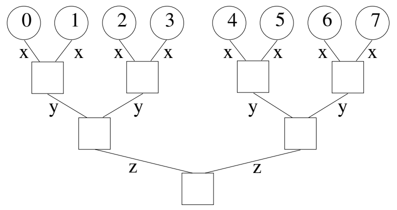


Fig. 3: Graph model for the nodes in the Intel cluster

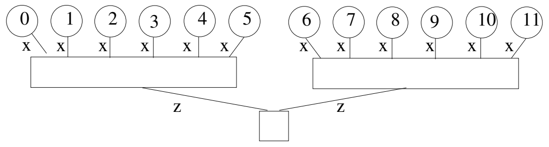
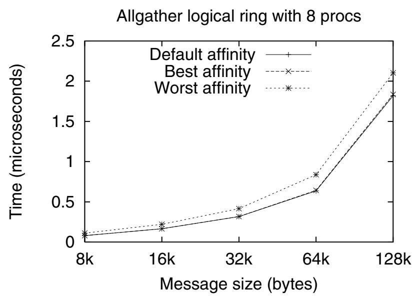


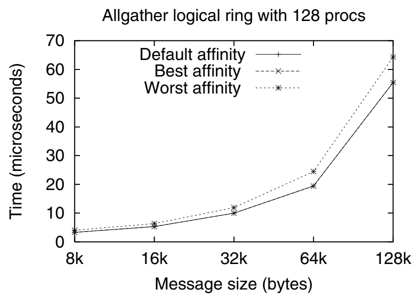
Fig. 4: Graph model for the nodes in the AMD cluster

As can be seen from the figures, when communications are within one node (all 8 processes cases), process affinity has significant impacts on all cases for both platforms: the best affinity and worst affinity result in significant different total communication times. This reveals the NUMA nature of the SMP nodes and indicates that process affinity can significantly affect the intra-node communication performance. The Jaguar cluster uses the most recent AMD processors, which is one generation newer than those in the Draco cluster. As a result, the intra-node communication is more efficient in the Jaguar cluster. For example, both with the best affinity setting, 8-process recursive-doubling all-gather of 128KB data takes 0.98 microseconds on Jaguar and 1.68 microseconds on Draco. Moreover, the impact of affinity is also less on Jaguar than on Draco. For example, for the logical-ring all-gather of 128KB data, the best affinity results in 17% improvement over the worst affinity on Jaguar and 100% improvement on Draco. This demonstrates the superiority of the newer generation Jaguar cluster.

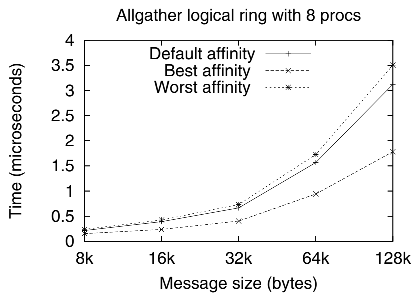
For the 128-process cases, however, processor affinity only has impacts on two schemes with smaller inter-node to intra-node communication ratios: the logical ring all-gather and the binomial tree broadcast algorithm. For the recursive doubling algorithm, even though process affinity affects intra-node communication performance, it does not change the overall communication time for the whole pattern. This is due to the significant inter-node communications that dominate the overall performance in this pattern. Another interesting observation is that for the logical ring algorithm and the binomial tree broadcast algorithm, the overall communication times are



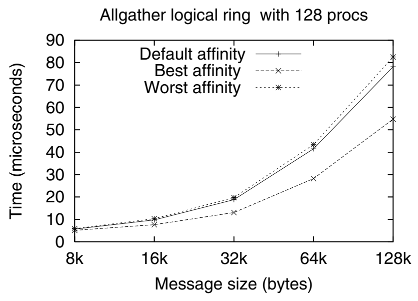
(a) 8 processes on the Jaguar



(b) 128 processes on the Jaguar

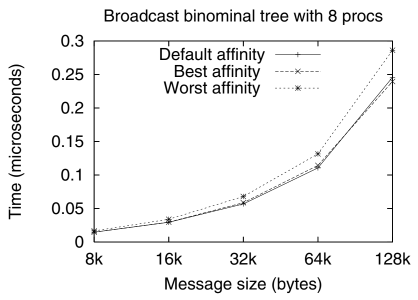


(c) 8 processes on the Draco

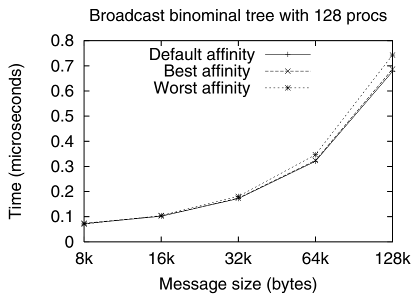


(d) 128 processes on the Draco

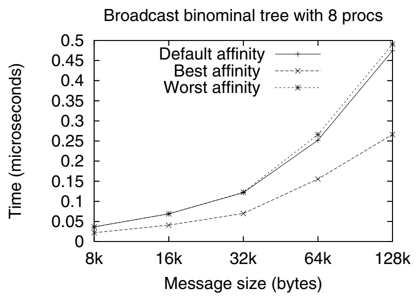
Fig. 5: Logical Ring All-gather Results



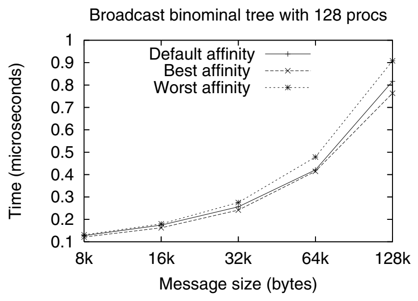
(a) 8 processes on the Jaguar



(b) 128 processes on the Jaguar



(c) 8 processes on the Draco



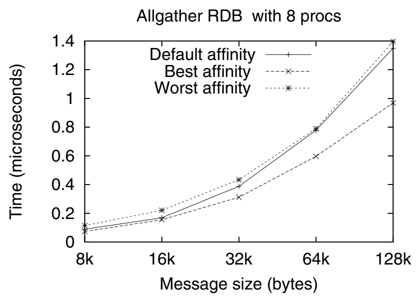
(d) 128 processes on the Draco

Fig. 6: Binomial Tree Broadcast Results

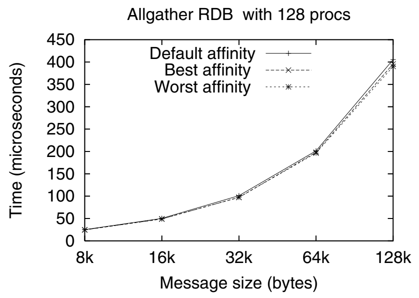
similar for both clusters with the best affinity settings. This indicates that the link bandwidth on the Jaguar cluster must be similar to that on Draco (20Gbps). However, for the more complex recursive-doubling all-gather, the communication times are much higher on Jaguar than on Draco. This is because Draco is connected by a single crossbar switch while Jaguar is connected by a large Interconnect, which can have network contention with complex communication patterns [12], [14].

B. Random communication pattern benchmarks

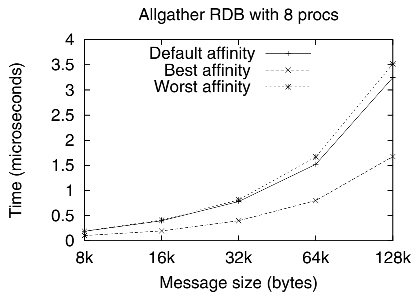
The results for the collective communication benchmarks clearly show that process affinity may or may not have impacts on a given communication pattern. It tends to significantly affect the performance communication patterns with low inter-node to intra-node communication ratios and has a negligible impact on communication patterns with high inter-node and intra-node communication ratios. In this experiment, we will use the random communication benchmark that allows the control of inter-node and intra-node communication ratios to further quantify the impacts of processor affinity, and to study the impact on different communication patterns and message sizes. All the results reported in this section are from Draco, the results for Jaguar have a similar trend. Each data



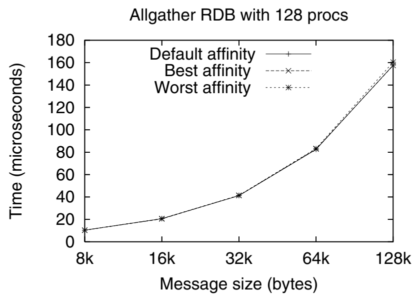
(a) 8 processes on the Jaguar



(b) 128 processes on the Jaguar



(c) 8 processes on the Draco



(d) 128 processes on the Draco

Fig. 7: Recursive Doubling All-gather Results

point reported is the average of 32 random samples. All the experiments are performed on 16 nodes (128 cores).

Figure 8 shows the effect of different inter-communication to intra-communication ratios. In this experiment, each node has a fixed of 15 random intra-node communications. Different inter-node to intra-node communication ratios are achieved with different numbers of random inter-node communications. The x-axis is the ratio. The y-axis is the performance improvement percentage that the best affinity achieves over the worst affinity. As can be seen from the figure, processor affinity has less effect as the ratio increases, which is consistent with

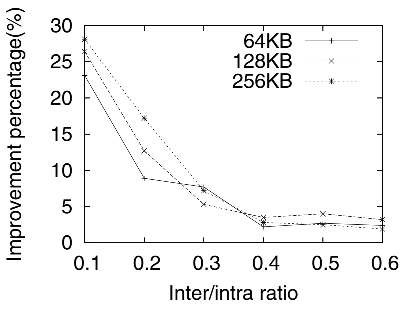


Fig. 8: Effect of the inter- and intra- communication ratio

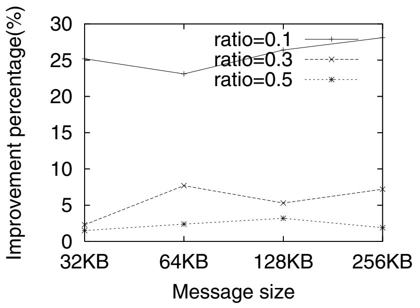


Fig. 9: Impacts on different message sizes

the results in the collective communication benchmarks. This applies to different message sizes.

Figure 9 shows the effect of processor affinity on different message sizes. This is a different presentation of the data in the last experiment. The figure shows that processor affinity has similar effect on different message sizes. When the message size is sufficiently large (32KB in our experiment), the impact of processor affinity is similar on different sizes.

Figure 10 shows the impacts on different communication patterns. The legend X/Y in the figure denotes that there are X intra-node communications within each node and the inter-node and intra-node ratio is Y. As can be seen from the figure, processor affinity has more impacts on sparse pattern with a small ratio such as the 15/0.1 patterns. When the number of communications is large, even with the same ratio, the impact is not as significant because with the same ratio, the number of inter-node communications is larger, which reduces the effect of processor affinity.

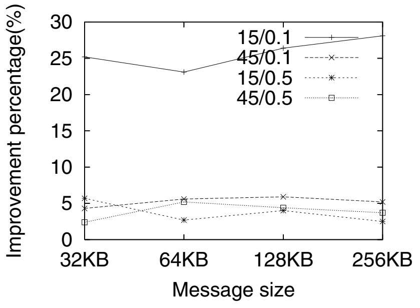


Fig. 10: Impacts on different communication patterns

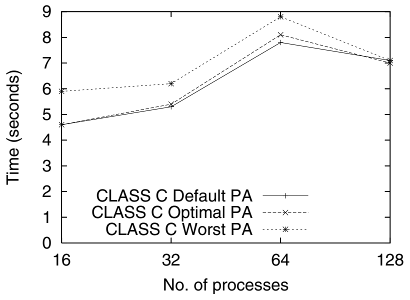
No. of processes		16	32/36	64	121/128
Jaguar	CG	0.143	0.231	0.407	0.600
	LU	0.200	0.486	0.534	0.571
	MG	0.219	0.433	0.992	0.989
	SP	0.500	0.500	0.959	1.793
Draco	CG	0.200	0.231	0.280	0.600
	LU	0.200	0.429	1.000	0.571
	MG	0.337	0.685	1.032	1.725
	SP	0.286	0.500	1.816	2.000

TABLE I: Inter-node/Intra-node communication ratios

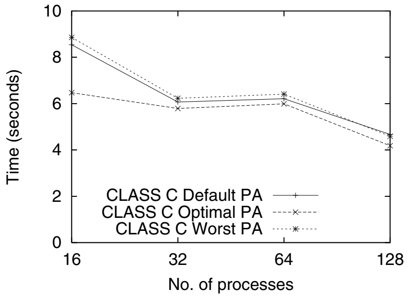
In summary, the impact of processor affinity is significantly affected by the inter-node and intra-node communication ratio and can be very significant when the ratio is small. Processor affinity has similar effects on communication patterns with different message sizes when the size is sufficiently large. For the same inter-node and intra-node communication ratio, processor affinity has more impact on patterns with less communications.

C. NAS benchmarks

This section reports the results for the NAS parallel benchmarks, including CG, LU, MG, and SP. We use class C problem size in all experiments and run the benchmarks with 16, 32/36, 64, and 128/121 processes. The inter-node and intra-node communication ratios of the four benchmarks in term of the total amount of data communicated are summarized in Table I. Figures 11 to 14 show performance results for the benchmarks. Since all of these benchmarks are dominated by the computation time, to understand the impacts on communications, we show the results for the total communication times in these programs. The communication time is obtained through PMPI that captures the entry and exit times for each of the communication routines.

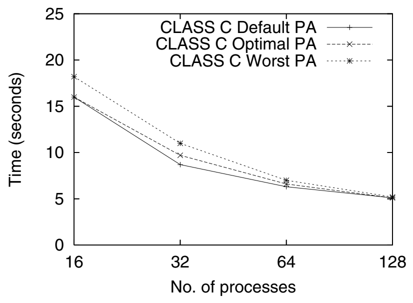


(a) Jaguar

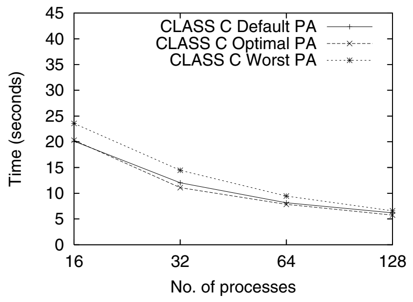


(b) Draco

Fig. 11: CG Results



(a) Jaguar



(b) Draco

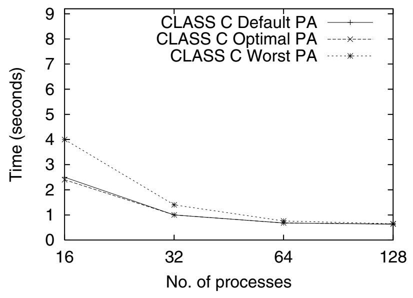
Fig. 12: LU Results

The results confirm that the inter-node intra-node communication ratio is a good indicator to predict whether the overall performance will be affected by processes affinity. With few exceptions, when the ratio is larger than 0.8, the impact of processor affinity is negligible and when the ratio is smaller than 0.5, the impact is noticeable. For all of the four benchmarks, as the number of nodes increases, the impact of processor affinity becomes less significant. On Jaguar, the impact for all four benchmarks becomes negligible with 128 processes.

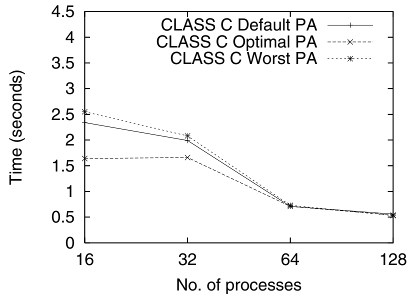
In some cases, the default affinity performs better than our "optimal" affinity. There are two reasons for this. First, default affinity allows processes to migrate among cores. This sometimes can have a better performance than fixing the affinity setting since it allows more load balancing among cores. Second, our "optimal" affinity is computed based on the communication trace for the whole program, which may yield lower performance when the program has phases of communications where in each phase, a subset communications dominate the performance. In this case, our "optimal affinity" may result in sub-optimal results.

VI. Conclusion

In this paper, we study the impacts of process affinity on MPI performance on SMP-CMP clusters. Through extensive experiments, we show that the impact of processor affinity is closely related to the inter-node to intra-node communication ratio. In particular, processor affinity can significantly affect the performance of patterns with sparse communications when the message size is sufficiently large. These results can be used as a guideline to decide whether processor affinity should be considered in several areas including the design and

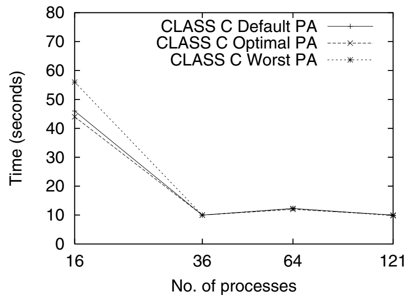


(a) Jaguar

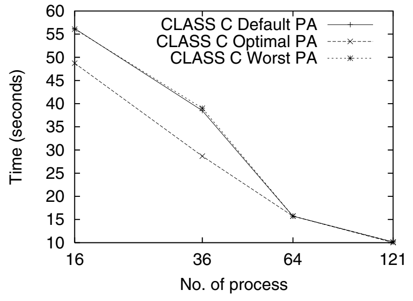


(b) Draco

Fig. 13: MG Results



(a) Jaguar



(b) Draco

Fig. 14: SP Results

evaluation of MPI collective communication algorithms, the development of performance models, and the understanding of MPI application performance in general.

Acknowledgment

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725, and was partially supported by DOE grant DE-FG05-08OR23338 and NSF grant CCF-0541096.

References

[1] Top 500 Supercomputer Sites. http://www.top500.org/.

[2] MPI over InfiniBand Project. http://mvapich.cse.ohio-state.edu/projects/mpi-iba.

[3] OPENMPI. http://www.open-mpi.org/.

[4] Juguar. http://www.nccs.gov/computing-resources/jaguar/.

[5] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[6] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter. Characterization of scientific workloads on systems with multi-core processors. *IEEE International Symposium on Workload Characterization*, pages 225–236, 2006.

[7] D. Buntinas, G. Mercier, and W. Gropp. Implementation and shared-memory evaluation of MPICH2 over the nemesis communication subsystem. *European PVM/MPI*, 2006.

[8] L. Chai, Q. Gao, and D. K. Panda. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 471–478, Washington, DC, USA, 2007. IEEE Computer Society.

[9] L. Chai, A. Hartono, and D. Panda. Designing high performance and scalable MPI intra-node communication support for clusters. *IEEE International Conference on Cluster Computing*, pages 1–10, 2006.

[10] L. Chai, P. Lai, H. Jin, and D. K. Panda. Designing an efficient kernel-level and user-level hybrid approach for MPI intra-node communication on multi-core systems. In *the 37th International Conference on Parallel Processing*, pages 222–229, 2008.

[11] E. Gabriel et. al. Open MPI: Goals, concept, and design of a next generation MPI implementation. *European PVM/MPI*, pages 97–104, 2004.

[12] P. Geoffray and T. Hoefler. Adaptive routing strategies for modern high performance networks. *16th IEEE Symposium on High Performance Interconnects*, 2008.

[13] B. Goglin. High throughput intra-node MPI communication with Open-MX. *the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, 2009.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage switches are not crossbars: Effects of static routing in high performance networks. *IEEE International Conference on Cluster Computing*, pages 116–125, 2008.

[15] K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop, and D. K. Panda. Designing multi-leader-based allgather algorithms for multi-core clusters. In *9th Workshop on Communication Architecture for Clusters (CAC 09)*, pages 1–8, May 2009.

[16] R. Kumar, A. Mamidala, and D. Panda. Scaling alltoall collective on multi-core systems. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[17] A. Mamidala, R. Kumar, D. De, and D. Panda. MPI collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *the 8th IEEE International Symposium on Cluster Computing and the Grid*, pages 130–137, May 2008.

[18] H. Pourreza and P. Graham. On the programming impact of multi-core, multi-processor nodes in MPI clusters. *the 21st International Symposium on High Performance Computing Systems and Applications*, 2007.

[19] Y. Qian and A. Afsahi. Efficient shared memory and rdma based collective on multi-rail qsnetii SMP clusters. *Cluster Computing*, 11(4):341–354, 2008.

[20] R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP parallel programming on clusters of multi-core smp nodes. *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 427–436, February 2009.

[21] T. Technologies. White paper: Processor affinity. November 2003. Available at: http://www.tmurgent.com/WhitePapers/ProcessorAffinity.pdf.

[22] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[23] X. Wu, V. Taylor, C. Lively, and S. Sharkawi. Performance analysis and optimization of parallel scientific applications on CMP cluster systems. *ICPP SMECS Workshop*, September 2008.