Locality and Topology aware Intra-node Communication Among Multicore CPUs

Teng Ma, George Bosilca, Aurelien Bouteiller and Jack J. Dongarra

Innovative Computing Laboratory,
University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
{tma, bosilca, bouteill, dongarra}@eecs.utk.edu

Abstract. A major trend in HPC is the escalation toward *manycore*, where systems are composed of shared memory nodes featuring numerous processing units. Unfortunately, with scale comes complexity, here in the form of non-uniform memory accesses and cache hierarchies. For most HPC applications, harnessing the power of multicores is hindered by the topology oblivious tuning of the MPI library. In this paper, we propose a framework to tune every type of shared memory communications according to locality and topology. An implementation inside Open MPI is evaluated experimentally and demonstrates significant speedups compared to vanilla Open MPI and MPICH2.

1 Introduction

Because the emergence of thermic and power issues have prevented further performance improvements through the usual frequency scaling, CPU vendors have resorted to multicore architectures to deliver the expected level of performance progression. Unfortunately, incorporating more processing units does not give an instant and automatic speed boost to applications; programmers have to take into account numerous issues posed by the intrinsic parallel and heterogeneous nature of multicore chips. Although HPC developers have become proficient at harnessing the power of parallel systems, through the use of various programming models such as Single Process Multiple Data (SPMD) and tools like MPI or OpenMP, straight out applications of those paradigm on cluster of multicores types of architecture have exhibited disappointing performance [1]. To enable the integration of more cores inside a computing node, vendors are forced to expose more complex architectures, exhibiting Non Uniform Memory Accesses (NUMA) and several levels of partitioned cache hierarchies. Furthermore, design and implementation of multi-core architecture present a large diversity among vendors. As an example, Intel's Tigerton CPUs feature a SMP architecture, while AMD's Istanbul and Intel's Nehalem exhibit NUMA characteristics. In a node, some cores reside on different sockets, interconnected by network-style fast connections such as Intel's QuickPath and AMD's HyperTransport. Even inside a single die, one can encounter different L2 caches, shared between exclusive

groups of cores, so that two cores of the same processor might or might not share the same level of cache, depending on their respective position on the die.

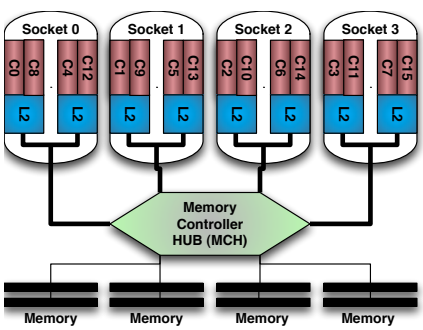
While hybrid approaches and novel programming models are being investigated, the large majority of applications available in the HPC ecosystem today are based on the message passing paradigm (using the MPI standard). Converting every and each of those applications to take into account the fine subtleties of the various and changing vendor implementations of multicore systems would impose a significant and lasting burden to the community. Among the issues preventing message passing from delivering performance on cluster of multicore systems is the use of a flat set of tuning parameters for all shared memory communications, regardless of the underlying hardware architecture, more precisely the distance to different levels of cache and memory and the physical topology imposed by the chips. In this paper, we propose to alleviate this issue by providing a topology aware framework inside the message passing middleware, in order to unleash legacy application performance on the most recent architectures without shifting the programming model. The prominent feature of this framework is to optimize intra-node communication by selecting the optimal tuning parameter set at runtime. Multiple communication parameter sets are provided and can be selected, according to the run-time placement of the MPI processes and considering the topology of the underlying hardware.

The rest of this paper is organized as follows: Section 2 introduces the related work on multi-core intra-node communication. Section 3 formulates and outlines the extent of the problem when considering modern multicore processors. Then Section 4 describes our framework designed to combine locality and topology information with intra-node communication, and its implementation in a leading MPI implementation. A performance study is presented in the Section 5, substantiating the benefits of this approach when compared to the Open MPI and MPICH2 implementations. Finally, Section 6 concludes the paper with a discussion of the results and future directions.

2 Related Work

MPICH2 [2] and OpenMPI [3,4] are the two major implementations of the MPI standard. Both feature an optimized device to handle shared memory communications: Nemesis [5] for MPICH2 and the SM BTL for Open MPI. In both MPI implementations, large messages are divided into fragments to establish a pipeline. The smallest message to use the pipeline protocol as well as the fragment size are examples of crucial parameters to reach maximum bandwidth without sacrificing latency. The OPTO tool [6] has been proposed to optimize the run-time parameters of the Open MPI environment. It uses a brute-force searching of the parameter space by evaluating benchmarks such as NetPipe, for point-to-point communication, and SkaMPI, for collective communication. This set of *tuned* parameters is then used for every communication of any application.

In regular MPI shared memory implementations, any transfer actually involves two memory copies: one from the user buffer to the shared memory buffer



(a) Four sockets Intel Tigerton node (b) Two sockets Intel Nehalem node

Fig. 1. Architecture comparison between 2 generations of Intel multicore CPUs

and another to the destination user buffer. The LiMIC [7] kernel module can decrease the number of necessary memory copies to one by doing the memory movement with kernel access rights. KNEM [8] is a similar kernel module that also features DMA (Direct Memory Access) copy by using Intel I/O acceleration technique (I/OAT). DMA copy can decrease cache pollution and CPU noise from communication. However, DMA performance suffers when multiple communications overload a single DMA device, which is very likely with the current trend to increase the number of cores. In that context, rather than easing the tuning process, using kernel-based DMA methods is another parameter that changes according to the communication workload.

Several efforts have proposed to embrace the hierarchical nature of Grid systems network [9,10,11]. These papers propose different approaches to map the collective communication topology to the actual network topology, an idea that applies as well to multicore processors. Yet, our work focuses on the optimization of point-to-point message, and its indirect improvement on collective communication performance. The optimization and tuning of the collective algorithm itself, according to the hardware topology, is left for future works.

While shared memory communication tuning has been an active research area, using the underlying hardware topology to define different tuning parameter sets, as we propose in our framework, has never been attempted.

3 Multicore and Multifarious Hierarchies

Modern CPUs exhibit several levels of cache, with non uniform memory accesses. The communication distance between two cores of a single node varies depending on those hierarchies. Furthermore, each vendor exhibits different characteristics that tends to radically change between successive CPU generations. Figure 1(a) and Figure 1(b) illustrate such differences by describing two typical cluster nodes featuring different generations of Intel multicore CPUs. Figure 1(a) shows the architecture of a node with four Intel's Tigerton CPUs (16 cores). In this machine,

all sockets are interconnected by one memory controller. Thus, the apparent distance to the memory is the same for every core. However, three different communication path exist with distinct costs. The first one is between core 0 and core 8 which are on the same die and share the L2 cache. While core 0 and core 4 do not share L2 cache, communication inside the same socket are significantly faster than resorting to the FSB (front side bus). Between core 0 and core 2, hosted in different sockets, the FSB is the only option. Figure 1(b) describes the architecture of a node with two Intel Nehalem CPUs (8 cores). Each processor has an independent memory controller, which makes it a NUMA architecture. While all cores of a socket share a common L3 cache, the Nehalem architecture also exhibits different communication performance whether the cores are on the same socket or not.

SPMD had been a very successful programming model for single core architectures. Consequently, numerous applications and libraries have been developed following this approach, and have benefited from its easy portability across different vendors, and excellent level of performance. However, in the context of multicore CPUs, not considering the locality and topology of the cores distribution inside the CPUs dreadfully affects the overall application experienced communication performance.

Nowadays MPI implementations provide a specific optimized device to handle shared memory communications. This device is usually applied directly to core-to-core communications with a single set of tuning parameter oblivious of the topology between the sender and receiver cores. As an example, in the Nemesis device of MPICH2, when the message size is smaller than `PIPELINE_THRESHOLD` (128KBytes), the `copy_limit`, defining the pipeline size, is set to 16KBytes. For larger messages, this parameter is changed to `MPID_MEM_COPY_BUF_LEN` which is often 32KBytes. Open MPI also has a set of similar communication parameters (`btl_eager_limit` and `btl_max_send_size`) which are used for protocol switch point and pipeline size respectively. Unlike MPICH2, in Open MPI users can tune those parameters without recompiling the MPI library. Despite this added flexibility, users rarely have the expertise and time to properly tune parameters for the communication pattern of their application, leading most runs to use the default parameters. Furthermore, for any run of an application, a single set of tuning parameters can be used. Therefore, it is impossible to apply a different set of tuning parameters to communications to account for different characteristics of the links between cores. The experimental section of this paper (5) provides an evaluation of the extent of the performance issue induced.

4 Multi-tuning Framework

Because Open MPI is based on a modular and component model while at the same time retaining outstanding performance, it is a very convenient vessel to investigate new features. Thus, while the principles presented are generic, our topology aware multicore communication framework is implemented into Open MPI. The framework is composed of three main components: the rule dis-

CPU	locality	btl_eager_limit	pipe_size	use_knem	DMA_min
Tigerton	no shared L2 cache	2096	0.5 * L1 cache	1	2196608
Nehalem EP	no shared L2 cache	4192	0.5 * L1 cache	0	null
Tigerton	shared L2 cache	2096	L1 cache	1	4804864

Table 1. An example of rule discovery table

covery module, the machine topology discovery module and the runtime communication tuning module.

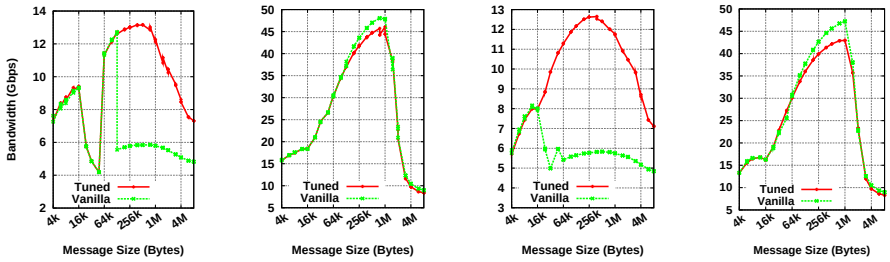
The rule discovery module is used offline to construct a table storing knowledge about best tuning parameters for a particular architecture. In this table, we store common knowledge about the relationship between CPU architecture, locality, topology and tuning parameters. Table 1 is an example of a generated rule table, where the tuning of various pipeline length and the use of a DMA engine is governed by the sharing of an L2 cache. Rules are inferred from a mathematical model taking into account the size of the L1 data cache, L2 cache and the sharing of cache hierarchies between cores. As an example, if two cores share the L2 cache, the heuristic is to use half of L1 data cache size as the pipeline size. Most rules depends on the cache reuse policy and the snoopy cache protocol. Different sets of rules have been defined for different families of processors. While more experimental evaluations are needed to assert the soundness of the models proposed, the results presented in the experimental section of this paper are encouraging. Should some architecture be difficult to describe with a mathematical model, the previously discussed parameter exhaustive OPTO tool could also be used to build the rule table.

The machine topology discovery module discover, once for every run, all information about the cache hierarchies, core mapping and proximity between each pair of cores. This module is based on the Portable Hardware Locality (hwloc) [12] project. It provides a portable abstraction (OS, versions, architectures, etc.) of the hierarchical topology of modern CPUs, including NUMA memory nodes, sockets, shared caches, cores and hyper-threading. It also gathers various system attributes such as cache and memory information.

Based on the tables generated by the two discovery modules, the runtime communication tuning modules instantiates several distinct SM BTL with adequate tuned parameters for each type of communications. Then, among the available instantiation of the SM BTL, for every message, the best one [13] is selected to actually transfer the data, based on the rules applied to this type and size of message and the source to destination core distance.

5 Experimental Evaluation

Experimental Conditions. Our experimental setup includes two different Intel based machines. The first one is based on four Intel Xeon E7340 at 2.4GHz (Tigerton), as described by Figure 1(a). Its L1 data cache is 32KB and L2 data cache is 4MB. The four sockets are interconnected each other by the front side bus. The second system is based on two Intel Xeon E5520 at 2.27GHz (Nehalem),



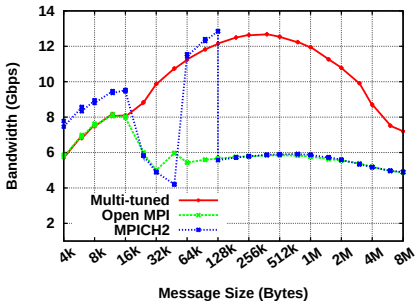
(a) MPICH2 off-die (b) MPICH2 on-die (c) Open MPI off-die (d) Open MPI on-die

Fig. 2. Impact on bandwidth of pipeline fragment size tuning according to core distance on the Tigerton machine for MPICH2 and Open MPI

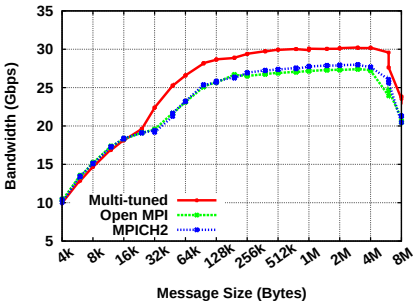
as described by Figure 1(b). Its L1 data cache is 32KB, and each core has an independent L2 data cache whose size is 256KB. CPUs are interconnected by Intel QuickPath. The same operating system (Linux 2.6.30) is deployed on both machines. MPICH2-1.2.1 and Open MPI trunk (r22930) are used. We used NetPIPE [14], Intel MPI benchmarks [15] and the NAS parallel benchmarks [16] to evaluate the performance of our tuning framework. All benchmarks are compiled with gcc 4.1.2, with the -O3 flag.

Assessment of the Severity of the Performance Issues. The first set of experiments evaluates the performance loss incurred by using a single set of tuning parameters, regardless of core locality. Figure 2 presents the performance comparison between a vanilla version of MPICH-2 and Open MPI with a similar version where pipeline size has been hand-tuned for maximum inter-socket bandwidth. In vanilla MPICH-2, for messages larger than 128KB, the pipeline size switches from 16KB to 32KB. As the steep bandwidth drop illustrates in Figure 2(a), this is a very inappropriate tuning for communications between cores located in different sockets. The hand tuned version, that retains the original 16KB pipeline for larger messages, is capable of sustaining a higher bandwidth, up to a very significant 2.5 times improvement. Open MPI exhibit the same behavior (Figure 2(c)), illustrating that the issue is not implementation specific. However, as illustrated by the Figures 2(b) and 2(d), when communicating inside the same die, the default parameters are perfectly tuned and perform slightly better. While the hand tuned parameters yield significant benefits in certain cases (inter-socket communications), using them in certain cases decrease performance, illustrating the need for using *simultaneously* different sets of tuning parameters for different types of communications.

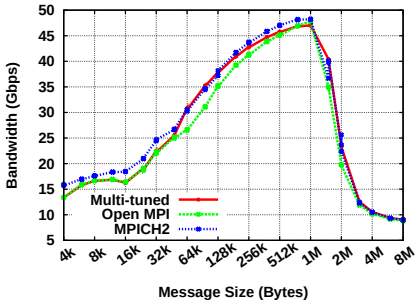
Effectiveness of the Multi-tuning Framework. The four Figures 3 presents the comparison between vanilla MPICH2, vanilla Open MPI and multi-tuned Open MPI in the NetPIPE ping-pong benchmark for a variety of machines and core distributions. The bandwidth values for message smaller than 4KB have been removed for clarity, as the performance of the three versions were similar. On the



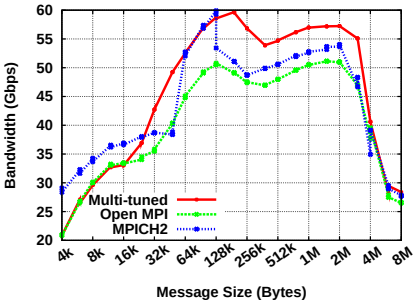
(a) Tigerton, inter-socket $C_0 \rightleftharpoons C_2$



(b) Nehalem, inter-socket $C_0 \rightleftharpoons C_1$

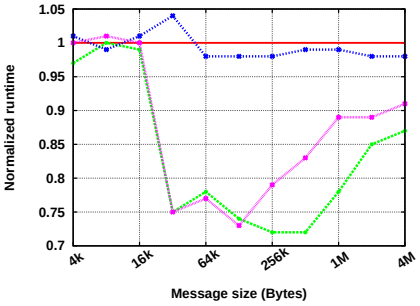


(c) Tigerton, intra-socket $C_0 \rightleftharpoons C_8$

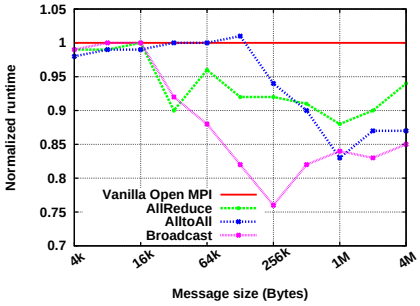


(d) Nehalem, intra-socket $C_0 \rightleftharpoons C_2$

Fig. 3. Bandwidth of the ping-pong test for vanilla MPICH2, vanilla OpenMPI and multi-tuned Open MPI



(a) Tigerton platform



(b) Nehalem platform

Fig. 4. Run time of the IMB collective tests of the multi-tuned Open MPI (normalized to the vanilla Open MPI performance, lower is better)

Tigerton machine, the multi-tuned version outperforms both vanilla Open MPI and MPICH2 for inter-socket communications (Figure 3(a)), thanks to using better tuning. Contrarily to the naive hand-tuned version presented in the previous experiment (Figure 2(d)), it does not suffer from any performance degradation for intra-socket communication (Figure 3(c)). The same holds with the Nehalem processor; though initially MPICH2 performs better than Open MPI, the multi-

tuned Open MPI version outperforms both vanilla MPI for intra-socket communications. Though all cores are on the same die, they don't share L2 cache, which is the prominent performance affecting factor.

name	Nehalem (8 cores)			name	Tigerton (16 cores)		
	Open MPI	Tuned	Speedup		Open MPI	Tuned	Speedup
IS.C	3.72s	3.48s	6.9%	IS.C	6.27s	6.22s	0.8%
FT.B	15.81s	15.31s	3.3%	FT.B	24.53s	24.45s	0.32%
LU.C	278.34s	276.81s	0.55%	LU.C	209.07s	204.64s	2.16%
MG.B	2.60s	2.57s	1.2%	MG.B	4.94s	4.91s	0.61%
CG.C	46.42s	46.34s	0.17%	CG.C	97.31s	96.21s	0.93%

Table 2. Run time of the NAS benchmarks

Collective Communications. Figure 4(a) and Figure 4(b) present the run time of multi-tuned OpenMPI for Broadcast, AlltoAll, and AllReduce collective communication, on the Tigerton and Nehalem machines, normalized to the run time of the similar algorithm in vanilla Open MPI. In this experiment, the multi-tuning only takes place at the the point-to-point communication level underlying the collective algorithm; the collective algorithm itself is left unchanged. For messages smaller than 4KB, the physical page size, multi-tuned and vanilla Open MPI always use the same eager protocol, exhibiting equal performance (thus not presented on the graph); yet significant performance improvement are visible for larger message sizes.

In the AlltoAll test, while multi-tuned Open MPI reduces the execution time by only 2% on the Tigerton platform, it yields up to 17% improvement on the Nehalem platform. This result can be explained by the communication pattern of the shared memory AlltoAll collective operation: it does not use a tree topology. On the Tigerton platform, the cross bandwidth of the FSB is consequently easily saturated by this naive algorithm, negating the inter-socket bandwidth benefit achieved on the simple point-to-point benchmark. As multi-tuning does not yield much gains for the intra-socket communication on the Tigerton architecture, the overall benefit is small. On the contrary, the Nehalem platform features only two sockets connected through the much faster QPI interface and adapts better to cross-traffic. Moreover, intra-socket point-to-point bandwidth is also improved on this system, which transfers as well to the collective performance.

In the AllReduce test, multi-tuned OpenMPI benefits from a 12% run time reduction on Nehalem and up to 23% for the very common 256KB message size on Tigerton. Both platform exhibit a close to 25% performance improvement for some message sizes on the Broadcast collective operation. In these two collectives, the shared memory collective algorithm uses a tree topology which does not saturate the inter-socket link; therefore, benefits of multi-tuning on point-to-point performance are reflected in the collective performance. For the entire range of message sizes, multi-tuned collective operations compare favorably to vanilla Open MPI, except for 32KB on the Tigerton platform.

Applications. Additionally, we used application benchmarks to evaluate the performance of our framework. We used IS, FT, LU, MG and CG from the NAS benchmarks. Table 2 shows the comparison between the runtime of the multi-tuned version of Open MPI and vanilla Open MPI for these benchmarks. Compared with regular MPI, the multi-tuned approach always decreases the overall application runtime. As communication are using the extremely fast shared memory device, the communication to computation ratio is balancing toward computation bound performance. As a consequence, the overall impact of communication performance on the application runtime is small, an effect more pronounced on the benchmark achieving good communication overlap by computations such as MG and CG. Though the CG benchmark is communication intensive, only its latency bound communications are difficult to overlap, an area where tuning is already adequate by default. The FT benchmark, which uses an all-to-all collective communication, exhibits a similar performance profile as the AlltoAll test, with almost no gain on the Tigerton platform but some improvement on the Nehalem. The maximum performance improvement of multi-tuning is achieved in the communication intensive IS benchmark on the Nehalem platform, with close to 7% application run time improvement.

6 Conclusion and Future Work

In this paper, we studied the problem of intra-node communication inside multicore CPUs. Our experiments show that ignoring the locality and topology information in the MPI software stack is an obstacle to harness the optimal communication performance on multicore systems. We then introduced a framework to 1) build tuning rules for different models of CPUs, 2) discover the run-time information such as CPU type, cache size, locality and etc. and 3) take advantage of this knowledge to finely tune the internals of the MPI library for different kind of communications. Our experiments show that the multi-tuned Open MPI version based on this framework, always exhibits better application performance, thanks to improved communication speed for both point-to-point and collective communication patterns. With the future increase in the number of cores per node this benefit is expected to be magnified.

Future Works While fine tuning of point-to-point communication to adapt to the multicore topology has proven beneficial indirectly to the collective communications, tuning the collective itself according to the same information has the potential to further increase performance. We expect to witness the same hardware locality dependent tuning for the selection and parametrization of the collective algorithm itself, an area where we believe our multi-tuning approach will be valuable.

References

1. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: Parallel, Distributed and Network-based Processing. (2009) 427 –436

2. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing **22**(6) (1996) 789–828
3. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (2004) 97–104
4. Graham, R.L., Woodall, T.S., Squyres, J.M.: Open MPI: A flexible high performance MPI. In: Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics, Poznan, Poland (2005)
5. Buntinas, D., Mercier, G., Gropp, W.: Design and evaluation of Nemesis, a scalable, low-latency, message-passing communication subsystem. Cluster Computing and the Grid, 2006. Sixth IEEE International Symposium on **1** (2006) 10–20
6. Chaarawi, M., M.Squyres, J., Gabriel, E., Feki, S.: A tool for optimizing runtime parameters of Open MPI. In: Proceedings, 15th European PVM/MPI Users' Group Meeting. Number 5205 in LNCS, Springer Verlag (2008) 210–217
7. Jin, H.W., Sur, S., Chai, L., Panda, D.: LiMIC: support for high-performance MPI intra-node communication on linux cluster. Parallel Processing, 2005. ICPP 2005. International Conference on (2005) 184–191
8. Buntinas, D., Goglin, B., Goodell, D., Mercier, G., Moreaud, S.: Cache-Efficient, Intranode Large-Message MPI Communication with MPICH2-Nemesis. In: Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009), Vienna, Austria, IEEE Computer Society Press (2009) 462–469
9. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: Magpie: Mpi's collective communication operations for clustered wide area systems. In: Proceedings of the 1999 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'99). (1999) 131–140
10. Karonis, N.T., de Supinski, B.R., Foster, I., Gropp, W., Lusk, E., Bresnahan, J.: Exploiting hierarchy in parallel computer networks to optimize collective operation performance. The 14th International Parallel and Distributed Processing Symposium (2000) 377
11. Filgueira, R., Singh, D.E., Pichel, J.C., Isaila, F., Carretero, J.: Data Locality Aware Strategy for two-phase Collective I/O. In: VECPAR 2008: 8th Intl. Conference High Performance Computing for Computational Science. (2008) 137–149
12. Broquedis, F., Clet Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R.: hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In: The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing. (2010)
13. Shipman, G.M., Woodall, T.S., Bosilca, G., Graham, R.L., Maccabe, A.B.: High performance RDMA protocols in HPC. In: Proceedings, 13th European PVM/MPI Users' Group Meeting. LNCS, Bonn, Germany, Springer-Verlag (2006)
14. Snell, Q.O., Mikler, A.R., Gustafson, J.L.: NetPIPE: A network protocol independent performance evaluator. In: in IASTED International Conference on Intelligent Information Management and Systems. (1996)
15. Intel: Intel MPI benchmarks 3.2. http://software.intel.com/en-us/articles/intel-mpi-benchmarks/ (2010)
16. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The NAS parallel benchmarks. Technical report, The International Journal of Supercomputer Applications (1991)