Optimizing MPI Communication within large Multicore nodes with Kernel assistance

Stéphanie Moreaud, Brice Goglin, David Goodell, Raymond Namyst

Optimizing MPI Communication Within Large Multicore Nodes with Kernel Assistance

Stéphanie Moreaud, Brice Goglin, Raymond Namyst
INRIA, LaBRI, Université of Bordeaux
351, cours de la Libration
F-33405 Talence – France
Email: {smoreaud,goglin,namyst}@labri.fr

David Goodell
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, USA
Email: goodell@mcs.anl.gov

***Abstract*—As the number of cores per node increases in modern clusters, intra-node communication efficiency becomes critical to application performance. We present a study of the traditional double-copy model in MPICH2 and a kernel-assisted single-copy strategy with KNEM on different shared-memory hosts with up to 96 cores.**

We show that KNEM suffers less from process placement on these complex architectures. It improves throughput up to a factor of 2 for large messages for both point-to-point and collective operations, and significantly improves NPB execution time. We detail when to switch from one strategy to the other depending on the communication pattern and we show that I/OAT copy offload only appears to be an interesting solution for older architectures.

I. Introduction

The widespread availability of multicore processors and NUMA architectures leads to an increasing complexity inside computing nodes, with many cores, shared caches, sockets and NUMA nodes. MPI is the standard inter-node communication interface for clusters and it is still widely used for intra-node communication even though many efforts target hybrid programming models such as MPI +OpenMP [1].

Achieving high-performance in modern multicore clusters requires both inter-node and intra-node communications to be efficient. Most high-performance MPI implementations such as MPICH2 offer low latency for small intra-node messages [2] but still fail to easily achieve high-throughput for large messages. Moreover, the increasing complexity and non-uniformity of modern machines causes performance to vary significantly with process locations and the hardware resources they share.

In this article we target the optimization of large message intra-node communication in the context of wide multicore machines. Previous work [3], [4] introduced operating system assistance as a way to improve large message throughput. We present an in-depth study of this solution in the context of complex shared-memory machines. Their hierarchical architecture topology has a strong influence on the performance of data transfers and it significantly impacts the choice between existing communication strategies.

The remainder of the article is organized as follows. We introduce in Section II our motivations, objectives and methodology. The study of point-to-point and collective operations is then presented in Sections III and IV. Before concluding, some application performance results are presented in Section V while Section VI discusses our results.

II. High-performance intra-node MPI communication

A. Background

The emergence of multicore processors drives many research projects towards designing shared memory programming models such as OpenMP [5]. However, the MPI +OpenMP hybrid programming model can be difficult to use and is not appropriate for many applications [1]. Therefore, applications often rely on the MPI model even for intra-node communication.

Naively implementing intra-node communication in MPI via the network can have substantial overhead. That is why popular MPI implementations offer a dedicated intra-node communication strategy bypassing the network subsystem and try to move data from one process to another as fast as possible.

B. Traditional double-copy implementation

Passing messages from one process to another requires either specific support from the operating system or the use of a shared memory buffer. The former however causes the overhead of system calls (about 100 ns under Linux on current Intel processors) to increase latency. Most popular MPI implementations, such as Open MPI [6], MPICH2 [2] and MVAPICH [7], thus rely on the second model which is also more portable because it only relies on standard shared memory support from the operating system. This method always results in two copies, one from the source buffer into the intermediate shared buffer and another out of the shared buffer into the destination buffer. If two processors are participating in the transfer, the copies might overlap to some degree, one thereby partially hiding the cost of the other.

However, this method requires both processors to actively take part in the transfer, which prevents them from performing useful application computation. It also pollutes the caches by evicting application data from it as the copy operation is being performed [8]. In the end, this strategy shows very interesting latency for small messages but it is not recommended for large messages [3], [4].

C. Kernel and hardware assistance

Large message performance issues in the double-copy implementations have been widely noticed and thus lead to the development of kernel-assisted strategies that rely on a single copy. Such strategies appeared in many interconnect-specific stacks, such as MYRICOM MX, QLOGIC PSM and OPEN-MX [9]. Some hardware and operating system specific optimizations were also proposed. For instance, some particular memory management features enable improvement of both latency and throughput [10]. Some advanced data transfer capabilities in the memory controller also help large message throughput [9]. Also, it has been shown that using the software loopback of modern network interface [11] may significantly improve performance.

We focus in this article on offering high-throughput intra-node messaging for generic and portable MPI implementations. MPICH2 is a widely portable, high-performance implementation of version 2.2 of the MPI standard [12]. It uses a communication subsystem called NEMESIS, which employs a double-copy scheme for intra-node communication and various networks for inter-node communication. It offers very low latency [2] but its large message throughput suffers from the double-copy model.

In [8] we evaluated several methods for large-message communication over shared memory, We previously experimented with single-copy transfers within OPEN-MX, an ETHERNET-specific message-passing stack [9]. OPEN-MX is even able to offload this copy on I/OAT hardware [9], but it requires ETHERNET networking for inter-node communication. It led to the development of the generic KNEM module to improve performance of generic purpose MPI implementations thanks to kernel-assistance [4] in Linux. KNEM is now used by both MPICH2 and OPEN MPI[1] .

LIMIC2 [13] also implements a similar single-copy model for the MVAPICH stack. It however raises serious security concerns that prevent it from being deployed in production machines.[2] Moreover, LIMIC2 does not offer vectorial buffer support, asynchronous request processing and I/OAT copy offload as KNEM does.

I/OAT is a set of hardware features implemented in modern INTEL memory controllers [14]. I/OAT copy offload was originally designed as a way to improve datacenter networking performance by reducing the TCP stack overhead on the receiver side. It has been proven to also help moving large amounts of data between processes [15]. We further explained in [4] that the combination of all strategies, double-copy, KNEM kernel-assisted single-copy and KNEM I/OAT-offloaded copy, is necessary to optimize performance depending on message size and hardware cache characteristics.

We now intend to give a deeper look at this problem in the context of both point-to-point and collective operations in large shared-memory machines. We look at complex hardware architectures and communication patterns and try to dynamically choose between the available strategies depending on process placement and hardware characteristics.

III. POINT-TO-POINT OPERATIONS

We first look at point-to-point operations so as to understand the performance impact of process placement on complex hardware architectures.

A. Experimental platforms

Our experimental platform is composed of 4 different INTEL-based machines:

Bill8 is a dual-socket quad-core 2.33 GHz XEON *Clovertown* E5345 with 4 MiB L2 caches shared between pairs of cores.

Hannibal8 is a dual-socket quad-core 2.66 GHz XEON *Nehalem* X5550, with a 8 MiB shared L3 cache per socket. Hyperthreading is ignored.

Idkonn24 is a quad-socket hexa-core 2.66 GHz XEON *Dunnington* X7460, with 3 MiB L2 caches shared between pairs of cores and a 16 MiB shared L3 cache per socket.

Bertha96 is an IBM x3950M2 assembling 4 Idkonn24-like machines into a single cache-coherent host with 4 NUMA nodes and 4 hexa-core processors per node. It is the only machine that does not offer I/OAT copy offload.

Numerous relative process placements are possible: 2 processes may run on 2 cores that share a L2 and/or L3 cache, they may also run inside the same socket without sharing a cache (on Bill8), inside different sockets of the same NUMA node, or inside different sockets of different NUMA nodes (on Bertha96).

MPICH2 1.2 was configured with NEMESIS default double-copy implementation, KNEM kernel-assistance, and KNEM I/OAT copy-offload model. KNEM 0.6.0, *Intel MPI Benchmarks* 3.2 [16] and *NAS Parallel Benchmarks* 3.3 [17] were used.

To better suit IMB and NPB requirements, only power-of-two numbers of processes were used. The last 2 cores of hexa-core processors on Idkonn24 and Bertha96 were ignored as if using quad-core sockets.

[1] Will be available in OPEN MPI 1.5.

[2] LIMIC2 passes kernel pointers in user-space without verifying that what the application passes back is valid. It lets any process crash the OS and read any other process' memory virtually unhindered.

B. Impact of cache sharing

Intra-node communication performance is subject to the influence of cache sharing between the processing cores since memory copies between the same buffers are repeated multiple times. Indeed caches have a significant impact on local communication performance [4].

IMB supports an *offcache* option preventing such repeated use of the same buffers. To clarify these caching issues, Figure 1 presents a IMB Pingpong between 2 cores not sharing a cache on Bill8. Comparing both NEMESIS and KNEM with and without *offcache* shows a huge performance difference, from about 1 GiB/s to 4 GiB/s with KNEM. This fake throughput is caused by buffer being reused and the data not being transfered for real. The only case where caching does not matter is when I/OAT is used since the data is directly transfered between memory buffers without any hardware cache in-between.

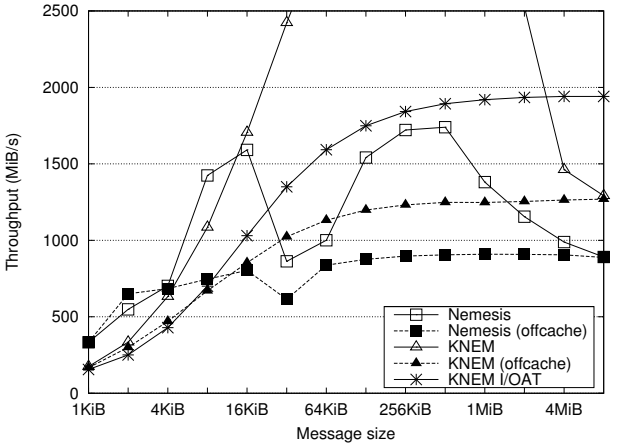

Figure 1. IMB Pingpong on Bill8 between 2 cores of different sockets, using Nemesis, KNEM with and without I/O AT, depending on whether the IMB Offcache option is enabled.

We assert that most real applications use the caches for actual computation and thus do not benefit that much from caching of communication buffers. For this reason, all IMB tests in the rest of the article will be presented with *offcache* enabled, assuming it better represents the performance that real applications may expect. While this methodology shows lower throughput than [3], [4], it fortunately brings comparable behaviors, especially regarding the threshold that determines when to switch from NEMESIS to KNEM: KNEM becomes interesting once the message size passes 16 KiB. It is also worth noticing here that I/OAT copy offload brings interesting performance improvements (up to 80%) as soon as KNEM is used. We observed the same behavior when binding processes inside a shared L2 cache.

C. Impact of process placement

We now look at the actual impact of process placement on the NEMESIS and KNEM performance. Figure 2 shows the Pingpong throughput on Bertha96 (I/OAT is not supported) when processes share a L2 and/or L3 cache, are placed in different sockets of the same NUMA node, or in different nodes. The first interesting result is that NEMESIS performance depends a lot more on process placement (by a factor of 6) than KNEM (20%). This is related to NEMESIS double-copy using caches much more and thus depending greatly on caches being shared between cores. In the end, NEMESIS performs better (by about 50%) for all message sizes when processes share caches. However, processes in different sockets or different nodes obtain approximately twice the throughput with KNEM since much less memory copies occur between these distant sockets or nodes.

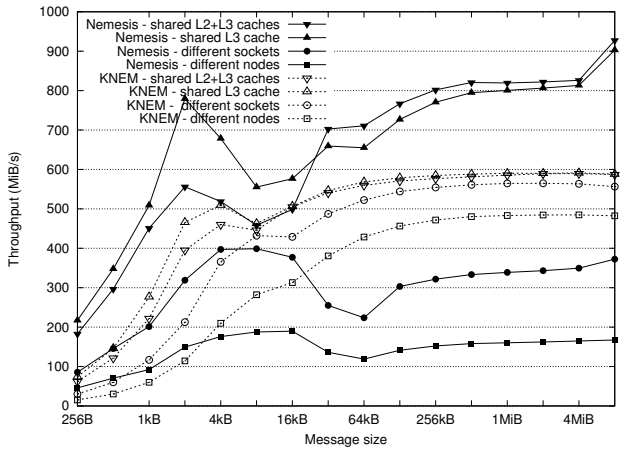

Figure 2. IMB Pingpong on Bertha96 (Offcache) depending on the process binding.

Bill8 and Idkonn24 show similar behaviors with obviously less impact from process placement since their hardware topology is less complex. I/OAT copy offload brings interesting performance improvement as shown from Figure 1. Hannibal8 however exhibits a different behavior: its processor-driven memory copy is much faster than I/OAT copy offload. We speculate that this is caused by I/OAT hardware development lagging behind overall processor development. This is especially apparent when comparing the *Nehalem* microarchitecture and its QPI memory interconnect to the INTEL *Core* microarchitecture (Hannibal8 versus Bill8/Idkonn24).

Moreover, while I/OAT performance does not depend on process location inside a NUMA node (since it does not involve any cache), it depends on the distance between the memory and the I/OAT device. Indeed, Hannibal8 has two I/OAT devices, one in each I/O hub near each NUMA node. Using an I/OAT device to copy memory buffers in the distant NUMA node decreases performance by about 40%. Furthermore, there is currently no easy way to choose the right I/OAT device when offloading copies on LINUX.

D. Thresholds

Previous sections show different behaviors depending on architecture and process placement. Nevertheless we may derive the following heuristics:

- NEMESIS is generally faster than KNEM for small messages even when no cache is shared.
- When no cache is shared between the processing cores, KNEM should be used for medium and large messages, starting at approximately 16 KiB.
- Bertha96 is the only machine where NEMESIS is interesting for large messages when a cache is shared. We suspect that this result is related to the IBM-specific chipset and cache-coherency protocol in this host.
- When available, I/OAT is usually interesting starting at about 16 KiB, except on the latest INTEL *Nehalem* architecture.

These results let us implement automatic switching between NEMESIS, KNEM and KNEM with I/OAT depending on process placement and on the underlying hardware. However, as discussed later in Section VI, such a simple point-to-point model may hardly work for real applications with collective operations or concurrent point-to-point communications.

IV. COLLECTIVE OPERATIONS

Collective operations are potentially a good target for our work since they involve many processes and complex communication patterns. The influence of cache polluting memory copies on these operations has been shown in [4]. Now that we have exhibited point-to-point behaviors, we evaluate how they apply to different collective patterns on our large shared-memory platforms.

A. All-to-One patterns: IMB Reduce

Figure 3 shows the aggregated throughput[3] between 8 processes doing an IMB Reduce collective operation on Hannibal8. NEMESIS is still faster than KNEM for small messages while KNEM becomes useful (+20%) for messages larger than 128 KiB. We actually observe the same behavior and threshold on different machines and with different numbers of processes. The Allreduce operation also shows comparable behavior with a slightly smaller threshold.

It has to be noted that I/OAT copy offload does not improve performance, even with large messages on machines where it was useful in the point-to-point benchmarks. Our feeling about it is that both I/OAT and processor-directed memory copies saturate the host memory bus when many processes transfer large messages. The aggregated performance is thus limited by the memory bus and not by the actual copy implementation. However, since the threshold does not seem to vary with the number of processes, we assume that no memory bus saturation occurs for medium messages (up to 1 MiB).

[3]Computed using the total amount of data transfered for each collective operation.

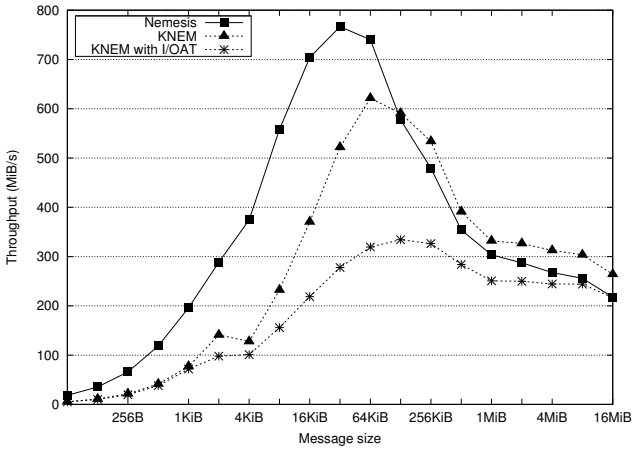

Figure 3. Aggregated throughput during a IMB Reduce with 8 processes on Hannibal8.

B. One-to-All patterns: IMB Scatter

Figure 4 shows the aggregated throughput between 16 processes doing a IMB Scatter collective operation on Idkonn24. As shown with Reduce in the previous section, NEMESIS appears better up to 128 KiB while KNEM is about 30% better for large messages. Again, this behavior and threshold do not vary much with the experimentation host. It is also worth noticing that other One-to-All patterns such as Broadcast show similar behaviors.

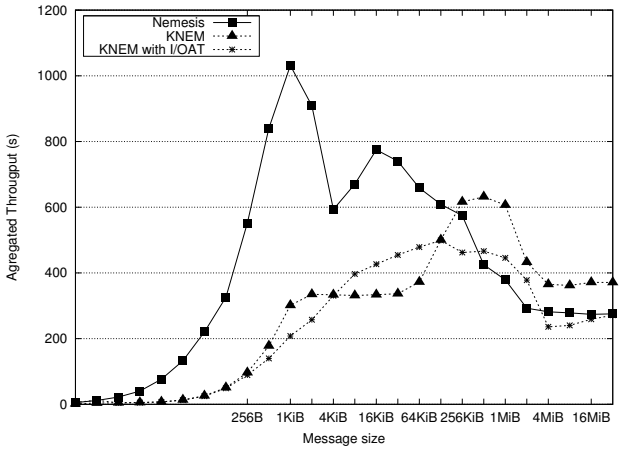

Figure 4. Aggregated throughput during a IMB Scatter with 16 processes on Idkonn24.

I/OAT still does not seem to help much, except on Bill8 where it brings about 10% throughput improvement for large messages. An explanation for this difference between Reduce and Scatter could be related to KNEM driving memory copies for the receiver side: since the same process performs

all receives during a Reduce, it may generate more memory contention than different processes performing receives during a Scatter.

C. All-to-All patterns: IMB Alltoall

All-to-all communication patterns are supposed to exhibit the biggest dependency on architecture topology and process placement because of their intense data transfers between all processes should cause memory contention and cache pollution [4]. Figures 5 and 6 present IMB Alltoall aggregated throughput on Bill8 and Bertha96 with respectively 8 and 64 processes.

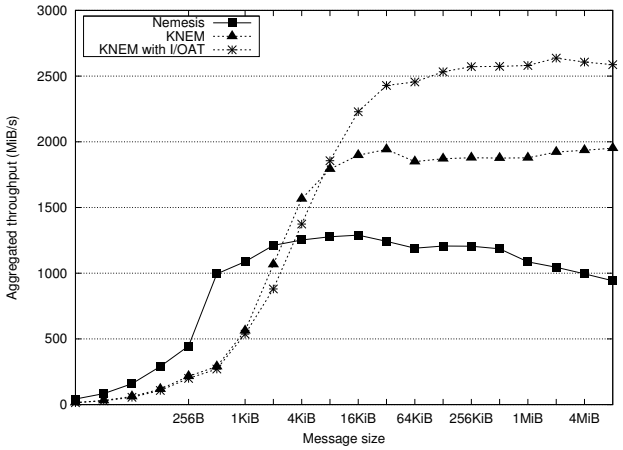

Figure 5. Alltoall on Bill8.

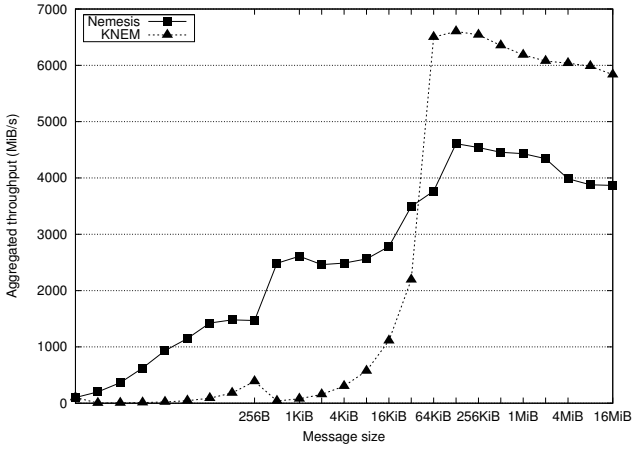

Figure 6. Alltoall on Bertha96 (64 processes).

KNEM always improves throughput on our experimental platforms for medium and large messages, from 50% on Bertha96 up to a factor of 2 on other hosts. This result shows that communication intensive patterns that involve a lot of contention benefit significantly from our single-copy kernel-assisted model. The fact that the improvement is smaller on Bertha96 suggests that the communication pattern saturates the host memory bus (64x63 messages are transmitted by this algorithm), causing the double-copy drawbacks to decrease performance substantially less.

Once again, I/OAT copy offload benefits on Bill8 since it brings another 30%, while it decreases the throughput on Hannibal8 and Idkonn24 by 25% and 60% respectively.

The dramatic KNEM performance increase on Bertha96 near 32 kiB is related to MPICH2 switching from a simple All-to-All algorithm (where 63 sends and receives are immediately posted) to a better organized one (where 8 pair-wise exchanges are consecutively performed). One reason for KNEM being so slow with the first algorithm (when using small messages) could be that each KNEM request has to find where to copy data from by looking up a descriptor within a list of 63 entries. In the second algorithm, each process communicates with a single other process during each pair-wise exchange so the list contains a single descriptor. Fortunately, this potential scalability issue would only be significant when many processes issue very small communication requests at the same time. The next release of KNEM (0.7) addresses this issue using hash tables.

This All-to-All pattern brings a new result: the threshold for switching from NEMESIS to KNEM appears inversely proportional to the number of processes. This confirms the fact that the more processes, the more contention, the more KNEM help thanks to reduced memory copies. However Bertha96 does not show the behavior, it could be related to much more contention occurring on this very large machine, but we do not fully understand this behavior yet.

Finally, it should be noted that we have observed somewhat similar behaviors with some other All-to-all patterns such as Allgather, but the KNEM improvement is slower and not as clear. It could be related to MPICH2 using complex algorithms with message splitting and/or aggregation that make performance harder to understand.

V. NAS Parallel Benchmarks

Previous works [3], [4] have shown that most of the NAS Parallel Benchmarks exhibit insignifiant performance dependencies on the communication strategy due to their small amount of communications. Only FT and IS (known to use large messages) get actual performance improvement thanks to KNEM. Table I summarizes the execution times of these interesting benchmarks on our experimental platforms.

As expected, using KNEM widely improves application performance on all machines by up to 38% for IS and 12% for FT with 64 processes running on Bertha96. Moreover, the improvement does not vary much with the class (the size of the problem). This indicates that using a kernel-assisted intra-node communication mechanism such as KNEM should help not only microbenchmarks but also real applications by reducing contention and cache pollution.

Machine	Benchmark	Nemesis	KNEM	Speedup
Hannibal8	ft.B.8	14.60 s	13.90 s	**+5%**
	ft.C.8	63.77 s	60.31 s	**+5.7%**
	is.B.8	0.70 s	0.60 s	**+16.6%**
	is.C.8	2.81 s	2.41 s	**+16.6%**
Bill8	ft.B.8	40.43 s	36.02 s	**+12.2%**
	ft.C.8	175.46 s	158.36 s	**+10.8%**
	is.B.8	2.42 s	1.89 s	**+12.2%**
	is.C.8	10.04 s	8.02 s	**+25.2%**
Idkonn24	ft.B.16	24.14 s	21.70 s	**+11.2%**
	ft.C.16	97.65 s	85.73 s	**+13.9%**
	is.B.16	1.29 s	0.99 s	**+30.3%**
	is.C.16	5.88 s	4.43 s	**+32.7%**
Bertha96	ft.C.64	31.91 s	28.82 s	**+10.7 %**
	ft.D.64	727.37 s	645.36 s	**+12.7 %**
	is.C.64	3.17 s	2.29 s	**+38.4 %**
	is.D.64	65.00 s	52.72 s	**+23.3 %**

Table I
NAS PARALLEL BENCHMARKS EXECUTION TIME.

Moreover we observed that adding I/OAT copy offload to KNEM does not increase performance on Idkonn24 and Hannibal8, while it offers little extra improvement on Bill8 (+6.1% for ft.C.8, +0% for is.C.8). This result confirms our observations from previous sections: I/OAT is mostly interesting for point-to-point operations with limited contention on old architectures (with slow processor-directed memory copies).

VI. DISCUSSION

The experiments presented in the previous sections clearly show that KNEM improves performance for medium and large messages, by 50% to 100% depending on the communication pattern. It also suffers less from process placement since fewer memory copies means less cache usage and thus fewer cacheline bounces as shown previously [4]. Our largest machine (Bertha96) shows that memory contention under heavy communication patterns limit KNEM improvements but this behavior is limited to All-to-all patterns. It does not prevent KNEM from significantly helping applications such as the NAS Parallel Benchmarks.

One important result of our study is that I/OAT copy offload only helps older INTEL architectures (Bill8 is 2 generations old), likely because the performance of I/OAT did not improve in the last years while the processor memory performance improved a lot (especially on *Nehalem*). Still, apart from preventing cache pollution, I/OAT also has the advantage of enabling the overlap of memory copies. KNEM already offers the corresponding asynchronous abilities [4] but MPI applications cannot benefit from it for complex communication patterns since non-blocking collectives [18] are not standardized yet.

As expected, NEMESIS user-space double-copy model offers the best small message latency while KNEM kernel-assisted single-copy is much faster for large messages most of the time. The threshold between these modes does not depend much on the architecture but it depends on the communication pattern, and even on the number of processes for All-to-all. Contrary to what we envisioned in [4], predicting these thresholds by looking at cache sizes does not look feasible anymore. The IMB *offcache* option revealed performance for individual operations in a more realistic way, but collective experiments suggest that contention between concurrent data transfers significantly impact their behavior. It prevents us from easily predicting optimal thresholds for random applications, and it leads us to think that auto-tuning could be an interesting way to tackle this problem.

VII. CONCLUSION AND FUTURE WORK

The widespread availability of multicore processors lead to the emergence of clusters with many cores, shared caches and NUMA nodes. Intra-node communication has become critical to the overall system performance, and kernel-assistance and copy offload are interesting solutions to improving this performance.

We presented an in-depth analysis of intra-node MPI communication with MPICH2 and our KNEM Linux kernel module[4]. Point-to-point, collective and application performance was evaluated on different multicore machines, using the last 3 generations of INTEL platforms and up to 96 cores in a single shared-memory node. Our results show that kernel-assistance in reducing the number of memory copies is indeed an interesting solution for messages bigger than about 50 KiB. It provides up to twice better throughput and 30% speedup on some NAS Parallel Benchmarks, even on large NUMA machines. KNEM performance also suffers less from process placement than the traditional double-copy model in user-space. However we revealed some complex behaviors for collective operations that could prevent us from easily dynamically choosing the right communication strategy. We also demonstrated that I/OAT copy offload currently only helps old hosts with poor memory performance, even if it might be interesting for overlap in upcoming MPI non-blocking collectives.

More multicore platforms, such as AMD hosts or upcoming *Nehalem-EX* machines are now being studied, without revealing new behaviors yet. Our primary focus is however to improve the KNEM interface to better help MPI implementations benefit from kernel-assistance. One way to do so is to extend the current KNEM interface (send/receive-oriented [4]) to deal with collective requirements such as multiple accesses to a single buffer. The receiver-directed data transfer will also be relaxed to better address All-to-one needs (see Section IV-B), and some possible scalability concerns will be looked at (see IV-C). This new KNEM interface will be used inside the both blocking and non-blocking collective engines of the MPI implementation so

[4]Available at http://runtime.bordeaux.inria.fr/knem/.

as to tightly integrate the algorithms with the decision of how to perform each actual data transfer.

ACKNOWLEDGMENTS

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

This work was carried out in the framework of the INRIA Associate Team program MPI-Runtime.

REFERENCES

[1] F. Cappello and D. Etiemble, "Mpi versus mpi+openmp on ibm sp for the nas benchmarks," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 2000, p. 12.

[2] D. Buntinas, G. Mercier, and W. Gropp, "Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem," *Parallel Computing, Selected Papers from EuroPVM/MPI 2006*, vol. 33, no. 9, pp. 634–644, Sep. 2007.

[3] L. Chai, P. Lai, H.-W. 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 *Proceedings of the IEEE International Conference on Parallel Processing (ICPP-2008)*. Portland, Oregon: IEEE Computer Society Press, Sep. 2008.

[4] D. Buntinas, B. Goglin, D. Goodell, G. Mercier, and S. Moreaud, "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, Sep. 2009.

[5] "OpenMP: The OpenMP API specification for parallel programming," http://openmp.org.

[6] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, Sep. 2004, pp. 97–104.

[7] Network-Based Computing Lab, The Ohio State University, "MVAPICH: MPI for InfiniBand over VAPI Layer," http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[8] D. Buntinas, G. Mercier, and W. Gropp, "Data Transfers between Processes in an SMP System: Performance Study and Application to MPI," *Parallel Processing, 2006. ICPP 2006. International Conference on*, pp. 487–496, Aug. 2006.

[9] B. Goglin, "High Throughput Intra-Node MPI Communication with Open-MX," in *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2009)*. Weimar, Germany: IEEE Computer Society Press, Feb. 2009. [Online]. Available: http://hal.inria.fr/inria-00331209

[10] R. Brightwell, T. Hudson, and K. Pedretti, "SMARTMAP: Operating System Support for Efficient Data Sharing Among Processes on a Multi-Core Processor," in *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing, SC 2008*. Austin, TX: ACM Press, Nov. 2008.

[11] M. Koop, W. Huang, K. Gopalakrishnan, and D. K. Panda, "Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand," in *16th IEEE Int'l Symposium on Hot Interconnects (HotI16)*, Palo Alto, CA, Aug. 2008.

[12] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard, Version 2.2," September 2009, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.

[13] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda, "Lightweight Kernel-Level Primitives for High-Performance MPI Intra-Node Communication over Multi-Core Systems," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'07)*, Austin, TX, Sep. 2007.

[14] A. Grover and C. Leech, "Accelerating Network Receive Processing (Intel I/O Acceleration Technology)," in *Proceedings of the Linux Symposium*, Ottawa, Canada, Jul. 2005, pp. 281–288.

[15] K. Vaidyanathan, L. Chai, W. Huang, and D. K. Panda, "Efficient Asynchronous Memory Copy Operations on Multi-Core Systems and I/OAT," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'07)*, Austin, TX, Sep. 2007.

[16] "Intel MPI Benchmarks," http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219847.htm.

[17] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks," *The International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, Fall 1991.

[18] T. Hoefler, A. Lumsdaine, and W. Rehm, "Implementation and performance analysis of non-blocking collective operations for MPI," in *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing, SC 2007*. Reno, NV: ACM Press, Nov. 2007.