Impact of Kernel-Assisted MPI Communication over Scientific Applications: CPMD and FFTW

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

Innovative Computing Laboratory,
EECS, University of Tennessee
{tma,bouteill,bosilca,dongarra}@eecs.utk.edu

Abstract. Collective communication is one of the most powerful message passing concepts, enabling parallel applications to express complex communication patterns while allowing the underlying MPI to provide efficient implementations to minimize the cost of the data movements. However, with the increase in the heterogeneity inside the nodes, more specifically the memory hierarchies, harnessing the maximum compute capabilities becomes increasingly difficult. This paper investigates the impact of kernel-assisted MPI communication, over two scientific applications: 1) Car-Parrinello molecular dynamics(CPMD), a chemical molecular dynamics application, and 2) FFTW, a Discrete Fourier Transform (DFT). By focusing on the usage of Message Passing Interface (MPI), we found the communication characteristics and patterns of each application. Our experiments indicate that the quality of the collective communication implementation on a specific machine plays a critical role on the overall application performance.

1 Introduction

Enhanced by multi-core and many-core nodes, clusters of workstations are widely used for scientific computing, where MPI has been the de facto programming paradigm for scientific computing parallel applications. To fully exploit the potential of multi-core nodes, domain users can adopt different programming models, prominently a pure MPI approach, but also hybrid programming (e.g. MPI+OpenMP or MPI+multithreading), virtual shared memory systems (e.g. OpenMP only), HPF (high performance FORTRAN) and etc. Compared with other approaches, the pure MPI approach has the benefit of the portability, allowing application developers to implement the code once, and then run it everywhere. However, the overhead of MPI intra-node communications from excessive memory copy is a major concern. In the pure MPI approach, each MPI process is bound to a core for performance reasons. The most common approach for delivering messages between MPI processes, running on shared memory multi-core nodes, has been to establish a shared memory between the two processes. The sender copies a message into the shared memory zone and the receiver copies it out to the target buffer, resulting in a double copy for each point-to-point communication. This copy-in/copy-out approach wastes, not only CPU cycles,

Y. Cotronis et al. (Eds.): EuroMPI 2011, LNCS 6960, pp. 247–254, 2011.

but also memory bandwidth, especially for large messages and collective communication. With more cores and deeper memory hierarchies, it becomes more difficult, for a shared memory approach, to deliver optimal performance in a generic way.

Kernel-assisted memory copy can alleviate this issue by using system calls to offload the copy to the kernel. Because the kernel has a physical view of the memory space for both processes (the source and the destination), it can perform memory copies directly from the source memory to destination memory without an intermediate buffer. KNEM is a Linux kernel module that enables high-performance, inter-process, single-copy memory copies. It offers support for asynchronous and vector data transfers. MPI communities have realized the importance of integrating kernel assisted memory copy to MPI intra-node communications. Open MPI, since version 1.5, includes KNEM support in its shared memory point-to-point communications component. MPICH2, since version 1.1.1, uses KNEM in the DMA LMT to improve large message performance within a single node. The work in [1,2] has shown that KNEM-enabled MPI communication significantly improves the performance of some micro- and macro-benchmarks. However, the performance of real scientific applications using KNEM-enabled MPI communication has yet to be asserted. This paper focuses on the impact of KNEM-enabled MPI communication on scientific applications.

We selected two applications: CPMD [3] and FFTW [4]. These applications come from different scientific areas: from molecular dynamics to signal processing. They are widely known and used in the engineering and scientific computing communities. The CPMD code is a parallel plane wave/pseudo-potential implementation of density functional theory, particularly designed for ab-initio molecular dynamics [3]. FFTW, "Fastest Fourier Transform in the West", is one of the most popular libraries for computing discrete Fourier transforms (DFTs), developed by Matteo Frigo and Steven G. Johnson [4]. Both applications are developed around a core, involving both point-to-point and collective communications, and can be considered as lightly communication-intensive applications. Using a lightweight MPI profiling software (mpiP), we investigate the communication overhead distribution in each application, including the percentage of MPI runtime in the application runtime, the percentage of each MPI call runtime in the whole communication time, and the message size distribution.

The remainder of this paper is organized as follows: Section 2 introduces the related work about the shared memory and the kernel assisted approach. Section 3 depicts the two parallel applications used in this paper, followed by Section 4 where the experimental results of CPMD and FFTW are presented.

2 Related Work

With the increasing complexity of the node architecture, shared memory performance remains a critical corner-stone in MPI application performance. As such, significant efforts have been deployed to improve the MPI intra-node communication performance. Darius Buntinas *et al.* proposed single-copy communication to speed up large message point-to-point communication for MPICH2

(based on vmsplice and KNEM [1]). The KNEM assisted approach outperforms the standard transfer method in the MPICH2 implementation when no cache is shared between the processing cores, or when very large messages are being transferred. Even simply using KNEM assisted point-to-point communication underneath collective communication achieved a significant improvement [1,2]. Within Open MPI, a similar approach was implemented, with further emphasis on auto-tuning and performance portability [5]. KNEM based memory operation (with features such as persistent memory registration and copy direction control) has been leveraged from within the collective algorithm itself, allowing for most copies to happen in parallel in 'rooted' communications (e.g. one-to-all or all-to-one). Furthermore, the hardware features and the collective communication topology are mapped in order to minimize the volume of data transiting between distant memory hierarchies. Overall these improvements demonstrated substantial speedups of the communication operations on multicore systems [6]. However, the evaluation of the benefits of these approaches has been mostly centered around synthetic benchmarks.

From another perspective, several works have focused on determining the properties of parallel applications in the context of hierarchical systems [7,8]. Recent works have investigated the benefits of hybrid programming in the context of multicore nodes [9]. However, the conclusions of these studies are challenged by the performance now permitted by kernel assisted copies within MPI. Our present work focuses specifically on investigating the behavior of prominent application, taking into account this novelty.

3 Applications

Car-Parrinello Molecular Dynamics(CPMD) is a plane wave/pseudo-potential implementation of density functional theory, particularly designed for ab-initio molecular dynamic [3]. CPMD simulations use the most fundamental approaches to model condensed phases. Dynamic equations of motion are solved for the ions with the inter-ionic forces computed from the valence electron density, which is solved for at each time step using density functional theory. In the case of methane, a CPMD simulation consists of one C and four H ions and eight valence electrons per molecules. The ground state electron density is computed at each time step. And polarization and other short range forces are also taken into account in the CPMD [3]. CPMD provides several standard simulations, such as C-120, Si-64, water-32, etc. We selected the methan-fd-nosymm test in our experiments, that uses the finite-difference (FD) method, based on a discretization of the differential operator [10], without molecular symmetry.

FFTW, "Fastest Fourier Transform in the West", is one of the most popular libraries to compute discrete Fourier transforms (DFTs). FFTW can handle inputs with one or more dimensions, arbitrary size, and both real and complex data [4]. FFTW also features a MPI-based distributed implementation. To compute the FFT of a multi-dimensional array, each processor first transforms all the dimensions of the data that are completely local to it (rows). Then, the processors perform a transpose of the data in order to get the remaining dimension

local to a processor (columns). This dimension is then Fourier transformed, and then the dataset is transposed, back to its original order.

4 Experiments

4.1 Experimental Conditions

Our experimental platform (named IG) is a 48 core AMD NUMA machine with 128GB of memory. The system is composed of 8 sockets with a six-core 2.8 GHz AMD Opteron 8439 SE, 5 MB L3 caches and 16 GB memory per NUMA node. The sockets are further divided as two sets of 4 sockets on two separate boards connected by a low performance interlink.

The Linux Red Hat 4.1.2 (2.6.35.7 kernel) operating system is used on the machine, with the KNEM (version 0.9.5) kernel memory copy module. The MPI implementation is Open MPI (trunk r24549), with mpiP (version 3.2.1) [11] to profile and record MPI usage. Inside Open MPI, two different setups are compared: the SM setup uses the tuned collective module [12] and the SM point-to-point Byte Transfer Layer (BTL); the KNEM setup uses the KNEM collective module and the SM/KNEM BTL [6] and is hence benefiting from kernel assisted memory copies for messages larger than 4KB. Because KNEM collective module implemented a subset of MPI collective operations: Broadcast, Gather(v), Scatter(v), Allgather(v), and Alltoall(v), operations not implemented in KNEM collective module such as Reduce, Allreduce, and etc. will use Open MPI's Basic collective module. The mapping between physical cores and MPI processes is identical for both setups, regardless of the underlying communication components.

The CPMD software (version 3.13.2) [3] is configured as 'LINUXMPI' with the BLAS/LAPACK libraries (LAPACK 3.3.0). We selected one simulation from CPMD's vibrational analysis tests: methan-fd-nosymm.inp. FFTW-3.2.alpha [4] is configured with MPI support. Our input for the FFT mpi-bench is [1500×1500 20 20] with the verification(-y), which stands for a 1500×1500 complex DFT, a 20 complex DFT, and a 20 complex DFT.

4.2 CPMD

Table 1 compares the execution time breakdown, into compute time and communication time, of the CPMD application on a large multicore node, between the two communication modes (KNEM and SM, differing in their use of kernel assisted memory copies). As expected, the computation execution time remains generally constant when changing the communication mode (201s versus 194s). The major performance difference between the two setups lies in the communication overhead (MPI time), which occupies 26.9% of the overall application runtime for the KNEM-enabled mode, while it rises to 54.1% when using the regular SM communication mode. The CPMD application makes extensive use of all-to-all collective communications, which enjoy a threefold speedup when using the KNEM-enabled approach, translating into a 1.5× application speedup in the methan-fd-nosymm test case.

Table 1. Total application time and MPI time for CPMD's methan-fd-nosymm test between Open MPI's KNEM mode and SM mode, with 48 MPI processes on IG's 48 cores

	Total Application time(sec)	MPI time(sec)
KNEM-enabled	276	74.4
SM-enabled	423	229

Table 2. Sum of all processes' execution time for the 5 most used MPI functions in CPMD's methan-fd-nosymm using shared memory and KNEM (48 processes on IG's 48 cores)

KNEM-enabled			SM-enabled		
Call	Time(millisec)	MPI%	Call	Time(millisec)	MPI%
Alltoall	2.88e+06	90.32	Alltoall	1.02e+07	97.71
Bcast	1.15e+05	3.59	Bcast	1.08e+05	1.03
Allreduce	1.12e+05	3.52	Allreduce	1.05e+05	1.00
Barrier	7.01e+04	2.20	Allreduce	8.6e+03	0.08
Allreduce	7.49e+03	0.23	Recv	6.17e+03	0.06

Table 2 presents the accumulated time, over all processes, spent in the five most time consuming MPI functions. In both cases, using KNEM or SM, the AlltoAll operation takes more than 90% of the MPI execution time. However, compared with SM-based communication, the KNEM version reduces the Alltoall cost from $10,000$s to about $2,900$s. Based on the statistics gathered using mpiP, the average message size for each AlltoAll operation is 24KBytes, a size in the range where KNEM is beneficial to collective operation performance (bigger than 4KB). Here, Allreduce in the KNEM-enabled setup is worse than in the SM-enabled setup(1.12e5 vs 1.05e5), because allreduce in the KNEM setup actually triggers Open MPI's Basic collective module, which is a simple and basic implementation of collective operations without any optimization. It's not a surprise for a Tuned collective operation to outperform a Basic collective operation.

Figure 1 shows a strong scaling performance of CPMD's methan-fd-nosymm tests, using the KNEM-enabled and SM communication modes. In this experiment, the process i is bound to core i, for all modes. The KNEM-enabled MPI communication outperforms the SM MPI communications, regardless of processes in use and the number of NUMA nodes. The CPMD application benefits from a better scalability, when increasing the number of cores, with the KNEM-enabled MPI operations. The SM communications do not permit the application to scale to more than 24 processes (the limit where all processors are on the same system board, in this machine), because the SM communications are oblivious to the underlying hardware topology. On the other hand, the KNEM-enabled operations enable the application to benefit from the expected scalability for the CPMD application, even though the NUMA topology is extremely challenging on this platform.

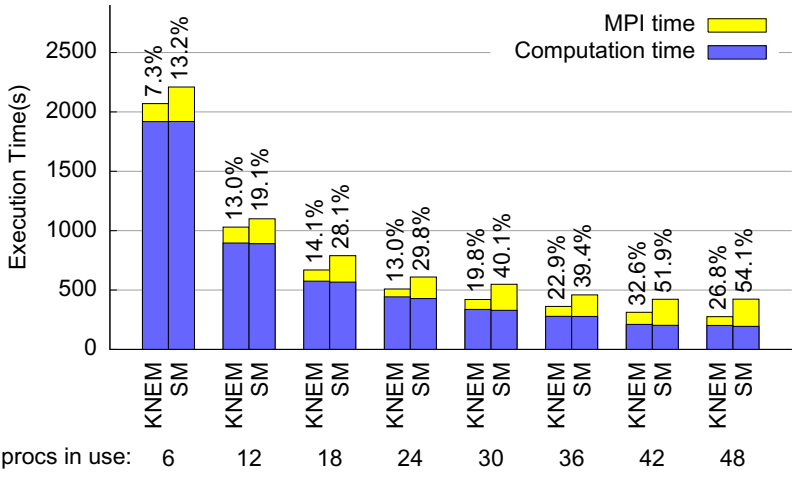

Fig. 1. Strong Scaling for CPMD's methan-fd-nosymm test over KNEM and shared memory. Processes are bound to IG's cores in a compact way (rank i is bound to core i).

4.3 FFTW

The next application is the Discrete Fourier Transform library, from FFTW. A detailed view of the contribution to the cumulative communication time of the 5 most used MPI functions of this application is presented in the table 3. One can notice that the most time consuming MPI call (the broadcast communication) enjoys nearly an order of magnitude improvement when using the KNEM collective component (from 6070s down to 959s). Even point-to-point communications see their performance improve, but less significantly. As an example, the sendrecv based on KNEM is 1.37× faster than the one based on shared memory. Similarly with the previous application, the average message size (see table 4) is larger than the message size where KNEM enabled communications become beneficial. For the broadcast collective, the average size is 14MB, and the average size for the sendrecv point-to-point communication is 16KB. The execution time of KNEM Scatterv is a little more than Tuned Scatterv here, because messages in

Table 3. Cumulative time of the five most used MPI calls in a 48 processes FFTW (1500x1500, 20, 20, with verification), running with KNEM or SM components, on the 48 cores of IG

KNEM-enabled Mode			SM-enabled Mode		
Call	Time (millisec)	MPI%	Call	Time (millisec)	MPI%
Bcast	9.59e+05	61.98	Bcast	6.07e+06	88.9
Sendrecv	3.52e+05	22.78	Sendrecv	4.82e+05	7.07
Scatterv	1.65e+05	10.69	Gatherv	1.39e+05	2.03
Gatherv	5.86e+04	3.79	Scatterv	1.33e+05	1.95
Comm_dup	4.33e+03	0.28	Comm_dup	1.81e+03	0.03

Table 4. Aggregate Sent Message Size for each MPI calls in FFTW. 48 processes on IG, one rank per core.

Call	Count	Total (bytes)	Avrg (bytes)	Sent%
Bcast	19488	2.8e+11	1.44e+07	94.20
Sendrecv	710888	1.14e+10	1.61e+04	3.84
Gatherv	19488	5.83e+09	2.99e+05	1.96
Bcast	288	1.47e+05	512	0.00
Bcast	288	1.47e+05	512	0.00

the FFTW's scatterv operation is smaller than KBytes and the overhead of trapping into kernel and distributing cookies offset the benefits of KNEM kernel copy.

Finally, the table 5 presents the total execution time, and the MPI contribution to that total, for both KNEM-enabled and SM communication modes. Thanks to the benefits on the broadcast operations, the KNEM-enabled communications induce a dramatic threefold reduction of the time spent communicating, from 143s to 38s. As FFTW is a communication intensive application, the decrease of the communication contribution to the execution time translates into a major improvement of the overall execution time (doubled performance). The last row of Table 5 indicates the performance of the OpenMP version of this same application. Although this version cannot run on distributed memory clusters, it is indicative of the performance attained by a tailored approach on this shared memory architecture. The introduction of KNEM-enabled communications have greatly reduced the efficiency gap between the OpenMP approach and the MPI approach on shared memory machines. However, the OpenMP code is still twice as fast; but it lacks the capability to span over multicore cluster.

Table 5. Total application time for the FFTW's application when using OpenMP(48 threads) or pure MPI over different communications: the KNEM-enabled or the shared memory-enabled communication with 48 processes (rank i bound to core i).

	Total Application time(sec)	MPI time(sec)
KNEM-enabled mode	107	38.1
SM-enabled mode	216	143
OpenMP mode	49.5	N/A

5 Conclusion

A lot of MPI users spend significant time porting their pure MPI applications to a hybrid model (usually MPI+OpenMP) to exploit the full potential of multicore architectures. Inefficient shared memory-based communications are an important factor forcing domain users to look for alternative solutions inside the nodes. From the experiments presented in the previous Section, classical shared memory communications have certain difficulties to provide good point-to-point and collective performance, when the number of cores and consequently the complexity of the memory hierarchy increase. However, kernel-assisted single-copy

approaches have the potential to alleviate this issue by offloading the memory copies into the kernel, reducing the number as well as their impact on the memory bus by a factor of two. Experiments show the KNEM-enabled MPI communication can increase application performance, and expose a better scalability than the classical shared memory approach when integrating more resources inside computing nodes. With more cores, increased core heterogeneity, and deeper memory hierarchies, kernel assisted MPI communication provides dependable performance, and offers a better alternative to hybrid approaches, while still retaining the simplicity of a single programming model.

References

1. 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), pp. 462–469. IEEE Computer Society Press, Vienna (2009)
2. Moreaud, S., Goglin, B., Goodell, D., Namyst, R.: Optimizing MPI Communication within large Multicore nodes with Kernel assistance. In: CAC 2010: The 10th Workshop on Communication Architecture for Clusters, Held in Conjunction with IPDPS 2010. IEEE Computer Society Press, Atlanta (2010)
3. Hutter, J., Iannuzzi, M.: CPMD: parrinello Molecular Dynamics, `http://www.cpmd.org/`
4. Frigo, M., Johnson, S.: The Design and Implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005)
5. Ma, T., Bosilca, G., Bouteiller, A., Dongarra, J.J.: Locality and topology aware intra-node communication among multicore CPUs. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS, vol. 6305, pp. 265–274. Springer, Heidelberg (2010)
6. Ma, T., Bosilca, G., Bouteiller, A., Goglin, B., Squyres, J., Dongarra, J.: Kernel Assisted Collective Intra-node Communication Among Multicore and Manycore CPUs. Research report (2010)
7. Vetter, J.S., Mueller, F.: Communication characteristics of large-scale scientific applications for contemporary cluster architectures. J. Parallel Distrib. Comput. 63, 853–865 (2003)
8. Plaat, A., Bal, H.E., Hofman, R.F.H., Kielmann, T.: Sensitivity of parallel applications to large differences in bandwidth and latency in two-layer interconnects. Future Generation Computer Systems 17(6), 769–782 (2001)
9. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 427–436. IEEE Computer Society Press, Washington, DC, USA (2009)
10. scholarpedia.org: Finite difference method, `http://www.scholarpedia.org/article/Finite_difference_method`)
11. Vetter, J.S.: mpiP: Lightweight, Scalable MPI Profiling, `http://mpip.sourceforge.net/`
12. Fagg, G.E., Bosilca, G., Pješivac-Grbović, J., Angskun, T., Dongarra, J.: Tuned: A flexible high performance collective communication component developed for Open MPI. In: Proccedings of DAPSYS 2006, pp. 65–72. Springer, Innsbruck (2006)