

Impact of Network Sharing in Multi-core Architectures*

G. Narayanaswamy
Dept. of Computer Science
Virginia Tech
cnganesh@cs.vt.edu

P. Balaji
Mathematics and Computer Science
Argonne National Laboratory
balaji@mcs.anl.gov

W. Feng
Dept. of Computer Science
Virginia Tech
feng@cs.vt.edu

Abstract

As commodity components continue to dominate the realm of high-end computing, two hardware trends have emerged as major contributors—high-speed networking technologies and multi-core architectures. Communication middleware such as the Message Passing Interface (MPI) uses the network technology for communicating between processes that reside on different physical nodes, while using shared memory for communicating between processes on different cores within the same node. Thus, two conflicting possibilities arise: (i) with the advent of multi-core architectures, the number of processes that reside on the same physical node and hence share the same physical network can potentially increase significantly, resulting in *increased* network usage, and (ii) given the increase in intra-node shared-memory communication for processes residing on the same node, the network usage can potentially decrease significantly.

In this paper, we address these two conflicting possibilities and study the behavior of network usage in multi-core environments with sample scientific applications. Specifically, we analyze trends that result in increase or decrease of network usage, and we derive insights into application performance based on these. We also study the sharing of different resources in the system in multi-core environments and identify the contribution of the network in this mix. In addition, we study different process allocation strategies and analyze their impact on such network sharing.

1 Introduction

High-end computing (HEC) systems are increasingly being characterized by nodes built out of commodity components. Two of the significant trends in the HEC domain have been the dramatic improvements in networking technology (using high-speed network accelerators) and in processor technology (with the advent of multi-core architectures). With respect to the networks, several technologies are available in the market, including 10-Gigabit Ethernet [5–7], Myrinet [10], and InfiniBand (IB) [8]. With respect to multi-core processors, quad-core processors from Intel and AMD are considered commodity today. Processors with higher number of cores (e.g., Intel Xscale) and multithreading within each core (e.g., SUN Niagara) are also becoming available. As these two trends emerge, it is becoming increasingly important to analyze their interaction.

Scientists typically use standard parallel programming models to develop their applications over HEC systems in a portable manner. The Message Passing Interface (MPI) is the *de facto* standard in such programming models and is used by a vast majority of scientific applications. With the growing importance of

multi-core environments, most implementations of MPI are optimized on such environments by using the network technology for communicating between processes that reside on different physical nodes, while using shared memory for communicating between processes on different cores within the same node. Using shared memory within the node typically reduces the network overhead, resulting in higher performance. Based on such a design for MPI implementations, two conflicting possibilities arise: (i) with the advent of multi-core architectures, the number of processes that reside on the same physical node and hence share the same physical network can potentially increase significantly resulting in *increased* network usage and (ii) given the increase in intra-node shared-memory communication for processes residing on the same node, network usage can potentially *decrease* significantly.

Based on these two conflicting possibilities, it is not clear whether modern multi-core architectures add extra requirements on networks, thus requiring future HEC systems to scale up network capacity further, or whether the increase in intra-node shared memory communication compensates for the increase in network sharing, thus not requiring any changes. Thus, depending on the application communication pattern and the layout of processes across nodes, interesting questions about network sharing and scalability need to be studied.

In this paper, we address these two conflicting possibilities and study the behavior of network usage in multi-core environments with sample scientific applications within the NAS parallel benchmark suite. Specifically, we analyze trends that result in increase or decrease of network usage and derive insights into application performance based on these. We also study the sharing of different resources in the system in multi-core environments and identify the contribution of the network in this mix. Further, we study different process allocation strategies and analyze their impact on such network sharing. Our experimental results demonstrate that for some applications multi-core architectures can significantly hamper performance because of the increased network sharing, while for others the performance can stay constant or even improve because of the better intra-node communication.

The rest of the paper is organized as follows. Section 2 presents some background on multi-core architectures and Myrinet. Section 3 explains some of the networking issues in multi-core architectures that are of interest to us. Our experimental evaluation is presented in section 4. In Section 5 we briefly discuss related work and conclude in section 6.

2 Background

In this section, we present an overview of multi-core architectures and the Myri-10G Myrinet network.

*This work was supported in part by the National Science Foundation Grant #0702182, the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract and the Department of Computer Science at Virginia Tech.

2.1 Overview of Multi-core Architectures

For many years, hardware manufacturers have been replicating components on processors to create multiple pathways allowing more than one instruction to run concurrently with others. Duplicate arithmetic and floating-point units, coprocessing units, and multiple thread contexts on the same processing die are examples of such replication. Multi-core processors are considered to be the next step in such hardware replication, where two or more (mostly) independent execution units are combined onto the same integrated circuit.

Multi-core architectures are at a high level similar to multi-processor architectures. The operating system deals with multiple cores in the same way as multiple processors, by allocating one process to each core at a time. Arbitration of shared resources between the cores happens completely in hardware, with no intervention from the OS. However, multi-core processors also differ significantly from multi-processor systems. For example, in multi-core processors, both computation units are integrated on the same die. Thus, communication between these computation units does not have to go outside the die and hence is independent of the die pin overhead. Further, architectures such as the current Intel multi-cores, as shown in Figure 1, provide a shared cache between the different cores on the same die. This makes communication even simpler by eliminating the need for complicated cache-coherency protocols.

Figure 1: Intel dual-core dual-processor system

However, multi-core processors also have the disadvantage of more shared resources as compared to multi-processor systems. That is, multi-core processors might require different cores on a processor die to block waiting for local shared resources to get freed when it is being used by a different core. Such contention is even higher when the ratio of the number of cores on the system increases as compared to the other resources (e.g., multi-core systems with multiple thread contexts). Further, for architectures such as AMD NUMA, each processor in a multi-processor system has access to its own memory, and hence overall memory bandwidth essentially doubles with the number of processors. For multi-core systems however, the overall memory bandwidth does not change.

2.2 Overview of Myrinet Network

Myri-10G [10], the latest generation Myrinet developed by Myricom, is a low-latency wormhole routing based high-speed interconnect and supporting data transfers at the rate of 10 Gbps. The Myrinet network interface card (NIC) has a user-

programmable processor and DMA engines that eases the design and customization of software communication stacks. MX (Myrinet Express) is a high-performance, low-level, message-passing software interface tailored for Myrinet. The Myri-10G NICs, switches, and associated software support both Ethernet (MXoE) and Myrinet (MXoM) protocols at the link level. The basic MX-10G communication primitives are non-blocking send and receive operations. Our network consists of the Myri-10G NICs connected by a 24-port Myrinet switch. The NICs are connected to the host via a 133 MHz/64 bit PCI-X bus. They have a programmable LANai processor running at 300 MHz with 2 MB on-board SRAM memory.

3 Networking Issues in Multi-cores

In this section, we cover some of the challenges faced in multi-core environments with respect to networking.

3.1 Sharing of Network Resources

One of the important questions when designing high-end systems based on commodity components is network scalability, specifically, whether the network can cope up with the CPU in terms of the network data being sent. An important advantage of multi-core architectures is the ability to multiplex network data streams over a single network hardware medium, which potentially helps in better use of network resources. Also, latency between application processes can decrease as more and more traffic goes over intra-node communication media instead of over the network. This is good for commodity applications but may affect performance of scientific applications because of sharing of network resources. Similarly, sharing of processor resources can be both beneficial and harmful. For example, shared caches in multi-core architectures can reduce latencies between processes to the scale of nanoseconds, but at the same time introduce contention for those resources.

3.2 Process Allocation Schemes

In a multi-core cluster, the processes can be arranged among the nodes in several ways. Applications typically have fixed communication patterns, and allocation schemes provide us the flexibility of modifying which processes get colocated on the same node. Thus, depending on the allocation scheme, the amount of network sharing might increase or decrease. We look at two common allocation schemes in this paper: cyclic and blocked allocation.

Cyclic allocation allocates each subsequent process cyclically to the next node in the ring of nodes. For example, with a total of 16 processes and 8 nodes, process ranks 0 and 8 will get assigned to node 0, ranks 1 and 9 to node 1, and so on. This allocation ensures good load balance among all nodes. In blocked allocation, blocks of processes are assigned to each node in turn. For example, with 16 processes, 8 nodes and a block size of 2, process ranks 0 and 1 get assigned to node 0, ranks 2 and 3 to node 1, and so on.

The process allocation scheme can play an important role in the kind of communication performed by a process. For example, for an application that does mostly neighbor communication in a 1-D chain of processes, blocked allocation will probably turn out to be better. The reason is that the neighbor processes that

Figure 2: Evaluation of network sharing: (a) 16X1 vs 8X2 co-processor and (b) 8X2 virtual processor vs 4X4

a process communicates with are more likely to be on the same node. The result can be significant reduction in network communication, thereby potentially improving performance. With more cores on a node, the situation doesn't improve further, however, since there are only a constant number of neighbors.

In a 2-D grid of $N \times N$ processes performing neighbor communication with M cores in a node, again blocked allocation works better than cyclic allocation in localizing more neighbors when $N > M$. When M and N are equal, the same number of neighbors coexist with both cyclic and blocked allocation. The same holds true for a 3-D grid of processes as well. Thus, for neighbor communication, there are higher chances that more neighbors will co-exist with blocked allocation.

As another example, for an application which performs tree-like regular long distance communication, a cyclic allocation strategy might be a better choice, as it might localize many of the communicating processes within a node. For applications running on large clusters with hierarchical layers of switches, allocation schemes that localize branches of trees within the lowest hierarchy might be more beneficial.

4 Performance Evaluation

In this section, we present our performance evaluation results of the NAS Parallel benchmark suite. We follow two different evaluation methodologies. In Section 4.3, we analyze the impact of network and processor sharing in the performance of applications. In Section 4.4, we show results with different process allocation schemes. We show results with class B of the NAS benchmarks, but we note that we got similar results for classes A and C.

4.1 Experimental Setup

Each node in our 16-node cluster setup is a custom-built, dual-processor, dual-core AMD Opteron 2.55 GHz system having 4 GB of DDR2 667 MHz SDRAM. The four cores in each system are organized as cores 0 and 1 on processor 0 and cores 2 and 3 on processor 1. Each core has a separate 1 MB L2 cache. All machines run Ubuntu Fiesty with kernel version 2.6.19 and are equipped with Myri-10G network interface cards connected to a Myrinet switch. The MPI library used is MPICH2-MX v1.0.6. All experiments were run at least three times with the processor affinity of each process set to a fixed core to remove the impact of operating system scheduling anomalies.

4.2 Configurations Used in Experiments

This section describes the configurations on which we ran our experiments. We use 16 processes for all the NPB benchmarks because this covers the maximum number of benchmarks and configurations for our setup. We note that 16 processes can be run on different configurations on a multi-core architecture with four cores. Picking only those with constant number of processes on a node, we end up with three configurations:

- 16X1 – 16 nodes, one process on one of the four cores
- 8X2 – 8 nodes, 2 processes, on two of the four cores
- 4X4 – 4 nodes, 4 processes, one on each core

We start by observing that between each of the three configurations there are increased levels of network sharing. With 16X1, there is no network sharing since each node runs only one application process. With 8X2, however, two processes in each node use the same network interface card. Hence there is two times more network sharing than with the 16X1 case. With 4X4, four processes use the same NIC, thus making the network sharing four times greater than with the 16X1 case. In our experiments, we ran the 4X4 configuration with cyclic allocation of processes between nodes.

To consider the effects of processor sharing, we split the 8X2 into two cases again. Our setup consists of a dual-core dual-processor system and hence the two processes can be run in two different modes:

- 8X2 co-processor mode: two processes, each running on a different processor
- 8X2 virtual processor mode: two processes, both run on the same processor

In the virtual processor mode, there is increased sharing of processor resources because both processes are run on the same processor.

4.3 Impact of Network Sharing

We start by evaluating the impact of network sharing by running the various NPB benchmarks over each of the three configurations described above. Figure 2 shows the impact network resource sharing can have on the performance. As shown in Figure 2(a), as we move from 16X1 to 8X2 co-processor mode, the performance of all the benchmarks drops (as much as 27% for IS). The reason is the increased network sharing in the 8X2 configuration, where two processes have to share the same network device. Since only one process has been added to every

Figure 3: Network communication time: (a) 16X1 vs 8X2 co-processor and (b) 8X2 virtual processor vs 4X4

node, the chances that a process will communicate predominantly with the process colocated in its node are slim.

In Figure 2(b) on the other hand, the performance drop is seen mainly for CG, FT, and IS, while the other benchmarks perform similarly or show improved performance (in the case of MG) between the two configurations. Here we see mixed benefit in moving to 4X4 because more number of processes are colocated in the same node. Thus, potentially, more shared memory communication can happen reducing the possibility of network sharing.

To analyze the level of network sharing in the above results, we profile the network communication time in each of these configurations. Since we are using Myrinet’s MX protocol, we profile the time spent in the `mx_isend()` and `mx_test()` calls. This time represents the time spent by the network in sending the data out and thus is an indicator of the overhead of network sharing. Figure 3 shows the normalized total time spent in `mx_isend()` and `mx_test()` calls for the various configurations. As seen in Figure 3(a), there is an increase in the network communication time for all the benchmarks between 16X1 and 8X2 co-processor mode. In other words, moving to the 8X2 co-processor mode results in more time being spent for network communication because the network resources are being shared. Also, the amount of intra-node communication remains comparatively low, so it is difficult to observe any significant benefit from the reduced latency. Of 15 other possible processes with which a process can communicate, only one results in intra-node communication. Thus, there is a 93% chance that a process will communicate over the network with another process. These results mimic the performance results where all benchmarks observe a decrease in performance when moving to 8X2 co-processor mode.

In Figure 3(b), however, the network communication increases only for the CG, FT, and IS benchmarks, while for all others it drops. This again clearly mimics the performance results as seen in Figure 2(b). In this case, moving from the 8X2 virtual processor mode to 4X4 mode results in two processes getting added to the same node. Thus there is increased capability to perform intra-node communication. Compared to a 93% chance of network communication with the 8X2 case, there is only 80% chance with the 4X4 case that a process will communicate over the network with another process. We analyze our results further by profiling the amount of data sent over the network as compared to intra-node communication for all the

benchmarks. The data-size analysis shows a similar trend as the network communication time and corroborates the performance results we get. For dearth of space, we present those results in [11].

To make our analysis of network sharing more comprehensive, we also need to analyze the effect of processor sharing. To do this, we compare the performances of 8X2 co-processor and 8X2 virtual processor modes. For the co-processor mode, we run the processes in cores 0 and 2, while for the virtual processor mode we run the processes on cores 2 and 3.

Figure 4(a) shows the performance of co-processor and virtual processor modes in the 8X2 configuration for all the benchmarks. We observe a substantial performance difference between the two modes for all the benchmarks (up to 53% as in the case of SP). This shows that sharing of processor resources can be very detrimental for the application.

We verify our results with processor sharing by using PAPI to count various hardware performance counters. We first measure the number of L2 cache misses. As shown in Figure 4(b), the virtual processor mode sees increased L2 cache misses ranging from 27% more misses in the case of FT up to 48% more in the case of MG.

We profile the benchmarks for two types of CPU stall cycles as well: those stalling for any resource and those stalling for memory accesses. Here we show results only for the CG and SP benchmarks; the results for the other benchmarks are similar. Figure 5 shows the normalized number of CPU stall cycles waiting for resource and memory for CG and SP benchmarks. From the graphs, we can see that the virtual processor mode has more resource stalls than does the co-processor mode. SP observes up to 73% more resource stalls cycles and 66% more memory stalls, whereas in the case of CG, it is 14% and 17%, respectively.

4.4 Analysis of Allocation Schemes

In this section, we take a different approach for investigating network sharing impacts, by performing a comparative study of using the cyclic and blocked allocation schemes with the NPB benchmarks. We run the experiments on 64 processes, with four processes on each of the 16 nodes. Figure 6(a) shows the performance of various NPB benchmarks with cyclic and blocked allocation on class B data sizes. The results show that the CG benchmark sees an improvement in performance (17%)

Figure 4: Analysis of processor sharing: (a) performance and (b) L2 cache misses

Figure 5: CPU stall cycles

while for the other benchmarks, performance remains the same or drops.

To further understand the reasons behind the trends observed, we profile the network communication time of the benchmarks similar to the profiling done in section 4.3. Figure 6(b) shows the normalized total communication time for each of the benchmarks for cyclic and blocked cases. From the graph, we observe that CG observes a substantial reduction in communication time when running in blocked allocation mode. For all other benchmarks, there is an increase in network communication time. We note here that MG observes more than a six fold increase in communication time, which explains why the performance of MG drops heavily when using blocked allocation. The data size analysis shows that the amount of data communicated over the network for CG halves when moving from cyclic to blocked allocation, while MG sees a slight increase. This also verifies the performance results that we observe. Refer to [11] for the data size analysis results.

4.5 Application Processing Pattern Analysis

The previous sections evaluated application performance from the viewpoint of system and network characteristics. In this section, we tie in the analysis developed in previous sections to the application communication patterns.

The CG benchmark performs communication within groups of four processes with certain boundary nodes communicating between the groups. As an example, Figure 7 shows the communication pattern of CG with 16 processes. This pattern clearly

shows that any allocation scheme that localizes the groups of four processes within a node will have good performance improvement. For example, if each of the group of four processes are localized within a node, the only network communication is between the boundary nodes. Thus any allocation scheme that optimizes this strategy will get better performance. We see this result with blocked allocation in the 16X4 case, which performs better than the cyclic allocation (see Figure 6).

The FT benchmark performs an all-to-all exchange within subcommunicators along the row and column in a processor grid. Thus, having more cores in a node allows some processes in either the row or the column subcommunicators to be local to a node. But the communication as part of the other subcommunicator still has to go through the network. Although some amount of network communication is saved, there is still sufficient sharing of network resources. Similarly, choosing an appropriate allocation scheme might help in localizing all the nodes of a sub-communicator, but still there is enough network traffic between the other subcommunicator to nullify this advantage. In our results, we see a similar behavior, where the performance drops for FT when moving from 16X1 to 4X4 because of the increased sharing of the network but remains the same for the cyclic and blocked allocation strategies. The IS benchmark has a similar analysis as FT as it also does predominantly all-to-all exchanges. This analysis for FT and IS ties in well with the network data size analysis results shown in [11]. Designing efficient network topologies for FT and IS can be a challenging task given the all-to-all pattern.

MG has an interesting pattern wherein there is some clustered communication in groups of 4, but these clusters themselves are grouped in clusters of 16. Each process communicates with another process which is at increasing distances of increasing powers of two from it. Thus, any process allocation strategy that puts processes at distances of powers of two on the same node will be beneficial for the application. For example, when the number of nodes is a power of two, cyclic allocation will put such processes on the same node. This situation explains why MG performs better with cyclic allocation than blocked allocation with 64 processes and also why the 4X4 cyclic configuration performs better than the 8X2 configuration.

BT, LU, and SP follow complex communication patterns that make analysis from the processing pattern difficult. Changes in configurations or allocation schemes may not significantly

Figure 6: Cyclic vs blocked: (a) performance and (b) network communication time

affect the amount of network sharing. For example, our results in previous sections don't seem to follow any major trends.

In summary, we saw in Section 4.3 that network sharing does affect the performance of applications, although the results might pale in comparison with the effects of processor sharing. Nevertheless, network sharing is an important concern that has to be addressed. We also saw that using a different process allocation strategy has the potential to reduce the effects of network sharing. Furthermore, knowledge of the application pattern can give better ideas for designing better run-time configurations for applications.

Figure 7: CG pattern

5 Related Work

A lot of work has been proposed on optimizing application performance on multi-core architectures. In [3], Curtis-Maury et al. look at OpenMP communication on multi-core processors. Chai et al., in [2], look at the performance of applications based on the amount of intra-CMP, inter-CMP and inter-node communication performed. We investigate the problem with a different approach by looking at the amount of sharing of network resources. In [1], Alam et al. perform extensive characterization of various scientific workloads on the AMD multi-core processor. But their work looks only at a single multi-core node, whereas we look at a cluster of nodes and at the impact of the network as well.

Similarly, many articles and papers have investigated the communication patterns of various applications and benchmarks [4, 9, 12, 13]. But none of these papers focus on multi-core architectures in their evaluation, which we address here.

6 Conclusions

With the advent of multi-core architectures, designers of high-end systems are faced with the challenge of ensuring that the interconnection network scales well with more processing cores. We analyze this problem by studying the impact of network sharing on multi-core architectures. Our results indicate that

network sharing can have a significant impact on performance, although sharing of processor resources has a much bigger impact. With a good understanding of the application communication pattern, a different process allocation strategy could potentially reduce the effects of network sharing. For future work, the network sharing analysis studied in this paper can be incorporated into MPI process managers (such as `mpd`, `lamboot` etc) which can lay out processes more intelligently across nodes.

References

- [1] 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. In *IISWC*, pages 225–236, 2006.
- [2] 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 *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 471–478, 2007.
- [3] M. Curtis-Maury, X. Ding, C. D. Antonopoulos, and D. S. Nikolopoulos. An evaluation of openmp on current and emerging multi-threaded/multicore processors. In *First International Workshop on OpenMP*, Eugene, Oregon, June 2005.
- [4] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *20th Annual International Symposium on Computer Architecture*, pages 2–13, May 1993.
- [5] D. Dalessandro, P. Wyckoff, and G. Montry. Initial performance evaluation of the neteffect 10 gigabit iwarp adapter. In *RAIT '06*, 2006.
- [6] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance characterization of a 10-gigabit ethernet toe. In *IEEE HotI*, Palo Alto, CA, 2005.
- [7] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-gigabit ethernet for networks of workstations, clusters and grids: A case study. In *SC '03*, 2003.
- [8] InfiniBand Trade Association. <http://www.infinibandta.org/>.
- [9] J. Kim and D. J. Lilja. Characterization of communication patterns in message-passing parallel scientific application programs. In *CANPC '98: Proceedings of the Second International Workshop on Network-Based Parallel Computing*, pages 202–216, London, UK, 1998. Springer-Verlag.
- [10] Myricom. Myrinet home page. <http://www.myri.com/>.
- [11] G. Narayanaswamy, P. Balaji, and W. Feng. Impact of network sharing in multicore architectures. Technical report, Computer Science department, Virginia Tech, March 2008.
- [12] R. Riesen. Communication patterns. In *Workshop on Communication Architecture for Clusters (CSC 2006)*, Rhode Island, Greece, April 2006.
- [13] J. S. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J. Parallel Distrib. Comput.*, 63(9):853–865, 2003.