SAND2009-3205C

Catamount N-Way Performance on XT5

Ron Brightwell, Suzanne Kelly, Jeff Crow
Scable System Software Department
Sandia National Laboratories
Albuquerque, New Mexico 81785–1319
{rbbrigh,smkelly,jecrow}@sandia.gov

Trammell Hudson
Operating Systems Research
1527 16th NW #5
Washington, DC 20036
hudson@osresearch.net

This paper provides a performance evalution of the Catamount N-Way (CNW) operating system on a dual-socket quad-core XT5 platform. CNW provides several operating system-level enhancements for multicore processors, including the SMARTMAP technology for single-copy MPI messages and the ability to easily choose between 4 KB and 2 MB memory pages. Our evaluation will include an analysis of the performance of important micro-benchmarks and applications.

I. Introduction

Catamount is a third-generation lightweight compute node operating system [1] developed by Sandia National Laboratories along with Cray, Inc., as part of the Red Storm [2] project. Red Storm was a collaborative development between Sandia and Cray that has resulted in the commercially successful Cray XT series of massively parallel computers.

The Catamount lightweight kernel differs in many ways from a traditional general-purpose operating system. One important difference is memory management. Unlike general-purpose operating systems that support demand paging, Catamount's memory model is significantly less complex and allows for several key optimizations for distributed memory parallel computing applications. One such feature, SMARTMAP [3], allows MPI processes running on a multi-core processor to directly read and write each others' memory. This capability has been shown to provide several significant performance improvements for intra-node data movement on dual- and quad-core processors.

Recently, we have completed an initial port of Catamount to the Cray XT5 system, where a compute node has dual sockets and each node contains a quad-core AMD Opteron processor. We present results from several communication micro-benchmarks that measure both the performance of point-to-point and collective operations.

The rest of this paper is organized as follows. The next section provides a brief overview of Catamount. Section III provides performance results for several micro-benchmarks, followed by a summary of relevant results and futre work in Section IV.

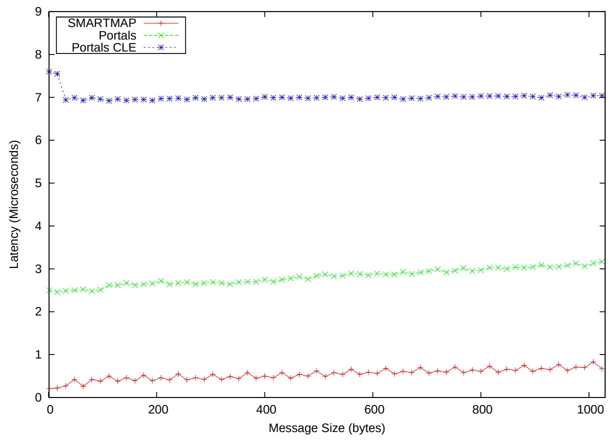
II. Catamount

Catamount [1] is a third-generation compute node operating system developed by Sandia National Laboratories with Cray, Inc., as part of the Red Storm project [2]. Red Storm is the prototype for what has become the commercially successful Cray XT line of massively parallel processing systems. Catamount has several unique features that are designed to optimize performance and scalability specifically for a distributed memory message passing-based parallel computing platform.

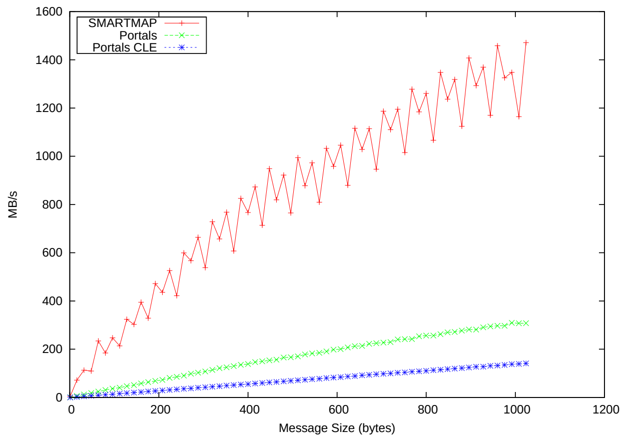
One such important feature is memory management. Unlike traditional full-featured operating systems, Catamount does not support demand-paged virtual memory and uses a linear mapping from virtual addresses to physical pages of memory. This approach can potentially have several advantages. For instance, there is no need to register memory or "lock" memory pages involved in network transfers to prevent the operating system from unmapping or remapping pages. The mapping in Catamount is done at process creation time and is never changed during the life of a process.

SMARTMAP takes advantage of Catamount's simple memory management model, specifically the fact that Catamount only uses a single entry in the top-level page table mapping structure (PML4) on each X86-64 (AMD Opteron or Intel EM64T) core. Each PML4 slot covers 39 bits of address space, or 512 GB of memory. Normally,

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



(a) Latency



(b) Bandwidth

Fig. 1: SHMEM Put Performance

Catamount only uses the first entry covering physical addresses in the range `0x0` to `0x007FFFFFFFFF`. The X86-64 architecture supports a 48-bit address space, so there are 512 entries in the PML4.

Each core writes the pointer to its PML4 table into an array at core 0 when a new parallel job is started. Each time the kernel enters the routine to run the user-level process, it copies all of the PML4 entries from each core into the local core. This allows every core on a node to see every other core's view of the virtual memory across the node, at a fixed offset into its own virtual address space.

Another feature of Catamount is that the mapping of virtual addresses for the same executable image is identical across all of the processes on all of the nodes. The starting address of the data, stack, and heap is the same. This means that the virtual address of a variable with global scope is the same in every process. A "local" virtual addresses can be converted to a "remote" virtual address by simply flipping a few bits at the upper part of the address. This makes it extremely easy for one process to read and write the corresponding data in another process's address space running on a different core of the same processor.

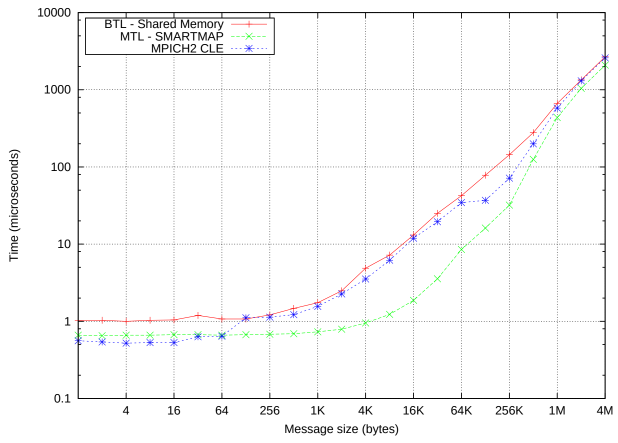
Catamount's memory management design is much simpler than a general-purpose OS like Linux. Linux memory management is based on the principle that processes execute in different address spaces and threads execute in the same address space. Most architecture ports, X86-64 included, maintain a unique set of address translation structures (e.g., a page table tree on X86-64) for each process and a single set for each group of threads. Our mapping strategy operates differently in that a process's address space and associated translation structures are neither fully-unique or fully-shared. For example, our map on the X86-64 architecture maintains a unique top-level page table (the PML4) for each process; however, all processes share a common set of leaves linked from this top-level table. Linux memory management does not support this form of page-table sharing, so each process must be given a replicated copy of each shareable leaf. This results in more memory being wasted on page tables (2 MB per GB of address space on X86-64) and a larger cache footprint than necessary. Modifications to Linux to support sharing a single page table entry for shared memory mapped regions have been proposed, but the changes have not been accepted in the mainline kernel.

III. Results

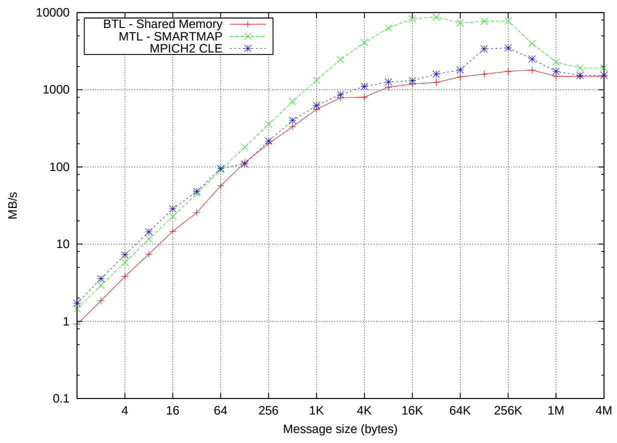
In addition to the dual-socket quad-core node running Catamount, we also include performance results from a 2.1 GHz single-socket AMD Opteron running Cray's Compute Node Linux Environment (CLE) and production MPICH2 implementation that uses shared memory for intra-node transfers.

A. *SHMEM Performance*

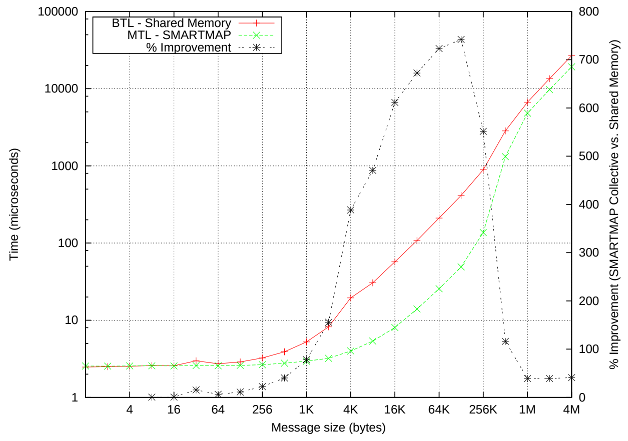
Figure 1 compares the ping-pong latency and bandwidth performance for a Cray SHMEM put operation. Single-byte latency for the SMARTMAP implementation is 210 ns, while the Portals latency is 2.5 μs. Curiously, the



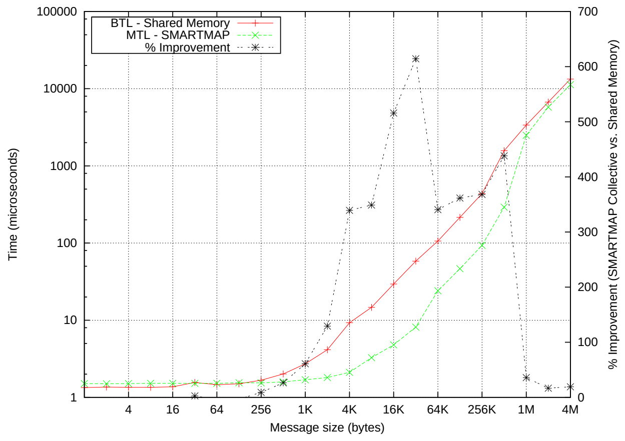
(a) MPI PingPong Latency



(b) MPI PingPong Bandwidth



(c) MPI Exchange



(d) MPI Sendrecv

Fig. 2: IMB MPI Point-to-Point Results

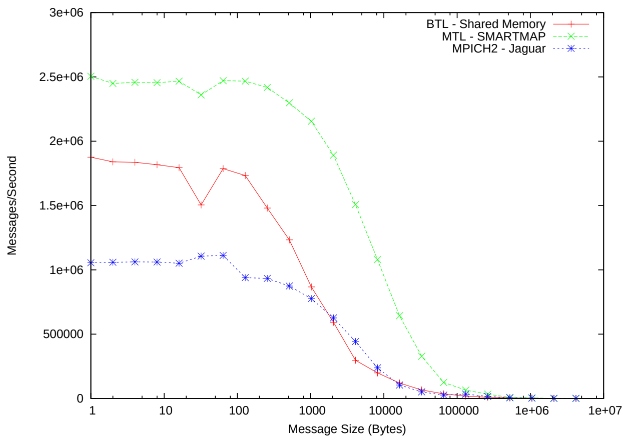
performance of CLE/MPICH2 is significantly worse at 7.6 μs. As for bandwidth, SMARTMAP is able to achieve a peak bandwidth of nearly 1500 MB/s, while the two Portals-based transports only achieve a little more than 300 MB/s.

B. MPI Point-to-Point

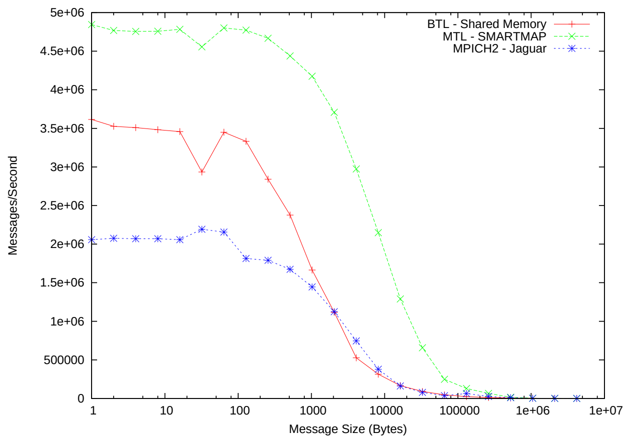
Figure 2 shows the performance of MPI peer communication performance using the MPI implementation describe in [4]. MPI ping-pong latency for a 0-byte message is a little more than 600 ns. The CLE/MPICH2 implementation for the Cray CLE environment is slightly better at just over 500 ns, but these results were run on a slightly slower processor. The Open MPI implementation has not gone through the extensive performance optimizations that the production MPI from Cray has. Using POSIX-style shared memory emulated by SMARTMAP is slightly slower, but still achieves a ping-pong latency less than a microsecond.

As with latency, ping-pong bandwidth performance for the Cray CLE implementation is slightly better than SMARTMAP out to 64 bytes, at which point the SMARTMAP implementation begins to win. The single-copy advantage of SMARTMAP allows it to maintain a performance advantage for messages less than about 2 MB, at which point all implementations most likely achieve full memory bandwidth.

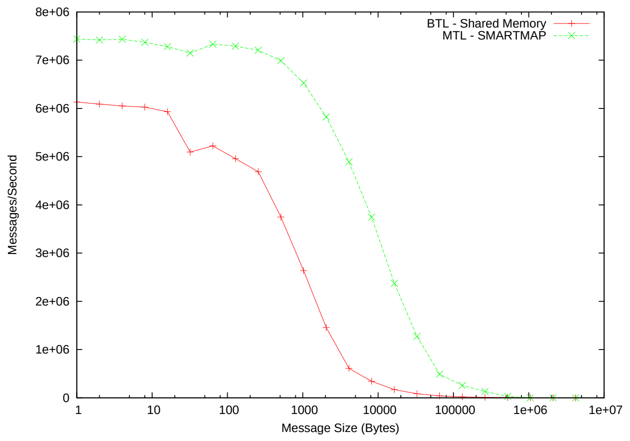
The other two benchmarks in Figure 2, Sendrecv and Exchange, show the performance of 8 processes exchanging point-to-point messages using slightly different MPI peer communication functions. Each graph shows the percentage



(a) One pair



(b) Two pairs



(c) Four pairs

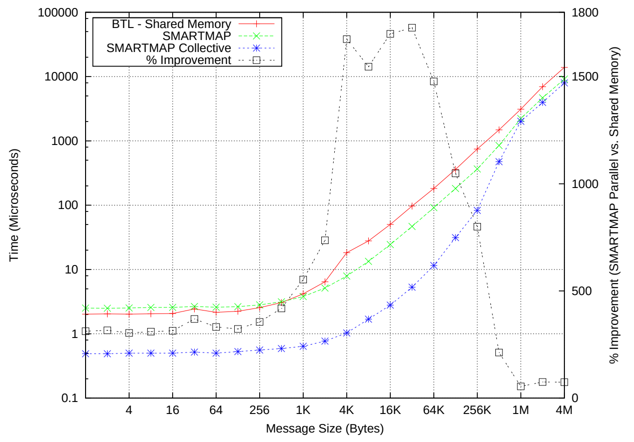
Fig. 3: MPI Message Rate

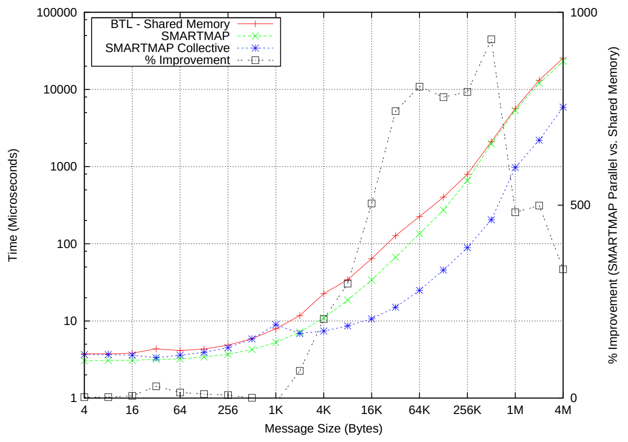
improvement of SMARTMAP relative to shared memory. For Sendrecv, SMARTMAP is able to achieve more than a factor of six improvement for 32 KB messages, and SMARTMAP achieves more than a seven times improvement at 128 KB messages for the Exchange benchmark.

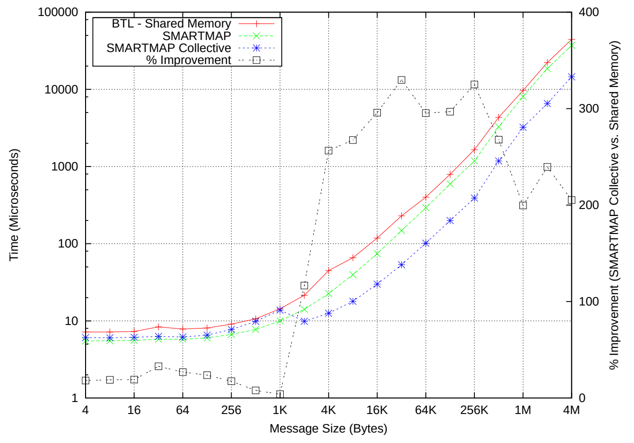
Figure 3 shows message rate performance for one, two, and four pairs of processes. The first two graphs also show the performance of the Cray CLE MPICH implementation, which has the worst performance of the three. The SMARTMAP implementation is able to achieve more than twice the message rate of the Cray implementation. More investigation is needed to understand why this implementation performs so poorly. The message rate essentially stays constant between one and two pairs of communicating processes. In contrast, the SMARTMAP implementation continues to increase in performance, achieving nearly 7.5 million messages per second for 4 pairs of processes.

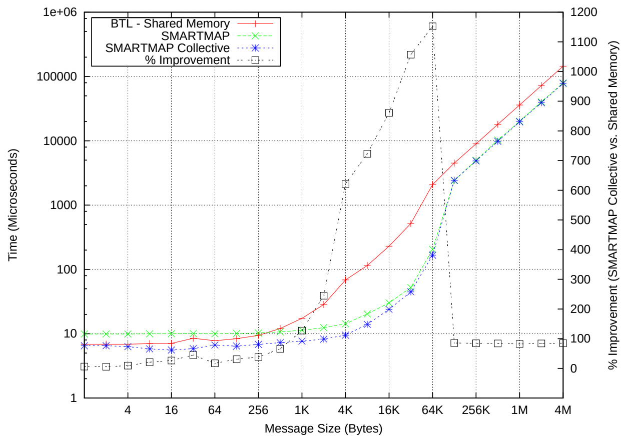
C. *MPI Collectives*

Figure 4 shows the performance of several MPI collective communication operations for eight processes on a single node. For the Broadcast benchmark, SMARTMAP is able to achieve nearly 18 times better performance over the shared memory implementation. The advantage is not nearly as pronounced for the Reduce, where SMARTMAP only achieves nearly nine times the performance. For the Allreduce benchmark, SMARTMAP is able to achieve a respectable factor of three improvement, but for Alltoall, SMARTMAP outperforms shared memory by nearly 12 times. Finally, SMARTMAP also significantly outperforms the others for the Barrier benchmark.


(a) MPI Broadcast


(b) MPI Reduce


(c) MPI Allreduce


(d) MPI Alltoall

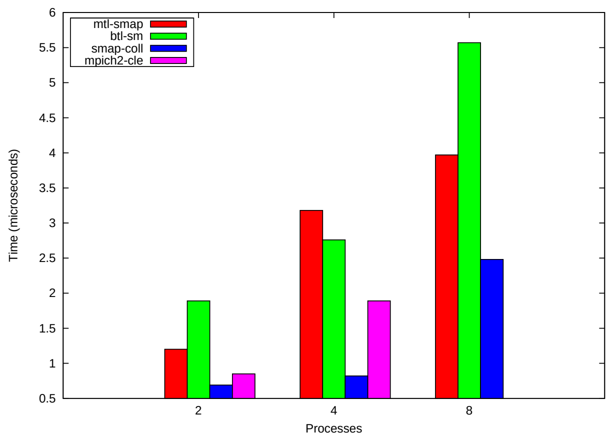

(e) MPI Barrier

Fig. 4: IMB MPI Collective Performance

IV. Summary

This paper provides an initial set of performance results for several communication micro-benchmarks using the Catamount N-Way lightweight kernel on a dual-socket quad-core node of a Cray XT5 system. Catamount provides a feature called SMARTMAP that allows for single-copy data transfers between the processes on a multi-core processor. SMARTMAP preserves the independent address space of a process, but also allows parallel processes to behave as threads in a node-wide global address space. The advantage of SMARTMAP for Cray SHMEM and MPI communication continues to improve as the number of cores on a node increase.

These performance results were from an initial port of the Catamount N-Way kernel. There is much left work to be done for a production environment. Currently Catamount does not take into account the fact that each socket has its own local memory. Catamount assumes the cost of accessing memory is uniform. In the future, we plan to make the initial allocation and assignment of local memory socket-aware. We would also like to perform an in-depth analysis of application performance on dual-socket nodes.

References

[1] S. M. Kelly and R. Brightwell, “Software architecture of the light weight kernel, Catamount,” in *Proceedings of the 2005 Cray User Group Annual Technical Conference*, May 2005.

[2] W. J. Camp and J. L. Tomkins, “Thor’s hammer: The first version of the Red Storm MPP architecture,” in *In Proceedings of the SC 2002 Conference on High Performance Networking and Computing*, Baltimore, MD, November 2002.

[3] 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 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC’08)*, November 2008.

[4] R. Brightwell, “A prototype implementation of MPI for SMARTMAP,” in *Proceedings of the 15th European PVM/MPI Users’ Group Conference*, September 2008.