Achieving Computational I/O Efficiency in a High Performance Cluster Using Multicore Processors[*]

Li Ou, Xin Chen, Xubin (Ben) He
Department of Electrical and Computer Engineering
Tennessee Technological University, Cookeville, TN 38505, USA
{lou21, xchen21, hexb}@tntech.edu

Christian Engelmann, Stephen L. Scott
Computer Science and Mathematics Division
Oak Ridge National Laboratory,Oak Ridge, TN 37831, USA
{engelmannc,scottsl}@ornl.gov

Abstract

Cluster computing has become one of the most popular platforms for high-performance computing today. The recent popularity of multicore processors provides a flexible way to increase the computational capability of clusters. Although the system performance may improve with multicore processors in a cluster, I/O requests initiated by multiple cores may saturate the I/O bus, and furthermore increase the latency by issuing multiple non-contiguous disk accesses. In this paper, we propose an asymmetric collective I/O for multicore processors to improve multiple non-contiguous accesses. In our configuration, one core in each multicore processor is designated as the coordinator, and others serve as computing cores. The coordinator is responsible for aggregating I/O operations from computing cores and submitting a contiguous request. The coordinator allocates contiguous memory buffers on behalf of other cores to avoid redundant data copies.

1 Introduction

Cluster computing [6] has become one of the most popular platforms for high-performance computing today, because of its high performance-cost ratio. In high-performance computing (HPC) clusters, standard message-passing systems, such as Message Passing Interface (MPI) or Parallel Virtual Machine (PVM), are widely used to achieve parallelism in applications. A complex scientific computation can be decompositioned into multiple smaller parallel tasks, and each task is computed by a MPI process. Generally, the ideal case is that the number of parallel processes spawned for the computation is equal to the number of physical processors in the cluster, therefore parallel tasks are executed faster in a cluster with more processors.

Traditionally, we scale a cluster by increasing the number of computing nodes, or by adapting the symmetric multi-processing (SMP) architecture for each node. The recent popularity of multicore processors provides a flexible solution to increase the computational capability of clusters. Parallel applications can benefit from multicore processors [4, 7], because each core is a physical processor, and multiplying the number of processors simply multiplies the number of processes spawned for the parallel tasks, and allowing such tasks to be executed faster. Meanwhile, utilization of the processors increases with multicore proces-

[*] This research was partially sponsored by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. It was also partially sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory,which is managed by UT-Battelle, LLC under Contract No. DEAC05-00OR22725. The work performed at Tennessee Tech University was partially supported by the U.S. National Science Foundation under Grants No. OCI-0453438 and CNS-0617528.

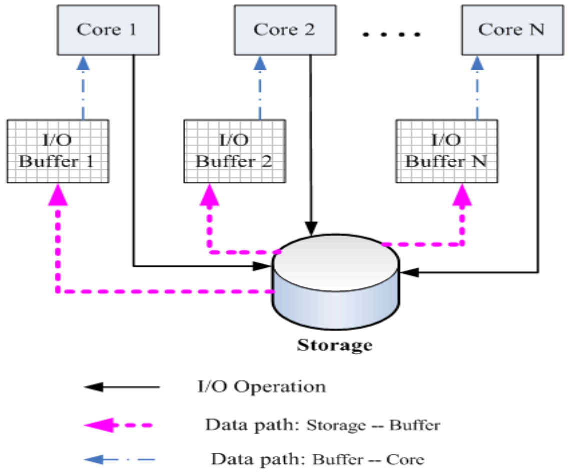

Figure 1. Simple parallel I/O in multicore processors.

sors.

Although system performance may improve by applying multicore processors in a cluster, issuing simultaneous processes may introduce overhead. Previous research [2] shows that for parallel applications which are sensitive to cache size or require intensive communications [5], the system may suffer from performance degradation after enabling multiple logical processors. Multicore processors experience the same problems because multiple cores within one die share the same L2 cache, memory, and I/O channels, and thus exacerbate the resource contentions. First, multicores of the same physical CPU compete for the same L2 cache, which potentially generates more cache-miss, and thus stalls processors more frequently. Second, memory access speed is limited by the shared memory bus, and multiple processes running on the same die may increase memory contention. Finally, more I/O requests may saturate the bus, and furthermore, increase the latency by issuing multiple non-contiguous disk accesses.

The current design, multicores are configured symmetrically from the perspective of computational capacity. In a typical parallel application, each process is responsible to process one part of the whole dataset: It reads the correspondent data from the parallel file system, processes data locally, then writes the data back to the file. Multiple symmetrical cores submit separate I/O operations independently with their own pre-allocated buffers (Figure 1). This results in a large number of I/O operations, each of which is often for a very small amount of data. This approach typically performs poorly for parallel applications because of the overhead of multiple operations and non-contiguous disk accesses. Multiple non-contiguous disk accesses may be aggregated into contiguous accesses with collective I/O, such as the interfaces provided by MPI-IO, but there is the cost of inter-nodes communication and in-memory permutation. It is inefficient, because most of the time, assignment of the dataset to a process does not consider the location of the processors: do the multiple processors physically reside in the same node?

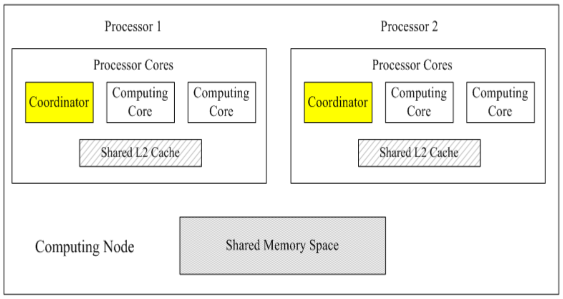

Figure 2. Architecture of asymmetric computation for multicore processors.

2 Asymmetric Collective I/O for Multicore Processors

We propose a new multicore paradigm called *asymmetric computation* (Figure 2). In our configuration, one core in each multicore processor is designated as the coordinator, and others serve as computing cores. The efforts of I/O operations from computing nodes are coordinated by the coordinator with *asymmetric collective I/O*. In asymmetric collective I/O, the coordinator aggregates multiple I/O requests from computing cores to one contiguous request with a large buffer and sends it to the storage at one time (Figure 3). The computing cores do not really commit I/O requests to storage. They inform the coordinator with their I/O operation parameters. After gathering I/O information from each computing core, the coordinator tries to

Computing Core	Coordinator

char *read (file, size) {
inform coordinator with (file,size);
Barrier; /* wait for all cores*/
wait message from coordinator;
return buffer address;
}
	char *read (file, size) {
	Barrier; /* wait for all core*/
	Aggregate I/O operation;
	Allocate a contiguous buffer;
	send I/O read;
	assign buffer to each core;
	wake up each core with buffer address;
	return buffer address;
	}

Table 1. Asymmetric collective I/O operations of computing core and coordinator.

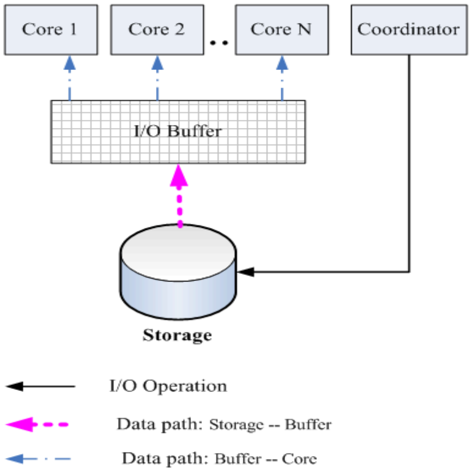

Figure 3. Asymmetric collective parallel I/O for multicore processors.

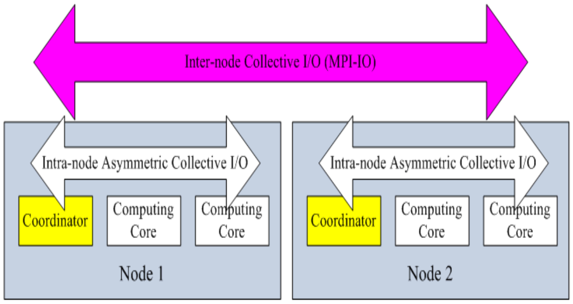
Inter-node Collective I/O (MPI-IO)
Intra-node Asymmetric Collective I/O
Coordinator
Computing Core
Computing Core
Node 1
Intra-node Asymmetric Collective I/O
Coordinator
Computing Core
Computing Core
Node 2

Figure 4. Hierarchy collective I/O for multiple nodes with multicore processors.

combine multiple I/O operations into one contiguous access with data sieving [3]. The coordinator sends the new requests on behalf of all cores. One of the important aspects of asymmetric collective I/O is that the computing cores do not need to allocate buffers for their I/O system calls. After aggregating the I/O operations, the coordinator allocates one buffer for each contiguous request. Once I/O operations complete, the coordinator informs the computing cores of the addresses of buffers containing data, they requested. This design avoids additional memory copies of the buffers allocated by the computing cores and buffers the coordinator uses to submit I/O operations. Table 1 gives an example of how file read operation is processed by the computing cores and the coordinator.

The asymmetric collective I/O may be utilized as intra-node optimizations for a hierarchical inter-node collective I/O (Figure 4). If a node is configured with multiple multicore processors, the coordinators from each processor negotiate with each other, aggregate I/O requests across processors, and chose one coordinator to send requests, on behalf of all cores of the node. The I/O operations may be further optimized by using collective I/O of MPI-IO, if multiple nodes with multicore processors are involved. One coordinator from each node first aggregates I/O requests of multiple cores within the node, then the coordinators representing each node use MPI-IO to further aggregate I/O operations, and permute data among nodes. After the completion of inter-node collective I/O, the coordinator distributes data to multiple cores by assigning them a correspondent buffer address.

In most parallel applications, each process is designated to process one part of an entire dataset during the initialization phase. I/O performance may be further improved by assigning contiguous dataset to the processors belonging to the same node, instead of waiting for the moment when I/O requests are issued to convert non-contiguous disk accesses to contiguous accesses with collective I/O. Multicore processors in the same node process adjoint data. Therefore, when multiple cores issue I/O requests at the same time, a simple synchronous mechanism is able to combine the adjoint datasets belonging to different cores into a larger contiguous I/O access for sending datasets to a disk server. At the initialization phase, a contiguous dataset is assigned to a node, and inside the node, the coordinator eventually decomposes the dataset and assigns the subset to each core.

3 Implementation and Evaluation

We are implementing the asymmetric collective I/O and integrating it into MPI-IO packages (Figure 5) reusing the collective IO system call interface provided by MPI-IO. Our add-in component distinguishes between intra-node and inter-node collective I/O and optimizes I/O performance for multicore processors. Once the proof-of-concept prototype is complete, we plan to use a parallel I/O benchmark, such as NASA BTIO benchmarks [1], to evaluate our design under various configurations of multiple nodes with multicore processors.

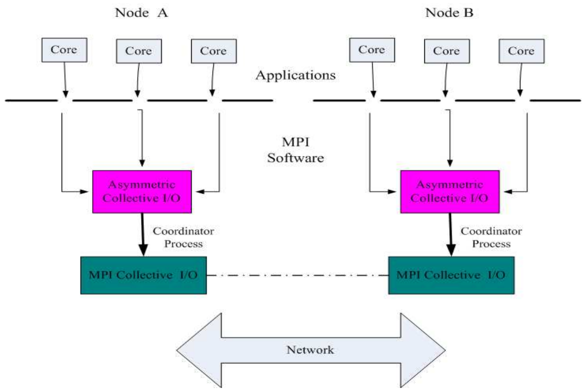

Figure 5. Software architecture of asymmetric collective I/O.

4 Conclusions and Future Work

In this paper, we propose an asymmetric collective I/O for multicore processors to improve multiple noncontiguous accesses. In our configuration, one core in each multicore processor is designated as coordinator, and others are computing cores. The computing core does not really commit I/O requests to storage. The coordinator aggregates multiple I/O operations into one contiguous access with data sieving on behalf of computing cores. The coordinator allocates contiguous memory buffers for other cores to avoid redundant data copies.

The asymmetric collective I/O may be further utilized as intra-node optimizations for a hierarchical inter-node collective I/O.

References

[1] NASA Ames Research Center, NAS application I/O (BTIO) benchmark. 1996.

[2] O. Celebioglu, A. Saify, T. Leng, J. Hsieh, V. Mashayekhi, and R. Rooholamini. The performance impact of computational efficiency on HPC clusters with hyper-threading technology. *Proc. of IPDPS*, 2004.

[3] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. PASSION: Parallel and scalable software for input-output. *report num. SCCS-636, ECE Dept., NPAC and CASE Center, Syracuse University*, pages 38–54, September 1994.

[4] D. Geer. Industry trends: Chip makers turn to multicore processors. *IEEE Computer, 38(5)*, May 2005.

[5] X. He, L. Ou, M. Kosa, S. Scott, and C. Engelmann. A unified multiple-level cache for high performance storage systems. *International Journal of High Performance Computing and Networking*, 5(1), 2007.

[6] M. Seager. Linux clusters for extremely large scientific simulation. In *IEEE International Conference on Cluster Computing*, 2003.

[7] W. Shi, H.-H. S. Lee, L. Falk, and M. Ghosh. An integrated framework for dependable and revivable architecture using multicore processors. *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, June 2006.