# Scalable Parallel File Write from a Large NUMA System

Dong In D. Kang, John Paul Walters, and Stephen P. Crago
Information Sciences Institute
University of Southern California, Arlington, VA 22203
{dkang, jwalters, crago}@isi.edu

***Abstract*— As the number of the sockets and cores within a cache-coherent NUMA (Non-Uniform Memory Access) system increases, parallel file I/O performance is affected more by processor affinity and synchronization overhead within the system. In this paper, we report on the performance impact of processor affinity and page caching overhead on parallel write operations to a single shared file on a large NUMA system. We did experiments using two configurations of an HPE Superdome Flex system – one with 12 sockets and the other with 32 sockets, both having 24 cores per socket, OpenMPI, and the Lustre parallel file system. Our results show that processor affinity and page caching overhead can result in large performance variation depending on the number of MPI processes. We observe that page caching is useful for parallel file writes with a small number of MPI processes, but it is not scalable on a large NUMA system. Parallel file writes without page caching are scalable but achieve higher performance only with a large number of MPI processes.**

***Keywords—NUMA, page cache, processor affinity,***

## I. Introduction

NUMA memory systems are a defining characteristic of today's many-socket cache-coherent shared memory architectures. The HPE Superdome Flex is one example of a large cache coherent NUMA machine and can have up to 32 sockets [1]. Processor affinity in a NUMA architecture affects memory access latency since it determines the relative distance between processor and memory. The latency of a cache coherent memory access between two remote sockets is greater than the latency within a socket. As the size of a NUMA system grows, such latencies are likely to increase and affect overall performance more. Page caches store read/write data from files in physical solid-state memory to accelerate file accesses. Any subsequent read from the data in the page cache greatly benefits from the high speed of the physical memory. The speed of file writes also increases because writing to physical memory is much faster than writing to the disk. Page caches are used widely and are turned on by default in Linux kernel. However, page caching incurs locking overhead when multiple processes access the page cache. The locking overhead may increase as the size of a NUMA system grows.

In this paper we investigate the effect of processor affinity and Linux page caching overhead of large NUMA machines on the performance of parallel file write operations. We used the IOR benchmark 3.3.0+dev [2] and a custom benchmark on HPE Superdome Flex servers to test sequential accesses to a parallel file.

## II. System Architecture and Test Method

We measured performance of parallel writes on a single shared file on two HPE SDF systems. Our smaller HPE SDF has 12 sockets with 24 cores per socket and has 9 TB of memory. Its parallel file system is Lustre [3] v.2.12.0 with 4 OSTs. The larger HPE SDF has 32 sockets with 24 cores per socket and has 21 TB of memory. Its parallel file system is Lustre v.2.12.3 with 118 OSTs. The SDF and the file system are connected with a single 100 Gb InfiniBand adapter.

We used IOR [2] to measure the performance of parallel writes on a single shared file on the smaller HPE SDF system having 12 sockets. We used both MPI and POSIX I/O in the IOR test. We used the following parameters of IOR: block size 32 MB, fsync for POSIX, check write, transfer size 1 MB, segment count 8, and iteration count 5. We used OpenMPI 4.0.2. We used ROMIO to measure the MPI file I/O performance with DirectIO. We used a custom MPI benchmark to measure the performance on the big HPE SDF system having 32 sockets. Its transfer size is 4 MB, and each MPI process had 0.5 GB of data. The benchmark is similar to what IOR does to measure the performance of parallel file writes. We used OpenMPI v. 4.0.1 and POSIX I/O.

## III. Two Performance Knobs

### A. Processor Affinity – Socket vs. Core

We measured the effect of processor affinity on the performance of parallel file write operations to a single shared file on HPE SDF having 12 sockets. In this paper we compared two opposite processor allocation approaches provided by OpenMPI– *map-by-socket* and *map-by-core*. With *map-by-socket*, the processes are bound to successive sockets in a round-robin order. With *map-by-core* the processes are bound to cores in a socket. When the number of cores in the socket are all bound, the cores in the next socket are used. With 24 MPI processes, *map-by-core* places all processes in a single socket. With 12 MPI processes, *map-by-socket* places one process per socket.

The dotted lines in Figure 1 show the MPI I/O performance of IOR with page caching enabled. POSIX I/O performance is similar to MPI I/O performance, but it is not presented in this paper. IOR performance with *map-by-socket* does not show any speedup with more cores. This is problematic for applications whose MPI processes are spread across many sockets that write to a shared file simultaneously. According to our results, it is better to place I/O processes on a single socket if possible. However, such changes may require source code changes and may not be feasible. On the other hand, IOR performance with *map-by-core* is neither as high nor as scalable as we may expect. Its performance is maximized between 12 and 24 cores, which keeps execution in a single socket, and performance drops when more than one socket is used. The performance difference due to processor affinity is maximized when the number of MPI processes is between 12 and 24 for the 12-socket HPE SDF.

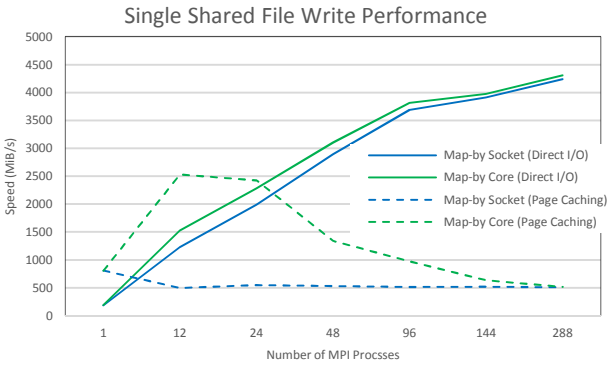


Figure 1. IOR File-Write Performance on HPE SDF with 12 sockets

### B. *Buffered I/O vs. Direct I/O*

We used the *likwid* tool to collect hardware performance counter data while running IOR. It turns out that the percentage of CPUs being in the active state (C0 state of Intel CPU) shows big differences between *map-by-core* and *map-by-socket*. For IOR, the percentage of time in a non-active state with *map-by-socket* is 40-50% higher than the percentage of time being in a non-active state with *map-by-core*. This means more time is spent idling while waiting for data and synchronization. Additionally, the performance trend of *map-by-socket* gives a hint of possible serialization due to high page caching synchronization overhead across multiple sockets.

Linux provides buffered I/O and direct I/O for file I/O. Buffered I/O uses the page cache of the kernel, while direct I/O does not use the page cache but reads and writes directly from storage. IOR provides an option to bypassing page caching for POSIX I/O testing. For MPI I/O, IOR does not support direct I/O but relies on MPI. We built OpenMPI with ROMIO to bypass page caching for MPI I/O. The solid lines in Figure 1 show MPI I/O performance of IOR using direct I/O. POSIX I/O performance is similar to MPI I/O performance, but it is not presented in this paper. Page caching gives a large performance boost for smaller numbers of MPI processes over direct I/O because page caches are far faster than storage. This trend holds with *map-by-core* up to 24 MPI processes, which fits into a single socket. However, IOR with direct I/O is more scalable and outperforms IOR with page caching with more than 24 MPI processes.

## IV. Performance Impact of NUMA System Size

We ran similar tests on the larger HPE SDF system using a custom benchmark due to unavailability of IOR on the system. We chose a stripe size of 4 MB and a stripe count of 32. Since there are significant differences between the Lustre file systems of the two HPE SDF system, direct comparison of the parallel file write performance between the systems may not be meaningful. However, the trend within a system can reveal the effect of the processor affinity and the effect of page caching. The effect of processor affinity shows a similar trend to that on the smaller HPE SDF system in Figure 2 when page caching is used. Again, the performance drops when the MPI processes are mapped across more than one socket with the page cache enabled. The *Map-by-Core (Direct I/O)* performance in Figure 1 and Figure 2 shows much higher performance of the Lustre file system on the larger HPE SDF. The gap between the performance of *map-by-core* and *map-by-socket* is much larger on the larger HPE SDF system. This can be explained in two ways. First, the smaller HPE SDF system is limited mainly by the slower Lustre file system. Second, the synchronization overhead across sockets on the larger HPE SDF becomes a primary bottleneck when MPI processes are spread across the whole system.

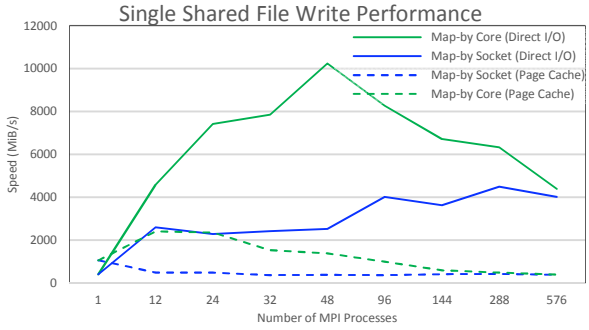


Figure 2. Parallel File-Write Performance on HPE SDF with 32 sockets

## V. Related and Future Work

Lustre over-striping can increase shared file I/O performance [4]. The Lustre systems we used are older versions and lack over-striping support. We plan to study the effect of over-striping as future work. There are approaches to improving I/O performance with user-level caching using delegate nodes [5] and using asynchronous I/O [1]. Our work can be used as a guide on how to allocate delegated I/O nodes in large NUMA machines. Asynchronous I/O may spread I/O traffic in time but does not increase peak throughput. Asynchronous I/O can also get benefits from our findings on large NUMA machines.

## VI. Conclusion

This paper showed that processor affinity and page caching overhead for shared file access within a large NUMA system can affect parallel file I/O significantly. We plan to do an in-depth analysis of the overhead. We will also consider more real-world parallel file I/O patterns.


### *Acknowledgment*

The authors would like to thank Mr. Jeremy Filizetti for his discovery of buffered I/O limitations on the HPE SDF.


## References


[1] HPE SUPERDOME FLEX SERVERS (https://www.hpe.com/us/en/servers/superdome.html)

[2] IOR benchmark (https://github.com/hpc/ior)

[3] Lustre (http://lustre.org/)

[4] M. Moore, P. Farrell, “Exploring Lustre Overstriping For Shared File Preformance on Disk and Flash,” 2019 Lustre User Group (LUG) conference, Houston, TX, May, 2019.

[5] A. Nisar, W. Liao and A. Choudhary, "Scaling parallel I/O performance through I/O delegate and caching system," SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Austin, TX, 2008.

[6] H. Tang, Q. Koziol, S. Byna, J. Mainzer and T. Li, "Enabling Transparent Asynchronous I/O using Background Threads," 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW), Denver, CO, USA, 2019.



Identify applicable funding agency here. If none, delete this text box.