Supporting I/O-Intensive Workloads on the Cell Architecture

M. Mustafa Rafique, Ali R. Butt, and Dimitrios S. Nikolopoulos*
Virginia Tech

ABSTRACT

Recent advent of the asymmetric multi-core processors such as Cell Broadband Engine (Cell/BE) has popularized the use of heterogeneous architectures in High-End Computing. Data and I/O intensive workloads have largely been ignored in this domain. We take the first steps in supporting I/O intensive workloads on Cell and deriving some guidelines for optimizing the execution of I/O workloads on heterogeneous architectures. We explore various performance enhancing techniques for such workloads on an actual Cell/BE system. Among the techniques we explore, an asynchronous prefetching-based approach, which uses the PowerPC core of the Cell/BE for file prefetching and decentralized DMAs from the synergistic processing cores (SPE's), improves the performance for I/O workloads that include an encryption/decryption component by 30.2%, compared to I/O performed naïvely from the SPE's.

1. INTRODUCTION

Recent advent of the Cell Broadband Engine (Cell/BE) processor [1] attests to the potential of emerging asymmetric multi-core architectures for accelerating applications that require high-performance computation on streaming data [2]. This work explores the use of the Cell/BE in I/O-intensive applications. With modern high-performance computing applications generating and processing exponentially increasing amounts of data, the parallel processing capabilities, on-chip data transfer bandwidth, and computation/DMA overlap mechanisms of the Cell/BE render it a viable platform for high-performance I/O.

So far, there is little understanding of how I/O operations interact with the architecture of the Cell/BE. The implications of such Cell/BE characteristics as asymmetry, DMA/computation overlap, and software management of disjoint address spaces, on the I/O software stack have not been explored. We address these important questions and make the following contributions: (i) A study of the I/O path in the current Cell/BE operating system and runtime environment; (ii) An exploration of alternative I/O methods on the Cell architecture; (iii) An investigation of data prefetching techniques for improving I/O performance on the Cell/BE; and (iv) An evaluation and recommendation of appropriate methods for handling I/O intensive workloads.

2. MOTIVATION AND BACKGROUND

We are concerned with the implementation of efficient I/O schemes for data-intensive applications on asymmetric multi-core processors. Applications typically offload compute-intensive kernels to the specialized cores of these processors, exploiting hardware capabilities such as SIMD units to accelerate the processing of streaming data. Clearly, such capabilities are relevant to I/O-intensive applications with significant data processing components such as encryption and compression.

An important design consideration for the I/O subsystem on asymmetric multi-core processors is the distribution of the I/O processing path between heterogeneous cores. Current designs run the operating system on the commodity "host" cores of the processor (e.g. the PowerPC PPE of the Cell) and route I/O requests made from the "accelerator" cores (e.g. the SPE's of the Cell) through the host cores. While this approach simplifies system software it imposes bottlenecks during parallel I/O from the accelerator cores.

In contrast to conventional processors, asymmetric multi-core processors delegate more control of the memory hierarchy to software. Cell exposes the local store of each accelerator core directly to software, through a DMA mechanism, which further enables overlap of multiple DMA requests with computation on each core. This capability extends naturally to the I/O subsystem, which should properly stage data from the disk, to off-chip memory, to on-chip memory, so that the non-overlapped data transfer latency is minimized. The design space for data staging in the I/O system involves tuning of the unit of data transfer between layers of the memory hierarchy, synchronous and asynchronous prefetching algorithms, and synchronization and communication mechanisms between host cores and accelerator cores to coordinate I/O requests.

3. I/O IMPROVING TECHNIQUES

In this section, we present and evaluate a number of I/O improving techniques for the Cell/BE architecture. For our evaluation, we use different I/O workloads executed on a Sony Play Station 3 (PS3). The PS3 is a hypervisor-controlled platform. It has 6 active SPE's with 256 KB local storage, 256 MB of main memory of which about 200 MB is directly accessible to the OS, and a hard disk of 60 GB. Although Cell/BE has 8 SPE's, on the PS3, one SPE is reserved for running the hypervisor and another SPE is deactivated. Accesses to storage devices, including the disk, are routed through the hypervisor with dedicated hypercalls and their completion is communicated to the OS through virtual interrupts. Due to the proprietary nature of the PS3 hypervisor, it is not possible to assess its imposed overhead on accesses to storage devices for the purpose of this work.

Available I/O bandwidth: To determine the I/O bandwidth available in our test machine, we first measured the time it takes to read a large file (2 GB) in block sizes of 4 KB and 16 KB, both at the PPE and a SPE. We observed that the read time at the SPE is similar to that at the PPE (4966 ms). Moreover, using a larger block size has negligible effect at PPE, but provides better throughput (8.9%) at SPE. We note that the improved SPE throughput is due to the matching of the block size with the 16 KB DMA request size of our test machine. Further, the average observed I/O throughput is reduced by 6.4% for the same amount of data when we repeated this test on all the six SPE, because of increased contention for the EIB, compared to the case of using a single SPE.

Workload: In the next set of experiments, we used a

*{mustafa, butta, dsn}@cs.vt.edu

	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
Read at PPE	-	1057	1509	1039	1310
Read at SPE	693	351	389	-	-
DMA read at SPE	-	-	-	87	172
Process at SPE	2458	2499	2558	2347	2504
DMA write at SPE	-	-	-	524	923
DMA wait at SPE	-	-	-	-	0.12
Write wait at PPE	-	-	-	-	1221
Write at SPE	2245	2098	2400	-	-
Write at PPE	-	-	-	1607	1455
Total	6397	6794	6258	6008	4468

Table 1: Average timing breakdown (in msec.) of various tasks for different I/O schemes.

	Scheme 1	Scheme 5
Read at 1 SPE	426	-
Read at PPE	-	2381
DMA read at 1 SPE	-	494
Process at 1 SPE	1052	859
DMA write at 1 SPE	-	317
DMA wait at 1 SPE	-	26
Write wait at PPE	-	213
Write at 1 SPE	1412	-
Write at PPE	-	1381
Total (6 SPE's)	3965	3640

Table 2: Average timing breakdown (in msec.) when all 6 SPE's are used.

workload that reads a file, encrypts it with a 256-bit key, and writes the result back to the disk. We also vectorized the computation phase of our workload to achieve 42.1% better performance on SPE's compared to the non-vectorized version. We chose a 64 MB file as input so that the entire file can be kept in the 200 MB available memory and any side-effects due to buffer caching are removed.

Effect of file caching: All system calls on the Cell/BE including I/O calls from the SPE's are handled by the PPE. This implies that once a file (or portion of a file) is accessed by the PPE it may be in memory when the file is subsequently accessed from a SPE or the PPE, and the later accesses can be serviced quickly. We confirmed this empirical observation by first flushing any cache by reading a large file, and then repeatedly reading the file both at the PPE and SPE. We found that a read at SPE is 82.9% faster, on average, if it follows a read for the same data at the PPE.

File caching techniques: Next, we explored five schemes for doing prefetching to improve I/O performance of our workload. Table 1 shows the results.

In Scheme 1, our base case, all the workload tasks are performed on the SPE. We still utilize the PPE, though, to invoke the tasks as a single program on the SPE.

In Scheme 2, PPE first pre-reads the entire file causing it to be brought in memory. Then the program from Scheme 1 is executed as before. We observe that although the *Read at SPE* for this Scheme is much faster, the time it takes to read the file at the PPE is 52.5% longer compared to *Read at SPE* in Scheme 1. We believe this is due to the PPE flooding the I/O controller queue, and lack of overlapping opportunities between computation and I/O in a sequential read compared to the read and process cycle of Scheme 1.

In Scheme 3, a separate thread prefetches the file in memory. The SPE still runs the program of Scheme 1. Simultaneous prefetching and processing can remove the bottleneck faced in Scheme 2. The prefetching thread improves SPE I/O implicitly by bringing data in memory before it is requested by the SPE. A lagging thread does not pose a synchronization issue because in this case the SPE requests will be serviced from the disk, which in turn will make the thread read the file faster and get ahead of the SPE. Results for this Scheme show that although the I/O times for individual steps increased, better I/O computation overlapping resulted in an overall improvement of 7.9%, compared to Scheme 2.

In Scheme 4, we explicitly prefetch the file at the PPE and give the SPE the address of memory where the file data is available. The SPE program is modified to not do direct I/O, rather use DMA on the addresses provided by the PPE. Results show that the *DMA read at SPE* takes 75.2% and 77.6% less time than *Read at SPE* in Scheme 2 and Scheme 3, respectively. However, the synchronous reading of file causes the overall times to not improve as much: 11.6% and 4.0% compared to Scheme 2 and Scheme 3, respectively.

In Scheme 5, we also use a prefetching thread at the PPE, but instead of the implicit approach of Scheme 3, we use explicit DMA transfers between the PPE and the SPE. To ensure that the prefetching thread on the PPE and the processing thread on the SPE are properly synchronized we employ a "shared" *status location* in main memory to exchange information regarding what portions of the file have been read (processed), with the "sharing" enabled via repetitive DMA's. Results show that Scheme 5 achieves 30.2%, 28.6%, and 25.6% improvement in overall performance compared to Scheme 1, Scheme 3, and Scheme 4, respectively.

Scalability test: Next, we extended Scheme 5 to use all the 6 available SPE's, and compared it with a scaled version of Scheme 1. The workload size for Scheme 1 is set to 10.67 MB (64/6) per SPE to give a total of 64 MB. Similarly, for Scheme 5, the workload is evenly distributed among the 6 SPE's, and the PPE performs all the I/O. Table 2 show the results. The reduced file-reading time at each SPE is due to each SPE processing only a fraction of the file. Overall, Scheme 5 performs better (8.2%) than the approach of Scheme 1, even when all available SPE's are used.

4. CONCLUSION

We investigated the use of Cell/BE architecture for supporting I/O intensive workloads involving significant computation components, such as encryption. We observe that the current operating system facilities for performing I/O directly on SPE's are limited, and do not provide judicious use of the available resources. A particular concern is that currently, I/O on SPE's is redirected to the PPE, hence creating a central bottleneck. We have presented an asynchronous prefetching-based approach that shows promising results. However, we argue that a fundamentally better approach would be to extend SPE support libraries with I/O functionality, thus removing the dependence on the PPE, simplifying the SPE program design for I/O intensive workloads, and improving overall performance.

5. REFERENCES

[1] T. Chen, R. Raghavan, J. Dale, and E. Iwata. Cell Broadband Engine Architecture and its First Implementation – A performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007.

[2] B. Gedik, R. Bordawekar, and P. S. Yu. Cellsort: High performance sorting on the cell processor. In *Proc. VLDB*, 2007.