Achieving Real Data Throughput for an FPGA Co-Processor on Commodity Server Platforms

Roger Chamberlain, Berkley Shands, and Jason White

Roger Chamberlain, Berkley Shands, and Jason White, “Achieving Real Data Throughput for an FPGA Co-Processor on Commodity Server Platforms,” in *Proc. of 1st Workshop on Building Block Engine Architectures for Computers and Networks*, Boston, MA, October 2004.

Data Search Systems, Inc. and
Washington University in St. Louis

Achieving Real Data Throughput for an FPGA Co-Processor on Commodity Server Platforms

Roger Chamberlain*†, Berkley Shands†, Jason White*
*Data Search Systems, Inc., St. Louis, Missouri
†Dept. of Computer Science and Engineering, Washington University, St. Louis, Missouri
roger@wustl.edu, berkley@wustl.edu, jwhite@dssimail.com

Abstract

Significant investigation has taken place in the area of exploiting reconfigurable logic as a co-processor in embedded and high-performance computer systems. This paper examines the problem of delivering high throughput data to the co-processor. Data is retrieved from magnetic storage and delivered to the co-processor through datapaths available in inexpensive, commodity server platforms. End-to-end data throughput of 667 MB/s is achieved.

1. Introduction

Co-processor building blocks have received a great deal of attention over the years. From the early days, when memory management units and floating-point units were off-chip, to now, when a functional unit is more likely to perform an encryption/decryption operation, architectural issues of how the unit interacts with the processor are crucial to overall performance. In this paper, we will use the term co-processor to refer to any functional unit that assists the primary system processor in executing the computing requirements of an application or set of applications. The co-processors that our group has implemented are constructed using Field Programmable Gate Array (FPGA) technology, although Application Specific Integrated Circuit (ASIC) technology is also commonly used.

When the computational granularity of the function to be performed is low, it is important to have a co-processor tightly coupled to the primary processor. On the other hand, course-granularity functions can be more loosely coupled to the primary processor and still provide excellent system-level performance.

There are a host of computationally intensive, course-granularity functions that have historically warranted co-processor designs. Examples such as digital signal processing in embedded systems, graphics in display systems, and encryption/decryption in secure systems have typically had higher compute requirements than I/O requirements. With recent advances in the computing capability of FPGA technologies, I/O throughput is now often the performance limiting factor.

The focus of our work has been the effective processing of data stored on disk. The dramatic improvements in magnetic storage density over the past several years (a 100% compound annual growth rate for the last decade) have outstripped the improvements in semiconductor technology and associated improvements in processor performance. The result is dramatically lower prices for magnetic storage and an inability to keep up with the computing requirements for serious analysis of large data sets.

We have designed a number of application modules that address the computing needs of large data sets. These include both exact and approximate keyword searching of unstructured text [1,2,3], biosequence similarity search [4], structured data search, etc. Others have built modules for encryption, compression, hashing, and a host of other applications [5,6].

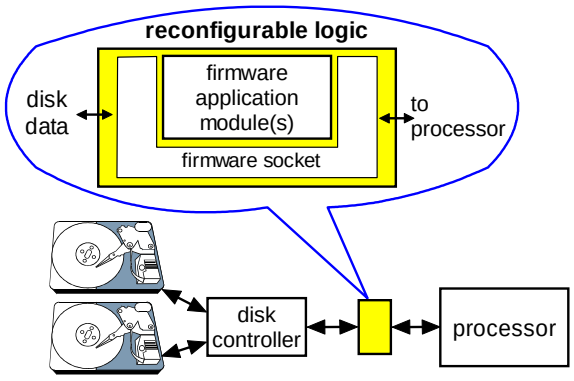

Figure 1. System organization.

The co-processor positioning and nominal high-volume data flow in our system are

illustrated in Figure 1. Data flows off the disk subsystem into the FPGA-based co-processor. The FPGA provides reconfigurable logic that has its function specified via firmware. We have designed and built a standard firmware socket that handles data movement requirements into and out of the FPGA, providing a consistent application interface to firmware application modules that are deployed within the co-processor.

Figure 2 shows the framework for the deployment of applications on our system. The top three layers represent functionality that is executed in software on the system processor, while the bottom two layers represent functionality that is executed in firmware on the co-processor FPGA. The central three layers of this framework are fairly application independent, supporting the data movement between the application software at the top of the framework and the firmware application module(s) at the bottom of the framework. It is the performance of these central framework layers that are the focus of this investigation. As a result, the performance figures presented are independent of the application.

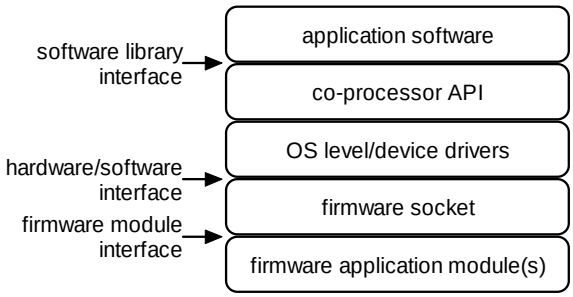

Figure 2. Application deployment framework.

This paper examines the I/O throughput capabilities of the above system concept deployed on a pair of commodity server platforms. Data is read from a disk storage subsystem and delivered at a high rate to the FPGA-based co-processor. Within the FPGA, the firmware socket provides the input data to the firmware application module and delivers output to the processor. We demonstrate high data throughput on relatively inexpensive hardware, enabling the use of co-processing techniques on a wider class of otherwise data-limited applications.

The remainder of the paper is organized as follows. Section 2 describes the platforms on which the performance experiments were run. Section 3 presents the experimental results and discusses the implications of the results for the co-processor applications developer. Section 4 concludes and describes future work.

2. Experimental Platforms

The experimental platforms examined include a pair of generic, commodity servers. The first is based on the Intel Xeon [7], the second is built around the AMD Opteron [8].

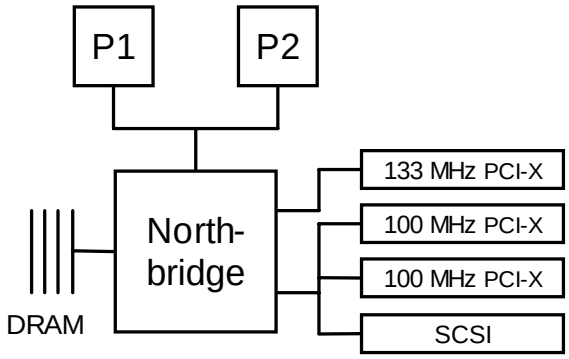

Figure 3. Xeon system.

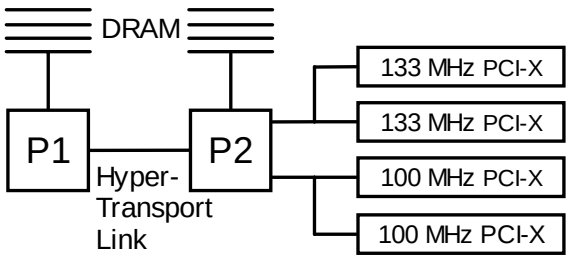

Figure 4. Opteron system.

The Intel server is a dual processor 2.6 GHz Xeon with HyperThreading enabled, 2 GB of memory, a SuperMicro X5DA8 motherboard with an on-board Adaptec AIC-7902 dual channel Ultra320 SCSI controller, and an additional Adaptec 39320A-R dual channel SCSI controller. This system was constructed at a cost of under $3,000. Figure 3 illustrates the pertinent features of the motherboard in this server.

The AMD server is a dual processor 2.0 GHz Opteron with 8 GB of memory, a Tyan S2885 motherboard and a pair of Adaptec

39320A-R dual channel SCSI controllers. This system was constructed at a cost of under $8,000. The pertinent features of the motherboard are illustrated in Figure 4.

The system software for both of the platforms includes RedHat ES3.0 for i686 and AMD64 (x86_64) with a standard Linux 2.6.6 kernel supplemented with an updated Adaptec driver (revision 2.0.12). For file system read measurements, a software-based RAID0 ext3 file system is used.

The FPGA co-processor is on a PCI-X board (manufactured by AvNet) that we have inserted into one of the 100 MHz PCI-X slots available on each motherboard. The FPGA itself is a Virtex-II series XC2V4000 part.

The disk storage subsystem includes a pair of StorCase 14-bay Infostations with a mixture of Seagate Cheetah ST336753LC and Fujitsu MAS3735NC Ultra320 SCSI disk drives. The drives are all 15,000 rpm drives and are each configured to use a single 36 GB partition. (The second 36 GB partition on the Fujitsu drives is left unused for the experimental results presented here.) The organization of the disk subsystem is illustrated in Figure 5. Two independent SCSI channels are supported on each of two independent disk controller chips, for a total of 4 parallel SCSI channels. Between 4 and 7 drives are attached to each SCSI channel, resulting in a maximum storage capacity of 1 TB.

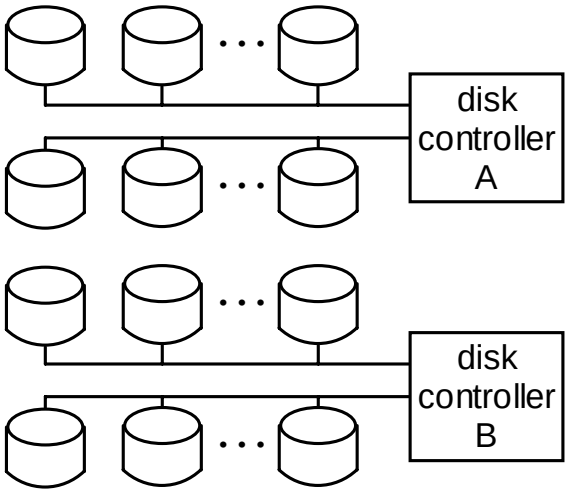

Figure 5. Disk storage subsystem.

Disk controllers A and B are alternately positioned within the two server systems, either both on a common PCI-X bus or separated across 2 PCI-X busses. Data striping (both for raw reads and reads from the RAID0 file system) alternates first across controllers, then across SCSI channels within a controller, and finally across the drives on an individual SCSI channel.

3. Performance Measurements

In general, it is possible to support multiple co-processors within a single platform, enabling performance gains when performance is limited by the FPGA computing rate. As a result, the performance results presented here are separated into a pair of categories. We first present end-to-end throughputs achievable from the disk drive subsystem into the co-processor. The rest of the results are focused on moving data into main memory from the disk and/or from main memory to/from the co-processor. All of the throughput results presented here were measured by timing the total transfer time of data sets that range from 2 GB to 500 GB.

3.1 End-to-End Performance

The highest end-to-end throughput results were measured on the Opteron platform using our newly developed firmware socket interface operating at 100 MHz. Sustained throughput for this configuration is 667 MB/s, within 83% of the peak rate of the PCI-X bus.

The highest measured bus utilization occurred using the previous edition of the socket interface design, which operated at 66 MHz on the PCI-X bus. A sustained rate of 490 MB/s on either platform was achieved, for a bus utilization of 98%. This is tempered, however, both by the fact that the bus speed was limited to 66 MHz operation and by the fact that the data transfer was heavily in one direction. The application module loaded into the FPGA for this test was a search function that returned very few hits. Hence, the bulk of the data was inbound into the co-processor, with very little data volume outbound. In addition, sustaining this data rate on the Xeon platform required “zero copy” data movement from user space into the FPGA (avoiding a data copy through kernel space). As a side note, informing the OS that it can replace specific data blocks in the file cache drops the processor utilization in the above experiments from about 10-15% down to about

3%. This is accomplished through the use of the `fadvise()` system call. The data movement itself is being accomplished by the DMA functionality built into the firmware socket interface on the FPGA.

3.2 Read Performance

Given that the end-to-end performance is currently limited by the PCI-X bus path to and from the co-processor, we next turn our attention to the data throughput achievable reading from the disk array. This is of interest for two reasons. First, it enables us to assess potential limits on end-to-end throughput once the firmware socket interface is operating at 133 MHz (the current top end for PCI-X), and second, our infrastructure (both firmware socket interface on the FPGA and software drivers and kernel modifications within the OS) supports multiple co-processors in a single system. Unless otherwise stated, all the read experiments use a pair of concurrent read threads.

The first set of disk array read experiments investigates raw I/O performance, performing contiguous logical block reads without the presence of a file system. Striping the reads across 4 SCSI channels with 5 drives per SCSI channel, the Xeon system was capable of sustaining a data throughput of 785 MB/s and the Opteron system was capable of 955 MB/s sustained throughput. Increasing the number of disk drives per SCSI channel above 5, the capacity of the disk subsystem goes up, but the I/O read throughput is essentially unchanged.

The next set of read experiments investigates disk array performance when a RAID0 file system is present. On the Xeon platform, the highest sustained data throughput on a read from the 1 TB capacity file system was 392 MB/s. On the Opteron platform, the equivalent sustained throughput was 600 MB/s. Using a pair of independent file systems, one on each Adaptec controller, each file system having 2 SCSI channels and 14 drives, yields a total sustained throughput of 700 MB/s (395 MB/s + 305 MB/s).

Figure 6 and Figure 7 illustrate the variability in read performance off the full 1 TB RAID0 file system (7 drives per SCSI channel) as a function of the OS-provided read ahead parameter. These experiments were performed on the Opteron platform, with a read block size of 11 KB.

Figure 6 represents the accumulated results from 4000 individual experiments, with the read ahead parameter ranging from 1 to 4000 pages. Clearly, there are 4 distinct performance bands present, indicating a strong sensitivity to this individual parameter in the achievable data throughput.

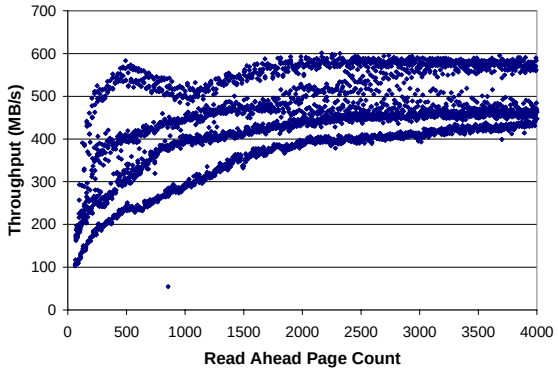

Figure 6. Read performance as impacted by read ahead parameter.

Figure 7 explores this performance sensitivity by zooming in on a small region of the graph. Here, the structure of the variation in performance is quite evident, exhibiting a regular periodic shape. This graph structure is replicated almost verbatim in a common experiment executed on the Xeon platform, shown in Figure 8. While the overall throughput is somewhat lower on the Xeon system, the periodic nature of the performance variation is remarkably similar, even to the point of having the same fundamental frequency.

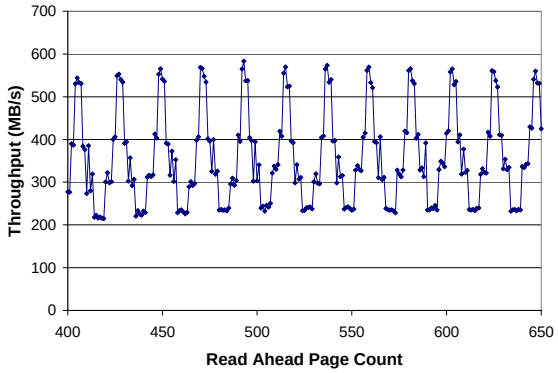

Figure 7. Detail of read performance as impacted by read ahead parameter.

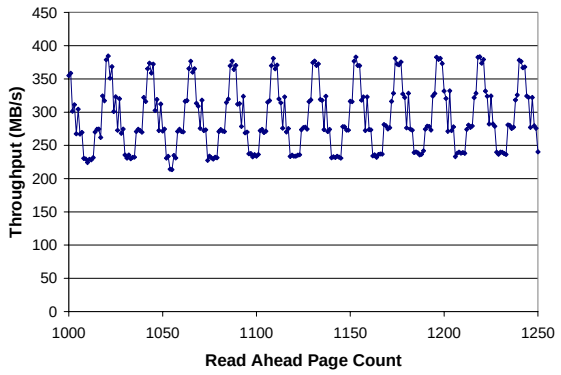

Figure 8. Detail of read performance on Xeon platform.

The final experiment in this set examines the performance sensitivity to another tuning parameter, the block size employed for file read operations. Figure 9 shows these results for the Opteron platform. Again, we see significant performance impact on the part of this parameter, and in this case the pattern is much less regular than for the read ahead parameter examined previously.

These experiments point to the explicit need for empirical characterization of actual performance in systems of this type. In each case, a reasonable expectation would be that above some threshold, the parameters investigated above would have minimal (if any) performance impact. Yet, their influence on performance is not only clear, it is dramatic, well into regions that were unexpected. Excellent throughputs can be achieved, but not without concerted effort that includes empirically-driven performance tuning.

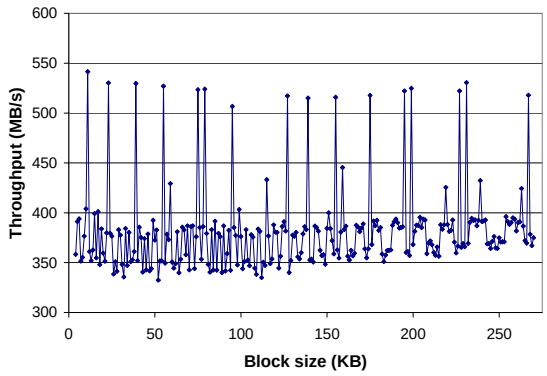

Figure 9. Read performance as a function of block size.

3.3 Modified Kernel

The performance figures presented so far were shared with the Linux development community and, as a result, some changes were made in the read ahead infrastructure for subsequent versions of the kernel [9]. Figure 10 shows the results of repeating the experiment of Figure 9 under the 2.6.9 kernel using a single read thread and a half-size disk array (only one of two controllers). The highly variable performance seen earlier is effectively eliminated.

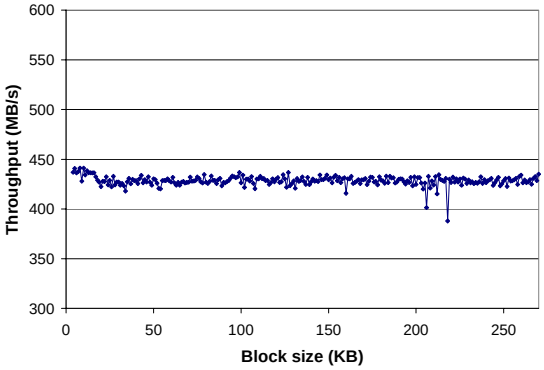

Figure 10. Read performance as a function of block size (2.6.9 kernel).

Figure 11 and Figure 12 show the results of repeating the experiment which explores read performance as a function of the read ahead parameter under the 2.6.9 kernel and a half-size disk array. We are back to two read threads in this experiment. Here, the regularity of the performance variation is now dramatically lower.

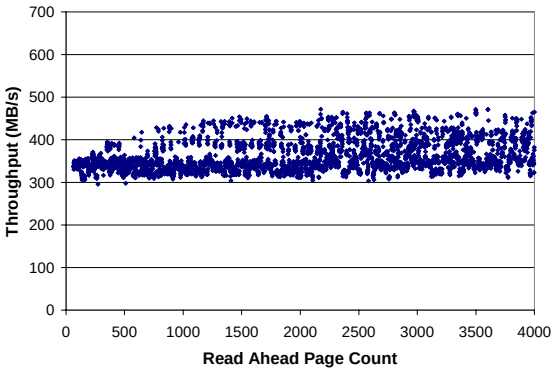

Figure 11. Read performance as impacted by read ahead parameter (2.6.9 kernel).

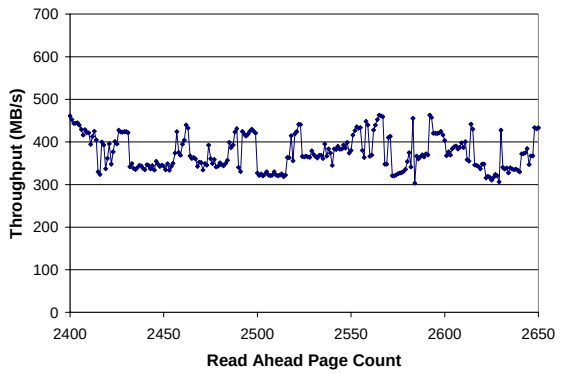

Figure 12. Detail of read performance as impacted by read ahead parameter (2.6.9 kernel).

4. Conclusions and Future Work

This paper has presented an empirical study of achievable data throughput for an FPGA-based co-processor on a pair of commodity server platforms. The sustained end-to-end measured throughput of 667 MB/s speaks to the enormous capacity of these affordable, off-the-shelf systems.

These performance levels, however, are not achieved without considerable tuning of multiple parameters within the system, often relying on empirical measurement to guide the tuning process. Our initial intuition with respect to both the form and the magnitude of the performance sensitivity to these parameters was significantly in error (consistently under-estimating the performance sensitivity across the board).

Our current work is focused in two areas. First, we are expanding the library of available functions that we can deploy as application modules within the FPGA. Second, we are accelerating the operating frequency of the firmware socket interface so that it can support the 133 MHz PCI-X bus.

Finally, all of the file system reads presented here are for a RAID0 system. We have made some preliminary measurements using a software-based RAID5 file system. The throughput of this file system is limited, however, to no greater than about 250 MB/s, independent of the server platform or disk subsystem configuration. In future work, we anticipate moving some of the required data processing needed to support a RAID5 file system (e.g., the data integrity computations) into the FPGA co-processor. This would enable us to provide similar high-throughput performance in a high-availability environment as well.

References

[1] R.D. Chamberlain, R.K. Cytron, M.A. Franklin, and R.S. Indeck, "The *Mercury* System: Exploiting Truly Fast Hardware for Data Search, in *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os*, September 2003, pp. 65-72.

[2] Q. Zhang, R.D. Chamberlain, R.S. Indeck, B. West, and J. White, "Massively Parallel Data Mining Using Reconfigurable Hardware: Approximate String Matching," in *Proceedings of Workshop on Massively Parallel Processing*, April 2004.

[3] M. Franklin, R. Chamberlain, M. Henrichs, B. Shands, and J. White, "An Architecture for Fast Seaching of Unstructured Data," to appear in *Proceedings of International Conference on Computer Design*, October 2004.

[4] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, and J. Lancaster, "Biosequence Similarity Search on the *Mercury* System," in *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processor*s, September 2004.

[5] *Proceedings of Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 1993 to 2004.

[6] *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*, 1993 to 2004.

[7] Intel Corporation, "Building Cutting-Edge Server Applications," Whitepaper, 2002.

[8] C.N. Keltcher, K.J. McGrath, A. Ahmed, and P. Conway, "The AMD Opteron Processor for Multiprocessor Servers," *IEEE Micro*, **23**(2):66-76, March-April 2003.

[9] Andrew Morten, Ram Pai, and Shane Shrybman, personal communication.