Zero-copy Queues for Native Signal Processing Using the Virtual Memory System

Gregory E. Allen, Paul E. Zucknick, and Brian L. Evans
Applied Research Laboratories, and Dept. of Electrical and Computer Engineering
The University of Texas at Austin
P.O. Box 8029, Austin, TX 78713-8029
{gallen,zucknick}@arlut.utexas.edu, bevans@ece.utexas.edu

Abstract—**The high performance of current general-purpose processors makes it feasible to perform digital signal processing on the main CPU of a workstation. As the performance gap between CPU speed and memory speed continues to increase, it becomes obvious that DSP applications on workstations must address memory performance. We present a technique for using the virtual memory system to implement zero-copy queues. We present benchmark results for a frequency-domain FIR filter using this technique, compared to manual data circularity management. The technique presented can significantly reduce the overhead of data management when processing continuous, overlapping streams of data.**

I. Introduction

Current high-performance general-purpose CPUs make it feasible to perform substantial signal processing on the main CPU of a workstation. The term "native signal processing" (NSP) has been used to describe this approach [1]. With the various Single Instruction Multiple Data (SIMD) instruction sets included in modern CPUs, several GFLOPS of computational power is available per core with single-precision floating-point numbers. With multi-core CPUs and multi-CPU workstations becoming available, single workstations are capable of tens of GFLOPS, and these numbers are increasing. Highly optimized libraries are available for common DSP tasks, making these SIMD architectures accessible without programming for them directly. Examples include Intel's Math Kernel Library (MKL) [2] and the Vector Signal & Image Processing Library (VSIPL) [3] .

Although memory speeds have also been increasing, they have not kept pace with the rate of increase for CPU arithmetic performance. In high-throughput signal and image processing systems, memory bandwidth and latency can become large bottlenecks. As CPU parallelism increases, the contention for a slower shared memory becomes even more problematic. Although high-performance signal processing libraries are available for many common algorithms, the user programmer typically must manage data input and output, and assure that the data is arranged in memory in whatever manner the library requires. Copying or rearranging high-throughput stream data can be a very expensive operation, but having data in the proper arrangement in memory can make astounding differences in library performance. Time spent by the CPU to copy data or time spent waiting on a slower memory is time spent not performing DSP.

Supported by the Independent Research and Development program at Applied Research Laboratories: The University of Texas at Austin.

Many common DSP algorithms operate on continuous, overlapping (or "sliding window") streams of data, e.g. FIR filters or overlap-and-save FFTs. On hardware DSP processors, modulo (or "circular") addressing modes are available for efficient implementation of these algorithms [4]. The appearance of a circular queue in memory is maintained with the addressing hardware, such that no copying of data by the DSP is necessary to maintain circularity. However, general-purpose processors seldom have such modulo addressing modes, so this circularity must be maintained in software. Although it is possible to write a signal processing kernel function to manage this circularity, it requires additional address computation overhead, and is therefore seldom done. Indeed, libraries are typically written to operate directly on contiguous blocks of data. Therefore, some amount of copying is generally necessary to manage circularity and overlapping portions of data. For large overlaps with high-throughput stream data, this can be a significant overhead.

We use the virtual memory manager (VMM) to achieve the equivalent of modulo addressing for native signal processing on Unix workstations. This effectively emulates circular buffers, and permits zero-copy queue implementations. That is, no copying is necessary to manage a continuous, overlapping stream of data. The appearance of circularity is maintained by the virtual memory system.

II. Manual Overlap Management

Because general-purpose processors generally lack modulo addressing modes, software must copy data in order to manage memory for algorithms operating on continuous, overlapping streams of data. This copying is undesired overhead that would not be necessary on a DSP processor.

Fig. 1 shows the copy operations that would be necessary for software to manage a buffer used with a sliding window algorithm. For a continuous stream of data, the following steps would by repeatedly executed:

1. copy old overlap data from end to start of buffer
2. copy new stream data to be after overlap data
3. execute algorithm on contiguous buffer

The needed overlap data length depends on the algorithm being executed. For an FIR filter, it is the filter order. Increasing the length of the buffer decreases the relative amount of

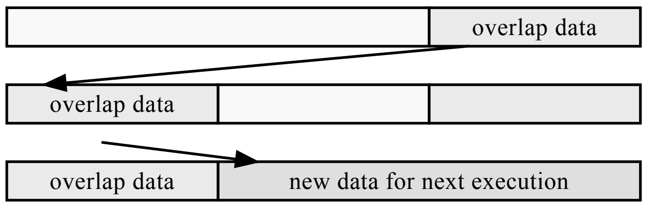

Fig. 1. Copies necessary for manual management of data overlap.

copying overhead at the expense of more memory usage.

III. Circularity with the VMM

By using the Virtual Memory Manager (VMM) on a Unix system, we can achieve the equivalent of circular buffers, eliminating the need for manual buffer management via copying of overlap data.

The Portable Operating System Interface (POSIX) [5] is an open set of standard operating system interfaces based on Unix, produced by the IEEE and recognized by ISO and ANSI. As such, it is portable across a large number of platforms. POSIX provides a standard operating system call, `mmap(···)`, for mapping files or devices into a process's virtual memory map with the VMM.

By mapping the same file (or shared memory object) to multiple, contiguous locations, we have achieved virtual circularity. This mapping is maintained by the CPU's virtual-to-physical address translation hardware. The user process has contiguous access to a block anywhere in the buffer up to the length of the mirrored data. One must never access memory past the final mapping, but this can easily be avoided by performing a modulus operation after incrementing any pointer that accesses the buffer. There can be some overhead incurred in using virtual memory mappings, but there is the benefit of avoiding copies of high-throughput data. Fig. 2 shows the memory layout of a queue using this technique.

The following steps are used to create a circular buffer:

1. create an object that can be mapped with `mmap(···)` (such as POSIX shared memory or a temporary file)
2. map the object with `mmap(···)`, typically allowing the operating system to assign the virtual address
3. map the object again with `mmap(···)`, specifying the virtual address adjacently after the previous mapping

We have demonstrated this technique to work on Linux (x86 and PowerPC), MacOS X, and AIX.

IV. Measuring Performance

We quantify the performance improvement of these zero-copy queues by benchmarking a commonly implemented algorithm – a frequency domain FIR filter. We use the well-known "Fastest Fourier Transform in the West" (FFTW) library [6] for our forward and inverse FFTs, and a SIMD vector complex multiplication function. Our point of comparison is the manual overlap management approach, as would be necessary without the zero-copy memory management technique. Before each forward FFT, overlap data is copied from the tail of the buffer to the head, and the new data is copied in. After each inverse FFT, the "good" (non-circularly-aliased) result data is copied to an output buffer.

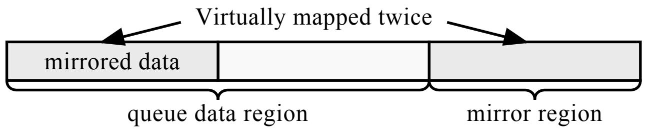

Fig. 2. Creating the appearance of a circular queue with virtual memory.

By varying algorithm parameters during the benchmark, such as filter length and FFT length, we are able to make some general performance evaluations about the zero-copy queues. We use power-of-two FFTs from 16 to 65536, and vary the filter length as a percentage of the FFT length over several steps. Note that the filter length is the same as the overlap length, so this test is also varying the percentage of overlap. We do not propose that all of the benchmarked cases are a desirable approach for implementing such an FIR filter in a signal processing system, but we are interested in varying the parameters to explore the zero-copy queue performance over a wide workload.

We run these benchmarks in two different POSIX-compliant environments, with minimal software changes necessary between the platforms. We benchmark with a data size large enough to prevent it from fitting in cache (typically over 100MB), and measure the execution time as an average over 10 trials. We compute MFLOPS from the measured execution time for a forward and a reverse FFT, and a complex multiply, as

$$10N \log_2 N + 6N. \tag{1}$$

This is the same method that the FFTW project uses to compute the number of FLOPs in an FFT.

V. Results

We present benchmark results from two architectures and environments: a 2.5 GHz PowerPC 970 (Apple Power Mac G5) running MacOS X 10.4.6, and a 1.8 GHz AMD Opteron running 64-bit Red Hat Enterprise Linux 4 (RHEL4). The Opteron processor is installed in a Shuttle SN21G5 system. In all cases, we use single-precision floating point with SIMD instruction set libraries. These benchmarks use only a single thread, and therefore only a single CPU core.

A. *MacOS X on PowerPC*

Fig. 3 shows a subset of the performance results on the PowerPC system. For FFT lengths 512, 1024, and 2048, we plot performance versus overlap percentages, for both zero-copy and manual copy. For any given FFT length there is a visible performance gain in the zero-copy version versus the manual copy version.

It is significant to point out the absolute performance that was measured. The benchmark shows that the processor is operating at 4 to 6 GFLOPS, which is 20 to 30 percent of the theoretical peak. This is about 2 floating-point operations per clock cycle. It is also interesting to note that in each case, the performance increases nearly linearly as the overlap increases. This is unsurprising, because the overlap data has already been read from main memory into cache. These particular FFTs are small enough to fit in cache, so the

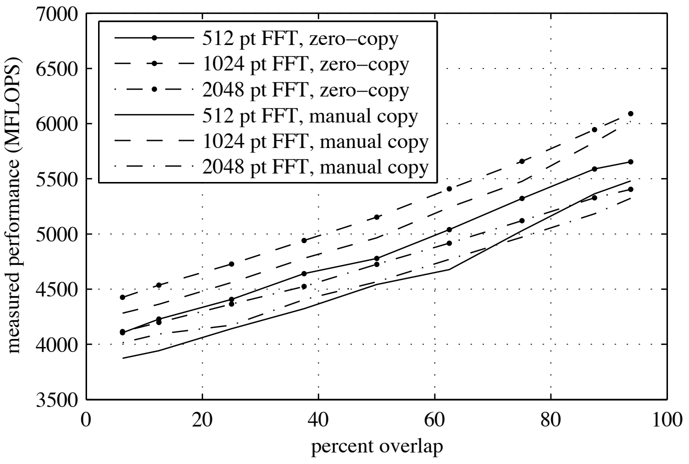

Fig. 3. Partial PowerPC performance results versus overlap for zero-copy and manual copy queues. Zero-copy queues have higher performance.

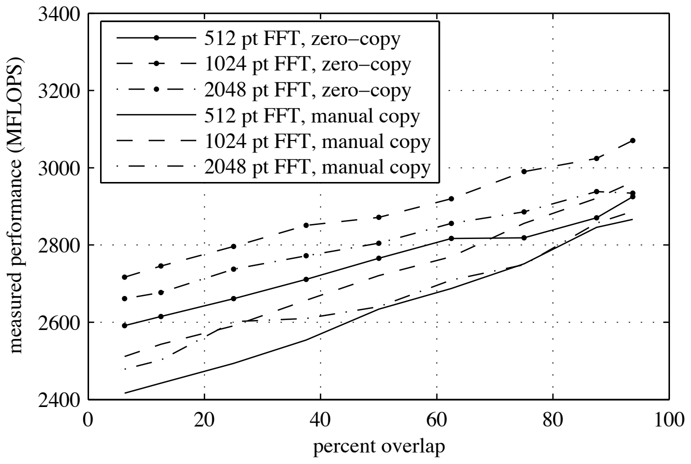

Fig. 5. Partial Opteron performance results versus overlap for zero-copy and manual copy queues. Zero-copy queues have higher performance.

overlap percentage is equivalent to the percentage of cache pre-warming.

Fig. 4 shows PowerPC results as percent performance improvement for zero-copy queues over the manual copy method, versus the FFT length. The percent performance improvement generally wanes as the FFT length grows. This is unsurprising because the execution time overhead for the manual copying is linear with the FFT length, but the FFT execution time grows super-linearly as shown in (1). In general, one would expect the percentage performance improvement to decrease as the size of the (non-overhead) workload increases.

B. RHEL4 on Opteron

Fig. 5 shows a subset of the performance results on the Opteron system. Again we plot performance versus overlap percentages for FFT lengths 512, 1024, and 2048, for both zero-copy and manual copy. Again there is a visible performance gain for zero-copy versus the manual copy version.

On the Opteron, the benchmark is operating at just under 3 GFLOPS, which is about 20 percent of the theoretical peak. This is about 1.7 floating-point operations per clock cycle. Again, performance increases nearly linearly as the overlap increases. Note that FFTW is not included as a package on RHEL4, so we built it from the source code package provided by the Fedora Extras repository. The package as provided performed at only about 1.5 GFLOPS. We achieved the presented performance by modifying the build flags to optimize for the Opteron.

Fig. 6 shows Opteron results as percent performance improvement for zero-copy queues over the manual copy method, versus the FFT length. When compared to the PowerPC results in Fig. 4, the shortest FFTs do not have as much performance improvement, and the middle FFTs do not dip as low. However, the general trend is similar.

We have shown that on both the PowerPC and the Opteron, zero-copy queues can significantly increase performance over manual copies for a wide range of potential workloads.

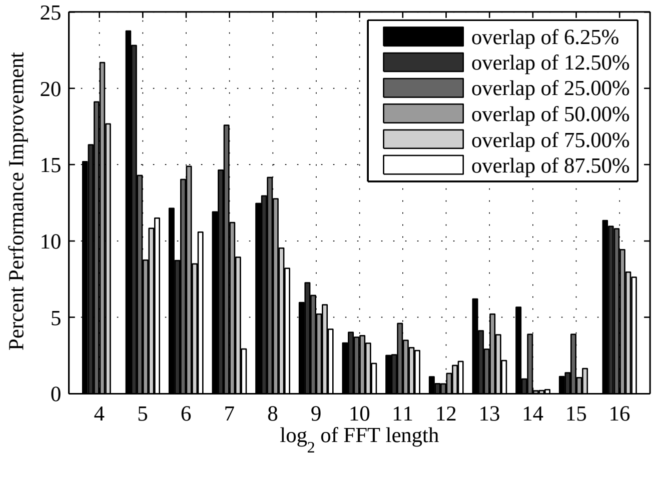

Fig. 4. Percent performance increase on PowerPC for zero-copy queues over manual copy.

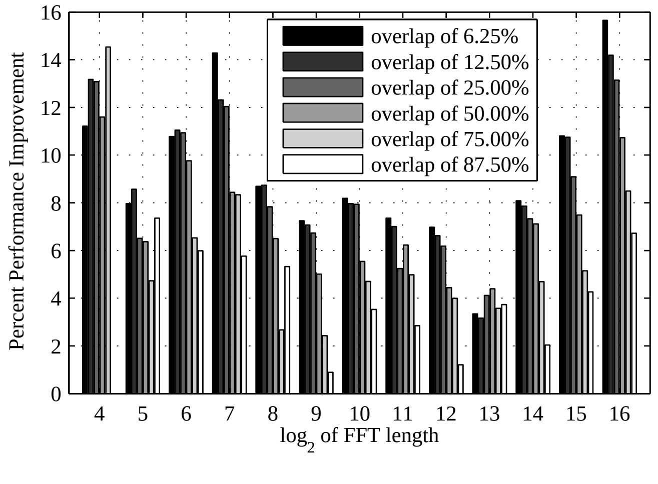

Fig. 6. Percent performance increase on Opteron for zero-copy queues over manual copy.

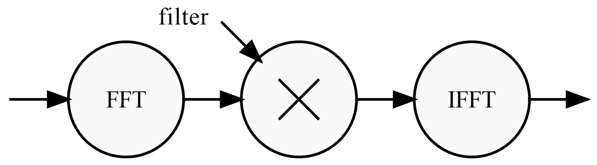

Fig. 7. A CPN program for a frequency-domain FIR filter. Nodes can execute concurrently and have zero-copy queues connecting them.

VI. Computational Process Networks

Zero-copy queues using the virtual memory manager came as a side-effect of other ongoing work: an implementation of Computational Process Networks (CPN) [7].

The Process Network (PN) model was introduced by Kahn [8] in 1974, and is a dataflow model in which concurrent processes communicate (only) over one-way FIFO queue channels. In a PN program graph, edges represent queue channels, and nodes represent processing elements. Process Networks naturally model functional parallelism, can model data parallelism, and are generally well-suited for modeling signal processing systems. Kahn showed that a PN program is *determinate* – the results produced on all queues are the same for every possible execution order, including concurrent execution. PN has therefore found some use in parallel and distributed signal processing systems. In the example program of Fig. 7, each node could be executed concurrently, and have zero-copy queues connecting them.

Computational Process Networks (CPN) is our extension of the PN model, and adds the concept of firing thresholds similar to Computation Graphs [9]. With firing thresholds, nodes may access more queue data than they consume, meaning that they can operate on continuous, overlapping streams of data without the need to perform manual data copying for overlap management. In fact, nodes can operate directly from queue memory, which further reduces copying and also decouples computation from communication.

By reducing the overhead of copying, zero-copy queues reduce overall system memory bandwidth usage, and can therefore increase scalability of a CPN program on a symmetric-multiprocessing system. Although one can also reduce relative copying overhead by increasing the processing size at each node, there are advantages to using smaller granularity. Smaller nodes increase the potential concurrency in the system, and also increase the design space for mapping the system to a parallel implementation.

We have released a high-performance CPN framework implementation using POSIX threads with source code available [7]. This release also contains the source code for the zero-copy queues presented in this paper. The presented zero-copy queue approach has been used extensively in our implementation of CPN, and has been successfully fielded in our real-time sonar beamformer on a workstation [10].

VII. Conclusion

For high-throughput signal and image processing systems, zero-copy queues can eliminate the copies necessary for overlap management, and can make a significant, measurable difference. We demonstrated this for a wide workload on multiple platforms.

There are some issues to be aware of while using the VMM in this manner. In using zero-copy queues for various applications, we have encountered a performance problem with respect to N-way set-associative caches on the PowerPC. The base address of a zero-copy queues is always page aligned. This had the base addresses from many different queues hitting the same cache set. We resolved this issue (achieving better cache utilization and therefore overall performance) by staggering data in the different queues such that they hit different cache sets.

We also note that extreme overuse of zero-copy queues is likely to give diminishing returns. If too many maps are made in a single process and the virtual memory translation lookaside buffer (TLB) begins to overflow, the penalty for recovering from TLB misses will likely exceed the performance gained by reducing the overhead of data copying. That said, we have used several dozen zero-copy queues in processes and found them to perform admiribly.

The benefit of using zero-copy queues depends on the workload, but as data throughput grows or the computation size gets smaller, the relative overhead due to copying for overlap management typically becomes more significant. The CPU time saved from copying will free the processor for other operations, ultimately making a more efficient implementation. Such high-throughput workloads are the type that Computational Process Networks [7] were designed to handle.

References

[1] W. Chen, H. Reekie, S. Bhave, and E. Lee, "Native Signal Processing on the UltraSPARC in the Ptolemy Environment," in *Proc. IEEE Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, Nov. 1996, pp. 1368–1372.

[2] "Intel math kernel library (MKL) web page," http://www.intel.com/software/products/mkl/.

[3] "Vector signal and image processing library (VSIPL) web page," http://www.vsipl.org/.

[4] J. Eyre and J. Bier, "DSP processors hit the mainstream," *IEEE Computer*, vol. 31, no. 8, pp. 51–59, 1998.

[5] "IEEE POSIX standards web page," http://standards.ieee.org/catalog/olis/posix.html.

[6] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proc. of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[7] "Computational process networks (CPN) web page," http://www.ece.utexas.edu/~allen/CPN/.

[8] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing*, pp. 471–475, Aug. 1974.

[9] R. M. Karp and R. E. Miller, "Properties of a model for parallel computations: Determinacy, termination, queueing," *SIAM Journal*, vol. 14, pp. 1390–1411, Nov. 1966.

[10] G. E. Allen and B. L. Evans, "Real-time sonar beamforming on workstations using process networks and POSIX threads," *IEEE Trans. on Signal Processing*, pp. 921–926, Mar. 2000.