Optimization of Vertical and Horizontal Beamforming Kernels on the PowerPC G4 Processor with AltiVec Technology

EE382C: Embedded Software Systems
Final Report

David Brunke
Young Cho
Applied Research Laboratories:
The University of Texas at Austin

Abstract

Real-time digital sonar beamforming is a computationally intensive algorithm that has been implemented in the past primarily in custom embedded hardware. With recent advancements in native signal processing extensions for general-purpose processors, it is possible to implement sonar beamforming using off-the-shelf hardware. A previous implementation on a Sun UltraSPARC multiprocessor suggests a very promising platform for transitioning such applications to general-purpose systems. In this paper, we continue the previous implementation by modifying the beamforming kernels to use AltiVec, a new native signal processing extension from PowerPC. AltiVec is a Single Instruction Multiple Data (SIMD) architecture capable of executing up to four 32-bit floating-point multiply and accumulate (MAC) operations per instruction. Although our benchmarks show some performance increase using AltiVec, there are some problems that prevented us from obtaining better results. First of all, the compiler, GCC, is not properly optimized. Second, a cache architectural problem caused the cache to be used inefficiently. Third, there is overhead in aligning the data properly into the registers. Overall, we believe that despite the lack of a significant speedup, AltiVec is promising for these types of applications.

1.0 Introduction

In last few decades, the Digital Signal Processor (DSP) market has grown substantially to meet the demand of the high performance signal processing community. Such growth in the signal processing market has allowed the general computing communities to incorporate the technology into their applications. As a result, general-purpose processors began to embed signal processing architectures into their own processing cores to provide economic system solutions for computationally intensive applications [1].

Real-time digital sonar beamforming is one such application once only feasible on custom hardware that can now be successfully implemented on commercial, off-the-shelf computers with native signal processing extensions. One recent implementation uses a commercial general-purpose 8-way symmetric multiprocessor (SMP) workstation from Sun Microsystems [2]. The beamforming kernels exploit the inherent data parallelism by using Single Instruction Multiple Data (SIMD) arithmetic operations available in the Visual Instruction Set (VIS) extensions to the UltraSPARC processor. By using a sixteen 333-MHz UltraSPARC Enterprise server, a real-time beamformer delivering 4 GFLOPS on 160 MB/s of streaming data was realized.

The goal of our research is to further explore the effectiveness of the embedded extensions by optimizing and assessing the performance of beamforming kernels using AltiVec from PowerPC, which is one of the newest native signal processing instruction sets. We also plan to analyze the results obtained from the two embedded signal processing extensions to assess the architectural advantages and disadvantages.

2.0 Native Signal Processing Extensions

Many high performance embedded applications are programmed on systems with a few general-purpose processors as system controllers with a larger number (possibly hundreds) of specialized DSPs to perform scientific calculations. However, this type of system has many disadvantages due to different programming platforms and unequal performance advances in the two separate technologies. Therefore, many manufacturers of high performance general-purpose processors are integrating sets of native signal processing instructions onto their processor cores to offer solutions requiring fewer processors.

2.1 UltraSPARC VIS

The Visual Instruction Set (VIS) is a set of signal processing instructions based on the SIMD architecture. The floating-point data of the UltraSPARC processor core is enhanced with graphics integer units to support VIS. With 50 new CPU and 64-bit registers, VIS can perform integer operations on multiple words with a single instruction. Thus, VIS can achieve up to four times speedup with 8-bit by 16-bit fixed-point multiplication using the SIMD arithmetic logic [4].

2.2 PowerPC AltiVec

The AltiVec vector unit is sectioned into a separate sub-unit of the processor as are the floating-point and integer units. As shown in Fig. 1, the vector unit has its own 32 by 128-bit wide register file for use with 150 new floating point and integer SIMD instructions. It allows execution of up to four 32-bit floating point MAC operations per instruction [5]. AltiVec is potentially a much more powerful signal processing extension than VIS due to its greater logic resources.

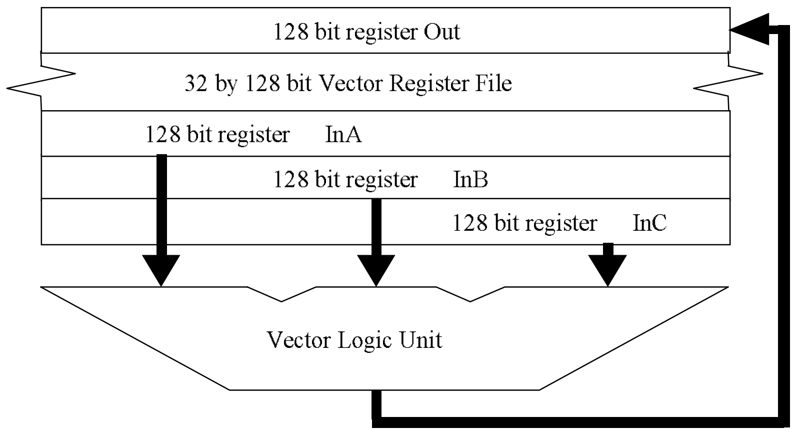


Fig. 1: Block Diagram of PowerPC AltiVec Unit Architecture

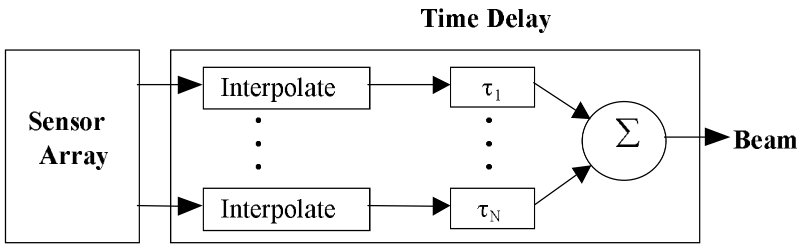


Fig. 2: Digital Interpolation Beamformer

3.0 Beamforming Algorithm

Conventional sonar beamformers use the signals collected from sensor elements to determine from what direction the sonar signal returns after deflecting off of an object. This conventional horizontal time-domain beamforming algorithm consists of appropriately delaying and summing the weighted outputs of an array of sensor elements. The weighting of the sensor outputs helps to improve the spatial response [3].

The problem with this conventional approach is that it requires a sample rate that is several times the Nyquist rate for adequate time delay resolution. This is undesirable because it requires additional bandwidth for the overall system. A practical solution employs digital interpolation with Finite Impulse Response (FIR) interpolation filters to achieve a satisfactory time delay resolution [3]. This solution is shown in Fig. 2. Analog data is sampled at a given sampling interval, and then followed by interpolation, time delay, and summation.

3.1 Previous Implementation

The previous implementation we are building upon adds vertical beamforming to the approach in [3] to enable projection of a 3-D underwater image [2]. In addition, the interpolation for the horizontal beamforming kernel is simplified by using a two-point FIR filter for the digital interpolation. A two-point FIR filter is possible without a critical loss of resolution because the sampling rate is set to two times the Nyquist sampling rate. The overall system description is shown in Fig. 3.

The vertical beamformer computes three sets of data, each of which is sent to a horizontal beamformer. Thus, three dot products are computed with each column of 10 vertical transducers (staves) and three coefficient vectors; each contributes to the vertical resolution. In the benchmark results, the vertical kernel using UltraSPARC VIS (205 MFLOPS) almost triples the performance given by the non-VIS integer implementation (71 MFLOPS).

4.0 New Implementation

Using recent releases of G4 PowerPC processors with AltiVec and the AltiVec enabled C compiler, we confirm the evaluations assessed with various simulators [7]. We evaluate the results of the vertical and horizontal beamforming kernels programmed with AltiVec on the G4 processor in a real-time working environment.

4.1 Platform

We implement the kernels in the Linux operating system using the GCC compiler provided to us by Motorola, which can compile code with AltiVec instructions in C/C++. Due to the structural differences between the VIS and AltiVec instruction architectures, we change the format of the data to fully utilize the AltiVec extensions for the kernels [5,8,9].

4.2 Process Network Programming Model

Under the process network domain, the output of the vertical beamformer is a queue. In the previous implementation, the queue was allocated contiguously with each set of stave samples addressed

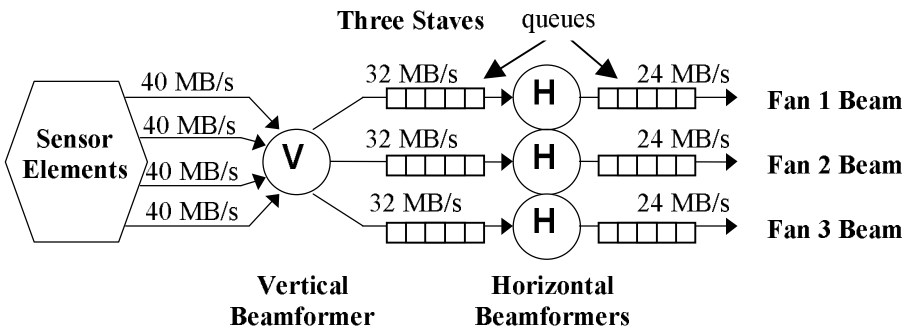


Figure 3: Block Diagram of the 3-D sonar beamformer

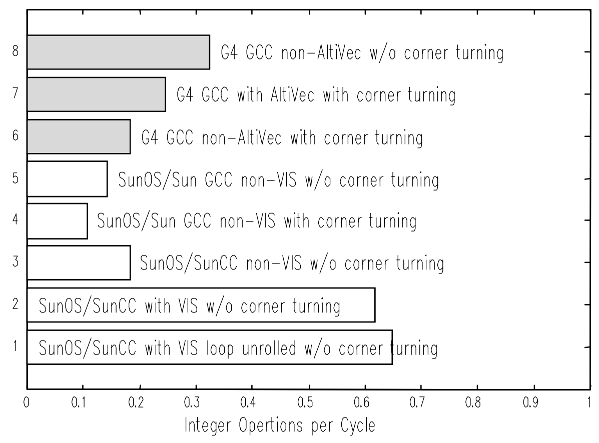
consecutively according to the time at which they were sampled. Since the two-point interpolation in the horizontal beamformer uses the samples collected from a stave two consecutive times, the original queue structure would require the sample points to be referenced with two different addresses distanced by the size of a stave. By transposing the queue, the samples for each stave become contiguous according to the sampling times. Such an arrangement would allow each vector load instruction to load up to four samples required for the calculation at once. We refer to the transposing of the queue as corner turning.

Even though the corner turning may benefit the vector load, it introduces a few problems. First, the index calculation becomes more complex, requiring the queue size to be added or subtracted to address the adjacent set of stave samples. Second, due to the necessary arrangement of the queue in memory, it is restricted from growing dynamically. The effect of the first problem is inevitable without significant change in our data structure. However, the effect of the second problem disappears when we allocate the queue to be large enough to hold all the data at one time. It is possible to allocate such a queue size due to the well-known behavior of the real-time system with maximum input data bandwidth.

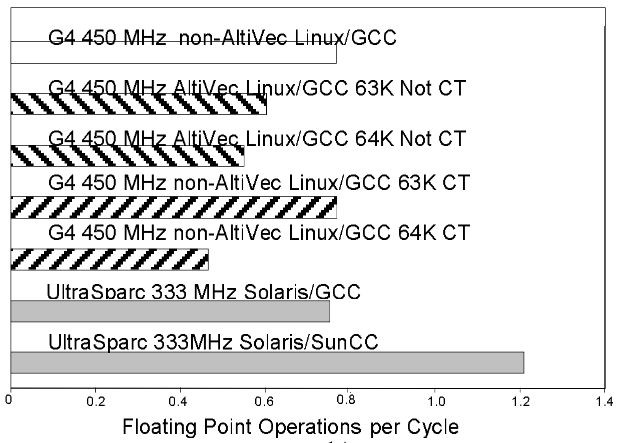
4.3 Beamforming Kernels

In the vertical beamformer, we compute three dot products, each of which must be converted to floating-point before it is sent off to the horizontal beamformer. We perform the dot product of eight 16-bit elements by using a vector multiply accumulate followed by a vector sum operation. To accommodate the horizontal beamforming kernel, the output is stored into a corner turned queue. Thus, the resulting code for the corner turning increased the necessary calculation for the queue address. We also add loop unrolling to the kernel to improve the performance, which requires using the vector permute operation to rearrange the data appropriately.

The horizontal beamformer then loads the corner turned samples using the vector load operation. Although the number of loads are reduced due to the AltiVec vector instructions, the alignment



a)



b)

Fig 4: Optimum Kernel Performance Graphs

constraints for the instructions require a greater number of vector permutations to be done on the loaded samples. Once the correct samples are arranged in the vector register, a powerful floating-point vector multiply and accumulate instruction is executed on four samples at once allowing up to eight times the speedup from the standard floating point instructions. The results are once again placed in a corner turned output queue of the horizontal beamformer.

5.0 Results

Fig. 4a shows the performance for the vertical kernel, where the previous implementation with VIS (as shown in white) is also shown as a comparison. Although we did implement loop unrolling in the kernel, we were unable to verify these results, so we do not show them here. However, our preliminary results indicate a significant speedup with loop unrolling with AltiVec implementations over the VIS in vertical kernel.

For the horizontal kernel, performance is measured from several versions of loop unrolled kernels. Each versions of kernels are written to evaluate the effects of different types of queues, compilers, and platforms. Fig. 4b shows the optimal performance measurements in terms of operations per instruction cycle of each kernel. Due to different optimizations used in compilers and the processor architecture, each version of the kernels gave different optimal number of loop unrolling iterations for the generated code.

5.1 Analysis

As performance results indicate, compiler plays a large role in the performance. GCC is a generic, non-commercial version of the compiler that does not consider many potential architectural advantages of each processor. Even with optimization and architectural tuning flags, GCC compiled code performed as much as 50% slower than the SunCC compiled code on the same machine.

While adding the VIS to the vertical kernel increased the performance by two to three folds, adding the AltiVec instructions to the kernels decreased the speed. Although such results were not expected, the most likely source of the explanation based on the measurement is that the maturity of the compilers. While GCC is not tuned for any specific processor architecture, SunCC from Sun is very successful in generating code that efficiently utilizes the advantages of the UltraSPARC processor. In fact, the decrease in performance for AltiVec is most likely due to lack of optimization in the compiler to architecture specific nature of the native signal processing extensions.

The results show that there also was slight decrease in performance in the corner turned version of the kernels. One plausible, yet unverified, reason for this slow down is because of the extra pointer arithmetic when determining the address of the output. In addition, as in the case of the horizontal kernel, there could be a problem with the utilization of the cache. The queue size may effect whether or not there are many cache misses.

Lastly, about 20% difference in performance was evident when the memory allocated for the queue in PowerPC was different. The result show that 64 KB aligned queue performed poorly compared to the 63KB aligned queue in PowerPC, whereas UltraSPARC performed best with the 64KB align queue. This is indication of cache architectural differences between UltraSPARC and PowerPC. It is also depended on the compilers' decision in how to use the available resources.

6.0 Conclusion

This project benchmarks a computationally intensive algorithm, sonar beamforming, on the AltiVec native signal-processing technology. We found several issues that kept us from getting the speedup we expected. First of all, the compiler, GCC, is not properly optimized. It did not provide the kind of performance we need to fully exploit the AltiVec architecture. Second, the cache architectural problem caused the cache to be used inefficiently. This was seen with the corner turning for the vertical kernel, as there was most likely some performance loss because of the cache misses. Third, we verify that with the horizontal kernel there is overhead in aligning the data properly into the registers, which is done with the vector permute instruction. Efficient methods to deal with this issue are needed to yield a performance boost. Overall, we believe that despite the lack of a significant speedup, AltiVec is promising for these types of applications.

7.0 References

[1] J. Bier, "DSP on General Purpose Processors," *MicroDesign Resources Dinner Meeting Slides, Berkeley Design Technology, Inc.*, Jan. 1997.

[2] G. E. Allen and B. L. Evans, "Real-Time Sonar Beamforming on Workstations Using Process Networks and POSIX Threads." *IEEE Trans. on Signal Processing*, vol. 48, no. 3, pp. 921-926, March 2000.

[3] R. G. Pridham and R. A. Mucci, "A Novel Approach to Digital Beamforming." *Journal Acoustical Society of America*, vol. 63, no. 2, pp. 425-434, Feb. 1978.

[4] Sun Microsystems, "VIS Instruction Set User's Manual." *http://solutions.sun.com/embedded/databook/pdf/manuals/805-1394-01.pdf.*

[5] AltiVec Programming Environment/Interface Manual, *Motorola Inc.*, 1998.

[6] G. E. Allen, B. L. Evans, and L. K. John, "Real-Time High-Throughput Sonar Beamforming Kernels Using Native Signal Processing and Memory Latency Hiding Techniques", *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 25-28, 1999, vol. I, pp. 137-141, Pacific Grove, CA.

[7] H. Nguyen and L. K. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology." *Proc. ACM Int. Conf. on Supercomputing*, June 20 - 25, 1999, pp. 11-20, Rhodes Greece.

[8] Motorola, "AltiVec Technology." *http://www.mot.com/AltiVec.*

[9] AltiVec Information Source. *http://www.altivec.org.*

[10] J. D. Allen and D. E. Schimmel, "Issues in the Design of High Performance SIMD Architectures." *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, pp. 828-839, Aug. 1996.