Implementations of Signal Processing Kernels using Stream Virtual Machine for Raw Processor

Jinwoo Suh, Stephen P. Crago, Dong-In Kang, and Janice O. McMahon
University of Southern California – Information Sciences Institute
3811 N. Fairfax Drive, Suite 200, Arlington, VA 22203
{jsuh, crago, dkang, jmcmahon}@isi.edu

Introduction

Stream processing exploits the properties of the stream applications such as parallelism and regularity. DARPA's Polymorphous Computing Architectures (PCA) program is developing both hardware and software that support stream (and thread) processing with a two-level compiler infrastructure. The Morphware Forum was formed to develop standard software interfaces to promote common interfaces and software reuse [2]. The Stream Virtual Machine (SVM) framework [1] is the intermediate language between the high-level compiler and low-level compiler for streaming applications. We implemented a library-based version of the SVM for the Raw architecture. The Raw SVM library in conjunction with Raw C compiler comprises a functional low-level SVM compiler. We implemented a matrix multiplication and FIR bank kernel using the library and optimization manual optimization to identify performance issues surrounding the SVM.

Stream Virtual Machine and Raw Processor

The SVM approach consists of a High Level Compiler (HLC), Stream Virtual Machine (SVM), and Low Level Compiler (LLC). Input to the HLC is a stream program. The HLC is responsible for parallelism detection, load balancing, coarse-grain scheduling of stream computations, and memory management for streaming data [1]. The output code from the HLC is based on the SVM API. The SVM code is then compiled using the LLC to generate binary code for a target platform. The LLC is responsible for software pipelining, detailed routing of data items, management of instruction memory, and interfacing between stream processors and control processors [1]. With this approach, when a new language or a new architecture is introduced, it can leverage existing software in the HLC or LLC.

The authors gratefully acknowledge the extraordinary support of the MIT Raw team for the use of their compilers, simulators, Raw chip, and their generous help.
The authors acknowledge Morphware Forum [2] for the Streaming Virtual Machine standard upon which this implementation was performed.
This effort was sponsored by Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Laboratory, USAF, under agreement number F30602-01-C-0171. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, or the U.S. Government.

Raw is a research processor chip implemented at MIT [4]. The current Raw implementation contains 16 tiles on a chip connected by a very low latency 2-D mesh network through switch processors.

Signal Processing Kernels

Matrix multiplication, $C = AB$, calculates an output matrix C from A and B, where A, B, and C are matrices. We implemented a systolic matrix multiplication on Raw. The Raw tiles are utilized as shown in Figure 1. The input matrix A travels from left to right, and the input matrix B matrix travels from top to bottom. The result matrix C is sent to the tiles on the right.

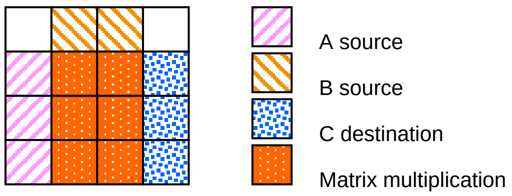

Figure 1: Matrix multiplication

The FIR bank application kernel is part of the kernel benchmark suite [2] specified by Lincoln Laboratory the PCA program. The FIR bank implements a set of M FIR filters and each FIR filter m, $m \in \{0, 1, \ldots, M\text{-}1\}$, has a set of impulse response coefficients $wm[k]$, $k \in \{0, \ldots K\text{-}1\}$. It is mathematically specified as:

$$y_m[i] = \sum_{k=0}^{K-1} x_m[i-k]w_m[k], \text{ for } i = 0,1,\ldots,N-1 \cdot$$

The filter computations are distributed over Raw tiles such that each processor tile has M/T Filters, where T is the number of tiles. Each tile computes its own filters. Therefore, there is no communications between tiles. The FIR is implemented in the frequency domain because the frequency domain computation is more computational efficient than the time domain for data sizes. Therefore the kernel requires an FFT operation, a complex product, and an IFFT operation. The parameters for the results reported in this paper are: K=12, N=32, and the length of the input data is 1024 complex elements.

Implementation Results

We started with C source code compiled using the HLC [3]. The output from the HLC was then compiled using the gcc-based Raw compiler in conjunction with the SVM library on Raw. The code was then executed on a Raw prototype board. This experiment is the first instance of Morphware software executing on PCA hardware and verified functionality of the SVM framework on Raw. However, the performance using the full path is not yet meaningful since the HLC is still in an early stage of development and focus has been on functionality.

We also implemented the FIR bank in optimized SVM code to allow us to understand performance issues of the SVM library. Furthermore, we performed several manual optimizations on the resulting code. For example, we statically assigned communication streams to specific networks on Raw to eliminate software overhead for managing multiple streams on a network. Another optimization is using network ports as direct operands, which is allowed on Raw because network ports are mapped to registers names to eliminate load and store instructions.

The performance results for matrix multiplication are shown in Figure 2. The x-axis is the number of words per communication packet and the y-axis is number of cycles per multiplication-addition pair. Thus, the lower bound is two cycles since each multiplication and addition needs one cycle to execute.

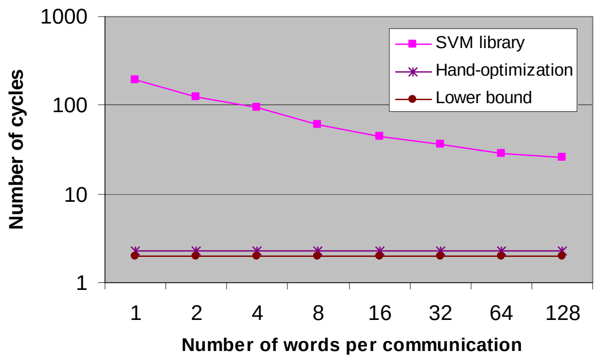

Figure 2. Matrix multiplication results

The curve named "SVM Library" shows the performance when a full, general implementation of the SVM API is used. The curve shows that the initial cost of the communication using library approach can be amortized over a long sequence of data, although significant overhead is still incurred compared to an optimization application implementation. The curve named "Hand-optimization" shows the performance when several optimizations were applied. The optimized results show that it takes only about 10% more overhead than the theoretical lower bound.

The implementation results for the FIR filter are shown in Figure 3 and Figure 4. In Figure 3, Lower bound denotes the number of cycles per floating point operation per tile when only floating-point operations are considered. Since each tile can compute one floating-point operation per cycle, the lower bound is 1. The "practical" lower bound denotes the number of cycles per floating point operations per tile when load/store operations are considered as well as the number of floating point operations. Note that the load/store operations were inevitable in our FIR bank implementation that uses Radix-4 FFT. Our results show that the hand-optimized results are very close to the "practical" lower bound with only about 10% overhead.

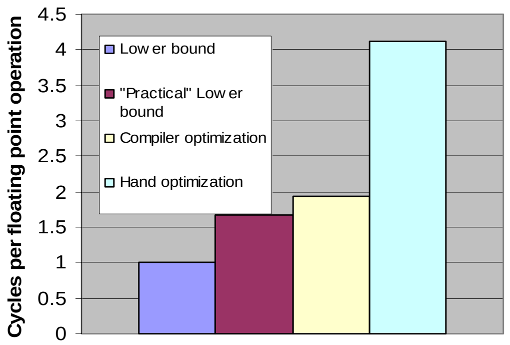

Figure 3. Throughput results for FIR bank (data from cache)

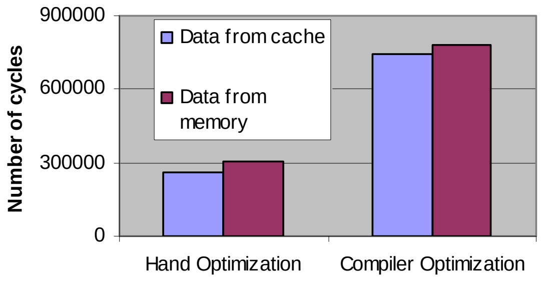

Figure 4. Latency results for FIR bank

Figure 4 shows the effect of accessing data from memory. It takes about 16% additional cycles when data is accessed from main memory instead of being pre-loaded into the local memory.

Conclusion

The authors have presented implementation results for the SVM library for the Raw processor. The library approach enables a full path from a high-level language to the processor and a quick validation of the SVM API.

We applied several manual optimizations to understand the performance issues in SVM framework and were able to obtain the performance very close to the theoretical peak performance of the kernels. We expect similar performance improvement can be obtained using optimizations when the HLC and LLC are mature enough.

References

[1] P. Mattson, W. Thies, L. Hammond, M.V. Raytheon, "Streaming Virtual Machine Specification," Version 1.0, http://www.morphware.org , July 2004.

[2] Morphware Forum, http://www.morphware.org, 2005.

[3] Reservoir Labs., "R-Stream - Streaming Compiler," http://www.reservoir.com/r-stream.php, 2005.

[4] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," ISCA, February 2003.