Native Signal Processing With Altivec In the Ptolemy Environment

Ken Aponte and Ken Logan

March 8, 2000

Abstract

In the near future, media processing (i.e. creation, encoding/decoding, processing, display, and communication of digital multimedia such as images, audio, video, and graphics) is expected to become one of the dominant computing workloads [3]. Native signal processing (NSP) extensions for various general-purpose processor (GPP) architectures have been introduced in an effort to make media processing on workstations and personal computers easier. Due to incredible progress in microprocessor performance, it has become possible to perform many media processing tasks solely on general-purpose processors augmented with NSP extensions, even tasks that historically required special purpose hardware or specialized digital signal processors (DSP).

In [8] it was found that using MMXä to enhance DSP and multimedia applications was a 'difficult chore' and in [5] the level of difficulty in programming with Sun's VISä extensions was compared to that of DSP programming. Regardless of the NSP extension being used, usability problems arise because the legacy software development environment for general-purpose processors is not easily amenable to the single instruction multiple data (SIMD) semantics used in the NSP extensions. Our objective in this survey is to summarize the NSP extensions available, explain problems that software developers frequently encounter using the extensions, and discuss a possible strategy to solve this problem using code-generation facilities built into the prototyping environment called Ptolemy [4].

1 Native Signal Processing Architectures

Many of the modern general-purpose processor architectures now include NSP extensions. PA-RISCä was the first instruction set architecture to introduce multimedia extensions, MAX-1ä (Multimedia Acceleration extensions), in products introduced in January 1994 [7]. Following MAX-1, were the Visual Instruction Set (VIS) for Sun workstations as well as MAX-2ä for the 64 bit PA-RISC 2.0 processor [7]. In 1996, extensions for Silicon Graphics MIPSä (MDMXä) and Digital Alpha (MVIä) were announced [9]. Also, in 1996 Intel announced MMX (Multimedia extensions), which shipped in 1997. The Intel MMX extensions did not include any new floating-point data types, but later Intel announced SSEä (Streaming SIMD Extensions) and to fill the void. In May of 1998, Advanced Micro Devices shipped a processor implementing 3D-NOWä extensions, which were vector floating-point extensions to complement MMX. In 1998, Motorola announced its Altivecä technology for the PowerPCä architecture and in September of 1999, Motorola shipped the PowerPC MPC7400 (the first PowerPC to implement Altivec technology).

The workloads targeted by NSP extensions usually involve relatively simple computations that are performed repeatedly on a stream of data. It is often possible to concurrently operate on several samples in such streams of data. For this reason, all of the NSP extensions identified utilize SIMD (Single Instruction Multiple Data) semantics[1]. Almost all the NSP extensions use a 64-bit datapath; the HP MAX-1 uses a 32-bit datapath, while the PowerPC Altivec technology uses a 128-bit datapath [9]. The SIMD instructions operate on vectors that contain multiple 8, 16, or 32 bit elements. For a data stream composed of eight bit samples, an eight times speedup is possible using a SIMD instruction with a 64 bit register when the algorithm can be fully vectorized.

2 Problems Encountered Using NSP Extensions

The SIMD semantics common in NSP extensions require that 'packed' data types be used in programs. Currently, programmers must modify existing applications or write applications explicitly for a certain NSP extension. In [1, 5, and 8] it was stated that this requirement presents significant usage problems for NSP extensions. Since the new data-types and operations on them are not standardized, code utilizing them is not portable at the source level between different NSP extensions. Using libraries provided by the processor vendors doesn't remedy this portability problem, due to the fact that the library API's are also not standardized. Compiler support typically uses function inlining or macro calls in code segments that will benefit from using the NSP extensions [1]. Ideally, compilers of the future will be able to analyze and 'auto-vectorize' code to be optimized for a given NSP extension. This would make the new semantics invisible to the programmer, but such compilers do not currently exist [1]. A possible solution to this problem is finding an abstract representation that can efficiently be converted to software for an arbitrary NSP extension.

3 Code Generation for Native Signal Processing using Ptolemy

Ptolemy is a software environment for the simulation and prototyping of heterogeneous systems [6]. It provides the designer a means of integrating both software and hardware into a single contiguous environment. This, in turn, facilitates the system hardware-software partitioning and any re-synthesis of said partition at subsequent stages of the design cycle. This is accomplished in Ptolemy by providing object-oriented foundation that is un-restrictive in its definition thereby allowing new ‘domains’ to be easily defined. These domains represent different models of computation or, in our case, a target architecture using the code-generation facilities built into Ptolemy. Once implemented, these domains can be interwoven and manipulated. Thus, Ptolemy provides a heuristic approach to what usually amounts to, in a heterogeneous system, a very difficult problem.

Given a block diagram description of an algorithm, [The term] code generation refers to the synthesis of software corresponding to the algorithm [6]. The actors or stars we intend to develop will produce C source code including Motorola’s Altivecä extensions for the PowerPCä architecture.

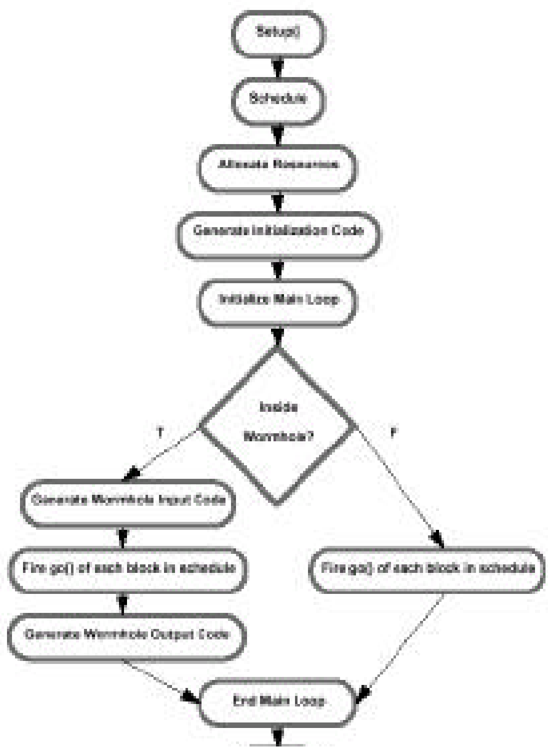
Setup()
Schedule
Allocate Resources
Generate Initialization Code
Initialize Main Loop
Inside Wormhole?
T
F
Generate Wormhole Input Code
Fire go() of each block in schedule
Fire go() of each block in schedule
Generate Wormhole Output Code
End Main Loop

Figure 1: Software Synthesis in the Ptolemy Environment

Ptolemy has a complete suite of base objects that facilitate the implementation of new code generation domains. New domains are derived from these objects in the same manner new objects are derived from base classes in C++. Once implemented, these new domains follow the semantics of the model of computation of the system in which they are placed by virtue of wormholes. A simpler approach is to develop code generation stars and targets in the CGC, or C-Code Generation Domain, built into Ptolemy. Targets in Ptolemy specify how the generated code within a domain will be collected, specifies and allocates resources and defines code necessary for proper initialization of the platform [2]. With these blocks and targets, a system is developed and compiled under a specific scheduling algorithm as shown in Figure 1.

These code generation techniques have been implemented on a few architectures previous to our work. Figure 2 at right shows one such implementation for the UltraSparc Visual Instruction Set (VIS). Note that primitive stars, VISAddSh or VISMpyDblSh (floating point add and multiply), are coded and then become the basis for larger blocks, such as, VISFIR (finite impulse response filter). The conversion blocks are necessary to pack and unpack the SIMD operands.

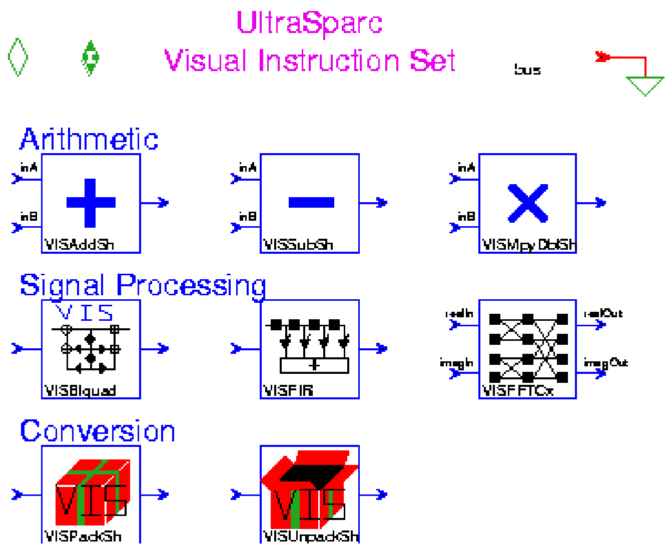

Figure 2: Example Actors for the UltraSparc NSP Extensions, VIS.

In [5], Lee and Chen, et al, used the code generation facilities in Ptolemy to implement the actors in Figure 2. They then demonstrated a real-time audio application running on an UltraSparc I. Similar work was conducted by Lee and Kalavade in [6] using the Motorola DSP56000 as the target architecture

The authors in both cases demonstrated the following:

- The advantages of actor and library re-use in the Ptolemy environment. Optimizing existing algorithms to support native code is a very difficult task. The existence of a library of primitive actors eases this transition.
- The flexibility of an abstract environment for the development of heterogeneous systems. Ptolemy provides a graphical interface for the description of systems. Stars represent code blocks. The designer need not know the internal workings of the star in order to explore the design space in a unified environment.
- The visible partitioning between the hardware and software of a system is readily visible. Ptolemy makes the optimization and re-synthesis of the partition easier by facilitating the manipulation of stars within the environment.

4 Altivec Technology Overview

We have chosen Motorola's Altivec technology as the NSP extension to use in our experimentation. Altivec is unique among the NSP extensions because it adds support for a separate 128-bit vector multimedia unit in the processor [3]. Altivec is an interesting NSP extension to study for several reasons. Firstly, there have been few Altivec papers published in the academic literature, probably due to Altivec's relatively recent availability as an NSP extension in the MPC7400 PowerPC processor. Secondly, the necessary tools for our planned experiments (Ptolemy, a compiler supporting the

Altivec C programming model, and a MPC7400 timing simulator) are publicly available. Lastly, Altivec technology possesses features that set it apart from the other NSP extensions.

Altivec is the only NSP extension that we identified through our research which offers thirty-two 128 bit wide dedicated vector registers. Unlike the Intel x86 compatible NSP extensions, there is no performance penalty associated with a 'context switch' to switch in or out of vector mode. In fact, it is possible to write code that uses the integer unit, vector unit, and floating point unit concurrently on a PowerPC processor that implements Altivec [10]. The vector registers provide 8-way parallelism for 16-bit signed and unsigned integers and 16-way parallelism for 8-bit signed and unsigned integers. Saturation arithmetic and a rich variety of instructions are included in the Altivec instruction set.

5 Conclusion

We have reviewed the NSP extensions currently available, studied the implications of the SIMD semantics inherent in all of the current NSP extensions and reviewed the code generation capabilities of the Ptolemy environment. In order to take advantage of the power inherent in Ptolemy, the authors plan to implement several Altivec primitives in Ptolemy in the manner described above. These primitives could eventually become the backbone of a larger heterogeneous system. There are several good primitive candidates for implementation in Ptolemy. First and foremost the authors will port those algorithms developed in Ptolemy for the UltraSparc [5] and possibly for the Motorola 56000 [6] to

Altivec. This will be accomplished by adding the ability to generate Altivec extended C code to existing Ptolemy stars.

Our main goal will be to evaluate the effectiveness of the Altivec NSP extension for a few common signal processing kernels and/or applications in a scenario that involves no hand-written assembly code. Code written for the NSP extensions is non-portable when written in assembly. If vector oriented C code (using a standardized C syntax) could be used to describe an algorithm and then compiled into efficient code, then some signal processing algorithms could be written to be portable at the source level. The authors don’t believe that vector oriented C code will meet the performance demands of NSP in all cases, but that vector oriented C code is a good alternative, whose performance is worthy of evaluation.

In addition to our main goal of evaluating effectiveness of the Altivec NSP extension on code generated by a C compiler, we will do our work in the Ptolemy environment. We have discussed the advantages of using the code generation domain of Ptolemy. As a means for evaluating performance differences between scalar and vector implementations of the algorithms we study, we will use a publicly available and highly accurate timing simulator for the MPC7400.

References

1. T.M. Conte, P.K. Dubey, M.D. Jennings, R.B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song, A. Wolfe, “Challenges to Combining General-Purpose and Multimedia Processors,” IEEE Computer, Dec. 1997, vol.30, no.12 p.33-7.

2. J.L. Pino, S. Ha, et al., “Software Synthesis for DSP Using Ptolemy”, Journal of VLSI Signal Processing, 1995, Vol. 9, pp. 7-21.

3. P. Ranganathan, S. Adve, N.P. Jouppi, “Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions,” Proc. of the 26th International Symposium on Computer Architecture, 1999, pp.124-135

4. Buck, J, Ha, S., et al., “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” *International Journal of Computer Simulation* special issue on ``Simulation Software Development”, 1994, vol. 4, pp. 155-182.

5. W. Chen, H.J. Reekie, S. Bhave, E.A. Lee, A. Singh, “Native Signal Processing on the Ultrasparc in the Ptolemy Environment,” Conference Record of Thirtieth Asilomar Conference on Signals, Systems and Computers, 1997, Vol. 2, pp. 1368-72.

6. A. Kalavade and E. Lee, “A Hardware-Software Codesign Methodology for DSP Applications,” IEEE Design and Test of Computers, Sep 1993, Vol. 103, pp. 16-28.

7. R.B. Lee, “Multimedia Extensions for General-Purpose Processors,” IEEE Workshop on Signal Processing Systems, 1997, pp. 9-23.

8. R. Bhargava, L.K. John, B.L. Evans, R. Radhakrishnan, “Evaluating MMX Technology using DSP and Multimedia Applications,” Proc. 31st Annual ACM/IEEE International Symposium on Microarchitecture, 1998, pp.37-46.

9. C.G. Lee, M.G. Stoodley, “Simple Vector Microprocessors for Multimedia Applications,” Proc. of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, 1998, pp. 25-36.

10. J. Tyler, J. Lent, A. Mather, Huy Nguyen, “Altivec: Bringing Vector Technology to the PowerPC Processor Family,” 1999 IEEE International Performance, Computing and Communications Conference, 1999, pp. 437-44.