The Impact of Programming Difficulty on Hardware Obsolescence

James Steed (jsteed@gedae.com), William Lundgren (wlundgren@gedae.com), Kerry Barnes (kbarnes@gedae.com)
Gedae, Inc., 1247 N. Church St., Suite 5, Moorestown, NJ 08057

Programming a real time system based on multiple Digital Signal Processors (DSPs) is a difficult task. Few programmers can fully harness the power of a DSP because it requires specialized knowledge of the processor's architecture and assembly language. To compensate for this lack of specialized knowledge, vendors and third parties provide optimized vector libraries which implement a select group of algorithms as well as compilers that optimize specific code constructs. Programs rely on this middleware to be broad and flexible enough to allow their applications to meet real time requirements. Processing data in real time often requires high amounts of parallelism. Programming large projects across multiple processors requires much foresight in effectively partitioning the work and ensuring deadlock and other runtime issues are avoided once all the parts of the application are brought together.

Since the early 1990s, boards of multiple DSP chips have been one of the leading target architectures in real time systems. A series of architectures and instruction sets have followed, from SHARCs to Power-PCs to AltiVec Power-PCs to TigerSHARCs. Each successive generation of hardware provided a small increase in clock speed and more processors on a single board, along with an adjustment to the architecture or instruction set that usually made legacy code nonportable. While these platforms provided a stable framework to address the issues of software development, few have addressed these two issues – how to affectively program a wide range of algorithms for a processor without intimately knowing its architecture and assembly code and how to program multiple processors in a manner that is not fraught with risk. Without addressing these two issues, programs become tied to a certain distribution of processing across a certain number of processors, with little hope of being able to improve that distribution or better use the processor's power if problems arise during development.

Increasingly, field programmable gate arrays (FPGAs) are being used alongside DSPs as a method for meeting high data flow requirements through massive parallelism, and multicore processors (such as the Cell processor) are being used in place of DSPs because they provide parallelism on a single chip. The increased use of these two types of processors does not present new problems. Rather, it heightens these longstanding problems that developers have long tried to ignore. Processors like the SHARC offered high efficiency years ago, but the difficulty of programming that architecture in assembly did not allow for that efficiency to be fully utilized. Because the problem was addressed largely through middleware, programmers could only address optimizing their applications through a limited toolbox instead of attacking their problems with their intelligence and creativity. The Cell processor offers a vector board of chips which are a vector of processors that use vector ALUs. This hierarchy of parallelism can be cloaked with middleware, but doing so increases the chances it too will be replaced by a new architecture before its full power is harnessed.

While many have ignored the root issues of implementing parallel digital systems, several tools have been developed to address these issues. Gedae is an integrated design environment for software development on boards of DSPs or distributed networks (e.g., Linux clusters). Gedae greatly simplifies the task of software development on these systems by automatically implementing the distributed control needed to run an application on a multiprocessor system (such as the mode control construct shown in Figure 1). Gedae is able to provide efficient code through a three-pronged approach: 1) utilize optimized vector libraries for common algorithms like addition and FFTs, 2) create compiler-like optimizations through knowledge of the target architecture, and 3) intelligently preplan the execution of the application so that order of execution, memory management, communication patterns, and other issues are determined as much as possible before runtime. Through the use of tools like Gedae, we can both slow down hardware obsolescence by making current hardware more programmable, thus increasing utilization, as well as more rapidly transition to emerging technologies by handling that transition through the tool.

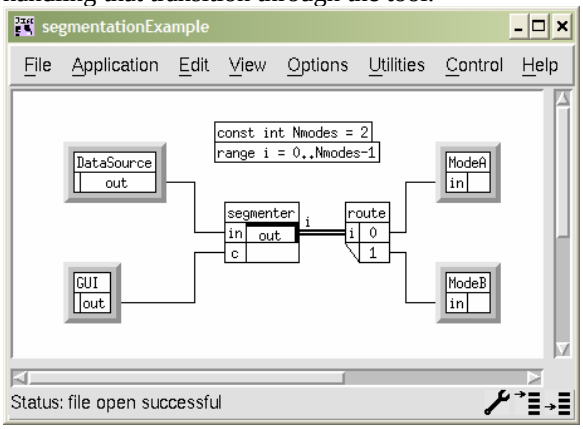

Figure 1 – Distributed control – such as mode switching – is generated automatically.

Automated Handling of Parallelism

Multicore processors, such as the Cell processor, present several challenges to programmers who wish to fully exploit the special purpose cores. The Cell is a system on a chip design (SoC) which houses 9 processing cores. One of the 9 cores is a central processing unit, or "Power Processing Element" (PPE). The other 8 cores – called "Synergistic Processing Elements" (SPE) – are designed for very compute-intensive, special purpose tasks. To

effectively program the chip, the programmer must tackle the hard problems of distributing work to many processors in order to take advantage of the architecture. Distributed control is often one of the most difficult tasks of software development, and distributing that control to lightweight cores requires that the control processing also be lightweight and highly customized to the task at hand.

The programming of an SPE also requires specialized knowledge, either in using the assembly language for the SPE or the data types and functional interface of its C/C++ extensions. A developer must have intimate knowledge of the vector ALU and its pipeline characteristics to effectively use the SPE. Once an SPE is hand coded using these language extensions, the code is tied to the Cell architecture and has very limited portability. An alternate multicore architecture, e.g., one with 3 PPEs instead of 1 PPE and 8 SPEs, could not be coded with the same language. A developer given the task to port an application for the Cell to this alternate architecture would likely have to start from scratch.

Autocoding using Gedae addresses these problems directly. Using the same technology it developed for addressing boards of DSP processors, Gedae can automatically implement the distributed control for heterogeneous multicore processors like the Cell architecture. The distributed control includes a predefined component, the Runtime Kernel (RTK), which is lightweight enough to run on processors such as SHARCs and Cell SPEs, and code-generated components for processors and FPGAs with insufficient resources to run the RTK. Because the generated implementation is built to run on a Virtual Machine, as shown in Figure 2, functionality specified in Gedae is not tied to a single architecture. Applications developed for DSP boards in Gedae can be quickly transitioned to the Cell and future multicore architectures when the RTK is placed on each core.

Automated Architecture-Based Optimization

Gedae-SFG (Signal Flow Graph) is a single sample extension to the Gedae programming language that enables compiler-like optimizations. While core Gedae primitives

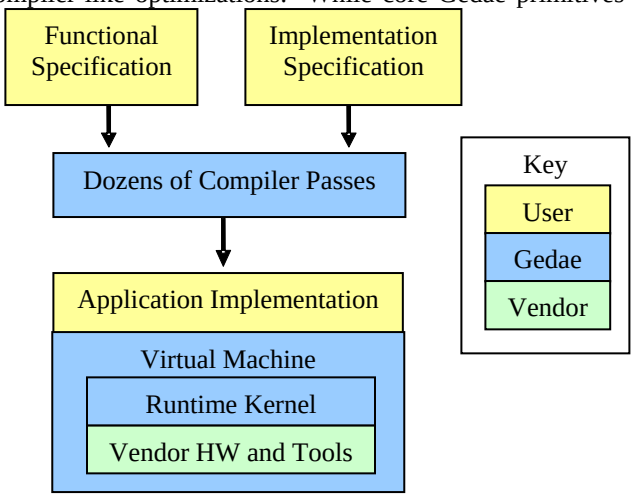

Figure 2 – Core Gedae implementation process

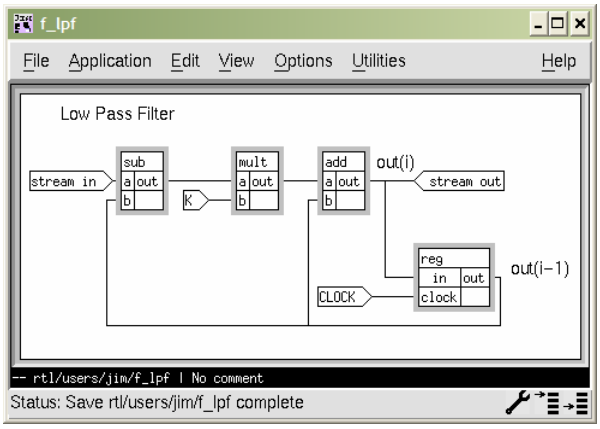

Figure 3 – Low pass filter implemented in Gedae-SFG

are written in Ansi-C (with Gedae-specific extensions, providing efficiency through links to optimized vector libraries), Gedae-SFG can be used to export code in any programming language, whether it's the same generic Ansi-C, VHDL for generating firmware, or assembler specific to a fixed processor. The language allows the algorithm to be viewed at fine detail by both the developer and the tool. This visibility enables the tool to automate many of the planning issues associated with using the processor efficiently, e.g., moving data through registers and local memory and fully utilizing a vector ALU.

Much like Gedae's core language, the Gedae-SFG graph (such as the low pass filter shown in Figure 3) specifies only the functionality of the graph without regard to the target or its programming language. Through the Language Support Package (LSP), target code is exported to implement the application, and generic Ansi-C code is created for simulation.

While this technology was developed to support FPGAs, the technology is also powerful for autocoding assembly-level code for fixed architectures, from older architectures like the SHARC processor to emerging technologies like an SPE on the Cell processor. To allow for these target-based optimizations, Gedae is provided with information about the target architecture on the processor level, including the operations and byte widths for each ALU path, the size of the register file, the size of the local storage, and how all the components are connected. By entering this information into Gedae's embedded configuration, Gedae is able to create a virtual model of the target processor that includes those components, and use that model to tailor the code generation to make maximum use of the components.

References

[1] Analog Devices, System Development and Programming for the ADSP-21161 SHARC Processor Workshop Slides.

[2] N. Blachford, Cell Architecture Explained, 2005.

[3] J. Kahle, et al., Introduction to the Cell Multiprocessor, IBM Journal of Research and Development, 2005.

[4] W. Lundgren, et al., Autcoding Sensor Processing Applications to Run on DSPs and FPGAs, EMRS DTC Conference, 2006.