Compiler Driven Architecture Design Space Exploration for DSP Workloads: A Study in Software Programmability Versus Hardware Acceleration.

Michael C. Brogioli and Joseph R. Cavallaro
School of Electrical and Computer Engineering
Rice University
Houston, Texas 77005
Email: brogioli@alumni.rice.edu, cavallar@rice.edu

Abstract— **Wireless communications and video kernels contain vast instruction and data level parallelism that can far outstrip programmable high performance DSPs. Hardware acceleration of these bottlenecks is commonly done at the cost of software flexibility. Many vendors, however, view software as intellectual property and prefer a software solution that is a proprietary implementation. The paper uses a research compiler for architectural design space exploration to present comparisons between compiler generated scalable software programmable DSP architectures versus hardware acceleration implementations. It shows that scaled up compiler generated software programmable DSP architectures can be attractive alternatives to non-programmable hardware acceleration.**

I. Introduction

Wireless telecommunications and multimedia kernels often exhibit amounts of instruction and data level parallelism that can far outstrip the computational ability of high performance software programmable VLIW style DSP cores. As such, often portions of the application in question are accelerated in hardware to achieve the required performance. The cost in doing this is the loss in software programmability and the flexibility of a software implementation. At the same time, many companies view their software as intellectual property, and have a strong desire for a software based solution that is a proprietary implementation. Examples of this are channel equalization kernels for baseband processing in wireless infrastructure, whereby vendors often maintain their own proprietary version implemented in highly optimized software on one or more programmable DSP cores. In deciding to offload functionality to hardware acceleration there are a number of design decisions which must be considered, many of which are sacrificed in one implementation versus another.

The paper presents application specific compiler driven architecture design space exploration for channel estimation in mobile wireless receivers. It compares the tradeoffs between compiler driven software programmable DSP implementations versus hardware based accelerator implementations. The studies are based around the Texas Instruments TMS320C64x DSP architecture. For each workload, the compiler can determine the appropriate programmable multiclustered VLIW DSP architecture for performance. In essence the maximum performance attainable in a programmable VLIW DSP implementation is found. This implementation is then compared against an FPGA hardware acceleration implementation in terms of not only FPGA performance but heterogeneous system performance. The contributions of this paper are an upper limit analysis, with the tradeoffs in programmability versus performance for workloads commonly considered computational bottlenecks in these application domains. The conclusions show that for certain workloads in this application domain, highly parallel software programmable VLIW based DSP architectures can approach that of an FPGA based hardware implementation, while maintaining the flexibility of a traditional software programmable implementation. For some workloads, as much as a 10x increase in performance is achieved while maintaining a software programmable solution that approaches the performance of the FPGA based implementation.

II. Background

Simulation has been an established design methodology in architectural exploration for a number of years. By modelling common system components at a high level in software, users can prototype systems much more rapidly [3], [4], [13]. Limitations in rapidly prototyping heterogeneous systems stem from simulated hardware not being well abstracted in software. In modelling heterogeneous systems, there is not a one-to-one mapping between system hardware and simulator software modules, nor well defined interfaces between software modules to accurately model components such as data busses, memory arbitration engines, etc. This results in an inherent lack of coupling between system timing and data flow, and can adversely effect system modelling accuracy.

There are commercially and academically available toolsets for modelling heterogeneous hardware at various levels. Early systems such as SOS from the University of Southern California targeted synthesis of heterogeneous multiprocessor systems

978-1-4244-5827-1/09/$26.00 ©2009 IEEE

based on task level partitioning of an application [9]. Similar to this was the PICO-NPA system from Hewlett Packard, which was a program-in, chip-out system [11]. While these tools provide a solution to the problem at the computational bottleneck level, they fail to provide insight into the overall system component interactions, and application software.

Commercially available solutions include the Seamless system from Mentor Graphics [6]. Seamless provides a hardware/software co-verification environment for multiprocessor heterogeneous environments with custom processing units such as FPGAs. While these tools are useful in the industrial setting, their closed source nature limits the amount of architectural exploration that can be performed.

Additionally, when designing a VLIW based DSP architecture as a compiler target, it is desirable to have an orthogonal architecture as a compiler friendly design. Due to the strict power consumption demands, most DSPs do not include out of order execution hardware, but instead rely on the compiler for performance of a VLIW based design. Also, due to the ornery nature of most DSP instruction sets, it is often difficult to build an optimizing compiler for the architecture.

There has been a significant body of work in scheduling code for VLIW based architectures common in the DSP community. There are powerful commercial compilers which vendors provide for their DSP architectures. Two examples of commercially available DSP compilers are the Texas Instruments' TMS320C6x compiler infrastructure and the Philips Trimedia compiler[8], [12].

There is a significant body of work, as well, focused around clustered VLIW architectures, which use multiple smaller register files rather than one monolitic central register file. The Yale Bulldog compiler was one of the first to examine the effects of VLIW cluster partitioning on instruction scheduling [5]. Rao et al. explicitly consider the problem of generating code for VLIW machines with clustered data paths [10], referred to as EPIC architecures. Ozer et. al, proposed a Unified Assign and Schedule algorithm coupling instruction partitioning and scheduling between VLIW clusters at the same time [7].

III. Experimental Setup

To evaluate the runtime performance of the channel equalization workloads in this paper, custom simulation and compilation frameworks are used. In evaluating the performance of the hardware/software partitioned workloads, the Spinach simulation infrastructure for DSPs and embedded systems is used [2], [14]. The Spinach simulation infrastructure is a reconfigurable simulator framework for rapidly prototyping heterogeneous systems. Simulators created with the Spinach framework are bit-true and cycle accurate, and run true compiled code executables. Simulators can model Texas Instruments' TMS320C6x DSPs, MIPS cores, Xilinx style FPGAs, as well as system level components common to embedded systems such as SRAMs, DRAMs, caches, DMA engines etc.

Table I shows the simulation system parameters for the hardware software partitioning between programmable TMS320C64x device and FPGA based hardware accelerators. It should be noted that all simulations of the hardware software partitionings are running actual compiled code executables created with the Texas Instruments' Code Composer Studio.

TABLE I
Baseline Spinach HW/SW Simulation Parameters

Simulation Parameters	*Value*
DSP Architecture	Texas Instruments TMS320C6401 Core
System Clock Rate	167MHz
Instruction Memory Bandwidth	256b on–chip
Data Memory Bandwidth	256b on–chip 32bB off–chip
Instruction Memory Size	64KB on–chip
Data Memory Size FPGA Model	64KB on–chip Xilinx Virtex II
FPGA Bus Bandwidth	32 bits
DMA Bus Bandwidth	32 bits per clock cycle bidirectional
Bus Arbitration	Round Robin

In experiments where the retargetable compiler is used for DSP design space exploration, the RISD retargetable compiler infrastructure for scalable clustered VLIW DSP architectures is used [1]. RISD is a source to source recompilation infrastructure used for design space traversal of multiclustered VLIW based architectures. The tool flow of the compiler is shown in Figure 1. Shaded blocks in Figure 1 are part of the proprietary Texas Instruments' toolchain, whereas all non-shaded blocks are part of RISD. The Texas Instruments' C compiler is used to generate assembly for the baseline TMS320C64x device. RISD then consumes this assembly, parses it, and reconstructs the control/data flow graphs. A register deallocation phase is performed, then the VLIW code scheduler reschedules and repartitions the code for the desired target specified in the machine description file. RISD supports architectural parameters such as number of functional units per VLIW cluster, register file size per VLIW cluster, ALU mix per VLIW cluster, total number of VLIW clusters, and VLIW cluster interconnect topology.

The code scheduler either reschedules for the input machine definition, or can traverse the design space looking at the tradeoffs between various architectures. Multiple scheduling algorithms are supported by the RISD framework. In these experiments, a version of the Unified Assign and Schedule algorithm developed by Ozer et. al for performing instruction scheduling and partitioning on a two-way clustered VLIW architecture is adapted [7]. The algorithm is extended to support greater numbers of register files and VLIW clusters by adding a point-to-point scheduling functionality for copying values across clusters not adjacent to each other.

IV. Workload and Hardware/Software Partitioning Results

.

This section investigates the runtime performance improvements of a hardware/software partitioning for 3.5G wireless

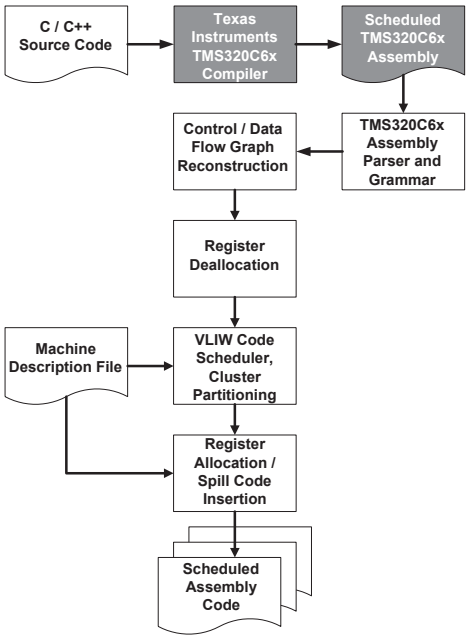

Fig. 1. RISD Compiler Tool Chain Flow

handset receiver channel equalization algorithms. Channel equalization is one of the largest computational bottlenecks in the receiver, and also consists of numerous intensive kernels that are common amongst various signal processing systems. As such, performance bottlenecks in a software-only implementation executing on a dedicated DSP core can limit data rates of the overall system.

Table II shows the various functional blocks that make up major components of a software-only implementation of the channel equalizer. Numbers are collected running on a dedicated single core Texas Instruments' TMS320C64x core system as simulated by the Spinach simulator described in Section III. The major bottlenecks are the Channel Estimation, FFT, FFT post processing routines, Inverse FFT, FIR Filter and Despread and Descramble. numbers are generated by hand written C code compiled with the Texas Instruments' Code Composer Studio compiling at -O3 optimization level, on out of the box code. Referring to the system configuration listed in Table I, these performance bottlenecks result in the software only implementation achieving only 13.4% of HSDPA line rate.

The profile data listed in Table II correlates to only making 13.4% of the data rate required for the channel equalizer. Various data and computational bottlenecks in the system have prevented the software only solution from achieving line rate. As a solution, this paper evaluate the performance increases of a heterogeneous DSP/FPGA based hardware/software implementation of the channel equalizer workload. Using a defined set of criteria that takes into account: spatial locality of data, data level parallelism, computational complexity of the algorithm, and in some cases task level parallelism, the software bottlenecks are iteratively partitioned into FPGA style

TABLE II
CHANNEL EQUALIZATION APPLICATION PROFILE DATA: ALL ALGORITHMIC PARTS EXECUTED IN SOFTWARE ON TMS320C64X SIMULATED DSP

Program Task	*Percentage Runtime*
Channel Estimation:	7.06%
Covariance Matrix:	0.34%
Circulation of Covariance Matrix:	0.19%
FFT:	9.90%
IFFT:	4.06%
FIR Filtering:	73.31%
Despread/Descramble:	3.97%

accelerators and performance improvement is determined by simulating the entire system.

Figure 2 below shows the hardware and software partitioning of the channel equalization workload when computation is offloaded from software executing on the DSP to various FPGA accelerators in the system. All accelerators are modeled as Xilinx Virtex-II devices, and each kernel offloaded is placed in its own FPGA. All data transfer from host processor DSP memory to FPGA accelerators takes place via DMA communication setup by the host DSP.

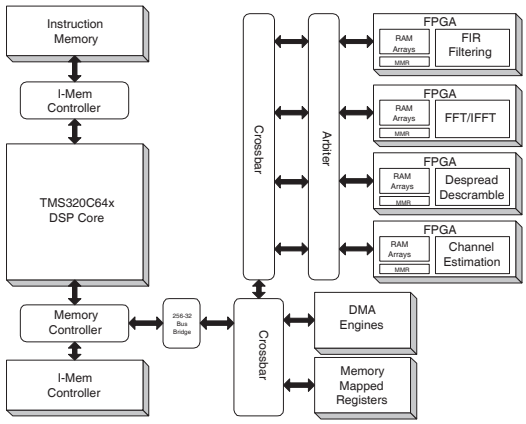

Fig. 2. DSP/FPGA Hardware Software Partitioning of Channel Equalization on Spinach Simulator Instance, Running Compiled Code Application

Figure 3 shows the performance gains in offloading each of the bottlenecks in the software only implementation to various accelerators in the system. This is the actual simulator topology of the Spinach simulator modelling both the programable DSP cores, FPGA accelerators, and system interconnect. The Y-axis of this graph shows the overall channel equalizer performance. The X-axis shows the various configurations of hardware/software partitioning. The first stacked bar graph along the X-axis is the software only implementation executing on the dedicated TMS320C64x DSP core. The second stacked bar has the DSP software with the FIR offloaded into FPGA acceleration. The third stacked bar contains FIR, FFT and IFFT offloaded into FPGA acceleration. The fourth bar contains FIR, FFT, IFFT and Despread Descramble offloaded into FPGA while the fifth stacked bar contains all bottlenecks

offloaded into FPGA acceleration.

It is shown that with a fully partitioned system, performance improvements on the order of 11x are achieved. These numbers are mildly pessimistic as all computational kernels offloaded are placed in a unique FPGA, rather than mapping multiple components to the same FPGA which would be possible with newer Xilinx Virtex family devices. It should also be noted that these performance improvements include the DMA transfer overhead and bus contention that is often a problem in a real system. That is to say, all bytes moved from DSP local host memory to FPGA accelerator and back are accounted for in the overall system runtime. Further detailed information can be found in [1].

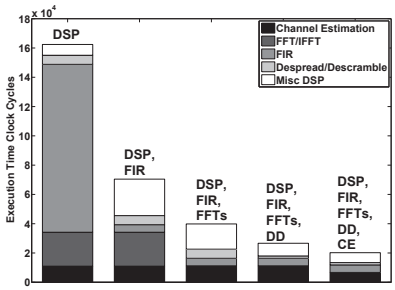

Fig. 3. Channel Equalizer Program Performance on TMS320C64x with Various Hardware Acceleration in FPGA running on Spinach Simulated System

V. Compiler Driven Design Space Exploration

As was mentioned in Section I, it is often desirable for vendors to have software only solutions of their key algorithms as it allows for flexibility and customization of IP that distinguishes their technology. While impressive results are seen for the above channel equalization workloads when accelerated with FPGA based coprocessors, the question arises as to how well a software programmable device could compete. In this section, the RISD compiler described in Section III is used to iteratively scale up the baseline programmable DSP architecture to suit the needs of each computational bottleneck in the channel equalizer.

In this section, the RISD compiler scales the number of VLIW clusters on the target architecture in attempts to meet the performance of the hardware acceleration discussed previously. That is to say, the default VLIW cluster of the baseline Texas Instruments' TMS320C64x is cloned multiple times by the compiler as needed for the newly scaled up architecture. The TMS320C64x is a two way clustered VLIW architecture, with each cluster containing four functional units. Each register file has 32 entries with a single crossbar connecting each VLIW cluster for exchange of a single operand between register files per clock cycle. In scaling the DSP architecture, all architectural parameters are kept constant other than the number of VLIW clusters used on the chip. When more than two VLIW clusters used by the compiler, it is assumed there is a point to point crossbar connecting various register files. The compiler must take into account moving operands across multiple clusters in the event that a multi-cluster hop is needed.

A. Measuring Performance of Re-scheduled VLIW Code

As the number of VLIW clusters used in the compiler chosen architecture increases, the runtime performance of the recompiled code is measured using a combination of static code schedule analysis and runtime profiling data. Using the runtime profiling information collected in the previous simulations, the compiler knows the execution count of each basic block in the kernel. The compiler then couples this information with statistical measurements of the code schedule length of each basic block in the kernel, as all code is scheduled statically at compile time without runtime out of order execution. Equation 1 shows how the final cycle count of the rescheduled kernel is determined for each of the various proposed multiclustered VLIW architectures.

$$total\_runtime_{resched} = \sum_{block_0}^{block_N} (resched\_runtime * block\_execution\_count)$$

B. Limitations of DSP Source Code Recompilation

The numbers presented in this section are somewhat pessimistic due to a number of limitations in rescheduling source code from the assembly level. It should be noted that assembly to assembly source recompilation was required due to the closed source nature of the Texas Instruments' proprietary compiler. One limitation is that the original assembly may have additional instructions such as spill code insertion which are not part of the original kernel's computation, but were required for execution on the original TMS320C64x architecture. Another limitation is that there may be false dependencies in the original input assembly which the RISD compiler can not remove. A final limitation is that the version of the RISD compiler used in these studies schedules code on basic block boundaries, and does not make attempts to hoist instructions across basic blocks or out of loops.

Figure 4 below shows the normalized program runtimes of each of the kernels that were bottlenecks in the original software only implementation executing on a TMS320C64x DSP core. For each of the five kernels, the number of VLIW clusters used in the compiler proposed target architecture is scaled. The first grouping on the X-axis is the performance executing natively on the TMS320C64x itself, while the second grouping onward shows the performance of the same code recompiled by the RISD compiler as the number of VLIW clusters is scaled.

The first interesting point, as the VLIW clusters are varied, is it appears that the RISD framework outperforms the Texas Instruments' native compiler for a two way clustered VLIW architecture. This is in fact not true, as in these studies spill code insertion is turned off in RISD. The reason for this being that while spill code is certainly an issue for many

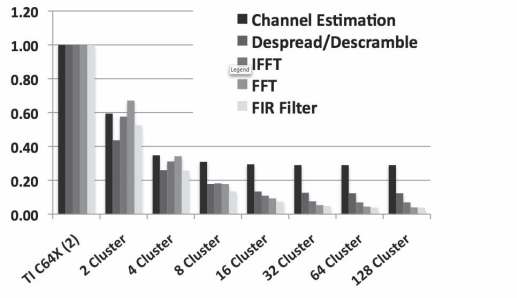

Fig. 4. Compiler Driven DSP Architecture Program Runtimes (of Compiled Code)

kernels when only two VLIW clusters are used, by the time the number of VLIW clusters is scaled up to eight or sixteen spill code is no longer an issue. That is to say, when the hardware resources meet the demands of the application, spill code is far less of an issue. The delta in not having spill code inserted for the two way VLIW clustering as scheduled by RISD is also due to the fact that RISD aggressively unrolls loops and attempts to expose as much ILP as possible without regard for spill code insertion in these studies. Again, this was done under the assumption that as the hardware scaled up to meet the demands of the application spill insertion would be negligible. In essence the high level of spill code insertion needed in that case is artificially created by the scheduling algorithms used in this study, and could be avoided in a production environment.

For most all kernels in the system, it can be seen that by the time eight to sixteen VLIW clusters are used there appears to be a sweet spot in load balancing. When most kernels are scheduled on a sixteen cluster VLIW architecture, improvements on the order of 10x can be seen. In these studies, virtually no spill code is needed when register file sizes are between 32 and 48 entries per cluster. One noted exception in performance gain is the channel estimation kernel, which does not scale in accordance with the others. This is mostly an artifact of the code shape of the input kernel. The amount of parallelism was restricted due to loop structures with non-trivial control flow. This severely limited the amount of ILP that this particular version of RISD could extract at compile time. The loops themselves could have been restructured by the application programmer to expose far more parallelism which was a point for later studies. In summary, however, for most computational bottlenecks the performance of a software only solution can be scaled to as much as 10x by scaling the hardware accordingly.

VI. Conclusions

This paper shows that there exist significant computational bottlenecks in modern DSP workloads that can far outstrip the performance of modern high performance programmable DSP architectures. As such, the paper illustrates the whole system performance gains that can be achieved by hardware accelerating various bottlenecks in the software only system. Due to may vendors prefering a software based implementation of their proprietary algorithmic intellectual property, however, the paper investigates compiler driven massively VLIW clustered DSP architectures as an alternative to hardware acceleration in the system. The paper shows that for key bottlenecks in the system often accelerated in hardware, software programmable compiler driven DSP architecture designs can often provide similar performance gains.

Acknowledgment

This work was supported in part by Nokia, Nokia Siemens Networks, Texas Instruments, Xilinx, and by NSF under grants CCF-0541363, CNS-0551692, CNS-0619767, EECS-0925942 and CNS-0923479.

References

[1] M. C. Brogioli. *Reconfigurable Heterogeneous DSP/FPGA Based Embedded Architectures for Numerically Intensive Computing Workloads*. PhD thesis, Rice University, Houston, Texas, USA, May 2007.

[2] M. C. Brogioli and J. R. Cavallaro. Modelling Heterogeneous DSP/FPGA System Partitioning with Extensions to the Spinach Simulation Environment. In *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, October 2005.

[3] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogenous Systems. *Int. Journal in Computer Simulation*, 4(2), 1994.

[4] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[5] J. R. Ellis. Bulldog: A Compiler for VLIW Architectures. In *The MIT Press*, 1985.

[6] Mentor Graphics. Seamless Co–Verification, Texas Instruments Processor Support Packages. www.mentor.com/seamless.

[7] E. Ozer, S. Banerjia, and T. Conte. A New Approach to Scheduling for Clustered Register File Microarchitectures. In *Proceedings of the 31st Annual International Symposium on Microarchitecture*, pages 308–315, December 1998.

[8] Phillips Semiconductors. Phillips Trimedia Line of Processors. http://www.semiconductors.philips.com/pip/.

[9] S. Prakash and A. Parker. SOS: Synthesis of Application-Specific Heteroeneous Multiprocessor Systems. In *Journal on Parallel and Distributed Computing, Volume 1*, pages 338–351, 1992.

[10] B. R. Rau, V. Kathail, and S. Aditya. Machine Description Driven Compilers for EPIC and VLIW Processors. *Design Automation for Embedded Systems*, 4(2/3), 1999.

[11] R. Schreiber, S. Aditya, B. Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. PICO-NPA: High-Level Synthesis of Nonprogrammable Hardware Accelerators. Tech. Report HPL-2000-31, Hewlett–Packard Company, May 2000.

[12] Texas Instruments. TMS320C62XX CPU and Instruction Set Reference Guide. http://dspvillage.ti.com.

[13] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural Exploration with Liberty. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 271–282, November 2002.

[14] P. Willmann, M. C. Brogioli, and V. Pai. Spinach: A Liberty–Based Simulator for Programmable Network Interface Architectures. In *Languages, Compilers, and Tools for Embedded Systems*, 2004.