Exploiting Data Context Switching on a VLIW Coprocessor

Ali Ibrahim, Al Davis

School of Computing, University of Utah
Salt Lake City, UT 84112
{ibrahim | ald}@cs.utah.edu

Abstract

Traditional loop transformations improve program performance by increasing memory usage efficiency. These tactics are limited by the existence of loop carried dependencies. Data context switching is an additional transformation technique which improves performance in high-performance configurable processors when loop carried dependencies are present. Since data-context switching requires larger local memories, higher memory bandwidth and a very large register file, this tactic has not been useful for the inherently energy constrained embedded system domain. This paper demonstrates that the problems imposed by data context switching can be solved by supporting additional architectural mechanisms which support cluster pipelining, modulo scheduling, addressing modes for register files, and independent address generation units. These features are part of the ACT (Adaptive Cellular Telephony) coprocessor. Utilizing data context switching improves the performance of a loop carried dependency IIR filter application by a factor of 2.15 when compared to an unoptimized naive ACT implementation. The optimized ACT implementation is 7.09 times faster than an equivalent DSP implementation and is better than an Intel Xscale implementation by a factor of 22.7.

1. Introduction

The performance of current and future embedded processors has to be high enough to support the intensive arithmetic requirements of current and future embedded applications such as wireless cellular telephony. The additional challenge is to do so under stringent energy budgets. The rapid rate of change of cellular telephony requirements motivates an additional desire to create architectures which are also flexible. Achieving high performance and flexibility within a limited power budget is a difficult task. Traditionally, low power and high performance have been achieved by the utilizing application specific integrated circuits (ASICs). The problem is that ASICs are intrinsically inflexible and even minor changes to the application cause another lengthy and costly ASIC design cycle. Neither are commensurate with the rapid rate of change and the cost structure of the embedded systems or cellular telephony domains. General purpose processors (GPPs) are flexible but consume too much energy; DSPs and embedded processors are relatively energy efficient but do not have adequate performance. This motivates the search for an alternate approach.

The key to the solution is to exploit opportunities for customization for a particular application domain rather than for an individual compute intensive kernel in a specific algorithm. The application domain which motivates this work is wireless cellular telephony, where standards are evolving rapidly to support the higher bandwidth requirements imposed by the shift from simple voice transport to uses which support voice and both static data and increasingly higher fidelity/resolution real-time multimedia voice/video streams. Achieving this daunting objective will require performance and energy oriented optimizations at all levels of the architecture while retaining the requisite flexibility imposed by the dynamics and cost structure of the cellular telephony domain.

Cellular telephony and multimedia applications have some important characteristics that provide an opportunity. They are stream oriented tasks which process an input stream of frame organized data to produce a stream of output frames. The computations do involve keeping some state information but fortunately the memory requirements for the input, output, and state data are relatively modest. As a result, the applications have a characteristic structure that consists of a compute intensive nest of regular loops. Hence, loop transformations have been used extensively to improve program performance. Typical transformations are loop unrolling, fusion, and interchange. The primary benefits of these traditional transformations are due to improved memory utilization efficiency and increased instruction level parallelism (ILP) in the transformed codes. However these transformations have limited applicability when loop carried dependencies exist in applications such as echo cancellation, voice over IP (VoIP), cryptography, adaptive rake receiving [7, 9], equalization, and multiuser interference cancellation.

The high performance reconfigurable Chameleon processor [2] improved the execution time of dependent loops by using a data context switching technique [1]. A data context is effectively the state of the outer-most iteration of the loop nest. Significant improvement resulted from interleaving these outer loop contexts. However this requires a large local memory to store suspended data contexts, a larger register file, and higher register-memory bandwidth. These requirements are problematic in the energy constrained embedded processing domain. This paper demonstrates that data context switching can be supported in an embedded low-power VLIW coprocessor by adding architectural mechanisms to support inter-cluster pipelining, independent address calculation structures for both registers and memory, modulo scheduling, execution retiming, and loop control.

The primary purpose of this paper is to provide a detailed operational view of how data context switching and the additional architectural mechanisms can be utilized in the context of a single characteristic infinite input response filtering (IIR) application. The next section describes the high level architecture of ACT. More detailed descriptions and evaluations on a broader set of telephony and general DSP applications can be found in [4, 5]. A simple IIR filter is then described as a loop carried dependency example. Section 4 illustrates how the IIR application is mapped onto the ACT architecture. The results and conclusions of this approach are then discussed.

2 Architecture

The high-level ACT architecture is shown in Figure 1. Data is pushed into the coprocessor via the input SRAM by a host processor or memory controller. The results are posted to the output SRAM and are removed by the host processor. The ACT coprocessor is responsible for the execution of compute intensive tasks.

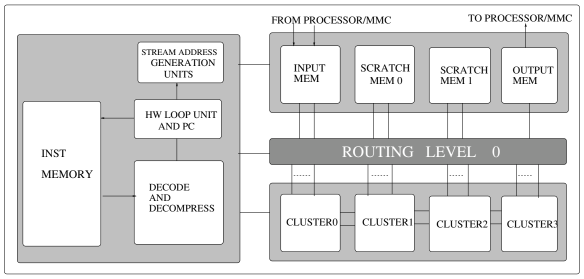

Figure 1: Coprocessor organization

The high level design can be functionally partitioned into storage, four execution clusters, address generators, hardware loop unit, and the decoding/decompress unit. Each cluster in Figure 2, consists of three integer execution units (XUs), and two 32-entry register files (each with 1 read and 1 write port). The ACT architecture has been implemented but not yet fabricated in a .25μ TSMC bulk CMOS process and runs at 300 MHz. All of the XUs have single cycle latency with the exception of a pipelined multiply/shift unit which takes two cycles. Routing level 0 consists of a set of 6:1 multiplexors and routing level 1 consists of 5:1 multiplexors. A more detailed description of the ACT architecture and its performance on a broader set of telephony and DSP benchmarks can be found in [5]. Each register file is divided into 2 windows. One window is used as a typical static register file. The other window, is accessed using simple address modes which support efficient renaming. Memory addresses are generated using separate partially configurable address generation units for each memory. Details of the AGU's and modally addressed register file (MARF) can be found in [4], and have been shown to accelerate a wide variety of DSP algorithms. Both the MARF and AGU mechanisms will be subsequently shown to have additional utility in enabling data context switching.

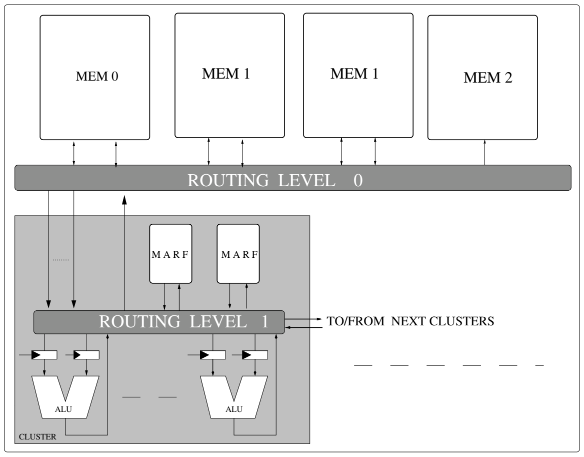

Figure 2: Clusters organization.

3 Loop Carried Dependencies

Adaptive filters are characteristic examples exhibiting loop carried dependencies. The key attribute of these filters is that they are self-adjusting. They are used in applications which require that the filter characteristics adapt to input signal conditions. An adaptive MMSE rake receiver combines the benefits of both rake [3] (widely used in 3G cellular telephony codes) and MMSE receivers exhibits similar characteristics. In this algorithm, multiple adaptive correlating receivers, called fingers, are used to enhance the received signal quality. An infinite Impulse Response (IIR) is simple example of a loop carried dependency code. IIR filters are an important component of larger applications such as VoIP.

In [1], context switching was implemented on a FPGA, and a high performance reconfigurable processor. In order to facilitate an accurate comparison, this algorithm has been chosen as the driving example in this paper to illustrate the benefits of the ACT coprocessor. They also developed a DSP reference implementation which did not take advantage of context switching. It is important to note that this paper's DSP comparison is also based on this reference implementation. While IIR is used here for pedagogical purposes, it is important to note that similar tactics have proven useful for a much wider variety of DSP and telephony algorithms.

The pseudo code for a VOIP IIR filter is:

```
$for\ i = 1\ to\ H\ do:$
    $for\ j = 1\ to\ S\ do:$
        $tot = 0$
        $for\ k = 1\ to\ C\ do:$
            $tot = tot + w[i,k] * y[i, j-k-1]$
        $end$
    $end$
    $y[i,j] = w[i,0] * x[i,j] - tot$
$end$
```

In this algorithm, H is the number of channels, S is the number of samples per channel, and C is the number of filter taps. Each iteration of the outer most loop 'i' represents a particular data context. This particular example uses H contexts. By interleaving the execution of the filter taps between different contexts, the performance can be improved significantly. The problem with this approach is that a large memory is required to keep the coefficients of all channels, and the large number of temporary data values must be stored for each filter tap output.

The performance of this loop on a DSP or GPP can be improved by unrolling the inner most loop in order to increase ILP. Unfortunately unrolling only works for the inner-most loop. Unrolling the outer loop has more ILP potential but will not improve performance due to saturation effects due to available memory-register bandwidth and the size of the register file. Improving address calculation efficiency will also have a significant impact on performance. For this IIR application, modulo addressing is a dominant theme. Attempting to take advantage of data context switching in a processor which performs address calculations and other arithmetic requirements on the same execution units will exhibit reduced performance due to XU saturation effects. The ACT processor architecture employs independent AGU resources in order to free up the cluster XUs for real work. These AGUs are partially configurable to support multi-dimension strided accesses to the attendant SRAMs. The overall ACT architecture will be shown to efficiently evaluate algorithms containing loop carried dependencies.

4 Mapping

Each cluster in ACT has two 32-entry register files. Each register file has only a single read and a single write port. Multiple read and write ports improve bandwidth but consume an incommensurate amount of power. The 2-level routing structure of the ACT architecture provides sufficient data routing flexibility to counteract this apparent bandwidth defect while consuming less energy than a multiported register file. Data context switching requires multiple register mappings and a considerable amount of register renaming. If this can be done, data storage can be reduced and performance can be significantly improved.

Figure 3 shows the memory map for a number of data channels (CH0,..,CHn) and for a number of temporary channels(TMP0,..,TMPn) which store the filter history y[j-k]. Memory blocks used to store TMPi are accessed as a circular buffer. The addressing mode for CHi depends on the amount of context switching that can be employed. For instance, if the total number of contexts is N and all of them can be switched, then simple linear addressing will suffice. If N is implementation limited due to memory capacity, then a circular addressing mode will be necessary. A similar argument holds for coefficient addresses if they do not fit in the register file.

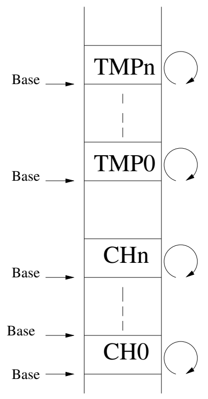

Figure 3: Memory mapping.

In ACT [4] the partially configurable AGUs and MARF units provide the required flexibility. ACT AGUs support both simple (1 dimensional strided and 2 dimensional unit strided accesses) and up to four complex (2 dimensional non-unit strided accesses) address patterns. The MARF unit supports 2 register windows, one of which is directly accessed in the traditional fashion while the other is accessed via a set of simple address modes. One of the MARF modes supports circular addressing. The combination of the AGU and MARF capabilites, coupled with 4 scratch RAM ports can efficiently support up to 8 IIR data contexts. The 8 contexts, given that C=10, requires 88 coefficients which easily fit into the available register space of the four cluster ACT design.

In order to avoid additional temporary storage required by data context switching, the execution of each cluster has to be retimed so that the result of one context in a particular cluster is accumulated with the result of the same context in a different cluster. This retiming model is illustrated in figure 4. In this mapping, the execution of each context is pipelined through the four cluster series. Once the accumulated values reach the final cluster they are then sent to memory. Figure 5 shows the execution of the first stage of the pipeline. Routing levels 1 and 0 are configured to support the particular needs of the application. The primary advantage that ASICs derive is the ability to utilize application specific pipelined data-paths. The ACT data-path derives the same advantage albeit more flexibly and with slightly less efficiency. Both the flexibility and efficiency differences are due to routing data through the 2-level mux-based routing layers rather than using an ASIC-like hardwired interconnect. Since routing in ACT is under software control, ACT supports creation of a different pipeline on every cycle. Figure 6 shows the execution of the first and second context in the first and second stage of the pipeline utilizing cluster0 and cluster1. These two figures illustrate how one of the inputs is initialized to the accumulated value from the previous context which was executed in the preceding cluster.

This retiming mechanism, even though it alleviates the additional storage problem, increases the scheduling time of the program since the inter-cluster pipeline has to be filled and drained in each interleaved iteration of the 8 contexts. In order to avoid the fill-drain parasitics, which degrades performance by 37.5% in the IIR case, the loop also needs to be modulo scheduled. Modulo scheduling [8] is a technique used to increase throughput by initiating future iterations of a loop before the current iteration finishes.

5 RESULTS

ACT performance measurements were obtained using the commercial Synopsys EDA too suite on a fully synthesized .25μm Verilog- and Module Compiler based implementation. A full clock tree and worst case RC wire loads are included in the model. For the general purpose processor measurements, a low power 400MHz .18μm Intel XScale (StrongARM) PXA250 system has been used. The DSP implementation was taken from the literature [1].

Table 1 shows that by applying context switching, inter- and intra-cluster pipelining, retiming, and modulo scheduling, the performance of the loop-carried-dependency IIR code has been improved by 2.15 when compared to an unoptimized (ACT-NOPT) naive mapping of the IIR code onto ACT which does not take advantage of data context switching. When compared to the DSP implementation, the optimized ACT implementation is 7.64 times faster. However this assumes that the host processor is capable of feeding the two scratch SRAMs without stalling the XU resources which is not possible in the ACT architecture.

More precisely, ACT's input SRAM is double buffered and dual ported to allow full concurrent access by the host and ACT coprocessor while the scratch SRAMs are not double buffered due to general energy considerations. For the IIR application however, it is more efficient to have the host processor directly load the scratch SRAMs rather than the input SRAM which then would require the ACT coprocessor to copy the data to the scratch SRAMs prior to ex-

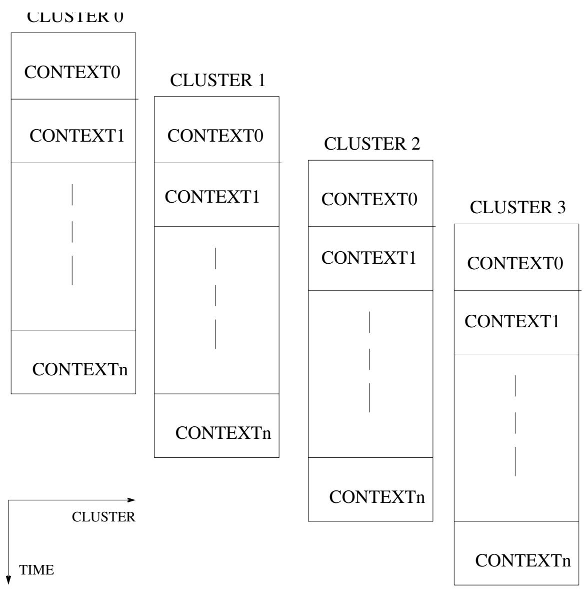

Figure 4: Execution Retiming.

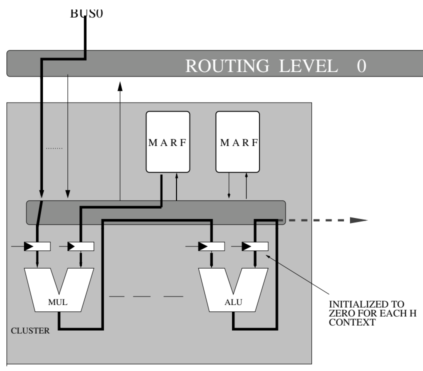

Figure 5: Context Mapping (a).

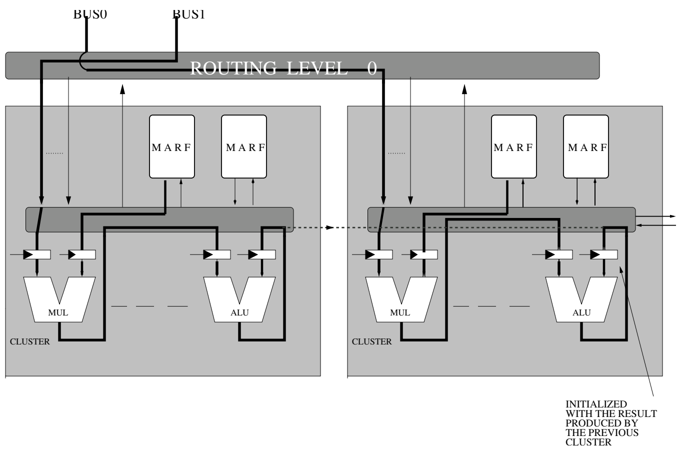

Figure 6: Context Mapping (b).

ecution of the IIR mapping that has been illustrated. The downside of this approach however is that while the scratch SRAM's are being loaded by the host, the clusters will stall. This delay adds an additional .53 μsec to the 8 channel execution time which degrades performance by 7.6% to achieve an aggregate benefit of 7.09 times the performance of the DSP implementation [1]. The overall ILP for the IIR application was 11.6 for the AGUs and 7.5 for the XUs.

The embedded processor Xscale is capable of sustaining data rates of 3.92 MS/s. ACT, on the other hand, sustains a data rates that is 22.7 times that of the Xscale. The fact that the Xscale does not perform well in this type of application is primarily due to its small register file size, bandwidth limitations, and more importantly on the fact that address calculations must be done in the main execution pipeline.

	Exec. Time (μsec)	Num. of Channels
DSP C62X (300MHz)	660	100
ACT-OPT (300MHz)	6.91	8
ACT-NOPT (300MHz)	14.9	8
StrongARM (400MHz)	161	8

Table 1: Performance results

6 CONCLUSIONS

ACT has been designed to accelerate 3G wireless baseband algorithms [6] under a stringent energy consumption budget. ACT incorporates a variety of architectural mechanisms that enhance performance for a limited increase in energy. Data context switching has proven to be a useful performance acceleration mechanism for applications with loop carried dependencies in high performance processors. However, the attendant increase in memory and register footprints and the commensurate increase in register-memory bandwidth have previously made this tactic inaccessible in the embedded domain. This paper shows that key features of the ACT architecture enable the data context switching tactic to be employed in the embedded domain. These mechanisms include: independent AGUs, register addressing modes, modulo scheduling, and execution cluster retiming. The result is a significant improvement over traditional embedded or DSP processor implementations.

[1]Note that it would be easy to double buffer the scratch SRAMs as well but the energy disadvantage, which is not the focus of this paper, would make this an unwise design choice.

7 Acknowledgments

The authors are indebted to Mike Parker, Cray Inc. for his contributions to the development of the ACT architecture and for his assistance in developing a hardware testbed which allow us to measure application performance on the Intel XScale processor.

References

[1] K. Bondalapati. Modeling and mapping for dynamically reconfigurable hybrid architectures. *PhD. Thesis*, 2001.

[2] Chameleon Systems Corp. http://-www.chameleonsystems.com.

[3] H. Holma and A. Toskala. WCDMA for UMTS: Radio access for third generation mobile communications, 2nd edition, john wiley & sons, 2002.

[4] A. Ibrahim and A. Davis. Address acceleration mechanisms for an adaptive cellular telephony processor. *Submitted to IEEE International Conference on Multimedia and Expo (ICME05)*, 2005.

[5] A. Ibrahim, A. Davis, and M. Parker. ACT: A low power VLIW cluster coprocessor for DSP applications. *Submitted to EuroMicro Conference on Real-Time Systems (ECRTS05).*, 2005.

[6] A. Ibrahim, M. Parker, and A. Davis. Energy efficient cluster coprocessors. *International Conference on Acoustics, Speech, and Signal Processing*, May 2004.

[7] J.Yi and J. H. Lee. Rake receiver with adaptive interference cancellers for a DS-CDMA system in multipath fading channels. *Proc. IEEE VTC2000*, pages 1216–1220, 2000.

[8] B. R. Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. *Proceedings of the 27th annual international symposium on Microarchitecture*, pages 63–74, 1994.

[9] G. Rice, D. Garca-Als, I. Stirling, S. Weiss, and R. Stewart. An adaptive MMSE rake receiver. *34th Asilomar Conference on Signals, Systems and Computers*, 2000.