ENERGY EFFICIENT CLUSTER CO-PROCESSORS

Ali Ibrahim, Mike Parker, Al Davis

School of Computing, University of Utah
{ibrahim | map | ald}@cs.utah.edu

ABSTRACT

New 3G wireless algorithms require more performance than can be currently provided by embedded processors. ASICs provide the necessary performance but are costly to design and sacrifice generality. This paper introduces a clustered VLIW coprocessor approach that organizes the execution and storage resources differently than a traditional general-purpose processor or DSP. The execution units of the coprocessor are clustered and embedded in a rich set of communication resources. Fine grain control of these resources is imposed by a wide-word horizontal micro-code program. The advantages of this approach are quantified on a suite of six algorithms that are taken from both traditional DSP applications and from the new 3G cellular telephony domain. The result is surprising. The execution clusters retain much of the generality of a conventional processor while simultaneously improving performance by one to two orders of magnitude and by reducing energy-delay by three to four orders of magnitude when compared to a conventional embedded processor such as the Intel XScale.

1. INTRODUCTION

Despite the rapid rate of embedded and DSP processor improvement, a number of important signal processing tasks such as 3G cellular telephony [1] cannot be adequately supported in the embedded space without employing ASICs. ASICs are efficient in terms of area, power and performance but lack flexibility and suffer from expensive design costs and time. ASIC accelerators for DSP systems take advantage of several common properties of DSP algorithms. The majority of the processing is dominated by regular inner loops which process streams of input signal data. The high level of parallelism makes them amenable to processor specialization. Similar properties hold for other important application domains: encryption/decryption, media encode/decode, speech or visual feature recognition, etc.

This paper presents an alternative approach to processor specialization which retains much of the generality and programmability of traditional processor solutions while achieving both performance and energy consumption levels that are closer to an ASIC approach. From a high-level perspective, the approach relies on two simple changes to a conventional processor architecture. First the execution resources are organized as a cluster that is embedded into a rich set of communication resources. Second, the movement of data through the communication infrastructure, and the control

of execution units is orchestrated by a program that provides fine-grained control for the physical resources. This program is essentially a wide-word horizontal micro-code. The combination of these two ideas allows special purpose computational pipelines to be dynamically established that resemble data flows found in an ASIC implementation. The fine-grained software control provides considerable generality since these pipelines can be instantly dynamically reconfigured to support a new processing phase. Algorithms which map poorly onto the communication and execution resources still run but at reduced efficiency.

This architecture therefore represents a middle ground between general purpose embedded processors and ASICs. The approach is evaluated by manually mapping a set of six signal processing applications onto a CMOS implementation of the architectures. Three of these algorithms are taken from conventional signal processing. The first is an 8x8 matrix multiply (Matmult). The second selects the maximum element from a 128 element vector (Vecmax). The third application, Dotp_sqr takes two vectors $\mathbf{V1}$ and $\mathbf{V2}$ and produces two dot products $\mathbf{V1} \cdot \mathbf{V1}$ and $\mathbf{V1} \cdot \mathbf{V2}$.

The other 3 applications were selected from the new 3G wireless telephony standard. T-FIR is a 16-tap transpose FIR filter. RAKE [2] extracts signals from multipath aliasing effects. Our implementation involves four complex correlation fingers. TURBO is a complex encoding application that exhibits superior error correction capabilities. This implementation contains 2 max-log-MAP modules, an interleaver, and a de-interleaver which implements the algorithm described in [3].

Both RAKE and TURBO pose implementation difficulties due to their high storage demands. For TURBO we limit the frame size to 500 and use a sliding window technique where the window is limited to 90. This allows the input frame data and temporary variables to fit within the memory resources in the coprocessor. For the RAKE receiver, the maximum temporal difference in multi-path delays will determine the necessary storage for the input samples. Since the total storage needed to implement RAKE in the coprocessor exceeds local memory requirements, the spreading code must be continually supplied in pieces by the host processor.

The performance and corresponding energy consumption for these six algorithms are compared on 3 different implementations: a special purpose ASIC, mapping the algorithms onto the cluster architecture, and compiled versions of the applications running on an Intel XScale (StrongARM) processor. It is important to note that a common circuit design trade-off is to pursue energy efficiency or high performance. Hence comparing architectures using a single metric is misleading. Horowitz [4] argues that a more rele-

vant comparison of architectural merit should be based on the rate of work per energy or an energy delay product. The cluster based coprocessor approach will be shown to achieve an energy-delay efficiency close to that of an ASIC while retaining a reasonable level of generality.

2. ARCHITECTURE

The architecture presented here has a 16-bit wide internal datapath to match the worst case need of the six test applications. Some of the codes also operate on single- or 8-bit data, and in these cases the appropriate sized field is extracted from the 16-bit native values. The high level organization of the coprocessor is shown in Figure 1. In this organization, data is pushed into the coprocessor via the input SRAM by a host processor. The input SRAM is dual ported to provide simultaneous access to the coprocessor and the host. Similarly, results are posted to the output SRAM and are removed by the host processor. The coprocessor can then handle compute intensive tasks and the host processor is only responsible for course grained copying of input and output frames.

The coprocessor core is divided into storage and execution components. Storage consists of two dual-ported scratchpad SRAMs to store local data, and an instruction SRAM to store the microcode. Four local data ports are required in order to avoid performance losses due to memory starvation at the execution resources. The execution part consists of four coarse-grained programmable units (PUs) and four address generation units (AGUs). Each PU further contains multiple execution units such as ALUs. These resources and the data steering required to move data between them is under software control. The control program is essentially horizontal microcode and resembles a fine-grained VLIW architecture. The main difference between this approach and a typical VLIW processor is that the communication infrastructure between the execution resources is richer, more flexible, and is controlled at a more fine-grained level than is found on VLIW designs.

Each AGU is associated with a particular PU. AGUs provide addresses to either the scratchpad, input or output SRAMs. Read-write control, and the target SRAM for a particular AGU's address is provided by the microcode. An AGU contains two registers and an adder. It can accept a data value from the associated PU or an immediate value from the micro-code. AGUs effectively support a variety of common address modes: immediate, indirect, displaced, and strided. AGUs do not support scaled addressing since all of the memories are simply organized as an array of 16-bit words.

The PUs are interconnected by 4 general intra-cluster busses and are fully connected to each other by full-duplex point to point links. The busses are used to feed the PU's from the memories and the point to point links allow execution pipelines to be effectively set up between PUs. The interconnection scheme is illustrated in Figure 2. Note that each of the 4 general busses are driven by the output of a 5 to 1 mux. These muxes are used to select between

Fig. 2. Cluster interconnection
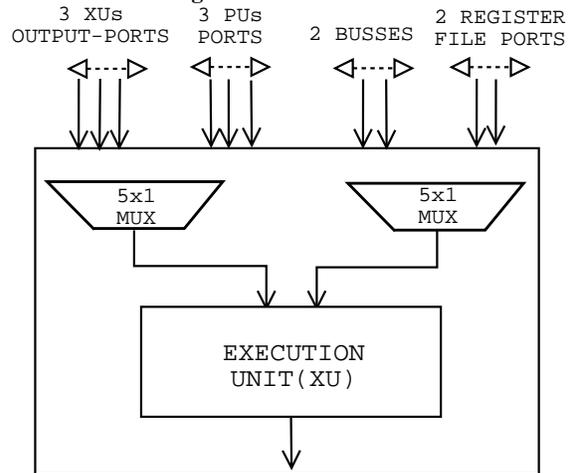
Fig. 3. Execution Unit

the 2 input SRAM ports, one read port from each of the scratch SRAM's, and an immediate value from the instruction.

The 4 cluster outputs are routed to the 4 scratchpad SRAM ports. Each of the cluster outputs is driven by a 2:1 mux, the M boxes in Figure 2. The inputs of the 2:1 muxes come from the full duplex point to point inter-PU links. This allows any PU to write to either scratch SRAM. The main advantage of this multiplexing model is that it provides flexibility in steering data from the memories to the appropriate execution unit inside each of the PUs and similarly to steer data back to the scratch memories.

The common 5:1 mux's reflect performance goals. In order to evaluate the full system behavior of this architecture, we have also designed a reference microprocessor that runs at 300 MHz in our .25μm CMOS process. The cluster was designed to meet this target. Larger multiplexors would reduce the frequency and smaller multiplexors would reduce the generality which in turn limits program mapping options and adversely impacts performance.

Each PU consists of three integer execution units (XUs), illustrated in Fig. 3 and an 8 entry register file with 2 read and 2 write ports. The XUs, ALU0, ALU1, and MS work as follows.

Fig. 1. Coprocessor organization

Both ALU's have single cycle latency and support add, subtract, and AND. ALU0 also supports byte-select and a stylized bit-word multiply used in a stylized complex number multiplication common in 3G algorithms. ALU1 also supports compare-select and XOR. The MS unit provides either multiply or shift. This combination reflects the fact the multiply and shift are rarely simultaneously needed. XU's take 2 inputs and provide a single output.

The outputs of the 5:1 muxes are registered. Each XU receives its inputs from these pipeline registers. When viewed individually the sources feeding each mux is somewhat confusing. However when viewed as a pair, the view becomes consistent. Namely each XU can be fed by any of the 5 XU outputs (2 register file, 1 each from the 2 ALUs, and 1 from the multiply-shift), any of the 3 point to point inter-PU links and 2 of 4 possible busses. At first glance the ability to get inputs from only half of the busses would appear to be a defect. However each bus is driven by a 5:1 mux as well. This two-level multiplexing strategy in driving input operands to the appropriate XU removes the disadvantage. The result is a rich and reasonably general data routing capability which when controlled by the micro-code allows a wide variety of applications to be efficiently mapped onto the architecture. As with any general purpose device, an inefficient mapping still runs but will be inefficient in terms of performance and power consumption.

The use of two ALUs in each PU and four PUs in the coprocessor seems to be a good balance in terms of executional resources to communication and storage resources. Further increase in the number of ALUs or PUs does not generally increase performance, but suffers the disadvantage of increasing the capacitance of the PU inputs and increase the multiplexing delay on the PU outputs. Furthermore, the communcation paths and limited memory ports cannot support the bandwidth required to efficiently feed the additional compute resources.

The input, output, and scratchpad SRAMs are all 16-bits wide. The scratch and output SRAMs each have 1024 entries, while the input SRAM has 2048 entries. However the input SRAM is operationally organized as two 1024 word banks. In general, these banks are used in a classical double buffer fashion, with one actively in use by the coprocessor while the other is being written by the host. The SRAM capacities are influenced by the frame sizes that were chosen to test the 3G baseband algorithms. In our $.25\mu$ CMOS process, leakage power is very small. SRAM power consumption is therefore more dependent on the number of sense amps than on memory capacity. The only wide memory is the micro-code SRAM which is currently a 288-bit wide (unoptimized) 128-word single port synchronous memory.

Operationally, large numbers of low-level physical resources (muxes, ALUs, registers, multiple small memories, etc.) are controlled by a 288-bit wide horizontal micro-code word. The microcode controls everything on a per clock basis: data steering, register load enables, operation selection, address mode, and memory module selection. The flexibility of software control of finegrained physical resources allows a wide variety of algorithms to be mapped onto this architecture. The specific structure of the algorithm will result in more or less efficient mappings. The only show stopper is if the algorithm requires an operation type that isn't supported by the cluster, e.g. floating point operations or integer divide. The structure supports a rich set of options for steering data, setting up computational pipelines either between PUs, within a PU or both. The execution resources are somewhat modest by comparison, but the fact that they can be rapidly configured into custom execution pipelines will be shown to have a significant advantage in terms of energy and performance.

3. RESULTS

Energy and performance numbers for the coprocessor are calculated using Synopsys Nanosim, a commercial Spice level circuit simulator, on a fully synthesized and back-annotated $.25\mu$m Verilog- and Module Compiler-based implementation. A full clock tree and worst case RC wire loads are included in the simulated model. For each of the benchmarks, the microcode corresponding to the benchmark is loaded into program memory and the circuit is simulated in Nanosim for the duration of several input packets. The RMS current reported is used to calculate energy consumption.

In all of the algorithms, the wide instruction SRAM is accessed on every cycle, hence the instruction fetch power is consistent across the benchmarks. Transpose FIR has very light data storage requirements, as there are enough internal registers in the cluster to store both intermediate values as well as the necessary coefficients. The only data read or stored are the input and output values, once every 4 cycles. Dotp_sqr consumes 4 data inputs per cycle. Hence transpose FIR and Dotp_sqr represent the two opposite ends of the spectrum in terms of minimal and maximal data SRAM power requirements.

ASIC and general-purpose processor numbers are presented in Table 1, where all of the power numbers have been normalized to an $.18\mu$m process. These values were scaled by feature-size λ using the the method described by Gonzalez and Horowitz [4], using a conservative[1] exponent of 2. Energy and performance numbers for the ASIC versions of $.18\mu$m turbo and RAKE are taken from published results [5, 2]. Both these implementations differ slightly from the cluster and XScale versions. In particular, the FlexRake implementation employs an 8-bit datapath rather than the 16-bit datapath used in our implementations. The turbo ASIC utilizes a look-up table based correction that is not implemented in our work. The implementations on the cluster and the algorithms measured on the XScale are the same.

The transpose FIR, Vecmax, Matmult, and Dotp_sqr ASIC implementation were done by us using a $.25\mu$m Verilog implementation using Nanosim. The transpose FIR ASIC utilizes 16 multipliers and 15 adders that work in parallel to compute one output per cycle in a straightforward fashion. Vecmax computed a max of four independent streams in parallel and combines the four results at the end of a packet. Matmult operates by doing one eight word vector by eight word vector multiply per cycle in a pipelined fashion, using eight multipliers. Dotp_sqr also uses eight multipliers. Four are used to compute the dot product, and the other four compute the square. The multiplier results are summed and finally accumulated in a pipelined fashion.

Energy and performance numbers were taken from measurements on a low-power 400MHz $.18\mu$m Intel XScale (StrongARM) PXA250 system. This system has been augmented to permit the measurement of the average current consumed by the processor and memory module via a digital oscilloscope and a non-intrusive current probe. Each of the algorithms studied has been implemented in C/C++ and compiled with the GNU GCC/G++ compiler at optimization level O3, with loops unrolled.

---

[1]By way of comparison, the FlexRake ASIC implementations described [2] exhibit a scaling exponent of approximately 3.18. The Gonzalez and Horowitz paper [4] recommends a conservative value of 2 and an aggressive value of 3.

Benchmarks	ASIC		Cluster		StrongARM	
	power	datarates	power	datarates	power	datarates
T-FIR	321mW	300MS/s	385mW	75.0MS/s	1370mW	3.33MS/s
Rake	1.55mW	47.0MC/s	486mW	534MC/s	1360mW	5.72MC/s
Turbo	292mW	2.00Mb/s	643mW	0.897Mb/s	1330mW	0.0163Mb/s
Vecmax	15.5mW	8.82Mvec/s	481mW	8.82Mvec/s	1330mW	0.511Mvec/s
Matmult	87.2mW	4.69Mmat/s	456mW	2.08Mmat/s	1370mW	0.160Mmat/s
Dotp_sqr	119mW	9.38Mvec/s	554mW	4.62Mvec/s	1330mW	0.256Mvec/s

Table 1. Power dissipation and data rates scaled to .18μm process

Table 2 compares the energy-delay product overhead of the coprocessor and XScale implementations with respect to the baseline ASIC implementations. In general, the coprocessor is within one to two orders of magnitude of the energy-delay of an ASIC implementation and is three to four orders of magnitude more efficient that a general-purpose processor implementation.

The coprocessor is capable of sustaining data rates 5-80 times that of the general-purpose processor. When compared to the ASIC, the coprocessor performance varies. The ASIC version of RAKE is tuned to run at a particular correlation rate. The coprocessor version is capable of running at 11.4 times the speed of the ASIC version. In this case, the coprocessor architecture could be used to process tasks between batches of packets, it could be powered-down to save power, or the design could be voltage scaled to reduce power. For the transpose FIR, the ASIC is not tuned to any particular data rate, but was synthesized for performance. The fact that a coprocessor cluster can only sustain 1/4 of the performance of the ASIC implementation is not surprising, in that it has 1/4 the number of multipliers. For turbo, the coprocessor implementation can sustain data rates at just under half that of the ASIC in a single cluster. If higher data rates are necessary, multiple clusters can be combined to improve the combined data rate.

4. CONCLUSIONS & RELATED WORK

ASICs have traditionally been used in the implementation of wireless applications that simultaneously require high performance and power efficiency [6, 7]. Texas instruments has improved the performance of their DSP processors by adding application specific hardware support for Viterbi/Turbo decoding and chip despreading [8]. Fine-grained reconfigurability can be achieved by using FPGAs and ASICs as coprocessors for DSP processors [9]. While this approach provides more options for mapping the application onto the architecture, FPGAs have much lower performance and consume much higher power than an ASIC approach. The cluster approach provides more rapid reconfigurability as well as increased performance and reduced power consumption than is available from an FPGA approach.

Fine-grain software control of clustered execution units has been shown to have a significant performance and energy advantage over conventional processors and the performance-efficiency of this cluster approach is close to what can be provided by specialized ASIC implementations but without sacrificing generality. It is important to note that this work was initially motivated by

the need to support the processing demands of 3G wireless standards in a thermal budget commensurate with the embedded mobile processing domain. It became apparent that the idea was even more general than we had intended. At this point we investigated more conventional DSP algorithms and found that the generality claim was in fact true. The result is an architecture that is powerful enough to support the severe real-time processing demands of 3G cellular algorithms, yet flexible enough to be utilized for a broad range of other signal processing duties. The energy-delay product of this design has been shown to improve on that of the Intel XScale by three to four orders of magnitude and is within one to two orders of magnitude of a custom ASIC. The architecture is capable of a sustained performance improvement of 5-80 times that of a general purpose embedded processor.

The current problem with this effort is that manually mapping and scheduling applications onto the cluster is tedious and time consuming. The next step, to automate this process, is in progress.

5. REFERENCES

[1] J. Rabaey, "Beyond the third generation of wireless communications, keynote presentation," in *IEEE International conference on Information and Communications security*, Dec 1999.

[2] L. Harju, M. Kuulusa, and J. Nurmi, "A flexible Rake Receiver Architecture for WCDMA mobile terminals," *IEEE Workshop on Signal Processing Systems*, pp. 177–182, Oct 2002.

[3] M. Marandian, J. Fridman, Z. Zvonar, and M. Salehi, "Performance analysis of turbo decoder for 3GPP standard using the sliding window algorithm," *Personal, Indoor and Mobile Radio Communications*, vol. 2, pp. 127–131, 2001.

[4] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose processors," *IEEE Journal of Solid State Circuits*, pp. 1277–1284, Sept 1996.

[5] Bickerstaff, M. Garrett, D. Prokop, T. Thomas, C. Widdup, B. Gongyu Zhou Nicol, and C. Ran-Hong Yan, "A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18um CMOS," *IEEE International Solid-State Circuits Conference*, pp. 124–451, 2002.

[6] S. Morris, "Signal processing demands shape 3G Base Stations," *Wireless Systems Design Magazine*, Nov 1999.

[7] G. Masera, G. Piccinnini, M.R. Rock, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Transactions on VLSI Systems*, Sept 1999.

[8] "Channel card design for 3G infrastructure equipment," Tech. Rep., SPRY048, Texas Instruments, 2003.

[9] H. Blume, H. Hubert, H. T. Feldkamper, and T.G. Noll, "Model-based exploration of the design space for heterogeneous systems on chip," in *IEEE International Conference on Application-specific Systems, Architectures and Processors*, July 2002, pp. 29–40.

Benchmarks	Cluster	StrongARM
T-FIR	19.2	34600
Rake	2.43	59300
Turbo	10.9	68300
Vecmax	31.1	25600
Matmult	26.5	13400
Dotp_sqr	19.1	14900

Table 2. Energy-Delay inefficiency with respect to ASIC