ACT: A Low Power VLIW Cluster Coprocessor for DSP Applications

Ali Ibrahim, Al Davis, Mike Parker

School of Computing, University of Utah
Salt Lake City, UT 84112
aibrahim@ttb.siemens.com, ald@cs.utah.edu, map@cray.com

Abstract

The ACT (Adaptive Cellular Telephony) coprocessor architecture is described and analyzed using a set of widely used DSP algorithms. Performance and power are compared to equivalent implementations on ASIC and embedded processor platforms. Flexibility is achieved by fine-grain program control of communication and execution resources. Compression techniques, simple addressing modes for large single-ported distributed register files, and configurable address generation units provide performance and energy efficiency. An energy-delay reduction of two to three orders of magnitude is achieved when compared to a conventional embedded processor such as the Intel XScale.

1 Introduction

Future digital processing algorithms for portable devices require both energy efficient performance and flexibility. The traditional approach for applications requiring both performance and low-power is to employ ASICs for compute intensive components. In areas where applications evolve rapidly, flexibility is also an important factor and a general purpose or embedded processor approach has often been used for this reason. For applications such as wireless communications, voice, and video processing: ASICs are too inflexible and costly; low-power processors do not have sufficient computational power; and general purpose processors consume too much power. This situation motivates this investigation of an alternative approach.

The ACT architecture takes advantage of the stream oriented nature of wireless communication applications, in particular 3G and 4G cellular telephony, which provide the impetus for this work. The majority of the processing is dominated by regular inner loops which process streams of input signal data. A similar situation exists for speech processing, encryption/decryption, and media encode/decode. Operations in these codes can be classified into two categories: useful and overhead. Useful operations perform the computational actions, while overhead operations consist of branch and address generation operations.

In [10], we presented the architectural basis for this approach which retained much of the generality of a traditional processor while achieving energy and performance characteristics close to that of an ASIC implementation. This earlier approach employed multi-ported distributed register files and multiple scratchpad SRAMs to feed a cluster of execution resources that are embedded in a rich communication structure. Fine-grain control is orchestrated by a wide-word horizontal microcode program. This 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 effectively be instantly reconfigured to support a new processing phase. Algorithms which map poorly onto the communication and execution resources still run, but at reduced efficiency. A drawback of this previous effort is that the wide instruction memory potentially consumed approximately 50% of the total active power.

This paper introduces improved architectural features which further enhance performance and which reduce the power consumption of the instruction memory. The changes include: a compression strategy for instructions; single read-write ports instead of multi-ported distributed registers; simple address modes were added to these registers which provide a renaming capability, thereby removing the traditional need for loop unrolling and a better form of rotating registers [6]; flexible stream address generators (AGUs) which reduce the instruction width; and a hardware loop unit which reduces branch overhead. Space limitations prevent describing all of the new features in detail. Therefore, this paper describes the general architecture and analyzes the benefits of the compression technique, the interconnect and data forwarding strategy, and the new single-ported distributed register files.

2 Architecture Description

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. This allows the coprocessor to handle compute intensive tasks while the host processor is responsible for coarse grained copying of the input and output frames.

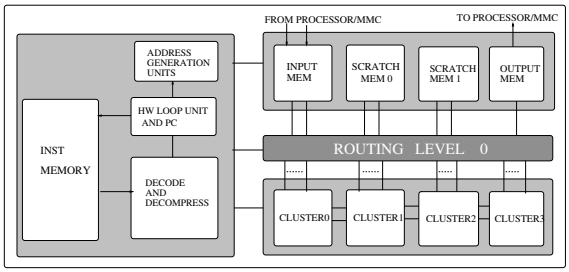

Figure 1: Coprocessor Organization.

The coprocessor core is divided into storage, four execution clusters, address generators, hardware loop unit, and the decode/compression 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. Both ALUs have single cycle latency and support add, subtract, AND, and OR. ALU0 also supports byte-select and a stylized bit-word multiply used in the complex number multiplication common in 3G algorithms as explained in the next paragraph. ALU1 also supports compare-select and XOR. The MS unit provides either multiply or shift. The MS combination reduces multiplexer width and reflects the fact that multiply and shift are rarely simultaneously needed. XUs take 2 inputs and provide a single output.

Complex bit-word multiplication is commonly used in 3G wireless algorithms such as the rake receiver, channel estimation, and cell search. These operations in general purpose processors (GPPs) can be calculated by the use of an if/else condition. This approach limits instruction level parallelism (ILP) and increases energy consumption due to the need for a wider data-path. In this architecture, this problem was resolved by the use of a semi-reconfigurable ALU0. Figure 3 shows the design of ALU0 which is the same in all clusters. This unit has 2 special registers that contain the configuration bits. For complex multiplication, the input bits are first loaded as a group of 16 bits into the ALU0 special configuration registers and used later as opcodes and which effectively transform bit-word multiplication into addition and subtraction.

The choice of two ALUs in each cluster and four clusters in the coprocessor provides a good empirical balance between execution, communication, and storage resources. Increasing the number of ALUs or clusters does not generally increase performance, but suffers the disadvantage of increasing the capacitance of the cluster inputs and increases the multiplexing delay on the PU outputs. More importantly, the communication paths and limited memory ports cannot support the bandwidth required to efficiently feed additional compute resources.

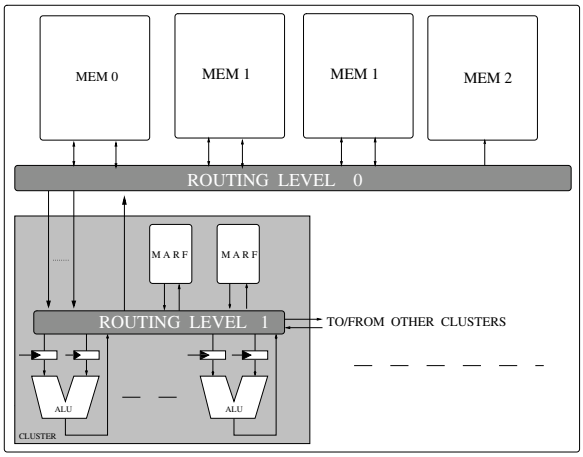

Figure 2: Cluster Organization.

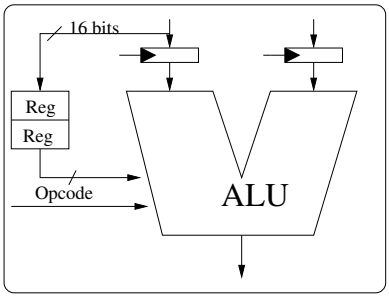

Figure 3: ALU0.

In application-specific embedded systems, such as 3G cellular telephony, a significant number of variables produced by a particular function unit are consumed almost immediately by another function unit. Hence these value lifetimes span only a few instructions. In [10], more than 80% of such values were consumed immediately in algorithms such as the turbo decoder and rake receiver. Directly forwarding these values decreases power consumption by eliminating the corresponding register file accesses. To efficiently forward the data between different stages of the pipelines, two routing levels are used as shown in Figure 2. Routing level 1 is used for inter- and intra-cluster communications. Routing level 0 is used for inter-cluster and memory communications.

In routing level 1 each XU input is connected to a 5:1

mux. The outputs of the 5:1 mux's are registered. Each XU receives its inputs from these pipeline registers. When viewed individually, the sources feeding each mux are confusing. A consistent view is evident for the pair. Namely each XU can be fed by any of the 5 XU outputs (2 register files, 2 ALUs, and the MS), any of the 3 point to point inter-cluster links and 2 of 6 possible busses. At first glance, the ability to get inputs from only one third of the busses would appear to be a defect. However each bus is driven by a 6:1 mux located in routing level 0. This two-level multiplexing strategy in driving input operands to the appropriate XU removes the disadvantage [1]. 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.

Register files are clear thermal hot spots in modern processors since they are both the source and sink for most operations. In Imagine [17], it has been shown that for N functional units, the power of a centralized register scales as N^3, where N is the number register ports. Power scales linearly with the number of registers. Power in Imagine is reduced by using distributed register files with single read and write ports. The read ports are mapped on a one to one basis to function unit input ports. The output ports, on the other hand, drive busses which are connected to the register write ports.

The register file strategy for our design is similar to Imagine, but both the number of read and write ports have been reduced. This reduces the bandwidth available to the execution resources but the potential disadvantage is avoided by using a richer communication structure which enables forwarding, and multiple scratchpad memories. This architecture employs two 32-entry register files in each of the 4 clusters.

Each register file is split into the two adjacent windows shown in Figure 4. The size of the windows are explicitly allocated under program control by setting a tail pointer in WP (Window Pointer) Unit to the first window . The second window is addressed normally as a static register file and the first window is effectively accessed like a rotating register file using simple address modes in the address generator unit for the register file (AGURF). Each AGURF has two pointers and performs increment and decrement by a constant with wrap around, and modulo addressing to create a circular buffer. This is particularly useful in filter algorithms. In the algorithms we have tested, two pointers in each AGURF are sufficient. If the data in a register file needs to be accessed with a different stride and base pointer, a new pointer can be loaded from the instruction or from the static portion of the register file, since each AGURF is connected to the output of the register file (not shown in the figure).

Each modally addressed register file (MARF) tail pointer can be individually specified. This organization allows significant register file specialization for different computation phases, provides hardware support for rotating registers and thus decreases the number of instructions associated with loop unrolling, and reduces the instruction width. This reduces instruction memory power consumption.

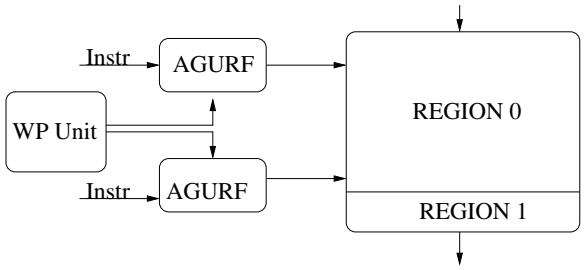

Figure 4: MARF.

The hardware loop unit controls loops to eliminate the extra computational overhead associated with branches and loop increments. These techniques have been employed in many commercial processors, e.g. [15]. The main differences are in the number of supported loops. This design supports a depth of 4 since that is sufficient for a wide range of wireless communication and general DSP algorithms.

Since the computation in our target applications are dominated by regular inner-loop calculations, address calculations on a separate address generation unit (AGU) can significantly improve parallelism since the execution units of the coprocessor do not need to compute addresses. In designing the AGUs we have followed a similar approach to that of the other XUs in the execution clusters. In this approach, each AGU has a number of function units and mux's. The mux's are configured every cycle to support the requisite data flow for address calculations. However this approach, by itself, requires increased instruction bits for AGU opcodes, constants, and mux selects. This approach contains more generality than is commonly useful. For instance, most of the mobile 3G wireless baseband algorithms operate on one dimensional data frames. Image and video algorithms operate on two dimensional data. Since the focus is on 3G algorithms, the AGUs were designed to accelerate one dimensional data accesses and unit strided two dimensional data accesses. In order to support more complex address calculations such as non-unit-strided two dimensional addresses, configuration tables have also been added. In this technique, each AGU has a four entry table. Each entry contains the necessary bits to control the AGU in the complex mode. The result provides simple addresses to an arbitrary number of regular streams, and complex address patterns for up to four streams per AGU and does not require additional instruction bits that would be necessary

[1] Merging routing level 0 and level 1 would increase the cycle time of the coprocessor.

to directly support complex address patterns.

The micro-code controls everything on a per clock basis: data steering, register load enables, operation selection, address mode, and memory module selection. The flexibility of fine-grained software control provides efficient execution of a wide variety of algorithms. However, the power consumed by the wide instruction word can be a problem. Both the architecture and the regularity of the stream based codes present an opportunity to employ compression techniques (Section 3).

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. The input SRAM is organized as two 1024 word banks. In general, these banks are used in a double buffered fashion, with one buffer 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 width of the memory (e.g. number of sense amps) than on memory capacity.

Clock gating is used to limit power consumption of unused components. Since all internal pipeline registers of the coprocessor have load enable signals controlled by the program, they effectively provide clock gating for logic not used in any particular cycle.

3 Microcode Compression

Compression generally exploits redundancy. Various techniques emerged for RISC processors and were then applied for VLIW architectures [11]. However, memory reduction rather than energy dissipation was the driving force behind these algorithms. More limited approaches have focused on energy reduction. In [3] the decompression block was placed between the main memory and the processor, and hence did not show the impact of compression on the processor. In [12], the processor was included in the analysis by placing the decompression block between the cache and the CPU. A 19% improvement for data intesive algorithms was achieved. However power estimates were derived from an analytical model rather than low-level circuit details.

In [10], we observed that computations in the target applications exhibit two forms of program parallelism, ILP and data level parallelism (DLP). VLIW processors primarily improve ILP by exploiting DLP. SIMD architectures directly reduce instruction width by applying a single instruction directly on multiple data.

Two decompression techniques were tried as shown in Figure 5. The first includes a small local instruction memory that contains the most frequently executed instructions of modulo scheduled loops [16]. Block A in Figure 5 represents the small local instruction memory. The size of this local instruction memory (LIM) is 32 to match the application requirements. Algorithms with higher requirements will not be fully compressed. The compressed instruction simply indexes the common code in the LIM. The size of the indexed code is contained in the microinstruction. This number is used in a counter, Block B, which controls access to the common code in the LIM.

The result is that there are two types of instructions: long and compressed. One third of the long instruction is enough to address the entire LIM. The instruction memory is therefore divided into 3 banks. The number of accessed banks depends on the instruction type determined by block E in Figure 5. The memory generator tool (a commercial tool provided by Artisan) shows that the power savings of this strategy on average is limited to 20%. To improve on this, an additional approach that exploits the SIMD presence in these algorithms was also studied. The long instruction, in this case, could potentially be reduced to a single short instruction that operates on multiple data. This short instruction is then stored in one of the banks which is decoded in block C [2] whenever the instruction is accessed. The number of SIMD instructions is currently limited to eight to avoid increasing the cycle time of the decompression stage. Finally, the 3:1 mux in Figure 5 selects between the outputs of the SIMD, LIM, and block D which forwards the uncompressed instructions. The results of the SIMD compression will be presented in Section 5

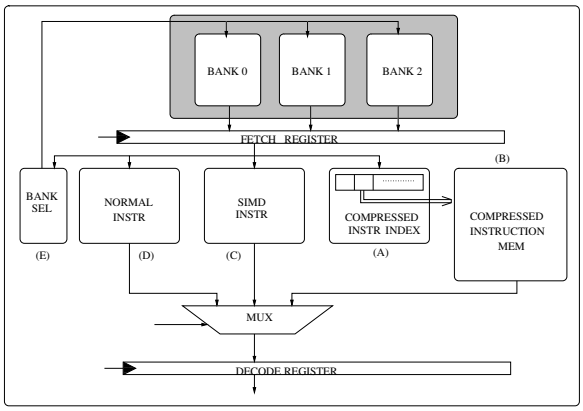

Figure 5: Compression.

4 Algorithms

Several DSP algorithms have been mapped to the coprocessor to test its performance, energy efficiency, and flexibility.

[2] Placing this block at the output of LIM would increase the cycle time of the decompression stage

Rake: Complex correlation on bits and words are used often in 3G wireless algorithms. The Rake receiver and cell search [1] are examples of these algorithms. The primary calculations are complex multiply-accumulate operations expressed as:

$for(i = 1; i < SF; i++)$
$S = S + IN[i + delay] * PN[i]\}$; [3]

This equation represents a correlation for a particular propagation path. Typically, correlations over multiple paths (e.g. 4) need to be performed in 3G application suites.

Finite Impulse Response (FIR): a common filtering algorithm implemented by using a series of delays, multipliers, and adders to create the filter's output.

Transpose Finite Impulse Response (TFIR): Another common form of FIR filter.

Dot product(dotp): Computes the dot-product of two vectors of size N elements.

Vector Product and Square(dotp_sq): takes two vectors $\mathbf{V1}$ and $\mathbf{V2}$ and produces two dot products $\mathbf{V1} \cdot \mathbf{V1}$ and $\mathbf{V1} \cdot \mathbf{V2}$. This can be used to compute G in the Vector-Sum Excited Linear Prediction [7] (VSELP) coder.

Maximal Element of a Vector(Vecmax): Finds the maximum value in a vector of size N.

Matrix Multiplication(Matmult): Performs the multiplication between two 8x8 matrices.

Strided Matrix Multiplication(STMatmult): Same as Matmult, however rows and columns of the matrices are accessed in a strided fashion.

Sum of Absolute Differences(SAD): It is useful in the block match kernel of an MPEG-2 encoder. This algorithm operates on a pair of 8 bit inputs. It produces the sum of the absolute difference between corresponding pixels of the two 16x16 macroblocks as shown in the following pseudo-code:

$for(i = 1; i < 16; i++)$
$for(j = 1; i < 16; i++)$
$diff = block0[i][j] - block1[i][j]$
$if(diff < 0)$
$diff = -diff$
$sad+ = diff$

5 RESULTS

The coprocessor power and performance were measured using using Synopsys Nanosim, a commercial Spice level circuit simulator, on a fully synthesized and back-annotated .25μm Verilog- and Module Compiler based implementation. A full clock tree and worst case RC wire loads are included in the model. All algorithms were first written in a high level stream oriented language and then modulo scheduled. Conflicts on register file write ports can be resolved by storing the data in the post-mux registers for any number of cycles since their load-enable lines are under program control. This effectively creates a distributed set of single entry registers files. The scheduled code is then compressed using the above mentioned techniques. The microcode corresponding to each benchmark is then loaded into the instruction memory and the LIM. The circuit is then simulated in Nanosim for the duration of several input packets. The RMS current is used to calculate energy consumption.

In the general purpose processor case, energy and performance numbers were taken from measurements on a low-power 400MHz .18μm 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.

The FIR, TFIR, Dot Product, Vector Product and Square, Maximum Value of a Vector, SAD, matrix multiplication and strided matrix multiplication ASIC implementations and power numbers were developed using a .25μm Verilog implementation using Nanosim. The Rake ASIC implementation and energy numbers are taken from the literature [9].

Two types of experiments were performed: uncompressed mode, and compressed mode. In the uncompressed mode, the main instruction memory is accessed on every cycle and hence the power dissipation due to instruction fetch is the same for all benchmarks. In compressed mode, both the LIM and SIMD decompression units were used. This decreases power depending on the compression type.

Figure 6 illustrates the energy-delay product of the coprocessor and XScale, normalized to the ASIC (no compression). Power numbers have been normalized to an .18μm process. These values were scaled by the feature-size λ using the method described by Gonzalez and Horowitz [8] at their conservative value of 2.

Figure 6 shows that the coprocessor energy-delay product varies between one and two orders of magnitude in the uncompressed mode compared to the XScale. The data rate (shown in Figure 7) sustained by the coprocessor, on the other hand, varies. When compared to an ASIC for TFIR, the coprocessor is able to achieve almost 1/4 the performance of the ASIC (Note: the ASIC has 4 times as many multipliers). In the FIR case, on the other hand, the coprocessor is capable of sustaining 1/6 the performance of the ASIC. This was expected since the FIR implementation requires data transfers between the register files in different clusters. Such a transfer requires the use of an XU since the register files are not directly connected. The effective data transfer rate will decrease for filters with a higher number of taps. For Rake, the coprocessor runs at 11 times the speed of

[3] S, IN and PN are complex numbers. PN are bits, and IN are words.

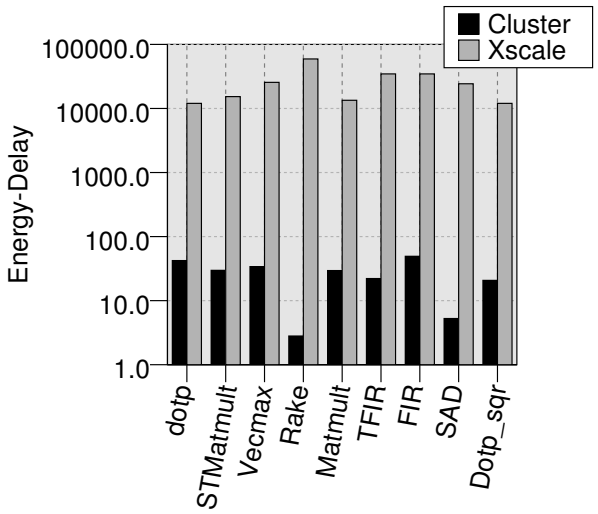

Figure 6: Energy-delay of the cluster and the Xscale with respect to ASIC (cluster/ASIC,no compression) (Xscale/ASIC)

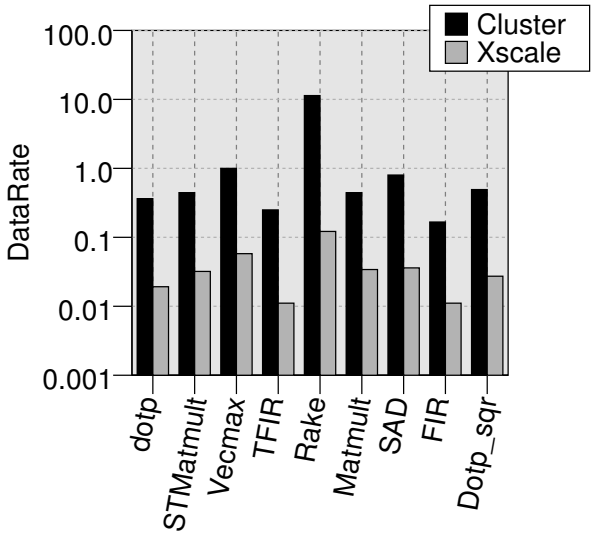

Figure 7: Data Rate of the cluster and the Xscale with respect to ASIC

the ASIC. This particular ASIC was tuned to barely exceed real-time performance. The excess performance of the coprocessor provides the opportunity to work on another task or it can be powered down. For SAD, the coprocessor and the ASIC have almost the same data rate since the ASIC was designed to perform four SAD operations per cycle.

FIR, TFIR have very regular addressing modes, and high data reuse which makes them a good candidate for the MARF. The only data read or stored from/to memory are the input/output values once every N cycles (N equals 4 for a filter with 16 coefficients). Vector product and square, on the other hand consume 4 data inputs per cycle and does not present any data reuse opportunity.

Figure 8 shows that the instruction memory energy savings due to compression ranges between 17 and 38%. Energy savings for most of the algorithms, excluding FIR and TFIR, range between 17% and 21% and are due to the small instruction memory. Energy savings for FIR and TFIR were 27% and 38% and are mostly due to the SIMD compression. Taking advantages of SIMD compression in this architecture was difficult since opcodes and mux selects in all routing levels have to be the same. Since FIR and TFIR operate on data stored in the local register files, SIMD scheduling is straightforward.

6 CONCLUSIONS & RELATED WORK

Current embedded applications require architectures that are flexible, high-performance, and consume minimal power. Neither ASICs or existing general purpose processor strategies are adequate. ASICs lack flexibility; embedded processors do not provide the requisite performance; and performance microprocessors consume too much power. Hybrid approaches exist. Texas Instruments has improved the performance of their DSP processors by adding application specific hardware support for Viterbi/Turbo decoding and chip despreading [1]. Fine-grained reconfigurability can be achieved by using FPGAs and ASICs as coprocessors for DSP processors. 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.

Research in coarse-grained reconfigurable architectures focuses on the design of general purpose systems which can be efficiently applied to particular applications [5]. For instance, the Pleiades project at UC Berkeley [2] focuses on an architectural template for low-power, high-performance multimedia computing. The Pleiades design methodology assumes a very specific algorithm domain. The Chameleon system [4] has also introduced a reconfigurable communications processor. The system delivers high performance, however it was designed for base station algorithms where power consumption is less critical than in the handset.

The key in designing for low power, high performance, and flexibility relies on finding opportunities for customization for a particular domain. There could be a high number of parameters involved in this process (memory system, single- vs. multi-cluster, bypass logic, register files, compression, and function unit design). Each of these param-

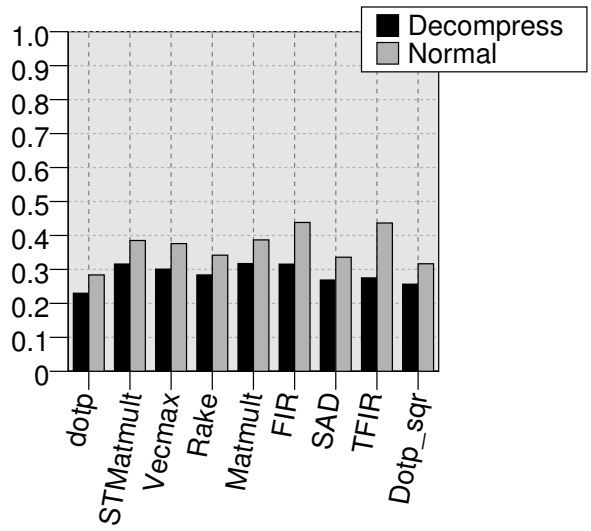

Figure 8: Instruction power reduction due to compression

eters can have a big effect on the performance, power, and flexibility.

VLIW architectures have been adopted for signal processing applications for their power and performance advantages. These architectures are simple extensions of single-issue load-store architectures. Hence, every function unit port has a dedicated register file port. The register file size, number of ports, and limitations in the communication interconnect have a major effect on the performance and power dissipation. The results in [10] shows clearly how ACT significantly outperforms many VLIW DSP processors [13, 19, 14] such as the ST120 from STMicroelectronics [19]. Note that the energy-delay product is a more relevant metric, but the literature does not provide these data for all our benchmarks.

In this paper, a multiple cluster fine-grain VLIW architecture has been described. The architecture was first designed for the wireless domain, but has been shown to be flexible for many other digital signal processing algorithms. The benefits of clustering, register files, distributed memories, compression techniques, and data forwarding have been analyzed.

In [17], the number of write ports was decreased by using shared busses. In [18], power consumption was reduced by forwarding data using bypass logic instead of accessing the register file. In this paper, both read and write ports are reduced which potentially limit register bandwidth. This potential defect is avoided by the use of a multi-level program-controlled interconnect structure. A scheduler minimizes routing conflicts to efficiently map the program onto the communication and execution resources.

The disadvantage of such a fine grained controlled VLIW approach is a wide instruction width which consumes too much power. Compression techniques have been presented which reduce this problem by 17% to 38%.

References

[1] Channel card design for 3G infrastructure equipment. Technical report, SPRY048, Texas Instruments, 2003.

[2] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. M. Rabaey. Evaluation of a low-power reconfigurable DSP architecture. In *IPPS/SPDP*, pages 55–60, 1998.

[3] L. Benini, A. Macii, E. Macii, and M. Poncino. Minimizing memory access energy in embedde systems by selective instruction compression. *IEEE Transactions on VLSI Systems*, 10(5), October 2002.

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

[5] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.

[6] G. Doshi, R. Krishnaiyer, and K. Muthukumar. Optimizing software data prefetches with rotating registers. *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages p.257–267, Sept 2001.

[7] I. Gerson and M. Jasiuk. Vector sum excited linear prediction (VSELP) speech coding at 8 kbits/s. *Proc. ICASSP*, pages 461–464, 1990.

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

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

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

[11] J.S.Parakash, P.Shankar, and Y.N.Srikant. A simple and fast scheme for code compression for VLIW processors. *Data Compression Conference*, 2003.

[12] H. Lekatsas, W. Wolf, and rg Henkal. Code compression for low power embedded system design. *37th Conference on Design Automation (DAC'00)*, June 2000.

[13] K. Loo, T. Alukaidey, S. Jimaa, and K. Salman. SIMD technique for implementing the Max-Log-MAP algorithm. *GSPx*, 2003.

[14] 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*, 2:127–131, 2001.

[15] P.Lapsley, J.Bier, A.Shoham, and E.A.Lee. DSP processor fundamentals: Architectures and features, chapter 8. *IEEE Press series on Signal Processing*, 1997.

[16] 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.

[17] S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens. A bandwidth-efficient architecture for media processing. In *International Symposium on Microarchitecture*, pages 3–13, 1998.

[18] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon. Exploiting data forwarding to reduce the power budget of VLIW embedded processors. *Proceedings of the conference on Design, automation and test in Europe*, pages 252–257, 2001.

[19] STMicroelctronics Inc. http://www.stm.com/.