Performance Optimization with Scalable Reconfigurable Computing Systems

Rama Sangireddy, Prabhu Rajamani
Dept. of Electrical Engineering
University of Texas at Dallas
Richardson, TX 75080, USA
{*rama.sangireddy,pxr042000*}*@utdallas.edu*

Shwetha Gaddam
Dept. of Electrical Engineeing
Osmania University College of Engineering
Hyderabad, AP, India
shwetha.gaddam@gmail.com

***Abstract*— *The total time to execute an application, the energy consumed, and the flexibility to manage a large set of applications are among the most important performance parameters used to measure the quality of a computing system. Superior architectures with flexible reconfigurable arrays lead to innovation beyond the limits of traditional silicon. The incorporation of on-chip reconfigurable computing elements generally improves execution time. However, the amount of energy consumed to deliver the required level of performance is an important consideration, to prolong the battery life in portable and mobile devices. In this paper, we have proposed and designed a novel scalable array architecture and explored the performance and energy trade-offs for various applications by scaling various system parameters like hardware resources, operational granularity, and voltage supply. The scalable coprocessor design for mapping Discrete Cosine Transform (DCT) is implemented with 8 taps resulting in an area of 0.0024mm^2 at 0.18μ technology. The coprocessor to run 16 taps of convolution function results in an area of 0.0099mm^2, while a 256 tap convolution function is designed at an area cost of 0.1585mm^2. When the MPEG decode application is executed in the proposed architecture, with the DCT function computed in the scalable coprocessor, the total execution time is reduced to around 24%, and the energy consumed is reduced to around 28% of that consumed in the base architecture without a coprocessor. Further, as the coprocessor's supply voltage is scaled down from 1.8 to 1.0 volts at 0.18μ technology, the relative total execution time varied only slightly (from 23.65% to 24.78%), while resulting in considerable reduction in the energy consumed (from 28.12% to 23.8%). For the FIR application, energy consumption reduced up to 36% when hardware resources are scaled and up to another 12% when voltage is scaled, while execution time reduced up to 50% when hardware resources are scaled and increased up to 15% when voltage is scaled. The study also reveals interesting performance patterns for various applications ike CJPEG, MPEG decode/encode, FIR, and IIR, depending on the their characteristics.*

I. Introduction

To bring adaptability, scalability, and performance effectiveness to the hardware computing systems in processing a wide range of applications, one of the recent trends has been to integrate both spatial reconfigurable compute engines and temporal structures, like general purpose processor (GPP) elements, into a single chip. The compute-intensive portion of an application is mapped onto the spatial compute engine so that the data flows directly from source to sink, as shown in Figure 1(a). For the remaining portions of the application that are rarely repetitive and are not compute-intensive, configuring the reconfigurable hardware for every small segment of computations becomes a bottleneck, and the reconfigurable device may get bogged down. Hence, such portions of code are preferred to be processed in the temporal structure as shown in Figure 1(b). In fact, current field programmable gate array (FPGA) architectures have started embedding one or more RISC based GPP cores in the reconfigurable fabric.

The portions of the application that are not compute-intensive are decomposed to the granularity of a typical micro-instruction, and are processed in the temporal GPP engine. The compute-intensive portions are decomposed into configurations of a particular granularity, and such configurations are executed in the reconfigurable spatial hardware. Though the total amount of hardware resources placed on chip is fixed, the amount of hardware resources available for a particular task is not known a priori in a multitasking environment where the system-on-chip (SOC) is expected to simultaneously execute multiple processes/threads. The total application execution time depends on the ability of the system to decompose a configuration to a desired granularity in real-time based on the available amount of hardware and its flexibility to map such configuration. The smaller the amount of on-chip reconfigurable hardware that is available, the smaller will be the configuration granularity. In such case, there will be more configurations of the hardware, and so longer execution time. The larger the configuration size that can be accelerated in hardware, the better the overall execution time.

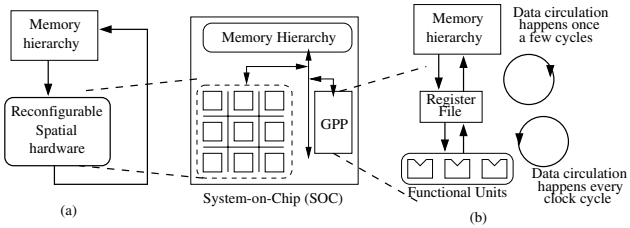

Fig. 1. A system-on-chip (SOC) with spatial and temporal elements.

Further, current day processors are required to deliver higher performance even while keeping the levels of power dissipation at a minimum, due to the necessity of prolonging the battery life in mobile and portable devices. Reduction in total energy consumption in processing an application can be achieved if an optimum combination of computation time and average power consumed is obtained. The *dynamic voltage scaling* technique involves a known fact that reduction in supply voltage to the CMOS circuit will lead to reduced power consumption, as the power is a quadratic function of voltage. But then, the delay in the CMOS circuit scales inversely with voltage thus leading to slower computing frequencies and hence longer execution time. In this paper, we adopt the technique of *dynamic voltage scaling* to study the performance and energy trade-offs in the application-specific architectures that are proposed, and evaluate an optimum supply voltage value for an ideal combination of power and computation time to reduce the overall energy consumed. In the proposed architecture, the voltage supply and the frequency of the RISC based main processor is kept constant, while the supply voltage and hardware resource availability in the coprocessor is scaled to study the variation in total energy consumed and the application execution time.

The contributions of the paper are as follows. The paper proposes and designs a novel *scalable array architecture*, at circuit and microarchitecture level, to map functions of DCT/IDCT and

Proceedings of the 19th International Conference on VLSI Design (VLSID'06)
1063-9667/06 $20.00 © 2006 IEEE

convolution, that are core compute-intensive functions in various multimedia applications. The paper then implements the scalable coprocessor design in the processor microarchitecture simulation tool set and evaluates the variation in total execution time and the energy consumed with the scaling of hardware resources available for the application, and the voltage in the coprocessor. The paper brings forth the trade-offs involved in the execution time and energy based on the characteristics of various multimedia applications, and lays foundation to devise mechanisms for the computing system to dynamically scale system parameters such as hardware composition, application operational granularity, voltage, frequency, and power based on various factors like hardware resource availability, software system support, and effects on performance of concurrent applications.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III, the organization of proposed architecture is discussed, with the microarchitecture design details of the scalable reconfigurable elements. In Section IV, we provide an overview of the architecture simulator used to implement the proposed architecture. Section V presents the study of trade-offs involved in total execution time and energy consumed with scaling of hardware resources and voltage in the proposed architecture. Section VI concludes the discussion with some insights into future work.

II. Related Work

The first known attempts to use the memory elements for computation are that of by Kautz [1] and Stone [2]. Subsequently, the approach of lookup table based computation was accepted, and researchers started to look into building scalable computing systems using such reconfigurable computing elements. Attention has also been drawn towards the significance of a reconfigurable coprocessor coupled with a GPP [3], [4]. In just over a decade, such designs have proceeded from niche to mainstream [5]. In the earliest models of reconfigurable computing architectures, fine grain reconfigurable devices, like FPGAs, were connected as a cluster of computing elements controlled by a GPP. The FPGAs were located off chip, thereby making communication between the microprocessor and the FPGAs a performance bottleneck. Subsequently, instead of using the FPGAs, the reconfigurable computing community preferred to use coarse grain reconfigurable arrays with datapath widths greater than one bit [6]- [21], since the fine-grained architectures are much less efficient due to a huge routing area overhead. Goldstein $et\ al$ [8] proposed Piperench, a pipelined reconfigurable fabric architecture to accelerate streaming multimedia applications. Ye $et\ al$ [9] developed *Chimaera*, a micro-architecture that integrates a reconfigurable functional unit (RFU) into the pipeline of a dynamically scheduled superscalar processor. Razdan $et\ al$ [10] explored ways to incorporate hardware-programmable resources and described a compilation/synthesis system that automatically exploits the resources to improve the performance of general purpose applications. The Garp architecture [11] combines reconfigurable hardware with a standard MIPS processor on the same die to exploit the better features of both. Wittig $et\ al$ [12] developed a processor architecture called *OneChip*, which combines a fixed-logic processor core with reconfigurable logic resources. Using the programmable components, the performance of speed-critical applications was improved by customizing OneChip's execution units, or flexibility was added to the glue logic interfaces of embedded applications. Kim $et\ al$ [22] proposed and developed a reconfigurable functional computing cache that operates as a conventional cache memory or a specialized computing unit depending on the application requirements.

III. Scalable Array Architecture

The proposed *scalable array architecture* is built by incorporating a coarse grain reconfigurable processing element that is tightly coupled with a GPP, with minimal area over head. The reconfigurable coprocessor consists of a two dimensional array of multi-bit output lookup tables (LUTs), where a few rows of LUTs can be formed to process one tap of computation of a specialized function.

The Discrete Cosine Transform (DCT) and convolution algorithms are common compute-intensive functions in signal and image processing applications for pattern recognition, edge detection, etc. Mapping these two algorithms to a hardware fabric can substantially improve the execution time of common multimedia applications like MPEG decode/encode, CJPEG from UCLA Mediabench [23], and FIR/IIR from the TMS320C6000 benchmarks [24]. Image processing applications can be decomposed such that these compute-intensive functions can be mapped to and accelerated in a reconfigurable hardware fabric. Further more, these compute-intensive functions can be decomposed to be implemented as multiple configurations of the multiply-and-accumulate (MAC) and distributed arithmetic (DA) operations. Figure 2 shows an LUT array based design that integrates both the convolution function and the DCT/IDCT function into one circuit. One stage of convolution, consisting of a multiplier and an adder, is implemented using three rows of LUTs. The first row implements an 8-bit constant coefficient multiplier to generate two 4x8 partial products. The second row (excluding 16x16 ROM) implements the addition of the partial products. The third row implements a 24-bit adder to accumulate up to 256 taps of FIR filter. Similarly, implementation of one tap of DCT/IDCT is realized using a 16x16 ROM and one LUT row (the third LUT row in Figure 2). In both the functions, the carry select adder scheme is used to implement a faster addition operation, thus minimizing the propagation delay for the entire operation. Thus, the routing structure for computation consists of only the registers, bus lines and 2-to-1 multiplexers to enable the data flow between rows of LUTs. An application specific integrated circuit (ASIC) design yields a faster processing of a function. However, to process multiple functions we then have to incorporate multiple ASIC cores on the chip leading to inefficient of silicon real estate. It is noted that, the two functions are implemented in a single reconfigurable hardware circuit without additional overhead in the area and the computation delay. Thus, a reconfigurable coprocessor can be designed to process a class of multiple functions that have similar characteristics, and it may also be easy to incorporate the coprocessor tightly coupled with the main processor and the cache hierarchy.

Further, with such a design it is possible for the system to dynamically choose the number of taps a function can be processed in the hardware depending on various parameters like resource availability and power consumption. Besides, in a system with ability to run multiple concurrent applications, it is possible to run multiple applications simultaneously with portions of hardware shared depending on the applications' computational needs.

A. Computation Delay and Power Consumption

Dynamic voltage scaling (DVS) [25] is one of the popular techniques that have been employed in microprocessors for significant reductions in the power consumption. The DVS technique scales the supply voltage, and consequently the frequency of operation, thus controlling the overall throughput. We performed an analysis of variation in the delay and power consumed in one step of DCT and convolution functions, as the supply voltage is scaled from 1.8

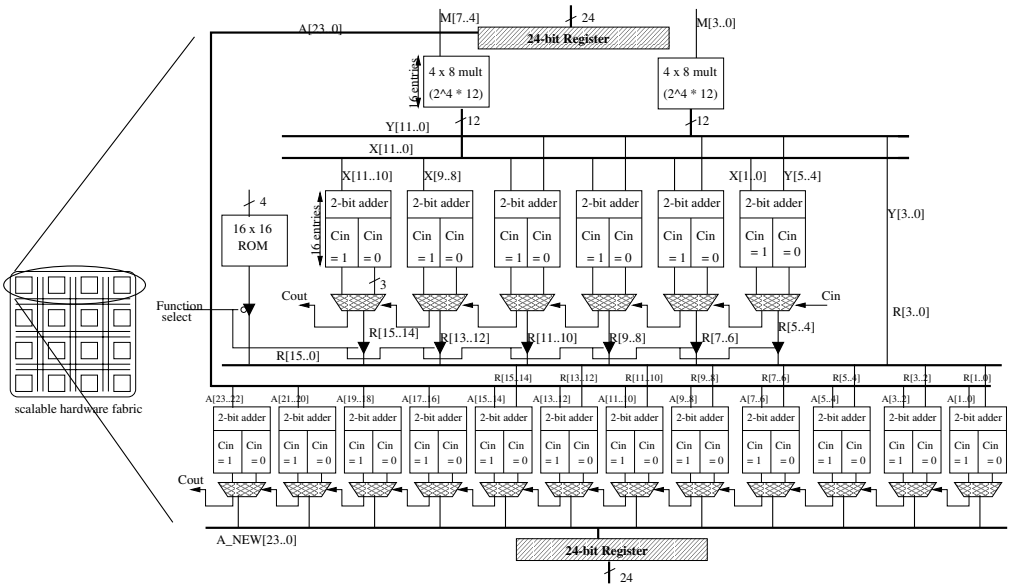

Fig. 2. Scalable hardware circuit with integration of convolution and DCT/IDCT. Convolution uses all the components except 16x16 ROM. DCT/IDCT uses a 16x16 ROM and third LUT row.

to 1.0 volts. Static complementary CMOS design was used for the design using 0.18μ technology, and Cadence tool set was used for laying out the schematic, and to measure the area, delay and power dissipation. The variation in the circuit delay and power consumed in one stage of convolution is shown in Figure 3. It is observed that when supply voltage is scaled from 1.8 to 1.0 volts, power consumed in one tap of the convolution function decreases 8.72 times (from 11.6 mW to 1.33 mW) while the delay increases 2.5 times (from 1.1 ns to 2.8 ns). One tap of the convolution function is measured to occupy a silicon area of $0.000619mm^2$. The figure shows a similar analysis performed to obtain the characteristics for the DCT/IDCT function. With respect to the voltage scaling, it was observed that the power consumed in one tap decreases from 4.95 mW to 0.57 mW, while the delay doubles from 0.39 ns to 0.79 ns. The implementation of one tap takes an area of $0.000308mm^2$.

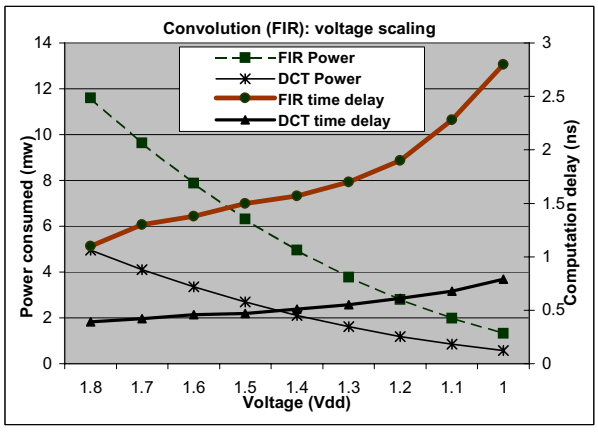

Fig. 3. Effects of voltage scaling in circuit for one tap of convolution and DCT/IDCT functions.

IV. Architecture Simulator for Proposed Design

For simulation purposes, the proposed architecture is implemented by making appropriate modifications in the Simplescalar tool set [26]. The source code of the simplescalar tool set is modified to incorporate the coprocessor design and the microarchitecture is modified accordingly to enable the communication between the main processor and the coprocessor. There are three steps involved in performing a computation in the coprocessor – configuration, loading input, and storing output. All these steps require only one class of instructions, similar to load/store operations in the base processor. To perform these operations, new instructions named as $coproc$ instructions are added. The format of the $coproc$ instructions is same as the conventional load/store instructions except for the target field. The detailed format of the $coproc$ instructions is described in our earlier work [27].

With the current instruction format, it can be assumed that four different functions can be implemented in the coprocessor. However, depending on the reconfigurability of the coprocessor, this can be modified. It has to be noted that, as the reconfigurability of the coprocessor increases to implement many functions, the performance and power consumption degrades due to the increased interconnection and routing structure density. In this paper, we have initially designed the coprocessor in two flavors (DCT/IDCT and convolution) as an LUT based array to process core compute intensive functions. In the latter portion of the paper, the advantage of such design comes to see as we integrate both the functions into one design without much overhead in area and computation delay. The individual instruction operations are explained briefly in the next subsection.

A. Mechanism for the computation in coprocessor

In an out-of-order wide-issue processor, any instruction which does not have a dependency with preceding instructions can be issued and executed at any time if the required resources are available. In addition, in a speculative execution, the next instruction stream in a code sequence can be executed speculatively. The out-of-order issue and execution may also happen among $coproc$ instructions when there is no explicit dependency between $coproc$ instructions. However, the $coproc$ instructions that load input data to the coprocessor for execution, and the $coproc$ instructions that store results from computation in coprocessor, must not be issued and executed until the coprocessor has been configured. A speculative execution mechanism may issue the $coproc$ instructions in any order. To avoid this type of exception, a special two-bit coprocessor state register is included. The two-bit state information is organized as follows.

- 00 : non-configuration/end-computation - coprocessor is in idle mode;

- 01 : CONF - coprocessor is in configuration mode i.e., the configuration data is currently being loaded into the coprocessor;
- 10 : CONF-DONE - end of configuration of coprocessor, ready to execute data anytime;
- 11 : EXE - coprocessor is in execution mode

All the $coproc$ instructions must access the coprocessor state register according to the FID field in the microcode and then check the current state with its CMD field. If it is an allowed state, the $coproc$ instructions can be issued. Otherwise, the $coproc$ instructions are stalled until the corresponding state is resolved.

The configuration of coprocessor from a cache module simply implies loading all the contents of LUTs required to construct a computing unit. A normal cache read with a small modification directs specific data into the designated LUT. The required configuration data for the coprocessor resides in a reserved memory (address) space in main memory. The configuration is loaded into main memory when the system boots up. The corresponding $coproc$ instruction that starts configuration of the coprocessor sets the corresponding state register. The subsequent $coproc$ instructions load the configuration lines to the coprocessor without changing its state.

The $coproc$ instructions that send input data to the coprocessor for execution simply queue the data into an input buffer (IBUF) dedicated to the coprocessor. The IBUF provides data in-order for the coprocessor unit. If no slot in IBUF is available, the following instructions including conventional instructions fetched from memory are stalled until IBUF is again available. By queuing the data from $coproc$ instructions into IBUF in-order, the input data to be processed is provided to the coprocessor in a correct order. This is like a reorder buffer mechanism for input data of the coprocessor unit to remove the impact of out-of-order execution.

A whole function in an application may not be mapped to a coprocessor as a computing unit at one time. For instance, in an FIR filter, if the number of taps for the filtering coefficients is larger than the number of physical taps implemented in the coprocessor, we configure the first set of taps in the coprocessor and then reconfigure it partially for the next set of coefficients at run-time.

After a computation is completed in the coprocessor, the output data is queued into an output buffer (OBUF). The OBUF is a simple FIFO register file since the queued data is already in-order. The output data in OBUF is stored into memory by the $coproc$ store instructions. The $coproc$ terminate execution instruction sets the coprocessor state into the non-configuration/end-execution mode after finishing an entire computation. This setting is done in commit stage to avoid mis-execution of pending $coproc$ instructions. If the same computation with the current configuration is performed in the near future, the coprocessor state is not changed, and the state register is set to "10" to be ready for next execution.

V. Performance of Scalable Array architecture

To evaluate variations in overall execution time and energy consumption, we implemented a coprocessor with DCT and convolution functions, in a wide-issue processor pipeline using the Simplescalar tool set [26]. For the main processor, we simulated a 4-wide out-of-order issue processor with 32 KB L1 I-cache (2-way 1 cycle), 32 KB L1 D-cache (4-way 1 cycle), 256 KB L2 cache (4-way 6 cycle), 100-cycle memory latency, and with other default parameters in the tool. The source code of the Simplescalar tool set is modified to incorporate the coprocessor design, and the microarchitecture is modified accordingly to enable the communication between main processor and coprocessor. The performance of the proposed architecture is assessed based on the number of cycles it takes to execute each application as compared to a conventional GPP without a coprocessor. This also includes the number of cycles required to configure the coprocessor. For example, to configure a coprocessor with 8 taps of DCT/IDCT function, it takes a total of 256 cycles, since each line in an LUT row can be configured in one cycle. We ran the simulations for MPEG decode/encode, CJPEG, FIR, and IIR applications, for more than one billion instructions each, in the base architecture. Then the same portion for each application is run in the proposed architecture. It is to be noted that, the core function is mapped and computed in the coprocessor in the proposed architecture, while it is computed in the main processor itself in the base architecture. The coprocessor design for mapping DCT/IDCT is implemented with 8 taps and requires an area of $0.0024mm^2$, 16 taps of convolution function takes an area of $0.0099mm^2$, and 256 taps of convolution takes an area of $0.1585mm^2$ in 0.18μ technology. To estimate the power dissipated in the proposed architecture, we used architectural power simulator of Wattch [28]. Wattch is a power estimation model built over the Simplescalar tool. It measures the utilization of various processor components, and during the simulation feeds these utilization numbers into a high-level power model to estimate the energy behavior of the processor. The power measurements made for the coprocessor designs, as discussed in earlier sections, are incorporated with the necessary modifications in the tool. It is to be noted that, for estimating power consumption in the coprocessor the number of taps implemented is taken into account.

It is observed that, when the core function is executed in the reconfigurable coprocessor, the total number of instructions executed, the number of load and store instructions and the number of accesses to L1 data and instruction caches have reduced. From the estimation of energy consumed for the application execution, in the following sections of the paper we show that the effect of power dissipation overhead due to additional coprocessor on the chip is offset by the reduced power consumption in the main processor due to its reduced utilization. In other words, due to a significant reduction in the overall execution time of the application, it will result in savings in overall energy consumption.

A. Results and Analysis

The proposed architecture is evaluated for total execution time and overall energy consumption for an application with the scaling of hardware resources and supply voltage in the coprocessor. Figure 4 shows the relative total execution time and the relative overall energy consumed for the execution of MPEG decode, MPEG encode, CJPEG, abd IIR applications. The chart shows the values relative to those computed in the base processor architecture without a coprocessor. It can be seen that, when the MPEG decode application is executed in the proposed architecture with the DCT/IDCT function computed in the coprocessor, the total execution time is around 24% of the time taken to execute the application in the base processor. Also, the energy consumed is around 27.5% of that consumed in the base architecture. This large speed-up is due to the significant acceleration of the DCT/IDCT function in the coprocessor, and the significant fraction of the core function in the application. Hence, it is observed that the overall speed-up is largely proportional to the fraction of the core function, in the entire application, that is mapped to the coprocessor. Further, it can be observed that as the voltage is scaled down from 1.8 to 1.0 volts at 0.18μ technology, the relative total execution time increases slightly (from 23.65% to 24.78%), while resu[illegible]

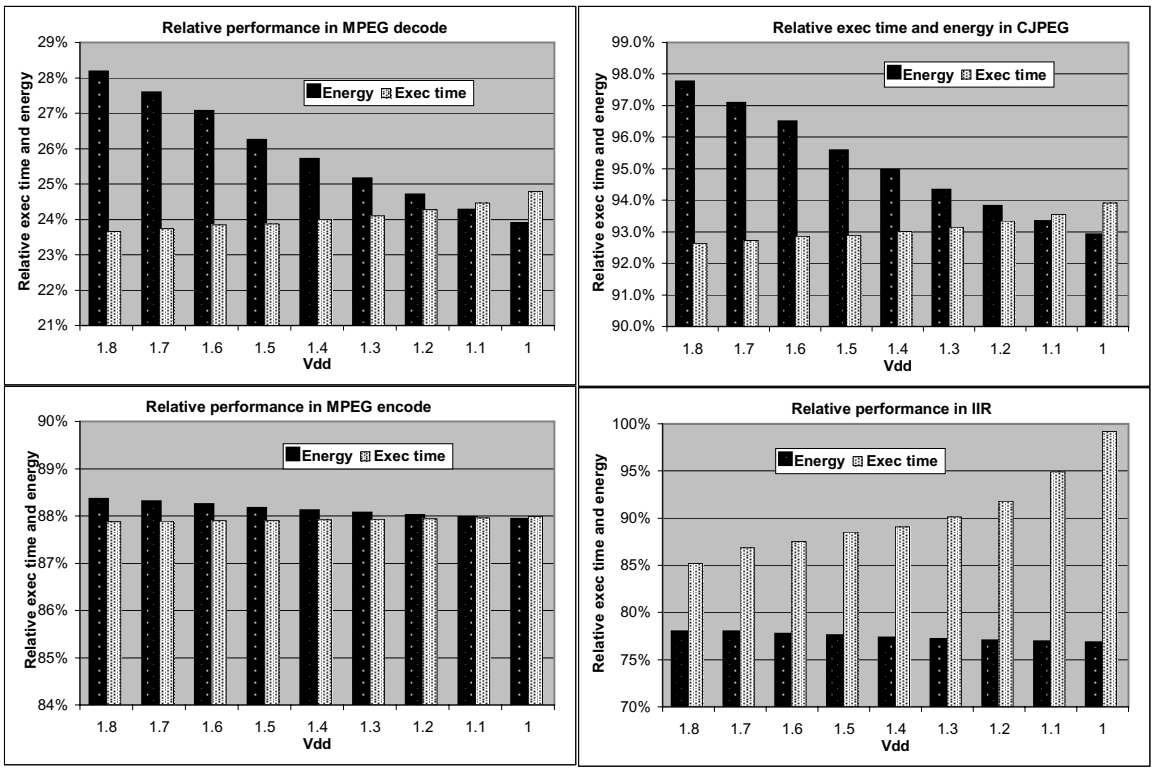

Fig. 4. Relative to the base processor, variation of total application execution time and energy consumption for MPEG decode, MPEG encode, CJPEG, and IIR applications w.r.t supply voltage scaling.

in considerable amount of reduction in the energy consumed (from 28.12% to 23.8%).

When the MPEG encode application is executed in the proposed architecture with the DCT/IDCT function computed in the coprocessor, the total execution time is around 88% of the time taken to execute the application in the base processor. Also, the energy consumed is around 88.2% of that consumed in the base architecture. This smaller reduction in execution time is due to the fact that the MPEG encode application has a small fraction of the core function to be mapped to the coprocessor, and hence the code that is computed in the main processor is predominant. For the same reason, it can be observed from the Figure 4, that as the voltage is scaled down from 1.8 to 1.0 volts, the relative total execution time and the overall energy consumed do not change much. For the CJPEG application, the voltage scaling leads to considerable gain in overall energy consumed, even while keeping the performance gain almost the same. The figure also shows the relative execution time and the energy consumed when the IIR application is executed in the proposed architecture, with the 16 taps of the convolution function designed in the coprocessor. It can be inferred that, with this application, the voltage scaling does not lead to any gain in overall energy consumed, though it considerably reduces the performance gain.

For FIR application with 16 taps of convolution, the voltage scaling does not lead to any gains in overall energy consumed, though it reduces the performance gain, as shown in Figure 5. For FIR application with 256 taps of convolution function, it is observed that as a large number of taps are implemented in the coprocessor (with corresponding increase in area and power), a large amount of speedup is achieved, with the total application execution time reduced to around 25%. Also, as the voltage is scaled, there is a noticeable gain in the energy consumed and a reduction in the performance gain.

Overall, it can be observed that total energy consumption in executing an application in the proposed architecture is smaller as compared to that in the base processor. This is mainly due to the reduced energy consumption in the main processor due to the significantly reduced activity in all the on-chip components of the processor. And, the most important fact is that the reduction in overall energy consumption is achieved along with a higher performance in executing the multimedia applications.

VI. Conclusions and Future Work

The amount of energy consumed to deliver the required level of performance is an important consideration, to prolong the battery life in portable and mobile devices. In this paper, we proposed a novel *scalable array architecture* and explored the execution time and energy trade-offs for various applications by scaling hardware resources, operational granularity, and voltage supply. Keeping in pace with the current requirement of power-aware architectures, we showed that the coprocessor based system architecture delivers higher performance while providing with significant savings in energy dissipation in computing various multimedia applications. When the MPEG decode application was executed in the proposed architecture, the total execution time was reduced to around 24%, and the energy consumed was reduced to around 27.5% of that consumed in the base architecture. Further, the scaling of voltage resulted in considerable reduction in the energy consumed (from 28.12% to 23.8%), while total execution time increased slightly (from 23.65% to 24.78%). A similar study of the performance and energy characteristic behavior of other multimedia programs like MPEG encode, CJPEG, FIR, and IIR was also conducted. Overall, it was observed that the gain in performance and energy consumed are dependent on the application characteristics and the amount of hardware resources utilized in the coprocessor.

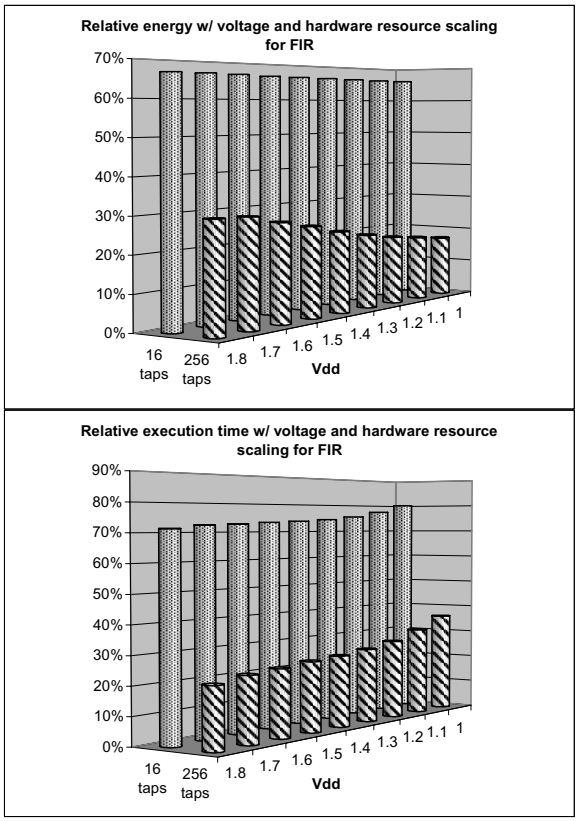

Fig. 5. Relative to base processor, variation of total application execution time and energy consumption for FIR application, with scaling of hardware resources and supply voltage.

Based on the characteristics of various applications, the future work will design and develop scalable hardware systems with hardware/software codesign, that can simultaneously run multiple applications, and optimize multiple critical performance parameters for each application. Appropriate techniques/algorithms will be developed to dynamically scale the hardware resource usage, supply voltage, operational granularity, etc., and will be implemented at the hardware level (with appropriate control logic in the main processor), or at the software level (with assistance from compiler or/and the operating system), depending on the characteristics of the application and the composition of computing system.

REFERENCES

[1] W. Kautz, "Cellular Logic-in-Memory Arrays", *IEEE Transactions on Computers*, Volume: C-18, Issue: 8 , August 1969, pp. 719-727.

[2] H. S. Stone, "A Logic-in-Memory Computer", *IEEE Transactions on Computers*, January 1970, pp. 73-78.

[3] A. DeHon, "DPGA-coupled microprocessors: commodity ICs for the early 21st Century", *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 31-39.

[4] A. DeHon, "The Density Advantage of Configurable Computing", *IEEE Computer*, Volume: 33, Issue: 4, April 2000, pp. 41-49

[5] R. Hartenstein, "The digital divide of computing", *Proc. The first conference on computing frontiers*, Italy, April 14-16, 2004.

[6] E. Mirsky et al., "MATRIX: a Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources" *Proc. the IEEE Symposium on FPGAs for Custom Computing Machines*, April 1996, pp. 157-166.

[7] D. C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," *Proc. 20th Anniversary Conference on Advanced Research in VLSI*, 1999, pp. 23-40.

[8] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer, "PipeRench: a coprocessor for streaming multimedia acceleration," *Proc. 26th International Symposium on Computer Architecture*, 1999, pp. 28-39.

[9] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," *Proc. 27th International Symposium on Computer Architecture*, 2000, pp. 225-235.

[10] R. Razdan and M. D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," *Proc. 27th Annual International Symposium on Microarchitecture*, 1994, pp. 172-180.

[11] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler," *IEEE Computer*, Vol. 33, Issue: 4, Apr. 2000, pp. 62-69.

[12] R. D. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, pp. 126-135.

[13] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger, "KressArray Xplorer: a new CAD environment to optimize reconfigurable datapath array", *Proc. The 2000 conference on Asia South Pacific design automation*, Japan, January 2000, pp.163-168.

[14] Takashi Miyamori and Kunle Olukotun, "REMARC (abstract): reconfigurable multimedia array coprocessor", *Proc. The 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, February 22-25, 1998, pp. 261.

[15] H. Singh, et al., "MorphoSys: An Integrated Re-configurable Architecture", *Proc. The NATO RTO Symposium on System Concepts and Integration*, April 20-22, 1998.

[16] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications", *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, February 21-23, 1999, pp.135-143.

[17] J. Becker et al., "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems", *Proc. FCCM 2000*, April 17-19, 2000.

[18] Xinan Tang, Manning Aalsma, and Raymond Jou, "A Compiler Directed Approach to Hiding Configuration Latency in Chameleon Processors", *Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, August 2000, pp.29-38.

[19] http://www.sidsa.com

[20] J. Rabaey, "Reconfigurable Processing: The Solution to Low-Power Programmable DSP", *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP '97, April 1997.

[21] M. Herz, "High Performance Memory Communication Architectures for Coarse-Grained Reconfigurable Computing Architectures", Ph. D. Dissertation, Universitaet Kaiserslautern, January 2001.

[22] H. Kim, A. K. Somani, and A. Tyagi, "A Reconfigurable Multifunction Computing Cache Architecture", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9, Issue: 4 , August 2001, pp. 509-523.

[23] Chunho Lee, M. Potkonjak and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", *Proc. Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 330-335.

[24] Texas Instruments, "TMS320C6000 benchmarks", 2000, http://www.ti.com/sc/docs/products/dsp/c6000/62bench.htm

[25] T. Burd and R. Brodersen, "Design issues for dynamic voltage scaling," *Proc. IEEE International Symposium on Low Power Electronics and Design*, 2000, pp. 9-14.

[26] Doug Burger and Todd M. Austin, "The SimpleScalar Tool Set, Version 3.0", Computer Sciences Department Technical report # 1342, University of Wisconsin-Madison, June 1997.

[27] R. Sangireddy, H. Kim, and A. K. Somani, "Low-power High-performance Reconfigurable Computing Cache Architectures," *IEEE Transactions on Computers*, Vol. 53, Issue: 10, October 2004, pp. 1274-1290.

[28] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", *Proc. 27th International Symposium on Computer Architecture*, 2000, pp. 83 -94.