Handling Architecture-Application Dynamic Behavior in Set-top Box Applications*

Naeem Zafar Azeemi
Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms,
Institute of Communications and Radio Frequency Engineering,
University of Technology Vienna, Gusshausstrasse 25/389, A-1040 Vienna Austria
Email: nzafar@nt.tuwien.ac.at

***Abstract*— Future ubiquitous computing will be consisted of set-top boxes and handheld devices based on enormous computation power and capability to handle multimedia applications workload. Efficient architecture utilization and optimal application binary will be two important performance metrics for these embedded systems. In real-time system, the conventional method of system design cannot be used. In these methods, the cycle and code size are primarily considered, while power dissipation issue is completely ignored, that inevitably lead to expensive cooling mechanism and eventually increase the system overall cost while reducing reliability. An integrated approach that considers energy-cycle performance at architecture as well as application level is essential for energy efficient application development. This paper focuses on distributed optimization paradigm, based on vertical profiling. The proposal is based on enhance local optimization and peer-to-peer (P2P) source code compilation across application build flow. The earlier procedures incorporate additional steps for pre/post profiling during compilation, scheduling and linking phases. This iterative activity is carried out by methods based on wait-for-graphs or profile monitors. These methods introduce a centralized evaluation at source code (for each transformation and monitored parameters during successive approximation). The proposal introduces the asynchronous evaluation in source to source (sts) process. As a result the code transformation do not trapped in local optimization, rather look for global optimization (for both energy and cycle). Methodology is readily adaptable in an iterative compilation environment where application source code is optimized to satisfy user constraints on code size, energy, execution time and optimal target architecture usage. Experimental results show that our approach enhances parallelism upto 38% in G-721 speech codec, increases architecture correlation upto 17% in video transcodec H-264L, improves power efficiency as much as 43% for resVQ DSP algorithms. The technique incurs low overheads and enhances the application architecture correlation.**

***Keywords*—set-top boxes, embedded systems, low energy, multimedia algorithm**

I. INTRODUCTION

Since the first inception, set-top boxes are the most lucrative choice for the implementation of multimedia function in household appliances [1,2]. Because off-the-shelf hardware prices constantly decrease, embedded systems are more affordable to be purchased than before. Embedded systems are often mass-produced, so the cost savings may be multiplied by millions of items. For high volume systems such as portable music players or mobile phones, minimizing cost is usually the primary design consideration.

Unlike a general-purpose computer, such as a personal computer, an embedded system performs one or a few pre-defined tasks, usually with very specific requirements, such as:

- Is the architecture suitable?
- Are the processing components powerful enough?
- Is the operating system suitable?
- Is the fault tolerance is adequate?

Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product. Engineers typically select hardware that is just "good enough" to implement the necessary functions [2,3,5,7]. For example, a digital set-top box for satellite television has to process large amounts of data every second, but most of the processing is done by custom integrated circuits. The embedded CPU "sets up" this process, and displays menu graphics, etc. for the set-top's look and feel. The software written for embedded systems is often called firmware, and is stored in ROM or Flash memory chips rather than a disk drive [4,8,9]. It often runs with limited hardware resources: small or no keyboard, screen, and little RAM memory.

Embedded systems reside in machines that are expected to run continuously for years without errors, and in some cases recover by themselves if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided. Recovery from errors may be achieved with techniques such as a watchdog timer that resets the computer unless the software periodically notifies the watchdog.

In contrast to the desktop computer, there are many different CPU architectures [1,8], used in embedded designs such as ARM, MIPS, x86, PIC, 8051, Atmel AVR, Motorola 68k, PowerPC etc. A common configuration for very-high-volume embedded systems is the system on a chip (SoC), an

**This work has been funded by Chritian Doppler Laboratory for Design Methodology of Signal Processing*

application-specific integrated circuit (ASIC), for which the CPU was purchased as intellectual property to add to the IC's design.

The remainder of this paper is organized as follows: Motivation is explained with background of embedded software in Section II. Iterative compilation and optimization strategy for our approach is presented in Section III. The experimental setup and benchmark applications are explained in Section IV. Case studies showing the significance of our methodology are given in Section V. Finally, conclusions are given in Section VI.

II. EMBEDDED SOFTWARE ARCHITECTURES

As mentioned above the embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reason such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

In the same vein the resident applications on set-top box are also an important contributing factor, they are middleware program that serves to "glue together" or mediate between two separate and usually already existing programs [10]. These are updated, often automatically, by the network operator via the data streams that the set-top receives from the network operator. It allows set top boxes and network operators to talk to each other. PC/104 is a typical base for small, low-volume embedded and ruggedized system design. These often use DOS, Linux, NetBSD, or an embedded real-time operating system such as QNX or Inferno. It often involves coordinating DRM, electronic program guide data and the software interface on a DVR, and/or with the cable backend. It includes an application manager, the virtual machine (such as Java Virtual Machine™), the interactive engine, the libraries and databases. Middleware becomes particularly handy if there are a number of different programs, platforms and software.

A common model for this kind of design is a state machine [1,14,18], which identifies a set of states that the system can be in and how it changes between them, with the goal of providing tightly defined system behaviour. The task functionality of middleware can be implemented in single state machines, multiple state machines, flat state machines, and hierarchical state machines. Such task would create and delete state machine objects as and when required. Following embedded software architectures are commonly used:

1. Simple control loop architectures (SCPA)

Simplicity is the embedded system's strength and on small pieces of software the loop is usually so fast that nobody cares that its timing is not predictable. It is common on small devices with a stand-alone microcontroller dedicated to a simple task. weaknesses of a simple control loop are that it does not guarantee a time to respond to any particular hardware event (although careful design may work around this), and that it can become difficult to maintain or add new features.

2. Interrupt controlled system architectures (ICSA)

Whether a uniprocessor or multiprocessor, generally embedded systems are predominantly interrupt controlled. This means that tasks performed by the system are triggered by different kinds of events. These kinds of systems are used if event handlers need low latency and the event handlers are short and simple. Usually these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays. The tasks performed in the interrupt handlers should be kept short to keep the interrupt latency to a minimum. Some times longer tasks are added to a queue structure in the interrupt handler to be processed in the main loop later. This method brings the system close to a multitasking kernel with discrete processes.

3. Nonpreemptive multitasking architectures (nonPMA)

The programmer defines a series of nonpreemptive tasks, and each task gets its own environment to "run" in. A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. An architecture with similar properties is to have an event queue, and have a loop that processes the events one at a time. In such architecture, adding new software is easier, by simply writing a new task, or adding to the queue-interpreter.

4. Preemptive multitasking (PMA)

This is the level at which the system is generally considered to have an "operating system", and introduces all the complexities of managing multiple tasks running seemingly at the same time. In this type of system, a low-level piece of code switches between tasks based on a timer. Any piece of task code can damage the data of another task; they must be precisely separated. Access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.

5. Microkernels

The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. A microkernel is a logical step up from a real-time OS, where user mode processes implement major functions such as file systems, network interfaces, etc. In general, microkernels succeed when the task switching and intertask communication is fast, and fail when they are slow.

6. Monolithic Kernels

These architechture environemt adopt a full kernel with sophisticated capabilities that gives the programmers a full environment similar to a desktop operating system like Linux or Microsoft Windows, and is therefore very productive for development; on the downside, it requires considerably more hardware resources, is often more expensive, and because of the complexity of these kernels can be less predictable and reliable. Embedded Linux and Windows CE are the common examples of such kernela. These kernels are popular in powerful embedded devices such as wireless routers and navigation systems.

The embedded application development life cycle starts with high level implementation of algorithm implementation as

shown in Figure 1.1. However, the manner and extent to which we must adapt those techniques to deal with highly distributed, heterogeneous, mobile, embedded environments remain largely unexplored [13,15,17]. One area from which we might gain leverage in the ubiquitous and embedded systems domain is software architecture. Here the author draws general distinctions between "traditional" software architectures and those targeted at embedded systems. Followed by is the DSP compiler for target platform. Though mostly optimization is done at the compilation level, but traditional compiler generates poor binary code, both in term of energy-cycle performance and good architectural usage. The idea in iterative compilation is to compile an application with different optimization strategies and then select the best result among these. We demonstrate that it is necessary to use an application dynamic profile at all layers to understand existing performance problems such as poor architecture usage, increased execution time, and high energy consumption.

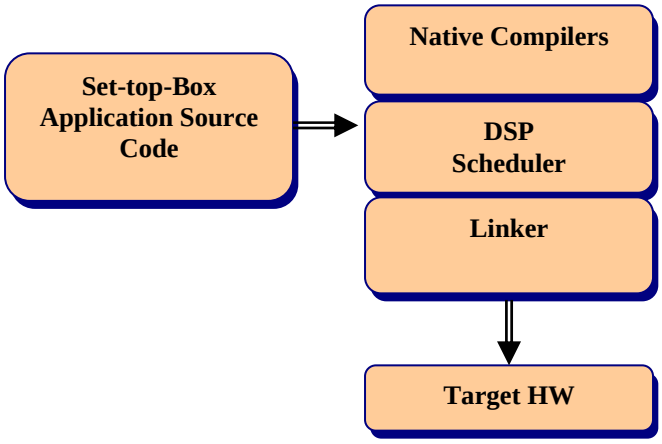

Figure 1.1. A Typical Set-top Box Application Build Flow

We add several implementations of energy efficient techniques for embedded DSP compiler [3, 16, 17] to generate high performance computing applications, they are mentioned below:

- Application static and run time profile are captured during the application build flow that is further used to build an optimization space and finding the good one eventually.
- System behavior is improved through correlation of profile information at different levels.
- This iterative activity is carried out by methods based on wait-for-graphs or profile monitors. These methods introduce a centralized evaluation at source code (for each transformation and monitored parameters during successive approximation).
- Results are analyzed for different architecture performance metrics such as parallelism, cache utilization, energy saving, operations per cycle. Further applications performance is evaluated for efficient architecture utilization.

III. Energy Aware Profiling Technique

In this section, we will discuss how to capture the profile monitors at different layers of application build flow as shown in Figure 1. This scheme helps to obtain best performance speedup for set-top box applications, but increases application size and not an appropriate choice where memory footprint is an important constraint.

A. Background

Energy reduction can be achieved by exploiting the idleness of system components. Different techniques can be used to exploit the system component behavior, which is an inevitable outcome of software execution. Software controls indirectly the dynamic part of total energy consumption, which is CV^2, where V is the operating voltage of CPU and C is the switching capacitance [9, 10]. At single core voltage CPU, the compute data activity gives rise to switching capacitance; reducing this component down would decrease the energy consumption of application at its life time.

In VLIW processors, many of the components in the CPU are not completely utilized during the program execution. Primary reason for such slack is the poor architecture-application correlation. This indicate that for an energy efficient application binary there is a need to gather more detailed profiles, containing information about system behavior on various levels (Figure 1.1). The goal of profiling is to find cause-effect relations between performance phenomena and finally generating an architecture efficient code. Our energy-aware framework [3] embodies a series of profiling stages that enable the optimization process.

B. Set-top-Box Application Performance Monitors(stbAPM)

In proposed framework, the impact of code transformations is fed back to transformation engine to identify performance critical bottlenecks. This mechanism requires extensive program execution analysis to get stbAPM and eventually a good code. The accuracy of transformed code is checked against the performance of original code at target platform. The profiling stages described in Figure 1.1, detects if binary is an efficient energy application, then if needed code blocks can be restructured or transformed with transformation engine using optimal transformation scheme suggested by genetic algorithm. Followed by, basic blocks of energy-cycle critical code are located, and when necessary, converted using conventional loop optimization schemes, such as loop unrolling, loop fusion, decision tree grafting. Detail of this scheme is mentioned in [15]. stbAPM are typically used to converge to optimal CPU and cache usage. In [17], intermediate trace files are generated during the code processing flow to produce performance monitors, E.g., code size, execution time, number of cache miss, scheduling factor, and slot utilization et.. After

simulation, these parameters are used to compute transformation control factors such as unrolling factor, grafting depth and blocking metrics (explained in [17]). Successively, after each cycle, each of these parameters is computed again and is compared to constraints mentioned in the user constraint file. This file contains user constraints, to be used in maximizing objective function.

IV. TRACE-BASED RUNTIME APPLICATION OPTIMIZATION (TBRAO)

The overhead of finding optimal transformation space for six objectives mentioned above inevitably leads to NP hard solution. In tbRAO, we approximate the optimization problem as the multiple objective optimization of energy saving and execution cycles. The individual candidate points in transformation space are chosen with a uniform probability distribution. They are profiled later by evaluating the application profile at the target architecture. The selected individual transformations are updated based on their success, execution cycles and energy saving factor of the sequence as a whole. Transformations contributing to better performance are rewarded while those resulting in performance losses are penalized. Thus, future sample points are more likely to include previously successful transformations more frequently and search their neighborhood more intensively.

Note that we have been using the on chip as well as off chip access metrics in our tbRAO framework as the indication to the performance. We do not account for the overlapping part for the inter-process communication and this is addressed in our next part of work, not discussed here. Our algorithm currently considers of restructuring legacy code only one source to another source at a time.

A. Multimedia Workloads

We obtain the stbAPM by performing a series of experiments on VLIW processors. The benchmarks were compiled at evaluation board operating at 266 MHz. The benchmark profiles were obtained using the profiling simulator, and the performance data was collected using our framework [14]. We use 6 programs from diversified domains as benchmark. The important characteristics of these codes are given in [15,17]. Multimedia applications use DSP algorithms and streaming data schemes to compute and later to produce high throughput for real time video or audio applications. The quality of throughput depends on the application domain, e.g., bandwidth and frame rate for a typical MPEG-1 application is different at mobile device and set-top box. We chose the applications for their importance in real systems and to be representative enough to make the inferences in this study. This application set contains speech codecs (G-721), video codecs (MPEG-1), and generic DSP algorithms (m64, resVQ etc.).

V. RESULTS AND DISCUSSION

Applications are the driving agent in embedded systems, and an important energy contributing factor to optimal architecture utilization. The benchmarks were validated against precompiled binaries provided in the original distributions of the benchmark suites. Application-Architecture Correlation (AAC) in our result is obtained by the ratio of execution of code at an infinite resource machine to the finite resource machine. For our target platform we compared it with 9- issue slot machine with instruction level parallel operation constraints as mentioned in [15]. Figure 5.1- Figure 5.7 shows the percentage improvement in each benchmark application to the base line code. Inherently due to highly branch oriented coding dVQ (differential Vector Quantization) and H-264L has higher AAC i.e. 16.5% and 16% respectively. Audio compression codec G-721 is dominated with deep nesting, an implicit feature of its wavelet algorithms, those results into average AAC (13%). Whereas other applications and transcodecs reflects moderately for the AAC, e.g., MPEG1 (2%), resVQ (15.5%) in Figure 5.1.

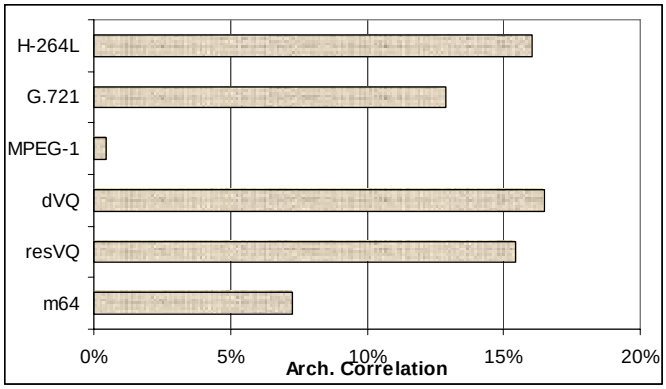

Figure 5.1. Proportional Architectural Correlation in Multimedia Benchmark

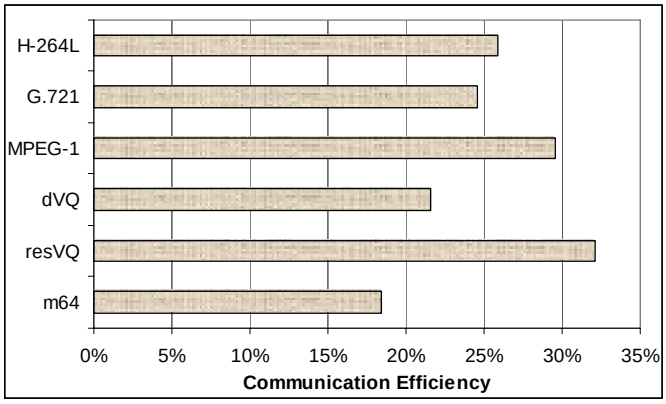

Figure 5.2. Proportion of Communication Bus Efficiency over the Baseline Version in Multimedia Benchmark

Despite being highly localized, native compiler is inefficient to utilize on-chip register to reduce down the off-chip traffic and give rise to energy consumption as well as cycle count. E.g., for generic DSP algorithm dVQ, the low spatial locality (21%), causes higher communication (22%), see Figure 5.2, entailed by low inherent parallelism (25%), eventually leads to

small improvement in power saving (23%) and execution cycles (43%).

Our hardware architecture offers high degree of parallelism, a favorite choice for applications pertaining higher temporal and spatial data independence. MPEG1, H-264L and resVQ reflect such behavior in Figure 5.2, Figure 5.3, and Figure 5.5.

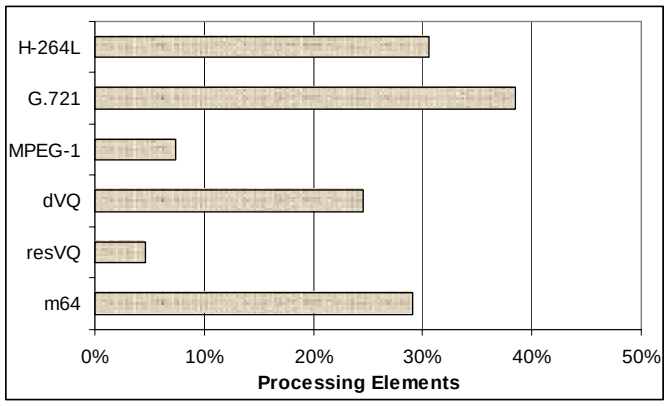

Figure 5.3. Comparative Parallelism Pertain by Multimedia Benchmark Applications.

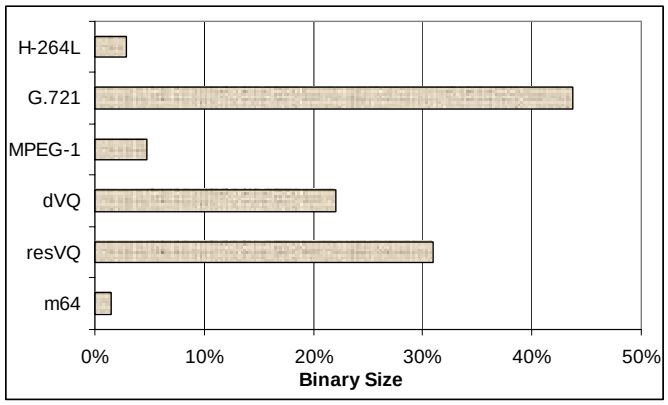

Figure 5.4. Percentage Increase in Application Size after the Proposed Transformation.

A careful consideration to Figure 5.7 reveals the fact that VLIW architecture is well suited for MPEG-1 transcodecs, the implicit spatial and temporal parallelism in MPEG1 algorithm let it to exploit CPU 9-issue slots up to 19%, raising effective internal communication to 29%, leading to an energy as well as speed efficient application.

A. Set-Top Box Performance Issues Analyses

Primary objective of this work was two fold:

First to find architecture pro applications both in term of optimization and algorithmic implementation.

Second the degree of optimality that architecture provides to a candidate application for energy and cycle efficiency.

Applications runtime profile in Figure 5.5, Figure 5.6 and Figure 5.7, clearly conclude the fact that there is a strong correlation between the AAC, code spatial locality, processing parallelism and power saving factor.

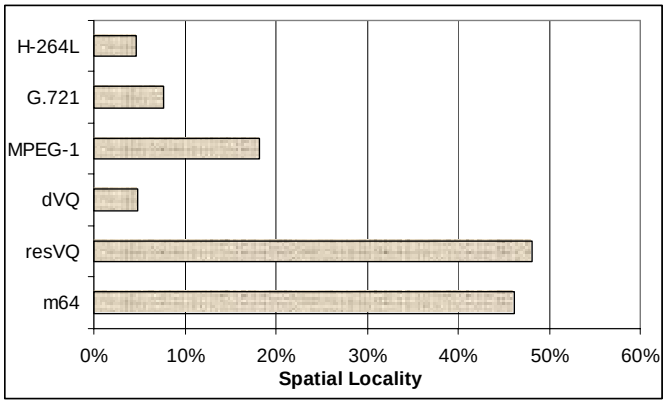

Figure 5.5. Spatial Locality Performance Sensitivity in Multimedia Benchmark.

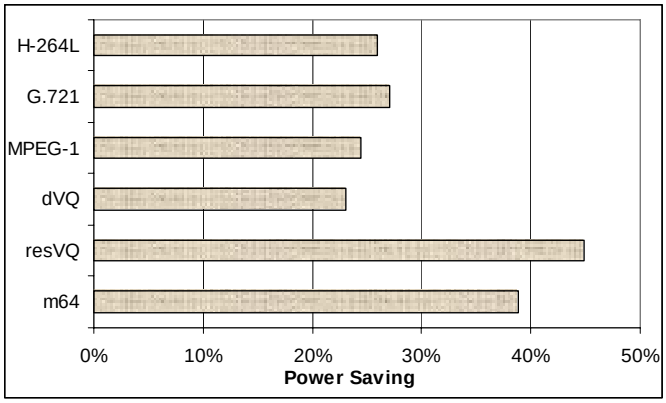

Figure 5.6. Proportional Power Saving Performance over the Baseline Version in Multimedia Benchmark

The average cycle efficiency of video transcodecs MPEG1, H264L and image compression algorithm resVQ is higher than the other algorithms due to higher spatial and temporal code execution locality. While dVQ and speech codec e.g., G-721 are not suitable applications for our hardware. Figure 5.6 depicts this fact. On the other hand power saving is more prominent in MPEG1 (23%), resVQ (43%), and m64(38%)as compared to original implementation of code.

When the data dependency and hence parallelism (α) in application is not higher than 11, the power saving get notably improvement due to potential opportunities for loop unrolling and maximum on-chip register usage. The results in Figure 5.7, show one order of magnitude enhancement in the application performance, 44% enhancement in the average cycle efficiency per baseline code for differential vector quantization algorithm (dVQ) and 41% enhancement in cycle efficiency per baseline code for MPEG1 transcodec. Note that for video codec, the input stream 'flowrgarden' was obtain from public domain website.

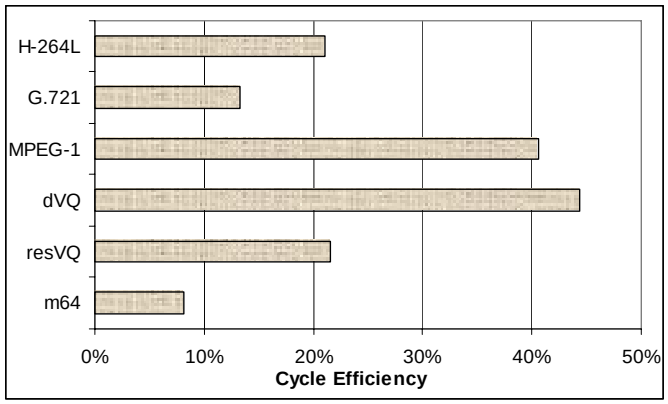

Figure 5.7. Proportional Cycle Efficiency over the Baseline Version in Multimedia Benchmark

Multiple exiting edge causes repeated decision tree grafting and, result in too much power consumption and also increase in code size. tbRAO help us replace the 'weak transformations' by 'good transformations', in which architecture utilization is efficient. Moreover tbRAO also balances the effective architecture utilization. So, in all benchmarks we tested, the power variance improved smoothly. It is a very good property of our tbRAO framework.

VI. CONCLUSIONS

We have presented a framework tbRAO embedded with sbAPM for set-top box applications. We have studied the relationship of the power efficiency, spatial and temporal locality with the impact on code size.

Our method achieves considerably better architecture utilization and power efficiency over the current embedded middleware compilation environment, which does not consider the power optimization as a forefront objective. In order to get better performance, our methods introduce a centralized evaluation at source code (for each transformation and monitored parameters during successive approximation). The proposal introduces the asynchronous evaluation in source to source (sts) process. As a result the code transformation do not trapped in local optimization, rather look for global optimization (for both power and cycle).

Methodology is readily adaptable in an iterative compilation environment where application source code is optimized to satisfy user constraints on code size, energy, execution time and optimal target architecture usage.

ACKNOWLEDGMENT

This work is supported by ÖAD-Pakistan scholarship program initiated by Prof. Dr. Atta-ur-Rahman chairman HEC and Federal Minister Pakistan.

The author would like to thank Prof. Dr. Markus Rupp, Prof. Dr. Arpad Scholtz and Christian Doppler Laboratory at Institute of Communication and Radio-Frequency Engineering, Vienna University of Technology for their support and kind input during this work.

REFERENCES

[1] Nenad Medvidovic, "Software Architectures and Embedded Systems: A Match Made in Heaven?," *IEEE Software*, vol. 22, no. 5, pp. 83-86, Sept/Oct, 2005.

[2] E. Di Nitto and D.S. Rosenblum, "Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures," *Proc. 21st Int'l Conf. Software Eng.* (ICSE 99), IEEE CS Press, 1999, pp. 13–22.

[3] N. Zafar Azeemi, "Power Aware Framework for Dense Matrix Operations in Multimedia Processors," Proc. of the IEEE 9th International Multi-topic Conference, Dec. 2005.

[4] Parameswaran, S. "Code placement in hardware/software co-synthesis to improve performance and reduce cost," Proc. of the Conference on Design, Automation and Test., pp 626-632, 2001.

[5] S. Bashford and R. Leupers, "Constraint driven Code Selection for Fixed-Point DSPs," Proc. of the 36th Design Automation Conference (DAC), Nov. 1999.

[6] E.A. Lee, "Embedded Software," *Advances in Computers,* E. Zelkowitz, ed., Academic Press, 2002.

[7] M. Mikic-Rakic and N. Medvidovic, "Adaptable Architectural Middleware for Programming-in-the-Small-and-Many," *Proc. ACM/IFIP/USENIX Int'l Middleware Conf.* (Middleware 03), LNCS 2672, Springer-Verlag, 2003, pp. 162–181.

[8] Chandrakasan A. And Brodersen R. " Low Power Digital CMOS Design," Kluwer, 1995.

[9] C. Gebotys, R. Gebotys, S. Wiratunga, "Power minimization derived from architectural-usage of VLIW processors," Proc. of the Annual ACM IEEE Design Automation Conference, pp. 308-311, June 2000.

[10] C. Gebotys, R. Gebotys, "Statistically based prediction of power dissipation for complex embedded DSP processors," Micro-processors and Microsystems Journal, vol. 23, pp. 135-144, 1999.

[11] G. Fursin, M. O'Boyle, P. Knijnenburg, "Evaluating iterative compilation," Proc. of Languages and Compilers for Parallel Computers (LCPC'02), College Park, MD, USA, 2002.

[12] V. Tiwari, S. Malik, A. Wolfe, "Compilation techniques for low energy," Proc. of the ISLPED, Oct 1994.

[13] N. Z. Azeemi, M. Rupp, "Muticriteria Low Energy Source Level Optimization of Embedded Programs," Proc. of the IEEE Informationstagung Mikroelektronik 2006, pp. 150-158, Oct. 2006.

[14] TM1300 Data Book, Philips Electronic, North America Corporation, pp. 3.1-3.16, Oct 1999.

[15] N. Zafar, M. Rupp, "Energy-aware source-to-source transformations for a VLIW DSP processor," Proc. of the IEEE 17th ICM 2005, pp. 133-138, Dec. 2005.

[16] T. Baeck. Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996

[17] N. Zafar Azeemi, "A Framework for Architecture Based Energy-Aware Code Transformations in VLIW Processors," Proc. of the IEEE International Symposium on Telecommunications (IST 2005) pp.393-398. Sep. 2005.

[18] M. Lee, V. Tiwari, S. Malik, M. Fujita, ''Power Analysis and Minimization Techniques for Embedded DSP Software,'' Proc. of the IEEE Trans on VLSI Design, pp.123-135, March 1997.