Multicriteria Energy Efficient Source Code Compilation for Dependable Embedded Applications

Naeem Zafar Azeemi†

COMSAT Institute of Information Technology
Plot # 30, Sector H-8, Islamabad
Email: nzafar@nt.tuwien.ac.at

Abstract

The growing trend towards ubiquitous computing at handheld devices has brought the battery life time issue to the forefront. Scaling down the integrated circuit geometry has given rise to other issues e.g., leakage current and dynamic energy. Systems are software running on hardware; software directs the hardware components and is major contributor to the energy consumption. In this paper, we present compiler directed technique that take advantage of optimization slacks, scheduling slacks and linker slacks to optimize the dynamic energy consumption. Our framework is implemented in two phase. In first phase, profile of software application as well as underlying hardware is captured, followed by the code transformation in second phase. The optimization search engine is powered by genetic algorithm. We present results for 10 widely used multimedia applications, and analyze their behavior for parallelism, live CPU register usage, anticipated scheduling, CPU bus activity, processing units utilization and binary code size. Finally impact of these factors are studied on objective functions e.g., speedup, energy saving etc... Our result show that a unified scheme optimizes embedded source code better than the conventional multiphase approaches in VLIW processors.

Keywords:

Dependable Embedded Applications, Low Energy, VLIW (Very Long Instruction Word) Processor, Genetic Algorithm.

1 Motivation and Context

The demand for handheld multimedia applications has exploded in the recent years. In the same vein persistent increase in computer performance has been accompanied by a commensurate increase in energy dissipation. The energy efficiency of these systems today depends heavily on their software design [1, 2, 3]. As a result the efficient source code and energy consumption optimization has become the primary requirement of embedded system-level design methodologies. In energy sensitive scenario it is vital to research new energy optimization techniques, which should focus on optimizing energy saving while keeping performance constraints such as execution cycles and architectural usage [4, 5, 9, 10].

The tradeoff to consider energy issue at both application and architecture level includes the offline time of compilation and optimization [5, 6]. Knowing the fact, embedded applications are architecture dependable applications, which has to run for the lifetime of embedded system, the offline optimization time can be considered as worth. Though compilers have employed computation and data reordering to improve locality, this still requires expert analysis due to the obscured parallelism and communication patterns in traditional languages such as embedded C, embedded C++ etc..

The idleness of system components is an important technique to reduce energy consumption. 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-

† Currently working with Chritian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at Inst. Of Comm. and Radio Frequency Engg., University of Technology Vienna.

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.

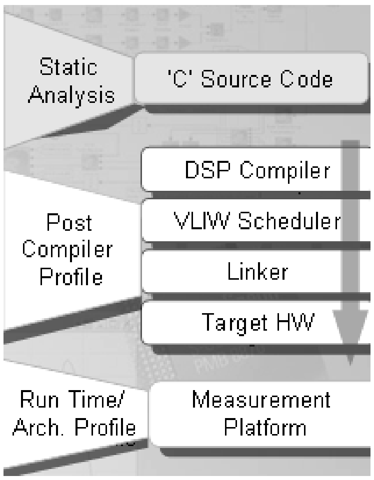


Figure 1.1. Application transformation layers and monitors.

In this paper, we make the following contributions:

- **Profiling of Application static and runtime behavior**: the main goal of such vertical profiling is to further improve the understanding of system behavior through correlation of profile information at different levels.
- **Evaluation of six energy control factors**: cache hits, processing units, anticipated scheduling factor, run time on chip live registers, CPU bus activity, and binary code size. These dynamic energy parameters have been integrated into energy aware framework built upon the Nexperia PNX 1302 native compiler environment.
- **Integration of iterative compilation engine** that utilize the Nexperia PNX 1302 native compiler environment into the genetic algorithm. In the latter case, the proposed technique eventually finds an optimal code transformation scheme for both cycle and energy efficiency.
- Mechanism is experimentally compared with the baseline version of 10 multimedia applications to their transformed code.

The remainder of this paper is organized as follows: Target architecture and framework introduction is reviewed in Section 2. Iterative compilation and optimization strategy for our approach is presented in Section 3. The experimental setup and benchmark applications are explained in Section 4. Case studies showing the significance of our methodology are given in Section 5. Finally, conclusions are given in Section 6.

2 Iterative Compilation Methodology

Here we present our energy aware framework flow [6], which aims to automate most parts of embedded system software optimization for a given VLIW processor. The embedded application development life cycle starts with high level implementation of algorithm implementation as shown in Figure 1.1. 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. Unlike in [8], we use an entirely distinct approach to prune the optimization space. We consider this problem as a single task, where all desired aims have to be taken into account simultaneously. In contrast to [13], the objective function is maximized by using the genetic algorithm (GA) [11, 12]. The fitness function of a genetic algorithm represents the objective function of the underlying optimization problem and thus has an essential impact on the optimization progress of the genetic algorithm. Our energy-aware framework [6] embodies a series of profiling stages that enable the optimization process.

2.1 Application Performance Monitors

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 [8].

2.2 Code Restructuring

Profiling is typically used to converge to optimal CPU and cache usage. 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 a good code. In [6], 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 [6]). 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.

3 Optimization Algorithm

We formulate the optimization problem as the multiple objective optimization of:

1. Energy saving
2. Operations per cycle (OPC)[2]

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, i.e. OPC 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. We incorporate the model already published in [6].

4 Experimental Methods

To validate our results, we present three different optimizations metrics e.g., static, compile time and, run time. These metrics are shown in Section 5. Section 4.1 describes our benchmarks for framework evaluation.

4.1 Benchmark Codes

To evaluate the effectiveness of our scheme, we used a suite of 10 multimedia applications from different benchmark sets. The important characteristics of these codes are given in [6, 8]. 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-2 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 MPEG-1 transcodec, MPEG-2 transcode, G-728 transcodec and generic DSP algorithms (iir, fir, dct, idct etc..). We obtained codes for these applications form various public domains sources.

5 Results and Discussion

The benchmarks were compiled for the Nexperia DSP PNX1302 platform on 200MHz board running pSoS operating system, using the TriMedi C compiler cc and the optimization level –O3 –G. The output file sizes were selected to yield at least several tens of billions of instruction executions for all of our computational biology benchmarks. For our benchmark codes, we skipped the first 100 million instructions and collected data for the next 1 billion instructions or until completion. All analyzed benchmarks were validated against precompiled binaries provided in the original distributions of the benchmark suites. We use the TriMedia *sim* [7] simulator v2.2.2 of tcs2.0007 to run the benchmarks and collected data. The benchmark profiles were obtained using the TriMedia *tmprof* profiling simulator, and the performance data was collected using our framework [8].

5.1 Impact of Anticipated Scheduling

We obtain scheduling factor 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 5-issue slot machine with instruction level parallel operation constraints as mentioned in [7]. Figure 5.1(a) shows the percentage improvement in each benchmark application to the base line code. Inherently due to highly branch oriented coding Viterbi decoder and MP3 has higher scheduling factor i.e. 17% and 20% respectively. Image compression codec JPEG2000 is dominated with deep nesting, an implicit feature of its wavelet algorithms, those results into low scheduling factor (2%). Whereas video transcodecs reflects moderately for the scheduling factor, e.g., MPEG2 (11%), H263 (11%).

5.2 Impact of On-chip Units

Our hardware architecture offers high degree of parallelism, a favorite choice for applications pertaining higher temporal and spatial data independence. MPEG4, MPEG2 and JPEG2000 reflect such behavior in Figure 5.1(b). 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 idct (inverse direction cosine transform), the low cache hit (12%), causes low bus activity (1%), entailed by low anticipated scheduling factor (1%), eventually leads to small improvement in energy saving (12%) and reduction

[2] Operation per cycle is obtained from number of executed operations and execution cycle.

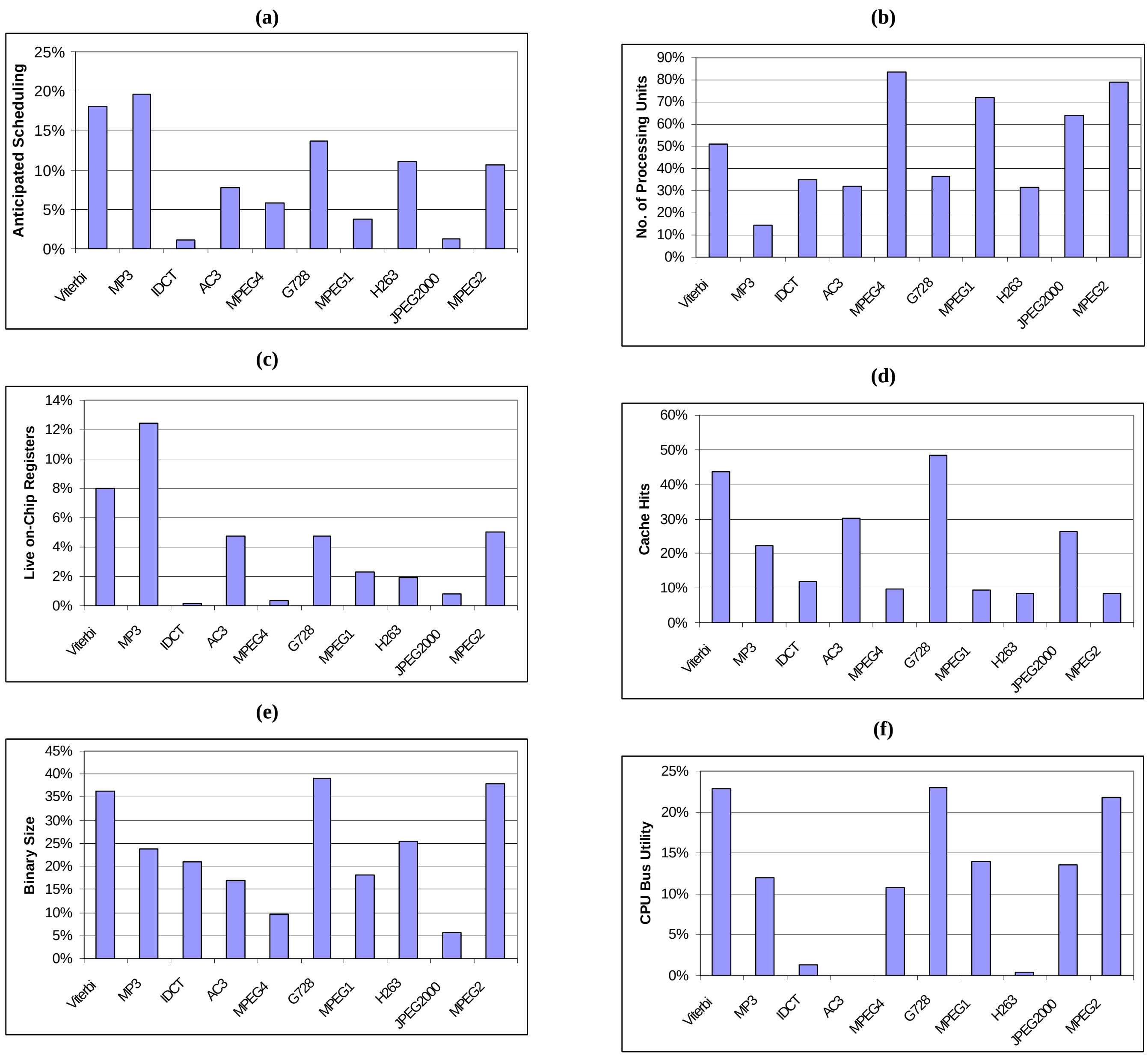


Figure 5.1. (a-f) Sensitivity of architectural features to benchmark codes

in execution cycles (24%). A careful consideration to Figure 5.1 (a-f) reveals the fact that VLIW architecture is well suited for video transcodecs e.g., MPEG2. The implicit spatial and temporal parallelism in MPEG2 algorithm let it to exploit CPU 5-issue slots upto 79%, raising bus activity to 22%, leading to an energy as well as speed efficient application.

5.3 Performance Evaluation

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.

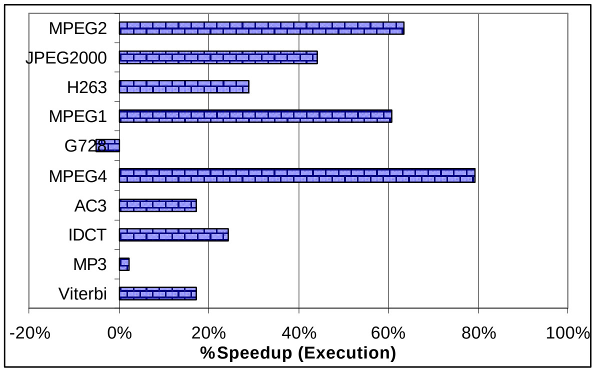
5.3.1 Speedup Efficiency

Applications runtime profile in Figure 5.1, clearly conclude to the fact that there is strong correlation between the scheduling factor, cache miss, processing unit usage and speedup factor. The average cycle efficiency of video transcodecs MPEG2, MPEG1 and MPEG4 is higher than the other algorithms due to higher spatial and temporal code execution locality. While Viterbi decoder and speech codec e.g., G-728 are not suitable applications for our hardware Figure 5.2 (a).

5.3.2 Energy Efficiency

Figure 5.2(b) gives the energy saving over the baseline code for all benchmark applications. Note that MPEG1 (36%), MPEG2 (45%), MPEG4 (40%), JPEG2000 (23%) appeared as highly energy efficient application as compared to other applications in benchmark. This improvement is result of architectural utilization and we have already discussed them above.

(a)



(b)

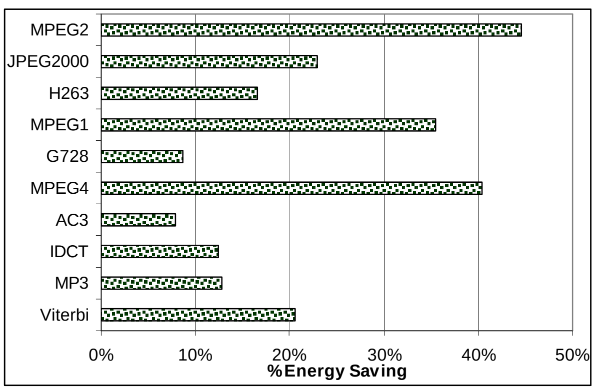


Figure 5.2. Benchmark evaluation for speedup (a) and energy saving (b) after transformations.

6 Conclusions

This work presents a novel approach to optimize energy consumption of dependent multimedia applications for VLIW architecture. The basic idea is to profile application at static, compile time, and run time. Based on the measured profile, optimization space is searched using genetic algorithm for a good solution. Scheme is evaluated for widely used multimedia applications. We present results for 10 widely used multimedia applications, and analyze their behavior for parallelism, live CPU register usage, anticipated scheduling, CPU bus activity, processing units utilization and binary code size. Our result show that the unified scheme optimizes embedded source code better than the conventional multiphase compilation, linking, scheduling for VLIW processors.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

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

[13] 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.