Architecture-Aware Hierarchical Probabilistic Source Optimization [1]

N. Z. 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
Emails: nzafar@nt.tuwien.ac.at

Abstract

This paper focuses on architecture-aware source-level code transformation methods for low energy consumption in complex VLIW processors. Though energy issue has long been addressed at hardware level, but a significant contribution of software applications can not be ignored. Successive energy-cycle cost guarded parametric optimization are applied in proposed iterative compilation environment to evaluate an application expression profile on complex multimedia processors. The optimization is based on a multi objective function. The constraints of this objective function are formulated using a penalty method; a genetic algorithm finds solutions eventually. The proposed new code transformation methodology determines appropriate parameters for compiler optimization in order to satisfy user constraints on code size, energy, execution time and optimal target architecture usage. The set of experiments presented here characterize the newly introduced approach, while giving an idea about the architecture-based energy-cycle performance issues. Experimental results show that our approach reduces cache misses by an average of 33% (max. 67%), improves typically energy dissipation up to 23% and CPU performance up to 80% for an MPEGdec video algorithm. The approach is general and can easily be integrated in multimedia processor compilers.

Keywords:
Multi-Objective Optimization, Source Transformation, Energy-Aware, Multimedia Applications

1 Introduction

Implementation of complex frame-based media functions on high performance multimedia processors with small form factor in recent years have been accompanied by a tremendous increase in power and energy dissipation. The later is implicitly connected to a demand for expensive packaging and cooling technology, an increase in product cost, and a decrease in product reliability in all segments of the computing domains. Moreover, the higher power/energy dissipation has significantly reduced battery life in portable systems.

The energy efficiency of these systems depends heavily on their software design [1]. Multimedia compilers are used to generate machine executable codes. Traditional compilation techniques generate poor machine code for handling irregular multimedia DSP processor architectures with respect to architecture usage, execution time, code size, and energy consumption [5]. 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 'C'. Fueling this trend is the conventional application programmer's coding styles. Generally embedded programs are developed in isolation, mostly ignorant to the application-architecture correlations; leads to poor machine code generation. The task of machine code generation can be subdivided into the subtasks code selection, instruction scheduling, register allocation as well as address code generation. Finding an optimal solution for each subtask is known to be an NP-hard optimization problem.

The above issues indicate that for a 'good' energy-cycle performance there is a need to gather more detailed profiles, containing information about system behavior on

[1] This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

various levels (see Figure 1.1). The main goal of such multi-level profiling is to further improve the understanding of system behavior through correlation of profile information at different levels.

This paper introduces the concept of energy-cycle cost guarded parametric optimization for iterative compilation. The idea in iterative compilation is to compile an application with different optimizations strategies and then identify the optimization among these. We demonstrate that it is necessary to use an application expression profile at all layers to understand existing performance problems such as poor architecture usage, increased execution time, and high energy consumption.

Unlike [2], we use an entirely distinct approach to prune the optimization space. For a candidate application, a set needs to be selected against objectives relating to user constraints. This set results in a huge size of the search space and its search complexity makes it necessary to find a balanced trade-off between space exploration and a focused search. We consider this problem as a single task, where all desired aims have to be taken into account simultaneously. In contrast to [3], the objective function is maximized by using the genetic algorithm (GA). 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. Hence, transformation framework [19] enables code restructuring according to different objectives by specifying a suitable fitness function such as minimization of execution time and energy aware optimization.

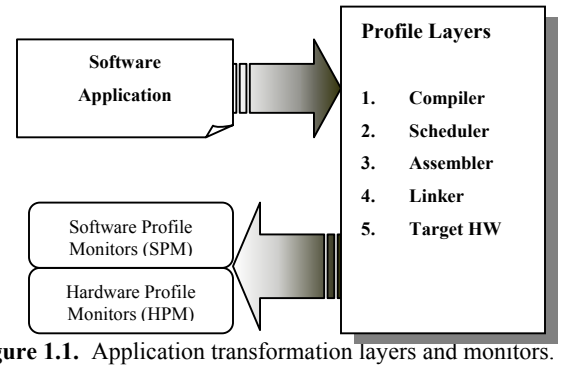

Figure 1.1. Application transformation layers and monitors.

The main contributions of this paper are:

- Application expression profiling at all levels is considered, for capturing and correlating performance problems across multiple execution layers.
- Multiobjective optimization function is used, where constraints are formulated using a penalty method and a genetic algorithm is used for solving the optimization problem.
- Readily portable to other embedded processors.

This paper is organized as follows: Section 2 describes briefly our low energy source-to-source transformation framework. Optimization model is presented in Section 3. Section 4 shows our experimental results. Relevant work on energy estimation and optimization is summarized in Section 5 and we conclude in Section 6 with some final remarks.

2 Transformation Methodology

In this work, we further extend our profile monitors, not only in hardware profile monitors (HPM) as well as in all application transformation layers (as shown in Figure 1.1). In our energy-aware framework published in [19] we use HPM to analyze the application expression profile. We add software performance monitors (SPM) to observe the behavior in the layers above the hardware. A typical list of HPMs and SPMs is shown in Table 1 that we use in our extended framework.

2.1 Transformation Flow and Performance Monitors

In this section we described the methodology flow in our framework [19]. The 'C' source code is processed successively for static code analysis, post compiler analysis and finally for scheduling analysis. A processor descriptor file is used to provide architecture information to compiler, scheduler and finally to the machine code generator. This file contains a list of pseudo and machine operations, the latency of the operations, the opcodes, the slot assignment schemes, the processor operating frequency, the instruction cache feature (associativity, block size, number of sets) as well as main memory features (size, order, read/write latencies).

Table 1. Profile Monitors

Software Profile Monitors (SPM)	Hardware Profile Monitors (HPM)
• Loop size	• Main memory features
• Operation length	• Cache features
• No. of useful inst.	• Execution time
• Code size	• Energy dissipation
• Scheduling factor	• Instruction completed
• Cyclomatic complexity	• Cache misses
• Exec. edge probability	• Slot utilization

During the code processing flow, intermediate trace files are generated to obtain SPMs, e.g., code size, execution time number of cache miss (for both instruction and data caches), highway usage, scheduling factor, and slot utilization. After simulation these parameters are used to compute transformation control factors such as unrolling factor, grafting depth and blocking metrics. Successively, after each cycle, all these parameters are computed again

and are compared to constraints mentioned in the user constraint file. This file contains user constraints, to be used in maximizing objective function.

Energy is measured at the target platform (the setup is explained in [19]. All these parameters are fed back to the transformation cost analyzer. In each successive transformation it is decided that whether energy-cycle performance has been optimized or not. Source code is optimized by undergoing code restructuring schemes known as loop unrolling, decision tree grafting, and loop tiling.

3 Optimization Algorithm

Typically, machine code generation process is performed in two phases: front end and back end compilation. The front end analyses the source code to build an internal representation of the program, called the intermediate representation (IR). Whereas, back end analyses uses a call graph [1] and a control flow graph for the code optimization that is later translated into the output language, usually the native machine language of the system (target architecture). Thus, energy aware optimizations made in early code generation phases are potentially nullified in subsequent optimization steps. For example in tree based compilation techniques transition rate of successive instructions changes if spill code [21] is inserted in a separate phase. On the other hand, performing a graph based code selection and a phase coupled code generation results in solving a more complex problem. Special optimization methods are required, which are capable of finding 'good solutions' rather than optimal solutions in a huge search space in polynomial time. Genetic algorithms (GA) have proven to solve complex optimization problems by imitating the natural evolution process [20].

3.1 Multicriteria Objective Function

We optimize the energy-cycle competition by introducing them as a single task, where all desired aims have to be taken into account simultaneously. This is achieved by an optimization of a multiple criteria, objective function. The desired aims of architecture-based energy-cycle optimization are formulated as penalty terms of such objective function. The maximization of objective function is achieved using a genetic algorithm (GA). A genetic algorithm belongs to the class of stochastic optimization methods [26]. Although it does not guarantee finding of the global optimal solution, the result is a good approximation of it. The concept of the GA allows for working parallel with many feasible solutions (individuals) by operating between these solutions. Because of working with many solutions in parallel, it is improbable that the genetic algorithm stalls in any local optimum and thus likely that it finds the global solution.

Assume that all possible transformations are known. The assumption is sound because optimization space is in practice limited by architectural constraints, e.g., number of available functional units, best fit for code block in cache. By using static and runtime parameters of certain applications, the transformation scheme is determined for every possible code restructuring.

We have two objects for the optimization: operations per cycle C and energy efficiency E. For every measure point of the transformation space, it is demanded that:

1. The system dependent limits must be satisfied i.e., successive architecture utilization (in terms of functional units, internal register usage, best cache fit) must be greater than a predefined, system constraints (execution cycle and energy threshold).
2. The smooth optimization over two samples of code is defined by minimum and maximum limits of transformed code. Therefore, predecessor transformation scheme must overlap the successor in order to follow this smooth optimization. Note that, if the output profile of code is between these limits then it must lie on smooth curve for optimization.

The problem is now to find that number m, m < n, of n transformation possibilities and their yielded code profile (both static and runtime) that maximizes our two targets.

We formulate the above optimization problem as the following multiple objective optimization problem with two components C and E:

$$\underset{X}{MAX}\{f(X)\} = \underset{X}{MAX}\{j_1 C(X) + j_2 E(X)\}, \qquad (1)$$

subject to individual *X*. An individual contains the information of transformation space and previous iteration. We use the GA, and model the constraints as a penalty term of the fitness function *f(X)*. The first term of the fitness function, $0 \le C(X) \le 1$, denoted the achieved fraction of the OPC* for total transformation space. The second term $0 \le E(X) \le 1$, denotes the fraction of points where the energy saving is fulfilled. Coefficients $0 \le j_1 \le 1$, $0 \le j_2 \le 1$ are weight factors to the criteria and they define the importance of different criteria with respect to each other, e.g. if $j_1=0$ and $j_2=1$ the method optimizes only energy saving. The individual sample points in transformation space are chosen with a uniform probability distribution. They are profiled later by evaluation the application expression 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

* Operation per cycle is obtained from number of executed operations and execution cycle.

successful transformations more frequently and search their neighborhood more intensively.

4 Experiments and Results

We have integrated our version of the transformation methodology into our native compiler environment [7][19] and have evaluated its ability to restructure applications compiled with aggressive transformations. We use benchmark applications as enlisted in Table 2.

4.1 Energy-cycle Performance

To evaluate performance, we compile the benchmark at native compiler for all optimizations turned on. This baseline code is further examined for successive transformation. We obtained the application expression profile (AEP) during different stages. Table 3 and Figure 4.1 show such profile monitors (both HPMs and SPMs) for example MPEGdec application. Note that our focus in this paper is on optimizing the application execution time and dissipated energy simultaneously.

Table 2. DSP benchmark applications

Applications	Description
MPEGenc	High bit-rate video coding based on the MPEG-2 video coding standard.
MPEGdec	
H264enc	Low bit-rate video coding based on the H.263 standard.
H264dec	
G.728dec	High bit rate speech coding based on G.728 standard.
G.728enc	
M100	Matrix multiplication 100x100

Table 3. MPEGdec, average static code metrics [4]

Metrics	Values
Average Cyclomatic complexity	40
Average Modified complexity	40
Average Strict complexity	42
Average Essential complexity	5

Each transformation iteration (Iter-1 to Iter-7) shown in the Figure corresponds to percent relative change to the original profile for the baseline version of MPEGdec. E.g., each successive iteration give rise to code size from 15% to 87%, while first iteration has reduced the execution time by 6% (see -6% in Iter-1 column). Similarly energy consumption is also decreased by 1% (see –1% in Iter-1 column). We observe that the performance improvement in terms of timing is correlated positively with the energy reduction. From Figure 4.1 we see the variation of slot utilization and scheduling factor causes the similar variation in execution time. Similarly instruction and data cache miss leads to variation in energy consumption. Loop unrolling improves the program execution by increasing instructions level parallelism (ILP) thus increasing size correspondingly. For MPEGdec example, execution time is reduced by 80% with a significant decrease in energy consumption, i.e., 23%.

4.2 Architectural Utilization

Functional unit utilization exploits parallelism and is shown here as slot utilization. While each loop tiling has increased misses in instruction cache (Iter-2 to Iter-3). Moreover an implicit improvement in energy can be observed due to the impact of decision block grafting made on the scheduling factor. An optimal grafting depth [19] can cause the scheduling factor to grow higher with a benefit of better data/address highway usage and slot utilization as shown in Iter-6 to Iter-7.

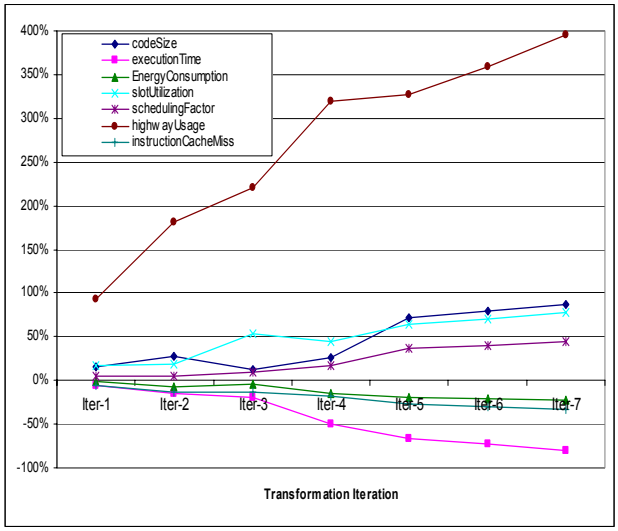

Figure 4.1 . MPEGdec profile for successive transformations

4.3 Cache Misses and Energy Dissipation Correlation

It is a well known fact that off-chip memory traffic leads to higher energy consumption, we develop a tool to visualize the energy consumption corresponding to each cache miss at function level. This helps to identify hot spots in a function that are memory hogging. Figure 4.1, depicts such relation in two consecutive functions in MPEGdec application during execution. Function getAffNeighbour ends execution at 0x0038000, and next function encode_one _macroblock starts executing. The codec was tested for video segmentation 'flowergarden'. Higher spikes can be observed in the address range 0x00600000 to 0x00700000 due to cache misses. A higher miss rate increase off-chip traffic which leads to a

significant increase in energy dissipation and is depicted in Figure 4.2.

4.4 Perturbation in Source-to-Source Transformation

The inevitable instrumentation of SPMs to the application can give rise to perturbation, the very behavior that it is trying to understand. For example,

- Instrumentation for collecting data on a software performance monitor may change the cache behavior of the application.

We address this problem by continuous recording of HPMs at HP54720D Oscilloscope with and without SPMs. Each iteration is made prior to transformation for 10 runs for all benchmark applications. The conjecture is made on visual comparison between the observed traces.

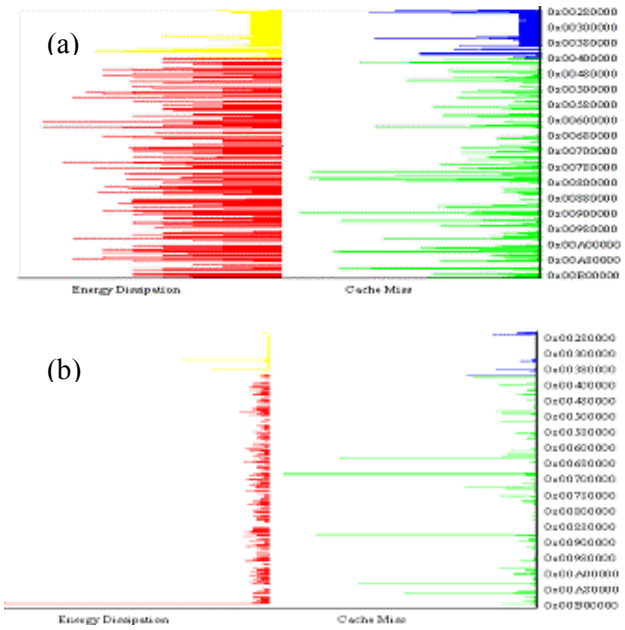

Figure 4.2.(a) Cache miss (right) and corresponding energy dissipation (left) – before transformation **(b)** Cache miss (right) and corresponding energy dissipation (left) – after transformation.

5 Related Work

Energy-cycle optimization issue has been addressed at different hardware specification layers (circuit, gate, register-transfer or behavioral); an overview can be found in [6]. These approaches are mostly revolving around either simulation of functional units in a processor or direct measurement of electrical parameters on some target hardware. A functional unit based power profiler in [9] records the traces of previous states, correlated switching capacitance and information about the current states of functional units. An extension to this work is produced as cycle-level energy estimation [10]. Cycle-level energy consumption is measured on a hierarchical decomposition of the architectural features of the target processor. However, their profiler does not give estimates of the power consumption for a given sequence of instructions. Numerous techniques have been discussed to explore the impact of source code transformations on families of hardware architectures [11]. They used instruction-level simulation to measure the effects of code transformation on energy. On the other hand, considering the processor as the most energy-critical system component, other approaches [11] focused instead on the number of processor cycles. Thus, loop unrolling and procedure in-lining were used to reduce the number of processor cycles, while data locality was improved by cache size optimization. Results obtained in direct measurement-based techniques are closer to the actual energy behavior of the processor, because the information are measured at actual hardware. A current measurement based technique is used in [12]. An energy model is made based on recording the unique base cost for each instruction and the inter-instruction effects. However, recording this inter-instruction effect significantly increases the size of the power cost table. Only a few researchers have verified these values as actual physical savings in energy [13, 14].

Some researchers [15, 16] tried to model the complex energy behavior of processors in terms of usage of their various functional units, mainly targeted for VLIW. No matter what architecture is in use, inevitably energy efficiency of the overall system depends heavily on software applications. Low energy software design can be achieved in different ways; either by energy aware selection of the algorithms [17] or by code transformations [8]. Attention has also been given to explore architecture-level models to be used in conjunction with higher level tools or as part of a simulation environment [18].

6 Conclusions

In this paper, we demonstrate that it is necessary to profile application expression at all layers to understand existing performance problems such as poor architecture usage, increase in execution time, and high energy consumption. We use a GA to prune the optimization space. The proposed methodology facilitates the programmer in being the strategist. A goal-driven canned set of transformations may improve the application significantly. We examined the performance improvement across typical multimedia applications. Our experimental results indicate that our approach reduces cache misses by an average of 33% (max. 67%), improves typically energy dissipation up to 23% as well

as CPU performance up to 80% for an MPEGdec video algorithm.

Our results are important for developing a general methodology for energy-aware embedded DSP software since low power is critical to complex DSP applications in many cost sensitive markets. In addition, we expect to be able to model the application conformance to the target architecture based on their expression profile in terms of energy consumption, execution time and architecture usage.

7 Acknowledgment

This work is supported by ÖAD-Pakistan scholarship program initiated by Prof. Dr. Atta-ur-Rahman chairman HEC and Federal Minister Pakistan. We thankful to Prof. Dr. Markus Rupp, Prof. Dr. Arpad Scholtz and Christian Doppler Laboratory at The Institute of Communication and Radio-Frequency Engineering, Vienna University of Technology for their support and kind input during this work.

References

[1] 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.

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

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

[4] http://hissa.ncsl.nist.gov/sw_assurance/strtest.html

[5] 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.

[6] W. Ye, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, ''The design and use of SimplePower: A cycle-accurate energy estimation tool,'' Proc. of the Annual ACM IEEE Design Automation Conference, pp. 340-345, June 2000.

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

[8] David B. Loveman, "Program improvement by source to source transformation," Proc. of the Annual ACM SIGACT-SIGPLAN symposium on Principles on programming languages, pp.140-152, 1976.

[9] H. Mehta, R. M. Owens, M. J. Irwin, ''Instruction level power profiling, '' Proc. of the International Conference on Acoustics, Speech and Signal Processing, Oct.1996.

[10] R. Y. Chen, M. J. Irwin, R. S. Bajwa, ''An architectural level power estimator,'' Proc. of the Power-Driven Microarchitecture Workshop, 1998.

[11] Y. Li, J. Henkel, "A Framework for Estimating and Minimising Energy Dissipation of Embedded HW/SW Systems," Proc. of the Design Automation Conference. pp. 188-193, Nov. 1997.

[12] Tiwari, S. Malik, A. Wolfe, ''Power analysis of embedded software: A First step towards software power minimization,'' Proc. of the IEEE Transactions on VLSI Systems, vol. 2(4), pp. 437-445, Dec. 1994.

[13] 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.

[14] 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.

[15] J. T. Russell, M. F. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," Proc. of the International Conference on Computer Design, Oct. 1998.

[16] 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.

[17] C. H. Gebotys, R. J. Gebotys, "An empirical comparison of algorithmic, instruction, and architectural power prediction models for high performance embedded DSP processors," Proc. of the International Symposium on Low Power Electronics and Design, 1998.

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

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

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

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