American Journal of Applied Sciences

Original Research Paper

Android Based Energy Aware Framework for Porting Legacy Applications

Naeem Zafar Azeemi

College of Engineering and Information Technology, Al Ain Univeristy of Science and Technology, Al Ain, UAE

Article history
Received: 24-09-2014
Revised: 15-12-2014
Accepted: 30-12-2014

Abstract: Trend is growing towards using complex multimedia functions on smaller devices. In this study, we explore the effect of migrating legacy signal processing software applications algorithms from large form factor devices to the smaller one such as handheld mobile devices known as Energy Conscious Mobile Computing Systems (EConMCS). We concentrate on Source Code Volatility (SCV), including inherent algorithm complexity and the developer implementation. We identify code Transformation Steering Factors (TSF), such as loop unrolling factor, decision tree grafting factor and their relation to SCV. The impact of TSF is discussed for different multimedia applications in native Digital Signal Processor (DSP) compiler optimization while switching between different transformation schemes. Our results show that SCV can be minimized by using an architecture-centric algorithm that both enables the effective use of underlying hardware architectures and the memory access required to optimize energy consumption. The coded spatial access is implicitly dependent on layout, content and location of options and legibility that relates to a developer's implementation of loops, code blocks and decision trees. The compiler-centric transformation model minimizes the effect of legacy code migration for multimedia applications. Results are exposed for the transformation of typical DSP applications and a video transcodec MPEG-4.

Keywords: Multimedia Applications, Legacy Code, Embedded Systems, Source-to-Source Transformation (StS), Source Code Volatility (SCV)

Introduction

Several factors contribute to make the multimedia system a performance bottleneck. Increasing demand of intensive multimedia functions in a small form factor and pervasive computing has tightened the design space (Ye *et al*., 2000; Mehta *et al*., 1996; Chen *et al*., 2012). With the explosive growth of hand-held battery operated embedded systems, the issue of their energy consumption has gained importance. VLIW DSP processors are the most lucrative choice to such an application domain for their optimal performance delivery in high data throughput at low power (Chang *et al*., 2000; Klass *et al*., 2010; Mehta *et al*., 1987).

Hitherto energy dissipation has mostly been addressed at hardware level (dynamic supply voltage scaling, operating frequency control) but the current drive towards ubiquitous computing shifted the focus to executing software running on underlying system hardware.

Researchers (Esakkimuthu *et al*., 2000; Li and Henkel, 1997; Cathoor *et al*., 2014; Tiwari *et al*., 2012) have revealed that a large fraction of the computational load imposed by applications is handled by the CPU and it is the largest contributor to the overall energy budget. In general, CPU energy consumption depends on the type of workload imposed by applications. Therefore a strong correlation between the application binary and underlying hardware architecture leads to an efficient Energy Conscious Mobile Computing System EConMCS as shown in Fig. 1.1.

We define an energy-cycle cost model together with a source-to-source transformation methodology, suitable for embedded systems based on VLIW cores. The system level methodology includes generalized energy models for each module, composing the system architecture (processing unit, on-chip/off-chip memory units, address/data highway etc.) and SW application parameters as shown in Fig. 1.2.

Unlike (Klass *et al*., 2010; Mehta *et al*., 1987; Lee *et al*., 2011), we explore following aspects of application expression as compared to conventional techniques:

Science Publications

© 2014 The Naeem Zafar Azeemi. This open access article is distributed under a Creative Commons Attribution (CC-BY) 3.0 license.

DOI: 10.3844/ajassp.2014.1980.1987

- The impact of algorithmic complexity and developer's implementation: These effects are directly related to source code volatility and hence the architecture-application performance
- Integration in DSP Native Compilation Environment (NCE). That utilizes the conventional Software Development Environment (SDE) to produce battery efficient embedded applications
- Results are exposed for five optimization iteration at a typical signal processing algorithm

The remainder of this study is organized as follows. Relevant previous research on energy estimation and optimization is summarized in the next section. A detailed energy cost model and a successive transformation methodology is proposed in section 3. Experimental results are reported in section 4, finally in section V we draw some conclusions and outline extensions as well as improvements to our future work.

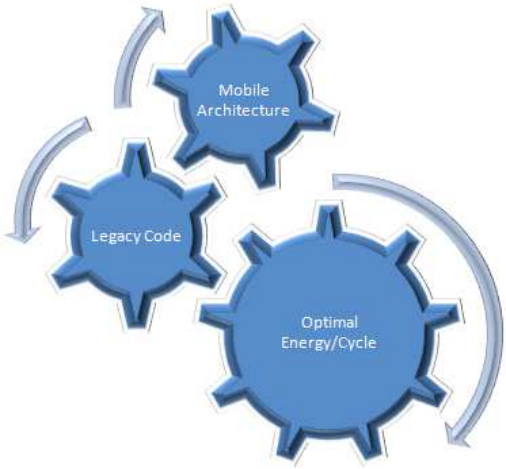

Fig. 1.1. Correlation between software application and hardware architecture

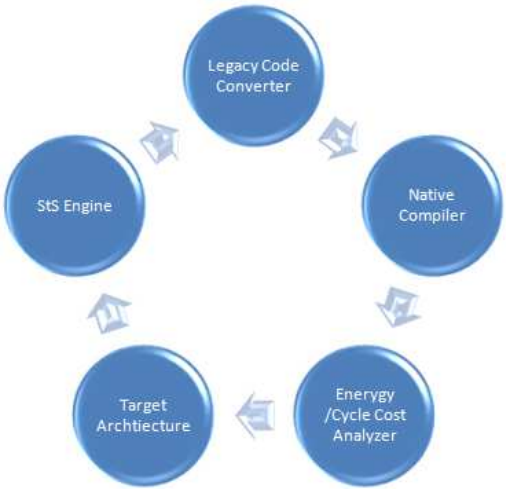

Fig. 1.2. A simplified methodology flow

Related Research

In recent years, numerous technique have evolved to address the energy consumption issue at different hardware specification layers (circuit, gate, register-transfer or behavioral); an overview can be found in (Ye *et al*., 2000). Many tools exist for power estimation and optimization at these levels, more work is needed in the area of energy analysis or optimizations at micro-architecture, architecture or system level. Approaches used in most of these tools can be broadly divided into two categories; either simulation of functional units in a processor or direct measurement of electrical parameters on some target hardware.

In simulation-based methods, energy consumption is estimated by calculating the energy consumption of various components in the target processor through simulations at different levels. Simple Power concentrates on modeling target architectures (Ye *et al*., 2000). A functional unit based power profiler in (Mehta *et al*., 1996) registers the history of previous states, information about the current states of functional units and correlated switching capacitance. Cycle-level energy estimation is reported (Chen *et al*., 2012), as an extension to (Mehta *et al*., 1996; Su *et al*., 2013). A gate-level analysis tool is used to analyze the effect of sequential execution of different instructions in (Klass *et al*., 2010).

Numerous techniques have been discussed in (Li and Henkel, 1997) to explore the impact of source code transformations on families of hardware architectures (Mehta *et al*., 1987). They used instruction-level simulation to measure the effects of code transformation on energy (Mehta *et al*., 1987; Esakkimuthu *et al*., 2000). On the other hand, considering the processor as the most energy-critical system component, other approaches (Li and Henkel, 1997) 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. Implicitly assuming data memory access as the dominant factor for both energy and performance researchers in (Cathoor *et al*., 2014) applied extensive loop transformations to improve locality and hence reduction in data accesses.

Direct measurement-based techniques are more fine-grained approaches than the simulation based methods. In these approaches software is characterized by examining the energy consumption obtained from real hardware.

A current measurement based technique is used in (Tiwari *et al*., 2012). However, recording this inter-instruction effect significantly enhance the table volume. Attention has also been given to exploring architecture-level models to be used with higher level tools or as part of a simulation environment. Microprocessors (Esakkimuthu *et al*., 2000; Gebotys and Gebotys, 2011),

DOI: 10.3844/ajassp.2014.1980.1987

controllers (Su *et al*., 2013), instruction registers, memory units, are prominent contributor to power dissipation. Researchers have tried to schedule operations (Su *et al*., 2013), or swap operands (Tiwari *et al*., 2012) to reduce data bit switching. Researchers have also employed parallel instructions to improve performance which also reduced energy such as using parallel data transfer instructions (Lee *et al*., 2011).

Only a few researchers have verified these values as actual physical savings in energy (Lee *et al*., 2011; Gebotys and Gebotys, 2011). An instantaneous power measurement model is presented in (Russell and Jacome, 1998). There, a software energy (Mehta *et al*., 1987) estimation model is proposed by measuring electrical parameters on a digitizing oscilloscope.

In contrast to above approaches (Gebotys *et al*., 2000; Gebotys and Gebotys, 1998) used a regression analysis to predict the energy consumption of software. The prediction is used to minimize the energy consumption with respect to the average current drawn. Some researchers (Gebotys *et al*., 2000; Sami *et al*., 2000) tried to model the complex energy behavior of VLIW processors. The estimation of a given transformation impact (Gebotys and Gebotys, 1998; Tiwari *et al*., 2007; Loveman, 1976) on low energy is the most critical part in code restructuring and this study proposes a strategy to this issue in the next section.

Source Code Transformation Methodology

As discussed above, a SW application may be subjected to real time performance constraints of time, space and energy targeting execution on high performance DSP processors. Constraint-driven optimization to the application can be achieved by set of rules for manipulating various representations of a program. These rules allow exploitation of local or global invariance within the program according to a measured or a speculated performance cost function. In this section we shall propose an energy-cycle cost formulation for source-to-source transformation to improve energy-cycle performance of an application.

We have assumed that any typical multimedia algorithm can be coded as a tree-structured representation of a program and that the source-to-source transformations are expressible as pattern-directed rearrangements of coded text.

Figure 3.1 depicts the methodology framework. The VDF file contains instruction set operation code, implicit latencies and their mnemonics, the operations, opcodes, slot assignment schemes, processor operating frequency, instruction cache feature (associativity, block size, number of sets) and main memory features (size, order, read/write latencies). All naming conventions specific to VLIW architecture we used here are followed in (TM1300 Data Book, 1999).

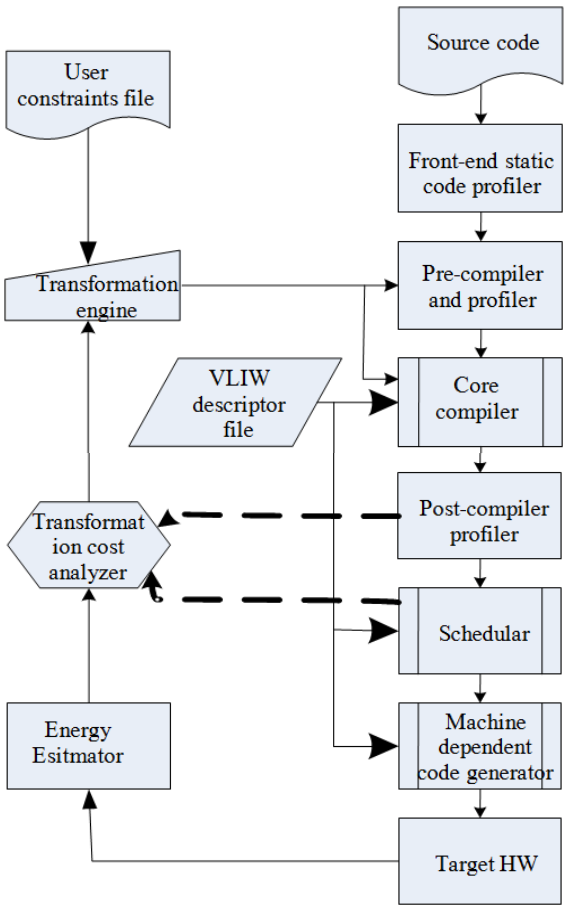

Fig. 3.1. Transformation methodology

The transformation space for steering parameter can be represented in following vector notations:

$$w_j^\theta = \left\{ \begin{matrix} freq, instructionCacheFeatures, \\ dataCacheFeatures, mainMemoryFeatures \end{matrix} \right\}$$

$$w_k^\theta = \left\{ \begin{matrix} codeSize, \max UnrollingFactor, \\ \max GraftingDepth, cacheLineSize \end{matrix} \right\}$$

$$w_p^\theta = \left\{ \begin{matrix} executionTime, Energy, \\ slotUtiliztion, schedulingFactor \end{matrix} \right\}$$

$$w_m^\theta = \{decisionTreeGrafting, loopUnrolling, loopBlocking\}$$

where, $\left(w_j^\theta, w_k^\theta, w_p^\theta\right)$ are static and runtime parameters and w_m^θ are transformation operation.

A User Constraint File (UCF) holds the result of transformed outcome that is the desired values for code, execution time, energy and allowed percentage cache miss.

If the transformation outcome is not sufficient to satisfy the accuracy constraints (i.e., given in UCF file), the quality of transformation controlling factor (elaborated in section III.C) changed and verified through simulation.

DOI: 10.3844/ajassp.2014.1980.1987

Additional benefits are gained by combining traditional compiler optimization algorithm, such as constant and variable propagation, dead code elimination, strength reduction etc.

Transformation Cost Model

Our first goal is to simplify the complexity of the processor energy model without sacrificing the accuracy of the results. The second goal is to introduce a methodology that automatically rebinds the instruction set with respect to the average functional energy cost, in order to converge to a highly effective design space.

For a given Mediabench application θ composed of a finite number of code blocks, transformation space is defined as:

$$S\left(\left(w_j^\theta, w_k^\theta, w_p^\theta\right) w_m^\theta\right)$$

We obtain w_j^θ from processor datasheet (TM1300 Data Book, 1999), w_k^θ is acquired after the pre-compiler and profiler stage in Figure 3.2. whereas w_p^θ is an outcome of the simulation at the target hardware.

For an MPEG-4 example these measured values are shown in Table 1. The parameter w_m^θ is processed in a feedback loop where transformation cost is analyzed followed by a transformation engine that decides whether the code should be transformed as proposed.

We assume that the application θ can be broken down into a set of blocks B e.g., decision blocks, data blocks, computation blocks. The total application execution time for the baseline version can be written as:

$$\tau_0(\theta^0) = \sum_{j=1}^{m}\sum_{i=1}^{n}((z_{ij}^0 q_{ij}^0 B_j^0) + \eta_{ij}^0 + \rho_{ij}^0)\ \forall m, n \in \theta \quad (1)$$

Where:

z_i^0 = Instruction cycle count,

q_i^0 = Number of instructions in block B_i^0,

η_i^0 and ρ_i^0 = Number of instruction and data cache stall cycles respectively.

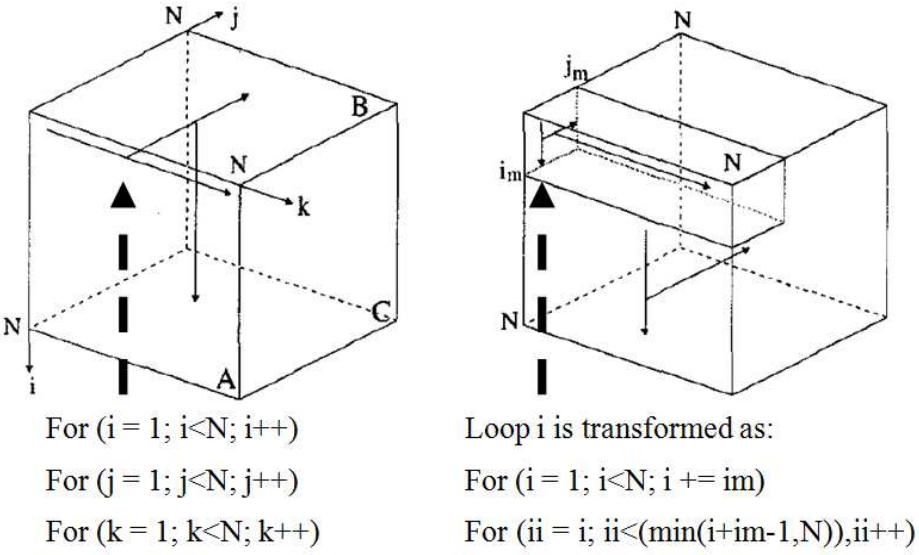

Fig. 3.2. Loop tiling-an ijk loop example; original loop (left), blocking for i loop (right) (Lam *et al*., 2011)

Table 1. Successive transformations for MPEG-4 example

Parameters	Iter-1 (%)	Iter-2 (%)	Iter-3 (%)	Iter-4 (%)	Iter-5 (%)
Size of binary	0	11	-2	10	17
Time of execution	33	63	63	72	75
Energy	7	23	15	28	30
Slot utilization	10	12	41	32	53
Scheduling factor	0	0	1	32	95
Highway usage	46	170	179	266	303
Instruction cache miss	1	2	2	3	4
Data cache miss	0	2	2	2	4
Data cache conflict	0	-100	-100	-200	0
Data bank alignment	-60	-240	-280	-540	-760

DOI: 10.3844/ajassp.2014.1980.1987

All of them are an outcome of the static and runtime execution of application as shown in Table 1.

For any p-th transformation (iteration) with cycle reduction function $\varphi_p>0$, the execution time can be written as:

$$\tau_p(\theta^p) = \varphi_p \tau_0(\theta^0)$$

And for optimization:

$$\tau_p(\theta^p) < \tau_0(\theta^0)$$

Similarly the energy dissipation for baseline version can be written as:

$$\varepsilon_0(\theta^0) = \sum (p_0 + p_{mp} + p_{q/q-1})$$
$$\sum_{j=1}^{m} \sum_{i=1}^{n} ((z_i^0 q_i^0 B_j^0) + \eta_i^0 + \rho_i^0) \ \forall m, n \in \theta \quad (2)$$

Where

p_0 = Power consumption of the idle target processor.
p_{mp} = Power consumption of the monitor program.
$p_{q/q-1}$ = Power consumption of instruction *q* while *q-1* has been executed.

For any p-th transformation (iteration) with energy reduction function $\Psi_p>0$, the energy dissipation can be written as:

$$\varepsilon_p(\theta^p) = \psi_p \varepsilon_0(\theta^0)$$

And for optimization:

$$\varepsilon_p(\theta^p) < \varepsilon_0(\theta^0)$$

Convergence Criteria for Optimization

Now given an instance to optimize both execution time and energy of application software, the transformation space is very complex.

Finding solution to this is clearly NP-complete; as parameters defined in space *S* have a large number of possibilities to get optimal solution for our goal.

We solve this problem by defining 5-tuple transformation rule as shown below:

$$r = \{codeSize, executionTime, Energy, cacheMiss, slotUsage\}$$

Each individual tuning parameter in *r* can have a value from {1,0,-1},

For example in an idct example cycle driven r_φ could be:

$$\{1,-1,0,-1,1\}$$

It allows successive transformation steps to increase code size, lower execution time, maintain same level of energy as obtained in the previous iteration, decrease cache miss and exploit more parallelism with higher slot usage.

We shall discuss more formation of the rule tuple in section III.D. Now we shall formulate steering factors that control transformations in w_m^θ.

Methodology Control Variables and Their Relations

In this section, we describe the cost estimators of the transformation techniques which determine when to cease iterations in the transformation engine shown in Fig. 3.2.

Loop unrolling (k), we propose a simple and novel unrolling strategy to find the optimal unrolling factor with a single set of profiling measurements. A successive loop unrolling factor for the i-th iteration is shown here:

$$K_i = (loopSize.VLIWinstSize) / (cacheBlockSize.associativity.noOfSets)$$

In our case the instruction cache block size is 64, associativity 8 and number of sets are 64. While instruction cache hit ratio is obtained during simulation as shown in Table 1.

Decision tree is the scheduling equivalent of an extended basic block. It is a code region that has a single entry point and zero or more exit edges leading to other decision trees or function exits.

We compute the grafting depth ς in terms of code size Ω, probability of execution edges in a tree ϑ and number of execution counts ν. For a decision tree block *j,* grafting depth is formulated as:

$$\varsigma_j = \Omega_j \nu_j \vartheta_j$$

Based on cache size we decide the maximum depth factor ς_{max}. ς_{max} is the largest depth factor not to increase the code size larger than the instruction cache size that i.e.,:

$$\varsigma_{max} = \text{instructionCacheSize} / \varsigma_j$$

Thus, the optimal depth factor ς_{opt} is the greatest divisor of Ω but smaller than ς_{max}.

Block algorithms use data and computation diagrams, rectangular parallelepipeds that shows the iteration space of an algorithm with the operations inside and the data on the faces.

DOI: 10.3844/ajassp.2014.1980.1987

A typical threefold nested counting loop (ijk-loop) is shown in Figure 3.2. The arrows show the order of operations and accesses to data. In this case a block algorithm can be obtained by performing two transformations to the algorithm. First, each of the three original nested loops is partitioned into two loops, an internal computation loop and an external control loop.

We use two performance metrics closely related to each other (Lam *et al*., 2011). On one side, we use Cycle Per Instructions (CPI) that is computed from number of execution cycle and code size both are obtained from profiler. On the other side data cache misses and data bank conflicts count. Both of them show directly block algorithm performance in terms of cache overhead.

Methodology Flow-Case Study

For a typical MPEG-4 example we obtain an initial measurement after simulating the baseline code once. This provides code size, execution time, both instruction and data cache miss rate, data bank conflicts, scheduling factor and energy. There are many other parameters that are obtained directly or indirectly from the profiler are not tabulated for e.g., foreground memory (internal register) used, number of slot assigned in individual cycle.

We will use them to refine our model to high granularity in future. In the transformation cost analyzer block all these measurements are used to compute the unrolling factor K, grafting depth ς and block performance metrics. At the transformation engine they are further used to decide, whether the current code should go for code conversion or not.

Example

If measured energy is higher, then the energy constraints are set by the user in user constraint file then further unrolling, tiling and grafting would be required. In this case the energy driven transformation rule for r_{Ψ} will be {1,0,-1,0,1}, that can be interpreted as for next iteration code size shall be increased, number of execution cycle shall constant, energy count shall go lower, cache hit shall remain same and slot usage will be increased further. Each successive transformation shall bring all cost factors close to the user constrained region as defined in the user constrained file.

Experimental Platform

Typically, VLIW core based evaluation boards have dual supply voltages, one for the core V_{dd} and other for peripherals V_{cc}. Therefore, its power dissipation contains two components of currents, i.e., I_{dd} and I_{cc}.

The core voltage in our target processor board based on Star Core SC1100 is V_{dd} = 2.5V, whereas Peripheral voltage V_{cc} is adjustable to 3.3V or 5.0V (we set it to 3.3V).

Although traditional digital multimeter (e.g., FLUKE 85) can be used to measure processor currents (Idd and Icc), switching activity between multitudes of states in VLIW processors cannot be observed by these dual-slope mode slow sampling measurement devices.

In order to record the impact of non-periodic behavior of programs, we use HP54720 Hewlett Packard Programmable digitizing oscilloscope, HP54721A Hewlett Packard Amplifier plug-in and PNX1302 evaluation board.

Results

In line with the proposed methodology described above, we measured static and architecture driven application parameters in different profiling stages enlisted in Table 1.

There are several cogent observations that can be made from our study to test applications, e.g., transformations are not applied in random order; an attempt to transformation is only made when transformation engine decides controlling parameter (K, ς and block performance metrics) are within limits and desired performance variables (execution time, energy) are closely approached.

Table 1 shows results for successive transformations applied to the baseline version of a typical MPEG-4 example. Note that the code size is increased in the beginning due to loop unrolling but it does increase processor functional unit utilization. Successive application of transformation based on 5-tuple rules improves instruction rebinding that increases scheduling factor.

Note that scheduling factor is computed as a relative measure, which is ration between the mappings at available functional units (mentioned in VDF file) to the infinite functional units (and ideal machine). This gives us better cycle improvement upto 75% (shown as executionTime) and lower energy consumption to 30%.

An inappropriate 5-tuple rule selection could lead to underutilization of internal registers and hence adds an offset to energy consumption in comparison to the previous iteration; one such observation can be made as from iter-2 to iter-3. The payoff for both energy and cycle cost factors (Ψ, φ) in this particular case are depicted in Figure 4.1.

Here, we summarize some interesting conclusions from Figure 4.1.

First we have found that the most difficult problems are concerned with transformation ordering and information gathering.

Second, although a transformation may be applicable, it may not win an improvement in the program.

Third, the distraction between machine-dependent and machine-independent portions of our transformation methodology is more subtle than it appears. A transformation on a program may be machine independent, in the usual sense, but the reason for applying it may well depend on the target machine architecture.

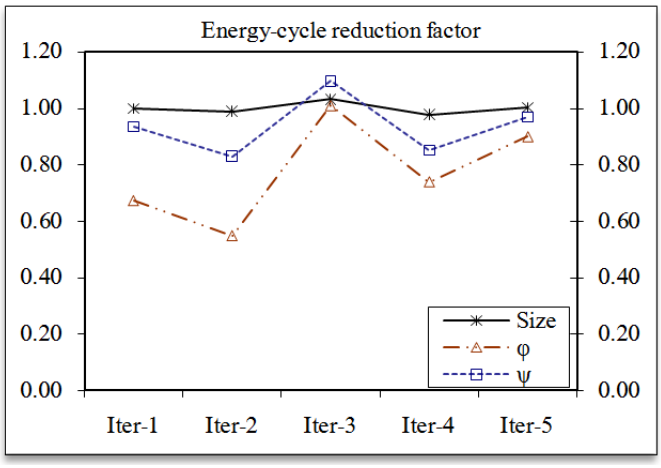

Fig. 4.1. Energy-cycle reduction factor versus Transformations (for MPEG-4 example)

Table 2. Energy-cycle cost factor for optimized applications

App/Transf.	fir	iir	Dct	idct	nlivq	m100
Size	1.76	1.38	0.99	1.54	1.03	2.32
Φ	0.90	0.67	0.55	0.98	1.01	0.74
Ψ	0.97	0.93	0.83	0.84	1.10	0.85

Fourth, a number of interesting transformations were identified. In particular the concept that a variable use may on occasion be replaced by an expression representing an assertion about the value of the variable is quite powerful.

We apply our technique to well know computational intensive examples from Mediabench fir, iir, dct, idct and two data intensive applications: Nonlinear vector quantization (nlivq) for image zooming application and matrix multiplication (m100).

Results energy/cycle cost factor for optimal transformation are shown in Table 2.

Conclusion

In this study, we explore the effect of migrating legacy signal processing software applications algorithms from large form factor devices to the smaller one such as handheld mobile devices. We concentrate on source code volatility, including inherent algorithm complexity and the developer implementation. Successive transformations are steered by a set of rules, generated in each iteration based on loop unrolling factor, grafting depth and blocking factor.

The proposed methodology facilitates the programmer to be the strategist. A goal-driven canned set of transformations may improve the application significantly. The approach is illustrated using functional unit usage within a VLIW architecture and identifies a new operation rebinding technique for low power which improves energy dissipation for a MPEG-4 example. This improvement is primarily achieved by improving the number of CPU cycles (execution time), cache memory access (both instruction and data cache) and exploiting architectural usage especially increasing slot utilization.

The approach is general and results are verified with real power measurements at StarCore Media Processor.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Cathoor, F., S. Wuytack. E. De Greef, F. Balaaa and L. Nachtergaele *et al*., 2014. Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design.

Chang, N., K.H. Kim and H.G. Lee, 2000. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. Proceedings of the International Symposium on Low Power Electronics and Design, Jul. 25-27, Rapallo, Italy, pp: 185-190. DOI: 10.1145/344166.344576

Chen, R.Y., M.J. Irwin and R.S. Bajwa, 2012. An architectural level power estimator. Proceedings of the Power-Driven Microarchitecture Workshop.

Esakkimuthu, G., N. Vijaykriehnan, M. Kandemir and M. Irwin, 2000. Memory system energy: Influence of hardware-software optimizations. Proceedings of the International Symposium on Low Power Electronics and Design, Jul. 25-27, Rapallo, Italy, pp: 244-246. DOI: 10.1145/344166.344612

Gebotys, C. and R. Gebotys, 2011. Statistically based prediction of power dissipation for complex embedded DSP processors. Microproc. Microsyst. J., 23: 135-144. DOI: 10.1016/S0141-9331(99)00030-7

Gebotys, C., R. Gebotys and S. Wiratunga, 2000. Power minimization derived from architectural-usage of VLIW processors. Proceedings of the Annual ACM IEEE Design Automation Conference, Jun. 05-09, Los Angeles, CA, USA, pp: 308-311. DOI: 10.1145/337292.337426

Gebotys, C.H. and R.J. Gebotys, 1998. An empirical comparison of algorithmic, instruction and architectural power prediction models for high performance embedded DSP processors. Proceedings of the International Symposium on Low Power Electronics and Design, Aug. 10-12, Monterey, CA, USA, pp: 121-123. DOI: 10.1145/280756.280824

Klass, B., D.E. Thomas, H. Schmit and D.F. Nagle, 2010. Modeling inter-instruction energy effects in a digital signal processor. Proceedings of the Power-Driven Microarchitecture Workshop.

Lam, M.S., E.E. Rothbcrg and M.E. Wolf, 2011. The cache performance and optimizations of blocked algorithms. Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, Apr. 08-11, Santa Clara, CA, USA, pp: 63-74. DOI: 10.1145/106972.106981

Lee, M., V. Tiwari, S. Malik and M. Fujita, 2011. Analysis and minimization techniques for embedded DSP software. IEEE Trans VLSI Design.

Li, Y. and J. Henkel, 1997. A framework for estimating and minimizing energy dissipation of embedded HW/SW systems. Proceedings of the 35th annual Design Automation Conference, Jun. 15-19, San Francisco, CA, USA, pp: 188-193. DOI: 10.1145/277044.277097

Loveman, D.B., 1976. Program improvement by source to source transformation. Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages, (PPL' 11), ACM New York, NY, USA, pp: 140-152. DOI: 10.1145/800168.811548

Mehta, H., R. Owens, M. Trwin, R. Chen and D. Ghosh, 1987. Techniques for low energy software. Proceedings of the International Symposium on Low Power Electronics and Design, Aug. 18-20, IEEE Xplore Press, Monterey, CA, USA, pp: 72-75. DOI: 10.1145/263272.263286

Mehta, H., R.M. Owens and M.J. Irwin, 1996. Instruction level power profiling. Proceedings of the International Conference on Acoustics, Speech and Signal Processing, May 7-10, IEEE Xplore Press, Atlanta, GA, pp: 3326-3329. DOI: 10.1109/ICASSP.1996.550589

Russell, J.T. and M.F. Jacome, 1998. Software power estimation and optimization for high performance, 32-bit embedded processors. Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors, Oct. 5-7, IEEE Xplore Press, Austin, TX, pp: 328-333. DOI: 10.1109/ICCD.1998.727070

Sami, M., D. Sciuto, C. Silvano and V. Zaccaria, 2000. Instruction-level power estimation for embedded VLIW cores. Proceedings of the 8th International Workshop on Hardware/Software Codesign, May 03-05, San Diego, CA, USA, pp: 34-38. DOI: 10.1145/334012.334019

Su, C.L., C.Y. Tsui and A.M. Despain, 2013. Saving power in the control path of embedded processors. IEEE Design Test Compute., 11: 24-31. DOI: 10.1109/54.329448

Tiwari, V., S. Malik and A. Wolfe, 2007. Compilation techniques for low energy.

Tiwari, V., S. Malik and A. Wolfe, 2012. Power analysis of embedded software: A First step towards software power minimization. IEEE Trans. VLSI Syst., 2: 437-445. DOI: 10.1109/92.335012

TM1300 Data Book, 1999. Philips Electronic, North America Corporation.

Ye, W., N. Vijaykrishnan, M. Kandemir and M.J. Irwin, 2000. The design and use of SimplePower: A cycle-accurate energy estimation tool. Proceedings of the Annual ACM IEEE Design Automation Conference, Jun. 5-9, IEEE Xplore Press, pp: 340-345. DOI: 10.1109/DAC.2000.855333