Power Aware Framework for Dense Matrix Operations in Multimedia Processors

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

ABSTRACT

In this paper we analyze¹ the use of Decision Tree Grafting, Blocking and Loop Unfolding to improve the performance of dense matrix computations on high performance multimedia processors. The analysis focuses on the practical aspects that can be observed when programming on present DSP processor with multilayered memory levels. The problem is studied on the Philips Nexperia processor. The experimental evaluation of the proposed approach results into better exploitation of functional units, memory hierarchy and highway usage of the target processor. The advantages of the proposed interactive code transformation approach are two-folds. First, effort in optimization is spent only when the program measurement (transformation cost) determines that the effort is necessary and potentially beneficial, and only on those portions of the program where the energy/cycle performance payoff appears to be high. Second, by concatenating subsequent energy/cycle profile-driven low level transformations for higher level manipulations, the system will provide the programmer with a powerful toolset. The approach is illustrated using functional unit usage within a VLIW architecture for low power, which improves energy dissipation up to 34% and CPU performance up to 87% for an idct example.

1. INTRODUCTION

In portable embedded systems energy dissipation is directly linked to battery size, weight, packaging, cooling and operating time. 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. The energy efficiency of these systems and subsystems depends heavily on their software design. 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.

This paper introduce an energy-cycle cost model and a source-to-source (StS) [21] transformation methodology, suitable for embedded systems based on VLIW cores. Figure 1.1 depicts simplified view of system level framework that includes generalised 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.

Figure 1.1: Methodology flow.

The rest of this paper 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 5 we draw some conclusions and outline extensions as well as improvements to our future work.

2. RELATED RESEARCH

Recently attempts has been made to address energy consumption issue at architecture layers along with many optimization techniques mostly around hardware level. Software level energy estimation has been overlooked by focusing more on hardware targets and application domains where some simplifying assumptions hold. Most of these approaches revolve around mainly two techniques, either direct measurement of electrical parameters on some target

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

hardware for energy or functional simulation of the processor. In simulation-based methods, energy consumed by software is estimated by calculating the energy consumption of various components in the target processor through simulations at different levels. A register transfer level (RTL) energy simulator called SimplePower is proposed in [1], concentrated on modeling target architectures. A functional unit based power profiler in [2] registers the history of previous states, information about the current states of functional units, and correlated switching capacitance. Cycle-level energy estimation is reported [3], as an extension to [2]. The method estimates the cycle-level energy consumption based on a hierarchical decomposition of the architectural features of the target processor. However, their profiler does not give a framework that can be used to estimate the power consumption of a given sequence of instructions. A cycle-level energy consumption using a measurement at developed hardware is proposed in [4]. It is shown that the energy consumption of software is dependent on the properties of instructions, such as register numbers, immediate operands, etc. A gate-level analysis tool is used to analyze the effect of sequential execution of different instructions in [5]. In their approach, the inter-instruction energy effect is modeled by additional energy consumption observed when each instruction is executed after a NOP instruction. A common drawback of these simulation-based energy models is that they do not provide a mechanism that can calculate the energy consumption of software directly from the instruction sequence. Numerous techniques have been discussed in [8] to explore the impact of source code transformations on families of hardware architectures [6]. They used instruction-level simulation to measure the effects of code transformation on energy [7, 6]. On the other hand, considering processor as the most energy-critical system component, other approaches [8] 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 [9] applied extensive loop transformations to improve locality and hence reduction in data accesses.

Another fine-grained approach is measurement-based where software is characterized by examining the energy consumption obtained from real hardware. Results obtained in such arrangement are closer to the actual energy behavior of the processor, because the information is acquired from the hardware itself through measurements. A current measurement based technique is used in [10]. In this approach, the energy model is given by a power cost table that records 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. Attention has also been given to exploring architecture-level models to be used with higher level tools or in a simulation environment. Microprocessors [7, 12], controllers [13], instruction registers, memory units, are prominent contributor to power dissipation. Researchers have tried to schedule operations [13], or swap operands [10] 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 [11]. Only a few of these researchers have verified these values as actual physical savings in energy [11,12]. An instantaneous power measurement model is presented in [15]. There, a software energy [6] estimation model is proposed by measuring electrical parameters on a digitizing oscilloscope. Some researcher [16,18,19] tried to model the complex energy behavior of processors in terms of usage of their various functional units, mainly targeted for VLIW. In contrast to above approaches [16,17] 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.

3. TRANSFORMATION FRAMEWORK

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.

3.1 StS-TRANSFORMATION ALGORITHM AND COST MODEL

The methodology framework is shown in Figure 3.2. The source code is processed successively for static code analysis, post compiler analysis and finally for scheduling analysis. A VLIW processor descriptor file (VDF) is used to provide architecture information to compiler, scheduler and finally to the machine code generator. The VDF file contains list of pseudo and machine operations, latency of the operations, operating frequency, instruction cache feature (associativity, block size, number of sets) and main memory features (size, order, read/write latencies). Intermediate trace files are generated during code processing flow (i.e. pre/post compiler, scheduler etc..) to produce code size, execution time number of cache miss (for both instruction and data caches), data cache conflicts, data bank alignment, highway usage, scheduling factor, and slot utilization. These parameter are formerly define as follows:

$$\begin{aligned} x_j^\theta &= \{\text{freq}, \text{instructionCacheFeatures}, \text{dataCacheFeatures}, \text{mainMemoryFeatures}\} \\ x_k^\theta &= \{\text{codeSize}, \text{max UnrollingFactor}, \text{max GraftingDepth}, \text{cacheLineSize}\} \\ x_p^\theta &= \{\text{executionTime}, \text{Energy}, \text{slotUtilization}, \text{schedulingFactor}\} \\ x_m^\theta &= \{\text{decisionTreeGrafting}, \text{loopUnrolling}, \text{loopBlocking}\} \end{aligned}$$

where $(x_j^\theta, x_k^\theta, x_p^\theta)$ are static and runtime parameters and x_m^θ are transformation operation.

Energy is measured at the target platform (the setup is explained in Section 3.4). 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, loop tiling. 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 3.3) changed and verified through simulation.

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

$$s((x_j^\theta, x_k^\theta, x_p^\theta), x_m^\theta)$$

We obtain x_j^θ from processor datasheet [14], x_k^θ is acquired after the pre-compiler and profiler stage in Figure 3.1. whereas x_p^θ is an outcome of the simulation at the target hardware. For an idct example these measured values are shown in Table. 1. The parameter x_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:

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

where z_i^0 = instruction cycle count, q_i^0 = number of instructions in block B_j^0 , η_{ij}^0 and ρ_{ij}^0 number of instruction and data cache stall cycles respectively. 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:

$$\tau_p(\theta^p) = \varphi_p \tau_0(\theta^0), \quad \text{and for optimization} \\ \tau_p(\theta^p) < \tau_0(\theta^0).$$

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

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

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:

$$\varepsilon_p(\theta^p) = \psi_p \varepsilon_0(\theta^0), \quad \text{and for optimization} \\ \varepsilon_p(\theta^p) < \varepsilon_0(\theta^0).$$

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:

$$r = \{sourceSize, executionCycle, Energy, cacheMiss, slotUsage\}$$

Figure 3.1: Transformation methodology

Each individual tuning parameter in r can have a value from $\{1,0,-1\}$. We shall discuss more formation of the 5-tuple rule in Section 3.3. Now we shall formulate steering factors that control transformations in x_m^θ .

3.2 TRANSFORMATION COST ESTIMATORS

In this section, we describe the cost estimators of the transformation techniques which determines when to cease iterations in the transformation engine shown in Fig. 3.1. We consider the following transformation techniques: loop tiling, loop unrolling and decision tree grafting.

Block algorithms have better foreground memory such as register and cache reuse, it also gives better cache locality. We use data and computation diagrams, a rectangular parallelepipeds that shows the iteration space of an algorithm with the operations inside and the data on the faces. A typical threefold nested counting loop (ijk-loop) is shown in Fig. 3.2.

We use two performance metrics closely related to each other. 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. The tile size is chosen using block algorithms as proposed in [17].

Figure 3.2: Loop tiling- an ijk loop example ; original loop (left), blocking for i loop (right)

Loop unrolling is performed to exploit the degree of parallelism available in architecture and is controlled by a preset unrolling factor K . We compute K in each successive transformation based on measured parameter during profiling. Thus, to find an optimal K , typically iterative simulations are required with the change of the unrolling factor and hence finding an optimum is a complex and time consuming task. 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:

$$K_i = (\text{loopSize.VLIWinstSize}) / (\text{cacheBlockSize.associativity.noOfSets})$$

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. Decision tree grafting replaces a particular exit of a decision tree with a copy of the destination decision tree. This eliminates a potential branch operation and increases the available parallelism of the current decision tree, at the cost of increased code size.

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 v . For a decision tree block j , grafting depth is formulated as:

$$\varsigma_j = \Omega_j \vee_j \vartheta_j$$

Based on cache size we decide the maximum depth factor $\varsigma_{\max}$. $\varsigma_{\max}$ 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 $\varsigma_{\max}$.

3.3 EXAMPLE - METHODOLOGY FLOW

For a typical idct example we obtain an initial measurement after simulating the baseline code once. This provides source code size, execution time, both instruction and data cache miss rate, data bank conflicts, scheduling factor and energy.

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. E.g. if measured energy is higher then the energy

constraints 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.

3.4 EXPERIMENTAL SETUP

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. Application current consumption is recorded by measuring differential voltage drop across current sensing resistor at the processor input. The energy consumption is computed by multiplying the execution time by the average power consumption measured.

The experiment was conducted with a PNX1301 evaluation board, its processor was configured to run at 180MHz while the SDRAM (32Mbyte) was running at 120MHz. Although this processor supports a block mode power management scheme, it was not activated throughout the experimentation.

4. RESULTS AND DISCUSSION

In line with the proposed methodology described above, we measured static and architecture driven application parameters in different profiling stages enlisted in Table 1. Note that our focus in this paper is on optimizing the application execution time and dissipated energy both.

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 idct example. Each transformation iteration (iter-1 to iter-5) shown in the table corresponds to relative improvement to the same statistics obtained in the baseline version of idct. 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, that is ration between the mapping at available functional units (mentioned in VDF file) to the infinite functional units (and ideal machine). This gives us

better cycle improvement upto 87% (shown as executionTime) and lower energy consumption to 34%.

Functional unit utilization exploits parallelism and is shown here as slot utilization. While each loop tiling has increased misses in both cache, also data bank alignment was disturbed.

Table 1. Successive transformations for IDCT example.

Relative Measures	Iter-1	Iter-2	Iter-3	Iter-4	Iter-5
Code Size	15%	23%	50%	67%	54%
Execution Time	10%	18%	18%	54%	87%
Highway Usage	16%	110%	179%	201%	326%
Scheduling Factor	1%	1%	5%	16%	45%
Slot Utilization	33%	37%	41%	27%	49%
Internal Register Used	9%	41%	53%	45%	67%
Energy	12%	19%	11%	24%	34%
Instruction Cache Miss	10%	10%	21%	17%	19%
Data Cache Miss	2%	1%	2%	3%	6%
Data Cache Conflict	0%	-67%	-98%	-180%	-110%
Data Bank Alignment	-66%	-140%	-286%	-320%	-300%

Moreover an implicit improvement to energy can be observed due to the impact of decision block grafting made on scheduling factor. An optimal grafting depth (see Section 3.3) can cause the scheduling factor to grow higher with a benefit of better data/address highway usage and slot utilization as shown in iter-4 to iter-5.

Figure 4.1. Energy/cycle cost factor for Medibench Applications.

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.

Figure 4.1 illustrates many interesting conclusions. 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. 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).

Table 2. Energy-cycle cost factor for optimized applications.

App./ Transf.	fir	iir	dct	idct	nlivq	m100
Size	0.76	0.38	-0.01	0.54	0.03	1.32
$\phi(c.f)$	0.10	0.33	0.45	0.02	0.01	0.26
$\psi(c.f)$	0.03	0.07	0.17	0.16	0.10	0.15

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

5. CONCLUSIONS

An energy-cycle performance driven source-to-source transformation framework is proposed. Successive transformations are steered by a proposed energy-cycle cost model. The proposed methodology facilitates the programmer to be the strategist and 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.

We demonstrate framework for several multimedia benchmarks. Utilizing this framework for source transformation, the cycle performance can be increased by 87% while decreasing energy consumption by a factor 0.34 for idct. The approach is general and results are verified with real power measurements at Philips Nexperia Media Processor. Followed by this work, we expect to be able to model the energy consumption of multi-cycle instructions. In addition, it will allow us to model the stalls due to the flushes by branch instructions and due to various other reasons such

as data dependency and resource conflicts. In the same vein, this result is 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.

6. ACKNOWLEDGMENT

This work is supported by Österreichischer Austauschdienst (ÖAD)-Pakistan scholarship program. We thank to 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.

7. REFERENCES

- [1] W. Ye, N. Vijaykrishnan, M. Kandemir, and 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.
- [2] H. Mehta, R. M. Owens, and M. J. Irwin, "Instruction level power profiling," Proc. of the International Conference on Acoustics, Speech and Signal Processing, 1996.
- [3] R. Y. Chen, M. J. Irwin, and R. S. Bajwa, "An architectural level power estimator," Proc. of the Power-Driven Microarchitecture Workshop, 1998.
- [4] N. Chang, K.-H. Kim, and H. G. Lee, "Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI," Proc. of the International Symposium on Low Power Electronics and Design, pp. 185-190, July 2000.
- [5] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle, "Modeling inter-instruction energy effects in a digital signal processor," Proc. of the Power-Driven Microarchitecture Workshop, June 1998.
- [6] H. Mehta, R. Owens, M. Trwin, R. Chen, and D. Ghosh, "Techniques for Low Energy Software," ISLPED, pp. 72-75, 1987.
- [7] G. Esakkimuthu, N. Vijaykrishnan, M. Kandemir, M. Irwin, "Memory system energy: influence of hardware-software optimizations," ISLPED, pp. 244-246, 2000.
- [8] Y. Li and J. Henkel, "A Framework for Estimating and Minimising Energy Dissipation of Embedded HW/SW Systems," Design Automation Conference, pp. 188-193, 1997.
- [9] F. Cathoor, S. Wuytack, E. De Greef, F. Balaa, L. Nachtergaele, and A. Vandecappelle, "Custom Memory Management Methodology: Exploration of Memory Organization for embedded Multimedia System Design," Kluwer, 1998.
- [10] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A First step towards software power minimization," IEEE Transactions on VLSI Systems, vol. 2(4), pp. 437-445, Dec. 1994.
- [11] M. Lee, V. Tiwari, S. Malik, M. Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software," IEEE Trans on VLSI Design, pp.123-135, March 1997.
- [12] 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.
- [13] C. L. Su, C. Y. Tsui, A. M. Despain, "Saving power in the control path of embedded processors," IEEE Design and Test Comp, pp.24-30, Dec. 1994.
- [14] TM1300 Data Book, Philips Electronic, North America Corporation, pp. 3.1-3.16, Oct 1999.
- [15] J. T. Russell and 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, and 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] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "An instruction-level functionality-based energy estimation model for 32-bits microprocessors," Proc. of the Annual ACM IEEE Design Automation Conference, pp. 346-351, June 2000.
- [19] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, "Instruction-level power estimation for embedded VLIW cores. In Proceedings of the International Conference on Hardware/Software Codesign," pp. 34-38, May 2000.
- [20] V. Tiwari, S. Malik, A. Wolfe, "Compilation techniques for low energy," ISLPED, Oct 1994.
- [21] 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.