# Techniques for Low Energy Software

Huzefa Mehta[1], Robert Michael Owens, Mary Jane Irwin,
Rita Chen, Debashree Ghosh,

Department of Computer Science and Engineering,
The Pennsylvania State University,
University Park, PA 16802.

[1]Equator Technologies,
Cambell, CA 95008.

**Abstract**

The energy consumption of a system depends upon the hardware and software component of a system. Since it is the software which drives the hardware in most systems, decisions taken during software design has significant impact on the energy consumption of the processor. The paper focuses on decreasing energy consumption of a processor using software techniques. A novel compiler technique is proposed which reduces energy consumption by proper register labeling during the compilation phase. The idea behind this technique is to reduce the energy of the processor by reducing the energy of the instruction register (also the instruction data bus) and the register file decoder by encoding the register labels such that the sum of the switching costs between all the register labels in the transition graph is minimized. There is no hardware penalty since this is purely a compiler optimization. Results on benchmarks show that the energy consumption of the DLX processor can be reduced by 9.82% (maximum) and 4.25% (average) (as measured by DLX energy simulator). In addition several compiler techniques such as loop unrolling, software pipelining, recursion elimination and of effects of different algorithms on power and energy consumption are studied. This evaluation methodology is useful for computer architects to evaluate energy improvements of their hardware, compiler writers to evaluate energy of the compiled code and program writers to evaluate energy of data structures and algorithms.

## 1 Introduction

Today's RISC processors rely on compilers to perform key optimizations to improve performance. The ability of compilers to manage these tasks effectively has given computer architects greater flexibility in making architectural design decisions. Battery energy has become a key commodity in recent years with the widespread use of portable computing. A lot of research has been done, focusing on power consumption analysis and finding ideas and solutions to decrease power dissipation in a processor. All levels (circuit and systems) of design now increasingly stress reducing power consumption. A good survey of work done at the technology level, circuit, architecture and system is given in [2]. It has been only recently that work has been initiated at the system (software) level. This is because most effort had been concentrated on building the right hardware models and tools for analysis. Power analysis at the system level requires accurate energy models of the hardware modules used in the design. Power analysis work is done at the architecture or system level Tiwari et al. [15] perform energy analysis of programs by direct simulation using instruction level power models. Mehta et al. [10] perform direct energy simulation and profiling of the program using high level power models of modules. Other analysis of programs include one by Monterio and Devadas [12]. The authors describe an approach to estimate the average power dissipation in sequential logic circuits under user specified input sequences or programs.

A lot of work has been done to build accurate high level power models (instruction or module level). Tiwari et al. [15] build the instruction level power models after the design has been completed using actual current measurements of the processor chip as it executes instruction patterns. Landman et al. [6, 7, 4, 5] build power models of hardware units by using statistical analysis. Mehta et al. [11] build energy models of hardware modules by clustering energy patterns under a given error margin. Benini et al. [1] develop optimal linear power models are obtained with methods of least squares fitting and its generalization to a recursive procedure called tree regression. Most of the work revolves around characterization of total input-output activity with the energy dissipation. These results although valid for a large class of circuits it is not a general result.

High-level power/energy modeling has been a key facilitator for system level design issues. Work has been done by Tiwari et al. [15, 14, 16, 8, 9] to reduce the energy consumption of the processor by doing instruction level optimizations. These optimizations include efficient usage of memory banks, reordering instructions to reduce switching, reduction of memory operands, operand swapping in the Booth multiplier. Su and Despain try to minimize energy by rescheduling the instructions and using Gray code addressing [13].

The register relabeling technique concentrates on optimization of the switching in the instruction registers and the register file decoder by reordering the register labels of the generated code. A sample trace of the code is obtain the transition frequencies between register ids in the instruction register and the decoder. This is used to obtain new labels which minimize the switching cost in the instruction register and the decoder. This optimization can done however on intermediate code also. This work although demonstrated on DLX is directly applicable to compilers on other machines for minimization of power. Energy experiments with other software techniques are demonstrated using the simulator. This evaluation methodology is useful for computer architects to evaluate energy improvements of their hardware, compiler writers to evaluate energy of the compiled code and program writers to evaluate energy of data structures and algorithms.

The rest of the paper is organized as follows. Section 2 discusses the simulation methodology. Section 3 discusses the register labeling technique

which can be used to reduce switching in the instruction register and decoder. In section 4 we demonstrate several other software and compiler techniques. Section 5 concludes with summary and comments on future work.

## 2 Simulation methodology

Each access to a module consumes different amount of energy and so instead of modeling the 'average energy' we model the energy per access of the module or the switched capacitance per access. IRSIM-CAP is used to obtained the switched capacitances from layout netlists. IRSIM-CAP is an enhanced version of IRSIM and has been calibrated to within $10-15\%$ of SPICE [6].

### 2.1 Energy characterization of modules

For the energy simulation/profiling methodology to be successful it is necessary to estimate accurately and quickly the energy dissipation of a module per input transition. Each of the modules are generated using the Octtools suite with 1.2u technology. The energy tables are generated using IRSIM-CAP. The following are the different types of characterization.

- Bit independent: Modules such as logic operations in the ALU and bit shifters are *bit-independent* since the operation does not depend on the values not in the same bit-slice. The total sum of the energy for the components can be summed up by adding the energy dissipation by individual bits.
- Bit dependent: Arithmetic operations in the ALU, decoders, multipliers, adders, multiplexers are *bit-dependent* since each operation depends upon the bit values in other bit-slices. Clustering is used for the transition vectors for each of these modules and an energy table is obtained. The compressed energy tables generated by clustering algorithms presented in [11] is used for the simulator.

### 2.2 Simulator engine

The approach adopted for this research is to apply empirical methods to low power software and hardware research. This entails a series of experiments in which a set of chosen applications can be used to expose the high energy consumers of the architecture and vice-versa. By studying the breakdown of energy consumption in different components, algorithms can exploit or get around certain features of an architecture. The base architecture in this case is chosen to be DLX. Although we have fixed the basic machine, it may also be feasible to find an appropriate architectural change to improve the energy. The accuracy of estimation by estimating the energy of each micro-instruction is about 8% when compared to IRSIM-CAP [10]. Each instruction pipeline stage of DLX is modeled and depending on each activity the control steps are activated for that state. The DLX simulator core was obtained from Stanford [3]. This is an interactive simulator which loads DLX assembly programs and simulates the operation of a DLX computer on those programs. The main simulation loop of the simulator has been instrumented such that the energy expenditure of the micro-operations of each instruction can be accounted for.

## 3 Register relabeling

Naive register labeling can incur significant bit changes in consecutive source and destination fields of the instruction words. Register switching is directly proportional to the number of bits switched in the register. Register labeling is a phase where variables are assigned their register ids during the register graph coloring of register allocation. This process is simple and allocates the first available unique id (or may depend upon the peculiarities of the implementation). With the view of the emphasis towards low power/energy it is necessary to assign register labels which minimizes switching in the instruction register. To provide motivation we present a kernel used in SAXPY in figure 1, which shows the initial unoptimized labeling and the final optimized labeling. The optimized labeling has a total improvement of 15.60% where the variable mapping of $z[i]$ and $y[i]$ are swapped. This kernel is a tight loop executing several times and hence optimizing these loops could result in significant reduction in switching in the registers.

Figure 1: SAXPY kernel (z[i] = a*x[i] + y[i];)

| Unoptimized | | | | Optimized | | | |
|---|---|---|---|---|---|---|---|
| MULTF | F0 | F1 | F2 | MULTF | F0 | F1 | F2 |
| ADDF | F3 | F0 | F4 | ADDF | F4 | F0 | F3 |
| | | | | Reg improv=15.54% | | | |
| | | | | Dec improv=21.71% | | | |
| | | | | Total improv=15.60% | | | |

### 3.1 Switching cost calculation

We describe in detail our instruction register energy model and the register file decoder model since this information is used as switching cost in the compiler.

- **Register energy model:**
  Significant energy is consumed in the instruction register and the instruction data bus. During the execution of a program the instruction register undergoes changes in state due to loading of new instruction words. Instruction words which have fixed source and destination fields can be minimized for bit switching. When the memory access paths (data and instruction) are separate, the switching on the instruction bus is also reduced when the switching in the instruction word is reduced. Although minimizing the switching inside the registers, decreases the switched capacitance, it is interesting to note that even a 0 to 0 transition consumes substantial energy. This is because of the switching of the clock transistors. This can be taken care of by 'guarding' the clock at the inputs of the registers.
- **Decoder energy model:**
  The register file decode logic which drives the word lines in the register file consumes significant energy. Modeling energy transitions in the decode logic is tricky since 'less bit switching' at the input does not imply 'less switching energy'. The energy transition in the decode logic is characterized for certain switching patterns by clustering technique shown in [11].
- **Putting it all together:**
  This switching cost function represents the energy consumption in the instruction registers and decoders for a specific labeling. This includes all switching in the instruction register and decoder due to changes in encoding.

### 3.2 Encoding algorithm

The easiest option to encode is to assign an initial labeling and then exchange labels if the cost function (total switching) of the graph decreases. The problem with this approach is that there are ${}^nC_2$ choices for each swap. Exploring all the possible choices is combinatorially explosive problem. The other option is to find disjoint edges with maximum costs (maximum weight matching problem). These disjoint edges can then 'greedily' be coded using swaps and the problem reiterated till no swaps can be performed. Our approach is to greedily carry out 'maximum cost decrease' exchanges till no further exchanges are possible. The algorithm performs the most greedy swap at every step. The reason why a greedy algorithm is chosen it gives sufficiently good results in a very quick time. Also, the intent of this work is to show the applicability of the method and result (not the optimization algorithm). For better mapping a more sophisticated optimization algorithm can be used.

A register transition graph is built from this trace of the registers. The graph is populated with unused labels. This adds additional nodes to the graph, however this step ensures that all possible labels of width $log(n)$ are explored. Then a new graph is created with edge weights equal to switching cost of the edge. The pair of nodes are selected whose encodings when swapped result in the maximum decrease in switching cost. The identified nodes are swapped if the switching cost reduces and relevant other weights in the graph are updated. This continues till there is no additional reduction in total switching. Constructing a graph with modified weights takes $O(n^2)$ time. Hence the total time for each swap iteration is $O(n^3)$.

Table 1: Results of relabeling optimization on DLX (number of registers = 32)

| Code | Time (secs) | Optimization % improvement | | |
|---|---|---|---|---|
| | | Optimizer | DLX energy simulation | |
| | | reg (15 bits) +dec | reg (32 bits) +dec | total |
| best | 1.1 | 23.54 | 21.53 | 9.82 |
| average | 3.42 | 18.15 | 13.26 | 4.25 |

### 3.3 Implementation and results

A register trace (extracted from the instruction trace) is generated dlxsim by running the program through sample data. This is necessary since in control dominated programs the instruction trace is strongly dependent on data, and hence it is not possible to know the exact instruction sequences just by looking at the program. Loop variables in certain applications (signal and image processing) are known and hence only in these applications can the instruction trace be extracted statically. This register trace lists the source and the destination register operands as executed by the program. Several benchmark kernels are evaluated for optimized energy. These kernels are excerpts from large FORTRAN programs that have been judged to provide a good measure of large scale computer performance. These kernels are recoded in C and are run independently. None of the benchmarks required more than nine iterations to converge to a locally optimum labeling. The total energy reduction of the processor is hence 9.82% maximum to 4.25% average. The algorithm takes $O(n^3)$ per swap. All the benchmarks take less than 5.5 CPU secs and 3.42 CPU secs on average.

## 4 Other compiler/software techniques

In the following subsections we investigate the effects of different compilation techniques and algorithm and benchmarks on power and energy consumption. Some compiler optimizations which are done are loop unrolling and software pipelining to increase instruction level parallelism. We investigate their effects with respect to power/energy consumption.

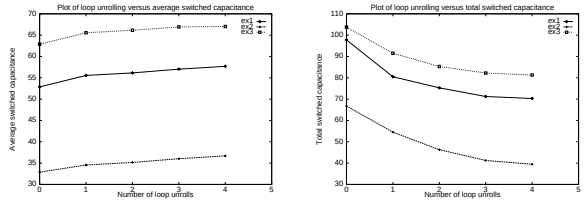
- Loop unrolling: Loop unrolling is one of the techniques to increase instruction level parallelism by decreasing the number of control statements which execute in one loop by creating longer sequences of straightline code. Energy of the loop is proportional to $a + \frac{b}{(n+1)}$ where $a$ and $b$ are constants (corresponding to energy of straightline and control code) and $n$ is the loop unrolling factor. By increasing the loop unrolling the contribution of the second term decreases and the first term dominates. Figures 2(a)(b) shows the average switching energy per cycle (power) and the total switching energy versus the loop unrolling factor

  *Loop unrolling decreases the number control operations and overhead instructions. The overhead is code space. There is a diminishing return, however from excess unrolling. (Other issues concerning register file size and instruction buffer size have not been evaluated). As the loop unrolling factor increases, the average switching energy per cycle increases while the total energy of the program decreases.*

- Software pipelining: Software pipelining is a technique for re-organizing loops such that each iteration in the software-pipelined loop is formed from instruction sequences chosen from different iterations in the original code segment. The major advantage of software pipelining over straight loop unrolling is that software pipelining consumes less code space. Figures 3(a)(b) shows the original and software pipelined code for some examples.

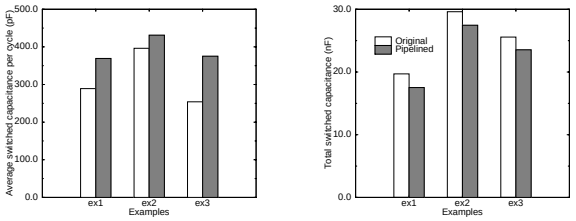
  *Software pipelining decreases the number of stalls by fetching instructions from different iterations. Hence, total energy consumption reduces due to this reduction in stalls and furthermore the program takes fewer cycles to finish (hence average energy per cycle is greater).*

- Eliminating recursion: Compilers usually execute recursive procedures by using a stack that contains pertinent information, including



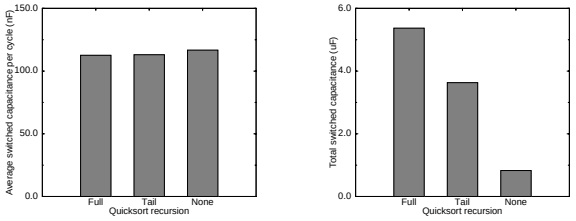

(a) Average switched capacitance ($\propto$ power) (b) Total switched capacitance ($\propto$ energy)

Figure 2: Loop unrolling (100 iterations)



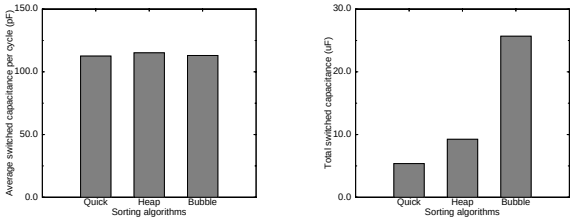

(a) Average switched capacitance ($\propto$ power) (b) Total switched capacitance ($\propto$ energy)

Figure 3: Software pipelining



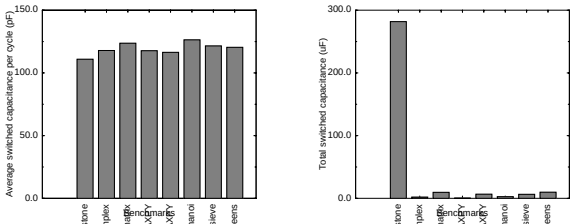

(a) Average switched capacitance ($\propto$ power) (b) Total switched capacitance ($\propto$ energy)

Figure 4: Recursion elimination (quicksort)




(a) Average switched capacitance ($\propto$ power) (b) Total switched capacitance ($\propto$ energy)

Figure 5: Sorting algorithms (100 random integer numbers)




(a) Average switched capacitance ($\propto$ power) (b) Total switched capacitance ($\propto$ energy)

Figure 6: Benchmarks

the parameter values, for each recursive call. Good compilers usually provide a feature called tail recursion in which the recursion occuring at the end (which is not usually necessary) can be eliminated. Figures 4(a)(b) shows energy evaluation with the number of recursion calls for the sorting program quicksort (full, tail and no recursion).

*Recursion elimination saves much of the overhead usually associated with calls (saving register, manipulating displays, pushing activation records and so forth). This is also true for inline expansion of a subprogram. In fact, other optimizations become possible because the actual parameters used in a call become visible in the subprogram body. Especially in the case of actual parameters that are literals, opportunities for folding and deleting of unreachable code can be expected. The total switched capacitance drops significantly as recursion decreases however the switched capacitance per cycle (power) increases because the number of expensive memory and register overhead operations (push, pop) decreases and hence the number of cycles which use the stack decreases.*

- Algorithm evaluation: In a manner similar to profiling for performance, simulation and profiling for energy is important to evaluate algorithms and optimize code writing. Several sorting algorithms were evaluated: Quicksort, heapsort and bubblesort (Figures 5(a)(b)). We observe that quicksort has less total capacitance than heapsort because heapsort uses more pointer arithmetic to manipulate the heap than quicksort, which uses more 'neighboring' operations to increment indexes. Other benchmark algorithms are also run on the simulator (Figures 6(a)(b)).

  *Algorithms can be evaluated by evaluating their energy breakdowns. Analyzing the benchmark algorithms can expose some of the hot spots in the architecture and failings of an architecture.*

## 5 Conclusions

We have identified one important technique (strategy and algorithm) for software level energy optimization. It is important to note that there is no hardware penalty since this is totally a compiler optimization. Other compiler techniques and algorithm and benchmarks and effects on power and energy have been evaluated.

The total energy reduction of the processor is 9.82% maximum to 4.25% average. The algorithm takes $O(n^3)$ per swap and for most of the benchmarks take less than 5.5 CPU secs and 3.42 CPU secs on average.

The optimizer is a post-pass, however this phase can be taken care of on intermediate code. An important thing to note is that we have used a naive greedy algorithm. This is because our aim is to demonstrate significant improvements can be achieved due to this technique. A more sophisticated algorithm can be used to obtain better improvements in energy at the expense of higher running time.

We have experimented with a few compiler optimization techniques such as loop unrolling, software pipelining and recursion elimination and some algorithms and benchmark programs. Direct extensions to this evaluation methodology include:

- Energy characterization of different data structures and algorithms
- Considering energy characterization of caches
- Considering effects of register file and instruction buffer size on loop unrolling

## REFERENCES


[1] Benini L., Bogliolo A., Favalli M., and Micheli G. Regression models for behavioral power estimation. In *International workshop on power and timing modeling, optimization and simulation*, 1996.

[2] Devadas Srinivas and Malika Sharad. Tutorial: A survey of optimization techniques targeting low power. In *32nd Design Automation Conference Proceedings*, pages 242–247, June 1995.

[3] Hostetler Larry B. and Mirtich Brian. Dlxsim: A simulator for DLX.

[4] Landman Paul E. and Rabaey Jan M. Architectural power analysis: The dual-bit type method. In *EDAC-EUROASIC*, pages 370–377, 1993.

[5] Landman Paul E. and Rabaey Jan M. Power estimation for high level synthesis. In *EDAC-EUROASIC*, pages 361–366, 1993.

[6] Landman Paul E. and Rabaey Jan M. Black-box capacitance models for architectural power analysis. In *Proceedings of the International Workshop on Low Power Design*, pages 165–170, April 1994.

[7] Landman Paul E. and Rabaey Jan M. Activity-sensitive architectural power analysis for the control path. In *Proceedings of the International Workshop on Low Power Design*, pages 93–98, April 1995.

[8] Lee Mike and Tiwari Vivek. A memory allocation technique for low-energy embedded DSP software. In *Proceedings of the 1995 Symposium on Low Power Electronics*, 1995.

[9] Lee Mike, Tiwari Vivek, Fujita Masahiro, and Malik Sharad. Power analysis and low-power scheduling techniques for embedded DSP software. In *8th International Symposium System Synthesis*, 1995.

[10] Mehta Huzefa, Owens Robert Michael, and Irwin Mary Jane. Instruction level power profiling. In *International Conference on Acoustics, Speech and Signal Processing*, 1996.

[11] Mehta Huzefa, Owens Robert Michael, and Irwin Mary Jane. Module energy characterization using clustering. In *Proceedings of Design Automation Conference*, June 1996.

[12] Montereiro Jose and Devadas Srinivas. Techniques for the power estimation of sequential logic circuits under user-specified input sequences and programs. In *Proceedings of the International Workshop on Low Power Design*, pages 33–38, April 1995.

[13] Su C., Tsui C., and Despain Alvin. Low power architecture design and compilation techniques for high-performance processors. In *IEEE Symposium on Low Power Electronics*, pages 49–58, March 1994.

[14] Tiwari Vivek, Malik Sharad, and Wolfe Andrew. Compilation techniques for low energy: An overview. In *Proceedings of the 1994 Symposium on Low Power Electronics*, 1994.

[15] Tiwari Vivek, Malik Sharad, and Wolfe Andrew. Power analysis of embedded software: First step towards software power minimization. In *Proc. of the Int'l Conference on Computer Aided Design*, pages 384–390, Nov 1994.

[16] Tiwari Vivek, Pranav Asher, and Malik Sharad. Technology mapping for low power. In *Proceedings of the 30th Design Automation Conference*, pages 74–79, June 1993.