# Compiler Optimization on Instruction Scheduling for Low Power *

Chingren Lee Jenq Kuen Lee TingTing Hwang
Dept. of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan
Shi-Chun Tsai
Dept. of Information Management, National Chi-Nan University, Pu-Li, Nan-Tou, Taiwan

## Abstract

*In this paper, we investigate the compiler transformation techniques to the problem of scheduling VLIW instructions aimed to reduce the power consumption on the instruction bus. It can be categorized into two types: horizontal and vertical scheduling. For the horizontal case, we propose a bipartite-matching scheme. We prove that our greedy algorithm always gives the optimal switching activities of the instruction bus. In the vertical case, we prove that the problem is NP-hard, and propose a heuristic algorithm. Experimental results show average 13% improvements with 4-way issue architecture and average 20% improvement with 8-way issue architecture for power consumptions of instruction bus as compared with conventional list scheduling for an extensive set of benchmarks.*

## 1 Introduction

The push for low power design has recently gained growing importance in designing various computer systems and embedded systems. For that reason, we will study the aspect of compiler transformations to reduce power consumptions for such a system.

In CMOS circuits, power is dissipated in a gate when the gate output changes from 0 to 1 or from 1 to 0. Minimization of power dissipation can be considered at algorithmic, architectural, logic and circuit levels. Studies on low power design are abundant in the literature [1, 2, 3] in which various techniques were proposed to synthesize designs with low transitional activities.

Recently, new research directions in reducing power consumptions have begun to address the issues of arranging software at instruction-level to help reduce power consumptions [4, 5]. The energy reductions through software modifications are essentially free, as no extra hardware cost or overhead is needed. Previous improvements with software re-arrangements include the value locality of registers [4] and the swapping of operands for booth multiplier [5]. This new direction brings an interesting issue in the compiler participation in software re-arrangements for reducing power consumptions for applications and systems. In this paper, we investigate the compiler transformation techniques to the problem of scheduling VLIW instructions aimed to reduce the power consumption of the VLIW architectures on the instruction bus.

The energy, $E$, consumed by a program, is given by $E = P \times T$, where $T$ is the number of execution cycles of the program [5] and $P$ the average power. The average power $P$ is given by $P = \frac{1}{2} \cdot C \cdot Vdd^2 \cdot f \cdot E$, where $C$ is the load capacitance, $Vdd$ the supply voltage, $f$ the clock frequency, and $E$ the transition count. In the compiler optimizations, if we optimize programs for the performances, $T$ will be reduced, so does energy consumption. If the compiler performs software refinements to reduce $P$, without software performance penalty, it will also reduce the energy consumptions. Therefore, it is preferable that any power minimization technique should incur no performance penalty.

Judging from the power equation, it is clear that power can be reduced by the product of capacitance loading and transition activity. Since bus wires have large capacitance loading, the reduction of transition activities of buses will be very effective in reducing total power consumption. Hence, we will study VLIW instruction scheduling techniques aimed at the reduction of transition activity in the **instruction bus**. We will first present a cost model to estimate the bus switching activities of instruction executions on VLIW architectures. Based on the model, we further develop instruction scheduling techniques to reduce the power consumption in the bus level.

*The correspondence author is Jenq Kuen Lee. His e-mail is jklee@cs.nthu.edu.tw, phone number is 886-3-5715131 EXT. 3519, FAX number is 886-3-5723694. This work was supported in part by NSC of Taiwan under grant no. NSC-88-2219-E-007-006, NSC-89-2219-E-007-001 and NSC-89-2215-E-007-022.

The problem can be divided into horizontal scheduling and vertical scheduling categories. For horizontal case, we propose a bipartite-matching scheme for instruction scheduling. We prove at that our greedy algorithm always gives the optimal bus switching activities for given VLIW instruction scheduling policies. For vertical case, we prove that the problem is NP-hard and propose a heuristic algorithm to solve it.

The remainder of the paper is organized as follows. Section 2 describes our experimental VLIW platform, and cost model for power consumption in the bus level. Section 3 proposes the policies for low power VLIW code generation. Section 4 gives our experimental results. Finally, section 5 concludes this paper.

# 2 Machine Architecture and Cost Model

## 2.1 Machine Architecture

Figure 1 shows an example of our target machine architecture on which our optimization is based. We focus on the reduction of switching activities for instruction bus. The abstract VLIW machine has several execution units. In the example given in Figure 1, unit 1, 2, 3 are integer ALUs with integer multiplication, division, and logic operation unit. Unit 4 is either the same as other unit, or will be performing branch/flow control, or executing a load/store function. In this architecture, a VLIW instruction can only issue one load/store (or branch/flow control) microinstruction and three integer/logic microinstructions at the same time. Also, it can perform four integer/logic microinstructions only. without load/store or branch microinstructions.

Figure 1 is also the architecture model by which we carry out our experiments later in Section 4. The length of an VLIW instruction in our experiment will be 128 bits. Memory addressing is byte address. Also, we use real executable instructions of Alpha chip for experiments. As Alpha is a 64 bit CPU and uses 64 bit data bus, our experimental VLIW machine will have 32 integer registers (R0 through R31), and each is 64 bits wide. We assume this VLIW machine assign an 128 bit instruction into 4 function units per instruction fetch. The instruction bus is 128-bit wide.

## 2.2 Cost Model

We will use *hamming distance* as our cost model to estimate the transition activity in the instruction

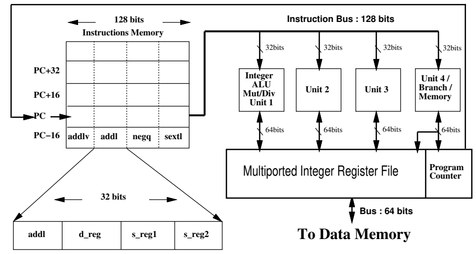


Figure 1: Our VLIW Machine Architectures and Bus Models

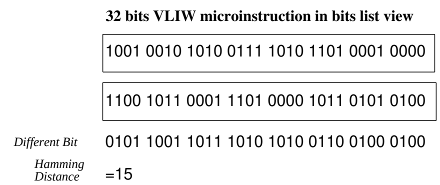


Figure 2: Hamming Distance

bus. Hamming distance is the number of bit differences between two binary strings. For example, the hamming distance of two adjacent 32-bit microinstructions shown in Figure 2 is 15.

Suppose $X$ and $Y$ are two consecutive VLIW instructions with $k$ way issues. The instruction components of $X$ and $Y$ are $(x_1, x_2, ..., x_k)$ and $(y_1, y_2, ..., y_k)$, respectively. Then, the bus transition cost, $H(X, Y)$, for instruction $Y$ after the issue of $X$ is defined as,

$$H(X, Y) = \Sigma_{i=1}^{k} \; h(x_i, y_i),$$

where $h$ is the hamming distance between two instruction components.

# 3 Instruction-Scheduling Policies for Low Power

Both high performance and low power are two important objectives of compiler optimization. However, since degradation of performance has negative effect not only on performance but also energy consumption, we require that any power minimization technique incurs no software performance penalty. Therefore, we propose a two-phase instruction scheduling approach. In the first phase, instructions are scheduled for performance. Then, in the second phase, a scheduler is employed to re-arrange the codes produced by the first phase for low power without incurring performance penalty.

In this work, list scheduling algorithm will be used in the first phase for performance optimization. List scheduling programs are easy to write, and can compact original microinstructions approximately as fast as linear analysis. However, any conventional VLIW instruction scheduler can be used.

## 3.1 Horizontal Scheduling

We first propose a horizontal scheduling algorithm to re-schedule instruction components of an instruction to minimize transition activity of instruction buses, i.e., microinstructions of an instruction are to be re-scheduled for different instruction buses but executed in the same instruction. It is formally defined as follows.

We focus on the basic block of a program. Suppose in the basic block of a program we have $n$ VLIW instructions, and they are $X_1, X_2, ..., X_n$, where the instruction components of instruction $X_i$ is given as

$$X_i = (x_{i,1}, x_{i,2}, ..., x_{i,k}), 1 \leq i \leq n$$

Then, our goal is to find an instruction scheduling so that the total hamming distance of $n$ consecutive VLIW instructions $X_1^{'}, X_2^{'}, ..., X_n^{'}$ is minimized. That is, we want to minimize

$$\Sigma_{j=1}^{n-1} \ H(X_j^{'}, X_{j+1}^{'}),$$

where $X_i^{'} = (\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,k})$, $1 \leq i \leq n$, and $(\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,k})$ is a permutation of $(x_{i,1}, x_{i,2}, ..., x_{i,k})$, for $1 \leq i \leq n$.

Our horizontal scheduling algorithm proceeds to re-schedule microinstructions from the first instruction to the last. Initially, the first instruction is re-scheduled without changing it. Iteratively, the next instruction is re-scheduled to minimize hamming distance between the instruction and the last instruction already scheduled. We model the re-scheduling of microinstructions of an instruction as a weighted bipartite graph matching. Let the bipartite graph $G = (UpLayer \cup LowLayer, E)$ be constructed, where $UpLayer$ and $LowLayer$ are bipartite. Each $u_i \in UpLayer$ represents a microinstructions in the last instruction already scheduled and $l_i \in LowLayer$ represents a microinstruction of an instruction to be scheduled. There is an edge linking $u_i$ and $l_i$ if microinstruction $l_i$ can be assigned to the same bus as the microinstruction $u_i$. The weight on the edge is defined as $-h(u_i, l_i)$, the hamming distance of $u_i$ and $l_i$. Note that the negative sign is used because a maximum weighted matching will be taken. Figure 3 illustrates the construction of a bipartite graph. In

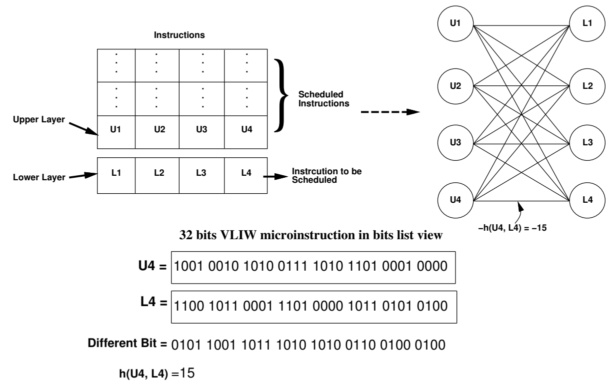


Figure 3: An example of bipartite matching for horizontal scheduling.

this figure, there are four microinstructions in each instruction. Instruction represented by UpperLayer is the last instruction already re-scheduled. Instruction represented by LowLayer is the instruction to be re-scheduled. The weight on the edge of $u_i$ and $l_i$ is $-h(u_i, l_i)$.

Now, we apply maximum weight bipartite matching [6] on graph. If microinstruction $l_i$ is matched with microinstruction $u_i$, $l_i$ will be assigned to the same bus as $l_i$. The matching procedure repeat till all instructions are re-scheduled. This matching procedure re-schedules microinstructions so that the hamming distance between the instruction of $UpLayer$ and the instruction of $LowLayer$ is minimum. This greedy bipartite matching algorithm can actually produce optimal solution for our horizontal scheduling problem. It can be proved by induction [7].

This algorithm presents the essential idea of our low power optimization. It requires function units of target VLIW architectures be identical, as we can only perform microinstruction swapping with identical function units on target host so that there will always be no performance penalty. In the practice of modern VLIW architectures, our algorithm needs to be refined to work with architecture constraints, as function units are normally classified into several classes in most VLIW architecture design. The swapping can only be done with function units of the same class. This constraint can be implemented by constructing bipartite edges only among microinstructions using functions units in the same class. In addition, this algorithm will interact the data bus's activities while CPU can issue more than 1 load/store microinstructions. Its influence on instruction cache is depended on the caching policies.

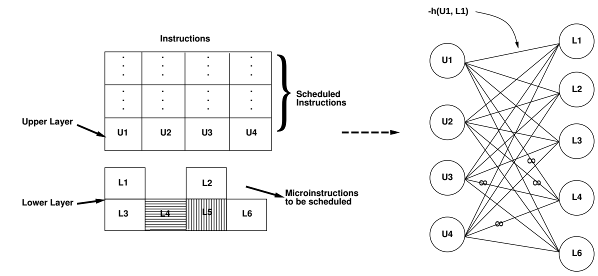


Figure 4: An example of bipartite matching for vertical scheduling.

### 3.2 Vertical Scheduling

While the discussions in the previous sub-section are based on instruction scheduling with horizontal movements, our vertical scheduling allows microinstructions to move across instructions. Since we want to optimize power consumption without degrading performance, cares have to be taken so that no performance penalty are incurred. Vertical scheduling is formally defined as follows.

Suppose we are given a performance-optimized VLIW scheduling, $X$, data dependence graph, $DDG$, for a basic block, $B$, and the total cycle time, $T$, for the execution of $X$. Let the given scheduling $X$ be $< X_1, X_2, ..., X_n >$ and the instruction components of instruction $X_i$ are given below.

$$X_i = (x_{i,1}, x_{i,2}, ..., x_{i,k}), 1 \leq i \leq n.$$

The vertical scheduling is to find a scheduling, $Y$, where $Y$ is $< Y_1, Y_2, ..., Y_n >$, and the instruction components of instruction $Y_i$ are given below,

$$Y_i = (y_{i,1}, y_{i,2}, ..., y_{i,k}), 1 \leq i \leq n.$$

The vertical scheduling, $Y$, needs to satisfy the following constraints.

1. There exists a bijection $\delta : A \rightarrow B$, where $A = \bigcup_{i=1}^{n} \{x_{i,j} | x_{i,j} \in X_i\}$, and $B = \bigcup_{i=1}^{n} \{y_{i,j} | y_{i,j} \in Y_i\}$.

2. The cycle time of Y is less than or equal to $T$

We then try to find the scheduling Y for the minimum bus transition activities, so that $\Sigma_{j=1}^{n-1} H(Y_j, Y_{j+1})$ is minimized. This problem to find the minimum scheduling $Y$ for the vertical case is NP-Hard. The detailed proof is related to "Hamiltonian Path Problem"can be find here [7]. Since the problem is NP-hard, we will propose a heuristic algorithm based on allowable moving windows of instruction sets and bipartite matching techniques.

Our algorithm is given an initial instruction placement $X$ as input. The initial placement can be obtained from conventional instruction scheduling for performance optimization. Our heuristic algorithm is to find a new scheduling $Y$ so that bus transition activities are minimized as much as possible. Other given inputs to our algorithm for the re-scheduling of instructions of a basic block are data dependence graph ($DDG$) and critical path information. The data dependence graph specifies the execution order among the components of given instructions to obey the original program semantics. The critical path information specifies how far an instruction component can be placed without incurring software performance penalty.

Our vertical scheduling algorithm proceeds to reschedule microinstructions from the first instruction to the last. First, a window size, $w$, needs to be specified, which defines the number of instructions that are allowed to be moved in each iteration. Because this problem is NP-Hard, a larger $w$ may cause unacceptable compilation time. In fact, horizontal scheduling is a special case of vertical scheduling with $w = 1$. Initially, the first instruction is re-scheduled without changing it. Iteratively, the next $w$ instructions are candidates to be selected and to be re-scheduled to minimize hamming distance. The microinstructions in the next $w$ instructions have to satisfy the data dependence constraint and critical path constraint. To satisfy data dependence constraint, it is required that if the parents of a microinstruction have not been assigned, the microinstruction should be deleted from the window. To satisfy critical path constraint, it is required that microinstructions that are on the critical path should be only re-scheduled by horizontal move.

We also model the vertical re-scheduling of microinstructions as a weighted bipartite matching. A bipartite graph $G = (UpLayer \cup LowLayerSet, E)$ is constructed, where $UpLayer$ and $LowLayerSet$ are bipartite and $LowLayerSet$ represents the microinstructions in the next $w$ instructions that satisfy data dependence constraint. Each $u_i \in UpLayer$ represents a microinstructions in the last instruction already scheduled and $l_i \in LowLayerSet$ represents a microinstruction to be scheduled. There is an edge linking $u_i$ and $l_i$ if microinstruction $l_i$ can be assigned to the same bus as the microinstruction $u_i$.

There are two ways to define the weights on edges. The first way is that if $l_i$ is on the critical path, the weights on all edges linking $l_i$ are defined as $\infty$, which guarantees that $l_i$ will be selected in the matching. Otherwise, the second way is that the weight on an edge linking $l_i$ and $u_i$ is defined as $-h(u_i, l_i)$, the hamming distance of $u_i$ and $l_i$. Figure 4 illustrates the construction of a bipartite graph for vertical scheduling. In this example, the window size is 2, $l_4$ is on the critical path and $l_2$ is the parent of $l_5$ in the data dependence graph. Therefore, edges linking $l_4$ are set to $\infty$ and $l_5$ is deleted from the window.

# 4 Experiments

We use the Alpha-based VLIW architecture described in Figure 1 of Section 2 as the target architecture for our experiments. The proposed scheduling policy is incorporated into the compiler tool with SUIF [8] and MachSUIF Library [9]. Figure 5 shows the three phases of compilations in incorporating our algorithms into SUIF and MachSUIF systems. We perform general compiler behaviors in first and second phases. Finally, we work on the optimizations with low-power issues. In this phase, we load the almost-executable outputs from MachSUIF library, and perform list scheduling to get VLIW instruction sequences. We then execute bipartite matching algorithm to schedule instructions to reduce the power consumption of VLIW architectures in the instruction bus. The Alpha assembly code (.s code) generated by our software is annotated with additional information for VLIW instructions so that the ATOM simulator can pick up the VLIW instruction information.

Figure 6 gives the experimental result for the simulations on a 4-way issue architecture described in Section 2. The line with the black color is the base information which is the switching activities of instruction bus for programs scheduled by list scheduling. It's used as the base line. The line with the white color is the switching activities of instruction bus for programs scheduled by our proposed multi-layer bipartite-matching scheme with horizontal scheduling. The improvement (reduction) with switching activities ranges from 3% to 19% among test suites, and with the average of 13%. The test suites in the experiment include three parts. The first five suites of Figure 6 are from the common benchmarks listed in FAQ of comp.benchmarks [10]. The next suite is the grep utility routine taken from GNU Grep v2.2. The rest of the 22 test suites in Figure 6 are from GNU TextUtils v1.22. The line with gray color in the figure is the worst case you can do with switching activities of instruction bus for given programs. As the list scheduling does not concern about the power consumption of

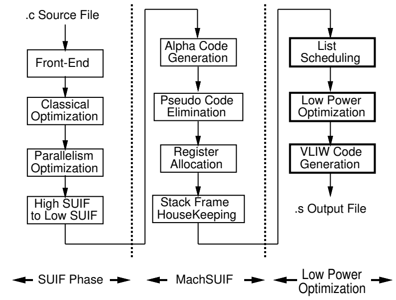


Figure 5: Our compiler phases on low power optimizations

the program, the worst case gives a reference point for our experiments. We are also in the process of evaluating more test suites for experiments.

Focus is now directed to Figure 7 which gives the experimental result for the simulations on an architecture similar to the previous experiment but with 8-way issues. Again, the black color line is the base information which is the switching activities of instruction bus for programs scheduled by list scheduling. The white and gray color lines are representing the results with multi-layer bipartite-matching scheme(horizontal scheduling) and the worst case, respectively. The average improvement (reduction) with switching activities is around 20% over the list scheduling by incorporating our schemes.

Figure 8 is the vertical scheduling experimental results for 4-way issue architecture. The line with gray color is the best case in 4-way issue horizontal scheduling. The black line is the vertical scheduling with $w = 4$. The white line is the vertical scheduling with $w = 8$. The enhancement from horizontal scheduling to vertical scheduling is around 6.0% to 9.5%.

# 5 Conclusion

In this paper, we first described a model for calculating the bus switching activities of instruction executions on VLIW architectures. Based on the model, we investigate the compiler transformation techniques to schedule VLIW instructions aiming to reduce the power consumption of VLIW architectures in the instruction bus. Our experiment is done on Alpha-based VLIW architectures and ATOM simulator. Our compiler is implemented based on SUIF and MachSUIF, and by incorporating our proposed schemes. Experimental results show significant improvements with power consumptions over conventional list scheduling

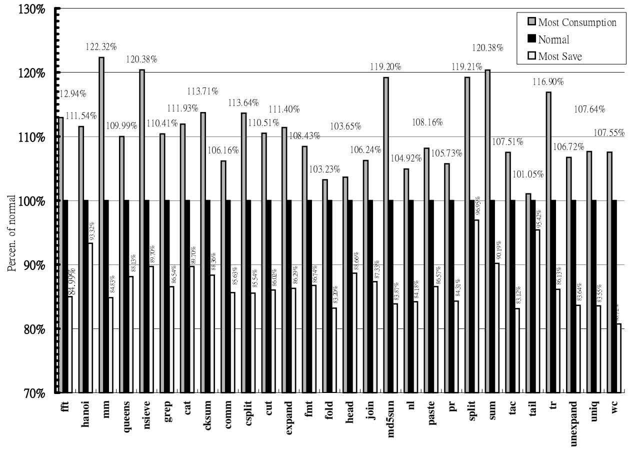


Figure 6: Horizontal Scheduling Switching Activities with 4-way Issues

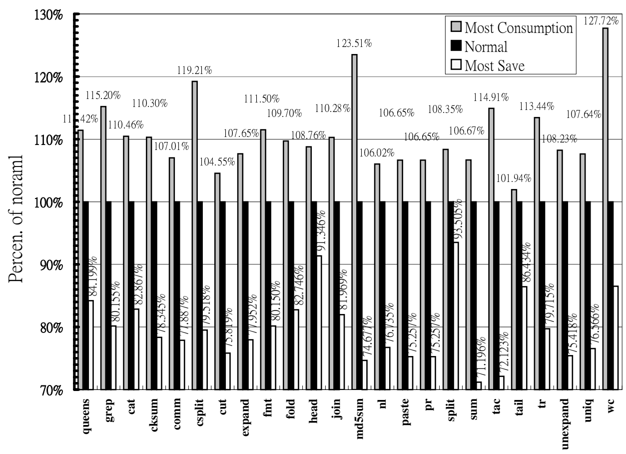


Figure 7: Horizontal Scheduling Switching Activities with 8-way Issues

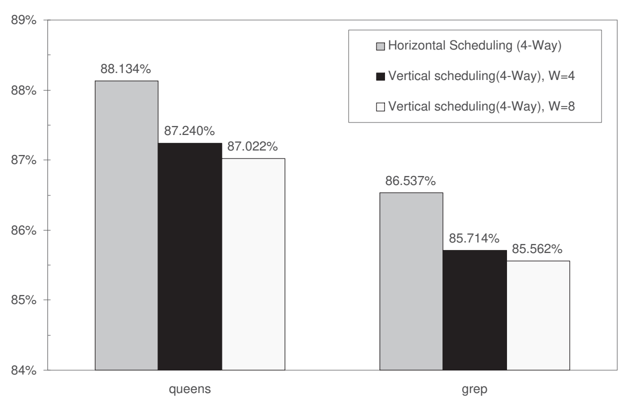


Figure 8: Vertical Scheduling Switching Activities with 4-way Issues

for an extensive set of benchmarks by incorporating our proposed schemes. We think our work is important for a class of systems, high-performance embedded systems, where we need to address both the high-performance computing and power consumption issues.

# References

[1] C.Y. Tsui, and M. Pedram, and A.M. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation", *Proc. of 30th Design Automaton Conf.*, pp.68-73, June 1993.

[2] M. Alidina, and J. Monteiro, and S. Devadas, and A. Ghosh, and M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power", *Proc. of ICCAD-94*, pp. 74-81, 1994.

[3] Inki Hong, and Darko Dirovski, et.al., "Power Optimization of Variable Voltage Core-Based Systems", *Proc. of 35th DAC*, pp. 176-181, 1998.

[4] Jui-Ming Chang, Massoud Pedram. "Register Allocation and Binding for Low Power", *Proceedings of Design Automaton Conference*, San Francisco, USA, June 1995.

[5] Mike Tien-Chien Lee, and Vivek Tiwari, and Sharad Malik, and Masahiro Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software", *IEEE Transactions on VLSI Systems*, Vol. 5, no. 1, pp. 123-133, March 1997.

[6] Michael L. Fredman, and Robert Endre Tarjan, "Fibonacci Heap and Their Uses in Improved Network Optimization Algorithms", *Journal of the Association for Computing Machinery*, Vol. 34, No. 3, Page 596-615, July 1987.

[7] Ching-ren Lee, "Compiler Optimization on Advacned Processors for Low Power", *Master Thesis*, Dept. of Computer Sciense, National Tsing Hua Univ, Taiwan, 1999.

[8] Stanford Compiler Group, *The SUIF Library*, Stanford Compiler Group, Stanford, March 1995.

[9] Michael D. Smith, *The SUIF Machine Library*, Division of of Engineering and Applied Science, Harvard University, March 1998.

[10] Al Aburto, *collections of common benchmarks of FAQ of comp.benchmarks USENET newsgroup*, ftp site: ftp.nosc.mil/pub/aburto.