Compiler Optimization on VLIW Instruction Scheduling for Low Power

CHINGREN LEE, JENQ KUEN LEE, and TINGTING HWANG
National Tsing-Hua University, Taiwan
and
SHI-CHUN TSAI
National Chi-Nan University, Taiwan

In this article, we investigate compiler transformation techniques regarding the problem of scheduling VLIW instructions aimed at reducing power consumption of VLIW architectures in the instruction bus. The problem can be categorized into two types: horizontal scheduling and vertical scheduling. For the case of horizontal scheduling, we propose a bipartite-matching scheme for instruction scheduling. We prove that our greedy bipartite-matching scheme always gives the optimal switching activities of the instruction bus for given VLIW instruction scheduling policies. For the case of vertical scheduling, we prove that the problem is NP-hard, and we further propose a heuristic algorithm to solve the problem. Our experiment is performed on Alpha-based VLIW architectures and an ATOM simulator, and the compiler incorporated in our proposed schemes is implemented based on SUIF and MachSUIF. Experimental results of horizontal scheduling optimization show an average 13.30% reduction with four-way issue architecture and an average 20.15% reduction with eight-way issue architecture for transitional activities of the instruction bus as compared with conventional list scheduling for an extensive set of benchmarks. The additional reduction for transitional activities of the instruction bus from horizontal to vertical scheduling with window size four is around 4.57 to 10.42%, and the average is 7.66%. Similarly, the additional reduction with window size eight is from 6.99 to 15.25%, and the average is 10.55%.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*optimization*; B.6.3 [**Hardware**]: Logic Design—*Design Aids*

General Terms: Algorithms, Languages

Additional Key Words and Phrases: Compilers, instruction bus optimizations, low-power optimization, VLIW instruction scheduling

1. INTRODUCTION

The push for low-power design has recently gained growing importance in designing various computer systems and embedded systems. For example, an

Authors' addresses: C. Lee, J. K. Lee, T. T. Hwang, Department of Computer Science, National Tsing-Hua University, 300, Hsinchu, Taiwan; email: tingting@cs.nthu.edu.tw; S.C. Tsai, Department of Information Management, National Chi-Nan University, Pu-Li, Nan-Tou, Taiwan.
This version is a revised version. A preliminary version of this work appears in the *Proceedings of the Thirteenth International Symposium on System Synthesis* (*ISSS 2000*, Sept. 2000).

embedded system with a number of DSP processors with respective VLIW architecture must satisfy the requirements of high performance and low power consumption. In such a high-performance embedded system we need to address the issues of both high-performance computing and low power consumption. Much work on aggressive compiler and software optimization has put emphasis on delivering high-performance computing [Landskov et al. 1980; Bernstein and Rodeh 1991; Fisher 1983; Fisher et al. 1984]. However, less attention was paid to reducing power during compiler optimization. For that reason, we 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 [Chandrakasan et al. 1992]. Studies on low power design are abundant in the literature [Roy and Prasad 1992; Prasad and Roy 1993; Tsui et al. 1993; Benini and De Micheli 1995; Hachtel et al. 1994; Alidina et al. 1994; Hong et al. 1999] in which various techniques are proposed to synthesize designs with low transitional activities.

Recently, new research directions have begun to address the issues of arranging software at instruction-level to help reduce power consumption. Tiwari et al. [1994a, b] have studied scheduling techniques for reducing power consumption. These works are more related to instruction selections, which are based issues such as register accesses and lower latency instructions. Tiwari et al. also took into consideration reducing circuit-state overhead. Su et al. [1994] proposed an instruction scheduling technique called cold scheduling with Gray code addressing. Cold scheduling reduces the amount of switching activity in the control path. Chang and Pedram [1995] focused on register assignments and formulated the problem as a max-cost flow problem for power optimization. Leupers and Marwedel [1996] presented algorithms that yield high utilization of parallel AGUs by computing appropriate memory layouts for program variables, and which immediately apply to contemporary DSPs. Ye et al. [2000]. presented a power-conscious postcompilation optimization by relabeling the register fields of the compiler-generated instructions. Bellas et al. [2000] attempted to reduce L1 cache activity by placing a small L0 cache, called the L-Cache, between the L1 and the processor. They proposed a method that takes the responsibility for code modification and allocation of instructions into the L-Cache.

This new direction raises an interesting issue in compiler participation in software rearrangements for reducing power consumption for applications and systems. In this article, we report an important aspect of compiler participation in software arrangements on VLIW architectures. We investigate compiler transformation techniques with regard to the problem of scheduling VLIW instructions aimed at reducing the power consumption of VLIW architectures in the instruction bus.

The energy E consumed by a program is given by $E = P \times T$, where T is the execution time of the program [Tiwari et al. 1996; Lee et al. 1997] and P is the average power. The average power P is given by $P = \frac{1}{2} \cdot C \cdot Vdd^2 \cdot f \cdot D$, where C

is the load capacitance, Vdd the supply voltage, f the clock frequency, and D is the transition density. In compiler optimizations, if we optimize programs for the performances, T will be reduced as will energy consumption. If the compiler performs software refinements to reduce P, without software performance penalty, it will also reduce the energy consumption. Therefore, it is preferable that any power minimization technique 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. According to previous work [Irwin 1999], buses are a significant source of power dissipation due to high switching activities and large capacitive loading. The power dissipation of buses on the DEC Alpha 21064 processor is more than 15% of the total power consumption, and the power dissipation of buses on the Intel 80386 processor is more than 30% of the total power consumption. Hence, in our work, we study compiler transformation techniques to schedule VLIW instructions aimed at the reduction of transition activity in the instruction bus. We first present a cost model to estimate the bus switching activities of instruction executions on VLIW architectures. Based on the model, we further develop compiler transformation techniques to schedule VLIW instructions for reducing power consumption in the bus level.

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

Finally, our experiment was done on an Alpha-based VLIW architecture and ATOM simulator [Digital Equipment Corp. 1994] and the compiler incorporated by our proposed bipartite-matching schemes was implemented based on SUIF [Stanford Compiler Group 1994] and MachSUIF [Smith 1998]. Experimental results with an extensive set of benchmarks showed significant improvements for switching activities as compared with conventional list scheduling.

The remainder of the article 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 the article.

2. MACHINE ARCHITECTURE AND COST MODEL

2.1 Machine Architecture

Figure 1 shows an example of the target machine architecture on which our optimization is based. We focus on the reduction of switching activities for the instruction bus. The abstract VLIW machine has several execution units. In

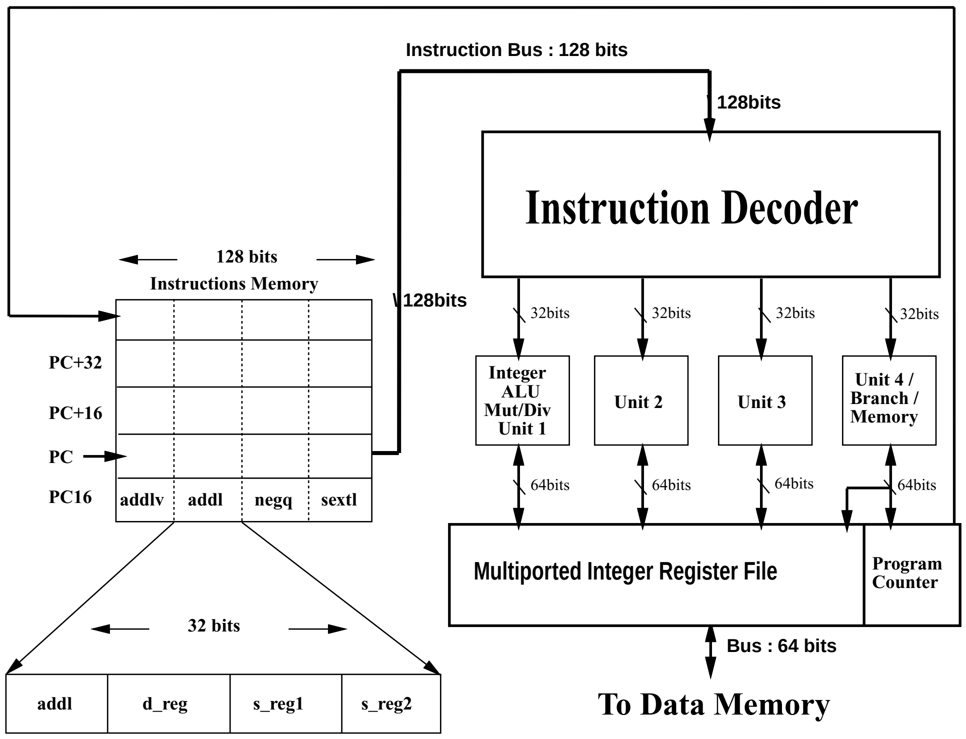

Fig. 1. Our VLIW machine architectures and bus models.

the example given in Figure 1, units 1 through 3 are integer ALUs with integer multiplication, division, and logic operation units. Unit 4 is the same as other units, performs branch/flow control, or executes 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 without load/store or branch microinstructions [Hennessy and Patterson 1996].

Figure 1 is also the architecture model by which we carry out our experiments later in Section 4. Our approach goes to minimize the switching activities of the bus from instruction memory to instruction decoder. The length of an VLIW instruction in our experiment is 128 bits. Memory addressing is byte address. Also, we use real executable instructions of the Alpha chip for experiments. We assume this VLIW machine assigns an 128 bit instruction into four function units per instruction fetch. The instruction bus is 128 bits wide.

2.2 Cost Model

The average power P is given by $P = \frac{1}{2} \cdot C \cdot Vdd^2 \cdot f \cdot D$, where C is the load capacitance, Vdd the supply voltage, f the clock frequency, and D is the transition density. Judging from the power equation, it is clear that power can be reduced by the product of capacitance loading and transition activity. Since

32 bits VLIW microinstruction in bits list view

1001 0010 1010 0111 1010 1101 0001 0000

1100 1011 0001 1101 0000 1011 0101 0100

Different Bit 0101 1001 1011 1010 1010 0110 0100 0100

Hamming Distance =15

Fig. 2. Hamming distance.

bus wires have large capacitance loading, the reduction of transition activities of buses will be very effective in reducing total power consumption. We use *hamming distance* as our cost model to estimate the transition activity in the instruction 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 $(x_1, x_2, \ldots, x_k)$ and $(y_1, y_2, \ldots, y_k)$, 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 a negative effect not only on performance but also energy consumption, we require that any power minimization technique incur 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 rearrange the codes produced by the first phase for low power without incurring a performance penalty.

In this work, a list scheduling algorithm [Landskov et al. 1980] is 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 [Landskov et al. 1980]. However, any conventional VLIW instruction scheduler [Landskov et al. 1980; Bernstein and Rodeh 1991; Fisher 1983; Fisher et al. 1984] can be used. Since the list scheduling algorithm is well-documented, we focus on algorithms as presented in the following subsections to reschedule instructions for power optimization.

3.1 Horizontal Scheduling

We first propose a horizontal scheduling algorithm to reschedule instruction components of an instruction to minimize transition activity of instruction buses; that is, microinstructions of an instruction are to be rescheduled 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 this basic block we have n VLIW instructions, and they are $X_1, X_2, \ldots, X_n$, where the instruction components of instruction X_i are given as

$$X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k}),$$

where k is the number of microinstructions within one VLIW instruction. Then our goal is to find an instruction scheduling so that the total hamming distance of n consecutive VLIW instructions $X'_1, X'_2, \ldots, X'_n$ is minimized. That is, we want to minimize

$$\Sigma_{i=1}^{n-1} H(X'_i, X'_{i+1}),$$

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

Our horizontal scheduling algorithm proceeds to reschedule microinstructions from the first instruction to the last. Initially, the first instruction is rescheduled without changing it. Iteratively, the next instruction is rescheduled to minimize hamming distance between the instruction and the last instruction already scheduled. We model the rescheduling 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 microinstruction 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 scheduled into the same position, or slot, in the whole VLIW instruction as the position of microinstruction u_i in the u_i's VLIW instruction, and does not violate the hardware constraints. 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 original version of the bipartite-matching algorithm is the maximum weight bipartite-matching algorithm. In our application of microinstruction scheduling, we "minimize" the transitional activities. So, we present our algorithm in a negative sign version. Figure 3 illustrates the construction of a bipartite graph. In this figure, there are four microinstructions in each instruction. The instruction represented by UpLayer is the last instruction already rescheduled. The instruction represented by LowLayer is the instruction to be rescheduled. The weight on the edge of u_i and l_i is $-h(u_i, l_i)$.

Now, we apply the maximum weight bipartite-matching [Fredman and Tarjan 1987] algorithm to the graph to get the matched graph. The matched graph has n links, where n is the same as the number of microinstructions in one VLIW instruction, and the matched graph has maximal weight. In other words, the matched graph has minimal switching activities between these two partitions. Each microinstruction in *UpLayer* will match an only one microinstruction in *LowLayer*. If microinstruction l_i is matched with microinstruction u_i, l_i will be scheduled into the same position as l_i in the VLIW instruction binary format. The matching procedure repeats untill all instructions are rescheduled. This matching procedure reschedules microinstructions so that the hamming

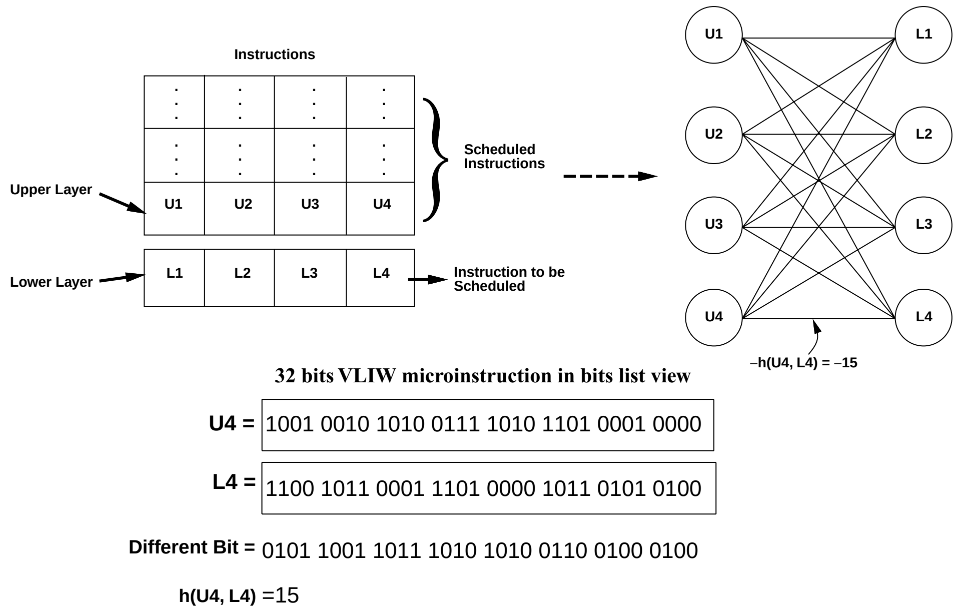
Instructions
Upper Layer
Lower Layer
U1
U2
U3
U4
L1
L2
L3
L4
Scheduled Instructions
Instruction to be Scheduled
−h(U4, L4) = −15
32 bits VLIW microinstruction in bits list view
U4 = 1001 0010 1010 0111 1010 1101 0001 0000
L4 = 1100 1011 0001 1101 0000 1011 0101 0100
Different Bit = 0101 1001 1011 1010 1010 0110 0100 0100
h(U4, L4) =15

Fig. 3. An example of bipartite-matching for horizontal scheduling

```
Input :   Array $X$ of VLIW instruction sequence with $n$ elements, i.e., $X_1, X_2, \ldots, X_n$.
          Each VLIW instruction in $X$ has $k$ microinstructions.
Output:   New optimal $X$ on power consumption.

Begin
while($i = 1$ to $n - 1$)
    $X'_{i+1}$ = maximum_weight_bipartite_matching($X_i, X_{i+1}$);
    /* maximum_weight_bipartite_matching(UpLayer, LowLayer) return an
      VLIW instruction NewLowLayer, which is a permutation of LowLayer, such
      that $-1 \times$ hamming_distance(UpLayer, NewLowLayer) is the maximum */
    Replace $X_{i+1}$ with $X'_{i+1}$;
end_while
End
```

Fig. 4. Multistage bipartite-matching algorithm to schedule VLIW instructions to minimize bus transition activities

distance between the instruction of *UpLayer* and the instruction of *LowLayer* is minimum. Figure 4 details this process.

This greedy bipartite-matching algorithm can actually produce an optimal solution for our horizontal scheduling problem. In the following, we show the proof.

THEOREM 1. *The algorithm in Figure* 4 *always gives the optimal solution to the problem of minimization of transition activity of the instruction bus when microinstructions are rescheduled with horizontal moves for a given VLIW scheduling of a basic block with n consecutive instructions.*

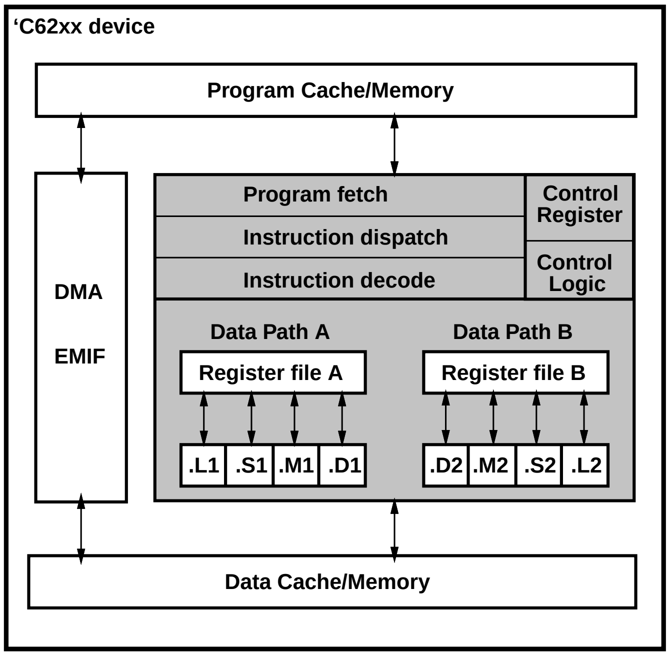

Fig. 5. TI TMS320C62XX DSP processor, VLIW architecture.

PROOF. Suppose we are given a VLIW scheduling for a basic block. Let the given scheduling be $X_1, X_2, \ldots, X_n$. The instruction components of X_i are $(x_{i,1}, x_{i,2}, \ldots, x_{i,k})$. Our goal is to show that the above algorithm minimizes the transition cost. That is, we want to find n consecutive VLIW instructions $X_1^{'}, X_2^{'}, \ldots, X_n^{'}$ to minimize $\Sigma_{i=1}^{n-1} H(X_i^{'}, X_{i+1}^{'})$, where $X_i^{'} = (\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,k}), 1 \leq i \leq n$, and $(\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,k})$ is a permutation of $(x_{i,1}, x_{i,2}, \ldots, x_{i,k})$.

It is clear that the best possible of $H(X_i', X_{i+1}')$ can be found by the above-mentioned maximum weighted matching algorithm for $i = 1, \ldots, n-1$. Let p_i be the cost between X_i and X_{i+1} obtained from the above algorithm. Let $C = \sum_{i=1}^{n-1} p_i$. We prove by contradiction. Suppose there is a better scheduling with cost $C' < C$. Let q_i, $i = 1, \ldots, n-1$, be the cost between X_i and X_{i+1} under this scheduling. We have $C' = \sum_{i=1}^{n-1} q_i < \sum_{i=1}^{n-1} p_i$. Then there exists an i such that $q_i < p_i$. But we know p_i is the best possible for X_i and X_{i+1}, a contradiction. Thus the above algorithm does find the best transition cost. □

Note that the construction of edges in a bipartite graph needs to take architecture constraints into consideration. For example, Figure 5 [Texas Instruments 1997] gives an overview of the TI TMS320C62XX DSP processor, where the function units are separated into several classes (four classes in this case). 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 function units in the same class. For example, in our model, multiply instructions can be assigned into four given function units, but in the case of TMS320C62XX, multiply operations can be done by the .Mx (M1 or M2) units.

3.2 Vertical Scheduling

Although the discussions in the previous subsection 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, care has to be taken so that no performance penalty is 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 $\langle X_1, X_2, \ldots, X_n \rangle$ and the instruction components of instruction X_i:

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

The vertical scheduling is to find a scheduling Y where Y, is $\langle Y_1, Y_2, \ldots, Y_n \rangle$, and the instruction components of instruction Y_i are:

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

The vertical scheduling Y needs to satisfy the following constraints.

(1) There exists a bijection $\delta : A \to B$, 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 new scheduling obeys the original dependence, DDG.
(3) 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 $\Sigma_{i=1}^{n-1} H(Y_i, Y_{i+1})$ is minimized. The following theorem shows that the problem to find the minimum scheduling Y for the vertical case is NP-hard. For completeness, we give a simple and self-contained proof for Theorem 2, which could also be proved by reducing from other NP-complete problems [Papadimitriou 1995].

THEOREM 2. *Suppose we are given a VLIW scheduling X for a basic block B. Let the given scheduling X be $\langle X_1, X_2, \ldots, X_n \rangle$ and the instruction components of X_i be $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$. Then the problem of finding a scheduling Y, where Y is $\langle Y_1, Y_2, \ldots, Y_n \rangle$, and a bijection δ satisfying the following criteria is NP-hard.*

(1) *There exists a bijection $\delta : A \to B$, 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 new scheduling obeys the original dependence, DDG. More formally, $\forall x_{i_1,j_1}, x_{i_2,j_2} \in A$. Suppose there is a dependence between x_{i_1,j_1} and x_{i_2,j_2} in DDG, and $\delta(x_{i_1,j_1}) = y_{i'_1,j'_1}$, $\delta(x_{i_2,j_2}) = y_{i'_2,j'_2}$. We will have (i'_1, j'_1) precede (i'_2, j'_2). It means i'_1 is smaller than i'_2.*
(3) *The scheduling Y minimizes the bus transition activities; that is, Y minimizes $\Sigma_{i=1}^{n-1} H(Y_i, Y_{i+1})$, where the instruction components of instruction Y_i*

are $(y_{i,1}, y_{i,2}, \ldots, y_{i,k})$*, where* k *is the number of microinstructions within one VLIW instruction.*

Now we want to show that the vertical scheduling is NP-hard. The extreme case of this problem is that the DDG is an empty set. In that case, virtually every instruction component of the instruction is allowed to move freely either vertically or horizontally. Then the scheduling algorithm will take the longest execution time of any other cases with data dependence constraints. We show that a special case of this problem is indeed NP-hard. Thus it implies that the general case is NP-hard. More specifically, we show a polynomial time reduction from the famous Hamiltonian path problem to our special case. We review the definition of the Hamiltonian path problem [Papadimitriou 1995] before we give the proof. The problem is simply this: given a graph, is there a path that visits each node exactly once? It is well known that the Hamiltonian path problem is NP-complete.

PROOF. It is sufficient to prove NP-hardness for $k = 1$. In this case, the problem turns out to be finding a permutation π for $x_{i,1}$s, where $1 \leq i \leq n$, such that $\sum_{i=1}^{n-1} H(x_{\pi_i,1}, x_{\pi_{i+1},1})$ is minimized. We reduce the Hamiltonian path problem to our special case.

Given an instance G of the Hamiltonian path problem with n vertices, for each edge of G we assign weight 1 and assign M to each nonedge vertex pair, where M is an integer and $M > n$. The *nonedge vertex pair* means the vertex pair without connected edges. That is, we construct a complete graph with weights 1 and M from G. Let's call the new graph G'. Now we can see G has a Hamiltonian path if and only if the corresponding permutation has minimal summation $n - 1$ over G'. The corresponding permutation is the corresponding problem in our vertical scheduling. It is clear that the reduction is polynomial. The reduction is problem reduction. The conventional usage in reducing a problem into another problem when trying to show a problem is NP-complete or NP-hard. Therefore, our special case is NP-hard, and so is the general version. □

Since the problem is NP-hard, we propose a heuristic algorithm based on allowable moving windows of instruction sets and bipartite-matching techniques (see Figure 6).

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 rescheduling of instructions of a basic block are the data dependence graph 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 a software performance penalty. The "critical path" is defined as the following. Given the required arrival time of computational output data that is determined by the timing constraint, we can compute the required arrival time of

```
Input :  1. Array $X$ of VLIW instruction with $n$ elements,i.e., $X_1, X_2, \ldots, X_n$, and
            the instruction components of $X_i$ be $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$.
         2. Register Dependence Graph ($DDG$) and critical path information.
         3. Allowable moving window size $w$.
Output:  A new scheduling $Y$ with lower power consumption.
Begin
    Let $Y_1 = X_1$.
    Construct $LowLayerSet$ with all microinstructions in $X_2, X_3, \ldots, X_w$.
    for($i = 1; i < n; i++$)
    {
        a) Add $X_{i+w}$ into $LowLayerSet$.
        b) Let $Y_i$ be $UpLayer$.
        c) Build bipartite graph $BG$ with $UpLayer$ and $LowLayerSet$, where
           the weight of edge from $u \in UpLayer$ to $l \in LowLayerSet$ is $-h(u, l)$.
        d) Remove all edges linked to $l_q \in LowLayerSet$, if $l_q$ is data dependent
           on $l_p \in LowLayerSet$.
        e) Suppose $l \in LowLayerSet$ is on critical path; we replace the weight of
           edges from $UpLayer$ to $l$ by $\infty$.
        f) Perform maximum weight bipartite matching on $BG$ to select new $Y_{i+1}$.
        g) Remove all $y \in Y_{i+1}$ from $LowLayerSet$.
    }
End
```

Fig. 6. A heuristic algorithm based on weighted bipartite-matching and allowable window for vertical scheduling.

each node within the DDG by traversing the DDG from the required input data to the computational output data. For any node $C_i \in DDG$, the timing slack is defined as

$$T\_Slack(C_i) = Required\_ArrTime(C_i) - Actual\_ArrTime(C_i).$$

The critical node is a node whose timing slack is negative, and a critical path is a signal path where the timing slack of every node down to this path is negative.

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. Initially, the first instruction is rescheduled without changing it. Iteratively, the next w instructions are candidates to be selected and rescheduled to minimize hamming distance. The microinstructions in the next w instructions have to satisfy the data dependence and critical path constraints. To satisfy the data dependence constraint, it is required that if the parents of a microinstruction have not been assigned, the microinstruction should be removed from the window, and wait for rescheduling in next iteration. The "parents of a microinstruction" indicates those other microinstructions that will generate the computational outputs needed by this microinstruction. To satisfy the critical path constraint, it is required that microinstructions that are on the critical path be rescheduled by only a horizontal move.

We also model the vertical rescheduling of microinstructions as a weighted bipartite-matching. A bipartite graph $G = (UpLayer \cup LowLayerSet, E)$ is

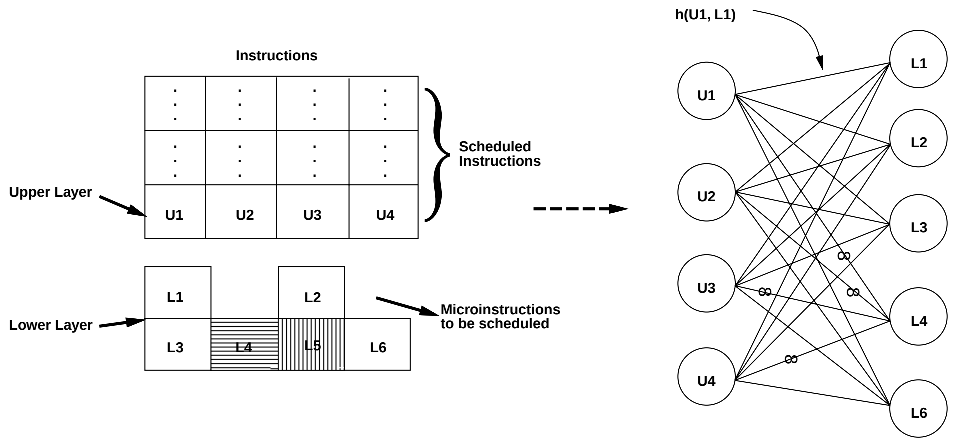

Fig. 7. An example of bipartite matching for vertical scheduling

constructed, where *UpLayer* and *LowLayerSet* are bipartite and *LowLayerSet* represents the microinstructions in the next w instructions that satisfy the data dependence constraint. Each $u_i \in \mathit{UpLayer}$ represents a microinstruction in the last instruction already scheduled and $l_i \in \mathit{LowLayerSet}$ 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 l_i is on the critical path. The weights on all edges linking l_i are defined as ∞, which guarantees that l_i will be selected in the matching. Otherwise, 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 7 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 ∞ and l_5 is deleted from the window. Figure 6 gives our heuristic algorithm.

4. EXPERIMENTS

We use the Alphalike 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 [Stanford Compiler Group 1994] and MachSUIF Library [Smith 1998]. Figure 8 shows the three phases of compilations in incorporating our algorithms into SUIF and MachSUIF systems. In the first phase of SUIF compilation, we use the SUIF library to perform classical compiler optimizations and exploit as much parallelism as possible. Next, we follow the MachSUIF's recommended steps to perform machine-dependent optimizations for the Alpha chip. In the third phase of our compilations, we work on the optimizations with low-power issues. In this phase, we load the almost-executable outputs from the MachSUIF library, and perform list scheduling to get VLIW instruction sequences. We then execute the 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

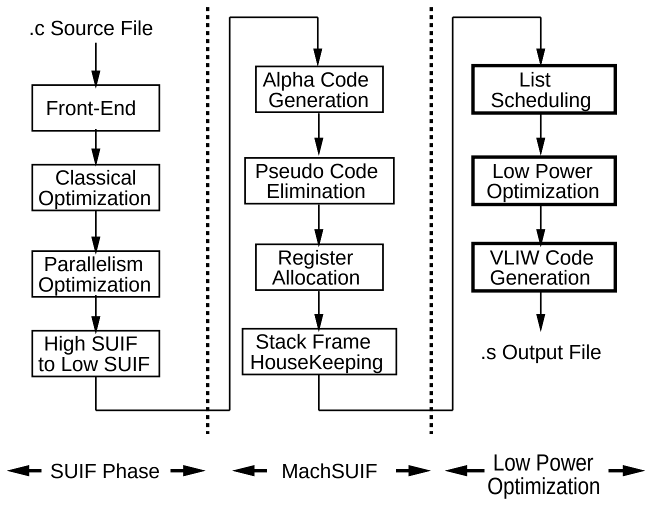

Fig. 8. Our compiler phases on low power optimizations.

our software is annotated with additional information for VLIW instructions so that the ATOM simulator [Digital Equipment Corp. 1994] can pick up the VLIW instruction information. Some of the key items in the software implementation are described below. In the implementation steps for VLIW scheduling and low power optimization, we started from the class *machine_instr* in MachSUIF library. The class *machine_instr* is an extended class from the super class *in_gen* in the SUIF library. We use class *machine_instr* and all its methods to get information about hardware architecture. In addition, we use "machineUtil.*" source files for setting up compilation steps. Once the code is generated, we use the ATOM simulator on the DEC Alpha UNIX v4.0 for simulations. Our cost model, which is described in Section 2 for switching activities and hamming distance, is incorporated into the ATOM for simulations.

Figure 9 gives the experimental result for the simulations on a four-way issue architecture described in Section 2. The gray line represents the base information which is the switching activities of the instruction bus for programs scheduled by list scheduling. It's used as the baseline. The white line represents the switching activities of the instruction bus for programs scheduled by our proposed bipartite-matching scheme with horizontal scheduling. The improvement (reduction) in switching activities ranges from 3.05 to 19.28% among test suites, with an average of 13.30%. The test suites in the experiment include three parts. The first five suites of Figure 9 are from the common integer benchmarks listed in FAQ of comp.benchmarks [Aburto et al. 1997]. The next suite is the text grep utility routine taken from GNU Grep v2.2. The rest of the 22 test suites in Figure 9 are from GNU TextUtils v1.22. Those text-processing utilities are all integer programs. The integer benchmarks tested with their self-attached data set and we used those text utilities to process their own source files.

Focus is now directed to Figure 10 which gives the experimental result for the simulations on an architecture similar to the previous experiment but

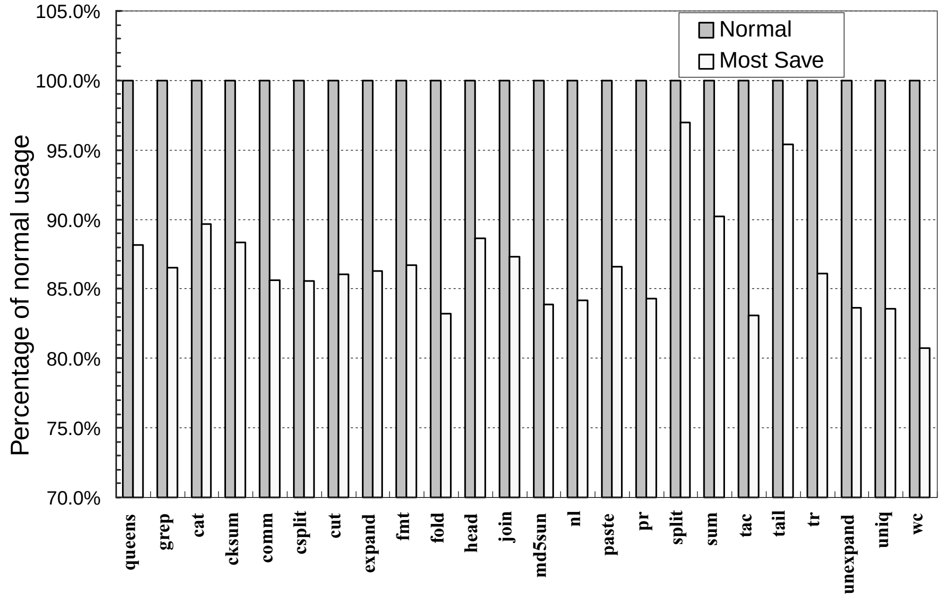

Fig. 9. Switching activities with four-way issues.

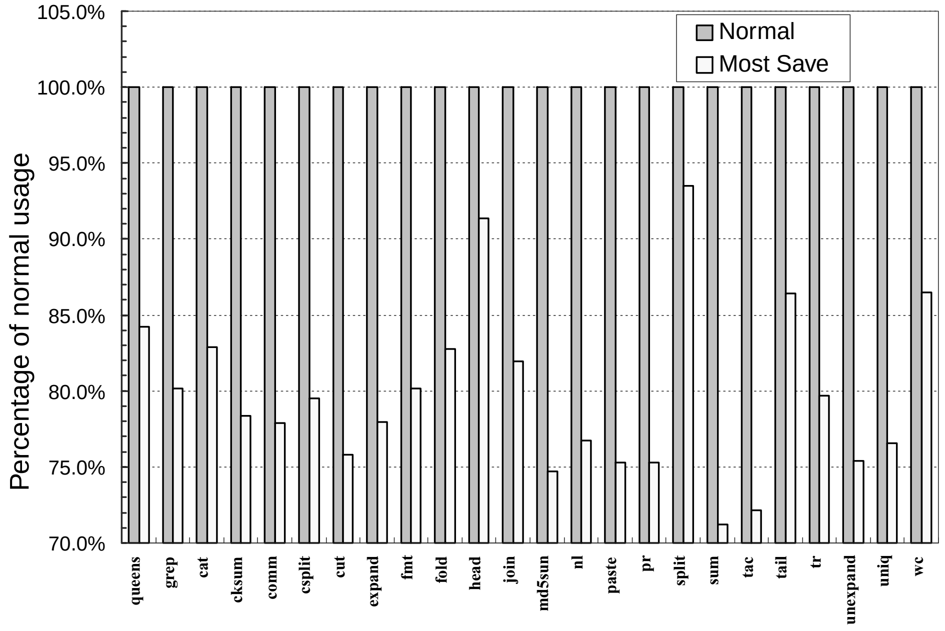

Fig. 10. Switching activities with eight-way issues.

with eight-way issues. Again, the gray line is the base information which is the switching activities of the instruction bus for programs scheduled by list scheduling. The white and gray color lines represent the results with horizontal scheduling. The average improvement (reduction) in switching

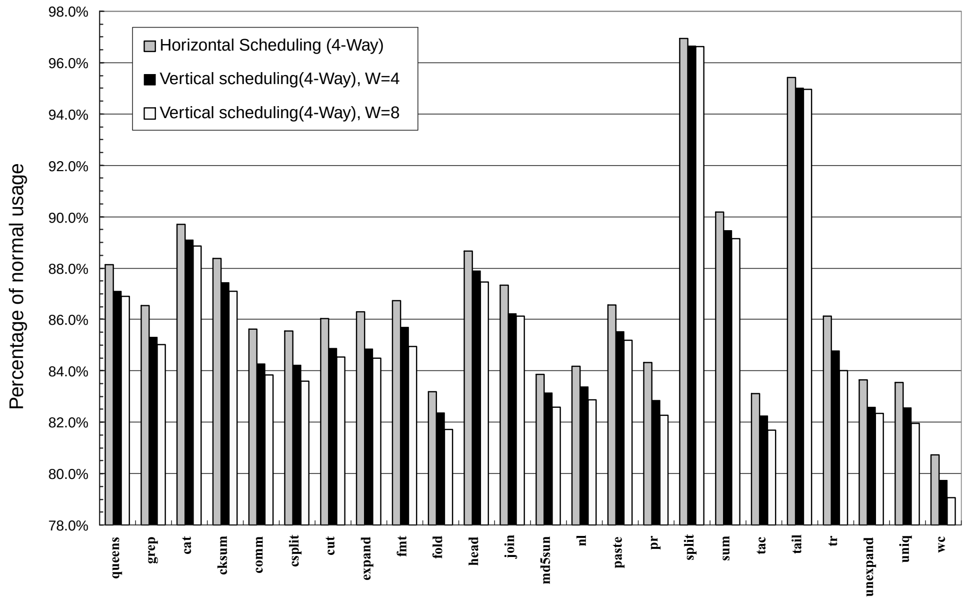

Fig. 11. Vertical scheduling switching activities with four-way issues.

activities is around 20.15% over the list scheduling by incorporating our schemes.

Figure 11 shows the experimental results for vertical scheduling with a four-way issue architecture. The gray line is the best case in four-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 additional enhancement from horizontal scheduling to vertical scheduling with window size four is around 4.57 to 10.42%, and the average is 7.66%. Similarly, the additional enhancement with window size eight is from 6.99 to 15.25%, and the average is 10.55%. Recently we learned the related work done by Ye et al. [2000]; they relabel the registers to reduce the transition on the instruction bus. Their performance gain is around 10%. We feel their methods differ from ours, but the performance improvements are in line. An interesting open problem that remains to be explored is how to combine these two schemes.

5. CONCLUSION

In this article we first described a model for calculating the bus switching activities of instruction executions on VLIW architectures. Based on the model, we investigated the compiler transformation techniques to schedule VLIW instructions aimed at reducing the power consumption of VLIW architectures in the instruction bus. Our experiment was done on Alpha-based VLIW architectures and the ATOM simulator. Our compiler was implemented based on SUIF and MachSUIF, and by incorporating our proposed schemes. Experimental results showed significant improvements in power consumption over conventional list scheduling for an extensive set of benchmarks by incorporating our proposed schemes. We think our work is important for a class of high-performance embedded systems, where we need to address both high-performance computing and power consumption issues.

REFERENCES

ABURTO, A., SILL, D., AND THOMPSON, D. 1997. *comp.benchmarks FAQ*. Computer Sciences Department, University of Wisconsin, http://www.cs.wisc.edu/ thomas/comp.benchmarks.FAQ.html.

ALIDINA, M., MONTEIRO, J., DEVADAS, S., GHOSH, A., AND PAPAEFTHYMIOU, M. 1994. Precomputation-based sequential logic optimization for low power. In *IEEE Trans. VLSI Systems 2*, 4 (Dec.), 426–436.

BELLAS, N., HAJJ, I. N., POLYCHRONOPOULOS, C. D., AND STAMOULIS, G. 2000. Architectural and compiler techniques for energy reduction in high-performance microprocessors. *IEEE Trans. VLSI Syst. 8*, 3 (June), 317–326.

BENINI, L. AND DE MICHELI, G. 1995. State assignment for low power dissipation. *IEEE J. Solid-State Circ. 30*, 3 (March), 258–268.

BERNSTEIN, D. AND RODEH, M. 1991. Global instruction scheduling for superscalar machines. In *Proceedings of Conference on Programming Language Design and Implementation*. ACM Press, Toronto, Ontario, Canada, 11–22.

CHANDRAKASAN, A. P., SHENG, S., AND BRODERSEN, R. W. 1992. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits 27*, 4 (Apr.), 473–484.

CHANG, J.-M. AND PEDRAM, M. 1995. Register allocation and binding for low power. In *Proceedings of the Design Automation Conference*. ACM Press, San Francisco, California, United States, 29–35.

DIGITAL EQUIPMENT CORPORATION 1994. *ATOM User Manual*. Digital Equipment Corporation, Maynard, Massachusetts, United States.

FISHER, J. A. 1983. Very long instruction word architectures and the eli-512. In *Proceedings of the International Symposium on Computer Architecture*. IEEE Computer Society Press, Stockholm, Sweden, 140–150.

FISHER, J. A., ELLIS, J., RUTTENBERG, J., AND NICOLAU, A. 1984. Parallel processing: A smart compiler and a dumb machine. In *Proceedings of the Symposium on Compiler Construction*. ACM Press, Montreal, Canada, 37–47.

FREDMAN, M. L. AND TARJAN, R. E. 1987. Fibonacci heap and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery 34*, 3 (July), 596–615.

HACHTEL, G. D., HERMIDA, M., PARDO, A., PONCINO, M., AND SOMENZI, F. 1994. Re-encoding sequential circuits to reduce power dissipation. In *Proceedings of International Conference on Computer-Aided Design*. IEEE Computer Society Press, San Jose, California, United States, 70–73.

HENNESSY, J. L. AND PATTERSON, D. A. 1996. *Computer Architecture - A Quantitative Approach, 2nd Edition*. Morgan Kaufmann Publishers, Inc, 340 Pine Street, 6th Floor San Francisco, California 94104, United States.

HONG, I., KIROVSKI, D., QU, G., POTKONJAK, M., AND SRIVASTAVA, M. B. 1999. Power optimization of variable-voltage core-based systems. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 18*, 12 (Dec.), 1702–1714.

IRWIN, M. J. 1999. Tutorial : Power reduction techniques in soc bus interconnects. In *IEEE International ASIC/SOC Conference*. IEEE Computer Society Press, Washington, D.C., United States.

LANDSKOV, D., DAVIDSON, S., AND SHRIVER, B. 1980. Local microcode compaction techniques. *ACM Comput. Surv. 12*, 3 (Sept.), 261–294.

LEE, C., LEE, J. K., AND HWANG, T. 2000. Compiler optimization on instruction scheduling for low power. In *Proceedings of 13th International Symposium on System Synthesis*. ACM Press, Madrid, Spain, 55–60.

LEE, M. T.-C., TIWARI, V., MALIK, S., AND FUJITA, M. 1997. Power analysis and minimization techniques for embedded dsp software. *IEEE Trans. VLSI Systems 5*, 1 (Mar.), 123–133.

LEUPERS, R. AND MARWEDEL, P. 1996. Algorithms for address assignment in dsp code generation. In *Proceedings of International Conference on Computer-Aided Design*. IEEE Computer Society Press, San Jose, California, United States, 109–113.

PAPADIMITRIOU, C. H. 1995. *Computational Complexity*. Addison-Wesley, 75 Arlington Street, Suite 300, Boston, Massachusetts 02116, United States.

PRASAD, S. C. AND ROY, K. 1993. Circuit activity driven multilevel logic optimization for low power reliable operation. In *Proceedings of the European Design Automation Conference*. IEEE Computer Society Press, Paris, France, 368–372.

ROY, K. AND PRASAD, S. C. 1992. Syslop: synthesis of cmos logic for low-power applications. In *Proceedings of the International Conference on Computer Design*. IEEE Computer Society Press, Cambridge, Massachusetts, United States, 464–467.

SMITH, M. D. 1998. *The SUIF Machine Library*. Division of Engineering and Applied Sciences, Harvard University, http://www.eecs.harvard.edu/hube/software/v130/machine.html.

STANFORD SUIF COMPILER GROUP. 1994. http://suif.stanford.edu/suif/suif1/docs/suif_toc.html.

SU, C.-L., TSUI, C. Y., AND DESPAIN, A. M. 1994. Low power architecture design and compilation techniques for high-performance processors. In *Proceedings of IEEE COMPCON*. IEEE Computer Society Press, San Francisco, California, United States, 489–498.

TEXAS INSTRUMENTS INCORPORATED 1997. *TMS320C62xx CPU and Instruction Set Reference Guide*. Texas Instruments Incorporated, 12500 TI Boulevard, Dallas, Texas 75243-4136, United States.

TIWARI, V., MALIK, S., AND WOLFE, A. 1994a. Compilation techniques for low energy: An overview. In *Proceedings of the Symposium on Low Power Electronics*. IEEE Computer Society Press, San Diego, CA, United States, 38–39.

TIWARI, V., MALIK, S., AND WOLFE, A. 1994b. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. VLSI Systems 2*, 4 (Dec.), 437–445.

TIWARI, V., MALIK, S., WOLFE, A., AND LEE, T.-C. 1996. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing Systems 13*, 2 (Aug.), 1–18.

TSUI, C.-Y., PEDRAM, M., AND DESPAIN, A. M. 1993. Technology decomposition and mapping targeting low power dissipation. In *Proceedings of the Design Automation Conference*. ACM Press, Dallas, Texas, United States, 68–73.

W. YE, N. V., KANDEMIR, M., AND IRWIN, M. J. 2000. The design and use of simplepower: A cycleaccurate energy estimation tool. In *Proceedings of the Design Automation Conference*. ACM Press, Los Angeles, California, United States, 340–345.

Received October 2000; revised July 2001; accepted February 2003