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 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 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 ATOM simulator, and our compiler incorporated by our proposed schemes is implemented based on SUIF and MachSUIF. Experimental results of horizontal scheduling optimization show average 13.30% improvements with 4-way issue architecture and average 20.15% improvement with 8-way issue architecture for power consumptions of instruction bus as compared with conventional list scheduling for an extensive set of benchmarks. The additional enhancement from horizontal scheduling to vertical scheduling with window size 4 is around 4.57% to 10.42%, and the average is 7.66%. Similarly, the additional enhancement with window size 8 is from 6.99% to 15.25%, and the average is 10.55%.

Keywords: Low-Power Optimization, Compilers, SUIF Tool, MachSUIF Tool.

1 Introduction

The push for low power design has recently gained growing importance in designing various computer systems and embedded systems. For example, the embedded system which has a number

*The correspondence author of this work is TingTing Hwang. Her e-mail address is tingting@cs.nthu.edu.tw, FAX number is 886-3-5723694, phone number is 886-3-5715131 EXT. 3540, and postal address is Prof. TingTing Hwang, Department of Computer Science, National Tsing-Hua University, 300, Hsinchu, Taiwan. The preliminary version [1] of this work appears in 13th International Symposium on System Synthesis(ISSS 2000, Sept. 2000).

of DSP processors with respective VLIW architecture must satisfy requirements of high performance and low power consumption. To such a high-performance embedded system, we need to address both the issues of high-performance computing and low power consumption. Much work on aggressive compiler and software optimization has put emphasis on delivering high-performance computing [2, 3, 4, 5]. However, less attention was paid to reducing power during compiler optimization. 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 [6]. Studies on low power design are abundant in the literature [7, 8, 9, 10, 11, 12, 13] 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 [14, 15]. To be noticeable is that 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 [15] and the swapping of operands for booth multiplier [14]. 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 report an important aspect of compiler participation in software arrangements on VLIW architectures. We investigate the compiler transformation techniques to the problem of scheduling VLIW instructions aimed to reduce 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 [14] 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 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, in our work, we will study compiler transformation techniques to schedule VLIW instructions 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 compiler transformation techniques to schedule VLIW instructions for reducing the power consumption in the bus level.

The problem can be classified into horizontal scheduling and vertical scheduling categories. For horizontal scheduling, we propose a bipartite-matching scheme for instruction scheduling. We prove that 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 is done on an Alpha-based VLIW architectures and ATOM simulator [16] and our compiler incorporated by our proposed bipartite-matching schemes is implemented based on SUIF [17] and MachSUIF [18]. Experimental results with an extensive set of benchmarks show significant improvements for power consumptions as compared with conventional list scheduling.

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

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 without load/store or branch microinstructions [19].

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. We assume this VLIW machine assign an 128 bit instruction into 4 function units per instruction fetch. The instruction bus is 128-bit wide.

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 bus wires have large capacitance loading, the reduction of transition activities of

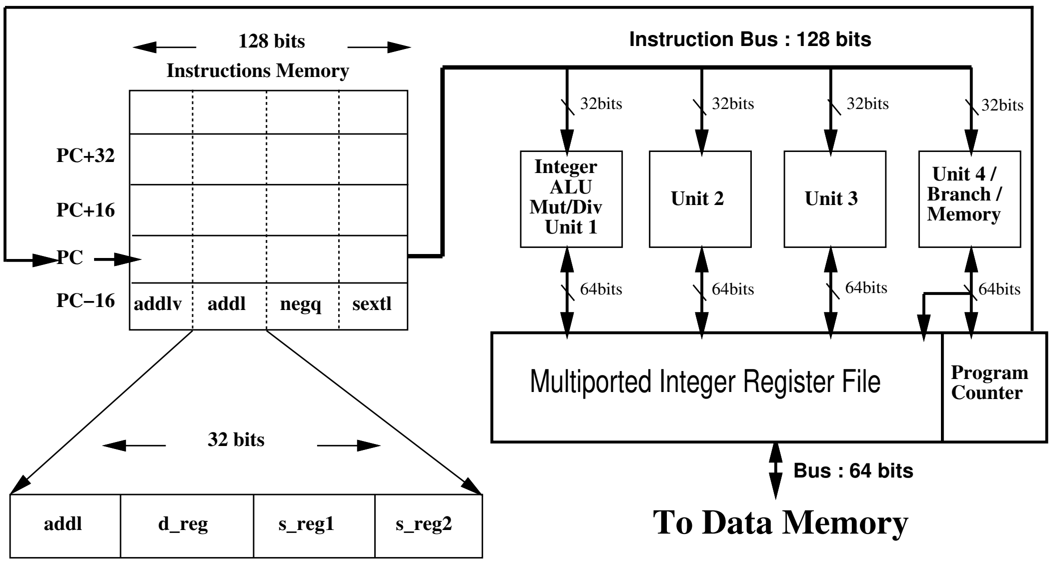

Figure 1: Our VLIW Machine Architectures and Bus Models

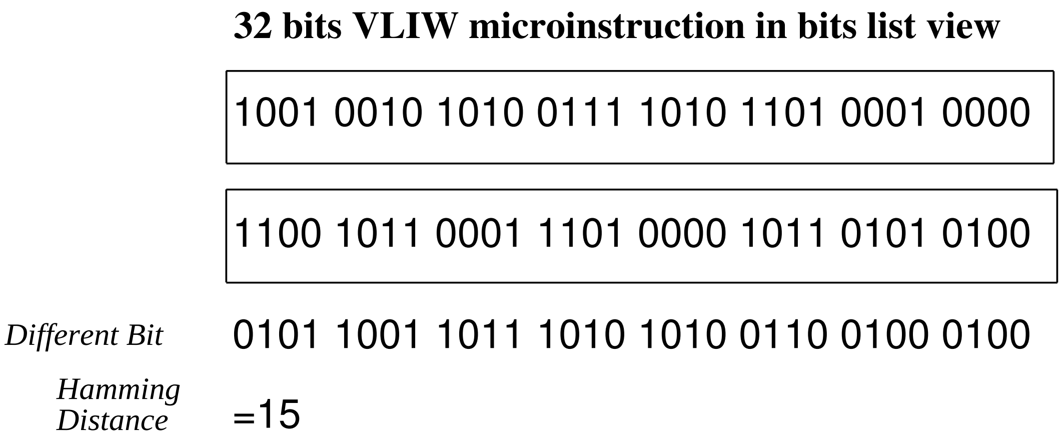

Figure 2: Hamming Distance

buses will be very effective in reducing total power consumption. We will 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 $(x_1, x_2, ..., x_k)$ 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 [2] 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 [2]. However, any conventional VLIW instruction scheduler [2, 3, 4, 5] can be used. Since list scheduling algorithm is well-documented, we will focus on algorithms as presented in the following subsections to re-schedule instructions for power optimization.

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

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

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 $X_1^{'}, X_2^{'}, ..., X_n^{'}$ is minimized. That is, we want to minimize

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

where $X_i^{'} = (\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,k}), 1 \leq i \leq n$, 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

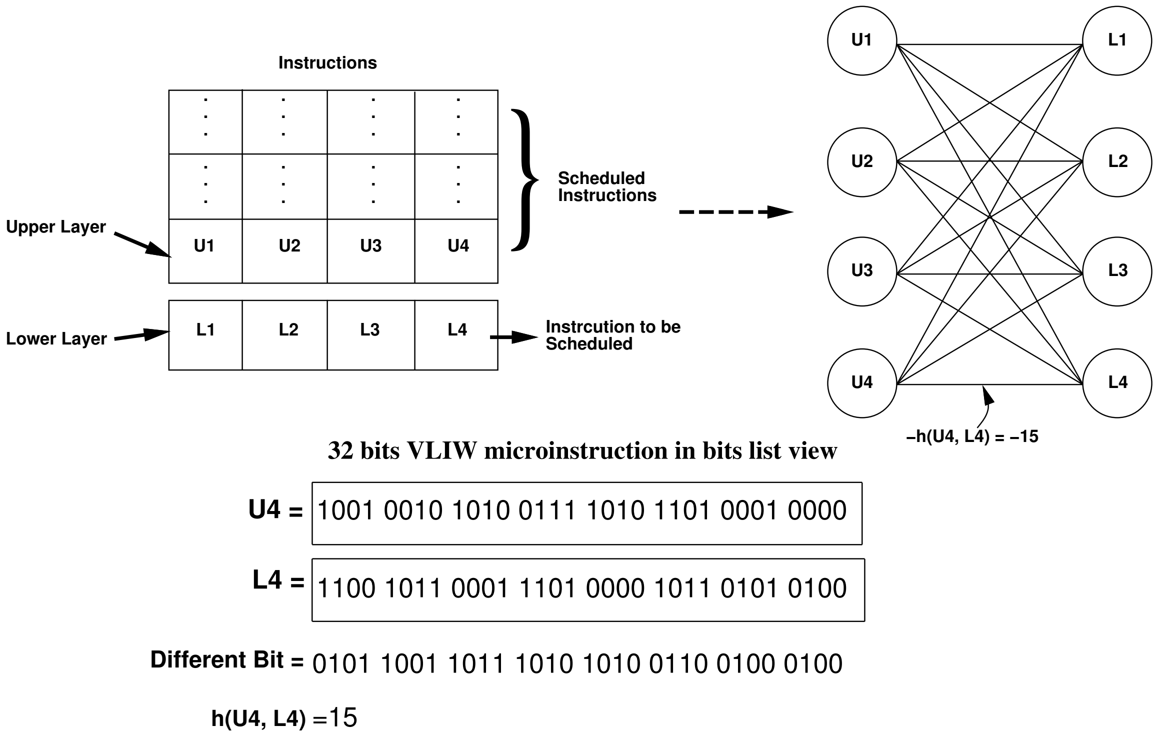

Figure 3: An example of bipartite matching for horizontal scheduling.

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 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 [20] 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. Figure 4 is the detail of this process.

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

Input : Array X of VLIW instruction sequence with n elements, i.e. X_1,X_2, ...,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

Figure 4: (`Algorithm 1`) Multi-stage Bipartite-Matching Algorithms to schedule VLIW instructions to minimize bus transition activities.

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

Proof 1
Suppose we are given a VLIW scheduling for a basic block. Let the given scheduling be $X_1, X_2, ..., X_n$. The instruction components of X_i is $(x_{i,1}, x_{i,2}, ..., x_{i,k})$.

Our goal is to show that the above algorithm minimizes the transition cost. I.e., we want to find n consecutive VLIW instructions $X'_1, X'_2, ..., X'_n$ to minimize $\Sigma_{i=1}^{n-1} \ H(X'_i, X'_{i+1})$, where $X'_i = (\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,k}), 1 \le i \le n$, and $(\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,k})$ is a permutation of $(x_{i,1}, x_{i,2}, ..., x_{i,k})$.

We prove by induction. It is clear for $n = 2$. Now suppose it holds for $X_1, X_2, \cdots, X_{s-1}$ and let the optimal transition cost obtained from our algorithm be C, which is $\sum_{i=1}^{s-2} H(X'_i, X'_{i+1})$. Consider an extra instruction X_s. By applying our algorithm we will find the minimum weighted bipartite matching between X_s and X'_{s-1}. Let the minimum weight be p. Suppose there is a smaller transition cost for $X_1, \cdots, X_s$, say C', which is smaller than $C + p$. Since p is the smallest possible between X_s and X_{s-1}, we have $(C' - p) < C$, which means that C is not the optimal transition cost for $X_1, \ldots, X_{s-1}$. It contradicts our assumption. Thus, there cannot be a smaller cost than $C + p$ for scheduling $X_1, \cdots, X_s$. This shows our algorithm finds the optimal transition cost.
End of Proof

Note that the construction of edges in a bipartite graph need to take architecture constraints into

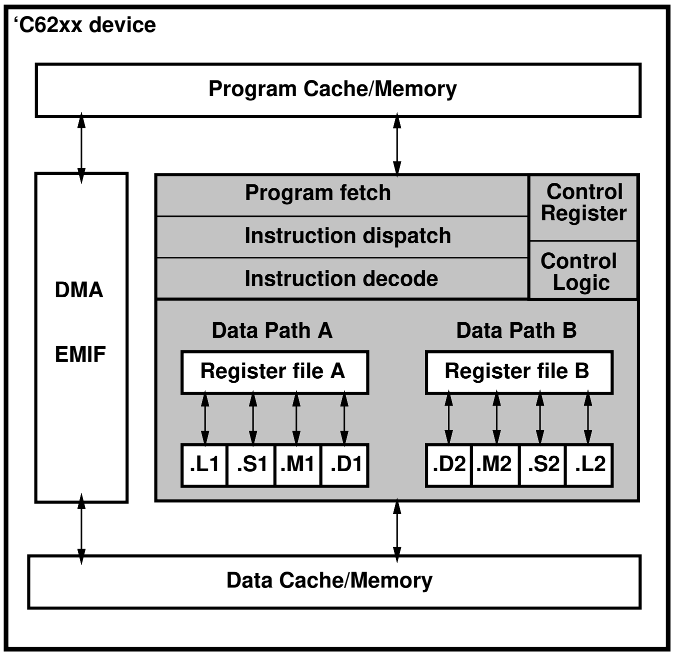

Figure 5: TI TMS320C62XX DSP Processor, VLIW Architecture.

consideration. For example, Figure 5[21] gives the overview of TI TMS320C62XX DSP processor, where the function units are separated into several classes (4 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 functions units in the same class. For example, in our model, multiply instructions can be assigned into 4 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

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,

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

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.

Theorem 2 Suppose we are given a VLIW scheduling X for a basic block B. Let the given scheduling X be $< X_1, X_2, ..., X_n >$ and the instruction components of X_i be $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,k})$.

Then the problem to find a scheduling Y, where Y is $< Y_1, Y_2, ..., Y_n >$, 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') precedes (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 is $(y_{i,1}, y_{i,2}, ..., y_{i,k})$, where k is the number of microinstructions within one VLIW instruction.

Now, we want to show that the vertical scheduling is NP-hard. As the extreme case of this problem is that the DDG (data dependence graph) is an empty set. In that case, virtually, every instruction component of the instruction is allowed to move freely either vertically or horizontally. The selection of the minimum bus transition activities in this case will result in worst performance results. We will 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 Hamiltonian Path Problem [22] 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 2
It is sufficient to prove the 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 non-edge vertex pair, where M is an integer and $M > n$. 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'. It is clear that the reduction is polynomial. Therefore, our special case is NP-hard, and so is the general version.
End of Proof

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 re-schedule 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 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

Input : 1. Array X of VLIW instruction with n elements,i.e.X_1,X_2, ...,X_n, and the instruction components of X_i be $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,k})$.

2. Dependence Graph (DDG) and critical path information.

3. A 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 $X_2, X_3, ..., X_w$.

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 ∞.

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

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

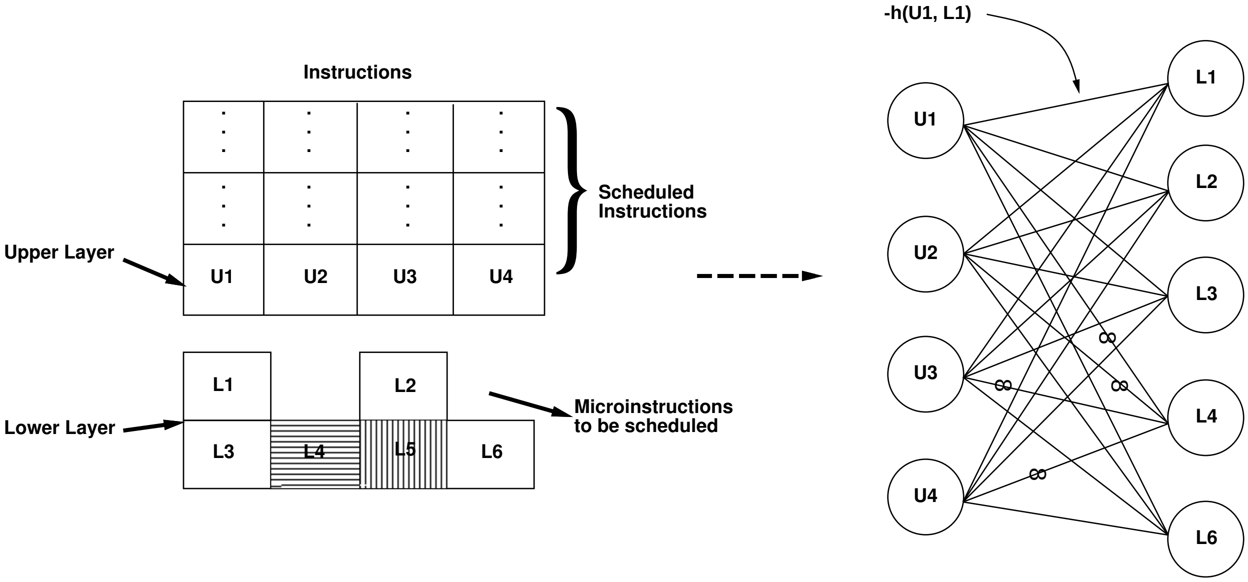

Figure 7: An example of bipartite matching for vertical scheduling.

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 Alpha-like 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 [17]and MachSUIF Library [18]. 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 many parallelism as possible. Next, we follow the MachSUIF's recommended steps to perform machine-dependent optimizations for 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 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 [16] 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 his super class *in_gen* in SUIF library. We use class *machine_instr* and all his 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 ATOM simulator on DEC Alpha UNIX v4.0 for simulations. Our cost model described in Section 2 for switching activities and hamming distance is incorporated into ATOM for simulations.

Figure 9 gives the experimental result for the simulations on a 4-way issue architecture described in Section 2. The line with the gray 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 bipartite matching scheme with horizontal scheduling. The improvement (reduction) with switching activities ranges from 3.05% to 19.28% among test suites, and with the average of 13.30%. The test suites in the experiment include three parts. The first five suites of Figure 9 are from the common benchmarks listed in FAQ of comp.benchmarks [23]. The next suite is the 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.

Focus is now directed to Figure 10 which gives the experimental result for the simulations on an architecture similar to the previous experiment but with 8-way issues. Again, the gray 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 horizontal scheduling. The average improvement (reduction) with switching activities is around 20.15% over the list scheduling by incorporating our schemes.

Figure 11 is the experimental results for vertical scheduling with a 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 additional enhancement from horizontal scheduling to vertical scheduling with window size 4 is around 4.57% to 10.42%, and the average is 7.66%. Similarly, the additional enhancement with window size 8 is from 6.99% to 15.25%, and the average is 10.55%.

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 transforma-

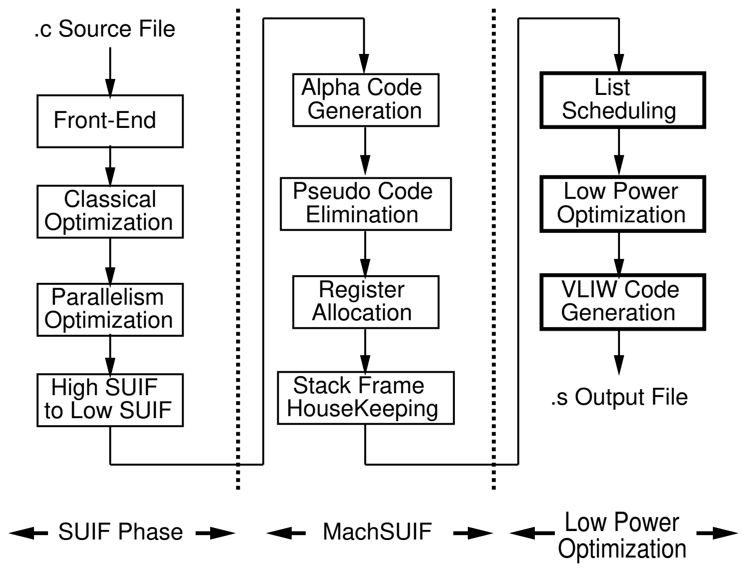

Figure 8: Our compiler phases on low power optimizations

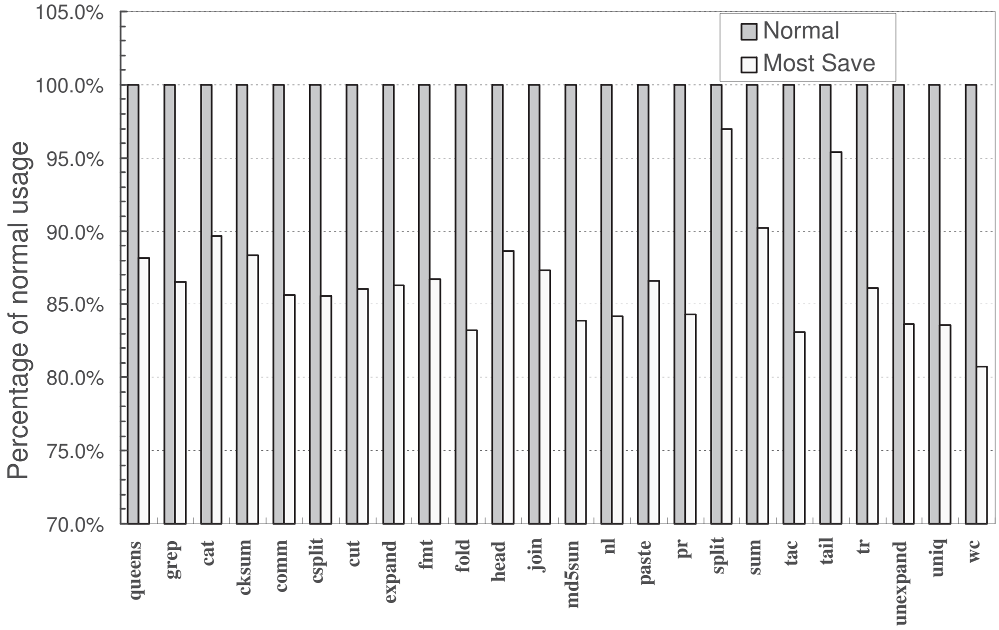

Figure 9: Switching Activities with 4-way Issues

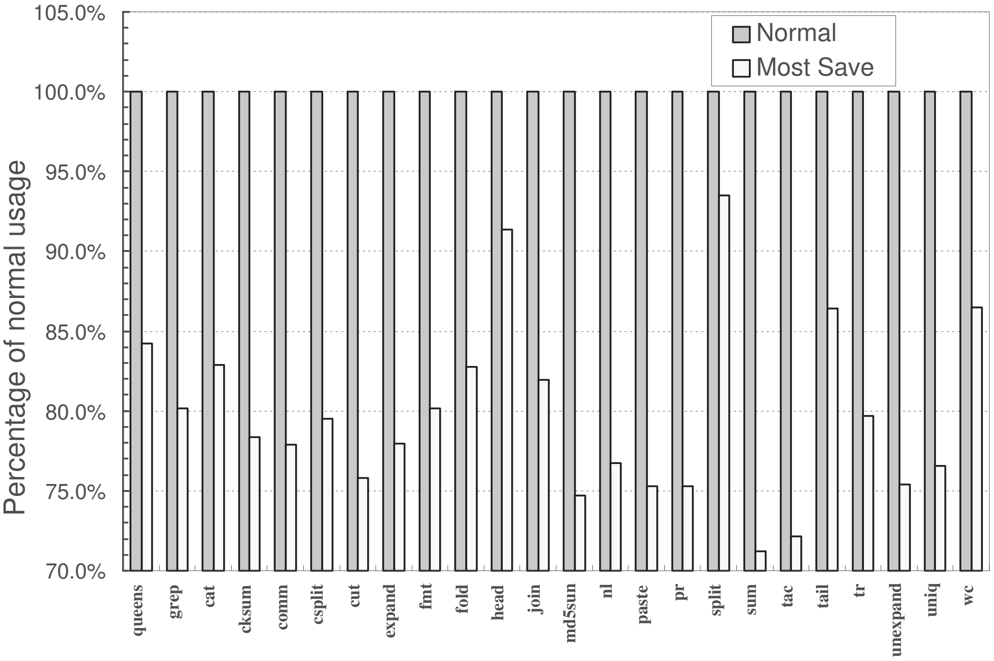

Figure 10: Switching Activities with 8-way Issues

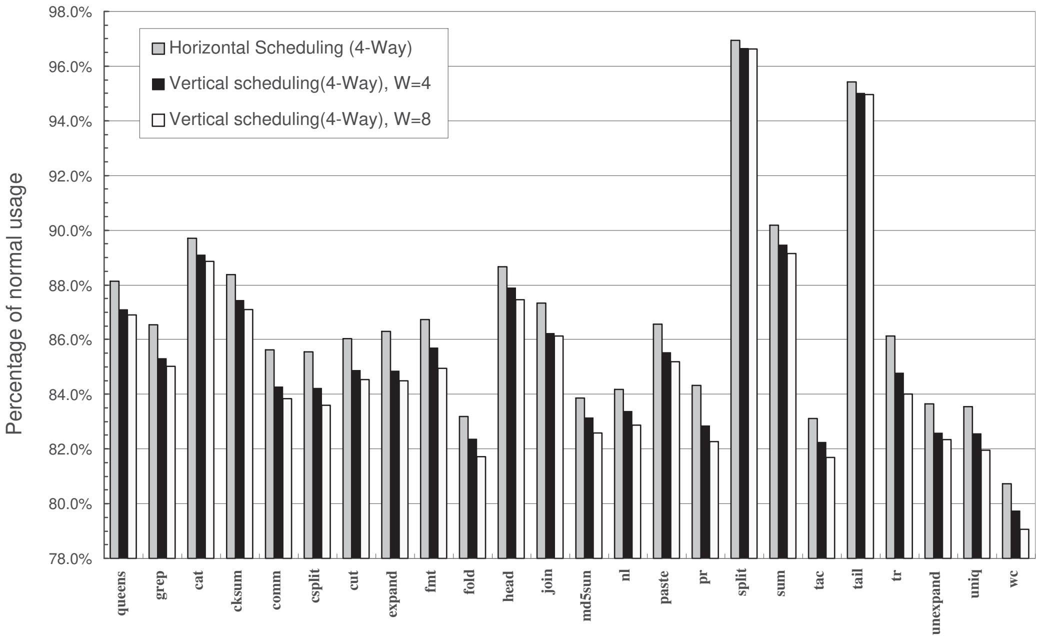

Figure 11: Vertical Scheduling Switching Activities with 4-way Issues

tion techniques to schedule VLIW instructions aimed 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 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] Chingren Lee, and Jenq Kuen Lee, and TingTing Hwang "Compiler Optimization on Instruction Scheduling for Low Power", *Proceedings of 13th International Symposium on System Synthesis*, September 2000.

[2] David Landskov, and Scott Davidson, and Bruce Shriver, "Local Microcode Compaction Techniques", *ACM Computing Surveys*, pp.261-294, September 1980.

[3] D. Bernstein, and M. Rodeh, "Global Instruction Scheduling for Superscalar Machines", *Proceedings of SIGPLAN '91 Conf. on Programming Language Design and Implementation*, June 1991.

[4] J. A. Fisher, "Very Long Instruction Word Architectures and the ELI-512", *Proceedings of the 10th International Symposium on Computer Architecture*, pp. 140-150, 1983.

[5] J. A. Fisher, and J.R. Ellis, and J.C. Ruttenberg, and A. Nicolau, "Parallel Processing: A Smart Compile and a Dumb Machine", *Proceedings of the ACM SIGPLAN 1984 Symposium on Compiler Construction*, 1984.

[6] A. P. Chandrakasan, and S. Sheng, and R. W. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, no. 4, pp. 473-484, April 1992.

[7] K. Roy, and S. C. Prasad, "SYCLOP: Synthesis of CMOS Logic for Low Power Applications", *Proceedings of the ICCD*, pp. 464-467, 1992.

[8] S. C. Prasad, and K. Roy, "Circuit Activity Driven Multilevel Logic Optimization for Low Power Reliable Operation", *Proceedings of the EDAC'93 EURO-ASIC*, pp. 368-372, February 1993.

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

[10] Luca Benini, and G. De Micheli, "State Assignment for Low Power Dissipation", *IEEE Journal of Solid State Circuits*, Vol. 30, no. 3, pp.258-268, March 1995.

[11] G. Hachtel, and M. Hermida, and A. Pardo, and M. Poncino, and F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation", *Proceedings of ICCAD' 94*, pp. 70-73, 1994.

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

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

[14] 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.

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

[16] Digital Equipment Corporation, *ATOM User Manual*, Maynard, Massachusetts, Mar. 1994.

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

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

[19] John L. Hennessy, and David A. Patterson, "Computer Architecture - A Quantitative Approach, 2nd Edition", *Morgan Kaufmann Publishers, Inc*, 1996

[20] 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.

[21] Texas Instruments, *TMS320C62xx CPU and Instruction Set Reference Guide*, Texas Instruments Digital Signal Processing Solutions, 1997.

[22] C. Papadimitriou, "Computational Complexity", *Addison Wesley*, 1995.

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