A Branch and Bound Algorithm for Fast Power-Aware Scheduling of VLIW Instructions

Shu Xiao and Edmund M-K. Lai
School of Computer Engineering, Nanyang Technological University, Singapore 639798
Email: shu_x@pmail.ntu.edu.sg, asmklai@ntu.edu.sg

***Abstract*— An instruction word in VLIW (very long instruction word) processors consists of a variable number of individual instructions. Therefore the power consumption variation over time significantly depends on the parallel instruction schedule generated by the compiler. Sharp power variations across time cause power supply noises, degrade chip reliability and accelerate battery exhaustion. Hence power variation reduction without compromising execution speed is an important instruction scheduling constraint in embedded VLIW systems. The instruction scheduling problem is generally known to be NP-complete and in need of an effective and efficient algorithm to solve it. This paper proposes a branch and bound algorithm for instruction scheduling of VLIW architectures that effectively minimizing power variation without degrading the speed. The problem is solved by a branch and bound algorithm by adaptively adding problem-specific constraints (cuts) and applying a tight lower bound. Our experimental results demonstrate the ability of our algorithm for efficiently minimizing time variation of power consumption.**

I. Introduction

Multimedia processing software typically exhibit high instruction-level parallelism (ILP). A single instruction word of VLIW (very long instruction word) processors contains a variable number of individual instructions which are executed on different functional units in parallel. Therefore, VLIW processors provide a means to efficiently exploit ILP because of their parallel processing power. The instruction scheduling techniques used by VLIW compilers are thus essential to improve the execution speed. However, depending on the parallel schedule generated by the compiler, power consumption variation over time can also be significantly different. TMS320C6711 from Texas Instrument is an 8-issue VLIW processor. Data in Appendix I quoted from [1] shows the impact of instruction scheduling on power variation of TMS320C6711 processor core. Since sharp power variations across time steps cause power supply noises, degrade chip reliability and accelerate battery exhaustion [2], [3], power variation reduction without compromising execution speed becomes an important instruction scheduling constraint in embedded VLIW systems. This approach is motivated by the observation that a large number of instructions in the parallel schedules have schedule slacks [4], i.e., each of these instructions can be scheduled in one of the many time steps without degrading the performance of the schedule. Thus it is possible to change the issue of certain non-critical instructions and obtain a schedule with balanced power consumption without performance degradation.

Some instruction scheduling algorithms for ILP processor have been proposed. However, most of them produce schedules that meet deadline constraints. Published works on instruction scheduling of ILP processors for power variation minimization are still relatively few. Among them, Yun [5] extended iterative modulo scheduling by adding a heuristic for power-aware scheduling of VLIW processor cores. Yang [4] proposed a mixed integer programming formulation to derive the optimal schedules. In contrast with the heuristic in [5], the integer programming formulation has the potential to derive optimal results and it can be used to evaluate any heuristic algorithm. Yang [4] used a commercial library (ILOG CPLEX) to obtain the solutions. The experimental results and examples in [4] did motivate power variation reduction over time in VLIW architecture. However, no specific information about the computation time was provided. In our experiments for replicating their approach, we found that due to the lack of problem-specific information, the average time used by CPLEX90 to solve the mixed integer problem is quite unacceptable for ILP compilers.

In this paper, the mixed integer program is also used to formulate the problem. However, we propose a branch and bound algorithm to efficiently solve this problem of scheduling of VLIW instructions without compromising the speed. The algorithm adaptively adds problem-specific constraints (cuts) and applies a tight and fast lower bound algorithm to guide the search for the optimal solution. The algorithm is evaluated on instruction blocks of various sizes. The results show our algorithm gets the same optimized power variation values while an average improvement in required computation time of 68.62% compared with CPLEX90 as [4].

The rest of this paper is organized as follows. In Section II we present our mixed integer programming formulation of the problem. A customized branch and bound algorithm is developed in Section III to solve the problem. The effectiveness and efficiency of the algorithm is evaluated and the results are presented in Section IV.

II. Problem Formulation

A. Power Model

VLIW processors use very long instruction words to execute multiple instructions simultaneously on seperate functional units (see Fig. 1). Each instruction takes different amount of time to execute. We divide the time line into equal length time slots. A power cost p_i^j is associated with each instruction

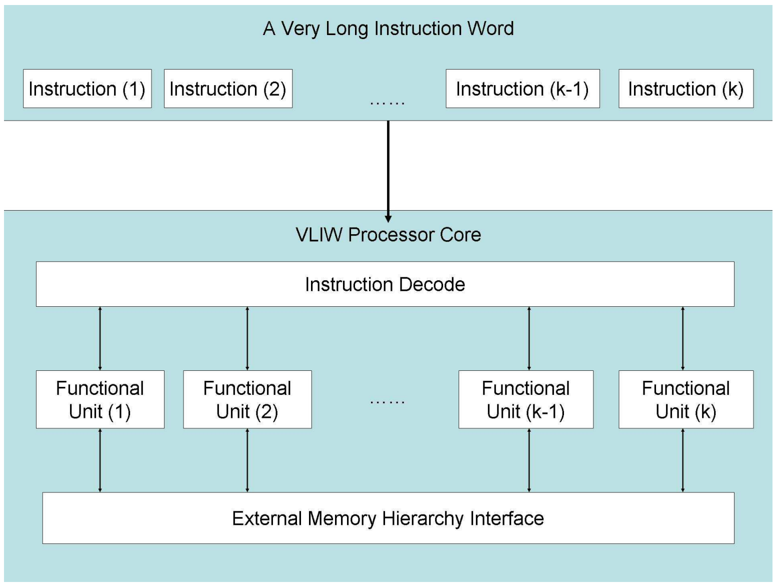

Fig. 1. A simplified model of a VLIW processor core.

i which represents the power consumed by this instruction in the *j*-th time slot.

Given a program execution schedule N, let N_i represents the very long instruction word executed at the *i*-th time slot of N. Let n_j be an instruction in N_i. Then power consumption at the *i*-th time slot P^i is the sum of power consumed by all the executing instructions, either started in this time slot or before is given by

$$P^i = \sum_{0 \le k < i} \sum_{n_j \in N_{i-k}} p^k_{n_j} \tag{1}$$

where k is an integer.

Note that the proposed branch and bound algorithm can be easily extended to work with more accurate power estimation models, because the proposed algorithm does not depend on a particular power estimation technique.

B. Notations

The following notations will be used:

- N is the set of n target instructions to be scheduled.
- T is the set of t time slots, where t is the performance deadline.
- $x^k_i = \begin{cases} 1, & if\, instruction\, i\, is\, allocated\, to\, time\, slot\, k \\ 0, & otherwise. \end{cases}$
- X is the set of n variables x^k_i, which equal to 1.
- U is the set of u functional unit types.
- c_j is the number of functional units of type j.
- $a^j_i = \begin{cases} 1, & if\, instruction\, i\, corresponds\, to \\ & \quad functional\, unit\, type\, j \\ 0, & otherwise \end{cases}$
- E is the set of v dependency pairs $< l, m >$, where instruction m depends on instruction l.
- D_i is the number of the execution stages of instruction i.
- P^k is the total power consumption in time slot k.
- M is the average power consumption over all the t time slots.

For the sake of simplicity we introduce the function

$$\varepsilon(x) = \begin{cases} 1 & if\, x \ge 1 \\ 0 & otherwise \end{cases} \tag{2}$$

where x is an integer.

C. Mixed Integer Program

According to the power estimation formula (1), power consumption at each time slot and the average power consumption can be computed by

$$M = \left(\sum_{k=1}^{t} P^k \right) / t \tag{3}$$

where

$$P^k = \sum_{f=0}^{\min(D^{\max}-1, k-1)} \sum_{i=1}^{n} x^{k-f}_i \varepsilon(D_i - f) p^f_i \tag{4}$$

and $D^{\max} = \max_{\forall i \in N}(D_i)$. Then, the power deviation at any given time slot k is computed as

$$PV^k(P^k) = |P^k - M| \tag{5}$$

with the total deviation given by

$$PV(X) = \sum_{k=1}^{t} PV^k(P^k) \tag{6}$$

The optimization problem of instruction scheduling of VLIW architectures for balanced power consumption can be formulated as a mixed integer program **P1** with objective function (7) and constraints (8)-(13).

$$\min \; PV(X) \tag{7}$$

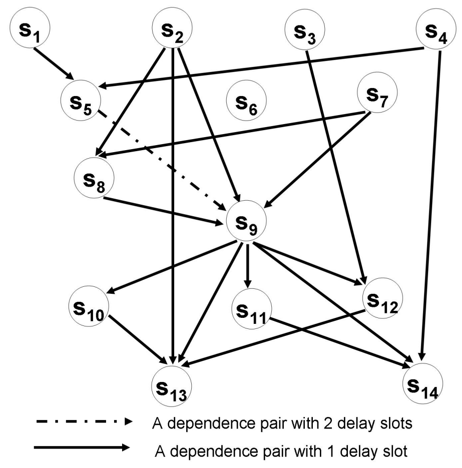

Fig. 2. A data dependence graph for instructions in Example 3.1

subject to

$$X = \bigcup_{i,k:x_i^k=1} \{x_i^k\} \tag{8}$$

$$x_i^k \in \{0,1\} \quad for\, each\ i = 1,...,n; k = 1,...,t \tag{9}$$

$$\sum_{k=1}^{t} x_i^k = 1 \quad for\, each\ i = 1,...,n \tag{10}$$

$$\sum_{k=1}^{t} kx_i^k + D_i - 1 \le t \quad for\, each\ i = 1,...,n; k = 1,...,t \tag{11}$$

$$\sum_{i=1}^{n} a_i^j x_i^k \le c_j, \quad for\, each\ j = 1,...,u; k = 1,...,t \tag{12}$$

$$\sum_{k=1}^{t} kx_m^k - \sum_{k=1}^{t} kx_l^k \ge D_l \quad \forall <l,m> \in E \tag{13}$$

The objective function (6) is given as a sum of the power deviations defined by (5) over all time slots. The set X is called a schedule. Condition (10) states that each instruction can only be issued once. Finally, (13) are dependency constraints, (11) are the deadline constraints and (12) are the resource constraints.

For the program **P1**, we need to find a feasible schedule X that minimize the value of the function $PV(X)$ with all the constraints satisfied. This problem is NP-complete. In the next section, we describe a branch and bound method that allows us to find a optimal schedule within a reasonable amount of time.

III. Branch and Bound Algorithm

Starting with an initial schedule X_1, a sequence of schedules X_r are generated until the optimal schedule is found. The initial schedule X_1 can be one produced through a conventional list scheduling algorithm.

The branch and bound algorithm requires three important elements:

1) the lower bound estimate for a schedule X_r,
2) the rules for branching to a set of new schedule X_s from a certain schedule X_r, and
3) the rules for selecting a certain schedule X_r from the produced schedule pool.

A. Lower Bounds

Given the current schedule X_r, its lower bound is estimated to check if a better schedule may be found in successors that can be generated from the schedule X_r. If the lower bound for X_r is larger than the current objective value, then X_r is removed from the schedule pool.

Let P_{total} denote the sum of power consumption over all the time slots. It can be expressed as

$$P_{total} = tM \tag{14}$$

Let

$$P_u^k = \sum_{f=0}^{\min(D^{\max}-1,k-1)} \sum_{i:x_i^{k-f} \in U_r} x_i^{k-f} \varepsilon(D_i - f) p_i^f \tag{15}$$

denote the power consumption in time slot k due to all the rescheduled instructions of the schedules from X_1 to X_r.

Now we introduce two constraints

$$\sum_{k=1}^{t} P^k - P_{total} = 0 \tag{16}$$

$$P^k - P_u^k \ge 0 \quad for\, k = 1,...,t \tag{17}$$

Constraint (16) requires that the total power consumption over all time slots in X_r or any of its successors equals P_{total}. Condition (17) guarantees that each P^k must at least be the power consumption P_u^k associated with the rescheduled instructions in each time slot k.

The lower bound for the current schedule X_r can therefore be the minimum of (6) using the values of (14)-(17). Because integer constraints (9) are relaxed, execution of an instruction may be divided and scheduled to multiple time slots. Execution time in each time slot is less than a whole time slot. As a result, power consumption at each time slot P^k can be any values satisfying constraints (16) and (17). Then the obtained minimum of total power deviation (6) will not be a tight lower bound.

We reduce the impact caused by this relaxation by restricting how execution of an instruction may be divided. Among the instructions which are not rescheduled in X_r, suppose the smallest power consumption of a single instruction is $P_{X_r}^{\triangle}$. Execution of an instruction, which has not been rescheduled, is divided into equal smaller slices with a residue if any. Each slice has a power consumption equal to $P_{X_r}^{\triangle}$. In this way, the values of P^k are limited to sums of some smaller execution slices. The algorithm to obtain the lower bound for the current schedule X_r is outlined as follows.

Algorithm 3.1: Lower bound of the current schedule X_r

Let *bound*, $f^k (k = 1$ to $t)$, p and f^W be variables used in this algorithm. The lower bound equals the value of *bound* when this algorithm stops.

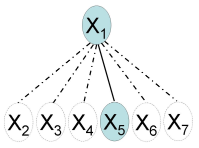

Fig. 3. An example for illustrating branching rules: only a successor X_5 is generated from X_1.

Step 1. $bound = 0$. For $k = 1$ to t, calculate P_u^k according to (15). For $k = 1$ to t, $f^k = P_u^k$.

Step 2. Calculate $P_{X_r}^{\triangle}$, which is the smallest power consumption of a single instruction among the instructions which are not rescheduled in X_r.

Step 3. Given an instruction which has not been rescheduled in X_r, let p_i be its power consumption. $p = p_i$.

Step 4. $p = p - P_{X_r}^{\triangle}$. Let f^W be the smallest among f^k ($k = 1$ to t). $f^W = f^W + P_{X_r}^{\triangle}$. If $p >= P_{X_r}^{\triangle}$, then go to Step 4. Otherwise go to Step 5.

Step 5. If all the instructions which have not been rescheduled in X_r have been processed by Step 4, then go to Step 6. Otherwise go to Step 3.

Step 6. If, for $k = 1$ to t, $f^k > M$, then $bound = f^k - M + bound$. $bound = 2 * bound$. Stop the algorithm.

B. Branching Rules

First, note that the lower bound formulation in Section III-A relaxes the integer constraints (9) and (12). As a result, some instructions may not be scheduled to start at the beginning of a time slot. The impact caused by this relaxation can be reduced by imposing the rule that the larger the power consumed by an instruction, the earlier it is rescheduled. In this way, the lower bound obtained remains tight.

Second, once an instruction i is selected to be rescheduled, we propose four conditions to decide whether rescheduling to time slot j is feasible. Let X_s be the new schedule after i is rescheduled to j. The first condition is that for X_s, the power consumption for time slot j should not be larger than the peak power of the current best schedule, expressed as

$$P_u^j \leq Peak^{X_z} \tag{18}$$

where $Peak^{X_z}$ is the peak power of the recorded current best schedule X_z.

Since constraints (13) only describe direct dependencies in the whole dependence graph, we need to extend them to include all indirect dependence information. Let

$$D_{i,j} = \begin{cases} L_{i,j}\ if\ j\ (in)directly\ depends\ on\ i \\ 0\ otherwise \end{cases} \tag{19}$$

where $L_{i,j}$ is the length of the maximum path from *i* to *j* in the dependence graph, if *j* (in)directly depends on *i*. Then, constraints (13) can then be extended as

$$\sum_{k=1}^{t} kx_j^k - \sum_{k=1}^{t} kx_i^k \geq D_{i,j} \quad \forall i,j \in N,\ if\ D_{i,j} > 0 \tag{20}$$

The second condition is that instruction i in time slot j should satisfy constraints (20).

Given constraints (11) and (20), those time slots in which instruction *i* definitely can not be allocated to can be expressed as

$$\begin{aligned} x_i^k = 0 \quad k = 1, \ldots, D_i^{front}, D_i^{front} \geq 1 \\ x_i^k = 0 \quad k = t - D_i^{back} + 1, \ldots, t, D_i^{back} \geq 1 \end{aligned} \tag{21}$$

where

$$\begin{aligned} \forall i \in N, D_i^{front} &= \max_{\forall h \in N}(D_{h,i}) \\ D_{i,K} &= \max_{\forall j \in N}(D_{i,j}) \\ D_i^{back} &= \max_{\forall j \in N}(D_{i,j}) + D_{i,K} \end{aligned}$$

The third condition is that instruction i in time slot j should satisfy constraints (21).

The fourth condition is that instruction i in time slot j should satisfy constraints (12). If any of these constraints is violated,instruction *i* cannot be scheduled to time slot j. Therefore the branch to reschedule *i* to *j* will not be included in the schedule pool.

These rules help to greatly cut infeasible branches, and thus the size of the produced schedule pool is greatly reduced.

The following simple example illustrates the use of the branching rules.

Example 3.1: Suppose we are given a set of fourteen instructions $\{s_1, s_2, \ldots, s_{14}\}$ and the data dependence graph for these instructions is as shown in Fig. 2. Suppose the target VLIW processor has four functional unit types: integer, float, branch, memory. The available functional units of every types are given as: integer (4), float (2), branch (1), memory (2). The mapping between functional unit types and instructions is given as:

- *integer:* $s_1, s_2, s_3, s_4, s_6, s_7, s_{10}, s_{11}$
- *float:*
- *branch:* s_9, s_{14}
- *memory:* s_5, s_8, s_{12}, s_{13}

Let the initial schedule be the performance optimized VLIW schedule $X_1 = \{x_1^1, x_2^1,$ $x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^4, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$. Thus the total number of time slots needed to execute this schedule is $t = 6$.

First, suppose the sequence of instructions to be rescheduled is

$$\{s_9, s_{14}, s_1, s_2, s_3, s_4, s_6, s_7, s_{10}, s_{11}, s_5, s_{13}, s_8, s_{12}\}$$

according to the rule that the larger the power consumed by an instruction, the earlier it is rescheduled. Then the first instruction to be rescheduled is instruction s_9.

Now there is only a root node X_1 in the branch and bound tree to date. Thus the list of active leaf schedules now is $\ell = \{X_1\}$ and X_1 is selected to branch.

Therefore we reschedule instruction s_9 from the schedule X_1. There would be six successors generated from X_1 if instruction s_9 was rescheduled to each of the six time slots as

TABLE I

EXPERIMENTAL RESULTS ON TRIMARAN'S BENCHMARK PROGRAM

Dim.	Source	Root(mA)	Opt.(mA)	Nodes	TT(sec.)	CPLEX(sec.)	TOpt.
(6,14)	Wave	146.00	127.33	1070	0.03	0.05	40.00%
(11,11)	Fib	229.09	229.09	182	0.01	0.07	85.71%
(9,14)	Wave	224.89	130.67	5965	0.11	0.13	15.38%
(13,13)	Wave	271.38	271.38	156	0.01	0.07	85.71%
(10,22)	Bmm	360.00	119.6	907	0.10	0.19	47.37%
(15,15)	Fib-mem	313.6	313.6	482	0.01	0.10	90.00%
(12,19)	Fir	313.67	201.67	537	0.06	0.10	40.00%
(12,22)	Bmm	386.00	64.00	4108	0.54	2.16	75.00%
(17,16)	Fib-mem	234.35	206.35	386	0.03	0.57	94.74%
(20,21)	Wc	408.60	408.60	3179	0.18	0.22	18.18%
(13,35)	Bmm	710.77	250.62	4269	0.47	0.58	18.97%
(23,22)	Bmm	248.35	220.35	753	0.09	5.43	98.34%
(23,24)	Bmm	474.43	474.43	782	0.06	0.27	77.78%
(25,24)	Bmm	251.52	223.52	3095	0.34	19.29	98.24%
(29,29)	Bmm	608.28	608.28	396	0.04	1.44	97.22%
(31,30)	Bmm	258.58	230.58	1499	0.26	10.04	97.41%
(33,33)	mm-dyn	692.36	692.36	4377	0.85	1.96	56.63%
(35,34)	mm-dyn	261.94	233.94	6914	1.59	108.47	98.53%
						Average:	68.62%

shown in the branch and bound tree in Fig. 3. The six possible successors are:

X_2: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^1, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$
X_3: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^2, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$
X_4: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^3, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$
X_5: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^4, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$
X_6: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^5, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$
X_7: $\{x_1^1, x_2^1, x_3^1, x_4^1, x_5^2, x_6^2, x_7^2, x_8^3, x_9^6, x_{10}^5, x_{11}^5, x_{12}^5, x_{13}^6, x_{14}^6\}$

However, X_2, X_3, X_4, X_6 and X_7 are not feasible successors. Let $i = 9$ in constraints (21), we have

$$\begin{aligned} x_9^k = 0 \quad k = 1, \ldots, G_9^{front} \\ x_9^k = 0 \quad k = 6 - G_9^{back} + 2, \ldots, 6 \end{aligned} \tag{22}$$

Given the data dependence graph (see Fig. 2),

$$G_9^{front} = \max_{\forall h \in N}(G_{h,9}) = G_{1,9} = 3$$

$$\max_{\forall l \in N}(G_{9,l}) = G_{9,13} \quad thus \;\; G_9^{back} = G_{9,13} + D_{13} = 2 + 1 = 3$$

Replace G_9^{front} and G_9^{back} in (22) with the obtained values:

$$x_9^k = 0 \quad k = 1, 2, 3, 5, 6 \tag{23}$$

Thus, X_2, X_3, X_4, X_6 and X_7 are not feasible successors because they violate constraints (23).

Therefore only a successor X_5 is generated from X_1 in the branch and bound tree (see Fig. 3). Thus replace X_1 with X_5 in the list of active leaf schedules $\ell = \{X_5\}$.

C. Selection Rules

Given a pool of possible schedules, a random selection strategy is adopted. This means that we assume that the schedules in the pool has equal probability of leading to the optimal solution.

IV. PERFORMANCE EVALUATION

The VLIW processor we used for evaluating our algorithm is the TMS320C6711 [6] which is a VLIW digital signal processor. The input to our algorithm is a schedule of an instruction block produced by Trimaran's ILP compiler [7]. Our branch and bound algorithm reschedules the one produced through Trimaran to minimize its power variation across time steps. It was implemented in C and runs on an Intel Pentium 4 2.80GHz personal computer with 512MB RAM under Microsoft Windows 2000 for all our experiments. All program instances we used were taken from the benchmarks with Trimaran. Instruction blocks with a wide range of problem dimensions, which are characterized by the number of instructions and the number of total time slots allowed, are selected for our experiments.

Our algorithm is compared with the solutions produced by the ILOG CPLEX90 mixed integer solver directly in terms of the time required to find the optimal schedule. This approach is used by [4]. Table I shows the results of 18 problem instances. For each problem instance, the problem dimension (Dim) indicates the number of time slots and the number of instructions involved in the instruction block. Power variation of the original schedule produced by Trimaran's ILP compiler is shown in the column "Root". The performance of our branch and bound algorithm is indicated by the optimal value of the objective function obtained ("Opt"), the number of nodes visited before the optimal schedule ("Nodes") and the total CPU time in seconds before the algorithm terminates ("TT"). In contrast, the computation times required by CPLEX90 to solve the same problems are shown in the column "CPLEX". "TOpt" is the percentage of computation time required by our algorithm relative to that of using CPLEX90. These results show that our algorithm achieves the same optimized power variation values while it is on average 68.62% more efficient compared with using CPLEX90.

V. Conclusions

In this paper, we presented an efficient VLIW instruction scheduling algorithm that is able to achieve power balance without degrading the speed. The problem is being formulated as a mixed integer program. Our algorithm is a branch and bound algorithm that solves this program efficiently. Its effectiveness is demonstrated through a set of benchmark signal processing programs. The results show that our algorithm achieves the same optimized power variation values as reported in earlier work [4]. However, the required computation time to arrive at the solution in improved by an average of 68.62%. It should be emphasized that our algorithm is independent of the power model for the VLIW processor.

References

[1] "TMS320C62x/C67x power consumption summary," *Texas Instruments, Application Report, SPRA486C*, Jul. 2002.

[2] M. Pedram and Q. Wu, "Battery-powered digital CMOS design," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 10, no. 5, pp. 601–607, Oct. 2002.

[3] T. Simunic, L. Benini, and G. D. Micheli, "Energy-efficient design of battery-powered embedded systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Aug. 1999, pp. 212–217.

[4] H. Yang, G. R. Gao, and C. Leung, "On achieving balanced power consumption in software pipelined loops," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Grenoble, France, Oct. 2002, pp. 210–217.

[5] H. Yun and J. Kim, "Power-aware modulo scheduling for high-performance VLIW processors," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Huntington Beach, California, USA., Aug. 2001, pp. 40–45.

[6] *TMS320C6000 CPU and instruction set reference guide*, Texas Instruments, Oct. 2000, reference Guide, SPRS088E.

[7] Trimaran. Trimaran: an infrastructure for research in instruction-level parallelism. http://www.trimaran.org. A collaborative effort between Compiler and Architecture Research Group at Hewlett Packard Laboratories, IMPACT Group at the University of Illinois and Center for Research on Embedded Systems and Technology at the Georgia Institute of Technology. [Online]. Available: http://www.trimaran.org

Appendix I
TMS320C6711 processor core power variation

TMS320C6711 from Texas Instrument is an 8-issue VLIW processor. Table II obtained from an application report of Texas Instrument [1] shows the impact of instruction scheduling for VLIW processors on power variation. It details the power consumption of the TMS320C6711 CPU and internal memory at 50% high / 50% low activity level and 75% high / 25% low activity level separately. The column "Percentage" is the percentage of the power consumption of CPU and internal memory to the total power consumption including CPU, internal memory, peripherals and I/O. Table III shows the characteristics associated with high activity and low activity for TMS320C6711 separately. In order to illustrate the impact of instruction scheduling for VLIW processors on power variation, we compute the power consumption of CPU and internal memory at 100% high / 0% low activity level and 0% high / 100% low activity level separately. Linear interpolation is used to compute the result. The power consumption of CPU and internal memory at 100% high / 0% low activity level is 1.01W. The power consumption of CPU and internal memory at 0% high / 100% low activity level is 0.45W. Here we can see the difference between eight instructions executed in parallel and two instructions executed in parallel.

TABLE II
Power consumption statistics of TMS320C6711 (150MHz)

Activity Level	**CPU and Internal Memory (W)**	**Percentage**
50% High, 50% Low	0.73	55%
75% High, 25% Low	0.87	57%

TABLE III
Summary of activity levels of TMS320C6711

Activity Level	**CPU Activity Level**	**Program Memory Access Rate**	**Data Memory Access Rate**
High	8 instructions	100%	100% CPU, 50% DMA
Low	2 instructions	25%	12.5% CPU