On Achieving Balanced Power Consumption in Software Pipelined Loops*

Hongbo Yang
Dept of ECE
Univ of Delaware
Newark, DE 19716
hyang@capsl.udel.edu

Guang R. Gao
Dept of ECE
Univ of Delaware
Newark, DE 19716
ggao@capsl.udel.edu

Clement Leung
Dept of ECE
Univ of Delaware
Newark, DE 19716
leung@capsl.udel.edu

ABSTRACT

While a significant body of work in compilers has been devoted to reducing energy consumption in embedded systems, the role of a compiler in harnessing the power variation has not been widely explored. Since sharp power variations across time steps cause power supply noises and degrade reliability of functional blocks, power variation is a design constraint in embedded systems. With the advent of high performance embedded systems and extensive deployment of fine grain clock-gating, reducing variations in power is becoming increasingly important.

This paper studies how compilation techniques, more specifically instruction scheduling, can ameliorate variations in power due to functional units during program execution. By extending our previous work on rate-optimal software pipelining, this paper formulates the problem of constructing a performance-optimal schedule that minimizes power variations as an *integer linear programming* (ILP) problem. The formulation can be solved using an ILP solver. We applied our approach on `SPEC NAS` benchmarks to construct software pipelined schedules that have minimum power variations. The benchmarks are executed on the Wattch power simulator. In comparison to the original (power-unaware) scheduler implemented in the MIPSpro compiler, our power-aware approach generates schedules which have significantly lower power variations while maintaining the same performance. Such schedules have the potential to reduce hardware cost on power delivery in designing embedded systems.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*Code generation, Compilers, Optimization*

*The work is supported in part by DARPA PAC/C program, Silicon Graphics Inc(SGI) and Delaware Research Partnership(DRP) program.

General Terms

Languages, Algorithms

Keywords

Instruction Level Parallelism, Software Pipelining, Power-Aware Compilation

1. INTRODUCTION

Reducing power consumption in embedded systems has attracted a lot of attention from various perspectives. From the compiler design viewpoint, energy reduction and power-performance tradeoffs have been extensively studied [25, 14, 21, 26, 19, 23, 34]. Variation in power during program execution is of equal importance. However this has not been widely explored. In high performance embedded systems, the number of on-chip activities varies drastically on a cycle-by-cycle basis. This translates into a large di/dt, which results in power supply voltage noise. The power delivery system needs to handle the generated noise and it undesirably increases the hardware packaging cost. Large power variation is usually correlated with high peak power and large number of “hot spots”, both are detrimental to functional unit reliability as they adversely affect chip temperature [13].

Ironically, power variations are often exacerbated by commonly used power-saving circuit design techniques such as clock-gating. The rationales behind using clock-gating are that (i) functional blocks are not fully utilized and (ii) a large proportion of power budget is dispensed to the clock network [10]. In a power-aware design where clock-gating is pervasively employed, each functional block is equipped with conditional clock to turn off the clock when the functional unit is inactive. While this helps in reducing power consumption, it brings large power variations as on-chip activities can vary vastly when pipeline stalls and power is directly related to activity factors [27, 32]. Thus, extensive use of power-saving techniques demands effective approaches to address the increasingly important power variation problem.

The work of Grochowski et al., addresses the power variations problem by introducing extra functional blocks to keep power within a certain range [12]. If the instantaneous power is too low (lower than the lower bound of this range), input clock for a subset of functional blocks that are idle is enforced to toggle. In case the instantaneous power is higher than the upper bound, a subset of the extra functional blocks are throttled to reduce power. Our work, which focuses on compiler techniques, complements the hardware mechanism by reducing spikes in the power curve, thus reducing the number of times the compensating and the throttling mechanisms are invoked. The idea is to re-order instructions at compile-time based on an accurate estimate of power consumption for each cycle, with the objective of minimizing overall power variation. Our research is motivated by: (1) a large number of instructions have *schedule slacks* — i.e., each of these instructions can be scheduled in one of the many time steps without degrading the performance of the schedule; (2) modern processor design tends to make functional units fully pipelined to reduce *structural hazard*, Itanium being one such example [17]. Thus it is possible to delay the issue of certain non-critical instructions and obtain a schedule with balanced power consumption without bringing performance degradation. The generated schedule keeps performance optimality while having lower peak power, fewer hot spots, and lower di/dt. Henceforth such a schedule is referred to as a *power-friendly* schedule.

Software pipelining [20] is an effective technique used to exploit *instruction level parallelism* across iterations in a loop. It plays an important role in optimizing program performance on superscalar and VLIW processors. Recent research also shows that it reduces energy consumption of embedded applications [31]. With the advent of embedded processors of higher performance [22], software pipelining is expected to be prevalently used to compile embedded applications. In the past, resource constrained software pipelining has been studied extensively and a number of *modulo scheduling* algorithms [8, 15, 20, 29] have been proposed. This paper extends our previous works on achieving rate-optimality in software pipelining [3] to obtain a new power-aware rate-optimal software pipelining method for VLIW architectures. The proposed approach can minimize power variations in a software pipelined schedule without sacrificing its performance or increasing the energy consumption. In particular, this paper addresses the following problem:

Problem P Given a loop $\mathcal{L}$ and machine architecture $\mathcal{M}$, generate a rate-optimal software pipelined schedule that minimizes $\sum_t |P_t - P_{avg}|$, where P_t denotes the power consumed at clock cycle t of the schedule and P_{avg} denotes the average power consumed during the execution of loop $\mathcal{L}$.

This paper presents an integer linear programming (ILP) formulation for the above problem. Although the ILP-based instruction scheduling approach, in general, has higher time complexity than a heuristic algorithm, it is often acceptable in the context of compiling for embedded processors, where the application is compiled once at the vendor site and run repeatedly at the customer site. Further, even in the context of compiling for general purpose processors, as argued in an earlier work [30], ILP based scheduling methods are still useful for the evaluation of (performance) bounds that can be achieved by any heuristic-based method.

We chose MIPSpro to be the base compiler in our experiment on power-aware software pipelining method. MIPSpro is a production quality compiler with a rich set of optimizations developed by SGI targeting MIPS processors. To facilitate power simulation, we modified the MIPSpro code generator and re-engineered the MIPSpro compiler to generate SimpleScalar [6] assembly code. Further we implemented our ILP approach and plugged it in the MIPSpro compiler. By turning it on and off, we generate power-friendly and power-unaware schedules respectively. Power simulation is then performed using the Wattch simulator [5] with our enhancements to model power consumption in functional units more precisely. We compared our approach with the existing scheduling method in MIPSpro. The results reveal that our approach can generate schedules which reduce power variations, hot-spots, and peak power.

The rest of this paper is organized as follows: Section 2 presents an example that illustrates power variations in a software pipelined schedule and how it can be alleviated by instruction re-ordering. In Section 3 we develop an integer linear program formulation for Problem **P**. Our experimental framework is described in Section 4. Section 5 reports our experimental results on `SPEC NAS` benchmarks. Concluding remarks are presented in Section 7.

2. MOTIVATING EXAMPLE

In this section we motivate the problem of generating power-friendly software pipelined schedules which retain performance optimality with the help of an example.

Consider a hypothetical VLIW architecture with 1 adder and 2 multipliers. The execution latencies, the number of stages, and the number of FUs available are shown in Table 1. The adder takes one cycle to complete an add operation. The two multipliers are symmetric and fully pipelined, and have a latency of 2 cycles. To simplify our discussion, we do not restrict the number of instructions that can be issued in a single cycle. In modeling power consumption, we assume that fine grain clock-gating is employed, i.e., input clock is gated on a stage-by-stage basis. This is in compliance with the state-of-the-art power-conscious design in practice [11, 12].

We assume that the adder consumes 1 unit of power when it is active. For the multiplier, each of its stages consumes 2 units of power when it is active. We assume that an inactive stage does not consumes any power[1]. These architectural assumptions are just used to illustrate the effectiveness of instruction re-ordering. Our formulation neither depends on these assumptions nor the specific latency/power consumption values used.

Table 1: Target Machine Configuration

Type of functional unit	Number of instances	Pipeline stages	Active power per stage
Adder	1	1	1
Multiplier	2	2	2

Consider the DDG shown in Figure 1, where there are three instructions (two multiply and one add instructions) in the loop. Table 2 gives a feasible software pipelined schedule for the DDG. The repetitive pattern for the software pipelined schedule is shown in the shaded columns. The power consumed by the functional units at each time step is also shown in the last row of Table 2. As shown in the table, in cycle 3, instructions $s0$ and $s2$ are under execution and are in the first stage of the two multipliers while the adder is idle. Thus the total amount of power consumed is 4. At

[1]The amount of power consumed by a pipeline stage when it is inactive (referred to as *leakage power*) is assumed to be zero, only to simplify our discussion. The formulation presented afterward does not depend on this assumption.

cycle 4, again, the active instructions are $s0$ and $s2$ (in the second stage of the two multipliers), and the total amount of power consumed is 4.

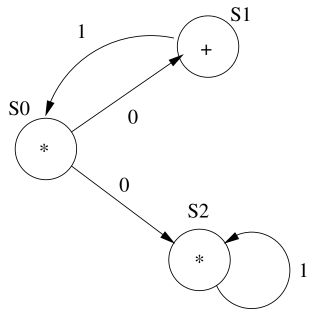

Figure 1: A sample DDG

Table 2: A feasible schedule for the DDG in Figure 1

Iteration	Time Steps						
	0	1	2	3	4	5	6
0	s0		s1	s2			
1				s0		s1	s2
2							s0
Running instruction	s0	s0	s1	s0,s2	s0,s2	s1	s0,s2
Power	2	2	1	4	4	1	4

For the schedule shown in Table 2, the peak power dissipated is 4, and this happens at time steps 3 and 4 (or time steps 0 and 1 in the repetitive pattern). In contrast with the two hot spots which consume 4 units of power, the power consumed at time step 5 (or time step 2 in the repetitive pattern) is pretty low. An interesting question is that although the energy consumption (related to average power) for a rate-optimal schedule cannot be altered in the context of this paper, is it possible to derive a more power-friendly schedule that reduces the power consumed in cycle 3 or 4 at the expense of increasing the power consumed in cycle 5? The answer is yes. Another schedule which keeps rate optimality, with the same energy consumption but reduced hot spots is shown in Table 3. The power-friendly schedule shown in Table 3 has the following advantages over the earlier schedule (shown in Table 2): (i) the number of hot spots are reduced, the peak power is consumed only in cycle 4 (or cycle 1 in the repetitive pattern); (ii) the *step power* — the difference in power consumed between two consecutive time steps — is reduced; the maximum step power is 3 for the schedule shown in Table 2 and it happens twice (between cycles 4 and 5 and cycles 5 and 6) whereas, in the power-friendly schedule, the maximum step power is 2 and it happens only once (between cycles 3 and 4). Another potential benefit of a power-friendly schedule is that it can have a lower peak power. This will be studied in experiments in Section 5.

Table 3: A power-friendly schedule for the DDG in Figure 1

Iteration	Time Steps						
	0	1	2	3	4	5	6
0	s0		s1		s2		
1				s0		s1	
2							s0
Running instruction	s0	s0	s1	s0	s0, s2	s1,s2	s0
Power	2	2	1	2	4	3	2

Comparing the two schedules shown in Tables 2 and 3, we observe that the power-friendly schedule is obtained by placing an instruction that has slack ($s2$ in this case) at a cycle where power consumption is low. The following section presents our approach to achieve a balanced power consumption by exploiting these schedule slacks.

Before proceeding to our study, a few remarks are in order. While other related works address power consumption for a wide range of components including cache [1], bus [25, 21], issue logic [9], etc., our approach focuses on functional units. The reason is that functional units consume a large proportion of power. This is especially the case in VLIW architectures which have a simple instruction issue mechanism. Also the increasing issue-width of a VLIW processor — to obtain higher instruction level parallelism — often requires a large number of functional units which further stresses the importance of power consumption by functional units. This is evident from the large body of related research on reducing the power consumption of functional units [34, 18, 28, 7]. Last, since our approach does not exclude the adoption of power reduction techniques targeting other components, it is even more rewarding to apply our approach in combination with those techniques.

3. INTEGER LINEAR PROGRAMMING FORMULATION

In this section we extend the integer linear programming formulation for software pipelining to model power consumption.

Let the number of nodes in the DDG be N and let II be the initiation interval. In this paper we consider only modulo scheduling. In modulo scheduling, the schedule time of operation i in the jth iteration is given by:

$$t_{i,j} = t_i + j * II$$

where t_i is the schedule time of operation i in the first iteration. We use the vector Γ of length N to represent the schedule time of the operations in the first iteration. The vector Γ comprises variable $t_0, \cdots, t_{N-1}$. We use an $II \times N$ matrix A to represent the repetitive pattern of instructions in the modulo schedule. In matrix A, element $a_{t,i} = 1$ if and only if instruction i is issued at time step t in the repetitive pattern. It is easy to see that

$$a_{t,i} = 1 \Longleftrightarrow t_i \bmod II = t$$

Further Γ and A are related by the following equation.

$$\Gamma = II \cdot \kappa + A^T \times [0, 1, \cdots, II-1]^T \qquad (1)$$

where κ is an N-element vector whose ith element is k_i. Equation 1 implicitly defines k_i as

$$k_i = \left\lfloor \frac{t_i}{II} \right\rfloor$$

Since each instruction can only occur once in the repetitive pattern, the following equation must be satisfied for a legal schedule:

$$\sum_{t=0}^{II-1} a_{t,i} = 1 \text{ for all } i \in [0, N-1] \quad (2)$$

Let R_r denote number of resources of type r and $\zeta(r)$ denote the set of instructions in the loop that use resources of type r. Resource constraint in the repetitive pattern is captured by the following inequality:

$$\sum_{i \in \zeta(r)} a_{t,i} \leq R_r \text{ for all } t \in [0, II-1], \text{ and for all } r \quad (3)$$

Inequalities 1 to 3 formulate the rate-optimal software pipelining problem [3]. Next we study power consumption and its formulation which are contributions of this paper. To model the power consumption of pipeline stages of a power-conscious design where stage level clock-gating is employed, we need to capture activities of each stage precisely. For this purpose, we introduce an $II \times N$ matrix U whose elements can take a value 0 or 1. Each element $u_{t,i}$ is 1 if instruction i, at time step t, is in any of the stages of the pipelined functional unit dedicated for its execution. The variable $u_{t,i}$ is defined by matrix A as:

$$u_{t,i} = \sum_{\tau=0}^{d_i-1} a_{t-\tau,i} \text{ for all } t \in [0, II-1] \text{ and } i \in [0, N-1] \quad (4)$$

At time step t, the number of instructions being executed in functional units of type r is

$$\sum_{i \in \zeta(r)} u_{t,i}$$

Let C_r denote the power consumed by each stage of a functional unit of type r. The power consumed by all instances of type r functional units at time step t is

$$C_r \cdot \sum_{i \in \zeta(r)} u_{t,i}$$

Thus the total amount of power consumed at time step t can be represented by

$$\sum_r \left(C_r \cdot \sum_{i \in \zeta(r)} u_{t,i} \right)$$

Next we calculate the average power of the repetitive pattern. An easy way to get the average power is to average the power consumed over the repetitive pattern. However, we note that with our power model, the average power consumed for a loop is constant regardless of the schedule. Hence P_{avg} for the formulation can be obtained from any (rate-optimal) software pipelined schedule by the following formula:

$$P_{avg} = \frac{\sum_r \left(C_r \sum_{i \in \zeta(r)} d_i \right)}{II}$$

The software pipelined schedule desired in this paper is the one which has smooth power profile, which means that the actual power consumed at each clock cycle is as close to the average power as possible. The difference between the actual power consumed at time step t and the average power is:

$$\sum_r \left(C_r \cdot \sum_{i \in \zeta(r)} u_{t,i} \right) - P_{avg}$$

Hence our objective function is to minimize the value of the following expression:

$$\sum_{t=0}^{II-1} \sum_r \left(C_r \cdot \sum_{i \in \zeta(r)} u_{t,i} \right) - P_{avg}$$

To formulate this as an integer linear programming problem, we introduce two inequalities as follows:

$$D_t \geq \sum_r \left(C_r \cdot \sum_{i \in \zeta(r)} u_{t,i} \right) - P_{avg} \quad (5)$$

$$D_t \geq P_{avg} - \sum_r \left(C_r \cdot \sum_{i \in \zeta(r)} u_{t,i} \right) \quad (6)$$

and define the objective function as:

$$\min \sum_{t=0}^{II-1} D_t \quad (7)$$

Although we do not explicitly specify an upper bound for D_t, the objective function (Equation 7) ensures that D_t for all t in $[0, II-1]$ will get its smallest permissible value. Thus D_t must be equal to the maximum of the right hand sides of Inequalities 5 and 6.

Thus a software pipelined schedule that has balanced power profile can be achieved by solving the integer linear programming problem with the objective function 7 under the constraints 1 – 6.

4. EXPERIMENTAL FRAMEWORK

We implemented our power-aware software pipelining approach in the MIPSpro compiler, a production-quality compiler developed by SGI targeting MIPS processors. To conduct our experiments, the most relevant functional modules of MIPSpro compiler are the software pipelining module [30] and the integrated global and local scheduling(IGLS) module [24]. A given data dependence graph (DDG) is scheduled by the software pipelining method, if the compiler deems the computation as software-pipelineable, or by the IGLS method otherwise. To evaluate the effectiveness of our approach against performance-oriented but power-unaware ones (both IGLS and software pipelining fall in this category), we plugged-in our ILP approach for both software pipelining and IGLS scheduler.

To facilitate power simulation, we re-engineered the MIPSpro compiler to generate SimpleScalar [6] assembly code. The re-engineered compiler is semantically correct, and uses the SimpleScalar's machine configuration information to generate efficient schedules. We chose Wattch [5] simulator to obtain power consumption. Wattch is an activity-based power simulator built on top of SimpleScalar. However, the current Wattch implementation makes an over-simplified assumption that a functional unit is accessed by the instruction executing on it only for a single cycle. We enhanced Wattch to model the pipeline stages for each functional unit precisely.

Another important modification required is that SimpleScalar simulates superscalar processors while we want to study the effect of our software pipelining method on VLIW architectures, where run-time instruction issue behavior strictly conforms to the schedule generated by the compiler. We used the annotation field of SimpleScalar instruction word for this purpose. The annotation field has sixteen bits and it is intended to be used to extend ISA when needed. We used the annotation field to carry issue cycle for each instruction designated by the compiler. Compiler annotates the assembly instructions according to the schedule it constructed. While simulating its execution, simulator observes the annotation bits and only instructions that have this issue cycle value in the annotation field can be issued in a given cycle.

In our experiments we assumed a VLIW architecture which can issue up to 4 instructions in each cycle. The architecture has 4 integer ALUs, 2 integer MULTs, 4 floating point ALUs and 4 floating point MULTs. The **ALU** unit carries out simple arithmetic calculations like `add` and `sub`, etc., and the **MULT** function unit carries out more complicated arithmetic operations like `mul`, `div`, and `sqrt`. In our approach we optimized the power variations in floating point functional units only since they consume more power than integer units for the benchmarks we studied. However, our approach is not restricted to any particular type of functional units.

We use the `NAS` kernel benchmarks from `SPEC` floating point benchmark suite. We report performance results for 5 of the 7 benchmarks which can be compiled by our reengineered MIPSpro compiler and run correctly on the simulator. The benchmarks used are `mxm` (matrix multiply), `cfft2d` (complex 2D-FFT), `cholsky` (Cholsky solver for systems of linear equations), `btrix` (vectorized block tridiagonal solver) and `vpenta` (inverts 3 pentadiagonal matrices simultaneously).

The benchmarks are compiled at optimization level `-O2` and `-O3`, with software pipelining explicitly enabled. With this setting, most optimization modules, including aliasing analysis, strength reduction, common subexpression elimination, global and local register allocation, loop unrolling and integrated global and local scheduling, are invoked. At optimization level 3, loop nest optimization is turned on, in which a precise data dependence analysis is performed. For our ILP implementation, the data dependence graphs (DDGs) of the basic blocks for software pipelining or IGLS are extracted and an ILP problem is formulated for each DDG. The ILP problem is solved by a commercial ILP solver, CPLEX [16], and the resulting schedule is fed back into the compiler; the compiler finishes the remaining steps of compilation process and generates the assembly code according to the schedule decision made by CPLEX. It should be noted here that in the MIPSpro compiler, if a DDG is not software-pipelineable, it will be scheduled by IGLS more than once during the compilation, each time with a different objective. Our ILP approach is applied at the last invocation of IGLS (i.e., after global and local register allocation), since at this pass the instructions resemble the final assembly code more closely.

5. EXPERIMENTAL RESULTS

Our ILP formulation is seeking for a schedule which has smooth power variation, so that peak power, number of hot spots, and average di/dt are reduced. In this section we present our experimental results which address the following questions:

(i) To what extent is the number of hot spots (number of cycles at which the power consumption is relatively high) reduced without increasing average power or execution time?

(ii) Quantitatively, what is the impact of accurate data dependence analysis on the effectiveness of our approach?

(iii) To what extent is di/dt reduced?

5.1 Reduction of Hot Spots and Peak Power

We use P_p to denote the peak power for a particular program. To describe hot spots more precisely, we define a function $T(x)$ which represents the number of cycles at which the instantaneous power is higher than x during the execution of a particular program. Hot spots are those cycles in which the instantaneous power is greater than certain threshold power. Thus the number of hot spots is denoted by $T(P_{Th})$ where P_{Th} is the threshold power. Definition of P_{Th} may differ from processor to processor. For an overall comprehension of power distribution throughout program execution, Table 4 shows the proportion of $T(x)$ to total execution cycles for each benchmark studied, where x ranges from P_p to $0.6 \times P_p$. These numbers are obtained when the benchmark programs are compiled with the original MIPSpro (power-unaware) scheduler. From the table, we can see that in 3 of the applications (`btrix`, `fft` and `vpenta`), in more than 20% of the execution cycles, the instantaneous power is more than 60% of the peak power. In `fft`, the instantaneous power is greater than 90% of the peak power in more than 10% of the execution cycles.

Table 4: Power Distribution for `SPEC NAS` Benchmarks

Benchmark	$T(P_p)$	$T(0.9\times P_p)$	$T(0.8\times P_p)$	$T(0.7\times P_p)$	$T(0.6\times P_p)$
`btrix`	0.2%	0.8%	12.2%	12.2%	30.6%
`cholsky`	0.2%	1.5%	1.7%	1.7%	4.2%
`fft`	6.8%	10.6%	15.7%	23.1%	23.1%
`mxm`	2.1%	3.7%	6%	9.8%	18.2%
`vpenta`	0.1%	1.3%	10.5%	10.5%	35.7%

Table 5 shows the effectiveness of our approach where P_{Th} ranges from P_p to $0.6 \times P_p$. Each column shows the percentage reduction in $T(P_{Th})$ when power-aware scheduling is turned on compared to that when the program is compiled with the original MIPSpro compiler (with power-unaware scheduler). Note that the peak power of a program with the power-aware scheduling method turned on may be different from that of the program when a power-unaware scheduling method is used. In the following discussion, P_p always represents the peak power of a program with a power-unaware scheduler (i.e., the original MIPSpro scheduler). Our ILP approach successfully reduces the peak power for `mxm` and `vpenta`. This is shown in column 2 of Table 5 which report 100% reduction under $T(P_p)$. For the other three benchmarks, our approach does not reduce peak power, although it successfully reduces the number of cycles in which the peak power is consumed, by 60.6%, 30% and 73.2% respectively for `btrix`, `cholsky`, and `fft`. Table 5 also shows the reduction in the number of cycles at which instantaneous power is higher than $x \times P_p$, for values of x ranging from 0.7 to 0.9, for the various benchmarks. The negative values (of percentage reduction) in the last few columns of this table may, at first sight, appear as a negative result. But this is not the case and is reasoned as follows. As the number of execution cycles remains unchanged (i.e., $T(0)$ remains the same) for power-aware and power-unaware schedules, a decrease in $T(x)$ for larger values of x will inevitably increase $T(x)$ for some lower x. This shows as negative reduction in $T(x)$ for low values of x. More specifically, for `btrix`, `cholsky`, `mxm` and `vpenta` the increase in $T(x)$ happens for $x = 0.6 \times P_p$, while for `fft` it happens for $x = 0.8 \times P_p$. Detailed study reveals that `fft` has higher average power consumption than the other benchmarks. This can be inferred to some extent from the distribution of hot spots with the original scheduler (refer to Table 4).

Table 5: Reduction of Hot Spots on SPEC NAS Benchmarks

Benchmark	$T(P_p)$ Red.	$T(0.9\times P_p)$ Red.	$T(0.8\times P_p)$ Red.	$T(0.7\times P_p)$ Red.	$T(0.6\times P_p)$ Red.
btrix	60.6%	49.2%	25.7%	25.7%	-17.1%
cholsky	30%	2.8%	0.4%	0.4%	-77.7%
fft	73.2%	31.1%	-9.2%	-13.2%	-13.2%
		$T(0.8\times P_p)$ Red.	$T(0.7\times P_p)$ Red.	$T(0.6\times P_p)$ Red.	$T(0.5\times P_p)$ Red.
mxm	100%	83.4%	65.1%	31%	-22.1%
vpenta	100%	47.2%	47.2%	2.4%	-9.4%

5.2 Impact of Dependence Analysis on Generating Power-Friendly Schedule

Dependence analysis is crucially important for instruction scheduling (including software pipelining). Accurate dependence analysis [2] at compile-time renders data dependence graphs of fewer dependence pairs, thus giving the instruction scheduler more freedom in scheduling instructions. With reduced (true) dependences, our ILP approach is more likely to generate a schedule with a better objective value. Table 6 depicts power distribution over total execution cycles for all benchmarks compiled at `-O3`, at which accurate compile-time dependence analysis is turned on, and with the original MIPSpro (power-unaware) scheduler. Table 7 shows the effectiveness of applying our approach by presenting the percentage reduction in $T(x)$ compared with the original scheduler, where x ranges from P_p to $0.6 \times P_p$.

Benchmarks `cholsky` and `vpenta` benefit a lot from our approach if compiled with accurate data dependence analysis. For `vpenta`, the power-unaware scheduler generates sporadic power spikes which consumes power higher than $0.7 \times P_p$. Although this happens only for 0.2% of the execution time, as discussed in [12], such spikes put additional burden on the circuit design to address them, which, in turn, can result in increased average power or reduced performance. Our power-aware scheduler generates a better schedule and completely removes them. For `cholsky`, at -O2 our ILP approach is only effective at reducing $T(P_p)$ by 30% and the proportion of $T(P_p)$ to the number of total execution cycles is 0.2%. However with `-O3` optimization, our approach reduces $T(0.7 \times P_p)$ by 40% and the proportion of $T(0.7 \times P_p)$ to the total execution cycles is 18.7%. Lastly, the code generated with `-O3` optimization has undergone loop transformations, it is radically different from that generated with `-O2`. This makes the effectiveness of our ILP approach vary across benchmarks. However, comparing Table 5 and Table 7 we can conclude that there is a large potential of achieving balanced power consumption applying our ILP approach with accurate data dependence analysis.

Table 6: Power Distribution for SPEC NAS Benchmarks Compiled with -O3 Optimizations

Benchmark	$T(P_p)$	$T(0.9\times P_p)$	$T(0.8\times P_p)$	$T(0.7\times P_p)$	$T(0.6\times P_p)$
btrix	2%	5.8%	16.5%	16.5%	36.3%
cholsky	0.3%	11.8%	18.7%	18.7%	21.5%
fft	5.6%	9.9%	15.6%	22.9%	22.9%
mxm	7.5%	3.9%	5.3%	7.5%	11.5%
	$T(P_p)$	$T(0.7\times P_p)$	$T(0.6\times P_p)$	$T(0.5\times P_p)$	$T(0.4\times P_p)$
vpenta	0.1%	0.2%	6%	9.8%	20.1%

Table 7: Reduction of Hot Spots on SPEC NAS Benchmarks with -O3 Optimizations

Benchmark	$T(P_p)$ Red.	$T(0.9\times P_p)$ Red.	$T(0.8\times P_p)$ Red.	$T(0.7\times P_p)$ Red.	$T(0.6\times P_p)$ Red.
btrix	69.4%	21.5%	18.2%	18.2%	-10.1%
cholsky	52.3%	68.5%	40.1%	40.1%	-12.2%
fft	100%	61.2%	-0.1%	-23.6%	-23.6%
mxm	100%	100%	83.9%	73.8%	-15.9%
		$T(0.7\times P_p)$ Red.	$T(0.6\times P_p)$ Red.	$T(0.5\times P_p)$ Red.	$T(0.4\times P_p)$ Red.
vpenta	100%	100%	92.9%	34.8%	-67.8%

5.3 Reduction of Step Power

Step power, the difference between the instantaneous power consumed between two consecutive time steps, must be within a specific range to make functional units work properly. Extra circuitry (power delivery system) is needed if the step power exceeds the largest permissible threshold. Our objective function also reduces step power (related to di/dt), thus relieving the burden on the power delivery system. To show how our approach helps, we define the maximum step power in the following way. The step power $SP(i)$ at time step i is defined as

$$SP(i) = |P_{i+1} - P_i|,$$

and the maximum step power $SP_{\max}$ is defined as

$$SP_{\max} = \max\ (SP(i)).$$

Note that $SP_{\max}$ can also change for a program when the power-unaware scheduler is replaced with a power-aware scheduler. We use $SP_{\max}$ to denote the maximum step power when the power-unaware scheduler is used. Lastly, let $TS(x)$ denote the number of cycles in which step power is higher than x.

Table 8 shows the effectiveness of our approach in reducing $TS(x)$, where x ranges from $SP_{\max}$ and $SP_{\max} \times 0.6$. As shown in the table, our approach reduces the maximum step power for all benchmarks except `cholsky`. For `cholsky`, number of cycles at which step power reaches maximum value is largely reduced by 89.8%. For all other applications, the number of cycles in which the maximum step power occurs is reduced completely (100%) by our power-aware scheduling method. Our approach significantly reduces the $TS(0.9 \times SP_{\max})$. As before, the significant reduction in $TS(x)$ for larger values of x results in an increase in $TS(x)$ for lower values of x.

Table 8: Reduction of Step Power on `SPEC NAS` Benchmarks with `-O3` Optimizations

Benchmark	$TS(SP_{\max})$ Red.	$TS(0.9\times SP_{\max})$ Red.	$TS(0.8\times SP_{\max})$ Red.	$TS(0.7\times SP_{\max})$ Red.	$TS(0.6\times SP_{\max})$ Red.
`btrix`	100%	100%	100%	-14.8%	-26.7%
`cholsky`	89.8%	43.5%	43.5%	-3.4%	-3.4%
`fft`	100%	-1.8%	-1.8%	-0.1%	-0.1%
`mxm`	100%	18.7%	18.7%	-2.5%	-2.4%
`vpenta`	100%	100%	-2%	-3%	2%

6. RELATED WORK

Peak power, hot spots, and step power are important constraints in embedded processor design since they are closely related to the chip temperature, reliability, and power supply noise [13, 12, 4, 27]. From a compiler perspective, Toburen proposed an enhanced list scheduling method by adding a new power constraint in issuing instructions [33]. Yun et al extended *iterative modulo scheduling* by adding heuristics from power perspective with the objective of achieving balanced power consumption. Our work differs from theirs in the following aspects: (i) our ILP formulation has the potential to derive optimal results and it can be used to evaluate any heuristic algorithm; (ii) the power model used in our approach is based on a power-conscious design where clock-gating is pervasively used; (iii) we have quantitatively studied the role of precise data dependence analysis on achieving balanced power consumption.

7. CONCLUSIONS

Balanced power consumption is desirable for embedded system design since it has lower peak power, fewer hot spots, and lower step power. This paper studies how to achieve balanced power consumption for software pipelined loops without compromising performance. We have proposed an integer linear programming framework for this problem, based on a power-conscious design where fine grain clock-gating is employed. Our approach is implemented in a re-engineered MIPSpro compiler targeting SimpleScalar processor. We also enhanced the power model for functional units in the Wattch power simulator. Our experimental results reveal that the proposed approach can generate better schedules (more power-friendly) in terms of hot spots, step power, and peak power in comparison with a power-unaware scheduler implemented in the MIPSpro compiler.

8. ADDITIONAL AUTHORS

Additional authors: R. Govindarajan (Indian Institute of Science, Bangalore, 560 012, India,
email: `govind@csa.iisc.ernet.in`) and Haiping Wu (University of Delaware, Newark, DE 19716, USA,
email: `hwu@capsl.udel.edu`).

9. REFERENCES

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proc. of the 32nd Ann. Intl. Symp. on Microarchitecture*, pages 248–259, Haifa, Israel, Nov. 16–18, 1999.

[2] R. Allen and K. Kennedy. Automatic translation of FORTRAN programs to vector form. *ACM Trans. on Programming Languages and Systems*, 9(4):491–542, Oct. 1987.

[3] E. R. Altman, R. Govindarajan, and G. R. Gao. Scheduling and mapping: Software pipelining in the presence of structural hazards. In *Proc. of the ACM SIGPLAN '95 Conf. on Programming Language Design and Implementation*, pages 139–150, La Jolla, Calif., Jun. 18–21, 1995.

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. of the Seventh Intl. Symp. on High Performance Computer Architecture*, Mexico, Jan 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th Ann. Intl. Symp. on Computer Architecture*, pages 83–94, Vancouver, Brit. Col., Jun. 12–14, 2000.

[6] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, Univ. of Wisconsin, Madison, WI, 1997.

[7] J. Casmira and D. Grunwald. Dynamic instruction scheduling slack. In *Proc. of the 2000 KoolChips workshop*, Monterey, California, Dec. 2000.

[8] J. C. Dehnert and R. A. Towle. Compiling for Cydra 5. *J. of Supercomputing*, 7:181–227, May 1993.

[9] D. Folegnani and A. González. Energy-effective issue logic. In *Proc. of the 28th Ann. Intl. Symp. on Computer Architecture*, pages 230–239, Göteborg, Sweden, Jun. 30–Jul. 4, 2001.

[10] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9), 1996.

[11] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proc. of the 35th Design Automation Conf.*, San Francisco, CA, June 1998.

[12] E. Grochowski, D. Ayers, and V. Tiwari. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Proc. of the Eighth Intl. Symp. on High-Performance Computer Architecture*, Cambridge, MA, Feb 2002.

[13] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, Feb 2001.

[14] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic frequency and voltage

scheduling. In *Workshop on Power-Aware Computer Systems (PACS'00)*, Cambridge, MA, Nov 2000.

[15] R. A. Huff. Lifetime-sensitive modulo scheduling. In *Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation*, pages 258–267, Albuquerque, N. Mex., Jun. 23–25, 1993.

[16] ILOG. CPLEX mixed integer solver. http://www.cplex.com.

[17] Intel. *Itanium Processor Microarchitecture Reference for Software Optimization*, Aug. 2000.

[18] J.Seng, E.Tune, and D.Tullsen. Reducing power with dynamic critical path information. In *Proc. of the 34th Annual Intl. Symp. on Micro-architecture*, Austin, TX, Dec 2001.

[19] M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimizations on system power. In *Proc. of the 37th Conf. on Design Automation (DAC-00)*, pages 304–307, NY, Jun. 5–9 2000.

[20] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pages 318–328, Atlanta, Geor., Jun. 22–24, 1988.

[21] C. Lee, J. K. Lee, and T. Hwang. Compiler optimization on instruction scheduling for low power. In *Proc. of 13th Intl. Symp. on System Synthesis*, Madrid, Spain, Sept 20-22 2000.

[22] S. Leibson. Xscale (strongarm-2) muscles in. *Microprocessor Report*, Sept 2000.

[23] T. Li and C. Ding. Instruction balance and its relation to program energy consumption. In *Proc. of Intl. Workshop on Languages and Compilers for Parallel Computing*, Kentucky, Aug 2001.

[24] S. Mantripragada, S. Jain, and J. Dehnert. A new framework for integrated global local scheduling. In *Proc. of the 1998 Intl. Conf. on Parallel Architectures and Compilation Techniques*, pages 167–174, Paris, Oct. 12–18, 1998.

[25] M.Kandemir, N. Vijaykrishnan, M. J. Irwin, W. Ye, and I. Demirkiran. Register relabeling: A post-compilation technique for energy reduction. In *Workshop on Compilers and Operating Systems for Low Power 2000 (COLP'00)*, 2000.

[26] M.T.-C.Lee, V.Tiwari, S.Malik, and M.Fujita. Power analysis and minimization techniques for embedded DSP software. *IEEE Trans on Very Large Scale Integration(VLSI) Systems*, 5(1), Mar 1997.

[27] M. D. Pant, P. Pant, D. S. Wills, and V. Tiwari. An architectural solution for the inductive noise problem due to clock-gating. In *1999 Intl. Symp. on Low Power Electronics and Design Proceedings*, San Diego, CA, 1999.

[28] R. Pyreddy and G. Tyson. Evaluating design tradeoffs in dual speed pipelines. In *Workshop on Complexity-Effective Design*, Goteborg, Sweden, Jun 30 2001.

[29] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Ann. Intl. Symp. on Microarchitecture*, pages 63–74, San Jose, Calif., Nov. 30–Dec.2, 1994.

[30] J. Ruttenberg, G. R. Gao, A. Stouchinin, and W. Lichtenstein. Software pipelining showdown: Optimal vs. heuristic methods in a production compiler. In *Proc. of the ACM SIGPLAN '96 Conf. on Programming Language Design and Implementation*, pages 1–11, Philadelphia, May 22–24, 1996.

[31] T. Simunic, L. Benini, and G. D. Micheli. Energy-efficient design of battery-powered embedded systems. *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 9(1), Feb 2001.

[32] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proc. of the 1998 Conf. on Design Automation (DAC-98)*, pages 732–737, Los Alamitos, CA, Jun. 15–19 1998.

[33] M. C. Toburen. Power analysis and instruction scheduling for reduced di/dt in the execution core of high-performance microprocessors. Master's thesis, North Carolina State University, 1999.

[34] W. Zhang, N.Vijaykrishnan, M.Kandemir, M.Irwin, D.Duarte, and Y. Tsai. Exploiting VLIW schedule slacks for dynamic and leakage energy reduction. In *Proc. of the 34th Annual Intl. Symp. on Microarchitecture*, Austin, TX, Dec. 2001.