Transition Aware Scheduling: Increasing Continous Idle-periods in Resource Units

K. Ananda Vardhan, Y. N. Srikant
nandu@csa.iisc.ernet.in, srikant@csa.iisc.ernet.in

IISc-CSA-TR-2004-12
http://archive.csa.iisc.ernet.in/TR/2004/12/

Computer Science and Automation
Indian Institute of Science, India

January 2004

Transition Aware Scheduling: Increasing Continous Idle-periods in Resource Units

K. Ananda Vardhan, Y. N. Srikant
nandu@csa.iisc.ernet.in, srikant@csa.iisc.ernet.in

Abstract

Static power consumption has become a significant factor of the total power consumption in a system. Circuit level switching techniques reduce static power consumption by exploiting idle periods in processor components and placing them into low power modes or turning them off completely. In this paper, we propose a modified automaton-based list scheduling technique that augments the circuit level techniques by reducing the number of transitions between power modes, thereby increasing the of idle periods in resource units. Our scheduler uses a global resource usage vector and the usage vector of the last issued instruction(s) to select instruction(s) from the ready list such that resource units common to the last issued instruction and the selected instruction(s) are continuously active without creating a transition into low power mode. We estimate the power consumed in resource units using an energy model parameterized by the number of idle cycles, active cycles and transitions. We have implemented our algorithm in gcc and our simulations for different classes of benchmarks using an ARM simulator, with single and multi issue, indicate an average energy savings of 4%-15% in resource units with an average 0.25%-0.8% increase in the number of execution cycles.

1 Introduction

Power consumption in processor cores has become an important concern both in architectural design as well in compiler construction. With the threshold voltages reaching low values, the static power consumption, which is exponentially and indirectly proportional to the threshold voltages, is dominating the dynamic power consumption in all the units of the processor. Static power consumption in functional units like integer arithmetic unit (ALU), floating point unit(FPU), multiplication and accumulation unit (MAC), data cache ports and data memory buses is also significant. Studies show that with the technology trend static power consumption is going to increase linearly whereas active power consumption is going to remain almost constant[22], the former thereby constituting significant portion of the total power consumption.

Static power consumption can be reduced if the number of continuous idle cycles is increased. This provides a good opportunity to hardware driven optimization techniques such as leakage biased bitlines [14] or input vector control to exploit the idle periods of the functional units so that the unit can be placed in low power mode or turned off completely. As indicated by many proposals, the static power consumption in the low power mode is negligible compared to that in active mode of the unit. If the continous active period is increased and also continous idle period is increased then the circuit level switching techniques can be provided with a lot of opportunity. This can be achieved by reducing the number of transitions between the power modes and thereby reduce the other significant constituent of the total power consumption, the transition energy. [13][14] [22] show that transition energy, depending on the controlling technique vary between 1 to 10% of the active energy. Moreover the transition delay ranges from 3-30 cycles in 0.07u technology [22] [14]. The number of transitions between active and idle modes can be reduced so that the saved transition delay can be added to the idle period of the functional unit. In this way, continuous idle periods of units can be increased. This reduces the frequent discharging and charging of the capacitance in the circuits of functional units, thus reducing the power consumed due to transitions. It also increases the chances of staying in an idle period or active period continuously for longer durations therby using the resources more efficiently, both from the performance and power point of view. If only the circuit level switching techniques are employed in order to reduce the static power consumption, transitions can cause certain amount of performance impact too. So reducing the number of transitions lessens the performance impact too. *This paper focuses on increasing the average idle periods that a functional unit can be in and also on reducing the number of transitions that a functional unit can undergo*. Regularization of the usage means changing the pattern with which a functional unit is accessed , so that the functional unit stays in an active state continuously as far as possible without becoming idle and also stays in an idle state continuouly for a significant number of cycles. Regularization of the usage of resource units such as ALU, FPU, MAC and memory port (memory unit) can be achieved through static techniques. Hardware techniques need more complex logic to implement these techniques, whereas a compiler writer has the information about these units and their usage patterns are provided to him through the machine description. The total power consumption in a unit or port is given by.

$$E_{total} = E_{active} + E_{transition} + E_{idle} \tag{1}$$

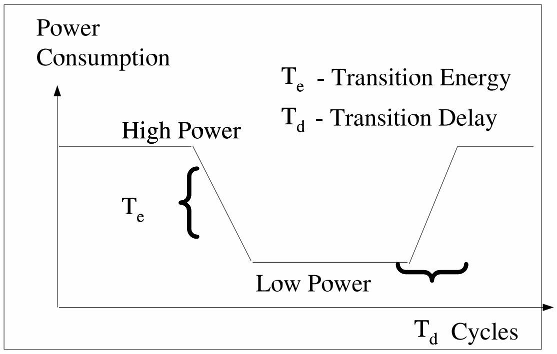


Figure 1: Behaviour of a functional unit. This shows the transition from a high power mode to a low power mode.

We propose to achieve regularization during instruction scheduling.

The remainder of the paper is organized as follows. In section 2 we discuss the background, motivation and the energy model that we use to estimate the energy consumption in the resource unit that we are trying model. In section 3 we discuss the instruction scheduling algorithm, section 4 contains the experimental methodology along with the results we discuss, in section 5 we discuss related work and finally in section 6, conclusions of this work are presented.

2 Background and Motivation

Different hardware techniques have been implemented in today's processors to place an idle circuit in low leakage power mode. These techniques implement different controling mechanisms in order to balance between transition energy and transition delay. More the transition delay, more the performance penalty but less transition delay (sharp transition) can use more transition energy. The break-even time period is calculated as the minimum time that a circuit should be in idle state so that the overhead of transition is nullified. The breakeven time is estimated to be anywhere between 10 to 30 cycles. The figure 1 shows how the behaviour of a unit be when a circuit level switching technique is implemented. When there is a transition from a high power mode to a low power mode, there is extra power loss and also a delay that is incurred. The total transition energy loss can vary between 1 to 10 % of the total active energy consumption. If the transition delay is critical, then the transition energy loss can even be almost 50% of the dynamic active energy consumption. Moreover, the static active leakage energy required for being in the active state without the unit being used is estimated to be almost 5-10 % of the dynamic energy consumption. The trend shows that these components are becoming important factors as technology improves. Some proposals try to reduce the break-even time and transition energy consumption [14]. In order to avoid the implications of tradeoff, new methods to reduce power consumption at architecture level and to use compiler support are essential. Many architectural and compiler research proposals targeted the first component of the equation 1. The number of accesses to memory[19][23] [21], the cache configuration, [5] the number of bit transitions on the buses[1] and the amount of traffic between L1 and L2 caches constitute the majority portion of E_{active}. As we have observed, with the trend, the second and third components are also becoming important and these components have an adversary effect on performance too, i.e., execution time, because of the introduction of transition delays.

Cycle	Instruction	ALU	MUL	RP
i	ld r1, 10[r2]	1	0	0
i+1	add r1,r1,r4	1	0	1
i+2	mul r3,r1,r3	0	1	0
i+3	ld r5,10[r2]	1	0	0
i+4	add r4,r1,0x04	1	0	1
i+5	mul r4,r1,r4	0	1	0
i+6	add r1,r5,r3	1	0	0

Figure 2: This is a valid schedule generated by a traditional scheduler.

Cycle	Instruction	ALU	MUL	RP
i	ld r1, 10[r2]	1	0	0
i+1	ld r5,10[r2]	1	0	1
i+2	add r1,r1,r4	1	0	1
i+3	add r4,r1,0x04	1	0	0
i+4	add r1,r5,r4	1	0	0
i+5	mul r3,r1,r3	0	1	0
i+6	mul r4,r1,r4	0	1	0

Figure 3: Scheduling the instructions with a different heuristic

2.1 Motivating Example

Let us take the code segment in figure 2. The figure shows how different functional units, ALU,MAC and memory ports (RP) are used. Note that functional unit usage in each line does not correspond to demands of the concerned instructon alone. What is shown is the cumulative demand of previous instruction(s) and the current instruction. The load instruction which is scheduled in the cycle *i* will use an ALU in the first cycle and a read port RP in the next cycle. The initial stages of the pipeline are not modeled as the initial pipeline stages are common to all the instructions. Everytime there is a switch from 1 to 0 and 0 to 1, there is a transition from the active mode to idle mode and vice versa. As we see the add instruction which is scheduled in the *i+1* cycle, a mul instruction scheduled in *i+2* are forcing a transition in the ALU. Figure 3 shows how to rearrange these instructions such that the number of transitions is reduced. The two load operations are scheduled one after another, the multiplication operations are scheduled last and the add operations are scheduled closer to each other. This reduces the number of transitions and also increases the continuous idle-period in each functional unit. The schedule in Fig 2 is a valid schedule and is possibly generated by a traditional scheduler such as list scheduler. The heuristics adopted by a traditional scheduler suit well if the aim is to just avoid stalls and improve the total execution time. But in addition to these challenges, the new transition aware scheduler needs to group or schedule together instructions which use the same set of functional units.

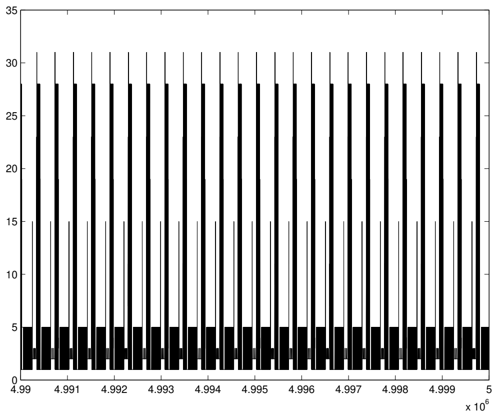


Figure 4: A snapshot of continuous idle periods when optimization for transitions is not introduced. Note the short and small number of continuous idle periods (sparse).Y-axis - Continuous idleperiods

2.2 Dynamic Resource Usage

In this paper a new optimization technique for reducing the number of transitions is presented. We applied our optimization technique on a benchmark *sha*, obtained the continuous idle periods in ALU and transitions from high power mode to low power mode and vice versa. Figures 4 and 5 show plots of these metrics with and without the optimization applied. The figures are the snapshots of the continuous idleperiods in a window of 100000 transitions. The total number of execution cycles with and without optimization differ by only 0.6 %. This indicates that in the window that we captured, both the programs are executing almost same part of the program. We observe short durations and less number of idle cycles in figure 4. The behaviour has been transformed, for the same number of transitions, into long and more number of continous idle cycles after applying the optimization as seen in figure 5. This indicates that there is an opportunity to create extra idle periods in functional units.

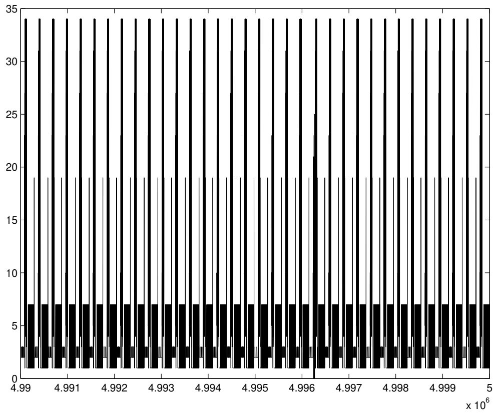


Figure 5: A snapshot of continuous idle periods when optimization is introduced. Note the long and large number of continuous idle periods (dense).Y-axis - Continuous idleperiods.

The different functional units that we consider are ALU, MAC, FPU and a memory port. We assume single issue processors, initially, such as most of the present day embedded processors and a single instance of each functional unit. We also applied the technique to multi-issue processors. These units are exposed to the compiler through a machine description of the target machine. The other units, which are implicitly affected by accessing these units are writebuffer, memory buses, buses that connect MAC and FPU co-processors. *A transition from active to idle mode can be caused if there is an idle period of more than one cycle*. The reason for a functional unit being idle can be a cache miss, branch misprediction, functional unit unavailability being ahead of it, or a dependent instruction blocked in the pipeline. The machine description associates an instruction with a set of functional units and the time for which each functional unit is used. Thus an instruction can access any subset of these components during any cycle of its operation in the pipeline. The problem that we tackle may be stated as follows:

By using the resource usage information available in the machine description, regularize the usage so that the number of transitions of functional units are reduced and the duration of idle period of functional unit is increased We propose changes to a standard automaton based instruction scheduler which, while scheduling a basic block schedules similar instructions so that the usage of different functional units remains constant.

2.3 The Energy Model

An energy model aiming to capture the total power consumption in functional units should take into consideration the following parameters.

1. Number of active cycles and idle cycles of different functional units
2. Static idle energy, transition energy and dynamic energy which varies with the circuit level control techniques that are used.
3. The number of transitions in the functional unit.

In addition to functional units, we wish to model the energy leakage in the read ports and write ports of the data cache whose usage information can also be provided to the compiler. There are many power models suggested in the literature like Wattch [4], Simpower [12] and Butts-Sohi [9]. These models can also be used with the appropriate constants like dynamic voltage settings, activity factors, sensitivity factors and leakage factors, but the model which is appropriate to our present discussion is derived from [13]. The authors consider the model for estimating different policies of their circuit level leakage reduction scheme with various design and technology dependent constants. The model captures the dynamic behaviour of the functional unit through state of each functional unit and change in the access pattern of the unit. The $E_{active}, E_{transition}$ and E_{idle} are given by the following equations.

$$E_{active} = N_{active}(\alpha + (1 - D)p) + N_{active}(\alpha sp + (1 - \alpha)p) \tag{2}$$

$$E_{transition} = N_{transitions}(1 - \alpha + E_{sleep}/E_{active}) \tag{3}$$

$$E_{idle} = N_{idle}sp \tag{4}$$

$$N_{active} = N_{actualactive} + N_{transitions}/2 \tag{5}$$

N_{active} in equation 2, is the number of active cycles in which the unit is in active mode. $N_{transitions}$ is the number of cycles in which the unit is in transition mode. $N_{actualactive}$ is the number of cycles in which the unit is used. N_{idle} is the number of cycles the unit is unused state.

α denotes the activity factor, i.e the fraction of charged cells to discharged cells in the logic. This factor is depenedent on the application. D is the amount of cycle period required to charge a cell p is the leakage factor fraction of static leakage in idle state to the static leakage in the active mode, s is the leakage energy fraction of the dynamic energy E_{sleep} is the energy consumed in the sleep mode. E_{active} is the energy consumed in the active mode. These latter defined six parameters are dependent on the technology and design of the logic. We choose these values from [13] for all the functional units and ports. This model considers the parameters that we have listed above.

3 Scheduling Algorithm

We modify the list scheduling algorithm for basic blocks to achieve reduction in the number of transitions in functional units . The modified problem of the scheduling can be stated as follows.

Given an instruction sequence as a data dependence graph and the resource usage information in the form of a bit vectors, minimize the number of bit transitions across the usage vectors of the scheduled instruction sequence generated.

The algorithm shown in figure 6 provides a mechanism for fetching the global resource usage vector by querying the scheduling automaton [18] that is constructed from the machine description provided. Then it computes the "closeness" between the vectors and assigns priorities to each instruction. The ready list is sorted based on the priorities of the instructions. The scheduler uses the scheduling automaton [18] to chose instructions one at a time from the readylist and schedules them in successive time slots. If no operation is ready then no-ops are introduced till some instruction becomes ready. The scheduler tries to avoid introducing no-ops as in a traditional scheduler.

3.1 Global resource usage vector

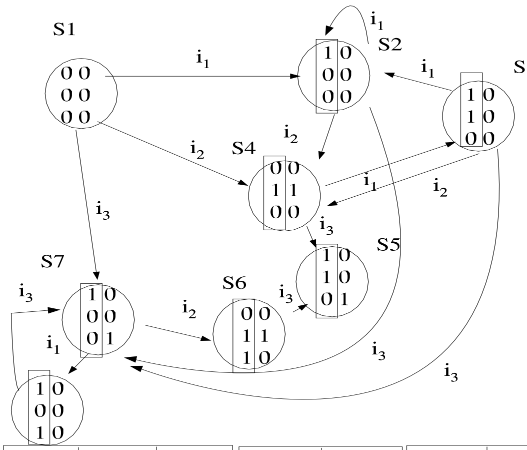
An important part of the algorithm is to obtain the global resource usage vector (GRV), which contains information on the usage of each of the functional units during the present scheduling cycle. Some instructions scheduled in the previous cycles continue to consume different resources during the current cycle, thus changing the global usage in every cycle. To obtain this vector, we modified the pipeline description model that is used to detect structural hazards during scheduling. The pipeline description model can be of any form, ie a resource reservation

```
Input:A Basic block in the form of a data
dependency graph
Output: Scheduled sequence of instructions
```

```
begin
init(Readylist);schedtime=0;
while(!all insns in BB have been scheduled)
 do
    Global=GetGlobal (PresentState)
   while (!ReadyList)
     do
         for(each element in Readylist)
          do
            op=Element of ReadyList;
            Closeness=Close(op,Global);
            ChangePriority(op,Closeness);
          done
            insn=ChooseInsn(ReadyList)
        if(insn)
          ScheduleInstruction(schedtime,insn)
      schedtime=schedtime+1;
      update(Readylist);
     done
 done
end
```

Figure 6: The list scheduler is modified to incorporate the information of global resource usage and compute the closeness of an instruction with respect to the global usage.

table(RRT) or an automaton based model. We use the automaton model proposed by [16] and later enhanced by many authors [18] for two particular reasons:this model implicitly captures the global resource usage that we need during the construction of the automaton and this reduces the scheduling time and it is easy to specify new resources other than units that are used by traditional scheduler. For example we added memory port as another resource component in our description. The number of states increase with the addition of new components in the description. Every state consists of a vector which is the global resource usage, which is updated during the building of the automaton. The modified algorithm to build the automaton is described below. The input is a machine description and it generates the automaton along with the global resource usage information in each of the state. Figure 7 shows a snapshot of an automaton to model a pipeline. The figure shows three instruction classes i_1, i_2 and i_3 with their reservation tables and the automaton built using these classes. The two-column matrix in each state is the resource usage matrix in that state(for two cycles). The first column of this matrix is the GRV for the current cycle. Since our machine model is a single issue model, each state is a cycle advancing state [18] and hence we inspect the second column for any resource conflicts while deciding which instructions are used to issue during the automaton building process. If there are no conflicts, then as in [18] we left-shift the state matrix by one column and OR it with the reservation table of an instruction to generate the matrix for the new state. The rest of the procedure is the same as in [18].



	Cycle 1	*Cycle 2*	*Cycle 1*	*Cycle 2*	*Cycle 1*	*Cycle 2*
Unit 1	1	0	0	0	0	0
Unit 2	0	0	1	1	1	1
Unit 3	0	0	0	0	0	0
	i_1		i_2		i_3	

Figure 7: A scheduling automaton.

3.2 Closeness Heuristic

The next part of the algorithm is to calculate the closeness between the instructions in the ready list and the global resource vector. From the machine description, for each instruction class we associate a resource usage vector (RUV). For example in the figure 7, RUVs are i_1: 100, i_2: 020, i_3: 101s. The closeness of the resource usage vectors indicate how many transitions can be possible if the instruction in observation is scheduled in the next cycle. The resource vector corresponding to each instruction class is a weighted vector where each element corresponds to the number of cycles that a particular unit is used. (i.e sum of the 1's in each row of the RUV). During the every iteration of the scheduling process the global resource vector is OR-ed with the last scheduled instruction's RUV. In this operation we use a bit-version of an RUV that contains 1's for its non zero elements to obtain a resource reservation vector(RRV). By OR-ing the two vectors: GRV and the bit vector acquired from the RUV,i.e. the RRV, the current as well as the future usage of the last scheduled instruction can be captured. Let two vectors are $< i_1, i_2, i_3 > and < j_1, j_2, j_3 >$ respectively. the *Euclidean distance* is calculated as

$$\sqrt{(i_1 - j_1)^2 + (i_2 - j_2)^2 + (i_3 - j_3)^2}$$

The *Euclidean distance* between the global resource and RRVs gives an estimate of how close these two vectors are. If this value for an instruction in the ready list is minimal, then that instruction need to be scheduled as early as possible because it is going to cause less number of transitions.

In a traditional scheduler, the priority that is defined for each instruction before scheduling, is computed using the length of the dependency chain from the instruction node to a leaf of the instruction. In our algorithm we introduce an additional priority, *power consumption priority*, that depends on the closeness factor. This priority for an instruction as high (in the current list), if its closeness is low. We sort the ready list first based on the power consumption priority and then on dependency chain length priority.

3.3 Working Example

Let us take a data dependency graph (DDG) as shown in the figure 8. The nodes are annotated by the instruction id and the functional units they use. *alu,mp* means that the instruction accesses alu in the first cycle and memory port in the second cycle. *mul*2* means that the multiplication unit, MUL is used for 2 cycles, therefore the issue latency is 2 cycles. We assume a single cycle latency for both *alu* and *memory port*. The weight on an edge indicates the result latency. The scheduler output is given in figure 9. The second column indicates the dependency chain priority. The last column indicates the closeness values for the instructions in the readylist in the same order as they appear in eigth column. With a tranditional list scheduler instruction C gets higher priority than E because of its dependency chain length. Thus C gets scheduled earlier than E. But when the closeness heuristic is used, E gets higher priority than C, and thus E gets scheduled earlier and closer to A and B. Similarly, after D is scheduled, with traditional heuristics E gets scheduled, where as with closeness criteria, after instruction D, instruction G is scheduled as they both use the same functional unit. If the GRV is observed (shown in the sixth and ninth columns) we note that the continuous idle cycles of ALU, MP and MUL have increased. The number of transitions in ALU and MUL have reduced from

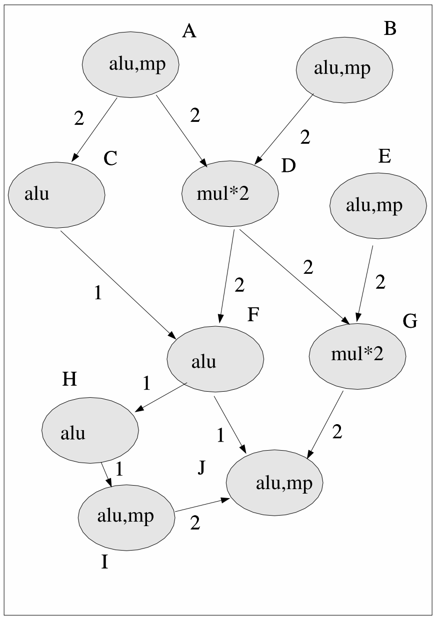


Figure 8: An example DDG

4 to 2, and in memory port from 5 to 3. But with the closeness heuristics there is a no-op in the sequence after D is scheduled. This is because there are no ready instructions in the ready list at that point in time. This is a performance penalty in terms of execution cycles when we use the closeness heuristic. However our simulation studies show that this impact is not so significant when compared to the power consumption benefits.

4 Experimental Results

The scheduling algorithm is implemented in GCC 3.4 in the postpass scheduling phase. The compiler is a cross compiler which generates ARM v5 code. An XScale simulator XTREM [6] which simulates the XScale [3] pipeline and also computes the dynamic power consumption in each of the functional unit is used. The Xscale core has three pipelines one for each of memory, multiplication and ALU operations. The address calculation is done at the ALU stage, whereas multiplication operations are decoded and issued to MAC which is a coprocessor. We have assumed a similar pipeline for floating point operations and have included a floating

Insn	Priority	Cycle	Insn	List	GRV	Insn	list	GRV	Closeness
A	5	i	A	B,E	100	A	B,E	100	0,0
B	5	i+1	B	C,E	101	B	E,C	101	0.1
C	4	i+2	C	D,E	101	E	C,D	101	$1,\sqrt{3}$
D	4	i+3	D	E	010	C	D	101	$\sqrt{3}$
E	2	i+4	E	F	110	D		010	
F	3	i+5	F	H,G	100		G	010	0
G	1	i+6	H	G,I	100	G	F	010	$\sqrt{2}$
H	2	i+7	I	G,J	100	F	H	110	0
I	1	i+8	G	J	010	H	I	100	1
J	0	i+10	J		110	I	J	100	0
		i+11			001	J		101	
		i+12			001			101	

Figure 9: Scheduler Output using tranditional heuristics and Closeness Heuristics

point unit in the simulator. As XScale [1]doesnot have floating point instructions we did not tune the gcc to generate XScale code rather used the ARM ISA. The floating point instructions are decoded and issued to a floating point unit. The simulator collects the number of active and idle cycles and number of transitions that have occured in each unit modeled. The usage of each functional unit is monitored and statistics are collected in each cycle. The benchmarks are chosen from Mibench [7] (adpcm,sha,jpeg,blowfish,susan), MediaBench [2] (mpeg2,gsm), NetBench(dh) and Spec2000(bzip,mesa) [20] suites. The minispec [11] data is used for simulated SPEC benchmarks. The effectiveness of algorithms is determined by the instruction mix in the programs and the available parallelism so that the dependency across different types of instructions makes the scheduler harder to optimize.

We collect metrics for two different schemes. The first scheme is with only circuit level switching techniques being implemented and the second one has circuit switching techniques and the scheduling algorithm being applied on the code. We calculate the power consumption using the energy model described in section 2. The processor characteristics and the parameters of the energy model are shown in table 1. The simulator has been modified to capture the number of cycles during which the unit is in idle state, the number of transitions between power modes and number of cycles during which the unit is active. Two counters are maintained

[0]The Xscale simulator [6] already has code for a floating point unit but it was disabled, we simply enabled it.

Fetch Bandwidth	1
Issue Width	Single issue In order
FUnits	IntALU-1,MAC-1 FPU-1,Memport-2
Data Cache I-Cache	L1 32KB fully associative L1 32KB fully associative
Predictor	128 entry direct 2 bit predictor
Physical Registers	32
α	0.75 0.5 0.25
p	0.05
s	0.01
E_{sleep}/E_{active}	0.001

Table 1: Processor Parameters

for each functional unit, one for idle cycles and the other for transitionst. The total number of active cycles is calculated as the difference between total number of execution cycles and the number of idle cycles. The counters are implemented as depicted as in the finite state automaton shown in figure 10. The transitions occur every cycle depending on the functional unit state. After these values are collected, the energy model is used to calculate the total power consumption of different units.The statistics are collected for various values of activity factor α: 0.25,0.5 and 0.75. We chose these activity factors as [13] indicates that these are reasonable values for most of the cicuit logic that is used to implement functional units.

4.1 Power Consumption Estimation

The energy savings that are plotted are the percentage decrease in the total power consumption of the second scheme with respect to the first scheme, i.e. circuit level switching technique is implemented without out scheduling algorithm being used. The actual power consumption without any circuit level technique is not shown. Figure 11 shows the energy savings for an Integer ALU functional unit for various activity factors. With an increase in activity factor the impact of static leakage energy is seems to be reducing. Most of the simulated benchmarks show a significant impact on the usage of this functional unit. As an ALU is used for integer arithmetic operations as well as for memory operations, bringing these operations closer while scheduling, creates a gap between the usages of the ALU.

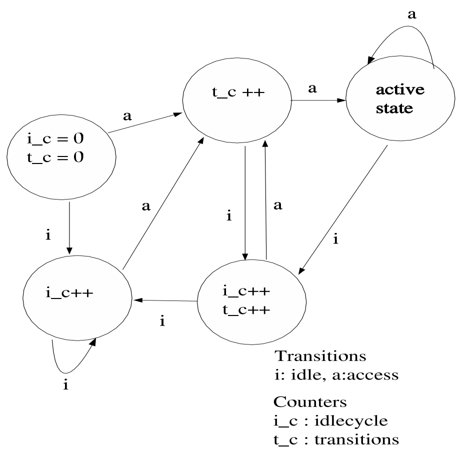


Figure 10: State diagram to depict the functional of the counters to count number of idle cycles and number of transitions in each functional unit.

This can be due to the change in the pattern in which dynamic events such as L1 data cache miss occur. In XScale, the load operation result latency is 2-3 cycles and this causes a dependent instruction to wait for the result. In such a case without our scheduling technique, the traditional list scheduler would make the pipeline stall and this causes a transition in the ALU. But with our technique an opportunity for scheduling a low priority instruction ahead of a high priority instruction exists (based on closeness), so that this results in avoiding a transition. Current circuit logic techniques, which exploit very short intervals of idle periods to save power consumption, may start a transition in the unit from high power mode to low power mode even for short idle periods. In some of the benchmarks like jpeg and mpeg the total number of execution cycles reduce because of this effect but this increases energy consumption. An average of 5% of reduction in the total power consumption is achieved. Figures 12 and 13 show the impact of the scheduling technique on mul and floating point functional units respectively. As observed earlier these units are hardly utilised by most of the benchmarks. This may be due to strength reduction optimization applied on these benchmarks. *mesa, jpeg, gsm* and *fft* show a significant savings in energy with respect to these units as there is a good mix of instructions that use these units in these benchmarks. The total energy savings on average are 7-10 % in MAC unit and 4-7 % in the floating point unit. The remaining benchmarks do not have any significant activity in these units. We have also measured the memory port power consumption for these benchmarks. We have modeled the port of the data cache. Traditionally, data cache usage access patterns

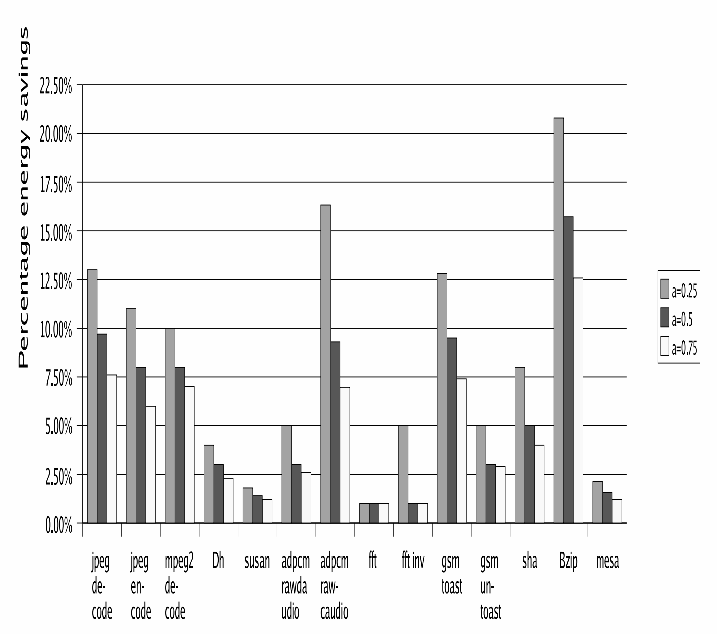


Figure 11: Percentage energy savings ALU

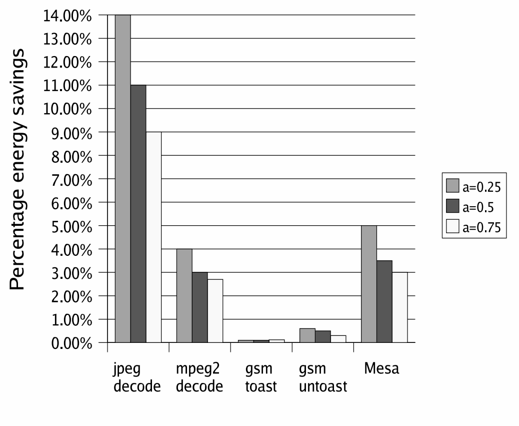


Figure 12: Percentage energy savings in MUL unit

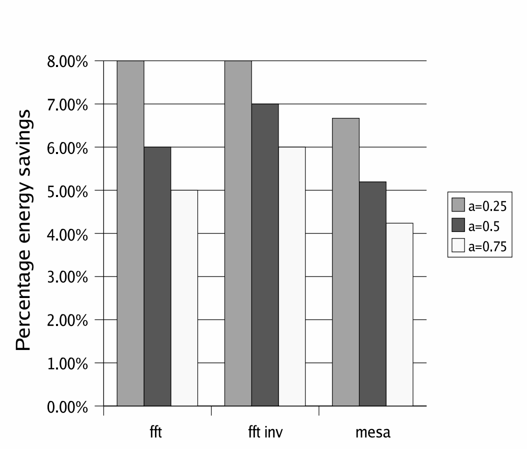


Figure 13: Percentage energy savings FPU

are observed to be non coherent. This is due to the low temporal locality that is exibited by these programs. Figure 14 shows that the amount of energy savings is not much when compared to ALU.

When circuit level switching techniques are implemented, the transition delay would cause the total number of execution cycles to increase. With our scheduling scheme proposed in this paper, the total number of transitions are reduced. This has an impact on the penalty due to the transitions. The performance penalty *reduction* in the total number of execution cycles has also been calculated for both the schemes. The extra execution cycles incurred due to circuit switching techniques is given by

$$N_{actual\_execution\_time} = N_{execution\_time} + N_{transitions} * penalty \tag{6}$$

The penalty can range from 3 to 30 cycles depending on the technology, control characterictis and the circuit level optimization technique. We chose a moderate figure of 5 cycles as penalty due to [22]. The penalty reduction in usage of ALU determines the overall penalty as this is the most used functional unit. This is shown in figure 15

4.2 MultiIssue Processor energy estimation

We have used the heuristic and compiled the benchmarks for a processor with issue width 4. We have assumed a hypothetical inorder machine with multiple ALUs and multiple FPUs. We assume that the functional units are used in some order and the issue logic would issue instructions to the units in this fixed order. This means

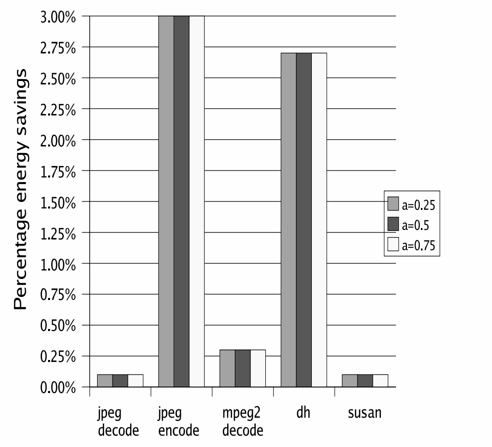


Figure 14: Percentage energy savings in Memory Port

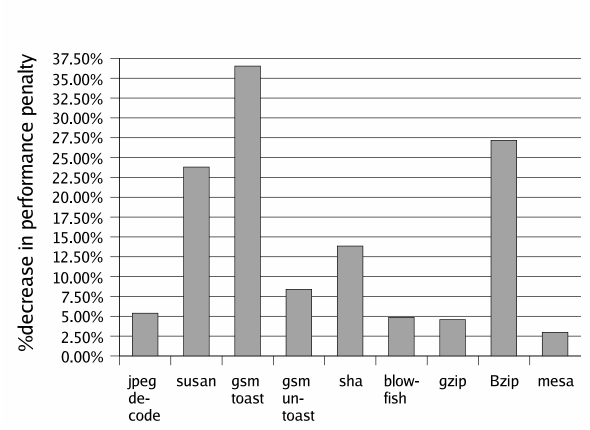


Figure 15: Percentage savings in the performance penalty over a circuit level technique without our scheduling technique being performed

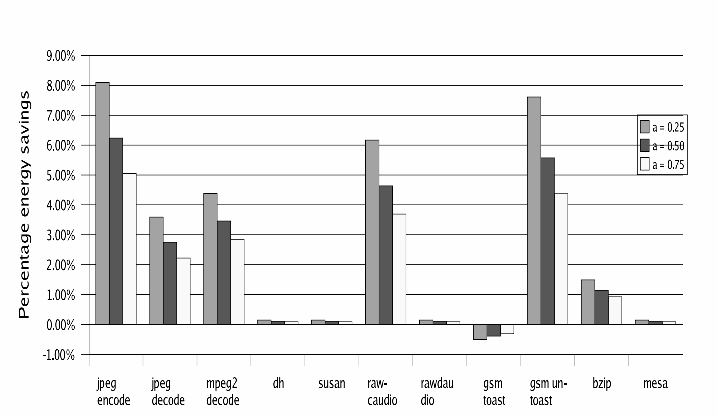


Figure 16: Percentage energy savings in MultiIssue model with Multiple ALUs

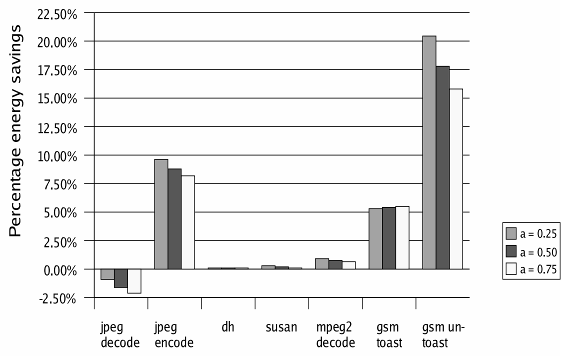


Figure 17: Percentage energy savings in memory port

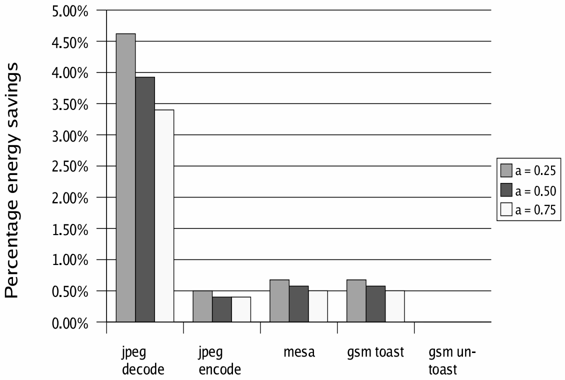


Figure 18: Percentage energy savings in multiplication unit

that with multiple ALUs, ALU1, ALU2, ALU3 and ALU4, an ALU instruction is issued to ALU2 only if ALU1 is busy. N_{idle} and $N_{transitions}$ are measured for all the ALUs and FPUs together. Fetch bandwidth has been increased to 4. In the scheduling automaton not every state is a cycle advancing state. The GRV is the first column in resource usage matrix of the cycle advancing state. The remaining parameters are similar to that of the Xscale processor. The findings, shown in figures 16 through 19 indicate a reduction in the energy consumption. There is a reduction except in benchmarks like *jpeg decode* and *gsm toast*. The energy savings are reduced in these benchmarks due to the reduction in the number of idle cycles The total number of execution cycles in these two benchmarks have come down by 0.5-1.0 % indicating a reduction in the number of idle cycles. The savings in the FPUs are marginal. This is because of the scattered occurance of the FP instructions even after the Closeness heuristic is applied.

Figure 20 indicates the reduction in the penalty compared to a processor with only the circuit level switching techniques and no closeness heuristic scheduling applied. Figure 20 shows that the performance penalty on an average is less than 1% compared to the 3% on a single issue processor. *gsm-toast,jpeg encode* and *decode* exibit reduction in the number of cycles.

5 Related work

There have been a number of proposals to reduce total power consumption through different compiler techniques, targeting different units of the system. In [15] the

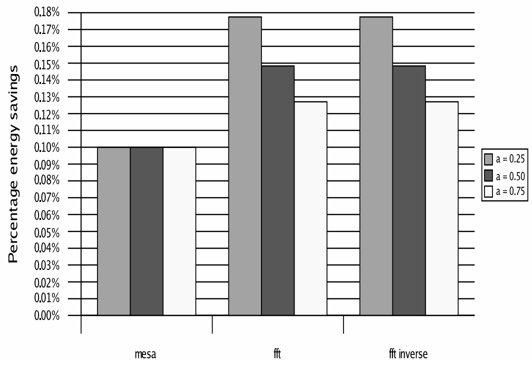


Figure 19: Percentage energy savings in FPUs

power consumption in the functional units is reduced by constructing a power aware control flow graph during compilation and performing static analysis over it to analyse the usage of each functional unit in a basic block and insert specialized instructions at the begining and ending of the basic blocks. These instructions are used to control the functional units. In their methodology, even if we assume that the ISA can be extended without extra frills, it still calls for certain amount of hardware support at the *decoding stage* of the instructions. Moreover, the technique does not consider the idle times in ALU units and Ports. But as shown in [13] the profit of turning these units off for shorter durations is sizeable. In [17] the authors provide a theoretical limit for the peak energy consumption in functional units with an unbalanced distribution of instructions. This limit was used in [8] inorder to reduce the peak power consumption in functional units. The aim of these works was to reduce the variation in power consumption due to various functional units by balancing the usage throughout the program. But the techniques result in very less average idle cycle time for each functional unit thereby increasing the static power consumption. Static power consumption is increasingly becoming a bottleneck as the technology is reaching 0.07um stage and the leakage is going to be significant portion of the total power consumption. This motivated us to look into the chance of increasing the average idle cycle time of functional units. In the context of VLIW data paths, [1] proposed an instruction scheduling scheme to reduce the power consumed by the instruction bus in the fetch unit. They provide an optimal algorithm for horizontal scheduling which schedules micro instructions' packets and vertical scheduling which provides an ordering inside each instruction packet. In [21] VLIW functional units are are assumed to run at various voltage

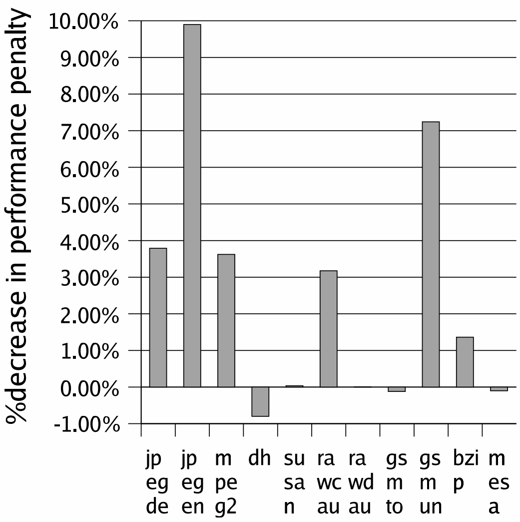


Figure 20: Multiissue processor performance penalty

levels and instructions are issued to these units based on the slack for each instruction. Their technique exibits a significant reduction in power usage when the performance degradation threshold is very large. In addition the transition energy costs in the functional units when they operate at different voltage levels are not considered. In [10] a critical path predictor is used to assign critical instructions to functional units that operate at high voltage levels whereas other instructions are issued at low voltage level units. The available benefit varies with the accuracy of the branch predictor, which costs additional power.

6 Conclusion

In this paper a scheduling technique is proposed through which the number of transitions between active mode and idle modes in the units ALU, MUL, FPA units and memory ports are reduced, there by increasing the number of continuous idle cycles of each unit. This reduces the static power as the unit will be in an idle state for a longer time. Moreover, the reduction in the number transitions will reduce the transition delay and transition energy which significantly impact the total execution time and power consumption when cicuit level optimization techniques such as biased bitlines or input vector control are used. We have evaluated this technique with an analytical energy model available in the literature. Simulation results indicate a significant reduction in the total power consumption of the units and also a reduction in the performance penalty introduced by the transitions. This work shows how the grouping of "similar" instructions in more complex architectures with large issue width will result in energy savings in the units.

References

[1] C.Lee, J.K.Lee, T.T.Hwang, and S.C.Tsai. Compiler optimization on VLIW instruction scheduling for low power. In *ACM Transactions on Design Automation of Electronic Systems*, pages 252–268, 2003.

[2] C.Lee, M.Potkonjak, and W.H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*, pages 330–335.

[3] Intel Corporation. Intel XScale microarchitecture for the PXA255 processor: User's manual.

[4] D.Brooks, V.Tiwari, and M.Martonosi. Wattch:a framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, pages 83–94.

[5] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches:simple techniques for reducing leakage power. In *International Symposium on Computer Architecture*, pages 148–157.

[6] G.Cantures, M.Martonosi, J.Peng, R.Ju, and G-Y.Lueh. XTREM: A power simulator for the intel XScale core. In *Conference on Language, Compiler and Tool Support for Embedded Systems*, pages 115–125.

[7] M.R. Guthaus, J.S. Ringenberg, D.Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE Annual Workshop on Workload Characterization*.

[8] G.Gao H.Yang and C.Leung. On achieving balanced power consumption in software pipelined loops. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 210–217, 2002.

[9] J.A.Butts and G.S.Sohi. A static power model for architects. In *International Symposium of Computer Architecture*, pages 191–201.

[10] J.S.Seng, E.Tune, and D.Tullsen. Using criticality for power consumption. In *International Symposium on Microarchitecture*, pages 114–123, 2002.

[11] AJ KleinOsowski and D.J. Lilja. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. In *Computer Architecture Letters*, 2002.

[12] N.Vijayakrishnan, M.Kandemir, M.J.Irwin, H.S.Kim, and W.Ye. Energy-driven integrated hardware-software optimizations using simpower. In *International Symposium on Computer Architecture*, pages 95–106.

[13] S.Dropso, V.Kursun, D.Albonesi, S. Dwarakadas, and E.G.Friedman. Managing static leakage energy in microprocessor functional units. In *International Symposium on Microarchitecture*, pages 321–332, 2002.

[14] S.Heo, K.Barr, M.Hampton, and K.Arisovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *International Symposium on Computer Architecture*, pages 137–147, 2002.

[15] S.Rele, S.Pande, S.Onder, and R.Gupta. Optimizing static power dissipation by functional units in superscalar processors. In *International Conference on Compiler Construction*, pages 261–275, 2002.

[16] T.A.Proebsting and C.W.Fraser. Detecting pipeline structural hazards quickly. In *Symposium on Principles of Programming Languages*, pages 280–286.

[17] T.Li and C.Ding. Instruction balance and its relation to energy consumption. In *International Workshop on Languages and Compilers for Parallel Computing*, 2001.

[18] V.Bala and N.Rubin. Efficient instruction scheduling using finite state automata. In *International Symposium on Microarchitecture*, pages 46–56.

[19] V.Delaluz, M.Kandemir, N.Vijayakrishnan, and M.J.Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 138–147, 2000.

[20] www.spec.org. SPEC benchmark suite. 2000.

[21] W.Zhang, M.Karakoy, M.Kandemir, and G.Chen. Compiler support for reducing leakage energy in data cache. In *International Conference on Supercomputing*, pages 76–85.

[22] Y.F.Tsai, D.Duarte, N.Vijayakrishnan, and M.J. Irwin. Impact of process scaling on efficacy of leakage reduction schemes. In *International conference on IC Design and Technology*, 2004.

[23] W. Zhang, N.Vijayakrishnan, M.Kandemir, M.J.Irwin, David Duarte, and Yuh-Fang Tsai. Exploiting VLIW schedule slacks for dynamic and leakage energy reduction. In *International Symposium on Microarchitecture*, pages 102–113.