The 2004 IEEE Asia-Pacific Conference on
Circuits and Systems, December 6-9, 2004

ON THE EFFECTIVENESS OF DYNAMICALLY ALLOCATING RESOURCES ACROSS PROGRAM EXECUTION PHASES FOR MEDIA WORKLOADS

Subhasis Banerjee, G.Surendra, S.K.Nandy
CAD Lab, SERC, Indian Institute of Science, Bangalore - 560 012; INDIA
E-mail : [subhasis@cadl, surendra@cadl, nandy@serc].iisc.ernet.in

ABSTRACT

Processing embedded applications is essentially a trade-off between power and performance. Increasing level of complexity in present day microprocessor at the expense of more power call for different optimization methodologies at architecture level. The study of general characteristics of program execution phases gives insight to dynamically reconfigure or enable/disable additional resources on-demand basis. This leads to significant amount of power saving with negligible or tolerable performance degradation. In this paper we characterize execution of such programs into execution phases based on their dynamic IPC profile. We show that program execution of selected phases (based on IPC profile) can be dynamically boosted by activating additional standby functional units which are otherwise powered down for saving energy. Through simulation we show that speedup ranging from 1.1 to 1.25 can be achieved while reducing the energy-delay product (EDP) for most of the media benchmarks evaluated.

1. INTRODUCTION

Power dissipation and energy consumption are becoming primary design concerns in both embedded System on a Chip (SoC) solutions and high performance processors. This is primarily due to clock rates and die sizes which are constantly increasing with advances in technology scaling. Power translates directly to heat which may lead to thermal runaway, junction fatigue, localized hot spots and other reliability problems. New packaging techniques and separate cooling solutions may not be cost effective in SoC platforms consisting of multiple (possibly heterogeneous) processors. The design of complex systems involves analyzing the interaction between different components of the system (both hardware and software) and their impact on power, performance and area. It is becoming increasingly necessary to evaluate power and energy consumption at different stages of the design and make important decisions early in the design process. Research in architectural level power optimizations aim at analyzing the impact of micro-architectural parameters (both resources and data dependencies) on power and performance and designing architectures that are *power aware*. Runtime optimization of processor resources is becoming increasingly important from the perspective of power aware processor architecture. With the increasing complexity of the application programs, the processing requirement is growing up. Additional resources, when incorporated into the processor, increase the performance and also the power dissipation. Runtime optimization techniques address both performance and power based on certain characteristics of the program executed. During our study of program behavior analysis, we find that there exists distinct, periodic *phases* of execution characterized by IPC, Reorder Buffer (RoB) occupancy, instruction issue rate, branch miss rate, cache miss rate etc. All these parameters show a definite pattern during execution when studied for a longer period of time. Our simulations show that the EDP does not changes significantly during the execution even though a speedup of 10 to 20% is achieved by allocating additional functional units at different phase of a program.

In our experiment with a four way issue superscalar processor we get average IPC just above 2, as shown in figure 1 for mpeg2decode benchmark. We observe that the program phases are control dominated as it remains invariant of the input data stream. In the figure each point in time axis represents average IPC over an interval of 10,000 cycles. The average variation of IPC over total execution time for the benchmarks studied is from 40% to 68% of the commit width. There is significant amount of ILP(Instruction Level Parallelism) unutilized by the processor due to resource constraint and/or true data dependence. In this paper we study the effect on execution time and the EDP by scaling the functional units and the instruction fetch window when the dynamic IPC attains a value over some threshold during program execution. Though the scaling of processor resources increases the power consumption, we find that the variation in EDP is negligible.

This work is partially supported by a research project from Philips Research.

2. RELATED WORK

There exists hardware and software profiling schemes to dynamically optimize programs during execution. In[2], the *ProfileMe* approach taken by Dean *et. al.* provides hardware support for out-of-order superscalar engine. Detection of program *hot spots* by using hardware detector is proposed by Matthew C. Martin *et al.* in [3]. The hot spot detector identifies a group of frequently executed basic blocks and uses the information to steer compiler to optimizations at runtime. An analysis of basic block distribution in [1] leads to an automated approach to identify a smaller subset of code which represents overall program characteristics. The periodic behavior of the program, observed in [1] actually reflects the behavior of basic blocks that are frequently executed in a section of a code. In our work we identified different state of processor utilization by using dynamic IPC profile and issue rate and decision of resource allocation is taken at every different phase. Marculescu [6] propose a mechanism to dynamically adapt the fetch and execution bandwidth based on profile information at the basic block level using a compiler based scheme. Brooks *et al.* [7] observe that many modern processors are over-provisioned and minimize power consumption in functional units by exploiting the fact that the sizes of operands are usually small for many programs. In this work, we borrow the concept of dynamically turning on/off resources to compensate for the throttling at fetch stage which tends to inhibit performance.

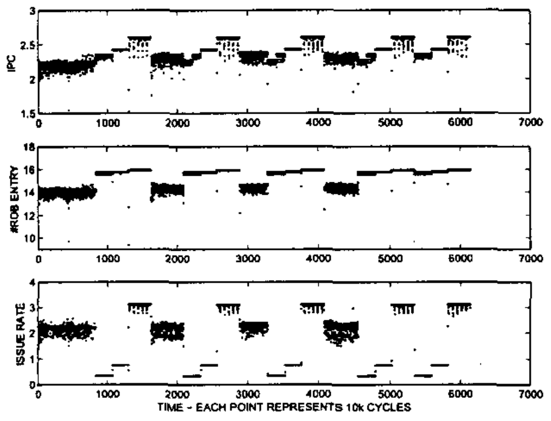

Figure 1: Variation of dynamic IPC, number of RoB entry and number of instruction issued per cycle with time for *mpeg2decode* using two different data inputs

3. SIMULATION FRAMEWORK

We use SimpleScalar3.0 [8] for the PISA instructions to simulate a dynamically scheduled superscalar processor. For power estimation we use Wattch [9], a performance and power analysis simulation tool that is built on top of SimpleScalar. Just like other architectural power analysis tools, Wattch relies on activity based power models to estimate power demands. It incorporates conditional clocking (*cc*) at different levels. In one scheme of conditional clocking option a component consumes full power when accessed and zero power when idle. In a more realistic scheme, a component consumes linearly scaled power based on its usage and 10% of base power when idle. Wattch provides architecture researchers with enormous flexibility in designing and tweaking various components to study various power-performance trade-offs primarily because it is derived from the SimpleScalar framework. The authors of Wattch compare their power estimate with other tools and claim an accuracy to within 10% of layout-level power tools. More details on the working of Wattch and possible shortcomings can be found in [10] [11] and [9]. Throughout this paper, we use the *cc3* style of clock gating provided by Wattch which scales power linearly with unit usage. Inactive units still dissipate 10% of its maximum power rather than drawing zero power. This is more realistic in industrial clock-gated circuits [9].

Processor Model Parameters

Table 1 shows the parameters used for carrying out simulations. The *base configuration* parameters roughly match those of a embedded processor. The *extended configuration* parameters become effective only when we attempt to dynamically change the number of resources based on available parallelism. Though all *base configuration* parameters can be changed, we vary only the *issue width* and the number of functional units in the *extended configuration* since IPC is mainly affected by these parameters. In other words, the processor configuration can be changed from *base* to the *extended* configuration and vice-versa with unused units being gated with the *cc3* policy. We use process parameters for a .18μm process at 600MHz in all our simulations. We evaluate our ideas on a set of benchmarks from MiBench[12]. The benchmarks used are *adpcm, jpeg, mpeg2 encode/decode*, and *mp3 encoding (lame)*. We compile the benchmarks with -O3 optimization, simulate them to completion and use the standard inputs available in public domain.

Table 1: Processor parameters for *base* and *extended* configurations in our simulations.

Parameter	value
Fetch Queue Size	8 instructions
RUU Size	64 instructions
LSQ Size	16 instructions
Fetch width	4 instructions/cycle
Decode width	4 instr/cycle
Issue width (OoO)	4 instr/cycle (8 *extended*)
Commit width	4 instr/cycle (in-order)
Functional Units	4 int ALUs (8 *extended*), 1 int mult/div (2 *extended*) 2 mem ports (4 *extended*)
Branch Predictor	Combined, Bimodal 4K table 2-Level 1K table, 10 bit history 4K chooser
BTB	256 entry, 2-way
Return address stack	8-entry
Mispredict penalty	3 cycles
L1 data cache	64K, 1-way, LRU 32B block, 1 cycle latency
L1 instruction cache	64K, 1-way, LRU 32B block, 1 cycle latency
L2 cache	Unified, 512kB, 4-way, LRU 32B block, 10 cycle latency
Memory	40 cycle first chunk latency
TLB	32 entry (itlb), 32 entry (dtlb), 4-way, 20 cycle miss latency

4. INTERACTION BETWEEN PROCESSOR PARAMETERS

It is practically impossible to simulate all possible combinations of a set of parameters to estimate the interactions between them. To evaluate the interactions between parameters we chose Plackett and Burman design to limit the number of simulation about the number of parameters considered [4]. This design is a two level fractional factorial design for studying $k = N$-1 variable in N runs where N is a multiple of 4 [5]. In P-B (Plackett-Burman) design all the k parameters are varied simultaneously over N simulations. An improvement over basic P-B design is fold-over P-B design which incorporates $2N$ simulations. Table 2 shows a smaller version of P-B design matrix with fold-over for k = 7, N = 8. A +1 value for a parameter indicates a high value and -1 indicates the low value. The first row is a sequence which is fixed for a given value of k. Row 2 to row N-1 is generated by circular right shift of the previous row. Row

N is formed by taking all the values to be at lower end (-1). For a fold-over P-B design matrix, N more rows are added with the base design matrix. The sign of each entry of the additional rows is opposite of the corresponding entries of the original matrix. Each row indicates the configuration of micro-architecture parameters A, B, C, D, E, F, and G with different combinations and the result of the metric (*e.g*, IPC) is given at the last column. The effect of each parameter on the metric can be found by taking the weighted sum of the metric with the elements of the column corresponding to the parameter. For example, the effect of A can be computed as follows :

$\mathrm{Eff}_A = \{(1^*X_1) + (-1^*X_2) + (-1^*X_3) + \ldots + (1^*X_{16})\}$.

We considered $k = 19$ parameters to be studied over the range of $N = 40$ simulations. The first row of the P-B design matrix is given as +1, +1, -1, -1, +1, +1, +1, +1, -1, +1, -1, +1, -1, -1, -1, -1, +1, +1, -1 [5]. A P-B matrix of size 40x19 is generated using the method described above. Values are *deliberately* chosen to be slightly higher and lower than the range of normal values. To assign a number corresponding to the effect of a particular parameter, the weighted sum of a metric (IPC or Power) is taken column-wise. Table 3 shows the rank of each parameter which indicate the influence of it on a particular metric.

Table 2: Plackett-Burman matrix for k = 7 {Parameters A, B, C, ... , G could be any of the parameters from Table 3}

A	B	C	D	E	F	G	Metric
+1	+1	+1	-1	+1	-1	-1	X_1
-1	+1	+1	+1	-1	+1	-1	X_2
-1	-1	+1	+1	+1	-1	+1	X_3
+1	-1	-1	+1	+1	+1	-1	X_4
-1	+1	-1	-1	+1	+1	+1	X_5
+1	-1	+1	-1	-1	+1	+1	X_6
+1	+1	-1	+1	-1	-1	+1	X_7
-1	-1	-1	-1	-1	-1	-1	X_8
-1	-1	-1	+1	-1	+1	+1	X_9
+1	-1	-1	-1	+1	-1	+1	X_{10}
+1	+1	-1	-1	-1	+1	-1	X_{11}
-1	+1	+1	-1	-1	-1	+1	X_{12}
+1	-1	+1	+1	-1	-1	-1	X_{13}
-1	+1	-1	+1	+1	-1	-1	X_{14}
-1	-1	+1	-1	+1	+1	-1	X_{15}
+1	+1	+1	+1	+1	+1	+1	X_{16}

It is found from table 3 that RUU size, number of integer ALUs, L1 cache size are primary factors to be considered in dynamically reconfiguring the processor resources during phases of varying activity. We compute average IPC in a window of execution cycles, which is a representative of the phase of the program bounded by that window. In our experiment the number of cycles over which IPC is averaged to represent a state or phase of the processor, is 10,000. When the processor enters into a state or phase it remains there for sometime and any measure taken to improve performance should sustain for that period of time. The smallest unit for which a resource configuration decision is made is taken as 10,000 cycles. A too small interval will lead to over-configuring the functional units and a too large interval will miss opportunities. We find 10,000 cycle period is a reasonable approximation. In this present paper we scale resources depending on the state of processor resource utilization. The base and extended state of processor resources are shown in table 1.

5. FUNCTIONAL UNIT UNDERUTILIZATION

During the phase of functional unit underutilization the instructions in pipeline wait for free functional unit. To improve performance (though at the cost of average power/cycle), we allocate or power on additional resources. These resources are in a standby mode (powered off or gated) during normal mode of processor operation. When the potential for increased parallelism exists, these resources are powered on (switched to an active mode) to cater to the additional demand thereby increasing the IPC.

Table 3: Plackett-Burman method to identify the parameters that influence IPC and Power

Parameters	Low value	High value	Rank IPC	Rank Power
RUU size	8	128	1	2
Int-ALU num	1	8	2	4
D1 cache size	4 kB	128 kB	3	3
LSQ size	4	32	4	5
L2 latency	25 cycle	5 cycle	5	8
D1 cache lat	4 cycle	1cycle	6	6
L1 cache size	4 kB	128 kB	7	1
L1 cache lat	4 cycle	1 cycle	8	9
BTB entry	16	512	9	16
L2 cache asc	1 way	8 way	10	13
Mem lat	100 cycle	20 cycle	11	11
D-TLB size	16 entry	128 entry	12	19
Br pred.	bimod	perfect	13	12
Fetch size	4	32	14	14
L2 cache size	128 kB	2 MB	15	10
Mem b/w	4 bytes/c	32 bytes/c	16	18
FP-ALU num	1	4	17	7
I-TLB size	16 entry	128 entry	18	15
Br. mis.lat	1 cycle	8 cycle	19	17

We use a scheme to power on/off the additional resources based on the following observations:-

(*i*)When the processor issues instructions at its maximum issue rate it continues to issue at the same (sometimes at nearly the same) rate for a certain number of cycles. It is observed that on average the processor issues 4 instructions/cycle more than 60% of the time. The additional resources are powered on (which are used in subsequent phases) if the IPC is above some threshold value in the previous phase. The resources are powered off if the maximum issue rate is not honored *i.e.* when the processor is underutilized.

(*ii*) Another heuristic that was evaluated was to turn on the additional resources when the number of instructions that are ready to be issued is greater than the issue width of the *base configuration*. In addition to increasing the resources, we impose fetch throttling (with 2 cycles stall) which tends to reduce power marginally.

5.1 Resource Allocation in Presence of Fetch Stall

We study the runtime behavior of program in terms of IPC and issue rate during execution. As discussed in the previous section we identify the phase or state of the processor based upon dynamic IPC (averaged over 10,000 cycles). Additional resources are powered on when the processor attains a state where it records IPC more than P. Our study of behavior of different programs based on IPC, average RoB occupancy and average issue rate, shows that these parameters exhibit discontinuity in their profile (see figure 1). This behavior strengthen the scheme of selecting a threshold for transition from one state of processor configuration to another. The scheme for allocation of resources is as follows:

```
get_processor_state_id();
if
((processor_state_id == base) && dynamic_IPC > P)
switch_to_extended_state;
else if
((processor_state_id == extended) && dynamic_IPC <Q)
switch_to_base_state;
```

When the processor is executing at the extended state the number of instructions in the ready queue should be higher to provide adequate number of instructions to the functional units. Non availability of the ready instructions due to true dependence at this phase will leave the functional unit idle and the power consumption will be more and hence the processor is switched over to the base configuration.

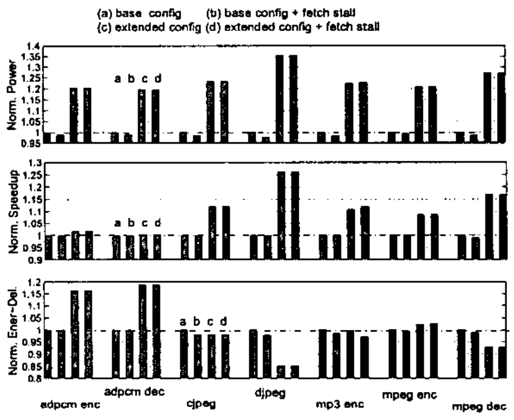

Figure 2: Speed up and EDP with Resource Allocation in Presence of Throttling

The choice of P and Q is made based on analysis of program phases for different benchmarks used in this experiment. When additional functional units are switched on, we expect ILP to be more than or equal to the issue width of *base configuration*. With this justification we select Q as 4. We set P equal to 2.5 based on the fact that most of the benchmarks show ILP around 2. Figure 2 shows the normalized IPC, power and EDP with dynamic resource increase (*extended configuration*) with and without fetch throttling. Bar (a) gives the base value with respect to which normalization is done. All benchmarks except *adpcm* show an average 1% to 3% power savings with negligible performance degradation. The EDP also decreases with throttling and resource allocation indicating that introducing stalls is beneficial for most benchmarks considered. With the extended configuration, the performance improves in all programs (very little improvement in adpcm due to limited parallelism) with increased power dissipation. JPEG decoding yields the best improvement in the EDP (15% reduction) with a 25% improvement in speedup and 35% extra power dissipation for the extended configuration.

6. CONCLUSIONS AND FUTURE WORK

In this paper we analyzed the phased behavior of program to determine the execution state of processor and quantify it in terms of IPC and issue rate. Optimizations by incorporating resource scaling with stall at pipeline stages, produced improvement in EDP with speedup by 1.15 on average. Runtime optimization of execution speed and processing power depends on efficient detection of program phases. Power, which is an important design parameter, depends on the instruction density in pipeline stages. Effective utilization of resources minimize the occupancy of instruction in the pipeline. In our scheme additional stalls clear the pipeline congestion and on average 10% power can be saved with negligible reduction in performance. Through this work we establish the need for automatic identification of program phases and corresponding micro-architectural support to optimize these program phases at runtime.

7. REFERENCES

[1] Timothy Sherwood, Erez Perelman and Brad Calder. "Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications" *I*nProc. of Intn'l Conf. on Parallel Architecture and Compilation Technique, Sept. 2001

[2] J. Dean, J.E. Hicks, C.A. Waldspurger, W.E. Weihl and GChrysos, "Profileme: Hardware support for instruction level profiling on out of order processors," In *Proc. of the 29th Annual Intl. Symp. on Microarchitecture,* December 1997.

[3] Matthew C. Martin, *et. al.*"A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization," In *Proc. of the Intl. Symp. on Computer Architecture, 1999*

[4] Joshua J. Yi, David J. Lilja and Douglas M. Hawkins. "A statitically rigorous approach for improving simulation methodology", In *Proc. of High Performance Computer Architecture, 2003.*

[5] Douglas C. Montgomery. "Design and Analysis of Experiments", *5th Edition, John Wiley Inc.*

[6] D.Marculescu, "Profile-Driven Code Execution for Low Power Dissipation," *In Proc. of Intn'l Symp. on Low Power Electronics and Design,* 2000.

[7] D.Brooks, and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance," *HPCA 1999.*

[8] D. Burger, T.M Austin, and S. Bennet, "Evaluating Future Microprocessors: The SimpleScalar Tool Set," Technical Report CS-TR-96-1308, University of Wisconsin-Madison, July 1996.

[9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for Architectural-Level Power Analysis and Optimization," In *ISCA, June 2000.*

[10] S. Ghiasi and D. Grunwald, "A comparison of two architectural power models," *In Workshop on Power-Aware Computer Systems,* Nov 2000.

[11] J. L. Aragon, J. Gonzalez, and A. Gonzalez, "Power-Aware Control Speculation through Selective Throttling," *In HPCA 2003.*

[12] M. Guthaus, *et. al.* "MiBench: A free, commercially representative embedded benchmark suite," *IEEE 4th Annual Workshop on Workload Characterization,* Dec 2001.