Exploring Efficient Operating Points for Voltage Scaled Embedded Processor Cores

Marcio Buss, Tony Givargis, and Nikil Dutt
Department of Computer Science
Center for Embedded Computer Systems
University of California, Irvine, 92697
{marcio,givargis,dutt}@ics.uci.edu

ABSTRACT

Portable and battery operated devices pose a unique design challenge in terms of performance requirements, low power constraints, and short design cycles. Embedded soft cores, on the other hand, provide functional flexibility and guarantee rapid design and thus are gaining popularity in designing such portable and battery operated devices. To address the low power needs, dynamic voltage scaled (DVS) processors provide a new tradeoff dimension to the designer. This work proposes an application-specific design space exploration framework for selecting energy-efficient operating points in an embedded soft core. Specifically, we address the problem of selecting an appropriate number of operating voltage/frequency points and the distribution of these points along the valid voltage span of a processor, given the application that is to be executed on the processor. Furthermore, we provide a static intra-task scheduling technique that reduces energy consumption (4-20% in our experiments) even when the worst-case application execution time does not leave any slack for effective voltage scaling. We have experimentally verified our techniques on a large set of embedded benchmarks selected from MiBench, PowerStone and MediaBench.

1. INTRODUCTION

There is a growing demand for portable and battery operated devices whose functional complexity is constantly rivaling or even exceeding the performance demands of desktop computing applications. Examples of such devices include cellular phones, personal digital assistants, mobile videophones and a plethora of similar multi-media, communication, and networking systems. Given a familiar programming model, well evolved tool chains, functional flexibility, and often adequate performance, processor *soft cores* are increasingly playing an important role in the design of such portable and battery operated devices, especially under towering time-to-market pressures.

However, given their general-purpose nature and consequently energy inefficiency, the use of processor soft cores in low power applications poses a significant design challenge. To address this problem, embedded processor core vendors are offering an increasing degree of customization potential (e.g., data-path and memory tuning, instruction-set tailoring, register file sizing, etc. [19][9]) in order to provide a designer with a performance/power tradeoff space and means to better tailor a processor core to the needs of a specific application.

Additionally, in recent years, dynamic voltage scaled (DVS) processors have offered a new dimension in low power design. In a DVS processor the operating voltage (and proportionally the operating frequency) can be dynamically changed by software. In CMOS, the energy consumption of a system is proportional to the square of the operating voltage [1], thus, the objective of any DVS system is to run the processor at the lowest operating voltage while meeting all task deadlines for optimal energy use. Toward this goal, and with much success, a large body of work has focused on static and dynamic scheduling algorithms and design methods.

A major challenge in the use of DVS soft cores within a codesign flow is the determination of efficient operating points that are tuned for specific applications. However, much of the previous work has operated under the assumption that a DVS processor has a fixed number N of operating points (i.e., voltage/frequency points) which are uniformly distributed along the valid voltage span of a processor. In this work, we demonstrate that the choice of N and the distribution of these points along the frequency/voltage axis is a significant design decision. Specifically, we argue that these choices should be part of the design space of a processor soft core and with respect to the application that is eventually to be executed on the processor. As with other customization potentials (e.g., data-path and memory tuning, instruction-set tailoring, register file sizing, etc.), the DVS subsystem of a processor soft core needs to be explored in an iterative codesign methodology. In this work, we also outline a static intra-task scheduling technique that reduces energy consumption even when the worst-case application execution time does not leave any slack for effective voltage scaling.

The remainder of this paper is organized as follows. In Section 2, we outline related work. In Section 3, we formulate

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

the problem. In Section 4, we give our solution. In Section 5, we provide our experimental results. In Section 6, we state our conclusions and future work directions.

2. RELATED WORK

Dynamic processor voltage scaling was first exploited by Govil et al. [3] and Weiser et al. [21]. These works applied the classical speed/energy tradeoff on a collection of UNIX workstation benchmarks. In their works, the microprocessor frequency was scaled in proportion to the global processor utilization, computed in fixed intervals of time. Hence, the granularity of voltage scaling was very high.

In later work, Yao et al. [22] have contributed an optimal preemptive offline scheduling algorithm taking as input a set of independent tasks with arbitrary arrival times and deadlines on a variable speed processor. Several researchers have followed a slightly different path and have addressed power issues in event-driven systems by presenting various techniques for shutting down the system or parts of the system [5][16].

Hong et al. [4] have analyzed the theoretical scheduling of a hard real-time set of tasks, using worst-case instead of average-case work load estimates. In their work, they present an online scheduling algorithm that optimally schedules a set of active real-time tasks. The same authors have also described a design methodology for DVS processor core design targeted for real-time system-on-chip architectures. Furthermore, they have provided an offline scheduling heuristic for non-preemptive real-time tasks as well as the selection of other architecture configurations (e.g., instruction and data cache size).

Lee et al. [7] have proposed an intra-task voltage scheduling strategy where each task is partitioned into fixed-length timeslots. Although they report significant improvement in the energy reduction, they do not provide a systematic guideline for selecting the best program locations where voltage scaling instructions should be inserted.

Shin et al. [15] have proposed an intra-task voltage scheduling algorithm based on a static analysis technique, in order to convert a DVS-unaware program into a DVS-aware one. While their approach is similar to our work, they do not consider tailoring operating frequencies for a given benchmark in a design exploration context.

Quan and Hu [14] present a technique to determine voltage settings for a variable voltage processor that utilizes a fixed priority assignment to schedule jobs. Their approach also produces the minimum constant voltage needed to feasibly schedule the entire job set. In their approach, they assume a processor whose voltage can be set to any value in a continuous interval. In our work, we seek to discover and analyze the number of operating points and their specific distribution for optimal power savings.

Our work is an attempt at exploring the design tradeoffs, in terms of energy and real-time timing constraints, presented by the voltage scaling subsystem design. In other words, while previous works have taken as input a prearranged N uniformly distributed DVS points and generated optimal (or near-optimal) schedules, our work goes a step further by treating N and the distribution as additional design parameters to be optimized. Since flexibility is afforded by emerging soft processor cores, exploration of this space becomes important for codesign of future embedded systems.

3. PROBLEM FORMULATION

3.1 Basic Terminology

We denote a real-time task τ as having a corresponding deadline $D\tau$. The task τ is represented by its control flow graph $G\tau = (V,E)$, where V is the set of basic blocks and E corresponds to the set of control flow edges. Two distinct basic blocks, $ENTRY \in V$ and $EXIT \in V$, represent τ's entry and exit nodes. $ENTRY$ has no predecessor nodes and $EXIT$ has no successor nodes. The successor basic blocks of any $v \in V$ are denoted by $succ(v)$. $e \in E$ is said to be a back edge when its destination node dominates [10] its source node. It is usually the set of edges defining loops.

Each basic block i is annotated with its non-zero latency in number of cycles at maximum frequency, C_i (since the execution time of a given basic block depends on its actual slowdown factor, we use the block's number of cycles as the reference). η_i represents the slowdown factor assigned to block i, and can be viewed as the normalized frequency: at any given instant, η_i is the ratio of the scheduled frequency to the maximum frequency of the processor. Since the voltage/frequency scaling decisions are made statically we ignore the time and energy overhead incurred in changing the frequency and voltage of the processor (this overhead is instead folded into the basic block's execution time). $D\tau$ is set to be equal to τ's worst-case execution time.

3.2 Variable speed processors

A wide range of processors already support multiple voltage and frequency levels. Among the most popular we find Intel StrongARM processors [17], Transmeta Crusoe [2] and Intel XScale [13]. Intel Pentium M processors, primarily used on mobile devices, also present a proprietary DVS technology called SpeedStep [12]. We assume that future soft cores will therefore allow specification of voltage/frequency levels as parameters in their instantiation.

Voltage and frequency levels are tightly coupled. Whenever the processor's operating frequency changes, its supply voltage is accordingly scaled. Therefore, when we perform a slowdown we change both the frequency and voltage of the processor. We assume that the frequency can vary from f_{min} to f_{max}, or that η_i is located in the range $[\eta_{min},1]$, where $\eta_{min} = \frac{f_{min}}{f_{max}}$.

4. PROPOSED APPROACH

Our technique is a three-phase approach based on static timing analysis that (1) assigns an "optimal" slowdown factor for each basic block on each task, (2) computes the set of (tailored) operating frequencies by looking at the entire application results, and (3) reassigns a proper speed to each basic block such that only valid operating frequencies – previously computed during Phase 2 – are used. In other words, the first phase computes static slowdown factors under the assumption that the processor could run at any possible speed ranging from $f_{min} = 0$ to f_{max}. The goal here is

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

to assign a speed to each basic block so that the energy consumption is optimally minimized while the timing requirements are still satisfied. Then, during Phase 2, a sub-set of all existing slowdown factors is chosen to serve as the actual set of operating frequencies for the application being analyzed. This characterizes an application-tailored DVS strategy. Finally, Phase 3 reassigns each basic block's speed to the smallest operating frequency that still guarantees that timing requirements are met.

We note that the actual number of operating frequencies to be found during Phase 2 is an input parameter to our algorithm, which defines the solution space. In other terms, during design space exploration, and for a given application, we exercise the three-phase approach multiple times, each time considering a different number of operating points to be encountered. In this particular work we have exploited 2, 3, 4, 8, 12, and 16 operating frequencies. In the remaining subsections, we take a look at each of these three phases in detail.

4.1 Phase 1: Speed Assignment

The first phase of our technique is formulated as a linear programming (LP) method. Within a given task, each basic block i has a corresponding start time S_i. We say that the block's start time plus its execution time cannot exceed any of its successors' start times. The execution time of a basic block is defined as its number of cycles C_i multiplied by the inverse of its slowdown factor η_i, defined as η_i' (if η_i is 0.5, indicating 50% of f_{max}, η_i' is equal to 2). In mathematical terms,

$$S_i + C_i\eta_i' \leq S_j, \text{ for each } j \in succ(i), \text{ and} \qquad \text{(a)}$$

$$S_{entry} = 0. \qquad \text{(b)}$$

The exit basic block has a null successor set, thus:

$$S_{exit} + C_{exit}\eta_{exit}' \leq D, \qquad \text{(c)}$$

where D is the task's deadline and it is set to be equal to the task's worst-case execution time in number of cycles (WCEC). The bounds for the LP formulation to be solvable are:

$$1 \leq S_i \leq D, \text{ for each } i \neq entry, \text{ and} \qquad \text{(d)}$$

$$\eta_i' \geq 1. \qquad \text{(e)}$$

The objective function is to maximize each possible slack time within the task's boundary.

$$maximize \sum \eta_i'. \qquad \text{(f)}$$

The objective function's rationale is that it tries to maximize any internal slack available within the application. This is a simple example used in this work, however more complex objective functions are also available in our framework.

Notice that by generating instances of Equation (a) for all basic blocks $i \neq exit$ within a task τ, we are implicitly covering all possible execution paths for τ without the need of enumerating all of them.

Consider for example a real-time task τ shown in Figure 1(a). This example has been extracted from the `mpeg` benchmark. Each node in Figure 1(a) represents a basic block and each edge indicates a possible flow of control between two basic blocks. The number within each node indicates its index, whereas the number outside the node indicates its latency in number of cycles. The worst-case execution cycles (WCEC) can be computed as the cost of the longest path from *ENTRY* to *EXIT*. In this paper we consider that the number of maximum loop iterations is user-provided. Therefore, we unroll any existing loop within each task in order to get a loop-less graph and compute the task's WCEC. This way, each block has exactly one start time assigned to it.

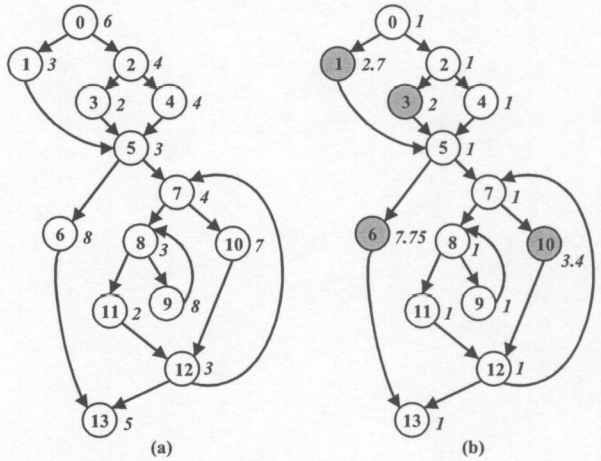

Figure 1: (a) Control-flow graph of a real-time task τ from `mpeg`. (b) Computed slowdown factors.

After generating LP equations for τ, an LP solver can be used to determine "optimal" results for all start times and slowdown factors. These results can be back annotated into the task's basic blocks. Figure 1(b) shows the obtained slowdown factors for τ. Notice that by assuming $deadline = WCEC$ a number of blocks ended up having η' = 1, or f_{max}. However, it turns out that some basic blocks belong to alternative execution paths that present an internal slack time. We call these blocks as "stretchable" nodes, since they are located along unbalanced paths where the execution may branch and traverse a shorter distance to the *exit* node. For instance, basic block *6* in Figure 1(a) has η' = 7.75 - the execution paths from its predecessor basic block (*5*) towards the *exit* node are clearly unbalanced. The same is observed for nodes *1, 3* and *10*.

The question then becomes how many of those "stretchable" basic blocks actually exist in a typical embedded application. If a reasonable percentage of all basic blocks belong to this category, and if they are reasonably frequently executed considerable energy savings can be obtained even within our tight preliminary assumption that $deadline = WCEC$. Savings are achieved by slowing down the operating frequency for those blocks where $\eta' > 1$. Indeed, our experiments show that for the vast majority of the benchmarks analyzed, approximately 30% of the basic blocks have $\eta' > 1$. Furthermore, when $deadline > WCEC$, we can expect much larger energy savings.

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

4.2 Phase 2: Finding Operating Frequencies

Recall that Phase 1 assigns static slowdown factors for each task's basic blocks regardless of any constraint such as a limited number of operating frequencies. In other words, it assumes that the processor could run at any possible speed ranging from $f_{min} = 0$ to f_{max}. In practice, however, only a few operating frequencies are feasible.

Therefore, an additional step of our strategy is to determine which sub-set of all individual slowdown factors obtained in Phase 1 will represent the actual set of N operating frequencies for the benchmark being analyzed. This tailoring analysis is done by first grouping together all basic blocks from all tasks $\tau_1, \tau_2, \tau_3, .., \tau_m$ that ended up having the same slowdown factor, and then creating a frequency distribution graph out of these grouped results. Figure 2 shows an example of such a graph. The y axis corresponds to the number of occurrences of a particular slowdown factor, which in turn are represented along the x axis. Each vertical bar, therefore, tells us how many basic blocks within the entire benchmark have been assigned with a specific slowdown factor. For this analysis the actual η is used instead of η'.

Determining the frequency distribution graph, however, is not sufficient. Basic blocks in general have different latencies, and hence different power figures even when executing at the same clock frequency. For instance, among the 50 basic blocks being represented by the vertical bar at $\eta = 0.23$ in Figure 2, 20 of them might have a latency of 10 cycles, 20 a latency of 5 cycles and the remaining 10, 3 cycles. Two distinct vertical bars with the same height should therefore not be impartially handled.

A more accurate metric is to use an estimation of the energy consumption corresponding to each vertical bar on Figure 2, thus creating a *energy distribution graph*. This estimation does take into account the total latency for each bar. More precisely, by using execution traces and some profile information, we compute an energy-corresponding value while still assuming that any slowdown factor is possible. We call this value as the *optimal* (i.e., ideal) energy consumption, in the sense that we still assume a processor with any possible number of operating frequencies.

The energy value is proportional to $TotalCycles_j \eta_j V_j^2$, where $TotalCycles_j$ is the total number of cycles for vertical bar j, and V_j is η_j's corresponding voltage. Actually, since each individual η_j has a specific operating voltage V_j, we simplify this step by assuming that the energy is proportional to $TotalCycles_j \eta_j^3$.

Figure 3 shows the energy distribution graph resulting from Figure 2. Note that it is very similar to Figure 2, since it is simply a *weighted* frequency distribution.

The next and final step of Phase 2 relies on the *cumulative energy distribution* graph built on top of the energy distribution graph shown in Figure 3. This is straightforwardly done by iteratively accumulating each vertical bar's energy from $\eta = 0$ up to $\eta = 1$. Figure 4 shows the cumulative energy distribution graph corresponding to Figure 3. Each pair (x_1, y_1) on the resulting curve tells us that y_1 percent of the total energy consumption are due to basic blocks with a slowdown factor less than or equal to x_1.

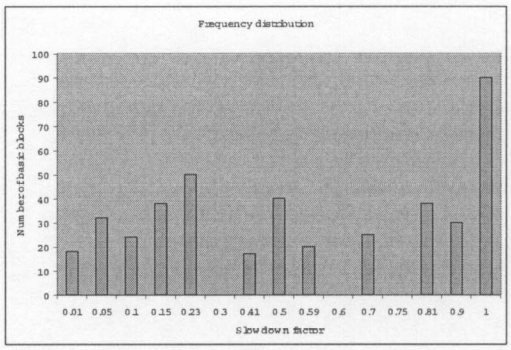

Figure 2: Frequency distribution graph.

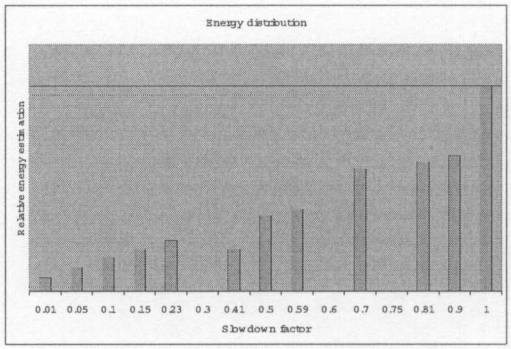

Figure 3: Energy distribution graph.

In order to find the actual set of N operating frequencies, we divide the 0-to-100% range of the y axis into N equal intervals, further projecting the corresponding $y_1, .., y_n$ values onto the x axis. The intuition here is that we want to find those particular slowdown factors that "accumulate" each n-*th* interval. The key fact for this assumption is that the influence of those basic blocks located at the lower end of the x axis (in Figure 4) is much smaller than that of those blocks located at the upper end. In other words, since a small η will have a corresponding small voltage level, the impact of a particular basic block on the total energy consumption is not linear along the x axis on Figure 4, but in fact depends on its actual position on that axis. This will lead into a non-uniform distribution of operating frequencies, often clustering them at the upper end.

This is a major observation not taken into account by current voltage scaling strategies. That is to say, existing techniques consider a pair (f_{min}, f_{max}) and then divide this range into a number of equally sized intervals. However, in an application-specific voltage scaling point of view, operating frequencies might turn out to have a non-uniform pattern in order to minimize the total energy consumption.

In summary, the goal of phase 2 is to find the *characteristic curve* for a given application. Different applications will have different curves, depending on the application's charac-

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

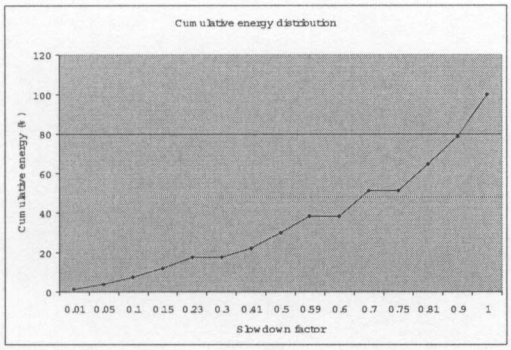

Figure 4: Cumulative energy distribution graph.

teristic (data-flow dominated, control-flow dominated, etc.) The operating frequencies are further extracted from this characteristic curve.

4.3 Phase 3: Speed Reassignment

Recall that the goal of Phase 2 was to find a tailored and limited set of operating frequencies, a constraint, for a given benchmark. The input to Phase 2, in turn, was the LP results of Phase 1, which generated an ideal number of slowdown factors for the basic blocks regardless of such constraint. Therefore, an extra adjustment is required in order to re-assign only valid operating frequencies for the entire set of basic blocks within the application. In this paper, we use a conservative approach where we use the smallest operating frequency that is still greater than the block's "optimal" slowdown factor. This assumption guarantees that no basic block will be over-delayed.

5. EXPERIMENTAL RESULTS

We have selected eleven different benchmarks for our experiments, namely those listed in Table 1. Most of these benchmarks are from MiBench [18], Powerstone [8] and MediaBench [20]. All our tools have been built on top of the WARTS/EEL platform [6].

Our tool flow is as follows. Each benchmark is compiled by `gcc`. A prototype analyzer, written by us, accepts assembly language code, disassembles it, identifies the basic blocks, and constructs the control flow graph (CFG) for each benchmark. Then, given the benchmark's CFG, our analyzer automatically generates the set of LP equations into a (".lp") file which also includes the necessary bounds and objective function. The `cplex` solver is subsequently used to solve the linear system and to emit the output (".output") file with the corresponding results for each block's slowdown factor η_i' and start time S_i. Each benchmark takes approximately 1 second to be analyzed and solved by `cplex`. The next step is to read the LP results from all ".output" files and back-annotate each benchmark's CFG. Finally, the frequency distribution graph, energy distribution graph and cumulative energy distribution graph (see Phase 2) are computed by looking at all benchmarks results. The cumulative energy distribution graph is further used in order to find a tailored set of N operating frequencies, as explained in the previous section. The three-phase approach is repeatedly exercised for $n = 2, 3, 4, 8, 12$ and 16.

Program	Description
lame	MP3 encoding engine
jpeg	Image compression and decompression
compress	Data compression tool
engine	Engine controller
mpeg	Video encoder
madplay	MPEG audio decoder
tiff2ps	tiff to postscript conversion
gsm	Speech transcoding
basicmath	Basic math operations
adpcm	Voice encoder
dijkstra	Shortest path algorithm

Table 1: Description of benchmark programs.

For practical reasons, it is worth pointing out that, a slight modification on Phase 2 was actually implemented in our experimental framework. Recall that in Phase 2 the y axis of the cumulative energy graph was divided into N equal intervals, and the corresponding x values were collected to form the set of N tailored operating frequencies. In our actual implementation, however, we took a more aggressive (and greedy) approach. Instead, we divide the 0-to-100% range into $m >> n$ segments. We subsequently collect all m operating frequencies at $x_1, x_2, .., x_m$ and generate all possible combinations of m elements taken N at a time. For each combination, we computed an estimate of the *actual* energy consumption that would result if that specific combination was chosen to be the final set of N operating frequencies. This is done by (a) carrying out Phase 3 over each combination and (b) using the set of equations presented in the previous section to compute the energy estimates. Intuitively, we do an exhaustive search in a broader sub-set of all slowdown factors in order to select the specific combination that results in the lower energy consumption. Of course, there is a computational effort limit on the value of m (we have adopted $m = 32$).

The effectiveness of our technique is evaluated by comparing the results of (tailored) voltage-scaled systems against those of fixed-voltage systems. Figure 5 shows the amount of energy savings that can be achieved for the benchmarks listed in Table 1 as the number of operating frequencies grows from 2 up to 16. The results shown are relative to running each benchmark at maximum processor speed only, or 1 operating frequency. Note that these results are generated under the assumption that $deadline = WCEC$ for each benchmark. In other words, a negligible amount of savings would be expected. However, our experiments show that savings up to 20% can be reached even under this tight constraint.

More realistic scenarios, where a slack time is allowed for each benchmarks, clearly would give much better results, as shown in Figure 6. This figure depicts the different amount of energy savings when incremental degrees of slack are allowed for *every* task in the `tiff2ps` benchmark (8 operating frequencies). `WCEC` corresponds to our preliminary assumption that $deadline = WCEC$, whereas the other values are increments of 20, 50, and 100% over the WCEC. Allowing exactly the same relative amount of slack for every task in the application, though, is not quite realistic.

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

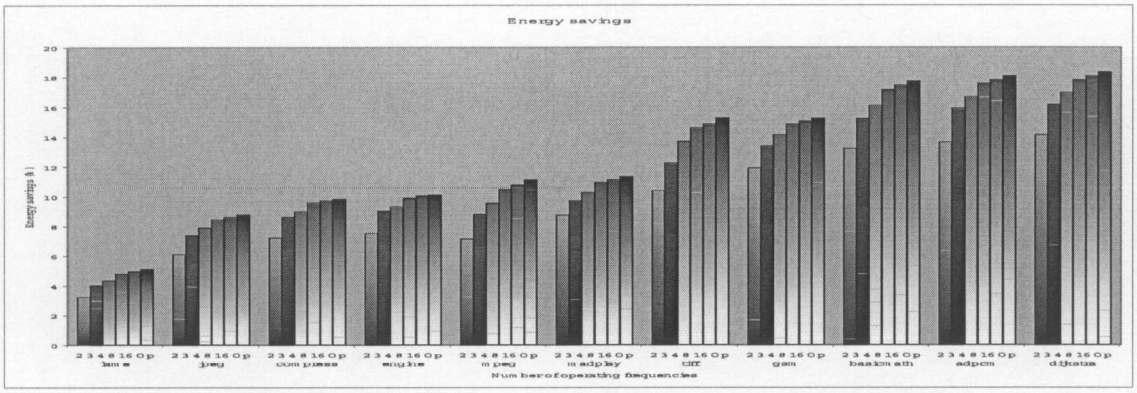

Figure 5: Energy savings for the benchmarks listed in Table 1.

As can be seen in Figure 5, the energy savings afforded by our technique (and voltage-scaling in general) are extremely application dependent. The same observation has been verified in previous works [11]. For some applications, savings of only 5% are observed, while others present values such as 20%. The last vertical bar in each benchmark (labeled "*Op*") corresponds to the "optimal" amount of savings that could be achieved for that benchmark. By "optimal" we mean that an unlimited number of operating frequencies could be provided by the processor.

Observe that the amount of energy savings increase as the number of operating frequencies is incremented. This leads us to the conclusion that there must exist a particular number of operating frequencies resulting in the best trade-off between energy savings and implementation issues. Our experiments show that 8 operating frequencies are generally sufficient for the vast majority of the benchmarks analyzed.

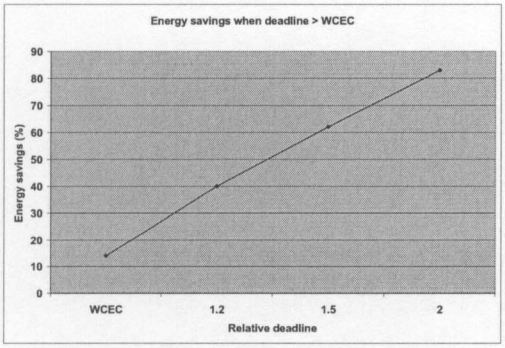

Figure 6: Energy savings for tiff2ps (8 operating frequencies) when the deadline is expanded.

For a given pair (benchmark, N), another set of interesting results is associated with the specific placement (i.e., distribution) of the operating frequencies that returned the greatest amount of energy savings. Table 2, Table 3, Table 4, and Table 5 illustrated that optimal (i.e., ideal) operating frequencies are usually non-uniformly distributed along the frequency spectrum, suggesting a different perspective for DVS strategies. For example, when $n = 4$ operating frequencies are sought for jpeg benchmark, the obtained values are $0.37f_{max}$ $0.623f_{max}$ $0.86f_{max}$ and $1.0f_{max}$.

f_1	f_2
0.627	1.0

Table 2: Operating frequencies placement for jpeg considering 2 points.

f_1	f_2	f_3	f_4
0.37	0.62	0.86	1.0

Table 3: Operating frequencies placement for jpeg considering 4 points.

f_1	f_2	f_3
0.46	0.8	1.0

Table 4: Operating frequencies placement for mpeg considering 3 points.

In summary, in embedded system design, where the application set is a priori known, our extensive experiments confirm that careful selection of the number and distribution of operating points in a DVS processor can result in a more efficient hardware architecture as well as more effective software scheduler.

6. CONCLUSION

In this work, we have presented a design space exploration strategy for DVS sub-system of embedded soft cores. Specifically, we have demonstrated that the choice in the number of processor operating points N and the distribution of these points along the voltage axis is a significant design decision. Moreover, we have argued and experimentally verified that these choices should be part of the design space of a processor soft core and with respect to the application that is eventually to be executed on the processor. As with other customization potentials (e.g., data-path and memory tuning, instruction-set tailoring, register file sizing, etc.), the DVS subsystem of a processor soft core needs to be explored in an iterative codesign methodology.

Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)
0-7695-2044-8/03 $ 17.00 © 2003 IEEE

f_1	f_2	f_3	f_4
0.46	0.66	0.86	1.0

Table 5: Operating frequencies placement for mpeg considering 4 points.

Our future work will focus on incorporating more flexible task timing specification, which would inevitably broaden the design space and facilitate richer power/performance tradeoffs. Include the enery/timing costs for frequency and voltage changes is also an additional step required for a more precise and realistic scenario.

7. ACKNOWLEDGEMENT

This work was generously supported, in parts, by an NSF (grant number 0205712) and CNPq Brazilian Research Council (grant number 200346/01-6).

8. REFERENCES

[1] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *28th Hawaii International Conference on System Sciences (HICSS'95)*, pages 288,297, 1995.

[2] T. Cruose. Transmeta inc. http://www.transmeta.com/technology.

[3] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *Proc. 1st Int'l Conference on Mobile Computing and Networking*, Nov. 1995.

[4] I. Hong, M. Potkonjak, and M. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processors. In *IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1994.

[5] C. Hwang and A. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 28,32, 1997.

[6] J. Larus. Wisconsin architectural research toolset. http://www.cs.wisc.edu/ larus/eel.html.

[7] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Design Automation Conference*, 2000.

[8] A. Malik, B. Moyer, and D. Cermak. A low power unified cache architecture providing power and performance flexibility. In *International Symposium on Low Power Electronics Design*, 2000.

[9] MIPS. Mips inc. http://www.mips.com.

[10] S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, 1997.

[11] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the iparm microprocessor system. In *ISLPED'00*, pages 96,101, 2000.

[12] I. P. Processors. Intel inc. http://www.intel.com/mobile/processors/pentiumIII.htm.

[13] I. X. Processors. Intel inc. http://developer.intel.com/desing/intelxscale.

[14] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the 38th Conference on Design Automation*, pages 828,833, 2001.

[15] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Design Automation Conference*, 2001.

[16] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems*, 4(1):42,55, 1996.

[17] I. StrongARM. Amd inc. http://www.amd.com/armtech/StrongARM.

[18] M. B. Suite. University of michigan, ann arbor. http://www.eecs.umich.edu/mibench.

[19] Tensilica. Tensilica inc. http://www.tensilica.com.

[20] L. A. University of California. Mediabench benchmark suite. http://cares.icsl.ucla.edu/MediaBench.

[21] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. 1st Symp. on Operating Systems Design and Implementation*, pages 13,23, Nov. 1994.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Foundations of Computer Science*, pages 374,382, 1995.