An intra-task DVS algorithm exploiting path probabilities for real-time systems

G. Sudha Anil Kumar and G. Manimaran
Real Time Computing & Networking Laboratory
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA
Email: {anil,gmani}@iastate.edu

***Abstract*— In this paper, we present a novel intra-task Dynamic Voltage Scheduling (DVS) algorithm based on the knowledge of frequently executed paths in the control flow graph for real time systems. The basic idea is to construct a common path composing all the frequently executed paths (hot-paths) and perform DVS scheduling based on this common path, rather than the most probable path. We compare the performance of the proposed algorithm with an existing algorithm. The preliminary results show that the proposed algorithm performs better than the existing algorithm.**

I. Introduction

Portable embedded devices, such as personal digital assistants, mobile phones and palmtops have become extremely popular in the recent past. These devices rely on batteries for power supply and their operation is limited by the available battery life. Therefore, efficient utilization of energy is one of the key challenges in the design and operation of embedded devices. Most of the embedded processors are based on CMOS technology, where the energy dissipated per cycle is directly proportional to the square of the supply voltage,V_{dd} [8]. A widely used technique that exploits this characteristic is the DVS, whose goal is to choose the supply voltage and operating frequency as per the performance level required by the tasks. Several energy aware DVS algorithms have been proposed for real-time systems[1], [6], [3].

The existing real-time DVS (RT-DVS) algorithms can be broadly classified into two categories based on the system's constraints:

- Energy aware algorithms: The objective of this class of algorithms is to minimize the energy consumption while meeting all the deadlines. The algorithms proposed in [1] belong to this class, where the goal is to dynamically exploit the slack created due to early completion of tasks.
- Reward aware energy constrained algorithms: The objective of this class of algorithms is to meet as many deadlines as possible and maximize the total reward of the system while operating within a given energy budget. The heuristic algorithms proposed in [6] belong to this class.

The RT-DVS algorithms can further be classified into intra-task and inter-task DVS algorithms based on the granularity at which the voltage scaling is performed. The intra-task voltage scaling algorithms [2], [3], [9] adjust the supply voltage within a task boundary. The inter-task voltage scaling algorithms [1], [6] perform voltage scaling on a task by task basis.

II. Background and Motivation

Intra-task DVS algorithms typically work with the control flow graph (CFG) of the real-time programs. CFG represents the block level control flow structure of the program. Each node in the CFG of the program denotes a basic block of computation. The edges in the CFG indicate the control dependency between the blocks.

The objective of an intra-task voltage scheduling algorithm for real-time programs is to assign proper clock frequency to each of the basic blocks so as to minimize the total energy consumption while meeting the task deadline. Ideally, each basic block can be operated at any voltage point which lies in the operational range of the processor. However, current commercial processors supply a fixed number of discrete voltage (and corresponding frequency) levels [10]. Therefore, each basic block needs to be operated in one of the discrete supply voltage levels. In this paper, we assume the processor supports a fixed number of supply voltage (and corresponding frequency) levels.

A. Related Work

Lee et. al. [7] introduced the basic idea of intra-task voltage scheduling. Shin et.al. [2] extended this work with a worst case execution path based scheme which does not consider the likelihood of different possible execution paths. However, programs typically display a high degree of path locality, that is, only a small fraction of total possible paths execute most of the time [5].

Seo et al.[9] take the path locality into account by considering the branch probabilities of the CFG. Based on the branch probabilities, the proposed algorithm achieves optimal average energy savings. However, obtaining all the branch probabilities for a large program (with large degree of path locality) is impractical. On the other hand, the less detailed information like the most frequently executed paths (hot-path information) is much easier to obtain as it incurs less profiling.

Shin et. al. [3] proposed a hot-path information based intra-task DVS scheme (RAEP), which chooses one of the hot-paths and perform voltage scaling at each basic block that gives the best possible energy savings when the chosen

path is executed. However, this heuristic scheme does not always achieve the minimum energy consumption, since there can be more than one hot path.

B. Motivation

The optimal frequency with respect to a particular execution path in the CFG depends on its length, where path length is defined as the execution time of the path when operated at the maximum frequency. Therefore, operating at a frequency that is based on the lengths of the hot-paths results in the best energy savings.

The RAEP algorithm takes the above approach by considering one of the hot-paths (most probable hot path). It operates at a frequency closest to the optimal frequency of the chosen path. However, there could be several hot paths of varying lengths which would mean the non-chosen hot-paths may together contribute a higher probability of execution.

Consider the following example with (CFG shown in figure 1) three hot-paths: ($p_1(B1, B2, B8)$, $p_2(B1, B3, B8)$, $p_4(B1, B5, B8)$). Let the respective execution probabilities be 0.35, 0.30 and 0.30. The probabilities for the other paths are unknown. For this example, the RAEP considers path p_1 only, though it contributes less to the total energy consumption as compared to executing paths p_2 and p_4 together. Therefore, considering just the most probable hot-path may not be very effective in energy savings.

Consider a slightly different situation, in which the CFG has two equally likely hot-paths of considerably different path lengths and hence different optimal frequency values. Operating the program at a frequency based on just one of them will result in considerable energy loss if the program executes the other hot-path. This example further motivates to perform DVS scheduling considering all the hot-paths rather than a single hot-path.

In this paper, we present an energy aware intra-task DVS algorithm which considers all the hot-paths together. The algorithm is described in the next section.

III. Common hot path (CHP) based intra-task algorithm

The CHP algorithm considers all the hot paths together. The basic idea is to combine all the hot-paths into a single common path which represents a (virtual) hot-path that is common in length to majority of the hot-paths.

The common hot path is formed by first composing all the paths into a single path of computation called the common-total-path, tp_c, which represents the longest path that the program can ever take. Each computation unit in the tp_c is contributed by one or more paths in the CFG. In figure 1, the first 15 units of the tp_c are contributed by all the paths, while the last 10 units of the middle 110 units in the tp_c are contributed by the path $p_5(B1, B6, B8)(110units)$ alone.

The computational units contributed by majority of the hot paths constitute the common-hot-path, hp_c. In figure 1, the highlighted 130 units of computation, form the hp_c. The hp_c represents the path that is common to majority of the hot paths. Therefore, performing DVS scheduling based on this common hot path length would be beneficial to all the hot paths rather than a single hot-path.

We use the following notations in the rest of the paper:

- D: deadline of the task.
- t_c: current time.
- t_l: time remaining until the deadline, $(D - t_c)$.
- $l(p_i)$: length of a particular path p_i.
- n_h: number of hot paths.
- f_i: normalized frequency(normalized with respect to the maximum frequency) at which the basic block b_i is operated.
- $wcet(b_i)$: represents the worst case execution time of the task starting from block (b_i) to the completion when operated at the maximum frequency.

Following is the detailed description of the CHP algorithm. The algorithm traverses the CFG in a breadth first search fashion and assigns the operating frequency for each basic block.

The CHP based Algorithm

Input: CFG graph, List of hot-paths, processor frequency levels

Output: Frequency assignment to each basic block.

Algorithm steps:

For each basic block b_i, perform the following four steps:

- step 1: Find the $wcet(b_i)$ and construct a single path of length equal to $wcet(b_i)$. This forms the tp_c of block b_i.
- step 2: The computation units of the tp_c which are common to at least $n_h/2$ hot-paths are recognized (and marked) as the common-hot-path, hp_c. The sum of all the marked computational units forms the path length of the hp_c.
- step 3: The operating frequency for b_i is chosen so as to operate the common-hot-path at its minimum possible frequency(normalized), which is given by:

$$f_i = \frac{l(hp_c)}{t_l - (l(tp_c) - l(hp_c))} \quad (1)$$

- step 4: The smallest discrete frequency level which is greater than (or equal to) f_i is chosen as the b_i's operating frequency.

The asymptotic run time of this algorithm is $O(v + e)$, where v and e represent the number of basic blocks and number of edges in the CFG respectively.

A. Illustrative Example

Consider the CFG shown in figure 1 with three hot paths. Paths $p_1(B1, B2, B8)$, $p_2(B1, B3, B8)$, and $p_4(B1, B5, B8)$ are the hot paths with $p2$ being the most probable hot path. The execution probability of the paths p_1, p_2 and p_4 are 0.35, 0.30 and 0.30 respectively. The probabilities of the other paths are unknown. In this example, we assume the processor can operate at any of the ten equally spaced discrete frequency levels (normalized with respect to the maximum frequency)in the

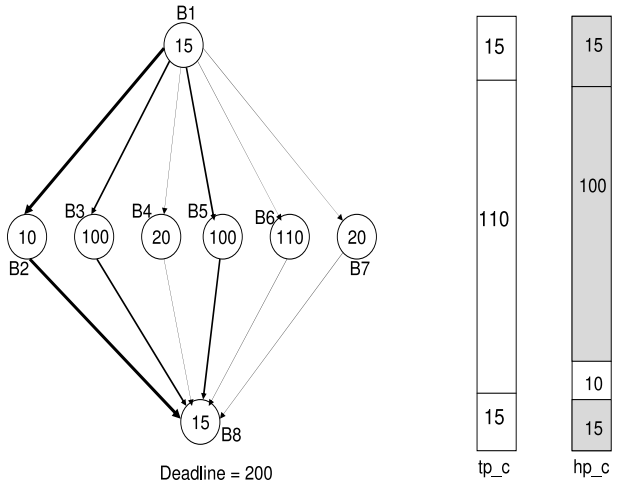

Fig. 1. Working of the CHP algorithm

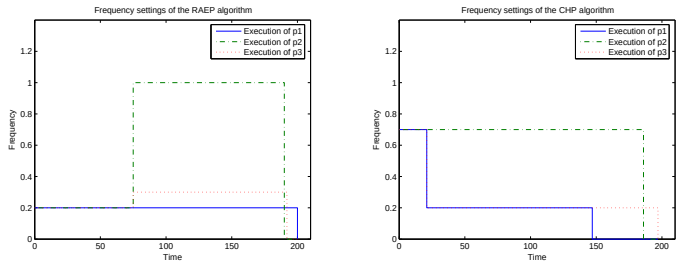

Fig. 2. Frequency settings of RAEP and CHP algorithms

range $[0.1, 1.0]$. The numbers in each basic block represent the computation time of the block when the processor is operated at the maximum frequency.

The RAEP calculates the frequency of each basic block based on the most probable path [3]. The operating frequency of block B1 is calculated as follows:

$$\frac{l(p_1)}{t_l} = \frac{40}{200} = 0.20 \tag{2}$$

Operating $B1$ at this frequency results in operating at the maximum frequency on the execution of longer paths (say p_2 or p_4) as shown in figure 2. Since paths p_2 and p_4 together constitute a higher probability of execution, the RAEP executes at the maximum frequency for most of the program runs. This results in a high average energy consumption.

The proposed CHP scheme calculates the operating frequency of each basic block by considering all the hot paths. The following is the step by step execution of the CHP algorithm for basic block B1:

- The worst case path of the CFG rooted at $B1$ is $(B1, B6, B7)$ with a path length of 140 computation units. Therefore a single path with $l(tp_c) = 140$ is constructed (as shown in figure 1).
- In step 2, the computation units that are common to majority (two in this case) of the hot-paths are marked. This is done in a breadth first search fashion considering just the hot-paths. The block $B1$ is common to all the hot-paths, so the first 15 units (equal to the size of $B1$) get marked in the tp_c. In the next level, two of the hot paths (p_2 and p_4) will execute 100 computational units. Therefore, 100 units are marked as the common (in length) computation units. The block $B8$ is again common to all the hot-paths and hence 15 more units get marked in tp_c. Therefore, $l(hp_c) = 130$. The hp_c is shown shaded in figure 1.
- The operating frequency for the block $B1$ is calculated using equation (1) as:

$$f_i = \frac{130}{200 - (140 - 130)} = 0.68 \tag{3}$$

- The smallest operational frequency level which is greater than 0.68 is 0.7, hence $B1$ is operated at 0.7.

The program continues to execute at the same frequency if it executes one of the two hot-paths p_2 and p_4. On the other hand, it reduces the frequency when it executes path p_1. Since, both paths p_2 and p_4 together execute with a higher probability, CHP consumes lesser energy for most of the times the program is run. This results in a lower average energy consumption than the RAEP scheme. For this example, CHP shows an improvement of 40% over the RAEP scheme.

IV. Preliminary Results

A. Performance evaluation

We have compared the performance of the proposed CHP scheme with the existing RAEP scheme. The performance metric is the normalized average energy consumption (normalized with respect to the DVS unaware scheduler).

We have evaluated the average energy consumption of the above schemes on the basic fan graph [4]. The basic fan graph (shown in figure 6) has n_h hot paths and n_l non hot-paths. All the n_h hot paths together execute with a probability of 0.95. The most probable path executes with a probability p_m and has a path length equal to $(l_1 + x + y)$ while the remaining hot paths execute with equal probabilities and each has a length of $(l_2 + x + y)$. Each of the non hot-paths have a path length equal to $(l_3 + x + y)$.

We have assumed $l_1 = 10$ and $x = y = 15$ for all our performance studies and varied the following parameters:

- Hot-path length ratio:

$$r = \frac{l_1}{l_2} \tag{4}$$

- non hot-path length ratio:

$$r' = \frac{l_1}{l_3} \tag{5}$$

- slack factor:

$$s_f = \frac{D - wcet(B1)}{D} \tag{6}$$

- p_m = probability of the most probable path

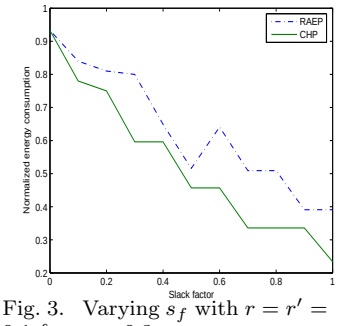

Fig. 3. Varying s_f with $r = r' = 0.1$ & $p_m = 0.3$

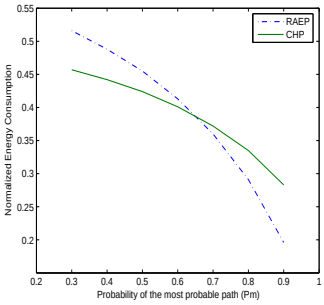

Fig. 4. Varying p_m with $r = r' = 0.1$ & $s_f = 0.5$

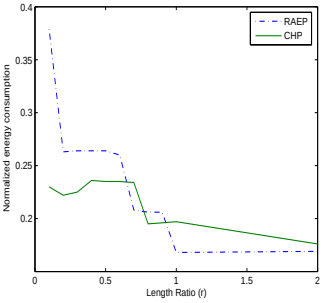

Fig. 5. Varying r with $r' = 0.5$, $p_m = 0.3$ & $s_f = 1.0$

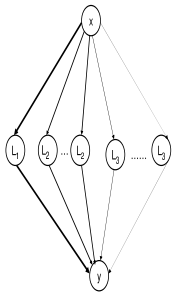

Fig. 6. Basic fan-graph

B. Results

Figure 3. shows the relative performance of the above two schemes varying the slack factor. At slack factor equal to zero, both the schemes behave similarly. In general, with the increasing slack factor both the schemes operate at relatively lower frequencies and hence consume less energy. However, it is interesting to note that the RAEP consumes more energy when the slack factor is increased from 0.5 to 0.6. This counter intuitive behavior of RAEP is due to its greedy nature of lowering frequency.

On the other hand, CHP performs consistently better than RAEP through out the range. It shows about 7%, 25% and 40% improvement at slack factors 0.1, 0.3 and 1.0, respectively.

Figure 4. shows the relative performance of the above two schemes varying the probability of the most probable path (p_m). CHP performs better than RAEP at lower values of p_m, while RAEP performs better at higher values of p_m. This is due to the following reason: as the probability (p_m) increases, the most probable path becomes increasingly important as it contributes more to the average energy consumption; therefore, scheduling based on the most probable path at higher values of p_m will be effective. On the other hand, at lower values of p_m, all the hot-paths are roughly of equal probability; therefore, scheduling based on all the hot-paths would be helpful. Consequently, CHP performs better than RAEP at lower values of p_m and at higher values of p_m RAEP performs better than the CHP. The exact point of crossover is dictated by the path length ratio r.

Figure 5. shows the relative performance of the two schemes varying r. This graph shows the effect of path length variations. At lower values of r, the most probable path is significantly different from other hot-paths, therefore, scheduling based on the most probable path will not be very effective. Consequently CHP performs better than RAEP in the graph. At $r = 1$, all the hot-paths are of equal length, therefore scheduling based on the most probable path will be very effective. Consequently, RAEP performs better than CHP for $r \geq 1$. The crossovers prior to the point $r = 1$ are due to the discrete frequency settings.

V. Conclusion and Future Work

In this paper, we proposed an energy aware intra-task DVS algorithm which takes path locality into account by considering the frequently executed paths. We have evaluated the proposed CHP scheme with an existing algorithm (RAEP) and observed that CHP performs better than RAEP algorithm for the basic fan graph in the following cases:

- when all the hot-paths are (almost) equally likely.
- when the most probable hot-path has considerably different path length than the other hot-paths.
- when the slack is non-zero.

Our performance studies are preliminary and require further analysis. We plan to perform more rigorous performance studies to understand the behavior of the proposed CHP scheme better. We also plan to devise an integrated intra-task DVS algorithm which combines the benefits of RAEP and CHP to offer the best performance for all execution scenarios.

References

[1] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *ACM Symposium on Operating System Principles*, 2001, pp.89-102.

[2] D. Shin, J. Kim, and S. Lee. "Intra-task Voltage Scheduling for Low-energy Hard Real-Time Applications," *IEEE Design and Test of Computers*, Vol. 18, No. 2, 2001, pp.20-30.

[3] D. shin and J. Kim, "A Profile-based Energy-Efficient Intra-task Voltage Scheduling Algorithm for Hard Real-Time Applications," *in Proc. of International Symposium on Low Power Electronics and Design(ISLPED)*, Aug. 2001.

[4] H. El-Rewini and H. H. Ali, "Static Scheduling of Conditional Branches in Parallel Programs," *Journal of parallel and Distributed computing.*, Vol. 24, Jan. 1995, pp.41-54.

[5] T. Ball and J. R. Larus. "Using Paths to Measure, Explain, and Enhance Program Behavior," *IEEE Computer.*, Vol.33, No.7, 2000, pp.57-65.

[6] T. A. AlEnawy and H. Aydin, "Energy-Constrained Performance Optimizations for Real-Time Operating Systems," *Workshop on Compilers and Operating Systems for Low Power (COLP)*, Sept. 2003.

[7] S. Lee and T. Sakurai, "Run-Time Voltage Hopping for Low-Power Real-Time Systems," *in Proc. of ACM Design Automation Conference(DAC).*, 2000, pp.806-809.

[8] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," *in Proc. of International conference on System Sciences.*, Jan. 1995, pp.288-297.

[9] J. Seo, T. Kim, K. S. Chung, "Profile-Based Optimal Intra-task Voltage Scheduling for Hard Real-Time Applications," *in Proc. of ACM Design Automation Conference(DAC)*, June 2004, pp.87-92.

[10] Transmeta Corporation. Crusoe Processor. http://wwww.transmeta.com, June 2000.