

An intra-task DVS algorithm exploiting program path locality for real-time embedded systems

G. Sudha Anil Kumar and G. Manimaran

Real-Time Computing and Networking Laboratory
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA
{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 embedded 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 (energy consumption) of our algorithm with a recently proposed algorithm. Our simulation results show that the proposed algorithm performs better than the existing algorithm for most of the simulated conditions. We also identify interesting research problems in this context.

1 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} [1]. A widely used technique that exploits this characteristic is the DVS, whose goal is to minimize the energy consumption by choosing 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 [2–6].

The existing real-time DVS (RT-DVS) algorithms can be broadly 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 [3–6] adjust the supply voltage within a task boundary. The inter-task voltage scaling algorithms [2] perform voltage scaling on a task by task basis.

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 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 [7]. 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.

2 Related work and motivation

Lee et. al. [3] introduced the basic idea of intra-task voltage scheduling. Shin et.al. [5] 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 [8].

Seo et al.[6] 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 varying 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. [4] proposed a hot-path information based intra-task DVS scheme (RAEP), which chooses one of the hot-paths (where a hot-path is a path that exhibits high execution locality) 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, because there could be more than one hot path.

2.1 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 based on the lengths of the hot-paths results in best energy savings.

The RAEP algorithm takes the above approach by considering one of the hot-paths (the 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. For example, commercial programs like MPEG-4 video decoder & encoder, compress and gcc typically have more than 15 hot-paths [8]. In such programs, the non-chosen hot-paths may together contribute a higher probability of execution and therefore RAEP which chooses just the most probable path cannot be very effective in minimizing the energy consumption.

Consider the following example with three hot-paths (CFG shown in figure 1.(a), the hot-paths are represented with thick edges): $p_1(B1, B2, B8)$, $p_2(B1, B3, B8)$, $p_4(B1, B5, B8)$. 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 savings as compared to executing paths p_2 and p_4 together. Therefore, considering just the most probable hot-path may not be effective in maximizing energy savings.

In this paper, we present an intra-task DVS algorithm which considers all the hot-paths together. The proposed algorithm constructs a common hot-path composing all the hot-paths and performs DVS scheduling based on this common hot-path. We have presented the preliminary idea of this paper in [9].

The proposed intra-task DVS algorithm can handle all possible CFG structures. To demonstrate its wide applicability, we follow the branch graph CFG model introduced in [10] which is typical in expressing structured programming constructs. In this model, a CFG is modeled as a branch graph consisting of a collection of components in series and/or parallel combinations. Each component is a basic fan graph which is defined as follows. A directed acyclic graph with $n + 2$ ($n > 1$) nodes is called a basic fan graph if it has n independent nodes with one common parent and one common child. An example of the basic fan graph with 8 nodes is shown in figure 1.(a).

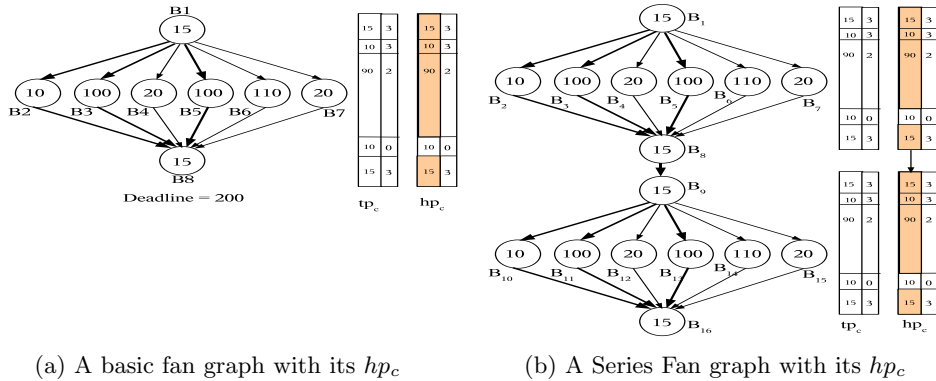


Fig. 1. Working of the CHP algorithm

The rest of the paper is organized as follows. In section 3, we present the algorithm for the basic fan graph with an illustrative example. In section 4, we extend the basic fan graph algorithm and demonstrate its working on more complex CFGs. In section 5, we present our simulation results. Finally, in section 6, we make several concluding remarks.

3 Common hot path (CHP) based intra-task algorithm

The proposed 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.(a), 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 $p_5(B1, B6, B8)$ alone.

The computational units contributed by majority of the hot paths constitute the common-hot-path, hp_c . In figure 1.(a), the number of contributing hot-paths (known as the hot-path count) is shown adjacent to the corresponding computation units. For example, the first 15 units are contributed by all the three hot-paths and therefore, the corresponding hot-path count is three; where as the last 10 units are not contributed by any of the hot-paths and therefore, the corresponding hot-path count is zero. The computational units with hot-path count greater than or equal to two (majority in this case) are marked to constitute the hp_c . In figure 1.(a), the highlighted 130 units of computation, forms 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 based on 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 .
- f_i : frequency at which the basic block b_i is operated.
- $wcet(b_i)$: remaining WCET of the task starting from block (b_i) .

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:

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

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 .
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 .
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:

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

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 time complexity 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.

3.1 Illustrative Example

Consider the CFG shown in figure 1.(a) 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 p_2 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 range $[0.1, 1.0]$. The numbers in each basic block represent the computation time of the block when operated at the maximum frequency.

The RAEP calculates the frequency of each basic block based on the most probable path [4]. The operating frequency of block B1 is calculated as follows:

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

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 $B1$:

1. 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.(a).
2. 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.(a).
3. The operating frequency for block $B1$ is 0.68 calculated using equation (1).
4. 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 compared to the RAEP scheme. For this example, CHP shows an improvement of 40% over RAEP.

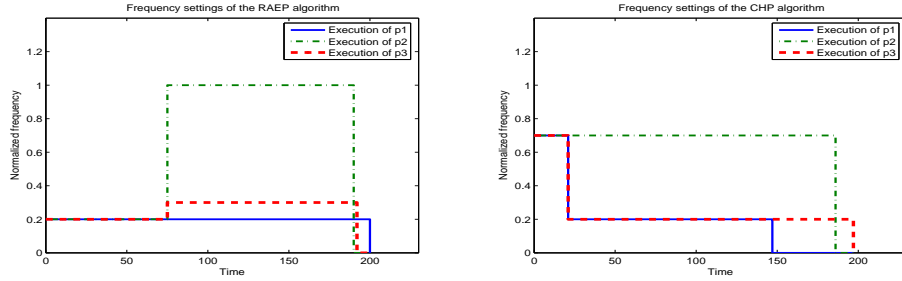


Fig. 2. Frequency settings of the two algorithms for the basic fan graph

4 CHP on complex CFG structures

CFGs of typical application programs will have more complex structure than the basic fan graph. A complex CFG with two or more fan graphs may be viewed as either series or parallel arrangement (or a combination of both) of the basic fan graphs. A CFG can also have loops in addition to the above combinations. The common hot-path formation technique is non-trivial for such complex CFGs and therefore, we demonstrate CHP formation techniques for the complex CFGs. In particular, we present the technique in detail for the following complex CFGs: Series Fan Graph (SFG), Parallel Fan Graph (PFG), Basic Loop Fan Graph (BLFG), Any Combination Fan Graph (ACFG). Once the CHP is composed for a given complex CFG, the operating voltage is determined as in step 3 of the CHP based algorithm (section 3).

The following *generalized_chp* procedure is applied to the complex CFGs:

1. Firstly, recognize all the basic fan graphs in the complex CFG. The basic fan graphs in a CFG can be recognized by determining all the branching nodes which have a unique grandchild. Each such branching node forms the head of a basic fan graph and the grandchild will be the exit node.
2. Secondly, for each basic fan graph recognized, construct the total common path along with the hot-path counts as discussed in the previous section.
3. Thirdly, combine all the tp_c s by taking hot-path counts into consideration to form the final hp_c . The working of this step depends on structure of the CFG, that is, whether the basic fan graphs are in series or parallel. This step is elaborated in detail for each of the above four complex CFGs.

4.1 CHP formation for an SFG

An SFG has two or more basic fan graphs one followed by the other. Figure 1.(b) shows an SFG with two basic fan graphs. Each hot-path in an SFG can be visualized as a concatenation of k partial paths, where the i th ($i \leq k$) partial path is a part of the i th basic fan graph. In the 2-SFG ($k = 2$) shown in figure 1.(b), the hot-path $p_1(B_1, B_2, B_8, B_9, B_{10}, B_{16})$ has two partial paths: $p_{11}(B_1, B_2, B_8)$ and $p_{12}(B_9, B_{10}, B_{16})$. Similar to every hot-path, the CHP (yet to be formed) will have k concatenated partial CHPs each formed independently from each of the k basic fan graphs. Therefore the final hp_c of the SFG can be obtained by concatenating the tp_c s obtained as a result of the first two steps of the *generalized_chp* algorithm and marking the computational units which have hot-path counts greater than $n_h/2$.

4.2 CHP formation for a PFG

A PFG has two or more basic fan graphs as alternatives following a branching basic-block. Figure 3.(a) shows a PFG with two basic fan graphs. The procedure to find the final hp_c of a PFG is little more involved. The basic idea is to find the final tp_c (the longest of all tp_c s) and update its hot-path counts by considering every other tp_c . Once the final hot-path counts are available the algorithm marks the computational units contributed by majority of the hot-paths to obtain final hp_c . We skip further details of this technique due to space constraints. However, we present an illustrative example describing the method as follows.

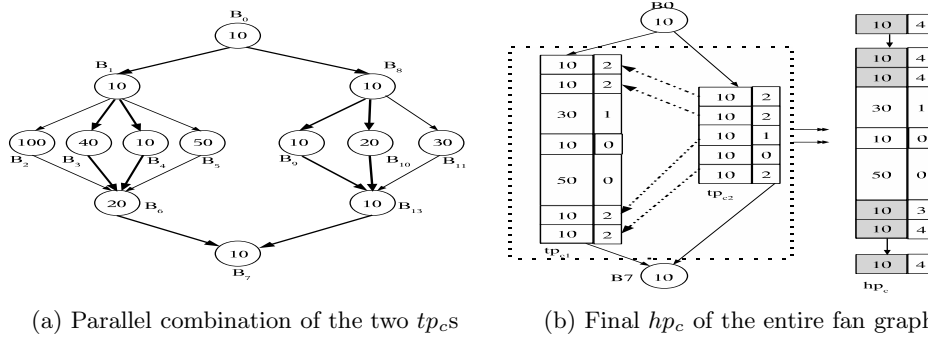


Fig. 3. Construction of hp_c for a parallel fan graph

In figure 3(b), the tp_{c1} being the longest will be the final tp_c . The hot-path counts of tp_{c1} are updated as follows: consider the first computation block of 10 units in tp_{c2} which has a hot-path count of 2 (the largest count), this is combined with the 10 units of tp_{c1} which has the largest hot-path count. This will result in a hot-path count of 4 for the first 10 units in the final tp_c . Similarly, the next 10 units of tp_{c2} are combined with that of tp_{c1} and so on. Finally we will have a single tp_c along with the updated hot-path counts. All the computation units which have a hot-path count greater than $n_h/2$ will form the final hp_c as shown.

4.3 CHP formation for a BLFG

A BLFG has one basic fan graph within a loop. The general procedure to handle a loop is to find the tp_c of the CFG ignoring the loop. Once the tp_c for the basic fan graph is found, the loop has to be unrolled. Since a straightforward loop unrolling can be very expensive, the loop is unrolled twice in a fashion that captures the effect of iterations. We skip further details due to space constraints.

4.4 CHP formation for an ACFG

An ACFG consists of several basic fan graphs arranged in a complicated fashion. Typical application programs fall into this category. Interestingly, any complicated CFG can be viewed as a series-parallel combination of basic fan graphs. Therefore, the hp_c of an ACFG can be obtained by recognizing all the basic fan graphs and the series-parallel relationships between them. More specifically, the procedure to handle ACFGs is as follows: Firstly, find all the basic fan graphs.

Secondly, find the loops in the program and construct their tp_{cs} as in section 4.3. Thirdly, find series basic fan graphs (or their corresponding tp_{cs}) and apply the technique discussed in section 4.1. Finally, find parallel basic fan graphs (or their corresponding tp_{cs}) and apply the technique discussed in section 4.2.

5 Simulation studies

We have compared the performance of the proposed CHP scheme with the existing RAEP scheme and the clairvoyant algorithm. The performance metric is the *normalized average energy consumption* (normalized with respect to the DVS unaware scheduler). The clairvoyant algorithm by definition knows the exact path the program will execute and hence operates at the corresponding optimal frequency. Therefore, clairvoyant algorithm represents the theoretical lower bound of the energy consumption. We have simulated the average energy consumption of the above schemes on randomly generated ACFGs. Each ACFG was generated with 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) while the remaining hot paths execute with equal probabilities and each has a length of $(1 + \alpha)l_2$. Each of the non hot-paths have a path length equal to $(1 + \beta)l_3$. Where α and β are uniformly chosen from $[0, 1]$. We assumed $l_1 = 1000$ for all our performance studies and varied the following parameters:

- Hot-path length ratio: $lr_1 = \frac{l_1}{l_2}$; • Non hot-path length ratio: $lr_2 = \frac{l_1}{l_3}$
- Slack factor: $s_f = \frac{D-wcet(B_1)}{D}$; • p_m : probability of the most probable path

5.1 Results and Discussions

Effect of the path length ratio: Figure 4(a) shows the relative performance of the CHP and RAEP schemes varying the hot-path length ratio (lr_1). This graph shows the effect of path length variations and has three disjoint regions of interest defined by the value of lr_1 . The region with the value of lr_1 very close to one (unity region) corresponds to the case where all the hot-paths have approximately the same path length. In this region, performing DVS scheduling based on the most probable path (or any single path) will be very effective. Consequently, RAEP performs slightly better than CHP in the unity region.

The region which is left to the unity region (left region) corresponds to the case where most of the hot-paths are much longer than the most probable path. In this region, RAEP which considers the (shorter) most probable path alone performs aggressive voltage down scaling in the beginning and ends up increasing the voltage when the other (longer) hot-paths execute. On the other hand, CHP which considers all the hot-paths together performs conservative voltage scaling considering the fact that majority of the hot-paths are long in length.

Similarly, the region which is right to the unity region (right region) corresponds to the case where most of the hot-paths are much shorter than the most

probable path. In this region, RAEP which considers the (longer) most probable path alone performs conservative voltage up scaling in the beginning and ends up decreasing the voltage or even leaving the slack unutilized when the other (shorter) hot-paths execute. On the other hand, CHP which considers all the hot-paths together performs aggressive voltage scaling considering the fact that majority of the hot-paths are short in length.

Effect of the slack factor: Figures 4(b) & 4(c) show the relative performance of the two schemes varying the slack factor (s_f) for different values of lr_1 corresponding to the left and right regions discussed above. We have chosen $p_m = 0.3$ for this set of experiments. In general, with the increasing slack factor both the schemes operate at relatively lower frequencies and hence consume less energy. CHP performs consistently better than RAEP throughout the range. It shows an improvement of 21% at $s_f = 0$ and an improvement of 67% at $s_f = 1.0$ for $lr_1 = 0.3$ case (left region). Similarly, CHP shows an improvement of 27% at $s_f = 0$ and an improvement of 50% at $s_f = 1.0$ for $lr_1 = 1.0$ case (right region).

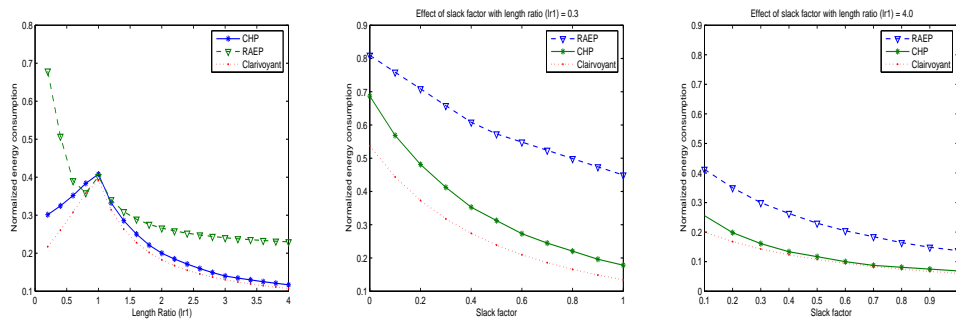


Fig. 4. (a) Effect of path length ratio (b) & (c) Effect of slack factor

Effect of the probability of the most probable path: Figures 5(a) & 5(b) show the relative performance of the above two schemes varying the probability of the most probable path (p_m) for different values of lr_1 corresponding to the left and right regions discussed in the previous result. We have chosen $s_f = 0.5$ for this set of experiments. 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 savings; 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 lr_1 . The crossover point for the $lr_1 = 0.3$ case (left region) is at $p_m = 0.60$ whereas the crossover for the $lr_1 = 4.0$ case (right region) is at $p_m = 0.75$.

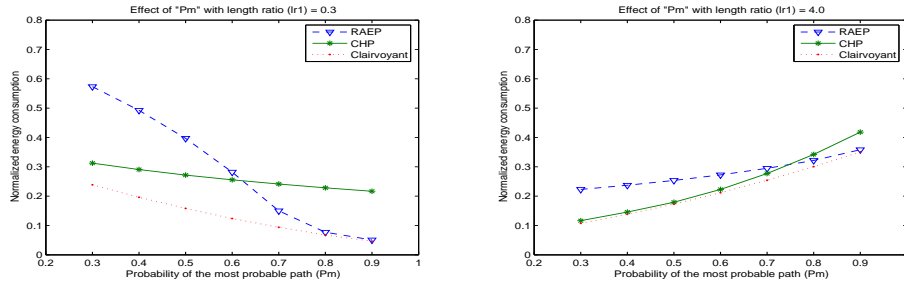


Fig. 5. (a) & (b) Effect of the path probability (P_m) on energy consumption

6 Conclusion and future work

In this paper, we proposed a novel energy aware intra-task DVS algorithm which exploits the knowledge of frequently executed paths. We have evaluated the proposed CHP scheme with an existing scheme (RAEP) and the clairvoyant algorithm through simulation studies on randomly generated ACFGs. We observed that CHP performs better than the RAEP scheme in the following two cases: First, when all the hot-paths are almost equally likely. Second, when the most probable hot-path has considerably different path length than other hot-paths.

We plan to evaluate the proposed scheme on commercial programs like MPEG video decoders to demonstrate its applicability to real world programs. The current scheme results in significant energy gains assuming offline information like the number of loop iterations, etc. In our future work, we plan to relax this assumption.

References

1. 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.
2. 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.
3. 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.
4. D. Shin, J. Kim, and S. Lee. "Intra-task voltage scheduling on DVS-enabled hard real-time systems," *IEEE transactions on CAD*, Vol. 24, Issue 9, Sep. 2005.
5. D. Shin, J. Kim, and S. Lee. "Intra-task Voltage Scheduling for Low-energy Hard Real-Time Applications," *IEEE D & T of Comp.*, Vol. 18, No. 2, 2001, pp.20-30.
6. 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.
7. Transmeta Corporation. Crusoe Processor. <http://www.transmeta.com>, June 2000.
8. T. Ball, P. Mataga and M. Sagiv, "Edge Profiling versus Path Profiling: The Show-down" in *Proc. of ACM SIGPLAN-SIGACT symposium on principles of programming languages*, January 1998, pp.134-148.
9. G. S. Anil Kumar and G. Manimaran, "An intra-task DVS algorithm exploiting path probabilities for real-time systems" *SIGBED Review, special issue on 11th IEEE RTAS Work-in-Progress*, Vol. 2, No. 2, April 2005.
10. H. El-Rewini and H. H. Ali, "Static Scheduling of Conditional Branches in Parallel Programs," *Journal of parallel and Distributed Comp.*, Vol. 24, Jan. 1995, pp.41-54.