Fast and Efficient Voltage Scheduling by Evolutionary Slack Distribution

Bita Gorji-Ara, Pai Chou, Nader Bagherzadeh, Mehrdad Reshadi

Center for Embedded Computer Systems,
University of California, Irvine,
{bgorjiar, chou, nader}@ece.uci.edu, reshadi@ics.uci.edu

David Jensen

Rockwell Collins,
Cedar Rapids, IA
dwjensen@rockwellcollins.com

Abstract - To minimize energy consumption by voltage scaling in design of heterogeneous real-time embedded systems, it is necessary to perform two distinct tasks: task scheduling (TS) and voltage selection (VS). Techniques proposed to date either are fast but yield inefficient results, or output efficient solutions after many slow iterations. As a core problem to solve in the inner loop of a system-level optimization cycle, it is critical that the algorithm be fast while producing high quality results. This paper presents a new technique called *Evolutionary Relative Slack Distribution Voltage Scheduling* (ERSD-VS) that achieves both speed and efficiency. It addresses priority adjustment and slack distribution issues with low cost heuristics. Experimental results from running publicly available testbenches show up to 42% energy saving compared to a published technique called EVEN-VS. It also shows up to 70 times speed improvement compared to an efficient technique called EE-GLSA.

I. Introduction

Design of low power distributed embedded systems is usually an iterative process that includes task scheduling (TS) and voltage selection (VS), in order to meet both performance and power constraints. TS is responsible for meeting the timing and dependency constraints on the tasks, and VS is responsible for the energy reduction through the selection of a voltage-frequency combination (voltage mode) for each of the tasks. The iterative design process makes the runtime of VS and TS algorithms as important as their efficiency. Scheduling of dependent tasks on a multi-processor system is an NP-complete problem [6]. One of the well known heuristics for this problem is priority-based list scheduling, in which the scheduling decisions are made based on priority of the tasks. The priority values are usually calculated based on the tasks execution delay and deadline.

An efficient voltage scheduling algorithm must address two major issues: assigning appropriate amount of slack time to the right set of task (Slack Distribution); and updating task priorities/ordering to make the selected voltage modes schedulable (Task Priority Adjustment); Note that changing the voltage mode of a task during slack distribution will change its execution delay and consequently, its priority. Schedule of the tasks should be updated to accommodate new changes. Addressing the slack distribution and priority adjustment issues appropriately can yield energy efficient results. However, it may increase the complexity of VS and TS algorithms.

Luo and Jha [8] address the slack distribution issue by evenly partitioning slack intervals among the tasks located before the interval. Schmitz et al [9][10] show that this technique is not efficient enough and propose a step by step slowdown technique that sorts tasks based on their energy saving potential. The algorithm then slows down the most energy saving ones by a Δt_{min} (assuming that the corresponding mode is available). They also explore different task priorities using a genetic algorithm (GA) that produces various priority patterns. They have reported up to 43% more energy saving compared to [8], while their algorithm runtime has increased up to 78 times. The complexity of the proposed algorithm is $O(p \cdot i \cdot m \cdot n^2 \cdot \log(n))$, where p is the size of population in GA, i is the number of iterations, n is the number of tasks and m is a factor related to Δt_{min}. Gruian and Kuchcinski [5] have proposed a DVS-based constructive list scheduling technique that dynamically re-computes task priorities based on average energy dissipation. If the found schedule does not meet the specified deadline, priorities of tasks on the critical path are increased and all tasks are re-scheduled. The complexity of their algorithm is $O(V \cdot M \cdot n^3 \cdot \log(\max(a)))$ where n is number of tasks. Bambha et al [1] have used Monte Carlo and simulated heating algorithms to find an optimized voltage mode for each task. They have reported a runtime of several hundred of seconds for the average size testbenches.

This paper presents a new technique called *Evolutionary Relative Slack Distribution* (ERSD) voltage scheduling that efficiently addresses slack distribution and priority adjustment issues. The algorithm iteratively generates lower power mode sets and evaluates their validity by performing a best effort scheduling. To generate a new mode set, the algorithm randomly slows down some of high power tasks. Experimental results from running publicly available benchmarks show up to 42% more energy saving compared to [8]. Also, the results show up to 70 times speedup compared to [10], while generating equally energy-efficient results.

This paper is organized as follows. Section II, presents an example to show the importance of a good slack distribution algorithm. Section III formulates the optimization problem on costs associated with a task graph model. We describe the ERSD-VS algorithm in Section IV, followed by the experimental results and analysis in Section V. Section VI summarizes the contributions and concludes the paper.

II. Importance of efficient slack distribution

Figure 1 shows a simple schedule where two tasks (τ_1 and τ_2) with the execution delay of d are followed by a slack time of the same size (d). We assume that the resource has three modes: m_1, m_2 and m_3; where, m_1 is the fastest mode with supply voltage of 3 volts, frequency of F and power consumption of 1W. The energy consumption of the original schedule is equivalent to the area of the boxes denoted by E_1. There are two possible ways of consuming the slack time:

0-7803-8175-0/04/$17.00 ©2004 IEEE.

assigning it to one of the tasks; or dividing it between the two. In the first case, τ_1 runs in mode m_1, and τ_2 runs in mode m_3. In this way, the energy consumption reduces from E_1 to $0.62 \times E_1$. In the second case, both tasks run in mode m_2, and the overall energy consumption is $0.43 \times E_1$.

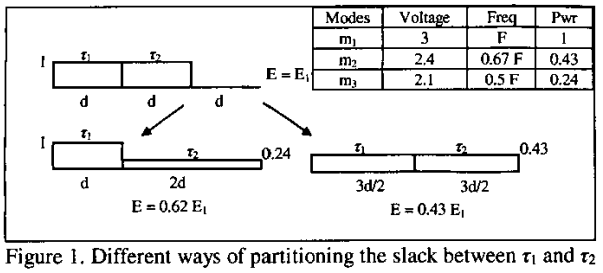

Figure 1. Different ways of partitioning the slack between τ_1 and τ_2

In general, the same amount of slack time can be applied towards saving a different amount of energy, depending on the task that it is assigned to. In this example, there are only two tasks, and their execution delay, power consumption and deadline are the same. However, for heterogeneous systems, the algorithm should handle many tasks with various characteristics. In Section IV, we present our heuristic for addressing this issue.

III. Problem formulation

In this paper, we investigate the voltage scheduling aspect of system synthesis process. Therefore we assume that the allocation of PEs and the mapping of the tasks are already done. A system is usually represented by its application and architecture. The architecture is represented as a set of processing elements *PE* and communication channels *L*. A processing element may operate at different voltage levels and consume different amounts of power. These voltage (power) levels are represented by *voltage modes*. The set of voltage modes for a processing element p_j is given by the non-empty set $VM_j=\{m_{j,1}, \ldots, m_{j,max}\}$. We denote the fastest mode of p_j by $fastestMode(p_j)$. Each voltage mode m has its own frequency, $freq(m)$, and power consumption, $Pwr(m)$. The application is represented by a set of periodic task graphs $\{TG_1, \ldots, TG_N\}$. All task graphs have the same period and their own arrival time and deadline[1]. We represent the task graph set in a directed acyclic graph $G(T, C)$ where $T=\{\tau_1, \tau_2, \ldots, \tau_n\}$ represents tasks and $C=\{c_{i,j} = (\tau_i, \tau_j, \omega_{i,j})\}$ represents data dependency of task τ_j to τ_i (its predecessor); and $\omega_{i,j}$ indicates the communication delay of data to be transferred. A task may be associated with a deadline $\delta(\tau)$ by which its execution must be finished. T_d is the set of tasks τ for which $\delta(\tau)$ is specified. T_d must at least contain *sink tasks* (tasks with no dependents) and if a sink does not have a deadline, then it is assigned the same deadline as its TG. Each task τ is mapped to a processing element $proc(\tau)$, and has an execution delay $t_{exec}(\tau)$ in the fastest mode of that processor. In addition to the mode of the processor, some specific characteristics of individual tasks may affect their power consumption. For example, a specific processor may consume more power for floating point operations than for integer operations. To capture this effect, we represent each task by a *power dissipation factor*, $Pwr\_Factor(\tau)$, that can be extracted through profiling and measurement [2][3].

The goal of the optimization algorithm is to find a mode set M corresponding to the task set T so that the dynamic energy dissipation E_{total} is minimized and no deadline and precedence constraints are violated.

Therefore, we want to minimize:

$$E_{total} = \sum_{i \in [1,n]} \left(Pwr(\tau_i, m_i) \cdot t_{exec}(\tau_i, m_i) \right) \quad (1)$$

Assuming the task $\tau_i \in T$ runs in mode $m_i \in M$:

$$Pwr(\tau_i, m_i) = Pwr\_Factor(\tau_i) \cdot Pwr(m_i) \quad (2)$$

$$t_{exec}(\tau_i, m_i) = t_{exec}(\tau_i) \cdot freq(fastestMode(proc(\tau_i))) / freq(m_i) \quad (3)$$

Further, no hard real-time deadline violation is allowed:

$$\chi(\tau_i, m_i) \le 0, \ \forall \tau_i \in T_d \quad (4)$$

where,

$$\chi(\tau_i, m_i) = t_s(\tau_i) + t_{exec}(\tau_i, m_i) - \delta(\tau_i), \ \forall \tau_i \in T_d \quad (5)$$

Here, t_s denotes the task start time assigned by the scheduler. Note that a positive value of χ shows the amount of deadline miss, while a negative value of χ shows the amount of slack time available after task execution. We denote the overall deadline violation of task set T (running in mode set M) by $\chi_T(M)$, which is defined as the maximum of all deadline violations:

$$\chi_T(M) = \underset{i \in [1,n]}{Max}(\chi(\tau_i, m_i)) \quad (6)$$

Furthermore, the execution order of the tasks and communications must be respected and is expressed as follows:

$$t_s(\tau_i) + t_{exec}(\tau_i) + \omega_{i,j} \le t_s(\tau_j), \ \forall \tau_i, \tau_j \in T, c_{ij} \in C \quad (7)$$

In addition, no execution overlap on any resource $\pi \in (PE \cup L)$ is permitted:

$$interval(\tau_i, \pi) \cap interval(\tau_j, \pi) = \emptyset, \ \forall \tau_i, \tau_j \in T \quad (8)$$

where $interval(\tau, \pi)$ is execution interval of task τ on resource π, represented by the start and end time of the task.

IV. ERSD Voltage Scheduling Algorithm

Figure 2 shows the block diagram of ERSD approach. The ERSD algorithm starts by selecting the fastest mode set, and proceeds by iteratively evolving it into a more energy efficient one. The evolution of the modes is performed by random slowdowns steered by the *slowdown probability* (SDP) values. In each iteration, the execution delays of the tasks are updated for the selected mode (Equation 3). Then, the scheduling algorithm re-calculates the priority values (priority adjustment) and re-computes the schedule. If the generated schedule is valid, then the new mode is selected for the next evolution; otherwise, the last valid mode will be selected. In this section, we first present our heuristic for calculation of slowdown probability and then describe the details of the algorithm itself.

[1] By considering the *hyper-period* of any set of periodic task graphs and repeating them accordingly, such a set can be constructed.

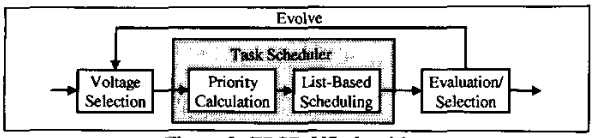

Figure 2. ERSD-VS algorithm

A. *Calculating slowdown probability*

Slowdown probability of a task is used to direct random slowdowns. In heterogeneous embedded systems, different resources have different power consumptions. Tasks that are mapped to high power resources are better candidates for slowdown than those mapped to low power resources. The power consumption of each task depends on the mapped resource, assigned mode and its individual power characteristics.

In each iteration of our algorithm, the SDP of the task τ_i is calculated by:

$$SDP(\tau_i) = \begin{cases} \dfrac{k \cdot Pwr(\tau_i, m_i)}{avePwr} & k \cdot Pwr(\tau_i, m_i) \geq avePwr \\ 1 & otherwise \end{cases} \quad (11)$$

Where,

$$avePwr = \frac{\sum_{i \in [1,n]} Pwr(\tau_i, m_i)}{n} \quad (12)$$

and k is a constant used for adjusting the range of SDP. Note that Equation 11 calculates the proportion of the power consumption of task τ_i to the average power consumption of all tasks. Therefore, it assigns higher values of SDP to high power tasks.

B. *Voltage selection algorithm*

Figure 3 shows the pseudo code of our voltage selection algorithm. It starts by selecting the fastest mode set for the tasks that must be schedulable and has the highest energy consumption. Then it calculates execution time and power of the tasks (line 3) that is used by scheduler (line 4). In each iteration of the loop (lines 5-12), a new mode set is generated from a previous one by random slowdown of tasks (line 6), and is used to produce a new schedule (line 8). If the new mode is schedulable, then it is selected for the next iteration; otherwise, the last schedulable mode is used in the next iteration.

```
01 ERSD_VS ( )
02   optMode = SELECTTHEFASTESTMODESET( );
03   CALCULATEEXECDELAYANDPOWER( optMode );
04   SCHEDULE( optMode );
05   while( noOfIter < 1000 and noOfUselessIter < 100)
06     evMode = EVOLVE( optMode );
07     CALCULATEEXECDELAYANDPOWER( evMode );
08     SCHEDULE( evMode );   //recalculates priorities then schedules
09     noOfIter ++
10     noOfUselessIter ++
11     if (χT(evMode) ≤ 0)   // if evMode is schedulable
10       optMode = evMode;
11       noOfUselessIter = 0
12   return optMode
```
Figure 3. main loop of voltage selection algorithm

In the EVOLVE() function shown in Figure 4, first the slowdown probabilities of all tasks are calculated using Equation (11). Next, if the amount of slack time $\chi_T(M)$ is large, then all the tasks in set T are randomly slowed down (coarse grain slack distribution). If the amount of slack time is small, however, then a subset of tasks $S_d \subset T_d$ is randomly selected and the slack time of each task $\tau \in S_d$ is distributed among its predecessors (fine grain slack distribution). The coarse grain slack distribution operates on all the tasks, while the fine grain one operates on the tasks that have some slack time to further improve the results.

Figure 5 shows the function used to randomly slow down the predecessors of a task. The chance of slowdown is first given to the task itself and then to its predecessors (if there is any more slack time). The RANDOMSLOWDOWN function shows how the random slowdown decisions are made based on SDPs. If a randomly selected number r is smaller than the SDP, then the task is slowed down. The function outputs the amount of consumed slack time.

```
EVOLVE( ModeSet M ){
  calculate SDP for each task using M.
  evMode = M
  if ( χT(M) is not small)
    // randomly slows down all the tasks in T
    RANDOMLYSLOWDOWNALL(T, χT(M), evMode)
  else
    randomly select Sd⊂Td
    for τ∈Sd
      RANDOMLYSLOWDOWNPREDECESSORS( τ, −χ (τ), evMode)
  return evMode
```
Figure 4. evolve function

In the ERSD-VS algorithm, except for EVOLVE(), all the functions in the loop have linear complexity (O(n)). For EVOLVE() function, the complexity of coarse grain slowdowns is linear while the complexity of fine grain slowdown is O($n_d \times (n - n_d)$). Therefore the overall complexity of ERSD-VS is O($i \times n_d \times n$), where n_d is the total number of tasks whose deadline is explicitly specified (including all sink tasks), and n is number of all tasks.

```
RANDOMLYSLOWDOWNPREDECESSORS (task τ, slack , evMode)
  if(slack>0)
    slack = slack − RANDOMSLOWDOWN(τ, evMode)
    for pred ∈ Predcessors(τ)
      if (slack >0)
        RANDOMLYSLOWDOWNPREDECESSORS (pred, slack , evMode)

RANDOMSLOWDOWN(τ, evMode)
  r = generate a random number
  if ( r < SDP(τ) )
      evMode[τ] = evMode[τ] − 1    // slows down one level
      return CALCULATECONSUMEDSLACK()
  return 0
```
Figure 5. Speed up function

V. Experimental Results

The proposed voltage scheduling (VS) approach was applied to a set of TGFF-based testbenches generated by Schmitz et al ([10]). In these benchmarks, the task graphs are mapped to a set of PEs, some capable of DVS. Table 1 shows the results of running different VS algorithms on the testbenches. The second column of the table shows the complexity of each testbench in terms of the numbers of tasks and edges. The third column represents the results of a voltage scheduling algorithm that evenly distributes the slack times (EVEN-VS [8]). The forth column shows the result of EE-GLSA approach [10] that combines an efficient slack

distribution algorithm with a Genetic Algorithm based List Scheduling (GALS). The GALS explores various task ordering and priorities to achieve the maximum saving opportunities. Also, for these testbenches, they assumed that there is no limitation in the number of possible voltage modes. The third and fourth columns are the results reported in [10]. They report a worst-case run-time of 0.23s and 17.99s for EVEN-VS and EE-GLSA respectively by running their C++ code on a Pentium III/750MHz/128MB Linux PC. Among them, EVEN-VS runs faster than EE-GLSA but EE-GLSA generates more energy efficient results.

The fifth column shows the results of ERSD-VS (our voltage scheduling heuristic) combined with a simple priority-based list scheduling that uses an As Late As Possible Schedule (without resource constraint) to calculate priority values. To provide a fair comparison, we consider 20 voltage modes for each processing element. The sixth column shows the run-time of ERSD-VS for different testbenches. The algorithm is implemented in C++ and was executed on a Pentium III/700MHz/128MB Linux PC. ERSD-VS saves up to 42% compared to EVEN-VS, and it is very competitive with EE-GLSA. Note that while the quality of ERSD-VS is comparable with EE-GLSA (the best published results), it runs 70 times faster than EE-GLSA.

Test benches	No of tasks/edges	Reported in [10]		ERSD-VS (our approach)	
		EVEN-VS (saving %)	EE-GLSA (saving %)	(saving %)	CPU time (s)
tgff 1	8/9	45.50	71.05	69.63	0.0197
tgff 2	26/43	2.80	26.79	27.1	0.0393
tgff 3	40/77	25.98	69.18	68.86	0.0984
tgff 4	20/33	6.66	12.99	12.6	0.1378
tgff 5	40/77	5.34	17.14	19.15	0.0787
tgff 6	20/26	1.23	1.61	1.59	0.1575
tgff 7	20/27	10.16	29.90	30.41	0.0393
tgff 8	18/26	7.28	13.83	13.77	0.0393
tgff 9	16/15	2.25	24.85	19.31	0.0393
tgff 10	16/21	26.08	35.77	35.08	0.0197
tgff 11	30/29	1.28	16.96	16.83	0.0787
tgff 12	36/50	3.14	5.11	4.99	0.2755
tgff 13	37/36	16.73	20.71	20.48	0.0787
tgff 14	24/33	12.78	28.12	28.3	0.0591
tgff 15	40/63	0.84	4.15	4.3	0.0984
tgff 16	31/56	16.63	29.88	29.22	0.0787
tgff 17	29/56	13.06	22.20	21.4	0.118
tgff 18	12/15	0.00	23.44	22.74	0.0197
tgff 19	14/19	20.63	27.84	26.92	0.0591
tgff 20	19/25	37.77	52.30	47.9	0.0393
tgff 21	70/99	0.07	19.45	20.25	0.0984
tgff 22	100/135	13.48	29.10	33.66	0.1968
tgff 23	84/151	6.70	23.20	26.18	0.2558
tgff 24	80/112	0.06	8.53	10.02	0.0787
tgff 25	49/92	1.50	20.16	23.85	0.0984
Average		11.08216	25.3684	25.38	0.0703

Table 1. Energy savings by running different VS algorithms on testbenches of [10]

VI. Summary and Future Works

This paper presents a new technique called *Evolutionary Relative Slack Distribution* (ERSD) voltage scheduling that addresses both the slack distribution problem and priority adjustment, the two major issues of voltage scheduling. Our heuristic calculates a power-oriented mode transition probability for each task that directs the random *evolution* of mode sets towards the most energy saving mode. Experimental results from running publicly available benchmarks show up to 42% energy saving over a published technique called EVEN-VS. Also, ERSD-VS is competitive with an energy efficient technique called EE-GLSA while achieving up to 70 times improvement in algorithm runtime. Because of its efficiency, ERSD-VS is a good candidate for the inner most loop of the design space exploration algorithms.

References

[1] N. Bambha, S. Bhattacharyya, J. Teich, and E. Zitzler, "Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors". In *Proc. CODES*, pages 243-248, April 2001.

[2]C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "Energy estimation for 32 bit microprocessors". In *Proc. CODES*, pages 24-28, May 2000.

[3]S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits". In *Proc. DAC*, pages 242-247, 1995.

[4]R.P. Dick and N.K. Jha, "MOCSYN: multi-objective core-based single-chip system synthesis". In *Proc. DATE* , pp. 263-270, Mar. 1999.

[5]F. Gruian and K. Kuchcinski, "LEneS: task scheduling for low-energy systems using variable supply voltage processors". In *Proc. ASP-DAC*, pages 449-455, Jan 2001.

[6]M.R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY, 1979.

[7]M. Grajcar, "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system". In *Proc. DAC*, pages 280-285, 1999.

[8]J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems". In *Proc. ICCAD*, pages 357-364, Nov 2000.

[9]M.T. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimisation in distributed systems". In *Proc. ISSS*, pages 250-255, Oct 2001.

[10]M.T. Schmitz and Bashir M. Al-Hashimi, Petru Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems", In *Proc. DATE*, 2002.