An Efficient Voltage Scaling Algorithm for Complex SoCs with Few Number of Voltage Modes

Bita Gorjiara, Nader Bagherzadeh, Pai Chou
Department of Electrical Engineering and Computer Science
University of California, Irvine
{bgorjiar, nader, chou}@ece.uci.edu

ABSTRACT

Increasing demand for larger high-performance applications requires developing more complex systems with hundreds of processing cores on a single chip. To allow dynamic voltage scaling in each on-chip cores individually, many on-chip voltage regulators must be used. However, the limitations in implementation of on-chip inductors can reduce the efficiency, accuracy and the number of voltage modes generated by regulators. Therefore the future voltage scheduling algorithms must be *efficient*, even in the presence of few voltage modes; and *fast*, in order to handle complex applications. Techniques proposed to date, need many fine-grained voltage modes to produce energy efficient results and their quality degrades significantly as the number of modes decreases. This paper presents a new technique called *Adaptive Stochastic Gradient* Voltage and Task Scheduling (ASG-VTS) that quickly generates very energy efficient results irrespective of the number of available voltage modes. The results of comparing our algorithm to the most efficient approaches (RVS and EE-GLSA) show that in the presence of only four valid modes, the ASG-VTS saves up to 26% and 33% more energy. On the other hand, other approaches require at least ten modes to reach the same level of energy saving that ASG-VTS achieves with only four modes. Therefore our algorithm can also be used to explore and minimize the number of required voltage levels in the system.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms: Algorithms and Design.

Keywords: Dynamic Voltage Scaling (DVS), scheduling, power management, optimization, stochastic gradient search, heterogeneous systems, and multi-processor systems.

1 INTRODUCTION

Design of future embedded systems becomes more challenging due to the increasing demand for larger high performance applications. Scaling the technology to deep submicron allows placement of hundreds or even thousands of processing cores on a single chip. Managing dynamic and leakage power at that scale poses a major challenge for future designs. The fact that dynamic power and static power have quadratic and exponential relationship to the supply voltage respectively [22] necessitates voltage scaling in components and subcomponents of a chip. To allow dynamic voltage scaling in each of the on-chip cores, it is required to have many on-chip voltage regulators that can provide DVS modes as well as shutdown mode. Currently on-chip regulators cannot provide shutdown mode, and have low efficiency due to the low accuracy of on-chip inductors [5]. Such limitations can lead to significant reduction in the number and accuracy of the available voltage levels especially in deep submicron and SoCs with many processing cores.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

Design of low power embedded systems is usually an iterative process that explores different *resource allocations* and *task mappings* to meet performance, power and cost constraints. For each of the system configuration generated during design space exploration, the application tasks are scheduled on the mapped resources (task scheduling) to meet real-time deadlines. The available slack intervals in the schedule are utilized by voltage (and frequency) scaling algorithm to reduce energy consumption. Figure 1 shows the flow of a typical system design space exploration process [4] that uses two nested genetic algorithms (GAs) to generate various system configurations with different resource allocations and task mappings. The task scheduling and voltage scaling algorithms are in the inner-most loop of this iterative process and therefore must have a very low algorithm complexity in order to handle large applications with too many tasks.

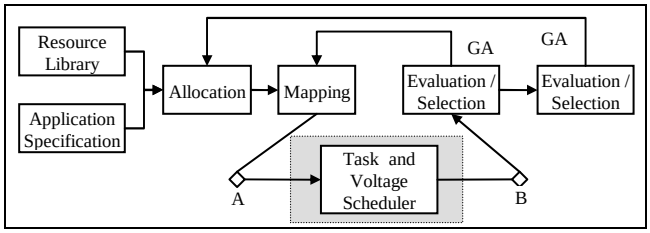

Figure 1. The Design Space Exploration Process proposed in [4]

So far, the algorithms proposed for voltage scheduling either are not very energy efficient, or have a high order of complexity, and/or need many fine-grained supply voltage levels (voltage modes) to generate efficient results. Some of the approaches even formulate the problem for continuous voltage values. If few voltage modes are provided in a system, then they map the generated continuous solution to a valid mode with a relatively high energy penalty.

This paper presents a new technique called *Adaptive Stochastic Gradient* Voltage and Task Scheduling (ASG-VTS) that selects voltage modes for a set of dependent tasks mapped to a heterogeneous system so that the energy consumption is optimized and no real-time deadline is violated. Our algorithm has a low complexity and produces highly energy efficient results even in the presence of few voltage modes. To achieve high energy efficiency, we have developed a discrete stochastic heuristic for slack distribution that is combined with iterative adjustment of task ordering. Whenever a local minimum is found, ASG-VTS stochastically re-claims some of the assigned slack time (slack recovery), and restarts the slack distribution process in order to search a broader space. Our experimental results show that our

proposed search algorithm can quickly produce very energy efficient results. To reduce the number of iteration in each slack distribution cycle, we have carefully defined the *stochastic directors* (mode transition probabilities) based on the energy gradient and the execution delays of the tasks. To further reduce the complexity of slack distribution, we introduce the notion of *time-based relatives* of a task (Section 4.2) as a heuristic that helps finding the candidates for slowdown and speedup quickly and efficiently. We compare our algorithm with two of the most energy efficient approaches: RVS [19] and EE-GLSA [15]. Experimental results from running publicly available tight-deadline benchmarks show that with only four modes to choose from, ASG-VTS can save up to 26% and 33% more energy compared to RVS and EE-GLSA. Also, ASG-VTS can reduce the number of required voltage modes without any significant energy loss. In fact, by reducing the number of modes from thirty to four, ASG-VTS's results are degraded by up to 5.26% (avg. 1.55%) while RVS and EE-GLSA lose up to 28% (avg. 7.3%) and 34.8% (avg. 13.1%), respectively. Furthermore, ASG-VTS runs 2.1 and 150 times faster than RVS and EE-GLSA, and thus it is a better choice for the inner-most loop of the design space exploration. We have also developed a web-based interface for the optimization engine that runs our XML-based system description format.

This paper is organized as follows. Section 2 presents the related works. Section 3 formulates voltage scheduling as an optimization problem. The ASG-VTS algorithm is described in Section 4, followed by the experimental results and analysis in Section 5. Section 6 summarizes the contributions and concludes the paper.

2 RELATED WORKS

There is a large body of work on voltage scheduling of tasks on multi-processor systems. To generate energy efficient results the voltage-scaling algorithm must properly distribute the available slack time among the tasks. Also, the task scheduler needs to explore and adjust the ordering of the tasks to increase energy saving opportunities [14]. Different algorithms have addressed one or both of these issues differently. Luo and Jha [12] propose a voltage scaling algorithm that evenly distributes slacks among the tasks located before the slack interval. The algorithm has a low complexity; but it is not very efficient. Gruian and Kuchcinski [7] have proposed a DVS algorithm that selects voltage modes and decides about the execution order of the tasks based on priority values that are computed from the amount of energy consumption, delays, and deadlines of the tasks. The priority values are iteratively refined until a valid energy-efficient solution is generated. The complexity of their algorithm is $O(n^3)$, where n is number of tasks. Bambha et al [1] have used Monte Carlo and simulated annealing algorithms to find optimized voltage modes for all tasks. The time complexity of each iteration in their algorithm is low; however, in practice, it takes many iterations to converge. As a result, it is very slow: for instance, a runtime of up to 1200 seconds is reported for the testbenches with 14 to 28 tasks. Schmitz et al [14][15] developed an iterative slack distribution technique called EE-GLSA that sorts tasks based on their energy saving potential, and then assigns a slack time of Δt_{min} to the most eligible tasks while avoiding the slack assignments that violate real-time deadlines. In order to maximize energy saving, they apply the voltage scaling algorithm to various schedules generated by a genetic algorithm. They showed that their approach significantly outperforms the above approaches in energy efficiency. The order of EE-GLSA 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}. Since they solve the problem using continuous voltage values, the result must be mapped to the closest discrete modes with some energy penalty. Zhang et al [17] developed an Integer Linear Programming (ILP) formulation to optimally solve the voltage selection problem for a *fixed ordering* of tasks under the assumption of having continuous voltage values. Although they optimally solve the slack distribution problem, the amount of energy saving is bounded by the fixed ordering of the tasks in the schedule. Furthermore, the generated continuous solution can lose its optimality after it is mapped to discrete modes and may be degraded significantly. They did not report the run time of their algorithm. In [11], a similar ILP formulation with continuous voltage levels is used. Since the ILP has a very long runtime, they have used a partitioning heuristic to compromise the optimality with the speed. In [19], a voltage selection technique (RVS) is proposed that randomly distributes the slack time among the high-power tasks over several iterations. It adjusts the ordering of tasks to make the selected modes schedulable. Although the algorithm searches in discrete space, it needs many fine-grained voltage modes to generate efficient results. To the best of our knowledge, both EE-GLSA and RVS generate energy-efficient results and are relatively fast. However, as Section 5 shows that for a limited number of voltage modes, the quality of their results degrades significantly.

3 PROBLEM FORMULATION

This paper investigates the voltage scheduling aspect of the system design process. Therefore we assume that proper processing elements (PE) are allocated and tasks are already mapped to them. A system is usually represented by its *architecture* and *application*. The architecture is represented as a set of processing elements *PE* and communication channels *L*. Processing elements include general-purpose processors, DSPs, FPGAs and ASICs. A PE may operate at different voltage levels and hence 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 denoted by non-empty set $VM_j=\{m_{j,1}, \ldots, m_{j,maxj}\}$. 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, $Pwr(m)$:

$$freq(m) = k \cdot (V_{dd} - V_t)^2 / V_{dd} \quad (1)$$

$$Pwr(m) = C_L \cdot N_{0\to1} \cdot f \cdot (V_{dd})^2 \quad (2)$$

where V_{dd} is the supply voltage, C_L is the switching capacitance, $N_{0\to1}$ denotes the switching activity, k is a circuit-dependent constant and V_t is the circuit threshold voltage.

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*. A task graph is 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 dependencies between pairs of tasks τ_i and τ_j, 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 the deadline of its corresponding TG is assigned to it. 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. We model this effect by *power*

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

dissipation factor Pwr_Factor(τ) that can be extracted through profiling and measurement [2][3]. The total energy consumption of task set T in a selected mode vector M=(m_1,…, m_n) can be calculated by:

$$E_T(M) = \sum_{i=1}^{n} \left(Pwr(\tau_i, m_i) \cdot texec(\tau_i, m_i)\right) \quad (3)$$

where m_i is the mode of $proc(\tau_i)$ during the execution of τ_i and,

$$Pwr(\tau_i, m_i) = Pwr\_Factor(\tau_i) \cdot Pwr(m_i) \quad (4)$$

$$t_{exec}(\tau_i, m_i) = texec(\tau_i) \cdot freq(fastestMode(proc(\tau_i))) / freq(m_i) \quad (5)$$

The overall deadline violation of task set T in mode vector M is calculated by:

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

where $\chi(\tau_i, m_i)$ is the deadline violation of task τ_i in mode m_i:

$$\chi(\tau_i, m_i) = t_s(\tau_i) + t_{exec}(\tau_i, m_i) - \delta(\tau_i) \quad (7)$$

Here, t_s denotes the task start time assigned by the scheduler. Note that a positive value of χ indicates the amount of deadline violation, while a negative value of χ represents the amount of slack time available after task execution.

The goal of the optimization algorithm is to find a mode vector M such that the cost function Ψ is minimized:

$$\Psi(M) = \begin{cases} E_T(M) & if\ \chi_T(M) \le 0 \\ \infty & if\ \chi_T(M) > 0 \end{cases} \quad (8)$$

4 THE ASG-VTS APPROACH

In voltage scheduling the strategy used for slack distribution can significantly effect the amount of energy saving. In general, the same amount of slack time can be applied toward different amount of energy savings, depending on the tasks to which the slack is assigned. Figure 2 shows the flow of ASG-VTS algorithm. ASG-VTS iteratively distributes the slack time. It initially selects the fastest voltage mode and then derives a new mode by slack distribution. Next, it calculates the execution delay and the priority of the tasks for the generated mode, followed by list-based scheduling. If no real-time deadline is violated, then the old mode is replaced by the new one for the next iteration of slack distribution. Otherwise, the new mode is either discarded or is engaged in a slack recovery process. The slack distribution and recovery are performed by adding a *deviation vector* to the current mode. The elements of the deviation vector are assigned −1, 0 or 1 to indicate slowdown, no change, or speedup of a task respectively. These values are stochastically selected based on the *slowdown and speedup probabilities* (SDP and SUP). In this section, we first discuss the details of slack distribution as well as calculation of SDP and SUP. Then we present our task selection heuristic used in slack distribution and recovery. Finally we present the pseudo-code of the algorithm.

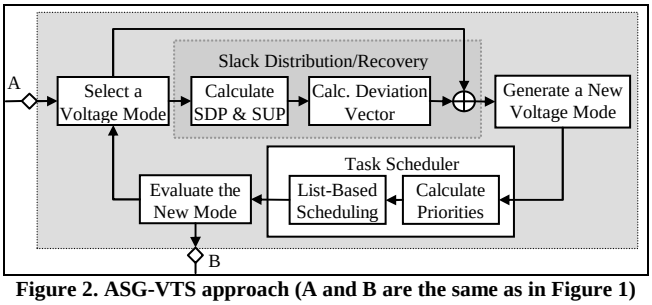

Figure 2. ASG-VTS approach (A and B are the same as in Figure 1)

4.1 Slack Distribution Heuristic

In this paper, we use a variation of stochastic gradient search to explore different ways of slack distribution. Generally, in stochastic gradient search approaches, the probability of changing a variable in each iteration is calculated based on the gradient of the cost function with respect to that change [16]. In other words, the changes that can decrease the cost function more are given a higher chance of occurrence. In our problem, the gradient of the cost function Ψ respective to the change of a mode $m_i \in M$ is calculated using:

$$\frac{\Delta\Psi}{\Delta m_i} = \begin{cases} \frac{\Delta E_T(M)}{\Delta m_i} & if\ \chi_T(M) + \frac{\Delta\chi_T(M)}{\Delta m_i} \le 0 \\ \infty & if\ \chi_T(M) + \frac{\Delta\chi_T(M)}{\Delta m_i} > 0 \end{cases} \quad (9)$$

where $\Delta E_T(M)/\Delta m_i$ is the gradient of energy consumption and $\Delta\chi_T(M)/\Delta m_i$ is the gradient of deadline violation with respect to changing mode m_i. The slowdown and speedup probabilities are defined such that they favor the decrease of $\Psi(M)$. The slowdown probability of task τ_i can be defined as follows:

$$SDP(\tau_i) = \begin{cases} 0 & if\ \frac{\Delta\Psi}{\Delta m_i} \ge 0 \\ norm\left(-\frac{\Delta\Psi}{\Delta m_i}\right) & if\ \frac{\Delta\Psi}{\Delta m_i} < 0 \end{cases} \quad (10)$$

where *norm*() is a normalizing function. This equation means that the tasks that save the most energy and do not cause any deadline violation will be assigned a higher slowdown probability. Although calculation of energy gradient has a low cost (based on Equation 3), calculating the gradient of deadline violation function is very costly because it requires re-running the scheduling algorithm. Therefore, we define the slowdown probability based on energy gradient and a delay factor that is representative of potential deadline violations. Thus:

$$SDP(\tau_i) = energyCnst \cdot norm\left(-\frac{\Delta E}{\Delta m_i}\right) + delayCnst \cdot delayFctr(\tau_i) \quad (11)$$

where *delayFctr* is:

$$delayFct(\tau_i) = 1 - norm\left(\frac{t_{exec}(\tau_i, m_i)}{aveT_{exec}}\right) \quad (12)$$

The $aveT_{exec}$ is the average execution delay of all tasks. The first term of Equation (11) means that the tasks whose slowdown saves the most energy are assigned a higher probability of slowdown. The second term means the tasks with relatively high execution delays are assigned a lower probability of slowdown. Note that this term avoids slowdown of the tasks that already have high execution delays. The constants *energyCnst* and *delayCnst* are used to adjust the effect of each term. We further define speedup probability as follows:

$$SUP(\tau_i) = 1 - SDP(\tau_i) \quad (13)$$

To understand the effectiveness of our SDP formulation, consider the following example: assume that we have two tasks *Task*1 and *Task*2, where *Task*2 consumes more power than *Task*1 while both have the same execution delay. To maximize energy saving a greater portion of the slack time must be assigned to *Task*2. Figure 3.(a) shows how the definition of SDP helps achieving this goal. Here, the slack distribution is performed by iteratively reducing the voltage modes. After each slowdown the SDP of the corresponding task slightly drops because of an increase in execution delay and a small decrease in its energy gradient. As the optimization proceeds, several tasks, including *Task*2 (others are not shown), are slowed down and hence the $aveT_{exec}$ will increase (Figure 3(b)). As a result,

SDP of *Task*1 gradually increases. At some point, *Task*1 will also be slowed down. However, overall *Task*2 is slowed down more often and therefore has consumed a greater portion of available slack time compared to *Task*1.

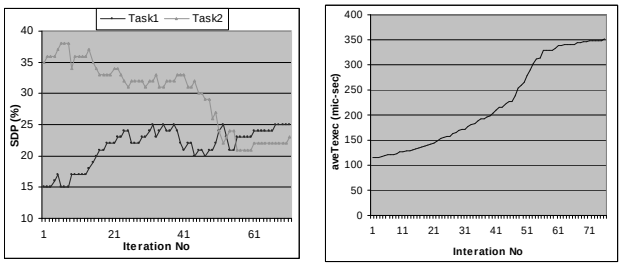

Figure 3. (a) SDP of two tasks and (b) $aveT_{exec}$ in different iterations

4.2 Task Selection Heuristic for Slack Distribution/Recovery

As mentioned earlier, the algorithm determines the tasks that have violated their deadlines ($\chi>0$), and the ones that have some slack time ($\chi<0$). To eliminate the deadline-miss time of a task, ASG-VTS randomly speeds up some of the tasks that have caused the deadline violation (relative tasks). We define *relatives* of a task τ as the set of all tasks whose execution delay will affect the finish time of τ. By definition, whenever a task has missed its deadline, speeding up a subset of its relatives must be able to fix the problem (assuming the application is schedulable in the highest voltage mode). The set of relatives includes predecessors of a task in the task graph as well as its resource-based relatives. We define *resource-based relatives* of a task τ, $R(\tau)$, as those tasks that are mapped to the same resource as τ and are finished between the arrival of τ and its start time. Hence, the finish time of the resource-based relatives of τ will affect the finish time of τ itself. Note that the set of resource-based relatives of tasks may change in different iterations because the slack distribution and recovery affects execution delay of the tasks and their schedule. We recursively define the *relatives* of a task τ as:

$$relatives(\tau) = \{\tau\} \bigcup \{ \bigcup_{s \in Pred(\tau) \cup R(\tau)} relatives(s) \} \qquad (14)$$

where $Pred(\tau)$ is the set of predecessors of τ in task graph. Whenever a task misses its deadline, the set of its relative tasks becomes the candidates for speedup. Also, the slack time after a task can be distributed among its relatives. Note that each task is considered a relative of itself and is therefore a candidate for speedup or slowdown.

Most of the techniques proposed so far extract the set of relatives of a task for slack distribution. However, extracting the set of relatives is complex and time consuming, because keeping track of resource-based relatives requires an additional data structure that captures and updates the links between consecutive tasks mapped to the same resource. Note that the links may change as the slack is iteratively distributed. To avoid the overhead of constructing and updating the links between the relative tasks, we approximate the above relationship with one that is easier to compute: *time-based relationship*.

We define the set of *time-based relatives* of a task τ, $TBR(\tau)$, as the set of all tasks whose finish times lie within the live interval of τ. The *live interval* of a task τ is the interval between the arrival of the host TG and τ's finish time. This set includes some of the real relatives, such as predecessors and resource-based relatives, and some other non-relative tasks. The advantage with this approximation is that extracting time-based relatives is simple and fast. More importantly, it does not require any modification to the task scheduler. Using the time-based relationship reduces the order of slack distribution algorithm at least by a factor of *n*.

4.3 ASG-VTS Algorithm

Figure 4 shows the pseudo-code of our voltage selection algorithm. It starts by selecting the fastest mode vector, which must be schedulable and has the highest energy consumption. The CALCULATEEXECDELAY function (lines 3, 7) calculates the new execution delay of all tasks based on the selected mode vector. The new execution delays are used by the SCHEDULE algorithm to generate a new task schedule. Scheduling of dependent tasks on a multi-processor system is an NP-complete problem [8]. ASG-VTS uses priority-based list scheduling algorithm, which is a well-known heuristic. After generating the initial schedule for the fastest mode (lines 2-4), in each iteration of the loop (lines 5-12), a new mode vector is generated by evolving the previous one (line 6) and is used to produce a new schedule (line 8). If the evolved mode is more energy efficient than the best mode so far (*optMode*), and the new schedule is valid, then the evolved mode is selected for the next iteration. Otherwise (line 13), the selection will be based on a probability function. This function gives a higher chance of selection to the better mode.

```
01 ASG-VTS_VS ( )
02   optMode = M1 = SELECTTHEFASTESTMODESET( )
03   CALCULATEEXECDELAY ( M1 )
04   currSched = SCHEDULE( M1 )
05   while( noOfIter < 10000 and noOfUselessIter < 100)
06     M2 = EVOLVE( M1 )
07     CALCULATEEXECDELAY ( M2 )
08     currSched = SCHEDULE( M2 )
09     if (χT(M)≤0 and M2 is better than optMode)    //no deadline is missed
10       optMode = M1 = M2
11       noOfUselessIter = 0
12     else
13       M1 = select between M1 and M2
14       noOfUselessIter ++
15     noOfIter ++
16   return optMode
```

Figure 4. ASG-VTS algorithm

```
EVOLVE( M )
  Calculate SUP and SDP for each task using mode vector M.
  ΔM = (0, 0, …)                          // initialize the mode deviation vector with 0
  if no deadline is violated
    SLOWDOWN( T, availableSlack, ΔM)   // stochastically slows down all the tasks
  else
    for all τ∈ Td
      if (χ(τ) > 0 )                      // τ has missed its deadline
        if (χ(τ) is small)
          SPEEDUP( Pred(τ), χ(τ), ΔM)
        else
          SPEEDUP( TBR(τ), χ(τ), ΔM)
      if (χ(τ) < 0 )                      // τ has some slack
        if (χ(τ) is small)
          SLOWDOWN( Pred(τ), −χ(τ), ΔM)
        else
          SLOWDOWN( TBR(τ), −χ(τ), ΔM)
  return M + ΔM
```

Figure 5. EVOLVE algorithm

Figure 5 shows EVOLVE algorithm that calculates mode deviation vector ΔM and returns the evolved mode. It starts by calculating speedup and slowdown probabilities of all tasks using Equations (11) and (13). Then, if no real time deadline is violated, it stochastically slows down the entire task set *T*. In case of deadline violation, it performs slack distribution and recovery by using the tasks in T_d as starting points. As defined in Section 3, T_d is a set of tasks whose deadline is explicitly specified and includes all the sink

tasks. For each $\tau \in T_d$, if the amount of deadline-miss (slack) time is relatively small, the speed-up (slow-down) operation is performed only on its predecessors, *Pred*(τ). However, if the amount of deadline-miss (slack) time is large, the mode transitions will be performed on time-based relatives of τ, *TBR*(τ).

Figure 6 shows SLOWDOWN() function that stochastically distributes the *slack* among elements of set *S*. As long as *slack* is not zero, a task τ is selected *stochastically* based on its SDP, and its voltage mode is reduced by one (slowdown). To do so, a random number between 0 and 1 is generated, and if it is less than SDP(τ) then τ is slowed down. SPEEDUP() function is similar to SLOWDOWN() function.

In the ASG-VTS algorithm, except for EVOLVE(), all of the functions in the loop have linear complexity. In EVOLVE(), SPEEDUP() and SLOWDOWN() functions are called for all the members of T_d. The worst case, SPEEDUP() and SLOWDOWN() functions must process all the predecessors or time-based relatives of a task. Therefore the complexity of ASG-VTS is $O(i \cdot n_d \cdot n)$, where i is the number of iterations, n_d is the total number of tasks whose deadline is explicitly specified (including all sink tasks), and n is number of all tasks.

```
SLOWDOWN ( S, slack, ΔM ) {
  while ( (slack > 0 ) and (S ≠ ∅) )
    select τ randomly from S
    r = generate a random number
    if ( r < SDP(τ) )
      ΔM[τ] = − 1
      update slack
    S = S − { τ }
```

Figure 6. SLOWDOWN function

5 EXPERIMENTAL RESULTS

We compare our algorithm to EE-GLSA and RVS in terms of energy savings, runtime complexity and sensitivity to the number of voltage modes. To the best of our knowledge, EE-GLSA and RVS are the best published approaches both in terms of performance and energy savings. EE-GLSA and ILP approaches [17][11] formulate the optimization problem for continuous voltage values. In the mode sensitivity comparisons of this section, we use EE-GLSA as a representative for other continuous approaches. Schmitz et al [15] have presented the results of EE-GLSA on a set of tight-deadline benchmarks using a Pentium III/750 MHz PC. We also used the same set of benchmarks and a similar PC (PIII/700 MHz) to produce the results of RVS and ASG-VTS. The second column of the Table 1 presents the characteristics of the benchmarks in terms of the number of tasks and edges in the task graphs.

The first part of Table 1 (columns 3 to 5) shows the energy saving results of EE-GLSA, using continuous voltage values, as well as that of RVS and ASG-VTS, using 30 fine-grained modes[†]. We observed that using more than 30 modes does not improve the amount of energy savings any further. Note that, when given 30 modes, all three algorithms achieve comparable energy savings. However as the number of modes is reduced the algorithms start to behave differently. Note that even for a small number of modes, exhaustive search is not a viable option. For example, for a system with 20 tasks and four modes the number of possible mode permutations is 4^{20} or 10^{12}. Assuming the optimization algorithm can process 1000 permutations per second, it takes 34 years to find the optimal solution by exhaustively searching the entire space.

[†] The modes are generated by dividing the voltage range into 30 or 4 steps

The second part of Table 1 (columns 6 to 8) shows the energy saving result of the algorithms when only four voltage modes are given[†]. In continuous approaches, such as EE-GLSA, the optimized solution is produced using continuous and potentially unavailable voltage levels. To map the solution to a valid one while meeting all real-time deadlines, Schmitz et al suggest replacing unavailable modes by the next higher available ones. However, this mapping results in energy penalty. In discrete approaches, such as RVS and ASG-VTS, the mapping is not needed because the optimization is based on only the available modes. Furthermore, ASG-VTS applies complementary cycles of slack distribution and recovery, which enables searching a broader space and finding more optimized solutions under the constraint imposed by the limited number of modes. As shown in Table 1, when number of modes is reduced to four, the ASG-VTS loses 1.55% energy in average while RVS and EE-GLSA lose 7.3% and 13.1%, respectively. In the worst case RVS and EE-GLSA lose as much as 28% (tgff1) and 34.8% (tgff10) respectively while ASG-VTS loses only 5.26% (tgff20). This shows that the result of continuous approaches, even if calculated optimally, can not only lose its optimality after the mapping, but also degrade significantly.

Test benches	No of tasks/edges	30 modes (savings %)			4 modes (savings %)		
		EE-GLSA [15]	RVS [19]	ASG-VTS	EE-GLSA	RVS	ASG-VTS
tgff 1	8/9	71.05	69.63	69.89	43.25	41.6	66.67
tgff 2	26/43	26.79	27.1	27.69	0.23	1.03	26.7
tgff 3	40/77	69.18	68.86	71.81	64.72	57.33	66.72
tgff 4	20/33	12.99	12.6	12.71	10.18	11.18	11.83
tgff 5	40/77	17.14	19.15	18.92	0.35	15.44	18
tgff 6	20/26	1.61	1.59	1.54	1.17	1.41	1.53
tgff 7	20/27	29.90	30.41	30.25	0.65	20.15	28.29
tgff 8	18/26	13.83	13.77	14.16	12.84	13.56	13.56
tgff 9	16/15	24.85	19.31	16.1	0.23	9.64	19.19
tgff 10	16/21	35.77	35.08	35.38	0.98	12.28	33.9
tgff 11	30/29	16.96	16.83	15.49	5.67	15.22	16.04
tgff 12	36/50	5.11	4.99	4.99	4.05	3.84	4.31
tgff 13	37/36	20.71	20.48	20.94	10.5	17.66	19.37
tgff 14	24/33	28.12	28.3	28.07	19.42	24.29	26.92
tgff 15	40/63	4.15	4.3	4.47	3.93	4.02	4.3
tgff 16	31/56	29.88	29.22	29.8	17.43	27.95	28.12
tgff 17	29/56	22.20	21.4	21.97	19.27	20.29	20.76
tgff 18	12/15	23.44	22.74	22.45	0.76	19.26	20.48
tgff 19	14/19	27.84	26.92	27.17	23.64	23.53	26.16
tgff 20	19/25	52.30	47.9	50.69	34.61	41.48	45.43
tgff 21	70/99	19.45	20.25	21.37	0.36	2.04	19.89
tgff 22	100/135	29.10	33.66	34.45	24.39	32.49	33.14
tgff 23	84/151	23.20	26.18	25.08	1.34	18.31	21.75
tgff 24	80/112	8.53	10.02	10.22	0.22	3.01	7.81
tgff 25	49/92	20.16	23.85	24.41	0.65	15.14	20.28
Average		**25.37**	**25.38**	**25.60**	**12.2**	**18.08**	**24.05**

Table 1. Energy Savings of all algorithms for testbenches of [15]

Figure 7 shows the average energy saving computed by applying the three algorithms to all of the benchmarks over different number of modes. As shown, the energy saving achieved by ASG-VTS using four modes is achievable by RVS and EE-GLSA using at least ten modes. This also shows that ASG-VTS can reduce the number of required modes without any significant energy loss (1.55% on average).

As mentioned earlier ASG-VTS searches a broader space for better optimization. However, it runs faster than the other algorithms. Our heuristic in calculation of Slowdown Probability (Section 4.1) significantly helps reducing the number of iterations in slack distribution cycles. Furthermore, each iteration in ASG-VTS has a

very low algorithm complexity partly because of our low cost heuristic for task selection (Section 4.2). The reported *worst case* execution time of EE-GLSA is 17.99 seconds while that of RVS and ASG-VTS are 0.255s and 0.12s respectively (in presence of 30 voltage modes and on a similar machine). Therefore, ASG-VTS runs 2.1 times faster than RVS, and 150 times faster than EE-GLSA. Since ASG-VTS has a very short run time, it is ideal for the inner-most loop of design explorations.

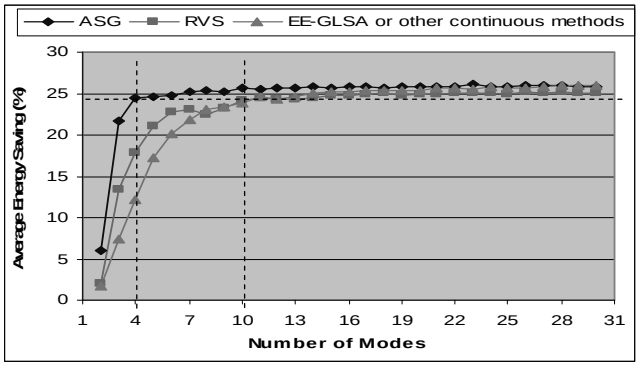

Figure 7. Sensitivity of different algorithms to the number of modes

Table 2. Energy Savings of AS-VTS alg. on E3S benchmarks [6]

Test benches	No of tasks/edges	Saving (%) 30 mode	Saving (%) 4 mode
Consumer (Multimedia)	27/24	63.9	62.6
Networking	31/21	33.24	32.7
Automotive	28/24	27.02	26.2

In addition to the above benchmarks, we ran our optimization algorithm on another set of publicly available benchmarks [6] developed based on data from the Embedded Microprocessor Benchmark Consortium (EEMBC). The benchmarks describe the application task graphs as well as a resource library. For each application, we allocated a set of resources from the resource library and mapped the tasks to them. Table 2 shows the characteristics of benchmarks and the result of optimization. After reducing the number of modes to four, the energy degradation of the algorithm is similar to the previous results.

We have developed a publicly available web interface for the ASG-VTS optimization engine [18]. The users can upload the system description file and run the optimizer to get the optimized modes.

6 CONCLUSIONS

This paper presents a new technique called *Adaptive Stochastic Gradient* Voltage and Task Scheduling (ASG-VTS) that selects voltage modes for a set of dependent tasks mapped to a heterogeneous system so that the energy consumption is optimized and no real-time deadline is violated. Our algorithm has a low complexity and produces highly energy efficient results even when limited to few voltage modes. To achieve high energy efficiency, we have developed a discrete stochastic heuristic for slack distribution, which is combined with iterative adjustment of tasks ordering. Whenever a local minimum is found, ASG-VTS performs slack recovery by stochastically reclaiming some of the assigned slack time, and restarts the slack distribution process to search a broader space. To further reduce the complexity of slack distribution, we introduced the notion of time-based relatives of a task as a heuristic to quickly and efficiently find the candidates for slowdown and speedup. The results of comparing our algorithm to the most efficient approaches (RVS and EE-GLSA) show with only four valid modes, the ASG-VTS saves up to 26% and 33% more energy compared to RVS and EE-GLSA, while being up to 2 and 150 times faster, respectively. Our algorithm is an ideal choice for design space exploration as well as mode exploration.

ACKNOWLEDGEMENTS

This research was sponsored in part by DARPA under contract 4500942474 and in part by NSF under grant CCR-0205712.

7 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. Design Automation Conference*, 1995.

[4] R.P. Dick and N.K. Jha, MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis. In *Proc. Design, Automation and Test in Europe*, 1999.

[5] D. Duarte, Y. Tsai, N. Vijaykrishnan and M. J. Irwin, Evaluating Run-Time Techniques for Leakage Power Reduction, In *Proc. VLSID*, 2002.

[6] Embedded System Synthesis Benchmarks Suite (E3S): http://helsinki.ee.princeton.edu/~dickrp/e3s/

[7] F. Gruian and K. Kuchcinski, LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors. *ASP-DAC*, pages 449-455, Jan 2001.

[8] 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.

[9] M. Grajcar, Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In *Proc. DAC*, 1999.

[10] T. Ishihara and H. Yasuura, Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. ISLPED*, pages 197-202, 1998.

[11] L. Leung, C. Tsui, W. Ki, Minimizing Energy Consumption of Multiple-Processor-Core Systems with Simultaneous Task Allocation, Scheduling and Voltage Assignment, In *Proc. ASPDAC2004*

[12] 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.

[13] A. Manzak and C. Chakrabarti, Variable Voltage Task Scheduling for Minimizing Energy or Minimizing Power. In *Proc. ICASSP*, pages 3239-3242, 2000.

[14] 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.

[15] 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.

[16] J. Spall, *Introduction to Stochastic Search and Optimization*, John Wiley & sons, Inc., 2003.

[17] Y. Zhang, X. Hu, D. Chen, Task Scheduling and Voltage Selection for Energy Minimization, In *Proc. DAC*, June 2002.

[18] http://www.ece.uci.edu/~bgorjiar

[19] Bita Gorjiara, Pai Chou, Nader Bagherzadeh, Dave Jensen, Mehrdad Reshadi, Fast and Efficient Voltage Scheduling by Evolutionary Slack Distribution, In *Proc. ASP-DAC*, 2004

[20] S. Wolf. Silicon processing for the VLSI era Volume 3 - The submicron MOSFET. Lattice Press, 1995, pp. 213-222.