

A Genetic Algorithm for Energy Efficient Device Scheduling in Real-Time Systems

Lirong Tian* and Tughrul Arslan*[†]

* School of Engineering and Electronics, Edinburgh University, UK.
tianlrcn@yahoo.com.cn, Tughrul.Arslan@ee.ed.ac.uk

[†] Jet Propulsion Laboratory, Avionics Section, Pasadena, CA 91109, USA.
Tughrul.Arslan@jpl.nasa.gov

Abstract- Most embedded systems have tight constraints on power consumption because the amount of power available to these systems is limited due to the limitation of battery life. For this reason energy consumption is an important parameter in evaluating performance of embedded systems. DPM (dynamic power management) has gained considerable attention over the last few years as a way to save energy in device that can be turned on and off by operating system control. Scheduling is very important for DPM since it directly affects the efficiency of DPM. We have implemented a Genetic Algorithm customised generates a near-optimal device schedule for a set of real-time tasks, with the goal of minimising the power consumed. When compared with other schedulers, the Genetic Algorithm based system is shown to have less memory and time requirements and scale far better as the problem complexity is increased.

1 Introduction

Power is becoming a critical constraint for designing embedded applications, because the amount of power available to these systems is limited due to limitation of battery life [1]. For this reason energy consumption is an important parameter in evaluating performance of embedded systems. DPM (dynamic power management) has gained considerable attention over the last few years as a way to save energy in device that can be turned on and off by operating system control [2]. Some successful designs have been made using this technique [3,4].

There are several approaches to energy conservation via power management, [5] classifies these approaches into several broad categories which include CPU-directed, I/O-directed, Real-time and Non-real-time, etcetera. Most research on real-time DPM techniques has been CPU-centric [6,7,8]. I/O-centric DPM has been studied extensively for non-real-time environments [9,10,11]. Because the inclusion of deadlines as an added design constraint makes DPM for real-time systems more complex, the non-real-time solution is not suitable for real-time systems. [5] puts forward a Low Energy Device Scheduler (LEDES) for real-time I/O-centric DPM. [12] describes a pruning-based algorithm to adapt LEDES. These schedulers do make improvement in terms of energy efficient in some extent, however these systems do not scale well, in terms of memory and execution time as the number of jobs is increased.

Genetic algorithms are an artificial intelligence tech-

nique based on the principles of evolutionary theory. A series of controlled operations on binary strings are performed, there emulate the biological evolutionary model. This procedure forms an effective search and optimization algorithm, and is used in many real world problems[13]. But it is not extensively used in solving energy efficient problem. In this paper, we investigate a way to implement real-time I/O-centric DPM scheduling by using GA (Genetic Algorithm), and compare the efficiency with other existing schedulers.

2 Problem Description

Unlike a desktop system, embedded systems are more likely to deal with periodic real-time tasks. For such a system, deadlines must be met, so a more accurate schedule is needed for the system. On the other hand as the tasks are periodic, it is possible to have an accurate schedule for these tasks in terms of energy saving. In a multi-tasking real-time system, different tasks usually use different devices, and to save energy, unused devices can be put into sleep mode. However frequently switching a device between the active and sleep states also results in energy consumption. For further energy saving, a schedule for these tasks must reduce the transitions as much as possible. The scheduler we present here takes as input a predetermined set of tasks and a device usage list for each task. The output is a task execution sequence such that the power consumption of the devices is near minimized. We define a set $T = \{t_1, t_2, \dots, t_n\}$ of n periodic tasks. Each task t_i in T , has the following parameters, which are also shown in table 1:

- periodic time p_i ;
- execution time e_i ;
- device set K_i , consisting of all the I/O devices used by t_i .

Because the tasks are periodic, it is possible to find a global period H such that in every H interval, all of the tasks are repeated. If we get the optimal schedule for these tasks during the first H interval, the same schedule can be applied to other H intervals. Here we define the global period H as the smallest common multiple of the periods of all tasks in T . Associated with each task set T , there is a job set $J = \{j_1, j_2, \dots, j_m\}$ (where $m = \sum_{i=1}^n H/p_i$) consisting of all the executions of each task t_i in T during the first global period H . We assume that each task t_i is submitted once every p_i interval, and that it must be executed before the next p_i interval. The deadline for each job is taken as the sum of its submit time and the period p_i . If we define the delay as the

[†]Dr. Tughrul Arslan is currently working as a research scientist at the Jet Propulsion Laboratory, Avionics Section, Pasadena, CA 91109.

Table 1: Task Parameters Needed for the Schedule.

Task name	Execution time	Period	Device
t_1	ϵ_1	p_1	K_1
t_2	ϵ_2	p_2	K_2
...
t_n	ϵ_n	p_n	K_n

difference between the job start time and its submit time, then to satisfy the deadline, the minimum and maximum delay should follow equation (1) and (2) respectively.

$$\text{Mindelay}_i = 0 \quad (1)$$

$$\text{Maxdelay}_i = \text{deadline}_i - \text{submit}_i - \epsilon_i \quad (2)$$

Supposing that all tasks submitted their first job at time 0, we can get the submit time, deadline, mindelay and maxdelay for each job in $J=\{j_1, j_2, \dots, j_m\}$, as shown in table 2.

3 Genetic Algorithm Implementation

A straight-forward approach to determining an energy-optimal schedule is to exhaustively enumerate all possible schedules for the job set and then to select the one with the minimum energy. However, such an approach, being extremely time and memory-intensive even for small job sets, is infeasible in practice. A pruning-based method is proposed in [12]. It is more efficient than an exhaustive search, however the time and memory requirements still increase rapidly as the number of jobs is increased. In this paper we investigate the implementation of a scheduler that is based around Genetic Algorithm. A Genetic Algorithm (GA) is a search algorithm inspired by the mechanics of natural selection. The algorithm combines survival of the fittest with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of a human search [14]. In this paper we use a basic genetic algorithm which is a modified version of that is posted in [14] to search for the near-optimal device schedule for periodic real-time tasks. The most important two elements of our GA are the chromosome and fitness function. These will be discussed in the following section.

3.1 The Chromosome

For our specific scheduling problem, the start sequence for multiple jobs must be generated simultaneously, so the multi-parameter coding method should be adopted. To construct a multi-parameter coding, we simply concatenate many individual parameters. Each individual parameter is defined as a sub-chromosome, representing the delay between start time and submit time for each job. The chromosome structure is shown in figure 1.

All of the sub-chromosomes have the same length, and the length is decided by the maximum delay of all jobs. Suppose:

$$M = \max\{\text{Maxdelay}_1, \text{Maxdelay}_2, \dots, \text{Maxdelay}_m\}$$



Figure 1: Chromosome Structure

We can find the sub-chromosome length SL using:

$$SL = \lceil \log_2(M + 1) \rceil \quad (3)$$

So, the chromosome length is:

$$L = SL * m \quad (4)$$

To satisfy the deadline requirement, from table 2 we can see that the delay time must be within $[\text{Mindelay}, \text{Maxdelay}]$, but the value for the sub-chromosome can be decoded as any value within $[0, 2^{SL}]$ (where SL is the length of binary string for the sub-chromosome), so we need to map the decoded unsigned integer linearly from $[0, 2^{SL}]$ to the range of delay times within $[\text{Mindelay}, \text{Maxdelay}]$. The mapping equation is:

$$\text{delay}_i = (\text{Maxdelay}_i - \text{Mindelay}_i + 1) * x_i / (2^{SL} - 1) \quad (5)$$

Where x_i is the value of a sub-chromosome, $x_i \in [0, 2^{SL}]$.

3.2 Fitness Function and the Reduction of Device Power Consumption

To achieve the minimal power consumption, the schedule should (i) minimize the time for which the devices are powered up, (ii) maximize the time for which the devices are shutdown, (iii) minimize the number of device transitions. To satisfy these requirement, the schedule is based on the following criterion. First, a device is put into a working state only when it is being used by the executing job which guarantee the minimized power up time and the maximized sleep time for a device. Second, device transitions between the working and the sleeping states are reduced as much as possible. This will reduce transition energy, because the frequent transitions between the different power states of a device is very energy consuming. From this point of view, if two or more jobs which are using the same device can be executed successively, it will reduce the transitions of the device and extend the sleep time for other devices. To find a schedule with many appearances of such a condition is our objective. In order to evaluate a schedule, we define the number of transitions avoided during one global period as its score. see equation (6) and (7).

If the deadline is not satisfied:

$$\text{Score}_i = 0; \quad i = 1, 2, \dots, k \quad (6)$$

Table 2: Jobs to be Scheduled in the First Global Period.

Job	Submit time	Deadline	Min delay	Max delay
jl_1	0	p_1	0	$p_1 - \epsilon_1$
jl_2	p_1	$p_1 * 2$	0	$p_1 - \epsilon_1$
...
jl_{l_1}	$p_1 * (l_1 - 1)$	$p_1 * l_1$	0	$p_1 - \epsilon_1$
$j2_1$	0	p_2	0	$p_2 - \epsilon_2$
$j2_2$	p_2	$p_2 * 2$	0	$p_2 - \epsilon_2$
...
$j2_{l_2}$	$p_2 * (l_2 - 1)$	$p_2 * l_2$	0	$p_2 - \epsilon_2$
...
jn_1	0	p_n	0	$p_n - \epsilon_n$
jn_2	p_n	$p_n * 2$	0	$p_n - \epsilon_n$
...
jn_{l_n}	$p_n * (l_n - 1)$	$p_n * l_n$	0	$p_n - \epsilon_n$

If the deadline is satisfied:

$$Score_i = T(i)_{max} - T(i)_s; \quad i = 1, 2, \dots, k \quad (7)$$

Where $score_i$ represent the score for device i . $T(i)_{max}$ is defined as the maximum number of transitions for device i without reasonable schedule during one global period time H , and $T(i)_s$ represent the transitions needed by the evaluated schedule during the same period. k is the number of devices in the system. If we define the power consumed by transaction for wake up and shut down as $p(i)_w$ and $p(i)_s$, while the time for wake up and shut down defined as $t(i)_w$ and $t(i)_s$. The fitness is defined as the power saved by avoided transitions through reasonable schedule. It is calculated through the following equation:

$$Fitness = \sum_{i=1}^k [p(i)_w * t(i)_w + p(i)_s * t(i)_s] * Score \quad (8)$$

3.3 GA Operation

Our GA operates through the following stages: initialization of the population, generation of new populations, and fitness evaluation. An initial population of random binary strings are produced by the system. Each member of the population is a potential scheme for the schedule. Each scheme is evaluated for its fitness according to the fitness function. The system keeps track of the current highest fitness value so far. The highest fitness value of each new generation is compared with this value and larger, the current highest fitness is replaced by the new value, otherwise the value is kept the same. To produce a new generation, we use Roulette Wheel selection to select the parents, then apply crossover and mutation operators to make out the new generation. The same process is repeated until the termination condition is satisfied. That is the algorithms converges.

The Generations needed to get the high fitness schedule is mainly dependent on the population sizes of GA. While for the fixed population size, the fitness value is dependent on the generations. The results are presented in figure 2 and figure 3 respectively.

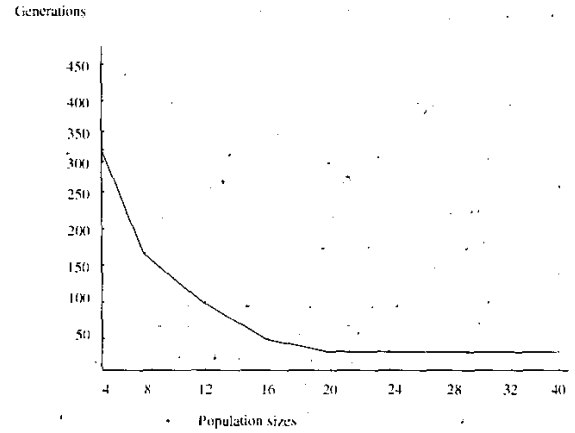


Figure 2: Graph to show population size against the number of generation of the GA

Figure 2 shows the population size against the number of generation of the GA to obtain high fitness. It could be deduced from the figure that as the size of the population increases, the number of generation is reduced to obtain a same performance. However with population size 20 and above, the generations is a constant to reach the same performance. For this reason a population size of 20 was selected to provide a trade-off between memory size and performance. Figure 3 demonstrates a typical run for the GA. A solution is found in this case after 45 generations. The parameters used in our GA are:

- Population Size=20;
- Max Generation=50;
- Crossover Probability=0.6;
- Mutation Probability=0.01.

4 Results and Conclusions

The shortcomings in memory and time requirements become obvious for traditional algorithms (enumerative, ran-

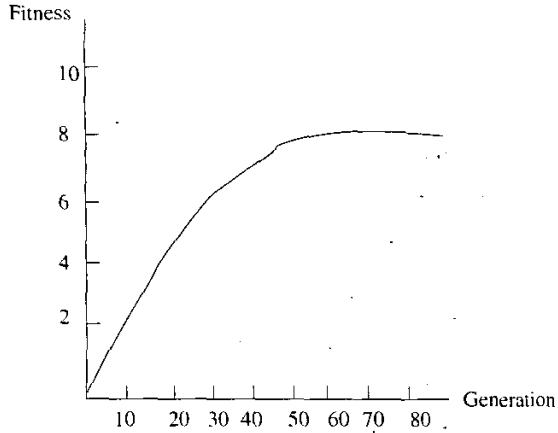


Figure 3: Graph to show generation against the fitness of the GA

dom as well as mathematic methods) which deal with the scheduling problem as the problem size is increased. [12] proposed a pruned algorithm aimed at the real-time device scheduling problem, as mentioned above, and made a comparison with enumerative algorithms (see table 3 and table 4).

Table 3 lists memory requirements for two periodic tasks with different periods. From that, we can see that the pruned algorithm uses much less memory than the exhaustive one, but the memory requirements still increase quickly as the number of scheduled jobs is increased. The unit of memory size in table 3 is a vertex, which is 3-bytes of data including time and energy information for the algorithm. In the first line, the periods of the two tasks are 4 ms and 5ms respectively, the global period is 20ms, and there are 9 jobs in each global period that need to be scheduled. 238 vertices are needed, which is about 714 bytes. At the bottom line, for 17 jobs, 959872 vertices are needed, which is about 2879616 bytes. The memory increases about 3344 times.

Table 4 compares the time requirements between the exhaustive and the pruned algorithms. We can see that in terms of time consumed, the pruned algorithm is much better than the exhaustive one, but the time taken increases rapidly as the number of scheduled jobs increases. The execution time increases 18623 times when the number of jobs changed from 9 to 17.

In order to evaluate our GA-based scheduler, we have used the same example as used in [12]. With our GA-based scheduler, the memory requirements increase as the population size increases, and the time required is decided by the number of generations needed to achieve a suitable fitness.

Because our GA based scheduler only use the information of current and parent generation, the useful information for calculation is the old population, the new population, the old chromosomes, the new chromosomes and the best fit chromosome. Each population is a structure which will take up 24 bytes. Because the maximum delay for all jobs is less than 8 (see table 2). The sub-chromosome length can be calculated out as 3 bits from equation (3), then the chro-

mosome length would be $3\text{bits} \times 17 = 51\text{bits}$, it will take up 7 bytes. So, for a population size of 20, the total memory requirements is shown below. It indicates that the memory requirements of our GA is much less than both exhaustive and pruned algorithms in most cases and it is not increase with the the number of scheduled jobs increased.

Memory requirements for GA-based scheduler
 $= 24 \times 40 (\text{populations}) + 7 \times 40 (\text{chromosomes}) + 7 (\text{best fit chromosome})$
 $= 1247 \text{ bytes}$

With the population size 20, crossover probability 0.6 and mutation probability 0.01, the generations needed to get the near-optimal schedule is shown in table 5. To reduce the random noise, table 5 is an average result obtained by running the process 50 times. From table 5, we can find that the number of generations needed to get the required fitness is less than 50 for all the task sets listed in table 3, in fact it takes only a few minutes to get the near-optimal schedule on our Sun workstation. Though the performance of our Sun workstation is better than which pruned algorithm was running on, the execution time improvement is much better than the performance difference of the 2 platforms. It shows that our GA based scheduler is more competitive. From table 5 we can also find that the number of generations does not increased with the number of scheduled jobs, which means that the executing time scales well with the problem complexity is increased.

For the task sets listed in table 3, Our GA based scheduler get the same schedule quality as the pruned algorithm. Though GA based scheduler can not guarantee to get the optimal schedule in all cases, the near-optimal quality can be accepted by most applications. Compared with the memory and time benefits, our GA based scheduler is much reasonable, especially for large number of scheduled jobs.

The discussion above shows that our GA-based scheduler is not only attractive in less memory and time requirements, but also the well scaled feature for complex problems.

5 Future Work

Generally, an embedded system consists of two components: a software component which is application-specific software running on a dedicated processor, and a hardware component, consisting of application specific circuits. Power dissipation takes place within hardware, and a programmable processor changes its instantaneous internal activity, and duration of operation based on the software being run on it [15]. The overall power dissipation of the system is impacted by both hardware and software. This paper compared the efficiency of an GA-based device scheduler with other schedulers. In future we will do some work on HW/SW co-design which means we will consider not only the scheduler but also the hardware platform it runs on as well.

Table 3: Comparison of Memory Requirements Between Exhaustive and Pruned (this table is taken from [12]).

Task set	Global Period	Job No.	Exhaustive	Pruned
$t_1 = 4, t_2 = 5$	20	9	1512	238
$t_1 = 5, t_2 = 6$	30	11	252931	4110
$t_1 = 5, t_2 = 7$	35	12	29640934	13818
$t_1 = 5, t_2 = 8$	40	13	23033089	43783
$t_1 = 5, t_2 = 9$	45	14	out of MEM	84107
$t_1 = 5, t_2 = 11$	55	16	out of MEM	592091
$t_1 = 5, t_2 = 12$	60	17	out of MEM	959872

Table 4: Comparison of Time Requirements Between Exhaustive and Pruned (this table is taken from [12]).

Task set	Global period	Job No.	Exhaustive	Pruned
$t_1 = 4, t_2 = 5$	20	9	< 1s	< 1s
$t_1 = 5, t_2 = 6$	30	11	2.3s	< 1s
$t_1 = 5, t_2 = 7$	35	12	28.2s	4.6s
$t_1 = 5, t_2 = 8$	40	13	7m15s	35.2s
$t_1 = 5, t_2 = 9$	45	14	out of MEM	2m29.5s
$t_1 = 5, t_2 = 11$	55	16	out of MEM	2h24m15s
$t_1 = 5, t_2 = 12$	60	17	out of MEM	5h10m23.2s

Table 5: GA Experimental Results.

Task set	Global period	Job No.	Max Fitness	AVG Fittest Generation
$t_1 = 4, t_2 = 5$	20	9	2	48
$t_1 = 5, t_2 = 6$	30	11	3	24
$t_1 = 5, t_2 = 7$	35	12	4	24
$t_1 = 5, t_2 = 8$	40	13	4	40
$t_1 = 5, t_2 = 9$	45	14	5	32
$t_1 = 5, t_2 = 11$	55	16	5	32
$t_1 = 5, t_2 = 12$	60	17	6	24

Bibliography

- [1] Mike Tien-Chien Lee, Vivek Tiwari, Sharad Malik, and Masahiro: Power analysis and minimization techniques for embedded DSP software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol 5, No. 1, March 1997.
- [2] Sandy Irani, Sandeep Shukla, and Rajesh Gupta: Competitive analysis of dynamic power management strategies for systems with multiple power saving states. *Design Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 117-123, 2002.
- [3] Jinwoo Suh, Dong-In Kang, and Craga, S. P: Dynamic power management of Multiprocessor systems. *Parallel and Distributed Processing Symposium, Proceedings International, IPDPS* pp 97-104, 2002.
- [4] J. Lorch and A. Smith: Software strategies for portable computer energy management. *IEEE Personal Communication Mag*, vol 3, No. 5 pp 60-73 June 1998.
- [5] Vishnu Swaminathan, Krishnendu Chakrabarty and S. S. Iyengar: Dynamic I/O power management for hard real-time systems. *Proc. Intl. Symp. Hardware/Software Co-Design (CODES)*, pp. 237-243, 2001.
- [6] G. Quan and X. Hu: Energy efficient fixed-priority scheduling for real time systems on variable voltage processors. *Proc. DAC*, pp. 828-833, 2001.
- [7] A. Sinha and Chandrakasan: Energy efficient real-time scheduling. *Proc. Intl. Conf. CAD*, pp. 458-463, 2001.
- [8] Y. Shin, K. Choi, and T. Sakurai: Power optimization of real-time embedded systems on variable speed processors. *Proc. Intl. Conf. CAD*, pp. 365-368, 2000.
- [9] Y-H. Lu, L. Benini and G. De Micheli: Operating system directed power reduction. *Proc. Intl Conf. Low-Power Electronics and Design*, pp. 37-42, 2000.
- [10] L. Benini, A. Bogliolo, G. A. Paleologo and G. De Micheli: Policy optimization for Dynamic power management. *IEEE Trans. CAD*, vol 16, No. 6, pp. 813-833, June 1999.
- [11] E-Y. Chung, L. Benini and G. De Micheli: Dynamic power management using adaptive learning tree. *Proc. Intl. Conf. CAD*, pp. 274-279, 1999.
- [12] Vishnu Swaminathan, Krishnendu Chakrabarty: Pruning-based energy-optimal device scheduling for hard real-time systems. *Proc. Intl. Symp. Hardware/Software Co-Design (CODES)*, pp. 175-180, 2002.
- [13] M. J. O'Dare and T. Arslan: System design for test using a genetically based hierarchical ATPG system. *Digest of IEE Colloquium on System Design for Testability*, pp 9/1-9/5, 1995.
- [14] David E. Goldberg: *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley Publishing Company, INC. 1989
- [15] Kaushik Roy, Rabindra Roy, and Tan-Li Chou: Design of low power digital system. *Design Low Power Digital Systems, Emerging Technologies* pp. 137-204, 1996.