Optimization for Real-Time Systems with Non-convex Power versus Speed Models

Ani Nahapetian, Foad Dabiri, Miodrag Potkonjak, Majid Sarrafzadeh

Computer Science Department,
University of Calfornia Los Angeles (UCLA)
{ani, dabiri, miodrag, majid}@cs.ucla.edu

Abstract. Until now, the great majority of research in low-power systems has assumed a convex power model. However, recently, due to the confluence of emerging technological and architectural trends, standard convex models have been invalidated for the proper specification of power models with different execution speeds. For example, the use of a shutdown energy minimization strategy to eliminate leakage power in multiprocessor systems results in a non-convex trade-off between power and speed. Non-convexity renders the majority of previous power management schemes, algorithms, and even basic theorems invalid. For instance, the main premise that one has to run continuously using a single speed in order to minimize energy consumption for constant computation requirements is not valid anymore.

We study techniques for energy minimization where the power versus speed curve has a non-convex shape. We first identify and quantify sources of non-convexity. Minimizing energy when the power-speed model is non-convex is an NP-complete problem, even in the canonical and simple case where a task is to execute a specified amount of computation without dependencies, in a given amount of time. We address this problem using a non-linear function minimization based approach and demonstrate that on average the new solution saves at least 40% more energy on industrial processors than techniques that follow the convexity paradigm. Then we address common real-time task scenarios where the power-speed model is non-convex. Specifically, we introduce a heuristic for scheduling tasks onto a multiprocessor system with a non-trivial start-up cost and compare its performance to our mixed integer linear programming (MIP) formulation. We experimentally compare our neighbors heuristic with the well-known average rate algorithm, and find that it results in a 106% improvement while being only 14% worse than the optimal MIP solution.

1 Introduction

Traditionally, low power research has focused on a power model where the relationship between power consumption and processor speed is convex. Convexity has a number of profound ramifications when energy is minimized using variable voltage strategies. For example, running the processor at the lowest speed possible continuously, while still meeting the task deadline, has been the most advantageous strategy. Also, it is well known, that convex objective functions are much more amenable to

both heuristic and provably optimal minimization [4]. All dynamic voltage scaling research has essentially been governed by this fact.

There exist rapidly emerging application and technology scenarios, however, where the relationship between power and processor speed is not convex, including the following situations: (i) multiprocessor systems with powering-up power cost; (ii) scaled down CMOS devices where the increased impact of leakage power dominates the power consumption. (Leakage power does not have a convex relationship with processor speed [7]); (iii) systems with simultaneously adaptive V_{dd} and V_t. (Energy gains can be made by simultaneously varying V_{dd}, through dynamic voltage scaling, and V_t, through adaptive body biasing, where V_{dd} is the supply voltage and V_t, is the threshold voltage [17]. Lowering the threshold voltage increases the processor speed, but at the expense of increasing the leakage power creating a non-convex relationship between power and speed.) (iv) Finally, there is an important emerging class of systems that are subject to non-convex power minimization: subthreshold ultra low power circuits. Recent work at MIT, University of Michigan, and Purdue University have characterized the power versus speed of execution curve for subthreshold ultra low power as non-convex using circuits, CAD, and architectural techniques [5][6][14][18][20][23][24][28]. There is a widely held opinion that subthreshold logic will dominate several rapidly growing segments of ICs and computer and communication architectures.

The optimization process under these new conditions will have numerous and profound consequences and will significantly differ from the current variable voltage approaches. As we mentioned, given a convex power model a single speed properly chosen minimizes the energy consumption. In the non-convex case, on the other hand, the selection of two or more different speeds minimizes energy consumption. Even piecewise convex curves cannot be handled using a single speed, instead, they are best handled, as other non-convex curves, using two speeds. Therefore, with pending system and technology solutions, non-convex power models will dominate the spectrum of power-constrained systems.

TABLE 1. EXAMPLE DATA POINTS

Convex Power Model		**Non-convex Power Model**	
Speed (million cycles/sec)	**Power (W)**	**Speed (million cycles/sec)**	**Power (W)**
0	0	0	0
1	1	1	0.5
2	4	2	2
3	8	3	2.5

Let us examine the difference between a convex and a non-convex power-speed curve with the following example. Assume we are given a task that requires 20 million cycles of computation in 10 seconds. With a convex power to processor speed curve, as given in Table 1, the optimal speed to run the tasks is 2 million cycles/second for 10 seconds at 4 W. On the other hand, with a non-convex curve, also

given in Table 1, the optimal schedule would be to run the task at 3 million cycles/second for 5 seconds and at 1 million cycles/second for 5 seconds. The total energy cost would be 15 J (= 2.5 J + 12.5 J), which is less than the 20 J consumed if we had run at 2 million cycles/sec for the entire interval. This example highlights the new paradigm of non-convex power minimization.

(1) We can no longer run the tasks at their slowest possible speed.

(2) We cannot assume that a single speed will be used by the optimal solution.

Our goal is to develop techniques that under the general non-convex power model address scheduling onto a multiprocessor system with a startup cost for each processor. First, we solve the problem of scheduling single tasks onto a multiprocessor system. Then we solve the more complex problem of scheduling multiple tasks, with arrival times, deadlines, and cycles of computation, onto a multiprocessor system with startups costs, using the solution of the first problem as the enabling procedure. The major contributions of this paper are the following. We introduce and discuss the new paradigm where there is a non-convex power relation between power and processor speed. We solve the fundamental problem of scheduling a single task on to a multiprocessor system, by formulating the problem as non-linear function minimization. We introduce the neighbors heuristic for scheduling tasks onto a multiprocessor system with a significant startup cost. We also formulate a mixed integer programming (MIP) formulation to solve the problem. Finally, we demonstrate the significant improvement possible by experimentally comparing the neighbors heuristic, the MIP approach, and the average rate algorithm.

2 Power Model and Related Work

We consider the case where a startup cost is incurred for transitioning a processor from the sleep state to the on state. We use the startup cost calculated by Jejurikar et al [13] that estimates the cost of changing the state of the processor to be 483μJ, based on several assumptions including ones about the cache state. This cost dominates, if it is incurred. Aside from the startup cost, we incur a cost for keeping a processor on. We use the following formula. $E_{on} = P_{on} / f$ where P_{on} is taken to be 0.1Watts, similar to [13], and f is the frequency, which we take to be 300MHz, the minimum frequency of the Transmeta Crusoe processor. Our power model is based on real processors, specifically, the Transmeta Crusoe processor Model TM5500 [22] and AMD-K6-IIIE+500 ANZ processor [1]. Although we use actual processor values for our experimentation and for our problem abstraction, we do make a few simplifying assumptions. We assume that the cost of transitioning between different voltage values was zero. We also assume that the transition time is negligible. These assumptions are common in the recent literature [11][13][15][26].

Energy minimization techniques can be classified in two broad groups. The first, called dynamic power management (DPM), aims to shutdown processors when they are idle. The second, dynamic voltage scaling (DVS), dynamically varies the voltage

supplied to the processor, to provide just in time execution of tasks. Benini et al provide a survey in [2][3]. Irani et al combine the two methods, DPM and DVS, for systems with DVS and multiple power modes [11].

A large portion of the research in power scheduling algorithms has focused on uniprocessors. The scheduling and assignment of tasks onto multiprocessors generally has been solved utilizing heuristics that have a two-phase approach. They assign jobs to the resources, then they allocate the voltages for the processors, assuming the job assignment determined in the first phase [29][30]. Yu and Prasanna [27] examine the two problems of assignment of jobs to resources and the determining of the voltage levels in a joint manner. They formulate the problem as an integer linear program (ILP), and they utilize a linear relaxation heuristic (LP-relaxation) to solve the problem.

The majority of the related work in DVS follows a convex power model [11][15][21][26][27]. In general, they assume a quadratic relationship between power and processor speed. Jejurikar et al [13] consider the effect of leakage power, but they do not address the problem of scheduling in the non-convex region of the power curve.

3 Fundamental Problem

Let us consider a constrained yet fundamental case of the scheduling problem to gain intuition and to create a powerful procedure for the final steps of the energy optimization in more complex scenarios. The problem is the following. Given a single required average speed at which to run the processor for a period of time, determine the speeds at which to run each segment of the time interval in such a way that the total energy is minimized given a non-convex power-speed relationship. The problem is NP-complete. We prove the claim using reduction from the well-known knapsack problem [8]. The knapsack problem is reduced in polynomial time to the fundamental problem by mapping the weight of the objects to the energy of a chosen speed for a given period of time and by mapping the value of the objects to the speed chosen. The resulting problem is a discretized instance of the fundamental problem, given below.

Instance: Finite set P, for each $p \in P$ an energy $e(p) \in Z^+$ and a speed $s(p) \in Z^+$, and positive integers C and E.

Question: Is there a subset $P' \subseteq P$ such that $\sum_{p \in P} e(p) \cdot s(p) \leq E$ and such that $\sum_{p \in P} e(p) \cdot s(p) \geq C$.

To solve the fundamental problem for multiple processors, we start with multiple energy versus speed curves that correspond to using a different number of processors, and we juxtapose the curves to obtain a new energy versus speed curve. Then given an average speed and a time interval, nonlinear function minimization is used to determine the most energy efficient selection of the number of processors and their

execution speeds. To create the new function, we examine k possible energy versus processor speed mappings, where k is the number of processors. The k curves are then combined, and the minimal assignment and schedule is found.

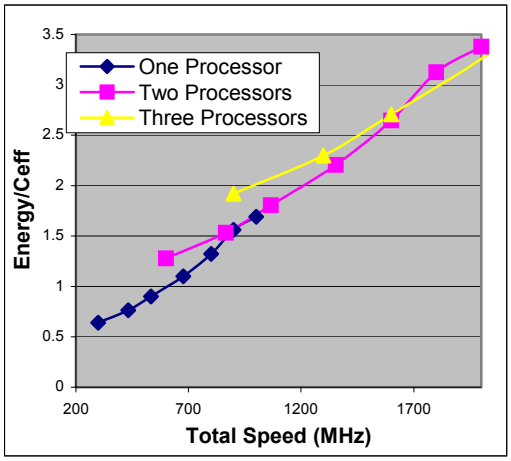

Figure 1. Energy vs. Speed Curves for 3 Types of Multiprocessor Systems

Figure 1 graphs three different curves, where each curve represents the energy consumption (divided by the effective capacitance) for running at the given speed. Each of the curves assumes a certain number of processors available for use. The first curve graphs the energy versus speed curves for the simple case, where there is a single processor, using the data values from the slides associated with [13]. The second curve graphs the case where there are two processors, each running at the same speed. The speed is two times that of the uniprocessor case, because there are two processors. The energy is equal two times the energy required if a single processor were running at half the speed, basically the speed of each of the two processors. This analysis is carried out for k processors or in the case of Figure 1 for three processors. To obtain the solution for the fundamental problem, a nonlinear equation minimization function can be used to determine the number of processors and their speeds that consume the minimum amount of energy. The NLP has been omitted due to space limitations.

As illustration, we ran a nonlinear minimizer, specifically the Powell, or Direction Set, Method in Multidimensions [19] on a four processor system. The results are shown in Figure 2. Figure 2 shows that NLP solution is able to consistently improve by on average 45.8% on the solution obtained with traditional convex scheduling techniques. Note that we assumed the processors exhibit no cost in starting up and/or changing their voltage values. The results are obtained using a 1000 random restarts for the Powell method. The results may somewhat improve with a larger number of restarts. Once we have the solution to the fundamental problem, we can use it to simplify our overall problem. As long as we can determine the total number of cycles that are needed to be executed in an interval, then the solution given in this section can be

applied to the determine the most energy efficient execution of the cycles.

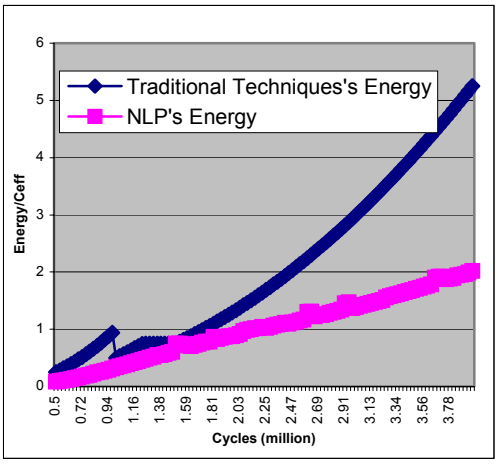

Figure 2. Energy Consumption using NLP Solution

4 Scheduling for Non-convex Power Model

In this section, we highlight a provably optimal ILP-based approach for power minimization in systems subject to non-convex power-speed relationship, as well as a new heuristic approach for the same problem. Before addressing energy minimization with non-convex power-speed models, we formally define the addressed problem of minimizing energy for a set of tasks with known computational requirements and arrival and deadline times.

Let us consider the problem of scheduling tasks onto multiprocessors with a startup cost. The problem requires determining the number of processors to use and speed at which to execute the cycles at any given time, such that the energy consumption is minimized. More formally, we are given as input a set of *n* tasks. Each task is characterized by the following: a_i – arrival time before which task *i* cannot be executed, d_i – deadline by which time task *i* must be completed, c_i – cycles of computation associated with task *i*.

We assume a hard real-time scenario where tasks are periodic, and where the deadline for each period is equal to its worst-case execution time (WCET), as is common among the literature [11][13][15][16]. We are also given a set of homogenous processors, whose applied voltages can be varied dynamically. We assume homogenous processors for the sake of clarity, but the algorithms and software tools can easily be extended to handle heterogeneous processors as well. We consider processors that can be transitioned into sleep mode to save energy, if they are idle, and we assume that the processors have a start-up cost associated with them.

We formulate the problem as a mixed integer programming (MIP) problem and

solve it optimally using an MIP solver such as CPLEX [10]. Although MIP in the worst case can take an exponential amount of time, this MIP formulation is often fast for instances of practical interest. For D total time units and N tasks, there are D·N continuous variables and 2N integer variables. There are six types of constraints, with a total of 3N + 3D constraints. There is on average 3D/N + (N+5)/D variables per constraint. The MIP formulation has been omitted due to space limitations

For the problem formulated in section 4, we have also developed the neighbors heuristic. Unlike the MIP, the heuristic does not always produce the optimal solution; however, it is useful for large problem instances where the MIP may be prohibitive. Also, it allows the addition of nonlinear constraints to the problem, such as allowing the processors to go to sleep at any time instance.

Intuitively, the heuristic evens out the load on neighboring intervals, by reducing the instances of change in the number of processors used. It also attempts to run the active processors with the voltage values that were determined to be advantageous by the solution of the fundamental problem. The psuedocode for the heuristic is given below.

The heuristic initially assigns tasks to execute based on their average rate, given by the equation $Average\ rate_i = \frac{c_i}{d_i - a_i}$. The solution is iteratively improved by evening out the load on neighboring intervals, by reallocating the cycles of overlapping tasks. We iteratively choose the two intervals that have the largest density difference that share a common task. If there are multiple tasks to choose from, we move the task that is less likely to be redistributed in a following iteration. Finally, we optimize each interval using the solution for the fundamental problem.

The intuition behind the neighbors heuristic is based on the fact that the less change there is in the required average speed, the better the speed assignments can be. Although, the neighbors heuristic will not be useful for all speed energy tradeoff, it is very useful for the common case in multiprocessor systems where, roughly, the larger the computation requirement the less efficient the speed energy tradeoff is, especially if we are dealing with a piecewise convex model.

Neighbors Heuristic(tasks)
for all tasks Assign the tasks to their execution intervals at their average rate **for** a certain number of iterations **for** each interval *i* and its neighboring interval *j* Choose the task belonging to both intervals that has the earliest deadline Move the cycles of execution of the chosen task from the more dense interval to the less dense interval until the intervals are even or until all of the common task has been moved **for** each interval Optimize the voltage settings using the fundamental problem's solution

5 Experimental Results

We evaluated the effectiveness of the neighbors heuristic experimentally against randomly generated task sets, which are standard among the related work [13][15][21][27]. The tasks' values were based on the work of Kwon et al [15], by choosing arrival times and deadlines to be in the range of 1 to 20 seconds and cycles of execution from the range of 1 to 400 million cycles. For comparison purposes, we also implemented the following heuristics. (1) The average rate algorithm [26] assigns tasks to their execution intervals to be executed at their average rate. The number of processors is based on the number of cycles to be executed at that time instance. (2) The contention-based algorithm is an enhancement to the average rate algorithm. It moves tasks from high contention intervals to low contention intervals, where an interval with low contention is characterized as having a small number of live tasks during the interval. The contention-based heuristic chooses the interval with the least contention to which to move task cycles. Among the tasks that are live in that interval, the task with the largest density is moved to the interval. (3) The flattening heuristic aims to flatten the scheduled execution of cycles by eliminating peaks and valleys in the schedule to decrease the number of startups. As in the contention-based and neighbors heuristic, the tasks are initially scheduled according to the average rate algorithm. Then, the solution is iteratively improved, by choosing two intervals that have the largest density difference that share a common task. The common task is redistributed so that the intervals are even in terms of density. Of course, we are limited by the number of cycles belonging to the chosen task, and thus cannot move more cycles than the task requires. As the process of flattening is done iteratively, the schedule will eventually flatten out, if intervals have multiple tasks in common. If there are multiple tasks that lie in both of the intervals, then the task with the greatest density is chosen for redistribution.

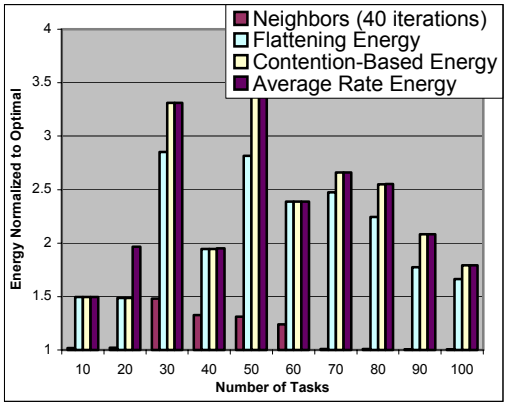

Figure 3. Energy vs. Number of Tasks for Various Approaches

Figure 3 present the results obtained from our experimentation. The bar chart dis-

plays the energy consumption normalized to the optimal value for 10 different task sets of varying sizes for each of the four heuristics: the average rate heuristic, the contention-based heuristic, the flattening heuristic, and the neighbors heuristic. The neighbors heuristic dramatically out-performs all of the other heuristics. The heuristic is on average 106% better than the well-known average rate heuristic and 84% better than the flattening heuristic, yet is it only 14% worse than the optimal solution. For the task sets with 70, 80, 90, and 100 tasks the results are equal to the optimal values.

6 Conclusion

In this paper, we addressed energy minimization for non-convex power-speed models, which are poised to dominate future applications and technologies. We first addressed the fundamental problem of scheduling onto a multiprocessor system a computation requirement for a given interval. We, then, addressed the more complicated problem of scheduling tasks for systems with a large startup cost. First, we formulated the problem as a mixed integer-programming problem and showed how it could be solved very efficiently. Secondly, we introduced the neighbors heuristic, which evens out the density of neighboring intervals initially produced by the average rate schedule. We carried out extensive experimentation to quantify the quality of our approaches.

References

[1] 1. AMD PowerNow! Technology Platform Design Guide for Embedded Processors. AMD Document number 24267a, December 2000.

[2] L. Benini and G. DeMicheli. System-Level Power: Optimization and Tools. International Symposium Low Power Embedded Design (ISLPED), 1999.

[3] L. Benini, A. Bogliolo and G. DeMicheli. A Survey of Design Techniques for System-Level Dynamic Power Management. IEEE Transactions on VLSI Systems, vol. 8, No. 3., June 2000.

[4] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, Cambridge, 2004.

[5] B.H. Calhoun, A.P. Chandrakasan. Characterizing and Modeling Minimum Energy Operation for Subthreshold Circuits. International Symposium Low Power Embedded Design (ISLPED), 2004

[6] B.H. Calhoun, A. Wang, A.P. Chandrakasan. Device sizing for minimum energy operation in subthreshold circuits. Custom Integrated Circuits Conference (CICC), 2004

[7] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low- Power CMOS digital design. IEEE Journal of Solid-State Circuits, vol. 27, No. 4, pp.473-484, 1992.

[8] M.R. Garey and D.S. Johnson. Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, 1979.

[9] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M. B. Srivastava. Power optimization of variable-voltage core-based systems. Design Automation Conference (DAC), 1998

[10] ILOG CPLEX http://www.ilog.com/products/cplex/

[11] S. Irani and R. Gupta. Algorithms for Power Savings. Symposium on Discrete Algorithms, 2003
[12] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. International Symposium Low Power Embedded Design (ISLPED), 1998
[13] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. Design Automation Conference (DAC), 2004.
[14] J. Kao, S. Narendra, A. Chandrakasan. Subthreshold leakage modeling and reduction techniques. International Conference on Computer Aided Design (ICCAD), 2002.
[15] W. C. Kwon and T. Kim. Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. Design Automation Conference (DAC), 2003.
[16] C. L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. Journal of ACM, v.20, No.1, pp.46-61, 1973.
[17] S. M. Mastin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. International Conference on Computer Aided Design (ICCAD), 2002.
[18] L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth. Energy Optimization of Subthreshold-Voltage Sensor Network Processors. International Symposium on Computer Architecture (ISCA), 2005.
[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical Recipies in C: The Art of Scientific Computing. Cambridge University Press, New York, NY, 1994.
[20] H. Soeleman, K. Roy, B.C. Paul. Robust subthreshold logic for ultra-low power operation. IEEE Transactions on VLSI Systems, Vol. 9, No. 1, 2001.
[21] Y. Shin and K. Choi. Power Consious Fixed Priority Scheduling for Hard Real-Time Systems. Design Automation Conference (DAC), 1999.
[22] Transmeta Crusoe Data Sheet. https://www.transmeta.com
[23] A. Wang, A.P. Chandrakasan. A 180mV FFT processor using subthreshold circuit techniques.Solid-State Circuits Conference, 2004. Digest of Technical
[24] A. Wang, A.P. Chandrakasan, S.V. Kosonocky. Optimal supply and threshold scaling for subthreshold CMOS circuits. International Symosium on VLSI (ISVLSI), 2002.
[25] W. Wolf and M. Potkonjak. A Methodology and Algorithms for the Design of Hard Real-Time Multi-Tasking ASICs. ACM Transaction on Design Automation of Electronic Systems (TOADES), v.4, No.4, pp. 430-459, 1999
[26] F. Yao, A. Demers and S. Shenker. Scheduling for Reduced CPU Energy. IEEE Annual Foundations of Computer Science (FOCS), 1995.
[27] Y. Yu and V. Prasanna. Resource Allocation for Independent Real-Time Tasks in Heterogeneous Systems for Energy Minimization. International Conference on Computer Aided Design (ICCAD), 2001.
[28] B. Zhai, D. Blaauw, D. Sylvester, K. Flautner. Theoretical and practical limits of dynamic voltage scaling. Design Automation Conference (DAC), 2004.Y. Zhang, X. Hu, and D. Z. Chen. Task Scheduling and Voltage Selection for Energy Minimization. Design Automation Conference, 2002.
[29] D. Zhu, R. Melhem, and B. Childeres. Scheduling with Dynamic Voltage/Speed Adjustment using Slack Reclamation in Multi-processor Real-Time Systems. IEEE Real-Time Systems Symposium, 2001.
[30] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M. B. Srivastava. Power optimization of variable-voltage core-based systems. Design Automation Conference (DAC), 1998.