Algorithmic Problems in Power Management

Sandy Irani
School of Information and Computer Science
University of California, Irvine

Kirk R. Pruhs *
Computer Science Department
University of Pittsburgh

1 Introduction

We survey recent research that has appeared in the theoretical computer science literature on algorithmic problems related to power management. We will try to highlight some open problems that we feel are interesting. This survey places more concentration on lines of research of the authors: managing power using the techniques of speed scaling and power-down which are also currently the dominant techniques in practice.

1.1 Motivation

The power consumption rate of computing devices has been increasing exponentially. Since the early 1970s, the power densities in microprocessors have doubled every three years [34]. This increased power usage poses two types of difficulties:

- **Energy Consumption**: As energy is power integrated over time, supplying the required energy may become prohibitively expensive, or even technologically infeasible. This is a particular difficulty in devices that rely heavily on batteries for energy, and will will become even more critical as battery capacities are increasing at a much slower rate than power consumption. Anyone using a laptop on a long flight is familiar with this problem.

- **Temperature**: The energy used in computing devices is in large part converted into heat. For high-performance processors, cooling solutions are rising at $1 to $3 per watt of heat dissipated, meaning that cooling costs are rising exponentially and threaten the computer industry's ability to deploy new systems [34]. In May 2004 Intel publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. Intel scrapped the development of its Tejas and Jayhawk chips in order to rush to the marketplace a more efficient chip technology. Designers said that the escalating heat problems were so severe that they threatened to cause its chips to fracture [27]. For a striking example of the grievous effect of removing the fan from a modern processor, see `http://www.cs.pitt.edu/~kirk/cool.avi`.(You will need a DivX codec installed.)

These two factors have resulted in power becoming a first-class design constraint for modern computing devices [28].

There is an extensive literature on power management in computing devices. Overviews can be found in [11, 28, 36]. All of these techniques that have been investigated are similar in that they reduce or eliminate power to some or all components of the device. Sensor networks have emerged as an important new

*Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, and CCF-0448196.

paradigm in which power-aware computation is absolutely critical. The explosive interest in sensor networks is the result of the development of low-cost, low-power mutifunctional sensor devices, such as the Smart Dust Mote [1, 22], that are small in size and communicate untethered at short distance.

There is an inherent conflict between power reduction and performance; in general, the more power that is available, the better the performance that can be achieved. As a result, it is generally proposed that power reduction techniques be preferentially applied during times when performance is less critical. However, this requires a policy to determine how essential performance is at any given time and how to apply a particular power reduction technique. Current tools and mechanisms for power management are inadequate and require more research [14]. Furthermore, there is a growing consensus that these policies must incorporate information provided by applications and high levels of the operating system in order to achieve necessary advances [14].

We advocate formalizing power management problems as optimization problems, and then developing algorithms that are optimal by these criteria. The goal is to develop effective algorithms for specific problems within the domain of power management as well as to build a toolkit of widely applicable algorithmic methods for problems that arise in energy-bounded and temperature-bounded computation.

2 Speed Scaling

2.1 Formulation as a Scheduling Problem

Speed scaling involves dynamically changing the voltage and/or frequency/speed of the processor. A processor consumes less power when it is run at a lower speed. Both in academic research and practice, dynamic voltage/frequency/speed scaling is the dominant technique to reduce switching loss, which is currently the dominant form of energy consumption in microprocessors [11, 28, 36]. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. Informally, speed scaling problems involve determining the speed of the processor at each point in time.

Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shankar [37]. Yao *et al.* [37] propose formulating speed scaling problems as scheduling problems. The setting is a collection of tasks, where each task i has a release time r_i when it arrives into the system, and an amount of work w_i that must be performed to complete the task. A schedule specifies which task to run at each time, and at what speed that task should be run.

In particular, Yao *et al.* [37] consider the case that there is also a deadline d_i associated with each task that specifies the time by which the task should be completed. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks or insure a certain quality of service to each task [12]. Yao *et al.* [37] assume that the system's performance measure is deadline feasibility; that is, each task must finish by its deadline.

They study the problem of minimizing the total energy used subject to the deadline feasibility constraints. Bansal, Kimbrel and Pruhs [7, 8] study the problem of minimizing the maximum temperature attained subject to the deadline feasibility constraints.

2.2 Energy and Temperature

Before proceeding further, we need to explain how speed, power, energy, and temperature are modeled, and how they are related. Yao *et al.* [37] assume a continuous function $P(s)$ such that if the device runs at speed s, then it consumes power at a rate of $P(s)$. For example, the well known cube-root rule for CMOS based devices states that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$, the power is proportional to the speed cubed [11]. Yao *et al.* [37] only assume that

$P(s)$ is strictly convex. This assumption implies that that the slower a task is run, the less energy is used to complete that task. Some simplicity of analysis, and little loss of applicability, comes from assuming that $P(s) = s^\alpha$ for some constant $\alpha > 1$.

The total energy used by the system is then $\int_0^\infty P(s(t))dt$ where $s(t)$ is the speed of the device at time t.

We now turn our attention to temperature. Cooling, and hence temperature, is a complex phenomenon that can not be modeled completely accurately by any simple model [33]. In [7], Bansal Kimbrel and Pruhs propose a model in which the environmental temperature is assumed to be constant. While this assumption certainly is not strictly true, the hope is that it is sufficiently close to being true that insight gained with this model will be useful in real settings. They also assume that the rate of cooling of the device adheres to Fourier's Law. Fourier's law states that the rate of cooling is proportional to the difference in temperature between the object and the environment. Without loss of generality one can scale temperature so that the environmental temperature is zero. A first order approximation for the rate of change T' of the temperature T is then $T' = aP - bT$, where P is the supplied power, and a, b are constants.

Some modern processors are able to sense their own temperature, and thus can be slowed down or shut down so that the processor temperature will stay below its thermal threshold [34]. If one views `http://www.cs.pitt.edu/~kirk/cool.avi`, this is the reason why the Pentium only slows down, and doesn't fry like the AMD processor.

Bansal and Pruhs [8] show that the maximum temperature is within a factor of 4 of a times the maximum energy used over any interval of length $\frac{1}{b}$. This observation also shows that there is a relationship between total energy and maximum temperature optimization and simplifies the task of reasoning about temperature. If the cooling parameter b is 0 then the temperature minimization problem becomes equivalent (within a constant factor) to the energy minimization problem. This also explains why some algorithms in the literature for energy management are poor for temperature management, that is, these algorithms critically use the fact that the parameter $b = 0$. If the cooling parameter b is ∞ then the temperature minimization problem becomes equivalent to the problem of minimizing the maximum power, or equivalently minimizing the maximum speed. We say that an algorithm is *cooling oblivious* if it is simultaneously $O(1)$-approximate for minimizing the maximum temperature for all values of a and b in the temperature equation $T' = aP - bT$, Thus a cooling oblivious algorithm is also $O(1)$-approximate for total energy and maximum speed/power. The energy minimization problem, when the speed to power parameter α is ∞ is also equivalent to minimizing the maximum power.

2.3 Energy Minimization with Deadline Feasibility

Yao, Demers and Shankar [37] study the problem of minimizing the total energy used to complete all tasks subject to the deadline feasibility constraints. They give an offline greedy algorithm (YDS) that optimally solves this problem. The algorithm YDS proceeds in a series of iterations. During each iteration, tasks in the maximum intensity interval are scheduled Earliest Deadline First at a speed equal to the intensity of this interval; the intensity of a time interval is defined to be the sum of the work requirements of all tasks whose release time and deadline are both contained within the interval divided by the length of an interval. The newly scheduled time interval is then blacked out, and all the remaining tasks must be executed during the remaining time that is not blacked out. It was shown by Bansal and Pruhs [8] that the energy optimality of the YDS schedule follows as a direct consequence of the well known KKT optimality conditions for convex programs.

Theorem 1. *The YDS algorithm is optimal for energy minimization.*