

# Energy-Efficient Duplex and TMR Real-Time Systems \*

Appeared in the IEEE Real-Time Systems Symposium, Dec 2002

Elmootazbellah (Mootaz) Elnozahy  
System Software Department  
IBM Austin Research Laboratory  
Austin, TX 78758  
mootaz@us.ibm.com

Rami Melhem, Daniel Mossé  
Computer Science Department  
University of Pittsburgh  
Pittsburgh, PA 15260  
{melhem, mosse}@cs.pitt.edu

## Abstract

*Duplex and Triple Modular Redundancy (TMR) systems are used when a high-level of reliability is desired. Real-Time Systems for autonomous critical missions need such degrees of reliability, but energy consumption becomes a dominant concern when these systems are built out of high-performance processors that consume a large budget of electrical power for operation and cooling. Examples where energy consumption and real time are of paramount importance include reliable computers onboard mobile vehicles, such as the Mars Rover, satellites, and other autonomous vehicles.*

*At first inspection, a duplex system uses about two thirds of the components that a TMR system does, leading one to conclude that duplex systems are more energy-efficient. This paper shows that this is not always the case. We present an analysis of the energy efficiency of duplex and TMR systems when used to tolerate transient failures. With no power management deployed, the analysis supports the intuitive impression about the relative superiority of duplex systems in energy consumption. The analysis shows, however, that the gap in energy consumption between the two types of systems diminishes with proper power management. We introduce the concept of an optimistic TMR system that offers the same reliability and performance as the traditional one, but at a fraction of the energy consumption budget. Optimistic TMR systems are competitive with respect to energy consumption when compared with a power-aware duplex system, can even exceed it in some situations, and have the added bonus of providing tolerance to permanent faults.*

## 1. Introduction

There are many commercial, technical and environmental motivations to reduce energy consumption in computing systems. Modern systems continue to deploy high-performance processors with increasing energy requirements for operation and cooling. This creates and motivates the need for more energy-conscious

designs and algorithms. In this paper, we study the problem of reducing power consumption in fault-tolerant architectures based on component replication. These systems are typically used when high degrees of reliability and availability are required [25]. We focus on two types of systems used for this purpose, namely duplex and triple modular redundancy (TMR) systems. These systems can tolerate one transient fault of any type that can be detected by either checking for state divergence (duplex systems) or vote discrepancy (TMR systems).

Power management through dynamic voltage scaling (DVS) as well as fault tolerance through replication have been well studied in the context of real-time systems (RTSs). However, researchers have not addressed the problem of combining energy consumption and fault tolerance in RTSs. There are many situations where it is important, however, to consider power management as a central component in a fault-tolerant system. For example, it is important to conserve the energy used by reliable computers onboard autonomous vehicles. The conservation allows these systems to reduce the battery weight and the overhead of generating the energy onboard. It also allows these systems to operate at lower temperatures, reducing the cooling needs and reducing the amount of white noise (thus increasing reliability).

We focus on duplex and TMR systems. At first inspection, one may correctly conclude that a duplex system consumes only two thirds of the energy of a TMR system that uses the same machine types. We present an analysis that confirms this intuition when no power management is used. However, we present a somewhat surprising result, in showing that with proper power management, the difference in energy consumption between the two types diminishes. Therefore, with this differentiator all but eliminated, one can focus on the other merits of the selection between the two types, such as extended reliability, performance, and so forth.

The paper first develops a simple theory for power management in duplex/TMR real-time systems. Using this theory, we use a mathematical analysis to understand and compare the energy behaviors of the architectures under study. When appropriate, we use closed form solutions to drive the analysis. When such closed forms are not available, we iteratively solve the equations to obtain the desired solutions. We drive the models that we developed using parameters obtained from actual measurements in our laboratories or from manufacturer's data sheets. The contributions of

---

\*This research has been supported in part by The Defense Advance Research Projects Agency under contract F33615-00-C-1736. We acknowledge the Trademarks and copyrighted material mentioned here as the property of their owners.

the paper can be summarized as follows:

- It develops a simple theory for power management in the context of duplex and TMR RTSs.
- It derives the best power management policies for duplex systems based on DVS and on controlling the insertion of synchronization points in the execution stream, to ensure deadlines are met.
- It derives the best power management policies for TMR systems based on hibernation and DVS.
- It suggests a new architecture that we call *optimistic TMR* that virtually eliminates the difference in energy consumption between duplex and TMR systems.

As in a study that explores uncharted research areas, we do not make any claims that the study is complete or practical. Ours is a first stab at understanding and formulating the problem, and setting up models that could be used to reason about, and develop solutions for, the problem of power management in reliable RTSs. There are obvious limitations of the study; for instance, we do not consider a mix of computations with different power functions or tasks that finish before their predicted worst-case workload. We believe that our contributions are important and substantial in setting the stage for further work in this area, as it is important to understand the theoretical underpinning of the problem. We also show the somewhat surprising result of how power-aware TMR systems can be very competitive with power-aware duplex systems.

The remainder of this paper is organized as follows. Section 2 describes the computation and power management models, states the assumptions about the duplex and TMR systems, and illustrates the failure model. We examine duplex and TMR systems with no power management in Section 3, showing the superiority of the former type. We then examine power management techniques for duplex and TMR systems in Sections 4 and 5, respectively. Section 6 then compares the two types of architecture and establishes the somewhat surprising result of the vanishing difference between the two types of architectures under proper power management. Finally, Section 7 presents related work and Section 8 concludes the paper.

## 2. System Model

This section describes the computation and failure models, and shows how they interact with the power consumption model. It also describes our assumptions about duplex and TMR systems.

### 2.1. Computation Model

As usual in real-time research, we consider a set of tasks  $\tau_i$ , each of which has a period  $p_i$  associated with it, and a worst-case

execution time,  $c_i$ , which is required for execution assuming maximum processor frequency,  $f_{max}$ . We normalize the frequencies such that  $f_{max} = 1$ , and thus the worst-case execution time  $c_i$  can also be seen as the worst-case number of cycles needed to execute  $\tau_i$ . The utilization of each task is  $U_i = c_i/p_i$  and the total system utilization is given by  $U = \sum U_i$ . If  $U \leq 1$ , then the system is schedulable according to EDF scheduling. In case  $U < 1$ , we can allot more time than  $c_i$  for each task to execute, namely  $D_i = c_i/U$  and still guarantee that all tasks will finish within their deadlines. Each task will have an allotted amount of time to execute, which will not interfere with other tasks. Hence, in this paper, we will only consider a single task  $\tau$  with an allotted time  $D$ . We note that the same reasoning applies to scheduling algorithm other than EDF.

The problem we address is: “Run  $\tau$  to complete within  $D$  despite the possibility of one transient fault, while minimizing energy consumption.”

The opportunity to save energy depends on the amount of reduction in frequency that would still allow the computation to complete before the deadline. Intuitively, the slower the task can be executed, the more energy can be saved. If the frequency is  $f$ , then the time by which  $\tau$  completes is  $\frac{c}{f}$ . Since  $f_{max} = 1$ , to finish within  $D$ , we need<sup>1</sup>

$$\frac{c}{f} \leq D \tag{1}$$

Equation (1) gives the minimum frequency  $f_{min}$  at which the task must execute to meet the deadline,  $f_{min} = \frac{c}{D}$ . Let  $\sigma = \frac{f_{min}}{f_{max}} = \frac{c}{D}$ , which gives an intuitive indication of the relative load that  $\tau$  would impose on the system to finish by time  $D$ . As  $\sigma$  approaches 1, the processor will have to operate close to its maximum frequency and there is very little room for manipulating the frequency to conserve the dynamic component of energy. Inversely, as  $\sigma$  approaches 0, there is a greater degree of flexibility to reduce energy consumption by operating at a very low frequency. Note that there is a frequency that corresponds to the lowest voltage value below which the circuits cannot operate. Reducing the frequency below this value is not useful because it simply slows down the processor without an attendant saving in power (since the voltage cannot be lowered any further). For simplicity, we ignore this minimum voltage, assuming that the load imposed in the system is high enough. Systems with load less than that are not interesting and it is easy to incorporate such minimum voltage constraints in the model.

### 2.2. Energy Model

The power budget of a system consists of two components, namely static and dynamic. The static component represents the power that the system continuously draws regardless of the current level of activity. This power goes to refresh memory, keep the

<sup>1</sup>Note that  $f_{max} \neq 1$ , we can define  $f' = f/f_{max}$  and replace  $f$  by  $f'$  throughout the paper. Further, wherever  $c$  appears, we should have  $c/f_{max}$ .

peripheral devices up, maintain the static Random Access Memory in the processor’s caches, etc. The static power component also includes power lost to leakage currents in the system circuits. The dynamic component reflects the power that varies with the level of activity of the system. This power goes to switching the transistors and capacitors to execute instructions and manipulate data. Previous research has shown that the dynamic power component depends on the frequency of operation and the voltage levels for circuit switching [8]. Thus, the power consumed,  $P$ , can be expressed as  $P = P_s + \alpha fV^2$ , where  $P_s$  is the static power component,  $\alpha$  is a capacitance constant,  $f$  is the frequency of operation, and  $V$  is the switching voltage. Further, the frequency and voltage are related by a quasi-linear relation [21],  $V = f^\beta$ , where  $\beta \leq 1$  is a constant. This is true generally for voltages above a minimum value below which the circuits can no longer function due to noise and other factors.

For convenience, we express the power consumed as a function of frequency:

$$P = P_s + \alpha f^m \tag{2}$$

where  $2 \leq m \leq 3$ . Equation (2) establishes a method for reducing the dynamic power component of the budget by lowering the frequency [8]. In practice, the system reduces the processor frequency to the level that is sufficient to carry out the computation, and then it reduces the voltage to the minimum level that permits the desired frequency to be maintained. This is often called dynamic voltage scaling (DVS).

We adopt the conservative and simplifying assumption that the time a processor takes to complete  $\tau$  is inversely proportional to its operating frequency. This assumption is somewhat standard in power management literature [30]. It usually simplifies the analysis, and leads to conservative estimates of the energy that can be saved by manipulating the frequency of the system. To understand why, consider that at high CPU frequencies, the processor wastes more cycles (and consequently energy) upon cache misses due to the large difference between memory and processor speeds. At lower CPU frequencies, however, the processor does not wait as many cycles, leading to better energy utilization and less performance overhead.

### 2.3. Failure Model and Fault-Tolerant Architecture

The system is required to tolerate one transient fault. We consider two conventional architectures that can serve this purpose, namely duplex systems [22] and TMR systems [25].

#### 2.3.1. Duplex Systems

A traditional duplex system consists of two identical machines that run the same program and are controlled by a fault-tolerant clock. Each machine has its own storage and processor. We assume that an I/O device exists to synchronize input to, and out-

put from, the machines. Periodically, the two machines verify that their states are identical at pre-defined *synchronization points*. The verification can take place in various forms, for example by a straight comparison of the states, or by computing a hashing function of each state and comparing the results. If the states are identical, the state is saved in a checkpoint to enable restart should errors occur in the future. Otherwise, a transient fault must have occurred to cause the states of the machines to diverge, and thus the machines roll back to the saved state from the last successful synchronization.

A duplex system must reserve sufficient time to recover from one fault by rolling back to the previously synchronized state and re-executing from that point. We call this the *recovery time*. Take for example a computation  $\tau$  with  $\sigma = 0.6$ , a duplex system can execute  $\tau$  in  $0.6D$ . This leaves  $0.4D$  (i.e., 40% of the allotted time) for recovery time. Since this is not sufficient to re-execute the computation should a failure occur, therefore the duplex system must insert a synchronization point at about  $0.3D$ . The amount of computation *at risk* is therefore  $0.3D$ , well within the available time for recovery ( $0.4D$ ).

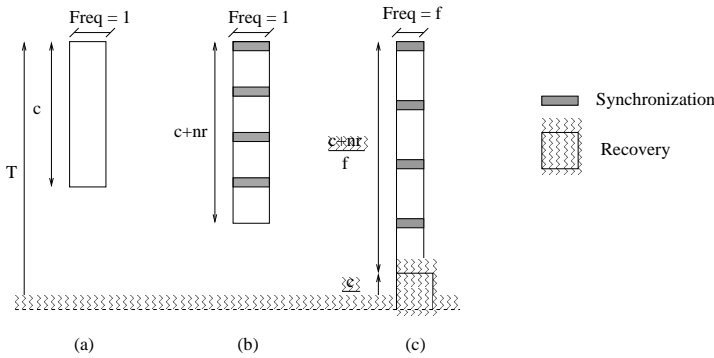
Clearly, a task  $\tau$  with  $\sigma = 1$  (i.e., must run at  $f_{max}$  to complete at exactly the deadline) cannot use a duplex system, because a fault would cause a rollback and thus a deadline to be missed. This observation leads to a tradeoff: As the value of  $\sigma$  gets closer to 1, the amount of recovery time that can be reserved gets smaller, and therefore the system must synchronize the states more frequently. Synchronizing the state, however, incurs overhead and consumes cycles that are otherwise available to the computation. Ultimately, the overhead of synchronization may consume all the available time and renders the computation impossible as  $\sigma$  progressively approaches 1.

The synchronization points serve the additional purpose of reducing energy consumption by allowing the system to execute at a lower speed/frequency. Let  $r$  be the number of cycles necessary to synchronize the state and take a checkpoint. Typically,  $r$  is much smaller than  $c$ . Assume further that state synchronization occurs at regular intervals. If the number of such synchronization points is  $n$ , and  $f$  is the desirable frequency to operate to lower energy consumption, then Equation (3) describes how the time can be allocated.

$$\frac{c + nr}{f} + \frac{c}{n} \leq D \tag{3}$$

Equation (3) states that the time is divided between running the computation itself ( $c$ ) and synchronizing states ( $nr$ ) at frequency  $f$ , and allowing for  $\frac{c}{n}$  time to roll back. Notice that we attempt to reduce the recovery time by running the processor at maximum frequency during rollback. This is a reasonable decision given that failures are infrequent and we want to reserve the minimum amount of time that allows for recovery. This strategy leaves as much time as possible to carry out the computation itself, allowing for the largest slowdown, saving the most amount of energy.

We can see now that synchronization points help in reducing energy consumption. By taking more synchronization points, the amount of time reserved for recovery  $\frac{c}{n}$  becomes smaller. There is a tradeoff, however, since the synchronization points consume time away from the available time to run the computation. The system thus reaches a point where adding more synchronization points can start having a negative effect by taking away too much time from the computation itself, forcing the processor frequency  $f$  to increase to satisfy Equation (3). Figure 1 illustrates this rea-



**Figure 1. Equally spaced synchronization points can help reduce energy consumption.**

soning: a computation is represented by a rectangle whose area is the number of CPU cycles needed for execution. The width of the rectangle represents the CPU execution frequency, and its height represents the time taken for execution. Figures 1(a) and 1(b) show the execution of  $\tau$ , at  $f_{max} = 1$ , without synchronization and with  $n = 4$  synchronization points, respectively. Figure 1(c) shows that we can reduce the frequency of executing  $\tau$  (with the 4 synchronization points) to  $f$  as long as the difference between the deadline,  $D$ , and the time for executing  $\tau$  at frequency  $f$  is at least enough for the overhead of synchronization points ( $nr$ ) and the time necessary for a potential rollback, ( $\frac{c}{n}$ ), at maximum speed.

### 2.3.2. TMR Systems

A TMR system consists of three identical machines that run the same program and are controlled by a fault-tolerant clock. Each machine has its own independent storage and processor. A fault-tolerant voting device receives the output of the three machines and compares them. If all output values are identical, the voting device releases them and computation continues. Otherwise, one of the output values diverges from the output in the other two machines. In this case, an error must have occurred in the failed machine and the voting device releases the value of the two functioning machines. The malfunctioning machine is either taken off-line for (permanent faults) repair or is rejuvenated by copying the state of one of the functioning machines (transient faults).

When reliability and performance are concerned, a TMR system has more desirable features than a duplex system. A TMR system can tolerate the loss of one machine (effectively turning itself into a duplex system).<sup>2</sup> It also incurs lower performance overhead than a duplex system during failure-free operation, since it does not need to pay the price for the state synchronizations. Additionally, TMR may be the only viable choice for computations with  $\sigma$  close to 1, since such workloads may not be able to afford the additional time that must be reserved for rollback in duplex systems. On the other hand, one would intuitively conclude that a duplex system is likely to be more energy-efficient, since it has only two machines. Therefore, a tradeoff is emerging: a duplex system offers less reliability and higher performance overhead, but consumes less energy, and vice versa for TRM systems.

## 3. Base Case: No Power Management

In this section, we give a quantitative evaluation of the difference between duplex and TMR systems, with respect to energy consumption while guaranteeing deadlines even if a transient fault occurs, when no power management is deployed (that is, tasks run at  $f_{max}$ ). Under such conditions, the processors of a duplex system operate at maximum frequency to execute  $\tau$ , finishing by time  $T_d$  which consists of the time to run  $\tau$  (i.e.,  $\sigma D = c$ ) and the overhead allocated to synchronization points:

$$T_d = c + nr = \sigma D + nr \quad (4)$$

Define  $\rho = \frac{r}{D}$ , which expresses the overhead of one state synchronization operation relative to the available time  $D$ . The higher the value of  $\rho$  is, the higher the synchronization overhead. We thus obtain  $T_d = D(\sigma + n\rho) = c + nr$ . The energy consumed by the duplex system consists of the energy consumed by the dynamic power component for the duration  $T_d$  and the static power component for the entire duration  $D$  for both processors:

$$E_d = 2 \times (T_d \alpha f_{max}^m + DP_s) \quad (5)$$

The TMR system, using a similar analysis, finishes by time  $T_t = \sigma D = c$  as the processors also run at maximum frequency. The energy is thus given by

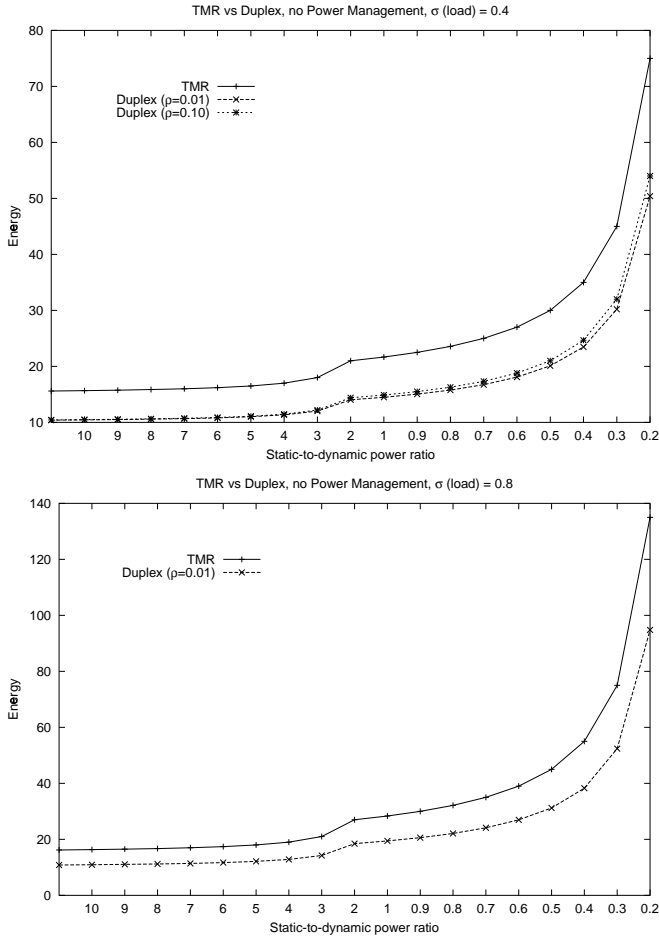
$$E_t = 3 \times (T_t \alpha f_{max}^m + DP_s) \quad (6)$$

The difference between the energy consumed by both types of systems is (from Equations 4- 6):

$$\delta E = E_t - E_d = c \alpha f_{max}^m + DP_s - 2nr \alpha f_{max}^m \quad (7)$$

Equation (7) predictably states that compared to a duplex system, a TMR system has to pay the static and dynamic energy components for the additional machine that it uses. On the other hand,

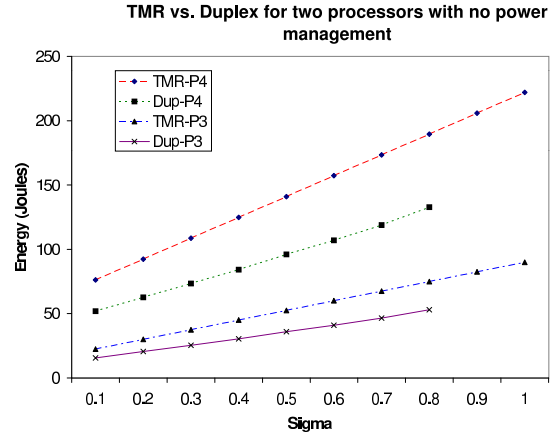
<sup>2</sup>Our assumption about the failure model is one transient fault during a computation. So, the advantage that a TMR system has over a duplex in this regard is not centric to our discussion, as we do not consider permanent machine failures.



**Figure 2. Energy consumption for Duplex vs. TMR systems with no power management**

the duplex system incurs the energy consumed by the periodic synchronization. This equation supports the intuitive notion that a duplex system is more energy-efficient, unless the overhead of synchronization or the number of synchronization points is excessive.

Figure 2 shows a comparison between the two types of systems for two values of  $\sigma$  and of  $\rho$ . For each value, a chart shows the variation of the energy as a function of the relative ratio of  $P_s/\alpha$ , which characterizes the different machine “types” (here  $P_s = 5W$  [14], but the absolute numbers here are not important). The left side of the horizontal axis represents machines where the static power consumption dominates, such as embedded systems in which a low-power embedded processor is a small contributor to the entire system’s power budget. The right side of the horizontal axis represents machines where the dynamic component prevails, such as modern systems with a high-end, high-performance processor whose power consumption is a substantial portion of overall system’s power budget. For example, the value



**Figure 3. Energy consumption for TMR and Duplex for two processor types with no power management.**

of  $P_s/\alpha = 0.2$  in the figure corresponds to a Pentium III system that we measured in our laboratory [3]. The different values of  $\sigma$  show how the systems act in response to different workloads, with the workload increasing with the value of  $\sigma$  (recall that  $\sigma = c/D$  and note the different scales of the Y axes for each graph). Finally, we analyze for different values of  $\rho$  to show the effect of the overhead of state synchronization in duplex systems. We have experimented with  $\rho$  ranging from 0.01 to 0.1, which are representative of the overhead found in checkpointing systems. For clarity, we show results only for 0.01 and 0.1.

By inspecting the graphs, there are two important observations. First, the TMR system consistently consumes more energy than the duplex system, by a wide margin. Predictably, this is due to the fact that for typical values, the overhead of synchronization in a duplex system does not cause sufficient energy consumption compared to the power drawn by the third machine in a TMR system. Second, as  $\sigma$  increases, the duplex system becomes less of a viable option. In fact, for  $\sigma \geq 0.8$ , the duplex system cannot function for  $\rho \geq 0.01$ . At such high values of  $\sigma$ , it is not possible to reserve recovery time regardless of the number of synchronization points. This case is illustrated in Figure 3, which shows the energy consumption as the workload ( $\sigma$ ) varies when two systems are built using Pentium IV (P4) and Pentium III (P3) processors. The figure assumes  $\rho = 0.01$ , and shows the duplex system unable to function (i.e., misses deadlines) beyond  $\sigma = 0.8$ . Again, the figure also reinforces the first conclusion about the superiority of duplex systems in energy consumption when the workload is low (small values of  $\sigma$ ).

To summarize, when there is no power management, the analysis establishes the superiority of duplex over TMR systems in most situations with respect to energy consumption, except for high-intensity workloads. We now turn our attention to applying power management to both duplex and TMR systems.

## 4. Power Management For Duplex Systems

We examine two policies for conserving energy in duplex systems. The first policy is dynamic voltage scaling and targets the dynamic power component of the system. The second, hibernation, targets the static power component of the system.

### 4.1. Dynamic Voltage Scaling

Above we discussed the tradeoff of having fewer or more synchronization points in a task. This was based on Equation (3), which established a safety requirement for the duplex system. We now derive the frequency,  $f$ , that will allow  $\tau$  to finish within  $D$ , while minimizing the dynamic component of the power consumption and allowing for the synchronization overhead, the time to roll back from a fault and the time for re-execution. Hence, from Equation 3,  $f$  can be given by

$$f \geq \frac{c + nr}{D - \frac{c}{n}} = \frac{n\sigma + n^2\rho}{n - \sigma} \quad (8)$$

For a given number of synchronization points,  $n$ , Equation (8) establishes a safety requirement by defining a lower bound on the frequency at which the processor should execute to finish  $\tau$  within its deadline. Reducing the frequency below this level will reduce the energy required to run  $\tau$ , but recovery within the deadline may not be possible if a fault occurs.

The energy,  $E$ , consumed during the execution of  $\tau$  is the product of the time it takes to execute  $\tau$  and power consumed [8]. Thus:

$$E = 2 \times ((P_s D) + (\alpha f^m) (\frac{c + nr}{f})) \quad (9)$$

The goal of the analysis is to find  $n$  which gives the frequency that minimizes the energy consumption given by Equation (9) subject to the constraints in Equation (8). Since this frequency is a function of  $n$  from Equation (8), we can derive the value of  $n$  which minimizes  $E$  by differentiating Equation (9) with respect to  $n$  and equating the result to zero. Note from Equation (8) that  $f$  depends on  $n$ , and  $\frac{df}{dn}$  denotes the differentiation of  $f$  with respect to  $n$ .

$$\begin{aligned} \alpha(m-1)f^{m-2}(c+nr)\frac{df}{dn} + \alpha f^{(m-1)}r &= 0 \\ (m-1)(c+nr)\frac{df}{dn} + fr &= 0 \\ (m-1)(c+nr)(n\sigma + n^2\rho) - (n-\sigma)(\sigma + 2n\rho) + \\ r(n\sigma + n^2\rho)(n-\sigma) &= 0 \end{aligned}$$

This results in a cubic equation in  $n$  with two non-positive solutions (which we ignore), and one optimal positive value of  $n$  given by:

$$n = \frac{\sigma}{2m} \left( (2m-1) + \sqrt{(2m-1)^2 + \frac{4m(m-1)}{\rho}} \right) \quad (10)$$

For the common case of  $m = 3$ , we obtain  $n = \frac{\sigma}{6} (5 + \sqrt{25 + \frac{24}{\rho}})$  and for  $m = 2$ , we obtain  $n = \frac{\sigma}{4} (3 + \sqrt{9 + \frac{8}{\rho}})$ . When the optimal  $n$  is not an integer, an approximation must be made by taking  $\lfloor n \rfloor$  or  $\lceil n \rceil$  depending on which value yields lower energy consumption.

### 4.2. Hibernation and voltage scaling

In this mode, the system puts itself in a hibernation mode when it does not have a computation to execute. The important advantage of this mode is that the static component of the power consumption is all but eliminated when the system idles. Modern technology supports hibernation in a manner that allows the processor and the system in general to respond almost immediately to new events or requests. Hibernation support may consist, for instance, of putting the processor in a “deep sleep” and “deeper sleep” modes [15], and switching the memory off the bus and putting it in self-refresh mode. Using this technology, the processor and main memory consume almost no energy and can be back online in a few milliseconds, or even microseconds. Our measurements in the lab show that with the energy consumed by memory drops by 99% in hibernation mode, while the specifications of the Intel Pentium III processor show that power drops to 0.47W in “deeper sleep mode” from a maximum of 22W, a ratio of 98% [15].

Hibernation motivates “finishing” the computation as soon as possible (i.e., at the highest speed) so that the static component of the power consumption can be conserved. On the other hand, dynamic voltage scaling requires that the processor execute at the minimum speed to minimize the dynamic power component. Expectedly, the tradeoff depends on the actual constants that characterize the power and energy equation. We now find the frequency that would minimize the total energy consumption. Assuming that energy during hibernation is insignificant, Equation (5) reduces to:

$$E_d = 2 \times (P_s + \alpha f^m) \times (\frac{c + nr}{f}) \quad (11)$$

We can find  $f$  that minimizes the energy consumption by minimizing  $E_d$ , subject to the safety constraint specified by Equation (8). It is not clear if there is a closed form for the solution. We solve the equation iteratively and, as discussed in the next section, the solution will depend on the “type of machine” that is used by the system. For those where the static power consumption dominates, setting the frequency to the maximum proves to be most advantageous, and vice versa.

### 4.3. Evaluation

Figure 4 shows a comparison of relative energy consumption between three cases, namely no power management (NoPM), dynamic voltage scaling (DVS), and DVS+hibernation (Hibern), for various values of  $\sigma$  and with  $\rho = 0.01$ . The conclusions for other

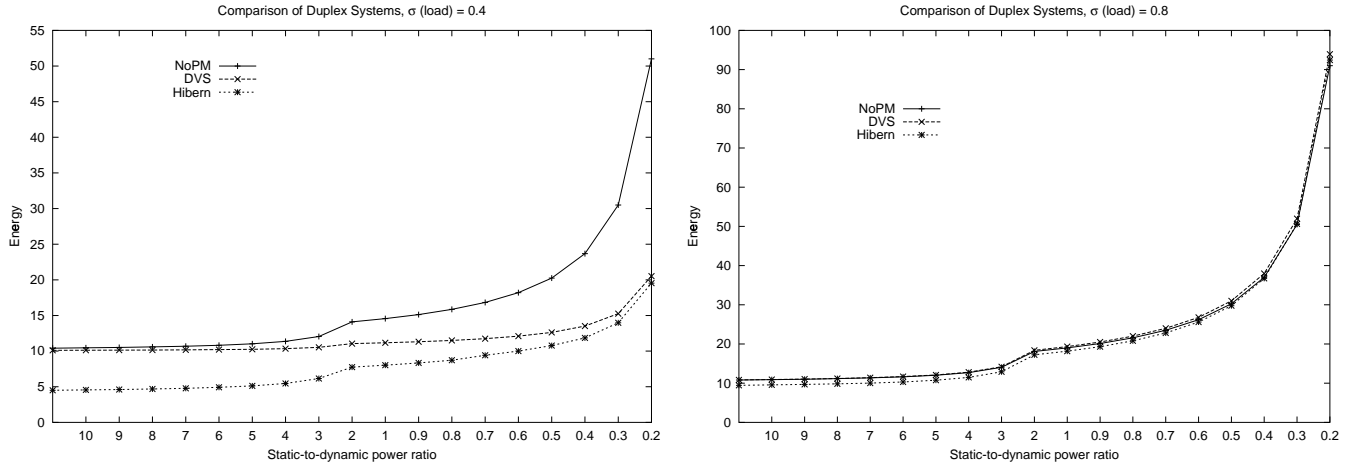


Figure 4. Effectiveness of power management for duplex systems,  $\rho = 0.01$

values of  $\sigma$  and  $\rho$  are essentially the same and we do not show the other figures for brevity. The notation and organization of the figure are identical to Figure 2.

As the workload increases (higher values of  $\sigma$ ), the energy consumption increases—note the different scales of the Y axes in each graph—and the differences among the schemes blur. This is predictable, since higher workloads require the machine to operate close to the maximum frequency for almost the entire duration, negating any benefit from dynamic voltage scaling or hibernation. Further, for machines where the static component dominates (left side of X axis), hibernation is most effective, as it eliminates the constant draw of energy. However, for machines where the dynamic power component dominates (right part of the X axis in each graph), the performance of Hibern approaches that of DVS, since the influence of the static component decreases.

In practice, since different computations will have different values of  $\sigma$ , it follows that applying the hibernation policy combined with DVS will yield the best results in general, outperforming the other two policies for machines where the static power component dominates, and performing as good as the other policies when either the dynamic power component dominates or the workload is too high.

## 5. Power Management For TMR Systems

### 5.1. Dynamic Voltage Scaling

Dynamic voltage scaling is much simpler for TMR systems compared to Duplex systems. Simply, since for idle processors the static component of power consumption is fixed for the duration  $D$ , the frequency  $f$  (or  $f'$  if  $f_{max} \neq 1$ ) is set to  $\sigma$ . This guarantees that the processors execute at the minimum frequency that allows the computation to complete within the deadline. As a result, the dynamic component of the power consumption is re-

duced to the minimum on every machine.

### 5.2. Hibernation and Voltage Scaling

Similar to the hibernation mode in duplex systems, the TMR system puts itself in a hibernation mode when there are no tasks to execute. This mode is combined with dynamic voltage scaling to strike a balance that minimizes overall energy consumption. The tradeoff still exists between the need to execute quickly and minimize the static power consumed and to execute slowly to reduce the dynamic power component. Recalling that  $P = P_s + \alpha f^m$  from Equation (2), we now find the frequency that would minimize the total energy consumption:

$$E_{TMR} = 3 \times P \times \frac{c}{f} = 3c \left( \frac{P_s}{f} + \alpha f^{(m-1)} \right) \quad (12)$$

We find  $f$  that minimizes  $E$  by solving the differential equation  $\frac{dE}{df} = 0$  which yields

$$f = \sqrt[m]{\frac{P_s}{\alpha(m-1)}} \quad (13)$$

Note that typically  $f > \sigma$ , unless the load is very low; in other words, the task will typically finish well ahead of its deadline. In any case, we need to guarantee that  $\frac{c}{f} \leq D$  and therefore we set  $f = \max \left\{ \sqrt[m]{\frac{P_s}{\alpha(m-1)}}, \frac{c}{f}, f_{min} \right\}$ .

### 5.3. Optimistic TMR Systems—A Step Further

Further optimization of energy consumption is possible if we consider that state divergence (caused by faults) is rare. We use this fact to devise the following mechanism: we run two of the machines that belong to the TMR system at the frequency computed from Equation (13). If no failure occurs, the votes of the two machines will be identical and can be released, at which time

the three machines are put in hibernate mode. The third machine serves as a *slow* arbiter in the case if state divergence is detected when the two main machines vote their output. In such a case, the third machine will compute its output and acts as a “tie-breaker,” selecting the output that matches its own as the correct one. Thus, at the beginning of the computation we run the third machine at the minimum possible frequency that allows it to make enough progress such that if state divergence is detected, it can “take over” and compute at maximum frequency to break the stalemate before the deadline. Depending on the workload, the third machine could be put to hibernate while the other two are computing, or we may have to set it to execute at a low frequency that consumes the least possible power and yet allow the machine to take over and produce the output if needed. We call this scheme *optimistic TMR*. The optimism stems from the assumption that the two other machines will always vote identically.

We illustrate the idea by the following example. Suppose  $\sigma = 0.3$ , and that  $f = f_{max}$ . The two machines will finish at time  $0.3D$ . Since there is ample time to run the computation should state divergence is detected at time  $0.3D$ , the third machine is put to hibernate from the beginning. If the two machines agree in their votes, then we have produced the correct output without consuming any energy in the third machine. If state divergence, however, is detected, the third machine can execute and produce the needed result by time  $0.6D$ .

Note that when  $\sigma$  increases, we may have to put the third machine to work at a low frequency. Consider the example where  $\sigma = 0.6$ , and again we run at  $f = f_{max}$ . Again, the output will be ready by  $0.6D$ , leaving about  $0.4D$  for the third machine to compute the output. Therefore, the third machine must be put to execute a  $0.2D$  worth of computation just in case state divergence is detected. Therefore, the third machine should be put to work at  $0.33f_{max}$  for the interval up to time  $0.6D$ . This will guarantee that the third machine can take over and compute the required output by time  $D$ . In fact we can use the scheme in Section 5.2 to determine the frequency at which the third machine is to execute, given that it may or may not need to execute at the highest frequency.

#### 5.4. Evaluation

Figure 5 shows a comparison of relative energy consumption between four cases, namely no power management (NoPM), dynamic voltage scaling (DVS), DVS+hibernation (Hibern), and the optimistic TMR system (Optim). The notation and organization of the figure are identical to Figures 2 and 4 and will not be repeated here for brevity. The figure shows the analysis for  $\rho = 0.01$  and  $\sigma = 0.4, 0.8$ , but results are similar for other values of  $\sigma$  and  $\rho$ .

Similar to the case of duplex systems, the differences among the schemes blur for high values of  $\sigma$ . Again, higher workloads require the machine to operate close to the maximum frequency

for almost the entire duration, negating any benefit from dynamic voltage scaling or hibernation, and preventing the third machine from being put to work at a sufficiently low frequency to produce a substantial saving in energy consumption. Further, the optimistic TMR system performs best or at least as good as the other schemes. The savings are more pronounced for machines where the static-power component dominates, or for small values of  $\sigma$ , or both. This demonstrates the effectiveness of the optimistic TMR method.

It follows from the analysis that using optimistic TMR is preferable when hardware redundancy is needed.

### 6. Comparison Between Duplex and TMR Systems

Now we turn to the central question in this paper. We have established in Section 3 that duplex systems are superior to TMR systems in energy consumption absent power management and for workloads that allow a duplex system to be deployed ( $\sigma$  not too close to 1). It is useful now to consider the effect of incorporating the power management techniques described in Sections 4 and 5 on the overall comparison.

Figure 6 shows a comparison of relative energy consumption between the optimistic TMR and three duplex systems that use hibernation and dynamic voltage scaling, but differ in the synchronization overhead  $\rho$ . The notation and organization of the figure are identical to Figures 2 and 4 and will not be repeated here for brevity.

Analyzing the data in the figure shows that there is no substantial difference in energy consumption among the four configurations under study. Predictably, duplex systems with higher  $\rho$  tend to consume more energy. But the striking conclusion that the figure portrays is that with proper power management, a TMR system is competitive with duplex systems. In the case where the workload was low, the TMR system consistently consumed less energy than the duplex systems with the parameters that we studied. To understand why, we observed that, under low  $\sigma$ , the optimistic TMR system effectively behave like a duplex system without the overhead of synchronization. The third machine will be effectively turned off all of the time of fault-free operation since, even after the two functioning machines produce their votes, there will be sufficient time for the third machine to do its own computation in case an error is detected (outputs disagree).

When the load is moderate ( $\sigma = 0.4$ ), the TMR system’s behavior is almost indistinguishable from the three duplex systems that were studied. There are two reasons for this, first the duplex systems do not operate at the optimal frequency point because the optimal value of  $n$  is not an integer. Additionally, the overhead of synchronization and hibernation conspire to reduce the energy consumption advantage of the duplex systems.

Finally, as  $\sigma$  approaches 1, the effectiveness of power management in general decreases across the two types of systems. This

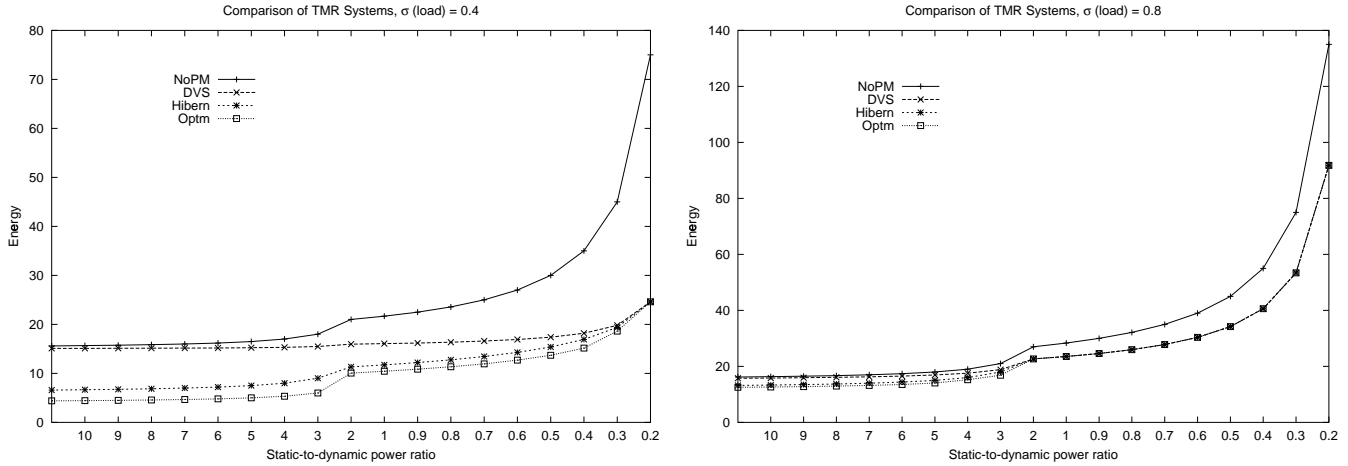


Figure 5. Effectiveness of power management for TMR systems,  $\rho = 0.01$ .

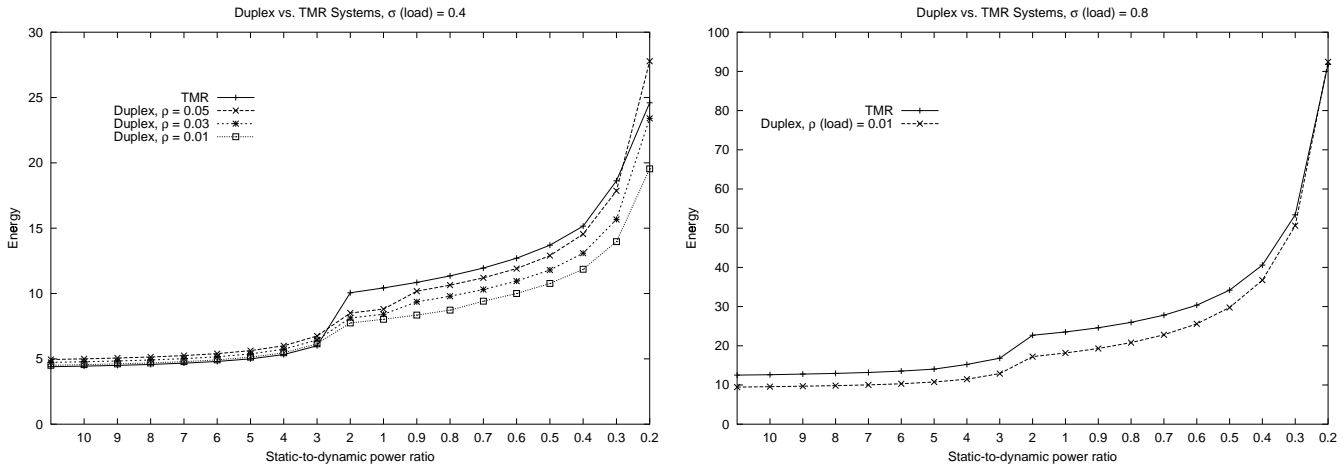


Figure 6. A comparison between duplex and TMR systems when using the most effective power management policy for each.

makes the TMR system consume more energy as it becomes more difficult to hibernate the machines (see the figure for  $\sigma = 0.8$ ), with a difference that reaches 25% over a duplex system. However, in that range, the duplex system becomes very sensitive to the synchronization overhead and fewer configurations can operate in that range (in fact only for  $\rho \leq 0.01$ ), thus negating the advantage that duplex systems may have as  $\sigma$  increases.

To summarize, proper power management techniques make it possible for TMR systems to wipe out their disadvantages in energy consumption compared to duplex systems. At low-intensity workloads, an optimistic TMR system actually outperforms a duplex system. At moderate workloads, a duplex system behaves slightly better than a TMR system, but the energy consumption is almost indistinguishable between the two system types. Only when  $\sigma$  approaches 1 that the duplex system restores some of its

advantage, but only to give them up quickly as the sensitivity to the synchronization overhead make them infeasible in that range. This rather surprising result concludes our study.

## 7. Related Work

Duplex and TMR systems are standard techniques for building fault-tolerant systems. Vaidya compared the reliability obtained from both techniques [29], and Pradhan and Vaidya described duplex systems in great detail [22]. Siewiorek and Swarz describe early implementations of TMR systems along with some fundamental concepts [25] and countless references are available on the subject.

Both the distribution of power consumption and methods for managing it have been studied extensively in conventional sys-

tems such as laptops, personal digital assistants (PDAs), and servers. For example, one power consumption study is a detailed analysis of the energy consumed by the various components of Apple Macintosh laptop computers [17]. Other studies have proposed policies for managing energy use in server clusters by powering machines on and off [3, 4]. However, we are not aware of any work that addresses energy management in fault-tolerant duplex and TMR systems.

Many processor architectures and microarchitectures incorporate power-saving features [15]. More recently developed and less widely deployed today are new memory chip architectures that are incorporating similar “spin down” states so that the system can actively manage the power used by main memory [23]. In addition, several research efforts are focusing on new power management mechanisms employed at the operating system [28] and application layers [7]. Techniques for dynamically controlling processor temperature [24] have also been used to indirectly control energy consumption. Our work builds on these studies by demonstrating the value of these mechanisms in fault-tolerant systems.

DVS was first proposed and studied in [30], but it has been further explored in a variety of contexts. For example, for RTSs, DVS schemes focus on minimizing energy consumption in the system while still meeting the deadlines. Yao et al. [31] provided a static off-line scheduling algorithm, assuming aperiodic tasks and worst-case execution times (WCET). Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of periodic requests are proposed in [12]. Non-preemptive power aware scheduling is investigated in [11]. For periodic tasks with identical periods, the effects of having an upper bound on the voltage change rate are examined in [13]. Slowing down the CPU whenever there is a single task eligible for execution was explored in [27]. DVS in the context of soft deadlines was investigated in [18]. Cyclic and EDF scheduling of periodic hard real-time tasks on systems with two (discrete) voltage levels have been investigated in [16]. The static solution for the general periodic model where tasks have potentially different power characteristics is provided in [1, 2]. Real-time applications exhibit a large variation in actual execution times [6] and WCET is too pessimistic. Thus, a lot of research was directed at dynamic slack-management techniques [2, 10, 20, 26]. Many other DVS papers appeared in recent conferences and workshops, such as COLP (Compiler and Operating Systems for Low Power) and PACS (Power-Aware Computing Systems).

Clearly, commercial efforts have been underway to put in place standards for power management. Examples include ACPI (Advanced Configuration and Power Interface [5]), and Microsoft’s OnNow initiative [19]. It is conceivable that some of these standards could be of use in energy-aware reliable systems. However, we believe that such use could prove problematic since all these standards were drafted without consideration to reliability. It is an open question whether energy-aware fault-tolerant systems will need their own set of standards or they could use existing ones, or

modifications thereof.

## 8. Conclusions and Future Work

We have presented a simple theory for power management in duplex and TMR systems when a limited amount of time is available for execution of tasks. This theory confirms the intuition that duplex systems are superior to TMR systems when no power management is deployed. We then developed several techniques for power management in duplex systems, showing how to insert synchronization points to enable DVS and still guarantee deadlines. Analysis of this method shows the sensitivity to the overhead of state synchronization and that this sensitivity becomes a limiting factor that may not allow these systems to function beyond a certain workload factor. We then combined hibernation which targets the static power component of the system with DVS to arrive at a policy that works well for a wide range of workloads and synchronization overhead.

We then turned our attention to TMR systems, showing with a simpler analysis how DVS and hibernation could be combined to improve energy consumption drastically compared to the base case. The best configuration was obtained from what we call optimistic TMR systems that attempt to turn off or slow down one of the three machines to conserve energy. The result is a TMR system that can have the same reliability and performance characteristics of traditional ones, but with an energy consumption profile that almost matches that of the optimized duplex systems. This somewhat surprising result states that TMR systems, with proper power management, can be competitive with duplex systems while providing better fault coverage.

While we are satisfied with this first step, there is a lot of work that needs to be developed. Understanding energy consumption during failure recovery is of interest. In this paper, we considered failures rare and acted accordingly. In failure-prone environments such as space ships, it will be necessary to revise these policies to accommodate frequent clustered failures. It is also necessary to understand how this model can be extended to handle the interactions of the system with the surrounding environment or other systems. We are currently working on these issues.

## References

- [1] H. Aydin, R. Melhem, D. Mossé, P.M. Alvarez: Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics, *Proceedings of the 13<sup>th</sup> Euromicro Conference on Real-Time Systems (ECRTS’01)*, Delft, Netherlands, June 2001
- [2] H. Aydin, R. Melhem, D. Mossé and P. M. Alvarez: Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems, *Proceedings of Real-Time Systems Symposium, 2001*

- [3] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The Case for Power Management in Web Servers. In Robert Graybill and Rami Melhem, editors, *Power-Aware Computing*. Kluwer/Plenum Series in Computer Science, January 2002.
- [4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *18th Symposium on Operating Systems Principles (SOSP)*, October 2001.
- [5] Compaq et al. ACPI Specification, Version 2.0, 2000.
- [6] R. Ernst and W. Ye: Embedded Program Timing Analysis based on Path Clustering and Architecture Classification, *Computer-Aided Design (ICCAD'97)*, pp. 598-604, 1997
- [7] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 48–63, 1999.
- [8] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, September 1996.
- [9] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithm for Dynamic Speed-Setting of a Low-Power CPU. In *Mobile Computing and Networking*, 1995.
- [10] F. Gruian: Hard Real-Time Scheduling Using Stochastic Data and DVS Processors, *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 46-51, 2001
- [11] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava: Power Optimization of Variable Voltage Core-based Systems, *Proceedings of the 35<sup>th</sup> Design Automation Conference (DAC'98)*, 1998
- [12] I. Hong, M. Potkonjak and M. Srivastava: On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processors, *Computer-Aided Design (ICCAD'98)*, pp. 653-656, 1998
- [13] I. Hong, G. Qu, M. Potkonjak and M. Srivastava: Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors, *Proceedings of the 19<sup>th</sup> IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, December 1998
- [14] Intel Corporation. Pentium III Technical Specifications, 2000.
- [15] Intel Corporation. Mobile Intel Pentium III Processor-M Datasheet October 2001. Order Number: 298340-002, October 2001.
- [16] C. M. Krishna and Y. H. Lee: Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems, *Proceedings of the 6<sup>th</sup> IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D. C., May 2000
- [17] J.R. Lorch and A.J. Smith. Energy Consumption of Apple Macintosh Computers. *IEEE Micro*, 18(6), November/December 1998.
- [18] J. R. Lorch and A. J. Smith: Improving Dynamic Voltage Scaling Algorithms with PACE, *Proceedings of the ACM SIGMETRICS 2001 Conference, Cambridge, MA, June 2001*
- [19] Microsoft Corp. *PC99 System Design Guide*. Microsoft Press, 1999.
- [20] D. Mossé, H. Aydin, B. Childers, R. Melhem: Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications, *Workshop on Compilers and Operating Systems for Low Power (COLP'00)*, Philadelphia, PA, October 2000
- [21] Kevin Nowka. Private communication. IBM Research, Austin, TX.
- [22] D. K. Pradhan and N. H. Vaidya. Roll-forward and rollback recovery: Performance-reliability trade-off. In *FTCS-24: 24th International Symposium on Fault Tolerant Computing*, pages 186–195, Austin, Texas, 1994. IEEE Computer Society Press.
- [23] Rambus Corporation. Rambus Technology Overview, Feb 1999.
- [24] Erven Rohou and Michael D. Smith. Dynamically Managing Processor Temperature and Power. In *2nd Workshop on Feedback-Directed Optimization*, November 1999.
- [25] D. Siewiorek and R. Swarz. The theory and practice of reliable system design, 1982.
- [26] D. Shin, J. Kim and S. Lee: Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications, *IEEE Design and Test of Computers*, 18(23):20-30, March 2001
- [27] Y. Shin and K. Choi: Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems, *Proceedings of the 36<sup>th</sup> Design Automation Conference (DAC'99)*, 1999
- [28] A. Vahdat, A. Lebeck, and C. Ellis. Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. In *9th ACM SIGOPS European Workshop*, September 2000.
- [29] Nitin H. Vaidya. Comparison of duplex and triplex memory reliability. *IEEE Transactions on Computers*, 45(4):503–507, 1996.
- [30] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *First Symposium on Operating Systems Design and Implementation*, pages 13–23, Monterey, California, U.S., 1994.
- [31] F. Yao, A. Demers and S. Shankar: A Scheduling Model for Reduced CPU Energy, *IEEE Annual Foundations of Computer Science*, pp. 374 - 382, 1995