Dynamic Power Management in Wireless Sensor Networks

Amit Sinha

Anantha Chandrakasan
Massachusetts Institute of Technology

Power-aware methodology uses an embedded microoperating system to reduce node energy consumption by exploiting both sleep state and active power management.

WIRELESS DISTRIBUTED microsensor networks have gained importance in a wide spectrum of civil and military applications.[1] Advances in MEMS (microelectromechanical systems) technology, combined with low-power, low-cost digital signal processors (DSPs) and radio frequency (RF) circuits have resulted in the feasibility of inexpensive and wireless microsensor networks. A distributed, self-configuring network of adaptive sensors has significant benefits. They can be used to remotely monitor inhospitable and toxic environments. A large class of benign environments also requires the deployment of a large number of sensors such as for intelligent patient monitoring, object tracking, and assembly line sensing. These networks' massively distributed nature provides increased resolution and fault tolerance compared to a single sensor node. Several projects that demonstrate the feasibility of sensor networks are underway.[2]

A wireless microsensor node is typically battery operated and therefore energy constrained. To maximize the sensor node's lifetime after its deployment, other aspects—including circuits, architecture, algorithms, and protocols—have to be energy efficient. Once the system has been designed, additional energy savings can be attained by using dynamic power management (DMP) where the sensor node is shut down if no events occur.[3] Such event-driven power consumption is critical to maximum battery life. In addition, the node should have a graceful energy-quality scalability so that the mission lifetime can be extended if the application demands, at the cost of sensing accuracy.[4] Energy-scalable algorithms and protocols have been proposed for these energy-constrained situations.

Sensing applications present a wide range of requirements in terms of data rates, computation, and average transmission distance. Protocols and algorithms have to be tuned for each application. Therefore embedded operating systems (OSs) and software will be critical for such microsensor networks because programmability will be a necessary requirement.

We propose an OS-directed power management technique to improve the energy efficiency of sensor nodes. DPM is an effective tool in reducing system power consumption without significantly degrading performance. The basic idea is to shut down devices when not needed and wake them up when necessary. DPM, in general, is not a trivial problem. If the energy and performance overheads in sleep-state transition were negligible, then a simple greedy algorithm that makes the system enter the deepest sleep state when idling would be perfect. However, in reality, sleep-state transitioning has the overhead of storing processor state and turning off power. Waking up also

0740-7475/01/$10.00 © 2001 IEEE

takes a finite amount of time. Therefore, implementing the correct policy for sleep-state transitioning is critical for DPM success.

While shutdown techniques can yield substantial energy savings in idle system states, additional energy savings are possible by optimizing the sensor node performance in the active state. Dynamic voltage scaling (DVS) is an effective technique for reducing CPU (central processing unit) energy.[5] Most microprocessor systems are characterized by a time-varying computational load. Simply reducing the operating frequency during periods of reduced activity results in linear decreases in power consumption but does not affect the total energy consumed per task. Reducing the operating voltage implies greater critical path delays, which in turn compromises peak performance.

Significant energy benefits can be achieved by recognizing that peak performance is not always required and therefore the processor's operating voltage and frequency can be dynamically adapted based on instantaneous processing requirement. The goal of DVS is to adapt the power supply and operating frequency to match the workload so the visible performance loss is negligible. The crux of the problem is that future workloads are often nondeterministic.

The rate at which DVS is done also has a significant bearing on performance and energy. A low update rate implies greater workload averaging, which results in lower energy. The update energy and performance cost is also amortized over a longer time frame. On the other hand, a low update rate also implies a greater performance hit since the system will not respond to a sudden increase in workload.

We propose a workload prediction strategy based on adaptive filtering of the past workload profile and analyze several filtering schemes. We also define a performance-hit metric, which we use to judge the efficacy of these schemes. Previous work evaluated some DVS algorithms on portable benchmarks.[6]

System models

The following describes the models and policies, derived from actual hardware implementation.

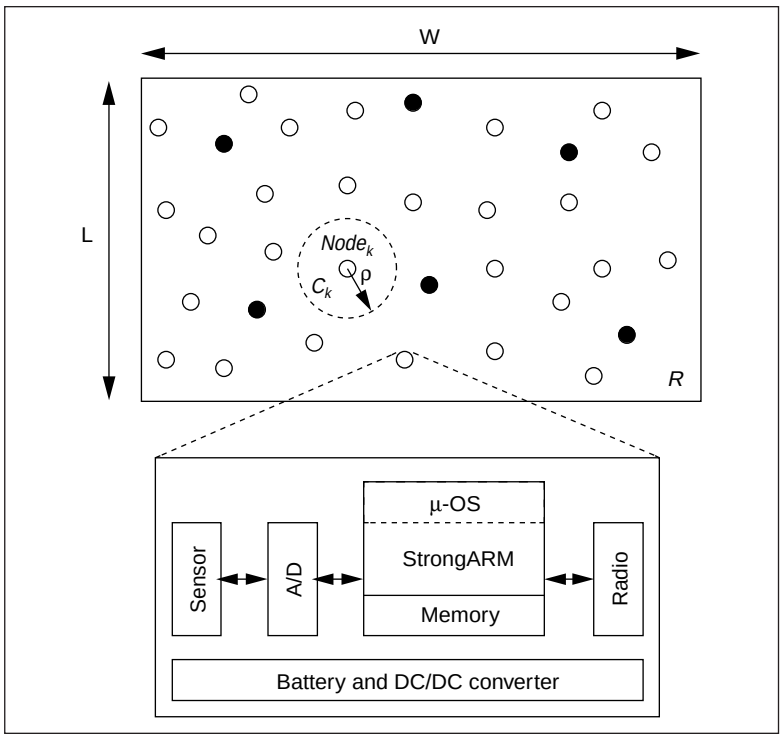

Figure 1. Sensor network and node architecture.

Sensor network and node model

The fundamental idea in distributed-sensor applications is to incorporate sufficient processing power in each node so that they are self-configuring and adaptive. Figure 1 illustrates the basic sensor node architecture. Each node consists of the embedded sensor, analog-digital converter, a processor with memory (which, in our case, is the StrongARM SA-1100 processor), and the RF circuits. Each component is controlled by the microoperating system (μOS) through microdevice drivers. An important function of the μOS is to enable power management. Based on event statistics, the μOS decides which devices to turn off and on.

Our network essentially consists of η homogeneous sensor nodes distributed over rectangular region ρ with dimensions $W \times L$. Each node has visibility radius r. Three different communication models can be used for such a network:

- direct transmission (every node transmits directly to the base station),
- multihop (data is routed through the individual nodes toward the base station), and
- clustering.

Table 1. Useful sleep states for the sensor node.

Sleep state	StrongARM	Memory	Sensor, analog-digital converter	Radio
s_0	Active	Active	On	Tx, Rx
s_1	Idle	Sleep	On	Rx
s_2	Sleep	Sleep	On	Rx
s_3	Sleep	Sleep	On	Off
s4	Sleep	Sleep	Off	Off

Tx=transmit, Rx=receive.

If the distance between the neighboring sensors is less than the average distance between the sensors and the user or the base station, transmission power can be saved if the sensors collaborate locally. Further, it's likely that sensors in local clusters share highly correlated data. Some of the nodes elect themselves as cluster heads and the remaining nodes join one of the clusters based on minimum transmission power criteria. The cluster head then aggregates and transmits the data from other cluster nodes. Such application-specific network protocols for wireless microsensor networks have been developed. They demonstrate that a clustering scheme is an order of magnitude more energy efficient than a simple direct transmission scheme.

Power-aware sensor node model

A power-aware sensor node model essentially describes the power consumption in different levels of node sleep state. Every component in the node can have different power modes. The StrongARM can be in active, idle, or sleep mode; the radio can be in transmit, receive, standby, or off mode. Each node sleep state corresponds to a particular combination of component power modes. In general, if there are N components labeled (1, 2, …, N) each with k_i sleep states, the total number of node sleep states is $\prod k_i$. Every component power mode has a latency overhead associated with transitioning to that mode. Therefore each node sleep mode is characterized by power consumption and latency overhead. However, from a practical point of view not all sleep states are useful.

Table 1 enumerates the component power modes corresponding to five different useful sleep states for the sensor node. Each of these node sleep modes corresponds to an increasingly deeper sleep state and is therefore characterized by an increasing latency and decreasing power consumption.

These sleep states are chosen based on actual working conditions of the sensor node; for example, it does not make sense to have memory active and everything else completely off. The design problem is to formulate a policy for transitioning between states based on observed events so as to maximize energy efficiency.

The power-aware sensor model is similar to the system power model in the Advanced Configuration and Power Interface (ACPI) standard.[7] An ACPI-compliant system has five global states. SystemStateS0 (corresponding to the working state), and SystemStateS1 to SystemStateS4 (corresponding to four different sleep-state levels). The sleep states are differentiated by power consumed, the overhead required in going to sleep and the wake-up time. In general, a deeper sleep state consumes less power and has a longer wake-up time. Another similar aspect is that in ACPI the power manager is an OS module.

Event generation model

An event occurs when a sensor node picks up a signal with power above a predetermined threshold. For analytical tractability, we assume that every node has a uniform radius of visibility, r. In real applications, the terrain might influence the visible radius. An event can be static (such as a localized change in temperature/pressure in an environment monitoring application) or can propagate (such as signals generated by a moving object in a tracking application).

In general, events have a characterizable (possibly nonstationary) distribution in space and time. We will assume that the temporal

event behavior over the entire sensing region, R, is a Poisson process with an average event rate given by λ_{tot}. In addition, we assume that the spatial distribution of events is characterized by an independent probability distribution given by $p_{XY}(x,y)$. Let p_{ek} denote the probability that an event is detected by node$_k$, given the fact that it occurred in R.

$$p_{ek} = \frac{\int_{C_k} p_{XY}(x,y)\,dxdy}{\int_{R} P_{XY}(x,y)\,dxdy} \quad (1)$$

Let $p_k(t,n)$ denote the probability that n events occur in time t at node$_k$. Therefore, the probability of no events occurring in C_k over threshold interval T_{th} is given by

$$P_k(T_{th},0) = \sum_{i=0}^{\infty} \frac{e^{-\lambda_{t0t}T_{th}}(\lambda_{t0t}T_{th})^i}{i!}(1-P_{ek})^i = e^{-P_{ek}\lambda_{t0t}T_{th}} \quad (2)$$

Let $P_{th,k}(t)$ be the probability that at least one event occurs in time t at node$_k$.

$$P_{th,k}(T_{th}) = 1 - P_k(T_{th},0) = 1 - e^{-P_{ek}\lambda_{t0t}T_{th}} \quad (3)$$

That is, the probability of at least one event occurring is an exponential distribution characterized by a spatially weighted event arrival rate $\lambda_k = \lambda_{tot} \times p_{ek}$.

In addition, to capture the possibility that an event might propagate in space, we describe each event by position vector $\bar{p} = \bar{p}_0 + \int \bar{v}(t)dt$. In this equation, $\bar{p}_0$ is the coordinates of the event's point of origin and $\bar{v}(t)$ characterizes the event's propagation velocity. The point of origin has a spatial and temporal distribution described by Equations 1, 2, and 3. We have analyzed three distinct classes of events:

- $\bar{v}(t)$=0, the events occur as stationary points;
- $\bar{v}(t)$ = constant, the event propagates with fixed velocity (such as a moving vehicle); and
- $|\bar{v}(t)|$ = constant, the event propagates with fixed speed but random direction (such as a random walk).

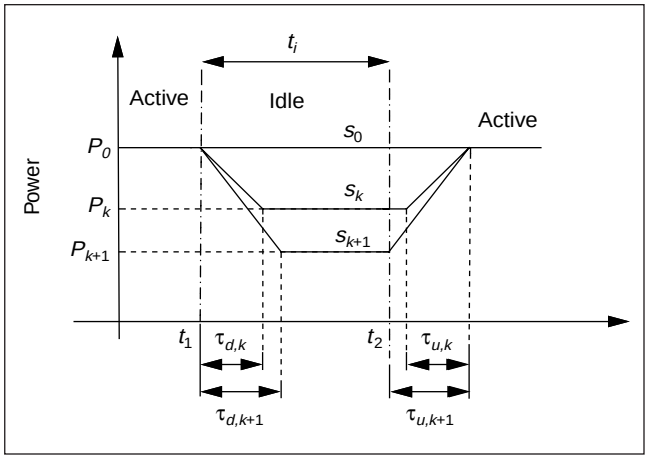

Figure 2. State transition latency and power.

Sleep-state transition policy

Assume an event is detected by node$_k$ at some time. The node finishes processing the event at t_1 and the next event occurs at time $t_2 = t_1 + t_i$. At time t_1, node$_k$ decides to transition to sleep state s_k from the active state s_0, as shown in Figure 2. Each state s_k has power consumption P_k, and the transition times to it from the active state and back are given by $\tau_{d,k}$ and $\tau_{u,k}$. By our definition of node sleep states, $P_j > P_i$, $\tau_{d,i} > \tau_{d,j}$, and $\tau_{u,i} > \tau_{u,j}$ for any $i > j$.

We now derive a set of sleep time thresholds $\{T_{th,k}\}$ corresponding to states $\{s_k\}$, $0 \le k \le N$, for N sleep states. Transitioning to sleep state s_k from state s_0 will result in a net energy loss if idle time $t_i < T_{th,k}$ because of the transition energy overhead. This assumes that no productive work can be done in the transition period, which is invariably true. For example, when a processor wakes up, it spends the transition time waiting for the phase-locked loops to lock, the clock to stabilize, and the processor context to be restored. The energy saving from a state transition to a sleep state is given by

$$\begin{aligned} E_{save,k} &= P_0 t_i - \left(\frac{P_0 + P_k}{2}\right)(\tau_{d,k} + \tau_{u,k}) \\ &\quad - P_k(t_i - \tau_{d,k}) \\ &= (P_0 - P_k)t_i - \left(\frac{P_0 - P_k}{2}\right)\tau_{d,k} - \\ &\quad \left(\frac{P_0 - P_k}{2}\right)\tau_{u,k} \end{aligned} \quad (4)$$

Such a transition is only justified when $E_{save,k} >$

Table 2. Sleep state power, latency, and threshold.

State	Pk (mW)	tk (ms)	Tth,k
s_0	1,040	Not applicable	Not applicable
s_1	400	5	8
s_2	270	15	20
s_3	200	20	25
s_4	10	50	50

0. This leads us to the threshold

$$T_{th,k} = \frac{1}{2}\left[\tau_{d,k} + \left(\frac{P_0 + P_k}{P_0 - P_k}\right)\tau_{u,k}\right] \quad (5)$$

This equation implies that the longer the delay overhead of the transition $s_0 \rightarrow s_k$, the higher the energy-gain threshold; and the more the difference between P_0 and P_k, the smaller the threshold. These observations are intuitively appealing, too.

Table 2 lists the power consumption of the sensor node described in Figure 1 in its different power modes. Since the node consists of off-the-shelf components, it's not optimized for power consumption. However, we will use the threshold and power consumption numbers detailed in Table 2 to illustrate the basic idea. The steady state shutdown algorithm is

```
If (eventOccurred() = true) {
        processEvent();
        ++eventCount;
        lambda_k =
eventCount/getTimeElapsed();
        for( k = 4; k>0; k— )
                if( computePth( Tth(k) ) <
    pth0 )
                        sleepState(k);
}
```

When node$_k$ detects an event, it awakes and processes the event (this might involve classification, beam forming, transmission, and so forth). It then updates a global (eventCount) counter that stores the total number of events registered by node$_k$. Average arrival rate λ_k for node$_k$ is then updated. This requires use of a μOS-timer-based system function call, getTimeElapsed(), which returns the time elapsed since the node was turned on. The μOS then tries to put the node into sleep state s_k (starting from deepest state s_4 through s_1) by testing the probability of an event occurring in corresponding sleep time threshold $T_{th,k}$ against system defined constant $p_{th,0}$.

Missed events

All the sleep states except state s_4 have the actual sensor and analog-digital conversion circuit on. Therefore, if an event is detected (that is the signal power is above a threshold level) the node transitions to state s_0 and processes the event. The only overhead involved is latency (worst-case being about 25 ms). However, in state s_4, the node is almost completely off and it must decide on its own when to wake up. In sparse-event sensing systems (for example vehicle tracking, seismic detection, and so forth) the interarrival time for events is much greater than sleep time thresholds $T_{th,k}$. Therefore, the sensor node will invariably enter the deepest sleep state, s_4.

The processor must watch for preprogrammed wake-up signals. The CPU programs these signal conditions prior to entering the sleep state. To wake up on its own, the node must be able to predict the next event's arrival. An optimistic prediction might result in the node waking up unnecessarily; a pessimistic strategy will result in some events being missed.

Being in state s_4 results in missed events, as the node isn't alerted. What strategy is used is a design concern based on the criticalness of the sensing task. We discuss two possible approaches:

- Completely disallow s_4. If the sensing task is critical and events cannot be missed this state must be disabled.
- Selectively disallow s_4. This technique can be used if events are spatially distributed and not all critical. Both random and deterministic approaches can be used. In the clustering protocol, the cluster heads can have a disallowed s_4 state while normal nodes can transition to s_4. Alternatively, the scheme that we propose is more homogeneous. Every node$_k$ that satisfies the sleep threshold condition for s_4 enters sleep with

a system-defined probability p_{s4} for a time duration given by

$$t_{s4,k} = -\frac{1}{\lambda_k} ln(p_{s4}) \quad (6)$$

Equation 6 describes the steady-state node behavior. The sleep time is computed so the probability that no events occur in $t_{s4,k}$ that is $p_k(t_{s4,k},0) = p_{s4}$. However, when the sensor network is switched on and no events occur for a while, λ_k is zero. To account for this, we disallow transition to state s_4 until at least one event is detected. We can also have an adaptive transition probability, p_{s4}, which is zero initially and increases as events are detected. The probabilistic state transition is described in Figure 3.

The advantage of the algorithm is that efficient energy trade-offs can be made with event detection probability. By increasing p_{s4}, the system energy consumption can be reduced while the probability of missed events will increase and vice versa. Therefore, our overall shutdown policy is governed by two implementation-specific probability parameters, $p_{th,0}$ and p_{s4}.

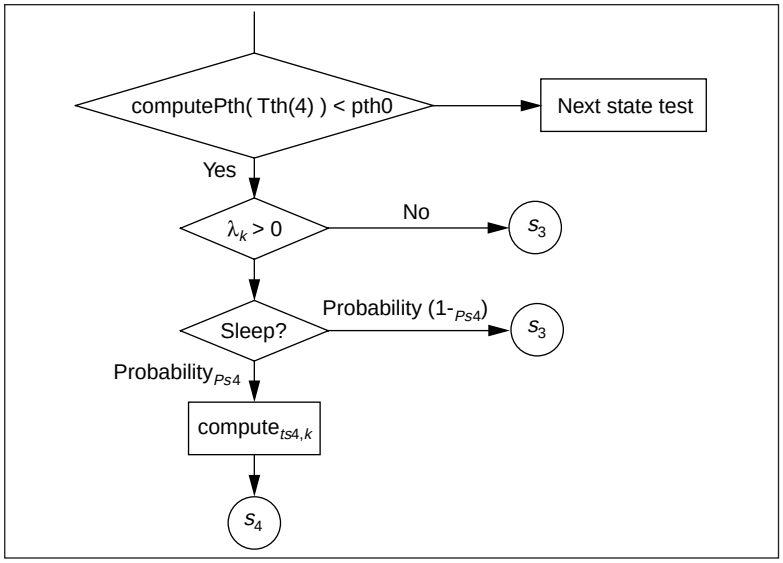

Figure 3. Transition algorithm to almost-off s_4 state.

Results

We have simulated a $\eta = 1{,}000$ node system distributed uniformly and randomly over a 100-m × 100-m area. The visibility radius of each sensor was assumed to be $\rho = 10$ m. The sleep state thresholds and power consumption are shown in Table 2. Figure 4 shows the overall spatial node energy consumption over for an event with a Gaussian spatial distribution centered around (25, 75). The interarrival process follows Poisson distribution with λ_{tot} equal 500 per second. It can be seen that node energy consumption tracks event probability. In the scenario without power management, there is uniform energy consumption at all the nodes.

One drawback to the whole scheme is that is has a finite and small window of interarrival rates λ_{tot} over which the fine-grained sleep states can be used. In general, the more differentiated the power states (that is, the greater the difference in their energy and latency overheads) the wider the interarrival time range in which all sleep states can be used.

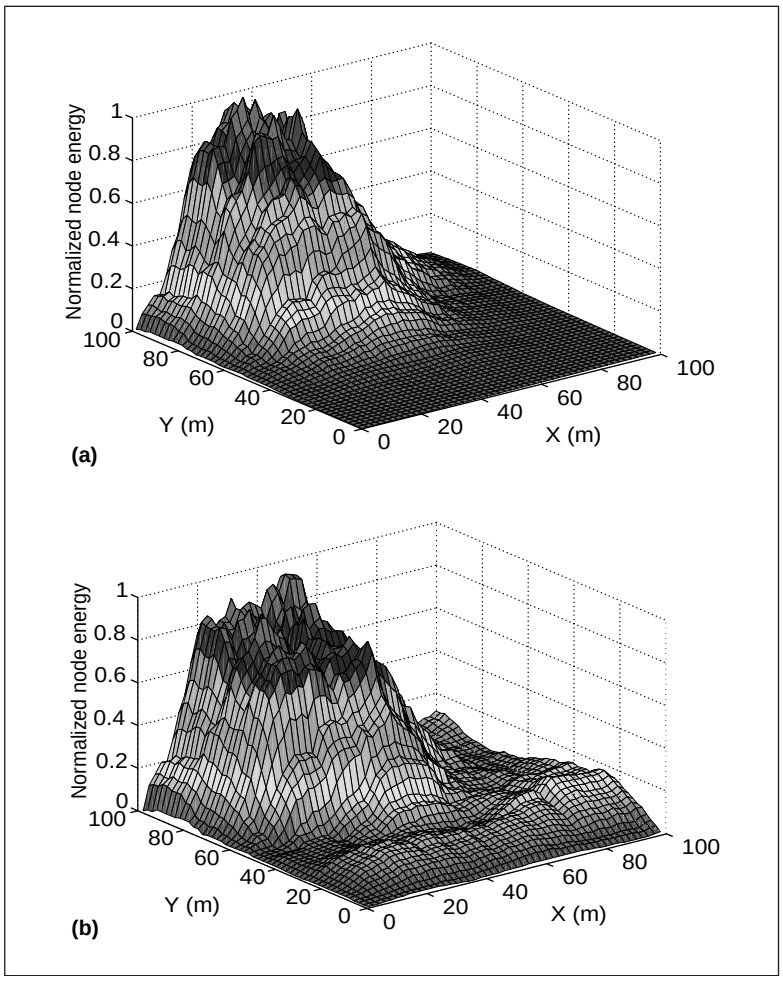

Figure 4. Simulation of a 1,000-node system: (a) Spatial distribution of events (Gaussian) and (b) spatial energy consumption in the sensor nodes.

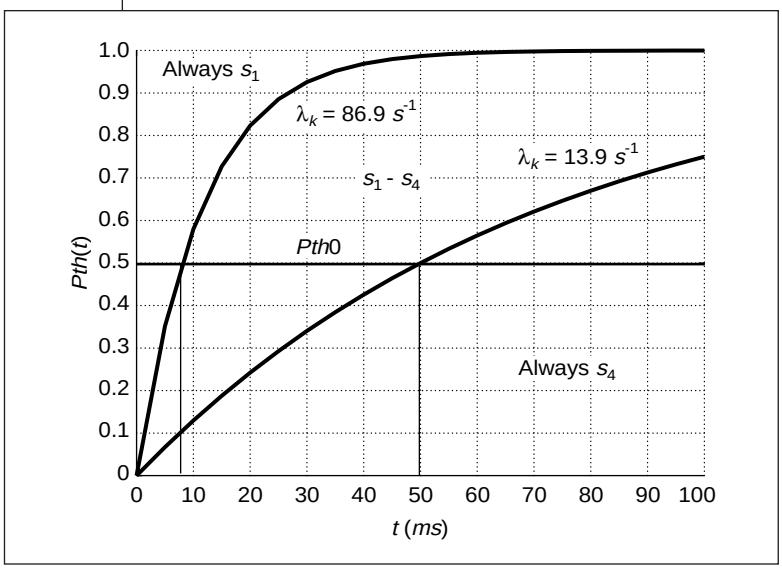

Figure 5. Event arrival rates at a node.

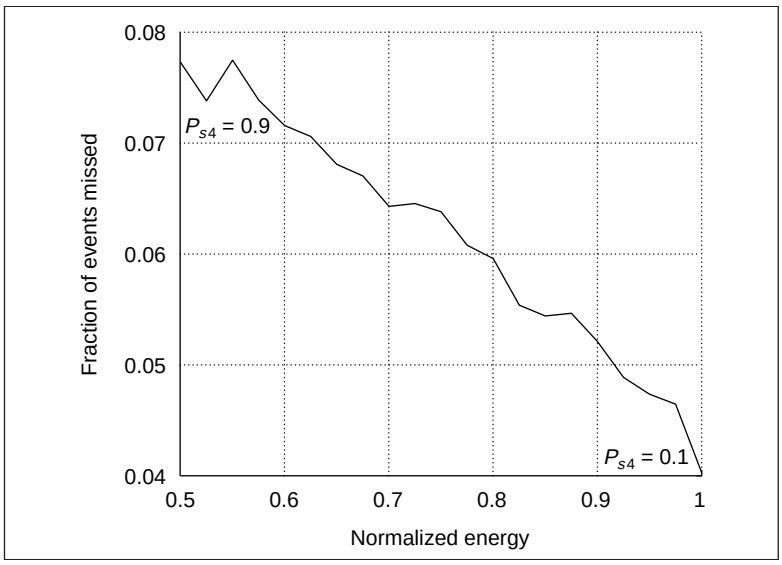

Figure 6. Fraction of events missed compared to energy consumption.

Figure 5 shows the range of event arrival rates at a node (λ_k) over which the states s_1 to s_3 are used significantly. If $\lambda_k < 13.9$ s^{v1}, transition to state s_4 is always possible. (That is, the threshold condition is met. Actual transition, of course, occurs with probability p_{s4}.) Similarly, if $\lambda_k > 86.9$ s^{-1}, the node must always be in the most active state. These limits have been computed using nominal $p_{th,0} = 0.5$. Using a higher value of $p_{th,0}$ would result in frequent transitions to the sleep states. If events occur fast enough, this would result in increased energy dissipation associated with wake-up energy cost. A smaller value of $p_{th,0}$ would result in a pessimistic scheme for sleep-state transition and therefore lesser energy savings.

Figure 6 illustrates the energy-quality trade-off of our shutdown algorithm. Increasing the probability of transition to state s_4 (that is, increasing p_{s4}), saves energy at the cost of the increased possibility of missing an event. Such a graceful degradation of quality with energy is highly desirable in energy-constrained systems.

Variable-voltage processing

Different sensing applications will have different processing requirements in the active state. Having a processor with a fixed throughput (equal to the worst-case workload) is necessarily power inefficient. Having a custom digital signal processor for every sensing application is not feasible both in terms of cost and time overhead. However, energy savings can still be obtained by tuning the processor to deliver just the required throughput.

Let's consider a case where a fixed task has to be done by a processor every T_0 time units. If the processor can accomplish the task in $T < T_0$ time units, it will basically be idling for the remaining $T_0 - T$ time units.

However, if we reduce the operation frequency so the computation can be stretched over entire time frame T_0, we can get linear energy savings. Additional quadratic energy savings can be obtained if we reduce the power supply voltage to the minimum required for that particular frequency. First-order CMOS (complimentary metal-oxide semiconductor) delay models show that gate delays increase with decreasing supply voltage, while switching energy decreases quadratically.

$$Delay \propto \frac{V_{dd}}{\left(V_{dd} - V_t\right)^2}$$
$$Energy \propto CV^2{}_{dd} + V_{dd} I_{leak} \Delta t \qquad (7)$$

In these equations, V_{DD} is the supply voltage, and V_t is the gate threshold voltage.

The time-energy trade-off involved in this technique is best illustrated by a simple example. Suppose a particular task has 75% proces-

sor utilization when the processor runs at 200 MHz and 1.5 V. By reducing clock frequency to 150 MHz and voltage to 1.2 V (the minimum required for that frequency), the program's energy consumption decreases by approximately 52% without any performance degradation.

Energy workload model

Using simple first-order CMOS delay models, it has been shown that the energy consumption per sample is

$$E(r) = CV_0 2T_s f_{ref} r\left[\frac{V_t}{V_0} + \frac{r}{2} + \sqrt{r\frac{V_t}{V_0} + \left(\frac{r}{2}\right)^2}\right]^2 \quad (8)$$

where C is the average switched capacitance per cycle; T_s is the sample period; f_{ref} is the operating frequency at V_{ref}; r is the normalized processing rate, that is, $r = f / f_{ref}$; and $V_0 = (V_{ref} - V_t)^2/V_{ref}$ with V_t being the threshold voltage.[5] The normalized workload in a system is equivalent to the processor utilization.

The OS scheduler allocates a time slice and resources to various processes based on their priorities and state. Often, no process is ready to run, and the processor simply idles. Normalized workload w over an interval is simply the ratio of the non-idle cycles to the total cycles, that is w = (total_cycles – idle_cycles) / total_cycles. The workload is always in reference to the fixed maximum supply and maximum processing rate.

In an ideal DVS system, the processing rate is matched to the workload so there are no idle cycles, and utilization is maximized. Figure 7 shows the plot of normalized energy compared with workload (as described by Equation 8) for an ideal DVS system. The graph's important conclusions are that averaging the workload and processing at the mean workload is more energy efficient because of the convexity of the $E(r)$ graph and Jensen's inequality: $\overline{E(r)} \geq E(\overline{r})$.

System model

Figure 8 shows a generic block diagram of the variable voltage processing system. The task queue models the various events sources for the processor. Each of the n sources produces events at an average rate of λ_k, $(k = 1, 2, \ldots, n)$. An OS scheduler manages all these tasks and decides which process will run. The average rate at which events arrive at the processor is $\lambda = \sum\lambda_k$.

The processor in turn offers a time-varying processing rate $\mu(r)$. The OS kernel measures the idle cycles and computes normalized workload w over some observation frame. The workload monitor sets processing rate r based on current workload w and a history or workloads from previous observation frames. This rate r in turn decides the operating voltage $V(r)$ and operating frequency $f(r)$, which are set for the next observation slot. The problems addressed are twofold: What kind of future workload prediction strategy should be used? What is the duration of the observation slot—that is how frequently should the processing rate be updated? The overall objective of a DVS system is to

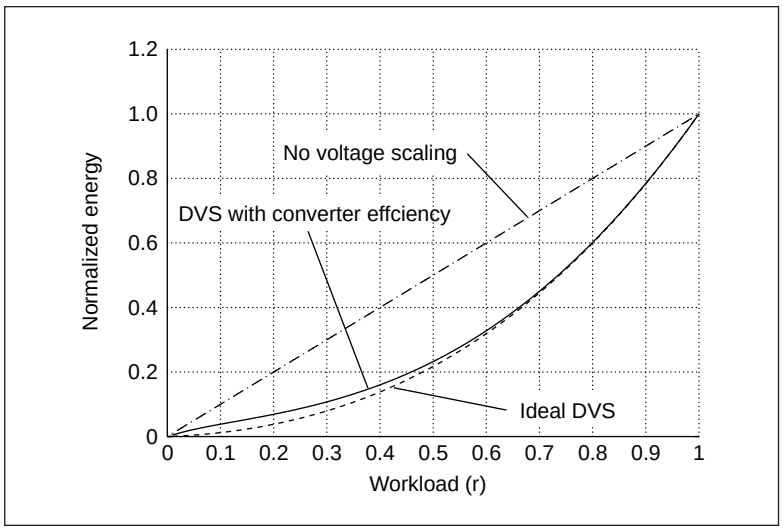

Figure 7. Energy consumption compared with workload.

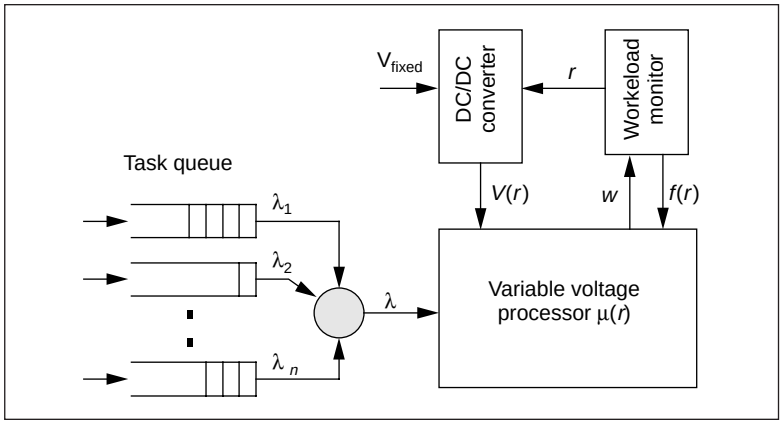

Figure 8. Block diagram of a DVS processor system

minimize energy consumption under a given performance requirement constraint.

Prediction algorithm

Let the observation period be T. Let $w(n)$ denote the average normalized workload in the interval $(n-1)T \le t \le nT$. At time $t = nT$, we must decide what processing rate to set for the next slot, that is $r(n+1)$, based on the workload profile history. Our workload prediction for the $(n+1)$th interval is

$$w_p[n+1] = \sum_{k=0}^{N-1} h_n[k]w[n-k] \quad (9)$$

where $h_n(k)$ is an N-tap, adaptable finite-length impulse response filter. This FIR filter's coefficients are updated in every observation interval based on the error between the processing rate (which is set using the workload prediction) and the workload's actual value.

Most processor systems will have a discrete set of operating frequencies, which implies that the processing rate levels are quantized. The StrongARM SA-1100 microprocessor, for instance, can run at 11 discrete frequencies in the range of 59 to 206 MHz.[8] Discretization of the processing rate does not significantly degrade the energy savings from DVS.

Let us assume that there are L discrete processing levels available so

$$r \in R_L, R_L = (1/L, 2/L, ..., 1) \quad (10)$$

where we assume uniform quantization interval $\Delta = 1/L$. We also assume that the minimum processing rate is $1/L$ since $r = 0$ corresponds to the complete off state. Based on workload prediction $w_p(n+1)$, processing rate $r(n+1)$ is set

$$r(n+1) = \lceil w \times (n+1)/\Delta \rceil \times \Delta \quad (11)$$

is the processing rate set to a level just above the predicted workload.

Filter type

We have explored four types of filters. We present the basic motivation behind each filter and prediction performance of each filter.

Moving average workload (MAW). The simplest filter is a time-invariant moving average filter, $h_n(k) = 1/N$ for all n and k. This filter predicts the workload in the next slot as the average of the workload in the previous N slots. The basic motivation is if the workload is truly an Nth-order Markov process, averaging will result in workload noise being removed by low-pass filtering. However, this scheme might be too simplistic and may not work with time-varying workload statistics. Also, averaging results in high-frequency workload changes are removed and as a result instantaneous performance hits are high.

Exponential weighted averaging (EWA). This filter is based on the idea that the effect of a workload k slots before the current slot lessens as k increases. That is, this filter gives maximum weight to the previous slot, lesser weight to the one before, and so on. The filter coefficients are $h_n(k) = a^{-k}$, for all n, with a chosen so $\sum h_n(k) = 1$ and is positive. The idea of exponential weighted averaging has been used in the prediction of idle times for DPM using shutdown techniques in event-driven computation. There, too, the idea is to assign progressively decreasing importance to historical data.

Least mean square (LMS). It makes more sense to have an adaptive filter whose coefficients are modified based on the prediction error. Two popular adaptive filtering algorithms are the LMS and the recursive-least-squares (RLS) algorithms.[9] The LMS adaptive filter is based on a stochastic gradient algorithm.

Let the prediction error be $w_e(n) = w(n) - w_p(n)$, where $w_e(n)$ denotes the error, and $w(n)$ denotes the actual workload as opposed to predicted workload $w_p(n)$ from the previous slot. The filter coefficients are updated according to the following rule

$$h_{n+1}(k) = h_n(k) + \mu w_e(n)\, w(n-k) \quad (12)$$

where μ is the step size.

Use of adaptive filters has its advantages and disadvantages. On the one hand, since they are self-designing, we do not have to worry about individual traces. The filters can learn from the workload history. The obvious problems involve convergence and stability. Choosing

the wrong number of coefficients or an inappropriate step size can have very undesirable consequences. RLS adaptive filters differ from LMS adaptive filters in that they do not employ gradient descent. Instead, they employ a clever result from linear algebra. In practice they tend to converge much faster but they have higher computational complexity.

Expected workload state (EWS). The last technique is based on a pure probabilistic formulation and does not involve any filtering. Let the workload be discrete and quantized like the processing rate, as shown in Equation 10, with state 0 also included. The error can be made arbitrarily small by increasing the number of levels, L. Let $P = [p_{ij}]$, $0 \le i \le L$ and $0 \le j \le L$, denote a square matrix with elements p_{ij} such that $p_{ij} = Probability\{w(r+1) = w_j \mid w(r) = w_i\}$, where w_k represents the kth workload level out of $L + 1$ discrete levels. P, therefore, is the state transition matrix with the property that $\Sigma_j P_{ij} = 1$. The workload is then predicted as

$$w[n+1] = E\{w[n+1]\} = \sum_{j=0}^{L} w_j P_{ij} \qquad (13)$$

where $w(n) = w_i$ and $E[w(n+1)]$ denotes the expected value. The probability matrix is updated in every slot by incorporating the actual state transition. In general the $(r+1)$th state can depend on the previous N states (as in a Nth order Markov process) and the probabilistic formulation is more elaborate.

Figure 9 shows the prediction performance in terms of root-mean-square error for the four different schemes. If the number of taps is small, the prediction is too noisy. With too many taps, there is excessive low-pass filtering. Both situations result in poor prediction. In general, we found that the LMS adaptive filter outperforms other techniques and produces the best results with three taps. Figure 10 shows the adaptive prediction of the filter for a workload snapshot.

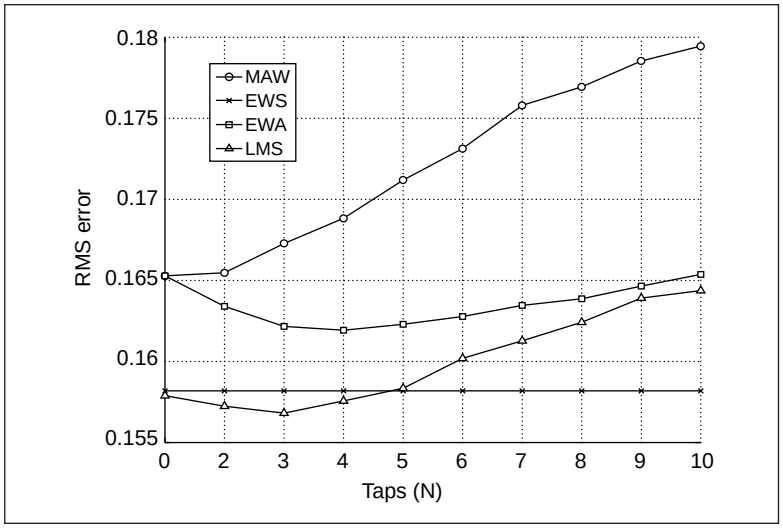

Figure 9. Prediction performance of the different filters.

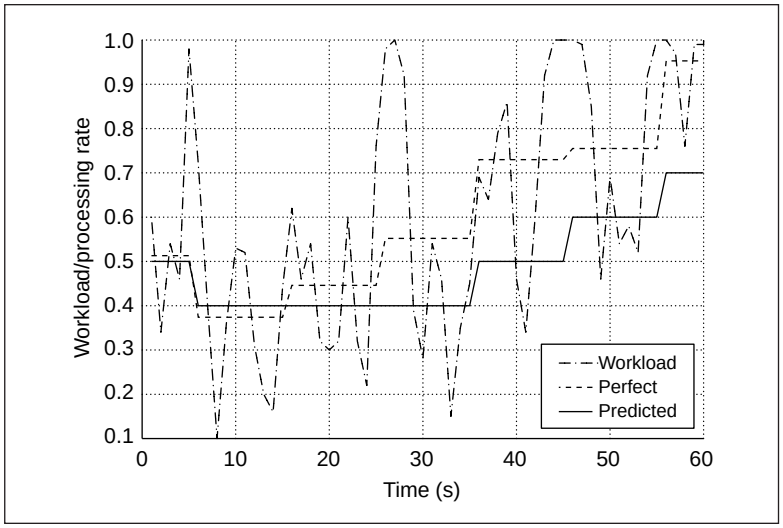

Figure 10. Workload tracking by the LMS filter.

Performance hit function

Performance hit $\phi(\Delta t)$ over time frame Δt is defined as the extra time (expressed as a fraction of Δt) required to process the workload over time Δt at the processing rate available in that time frame.

Let $\bar{w}_{\Delta t}$ and $\bar{r}_{\Delta t}$ denote the average workload and processing rates over the time frame of interest Δt. The extra number of cycles required (assuming $w_{\Delta t} > r_{\Delta t}$) to process the entire workload is $(\bar{w}_{\Delta t} f_{max} \Delta t - r_{\Delta t} f_{max} \Delta t)$ where f_{max} is the maximum operating frequency. Therefore the extra amount of time required is simply $(w_{\Delta t} f_{max} \Delta t - \bar{r}_{\Delta t} f_{max} \Delta t) / \bar{r}_{\Delta t} f_{max}$. Therefore,

$$\phi(\Delta t) = \frac{\left(\bar{w}_{\Delta t} - \bar{r}_{\Delta t}\right)}{\bar{r}_{\Delta t}} \qquad (14)$$

If $\bar{w}_{\Delta t} < \bar{r}_{\Delta t}$, the performance penalty is negative. The way to interpret this is that it is a slack

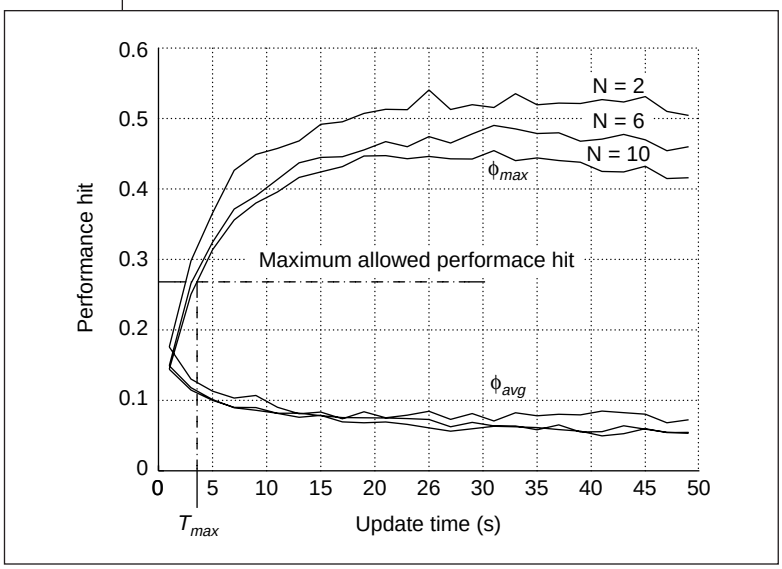

Figure 11. Average and maximum performance hits.

or idle time. Using this basic definition of performance penalty we define two different metrics: $\phi_{\max}{}^{T}(\Delta t)$ and $\phi_{\text{avg}}{}^{T}(\Delta t)$, the maximum and average performance hits measured over Δt time slots spread over observation period T.

Figure 11 shows the average and maximum performance hit as a function of update time T for a moving average prediction using two, six, and 10 taps. The time slots used were $\Delta t = 1$ s, and the workload trace was that of the dial-up server. The results have been averaged over one hour. While the maximum performance hit increases as T increases, the average performance hit decreases. This is because as T increases the excess cycles from one time slot spills over to the next one. If the slot has a negative performance penalty (that is slack/idle cycles), then the average performance hit over the two slots decreases and so on. On the other hand, as T increases, the chances of an increased disparity between the workload and processing rate in a time slot is more and the maximum performance hit increases.

This leads to a fundamental energy-performance trade-off in DVS. Because of the convexity of the $E(r)$ relationship and Jensen's inequality, we would always like to work at the overall average workload. Therefore, over one-hour, for example, the most energy efficient DVS solution is one where we set the processing rate equal to the overall average workload. In other words, increasing T leads to increased energy efficiency.

On the other hand, increasing T also increases the maximum performance hit. In other words, the system might be sluggish in moments of high workload. Maximum energy savings for a given performance hit involves choosing maximum update time T so the maximum performance hit is within bounds, as shown in Figure 11.

Optimizing update time and taps

The conclusion that increasing update time T results in the most energy savings is not completely true. This would be the case with a perfect prediction strategy. In reality, if the update time is large, the cost of an overestimated rate is more substantial and the energy savings decrease. Since we are using discrete processing rates (in all our simulations the number of processing rate levels is set to 10 unless otherwise stated), and we round off the rate to the next higher quantity, using larger update times results in higher overestimation cost.

A similar argument holds for number of taps N. A very small N implies that the workload prediction is very noisy, and the energy cost high because of widely fluctuating processing rates. A very large N, on the other hand, implies that the prediction is heavily low-pass filtered and therefore sluggish in responding to rapid workload changes. This leads to a higher performance penalty. Figure 12 shows the relative energy plot (normalized to the no-DVS case) for the dial-up server trace. The period of observation was one hour. The energy savings showed a 13% variation based on which N and T were chosen. Again, the filter was the average moving type.

Results

Table 3 summarizes our key results. We have used one-hour workload traces from three different processors over different times of day. The energy savings ratio (ESR) is defined as the ratio of the energy consumption with no DVS to the energy consumption with DVS. Maximum savings occur when we set the processing rate equal to the average workload over the entire period. This is shown in the maximum ESR column and we can see that energy savings from a factor of two to a few 100 s is possi-

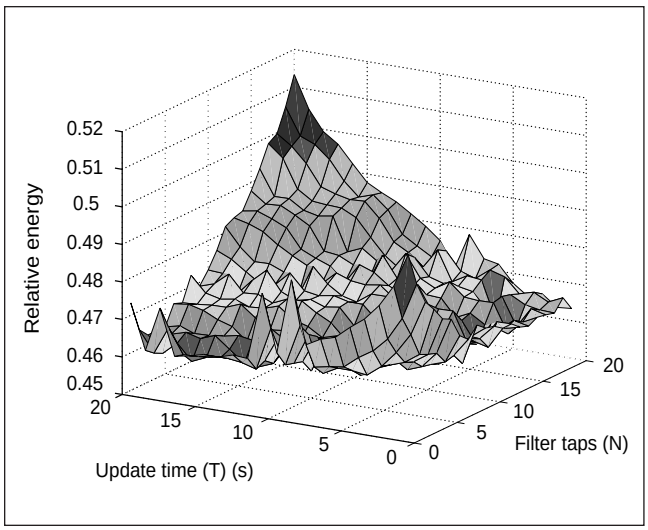

Figure 12. Average and maximum performance hits.

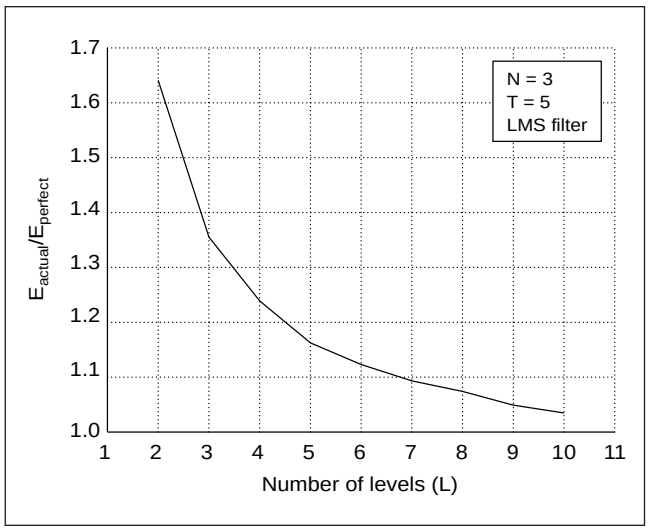

Figure 13. Effect of number of discrete processing levels, *L*.

ble depending on workload statistics. Maximum savings is not possible for two reasons: The maximum performance hit increases as the average duration is increased, and it is impossible to know the average workload over the stipulated period beforehand. The filters have $N = 3$ taps and update time $T = 5$ s, based on our previous discussion and experiments.

The perfect column shows the ESR for the case where we had a perfect predictor for the next observation slot. ESR maximum / ESR perfect reflects the factor by which energy savings is reduced because of update every T seconds.

The actual column shows the ESR obtained by the various filters. In almost all our experiments the LMS filter gave the best energy savings. The last two columns are the average and maximum performance hits. The average performance hit is around 10% while the maximum performance hit is about 40%.

Finally, the effect of processing-level quantization is shown in Figure 13. As the number of

Table 3. DVS energy savings ratio ($E_{no\ DVS}/E_{DVS}$), for $N = 3$ and $T = 5$ s.

		Energy savings ratio (ESR)			ESR comparison		ϕ_{avg}	ϕ_{max}
Trace	**Filter**	**Maximum**	**Perfect**	**Actual**	**Maximum/perfect**	**Perfect/actual**	**(%)**	**(%)**
Dial-up								
server	MAW	2.9	2.4	2.2	1.2	1.10	10.6	34.8
	EWS			2.1		1.11	10.8	36.3
	EWA			2.2		1.09	10.6	35.4
	LMS			2.3		1.03	14.7	43.1
File								
server	MAW	76.7	23.5	16.7	3.3	1.41	12.6	42.8
	EWS			15.7		1.50	7.4	33.8
	EWA			16.7		1.41	9.2	37.4
	LMS			19.6		1.20	14.1	47.7
User								
workstation	MAW	445.9	275.2	52.7	1.6	5.22	3.6	35.3
	EWS			59.5		4.63	3.8	35.1
	EWA			52.1		5.28	3.7	35.6
	LMS			53.0		5.19	3.9	36.0

discrete levels, L, increases, the ESR gets closer to the perfect-prediction case. For $L = 10$ (as available in the StrongARM SA-1100) the ESR degradation due to quantization noise is less than 10%.

AT PRESENT version II of the μAMPS sensor node has been implemented. We have ported a real-time operating system, eCOS, to run on the StrongARM processor. The OS supports dynamic voltage and frequency scaling and we are working on drivers that will allow hierarchical shutdown. The node has a very compact form factor.

Subsequent versions of the μAMPS sensor node will use a system-on-ship approach with an RF front end also built on the chip. Based on experiments with version II, we will add or remove functionality as needed. The StrongARM processor will be replaced with a dedicated DSP type architecture tuned for programmable sensing applications. The power consumption of such a node will be at least 3-4 orders of magnitude lower than the current version and will have the capability to run from ambient energy harvested from the sensing environment itself. ■

Acknowledgment

This research is sponsored by the Defense Advanced Research Project Agency (DARPA) Power-Aware Computing/Communication Program and the Air Force Research Laboratory, Air Force Material Command, under agreement number F30602-00-2-0551.

■ References

1. A.P. Chandrakasan et al., "Design Considerations for Distributed Microsensor Systems," *Proc. Custom Integrated Circuits Conf.*, IEEE, Piscataway, NJ, 1999, pp. 279-286.
2. "The MIT μAMPS Project," http://www-mtl.mit.edu/research/icsystems/uamps/
3. L. Benini and G.D. Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*, Kluwer Academic Pub., NY, NY, 1997.
4. A. Sinha, A. Wang, and A.P. Chandrakasan, "Algorithmic Transforms for Efficient Energy Scalable Computation," *Proc. Int'l Symp. on Low Power Electronics and Design*, 2000, pp. 31-36.
5. V. Gutnik and A.P. Chandrakasan, "An Embedded Power Supply for Low-Power DSP," *IEEE Trans. VLSI Systems*, vol. 5, no. 4, Dec. 1997, pp. 425-435.
6. T. Pering, T. Burd, and R. Broderson, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Proc. Int'l Symp. on Low Power Electronics and Design*, 1998, pp. 76-81.
7. Advanced Configuration and Power Interface, http://www.teleport.com/acpi.
8. Intel StrongARM Processors. http://developer.intel.com/design/strong/sa1100.htm.
9. P.S.R. Diniz, *Adaptive Filtering Algorithms and Practical Implementation*, Kluwer Academic,1997.

Amit Sinha is a PhD candidate in electrical engineering and computer science at the Massachusetts Institute of Technology. His research interests include low-power systems and software. Sinha has a BTech in electrical engineering from the Indian Institute of Technology, Delhi.

Anantha Chandrakasan is an associate professor of electrical engineering and computer science at the Massachusetts Institute of Technology. His research interests include the energy-efficient implementation of digital signal processors and wireless microsensor networks. Chandrakasan received a PhD in electrical engineering and computer science from the University of California at Berkeley. He is an elected member of the Solid-State Circuits Society AdCom.

■ Direct questions and comments about this article to Amit Sinha, Department of EECS, Room 38-107, Massachusetts Institute of Technology, Cambridge, MA, 02139, +1-617-253-0164, sinha@mit.edu.