Proactive vs. Reactive Idle Power Control

K. Kant, Intel Research
J. Alexander, DEG
{krishna.kant | jimbo.alexander}@intel.com

ABSTRACT

In this paper we study power control algorithms that attempt to reduce power consumption of a resource by taking advantage of available low power states. The low power exit for such algorithms could be either "reactive" (i.e., triggered by traffic arrival) or "proactive" (based on prediction of traffic arrival time). We compare the two approaches by considering a class of exponential smoothing based prediction algorithms that can be implemented effectively at high speeds. We show that proactive algorithms can provide some added benefits at moderate traffic loads. We also discuss detailed implementation of an algorithm that combines the two approaches and is appropriate for implementation in QuickPath, PCI-E, DMI and other platform components.

Keywords: Exponential smoothing, Idle power control, Performance modeling

1 Introduction

Recently, the focus in computer systems has shifted from purely performance to good performance at lowest possible power consumption. This has resulted in attempts to reduce both the idle and active power consumption of the platform. Idle power reduction techniques put idle resources into an appropriate low power state. The important attributes of low power states are power consumption, entry latency and exit latency. For example, most platform links support at least two low power states, called L0s and L1 respectively, in addition to the operating state L0. For L0s state, the power consumption is only 20-50% of L0 power and the entry/exit latencies are typically in 10's-100's of ns range. In contrast, L1 power consumption is generally very small but the exit latencies can range in multiple microseconds. The very high exit latency generally makes L1 unsuitable for use during the operational (i.e., package C0) state.

In this paper we address the question of whether *proactive* algorithms that attempt to predict low power duration and exit proactively can provide good power control, and study how they stack up against the simpler *reactive* algorithms whose exit is triggered by the traffic. Although the notion of a proactive and reactive algorithms is well known [1], we believe that this paper provides the first comprehensive study of relative merits of the two in a realistic setting. The paper also discusses the design of a simple but effective proactive algorithm that is suitable for HW implementation for high speed serial links (e.g., QuickPath, PCI-E, FBD, etc.) It is important to note in this regard that although the space of complex predictive algorithms (e.g., see [3]) is very large; most such algorithms will be too expensive to implement at the multi-Gb/sec speeds that we are interested in.

The focus of this paper is entirely on power control based on the behavior of the resource in question. It is certainly possible (and often beneficial) to power control one resource based on the state of another; however, we do not consider those aspects here.

The rest of the paper is organized as follows. Section 2 introduces essential characteristics of power control algorithms and performance metrics. Section 3 provides an overview of the algorithms that we considered. Section 4 discusses our evaluation methodology and shows detailed performance comparison of various algorithms. Section 5 considers HW implementation issues for our preferred proactive algorithm called S-ESA. Finally section 6 concludes the paper and discusses future work.

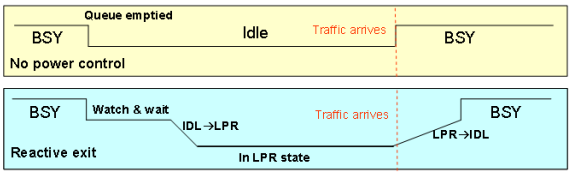

Fig 1: (a) No power control, (b) Reactive control

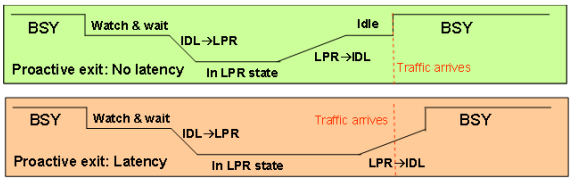

Fig 2: Illustration of Proactive Power Control

2 Power Control Basics

Although some resources provide multiple usable low power states (e.g., fast and slow CKE for memory ranks), we focus on a single low power state here. In this case, there are 3 states of a resource: busy (BSY), idle (IDL), and low power (LPR). The power states often use device specific names (e.g., L0, L0s, L1 for links); however, for generality we avoid such names.

The power control algorithm only has to make two crucial decisions: (a) when to go into LPR state, and (b) when to exit LPR state. These aspects along with appropriate metrics for evaluating a power control algorithm are discussed in the following.

2.1 Entering Low Power State

Because of the LPR entry/exit cost, an immediate jump to LPR when the resource goes idle is usually undesirable. Instead, the general technique is to monitor the idle period for a certain duration, henceforth called *runway*, and if the resource is still idle, start transition to LPR state. In the simplest case, the runway duration may be fixed based on power savings vs. tolerable latency tradeoff. In a more sophisticated algorithm, the runway may be estimated based on the durations of idle periods in recent past and perhaps other indicators of activity. In section 3 we shall discuss a few algorithms to estimate the runway.

2.2 Exiting Low Power State

As already discussed, an exit could be either *reactive* or *proactive*, as illustrated in Figs. 1 and 2. The reactive method is trivial to implement and maximizes the LPR residence time, thereby yielding the maximum power savings. Unfortunately, the request responsible for the resource wakeup will experience the full exit latency before it can be served. If the exit latency is large compared with the transaction service time (as is often the case with links), a reactive exit may be undesirable at moderate resource utilizations where power saving opportunity still exists but significant additional latency is not acceptable. However, at very low resource utilizations, the additional latency impact is not very important, and a reactive exit is optimal. Thus the proactive algorithms discussed here are intended primarily to allow us extend power control to higher resource utilization levels.

Ideally a proactive exit should start LPR→IDL transition just in time to avoid any additional delay to the next request and yet achieve good power savings. Thus a crucial aspect of proactive exit is an accurate prediction of the idle period D. The accuracy of the prediction is a function of two attributes: (a) sophistication of the prediction methodology (including the amount of historical information maintained and used), and (b) the inherent predictability of successive gaps. Given the requirement of very high speed operation, the prediction must necessarily be very simple. More important, however, is the fact that predictability is inherently constrained by the correlation between successive gap sizes. *If successive idle period durations are nearly independent, the past tells us nothing about the future and a sophisticated history based prediction methodology is worthless.*

2.3 Evaluation Metrics

In this section we briefly discuss how to judge the performance of various power control algorithms. The two fundamental metrics are *average efficiency* (F) and *average additional latency* (L_p) introduced by the power

control. Both of these quantities depend on the resource utilization $U_{\rm rsrc}$, and we shall often show F and L_p as functions of $U_{\rm rsrc}$ for emphasis. The efficiency F is defined as the fraction of idle period (or "gap") duration for which an algorithm is able to keep the resource in LPR state. Efficiency directly translates into the power savings. That is, $\mathcal{P}'_{\rm idle}$, the average power with power control, can be related to IDL and LPR mode power consumption (P_{IDL} & P_{LPR}) by the trivial equation:

$$\mathcal{P}'_{\rm idle} = (1 - F(U_{\rm rsrc}))P_{IDL} + F(U_{\rm rsrc})P_{LPR} \tag{1}$$

Adding active power to the idle power gives the total power, denoted by $PP'_{\rm tot}$. For links, the active power is almost the same as idle power and there is no distinction between $\mathcal{P}'_{\rm tot}$ and $\mathcal{P}'_{\rm idle}$. However, for memory, the difference can be very substantial.

Let us now consider the latency metric L_p. The last illustration in Fig. 2 shows the case where the proactive algorithm incurs some latency $\leq \eta_x$ because our predicted idle period is too long. It is also possible that the traffic arrives while the resource is in the process of entering the LPR state. Generally, there is no way to short-circuit the process, and the resource must be allowed to enter LPR state fully only to be forced to exit immediately. Therefore, the power control latency is upper bounded by $\eta = \eta_e + \eta_x$.

2.4 Quality Metric

For any well designed power control algorithm, a higher efficiency almost always comes at the cost of a higher latency. In view of this, it becomes difficult to compare algorithms on the basis of these metrics since we want higher efficiency *and* lower latency. To this end, we define a *quality metric* involving both efficiency and latency by considering the case of a CPU accessing memory via a link. Although the same analysis may not apply for PCI-E or other links, the methodology provides a much more meaningful metric than an arbitrarily chosen one (e.g., efficiency divided by latency).

We assume that the performance metric of interest is the workload throughput λ. This can be related to the memory access latency using standard performance projection equations involving CPI, MPI, blocking factor, and path length. The net result is as follows: Let L_m denote the base memory access latency and L_p the additional latency when power control is used for link and/or memory. Also, let λ and λ' denote, respectively, the throughputs w/o and with power control. Then

$$\lambda = \frac{C\,U_{cpu}}{1 + \beta L_m}, \qquad \lambda' = \frac{C\,U_{cpu}}{1 + \beta(L_m + L_p)}, \tag{2}$$

where C and β are appropriate constants. For realistic situations $\beta \ll 1$. In particular, for SPECweb2005 running on a link based platform, $\beta \approx 0.01$. Now, an obvious quality metric (denoted as QM_1) is performance per watt, i.e., $QM_1 = \lambda'/\mathcal{P}'_{tot}$. Assuming that the utilization of the resource of interest is proportional to CPU utilization, the main problem with QM_1 is its monotonic decrease with U_{cpu}, which makes algorithmic differences hard to see. QM_1 also strongly depends on baseline power consumption. To eliminate this, we define another metric, QM_2 as the ratio of performance per watt with and w/o the power control, i.e., $QM_2 = \frac{\lambda'/\mathcal{P}'_{\rm tot}}{\lambda/\mathcal{P}_{\rm tot}}$.

3 Power Control Algorithms

3.1 Reactive Algorithms

The most basic power control algorithm is one that uses reactive exit and a fixed runway R, and we refer to it as B-REA (for basic reactive algorithm). In order to provide a good control over latency for bursty traffic, the runway R needs to be chosen fairly large. We assume $R = 4\eta$, as it provides a good compromise between latency and efficiency. Such an algorithm (but with settable value of R) is currently commonly used for memory ranks and platform links.

A reactive algorithm can be made smarter by controlling its runway based on the current gap size. In particular, if the *current gap* is less than some minimum gap $G_{\rm min}$, the runway is increased by a small amount Δ so as to lessen the probability of going into LPR state. Otherwise, the runway is decreased by the same

amount Δ so long as it doesn't go below some limit $R_{\min}$. This adaptation effectively makes the entry to LPR state more aggressive as the resource utilization decreases. We call this as A-REA (adaptive reactive) algorithm. For the results shown here, we assume $G_{\min} = 4\eta$, $R_{\min} = \eta/2$, and $\Delta = \eta/8$.

3.2 Proactive Algorithms

All proactive algorithms considered here attempt to predict the low power duration by computing an exponentially smoothed estimate of the current gap size. That is, if G_n is the gap estimate at nth step, and g_n the current gap, we have:

$$G_n = (1 - \alpha)\, G_{n-1} + \alpha g_n \tag{3}$$

where $0 < \alpha < 1$ is the smoothing constant. Exponential smoothing was chosen because of its very low computational complexity. In particular, it does not require any array storage and if α is chosen as an inverse power of 2, the calculation in the above equation can be accomplished with only shifts, adds, and subtracts.

For a stationary stochastic process, the exponentially smoothed estimate quickly converges to the average value. This means that the estimate will frequently exceed the actual gap and thereby cost exit latency. To remedy this problem we instead make our gap estimate *biased* by using two different smoothing constants, say α and β where $\beta < \alpha$. Basically, β is used when the new gap is larger than the current estimate, and α is used otherwise. The net result is that the estimated gap rises sluggishly but decreases more aggressively.

The successive gap sizes for a resource often show a very nonuniform behavior with some very large gaps sprinkled in-between generally smaller gaps. To handle this, we propose a 2-phase operation. Phase-1 operates as described above: i.e, an initial runway followed by a *timed sleep* for a predicted duration D. If this phase ends w/o any traffic arrival, the algorithm enters phase-2, where it stays in IDL state for some time that we call "second runway". If no traffic arrives by the end of the second runway, the algorithm goes into LPR state for an indefinite period and will exit only reactively. We call the phase-2 LPR usage as *untimed sleep* or *hibernation*, in order to distinguish it from the *timed sleep*. If entry time to phase-2 LPR is controlled by a fixed timer, it is possible to reduce this algorithm to a purely reactive algorithm at low utilizations as discussed in section 5.

3.3 Specific Proactive Variants

The proactive variants considered here differ essentially on how the runways are controlled. Since all algorithms are based on exponential smoothing, we denote all of them as ESA (exponential smoothing algorithm). The three variants considered here are described below.

3.3.1 Simple ESA

The simplest version, henceforth called S-ESA, estimates the first runway $R^{(1)}$ as follows:

$$R_n^{(1)} = \begin{cases} R_{\min} & \text{if } G_n > (R_{\min} + \eta) \\ R_{\min}(1 + \text{Limit}/8) & \text{otherwise} \end{cases} \tag{4}$$

where $R_{\min}$ is the minimum runway and Limit (a power of 2) is a control parameter. Choosing $R_{\min}$ as $1/2$ of the entry+exit cost, i.e., $R_{\min} = \eta/2$ provides a good balance between latency and efficiency at moderate utilizations. The above equation implies that if the gap is found large enough in step n, we set $R_n^{(1)}$ to the minimum value for the next step, and otherwise we set it to a large value. In effect, this makes the algorithm more aggressive when it sees larger gaps, although the control is a rather crude band-bang type of control. Because of exponential smoothing, this seems to be adequate.

Let $R_n^{(2)}$ denote the second runway – the one related to hibernation. In S-ESA, this runway is fixed at a multiple of the total transition latency, i.e., $R_n^{(2)} = \eta\,\text{Limit}$ for all n. Since hibernation incurs a large latency penalty, $R_n^{(2)}$ is deliberately kept large, irrespective of the current gap estimate.

In order to show the importance of phase-2 of ESA, we shall also show results for a version of Simple ESA that uses only phase 1. We shall call this variant as T-ESA (trivial ESA).

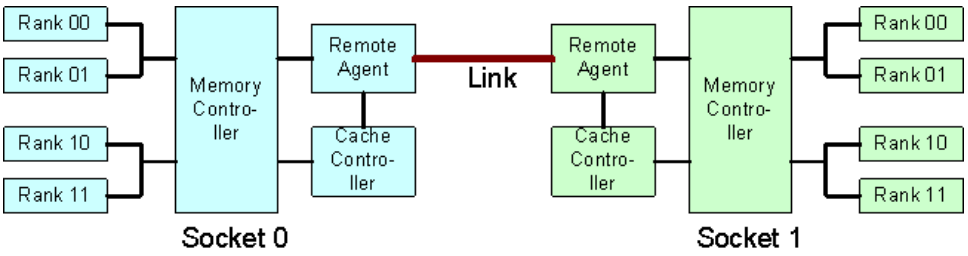

Fig 3: Simulated System Model

3.3.2 Utilization based ESA

The basic idea of utilization based ESA, henceforth called U-ESA, is to make the algorithm more aggressive in entering LPR as the gap size increases (or the utilization goes down). The resource utilization U_n can be estimated as $B_n/(B_n + G_n)$ where B_n is the exponentially smoothed estimate of the nth *busy period*, i.e., period for which the resource was continuously non-idle. B_n is estimated using a equation similar to that for G_n, i.e.,

$$B_n = (1 - \delta)B_{n-1} + \delta b_n \tag{5}$$

where δ is the smoothing coefficient for busy periods. With Limit again as an algorithm parameter, we then set

$$R_n^{(1)} = R_n^{(2)} = U_n R_{\min}\mathrm{Limit} \tag{6}$$

The main problem with this runway setting is the need for multiplication/division, which makes HW realization difficult. We address this issue by using an approximate value $U'_n = 2^{-K_n}$ where K_n is chosen as the smallest value such that shift_left$(B_n, K_n) > G_n$. Because of exponential smoothing, both B_n and G_n change gradually; therefore, K_n is mostly expected to be $K_{n-1} \pm 1$ and can be updated easily.

4 Comparative Algorithm Performance

4.1 Evaluation Model and Workload

For evaluating various algorithms, we built a detailed simulator called LMPOWER that implements a 2-socket platform shown in Fig. 3. Each socket hosts 12 HW threads, each of which executes the same workload. LMPOWER implements a QuickPath like inter-socket link and DDR3 memory controller in significant detail. However, the simulator has a rather simple model of CPU behavior which enforces a limited instruction level parallelism (ILP) per thread, henceforth denoted as N_{thrd}. Thus, whenever a thread has N_{thrd} outstanding memory transactions, it must stall until one of those operations is done. (We chose $N_{thrd} = 4$ for the results here, but it can be random variable that is re-instantiated at the end of each phase.) Other resource limitations such as those arising from link flow control, electrical throttling, memory refresh, limited channel buffers, etc. all ultimately result in thread stall and hence affect throughput in the model.

Various power control algorithms were implemented in detail for both the link and memory ranks. The links support L0s and L1 state with a horizon based promotion from L0s to L1. The memory model supports both "fast" and "slow" CKE and associated "register mode" power control.

The model was calibrated using the best available estimates for variety of link and memory parameters such as speeds, latencies, power consumption, etc. It turns out that the link L0s cost (η) is approximately 20x the average service (transmission) time of link packets. In contrast, η for fast CKE is less than 2x of channel service time. Consequently, the algorithm behavior is generally quite different for links vs. memory.

The model was driven in two ways: (a) using artificially generated traffic that follows traffic specified characteristics, and (b) using the available traces. The analytic traffic generation provides complete freedom in choosing the inter-arrival time distributions and correlation between successive arrivals. (Recall that correlation is crucial for the success of a predictive algorithm). In the following, we present results assuming a Zipf inter-arrival time distribution with a reasonably heavy tail in order to mimic real-life traffic. However, we did not

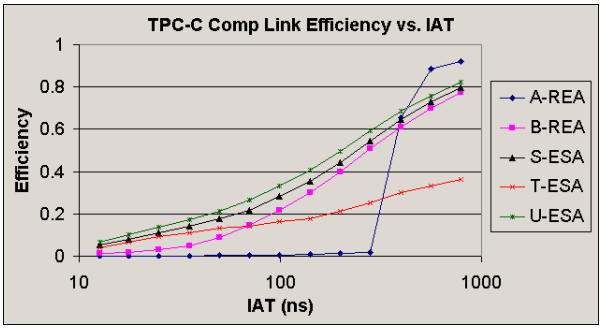

Fig 4: TPC-C Comparative Link Efficiency

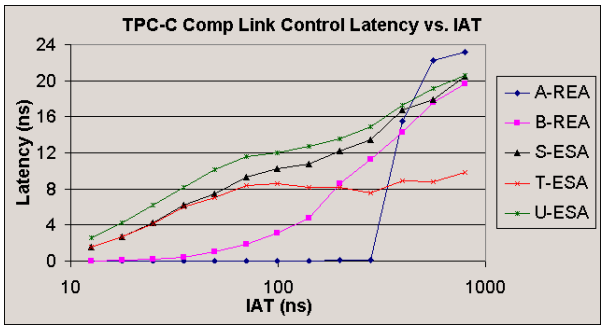

Fig 5: TPC-C Comparative Link Control Latency

artificially introduce correlation into the traffic and this keeps the results conservative (since less correlation means poorer predictibility for proactive algorithms).

Although the results using real traces can be more credible, there were two major deficiencies in the available traces: (1) available traces were for bus-based systems, whereas we want to model link based systems, and (2) nearly all the available traces were for 100% CPU utilization level, whereas we need the traffic at a variety of utilization levels. We handled (1) by modifying the bus traces to look more like link traces (by adding snoops). We handled (2) by a simple stretching of 100% utilization traces, but unfortunately, such a stretching is not very realistic. In view of these deficiencies, trace based results merely represent another data point, rather than something much closer to reality than analytic results. In fact, in view of the fact that even perfect traces would come from a benchmark rather than a real system, analytic results may well be more credible!

4.2 Evaluation Results

We compared the performance of all the algorithms discussed earlier in the paper, namely A-REA, B-REA, S-ESA, T-ESA and U-ESA. These algorithms can be applied equally well to both links and memory ranks; however, we shall continue to focus on links. The reason is that the entry/exit latencies for memory power control are rather small and hence a simple reactive algorithm is optimal. Nevertheless, in order to consider an overall system performance and power impact, power control was always applied to both links and memory ranks.

We exhibit performance of various algorithms as a function of inter-arrival time (IAT) of link/memory transactions. The IAT was scaled up successively by a factor of $\sqrt{2}$ in order to cover a span from 10 ns to 1000 ns. The smallest interarrival time shown in all these figures (10) corresponds to about 30% link and memory channel utilization and 100% CPU utilization.

Figs 4 and 5 show, respectively, the efficiency and additional latency added due to power control. The overall power consumption is closely related to the efficiency of the algorithm and thus not shown.

The first thing to note about these graphs is that the dynamic control of runway for A-REA does not perform very well. This is because the control continuously attempts to "hunt" for the right runway and in the process ends up either on one extreme or the other. We have repeatedly seen this type of erratic looking behavior with a number of "hunting" type of algorithms that we tried out. The take off point depends on the algorithm parameters and traffic characteristics.

Ideally, we would like to see a monotonic increase in efficiency and latency of the algorithm as the link utilization goes down. This follows from the fact that as the utilization decreases, the latency becomes less important and efficiency becomes more important. The B-REA algorithm appears quite well behaved in this regard. It provides lower efficiency but at lower latency cost as compared to the proactive algorithms; however, this should by no means be read as a general statement. The relative efficiency/latency very much depend on the algorithm parameters and the traffic characteristics. The only general observation is that with comparable parameterization, at sufficiently low utilization level, B-REA will beat proactive algorithm in efficiency since it can take advantage of the entire gap.

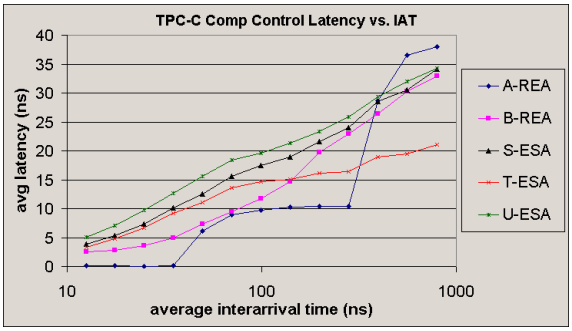

Fig 6: TPC-C Comparative Control Latency

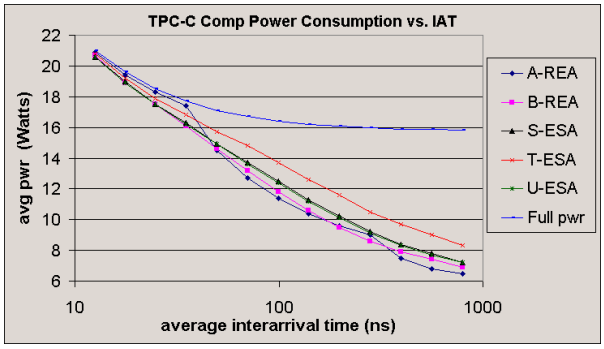

Fig 7: TPC-C Comparative Power Consumption

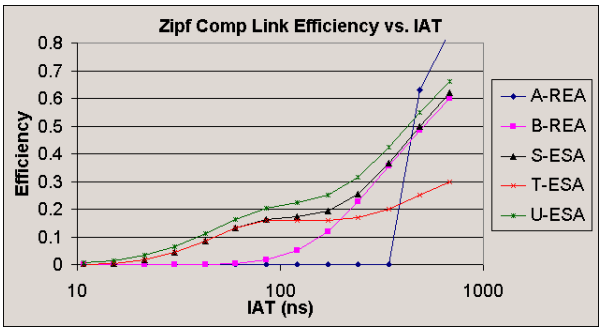

Fig 8: Zipf Comparative Link Efficiency

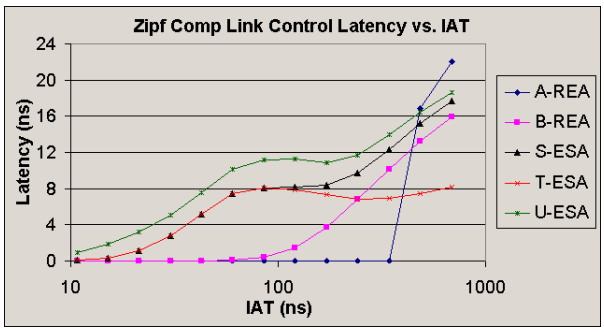

Fig 9: Zipf Comparative Link Control Latency

Of the proactive algorithms, T-ESA consistently performs rather badly. This can be seen, for example, by its flattening behavior which is undesirable. This clearly points to the fact that a single proactive phase isn't adequate. The remaining two algorithms, namely S-ESA and U-ESA perform similar in nearly all the cases, with U-ESA giving somewhat better efficiency at somewhat higher latency. This is a bit surprising in view of the fact that U-ESA tries to do a better job of controlling the runway. Since S-ESA is simpler and does not require estimation of the on-time, it would be preferred.

Figs 6 and 7 show the added average power control latency and average power consumption for the link-memory combine. An even more erratic behavior of A-REA is obvious, but may be somewhat surprising. This is because each resource has its own "take-off" point for A-REA, and thus the combination results in a more complex behavior. The other algorithms are still well behaved. In particular B-REA has somewhat lower latency but similar power consumption than proactive algorithms. Overall, it seems that B-REA has a bit of an edge here.

Figs 8 and 9 show efficiency and latency for the inter-socket link under Zipf traffic. It is seen that B-REA in this case provides very low efficiency (and hence latency) at moderate utilizations but eventually catches up with U/S-ESA. Basically, these graphs show there are regions (i.e., moderate utilization) where a proactive algorithm could be useful. Needless to say that higher efficiency for proactive algorithms is accompanied with higher latency. However, if we had used a smaller runway for B-REA, we could raise its efficiency & latency, but a rapid rise in the latency of B-REA could be undesirable.

Figs 10 and 11 show the added average power control latency and the quality metric QM_2. The combined effect of non-smooth behavior of multiple resources becomes more obvious here for both A-REA and B-REA. It is interesting to note that the proactive algorithms still show a smooth behavior except for a small "hump" developing in the middel. The "hump" can be made smaller by controlling some of the U/S-ESA parameters, as discussed later.

The quality metric shown in Fig 11 indicates that the proactive algorithms do not particularly distinguish themselves from reactive ones; however, they show a "smoother" behavior.

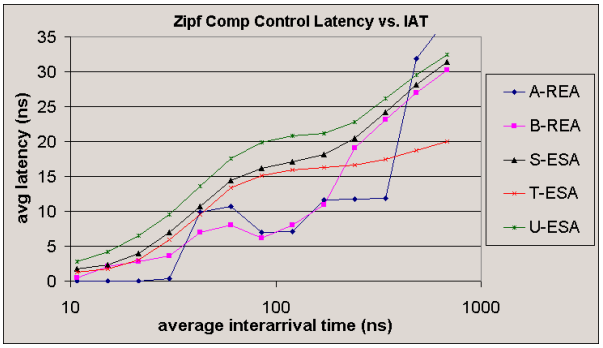

Fig 10: Zipf Comparative Control Latency

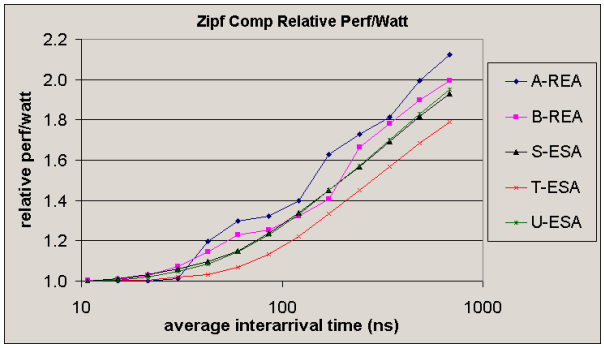

Fig 11: Zipf Comparative Relative Perf/Watt

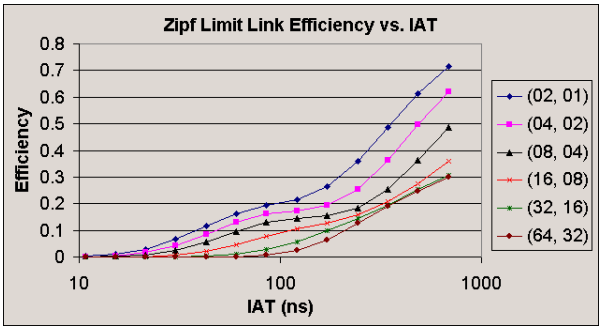

Fig 12: Zipf Link Efficiency vs. Limit

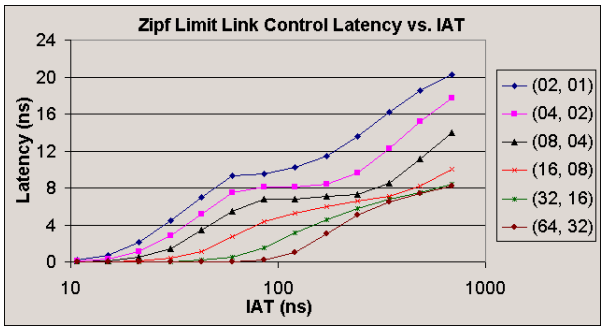

Fig 13: Zipf Link Control Latency vs. Limit

5 Implementation of S-ESA Algorithm

With S-ESA proven as a simple and effective proactive algorithm, we further examine its implementation issues. There are two key parameters for controlling S-ESA behavior which need to be made adjustable in an implementation.

1. Minimum runway size $R_{\min}$. We chose the default value of $0.5\,\eta$ in our results; however, the runway can be adjusted depending on the desired latency sensitivity.

2. Limit parameter, which controls the second runway and the dynamic changes to the first runway. We shall shortly show the impact of Limit on algorithm performance.

Figs 12 and 13 show the efficiency and latency for S-ESA with various values of Limit. Here we used S-ESA for both links and memory ranks and hence the legend is marked as a pair of limits, i.e., (Link Limit, Memory Limit). Of course, only the link Limit is relevant here. Since a larger limit will stretch out both runways of S-ESA, both latency and efficiency decrease as the Limit value goes up. We also see a diminishing returns behavior when Limit is large. We can use these graphs to dial the desired latency.

Figs 14 and 15 show the impact of minimum runway $R_{\min}$ on the behavior. As with the Limit, we have a pair of numbers in the legend, first one for the link and second for the memory. ($R_{\min}$ is actually the given number multiplied by the cost η.) As expected, an increase in runway decreases both efficiency and latency. A more interesting observation is how the "hump" in the curves gets smaller with increasing runway. Our default choice of default runway of $1/2\,\eta$ is a compromise between efficiency and hump magnitude.

We now consider an actual HW implementation of S-ESA. One important issue in HW implementation of proactive algorithms is the "precision" required. A HW implementation must necessarily discretize all time measurements in units of a "time-slot" τ. (Note that τ itself is some multiple of the underlying clock period.) Furthermore, all time periods must be represented using certain number of bits, which we denote as N. This

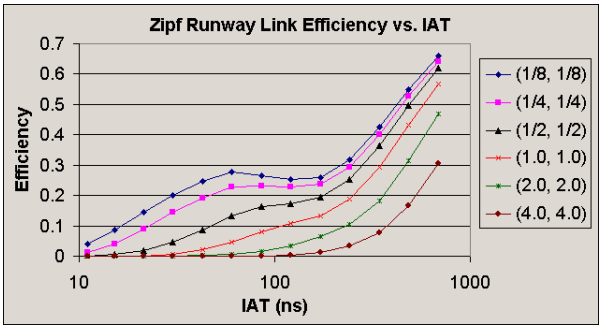

Fig 14: Zipf Link Efficiency vs. Runway

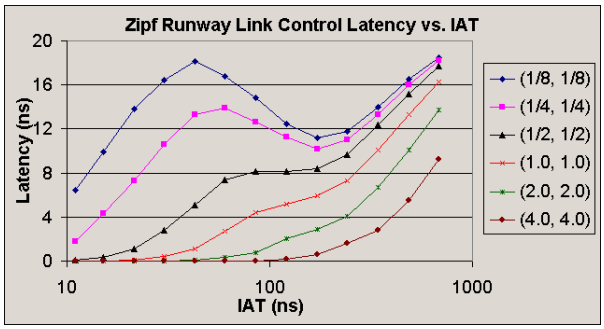

Fig 15: Zipf Link Control Latency vs. Runway

would allow time periods in the range τ and $(2^N-1)\tau$. We determined experimentally that $N \geq 11$ was required to provide a resonable range of gap sizes and we chose $N = 12$. The time-slot τ also needs to be chosen carefully – it must be large enough to work well for large gaps, but small enough not to lose discrimination for small gaps. A τ value of the order of 4-8 times the average service time of a request is a resonable compromise in this regard.

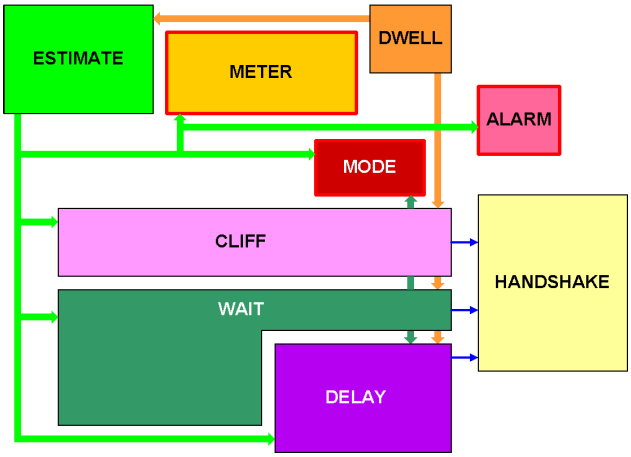

Fig 16: Block diagram of S-ESA

Another important issue is that of precision in the exponential smoothing. The calculations must be done at higher precision than N bits, else the result will have a very high truncation error. We determined experimentally that additional 4 bits of precision in exponential smoothing is adequate to achieve good accuracy.

Fig. 16 shows the block diagram of S-ESA and includes blocks for runway estimation (WAIT), LPR time computation (DELAY), exponential smoothing (ESTIMATE), performance metering (METER), performance indication (ALARM), and mode change (CLIFF). The CLIFF part allows the implementation to cut out the proactive part at very low utilizations, since the reactive algorithm is optimal below certain level of activity. Basically, the CLIFF encodes the starting time of second S-ESA phase relative to the beginning of first phase and this time is made fixed (rather than a variable as suggested in the earlier discussion). In this case, if CLIFF timer expires while the resource is in time-sleep mode (phase 1), the resource will continue on in the low-power mode. The metering and alarm parts are optional and are intended to provide statistics/alarm to external power control entity. The statistics could be used by the power control unit to make other power control decisions or perhaps even tweak the S-ESA parameters dynamically. The overall implementation amounts to about 4000 gates w/o the metering/alarm part and about 9000 gates with them. (The metering part does involve variance calculation and could be reduced if only the mean values are computed.) Either of these sizes are small enough so as to allow a separate S-ESA module for each link port (and each memory rank if desired).

6 Conclusions and Future Work

In this paper we studied proactive idle time power control algorithms and compared them to simpler reactive algorithms. We find that exponential smoothing based proactive algorithms show a consistent behavior under a wide variety of traffic conditions and can be useful in extending the operating region towards higher utilization levels. We also showed that the proposed simple ESA algorithm has a good behavior over a wide range of

utilizations. It has two control parameters, namely minimum runway $R_{\min}$ and Limit, which can be used to control latency over a rather wide range. The algorithm is designed to revert to reactive mode at very low utilizations where reactive exit is optimal.

The power control algorithms discussed here can be used in a variety of other contexts such as selectively putting CPU cores in non-operational states (e.g., C6), putting cache blocks in low power mode, power control of NIC cards and indeed that of entire platforms. However, depending on the tolerable complexity and available power states, further enhancements may be made. For example, one enhancement already in place for links is the low-power exit in one direction of the link triggering exit in the other direction. Such a coupling can significantly reduce the power control latency w/o affecting the efficiency of the algorithm.

Acknowledgements: The authors would like to acknowledge discussions with Rahul Khanna and Raj Ramanujan.

References

[1] V. Venkatachalam and M. Franz, "Power Reduction Techniques for Microprocessors", ACM computing surveys, Vol 37, NO 3, Sept 2005, pp 195-237. (http://www.ics.uci.edu/ vvenkata/finalpaper.pdf)

[2] V. Raghunathan, M. B. Srivastava, and R. K. Gupta. A survey of techniques for energy efficient on-chip communication. In Proc. the 40th Conference on Design Automation, 2003.

[3] P.J. Brockwell and R.A. Davis, *Introduction to Time Series and Forecasting*, Springer-Verlag, 1996.