GANGSTA: An Energy Aware Arbiter for Multicore Systems

Sean M. Arietta, William P. Burns

Abstract

The increasing prevalence of multicore machines creates an opportunity to expand on the notion of green computing. By expanding the overall goals of a general purpose scheduler to consider energy use, we were able to investigate the tradeoffs between execution time and power consumption in multicore commodity systems. Our method is based on the idea that at any given moment a computer does not necessarily utilize all of its cores. This creates an opportunity to reduce power consumption by effectively turning superfluous cores off. In this research, we describe an energy aware arbiter for multicore systems having the goal of minimizing energy while at the same time leveraging performance. We have devised an algorithm that turns off power to processing cores that have entered an idle state as a result of I/O blocking and/or process termination. We present our experiments as a function of the power needed to actually perform an on or off operation on an individual core. Since this functionality is not present in current technology this research serves as an indicator for CPU designers who wish to implement a power aware architecture. Our experiments show that we are able to diminish the total power used by the system by an order of magnitude with an approximate decrease in performance of 13% in the worst case and less than 1% in the best case.

1 Introduction

It is estimated that the 200 million computers in use within the United States consume approximately 85 billion kWh of electricity per year [Horowitz et al. 2003]. Current processors do not adequately address this overwhelming issue of power consumption. When energy efficiency is considered in practical implementations - laptops for instance - it is often the case that the actual part of the machine that attempts to minimize energy is the hardware and not the software ([Kin et al. 1997], [Burd and Brodersen 1995]). More mainstream versions of minimized power consumption include the 'hibernate' ability available in most newer operating systems ([Int 2004]). While this does have the desired effect of lower energy costs, it does so at the expense of rendering the system completely useless. A more appropriate approach to this problem is to have the operating system itself attempt to manage its processors in a more energy-aware manner without incuring extreme computation degradation (such as temporarily turning off the entire system).

We propose a power aware core arbiter to extend the general purpose operating system scheduler's goals to include energy efficiency. By varying the number of powered on cores we are able to decrease energy use of a system while at the same time mitigating performance degradation. Our simple observation is that in many multicore systems it may not be necessary to utilize all of the cores at any given instant. We exploit this idea by devising a strategy for executing processes across a variable number of cores with the goal of decreasing the total power consumed by the system. This is accomplished by either turning off an unnecessary core to decrease power consumption or turning on an additional core to mitigate performance degradation. In essense, our arbiter is a specialized scheduler that makes decisions about when to turn cores on and off. For the purposes of this paper, however, we will refer to it simply as an arbiter as it is not changing the order of processes from an already existent 'black box' scheduler, but rather regulating the number of usable cores.

Although we understand that this research is relevant to common operational workloads we focus specifically on single-threaded batched workloads. Our workloads consist of a set of processes that compete for cores to perform computation and periodically submit I/O requests. These processes 'wake up' at a specific time, perform their set of computations and I/O requests, and then terminate. The direct uses of this type of system can be seen in the context of overnight backups, large-scale periodic computations (such as in the Physics community), web-crawling, etc.

We structure this paper by first introducing relevant previous work in Section 2. We then describe the environment within which we tested our approach in Section 3. Following that, we describe the design of our energy-aware arbiter in Section 4. In Section 5.2 we describe the energy benefits of our system and follow this with an analysis of performance degradation in Section 5.3. Section 6 considers the limitations of our system and recommended future work. Finally in Section 7 we provide the reader with the conclusions drawn from this work.

2 Previous Work

Our most motivating pieces of research are that of [Vahdat et al. 2000] and [Hu et al. 2004]. The former advocates revisiting the design of operating systems to respect consumption of power. Of course we do not attempt to devise an entire operating system (just an arbiter). We see our work as one small part of their greater goal of a complete operating system that concerns itself with minimizing power at the risk of performance degradation. The latter is a systematic test of the power consumption gains one can achieve in the presence of power-gating. Power-gating is a process by which the power consumed by a core is slowly reduced to decrease its power consumption. They show encouraging results but do not consider completely turning cores off as we do in this work.

The method of [Meng et al. 2008] seems to be the most recent applicable research in this area. They describe an adaptive technique that balances a process load across multiple processors with the goal of meeting a global power budget. This is similar to our work in that we too wish to find a distribution of the process load across multiple processors. The main difference is that they achieve this by managing the constituent resources of the system (voltage regulation, cache resizing, etc). Moreover, current systems do not necessarily provide the necessary capabilities to perform these tasks and so we can consider ourselves more practical in the short-term. We avoid the complexities associated with trying to balance these very intricate interactions by simply noting that it is often the case that a processing core can simply be turned off when it is not in use.

We also note the work of [Yang et al. 2001], [Jejurikar and Gupta 2002], and [Mishra et al. 2003]. All of these research efforts are concerned with energy-aware scheduling in various configurations (real-time, embedded, distributed systems). Our method should be considered parallel to these research projects, but different in approach and application domain.

3 Simulator

As we stated before, the necessary hardware to take advantage of our arbiter is not currently in production. In order to test our approach and draw conclusions on our success we were required to

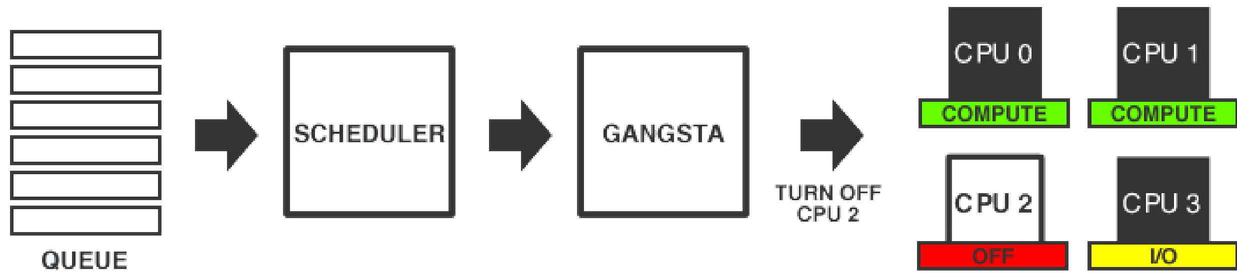

Figure 1: A graphical overview of our system. Processes exist in a process queue. A black-box scheduler schedulers these processes as it normally would. GANGSTA receives the scheduled jobs and decides whether to turn cores on or off depending on the execution state of each core and the processes handed to it by the black-box scheduler.

perform all evaluation in simulation. Our simulator was designed in Java and was specially tailored to enable us to conduct the experiments detailed in the following sections. We first detail the assumptions that were made in the design of the simulator, followed by a detailed explanation, and finally a discussion about its verification.

3.1 Assumptions

Unfortunately since we were forced to perform our emperical studies in simulation we had to explicitly assume certain aspects of our environment. Our basic assumption is that our arbiter is to be used for batch style workloads whereby a set of processes compete for CPU resources (in the form of execution cores). In many cases our design decisions were made to reflect our best attempts to quantify our specific problem. We outline some of the major assumptions we made in the following subsections. We would like to remain explicit that we are aware of some of the engineering challenges faced by an architect in designing a system to incorporate our energy arbiter including flushing of dirty cache lines, feedback noise caused by abrupt power disconnection, and power-gating to specific cores [Hu et al. 2004]. Describing the implementation of the strategies we present is beyond the scope of this paper and our research goals. Rather we are providing a proof of concept to the benefits of using an energy-aware arbiter.

3.1.1 I/O

We made no definitive assumptions about the available I/O. In our simulations the system effectively has infinite I/O resources. We felt this was appropriate as it would only convolute our experimentation to involve ourselves in the process of writing I/O schedulers and/or guessing the number and type of I/O devices in ’typical’ systems. We did, however, certainly want to include I/O in our simulation for completeness. Our treatment of I/O is that any process at any time can submit an I/O request. Our workload generator (see Section 5.1) chooses the number of cycles the I/O requests consume to reflect a specific compute to I/O ratio. We use [Patterson and Hennessy 2007]’s assertion that I/O is usually on the order of one million times slower than on chip memory to weight the amount of cycles that should actually be spent on I/O given a specific compute to I/O ratio. For instance, if the compute to I/O ratio goal is 1, then the number of cycles spent in I/O should be approximately one million times more than those spent on the cores. This is included so that neither computation nor I/O is weighted unfairly.

When a process does request I/O it can be removed from the black-box scheduler’s queue, effectively freeing a core for another process to execute on. Often it is during these times that our arbiter can provide a win. If there are no more available processes waiting for a core when the process that requested I/O is dequeued then there is an idle core. Our system exploits idle cores to conserve energy as explained in Section 4.

3.1.2 Cores

Since some of the operations we are advocating have not yet been implemented we were forced to make heavier assumptions concerning execution cores in our simulation. Specifically, there has been no publicly available processing unit that has included a feature to fully turn them off. Therefore, we had to make assumptions about how many cycles it would take to turn a core completely off and then turn it back on. We imagined that in the worst case all of the registers of the machine would have to be saved to main memory to turn the core off and then reloaded to turn it back on. We made a conservative estimate of 1,000 cycles for both operations [Mogul and Borg 1991]. We derived this from the observation that when a core needs to be turned off, it is essentially performing a context switch to a null process instead of a normal process. We are confident that this is likely a higher penalty than what an actual production system could provide. In fact, if such a system included a small solid-state storage for each core for exactly this purpose then there would be a dramatic reduction in latency.

A core in our simulator is modeled as a single execution device consisting of three states: running, idle, and off. Running corresponds to a core that has a process executing on it. A core becomes idle when a process ceases to execute on it either from an I/O request or from termination of the process. The off state corresponds to when a core is physically turned off and not using any power. Any process can be scheduled on any core and can induce a state change, however only our arbiter can force a core to transition from idling to off or from off to idling. In addition, any core that is in an idling state can enter the running state via the black-box scheduler’s decision.

3.1.3 Schedulers and GANGSTA

We do not explicitly account for cycles consumed by either the black-box schedulers or the arbiter. We chose this implementation so that we would not complicate our experiments and results. Unfortunately this model does not allow us to test the extra cpu overhead that would be needed for the arbiter to function. We admit this as a small shortcoming of our implementation but note that our algorithm is a few short statements that perform a very terse set of conditional jumps. These could easily be incorporated into any of

the existing black box schedulers with little to no overhead. In fact, a fixed-function unit may also be appropriate that would effectively mitigate any overhead incurred on our behalf.

3.2 Design

Our primary goal for the design of our simulator was to test our algorithm. We set out to ensure we could generate accurate results without an over-complicating implementation. To achieve this goal, we decided to take an object-oriented approach using Java. This allowed us to create a generic template for our process schedulers since most scheduling implementations exhibit a high rate of redundancy. Additionally, this allowed us to ensure changes we made were reflected in every simulation regardless of the scheduler used.

For this project we implemented the most common scheduling algorithms employed today: First Come First Serve, Shortest Job First, Round Robin, and Multi-Level Feedback Queue. We refrained from implementing a Priority Queue as we would have had to arbitrarily assign priorities to processes in order to differ from First Come First Serve. We did not want to limit ourselves by having to make these types of assumptions so we excluded it from the simulator. To remain consistent across the various schedulers, we defined a generic Job class which represented a single process. Thus, a simulation consists of scheduling and running an array of Jobs, which we refer to as a Workload. Our simulator is fed one Workload at a time. Each Job in the Workload contains information regarding the time it should enter the processing queue along with a series of numbers indicating clock cycles to be simulated for a series of compute and I/O operations.

More specifically, we define a Job as a tuple t, b. The first variable, t, represents the arrival time of the process. Prior to this time the process resides in a waiting queue. The process scheduler is unaware that this process exists until it leaves the waiting queue as is placed on the processing queue. The second variable, b, is actually an array of 2-tuples c, i where c is the length of a single compute burst and i is the length of a single I/O burst. A Job can contain an arbitrary number of cpmpute and I/O bursts. Generally these are just specified in a text file or, in the case of our experiments, generated externally and then saved to text files.

3.3 Verification

The verification of our simulator was paramount to this research. Because we so heavily rely on its output we worked very hard to ensure it was verified. We set out to accomplish this by generating a series of tests on each of the black-box schedulers. For each test, we computed the resulting output by hand and checked it against the simulator output. We found several bugs as a result of these tests. After fixing all of the errors, we found the behavoir of our simulator to be in exact accordance.

4 The Arbiter

The goal of our paper is provide an energy-aware arbiter that is capable of easy integration with an existing scheduler. Our design reflects an approach that we believe balances the degradation of performance while still maintaining a low power profile. We had discussed other ideas but ruled them out for one reason or another. For instance, we had originally planned on implementing a predictive arbiter that could possibly infer the execution style of a process to predict the likelihood we could turn a core on or off. We set out to cast the problem in the context of a Markov chain and then use Bayes' Theorem to compute the maximum a posteriori estimate of the next execution cycle of a process given the current execution state and a prior reflecting the aggregated statistics of the processes' execution history. We ruled this method out as it was too computationally intensive and not necessarily better than our final implementation.

Algorithm 1 Pseudocode for GANGSTA

```
for all C ∈ Total Cores do
    if state(C) == IDLE AND isEmpty(QUEUE) then
        Set state(C) = OFF
    end if
    if !isempty(QUEUE) AND state(C) == OFF then
        Set state(C) = ON
    end if
end for
```

The method that we eventually converged to is terse but efficient and generally effective. The basic pseudo-code is given in Algorithm 1. At each simulation step, we determine whether there are any idling cores. If there are idling cores and there is no process waiting in the process queue then we turn that core off. The idea behind this simplistic algorithm is that processes often block for I/O for long periods of time during which there may not be another process that we can load onto a core via a context switch. This is not necessarily generally true of general purpose systems, but is often the case in batch systems.

As stated in Section 3.1.3, because of the lightweight characterstics of our arbiter - in terms of cycles needed to decide whether or not to turn a core off - we are confident that its inclusion in an existing scheduler will be trivial. In addition, we can imagine cases where it would behoove a designer to include GANGSTA in a fixed-funtion unit included in the actual CPU package.

Although there are cases in which our arbiter will degrade performance significantly, these are relatively rare in the context batch systems. For a more detailed explanation of failure cases please see Section 6. Despite these rare cases, our arbiter does attempt to mitigate performance penalties. Since it never turns off a core that is currently in use or could be in use by a process waiting on the process queue, it does not incur harsh performance penalties. These qualities, taken as a whole, respect our goal of creating an energy-aware arbiter that does not incur a high performance overhead.

5 Evaluation and Analysis

The fundamental research question we are tying to answer is: 'What are the benefits of using a lightweight power-aware arbiter in tandem with an existing process scheduler.' We have explained in the previous sections our assumptions and strategies for developing such a system in light of this goal. In this section we are interested in determining the efficacy of our approach. We ran a series of tests, which are described in the following sections in more detail, to test our method under a variety of different conditions. In all experiments we used a system with 16 cores. We chose this number as it is slightly higher than typical core counts in current systems, but it is available. We wanted to be as up-to-date as possible while still demonstrating our effectiveness in the pressence of many cores. The first question we were interested in answering was how much power are we able to save with our arbiter in place. Section 5.2 is focused on this point. Next, we were interested in the performance reduction engendered by our arbiter. Finally, Section 5.3 is a detailing of the experiments we ran to measure this effect.

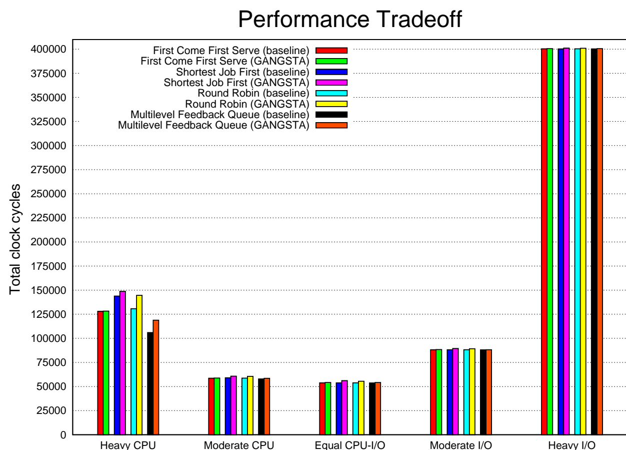

Figure 2: Performance results for our arbiter paired without our arbiter. Note that the y-axis indicates the number of total cycles we measured for each of the Workloads. This number by itself is not as important as the relative height of corresponding bars. In general GANSTA performs almost as well as the normal black-box scheduler without any energy awareness.

5.1 Workloads

As aforementioned, we are primarily concerned with batch systems in this paper. With this goal in mind we designed five Workloads to best sample the types of batch systems our arbiter would be incorporated into. Each of the workloads varies in the ratio of compute to I/O bursts included in the Workload file (detailed in Section 3. The five Workloads we generated correspond to ratios of 100:1, 10:1, and 1:1. The 100:1 and 10:1 ratios were generated for both compute to I/O and I/O to compute Workloads, hence there were five total Workloads. We introduced randomness into our Workloads by psuedo-randomly sampling a uniform distribution to determine the t, b tuples described in Section 3.2. Each Workload in our tests consisted of 1000 processes derived in this manner with a particular compute to I/O ratio. We decided to create Workloads in this way so that we could report how useful our arbiter would be given a particular style of use.

5.2 Energy Consumption Analysis

In this section we describe and analyze the main contribution of this paper. Our experiments were run with the Workloads explained in the previous section. For each Workload, we ran our simulation for each of the black-box schedulers we considered once without GANGSTA and then once with GANGSTA attached. Recall we implemented the First Come First Serve, Shortest Job First, Round Robbin, and Multilevel Feedback Queue black-box scheduling algorithms. The scheduler outputs the total number of cycles it took to complete the job T, the total number of cycles that the cores used collectively while in a run state T_{RUN}, the total number of cycles the cores used collectively while in an idle state T_{IDLE}, the total number of times any core was turned off ($OFFS$), and the total number of times any core was turned on (ONS). From this information, we computed the total power used during the course of a Workload from:

$$\frac{T_{RUN} \cdot C_{RUN} + T_{IDLE} \cdot C_{IDLE} + (ONS + OFFS) \cdot \alpha \cdot T_{OFF}}{T} \tag{1}$$

Recall that we have set T_{OFF} to be 1,000 as described in Section 3.1.2. Notice that we have one variable α which is the total amount of power that would be consumed while turning a core off or on. In our experiments, we let α vary from 0W to 100W as we found from [Chin 2006] that even at high workloads, the power consumption of a core is below this threshold. We left this as a parameter for two reasons. First, since current CPU's do not include this functionality we did not want to make an incorrect assumption. Second, we see this work as a way for architects to determine whether GANGSTA would be useful on their platform. For our purposes, we used the results of [Chin 2006] who describe an experiment that tests the power consumption of an Intel PD-820 running at 2.8 GHz. We chose this processor because in their tests, they show no reduction in clock speed as a result of idling. Based on their observations we set C_{RUN} to be 90.5W and C_{IDLE} to be 26.7W.

Figure 3 shows the results of our experiments. As you can see in many cases we always consume less power than a normal system, particularly in either very I/O or very compute intensive Workloads. This is because when we turn a core off in one of these cases, it is unlikely that the processes will deviate from their current execution state. In the case of heavy computation, processes are not likely to interlace much I/O with their computation so when a core is idle with nothing left in the queue, it is unlikely that a process will be coming back from an I/O block and thereby cause a core that may have just been turned off to be turned right back on. Similarly, when a process does any moderate to heavy amount of I/O, it is unlikely that the state of the processes will change. This is primarily due to the long duration needed to complete I/O. In situations where processes do not tend to follow a very predictable execution path (moderate computation or equal computation and I/O), the arbiter erroneously turns cores off that are likely to be needed in the near future. In these cases, there is a high power penealty because we are incurring the turn of and turn off overhead many times throughout the execution of a program.

Our results indicate that we are able to save an order of magnitude in almost all cases. These are encouraging results considering the very straightforward algorithm we described in Section 4. We are confident that our experiments indicate the practicality of GANGSTA in real-world systems. We provide these graphs as a way for architects to determine the usefullness of GANGSTA in their systems. We are careful to note that there is some overhead associated with using GANGSTA. The performance degradation is described in detail in the next section.

5.3 Performance Analysis

No discussion of a computer science topic would be complete without a section on performance. Although our primary motivation is to conserve power, as with any piece of research we are inherently concerned with the degradation of performance that comes as a result of using our arbiter. We used the same Workloads as we did in Section 5.2. To determine the relative performance observed when using GANGSTA versus using just the black-box scheduler under inspection, we simply took the ratio of the two T's reported by the simulator. Since this is a measure of the total amount of time it takes for a Workload to finish it is our best measure of performance. Figure 2 shows the results of these experiments. Notice that in almost all cases we incur very little overhead further supporting our claim that energy-aware arbitration is a worthwhile facet to implement in next generating multicore systems. Even in the worst case we are within 15% of the performance of the system without the arbiter.

6 Limitations and Future Work

The biggest limitation with our system is that it must make a decision without any knowledge of how good or bad that decision may be. We had originally planned to test a probabilstic algorithm set in the context of a Markov Chain. We decided against this approach because we were afraid of the large overhead that might be associated with attempting to make a decision at each time step. Recall that our algorithm is a very terse set of conditionals. However, we do anticipate that such a system may be able to outperform our own because we effectively throw away any past history of a process's execution lifetime. One direction of future work is dedicated to this point. We are confident that the current version of GANGSTA is useful, but we are also optimistic about improvements in this specific context.

Another area of future work we would like to consider is the scalability of our algorithm. Currently we operate under the assumption of 16 cores, which is a proper number in today's computing environment. However, in the future, we hope to expand GANGSTA to scale with more cores. Since GANGSTA is a global algorithm we anticipate this being a very trivial exercise but nonetheless important. We excluded an exact treatment of the question of scale in this paper as we did not set out to prove the scalability of our approach, merely that it could benefit current processors.

7 Conclusions

We have described GANGSTA, an energy aware arbiter for multicore systems. Our approach requires very little additional complexity to an already existing process scheduler, but is effective in reducing the overall power consumption of a process workload without incurring high performance penalties. We have shown results that indicate we have succeeded in developing an arbiter that meets these design goals. It is our hope that the results presented in this paper act as a guide to manufacturers of next-generation processors who wish to reduce the energy footprint of their products. In addition, we wish to encourage more research in the area of energy-aware computing. We hope that GANGSTA will serve as one small part in a greater whole that seeks to reduce the overwhelming energy footprint of computers in the modern world.

References

BURD, T. D., AND BRODERSEN, R. W. 1995. Energy efficient cmos microprocessor design. In *In Proc. of The HICSS Conference*, 288–297.

CHIN, M., 2006. Desktop cpu power survey. http://www.silentpcreview.com/article313-page1.html, April.

HOROWITZ, N., FOSTER, S., AND C., C. 2003. Laptop computers: How much energy do they use and how much can we save. *Natural Resources Defense Council.*

HU, Z., BUYUKTOSUNOGLU, A., SRINIVASAN, V., ZYUBAN, V., JACOBSON, H., AND BOSE, P. 2004. Microarchitectural techniques for power gating of execution units. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, ACM, New York, NY, USA, 32–37.

INTEL CORPORATION. 2004. *Mobile Intel Pentium 4 Processor with 533 MHz Front Side Bus Datasheet*, January.

JEJURIKAR, R., AND GUPTA, R. 2002. Energy aware task scheduling with task synchronization for embedded real time systems. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, ACM, New York, NY, USA, 164–169.

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. H. 1997. The filter cache: an energy efficient memory structure. In *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, IEEE Computer Society, Washington, DC, USA, 184–193.

MENG, K., JOSEPH, R., DICK, R. P., AND SHANG, L. 2008. Multi-optimization power management for chip multiprocessors. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, New York, NY, USA, 177–186.

MISHRA, R., RASTOGI, N., ZHU, D., MOSSE, D., AND MELHEM, R. 2003. Energy aware scheduling for distributed real-time systems. *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* (April), 9.

MOGUL, J. C., AND BORG, A. 1991. The effect of context switches on cache performance. *SIGARCH Comput. Archit. News 19*, 2, 75–84.

PATTERSON, D. A., AND HENNESSY, J. L. 2007. *Computer Organization and Design: The Hardware/Sofware Interface*, 3rd ed. Morgan Kaufmann, 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA.

VAHDAT, A., LEBECK, A., AND ELLIS, C. S. 2000. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, ACM, New York, NY, USA, 31–36.

YANG, P., WONG, C., MARCHAL, P., CATTHOOR, F., DESMET, D., VERKEST, D., AND LAUWEREINS, R. 2001. Energy-aware runtime scheduling for embedded-multiprocessor socs. *Design Test of Computers, IEEE 18*, 5 (Sep-Oct), 46–58.

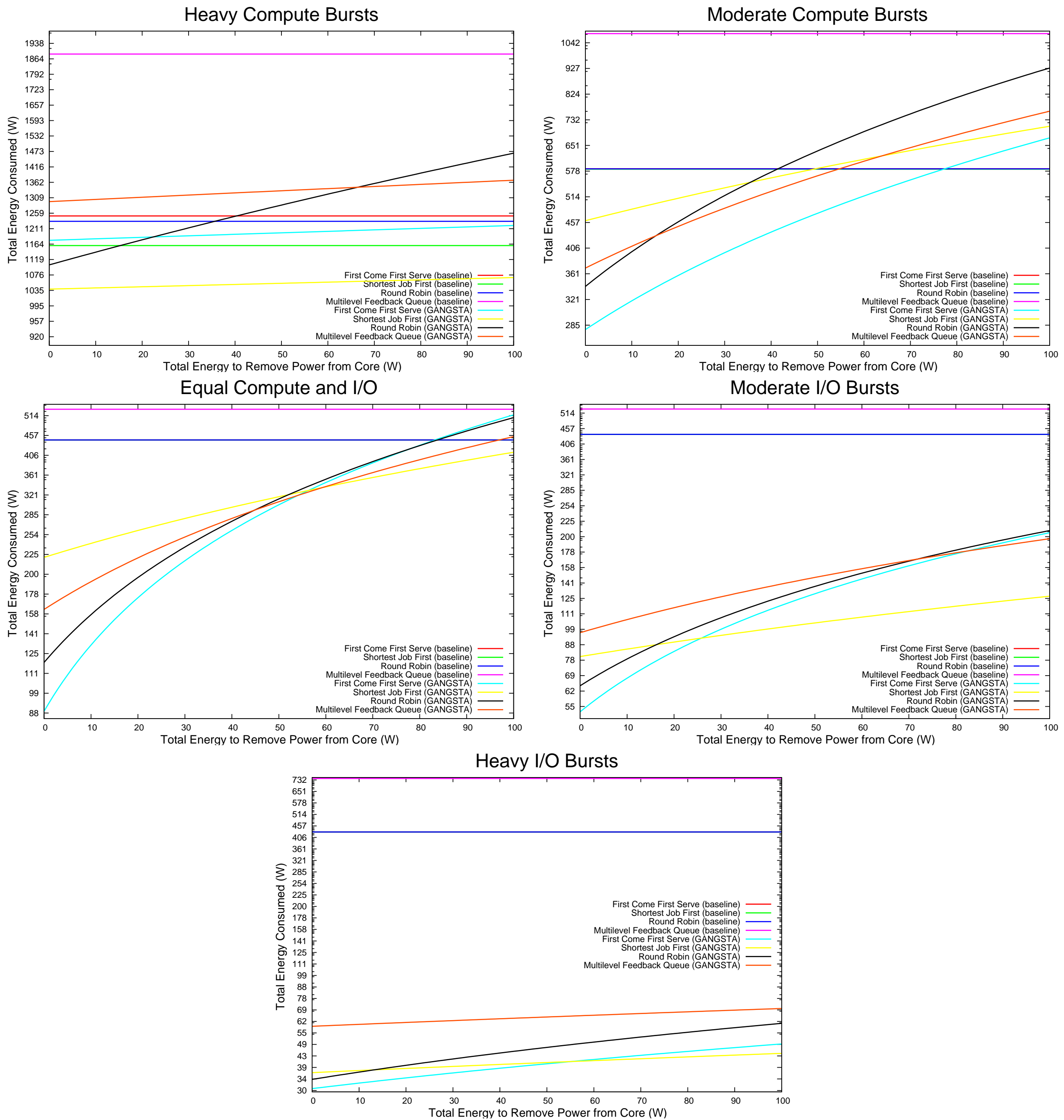

Figure 3: A series of graphs showing the results of our power consumption tests. In almost cases GANGSTA provides a reduction in power consumption, and often even in the limit of the turnoff penalty (α) GANGSTA can reduce power consumption. NOTE: the y-axis is in log scale to accentuate finer detail.