Balancing Responsiveness and Power Consumption in Real Time Operating Systems

Cesar Augusto Marcelino dos Santos, Jim Deverick
{cesaraugusto.dossantos, jdeverick}@richmond.edu

ABSTRACT
Real Time Systems have their request by market becoming more and more common. One of the important concerns related to it is power consumption. According to the results of this study, is possible to reduce it by choosing a proper process scheduler and still achieve a good responsiveness.

Keywords: real time systems, x86 architecture, operating systems, embedded systems, power consumption

INTRODUCTION
Real Time Systems are designed to achieve good responsiveness for determined processes. This not only means to give priority for this process, but also being quickly responded when does a request. In order to achieve such results, comprehension of what is a real time task, Operating System and its scheduler is necessary.

1. Real Time tasks
Real time task has the same properties as a regular process: it contains a Process Control Block (PCB), which has all the necessary information to restore a process when it is preempted. The main difference is that saving all this data and loading a new one should be optimized in order to be very fast, what is called context switching[1].

2. Operating Systems
There is no accurate definition for such wide concept, but can be seen as a program that acts as an intermediary between a user of a computer and the computer hardware. Its goals are execute user programs and make solving user problems easier, make the computer system convenient to use and use the computer hardware in an efficient manner[1].
The difference when a Real Time Operating System comes into play is that it has well-defined fixed time constraints, so processing must be done within constraint and correctness happen only in case constraints met[1].

4. Process scheduler
Scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance a system effectively or achieve a target quality of service.
Some examples of scheduling policies are First-Come, First-Served – FCFS (also called First-Int, First-Out – FIFO) and Round-Robin.
FCFS or FIFO means that a process, when called by the system to perform, will be executed until it completely finishes its task. For Round-Robin, each process gets a small unit of CPU time (time quantum). After this time has elapsed, the process is preempted and a new one starts to be executed. Later, the preempted process will be re-executed from where it stopped[1].
One characteristic that processes have is priority, which means that a process is more likely to be executed by the system than others. Every time a process with higher priority than the current performed one is waiting to be executed, it will preempt the process in progress and the system will start its execution.

PREVIOUS WORK
There are several ways of lowering a system power consumption, which can be done on a system level, behavior level, register-transfer level, logic level transistor level and layout level[2][3]. The steps required to reduce system power consumption are necessarily dependent on the specific RTOS (Real Time Operating System) and

processor being used. It was observed that the RTOS, itself, can consume a significant amount of power[2]. So it is important to find ways of measuring a system energy consumption. But measuring it is a complicated task, because there is not only one method, and each has their advantages and limitations.

One option is using proprietary framework given by the manufacturer[4], like Simics[5]. It helps because they are normally full-system simulators that run unchanged production binaries of the target hardware at high-performance speeds.

Also, using special development boards[6] or hardware are another solutions, like measuring power consumption directly from the circuit using oscilloscopes during the tests performances. Other examples are connecting the circuit to data acquisition boards [7] and developing special circuit to measure the current flow between an energy plug and a computer power plug[8].

Moreover, a framework for designing operating systems with power consumption constraints was done[9], modeling power states using UML diagrams. It was advocated that using it since the beginning of the system design would make the task of measuring and controlling power consumption over the system would be significantly simplified.

As well, simulators can be written to compare different techniques of lowering power consumption, using languages such as C++[8].

All those previous solutions rely on, directly or indirectly, designing a full hardware-software system to measure power or purchase some copyrighted product. But several tools are available to do some sort of power measurement on various levels, and most of them are freeware.

On a more circuitry level, Proteus[10] couldn't be used to perform any test, because is incapable of measuring the power supply current to the microcontroller. Other possibility was KTechLab[11] for Linux systems, but is no longer supported.

Other general purpose simulators like Piklab[12], SkyEye[13], gprof[14], gem5[15], Pikdev[16] are able to simulate a system on instruction level, but not at a resource level, which makes the software not able to measure power consumption. On the other hand, other simulators like armware[17], PowerScope[18], SimpleScalar[19], XIOSim[20], RVDS (Real View Development Suite)[21] and softgun[22] are hardware dependent, operating system dependent or are outdated projects, not allowing an accurate power consumption report for different systems.

Then there exist some specific simulators for power consumption, like Wattch[23], being a framework for SimpleScalar[19], but it is outdated project, not offering accurate measurements for nowadays technologies. energyAware[24] allows debugging and measuring energy during the system execution, allowing it to be paused and continue without affecting the report. But this software depends on having a specific development board connected to a computer. Finally, others like EPIC Explorer[25] and SimPower[26] are no longer supported projects.

METHODOLOGY

For this project, the first goal was to find a power measurement tool that could accurately give a report of total energy used by an application. But shouldn't be limited to user space level, otherwise many kernel tasks wouldn't be evaluated, and is desired to look the details of the system as a whole when the scheduling policy is changed.

Then, MARSSx86 (Micro-ARchitectural and System Simulator for x86-based Systems)[27] was used. It is a tool for cycle accurate full system simulation of the x86-64 architecture. It is integrated into the QEMU (Quick EMUlator)[28] full system emulation environment.

But MARSSx86 was capable of giving indirect measurements of the power consumption, more precisely, the system resources usage. This means that such report should be translated into power consumption, human-readable format (e.g. power in Watts). This task was accomplished by McPAT (MultiCore Power, Area, and Timing modeling framework)[29], being an integrated power, area and timing

framework for multicore architectures. Using the resource usage report given by MARSSx86, it displays a complete and accurate power consumption report for the whole system and separated by different system parts, like core, memory, network-on-chip, etc.
Finally, a RTOS should be used to test different scheduling policies and be the operating system which would have real time applications implemented. For this task, Ubuntu Linux was used, but without many common desktop services installed, like GUI. For the Ubuntu, the kernel was recompiled to another version and applied a real time patch, allowing hard real time applications. This means that by the time a hard real time process is spawned, the system should listen to this requisition as fast as possible, with minimum overhead.

1. MARSSx86

It is based on QEMU and PTLSim, being the latter a cycle-accurate simulation model for out-of-order (OOO) x86 CPUs, modeling the decomposition of x86 instructions into RISC-like µops (micro-operations) and using basic block buffers to form traces of x86 µops, as in many real x86 implementations[27].
QEMU provides variety of modules and a flexible framework that allows MARSS to realize a simulation environment where users can replace a device emulation module with a simulation module, like full system emulation (full platform virtualization), which means it emulates a full system, including a processor and various peripherals[28].. Further, QEMU does not use any special hardware or host kernel support to achieve full system emulation capabilities. MARSS thus realizes a very flexible simulation environment that enables users to study and modify any part of the virtual system without the need for specialized hardware support or host kernel modifications.
MARSS includes detailed, cycle-accurate models of a contemporary memory hierarchy for single-core and multicore processor chips, including coherent caches and a DRAM memory system.

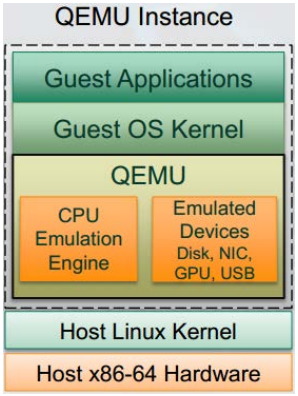

Figure 1: QEMU emulator operating on the system layers.

MARSSx86 supports seamless switching between the cycle-accurate simulation mode and the native x86 emulation mode of QEMU, permitting the fast-forwarding of simulation in the emulation mode to a region of interest where cycle-accurate simulation is needed.
MARSS's statistics framework allows users to collect detailed architectural performance statistics of specific regions of the simulated software, including kernel-level executions, in a single run.

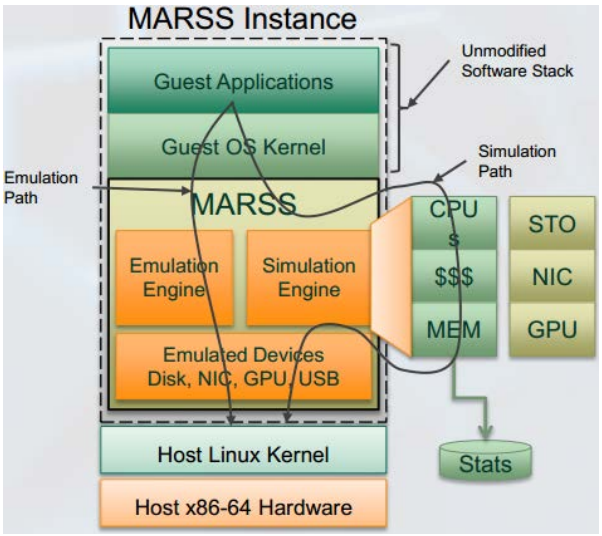

Figure 2: MARSS operating on the system layers. It is based on QEMU, and changed the necessary parts in order to give accurate simulation.

QEMU would emulate regular pipeline execution of each instruction, but MARSS analyzes the results during the execute and commit steps, as showed on Figure 3.

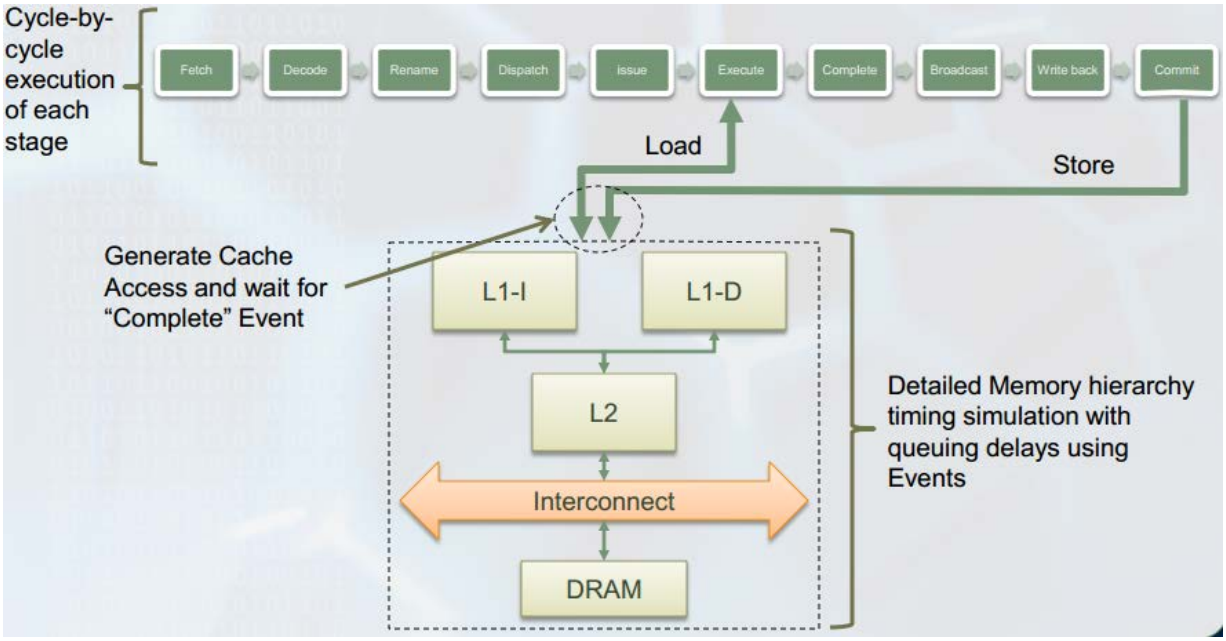

Figure 3: MARSS operating on the pipeline steps of an instruction execution

One more important detail is that, to emulate the hard drive QEMU emulates standard IDE controller and uses various types of disk image formats to store the data. For storing Virtual Machine disk data, QEMU uses raw or copy-on-write disk images. In MARSS, it is utilized QEMU's copy-on-write disk format, called 'qcow2' to create and store the snapshots of the VM machine and use them as checkpoints for simulation starting points.

Finally, QEMU (Emulation) does not maintain 'clock'. Simulation engine keeps track of number of executed cycles.

2. McPAT

At the microarchitectural level, McPAT includes models for the fundamental components of a chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking[29].

In order to generate a power consumption report, McPAT has a flexible XML interface to facilitate its use with many performance simulators, being able to measure energy for different architectures, as shown in Figure 4.

Before calling McPAT to compute runtime power numbers, the performance simulator needs to pass the statistics, namely, the activity factors of each individual component to McPAT via the XML interface file.

As shown in Equation 1, the activity factor of a component is the product of access

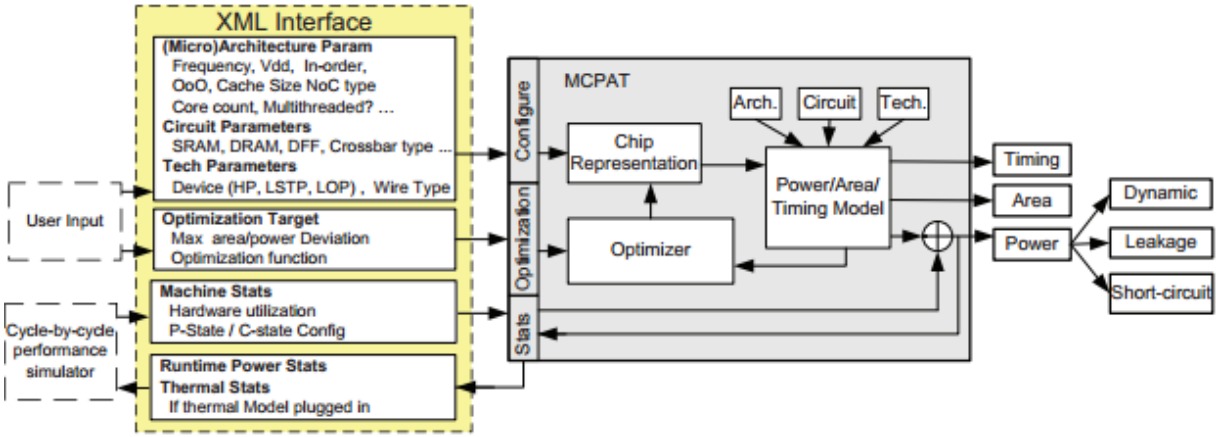

Figure 4: McPAT operation diagram

count of the component and the average hamming distance of all accesses for the time interval. By changing the time interval, one can change the granularity of runtime power consumption. If the performance simulator calls McPAT for runtime power computation every cycle, a cycle accurate power consumption profile will be generated, which will be useful to study real time power spikes.
It is important to highlight that, if the performance simulator calls McPAT for runtime power computation after power simulation is complete, an averaged power profile will be generated. This is the scenario for this study.

$$ActivityFactor = (AccessCount * ((\sum_{i=1}^{n} HammingDistance)/n))/n$$

Figure 5: McPAT activity factor calculation

In Equation 1, `n` is the cycle count for a given simulation period, `AccessCount` is the number of accesses to a specific component during the period, and the `HammingDistance` is the total number of flipped bits for two consecutive accesses. If the performance simulator cannot track the Hamming Distance, McPAT assumes that all bits are flipped per cycle.
For this study, is important to check only power consumption related to the system execution, so any technology factors shouldn't be taken into account. This means that leakage or short-circuit effects are not going to be taken into consideration, only dynamic power.

$$P_{dynamic} = \alpha \, x \, C \, x \, V_{dd} \, x \, \Delta V \, x \, f_{clk}$$

Figure 6: dynamic power equation used by McPAT

Dynamic power is the one that is spent in charging and discharging the capacitive loads when the circuit switch state, where C is the total load capacitance, V_{dd} is the supply voltage, ΔV is the voltage swing during switching, and f_{clk} is the clock frequency. C depends on the circuit design and layout of each Integrated Circuit component; we calculate it using analytical models for regular structures such as memory arrays and wires, along with empirical models for random logic structures such as ALUs. The activity factor α indicates the fraction of total circuit capacitance being charged during a clock cycle. We calculate α using access statistics from architectural simulation together with circuit properties.

3. Real Time Linux

In order to provide a real time environment for the tests, Ubuntu 11.04 'Natty', running Linux kernel 3.8.6 version. This pristine version was used in order to compare power measurements.
Concurrently, the same OS version was used, but also a real time patch was applied, giving hard real time capabilities to it. This patch implements the Completely Fair Scheduler (CFS)[30]. This project is a complete rewrite of the Linux task scheduler.
After applying the patch and recompiling, real time applications could have a hard real time aspect (previous applications on non-hard real time systems still work after the patch is applied. The difference is that applications will be executed with better responsiveness)[31].
For such, a simple "hello world" in C language now have the appearance as shown in Code 1.
The CFS scheduler provides 5 different policies, which 2 of them are real time. Differently from a global system scheduler, each process receives a policy, so this means that a process A can be round-robin, and process B be FCFS. During execution, if A comes first, it will be executed until its time quantum expires, then B will be fully executed, then A returns and starts being executed from the same point as it was.
For this study, both real time policies were used: FIFO (first-in, first-out policy) and RR (round-robin policy). Also, OTHER (standard round-robin time-sharing policy) was used for the pristine kernel, which is a non-real-time policy[32].

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <sys/mman.h>
#include <string.h>

#define MY_PRIORITY (49) /* we use
49 as the PRREMPT_RT use 50 as the
priority of kernel task lets and in-
terrupt handler by default */

#define MAX_SAFE_STACK (8*1024) /*
The maximum stack size which is
guaranteed safe to access without
faulting */

void stack_prefault(void) {
  unsigned char dum-
my[MAX_SAFE_STACK];
  memset(dummy, 0, MAX_SAFE_STACK);
  return;
}

int main(int argc, char* argv[]){
  struct sched_param param;

  /* Declare ourself as a real time
task */
  param.sched_priority =
MY_PRIORITY;
  if(sched_setscheduler(0,
SCHED_FIFO, &param) == -1) {
    perror("sched_setscheduler
failed");
    exit(-1);
  }

  /* Lock memory */
 if(mlockall(MCL_CURRENT|MCL_FUTURE)
== -1) {
   perror("mlockall failed");
   exit(-2);
 }

 /* Pre-fault our stack */
 stack_prefault();

  printf(“Hello World!”);
}
```

Code 1: a simple ‘Hello World’ real time application

MEASUREMENTS

The following steps were done (in that order):

- Execute Linux on MARSSx86
- Run simulation for 5 processes
- Convert the report generated into XML format for McPAT
- Generate McPAT report

As stated before, two versions of Linux were used: pristine and hard-real time. For both, the same application was written to sum a sequence of numbers. Each program spawns 5 different processes to do that sum, and each process will have a determined priority and scheduling policy. For comparison purposes, also a test executing 5 programs in a row was done, without spawning processes in parallel.
4 different test batteries were done: sum 100 first numbers (from 0 to 99), sum 1,000 numbers, sum 10,000 numbers and sum 100,000 numbers. Each battery was done 5 times.
Each of those batteries had a different scheduling policy: FIFO and RR (for the real time kernel) and OTHER (for the pristine kernel).
All the simulations were run in single core mode.

After the tests were finished, a Python script provided by MARSS developers was used to convert the generated report into a McPAT XML format. In order to do it, an architecture should be chosen among the ones available. Because MARSS accurately simulate x86 architecture, Penryn was selected, being a considerably recent architecture, which allows McPAT to give measurements coherent with current technology.

Then the report was generated using McPAT. Although it gives a detailed power consumption, only the dynamic power consumption for the overall system was compared.

RESULTS

After running all those tests on MARSSx86 and generating the report using McPAT, the following numbers were obtained, as well as those charts:

DESKTOP (PRISTINE) UBUNTU – "OTHER"										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	9.8863	9.7311	9.8524	9.8134	9.7625	9.8091	0.0567	0.0254	9.7594	9.8589
Parallel	9.5965	9.5509	9.4992	9.5269	9.5357	9.5418	0.0321	0.0143	9.5137	9.5700

REAL TIME UBUNTU – FIFO										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.5489	11.5070	11.5265	11.5996	11.5404	11.5445	0.0310	0.0139	11.5173	11.5716
Parallel	11.3269	11.3160	11.3836	11.3319	11.3243	11.3365	0.0241	0.0108	11.3154	11.3577
Priority	11.3900	11.3140	11.3314	11.3227	11.3123	11.3341	0.0288	0.0129	11.3089	11.3593

REAL TIME UBUNTU – RR										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.6208	11.6314	11.6289	11.5735	11.6260	11.6161	0.0216	0.0097	11.5972	11.6351
Parallel	11.3629	11.4490	11.3260	11.3429	11.3739	11.3709	0.0424	0.0189	11.3338	11.4081
Priority	11.3207	11.3520	11.3837	11.3663	11.3047	11.3455	0.0290	0.0130	11.3200	11.3709

Figure 7: Power Consumption (in W) for summing 100 numbers

DESKTOP (PRISTINE) UBUNTU – "OTHER"										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	9.7827	9.8504	9.7578	9.8627	9.7671	9.8041	0.0437	0.0195	9.7658	9.8424
Parallel	9.5309	9.5034	9.6223	9.5079	9.5389	9.5407	0.0430	0.0192	9.5030	9.5784

REAL TIME UBUNTU – FIFO										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.5826	11.6039	11.5298	11.5350	11.5310	11.5565	0.0308	0.0138	11.5294	11.5835
Parallel	11.3913	11.4671	11.3123	11.3555	11.3030	11.3658	0.0597	0.0267	11.3135	11.4182
Priority	11.2885	11.4024	11.3494	11.3243	11.3337	11.3397	0.0372	0.0166	11.3071	11.3723

REAL TIME UBUNTU – RR										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.6252	11.6904	11.5677	11.5978	11.4941	11.5950	0.0647	0.0289	11.5383	11.6518
Parallel	11.3085	11.3124	11.2903	11.3149	11.2755	11.3003	0.0151	0.0068	11.2871	11.3136
Priority	11.2953	11.3138	11.2750	11.3235	11.4154	11.3246	0.0483	0.0216	11.2822	11.3670

Figure 8: Power Consumption (in W) for summing 1,000 numbers

DESKTOP (PRISTINE) UBUNTU – "OTHER"										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	9.6776	9.6203	9.7401	9.7569	9.5962	9.6782	0.0634	0.0284	9.6227	9.7338
Parallel	9.4956	9.4218	9.4962	9.3640	9.4881	9.4531	0.0525	0.0235	9.4071	9.4992

REAL TIME UBUNTU – FIFO										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.4494	11.4719	11.4538	11.4901	11.4576	11.4646	0.0148	0.0066	11.4516	11.4776
Parallel	11.3049	11.2328	11.2830	11.2815	11.1915	11.2587	0.0411	0.0184	11.2227	11.2947
Priority	11.2339	11.1669	11.2548	11.2644	11.1889	11.2218	0.0378	0.0169	11.1886	11.2549

REAL TIME UBUNTU – RR										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	11.4345	11.4904	11.5005	11.4583	11.5117	11.4791	0.0285	0.0128	11.4541	11.5041
Parallel	11.2220	11.2331	11.3901	11.1725	11.2936	11.2623	0.0746	0.0334	11.1968	11.3277
Priority	11.2538	11.3558	11.2235	11.2263	11.2593	11.2637	0.0482	0.0216	11.2215	11.3060

Figure 9: Power Consumption (in W) for summing 10,000 numbers

DESKTOP (PRISTINE) UBUNTU – "OTHER"										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	9.0615	9.0505	9.0566	8.9196	8.9415	9.0059	0.0620	0.0277	8.9515	9.0603
Parallel	8.5551	8.5752	8.4822	8.5249	8.5207	8.5316	0.0318	0.0142	8.5037	8.5595

REAL TIME UBUNTU – FIFO										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	10.8794	10.9287	10.9182	10.9517	10.9918	10.9340	0.0372	0.0166	10.9014	10.9666
Parallel	10.5102	10.5094	10.4324	10.4032	10.4310	10.4572	0.0442	0.0197	10.4185	10.4959
Priority	10.4293	10.4151	10.4318	10.4730	10.4146	10.4328	0.0213	0.0095	10.4141	10.4515

REAL TIME UBUNTU – RR										
	Test 1	Test 2	Test 3	Test 4	Test 5	Mean	Standard Deviation	Standard Error	Confidence Minimum	Confidence Maximum
Sequential	10.9019	11.0653	10.8909	10.9718	11.0036	10.9667	0.0649	0.0290	10.9098	11.0236
Parallel	10.4755	10.4648	10.3730	10.4958	10.4487	10.4516	0.0422	0.0189	10.4146	10.4885
Priority	10.4256	10.4435	10.4659	10.4750	10.4451	10.4510	0.0175	0.0078	10.4357	10.4664

Figure 10: Power Consumption (in W) for summing 100,000 numbers

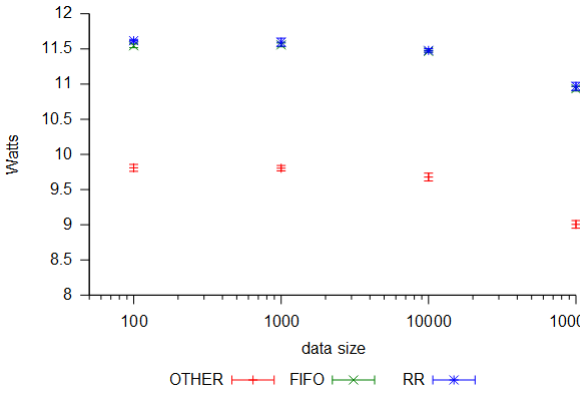

Figure 11: Simulation for 5 processes sequentially executed, using the pristine kernel

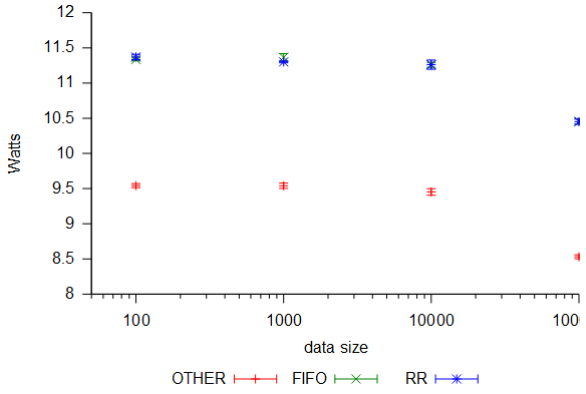

Figure 12: Simulation for 5 processes executed in parallel, using the pristine kernel

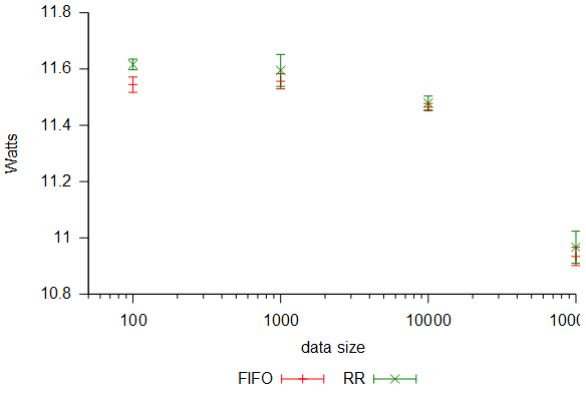

Figure 13: Simulation for 5 processes sequentially executed, using the real time kernel

First important detail stated previously is that McPAT calculate an averaged power, so as more computation is done, the numbers get reduced due to it (in other words, calculates power per computation).
Priority policy was applied only to the real time kernel, because preemption due to a higher priority process on a pristine kernel was not the goal of this study.

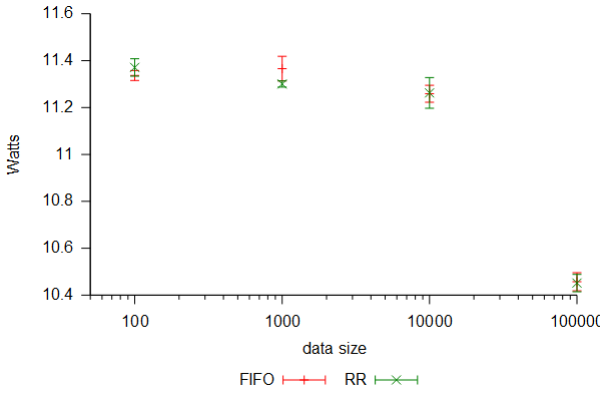

Figure 14: Simulation for 5 processes executed in parallel, using the real time kernel

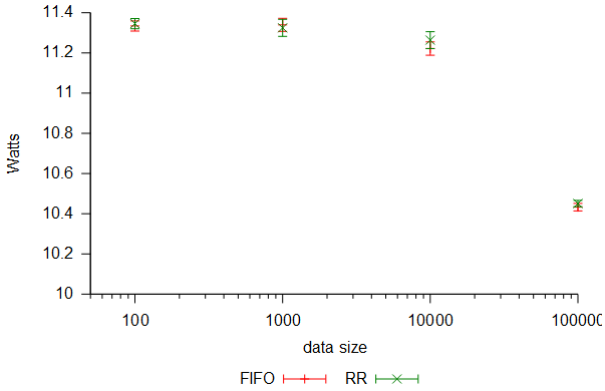

Figure 7: Simulation for 5 processes executed in parallel, applying different priorities, using the real time kernel

Instead, that's the feature for comparing power consumption between two different schedulers, as well as responsiveness, if desired.
It is clear that there's a trade-off for the system to offer quick response to a process (hard real time) and raising the system power consumption.

FUTURE WORK

This study was restricted to x86 Penryn architecture, so the same tests could be executed for other types of processors. But using the same idea as here (software simulator), or a simulator should be implemented from scratch, or would be necessary to find a simulator for other architectures.
Despite this, the tools used weren't designed to check process context switching, so responsiveness couldn't be really checked. But, in case a metric is necessary using those tools, MARSSx86 is able to display

the total number of cycles spent on a program execution, which can be converted into execution time. Also, as more cycles are necessary for a same workload means that context switching overhead is differentiated.

CONCLUSION

For the purpose of this study, MARSSx86 and McPAT were able to provide good simulations for a real time kernel and calculate power consumption for the whole system (user and kernel space). Besides the interface limitations and narrowed options of architectures to be tested, those results can be used to suppose for other similar designs.

Also, due to the standard scheduling policies provided by the Linux real time kernel, only FCFS and round-robin were used. If other schedulers were tested, they should be implemented into the kernel code, and them recompile the whole system. But recompiling the system doesn't interfere into the executable application file, so it should work as expected, but with different power consumption values and responsiveness[31].

Moreover, as expected, different scheduling policies imply into different power consumption values. Both FIFO and RR have really close results, so choosing between those schedules can be more a matter of system design.

REFERENCES

[1] SILBERSCHATZ, A. GALVIN, P. B. GAGNE, G. Operating System Concepts. Ninth Edition. 2012

[2] DICK, R. P. Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems. 2002.

[3] KOURTELLIS, N. Tackling Power Dissipation in Embedded Systems at the Power Supply Level. 2006.

[4] DICK, R. P. LAKSHMINARAYANA, G. RAGHUNATHAN, A. JHA, N. K. Power Analysis of Embedded Operating Systems. 2003.

[5] WINDRIVER. Wind River Simics: Full System Simulation. Details about this software can be seen at http://www.windriver.com/products/simics/

[6] CHEN, L. B. CHEN, Y. L. HUANG, I. J. A Real-Time Power Analysis Platform for Power-Aware Embedded System Development. 2001.

[7] ACQUAVIVA, A. BENINI, L. RICCÓ, B. Energy Characterization of Embedded Real-Time Operating Systems. 2001.

[8] PILLAI, P. SHIN, K. G. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. 2001.

[9] AGARWAL, A. FERNANDEZ, E. System Level Power Management for Embedded RTOS: an Object Oriented Approach. 2009.

[10] Labcenter Electronics. Proteus Virtual System Modeling (VSM). Details about this software can be seen at http://www.labcenter.com/products/vsm/vsm_overview.cfm

[11] PADRAH, Z. GRIMES, A. LUCAS, J. KTechLab: IDE for microcontrollers and electronics

[12] HADACEK, N. LAOÛT, S. GIBAUD, A. LANDAMORE S. FRANKLIN C. Piklab: IDE for applications Microchip PIC-based microcontrollers

[13] KANG, S. H. SkyEye: a very fast full system simulator

[14] FENLASON, J. STALLMAN, R. GNU gprof: The GNU Profiler

[15] BINKERT, N. et al. The gem5 Simulator

[16] GIBAUD, A. PiKdev, An IDE for the development of PIC based applications under KDE

[17] WEI, H. armware: This is an ARM emulator

[18] FLINN, J. SATYANARAYANAN, M. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications

[19] AUSTIN, T. M. SimpleScalar Tool Suite

[20] KANEV, S. WEI, G. BROOKS, D. XIOSim: Power-Performance Modeling of Mobile x86 Cores

[21] ARM Holdings. RVDS (Real View Development Suite): ARM Profiler Non-Intrusive Performance Analysis

[22] KARRER, J. Softgun - The embedded system simulator

[23] BROOKS, D. TIWARI, V. MARTONOSI, M. Wattch: A Framework for Architectural-Level Power Analysis and

Optimization
[24] Energy Micro AS. energyAware Profiler.
[25] ASCIA, G. CATANIA, V. PALESI, M. PATTI, D. EPIC Explorer: A Platform for Design Space Exploration of VLIW Architectures
[26] MDL group at the Penn State University. SimPower: cycle-accurate RT level energy estimation tool
[27] PATEL, A. AFRAM, F. CHEN, S. GHOSE, K. MARSSx86: A Full System Simulator for x86 CPUs. 2011
[28] QEMU: generic and open source machine emulator and virtualizer. Available at http://wiki.qemu.org/Main_Page
[29] LI, S. AHN, J. H. BROCKMAN, J. B. JOUPPI, N. P. McPAT 1.0: An Integrated Power, Area, and Timing Framework for Multicore Architectures. 2010.
[30] MOLNAR, I. GLEIXNER, T. [patch] Modular Scheduler Core and Completely Fair Scheduler [CFS]. 2007. Group email can be seen at http://lwn.net/Articles/230501/
[31] TS'O, T. HART, D. KACUR, J. The Real-Time Linux Wiki. 2010. Can be seen at https://rt.wiki.kernel.org/
[32] Linux Programmer's Manual. SCHED_SETSCHEDULER manpage.