

Modeling the Energy Consumption of Page Fault for Embedded Linux

Mickael Lanoë, Jérémie Guillot, and Eric Senn
Lab-STICC, Université de Bretagne Sud, France
eric.senn@univ-ubs.fr

Abstract

While operating systems are now largely used in embedded system design, their energy consumption is far from negligible. Being able to determine the part of this consumption in the system's overall energy budget is therefore essential. This paper proposes a methodology to model the power and energy consumption of virtual memory management mechanisms in complex operating systems. Of course, this work is only a part of a bigger project in which all the consuming components in embedded systems are considered. The virtual memory subsystem of a complete and recent Linux is studied here. A method is proposed to generate different categories of page faults, and to model the incurred time and energy penalties for different page allocation strategies. The precision of the model is presented, and finally checked against actual measurements for an image processing application.

Keywords : Energy, Page Fault, Embedded Linux

1. Introduction

1.1. Context

Embedded systems have now to perform multiple different tasks, from control oriented (user interfaces, adaptation to the environment, quality of services, ...), to data intensive (multimedia, audio, image, video processing, streaming, communications,...). They run on complex hardware such as multi-cores, possibly heterogeneous, together with different hardware accelerators and peripherals. In such a context, the use of an operating system to provide for a cooperative and preemptive multitasking environment, and for hardware abstraction to ease the applications programming and portability, is compulsory. Among the many services run by the operating system, we are here particularly interested in the virtual memory subsystem, which is directly connected to one of the three major energy consumption sources in embedded systems: the memory [7]. Being able to estimate the energy consumption of embedded applications as earlier as possible in the design process is now recognized as a necessity; this is however far from simple as soon as an operating system is involved.

1.2. State of the art

Several studies have first studied impact of an operating system on the energy consumption without actually proposing models. [4] and [5] shown that the energy consumption of a system can rise from 6% to 50% with an OS, depending on the application, and that it further increases with the processor frequency and supply voltage. [9] shown that the OS can consume from 1% to 99% of the processor energy depending on the services called. Instead of relying on measurements, many other approaches perform simulations to determine the energy consumption of OS [6], [16], [10], [14]. Power estimation is performed at the instruction level, and inherits the drawbacks of micro-architectural level power analysis : cycle-level simulations may prove very time consuming for large programs; they necessitate a low-level description of the architecture which is often difficult to obtain for off-the-shelf processors. For simple processors, instruction based power models commonly introduce an error from 10 to 15% [15]. For those reasons, only simple processor architectures are considered in these works; real-time behaviors and peripherals (Ethernet, controllers, memories, I/O drivers ...) are not taken into account. [13] proposed to use FPGA accelerated simulation to speed-up the system simulation, but used coarse energy models from components spreadsheets. In [10] the power simulator SoftWatt [12] gives the power of the CPU, memory hierarchy, and low-power disk subsystem, however no other devices is considered. Energy

models similar to low-level power simulators are again used with the same drawbacks as described before. The authors have demonstrated that not considering the routines diversity but only a flat average for the OS gives an error up to 178%. They did not considered multithreaded applications.

1.3. Proposition

In the frame of the project Open-PEOPLE (Open Power and Energy Optimization Platform and Estimator) [2], we propose a methodology to model the consumption of embedded operating system services. Our methodology is based on physical measurements realized on standard reconfigurable boards. We will focus on approaches intending to assess the power and energy consumption of the Linux virtual memory subsystem. This work follows our former works on the modeling of OS and embedded systems energy consumption. Especially, we proposed models to take into account complete real-time embedded systems, including complex processors, reconfigurable components (FPGA), and dedicated OS services such as scheduling, context switching, or inter-process communications [8, 11].

The remainder of this paper is organized as follows. Section 2 describes the Linux virtual memory subsystem. In Section 3, the test bench is presented, with the hardware platform, and the methodology to generate page faults in the complete system. Section 4 presents the duration and power consumption measurements. Section 5 describes models extrapolation to estimate the time and energy consumption overhead due to page fault. They are finally used to estimate the virtual memory energy consumption overhead for a JPEG application and a comparison with the real consumption validates our approach. We conclude the paper in section 6.

2. Virtual Memory

The embedded system runs multiple processes each with its own address space. The virtual memory subsystem divides physical memory into blocks that it allocates to different processes. Application programs have the impression of contiguous memory, while in fact it may be physically fragmented. Virtual memory systems can be divided into two categories: those with fixed-size blocks, called pages, and those with variable-sized blocks, called segments. The pages have typical sizes ranging from 4 to 64 KB when the segment sizes vary. We study here a page-based virtual memory system. The virtual addresses are translated by a combination of hardware and software components into physical addresses used to access the main memory as shown in Figure 1. The hardware and software components both manage a page table used to store the mapping between virtual and physical addresses. If the address translation mechanism raises a page fault exception, the operating system searches for a page on the main memory or on the secondary storage depending on the state of the running application and then updates the page table. There are two types of page faults, major and minor faults. Minor page faults are handled without IO accesses. Major page faults are handled by means of IO activity. The operating system that we analyze is the Xilinx Open Source Linux (2.6.29). We also applied the RT-Preempt patch to convert the kernel into a fully preemptible kernel (<http://rt.wiki.kernel.org>) to ensure stable latency and therefore facilitate time measurements. The Linux virtual memory subsystem implements a multilevel page table with page size of 4KB. An area of memory consists of a number of contiguous pages. Pages in the linear process address space are not necessarily resident in memory. Linux has an on-demand page allocation policy. Allocations made in the process are just reserved within the area and are only allocated when the hardware raises a page fault exception. Figure 2 shows the virtual

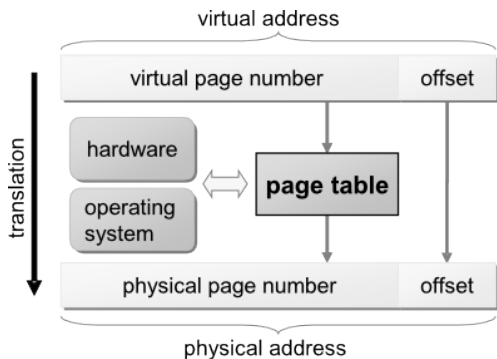


Figure 1: Virtual to physical address translation

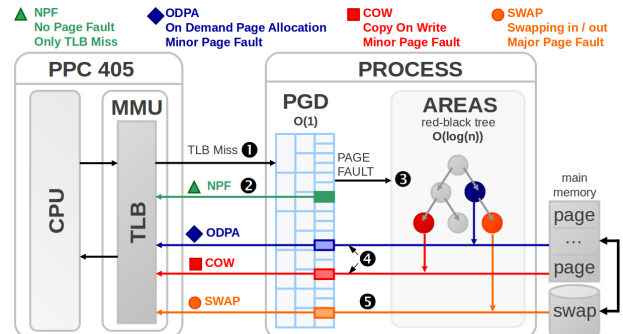


Figure 2: The Linux virtual memory subsystem

memory subsystem. When the processor raises a Translation Look-aside Buffer (TLB) miss exception ❶, the Linux kernel gets the physical address from the virtual address in its three level page table (Page Global Directory: PGD). If the entry is valid, the TLB is updated. In this case the algorithm complexity is $O(1)$ ❷. If the entry is invalid the kernel searches the memory area in a red-black tree ❸. The algorithm complexity is $O(\log(n))$. The page table is updated with the result and the process statistics are updated according to the page fault type; minor page fault ❹ or major page fault ❺. The Linux page swapping mechanism can swap code pages of an application to disk. It will take a long time to swap those pages back into the main memory. In a real-time system, the swap is avoided because it reduces the determinism of application latency. So the Linux page swapping mechanism is disabled in our study: there is no major page fault. The objective of the test in the next section is to measure the variation of the execution time and the power consumption according to the number of TLB misses ❷ and minor page faults ❹.

3. Test bench

We have performed power and energy consumption modeling measurements on an experimental platform composed of a XUP Virtex-II pro development board, a 256MB SDRAM and a compact Flash for the root file system. The embedded core (PowerPC 405), the memory management unit (MMU) and data and instruction caches was integrated into the same chip. The processor frequency was 300MHz and the bus frequency was 100MHz. The MMU performs address translation and protection functions. It divides storage into pages. The MMU uses the TLB for address translation. Each valid entry contains the virtual page number and its translation into a physical page number. The TLB contains 64 entries and is fully associative. There is no hardware support for page table management. The page-translation table is a software-defined data structure containing page translations. If a virtual address does not match an entry in the TLB, the CPU raises a TLB miss exception. The operating system provides the physical address from the page table and updates the TLB.

A power analyser sources and measures DC voltage and current into the XUP board. The processor core is powered by the 1.5V power supply, the SDRAM memory and FPGA I/O are powered by the 2.5V and physical peripheral devices are powered by the 3.3V. The test program do not use physical peripheral devices so we have conducted measurements on the processor core and SDRAM power supplies. The execution time and the number of page faults are respectively measured with the `clock_gettime()` and `getrusage()` POSIX functions. The methodology presented in the next sections is applied to the SDRAM power supply but can be also applied to the processor core power supply. Indeed, the approach is a simple and generic way to estimate the energy cost of page faults or TLB misses.

The test program writes to a buffer at various positions spaced as shown in Figure 3. For the same number of writes, the number of TLB misses increases with the spacing between two positions. The buffer is written to ask the kernel to allocate a new page for the first time. In order to have TLB misses, the buffer size must be greater than the TLB size. In our case, the TLB has 64 entries and Linux uses 4KB per page so the minimum buffer size is 256KB. Note that this size is greater than the processor data cache size (16KB). The number of iterations must be high enough to reach the number of page faults requested by the test configuration. The number of iterations must be constant for all tests to have the same number of executed instructions. The value must be greater or equal to the buffer size divided by the cache line size. The buffer size and the number of iterations are set at the beginning of the program. The offset is variable and is a multiple of the cache line size, up to the page size in order to be always in a cache miss situation (Figure 3). The PowerPC has a data cache line size of 32 bytes and the page size is 4KB so there is 128 values for the offset (32 to 4096). At the end of each test execution, we verify that there is no context switch which could distort the results. Finally there are 128 tests. Let us define `nwrite` (number of writes) to be equals to the number of iterations. The first test results in `nwrite/128` TLB misses, and the last results in `nwrite` TLB misses. Figure 4 shows the program pseudo code.

The test program is run under three different scenarios. The first scenario does not contain any page fault (NPF: *No Page Fault*). It allows studying the variation of execution time and power consumption according to the number of TLB misses. In this scenario, a first loop on the buffer allocates all the pages before running the test which therefore avoids any page faults. The second scenario generates 1024 page faults, the buffer being allocated on-demand by the Linux's page allocation strategy (ODPA: *On-Demand Page Allocation*). Pages are only allocated on the first access which generates the page fault in the test loop. The third scenario also generates 1024 page faults, it uses the Linux *copy on write strategy* (COW).

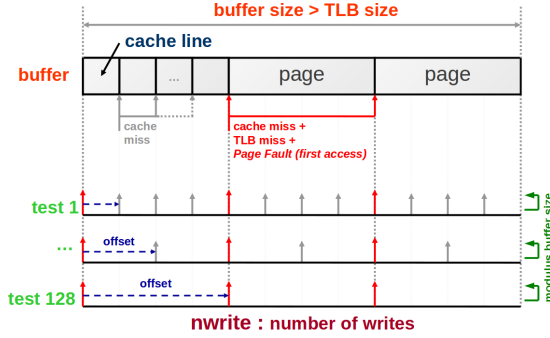


Figure 3: Writes to the buffer

```

for test = 1 to 128
  offset = test * cache.line.size
  pos = 0
  begin measurement (time, power)
    for i = 1 to nwrite
      buffer[ pos ] = data
      pos = pos + offset
      pos = pos modulus buffer.size
    end for
  end measurement
end for

```

Figure 4: Test program pseudo code

Two process shares the buffer until a buffer write occurs. A parent process allocates and initializes a buffer. It then creates a child process that writes to this buffer. In the COW strategy, allocation and page copy are delayed to the first writing and is therefore more expensive than ODP.

4. Results

Figure 5 shows measured tests durations for each scenario. For the first scenario (NPF), the variation of the execution time according to the number of TLB misses is approximated to: $T(s) = 149 \cdot 10^{-9} \cdot N_{TLBmisses} + 0,080$. This linear equation exhibits a very good regression coefficient ($R^2 = 0,999$). Its slope is the duration of one TLB miss, here 149 ns. Figure 6 shows the duration of one page fault for the second and third scenario (ODPA and COW). This value is obtained by subtracting the time without page fault (NPF) from the measured time, and by dividing the result by the number of page faults (1024). We verify here that the page fault duration is almost stable; about 51 μs for ODP and 114 μs for the COW strategy.

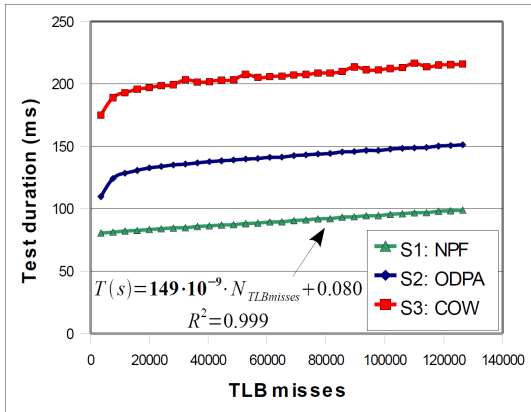


Figure 5: Test duration vs the number of TLB misses for each scenario

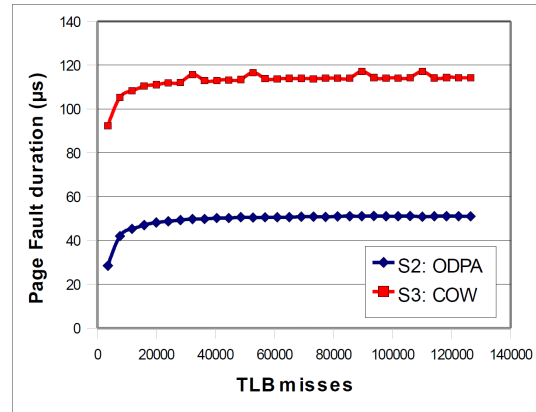


Figure 6: Page Fault duration vs the number of TLB misses for two scenarios

Figure 7 shows the power consumption measured for each test. We easily determine the power consumption during a page fault from the energy consumption during the test. Indeed, the final energy is the sum of the energy consumed by the loop, the energy consumed during TLB misses and the energy consumed during page faults.

Figure 8 shows the energy consumption as a function of the number of TLB misses. For the first scenario (NPF), the slope of the line is the energy of one TLB miss, here 396 nJ, which gives 2658 mW for the power consumption. The corresponding equation is (with again an excellent $R^2 = 0,999$): $E(J) = 396 \cdot 10^{-9} \cdot N_{TLBmisses} + 0,243$.

To compute the energy of one page fault for the ODP and COW scheme, we subtract the total energy from the energy computed without page fault (NPF) and divide the result by 1024 (i.e. number of page faults). The energy consumption is almost stable for each strategy : about 166 μJ and 340 μJ respectively for the ODP and COW strategies. The page fault power consumption is then about 3263 mW for the

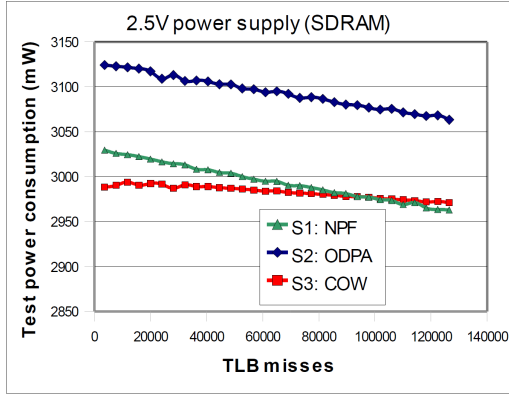


Figure 7: Test power consumption vs the number of TLB Misses for each scenario

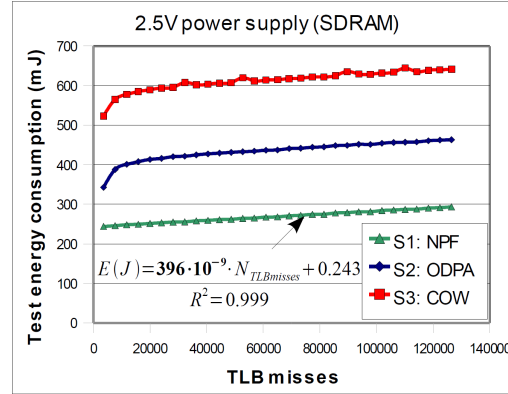


Figure 8: Test energy consumption vs the number of TLB misses for each scenario

Table 1: Duration, power and energy of one TLB miss and of one page fault according to the allocation strategy

MODEL		TLB miss		PAGE FAULT	
		NPF	ODPA	COW	
2.5V (SDRAM)	Duration	T (μ s)	0.149	51	114
		P (mW)	2658	3263	2976
		E (μ J)	0.396	166	340
1.5V (CPU CORE)	P (mW)	691	510	518	
	E (μ J)	0.103	26	59	

Table 2: Duration, power and energy of the JPEG application according to the allocation strategy

JPEG COMPRESSION ALGORITHM		PAGE FAULT			
		NPF	ODPA	COW	
Number of Page Fault		0	52	60	
2.5V (SDRAM)	Duration	T (μ s)	31762	34335	38853
		P (mW)	2909	2938	2921
1.5V (CPU CORE)	E (μ J)	92409	100864	113496	
	P (mW)	538	536	533	
	E (μ J)	17099	18408	20704	

ODPA strategy and 2976 mW for COW strategy. Table 1 summarizes the durations and the consumptions of one TLB miss and one page fault according to the allocation strategy used in the application. Current measurements on the SDRAM power supply (2.5V) were presented, but the methodology was also applied to measure the current from the processor power supply (1.5V). Table 1 depicts those two power consumptions. The energy cost of one page fault is at least 400 times higher than the energy cost of one TLB miss for the SDRAM power supply and at least 250 times higher for the processor power supply. Therefore the energy cost of TLB misses may be ignored in our estimation.

5. Model Validation

Our model was used to estimate the energy consumption overhead due to page faults during the execution of a JPEG encoder (lossy image compression algorithm, image size = 256 x 256). The application first reads a bitmap file from the compact flash. Then, it performs the compression and finally saves the jpeg file to the compact flash. In our study, we only care about the compression step. The power consumption of the compression is due to cache misses and page faults. The compression is run separately under NPF, ODPa and COW strategies. For each scenario the power consumption is measured. Table 2 shows the results for the three scenarios. The differences between measurements from the first scenario, and from two other scenarios are compared to the estimated energy cost due to page faults. To compute the error rate between the model and measurements we use the following average error metric: $\frac{|E-M|}{M}$ where E is the estimation and M is the measurement. As we can see in Table 3, the maximum error for the time estimation is 4% and the maximum error for the power estimation is 2%.

Table 3: Error rate of the estimation for the JPEG application according to the allocation strategy

MODEL PRECISION		MODEL		JPEG		ERROR	
		ODPA	COW	ODPA	COW	ODPA	COW
Duration	T/PF (μ s)	51	114	49	118	4%	4%
2.5V power supply	P (mW)	3263	2976	3286	2974	1%	0%
	E/PF (μ J)	166	340	163	351	2%	3%
1.5V power supply	P (mW)	510	518	509	508	0%	2%
	E/PF (μ J)	26	59	25	60	4%	2%

6. Conclusion

Virtual memory mechanisms affect the performance of an operating system, and more generally of the applications it supports. We have presented a methodology for modeling the power and energy consumption overhead due to virtual memory management in a complex operating system. We have studied the behavior of the Linux virtual memory subsystem, and its relation with the processor's translation look-aside buffer (TLB). We then build a model from measurements for different allocation strategies.

Our methodology appears quite efficient since the maximum error introduced by our model is less than 5%. In the JPEG application example, the maximum error between the estimated energy consumption due to page faults and the measurements is 4%.

It may be completed however by studying additional operating system mechanisms, like page swapping (swapping pages to disk causes major page faults in the system), the call stack resizing strategy, or the influence of the red-black tree size (used for memory region management). The model presented in this paper will be added to our Consumption Analysis Toolbox [1], which is integrated in the Open Source AADL Tool Environment [3] dedicated to model driven design for embedded systems. Also, it will be finally available in the Open-PEOPLE platform currently in the first phase of its development [2].

Bibliography

1. CAT - Consumption Analysis Toolbox. <http://sourceforge.net/apps/trac/lab-sticc/>.
2. The Open PEOPLE Project Website. <http://www.open-people.fr/>.
3. The Open Source AADL Tool Environment (OSATE). <http://www.aadl.info/>.
4. A. Acquaviva, L. Benini, et B. Ricco. Energy characterization of embedded real-time operating systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP01)*.
5. K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, et B. Jacob. The performance and energy consumption of three embedded real-time operating systems. In *Proceedings of CASES01*, Atlanta, Georgia, USA, November 2001.
6. K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, et B. Jacob. The performance and energy consumption of embedded real-time operating systems. *IEEE Transactions on Computers*, 11(52):1454–1469, November 2003.
7. Luca Benini et Giovanni de Micheli. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TECS)*, 5 no 2:115–192, 2000.
8. S. Dhouib, E. Senn, J-P. Diguët, D. Blouin, et J. Laurent. Energy and power consumption estimation for embedded applications and operating systems. *Journal of Low Power Electronics (JOLPE)*, 2009.
9. R. P. Dick, G. Lakshminarayana, A. Raghunathan, et N. K. Jha. Power analysis of embedded operating systems. In *Proceedings of the IEEE 37th Design Automation Conference (DAC-00)*, 2000.
10. Tao Li et Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 160–171, 2003.
11. J.P. Diguët S. Dhouib, E. Senn. Model driven high-level power estimation of embedded operating systems communication and synchronization services. In *Proceedings of the 6th IEEE International Conference on Embedded Software and Systems*, China, May 25-27 2009.
12. A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, et L. K. John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2002.
13. Dam Sunwoo, H. Al-Sukhni, J. Holt, et D. Chiou. Early models for system-level power estimation. In *Eighth International Workshop on Microprocessor Test and Verification, MTV07*, December 2007.
14. T.K. Tan, A. Raghunathan, et N.K. Jha. Energy macromodelling of embedded operating systems. *ACM Transactions on Embedded Computing Systems*, 4(1):231–254, February 2005.
15. Vivek Tiwari, Sharad Malik, Andrew Wolfe, et Mike Tien chien Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 13:1–18, 1996.
16. Xia Zhao, Yao Guo, Hua Wang, et Xiangqun Chen. Fine-grained energy estimation and optimization of embedded operating systems. In *International Conference on Embedded Software and Systems Symposia, ICESSE Symposia '08*, pages 90–95, July 2008.