Power Efficiency Study of Multi-threading Applications for Multi-core Mobile Systems

MARIUS MARCU, DACIAN TUDOR, SEBASTIAN FUICU,
SILVIA COPIL-CRISAN, FLORIN MATICU, MIHAI MICEA
Computer Science and Engineering Department
"Politehnica" University of Timisoara
Timisoara, Bd. V. Parvan, No. 2
ROMANIA
e-mail: marius.marcu@cs.upt.ro, dacian@cs.upt.ro, sebastian.fuicu@cs.upt.ro,
silvia_copil@yahoo.co.uk, maticu_florin@yahoo.com, micha@dsplabs.upt.ro

Abstract: - One constant in computing which is true also for mobile computing is the continue requirement for greater performance. Every performance advance in mobile processors leads to another level of greater performance demands from newest mobile applications. However, on battery powered devices performance is strictly limited by the battery capacity, therefore energy efficient applications and systems have to be developed. The power consumption problem of mobile systems is in general a very complex one and remained very actual for quite a long time. In this paper we aim to define a software execution framework for mobile systems in order to characterize the power consumption profile of multi-threading mobile applications. Study results for different thread libraries, multi-core processors and multithreaded parallelized applications are also presented.

Key-Words: - power consumption, multi-threading, multi-core, mobile applications, power profiling

1 Introduction

Today personal communication devices are more than voice call terminals. The evolution of portable and mobile computation systems towards an increased feature set as well as hardware and software requirements demands, together with the significant increase of market penetration in our modern society, is raising complex problems from a reasonable energy consumption level point of view under different usage scenarios [1, 14]. The computational demand of handheld mobile applications is continually rising, which for example have to process vast amounts of multimedia data or support multi-threading parallel applications. Unfortunately, the traditional approach of increasing computational power by steadily accelerating the processor clocks rated, cannot be pursued further as it would increase the power consumption by factors prohibitive for battery powered mobile devices. One way out of this dilemma is to distribute the computational load on multiple processor cores, because this architecture allows to reduce clock speeds and to minimize voltage supply, which in turn enhances power-efficiency. As such technology is not mature yet and chips are under design and not available on a broad audience for mobile devices, we propose a software approach on top of multi-core processors, by designing an open, flexible execution framework that minimizes the processor load and thus reducing the energy consumption.

One large representative of battery powered devices like the mobile handset is by its nature limited in battery capacity and thus does not fit for architectures with high power consumption. Thus the future mobile computing platforms for handsets will face a dramatic contradiction of increasing requirements on computational resources while keeping the power consumption at current levels or even decreasing it. Looking at the world of non-embedded personal or enterprise computing systems, this clearly shows a trend towards multi-core systems but with much less restrictions in the power consumption requirements. Other aspects differentiating the embedded communication systems from the enterprise computing systems are the requirements of hard real-time operations at least for the modem part of the system and the high level security [13, 15].

The power consumption problem of computing systems is in general a very complex one [1]

because each physical component from the system has its own consumption profile depending especially on the executed operation type. This means that together with the physical components, the software application layer has a big influence on the energy consumption [2].

Therefore, the main goal of our work is to design, implement and validate a software framework for power-aware mobile applications in order to reduce overall power consumption and increase the efficiency of the energy usage. The objective of our work for this paper is to create an application-framework that would allow the execution of different types of threads (by using multiple thread libraries for Windows OS), comparing their efficiency and measuring the power consumption on mobile devices. Using this framework we want to show how multithreading mobile applications influence the power consumption of the single-core/multi-core battery powered devices. The threads were created by use of Win32, Boost and Pthreads for Windows libraries.

2 Multithreading and multi-core mobile systems

2.1 Multi-core systems

Future handheld computing systems must bridge the contradiction between high computational resources and low power consumption. The continuous increasing market demanded functionality leads to a drastic increase of the cost factor of required computational hardware and software resources. As a result, designing a non-scalable and non-programmable hardware solution in order to meet the requirements is either very difficult to implement or prohibitively expensive. A programmable multi-core architecture should offer the optimal solution in terms of power consumption, performance, flexibility and cost. A multi-core architecture can be defined as an architecture consisting on multiple processing cores that are manufactured on the same integrated circuit. Most multi-core architectures have been driven by major market player like Intel, AMD, Sun or IBM as a primary solution to achieve higher processor performance and to overcome physical limitations like clock frequency and heat dissipation. The current situation in multi-core systems especially for handheld devices is in its infancy. According to the visions published in [8], it is believed that multi-core systems can supply a substantial system performance boost with reasonable power consumption.

One of the challenges of multi-core systems is task scheduling [9] [10]. The gap between multi-core architecture, scheduling algorithm and power consumption needs to be bridged especially for handled SMP multi-core systems. In the diversity of core packaging solutions and different operating systems running on multi-core systems, it is expected that task schedulers shall be architecture-aware. Closely related to the scheduling problem, investigations on heat dissipation and scheduling extensions toward temperature control and power consumption have been started for example in [11] and [12]. In spite of many efforts of both research and industrial communities on improving multi-core scheduling techniques from the energy efficiency point of view, there is little evidence on the convergence of different proposed solutions. Considering the vast number of possible systems and applications, we expect that power efficient task scheduling for handheld multi-core system to be one of the hot research topics for the next period.

While for years, ever-higher clock speeds granted that big application code would run faster, the rules are different for the multi-core processors of today. The problem is, that simply adding more cores to a microprocessor does not increase the speed or power of conventional application code. To gain the maximum performance for an application running on multi-core CPUs, application developers need to design their code for these new architectures [19]. New design patterns should be used for these applications, based on multiple execution threads exploiting problem concurrency, but power consumption must be also addressed for battery powered devices. Therefore we tried to design an energy efficient framework for multithreading mobile applications running on multi-core CPUs.

2.2 Multithreading libraries

Multithreading is a method of improving the execution performance of a process by the use of concurrency, that is allowing more than one thread to run independently of each other within that program. Since each thread could run on a different core at the same time, it is hoped that multithreading does not only improve efficiency of both single and multi-core processors, but it could also increase the battery life of mobile systems. When compared to the cost of creating and managing a process, a thread can be created with much less operating

system overhead and requires fewer system resources than managing processes.

There are three types of thread libraries [3]:

1. User-level libraries can use certain system calls or characteristics of the OS kernel, but their structures, code, and thread management are located in user-space. The kernel does not recognize and distribute individual threads to the processor, so all of them will run on a single core (e.g. Protothreads [4]).
2. Kernel-level libraries - are the ones that have very little code in user-space and use mostly system calls. These kind of libraries are very fast and simple, offering support for building other, more complex, structures on top (e.g. Win32 threads [5]).
3. Hybrid libraries – are created by building on top of a kernel library. The threads created and managed in user-space are mapped over one or more kernel level threads, by using system calls. BOOST [6] and POSIX Pthreads for Windows [7] are hybrid libraries, with 1:1 mapping ratio (one user-level thread mapped over one kernel level thread).

Win32 threads can be implemented through the system calls available in Win32 API. They are known as kernel-level threads, because the core of the OS is the only one managing them (creation, synchronisation, processor allocation). Each is identified by a block of data (ETHREAD block) residing in the system address space, together with the data structures it points to.

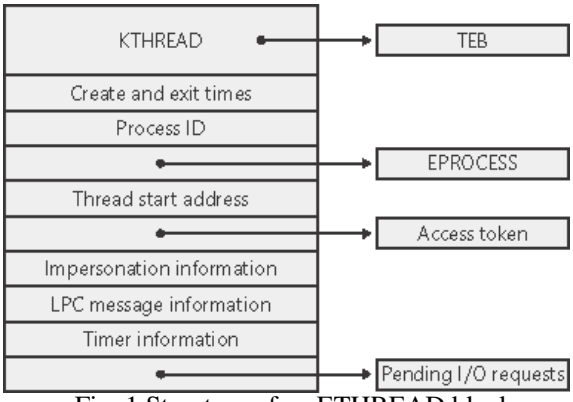

Fig. 1 Structure of an ETHREAD block

The KTHREAD block (Fig. 2) contains the information that the Windows kernel needs to access to perform thread scheduling and synchronization on behalf of running threads.

POSIX Pthreads for Win32 is an open-source thread library, written in POSIX 1003.1-2001 standard. It defines an API for multithreading applications, which can be informally grouped into three major classes: thread management, thread synchronization and condition variables.

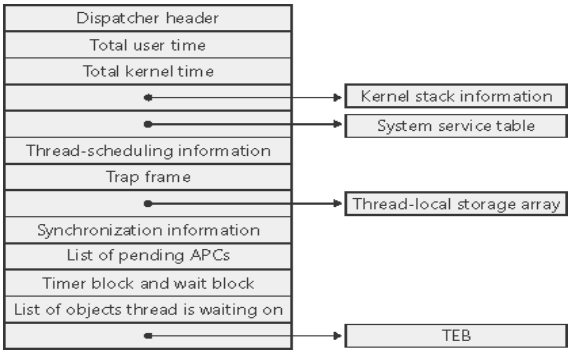

Fig. 2 Structure of a KTHREAD block

BOOST Threads is an open-source library, written in C++. It offers classes and methods for thread management and aims to ease the creation of portable, safe, efficient and flexible multithread applications.

Both thread libraries, POSIX Pthreads for WIN32 and BOOST Threads, offer the user the capability of accessing their source code. The code compilation on a WIN32 platform offers as result either a statically linked (.lib) or dynamically linked (.dll) library. In the POSIX Pthreads case, the compilation was performed for both INTEL X86 processors (Windows XP) and ARM processors (Windows Mobile 5.0). For both platforms, the choice was made for dynamically linked libraries to be offered as a result. In the BOOST Threads case, the compilation was performed only for INTEL X86 processors, resulting in a statically linked library..

3 Execution framework architecture

The general architecture of the application presented in figure 3 has a modular structure divided in several abstracting levels. On the low level of the framework application will use the operating system's drivers of different physical components took into account in the optimizing process of energy consumption: the processor, the battery, wireless chipset, main-board chipset, the memory etc. The kernel of the execution framework reads the available measurements through the monitoring drivers, and calculates the energy consumption of the running applications. It communicates with the external components through the application interface, by making use of specific energy consumption control messages.

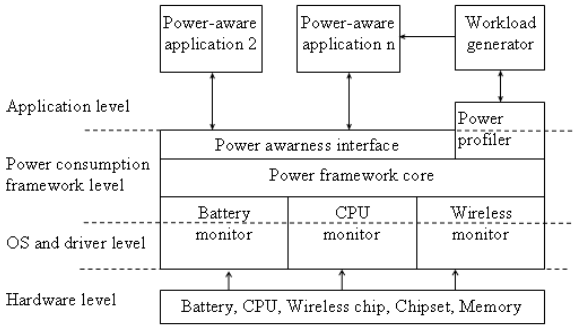

Fig. 3 Framework architecture

In order to show how different types of application level patterns influence the power consumption of a mobile device at the application level, we implemented a prototype of the framework. The prototype was written in C++ using MS Visual Studio 2005. The execution framework prototype source code is portable so it was built and tested on different Microsoft Windows platforms: Win32, Window Mobile 5.0 PocketPC and Windows Mobile 5.0 Smartphone.

The framework application is composed from a number of specialized modules (Fig. 2):

- Battery monitor - is a software module running at OS and drivers level, used to achieve real-time on-line power consumption measurements from battery device;
- CPU monitor - is a software module used to monitor CPU parameters such as load, temperature, etc.;
- Wireless monitor - is a software module implemented to monitor different parameters of wireless communication: signal power strength (RSSI), bandwidth, data transferred, etc.;
- other types of monitoring modules could also be implemented.
- Workload generator - it contains the multi-threading independent architecture presented bellow.
- Power profiler - logging and profiling module to save all monitoring values from all modules for offline analysis. This module

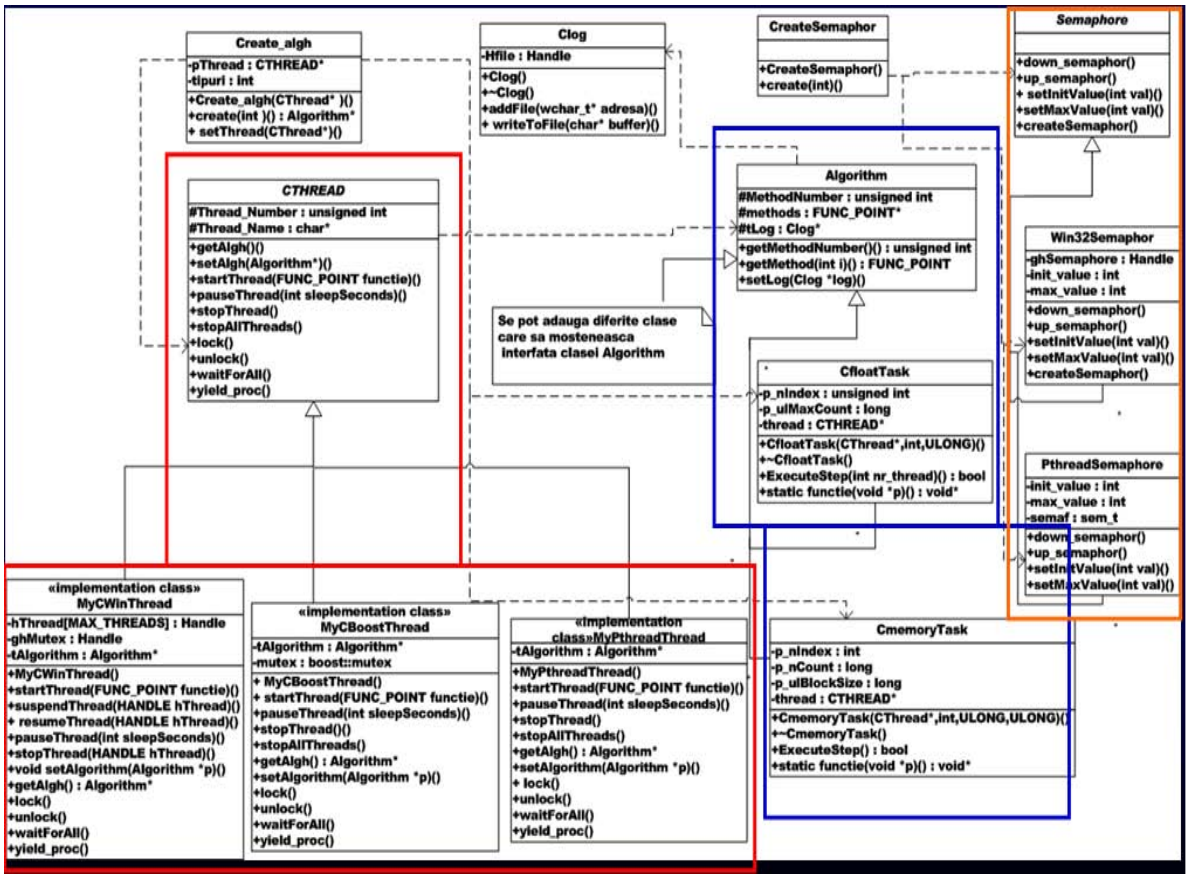

Fig. 4 Workload generator class diagram

is used for application power consumption profiling with respect to the used workload pattern.

- Power framework core and power framework API extract relevant monitoring data and provide it to the application level. These modules are used to implemented auto-adaptable mobile applications aware of their power consumption.

3.1 Workload generator

The code piece dealing with thread management (Workload generator in Fig. 3) was written also in Microsoft Visual Studio 2005 SP1 using C++. The methodology consisted of creating a set of classes that allow the user to create and run multiple algorithmic interactions on different types of threads (Fig. 4).

A first class created was CThread that defines the abstract interface for thread execution and management. Further, the MyCWinThread, MyCBoostThread and MyPthreadThread classes inherit this interface and implement its methods according to the particularities of the thread library. Another abstract class was Semaphore, that defines the interface for the management of semaphores used in thread synchronization. The classes Win32Semaphor and PthreadSemaphor implement this, but MyCBoostThread doesn't, as the semaphores were eliminated in Boost version 1.34.1, due to a security bug.

The abstract class Algorithm is inherited by all the classes that implement an algorithmic operation orchestration. This class contains several public methods that help with the creation of threads, and

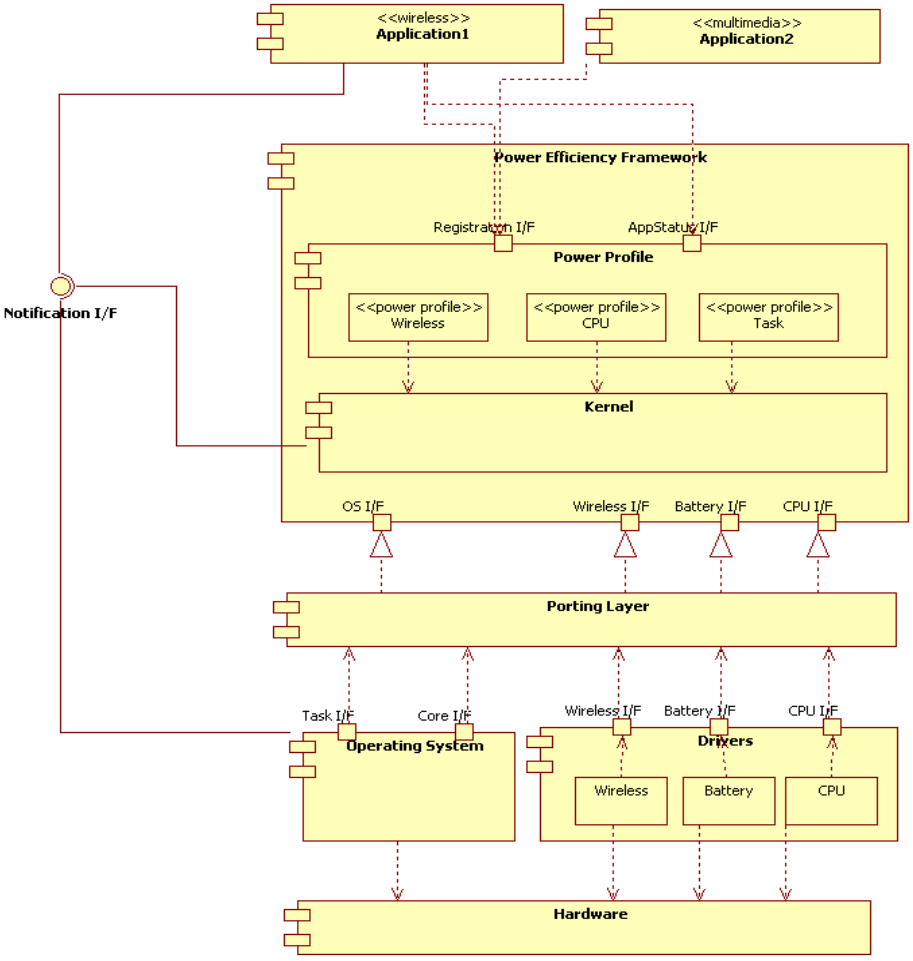

Fig. 5 Detailed framework architecture

can be used for the generation of both simple and multi-thread algorithms (classic multithreading problems, like producer-consumer), defined in their corresponding classes. Both the Algorithm type and the Semaphore type objects were created using the Factory method pattern.

3.2 Power framework core

The mechanism works on the principle of profiles (Fig. 5). The basic idea is that the monitoring framework defines a set of high level and domain specific profiles where applications are registering. The framework is informed by the application whenever a change occurs in the profile configuration data (e.g. download rate is changing). Based on the registered profiles and the available system measurements, the monitoring framework performs active monitoring and provides a feedback loop to applications or the operating system.

In the power efficiency monitoring framework, the profiles are playing a key role. We considered the following profile categories, which can be of course extended to other profiles if required:

– Wireless – the wireless profile is typically used by applications that are performing wireless communication. The profile defines the following parameters:
  - Download rate
  - Upload rate
  - Radio signal strength
– CPU – the CPU profile is typically used by applications that are performing a series of computations such as image/sound decoding or any other type of data processing. The profile has the following parameters
  - Thread ids – identifiers of the processing threads
  - Thread priorities – thread priorities for each of the defined processing threads
  - Thread wait state time – wait state time that the each thread issues during one loop
  - Thread/core mapping – mapping function from thread to processing cores in case of multi-core systems.

The applications are registering to one or more profiles, depending on their nature. Besides the registration interface, applications need to provide information on changes on their profile through a state changed interface.

4 Experimental results

We used the framework application we implemented to emphasize power consumption of different components of a mobile device in special multi-threading applications on mobile multi-core systems.

4.1 Experimental test-cases

In order to evaluate power efficiency profiles for multi-threading applications we elaborate a set of experiments, based on them we established a set of test cases. Every experiment ran for 30 minutes in the same environmental conditions.

Three hardware devices we used in our tests:

- Fujitsu-Siemens LOOX T830 and Qtek 8310 SmartPhones;
- Fujitsu-Siemens LOOX N560 PocketPC;
- Fujitsu-Siemens Intel Pentium IV dual core mobile 2000MHz laptop with 512 MB RAM.

The proposed test cases try to cover different aspects of multi-threaded applications:

- CPU power consumption;
- memory power consumption;
- thread-library power consumption;
- single-core, dual-core power consumption;

single-threaded, dual-threads and quad-threads power consumption.

4.2 Power benchmark profiling

A computer benchmark is typically a computer program that performs a strictly defined set of operations (a workload) and returns some form of result (a metric) describing how the tested computer performed. Computer benchmark metrics usually measure speed (how fast was the workload completed) or throughput (how many workloads per unit time were measured). Running the same computer benchmark on multiple computers allows a comparison to be made with respect to the applied workload.

The concept of benchmarking could be extended with another metric: the power consumption and we name it power benchmark. We used the concept of power benchmark profiling in our tests, therefore the concept is detailed described in [13].

A power benchmark must by able to distinguish the way power consumption is increasing with workload related to idle state consumption and the type of workload. Therefore, we define a power benchmark to be composed by three intervals (Fig. 6):

- the first time range [0-t_1), is intended for idle mode power consumption. In this step, the component does not execute anything, but the power saving mechanisms are prevented to occur.
- the second time range [t_1-t_2) represents the workload phase, when a certain stimulus is executed. SPEC CPU2000 or any type of other applications can be executed as workload.
- the last time range [t_2-t_3) represents the releasing phase intended for the component to reach again the idle state power consumption. In this step, the component does not execute anything, but the power saving mechanisms are also prevented to occur.

There are two ways the power benchmark can be implemented:

- fixed times power benchmark – the benchmark times t_1, t_2 and t_3 are predefined and constant. This kind of power benchmark shows the maximum component power consumption when a certain workload is applied for a constant period of time (t_2-t_1);
- fixed power consumption values power benchmark – the benchmark times are variable and benchmark power measured values are predefined and constant. This benchmark shows how many workload operations are executed per consumed energy.

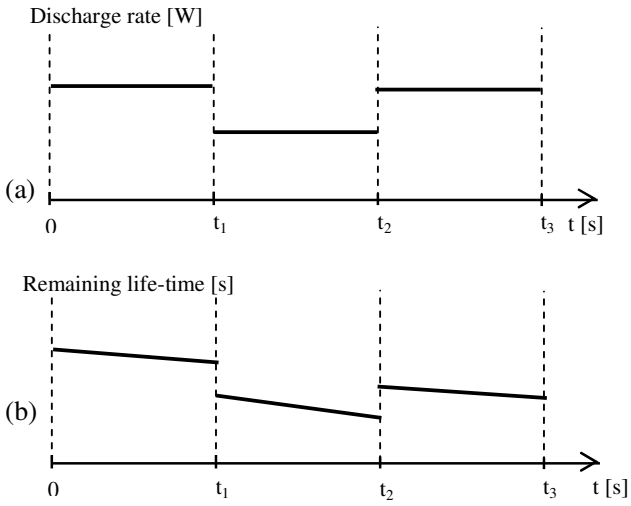

Fig. 6 Power benchmark definitions

4.3 Workload power consumption

By running the power benchmark with different workloads for CPU, we obtain the results depicted in Fig. 7 were obtained. In this case the benchmark shows the consumption of the mobile microprocessor when different workloads are applied: integer, memory and float. Power signatures in Fig. 7 are obtained for an Intel Pentium IV dual core mobile 2000MHz laptop with 512 MB RAM. For the mobile CPU power consumption we observed that there are no major differences between different types of CPU workload patterns (float, integer), but the CPU power consumption depends of CPU usage percent (CPU load).

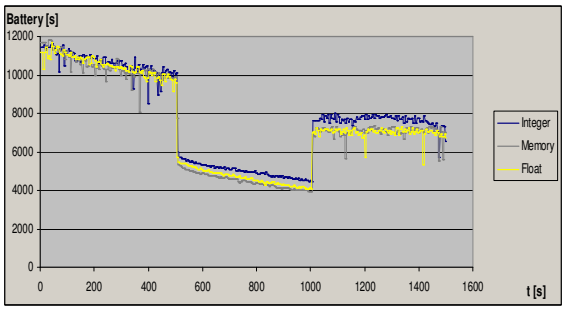

Fig. 7 Power signatures of the processor for different workloads

4.4 CPU load power consumption

We tried to run the same intensive computational workload at different CPU loading percentage in one thread and we measured their influence on power consumption and battery discharge. It can be observed that the usage of the processor under its higher load capacity could increase the battery lifetime for the same number of computations. After the workload was finished different battery status parameters were achieved depending of the used CPU load (Fig. 8). Therefore the same numbers of computations were executed with different power consumption.

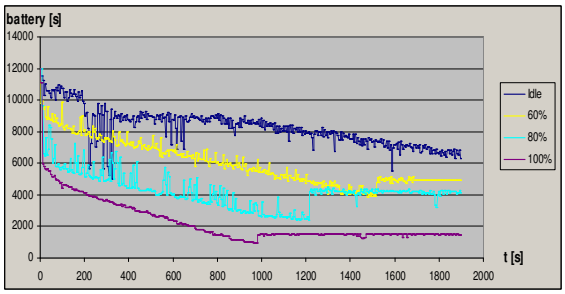

Fig. 8 Power signatures of the same workload at different CPU loads

4.5 Multithreading power consumption

Another test we run was to launch the same algorithm workload on different thread counts. We started with one thread and one algorithm on one dual-core laptop. The same algorithm was further executed on two threads, each thread on one CPU core. An increase in power consumption is observed when the second core is used (Fig. 9). When four threads were executed no significant increase of power consumption was observed.

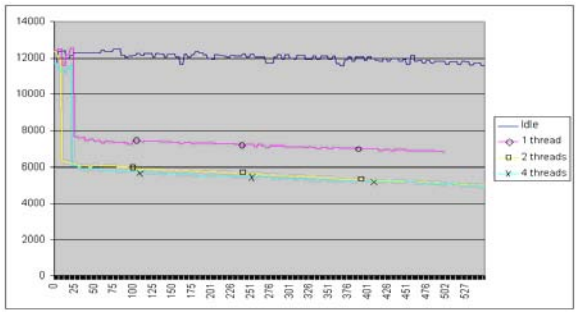

Fig. 9 Power signatures of different thread counts

4.6 Thread libraries power consumption

Next, we elaborated a set of tests in order to distinguish how the three thread libraries we used influence the power consumption. In these tests we run the same workload in 1, 2 or 4 threads using Win32, PThreads and Boost threads libraries. We each test 3 times on two different devices: one dual-core Intel processor notebook and one single core ARM processor Look T830 PocketPC

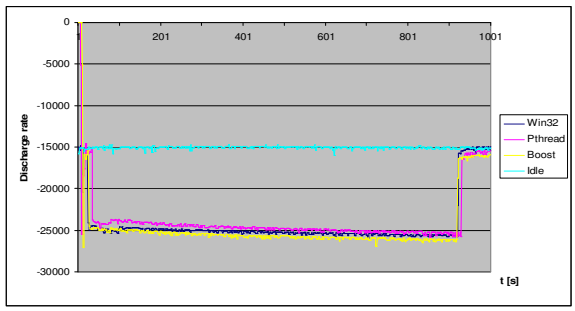

Fig. 10 Power signatures for 1 thread running within different thread libraries on a Notebook

We obtained the results in Fig. 10, 11, 12 and 13. It can be observed that there is no significant power consumption difference when different thread libraries are used. When running the floating point workload in one thread on the dual-core notebook the power consumption increase with around 10 W related to the idle state power consumption (15 W). When two threads are running with the same workload the power consumption increase is 15 W related to the idle power consumption and only 5 W related to the one threaded tests.

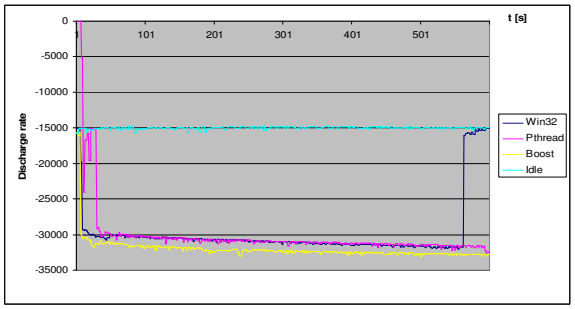

Fig. 11 Power signatures for 2 concurrent threads running within different thread libraries on a Notebook

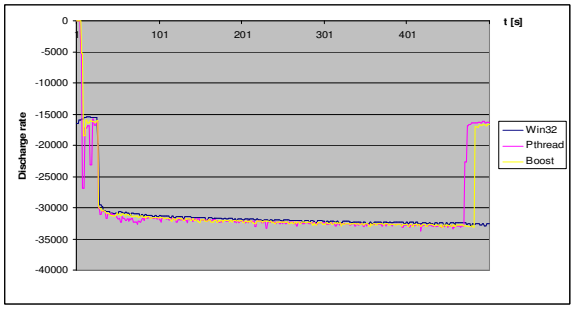

Fig. 12 Power signatures for 4 concurrent threads running within different thread libraries on a Notebook

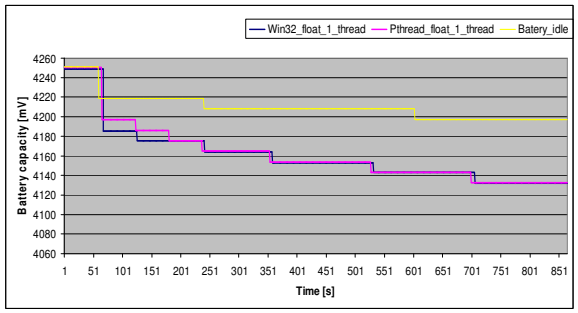

Fig. 13 Power signatures for 1 thread running within different thread libraries on a PocketPC

4.7 Multi-threaded producer-consumer power consumption

Running the implementation of well known producer-consumer problem on a single-core PocketPC device using Pthread and Win32 threads libraries the picture in Fig. 14 is obtained.

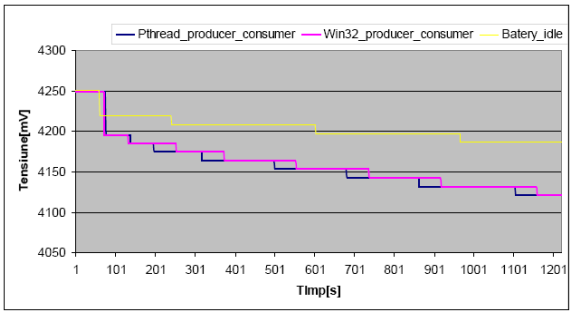

Fig. 14 Single-core producer-consumer power consumption (PocketPC)

On a dual-core CPU laptop the producer-consumer power profile is presented in Fig. 15.

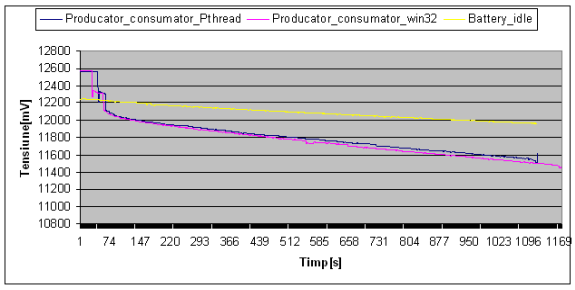

Fig. 15 Dual-core producer-consumer power consumption (Laptop)

4.8 Multi-threaded matrix multiplication power consumption

The last test was to implement a real life application computing a large amount of data which could be efficiently parallelized. We implemented a n x n matrix multiplication algorithm parameterized with the number of threads available in the threads pool. When the matrix multiplication algorithm is executed on a single-core CPU PocketPC device using one or two threads in the pool, the power profiles in Fig. 16 (Pthread library) and Fig. 17 (Win32) are obtained.

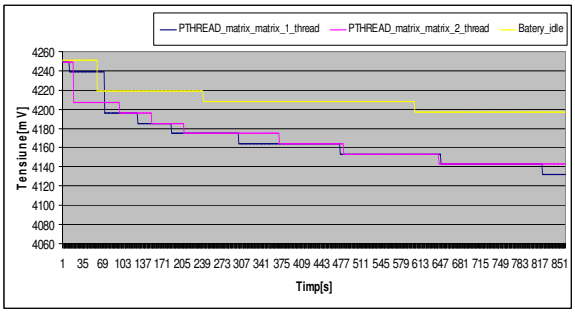

Fig. 16 Single-core matrix multiplication (PThread)

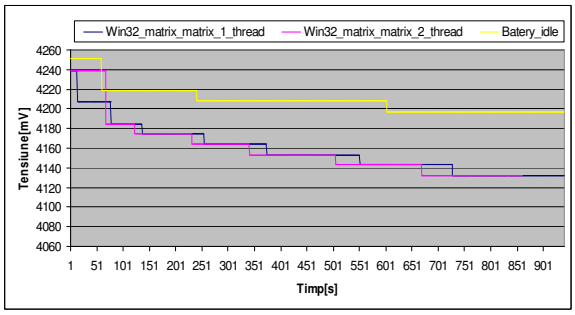

Fig. 17 Single-core matrix multiplication (Win32)

Running the matrix multiplication algorithm on a dual-core CPU Laptop using one or two threads the power profiles in figures 18, 19 and 20 are obtained. All three thread libraries are used: PThread, Win32 and Boost. When using two threads the energy efficiency of the algorithm is increasing with around 40-45% for all thread libraries.

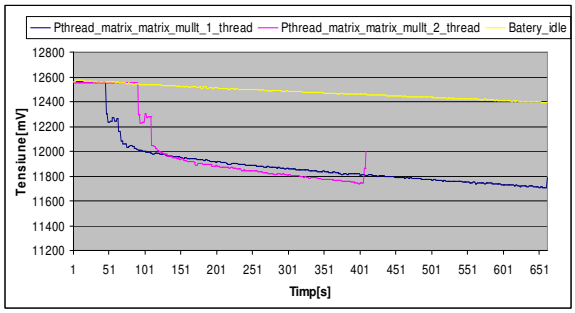

Fig. 18 Dual-core matrix multiplication (PThread)

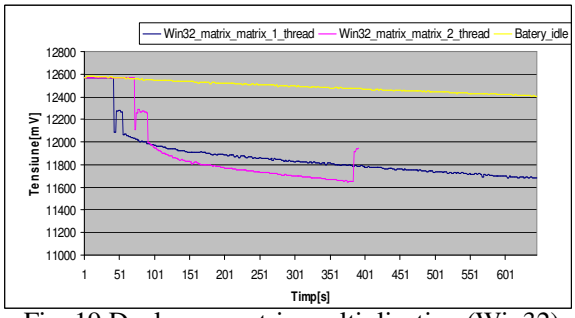

Fig. 19 Dual-core matrix multiplication (Win32)

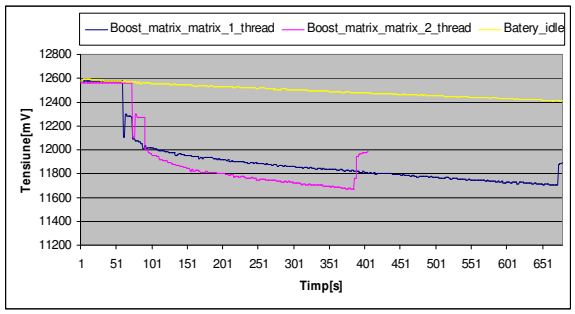

Fig. 20 Dual-core matrix multiplication (Boost)

We tried to run the matrix multiplication algorithm with different CPU loads and uneven thread balanced (Fig. 21).

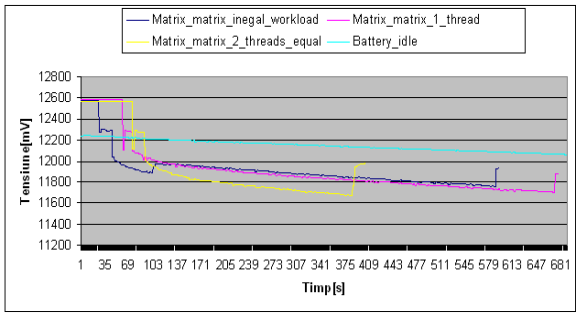

Fig. 21 Dual-core matrix multiplication with different CPU loads

5 Conclusion

In our work we tried to create an application framework that allows the execution of different types of threads (by using multiple thread libraries for Windows OS), comparing their efficiency and measuring the power consumption on mobile devices. We used this framework to show how multithreading mobile applications influence the power consumption of the single-core/multi-core battery powered devices. We tested the framework with Win32 threads and we have to run the same tests for Boost threads and PThreads.

Mobile systems based on multi-core processors will appear in short time, as today there are only laptop systems based on this kind of processors. When these mobile processor will appear, a lot of opportunities will come up, opportunities will address also the power consumption of multi-core CPU. Introducing DPM algorithms for allocating execution threads and processes on different cores depending on the present situation of the system's supply is another future direction for our tests.

References:

[1] Jacob Sorber, Nilanjan Banerjee, Mark Corner, and Sami Rollins, "Turduken: Hierarchical Power Management for Mobile Devices", The Third International Conference on Mobile Systems, Applications and Services, Mobisys2005, Jun. 6-8, 2005, USA.

[2] Lin Zhong and Niraj Jha, "Energy Efficiency of Handheld Computer Interfaces: Limits, Characterization and Practice", The Third International Conference on Mobile Systems, Applications and Services, Mobisys2005, Jun. 6-8, 2005, USA.

[3] PThreads Primer. A Guide to Multithreaded Programming. SunSoft Press.Prentice Hall Title. 1996. ISBN 0-13-443698-9.

[4] Protothreads - Lightweight, Stackless Threads in C, http://www.sics.se/~adam/pt/

[5] Microsoft® Windows® Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000. Microsoft Press. Copyright 2004. ISBN 0-73-561917-4.

[6] Boost C++ libraries, www.boost.org

[7] Open Source POSIX Threads for Win32 http://sourceware.org/pthreads-win32

[8] Wireless World Research Forum, Book of Visions 2001, http://www.wireless-world-research.org

[9] Suresh Siddha et all, Process Scheduling Challenges in the Era of Multi-core Processors, Intel Technology Journal, Volume 11, Issue 04, ISSN 1535-864X, November 2007

[10] Mohan Rajagopalan, Brian T. Lewis, Todd A. Anderson, Thread scheduling for multi-core platforms, Proceedings of the 11th USENIX workshop on Hot topics in operating systems, p.1-6, May 07-09, 2007, San Diego, CA

[11] Christiana Ioannou, Yiannakis Sazeides, Pierre Michaud, Martha Vasiliadou. Thermal Aware Multi-Core Scheduler, ACACES 2007, July 2007

[12] Zili Shao et al, Real-Time Dynamic Voltage Loop Scheduling forMulti-Core Embedded Systems, IEEE Transactions on Circuits and Systems II (TCAS-II), Volume 54, Issue 5, pp 445-449, 2007

[13] Marius Marcu, Mircea Vladutiu, Horatiu Moldovan, "Microprocessor Thermal Characterization using Thermal Benchmark Software", WSEAS TRANSACTIONS on COMPUTERS, Issue 11, Volume 5, ISSN 1109-2750, pp. 2628-2633, November 2006.

[14] T. Hubbard, R. Lencevicius, E. Metz and G. Raghavan, "Performance Validation on Multicore Mobile Devices", Proceedings of IFIP Working Conference on Verified Software: Tools, Techniques and Experiments, VSTTE2005, Zurich, Switzerland, 2005

[15] J. Levendovszky, A. Bojarszky, B. Karlocai, A. Olah, "Energy Balancing by Combinatorial Optimization for Wireless Sensor Networks", pp. 27-32, WSEAS TRANSACTIONS on COMMUNICATIONS, Issue 2, Volume 7, February 2008.

[16] Z. Toprak, Y. Leblebici, "A Low-Power Adaptive Bias/Clock Generator for Fine-Grained Voltage and Frequency Scaling in Multi-Core Systems", WSEAS

TRANSACTIONS on SYSTEMS, vol. 4, num. 12, 2005, p. 2390-2397.

[17] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, Joonwon Lee, "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors," IEEE Transactions on Parallel and Distributed Systems, 17 June 2008.

[18] D. Petcu, A. Eckstein, C. Giurgiu, "Adapting a Legacy Code for Ordinary Differential Equations to Novel Software and Hardware Architectures", WSEAS TRANSACTIONS on COMPUTERS, Issue 5, Volume 7, pp. 463-472, May 2008

[19] Chris Kanaracus, "Intel, Microsoft: Multi-core chips need new developer skills", Macworld, http://www.macworld.com/article/132630/2008/03/multicore.html, Mar. 2008.