Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments

Christian Bunse Hagen Höpfner Essam Mansour Suman Roychoudhury

School of Information Technology
International University in Germany
Campus 3, 76646 Bruchsal, Germany

first_name.last_name@i-u.de

Abstract

Most mobile and embedded devices are battery powered. Hence, their uptime depends on the energy consumptions of the used components. Developers made much effort to optimize hardware in order to reduce their energy consumption. However, we show in this paper that one also has to consider energy awareness in terms of software. In this study we focus on sorting algorithms, which are not only used directly by the user of a device but also very often implicitly by other algorithms. Our experiments show, that different sorting algorithms have different energy consumptions. Furthermore, the experiments show that there is no direct correlation between the time consumptions (complexity) of an algorithm and its energy consumption.

1. Introduction and Motivation

Managing data on mobile devices is always characterized by handling limited resources like memory, CPU performance, or energy supply. Mobile information systems should be able to adapt themselves in order to meet the users requirements. If a maximal uptime of the device is required, the user would (perhaps) agree to slow down the system or to avoid network communications, even if some data gets outdated. However, in order to realize such a resource substitution [12] one needs to have a deep understanding of the interdependencies between the resources and how they are used. In this paper we analyze the influence of using CPU and memory on the uptime of devices by measuring the energy consumptions of different sorting algorithms.

In computer science and mathematics, sorting algorithms are used to arrange elements in a specific order (e.g., numerical or lexicographical order). Efficient sorting is important to optimize the use of search and merge algorithms that require sorted lists. Furthermore, many data(base) management algorithms that implement join (e.g., Sort-Merge-Join) or duplicate eliminating set operations implicitly use sorting algorithms. Even simple operations like `SELECT DISTINCT` call a sorting method. Therefore, sorting is a hot topic since the dawn of computing and are therefore prevalent in introductory computer science classes. In such classes students learn about sorting algorithms and their classification according to the big-O-notation, their best, worst and average case analysis, as well as time-space tradeoffs, and lower bounds. Students tend to select Quicksort as the most favorable sorting algorithm. This is also taken further by the fact that Quicksort is the standard sorting algorithm of many programming languages and libraries.

Although, the appraisal of Quicksort is not wrong from the performance viewpoint. However, it does not reflect the complete picture especially with respect to energy consumption when executing a specific piece of software. Typically, there is a strong believe that software energy consumption is directly related to software performance (i.e., the number of processor cycles directly determines energy consumption). However, other factors such as memory usage or recursion do have an impact, too. The experiments described within this paper show that, at least for specific processor families, Quicksort might be the fastest sorting algorithm but not the most energy-saving one. The experiments show that Insertionsort provides the best rationale between performance and energy consumption.

The remainder of this paper is structured as follows: Section 2 lists related works. Section 3 briefly introduces the sorting algorithms researched for this paper. Section 4 describes the experimental setup. Section 5 presents the results of our experiments. Section 6 summarizes and concludes the paper and gives an outlook on future research.

2. Related Work

Due to the orientation towards mobile- and embedded-based systems, several research efforts have investigated into the topic of energy consumption. Optimizing energy consumption is one of the fundamental factors for an efficient battery-powered system. Jain et al. [14] classified these research efforts into four levels of abstraction; the logic design level, the processor level, the operating system level and the compiler level. However, we have classified the research in energy consumption into the two main categories software and hardware. The research, which belongs to the hardware category, attempts to optimize the energy consumption by investigating the hardware usage, such as [7, 19], and innovating new hardware devices and techniques, such as [29, 31].

The research in second category attempts to understand how the different methods and techniques of software and data management affecting the energy consumption in order to achieve a higher power saving. We have classified the research work of this category according to the main factors affecting the energy consumption. These factors are *networking*, *communication*, *application nature*, *memory management*, and *algorithms*.

In *networking*, different research efforts, such as [9, 24], provide new routing techniques that are aware of the energy consumed during routing packets. Others efforts of this category focus on providing energy-aware protocols for transmitting data in wireless networks generally, such as [23, 27], and ad-hoc networks, such as [10]. One of the fundamental techniques proposed to reduce communication is caching technique. The efforts belonging to the *communication* category introduced several energy-aware caching techniques, such as [26, 4, 32].

Due to Java platform independence, many applications are Java-based. In *application nature*, we consider the research efforts investigating into the topic of energy consumption for the java-applications and java virtual machine, such as [25, 16, 3, 8]. Memory energy is one of the major energy, which is to be saved. In *memory management*, several research efforts, such as [15, 21], have provided energy-aware memory management.

In battery-powered systems, it is not enough to analyze algorithms based only on time and space complexity. In the *algorithms* category, several research proposed energy-aware algorithms for specific functionalities, such as [14] supporting randomness, [22] focusing on cryptographic, and [28] investigating into wireless sensor networks.

The contribution of this paper is the analysis of the energy consumption for sorting algorithms. This paper belongs to the *algorithms* category. There has been hardly any research on the impact of sorting algorithms with respect to energy consumption. In addition to our recent study on sorting algorithms, we had previous focused on dynamic strategies like resource substitution as means of saving energy [12, 6]. Measurements by other researcher show, that substituting resources might save much energy. The authors of [30] found out that compressing a file by more then 10%, transmitting and decompressing it requires less energy than transmitting it uncompressed.

3. Sorting Algorithms

Sorting appears to be "easy" but its efficient execution by machines is inherently complex. Even today, sorting algorithms are still being optimized or even newly invented. When it comes to mobile systems and information retrieval efficient sorting is a major concern concerning performance and energy consumption. In the following we describe the set of sorting algorithms that were used in the context of this study. This set was defined to comprise the major algorithms that are either used in form of library functions (e.g., Quicksort), are easily programmable (e.g., Bubblesort) or that are regularly taught to IT students. In other words, our goal was to cover those algorithms that are in widespread use. More details on them can be found in standard text books on algorithms and data structures such as [17].

Bubblesort is a simple sorting algorithm that belongs to the family of comparison sorting. It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. Bubblesort has a worst-case complexity $O(n^2)$ and in the best case $O(n)$. Its memory complexity is $O(1)$.

Heapsort is a comparison-based sorting algorithm, and is part of the Selectionsort family. Although somewhat slower in practice on most machines than a good implementation of Quicksort, it has the advantage of a worst-case $O(n \log n)$ runtime.

Insertionsort is a naive sorting algorithm that belongs to the family of comparison sorting. In general Insertionsort has a time complexity of $O(n^2)$ but is known to be efficient on data sets which are already substantially sorted. Its average complexity is $n^2/4$ and linear in the best case. Furthermore Insertionsort is an in-place algorithm that requires a constant amount $O(1)$ of memory space.

Mergesort belongs to the family of comparison-based sorting. It has an average and worst-case performance of $O(n \log n)$. Unfortunately, Mergesort requires three times the memory of in-place algorithms such as Insertionsort.

Quicksort [11] belongs to the family of exchange sorting. On average, Quicksort makes $O(n \log n)$ comparisons to sort n items, but in its worst case it requires $O(n^2)$ comparisons. Typically, Quicksort is regarded as one of the most efficient algorithms and is therefore typically used for all sorting tasks. Quicksort's memory usage depends on factors such as choosing the right Pivot-Element, etc. On average, having a recursion depth of $O(\log n)$, the memory complexity of Quicksort is $O(\log n)$ as well.

Selectionsort belongs to the family of in-place comparison sorting. It typically searches for the minimum value, exchanges it with the value in the first position and repeats the first two steps for the remaining list. On average Selectionsort has a $O(n^2)$ complexity that makes it inefficient on large lists. Selectionsort typically outperforms Bubblesort but is generally outperformed by Insertionsort.

Shakersort [5] is a variant of Shellsort that compares each adjacent pair of items in a list in turn, swapping them if necessary, and alternately passes through the list from the beginning to the end then from the end to the beginning. It stops when a pass does no swaps. Its complexity is $O(n^2)$ for arbitrary data, but approaches $O(n)$ if the list is nearly in order at the beginning.

Shellsort is a generalization of Insertionsort. The algorithm belongs to the family of in-place sorting but is regarded to be unstable. The algorithm performs $O(n^2)$ comparisons and exchanges in the worst case, but can be improved to $O(n \log_2 n)$. This is worse than the optimal comparison sorts, which are $O(n \log n)$. Shellsort improves Insertionsort by comparing elements separated by a gap of several positions. This lets an element take "bigger steps" toward its expected position. Multiple passes over the data are taken with smaller and smaller gap sizes. The last step of Shell sort is a plain Insertionsort, but by then, the array of data is guaranteed to be almost sorted.

4. Experimental Setup

In order to evaluate the actual energy consumption of a processing core executing a certain software artifact a specific evaluation platform is needed. Energy consumption cannot be measured directly at the boards power supply due to various consumers (LEDS, transformers, etc.). We developed a evaluation/measurement platform that directly captures the energy consumption of the core processor (CPU) of an embedded system. It logs all measurement as well as the time the measurement was made to a log file. The log file is then used as basis for calculating the actual energy consumption of the system. In principle, the data acquisition rate should be adjustable. On the one hand this allows to measure energy consumption of single function calls, and on the other hand this allows capturing the energy consumption of long-running programs.

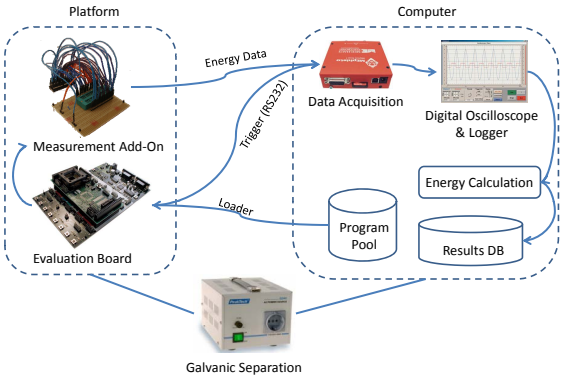

Figure 1. Platform Overview

Figure 1 shows an overview of the energy measurement platform. The reason for choosing a micro controller is twofold. First, they are in widespread use for embedded applications and second, they do *not* require an operating system to perform. Thus, measurements can be directly mapped to a specific software artifact (part of) without including effects or (hidden) tasks of the operating system.

The chosen evaluation board (i.e., STK500 or STK501 depending on the processors type) is flashed with an `.hex` image file using ATMEL's AVRStudio [1]. Once flashed and rebooted the program is automatically started. Every program sends a TTL level signal [18] at the start and end of its run. This signals are fed into the digital oscilloscope to trigger measurement and logging. Triggering was chosen in order to concretely and correctly measure energy related data of a program run. During program run, the digital oscilloscope measures the voltage drop at a sense resistor embedded into an add-on board (used to ease the change of processors, provide measurement points, etc.) and logs this data to a file. The collected data is then processed in order to calculate the consumed energy values which are finally stored.

4.1. Measurement Theory

Unfortunately, energy consumption cannot be directly measured since it is a function over time. However, energy consumption, measured in Joule, can indirectly be measured by tracking the power P_{CORE} that is consumed by

the processor core. This power is the product of the core voltage U_{CORE} and the current flow I_{CORE}:

$$P_{CORE} = U_{CORE} \cdot I_{CORE} \tag{1}$$

According to Kirchhoff's laws, the amperage or current that is required by the core I_{CORE} equals the current through a sense resistor R_{SENSE} :

$$I_{CORE} = I_{SENSE} \tag{2}$$

Following Ohm's law I_{SENSE} equals the quotient of the voltage U_{SENSE} that is measured and the resistor value R_{SENSE}. Thus, I_{SENSE} can be calculated by:

$$I_{SENSE} = \frac{U_{SENSE}}{R_{SENSE}} \tag{3}$$

Using Formulas 2 and 3 in Formula 1 thus yields:

$$\begin{aligned} P_{CORE} &= (U - U_{SENSE}) \cdot \frac{U_{SENSE}}{R_{SENSE}} \\ &= \frac{\left(U \cdot U_{SENSE} - U_{SENSE}^2\right)}{R_{SENSE}} \end{aligned} \tag{4}$$

The energy that is consumed by the processor is represented by the integral of the power P_{CORE} over time t. When using Formula 4 the consumed energy can be computed by measuring the voltage drop at the sense resistor over time t:

$$E = \int P_{CORE}(t)\, dt = \int \frac{U \cdot U_{SENSE} - U_{SENSE}^2}{R_{SENSE}} dt \tag{5}$$

Using mathematical laws this can be simplified to:

$$\begin{aligned} E &= \frac{1}{R_{SENSE}} \cdot \int \left(U \cdot U_{SENSE}(t) - U_{SENSE}^2(t)\right) dt \\ &= \frac{1}{100\Omega} \cdot \int \left(4.93V \cdot U_{SENSE}(t) - U_{SENSE}^2(t)\right) dt \\ &= \frac{1}{100\Omega} \Delta t \cdot \sum_{n=0}^{\frac{t}{\Delta t}} 4.93V \cdot U_{SENSE}(n \cdot \Delta t) \\ &\quad - U_{SENSE}^2(n \cdot \Delta t) \end{aligned}$$

Whereby R_{SENSE} equals 100Ω, *U*, as provided by the stabilized power supply, equals 4.93V, and n represents the number of samples per second that are collected by the data acquisition hardware. Furthermore, the sum transformation assumes that the time interval for one sample results to $\Delta t = \frac{1s}{n}$.

4.2. Evaluation Board

The Atmel AVR® STK500 board [2], suitable for developing software systems for AVR Flash micro-controllers, was chosen as the basis for the energy consumption measurement platform. It supports a variety of processors including the ATtiny and AT90 series, as well as support for various ATMega processors (e.g., ATmega8, ATmega8515, ATmega16, ATmega32, ...). In combination with the STK501 expansion board, there is even support for TQFP packages such as those for the ATMega103 or ATMega128). The STK500 board provides several measurement points and jumpers (e.g. for testing the board elements such as core or peripherals). Using the standard power supply one possible measurement point is the VTARGET jumper. VTARGET controls the operating voltage for the board. Unfortunately, the jumper control the operating voltage for all elements of the board including LEDs. Thus measurement data obtained at this jumper might mislead. Therefore, we decided to directly measure the core current. As the STK500 does not provide a measurement point for measuring the core current directly, a measurement add-on board that provides such a point was created. This add-on board also provides a sense resistor, $R_{SENSE} = 100\Omega$, through which power is fed to the core. The voltage drop at this resistor is used to measure the actual core amperage.

4.3. Measurement

In the context of this paper we conducted two different experiment series. The first series measured the energy consumption of different sorting algorithms with fixed size data (integer values), running on different AVR processors (e.g., ATMega 16, 32, and 128). The second series examined the same algorithms but with varying data sizes (array lengths from 0 to 1,000) that were executed on a ATMega128 processor. In addition, we equipped the processor with external SRAM (64 Kbyte) by adding an IDT 71124 memory chip to the STK501, to evaluate the impact of external storages.

Both series measured the consumed energy (in Joule) for every algorithm concerning the use of random (equivalent for all algorithms), pre-sorted and reverse sorted data to cover best, average and worst case scenarios. To level-out measurement errors each measurement cycle covers multiple executions of the algorithm using the same data.

In addition we rerun the second series using float values instead of integers. This was done in order to confirm the findings obtained in [13] concerning the impact of data types onto energy consumption. Results of all runs are presented in the following section.

5. Measurement Results

This section discusses the results of the different experimental runs concerning the energy consumption of sorting algorithms. Please note that all figures in this section use accumulated, non normalized values. Results are sums of

measurements results of random, sorted and reverse-sorted data and are accumulated for 1,000 cycles (1st series) or 500 cycles (2nd and 3rd series).

5.1. 1st Series Results

The first experiment series was performed in order to evaluate if software energy consumption, as widely believed, is strongly correlated to software performance (i.e., the number of execution cycles is solely responsible for energy consumption). Therefore we executed a number of standard sorting algorithms on different processors of the same processor family (i.e., AVR), whereby the consumed energy (see Figure 2), the execution time (see Figure 3) as well as the number of cycles (see Figure 4) were measured. The latter two were obtained by using the AVR simulator of AVRStudio.

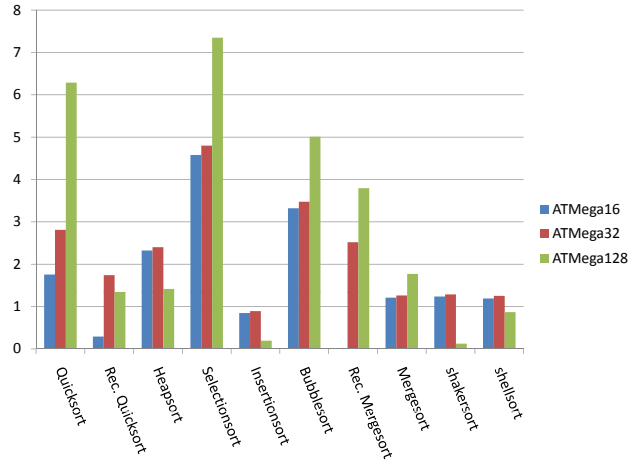

Figure 2. Energy Consumption in Joule

Initially, the results regarding the energy consumption of different sorting algorithms reveal that, in contrast to our assumption, sorting algorithms such as Insertionsort (Complexity of $O(n^2)$) require significant less energy than high-performance algorithms such as Quicksort (Complexity of $O(n \log n)$). When looking at the measurement results in detail (see Table 1[1]) it is obvious that this observation is valid and visible across different processors of the same family. Another observation is that energy consumption grows with the inbuilt flash memory size (15 KByte, 32 KByte, and 128 KByte). Interestingly, recursive Quicksort outperforms Insertionsort when running on the ATMega16 but is clearly behind at all other platforms. The reason is that both the table as well as the figure represent the accumulated values (random, sorted, reverse sorted), and recursive Quicksort failed (out of memory) to sort the sorted and reverse sorted data. Therefore the presented value is much lower.

[1]"Empty" table cells represent the missing of results for algorithms that could not be successfully executed on a specific processor. For example it was not possible to use the recursive Mergesort for sorting 1,000 integer values on the ATMega16 since the system ran out of memory

	ATMega 16	ATMega 32	ATMega 128
Quicksort	1.755812443	2.810119993	6.286183061
Rec. Quicksort	0.291297773	1.740904365	1.342775090
Heapsort	2.322971333	2.400955424	1.415536144
Selectionsort	4.578156931	4.799888724	7.346836611
Insertionsort	0.846760420	0.892424873	0.193540497
Bubblesort	3.320823135	3.474285035	5.009593535
Rec. Mergesort	n.a.	2.518289575	3.794298385
Mergesort	1.208766752	1.261702605	1.770662786
Shakersort	1.236529835	1.286544273	0.123606970
Shellsort	1.189738359	1.252544965	0.867869552

Table 1. Energy Consumption in Joule

Concerning execution time (Used clock speed is 4 MHz) and the number of cycles, measured by using the AVR Simulator, the results are as expected. Although the data shows some variation (based on known bugs of the simulator) in general execution times as well as cycles are equivalent across processors. This was expected since all processors share the same underlying general processor architecture (32-Bit RISC architecture).

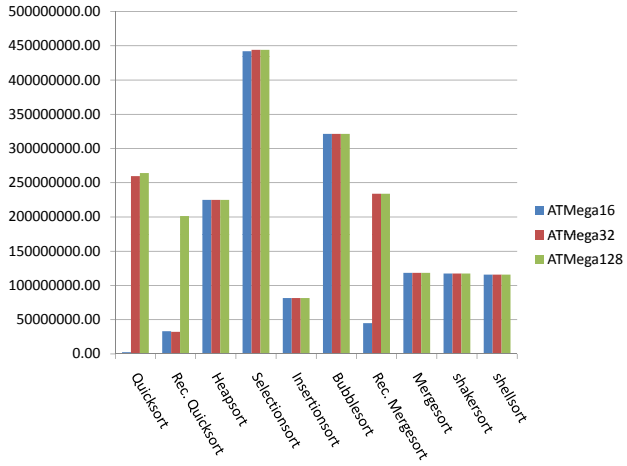

Figure 3. Execution Time in ms

When looking at the detailed measurement results (see Table 2) we can see that execution time follows the complexity class of the algorithms. However, it is interesting to see the differences in execution time between recursive (tuned) and non-recursive variants of the same algorithms (e.g., Quicksort). This leads to the assumption that the use of recursion might improve performance, but will definitively increase energy consumption.

	ATMega 16	ATMega 32	ATMega 128
Quicksort	2520.00	259640.00	237534.00
Rec. Quicksort	33120.00	32240.00	201320.00
Heapsort	224920.00	224920.00	224920.00
Selectionsort	442040.00	444120.00	443720.00
Insertionsort	81640.00	81600.00	81640.00
Bubblesort	321320.00	321320.00	321320.00
Rec. Mergesort	44920.00	233920.00	233920.00
Mergesort	118400.00	118400.00	118400.00
Shakersort	117400.00	117400.00	117400.00
Shellsort	115800.00	115800.00	115800.00

Table 2. Execution Time in ms

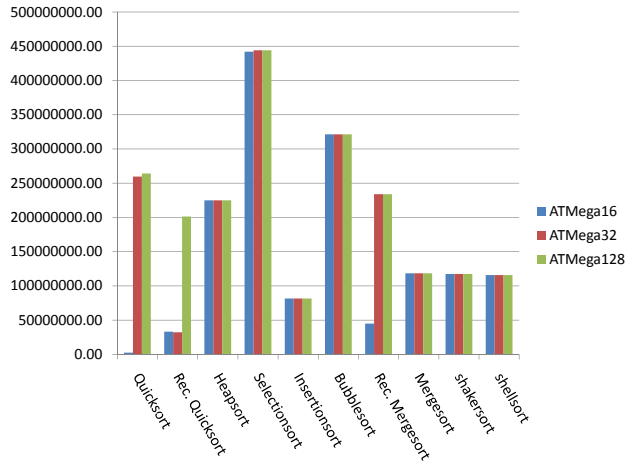

Figure 4. Used CPU Cycles

5.2. 2nd Series Results

The second experiment series had two different goals: First, to measure the energy consumption of sorting algorithms for growing data sizes (i.e., number of elements to be sorted). Second, to evaluate the impact of using external memory on energy consumption (i.e., beyond the energy needed for powering the memory chip).

Concerning the first goal, we measured the energy consumption of a algorithm subset of the first run (wrt. random, sorted and reverse sorted data) for increasing length of data (0 - 1,000 elements). The subset comprises Quicksort (standard and refined), Mergesort (standard and refined), as well as Insertionsort. In general, sorting varying (probably large) data sizes may require a huge amount of memory especially when it comes to recursive and non in-place algorithms.

The obtained measurement results (see Figure 5) in general confirm the results of the previous experiment series by showing that Insertionsort consumes significantly less energy than other algorithms such as Quicksort or Mergesort, although these belong to another (better[2]) complexity class.

Regarding the second goal, we extended the microprocessor's inbuilt SRAM memory (from 4 KByte to 132 KByte) by integrating an external memory chip (i.e., K6R1008C1C-JC12) to the evaluation board. At a first glance (comparing data of run 1 and 2) it is obvious that using external memory requires significantly more energy (i.e., for Insertionsort the energy consumption for sorting 1000 random elements raised from 0.03 to 4.11 Joule (500 execution cycles). The difference cannot be explained by the standard energy the additional chip requires since the differences between both curves strongly diverge with growing data size. This supports our assumption that moving data to/from external memory and addressing/managing these additional memory cells has a price and should be considered when planning a system.

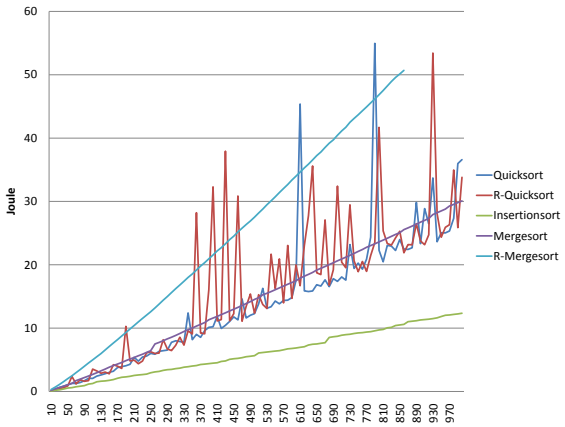

Figure 5. Comparison of Sums (rand, sort, rev)

In order to visualize the differences between algorithms and to indicate the growth of energy consumption we interpolated trend functions for each algorithm while watching the related square of the correlation coefficient, representing the quality of the trend, which should be as close as possible to one. It appears that for most cases these functions are quadratic (e.g., the trend function of Quicksort is $t(x) = x^{1.1388} \cdot 0.1533$, where x is the number of processed data items). The curves of these trend functions, using values between 0 an 10,000, are shown in Figure 6. Although growth and differences are nicely visualized the trend functions cannot be used for estimation purposes.

An interesting result shown in Figure 5 is that the energy consumption of "high-performance" algorithms such as Quicksort or Mergesort are quite close. In addition the curves related to Quicksort show several peaks due to large

[2]In fact, sorting 1,000 randomized elements with Insertionsort took 71.3 ms, whereas Quicksort needed 8.8 ms.

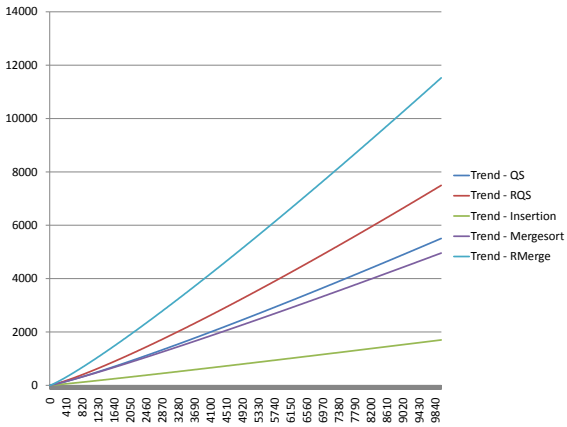

Figure 6. Trends

shifts within memory. These characteristics can be utilized in the context of managing energy consumption based on user profiles. If a user decides that he would like to use a high-performance algorithm while, at the same time, optimizing energy consumption the curves can be used to select the ”best” algorithm, user-driven depending on the size of the data to be sorted. In detail, the plan is to use quasi-interpolations based on trigonometric splines to interpolate the function of each curve. These functions can then be used as ”cost functions” at runtime.

However, before generalizing the obtained results by using cost functions, etc. we have to state that currently all results are limited to the AVR processor family. Generalization requires to rerun the experiments using different processor families.

5.3. Impact of Data types

According to [13] the energy consumption of an MPEG-algorithm was largely affected by the used data types (e.g., by replacing double by float the algorithm's energy consumption was reduced to 34% and the required number of cycles reduced to 35%. Following these ideas we are currently rerunning the 2nd experiment series using *float* instead of *integer* values. Initial results (see Figure 7) show that although data sizes were simply doubled (i.e., 2-Byte integer was replaced by 4-Byte floating point numbers), the algorithm requires significantly more energy. Interestingly, the differences are not simply correlated with a factor of two. Beneath additional memory requirements one reasons for this growth might also be the realization of the floating point unit (FPU) of the processor. AVR processors do not have a physical (i.e., hardware) FPU, but provide a software based simulation. This software FPU might be responsible for the extremely high energy requirements. In summary, the initial results indicate, that for developing energy-efficient applications it is important to select appropriate data types. This has not only an impact on stored data but also on the definition of hash functions, etc.

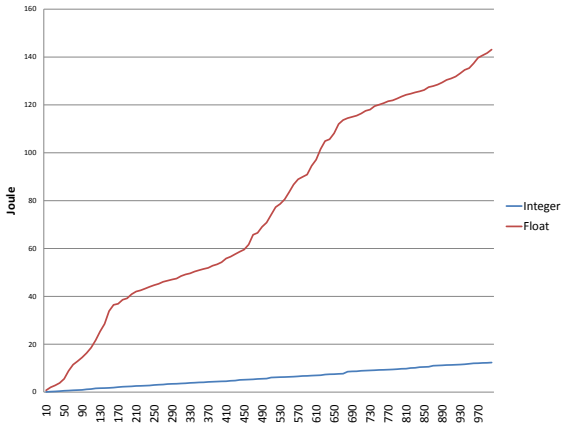

Figure 7. Insertionsort: Float vs. Integer

6. Summary, Conclusions and Outlook

The overall goal of our research is to realize mobile information systems that adapt their resource usage with regard to the users requirements. So far, such adaptations were mostly researched on the hardware level only. In this paper and its underlying experiments we analyzed the energy consumptions of sorting algorithms, as this class of algorithms is essential for managing data. Based on the findings of Marwedel [20] and our measurement results we see that memory requirements are crucial concerning energy consumption. Therefore, in-place sorting algorithms such as Insertionsort are more energy efficient than others. Furthermore, we discussed first experiments that show that not only the algorithm themselves but also the used data types influence the energy consumptions. We assume that processing floating point numbers requires more energy than processing integer values. However, more precise evaluations on this are part of ongoing research. Furthermore, we currently implement more complex algorithms like Sort-Merge-Joins and Nested-Loop-Join in order to measure their energy consumptions. In addition we plan experiments with more complex hardware platforms like ARM based systems and will also use mobile phones and PDAs in order to reach a more general understanding of the energy consumptions of algorithms.

References

[1] AVR Studio 4 . Product Website, Mar. 2009. http://atmel.com/dyn/products/tools_card.asp?tool_id=2725.

[2] Atmel Corporation, San Jose, Ca, USA. *AVR® STK500 User Guide*, Oct. 2008. available online `www.atmel.com/atmel/acrobat/doc1925.pdf`.

[3] C. Badea, A. Nicolau, and A. V. Veidenbaum. Impact of JVM superoperators on energy consumption in resource-constrained embedded systems. *ACM SIGPLAN Notices*, 43(7):23–30, July 2008.

[4] A. Bardine, P. Foglia, G. Gabrielli, and C. A. Prete. Analysis of static and dynamic energy consumption in NUCA caches: initial results. In *Proc. of the workshop on memory performance: dealing with applications, systems and architecture*, pages 105–112. ACM, 2007.

[5] B. Brejová. Analyzing variants of Shellsort. *Information Processing Letters*, 79(5):223–227, 2001.

[6] C. Bunse and H. Höpfner. Resource substitution with components — Optimizing Energy Consumption. In *Proc. of the 3rd ICSOFT*, volume SE/GSDCA/MUSE, pages 28–35, Setúbal, Portugal, July 2008. INSTICC press.

[7] J.-J. Chen and L. Thiele. Expected system energy consumption minimization in leakage-aware DVS systems. In *Proc. of the 13th ISLPED*, pages 315–320. ACM, 2008.

[8] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. M. Anderson. Quantifying the energy consumption of a pocket computer and a Java virtual machine. In *Proc. of the 2000 ACM SIGMETRICS intern. conference on measurement and modeling of computer systems*, pages 252–263. ACM, 2000.

[9] L. M. Feeney. An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. *Mobile Networks and Applications*, 6(3):239–249, June 2001.

[10] S. Gurun, P. Nagpurkar, and B. Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *Proc. of the 1st intern. workshop on decentralized resource sharing in mobile computing and networking*, pages 18–23. ACM, 2006.

[11] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[12] H. Höpfner and C. Bunse. Ressource Substitution for the Realization of Mobile Information Systems. In *Proc. of the 2nd ICSOFT*, volume Software Engineering, pages 283–289, Setúbal, Portugal, July 2007. INSTICC press.

[13] T. Hüls. Optimizing the energy consumption of an MPEG application. Master's thesis, TU Dortmund, Fakultät für Informatik, Dortmund, Germany, Mar. 2002. online available at `http://ls12-www.cs.tu-dortmund.de/publications/theses/downloads/huels.pdf.gz`.

[14] R. Jain, D. Molnar, and Z. Ramzan. Towards understanding algorithmic factors affecting energy consumption: switching complexity, randomness, and preliminary experiments. In *Proc. of the 2005 joint workshop on foundations of mobile computing*, pages 70–79. ACM, 2005.

[15] H. Koc, O. Ozturk, M. Kandemir, S. H. K. Narayanan, and E. Ercanli. Minimizing energy consumption of banked memories using data recomputation. In *Proc. of the ISLPED*, pages 358–362. ACM, 2006.

[16] S. Lafond and J. Lilius. Energy consumption analysis for two embedded Java virtual machines. *Journal of Systems Architecture*, 53(5-6):328–337, 2007.

[17] R. Lafore. *Data Structures and Algorithms in Java*. SAMS Publishing, Indianapolis, Indiana, USA, 2nd edition, 2002.

[18] D. E. Lancaster. *TTL Cookbook*. Sams, May 1974.

[19] N. Liveris, H. Zhou, and P. Banerjee. A dynamic-programming algorithm for reducing the energy consumption of pipelined system-level streaming applications. In *Proc. of the conference on Asia and South Pacific design automation*, pages 42–48. IEEE, 2008.

[20] P. Marwedel. *Embedded System Design*. Springer, 2007.

[21] O. Ozturk and M. Kandemir. Nonuniform Banking for Reducing Memory Energy Consumption. In *Proc. of the conference on design, automation and test in Europe*, pages 814–819. IEEE, 2005.

[22] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143, Feb. 2006.

[23] A. Seddik-Ghaleb, Y. Ghamri-Doudane, and S.-M. Senouci. A performance study of TCP variants in terms of energy consumption and average goodput within a static ad hoc environment. In *Proc. of the intern. conference on wireless communications and mobile computing*, pages 503–508, New York, NY, USA, 2006. ACM.

[24] S.-M. Senouci and M. Naimi. New routing for balanced energy consumption in mobile ad hoc networks. In *Proc. of the 2nd ACM intern. workshop on performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 238–241. ACM, 2005.

[25] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed java-based systems. In *Proc. of the 22nd int. conference on automated software engineering*, pages 421–424. ACM, 2007.

[26] H. Shen, M. Kumar, S. K. Das, and Z. Wang. Energy-efficient data caching and prefetching for mobile devices based on utility. *Mobile Networks and Application*, 10(4):475–486, Aug. 2005.

[27] H. Singh and S. Singh. Energy consumption of tcp reno, newreno, and sack in multi-hop wireless networks. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):206–216, June 2002.

[28] B. Sun, S.-X. Gao, R. Chi, and F. Huang. Algorithms for balancing energy consumption in wireless sensor networks. In *Proc. of the 1st intern. workshop on foundations of wireless ad hoc and sensor networking and computing*, pages 53–60. ACM, 2008.

[29] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger. A 90nm low-power FPGA for battery-powered applications. In *Proc. of the ACM/SIGDA 14th intern. symposium on field programmable gate arrays*, pages 3–11. ACM, 2006.

[30] J. Veijalainen, E. Ojanen, M. A. Haq, V.-P. Vahteala, and M. Matsumoto. Energy Consumption Tradeoffs for Compressed Wireless Data at a Mobile Terminal. *IEICE Transactions on Communications*, E87-B(5):1123–1130, May 2004.

[31] L. Wang, M. French, A. Davoodi, and D. Agarwal. FPGA dynamic power minimization through placement and routing constraints. *EURASIP Journal on Embedded Systems*, 2006(1), 2006.

[32] M. Zhang, X. Chang, and G. Zhang. Reducing cache energy consumption by tag encoding in embedded processors. In *Proc. of the ISLPED*, pages 367–370. ACM, 2007.