Energy-Efficient Sorting using Solid State Disks

Andreas Beckmann* and Ulrich Meyer
Goethe-Universität Frankfurt am Main
Frankfurt am Main, Germany
Email: {beckmann,umeyer}@cs.uni-frankfurt.de

Peter Sanders and Johannes Singler†
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {sanders,singler}@kit.edu

***Abstract*—We take sorting of large data sets as case study for making data-intensive applications more energy-efficient. Using a low-power processor, solid state disks, and efficient algorithms, we beat the current records in the JouleSort benchmark for 10 GB to 1 TB of data by factors of up to 5.1. Since we also use parallel processing, this usually comes without a performance penalty.**

***Keywords*—Energy Efficiency; Power; Sort; Solid State Disk; JouleSort Benchmark; Intel Atom; Algorithm Engineering**

I. Introduction

Computers (and their cooling devices) have become a major factor in the consumption of electrical energy. Also, battery lifetime is the main limiting factor for many applications of mobile devices. Hence, reducing energy consumption of computers is now an important economical and environmental goal.

While this has fundamentally changed the way chips are designed, there is less work on adapting the entire system, consisting of the hardware *and* the software running on it. A notable exception is the *JouleSort* benchmark introduced by Rivoire et al. [1] in 2007. JouleSort is a new category to the well-established Sort Benchmark[1], introduced by Turing award winner Jim Gray in 1985. As in the other categories of Sort Benchmark, the task is to sort 100-byte records that contain a 10-byte key. There are two classes to participate in, the highly tuned Indy class (only needs to sort the Sort Benchmark record type and may use prior knowledge about the key distribution) and the more general Daytona class (needs to be able to sort general inputs without significant loss of performance). Because of the stricter conditions, a Daytona result sets a lower bound for an Indy result. Orthogonal to the class, there are many categories, featuring different input sizes and metrics.

In the JouleSort category, the goal is to sort the input using as little energy as possible. The result is stated as sorted records per Joule. There are three size scales — 10^8, 10^9, and 10^{10} records, corresponding to 10 GB, 100 GB, and 1 TB of data.[2] We believe that JouleSort is an interesting framework for research in designing energy-efficient systems, since the sorting problem is at the same time non-trivial, simple to state, and of fundamental importance in many applications.

In this paper, we reconsider the design choices made in [1] and come to considerably different solutions, leading to a factor of up to 5.1 better performance per Joule. Instead of a large number of laptop disks, now, a small number of solid state disks (SSDs) wins. Instead of an ordinary (mobile) PC processor, we go down one scale on the ladder of processor categories and use the Intel Atom processor, which is designed for netbooks and home servers. It turns out that using parallelism, this does *not* imply significantly longer running times.[3]

Section II explains our hardware choices, Section III the algorithms used in our programs, and Section IV the implementation details. Then, Section V evaluates the performance of our approach with focus on the JouleSort benchmark. In Section VI, we interpret our results regarding their consequences for future work on energy-efficient sorting and sort benchmarks. We also discuss to what extent the results can be extrapolated to energy-efficient computing in general.

Related Work

Rivoire et al. introduce JouleSort in [1], giving excellent arguments on why energy-efficient sorting is interesting. With *CoolSort*, they present a highly efficient machine based on laptop technology, including a large number of 2.5" disks. This system is an order of magnitude more energy-efficient than systems currently used for database servers [2]. It uses the commercial *Nsort* [3] program for participating in the Daytona class. In their conclusion, the authors already predict that flash memory will play an important role in future JouleSort results. However, in [4], they report on several experiments including systems with SSDs and low-power processors, none of which beats CoolSort. Only for the mid-term[4] 2010 Sort Benchmark submission, they

*Supported in part by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, and by DFG grant ME 3250/1-2.
†Partially supported by DFG grant SA 933/3-2.

[1] http://sortbenchmark.org

[2] In this paper as in the Sort Benchmark regulations, 1 GB = 10^9 bytes, 1 TB = 10^{12} bytes, and in addition 1 MiB = 2^{20} bytes, 1 GiB = 2^{30} bytes.

[3] We do have longer running time in the 1 TB scale, but we will explain that this is a very special case.

[4] There were two Sort Benchmark deadlines in 2010, the mid-term one on January 1st, and the final one on May 15th.

Disk Model Unit	Capacity GB	Read MiB/s	Write MiB/s	Read W	Write W	Idle W	Efficiency MiB/J
SSD							
Intel X-25E	32	226	198	1.7 W	2.7 W	0.6 W	103
Intel X-25M	80	225	79	1.0 W	2.5 W	0.6 W	128
Samsung PB22-J	256	201	180	1.1 W	2.8 W	0.6 W	124
Super Talent FTM56GX25H	256	235	163	1.6 W	2.9 W	0.5 W	102
HDD							
Samsung HD502HI SpinPoint F2 EcoGreen (3.5")	500	106	108	6.6 W	6.6 W	3.7 W	16
Samsung HM500JI SpinPoint M7 (2.5")	500	87	87	2.3 W	2.3 W		28
Western Digital WD7500KEVT/00A28T0 (2.5")	750	82	82	2.0 W	2.0 W		41

Table I
ENERGY EFFICIENCY OF SSDS AND MODERN HDDS.

improve the 10 GB Daytona result from 2007 significantly by a factor of 2.1 with *FlashSort* [5]. Before, there was only a minuscule improvement for Indy 100 GB by *OzSort* in 2009 [6].

Andersen et al. [2] propose to build a Fast Array of Wimpy Nodes (FAWN) for fast server-style computing that uses AMD Geode processors and flash memory storage. It would be interesting to run the 10 GB JouleSort on one node of such a machine. Using FAWN for the larger inputs would be difficult since JouleSort requires input and output to be in a single file, and running a high performance distributed file system on FAWN sounds challenging. Moreover, FAWN uses Ethernet for communication between the nodes. It is not clear to us whether this is very energy-efficient for large networks, once one scales to a large number of nodes, because of the high bisection bandwidth required for the 1 TB JouleSort.

MapReduce has become a popular approach to large scale data processing. Its energy efficiency is considered in [7] where it is pointed out that optimizing the sorting phase inside the system is crucial for energy efficiency.

Lang and Patel [8] show how reducing clock frequencies helps to reduce the energy consumption of database queries (TPC-H style). We go one step further and propose to use low-clock rate processors optimized for energy efficiency.

II. HARDWARE CHOICE

Hardware components are usually most energy-efficient when they are fully loaded with necessary work. Usually, it is not useful to slow down operation artificially on a given device, because the energy savings do not outweigh the longer operation time. This is due to the idle power consumption, which may include underutilized peripheral components. Fully loading the system means fully loading all used components at the same time, resulting in what is called a "balanced" system in [1]. In our case, the crucial basic operations are I/O from/to non-volatile storage, performed by the disk controller and the disks, and internal computation, executed by the CPU.

In the field of non-volatile storage, a new technology has matured to become a competitor to hard disk drives (HDDs). *Solid state drives (SSDs)* combine many flash chips for HDD-like capacity. The absence of mechanical components allows for low access times and energy consumption, at least for reading. Still, for acceptable performance, accesses should be made in blocks of at least a few kilobytes, so SSDs cannot be treated as just slow RAM. *Writing* to SSDs should happen in even larger blocks, as we will explain later.

This newly available technology motivated us in tackling the JouleSort records. Together with the availability of lower-power CPUs, we saw a potential for significant improvements.

For the non-volatile storage, the criterion for comparing potential hardware is pretty clear. For a wide range of input sizes, the algorithm has to read and write every record twice (i. e. two passes). Therefore, we compare the devices by the average number of bytes they can read and write per Joule. This is justified since idle times will be avoided as much as possible, and also, the idle power consumption is low, in particular compared to the overall system.

Table I shows the average amount of data for reading and writing per Joule, for several HDDs and SSDs. The values are taken from [9], [10], [11].

The SSDs clearly outperform the HDDs, and moreover, they consume almost no power in idle mode, while the HDDs have a significant idle power consumption. The only drawback of the SSDs is their quite limited size and their high price (currently about 3 US$ per gigabyte, compared to 0.15 US$ per gigabyte for HDDs). But still, four 256 GB SSDs are sufficient even for the largest input considered here when using an in-place algorithm, and monetary cost is not included in the JouleSort metric.

The Intel X-25M looks the best theoretically, but is quite limited in size. We tried Samsung PB22-J, but they let us down with very erratic performance behavior. Instead, we chose Super Talent UltraDrive GX MLC 256GB (FTM56GX25H, firmware 1916) drives.

Independently of the choice of the disk model, the absolute I/O performance can still be varied by changing the number of SSDs attached, allowing to balance the I/O with computation.

The second important choice for JouleSort is the CPU.

However, this decision also limits the options for the close peripherals, i. e. the mainboard including the I/O controllers (or slots to inserts them) and memory slots. We figured that I/O is quite "cheap" in terms of energy consumption, so we wanted to attach as many SSDs as possible to the system. On the other hand, the number of attachable disks usually does not scale well with the power consumption. Systems that can handle very many disks (i. e. having many SATA ports or powerful slots to insert additional SATA controllers) are designed for power-hungry high-end processors, and usually have a quite high idle power, caused by power-wasting chipsets and peripherals. To compensate for this, the number of disks would have to be increased hugely, sooner or later hitting other system bottlenecks like the internal bus bandwidth, or suffering overheads from high-degree RAIDs, like the necessity of a very large block size.

Thus, we were looking for very power-saving systems that could handle a reasonable number of SSDs out-of-the-box. We decided on the Intel Atom processor because it was explicitly designed for building systems with little power overhead. Still, we had to choose between a single-core and a dual-core variant. For us, this decision turned out to be tied to the decision for the chipset and the mainboard (Atom processors are usually soldered directly on the board). To figure out what is best, we obtained two different systems, each having its particular pros and cons.

The Intel D945GSEJT mainboard comes with a single-core Atom N270 (2.5 W TDP[5]) and the low-power mobile chipset 945GSE (8 W TDP). This board's main advantage is its power-saving processor, but it features only two SATA ports, which in turn run at mere 1.5 Gibps (150 MiB/s effectively). This is too slow for our SSDs, and considering the low performance, it is odd that the chipset consumes about as much energy as the remaining system. Moreover, the maximum amount of RAM is limited to 2 GiB from a single DIMM, which is very little considered the 1 TB input.

The second system is based on a Zotac IONITX-A mainboard, equipped with an Atom 330. This processor consumes more than three times the power (8 W TDP) but features two cores and four hardware threads. The main advantage of this system is that its nVidia ION chipset provides four SATA ports that handle the SSD transfers at full speed. Moreover, it supports two DIMMs, for a total of 4 GiB of RAM. The 64-bit logical address space is less prone to fragmentation, which we experienced on the 32-bit Atom N270. A disadvantage of this chipset is that it consumes even more power than the Intel 945GSE, in part because it contains a quite powerful GPU that we are not utilizing at all.

Table II summarizes the key features of the two systems, Figure 1 shows a picture of the stronger one.

In [1], a quite exotic combination of hardware components was used, namely a mobile processor plus 13 laptop hard drives, attached to server-class RAID controllers. In contrast to this, our machines have only few components and would fit into a small ITX case. Their only extraordinary property is the use of the expensive SSDs. But note that smaller, cheaper disks could be used for the 10 GB and 100 GB inputs.

[5]Thermal Design Power

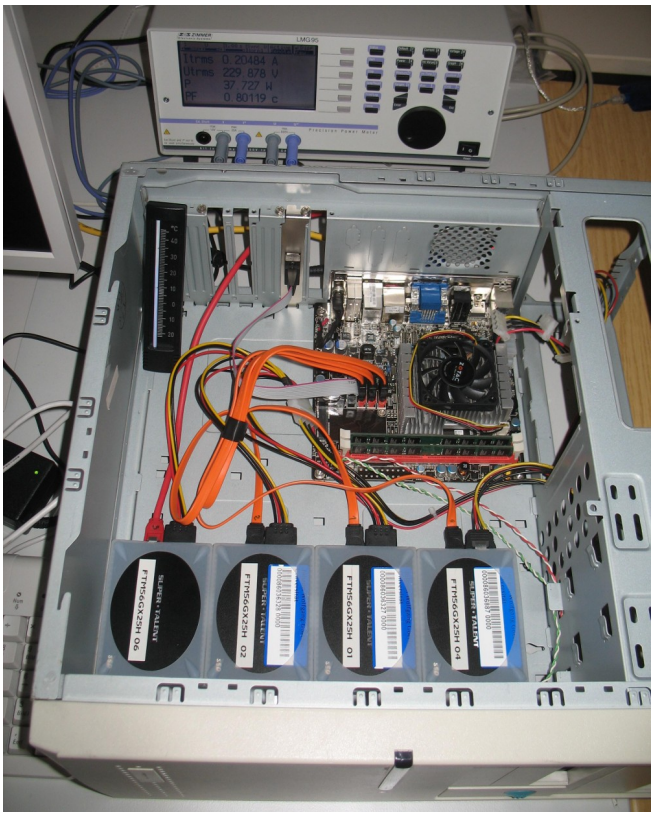

Figure 1. Our Zotac IONITX-A machine together with the power meter (displaying 37.7 W of current overall power consumption).

A. SSD Issues and Configuration

While USB flash drives and flash-based memory cards are ubiquitous nowadays, flash devices that are intended to replace HDDs have appeared only recently. These SSDs provide a capacity of several hundred gigabytes, whereas current HDDs store as much as 2 TB.

By accessing many flash chips in parallel, SSDs achieve bandwidths that are significantly higher than those of HDDs. While reading, they also feature very short access times, allowing random accesses on blocks in the kilobyte range with almost peak bandwidth. Writing is a completely different story, though. A data item can be written to only after being erased (flashed), which happens in comparatively large blocks. Thus, random writes of kilobyte-scale blocks can be very slow. Therefore, algorithms have to be adapted to this asymmetry between reading and writing [12].

Even worse, the maximum number of erase cycles is limited. To prevent early failure of the disk, the built-in *wear*

	Intel D945GSEJT		Zotac IONITX-A	
Component	**Type**	**Power**	**Type**	**Power**
Processor	Intel Atom N230 1 core, 2 threads, 1.6 GHz, x86	2.5 W	Intel Atom 330 2 cores, 4 threads, 1.6 GHz, x86_64	8.0 W
Memory	Kingston 1x2 GiB	2.5 W	Kingston 2x2 GiB	5.0 W
Chipset	Intel 945GSE	7.1 W	nVidia Ion	8.9 W
I/O	2x SATA 1.5 Gibps		4x SATA 3.0 Gibps	
Disks	2x Super Talent FTM56GX25H	4.2 W	4x Super Talent FTM56GX25H	8.4 W
Fan		1.0 W		1.0 W
OS drive		0.5 W		0.5 W
Total (behind power supply)		17.8 W		32.3 W
Total (overall)		20.0 W		36.3 W

Table II
THE TESTED HARDWARE VARIANTS. POWER CONSUMPTION FOR SORTING IS ESTIMATED BASED ON VENDOR DOCUMENTATION AND OWN TESTING (LOADED VS. UNLOADED COMPONENT AND INCLUDED VS. EXCLUDED COMPONENT). HIGHER OVERALL ENERGY VALUES ARE CAUSED BY LOSSES WITHIN THE EXTERNAL DC POWER SUPPLY (TYPICALLY RUNNING AT ABOUT 89% EFFICIENCY).

leveling methods strive to distribute the writes evenly [13]. This management can cause erratic performance behavior. Certain write patterns may lead to internal fragmentation, even causing a permanent performance degradation. Fortunately, this fragmentation can be reset by applying a low-level formatting operation (ATA SECURITY ERASE) to the disk. But in that case, of course, all data is lost. File systems can also help avoid fragmentation of the disk by reporting blocks that are not used anymore to the disk, by issuing operations called *trim* or *block discard*. However, hardware and operating system support is limited so far, e. g. the functionality is unavailable on RAID volumes.

The question arises how to treat storage devices that are unreliable in this sense in a benchmark. Best performance and reproducible results are achieved by formatting before the benchmark, but this is not a realistic model for production use. We study this question by running the test many times after formatting, evaluating whether degradation takes place.

III. ALGORITHMS

In this section, we describe the algorithms applied by our programs, which we use for the Indy results. For the Daytona results, we will use Nsort.

The amounts of data to handle do not fit into internal memory, i. e. the RAM. The rules also require both the input and the output to reside on disk.

For efficiency, we apply the *external memory* model here, which allows to transfer data from and to disk only in blocks of a certain size[6] B. We base our algorithms on multiway mergesort, which has been proven optimal for this setting [14]. It allows to sort up to $\mathcal{O}(M^2/B)$ records with two passes, where M is the internal memory size. Similar algorithms were already used several times by successful Sort Benchmark entries, e. g. *psort* [15].

The algorithm has two main phases. In the first phase, it splits the input into many *runs* of size $\mathcal{O}(M)$. In practice, the constant factor varies between 1 and 1/4, depending on the implementation. These runs are read from disk, sorted internally, and written back to disk. In the second phase, all runs are merged using a multiway merger. A *prefetch* sequence ensures that the right blocks are prefetched asynchronously, while the CPU performs the merging. Writing out the finished blocks is also overlapped with computation.

When looking closer at the algorithm, one notes that there are random block accesses only in one stage, namely when reading the blocks for merging. In contrast, run formation can transfer $\mathcal{O}(M)$ records at once. The output chunk size in writing the result in the merge phase is also only limited by the available buffer memory. So fortunately, the only operation that really needs random block access only *reads* data, a discipline that SSDs are particularly good in.

IV. IMPLEMENTATION

Our programs, namely EcoSort and DEMSort, support full overlapping of I/O and computation, in both phases using the techniques available from STXXL [16]. Due to the limited speed of the Atom processors, it was beneficial to further tune the internal sorts and merges.

In the Indy class of JouleSort, we are allowed to take full advantage of the fixed record size and the short keys that are chosen uniformly at random.

- In internal memory, the algorithm sorts only the keys, associated with an index, and permutes the sequence of full records later in parallel.
- Sorting the keys is done by applying parallelized least significant digit radix sort to the three most significant bytes of the key, and then sorting the partially sorted sequence by insertion sort. Giving prefetching hints to the distribution stage improves performance here.
- We use the parallelized function `multiway_merge` from the libstdc++ parallel mode, based on the Multi-Core Standard Template Library (MCSTL) [17].[7] The

[6] All sizes in this discussion are in number of records.

[7] The corresponding sort routine could also be used to provide fully comparison-based sorting. The overhead is significant, but not fatal.

parallel multiway merger reaches best performance if a single invocation can output at least B records. Having 3 blocks per run buffered in internal memory (2 for the merger to guarantee at least B elements available per run, the third block for prefetching) can only be done by further reducing the block size at the cost of I/O performance. Therefore, we use only 2 blocks per run and combine blocks that will run empty soon.
- We avoid repeated allocation/deallocation of memory for the large temporary buffers by using a free list.

As a consequence, the disks are working for 90% of the time, i. e. we are close to I/O-bound. Further tuning of internal computation just for the sake of consuming a little less power by the processor (by idling or reducing its clock rate) seemed not worthwhile, given the overall power consumption of the system. However, we have conducted tests on the number of threads to use (see Section V-C).

A. Sorting 10 GB and 100 GB

For the 10 GB and the 100 GB Indy category, we have developed the program *EcoSort* using the parallelized version of the Standard Template Library for XXL Data Sets (STXXL) [18].

The STXXL provides two sort implementations, `stxxl::sort()`, which writes the output back to the location of the input, and `stxxl::stream::sort()`. The latter does not need to know the number of elements to be sorted in advance, and produces output in a way that can be pipelined to further algorithmic steps. Combining the best features of these two sorters, we engineered EcoSort using direct (non-pipelined) access to the input blocks (like the standard sorter) and writing output to a new file (as usually done with the pipelined sorter). The following additional tuning measures were applied.

- Overlap loading the first run with initialization and writing the last run with cleanup operations.
- Reducing size of the first two runs to $\frac{1}{3}$ and $\frac{2}{3}$ of the regular size, thus loading the CPU earlier from the start.

B. Sorting 1 TB

For the largest scale, 1 TB of data, things get more complicated.

First of all, we have only 2.4 % more storage capacity than absolutely needed, so we have to sort *in-place*. Nsort is not capable of doing that, but overwriting the input disqualifies the result for the Daytona class anyway. There is just not enough disk space in the machine to qualify for 1 TB Daytona[8], so we limit ourselves to Indy for this size.

We used a variant of *DEMSort* [19], [20], which *is* able to sort in-place. DEMSort led the Indy GraySort and Indy MinuteSort Sort Benchmark categories in 2009. It is originally designed to run on distributed-memory clusters, but the overhead is only small when running on a single node. It was augmented with the features mentioned at the beginning of this section.

We get close to the $\mathcal{O}(M^2/B)$ limit for two passes, which we like to adhere to by all means. From the 4 GiB of RAM, the BIOS allows to use only 3.5 GiB. Subtracting the system memory usage and the program binary, we are left with only about 3.25 GiB. Due to overlapping, we need space M once for writing the last run and reading the next one, and once to store the current run, $0.16M$ for the keys (10 bytes key, 2 bytes filler, 4 bytes index), and another M for doing the permutation out-of-place. Thus, for run formation, M is limited to about 1 GiB or 10^7 records, finally resulting in $R = 973$ runs.

For the multiway merging, we need at least two buffer blocks per run, but have the full memory available, so this yields for the block size: $2BR \leq M \Leftrightarrow B \leq M/(2R) = 3.25 \cdot 10^7/(2 \cdot 973) = 16\,700$ records, or 1.67 MB. Finally, we chose a block size of 1 433 600 bytes, to allow a larger write buffer.

This configuration is very tight. Enlarging the input size, e. g. by a factor of 2, would require either more internal memory (hardware incapable), an additional pass (+50% running time), or even smaller blocks (further decreasing the effective bandwidth for merge reading).

To achieve external in-place operation, the blocks read in the merging phase must immediately be returned to the file system. In the version used for GraySort, this was achieved by creating a file *for each block* of the formed runs, and deleting it when obsolete. This incurred quite some file system overhead. Thus, we improved this by using a file *per run*, reducing the number of files by three orders of magnitude. The data is written in (block-wise) reversed order, so that in the merging phase, blocks are immediately returned to the file system after reading, by truncating the run file appropriately.

However, even if the runs are placed sequentially on disk (with respect to logical disk addressing), the output file will necessarily be fragmented. This is because the file system has to fill the gaps between the runs due to the lack of additional space. The SSDs can counter this by doing so-called *write combining*, but nevertheless, transferring a *logically* non-contiguous range will incur overheads.

To improve this situation that suffers from small blocks, we replaced the block I/O by *range I/O*, except for reading in the merge step. The program issues a single system call for reading/writing an entire run, and for writing a large chunk of the output. This is equivalent to a huge block size that amortizes all overheads. Also, this approach does not contradict the external memory model.

[8]Using 512 GB SSDs would have solved this problem, but given the additional monetary effort, this did not seem worthwhile.

Computer	Intel D945GSEJT	Zotac IONITX-A
Time	169±0.39 s	75.0±0.62 s
Energy	**3 665 J**	**2 912 J**
Average Power	21.5 W	38.6 W
Records per Joule	**27 300**	**34 300**

Table III
COMPARISON OF THE TWO MACHINES FOR 10 GB INPUTS. THESE MEASUREMENTS WERE MADE WITH A DIFFERENT POWER METER AND AN OLD SOFTWARE VERSION, AND THUS DO NOT COMPARE EXACTLY TO THE FINAL RESULTS.

Activity	230 V	19 V	Efficiency
Idle	25.3 W	22.3 W	88.4 %
Loaded	31.6 W	28.2 W	89.3 %
SSD First Write after TRIM	42.5 W	37.8 W	89.0 %
SSD Subsequent Writes	41.2 W	36.5 W	88.5 %
SSD Read	36.6 W	32.4 W	88.4 %

Table IV
EFFICIENCY OF THE POWER SUPPLY UNDER DIFFERENT LOADS (AVERAGE POWER CONSUMPTION IN A 10 MINUTE WINDOW). POWER CONSUMPTION MEASUREMENT IN FRONT OF AND BEHIND THE 19 V POWER SUPPLY WAS DONE IN SEPARATE RUNS.

V. EXPERIMENTS

The pretest subsumed in Table III shows that the D945GSEJT machine is less energy-efficient than the IONITX-A machine for the 10 GB input. We expect this margin to get even bigger for larger inputs, due to the worse I/O bandwidth and the very limited amount of RAM. Thus, we made the decision in favor of the IONITX-A machine for further experimentation.

A. System Details

We removed the included wireless module from the IONITX-A mainboard and disabled other unused hardware in the BIOS setup if possible. The machine had no keyboard or monitor attached, but was connected to the LAN and remotely controlled via secure shell (SSH). To preserve all SSD space for sorting, the OS was installed on a USB flash drive.

The Zotac IONITX-A board requires 19 V DC. We used the included 90 W power supply, which achieves about 89% efficiency, as shown in Table IV. A smaller power supply (Dehner SYS1319-3019, 30 W) actually consumed 5% more energy, despite the fact that it was better utilized.

As operating system, we employed the Debian *sid* distribution with a 2.6.33 Linux kernel. There was only a minimal set of services running, and no graphical user interface. Our programs were compiled with GCC 4.4.3, optimization switched to `-O3`. The RAID was managed in software by the *device mapper* of the Linux kernel. All partitions were formatted using XFS.

Both the hardware and the OS support Native Command Queuing (NCQ). However, pre-tests showed that this only helps for very small block sizes, so we configured EcoSort and DEMSort to not use it (only issue one request at at time). Nsort might have actually exploited NCQ, but we do not know for sure.

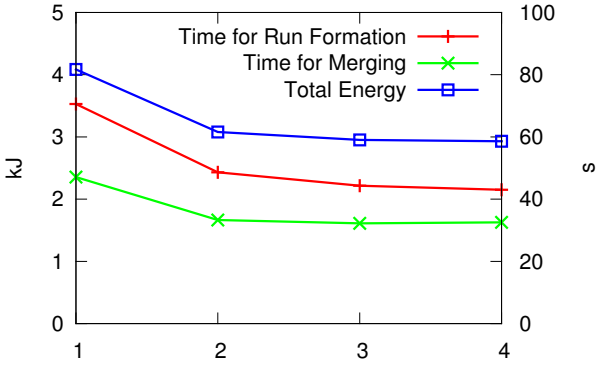

Figure 2. Evaluating the optimal number of threads.

B. Energy Measurement

For measuring the energy consumption, we used a ZES ZIMMER LMG95 precision power meter, which has an accuracy of less than ±0.1% for the applicable measurement range. This device was controlled via the serial port by the machine running the sort. The power measurement was started just before the sorting program and stopped right after, sampling power in 1-second intervals. We calculated the energy by multiplying the average power by the time reported by `/usr/bin/time` for the running time of the actual sort program. The temperature was about 23 °C throughout the tests, the CPU fan was running.

The results in time and energy consumption are averaged over five runs for each category, we also give the respective standard deviation.

All measured power values for our machine denote overall power consumption, i. e. they include losses within the DC power supply.

C. Setup

For all tests, we used four SSDs configured as a RAID-0. Sequential transfer tests showed a peak performance of about 1 000 MiB/s read and 600 MiB/s write for the RAID.

The input for all runs was generated by the *gensort* program provided by the Sort Benchmark committee, running in ASCII mode.

Before each program run, the (old) output file was removed. After each run, the output was checked for correctness, either by running *valsort* or by comparing its MD5 sum with a reference MD5 sum precomputed from a checked output.

The programs were allowed to consume up to 3.25 GiB of RAM for their operation.

For the 10 GB input, we removed one of the two 2 GiB RAM modules and allowed the programs to consume up to 1.75 GiB of RAM. This comes with a small performance penalty (running time increases up to 5 %) which is outweighed by reducing power consumption by 10 % (2.8 W idle, 3.8 W during the experiments). The run formation phase of the category was further restricted to 0.85 GiB of RAM in this case because the increased number of runs (49 instead of 24) resulted in better overlapping and lower running time without affecting the merging phase.

For 10 GB and 100 GB, we used a block size of 13 107 200 bytes.

For the TB scale using DEMSort, we lowered the block size to 1 433 600 bytes, as explained before. We newly generated the input before each run due to the lack of space, after the RAID was cleaned of all files, which implies that we newly created the temporary files and the output file each time.

For choosing the optimal number of threads, we conducted pre-tests for 10 GB. As shown in Figure 2, 4 threads for run formation have the best running time, and there is a tie between 3 and 4 threads for merging. The energy consumption goes along these lines, so we simply took 4 threads in all cases.

For Nsort (version 3.4.28, trial), we used the following command line:

```
nsort -format=size:100
-field=name:key10ascii,size:10,pos:1,char
-key=key10ascii -statistics -processes=4
-method=radix -memory=3250M
-file_system=/mnt/ssdraid0,
                transfer_size:16M,count:C
-temp:/mnt/ssdraid0/nsort/,
                transfer_size:16M,count:C
```

with `C=4` for 10 GB, and `C=2` for 100 GB.

D. Categorization

The results of our EcoSort and DEMSort programs count for the Indy class because our programs can handle only fixed-size records. Also, some optimizations rely on a certain key universe.

The Nsort program is generally acknowledged to adhere to the Daytona specifications. Hence, its results, which are a bit worse than the Indy results, count for the Daytona class.

E. Sort Benchmark Results

All Sort Benchmark results [21] are subsumed in Table V, accompanied by the respective checksums.

Overall, our Daytona results using Nsort for 10 GB and 100 GB are 6% and 11% worse than their respective Indy counterparts using EcoSort. The running time is 5% and 10% higher respectively, and the average power consumption is 1–2% higher.

Our results for **10 GB** improve over the 2007–2009 Daytona record [1] (11 600 records per Joule) by factors 3.5 (Daytona) and 3.7 (Indy), respectively. The absolute running times are also better than for those results. The achieved bandwidth of EcoSort is about 600 MiB/s averaged over reading and writing, equivalent to about 75% peak performance.

However, new competitors came into play in 2010. The mid-term 2010 Daytona entry FlashSort [22], [5] achieves 24 800 records per Joule, which is 38% less efficient than our respective result. Concerning the 2010 final entry FAWNSort [23], [24] featuring 44 900 records per Joule, the committee declared a tie (5% better) with EcoSort for the Indy category, and FAWNSort as the sole winner for Daytona (12% better than EcoSort).

However, apart from the name, there is nothing in the latter system that resembles a fast array of wimpy nodes. The authors utilize a single but powerful node with a low-power quad-core Intel Xeon processor and 12 GiB of RAM. The hardware for FlashSort is from a similar class, featuring a quad-core AMD CPU and 16 GiB of RAM. Thus, both competing entries used machines with more than 10 GB of RAM. This allows to store the full input in the main memory, needing only one I/O pass. However, in contrast to our approach, this does not scale, estimated to need about twice the time for inputs larger than the RAM size, possibly less than twice the energy, though. Adding arbitrary much RAM is usually not possible either, not to speak of the additional power consumption implied. For storage, SSDs like Intel X25-E and Fusion-IO drives are used, which are extremely expensive per capacity, and too small for a 100 GB run (Nsort needs three times the space temporarily).

For **100 GB**, our results clearly beat the 2007–2009 Daytona and Indy records [1], [6] (11 300/11 600 records per Joule) by factors 3.4 (Indy) and 3.2 (Daytona).

For the **1 TB** scale, however, the energy efficiency is far worse compared to the 100 GB scale. To find out the reason, we ran tests with the same parameter values for other input sizes in the gap between 100 GB and 1 TB,

	Daytona		
JouleSort Category (records)	**10^8**	**10^9**	**10^{10}**
Program	Nsort	Nsort	–
Memory Configuration	1 × 2 GiB	2 × 2 GiB	–
Data Volume	10 GB	100 GB	–
Number of records	100 000 000	1 000 000 000	–
Checksum	2faf0ab746e89a8	1dcd615efb9dfe11	–
Time	75.7±0.10 s	756±0.9 s	–
Energy	**2 485±4 J**	**27.94±0.03 kJ**	–
Average Power	32.8±0.02 W	37.0±0.02 W	–
Records per Joule	**40 249±59**	**35 789±44**	–
Typical Bandwidth Input Reads	230 MiB/s	240 MiB/s	–
Typical Bandwidth Temp Writes	230 MiB/s	240 MiB/s	–
Typical Bandwidth Temp Reads	330 MiB/s	300 MiB/s	–
Typical Bandwidth Output Writes	330 MiB/s	300 MiB/s	–

	Indy		
JouleSort Category (records)	**10^8**	**10^9**	**10^{10}**
Program	EcoSort	EcoSort	DEMSort
Memory Configuration	1 × 2 GiB	2 × 2 GiB	2 × 2 GiB
Data Volume	10 GB	100 GB	1 TB
Number of records	100 000 000	1 000 000 000	10 000 000 000
Checksum	2faf0ab746e89a8	1dcd615efb9dfe11	12a06cd06eeb64b16
Time	72.4±0.24 s	691±3.2 s	17 026±29 s
Energy	**2 345±9 J**	**25.09±0.11 kJ**	**571.8±1.1 kJ**
Average Power	32.4±0.07 W	36.3±0.01 W	33.6±0.03 W
Records per Joule	**42 635±168**	**39 853±183**	**17 489±33**
Typical Bandwidth Run Formation	540 MiB/s avg	600 MiB/s avg	711 MiB/s avg 867 MiB/s read 603 MiB/s write
Typical Bandwidth Merge	610 MiB/s avg	600 MiB/s avg	151 MiB/s avg 187 MiB/s read 127 MiB/s write

Table V
SUMMARY OF THE SORT BENCHMARK RESULTS.

namely 248 GB[9], 500 GB, 750 GB, and 875 GB, and so on, approaching 1000 GB. We even extended beyond the boundary of 1 TB by also evaluating 1006 GB and 1023 GB of input. The results stated in Figure 3 show that the degradation is correlated to the filling degree of the SSDs. The curve bends sharply starting from 875 GB, indicating that filling the SSDs almost completely is inadvisable. For 1023 GB of input, the shown result even deteriorates further for runs after first one, resulting in catastrophic performance. The performance degradation must be caused by increasing internal fragmentation, as explained in Section II-A. The I/O rate drops dramatically after run formation, when the merger has to fill the gaps on disk, namely from 603 MiB/s to 151 MiB/s (averaged over read and write). At least, for 1 TB, there is no further degradation for multiple runs in a row, even when a multiple of the disk capacity is written in total.

We still beat the previous record [1] (3 425 records per Joule for Indy) by a factor of 5.1. However, the cited result was not achieved using the low-power system mentioned before, but with solid server hardware accessing 12 disks and consuming 400 W. Thus, we cannot compete in terms of running time, our program takes 2.4 times as long as the competitor.

We had even worse results for the I/O bandwidth degradation when generating the input using regular OS-buffered I/O. Instead, we wrote 1 GB blocks to the RAID by piping the `gensort` output through `dd oflag=direct,nonblock obs=1G`. We consider this justified because one cannot blame the sorter for a non-optimally generated input.

Ultimately, the terabyte result is an artifact of the space scarceness in our system. Using twice as large disks would most likely have provided the performance of the 500 GB run, which is almost 30 000 rec/J, and would beat the previous record by a factor of more than 8, needing only 33% more absolute running time. We hope to achieve a similar result by using the block discard feature of the file system, as soon as this is supported for RAID volumes.

[9]We chose this rather odd number because it is the 2009 PennySort record, see Section VI-B.

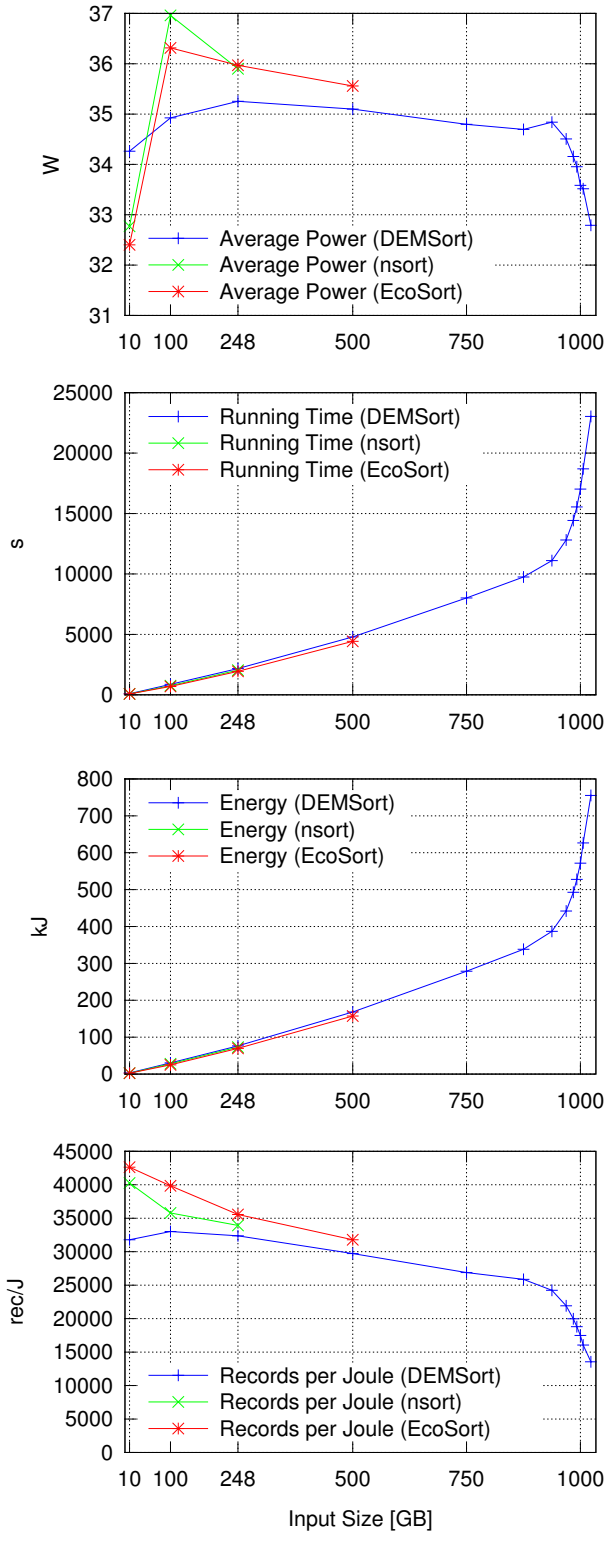

Figure 3. Performance for inputs of increasing size.

VI. Conclusions and Future Work

We have demonstrated that carefully chosen hardware together with tuned, parallelized sorting algorithms can sort large amounts of data considerably more energy-efficiently than previous systems. Most of the energy gain comes from the newer hardware components, selected after analyzing the performance requirements of the algorithms. The tuned software allowed us to improve their quite good balance a bit further.

The results presented here improve over our mid-term 2010 entries [25], and add Daytona results. Four of the five entries [21] lead their respective category after the final 2010 submission deadline, including the mentioned tie.

Future work will include experiments with trim-capable SSD firmware, in particular for the 1 TB scale. First tests have shown a big potential, the only problem remaining that trim does not cooperate with RAID on current Linux kernels.

In the following, we discuss some longer term implications of our results.

A. Implications for JouleSort

Storage devices: The trend to solid state memory instead of mechanical disks is probably there to stay. However, we have seen that the wear leveling algorithms of current SSDs are not yet mature enough to give predictable performance when the disks are close to full, and no trimming is performed. It is also interesting to note that at least for writing, the block sizes needed for efficient operation approach the sizes needed for mechanical disks at least if we use striping. Algorithmically, we could perhaps gain by using each SSD or even each flash chip as an independent disk. However, benchmark rules, file systems, and SSD controllers hinder this approach.

Processors: For physical reasons, you can get more energy efficiency by using processors with lower clock frequency that can work with lower voltage. We have demonstrated this by going down one step on the ladder from a laptop processor to one intended for netbooks and home servers. It is well possible that going down further to processors for mobile devices would allow to sort even more records per Joule. However, there is a danger that this process could render JouleSort unrealistic and tedious to work on because the absolute running times would become very large. Also, measuring power consumption of potentially battery-backed devices could lead to results that are hard to testify.

Parallelism: Parallelism can compensate the decreased speed of the individual processors. For example, our system is not slower than CoolSort [1], due in part to the fact that we use four parallel threads on two cores (and also use highly tuned algorithms). However, JouleSort gives no incentives for achieving low execution time and there is always some overhead for moving data, which is likely to become quite visible for multi-socket boards, NUMA machines, or even

clusters. Therefore, if one wants JouleSort also to cover large systems, one might consider to introduce a time limit, e. g., of one hour. This would probably have the effect that different hardware would be used for different input sizes, e. g., mobile devices for 10 GB, home servers for 100 GB, and clusters for 1 000 GB. Note that such a spread was intended by the original design of JouleSort [1]. Our system, by winning in all three categories, shows that this spread currently does not quite work. In order to enforce large machines, it also makes sense to look at larger inputs. Even 10 TB could now be sorted on a single-processor machine with eight 1.5 TB disks.[10]

Other components: The main reason why we did not use the even more energy-efficient smaller Atom N270 processor (2.5 W) was not its lower performance but the lack of an appropriate infrastructure. The chipset Intel 945GSE that we tried needs three times as much energy as the processor and supports only two SATA ports, which are also too slow for modern SSDs. It is likely that similar mismatches could be observed for mobile devices, which currently are not built for achieving high I/O bandwidth.

More generally, it seems likely that chipsets, network adapters, switches etc. will turn out to be crucial for the progress of energy-efficient sorting. The current trend to energy-efficient home servers is likely to improve the situation eventually. However, the Intel Pine Trail platform and the ICH9R chipset, which is available in the meantime, is quite disappointing in this respect. Its major drawback is that the disk controller is usually attached using a 1x PCIe link, which cannot handle the parallel bandwidth of several SSDs.

B. Relation to Cost-Efficient Sorting

PennySort is a different category of the Sort Benchmark. There, the program is allowed to use computing time equivalent to one (US$) cent, amortizing the machine over three years. However, the benchmark does not take into account energy cost. This was a reasonable assumption in 1998 when this benchmark category was first used. The hardware used for the 1999 record[11] [26] (Pentium II 333 MHz, 23.8 W maximum power consumption) certainly consumed less than 100 W, and had been running for only 917 s because of its comparatively high price, resulting in an energy cost of less than 0.2 cent (assuming 8 cent/kWh at the time[12]).

However, recent PennySort machines cost less than half the price of the 1999 machine and thus run more than twice as long for a penny. They have similar power requirements, but meanwhile, the energy price has risen by 50 %. Even if we assume that the 2009 PennySort leader *psort* is as efficient as the 2009 leaders in JouleSort (Daytona and Indy), the energy cost is almost four times as high as in 1999, approaching the amortized hardware cost (see Table VI). Assuming EcoSort's power efficiency, the cost of 0.25 cent is close to the result of 1999, but the machine used in this paper was much more expensive (about 3500 US$), so you cannot carry that over. With growing cost for electricity, it is quite likely that the energy cost will eventually dominate.[13]

Hence, it would be interesting to consider a PennySort-like benchmark that takes both hardware cost and energy cost into account. In this situation, home server hardware equipped with laptop disks might become an interesting competitor. It consumes less energy and may eventually cost less than the desktop systems used previously. We have demonstrated that the performance of a processor like the Intel Atom is adequate even with fast SSDs.

C. Scaling to Even Larger Inputs

Since DEMSort is already capable of parallel operation on a distributed-memory machines, the results are likely to scale to a cluster built of the machine type proposed here. The only further requirement would be a fast communication device, which delivers half of the bandwidth of the disks full-duplex, i. e. about 300 MiB/s in our case. Even if the power consumption would double or triple to include the network hardware, the records per Joule performance would still be very competitive. In comparison, we estimate the performance of the 100 TB GraySort leading result in 2009, which is using DEMSort, to sort only about 1 000 records per Joule (195 nodes $\times$ 450 W plus 2 kW for the Infiniband switch for 3 hours).

D. Beyond Sorting

If we extrapolate from our JouleSort results to other applications, we could consider whether hardware designed for home servers might not also be useful for professional server applications. In particular, for applications amenable to parallelization, we could hope for considerably lower energy consumption at the same overall performance. We would get servers with lower energy consumption, higher achievable processor density and low cost for the processors. SSDs would be used in systems with high requirements on I/O performance, or where small disks suffice. Otherwise, one could use mechanical disks without increasing the energy consumption too much.

Of course, we have to be careful with overgeneralization. So it is interesting to discuss what may change with other applications. For more I/O-bound applications, it makes even more sense to use slower processors and fast SSDs. For CPU-bound applications, it *also* makes sense to use

[10] The committee has actually reacted on our report by introducing a new 100 TB JouleSort category. This is aimed on clusters, as it is allowed to use multiple input and output files, just as in GraySort.

[11] There is no documentation on the 1998 record.

[12] Tariff information is based on the average US prices for residential customers taken from the Energy Information Administration http://www.eia.doe.gov/cneaf/electricity/epm/table5_3.html

[13] One could argue that energy prices for commercial customers are lower. However, this effect is compensated by the fact that hardware in computing centers needs to be cooled.

	HM-Sort 1999 (Daytona)	CoolSort 2007 (Daytona)	OzSort 2009 (Indy)	EcoSort (Indy)	psort (Indy)
Input Size	2.6 GB	248 GB			
Performance		11 300 rec/J	11 600 rec/J	35 558 rec/J	
Power	*<100 W*			35.9 W	
Time	917 s			1939 s	2211 s
Energy	*<97 kJ*	*221 kJ*	*216 kJ*	69.7 kJ	
Amortized system cost	1 cent				1 cent
Energy cost	*<0.20 cent*	*0.74 cent*	*0.72 cent*	0.25 cent	

Table VI
POWER CONSUMPTION FOR 248 GB (2009 PENNYSORT RECORD), ASSUMING A 12 CENT/KWH TARIFF (8 CENT/KWH FOR THE 1999 RESULT). VALUES WRITTEN IN *italics* ARE (BASED ON) ESTIMATIONS AND/OR EXTRAPOLATIONS.

processors that need less energy per instruction, as long as we can meet the performance requirements with increased parallelism. Most applications differ from a highly tuned sorter by consisting of many layers of software that is less highly tuned and often is not even in native code. In many cases, such code may suffer less than highly tuned code when going to weaker machines (less cache, less superscalarity) because such programs often do not need a big cache and are not very effective on using instruction parallelism anyway. Going to weaker machines is not a good idea if the application is both time-critical and has not been parallelized. However, with the advent of multicore computing, there is a lot of pressure to parallelize such applications anyway. What remains are applications that are time-critical and hard to parallelize, applications that need the big caches/main memories of high-end machines, and applications that do a lot of number crunching. The latter applications might move to yet other machine categories. For example, the most powerful supercomputers like IBM Blue Gene are also using low clock frequency to achieve better energy efficiency. Furthermore, applications with large memory requirements might be able to move to lower category machines at some point since Moore's law is still working for us in that respect.

To turn the above speculations into good advice, more research on energy-efficient computing, in particular more benchmarks are needed. There are the EnergyBench benchmarks (http://eembc.org/benchmark/power_sl.php) that add an energy efficiency category to each of several benchmark categories and the *SPEC Power* benchmark (http://www.spec.org/power_ssj2008/) on a transaction processing workload. Although such benchmarks yield good guidelines for evaluating hardware, we believe that much higher savings will be possible by optimizing hardware and software together. Here, we need more benchmarks like JouleSort that only specify the application interface and the inputs but neither the program nor the programming language. The transaction processing performance council (TPC) is in the process of extending its benchmarks to take energy efficiency into account (http://www.tpc.org/tpc_energy/default.asp) but there are no published results yet. Poess and Nambiar [27] analyze the power efficiency of the TPC-C benchmark.

REFERENCES

[1] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis, "JouleSort: a balanced energy-efficiency benchmark," in *ACM SIGMOD International Conference on Management of Data*, 2007, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247522

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: a fast array of wimpy nodes," in *22nd ACM Symposium on Operating Systems Principles*, 2009, pp. 1–14.

[3] C. Nyberg, C. Koester, and J. Gray, "Nsort: a parallel sorting program for NUMA and SMP machines." [Online]. Available: http://www.ordinal.com/NsortPara.pdf

[4] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza, "Models and metrics to enable energy-efficiency optimizations," *IEEE Computer*, vol. 40, no. 12, pp. 39–48, 2007.

[5] J. D. Davis and S. Rivoire, "FlashSort: JouleSort benchmark entry, 2010 Daytona 10 GB class," 2010, Sort Benchmark mid-term 2010 submission. [Online]. Available: http://sortbenchmark.org/flashsort_2010_Jan_01.pdf

[6] R. Sinha and N. Askitis, "Ozsort: Sorting 100 GB for less than 87 kJoules," 2009. [Online]. Available: http://sortbenchmark.org/OzJoule2009.pdf

[7] Y. Chen and T. X. Wang, "Energy efficiency of map reduce," 2008, UC Berkeley. [Online]. Available: http://www.eecs.berkeley.edu/~ychen2/cs262a/EnergyEfficientMapReduceReport-Final.pdf

[8] W. Lang and J. M. Patel, "Towards eco-friendly database management systems," in *4th Biennial Conference on Innovative Data Systems Research*, 2009.

[9] B. Feddern, "Platten-Karussel," *c't: Magazin für Computer Technik*, vol. 10/2009, pp. 104–107.

[10] ——, "Entdeckungsreise," *c't: Magazin für Computer Technik*, vol. 11/2009, pp. 100–104.

[11] ——, "Platten-Karussel," *c't: Magazin für Computer Technik*, vol. 21/2009, pp. 142–145.

[12] D. Ajwani, A. Beckmann, R. Jacob, U. Meyer, and G. Moruz, "On computational models for flash memory devices," in *8th International Symposium on Experimental Algorithms (SEA)*, 2009, pp. 16–27. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02011-7

[13] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138–163, Jun. 2005.

[14] A. Aggarwal and J. S. Vitter, "The Input/Output complexity of sorting and related problems," *Communications of the ACM*, vol. 31, no. 9, pp. 1116–1127, 1988.

[15] P. Bertasi, M. Bressan, and E. Peserico, "psort, yet another fast stable sorting software," in *8th International Symposium on Experimental Algorithms (SEA)*, 2009, pp. 76–88. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02011-7

[16] R. Dementiev and P. Sanders, "Asynchronous parallel disk sorting," in *15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, San Diego, 2003, pp. 138–148.

[17] J. Singler, P. Sanders, and F. Putze, "The Multi-Core Standard Template Library," in *Euro-Par 2007: Parallel Processing*, pp. 682–694.

[18] A. Beckmann, R. Dementiev, and J. Singler, "Building a parallel pipelined external memory algorithm library," in *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2009.

[19] M. Rahn, P. Sanders, J. Singler, and T. Kieritz, "DEMSort — distributed external memory sort," 2009. [Online]. Available: http://sortbenchmark.org/demsort.pdf

[20] M. Rahn, P. Sanders, and J. Singler, "Scalable distributed-memory external sorting," in *26th IEEE International Conference on Data Engineering (ICDE)*, 2010, pp. 685–688.

[21] A. Beckmann, U. Meyer, P. Sanders, and J. Singler, "Energy-efficient sorting using solid state disks," May 15, 2010, Sort Benchmark final 2010 submission. [Online]. Available: http://sortbenchmark.org/ecosort_2010_May_15.pdf

[22] J. D. Davis and S. Rivoire, "Building energy-efficient systems for sequential I/O workloads," Tech. Rep., Mar. 2010. [Online]. Available: http://research.microsoft.com/pubs/121686/JouleSort_TRv1.0.pdf

[23] V. Vasudevan, L. Tan, D. Andersen, M. Kaminsky, M. A. Kozuch, and P. Pillai, "FAWNSort: Energy-efficient sorting of 10 GB," May 15, 2010, Sort Benchmark final 2010 submission. [Online]. Available: http://sortbenchmark.org/fawnsort_2010.pdf

[24] V. Vasudevan, D. G. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru, "Energy-efficient cluster computing with FAWN: Workloads and implications," in *1st International Conference on Energy-Efficient Computing and Networking (e-Energy)*, Apr. 2010, pp. 195–204, invited paper.

[25] A. Beckmann, U. Meyer, P. Sanders, and J. Singler, "Energy-efficient sorting using solid state disks," January 1, 2010, Sort Benchmark mid-term 2010 submission. [Online]. Available: http://sortbenchmark.org/ecosort_2010_Jan_01.pdf

[26] B. Helmkamp and K. McCready, "HM-Sort: 1999 performance / price sort and pennysort," 1999. [Online]. Available: http://sortbenchmark.org/HMsort_1999PennySort.doc

[27] M. Poess and R. O. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results," in *34th International Conference on Very Large Data Bases*, 2008, pp. 1229–1240.