psort 2011 – pennysort, datamation, joulesort [*]

Paolo Bertasi, Federica Bogo, Marco Bressan, Enoch Peserico
Univ. Padova, Italy - psort@dei.unipd.it

Abstract

This memo reports the results of our *psort* (general purpose) sorting software on a number of hardware configurations. “Vanilla” *psort* sorted 10GB for 2122 joules on a Nokia N900 smartphone with an extra Kingston 16GB flash drive (joulesort 10GB Daytona+Indy benchmark). It sorted over 286.2 GB for 0.01$ of computing time on a PC equipped with an AMD 2.7GHz Sempron processor, 4 GB of RAM and 5 Samsung SpinPoint F4 HD322GJ drives (pennysort Daytona benchmark); on the same hardware, a hand tailored version of *psort* sorted over 334.4 GB for 0.01$ (pennysort Indy benchmark). Finally, a cluster version of *psort* sorted 100 MB in 40.5 milliseconds from disk to disk (and in 55 milliseconds from platter to platter) on a 16 PC cluster with 96 Samsung SpinPoint F3 HD103SJ drives and 10Gb/s Ethernet (datamation benchmark).

1 Introduction

This memo describes the *psort* entry for the 2011 sorting benchmarks – more specifically pennysort (both Indy and Daytona), datamation, and joulesort (10GB Daytona). Section 2 briefly describes the structure of *psort* (for a more in-depth description see [3, 2]). Section 3 describes the pennysort benchmark results. Section 4 describes the datamation benchmark results. Section 5 describes the joulesort benchmark results.

2 *psort*

psort is a fast, stable external sorting software originally designed for PC class platforms. The experiments described in this memo show, however, that it can also run surprisingly well on small embedded platforms and can be easily extended to run on small-to-medium sized PC clusters. Its architecture is described in detail in [2, 3]; however, in this section we briefly review its internals.

In a nutshell, *psort* is a “classic” (deterministic and stable) merge-based sorter; it sorts files viewed as sequences of (at most 2^{64}) records of arbitrary

[*]This work was support in part by project AACSE, and in part by Samsung through a generous equipment donation.

size (up to 2^{64} bytes), according to an arbitrary infix, by first sorting individual "runs" of data approximately the size of the main memory and then merging those runs into a single sorted run. *psort* is entirely merge-based even in the first phase, using $k-$way mergesort (with adaptive k) to sort the initial runs. Note how this is a marked difference from most other Pennysort competitors, that first distribute the records into small buffers according to a (usually 16 bit long) prefix of the key and then sort those buffers according to a variety of algorithms, sometimes randomized. The rationale behind *psort*'s purely merge-based implementation is providing a deterministic behavior: this guarantees stability and high performance even against adversarially chosen inputs. *psort* can use disks independently (as described e.g. in [4]), or as a single RAID logical volume. Furthermore, *psort* can efficiently exploit multicore processors; however, with the record footprint of the sorting benchmarks (10 byte keys with 90 bytes of payload), a single core always appears more than adequate for its needs.

3 Pennysort

The pennysort benchmark involves sorting as many 100 byte records as possible (according to the first, random 10 bytes), in 0.01$ of computer time (assuming the cost of a machine is amortized over three years). "Indy" software can be specifically optimized for the pennysort record format and key distribution; "Daytona" software must be general purpose sorting software.

This section describes our hardware, filesystem and OS choices for the pennysort benchmark (Subsection 3.1), the "Indy" optimizations – taylored to the hardware and the specific key distribution of the pennysort benchmark (Subsection 3.2) – and the actual test results for both the Daytona and the Indy version of *psort* (Subsection 3.3).

3.1 Hardware, filesystem and OS

Hardware – at least the parts critical to efficient external sorting – has not considerably improved in terms of price-performance in the last two years. Disk performance has increased somewhat; RAM and motherboard performance less so; the processor critical statistics (size and access speed of the various levels of cache) have remained almost unchanged. On the other hand, prices have slightly increased across the board.

After various tests, we selected as the best disk candidates the Hitachi HDS721050CLA362 500GB and the Samsung SpinPoint F4 HD322GJ 320GB. The Hitachi disks had on their side an extremely low price of 39$, and a slightly larger size of 500GB (meaning speed remained closer to the top speed achievable on the outer rim throughout the sort). The Samsung disks were, however, some $15-25\%$ faster on the outer rim and, perhaps more importantly, appeared more reliable (exhibiting far less performance variation over time and between different disks).

We paired our disks with the cheapest processor we could find (a Sempron Sargas 140 2.7 GHz), an ASRock 880 gm-le motherboard, and 2 banks of 2GB DDR3 1333 MHz 8-8-8-24 RAM.

We installed Gentoo GNU/Linux on our system (compiled with Huge Pages support), and created a number of XFS RAID 0 (stripe size: $128KB$) partitions, starting from the (faster) portion of the disk closer to the outer rim; each partition was just large enough to hold our input and output. Note how this size was different for the Daytona and Indy tests (see below); the number of partitions was also different – 2 for the Indy test and 3 for the Daytona test (for the latter, the new 2009 rules forbid overwriting the input). [2] motivates our choice of XFS as a filesystem. We also disabled any "smart" disk scheduling.

We compiled *psort* using gcc version 4.4.5.

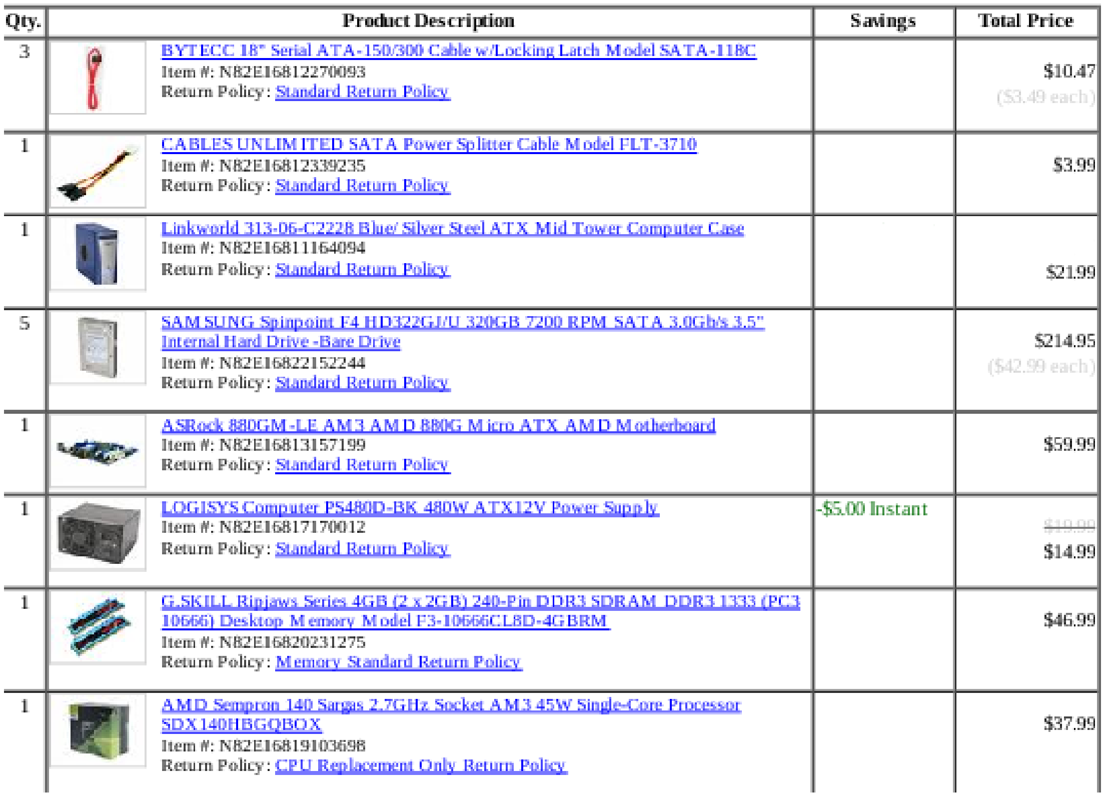
3.2 Indy optimizations

In an effort to push *psort* as far as it can go when hand-optimized for a specific machine and architecture, we retooled it into an Indy version optimized for the pennysort benchmark. We hard-coded all parameters (record and sort size, key position and length etc.), and re-coded the first phase so as to actually distribute records into bins according to the initial 16 bits of the key before actually sorting each bin. This latter sort uses "standard" *psort* merge-based code, with one catch: it only operates on the first 8 bytes of each key. Because of the random distribution of the keys, it is extremely unlikely for two keys to actually coincide on the first 8 bytes; we leave the check that this is really the case to the second phase of the sort, where the computational pressure is lower. We actually have code in place to deal with identical key prefixes and guarantee the final output is sorted, but it never gets invoked for the current sort sizes.

3.3 Pennysort results

A 451.36$ cost yielded a 2096 seconds time budget (see figure 3.3). In this time, "vanilla" *psort* sorted and merged 91 runs of 31457280 records each, for a total of 2862612480 records (of 100 bytes each) and a Daytona sort size of slightly more than 286.2 GB. This was achieved starting with the input (generated by the **gensort** software from the sort benchmark site) on the innermost (slowest) partition, writing the temporary output of the first phase on the outermost (fastest) partition, from which it was read during the second phase to produce a final output on the middle partition. For checking purposes, the sum of the 32 bit CRCs of all records was $55 : 50 : 23 : c7 : f7 : de : b3 : cc$.

In the same time, Indy *psort* sorted 116 runs (of 28835840 records of 100 bytes each), for a sort size of slightly more than 334.4 GB. This was achieved positioning the input on the faster, outermost partition, overwriting it with the temporary output at the end of the first phase, and writing the final output into a second, slower partition immediately closer to the spindle. For checking purposes, the sum of the 32 bit CRCs of all records was $63 : b0 : 3d : 80 : 31 : 9f : 72 : fa$.



Qty.	Product Description	Savings	Total Price
3	BYTECC 18" Serial ATA-150/300 Cable w/Locking Latch Model SATA-118C Item #: N82E16812270093 Return Policy: Standard Return Policy		$10.47 ($3.49 each)
1	CABLES UNLIMITED SATA Power Splitter Cable Model FLT-3710 Item #: N82E16812339235 Return Policy: Standard Return Policy		$3.99
1	Linkworld 313-06-C2228 Blue/ Silver Steel ATX Mid Tower Computer Case Item #: N82E16811164094 Return Policy: Standard Return Policy		$21.99
5	SAMSUNG Spinpoint F4 HD322GJ/U 320GB 7200 RPM SATA 3.0Gb/s 3.5" Internal Hard Drive -Bare Drive Item #: N82E16822152244 Return Policy: Standard Return Policy		$214.95 ($42.99 each)
1	ASRock 880GM-LE AM3 AMD 880G Micro ATX AMD Motherboard Item #: N82E16813157199 Return Policy: Standard Return Policy		$59.99
1	LOGISYS Computer PS480D-BK 480W ATX12V Power Supply Item #: N82E16817170012 Return Policy: Standard Return Policy	-$5.00 Instant	~~$19.99~~ $14.99
1	G.SKILL Ripjaws Series 4GB (2 x 2GB) 240-Pin DDR3 SDRAM DDR3 1333 (PC3 10666) Desktop Memory Model F3-10666CL8D-4GBRM Item #: N82E16820231275 Return Policy: Memory Standard Return Policy		$46.99
1	AMD Sempron 140 Sargas 2.7GHz Socket AM3 45W Single-Core Processor SDX140HBGQBOX Item #: N82E16819103698 Return Policy: CPU Replacement Only Return Policy		$37.99

Figure 1: The newegg price listing for our hardware. Note that we are not taking into account any instant savings for the final price calculation

4 Datamation

Datamation is the “original” sorting benchmark [1]: benchmarks such as pennysort, minutesort, joulesort [5, 7] etc. are all derived from it, and maintain its record format (a 10 byte key followed by an incompressible 90 byte payload). Datamation involves sorting 1 million such records in as little time as possible. When datamation was first proposed as a benchmark of transaction processing power, over twenty years ago, the sorting time was of the order of an hour. Ten years ago, it had already dropped to less than a second, becoming a benchmark of a system’s responsiveness rather than of its sustained throughput.

Still, responsiveness is an extremely important characteristic for many data management systems (webserver-linked databases, online game servers, automatic trading systems etc.) and one that none of the other sorting benchmarks currently addresses. Thus, while the focus of datamation may have shifted, it still remains a crucial sorting benchmark [6].

4.1 Hardware, filesystem and OS

We tested *psort* on a 16 PC Linux cluster provided by Univ. Padova Project AACSE. Each machine is equipped with 6 Samsung Spinpoint F3 HD103SJ (1TB) drives, generously donated by Samsung. These drives, exhibiting a peak transfer rate close to 150 MB/s, were the fastest 7200 rpm drives in mid 2010, when the cluster was set up. Each node runs a Nehalem core i7 950 CPU at 3.07Ghz, supported by an ASUS P6T SE motherboard with 3 banks of 4 GB tri-channel DDR3 running at 1.6 GHz. The cluster is connected by a dedicated 10Gb/s Ethernet through Myricom 10G-PCIE-8B-S cards and a Fujitsu XG2600 switch.

The operating system is Gentoo Linux – with a 2.6.36 kernel, patched with Con Kolivas’s (CK) kernel patch set. The drives are configured as a RAID 0 with 1024 KB stripe size, with maximum (2048 blocks) readahead, and XFS as a filesystem. On sequential reads, they saturate the ICHR10 southbridge of the motherboard when reaching the value of approximately 680 MB/s.

Perhaps more important for the datamation benchmark is the disk access latency. Each disk takes approximately 8.3 milliseconds to complete a revolution. According to specifications, the disk head can typically access a given track in less than 5 milliseconds “on average”. In theory, if one could align a read in such a way that it was all positioned on a single track (over 1.2 MB of data on the outer rim), this would entail reads of up to 1.2 MB per disk in less than 15 millisecond. However, such positioning is extremely difficult to achieve (due to the variable number of bytes per track) and in practice, we have seen slightly longer access times to read or write 1 MB on each disk (over 20 milliseconds). On the other hand, transfering data to/from the hard disk on-board cache is definitely faster: about $2-3$ milliseconds plus a time proportional to the transfer time at the peak bandwidth (i.e. about $11-12$ milliseconds to transfer $6MB$ from or to a RAID of 6 disks).

4.2 Cluster psort

Like virtually all cluster sorting software, cluster *psort* reads samples data from each node, determines a partition of the (sorted) data between the various nodes, sends each record to the appropriate destination node, and finally sorts and writes the record to disk. Data can actually be partitioned into a number of subsets that is a multiple of the number of nodes; this allows data that reaches each node to be already partially sorted, minimizing computational pressure and allowing the destination node to pipeline network I/O, computation, and disk I/O. In the case of the datamation benchmark, to minimize latency we replaced the portion of the code that determines the partition of the sorted data on a simple, static partition based on the random distribution of the keys; data were partitioned into 256 sets, with 16 sets assigned to each node.

The *psort* process is launched through command-line on one of the 16 nodes. Every node in the cluster listens on a specified UDP port for commands from the rest of the cluster, executing them through a `fork()` and an `exec()` – essentially a very minimal, but "general purpose" shell that allows the master node to create *psort* processes on the rest of the cluster. After reading and partitioning the data from its local RAID, each of the 16 nodes goes through 15 phases. During the i^{th} phase, the j^{th} node sends to the $((i+j)\texttt{mod}n)^{th}$ node "its" records – and signals to the $((j-1)\texttt{mod}n)^{th}$ node that it has done so, allowing it to enter the next phase. This simple barrier mechanism is crucial to avoid expensive collisions on the network.

Once every node has received all the records partitioned in 16 sets (with all records in each set smaller than all records in the next) it sorts and sends to disk each set. After this, each node signals the master node it has finished; the master process terminates after receiving such an acknowledgement from every node.

4.3 Datamation results

psort was compiled with `gcc` version 4.4.4, with flags `-O3 -mtune=native -march=native -funroll-loops -funsafe-loop-optimizations`.

To minimize the execution time, we placed the data on a small partition close to the outer rim of each disk, and before launching *psort*, we *read* the data once. This ensured that all the data was already in the disk onboard cache when we launched *psort* (there was slightly more than 1MB of data per disk). Similarly, each disk was configured with `hdparm -W1` to ensure that every write command returned as soon as the data was on the onboard cache of the disk, and not necessarily on the physical platter.

Out of 200 trial runs, the minimum was slightly below 40.5 milliseconds and the median slightly below 40.8 (see figure 4.3). This "cache-to-cache" time is approximately the interval between two consecutive TV frames, and, to the best of our knowledge, over an order of magnitude less than the last recorded datamation result [6].

While the onboard cache is part of the disk – and thus the "cache-to-cache"

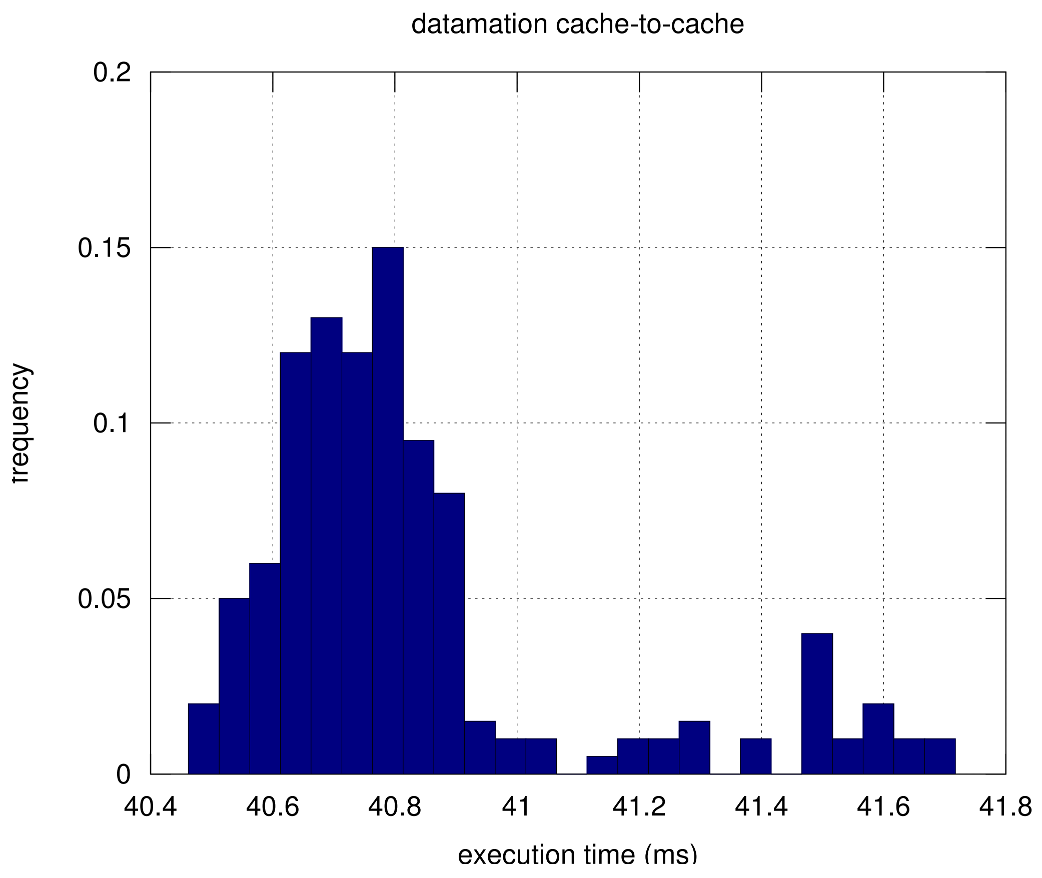


Figure 2: Datamation time distribution, sorting from and to the disk onboard cache

time *is* the disk-to-disk time of the Datamation benchmark, we also measured the “platter-to-platter” sort time, by flushing the disk onboard caches (with dummy reads) before launching *psort*, and by configuring the disks with `hdparm -W0` to ensure that no write command would return before the data was on the platter. This led to a somewhat larger variance in the sort times; out of 200 trial runs, the fastest took slightly less than 55 milliseconds to complete, the median 57 (see figure 4.3). For checking purposes, the sum of the 32 bit CRCs of all records was $7 : a1 : 9c : ff : 46 : 74 : 38$, and 0 duplicate keys were encountered when the output was (automatically) verified to be in sorted order.

5 Joulesort

To test its versatility, we also decided to test “vanilla” *psort* on a computing platform that was completely different from that for which it was originally

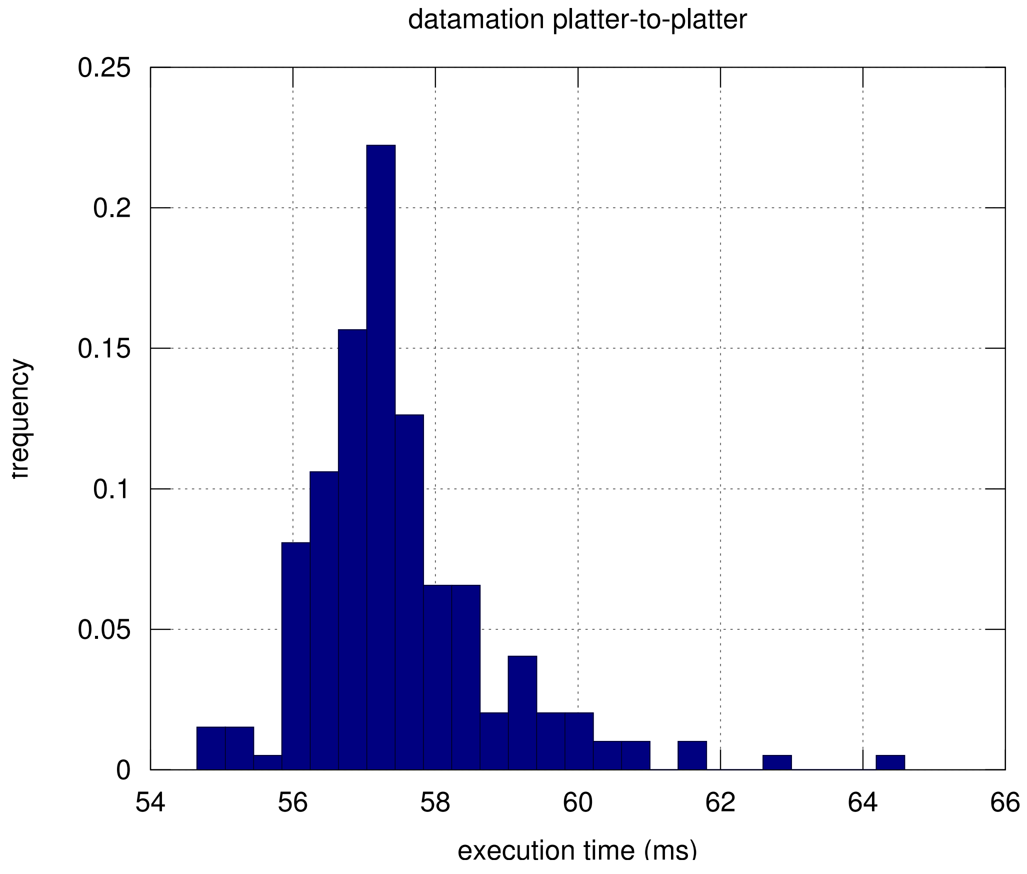


Figure 3: Datamation time distribution, sorting from and to the physical platter

designed: namely, a Nokia N900 smartphone. Surprisingly, *psort* managed to sort 10GB of data for slightly less energy than the 2010 joulesort record.

5.1 Hardware, filesystem and OS

The Nokia N900 is based on an ARM 7 Cortex A8 processor running at 600 MHz, Maemo 5 Linux, and an internal 32GB flash drive. We found that we had about 100MB of memory space available for running *psort*. We added a 16GB Kingston class 10 flash drive in the Nokia N900 expansion slot. We used ext3 as a filesystem for the internal card and vfat for the microSDHC. We cross compiled *psort* with the Scratchbox toolchain.

5.2 Test setup

We tested the total energy consumption with a Yokogawa WT 130 wattmeter (0.5% RDG + 0.3% MG error), sampling at 200 millisecond intervals.

Interfacing the Nokia N900 with the wattmeter was not easy – the main means of communication of the Nokia is its wireless system, which can drain a considerable amount of energy, distorting the measurement. Instead, we opted to have the Nokia N900 launch *psort* after a brief sleep period, emitting a beep immediately before beginning the sort, and a second beep immediately after completing it. To emit the beep, the sound system must be activated, creating a small energy spike that can be detected before the actual beep is emitted. Thus, the wattmeter starts measuring on the energy spike, is verified to be already measuring when the first beep is emitted, and stops measuring immediately after the second beep. This provides an upper bound to the total energy consumption (introducing an error that is still probably below 0.1% - less than 2 seconds, over more than 40 minutes of sorting time).

5.3 Joulesort results

We measured time and energy of 5 different *psort* trial runs, with all the subsystems of the Nokia N900 not instrumental to the sort (screen, wireless etc.) turned off. The average time was 2583 seconds. The average energy was 2122 joules (with an instrument error of at most 17 joules) and a standard deviation of 2.22 joules.

S1 time (s)	S1 energy (Wh)	S2 time (s)	S2 energy (Wh)	total energy (Wh)	total energy (J)
1430	0.305	1152	0.2842	0.5892	2121
1435	0.3060	1151	0.2839	0.59	2124
1438	0.3067	1149	0.2834	0.5901	2124
1430	0.305	1150	0.2837	0.5887	2119
1430	0.305	1152	0.2842	0.5892	2121

References

[1] A measure of transaction processing power. *Datamation*, 31(7):112–118, 1985.

[2] P. Bertasi, M. Bressan, and E. Peserico. psort, yet another fast stable sorting software. In *SEA*, pages 76–88, 2009.

[3] P. Bertasi, M. Bressan, and E. Peserico. psort, yet another fast stable sorting software – in press. *Journal of Experimental Algorithmics – in press*, 2011.

[4] R. Dementiev and L. Kettner. Stxxl: Standard template library for xxl data sets. In *In: Proc. of ESA 2005.*, pages 640–651.

[5] J. Gray. A measure of transaction processing 20 years later. *CoRR*, abs/cs/0701162, 2007.

[6] F. I. Popovici, J. Bent, B. Forney, A. Arpaci-dusseau, and R. Arpaci-dusseau. Abstract datamation 2001: A sorting odyssey, 2002.

[7] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD Conference*, pages 365–376, 2007.