

# *psort* 2009

Paolo Bertasi, Marco Bressan, Enoch Peserico  
Univ. Padova, Italy - [psort@dei.unipd.it](mailto:psort@dei.unipd.it)

## Abstract

This memo reports the results of our *psort* (general purpose) sorting software on a 5 disk 427.86\$ PC with 4 GB of RAM and an Athlon LE 1640 2.6 GHz processor. *psort* sorted  $67 \cdot 2^{25}$  records in 2211 seconds, for a Daytona sort size slightly larger than 224.8 GB. A version of *psort* hand-optimized for the Pennysort benchmark sorted  $74 \cdot 2^{25}$  records in the same time and on the same hardware, for an Indy sort size slightly larger than 248.3 GB.

## 1 Introduction

This memo describes the *psort* entry for the 2009 Pennysort (Indy and Daytona) sort benchmarks. Section 2 describes *psort* - both the “stock” Daytona version designed for general purpose sorting with strong worst case guarantees, and the Indy version specifically optimized for the Pennysort benchmark. Section 3 describes the PC on which the tests were performed. Section 4 describes the actual results of the tests.

## 2 *psort*

*psort* is a fast, stable external sorting software for PC class platforms that established the 2008 Pennysort Daytona sort record. It is described in detail in [1] (in press at the date of this writing - the preprint version can be found at <http://www.dei.unipd.it/~bertasi/psort>), but this section briefly reviews its internals.

In a nutshell *psort* is a “classic” (deterministic, stable) merge-based sorter that sorts files viewed as sequences of (at most  $2^{64}$ ) records of arbitrary 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; this is a marked difference from most other Pennysort competitors, that first distribute the records into small buffers according to a (usually 16 bit) 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 deterministic behavior that guarantees stability and high performance

even against adversarially chosen inputs. The same design goal motivates the choice of using parallel disks as RAID in all phases of the sort, rather than as “independent disks” (as described e.g. in [2]).

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 (with “standard” *psort* merge-based code).

### 3 Hardware (and OS)

We were somewhat disappointed by the fact that 2009 hardware did not show marked improvements and/or price drops compared to the hardware we had tested in 2008. The total cost of our test platform (including the “virtual” 35\$ assembly fee) at Newegg.com on March 31<sup>st</sup> was 427.86\$, slightly higher than last year’s (see end of memo). In particular, the best price/performance disks were the same as last year’s - Western Digital’s 160GB, 7200 rpm Caviar SE WD1600 AAJS with a peak transfer rate of approximately 100GB/s, at a price over 90% of last year’s price.

We paired a Gygabyte GA-MA74GM-S2 motherboard with a cheap, 2.6 GHz, AMD Athlon LE 1640 with 1MB L2 cache (at approximately the same price that we paid last year for the 2.4 GHz version - now unavailable) more for their price than for their performance. While this motherboard can support up to six disks, only five of ours could be fully exploited without saturating the bandwidth of the southern bridge - a sixth disk did not yield enough speed benefits to be worth the extra price. Also, this motherboard/processor combination (as the vast majority of AM2, rather than AM2+, architectures) successfully supports RAM only up to 800 MHz, rather than 1066, which makes RAM a slight bottleneck during the first phase of the sort. Still, better architectures (from the point of view of the memory and disk bandwidth - the CPU in itself was not a bottleneck) entailed multi-core processors and a quantum leap in price that, again, was not justified by sufficient increases in performance.

RAM was the only component that was significantly better than what was available last year - we could afford 2 banks of 2GB Dual Channel PC6400 (800MHz) RAM for approximately the same price paid last year for a pair of 1GB banks with the same characteristics. It is perhaps interesting to observe that this year’s market fluctuations meant that the RAM size doubled, while the processor’s L2 cache remained constant - contrary to the “average” trend that has the two quantities grow by approximately the same factor each year, and in particular has the size of the L2 cache (measured in cache lines) grow faster than the square root of the size of the RAM. This increased the pressure on the main memory bus and slightly increased the advantage of the distribution-based approach we implemented in our Indy version. We remark again that this is just a fluctuation - we expect the pressure on the main memory bus to decrease “on average” in the coming years.

We installed Gentoo GNU/Linux on our system (compiled with Huge Pages support), and created a number of XFS RAID 0 (stripe size: 1024KB) partitions, starting from the (faster) portion of the disk closer to the outer rim, each just large enough to hold our input and output. This size was different for the Daytona and Indy tests (see below); the number of partitions was also different, and in particular 2 for the Indy test and 3 for the Daytona test (for the latter, the new 2009 rules forbid overwriting the input). [1] motivates our choice of XFS as a filesystem. We also disabled any “smart” disk scheduling.

## 4 Pennysort results

A 427.86\$ cost yielded a 2211.19 seconds time budget. In this time “vanilla” *psort* sorted and merged 67 runs of  $2^{25}$  records each, for a total of 2248146944 records (of 100 bytes each) and a Daytona sort size of slightly more than 224.8 GB - approximately 24% more than last year (far below the average 60% yearly improvement of the last 10 years). 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. This meant that the performance of disks was slightly worse during the first phase, but since during this phase the RAM was (for a very small margin) the bottleneck, in the end we witnessed essentially the same performance that could be achieved overwriting the input and making heavier use of the outer portions of the disk. For checking purposes, the sum of the 32 bit CRCs of all records was  $43 : 00 : 33 : dd : 4d : b0 : 5f : 68$ , and 0 duplicate keys were encountered when the output was (automatically) verified to be in sorted order.

In the same time Indy *psort* sorted 74 runs (also of  $2^{25}$  records of 100 bytes each), for a total of 2483027968 records and a sort size of slightly more than 248.3 GB - approximately 31% faster than last year. 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  $4a : 00 : 26 : 68 : 09 : 7b : 2a : 29$ , and 0 duplicate keys were encountered when the output was (automatically) verified to be in sorted order.








## References

- [1] P. Bertasi, M. Bressan, and E. Peserico. *psort*, yet another fast stable sorting software. In *Proc. Symp. on Experimental Algorithms (SEA)*, pp. 76–88, 2009.
- [2] R. Dementiev and P. Sanders. Asynchronous parallel disk sorting. In *Proc. ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pp. 138–148.



**Shopping Cart**

[Print](#)

Qty.	Product Description	Savings	Total Price
3	 <a href="#">OKGEAR 18" SATA II Premium Round Cable, Support up to 3 GB/S Model OK18ARS11 - Retail</a> Item #: N82E16812123180 Return Policy: <a href="#">Standard Return Policy</a>		\$5.97 (\$1.99 each)
2	 <a href="#">Link Depot 15PIN SATA power Y cable Model Pow-SATA-Y - Retail</a> Item #: N82E16812189161 Return Policy: <a href="#">Standard Return Policy</a>		\$4.98 (\$2.49 each)
1	 <a href="#">Linkworld 72521-2121U+P04 Black/Silver ATX Tower Computer Case 430W Power Supply - Retail</a> Item #: N82E16811164123 Return Policy: <a href="#">Standard Return Policy</a>		\$32.99
5	 <a href="#">Western Digital Caviar SE WD1600AAJS 160GB 7200 RPM SATA 3.0Gb/s Hard Drive - OEM</a> Item #: N82E16822136075 Return Policy: <a href="#">30 Day Return Policy</a>		\$209.95 (\$41.99 each)
1	 <a href="#">GIGABYTE GA-MA74GM-S2 AM2+/AM2 AMD 740G Micro ATX AMD Motherboard - Retail</a> Item #: N82E16813128342 Return Policy: <a href="#">30 Day Return Policy</a>		\$54.99
1	 <a href="#">Kingston 4GB (2 x 2GB) 240-Pin DDR2 SDRAM DDR2 800 (PC2 6400) Dual Channel Kit Desktop Memory Model KVR800D2K2/4GR - Retail</a> Item #: N82E16820134730 Return Policy: <a href="#">Limited Non-Refundable 30-Day Return Policy</a>		\$37.99
1	 <a href="#">AMD Athlon 64 LE-1640 Orleans 2.6GHz Socket AM2 45W Single-Core Processor Model ADH1640DHBOX - Retail</a> Item #: N82E16819103239 Return Policy: <a href="#">Processors (CPUs) Return Policy</a>		\$45.99
Subtotal:			\$392.86
<b>Calculate Shipping</b> Zip Code: <input type="text" value="UPS Guaranteed 3 Day Service"/>		Shipping:	\$0.00