MendSort: Sorting 1 TB using less than 60K Joules

Igor Mendelev, Levi Mendelev, Yonathan Mendelev
mendsort@gmail.com
December 2023

1. Introduction

We describe a custom-built system that runs the JouleSort 1 TB benchmark (Daytona and Indy categories) using modern desktop-class hardware with 8 NVMe Gen4 SSDs and a 12-core 65 W AMD Ryzen 7900 CPU. It is able to perform a 1 TB sort with an average sort time of 304.4 seconds using an average of 195 W. **This equates to a total average energy use of 59,259 joules, which corresponds to 168,750 sorted records per joule.** This result is 2.33x more energy efficient than the RezSort (2021 Daytona) record [1], and 1.06x more energy efficient than the ELSAR (2022 Indy) record [2]. The sorting speed for our system is 4.53x faster than the RezSort record, and is 2.03x faster than the ELSAR record. Our system is also significantly less expensive than the previous record-holders: 21.6% cheaper than the system used for RezSort and 26.6% cheaper than the system used for ELSAR.

2. Hardware

Type	Model	Qty.	Total Cost
Motherboard	Gigabyte X670E AORUS PRO X	1	$320
CPU	AMD Ryzen 9 7900 (65 W TDP)	1	$370
RAM	G.Skill Trident Z5 RGB 96 GB (2 x 48 GB) DDR5-6400 CL32	1	$290
SSD	WD SN850X 2 TB	5	$540
SSD	WD SN850X 1 TB	3	$225
PCIe card	ASUS Hyper M.2 x16 Gen5	1	$85
Power supply	Corsair RM750x SHIFT 750 W 80+ Gold (with active PFC)	1	$110
CPU cooler	Thermalright Phantom Spirit 120 SE	1	$35
Case	Corsair 4000D Airflow ATX Mid Tower	1	$80
System cost			**$2,055**

Note: Photo of the system was taken with the glass side panel removed, but all tests were conducted with the panel installed. The Watts Up Pro power meter is located on top of the case, near the top right corner of the image.

Storage

The system has 8 SN850X NVMe SSDs. Of the 4 NVMe M.2 slots on the motherboard, 2 slots are directly connected to the CPU (1 x 2 TB and 1 x 1 TB SSD), and another 2 slots are connected through the X670E chipset (1 x 2 TB and 1 x 1 TB SSD). We also utilize all 4 NVMe M.2 slots on the ASUS Hyper M.2 Gen5 card (3 x 2 TB and 1 x 1 TB SSD). The ASUS Hyper M.2 card is connected to the primary PCIe Gen5 x16 slot on the motherboard. We enabled x4x4x4x4 bifurcation for the x16 slot in the BIOS, allowing all 8 SN850X SSDs to be used simultaneously at Gen4 speed. The NVMe speeds were verified using the *lspci -vv* command (*LnkSta* values).

Memory
The RAM is run at 5200 MT/s, which is the default for the Ryzen 7900.

Cooling
The system has four total fans: three 120mm fans and one small fan built into the PCIe card. One of the 120mm fans was installed inside the Thermalright CPU cooler, and the other two were attached inside the front panel of the case (see image above). All the fans were running throughout the sort. The ambient temperature was ~68° F.

Additional hardware details can be found at the PCPartPicker link [3].

3. Software

Operating System
The system was running Rocky Linux 9.3 OS [4] with kernel version 5.14.0-362.8.1.el9_3.x86_64, with the Balanced power profile (pre-defined in Rocky Linux 9.3).

Sorting
We used a trial version of Nsort 3.4.61 [5], specifying radix sort for the in-memory sorting. We ran Nsort with the following parameters:

```
-processes=24
-memory=72000M
-method=radix
-format=size:100
-field=name:key,size:10,off:0,character
-key=key
-statistics
-in_file=/data2/src/joule10B.txt,direct,transfer_size=64M
-out_file=/data2/dest/joule10Bnt.txt,direct,transfer_size=128M
-temp=/data1/tmp,direct,transfer_size=128M
```

We used the *gensort* utility to generate both regular and skewed unsorted 1 TB text files. Sorted files were validated using the *valsort* utility after each run.

Storage Configuration
The SSDs are divided into two separate storage volumes. One volume (data1) consists of 3 x 1 TB NVMe drives, and a 1 TB partition of a 2 TB drive. The other volume (data2) consists of 4 x 2 TB drives. For data1, we utilized an XFS filesystem on top of a striped LVM volume. For data2, we utilized an XFS filesystem on top of the RAID0 software array using the *mdadm* utility. The input file to be

sorted and the sorted output file were stored on the data2 volume. Temporary files created during Nsort runs were stored on the data1 volume. Optimal *transfer_size* values for this system and workload were determined empirically. We used the *fstrim* utility before each sort run to achieve optimal performance.

4. Measurement

Power meter

We used the Watts Up Pro power meter [6] to gather power usage data. Our desktop computer was connected to power through the Watts Up Pro AC outlet. The Watts Up Pro was connected to an M2 MacBook Air using the USB-A cable included with the Watts Up Pro and a USB-A to USB-C adapter. Using the open source *wattsup.py* Python utility [7], power usage data for each run was logged to a CSV file in 1 second intervals. We used the CSV files to calculate the energy usage for each sort run. The results were confirmed using the integrated Watts Up Pro display, which was set to show aggregate energy usage from the start of each sort run with 3-digit precision (0.1 Wh).

Timing

Both the system and MacBook Air were running *ntpd,* and the time discrepancy with time.gov throughout the runs was always 30 milliseconds or less, which is negligible compared to the length of the runs (~304 seconds). Each run was started and ended with the *date* command e.g.

```
date; time nsort -processes=24 ... ; date
```

Nsort itself was always run prefixed with *time* (i.e. *time nsort* ...) to show the number of milliseconds and CPU usage stats for that run. The start and end of the time intervals, used to compute average power for each run, were based on the output of the two *date* commands for that run. The runtime for each run (from the output of *time nsort* ...) was rounded up to the nearest tenth of a second, then multiplied by the average power, which was computed from that run's captured power meter logs. We discarded the fractional first and last seconds' power measurements for each run to ensure we didn't underestimate the average power; the resulting increase in the computed average power was negligible (<0.1%) for all runs.

5. Results

A typical 1 TB sort run used ~59 KJ (16.5 Wh), and took around 5 minutes 4 seconds for a regular (non-skewed) data file. The sorting time on the skewed data (generated using *gensort -a -c -s*) was 6 minutes 20 seconds, 1.22x slower than our regular (non-skewed data) performance, and used ~72 KJ (20 Wh).

Run #	**Time (s)**	**Power (W)**	**Energy (J)**
Run 1	304.9	194.7	59,364
Run 2	304.6	194.6	59,275
Run 3	304.5	194.5	59,225
Run 4	304.4	194.7	59,267
Run 5	303.7	194.8	59,161
Mean	**304.4**	**194.7**	**59,259**
SD	0.45	0.12	74
Skewed	380	189.6	72,048

Runtimes (as reported by the *time* command) are rounded up to the nearest tenth of a second.

The following is the statistics output generated by Nsort during Run 5:

```
Nsort version 3.4.61 (Linux-X64) using 27G of memory out of 70G
Pointer sort (radix) performed Sun Dec 31 13:17:29 2023
          Input Phase       Output Phase         Overall
Elapsed   181.54             122.06             303.60
I/O Busy   41.32     0%       55.21    46%       96.53
Action  User   Sys Busy    User   Sys Busy    User   Sys Busy
sort    3520 61.74 1973%   2754 52.80 2300%   6275   114 2105%
  Rssmax     Majflt    Minflt  Sort Procs Aio Procs/QueueSize RegionKB
113641.91M     0/12     17825          24         0/12            512
File Name                   ModeCntTran  Busy   Wait MB/sec  Xfers        Bytes     Records
Input Reads
  /data2/src/joule10B.txt    dir 4x64m    23%   1.05   5590  14902 1000000000000 10000000000
Temporary Writes
  /data1/tmp                 dir 10x128m  25%   0.00   5563   7630 1000002207744
Temporary Reads
  /data1/tmp                 dir 10x128m  34%   0.00   8267   7630 1000002207744
Output Writes
  /data2/dest/joule10Bnt.txt dir 4x128m   46%   0.69   8262   7451 1000000000000 10000000000
```

Based on the Nsort statistics output, the input phase of the sort (reading the input file, sorting runs of records, and writing those runs to the temporary files) took 182 seconds (60% of the total time), and the output phase (reading the runs from the temporary files, merging them, and writing the result to the output file) took 122 seconds (40% of the total time). Our system was CPU-bound with an average CPU usage of 88% (2105% out of a maximum 2400%). During the input phase, the average CPU usage was 82% (1973% out of 2400%), and both the disk read and write speeds were ~5.6 GB/s. During the output phase, the average CPU usage was 96% (2300% out of 2400%), and both the disk read and write speeds were ~8.3 GB/s.

We observed, both on the Watts Up Pro's built-in display during the test runs, and in the captured logs, power factor values in the 0.98-1.00 range, consistent with the power supply specifications (active PFC 0.989 @ 115 V).

We didn't observe any significant average power usage variations between the sort run phases in the logs (especially on non-skewed data), most likely because all of the CPU cores and NVMe disks were near their maximum power usage.

6. References

1. Reda, W., & Kostic, D. (2022, February 24). *RezSort: Sorting 1TB using Energy-efficient NVMe SSDs*. Sort Benchmark. Retrieved December 26, 2023, from http://sortbenchmark.org/RezSort2021.pdf
2. Kristo, A., Pillai, P., & Kraska, T. (n.d.). *Designing an energy-efficient, learning-enhanced algorithm to sort 1TB of ASCII data*. Sort Benchmark. Retrieved December 26, 2023, from http://sortbenchmark.org/ELSAR2022.pdf
3. Mendelev, L. (2023, October 30). System Parts List. PCPartPicker. Retrieved December 26, 2023, from https://pcpartpicker.com/list/jL4Myg
4. Rocky Enterprise Software Foundation. (n.d.). Rocky Linux. Retrieved December 26, 2023, from https://rockylinux.org/
5. Ordinal Technology Corp. (n.d.). Ordinal Technology - Nsort Home Page. Retrieved December 26, 2023, from http://www.ordinal.com/
6. Watts Up Pro. (n.d.). Vernier. Retrieved December 26, 2023, from https://www.vernier.com/files/manuals/wu-pro.pdf
7. Lui, P. (2023, January 2). wattsup.py. GitHub. Retrieved December 26, 2023, from https://github.com/paklui/wattsup