FAWNSort: Energy-efficient Sorting of 10GB

Padmanabhan Pillai, Michael Kaminsky,
Michael A. Kozuch
Intel Labs Pittsburgh

Vijay Vasudevan, Lawrence Tan
David Andersen
Carnegie Mellon University

1 Introduction

In this document, we describe our submission for the 2011 10GB JouleSort competition. Our system consists of a machine with a low-power desktop processor and seven flash drives, sorting the 10GB dataset in 15.55 seconds ($\pm$0.03s) seconds with an average power of 92.0W ($\pm$0.57W). This system sorts the 10GB dataset using only 1430 Joules ($\pm$9.9 J), providing 69944 ($\pm$489) sorted records per Joule. This improves energy used by 36% and sorted records per Joule by 56% when compared to the winning 2010 Daytona/Indy entry.

Our entry for the 10GB competition tries to use the most energy-efficient platform we could find that could hold the dataset in memory to enable a one-pass sort. This is the same approach taken in our 2010 entry, but with new, faster system components and drives. We decided to use a one-pass sort on this hardware over a two-pass sort on more energy efficient hardware (such as Intel Atom-based boards) after experimenting with several energy efficient hardware platforms that were unable to address enough memory to hold the 10GB dataset in memory. The low-power platforms we tested suffered from either a lack of I/O capability or high, relative fixed power costs, both stemming from design decisions made by hardware vendors rather than being informed by fundamental properties of energy and computing.

2 Hardware

Our system uses an Intel® Core™ i5-2400S, a 2.5 GHz quad-core processor (no hyperthreading, but TurboBoost-enabled) paired with 16GB of DDR3-1333 DRAM (4x 4GB DIMMS). The mainboard, a Gigabyte GA-H67A-UD3H-B3 based on the Intel® H67 chipset, provides 2x 6-Gb/s SATA, 3x 3-Gb/s SATA, and 1x eSATA ports.

We use the built-in processor graphics, and use the x16 PCIe slot for a HighPoint RocketRAID 640 card that provides an additional 4x 6-Gb/s SATA ports. As this slot connects directly to the processor, this configuration can allow the system to exceed the 20Gb/s theoretical limit of the DMI v2 bus, which connects all of the other slots and ports in the system. Unfortunately, the low-cost RAID card has its own internal bandwidth bottlenecks, and can only provide close to full bandwidth from two of its SATA ports.

For storage, we use 7 SATA-based Intel® 510 series SSDs, each with 120 GiB capacity. We connect 5 to the onboard SATA ports, and two to the RAID card. These drives are rated at 450 MB/s sequential read, and 200 MB/s sequential writes. We were able to achieve this in practice when the drives are connected to the onboard 6-Gb/s SATA ports. The 3-Gb/s SATA ports limit read speed to around 250 MB/s, while the RAID-card ports limit reads to about 375 MB/s. Full write speed was achieved on all of the ports.

Rounding out the hardware, we use an Antec EarthWatts 380W ATX power supply. This is a Bronze 80 Plus rated power supply, but it is unlikely that our very low power draw (typically $<$100W) is in the high-efficiency range of the supply. Our system is placed on a desk in an office/cubicle environment (not mounted in a chassis). For cooling, the power supply has an internal fan, and we use the stock heatsink and fan that comes with the retail cpu package. These provide sufficient cooling for sustained operation of all components at load in the 68–72^{o} F ambient office air.

We use a stock configuration in the BIOS, only changing the boot options, and enabling AHCI rather than legacy IDE mode for the onboard SATA ports. In particular, no overclocking, voltage tweaking, or fan control options were modified.

2.1 System price and power

All of the hardware components are commercially available and were purchased through online retailers (Amazon and Newegg). The motherboard ($140), processor ($200), memory (4x$40), and powersupply ($45) total $545. The RAID card ($180) and SSDs (7x$315) add another $2385. The total system price is under $3000, and is dominated by the flash storage components.

The total power consumption of the system peaks at about 100 W during some of the sort runs, but averages only 92 W during our best configured experiments. In other CPU-heavy tests, we see power dissipation as high as 107 W. The processor itself is rated at 65W TDP, including the built-in graphics pipelines, but in practice, we do not see power consumptions that high. Our system, including all 7 disks idles at about 46.5 W. Of this, about 6 W is due to the RAID controller.

```
nsort -processes=3 -memory=1500M
      -method=radix
      -format=size:100
      -field=name:key,size:10,off:0,character
      -key=key
      -statistics
      -file_system=/raid0,direct,
                   transfer_size:576K,count:30
      -out_file=/raid1/output,direct,
                   transfer_size:28M,count:30
      /raid0/inputdata
```

Figure 1: NSort Parameters Used

3 Software

All of our results are using Ubuntu Linux version 10.10 with its default kernel version 2.6.35. We only needed to compile and install a kernel module to enable the RAID controller card. We use a small partition on one drive formatted as ext4 as the OS/boot partition. For the main data storage, we create software RAID-0 sets across all of the drives (configuring the RAID controller to simply show the raw drives), with 64KB chunk size and formatted with XFS. Because Linux uses each of the partitions in a RAID set equally, and we get very different read bandwidth from drives on the different SATA ports, we experimented with adding an extra partition on the drives connected to the two fastest ports (9 partitions total on the 7 drives) to maximize read throughput. As the write throughput is drive limited, all of the SATA ports look alike, so a simple RAID set of 7 partitions on the 7 drives maximizes wrtie speed. We will show results with and without this read speed tweak.

We use the provided `gensort` utility to create the 10^8 100-byte records and use the provided `valsort` to validate our final output file. For the actual sorting, we use a trial version of NSort software (`http://www.ordinal.com`) with the parameters shown in Figure 1. We note that the transfer size for reads is set to the stripe set size (9*64=576KB, or 7*64=448KB, depending on the RAID configuration used). Also note that the runs never actully fully utilized all 4 cores, loading the system to around 300%. We are therefore able to tell Nsort to only use 3 threads without hurting performance, while obtaining very modest (0–2%) improvements to total energy.

Similar to previous entries that used NSort to compete for JouleSort [1, 2], we meet the 2011 designation for the Daytona category because NSort is a general sort software package.

4 Measurement

We measure the energy consumption during our sort experiment using a WattsUp Pro .NET power meter ([3]) specified as accurate to within 0.1%. We connect the power meter to our test machine using the onboard USB interface and use publicly available software for the power meter to log the power readings once per second. For each run, our execution script first starts the logging software, waits a few seconds for power measurements to start appearing in the log file, then runs the nsort command, waits for the sort to complete, and then terminates the power logging. The script inserts sort start and end messages into the power log file, so correlating the correct power measurements with the experiment is not a problem. Our script uses `/usr/bin/time` to measure and report the actual runtime of NSort.

Using the logs, we calculate the energy consumed by averaging the power values that are measured once per second over the duration of the run and multiplying that average power by the runtime reported by `/usr/bin/time`. We have to be careful in computing the average power over a run, since the initial and final 1-second power measurement intervals may only have the sort benchmarking running for parts of the intervals. We compute average power by discarding the two lowest power measurements of the relevant measurements intervals. For example, for our 15.6s experiment, we use the highest 14 values to average the power, ignoring the two lowest (i.e., first and last) values of the 16 pertinent entries. We use this calculated average power and multiply by the actual runtime of the experiment to calculate the total number of joules.

5 Results

Our results are summarized in the tables below. We first use the simple 7-disk RAID-0 configuration for the input and output files.

	Time (s)	Power (W)	Energy (J)	SRecs/J
Run 1	16.00	91.4	1462.4	68381
Run 2	16.01	92.1	1474.1	67838
Run 3	16.00	92.1	1474.1	67836
Run 4	16.05	91.8	1473.9	67846
Run 5	16.01	92.4	1478.6	67633
Run 6	15.93	92.6	1475.0	67796
Avg	16.00	92.1	**1473.0**	**67888.4**
Error	0.04	0.42	5.49	254.28

The statistics reported by Nsort during these runs indicate around 285% CPU utilization, 1700 MB/s, and 6.0s for the read phase, and 295% CPU utilization, 1060 MB/s, and 9.5s for write phase. `/usr/bin/time` reports 0.5–0.6s longer total run time than Nsort itself. As mentioned above, we use the reported number from `/usr/bin/time` to calculate the duration of the sort.

We next consider the read bandwidth optimized version, which takes advantage of the fact that the drives connected to the onboard 6-Gb/s SATA ports can produce nearly twice the read rate as those on the 3-Gb/s ports. We use a 9-partition RAID-0 set that puts an extra partition on each of these faster

drives to hold the input data. As write performance is balanced, we use the 7-partition RAID-0 set used previously for the output file.

	Time (s)	Power (W)	Energy (J)	SRecs/J
Run 1	15.53	92.1	1429.8	69942
Run 2	15.53	90.9	1410.9	70877
Run 3	15.53	92.4	1434.9	69693
Run 4	15.54	92.2	1433.1	69778
Run 5	15.55	92.0	1430.4	69912
Run 6	15.60	92.3	1439.7	69461
Avg	15.55	92.0	**1429.8**	**69943.7**
Error	0.03	0.57	9.91	488.85

The statistics reported by Nsort during these runs indicate around 315% CPU utilization, 1900 MB/s, and 5.5s for the read phase, and 295% CPU utilization, 1060 MB/s, and 9.5s for write phase. Here, too, `/usr/bin/time` reports 0.5–0.6s longer total run time than Nsort itself.

Our system improves upon the April 2010 Daytona and Indy winner by 36% in terms of energy, and 56% in terms of records sorted per Joule.

Acknowledgments

We would like thank Phil Gibbons and Guy Blelloc for their insightful discussions on sorting and potential bottlenecks.

References

[1] J. D. Davis and S. Rivoire. Building energy-efficient systems for sequential workloads. Technical Report MSR-TR-2010-30, Microsoft Research, Mar. 2010.

[2] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A balanced energy-efficient benchmark. In *Proc. ACM SIGMOD*, Beijing, China, June 2007.

[3] WattsUp. .NET Power Meter. http://wattsupmeters.com.