FAWNSort: Energy-efficient Sorting of 10GB

Vijay Vasudevan, Lawrence Tan, David Andersen
Carnegie Mellon University

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

1 Introduction

In this document, we describe our submission for the 2010 10GB JouleSort competition. Our system consists of a machine with a low-power server processor and five flash drives, sorting the 10GB dataset in 21.2 seconds ($\pm$0.227s) seconds with an average power of 104.9W ($\pm$0.8W). This system sorts the 10GB dataset using only 2228 Joules ($\pm$12 J), providing 44884 ($\pm$248) sorted records per Joule.

Our entry for the 10GB competition tried to use the most energy-efficient platform we could find that could hold the dataset in memory to enable a one-pass sort. 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 Xeon L3426 1.86GHz quad-core processor (with two hyperthreads per core, TurboBoost-enabled) paired with 12GB of DDR3-1066 DRAM (2 DIMMS were 4GB modules and the other 2 DIMMS were 2GB modules). The mainboard is a development board from 2009 based on an Intel 3420 chipset (to the best of our knowledge, this confers no specific power advantage compared to off-the-shelf versions of the board such as the SuperMicro X8SIL-F or Intel S3420GPV Server Board), and we used a Prolimatech "Megahalems" fanless heatsink for the processor.

For storage, we use 4 SATA-based Intel X25-E flash drives (three had a 32GB capacity and one had 64GB), and 1 PCIe-based Fusion-io ioDrive (80GB). We use a 300W standard ATX power supply (FSP300) with a built-in and enabled cooling fan.

The storage devices were configured as follows: one small partition of a 32GB X25-E contained the OS. The other three X25-Es, the leftover portions of the OS disk, and the Fusion-IO (partitioned into three 10GB partitions) were arranged in a single partition software RAID-0 configuration. Both the input and output file were located in a single directory within this partition. We use a Fusion-io in addition to 4 X25-Es because the SATA bus exists on the DMI bus with a bandwidth limitation of 10Gbps in theory and slightly less in practice. The Fusion-io is on the PCIe bus, which is independent of the DMI bus and has a much higher bandwidth to the processor and memory system. Using both types of devices together therefore allowed us to more easily saturate the I/O and CPU capabilities of our system.

2.1 System price and power

The Fusion-io 80GB ioDrive (~$3000) and the 4 Intel X25-E drives (~$400 for the 32GB drives and ~$800 for the 64GB drive) combine for approximately $5000 of the system cost. The power supply, processor, mainboard, power supply, and 12GB of DRAM cost approximately $1500 for a total system cost of approximately $6500. The system price is thus dominated by the flash storage components.

The total power consumption of the system peaks at about 116 W during the experiment, but as mentioned below, averages about 105W over the duration of the sort runs. While we do not have individual power numbers for each component during the experiment, the {processor, DRAM, motherboard, power supply} combination consumes about 31 W at idle, the Fusion-io adds 6W at idle, and each X25-E adds about 1W to the idle power consumption for a total of 43 W at idle with all components attached.

3 Software

All of our results are using Ubuntu Linux version 9.04 with kernel version 2.6.28 for driver compatibility with the Fusion-io device. We used ext4 with journaling disabled on the RAID-0 device. 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 sorting, we use a trial version of NSort software (`http://www.ordinal.com`) with the parameters shown in Figure 1.

```
nsort -o /raid0/output
      -processes=8 -memory=11600M
      -method=radix
      -field=name:key,size:10,off:0,character
      -key=key
      -field=name:val,size:89,off:10,character
      -statistics
      -file_system=/raid0,direct,
                   transfer_size:50M,count:2
      inputdata
```

Figure 1: NSort Parameters Used

Similar to previous entries that used NSort to compete for JouleSort [3, 4], we meet the 2010 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 ([6]) specified as accurate to within 0.1%. We connect the power meter to a separate machine using the onboard USB interface and use publicly available software for the power meter to log the power meter output once per second. For each run, we start the logging software on a separate machine before the NSort experiment and then run the nsort command on the sorting machine, letting the sort complete and the power draw return to idle before manually stopping the logging software. We use the timestamps available on the logging meter to correlate the power draw values with the sort experiment.

We use `/usr/bin/time` to measure the runtime of NSort. Finally, 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 use the power numbers corresponding to the highest values for which the sort benchmark ran for the full second. For example, for our 21.278s experiment, we use the highest 20 values to average the power, ignoring the first and last values of the 22 pertinent entries—the sort may have run during only parts of the first and last seconds over which the power meter was logging power, so we only include values for which we know all seconds have been accounted for. We use this calculated average power and multiply by the actual runtime of the experiment to calculate the total number of joules.

Our system is placed on a desk in an office/cubicle environment (not mounted in a chassis). We use a very large fanless heatsink for our processor, and the ambient cooling provided for the office environment was sufficient for sustained operation of all components at load.

5 Results

Our results are summarized in the table below:

	Time (s)	Power (W)	Energy (J)	SRecs/J
Run 1	21.278	105.4	2242.5	44593
Run 2	21.333	104.1	2219.8	45049
Run 3	21.286	104.9	2232.6	44791
Run 4	21.454	104.1	2233.7	44769
Run 5	20.854	106.0	2211.5	45218
Avg	21.241	104.9	**2228.0**	**44884**
Error	0.227	0.849	12.273	247.564

We log the statistics provided by NSort for comparison with [3]. The table below summarizes the information (Utilization measured out of a total of 800% and bandwidth measured in terms of MB/s for reading and writing the data).

	In CPU Util	Out CPU Util	Input BW (MB/s)	Output BW (MB/s)
Run 1	343	628	973.71	1062
Run 2	339	651	953.29	1074
Run 3	339	613	971.82	1056
Run 4	336	622	975.61	1050
Run 5	343	646	976.56	1081
Avg	340	632	970.198	1065
Error	3	16.078	9.626	12.759

Our system improves upon the January 2010 Daytona winner by nearly a factor of two, and also improves upon the January 2010 Indy winner by 26% [2]. The latter group's more recent entry closes this gap to 5% for the Indy designation and 12% for the Daytona designation.

6 Experiences

Those familiar with our prior work on energy-efficient cluster computing may find it surprising that our submission used a server-class system as opposed to a low-power component system like the Intel Atom. The reasons for this choice were dominated by the ability of our server system to hold the entire 10GB dataset in DRAM to enable a one-pass sort—in this case, the energy efficiency benefits of performing a one-pass sort outweighed the hardware-based energy efficiency of low-power platforms that must perform a two-pass sort. Our submission tried to use the most energy-efficient platform we could find that allowed for a one-pass sort, and this turned out to use the low-frequency Xeon platform described above. Below, we describe some details about what other systems we tried before settling on the entry system described above.

6.1 Alternative Platforms

We tried several alternative low-power configurations based on the Intel Atom as part of our research into the "low-power

components at massive-scale" FAWN architecture [1]. In particular, we began with the Zotac Ion board based on an Intel Dual-core Atom 330 (also used by Beckmann et. al) paired with 4 Intel X25-E drives. Without any special software tweaking, we were able to get approximately 35000 SRecs/J at an average power of about 33W. We also tried to use the NVidia GPU available on the Ion to do a portion of the sorting, but found that the I/O was the major bottleneck regardless.

We also experimented with a single core Atom board by Advantech paired with 1 X25-E, and a dual-core Atom Pineview development board with two X25-Es. These boards were both lower power than the Zotac Ion—the Pineview board moved from a three-chip to a two-chip solution, placing the graphics and memory controllers on-die, thus reducing chipset power slightly. We also tried attaching a Fusion-io board to a dual-core Atom system, but because the Fusion-io currently requires significant host processing and memory, the Atom could not saturate the capabilities of the drive and so was not currently a good fit.

6.2 Lessons learned

The fact that the Xeon system performed better than the Atom was not entirely surprising to us, given our recent experiences applying our FAWN approach to benchmarks other than sorting [5]. We have found that two factors play a big role in determining relative efficiency: non-linearities in dataset size and node configurations, and the lack of optimizations of general purpose software for "wimpy platforms". In the first case, the ability to store the 10GB dataset in memory on the Xeon platform but not on the Atom is identical to experiments run for matrix multiplication benchmarks, where the Xeon is more efficient for the matrix size range that fits in the 8MB cache of the Xeon but which exceeds the 1MB cache on the Atom. The second case is exemplified by our measurements of encryption benchmarks on both systems, and is corroborated by Beckmann et. al's concurrent submission to the Joulesort competition using the Zotac Ion platform, showing that rewriting software can be necessary to obtain the peak performance out of these low-power systems.

From the hardware perspective, the Atom platform appears capable of further improvements to energy efficiency of sorting, but is currently limited in practice due to several factors, none that appear to be fundamental but simply due to choices made by hardware vendors of low-power platforms:

- High idle/fixed cost power – the boards we've used have all idled at 15-20W even though their peak is only about 10-15W higher. Fixed costs affect both traditional processors and low-power CPUs alike, but the proportionally higher fixed-cost:peak-power ratio on the Atom diminishes some of the benefits of the low-power processor.
- Architecture/Bus limitations – Most of the Atom boards we had only had two SATA connectors on board. Supermicro recently released one that has 6 ports available, but the SATA bus is currently connected to the processor over the DMI bus. The DMI bus has a bandwith limitation of 10Gbps per direction—at approximately 250MB/s per X25-E for sequential reads, one requires only about 4 X25-Es before saturating the bus, so two additional ports are not useful to increase throughput, only for providing higher per-node capacity. Thus, we were unable to find a system that both provided lower fixed power costs (the Pineview) and one that provided a balanced amount of I/O to saturate the processor (the Zotac Ion).
- The Linux block layer is not optimized for SSDs and makes it difficult to saturate several modern flash drives with a dual-core Atom. For example, each block request triggers the kernel to add randomness to the kernel entropy pool, a cost that is normally low given the relatively slow speed of magnetic disks, but which is one cause of lower performance on I/O-bound but constrained systems. We are currently looking into many more software optimizations to see if we can make better use of the I/O capability available on newer Atom boards with more SATA ports. Moreover, since Linux version 2.6.28, there have been several modifications to the block layer that attempt to better support flash SSDs, which should improve performance for both low-power and traditional systems.

6.3 Outlook

If the low-power platforms could address the same amount of memory as the traditional servers to make a one-pass sort possible on both architectures, we expect that Atom-based platforms' improved hardware energy efficiency would best the Xeon platforms in sorting because of the better balance between processing and I/O. However, should the gap between per-node memory on the two platforms remain, the node with more memory will win out for workloads that critically depend on working set size. The power cost of DRAM may begin to dominate for those looking to enable a one-pass sort for larger datasets, and when comparing two-pass sorts on different architectures, a low-power platform, properly architected, should compete well for the small-to-medium dataset Joulesort competitions, up until the point that the network power or storage power starts to dominate.

On this last point, we will also note that it is possible to amortize the cost of a very fast network into the compute infrastructure itself. For example, a recent startup called SeaMicro (http://www.seamicro.com) has developed a 512-core Intel Atom machine that consumes a max of 2kW (not including the storage but including the 1.28Tbps networking backplane and I/O controllers)—an amortized cost of only 4W per "node" and consuming six times less power than today's Atom systems at peak. Assuming that

the I/O architecture can match the processing capability, we believe that such a system may be successful for the largest Joulesort competitions.

References

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, Oct. 2009.

[2] A. Beckmann, U. Meyer, P. Sanders, and J. Singler. Energy-efficient sorting using solid state disks. http://sortbenchmark.org/ecosort_2010_Jan_01.pdf, 2010.

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

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

[5] V. Vasudevan, D. G. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru. Energy-efficient cluster computing with FAWN: Workloads and implications. In *Proc. e-Energy 2010*, Passau, Germany, Apr. 2010. (invited paper).

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