

OzSort 2.0: Sorting up to 252GB for a Penny

Nikolas Askitis
Software Developer and Consultant
Department of Computer Science and Software Engineering,
The University of Melbourne.
naskitis@unimelb.edu.au
askitissn@gmail.com

March 30, 2010

Abstract

We present OzSort 2.0, a stable external merge sort software optimized for the requirements of PennySort Indy 2010. OzSort 2.0 is engineered to better exploit multi-core processors over its predecessor, and to further minimize cache misses. In this paper, we explain the workings of OzSort 2.0 and show how we sorted up to 252GB (2 516 582 400 records) for a Penny (< 1929s) using standard desktop PC components.

1 Introduction

OzSort 2.0 is based upon the classic external merge sort (1) which has two main stages: stage 1, *sorting* and stage 2, *merging*. Stage 1 simply breaks a dataset into homogeneous runs which are then sorted and written back out to disk. Depending on the number of records, the last run could be partially empty. Stage 2 then takes these sorted runs and merges them to create one completed sorted file.

OzSort 2.0 is catered for the Indy benchmark category. The Indy benchmark assumes 100 byte record sizes and that each record begins with a 10 byte record key, followed by 90 bytes of payload data. Indy also allows us to overwrite the input file with runs, which is beneficial with respect to performance. The total disk space required by Indy OzSort 2.0 is therefore $2NL$, where N is the number of records and L is the homogeneous length of the records. The Daytona benchmark category, on the other hand, does not allow the original input file to be overwritten and assumes that the user will specify the record size, the key size, and the key offset within the record (among other constraints). As such, the total space required by Daytona is $3NL$.

OzSort 2.0 is a 64-bit native program, meaning that it should only be used on a 64-bit computing architecture, though from experience, some older 32-bit processors (such as a Pentium IV) with 64-bit emulation are also compatible — though not recommended. In the following sections, we describe how we designed OzSort 2.0 and the choices we made with hardware, in order to maximize performance while minimizing overall system cost.

2 The sort phase (stage 1)

Our 2009 sorting phase involved the following steps:

1. Wait until we read a 2GB unsorted run from disk.
2. Break the run in half and spawn two threads to process each half in parallel.
 - (a) The two threads scan their 1GB portion, generating a set of 128-bit integer keys that represent the first 10 bytes of each record (the primary key), plus the record offset. The set of keys are then sorting using a customized iterative in-place quick sort routine.

3. Wait until both threads complete then merge the two 1GB portions together, writing the smallest key to an output buffer. Two homogeneous output buffers were employed, such that, when the first filled and was being written to disk, the second could continue accepting records from the merger.
4. While we are sorting, merging, and writing the run, a third thread works in the background partially prefetching the next run.

Generating 2GB runs in this manner is efficient with respect to sorting, since smaller runs can generally be sorted quicker than larger runs. However, this approach is not scalable. Thus, we require a sorting algorithm that can handle larger runs while remaining computationally efficient, and one that can exploit current multi-core architectures to further accelerate performance. Our research led to the implementation of a simple yet effective algorithm shown below, that is used by OzSort 2.0:

1. A run consists of 2^{25} records (about 3.35GB), which will reduce the number of runs generated, improving scalability.
2. Break the run into 2^5 homogeneous fragments called `microruns`. Each microrun will thereby have exactly 2^{20} records (and assuming the last run is fully occupied).
3. Read the next microrun from disk (or initially the first). On completion, immediately spawn a detached thread to process the microrun. Repeat this step until you have read in and spawned threads for 2^5 microruns. Note: threads should be detached to minimize memory consumption (join-able threads can increase the overall process size considerably when several are spawned and/or remain idle).
4. Each thread will work in the background generating a set of 128-bit integer keys for its assigned microrun, then sorting the set using a customized iterative in-place quick sort routine (an improved version from last year that is more computationally efficient).
5. We wait until all threads finish, then access the smallest 128-bit integer key from each microrun and store it into a sorted heap called a `merge heap`, which has a maximum capacity of 2^5 entries — each entry can store up to 192-bits (the 128-bit key following by its associated 64-bit microrun number).
6. Conduct a 2^5 -way merge using the `merge heap` as follows:
 - (a) Extract the smallest key (and its associated microrun number) from the heap.
 - (b) Copy the extracted key into the next vacant slot in an array called `ptr_set`. The `ptr_set` has a capacity equal to the output buffer.
 - (c) Store into the sorted heap the next smallest (128-bit) key from the associated microrun, or reduce the heap size by one if the microrun is exhausted.
 - (d) Repeat the above two steps until the `ptr_set` becomes full, in which case, we iterate through its keys and transfer the associated records to the output buffer. This step is implemented in a manner that reduces cache misses while prompting better use of out-of-order execution, compiler optimizations, and the instruction pipeline.
 - (e) Repeat the previous steps until the `merge heap` is exhausted.
7. Repeat from Step 3 until the input file is exhausted, completing stage 1.

An abstract representation of this algorithm is illustrated in Figure 1. There are several key features of this algorithm which have an advantage over last year's approach. Dividing a run into a set of 2^5 microruns makes good use of multi-core architectures, since it allows sorting to overlap disk I/O more effectively. A value of 2^5 microruns was found to offer a good compromise between sort time and merge time. Increasing the number of microruns improved sorting performance (up to a point), but it increased the time required to merge the microruns together.

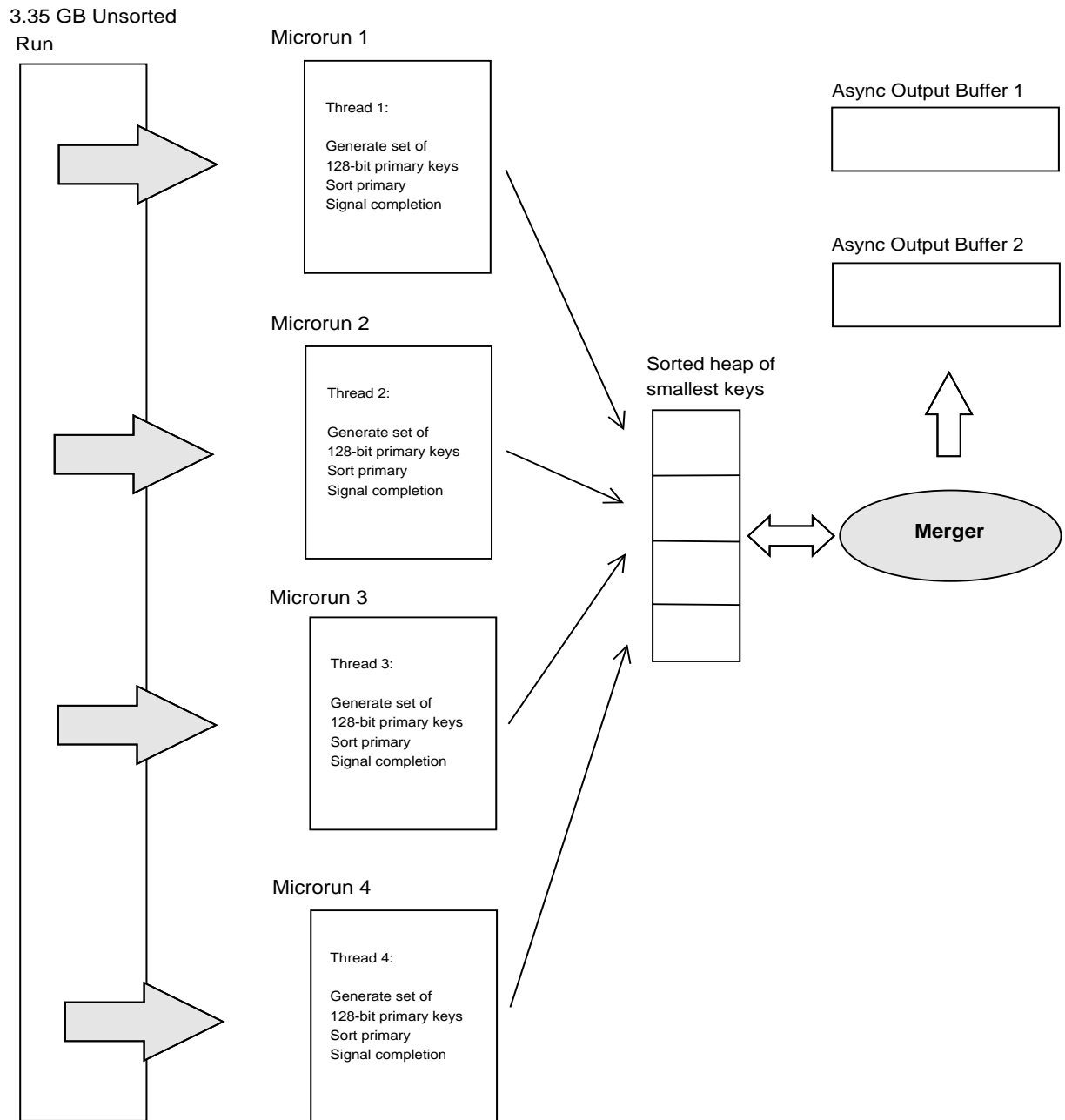


Figure 1: An abstract representation of the sort phase (stage 1), where a run is divided (as it is read from disk) into four homogeneous microruns (for example), which are processed independently and merged together to form a sorted run. This is a simple but practical algorithm that makes good use of current multi-core computing architectures.

Another key feature was HugeTLB, which we did not employ in our 2009 solution. With HugeTLB enabled, we can reduce the number of TLB misses incurred, thereby accelerating performance. Indeed, we observed a considerable improvement in the performance of OzSort 2.0 as a result. We include results with and without HugeTLB for comparison, later in this document.

The use of a `ptr_set`, as described in the algorithm above, was another important feature not considered in our 2009 submission. Transferring a record to an output buffer involves a random access to main memory, which will most likely incur a L2 cache miss, a TLB miss or both. By employing the `ptr_set` as described, we can hide some of the memory latency incurred by stimulating the hardware to transfer multiple records at once. In addition, we further promote parallelism by spawning a single join-able thread to transfer a (carefully selected) portion of the `ptr_set` in the background.

We also experimented with the output buffer size (the maximum number of records it can store before it is written out to disk) and observed that a size of 717440 records yielded near-optimal performance, probably due to better hardware/page alignment. We note, however, that we did not consider a dynamic output buffer which may insight further improvements in performance, since the inner, middle and outer regions of a conventional disk typically offer different (increasing) data bandwidths.

3 The merge phase (stage 2)

Stage 2 is presented with a series of 3.35GB sorted runs. Similar to stage 1, stage 2 breaks a run into a set of homogeneous *microruns*, containing 2^{19} records. It then proceeds with the following steps:

1. Read in the first microrun from the first run into memory.
2. Generate a 128-bit integer representation of the primary key of the first record in the microrun.
3. Store the 128-bit key into a sorted heap (the sorted heap has a capacity equal to the number of runs), along with its associated microrun number.
4. Read in the next microrun from the next run, and label it as the current microrun.
 - (a) Generate a 128-bit key representation of the primary key of first record in the current microrun.
 - (b) Store the 128-bit key into the sorted heap, along with its microrun number.
5. Repeat from step 4 until all microruns have been read into memory (overlapping I/O with computation).
6. Extract the smallest key from the sorted heap, along with its associated microrun number.
7. Fetch the required record from the microrun and transfer it to the output buffer (one of two output buffers, as implemented in stage 1; if the first buffer fills and is being written to disk, the second can continue accepting keys from the heap).
8. Fetch the next record from the associated microrun, generate its 128-bit integer key and store it into the sorted heap. If the microrun is exhausted, we fetch the next microrun from the associated run. Otherwise, the run is exhausted, in which case, we reduce the size of the heap by one.
9. Repeat step 6 until the sorted heap is exhausted, in which case, we flush out any remaining entries in the output buffers, completing stage 2.

The merge phase implemented for OzSort 2.0 is similar to approach used last year, except for some important changes. In our 2009 model, we employed a threaded prefetching mechanism to fetch the next set of microruns that will be accessed by the sorted heap into a prefetch buffer. The motivation was to try to hide some of the I/O seek costs incurred by fetching the next set of microruns from disk while the current are being processed. This approach worked, but the gains were small. First, reads and writes were interleaved on a single RAID drive, which hindered the effectiveness of prefetching — the operating system had to schedule multiple disk read and write commands, forcing the disk head to move more frequently.

Employing two RAID drives, one for reading microruns and the other for writing, would be a better but expensive option. Second, in order to allocate a prefetch buffer of sufficient size, we must reduce the microrun size. As a consequence, the number of seeks will increase which will hinder overall performance. As such, for OzSort 2.0, we eliminated the prefetch mechanism altogether, which allowed us to employ much larger microruns, thereby reducing the total number of seeks made.

Our 2009 model used the *strncmp()* routine to keep the heap sorted, which is simpler but computationally expensive. Hence, another key improvement in OzSort 2.0 is the use of fast in-place integer sorting (128-bit primary keys) to maintain the heap in sort order.

The employment of HugeTLB was another advantage over our previous design, as it can further accelerate performance due to a reduction of TLB misses. However, the performance benefits offered by HugeTLB during stage 2 were not as substantial as stage 1. Stage 1 conducts more localized memory accesses as it sorts microruns. This means that more TLB pages are likely to be reused, prompting better HugeTLB utilization. When we extract the smallest key from the heap in stage 2, however, we issue a random access to a large allocated portion of main memory to fetch the required record (once), which can greatly reduce access locality.

Note, however, that unlike in stage 1, in stage 2, the smallest key/record extracted from the sorted heap is immediately transferred to an output buffer. That is, we do not employ a *ptr_set* array, as we had described in stage 1. A *ptr_set* array did not yield notable performance gains when employed in stage 2, probably due to the increased cost of writing to disk as a result of reduced I/O access locality, which masked the benefits.

4 The system software

OzSort 2.0 was written in C and compiled using g++ version 4.4.3 with HugeTLB enabled, using the command: `g++ -ansi -pedantic -Wall -B /usr/local/share/libhugetlbfs/ -Wl,--hugetlbfs-align -O3 -fomit-frame-pointer -o ozsort ozsort.c -lrt`. We experimented with other compiler optimization options such as `-march`, but performance did not differ significantly from just using the `-O3` flag which is consistent to our results last year. We also tried the Intel `icc` compiler for Linux, but observed no notable improvement in performance (with respect to OzSort) against the standard g++ compiler.

We initially developed the software using Linux Kubuntu 9.10 which was easy to install and offered a high performance development environment. However, we encountered difficulties when we enabled HugeTLB, since we needed to reserve at least 1946 2MB pages using the command `hugeadm --pool-pages-min 2MB:1946`. With Kubuntu installed on a 4GB RAM machine, the most we could reserve was less than 1900 pages (after stopping most operating system services and running the O/S in console mode). Unfortunately, this was not sufficient to run OzSort 2.0 with HugeTLB enabled.

We were thus forced to install and configure Linux Gentoo which has a rather complicated, somewhat frustrating and time-consuming installation procedure. The plus side, however, is that Gentoo offers a lot more control over system services and configuration, and provides a fast and importantly, memory-efficient environment that was tuned for our system. Indeed, with Gentoo finally up and running, we were able to reserve up to 1950 2MB pages successfully.

4.1 Linux Kernel

In order to reserve at least 1946 2MB HugeTLB pages, we needed to configure and compile a custom Linux Kernel that was as small as possible while not compromising performance (using Gentoo source, Linux Kernel 2.6.31-gentoo-r6 SMP x86_64). The smaller the Kernel, the more memory becomes available for allocation by the user. After several trials, we eventually found a Linux Kernel configuration that produced a compressed `bzImage` file (the Kernel image file) of just 1.3MB (a typical size can be almost 5MB compressed). The `.config` file can be provided upon request, allowing you to reconstruct our Linux Gentoo Kernel (one tuned for the 2010 AMD AsRock MicroATX motherboard or equivalent).

4.2 File system

The XFS file system is well known to offer good I/O performance for large files (in some cases, XFS can approach the raw bandwidth offered by the hard drives). Although there are several other file systems available, XFS is generally the best option for this application, when used on a compiled Gentoo system. On Kubuntu 9.10, however, we found the JFS file system to be superior. On Gentoo (perhaps due to the fact that the system was compiled/tuned for our hardware), the performance of JFS was found to be slightly slower than XFS. We thus strongly recommend that you format a RAID drive using XFS (default settings are fine), if you intend to reproduce our results. On a standard Kubuntu system, on the other hand, we recommend the JFS filesystem (default settings).

4.3 RAID chunk size

We decided to set the raid chunk size to 256KB (set using the `mdadm` version 3.0.1), which is also the maximum block size supported by the current default XFS file system. We also tested 64KB and 128KB but did not notice any significant change in overall performance. We therefore recommend a chunk size of 256KB which was observed to work well with OzSort 2.0.

5 The hardware

Fortunately, we were able to re-use some of our hardware components from our 2009 entry. Our AsRock A780GM motherboard and our Seagate 7200.11 160GB SATA-II drives were still listed on newegg.com at the time of writing. The motherboard was available for the same price (excluding discounts) as last year and our hard drives were \$2 cheaper — a welcomed reduction in price considering that the RAID forms the most expensive component of our system.

5.1 Hard drives

The 7200.12 Seagate 160GB drives were also available from newegg.com this year and were only \$1 more expensive than the 7200.11 drives. We purchased one 7200.12 drive and tested it against one of our 7200.11 drives. Physically, the drives looked identical (apart from the different version/serial numbers on the drive label). We hypothesized that we would observe a marginal improvement in performance over our 7200.11 drive, though after some initial testing, we observed that this was not the case. Using the command: `dd if=/dev/md0 of=/dev/null iflag=direct bs=1024000000 count=10`, the 7200.12 was actually a little slower, offering a peak (outer-rim) throughput of about 114MB/s compared to the 126MB/s offered by our 7200.11 drive. We are thus confident that our 7200.11 drives still offer competitive performance at a budget price.

5.2 Memory

Last year, we purchased 4GB of high performance 5-5-5-15 GeIL Black Dragon DDR2 800 RAM valued at \$37.99. This year, we purchased another set of 4GB (2x2GB) of GeIL 5-5-5-15 Black Dragon series memory, since they are high performance units, but more importantly, they were the cheapest 4GB CAS-5 RAM units available at the time of writing. Nonetheless, their cost was \$72.99, \$35 more than last year. The cost of memory in general has risen considerably since early last year. One advantage of this years GeIL memory, however, is that it operates at 1066Mhz, giving us a small but welcomed performance boost. We initially developed the software by reusing our 4GB G. Skill (NT model) DDR2 800Mhz memory, which was available on newegg.com for \$76.99. However, we later replaced the unit with the cheaper and faster GeIL series. We did not consider low-latency DDR2 RAM modules this year (4-4-4-12 timings), since we know though preliminary trials that their performance gains generally do not compensate for their cost. However, if budget is not of constraint, we recommend employing 4-4-4-12 DDR2 800/1066 RAM modules, since we observed a notable improvement in the performance, particularly during stage 1.

5.3 Computer case

We were able to purchase a MicroATX computer case with a 400W power pack (that offered — among the standard power ports — two sata-power ports) from newegg.com at the start of the year, for only \$19.99. Given our knowledge with JouleSort, we know that a 400W power pack will provide ample power for our needs. The computer case, however, did present us with an interesting assembly challenge — since the case is somewhat smaller than a typical ATX-style computer case and we had 6 Seagate drives to squeeze in. Nonetheless, this setup proved adequate for PennySort.

5.4 Processor

The AMD Athlon 64 X2 2.7Ghz Kuma processor that we used last year is now obsolete (so is the AMD Athlon 64 LE-1640 used in JouleSort). As such, we purchased a new CPU which fortunately was available at a cheap price, though still not cheap enough to compensate for the high price of memory.

The AMD Athlon II 2.8Ghz 240 processor was available for \$56.99, \$3 dollars cheaper than our 2009 Kuma processor. In addition, the AMD 240 processor offered slightly faster clock speeds, a larger L2 cache per core and a faster FSB speed. As a plus, the AMD Athlon II 240 operates at a 65W, which is considerably cooler than Kuma.

OzSort 2.0 was initially developed using a AMD Phenom II X2 545 3.0Ghz processor with 7MB of shared cache. The 545 is obviously a more powerful processor than the 240, but its performance could not compensate for its high cost. Stage 1 was notably faster, but its performance during stage 2 was no better than the 240. These results are consistent with last year, where we observed that the single core AMD Athlon 64 LE-1640 rivaled the more powerful Kuma processor during stage 2.

5.5 Motherboard

We also purchased a new AsRock MicroATX motherboard listed on newegg.com (at the time of writing) — the AsRock A785GM-LE/128MB. This motherboard is the improved version of the A780GM that we used last year, offering a more powerful BIOS, upgraded chipsets, and native support for the AMD Athlon II X2 and AMD Phenom II X3/X4 processors (the A780GM does not provide native support for these processors; a BIOS update was required which is both risky and generally not as optimal as hardware specifically designed to support these processors). Physically, the motherboards looked very similar, with only minor changes to the south-bridge heatsink, the on-board labeling, and chip layout, etc.

However, unlike the A780GM, the A785GM-LE/128MB offered 128MB of side-port memory which is beneficial, since it eliminates the need for reserving 32MB (min.) of main memory for the on-board graphics card. The extra 32MB of memory allowed us to increase the output buffer size to the near-optimal value (on our machine) of 717 440 records, and also to allocate 1946+ 2MB HugeTLB pages. This, in turn, compensated for its more expensive price tag of \$64.99 (\$5 more than the A780GM).

We initially developed the software on a Gigabyte MA790X-UD4P motherboard which employs the AMD750 south-bridge chipset and 8 SATA-II ports. This motherboard, however, does not have an onboard graphics card so a temporary NVIDIA Geforce card (PCI-E) was used. It is a full-ATX motherboard and thus offers more expansion capabilities than the AsRock motherboards that we have used. It also worked well under Linux, though we did encounter initial problems when we booted the machine as it kept freezing during the BIOS/system initialization step of “Verifying DMI pool data ...”. We resolved this issue by updating the BIOS and ensuring that the bootable drive had its boot partition flagged as bootable, and that its MBR was set by the *lilo* boot loader. We did not encounter any of these issues with the AsRock motherboards, which worked flawlessly.

In addition, we found the AsRock motherboards offered a more user-friendly BIOS than the Gigabyte, and also required fewer device drivers to be installed into the Linux Kernel, allowing us to further shrink the Kernel size without compromising performance. In all, although the Gigabyte board is a quality board, the AsRock A785GM-LE was a better choice for our purposes, and was also observed (though preliminary trials) to be notably faster during stage 2.

We also considered the BioStar A760G motherboard, shown in our Appendix, since it offered 6 SATA-II ports and was available for only \$55.99 — \$9 dollars cheaper than our AsRock which increases the

| Device | Qty | Price (USD) | Qty × price | Total cost (USD) |
|--|-----|-------------|-------------|------------------|
| AsRock MicroATX A785GM-LE/128M | 1 | 64.99 | 64.99 | |
| AMD Athlon II X2 240 2.8Ghz 65W | 1 | 56.99 | 56.99 | |
| GeIL Black Dragon 4GB DDR2 1066Mhz CAS-5 | 1 | 72.99 | 72.99 | |
| Seagate Barracuda 7200.11 160GB SATA-II | 6 | 37.99 | 227.94 | |
| 400W MircoATX Case | 1 | 19.99 | 19.99 | |
| SATA-II data cable | 5 | 1.79 | 8.95 | |
| Power splitter 1-to-2 | 2 | 1.79 | 3.58 | |
| Assembly fee | 1 | 35 | 35 | |
| | | | | \$490.43 |

Table 1: *The PC components used to assemble our 2010 PennySort machine; prices are based on newegg.com between January 2010 and March 2010 and do not include any discounts. Screen shots are provided in the Appendix for your viewing.*

time budget by about 36 seconds. However, it does not offer native support for the AMD Athlon II series processors (a bios upgrade is required). Furthermore, it is not a particularly well-known motherboard compared to other popular brands such as Asus and AsRock, which will likely be an issue with Linux with respect to system stability and performance. Moreover, a key issue with this motherboard is that it offers no sideport memory, and so, it must reserve a minimum of 32MB of memory for its on-board graphics card. We know from our AsRock A780GM motherboard (which is a better built unit), that this reduction in total memory capacity will impose a upper-limit of about 1932 2MB HugeTLB pages. This is sufficient for stage 1, which can operate efficiently with only 1932 HugeTLB pages, but stage 2 requires slightly more memory — about 1946 HugeTLB pages. As a result, we would be forced to reduce the output buffer size, the microrun size, or both, in order to allow stage 2 to operate with only 1932 HugeTLB pages. On our A780GM motherboard, this action resulted in a notable decline in performance, mostly offsetting the 36 second budget gain. Hence, in conclusion, our AsRock A780GM/128MB motherboard was found to be a better and faster option, despite being a more expensive one. This is an interesting example of where sometimes, it is not always the cheapest (modern) hardware options that will yield the best budget vs. speed balance.

6 PennySort Results

The choice of hardware is a key aspect of this benchmark. The components we used for our final assembled PC are shown in Table 1. The time budget calculated for PennySort is shown in Table 2, and the results in Table 3. The time shown represents the total wall time required to execute the software, captured using the standard Linux *time* command. The time shown was averaged over 10 runs, the standard deviation of which was low. After each run, the RAID drive was unmounted, reformatted and remounted; main memory was also flushed with random data, to help ensure a consistent initial state between runs. After extensive testing on different hardware, we are confident that you should be able to reproduce our results on a machine with similar specifications and software. The output of *Valsort* is shown below:

```
root@ozsort:~$ valsort dataset.sorted;
Sort complete, now validating ...
Records: 2516582400
Checksum: 4b0029a671cb9034
Duplicate keys: 0
SUCCESS - all records are in order
```


| | | | |
|-----------------------|---|--------|---------|
| Total cost (USD) | | 490.43 | dollars |
| Total cost in pennies | | 49043 | cents |
| Penny sort life time | $(60 \times 60 \times 24 \times 365 \times 3) = 94608000$ | | seconds |
| Penny sort budget | $94608000 \div 49043 = \mathbf{1929.08}$ | | seconds |

Table 2: *The 2010 OzSort 2.0 PennySort budget calculated using the component costs shown in Table 1.*

| HugeTLB state | Stage 1 <i>sec</i> | Stage 2 <i>sec</i> | Total time <i>sec</i> | Budget <i>sec</i> |
|----------------------|------------------------------|------------------------------|---------------------------------|-----------------------------|
| on | 934.85 | 992.13 | 1926.98 | 1929.08 |
| off | 1003.97 | 996.91 | 2000.88 | 1929.08 |

Table 3: *The total wall time (in seconds) required by OzSort 2.0 for Indy PennySort 2010, with and without HugeTLB. The results shown are the average taken from 10 runs; the standard deviation was low. After each run, the 6-disk RAID drive was unmounted, reformatted, then remounted.*

7 Conclusion

OzSort2.0 is a re-engineered version of last years OzSort software — a fast and stable external sorting application that is designed for the requirements of the PennySort (Indy) benchmark. OzSort 2.0 is a more scalable and efficient sorting solution, sorting up to 252GB of data for a Penny using standard PC components. Although this is a gain of only around 6GB from last year (as a result of high memory prices), it was accomplished in markedly less time (1927s compared to 2150s). OzSort 2.0 can also offer competitive performance on a system without HugeTLB (which is common), sorting the same amount of data in about 2000s — only 73s longer. Hence, given a conservatively larger time budget, OzSort 2.0 can readily scale to much larger datasets – sorting over 300GB for a Penny is within range.

References

- [1] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. third edition, 1997.

Appendix B: BIOS configurations

7.1 BIOS configurations

We spent some time to explore the wealth of options provided by the AsRock BIOS. We first loaded the default BIOS settings, then updated the following:

OC Tweaker menu:

```
CPU Configuration:
  Overclock Mode: Auto
  CPU Active Core Control: All Cores
  HT Bus Speed: x10 2000Mhz
  HT Bus Width: 16 Bit
  Memory Clock: 533Mhz (DDR2 1066)
  Memory Timing:
    Power Down Enable: Disabled
    CAS Latency (CL): 5CLK
    TRCD: 5CLK
    TRP: 5CLK
    TRAS: 15CLK
  SidePort Clock Speed: Auto
```

Advanced menu:

```
CPU Configuration:
  Cool 'n' Quiet: Disabled
  Secure Virtual Machine: Disabled
  L3 Cache Allocation: All Cores

Chipset Configuration:
  OnBoard HD Audio: Disabled
  OnBoard Lan: Enabled
  Primary Graphics Adapter: Onboard
  Internal Graphics Mode: SIDEPORT

Storage Configuration:
  Onboard SATA Controller: Enabled
  SATA Operation Mode: AHCI

Floppy Configuration:
  Floppy A: Disabled

SuperIO Configuration:
  OnBoard Floppy Controller: Disabled
```

H/W Monitor menu:

```
CPU Quiet Fan: Disabled
```

Appendix C: Processor specifications

```
root@ozsort:~$ cat /proc/cpuinfo
processor      : 0
vendor_id    : AuthenticAMD
cpu family   : 16
model        : 6
model name   : AMD Athlon(tm) II X2 240 Processor
stepping     : 2
cpu MHz      : 2800.246
cache size   : 1024 KB
physical id  : 0
siblings     : 2
core id      : 0
cpu cores    : 2
apicid       : 0
initial apicid : 0
fpu          : yes
fpu_exception : yes
cpuid level  : 5
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8
apic sep mtrr pge mca cmov pat pse36 clflush mmx
fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb
rdtscp lm 3dnowext 3dnow constant_tsc rep_good
nonstop_tsc extd_apicid pni monitor cx16 popcnt
lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt
bogomips     : 5600.48
TLB size     : 1024 4K pages
clflush size  : 64
cache_alignment : 64
address sizes : 48 bits physical, 48 bits virtual
power management: ts ttp tm stc 100mhzsteps hwpstate
```

```
processor      : 1
vendor_id    : AuthenticAMD
cpu family   : 16
model        : 6
model name   : AMD Athlon(tm) II X2 240 Processor
stepping     : 2
cpu MHz      : 2800.246
cache size   : 1024 KB
physical id  : 0
siblings     : 2
core id      : 1
cpu cores    : 2
apicid       : 1
initial apicid : 1
fpu          : yes
fpu_exception : yes
cpuid level  : 5
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8
apic sep mtrr pge mca cmov pat pse36 clflush mmx
fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb
rdtscp lm 3dnowext 3dnow constant_tsc rep_good
nonstop_tsc extd_apicid pni monitor cx16 popcnt
```

```
lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt
bogomips      : 5600.16
TLB size      : 1024 4K pages
clflush size  : 64
cache_alignment : 64
address sizes  : 48 bits physical, 48 bits virtual
power management: ts ttp tm stc 100mhzsteps hwpstate
```

Appendix D: Available memory

The total available memory, as reported by our Gentoo Linux operating system.

```
root@ozsort:~$ cat /proc/meminfo
MemTotal:      4060440 kB
MemFree:       33992 kB
Buffers:       0 kB
Cached:        15728 kB
SwapCached:    0 kB
Active:        4668 kB
Inactive:      13356 kB
Active(anon):  1484 kB
Inactive(anon): 996 kB
Active(file):  3184 kB
Inactive(file): 12360 kB
Unevictable:   0 kB
Mlocked:      0 kB
SwapTotal:    0 kB
SwapFree:     0 kB
Dirty:        4 kB
Writeback:    0 kB
AnonPages:    2340 kB
Mapped:       1104 kB
Slab:         4792 kB
SReclaimable: 1256 kB
SUnreclaim:   3536 kB
PageTables:   512 kB
NFS_Unstable: 0 kB
Bounce:      0 kB
WritebackTmp: 0 kB
CommitLimit: 33420 kB
Committed_AS: 5216 kB
VmallocTotal: 34359738367 kB
VmallocUsed:  6244 kB
VmallocChunk: 34359731987 kB
HugePages_Total: 1950
HugePages_Free:  1950
HugePages_Rsvd:  0
HugePages_Surp:  0
Hugepagesize:  2048 kB
DirectMap4k:   4736 kB
DirectMap2M:  2091008 kB
DirectMap1G:  2097152 kB
```

Appendix E: Setting up the software RAID and HugeTLB

To aid in reproducing our results, we have provided a screen-shot of our partitions below using the Linux command “fdisk -l”. Your partition sizes can vary and the operating system can be kept on a separate disk if you prefer. In fact, we found that keeping the O/S on a separate disk proved to be a simple and convenient option during the initial development and testing phase, as it allowed for greater simplicity and flexibility in raid construction and testing. It also made it easier to move the raid drives between different machines, and it reduced the threat of losing the operating system and data due to partitioning/formatting errors, or damage caused by transit).

General rule-of-thumb: make sure the partitions used for the raid are of the same size and start at the same cylinder. Although software raid offers a lot of flexibility with regards to partition sizes and positions, maintaining uniform partition sizes/locations and the same brand/size disks can help improve overall performance. Also, you should always start your Linux raid partition from the first cylinder in each disk (which is usually the outer-rim, as was in our case).

```
Disk /dev/sda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000265b7
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sda1 | | 1 | 18630 | 149645443+ | fd | Linux raid autodetect |
| /dev/sda2 | * | 18631 | 18646 | 128520 | 83 | Linux |
| /dev/sda3 | | 18647 | 18778 | 1060290 | 82 | Linux swap / Solaris |
| /dev/sda4 | | 18779 | 19457 | 5454067+ | 83 | Linux |

```
Disk /dev/sdb: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0xace7b1b4
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sdb1 | | 1 | 18630 | 149645443+ | fd | Linux raid autodetect |

```
Disk /dev/sdc: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x5492c744
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sdc1 | | 1 | 18630 | 149645443+ | 83 | Linux raid autodetect |

```
Disk /dev/sdd: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x3cb500c0
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sdd1 | | 1 | 18630 | 149645443+ | fd | Linux raid autodetect |

```
Disk /dev/sde: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000234d8
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sde1 | | 1 | 18630 | 149645443+ | fd | Linux raid autodetect |

```
Disk /dev/sdf: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x707b6276
```

| Device | Boot | Start | End | Blocks | Id | System |
|-----------|------|-------|-------|------------|----|-----------------------|
| /dev/sdf1 | | 1 | 18630 | 149645443+ | 83 | Linux raid autodetect |

To generate the software raid, enter the following command:

```
./mdadm -create /dev/md0 -level=0 -chunk=256 -raid-devices=6
/dev/sda1 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1 /dev/sdf1;
```

Then type:

```
./mkfs.xfs -f /dev/md0
```

```
mkdir /mnt/raid;
```

Once done, open /etc/fstab and add the following line:

```
/dev/md0 /mnt/raid xfs users,async,exec,rw,dev,noatime,nodiratime,noauto 0 0
```

You should now be able to mount your raid:

```
mount /mnt/raid;
```

Our 6-disk RAID-0 setup offered a peak bandwidth (outer-rim) of 695MB/s, as reported by the following command:

```
dd if=/dev/md0 of=/dev/null iflag=direct bs=1024000000 count=5;
```

To setup HugeTLB, type the following (assuming your system is HugeTLB ready):

```
export HUGETLB_MORECORE=yes
mkdir -p /mnt/hugetlbfs
mount -t hugetlbfs none /mnt/hugetlbfs
hugeadm --add-temp-swap --pool-pages-min 2MB:1950
```

Appendix F: Custom Kernel size

A considerable amount of effort went into shrinking the Kernel (without compromising performance) to allow more memory to be allocatable to OzSort 2.0. The configuration file used is available upon request; *asrock-tiny* was the Kernel image used for the experiments. The others shown were some original test cases (note their size). *bzImageOriginal* for example, was the original image Kernel compiled from the default Kernel configuration file.

```
root@ozsort:~$ ls -al /boot
total 28849
drwxr-xr-x  3 root root   1024 Feb 22 21:53 .
drwxr-xr-x 18 root root   4096 Feb 16 18:24 ..
-rw-r--r--  1 root root      0 Jan 21 12:33 .keep
-rw-r--r--  1 root root      0 Feb 13  2020 .keep_sys-boot_lilo-0
-rw-----  1 root root 174592 Feb 19 08:58 .map
-rw-r--r--  1 root root 1867776 Feb 20 09:10 asrock
-rw-r--r--  1 root root 1566336 Feb 20 11:22 asrock-exp-sp
```

```
-rw-r--r-- 1 root root 1388928 Feb 20 11:06 asrock-exp-sz
-rw-r--r-- 1 root root 1632992 Feb 20 10:35 asrock-opt
-rw-r--r-- 1 root root 1447904 Feb 20 10:29 asrock-smaller
-rw-r--r-- 1 root root 1367488 Feb 22 21:49 asrock-tiny
lrwxrwxrwx 1 root root      1 Feb 13  2020 boot -> .
-rw-r--r-- 1 root root    512 Feb 16 23:02 boot.0800
-rw-r--r-- 1 root root    512 Feb 13  2020 boot.0840
-rw-r--r-- 1 root root    512 Feb 19 08:56 boot.0860
-rw-r--r-- 1 root root 2653632 Feb 13  2020 bzImageCustom
-rw-r--r-- 1 root root 4920032 Feb 13  2020 bzImageOriginal
drwx----- 2 root root   12288 Feb 13 08:16 lost+found
-rw----- 1 root root   109568 Feb 22 21:53 map
```

Appendix G: Screenshots

We have provided three screenshots of the components used for this competition, all of which were listed at the time of writing on www.newegg.com.

newegg.com Shopping Cart - Windows Internet Explorer

http://secure.newegg.com/ShoppingCart.aspx?SubSite=View

File Edit View Favorites Tools Help

Zoom In Zoom Out Refresh Stop

Windows Live Europe

What's New Profile Mail Photos Calendar MSN Share

Images Search

Weather News Maps

Download video amazon.com Translate webpage europe

Free Home

7 Item(s) (\$295.53) MY ACCOUNT

[login]

newegg.com

SEARCH

COMPUTER HARDWARE PCS & LAPTOPS NETWORKING ELECTRONICS HOME THEATER CAMERAS & CAMCORDERS SOFTWARE GAMING CELL PHONES HOME & OFFICE MORE

MY NEWEGG REVIEWS HELP & INFO

GO

Home > My Shopping Cart

MY SHOPPING CART

Update Qty's Remove Selected Move Selected To...

| | Qty. | Product Description | Savings | Total Price |
|---|------|---|-----------------|--------------------|
| <input type="checkbox"/> | 1 | OKGEAR 6" SATA II Cable Model GCCATASM - Retail Item #: N82E16812123276 Return Policy: Standard Return Policy | | \$179 |
| <input type="checkbox"/> | 1 | SYBA 5.1" Cable Model SY_CAB40007 - OEM Item #: N82E16812186068 Return Policy: Standard Return Policy | | \$179 |
| <input type="checkbox"/> | 1 | Seagate Barracuda 7200.11 ST3160813AS 160GB 7200 RPM SATA 3.0Gb/s 3.5" Internal Hard Drive - Bare Drive Item #: N82E16922148397 Return Policy: Standard Return Policy | | \$37.99 |
| <input type="checkbox"/> | 1 | ASRock A780LM AM2+AM2 (7600) Micro ATX AMD Motherboard - Retail Item #: N82E16813157153 Return Policy: Standard Return Policy | -\$5.00 Instant | \$59.99 \$54.99 |
| Protect Your Investment (expand for options) | | | | |
| <input type="checkbox"/> | 1 | ASRock A785GM-LE/128M AM3/AM2+AM2 AMD 785G Micro ATX AMD Motherboard - Retail Item #: N82E16813157179 Return Policy: Standard Return Policy | | \$64.99 |
| <input type="checkbox"/> | 1 | G.SKILL 4GB (2 x 2GB) 240-Pin DDR3 SDRAM DDR3 800 (PC2 6400) Dual Channel Kit Desktop Memory Model F2-6400CL5D-4GBN1 - Retail Item #: N82E16820217207 Return Policy: Standard Return Policy | | \$76.99 |
| <input type="checkbox"/> | 1 | AMD Athlon II X2 240 Processor 2.8GHz Socket AM3 65W Dual-Core Processor Model ADY240CCG0BOX - Retail Item #: N82E16619103688 Return Policy: CPU Replacement Only Return Policy | | \$56.99 |
| Calculate Shipping | | | | Subtotal: \$295.53 |
| Zip Code: <input type="text"/> | | | | Shipping: \$0.00 |
| UPS Guaranteed 3 Day Service <input type="button" value="GO"/> | | | | |
| <p>NEWEGG PROMISE Satisfaction is our total shopping satisfaction. Virtually all of Newegg.com's offerings are protected by a 30-day refund policy.</p> <p>Customer Protections</p> <p>Privacy We only share your personal information working on our behalf to complete your order, such as UPS and FedEx.</p> <p>Privacy Particulars</p> <p>Security We foil data hijackers. Newegg protects the security of your shopping information using Secure Sockets Layer (SSL) software.</p> <p>Security Specifics</p> | | | | |

My Wish Lists | Print Cart | Email Cart

Internet

Sunday, February 21, 2010 9:40 AM

start Newegg.com Shopping...

Figure 2: Appendix G1: Component prices of PC parts @ Newegg.com on 21st Feb 2010.

08:08 AM Friday 15 January 2010

Newegg.com Shopping Cart - Mozilla Firefox

http://secure.newegg.com/shopping/shoppingCart.aspx?Submit=view

1 Item(s) (\$19.99) [login]

MY NEWEGG REVIEWS HELP & INFO

COMPUTER HARDWARE PCS & LAPTOPS NETWORKING ELECTRONICS HOME THEATER CAMERAS & CAMCORDERS SOFTWARE GAMING CELL PHONES HOME & OFFICE MORE

Home > My Shopping Cart

MY SHOPPING CART

Update Qty Remove Selected Move Selected To...

| Qty. | Product Description | Savings | Total Price |
|------|--|--------------|-------------|
| 1 | Linkworld A617-01FEU-P08 Beige / Silver MicroATX Mini Tower Computer Case 400W Power Supply - Retail Item #: 1082E16811164338 Return Policy: Standard Return Policy | | \$19.99 |
| | | Subtotal: | \$19.99 |
| | | Shipping: | \$0.00 |
| | | Promo Code: | \$0.00 |
| | | Grand Total* | \$19.99 |

My Wish Lists | Print Cart | Email Cart

NEWEGG PROMISE
Satisfaction We guarantee total shopping satisfaction. Virtually all of Newegg.com's offerings are protected by a 30-day refund policy.

Customer Protections
Privacy We only share your personal information with third parties working on our behalf to complete your order, such as UPS and FedEx.

Privacy Particulars
Security We foil data hijackers. Newegg protects the security of your information during transmission by using Secure Sockets Layer (SSL) software.

Security Specifics

Calculate Shipping
Zip Code: UPS Guaranteed 3 Day Service GO

Redeem Newegg Gift Cards
Card Number: Security Code: Apply

Apply Promo Codes: Enter up to 5 promo codes. Please note that stacking is not permitted.
Promo Code: (For Example: code1.code2.code3.code4.code5) Apply

* Above total does not include shipping or taxes. Please input zip code to calculate your grand total.
Having problems with your cart? Check FAQ for help or try emptying your cart to start over.
view important shipping information.

Policy & Agreement | Privacy Policy | Newegg Canada © 2000-2010 Newegg Inc. All rights reserved.

Downloads | Newegg.com Shopping... | Downloads - File Mana... | jetway - Tech.On The ... | KCalc | [Newegg.com - Comp...]

javascrip.tatt | Downloads : nano

Figure 3: Appendix G2: Component price of PC part @ Newegg on 15th Jan 2010.

Windows Internet Explorer - Newegg.com Shopping Cart - Windows Internet Explorer
 http://secure.newegg.com/Shopping/ShoppingCart.aspx?Submit=view

2 Item(s) (\$128.99) MY ACCOUNT [login]

COMPUTER HARDWARE | POS & LAPTOPS | NETWORKING | ELECTRONICS | HOME THEATER | CAMERAS & CAMCORDERS | SOFTWARE | GAMING | CELL PHONES | HOME & OFFICE | MORE



SEARCH

MY NEWEGG | REVIEWS | HELP & INFO

Home > My Shopping Cart

MY SHOPPING CART

Update Q'tys | Remove Selected | Move Selected To...

| Qty. | Product Description | Savings | Total Price |
|------|---|-----------|-------------|
| 1 |  BIOSTAR A760C M2+ AM2+/AM2 AMD 760G Micro ATX AMD Motherboard - Retail Item #: N82E16813138138 Return Policy: Standard Return Policy | | \$55.99 |
| 1 |  GeIL Black Dragon 4GB (2 x 2GB) 240-Pin DDR2 SDRAM DDR2 1066 (PC2 8500) Dual Channel Kit Desktop Memory Model GB24GB8500C5DC - Retail Item #: N82E16820144242 Return Policy: Memory Standard Return Policy | | \$72.99 |
| | | Subtotal: | \$128.98 |
| | | Shipping: | \$0.00 |

Calculate Shipping
 Zip Code: UPS Guaranteed 3 Day Service

Redeem Newegg Gift Cards
 Card Number: Security Code:

Apply Promo Codes: Enter up to 5 promo codes. Please note that stacking is not permitted.
 Promo Code:
 (For Example: code1,code2,code3,code4,code5)

Grand Total: * \$128.99

* Above total does not include shipping or taxes. Please input zip code to calculate your grand total.
Having problems with your cart? Check FAQ for help or try emptying your cart to start over.
 ▶ view important shipping information.

NEWEGG PROMISE

Satisfaction
 We guarantee total shopping satisfaction. Virtually all of Newegg.com's offerings are protected by a 30-day refund policy.

Customer Protections

Privacy
 We only share your personal information with third parties working on our behalf to complete your order, such as UPS and FedEx.

Privacy Particulars

Security
 We foil data hijackers. Newegg protects the security of your information during transmission by using Secure Sockets Layer (SSL) software.

Security Specifics

Internet | Protected Mode On | Saturday, 20 March 2010 6:45 AM 20/03/2010

Figure 4: Appendix G3: Component prices of PC parts @ Newegg on 20th Mar 2010.