Nsort: a Parallel Sorting Program for NUMA and SMP Machines

Version 3.0

August 21, 2000

Chris Nyberg, Ordinal Technology Corp

Charles Koester, Ordinal Technology Corp

Jim Gray, Microsoft Corporation

Introduction

Ordinal™ Nsort™ is a high-performance sort program for SGI IRIX, Sun Solaris and HP-UX servers. Nsort allows its users to realize the full processing potential of their multi-processor, multi-disk Unix systems for sorting data. Unlike the slow, single-threaded Unix sort utility or other third-party sort programs that achieve only minor processing parallelism, only Nsort can use tens of processors and hundreds of disks to quickly sort and merge data.

Nsort utilizes multi-disk file systems to read and write files at more than 1 gigabyte per second. Since Nsort uses 64-bit addressing, it can perform a one-pass sort for data sets that fit in main memory. Two-pass sorts using temporary files can also be performed. Using a two-pass sort, Nsort performed a terabyte sort in 2.5 hours using 2.3 gigabytes of main memory.

Nsort has the options one finds in a full-function commercial sort package. It handles the standard data types found in scientific and commercial applications. Nsort can summarize fields for rollup reporting, select and edit records, and optionally eliminate duplicate records. File merge and file copy operations are also supported.

This paper describes Nsort's background, presents its performance sorting a terabyte of data, and compares its performance on an industry-standard benchmark. Nsort performance is presented for file copying and record selection, and sorting with varying numbers of processors, input sizes, key types, key lengths, numbers of keys and record lengths.

Background

Sorting was one of the first commercial applications for computers and is a classic problem in computer science [1]. Sorting is commonly used to bring together records with equal keys, or to facilitate later searching of data. Modern-day sort programs can include or omit records based on key value, reorder record fields, and aggregate field values by key value. For example, many utility and telephone companies use sorting to bring together service usage records by account number, subtotalling the service charges in the process.

Sorting is used in data mining applications to find patterns and trends. For example, many web sites facilitate the processing of billion-record web access logs by sorting the log records, either by requested page, requestor or time. In retailing, sorts are often used to aggregate data by various categories and produce rollup and cube reports.

Sorting is also a core utility for database systems in organizing, indexing, and reorganizing data. Pre-sorting data can dramatically reduce database load times:

- Database tables are usually organized in a B-tree with records in key order. Pre-sorted data can be directly loaded into a table without the database performing its own internal sort at a slow speed.
- In data warehouses, fact-table data is summarized by all combinations of its multiple dimensions. Building these aggregates with a sorting program is usually much more efficient than using the database [2].

IBM mainframes have long dominated the commercial sorting world. They combine high bandwidth disks and main memory systems, fast processors, and sophisticated sorting programs. IBM's DFSORT™ and Syncsort™ are the premier sorting products on mainframes.

A number of hardware trends, starting in the 1980s, allowed microprocessor-based, multiprocessor Unix machines to eclipse IBM mainframes in processing power:

- Microprocessors with faster and faster clocks were developed. These processors have become both very fast and relatively inexpensive. They also rely on multiple levels of cache memory to run at full speed.
- Symmetric multiprocessor machines (SMPs) were built to allow multiple processors to work on the same task.
- Non-Uniform Memory Access (NUMA) machines have been developed to get around the bandwidth limitations of a single memory bus SMP machines.
- Main memory prices decreased to the point that multi-gigabyte memories are now standard.
- 64-bit addressing was incorporated to allow a single process to exploit large memories.
- Commodity disks became cheaper and faster -- indeed, mainframes now use the same disks as other systems (albeit with a different interface).
- Volume managers and RAID storage processors have been developed that the allow the bandwidth of many disks to be combined as virtual disk devices whose speed is limited only by the raw hardware.

Nsort is designed to take advantage of today’s large, multiprocessor Unix systems. It can access files stored on multi-disk devices at high speeds. Nsort has sophisticated buffer management to overlap computation and I/O. Its multi-threaded code allows multiple processors to be used for one sort. Nsort's algorithms pay particular attention to processor-cache locality to make the best use of fast microprocessors. By using 64-bit addressing, Nsort can perform a one-pass sort for very large data sets. The size limitation for one-pass sorts is the user's budget for main memory, not a 2GB or 4GB barrier imposed by 32-bit addressing.

For a user, all these are fine points. The real question is how well does it work? Just how fast is Nsort, and how does it compare to other sort products? On a standard sort benchmark, Nsort running on an Origin2000 system is an order of magnitude faster than its competition. In a separate test, Nsort sorted a terabyte of data in 2.5 hours. The next section describes these results.

MinuteSort

MinuteSort is a standard sorting benchmark [3]. The benchmark measures the number of 100-byte records that can be sorted in one minute of elapsed time. The input records have 10-byte random keys. The minute limit includes the time to:

- read the input file
- sort the data
- create and write the output file

There are two categories for the MinuteSort benchmark: Indy (a custom, "benchmark special" sort program) and Daytona (a commercial, general purpose sort program). The first winner of the Indy MinuteSort benchmark was AlphaSort [3], a sort program designed to show that RISC processors could be used for high-performance sorting. It used striped input and output files to achieve high-bandwidth disk i/o - one input file and output file on each disk. AlphaSort also demonstrated that judicious use of processor caches is crucial to sort performance. Unfortunately, it's sort algorithm was extremely dependent on the first few bytes of each key containing evenly distributed values. All sort algorithms perform best with random data, but AlphaSort's performance would have dropped by two orders of magnitude if the first four bytes of all keys contained the same value.

Until Nsort in March 1997, there was not an official Daytona MinuteSort entry. However on October 1, 1996 Syncsort and DEC announced new sort performance results [4] on an SMP AlphaServer 8400 5/440 using 8GB of main memory and an unspecified number of processors and disks. These results used the same type of records as MinuteSort (100 bytes, random keys) and were touted as a "World Record" compared to "previous sorting records" (presumably AlphaSort):

Syncsort/DEC Results

Time	Sort Size
73 seconds	1 GB
378 seconds	5 GB

While the Syncsort/DEC results did not exceed any of the previous Indy MinuteSort results, they far surpass the speeds of commercial sorts on mainframes.

Nsort is a commercial product that does both one and two-pass sorts, handles many data types and is not particularly tuned to the MinuteSort benchmark. Over the past few years, we have run the MinuteSort benchmark to demonstrate the performance of Nsort on moderately large SGI Origin2000 systems.

In 1997 using an Origin2000 configuration of 14 195-Mhz R10000 processors, 7GB of main memory, and 49 disk drives, Nsort was able to sort 5.3GB of data in 58 seconds. This one-pass (no temporary files) sort used a single input file and single output file while setting a new record for Daytona MinuteSort. Nsort achieved a sort speed (data sorted per elapsed seconds) of 92 MB/sec.

In September 1997, Nsort set a new Daytona MinuteSort record by sorting 7.6GB in 60 seconds using a two-pass sort. The Origin2000 system included 32 processors, 8GB of main memory, and 121 disks:

- 1 system disk
- a 60-disk XLV volume for input and output files
- 60 temporary disks

A two-pass sort requires nearly twice as much work as a one-pass sort, since the data must be read from disk twice and written to disk twice. With Nsort, this extra work can be handled by adding more processors and disks.

In September 1999, in a public demonstration at the grand opening of SGI's FISC (Financial Industries Solutions Center) in New York City, Nsort again broke its Daytona MinuteSort record sorting 12 Gigabytes in 58 seconds. The sort speed for this one-pass sort was 206 MB/sec, more than an order of magnitude faster than Syncsort/DEC's speed of 13.2 MB/sec (5.0GB in 378 seconds). The Origin 2000 system consisted of 64 processors and 6 RAID storage processors, each with 10 disks. The RAID storage processors provided high-bandwidth disk i/o while simultaneously protecting against single disk failures.

Terabyte Sort

In order to demonstrate Nsort's ability to sort large data sets, we sorted a terabyte of data (10,000,000,000 100-byte records with random 10-byte keys) in 2.5 hours. The Origin2000 system for this September 1997 result included 32 processors, 8GB of main memory, and 559 4GB disks:

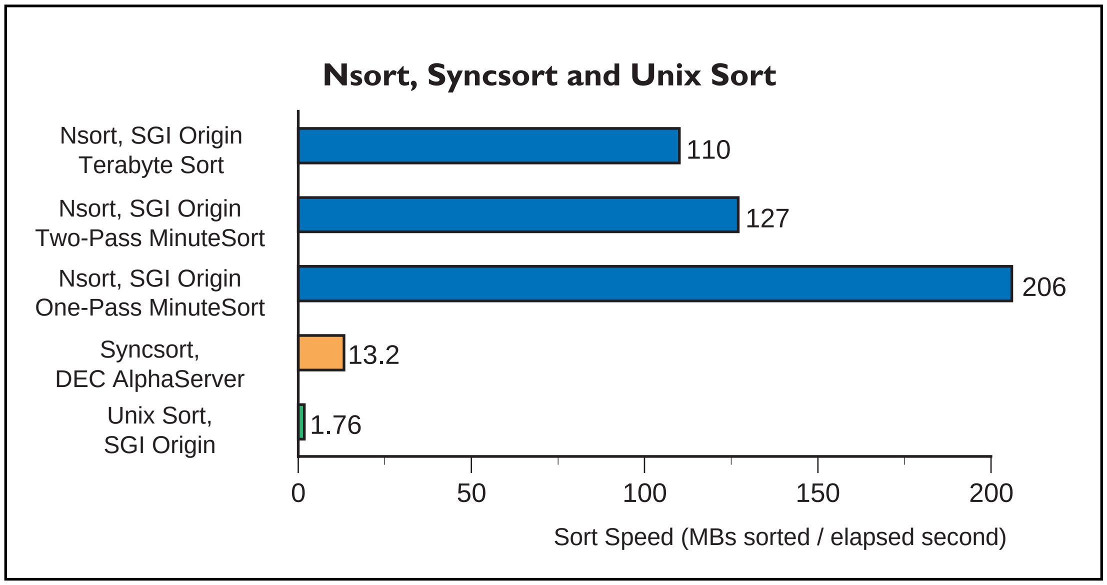
- 1 system disk
- a 280-disk XLV volume for input and output files
- 278 temporary disks

Nsort read a terabyte input file from the 280-disk file system, partially sorted the data and wrote it to the temporary disks. The partially sorted data was then read from the temporary files and merged to produce a 1-terabyte output file. To save on disk space, the input file was overwritten to produce the output file.

Note that the 110 MB/sec speed of the terabyte sort was not much lower than the 127 MB/sec speed of the two-pass MinuteSort, even though more than two orders of magnitude more data was sorted. Fairly uniform performance can be expected for two-pass sorts with data sizes between 1 gigabyte and 1 terabyte. For MinuteSort-type records, this is a consistent sort rate of more than a million records per second.

Competitive Comparison

The following bar chart compares the sort rates for the Nsort terabyte sort, Nsort two-pass MinuteSort, the latest Nsort one-pass MinuteSort, and Syncsort world record. Also included is a 1.76 MB/sec sort rate (1GB sorted in 567 seconds) for the standard Unix sort program measured on an Origin2000 in 1997.

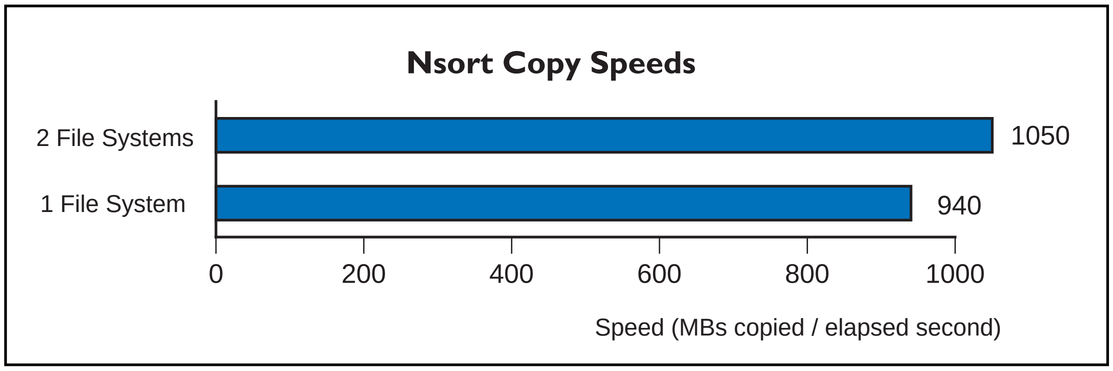


Copying

In addition to sorting, Nsort can perform file copying at very high speeds. Nsort has copied a 100GB file with the following rates:

- 1050 MB/sec copying from one 280-disk file system to another 280-disk file system
- 940 MB/sec copying within a single, 280-disk file system

The copy speeds are illustrated below:

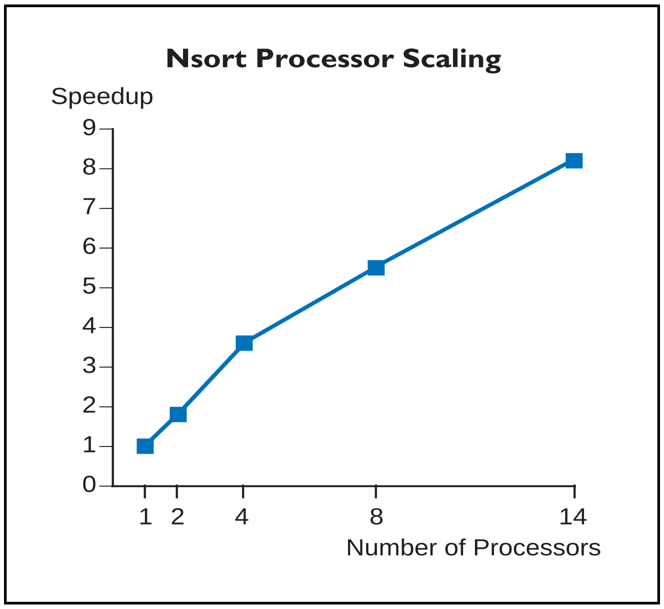


Processor Scaling

To demonstrate Nsort's scalability with multiple processors, we measured the elapsed times for a 2GB sort with increasing numbers of R10000 processors. For each sort, the elapsed time, the sort rate (MBs sorted per elapsed second), and the speedup (multiple processor efficiency relative to a single processor) were measured. The following table and graph show these results. The speedup from one processor to four is nearly linear, beyond four processors the speedup is still impressive. This shows that Nsort performance can be dramatically increased by using additional processors.

Origin2000 Nsort Results for 2GB, One-Pass Sorts

Processors	1	2	4	8	14
Elapsed Seconds	172	93	48	31	21
Sort Rate	12	22	43	66	98
Speedup	1	1.8	3.6	5.5	8.2



Size of Sort Data

The sort rate can vary depending on the size of the data sorted. The following table presents the sort rates for a variety of data sizes. Each sort was done with 8 R10000 processors, and used 100-byte records with random keys. For each case the three following results are shown:

elapsed seconds
sort rate (MBs sorted per elapsed second)
Nsort memory usage (MBs)

Sort Times, Rates and Memory Usage for a Variety of Sort Sizes

Number of Records	100k	200k	400k	1m	2m	4m	10m	20m	100 m	200 m
Sort Size (Bytes)	10m	20m	40m	100 m	200 m	400 m	1g	2g	10g	20g
One-Pass	1.0 10 111	1.3 15 113	1.6 25 140	2.2 45 222	3.2 63 335	5.5 73 573	12.8 78 1259	26.5 76 2432		
Two-Pass									227 44 225	429 47 295

For the small sorts, the startup and shutdown times for Nsort prevent the higher sort rates seen with larger sorts. If the input data cannot fit in main memory, a two-pass sort must be done (runs of sorted records are written to temporary disks then read back as they are merged and written to the output file).

Two-pass sorts are slower than one-pass sorts because they require more CPU use and twice the disk bandwidth, although they require much less main memory.

Key Types

All Nsort performance data presented so far used a single 10-byte character key containing random data. Performance data for a variety of key types will now be presented:

character	a series of ascii characters or unsigned binary 1-byte integers
integer	a signed binary number
floating	an IEEE floating point number
decimal	a decimal number represented as a series of ascii digits

For each key type, both 4 and 8 byte key lengths were tested. In all cases 10 million 100-byte records were sorted with one R10000 processor. Only one disk was available for these tests. The elapsed times of the sorts were all identical (bound by the speed of the disk), so the Nsort processor times are presented instead. (All tests presented so far have measured elapsed time.)

Key Type	Key Bytes	CPU Seconds
character	4	74
	8	73
binary	4	74
	8	74
floating	4	75
	8	75
decimal	4	77
	8	122

The above table shows the character, binary and floating key types all yield similar performance, either with 4-byte or 8-byte keys. This is because the keys contained randomly generated data, only the first 4 bytes of each key needed to be examined to resolve comparisons. Indeed the key length is almost irrelevant for random data. For longer keys, performance is a function of the distribution of the key data. Worst-case key distributions will be examined in the next section.

The key length of decimal data is relevant, even with random data. This is because Nsort must translate the ascii representation of the key into binary. As the length of a decimal key grows, it both takes longer to translate each key and increases the number of key translations that must be done.

Record and Key Length, Key Distribution, and Number of Keys

In this section we will examine Nsort single-process performance as a function of record length, key distribution, and number of keys. The following record sizes and corresponding input sizes were tested:

Record Size	4	8	20	40	100	200	400
Sort Size	256MB	512MB	640MB	800MB	1000MB	1200MB	1200MB
Number of Records	64m	64m	32m	20m	10m	6m	3m

For each record length, character keys of lengths of 4, 8, 20, 40, 100, 200 and 400 bytes were tested (as permitted by the size of the record). For each key length the worst-case key distribution was used. These key distributions force Nsort to examine all key bytes during the sort.

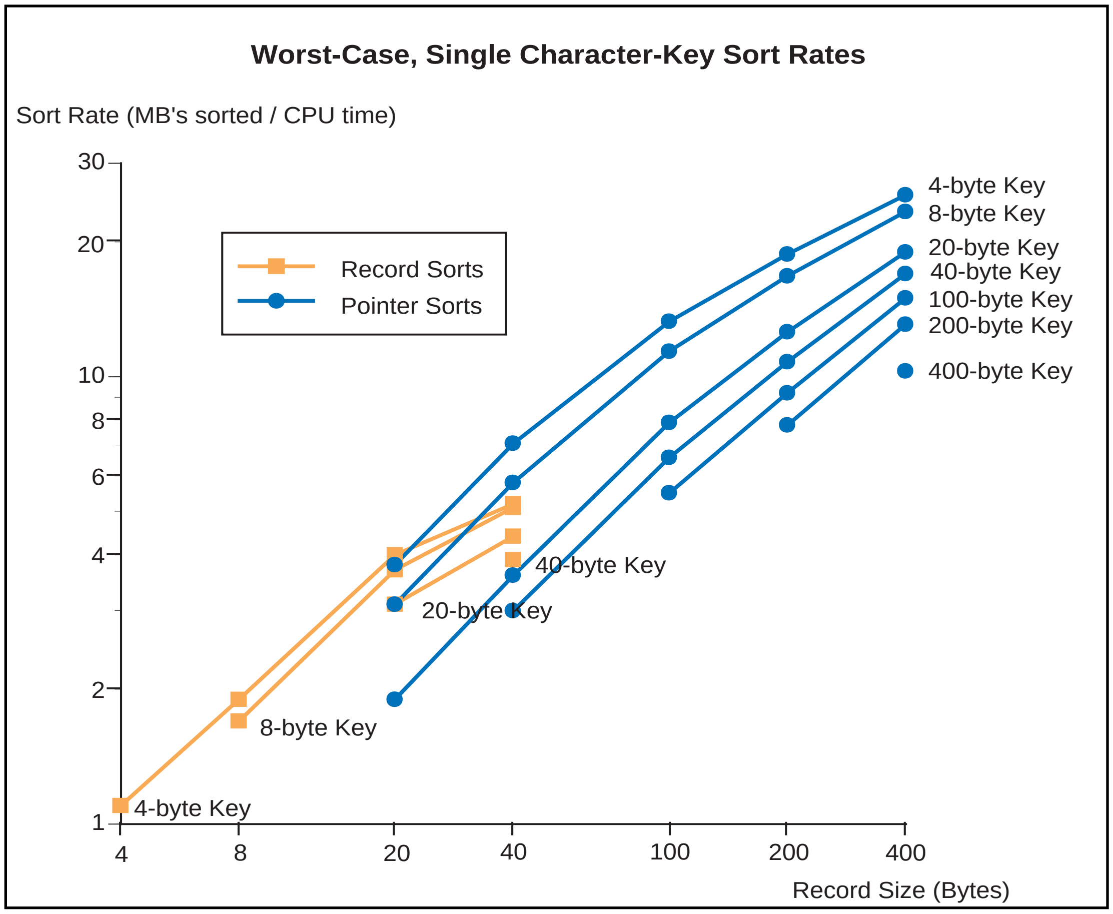
There are two internal methods Nsort uses to sort records, record sort and pointer sort. Both methods were used in the tests. With a record sort, records are moved in Nsort process memory in order to bring the records into sorted order. Record sorts tend to work best for short records (less than 32 bytes). With a pointer sort, a pointer to the record is moved inside Nsort process memory many times in order to arrange the records in sorted order. The record itself is copied only once for a one-pass sort, and twice for a two-pass sort. Pointer sorts tend to work better for longer records (longer than 32 bytes).

All of the sorts were one-pass sorts, used character keys, and were performed with one R10000 processor. As only one disk was available at the time of the tests, Nsort CPU times (not elapsed times) were used to calculate sort rates (Megabytes sorted per CPU second).

Both single keys and multiple keys were tested. The tests for single character-keys of varying lengths are now given. Record sort results are presented for record lengths of 40 bytes and lower. Pointer sort are presented for record lengths of 20 bytes and higher.

Worst-Case, Single Character-Key Sort Rates (MBs Sorted / CPU Time)

Sort Type	Key Bytes	Record Bytes						
		4	8	20	40	100	200	400
Record	4	1.1	1.9	4.0	5.2			
	8		1.7	3.7	5.1			
	20			3.1	4.4			
	40				3.9			
Pointer	4			3.8	7.1	13.3	18.8	25.5
	8			3.1	5.8	11.4	16.8	23.4
	20			1.9	3.6	7.9	12.6	19.0
	40				3.0	6.6	10.8	17.0
	100					5.5	9.2	15.0
	200						7.8	13.1
	400							10.3



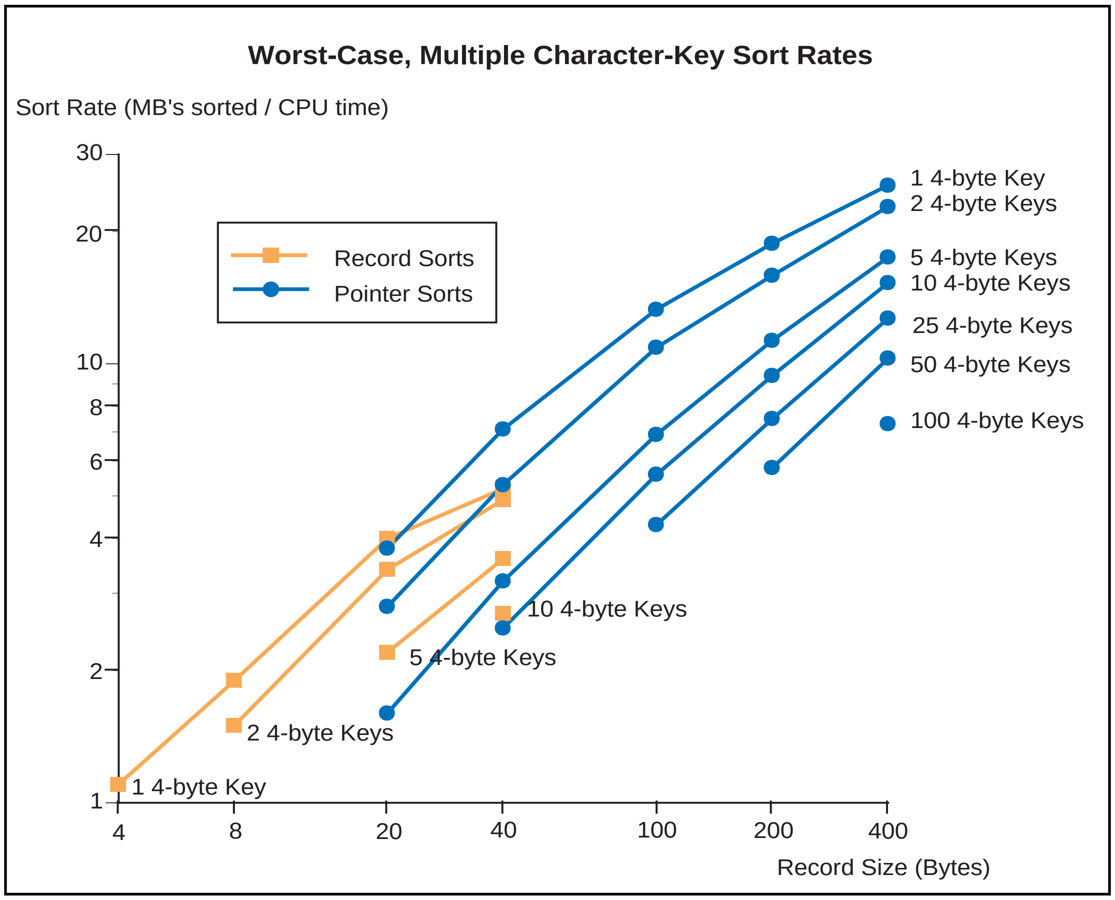
The primary trend of the above data is that the sort rate increases as the record size increases. The secondary trend is the sort rate decreases as the length of the key increases. The worst-case degradation is a factor of 2.5 as the length of the key increases from 4 bytes to 400 bytes.

Pointer and record sorts can be compared for the 20- and 40-byte record tests. Record sorts are always faster for 20-byte records. With 40-byte records, pointer sorts provide the best and worst sort rates, depending on the key size. This is because the longer key lengths cause additional cache misses with pointer sorts. Whereas with record sorts there are no additional cache misses with the longer keys.

Not all sorts in the real world use a single key. The results of using multiple 4-byte character keys are now presented. These tests use the same number of key bytes as the single character-key tests, but broken into separate 4-byte keys. For instance, instead of a single 20-byte key, 5 4-byte keys are used. As demonstrated in the Key Types section, the worst case for multiple 4-byte integer or floating keys should be similar to these character-key results.

Worst-Case, Multiple Character-Key Sort Rates (MBs Sorted / CPU Time)

Sort Type	Key Bytes	Record Bytes						
		4	8	20	40	100	200	400
Record	4	1.1	1.9	4.0	5.2			
	8		1.5	3.4	4.9			
	20			2.0	3.6			
	40				2.7			
Pointer	4			3.8	7.1	13.3	18.8	25.5
	8			2.8	5.3	10.9	15.9	22.8
	20			1.6	3.2	6.9	11.3	17.5
	40				2.5	5.6	9.4	15.3
	100					4.3	7.5	12.7
	200						5.8	10.3
	400							7.3



The use of multiple short keys instead of one large key causes Nsort to use additional CPU time and degrades the sort rate somewhat. These results are similar to the single key tests, with slightly worse degradation (a factor of 3.5 in the worst case). The primary and secondary trends observed in the single-key tests are still present:

- the sort rate increases with the record size
- the rate decreases as the number of keys increases

Importantly, there are no order-of-magnitude dropoffs in performance for non-random data as one would find in Indy MinuteSort programs. This is the kind of sound performance one expects from a commercial sorting program.

Conclusion

The Nsort program provides breakthrough speeds for commercial sorting. Its ability to access single files at high speeds, use 64-bit addressing, and scale performance with multiple processors make it the superior choice among commercial sorting programs.

Availability

Nsort is available now for SGI Origin and Sun Solaris servers. For pricing on IRIX, contact SGI, http://www.sgi.com. For pricing on Solaris, contact Ordinal. More information about Nsort can be found online at http://www.ordinal.com.

Please send comments about this paper to documentation@ordinal.com.

References

[1] Knuth, D.E., The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, MA, 1973.

[2] Kimball, R., The Data Warehouse Toolkit, John Wiley & Sons, New York, 1996, p. 222.

[3] Nyberg, C., T. Barclay, Z. Cvetanovic, J. Gray, D. Lomet, "AlphaSort: A RISC Machine Sort", Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, MN, 1994.

[4] Syncsort and Digital Equipment Corp., "SyncSort Announces World Record Set on Digital's AlphaServer System", Press Release, October 1, 1996.