Tiny Data Compression with td512

L. S. Leonard, February 14, 2022

# Introduction

Tiny data compression for as few as 64 bytes is not supported by standard compression programs. Now with td512 you can reasonably compress data from 16 to 512 bytes. td512 is available under the GPL-3.0 License at <https://github.com/lsleonard/tiny-data-compression>. Compared with QuickLZ, a fast compression program that is designed to compress smaller data sets, td512 gets as good or better compression for 512-byte blocks of most data types. Zstandard produces excellent compression but at one-third the speed of td512. Both QuickLZ and Zstandard steadily decline in compression ratio as the number of bytes decreases to 128, and at 64 bytes, produce compression only for highly compressible files. td512 has good compression at 64 bytes with the td64 interface. td512 combines extended text and string modes for 128 to 512 bytes and uses the td64 interface to compress any remaining bytes in the input. The td512 algorithm emphasizes speed, and based on data in this paper, gets 32% average compression for 512-byte blocks at 330 Mbytes per second on the Squash benchmark test data (see [https://quixdb.github.io/squash-benchmark/#](https://quixdb.github.io/squash-benchmark/)) running on a 2 GHz quad-core processor. For 64-byte blocks on this benchmark data, td512 gets 25% average compression at 290 Mbytes per second.

The Zstandard algorithm uses LZ77 and Huffman coding. Although Huffman coding, with its optimal compression using frequency analysis of values, works well for larger datasets, for tiny datasets the compression modes used in td512 approach or exceed the results of using the Huffman algorithm, and at a higher speed. With the focus on speed of execution, arithmetic coding is also not a practical algorithm for applications of tiny data. One area where high-speed compression using td512 might be applied is in a heap manager, an example of where a variety of data types might be encountered.

For compression and speed comparison with td512, this paper presents compression performance using QuickLZ, where its streaming mode supports compression of as few as 200 to 300 characters ([quicklz.com](http://quicklz.com/)). The final section of the paper discusses the fundamentals of td512 and td64, the compression interface that the td512 algorithm uses. The appendix shows performance data for 8- to 512-byte blocks. Also presented in the appendix are compression values from running Zstandard v1.5.1 (<http://facebook.github.io/zstd/>) and the arithmetic encoding program fpaq0 (<http://mattmahoney.net/dc/fpaq0.cpp>) on the Squash benchmark test data using 512-byte blocks (Table 3). These values give a picture of the compression achievable when runtime is less critical to the application.

# Comparison of Compression Performance: td512 and QuickLZ

The td512 algorithm is packaged with a testbed that runs the algorithm iteratively over an input file using a fixed block size until a final block, possibly smaller, concludes the run (see <https://github.com/lsleonard/tiny-data-compression>). In addition to running td512 with a 512-byte block size, the QuickLZ public distribution of version 1.5.1 Beta 7 was modified to iterate over a 512-byte block size. Both programs were run on the Squash benchmark test data on a MacBook with a 2 GHz Quad-Core Intel Core i5 processor. The test beds for td512 and QuickLZ tests read file data into memory and compute loop count based on file size to pick the best (fastest) compression and decompression speed.

Compression performance for td512 v2.1.8 and QuickLZ is shown in Table 1. Across all the benchmark files, average compression percent, assuming a block of data from one file is as likely as from another, for td512 is 32%, and for QuickLZ is 22%. QuickLZ gets better compression for file geo.protodata, and nci and ptt3 are only slightly better than td512. The file geo.protodata has 512-byte blocks with more than 64 unique values, which is the limit that is currently handled by td512 extended string mode. Other files with fewer unique values that primarily use extended string mode get similar results to QuickLZ because string mode is an algorithm that finds repeated strings, just as QuickLZ does. Text files compress well with the td64 text mode algorithm that uses weighted encoding of 23 predefined characters. Also, the files sum and x-ray are much better compressed by td512 than QuickLZ because these files contain frequently repeated values in small blocks. For example, 16 or more 0s in a 64-byte block that are compressed using a td64 compression mode called single value. Blocks that use either the text or single value modes get similar compression results for blocks from 64 to 512 bytes.

On average, td512 compresses about 15% faster and decompresses about 25% slower than QuickLZ on the Squash benchmark files. But the tradeoff is 45% better average compression for td512. Files with text or random data compress quicker with td512, while most other files compress quicker with QuickLZ. Some of the fast decompression with QuickLZ relates to lower compression than td512, though QuickLZ decompresses two to three times faster for half the files.

Table 1. Compression Performance Data: td512 v2.1.8 and QuickLZ with 512-Byte Blocks

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Squash Benchmark  Filename | td512  Compr.  Percent | QuickLZ  Compr.  Percent | td512  Compress  MB/sec | QuickLZ  Compress  MB/sec | td512  Decompress  MB/Sec | QuickLZ  Decompress  MB/Sec |
| alice29.txt | 36.89 | 8.43 | 327073120 | 224983712 | 232552000 | 557010944 |
| asyoulik.txt | 32.37 | 6.93 | 316108608 | 217324640 | 215825872 | 555235584 |
| cp.html | 30.31 | 19.25 | 253639184 | 264548384 | 238864080 | 528516128 |
| dickens | 38.07 | 4.67 | 392515360 | 226428352 | 199199600 | 617870016 |
| fields.c | 40.58 | 33.98 | 506818176 | 441584160 | 405454528 | 505976480 |
| fireworks.jpeg | -0.54 | -1.56 | 1043160960 | 438053408 | 2.4619E+10 | 2.048E+10 |
| geo.protodata | 9.24 | 13.82 | 197976624 | 263528896 | 463234368 | 619225152 |
| grammar.lsp | 42.68 | 40.15 | 520139776 | 469640800 | 520139776 | 495659552 |
| kennedy.xls | 62.42 | 59.87 | 414383904 | 429776288 | 517199392 | 631288768 |
| lcet10.txt | 36.45 | 8.31 | 348655232 | 225081216 | 223900320 | 525241376 |
| mozilla | 31.53 | 30.86 | 205898224 | 259013712 | 280963456 | 639153472 |
| mr | 44.44 | 30.15 | 199702848 | 257770544 | 612066560 | 646322816 |
| nci | 56.48 | 62.05 | 267136736 | 323940608 | 272033632 | 727729152 |
| ooffice | 16.11 | 12.18 | 143857072 | 216359840 | 217753584 | 556759488 |
| osdb | 1.35 | -1.74 | 165790240 | 265370832 | 659626176 | 1.0718E+10 |
| paper-100k.pdf | 10.73 | 9.41 | 605917120 | 411244960 | 2694736896 | 3657142784 |
| plrabn12.txt | 38.25 | 3.52 | 355879616 | 223913088 | 235283696 | 624892352 |
| ptt5 | 73.2 | 74.94 | 266190864 | 368689664 | 865457024 | 865133248 |
| reymont | 31.44 | 25.51 | 153286800 | 230760192 | 164913200 | 375435744 |
| samba | 40.6 | 33.56 | 218308208 | 264891456 | 245865328 | 586604416 |
| sao | -0.59 | -1.75 | 777855168 | 319103424 | 1.1529E+10 | 10970432512 |
| sum | 39.04 | 27.72 | 184734288 | 240503136 | 281176480 | 440558144 |
| urls.10K | 34.33 | 23.46 | 187874496 | 216159792 | 217633920 | 632959424 |
| webster | 28.35 | 17.60 | 170239936 | 217385648 | 175490288 | 453864288 |
| x-ray | 20.65 | -0.98 | 257936336 | 280883008 | 786399424 | 2389092864 |
| xargs.1 | 27.61 | 18.31 | 464970016 | 418891872 | 394042368 | 489739168 |
| xml | 43.5 | 35.74 | 189643072 | 255693856 | 198672368 | 470866816 |
| Average  Performance | td512  32.05 | QuickLZ  22.01 | td512  338358962 | QuickLZ  295241685 | td512  1758015084 | QuickLZ  2250386381 |

# Fundamentals of td512 and td64 Interfaces

With the td512 interface, you can call the td512 and td512d functions to compress and decompress 1 to 512 bytes. The td512 interface performs compression of 16 to 512 bytes, but accepts 1 to 15 bytes and stores them without compression. Along with its extended text and string modes, td512 acts as a wrapper that uses the td64 interface to compress blocks of 64 bytes until the final block of 64 or fewer bytes is compressed. Unlike td64, td512 outputs the number of bytes processed and a pass/fail bit is stored for each block compressed, and the compressed or uncompressed data is output.

You can call the td64 and td64d functions to compress and decompress 1 to 64 values. For fewer than 6 bytes, td64 calls the td5 interface. The td5 interface is not used by td512 because the number of bytes generated is often more than the number of values to compress. Compression of these miniscule datasets requires bit handling not supported by td512. The td64 interface returns pass (number of compressed bits) or fail (0) and outputs only compressed values. Decompression requires input of the number of original values and data that successfully compressed.

## Encoding Used by the td512 Interface

The td512 interface uses the text mode that is part of the td64 interface, an extended string mode, and td64 to compress data in memory. For 1 to 127 values, one block of 64 followed by the remaining values are processed by td64. For 128 to 512 values, the algorithm verifies that of the first 96 values, 94% are standard text characters and that ¾ are predefined characters. After text mode is confirmed, all values are checked for the high bit clear. Only the standard text mode is used. If text mode is not selected, the first 64 values are scanned for data that would be best compressed by td64. If only 1 or 2 or more than 40 unique values, or 35 or more repeats of a single value are found, then td64 is selected. If a single value occurs 18 times and there are more than 14 unique values, then both extended string mode and td64 are called on these 64 values and the mode that compresses best is selected. If extended string mode is called, it runs until 64 unique values are encountered. It then continues processing strings that start with values from these first 64 unique values until up to 64 more values not in the first 64 are encountered. If text mode or string mode fails, or string mode compresses fewer than all values, the remaining values are handed off in blocks of 64, or possibly fewer for the final block, to td64.

## Encoding Used by the td64 Interface

The td64 interface integrates the following encoding modes: fixed bit coding, text mode, single value mode, string mode, extended string mode, and 7-bit mode. Initially, the program searches up to 1/2 of the input values for unique values and count, accumulates a high-bit value, and counts the number of frequently occurring text characters. If more than the accepted limit of unique values is encountered, then if all characters have the high bit clear, 7-bit mode is used; otherwise, the program fails, assuming random data for this block. If the values match to indicate text data, text mode is used if it gets at least 11% compression. Processing continues by looking for any single value that occurs 25% percent of the time in addition to looking for new unique values. If the limit of unique values is exceeded, single value, if active, is used. If fewer than the limit of unique values is encountered, single value mode, if active, is used first if there are 5 or more unique values; otherwise, fixed bit coding is used. When the limit of unique values is exceeded, single value mode is used first, then string mode is used to encode repeated strings of two or more characters if no more than 32 unique values occurred. If string mode fails and all characters have the high bit clear, 7-bit mode is used. If all modes fail, the program fails to compress the input data. Each of the td64 compression modes is described next.

Fixed bit coding is well known as using only the number of bits required to encode the number of unique characters in a dataset. The td64 algorithm predetermines the maximum number of unique values in the data that will allow compression to occur. With these limits known ahead of time (see uniqueLimits25 in td64.c), decisions about compressibility can be made quickly. This mode is programmed to get 25% (16 unique values) to 83% (2 unique values) compression for 64 input values.

Text mode uses the most frequent characters as defined by Morse code plus space, carriage return and comma to identify text data that can be compressed (see Table 2). When 90 percent of the data values are one of these characters, as in most standard text, text mode gets 35% compression for 64 input values. From 3 to 7 bits are output for each character, based on frequency of occurrence. The adaptive text mode implemented in td64 supports replacement of the 8 lowest frequency characters with characters that are likely to occur in non-standard text. At present, XML (or HTML) and C code characters are handled (see XMLTextChars and CTextChars in td64.c).

Table 2: Most Frequently Occurring Text Characters Plus Space, Carriage Return, and Comma

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Bits | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 7 | 7 | 7 | 7 |
| Char | sp | e | t | a | i | n | o | s | h | r | d | l | u | c | cr | m | g | f | , | y | w | p | b |

Single value mode encodes a value that occurs at least 25% of the time in the data block as a 1 bit and all other values as a 0 bit plus an 8-bit value. This results in 11% compression for 64 input values, but is often much higher as checking for a single value is stopped when the number of occurrences reaches the minimum. When the data block being compressed contains fewer unique values than the unique limit, extended string mode is used to compress the non-single values.

String mode encodes repeated strings of at least two characters in length for data blocks with at most 32 unique values. Single repeated characters are also encoded and save at least one bit. The algorithm uses the first position where unique characters occur as the beginning for repeated strings, which requires at most 5 bits. This algorithm is the most time-intensive of the modes and is used after all but the 7-bit mode. String mode output is used if it gets at least 6% compression, or 12% when 7-bit mode can be used, for 64 input values. String mode encodes the unique values, when all high bits are 0s, by using 7-bit mode.

Extended string mode encodes repeated strings of at least two characters in length, as well as single repeated characters, for data blocks with up to 64 unique values. This mode functions similarly to string mode, but records all two-value occurrences rather than just the first one for each unique value. This means that locations for repeated strings require up to 9 bits for 512-character blocks and for repeated characters up to 6 bits. Extended string mode continues to function once 64 unique values are reached until up to 64 additional values not in the first 64 are encountered. Within the td64 interface, extended string mode is used only by single value mode.

7-bit mode encodes the 7 lower bits of each value, leaving off the high zero bit, in groups of 7 bytes. This mode gets 11% compression for 64 input values and is used last because other modes can get higher compression. 7-bit mode requires at least 16 input values.

## Encoding Used by the td5 Interface

The td5 interface uses three modes to encode 1 to 5 values:

* For 4 or 5 input values, encode 1 or 2 unique values.
* For 1 to 5 input values, encode frequently occurring text characters (see text mode below).
* For 2 or 3 input values, encode 2 unique 4-bit nibbles.

# Summary

This paper has shown that over the Squash benchmark data, td512 compresses 512-byte blocks at 32%, on average, while QuickLZ gets 22%. Although QuickLZ has much better decompression speed, td512 gets 47% better compression. The compression achieved by Zstandard is the best presented in this paper at 36.9%, but for most files the program takes at least twice as long to run as td512. A huge benefit of td512 is that its compression for 64 bytes, a number of values that very few compression programs can support, is 25%. Both QuickLZ and Zstandard produce limited compression for 128-byte blocks, and for 64-byte blocks only 4 benchmark files are compressed (table xxx), while td512 with 16-byte blocks, compresses all but three of the files that are compressible with 512-byte blocks. The arithmetic compressor fpaq0 supports 64-byte blocks and averages 26% compression (table xxx). Over the Squash benchmark data, td512 gets close to the 34% average compression using fpaq0 with 512-byte blocks, showing that td512 closely approaches the compression achieved by this form of arithmetic encoding.

The implementation of td512 uses fixed bit coding, text, single value, string, and 7-bit modes to compress data. This variety of encoding modes reflects the fact that data compression is data dependent. The td64 interface can be easily modified to support additional compression modes for special data types. New modes in the td64 interface will not affect the td512 interface.

## Appendix: Performance Data for td512

Table 3 shows performance data for td512 v2.1.8 run iteratively with 128- to 512-byte blocks, and fpaq0 and Zstandard v1.5.1 run with 512-byte blocks on the Squash benchmark data on a 2 GHz quad-core processor with percent compression and MB per second.

Table 3: Performance Data for td512 v2.1.8 on 128- to 512-byte Blocks and fpaq0 and zstd with 512-byte Blocks Run on the Squash Benchmark Test Data

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Filename | 128 | 128 Comp | 128 Decomp | 256 | 256 Comp | 256 Decomp | 512 | 512 Comp | 512 Decomp | fpaq0 | zstd |
| alice29.txt | 35.2 | 282693312 | 206082656 | 36.58 | 310385696 | 214209856 | 36.89 | 327073120 | 232552000 | 39.19 | 36.09 |
| asyoulik.txt | 30.26 | 272128256 | 206225696 | 31.78 | 302364736 | 210739056 | 32.37 | 316108608 | 215825872 | 35.94 | 34 |
| cp.html | 11.8 | 180904416 | 221648640 | 23.14 | 209387232 | 228865120 | 30.31 | 253639184 | 238864080 | 30.83 | 38.49 |
| dickens | 36.32 | 340759104 | 204630608 | 37.63 | 325949664 | 195351152 | 38.07 | 392515360 | 199199600 | 39.61 | 35.43 |
| fields.c | 27.46 | 297333312 | 377966080 | 36.07 | 401801792 | 479569888 | 40.58 | 506818176 | 405454528 | 36.10 | 46.22 |
| fireworks.jpeg | -1.51 | 918604480 | 1.3677E+10 | -0.74 | 976928576 | 2.0516E+10 | -0.54 | 1043160960 | 2.4619E+10 | -3.69 | -1.72 |
| geo.protodata | 7.48 | 195046064 | 449196960 | 10.92 | 174908560 | 355053888 | 9.24 | 197976624 | 463234368 | 9.84 | 21.75 |
| grammar.lsp | 30.8 | 366462112 | 531571424 | 39.18 | 390104864 | 531571424 | 42.68 | 520139776 | 520139776 | 39.51 | 53.24 |
| kennedy.xls | 59 | 278459712 | 489422048 | 62.53 | 336847904 | 474099456 | 62.42 | 414383904 | 517199392 | 59.72 | 68.27 |
| lcet10.txt | 35.65 | 303092320 | 217731632 | 36.38 | 330049504 | 220203312 | 36.45 | 348655232 | 223900320 | 39.13 | 36.84 |
| mozilla | 25.74 | 162342384 | 310111392 | 30.17 | 185236480 | 321246336 | 31.53 | 205898224 | 280963456 | 32.27 | 38.26 |
| mr | 45.03 | 155615008 | 596896768 | 45.79 | 202583744 | 638852032 | 44.44 | 199702848 | 612066560 | 56.16 | 49.38 |
| nci | 48.07 | 206492896 | 233901776 | 54.42 | 242843488 | 263799440 | 56.48 | 267136736 | 272033632 | 67.40 | 74.88 |
| ooffice | 12.06 | 126700416 | 196737936 | 16.22 | 138435056 | 196167072 | 16.11 | 143857072 | 217753584 | 21.63 | 21.21 |
| osdb | -0.7 | 140012832 | 1061539200 | 0.54 | 166298704 | 1022888896 | 1.35 | 165790240 | 659626176 | 13.39 | 5.2 |
| paper-100k.pdf | 8.24 | 581818176 | 2275555584 | 10.14 | 669281024 | 3103030272 | 10.73 | 605917120 | 2694736896 | 7.38 | 11.3 |
| plrabn12.txt | 36.45 | 343695456 | 224330064 | 37.78 | 357463648 | 225273968 | 38.25 | 355879616 | 235283696 | 39.08 | 34.06 |
| ptt5 | 72.07 | 205697808 | 835856640 | 72.9 | 246857136 | 945149184 | 73.2 | 266190864 | 865457024 | 84.02 | 80.69 |
| reymont | 23.42 | 115456480 | 144044560 | 28.06 | 138522672 | 161442192 | 31.44 | 153286800 | 164913200 | 37.05 | 38.7 |
| samba | 31.02 | 166014080 | 225101840 | 37.37 | 196084912 | 242727632 | 40.6 | 218308208 | 245865328 | 35.02 | 46.72 |
| sao | -1.56 | 782049408 | 1.0272E+10 | -0.78 | 804609344 | 1.1331E+10 | -0.59 | 777855168 | 1.1529E+10 | 3.33 | -1.76 |
| sum | 34.51 | 147076928 | 281176480 | 38.48 | 168458160 | 298750016 | 39.04 | 184734288 | 281176480 | 43.49 | 43.42 |
| urls.10K | 22.21 | 148778768 | 201344128 | 30.26 | 170120432 | 219265152 | 34.33 | 187874496 | 217633920 | 32.54 | 43.54 |
| webster | 19.03 | 138304016 | 169683232 | 24.72 | 157598704 | 175712688 | 28.35 | 170239936 | 175490288 | 34.35 | 38.55 |
| x-ray | 19.24 | 171062000 | 729155072 | 19.2 | 212642768 | 837540992 | 20.65 | 257936336 | 786399424 | 26.12 | 15.86 |
| xargs.1 | 28.67 | 404321760 | 404321760 | 28.63 | 430527776 | 418891872 | 27.61 | 464970016 | 394042368 | 35.53 | 39.82 |
| xml | 32.23 | 155449296 | 186989440 | 39.36 | 174141712 | 198076048 | 43.5 | 189643072 | 198672368 | 35.16 | 48.47 |
| Block Size  Performance | 128  26.97 | 128 Comp.  280976696 | 128 Decomp.  1293707273 | 256  30.61 | 256 Comp.  311867937 | 256 Decomp.  1630560745 | 512  32.05 | 512 Comp.  338358962 | 512 Decomp.  1758015084 | fpaq0  34.44 | zstd  36.92 |

Table 4 shows performance data for td512 v2.1.8 run iteratively with 32-, 64- and 96-byte blocks on the Squash benchmark test data on a 2 GHz quad-core processor. Performance is percent compression and MB per second. Processing of 96 values involves compression of the first 64 bytes followed by compression of the final 32 bytes. The compression achieved is approximately half the sum of the compression of 32- and 64-byte blocks.

Table 4: Performance Data for td512 v2.1.8 on 32-, 64- and 96-byte Blocks Run on the Squash Benchmark Test Data

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Filename | 32 | 32 Comp. | 32 Decomp. | 64 | 64 Comp. | 64 Decomp. | 96 | 96 Comp. | 96 Decomp. |
| alice29.txt | 30.54 | 179774240 | 195236208 | 34.31 | 202515312 | 206362272 | 33.08 | 173222096 | 202515312 |
| asyoulik.txt | 26.26 | 184629792 | 222738448 | 30.32 | 207593696 | 223933808 | 28.96 | 183546912 | 207938544 |
| cp.html | 16.67 | 175735712 | 265978384 | 21.3 | 224684944 | 278000000 | 19.83 | 188528736 | 273366656 |
| dickens | 31.46 | 206220464 | 195414816 | 35.28 | 209535712 | 209114416 | 34.01 | 202443952 | 209359248 |
| fields.c | 19.32 | 311888128 | 398214272 | 26.63 | 433009728 | 412962976 | 24.43 | 359677440 | 401801792 |
| fireworks.jpeg | -3.09 | 1089318528 | 5595136512 | -1.52 | 905095552 | 1.0258E+10 | -2.03 | 1000756032 | 1.0258E+10 |
| geo.protodata | 0.32 | 242016320 | 1210081664 | 3.38 | 253935760 | 1332449408 | 2.3 | 233901376 | 1222556800 |
| grammar.lsp | 19.62 | 374984512 | 403108352 | 23.89 | 443788992 | 393276416 | 22.92 | 413444416 | 396500000 |
| kennedy.xls | 26.71 | 294886592 | 718593152 | 46.78 | 225920144 | 425338272 | 40.08 | 245762304 | 438562176 |
| lcet10.txt | 31.39 | 204286256 | 208988256 | 34.98 | 210951056 | 222848032 | 33.81 | 200259984 | 210846832 |
| mozilla | 11.73 | 230431216 | 880046784 | 19.25 | 228815072 | 623370432 | 16.73 | 213482704 | 648097984 |
| mr | 42.46 | 321175232 | 1344829184 | 47.03 | 254292736 | 706080576 | 45.26 | 276652704 | 805376704 |
| nci | 34.98 | 231375936 | 638067968 | 46.88 | 150901696 | 287960480 | 42.92 | 157544912 | 318621984 |
| ooffice | 2.07 | 166235024 | 1051836544 | 7.25 | 182628080 | 554551296 | 5.5 | 175571248 | 633528128 |
| osdb | 2.28 | 153799104 | 930499456 | 0.57 | 240863648 | 4364208128 | 1.14 | 192100960 | 1774398976 |
| paper-100k.pdf | 3.87 | 575280896 | 1896296320 | 5.5 | 562637376 | 2497561088 | 5.05 | 517171744 | 2275555584 |
| plrabn12.txt | 31.56 | 216178112 | 203832912 | 35.35 | 218036640 | 222055760 | 34.08 | 206984960 | 209141056 |
| ptt5 | 70.52 | 299775712 | 1056000000 | 72.63 | 298554976 | 1087321984 | 71.9 | 292097888 | 1094277248 |
| reymont | 7.67 | 89754488 | 331492704 | 20.56 | 103341728 | 177696800 | 16.26 | 93879024 | 206223600 |
| samba | 20.27 | 164597616 | 284905792 | 26.35 | 171101856 | 245597040 | 24.28 | 157350304 | 254477360 |
| sao | -3.12 | 952323584 | 5255031808 | -1.56 | 839831360 | 8930965504 | -2.08 | 845707776 | 9031063552 |
| sum | 22.31 | 151146240 | 796666624 | 29.47 | 147644784 | 382400000 | 26.85 | 142686560 | 472098784 |
| urls.10K | 11.1 | 132145112 | 211408320 | 17.36 | 137907488 | 206556928 | 15.28 | 130354064 | 203917232 |
| webster | 18.89 | 141352080 | 212180032 | 24.17 | 154169712 | 192638544 | 22.42 | 140104848 | 194787200 |
| x-ray | 19.6 | 322975840 | 868440256 | 21.91 | 344677472 | 955059200 | 21.15 | 346977856 | 821226880 |
| xargs.1 | 24.15 | 346992544 | 397410272 | 27.7 | 384272736 | 378024384 | 26.35 | 354938944 | 381122944 |
| xml | 13.17 | 120045808 | 284489856 | 26.27 | 136119584 | 192796400 | 21.92 | 123636024 | 206477136 |
| Block Size  Performance | 32  19.73 | 32 Comp.  291826855 | 32 Decomp.  965071292 | 64  25.26 | 64 Comp.  291586216 | 64 Decomp.  1332106673 | 96  23.42 | 96 Comp.  280325399 | 96 Decomp.  1235244064 |

Table 5shows performance data for td512 v2.1.8 run iteratively with 8- and 16-byte blocks on the Squash benchmark test data on a 2 GHz quad-core processor. Performance is percent compression and MB per second. To get 8-byte blocks requires changing the value of MIN\_VALUES\_TO\_COMPRESS in td512.h to 8. The default value of 16 causes any number of values from 1 to 15 to be immediately copied to the output without compression.

Table 5: Performance Data for td512 v2.1.8 on 8- and 16-byte Blocks Run on the Squash Benchmark Test Data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Filename | 8 | 8 Comp. | 8 Decomp. | 16 | 16 Comp. | 16 Decomp. |
| alice29.txt | 7.5 | 192031568 | 161624864 | 22.9 | 176028928 | 184574032 |
| asyoulik.txt | 5.38 | 197755136 | 169619248 | 19.8 | 184357888 | 196822320 |
| cp.html | -6.29 | 207620256 | 289447072 | 8.44 | 217725664 | 261734032 |
| dickens | 8.42 | 187626720 | 155181040 | 23.82 | 198057712 | 174149472 |
| fields.c | -1.43 | 253409088 | 285897440 | 13.49 | 348437504 | 305479424 |
| fireworks.jpeg | -12.48 | 488464288 | 1558139264 | -6.21 | 854812480 | 3516942848 |
| geo.protodata | -10.14 | 319644192 | 878429632 | -3.06 | 435985280 | 1185880064 |
| grammar.lsp | -2.04 | 268738880 | 324651008 | 12.71 | 343070944 | 403108352 |
| kennedy.xls | -5.27 | 349065760 | 846828992 | 12.98 | 379139904 | 904871680 |
| lcet10.txt | 9.16 | 193802912 | 166181472 | 24.2 | 204286256 | 194155600 |
| mozilla | -3.81 | 355589120 | 752191488 | 5.75 | 395934656 | 943669248 |
| mr | 17.08 | 310909728 | 679981184 | 34.14 | 367591936 | 1049091392 |
| nci | 5.42 | 258228576 | 448359648 | 28.55 | 301014144 | 644713024 |
| ooffice | -8.85 | 392509376 | 1048967040 | -2.05 | 381483968 | 1437428096 |
| osdb | -11.26 | 282085472 | 897063424 | -2.16 | 282527968 | 928272768 |
| paper-100k.pdf | -5.85 | 433898304 | 1177011456 | 1.35 | 585142848 | 1551515136 |
| plrabn12.txt | 8.07 | 194534112 | 165474240 | 23.94 | 196999584 | 172524528 |
| ptt5 | 48.42 | 371895648 | 768287424 | 63.79 | 365278304 | 1051672128 |
| reymont | -12.82 | 260585168 | 1058489408 | 0.08 | 209827840 | 958935360 |
| samba | 1.35 | 217834992 | 276328480 | 14.47 | 212375040 | 279166880 |
| sao | -12.55 | 516557024 | 1588944896 | -6.25 | 714547648 | 2840557568 |
| sum | 0.28 | 291908416 | 579393920 | 14.38 | 246709680 | 597500032 |
| urls.10K | -8.66 | 177294704 | 221548448 | 5.05 | 169586240 | 212110880 |
| webster | -3.68 | 195754752 | 221685344 | 12.13 | 194221456 | 224051456 |
| x-ray | -2.95 | 293023520 | 567066368 | 12.55 | 358106848 | 804160192 |
| xargs.1 | 2.18 | 226814640 | 262694912 | 16.65 | 320668960 | 363257824 |
| xml | -8.63 | 233704096 | 382324608 | 5.74 | 227730064 | 357735264 |
| Block Size  Performance | 8  -0.12 | 8 Comp.  284121720 | 8 Decomp.  590067123 | 16  13.22 | 16 Comp.  328579620 | 16 Decomp.  805336281 |