DV1674 - Performance Optimization - Assignment 1

Daniele Gagni, Riccardo Storchi
Teacher: Suejb Memeti

1 Introduction

Our goal is to test the performance of the *pigz* (parallel implementation of gzip), a library that wraps the well-known *gzip* and leverages multiple processors and cores for optimized data compression. Our tests aim to observe the behavior of pigz while compressing **4 different randomly generated files** of different sizes (10MB, 100MB, 1GB, 5GB); for each file the compression is executed with **5 different thread configurations** (1, 2, 4, 8, 16 threads).

2 Method

2.1 System Under Test (SUT)

CPU Information

- Number of cores: `4`
- Number of threads: `8`
- Clock speed: `100MHz`
- Cache size: `8 MB`

Memory (RAM)

- Type: `DDR4`
- Total available memory: `16 GB`
- Memory speed: `2400 MT/s`

SSD Information

- Device size: `512 GB`
- Interface: `SATA 3`
- Max throughput (theoretical): `600 MB/s`

OS and Kernel

- OS: `Arch Linux x86_64`
- Kernel: `6.11.1-arch1-1`

2.2 Execution time collection

In order to measure the execution time of pigz, we run the `time` command for each combination of filesize and thread count.

2.3 Performance metrics collection

In order to gather performance metrics related to pigz execution, we used `pidstat`. `pidstat` is a command-line tool part of sysstat suite to monitor, display real-time performance statistics of individual processes or the entire system. To automate the metric collection for each file with different thread configurations, we implemented two shell scripts. The **first script** utilizes two nested loops to run the `pidstat` command shown below: the variable **${j}** iterates over different file sizes—10MB, 100MB, 1GB, and 5GB—while **${i}** iterates through the number of threads. This setup allowed us to generate file names that reflect both the file size and the number of threads used, with the output files saved as **.txt**.

```
pidstat -u -d -r -h -I 1 -e ./pigz -k -p $i test_files/testfile_${j}MB >
    pidstat_metrics/pidstat_${j}MB_${i}_thread.txt
```

In the **second script**, we utilized `awk`, a powerful Linux tool for text processing, frequently used for pattern matching, data extraction, and reporting tasks. This script processes the `.txt` files containing metrics generated by `pidstat`, retaining only the most relevant metrics: `%CPU`, `%MEM`, `kB_rd/s`, and `kB_wr/s`. The script follows a similar structure to the first one, using a double `for` loop to iterate over the `.txt` files, apply a regular expression to extract the metrics, clean the data, and convert `kB_rd/s` and `kB_wr/s` to MB/s.

2.4 Plotting

The graphs are generated starting from the `.csv` files using `gnuplot`

2.5 Flamegraphs

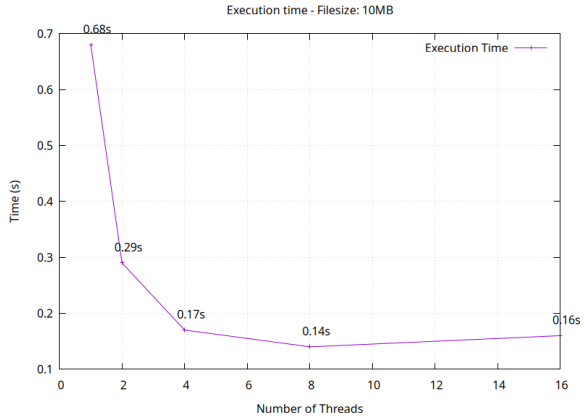
Flame graphs are a visualization of hierarchical data, created to visualize stack traces of profiled software so that the most frequent code paths can be identified quickly and accurately. The x-axis shows the stack profile population, sorted alphabetically (it is not the passage of time), and the y-axis shows stack depth, counting from zero at the bottom. Each rectangle represents a stack frame. **The wider a frame is, the more often it was present in the stacks**. In order to generate the flamegraphs, we used the following commands:

```
perf record --call-graph dwarf -- ./pigz -k -p 1 ./test_files/testfile_10MB
hotspot perf.data
```

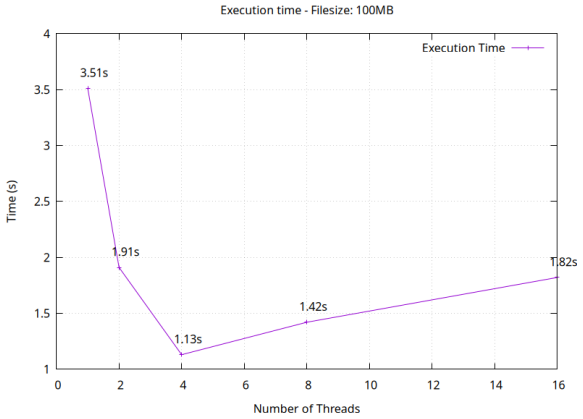
`perf record` is used to record performance data during the execution of a programme. It collects events related to system behaviour, such as CPU statistics, cache usage and other relevant metrics. The `perf` command generates a `perf.data` file, which is then provided as input to `hotspot`. `hotspot` is a standalone GUI for visualizing performance data. We used `hotspot` to display flamegraphs related to the execution of `pigz`.

3 Results

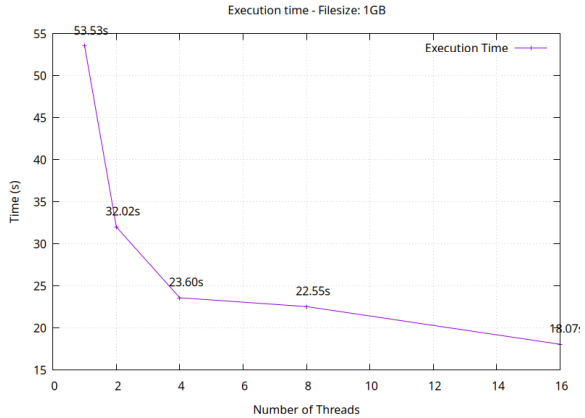
In this section, we present the results of our tests. Due to their fast execution times, it was not possible to gather significant metrics from the compression of 10MB and 100MB files: for this reason, the majority of the graphics will showcase data collected from the compression of the 1GB and 5GB files.



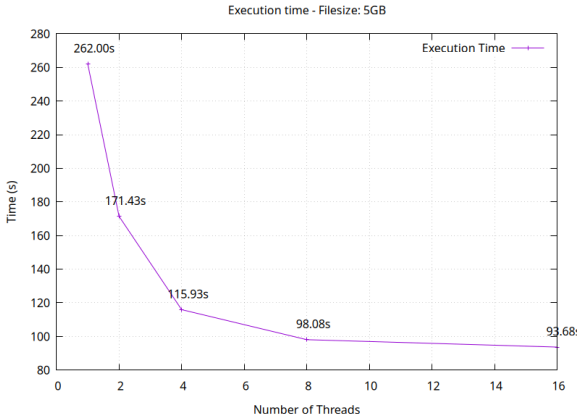
(a) Execution Time - 10MB testfile



(b) Execution Time - 100MB testfile

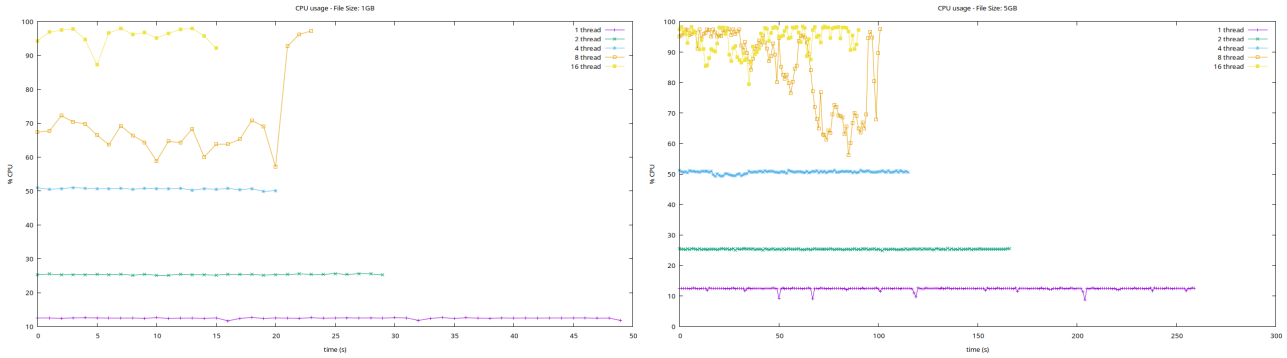


(c) Execution Time - 1GB testfile



(d) Execution Time - 5GB testfile

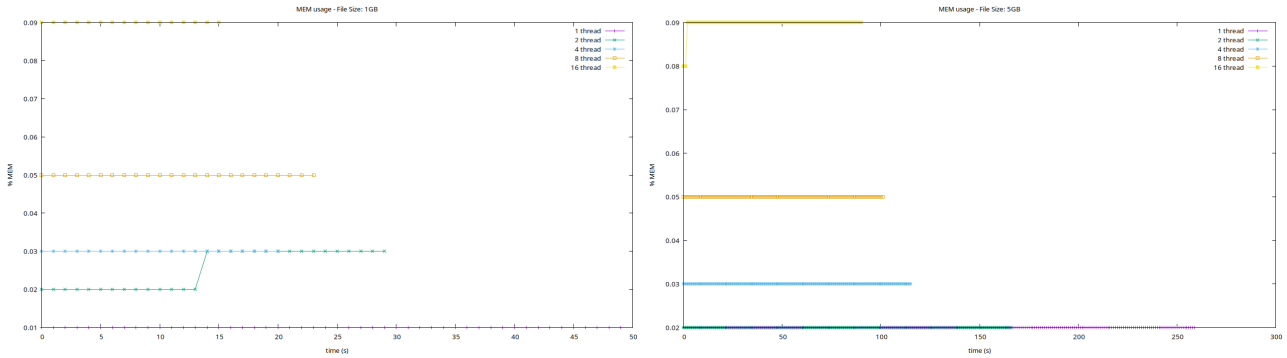
Figure 1: The graphs above display the execution time for each file size under test. On the x-axis we can find the thread count.



(a) CPU usage - 1 GB testfile

(b) CPU usage - 5 GB testfile

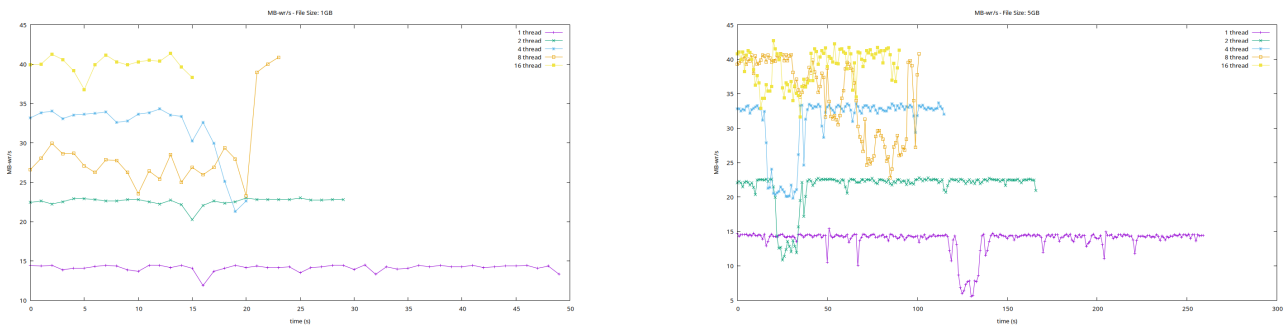
Figure 2: The graphs above show CPU usage for both the 1GB and 5GB test files across varying thread counts, with the third graph presenting the average CPU usage. Notably, at 16 threads, CPU usage approaches saturation, likely because the system under test (SUT) has 8 physical cores, making it difficult to efficiently manage 16 threads.



(a) Memory usage - 1 GB testfile

(b) Memory usage - 5 GB testfile

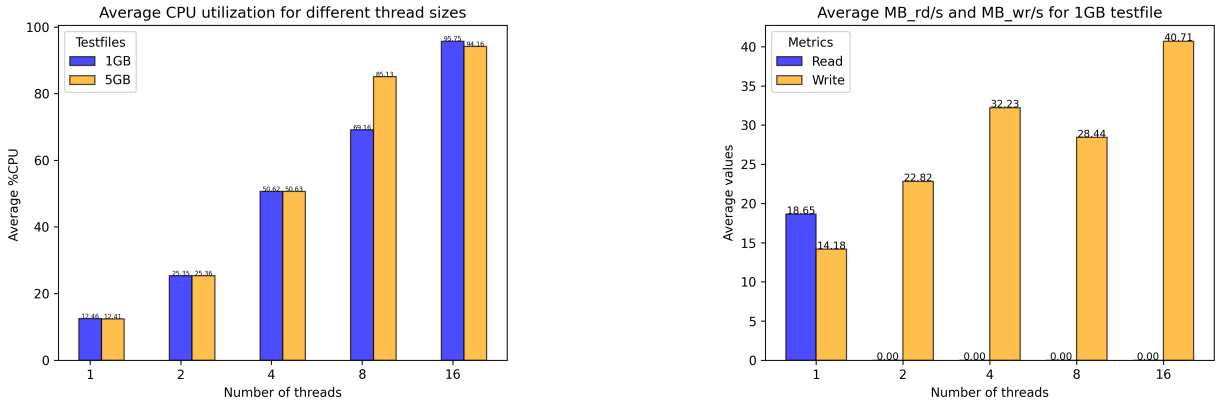
Figure 3: The graph above shows the percentage of memory usage by *pigz* when compressing the 1GB and 5GB testfiles, using varying numbers of threads.



(a) MB_wr/s - 1GB filesize

(b) MB_wr/s - 5GB filesize

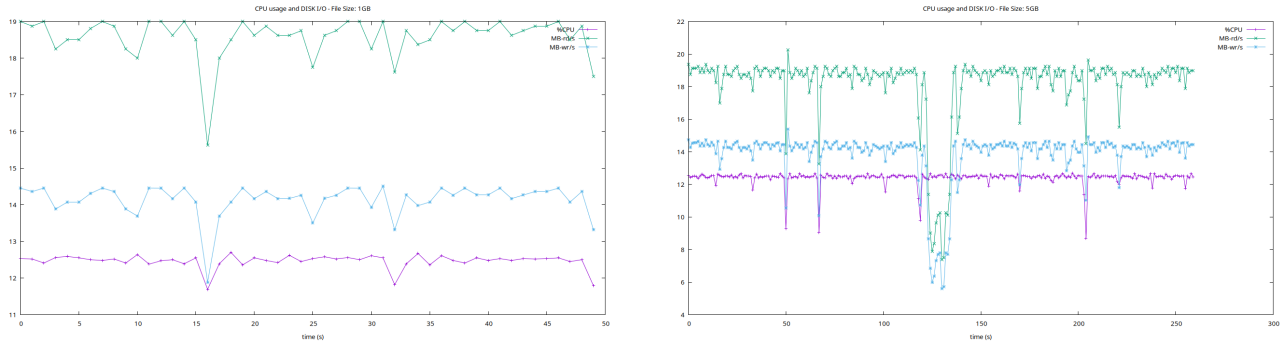
Figure 4: Disk I/O



(a) Average CPU usage

(b) Average Disk I/O - 1GB filesize

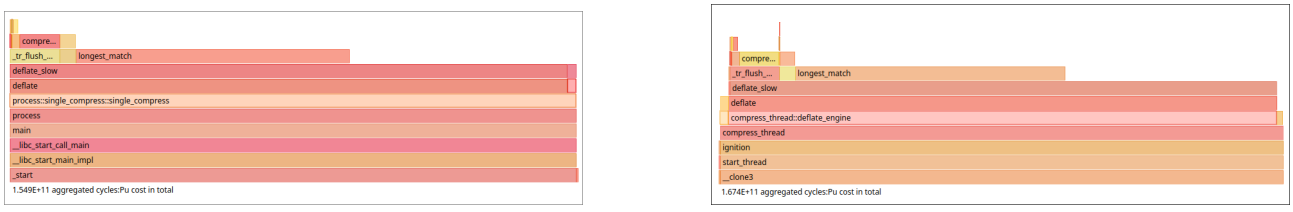
Figure 5: As shown by figure 5b, the average **MB_rd/s** is 0 across 2, 4, 8, and 16 threads. Data collected by `pidstat` indicated that the size of data read from the disk was in the range of a few kilobytes, with reads occurring far less frequently than writes. Since we were able to collect read metrics only in the single-threaded case, this could indicate that reads were not distributed across multiple threads, resulting in a higher value that could be displayed in the graph. For this reason, 4a and 4b only show **MB_wr/s** for different thread counts.



(a) 1GB file size, 1 thread

(b) 5GB file size, 1 thread

Figure 6: %CPU usage and disk I/O comparison. A clear trend emerges: ***whenever there is a drop in CPU usage, there is a corresponding decrease in both data read from and written to the disk***.



(a) Flame graph - 1GB test file, 1 thread

(b) Flame graph - 1GB test file, 4 threads

Figure 7: The figures above illustrate that the hotspot occurs in the `gzip` function `longest_match`, which is invoked by `deflate_slow`. For the sake of conciseness, we have included only the flame graphs for the 1GB file size (1 thread and 4 threads); the other flame graphs are similar to these two. In the case of a multithreaded execution, we can notice the presence of a `compress_thread` function, responsible for managing the various threads. Regardless of the number of threads used, the hotspot consistently remains at `longest_match`.

4 Analysis

4.1 Main Bottleneck

Our tests confirm that ***pigz* is CPU-bound**. This becomes especially evident with higher thread counts, as seen in Figure 2, where the CPU approaches saturation with 8 and 16 threads. While this may not be immediately noticeable at lower thread counts (see Figure 2), the reason is the `-I` flag in `pidstat`, which normalizes CPU usage across all cores. By removing the `-I` flag and using the `-t` flag to display CPU usage per thread, it becomes clear that the core dedicated to *pigz* is nearing full utilization. Screenshots of the `pidstat` output in Figures 8 and 9 show a high CPU load in both single-threaded and multi-threaded executions.

04:56:50 PM	UID	TGID	TID	%usr	%system	%guest	%wait	%CPU	CPU	Command
04:56:51 PM	1000	140338	-	98.00	2.00	0.00	0.00	100.00	7	pigz
04:56:51 PM	1000	-	140338	98.00	2.00	0.00	0.00	100.00	7	|__pigz

Figure 8: pidstat output - 1 thread

02:43:10 PM	UID	TGID	TID	%usr	%system	%guest	%wait	%CPU	CPU	Command
02:43:11 PM	1000	16216	-	397.00	9.00	0.00	0.00	406.00	5	pigz
02:43:11 PM	1000	-	16216	0.00	3.00	0.00	0.00	3.00	5	|__pigz
02:43:11 PM	1000	-	16217	0.00	4.00	0.00	1.00	4.00	7	|__pigz
02:43:11 PM	1000	-	16218	99.00	0.00	0.00	0.00	99.00	1	|__pigz
02:43:11 PM	1000	-	16219	100.00	0.00	0.00	0.00	100.00	4	|__pigz
02:43:11 PM	1000	-	16220	99.00	0.00	0.00	0.00	99.00	6	|__pigz
02:43:11 PM	1000	-	16221	100.00	0.00	0.00	0.00	100.00	3	|__pigz

Figure 9: pidstat output - 4 threads

Additionally, **the amount of data written to disk increases with the number of threads**, as shown in Figure 4. This suggests that increasing the number of threads has little to no impact on disk I/O performance. Even under the highest CPU load (5GB file with 16 threads), the maximum write throughput remains around **40 MB/s**. Considering that a SATA 3 interface (with a theoretical maximum throughput of 600 MB/s) is likely used, the `MB_wr/s` remains well below the saturation point, further confirming that disk I/O is not the limiting factor. Finally, as highlighted by Figure 6, where a drop in CPU usage consistently correlates with a decrease in both data read from and written to the disk, we can further confirm that the program is CPU-bound.

4.2 Impact of File Size and Thread Count on Performance

In figures 1c and 1d, it is evident that increasing the number of threads significantly reduces execution time, particularly up to 4 threads. While there is a further decrease with 8 and 16 threads, the improvement becomes more limited. On the other hand, in Figures 1a and 1b, the execution time with 8 and 16 threads is worse compared to 4 threads. This is likely because, for smaller file sizes, the overhead from thread synchronization outweighs the benefits of parallel compression. Figures 2a and 2b show a clear increase in CPU usage as the number of threads rises. Figure 5a shows that file size has little impact on CPU usage, as the %CPU is similar across different file sizes when the same number of threads is used.

4.3 Memory Usage

As figure 3 shows, even under the most demanding configuration (5GB file with 16 threads), the memory usage is limited to the **0.09%** of the available RAM. **This clearly shows that the program is not memory-bound**. Moreover, the % of memory allocated to *pigz* remains constant suggesting that the program is well optimized in terms of memory usage, requiring no additional memory beyond its initial allocation.

4.4 Effect of System Load

The majority of our tests were conducted with the SUT under light load. Since the cores used by `pigz` are consistently near full utilization, it is reasonable to expect reduced performance when running `pigz` on a heavily loaded system, as more applications will compete for CPU resources. This performance drop will be especially pronounced when using a high number of threads, as tests show that `pigz` alone demands substantial CPU resources, nearing saturation, even in the absence of other running applications.

4.5 Compression Efficiency

Figure 10 shows that the *compression ratio* (*compressed file size / original file size*) remains consistent at 0.76 across all file sizes. Additionally, the compressed file size remains unchanged regardless of the number of threads used during compression. This indicates that neither the execution time nor the number of threads has any impact on the final size of the compressed file.

5 Appendix

```
.rw-r--r--  7.6M daniele daniele  6 Oct 19:48  testfile_10MB_1_threads.gz
.rw-r--r--  7.6M daniele daniele  6 Oct 19:48  testfile_10MB_2_threads.gz
.rw-r--r--  7.6M daniele daniele  6 Oct 19:48  testfile_10MB_4_threads.gz
.rw-r--r--  7.6M daniele daniele  6 Oct 19:48  testfile_10MB_8_threads.gz
.rw-r--r--  7.6M daniele daniele  6 Oct 19:48  testfile_10MB_16_threads.gz
.rw-r--r--   76M daniele daniele  6 Oct 19:48  testfile_100MB_1_threads.gz
.rw-r--r--   76M daniele daniele  6 Oct 19:48  testfile_100MB_2_threads.gz
.rw-r--r--   76M daniele daniele  6 Oct 19:48  testfile_100MB_4_threads.gz
.rw-r--r--   76M daniele daniele  6 Oct 19:48  testfile_100MB_8_threads.gz
.rw-r--r--   76M daniele daniele  6 Oct 19:48  testfile_100MB_16_threads.gz
.rw-r--r--  761M daniele daniele  6 Oct 19:49  testfile_1GB_1_threads.gz
.rw-r--r--  761M daniele daniele  6 Oct 19:50  testfile_1GB_2_threads.gz
.rw-r--r--  761M daniele daniele  6 Oct 19:50  testfile_1GB_4_threads.gz
.rw-r--r--  761M daniele daniele  6 Oct 19:50  testfile_1GB_8_threads.gz
.rw-r--r--  761M daniele daniele  6 Oct 19:51  testfile_1GB_16_threads.gz
.rw-r--r--  3.8G daniele daniele  6 Oct 19:55  testfile_5GB_1_threads.gz
.rw-r--r--  3.8G daniele daniele  6 Oct 19:58  testfile_5GB_2_threads.gz
.rw-r--r--  3.8G daniele daniele  6 Oct 20:00  testfile_5GB_4_threads.gz
.rw-r--r--  3.8G daniele daniele  6 Oct 20:02  testfile_5GB_8_threads.gz
.rw-r--r--  3.8G daniele daniele  6 Oct 20:03  testfile_5GB_16_threads.gz
```

Figure 10: Size of the compressed files with varying number of threads