Analysing Roofline Model For SPEC CPU 2017 Benchmark Suite

Prathyush P.V Vishal Goel

Computer Architecture Project Report

Abstract: In this project, we aim to characterize the benchmarks in the SPEC CPU 2017 benchmark suite as memory bandwidth intensive or CPU-intensive using the Roofline Model technique. We also try to make sense of the execution behaviour of the benchmarks by looking at different phases of their execution. From our experiments, we found that out of the 17 benchmarks we ran, roofline model classified 5 of them as memory bandwidth intensive and the rest as CPU-intensive. While 8 benchmarks achieved an IPC of more than one (i.e. 4 GFlops/sec), only two benchmarks achieved an IPC of more than 2 (i.e. 8 GFlops/sec against the maximum of 64 GFlops/sec achievable on our system). We also show the graphs of CPU and memory usage of the benchmarks, which make it evident that the benchmarks do not utilise the memory bandwidth of our execution environment completely. Further, we found that enabling vectorization in compiler did not improve the performance of the benchmarks.

1 Motivation

Executing benchmark workloads on a computer system helps compare two systems based on their performance and also identify potential optimizations possible in both software or hardware of the system. One of the most widely used benchmark suites for general-purpose high-performance computing research has been the SPEC CPU [1] 2006 benchmark suite. However, it was replaced with the new SPEC CPU2017 suite, released in June 2017. The new suite is said to significantly influence design and optimization research for next-generation microprocessors, memory subsystems, and compilers. SPEC benchmarks are known to potentially take several days or weeks to run using simulators. Thus, it is imperative to characterise the workloads and only work on the useful subset of the suite.

2 Introduction to Roofline Model

Roofline Modelling[3] is a technique to relate processor performance with off-chip memory traffic. It uses the term "operational intensity" which means operations per byte of DRAM traffic, where total bytes accessed are those that go to the main memory after filtering by the cache hierarchy. Thus, operational intensity suggests the DRAM bandwidth needed by a kernel on a particular computer. The model uses operational intensity instead of otherwise popular arithmetic intensity because the latter measures traffic between the processor and cache whereas the former measures traffic between cache and memory, thus allowing to include memory optimizations into the bound and bottleneck model.

Roofline model ties together floating-point performance, operational intensity, and memory performance together in a two-dimensional graph. The Y-axis is attainable floating-point performance and the X-axis is operational intensity. Floating-point performance of the computer is shown by a horizontal line and peak memory performance (GFlops/second)/(GFlops/byte) is shown as a line at a 45-degree angle. These two lines intersect at the point of peak computational performance and peak memory bandwidth. Note that,

Attainable GFlops/sec = Min(Peak Floating Point Performance, Peak Memory Bandwidth x Operational Intensity)

For a given kernel, we can find a point on the X-axis based on its operational intensity. If we draw a vertical line through that point, the performance of the kernel on that computer must lie somewhere along that line.

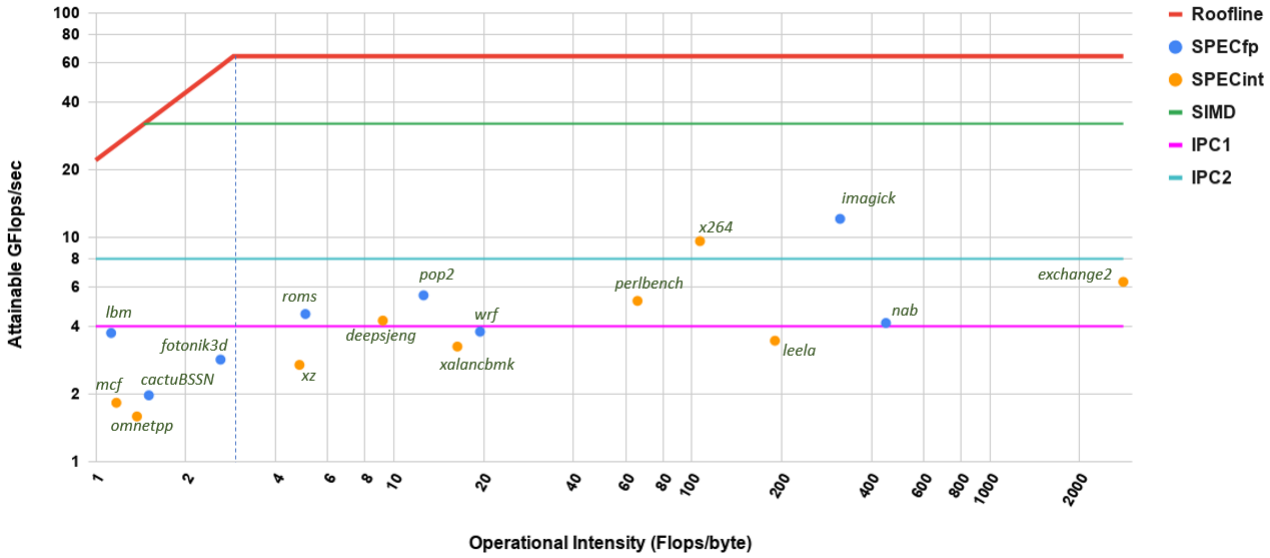


Figure 1: Roofline Model of SPEC CPU17

The Roofline sets an upper bound on performance of a kernel depending on its operational intensity. If we think of operational intensity as a column that hits the roof, either it hits the flat part of the roof, which means performance is compute bound, or it hits the slanted part of the roof, which means performance is ultimately memory bound.

3 Roofline Model for SPEC17

SPEC17 (short for SPEC CPU 2017) has two types of benchmark suites - SPECspeed and SPECrate. Each one of them has two variations (one for integer operations and other for floating point operations). While SPECspeed is about time-based execution, SPECrate focusses on throughput of the system. Thus, the former runs only one copy per benchmark whereas the latter runs multiple copies per benchmark. In this project, we only focus on SPECspeed benchmark suite. SPECfp and SPECint have 10 benchmarks each in the original suite out of which we have worked on 8 and 9 respectively. The rest of the benchmarks ($603.bwaves, 627.cam4, 602.gcc$) either took too long to execute or did not compile.

To analyse the benchmarks, we did all our experiments on an Intel Core i7-8700 machine clocked at 4 GHz. The peak memory bandwidth of the machine is 22.1 GBPS. The slanted line with slope 22.1 is the memory bandwidth ceiling. The cpu ceiling can be calculated using the formula,

Peak = 2 * FMA-Units * Vector-Width * Frequency

The machine we used has 2 FMA units and a vector register width of 256 bits. It can hold 4 double precision floating point numbers. The frequency is taken as 4 GHz. The peak performance of the machine turns out to be 64 GFlops/sec.

For each benchmark, we found the operational intensity (Flops/byte) where total number of number of floating point operations were found by subtracting number of load/store instructions from the total instructions and total bytes transferred from LLC to main memory were found by using number of LLC load misses and LLC store misses. Note that we multiplied the number of AVX instructions of each type (128-bit packed double precision, 128-bit packed single precision, 256-bit packed double precision, 256-bit packed single precision) with 2,4,4 and 8 respectively to get the effective total of flops correct. We used the perf command in linux to measure all the counters needed to build the model.

From our experiments, we found that out of the 17 benchmarks we ran, roofline model classified 5 of them as memory bandwidth intensive and the rest as CPU-intensive. While 8 benchmarks achieved an IPC of more than one (i.e. 4 GFlops/sec), only two benchmarks achieved an IPC of more than 2 (i.e. 8 GFlops/sec against the maximum of 64 GFlops/sec

achievable on our system). We also show the graphs of CPU and memory usage of the benchmarks, which make it evident that the benchmarks do not utilise the memory bandwidth of our execution environment completely. Further, we found that enabling vector processing did not improve the performance of the benchmarks.

4 Analysing Roofline Model for SPEC17

4.1 AI Benchmarks

The three benchmarks deepsjeng, leela and exchange2 use artificial intelligence to play the games of chess, Go and create Sudoku respectively. The number of memory accesses, TLB misses and number of CPU operations of all the three have been shown in Figure 2. Deepsjeng benchmark uses a highly recursive alpha beta tree search algorithm to find out the next chess move given a current state of the game. The algorithm does better depending on the amount of data it can store for each move it anticipates, and thus the benchmark is both CPU and memory bandwidth hungry. The input of the benchmark consists of 12 separate games, beginning of each of which is apparant from the spike in the TLB misses. Similarly, the other benchmarks leela (which uses a monte carlo tree search) and exchange2 (a sudoku generator) also use heavy recursion and thus, are compute-intensive. Further, as seen in the figure, sudoku generator does not require any data and thus has negligible memory bandwidth requirement, leading to a high operational intensity.

4.2 Media and Compression Benchmarks

xz benchmark has two different phases in the execution. xz decompresses and compresses input data and verifies its integrity. The refspeed workload run the workload on two inputs with different parameters. In the first run, it uses a memory buffer of size 6643 MB and a compression factor 4. For the second run it uses a buffer of size 1400 MB and a compression factor of 8. In Figure 3c the two runs are distinguishable. The second phase experiences significantly more LLC misses and TLB misses. Because of this, the CPU thoughput is reduced to half. The number of TLB misses are higher than LLC misses. This particular benchmark has a speedup of 16% when transparent huge paging is enabled in the kernel [data from Ashish Panwar].

x264 and imagick are video compression and image manipulation benchmarks respectively. Both load the media file in the beginning (explaining the high TLB misses in the beginning of the graphs in figures 3a and 3b). This is followed by the compute-intensive encoding and manipulation algorithms respectively. Owing to less memory bandwidth demand, these benchmarks achieve high operational intensity in the roofline model and are the only two benchmarks with an IPC of more than two.

4.3 Memory Bandwidth Intensive Benchmarks

Roofline model characterises five benchmarks (four shown in Figure 4, lbm did not compile) as memory intensive. Fotonik3d models computational electromagnetics, cactuBSSN models a vacuum flat spacetime to solve the Einstein equations from the EinsteinToolkit[1] and Omnetpp simulates a large 10 gigabit Ethernet network. All the three show a linear behaviour in the memory requests, CPU operations and TLB misses. Note that fotonik3d and omnetpp show high TLB misses and incur a speedup of 5% and 8% respectively with transparent huge pages enabled.

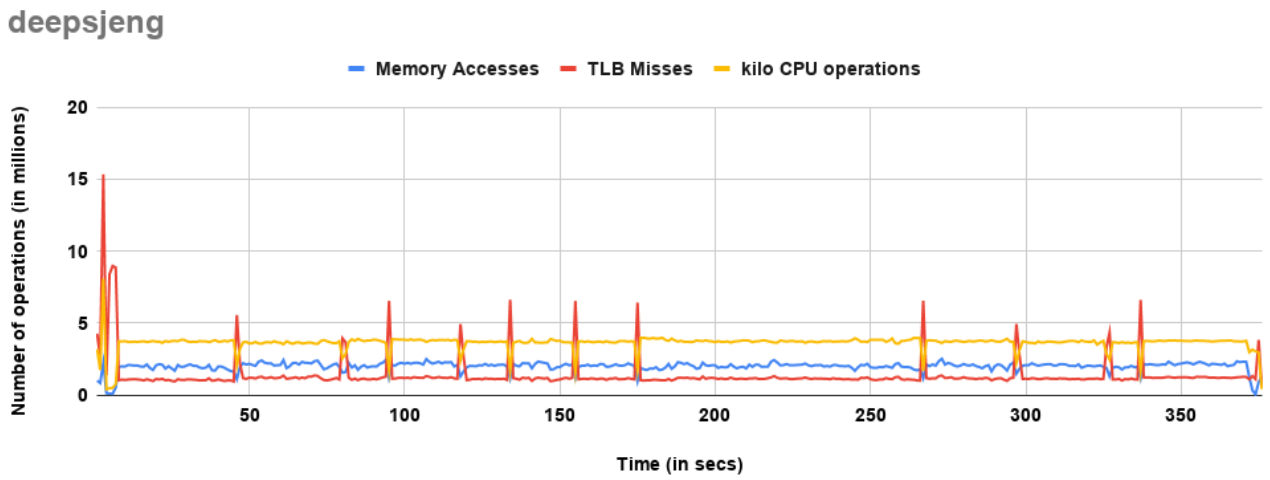
mcf is a benchmark which runs network simplex algorithm for route planning. The different iterations of the simplex algorithm are visible in Figure 4c. The first iteration takes around 100 seconds. But the time required for each iteration decreases gradually, as the algorithm approaches convergence. In the beginning of each iteration, it loads data from memory which is shown as a spike in the number of LLC misses. And after every iteration, high TLB misses are due to the change in the working set of the algorithm (i.e. addition and deletion of edges in the working set of the graph).

4.4 Other benchmarks

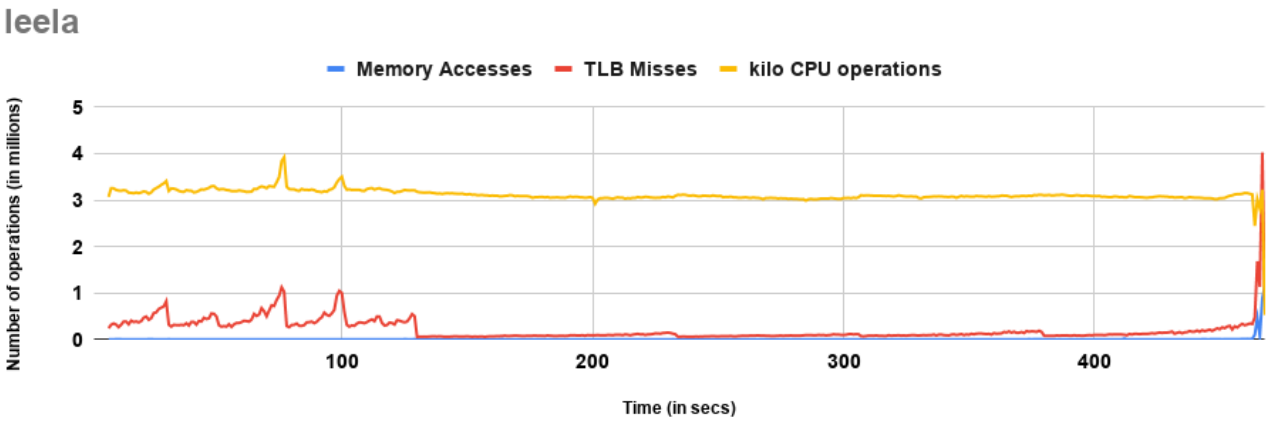
The remaining benchmarks (Figure 5) show a monotonic behaviour w.r.t. number of memory accesses, TLB misses and number of CPU operations, after the initial spikes of data loading.

5 Vectorization of Benchmarks

By default, all the benchmarks in SPEC17 are compiled with flag -fno-tree-loop-vectorize which prevents compiler from vectorizing any loops. The number of vector instructions executed for most of the benchmarks are 0 or less than 100.



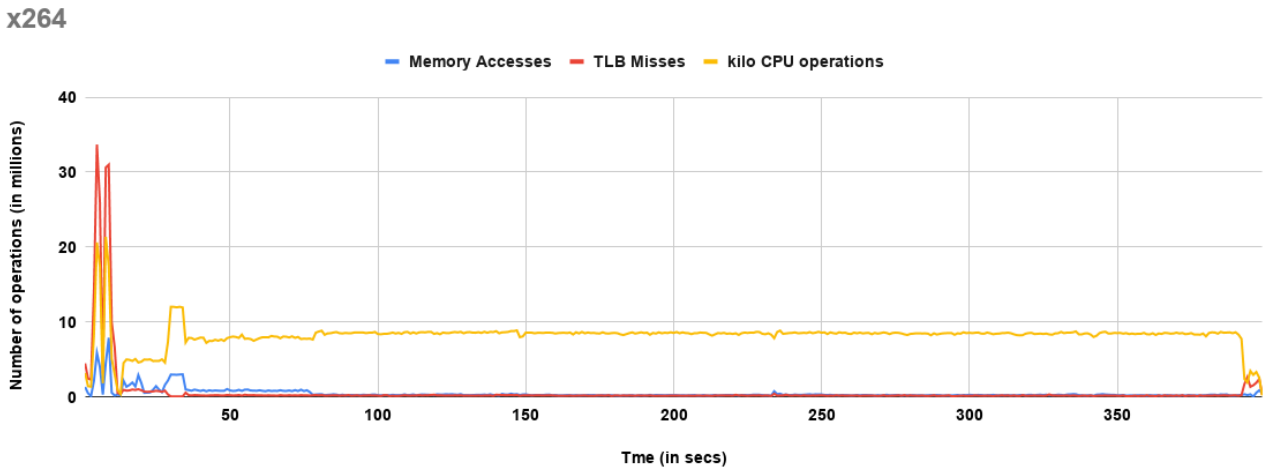
(a) Alpha-Beta tree search (Chess)



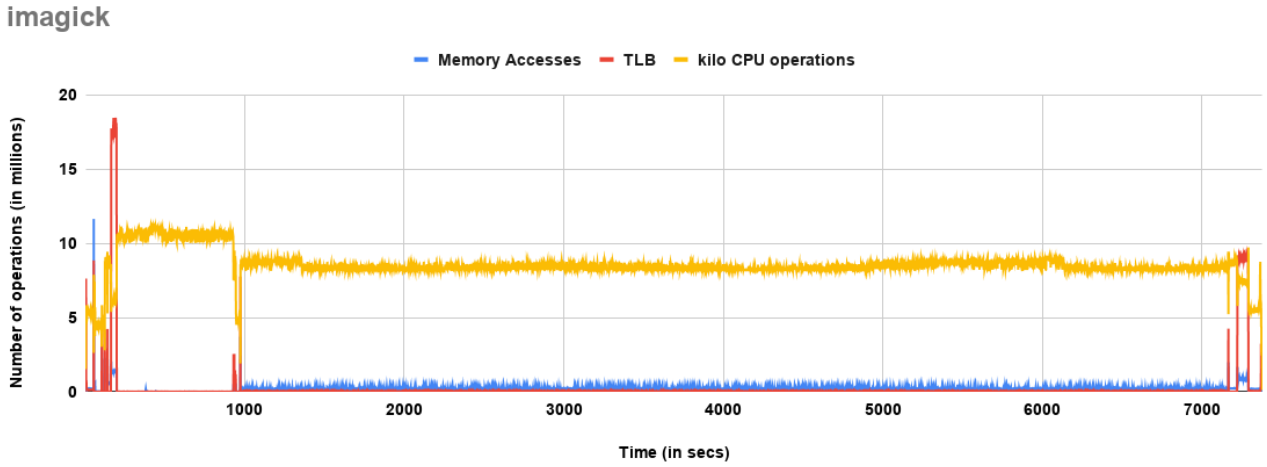
(b) Monte Carlo tree search (Go)

(c) Recursive solution generator (Sudoku)

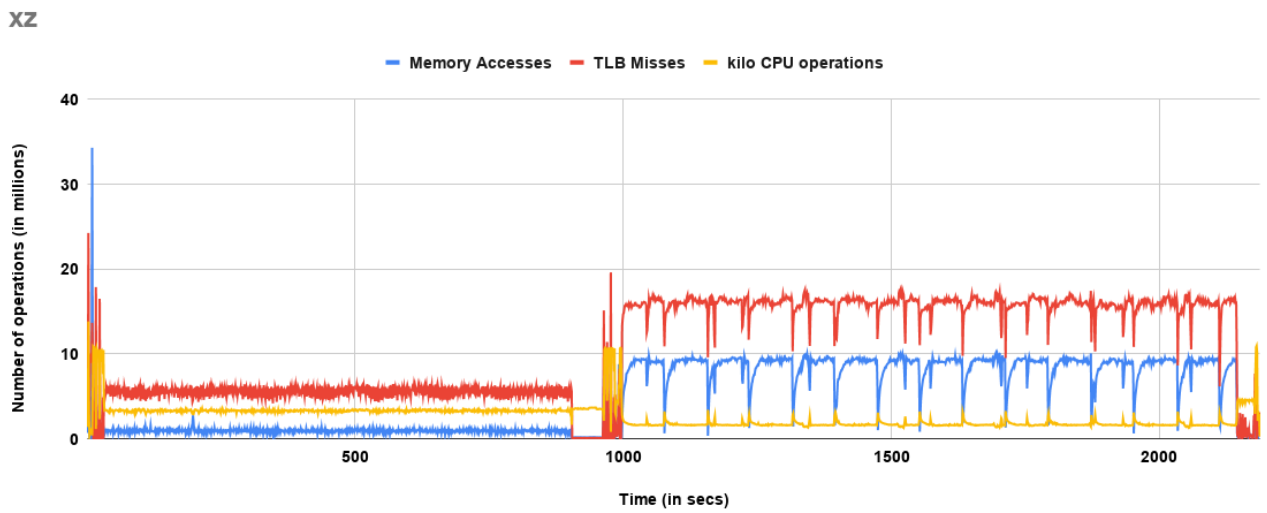
Figure 2: AI Benchmarks



(a) Video Compression

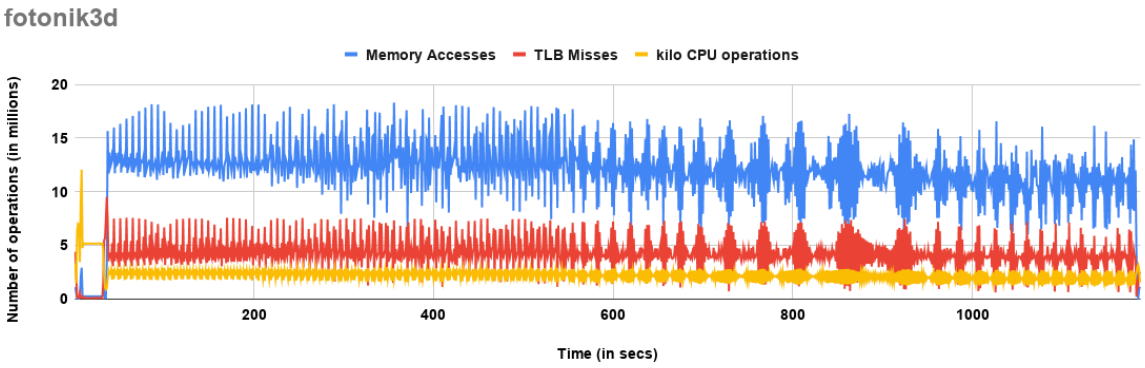


(b) Image Manipulation

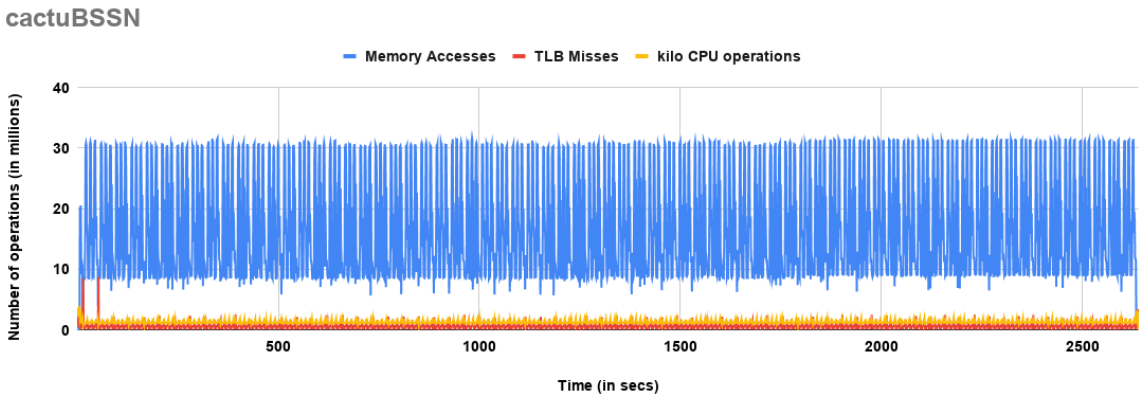


(c) General Data Compression

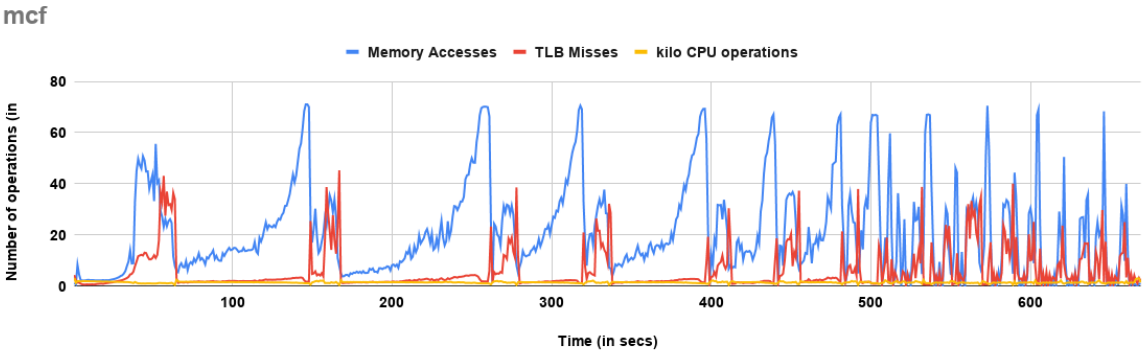
Figure 3: Media and Compression Benchmarks



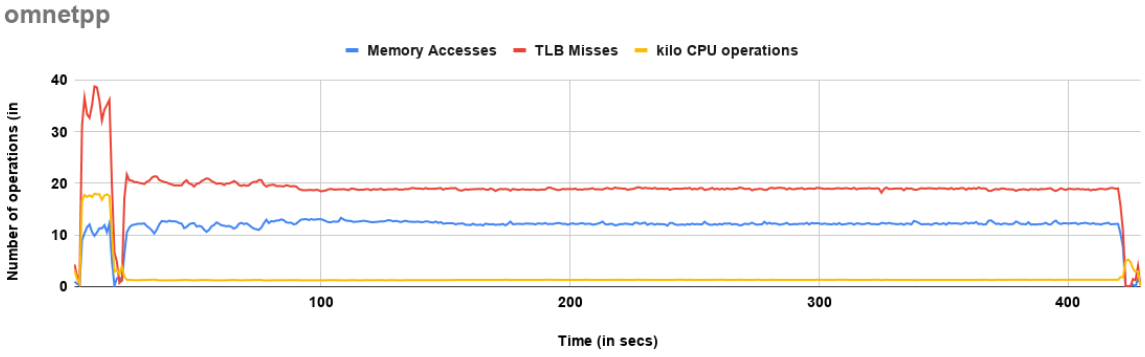
(a) Computational Electromagnetics



(b) Physics: relativity

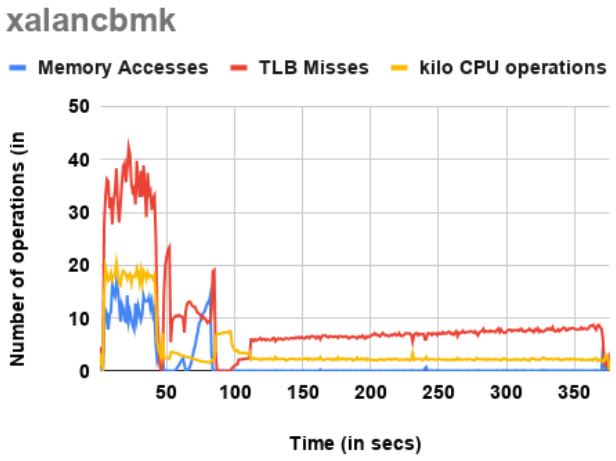


(c) Simplex Algorithm for route planning

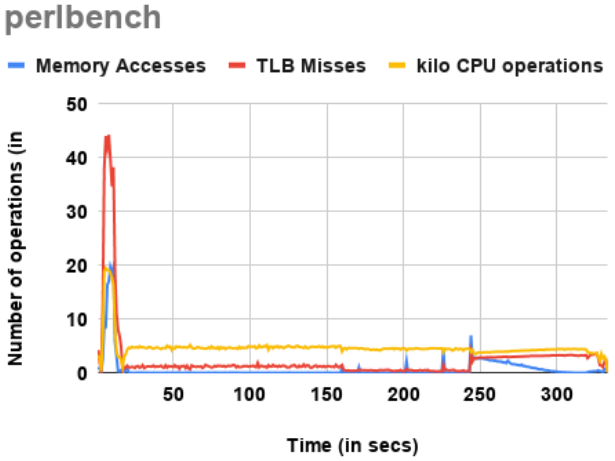


(d) Discrete Event simulation of a 10 Gigabit Ethernet

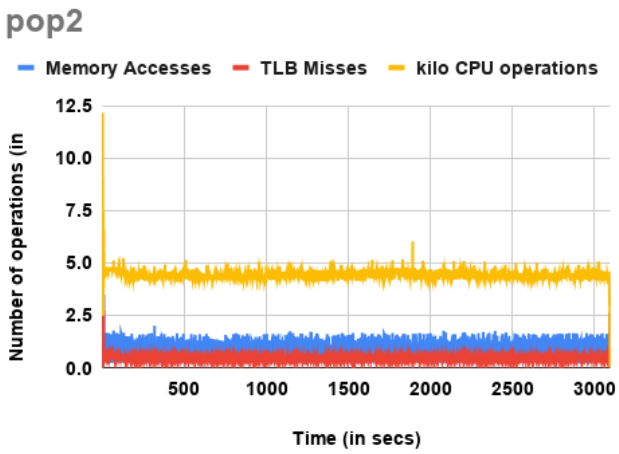
Figure 4: Memory Bandwidth Intensive Benchmarks



(a) XML to HTML conversion via XSLT

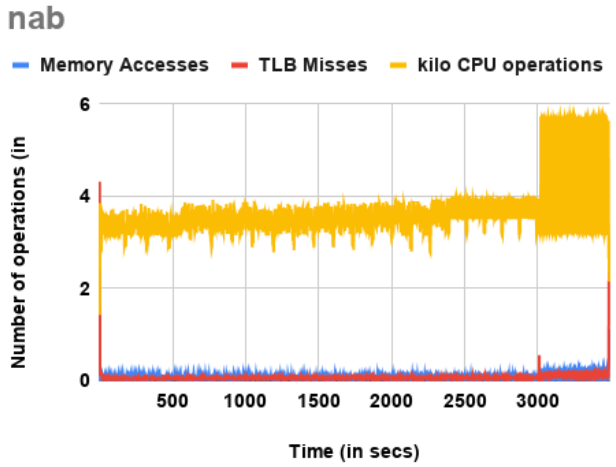


(b) Perl interpreter

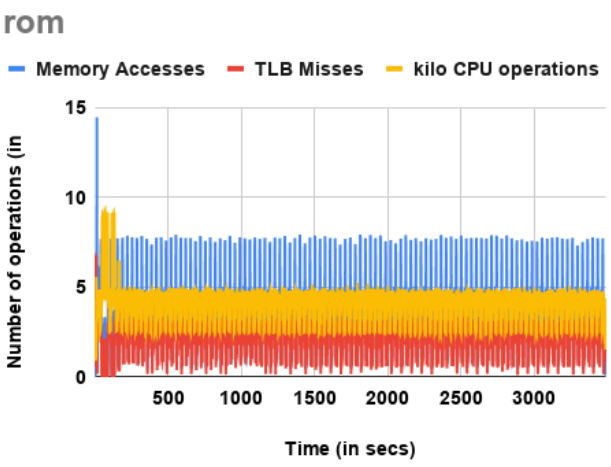


(c) Wide-scale ocean modeling

(d) Weather forecasting



(e) Molecular dynamics



(f) Regional ocean modeling

Figure 5: Other Benchmarks

We tried removing -fno-tree-loop-vectorize flag and adding -ftree-vectorize to see if any benchmark can be vectorized. Almost all the loops in the all benchmarks are not vectorized by gcc or clang. The number of vector instruction generated are negligible and it did not have any impact on performance. Further analysis is needed to see if we can vectorize workloads manually by rewriting the code using vector intrinsics.

Also all the benchmarks have IPC 3 and 10 Gflops peak performance. We ran all benchmarks in a system which has a peak of 64 Gflops, which can only be obtained with a significant amount of SIMD operations in the benchmarks. We tried running a hand optimized matrix multiplication code and it attained a performance of 47 Gflops, which is very high compared to any SPEC17 benchmark. Also these days many of the scientific workloads especially dense linear algebra libraries make heavy use of SIMD hardware support to achieve high performance. Most of the heavily used ML libraries or packages (like numpy and scipy for python) make use of BLAS like code to speedup linear algebra computations which almost achieve 95% of system peak. But there is no benchmark in SPEC17 that came close to this performance.

The effect of memory bus bandwidth is more visible when vector instructions are executed. This is because of the high volume of data being processed.

6 Related Work

There has been very limited work in characterizing workloads of SPEC17. We only found one paper [2], which characterizes applications of SPEC17 with respect to metrics such as instruction mix, execution performance, branch and cache behaviors. However, to our knowledge, the suite has not been analyzed for off-chip memory traffic (using the roofline model technique) by any other work.

References

[1] "SPEC CPU Benchmark Suites," https://www.spec.org/benchmarks.html

[2] Ankur Limaye and Tosiron Adegbija, *"A Workload Characterization of the SPEC CPU2017 Benchmark Suite", ISPASS April 2018*

[3] Samuel Williams, Andrew Waterman, and David Patterson, *"Roofline: An Insightful Visual Performance Model for Multicore Architectures", Communications of the ACM, April 2009*