RAPL energy profiling on AVX workloads

Gaetano Di Genio
Department of Computer Science
University of Salerno
Italy

***Abstract*—In some instances, processors may exhibit lower than expected speeds under heavy load, leading to an increase in system fan activity. This phenomenon is known as "throttling." While this issue may occur when running AVX workloads on Intel i9 processors, this work aims to present a general methodology to measure and compare results, using an Intel XEON system for analysis. The study provides a systematic approach to evaluate and compare the energy consumption and performance metrics of compute-intensive workloads, both with AVX and non-AVX scenarios. Finally, all results are plotted in order to compare them and to evaluate per presence or the absence of the throttling problem.**

I. Introduction

Motivation. In some cases the processor speed may appear below expected values when monitored under periods of heavy load, meaning that system fans may appear to increase as CPU load is applied.

This problem is known as "throttling".

A known case of this problem is when running AVX workloads on the Intel i9 processor, resulting in worser performance than running non AVX workloads or disabling it. [1]

This is because AVX workloads use wider registers and require higher power to switch than standard 64-bit registers. So for the processor to stay within current limits, the CPU automatically adjusts to a lower operating frequency for stability.

The purpose of this paper is to show a standard methodology, showing time and energy consumption evaluation of compute intensive workloads on an Intel XEON Gold 5218 system, both AVX and non-AVX workloads in order to see if the throttling problem discussed above also occurs on this specific system.

Related work. Many works exist that address the performance of AVX instructions, for example the paper "Early performance evaluation of AVX for HPC" [2] shows the impact of using AVX instructions in an HPC context, while the paper titled "Performance Evaluation of Matrix-Matrix Multiplications Using Intel's Advanced Vector Extensions (AVX)"[3] provides an assessment of matrix-matrix multiplications using Intel's AVX instructions.
In other words, both studies aim to explore the impact of AVX activation on performance and energy consumption in compute-intensive tasks.

II. Background:

Advanced Vector Extensions (AVX). It is a set of instruction set extensions introduced by Intel Corporation to enhance the performance of x86 processors in handling vectorized operations, built on top of the previous 128-bit SSE registers. AVX instructions enable the execution of multiple arithmetic operations simultaneously, significantly improving the efficiency of various computational workloads and better energy efficiency.

AVX workloads. AVX workloads refer to computational tasks that leverage AVX instructions for improved performance. AVX provides an expanded vector register width of 256 bits using YMM (YMM0-YMM15) registers, allowing for the simultaneous processing of multiple data elements in a single instruction. [4]

OpenMP. OpenMP (Open Multi-Processing) is an industry-standard API (Application Programming Interface) for parallel programming in shared-memory systems. With OpenMP, developers can add parallelism to their programs using compiler directives, pragmas, and runtime library routines. These directives and pragmas provide a high-level interface to express parallelism in the code, making it easier to exploit the available computational resources. It supports a wide range of parallelization techniques, including task-based parallelism, loop-level parallelism, and data parallelism.

Mandelbrodt set. The Mandelbrot set is a famous fractal in mathematics that exhibits intricate and self-repeating patterns. It is generated by iteratively applying a mathematical formula to complex numbers and determining if they remain within a specific boundary. The process involves calculating the behavior of each point in the complex plane, which results in the distinctive visual representation of the Mandelbrot set.

Prime95. Prime95[5] is a widely used benchmarking software that is primarily known for its stress testing capabilities. However, it also offers a range of benchmarking features that allow users to assess the performance of their system's CPU.

One of the key features of Prime95 as a benchmarking tool is its ability to execute highly computational workloads, specifically utilizing the Fast Fourier Transform (FFT) algorithm and also Trial Factoring. By performing complex calculations, Prime95 can stress the CPU and evaluate its processing power and efficiency. During benchmarking, Prime95 generates detailed reports that include information about the system's performance metrics, such as the computation time, number of iterations, and any errors encountered during the benchmark run. These reports can help users evaluate the stability and reliability of their system under heavy computational loads. Furthermore, Prime95 offers options to customize the benchmarking process, allowing users to adjust parameters like

the number of threads or the size of the FFTs. This flexibility enables users to tailor the benchmark to their specific hardware configuration and evaluate the performance under different scenarios.

perf. Perf is a powerful tool in Linux that enables energy measurement and analysis at different levels, including package-level and RAM-level energy measurement [6] **(core-level measurement is disabled on the machine used for this paper)**. With its capabilities, perf allows for a comprehensive understanding of power usage in the system.

For package-level energy measurement, perf utilizes the RAPL (Running Average Power Limit) interface. RAPL is an interface provided by some Intel and AMD processors that allows for monitoring energy consumption on various components of the system, including the CPU package.

By leveraging RAPL, perf can access energy consumption information through the interfaces provided by the Linux kernel. This enables perf to provide detailed measurements of energy consumption for the processors and RAM.

III. Proposed Method

In this section the proposed method is presented, which has been developed to address the objectives outlined in this paper. The key tools, algorithms, and techniques used are described, providing a clear and concise overview of the methodology employed.

Proposed method description. The proposed methodology revolves around leveraging three specific algorithms (Mandelbrot Set, Fast Fourier Transform, and Trial Factoring) as case studies to analyze their energy consumption characteristics.
The algorithms were executed on three different configurations: 1 thread, 8 threads, and 16 threads.
The energy consumption of each algorithm was measured using the Perf tool. The measurements were conducted with AVX enabled in one set of runs and disabled in another set of runs.
A comparison of the measurements aimed to identify potential issues related to throttling, wherein the CPU may reduce performance due to excessive power consumption.

Mandelbrodt set algorithm. For the implementation of Mandelbrodt set algorithm, OpenMP was used for parallelization, while AVX intrinsics were deliberately not employed. Instead, reliance was placed on auto-vectorization by the compiler to leverage vector instructions.
The full implementation code can be seen at https://github.com/gaetanodigenio/avxi9throttling.

Fig.1 shows the relevant part of the code:

Fast Fourier Transform and Trial Factoring algorithms. Those two algorithms are sourced from the Prime95 software. Prime95's implementation of the FFT algorithm provides an efficient approach for computing the discrete Fourier transform.
The Trial Factoring algorithm, also adopted from Prime95, enables an examination of energy consumption during large number factorization.

```
26      float t_init = omp_get_wtime();
27
28      #pragma omp parallel shared(pixels)
29      {
30          int iY;
31          #pragma omp for schedule(dynamic)
32          for(iY=0; iY<ImageHeight; iY++) {
33              double Cy=CyMin + iY*PixelHeight;
34              if (fabs(Cy)< PixelHeight/2) {
35                  Cy=0.0; // Main antenna
36              }
37              int iX;
38              for(iX=0; iX<ImageWidth; iX++) {
39                  double Cx=CxMin + iX*PixelWidth;
40                  double Zx=0.0;
41                  double Zy=0.0;
42                  double Zx2=Zx*Zx;
43                  double Zy2=Zy*Zy;
44                  /* */
45                  int Iteration;
46                  const int IterationMax=50;
47                  const double Bailout=2; // bail-out value
48                  const double Circle_Radius=Bailout*Bailout; // circle radius
49
50                  for (Iteration=0; Iteration<IterationMax && ((Zx2+Zy2)<Circle_Radius); Iteration++) { //
51                      Zy=2*Zx*Zy + Cy;
52                      Zx=Zx2-Zy2 + Cx;
53                      Zx2=Zx*Zx;
54                      Zy2=Zy*Zy;
55                  };
56
57                  if (Iteration==IterationMax) {
58                      //  interior of Mandelbrot set = black
59                      pixels[iY*ImageWidth + iX][0] = 0;
60                      pixels[iY*ImageWidth + iX][1] = 0;
61                      pixels[iY*ImageWidth + iX][2] = 0;
62                  }
63                  //
64                  else {
65              Gradient((double)((Iteration-log2(log2(sqrt(Zx2+Zy2))))/IterationMax,pixels[iY*ImageWidth + iX]);
66                  }
67              }
68          }
69      }
70
71      printf("Total time: %fs\n", omp_get_wtime() - t_init);
```

Fig. 1. Parallel algorithm implementation using OpenMP pragmas.

IV. Experimental Results

Experimental setup. The machine used to run the tests has an Intel Xeon Gold 5218 with 2 NUMA nodes each one with 16 cores and 2 threads per core (HyperThreading) (fig.2).
The operating system is Ubuntu 20.04.4 LTS.

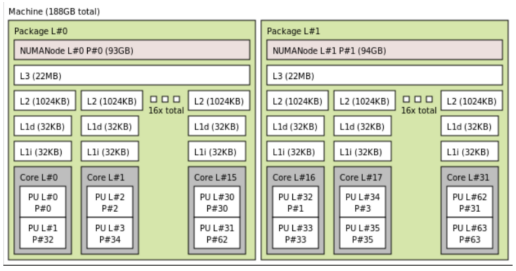

Fig. 2. Topology of the machine generated with the command *lstopo*.

Mandelbrodt set algorithm tests. The Mandelbrodt set algorithm is written in C and a GCC is used as a compiler.
The matrix input size is 40.000 x 40.000.
Tests are executed 10 times using the following commands to compile and run (fig. 3, fig. 4, fig. 5, fig. 6):

```
gcc-11 -mno-avx -mno-avx2 -mno-avx512f -O3 -fopenmp -o mandelbrodt_noavx mandelbrodt_set.c -lm
```

Fig. 3. Command to compile Mandelbrodt set program disabling AVX.

```
gcc-11 -mavx -O3 -fopenmp -o mandelbrodt_avx mandelbrodt_set.c -lm
```

Fig. 4. Command to compile Mandelbrodt set program enabling AVX.

```
'OMP_PLACES="cores(numcores)" OMP_PLACES="sockets(1)" OMP_NUM_THREADS=numcores ./programName
```

Fig. 5. Command to run executable specifying the number of cores, one socket is used, then the number of threads is equal to the number of the cores meaning that for each core only one thread is running (HyperThreading disabled).

```
OMP_PLACES="cores(numcores)" OMP_PLACES="sockets(1)" OMP_NUM_THREADS=numcores perf stat -r "numIterations" -e power/energy-pkg/
OMP_PLACES="cores(numcores)" OMP_PLACES="sockets(1)" OMP_NUM_THREADS=numcores perf stat -r "numIterations" -e power/energy-ram/
```

Fig. 6. Command to run the program using Perf, the top command measures energy package consumption, the command on the bottom measures the energy ram consumption.

Prime95 FFT and trial factoring algorithms tests. The idea is to execute the benchmark using 1, 8 and 16 cores (on the same socket), with only 1 thread per core (Hyperthreading disabled). The Prime95 tool is executed using the following command in fig. 7, fig. 8 and fig. 9:

```
numactl --physcpubind=0 --membind=0 perf stat -r 1 -e power/energy-pkg/ ./mprime -m
```

Fig. 7. Command to execute Prime95 using 1 cores of 1 NUMA node.

```
numactl --physcpubind=0-7 --membind=0 perf stat -r 1 -e power/energy-pkg/ ./mprime -m
```

Fig. 8. Command to execute Prime95 using 8 cores of 1 NUMA node.

```
numactl --physcpubind=0-15 --membind=0 perf stat -r 1 -e power/energy-pkg/ ./mprime -m
```

Fig. 9. Command to execute Prime95 using 16 cores of 1 NUMA node.

Once the executable is executed, the main menu (fig. 10) is printed, allowing to choose the algorithm and the number of threads to be executed (fig. 11):

```
        1.  Test/Primenet
        2.  Test/Workers
        3.  Test/Status
        4.  Test/Continue
        5.  Test/Exit
        6.  Advanced/Test
        7.  Advanced/Time
        8.  Advanced/P-1
        9.  Advanced/ECM
       10.  Advanced/Manual Communication
       11.  Advanced/Unreserve Exponent
       12.  Advanced/Quit Gimps
       13.  Options/CPU
       14.  Options/Resource Limits
       15.  Options/Preferences
       16.  Options/Torture Test
       17.  Options/Benchmark
       18.  Help/About
       19.  Help/About PrimeNet Server
Your choice:
```

Fig. 10. Number 17 (Benchmark) is chosen.

```
Your choice: 17

Benchmark type (0 = Throughput, 1 = FFT timings, 2 = Trial factoring) (0): 1

FFTs to benchmark
Minimum FFT size (in K) (7680): 2048
Maximum FFT size (in K) (7680): 2048
Benchmark with round-off checking enabled (N):
Benchmark all-complex FFTs (for LLR,PFGW,PRP users) (N):

CPU cores to benchmark
Number of CPU cores (comma separated list of ranges) (32): 1
Benchmark hyperthreading (N): n

Accept the answers above? (Y): y
```

Fig. 11. For the FFT, input size and number of cores must be chosen.

In the local.txt file, AVX can be enabled or disabled, setting respectively 1 or 0 to the variables shown in fig. 12:

```
CpuSupportsAVX=1
CpuSupportsAVX2=1
CpuSupportsAVX512F=1
```

Fig. 12. EVX enabled. To disable, 0 must be set to these variables.

In order to execute the trial factoring algorithm, 17 must be selected from the main menu, then 2 is selected (fig. 13):

```
Benchmark type (0 = Throughput, 1 = FFT timings, 2 = Trial factoring) (0): 2

CPU cores to benchmark
Number of CPU cores (comma separated list of ranges) (32): 1
Benchmark hyperthreading (N): n

Accept the answers above? (Y): y
```

Fig. 13. For the trial factoring, only number of cores can be specified.

Results. Here are reported the final plots showing the experiment results.
Important to clarify that for the parallel speedup, the base case is a single parallel thread, not the best serial execution.

Mandelbrodt set plots. For the Mandelbrodt set execution, average time, energy-pkg and energy-ram measurements are shown, all using 1, 8, 16 cores and AVX (enabled and disabled). All the measurements taken show consistent results, with no extreme variations (low variance). Also the absolute and relative speedup can be visualized on the plot (fig. 14).

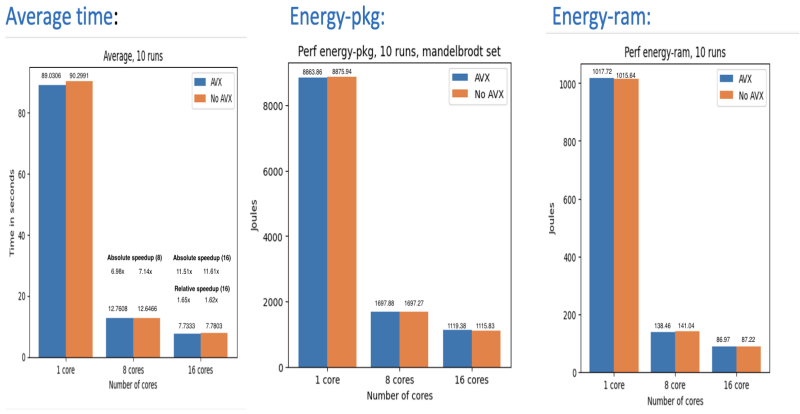

Fig. 14. Plot showing comparison between average time (AVX and no AVX) with absolute and relative speedups, Energy-pkg consumption in joule and Energy-ram consumption in Joule. All tests are executed 10 times.

Average time and median time are very similar (fig. 15):

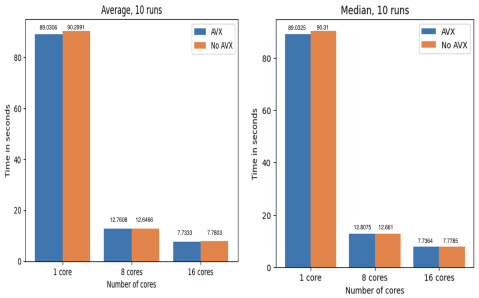

Fig. 15. Plot showing comparison between average time and median time.

Prime95 FFT plots. For the FFT algorithm, three inputs are tested: 2048k, 5120k and 7680k.
For each input, time execution in seconds, energy-pkg and energy-ram measurement are compared, both with AVX enabled (blue bars) and AVX disabled (orange bars), all using 1, 8 and 16 cores (1 thread per each core, no Hyperthreading).
Starting with **2048k input size** in fig.16:

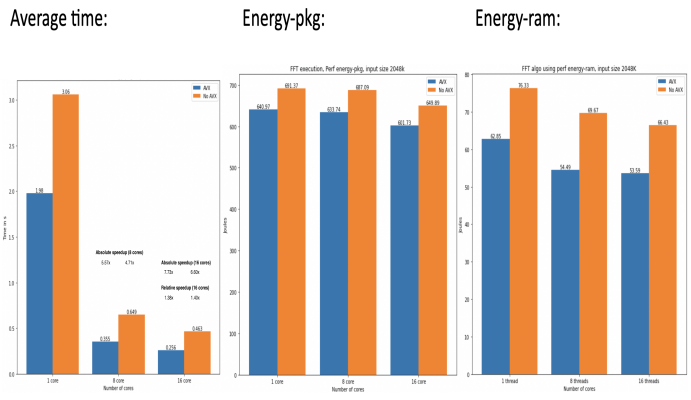

Fig. 16. Time, energy-pkg and energy-ram for 2048k input size.

Then **5120k input size** (fig. 17):

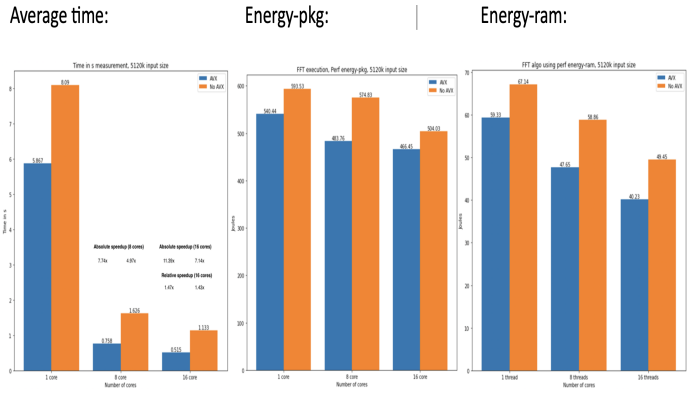

Fig. 17. Time, energy-pkg and energy-ram for 5120k input size.

Finally with **7680k input size** (fig. 18):

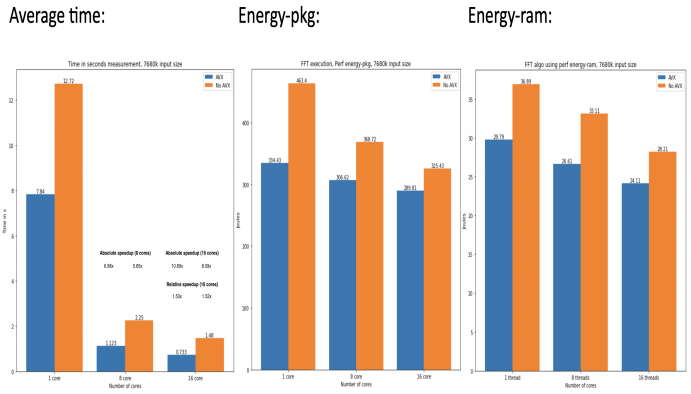

Fig. 18. Time, energy-pkg and energy-ram for 7680k input size.

Prime95 trial factoring plots. As shown in the previous paragraph, for the Trial Factoring algorithm execution only the number of cores and wheter to enable or not the Hyperthreading can be setted, no control over the inputs.
The algorithm tests incremental input size, from 61 to 77 bit length factors, multiple iterations for each input.
In the following plots, the time in seconds, the pkg and ram energy consumption of the entire execution (calculating the incremental input size above) is compared.
No possibility here to test individual input sizes (as for the FFT algorithm).
The fig. 19 shows the plots:

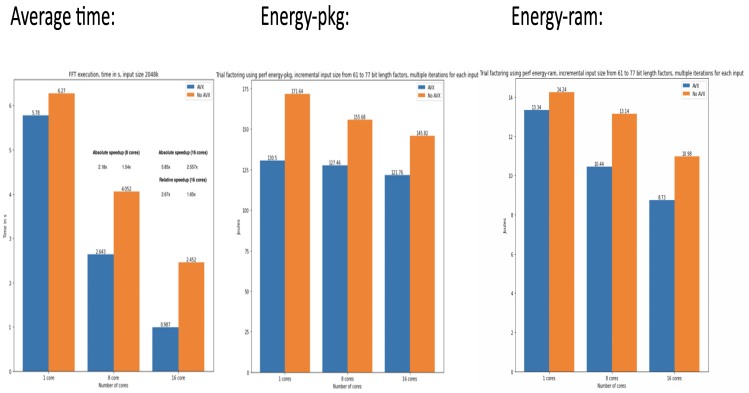

Fig. 19. Time, energy-pkg and energy-ram for incremental trial factoring inputs.

Comments.

- Regarding the plot shown in **fig.14**, the reason for the lack of scalability and the absence of noticeable differences in performance between the AVX-enabled and AVX-disabled executions can be attributed to the code analysis conducted using tools such as Godbolt. It revealed that the generated code predominantly consists of scalar instructions, with only a few vectorized instructions present, resulting in negligible impact.
  Further insights can be obtained by referring to the analysis available at the following link: https://godbolt.org/z/d8aqna4nY, where in the first panel there is the Mandelbrodt set code, in the second panel the execution of the program disabling AVX and in the third panel the execution of the program with AVX enabled.
- In **Figures 16, 17, 18, and 19**, it is evident that enabling AVX instructions leads to shorter execution times and lower energy consumption (both package and RAM) compared to their corresponding executions with AVX disabled. This clear difference demonstrates that the benchmarks are optimized for AVX usage, indicating effective code vectorization. Conversely, executions with AVX disabled consistently exhibit inferior performance, both in terms of execution time and energy consumption. It is worth noting that this contrast is particularly marked compared to the execution of the Mandelbrot set algorithm, which showed fewer significant differences

between AVX-enabled and AVX-disabled runs. The observed substantial disparities further emphasize that the code is well-adapted and efficiently tuned for execution with AVX active.

V. Conclusions

In conclusion, the experimental results unequivocally demonstrate that enabling AVX instructions leads to substantial improvements in execution times and energy consumption (both package and RAM) compared to disabling AVX, specifically when the considered algorithm leveraged and benefited from vectorized instructions. For instance, the Mandelbrot set algorithm demonstrated less pronounced differences between AVX-enabled and AVX-disabled executions, indicating that its code was less optimized for exploiting AVX capabilities.
For the FFT and Trial Factoring algorithms, the evident optimization of code vectorization for AVX execution suggests that throttling might have a reduced impact or be totally absent during AVX-optimized runs, contributing to the observed superior performance and energy efficiency.

The findings emphasize the significance of proper AVX optimization for maximizing computational efficiency and energy savings, underscoring the potential benefits of AVX utilization in the context of relevant compute-intensive tasks. In the end, the methodology developed and employed in this study proves to be versatile and applicable to any processor.

References

[1] i9 throttling, *Intel i9 Processor Throttling Under AVX (Advanced Vector eXtensions)* , URL: https://www.dell.com/support/kbdoc/it-it/000184687/intel-i9-processor-throttling-under-avx-advanced-vector-extensions.

[2] AVX instructions, *Early performance evaluation of AVX for HPC*, URL: https://www.sciencedirect.com/science/article/pii/S1877050911001050.

[3] Matrix multiplication, *Performance Evaluation of Matrix-Matrix Multiplications Using Intel's Advanced Vector Extensions (AVX)*, URL: https://www.sciencedirect.com/science/article/abs/pii/S0141933116302502.

[4] Programming with AVX, *On the energy consumption of Load/Store AVXinstructions*, URL: https://www.researchgate.net/publication/327893479_On_the_energy_consumption_of_LoadStore_AVX_instructions.

[5] Prime95, *Prime95 benchmarking software website*. URL: https://www.mersenne.org/.

[6] RAPL domains, *RUNNING AVERAGE POWER LIMIT – RAPL*, URL: https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl.