TDT 4260: Computer Architecture

Climbing Mount Blanc – Optimization Log

Submitted By: Amna Waris

Zahra Jenab Mahabadi

1) Handed out code:

- Virtual Machine

a) Execution time/CPU usage:

User time 53.36 s
System time 0.35 s
Total time elapsed: 54.08 s
CPU: 99%

b) Top 5 code portions with longest execution time:

Children	Self	Command	Shared Object	Symbol
99.91%	0.00%	image_processin	libc-2.31.so	[.]__libc_start_main
99.91%	0.00%	image_processin	image_processing_c	[.] main
99.07%	98.95%	image_processin	image_processing_c	[.]blurIteration
0.61%	0.19%	image_processin	image_processing_c	[.]convertToAccurate
0.45%	0.00%	image_processin	[kernel.kallsyms]	[k]page_fault

- Climbing Mount Blanc Website

Program was timed out after 90 seconds!

2) Cache and access patterns:

As can be seen above the blurIteration function takes the most time. In version 2 of the code, 'for' loops in the function were interchanged to compute values in the horizontal direction first and then the vertical direction. This would be beneficial as the cache usually fetches data that is placed horizontally together first.

- Virtual Machine

a) Execution time/CPU usage:

User time 44.51 s
System time 0.30 s
Total time elapsed: 45.17 s
CPU: 99%

b) Top 5 code portions with longest execution time:

Children	**Self**	**Command**	**Shared Object**	**Symbol**
99.90%	0.00%	image_processin	libc-2.31.so	[.]__libc_start_main
99.90%	0.00%	image_processin	image_processing_c	[.] main
98.81%	98.67%	image_processin	image_processing_c	[.]blurIteration
0.80%	0.22%	image_processin	image_processing_c	[.]convertToAccurate
0.60%	0.00%	image_processin	[kernel.kallsyms]	[k]page_fault

- Climbing Mount Blanc Website

  Program was timed out after 90 seconds!

3) Useless code and branches:

It was noticed that blurIteration (function that takes the longest time to run), was being called several times in main to run for each color channel separately. This is redundant and all the calculations were just done in one go in version 3. This will also remove the for loop in the main function and if/else statements (branches) in the blurIteration function.

- Virtual Machine

a) Execution time/CPU usage:

User time 11.95 s
System time 0.32 s
Total time elapsed: 12.55 s
CPU: 97%

b) Top 5 code portions with longest execution time:

Children	**Self**	**Command**	**Shared Object**	**Symbol**
99.68%	0.00%	image_processin	libc-2.31.so	[.]__libc_start_main
99.68%	0.00%	image_processin	image_processing_c	[.] main
96.44%	98.17%	image_processin	image_processing_c	[.]blurIteration
2.21%	0.82%	image_processin	image_processing_c	[.]convertToAccurate
1.47%	0.00%	image_processin	[kernel.kallsyms]	[k]page_fault

- Climbing Mount Blanc Website

  Time: 32.60 s
  Energy: 105.00 J
  EDP:3423.07 Js

4) Changing algorithm (using separable box filter):

Box blur is a separable filter. Rather than applying it in both directions simultaneously, it can be applied in one direction direct and can be applied to the intermediate image again in the other direction to get the final image. This was done in version 4. This lowers the complexity of the algorithm from O(Nr^2) to O(NR), where N is the number of pixels and r is the radius of filter.

- Virtual Machine

  a) Execution time/CPU usage:

  User time 2.15 s
  System time 0.39 s
  Total time elapsed: 2.74 s
  CPU: 92%

  b) Top 5 code portions with longest execution time:

Children	Self	Command	Shared Object	Symbol
98.21%	0.00%	image_processin	libc-2.31.so	[.]__libc_start_main
98.21%	0.00%	image_processin	image_processing_c	[.] main
79.03%	78.09%	image_processin	image_processing_c	[.]blurIteration
14.27%	3.63%	image_processin	image_processing_c	[.]convertToAccurate
11.44%	0.04%	image_processin	[kernel.kallsyms]	[k]page_fault

- Climbing Mount Blanc Website

  Time: 10.88 s
  Energy: 32.28 J
  EDP:351.21 Js

5) Changing algorithm (accumulation-using previously calculated 'sum'):

The previous algorithm calculated the sum each time the filter slid to a new position. In version 5, the algorithm was changed to use the previously generated sum value. When the window is sliding in, the sum was calculated as in the previous version. When the window is sliding out, the value that is moved out is subtracted from the sum. In between the extreme cases, the value to the left of the window is subtracted and to the right of the window is added. This would take the complexity down to O(N) from O(Nr).

When the algorithm was applied in both directions the resulting time was higher than when it was applied only in the x direction. This is because to make it work in the y direction, the outer for loops flipped in version 2 for cache and access patterns must be changed back, making the cache usage inefficient.

The version 5 has the algorithm changed only in the x direction and the same was kept when adding more optimization changes.

- Virtual Machine

a) Execution time/CPU usage:

User time 1.45 s
System time 0.38 s
Total time elapsed: 2.07 s
CPU: 88%

b) Top 5 code portions with longest execution time:

Children	Self	Command	Shared Object	Symbol
97.54%	0.00%	image_processin	libc-2.31.so	[.]__libc_start_main
97.54%	0.00%	image_processin	image_processing_c	[.] main
71.28%	71.16%	image_processin	image_processing_c	[.]blurIteration
18.99%	5.98%	image_processin	image_processing_c	[.]convertToAccurate
14.13%	0.00%	image_processin	[kernel.kallsyms]	[k]page_fault

- Climbing Mount Blanc Website

Time: 8.50 s
Energy: 23.58 J
EDP: 200.40 Js

6) Parallelization:

Parallelization can be used to run the code on different cores to make the execution time faster. The parallelization was done using OpenMP. Running the code on VM gave almost the same speed up because it has only one core, but a significant difference can be noted when it ran on CMB website.

Two approaches were taken to optimize the code through parallelization. In the first approach the parallelization was added to the for loops in the blurIteration function. This resulted in significant speedup on CMB website. In the second approach, parallelization was added to the main function to run each case (tiny, small, medium, large) on separate cores, this made the execution time worst. It might be because the images were declared as shared variables so that the threads can interact as the images need to be subtracted from one another at the end.

So, only the first approach was selected to be version 6.

- <u>Virtual Machine</u>

  User time 1.47 s
  System time 0.30 s
  Total time elapsed: 1.96 s
  CPU: 90%

- <u>Climbing Mount Blanc Website</u>

  Time: 3.84 s
  Energy: 17.58 J
  EDP: 67.52 Js

7) Vectorization:

Vectorization can be used to perform computations at once on different data. In version 7, vectorization was implemented for the calculation of sum and value for the three colors channels in the blurIteration function. We introduced a dummy for the struct that contains colors data to be able to use a vector of four. The results that we got after implementation had more user and system time but has been included in the report for comparison.

- <u>Virtual Machine</u>

  User time 3.12 s
  System time 0.43 s
  Total time elapsed: 3.75 s
  CPU: 94%

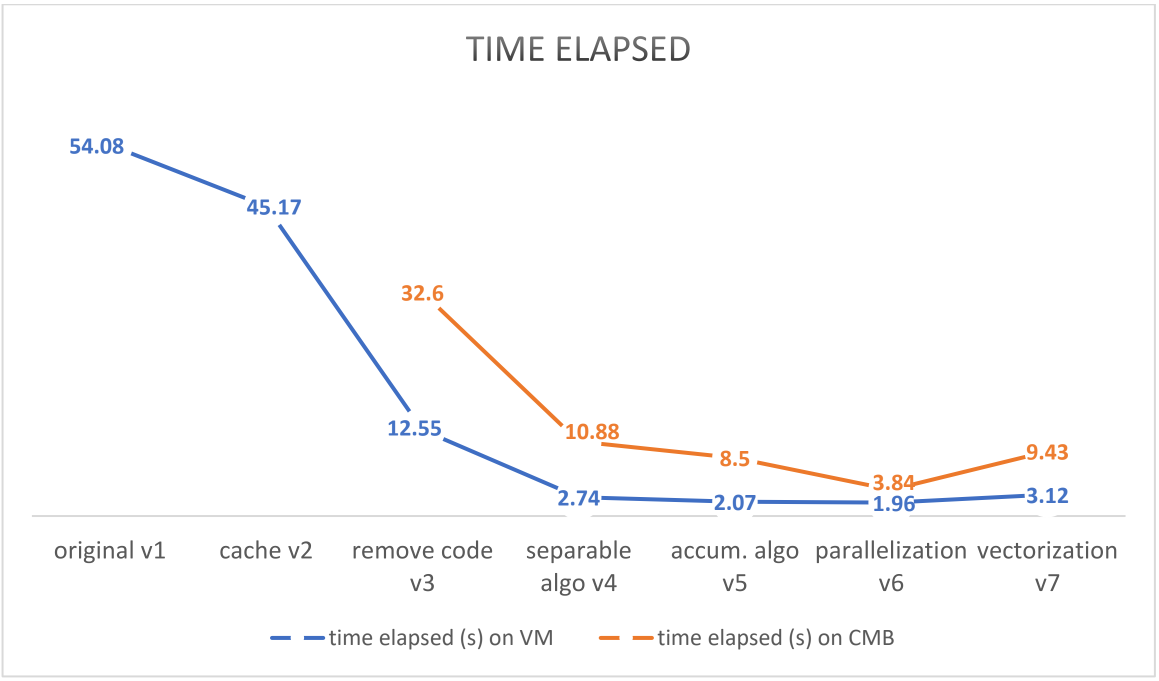
- Climbing Mount Blanc Website

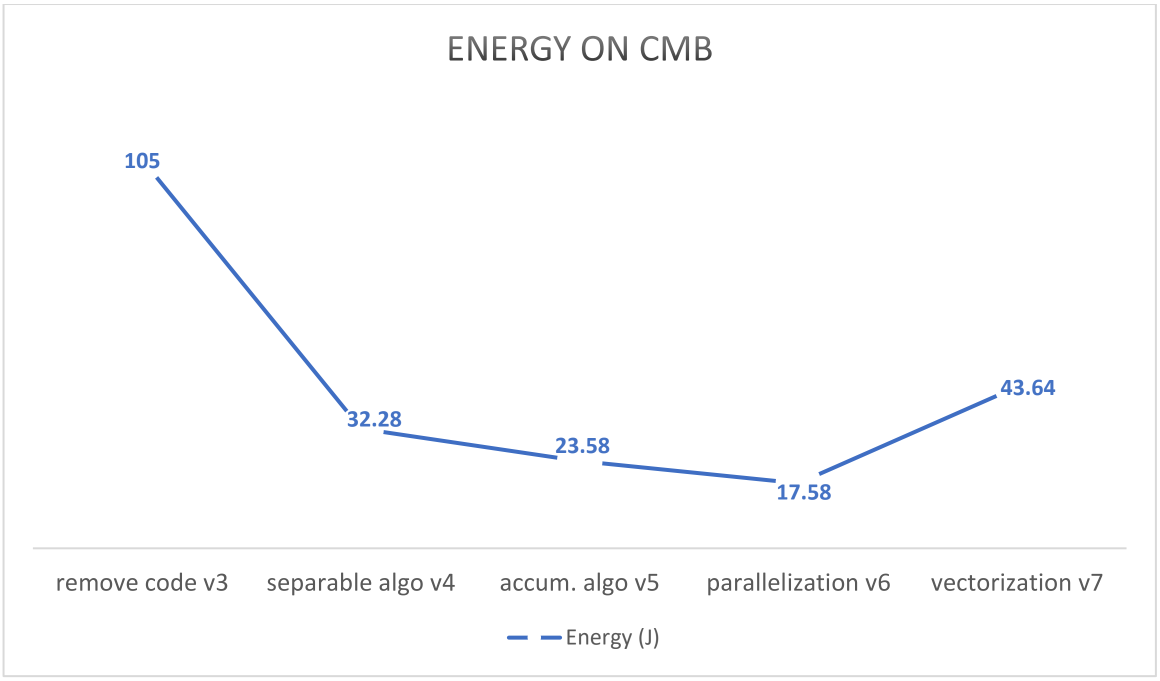
  Time: 9.43 s
  Energy: 43.64 J
  EDP: 411.50 Js

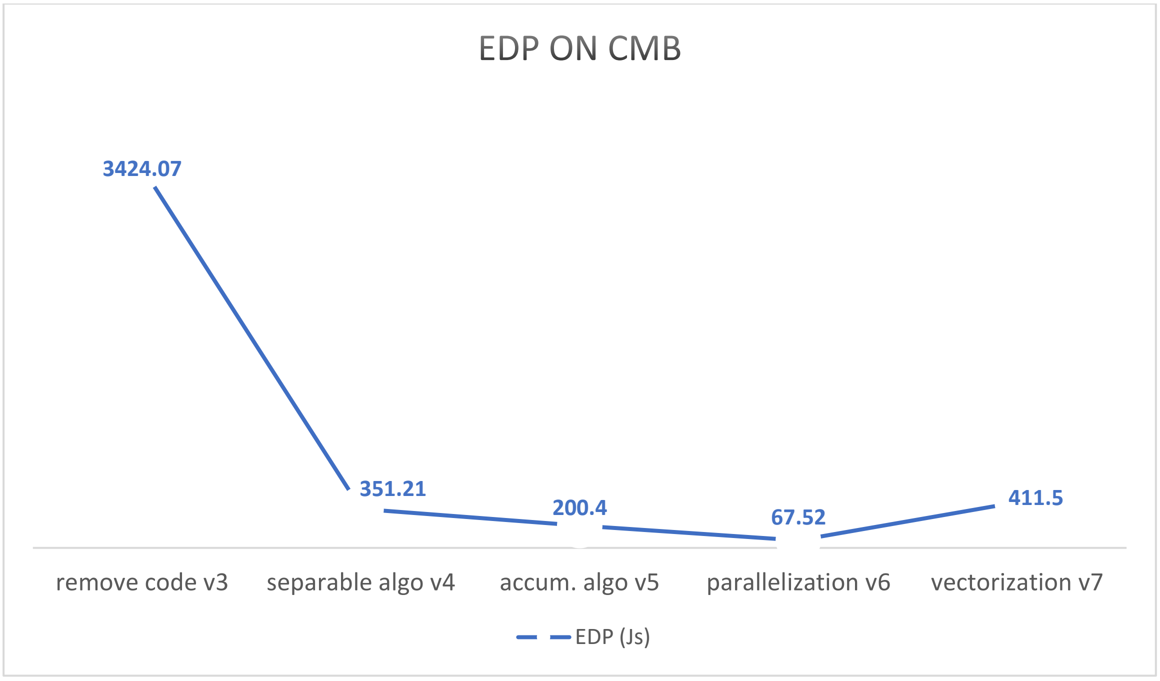
SUMMARY

The graphs below do not consider version 1 and version 2 for CMB because it was not optimized enough to run on CMB. Version 7 (vectorization) performs worse than version 6 but has been included in the report for comparison.

After applying different optimization methods, the best execution time of 1.96s on the VM and 3.84s on CMB was achieved. With a best energy of 17.58 J and EDP of 67.52 Js.




ENERGY ON CMB
105
32.28
23.58
17.58
43.64
remove code v3
separable algo v4
accum. algo v5
parallelization v6
vectorization v7
Energy (J)


EDP ON CMB
3424.07
351.21
200.4
67.52
411.5
remove code v3
separable algo v4
accum. algo v5
parallelization v6
vectorization v7
EDP (Js)