HPC Tools

Final Assignment

Haytam EL MERABETI
HPC Master student
elmerabeti@cy-tech.fr

Faculty of Computer Science

December 23, 2022

HPC Tools : Final Assignment

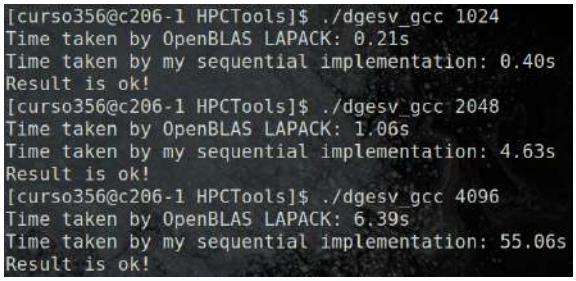
In the conclusion of my report in task 2, I spoke about how the O3 optimization flag was the most appropriate one for my code, paired with the profile guided optimozation and some code modifications compared to task 1. In this final task, I will try to use different tools to analyze my code on different aspects (memory, parallelization, vectorization, cache, etc), while always mesuring performance using the forementionned compiling configuration (O3 flag with PGO).

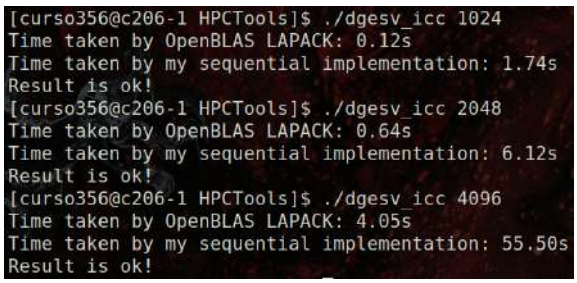
Memory :

Using Valgrind's Memcheck tool, I was able to detect some memory issues which led me to the modifications below :

- Freeing the space dynamically allocated to the pointer augMatrix.
  - This led me to realise that I didn't really need a dynamically allocated pointer, given that I already know the size that my array needs, so I changed augMatrix's type to a static double array of size n*2*n.
  - In the same time, I choose to use a static double array"ratio" instead of only a variable, in ordre to avoid concurrency.
- Adding several memory freeing functions to the main part of the code, although it is outside the scope of my implementation of dgesv.

These changes resulted in the following performance improvement (O3 + PGO):





The final outputs of Memcheck for gcc and icc without compiler opts and with param 256 (to make it kind of faster…) are in the file valgrind_output_gcc.txt & valgrind_output_icc.txt .

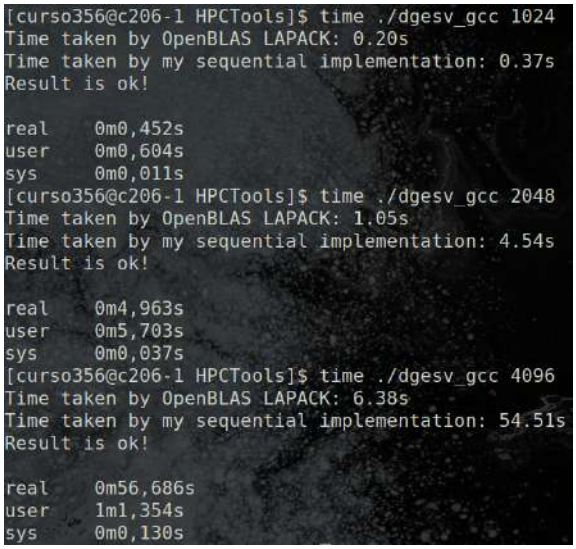
The results show no “definitely lost” or “indirectly lost” leaks for both gcc and icc, and the remaining leaks are not linked to my implementation of dgesv.

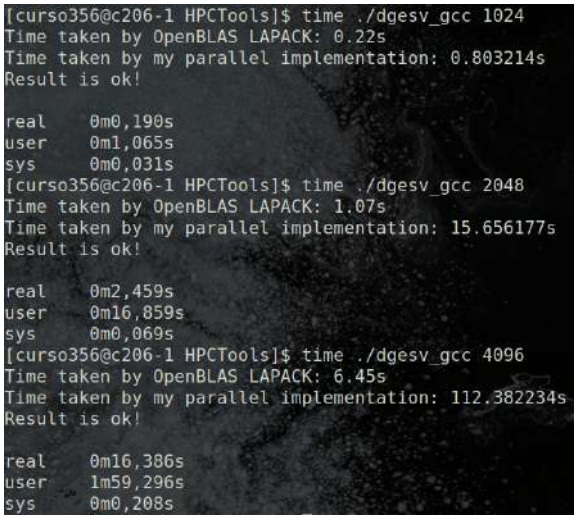
Parallelization :

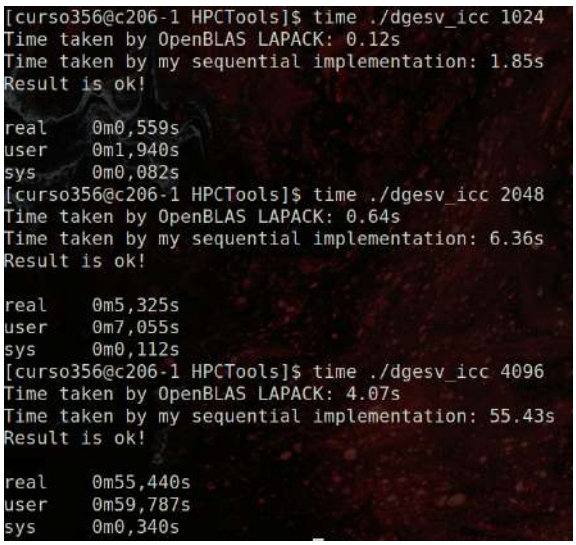
As parallelization wasn't the main goal of this final task, I only added two basic "#pragma omp for" directives in the following code parts :

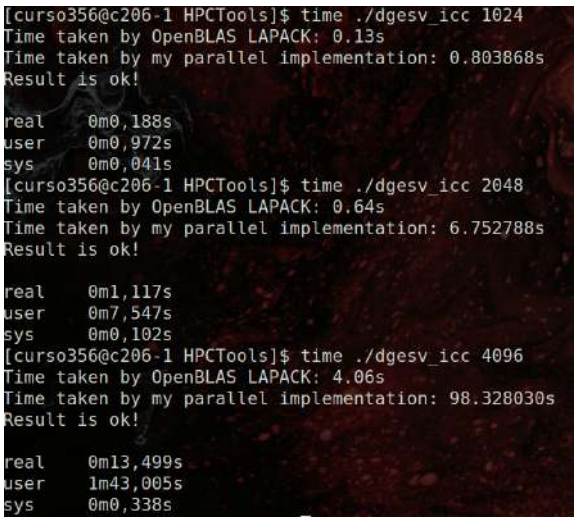
- The gauss jordan elimination.
- The computation of the unique solution b.

This led to improving the performance as follows :







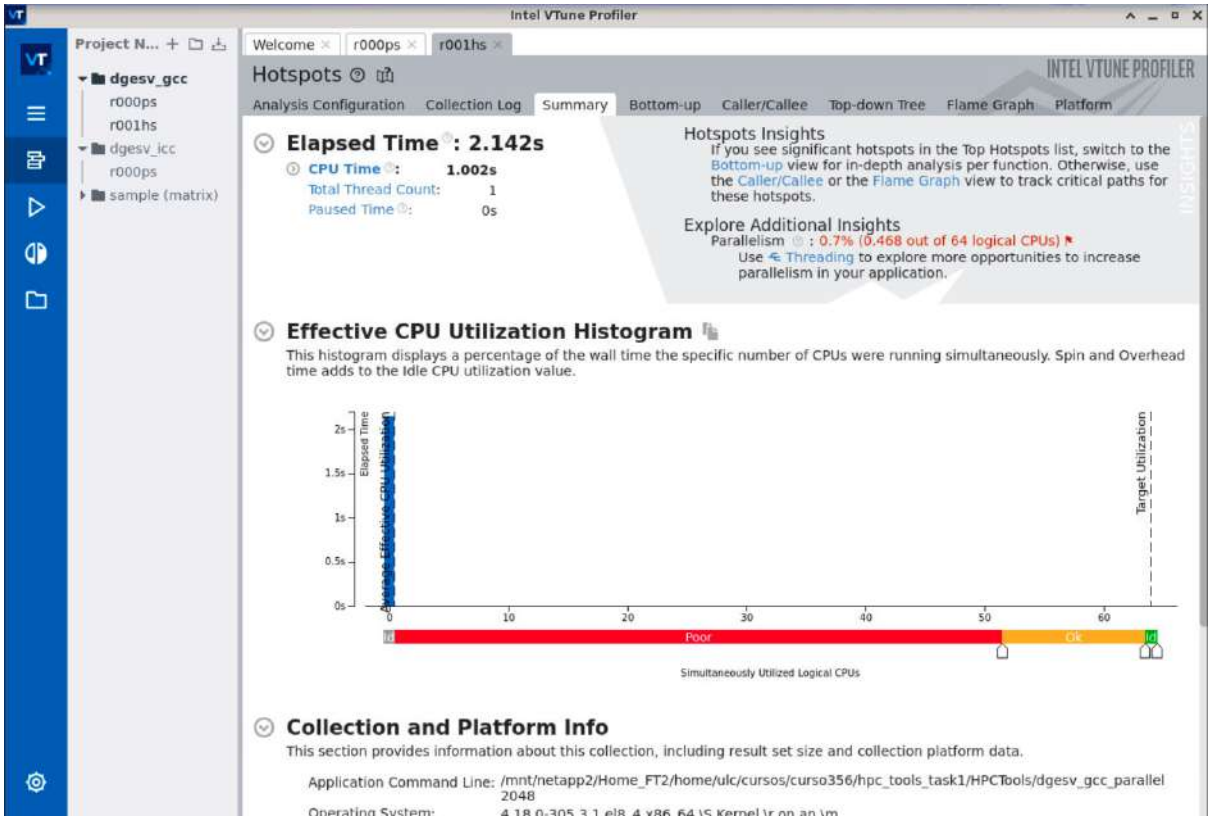


The following speedup values were calculated using 8 cores, and based on the “real” time of the function time, in order to have a common function for sequential and parallel code.

Speedups	gcc	icc

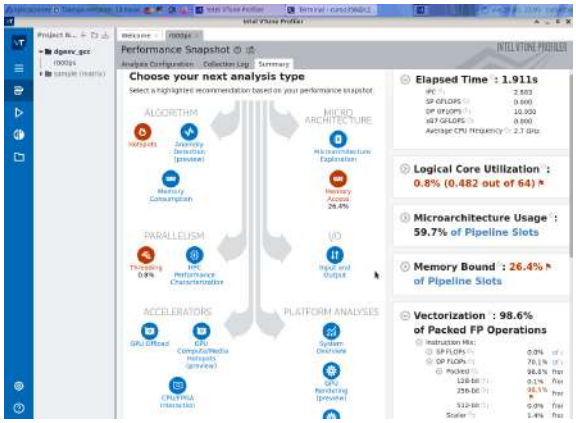
Speedups	gcc	icc
Small	2.38	2,97
Medium	2.02	4,77
Large	3.46	4,11

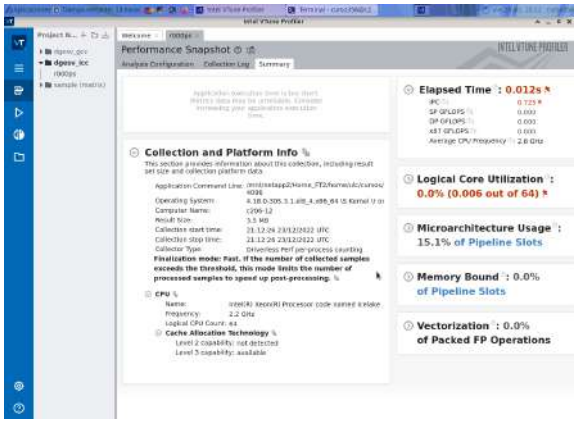
The output of Intel Vtune Amplifier is as follows :



Vectorization :

In this part, I used Intel Vtune Amplifier utility to further analyze my code, especially to know the status of vectorization in it.





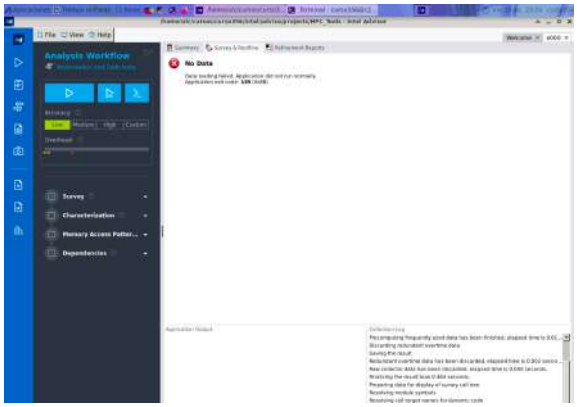
The results, as shown in the above image, present a score of nearly 99% in vectorization when compiling with gcc (icc version apparently had a short execution time, even when i used 8192 as param… so I couldn’t get much info for this version).

I would consider that the good vectorization of gcc version is thanks to :

- The O3 flag.
- The use of the static arrays.

For some unknown reasons to me, I wasn’t able to use Intel Advisor on my executable.

I had this screen :



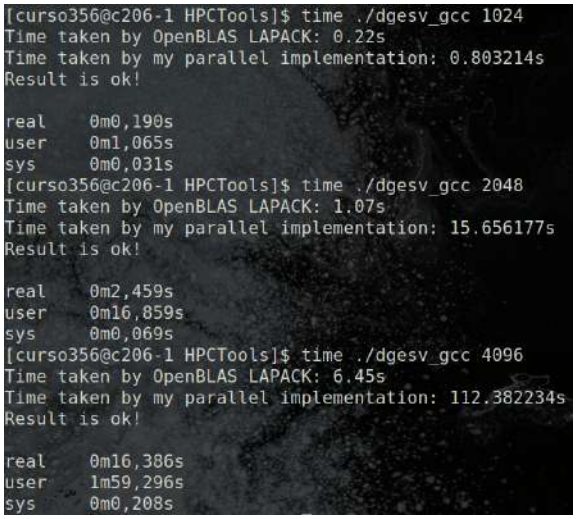
Cache :

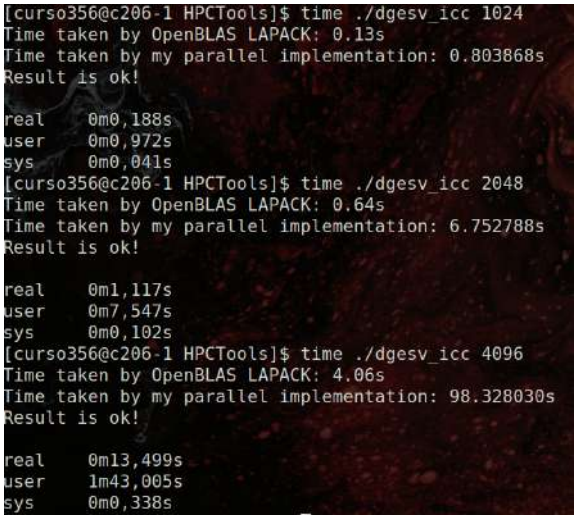
Using Valgrind's Cachegrind, I analysed my code to check whether there is any cache misses on the different levels. Using gcc, the output of Cachegrind is in the file Cachegrind_output.txt . I couldn’t find any special improvement in my implementation.

Final performances :

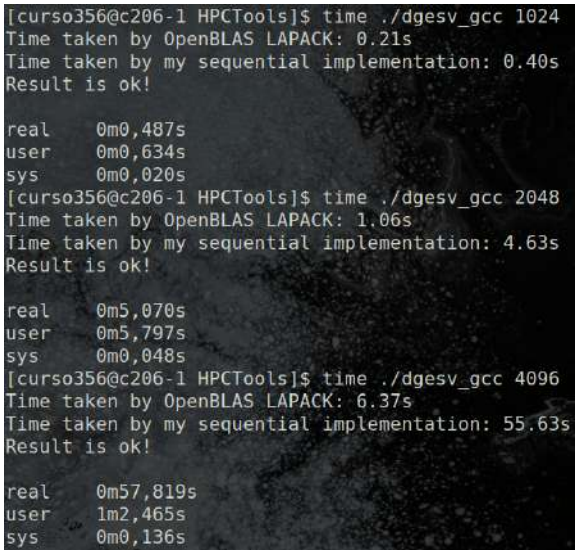
Several tools and optimizations later, my program performs as follows (the given executables are the most optimized ones) :

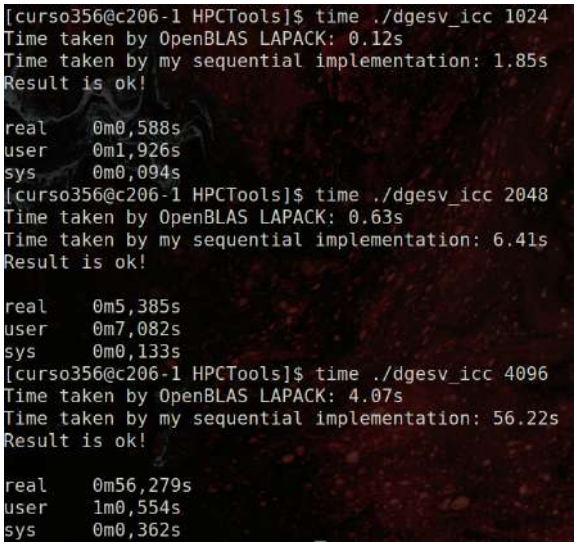
- O3 with PGO and Parallelization :





- O3 with PGO without Parallelization :





While I am sure I could further improve my code and optimize it, especially through a better use of the remaining tools that we discovered through this course, I am satisfied by what I was able to accomplish so far.