EuroCC@Turkey

https://eurocc.truba.gov.tr/

This document is prepared by EuroCC@Turkey for EuroCC under GA NO 951732

CASE STUDY REPORT

Code Modernization for Glass Industry

NCC Partner	*Sabancı University*
Company*	*www.sisecam.com.tr Şişecam* *Adnan Karadag -ADKARADAG@sisecam.com*
Expert	*Bekir Bediz, bekir.bediz@sabanciuniv.edu* *Kamer Kaya, kamer.kaya@sabanciuniv.edu*
Start & End Date	*14.03.2022 - 15.09.2022*

***Company** accepts that the Case Study Report is shared with the EuroCC Project and the community through the EuroCC@Turkey awareness creation activities and platforms.

1. Problem Identification

ŞİŞECAM fabricates glass products using custom-made ovens. They use a simulation framework that traces the glass particles inside the oven. In a nutshell, the longer the particles stay inside the oven, the better the quality of the product. To achieve this, they need to perform simulations with many particles; however, the computational cost increases enormously as the number of particles increases. To overcome this problem, the company works on code modernization and scalable parallelization of the code to decrease the design process of the ovens.

Here are some properties of the code:

- It is a single-core Fortran code.
- The code is a 3-dimensional code.
- It is a flow code: It uses data obtained via an independent flow method (from another code).
- The mesh has around 1 million nodes.

2. First Suggestion

ŞİŞECAM has an in-house built code for unique processes in Fortran but the code requires an old Operating System and Fortran compiler. They are currently using smaller models to make the process faster and want to increase the size to have more accurate simulations. After the previous case study, they agreed on converting the whole codebase to C++ and decided to allocate engineer(s) for this. In this case study, we will help them to convert the software to a modern architecture and discuss/recommend parallelization strategies on a multicore system with OpenMP.

The tentative project plan is as follows:

Timeline	**Tasks and Milestones**
15.07.2022	Initial PoC and first results
15.08.2022	Further benchmark and evaluation results
15.09.2022	Fixes, improvements, and final version

3. Solution Stage – I

First, the in-house code and development environment details are shared between Sabancı and ŞİŞECAM teams. The development environment included Microsoft Fortran 4.0, QuickWin 4.1, and Microsoft Developer Studio, all released in 1997. Due to the lack of support in recently updated Windows operating systems, the software could neither get input nor show console output.

The Fortran code translated to modern C++ line by line, except *goto* statements which translated as loop constructs for better readability. Then, the dead code is eliminated to reduce code size. Common logical patterns are converted to functions for better readability.

Afterwards, in-house FORTRAN code is modified to use hard-coded files as I/O and it is verified that both Fortran and C++ reimplementation have the same outputs for two different real-life test cases. In addition, the variables within both implementations are also verified to be in the same states using debuggers due to small differences between floating-point numbers that can cause chaotic differences in other use cases.

4. Solution Stage – II

In the second stage, we, EuroCC experts and engineers from ŞİŞECAM, removed some redundant computations such as some particle search steps (which might have been hard to notice in the original code but evident in the re-implementation). Also, for faster convergence, we have modified the convergence threshold used to accept a particle no longer moving to the size of an atom (1e-8 cm) from exact 0.

After, the code is parallelized on processed particles, and the batch size is increased to 1000 from 10. The reason for using a large batch size is the long-tailed running time of some of the particles; using a larger batch size and dynamic scheduling we were able to increase core utilization to >90% from <20% on the development machine (with 16-core/32-thread AMD Ryzen9).

On the longest running test case given by ŞİŞECAM, the Fortran code ran for 7 hours 14 minutes on the development machine, whereas unoptimized C++ code without redundant computations ran for 27 minutes 21 seconds. After optimizations, we were able to reduce the running time to 2 minutes 14 seconds on the development machine.

5. Stage 3

Precision analysis and numerical tests are performed with the ŞİŞECAM team. After discovering a bug in the new solution, the run time is reduced to 59 seconds in the development machine. For the use cases of ŞİŞECAM, we were able to achieve binary compatibility in our experiments. Unfortunately, divergence in the results is observed when we stress-tested both solutions with higher resolution scanning than currently in use; which allowed us to find a bug in the in-house solution to perform more iterations than it is intended.

6. Results and Achievements

In the project, we were able to translate a complicated numerical code that depends on depreciated technologies to modern C++. While translating, improvements in both maintainability and performance are done. In addition, the implementation is parallelized to use multiple cores with efficiency.

In our experiments on a server with 2x 20-core Xeon 6148 2.40 GHz, we were able to reduce running time to 50.30 seconds. Even when a single thread is used, the running time was 17 minutes 4 seconds, a huge speed-up from 7 hours and 14 minutes on the development machine. Using 40 threads, the new implementation was able to achieve 20.38x speedup which can be further improved with more advanced techniques.

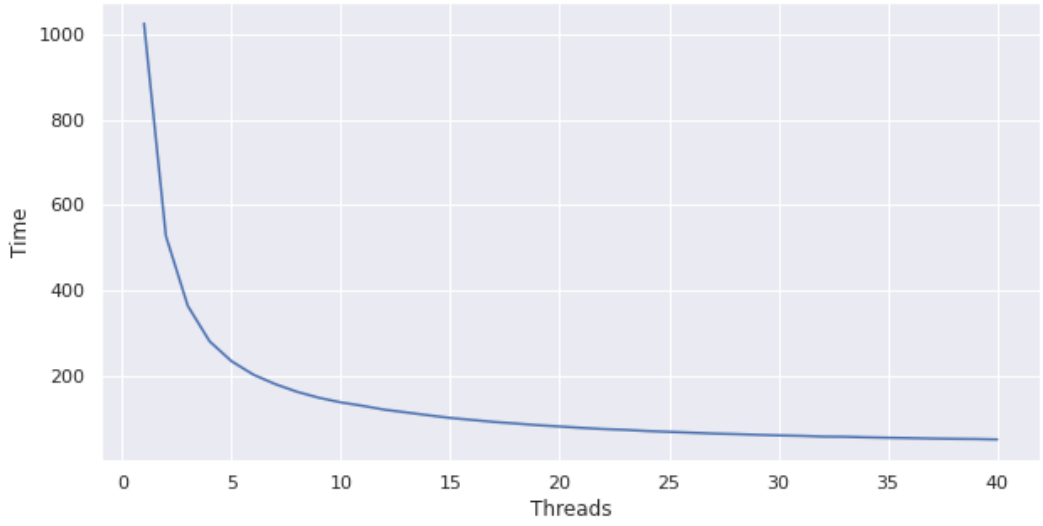

Figure 1. Execution time in seconds w.r.t number of threads used.

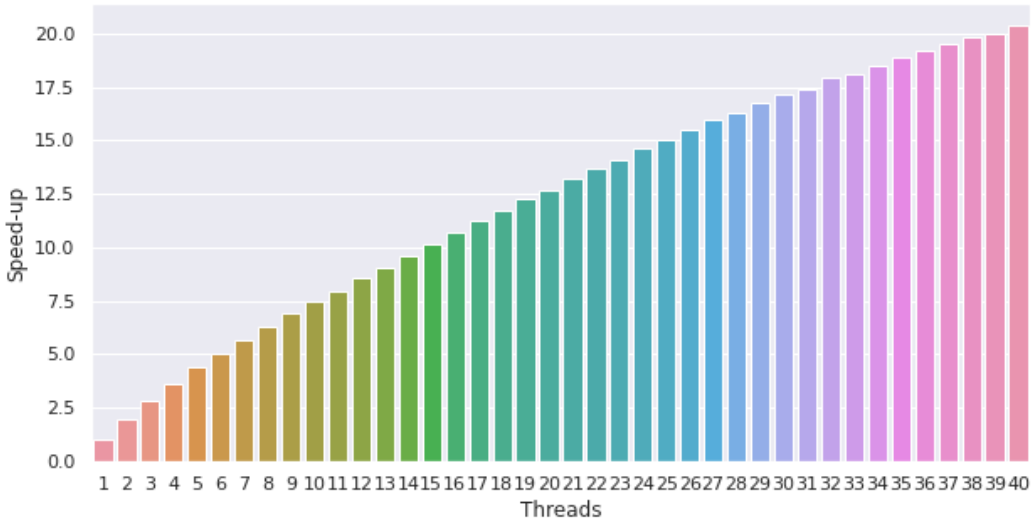

Figure 2. Speed-up achieved using multiple threads