**Development Methodology**

(talvez denominar Development Methodology and First Analysis)

In terms of development methodology, we define the following strategy: 1st) analyzing for all *loops dependencies*; 2nd) check the possibility of merges between the *for loops*; 3rd) in the *loops* that have resolvable dependencies, resolve them so that we can parallelized them; 4th) use a profiler to check the methods that have more calls and that affects more the performance of the program. During the development and analysis, this process had to be repeated to check if everything remains correctly parallelized after all the changes.

1st)

As said, we start by analyzing all the *for loops* dependencies. This way we can see all the loops that can be directly parallelized (using *omg parallel for*). In this first step, we detect:

* the loops in 4.2 (4.2.1 and 4.2.2) and 4.1.1 (the one with the update function) did not have dependencies. So they can be directly parallelized.
* The loops named as 4.3 have output-dependency.
* We didn’t detect any anti-dependency.
* QUAL O TIPO DE DEPDENCY NO 4.1 e 4

2nd)

After this, we then tried to check the possibility of merge’s between the *for loops* that have the same parameters. In this step, we analyze the 4.3 and 4.2.2 *for loops*: we tried several approaches but due to an anti-dependency detected with the merge, we conclude that this merge is impossible and return to the previous form.

3rd)

As said earlier, we detected an output-dependency on the 4.3 *loop for*. This dependency can be resolved by replicating the loop in two: a parallelizable one, that uses two auxiliar variables and the number of the thread, and the other that fill the final two variables with comparing each value in the auxiliar variables to check the final result. After this dependency has been resolved, we parallelized this *loop for*, improving the overall performance.

NOTA: DIOGO ESCALEIRA, SE QUISERES DAR MAIS ALGUM DETALHE SOBRE A IMPLEMENTAÇÃO QUE FIZESTE CASO ACHEM NECESSÀRIO.

VERIFIQUEM PF A PARTE A AMARELO PARA VER SE ESTÀ OKAY PORQUE NÃO PERCEBI a 100% ESSA ALTERACAO

4th)

In the last step of our strategy, we run a profiler: we explore multiple profiling tools (like Valgrind Profiler using massif visualizer for example) and then we decided that the best to help us understand the methods that have the most impact on our performance after all the improvements that we made, it was by using the CLion IDE profiling tool. With this tool we can check all the call stree and method list that provide us a knowledge of the methods that have more samples registered, meaning, the methods that are more callable. In the last version of our version, the majority of the calls were in the *upload* method and the others were calls to procedures concerning the *omp parallel*. We then try to improve the *upload* method but unsuccessfully.

IMAGEM DO PROFILING