Study and Optimization of a Finite Volume Application

Brito, Rui
PG22781
Department of Informatics
University of Minho
ruibrito666@gmail.com

Alves, José
PG22765
Department of Informatics
University of Minho
zealves.080@gmail.com

1 Introduction

High-Performance Computing (HPC) has been a fundamental process to science, in order to make viable certain simulations that require massive amounts of calculation. The use of several optimization techniques has helped decrease the execution time of different algorithms, providing critical results faster, thus helping research move forward at a faster pace.

This document will describe a study about improving the performance of *conv-diff*. This application calculates de heat diffusion of, let's say, a liquid while it spreads though an area. In this study we approach different ways to improve uppon the initial solution solution. The project was devised in three stages. The first one was analysing the original application, building its profile while also developing a better sequencial version. On the second stage, a shared-memory parallel version was made (*OpenMP*). Finaly on the third stage, a CUDA version is being developed, taking advantage of the massive paralellism modern GPU's have to offer.

This extended abstract presents the aplication's domain, explaining its uses and objectives. By knowing its domain and profile, a string foundation is created to develop all kinds of different optimizations. From here, an optimized sequencial version was developed, as well as an OpenMP shared-memory version. The ongoing and future work and a brief conclusion will be presented in the last sections of this extended abstract.

2 Case Study

The application analyzed for this study is *conv-diff(Convexion-Diffusion)*.This application simulates the way heat is tranfered in a fluid using the finite-volumes method.To compute the heat diffusion, the surface is represented as a mesh. Being represented by cells and edges, the algorithm will traverse all edges, calculating the contribution of the adjacent cells. This application rests in a Finite Volume Library (FVLib), which handles the structures and some of the logic functions necessary for the problem's solution.

The application's main objective is to compute a vector $\overline{\phi}$ such that $\overline{\phi} \longrightarrow G(\overline{\phi}) = \begin{pmatrix} 0 \\ 0 \\ \vdots \end{pmatrix}$ This is acomplished in three different stages:

1. We begin with a candidate vector ϕ
2. For each edge, we compute the flux F_{ij}, with i and j being the indices of the adjacent cells
3. For each cell, we compute $\sum |e_{ij}|F_{ij} - |c_i|f_i$

Thus: $\phi = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_I \end{pmatrix} \longrightarrow G = \begin{pmatrix} G_1 \\ \vdots \\ G_I \end{pmatrix}$

3 Profiling

The program consists in four major parts, reading the initial mesh from a file. Then, using the functions *makeResidual*, which calls the function *make-Flux*, the flux contributions are calculated and the vector phi is built, thus achieving a matrix free implementation. Following this, to calculate the deviation in the results from the previous operations, the function *LUFactorize*. Finally, both the meshes are writen to the output files, together with the error between them.

After analyzing the application, we conclude that the algorithm has a very high workload in the *LU-Factorize* function, comprising of more than 90% of the execution time.

4 Sequencial Optimization

After analysing the code and understanding the problem at hand, we began to notice some implementation errors, these errors were very easy to spot, such as reading the same variable repeatedly from a file and a very long chain of calculation with a very heavy division at the end which could clearly have been avoided. We changed all those aspects of the application because they were easy, requiring minimal effort. That being said, it doesn't mean they didn't pay results, computation time has been greatly reduced, with the aforementioned *LUFactorize* function taking an even more prominent role in our profile.

5 Shared Memory Parallel Optimization(OpenMP)

6 On-going and Future Work

After implementing both the sequencial version and the OpenMP version, a naïve implementation in CUDA was started. This version aims to take advantage of the graphics board performance using its vector units to diminish the compute time. For this a restructuring of the code is necessary removing most of the memory accesses while maintaining an abstraction of the system. After resolving this issue and other minor ones, we expect to achieve a boost in performance.

In future work, it is expected to develop a optimized CUDA version as well as a OpenMPI. A hybrid version using both CPU and GPU could also be a future implementation to have gains in performance. It is also expected to optimize several areas of code, questioning some decisions like using double-precision versus single-precision.

7 Conclusion

This extended abstract serves as a introduction to the study here presented. The initial results from the implementations of optimized versions, sequential and parallel, shows the capability of improvements in the computed time.

Through the development of the solutions some problems were presented, such as the mesh being disperse and the structures implemented with extensive use of pointers. This problems delayed the development of solutions, in particular the CUDA version. The decision of maintaining the abstraction of the system may prove to be a bold direction, but favorable in terms of comprehension.

In the future released paper a deep analysis of the results will be made, showing the performance improvements obtained.