Profiling FairShip to identify hotspots

Philipp A. Munkes

Abstract
The FairShip software was profiled with the Intel Parallel Studio XE 2017 software in search for computational Hotspots as a target for code optimization. No such Hotspots could be identified in the HNL simulation case. This Memo explains the installation and use of the Intel Parallel Studio XE 2017 software for profiling and outlines the findings.

1 CERN Summer Student 2017 for the SHiP collaboration
*Corresponding author: p.munkes@cern.ch

Contents
1 Prerequisites 1
2 Profiling 1
2.1 Analyzing the profiling output 1
3 Closing remarks 3

1. Prerequisites
A full installation of the FairShip software tree has to be present, preferably with debug symbols enabled. The installation notes for FairShip may be accessed here: https://github.com/ShipSoft/FairShip

Also a working installation of CVMFS pointing at the CERN repositories needs to be present and active. The appropriate installation notes are, as of June 2017, available at: https://cernvm.cern.ch/portal/filesystem/quickstart

2. Profiling
Before performing any analysis using the Intel Parallel Studio XE software, it needs to be (temporarily) installed from CVMFs. The latest version can be easily installed using the following command:

. /cvmfs/projects.cern.ch/intel/sw/psxe/linux/17-all-setup.sh

For the rest of this report we will only consider the applications amplxe-cl and amplxe-gui.

Amplxe performs its analysis in a two step process. First amplxe-cl collects information about the call stack and runtime of any function the program calls, before these results are analyzed in a second step using, for example, amplxe-gui. Amplxe provides Hotspot Analysis as a specialized profiling target out of the box, as such it is only necessary to specify the option --collect hotspots to amplxe-cl at collection time to perform this analysis.

The full command line for the analysis of an HNL simulation run with 1000 events would be:
amplxe-cl --collect hotspots -- python $FAIRSHIP/macro/run_simScript.py -n 1000

Analyzing the profiling output
Visualizing the results with amplxe-gui can be done by either specifying the path to a results folder in the call or by opening the .amplxe-file inside the results folder with the file selection dialogue. This causes amplxe to display the, pretty self-explanatory, summary page in figure 1.

In this figure the main computationally intensive code path encountered during an HNL simulation run with FairShip is highlighted and available for analysis. In the main tabular on the left all function calls made by the simulation macro are ordered by the computational time consumed by either themselves or functions called by them. The fraction of computational time consumed by the call trees sharply drops after G4Manager::Stepping, from a CPU Time percentage of 89.5% to 39.6%, suggesting this as an entry point for analysis. Causes for this drop could be either that the function is consuming a lot of computation time by itself or calling some computationally expensive functions. In the column directly beside the call tree percentage the CPU Time spent inside the function body, and not inside any called function, is displayed. The low percentage of 0.3% suggests this as a management function, mainly used to call the appropriate functions to do the desired calculations. These callees are displayed in the lower right tabular, where most of the computation time is spent during the calculation of the physical step length.

This is one of the most important steps during a Monte-Carlo simulation and it involves an analysis of the geometry and fields along the current path segments. Analyzing the computational time spent in this call tree can easily be done by clicking and expanding the call tree in the lower right tabular, which has been done in figure 3 for the most expensive call tree.
Profiling FairShip to identify hotspots — 2/3

Figure 1. Screenshot of the summary page

Figure 2. Overview over the function costs in an 1000 HNL-Events run.
This sub tree is dominated by the costs of analyzing the particle propagation in a magnetic field, although there are no obvious hot spots lending themselves as an easy target for optimizations. Repeating this process for the other call trees shows the user targets for optimization. The decision if a function lends itself to optimization is best taken in accordance to the time spent inside the function itself, as an optimization of a part of a program can only produce a speedup for that part. In the example none of the functions would be considered an ideal target for optimization, as most of them are not that computationally expensive.

3. Closing remarks

This notice shows the general process of profiling an application using the Intel Parallel Studio XE software. It also brushes on the process of selecting targets for optimization, which should be evaluated in terms of the potential speedup times the time spend in that part of the function. The Intel profiling software also enables the identification of hot loops and the direct search for functions with a lot of time spend in their function body. Optimizing compilers should also be considered, as they often have optimization passes that can remove a lot of the computational costs for particular classes of hot spots.

Lastly the cost for actually optimizing a function also needs to be taken into account. Spending the time to get a speedup of 25% in a function only taking up 0.5% of the runtime only gives a minuscule performance gain for the whole application, while spending the same time to get a 5% speedup in a function taking up 20% of the runtime would give a significantly better gain.