

CS 498 Program Optimization

Fall 2007

Term Project

This document describes the term project for CS498, Fall 2007. As you know this project is an important part of the course and its importance is reflected in that it contributes with 40% of the final grade. The project has several milestones. The main purpose of the milestones is to give you feedback on the project and no grade will be given at each milestone. Therefore, you will have the opportunity to improve your results for the final report which is the only one that will receive a grade. However, complete, high quality reports are expected at each milestone.

The main goal of the project is to optimize an application that will be chosen by each team. As you know, optimizing a program means rewriting it to achieve better performance. Each team may decide the type of performance targeted by the project (execution time, space occupied by the program, power consumption, ...). Probably the most natural type of optimization is to parallelize an application. This is an important type of optimization that is becoming increasingly important.

I expect each team to consist of two members, but I will be willing to consider smaller or larger groups. Each team should choose the application to be optimized. Students can select any application they want, but they should try to identify an application where they can apply the optimizations requested in the project. Examples of applications that can be chosen for this project are the following ones:

LAME	Educational tool to be used for learning about MP3 encoding
Mp3Gain	Adjusts mp3 files so that they have the same volume
Faac	Mpeg 2 and Mpeg 4 audio encoding software
Crypto++	C++ class library of cryptographic schemes
Ogg Vorbis	C++ class library of cryptographic schemes
CMU Sphinx	Speech Recognizer

The project will have three milestones and a final report. The objective of each milestone is purposely left fairly open to give you the opportunity to choose your own optimization strategy. The instructor will contact each team to schedule a meeting before each milestone to discuss their ideas and plans.

The milestones are:

October 5: A report is due describing the application you have chosen for the project, the type of optimization you want to perform and the reasons why you believe this application can be successfully improved. A report is due describing the baseline performance of the whole application and of each section of the application. A section can be a procedure, methods, a loop or a combination of these depending on the application. Application hotspots should be identified including the nature of the hotspots

and the source of possible inefficiencies such as cache misses or poor utilization of computations resources. You are encouraged to use Vtune to characterize each section of the application. Application hotspots should be identified including the nature of the hotspots and the source of possible inefficiencies such as cache misses or poor utilization of computations resources. Also, the baseline performance should be the “best” allowed by the compiler. Therefore, you should search the space of possible compiler options and your report should show the baseline performance of the whole application and of each section of the application before and after the compiler options have been tuned. See the Appendix for a discussion of tuning compiler switches.

October 26: A report is due describing the preliminary result of the optimization process and the discussion of unsolved problems and plans to overcome them.

November 30: A report is due of the optimization of the program with a detailed discussion of techniques applied, and a discussion of the improvement in performance for each section of the application and for the whole application. It is expected that this report will be fairly close to the final of the report. The final weeks of the semester should be left for exploration of alternative optimizations, but by this report a “significant” performance improvement should have been achieved by the team

December 10: Final report is due.

Appendix: Optimizing a program by choosing the appropriate compiler switches

Compilers have many options that can be enabled by the programmer via command line options or options embedded in the code. However, choosing the appropriate switches is hard, since the best choice depends on the application and the target architecture. As a result, compilers usually have a predefined set of options that are enabled with `-Ox`, where `x` is the level of optimization. The higher the optimization level, the more options are applied. Unfortunately, increasing the optimization level does not always result in the fastest execution. However, there is a setting of these compiler options that result in the best performance for a particular platform and application. The goal of this part of the project is to search for this setting for the application they select to optimize, but since exhaustive search is in general unfeasible (some compilers like `gcc` have more than 60 compiler options) you should choose a “reasonable” search strategy. Search strategies described in the literature include:

- 1) Non-parametric Inferential Statistics: “Automatic Selection of Compiler Options using Non-Parametric Inferential Statistics” from M. Haneda, P.M.W. Knijnenburg, and H.A.G. Wijshoff. Published in PACT 2005.
- 2) Zhelong Pan and Rudolf Eigenmann, Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning, The 4th Annual International Symposium on Code Generation and Optimization (CGO), March 2006.
- 3) Genetic algorithms. For instance, ACOVEA (Analysis of Compiler Options via Evolutionary Algorithm) can be used to find the best options to compile your application. <http://www.coyotegulch.com/products/acovea/index.html>

For this optimization problem each group can select the compiler and target architecture they prefer.

In the report due on October 4 you should include:

- 1) Values of the compiler switches that are found to be the best for the particular application, target architecture and compiler.
- 2) A table or plot of the execution times for all the different options evaluated during the search procedure. The table or plot should also include, for comparison purposes, the performance of the compiler flags `-Ox`, with `x` ranging from the lowest to the highest level of optimization.
- 3) Determine which are the section/s of the application (loops, procedures ..) which improved more with the compiler switches reported in 1) and what was the reason/s. The students must profile the application and monitor some events of the

application such as cache misses, tlb misses, etc. Tools that can be used to monitor the hardware events include the hardware counters of the machine that can be read using the PAPI library (<http://icl.cs.utk.edu/papi/>), or the INTEL Vtune performance analyzer(<http://www.intel.com/cd/software/products/asmo-na/eng/vtune/index.htm>).

Also interesting (although not required for this report) is to determine to what extent the best compiler options for a particular application and target architecture depend on the inputs to the application. If the compiler switches that result in the best performance vary depending on the inputs, it would be interesting to think how to extend the search to generate the optimal compiler settings for the different possible inputs to the application.

NOTE: The Linux INTEL software tools that include the Intel C, C++ and Fortran compiler and VTUNE are free for noncommercial software development (<http://www.intel.com/cd/software/products/asmo-na/eng/vtune/vlin/219771.htm>).