Homework #3
EC535, Spring 2025

Assigned: April 3, Thursday -- **Due:** April 11, Fri, 8pm US Eastern time

In an in-class exercise, you practiced designing an embedded system using software dataflow. In this homework, you will now practice *optimizing* an embedded system in software. You will be doing most of the homework on lab machines, using gcc as your compiler. You will also use the BeagleBone kit provided to you.

Performance Optimization

This homework is an exercise on **performance tuning**. In real life performance optimization scenarios, a designer is often faced with analyzing a large chunk of code including various files and functions. Therefore, the designer needs to have a systematic approach and a number of tools to analyze the performance bottlenecks in the code, and optimize the performance based on the results of the analysis.

In this assignment, your task is to reduce the running time (i.e., to speed up) the *qsort* application from the MiBench (https://vhosts.eecs.umich.edu/mibench/) embedded benchmark suite.

Tasks:

1. "`time`" the execution of qsort_large while running with the input file *input_large.dat*. You can copy the commands from the run-script, runme_large.sh, or you can directly run the run-script. Add the debugging flag `-g` and compiler optimization `-O` options to the compilation process (you should add to the Makefile) and time your application again. Note that it's good practice to run "`make clean`" after editing the Makefile.

2. Use `gprof` to profile the application. A quick reference file on using and understanding `gprof`: http://www.eecs.umich.edu/~sugih/pointers/gprof_quick.html

   A (more detailed) `gprof` manual is at this link: http://sourceware.org/binutils/docs-2.16/gprof/

   Recall that you will need to use "`-g -pg`" as compilation options. Modify the Makefile to add these options. After you type "`make`", observe the standard output on the screen and make sure these options are used by gcc.

   Once you have the necessary options added, you will need to run the application to get the profile data. If the compilation options are added correctly, running the application will create a *gmon.out* file. You will then use `gprof` on the gmon.out file to generate the profile.

3. Identify the function that takes the highest % of time in qsort_large. Record the % and the function.

4. Try running the code with and without various compiler optimization flags to explore the impact of compilation options on the execution time.

5. Based on your observation on the gprof output and the performance discussions we held in class this semester, try two different performance optimizations on the qsort_large program. One of these optimizations could be using a compiler optimization flag. The other one has to be an optimization through source code modification.

   Hints: You may find it useful to try using the gprof output to identify the performance-limiting sections of the code, avoiding branches in crucial loops, buffering I/O, using inline functions or macros for key routines, changing the order the code checks for delimiters, changing memory access routines, using various compilation optimization flags, and many others. When you are changing the source code, make sure to comment your changes using **//yourBUlogin** (in addition to other clarifying comments) as you will need to submit your new code.

You should compile and run the code with your two different optimizations (and with both optimizations if possible – *optional*) and time the execution. Use the same large input file while timing. **Make sure the code functionality is not altered by your optimization.** Record the following information:

Optimizations	Time for qsort_large	Time on BeagleBone
Default (w/o your optimizations, original code and makefile)		
<name of optimization-1>*		
<name of optimization-2>*		
(optional: opt.1 & opt.2 combined)		

* Explain each optimization with 1-2 sentences. Keep your explanation brief and clear.
* You are allowed to use compilation flags in only one of the optimizations, **not both.**
* Perform all the timing measurements on the same platform (back-to-back, either locally on a signals lab computer *or* remotely) to have consistency across your measurements. You are encouraged to run each instance multiple times and avoid the "outliers" (or take the median/mean) to get a more accurate idea of the performance.

Optimization for ARM-based embedded systems
Compile qsort_large for ARM (using `arm-linux-gnueabihf-gcc`) to run on your BeagleBone hardware. You can use the basic setup in Lab4 to get the board running and for transferring files. You won't need any kernel modules as you will be simply running the application on the board.

- Time the default (w/o your optimization) version and at least one of your optimized versions and compare results.
- Include a 1-2 sentence summary on how results differ on eng-grid/signals machines vs. on the embedded board.

Submission
Create a folder called yourBUlogin_HW3. Please write your answers to in a text file named BUlogin_answers.txt and put this file in the folder. **You must follow the formatting guidelines given in the sample txt file.**

You should also submit your code optimizations in the following way: please create sub-directory *opt1 (and opt2 if you have two different code optimizations) under your folder*, and put *only the files you modified* in these folders. When your modified files are copied into the default qsort folder, compiling via *make* and the default run scripts should work.

Make sure your explanations about your optimizations and your comments in the source code are brief but clear.

Add your timing results for BeagleBone in the txt file and add a short video (less than 20 seconds long) in the folder showing the qsort code running on the board.

Submit a zipped archive of your folder, yourBULogin_HW3.zip, on GradeScope.

Grading
20 points: Explanations in .txt file for the optimizations.
30 pts: Two working optimizations that achieve any speedup on signals machines (15 each)
10 pts: Proper commenting, easy-to-read coding style
15 pts: Competitive speedup for optimization (15 for fastest 7, 12 for next fastest 7, etc.) – *Applies to fully working solutions only*.
20 pts: Demonstration of default and optimized code running on BeagleBone.
5 pts: Explanation in .txt file regarding how results differ on grid vs. embedded platforms.