EECS 222: Embedded System Modeling
Winter 2020

Assignment 7 (shortened)

Posted:	February 11, 2020
Due:	February 19, 2020 at 6pm
Topic:	Performance estimation of the Canny Edge Decoder

1. Setup:

This assignment continues the modeling of our application example, the Canny Edge Detector. Using a prepared C++ model of the Canny application, we will perform performance estimation using profiling and timing measurements. In a first step, we will profile the application for relative timing so that we can identify the components with the highest computational complexity. In a second step, we then measure the absolute timing of the main Canny functions.

Again, we will use the same setup as for the previous assignments. Start by creating a new working directory, so that you can properly submit your deliverables in the end.

```
mkdir hw7
cd hw7
```

For functional validation, create again a symbolic link to the video stream files, as follows:

```
ln –s ~eecs222/public/video video
```

As a starting point for this shortened assignment, we will use here a prepared C++ model (which has been derived from the SpecC model in Assignment 6). You can copy this as follows:

```
cp ~eecs222/public/cannyA7.cpp canny.cpp
```

A corresponding **`Makefile`** is available as well:

```
cp ~eecs222/public/MakefileA7 Makefile
```

Start by inspecting the **`canny.cpp`** source file with your text editor. While you will find constructs with references to SpecC libraries, you can ignore those and consider this model as a clean C++ model of the Canny application at the point of Assignment 6.

2. Performance Estimation of the Canny Edge Detector

Step 1: Profile the application components

For an initial performance estimation of our Canny Edge Detector model, it is critical to identify the computational complexity of its main components. In other words, we want to find out which components can become a bottleneck in the implementation. To this end, we will profile our design model in this step.

We will use the profiling tools provided by the GNU community, namely **`gprof`**. In order to use the GNU profiler, you need to instrument your model (prepare it for profiling) by supplying the **`-pg`** option to the GNU compiler **`g++`**. Note that this option is already set in the provided **`Makefile`**, so you can compile the program with a simple **`make`** command.

After compilation, run your executable once, just as you would for regular simulation. In addition to the usual simulation output, this will produce a file **`gmon.out`** with profiling statistics that you can then analyze with **`gprof canny`**. This in turn will generate a detailed profiling report (in textual form) where you can inspect the function call tree and other results. For the computational complexity we are interested in, see the "flat profile" in the report.

In this step, we are only interested in the relative computational load of the components in the DUT (and we want to ignore all computation performed by the components in the test bench). Thus, assuming the total DUT load is 100%, we want to find out how much load each of the DUT components contribute.

For this first step, calculate the relative load of the DUT components as a percentage value and fill the results in the following complexity comparison table:

```
Gaussian_Smooth                         ...%
|------ Receive_Image    ...%
|------ Gaussian_Kernel  ...%
|------ BlurX            ...%
\------ BlurY            ...%
Derivative_X_Y                          ...%
Magnitude_X_Y                           ...%
Non_Max_Supp                            ...%
Apply_Hysteresis                        ...%
                                        ----
                                        100%
```

Submit the filled table in your text file **`canny.txt`** with a brief explanation of how you obtained these results.

Step 2: Measure the application performance on a reference platform

In order to obtain absolute timing information, we measure the application performance also on a reference platform. In the absence of an embedded prototyping board (which we would have available in a perfect world), we measure the delays of the major application functions on the simulator host.

For this purpose, instrument the provided model source code with timing measurement instructions. Note that we are interested only in the main components of the DUT. Specifically, you will find for each block of interest a corresponding `main` method in the provided source code, where timers can be inserted.

Add timing measurement constructs as follows:

1) Include the `time.h` header file in your model:

```
#include <time.h>
```

2) Create timer variables, as follows:

```
clock_t Tstart, Tstop;
double T1 = 0.0;
```

3) To start a timer, place the following statement right before the statements to be measured:

```
Tstart = clock();
```

4) To stop the timer, place the following statement right after the statements to be measured:

```
Tstop = clock();
```

5) Finally, to calculate the CPU time of the measured function in seconds, you can use a calculation like this:

```
T1 = (double)(Tstop-Tstart)/CLOCKS_PER_SEC;
```

6) Before the end of simulation, print the measured delays to the screen.

For simplicity, you may use global variables for the timers in this instrumentation.

For this second step, note both the absolute time measured (in seconds) and the relative delays of the components (as a percentage value) in a table similar to the following template:

```
Gaussian_Smooth          ...sec     ...%
|------ Receive_Image    ...sec     ...%
|------ Gaussian_Kernel  ...sec     ...%
|------ BlurX            ...sec     ...%
\------ BlurY            ...sec     ...%
Derivative_X_Y           ...sec     ...%
Magnitude_X_Y            ...sec     ...%
Non_Max_Supp             ...sec     ...%
Apply_Hysteresis         ...sec     ...%
                         ...sec     100%
```

Submit this filled table also in your text file **canny.txt**.

3. Submission:

For this assignment, submit the profiling and timing comparison results using the following deliverable:

canny.txt

Since we are combining Assignment 6 and Assignment 7 this week, copy the **canny.txt** file into your **hw6** directory and submit from there. Change into the parent directory of your **hw6** directory and run the **~eecs222/bin/turnin.sh** script.

Again, be sure to submit on time. Late submissions will not be considered!

To double-check that your submitted files have been received, you can run the **~eecs222/bin/listfiles.py** script.

For any technical questions, please use the course message board.

--
Rainer Dömer (EH3217, x4-9007, doemer@uci.edu)