EECS 222: Embedded System Modeling
Winter 2019

Assignment 7

Posted: February 19, 2019
Due: February 27, 2019 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. We will now use our model for initial performance estimation. 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 in the previous assignments, you have again the choice of using either SpecC or SystemC for your performance estimation. Both SLDLs are equally suitable for this assignment, but the profiling step uses different tools.

As starting point, you can use your own SLDL model which you have created in the previous Assignment 6. Alternatively, you may start from the provided solution for Assignment 6 which you can copy as follows:

```
cp ~eecs222/public/cannyA6_specc_ref.sc Canny.sc
cp ~eecs222/public/cannyA6_systemc_ref.cpp Canny.cpp
```

You may also want to reuse the `Makefile` from the previous assignments:

```
cp ~eecs222/public/MakefileA5SpecC ./
cp ~eecs222/public/MakefileA5SystemC ./
```

Again, depending on whether you choose SpecC or SystemC for your modeling, rename the corresponding file into the actual **`Makefile`** to be used by **`make`**.

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.

For the SpecC model, we will use the profiler integrated into the System-on-Chip Environment (SCE). Here, we will use the latest version of SCE, namely **`/opt/sce/bin/sce`**. When the GUI has started, create a new project (**`Project->New`**), import your source code (**`File->Import`**), add the model to your project (**`Project->AddDesign`**), and rename it to **`CannyA7`**. Next, compile (**`Validation->Compile`**) and simulate (**`Validation->Simulate`**) your model in SCE. By default, the model is automatically instrumented, so that you can run the profiler next (**`Validation->Profile`**). The GUI will then present you with the profiling results (i.e. computation counts) in numerical or graphical form (select desired behaviors, right-click, **`Graphs->Computation`**).

For the SystemC model 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++`**. After compilation, run your executable once, just as you would for regular simulation. 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 and which tool you used.

Step 2: Measure the application performance on a reference platform

In order to obtain absolute timing information, we measure the application performance 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 platform in this step.

For this purpose, instrument your model source code with timing measurement instructions, 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 function call to be measured:

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

4) To stop the timer, place the following statement right after the function call 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) Right before the end of simulation, print the measured delays to the screen.

For simplicity, you may use global variables for this timing 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:

```
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 second filled table also in your text file **`Canny.txt`**.

3. Submission:

For this assignment, submit the following deliverables:

`Canny.sc` *or* **`Canny.cpp`**
`Canny.txt`

As before, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. In addition, include the profiling comparison results and a brief explanation.

To submit these files, change into the parent directory of your **`hw7`** directory and run the **`~eecs222/bin/turnin.sh`** script. As before, note that the submission script will ask for both the SystemC and SpecC models, but you need to submit only the one that you have chosen for your modeling.

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)