EECS 222A
System-on-Chip Description and Modeling
Spring 2012

Assignment 4

Posted:	May 11, 2012
Due:	May 18, 2012 at 12pm (noon)
Topic:	Parallelization of Application Example

1. Setup:

We will use the same setup as for Assignment 3 and again use the latest SCE version:

```
source /opt/sce/bin/setup.csh
```

In order to use **`turnin`** to submit your deliverables, create a new directory named **`hw4`** (next to your **`hw3`** directory) and work there:

```
mkdir hw/hw4
cd hw/hw4
```

2. Application Example

We will continue with the project of designing a system-level model for the *Canny Edge Detector* algorithm. This assignment starts one step after the point where Assignment 3 ended. For your reference, we have provided a solution file.

```
cp ~eecs222/EECS222A_S12/canny_a3_ref.sc .
```

As discussed in Lecture 5, we have now inserted an additional level of hierarchy into the DUT. Also, some additional clean-up has been performed and a few other adjustments have been applied. So, for this Assignment 4, we have again prepared a source file to start from, namely **`canny_a4_start.sc`**.

The hierarchy tree of the corresponding **canny_a4_start.sir** model looks as follows:

```
sir_tree -blt canny.sir
B i o   behavior Main
B i l   |------ Monitor monitor
B i c   |------ Platform platform
B i s   |       |------ DUT canny
B i l   |       |       |------ Apply_Hysteresis apply_hysteresis
B i l   |       |       |------ Derivative_X_Y derivative_x_y
B i l   |       |       |------ Gaussian_Smooth gaussian_smooth
B i l   |       |       |------ Magnitude_X_Y magnitude_x_y
B i l   |       |       \------ Non_Max_Supp non_max_supp
B i l   |       |------ DataIn din
B i l   |       |------ DataOut dout
C i l   |       |------ c_img_queue q1
C i l   |       \------ c_img_queue q2
B i l   |------ Stimulus stimulus
C i l   |------ c_img_queue q1
C i l   \------ c_img_queue q2
```

To set up, use the following as starting point for this assignment:

```
cp ~eecs222/EECS222A_S12/canny_a4_start.sc canny.sc
cp ~eecs222/EECS222A_S12/Makefile .
cp ~eecs222/EECS222A_S12/golfcart.pgm .
cp ~eecs222/EECS222A_S12/ ref_golfcart.pgm
       _s_0.60_l_0.30_h_0.80.pgm .
```

As for Assignment 3, we provide again a **Makefile** and the reference image so that you can compile, run, and test your code quickly (type **make** in your shell or simply use Eclipse). Note, however, that in contrast to the compiler command in the previous **Makefile**, we now have enabled the parallel simulation feature of the latest SpecC compiler (see below for more discussion on this).

3. Tools

Please refer to the previous assignments regarding helpful Linux tools for this project. Again, you may use any text editor of your choice and use the SpecC compiler via the command line interface. Alternatively, we recommend our extended version of *Eclipse*, an open source IDE, which includes specific support for SpecC projects (and this assignment, in particular!).

3.1 Eclipse Update:

In addition to *(a) SpecC syntax highlighting*, *(b) Automatic compiling on save*, *(c) Outline View*, and *(d) Behavior Hierachy*, the SpecC-enhanced Eclipse now offers a new display that shows variable accesses and potential conflicts in parallel execution.

(e) Non-local Variable View: This is not open initially. To open it, select from the menu **`Window -> Show View -> Other`**, find category **`SpecC`**, and select **`Non-local Variables`**.

Before you can use the Non-local Variable View, please make sure both **`Behavior Hierarchy`** (**BH**, see instructions for Assignment 3) and **`Non-local Variables`** (**NV**) are visible in Eclipse. For example, if both views appear in the same sub-window beneath the editor after you open them (only one can be seen at a time), then you can drag **BH** or **NV** and drop it into the sub-window at the right side of the editor (so that you can see both).

Note that there are two ways to use **NV**:

1. Check data-flow for correct ports: if you select a leaf behavior in **BH**, the non-local variables accessed in that behavior will be displayed in **NV**. The variables listed are defined or used outside the selected behavior and essentially are inputs or outputs of the behavior. Consequently, these should be converted to ports for proper modeling.

2. Check data conflicts for correct parallelism: once you have created parallel behaviors, for example, **`par{A; B;}`**, you can multi-select the parallel behaviors by first selecting **`A`** in **BH** and then holding Ctrl on your keyboard when selecting **`B`**. The variables accessed by both **`A`** and **`B`** are then displayed in **NV**. More importantly, any potential data conflicts due to shared variables between **`A`** and **`B`** are highlighted in red. These are the variables which may cause erroneous parallel execution!

For this assignment, this new feature should proof to be very useful.

4. Instructions

Please time yourself for this assignment. At the end, we would like to know how many minutes this took for you. Thanks!

The purpose of this assignment is to introduce and explicitly specify potential parallelism in our application model.

As discussed in Lectures 5 and 6, we will focus our attention to the behavior **`Gaussian_Smooth`** which contains the highest amount of computation in the Canny application. The goal is to parallelize this block so that we can speed up the overall computation.

For the purpose of this assignment, we will aim at a maximum of 4 parallel blocks executing at the same time.

As discussed in class, we will decompose the behavior **`Gaussian_Smooth`** into three types of behaviors, namely a preparation step **`Prep`**, the horizontal image blurring **`BlurX`**, and the vertical image blurring **`BlurY`**. For each of these behaviors, multiple instances may be used in order to maximize the parallelism of the Gaussian Smooth method. How many instances are used, how they are connected, and which ones actually run in parallel, is to be answered as part of this assignment.

Hint on parallelization: Same as many other graphics applications, we can parallelize the image processing by splitting the picture into multiple parts along its rows or columns and work on those slices in parallel. Here, the blurring can be performed the same way. To do this, we recommend to pass the entire image to each parallel unit, and also pass in the range of rows or columns (via ports) that the unit is supposed to work on.

Hint on validation: In order to validate whether or not your parallelism works safely, it is useful to run the simulation also in parallel. For this, we have now enabled the parallel simulation feature of the latest SpecC compiler in the provided **`Makefile`**. Specifically, we now call **`scc`** with the option **`–par`** which instructs it to utilize multiple available cores on the host in parallel. In our case, please use the machines **`eta.eecs.uci.edu`** or **`theta.eecs.uci.edu`** for your simulation. Both have 2 cores each that the parallel simulator will use.

As discussed in class, this not only can provide you with faster simulation speed, it also helps in detecting concurrency problems in your model. In particular, with parallel simulation, it is highly likely that shared variables with access conflicts during parallel usage actually produce errors during simulation (which is what we want!).

Throughout your model recoding, make sure that it still compiles without any warnings, runs without any errors (even when parallel simulation is enabled), and generates exactly the expected output image.

You are done with this assignment when the **`Gaussian_Smooth`** behavior has been decomposed into the three behavior types and up to 4 instances of these run concurrently. Your model should not contain any global variables or global functions and your hierarchy should be “clean” for synthesis purposes (no “dirty” behavior should be part of the DUT).

Please note the time when you are done. Thanks!

5. Submission:

For this assignment, submit the following deliverables:

```
canny.sc
canny.txt
```

As before, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Be brief!

To submit the deliverables, change into the parent directory of your **`hw4`** directory and enter **`turnin`**. As in the previous assignments, the **`turnin`** command will locate the files listed above and allow you to submit them.

Remember that you can use the **`turnin`** tool to submit at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

Late submissions cannot be considered!

Extra credit: In addition to these deliverables, we would like to ask you to complete a short survey on your experience with the extended **`eclipse`**. Please check the course message board for this survey.

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