ESESC Tutorial Demos

MICRO-46

December 7, 2013

Contents

1 Building ESESC

1.1 Compile and run a default benchmark

1.1.1 Assumptions:

- Running ESESC on an x86-64 system that has either Ubuntu 12.04 or Arch Linux installed along with all necessary packages for the appropriate OS.
- The ESESC repository has been cloned from GitHub and is checked out in `~/projs/esesc`

1.1.2 Description:

For this demo you will compile ESESC in Release and Debug mode and then run a short benchmark and view the statistics that are generated.

1.1.3 Steps:

1. Make sure you have the latest version of the ESESC code:

```
cd ~/projs/esesc
git pull
```

2. Create the build directory:

```
mkdir -p ~/build/debug
mkdir -p ~/build/release
```

3. Compile ESESC:

```
cd ~/build/release
cmake ~/projs/esesc
make
```

4. For this demo we will use the release build, so create a run directory in the release sub-directory:

```
mkdir -p ~/build/release/run
```

5. Create local configuration files and binaries in the run directory:

```
cp ~/projs/esesc/conf/*.conf ~/build/release/run
cp ~/projs/esesc/bins/* ~/build/release/run
```

6. Run ESESC and view results:

```
cd ~/build/release/run
~/build/release/main/esesc
~/projs/esesc/conf/scripts/report.pl -last
```

7. Build ESESC in Debug mode before the break (needed for the next demo)

```
cd ~/build/debug
cmake -DCMAKE_BUILD_TYPE=Debug ~/projs/esesc
make
mkdir run
cd run
cp ~/projs/esesc/conf/* .
cp ~/projs/esesc/bins/* .
```

1.1.4 Note:

View file `~/build/release/run/esesc.conf`, the parameter benchname is the benchmark which will be run. By default the launcher executable is being used to load the Crafty benchmark.

2 Timing Model Demo

Assumptions:

Have completed debug build of ESESC from previous demo

- Source directory: `~/projs/esesc`
- Build directory: `~/build/debug`
- Run directory: `~/build/debug/run`

2.1 Debugging ESESC with GDB:

2.1.1 Description:

For this demo you will learn basic debugging techniques for ESESC.

2.1.2 Steps:

1. Run ESESC on GDB, and add a breakpoint:

   .*a*) From the run directory execute the command:
   `gdb ../main/esesc`

   .*b*) From GDB, type:
   `b OoOProcessor::execute()`

2. Start execution in GDB

   .*a*) In GDB, type:
   `r -c esesc.conf`

3. Continue the execution and check status

   .*a*) In GDB type:

   ```
   p this->dumpROB()
   c 1000
   p this->dumpROB()
   ```

4. Play around with available information

   .*a*) At the breakpoint, type
   `p globalClock`

   .*b*) Use the auto-completion to check other possibilities

5. Exit GDB:

   .*a*) Type:

   ```
   quit
   ```

3 Sampling Demos

Assumptions:

- Source directory: `~/projs/esesc`
- Build directory: `~/build/release`
- Run directory: `~/build/release/run`

3.1 Single-threaded with Time Based Sampling (TBS):

3.1.1 Assumptions:

Have ESESC built and complete steps from previous demo.

3.1.2 Description:

For this demo you will run the single-threaded benchmark crafty.

3.1.3 Steps:

1. Modify `esesc.conf` file in the run directory to select the sampler mode.
   .a) Open `esesc.conf` in a text editor.
   .b) Locate the line that says:
   `samplerSel = . . .`
   .c) If necessary modify this line to say
   `samplerSel = "TASS"`
   .d) Save `esesc.conf` and exit
2. Run ESESC
   .a) From the run directory execute the command:
   `../main/esesc`
   .b) View results with report.pl
   .c) From the run directory execute the command:
   `~/projs/esesc/conf/scripts/report.pl -last`
3. Look at the execution time and instruction breakdown in the report output.

3.2 Multi-threaded with Time Based Sampling (TBS):

3.2.1 Description:

For this demo you will run the multi-threaded benchmark `blackscholes`

3.2.2 Assumptions:

Complete steps from previous demo and have config files and executable in run directory.

3.2.3 Steps:

1. Modify `esesc.conf` file in the run directory to configure multiple CPU cores
   .a) Open `esesc.conf` in a text editor.
   .b) Locate the lines that say:
   ```
   cpuemul[0] = 'QEMUSectionCPU'
   cpusimu[0] = "$(coreType)"
   ```
   .c) Modify these lines to specify additional CPU cores by changing the `[0]` to `[0:3]`. This will specify 4 homogeneous cores. After the changes the file should contain the lines:
   ```
   cpuemul[0:3] = 'QEMUSectionCPU'
   cpusimu[0:3] = "$(coreType)"
   ```
2. Change sampling mode to Time Based Sampling (TBS).
   .a) In `esesc.conf` locate the line that says: `samplerSel = "TASS"`
   .b) Modify it to say `samplerSel = "TBS"`
3. Change the TBS parameters to skip instructions at the start of execution:
   .a) In `esesc.conf` locate the section `[TBS]`
   .b) Modify the `nInstSkip` and to skip 1 billion instructions so that the lines read as follows:
   ```
   nInstSkip = 1e9
   ```
4. Change the benchmark to blackscholes:
   .a) In `esesc.conf` locate the line that says:
   ```
   benchName = . . .
   ```
   .b) Modify it to say:
   ```
   benchName = "launcher -- blackscholes 2 blackscholes.input blackscholes.output"
   ```
   .c) Save `esesc.conf` and exit
5. Run ESESC
   .a) From the run directory execute the command:
   ```
   ../main/esesc
   ```
6. View results with report.pl
   .a) From the run directory execute the command:
   ```
   ~/projs/esesc/conf/scripts/report.pl -last
   ```
7. Look at the execution time and instruction breakdown in the report output.

3.3 Adding a new counter for statistics

3.3.1 Description:

This demo shows how to add a new counter to keep statistics. Many of the most useful counters are already included in ESESC, but it is likely that you will need to add new counters during your research. For this demo we will add a counter that counts the number of consecutive stores that are retired in an out-of-order processor.

3.3.2 Steps:

1. The first thing that needs to be done is to declare the counter in the appropriate class.

   .*a*) Open `~/projs/esesc/simu/libcore/OoOProcessor.h`

   .*b*) Locate the `OoOProcessor` class declaration. We will add a `GStatsCntr` (see `misc/libsuc/GStats.h` for more information) named `cons_st_ret` to the class. Since it is only used in `OoOProcessor` we can add it as the last private member. Add the following line before `protected:` and then save and close the file.

   ```
   GStatsCntr cons_st_ret;
   ```

2. Next we need to add the counter to the constructor for the class.

   .*a*) Open `~/projs/esesc/simu/libcore/OoOProcessor.cpp`

   .*b*) Locate the code for the `OoOProcessor` constructor. The `cons_st_ret` GStat will need to be initalized by calling its constructor and passing it a string that is the name of the new GStat. Usually it is a good idea to include the variable name with the GStat name. Processor stats can be instantiated as many times as there are cores, so they are prefaced with `P(%d)_` where `%d` is the processor number. Insert the following line before `,cluserManager`:

   ```
   ,cons_st_ret("P(%d)_cons_st_ret",i)
   ```

3. Next we need to add logic which determines when to increment the counter. We will do this in the `OoOProcessor::retire()` method. The method includes the statement `nCommitted.inc(dinst->getStatsFlag());` which is used to increment the total number of committed instructions. We will declare a bool to see if the previous instruction was a store. Then if it was we will increment our counter.

   .*a*) First define a bool outside of the ROB for loop (existing loop shown below)

   ```
   bool prev_st = false;
   for(uint16_t i=0 ; i<RetireWidth && !rROB.empty() ; i++)
   ```

   .*b*) Then inside the loop add the following code after the `nCommitted.inc(dinst->getStatsFlag());` statement. Then save and exit the file.

   ```
   if(dinst->getInst()->isStore()) {
     if(prev_st) {
       cons_st_ret.inc(dinst->getStatsFlag());
     }
     prev_st = true;
   } else {
     prev_st = false;
   }
   ```

4. Rebuild **debug** build of ESESC by executing `make` in the `~/projs/build/debug` directory.

5. Modify `esesc.conf` to use `crafty.armel` as the benchmark, and get rid of instruction skip at the start of execution.

   .*a*) Edit `esesc.conf` and locate `benchName` and change it to:

   ```
   benchName - "crafty.armel"
   ```

   .*b*) Locate `nInstSkip` in `[TBS]` and set it to `1`.

   ```
   nInstSkip = 1
   ```

6. Run ESESC by executing `../main/esesc < crafty.input` in the run directory.

7. Use `report.pl -last` to find the filename of the last report file that was generated. Then execute `grep "P(.)_cons_st_ret" <filename>` to see the new GStat counter.

4 Power Model Demo

Assumptions:

- Source directory: `~/projs/esesc`
- Build directory: `~/build/release`
- Run directory: `~/build/release/run`

4.1 Enabling the power model and viewing the power numbers.

4.1.1 Assumptions:

Have ESESC built and complete steps from previous demo.

4.1.2 Description:

For this demo you will run the single-threaded benchmark crafty

4.1.3 Steps:

1. Modify `esesc.conf` file in the run directory to select the sampler mode.
   .*a*) Open `esesc.conf` in a text editor.
   .*b*) Switch back to a single core configuration for the demo.
   Locate the lines that say:
   ```
   cpuemul[0:3] = 'QEMUSectionCPU'
   cpusimu[0:3] = "$(coreType)"
   ```
   Modify these to:
   ```
   cpuemul[0] = 'QEMUSectionCPU'
   cpusimu[0] = "$(coreType)"
   ```
   .*c*) Choose the right benchmark for this demo.
   ```
   benchName = "launcher -- stdin crafty.input -- crafty"
   ```
   .*d*) Locate the line that says:
   ```
   enablePower = false
   ```
   Modify this line to say:
   ```
   enablePower = true
   ```
   .*e*) Save `esesc.conf` and exit
2. Run ESESC
   .*a*) From the run directory execute the command:
   ```
   ../main/esesc
   ```
   .*b*) View results with report.pl
   .*c*) From the run directory execute the command:
   ```
   ~/projs/esesc/conf/scripts/report.pl -last
   ```
3. Look at the power numbers tabulated per sub-block and memory structure and the total chip power.

5 Thermal Model Demos

Assumptions:

- Source directory: `~/projs/esesc`
- Build directory: `~/build/release`
- Run directory: `~/build/release/run`

5.1 Thermal Run Output Files

5.1.1 Description:

For this demo you will do a full performance-power-thermal run with Crafty benchmark and check the generated thermal-related output files.

5.1.2 Assumptions:

Complete steps from demo 1 and have the config files and executable in run directory.

5.1.3 Steps:

1. Modify `esesc.conf` file in the run directory to enable power and thermal.

   .*a*) Open `esesc.conf`.

   .*b*) Locate the following line.
   `enablePower = false`

   .*c*) Modify this line to say.
   `enablePower = true`

   .*d*) Locate the following line.
   `enableTherm = false`

   .*e*) Enable thermal stage by changing false to true.
   `enableTherm = true`

   .*f*) Select the sampler.
   `samplerSel = "TBS"`

   .*g*) Choose `Crafty` benchmark.
   `benchName = "launcher -- stdin crafty.input -- crafty"`

   .*h*) Save `esesc.conf` and exit.

2. Run ESESC, from the run directory execute:

```
../main/esesc
```

3. Extract thermal metrics and statistics.

   .*a*) Execute the following command in run directory.
   `~/projs/esesc/conf/scripts/report.pl -last`

   .*b*) Locate `Thermal Metrics` table in the report.pl output.

- `maxT(K)`: Maximum temperature
- `gradT`: Gradient temperature across the chip
- `Reliability`: Measures mean time to failure based on EM, NBTI, Throttle cycle…
- `ChipLeak (W)`: Total chip leakage
- `ChipPower (W)`: Total chip power
- `Energy (J)`: Total chip energy
- `ThrotCycles`: Thermal throttling cycles
- etc.

.*c*) Check the other thermal related statistics in `esesc_microdemo.?????`

4. Plot time vs. total power vs. maximum temperature from the same thermal run that just finished.

.*a*) Open thermal trace file.

- `temp_esesc_microdemo.????? report`
- The first column is the time at which temperature is recorded.
- The other columns each belong to a particular block as indicated in the top row and contain temperature changes over time.
- Notice the highest temperature belongs to column 9, Register File.

.*b*) Exit.

.*c*) Open total power trace.

- `totalpTh_esesc_microdemo.?????`
- First column is the time.
- Second column is the total power (dynamic + scaled leakage power based on temperature) consumed by the chip.

.*d*) Exit.

.*e*) Temperature and total power traces both record the same timestamp. They can be plotted on the same graph.

.*f*) In your `run` directory, launch gnuplot command mode.

```
gnuplot
```

.*g*) In gnuplot environment, enter the following commands.
ref

.*i*) Set time on the x axis.

```
set xrange [0: 0.18]
set xlabel "Time"
```

.*ii*) Set temperature on the y1 axis. The temperature belongs to the hottest architectural block, Register File.

```
set yrange [300:380]
set ytics nomirror
set ylabel "Temperature"
```

.*iii*) Set power on the y2 axis.

```
set y2range [0:5]
set y2tics
set y2label "Total Power"
```

.*iv*) Plot data.

```
plot "temp_esesc_microdemo.?????" u 2:9 w lines t "Temperature" axis x1y1,
"totalpTh_esesc_microdemo.?????" u 1:2 w lines t "Total Power" axis x1y2
```

.*v*) Observe the changes in total power versus temperature.

.*vi*) Return to run directory.

```
exit
```

You have now finished a complete thermal run and checked the relevant report files.

5.2 Generating a New Floorplan

5.2.1 Description:

In this demo, you will generate a floorplan for a dual core chip.

5.2.2 Assumptions:

Complete steps from demo 1 and have the config files and executable in run directory.

5.2.3 Steps:

1. Modify `esesc.conf` file in the run directory to configure multiple CPU cores.

   .*a*) Open `esesc.conf`.

   .*b*) Locate the lines that say:
   ```
   cpuemul[0] = 'QEMUSectionCPU'
   cpusimu[0] = "$(coreType)"
   ```

   .*c*) Modify these lines to specify additional CPU cores by changing the [0] to [0:1]. This will specify 2 homogeneous cores. After the changes the file should contain the lines:
   ```
   cpuemul[0:1] = 'QEMUSectionCPU'
   cpusimu[0:1] = "$(coreType)"
   ```

   .*d*) Change the benchmark for multicore to multithreaded.
   ```
   benchName = "launcher -- fft -p2 -m20"
   ```

   .*e*) Save and exit.

2. Run the following commands in build directory to generate the floorplanning tool executable.
   ```
   make floorplan
   ```

3. In run directory, run the floorplan script to generate a new floorplan for the dual core config.
   ```
   ~/projs/esesc/conf/scripts/floorplan.rb BuildDir_Path SrcDir_Path RunDir_Path NameMangle
   ```

   - `BuildDir_Path`: Path to the ESESC build directory
   - `SrcDir_Path`: Path to the ESESC source directory
   - `RunDir_Path`: Path to the configuration files in run directory
   - `NameMangle`: A string to attach to 'floorplan' and 'layoutDescr' sections

4. Once floorplan script finishes running, the links to the floorplan in `pwth.conf` are updated.
   ```
   floorplan[0] = 'floorplan_NameMangle'
   layoutDescr[0] = 'layoutDescr_NameMangle'
   ```

5. New layout and floorplan definitions in `flp.conf` have been created under the following sections.
   ```
   [layoutDescr_NameMangle]
   [floorplan_NameMangle]
   ```

6. A copy of layout and floorplan description is also saved in `new.flp` in run directory.

You have now finished generating the new floorplan and can continue running ESESC in thermal mode for a suitable benchmark.

5.3 Full Thermal Run with Multicore Floorplan and Thermal Management Policy

5.3.1 Description:

For this demo you will run the multi-threaded benchmark FFT. You will enable and use two different thermal management policies: DVFS and thermal throttling and compare the results. You will also learn to enable and use graphics feature for floorplan thermal map.

5.3.2 Assumptions:

Complete steps from demo 1 and have the config files and executable in run directory. Complete demo 6 to setup dual core and generate the corresponding floorplan.

5.3.3 Steps:

1. Complete steps from demo 6.
2. Set the thermal management policy to thermal throttling.
   - .*a*) Open `esesc.conf`.
   - .*b*) Set thermal throttling temperature.
     `thermTT = 373.15`
   - .*c*) Save `esesc.conf` and exit.
   - .*d*) Open `pwth.conf`
   - .*e*) Disable turbo mode.
     `enableTurbo = false`
   - .*f*) Locate the following line.
     `enableGraphics = false`
   - .*g*) To dump temperature map related files, modify it to say.
     `enableGraphics = true`
   - .*h*) Save and exit.
3. Run ESESC
   `../main/esesc`
4. Run report.pl to extract results from the last ESESC report generated.
   `~/projs/esesc/conf/report.pl -last`
5. Look at the `Thermal Metrics` section of the `report.pl` output.
6. Create a short video from the thermal map `svg` snapshots in your run directory. Use the following command as an example.
   `convert -delay 0.3 lcomp-NORM_layer-2_smpltype-CUR_0.*` fft_tt.gif
7. View `fft_tt.gif` using a browser and observe the changes in temperature in various architectural blocks of each core.
8. Now change the thermal management policy to `dvfs_t` and re-run `esesc`.
   - .*a*) Open `pwth.conf`

.*b*) Enable turbo mode.
`enableTurbo = true`

.*c*) Enable DVFS based on temperature.
`turboMode = dvfs_t`

.*d*) Save and exit.

.*e*) Open `esesc.conf`

.*f*) Set thermal throttling temperature.
`thermTT = 373.15`

.*g*) Save and exit.

9. Run ESESC
   `../main/esesc`
10. Run report.pl to extract results.
    `~/projs/esesc/conf/report.pl -last`
11. Look at the `Thermal Metrics` section of the `report.pl` output.
12. Create a short video from the thermal map `svg` snapshots in your run directory. Use the following command as an example.
    `convert -delay 0.3 lcomp-NORM_layer-2_smpltype-CUR_0.*` fft_dvfs.gif
13. View `fft_dvfs.gif` using a browser and observe the changes in temperature in various architectural blocks of each core.