# International Workshop on *Design Principles for Next Generation Embedded Computing Systems*

**Amit Kumar Singh**

**School of Computer Science and Electronic Engineering**

**University of Essex**

**United Kingdom**


W: http://aksingh.co.uk/ E: a.k.singh@essex.ac.uk

# Programming Demonstration

# Mapping

- Mapping or CPU affinity – defines the number of cores and their types to be used by an application
- It has huge impact on execution time and thus energy consumption (Power * time)
- In Linux, defined using `taskset`
  - `taskset -c 4,5 ./ApplicationName`
  - `taskset -c 3-6 ./ApplicationName`

# What do you think of execution time when using higher number of cores?

Increases

Decreases

May increase or decrease

# DVFS and Linux Power Governors

- *Cpufrequtils* - is a Linux power management tool;
- “sudo apt-get install cpufrequtils” to install

- *user@host:~$ cpufreq-info*
  *Shows various information*
  - *hardware limits: 798 MHz - 2.00 GH*
  - *havailable frequency steps: 798 MHz, 1.06 GHz, 1.33 GHz, 1.60 GHz, 2.00 GHz*
  - *available cpufreq governors: userspace, ondemand, conservative, powersave, performance*
  - *current policy: frequency should be within 798 MHz and 2.00 GHz. The governor "conservative" may decide which speed to use within this range.*
  - *current CPU frequency is 798 MHz.*

# Cpufrequtils

```
$cpufreq-info -o


$cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_ava
ilable_governors


$echo "performance" >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_gov
ernor
```

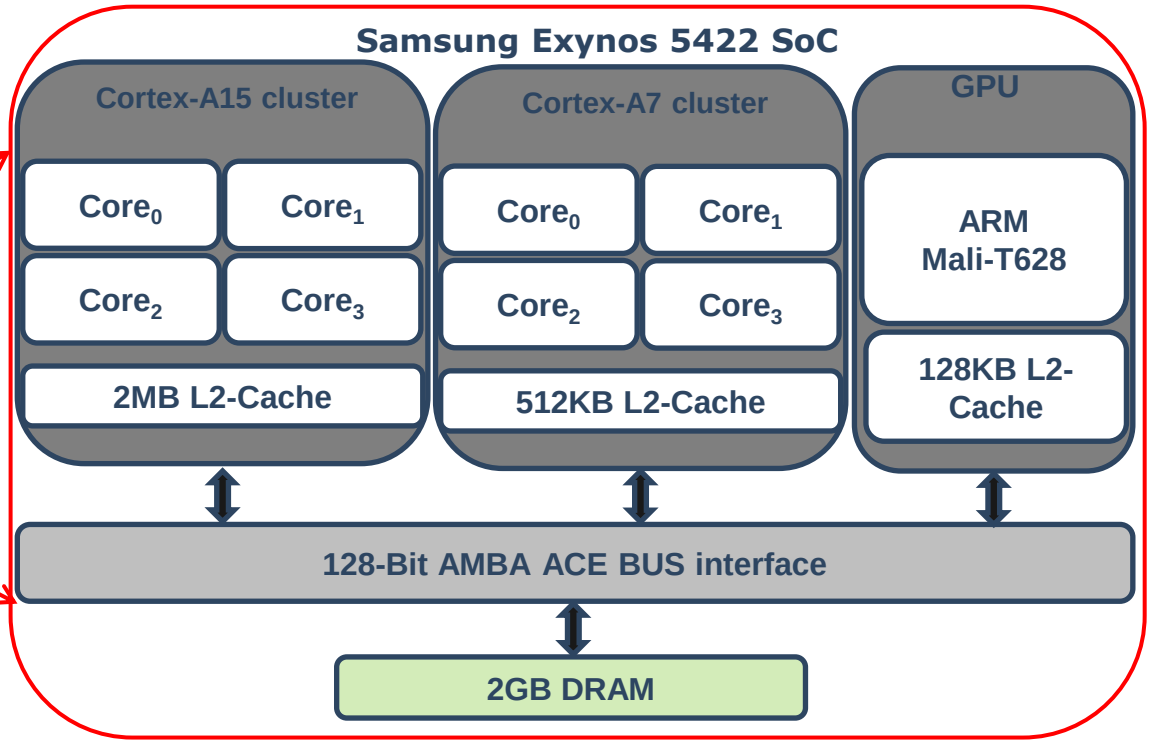
# When an application is running, which knob to change to achieve reduced execution time?

- Mapping
- DVFS (Voltage/Frequency)
- Both Mapping and DVFS (Voltage/Frequency)

# Multi-threaded Programming on Exynos 5422 MPSoC

## (Execution Time and Thermal Behaviour)

# Exynos 5422 MPSoC

# Multithreaded Applications/Benchmarks

- *MultiThreadBench Work-package*
  - We will explore the effect of mapping and DVFS (operating frequency) of CPU cores on the execution time.
- *RSABench Work-package*
  - We will explore the thermal behavior of 4 ARM big CPU cores

# *MultiThreadBench* Work-package

- Open up a terminal and go in *Benchmarks* directory

```
$cd Benchmarks
```

- Check the contents of the Benchmarks folder

```
$ls
```

- Change directory to *MultiThreadBench*

```
$cd MultiThreadBench
```

# *MultiThreadBench* Work-package

- Check the folder contents

```
$ls
```

  - *threadbench.c*
    - *Multi-threaded program*
  - *run_bench.sh*
    - A script including commands for
      - Compiling benchmark
      - Changing core frequencies
      - Changing mapping and Executing

# *MultiThreadBench* Work-package

- Compiling benchmark

```
$gcc -pthread threadbench.c -lm -o benchmark
```

- Changing core frequencies

```
$echo "900000" >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_m
ax_freq
```

  - big CPUs have 19 frequency scaling levels
    - 200 MHz to 2.0 GHz with each step of 100 MHz
  - LITTLE CPUs have 13 frequency scaling levels
    - 200 MHz to 1.4 GHz with each step of 100 MHz

# *MultiThreadBench* Work-package

- Changing mapping and Executing

```
$taskset -c 0-7 ./benchmark
```

  - The command '`taskset -c`' set the CPU affinity for a particular application
    - Ask the Linux task scheduler to pin the application to specific CPU cores

# Sample Exercises

- Changing operating frequency and looking execution time
  - Use all cores at lowest frequency
  - Use all cores at highest frequency
  - Use all cores at some intermediate frequency

- Changing mapping and looking execution time

# Thermal Behaviour Exploration

# *RSABench* Work-package

- Move into the RSABench sub-folder

```
$cd RSABench/
```

- It contains:
  - *Freq_Temp_reading.sh* – To read core frequency and temperature
  - *read_temperature.py* – to read/analyse temperature values of cores
  - *run_rsa.sh* – A set of commands to facilitate settings and execution
  - *show_temperature.sh* – to plot and show temperature but need more tooling

# *RSABench* Work-package - Steps

- Set frequency of cores

```
$echo "900000" >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
$echo "900000" >
/sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq
```

- Start reading core frequency and temperature (type command in another terminal)

```
$./Freq_Temp_reading.sh
```

- Define mapping and execute RSA benchmark

```
$taskset -c 6 openssl speed rsa
```

# *RSABench* Work-package – Steps in a Script

```
echo "[1] Set frequency of cores ….“
echo "900000" >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
echo "900000" >
/sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq
sleep 10

echo "[2] Reading core frequency and temeprature....through
other terminal as ./Freq_Temp_reading.sh“
#mate-terminal -e ./Freq_Temp_reading.sh

echo "[3] Define mapping and execute RSA benchmark....“
taskset -c 6 openssl speed rsa

echo "[4] RSA benchmark evaluation complete....“
sleep 3

echo "[5] Stopping core frequency and temeprature reading
collection....“
echo "Stop">Signal.txt
```

# Sample Exercises

- Change mapping and look cores' temperature used by the application while keeping frequencies fixed.

- Change frequency and look cores' temperature used by the application while keeping mapping fixed.

**Before executing, make sure that you delete the *output_temp.csv* file as it is used for the final analysis.**

**You can in fact delete all the output files.**

# Questions?