The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2021

ECE 379K: Machine Learning and Data Analytics for Edge AI

HW2 Assigned: Feb 25 DUE: Mar 11 (CST 11:59:59pm)

Work in groups of two students. Only one submission per group is required.
At the end of the PDF file add a paragraph where you clearly describe each member's contribution.

Introduction to HW2

Mobile systems are cyber-physical systems subject to constraints that are not usually present in traditional, high-performance computing systems. For instance, in contrast to the cloud server used in HW1, the power and thermal constraints are more severe on mobile systems due to the limited battery capacity and passive cooling techniques available in such scenarios. At the same time, modern ML models demand more performance from mobile devices. This conflicting set of constraints and demands requires system designers to reconsider the design of both hardware and model architectures.

In this assignment, you will work with a heterogeneous multiprocessing system, namely the ODROID MC1 platform (Figure 1), while targeting concrete power/thermal and performance objectives. The ODROID MC1 uses Exynos 5422, a heterogeneous multi-processor system-on-a-chip (MPSoC); this system consists of two clusters of ARM cores (organized into a big.LITTLE architecture) and a small GPU core [1]. As discussed in Lecture 6, the "big" cluster consists of four ARM cores designed for high performance, while the "LITTLE" cluster implements four cores intended for maximum power efficiency.

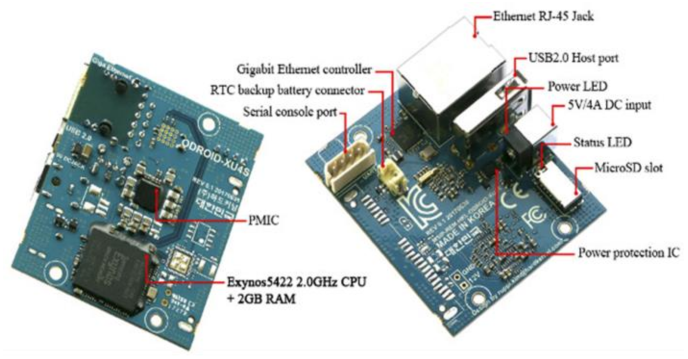


Figure 1. View of the Odroid MC1 platform

The MC1 platform includes several control "knobs" (e.g., the clock frequency of each cluster, task scheduling on individual cores, etc.) that allow us to keep the system within the specified performance, power and thermal constraints. This homework assignment targets the clock frequency scaling in the big cluster in order to control the system performance, power and thermal characteristics. To this end, you will directly experiment with running simple benchmarks while considering both the "cyber" (i.e., performance) and the "physical" (i.e., power/thermal) components of the MC1 manycore system.

The MC1 system setup

The ODROID MC1 allows precise thermal monitoring of each of the four cores in the *high-performance cluster* and the GPU core via a total of five thermal sensors spread across the chip. Additionally, the Linux `sysfs` interface allows to change the frequency of each CPU cluster, check the temperature of the big cores and GPU, and monitor the frequency values of the GPU and the two CPU clusters.

The `/etc/proc` filesystem interface allows for precise measurements of individual CPU loads. Alternatively, `psutil` in Python 2.7[1] offers functions to extract the times spent by the CPU on various tasks [2]. **See the Appendix section for details of setting up your MC1 board.**

Problem 1 [30p]: System physical and cyber characteristics

In this problem, you will explore the system physical and cyber characteristics with two types of workloads, namely a single-threaded throughput benchmark (*TPBench*) and two multi-threaded benchmarks (*blackscholes* and *bodytrack*). *TPBench* is an application that loads each individual core to 100 percent utilization with four types of instructions: floating-point multiply-accumulate, integer multiply-accumulate, floating point addition, and integer addition. The *blackscholes* and *bodytrack* (from the *PARSEC* [3] suite) are two multithreaded computational benchmarks. They are located in the *parsec_files* folder of your MC1 device. The *blackscholes* benchmark is a computational finance options pricing benchmark; *bodytrack* is an algorithm for tracking an unmarked person across several images. By testing the MC1 system with these benchmarks, you will gain experience in running simple benchmarks on this hardware platform and understand the effects of your design choices on the cyber-physical configuration.

Question 1: [12p] Keep the little cluster at 0.2GHz and the big cluster at 2GHz, then run the *TPBench* benchmark **only on core 4**. You can use `taskset` to constrain the execution of the benchmark to a single big core (see **Appendix A5**). The executable file of the benchmark is available under the 'HW2_ files' folder in the home directory of your device.

Use a sampling period of 200 milliseconds (ms) and record the following information over time:

- *Frequencies of the big and LITTLE clusters:* Use the *`sysfs`* paths provided in ***sysfs_paths.py*** to access resource voltage and frequency settings.
- *Board power*: Use *telnet* to log into power measurement device Smart Power 2 (e.g., Python's `telnetlib` [4]).
- *Individual CPU usage*: Use `psutil` [2] (or write your own workload calculation function) to get the average usage of each core. Include the I/O waiting time as the idle time for each core.
- *Temperatures of each thermal zone*: Record the temperatures of the four big cores as a function of time variation; these are the thermal zones 0, 1, 2, and 3 (zone 4 is the GPU). The paths are available in ***sysfs_paths.py***.

Use the recorded values to plot the ***system power*** [Watts], ***core usage*** [% utilization], and ***temperature*** (i.e., temperature for each big core) [°C] over time. Clearly label your plot(s) (i.e., X, Y axes, legend, and plot title). Submit your code files that you used to record the system outputs for the benchmark.

NOTE: For the MC1 chip, cores 5 and 7 have the thermal values swapped (i.e., the core 5 temperature shows the core 7 temperature and vice versa). Cores 0, 1, 2 and 3 correspond to the four LITTLE cores in the LITTLE cluster, while cores 4, 5, 6 and 7 correspond to the four big cores in the big cluster. The temperature sensors *only* monitor the four big cores; in other words, the thermal zones 0, 1, 2 and 3 correspond to the cores 4, 5, 6 and 7, respectively.

Question 2: [3p] How many phases of benchmark execution can you identify based on temperature dynamics? Do you see any difference in the temperatures range across the four big cores?

[1] Python 2.7 is recommended for this assignment; all necessary libraries to use Python 2.7 are already installed on the MC1.

Question 3: [10p] Keep the little cluster at 0.2GHz, then run the *blacksholes* and *bodytrack* benchmarks[2] on **all four big cores** with a frequency value of 2GHz. Also, set the number of threads to 4 while running the *bodytrack* benchmark.

Plot the ***system power*** [Watts], ***frequency of big cores***, and ***max big temp*** [°C] over time (*max big temp* = max (big core 4 temp, big core 5 temp, big core 6 temp, big core 7 temp)). Clearly label your plot(s) (i.e., X, Y axes, legend, and plot title).

Question 4: [5p] Complete *Table 1* with data you recorded while addressing Question 3 above[3].

Table 1

Benchmark	Run time [s]	Avg. power [W]	Avg. max temp [°C]	Max temp [°C]	Energy [J]
Blacksholes					
Bodytrack					

NOTE: Both *blacksholes* and *bodytrack* benchmarks can use all cores available on the MC1; therefore, you should use `taskset` to constrain their execution to only the big cores (see **Appendix A5**).

NOTE: The *run time* is the total time needed for the MC1 system to execute a particular benchmark. The *average power* is the mean of system power to execute a particular benchmark. The *average maximum temperature* is the mean of the *max big temp* for a benchmark. The *max temperature* is the peak value of *max big temp* for a benchmark. The *energy* used to run a benchmark is computed as the sum of all system power consumed for a benchmark × 0.2, where 0.2 is your sampling period.

IMPORTANT: Leave enough time (e.g., at least one or two minutes) between the end of one benchmark run to the start of the next benchmark run so the system thermals and loads can reach the idle steady-state.

Problem 2 [25p]: System power prediction

Use the thermal, power, voltage, and frequency data provided in the ***xu3_dataset.csv*** file to train a model (**on your computer using the Jupyter Notebook**) that can predict the *power* consumed by the *big cluster*. You are required to:

(i) Create a *classification* model to predict the states of the big cluster (i.e., cluster active or idle).

(ii) Create a separate *regression* model to predict the actual power values of the big cluster.

You may use *any* learning model (see *scikitlearn* [5], for example). An *active state* of the big cluster corresponds to a power consumption larger than 1W, while an *idle state* corresponds to a power consumption less than 1W.

Evaluate the accuracy of your models using the test datasets from the ***xu3_blackscholes.csv*** and ***xu3_bodytrack.csv*** files. Remember, you are not allowed to train your model using the test datasets.

Question 1: [20p] Fill in the entries of *Table 2* below based on the accuracy of your classification and mean-squared error (MSE) of your regression, using your training and test sets.

[2] To run the *blacksholes* and *bodytrack* benchmarks, refer to the README file in the 'HW2_ files' folder in the home directory of the provided user accounts.

[3] Make sure to check your implementation carefully, as your results in this problem will be used later in Problem 4.

You will get full marks for the classification if you can design a classifier that can achieve over 98% test accuracy. You will get full marks for the regression if you can design a regressor that can obtain a test MSE value less than 0.15.

In addition to the results below, **submit your Jupyter Notebook file** that prints the outcomes of executing your code. The outcomes that you need to print are: the training and test data shape, the features you used to train your models, and the training and test results. Your results in *Table 2* will also be verified with your code.

HINT: You may need to try and select the most useful features in order to improve the prediction results.

Table 2

Benchmark	Training accuracy [%]	Test accuracy [%]	Training MSE	Test MSE
Blacksholes				
Bodytrack				

Question 2: [5p] Visualize the confusion matrix for your classification results (only the test results are required) on the big cluster state. You can use any visualization tool, e.g., the *confusion_matrix* method provided by the *scikitlearn* library in Python.

Problem 3 [20p]: System temperature prediction

Use the thermal data of the big cores (i.e., cores 4, 5, 6 and 7) provided in the ***xu3_dataset.csv*** file to train a model (**on your computer using the Jupyter Notebook**) that can predict the *temperature* values for each of the big cores for the next time step.

Use the *MLPRegressor* model provided by the *scikitlearn* [5] Python library. Specifically, use the following parameters for your regressor: hidden_layer_sizes = (100, 60, 30), activation = 'relu', random_state = 0. You do NOT need to apply any regularization term. Evaluate the performance of your model using the test datasets from the ***xu3_blackscholes.csv*** and ***xu3_bodytrack.csv*** files.

Question 1: [10p] Fill in the entries of *Table 3* below based on your mean-squared error (MSE) results. In addition to the results below, **submit your Jupyter Notebook file** that prints the outcomes of executing your code.

Table 3

Benchmark	Test MSE (Core 4)	Test MSE (Core 5)	Test MSE (Core 6)	Test MSE (Core 7)
Blacksholes				
Bodytrack				

Question 2: [5p] Visualize your temperature predictions against the actual (recorded) values from the data, for the big core 4. Draw a separate plot for each of the ***xu3_blackscholes.csv*** and ***xu3_bodytrack.csv*** test datasets. In both plots, clearly label your X, Y axes; add a legend and a plot title.

Question 3: [5p] What other techniques can be used to further improve the performance of your regressor? List at least three such techniques.

Problem 4 [25p]: System thermal and CPU usage optimization

Build an *Ondemand* governor with a CPU usage threshold of 80% and a thermal limit threshold of 70°C in place. (The standard Linux *Ondemand* governor [6] also uses a CPU usage threshold of 80%). Set the proportional and integral terms (P and I, respectively) as 0.1.

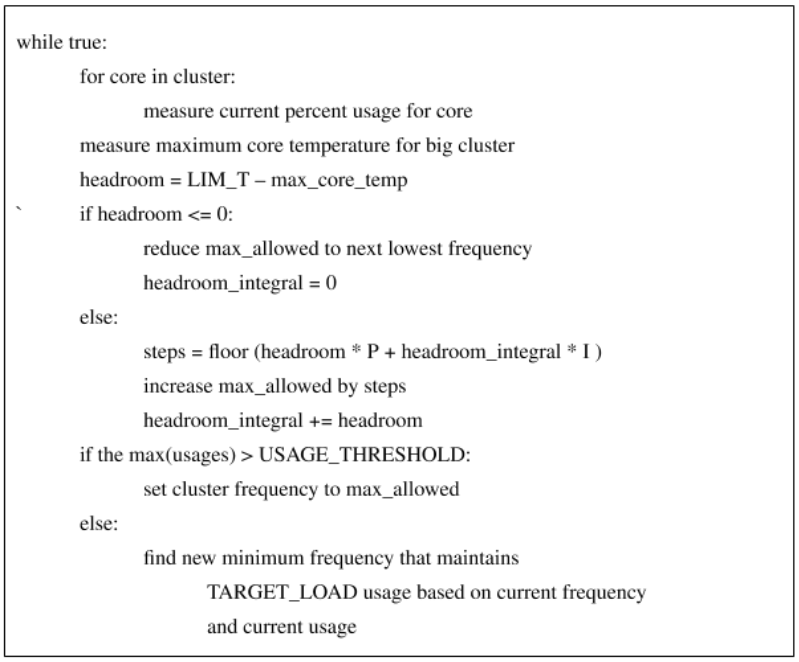
If the thermal threshold is not violated, update the maximum allowable frequency using proportional and integral gains (pseudocode given in the Figure 2 below), on the thermal headroom available. Run the *blacksholes* and *bodytrack* benchmarks across the four big cores at 2 GHz. The LITTLE cluster should remain at 200 MHz for all these runs.

Question 1: [20p] Complete *Table 4* below and compare your results against the results you got in Problem 2. How well does your governor comply with the imposed thermal limit? You need to submit the code file of your governor as well. Add necessary comment lines to your code so that it can be readable.

Table 4

Benchmark	Runtime [s]	Avg power [W]	Avg max temp [°C]	Max temp [°C]	Energy [J]
Blacksholes					
Bodytrack					

Question 2: [5p] What is the cyber-physical trade-off of implementing the proposed governor? Discuss such trade-offs by comparing the runtime, power consumption and thermal limits of each program you obtained in this problem against what you got in Problem 1 (Question 4).



```
while true:
    for core in cluster:
        measure current percent usage for core
    measure maximum core temperature for big cluster
    headroom = LIM_T – max_core_temp
`   if headroom <= 0:
        reduce max_allowed to next lowest frequency
        headroom_integral = 0
    else:
        steps = floor (headroom * P + headroom_integral * I )
        increase max_allowed by steps
        headroom_integral += headroom
    if the max(usages) > USAGE_THRESHOLD:
        set cluster frequency to max_allowed
    else:
        find new minimum frequency that maintains
            TARGET_LOAD usage based on current frequency
            and current usage
```

Figure 2. Ondemand Thermal Algorithm with Proportional-Integral (PI) Gains

Instructions/Hints

1. Include your solutions into a *single* zip file named **<GroupNo_FirstName1_LastName1_FirstName2_LastName2>.zip**. In the zip file, you should have:
   - A single PDF file containing all your results and discussions.
   - A README.txt file listing the purpose of all items in the zip file.
   - Your code files.

   NOTE: The group number (**GroupNo**) assigned for HW1 remains the *same* throughout the semester (i.e., for all homeworks and for the project as well).

2. **Verify your implementation as you proceed.**
3. Before you begin this assignment, read the entire description carefully to make sure you have all the tools needed to solve the problems.
4. **Start early!** This homework may take longer than you expect to complete.
5. If you fail to connect to your MC1 board, contact the TAs as soon as possible so they can help you fix the issue. Mention the Team you are a part of for faster identification of the device.

Good luck!

Appendix

A1. Connecting to the MC1 system over SSH

By now, you should have received a username, password, and IP address to remotely log into your assigned MC1 board. Each team of two students will have their own MC1 board and account to allow parallel code development.

To connect to an MC1 device from your computer, first make sure you are on the UT network or connected to it via VPN[4]. Then, open a terminal (such as bash) or a program such as Putty or Powershell and use the command:

```
ssh odroid@<IP address of your MC1>
```

The passwords are: `12345678`.

A2. Power logging over Telnet and Serial on MC1

To gather power readings, you need to access a Smart Power 2 (SP2) device for power monitoring to remotely measure the power of the entire MC1 platform. You will access this data from the SP2 over a telnet connection [4]. The MC1 boards are configured to automatically connect to the SP2 wireless link. You can use telnet to connect to the SP2 board over its Wi-Fi connection at the local IP address:

```
telnet 192.168.4.1
```

The SP2 *output format* is '<VOLTS>, <AMPS>, <WATTS>,<WATTS per HOUR>'. For Python development, you can use the *telnetlib* library instead.

A3. Setting the frequency of the MC1 board

To change the cores frequency, you need to first set the mode of *scaling_governor* for a given cluster to *'userspace'* before you can set the frequency using *scaling_setspeed*. Otherwise, you will get an error.

Additionally, the frequency values you set must belong to the list of available frequencies. Any frequency change on one core in a cluster will affect all the cores in that cluster. To reset the system frequency, you need to run your code as root (e.g., use `sudo`).

A4. Setting/reading frequencies and reading thermal and voltage cluster values

To help you better set/read the system parameters, the ***sysfs_paths.py*** file provided in HW2_files need to be loaded as a module in your Python data logging program. The variables in this file contain the partially defined paths needed for setting and reading frequency, voltage, and temperature on the MC1.

Paths whose variable names begin with '`fn_`' contain '{}' characters. Use `string.format()` to insert the core number into the path, e.g., to set/read the governor of the big cluster, you can write/read to/from the file path contained in the formatted string: `'fn_cluster_gov.format(4)'`, which will refer to: '/sys/devices/system/cpu/cpufreq/policy4/scaling_governor'.

Be sure to read the comments at the top of ***sysfs_paths.py*** and/or check the Linux *`sysfs`* documentation for more information.

[4] For information on connecting to the UT VPN, please follow the steps described in the UT Wikis.

A5. Core affinity for multithreaded applications

Some benchmarks in this assignment may use all cores available on the MC1 by default; when required, you should use `taskset` to constrain their execution to the big cores.

Each process and thread in Linux has a *core affinity* – the set of processors on which that task can be scheduled. The CPU affinity can be set in-program using libraries or at runtime by using the command: ‘`taskset --all-tasks <hexadecimal core mask> <command>`’.

Note that cores in the system are numbered from 0 to 7. Cores 0 through 3 are LITTLE cores and cores 4 through 7 are big cores. For example:

```
taskset --all-tasks 0x3 ./benchmark.out
```

will run a benchmark on cores zero and one, which are the first two cores in the LITTLE cluster. Using a mask of 0xAA will run a benchmark on cores 1, 3, 5, and 7. Figure 3 shows how two example hexadecimal bitmasks map to the cores in each cluster.

You can use `htop` to visualize resource utilization of each core on the MC1 as you run the benchmarks. By default, `htop` shows threads of a program as separate entries.

NOTE: The Odroid MCI uses the Exynos 5422, which can run all cores at the same time. The older version of the Exynos MPSoC (i.e., 5410) can run only one cluster at a time, i.e., either the little or the big cluster.

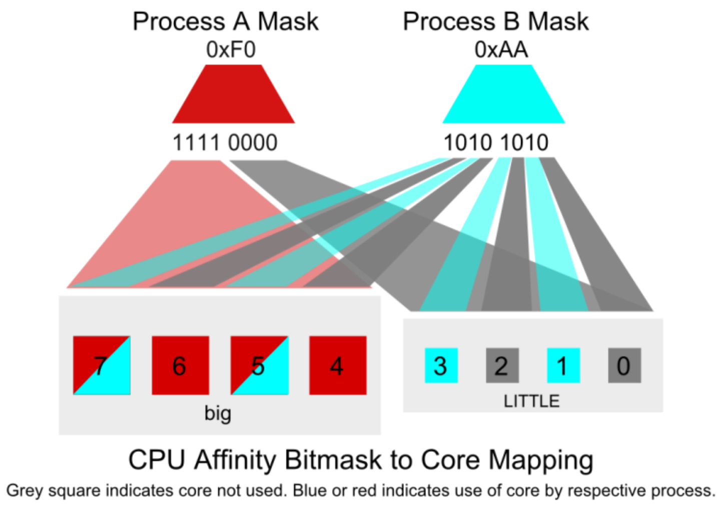

Process A Mask
0xF0
1111 0000
Process B Mask
0xAA
1010 1010
7
6
5
4
big
3
2
1
0
LITTLE
CPU Affinity Bitmask to Core Mapping
Grey square indicates core not used. Blue or red indicates use of core by respective process.

Figure 3. Diagram of two CPU affinity bitmask to core mappings

A6. Files transfer between computers

To transfer files between your local computer and the MC1, you can use the `scp` command. The general format is:

```
scp <sender address>:<path to send the file> <receiver address>:<path to receive the file>
```

For example, to transfer a file from your local computer to the MC1, you can open a terminal on your computer and type:

```
scp <local path to send the file> odroid@<Your MC1 address>:<MC1 path to receive
the file>
```

Similarly, if you want to receive a file from the MC1 board, open a terminal on your computer and type:

```
scp odroid@<Your MC1 address>:<MC1 path to send the file> <local path to receive
the file>
```

References

[1]	Samsung, "Samsung Exynos 5422," 2021. [Online]. Available: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/. [Accessed 9 January 2021].
[2]	G. Rodola, 24 October 2017. [Online]. Available: https://psutil.readthedocs.io/en/release-3.4.2/#quick-links.
[3]	Bienia, Christian, et al. "The PARSEC benchmark suite: Characterization and architectural implications." Proceedings of the 17th international conference on Parallel architectures and compilation techniques. 2008.
[4]	"Python Standard Library Documentation: telnetlib — Telnet client," Python Software Foundation, 13 December 2017. [Online]. Available: https://docs.python.org/2/library/telnetlib.html. [Accessed 9 January 2021].
[5]	Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine Learning research 12 (2011): 2825-2830.
[6]	V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in Proceedings of the Linux Symposium, Volume Two, Ottawa, Ontario, 2006.