

# COEN-4730 Computer Architecture

## HW #6

Dept. of Electrical and Computer Engineering, Marquette University  
*Cristinel Ababei*

### 1. Objective

Learn about developing parallel applications on CMPs. Verify these applications via simulations with GEM5 full system simulator and McPAT power calculator.

### 2. Background

The state-of-the-art CPUs contain dozens of cores on a single die. These new architectures present new challenges to both computer architects and programmers. In this assignment, we will try to tackle some of these challenges. Having now an understanding of mainstream multi-core CPU architectures, the purpose of this assignment is to learn how to develop parallel applications on such architectures, and how to analyze the performance in a real environment.

In this assignment you will map a C program onto a multiprocessor system. With the help of GEM5 full system simulator and McPAT power modeling framework, we will look at different configurations (e.g., the number of processors, block-size and associativity of different levels of caches, etc.). The goal is to optimize the Energy-Delay-Area-Product (EDAP) of the system. You can achieve this goal *by improving the original C code, using OpenMP, and/or using any other creative methods.*

We use OpenMP to create a parallel application. We use GEM5 simulator and McPAT power modeling framework to explore the effect of different architecture configurations.

#### OpenMP

OpenMP (Open Multiprocessing, <http://openmp.org>) is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems. In C and C++, the programmer can annotate source code with *pragmas* to tell the compiler how to parallelize the application. For example, compile the following code with a compiler that supports OpenMP:

```
int main(int argc, char* argv[]) {
    const int N = 100000;
    int i, a[N];
    #pragma omp parallel for
    for (i = 0; i < N; i++)
        a[i] = 2 * i;

    return 0;
}
```

The **for** loop will be executed in multiple threads on a multicore platform. By default, OpenMP uses the same number of threads as the number of cores in the system.

Below are several links to materials for learning more about OpenMP:

--OpenMP resources webpage:

<https://www.openmp.org/resources/>

--"An Overview of OpenMP" by Ruud van der Pas:

<https://www.openmp.org/wp-content/uploads/ntu-vanderpas.pdf>

--"Hands-On Introduction to OpenMP" with Code Exercises:

<https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf>

[https://www.openmp.org/wp-content/uploads/OMP\\_Exercises.zip](https://www.openmp.org/wp-content/uploads/OMP_Exercises.zip)

--Wikipedia:

<https://en.wikipedia.org/wiki/OpenMP>

***Note:** OpenMP is not (fully) automatic parallelization! Though in OpenMP the compiler automates some part of the parallelization process, one important aspect is that the compiler does not check (and certainly not prove) the correctness of the parallelized application. So, it is pretty much up to the programmer to annotate the code properly.*

### 3. Environment Setup

#### **GEM5 and McPAT**

You should have the tools already installed from the work done in the previous assignments.

#### **ALPHA**

You should also have almost all you needed for working on the ALPHA portion of this assignment.

**Cross-Compiler ALPHA:** The only thing that you need more is an ALPHA pre-compiled cross-compiler. It is included in the files provided for this assignment: **alphaev67-unknown-linux-gnu.tar.bz2** (NOTE: it is "gcc-4.3.2, glibc-2.6.1 (NPTL,x86/64)" from gem5 downloads website).

Place it inside a new folder that you should create inside gem5/, then, extract it:

```
>mkdir cross_compilers
```

```
>cd cross_compilers
```

Copy the provided file here, then extract it:

```
>tar -xjvf alphaev67-unknown-linux-gnu.tar.bz2
```

And also set permissions:

```
>chmod 777 *
```

#### **ARM**

For the ARM portion of this assignment, you need to download and unzip an appropriate disk image and binary. More specifically, we will use **aarch-system-2014-10.tar.xz**, which is included with the files provided for this assignment (should be available also at:

[https://www.gem5.org/documentation/general\\_docs/fullsystem/guest\\_binaries](https://www.gem5.org/documentation/general_docs/fullsystem/guest_binaries)).

Unzip it inside your **full\_system\_images\_ARM/** folder inside the gem5 folder.

```
> cd full_system_images_ARM
```

```
> tar xvf aarch-system-2014-10.tar.xz
```

You also need to compile an **ARM version** of the gem5 simulator.

First, add the M5\_PATH environment variable:

```
> cd
```

```
> emacs .bashrc
```

and add the following line to it:

```
export M5_PATH=/home/cristinel/hw4/gem5/full_system_images_ARM  
> source .bashrc
```

Then, let's build the gem5 simulator in FS mode with ARM support, optimizations turned on:

```
> cd /home/cristinel/hw4/gem5  
> scons FULL_SYSTEM=1 build/ARM/gem5.opt  
It may take a few good minutes to finish; so, be patient.
```

**Cross-Compiler ARM:** We need to install a proper cross compiler for the ARM architecture. So, do in a terminal:

```
> sudo apt-get install gcc-arm-linux-gnueabi  
If the above gives you some errors related to unmet dependencies, do the following steps:  
>sudo apt-get remove binutils-common  
>sudo apt-get install binutils-common  
>sudo apt-get install binutils-arm-linux-gnueabi  
>sudo apt-get install gcc-7-arm-linux-gnueabi  
>sudo apt-get install gcc-arm-linux-gnueabi
```

#### 4. Assignment

You are required to map an application onto a multicore processor with 4 cores for two different ISAs: ARM and ALPHA, and simulate using GEM5. The goal is to optimize the Energy-Delay-Area-Product (EDAP) for the given application. The application used in this assignment is described in the next Section. All the source files required are provided in the archive available on D2L. Place a copy of the provided files in **gem5/benchmarks/**.

Then, spend some time to:

- 1) Read about OpenMP first.
- 2) Study the source code (all files \*.c and \*.h) of the given application.
- 3) For each ISA (ARM and ALPHA), keep number of cores as 4, and the best architecture you found in the previous HW#5, iterate:
  - a) Modify the source code of the given application in order to parallelize it using OpenMP. Use your creativity to change it in order to exploit as much parallelism as possible. Your goal is to make the application run faster on the architecture.
  - b) Compile and then run the application with GEM5. Convert output of GEM5 into what McPAT needs using the provided script. Run McPAT. Compute EDAP and compare to the EDAP you got in the previous iteration. Your goal is to make the OpenMP specific changes such that the application runs faster and therefore EDAP becomes smaller.

Details about the above steps of each iteration are provided in “Getting and running the application” from the next section.

**Read Makefile to see how everything is accomplished. Cris will go in class over this too.**

#### 5. Given Application

To make the assignment more interesting and practical, we'll analyze the image-processing kernels from an industrial application: Organic-Light-Emitting-Diode (OLED) Printing.

### OLED Printing Application

The left side of Fig. 1 shows an OLED sheet. After zooming into the details, we can find a couple of repetitive structures (OLEDs), which are aligned side by side. During fabrication, organic materials are injected into these OLED "bowls". In order to improve the quality and yield, accuracy of the detected center of each OLED and the processing time are very important.

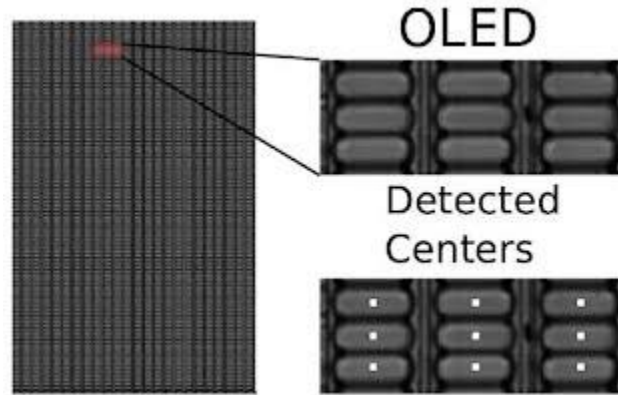


Fig. 1: Center-Localization of OLED-Printing Application

A vision pipeline to accurately localize the OLED centers is developed, which is shown in Fig. 2. A brief description of each kernel is given below.

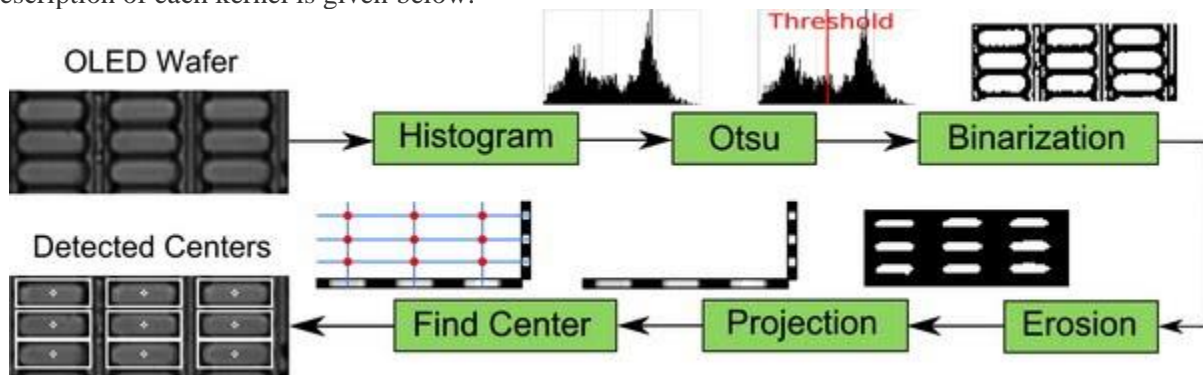


Fig. 2: Vision Pipeline for OLED Center Localization

1. After profiling the histogram of the input image, Otsu algorithm ([http://en.wikipedia.org/wiki/Otsu%27s\\_method](http://en.wikipedia.org/wiki/Otsu%27s_method)) is applied to find out the optimal threshold for binarization.
2. The input grey-level image is then binarized with this optimal threshold.
3. After binarization, erosion ([http://en.wikipedia.org/wiki/Erosion\\_%28morphology%29](http://en.wikipedia.org/wiki/Erosion_%28morphology%29)) step is done a couple of times to remove the irrelevant patterns (noise). The iteration number of the erosion step is dependent on the quality of the input image. In this assignment, only 1 iteration is used, and the erosion kernel is a 5x5 square.
4. The erosion step removes most of the noise. Now, we count the number of OLED pixels in each row (row projection) and the number of pixels in each column (column projection). A column vector and a row vector are built in this step. The purpose of row/column projection is to find out the possible OLED positions in each direction.

5. Then, we apply a threshold on the values of the row vector and column vector. What remains in the row/column vectors after this thresholding is a couple of line segments. The centers of each line segment are the rough centers of the OLEDs: the row vector provides the y positions and the column vector provides the x positions.
6. Finally, a bonding box around each center is drawn onto each OLED.

To simplify the assignment, we only focus on the mapping of **first four** kernels, i.e., Histogram, Otsu, Binarization, and Erosion. The C code you received also only contains these four kernels.

### Getting and Running the Application Code

The source code of the application is included in the archive given to you as part of this assignment. You are given two folders, one for ARM and one for ALPHA architectures.

#### First, to do experiments for ARM ISA

You must cd into and work in the corresponding provided folder.

```
>cd gem5/benchmarks/ffos_ARM/
```

Edit the file: **gem5/configs/common/SysPaths.py**

And replace:

```
path = [ '/dist/m5/system', '/home/guoqi/simulators/gem5/dist/m5/system' ]
```

```
path = [ '/dist/m5/system', '/home/cristinel/hw4/gem5/full_system_images_ALPHA' ]
```

With:

```
path = [ '/dist/m5/system', '/home/cristinel/hw4/gem5/full_system_images_ARM' ]
```

Then edit your **.bashrc** too:

```
>emacs ~/.bashrc
```

And add these lines at the end:

```
#export M5_PATH=/home/cristinel/hw4/gem5/full_system_images_ALPHA
```

```
export M5_PATH=/home/cristinel/hw4/gem5/full_system_images_ARM
```

Save the file and then source it:

```
>source ~/.bashrc
```

At this time, read the README file from ffos\_ARM/.

Then, edit to change **Makefile**, **scr\_cp2dm**, and **fs\_modified.py** according to your system.

If you only want to compile the application just type:

```
> make all
```

This should create the **ffos** executable. You can do it to just see the application being compiled; we will do more than this.

Note: In case that when running the Makefile, you face permission problems, you can do:

```
> chmod 777 *
```

inside the folder ffos\_ARM/.

The way we'll work is actually to run a script to do all the steps automatically for us. We compile and also run the GEM5 simulation, and also McPAT, for an architecture with four cores (default is one core) type in a terminal:

```
> sudo make NUM_CPU=4 run
```

Again, you **should read Makefile** to see what the above does. It basically does the following steps:

1. Compile the application if the executable is outdated.
2. Copy the executable **ffos** to disk image in preparation for full system simulation.
3. Run GEM5 full system simulation (takes a few minutes, depending on the system configuration and simulation parameters).
4. Convert GEM5 output (in *m5out/*) to an XML file (*power.xml*) that is compatible with McPAT.
5. Run McPAT and copy its output into some file, say *mcpat\_out.txt*.
6. The McPAT output *mcpat\_out.txt* contains the area and power figures of the simulated platform.

### Second, to do experiments for ALPHA ISA

You must cd into and work in the corresponding provided folder.

```
>cd gem5/benchmarks/ffos_ALPHA/
```

And essentially do the same steps that you did for the ARM architecture, but replace ARM with ALPHA.

### Verify the Correctness of Your Result

After parallelizing the application, you would like to make sure that your new source code still provides the correct functionality. To verify the correctness, please check whether or not the threshold value for binarization is still the same as the value that you get when you execute the serial version. The threshold value is printed out (see *printf("Threshold = %d\n", t)* in the source code). You can find it in the simulation output.

## 7. Deliverables

**Part one:** A single PDF report, uploaded on D2L, which shall include the following:

- 1) **Title** and your name.
- 2) **Abstract:** Describe briefly this assignment; use your own words (do not copy from this document). This should be done within one paragraph of several sentences.
- 3) **Discussion of changes to application:** Description of how you changed the application code, including OpenMP and/or other modifications.
- 4) **Discussion of results:** Discuss what platform configurations you have explored and what the results are. This should be done for both ARM and ALPHA architectures. Interpretation of the results with a comparison of the two architectures.
- 5) **Conclusion:** In one paragraph, discuss what are your conclusion(s) about your investigations and lessons you learned.
- 6) **References:** List any website, paper or technical report, articles that you used. Cite them appropriately throughout your presentation.

**Part two:** You must upload on D2L also an archive file containing your source code with your changes and the simulation scripts you used.

## 8. Credits and Thanks

This was adapted from the teaching materials of Prof. Henk Corporaal.