Arquiteturas de Alto Desempenho 2024/2025

Practical class AAD_P01 (2024-09-23 and 2024-09-24)

Tomás Oliveira e Silva

1 Objectives

Measure the performance (execution time) of a simple program that does some floating point computations

- across several programming languages, and
- for different processors (collective effort).

2 Work to be done

Do the following:

1. Create a directory (folder) for this course and a sub-directory to store all files for this specific practical class. For easy access, perhaps place the top folder in the desktop. Suggested folder names: `AAD_2024` and `AAD_2024/P01`. The documentation below will use these names and recommendations. For example, on a GNU/Linux system, we can do (on a terminal, running `bash`):

```
$ mkdir -p ~/Desktop/AAD_2024/P01
```

2. Download the archive `AAD_P01.tgz` from the elearning website.
3. Unpack its contents into the `P01` directory:

```
$ cd ~/Desktop/AAD_2024/P01
$ tar xzvf ~/Downloads/AAD_P01.tgz
```

4. If needed, install a C compiler, a Java SDK, and Python 3. Optionally you may also install a Fortran compiler. On a GNU/Linux system, more specifically `ubuntu`, we can do (on a terminal):

```
$ sudo apt install build-essential python3 default-jre default-jdk gfortran clang
$ gcc --version
$ python3 --version
$ java --version
$ javac --version
$ gfortran --version
# clang --version
```

   For Windows users, this should also work if you have the Windows Subsystem for Linux (WSL) installed.

5. Examine the source code of the various `MandelbrotCount` programs; there is one for `C`, another for `Java`, another for `Python3`, and another for `Fortran90`.
6. Examine the contents of the file `makefile`. It is used by the `make` program to automate all compilation tasks. You may also use an Integrated Development Environment (IDE), such as Visual Studio Code, to compile stuff, but it that case you **are on your own**; the teachers of this course will **not solve** any problems you may encounter while using one of them.
7. Compile and run the programs. All that is necessary to do this is already in the `makefile`. Just do:

```
$ make run_all
```

   This will compile all programs (if needed) and, afterwards, will run them all.

8. Actually, run the programs at least three times and write down the best execution time of each. Place your results on the following shared spreadsheet (clickable link).

   Spreadsheet URL: https://docs.google.com/spreadsheets/d/1O_FjxsIhLGouCo5uHsXFxNpDQRN82katzKNTuo6FuCY/edit?usp=sharing

   Each student should do this, using time measurements collected from his, or hers, computer.
9. (To be done later at home.) If you are proficient in another programming language, translate the program to that language and run it. Record the execution time in the shared spreadsheet. Is it fast?
10. If you are curious about the speed of your mobile phone, you may also run the program on a browser on your phone. All you have to do is load the following web page into the browser (clickable link).

    Web page URL: https://sweet.ua.pt/tos/AAD/P01/MandelbrotCount.html

    Please record the execution time in the shared spreadsheet. Is your mobile phone fast?
11. The Python code is kind of slow. Try to make it faster. If you have a recent version of Python you may use the so-called walrus operator `:=` to store in temporary variables expressions that will be recomputed later on). Does that improve things?
12. How about converting the python code to bytecode? Does it improve things? To convert, do this in a Python3 interpreter:

    ```
    >>> import py_compile
    >>> py_compile.compile("MandelbrotCount.py","MandelbrotCount.pyc",None,False,2)
    ```

    and run the code as follows:

    ```
    $ python3 MandelbrotCount.pyc
    ```

13. Finally, how about the speed of a compiled version of the Python code? For example, you can try the codon compiler tool. To install it, do

    ```
    $ curl -fsSL https://exaloop.io/install.sh >install.bash
    ```

    edit the install script to change the installation directory to your liking (say, `/opt/codon`), run the installer, and try it:

    ```
    $ bash install.bash
    $ export PATH=/opt/codon/bin:$PATH
    $ codon build -release MandelbrotCount.py
    $ time ./MandelbrotCount
    ```

    Comment out the `time.process_time()` calls, as they are not supported by codon. Warning: integers become 64-bit signed quantities; other deviations from standard Python likely exist.
14. Which programming language is best suited for High Performance Computing?
15. Clean up. Do:

    ```
    $ make clean
    ```

3 If you want to do more (Web Assembly)

C programs that only use standard libraries can usually be compiled into Web Assembly. To simplify running the code on any browser that supports Web Assembly, it is usually also necessary to provide a HTML page and some JavaScript glue code. Fortunately, there is a tool to do all this: `emscripten`. If you want to try it out, do the following:

1. Go to the `emscripten` web site (clickable link)

   URL: https://emscripten.org/index.html

   and read and follow the installation instructions.

2. After that, before using the Web Assembly compiler, called `emcc`, you will need to load some environment variables into your `bash` shell. The following assumes that the `emscripten` tool was installed at `/opt/emsdk`):

```
$ source /opt/emsdk/emsdk_env.sh
```

   Warning: the default shell on macOS is not `bash`; take appropriate action (switch to `bash`).

3. Everything is now ready. Use the `emcc` compiler to compile the C program. No changes to the source code are necessary. The `makefile` is your friend here, because the teacher has already placed in it what has to be done. Just do:

```
$ make WebAssembly
```

4. If all went well you should now have an HTML file, a JavaScript file and a Web Assembly file in your working directory.

5. To run the program in a local browser you can try `emrun` (take care, read the documentation first because you will need to pass the `--emrun` command line option to the `emcc` compiler). As an alternative, at least on GNU/Linux, you can launch a local web server as follows:

```
$ busybox httpd -p 127.0.0.1:8080 -h ~/Desktop/AAD_2024/P01
```

   Then, open a browser window and enter the URL

```
http://127.0.0.1:8080/MandelbrotCount.html
```

6. If you have your own web site you may also upload the three files, and give your browser the complete URL you have chosen for `MandelbrotCount.html`.

7. Finally, to clean up, do:

```
$ killall busybox
$ make clean
```

   (This assumes that you are not using `busybox` for something else.)

4 Code to be optimized in the second practical class

The code used in section 1 generates a so-called fractal — the famous Mandelbrot set — but it only counts the number of points of a square grid that satisfy a given condition. In the file `MandelbrotImage.c` you will find a simple modification of that code that actually produces a gray image. We will try to speedup that code in the next few practical classes.

The image is generated in the so-called plain Portable Gray Map format (PGM, clickable link).

PGM format description URL: `https://netpbm.sourceforge.net/doc/pgm.html`

That format is trivial to use, but since it produces large files, it can, and should, be converted later on a more frugal image format. All that is already done in the `makefile`. Just do

```
$ make MandelbrotSet.png
```

Be aware that the `makefile` uses a program named `convert` that may not be available in your system. It is usually installed by default on GNU/Linux systems. As usual, to clean up, i.e., to delete files than can be easily regenerated, do:

```
$ make clean
```