Arquiteturas de Alto Desempenho 2024/2025
Complementos sobre Arquiteturas de Computadores 2024/2025

First practical assignment — mining DETI coins

Tomás Oliveira e Silva

1 Introduction

A DETI coin is a file with exactly 52 bytes whose MD5 message-digest[1], when printed in hexadecimal, ends with at least 8 hexadecimal zeros (i.e., its last 32 bits are all 0). The file contents **must** begin with "`DETI coin `" (note the space at the end) and must end with a newline ('\n' in C). The other bytes may have arbitrary values, but it is strongly recommended that these other bytes encode utf-8 text.

Let n be the number of zeros bits at the end of the printed MD5 message-digest; that is its "power". As stated before, for a DETI coin we must have $n \geqslant 32$. Given that the probability of finding a DETI coin of power n using a file with random content is 1 in 2^n, a power n DETI coin is worth 2^{n-32} power 32 DETI coins.

For example, the MD5 message-digest of the file `deti_coin_example.txt`

```
$ od -Ad -c -t x1 deti_coin_example.txt
0000000   D   E   T   I       c   o   i   n           n   u   m   b   e
         44  45  54  49  20  63  6f  69  6e  20  20  6e  75  6d  62  65
0000016   r       o   f       t   h   e       a   t   t   e   m   p   t
         72  20  6f  66  20  74  68  65  20  61  74  74  65  6d  70  74
0000032           [   0   0   0   0   0   0   5   3   3   8   2   8   5
         20  20  5b  30  30  30  30  30  30  35  33  33  38  32  38  35
0000048   7   1   ]  \n
         37  31  5d  0a
0000052
```

is

```
$ md5sum deti_coin_example.txt
c0b9d8c5d2c98be1296bc13500000000  deti_coin_example.txt
```

so it is a power 32 DETI coin (the MD5 message-digest ends with `500000000` and 5 has 0 consecutive low order zero bits).

2 What is provided

The following files contain a reference implementation of the assignment. Study them in detail!

`deti_coin_example.txt` — The example given above of a DETI coin. It was computed with a custom version (in `deti_coins_cpu_special_search.h`, not provided!) of `deti_coins_cpu_search.h`.

`deti_coins.c` — The main program. It includes most of the code from other files. Turning a program feature on or off is then done by simply including or not the code for that feature in this file and testing if a C preprocessor macro is defined or not.

`cpu_utilities.h` — Code used by other parts of the program (reversing the order of the bytes in an integer, time measurements, …).

[1] See the Request for Comments 1321 for more details.

`md5.h` — The implementation of the MD5 message-digest for a message with exactly 52 bytes. This is done as a customizable C preprocessor macro so that it can be adapted as needed. It can be used for the CPU code, with or without using SIMD instructions, and for the CUDA code. This is used in several places, so study it with extra care.

`md5_test_data.h` — Generate random data to test the MD5 message-digest implementation.

`md5_cpu.h` — Reference implementation of the MD5 message-digest computation on the CPU (no SIMD instructions), and associated verification code.

`md5_cpu_avx.h` Reference implementation of the MD5 message-digest computation on the CPU using AVX instructions (for Intel and AMD processors), and associated verification code.

`md5_cpu_neon.h` — Reference implementation of the MD5 message-digest computation on the CPU using NEON instructions (for Arm processors), and associated verification code.

`cuda_driver_api_utilities.h` — Code to initialize a CUDA device, load a kernal function, allocate memory, and to run the kernel. It uses the CUDA driver API. It should be enough for the needs of this assignment.

`md5_cuda_kernel.cu` — Device code of the reference implementation of the MD5 message-digest computation using CUDA. Do **not use** this code as is when searching for DETI coins in the graphics card. Instead, modify it (and place the modified code in `deti_coins_cuda_kernel_search.cu`), so that each CUDA "thread" receives only the template of the DETI coin, uses its coordinates to modify the template in an unique way, makes one or more attempts to find a DETI coin, and stores any DETI coin found in a commom memory buffer (use atomic instructions to modify that common area). With all these modifications there is very little movement of data between the processor and the graphics card, and that is important to make searching for DETI coins as fast as possible.

`md5_cuda.h` CPU code of the reference implementation of the MD5 message-digest computation using CUDA, and associated verification code.

`deti_coins_vault.h` — Code to save in a buffer and latter on save to disk all DETI coins found in the search. For convenience, all DETI coins are stored in the same file.

`deti_coins_cpu_search.h` — Reference implementation of the search for DETI coins using one CPU thread, without SIMD instructions.

`test_vault.bash` — Bash script to test the file where the DETI coins are saved.

`makefile` — Makefile for the assignment. As you add your code (in `.h` files), it will be necessary to modify it.

3 What is to be done

Each group should **try** doing as much as possible of the following:

1. Convert the AVX code to AVX2 code. (Should be twice as fast!)
2. Write search functions that use the AVX code, AVX2 code, and, if you have hardware for it, NEON code. (The code for AVX and NEON should be almost identical…)
3. Write a search function that uses CUDA.
4. Do it with SIMD instructions and OpenMP.
5. Do it using a server and many clients.
6. Do it using Web Assembly.
7. How about an OpenCL implementation?

8. For each of the above, measure how many attempts (MD5 message-digest computations) you were able to do in **one hour**. (The teacher's code was able to make close to $\mathbf{5.6 \times 10^{13}}$ attempts in one hour on a GTX 1660 Ti graphics card; can you do close to that or even better?)
9. Compare the performance, i.e., the numbers of attempts per hour, of as many computing devices as possible (recent and old computers, micro-controllers, anything where you were able to make the program run).
10. Search for DETI coins with a special form (say, with part of your name embedded in it).
11. Extra stuff. For example, how about computing an histogram of
    (a) the values returned by `deti_coin_power()` during a CPU search, or
    (b) the wall time of each call to the CUDA search kernel?

4 Deliverable

One archive (either a compressed TAR archive or a ZIP file, no RARs or other archive file formats), containing

- the report (must be a PDF file), and
- all the source code (no executable files, please).

The archive name should be of the form `NNNNNN_MMMMMM.extension` where `NNNNNN` and `MMMMMM` are student numbers (use as many as needed) and `extension` is the file extension (`tgz` or `zip`).