Arquiteturas de Alto Desempenho 2025/2026

First practical assignment — mining DETI coins

Tomás Oliveira e Silva

1 Introduction

A 2025 DETI coin is a file with exactly 55 bytes whose SHA1 secure hash[1], when printed in hexadecimal, starts with `aad20250` (four bytes). The file contents must begin with `DETI coin 2 ` . That's 12 bytes (note the space at the end). The file must end with a newline character (`'\n'` in C) and this special character cannot appear elsewhere. The other bytes may have almost arbitrary values (`'\n'` is the only value that is forbidden). It is strongly recommended that these other bytes encode ASCII, or, at worst, UTF-8 text. You may use the backspace character, `'\b'`, to hide text that you do not want to see when the coin is sent to a text terminal.

Let n be the number of zeros bits after the mandatory `aad20250` signature. That is the coin's value. Given that the probability of finding a 2025 DETI coin of value v using a file with random content is 1 in 2^{32+v}, a 2025 DETI coin with value v is worth 2^v coins of value 0. For example, the SHA1 secure hash of the file `deti_coin_example.txt` is

```
prompt> od -Ad -c -t x1 deti_coin_example.txt
0000000   D   E   T   I       c   o   i   n       2       0  \b   0  \b
         44  45  54  49  20  63  6f  69  6e  20  32  20  30  08  30  08
0000016   0  \b   0  \b   0  \b   4  \b   4  \b   0  \b   8  \b   0  \b
         30  08  30  08  30  08  34  08  34  08  30  08  38  08  30  08
0000032   2  \b   6  \b   8  \b   8  \b   4  \b   A   A   D     303 251
         32  08  36  08  38  08  38  08  34  08  41  41  44  20  c3  a9
0000048       f   i   x   e   .  \n
         20  66  69  78  65  2e  0a
0000055
prompt> sha1sum deti_coin_example.txt
aad20250e53c016ccea3a7161715e050b7684a19  deti_coin_example.txt
```

so its value is 0, because `0xe53c...` has 0 zero consecutive bits in its most significant part. Sending this file to a terminal yields (sorry, the coin text is a mixture of English and Portuguese)

```
prompt> cat deti_coin_example.txt
DETI coin 2 AAD é fixe.
```

Observe that the é character is encoded in UTF-8 by the two byte sequence `0xc3`, `0xa9`, and that part of the file contents is being hidden by the use of backspaces.

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.

`aad_utilities.h` — Code used by other parts of the program (time measurements, …).

`aad_data_types.h` — Declaration of the data types used by the code.

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

[1] See the Request for Comments 3174 for more details.

`aad_sha1.h` — The implementation of the SHA1 secure hash for a message with exactly 55 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. If you are a C macros guru, you may even adapt it to avoid unnecessary data copying when searching for DETI coins.

`aad_sha1_cpu.h` — Reference implementation of the SHA1 secure hash computation on the CPU (without and with SIMD instructions).

`aad_sha1_cpu_tests.c` — Program to test the correctness of the CPU code.

`aad_sha1_cuda_kernel.cu` — Device code of the reference implementation of the SHA1 secure hash computation using CUDA. Do **not use** this code as is when searching for DETI coins in the graphics card.

`aad_sha1_cuda_test.c` CPU code used to verify the correctness of the CUDA kernel code.

`aad_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. The coins are stored one per line, with its file contents precedeed by its value.

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

`makefile` — Makefile for the assignment.

`rfc3174.txt` — The original description of the SHA1 secure hash algorithm.

3 What is to be done

Create a program to search for DETI coins. Each group should **try** doing as much as possible of the following:

mandatory Search for DETI coins using the CPU (without using SIMD instructions).

mandatory Search for DETI coins using AVX or NEON instructions.

optional Search for DETI coins using AVX2 instructions.

optional Search for DETI coins using AVX512F instructions.

mandatory Search for DETI coins using CUDA.

recommended Do it with SIMD instructions and OpenMP.

recommended Do it using a server and many clients.

recommended Do it using Web Assembly.

awesome Do it using Web Assembly and SIMD instructions.

optional How about an OpenCL implementation? oneAPI implementation? ROCm implementation?

required For each of the above, measure how many attempts (SHA1 secure hash computations) you were able to do in one minute.

recommended Compare the performance, i.e., the numbers of attempts per minute, of as many computing devices as possible (recent and old computers, micro-controllers, smart phones, anything where you were able to run the program).

required Search for DETI coins with a special form (say, with part of your name embedded in it).

required Compute an histogram of the wall time it takes to run a CUDA search kernel.

required Compute an histogram of the number of DETI coins found in each CUDA kernel run.

optional Something else (surprise the teacher).

4 Recommendations for the CUDA search kernel

The provided CUDA kernel computes the SHA1 secure hash of a 55-byte message. When one searches for DETI coins, transferring all the messages that are DETI coin candidates from the host to the device (graphics card) is extremely inefficient. Indeed, when you run the `aad_sha1_cuda_test` program you will observe that it takes much more time in data transfers that in the actual computation.

```
prompt> ./sha1_cuda_test
test_sha1_cuda(): 448.000 MiB bytes for the interleaved32_data[] array
test_sha1_cuda(): 160.000 MiB bytes for the interleaved32_hash[] array
initialize_cuda(): CUDA code running on a GeForce GTX 1660 Ti (device 0, CUDA 10.2.0)
sha1_cuda_kernel() passed (8388608 tests, 3215611683 secure hashes per second)
  host -> device --- 0.035838 seconds
  kernel ---------- 0.002609 seconds
  device -> host --- 0.012403 seconds
```

Instead, create another `cu` file with modified code, so that each CUDA thread generates on the fly the entire message. Parts of the message will be constant in all threads, and other parts will depend on the coordinates of the thread (to differentiate the messages in a unique way, so that different threads work on different messages). Also, pass one or two integers to the kernel, to be used in a part of the message of your choice, so that different kernel runs also work on different messages.

Consider trying all possibilities for one message character in a single kernel run. This will help reduce wasted time (smaller overheads because the number of kernel launches is smaller).

The DETI coins found by a CUDA kernel should be stored in a common memory buffer, which can be small because DETI coins are rare. Create a memory area (on the host and on the device), and pass its device memory pointer to the CUDA kernel. Let us call it `coins_storage_area`, and let us assume that it has 1024 `u32_t` words. The idea is that the first element of the buffer, `coins_storage_area[0]`, holds the index of the first buffer word that is not yet in use. So, initialize it with 1 before each kernel run and copy that to the device. When a coin is found, increment `coins_storage_area[0]` using an atomic instruction to reserve space for another coin, and, if the buffer is not yet full, copy the coin to the buffer:

```
  idx = atomicAdd(coins_storage_area,14u); // reserve space for one more coin (14 words)
  if(idx < 1024u - 14u) // make sure we do not write outside of the buffer area
  {
    coins_storage_area[idx +  0] = coin[ 0];
    ...
    coins_storage_area[idx + 13] = coin[13];
  }
```

After a kernel run the device memory should be copied to the host memory and then all coins found should be stored. Of course, instead of storing in the buffer the entire coin contents, it is enough to store the information that is not fixed in your code (for example, the first 3 words, which always have the same text, do not actually need to be stored…)

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.

5 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 must 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`). Upload it for evaluation on `https://elearning.ua.pt/`.

The report must have a cover page, with the name an student numbers, stating the percentage to time each one spent doing this assignment. All uses of AI should be clearly stated.