Universidade de Aveiro

Mestrado em Engenharia de Computadores e Telemática

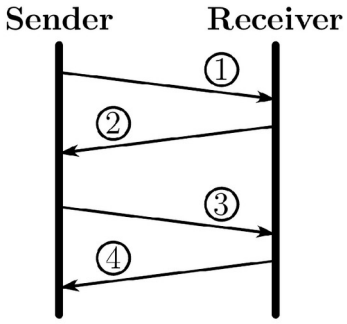
Arquitecturas de Alto Desempenho

Assignment 2 – Proof of Work

Academic year 2021/2022 Concept by Tomás Oliveira e Silva

There exist many situations in which one is interested in restraining, but not prohibiting, access to a resource. One way to accomplish this is to require a so-called proof-of-work before granting access to the resource.

For example, in an email system one may be interested in limiting the amount of junk mail (spam) that can be sent without impeding anyone from sending email. This can be done using the following exchange of messages (via a TCP socket, for example) between the two email systems (for each transaction a new token is generated using pseudo-random data).



1. [Request] I want to send an email to XYZ
2. [Challenge] Please send me a proof-of-work for the token T
3. [Data] Here it goes: PoW(T). Here goes the email message.
4. [Reply] message accepted/rejected

The proof-of-work function evaluates a custom modification of the MD5 message digest many times until a certain condition is met (mining bitcoins follows a similar approach). It receives the token (fourteen 32-bit words: unsigned int t[14]), and returns an array of two 32-bit words (unsigned int n[2])

$$[n_1, n_0] = \text{PoW}\ [t_{13}, t_{12}, \ldots, t_1, t_0].$$

The custom MD5 function receives an array of sixteen 32-bit words (unsigned int m[16]) and returns an array of four 32-bit words (unsigned int d[4])

$$[d_3, \ldots, d_0] = \text{cMD5}\ [m_{15}, m_{14}, \ldots, m_1, m_0].$$

The four least significant bits of m[0] encode an effort level used internally by the cMD5 function to control the number of rounds performed by the algorithm. The argument of the cMD5 function is constructed as follows

$$m_k = t_k\ (0 \le k \le 13) - m_{14} = n_0 - m_{15} = n_1.$$

The proof of work function has to find, and return, values for n1 and n0 for which the least 24 bits of d_0 are all zero. To verify the proof-of-work it is only necessary to evaluate the custom MD5 function once.

The fully functional, but not optimized, reference implementation accompanying this document uses the CUDA device to perform the proof-of-work work for 32 different tokens at the same time.

GRADING

- Optimize the threads launch grid, explain why the best grids are better than the other grids, and draw conclusions about the usefulness of offloading the computation to the CUDA device – 14 valores
- The reference implementation tries, for each token, only 2^{24} values for n_0 (keeping n_1 at 0) and so it may not finish the proof-or-work work for some of the tokens. Indeed, assuming that the values returned by the custom MD5 function are uncorrelated and uniformly distributed, the probability of finding at least one d_0 value with zeros in its 24 least significant bits after trying k different values of n_0 is $1 - (1 - 2^{-24})^k$, which, for $k = 2^{24}$, is only about 0.63212. To get a grade up to 17 try, for each token, $k = 2^{27}$ values for n_0; for this k, the probability of finding a good n_0 is close to 0.99966, which is reasonably close to 1. However, in most cases one ends up performing much more work than is actually needed. So, you must devise and implement a way to stop the computation as soon as a good value of n_0 is found – 17 valores.

DELIVARABLES

- an archive, named `POW_T$G#.zip` (where $, equal to 1, … ,4, means the lab number and #, equal to 1, …, 8, means the group number), of the CUDA files of your solution
- a pdf file, named `present.pdf`, up to 6 power point like pages, where the main ideas of the design and the conclusions of the work are presented.

DEADLINE

- January, 30, at midnight.