

Brute-Force Approach to Crack MD5 Hashes with MPI

Problem

Find an MPI routine to conduct a brute-force search for the initial string used to create an MD5 hash.

Background

Most passwords are stored in an encrypted form through the use of a hashing routine. The MD5 hashing routine was created by Ron Rivest at MIT in 1991. Widely used to encrypt passwords and SSL sessions, weaknesses were discovered in 1996 and 2004. It has been currently deprecated in favor of the SHA family of hashing functions.

The size of the hash – 128 bits – is considered small enough to conduct a birthday attack¹. The estimated number of computations to find a match would be 2^N where N is the number of bits (128). A computer capable of 12.25 Teraflops would take about three weeks to find a match using a serial approach². MD5CRK was a distributed project started in March 2004 with the aim of demonstrating that MD5 is insecure by using an analytic attack which was able to find a match in 1 hour.

There are a variety of products currently that use a parallel approach to brute-force attack MD5 hashes. Using CUDA based programs on an NVIDIA 8800 GPU is fairly popular. There are also MPI versions of John-the-Ripper (dictionary based attack) used to crack MD5 hashes.

Algorithm

The MD5 hashing routine consists of taking a variable length message input, pad it, then break it up into 512 bit blocks which are then modified by a series of four operations. The C code for the MD5 algorithm was taken directly from Ron Rivest's original code located at <http://people.csail.mit.edu/rivest/md5.c>

A brute force approach can then be taken:
original string → MD5 routine → hash
test string → MD5 routine → hash2

Ron Rivest



All possibilities of test strings starting from 1, ..., n characters of [a-z], [A-Z], and [0-9] combinations are tried until hash2 = hash which then defines the test string to be the original string.

Serial Approach

Using the md5.c code, a hash code is generated from a variable length input string which is outputted to a file. The crack program then reads in this hash and attempts to decipher the original string by generating strings, applying the MD5 routine, and then comparing to the read in hash. The program starts from “a” and continuing on every combination for the characters “a-z”, “A-Z”, and “0-9” until a hash match is made.

Parallel Approach

Domain decomposition consisted of evenly dividing the character combinations across the nodes. The approach was to divide the “a-z”, “A-Z”, and “0-9” by the number of processors who in turn process their piece of pie. The first node to find a hash match outputs to screen the node id and the test string that was used to get the hash match. An exit(1) is then issued by that node terminating all node processes.

Speed & Scaling

Time trials were conducted on the Amazon Cloud using a single Ubuntu 64-Bit 8-Core AMI Instance. Time trials were also run on the GMICE cluster with similar results. Posted results below are from the Amazon Cloud. Real time (v1) is done with the algorithm checking lowercase alpha [a-z], (v2) checks [a-z], [A-Z], and [0-9].

6-character initial string (“nvidia”)

cores	real time (v1)	real time (v2)
8	47.143s	6m12.584s
6	47.378s	7m6.536s
4	52.018s	8m9.712s
2	1m3.920s	13m26.489s

5-character initial string (“hello”)

cores	real time (v1)	real time (v2)
8	2.282s	1m7.380s
6	2.482s	2m9.844s
4	4.925s	3m16.926s
2	10.943s	7m9.671s

4-character initial string (“help”)		
cores	real time (v1)	real time (v2)
8	.336s	2.182s
6	.295s	3.952s
4	.273s	7.221s
2	.232s	8.031s

3-character initial string (“var”)		
cores	real time (v1)	real time (v2)
8	.337s	1.119s
6	.293s	1.004s
4	.274s	1.003s
2	.207s	.970s

Mathematically, the amount of combinations needed to be tried to crack an N-character password on average is determined by $(N^C) / 2$ where C is the number of characters in the character set, or 26 for lowercase alpha. Using a value of the typical dual-core machine is capable of generating and comparing 40 million hashes / sec.

Theoretical Times

var	.009s
help	1.67s
hello	9m25s
nvidia	17.65 hrs

Observed Serial Times on a Dual-Core Ubuntu AMI on the AWS

var	.065s
help	1.071s
hello	8m38.646s
nvidia	Quit at the 3 hour mark.

The Graphs

8x slower @ 6 characters w/8 procs when using an increased character set (+36)

30x slower @ 5 characters w/8 procs when using an increased character set

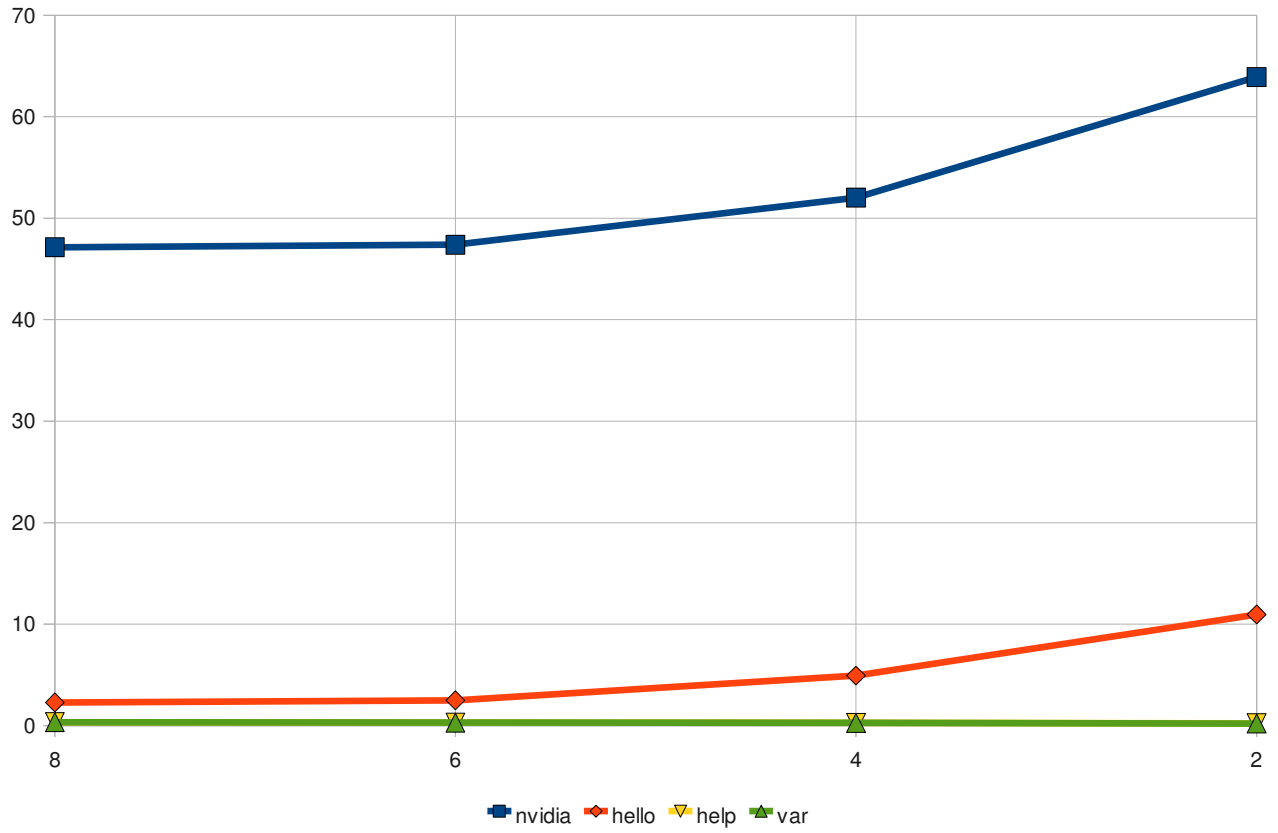
21x slower going from 5 to 6 characters w/8 procs on reduced char set

6x slower going from 5 to 6 characters w/8 procs on full char set

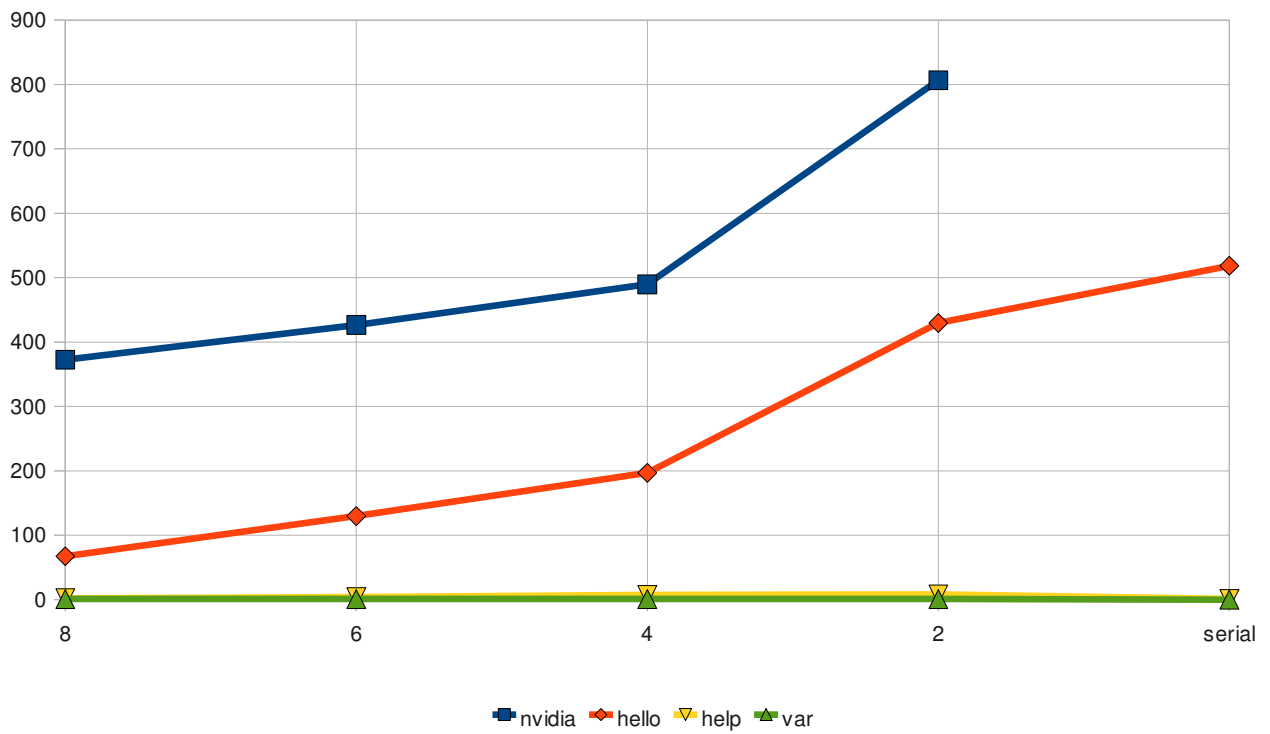
Serial times are faster until you get to 5-6 characters due to overhead

...

AWS V1



AWS V2



Graphical Speed-ups (cont)

90% speed-ups when adding two cores with a full character set, speed-ups with reduced char set not observed probably due to overhead – with increased char length, speed-ups should be observed on the reduced char set when adding nodes.

Problems Encountered

Due to the limited availability of the GMICE, the code was run on the Amazon Cloud with an Ubuntu 64-Bit 8-Core AMI instance. Later on when GMICE was up, time trials were run with similar results to the Amazon Cloud. Encountered a problem on the Amazon Cloud where it limited the real time processing to 1m30s if using an 8 core image. This was later resolved by using an updated AMI image.

Most time consuming portion was determining a method to divide up the data set, but it became straightforward once the characters/numbers were represented by their ascii values.

Summary / Conclusion

The problem above was not particularly complicated, however it does demonstrate an example of a code where the steps are time independent and lends extremely well to parallelization. This would be a good example to do on a GPU as the data sets do not depend on each other and memory coalescing should be inherent.

The brute force method is valid on any hash function to include SHA, MD5, and RSA. The code can be easily expanded to include additional characters in the set or length of password. Other methods of hash cracking include dictionary and rainbow table lookups which are generally quicker, but for passwords that include a salt (additional characters injected into the original string), brute-force remains a viable solution.

The addition of new character types increased the time by 8x with a 6-char maximum search. Adding a char-length increased the time 6-21x depending on the char set used. Modern day password policies require a full character set with a 14 character password minimum on DoD systems. However, since the problem does scale with increased processing power, the time required to crack such a password would not be an unsurmountable obstacle to a hacker if they got a hold of the original hash. With the use of GPUs and multiple processors, hashing routines should be considered a speed-bump when designing a security policy. Importance should also be given to the protection of the transmission line and limiting access to consoles to further deter intrusions.