Performance Comparison of Parallel Implementations of Cayley and SHA Hash Functions

Sourabh Kulkarni  
Department of Computer Science  
Rochester Institute of Technology  
Email: ssk3315@rit.edu

Abstract—Implementing hash functions to run on multi-core computers will reduce its running time. We aim to implement Cayley, SHA-256 and SHA-512 hash functions to run on multi-core computers and compare their performances. Cayley hash compresses the input message by hashing one bit at a time using composition of linear equations. SHA-256 and SHA-512 work on message blocks and produce fixed length digest as output. We parallelize these hash functions using parallel j-lanes hashing mode. Based on our experiments, SHA-2 outperforms Cayley hash function in terms of running time.

Keywords—Cayley, SHA-256, SHA-512, parallel j-lanes hashing, pj2.

I. INTRODUCTION

Hash functions are compression functions that take message of arbitrary length as input and produce a fixed length digest as output. Hash functions are often used as digital fingerprints to check message integrity. Most widely used hash functions are sequential algorithms and hence, don’t make best use of available resources of modern computers. Parallel computation of hash value may greatly reduce the total running time of the algorithm. In this paper we discuss parallel implementation of Cayley hash function and SHA-2 family of hash functions. We also test the following hypothesis: For a given security level, the parallel Cayley hash function outperforms the parallel SHA hash function[3].

Cayley hash function uses two linear equations over $F(p)$ (the field of integers modulo a prime P) to hash every 0 and 1 bit of input bit string. The hash is calculated using composition of these individual functions. The final digest of length 512 bits is constructed using coefficients of resulting linear equation. Cayley hash functions can easily be parallelized due to their homomorphic property i.e. $H(AB) = H(A)H(B)$ for any bit strings A and B[3]. The security level provided by Cayley hash function depends on field size and hence, can be easily modified to match it with security level provided by SHA functions for the purpose of performance comparison.

Many block hash functions like SHA[1] are hard to parallelize due to their sequential nature. We can use parallel tree mode to run such hash functions on multi-core computers[2]. Every core has its own buffer memory which is used to feed in the input data for computation. These buffers are filled in round robin fashion using input message blocks. Final digest is calculated using the concatenation of these intermediate digests produced by different cores.

The paper is organized as follows. In section 2 we discuss Cayley hash function and improvements in the same. Section 3 talks about parallel tree mode hashing and section 4 presents experimental setup and evaluation of results. In section 5 we give our final remarks about the results and conclude the paper.

II. SHA-2 HASH FUNCTIONS

SHA-2 (Secure Hash Algorithm) family of hash functions designed by National Security Agency (NSA) has six different hash functions. These hash functions mainly differ in input message block sizes and digest sizes. Out of these six, SHA-256 and SHA-512 are most widely used and popular hash functions. Therefore, for comparison with Cayley hash function, we only consider these two.

SHA-256 processes blocks of size 512 bits in every cycle of compression function, whereas SHA-512 takes in blocks of size 1024 bits. Both the hash functions pad input message if the size of message is smaller than required block size. SHA-256 produces digest of size 256 bits, whereas SHA-512 produces output digest of size 512 bits. Word sizes used by SHA-256 and SHA-512 are 32 and 64 respectively.

Both hash functions use six different logical functions in compression function. But, in SHA-256, all these functions operate on 32 bit words i.e. integer variables in java, while in SHA-512, these functions work on 64 bit words i.e. long variables in java. These six functions are namely $ch$, $maj$, $\sum_0$, $\sum_1$, $\sigma_0$ and $\sigma_1$. First two functions mainly use xor, and and negation operations. Remaining functions use right rotate and right shift operations. SHA-256 uses 64 constants of size 32 bits, while SHA-512 uses 80 constants of size 64 bits. Compression functions of SHA-256 and SHA-512 iterate 64 and 80 times respectively on each message block to find intermediate hash values.

Most important difference between SHA-2 and Cayley hash functions is the way intermediate hash value is computed. SHA-2 iteratively works on message blocks to compute hash values but Cayley extracts out the bits of the message and computes intermediate hash value. SHA-2 waits for fixed number of bits to arrive before starting the hash computation, whereas Cayley may process input as and when it becomes available.
III. Cayley hash function

Cayley hash function uses two linear equations to hash every bit of the input message. Unlike many conventional hash functions, Cayley hash function works on bits of the message one at a time instead of bytes or message blocks. Typically, Cayley hash function takes a bit stream as input and gives two 256-bit numbers as output which are used to construct hash of input message. The digest size and security level of the Cayley hash depends on size of these two numbers.

![Fig. 2. Working of Cayley hash function](image)

Cayley hash which we implemented is based on composition of linear equations of one variable over a field. For example, let \( f(x) = 2x + 3 \mod p \) and \( g(x) = 3x + 1 \mod p \), and input message with two bits 0 and 1 respectively. Every 0 bit in input message is replaced with \( f(x) \) and every 1 bit in input message is replaced with \( g(x) \). Now, composition of two functions i.e. \( f(g(x)) \) produce another linear function of \( 6x + 10 \). Coefficients in output function are used to build final hash or digest of the message. Let’s say, final linear equation after hashing every bit of the message is \( rx + s \) then the digest of the hash function is \( (r+s, s) \).

Example:

Input bit string = 010, \( f(x) = 2x + 3 \mod p \) and \( g(x) = 3x + 1 \mod p \). Input bit string after equation replacements = \( f(x)g(x)f(x) \) composition of \( f(x)g(x)g(f(x)) = g(f(x)) = 3f(x) + 1 = 6x + 10 \) Hashing final bit = \( f(g(f(x))) = 12x + 23 \) Digest = \( (12+23, 23) = (35, 23) \)

A. Implementation of Cayley hash function

During computation of Cayley hash, we need to work with linear equations of one variable and final digest also depends on such a linear function. Instead of working with these linear equations and keeping one variable throughout the computation of compression function, we can find the value of equation at two different points and then recover the linear function using properties of straight lines.

Let’s say first bit of the input message is 0 and is hashed with equation \( f(x) \). We find the value of the equation at two points say, \( p1 \) and \( p2 \) and continue hashing next bits by putting these values for variable of linear equations. Ultimately, we will get two points \( q1 \) and \( q2 \) which are guaranteed to be a pair of points on the final linear equation. Using two point form of straight lines, we recover the final linear equation back with its coefficients and compute the digest.

Using above example with same input bit string and linear equations, let two randomly chosen points are \( (2,0) \) and \( (5,0) \). Hashing first zero gives \( p1 = 2*2 + 3 = 7 \) and \( p2 = 2*5 + 3 = 13 \)

Similarly, hashing next 1 and 0 gives us \( p1 = 47 \) and \( p2 = 83 \). Now these \( (2,47) \) and \( (5,83) \) are the two points on final straight line. Using point-slope form of line, \( y = mx + c \). Slope \( m \) of the line can be calculated as \( (y2 - y1) / (x2 - x1) = 12 \). Equation of line becomes \( y = 12x + c \). To get the value of \( c \), we can substitute any of the two points in above equation. Here, \( c = 83 - 12*5 = 23 \). This is how we recover final linear equation.

B. Security of Cayley hash function

Collision Resistance of hash function is a good measure of security of any hash function. Hash collision means two strings which are hashed to same digest. The more hard it is to find pair of input messages having same hash values the more secure your hash function is. Another property of hash functions is pre-image resistance. This means reversing the hash function should be difficult. Given a hash value, finding an input message which has same hash value should be difficult. For any hash function to be secure, the digest length should be at least twice as long as digest length required for having pre-image resistance property. This is called as birthday attack.

Cayley hash may produce same hash value for two input bit strings if and only if length of one of the bit strings is at least \( \log_3 p \) [1]. \( p \) is field size for the hash computation. If coefficients of linear equations are of length 256 bits then \( p \) is of the order \( 2^{256} \). Therefore, for hash collisions, at least one of the bit strings should be as long as 162 bits. Hence, according to birthday attack, to break this hash function, a brute force search over bit strings of length more than 80 needs to be performed.

C. Modifying Security of Cayley hash function for comparison

For fair comparison between hash functions, security levels provided by respective hash functions should be same. Modifying security provided by Cayley is easier than SHA. We
change the field size and effectively the digest size of Cayley hash to match the security levels of two hash functions.

Security level of SHA-256 is 128 i.e. to break the hash function and find a collision, one needs to brute force over bit strings of length 128. Therefore, for comparison of Cayley and SHA-256, Cayley need to provide security of 128. As we discussed earlier, security level of Cayley is half of the value of \( \log_3 p \).

Therefore,

\[
\log_3 p = 256
\]

\[
p = 2^{406}
\]

Hence, field size used for calculating Cayley hash for comparison with SHA-256 needs to be 406.

Similarly, for comparison with SHA-512, field size needs to equal to 812. Modifying this field size means using bigger bit strings for coefficients of the linear equation and ultimately having bigger digest size for the hash function.

IV. PARALLEL J-LANES HASHING

Cayley hash function’s homomorphic property makes it easily parallelizable. Other conventional hash function does not have such inherent property to make it easily parallelizable. For example, SHA-256 has sequential dependency which makes it harder to implement to run on multi-core computers. To use available resources of the modern computers and reduce running time of hash computation, we use parallel j-lanes hashing [2] mechanism to parallelize the task of hash computation.

We implemented parallel j-lanes hashing as a framework which takes in any hash function and makes it possible to run on multi-core computers. It first splits the input message into j-lanes. Each core processes one lane or split of the message independent of other cores and at the end final digest is the hash value of the concatenation of hash values produced by each of the cores. We use buffers as temporary memory of each core. Each of the cores has its own instance of hash function and apply hash function for one final time on this concatenated string. The output is final digest of the input message.

As each core has its own instance of hash function, compression can be done independent of other cores. This overcomes the serial nature of the hash function. Based on the input hash function and number of cores used, the hash value produced will be different and hence, standardization of this process is necessary.

V. EXPERIMENTS AND RESULTS

We implemented Cayley hash, SHA-256, SHA-512 and parallel tree mode. For performance comparison of these hash functions we had two different modes of hash computation. One is while data is being streamed from external file and other is while data is already in the memory. First mode will provide running time comparison of hash functions while data is being streamed in where threads may need to wait for respective buffer to be filled. The second mode will help in running time comparison without any reading overhead on the threads. As discussed earlier, we also modify digest length of cayley hash depending on hash function it is being compared with.

CS department’s Champ multi-core parallel computer was used to run the experiments. Champ has 8 dual-hyperthreaded CPU cores which can run up to 16 threads at a time without context switching. Parallel Java 2 library developed by Prof. Alan Kaminsky was used to implement parallel programs. Also PJ2’s BigInteger class was used in the implementation of Cayley hash.

Following are the running times (msec) of SHA-256, cayley hash 1(digest size of 406 bits), SHA-512 and cayley hash 2(digest size of 812 bits) when data was being streamed in from a file of size 213 megabytes.

**TABLE I. RUNNING TIMES OF HASH FUNCTIONS IN MSEC (DATA STREAMING IN)**

<table>
<thead>
<tr>
<th>Cores</th>
<th>SHA-256</th>
<th>Cayley (digest=406)</th>
<th>SHA-512</th>
<th>Cayley (digest=812)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60739</td>
<td>283011</td>
<td>59397</td>
<td>351463</td>
</tr>
<tr>
<td>2</td>
<td>55778</td>
<td>86637</td>
<td>56438</td>
<td>123269</td>
</tr>
<tr>
<td>4</td>
<td>56683</td>
<td>71291</td>
<td>55903</td>
<td>75674</td>
</tr>
<tr>
<td>8</td>
<td>56322</td>
<td>54586</td>
<td>56103</td>
<td>70718</td>
</tr>
<tr>
<td>16</td>
<td>56163</td>
<td>52388</td>
<td>56066</td>
<td>53672</td>
</tr>
</tbody>
</table>

Fig 5 is plot of the running time (msec) comparison between SHA-256 and Cayley hash and fig 6 is plot of the running time comparison between SHA-512 and Cayley hash when data is being streamed in from a file of size 213 megabytes.

Following are the running times (msec) of SHA-256, cayley hash 1(digest size of 406 bits), SHA-512 and cayley hash 2(digest size of 812 bits) when data was already read in the memory of size 213 megabytes.
Fig. 5. Running time comparison SHA-256 v/s Cayley (data streaming in)

Fig. 6. Running time comparison SHA-512 v/s Cayley (data streaming in)

Fig. 7. Running time comparison SHA-256 v/s Cayley (data in memory)

Fig. 8. Running time comparison SHA-512 v/s Cayley (data in memory)

TABLE II. Running times of hash functions in msec (data in memory)

<table>
<thead>
<tr>
<th>Cores</th>
<th>SHA-256</th>
<th>Cayley (digest=406)</th>
<th>SHA-512</th>
<th>Cayley (digest=812)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6549</td>
<td>146732</td>
<td>9371</td>
<td>197241</td>
</tr>
<tr>
<td>2</td>
<td>3884</td>
<td>77672</td>
<td>4508</td>
<td>94678</td>
</tr>
<tr>
<td>4</td>
<td>2519</td>
<td>40590</td>
<td>2543</td>
<td>50229</td>
</tr>
<tr>
<td>8</td>
<td>1464</td>
<td>35256</td>
<td>1578</td>
<td>40137</td>
</tr>
<tr>
<td>16</td>
<td>852</td>
<td>24944</td>
<td>1156</td>
<td>30887</td>
</tr>
</tbody>
</table>

Fig. 9 and fig 10 are the plots of speedup and efficiency of the three hash functions respectively.

In both the modes of data reading, i.e. when data is in memory and data is being streamed in, SHA-2 finishes quicker than Cayley. In case of data being streamed in, running times of both the hash functions are almost identical when number of cores are more than two. This is because computation threads spend more time waiting for their buffers to be filled by reading thread. This indicates sequential overhead of reading file is more than hash computation.

When data is already read in memory, running time of Cayley is almost twenty times more than SHA-2. In this mode of comparison, we observe the speedup as we increase number of cores. This is because sequential dependency present in earlier mode is removed and no longer an overhead.

Fig. 7 is plot of the running time (msec) comparison between SHA-256 and Cayley hash and fig 8 is plot of the running time comparison between SHA-512 and Cayley hash when data is in memory of size 213 megabytes.
VI. FUTURE WORK

The reason behind longer running times of Cayley hash needs to be investigated. One possible way to validate and justify these running time is to mathematically calculate average number of operations required for hashing a block of data. In future, we can also try to optimize the implementations of hash functions to get better speedup and efficiencies. Investigating effectiveness of using cluster computers for hash computation can also be a scope of future work.

VII. CONCLUSION

From the plots it is clear that for our implementation of Cayley and SHA-2 hash functions, SHA-2 outperforms Cayley.

REFERENCES

