Multi-Thread Blockchain using OpenMP

A REPORT

submitted by

Wilson Vidyut Doloy (19BCE1603)

Rishank Pratik (19BCE1606)

Dinesh Sharma (19BCE1046)

in partial fulfilment for the award

of

B. Tech. Computer Science and Engineering

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

December 2021

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled **"Multi-Thread Blockchain using OpenMP"** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai 600127 in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology – Computer Science and Engineering** is a record of bonafide work carried out by me**.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

Wilson Vidyut Doloy (19BCE1603)

Rishank Pratik (19BCE1606)

Dinesh Sharma (19BCE1046)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled "**Multi-Thread Blockchain using OpenMP**" is prepared and submitted by **Wilson Vidyut Doloy (19BCE1603), Rishank Pratik (19BCE1606), Dinesh Sharma (19BCE1046)**. It has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology – Computer Science and Engineering (Undergraduate)** in Vellore Institute of Technology, Chennai, India.

Examined by:

Prof. Sudha A

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Sudha A**, Assistant Professor (Sr), School of Computer Science and Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Ganesan R**, Dean, School of Computer Science and Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr. P.Nithyanandam** for their support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

WILSON RISHANK DINESH

CONTENTS

LIST OF FIGURES

ABSTRACT

Blockchain is a technology that is developed using a combination of various techniques which include mathematics, algorithms, cryptography, economic models, etc. In today's Blockchain system millions of machines simultaneously perform calculation. The cost of resources needed such as powerful machines/CPUs becomes very high. This can be too much for single organizations who want to perform all calculations internally only. The experimental results show that the program works efficiently on different machines depending on the number of threads and the computer specifications. The purpose of this project is to save cost on the machines, working inside a single organization. So, in this project we have focused on minimizing the cost of those machines, therefore using multiple threads to perform the task of those machines.

OBJECTIVE

In today's blockchain system, it is known that millions of machines simultaneously perform calculations that take hours to satisfy a condition. All these devices can perform these calculations by means of data communication among themselves.

Similarly, in this project, the sharing of common data and parallel computing processes between today's computers will be implemented using threads. The purpose of the simulation is to implement the blockchain working application with multiple threads/cores instead of multiple machines.

INTRODUCTION

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An *asset* can be tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

Blockchain is an ever-growing list of records, called blocks, that are secured using cryptography. The blockchain consists of blocks connected to each other with the SHA 256 algorithm. The SHA 256 algorithm takes any length of character text as input and outputs a 256-bit text.

SHA 256 is a part of the SHA 2 family of algorithms, where SHA stands for Secure Hash Algorithm. Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against brute force attacks.
The significance of the 256 in the name stands for the final hash digest value, i.e. irrespective of the size of plaintext/cleartext, the hash value will always be 256 bits.

Hashing is the process of scrambling raw information to the extent that it cannot reproduce it back to its original form. It takes a piece of information and passes it through a function that performs mathematical operations on the plaintext. This function is called the hash function, and the output is called the hash value/digest.

The way that blockchain prevents simultaneous writes is by having some manner of block proposal mechanism in place across the system. These come in multiple flavors and are generally of the form “Proof of x”, where inconsistencies in the shared state can be resolved based on consensus on the proof of “x” provided by the publishers.

The proof condition acts as a mutex on the shared state, unlocking write permission for the publisher, and updating the state.

When using bitcoins or other cryptocurrencies, blockchain mining is a process that verifies each stage of the transaction. The people participating are known as blockchain miners, and their primary goal is to confirm the movement of cash from one computer in the network to another through a maze of computing gear and software.

The phrase "blockchain mining" describes the process of adding transaction data to the bitcoin blockchain at its most basic level. These Blockchain miners set up and run specialized Blockchain mining software on their computers, which allows them to safely interact with one another. When a machine downloads the software, joins the network, and starts mining bitcoins, it is referred to as a 'node.'

All of these nodes work together to interact with one another and process transactions in order to add new blocks to the blockchain, also known as the bitcoin network. There is nothing like a centralized authority—a regulatory agency, a governing body, a bank—in Blockchain to ensure bitcoin transactions are completed. Any user with mining hardware and Internet connection may join the mining community and participate. Proof of work is a tough mathematical challenge that is used to solve the procedure. Proof of work is required to validate the transaction and receive a payment for the miner. The miners compete amongst themselves to mine a specific transaction, with the miner who solves the riddle first receiving the prize. Miners are members in the network who have the requisite hardware and processing capacity to validate transactions.

RELATED WORKS

1. Zhang, Z., 2016. "*A multi-threaded cryptographic pseudorandom number generator test suite*." Naval Postgraduate School Monterey United States.
Concept: This paper spoke about generating random numbers suitable for cryptographic purposes using multi-threads.
Advantages: Error free and fast generation of random numbers for cryptographic purposes.
Disadvantages: Time consuming if considered for blockchain.

2. Shae, Zonyin, and Jeffrey Tsai. "*Transform blockchain into distributed parallel computing architecture for precision medicine*." In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1290-1299. IEEE, 2018.
Concept: This paper proposes mechanisms to transform the blockchain duplicated computing into distributed parallel computing architecture by transforming smart contract which features data driven from the ground up to support moving computing to native data strategy.
Advantages: Transaction validation
Disadvantages: Cannot work on characters. Only works on numeric values.

3. Samy, Hossam, Ashraf Tammam, Ahmed Fahmy, and Bahaa Hasan. "*Enhancing the performance of the blockchain consensus algorithm using multithreading technology*." *Ain Shams Engineering Journal* (2021).
Concept: Used Byzantine Fault Tolerance voting-based algorithm
Advantage: Used Byzantine Fault Tolerance voting-based algorithm that provides a higher throughput
Disadvantage: Did not optimize Proof-of-Work Algorithm.

4. Hazari, Shihab Shahriar. "*Design and development of a parallel Proof of Work for permissionless blockchain systems*." PhD diss., 2019.
Concept: Used Byzantine Fault Tolerance voting-based algorithm
Advantage: No two or more miners put the same effort into solving the same block
Disadvantage: Used a middle-man called "validator" to ensure fairness.

PROPOSED SYSTEM

Initialize Hash and Text definitions:

Text: “Hello there! I am from team Alioth. Thank you for accepting this project”

Hash: “0000000000000000000000000000000000000000000000000000000000000000”

Firstly, Data is created for the current block. With "initilizeHash" and "initializeText" texts, Block 1 "inputText" value is generated as:

inputText = Block number + initialize Text + Nonce

So, in our case, **inputText** = 1Hello there! I am from team Alioth. Thank you for accepting this project0000000000000000000000000000000000000000000000000000000000000011

where, nonce is the number of possibilities after which acceptable Hash is generated. Then the value of "inputText" is given to the SHA 256 algorithm. The resulting value is given to the Hash acceptance condition.

Hash acceptance condition consists of 2 sub-conditions:

1. The resulting hash value must start with 0 as much as the Block number.
2. The character to the right of the characters of zero must be nonzero.

For example, the result "0xxxx...." is acceptable for Block 1, while the result "00xxxx...." is not. If the condition is met, the next block is passed.

Now, suppose that the nonce and hash values are found as follows and the condition is satisfied.

- Nonce = 11
- Hash = 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b

In this case, inputText for Block 2 is as follows:

inputText = 21Hello there! I am from team Alioth. Thank you for accepting this project0000000000000000000000000000000000000000000000000000000000000001107c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b + Nonce

If the condition is met, the next block is passed and so on.

ARCHITECTURE

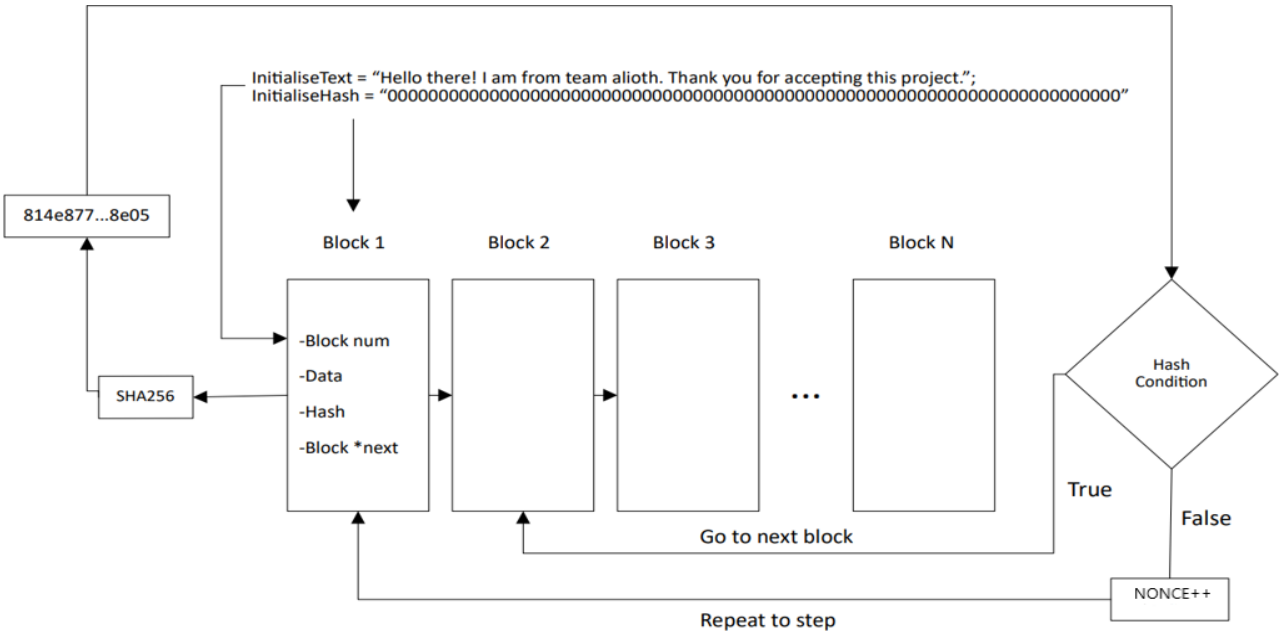


Fig 1: Architecture of Multi-Threaded Blockchain system

Software:

GCC Compiler: The GNU Compiler Collection, commonly known as GCC, is a set of compilers and development tools available for Linux, Windows, various BSDs, and a wide assortment of other operating systems. It includes support primarily for C and C++ and includes Objective-C, Ada, Go, Fortran, and D.

Environment – Visual Studio 2022: Microsoft Visual Studio is an IDE made by Microsoft and used for different types of software development such as computer programs, websites, web apps, web services, and mobile apps. It contains completion compilers, tools, and other features to facilitate the software development process.

MODULES

1. Input text:

The text will be entered by the user. It can be of any form. A phrase, a currency amount etc.

```
Text: 1Hello there! I am from team Alioth. Thank you for accepting this project
```

2. Hash value generation:

For each input text a unique hash will be generated. We will be using SHA256 cryptographic hash function for the same.

```
Hash accepted: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b          nonce: 11              tid: 3
Hash accepted: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b          nonce: 11              tid: 2
Hash found: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b             nonce: 11              tid: 4
```

3. Block generation:

As each transaction occurs, it is recorded as a “block” of data Those transactions show the movement of an asset that can be tangible. The data block can record the information as who, what, when, where, how much and even the condition.

```
# BLOCK: 1      tid: 0
Hash initialized: 0000000000000000000000000000000000000000000000000000000000000000      nonce: 0               tid: 0
Hash accepted: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b          nonce: 11              tid: 3
Hash accepted: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b          nonce: 11              tid: 2
Hash found: 07c9c32d9d6b70bb8bf4696f2fca0c1065ad839476f5d5f623afdd551a5e728b             nonce: 11              tid: 4
Text: 1Hello there! I am from team Alioth. Thank you for accepting this project00000000000000000000000000000000000000000000000000000000000000000000011
Block Run-time(s): 0.013        Total Run-time: 0:0:0
```

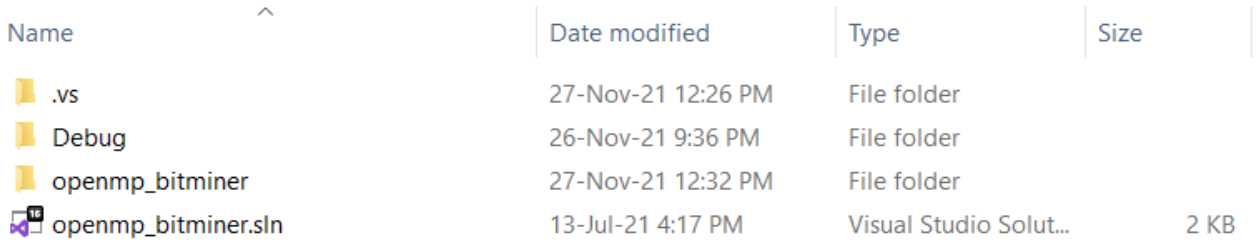
IMPLEMENTATION

#pragma omp barrier: To identify a synchronization point at which threads in a parallel region will wait until all other threads in that section reach the same point. This will calculate SHA256 value for initial text and then rest of processes will carry on.

#pragma omp critical: Specifies that the code is executed by one thread at a time. This will calculate our nonce value for each block in blockchain.

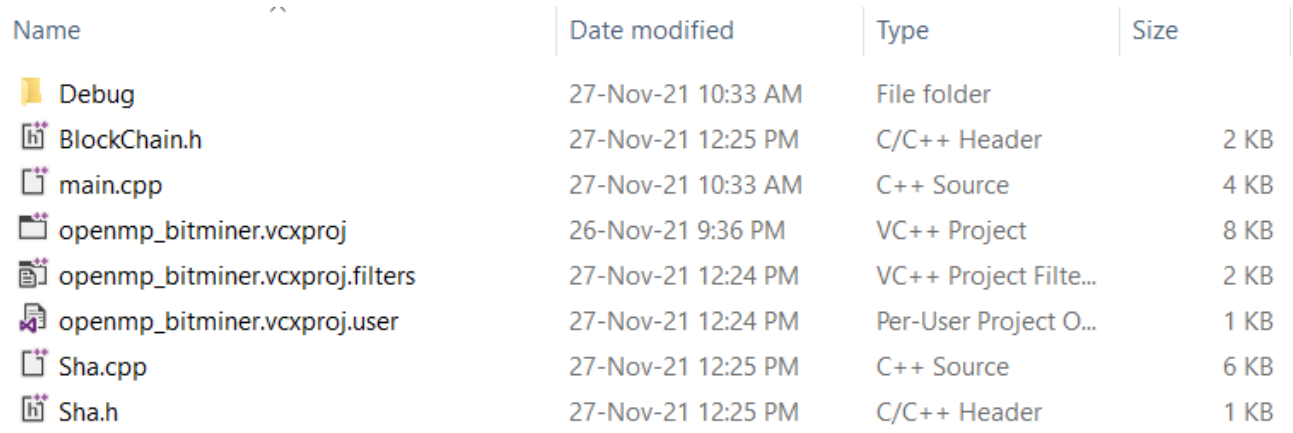
#pragma omp parallel num_threads(num): The number of threads the program will run parallel for. Entered by the user.

#pragma omp single nowait: Used to avoid the implied barrier at the end of the single directive. Once Private nonce value is accepted for other threads, print block information.



Name	Date modified	Type	Size
.vs	27-Nov-21 12:26 PM	File folder	
Debug	26-Nov-21 9:36 PM	File folder	
openmp_bitminer	27-Nov-21 12:32 PM	File folder	
openmp_bitminer.sln	13-Jul-21 4:17 PM	Visual Studio Solut...	2 KB

Fig 2: Project Files



Name	Date modified	Type	Size
Debug	27-Nov-21 10:33 AM	File folder	
BlockChain.h	27-Nov-21 12:25 PM	C/C++ Header	2 KB
main.cpp	27-Nov-21 10:33 AM	C++ Source	4 KB
openmp_bitminer.vcxproj	26-Nov-21 9:36 PM	VC++ Project	8 KB
openmp_bitminer.vcxproj.filters	27-Nov-21 12:24 PM	VC++ Project Filte...	2 KB
openmp_bitminer.vcxproj.user	27-Nov-21 12:24 PM	Per-User Project O...	1 KB
Sha.cpp	27-Nov-21 12:25 PM	C++ Source	6 KB
Sha.h	27-Nov-21 12:25 PM	C/C++ Header	1 KB

Fig 3: Files inside openmp_bitminer folder

RESULT

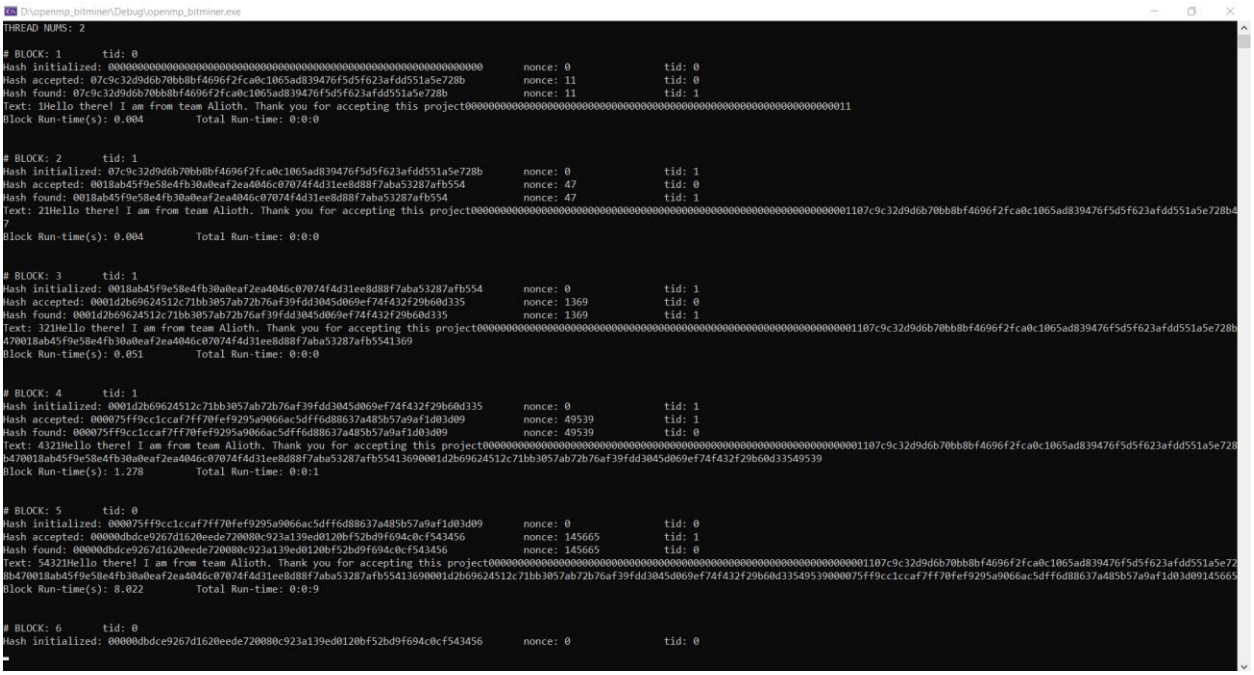


Fig 4: Results for default initilizeText and initializeHash values using 2 threads.

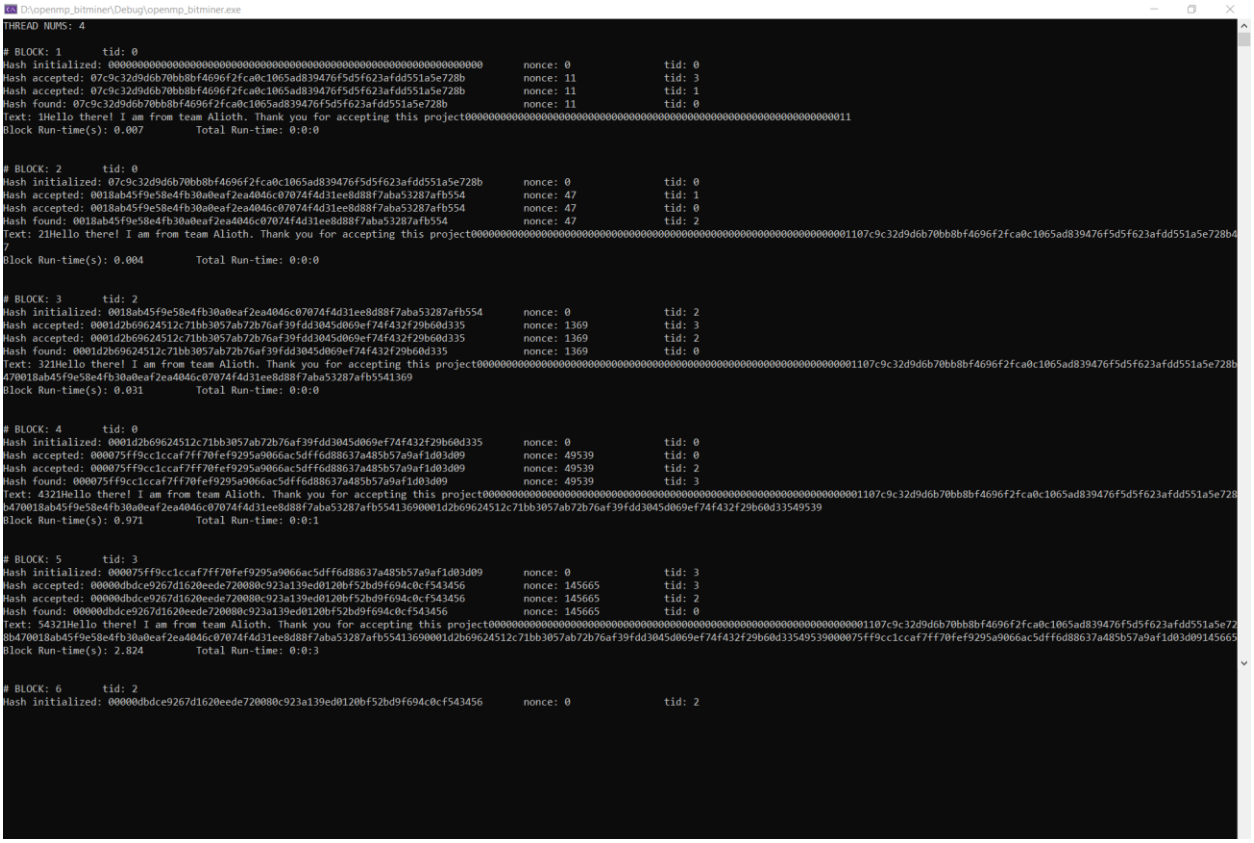


Fig 5: Results for default initilizeText and initializeHash values using 4 threads.

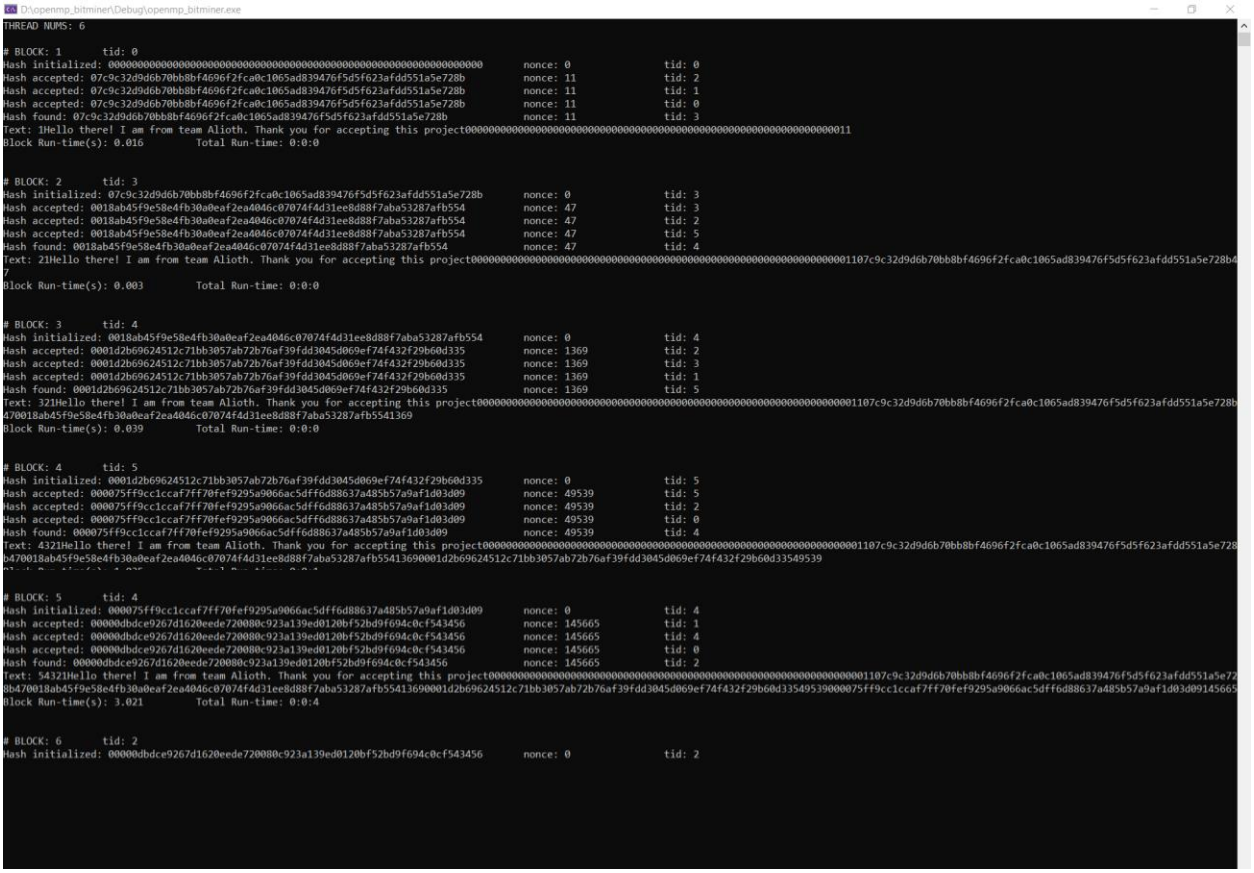


Fig 6: Results for default initilizeText and initializeHash values using 6 threads.

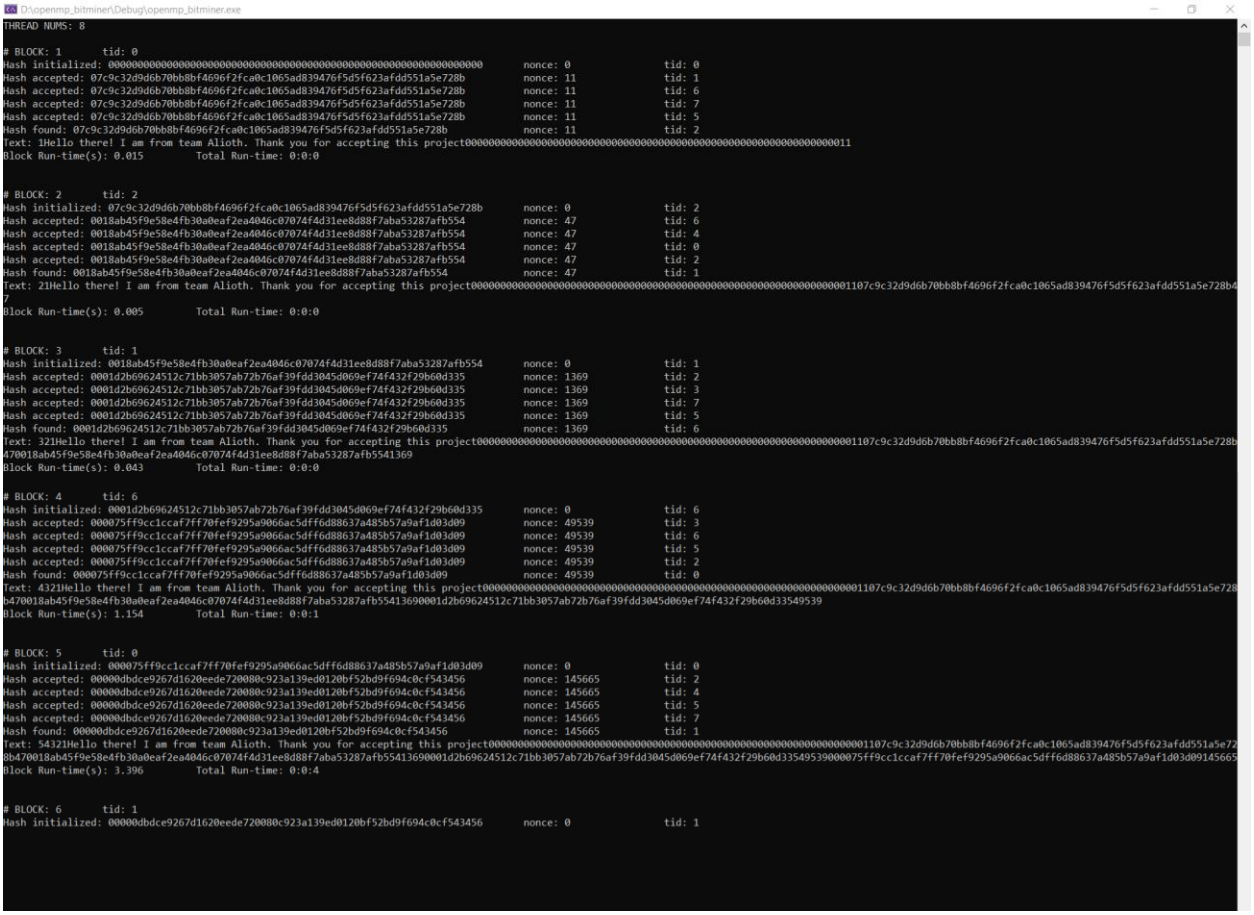
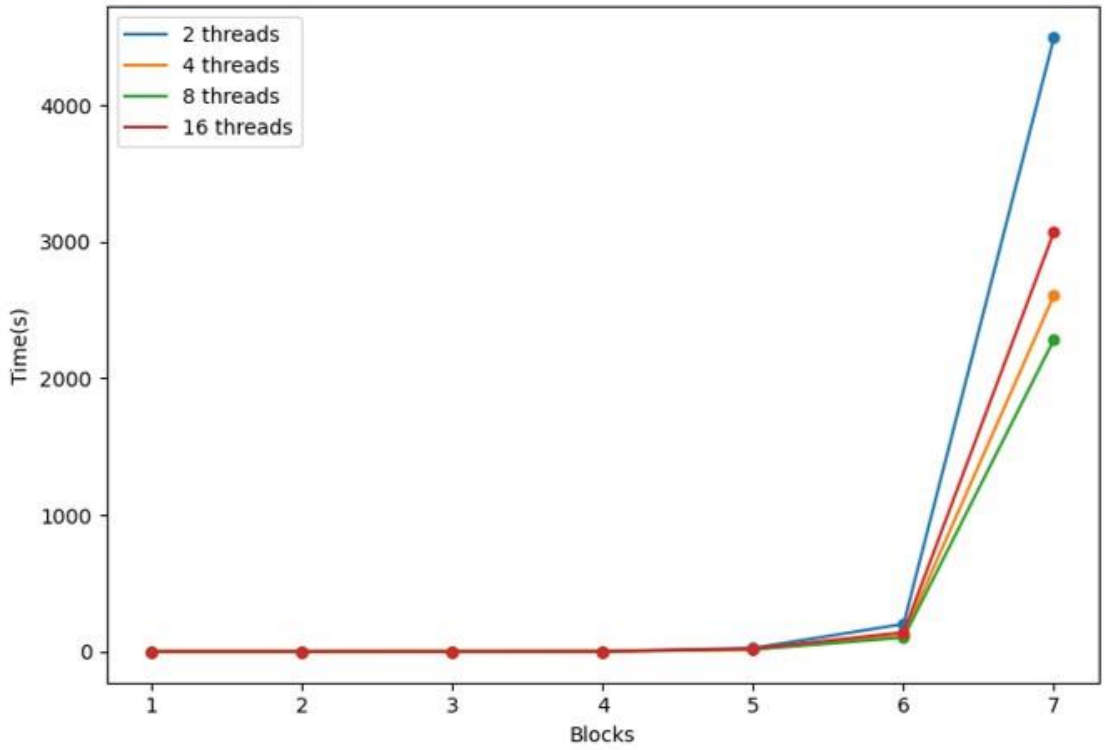


Fig 7: Results for default initilizeText and initializeHash values using 8 threads.

A blockchain of 6 blocks has been created within considerable amount of time for each value of entered number of threads. Block run times depending on the number of threads. Now, assume a CPU has 6 cores and 12 logical cores. Using 8 threads for this CPU is the most optimal solution.



Threads	Block1	Block2	Block3	Block4	Block5	Block6	Block7	Total
2	0.004	0.013	0.158	1.870	25.998	201	4497	1:18:47
4	0.004	0.014	0.171	1.354	16.500	117	2611	0:45:47
8	0.003	0.006	0.092	1.016	15.508	102	2286	0:40:05
16	0.251	0.192	0.834	1.400	21.695	140	3075	0:54:02

Fig 8: Blocks versus Computation time in seconds for a CPU with 6 cores and 12 logical cores

CONCLUSION

We successfully implemented Blockchain using OpenMP Multithreading. Though it is not distributed, it gives best results if used within a particular organization only. It reduces expenses for a Cloud or Distributed Network and limits it to a single machine without comprising much of the safety features. It also has a Low throughput on large volume of inputs

We have redesigned the traditional “Proof-of-Work” Algorithm to reduce some computation and better cryptographic algorithms to ensure maximum security and hence keep out validators. Our solution can work on all datatypes, be it transactional or stream of characters. But our Solution is very much dependent on Computer Specifications, so it is best advised to use Systems with higher specs.

REFERENCES

1. Zhang, Z., 2016. “*A multi-threaded cryptographic pseudorandom number generator test suite*.” Naval Postgraduate School Monterey United States.

2. Shae, Zonyin, and Jeffrey Tsai. "*Transform blockchain into distributed parallel computing architecture for precision medicine*." In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1290-1299. IEEE, 2018.

3. Samy, Hossam, Ashraf Tammam, Ahmed Fahmy, and Bahaa Hasan. "*Enhancing the performance of the blockchain consensus algorithm using multithreading technology*." *Ain Shams Engineering Journal* (2021).

4. Hazari, Shihab Shahriar. "*Design and development of a parallel Proof of Work for permissionless blockchain systems*." PhD diss., 2019.

APPENDIX-I

```
// Main.cpp
#include <stdlib.h>
#include <iostream>
#include "Sha.h"
#include "BlockChain.h"
#include <omp.h>
#include <iomanip>

using namespace std;

int main()
{
	string inputText = " Hello there! I am from team Alioth. Thank you for accepting this project ";	// ascii text
	string initializeHash = "0000000000000000000000000000000000000000000000000000000000000000";

	size_t threadNums;
	cout << "THREAD NUMS: ";
	cin >> threadNums;

	size_t nonce = 0;
	bool flag = false;
	size_t acceptNonce;			// acceptNonce will be approved by other threads
	size_t acceptCounter;		// accepted threads numbers

	WORD* sha256K = InitializeK();
	BlockChain blockChain;
	blockChain.AddBack(inputText, initializeHash);
	cout << "\n# BLOCK: " << blockChain.current->blockNum << "\t tid: " << omp_get_thread_num() << endl;
	cout << "Hash initialized: " << blockChain.current->initializeHash << "\t nonce: " << nonce << "\t \t tid: " << omp_get_thread_num() << endl;

	size_t entryCounter;
	omp_lock_t lock;
	omp_init_lock(&lock);
	double start = omp_get_wtime(), end;
	double globalTime = start;
	int sec, h, m;

#pragma omp parallel num_threads(threadNums)
	{
```

```
                string privateText, privateHash;
                size_t privateNonce;

#pragma omp critical
                {
                        privateNonce = nonce;
                        nonce++;
                }
#pragma omp barrier

                while (true)
                {
                        privateText = blockChain.GetText(privateNonce);          // text generate
for SHA256
                        privateHash = Sha256(privateText, sha256K);                      // hash
calculated

                        if (blockChain.Control(privateHash))
                        {
#pragma omp single nowait
                                {
                                        acceptCounter = 0;
                                        entryCounter = 0;
                                        acceptNonce = privateNonce;
                                        flag = true;
                                        while (acceptCounter < threadNums / 2) {}          // must
accepted N/2 threads
                                        while (entryCounter < threadNums - 1) {}           // wait
threadNums-1 threads

                                        // privateNonce accepted for other threads, print block in-
formations
                                        end = omp_get_wtime();
                                        cout << "Hash found: " << privateHash << "\t \t nonce: "
<< acceptNonce << "\t \t tid: " << omp_get_thread_num() << endl << "Text: " << privateText
<< endl << "Block Run-time(s): " << std::setprecision(3) << fixed << (end - start);
                                        start = omp_get_wtime();

                                        sec = int(end - globalTime);
                                        h = int(sec / 3600);
                                        m = int((sec - (3600 * h)) / 60);
                                        cout << "\t Total Run-time: " << h << ":" << m << ":" <<
(sec - (3600 * h) - (m * 60)) << endl;

                                        blockChain.AddBack(privateText, privateHash);
                // add next block
```

```
                                        nonce = 0;
                                        privateNonce = nonce;
                                        nonce++;

                                        cout << endl << endl << "# BLOCK: " << block-
Chain.blockCounter << "\t tid: " << omp_get_thread_num() << endl;
                                        cout << "Hash initialized: " << blockChain.current-
>initializeHash << "\t nonce: " << privateNonce << "\t \t tid: " << omp_get_thread_num() <<
endl;

                                        flag = false;
                                }
                        }
                        else {
                                // nonce couldnt found, nonce increment
                                omp_set_lock(&lock);
                                privateNonce = nonce;
                                nonce++;
                                omp_unset_lock(&lock);
                        }

                        if (flag)
                        {
                                privateText = blockChain.GetText(acceptNonce);
                                privateHash = Sha256(privateText, sha256K);

#pragma omp critical
                                {
                                        if (acceptCounter < threadNums / 2)                     // must
accepted N/2 threads => accepted total: N/2 + 1(main thread) threads
                                        {
                                                if (blockChain.Control(privateHash)) {
                                                        cout << "Hash accepted: " << privateHash
<< "\t \t nonce: " << acceptNonce << "\t \t tid: " << omp_get_thread_num() << endl;
                                                        ++acceptCounter;
                                                }
                                                else cout << "Error tid: " << omp_get_thread_num()
<< endl;
                                        }
                                        ++entryCounter;
                                }
                                while (flag) {}

                                omp_set_lock(&lock);
                                privateNonce = nonce;
                                nonce++;
```

```
                                omp_unset_lock(&lock);
                        }
                }
        }

        omp_destroy_lock(&lock);
        return 0;
}


//Sha.cpp
#include <cstring>
#include <fstream>
#include "Sha.h"
#pragma warning(disable:4996)

#define BLOCKSIZE 64
#define ROTRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))
#define CH(a,b,c) (((a) & (b)) ^ (~(a) & (c)))
#define MAJ(a,b,c) (((a) & (b)) ^ ((a) & (c)) ^ ((b) & (c)))
#define SIGMA0(a) (ROTRIGHT(a,7) ^ ROTRIGHT(a,18) ^ ((a) >> 3))
#define SIGMA1(a) (ROTRIGHT(a,17) ^ ROTRIGHT(a,19) ^ ((a) >> 10))
#define SUM0(a) (ROTRIGHT(a,2) ^ ROTRIGHT(a,13) ^ ROTRIGHT(a,22))
#define SUM1(a) (ROTRIGHT(a,6) ^ ROTRIGHT(a,11) ^ ROTRIGHT(a,25))
#define WORDTOCHAR(a, str)                          \
{                                                   \
   *((str) + 3) = (unsigned char) ((a)      );      \
   *((str) + 2) = (unsigned char) ((a) >>  8);      \
   *((str) + 1) = (unsigned char) ((a) >> 16);      \
   *((str) + 0) = (unsigned char) ((a) >> 24);      \
}
#define CHARTOWORD(str, x)                  \
{                                           \
   *(x) =   ((WORD) *((str) + 3)      )    \
          | ((WORD) *((str) + 2) <<  8)    \
          | ((WORD) *((str) + 1) << 16)    \
          | ((WORD) *((str) + 0) << 24);   \
}

WORD* InitializeK() {

   WORD* k = new WORD[64]
   { 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
```

```
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
    0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
    0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
    0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
    0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
    0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2 };

   return k;
}

void WordExtend(WORD* words) {

   for (size_t i = 16; i < 64; i++)
      words[i] = SIGMA1(words[i - 2]) + words[i - 7] + SIGMA0(words[i - 15]) + words[i - 16];
}

void WordCompress(WORD* abcdefgh, WORD* sha256K, WORD* expandedWords) {

   WORD temp1, temp2;
   for (int i = 0; i < 64; i++) {
      temp1 = abcdefgh[7] + SUM1(abcdefgh[4]) + CH(abcdefgh[4], abcdefgh[5], abcdefgh[6])
+ sha256K[i] + expandedWords[i];
      temp2 = SUM0(abcdefgh[0]) + MAJ(abcdefgh[0], abcdefgh[1], abcdefgh[2]);
      abcdefgh[7] = abcdefgh[6];
      abcdefgh[6] = abcdefgh[5];
      abcdefgh[5] = abcdefgh[4];
      abcdefgh[4] = abcdefgh[3] + temp1;
      abcdefgh[3] = abcdefgh[2];
      abcdefgh[2] = abcdefgh[1];
      abcdefgh[1] = abcdefgh[0];
      abcdefgh[0] = temp1 + temp2;
   }
}

void Transform(const unsigned char* message, WORD blockNum, WORD * sha256K, WORD*
sha256H)
{
   WORD expandedWords[64];
   WORD abcdefgh[8];

   const unsigned char* tempBlock;
```

```
    int i,j;
    for (i = 0; i < (int)blockNum; i++)
    {
        tempBlock = message + (i << 6);
        for (j = 0; j < 16; j++) {
            CHARTOWORD(&tempBlock[j << 2], &expandedWords[j]);
        }
        // kelime genisletme algoritmasi
        WordExtend(expandedWords);

        for (j = 0; j < 8; j++) {
            abcdefgh[j] = sha256H[j];
        }
        // word compress algoritmasi
        WordCompress(abcdefgh, sha256K, expandedWords);

        for (j = 0; j < 8; j++) {
            sha256H[j] += abcdefgh[j];
        }
    }
}

void Update(const unsigned char* message, WORD len, WORD* sha256K, WORD* sha256H,
unsigned char* msgBlock, WORD& msgTotalLen,
WORD& msgLen)
{
    WORD blockNum, remLen, tempLen;
    const unsigned char* shiftedMsg;
    tempLen = BLOCKSIZE - msgLen;
    remLen = len < tempLen ? len : tempLen;

    memcpy(&msgBlock[msgLen], message, remLen);
    if (msgLen + len < BLOCKSIZE) {
        msgLen += len;
        return;
    }

    len -= remLen;
    blockNum = len / BLOCKSIZE;
    shiftedMsg = message + remLen;

    Transform(msgBlock, 1, sha256K, sha256H);
    Transform(shiftedMsg, blockNum, sha256K, sha256H);
    remLen = len % BLOCKSIZE;
    memcpy(msgBlock, &shiftedMsg[blockNum << 6], remLen);
    msgLen = remLen;
```

```
    msgTotalLen += (blockNum + 1) << 6;
}

void Final(unsigned char* digest, WORD* sha256K, WORD* sha256H, unsigned char*
msgBlock, WORD& msgTotalLen, WORD& msgLen)
{
    WORD blockNum, tempLen, lenB;

    blockNum = (1 + ((BLOCKSIZE - 9) < (msgLen % BLOCKSIZE)));

    lenB = (msgTotalLen + msgLen) << 3;
    tempLen = blockNum << 6;

    memset(msgBlock + msgLen, 0, tempLen - msgLen);
    msgBlock[msgLen] = 0x80;

    WORDTOCHAR(lenB, msgBlock + tempLen - 4);
    Transform(msgBlock, blockNum, sha256K, sha256H);

    for (int i = 0; i < 8; i++)
        WORDTOCHAR(sha256H[i], &digest[i << 2]);

}

std::string Sha256(std::string input, WORD* sha256K)
{
    unsigned char *digest = new unsigned char[32]();
    unsigned char msgBlock[128];
    WORD msgTotalLen = 0, msgLen = 0;
    WORD sha256H[8]{ 0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f,
0x9b05688c, 0x1f83d9ab, 0x5be0cd19 };

    Update((unsigned char*)input.c_str(), input.length(), sha256K, sha256H, msgBlock, msgTo-
talLen, msgLen);
    Final(digest, sha256K, sha256H, msgBlock, msgTotalLen, msgLen);

    char buf[65];
    buf[64] = 0;
    for (int i = 0; i < 32; i++)
        sprintf(buf + i * 2, "%02x", digest[i]);

    delete[] digest;
    return std::string(buf);
}
```