1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **«Институт Компьютерных наук и кибербезопасности»**
5. **Высшая школа кибербезопасности и защиты информациия**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «**Исследование CUDA на примере хэширования алгоритмом MD5**»
2. по дисциплине «Языки программирования»
3. Выполнил
4. студент гр. 5131001/20003 Черникова В.М.

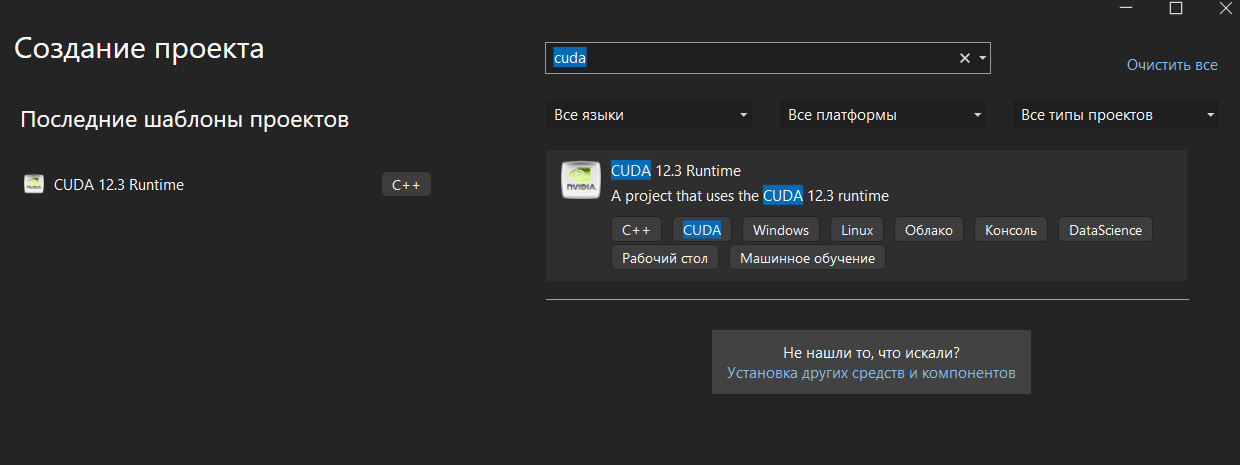
<*подпись*>

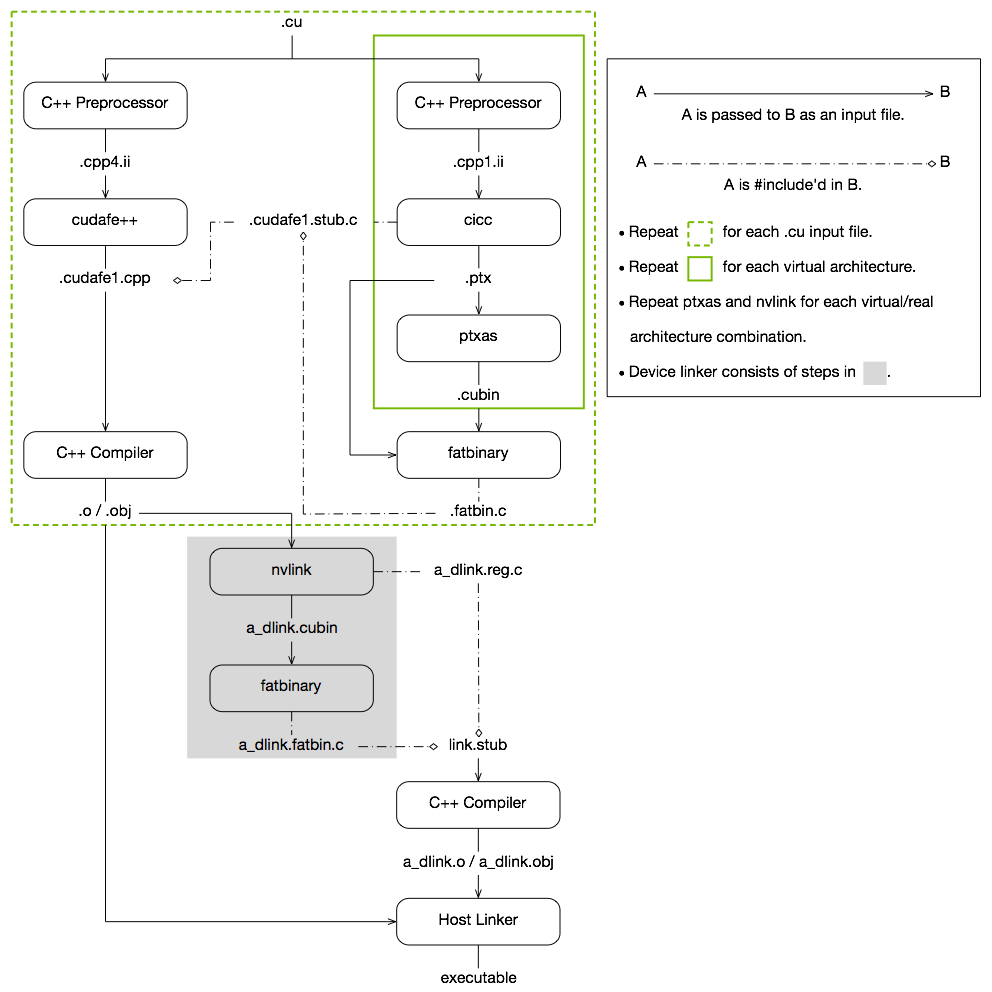
1. Преподаватель
2. Семьянов. П. В.

<*подпись*>

1. Санкт-Петербург
2. 2023
3. Начало

В работе была выбрана технология CUDA, так как OpenCL более сложен в установке, хотя может быть использован на любой видеокарте. Перед самой работы надо было проверить версию драйвера и скачать компилятор nvcc. После этого в Visual Studio появился шаблон проекта с готовыми настройками.

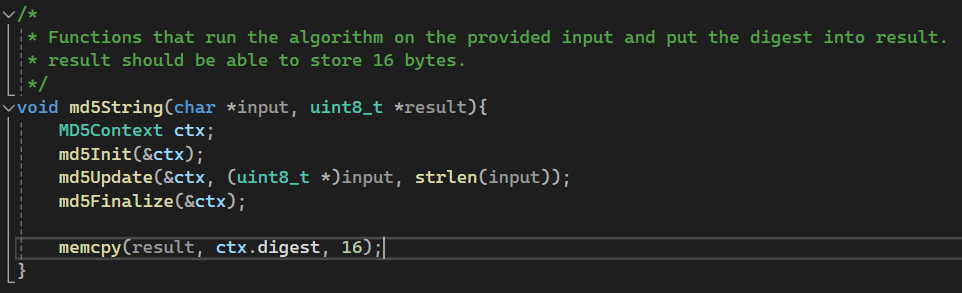




1. Основная часть
   1. Создание кода на CPU

* Поиск алгоритма MD5

Была найдена готовая библиотека для языка си на <https://github.com/Zunawe/md5-c>, которая включает в себя заголовочный файл и файл си. Главная функция, которая считает хэш - **md5String**(char \*input, uint8\_t \*result).



* Чтение из файла и подготовка исходных данных

Все входные и выходные данные представлены в виде развернутой строки, а не двумерного массива. Максимальная длинна слова и их количество выбирается в коде.

result = (uint8\_t\*)malloc(16\*n);

stroka = (char\*)malloc(n\*len\_slov);

memset(result, 0, 16\*n);

int i = 0;

//чтение из файла и запись в строку

{

FILE\* mf; // Переменная, в которую поочередно будут помещаться считываемые строки

char str[25]; //Указатель, в который будет помещен адрес массива, в который считана строка, или NULL если достигнут коней файла или произошла ошибка

char\* estr; // Открытие файла с режимом доступа «только чтение» и привязка к нему потока данных

printf("Открытие файла : ");

mf = fopen("slovar-13953518.txt", "r");

…

//Если файл не закончился, и не было ошибки чтения выводим считанную строку на экран

// i - итератор по кол-ву слов

int k = 0;//итератор по готовому слову

for (int j = 0; j < strlen(str) - 1; j++, k++)

{

if (str[j] != '\n') {

\*(stroka + i \* len\_slov + k) = str[j];

}

}

while (k < len\_slov)

{

\*(stroka + i \* len\_slov + k) = 0;

k++;

}

i++;

}

// Закрываем файл

printf("Закрытие файла : ");

if (fclose(mf) == EOF) printf("ошибка\n");

else printf("выполнено\n");

}

* Вычисления на CPU и подсчет времени

clock\_t start\_time = clock();

//-----------------

for (i = 0; i < n; i++)

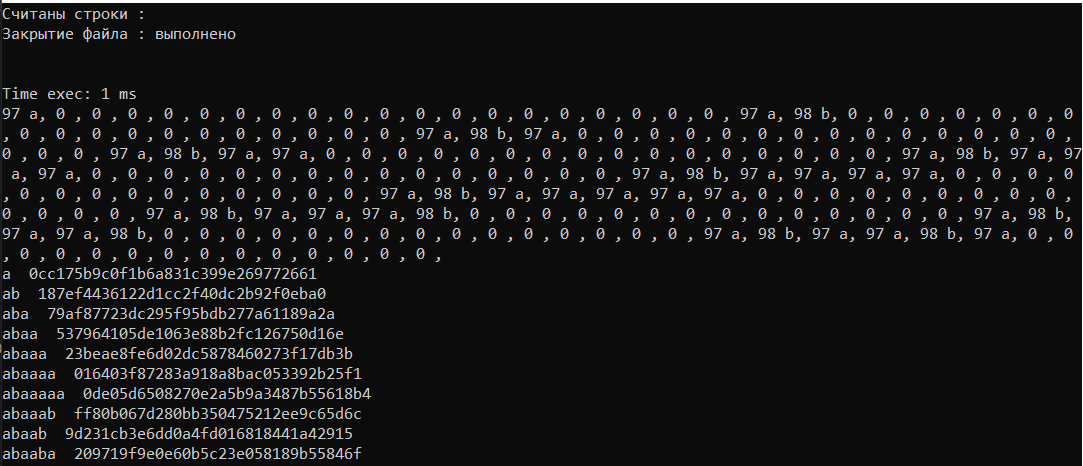
{

md5String((stroka + i \* len\_slov), (result+i\*16));

}

clock\_t end\_time = clock();

* Вывод в командной строке после работы программы:



Для каждой строки выделяется память размером len\_slov, и оставшееся место после строки заполняется нулями.

* 1. Создание кода на GPU
* Считывание и оформление данных выполнено так же, как для кода CPU.
* Оформление работы с CUDA выполнены в дополнительном блоке **addWithCuda ()**

cudaError\_t addWithCuda(char \*a, uint8\_t\* result, unsigned long long size)

{

char \*dev\_a = 0;

uint8\_t \*dev\_res = 0;

cudaError\_t cudaStatus;

// Choose which GPU to run on, change this on a multi-GPU system.

cudaStatus = cudaSetDevice(0);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU installed?");

goto Error;

}

// Allocate GPU buffers for three vectors (two input, one output) .

cudaStatus = cudaMalloc((void\*\*)&dev\_a, size \* sizeof(char) \* len\_slov);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_res, size \* sizeof(uint8\_t)\*16);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

// Copy input vectors from host memory to GPU buffers.

cudaStatus = cudaMemcpy(dev\_a, a, size \* sizeof(char) \* len\_slov, cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy failed!");

goto Error;

}

// Launch a kernel on the GPU with one thread for each element.

addKernel<<<(n + BLOCK\_SIZE) / BLOCK\_SIZE, BLOCK\_SIZE >>>(dev\_a, dev\_res);

// Check for any errors launching the kernel

cudaStatus = cudaGetLastError();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));

goto Error;

}

// cudaDeviceSynchronize waits for the kernel to finish, and returns any errors encountered during the launch.

cudaStatus = cudaDeviceSynchronize();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);

goto Error;

}

cudaStatus = cudaMemcpy(result, dev\_res, size \* sizeof(uint8\_t)\*16, cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy failed!");

goto Error;

}

Error:

cudaFree(dev\_a);

cudaFree(dev\_res);

return cudaStatus;

}

* Вызов функции CUDA в функции main (host), вывод получившихся данных и завершение программы

start\_time = clock();

// Add vectors in parallel.

cudaError\_t cudaStatus = addWithCuda(stroka, result, arraySize);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "addWithCuda failed!");

return 1;

}

end\_time = clock();

printf("\nTime exec: %0.f ms\n", (double)(end\_time - start\_time));

//консоль 10 первых слов

{

for (i = 0; i < 10; i++)

{

printf("%s ", (stroka + i \* len\_slov));

printf(" ");

print\_hash(result + i \* 16);

}

}

//запись все в файл

printf("Запись выходных данных\n");

FILE\* out\_file = fopen("output.txt", "w");

{

for (int i = 0; i < n; i++) {

//printf("%s ", (c + i \* 20));

fprintf(out\_file, "%s ", (stroka + i \* len\_slov));

fprint\_hash(out\_file, (result + i \* 16));

}

}

fclose(out\_file);

printf("Запись выполнена\n");

printf("\nTime exec: %0.f ms\n", (double)(end\_time - start\_time));

// cudaDeviceReset must be called before exiting in order for profiling and tracing tools such as Nsight and Visual Profiler to show complete traces.

cudaStatus = cudaDeviceReset();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaDeviceReset failed!");

return 1;

}

free(result);

free(stroka);

return 0;

}

* Программа ядра (device)

\_\_global\_\_ void addKernel(char \*a, uint8\_t\* res)

{

long long i = threadIdx.x + blockIdx.x \* blockDim.x;

if (i < n)

{

md5String (a + i \* len\_slov, res + i \*16);

\_\_syncthreads();

}

}

* Дополнительная функция для вывода хэша, ее можно использовать и на device, и на host

\_\_device\_\_ \_\_host\_\_ void print\_hash(uint8\_t\* p) {

for (unsigned int i = 0; i < 16; ++i) {

printf("%02x", p[i]);

}

printf("\n");

}

1. Анализ работы

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **len\_slov** | **CPU** | **GPU** |
| 10000 | 10 | 1 | 105 |
| 10000 | 20 | 1 | 111 |
| 10000 | 25 | 1 | 108 |
| 10000 | 30 | 2 | 93 |
| 10000 | 40 | 2 | 91 |
| 100000 | 10 | 151 | 96 |
| 100000 | 20 | 142 | 84 |
| 100000 | 25 | 143 | 92 |
| 100000 | 30 | 150 | 98 |
| 100000 | 40 | 151 | 114 |
| 1000000 | 10 | 3894 | 120 |
| 1000000 | 20 | 1431 | 99 |
| 1000000 | 25 | 1426 | 102 |
| 1000000 | 30 | 1430 | 109 |
| 1000000 | 40 | 11419 | 105 |
| 10000000 | 20 | 1E+22 | 191 |
| 10000000 | 25 | 1E+19 | 211 |
| 214194124 | 30 | 1E+48 | 15400 |

1. Вывод

В ходе работы был изучены основы использования технологии CUDA и выполнено сравнение времени работы алгоритма MD5 на CPU и GPU. При количестве входных данных более 100000 лучше использовать вычисления на видеокарте, они быстрее и стабильнее работают при больших объемах данных.

1. Приложение

Для использования библиотеки на GPU понадобилось дописать \_\_device\_\_ для каждой функции и используемых данных.

/\*

\* Derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm

\* and modified slightly to be functionally identical but condensed into control structures.

\*/

#include "md5.h"

/\*

\* Constants defined by the MD5 algorithm

\*/

#define A 0x67452301

#define B 0xefcdab89

#define C 0x98badcfe

#define D 0x10325476

static uint32\_t \_\_device\_\_ S[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,

4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21};

static uint32\_t \_\_device\_\_ K[] = {0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,

0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,

0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,

0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,

0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,

0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,

0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,

0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,

0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,

0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,

0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,

0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,

0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,

0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,

0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,

0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};

/\*

\* Padding used to make the size (in bits) of the input congruent to 448 mod 512

\*/

static uint8\_t \_\_device\_\_ PADDING[] = {0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

/\*

\* Bit-manipulation functions defined by the MD5 algorithm

\*/

#define F(X, Y, Z) ((X & Y) | (~X & Z))

#define G(X, Y, Z) ((X & Z) | (Y & ~Z))

#define H(X, Y, Z) (X ^ Y ^ Z)

#define I(X, Y, Z) (Y ^ (X | ~Z))

/\*

\* Rotates a 32-bit word left by n bits

\*/

\_\_device\_\_ uint32\_t rotateLeft(uint32\_t x, uint32\_t n){

return (x << n) | (x >> (32 - n));

}

/\*

\* Initialize a context

\*/

\_\_device\_\_ void md5Init(MD5Context \*ctx){

ctx->size = (uint64\_t)0;

ctx->buffer[0] = (uint32\_t)A;

ctx->buffer[1] = (uint32\_t)B;

ctx->buffer[2] = (uint32\_t)C;

ctx->buffer[3] = (uint32\_t)D;

}

/\*

\* Add some amount of input to the context

\*

\* If the input fills out a block of 512 bits, apply the algorithm (md5Step)

\* and save the result in the buffer. Also updates the overall size.

\*/

\_\_device\_\_ void md5Update(MD5Context \*ctx, uint8\_t \*input\_buffer, size\_t input\_len){

uint32\_t input[16];

unsigned int offset = ctx->size % 64;

ctx->size += (uint64\_t)input\_len;

// Copy each byte in input\_buffer into the next space in our context input

for(unsigned int i = 0; i < input\_len; ++i){

ctx->input[offset++] = (uint8\_t)\*(input\_buffer + i);

// If we've filled our context input, copy it into our local array input

// then reset the offset to 0 and fill in a new buffer.

// Every time we fill out a chunk, we run it through the algorithm

// to enable some back and forth between cpu and i/o

if(offset % 64 == 0){

for(unsigned int j = 0; j < 16; ++j){

// Convert to little-endian

// The local variable `input` our 512-bit chunk separated into 32-bit words

// we can use in calculations

input[j] = (uint32\_t)(ctx->input[(j \* 4) + 3]) << 24 |

(uint32\_t)(ctx->input[(j \* 4) + 2]) << 16 |

(uint32\_t)(ctx->input[(j \* 4) + 1]) << 8 |

(uint32\_t)(ctx->input[(j \* 4)]);

}

md5Step(ctx->buffer, input);

offset = 0;

}

}

}

/\*

\* Pad the current input to get to 448 bytes, append the size in bits to the very end,

\* and save the result of the final iteration into digest.

\*/

\_\_device\_\_ void md5Finalize(MD5Context \*ctx){

uint32\_t input[16];

unsigned int offset = ctx->size % 64;

unsigned int padding\_length = offset < 56 ? 56 - offset : (56 + 64) - offset;

// Fill in the padding and undo the changes to size that resulted from the update

md5Update(ctx, PADDING, padding\_length);

ctx->size -= (uint64\_t)padding\_length;

// Do a final update (internal to this function)

// Last two 32-bit words are the two halves of the size (converted from bytes to bits)

for(unsigned int j = 0; j < 14; ++j){

input[j] = (uint32\_t)(ctx->input[(j \* 4) + 3]) << 24 |

(uint32\_t)(ctx->input[(j \* 4) + 2]) << 16 |

(uint32\_t)(ctx->input[(j \* 4) + 1]) << 8 |

(uint32\_t)(ctx->input[(j \* 4)]);

}

input[14] = (uint32\_t)(ctx->size \* 8);

input[15] = (uint32\_t)((ctx->size \* 8) >> 32);

md5Step(ctx->buffer, input);

// Move the result into digest (convert from little-endian)

for(unsigned int i = 0; i < 4; ++i){

ctx->digest[(i \* 4) + 0] = (uint8\_t)((ctx->buffer[i] & 0x000000FF));

ctx->digest[(i \* 4) + 1] = (uint8\_t)((ctx->buffer[i] & 0x0000FF00) >> 8);

ctx->digest[(i \* 4) + 2] = (uint8\_t)((ctx->buffer[i] & 0x00FF0000) >> 16);

ctx->digest[(i \* 4) + 3] = (uint8\_t)((ctx->buffer[i] & 0xFF000000) >> 24);

}

}

/\*

\* Step on 512 bits of input with the main MD5 algorithm.

\*/

\_\_device\_\_ void md5Step(uint32\_t \*buffer, uint32\_t \*input){

uint32\_t AA = buffer[0];

uint32\_t BB = buffer[1];

uint32\_t CC = buffer[2];

uint32\_t DD = buffer[3];

uint32\_t E;

unsigned int j;

for(unsigned int i = 0; i < 64; ++i){

switch(i / 16){

case 0:

E = F(BB, CC, DD);

j = i;

break;

case 1:

E = G(BB, CC, DD);

j = ((i \* 5) + 1) % 16;

break;

case 2:

E = H(BB, CC, DD);

j = ((i \* 3) + 5) % 16;

break;

default:

E = I(BB, CC, DD);

j = (i \* 7) % 16;

break;

}

uint32\_t temp = DD;

DD = CC;

CC = BB;

BB = BB + rotateLeft(AA + E + K[i] + input[j], S[i]);

AA = temp;

}

buffer[0] += AA;

buffer[1] += BB;

buffer[2] += CC;

buffer[3] += DD;

}

/\*

\* Functions that run the algorithm on the provided input and put the digest into result.

\* result should be able to store 16 bytes.

\*/

\_\_device\_\_ void md5String(char \*input, uint8\_t \*result){

MD5Context ctx;

md5Init(&ctx);

size\_t len = 0;

while (input[len]) {

len++;

}

md5Update(&ctx, (uint8\_t \*)input, len);

md5Finalize(&ctx);

memcpy(result, ctx.digest, 16);

}

void md5File(FILE \*file, uint8\_t \*result){

char \*input\_buffer = (char\*)malloc(1024);

size\_t input\_size = 0;

MD5Context ctx;

md5Init(&ctx);

while((input\_size = fread(input\_buffer, 1, 1024, file)) > 0){

md5Update(&ctx, (uint8\_t \*)input\_buffer, input\_size);

}

md5Finalize(&ctx);

free(input\_buffer);

memcpy(result, ctx.digest, 16);

}