Capitolo 5

Codice parallelizzato: versione con openCL

5.1 Introduzione

In questo capitolo verrà presentato il secondo dei due codici parallelizzati. In particolare verranno descritti il funzionamento generale e di seguito saranno presentate le parti del codice cambiate rispetto al programma sequenziale.

5.2 Funzionamento Generale e Diagrammi di Flusso

A differenza delle situazioni precedenti, il programma è composto da 5 moduli. Oltre ai 4 precedenti, si aggiunge anche un altro: device.c.

Le parti parallelizzate con l'ausilio di openCL sono essenzialmente 2. La prima sostituisce il metodo sequenziale encode, quindi ha il compito di generare le diverse permutazioni.

Il kernel, che sostituisce encode, viene aperto tramite il metodo chiamaEncode contenuto in device.c. Esso viene chiamato nella funzione analyze, molto simile al precedente analyzeLength, contenuta nel modulo enum. Il codice del kernel è scritto in encode.cl. Ogni thread di openCL si occuperà di una cella dell’array risultato, buffer. Esso avrà una misura variabile, infatti dipenderà dalla lunghezza della password non codificata che si sta esaminando, da un minimo di 1 e un massimo di 10. Alla fine si avrà una permutazione memorizzata nell’array buffer. Dopo averla ottenuta, chiamaEncode termina e si ritorna in analyze. Le istruzioni successive saranno codificare la permutazione con md5 e trasformare il digest risultante in stringa. Di seguito si chiama il metodo find, che come nei casi precedenti si trova in sort.c. Questo chiama la seconda parte parallelizzata con openCL: il confronto tra la permutazione e l’array di password. Il codice del kernel è scritto in kernel.cl. In questo caso, ogni thread openCL si occuperà di una password e di una cella del risultato. Nella situazione in cui la combinazione esaminata e la password ad esso associato siano uguali, la cella sarà settata a 0, altrimenti ad 1. Alla fine dell’esecuzione del kernel, in chiamaKernel, sarà effettuata la verifica dell’array risultato: se è presente uno 0 allora la password è stata trovata e quindi si può procedere alla rimozione, altrimenti si ritorna in analyze e si procede con un’altra permutazione.  
Il resto del codice procede come nel frangente sequenziale.

5.3 Device.c

Questo modulo contiene solo il metodo chiamaEncode, che ha il compito di aprire il kernel, encode.cl, e creare tutti gli elementi necessari per far funzionare il framework.

5.3.1 chiamaEncode

**void chiamaEncode(uint8\_t\* buffer, uint8\_t\* alphabet, size\_t charsetLength, size\_t stringLength, uint64\_t counter){**

viene caricato il kernel e copiato nell’array source\_str

**FILE \*fp;**

**char \*source\_str;**

**size\_t source\_size;**

**fp = fopen("encode.cl", "r");**

**if (!fp) {**

**printf("Failed to load kernel.\n");**

**exit(1);**

**}**

**source\_str = (char\*)malloc(MAX\_SOURCE\_SIZE);**

**source\_size = fread(source\_str, 1, MAX\_SOURCE\_SIZE, fp);**

**fclose( fp );**

**source\_str[source\_size] = '\0';**

vengono prese le informazioni sulla piattaforma e i device

**cl\_platform\_id platform\_id;**

**cl\_device\_id device\_id;**

**cl\_int ret = clGetPlatformIDs(1, &platform\_id, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clGetPlatformIDs\n");**

**exit(1);**

**}**

**ret = clGetDeviceIDs(platform\_id, CL\_DEVICE\_TYPE\_GPU, 1,**

**&device\_id, NULL);**

**if (ret != CL\_SUCCESS) {**

**ret = clGetDeviceIDs(platform\_id, CL\_DEVICE\_TYPE\_CPU, 1,**

**&device\_id, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clGetDeviceIDs\n");**

**exit(1);**

**}**

**}**

viene creato il contesto di OpenCL

**cl\_context context = clCreateContext(0, 1, &device\_id, NULL, NULL, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in cl\_context\n");**

**exit(1);**

**}**

**cl\_command\_queue command\_queue = clCreateCommandQueueWithProperties(context, device\_id, 0, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in cl\_command\_queue\n");**

**exit(1);**

**}**

vengono create i buffer di memoria sul device per ogni array necessario al kernel

**cl\_mem b\_mem\_obj = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY,**

**stringLength\*sizeof(uint8\_t), NULL, NULL);**

**if (!b\_mem\_obj) {**

**printf("error in cl\_mem b\n");**

**exit(1);**

**}**

**cl\_mem alfa\_mem\_obj = clCreateBuffer(context, CL\_MEM\_READ\_ONLY,**

**36 \* sizeof(uint8\_t), NULL, NULL);**

**if (!alfa\_mem\_obj) {**

**printf("error in cl\_mem alfa\n");**

**exit(1);**

**}**

l’array contenente l’alfabeto viene copiato nella memoria di device create precedentemente  **ret = clEnqueueWriteBuffer(command\_queue, alfa\_mem\_obj, CL\_TRUE, 0,**

**36 \* sizeof(uint8\_t), alphabet, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueWriteBuffer\n");**

**exit(1);**

**}**

creazione del programma

**cl\_program program = clCreateProgramWithSource(context, 1,**

**(const char \*\*)&source\_str, NULL, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clCreateProgram\n");**

**exit(1);**

**}**

**ret = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**size\_t len;**

**char buff[16384];**

**printf ("failed to build program executable\n");**

**clGetProgramBuildInfo(program, device\_id,**

**CL\_PROGRAM\_BUILD\_LOG, sizeof(buff), buff, &len); exit(1);**

**}**

aperture del kernel

**cl\_kernel kernel = clCreateKernel(program, "encode\_k", &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in cl\_kernel\n");**

**exit(1);**

**}**

settaggio dei parametri necessary al kernel

**long slen = (long) stringLength;**

**long clen = (long) charsetLength;**

**long c = (long) counter;**

**ret = clSetKernelArg(kernel, 0, sizeof(cl\_mem), (void \*)&b\_mem\_obj);**

**ret |= clSetKernelArg(kernel, 1, sizeof(cl\_mem), (void \*)&alfa\_mem\_obj);**

**ret |= clSetKernelArg(kernel, 2, sizeof(long), (void \*)&clen);**

**ret |= clSetKernelArg(kernel, 3, sizeof(long), (void \*)&slen);**

**ret |= clSetKernelArg(kernel, 4, sizeof(long), (void \*)&c);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clSetKernelArg\n");**

exit(1);

}

esecuzione del kernel

**size\_t global\_item\_size = stringLength;**

**size\_t local\_item\_size = 8;**

**global\_item\_size = (global\_item\_size + local\_item\_size - 1) / local\_item\_size \* local\_item\_size;**

**ret = clEnqueueNDRangeKernel(command\_queue, kernel, 1, NULL,**

**&global\_item\_size, &local\_item\_size, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueNDRangeKernel\n");**

**exit(1);**

**}**

il risultato viene copiato in buffer

**ret = clEnqueueReadBuffer(command\_queue, b\_mem\_obj, CL\_TRUE, 0,**

**stringLength\*sizeof(uint8\_t), buffer, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueReadBuffer\n");**

**exit(1);**

**}**

liberazione risorse

**ret = clFlush(command\_queue);**

**ret = clFinish(command\_queue);**

**ret = clReleaseKernel(kernel);**

**ret = clReleaseProgram(program);**

**ret = clReleaseMemObject(b\_mem\_obj);**

**ret = clReleaseMemObject(alfa\_mem\_obj);**

**ret = clReleaseCommandQueue(command\_queue);**

**ret = clReleaseContext(context);**

**return;**

**}**

5.3.2 encode.cl

Encode.cl è il kernel aperto ed eseguito da encode.

Ogni thread di openCL controlla una cella del risultato.

**\_\_kernel void encode\_k (\_\_global char\* buffer, \_\_global const char\* alphabet, const long charsetLength, const long stringLength, long counter ) {**

**int i = get\_global\_id(0);**

**long c = counter;**

**if(i>= stringLength) return;**

**int j;**

**for (j = 0; j < i; j++) c /= charsetLength;**

**long a = a / charsetLength + (c % charsetLength);**

**a -= (a / charsetLength) \* charsetLength;**

**buffer[i] = alphabet[a];**

**counter/= charsetLength;**

**}**

5.4 Sort.c

In questo modulo, come nel caso descritto nel capitolo precedente, si trovano i metodi find e rimozione, in più è descritta anche la funzione che esegue il kernel di confronto tra stringhe: chiamaKernel.

5.4.1 Find

**size\_t find(char\*\* strings, char\* s, size\_t numLines, size\_t stringLength){**

**int flag ;**

chiama la funzione che eseguirà il kernel, il quale ha il compito di verificare la presenza della permutazione esaminata nell’array delle password

**flag = chiamaKernel(strings, s, numLines);**

Se il risultato è 0 allora la verifica è positive e quindi si può rimuovere la password decifrata dall’array tramite il metodo rimozione

**if (flag != 1){**

**numLines= rimozione(s, strings, numLines);**

**}**

**return numLines;**

**}**

Il metodo rimozione è uguale a quello visto nei casi precedenti**.**

5.4.2 chiamaKernel

Come chiamaEncode, chiamaKernel ha il compito di istanziare tutti gli elementi necessari per far fuzionare OpenCL. Il risultato sarà un flag, in caso la stringa sia presente sarà settato a 0, altrimenti a 1.

**int chiamaKernel(char\*\* hashes, char\* s, size\_t numLines){**

**int i;**

**int j;**

**int flag = 1;**

**int \*retu= (int\*)malloc(sizeof(int)\*LIST\_SIZE);**

**char \*C = (char\*)malloc(sizeof(char)\*LIST\_SIZE\*numLines);**

le password sono scritte con ordine row-major

**for (i = 0; i < numLines ; i++) {**

**for (j = 0; j < LIST\_SIZE; j++) {**

**C[i\*LIST\_SIZE+j] = hashes[i][j];**

**}**

**}**

**FILE \*fp;**

**char \*source\_str;**

**size\_t source\_size;**

**fp = fopen("kernel.cl", "r");**

**if (!fp) {**

**fprintf(stderr, "Failed to load kernel.\n");**

**exit(1);**

**}**

**source\_str = (char\*)malloc(MAX\_SOURCE\_SIZE);**

**source\_size = fread( source\_str, 1, MAX\_SOURCE\_SIZE, fp);**

**fclose( fp );**

**cl\_platform\_id platform\_id = NULL;**

**cl\_device\_id device\_id = NULL;**

**cl\_uint ret\_num\_devices;**

**cl\_uint ret\_num\_platforms;**

**cl\_int ret = clGetPlatformIDs(1, &platform\_id, &ret\_num\_platforms);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clGetPlatformIDs\n");**

**exit(1);**

**}**

**ret = clGetDeviceIDs(platform\_id, CL\_DEVICE\_TYPE\_ALL, 1,**

**&device\_id, &ret\_num\_devices);**

**if (ret != CL\_SUCCESS) {**

**ret = clGetDeviceIDs(platform\_id, CL\_DEVICE\_TYPE\_CPU, 1,**

**&device\_id, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clGetDeviceIDs\n");**

**exit(1);**

**}**

**}**

**cl\_context context = clCreateContext( NULL, 1, &device\_id, NULL, NULL, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in cl\_context\n");**

**exit(1);**

**}**

**cl\_command\_queue command\_queue = clCreateCommandQueueWithProperties(context, device\_id, 0, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in cl\_command\_queue\n");**

**exit(1);**

**}**

**cl\_mem ret\_mem\_obj = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY,**

**LIST\_SIZE \* sizeof(int), NULL, NULL);**

**if (!ret\_mem\_obj) {**

**printf("error in cl\_mem ret\n");**

**exit(1);**

**}**

**cl\_mem hash\_mem\_obj = clCreateBuffer(context, CL\_MEM\_READ\_ONLY,**

**LIST\_SIZE\* numLines \* sizeof(char), NULL, NULL);**

**if (!hash\_mem\_obj) {**

**printf("error in cl\_mem hash\n");**

**exit(1);**

**}**

**cl\_mem s\_mem\_obj = clCreateBuffer(context, CL\_MEM\_READ\_ONLY,**

**LIST\_SIZE \* sizeof(int), NULL, NULL);**

**if (!s\_mem\_obj) {**

**printf("error in cl\_mem s\n");**

**exit(1);**

**}**

**ret = clEnqueueWriteBuffer(command\_queue, hash\_mem\_obj, CL\_TRUE, 0,**

**LIST\_SIZE\* numLines \* sizeof(char), C, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueWriteBuffer\n");**

**exit(1);**

**}**

**ret = clEnqueueWriteBuffer(command\_queue, s\_mem\_obj, CL\_TRUE, 0,**

**LIST\_SIZE \* sizeof(char), s, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueWriteBuffer\n");**

**exit(1);**

**}**

**cl\_program program = clCreateProgramWithSource(context, 1,**

**(const char \*\*)&source\_str, (const size\_t \*)&source\_size, &ret);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clCreateProgram\n");**

**exit(1);**

**}**

**ret = clBuildProgram(program, 1, &device\_id, NULL, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**size\_t len;**

**char buff[16384];**

**printf ("failed to build program executable\n");**

**clGetProgramBuildInfo(program, device\_id,**

**CL\_PROGRAM\_BUILD\_LOG, sizeof(buff), buff, &len);**

**printf("%s\n", buff);**

**exit(1);**

**}**

**cl\_kernel kernel = clCreateKernel(program, "search", &ret);**

**ret = clSetKernelArg(kernel, 0, sizeof(cl\_mem), (void \*)&hash\_mem\_obj);**

**ret |= clSetKernelArg(kernel, 1, sizeof(cl\_mem), (void \*)&ret\_mem\_obj);**

**ret |= clSetKernelArg(kernel, 2, sizeof(cl\_mem), (void \*)&s\_mem\_obj);**

**ret |= clSetKernelArg(kernel, 3, sizeof(long), (void \*)&numLines);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clSetKernelArg\n");**

**exit(1);**

**}**

**size\_t global\_item\_size;**

**size\_t local\_item\_size = 8;**

**global\_item\_size = (global\_item\_size + local\_item\_size - 1) / local\_item\_size \* local\_item\_size;**

**ret = clEnqueueNDRangeKernel(command\_queue, kernel, 1, NULL,**

**&global\_item\_size, &local\_item\_size, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueNDRangeKernel\n");**

**exit(1);**

**}**

**ret = clEnqueueReadBuffer(command\_queue, ret\_mem\_obj, CL\_TRUE, 0,**

**LIST\_SIZE \* sizeof(int), retu, 0, NULL, NULL);**

**if (ret != CL\_SUCCESS) {**

**printf("error in clEnqueueReadBuffer\n");**

**exit(1);**

**}**

verifica del risultato finale

**int flagP =0;**

**for(j = 0; j < LIST\_SIZE; j++){**

**if(retu[j] != 0)**

**{**

**flagP = 1;**

**}**

**}**

**if(flagP == 1) flag=1;**

**else flag = 0;**

liberazione risorse

**ret = clFlush(command\_queue);**

**ret = clFinish(command\_queue);**

**ret = clReleaseKernel(kernel);**

**ret = clReleaseProgram(program);**

**ret = clReleaseMemObject(hash\_mem\_obj);**

**ret = clReleaseMemObject(s\_mem\_obj);**

**ret = clReleaseMemObject(ret\_mem\_obj);**

**ret = clReleaseCommandQueue(command\_queue);**

**ret = clReleaseContext(context);**

**free (retu);**

**return flag;**

**}**

5.4.2 kernel.cl

**\_\_kernel void search (\_\_global const char \*hashes, \_\_global int \*retu, \_\_global const char \*s, long numLines) {**

**int i = get\_global\_id(0);**

**if (i >= numLines) return;**

**int k;**

**for (k = 0; k< 33; k++){**

**if (hashes[i\* 33 + k] == s[k]) retu[k] = 0;**

**else{**

**retu[k] = 1;**

**}**

**}**

**}**