**Национальный Исследовательский Университет Информационных технологий, Механики и Оптики**

**Лабораторная работа № 4 по курсу «Параллельные вычисления»**

Выполнил:

Бедаш Д.С., гр. 5110

Проверил:

Соснин Владимир Валерьевич

Задание:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ФИО студента** | **Математическое ожидание** | **Дисперсия** | **Преобразование исходной матрицы** | **Способ формирования вектора с результатами** | **Метод сортировки вектора с результатами** |
| Бедаш Дмитрий Сергеевич | -45 | 65 | Умножение М1 на скаляр 5. | Выборочная дисперсия каждой строки | Shell sort |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 250 | 275 | 300 | 325 | 350 | 375 | 400 | 425 | 450 | 475 | 500 | 3000 |
| Delta\_ ms\_openmp\_best | 7303 | 10145 | 13473 | 15616 | 19855 | 32927 | 68732 | 98931 | 116154 | 155752 | 185752 |  |
| Delta\_ ms\_pthreads | 5708 | 7793 | 9603 | 13017 | 17819 | 23447 | 30207 | 49016 | 53411 | 83723 | 68427 |  |
| Delta\_ ms\_opencl | 1798 | 2071 | 2399 | 2742 | 3232 | 3468 | 3904 | 4360 | 5120 | 5840 | 6376 |  |
| SP\_opencl | 4,691324 | 5,665379 | 6,374323 | 7,805616 | 7,466275 | 10,49423 | 17,75179 | 16,07867 | 13,83574 | 16,93716 | 19,55693 | 36 |
| SP\_pthread | 1,477751 | 1,505582 | 1,592419 | 1,644234 | 1,354229 | 1,552182 | 2,29427 | 1,430206 | 1,3263 | 1,181432 | 1,822307 | 207042 |

**Характеристика системы**

**Процессор:**

model name : Intel(R) Core(TM) i5 CPU M 480 @ 2.67GHz

cpu MHz : 1197.000

cache size : 3072 KB

cpu cores : 2

clflush size : 64

cache\_alignment : 64

address sizes : 36 bits physical, 48 bits virtual

**ОЗУ – два модуля с одинаковыми параметрами:**

Memory Device

Array Handle: 0x001D

Error Information Handle: Not Provided

Total Width: 64 bits

Data Width: 64 bits

Size: 2048 MB

Form Factor: DIMM

Set: None

Locator: DIMM\_A

Bank Locator: DIMM\_A

Type: DDR3

Type Detail: Synchronous

Speed: 1333 MHz

Manufacturer: CE80

Serial Number: 9249EC3A

Asset Tag: 1051

Part Number: M471B5773DH0-CH9

Rank: 1

**OS:**

Description: Ubuntu 13.04

Codename: raring

Linux core version 3.8.0-35-generic

**GCC:**

COLLECT\_GCC=gcc

COLLECT\_LTO\_WRAPPER=/usr/lib/gcc/x86\_64-linux-gnu/4.7/lto-wrapper

Целевая архитектура: x86\_64-linux-gnu

Параметры конфигурации: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.7.3-1ubuntu1' --with-bugurl=file:///usr/share/doc/gcc-4.7/README.Bugs --enable-languages=c,c++,go,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.7 --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.7 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --enable-plugin --with-system-zlib --enable-objc-gc --with-cloog --enable-cloog-backend=ppl --disable-cloog-version-check --disable-ppl-version-check --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --with-tune=generic --enable-checking=release --build=x86\_64-linux-gnu --host=x86\_64-linux-gnu --target=x86\_64-linux-gnu

Модель многопоточности: posix

gcc версия 4.7.3 (Ubuntu/Linaro 4.7.3-1ubuntu1)

Вывод : В результате проведенной лабораторной работы был получен опыт распараллеливания при помощи opencl. Был достигнут значительный прирост производительности

Код Программы:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <sys/time.h>

#include <stdio.h>

#include <stdlib.h>

#include <gsl/gsl\_rng.h>

#include <gsl/gsl\_randist.h>

#include <math.h>

#include <sys/time.h>

#include <CL/cl.h>

#define MEM\_SIZE (128)

#define MAX\_SOURCE\_SIZE (0x100000)

void shellsort(float \*a,int n);

int main(int argc, char\* argv[])

{ int N;

N = atoi(argv[1]);

int width = N;

int height = N;

int i,j,k,r;

long delta\_ms;

gsl\_rng \* rgen = gsl\_rng\_alloc(gsl\_rng\_taus);

double a,b;

a=31,15;

b=-1,444445;

struct timeval T1, T2;

float \* A = (float \*)malloc(sizeof(float)\*width\*height);

float \* B = (float \*)malloc(sizeof(float)\*width\*height);

float \* C = (float \*)malloc(sizeof(float)\*width\*height);

float \* Res = (float \*)malloc(sizeof(float)\*width);

float \* D= (float \*)malloc(sizeof(float)\*width\*height);

cl\_device\_id device\_id = NULL;

cl\_context context = NULL;

cl\_command\_queue command\_queue = NULL;

cl\_mem memobjA = NULL;

cl\_mem memobjB = NULL;

cl\_mem memobjC = NULL;

cl\_mem memobjRes = NULL;

cl\_mem rowA = NULL;

cl\_mem colC = NULL;

cl\_program program = NULL;

cl\_kernel kernelMatrixMult = NULL;

cl\_kernel kernelVectMult = NULL;

cl\_kernel kernelVectSred = NULL;

cl\_platform\_id platform\_id = NULL;

cl\_uint ret\_num\_devices;

cl\_uint ret\_num\_platforms;

cl\_int ret;

//char string[MEM\_SIZE];

FILE \*fp;

char fileName[] = "./multi.cl";

char \*source\_str;

size\_t source\_size;

int row = width;

int col = height;

/\* Load the source code containing the kernel\*/

fp = fopen(fileName, "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 );

/\* Get Platform and Device Info \*/

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

ret = clGetDeviceIDs( platform\_id, CL\_DEVICE\_TYPE\_GPU, 1, &device\_id, &ret\_num\_devices);

/\* Create OpenCL context \*/

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

/\* Create Command Queue \*/

command\_queue = clCreateCommandQueue(context, device\_id, 0, &ret);

/\* Create Kernel Program from the source \*/

program = clCreateProgramWithSource(context, 1, (const char \*\*)&source\_str,(const size\_t \*)&source\_size, &ret);

/\* Build Kernel Program \*/

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

/\* Create OpenCL Kernel \*/

kernelMatrixMult = clCreateKernel(program, "matrixMultiplication", &ret);

kernelVectMult = clCreateKernel(program, "matrixVectorMultiplication", &ret);

kernelVectSred = clCreateKernel(program, "matrixVectorSred", &ret);

/\* Create Memory Buffer \*/

memobjA = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, width \* height \* sizeof(float), NULL, &ret);

memobjB = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, width \* height \* sizeof(float), NULL, &ret);

memobjC = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, width \* height \* sizeof(float), NULL, &ret);

memobjRes = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, width \* sizeof(float), NULL, &ret);

rowA = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, sizeof(int), NULL, &ret);

colC = clCreateBuffer(context, CL\_MEM\_READ\_WRITE, sizeof(int), NULL, &ret);

gettimeofday(&T1, NULL);

printf("Started\n");

for(r=0; r<100; r++){

//generate matrix

for(i = 0;i < width; i++) {

for(j=0;j<height;j++) {

\*(A+i\*height+j) = gsl\_ran\_gamma(rgen,a,b);;

}

}

// Copy the lists A and B to their respective memory buffers

ret = clEnqueueWriteBuffer(command\_queue,memobjA, CL\_TRUE, 0, width \* height \* sizeof(float), A, 0, NULL, NULL);;

ret = clEnqueueWriteBuffer(command\_queue, rowA, CL\_TRUE, 0, sizeof(int), &row, 0, NULL, NULL);

ret = clEnqueueWriteBuffer(command\_queue, colC, CL\_TRUE, 0, sizeof(int), &col, 0, NULL, NULL);

/\* Set OpenCL Kernel Arguments \*/

ret = clSetKernelArg(kernelVectMult, 0, sizeof(cl\_mem), (void \*)&memobjA);

ret = clSetKernelArg(kernelVectMult, 1, sizeof(cl\_mem), (void \*)&memobjB);

ret = clSetKernelArg(kernelVectMult, 2, sizeof(int), (void \*)&row);

ret = clSetKernelArg(kernelVectMult, 3, sizeof(int), (void \*)&col);

/\* Execute OpenCL Kernel \*/

size\_t globalThreads[2] = {width, height};

size\_t localThreads[2] = {40,40};

clEnqueueNDRangeKernel(command\_queue, kernelVectMult, 2, NULL, globalThreads, localThreads, NULL, 0, NULL);

/\* Copy results from the memory buffer \*/

ret = clEnqueueReadBuffer(command\_queue, memobjB, CL\_TRUE, 0, width \* height \* sizeof(float),B, 0, NULL, NULL);

// Copy the lists A and B to their respective memory buffers

ret = clEnqueueWriteBuffer(command\_queue,memobjA, CL\_TRUE, 0, width \* height \* sizeof(float), A, 0, NULL, NULL);

ret = clEnqueueWriteBuffer(command\_queue, memobjB, CL\_TRUE, 0, width \* height \* sizeof(float), B, 0, NULL, NULL);

ret = clEnqueueWriteBuffer(command\_queue, rowA, CL\_TRUE, 0, sizeof(int), &row, 0, NULL, NULL);

ret = clEnqueueWriteBuffer(command\_queue, colC, CL\_TRUE, 0, sizeof(int), &col, 0, NULL, NULL);

/\* Set OpenCL Kernel Arguments \*/

ret = clSetKernelArg(kernelMatrixMult, 0, sizeof(cl\_mem), (void \*)&memobjA);

ret = clSetKernelArg(kernelMatrixMult, 1, sizeof(cl\_mem), (void \*)&memobjB);

ret = clSetKernelArg(kernelMatrixMult, 2, sizeof(cl\_mem), (void \*)&memobjC);

ret = clSetKernelArg(kernelMatrixMult, 3, sizeof(int), (void \*)&row);

ret = clSetKernelArg(kernelMatrixMult, 4, sizeof(int), (void \*)&col);

/\* Execute OpenCL Kernel \*/

clEnqueueNDRangeKernel(command\_queue, kernelMatrixMult, 2, NULL, globalThreads, localThreads, NULL, 0, NULL);

/\* Copy results from the memory buffer \*/

ret = clEnqueueReadBuffer(command\_queue, memobjC, CL\_TRUE, 0, width \* height \* sizeof(float), C, 0, NULL, NULL);

// Copy the lists A and B to their respective memory buffers

ret = clEnqueueWriteBuffer(command\_queue,memobjC, CL\_TRUE, 0, width \* height \* sizeof(float), C, 0, NULL, NULL);

ret = clEnqueueWriteBuffer(command\_queue, rowA, CL\_TRUE, 0, sizeof(int), &row, 0, NULL, NULL);

/\* Set OpenCL Kernel Arguments \*/

ret = clSetKernelArg(kernelVectSred, 0, sizeof(cl\_mem), (void \*)&memobjC);

ret = clSetKernelArg(kernelVectSred, 1, sizeof(cl\_mem), (void \*)&memobjRes);

ret = clSetKernelArg(kernelVectSred, 2, sizeof(int), (void \*)&row);

/\* Execute OpenCL Kernel \*/

size\_t global\_item\_size = width;

size\_t local\_item\_size = 40;

/\* Execute OpenCL kernel as data parallel \*/

ret = clEnqueueNDRangeKernel(command\_queue, kernelVectSred, 1, NULL, &global\_item\_size, &local\_item\_size, 0, NULL, NULL);

//ret = clEnqueueNDRangeKernel(command\_queue, kernelVectSred, 2, NULL, globalThreads, localThreads, NULL, 0, NULL);

/\* Copy results from the memory buffer \*/

ret = clEnqueueReadBuffer(command\_queue, memobjRes, CL\_TRUE, 0, width \* sizeof(float),Res, 0, NULL, NULL);

shellsort(Res,width);

}

printf("\nStopt\n");

gettimeofday(&T2, NULL);

delta\_ms = 1000\*(T2.tv\_sec - T1.tv\_sec) + (T2.tv\_usec - T1.tv\_usec)/1000;

printf("\nN=%d. Milliseconds passed: %ld\n", N, delta\_ms);

ret = clFlush(command\_queue);

ret = clFinish(command\_queue);

ret = clReleaseKernel(kernelVectMult);

ret = clReleaseKernel(kernelMatrixMult);

ret = clReleaseKernel(kernelVectSred);

ret = clReleaseProgram(program);

ret = clReleaseMemObject(memobjA);

ret = clReleaseMemObject(memobjB);

ret = clReleaseMemObject(memobjC);

ret = clReleaseCommandQueue(command\_queue);

ret = clReleaseContext(context);

free(source\_str);

return 0;

}

void shellsort(float \*a,int n)

{

int j,i,m;

float mid;

for(m = n/2;m>0;m/=2){

for(j = m;j < n;j++){

for(i=j-m;i>=0;i-=m){

if(\*(a + i + m)>=\*(a + i))

break;

else{

mid = \*(a+i);

\*(a + i) = \*(a + i + m);

\*(a + i + m) = mid;

}

}

}

}

}

Код ядра

\_\_kernel

void matrixMultiplication(\_\_global float\* A, \_\_global float\* B, \_\_global float\* C, int width, int height ) {

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

int j = get\_global\_id(1);

float value=0;

for ( int k = 0; k < width; k++) {

value = value + A[mad(j,width,k)] \* B[mad(k,height,i)];

}

C[mad(width,j,i)] = value;

}

\_\_kernel void matrixVectorMultiplication(\_\_global float\* A, \_\_global float\* B, int width, int height ) {

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

int j = get\_global\_id(1);

B[mad(width,j,i)] = A[mad(j ,width,i)]\*5;

}

\_\_kernel void matrixVectorSred(\_\_global float\* C, \_\_global float\* Res, int width) {

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

float sred=0;

float num = 0;

Res[i]=0;

for (int j = 0; j < width; j++ ){

sred = sred + C[mad(j,width,i)];

}

sred = sred/width;

for (int j = 0; j < width; j++ ){

num = C[mad(j, width,i)] - sred;

Res[i] = mad(num , num , Res[i]) ;

}

Res[i]= Res[i] /width;

}