# VIENNA UNIVERSITY OF TECHNOLOGY

# 360.252 Computational Science on Many Core Architectures

Institute for Microelectronics

# Exercise 9

Authors: Camilo Tello Fachin 12127084

 $\label{eq:Supervisor:Dipl.-Ing.} Supervisor:$  Dipl.-Ing. Dr.<br/>techn. Karl Rupp

January 4, 2023





# **Abstract**

Here documented the results of Exercise 9.

# Contents

1	Libraries (4/5 Points)		1
	1.1	Boost.Compute (1/1 Points)	1
	1.2	Thurst (1/1 Points)	1
	1.3	VexCL (1/1 Points)	1
	1.4	$\label{eq:ViennaCL} \textbf{ViennaCL} \ (1/1 \ Points) \ \dots $	1
	1.5	Comparison to CUDA and OpenCL (0/1 Points)	1
2	HIP	(Points 3/3)	2
3	BOI	NUS: Relaxed Christmas Break (0/1 Point)	2
Αp	Appendices		
Α	CPF	CODE for Boost.Compute Library	3
В	CPF	P CODE for Thrust Library	5
C	CPF	P CODE for VexCL Library	6
D	CPF	P CODE for ViennaCL Library	7



# 1 Libraries (4/5 Points)

#### 1.1 Boost.Compute (1/1 Points)

Successfully implemented it and it yields the correct dot product outpout. (See Appendix A)

## 1.2 Thurst (1/1 Points)

Successfully implemented it and it yields the correct dot product outpout. (See Appendix B)

#### 1.3 VexCL (1/1 Points)

Successfully implemented it and it yields the correct dot product outpout. (See Appendix C)

### 1.4 ViennaCL (1/1 Points)

Successfully implemented it and it yields the correct dot product outpout. (See Appendix D)

#### 1.5 Comparison to CUDA and OpenCL (0/1 Points)

Altough It probably wouldve been interesting I just didnt have the time :-(.



# 2 HIP (Points 3/3)

Successfully Implemented the classical CG Algorithm in HIP. Whereby "implementing" I mean replacing CUDA commands to HIP commands. A bit trickier where the Kernel calls, But eventually made it run and it still converges and yields corrects results (Relative residual error smaller than 1e-6).

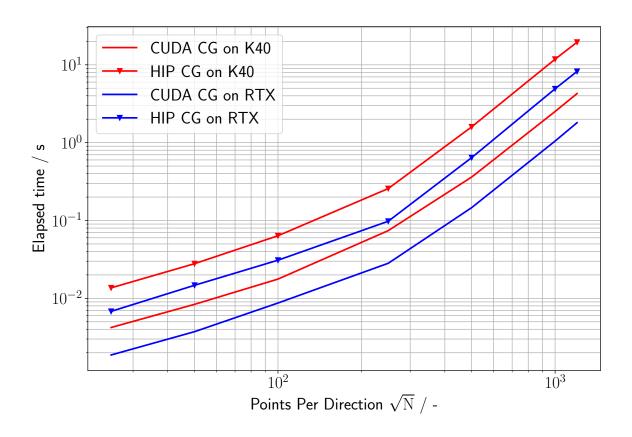


Figure 1: Comparison Conjugate Gradient Algorithm with CUDA and HIP

The HIP Implementation is slower for both GPU's!

# 3 BONUS: Relaxed Christmas Break (0/1 Point)

I had a very relaxed christmas break, therefore You could grant me the Bonus point! Jokes aside, I hope You had a very relaxed Christmas break! :-) Kind regards.



## A CPP CODE for Boost.Compute Library

CPP CODE for Boost.Compute Library

```
namespace compute = boost::compute;
 1
 2
 3
    BOOST_COMPUTE_FUNCTION(float, boost_multiplication, (float x, float y),
 4
 5
                                  return x * y;
 6
                             });
 7
     BOOST_COMPUTE_FUNCTION(float, boost_addition, (float x, float y),
 8
9
10
                                  return x + y;
11
                              });
12
     BOOST_COMPUTE_FUNCTION(float, boost_subtraction, (float x, float y),
13
14
                                  return x - 2 * y;
15
                              });
16
     int main(int argc, char *argv[])
17
18
19
20
         int i, N, N<sub>max</sub>, gentxt;
21
         float dotP;
22
         N_{\text{max}} = 100;
23
         gentxt = 0;
24
         for (i=1; i<argc&&argv[i][0]=='-'; i++) {
         if (argv[i][1]=='N') i++, sscanf(argv[i],"%d",&N); // commandline arg -N for adjusting max. count, if
25
              none given N=100
         if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if
26
               none given, no .txt NOT IMPLEMENTED
27
         }
28
         // get default device and setup context
29
30
         compute::device device = compute::system:: default_device ();
         compute::context context(device);
31
32
         compute::command_queue queue(context, device);
33
         // generate the 3 vectors: \boldsymbol{x}=\!\!\left(1,\!1,\!...,\!1\right) , \boldsymbol{y}=\!\!\left(2,\!2,\!...,\!2\right) and dotp
34
35
         std :: vector < float > host_vector_x(N_max);
36
         std :: vector < float > host_vector_y(N_max);
         std :: vector < float > host_vector_prod(N_max);
37
38
39
         std:: fill (host_vector_x.begin(), host_vector_x.end(), 1.0);
         std:: fill (host_vector_y.begin(), host_vector_y.end(), 2.0);
40
41
42
         // create the 3 vectors on the device
         compute::vector<float> device_vector_x ( host_vector_x . size (), context);
43
44
         compute::vector<float> device_vector_y ( host_vector_y . size (), context);
45
         compute::vector<float> device_vector_prod( host_vector_prod . size(), context);
```



```
46
47
         // transfer data from the host to the device
         compute::copy(host_vector_x .begin(), host_vector_x .end(), device_vector_x .begin(), queue);
48
         compute::copy(host_vector_y .begin(), host_vector_y .end(), device_vector_y .begin(), queue);
49
50
51
         // calculate x+y within x
52
         compute::transform(
53
             device_vector_x .begin(),
54
             device_vector_x .end(),
55
             device_vector_y .begin(),
             device_vector_x .begin(),
56
57
             boost_addition,
58
             queue
         );
59
60
61
         // calculate x-2*y within y
62
         compute::transform(
63
             device_vector_x .begin(),
64
             device_vector_x .end(),
65
             device_vector_y .begin(),
             device_vector_y .begin(),
66
67
             boost\_subtraction,
68
             queue
         );
69
70
         //after these two steps, vector x should have x_i + y_i in every entry, and vector y should have x_i -
              y_i in every entry and one can calculated the product in the thrid vector.
71
72
         //calculate x_i * y_i
73
         compute::transform(
74
             device_vector_x .begin(),
75
             device_vector_x .end(),
76
             device_vector_y .begin(),
             device_vector_prod .begin(),
77
78
             boost_multiplication ,
79
             queue
80
         );
         //copy values back to the host
81
         compute::copy(
82
83
             device_vector_prod .begin(), device_vector_prod .end(), host_vector_prod .begin(), queue
84
         );
85
86
         dotP = std::accumulate(device\_vector\_prod.begin(), device\_vector\_prod.end(), 0);
         std::cout << "<x+y, x-y> = "<< dotP<<" with N = "<< N_-max << std::endl;
87
88
89
         return 0;
90
```



## B CPP CODE for Thrust Library

#### CPP CODE for Thrust Library

```
int main(int argc, char *argv[]) {
   1
   2
                   int i, N, N_max, gentxt;
   3
                    float dotP;
   4
                   N_{\text{max}} = 100;
                    gentxt = 0;
   5
   6
                    for (i=1; i<argc&&argv[i][0]=='-'; i++) {
   7
                     \textbf{if} \ (\mathsf{argv}[i][1] == \mathsf{'N'}) \ i++, \ \mathsf{sscanf}(\mathsf{argv}[i], \mathsf{''} \% \mathsf{d''}, \& \mathsf{N}); \ // \ \mathsf{command line} \ \mathsf{arg} \ -\mathsf{N} \ \mathsf{for} \ \mathsf{adjusting} \ \mathsf{max}. \ \mathsf{count}, \ \mathsf{if} \ \mathsf{none} \ \mathsf{none}
                                  given N=100
                    if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if
   8
                                  none given, no .txt NOT IMPLEMENTED
   9
                   }
10
                    // generate vectors on host of lenght N_max
11
                    thrust :: host_vector < float> h_x(N_max);
12
                    thrust :: host_vector < float > h_y(N_max);
13
                    thrust :: host_vector < float > h_prod1(N_max);
14
15
                    thrust :: host_vector < float > h_prod2(N_max);
16
                    thrust :: fill (h_x.begin(), h_x.end(), 1);
                    thrust :: fill (h_y.begin(), h_y.end(), 2);
17
                    thrust :: fill (h_prod1.begin(), h_prod1.end(), 0);
18
19
                    thrust :: fill (h_prod2.begin(), h_prod2.end(), 0);
20
21
                   // transfer data to the device
22
                    thrust :: device_vector < float> d_x = h_x;
                    thrust :: device_vector < float > d_y = h_y;
23
24
                    thrust :: device_vector < float > d_prod1 = h_prod1;
25
                    thrust :: device_vector < float > d_prod2 = h_prod2;
26
27
                    thrust :: transform(d_x.begin(), d_x.end(), d_y.begin(), d_prod1.begin(), thrust :: plus < float > ());
28
                    thrust :: transform(d_x.begin(), d_x.end(), d_y.begin(), d_y.begin(), thrust :: minus < float > ());
                    thrust :: transform(d\_prod1.begin()\ ,\ d\_prod1.end()\ ,\ d\_prod2.begin()\ ,\ d\_prod1.begin()\ ,\ thrust ::\ multiplies < \textbf{float}
29
                                  >());
30
31
32
                    // transfer data back to host
                    thrust :: copy(d_prod1.begin(), d_prod1.end(), h_prod1.begin());
33
34
                   dotP = std::accumulate(d_prod1.begin(), d_prod1.end(), 0);
35
                   std :: cout << "<x+y, x-y> = " << dotP << " with N = " << N_-max << std::endl;
36
37
38
                   return 0;
39
            }
```



## C CPP CODE for VexCL Library

#### CPP CODE for VexCL Library

```
1
                    int main(int argc, char *argv[]) {
    2
                            int i, N, N<sub>-</sub>max, gentxt;
    3
                             double dotP;
    4
                            N_{\text{max}} = 100;
   5
                             gentxt = 0;
   6
                             for (i=1; i<argc&argv[i][0]=='-'; i++) {
    7
                              given N=100
                               \textbf{if } (\mathsf{argv}[i][1] == \mathsf{'g'}) \ i++, \, \mathsf{sscanf}(\mathsf{argv}[i], \,\, \mathsf{''}\, \%\mathsf{d''}, \,\, \&\mathsf{gentxt}); \,\, // \,\, \mathsf{command line } \, \mathsf{arg.} \,\, -\mathsf{g} \,\, \mathsf{for } \,\, \mathsf{generating} \,\, \mathsf{a} \,\,\, \mathsf{txt} \,, \,\, \mathsf{if} \,\, \mathsf{full} \,\, 
   8
                                                  none given, no .txt NOT IMPLEMENTED
   9
                             }
10
                            vex::Context ctx(vex:: Filter :: GPU && vex::Filter::DoublePrecision);
11
12
                             std::cout << ctx << std::endl; \ // \ print \ \ list \ \ of \ \ selected \ \ devices
13
14
15
                             std :: vector < double> x(N_max, 1.0), y(N_max, 2.0), prod(N_max, 0);
16
17
                            vex:: vector < double> X(ctx, x);
18
                            vex :: vector < double > Y(ctx, y);
19
                             vex :: vector < double> F1 = X + Y;
20
21
                            vex:: vector < double> F2 = X - Y;
22
                            vex:: vector < double> P = F1 * F2;
23
                            dotP = std::accumulate(P.begin(), P.end(), 0);
24
                             std :: cout << "<x+y, x-y> = " << dotP << " with N = " << N_{-}max << std::endl;
25
26
27
                            return 0;
28
```



## D CPP CODE for ViennaCL Library

#### CPP CODE for ViennaCL Library

```
int main(int argc, char *argv[]) {
    1
     2
                              int i, N, N<sub>-</sub>max, gentxt;
     3
                               double dotP;
     4
                              N_{\text{max}} = 100;
    5
                               gentxt = 0;
    6
                               for (i=1; i<argc&&argv[i][0]=='-'; i++) {
     7
                               given N=100
                                \textbf{if } (\mathsf{argv}[i][1] == \mathsf{'g'}) \ i++, \, \mathsf{sscanf}(\mathsf{argv}[i], \,\, \mathsf{''}\, \%\mathsf{d''}, \,\, \&\mathsf{gentxt}); \,\, // \,\, \mathsf{command line } \, \mathsf{arg.} \,\, -\mathsf{g} \,\, \mathsf{for } \,\, \mathsf{generating} \,\, \mathsf{a} \,\,\, \mathsf{txt} \,, \,\, \mathsf{if} \,\, \mathsf{full} \,\, 
    8
                                                    none given, no .txt NOT IMPLEMENTED
    9
                               }
10
                               viennacl:: vector < double > x = viennacl:: scalar_vector < double > (N_max, 1.0);
11
12
                               viennacl :: vector < double > y = viennacl:: scalar_vector < double > (N_max, 2.0);
13
                               viennacl :: vector < double> f1 = x + y;
14
15
                               viennacl :: vector < double> f2 = x - y;
                               viennacl :: vector < double > p = viennacl::linalg :: element_prod(f1, f2);
16
17
18
                               dotP = std::accumulate(p.begin(), p.end(), 0);
                               std :: cout << "<x+y, x-y> = " << dotP << " with N = " << N_-max << std::endl;
19
20
21
                               return EXIT_SUCCESS;
22
                    }
```