Vienna University of Technology

360.252 Computational Science on Many Core Architectures

Institute for Microelectronics

Exercise 9

Authors:
Camilo Tello Fachin
12127084

Supervisor:
Dipl.-Ing. Dr.techn. Karl Rupp

January 4, 2023

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Abstract

Here documented the results of Exercise 9.

Contents

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 ”implmenting” 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).

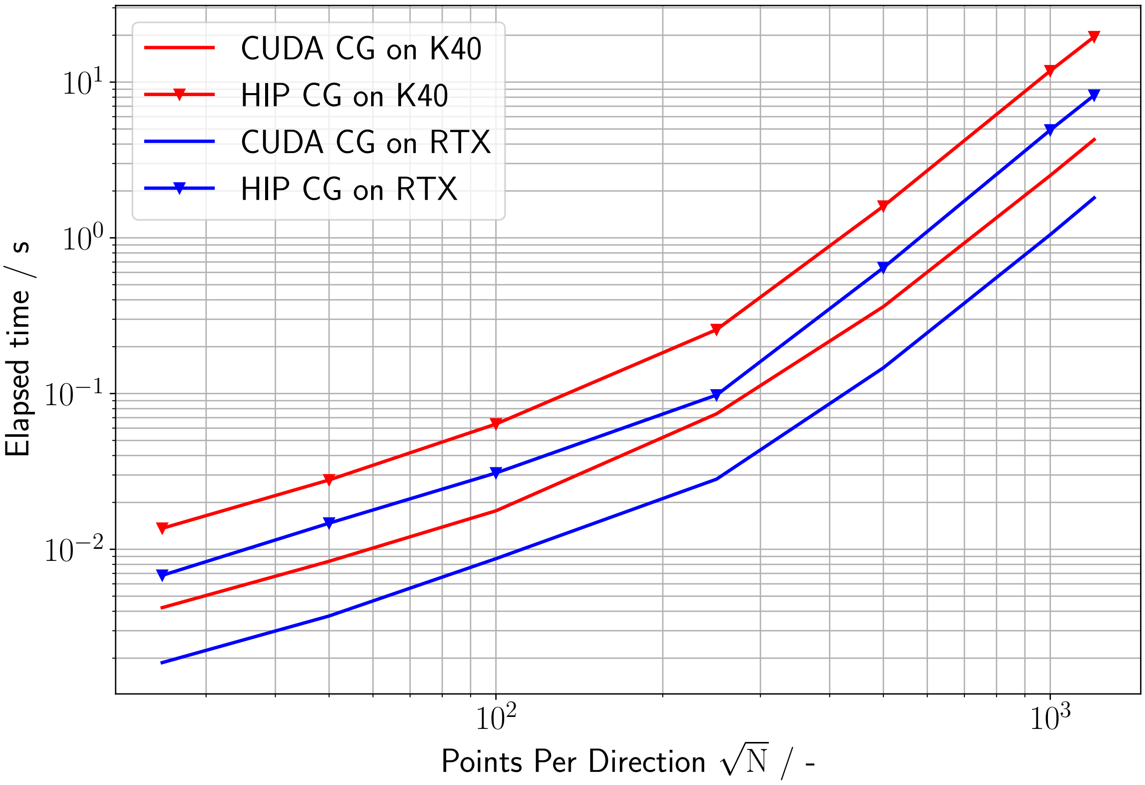

Figure 1: Comparison Conjugate Gradient Algorithm with `CUDA` and `HIP`

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

```cpp
namespace compute = boost::compute;

BOOST_COMPUTE_FUNCTION(float, boost_multiplication, (float x, float y),
                        {
                            return x * y;
                        });

BOOST_COMPUTE_FUNCTION(float, boost_addition, (float x, float y),
                        {
                            return x + y;
                        });

BOOST_COMPUTE_FUNCTION(float, boost_subtraction, (float x, float y),
                        {
                            return x - 2 * y;
                        });
int main(int argc, char *argv[])
{

    int i, N, N_max, gentxt;
    float dotP;
    N_max = 100;
    gentxt = 0;
    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 none given N=100
    if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if none given, no .txt NOT IMPLEMENTED
    }

    // get default device and setup context
    compute::device device = compute::system::default_device();
    compute::context context(device);
    compute::command_queue queue(context, device);

    // generate the 3 vectors: x =(1,1,...,1) , y =(2,2,...,2) and dotp
    std::vector<float> host_vector_x(N_max);
    std::vector<float> host_vector_y(N_max);
    std::vector<float> host_vector_prod(N_max);

    std::fill(host_vector_x.begin(), host_vector_x.end(), 1.0);
    std::fill(host_vector_y.begin(), host_vector_y.end(), 2.0);

    // create the 3 vectors on the device
    compute::vector<float> device_vector_x(host_vector_x.size(), context);
    compute::vector<float> device_vector_y(host_vector_y.size(), context);
    compute::vector<float> device_vector_prod(host_vector_prod.size(), context);

    // transfer data from the host to the device
    compute::copy(host_vector_x.begin(), host_vector_x.end(), device_vector_x.begin(), queue);
    compute::copy(host_vector_y.begin(), host_vector_y.end(), device_vector_y.begin(), queue);

    //calculate x+y within x
    compute::transform(
        device_vector_x.begin(),
        device_vector_x.end(),
        device_vector_y.begin(),
        device_vector_x.begin(),
        boost_addition,
        queue
    );

    //calculate x-2*y within y
    compute::transform(
        device_vector_x.begin(),
        device_vector_x.end(),
        device_vector_y.begin(),
        device_vector_y.begin(),
        boost_subtraction,
        queue
    );
    //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.

    //calculate x_i * y_i
    compute::transform(
        device_vector_x.begin(),
        device_vector_x.end(),
        device_vector_y.begin(),
        device_vector_prod.begin(),
        boost_multiplication,
        queue
    );
    //copy values back to the host
    compute::copy(
        device_vector_prod.begin(), device_vector_prod.end(), host_vector_prod.begin(), queue
    );

    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;

    return 0;
}
```

B CPP CODE for Thrust Library

CPP CODE for Thrust Library

```cpp
int main(int argc, char *argv[]) {
  int i, N, N_max, gentxt;
  float dotP;
  N_max = 100;
  gentxt = 0;
  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 none given N=100
  if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if none given, no .txt NOT IMPLEMENTED
  }

  // generate vectors on host of lenght N_max
  thrust::host_vector<float> h_x(N_max);
  thrust::host_vector<float> h_y(N_max);
  thrust::host_vector<float> h_prod1(N_max);
  thrust::host_vector<float> h_prod2(N_max);
  thrust::fill(h_x.begin(), h_x.end(), 1);
  thrust::fill(h_y.begin(), h_y.end(), 2);
  thrust::fill(h_prod1.begin(), h_prod1.end(), 0);
  thrust::fill(h_prod2.begin(), h_prod2.end(), 0);

  // transfer data to the device
  thrust::device_vector<float> d_x = h_x;
  thrust::device_vector<float> d_y = h_y;
  thrust::device_vector<float> d_prod1 = h_prod1;
  thrust::device_vector<float> d_prod2 = h_prod2;

  thrust::transform(d_x.begin(), d_x.end(), d_y.begin(), d_prod1.begin(), thrust::plus<float>());
  thrust::transform(d_x.begin(), d_x.end(), d_y.begin(), d_prod2.begin(), thrust::minus<float>());
  thrust::transform(d_prod1.begin(), d_prod1.end(), d_prod2.begin(),d_prod1.begin(), thrust::multiplies<float>());


  // transfer data back to host
  thrust::copy(d_prod1.begin(), d_prod1.end(), h_prod1.begin());
  dotP = std::accumulate(d_prod1.begin(), d_prod1.end(), 0);

  std::cout << "<x+y, x-y> = " << dotP << " with N = " << N_max << std::endl;

  return 0;
}
```

C CPP CODE for VexCL Library

CPP CODE for VexCL Library

```cpp
int main(int argc, char *argv[]) {
  int i, N, N_max, gentxt;
  double dotP;
  N_max = 100;
  gentxt = 0;
  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 none given N=100
   if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if none given, no .txt NOT IMPLEMENTED
  }

  vex::Context ctx(vex::Filter::GPU && vex::Filter::DoublePrecision);

  std::cout << ctx << std::endl; // print list of selected devices

  std::vector<double> x(N_max, 1.0), y(N_max, 2.0), prod(N_max, 0);

  vex::vector<double> X(ctx, x);
  vex::vector<double> Y(ctx, y);

  vex::vector<double> F1 = X + Y;
  vex::vector<double> F2 = X - Y;
  vex::vector<double> P = F1 * F2;

  dotP = std::accumulate(P.begin(), P.end(), 0);
  std::cout << "<x+y, x-y> = " << dotP << " with N = " << N_max << std::endl;

  return 0;
}
```

D CPP CODE for ViennaCL Library

CPP CODE for ViennaCL Library

```
int main(int argc, char *argv[]) {
  int i, N, N_max, gentxt;
  double dotP;
  N_max = 100;
  gentxt = 0;
  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 none given N=100
   if (argv[i][1]=='g') i++, sscanf(argv[i], "%d", &gentxt); // commandline arg. -g for generating a txt, if none given, no .txt NOT IMPLEMENTED
  }

   viennacl::vector<double> x = viennacl::scalar_vector<double>(N_max, 1.0);
   viennacl::vector<double> y = viennacl::scalar_vector<double>(N_max, 2.0);

   viennacl::vector<double> f1 = x + y;
   viennacl::vector<double> f2 = x - y;
   viennacl::vector<double> p = viennacl::linalg::element_prod(f1,f2);

  dotP = std::accumulate(p.begin(), p.end(), 0);
  std::cout << "<x+y, x-y> = " << dotP << " with N = " << N_max << std::endl;

  return EXIT_SUCCESS;
}
```