Computational Science on Many-Core Architectures Exercise 8

Example 1 Libraries (5 Points)

1) BoostCompute

Part of the important calculation from the main() part.

Listing 1: BoostCompute

```
1     std::vector<ScalarType> x(N,1);
2     std::vector<ScalarType> y(N,2);
3     std::vector<ScalarType> init(N,0);
4
5     compute::vector<ScalarType> d_x(x.size(), context);
6     compute::vector<ScalarType> d_y(y.size(), context);
7     compute::vector<ScalarType> d_xply(y.size(), context);
8     compute::vector<ScalarType> d_xmiy(y.size(), context);
9
10    // transfer data from the host to the device
11    compute::copy(
12    x.begin(), x.end(), d_x.begin(), queue
13    );
14    compute::copy(
15    y.begin(), y.end(), d_y.begin(), queue
16    );
17    compute::copy(
18    init.begin(), init.end(), d_xply.begin(), queue
19    );
20    compute::copy(
21    init.begin(), init.end(), d_xmiy.begin(), queue
22    );
23    compute::transform(d_x.begin(), d_x.end(),
24    d_y.begin(), d_xply.begin(), compute::plus<ScalarType>{}, queue
25    );
26    compute::transform(d_x.begin(), d_x.end(),
27    d_y.begin(), d_xmiy.begin(), compute::minus<ScalarType>{}, queue
28    );
29    timer.reset();
30    for (int i = 0; i < anz; i++)
31    {
32        dot = compute::inner_product(d_xply.begin(), d_xply.end(),
33        d_xmiy.begin(), 0.0, queue);
34    }
35    Boost_Time = timer.get()/anz;
36    std::cout << "dot-prod_Boost: " << dot << std::endl;
```

2) Thrust

Listing 2: Thrust

```
1  std::vector<ScalarType> x(N,1);
2  std::vector<ScalarType> y(N,2);
3  std::vector<ScalarType> v1pv2(N,0);
4  std::vector<ScalarType> v1mv2(N,0);
5
6  for (int i = 0; i < N; i++)
7  {
8      v1pv2[i] = x[i] + y[i];
9      v1mv2[i] = x[i] - y[i];
10 }
11 timer.reset();
12 for (int i = 0; i < anz; i++)
13 {
14     thrust::host_vector<ScalarType> h_v1 = v1pv2;
15     thrust::host_vector<ScalarType> h_v2 = v1mv2;
16
17     thrust::device_vector<ScalarType> d_v1 = h_v1;
18     thrust::device_vector<ScalarType> d_v2 = h_v2;
19
20     dot = thrust::inner_product(d_v1.begin(), d_v1.end(),
21      d_v2.begin(), start);
22 }
23 VexCL_Time = timer.get()/anz;
24 std::cout << "dot-prod_Thrust: " << dot << std::endl;
```

3) VexCL

Listing 3: VexCL

```
1 timer.reset();
2 vex::Reductor<double, vex::SUM> DOT(ctx);
3 for (int i = 0; i < anz; i++)
4 {
5     dot = DOT((X+Y)*(X-Y));
6 }
7 VexCL_Time = timer.get()/anz;
8 std::cout << "dot-prod_VexCL: " << dot << std::endl;
```

4) ViennaCL

Listing 4: ViennaCL

```
viennacl::vector<double> x_VIE = viennacl::scalar_vector<double>(N, 1.0);
viennacl::vector<double> y_VIE = viennacl::scalar_vector<double>(N, 2.0);
timer.reset();
for (int i = 0; i < anz; i++)
{
    dot = viennacl::linalg::inner_prod(x_VIE + y_VIE,x_VIE - y_VIE);
}
ViennaCl_Time = timer.get()/anz;
std::cout << "dot-prod_ViennaCL: " << dot << std::endl;
```

5) CPU

Listing 5: CPU kernel

```
std::vector<ScalarType> vecplus(std::vector<ScalarType> x,
std::vector<ScalarType> y, int sign)
{
    for (int i = 0; i < x.size(); i++)
    {
        x[i] = x[i] + sign * y[i];
    }
    return x;
}

ScalarType vec_dot(std::vector<ScalarType> x, std::vector<ScalarType> y)
{
    double sum = 0;
    for (int i = 0; i < x.size(); i++)
    {
        sum += x[i] * y[i];
    }
    return sum;
}
```

Listing 6: CPU benchmark

```
std::vector<ScalarType> x(N,1);
std::vector<ScalarType> y(N,2);
timer.reset();
for (int i = 0; i < anz; i++)
{
    reference = vec_dot(vecplus(x,y,1),vecplus(x,y,-1));
}
CPU_Time = timer.get()/anz;
std::cout << "reference result from CPU: " << reference << std::endl;
```

6) myopenCL

Listing 7: myopenCL kernel

```
1     const char *my_opencl_program = ""
2 "__kernel void vec_mult(__global double *x,\n"
3 "                       __global double *y,\n"
4 "                       __global double *result,\n"
5 "                       unsigned int N\n)"
6 "{\n"
7 "   int d = 0;\n"
8 "  for (unsigned int i  = get_global_id(0);\n"
9 "                    i  < N;\n"
10 "                    i += get_global_size(0))\n"
11 "{\n"
12 "    result[i] = (x[i]+y[i]) * (x[i]-y[i]);\n"
13 "}\n"
14 "}";
```

It calculates the product of $(x + y) \times (x - y)$ and returns an array with the size of N, where N is the vector size of each vector x and y. The summation of the final dot product is then implemented in main. The time measurement begins in the main where the kernel starts and ens after the final summation with a for loop.

7) myCUDA

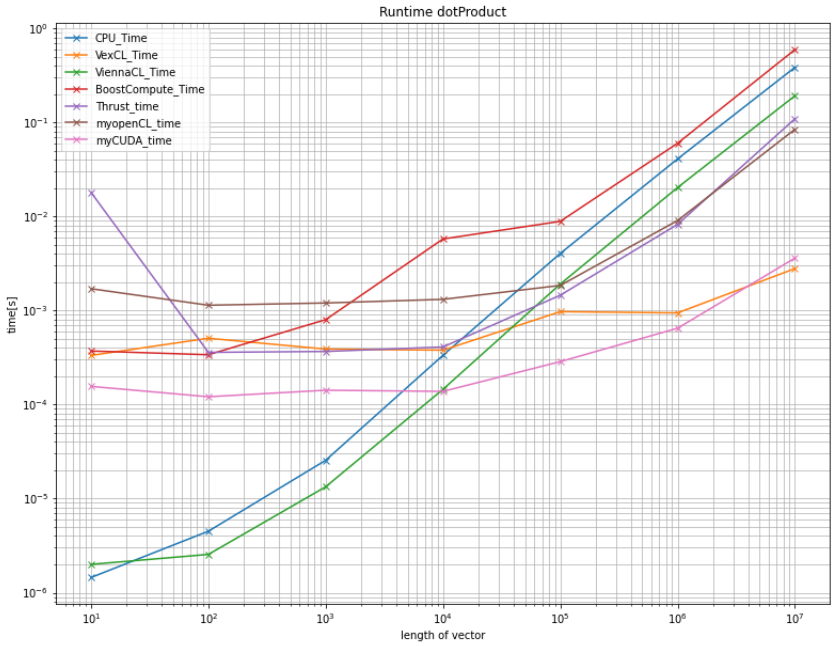
Listing 8: myCUDA benchmark

```
1  timer.reset();
2  for(int i=0; i < anz; ++i)
3  {
4      cudaMalloc(&cuda_dot, sizeof(ScalarType));
5      cudaMemcpy(cuda_dot, dot, sizeof(ScalarType), cudaMemcpyHostToDevice);
6
7      dot_product<<<256, 256>>>(cuda_x, cuda_y, cuda_dot, N);
8      cudaMemcpy(dot, cuda_dot, sizeof(ScalarType), cudaMemcpyDeviceToHost);
9      cudaDeviceSynchronize();
10 }
11 myCUDA_Time = timer.get()/anz;
12 std::cout << "Dot Product = " << *dot << std::endl;
```

The initialization in line 4 and 5 is important because then the dot product would be summed up over all iteration of *anz*.

8) Benchmark

For the benchmark I always measure the time for the implementation 20 times and take the mean value of it.



I also included an implementation on the single CPU with two different kernels to have a reference more.
The winner of the benchmark is the VecCL library also interesting is that the time for that library VexCL does not grow as much as the other times.
The BoostCompute library has an out layer for the first point at a vector size of 10 but then the times behave like the VexCL library until a vector size of 10^5. I expected that the CPU implementation is the slowest but the BoostCompute library is the slowest one.
MyCUDA implementation performs for the first 6 vector sizes better than all the others but for the last one VexCL wins.
MyopenCl has the same shape as the myCUDA implementation with an offset $\rightarrow$ summation for the final dot-product result for the MyopenCl implementation done in the CPU part.