

Computational Science on Many-Core Architectures Exercise 7

Example 1 Dot Product with OpenCL (4 Points)

a)

Listing 1: opencl 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  "    for (unsigned int i = get_global_id(0);\n"
8  "        i < N;\n"
9  "        i += get_global_size(0))\n"
10 "        result[i] = x[i] * y[i];\n"
11 "}"
```

This opencl kernel only multiplies the single entries and stored the results in the result vector.

Listing 2: Benchmark for the opencl

```

1  for (int i = 0; i < anz; i++)
2  {
3      //
4      // Set kernel arguments:
5      //
6      timer.reset();
7      err = clSetKernelArg(my_kernel, 0, sizeof(cl_mem),
8      (void*)&ocl_x); OPENCLERR_CHECK(err);
9
10     err = clSetKernelArg(my_kernel, 1, sizeof(cl_mem),
11     (void*)&ocl_y); OPENCLERR_CHECK(err);
12
13     err = clSetKernelArg(my_kernel, 2, sizeof(cl_mem),
14     (void*)&ocl_result); OPENCLERR_CHECK(err);
15
16     err = clSetKernelArg(my_kernel, 3, sizeof(cl_uint),
17     (void*)&vector_size); OPENCLERR_CHECK(err);
18
19     //
20     // Enqueue kernel in command queue:
21     //
22     err = clEnqueueNDRangeKernel(my_queue, my_kernel, 1,
23     NULL, &global_size, &local_size, 0, NULL, NULL); OPENCLERR_CHECK(err);
24
25     // wait for all operations in queue to finish:
26     err = clFinish(my_queue); OPENCLERR_CHECK(err);
```

```

27 }
28 opencl_time = timer.get();
29 std::cout << "Time for opencl kernel: " << opencl_time/anz << std::endl;

```

The for loop is only for the benchmark and $anz = 100$. I only add the line 13 and 14 to set the buffer for the result vector.

Listing 3: CUDA kernel

```

1 __global__ void GPU_dot (double *x, double *y, double *dot, unsigned int N)
2 {
3     unsigned int ind = threadIdx.x + blockDim.x*blockIdx.x;
4     unsigned int str = blockDim.x*gridDim.x;
5
6     double tmpsum = 0.0;
7     while(ind < N)
8     {
9         tmpsum = x[ind]*y[ind];
10        ind += str;
11    }
12    dot[threadIdx.x] = tmpsum;
13 }

```

Listing 4: CUDA benchmark

```

1 timer.reset();
2 for (int i = 0; i < anz; i++)
3 {
4     GPU_dot<<<128, 128>>>(cuda_X, cuda_Y, cuda_dot_GPU, N);
5     cudaDeviceSynchronize();
6     cudaMemcpy(dot_GPU, cuda_dot_GPU, sizeof(double), cudaMemcpyDeviceToHost);
7 }
8 //cudaMemcpy(dot_GPU, cuda_dot_GPU, sizeof(double), cudaMemcpyDeviceToHost);
9 GPU_time = timer.get();
10 std::cout << "Time for GPU kernel: " << GPU_time/anz << std::endl;

```

Listing 5: CPU kernel

```

1 void CPU_dot(ScalarType *x, ScalarType *y, ScalarType *result ,
2             unsigned int N)
3 {
4     for (int i = 0; i < N; i++)
5     {
6         result[i] = x[i] * y[i];
7     }
8 }

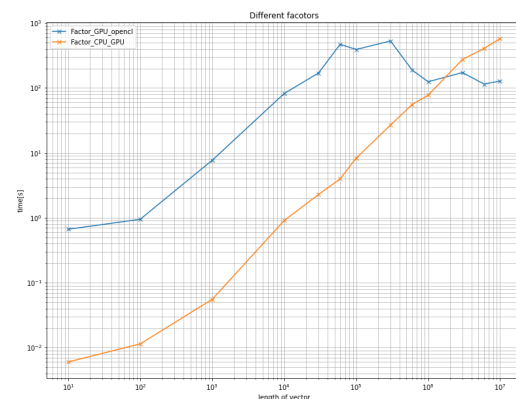
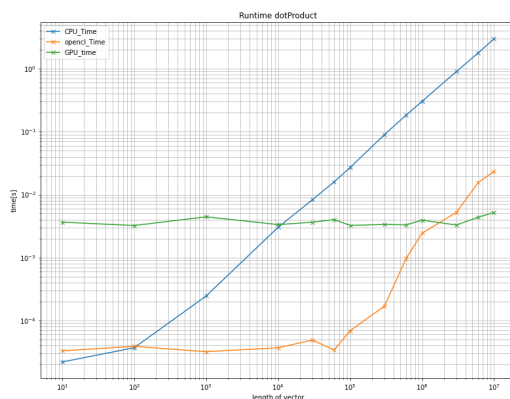
```

Listing 6: CPU benchmark

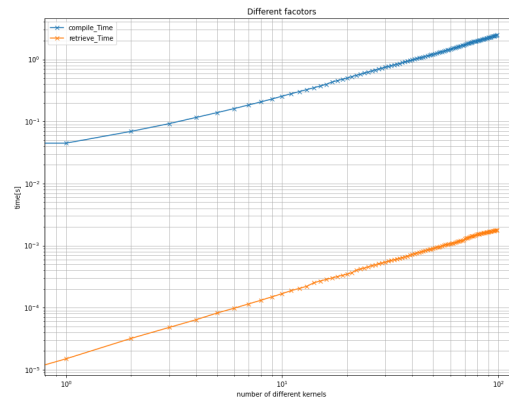
```

1 timer.reset();
2 for (int i = 0; i < anz; i++)
3 {
4     CPU_dot(X, Y, dot_CPU, N);
5 }
6 CPU_time = timer.get();
7 std::cout << "Time for CPU kernel: " << CPU_time/anz << std::endl;

```



On the left we could see that the different runtimes for the same number of entries of a vector and apply the dot-product on it. For the GPU-times it is not clear why the times are like this. I also switch the mode of the machine to the GTX 1080. On the right side I plotted the different factros which one is faster opencl or CPU and GPU or CPU. At a certain point opencl is over 200 times faster than the CPU one.



The time to create the kernels is linear with the numbers of M different kernels and also the time to retrieve it. The second case is much faster than the first.

For 1.4 I tried first to write a routine which adds a line with respect to a given number of M to create every time a different kernel and to measure the time. I tried this with a string implementation. That does not work properly because the declaration at the begin of the provided code is a *const char* and I got errors that I could not change a *const char* for that. Then I tried to implement it with *char* datatypes and that also does not work. So I wrote 100 different kernels with if conditions and add a line "d = 0;" which does not affect the implementation from the dot product and so I measure the time. My code is 4500 lines long so don't be afraid I could not find a better solution.