360.252 - Computational Science on Many-Core Architectures

WS 2020 - Exercise 9

Christian Gollmann, 01435044

Last update: December 27, 2020

Contents

1 OpenMP

The runtimes for Cuda an OpenCL I took from last exercise. For small N, OpenMP seems to perform pretty well, even better than the CUDA implementation. Due to the ease of implementation, the OpenMP code could be quite efficiently implemented from my side and use little overhead. However, as the problem becomes more compute intense, Cuda shows better performance since the execution times might not be governed by the implementations' overhead anymore.

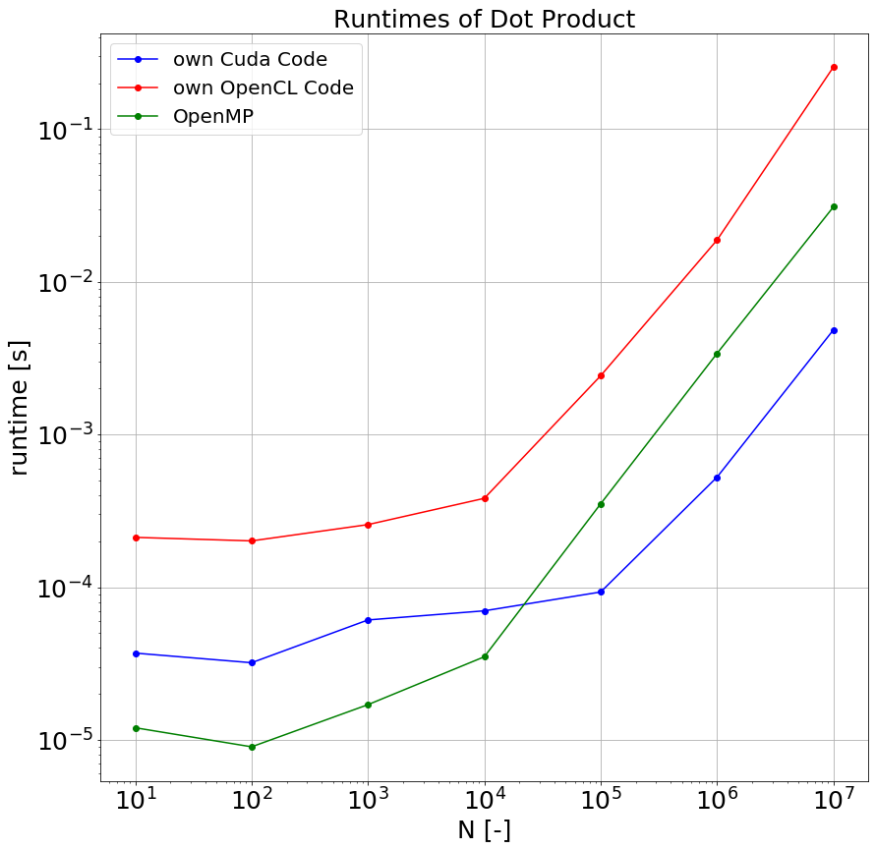

Figure 1: runtimes of dot product

Listing 1: Using OpenMP for GPU

```
1  int main()
2  {
3  Timer timer;
4
5      int N = 100;
6      double *x = (double*)malloc(sizeof(double) * N);
7      double *y = (double*)malloc(sizeof(double) * N);
8
9      for (size_t i=0; i<N; ++i) {
10         x[i] = 1;
11         y[i] = 2;
12     }
13
14     double dot;
15
16 std::vector<double> timings;
```

```
17 for(int reps=0; reps < 10; ++reps) {
18 timer.reset();
19     dot = 0;
20
21     #pragma omp target teams distribute parallel for map(to: x[0:N], y[0:N])
22     map(tofrom: dot) reduction(+:dot)
23     for (int idx = 0; idx < N; ++idx)
24     {
25         dot += (x[idx] + y[idx]) * (x[idx] - y[idx]);
26     }
27
28 timings.push_back(timer.get());
29 }
30
31 std::sort(timings.begin(), timings.end());
32 double time_elapsed = timings[10/2];
33
34 std::cout << "Time elapsed: " << time_elapsed << std::endl << std::endl;
35
36     std::cout << "Reduction result: " << dot << std::endl;
37     std::cout << "Expected result: " << (-3)*N << std::endl;
38
39     return EXIT_SUCCESS;
40 }
```