DD2356 Assignment 2
Programming with OpenMP

Yannis Paetzelt - paetzelt@kth.se

April 24, 2021

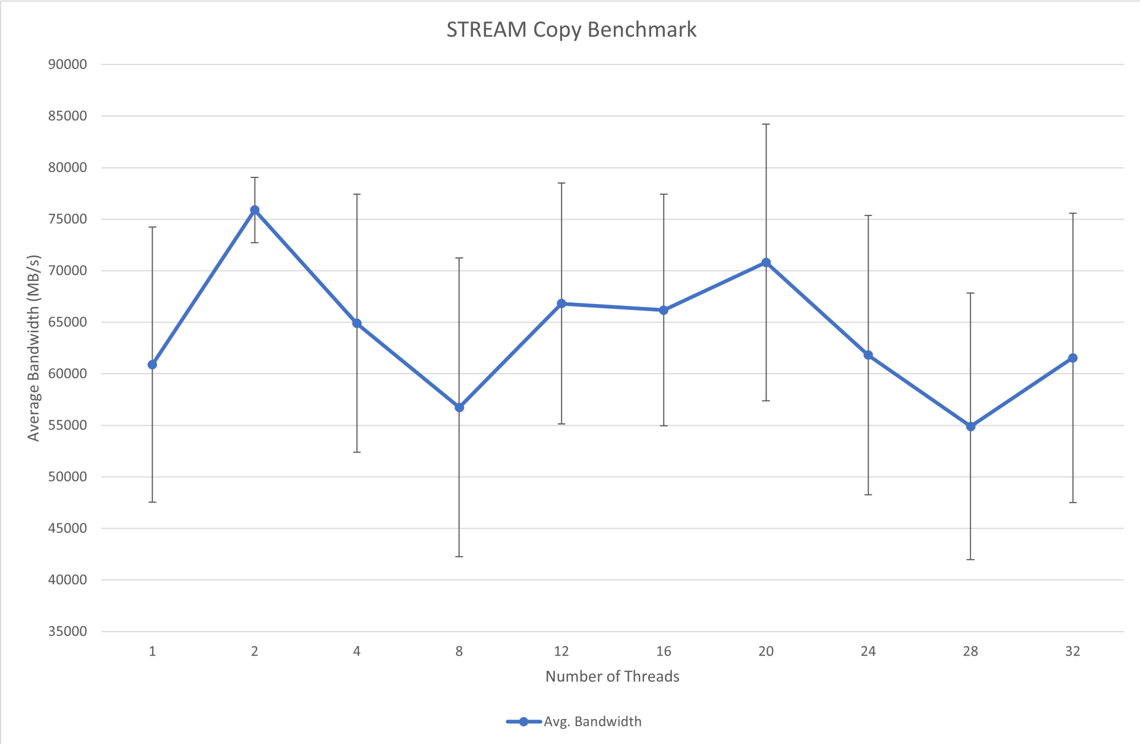
Code for this assignment can be found here.
(`https://github.com/K2017/DD2356/tree/master/Assignment-II`)

Exercise 1

1. Code for this exercise is available on the repository above, in the file called `hello.c`.
2. The flag(s) used to compile the program with OpenMP are `-fopenmp`, using GCC.
3. The code was run as usual on Beskow using `srun -n1 ./hello`.
4. There are two ways to set the default number of threads in OpenMP, the first is by using `omp_set_num_threads(n)` before a parallel region (works at runtime), and the other is to set the `OMP_NUM_THREADS` environment variable before executing the program.

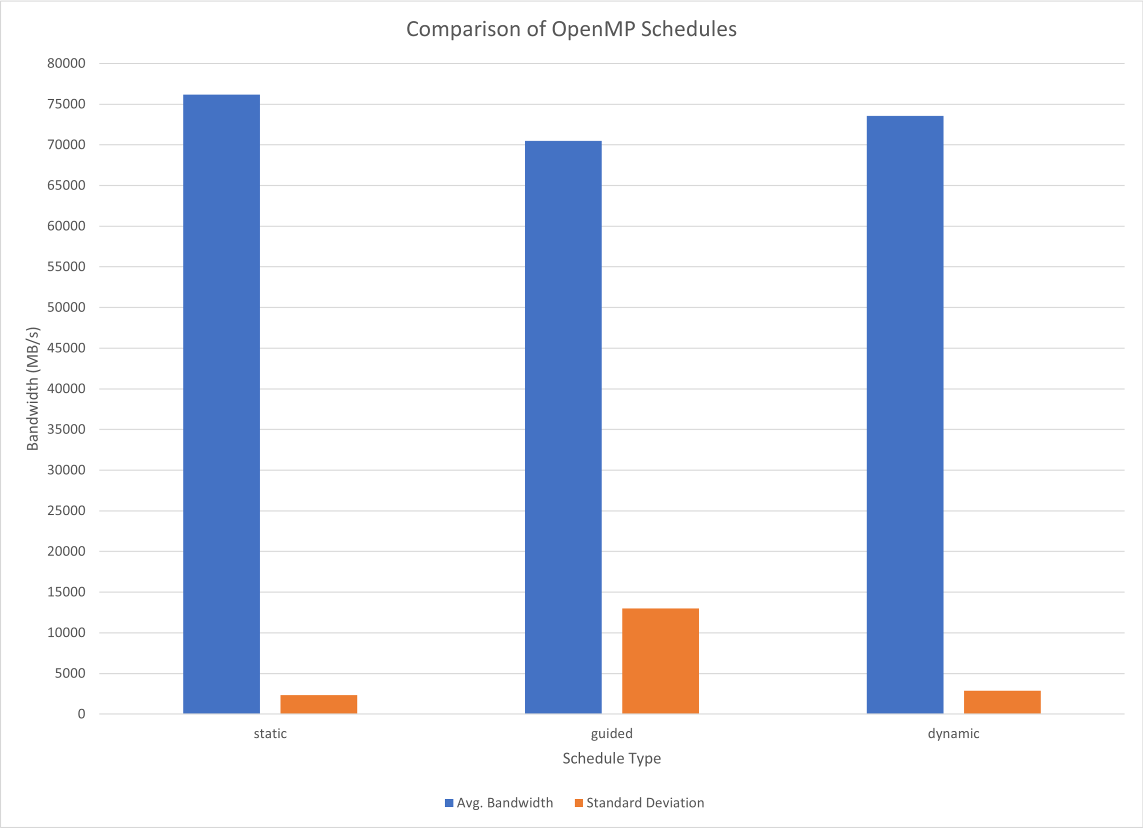
Exercise 2

1.



2. There does not seem to be any significant dependency on the number of threads when it comes to the average bandwidth as measured. As can be seen from the error bars, there is high variance in the bandwidth for individual runs, likely due to the dependence on IO device speed.

3.



4. For performance testing, the schedule was set using an environment variable, specifically `OMP_SCHEDULE=[static|guided|dynamic]`. The equivalent in code would be using the `schedule` clause in an OpenMP `pragma` expression.

Exercise 3

1. The `serial_sum` code was run 20 times with an array size of 100M. Results were printed to the console.

```
sum: 50001084.285723
Total time: 2.05274
Average time: 0.102637
Standard Deviation: 0.00102308
```

2. The execution time with `omp_sum` is much lower, but the resulting sum is incorrect. The threads are not adding to the `sum_val` atomically, causing some writes to be ignored leading to a lower sum.

```
sum: 1562295.812088
Total time: 0.255956
```

```
Average time: 0.0127978
Standard Deviation: 0.000662959
```

3. By protecting the requisite code region, we effectively force synchronous operation, with all of the overhead associated with threading. This leads to a massively increased runtime cost. The sum is correct however.

```
sum: 50001084.285743
Total time: 389.187
Average time: 19.4594
Standard Deviation: 0.533164
```

Running the program while varying the number of threads reveals a noticeable slowdown as we increase the number of threads. Note the array size was changed to 1M for this test.

```
Number of threads: 1
Total time: 0.37217
Average time: 0.0186085
Standard Deviation: 1.92847e-05
Number of threads: 2
Total time: 0.75966
Average time: 0.037983
Standard Deviation: 0.00134427
Number of threads: 4
Total time: 1.0295
Average time: 0.051475
Standard Deviation: 0.00206673
Number of threads: 8
Total time: 1.92024
Average time: 0.0960119
Standard Deviation: 0.0162802
Number of threads: 16
Total time: 2.96142
Average time: 0.148071
Standard Deviation: 0.0449779
Number of threads: 20
Total time: 3.03712
Average time: 0.151856
Standard Deviation: 0.0073062
Number of threads: 24
```

```
Total time: 2.55274
Average time: 0.127637
Standard Deviation: 0.0101507
Number of threads: 28
Total time: 3.26242
Average time: 0.163121
Standard Deviation: 0.00668847
Number of threads: 32
Total time: 3.8314
Average time: 0.19157
Standard Deviation: 0.0101418
```

Performing the same test with the program from question 2 shows a general speedup as the number of threads increases.

```
Number of threads: 1
Total time: 0.0171745
Average time: 0.000858727
Standard Deviation: 4.32215e-05
Number of threads: 2
Total time: 0.00887882
Average time: 0.000443941
Standard Deviation: 6.22288e-05
Number of threads: 4
Total time: 0.00482135
Average time: 0.000241068
Standard Deviation: 3.22301e-05
Number of threads: 8
Total time: 0.0028244
Average time: 0.00014122
Standard Deviation: 3.40775e-05
Number of threads: 16
Total time: 0.00163729
Average time: 8.18644e-05
Standard Deviation: 4.19298e-05
Number of threads: 20
Total time: 0.00125275
Average time: 6.26374e-05
Standard Deviation: 2.2673e-05
Number of threads: 24
```

```
Total time: 0.00188811
Average time: 9.44055e-05
Standard Deviation: 0.000196844
Number of threads: 28
Total time: 0.00098218
Average time: 4.9109e-05
Standard Deviation: 2.32447e-05
Number of threads: 32
Total time: 0.0423996
Average time: 0.00211998
Standard Deviation: 0.00188412
```

4. By avoiding putting everything in a critical section and instead using local computation, we can achieve significant speedup, while obtaining the correct result.

```
Number of threads: 1
Total time: 0.017125
Average time: 0.000856248
Standard Deviation: 3.27304e-05
Number of threads: 2
Total time: 0.00881266
Average time: 0.000440633
Standard Deviation: 4.91182e-05
Number of threads: 4
Total time: 0.00476323
Average time: 0.000238162
Standard Deviation: 2.74124e-05
Number of threads: 8
Total time: 0.00286901
Average time: 0.00014345
Standard Deviation: 3.89686e-05
Number of threads: 16
Total time: 0.00170757
Average time: 8.53783e-05
Standard Deviation: 4.87363e-05
Number of threads: 20
Total time: 0.0026004
Average time: 0.00013002
Standard Deviation: 0.000308691
Number of threads: 24
```

```
Total time: 0.0011481
Average time: 5.74049e-05
Standard Deviation: 2.5131e-05
Number of threads: 28
Total time: 0.0010656
Average time: 5.32802e-05
Standard Deviation: 2.40711e-05
Number of threads: 32
Total time: 0.00279677
Average time: 0.000139838
Standard Deviation: 0.000420056
```

The speedup is similar to the program in question 2.

5. Using a padded struct to remove false sharing results in similar performance.

```
Number of threads: 1
Total time: 0.0185797
Average time: 0.000928984
Standard Deviation: 2.6108e-05
Number of threads: 2
Total time: 0.00949813
Average time: 0.000474907
Standard Deviation: 4.2942e-05
Number of threads: 4
Total time: 0.00521531
Average time: 0.000260765
Standard Deviation: 2.29583e-05
Number of threads: 8
Total time: 0.00286039
Average time: 0.000143019
Standard Deviation: 2.32686e-05
Number of threads: 16
Total time: 0.00172997
Average time: 8.64986e-05
Standard Deviation: 3.86149e-05
Number of threads: 20
Total time: 0.00203175
Average time: 0.000101587
Standard Deviation: 0.000169211
Number of threads: 24
```

```
Total time: 0.00117651
Average time: 5.88256e-05
Standard Deviation: 2.20737e-05
Number of threads: 28
Total time: 0.00257025
Average time: 0.000128513
Standard Deviation: 0.000348476
Number of threads: 32
Total time: 0.00275025
Average time: 0.000137513
Standard Deviation: 0.000413398
```

Exercise 4

1. Without parallelization:

```
DFTW calculation with N = 8000
DFTW computation in 3.444311 seconds
Xre[0] = 8000.000000
```

   With:

```
DFTW calculation with N = 8000
DFTW computation in 0.662216 seconds
Xre[0] = 8000.000000
```

2. Running a performance test as requested yields the following.

```
DFTW calculation with N = 10000
DFTW computation in average of 1.332611 seconds
Standard Deviation: 0.002073
Xre[0] = 10000.000000
```

3.

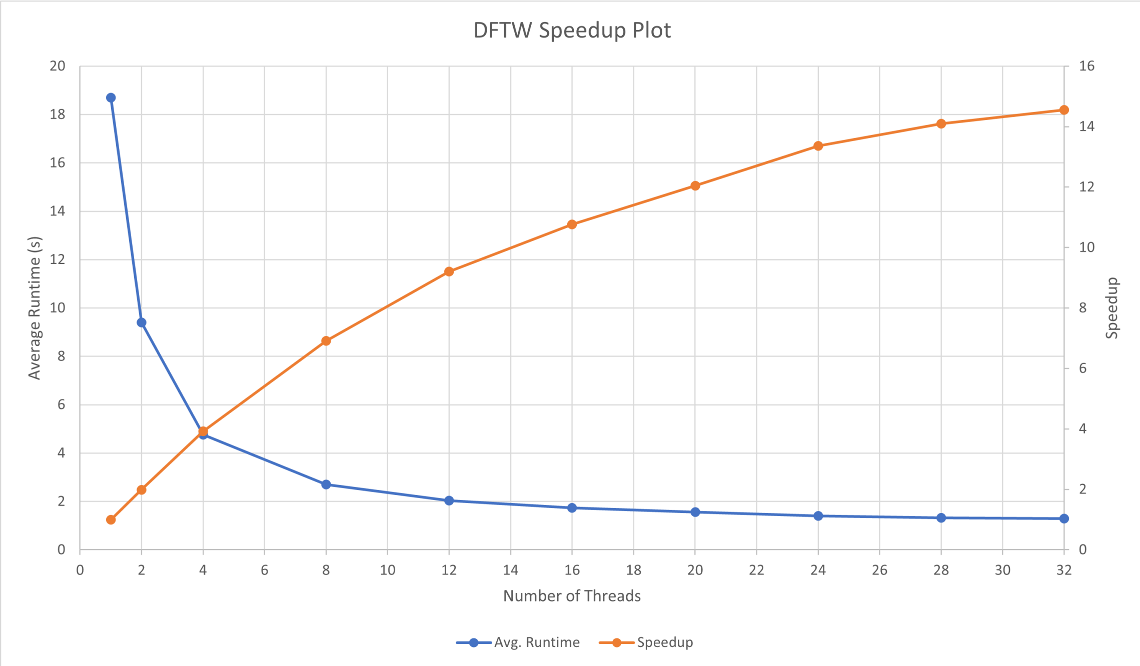


Figure 1: Speedup and average runtime while varying number of threads.

4. DFT input matrices are symmetric along the diagonal. One possible optimization could make use of this fact to reduce the number of iterations per loop and improve temporal locality. Another is blocking of the input matrices, allowing more efficient use of the cache.