Masterpraktikum Scientific Computing

High Performance Computing Tutorial 2

Session 2: OpenMP

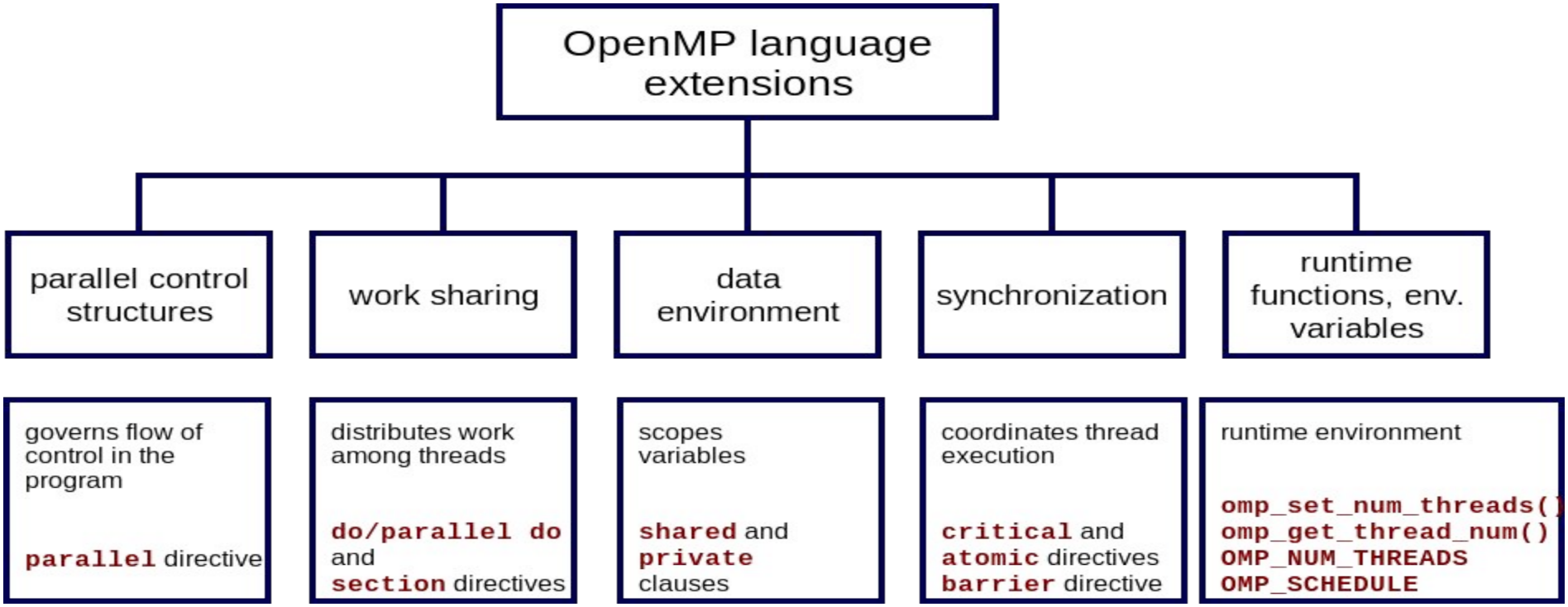
Authors
Mantosh Kumar (Matriculation number : 03662915)
João Trindade (Matriculation number : 03673251)

Course of Study: Master of Science Informatics

What is OpenMP

1. Open Multi-Processing
2. An API for writing multithreaded applications
3. A set of compiler directives and callable runtime library routines for parallel application programmers
4. Shared memory parallelism
5. Greatly simplifies writing multi-threaded programs in Fortran, C and C++
6. OpenMP functions are included in a header file labelled omp.h in C/C++
7. Standardizes last 20 years of SMP practice

OpenMP Constructs



Exercise 1: Shared memory PI calculation

Serial

```
for (i = 0; i < n; i++) {
   x = (i+0.5)/n;
   area += 1.0/(1.0 + x*x);
}
```

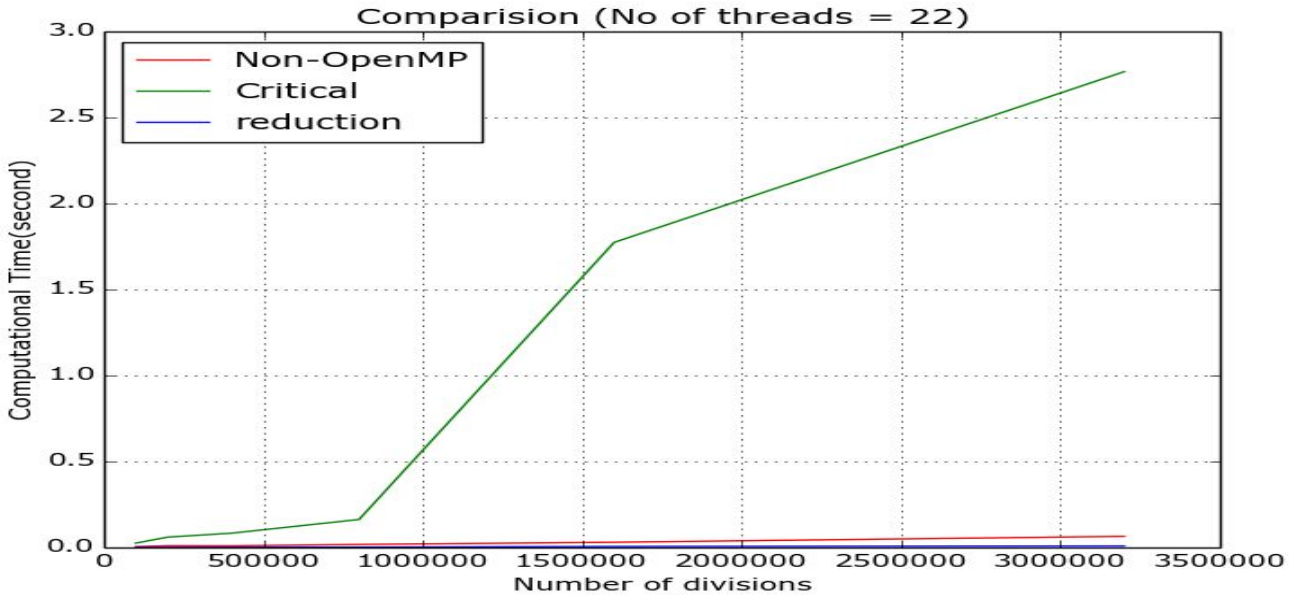
critical directive

```
#pragma omp parallel for private(x)
for (i = 0; i < n; i++) {
          x = (i+0.5)/n;
          #pragma omp critical
          area += 1.0/(1.0 + x*x);
}
```

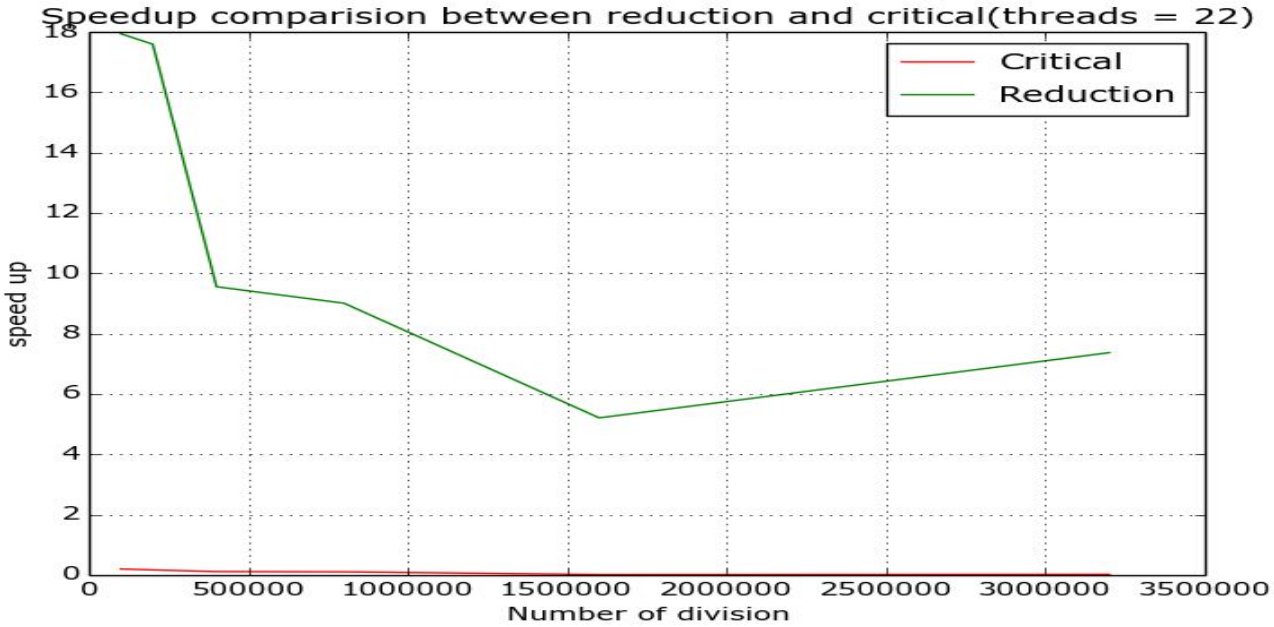
```
#pragma omp parallel for private(x) reduction(+:area)
for (i = 0; i < n; i++) {
   x = (i+0.5)/n;
   area += 1.0/(1.0 + x*x);
}
```

reduction clause

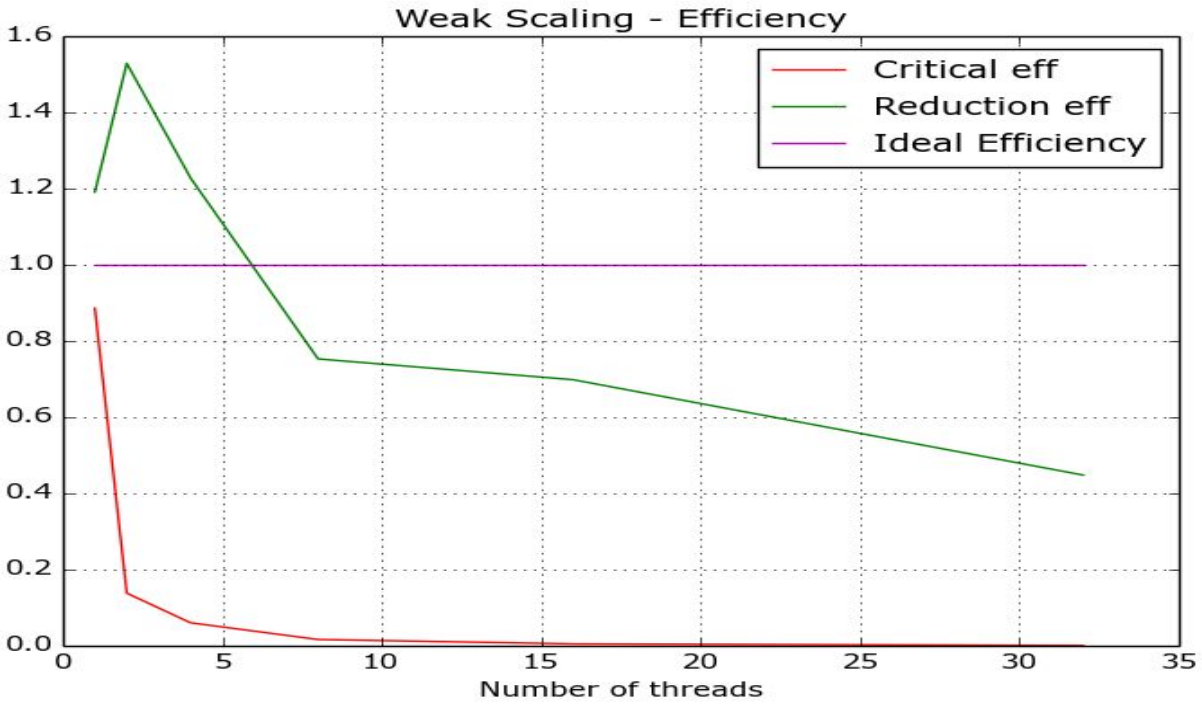
Ex 1: Comparison among these methods in terms of computation time



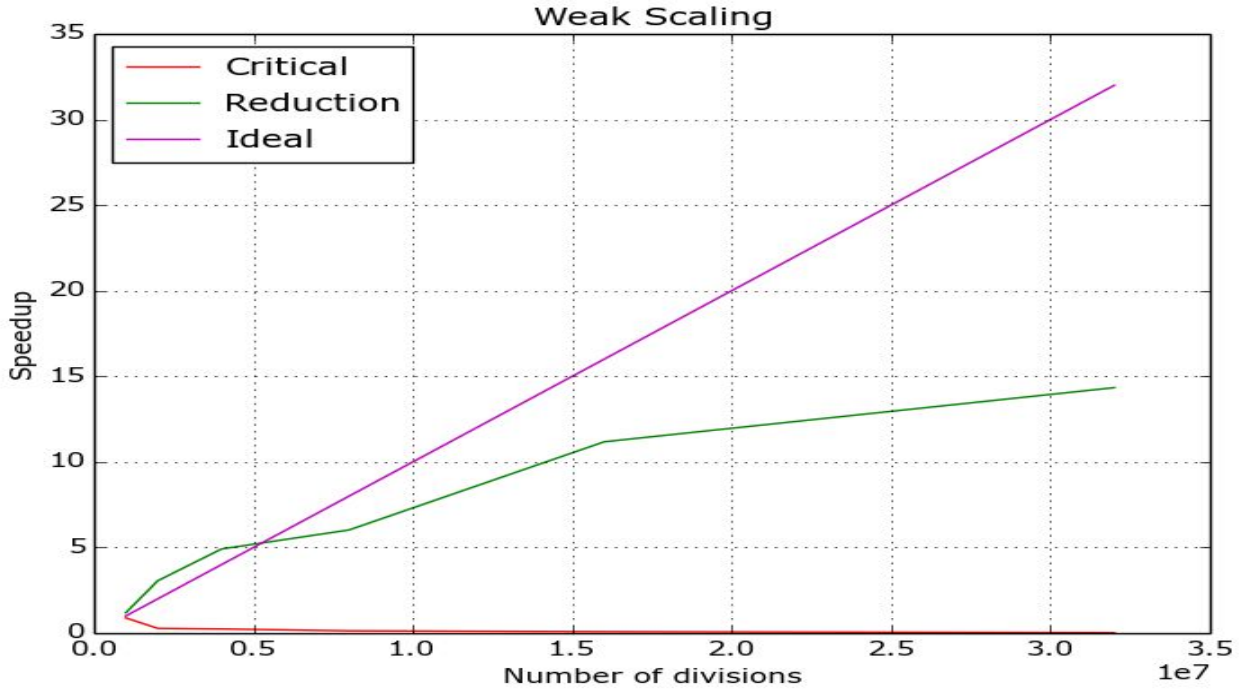
Ex 1: Comparison between reduction-critical methods in terms of speed up



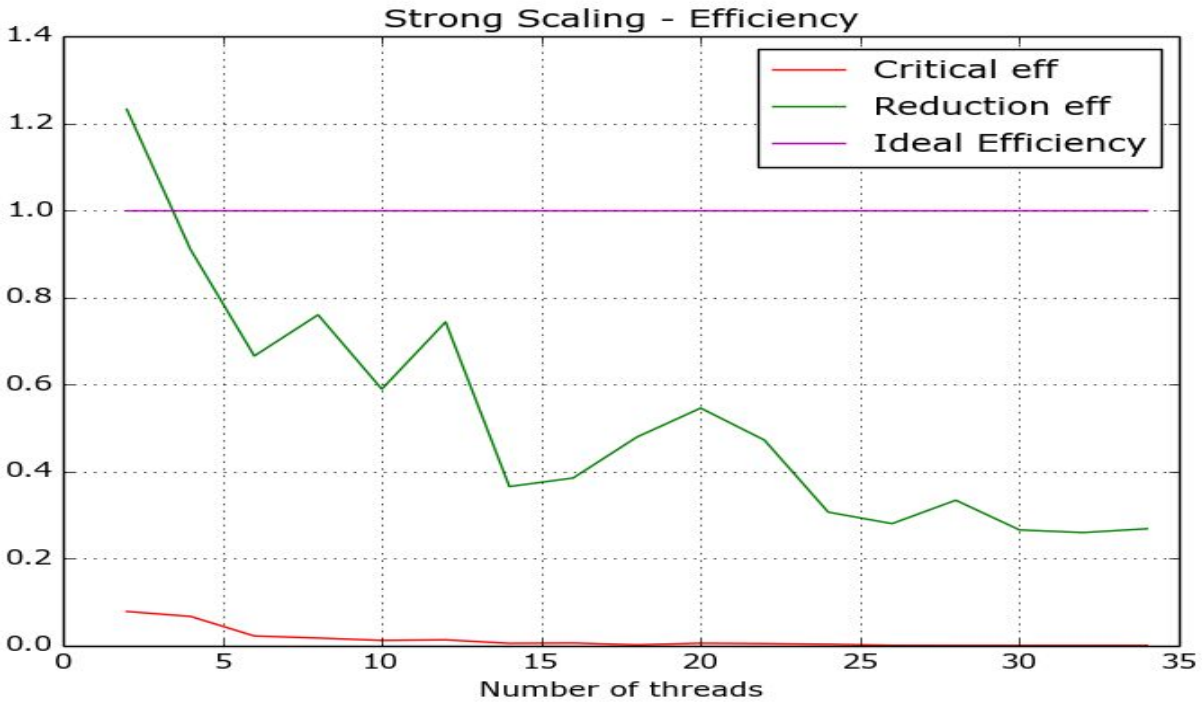
Ex 1 - Results - Weak Scaling (Efficiency vs Number of threads)



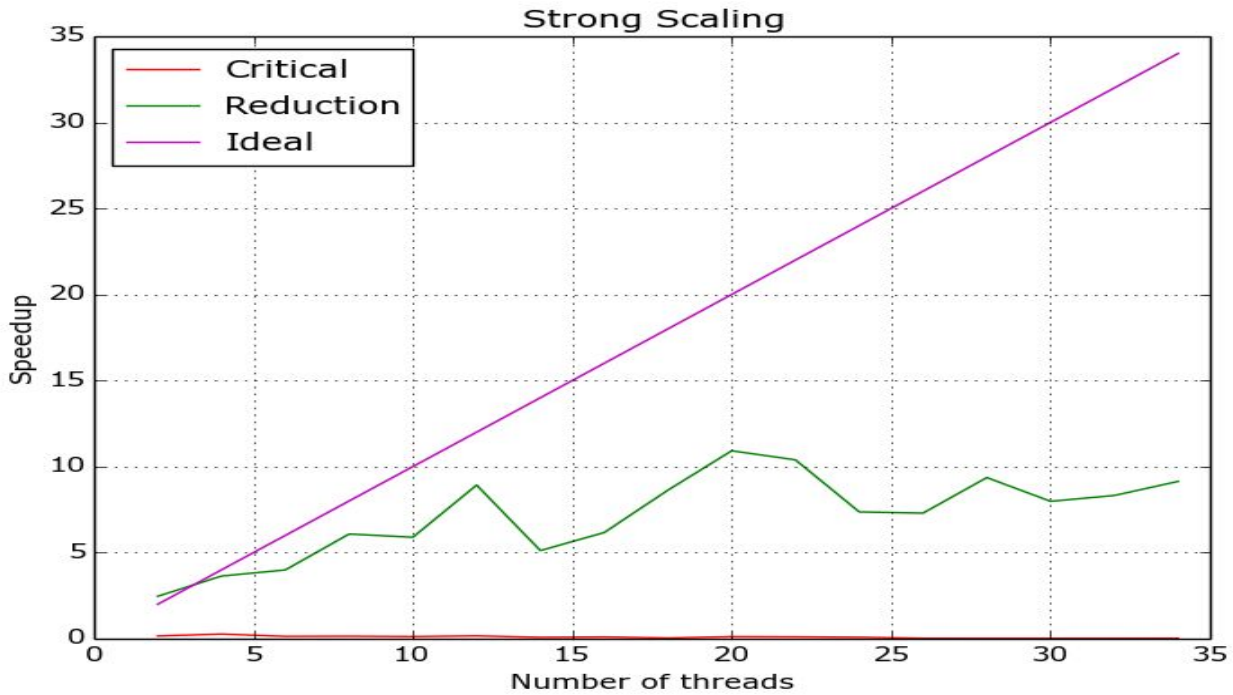
Ex 1 - Results - Weak Scaling (Speedup vs Number of problem size)



Ex 1 - Results - Strong Scaling - problem size=8000000 (efficiency vs Number of threads)



Ex 1 - Results - Strong Scaling - problem size=8000000 (speedup vs Number of threads)



Question 2

Copy : measures the transfer rate in the absence of arithmetic operations

Scale: adds one simple scalar operation to the Copy Benchmark.

Sum: accesses 3 memory values instead of just 2

Triad: Sum Benchmark with 1 scalar operation added to one of the fetched elements.

Question 2

Due to the unavailability of the CoolMuc2 the benchmark was run on CoolMac - Mac Cluster

2.2 Max Bandwidth - Using 16 Cores with OpenMP

```
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       116812.6440       0.0003       0.0003       0.0004
Scale:       77672.2963       0.0004       0.0004       0.0005
Add:         80919.0482       0.0006       0.0006       0.0006
Triad:      100212.3405       0.0005       0.0005       0.0006
```

From 77GB/s to 110 GB/s

Question 2

2.3 Half the cores & different Pinning Strategies

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	38468.8243	0.0009	0.0008	0.0009
Scale:	41502.0804	0.0008	0.0008	0.0009
Add:	35823.2370	0.0014	0.0013	0.0015
Triad:	35874.3036	0.0014	0.0013	0.0014

8 Cores by Default (granularity=thread,compact,1,0)

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	78627.8430	0.0005	0.0004	0.0005
Scale:	57877.4161	0.0006	0.0006	0.0007
Add:	68269.4446	0.0007	0.0007	0.0007
Triad:	67604.6313	0.0007	0.0007	0.0008

8 Cores Using 2 Sockets

KMP_AFFINITY="granularity=core,explicit,proclist=[0,1,2,3,8,9,10,11],verbose"

Question 2

2.4 Allocate the array in Non-Local Memory

Exploiting OpenMP “first-touch policy”

```
#pragma omp single nowait
for (j=0; j<N; j++) {
    a[j] = 1.0;
    b[j] = 2.0;
    c[j] = 0.0;
}
```

KMP_AFFINITY="granularity=core,explicit,proclist=[0,8,9,10,11,1,2,3],verbose"

```
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       52800.0504       0.0007       0.0006       0.0009
Scale:      41838.4439       0.0008       0.0008       0.0009
Add:        35270.9516       0.0014       0.0014       0.0016
Triad:      44121.5411       0.0011       0.0011       0.0012
-------------------------------------------------------------
```

Question 2

2.5 Allocate the array in Non-Local Memory

Intel Xeon E5-2670 specifications [link]
L1 Cache: 32KB /core
L2 Cache: 256 KB /core
L3 Cache: 20 MB

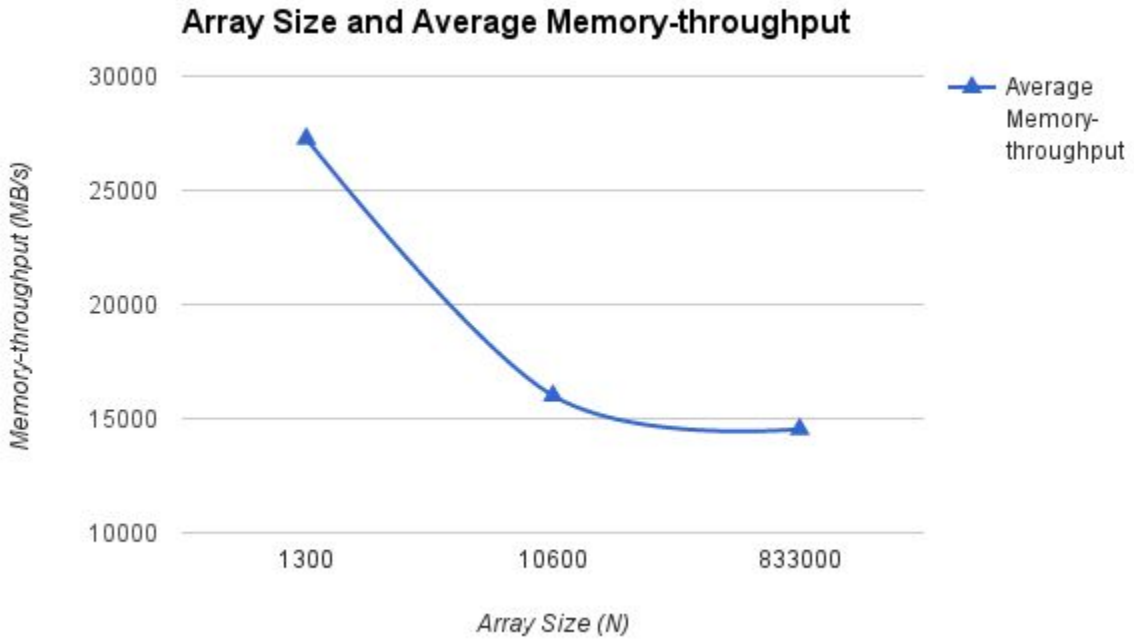
Stream Benchmark uses 3 Arrays of Double Values (8 Bytes)

L1 test: 32000/8 = 4000 Elements. 4000/3 = 1300(approx.) elements per array

L2 test 256000/8 = 32000 Elements. 32000/3 = 10600 (approx.) elements per array.

L3 test 20000000/8 = 2500000 Elements. 2500000/3 = 833000 (approx.) elements per array.

Question 2

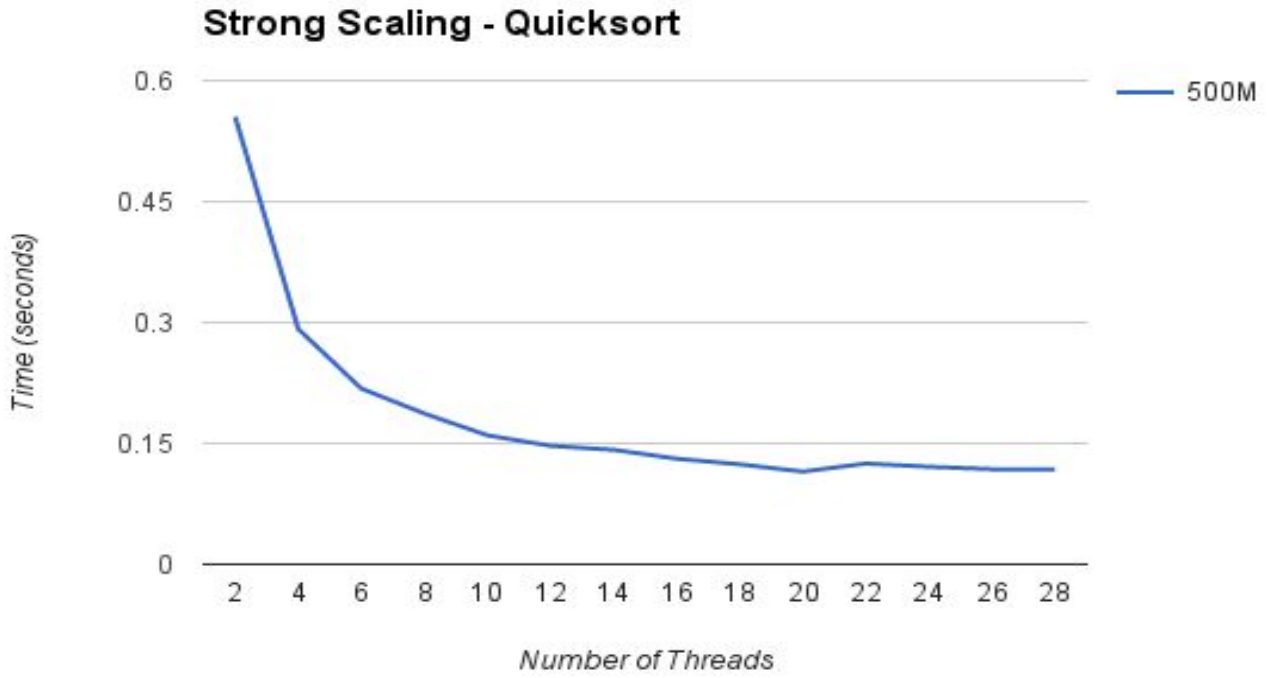

Array Size and Average Memory-throughput
30000
25000
20000
15000
10000
Memory-throughput (MB/s)
1300
10600
833000
Array Size (N)
Average Memory-throughput

Question 3

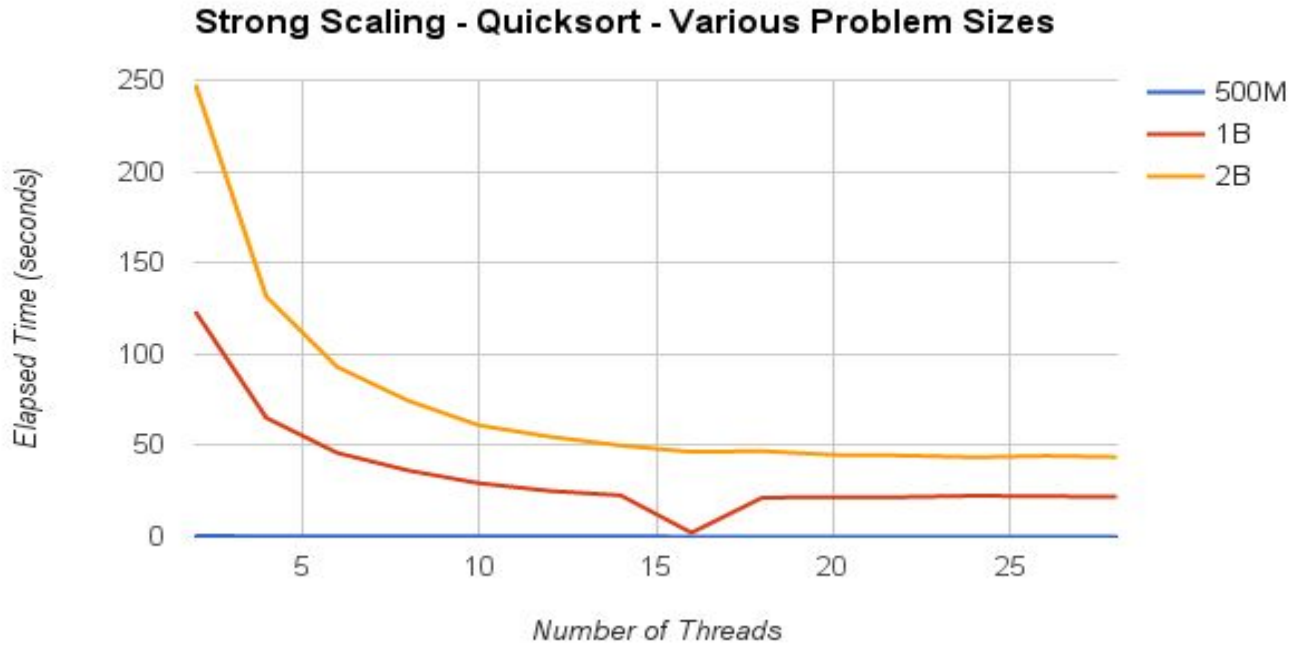
```
#pragma omp task final(right < THRESHOLD)
quicksort(data, right);

#pragma omp task final((length - left) < THRESHOLD)
quicksort(&(data[left]), length - left);
```

Threshold used was 10k

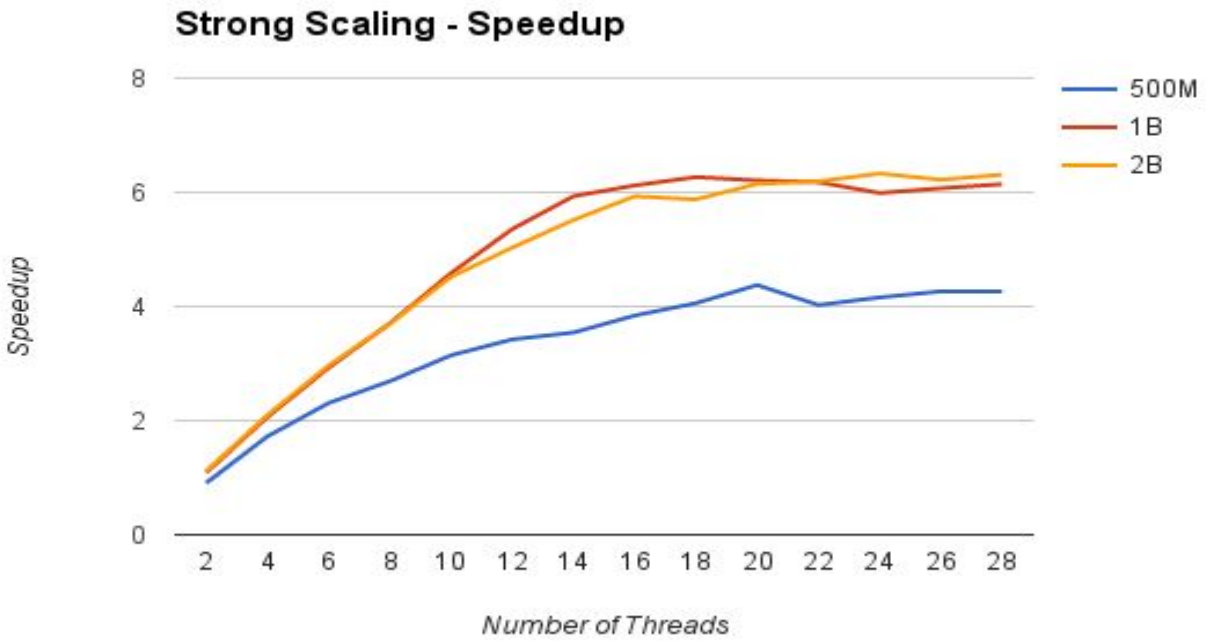
Question 3 - Results - N=500M



Question 3 - Results - N=500M, N=1B, N=2B



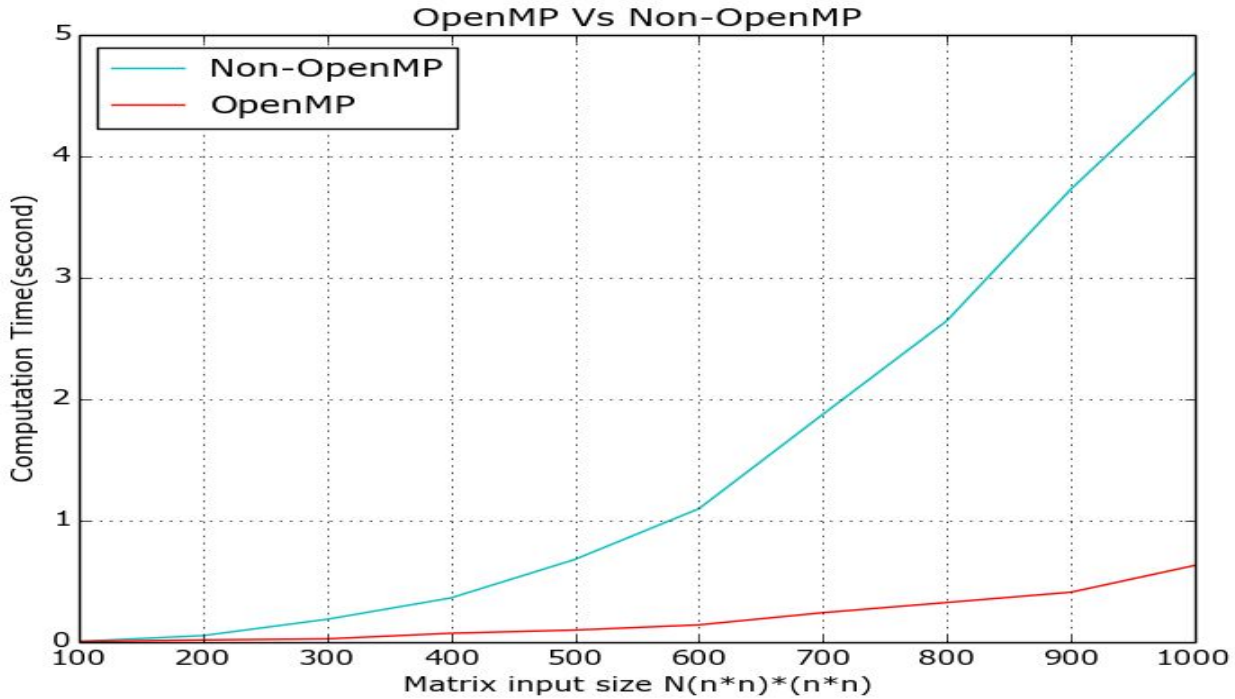
Question 3 - Results - Speedup - N=500M, N=1B, N=2B



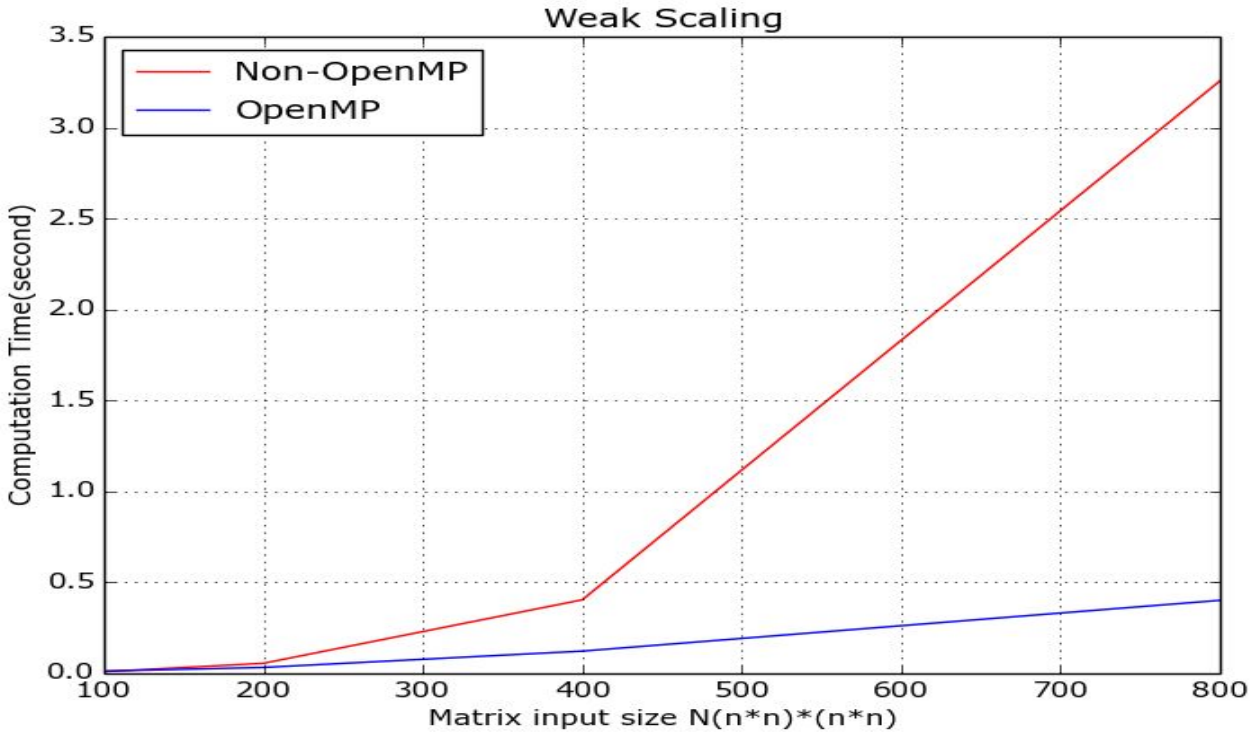
Question 4 - Matrix-Matrix-Multiplication II

```
/* Parallelizing matrix multiplication with OpenMP where each cache-block is handled by only one thread starts here*/
memset(c, 0, mem_size);
time_marker_t time = get_time();
#pragma omp parallel default(none) shared(block_size, n, a, b, c) private(i, j, k, ii, jj, kk) reduction(+:num_thread)
{
        num_thread += 1;

        #pragma omp for schedule(dynamic)
        for(i = 0; i < n; i += block_size){
                for(j = 0; j < n; j += block_size){
                        for(k = 0; k < n; k += block_size){

                                for(ii = i; ii < min(i + block_size, n); ii++) {
                                        for(jj = j; jj < min(j + block_size, n); jj++) {
                                                for(kk = k; kk < min(k + block_size, n); kk++) {
                                                        c[ii * n + jj] += a[ii * n + kk] * b[kk * n + jj];
                                                }
                                        }
                                }
                        }
                }
        }


        #pragma omp barrier
}
```

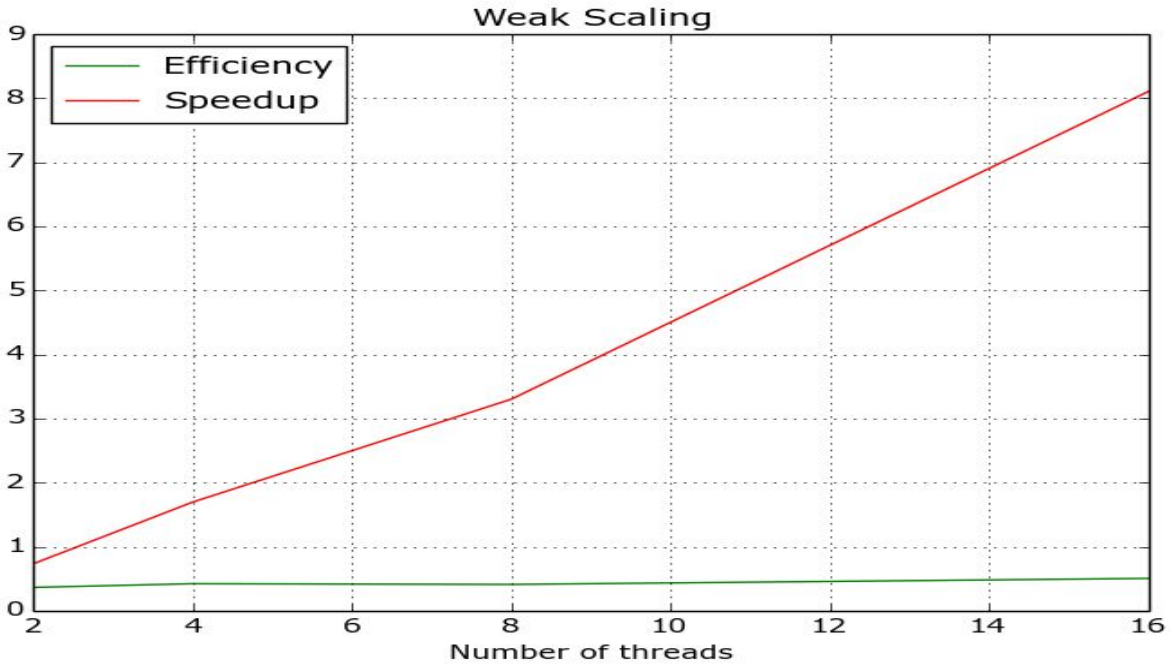
Question 4 - Results - OpenMP Vs Non-OpenMP: Number of threads = 20



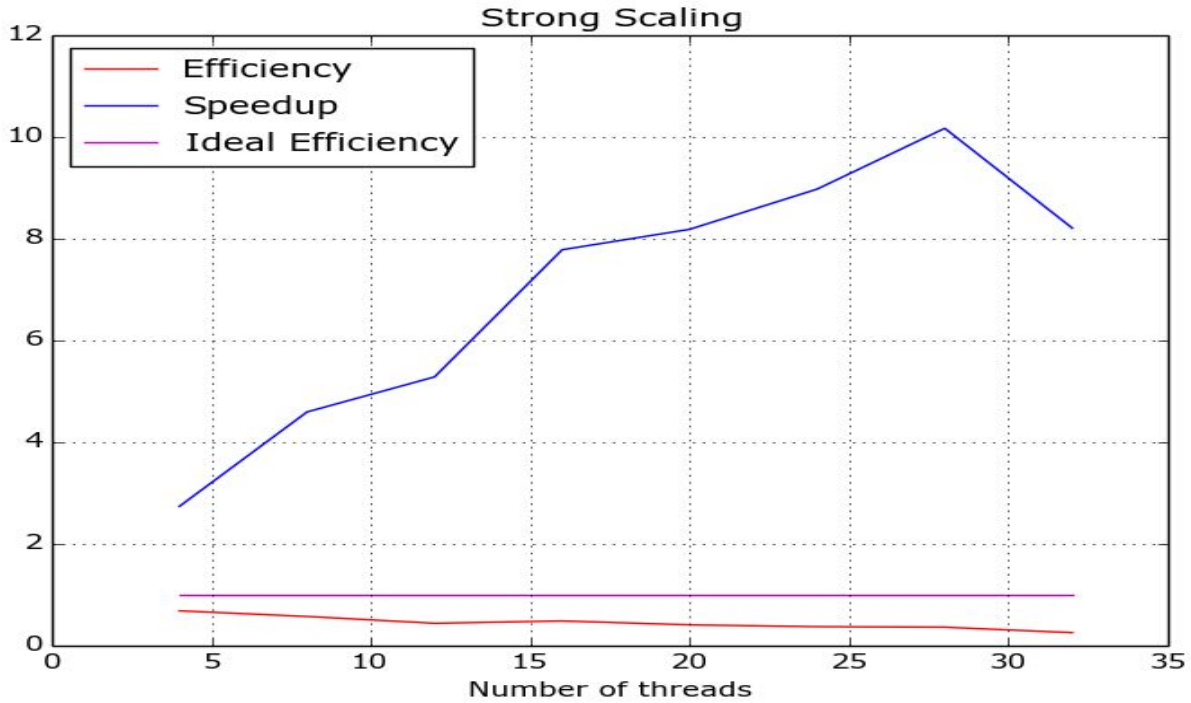
Question 4 - Results - Weak Scaling (Computational time vs matrix size)



Question 4 - Results - Weak Scaling (speedup and eff compare to number of threads)



Question 4 - Results - Strong Scaling (efficiency and speedup vs Number of threads)



Danke!