

This assignment covers shared memory parallelization with OpenMP. With shared memory parallelism we are able to use a single compute node up to its full extent. Together with MPI covered in the third assignment we will have all tools at hand to unleash the complete power of state-of-the-art supercomputers.

## Literature

- *OMP Tutorial*: <https://computing.llnl.gov/tutorials/openMP/>
- *Intel® 64 and IA-32 Architectures Optimization Reference Manual*: <http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html>
- Chapter *Thread Affinity Interface of the User and Reference Guide for the Intel® C++ Compiler*: <https://software.intel.com/en-us/node/522691>
- *How to optimize DGEMM*: <http://wiki.cs.utexas.edu/rvdg/HowToOptimizeGemm/>
- *GEMM: From Pure C to SSE Optimized Micro Kernels*: <http://apfel.mathematik.uni-ulm.de/~lehn/sghpc/gemm/index.html>

## 1 Shared memory $\pi$ -calculation

With  $\phi(x) = \frac{1}{1+x^2}$  we have  $\int \phi(x) dx = \arctan(x)$ .

Hence,  $\pi$  can be calculated through integration of  $\phi(x)$ : <http://mathworld.wolfram.com/InverseTangent.html>.

1. Develop a serial implementation that integrates function  $\phi(x)$  over  $[0, 1]$ .
2. Parallelize your application using OpenMP. Please implement two different versions using both the *critical directive* and the *reduction clause* in order to ensure the correct summation order!
3. Perform a scaling study of your algorithm.
  - Use `OMP_NUM_THREADS` in order to start your application with different thread counts.
  - Calculate the achieved speed-up and provide interpretations. Please perform weak and strong scaling studies.
    - Weak scaling*: if you double the number of threads you also double the problem size.
    - Strong scaling*: You keep the problem size constant but increase the number of threads.

**Hint** Use the mid-point rule for integration: Split the unit-interval into  $n$  equal sized sub-intervals with length  $h = \frac{1}{n}$ . For each mid-point  $\tilde{x}_i$  of each sub-interval, calculate the value of function  $\phi(\tilde{x}_i)$ . Afterwards, sum up all function values. In the end multiply the result with  $4 \cdot h$ .

Explain why this method works.

## 2 STREAM benchmark

The STREAM benchmark <http://www.cs.virginia.edu/stream/> is a popular benchmark to measure the sustainable memory bandwidth of high performance computers. Although the benchmark was designed to measure the bandwidth of main memory, it is possible to measure the bandwidth of the different cache levels by lowering the amount of data to be “streamed” appropriately.

1. Make yourself familiar with the STREAM Benchmark. Explain the different sub-benchmarks shortly: “Copy”, “Scale”, “Add” and “Triad”.
2. Follow the instructions to adjust the compilation settings accordingly and run the benchmark on CoolMUC2. What is the maximum memory bandwidth you can achieve utilizing all cores with OpenMP?
3. Use only 14 cores. Try different pinning strategies! Show that the performance increases if more than one socket is used.
4. Inform yourself about *non-uniform memory access (NUMA)*. Show the effect of NUMA by modifying the STREAM benchmark: “Allocate” the arrays in non-local memory by exploiting OpenMP’s first-touch policy.  
*Hint:* The tool `numactl` is very useful, try for example `numactl --hardware`.
5. Modify the STREAM benchmark, such that it operates in different cache levels: L1D, L2 and L3. Plot the array-size against the memory-throughput on a single core. Can you verify the cache sizes and bandwidths of the hardware specifications?

## 3 Quicksort

1. Parallelize the quicksort implementation given in `quicksort.c`. Please employ the task-concept of OpenMP 3.1. Use the `final` clause for stopping the parallelization of the recursion at a sufficient level of the recursion.
2. Examine the scalability (strong scaling) for different problem sizes and plot your results.

## 4 Matrix-Matrix-Multiplication II

1. Parallelize your application from the first assignment with OpenMP. Please ensure that each cache-block is handled by only one thread.

2. Provide weak- and strong-scaling results by using sufficient figures and plots.

## Deliverables

The following deliverables have to be handed in no later than 08:00 AM, Monday, 21st November, 2016. If there is no submission until this deadline, the exercise sheet is graded with 0 points. Small files (<1 MB in total) can be send as an attachment directly to *uphoff AT in.tum.de* and *chaulio.ferreira AT tum.de*, larger files have to be uploaded at a place of your choice, e.g. <https://github.com/>, <http://home.in.tum.de/>, <https://syncandshare.lrz.de>. In either case inform us about the final state of your solution via e-mail.

- A short report which describes your work and answers all questions in this assignment.
- All of your code.
- Slides for the presentation during the next meeting; remember to address all questions in the tasks.
- Output of all runs. Figures (e.g. scaling graphs) if applicable.
- Documentation how to build and use your code.