Homework 3

An implementation of Laplace’s equation using POSIX threads

Deadline: January 31 2018
\[ U_{i,j}^{n+1} = \frac{1}{4} \left( U_{i-1,j}^n + U_{i+1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n \right) \]

Laplace’s equation - threads

\[
\begin{align*}
\text{for } j &= 1 \text{ to } j_{\text{max}} \\
\text{for } i &= 1 \text{ to } i_{\text{max}} \\
U_{\text{new}}(i,j) &= 0.25 \times (U(i-1,j) + U(i+1,j) + U(i,j-1) + U(i,j+1))
\end{align*}
\]
• Assuming you have a 2 dimensional matrix stored in row-major format, compute a well defined number of iterations of the computation of the Laplace equation using multiple-threads coordinated manually.
  – Special attention should be payed to minimize the extra memory requirements
  – Hint: the algorithm is highly parallelizable (it should be visible from your performance graphs)

• Originally the matrix is initialized with 0 everywhere except the boundaries (first and last row and first and last column) which are initialized differently.

• Highlight the impact of using several threads to execute this algorithm by doing an analysis of the algorithm’s performance. Vary the number of threads and the problem size (weak and strong scaling) and comment on the results.
  – Present the average over multiple runs to account for any measuring error or outside effects
• Benchmarking of the Laplace algorithm should be measured excluding the threads creation and synchronization.
  – Create a pool of threads that can be reused (assume it is a global variable hidden in your library).

• A skeleton code is available on the class website (or can be obtained by email from the TA).