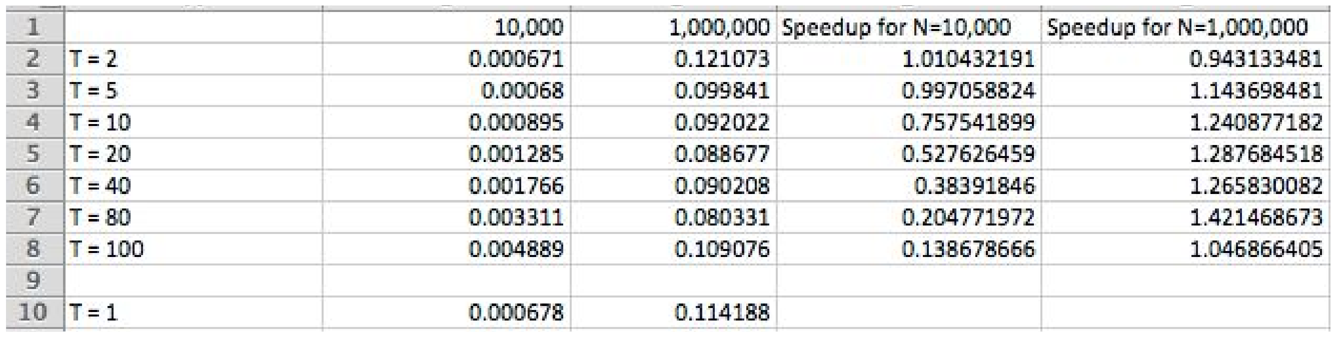
Laura Chevalier ~ lmc613
Professor Mohamed Zahran
Lab 2



1		10,000	1,000,000	Speedup for N=10,000	Speedup for N=1,000,000
2	T = 2	0.000671	0.121073	1.010432191	0.943133481
3	T = 5	0.00068	0.099841	0.997058824	1.143698481
4	T = 10	0.000895	0.092022	0.757541899	1.240877182
5	T = 20	0.001285	0.088677	0.527626459	1.287684518
6	T = 40	0.001766	0.090208	0.38391846	1.265830082
7	T = 80	0.003311	0.080331	0.204771972	1.421468673
8	T = 100	0.004889	0.109076	0.138678666	1.046866405
9					
10	T = 1	0.000678	0.114188		

The speedup for *n* = 10,000 actually gets worse as more threads are added. This suggests that the costs of using OpenMP when *n* = 10,000 do not outweigh the benefits for *t* > 2 (number of threads more than 2). Overheads like thread creation, communication (costs associated with memory access), load imbalance, and synchronization prevent the benefits of parallelization from generating increasing speedup for increasing *t*. Instead speedup decreases as we add more threads.

On the other hand, the speedup for *n* = 1,000,000 increases until *t* = 100. Because *n* is large, the benefits of parallelization with OpenMP outweigh its costs. A larger problem size merits a greater number of threads because the cost of their creation and the communication they require is not significant next to the benefits of executing the code in parallel. However, this is only true up to a point. The speedup for *t* = 100 was less than for *t* = 80 because at that point, the costs of parallelization begin to weigh heavier — creating and dealing with more and more threads eventually becomes less beneficial, until it reaches a point where you're better off not parallelizing at all.