Cameron Calv and Nicholas Sica
ECEC 413 Intro to Parallel Computer Architecture
Assignment 3: Particle Swarm Optimizer (OpenMP)

Parallelization

The code was parallelized by placing OpenMP pragmas around the functions that required parallelization. These functions included 'for' loops that were interpreted by OpenMP and split into the requested number of threads. The number of threads is specified at the command line and set with a single call to OpenMP early in the code.

Evaluating Parallelization

Below are tables outlining the runtimes of the program run for parallelization. Following the tables are a plot showing the difference of the execution times for various functions and parameters.

Table 1: Evaluating the execution time of the program on Drexel COE college's Xunil server. Data run on the two functions with range [-5.12, 5.12] for Rastrigin and [-500, 500] for Schwefel and with particle count and iterations set to 10,000.

	Number of Threads – Particles			
Parameters	1	4	8	16
Rastrigin D = 10	192.742935	991.824341	1189.312012	1236.008179
Rastrigin D = 20				
Schwefel D = 10				
Schwefel D = 20				

Table 2: Evaluating the speed-up of the program on Drexel COE college's Xunil server. Data run on the two functions with range [-5.12, 5.12] for Rastrigin and [-500, 500] for Schwefel and with particle count and iterations set to 10,000.

	Number of Threads – Particles			
Parameters	1	4	8	16
Rastrigin D = 10	192.742935	991.824341	1189.312012	1236.008179
Rastrigin D = 20				
Schwefel D = 10				
Schwefel D = 20				

Table 3: Evaluating the execution time per thread of the program on Drexel COE college's Xunil server. Data run on the two functions with range [-5.12, 5.12] for Rastrigin and [-500, 500] for Schwefel and with particle count and iterations set to 10,000.

	Number of Threads – Particles			
Parameters	1	4	8	16
Rastrigin D = 10	192.742935	991.824341	1189.312012	1236.008179
Rastrigin D = 20				
Schwefel D = 10				
Schwefel D = 20				

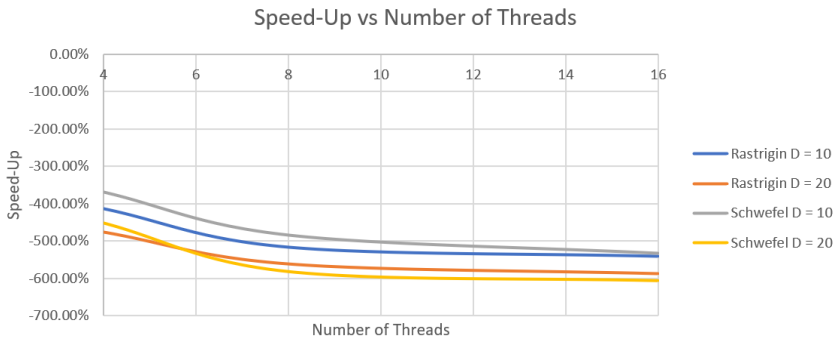

Figure 1: Plot showing the speedup per number of threads for all tests.

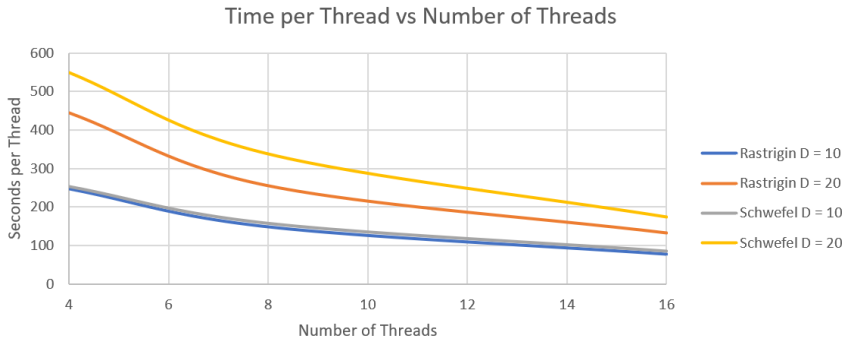

Figure 2: Plot showing the average amount of time per thread for all tests.

From the data it appears that increasing the thread count does not assist with the problem to a great extent. This may be because the problem size is not large enough for the effect of improvement to be seen easily. It is important to note though that the average time of the program for each thread added does not increase but decreases making it seem as though the issue is not primarily caused by a limit of cores.