|  |  |  |  |
| --- | --- | --- | --- |
| Solution type | Number of Slaves | Execution time  (seconds) | Explain the result |
| Sequential Solution | 1 | 8.423 | 1. |
| Static Task Pool | 2 | 5.298 | 2. |
| Static Task Pool | 4 | 4.188 | 3. |
| Dynamic Task Pool | 2 | 4.478 | 4. |
| Dynamic Task Pool | 4 | 2.377 | 5. |
| Dynamic Task Pool | 20 | 1.976 | 6. |

1. This result is the worst result in this table, because the sequential solution calculates a very heavy function for 400 times, without dividing the heavy work into parallel smaller pieces to take advantage of the computer cores.
2. This result is better than the sequential part, due to the parallel division of the whole task into two subtasks. Each subtask was performed within two different cores of the computer in the same moment, to save about 50% working time (not exactly 50%, because the MPI commands take additional time and there is a small part in the first of this program which couldn't run in parallel).
3. This result is better than the 2-processes static part, due to the parallel division of the whole task into another two subtasks – four subtasks. Each subtask was performed within four different cores of the computer in the same moment. In the beginning of this whole task, there is still a part which couldn't run in parallel.
4. This result is even better than the 2-processes static part. It takes the same considerations of the corresponding static program, but there is also an additional consideration which explains the better result: The heavy function operates in a way that some processes commit a relatively shorter job than some of the other ones. Here, for this part, as explained one process committed a shorter job than the other, due to the structure of the heavy function. The corresponding static 2-processes program does not take this into consideration, as the dynamic program does- and separates the whole task into approximately equal parts.
5. As the same considerations above, the dynamic task pool of 4 processes does a better job in terms of runtime, than the static one. The heavy calculations split off into smaller calculations, in terms of runtime measurement, to utilize this by keeping the 4 cores equally busy.
6. Like the same considerations above, as there are more processes to take subtasks, there is a more accurate, equally work division between them. It is important to add that there is a harder race condition between the processes, as there are more processes than cores (16 at the best)- that why the improvement in runtime here is not so significant.