Assignment 02

Exercise 1:

1. Write a sequential application pi\_seq in C or C++ that computes π for a given number of samples (command line argument). Test your application for various, large sample sizes to verify the correctness of your implementation.

Attached in the appendix

pi\_seq.cpp

1. Consider a parallelization strategy using MPI. Which communication pattern(s) would you choose and why?

Since the sampling process itself does not require any communication each process can produce random samples and check if they are inside or outside the circle. After all processes are finished for evaluating pi the only information which is required to be communicated is the number of samples inside the circle. This can be achieved by using an MPI\_Reduce with MPI\_SUM as reduction.

1. Implement your chosen parallelization strategy as a second application pi\_mpi. Run it with varying numbers of ranks and sample sizes and verify its correctness by comparing the output to pi\_seq.

Attached in the appendix

pi\_mpi.cpp

1. Discuss the effects and implications of your parallelization.

All times shown are the median of three executions.

We measured the performance using 1e8 and 5e8 samples. The parallel as well as the sequential version achieved a precision of 4 to 5 decimal digits. As expected, a run with more samples achieves a higher efficiency since we have a lower overhead compared to the total work. In the 5e8 sample run we have a linear increasing speedup but only attain 50% compared to the linear speedup marked in orange. In general, both efficiency plots show clearly that its better to increase the number of cores on one node before increasing the total number of nodes

Number of samples: 5e8

Sequential time: 36,374sec

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | Cores/Node | Total cores | Median [sec] | Absolute speedup | Efficiency |
| 1 | 2 | 2 | 34,7466 | 1,05 | 0,53 |
| 1 | 4 | 4 | 17,3954 | 2,09 | 0,52 |
| 1 | 8 | 8 | 8,71632 | 4,17 | 0,52 |
| 2 | 1 | 2 | 34,7556 | 1,05 | 0,53 |
| 2 | 2 | 4 | 17,4002 | 2,09 | 0,52 |
| 2 | 4 | 8 | 8,719 | 4,17 | 0,52 |
| 2 | 8 | 16 | 4,48093 | 8,12 | 0,51 |
| 4 | 1 | 4 | 17,4119 | 2,09 | 0,52 |
| 4 | 2 | 8 | 8,7375 | 4,16 | 0,52 |
| 4 | 4 | 16 | 4,41466 | 8,24 | 0,52 |
| 4 | 8 | 32 | 2,2472 | 16,19 | 0,51 |
| 8 | 1 | 8 | 8,7889 | 4,14 | 0,52 |
| 8 | 2 | 16 | 4,41282 | 8,24 | 0,52 |
| 8 | 4 | 32 | 2,25319 | 16,14 | 0,5 |
| 8 | 8 | 64 | 1,19985 | 30,32 | 0,47 |

Efficiency

Nodes

Cores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 8 |
| 1 |  | 0,53 | 0,52 | 0,52 |
| 2 | 0,53 | 0,52 | 0,52 | 0,52 |
| 4 | 0,52 | 0,52 | 0,52 | 0,5 |
| 8 | 0,52 | 0,51 | 0,51 | 0,47 |

Speedup:

Number of samples: 1e8

Sequential time: 7,27507sec

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | Cores/Node | Total cores | Time [sec] | Absolut speedup | Efficiency |
| 1 | 2 | 2 | 6,95832 | 1,05 | 0,53 |
| 1 | 4 | 4 | 3,47791 | 2,09 | 0,52 |
| 1 | 8 | 8 | 1,74942 | 4,16 | 0,52 |
| 2 | 1 | 2 | 6,96839 | 1,04 | 0,52 |
| 2 | 2 | 4 | 3,50473 | 2,08 | 0,52 |
| 2 | 4 | 8 | 1,7684 | 4,11 | 0,51 |
| 2 | 8 | 16 | 0,901192 | 8,07 | 0,5 |
| 4 | 1 | 4 | 3,50281 | 2,08 | 0,52 |
| 4 | 2 | 8 | 1,78711 | 4,07 | 0,51 |
| 4 | 4 | 16 | 0,920108 | 7,91 | 0,49 |
| 4 | 8 | 32 | 0,486324 | 14,96 | 0,47 |
| 8 | 1 | 8 | 1,80764 | 4,02 | 0,5 |
| 8 | 2 | 16 | 0,949553 | 7,66 | 0,48 |
| 8 | 4 | 32 | 0,505753 | 14,38 | 0,45 |
| 8 | 8 | 64 | 0,288551 | 25,21 | 0,39 |

Efficiency

Nodes

Cores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 8 |
| 1 |  | 0,52 | 0,52 | 0,5 |
| 2 | 0,53 | 0,52 | 0,51 | 0,48 |
| 4 | 0,52 | 0,51 | 0,49 | 0,45 |
| 8 | 0,52 | 0,5 | 0,47 | 0,39 |

Speedup:

Exercise 2:

1. Consider a parallelization strategy using MPI. Which communication pattern(s) would you choose and why? Are there additional changes required in the code beyond calling MPI functions? If so, elaborate!

The basic idea is to split up the stencil in equal sized parts and distribute them to all cores available. In this case we need to communicate after each time step. We decided to use MPI\_Send and MPI\_Recv for communication. Our strategy is to only send the edge cells to the neighbours. When sending we have to be aware of edge cases e.g. the process with rank 0 doesn’t need to send his left element. Furthermore, we need to consider the send and receive ordering to avoid deadlocks.

1. Implement your chosen parallelization strategy as a second application heat\_stencil\_1D\_mpi. Run it with varying numbers of ranks and problem sizes and verify its correctness by comparing the output to heat\_stencil\_1D\_seq.

Attached in the appendix

Heat\_stencil\_1D\_mpi.cpp

1. Discuss the effects and implications of your parallelization.

All measurements shown are the median of three executions. In both the sequential and the parallel version, the printing was disabled to omit I/O operation, hence only the actual calculation got measured.

When measuring first with a problem size of 1024 and 512000 timesteps it clearly can be seen that the speedup and the efficiency are very bad. This is because we have to communicate after each timestep. The only noticeably speedup was achieved when staying at one CPU and use 2 or 4 cores. All configurations using multiple nodes weren’t able to obtain good results.

For the second benchmark we reduced the number of time steps and at the same time increase the problem size. By decreasing the number of time steps we have less communication. In the efficiency and speedup plot we can see that performance increase compared to the first benchmark. By changing the problem size and reducing the communication we still achieve good results when increasing the number of nodes.

Problem size: 1024 Time steps: 512000

Sequential time: 6,434181sec

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | Cores/Node | Total cores | Time [sec] | Absolut speedup | Efficiency |
| 1 | 2 | 2 | 4,82032 | 1,33 | 0,67 |
| 1 | 4 | 4 | 3,59255 | 1,79 | 0,45 |
| 1 | 8 | 8 | 3,3086 | 1,94 | 0,24 |
| 2 | 1 | 2 | 9,41561 | 0,68 | 0,34 |
| 2 | 2 | 4 | 7,48037 | 0,86 | 0,22 |
| 2 | 4 | 8 | 11,1011 | 0,58 | 0,07 |
| 2 | 8 | 16 | 55,0837 | 0,12 | 0,01 |
| 4 | 1 | 4 | 10,5231 | 0,61 | 0,15 |
| 4 | 2 | 8 | 12,3811 | 0,52 | 0,07 |
| 4 | 4 | 16 | 65,6828 | 0,1 | 0,01 |
| 4 | 8 | 32 | 187,396 | 0,03 | 0 |
| 8 | 1 | 8 | 14,2207 | 0,45 | 0,06 |
| 8 | 2 | 16 | 16,2226 | 0,4 | 0,03 |
| 8 | 4 | 32 | 119,487 | 0,05 | 0 |
| 8 | 8 | 64 | 333,942 | 0,02 | 0 |

Efficiency

Nodes

Cores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 8 |
| 1 |  | 0,34 | 0,15 | 0,06 |
| 2 | 0,67 | 0,22 | 0,07 | 0,03 |
| 4 | 0,45 | 0,07 | 0,01 | 0 |
| 8 | 0,24 | 0,01 | 0 | 0 |

Speedup

Problem size: 65536 Time steps: 65536

Sequential time: 57,31sec

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | Cores/Node | Total cores | Time [sec] | Absolut speedup | Efficiency |
| 1 | 2 | 2 | 33,2202 | 1,73 | 0,87 |
| 1 | 4 | 4 | 16,7911 | 3,41 | 0,85 |
| 1 | 8 | 8 | 8,65194 | 6,62 | 0,83 |
| 2 | 1 | 2 | 33,677 | 1,7 | 0,85 |
| 2 | 2 | 4 | 17,3035 | 3,31 | 0,83 |
| 2 | 4 | 8 | 9,70184 | 5,91 | 0,74 |
| 2 | 8 | 16 | 6,46244 | 8,87 | 0,55 |
| 4 | 1 | 4 | 17,5248 | 3,27 | 0,82 |
| 4 | 2 | 8 | 9,79974 | 5,85 | 0,73 |
| 4 | 4 | 16 | 6,8209 | 8,4 | 0,53 |
| 4 | 8 | 32 | 23,3925 | 2,45 | 0,08 |
| 8 | 1 | 8 | 9,77527 | 5,86 | 0,73 |
| 8 | 2 | 16 | 6,54164 | 8,76 | 0,55 |
| 8 | 4 | 32 | 16,2254 | 3,53 | 0,11 |
| 8 | 8 | 64 | 43,6516 | 1,31 | 0,02 |

Efficiency

Nodes

Cores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 8 |
| 1 |  | 0,85 | 0,82 | 0,73 |
| 2 | 0,87 | 0,83 | 0,73 | 0,55 |
| 4 | 0,85 | 0,74 | 0,53 | 0,11 |
| 8 | 0,83 | 0,55 | 0,08 | 0,02 |

Speedup