|  |  |  |  |
| --- | --- | --- | --- |
|  | 100 | 1000 | 10000 |
| Sequential (OpenMP) | 1 | 1 | 1 |
| Sequential (MPI) | 0.9322 | 0.9590 | 0.9386 |
| MPI | 1.9018 (4 processes) | 8.6201 (16 processes) | 13.2707 (16 processes) |
| OpenMP | 2.1163 (4 threads) | 13.4118 (16 threads) | 14.8213 (32 threads) |
| MPI + OpenMP | 1.1710  (2 processes, 2 threads) | 10.6723  (4 processes, 4 threads) | 18.5321  (8 processes, 32 threads) |

Note: Cannot solve for problem size of 100000 due to the storage limit of 4GB on CIMS machine.

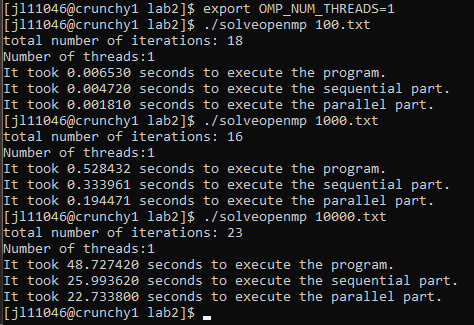
Note: Choices of number of threads are {2, 4, 8, 16, 32, 64}. Choices of number of processes are {2, 4, 8, 16, 32}.

Question 1: Are there differences between sequential OpenMP and sequential MPI? And why?

In terms of execution time, there are not much difference between sequential OpenMP and sequential MPI. Only that sequential MPI runs a little slower than sequential OpenMP, because the cost of creating processes and communications among processes are more expensive than those of threads.

Question 2: What is the best configuration for each problem size?

Screenshot showing the portion of sequential vs. parallel part in an OpenMP program:

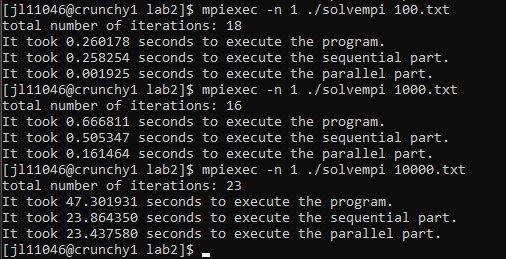


For problem size of 100, proportion of parallel part = 27.7%

For problem size of 1000, proportion of parallel part = 36.8%

For problem size of 10000, proportion of parallel part = 46.7%

Screenshot showing the proportion of sequential vs. parallel part in an MPI program:

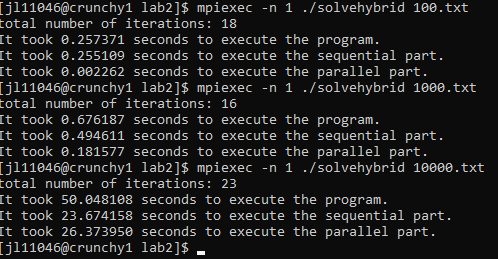


For problem size of 100, proportion of parallel part = 0.7%

For problem size of 1000, proportion of parallel part = 24.2%

For problem size of 10000, proportion of parallel part = 49.5%

Screenshot showing the proportion of sequential vs. parallel part in a hybrid program:



For problem size of 100, proportion of parallel part = 0.9%

For problem size of 1000, proportion of parallel part = 26.9%

For problem size of 10000, proportion of parallel part = 52.7%

The observation is that:

1. The proportion of the program that is parallelizable increases with the problem size. For a problem size as large as 10000, about half the program is parallelizable. This is because as problem size grows, our matrix calculations, which are parallelizable, grow faster than the sequential part of the program, such as handling input and output.
2. For a small problem size (such as 100), an OpenMP program will have more parallelizable portion than its MPI or MPI+OpenMP hybrid counterparts. This is because creating processes and communications among processes in MPI are more expensive than creating threads in OpenMP.

The general suggestion is that for problem of a small size, use OpenMP with few threads, or simply use the sequential code. For problem of medium size, use OpenMP with a larger number of threads. For problem of large size, use MPI+OpenMP hybrid to achieve best performance, or choose either pure OpenMP or pure MPI to achieve better efficiency.

Specifically, for our problem:

For problem size of 100, the best configuration is to use OpenMP (4 threads). This configuration gives a speedup of around 2.1, with an efficiency of about 52.9%.

For problem size of 1000, the best configuration is still to use OpenMP (16 threads). This configuration gives a speedup of 13.4, with an efficiency of about 83.8%

For problem size of 10000, the best configuration is the hybrid scheme (8 processors each with 32 threads). This configuration gives a speedup of around 18.5, with a poor efficiency of 7.2%. To achieve best efficiency while keeping reasonable speedup, it is recommended to use MPI, which gives a comparable speedup of around 13.3, with an efficiency of 82.9%.

omp\_get\_wtime() is used in the OpenMP program for calculating the program execution time and the parallel portion execution time.

MPI\_Wtime() is used in both the MPI program and the hybrid program for calculating the program execution time and the parallel portion execution time.

Data

