Technische Universität Bergakademie Freiberg

Computational Materials Science
Fakultät IV

HPC Programming Project

Google Page Rank

Vishal Soni

Matriculation Number: 66158

1. Compilation Instructions

1.1. Using `mpic++` Manually

For reference, compile using:

```
mpic++ -o hpc_code.out  MPI_GooglePageRank.cpp
```

2. Job Submission with PBS

2.1. PBS Job Script (`script.pbs`)

Below is PBS script used to submit the MPI program to the cluster:

```
#!/bin/bash
#PBS -N MPI_hpc
#PBS -q teachingq
#PBS -l select=1:ncpus=60:mem=248gb:mpiprocs=60
#PBS -l walltime=00:05:00
#PBS -o pbs_out.txt
#PBS -e pbs_err.txt
module load openmpi/gcc/11.4.0/5.0.3
PBS_O_WORKDIR=$HOME/hpc_Project
cd $PBS_O_WORKDIR
MATRIX_SIZE=10000
mpic++ -o hpc_code.out MPI_GooglePageRank.cpp
mpirun ./hpc_code.out $MATRIX_SIZE
```

2.2. PBS Job Script Description

From the script shown above, the job name is given as `MPI_hpc`. The resources are allocated as 1 node with 60 CPU cores, 248 GB of memory, and 60 MPI processes to enable parallel execution. The job is submitted to the `teachingq` queue and is allowed to run for a maximum wall time of 5 minutes.
Necessary modules for compiling and running MPI programs, specifically `openmpi/gcc`, are loaded within the script.
The matrix size for computation is defined as a 10000×10000 matrix using an environment variable `MATRIX_SIZE`. The C++ program `MPI_GooglePageRank.cpp` is compiled using the MPI C++ compiler `mpic++`. This generates an executable file named `hpc_code.out`, which is used for parallel execution with the command:

```
mpirun ./hpc_code.out $MATRIX_SIZE
```

This runs the program with the specified matrix size in all allocated MPI processes.

2.3. Submitting the Job

Submit your PBS script with:

```
qsub script.pbs
```

2.4. Monitoring the Job

Check the status using:

```
qstat uniquejobId
```

2.5. Output

The largest eigenvalue is given as an output of the program and can be found in `pbs_out.txt`.

```
cat pbs_out.txt
```

```
Final Rayleigh Quotient (approx largest eigenvalue): 1.000000
Total computation time with 60 processors: 0.460119 seconds.
```

3. Testing the Implementation

In order to test the implementation of Google page rank a calculation of sample matrix of size 6X6 was used in `Test_GooglePageRank.cpp`. Below are the output results for sample matrix.

```
Final PageRank vector (r): [0.023810, 0.023810, 0.277778, 0.095238,
   0.214286, 0.365079]
Final Rayleigh Quotient (approx largest eigenvalue): 1.000000
Stochastic matrix P:
0.000000 0.333333 0.000000 0.166667 0.000000 0.000000
0.333333 0.000000 0.000000 0.166667 0.000000 0.000000
0.000000 0.333333 0.000000 0.166667 0.333333 0.500000
0.333333 0.000000 0.000000 0.166667 0.333333 0.000000
0.333333 0.333333 0.000000 0.166667 0.000000 0.500000
0.000000 0.000000 1.000000 0.166667 0.333333 0.000000

Verifying Summation condition:
Sum of Final PageRank vector (r): 1.000
```

4. Strong and Weak Scaling

4.1. Strong Scaling

For strong scaling analysis, a matrix of size 10000×10000 is used while varying the number of processors, as summarized in Table 1. In this approach, the problem size remains constant while the number of processing units is increased, allowing observation of how efficiently the task is handled as more computational resources are allocated.
To perform this analysis, the PBS job script is modified for each run by adjusting the values of `mpiprocs` and `ncpus` to reflect the desired number of processors. Execution time and the corresponding eigenvalue outputs for each processor configuration are recorded in the `pbs_out.txt` file.

Table 1: Strong Scaling Results for a 10000×10000 Matrix

Number of Processors	Execution Time (seconds)
1	18.043947
2	9.026738
4	4.752397
8	2.469111
12	1.910474
24	1.069715
32	0.834858
40	0.676530
48	0.572386
54	0.510381
60	0.462724

4.1.1 Observation from Strong Scaling Results

As seen in Table 1, the execution time decreases with an increasing number of processors, indicating good strong scaling performance. Significant reductions in runtime are observed up to 12 processors. Beyond that, the improvements continue but at a slower rate, suggesting the impact of communication overhead. Overall, the parallel implementation scales efficiently for the fixed matrix size.

4.2. Weak Scaling

Weak scaling is analyzed by increasing both the matrix size and the number of processors proportionally, as shown in Table 2.
To perform this analysis, the PBS job script (`script.pbs`) is modified in each run by incrementally adjusting the number of processors (`ncpus` and `mpiprocs`) along with the corresponding matrix size.

Table 2: Weak Scaling Results: Matrix Size vs. Number of Processors

Matrix Size	Total Elements	Number of Processors	Execution Time (seconds)
1291	1.666.681	1	0.302345
1825	3.330.625	2	0.320632
2581	6.661.561	4	0.330409
3650	13.322.500	8	0.383312
5162	26.646.244	16	0.453717
7300	53.290.000	32	0.451471
10000	100.000.000	60	0.466934

4.2.1 Observation from Weak Scaling Results

From Table 2, it is observed that as both the matrix size and the number of processors increase proportionally, the execution time remains relatively stable. This indicates that the parallel implementation handles increased workloads efficiently, maintaining good weak scaling performance. Minor variations in execution time may result from system load, but overall, the scalability is well-preserved.