German Razo

CSCI551

Matrix Multiplication ijk Forms with MPI

**Form ijk**

**Description of the data and task partitioning used, and why you chose it:**

Every Row in Matrix A was divided and sent to different processes, and rows are contiguous in memory, and this is because matrix A is stored as a 1 dimensional array. For the task partitioning I used scatterv to share rows equally, so all cores do evenly work even if the size of the matrix of number of cores is odd. This way every process calculates the resultant product and sends back the result to process 0.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** | Time(s) |  |  |
| **1** | 1.697315e+03 | 1.695651e+03 | 1.699666e+03 |
| **4** | 8.067305e+02 | 8.167302e+02 | 8.136958e+02 |
| **8** | 4.130692e+02 | 4.145923e+02 | 4.279018e+02 |
| **12** | 2.927824e+02 | 2.805531e+02 | 2.755762e+02 |
| **16** | 2.164538e+02 | 2.066878e+02 | 2.185750e+02 |
| **20** | 1.769553e+02 | 1.761812e+02 | 1.662818e+02 |

Speedup and efficiency

|  |  |  |  |
| --- | --- | --- | --- |
| Number of cores | Time(s) | Speedup | Efficiency |
| 1 | 1.695651e+03 |  |  |
| 4 | 8.067305e+02 |  |  |
| 8 | 4.130692e+02 |  |  |
| 12 | 2.755762e+02 |  |  |
| 16 | 2.066878e+02 |  |  |
| 20 | 1.662818e+02 |  |  |

**Form ikj**

**Description of the data and task partitioning used, and why you chose it:**

Every Row in Matrix A was divided and sent to different processes, and rows are contiguous in memory, and this is because matrix A is stored as a 1 dimensional array. For the task partitioning I used scatterv to share rows equally, so all cores do evenly work even if the size of the matrix of number of cores is odd. This way every process calculates the resultant product and sends back the result to process 0.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** | Time(s) |  |  |
| **1** | 8.644907e+02 | 8.643214e+02 | 8.652361e+02 |
| **4** | 3.922419e+02 | 3.917590e+02 | 3.923145e+02 |
| **8** | 1.983915e+02 | 1.981765e+02 | 1.986629e+02 |
| **12** | 1.336930e+02 | 1.337139e+02 | 1.340961e+02 |
| **16** | 1.014745e+02 | 1.015999e+02 | 1.016746e+02 |
| **20** | 8.223055e+01 | 8.222957e+01 | 8.215704e+01 |

Speedup and Efficiency

|  |  |  |  |
| --- | --- | --- | --- |
| Number of cores | Time(s) | Speedup | Efficiency |
| 1 | 8.643214e+02 |  |  |
| 4 | 3.917590e+02 |  |  |
| 8 | 1.981765e+02 |  |  |
| 12 | 1.336930e+02 |  |  |
| 16 | 1.014745e+02 |  |  |
| 20 | 8.215704e+01 |  |  |

**Form kij**

**Description of the data and task partitioning used, and why you chose it:**

Every Row in Matrix A was divided and sent to different processes, and rows are contiguous in memory, and this is because matrix A is stored as a 1 dimensional array. For the task partitioning I used scatterv to share rows equally, so all cores do evenly work even if the size of the matrix of number of cores is odd. This way every process calculates the resultant product and sends back the result to process 0.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** |  |  |  |
| **1** | 8.680398e+02 | 8.681970e+02 | 8.698750e+02 |
| **4** | 3.970423e+02 | 3.968905e+02 | 3.967150e+02 |
| **8** | 2.012338e+02 | 2.012615e+02 | 2.011976e+02 |
| **12** | 1.359288e+02 | 1.355242e+02 | 1.354414e+02 |
| **16** | 1.033170e+02 | 1.028657e+02 | 1.032467e+02 |
| **20** | 8.262071e+01 | 8.338978e+01 | 8.335589e+01 |

|  |  |  |  |
| --- | --- | --- | --- |
| Number of cores | Time(s) | Speedup | Efficiency |
| 1 | 8.680398e+02 |  |  |
| 4 | 3.968905e+02 |  |  |
| 8 | 2.011976e+02 |  |  |
| 12 | 1.354414e+02 |  |  |
| 16 | 1.028657e+02 |  |  |
| 20 | 8.262071e+01 |  |  |

Observations, analysis, & conclusions

Balanced