**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 is divided and sent to different processes. Scatterv is utilized to have a balanced partitioning between processes, so no process gets more work than other processes. The approached that I used to have a balanced partitioning was to take the reminder of the division of the size of the processes and the size of the matrix, and if the reminder is greater than 0 it means that the result is odd otherwise is even and all processes get even rows, so if the result is odd from the reminder I add one more row from the matrix to process 0 and the rest of the reminder I add it to the next processors. For example, if the size of the matrix is 7 and there are 4 processes, the reminder from 7 % 4 is 3, so then I divide 7 / 4 which is 1, so all processes get 1 row, then since reminder is 3 I add 1 row to process 0 and process 2 and 3 gets one more row, so processes 1, 2 and 3 get 2 rows and process 3 gets 1 row, for a total of 7 rows, so this way data and task partitioning is balanced. In addition to this partitioning a variable called **sum** is used to take advantage of temporal locality because it is reference many times in the loop instead of using [i \* n + j], which would be inefficient and cache would be wasted. I chose this partitioning because it gave good performance an accomplished great partitioning among cores. Array B is broadcasted to all processes.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** | Time(s) |  |  |
| **1** | 1697.315 | 1695.651 | 1699.666 |
| **4** | 806.7305 | 816.7302 | 813.6958 |
| **8** | 413.0692 | 414.5923 | 427.9018 |
| **12** | 292.7824 | 280.5531 | 275.5762 |
| **16** | 216.4538 | 206.6878 | 218.5750 |
| **20** | 176.9553 | 176.1812 | 166.2818 |

Speedup and efficiency

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of cores** | **Time(s)** | **Speedup** | **Efficiency** |
| **1** | 1695.651 | 1 | 1 |
| **4** | 806.7305 | 2.101 | .52525 |
| **8** | 413.0692 | 4.105 | .513125 |
| **12** | 275.5762 | 6.153 | .51275 |
| **16** | 206.6878 | 8.203 | .51268 |
| **20** | 166.2818 | 10.197 | .50985 |

**Form ikj**

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

Every row in matrix A is divided and sent to different processes. Scatterv is utilized to have a balanced partitioning between processes, so no process gets more work than other processes. The approached that I used to have a balanced partitioning was to take the reminder of the division of the size of the processes and the size of the matrix, and if the reminder is greater than 0 it means that the result is odd otherwise is even and all processes get even rows, so if the result is odd from the reminder I add one more row from the matrix to process 0 and the rest of the reminder I add it to the next processors. For example, if the size of the matrix is 7 and there are 4 processes, the reminder from 7 % 4 is 3, so then I divide 7 / 4 which is 1, so all processes get 1 row, then since reminder is 3 I add 1 row to process 0 and process 2 and 3 gets one more row, so processes 1, 2 and 3 get 2 rows and process 3 gets 1 row, for a total of 7 rows, so this way data and task partitioning is balanced. In addition to this partitioning a variable called mid is used to take advantage of temporal locality because it is reference many times in the loop instead of using [(i \* n) + k], which would be inefficient and cache would be wasted. I chose this partitioning because it gave good performance an accomplished great partitioning among cores Array B is broadcasted to all processes.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** | Time(s) |  |  |
| **1** | 864.4907 | 864.3214 | 865.2361 |
| **4** | 392.2419 | 391.7590 | 392.3145 |
| **8** | 198.3915 | 198.1765 | 198.6629 |
| **12** | 133.6930 | 133.7139 | 134.0961 |
| **16** | 101.4745 | 101.5999 | 101.6746 |
| **20** | 82.23055 | 82.22957 | 82.15704 |

**Speedup and Efficiency**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of cores | Time(s) | Speedup | Efficiency |
| **1** | 864.3214 | 1 | 1 |
| **4** | 391.7590 | 2.206 | .55156 |
| **8** | 198.1765 | 4.3614 | .54517 |
| **12** | 133.6930 | 6.4649 | .538746 |
| **16** | 101.4745 | 8.517621 | .532351 |
| **20** | 82.15704 | 10.5405 | .527025 |

**Form kij**

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

Every row in matrix A is divided and sent to different processes. Scatterv is utilized to have a balanced partitioning between processes, so no process gets more work than other processes. The approached that I used to have a balanced partitioning was to take the reminder of the division of the size of the processes and the size of the matrix, and if the reminder is greater than 0 it means that the result is odd otherwise is even and all processes get even rows, so if the result is odd from the reminder I add one more row from the matrix to process 0 and the rest of the reminder I add it to the next processors. For example, if the size of the matrix is 7 and there are 4 processes, the reminder from 7 % 4 is 3, so then I divide 7 / 4 which is 1, so all processes get 1 row, then since reminder is 3 I add 1 row to process 0 and process 2 and 3 gets one more row, so processes 1, 2 and 3 get 2 rows and process 3 gets 1 row, for a total of 7 rows, so this way data and task partitioning is balanced. In addition to this partitioning a variable called mid is used to take advantage of temporal locality because it is reference many times in the loop instead of using [i \* n + k], which would be inefficient and cache would be wasted. I chose this partitioning because it gave good performance an accomplished great partitioning among cores. Array B is broadcasted to all processes.

**Table of timings**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Runs** | | |
|  | **1** | **2** | **3** |
| **comm\_sz (number of cores)** |  |  |  |
| **1** | 868.0398 | 868.1970 | 869.8750 |
| **4** | 397.0423 | 396.8905 | 396.7150 |
| **8** | 201.2338 | 201.2615 | 201.197 |
| **12** | 135.9288 | 135.5242 | 135.4414 |
| **16** | 103.3170 | 102.8657 | 103.2467 |
| **20** | 82.62071 | 83.38978 | 83.35589 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of cores** | **Time(s)** | **Speedup** | **Efficiency** |
| **1** | 868.0398 | 1 | 1 |
| **4** | 396.8905 | 2.18710 | .546775 |
| **8** | 201.197 | 4.314377 | .539292 |
| **12** | 135.4414 | 6.40896 | .5340807 |
| **16** | 102.8657 | 8.4385737 | .5274108 |
| **20** | 82.62071 | 10.50632 | .52531611 |

**Observations, analysis, & conclusions**

Thinking about how I would partition the task and data to every process was a difficult part and I put a lot of thought into to it, so every process gets balanced amount of work. I used the same partitioning for all the ijk forms, but of course the multiplication is different for all the forms, the only thing that have in common is the petitioned of work as I mentioned. Another important part of my code was to minimize the timing and I accomplished this by utilizing variables that would be referenced many times, for example as I mentioned instead of using [i \* n + k] many times I used a variable instead and this improved performance, so for the form **kij** by utilizing a variable **mid,** time for 60 cores went from 100 seconds to 82 seconds, which was a huge performance, so this was a good improvement in my program. As observed in the graphs speedup and efficiency resulted in good performance, cores from 1 to 60 maintain an efficiency of 50% and speedup was half the number of processors, which is good performance. Also, the graphs showed that speedup increases with more cores and efficiency decreases with less cores.