HPCSE II - Exercise 6

Anian Ruoss

May 30, 2019

Task 1

We parallelize the heat2D model for the single-grid case. For ease of communication with MPI we change the gridLevel struct to store the grids with double $*$ instead of double $**$. First, we establish the problem geometry with MPI_Dims_create(), MPI_Cart_create(), and MPI_Cart_shift(). Since the grid size is not a multiple of two we have to pay attention to the boundary cases. Every rank allocates its own subgrids for U, Un, Res, and f. The root rank then initializes the problem on the entire grid and distributes the initial values to each rank. Since this step is only performed once we do not employ custom data types even though this incurs some memory overhead due to data duplication.

a)

After every Jacobi iteration we need to exchange the boundaries between neighboring ranks. Since we cannot pass the grid communicator to applyJacobi() (as this method is declared in heat2d_mpi.hpp where MPI is not initialized yet), we move the loop over the downRelaxations out of applyJacobi(). Next, we define data types for the contiguous and non-contiguous boundaries. Finally, we only need to adapt the loop boundaries to the subgrid sizes.

b)

The calculateResidual() method is trivially parallelizable by adapting the loop boundaries to the subgrid sizes. For calculateL2Norm() we need to be careful not to loop over the ghost cells (we make use of the fact that the

Listing 1: Output from running the baseline sequential model on one full node.

```
Time (s)      Grid0 Total
--------------|---------|---------
Smoothing | 570.226 | 570.226
Residual | 204.614 | 204.614
Restriction | 0.000 | 0.000
Prolongation | 0.000 | 0.000
L2Norm | 219.811 | 219.811
--------------|---------|---------
Total | 994.652 | 994.652
--------------|---------|---------

Running Time : 994.954s
L2Norm: 70812.9164
```

Listing 2: Output from running the parallelized model on one full node.

```
Time (s)     Grid0 Total
-------------|---------|---------
Smoothing | 18.228 | 18.228
Residual | 6.370 | 6.370
Restriction | 0.000 | 0.000
Prolongation | 0.000 | 0.000
L2Norm | 9.528 | 9.528
-------------|---------|---------
Total | 34.126 | 34.126
-------------|---------|---------

Running Time : 35.273s
L2Norm: 70812.9164
```

Dirichlet boundary condition is zero). After computing the sum of squared residuals for every subgrid we reduce the sum with MPI before computing the L_2-norm.

Evaluation

We run the baseline sequential model and compare it with the execution times from running the parallelized model on one, two, and four full nodes on Euler. We display the running times in **????????**.

It can be observed that all runs compute the same final L_2-norm value. We display the strong scaling speed-ups and efficiencies in **??**. Both applyJacobi() and calculateResidual() achieve speedups that exceed 100%. This is due to

Listing 3: Output from running the parallelized model on two full nodes.

```
Time (s)      Grid0 Total
--------------|---------|---------
Smoothing | 8.896 | 8.896
Residual | 3.271 | 3.271
Restriction | 0.000 | 0.000
Prolongation | 0.000 | 0.000
L2Norm | 7.684 | 7.684
--------------|---------|---------
Total | 19.851 | 19.851
--------------|---------|---------

Running Time : 19.926s
L2Norm: 70812.9164
```

Listing 4: Output from running the parallelized model on two full nodes.

```
Time (s)      Grid0 Total
--------------|---------|---------
Smoothing | 3.993 | 3.993
Residual | 1.598 | 1.598
Restriction | 0.000 | 0.000
Prolongation | 0.000 | 0.000
L2Norm | 6.009 | 6.009
--------------|---------|---------
Total | 11.600 | 11.600
--------------|---------|---------

Running Time : 13.509s
L2Norm: 70812.9164
```

Table 1: Strong scaling analysis for the parallelized heat2D model.

	Nodes	Speed-up	Efficiency (%)
Smoothing	1	31.3	130
	2	64.1	134
	4	142.8	149
Residual	1	32.1	134
	2	62.6	130
	4	128.0	133
L_2-norm	1	23.1	96
	2	28.6	60
	4	36.6	38
Total	1	28.2	118
	2	49.9	104
	4	73.7	77

the fact that we do not include the communication time in the measured kernel time for simplicity of the implementation. Hence, it is more meaningful to analyze the speed-up and efficiency for the total running time. We observe that for one and two nodes we achieve perfect efficiency[1], whereas we are probably limited by an increased communication overhead when running on four full nodes. Lastly, we note that the sharp efficiency decline for calculateL2Norm() is due to the expensive collective MPI_Allreduce() operation.

Task 2

a)

We implement the first hybrid approach with MPI and OpenMP by adding #pragma omp parallel for collapse(3) in front of the jacobi step loop. This does not induce a race condition since all iterations are independent from each other.

[1] We believe that the efficiency is slightly above 100% as we avoid the loop overhead by commenting out the prolongation- and restriction-loops since we only consider the single-grid case.

Table 2: Comparison of intra-node and network communication effects.

Nodes	MPI Ranks / Node	OpenMP Threads / MPI Rank	Compute	MPI_Waitall	Total Time
1	24	1	4.53	1.04	5.59
1	4	6	9.97	4.46	14.43
2	2	6	9.53	13.39	23.00
4	1	6	9.16	67.85	77.21

Evaluation

To avoid inconsistent L_2-norm values when dividing the grid by 12 or 24 MPI ranks, we increase the grid size to 768. We use the following command to run hybrid jobs on Euler with N nodes, M MPI ranks and T OpenMP threads:

```
bsub -R fullnode -n 24*N
"unset LSB_AFFINITY_HOSTFILE;
OMP_NUM_THREADS=T mpirun -n M --map-by node:PE=T --report-bindings ./hybrid"
```

We first try to gauge the performance loss incurred by network communication. To this end, we limit the total number of processes to 24 and measure the running times when distributing these processes across MPI/OpenMP and multiple nodes. We display the running times in **??**.

The first row displays our baseline which uses 24 MPI ranks distributed evenly among both sockets of one node. Hence, our baseline relies on intra-node communication only. The second row displays the output from running our hybrid code with 4 MPI ranks with 6 OpenMP threads each, all running on the same node. The first two ranks are assigned to the first socket, the last two ranks to the second socket. We observe that — although the number of computed entries increases by a factor of 6 for every rank — the total compute time increases only by a factor of ≈ 2.2 meaning that we achieve an efficiency of $\approx 45\%$ when considering only OpenMP. Compared to the baseline, this hybrid model uses 6 times fewer ranks which means that the number of communications decreases from 92 to 8. However, every rank has to exchange ≈ 3.5 times more boundary entries and we observe that this incurs an increase in the overall MPI_Waitall time for the hybrid intra-node case. This effect becomes even stronger when we distribute the MPI ranks across multiple nodes. The third row displays the execution times for a run with 4 MPI ranks with 6 OpenMP threads each, of which 2 ranks are on a separate node. The last row displays the execution times for with 4 MPI ranks with 6 OpenMP threads each, where every rank is on a node of its own. For this case, where all communication is performed over the

Table 3: Runtime comparison of MPI and hybrid MPI/OpenMP models.

Nodes	MPI Ranks / Node	OpenMP Threads / MPI Rank	Compute	MPI_Waitall	Total Time
1	24	1	4.53	1.04	5.59
	2	12	10.46	7.89	18.35
2	24	1	1.73	5.20	7.05
	2	12	5.70	20.79	26.64
4	24	1	0.77	12.11	13.30
	2	12	3.11	11.96	15.20

network, we observe a substantial increase in MPI_Waitall time compared to the second row which relies on intra-node communication only.

Next, we try to measure whether using a hybrid model has a positive effect on the total running time. We run the hybrid model with 2 MPI ranks per node and 12 OpenMP threads per rank, making sure that each rank is assigned to its own socket. As a consequence we reduce the amount of network communication between MPI ranks. We compare our hybrid model against a pure MPI model on one, two, and four nodes and display the running times in **??**.

It can be observed that for one and two nodes the increased message sizes — due to the smaller number of MPI ranks — supersedes the reduced cost of less communication over the network. However, for four nodes the MPI_Waitall time is already smaller for the hybrid case and we would except the hybrid model to significantly outperform the MPI model on more nodes.