Exercise 3

Alex Mecklin, 907666

October 2025

1 OpenMP implementation

Loop level parallelism was introduced in the functions evolve_interior() and evolve_edges() in the core.c file. The interior points are updated in a nested loop over rows and columns. OpenMP was applied to these loops with the #pragma omp parallel for collapse(2) command, which allows the threads to share the work over both loop levels. For the edge points, OpenMP parallel loops were also added separately for each edge.

The MPI communication routines (exchange_init() and exchange_finalize()) were left sequential, because parallelizing them could cause race conditions and incorrect results, as they had MPI calls inside the functions.

CPUs per task –cpus-per-task=1 was also changed to –cpus-per-task=4 in the job script, so the program actually uses OpenMP parallelization as well.

2 Results

The heat equation solver was run on a large 4000 × 4000 grid with 1000 time steps to assess performance with and without OpenMP parallelization.

The iteration time and total runtime measured by time for both versions are summarized in Table 1.

Version	Iteration Time [s]	Total Runtime [s]
Original (no OpenMP)	4.480	7.048
OpenMP	4.522	7.108

Table 1: Comparison of solver runtime for the large grid with and without OpenMP.

2.1 Profiling

Profiling was performed on a smaller grid of size 400 × 400 with 100 time steps. Profiling on the original 4000 x 4000 grid with 1000 time steps was not performed because it caused significant slowdowns due to the tracing. Using the smaller grid allowed to collect meaningful trace data without overloading the system or causing job failures. Score-P was used to collect event traces and Scalasca was used to analyze the time spent in different regions of the code. SCOREP_TOTAL_MEMORY environment variable was also increased to 500M based on the profiler's recommendations.

Table 2: Profiling results with OpenMP.

Parameter	**Value**
Estimated aggregate size of event trace	2000 MB
Estimated requirements for largest trace buffer (max_buf)	257 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY)	265 MB

type	**max_buf [B]**	**time[s]**	**region**
ALL	269,311,798	15.50	ALL
USR	268,947,536	4.62	USR
OMP	278,000	4.87	OMP
MPI	75,598	5.85	MPI
COM	10,608	0.13	COM
SCOREP	56	0.04	SCOREP
USR	260,624,598	4.56	idx
USR	8,320,000	0.02	cmap
MPI	35,600	0.00	MPI_Irecv
MPI	35,600	0.01	MPI_Isend
OMP	34,800	0.00	!$omp parallel @core.c:68
OMP	34,800	0.00	!$omp parallel @core.c:104
OMP	34,800	0.00	!$omp parallel @core.c:120
OMP	34,800	0.00	!$omp parallel @core.c:136
OMP	34,800	0.00	!$omp parallel @core.c:152
OMP	10,400	4.67	!$omp for @core.c:68
OMP	10,400	0.04	!$omp implicit barrier @core.c:84
OMP	10,400	0.05	!$omp for @core.c:104
OMP	10,400	0.00	!$omp implicit barrier @core.c:118
OMP	10,400	0.05	!$omp for @core.c:120
OMP	10,400	0.00	!$omp implicit barrier @core.c:134
OMP	10,400	0.02	!$omp for @core.c:136
OMP	10,400	0.00	!$omp implicit barrier @core.c:150
OMP	10,400	0.02	!$omp for @core.c:152
OMP	10,400	0.00	!$omp implicit barrier @core.c:166
MPI	2,600	0.17	MPI_Waitall
COM	2,600	0.00	exchange_init
COM	2,600	0.01	evolve_interior
COM	2,600	0.00	exchange_finalize
COM	2,600	0.00	evolve_edges
USR	2,600	0.00	swap_fields

Table 3: Profiling results without OpenMP.

Parameter	**Value**
Estimated aggregate size of event trace	1997 MB
Estimated requirements for largest trace buffer (max_buf)	257 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY)	259 MB

type	**max_buf [B]**	**time[s]**	**region**
ALL	269,033,798	15.57	ALL
USR	268,952,736	8.46	USR
MPI	75,598	6.93	MPI
COM	5,408	0.13	COM
SCOREP	56	0.06	SCOREP
USR	260,624,598	4.08	idx
USR	8,320,000	0.02	cmap
MPI	35,600	0.00	MPI_Irecv
MPI	35,600	0.01	MPI_Isend
MPI	2,600	0.18	MPI_Waitall
COM	2,600	0.00	exchange_init
USR	2,600	4.20	evolve_interior
COM	2,600	0.00	exchange_finalize
USR	2,600	0.13	evolve_edges
USR	2,600	0.00	swap_fields
MPI	854	0.05	MPI_Recv
MPI	390	0.00	MPI_Cart_coords
MPI	122	0.28	MPI_Ssend
USR	104	0.00	malloc_2d
USR	104	0.00	free_2d
MPI	84	0.29	MPI_Cart_create
MPI	84	0.00	MPI_Finalize
MPI	84	5.91	MPI_Init
MPI	78	0.00	MPI_Cart_get
MPI	68	0.20	MPI_Barrier
SCOREP	56	0.06	heat_mpi
MPI	52	0.00	MPI_Cart_shift
MPI	52	0.00	MPI_Comm_size
COM	52	0.00	set_field_dimensions
COM	52	0.01	write_field
USR	52	0.03	save_png

2.2 Memory usage

The profiling tables show that the majority of memory is used by the USR and ALL regions, with maximum buffer sizes around 260 MB. Comparing the OpenMP and non-OpenMP versions, the memory requirements are almost identical: 265 MB with OpenMP and 259 MB without. This indicates that adding loop level parallelism with OpenMP did not reduce the memory footprint, as all threads share the same data arrays and no additional optimizations for memory usage were implemented.

2.3 Performance

Table 1 shows that adding OpenMP parallelization did not reduce the overall runtime on the large 4000 × 4000 grid. The iteration time and total runtime are very similar between the OpenMP and non OpenMP versions.

The profiling tables of the smaller 400 × 400 grid give further insight. Most of the computational work in the OpenMP version occurs in the !$omp for regions. These regions together account for roughly 31% of the total runtime, indicating that although parallel loops were applied, the remaining runtime is dominated by sequential MPI communication and other sequential code. This explains why the overall performance improvement is limited.

2.4 Scaling

2.4.1 Strong scaling

For each job when testing strong scaling, 4 CPUs per processor and a global grid size of 4000 x 4000 with 1000 time steps was used.

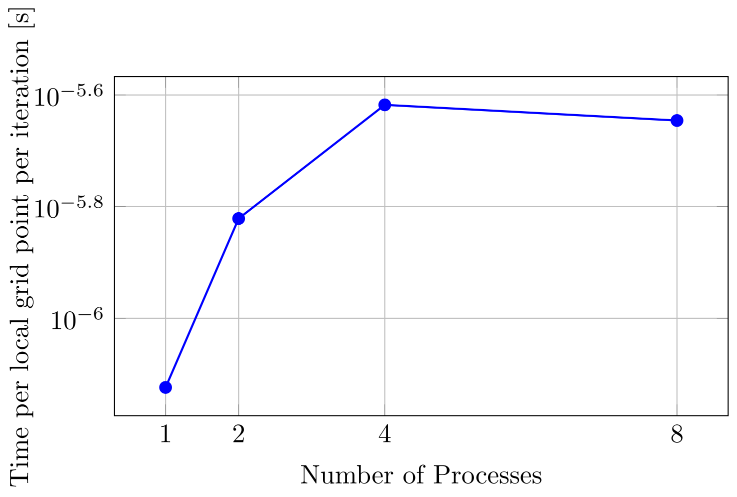

Figure 1: Strong scaling: time per local grid point per iteration as a function of number of processes.

As the number of processes increases from 1 to 8, the number of grid points per process decreases. The time per local grid point goes up when the number of processes increases from 1 to 4, because splitting the domain adds more communication and coordination work. From 4 to 8 processes, the time goes down a bit, showing that extra parallelism helps more than the overhead for these smaller subdomains.

2.4.2 Weak scaling

For weak scaling, the work per process should be constant, so the grid sizes in the Table 4 will be used to achieve that when increasing the amount of processes. Time step will be 1000 and CPUs per process will be 4 for all runs.

Table 4: Weak scaling setup: local and global grid sizes for different numbers of processes

Processes	**Local domain size**	**Total grid size**
1	4000 × 4000	4000 × 4000
2	4000 × 4000	8000 × 4000
4	4000 × 4000	8000 × 8000
8	4000 × 4000	16000 × 8000

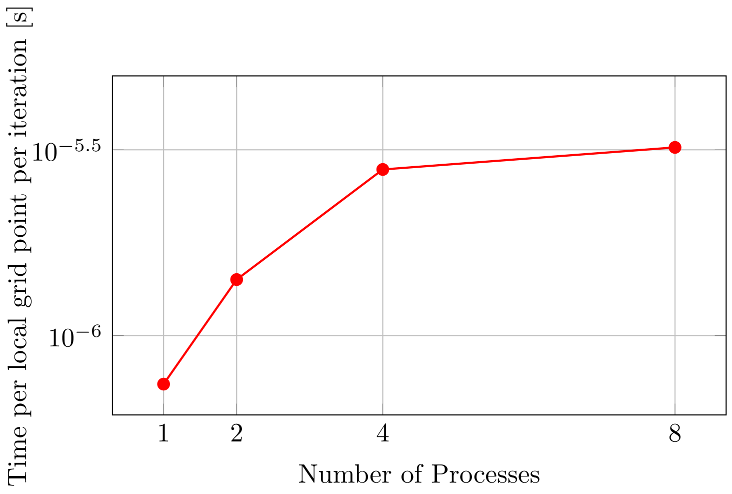

Figure 2: Weak scaling: time per local grid point per iteration as a function of number of processes. Each process keeps the local domain size fixed at 4000 × 4000.

The time per local grid point per iteration increases as the number of processes grows, even though each process handles the same amount of work. This shows that communication between processes and synchronization overheads become more significant as the total grid gets larger and more processes are added, which slows down the computation per point.