SOLUTIONS

Exercise 1: Detecting the focus of analysis (FOA)

a) Set MPIF77=scorep mpif77 in config/make.def. We compiled the application with the following parameters: `make bt-mz CLASS=W NPROCS=4`

b) The profile was generated with the following call:

```
OMP_NUM_THREADS=1 scan -s -e scorep_sum_W_1t mpiexec -np 4 bt-mz_W.4
```

c) The results are stored in the directory `scorep_sum_W_1t`. Open the generated profile with `square scorep_sum_W_1t`. Most of the execution time of the application is spent in the functions `x_solve, y_solve, z_solve` and their subfunctions with correspondingly 24.23%, 24.38% and 28.02% of the execution time.

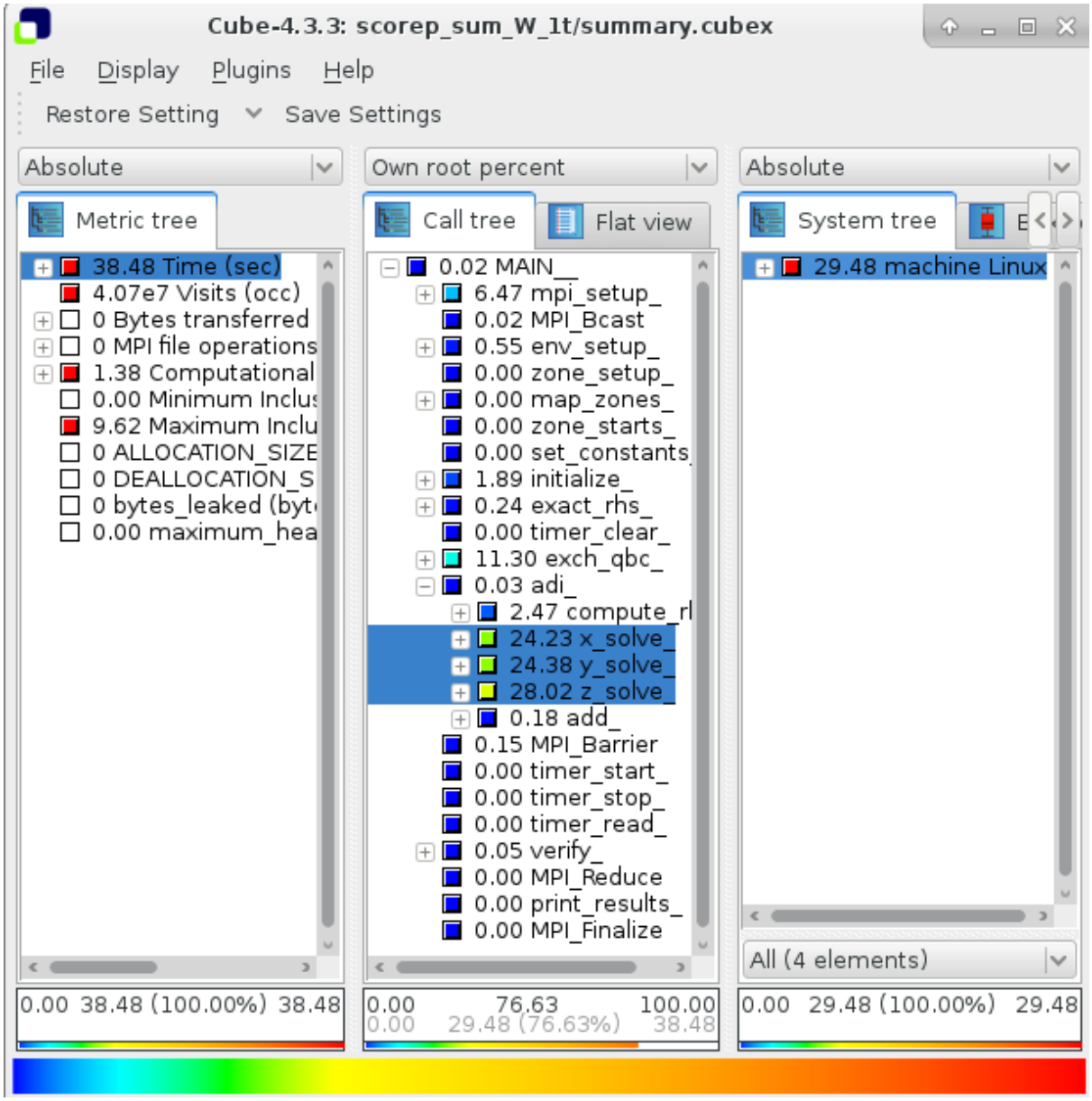

Figure 1: FOA of the application (Cube)

d) The application spends 82.2% of the execution time in computation and 17.7% in MPI operations.

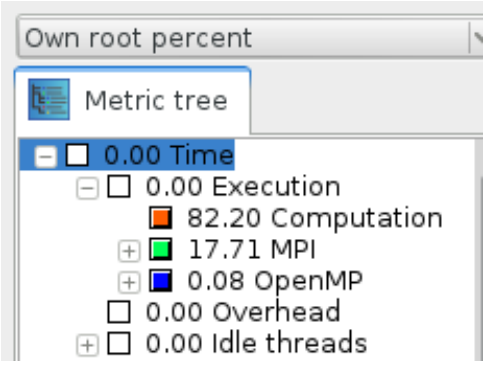

Figure 2: Time share of computation and MPI operations (Cube)

e) In the flat view after sorting by exclusive value you can see, that functions `binvcrhs`, `matmul_sub`, `matvev_sub` and do-loops in `x_solve`, `y_solve`, `z_solve` consume most exclusive execution time.

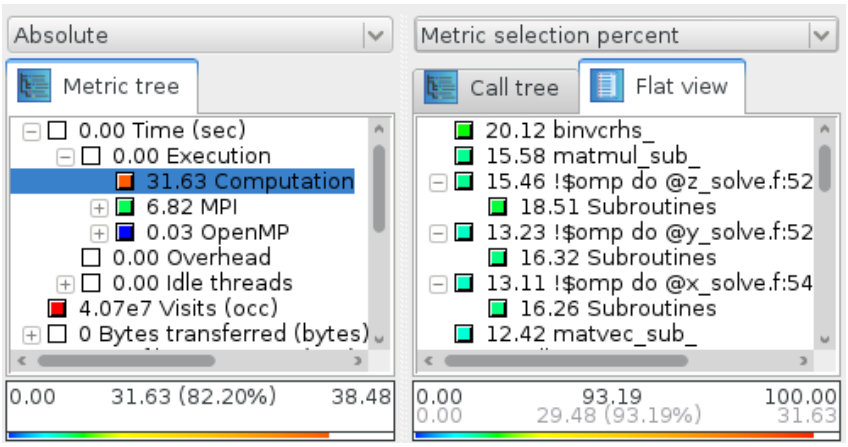

Figure 3: Hotspot functions of the application (Cube)

Exercise 2: MPI Communication and OpenMP Synchronization

a) 11.10% of the execution time of the application is spent for MPI communication (see Figure 4). Almost all the time is consumed in the MPI_Waitall operation (10.66%). In this time the unblocked data transfer run and the processes waited for each other at the end of the transfer until all processes became their data.

The communication load balance among processes is 83%. All processes spend almost the same time in MPI_Waitall. Observation of trace time line of the application can help better understand the behavior of processes.

Only one thread was running for this profile, that is why only few time is spent for OpenMP synchronization, which actually is overhead of the parallel region.

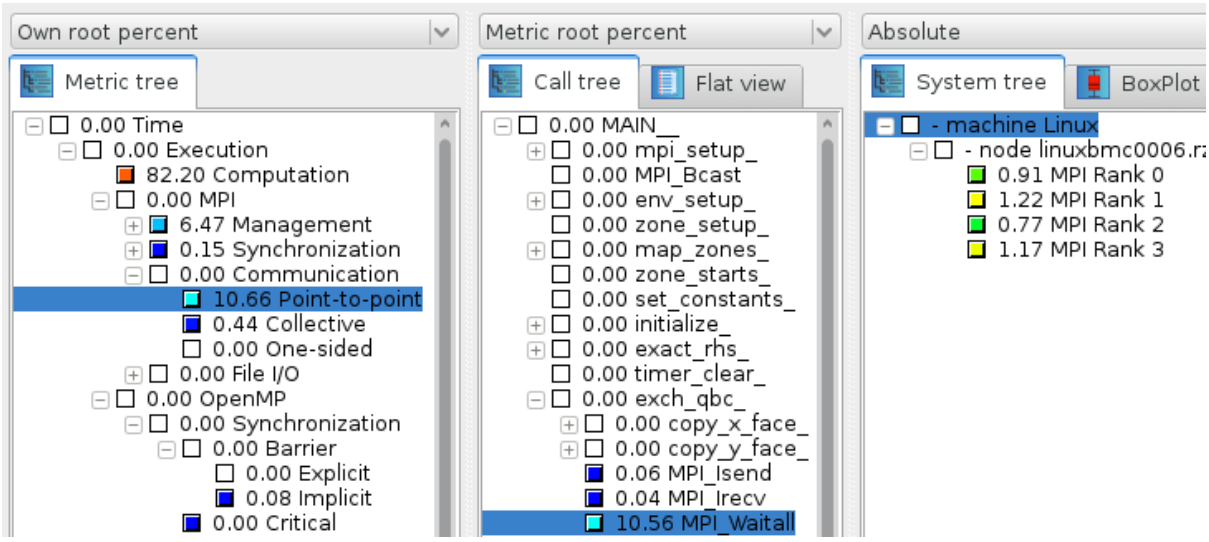

Figure 4: Time spent in communication (Cube)

Exercise 3: PAPI counters

b) To measure the number of executed instructions and spent cycles by processes a profile was generated with following call:

```
OMP_NUM_THREADS=1 SCOREP_METRIC_PAPI=PAPI_TOT_CYC,PAPI_TOT_INS scan -s -e
scorep_sum_W_papi_1t mpiexec -np 4 bt-mz_W.4
```

c) INS LB = sum INS / (#processes * max INS) = 1.19e11/(4*3.12e10) = 0.95 (see Figure 5). The instructions load balance is also very good. That means the processes get almost the same amount of work to compute.

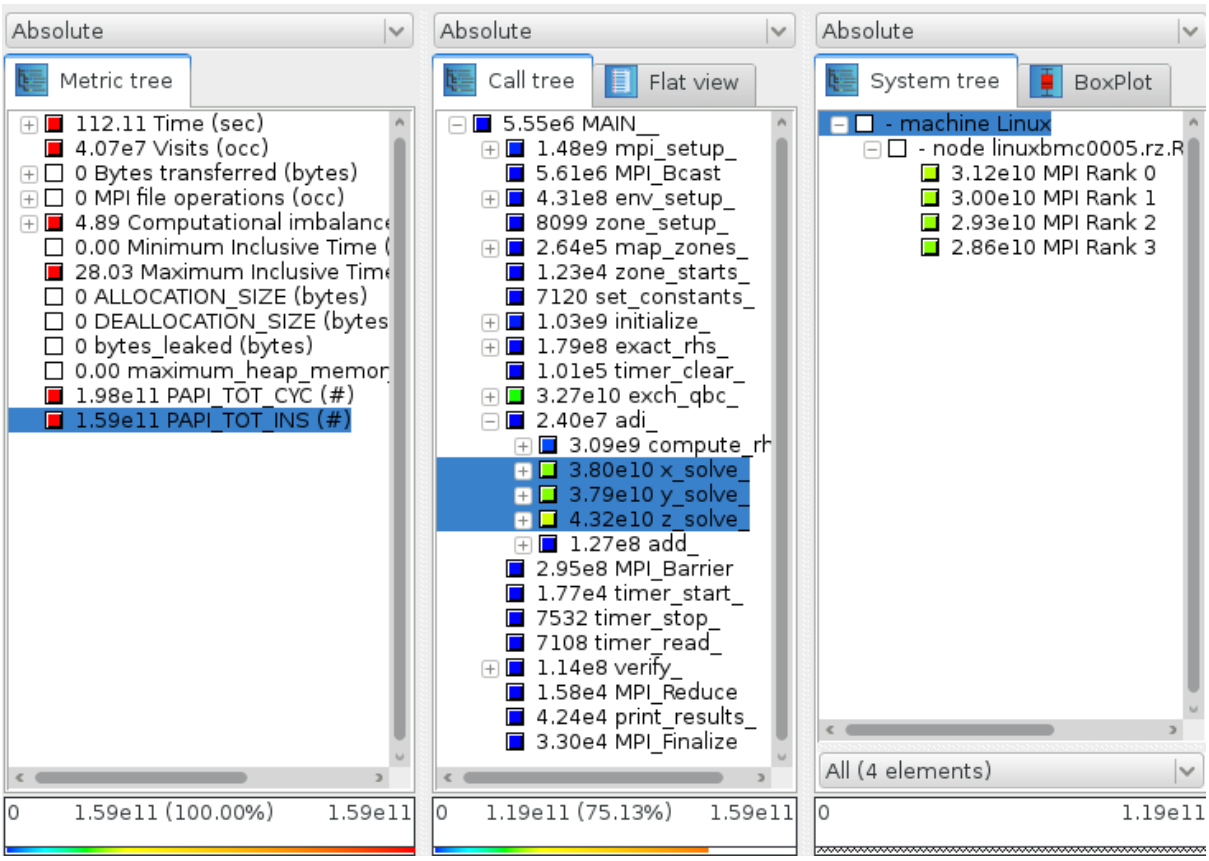

Figure 5: Distribution of the executed instructions among processes (Cube)

d) Each function performs bad with IPC lower one (see Table 1). A deeper analysis of these functions is needed to understand, why IPC is so low. The next step could be an analysis of the memory and vectorization behavior of the application.

e) A profile is generated with measuring of PAPI counters PAPI_L2_DCA and PAPI_L2_-DCM.

L2 cache miss ratio = L2_DCM/L2_DCA

From Table 1 you can see, that 1.3% of the second level cache accesses were missed in FOA. This is a small value and describes an optimal L2 cache usage.

Function	IPC	L2 cache miss ratio
FOA	0.73	1.3%
binvcrhs	0.81	0.36%
matmul_sub	0.74	0.28%
matvec_sub	0.68	0.97 %
do-loop in x_solve	0.72	1.1%
do-loop in y_solve	0.72	1.6%
do-loop in z_solve	0.71	2.4%

Table 1: Instructions per Cycle and cache miss ratio for the hotspot functions

Exercise 4: Functions filtering and creating of a Trace

a) Filter file can look so:

```
SCOREP_REGION_NAMES_BEGIN
EXCLUDE lhsinit_
binvrhs_
exact_solution_
copy*
SCOREP_REGION_NAMES_END
```

b) Generating a trace:

```
OMP_NUM_THREADS=1 SCOREP_TOTAL_MEMORY=249MB ESD_BUFFER_SIZE=500000
ELG_BUFFER_SIZE=20000000000 ELG_VT_MODE=1 SCOREP_ENABLE_TRACING=true
SCOREP_FILTERING_FILE=filter_file.txt scan -t -e scorep_trace_1t
mpiexec -np 4 bt-mz_W.4
```

Viewing the trace with *Vampir*:

```
vampir scorep_trace_t1/traces.otf2
```

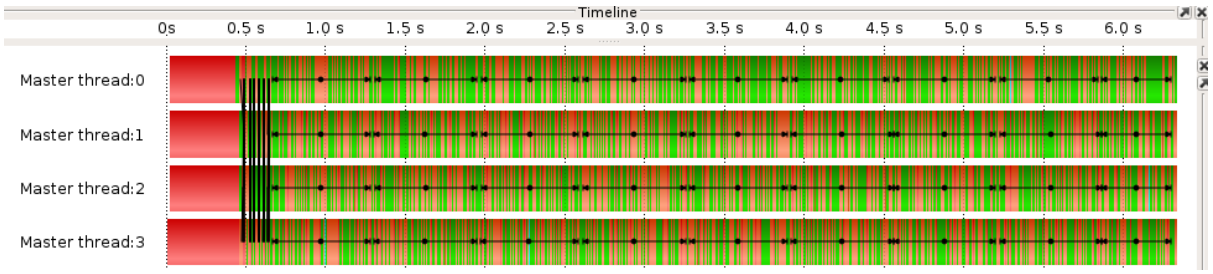

Figure 6: Timeline of the application (Vampir)

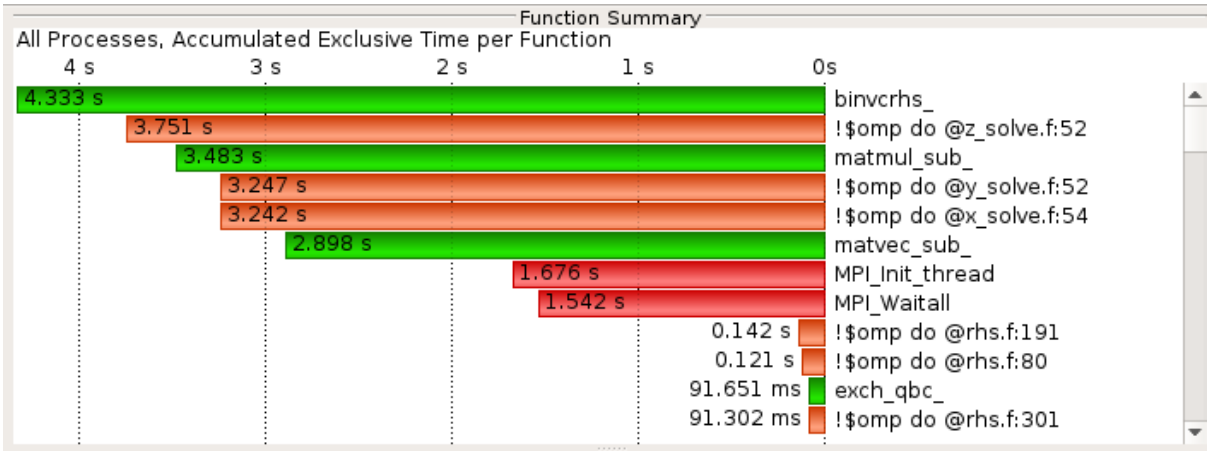

Figure 7: Function Summary of the application (Vampir)

Exercise 5: Efficiency metrics

a) The efficiency metrics were generated manually. Efficiency metrics for FOA are much better, because there is no MPI communications in the observed FOA.

```
SE = max comp time / max total time on ideal network

total time on ideal network = execution time - transfer time mpi

transfer time mpi = mpi time - wait time mpi - mpi io

wait time mpi = mpi latesender + mpi latereceiver + mpi earlyreduce +
+ mpi earlyscan + mpi latebroadcast + mpi wait nxn + mpi barrier wait +
+ mpi finalize wait

TE = max total time on ideal network / max total time

max total time = max execution time

ComE = SE + TE

LB = sum comp time / (max comp time * #processes)

PE = LB * ComE
```

	FOA	Full Application
Parallel Efficiency	92%	87%
Load Balance	93%	94%
Communication Efficiency	99.99%	93%
Serialization Efficiency	99.99%	99.5%
Transfer Efficiency	100%	93%

Table 2: Time efficiencies observed in the FOA