Hands-On Score-P/Scalasca/CUBE

Table of contents:

- Baseline measurement
    - Initial setup
    - Build benchmark
    - Run benchmark
- Instrumentation
- Filtering
- Explore profile with CUBE
- Scalasca trace analysis

Baseline measurement

In this part we are going to build and run a specific benchmark to identify how long it runs without any specific tools (also called as a reference/baseline run). A reference run provides a valuable point of comparison and context for performance analysis, enabling more informed decision-making and effective optimization efforts.

Initial setup

First of all let's login into CoolMUC-2 using ssh:

```
$ ssh -Y userid@lxlogin1.lrz.de
```

The **-Y** option is necessary to enable X11 forwarding. X11 forwarding is a SSH protocol that enables users to run graphical applications on a remote server and interact with them using their local display and I/O devices.

Now we need to create our own directory for the exercises:

```
$ mkdir -p $HOME/tw45
```

The **-p** prevents error messages if the specified directories already exists.

Then, we need to load required software, e.g. compiler, MPI, text editor:

```
$ module load intel intel-mpi/2019-intel nano
```

Build benchmark

Start by copying the tutorial sources to your working directory:

```
$ cd $HOME/tw45
$ tar zxvf /lrz/sys/courses/vihps/2024/material/NPB3.3-MZ-MPI.tar.gz -C .
$ cd $HOME/tw45/NPB3.3-MZ-MPI
```

For this tutorial we are going to use the NAS Parallel Benchmark suite (MPI+OpenMP version). It is available here, and includes three benchmarks written in Fortran77. You can configure the benchmark for various sizes and classes. This allows the benchmark to be used on a wide range of systems, from workstations to supercomputers.

> **INFO**
>
> NPB solves discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions. Each operates on a structured discretization mesh that is a logical cube. In realistic applications, however, a single such mesh is often not sufficient to describe a complex domain, and multiple meshes or zones are used to cover it.
>
> Multi-zone versions of NPB (NPB-MZ) are designed to exploit multiple levels of parallelism in applications and to test the effectiveness of multi-level and hybrid parallelization paradigms and tools. There are three types of benchmark problems derived from single-zone pseudo applications of NPB:
>
> - **Block Tri-diagonal (BT)** - uneven-sized zones within a problem class, increased number of zones as problem class grows
> - **Scalar Penta-diagonal (SP)** - even-sized zones within a problem class, increased number of zones as problem class grows
> - **Lower-Upper Symmetric Gauss-Seidel (LU)** - even-sized zones within a problem class, a fixed number of zones for all problem classes
>
> Benchmark Classes
>
> - Class **S**: small for quick test purposes
> - Class **W**: workstation size (a 90's workstation; now likely too small)
> - Classes **A**, **B**, **C**: standard test problems; ~4X size increase going from one class to the next
> - Classes **D**, **E**, **F**: large test problems; ~16X size increase from each of the previous classes
>
> MPI is used for communication across zones and OpenMP threads for computation inside zones. More technical details are provided in this paper.

Move into the NPB3.3-MZ-MPI root directory and check what is inside:

```
$ ls
bin/    common/  jobscript/  Makefile  README.install   SP-MZ/
BT-MZ/  config/  LU-MZ/      README    README.tutorial  sys/
```

Subdirectories `BT-MZ`, `LU-MZ` and `SP-MZ` contain source code for each benchmark, `config` and `common` include additional configuration and common code. The provided distribution has already been configured for the hands-on, such that it is ready to be build.

During this hands-on we will focus on `BT-MZ` exercise. It performs 200 time-steps on a regular 3-dimensional grid. It uses combination of MPI and OpenMP.

Type `make` for instructions

```
$ make
    ===========================================
   =      NAS PARALLEL BENCHMARKS 3.3        =
   =      MPI+OpenMP Multi-Zone Versions     =
   =      F77                                =
   ===========================================


   To make a NAS multi-zone benchmark type

         make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

   where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
         <class>          is "S", "W", "A" through "F"
         <nprocs>         is number of processes

   To make a set of benchmarks, create the file config/suite.def
   according to the instructions in config/suite.def.template and type

         make suite

 ***************************************************************
 * Custom build configuration is specified in config/make.def  *
 * Suggested tutorial exercise configuration for LiveDVD:      *
 *       make bt-mz CLASS=W NPROCS=4                           *
 ***************************************************************
```

To build application the following parameters need to be specified:

- The benchmark configuration benchmark name (bt-mz, lu-mz, sp-mz): `bt-mz`
- The number of MPI processes: `NPROCS=28`
- The benchmark class (S, W, A, B, C, D, E): `CLASS=C`

Alternatively, you can just use `make suite`.

```
$ make bt-mz CLASS=C NPROCS=28
    ===========================================
   =      NAS PARALLEL BENCHMARKS 3.3        =
   =      MPI+OpenMP Multi-Zone Versions     =
   =      F77                                =
   ===========================================

cd BT-MZ; make CLASS=C NPROCS=28 VERSION=
make[1]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-
MZ'
make[2]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-
MPI/sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/sys'
../sys/setparams bt-mz 28 C
make[2]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-
MZ'
mpif77 -c  -O3 -g -qopenmp       bt.f
mpif77 -c  -O3 -g -qopenmp       initialize.f
mpif77 -c  -O3 -g -qopenmp       exact_solution.f
mpif77 -c  -O3 -g -qopenmp       exact_rhs.f
mpif77 -c  -O3 -g -qopenmp       set_constants.f
mpif77 -c  -O3 -g -qopenmp       adi.f
mpif77 -c  -O3 -g -qopenmp       rhs.f
mpif77 -c  -O3 -g -qopenmp       zone_setup.f
mpif77 -c  -O3 -g -qopenmp       x_solve.f
mpif77 -c  -O3 -g -qopenmp       y_solve.f
mpif77 -c  -O3 -g -qopenmp       exch_qbc.f
mpif77 -c  -O3 -g -qopenmp       solve_subs.f
mpif77 -c  -O3 -g -qopenmp       z_solve.f
mpif77 -c  -O3 -g -qopenmp       add.f
mpif77 -c  -O3 -g -qopenmp       error.f
mpif77 -c  -O3 -g -qopenmp       verify.f
mpif77 -c  -O3 -g -qopenmp       mpi_setup.f
cd ../common; mpif77 -c  -O3 -g -qopenmp        print_results.f
```

```
cd ../common; mpif77 -c  -O3 -g -qopenmp        timers.f
mpif77 -O3 -g -qopenmp   -o ../bin/bt-mz_C.28 bt.o  initialize.o
exact_solution.o exact_rhs.o set_constants.o adi.o  rhs.o zone_setup.o
x_solve.o y_solve.o  exch_qbc.o solve_subs.o z_solve.o add.o error.o verify.o
mpi_setup.o ../common/print_results.o ../common/timers.o
make[2]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-
MZ'
Built executable ../bin/bt-mz_C.28
make[1]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-
MZ'
```

If compilation succeeds, you can find in the `bin` directory.

Run benchmark

Lets go to the `bin` directory, copy a prepared batch script and examine what it does:

```
$ cd bin
$ cp ../jobscript/coolmuc2/reference.sbatch .
$ nano reference.sbatch
```

Here is what you should see in your batch script:

```
#!/bin/bash
#SBATCH -o bt-mz.%j.out
#SBATCH -e bt-mz.%j.err
#SBATCH -J bt-mz
#SBATCH --clusters=cm2_tiny
#SBATCH --partition=cm2_tiny
#SBATCH --reservation=hhps1s24
#SBATCH --nodes=2
#SBATCH --ntasks=28
#SBATCH --ntasks-per-node=14
#SBATCH --get-user-env
#SBATCH --time=00:05:00

export OMP_NUM_THREADS=4

# Benchmark configuration (disable load balancing with threads)
export NPB_MZ_BLOAD=0
```

```
PROCS=28
CLASS=C

# Run the application
mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

To exit text editor you can use `Ctrl+X`

On CoolMUC-2 we are going to use:

- 2 standard compute nodes with 2x Intel Haswell 14-Core Processor each (28 cores / 56 threads)
- 56GB RAM per node
- 14 MPI ranks per node and 4 OpenMP threads per MPI rank

Now we are ready to submit our batch script:

```
$ sbatch reference.sbatch
```

> **INFO**
>
> To submit the job use `sbatch <script you want to submit>`.
>
> To check status of all your jobs use `squeue -M cm2_tiny --me`.
>
> To cancel specific job use `scancel -M cm2_tiny <jobid you want to cancel>`.

Once the job has finished you will see two files in your directory, one with standard output `bt-mz.<jobid>.out` and one with standard error output `bt-mz.<jobid>.err`. The former one should include all output provided by your application and the latter one only system specific output. Let's examine standard output file:

```
$ cat bt-mz.<jobid>.out
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

Number of zones:  16 x  16
Iterations: 200    dt:   0.000100
Number of active processes:    28

Use the default load factors with threads
```

```
Total number of threads:    112  (  4.0 threads/process)

Calculated speedup =    110.34

Time step    1
Time step   20
Time step   40
Time step   60
Time step   80
Time step  100
Time step  120
Time step  140
Time step  160
Time step  180
Time step  200
Verification being performed for class C
accuracy setting for epsilon =  0.1000000000000E-07
Comparison of RMS-norms of residual
          1 0.3457703287806E+07 0.3457703287806E+07 0.1092202750127E-12
          2 0.3213621375929E+06 0.3213621375929E+06 0.1320422658492E-12
          3 0.7002579656870E+06 0.7002579656870E+06 0.1496217033982E-13
          4 0.4517459627471E+06 0.4517459627471E+06 0.2280652586031E-13
          5 0.2818715870791E+07 0.2818715870791E+07 0.1486830094937E-14
Comparison of RMS-norms of solution error
          1 0.2059106993570E+06 0.2059106993570E+06 0.1540627820550E-12
          2 0.1680761129461E+05 0.1680761129461E+05 0.2132015705369E-12
          3 0.4080731640795E+05 0.4080731640795E+05 0.3084595553087E-13
          4 0.2836541076778E+05 0.2836541076778E+05 0.1026032398931E-12
          5 0.2136807610771E+06 0.2136807610771E+06 0.2335870996607E-12
Verification Successful


BT-MZ Benchmark Completed.
Class           =                        C
Size            =            480x  320x 28
Iterations      =                      200
Time in seconds =                    13.91
Total processes =                       28
Total threads   =                      112
Mop/s total     =                174439.35
Mop/s/thread    =                  1557.49
Operation type  =           floating point
Verification    =               SUCCESSFUL
```

```
Version          =                       3.3.1
Compile date     =                 04 Jun 2024
```

The most important metric in the output is "Time in seconds" which indicates how much time the application spent executing 200 iterations (pre and post. processing are excluded from the time measurement). Further, "Validation" is important as it indicates if the computation completed successfully (e.g. converged). Please write down the time value you received, as we are going to refer to its value in the next section.

INFO

For time measurements you can use `time` utility which is used to measure the execution time of a program or command. It provides information about how long a particular process took to execute, including user time, system time, and real time, i.e.

- **User time** is the time spent executing user-space instructions.
- **System time** is the time spent executing system calls.
- **Real time** is the actual time elapsed from start to finish, including all waiting and execution time.

It's a handy tool for performance analysis and optimization.

QUESTION

In this exercise we measured the basic performance metric, i.e. walltime. What else do you think can be used to measure the performance of the application in general and of the code you are working on?

Instrumentation

As a next step we are going to instrument our application, i.e. insert additional code into our program to collect performance data during its execution. Instrumentation can be done either manually by the programmer or automatically by tools like Score-P. The data collected includes information about user function calls, communication events, synchronization events, and more.

Score-P can automatically instrument the code by using e.g. compiler wrappers. This eliminates the need for manual modification of the source code and makes the process easier and less error-prone.

To use Score-P, we first need to make sure that all required software is available:

```
$ # Reload modules if needed
$ module load intel intel-mpi/2019-intel nano
$ # Load additional software being used in the following steps
$ module use /lrz/sys/courses/vihps/2024/modulefiles/
$ module load scorep/8.4-intel-intelmpi scalasca/2.6.1-intel-intelmpi
```

We loaded Scalasca trace tools at this stage as well to use convenience commands that allow to control execution measurement collection and analysis, and analysis report postprocessing. This is not necessary but highly recommended step to do.

Go to our work directory

```
$ cd $HOME/tw45/NPB3.3-MZ-MPI
```

Edit `config/make.def` to adjust build (see highlighted lines)

```
1  #-----------------------------------------------------------------------------
   -
2  #
3  #                 SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
4  #
5  #-----------------------------------------------------------------------------
   -
6
```

```
 7  #-----------------------------------------------------------------------
    -
 8  # Configured for generic MPI with GCC compiler
 9  #-----------------------------------------------------------------------
    -
10  #OPENMP = -fopenmp      # GCC compiler
11  OPENMP  = -qopenmp      # Intel compiler
12
13  #-----------------------------------------------------------------------
    -
14  # Parallel Fortran:
15  #
16  # The following must be defined:
17  #
18  # MPIF77     - Fortran compiler
19  # FFLAGS     - Fortran compilation arguments
20  # F_INC      - any -I arguments required for compiling MPI/Fortran
21  # FLINK      - Fortran linker
22  # FLINKFLAGS - Fortran linker arguments
23  # F_LIB      - any -L and -l arguments required for linking MPI/Fortran
24  #
25  # compilations are done with $(MPIF77) $(F_INC) $(FFLAGS) or
26  #                            $(MPIF77) $(FFLAGS)
27  # linking is done with       $(FLINK) $(F_LIB) $(FLINKFLAGS)
28  #-----------------------------------------------------------------------
    -
29
30  #-----------------------------------------------------------------------
    -
31  # The fortran compiler used for hybrid MPI programs
32  #-----------------------------------------------------------------------
    -
33  MPIF77 = mpif77
34
35  # Alternative variants to perform instrumentation
36  #MPIF77 = psc_instrument -t user,mpi,omp -s ${PROGRAM}.sir  mpif77
37  #MPIF77 = scalasca -instrument  mpif77
38  #MPIF77 = tau_f90.sh
39  #MPIF77 = vtf77 -vt:hyb -vt:f77  mpif77
40  MPIF77 = scorep --user  mpif77
41
42  # PREP is a generic macro for instrumentation preparation
43  #MPIF77 = $(PREP)  mpif77
44
45  # This links MPI fortran programs; usually the same as ${F77}
```

```
46 FLINK    = $(MPIF77)
47
48 #---------------------------------------------------------------------------
   -
49 # Global *compile time* flags for Fortran programs
50 #---------------------------------------------------------------------------
   -
51 FFLAGS  = -O3 -g $(OPENMP)
52
53 #---------------------------------------------------------------------------
   -
54 # These macros are passed to the compiler
55 #---------------------------------------------------------------------------
   -
56 F_INC =
57
58 #---------------------------------------------------------------------------
   -
59 # These macros are passed to the linker
60 #---------------------------------------------------------------------------
   -
61 F_LIB  =
62
63 #---------------------------------------------------------------------------
   -
64 # Global *link time* flags. Flags for increasing maximum executable
65 # size usually go here.
66 #---------------------------------------------------------------------------
   -
67 FLINKFLAGS = $(FFLAGS)
68
69
70 #---------------------------------------------------------------------------
   -
71 # Utilities C:
72 #
73
   41,0-1        58%
74 # Other allowed values are "randi8_safe", "randdp" and "randdpvec"
75 #---------------------------------------------------------------------------
   -
76 RAND   = randi8
77 # The following is highly reliable but may be slow:
78 # RAND   = randdp
```

In `config/make.def` we can set necessary flags for appropriate compilation, e.g. enabling OpenMP, optimisation flags, etc.

To enable instrumentation we added special wrapper `scorep` before actual compiler wrapper, e.g. `mpif77`. This will insert additional flags during compilation and add required libraries during linking phase.

WARNING

The `scorep` instrumenter must be used with the link command to ensure that all required Score-P measurement libraries are linked with the executable. However, not all object files need to be instrumented, thereby avoiding measurements and data collection for routines and OpenMP constructs defined in those files. Instrumenting files defining OpenMP parallel regions is essential, as Score-P has to track the creation of new threads.

Lets return to our root directory and clean-up:

```
$ cd $HOME/tw45/NPB3.3-MZ-MPI/
$ make clean
```

Next, we build the instrumented version of BT-MZ:

```
$ make bt-mz CLASS=C NPROCS=28
   ===========================================
   =      NAS PARALLEL BENCHMARKS 3.3        =
   =      MPI+OpenMP Multi-Zone Versions     =
   =      F77                                =
   ===========================================

cd BT-MZ; make CLASS=C NPROCS=28 VERSION=
make[1]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-
MZ'
make[2]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-
MPI/sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/sys'
../sys/setparams bt-mz 28 C
```

```
make[2]: Entering directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-MZ'
scorep --user  mpif77 -c  -O3 -g -qopenmp        bt.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        initialize.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        exact_solution.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        exact_rhs.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        set_constants.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        adi.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        rhs.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        zone_setup.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        x_solve.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        y_solve.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        exch_qbc.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        solve_subs.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        z_solve.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        add.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        error.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        verify.f
scorep --user  mpif77 -c  -O3 -g -qopenmp        mpi_setup.f
cd ../common; scorep --user  mpif77 -c  -O3 -g -qopenmp  print_results.f
cd ../common; scorep --user  mpif77 -c  -O3 -g -qopenmp  timers.f
scorep --user  mpif77 -O3 -g -qopenmp    -o ../bin.scorep/bt-mz_C.28 bt.o
initialize.o exact_solution.o exact_rhs.o set_constants.o adi.o  rhs.o
zone_setup.o x_solve.o y_solve.o  exch_qbc.o solve_subs.o z_solve.o add.o
error.o verify.o mpi_setup.o ../common/print_results.o ../common/timers.o
make[2]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-MZ'
Built executable ../bin.scorep/bt-mz_C.28
make[1]: Leaving directory '/dss/dsshome1/0C/hpckurs11/tw45/NPB3.3-MZ-MPI/BT-MZ'
```

As you might noticed now `scorep` stands before each compilation and linking command. This time executable was created in `bin.scorep` directory that allow us not to mess up with our baseline experiments.

Let's go to the directory where our new executable lies and copy batch script

```
$ cd bin.scorep
$ cp ../jobscript/coolmuc2/scorep.sbatch .
```

Let's examine what `scorep.sbatch` does by executing `nano scorep.batch`

```bash
#!/bin/bash
#SBATCH -o bt-mz.%j.out
#SBATCH -e bt-mz.%j.err
#SBATCH -J bt-mz
#SBATCH --clusters=cm2_tiny
#SBATCH --partition=cm2_tiny
#SBATCH --reservation=hhps1s24
#SBATCH --nodes=2
#SBATCH --ntasks=28
#SBATCH --ntasks-per-node=14
#SBATCH --get-user-env
#SBATCH --time=00:05:00

module use /lrz/sys/courses/vihps/2024/modulefiles/
module load scorep/8.4-intel-intelmpi
export OMP_NUM_THREADS=4

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum
#export SCOREP_FILTERING_FILE=../config/scorep.filt

# Benchmark configuration (disable load balancing with threads)
export NPB_MZ_BLOAD=0
PROCS=28
CLASS=C

# Run the application
mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

In highlighted line we set name of the directory where we store measurements. This is not required, but helps identifying the measurement later on.

> **INFO**
>
> Score-P measurements are configured via environment variables with the prefix `SCOREP_`. The full list of available variables and their description can be found by executing the following command `scorep-info config-vars --full`

Now we are ready to submit our batch script:

```
sbatch scorep.sbatch
```

Once your job complete check what is new in the execution directory

```
$ ls -l
bt-mz_C.28
bt-mz.657477.out
bt-mz.657477.err
scorep_bt-mz_sum
scorep.sbatch
```

What we see new there? `bt-mz.657477.err` includes stderr output, `bt-mz.657477.out` includes stdout output, and `scorep_bt-mz_sum` includes the measurement results collected by our instrumented application.

Let's examine what is inside measurement directory:

```
$ ls -1 scorep_bt-mz_sum/
MANIFEST.md
profile.cubex
scorep.cfg
```

The directory contains three files. `MANIFEST.md` includes the description of metadata, `profile.cubex` is an analysis report that was collected during the measurement, and `scorep.cfg` is a record of measurement configuration.

> **QUESTION**
>
> Open the stdout file and find the metric "Time in seconds". Compare it to our baseline measurement here. Has it increased or decreased? If so, by how much? What do you think was the reason for the change?

Filtering

Congratulations, we have made our first measurement with Score-P. But how good was the measurement? The measured execution gave the desired valid result, but the execution took a bit longer than expected! The instrumented run has a large increase in runtime compared to a baseline (around 46s versus 14s). Your runtime may vary slightly from our measurements. Even if we ignore the start and end of the measurement, it was probably prolonged by the instrumentation/measurement overhead.

To make sure you don't draw the wrong conclusions based on data that has been disturbed by significant overhead, it's often a good idea to optimise the measurement configuration before you do any more experiments. There are lots of ways you can do this, for example, by using runtime filtering, selective recording, or manual instrumentation to control the measurement.

However, in many cases, it's enough to filter a few frequently executed but otherwise unimportant user functions to reduce the measurement overhead to an acceptable level (based on experience, we consider 0-20% of runtime dilation as acceptable). The selection of those routines has to be done with care, though, as it affects the granularity of the measurement and too aggressive filtering might "blur" the location of important hotspots.

To understand where the overhead is coming from it is necessary to make scoring of the measurement. It can be done via the following command:

```
$ scorep-score scorep_bt-mz_sum/profile.cubex
```

As an output you will see the following:

```
Estimated aggregate size of event trace:                   160GB
Estimated requirements for largest trace buffer (max_buf): 6GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       6GB
(warning: The memory requirements cannot be satisfied by Score-P to avoid
 intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the
 maximum supported memory or reduce requirements using USR regions filters.)

flt     type    max_buf[B]        visits time[s] time[%] time/visit[us]
region
         ALL 6,282,548,755 6,586,867,463 5044.19   100.0           0.77  ALL
```

```
          USR 6,265,237,940 6,574,825,097 2257.25    44.7             0.34  USR
          OMP    17,537,080    10,975,232 2602.86    51.6           237.16  OMP
          MPI       985,204       339,446  180.12     3.6           530.62  MPI
          COM       738,530       727,660    3.93     0.1             5.41  COM
       SCOREP            41            28    0.03     0.0           934.60
SCOREP
```

As can be seen from the top of the score output, the estimated size for an event trace measurement without filtering applied is approximately 160GB, with the process-local maximum across all ranks being roughly 6GB.

The next section of the score output provides a table which shows how the trace memory requirements of a single process (column `max_buf`) as well as the overall number of visits and CPU allocation time are distributed among certain function groups. In current execution, the following groups are distinguished:

- `ALL`: All functions of the application.
- `MPI`: MPI API functions.
- `OMP`: OpenMP constructs and API functions.
- `COM`: User functions/regions that appear on a call path to an OpenMP construct, or an OpenMP or MPI API function. Useful to provide the context of MPI/OpenMP usage.
- `USR`: User functions/regions that do not appear on a call path to an OpenMP construct, or an OpenMP or MPI API function.
- `SCOREP`: This group aggregates activities within the measurement system.

> **INFO**
>
> There are more function groups available, e.g. `CUDA`,`OPENACC`,`MEMORY`,`IO`,`LIB`, etc. For more details consult with the documentation here.

As we can see from the scoring output, the `USR` group is making the biggest contribution to the trace memory requirements. To figure out which routines are causing the problem, we need to see a breakdown by function. To do this, we just need to run the following command:

```
$ scorep-score -r scorep_bt-mz_sum/profile.cubex
```

As an output you will see the following

```
Estimated aggregate size of event trace:                   160GB
Estimated requirements for largest trace buffer (max_buf): 6GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       6GB
(warning: The memory requirements cannot be satisfied by Score-P to avoid
 intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the
 maximum supported memory or reduce requirements using USR regions filters.)

flt     type    max_buf[B]        visits time[s] time[%] time/visit[us]
region
         ALL 6,282,548,755 6,586,867,463 5044.19   100.0           0.77  ALL
         USR 6,265,237,940 6,574,825,097 2257.25    44.7           0.34  USR
         OMP    17,537,080    10,975,232 2602.86    51.6         237.16  OMP
         MPI       985,204       339,446  180.12     3.6         530.62  MPI
         COM       738,530       727,660    3.93     0.1           5.41  COM
      SCOREP            41            28    0.03     0.0         934.60
SCOREP

         USR 2,014,873,848 2,110,313,472  913.03    18.1           0.43
binvcrhs_
         USR 2,014,873,848 2,110,313,472  553.30    11.0           0.26
matvec_sub_
         USR 2,014,873,848 2,110,313,472  718.20    14.2           0.34
matmul_sub_
         USR    88,951,746    87,475,200   31.80     0.6           0.36
lhsinit_
         USR    88,951,746    87,475,200   24.24     0.5           0.28
binvrhs_
         USR    64,926,576    68,892,672   16.66     0.3           0.24
exact_solution_
         OMP     1,398,960       411,648    0.18     0.0           0.43  !$omp
parallel @exch_qbc.f:204
         OMP     1,398,960       411,648    0.18     0.0           0.44  !$omp
parallel @exch_qbc.f:215
         OMP     1,398,960       411,648    0.19     0.0           0.45  !$omp
parallel @exch_qbc.f:244
         OMP     1,398,960       411,648    0.19     0.0           0.45  !$omp
parallel @exch_qbc.f:255
         OMP       702,960       206,848    0.93     0.0           4.49  !$omp
parallel @rhs.f:28
         OMP       699,480       205,824    0.12     0.0           0.57  !$omp
parallel @add.f:22
         OMP       699,480       205,824    0.21     0.0           1.01  !$omp
parallel @z_solve.f:43
         OMP       699,480       205,824    0.21     0.0           1.01  !$omp
```

```
parallel @x_solve.f:46
         OMP       699,480       205,824    0.21      0.0            1.02  !$omp
parallel @y_solve.f:43
         MPI       429,336       112,962    0.65      0.0            5.74
MPI_Irecv
         MPI       429,336       112,962    4.12      0.1           36.48
MPI_Isend
         OMP       418,080       411,648    2.28      0.0            5.53  !$omp
do @exch_qbc.f:204
         OMP       418,080       411,648    0.55      0.0            1.35  !$omp
implicit barrier @exch_qbc.f:213
         OMP       418,080       411,648    1.75      0.0            4.26  !$omp
do @exch_qbc.f:215
         OMP       418,080       411,648    0.47      0.0            1.14  !$omp
implicit barrier @exch_qbc.f:224
         OMP       418,080       411,648    2.81      0.1            6.82  !$omp
do @exch_qbc.f:244
         OMP       418,080       411,648    0.63      0.0            1.52  !$omp
implicit barrier @exch_qbc.f:253
         OMP       418,080       411,648    2.31      0.0            5.62  !$omp
do @exch_qbc.f:255
         OMP       418,080       411,648    0.52      0.0            1.27  !$omp
implicit barrier @exch_qbc.f:264
         OMP       210,080       206,848    0.44      0.0            2.15  !$omp
implicit barrier @rhs.f:439
         OMP       210,080       206,848   20.74      0.4          100.24  !$omp
do @rhs.f:37
         OMP       210,080       206,848   18.05      0.4           87.25  !$omp
do @rhs.f:62
         OMP       210,080       206,848    1.35      0.0            6.55  !$omp
implicit barrier @rhs.f:72
         OMP       210,080       206,848   31.36      0.6          151.61  !$omp
do @rhs.f:80
         OMP       210,080       206,848   29.51      0.6          142.68  !$omp
do @rhs.f:191
         OMP       210,080       206,848   23.38      0.5          113.02  !$omp
do @rhs.f:301
         OMP       210,080       206,848    5.61      0.1           27.13  !$omp
implicit barrier @rhs.f:353
         OMP       210,080       206,848    0.62      0.0            2.99  !$omp
do @rhs.f:359
         OMP       210,080       206,848    0.46      0.0            2.21  !$omp
do @rhs.f:372
         OMP       210,080       206,848   10.30      0.2           49.80  !$omp
do @rhs.f:384
```

```
         OMP      210,080       206,848    0.58      0.0             2.78  !$omp
do @rhs.f:400
         OMP      210,080       206,848    0.39      0.0             1.89  !$omp
do @rhs.f:413
         OMP      210,080       206,848    0.92      0.0             4.44  !$omp
implicit barrier @rhs.f:423
         OMP      210,080       206,848    3.17      0.1            15.33  !$omp
do @rhs.f:428
         OMP      209,040       205,824    8.36      0.2            40.62  !$omp
do @add.f:22
         OMP      209,040       205,824    0.89      0.0             4.35  !$omp
implicit barrier @add.f:33
         OMP      209,040       205,824  185.22      3.7           899.88  !$omp
implicit barrier @z_solve.f:428
         OMP      209,040       205,824  632.18     12.5          3071.45  !$omp
do @z_solve.f:52
         OMP      209,040       205,824  169.53      3.4           823.67  !$omp
implicit barrier @x_solve.f:407
         OMP      209,040       205,824  610.73     12.1          2967.25  !$omp
do @x_solve.f:54
         OMP      209,040       205,824  177.68      3.5           863.28  !$omp
implicit barrier @y_solve.f:406
         OMP      209,040       205,824  638.08     12.6          3100.13  !$omp
do @y_solve.f:52
         COM      209,040       205,824    0.81      0.0             3.91
copy_x_face_
         COM      209,040       205,824    0.75      0.0             3.63
copy_y_face_
         MPI      125,424       112,962   93.58      1.9           828.44
MPI_Waitall
         OMP       52,520        51,712    0.03      0.0             0.60  !$omp
master @rhs.f:74
         OMP       52,520        51,712    0.03      0.0             0.50  !$omp
master @rhs.f:183
         OMP       52,520        51,712    0.02      0.0             0.46  !$omp
master @rhs.f:293
         OMP       52,520        51,712    0.02      0.0             0.30  !$omp
master @rhs.f:424
         COM       52,520        51,712    0.31      0.0             6.09
compute_rhs_
         COM       52,260        51,456    0.22      0.0             4.27  adi_
         COM       52,260        51,456    0.36      0.0             6.94
x_solve_
         COM       52,260        51,456    0.35      0.0             6.73
y_solve_
```

```
        COM        52,260         51,456    0.35     0.0            6.88
z_solve_
        COM        52,260         51,456    0.29     0.0            5.62  add_
        USR        37,882         40,796    0.00     0.0            0.10
get_comm_index_
        OMP         6,960          2,048    0.01     0.0            2.70  !$omp
parallel @initialize.f:28
        COM         5,226          5,628    0.34     0.0           61.16
exch_qbc_
        OMP         5,200          5,120    0.00     0.0            0.80  !$omp
atomic @error.f:51
        OMP         5,200          5,120    0.00     0.0            0.28  !$omp
atomic @error.f:104
        OMP         3,480          1,024    0.01     0.0            5.34  !$omp
parallel @error.f:27
        OMP         3,480          1,024    0.00     0.0            1.90  !$omp
parallel @error.f:86
        OMP         3,480          1,024    0.00     0.0            1.96  !$omp
parallel @exact_rhs.f:21
        OMP         2,080          2,048    0.04     0.0           18.45  !$omp
implicit barrier @initialize.f:204
        OMP         2,080          2,048    0.19     0.0           94.36  !$omp
do @initialize.f:31
        OMP         2,080          2,048   11.81     0.2         5765.89  !$omp
do @initialize.f:50
        OMP         2,080          2,048    0.06     0.0           30.97  !$omp
do @initialize.f:100
        OMP         2,080          2,048    0.06     0.0           30.76  !$omp
do @initialize.f:119
        OMP         2,080          2,048    0.09     0.0           45.78  !$omp
do @initialize.f:137
        OMP         2,080          2,048    0.09     0.0           45.85  !$omp
do @initialize.f:156
        OMP         2,080          2,048    1.83     0.0          892.68  !$omp
implicit barrier @initialize.f:167
        OMP         2,080          2,048    0.07     0.0           33.67  !$omp
do @initialize.f:174
        OMP         2,080          2,048    0.07     0.0           33.21  !$omp
do @initialize.f:192
        OMP         1,040          1,024    0.15     0.0          143.32  !$omp
implicit barrier @error.f:54
        OMP         1,040          1,024    0.96     0.0          935.38  !$omp
do @error.f:33
        OMP         1,040          1,024    0.00     0.0            2.08  !$omp
implicit barrier @error.f:107
```

```
         OMP        1,040         1,024     0.02     0.0          17.27  !$omp
do @error.f:91
         OMP        1,040         1,024     0.00     0.0           2.48  !$omp
implicit barrier @exact_rhs.f:357
         OMP        1,040         1,024     0.21     0.0         201.46  !$omp
do @exact_rhs.f:31
         OMP        1,040         1,024     0.08     0.0          80.42  !$omp
implicit barrier @exact_rhs.f:41
         OMP        1,040         1,024     0.95     0.0         927.89  !$omp
do @exact_rhs.f:46
         OMP        1,040         1,024     1.00     0.0         974.77  !$omp
do @exact_rhs.f:147
         OMP        1,040         1,024     0.51     0.0         501.47  !$omp
implicit barrier @exact_rhs.f:242
         OMP        1,040         1,024     0.98     0.0         956.34  !$omp
do @exact_rhs.f:247
         OMP        1,040         1,024     0.27     0.0         264.42  !$omp
implicit barrier @exact_rhs.f:341
         OMP        1,040         1,024     0.02     0.0          20.34  !$omp
do @exact_rhs.f:346
         MPI          612           252     0.82     0.0        3266.14
MPI_Bcast
         USR          572           616     0.00     0.0           0.36
timer_clear_
         COM          520           512     0.02     0.0          47.11
initialize_
         COM          260           256     0.00     0.0          12.22
exact_rhs_
         COM          260           256     0.00     0.0           6.18
error_norm_
         COM          260           256     0.00     0.0           5.87
rhs_norm_
         MPI          204            84     0.44     0.0        5205.99
MPI_Reduce
         MPI          136            56     1.36     0.0       24257.44
MPI_Barrier
         MPI           52            56     0.00     0.0           1.83
MPI_Comm_rank
      SCOREP           41            28     0.03     0.0         934.60  bt-
mz_C.28
         MPI           26            28     0.00     0.0           4.69
MPI_Comm_size
         MPI           26            28    29.83     0.6     1065350.67
MPI_Comm_split
         MPI           26            28     0.01     0.0         352.54
```

```
MPI_Finalize
         MPI         26             28    49.31      1.0      1760964.30
MPI_Init_thread
         COM         26             28     0.11      0.0         3827.25
MAIN__
         COM         26             28     0.01      0.0          224.01
mpi_setup_
         COM         26             28     0.01      0.0          179.26
env_setup_
         USR         26             28     0.00      0.0           47.11
zone_setup_
         USR         26             28     0.01      0.0          262.41
map_zones_
         USR         26             28     0.00      0.0           32.67
zone_starts_
         USR         26             28     0.00      0.0            1.76
set_constants_
         USR         26             28     0.00      0.0          117.36
timer_start_
         USR         26             28     0.00      0.0            8.33
timer_stop_
         USR         26             28     0.00      0.0            1.11
timer_read_
         COM         26             28     0.01      0.0          263.89
verify_
         USR         26              1     0.00      0.0          523.75
print_results_
```

The detailed breakdown by region below the summary provides a classification according to these function groups (column type) for each region found in the summary report. Investigation of this part of the score report reveals that most of the trace data would be generated by about 6.8 billion calls to each of the three routines `binvcrhs`, `matmul_sub` and `matvec_sub` (these routines are highlighted), which are classified as `USR`. And although the percentage of time spent in these routines at first glance suggest that they are important, the average time per visit is below 270 nanoseconds (column `time/visit`). That is, the relative measurement overhead for these functions is substantial, and thus a significant amount of the reported time is very likely spent in the Score-P measurement system rather than in the application itself. Therefore, these routines constitute good candidates for being filtered (like they are good candidates for being inlined by the compiler). Additionally selecting the `lhsinit`, `binvrhs`, and `exact_solution` routines, which generates about 810MB of event data on a single rank with very little runtime impact.

Score-P allows users to exclude specific routines or files from being measured using a filter file. This file, written in a specific format, specifies what should be included or excluded. In our case, we define rules for certain functions between the keywords `SCOREP_REGION_NAMES_BEGIN` and `SCOREP_REGION_NAMES_END`, the keyword `EXCLUDE` indicating that functions must be excluded from the measurements. A typical Score-P filter file looks like this:

```
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs
    matmul_sub
    matvec_sub
    lhsinit
    binvrhs
    exact_solution
SCOREP_REGION_NAMES_END
```

We have prepared a filter file `scorep.filter`, which you can find here `NPB3.3-MZ-MPI/config/scorep.filt`. You may notice some differences from the example above, such as the use of asterisks (`*`) as bash wildcards, because some Fortran compilers handle `_` symbols in function names differently. We have also excluded timer functions from the measurement.

> **INFO**
>
> Just to let you know that the filter is safe to use. It doesn't prevent any of the listed routines from being executed. They are simply not recorded in the measurement, so they won't appear in the profile/trace explorer.

> **INFO**
>
> Please refer to the Score-P manual here for a detailed description of the filter file format, how to filter based on file names, define (and combine) blacklists and whitelists, and how to use wildcards for convenience.

The effectiveness of this filter can be examined by scoring the initial summary report again, this time specifying the filter file using the `-f` option of the `scorep-score -r -f ../config/scorep.filt scorep_bt-mz_sum/profile.cubex` command. This way a filter file can be incrementally developed, avoiding the need to conduct many measurements to step-by-step investigate the effect of filtering individual functions.

The output of the aforementioned command will look like this:

```
Estimated aggregate size of event trace:                   470MB
Estimated requirements for largest trace buffer (max_buf): 19MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       27MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=27MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type    max_buf[B]        visits time[s] time[%] time/visit[us]
region
 -       ALL 6,282,548,755 6,586,867,463 5044.19   100.0           0.77  ALL
 -       USR 6,265,237,940 6,574,825,097 2257.25    44.7           0.34  USR
 -       OMP    17,537,080    10,975,232 2602.86    51.6         237.16  OMP
 -       MPI       985,204       339,446  180.12     3.6         530.62  MPI
 -       COM       738,530       727,660    3.93     0.1           5.41  COM
 -    SCOREP            41            28    0.03     0.0         934.60
SCOREP

 *       ALL    19,298,841    12,083,275 2786.95    55.3         230.65  ALL-
FLT
 +       FLT 6,265,199,954 6,574,784,188 2257.24    44.7           0.34  FLT
 -       OMP    17,537,080    10,975,232 2602.86    51.6         237.16  OMP-
FLT
 -       MPI       985,204       339,446  180.12     3.6         530.62  MPI-
FLT
 *       COM       738,530       727,660    3.93     0.1           5.41  COM-
FLT
 *       USR        38,012        40,909    0.01     0.0           0.34  USR-
FLT
 -    SCOREP            41            28    0.03     0.0         934.60
SCOREP-FLT

 +       USR 2,014,873,848 2,110,313,472  913.03    18.1           0.43
binvcrhs_
 +       USR 2,014,873,848 2,110,313,472  553.30    11.0           0.26
matvec_sub_
 +       USR 2,014,873,848 2,110,313,472  718.20    14.2           0.34
matmul_sub_
 +       USR    88,951,746    87,475,200   31.80     0.6           0.36
lhsinit_
 +       USR    88,951,746    87,475,200   24.24     0.5           0.28
binvrhs_
 +       USR    64,926,576    68,892,672   16.66     0.3           0.24
exact_solution_
```

```
 -       OMP    1,398,960       411,648    0.18      0.0             0.43  !$omp
parallel @exch_qbc.f:204
 -       OMP    1,398,960       411,648    0.18      0.0             0.44  !$omp
parallel @exch_qbc.f:215
 -       OMP    1,398,960       411,648    0.19      0.0             0.45  !$omp
parallel @exch_qbc.f:244
 -       OMP    1,398,960       411,648    0.19      0.0             0.45  !$omp
parallel @exch_qbc.f:255
 -       OMP      702,960       206,848    0.93      0.0             4.49  !$omp
parallel @rhs.f:28
 -       OMP      699,480       205,824    0.12      0.0             0.57  !$omp
parallel @add.f:22
 -       OMP      699,480       205,824    0.21      0.0             1.01  !$omp
parallel @z_solve.f:43
 -       OMP      699,480       205,824    0.21      0.0             1.01  !$omp
parallel @x_solve.f:46
 -       OMP      699,480       205,824    0.21      0.0             1.02  !$omp
parallel @y_solve.f:43
 -       MPI      429,336       112,962    0.65      0.0             5.74
MPI_Irecv
 -       MPI      429,336       112,962    4.12      0.1            36.48
MPI_Isend
 -       OMP      418,080       411,648    2.28      0.0             5.53  !$omp
do @exch_qbc.f:204
 -       OMP      418,080       411,648    0.55      0.0             1.35  !$omp
implicit barrier @exch_qbc.f:213
 -       OMP      418,080       411,648    1.75      0.0             4.26  !$omp
do @exch_qbc.f:215
 -       OMP      418,080       411,648    0.47      0.0             1.14  !$omp
implicit barrier @exch_qbc.f:224
 -       OMP      418,080       411,648    2.81      0.1             6.82  !$omp
do @exch_qbc.f:244
 -       OMP      418,080       411,648    0.63      0.0             1.52  !$omp
implicit barrier @exch_qbc.f:253
 -       OMP      418,080       411,648    2.31      0.0             5.62  !$omp
do @exch_qbc.f:255
 -       OMP      418,080       411,648    0.52      0.0             1.27  !$omp
implicit barrier @exch_qbc.f:264
 -       OMP      210,080       206,848    0.44      0.0             2.15  !$omp
implicit barrier @rhs.f:439
 -       OMP      210,080       206,848   20.74      0.4           100.24  !$omp
do @rhs.f:37
 -       OMP      210,080       206,848   18.05      0.4            87.25  !$omp
do @rhs.f:62
 -       OMP      210,080       206,848    1.35      0.0             6.55  !$omp
```

```
implicit barrier @rhs.f:72
 -       OMP       210,080       206,848   31.36      0.6            151.61  !$omp
do @rhs.f:80
 -       OMP       210,080       206,848   29.51      0.6            142.68  !$omp
do @rhs.f:191
 -       OMP       210,080       206,848   23.38      0.5            113.02  !$omp
do @rhs.f:301
 -       OMP       210,080       206,848    5.61      0.1             27.13  !$omp
implicit barrier @rhs.f:353
 -       OMP       210,080       206,848    0.62      0.0              2.99  !$omp
do @rhs.f:359
 -       OMP       210,080       206,848    0.46      0.0              2.21  !$omp
do @rhs.f:372
 -       OMP       210,080       206,848   10.30      0.2             49.80  !$omp
do @rhs.f:384
 -       OMP       210,080       206,848    0.58      0.0              2.78  !$omp
do @rhs.f:400
 -       OMP       210,080       206,848    0.39      0.0              1.89  !$omp
do @rhs.f:413
 -       OMP       210,080       206,848    0.92      0.0              4.44  !$omp
implicit barrier @rhs.f:423
 -       OMP       210,080       206,848    3.17      0.1             15.33  !$omp
do @rhs.f:428
 -       OMP       209,040       205,824    8.36      0.2             40.62  !$omp
do @add.f:22
 -       OMP       209,040       205,824    0.89      0.0              4.35  !$omp
implicit barrier @add.f:33
 -       OMP       209,040       205,824  185.22      3.7            899.88  !$omp
implicit barrier @z_solve.f:428
 -       OMP       209,040       205,824  632.18     12.5           3071.45  !$omp
do @z_solve.f:52
 -       OMP       209,040       205,824  169.53      3.4            823.67  !$omp
implicit barrier @x_solve.f:407
 -       OMP       209,040       205,824  610.73     12.1           2967.25  !$omp
do @x_solve.f:54
 -       OMP       209,040       205,824  177.68      3.5            863.28  !$omp
implicit barrier @y_solve.f:406
 -       OMP       209,040       205,824  638.08     12.6           3100.13  !$omp
do @y_solve.f:52
 -       COM       209,040       205,824    0.81      0.0              3.91
copy_x_face_
 -       COM       209,040       205,824    0.75      0.0              3.63
copy_y_face_
 -       MPI       125,424       112,962   93.58      1.9            828.44
MPI_Waitall
```

```
 -       OMP       52,520        51,712    0.03     0.0            0.60  !$omp
master @rhs.f:74
 -       OMP       52,520        51,712    0.03     0.0            0.50  !$omp
master @rhs.f:183
 -       OMP       52,520        51,712    0.02     0.0            0.46  !$omp
master @rhs.f:293
 -       OMP       52,520        51,712    0.02     0.0            0.30  !$omp
master @rhs.f:424
 -       COM       52,520        51,712    0.31     0.0            6.09
compute_rhs_
 -       COM       52,260        51,456    0.22     0.0            4.27  adi_
 -       COM       52,260        51,456    0.36     0.0            6.94
x_solve_
 -       COM       52,260        51,456    0.35     0.0            6.73
y_solve_
 -       COM       52,260        51,456    0.35     0.0            6.88
z_solve_
 -       COM       52,260        51,456    0.29     0.0            5.62  add_
 -       USR       37,882        40,796    0.00     0.0            0.10
get_comm_index_
 -       OMP        6,960         2,048    0.01     0.0            2.70  !$omp
parallel @initialize.f:28
 -       COM        5,226         5,628    0.34     0.0           61.16
exch_qbc_
 -       OMP        5,200         5,120    0.00     0.0            0.80  !$omp
atomic @error.f:51
 -       OMP        5,200         5,120    0.00     0.0            0.28  !$omp
atomic @error.f:104
 -       OMP        3,480         1,024    0.01     0.0            5.34  !$omp
parallel @error.f:27
 -       OMP        3,480         1,024    0.00     0.0            1.90  !$omp
parallel @error.f:86
 -       OMP        3,480         1,024    0.00     0.0            1.96  !$omp
parallel @exact_rhs.f:21
 -       OMP        2,080         2,048    0.04     0.0           18.45  !$omp
implicit barrier @initialize.f:204
 -       OMP        2,080         2,048    0.19     0.0           94.36  !$omp
do @initialize.f:31
 -       OMP        2,080         2,048   11.81     0.2         5765.89  !$omp
do @initialize.f:50
 -       OMP        2,080         2,048    0.06     0.0           30.97  !$omp
do @initialize.f:100
 -       OMP        2,080         2,048    0.06     0.0           30.76  !$omp
do @initialize.f:119
 -       OMP        2,080         2,048    0.09     0.0           45.78  !$omp
```

```
do @initialize.f:137
 -       OMP       2,080         2,048    0.09     0.0           45.85  !$omp
do @initialize.f:156
 -       OMP       2,080         2,048    1.83     0.0          892.68  !$omp
implicit barrier @initialize.f:167
 -       OMP       2,080         2,048    0.07     0.0           33.67  !$omp
do @initialize.f:174
 -       OMP       2,080         2,048    0.07     0.0           33.21  !$omp
do @initialize.f:192
 -       OMP       1,040         1,024    0.15     0.0          143.32  !$omp
implicit barrier @error.f:54
 -       OMP       1,040         1,024    0.96     0.0          935.38  !$omp
do @error.f:33
 -       OMP       1,040         1,024    0.00     0.0            2.08  !$omp
implicit barrier @error.f:107
 -       OMP       1,040         1,024    0.02     0.0           17.27  !$omp
do @error.f:91
 -       OMP       1,040         1,024    0.00     0.0            2.48  !$omp
implicit barrier @exact_rhs.f:357
 -       OMP       1,040         1,024    0.21     0.0          201.46  !$omp
do @exact_rhs.f:31
 -       OMP       1,040         1,024    0.08     0.0           80.42  !$omp
implicit barrier @exact_rhs.f:41
 -       OMP       1,040         1,024    0.95     0.0          927.89  !$omp
do @exact_rhs.f:46
 -       OMP       1,040         1,024    1.00     0.0          974.77  !$omp
do @exact_rhs.f:147
 -       OMP       1,040         1,024    0.51     0.0          501.47  !$omp
implicit barrier @exact_rhs.f:242
 -       OMP       1,040         1,024    0.98     0.0          956.34  !$omp
do @exact_rhs.f:247
 -       OMP       1,040         1,024    0.27     0.0          264.42  !$omp
implicit barrier @exact_rhs.f:341
 -       OMP       1,040         1,024    0.02     0.0           20.34  !$omp
do @exact_rhs.f:346
 -       MPI         612           252    0.82     0.0         3266.14
MPI_Bcast
 +       USR         572           616    0.00     0.0            0.36
timer_clear_
 -       COM         520           512    0.02     0.0           47.11
initialize_
 -       COM         260           256    0.00     0.0           12.22
exact_rhs_
 -       COM         260           256    0.00     0.0            6.18
error_norm_
```

```
 -       COM         260           256    0.00      0.0          5.87
rhs_norm_
 -       MPI         204            84    0.44      0.0       5205.99
MPI_Reduce
 -       MPI         136            56    1.36      0.0      24257.44
MPI_Barrier
 -       MPI          52            56    0.00      0.0          1.83
MPI_Comm_rank
 -    SCOREP          41            28    0.03      0.0        934.60  bt-
mz_C.28
 -       MPI          26            28    0.00      0.0          4.69
MPI_Comm_size
 -       MPI          26            28   29.83      0.6    1065350.67
MPI_Comm_split
 -       MPI          26            28    0.01      0.0        352.54
MPI_Finalize
 -       MPI          26            28   49.31      1.0    1760964.30
MPI_Init_thread
 -       COM          26            28    0.11      0.0       3827.25
MAIN__
 -       COM          26            28    0.01      0.0        224.01
mpi_setup_
 -       COM          26            28    0.01      0.0        179.26
env_setup_
 -       USR          26            28    0.00      0.0         47.11
zone_setup_
 -       USR          26            28    0.01      0.0        262.41
map_zones_
 -       USR          26            28    0.00      0.0         32.67
zone_starts_
 -       USR          26            28    0.00      0.0          1.76
set_constants_
 +       USR          26            28    0.00      0.0        117.36
timer_start_
 +       USR          26            28    0.00      0.0          8.33
timer_stop_
 +       USR          26            28    0.00      0.0          1.11
timer_read_
 -       COM          26            28    0.01      0.0        263.89
verify_
 -       USR          26             1    0.00      0.0        523.75
print_results_
```

Below the (original) function group summary, the score report now also includes a second summary with the filter applied. Here, an additional group `FLT` is added, which subsumes all filtered regions. Moreover, the column `flt` indicates whether a region/function group is filtered (`+`), not filtered (`-`), or possibly partially filtered (`*`, only used for function groups).

As expected, the estimate for the aggregate event trace size drops down to 470MB, and the process-local maximum across all ranks is reduced to 19MB. Since the Score-P measurement system also creates a number of internal data structures (e.g., to track MPI requests and communicators), the suggested setting for the `SCOREP_TOTAL_MEMORY` environment variable to adjust the maximum amount of memory used by the Score-P memory management is 27MB when tracing is configured.

:::

With the `-g` option, `scorep-score` can create an initial filter file in Score-P format. See more details here.

:::

Let's modify our batch script `score.sbatch` to enable filtering (see highlighted lines):

```
1  #!/bin/bash
2  #SBATCH -o bt-mz.%j.out
3  #SBATCH -e bt-mz.%j.err
4  #SBATCH -J bt-mz
5  #SBATCH --clusters=cm2_tiny
6  #SBATCH --partition=cm2_tiny
7  #SBATCH --reservation=hhps1s24
8  #SBATCH --nodes=2
9  #SBATCH --ntasks=28
10 #SBATCH --ntasks-per-node=14
11 #SBATCH --get-user-env
12 #SBATCH --time=00:05:00
13
14 module use /lrz/sys/courses/vihps/2024/modulefiles/
15 module load scorep/8.4-intel-intelmpi
16 export OMP_NUM_THREADS=4
17
18 # Score-P measurement configuration
19 export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum_filt
20 export SCOREP_FILTERING_FILE=../config/scorep.filt
21
```

```
22 # Benchmark configuration (disable load balancing with threads)
23 export NPB_MZ_BLOAD=0
24 PROCS=28
25 CLASS=C
26
27 # Run the application
28 mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

In first highlighted line we added suffix `_filt` to create measurement directory with a different name. In the second one we provided name of the filter file which will be used during the measurement.

INFO

If you do not specify `SCOREP_EXPERIMENT_DIRECTORY` variable, the experiment directory is named in the format `scorep-YYYYMMDD_HHMM_XXXXXXXX`, where `YYYYMMDD` and `HHMM` represent the date and time, followed by random numbers.

If a directory with the specified name already exists, it will be renamed with a date suffix by default. To prevent this and abort the measurement if the directory exists, set `SCOREP_OVERWRITE_EXPERIMENT_DIRECTORY` to `false`. This setting is effective only if `SCOREP_EXPERIMENT_DIRECTORY` is set.

Now we are ready to submit our batch script with enabled filtering

```
$ sbatch scorep.sbatch
```

QUESTION

Open the freshly generated stdout file and find the metric "Time in seconds". Compare it to our baseline measurement here and our original instrumented run here. Has it increased or decreased? If so, by how much? Which routines in your opinion are safe to filter?

Explore profile with CUBE

Congratulations, now we collected our first measurements with acceptable runtime dilation. This new measurement should accurately represent the real runtime behavior of the BT-MZ application, and can now be postprocessed and interactively explored using the Cube browser. These two steps can be conveniently initiated using the following command:

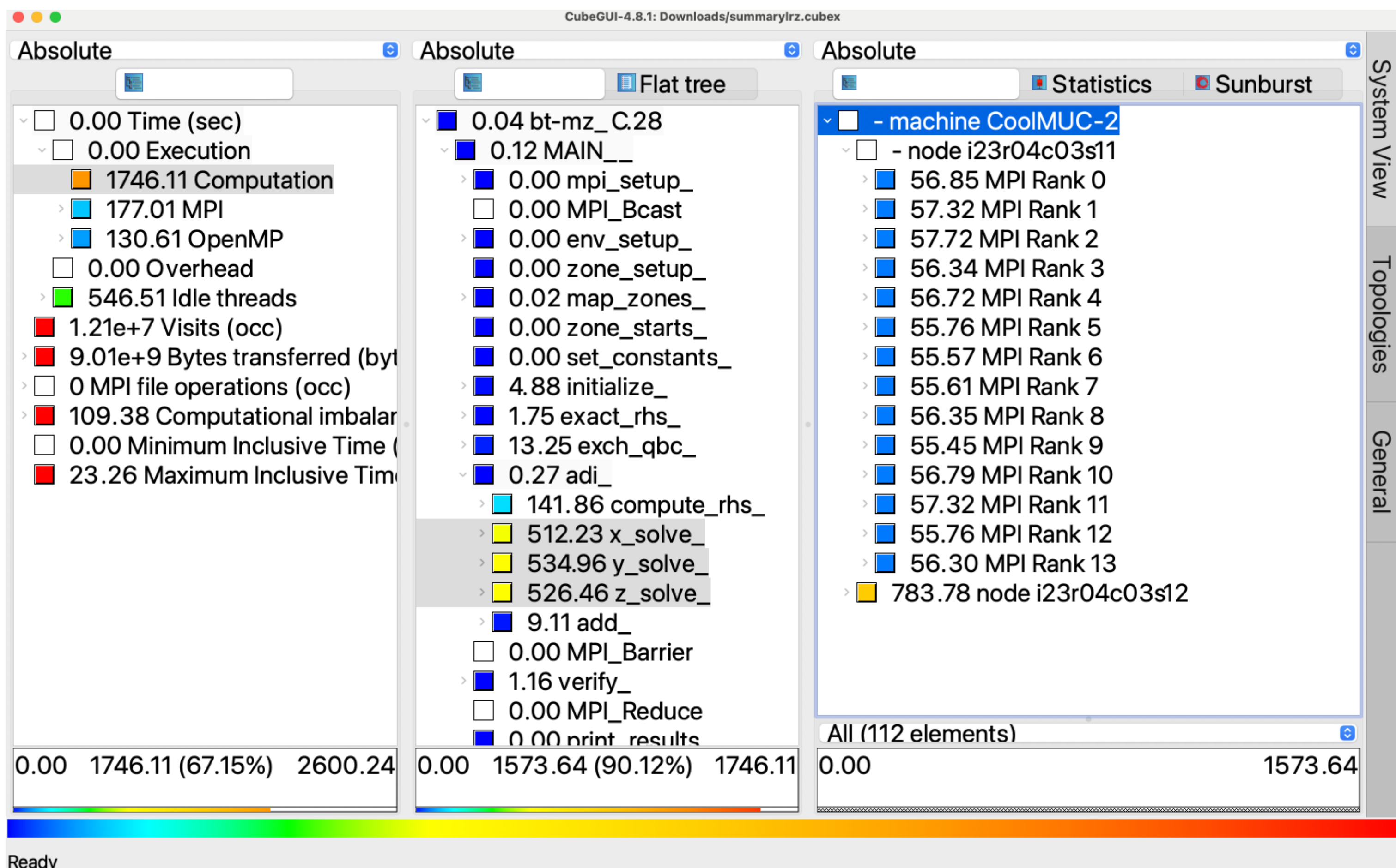
```
$ # Load modules if not loaded already
$ module load intel intel-mpi/2019-intel nano
$ module use /lrz/sys/courses/vihps/2024/modulefiles/
$ module load scorep/8.4-intel-intelmpi scalasca/2.6.1-intel-intelmpi
$ square scorep_bt-mz_sum_filt/
```

This command will post-process a `profile.cubex` and create a summary report `summary.cubex`, then open the CUBE browser.

> **INFO**
>
> Exploring profiles via the CUBE over SSH can be very slow due to the high data transfer rates and latency involved. To improve performance, it is recommended to copy the profile data to a local machine where CUBE is installed. By examining the profile locally, you can benefit from faster data access and more responsive analysis, leading to a more efficient and effective performance tuning process.

Cube is a generic user interface for presenting and browsing performance and debugging information from parallel applications. The Cube main window consists of three coupled panels containing tree displays or alternate graphical views of analysis reports. The left panel shows *performance properties* of the execution, such as time or the number of visits. The middle pane shows the *call tree* or a flat profile of the application. The right pane either shows the *system hierarchy* consisting of, e.g., machines, compute nodes, processes, and threads, a topological view of the application's processes and threads (if available), or a *box plot* view showing the statistical distribution of values across the system. All tree nodes are labeled with a metric value and a color-coded box which can help in identifying hotspots. The metric value color is determined from the proportion of the total (root) value or some other specified reference value, using the color scale at the bottom of the window.

A click on a performance property or a call path selects the corresponding node. This has the effect that the metric value held by this node (such as execution time) will be further broken down into its constituents in the panels right of the selected node. For example, after selecting a performance property, the middle panel shows its distribution across the call tree. After selecting a call path (i.e., a node in the call tree), the system tree shows the distribution of the performance property in that call path across the system locations. A click on the icon to the left of a node in each tree expands or collapses that node. By expanding or collapsing nodes in each of the three trees, the analysis results can be viewed on different levels of granularity (inclusive vs. exclusive values).



For example, in the figure we can see the distribution of the "Computation time" of the following three functions `x_solve`, `y_solve`, `z_solve` over 14 MPI ranks on node one and accumulated time across all MPI ranks on node two .

All tree displays support a context menu, which is accessible using the right mouse button and provides further options. For example, to obtain the exact definition of a performance property, select "Online Description" in the context menu associated with each performance property. A brief description can also be obtained from the menu option "Info".

INFO

To make effective use of the GUI please also consult the Cube User Guide or visit the CUBE YouTube channel.

QUESTION

Examine our BT-MZ measurements in the CUBE browser and try to answer the following questions

- What percentage of the total time is spent on computation, MPI, OpenMP?
- What is the name of the routine with the largest execution time (inclusive)?
- What is the name of the routine with the largest execution time (exclusive)?
- How many times has the `adi` routine been called?
- Which routines are the biggest contributors to the runtime?
- What is the min and max execution time of the `compute_rhs` routine across all threads (all MPI processes)?

Scalasca trace analysis

While summary profiles only provide process- or thread-local data aggregated over time, event traces contain detailed time-stamped event data which also allows to reconstruct the dynamic behavior of an application. This enables tools such as the Scalasca trace analyzer to provide even more insights into the performance behavior of an application, for example, whether the time spent in MPI communication is real message processing time or incurs significant wait states (i.e., intervals where a process sits idle without doing useful work waiting for data from other processes to arrive).

Trace collection and subsequent automatic analysis by the Scalasca trace analyzer can be enabled using the `-t` option of `scalasca -analyze`. Since this option enables trace collection in addition to collecting a summary measurement, it is often used in conjunction with the `-q` option which turns off measurement entirely. (Note that the order in which these two options are specified matters.)

Traces can easily become extremely large and unwieldy, and uncoordinated intermediate trace buffer flushes may result in cascades of distortion, which renders such traces to be of little value. It is therefore extremely important to set up an adequate measurement configuration (i.e., a filtering file and `SCOREP_TOTAL_MEMORY` setting) before initiating trace collection and analysis!

For our example measurement, scoring of the initial summary report with the filter applied estimated a total memory requirement of 27MB per process (see scoring report here). As this exceeds the default `SCOREP_TOTAL_MEMORY` setting of 16MB, use of the prepared filtering file alone is not yet sufficient to avoid intermediate trace buffer flushes. In addition, the `SCOREP_TOTAL_MEMORY` setting has to be adjusted accordingly before starting the trace collection and analysis.

INFO

Renaming or removing the summary experiment directory is not necessary, as trace experiments are created with suffix `trace`.

Make sure that all required software is available

```
$ # Load modules if not loaded already
$ module load intel intel-mpi/2019-intel nano
```

```
$ module use /lrz/sys/courses/vihps/2024/modulefiles/
$ module load scorep/8.4-intel-intelmpi scalasca/2.6.1-intel-intelmpi
```

Go to our work directory with already build executable and prepared filtering file

```
$ cd $HOME/tw45/NPB3.3-MZ-MPI/bin.scorep
```

Let's copy `scalasca.sbatch` to the current directory

```
$ cp ../jobscript/coolmuc2/scalasca.sbatch .
```

Let's examine what `scalasca.sbatch` does by executing `nano scalasca.batch`

```
1  #!/bin/bash
2  #SBATCH -o bt-mz.%j.out
3  #SBATCH -e bt-mz.%j.err
4  #SBATCH -J bt-mz
5  #SBATCH --clusters=cm2_tiny
6  #SBATCH --partition=cm2_tiny
7  #SBATCH --reservation=hhps1s24
8  #SBATCH --nodes=2
9  #SBATCH --ntasks=28
10 #SBATCH --ntasks-per-node=14
11 #SBATCH --get-user-env
12 #SBATCH --time=00:05:00
13
14 module use /lrz/sys/courses/vihps/2024/modulefiles/
15 module load scorep/8.4-intel-intelmpi scalasca/2.6.1-intel-intelmpi
16 export OMP_NUM_THREADS=4
17
18 # Score-P measurement configuration
19
20 export SCOREP_FILTERING_FILE=../config/scorep.filt
21 export SCOREP_TOTAL_MEMORY=27MB
22 #export SCAN_ANALYZE_OPTS="--time-correct"
23
24 # Benchmark configuration (disable load balancing with threads)
25 export NPB_MZ_BLOAD=0
26 PROCS=28
27 CLASS=C
```

```
28
29 # Run the application
30 scalasca -analyze -t mpiexec -n $SLURM_NTASKS ./bt-mz_$CLASS.$PROCS
```

In the first highlighted lines we set the measurement configuration, i.e. use the prepared filter file and set the required amount of memory for tracing based on scoring. And in the last highlighted line we enabled Scalasca trace analysis with the `-t` option.

Now we are ready to submit our batch script

```
sbatch scalasca.sbatch
```

After successful trace collection and analysis you should see freshly generated experiment directory `scorep_bt-mz_C_8x6_trace`. Let us examine what is inside this directory:

```
$ ls -1 scorep_bt-mz_C_8x6_trace
MANIFEST.md
profile.cubex
scorep.cfg
scorep.filter
scorep.log
scout.cubex
scout.log
traces
traces.def
traces.otf2
trace.stat
```

Among the already known files there are some new ones, e.g. a copy of the filter file `scorep.filt`, an OTF2 trace archive consisting of the anchor file `traces.otf2`, the global definitions file `traces.def` and the per-process data in the `traces/` directory. Finally, the experiment also includes the trace analysis reports `scout.cubex` and `trace.stat`, and a log file containing the output of the trace analyser (`scout.log`).

Let's examine `scout.log` if the trace analysis was successful:

```
$ cat scorep_bt-mz_C_8x6_trace/scout.log
S=C=A=N: Tue Jun  4 18:42:20 2024: Analyze start
```

```
/dss/dsshome1/lrz/sys/spack/release/22.2.1/opt/x86_64/intel-mpi/2019.12.320-gcc-wx7cjlg/compilers_and_libraries_2020.4.320/linux/mpi/intel64/bin/mpiexec -n 28 /lrz/sys/courses/vihps/2024/tools/scalasca/2.6.1/intel_intelmpi/bin/scout.hyb ./scorep_bt-mz_C_28x4_trace/traces.otf2
SCOUT   (Scalasca 2.6.1)
Copyright (c) 1998-2022 Forschungszentrum Juelich GmbH
Copyright (c) 2014-2021 RWTH Aachen University
Copyright (c) 2009-2014 German Research School for Simulation Sciences GmbH

Analyzing experiment archive ./scorep_bt-mz_C_28x4_trace/traces.otf2

Opening experiment archive ... done (0.013s).
Reading definition data    ... done (0.015s).
Reading event trace data   ... done (0.131s).
Preprocessing              ... done (0.181s).
Analyzing trace data       ... done (10.301s).
Writing analysis report    ... done (0.129s).

Max. memory usage         : 279.777MB

Total processing time     : 10.841s
S=C=A=N: Tue Jun  4 18:42:37 2024: Analyze done (status=0) 17s
```

There are no errors or warnings, so the analysis was successful.

> **INFO**
>
> Sometimes in `scout.log` the Scalasca trace analyzer warns about point-to-point clock condition violations. These violations happen when the local clocks of individual compute nodes are not properly synchronized, causing logical event order errors. For example, a receive operation might appear to finish before the corresponding send operation starts, which is impossible. Scalasca has a correction algorithm to fix these errors and restore the logical event order, while trying to keep the intervals between local events unchanged.
>
> To use this correction algorithm, you need to pass the `--time-correct` option to the Scalasca trace analyzer. Since the analyzer is started with the `scalasca -analyze` command, you set this option using the `SCAN_ANALYZE_OPTS` environment variable. This variable holds the command-line options for `scalasca -analyze` to pass to the trace analyzer. You can re-analyze an existing trace measurement using the `-a` option with `scalasca -analyze`, so you don't have to collect new data.

The additional time required to execute the timestamp correction algorithm is typically small compared to the trace data I/O time and waiting times in the batch queue for starting a second analysis job. On platforms where clock condition violations are likely to occur (i.e., clusters), it is therefore often convenient to enable the timestamp correction algorithm by default.

Similar to the summary report, the trace analysis report can finally be postprocessed and interactively explored using the Cube report browser, e.g. by using the `square` command

```
$ square scorep_bt-mz_C_8x6_trace/
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_bt-mz_C_8x6_trace/trace.cubex...
```

The report generated by the Scalasca trace analyzer (i.e. `trace.cubex`) is again a profile in CUBE4 format, however, enriched with additional performance properties, e.g. "Delay costs", "Critical path", etc. Examination shows that roughly half of the time spent in MPI point-to-point communication is waiting time, mainly in "Late Sender" wait state.

> **INFO**
>
> A detailed list and description of performance metrics one can be found here.

While the execution time in the `x_solve`, `y_solve` and `z_solve` routines looked relatively balanced in the summary profile, examination of the "Imbalance" in "Critical path" metric shows that these routines in fact exhibit a small amount of imbalance, which is likely to cause the wait states at the next synchronization point. This can be verified using the "Late Sender" in "Delay costs" metric, which confirms that the `x_solve`, `y_solve` and `z_solve` routines are responsible for significant amount of the "Late Sender" wait states.