Profiling and optimization exercise

NPBenchmark

- cp /tmp1/exercise.tar.gz .
- Move into the NPB3.3-MZ-MPI directory

```
[nvarini@bellatrix NPB3.3-MZ-MPI]$ 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                           *
 ***************************************************************
[nvarini@bellatrix NPB3.3-MZ-MPI]$
```

NPBenchmark

- What does it do?
  - Solves a discretized version of unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules

- Uses MPI & OpenMP in combination
  - 4 processes with 4 threads each should be reasonable
    - don't expect to see speed-up when run on a laptop!
  - bt-mz_W.4 should run in around 5 to 12 seconds on a laptop
  - bt-mz_B.4 is more suitable for dedicated HPC compute nodes
    - Each class step takes around 10-15x longer

Launch a hybrid parallel application

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 1:00:00
#SBATCH –p phcpc2016

module load gcc/4.8.3 mvapich2/2.0.1/gcc-4.4.7  scorep/1.4/gcc-4.83

export OMP_NUM_THREADS=4
time srun -n 4 ./bin/bt-mz_W.4 > output

=============================================
```

Let's add Score-p instruction

- Change back to directory containing NPB BT-MZ

```
% cd ..
```

- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: MPIF77

```
...
#---------------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------------
#MPIF77 = mpif77

# Alternative variants to perform instrumentation
...
MPIF77 = scorep mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK   = $(MPIF77)
...
```

Uncomment the Score-P compiler wrapper specification

Rebuild the application

- Return to root directory and clean-up

```
% make clean
```

- Re-build executable using Score-P instrumenter

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 W
scorep mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  scorep mpif77 -c  -O3 -fopenmp timers.f
scorep mpif77 –O3 -fopenmp -o ../bin.scorep/bt-mz_W.4 \
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
Built executable ../bin.scorep/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

Score-P environment variables

- Score-P measurements are configured via environment variables:

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
 [...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
 [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
 [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
 [...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
 [...]
SCOREP_METRIC_PAPI
  Description: PAPI metric names to measure
 [...]
SCOREP_METRIC_RUSAGE
  Description: Resource usage metric names to measure
 [... More configuration variables ...]
```

Launch a hybrid parallel application

Jobscript.sh

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 1:00:00
#SBATCH –p phpc2016

module load gcc/4.8.3 mvapich2/2.0.1/gcc-4.4.7  scorep/1.4/gcc-4.8.3

export OMP_NUM_THREADS=4
cd bin.scorep
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
time srun -n 4 ./bt-mz_W.4 > output

=====================================================
```

Score-P output

- Creates experiment directory ./scorep_bt-mz_W_4x4_sum containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```
% ls
...   scorep_bt-mz_W_4x4_sum
% ls scorep_bt-mz_W_4x4_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with CUBE / ParaProf

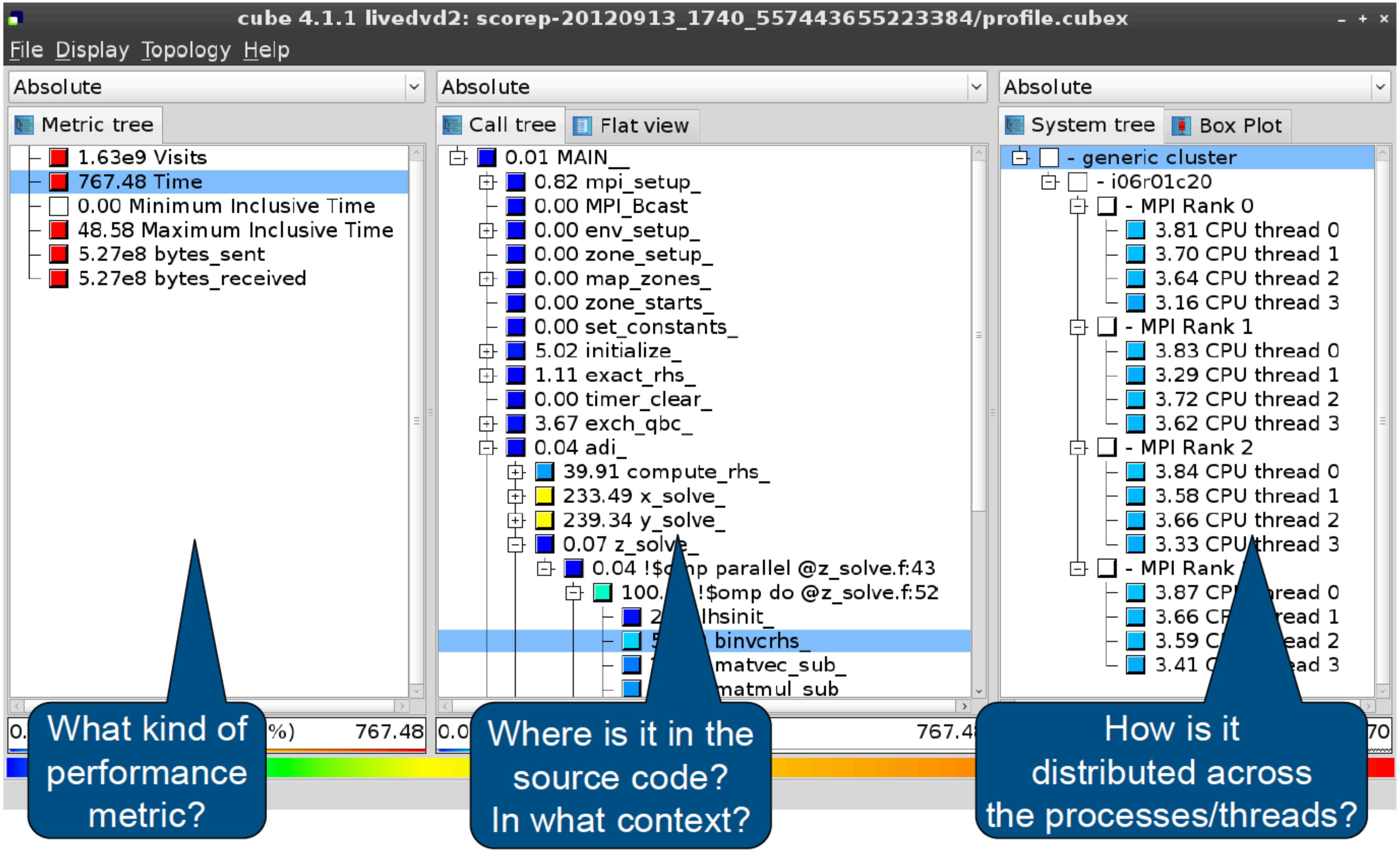
```
% cube scorep_bt-mz_W_4x4_sum/profile.cubex

             [CUBE GUI showing summary analysis report]

% paraprof scorep_bt-mz_W_4x4_sum/profile.cubex

         [TAU ParaProf GUI showing summary analysis report]
```

Analyze the data with CUBE



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SCITAS
INFORMATIQUE SCIENTIFIQUE & SUPPORT APPLICATIF
SCIENTIFIC IT AND APPLICATION SUPPORT

Score-P filtering

- Report scoring with prospective filter listing
  6 USR regions

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
binvcrhs*
matmul_sub*
matvec_sub*
exact_solution*
binvrhs*
lhs*init*
timer_*

% scorep-score -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   23MB
Estimated requirements for largest trace buffer (max_buf): 8MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       16MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=16MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)
```

23 MB of memory in total,
8 MB per rank!

Redo the experiment with filter file

- Set new experiment directory and re-run measurement with new filter configuration

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum_filtered
% export SCOREP_FILTERING_FILE=../config/scorep.filt
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:    4 x    4
 Iterations:  200    dt:    0.000800
 Number of active processes:       4
 Use the default load factors with threads
 Total number of threads:      16  (  4.0 threads/process)
 Use the default load factors with threads

 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 8.11
```

BT-MZ Tuned Summary Analysis Report

- Scoring of new analysis report as textual output

```
% scorep-score scorep_bt-mz_W_4x4_sum_filtered/profile.cubex
Estimated aggregate size of event trace:                   23MB
Estimated requirements for largest trace buffer (max_buf): 8MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       16MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=16MB 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  7,357,804 739,321   25.32   100.0          34.25  ALL
         OMP  6,882,860 685,952   16.64    65.7          24.26  OMP
         COM    371,956  45,944    3.90    15.4          84.87  COM
         MPI    102,286   7,316    4.78    18.9         653.21  MPI
         USR        728     109    0.00     0.0           2.41  USR
```

- Significant reduction in runtime (measurement overhead)
  - Not only reduced time for USR regions, but MPI/OMP reduced too!
- Further measurement tuning (filtering) may be appropriate
  - e.g., use “timer_*” to filter timer_start_, timer_read_, etc.

BT-MZ trace with scalasca (but with intel compiler)

- Re-run the application using Scalasca nexus with "**-t**" flag

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
% OMP_NUM_THREADS=4 scan -t mpiexec -np 4 ./bt-mz_W.4
S=C=A=N: Scalasca 2.1 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_W_4x4_trace experiment archive
S=C=A=N: Thu Jun 12 18:05:39 2014: Collect start
mpiexec -np 4 ./bt-mz_B.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:    8 x    8
 Iterations: 200     dt:    0.000300
 Number of active processes:      4

  [... More application output ...]

S=C=A=N: Thu Jun 12 18:05:58 2014: Collect done (status=0) 19s
  [... continued ...]
```

BT-MZ Trace

- Continues with automatic (parallel) analysis of trace files

```
S=C=A=N: Thu Jun 12 18:05:58 2014: Analyze start
mpiexec -np 4 scout.hyb ./scorep_bt-mz_W_4x4_trace/traces.otf2
SCOUT   Copyright (c) 1998-2012 Forschungszentrum Juelich GmbH
        Copyright (c) 2009-2012 German Research School for Simulation
                                Sciences GmbH

Analyzing experiment archive ./scorep_bt-mz_W_4x4_trace/traces.otf2

Opening experiment archive ... done (0.002s).
Reading definition data    ... done (0.004s).
Reading event trace data   ... done (0.130s).
Preprocessing              ... done (0.259s).
Analyzing trace data       ...
  Wait-state detection (fwd)      (1/4) ... done (0.575s).
  Wait-state detection (bwd)      (2/4) ... done (0.138s).
  Synchpoint exchange             (3/4) ... done (0.358s).
  Critical-path analysis          (4/4) ... done (0.288s).
done (1.360s).
Writing analysis report    ... done (0.121s).

Total processing time     : 1.924s
S=C=A=N: Thu Jun 12 18:06:00 2014: Analyze done (status=0) 2s
```

BT-MZ trace

- Produces trace analysis report in experiment directory containing trace-based wait-state metrics

```
% square  scorep_bt-mz_W_4x4_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report...
INFO: Displaying ./scorep_bt-mz_W_4x4_trace/trace.cubex...

                [GUI showing trace analysis report]
```

Score-P analysis of MD code

- Download the code ljmd-tg11.tar.gz from https://sites.google.com/site/akohlmey/software/ljmd
- Untar it into the scratch directory
- Cd 01_baseline
- Module load gcc/4.8.3 mvapich2/2.0.1/gcc-4.4.7 scorep/1.4/gcc-4.8.3
- Open the Makefile and set **F90=scorep gfortran CC=scorep gcc**
- Make
- Create a job script
- ====================================

```
#!/bin/bash -l
#SBATCH --nodes 1
#SBATCH –p phcpc2016
#SBATCH --time 1:00:00

module load gcc/4.8.3 mvapich2/2.0.1/gcc-4.4.7  scorep/1.4/gcc-1.4

export SCOREP_EXPERIMENT_DIRECTORY=ljmd_serial
./ljmd-f.x < argon_108.inp
```

==

Exercise2: Matrix-vector multiplication

- Fortran allocates the memory by column while C by row
- Wrong access order pattern causes penalties
- Matrix-vector multiplication is a fairly simple example
- y = y + A*x;
- Copy the file matvec.f90 from /tmp1/ exercise.tar.gz into your directory;
- Run the simulation and measure the time;
- Examine the archive generated;
- Repeat with –O3;

Exercise 3: matrix-transpose problem

- Go inside trasposta directory
- Analyse the function cclock.c
- Analyse the subroutine trasponi in test_trasp.f90
- Compile the code and analyse the output.

Matrix-matrix multiplication: dgemm

- C = alpha*A*B + beta*C
- Copy the file link … matmul into your local directory
- Insert the DGEMM call into the fortran code
- Recompile with intel:
  ifort -mkl -o matmul.x matmul.f90 cclock.o
- Run and measure the performance