Overview of Profiling on ARCHER2

29.09.2021 HPC for Wind Energy workshop tutorial
Evgenij Belikov
e.belikov@epcc.ed.ac.uk

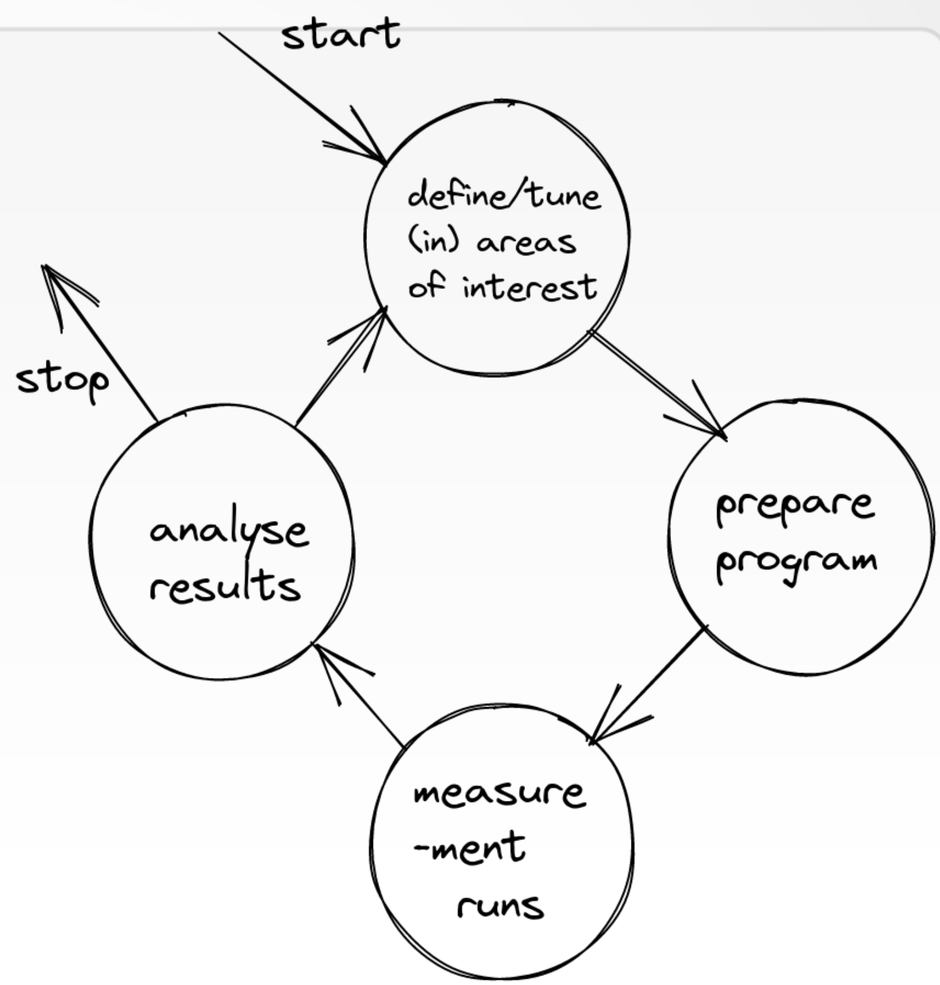
Overview

- **Part 1**: introduction to profiling (20min)
- ends vs means-based metrics
- assessing scalability
- sampling vs instrumentation
- tools overview
- **break** (10min)

- **Part 2**: hands-on exercise (90min)
- profile some code from NAS parallel benchmarks
- use Scalasca and/or CrayPAT on ARCHER2

What is profiling and why profile

- *profiling* is the *empirical* and *iterative* process of recording and assessing program behaviour
- profiling helps us understand and characterise program behaviour and assists *optimisation* (e.g. performance, scalability, *identifying bottlenecks*)
- diverse tools facilitate profiling of different aspects (e.g. comms, I/O)



NB: figure drawn in Excalidraw

Brief look on metrics

- *ends-based* metrics are higher-level aggregate value we wish to improve
- relatively easy to measure but can be fairly high-level
- e.g. program's elapsed time, 'science' per resource ratio
- give little hints towards possible optimisations

- *means-based* metrics are lower-level and more detailed; often time series; can be combined to form derived metrics
- may be tricky to measure and generate huge data
- e.g. time spent in specific functions, memory accesses, time spent waiting for messages to arrive
- help identify potential issues and explain behaviour

Common scalability metrics

- *Speedup* : run time on a single node (rather than core, as on ARCHER2 node is a unit of allocation) divided by the run time on multiple nodes
- *Efficiency* : speedup divided by the number of nodes (ideal = 1.0 = 100%)
- Load balance
- Computation to communication ratio

- *POP metrics ** :
- Global Efficiency
  - Parallel Efficiency
    - Load Balance Eff.
    - Communication Eff.
      - Serialisation Eff.
      - Transfer Eff.
  - Computation Eff.
    - IPC Scaling
    - Instruction Scaling

* Performance Optimisation and Productivity Centre of Excellence, http://www.pop-coe.eu/

Weak vs strong scaling

- *weak scaling*: we assess program behaviour change as we scale problem size with the number of nodes (e.g. we double the amount of work when doubling the number of nodes)
- identify scalability limits if work is abundant
- some bottlenecks appear as a growing fraction of total runtime (e.g. comms)

- *strong scaling*: we keep problem size constant as we increase the number of nodes
- assessing scaling limits for a particular problem size
- eventually there is no benefit from adding extra resources as we run out of work

Time profiling using timers

- most applications and scientific packages incorporate *timers* to record times of different program phases (or allow adding custom ones)
- May need to turn on a special runtime flag
- No specific profiling tool use needed

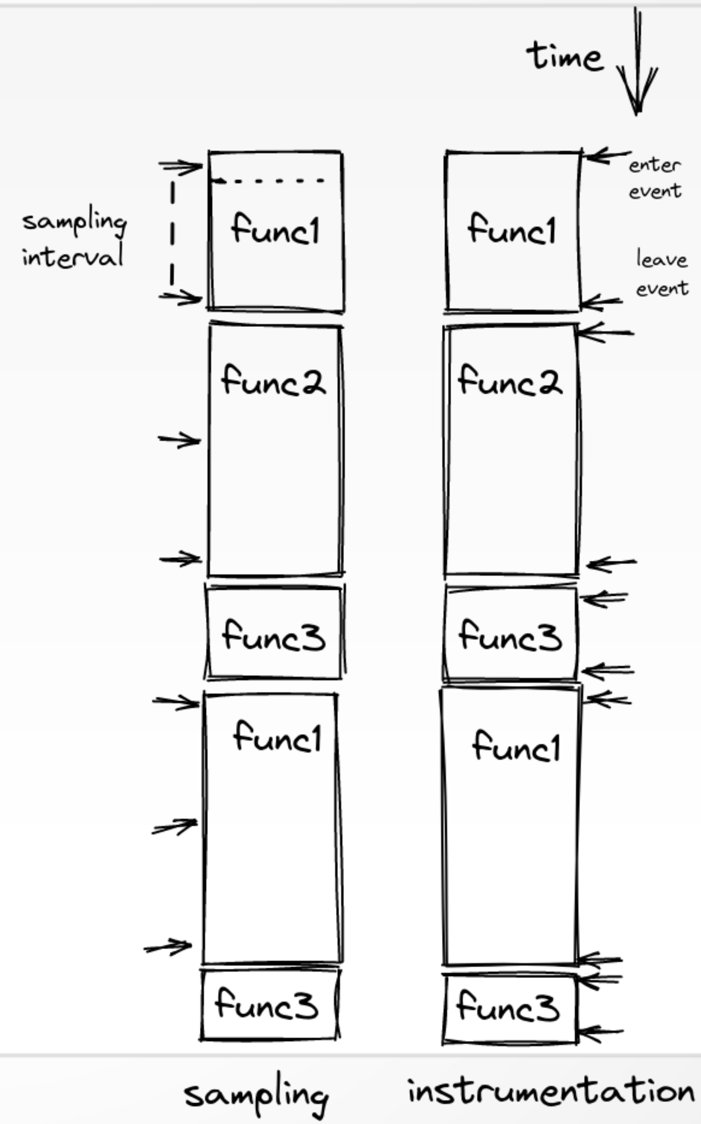
```
# run time from Slurm, for finished job by jobid
sacct -j <jobid> --format=JobID,JobName,Elapsed
```

```
********************************************************
   Finished running the atmosphere core
********************************************************


 Timer information:
    Globals are computed across all threads and processors

 Columns:
    total time: Global max of accumulated time spent in timer
    calls: Total number of times this timer was started / stopped.
    min: Global min of time spent in a single start / stop
    max: Global max of time spent in a single start / stop
    avg: Global max of average time spent in a single start / stop
    pct_tot: Percent of the timer at level 1
    pct_par: Percent of the parent timer (one level up)
    par_eff: Parallel efficiency, global average total time / global max total


   timer_name                                            total       calls
 1 total time                                        485.69940           1
 2  initialize                                       130.56295           1
 2  time integration                                 351.26449         100
 3   physics driver                                   58.56806         100
 4    calc_cldfraction                                 0.00199           1
 4    RRTMG_sw                                         1.85335           1
 4    RRTMG_lw                                         0.89821           1
 4    MYNN_sfclay                                      0.77612         100
 4    Noah                                             0.83311         100
 4    MYNN_pbl                                         6.06804         100
 4    bl_ysu_gwdo                                      4.14890         100
 4    Grell-Freitas                                   33.20711         100
 3   atm_rk_integration_setup                          5.35148         100
 3   atm_compute_moist_coefficients                    2.03910         100
 3   physics_get_tend                                 13.54701         100
 3   atm_compute_vert_imp_coefs                        2.85826         300
 3   atm_compute_dyn_tend                             56.61109         900
 3   small_step_prep                                   6.92453         900
 3   atm_advance_acoustic_step                        23.63283        1200
 3   atm_divergence_damping_3d                         3.33569        1200
 3   atm_recover_large_step_variables                 34.00798         900
 3   atm_compute_solve_diagnostics                    41.12899         900
 3   atm_rk_dynamics_substep_finish                   10.62201         300
 3   atm_advance_scalars                              15.36223         200
 3   atm_advance_scalars_mono                         51.82172         100
 3   microphysics                                     16.84774         100
 4    Thompson                                        12.42570         100
```

*output from an MPAS atmosphere run

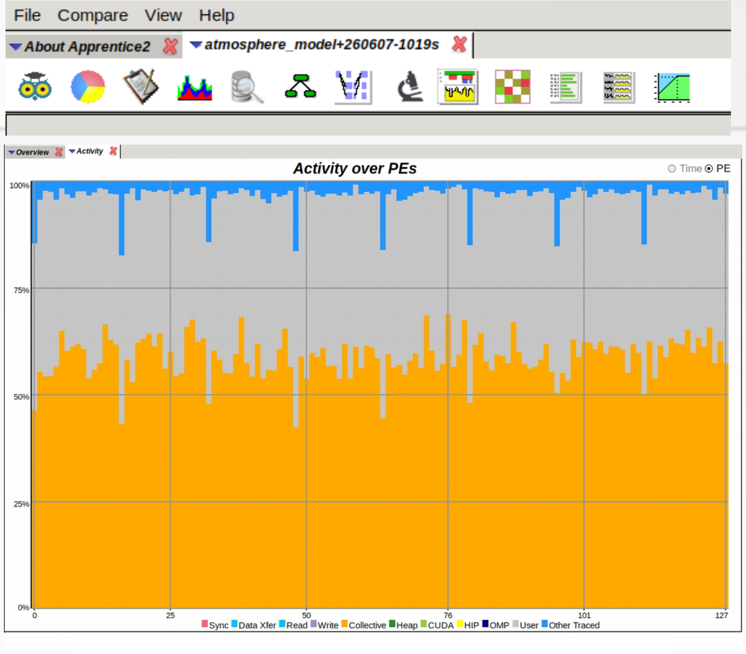
Sampling vs instrumentation

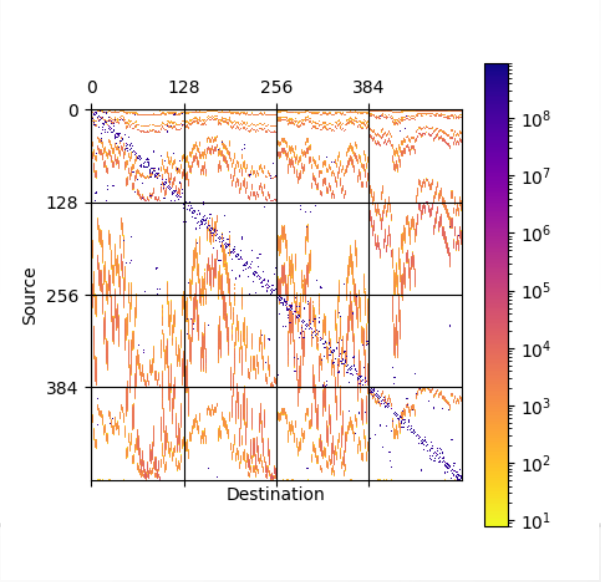
- *sampling* halts the program at defined intervals to take measurements (may miss small functions)
- *instrumentation* adds calls to code or binary to record events; can result in more accurate view but add larger overheads
- *tracing* as opposed to profiling, instead of summarising, keeps all events: most detailed, but creates huge logs and I/O may cause perturbation



CrayPAT, app2 (GUI), pattools

- CrayPAT is the default profiling suite on ARCHER2
- flexible/configurable to collect a broad variety of data per process (number of function calls, MPI messages sent/received, memory used etc)
- *pattools* * developed at EPCC for postprocessing, extracting tables as csv from logs and visualisation





* see https://github.com/pbartholomew08/pattools for installation and usage guide

CrayPAT basic usage

- load modules
  (known issue: perftools not visible by default, workaround is to restore PrgEnv twice)
- instrument and run (need to compile with -c and a separate link stage)
- view results (app2 GUI needs *ssh -X*)
- tracing available via *pat_build -w* (and options)
- for more info see ARCHER2 and HPE/Cray docs * and man pat_build / pat_report

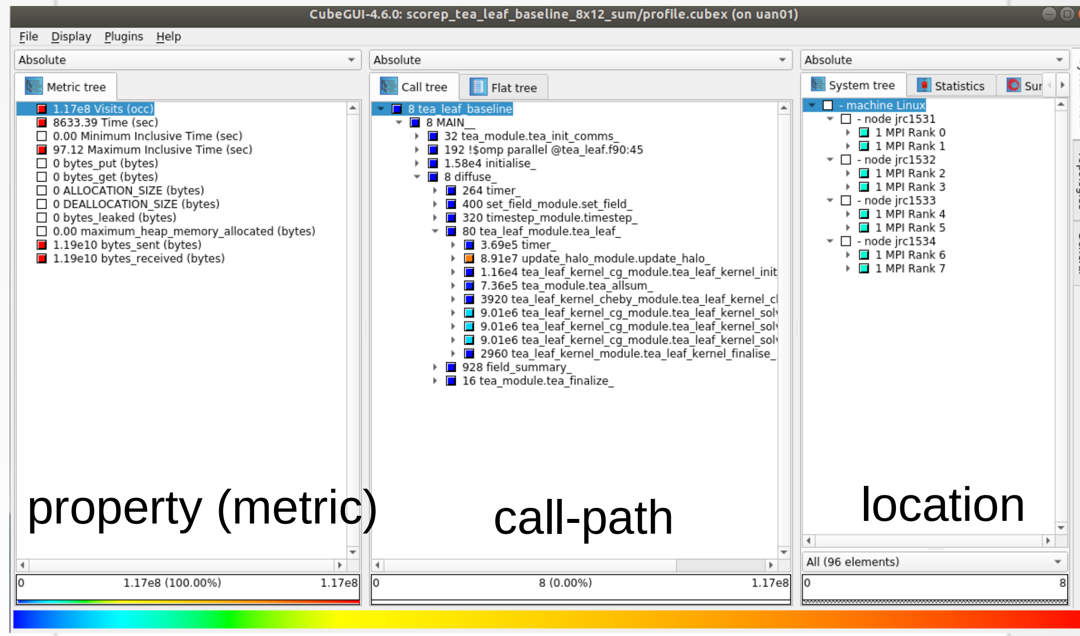
```
module restore -s PrgEnv-gnu
module restore -s PrgEnv-cray
module load perftools-base
module load perftools
```

```
# adjust COMPFLAGS in make.def
make bt-mz CLASS=C NPROCS=8
# in bin subdir, to get ...+pat exe

pat_build bt-mz_C.8
# run baseline and +pat executable
# to produce .xf file or experiment dir

# then generate text report
pat_report <experiment_dir>

export PAT_RT_SUMMARY=0
app2 <experiment_dir>
```

* https://docs.archer2.ac.uk/user-guide/profile/ & https://support.hpe.com/connect/s/

Scalasca, ScoreP, cube (GUI)

- *Scalasca* is a freely available profiling suite on ARCHER2, based on the ScoreP measurement infrastructure
- *cube* GUI helps examine measurements (metric, call tree, system tree)
- supports the portable Open Trace Format (OTF2)
- provides a user instrumentation API
- SIONlib (optional) for massively parallel I/O



official Scalasca website: http://scalasca.org/

Scalasca basic usage

- instrument: prepend *scorep --user* to compiler and add *-g* to compiler flags to keep debug symbols
- set environment & run with sbatch (use our reservation)
- analyse (using cube), need ssh -X, call
  cube profile.cubex
- for detailed information see ARCHER2 training materials * (we will use the same example exercise)

```
module restore PrgEnv-gnu
module load scalasca
```

NB: check with *ldd* that scorep objects were linked into the executable
(if you use Cmake an extra step is needed, check ScoreP manual for SCOREP_WRAPPER=off)

Hint: can define MPIF77 = $(PREP) ftn
in the config/make.def, then call
PREP="scorep --user" make <...>

```
export SCOREP_ENABLE_PROFILING=true
export SCOREP_EXPERIMENT_DIRECTORY=mydir
```

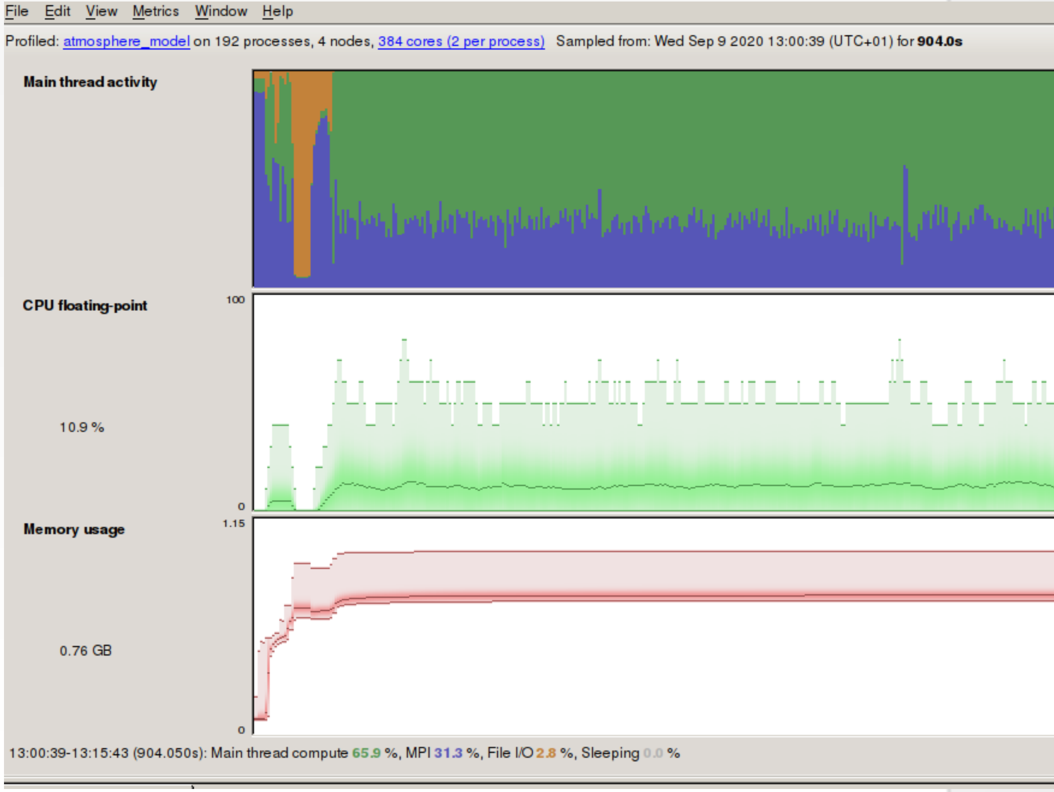
We can use a filter to reduce overhead and memory requirements

```
export SCOREP_FILTERING_FILE=../config/scorep.filt
```

* https://www.archer2.ac.uk/training/courses/210727-scalasca/ (videos)
https://github.com/EPCCed/archer2-scalasca-2021-07-27 (course materials)

ARM MAP

- scalable adaptive sampling; convenient GUI; may require relinking *
- timeline view of computation, memory, communication, I/O
- proprietary, currently not available to users directly



* also some setup is needed as /home is not visible on ARCHER2 compute nodes

Other tools (not yet supported)

- **Vampir**: proprietary, detailed visualisation of communication (e.g. from ScoreP traces)
- **BSC tools** (Extrae, Dimemas, ParaVer GUI), free, can visualise comms as timeline
- **TAU** suite (free, complex, many features)
- **HPC Toolkit** (free, timline)
- **Darshan** (free, focused on I/O; can generate summary pdf report)
- **Intel tools** * (VTune, Advisor; proprietary, complex; support for Roofline analysis)
- hardware performance counters; availability:

```
perf list
papi_avail
```

* NB: ARCHER2 is AMD-based

Summary

- Overview of basic profiling on ARCHER2
- profiling helps us chracterise program behaviour and identify performance/scalability issues
- use means-based metrics to explain observed changes after code changes
- measurement is better than guesswork

- Hand-on exercise
- Profile some code from NAS parallel benchmarks
- Familiarise yourself with basic Scalasca and/or CrayPAT usage on ARCHER2
- Ask for help in chat

these slides: https://github.com/jevbelikov/hpcwe-archer2-profiling-intro

Hands-On Exercise

OUR RESERVATION: ta040_199

- Login onto ARCHER2
- Copy example application from shared location* and build it
- Check and edit run scripts
- Perform a baseline run** without profiling (run from /work/...)
- Instrument the code using CrayPAT or Scalasca
- Run application with profiling on and assess the results
- (optional) perform a tracing run using a small problem size

* **/work/y23/shared/tutorial/NPB3.3-MZ-MPI.tar.gz**

** assuming low variation (in general better to have multiple runs)