Welcome!

Virtual tutorial starts at 15:00 GMT

Please leave feedback afterwards at:
www.archer.ac.uk/training/feedback/online-course-feedback.php

archer

epcc

THE UNIVERSITY OF EDINBURGH

Performance Analysis on ARCHER using CrayPAT

ARCHER Virtual Tutorial, 11th March 2015

Iain Bethune, ibethune@epcc.ed.ac.uk

Reusing this material

archer

epcc

THE UNIVERSITY OF EDINBURGH

Outline

- Overview of CrayPAT
- Perftools-lite
- Sampling Experiments
- Tracing Experiments
  - Automated Profile Analysis
  - Hardware counters
- CrayPAT GUI
- Using the CrayPAT API

Further Help

- ARCHER Best Practice Guide:
  - http://www.archer.ac.uk/documentation/best-practice-guide/performance.php
- CrayDoc:
  - http://docs.cray.com
  - Search "Using Cray Performance Measurement and Analysis Tools"
- Online help:
  - Man pages for the tools
  - `pat_help` utility
- ARCHER training archive:
  - http://www.archer.ac.uk/training/courses/craytools/

Why profile?

- For developers:
  - Understand what the most time-consuming parts of a program are
  - Understand communication patterns & problems
    - E.g. load imbalance, synchronisation costs
  - Tool to help direct development effort to for maximum benefits

- For users?
  - Understand why your program performs in a certain way
  - Help with choice of appropriate parameters, MPI processes…

Overview of CrayPAT

- Cray's Performance Analysis Toolkit (PAT)
  - Measuring and understanding performance of parallel codes on Cray systems
  - Parallel Programming languages / APIs:
    - MPI, OpenMP, CUDA, CAF, Chapel, Global Arrays, DMAPP, SHMEM…
  - Libraries:
    - BLAS/LAPACK/ScaLAPACK, FFTW, PETSc…
  - I/O:
    - ADIOS, HDF5, NetCDF, POSIX I/O, (MPI I/O)…

Overview of CrayPAT

- Compared with other tools
  - e.g. Allinea MAP, Intel TAC, Scalasca, TAU …

  + Works ‘out of the box’
  + Various levels of detail
  + Extreme customisability for expert users

  - Only on Cray Platforms
  - GUI not as powerful as e.g. MAP

Overview of CrayPAT

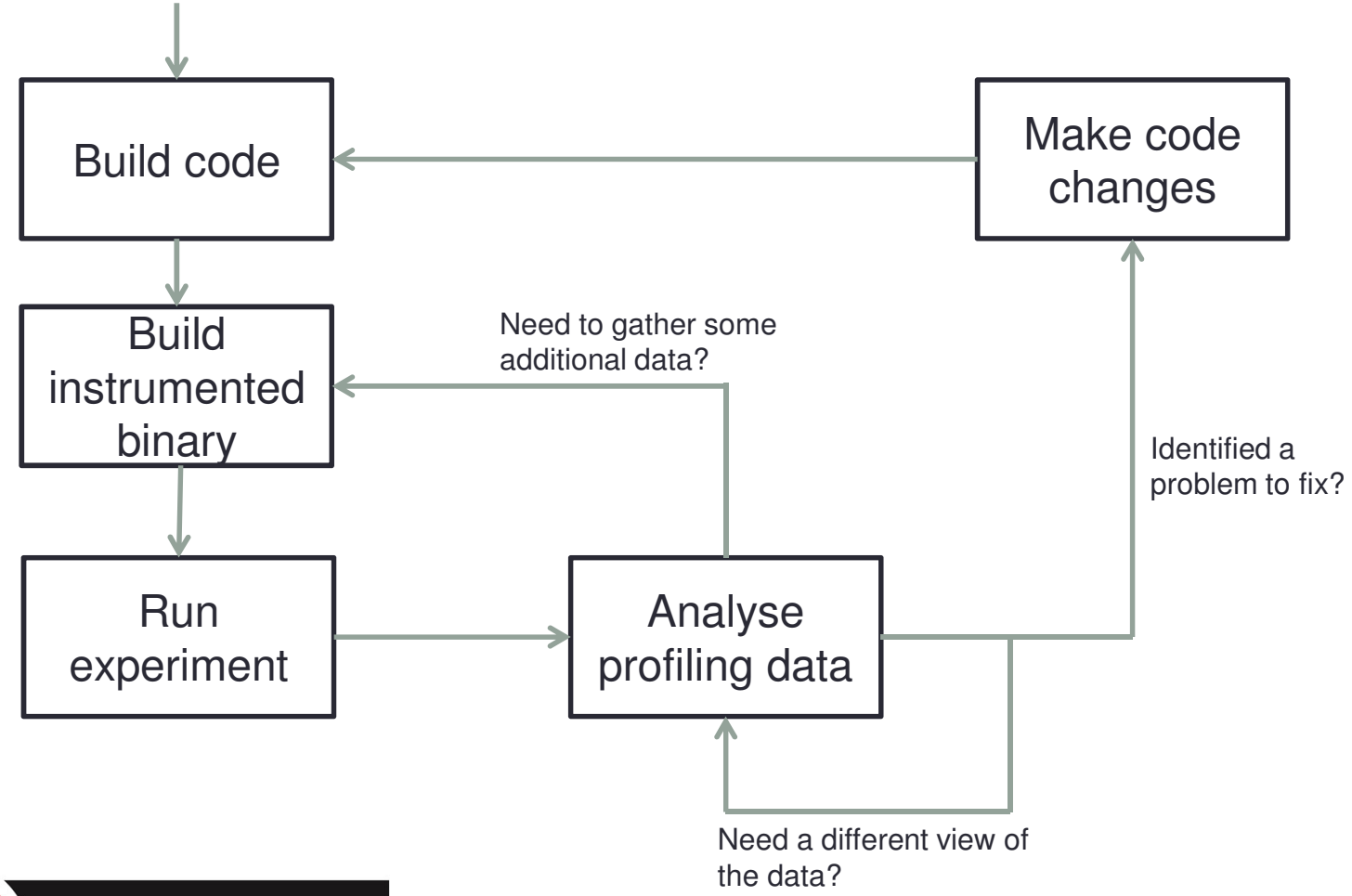
- Tools
  - `pat_build`
    - Instruments existing binaries for profiling
  - `pat_report`
    - Report generator, analyses data from profiling runs
  - Apprentice2
    - GUI for analysing profiling data
  - Reveal
    - GUI for code-level analysis, compile-time optimization feedback
    - NB. For Cray compiler only.

Overview of CrayPAT

- Choosing a suitable job for profiling
  - Program execution should be representative of real production job
  - Must be reasonably short, to avoid generating large data, waste AUs
  - Must be long (enough) to hide start-up, finalisation parts
  - Should include all the I/O of a normal job

- Example
  - Using CP2K - www.cp2k.org
  - H2O-64 benchmark - http://www.cp2k.org/performance#h20-64
  - Takes ~80s using 24 MPI processes, single node of ARCHER

Overview of CrayPAT



Perftools-lite

- Extremely easy introduction to profiling tools
  - Automatically gather profiling data during program execution
  - Basic reporting dumped into standard output at end of run
  - Generate CrayPAT data files for further analysis (if needed)

Perftools-lite

1. Load the perftools-lite module

```
module load perftools-lite
```

2. Build your program as normal
   - Use your configure, Makefile, build scripts etc.
   - Look for message at end:

```
INFO: creating the CrayPat-instrumented executable
'/home/z01/z01/ibethune/cp2k/exe/ARCHER/cp2k.psmp'
(sample_profile) ...OK
```

Perftools-lite

3. Run your program
   - Usual PBS job submission script

4. Basic profiling data appears at the end of job output
   - Overall job info
   - Top 10 most time-consuming functions
   - I/O, memory information
   - Report also saved in `*.rpt` file
   - A CrayPAT performance data file `*.ap2` also created for further analysis

Perftools-lite

See example files.

Sampling Experiments

- What is sampling?
  - Every so often (100 Hz default), look at the call stack of the program
  - Record which function is being executed (+ callers etc.)
  - A good starting point if you know nothing about the behaviour of a program
  - Low overhead (~1%)
  - Very easy to set up & run

Sampling Experiments

1. Load the perftools module

```
module load perftools
```

2. Build your program as normal
   - Use your configure, Makefile, build scripts etc.
   - NB. Compile and link stages must be separated

3. Build a sampling-instrumented program

```
pat_build -o cp2k.psmp+samp cp2k.psmp
```

Sampling Experiments

3. Run your program
   - Usual PBS job submission script
   - Change the name of the executable!

4. Once job has completed, CrayPAT will dump data file(s) into the run directory
   - `*.xf` file
   - Or, if running on large numbers of PEs, a directory containing several `*.xf` files

Sampling Experiments

5. Generate a report on the data

```
pat_report *.xf > report
```

- Produces a text report file
- Produces a portable performance data file `*.ap2`
- Produces a `*.apa` Automated Profiling Analysis file

Sampling Experiments

See example files.

Tracing Experiments

- What is tracing?
  - 'Trace intercept routines' inserted at entry and exit of routines
    - Records amount of time spend in each call of a function
    - Exact sequence of events in a program execution
    - Allows for checking state of hardware counters
  - Possible to generate endless detail about program execution
  - Moderate overhead (~5-10%), depending on what you choose to trace
  - Balance between detailed measurement and disturbing the experiment

Tracing Experiments

1. Load the perftools module

```
module load perftools
```

2. Build your program as normal
   - Use your configure, Makefile, build scripts etc.
   - NB. Compile and link stages must be separated

3. Build a tracing-instrumented program

```
pat_build [options] -o cp2k.psmp+trace cp2k.psmp
```

Tracing Experiments

- Pat_build options:
  - For full list see `man pat_build`
  - Tracegroups (`-g`)
    - e.g. mpi, lapack, omp
  - Tracing user functions
    - `-w` enables tracing user functions
    - `-T` trace specific functions
    - `-u` trace all visible user functions (use with extreme caution!)
- Complex to set up
  - Except if you only want to trace e.g. MPI library calls
  - This is where APA helps

Tracing Experiments

- Automated Profiling Analysis (APA)
  - From the sampling experiment report generation, a `*.apa` file was generated containing recommended options for pat_build to set up a tracing experiment

```
pat_build -O *.apa
```

  - Defaults:
    - Trace MPI calls
    - Gather default hardware counter group
    - Trace user functions with > 1% of samples, up to limit of 200
    - Very small functions (< 200 bytes) not traced to limit overhead

Tracing Experiments

3. Run your program
   - Usual PBS job submission script
   - Change the name of the executable!

4. Once job has completed, CrayPAT will dump data file(s) into the run directory
   - `*.xf` file
   - Or, if running on large numbers of PEs, a directory containing several `*.xf` files

Tracing Experiments

5. Generate a report on the data

```
pat_report *.xf > report
```

- Produces a text report file
- Produces a portable performance data file `*.ap2`

Tracing Experiments

- `pat_report` options
  - Report generation is (almost) endlessly customisable
  - There are several pre-defined reports that are a good place to start:
    - `-O profile` (default) – list of most expensive functions
    - `-O calltree` / `callers` – top-up / bottom up function calls
    - `-O ca+src` – as above, with line numbers
    - `-O load balance` – displays min/mean/max across Pes
  - Each table in the report lists which options are needed to generate it:
    - e.g. `Table option:`

```
-O profile
Options implied by table option:
  -d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE
```

Tracing Experiments

- Implied options are a good starting point for customisation
  - See `man pat_report` for full list of options
- Each table also suggests options for related tables, and additional `pat_report` flags
- Also, check the ‘Observations and suggestions’ section

Tracing Experiments

See example files.

CrayPAT GUI

- CrayPAT includes a GUI called Apprentice2
  - Reads the portable `*.ap2` file format
  - Graphical view of the calltree
  - Chart views of selected data
  - Hardware counters, activity graphs
  - Application trace available by setting `PAT_RT_SUMMARY=0` before running your application
  - Warning – v. large trace files (MBs -> GBs!)
- Can be run directly from ARCHER via X-windows

```
app2 &
```

- Or binaries available for Mac & Windows

`/opt/cray/perftools/6.2.2/share/desktop_installers/`

CrayPAT GUI

See example files.

Using the CrayPAT API

- For even finer-graining tracing, CrayPAT provides an API to control tracing
  - Start/stop tracing at certain points
  - Define regions within (or spanning) subroutine calls

```
PAT_region_begin(1, “region name”)
PAT_region_end(1)
```

  - Also a Fortran API

- Build application, then instrument binary with

```
pat_build -w -o cp2k.psmp+api cp2k.psmp
```

  - May also include `-g mpi` etc.

Using the CrayPAT API

- Application code with CrayPAT API calls now depends on CrayPAT library
  - Will not build without perftools module loaded

- If including in production code, protect CrayPAT calls with preprocessor defines.

That's all folks!

Questions?

Further Help

- ARCHER Best Practice Guide:
  - http://www.archer.ac.uk/documentation/best-practice-guide/performance.php
- CrayDoc:
  - http://docs.cray.com
  - Search "Using Cray Performance Measurement and Analysis Tools"
- Online help:
  - Man pages for the tools
  - `pat_help` utility
- ARCHER training archive:
  - http://www.archer.ac.uk/training/courses/craytools/

Goodbye!

Thanks for attending

Please leave feedback at:
www.archer.ac.uk/training/feedback/online-course-feedback.php

archer

epcc

THE UNIVERSITY OF EDINBURGH