UNIVERSITY of York

An introduction to Profiling

Physics Coding Club: 09/06/2017

D. Dickinson (d.dickinson@york.ac.uk)

Overview

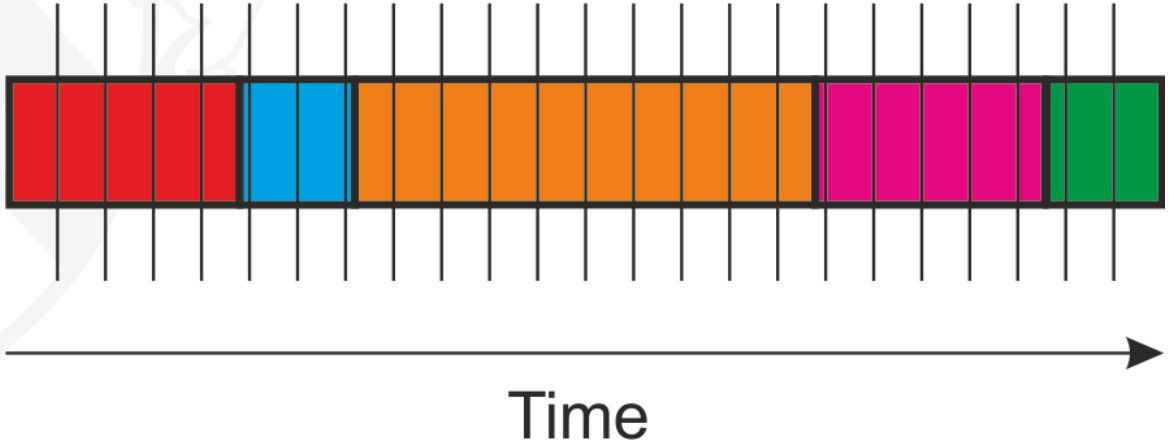
- What is meant by profiling?
- Why do we care about profiling?
- How do we do profiling?
  - Specific example using Scalasca
- Hands on session (if interested/working).

What is profiling?

- Essentially: *the process of measuring resource requirements of a program.*
- Often “profiling” refers to measuring time (or cycles) used by different sections of code.
- Can also measure memory requirements, I/O, communications etc.

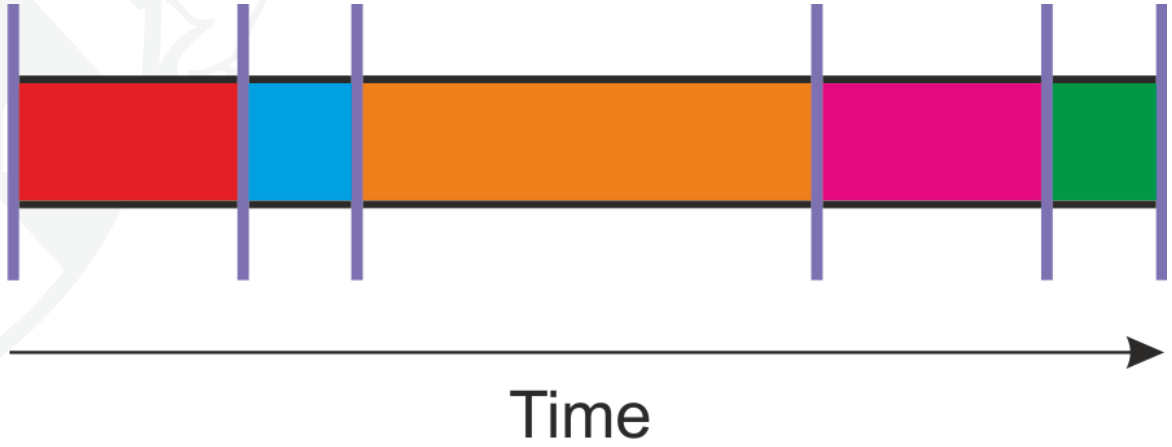
Types of profiling

- Several different types:
- Sampling : Interrupt and ask
  - Low overhead
  - Statistical approach → may need longer runs



Types of profiling

- Several different types:
- Instrument : Insert code to measure
  - Profile summarisation/Tracing
  - More detailed, have to watch out for overhead etc.



Types of profiling

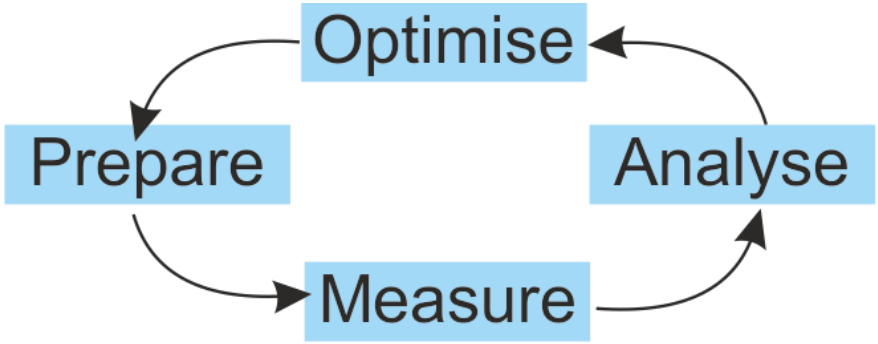
- Several different types:
- Sampling : Interrupt and ask
- Instrument : Insert code to measure
- Others available (e.g emulation/interception, event based etc.)
- Best choice depends on your aims, often a combination will be helpful.

Why profile?

- Generally most common reason is that you want to optimise resource usage of the code

  → Need to know where in the code dominant resource usage lives (i.e. what & where).

  → Need to understand cause of dominant resource usage (e.g. why).



Why profile?

- Generally most common reason is that you want to optimise resource usage of the code
- Can also be useful for other reasons:
  - Get overview of code path.
  - Look at how resource requirements scale (problem size, number of processors etc.)
  - Relative behaviour of different processes etc.
- **Better understanding of the operation of the code → more informed decisions about usage and development.**

How to profile?

- Can depend on which resources are of interest and the type of code (language, serial/parallel etc).
- Will briefly discuss *memory profiling* with **valgrind**, serial *cpu profiling* with **gprof**.
- Will have a more detailed demonstration of the *parallel* profilier **scalasca** which gives details of cpu and communication requirements (and possibly more).

Massif (Valgrind) – memory usage

- Massif is a *heap* profiler. It measures how much heap memory your program uses (can also measure the stack usage).
- Compile program with –g to ensure symbols available.
- Run prog as

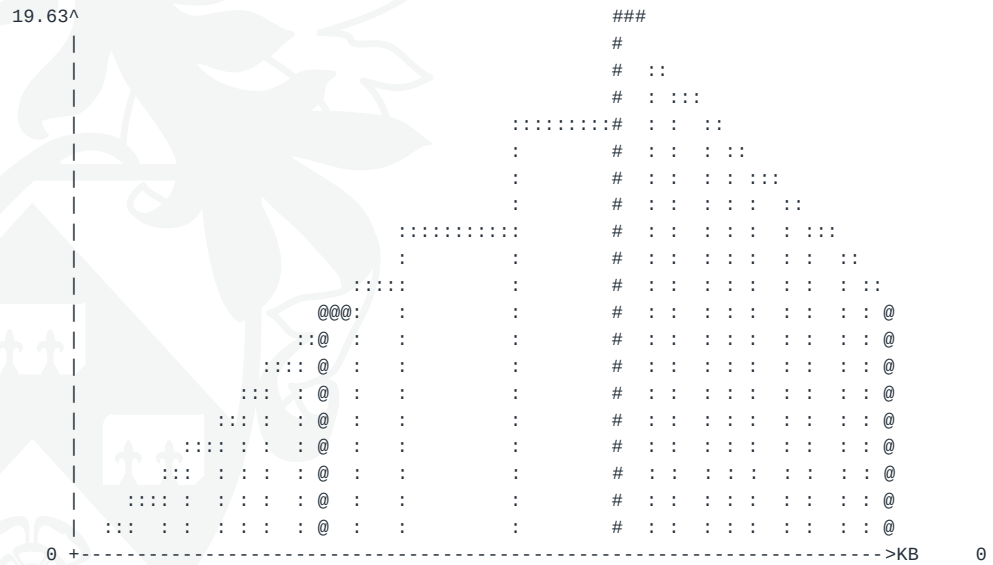
  >> valgrind --time-unit=B --tool=massif prog

- Results in file name massif.out.<pid> view with:

  >> ms_print massif.out.<pid>

Massif (Valgrind) – memory usage

- Will produce an ascii graph like



- Also some more detailed breakdown of where memory allocated.
- See http://valgrind.org/docs/manual/ms-manual.html .

Gprof

- Gprof is a performance analysis tool for capturing numbers of calls and time spent in routines. *(note actually two versions of gprof; gnu-gprof and "Berkeley Unix-gprof", little difference).*
- First must compile and link with profiling support, using gnu compiler family add '-pg' option to compile+link flags

```
gfortran -g -c myprog.f90 utils.f90 –pg
gfortran -o myprog myprog.o utils.o –pg
```

- Now run program myprog as usual (must exit cleanly). Produces gmon.out file.
- Can analyse with

```
gprof <options> ./myprog gmon.out > report.txt
```

Gprof

- Can produce a range of different outputs, including a flat profile/table like:

```
Flat profile:
     Each sample counts as 0.01 seconds.
       %   cumulative   self              self     total
      time   seconds   seconds    calls  ms/call  ms/call  name
      33.34      0.02     0.02     7208     0.00     0.00  open
      16.67      0.03     0.01      244     0.04     0.12  offtime
      16.67      0.04     0.01        8     1.25     1.25  memccpy
      16.67      0.05     0.01        7     1.43     1.43  write
      16.67      0.06     0.01                             mcount
       0.00      0.06     0.00      236     0.00     0.00  tzset
       0.00      0.06     0.00      192     0.00     0.00  tolower
       0.00      0.06     0.00       47     0.00     0.00  strlen
```

- See https://sourceware.org/binutils/docs/gprof/ .

Scalasca – Requirements

- Scalasca is a parallel profiler capable of measuring time, calls, communication (and other metrics) across a range of hardware (cpus, gpus, “novel” accelerator cards).
- Originally a standalone tool but with v2 now built on scorep instrumentation tool as well as the cube and otf analysis/format libraries.
  - More components to configure and compile.
  - More flexibility and compatibility (scorep underlies a number of different performance analysis tools).
- Often available on HPC systems.

Scalasca – Instrument

- First stage to using Scalasca is to ask it to instrument your code.
- Done by prefixing compiler command with ‘scalasca –instrument’ or ‘skin’:
  gfortran file.f90 –o file.o → skin gfortran file.f90 –o file.o
- Can detect if compilation is parallel (MPI/OpenMP), serial, on novel hardware etc.
- End result is just your normal executable.

Scalasca – Run (analyse)

- Now we have an instrumented executable we just need to run it for a (small representative) test case. Use the usual command but prefix with ‘scalasca –analyze’ or ‘scan’, e.g.

  scan mpirun –np 2 ./prog <options>

- Slight delay but then program will run as usual, produces a directory named something like scorep_prog_<np>_sum
- Contains several files including ‘profile.cubex’, could proceed to view this immediately, but…

Scalasca – Examine (explore)

- At this point raw data recorded. A lot of different things can be done now with this, often a could idea to do a little more analysis with ‘scalasca –examine’ or ‘square’:

  scalasca –examine –s scorep_prog_<np>_sum

- Produces ‘summary.cubex’.
- Now can use ‘cube’ to view + explore the derived data

  cube scorep_prog_<np>_sum/summary.cubex

Scalasca – Tips

- You've now got enough information to be able to use Scalasca to instrument, record and examine performance data, but some useful further tips.
- Instrumentation can introduce overhead → If the instrumented case is significantly slower than un-instrumented case then this is a worry.
- Can define a filter file which excludes routines matching given regex from instrumentation recording – used with '-f' option to scan (i.e. run time).

Scalasca – Tips

- Reported routine names can be ‘mangled’ – to enable demangling need to build scorep with libbfd support (provided by binutils) – need the libbfd headers. The command scorep-info config-summary reports features enabled or not.
- PAPI support enables recording hardware counters. Use papi_avail to report available counters. To record set the SCOREP_METRIC_PAPI env var,

  export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
- Often some limits for how many can record.

Scalasca – Tips

- To build a good filter file you can use
  scorep-score –r scorep_prog_<np>_sum |less
  To report which routines are responsible for the most recording. This will tell you time per visit/call as well → Filter out those near the top of the list with small time/call.
- Can pass the new filter to scorep-score to get an idea of how much the filter has reduced requirements without rerunning the main program.
- Can derive your own metrics in cube, possible to compare/merge etc. different runs using cube tools.

Resources

- General profiling and gprof : HPC course (http://www-users.york.ac.uk/~mijp1/teaching/4th_year_HPC/lecture_notes/Profiling.pdf)
- Archer led training sessions, see https://www.archer.ac.uk/training/ for upcoming and past courses (past course material typically available e.g. https://www.archer.ac.uk/training/course-material/2015/06/perfan_durham/).
- Valgrind::massif guidance at http://valgrind.org/docs/manual/ms-manual.html

Scalasca – Demo

#Login to yarcc: EITHER

wget http://www-users.york.ac.uk/~dd502/scalasca/test.txt

chmod u+x test.txt ; ./test.txt

#OR : Get the source code to GS2

svn checkout svn://svn.code.sf.net/p/gyrokinetics/code/gs2/trunk GS2_TRUNK

#Setup the modules

export MODULEPATH=$MODULEPATH:/opt/yarcc/Modules/physics/

module purge

module load gnu/6.3.0 openmpi/2.1.1 hdf5 NetCDF/4.4.1.1 NetCDF-fortran/4.4.4 scalasca

#Build with instrumentation

GK_SYSTEM=archer MAKEFLAGS=-IMakefiles make FC="scalasca -instrument mpif90" COMPILER=gnu-gfortran WITH_EIG= USE_NEW_DIAG= depend

<as previous with depend → -j gs2>

wget http://www-users.york.ac.uk/~dd502/scalasca/input.in

scan mpirun -np 2 ./gs2 input.in | tee OUTPUT