LProf

Version 1.7
December 2021
1 Introduction

MAQAO Lightweight Profiler (LProf) is the MAQAO module which allows to easily profile an application to detect hot functions and loops in two steps:

1) Data collection using sampling

LProf uses hardware counters to profile large-scale parallel applications (2000+ cores) with a very low overhead. It is also possible to provide a custom list of hardware counters to sample.

2) Data display

LProf output allows to quickly identify time-consuming functions and loops, observe the amount of time spent by the application between different categories (I/O, Runtime, etc…) and detect load balancing issues.
2 Running MAQAO LProf

2.1 Sequential Run Command

```
maqao lprof -- <application> [arg1 arg2 ...]
```

application’s name (or path if not located in the current directory)

application’s arguments, if any

2.2 Parallel Run Command

WARNING! Invocation of LProf in MPI has changed in recent versions of MAQAO.

MAQAO Version 2.4.4 and below: the command line begins with the MPI command.

```
mpirun -n <NB_PROCESSES> maqao lprof -- <application> [arg1 arg2 ...]
```

MPI launcher command

number of processes

From 2.4.5 version and higher: the `--mpi-command` option is required for interactive runs and `--batch-script`, for batch runs.

Interactive runs:

```
maqao lprof --mpi-command="mpirun -n <NB_PROCESSES>" \ 
-- <application> [args]
```

MPI launcher command

number of processes

Batch runs:

```
maqao lprof --batch-script=<jobscript> [--mpi-command="mpirun..."] \ 
[--batch-command=<submission command>] -- <application> [args]
```

Required only if jobscript extension (e.g "sbatch") is not recognized
In jobscript, application executable and its arguments have to be replaced by `<run_command>`.

```
$ cat jobscript.sh
...
mpirun -n 4 <run_command>  # instead of mpirun -n 4 <application> [args]
  # <mpi_command> <run_command>  # if mpi-command used
```

Since 2.12.0, you can (and must) inform Lprof about the maximum number of processes per node (if greater than 1), allowing it to set correct internal settings: `--maximum-processes-per-node`

Starting from 2.14.5, it is autodetected when missing but it is still recommended to set `--maximum-processes-per-node` if known and > 1.

### 2.3 Kernel samples exclusion

Since 2.12.0, kernel samples are not collected by default (recent Linux distributions do not allow this by default). To collect them:

If `sysctl kernel.perf_event_paranoid` returns 2 or more, this step must be performed first:

```
$ sudo sysctl -w kernel.perf_event_paranoid=1  # lost after reboot
$ sudo sh -c 'echo kernel.perf_event_paranoid=1 >>
/etc/sysctl.d/local.conf'  # persists after reboot
```

If `sysctl kernel.perf_event_paranoid` returns 1 or less:

```
$ maqao lprof --include-kernel ...
```
### 2.4 Options

To list all options along with their descriptions:

**maqao lprof --help**

<table>
<thead>
<tr>
<th>Name</th>
<th>Short Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--xp=</code></td>
<td>Specify the experiment directory</td>
<td>Directory’s name (string)</td>
</tr>
<tr>
<td><code>--mpi-command=</code></td>
<td>Specify command for interactive MPI run or replacement value for <code>&lt;mpi_command&gt;</code> in job script</td>
<td>Ex: “mpirun -n 4”</td>
</tr>
<tr>
<td><code>--ppn/maximum-processes-per-node=</code></td>
<td>Since 2.12.0, mandatory when using <code>--mpi-command</code> Optional but recommended starting from 2.14.5 if ppn &gt; 1</td>
<td>Ex on single node: lprof mpi-command=“mpirun -n 32” ppn=32</td>
</tr>
<tr>
<td><code>--batch-script=</code></td>
<td>Jobscript to submit to job scheduler</td>
<td>Path to jobscript (string)</td>
</tr>
<tr>
<td><code>--batch-command=</code></td>
<td>Command used to submit jobs, required if jobscript extension is not recognized. Currently recognized: .sbatch and .pbs</td>
<td>Ex: “sbatch”</td>
</tr>
<tr>
<td><code>--sampling-rate=</code></td>
<td>Number of collected samples per second</td>
<td>- highest (2000 Hz, btm=off recommended)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- high (1000 Hz, avoid btm=stack)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- medium (200 Hz, default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- low (50 Hz)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- lowest (10 Hz)</td>
</tr>
<tr>
<td><code>-ug=</code></td>
<td>Control (i.e. pause/resume) measurement via a</td>
<td>on (CTRL+Z)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or a delay in seconds</td>
</tr>
</tbody>
</table>
- **signal (Ctrl+Z)** or via a countdown

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ldi=</td>
<td>Scan debug information into all or specified (provided list) library(ies) to get loops details</td>
<td>on (all)</td>
</tr>
<tr>
<td>-btm=</td>
<td>Select backtraces (callchains) collection method</td>
<td>- <strong>fp</strong> (default, recompile application with -fno-omit-frame-pointer)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stack (higher overhead but no need to recompile application)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- branch (not really callchains but branch history, HW-dependent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>off</strong> (no callchains, lowest overhead)</td>
</tr>
</tbody>
</table>

### Advanced Options (collect step)

<table>
<thead>
<tr>
<th>Name</th>
<th>Short Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>--maximum-buffer-megabytes=</td>
<td>Allow to override Lprof memory footprint (default is 50 MB per CPU)</td>
<td>Maximum amount per node (Megabytes)</td>
</tr>
<tr>
<td>--engine=</td>
<td>Use another perf-events based sampling engine</td>
<td>- perf-low-ppn (selected by default when perf-events are available with max 4 processes per node)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- perf-high-ppn (selected by default when perf-events are available with more than 4 processes per node)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- no-perf (selected by default when perf-events are not available)</td>
</tr>
<tr>
<td>--evts=</td>
<td>Provide custom list of events to sample (CF maqao --list-events)</td>
<td>evt1_name@sample_period, ... or evt1_code@sample_period, ...</td>
</tr>
</tbody>
</table>
3 Display

The two common display modes are text (default) and HTML.

3.1 Text Output

3.1.1 Functions Hotspots

To display summary view:

```
maqao lprof -df xp=<EXPERIMENT_DIRECTORY>
```

```
hotspots' level: functions
experiment directory (name or path) containing data to display
```

![Figure 1 - LProf Output: Summary View (Functions)](image)

To display thread view:

```
maqao lprof -df xp=<EXPERIMENT_DIRECTORY> -dt
```

```
Thread ID | Hostname | Process ID | Walltime
----------|----------|------------|----------
```

![Figure 2 - LProf Output: Thread View (Functions)](image)
Loops Hotspots

To display summary view:

```
maqao lprof -dl xp=<EXPERIMENT_DIRECTORY>
```

![Figure 3 – LProf Output: Summary View (Loops)](image)

The above figure is truncated. In the actual output, four more columns are available on the right (same as functions mode):

Coverage (%), Time Min (s), Time Max (s) and Time w.r.t Walltime (s).

To display thread view:

```
maqao lprof -dl xp=<EXPERIMENT_DIRECTORY> -dt
```

![Figure 4 - LProf Output: Thread View (Loops)](image)
3.1.2 Generation of HTML results

This command generates an 'index.html' file into the `<EXPERIMENT_PATH>/html/` directory. Open this file into a web browser to see the results.

3.1.3 Interpretation of the Results
Refer to the Onview tutorial: [http://maqao.org/release/MAQAO.Tutorial.ONEVIEW.pdf](http://maqao.org/release/MAQAO.Tutorial.ONEVIEW.pdf)