

HPC-Europa: A Beginner's Guide to using Paraver on HPCx

Chris Johnson

February 23, 2009

1 Introduction

Paraver provides a large amount of functionality for analysing the behaviour of codes and in particular codes written to be run on multiple threads/processors. Using the functionality of Paraver you can uncover bottlenecks in your code and discover areas where the performance is not as good as expected. This is done by performing both visual and statistical analysis of the various calls the code makes during its runtime. From this information you may be able to make your code run more efficiently. In order to perform an analysis Paraver attempts to make the most complete use possible of the information created at runtime and as such collects a lot of information which would often be discarded once the run has completed. For example, on HPCx full use is made of the hardware counters provided on the machine.

However, the fact that Paraver provides so much functionality can initially make it quite a bewildering tool to use. In an attempt to overcome these difficulties we have provided this guide as a basic outline to show how to get started with using Paraver.

This intention of this guide is to provide information for users intending to use Paraver as a tool for profiling MPI codes on HPCx [Ltd02], although it is hoped the guide will be of more general use as most features will apply to other platforms where Paraver is supported. The main aim is to provide a set of useful hints for running Paraver without having to first read through all of the documentation or learn by trial and error.

2 Getting started

As is generally the case with profiling tools, the analysis consists of several stages. Firstly a trace has to be produced by running the code via another executable, secondly the raw trace files must be merged into one single trace file and thirdly the single trace

can be analysed using a visual tool. These three stages are independent and can be carried out at different times (or even on different machines).

The three stages are represented by three different executables installed on HPCx:

- 1 ompitrace (to create the trace).
- 2 ompi2prv (to merge the traces produced for each processor).
- 3 paraver (to view the final trace file).

Filters also exist between stages 2 and 3 which allow large traces to be cut horizontally (*ie* to select a subset of processor) or vertically (*ie* to produce a trace of a particular time interval). This is useful, and in fact often necessary, if the trace is large.

On HPCx all three of these executables are found in the following directory:

```
/usr/local/packages/paraver/bin/
```

which should be added to your PATH environment variable and it is also useful to set the environment variable OMPITRACE_HOME to this directory as well inside your batch script.

You will should also set your PARAVER_HOME to

```
/usr/local/packages/paraver/
```

in order to pick up the licence. This can either be done on the command line, or better still in your start-up scripts (*eg* .bash_profile and .bashrc).

Let us first concentrate on producing a trace.

2.1 Producing a trace [omptrace]

Paraver is a profiling tool, rather than a debugging tool, and as such will only provide useful analyses if used in conjunction with a fully working code which has not necessarily been optimised by other analysis. Once your code is fully debugged and tested you should be in a position to start creating a trace.

It is better to start with as small a trace as possible and so you should choose the dataset, number of iterations, etc. with this in mind. When compiling it is better not to use high levels of optimisation as this can confuse the analysis as the optimiser may in-line part of the code, unroll loops or change the names of functions in the binary executable. Ideally you should switch optimisation off or use an optimisation level no higher than -O3. If you encounter problems then turn the optimisation right down to -O0.

The two examples below show scripts run on HPCx which both produce a trace. The first example shows you how to produce a trace when running in the batch queue, the second when running interactively. You should note that the interactive example will

probably not produce meaningful results as you are sharing the node with other users but it can be a useful way of testing that you can produce a trace and seeing what MPI calls are made.

In order to run ompitrace you will need a `.rhosts` file in your home directory containing a list of every node on which your code is likely to run.

2.1.1 Paraver running in the batch queue

This job can be submitted using the `llsubmit` command.

```
#@ shell = /bin/ksh
#
#@ job_type = parallel
#
#@ cpus = 4
#@ wall_clock_limit = 00:20:00
#
#@ account_no = x01-fred
#
#@ node_usage = not_shared
#
#@ job_name = ping_pong
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue

export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export OMPITRACE_HOME=/usr/local/packages/paraver

$OMPITRACE_HOME/bin/omпитrace -v -r -nosw poe.real \
./ping_pong_paraver
```

2.1.2 Paraver running in the interactive queue

```
#@ job_type = parallel
#
#@ cpus = 4
#@ wall_clock_limit = 00:20:00
#
#@ account_no = x01-fred
```

```
#
#@ node_usage = shared
#
#@ class = inter32_1
#
#@ job_name = ping_pong
#@ notification = never
#
#@ queue
```

```
export MP_SHARED_MEMORY=yes
export OMPITRACE_HOME=/usr/local/packages/paraver
```

For this interactive example, you will need to execute a line similar to this:

```
$OMPITRACE_HOME/bin/omptrace -v -r -nosw poe.real
./exec -llfile ./batchscript
```

where `batchscript` is the name of the above script and `exec` is the name of your executable.

So what is the difference between this script and a normal script not using Paraver?

Firstly, we have set an environment variable:

- **OMPITRACE_HOME** This is where the Paraver tracing tool lives.

Secondly instead of calling “`poe ./ping_pong_paraver`” directly, we have added “`$OMPITRACE_HOME/bin/omptrace -v -r -nosw`” and changed “`poe`” to be “`poe.real`”.

So now let us look at the `omptrace` command itself.

<code>\$OMPITRACE_HOME/bin/omptrace</code>	<code>-v</code>	<code>-r</code>	<code>-nosw</code>	<code>poe.real</code>	<code>run.exe</code>
runs the paraver tracing tool	verbose	when compiled with <code>_r</code> option	no software clock	runs poe directly	the executable to be run

`-r` You should always compile using the re-entrant (`_r`) versions of the compilers, *ie* `mpxlf90_r` rather than `mpxlf90` and `mpcc_r` rather than `mpcc`, etc. so you will always need this option. You may not be able to produce a trace if you do not use the re-entrant compiler.

`-nosw` This switches off the software clock. Use this switch to get consistent times between nodes.

`poe.real` This must be called instead of `poe` which calls `poe.real` via another script which can not be understood by `omptrace`.

`-stdin:inputfile` Although not needed in this example, if you want to send input to the code via standard input, *ie* as you would using `do` using the ‘<’ symbol, then you will need to use this flag.

Typing

```
ompttrace -h
```

at the command line shows all the options available when running `ompttrace`.

In order to produce a trace, your executable does not have to have been compiled using any special flags, and you should not have to alter your code unless you want to add instrumentation to your code.

If you have problems producing a trace, please see the appendix 6 for help with diagnosing the problem.

3 Merging the raw trace files

After running your code through `ompttrace` you should receive some information in your output file which, if successful, will end with a line telling you to run a command similar to the following:

```
ompi2prv *.mpit -s *.sym -o some_code_outputXXXXXXXXX.prv
```

To create a trace you should run the command as instructed. If using the wildcard `*`, you will need to ensure that only the relevant `.mpit` and `.sym` files are present in the working directory, otherwise you must modify this command accordingly. The length of time it takes for this command to complete is an indication of how large the trace is. For larger traces you may need to run this command via a serial batch job.

This should produce a trace with a `.prv` extension which can be viewed using `Paraver`. A file with a `.pcf` extension should also be produced.

If the trace created is very large (*eg* of the order GBytes) then you may well consider cutting the trace (see next section).

4 Cutting the trace

The executable `cutter` lives in the same bin directory as the other executables. It allows you to cut a trace horizontally (*ie* to select a subset of processor) using the `-threads` flag or vertically (*ie* to produce a trace of a particular time interval) using the flags `-t` for absolute time or `-p` for a percentage of time. The available options can be found by running `cutter -help`.

For example, for a trace created on 32 threads:

```
cutter trace1.prv trace2.prv -p 0 100 -threads 1-2
```

gives you the whole time period for just 2 threads.

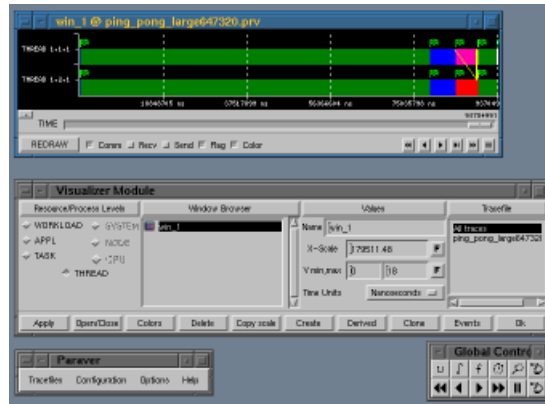


Figure 1: The initial screen.

`cutter trace1.prv trace2.prv -p 0 10 -threads 1-32`
 gives you the first 10 % of the trace for all 32 threads.

5 Viewing the trace

5.1 Running Paraver

The facility to view the trace you have just produced is known as “Paraver”. In order to run this you will need to have your PATH set correctly. The paraver executable lives in `/usr/local/packages/paraver/bin/paraver`. So assuming you have successfully created a trace, to view this you simply need to type

```
paraver tracefile.prv
```

where `tracefile.prv` is the name of your Paraver trace.

You should then see three windows, looking something like those in figure 1.

6 Commonly encountered errors

This appendix contains a list of the errors sometimes encountered when trying to create a trace. In most cases the output is shown followed by the error which is received to the standard error. Then a suggestion is given as to what may have caused the problem.

- At any point you may receive this error:

```
bash-2.05a$ paraver &
bash-2.05a$ connect: Connection refused
```

```
License not found. Needed a license for: HOSTID 0xac1f6433 SERIALID
0xd60b4c00 Version 3.3
```

This indicates a missing or incorrect licence file or it may be that the licence server is not running. In this case you should submit a query to support@hpcx.ac.uk. You can check if the licence daemon (paraverd) is running by typing:

```
bash-2.05a$ ps -fu paraver
      UID      PID      PPID      C      STIME      TTY      TIME  CMD
paraver 200964          1      0      Dec 06          -      0:00
/usr/local/packages/paraver/bin/paraverd
```

- ERROR: No more licenses available.
All licenses have been granted.

It may be that another user is using Paraver or that the system has not realised that your last session has finished. In the latter case simply trying again in 5 minutes should work.

-

OMPitrace tool (Version 1.2)

Tracing application: ./hello

Tracing Parameters:

- * Local clock
- * Tracing counters: user,parallel,calls,nompi

Parameters: 0: /usr/bin/poe

1: /hpcx/home/z001/z001/chrisj/helloworld/Fortran/./hello

-> Creating the target application <-

```
bash-2.05a$ more hello.11f51.60501.err
```

```
* Error: Appl.bcreate failed. bcreate status: ASC_failure
* Error: ompi_DPCL.C (3831)
*
```

This is due to poe being called directly, rather than calling poe.real. Replace poe with poe.real.

- OMPitrace tool (Version 1.1)

Tracing application: ./hello

Tracing Parameters:

- * Switch clock
- * Tracing counters: user,parallel,calls,nompi

Parameters:

```
0: /usr/bin/poe
1: /hpcx/home/z001/z001/chrisj/helloworld/Fortran/./hello
```

```
-> Creating the target application <-
bash-2.05a$ more hello.14f41.11729.err
```

```
* Error: Appl.bcreate failed. bcreate status: ASC_failed_rhost_check
* Error: ompi_DPCL.C (3299)
*
```

This is due to no .rhosts file being present in your home directory or the .rhosts file being incorrect - for example you may have hidden “ctrl-M” characters in your .rhosts file if this file was created in, or has been moved via, MS Windows. These “hidden” characters can be seen by opening the file in vi. They can also be removed in vi or converted using freely available dos2unix script.

- or when running interactively ...

```
-> Creating the target application <-
ERROR: 0031-808 Hostfile or pool must be used to request nodes
```

```
* Error: Appl.bcreate failed. bcreate status: ASC_terminated_pid
* Error: Permission denied
* Error: ompi_DPCL.C (3831)
*
```

This is due to no .rhosts file being present.

- At any point you may receive this error:

```
bash-2.05a$ paraver &
bash-2.05a$ connect: Connection refused
License not found. Needed a license for: HOSTID 0xac1f6433 SERIALID
0xd60b4c00 Version 3.3
```

Missing or incorrect licence file.

- * Error: PROBE init or PROBE fini haven't been installed.
Application can't be traced.
* Error: ompi_DPCL.C (4142)
*

This is usually due to linking with very aggressive options such as optimisation options.

7 .rhosts file

You should put a file called `.rhosts` in your home directory with the following entries (one per line).

References

[Ltd02] UoE HPCx Ltd. The hpcx website. Website, 2002. <http://www.hpcx.ac.uk>.