Debugging and Optimization of Scientific Applications - Exercises

26-28 October 2015, Cineca, Bologna.
PATC course

Objectives

The aim of the practical session is to introduce some of the tools available for debugging, tracing or profiling application codes. Students are invited to use their own applications for experimenting with these tools but we have also provided some examples.

Practical details

login to Eurora:

For this course we will use the Eurora cluster, located in Cineca. You log in to Eurora via ssh as follows:

```
ssh -l username login.eurora.cineca.it
```

Username and Password: Available from demonstrator
The username and password will be valid for the whole duration of the course plus a few days (ask the demonstrator for details). Each student is responsible for the use of the username assigned.

Training budget

The demonstrator will also communicate the budget ("account number") used for course. This must be inserted in the PBS batch scripts with the -A and -W flags. For example, if the account number is **train_cnov2015** then you must include:

```
#PBS -A train_cnov2015
#PBS -W group_list=train_cnov2015
```

Example applications and input

```
wget
https://hpc-forge.cineca.it/files/CoursesDev/public/2015/Introduction_to_HPC_Scientific_Programming:_tools_and_techniques/Bologna/exercises.tar.gz
tar zxvf exercises.tar.gz
cd exercises
```

Interactive PBS session:

```
qsub -l select=1:ncpus=4:mpiprocs=4,walltime=20:00 -A train_cnov2015
-W group_list=train_cnov2015 -I
```

Submitting a job script

```
qsub example.job
```

Example PBS job script:

```
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -N myjob
#PBS -o job.out
#PBS -e job.err
#PBS -W group_list=train_cnov2014
#PBS -A train_cnov2015

cd $PBS_O_WORKDIR
module load autoload openmpi
mpirun ./myexecutable
```

Remote Visualisation

Many of the profiling and debugging tools require a graphical client. Since graphics via X Windoes can be quite slow we recommend you download and install the RCM client for these exercises (available for Windows, Linux and MAC):

http://www.hpc.cineca.it/content/remote-visualization-rcm

Suggested Exercises

1. Performance Analysis of a program (e.g. DL_POLY) with Scalasca
2. Trace profiling with
    a. mpirun -trace
3. Debugging a program with Totalview.

Exercise 1. Performance analysis of a program with SCALASCA

Description

The molecular dynamics program DL_POLY (ref) comes in two versions:

1. Version 1.9 ("Classic") which implements a simple, replicated data approach for parallelisation;
2. Version 4.x which implements domain decomposition for the parallelism

The second version shows a much higher performance and parallel scalability than the first (which is in fact no longer used in production).
With Scalasca we can probe the reasons for this in terms of the program code and the MPI calls used.
Note that:

- The following describes the procedure for Scalasca versions 1.x whereas the current version is 2.x. The commands used for the latter version are slightly different but otherwise the analysis is identical.
- If you want to use your own code for the analysis just replace the references to the DLPOLY source and compilation with your own source code and Makefile.

Step 1. Download, copy or generate the source

For example, if we are using the DL_POLY example, go into the build directory:

```
cd ex1
```

Alternatively skip this and the next step and use directly the compiled version for DLPOLY:

```
module load dl_poly/1.9
exe=$(which DLPOLY.X.scalasca)
```

Step 2. Re-compile with the scalasca compiler wrapper.

```
cd dl_class_1.9
cp build/MakePAR source/Makefile
```

Edit Makefile:

```
$(MAKE) FC="skin mpif90" LD="skin mpif90 -o" \

module load profile/advanced
module load autoload scalasca
cd source
make intel
```

Step 3. Run program with scalasca

In a suitable directory copy the input files for DL_POLY

```
mkdir runs
cp input/* runs
```

```
cd runs
```

and create a batch job (e.g. job.pbs) with the following:

```
cd $PBS_O_WORKDIR

module load profile/advanced
module load autoload scalasca

# the path to the DL_POLY executable
exe=$HOME/dl_class_1.9/execute/DLPOLY.X

scalasca -analyze mpirun -np 4 $exe
```

Step 4. Run the batch job

```
qsub job.pbs
```

It should complete in a few minutes.

Step 5. Analyse the results directory

```
scalasca -examine epik_DLPOLY_4_sum
```

You can also get textual output without running the viewer with the following command:

```
scalasca -examine -s epik_DLPOLY_4_sum
```

Step 5. Compare with DLPOLY 4.x

Repeat Steps 3-4 of the above procedure but with DL_POLY version 4.04. We note that we do not provide the source code but a scalasca version is available via a module.

```
module load dl_poly/4.04
exe=$(which DLPOLY.Z.scalasca)
```

Discussion

1. Poorly scaling programs often show load imbalance between different processes. Which version shows the worst load balance and on which process (i.e. MPI rank) is this imbalance? Where in the source code is this found?
2. Parallel performance can be strongly influenced by MPI collective communications. What is the most commonly used collective call (in terms of % time) for both versions of the program?
3. Which version uses MPI-IO for file writing?

Exercise 2. Trace profiling with IntelMPI and ITAC

Step 1. Re-compile program with IntelMPI

The program to be analyzed needs to be compiled with InteMPI so if using the DL_POLY2 example of Exercise 2 you must copy the source code and re-compile it.
(the module version has been compiled with OpenMPI).

```
cd  dl_class_1.9/source
make clean
module load  autoload intelmpi
make intel
```

Step 2. Copy the input files and prepare a batch job

Run the newly compiled program with PBS, after copying the input files.

```
cd ex2
cp ../input/* ex2

#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -l walltime=0:30:00
#PBS -q private
#PBS -A <account_name>
#PBS -N jobname
#PBS -W group_list=train_cnov2014

exe="../ex1/dl_class_1.9/execute/DLPOLY.X"
module load autoload intelmpi
source $INTEL_HOME/itac/8.1.3.037/intel64/bin/itacvars.sh
mpirun -trace $exe
---------
qsub job.pbs
```

This procedure should create a series of files called DLPOLY.x.stf.*

Step 4. Run batch job and analyze trace file

After the run, analyze the trace with the traceanalyzer GUI.

```
source $INTEL_HOME/itac/8.1.3.037/intel64/bin/itacvars.sh
traceanalyzer  DLPOLY.X.stf
```

Discussion

How do the result of this analysis compare with Scalasca? (e.g. what is the most time-consuming MPI command?)

Exercise 3. Debugging a program with totalview

For this exercise you must use RCM (recommended) or a TurboVNC client with SSH tunnelling (an X environment alone will not work).

Step 1. Create a directory and copy the program files or use the example:

```
cd ex3/poisson_training
```

Step 2. Compile with a suitable compiler. Make sure you have specified the -g flag in the Makefile

```
module load autoload openmpi
make
```

Step 3. Launch a PBS job (see above) with these command lines:

(check the DISPLAY variable before launching qsub)

```
export DISPLAY=node97:8
cd $PBS_O_WORKDIR
module load totalview
# use this line for openmpi
totalview -np 4 ./poisson.exe
# this one for intelmpi
# mpirun -n 4 -tv ./poisson.exe
```

Try and find the program line causing the deadlock.