Exercises for debugging and optimization of Scientific Applications

27-30 November 2016, Cineca, Bologna

PATC course

Contents

Objectives

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

Using Cineca's HPC facilities

Login

The HPC system to be used for this course will be the **Galileo Supercomputer**. To login to this computer you can use the following command:

```
ssh –l username login.galileo.cineca.it
```

The username and password will be provided to you by the demonstrator.
For more information on the Galileo computer please see Cineca's user guide.

https://wiki.ugov.it/confluence/display/SCAIUS/UG3.2%3A+GALILEO+UserGuide

Download and preparation of tutorial files

To download the tutorial files directly on Galileo use the following command:

```
wget
https://hpcforge.cineca.it/files/CoursesDev/public/2016/Introduction to H
PC Scientific Programming: tools and techniques/Bologna/exercises.tar.gz
```

Unpack the file with the tar command:

```
tar zxvf exercises.tar.gz
```

Use of the PBS Batch system

For all but very short runs or compilations you should the PBS batch system which can be used in either in interactive mode or in batch mode. In order to use PBS at Cineca you must specify a valid ***account number*** (i.e. budget). Please ask the demonstrator for the budget to be used in this course and use this whenever you see <account> in the code templates or the instructions below.

Interactive PBS mode:

```
qsub -l select=1:ncpus=4:mpiprocs=4,walltime=20:00 -A <account> -I
```

Submitting a batch job:

```
qsub example.job
```

Example PBS job script (for MPI programs).

```
#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=<account>
#PBS -A <account>
cd $PBS_O_WORKDIR
module load autoload intelmpi
mpirun ./myexecutable
```

Remote Visualisation

Many of the profiling and debugging tools require a graphical client. Since graphics via X Windows 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

Important VTUNE Note: Please note that we have only a limited number of licenses for Vtune so if you are not able to use the program please do another exercise until a license becomes available.

Exercises

1-Practice with the Galileo HPC system.

The aim of this exercise is to just get familiar with the Galileo cluster, the PBS batch system and the Cineca environment in general and consists of different tasks which can be performed in any order. We have given some simple examples but if you have your own application feel free to try it out.

a. **Simple OpenMP and MPI programs**. In the **MPI-OMP** directory there are some simple OpenMP and MPI programs for both C and Fortran. We suggest you compile and run them under PBS with different numbers of threads and tasks (many of these programs were taken from John Burkardt's Homepage -see [1] for more information).
b. **MPI parallel scaling of DL_POLY**. As described later () this application exists in two versions with very different scaling behaviours. Sample input (valid for both versions) is given in the **input** sub-directory. You need to copy the input files in the run directory before running the job. The aim is produce strong scaling curves for both versions. In the profiling exercise we will attempt to explain the difference in scaling behaviour.

2-Optimisation and Vectorisation

Memory bandwidth and cache use with Stream and Triad benchmarks

In this exercise we will take the stream benchmark from ….

Vectorisation examples

Problem 1

In this directory you will find various Fortran and C programs named test1.f, test2.c, etc. BEFORE compiling them try to understand if they will vectorise or not. (Note that for the C programs you may have to specify the `-std=c99 flag`).

Problem 2

You will find a file vecadd.f90 or vecadd.c which does a simple vector addition. Compile the program with and without vectorisation and in single and double precision:

```
ifort -no-vec -o no-vec vecadd.F90

ifort -O2 -o vec vecadd.F90
```

Or for C:

```
icc -O2 -no-vec -o no-vec vecadd.c -lm

icc -O2 -o vec vecadd.c -lm
```

Depending on the time you have try the vector size n with these values, n=10,100,500,1000 etc.

Discussion

Thinking of caches, can you explain these results ? (HINT: For Haswell caches L1=64K, L2=256K, L3=8Mb)

3-Performance Analysis and profiling

In this exercise we will try different profiling and performance tools: SCALASCA, ITAC and VTune. The tasks can be done in any order.

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. On Galileo we already have versions of the two programs compiled with Scalasca. If you wish to use your own application then ask a demonstrator or see the lecture notes.

Step 1. Run program with scalasca

In a suitable directory copy the input files for DL_POLY

```
cd ex3

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

module unload openmpi ! not needed for scan and prevents dl_poly loading

module load dl_poly/1.9

exe=$DL_POLY_HOME/bin/DLPOLY.X.scalasca

scan mpirun -np 4 $exe
```

Step 2 . Run the batch job

```
qsub job.pbs
```

It should complete in a few minutes and create a sub-directory `epik_DLPOLY_4_sum.`

Step 3. 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 4. Compare with DLPOLY 4.x

Repeat Steps 3,4 of the above procedure but with DL_POLY version 4.04.

```
module load dl_poly/4.04

exe=$(which DLPOLY.Z.scalasca)
```

Discussion

1. What is the most time-consuming part of the code (in each version)?
2. 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?

3. 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?
4. Which version uses MPI-O for file writing?

Trace profiling with IntelMPI and ITAC

A quick way of getting profile information is with the Intel tool ITAC (Intel Trace Analyzer and Collector). This is convenient because the program does not need re-compiling but it only works with IntelMPI.

To use ITAC you should create a PBS script with the following lines (example DL_POLY).

```
module load autoload dl_poly/4.04

exe=$(which DLPOLY.Z)

source $INTEL_HOME/itac/9.0.2.045/intel64/bin/itacvars.sh

mpirun -np 4 -trace $exe
```

Once the run has finished you can then analyse the trace data (outside of PBS if you like):

```
traceanalyzer DLPOLY.Z.stf
```

Discussion

1. How much time is spent in MPI calls and how much in non-MPI routines? Which MPI call consumes the most time.
2. How is the computational load shared amongst the MPI processes?

Node Performance analysis with Vtune

Vtune is a sophisticated tool for performance profiling of serial, MPI, OpenMP or hybrid programs within a computational node.

Usage

If using RCM and you wish analyse a serial program or an OpenMP program with a **small** number of threads, the easiest way is to run vtune directly on the RCM login node. Otherwise submit a PBS session in interactive mode and set the DISPLAY accordingly (for example):

```
qsub -l select=1:ncpus=4,walltime=30:00 -A <account> -I

export DISPLAY=node168:6

module load autoload vtune

vtune&
```

Note that if using vtune in PBS then the only analysis available is *advanced-hotspots*.

Alternatively, run vtune in command line-mode under PBS, exit PBS and visualise the results with vtune on a login node:

```
amplxe-cl -collect advanced-hotspots -r results-dir -- ./fstream
```

Serial or OpenMP programs

For this exercise you can use the stream benchmark compiled with OpenMP. For example,

```
cd ../ex2/stream
```

in makefile `set FFLAGS=-g –O2 -qopenmp`

```
make clean
```

```
make
```

```
cp fstream ../ex3
```

To run vtune open an interactive session with PBS and create a script which runs the application. Then load the vtune module and within the graphics interface you then need to select the option **New Analysis -> Advanced Hotspots**.

Under the tab Project Properties select the application executable script and hit Start.

Discussion

For OpenMP programs you should focus on areas in Red in the main summary which indicate problems such as high ***CPI*** rate (Clock cycles per Instruction) and particularly ***SPIN*** rate, which indicates bottlenecks or periods in the code where the CPU is idle (e.g. due to OMP_CRITICAL or other barriers).

MPI

Vtune is not really the best option for MPI programs but it is possible to analyse the performance one rank at a time. In this case you should run vtune via the command line in PBS. For example,

```
module load autoload intelmpi/5.1.1—binary

module load  vtune

mpirun -n 4 -l amplxe-cl -quiet -collect advanced-hotspots -result-dir
results ./DLPOLY.Z
```

This will create a results directory for each MPI rank. You can then analyse each MPI rank with the vtune graphics interface:

```
vtune&
```

Select Open->Result and look for the .amplxe file

For more information see https://software.intel.com/en-us/node/544016.

4-Debugging with Totalview

Note that this exercise is best done using the RCM remote visualisation client.

Procedure:

1. `cd  ex4/poisson_training.`
2. Load an MPI module (**but not intelmpi/2017--binary**) and compile with the makefile provided.
3. Run the program with 2 and more than 2 tasks. What happens?
4. Try to find the problem with the totalview debugger: `mpirun -tv -a mpirun -n 4 ./poisson.exe`

References

1. John Burkardt Homepage, https://people.sc.fsu.edu/~jburkardt/f_src/prime_openmp/prime_openmp.html
2. NERSC, http:// http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/