EXERCISE 2
GETTING STARTED

Objectives

The purpose of this exercise is to :

- Learn how to get started with a simple code using Allinea MAP
- Gain knowledge of Allinea MAP GUI to optimize simple performance issues.

Description

The code "slow" is a simple MPI code written in Fortran. A deep knowledge of Fortran is not required. This program contains 2 routines :

- a simple multiplication calculation
- a simple square root calculation.

Those routines are correct but they are really terrible from a performance perspective ! Using Allinea MAP, we will improve the performance of those examples.

Walkthrough

First, we will need to use the workload scheduler in order to start the code. Please do not run any "make" or "mpirun" command on the front-end node!

To submit a job to the compute nodes, please use the following command:

```
$ oarsub -I
```

This will give you direct access to a compute node. From here, you can anything !

Let's compile the application cstartmpi. There is a makefile for this in the cstartmpi directory. Please not that the "-g" option is necessary to generate debugging informations.

```
$ cd exercise2

make
```

If you run this application with 4 processes, everything is fine:

```
mpirun -n 8 ./slow
```

After a 2 or 3 minutes, the job should finish.

Everything looks good - apart from the fact that this run is quite slow ! Let's see if there is anything we can improve by using Allinea MAP.

```
$ map -profile -n 8 ./slow
```

This command will run the program in the background to the end and will generate an output file called "slow_8p_[date].map". You can now open this file with the GUI of Allinea MAP and see what happened.

```
$ map ./slow_8p_[date].map
```

The Allinea MAP GUI will appear - and will enable the analysis of your program post-mortem. Note that this can be done on a 1 core allocation - no need to run on several nodes to open this file. With the right environment and setup, you could even do this locally on your laptop !

The GUI of Allinea MAP is divided in 3 parts :

- **Timeline** – this graph represents the behavior of your program over time on different metrics
- **Source code viewer** – we want to relate bottlenecks to specific lines of code to make it easy !
- **Lower tabs** – which contains several tabs :
    - Input/Output : what is printed in the standard or error output for instance
    - Project files : allows you to visualize the most time-consuming files and functions regardless of the call stack
    - Parallel stack view : shows the most time-consuming branches of your call stack

In Allinea MAP, there is some easy color codes to remember :

- **green** is **CPU** activity
- **blue** is **MPI** activity
- **red** is related to **memory**
- **orange** is related to **disk I/Os.**

Illustration 1: Allinea MAP display after you opened a .map file

The screenshot shows Allinea MAP after a performance file has been opened. Let's focus on the first half of the run.

We can easily notice that the "CPU Floating-point" metrics is quite low - which is not something we would expect from an HPC application. What does this relate to ? Why are the Flops so low ? What is my application doing ?

> ***On the timeline, select a time slice with a very low "CPU Floating point" value.***
>
> ***Allinea MAP will immediately redirect you to the lines of code this low performance is related to, on lines 11 (where the function is called) and 50 in your source code.***

At this stage, we know what part of the application is slow. We also know that 100% of the time is spent here is CPU : the indications next to the lines of codes are "green".

Nonetheless, we don't know "why" the code is behaving badly yet. To answer this question, let's review other metrics provided by Allinea MAP.

> ***Next to the timeline, do a right-click and select other metrics that could be helpful to understand what is going on.***

Illustration 2: After selecting the first half of the run duration and expanding the Parallel Stack View, Allinea MAP shows the relevant lines of code.

The different presets "MPI" or "I/Os" show values close to 0. Those won't help in this particular case. But if we look at the "CPU Instructions" or "Memory" presets - we learn something useful : the memory accesses are quite high !

What does this mean ? Having this information in mind, can you fix the problem ?

Exercise

The first half of the program is now fixed and after making sure we access our array "a()" using indexes in the right order, we have been able to speed up the first half of the execution quite drastically. It was running in approximately 90s initially and we managed to reduce this duration down to 35s ! We have divided the run duration by 3 !

There is still a serious issue on the second part of the application though. It looks like the amount of time spent in MPI is quite high.

Using Allinea MAP, can you figure out what the problem is ?