M2-BIG DATA
High Performance Computing
Profiling

Lab 1

Réalisé par :
Gaby Maroun

Encadré par :
Dr. Etancelin JM

16 octobre 2020

Explore 3 ways of profiling a code on the Bilbo cluster :

1 How to use the cluster :

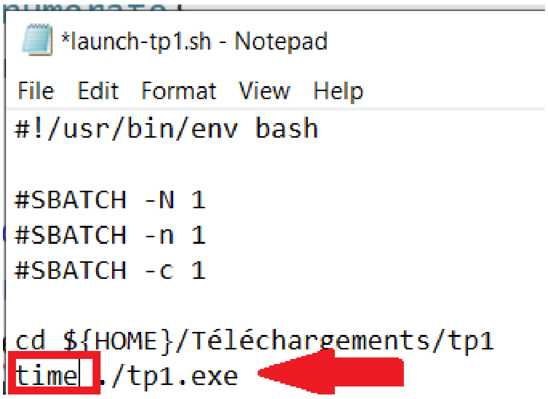
1. Login to cluster (x2go or ssh)
2. Prepare the codes, compile, explore results,
3. Submit jobs on to the resource manager (SLURM).

See more detailed information on the course page.

2 System profiling

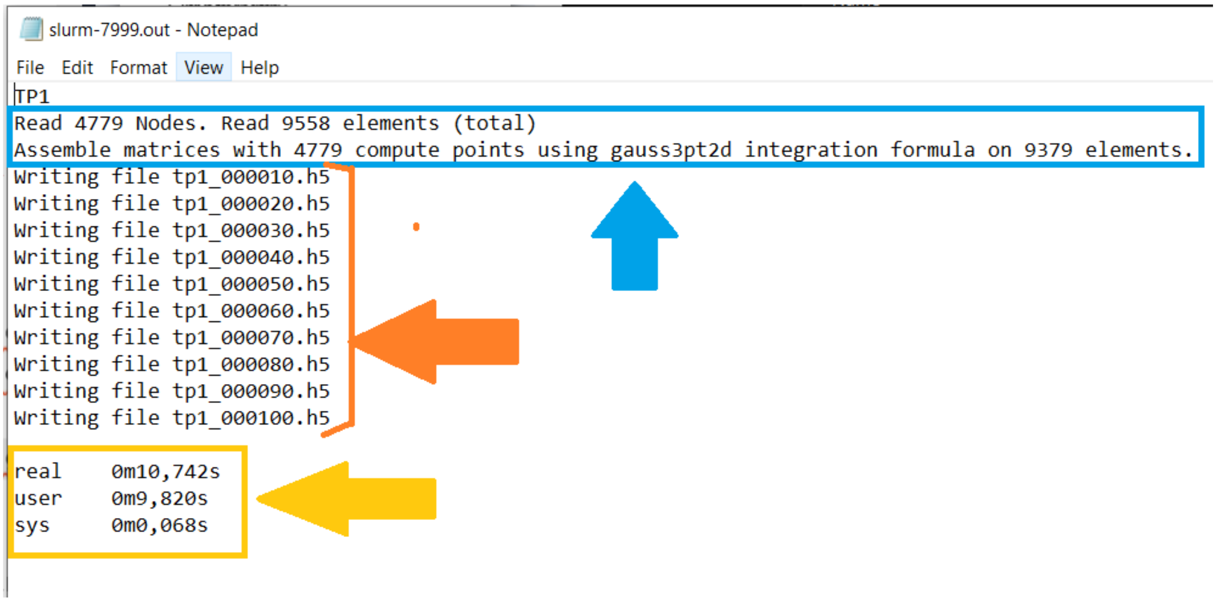
Get the sources of the program from the course page. It is better to copy the archive file and extract on the cluster tar xvf tp1.tar.gz

1. Build the code : **cd tp1** then **make**
2. Launch the code on computing resources : **sbatch launch-tp1.sh** The file **launch-tp1.sh** is a submission script containing resources need description and commands to launch on ressources. For example :



```
#!/usr/bin/env bash

#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1

cd ${HOME}/Téléchargements/tp1
time ./tp1.exe
```

3. Adapt your submission script in order to run the program through the system command time. Explore results and output in the file slurm-NNNNN.out and comment your results.

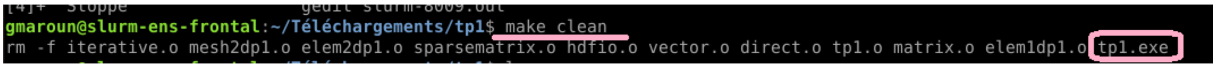


```
TP1
Read 4779 Nodes. Read 9558 elements (total)
Assemble matrices with 4779 compute points using gauss3pt2d integration formula on 9379 elements.
Writing file tp1_000010.h5
Writing file tp1_000020.h5
Writing file tp1_000030.h5
Writing file tp1_000040.h5
Writing file tp1_000050.h5
Writing file tp1_000060.h5
Writing file tp1_000070.h5
Writing file tp1_000080.h5
Writing file tp1_000090.h5
Writing file tp1_000100.h5

real	0m10,742s
user	0m9,820s
sys	0m0,068s
```

In the BLUE part, we can see the amount of nodes and elements read, and the amount of compute points using gauss3ptd2d integration formula to assemble matrices on the 9379 elements.
In the ORANGE part, a list containing the written in files ".h5".
*In the YELLOW part, we see the result of the **time** command represented by its three parameters real, user and sys. As of the precedent results, we can assume that the system or CPU took 0,068s to execute the code from the c file while in reality it took 10,742s.*

3 Gprof profiling

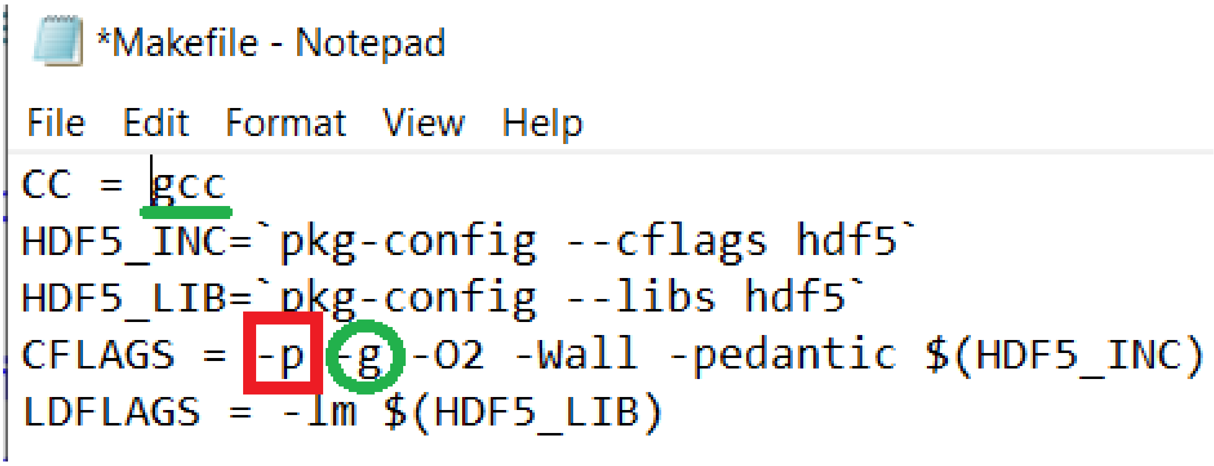
1. Clean the sources before re-compiling : **make clean**



```
[4]+  Stoppé                gedit slurm-8009.out
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ make clean
rm -f iterative.o mesh2dp1.o elem2dp1.o sparsematrix.o hdfio.o vector.o direct.o tp1.o matrix.o elem1dp1.o tp1.exe
```

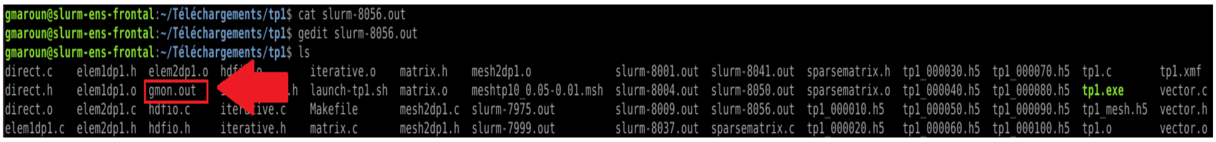
*After the cleaning, the tp1.exe won't be found anymore so we use the **make** to recreate it.*

2. Recompile the code using the **gcc** profiling option **-p** (check that the **-g** option is also present) by modifying the CFLAGS variable in the **Makefile**



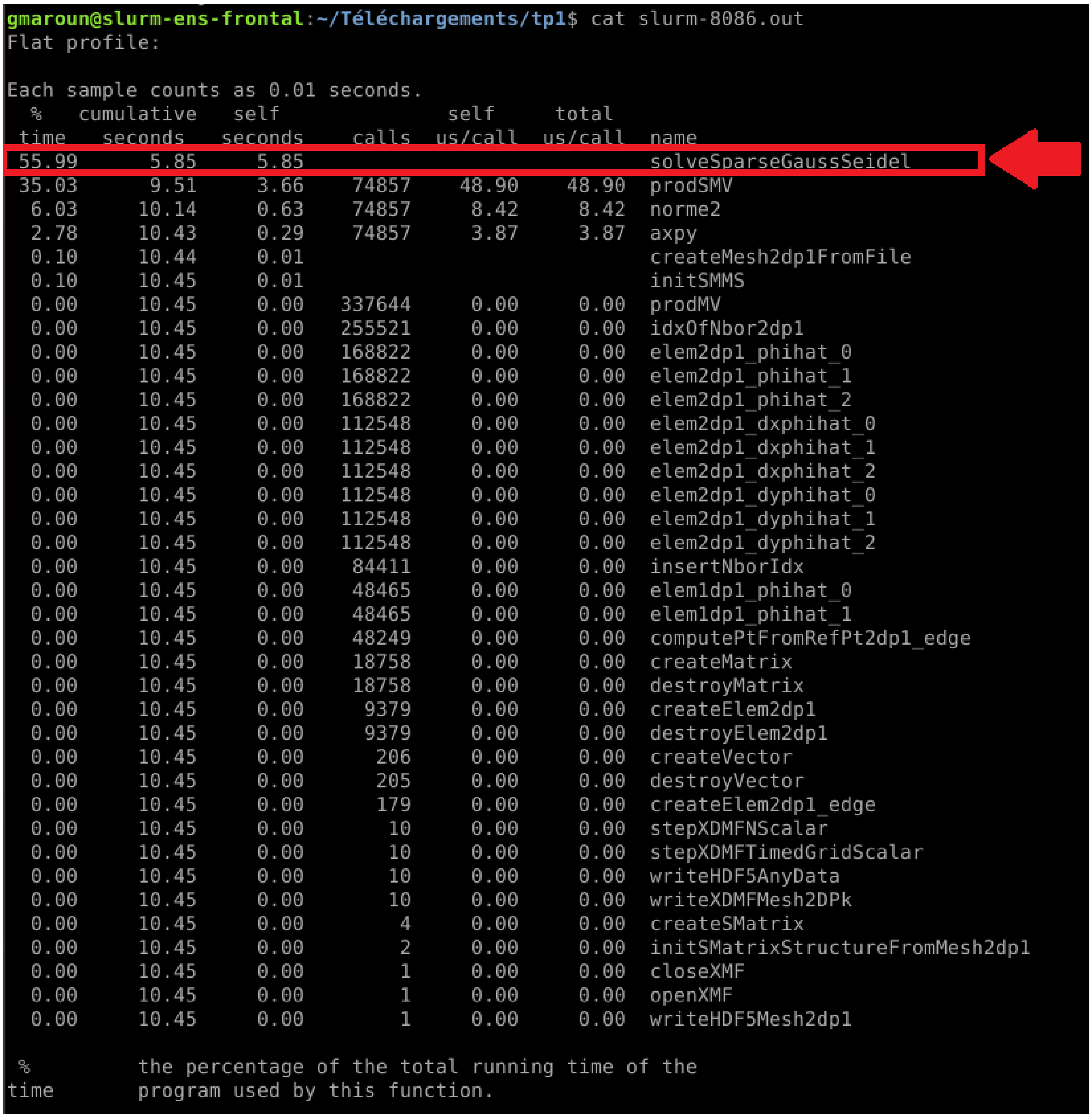
```
CC = gcc
HDF5_INC=`pkg-config --cflags hdf5`
HDF5_LIB=`pkg-config --libs hdf5`
CFLAGS = -p -g -O2 -Wall -pedantic $(HDF5_INC)
LDFLAGS = -lm $(HDF5_LIB)
```

3. Re-launch your execution script without the use of time command and explore the output. Check that the file **gmon.out** has been created properly

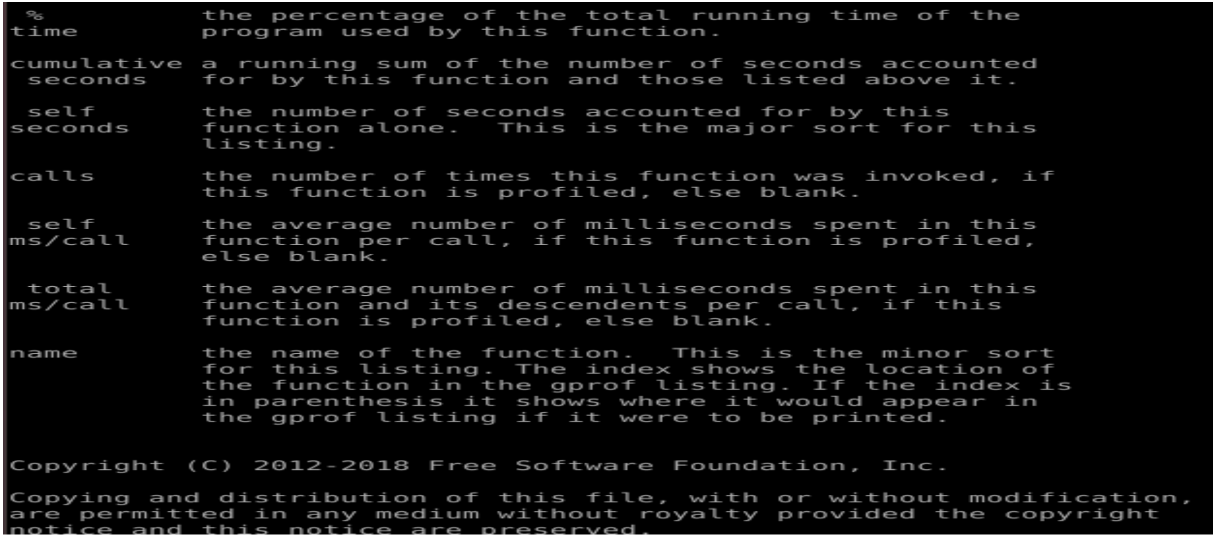


```
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ cat slurm-8056.out
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ gedit slurm-8056.out
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ ls
direct.c    elem1dp1.h  elem2dp1.o  hdfio.o      iterative.o    matrix.h    mesh2dp1.o               slurm-8001.out  slurm-8041.out    sparsematrix.h  tp1_000030.h5  tp1_000070.h5  tp1.c        tp1.xmf
direct.h    elem1dp1.o  gmon.out    [illegible]  launch-tp1.sh  matrix.o    meshtp10_0.05-0.01.msh  slurm-8004.out  slurm-8050.out    sparsematrix.o  tp1_000040.h5  tp1_000080.h5  tp1.exe      vector.c
direct.o    elem2dp1.c  hdfio.c     iterative.c  Makefile       mesh2dp1.c  slurm-7975.out           slurm-8009.out  slurm-8056.out    tp1_000010.h5   tp1_000050.h5  tp1_000090.h5  tp1_mesh.h5  vector.h
elem1dp1.c  elem2dp1.h  hdfio.h     iterative.h  matrix.c       mesh2dp1.h  slurm-7999.out           slurm-8037.out  sparsematrix.c    tp1_000020.h5   tp1_000060.h5  tp1_000100.h5  tp1.o        vector.o
```

4. Explore profile using **gprof –flat-profile ./tp1.exe gmon.out**



```
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ cat slurm-8086.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 55.99      5.85     5.85                             solveSparseGaussSeidel
 35.03      9.51     3.66    74857    48.90    48.90  prodSMV
  6.03     10.14     0.63    74857     8.42     8.42  norme2
  2.78     10.43     0.29    74857     3.87     3.87  axpy
  0.10     10.44     0.01                             createMesh2dp1FromFile
  0.10     10.45     0.01                             initSMMS
  0.00     10.45     0.00   337644     0.00     0.00  prodMV
  0.00     10.45     0.00   255521     0.00     0.00  idxOfNbor2dp1
  0.00     10.45     0.00   168822     0.00     0.00  elem2dp1_phihat_0
  0.00     10.45     0.00   168822     0.00     0.00  elem2dp1_phihat_1
  0.00     10.45     0.00   168822     0.00     0.00  elem2dp1_phihat_2
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dxphihat_0
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dxphihat_1
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dxphihat_2
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dyphihat_0
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dyphihat_1
  0.00     10.45     0.00   112548     0.00     0.00  elem2dp1_dyphihat_2
  0.00     10.45     0.00    84411     0.00     0.00  insertNborIdx
  0.00     10.45     0.00    48465     0.00     0.00  elem1dp1_phihat_0
  0.00     10.45     0.00    48465     0.00     0.00  elem1dp1_phihat_1
  0.00     10.45     0.00    48249     0.00     0.00  computePtFromRefPt2dp1_edge
  0.00     10.45     0.00    18758     0.00     0.00  createMatrix
  0.00     10.45     0.00    18758     0.00     0.00  destroyMatrix
  0.00     10.45     0.00     9379     0.00     0.00  createElem2dp1
  0.00     10.45     0.00     9379     0.00     0.00  destroyElem2dp1
  0.00     10.45     0.00      206     0.00     0.00  createVector
  0.00     10.45     0.00      205     0.00     0.00  destroyVector
  0.00     10.45     0.00      179     0.00     0.00  createElem2dp1_edge
  0.00     10.45     0.00       10     0.00     0.00  stepXDMFNScalar
  0.00     10.45     0.00       10     0.00     0.00  stepXDMFTimedGridScalar
  0.00     10.45     0.00       10     0.00     0.00  writeHDF5AnyData
  0.00     10.45     0.00       10     0.00     0.00  writeXDMFMesh2DPk
  0.00     10.45     0.00        4     0.00     0.00  createSMatrix
  0.00     10.45     0.00        2     0.00     0.00  initSMatrixStructureFromMesh2dp1
  0.00     10.45     0.00        1     0.00     0.00  closeXMF
  0.00     10.45     0.00        1     0.00     0.00  openXMF
  0.00     10.45     0.00        1     0.00     0.00  writeHDF5Mesh2dp1

 %         the percentage of the total running time of the
time       program used by this function.
```



```
 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.

Copyright (C) 2012-2018 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

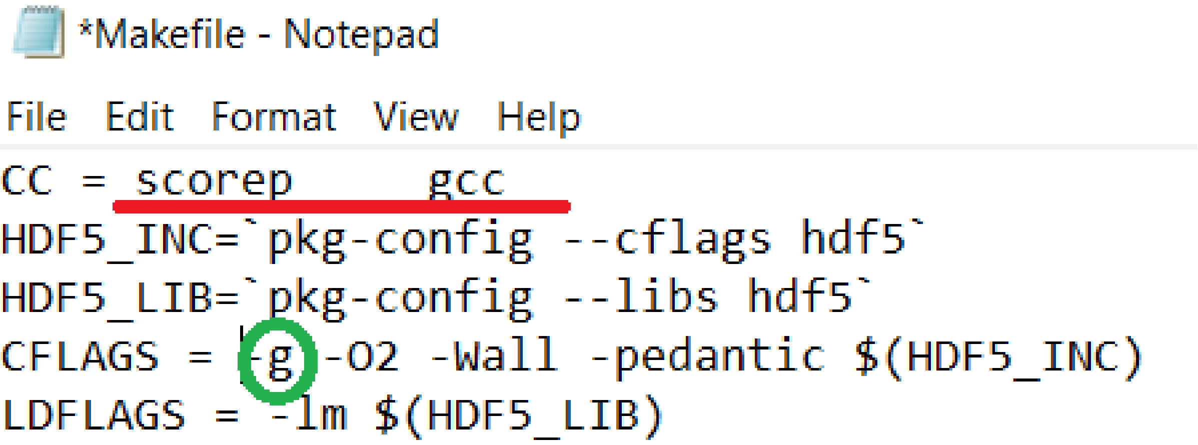
5. Find the most time-consuming function. Comment and justify your answer.

   As we can see from the screenshot that the most time-consuming function is the "solveSparseGaussSeidel" taking an all 5,85sec that is 55,99% of the total amount. So the function is taking an amount of time more than the half of the total which is quite a lot regarding the tasks. It's in these situation where we start thinking of parallelization as a solution.

4 Scalasca/ScoreP

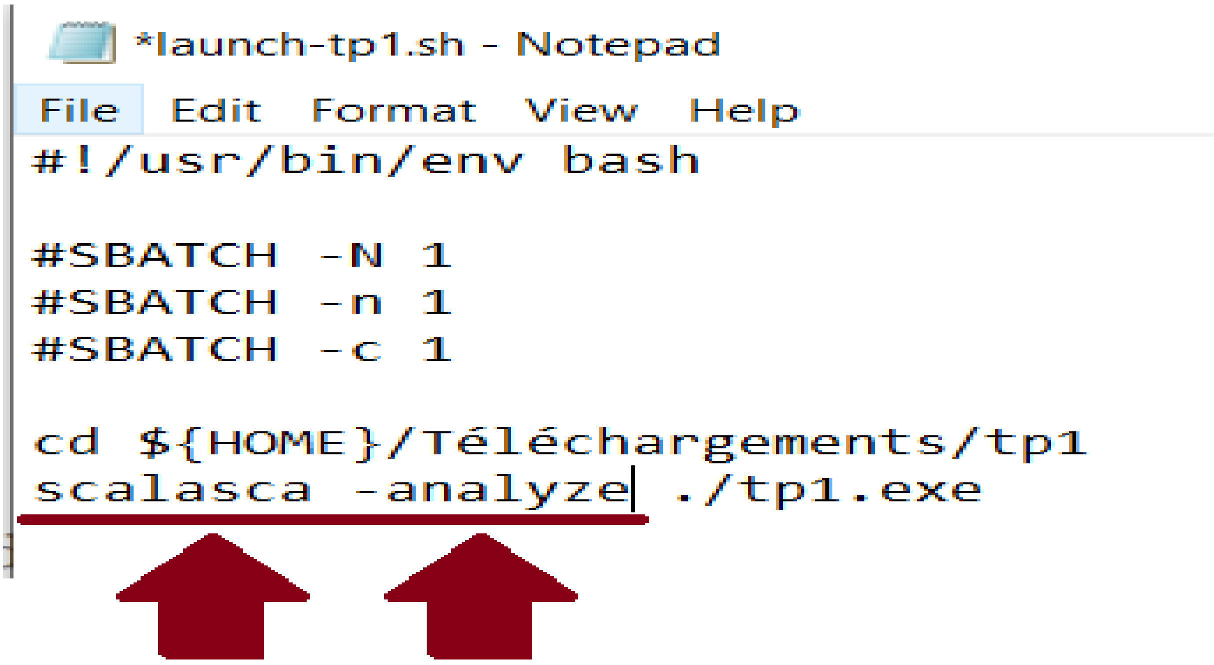
Profiling by automatic code instrumentation.

1. Recompile with **scorep** : **make clean** and then change **CC="scorep gcc"** and remove the **-p** added in previous part in the **Makefile**. In order to activate **scalasca**, you must load the module : **module load scalasca** in the terminal for re-building.



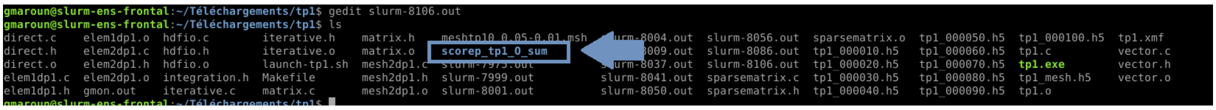
```
CC = scorep      gcc
HDF5_INC=`pkg-config --cflags hdf5`
HDF5_LIB=`pkg-config --libs hdf5`
CFLAGS = -g -O2 -Wall -pedantic $(HDF5_INC)
LDFLAGS = -lm $(HDF5_LIB)
```

2. Modify the submission script to launch the code through scalasca program : add the module activation before running the program with **scalasca -analyze ./tp1.exe**



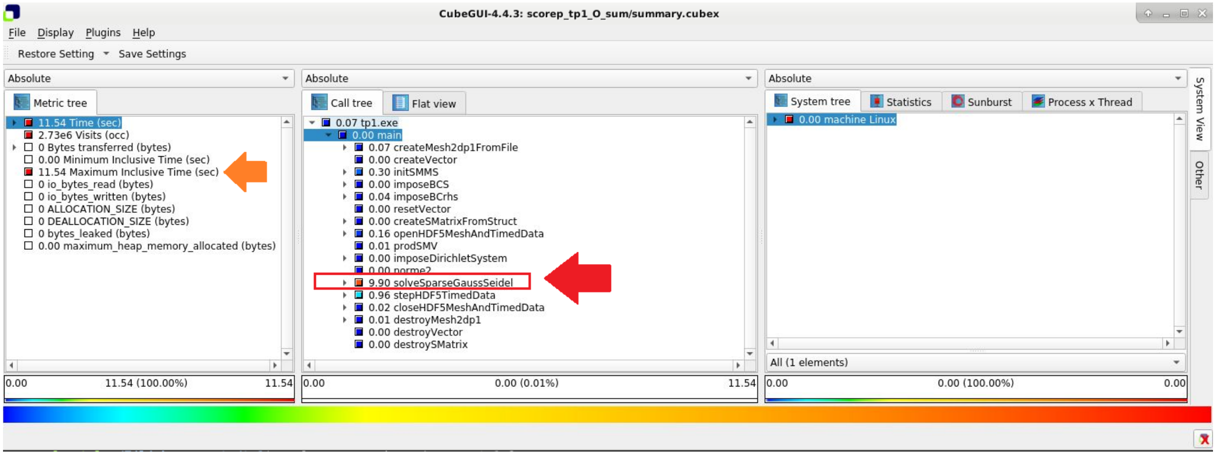
```
#!/usr/bin/env bash

#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1

cd ${HOME}/Téléchargements/tp1
scalasca -analyze ./tp1.exe
```

3. Check that the folder **scorep_tp1_O_sum** has been created properly.



```
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ gedit slurm-8106.out
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ ls
direct.c     elem1dp1.o  hdfio.c        iterative.h    matrix.h     meshtp10_0.05-0.01.msh  slurm-8004.out  slurm-8056.out  sparsematrix.o  tp1_000050.h5  tp1_000100.h5  tp1.xmf
direct.h     elem2dp1.c  hdfio.h        iterative.o    matrix.o     scorep_tp1_O_sum        slurm-8009.out  slurm-8086.out  tp1_000010.h5   tp1_000060.h5  tp1.c          vector.c
direct.o     elem2dp1.h  hdfio.o        launch-tp1.sh  mesh2dp1.c   slurm-7975.out          slurm-8037.out  slurm-8106.out  tp1_000020.h5   tp1_000070.h5  tp1.exe        vector.h
elem1dp1.c   elem2dp1.o  integration.h  Makefile       mesh2dp1.h   slurm-7999.out          slurm-8041.out  sparsematrix.c  tp1_000030.h5   tp1_000080.h5  tp1_mesh.h5    vector.o
elem1dp1.h   gmon.out    iterative.c    matrix.c       mesh2dp1.o   slurm-8001.out          slurm-8050.out  sparsematrix.h  tp1_000040.h5   tp1_000090.h5  tp1.o
gmaroun@slurm-ens-frontal:~/Téléchargements/tp1$ 
```

4. Explore the profile data using GUI : **scalasca -examine ./scorep_tp1_O_sum**



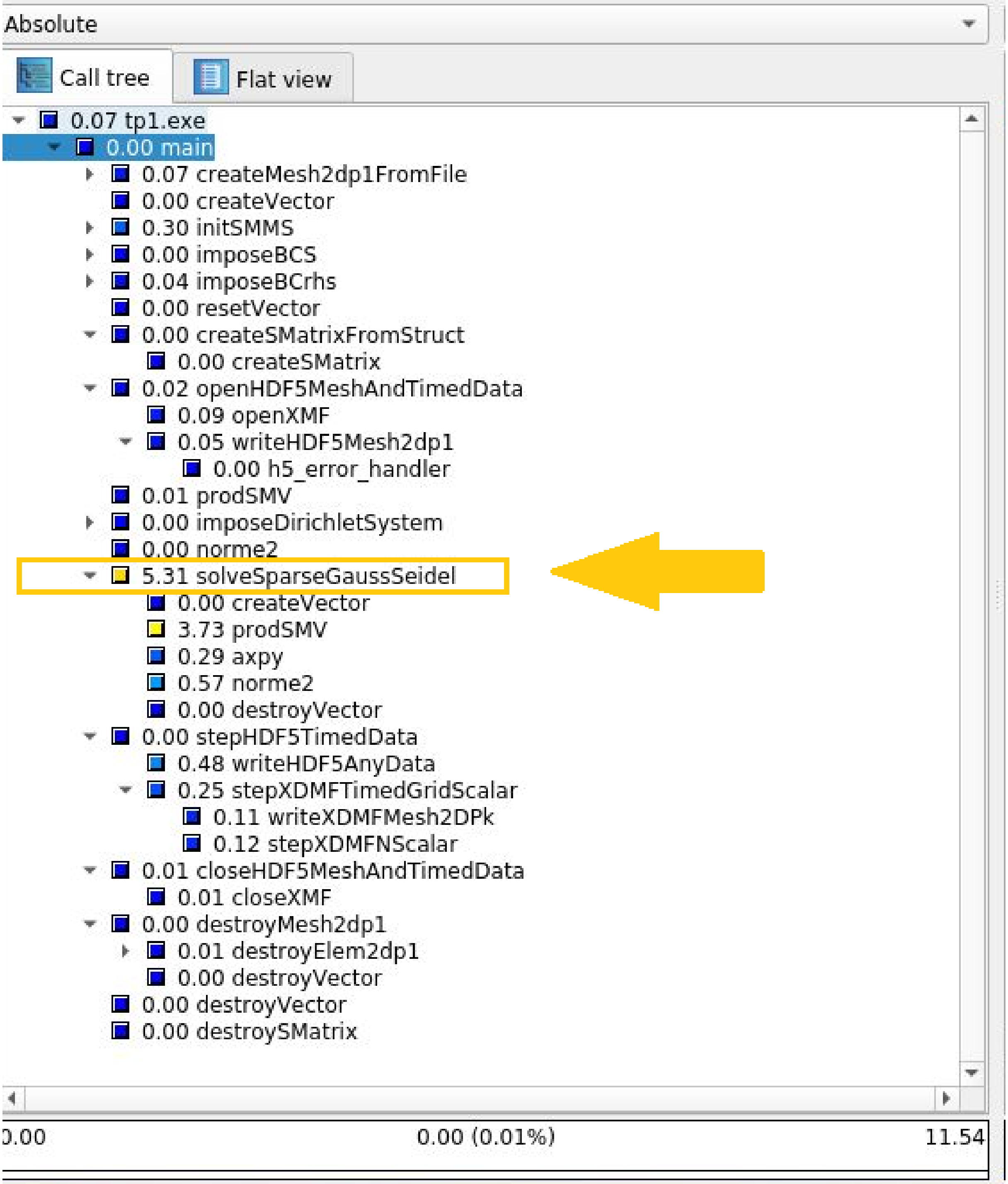
```
11.54 Time (sec)
2.73e6 Visits (occ)
0 Bytes transferred (bytes)
0.00 Minimum Inclusive Time (sec)
11.54 Maximum Inclusive Time (sec)
0 io_bytes_read (bytes)
0 io_bytes_written (bytes)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)

0.00      11.54 (100.00%)      11.54
```

```
0.07 tp1.exe
  0.00 main
    0.07 createMesh2dp1FromFile
    0.00 createVector
    0.30 initSMMS
    0.00 imposeBCS
    0.04 imposeBCrhs
    0.00 resetVector
    0.00 createSMatrixFromStruct
    0.16 openHDF5MeshAndTimedData
    0.01 prodSMV
    0.00 imposeDirichletSystem
    0.00 norme2
    9.90 solveSparseGaussSeidel
    0.96 stepHDF5TimedData
    0.02 closeHDF5MeshAndTimedData
    0.01 destroyMesh2dp1
    0.00 destroyVector
    0.00 destroySMatrix

0.00      0.00 (0.01%)      11.54
```

```
0.00 machine Linux

0.00      0.00 (100.00%)      0.00
```

*By using the **CubeGUI** with **scalasca**, we can be able to see the time of consumption taken by each function when in parallel mode.*

5. Find the most time-consuming function. Comment and justify your answer. Is it in coherence with the results of the previous part ?



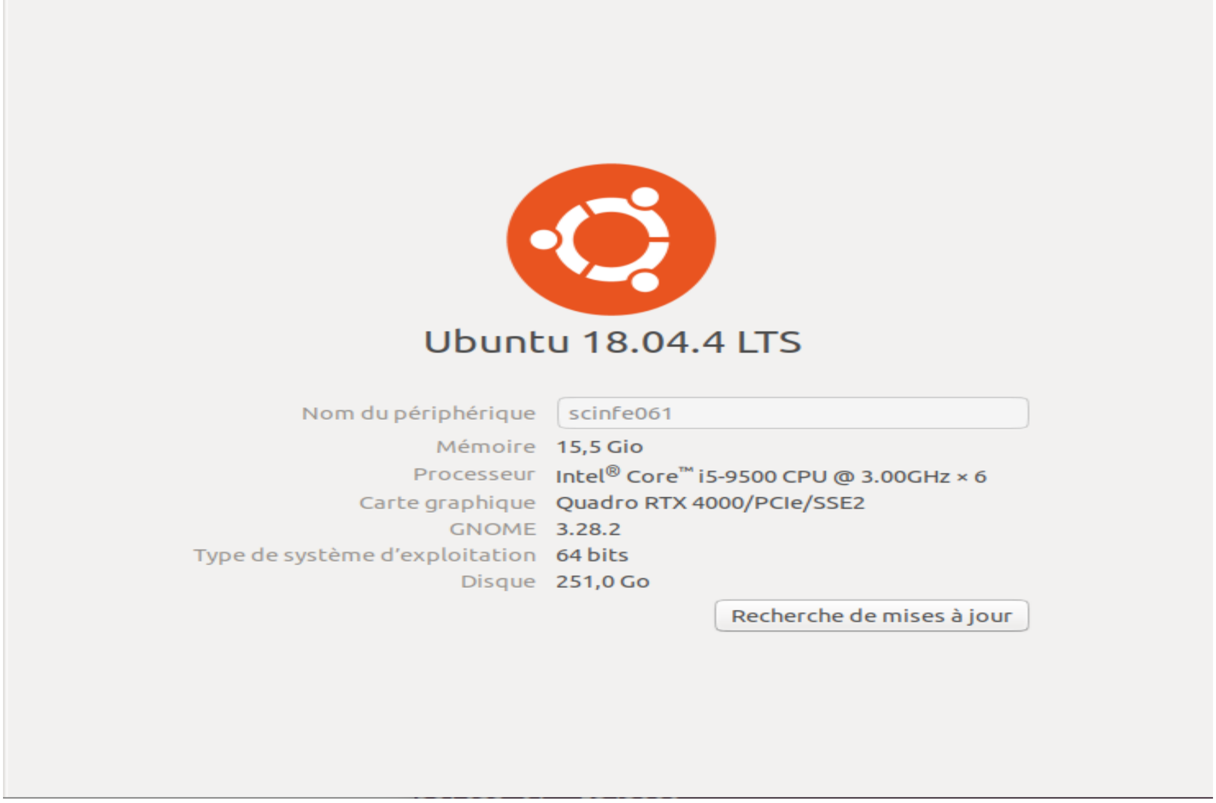
```
0.07 tp1.exe
  0.00 main
    0.07 createMesh2dp1FromFile
    0.00 createVector
    0.30 initSMMS
    0.00 imposeBCS
    0.04 imposeBCrhs
    0.00 resetVector
    0.00 createSMatrixFromStruct
      0.00 createSMatrix
    0.02 openHDF5MeshAndTimedData
      0.09 openXMF
      0.05 writeHDF5Mesh2dp1
        0.00 h5_error_handler
    0.01 prodSMV
    0.00 imposeDirichletSystem
    0.00 norme2
    5.31 solveSparseGaussSeidel
      0.00 createVector
      3.73 prodSMV
      0.29 axpy
      0.57 norme2
      0.00 destroyVector
    0.00 stepHDF5TimedData
      0.48 writeHDF5AnyData
      0.25 stepXDMFTimedGridScalar
        0.11 writeXDMFMesh2DPk
        0.12 stepXDMFNScalar
    0.01 closeHDF5MeshAndTimedData
      0.01 closeXMF
    0.00 destroyMesh2dp1
      0.01 destroyElem2dp1
      0.00 destroyVector
    0.00 destroyVector
    0.00 destroySMatrix
```

```
).00                 0.00 (0.01%)                 11.54
```

The most time-consuming function is also the "solveSparseGaussSeidel" same as before, so yes, in coherence. However, the time consumed while on parallel is 5,31sec, which is less by more than half of the second than the previous result.

6. Explain what function or what part of the code should be parallelized. What would be the ideal speedup based on these sequential results.

As it has already been understood, the "solveSparseGaussSeidel" function takes the most consumption time of the total and that's what makes it perfectly fit to be parallelized. *The speedup based on these sequential results is* $S_p = T_1/T_p = 5.85/5.31 = 1.101 sec$. *With an efficiency of* $E_p = S_p/p = 1.101 sec/6 = 0.1835 < 1$. We can conclude that it isn't so much efficient while in parallel mode.



Ubuntu 18.04.4 LTS

Nom du périphérique scinfe061
Mémoire 15,5 Gio
Processeur Intel® Core™ i5-9500 CPU @ 3.00GHz × 6
Carte graphique Quadro RTX 4000/PCIe/SSE2
GNOME 3.28.2
Type de système d'exploitation 64 bits
Disque 251,0 Go

As we can see from this photo, there are 6 processors working on this PC which means the ideal and most efficient speedup would be to have a $T_P = T_1/p = 5.85/6 = 0.975 sec$

7. Performs a second analysis after changing the variable **outputfreq** to 0 in the **tp1.c** file in order to deactivate writing the results in output files. Same question as previous one with this new experiment. Compare and comment the results.

We notice that the time consumed by the "prodSMV" function has dropped by 0.21s, and "solveSparseGaussSeidel" function still takes a lot of time to get executed with the parallelization as a solution

La fin.