Some infos to use/change provided test cases

pierre.ledac@cea.fr

Test cases :
Here is a list of 4 test cases which are of interest for us :

Name	**Kind**	**Mesh**	**Specificity**	**Initial mesh size**	**Initial MPI processes number**
VEF	Laminar flow with thermal coupling	3D tetrahedral	Laminar operators	3672+3528	2
VEF_LES	Turbulent flow (LES model)	3D tetrahedral	Runge Kutta 3, EF_stab and Muscl schemes, CG solver	9352	2
VEF_RANS	Turbulent flow (RANS model)	3D tetrahedral	Implicit and upwind schemes, CG solver (from Petsc)	42964	2
VDF_FT	Laminar diphasic flow	3D hexahedral	Lagragian mesh	16000+800	2

The mesh sizes are small in order to help during the validation of the OpenMP implementation. It is easy to quickly change the mesh size and number of MPI processes as indicated below.

The rule of thumb to optimize performance when partitionning a mesh in Trust is to target between 20000 and 30000 cells per MPI process : by reducing this ratio cells number/process, you will benefit from the higher bandwith on the L3 memory cache (if the problem solved by the MPI process fits into it), but in the same time, to keep a good parallel efficiency, you will not reduce too much, to keep the cost of MPI message exchange low compared to the work on the CPU.

How to install the Baltik project containg the test cases :
First, install the Trust and TrioCFD codes. I suppose it is already done.
Then initialize TrioCFD environment (source PATH_ROOT_TRIOCFD/env_TrioCFD.sh) and :

```
$ tar cvfz IAEC.tgz
$ cd IAEC
# You should edit there the project.cfg file and give the correct path to TrioCFD (last line) then:#
$ baltik_bin_configure && ./configure && make check_debug
```

It will configure, build the project and run the tests case located under IAEC/tests/Reference

If you want to change the Trust/TrioCFD, just copy sources files inside IAEC/src directory and run :

```
$ ./configure         # If files added/renamed/removed #
$ make debug          # To build the project #
$ make check_debug    # To run test cases #
```

How to check non-regression :
There are several tools available in the Trust environment to check for instance you have the same results during a sequential calculation and a parallel one.

You can use the « make check_debug » command for instance to automatically run the provided test cases under IAEC/tests/Reference :

```
$ cd IAEC
$ make check_debug
...

Searching test cases in /home/pl254994/tests/IAEC/tests/Reference. Please wait...

*********************************************
* 1/1 . test case : VEF
*********************************************
Partitioning for parallel calculation...End.
Running sequential calculation...End.
Number of differences : 0
Maximal relative error encountered : 0.00000e+00
Running parallel calculation on 2 cpus...End.
Number of differences : 0
Maximal relative error encountered : 1.93718e-07
Parallel tests cases :
    1 PARALLEL OK

-----------------------------------------

Complete results into the file:
/home/pl254994/tests/IAEC/build/tests/.tests_IAEC

Wed Mar 14 16:16:44 CET 2018 : Test results with the version dated Mar 14 15:54
Binary: /home/pl254994/tests/IAEC/IAEC
 ----------------------------------------------------------------------------------------
| START  | END    | CPU | NP | SPU | NDT |  GCP | ELEM | MEM  | PLOTS | STATE | NAME
 ----------------------------------------------------------------------------------------
|16:16:45|16:16:55| 10.6s|..1.|.....|...51|.....0|..3672|..27Mo|.......| OK    |VEF
|16:16:56|16:17:04|  8.4s|..2.|1.265|...51|.....0|..3672|..51Mo|....9/9| OK    |PAR_VEF
 --------------------------------------------------------------------------------------
```

In this example, the test case VEF is executed and compared to an archive result file (VEF.lml.gz stored under IAEC/tests/Reference/VEF) and after that, the PAR_VEF (which is the parallel version of the VEF test case) is executed on 2 CPUs. The result (PAR_VEF.lml) is then compared to the previous result (VEF.lml, not the archived one) to check that on this test case, the parallel simulation gives the same results than the sequential one.
Behind the « make check_debug » command, a binary tool named **compare_lata** (available under $TRUST_ROOT/exec) is comparing (with a relative tolerance) the results file (.lml or .lata files issued from Trust). The syntax is :

```
$ compare_lata result.lml my_reference.lml
Number of differences : 0
Maximal relative error encountered : 0.00000e+00
```

Changing just the 2th digit of one field (29.966 -> 29.976) will lead to a difference detected by the tool :

```
$ compare_lata VEF.lml ref.lml
```

```
Ecarts pour TEMPERATURE_ELEM_dom_fluide au temps: 5.10000000e+00 Erreur max: 3.333e-
04 ELEM seq 3526= 2.99766331e+01 ELEM par 3526= 2.99666328e+01  gmax  3.00005512e+01
composante 0
Number of differences : 1
Maximal relative error encountered : 3.33335e-04
```

So you can use either « make check_debug » command (and changing the trust command by inserting OMP_NUM_THREADS to specify the number of threads, ask me if you need help) or build you own tool to quickly check the non regression of OpenMP parallelization, I can imagine something like :

```
export OMP_NUM_THREADS=1
# Run on one thread :
trust VEF || exit
cp VEF.data VEF2.data
export OMP_NUM_THREADS=2
# Run on 2 OpenMP threads :
trust VEF2 || exit
compare_lata VEF2.lml VEF.lml || exit
# Partition the mesh used by VEF data file on 2 sub-domains :
make_PAR.data VEF 2
# Run on 2 MPI processes and 2 OpenMP threads per MPI process :
export OMP_NUM_THREADS=2
trust PAR_VEF 2 || exit
compare_lata PAR_VEF.lml VEF.lml || exit
```

I suppose here you know the purpose of the « make_PAR.data » command which is useful to quickly generate data files (and decomposed domains in .Zones files). The syntax is :

```
$ make_PAR.data datafile.data N
```

Where N is the number of sub-domains. The datafile should have some pragmas :
# BEGIN MESH #, # END MESH #, # BEGIN PARTITION #, etc... and the provided test cases normally have all this pragmas included.

So for a strong scaling study on the test case VEF, you will create the different data files with something like :

```
make_PAR.data VEF 2
make_PAR.data VEF 4
make_PAR.data VEF 8
...
```

How to change the computation duration in the data file ?
You will change the time step numbers (**nb_pas_dt_max** keyword) or the final time (**tmax** keyword) in the time scheme definition of the data file :

```
Scheme_euler_explicit sch # Euler time scheme #
Read sch
{
        tinit 0.
        tmax 5.
        nb_pas_dt_max 50000
        dt_min 1.e-6
        dt_max 1.e-1
        dt_impr 1.
        dt_sauv 20.
        seuil_statio 1.e-8
}
```

How to refine a mesh in a data file ?
For a weak scaling study (mesh size and cores number are increased in the same way in order to keep the number of cells per CPU core constant), you will use one of the two available keywords to refine the mesh for the first 3 test cases (tetra meshes):

```
Domaine domain_name
# BEGIN MESH #

... # Read or create the domain #

raffiner_anisotrope domain_name  # It will divide each tetra in 4 tetras in 3D #
raffiner_isotrope domain_name    # It will divide each tetra in 8 tetras in 3D #
# END MESH #
```

For the last test case using regular hexa cells (VDF), you will use **raffiner_isotrope** to create a mesh with 8 times more elements.

How to fully check TrioCFD non regression :
You will run in the Baltik project the command :

```
$ make check_all_debug   # Use the Baltik binary built in debug mode #
```

It will run all the test cases (~1313, several hours needed) of the TrioCFD code. By default, OMP_NUM_THREADS is set to the max number of cores of your PC, so each sequentiel test cases will run with m=OMP_NUM_THREADS threads and the results will be compared to the archived sequential result of the test case.
Then the test case will be run on n MPI processes (n may be 2 or higher), so the hybrid parallelism will be tested (n MPI processes using each m threads) and the result compared to the result obtained during the previous OpenMP run on m threads.

May be it could be better the first time to limit to 2 OpenMP threads with :

```
$ export OMP_NUM_THREADS=2
$ make check_all_debug
```

How to change manually (if needed) the partition of the mesh (and thus increase MPI process number) ?
Remember that the make_PAR.data command indicated previously tries to automatically change the partition of the mesh. But sometimes manually changing the partition may be necessary. So, the number of MPI processes is given by the partition number given in the data file by the **nb_parts** keyword.

```
# BEGIN PARTITION
Partition dom_solide
```

```
{
        Partition_tool metis { Nb_parts 2 }
        Larg_joint 1
        zones_name DOM1
}
Partition dom_fluide
{
        Partition_tool metis { Nb_parts 2 }
        Larg_joint 2
        zones_name DOM2
}
End
END PARTITION #
```

If the data file contains two domains, you will need to set the same number for **nb_parts**. In case of imbalance between the mesh sizes of the two domains (e.g. : fluid domain with 10 more cells than the solid domain), in order to keep a ratio cells/MPI process high enough on the smallest domain, you would partition last one with a smaller **nb_parts** (but use **nb_parts_tot** keyword equals to the **nb_parts** of the other domain) : in this case, on the smallest domain, only **nb_parts** MPI processes will solve the related problem and **nb_parts_tot-nb_parts** MPI processes will do nothing :

```
# BEGIN PARTITION
Partition dom_solide
{
        Partition_tool metis { Nb_parts 1 }
        Nb_parts_tot 2
        Larg_joint 1
        zones_name DOM1
}
Partition dom_fluide
{
        Partition_tool metis { Nb_parts 2 }
        Larg_joint 2
        zones_name DOM2
}
End
END PARTITION #
```