

# Guide d'utilisation de la version multi-GPU de CUTEFLOW

Vincent Delmas

16 novembre 2020

## Table des matières

<b>1</b>	<b>Avant-propos</b>	<b>2</b>
<b>2</b>	<b>Git pour la gestion des versions</b>	<b>2</b>
<b>3</b>	<b>Structure des dossiers</b>	<b>3</b>
<b>4</b>	<b>Fichiers sources et compilation du code</b>	<b>3</b>
<b>5</b>	<b>Découpe du maillage en sous domaines : <i>split_mesh</i></b>	<b>4</b>
<b>6</b>	<b>Lancement d'une simulation : Cutofflow</b>	<b>6</b>
6.1	Lancement de la simulation à partir d'une solution déjà générée . . . . .	8
6.2	Format des fichiers solutions . . . . .	8
<b>7</b>	<b>Lancement de plusieurs simulations</b>	<b>10</b>
7.1	Génération des fichiers de données . . . . .	10
7.2	Préparation du dossier <i>base_files</i> . . . . .	10
7.3	Lancement du script <i>multi.sh</i> . . . . .	11
7.4	Lancement du job array . . . . .	12
7.5	Récupération des fichiers résultats dans un seul dossier (Optionnel) . . . . .	13
<b>8</b>	<b>Combinaison des fichiers de solution</b>	<b>14</b>
<b>9</b>	<b>Combinaison des fichiers de solution pour Bluekenue</b>	<b>14</b>
<b>10</b>	<b>Traitement dans Paraview des fichiers *.vtk</b>	<b>15</b>
10.1	Tunnel SHH pour la visualisation avec Paraview sur les clusters de calcul en utilisant MobaXterm . . . . .	15
10.2	Visualisation des fichiers résultats générés par CUTEFLOW sur Paraview . . . . .	19

# 1 Avant-propos

On présente dans la suite le guide d'utilisation du code CuteFlow et des codes de pre/post-traitement associés. Ce code est hébergé sur CEDAR, un cluster de calcul de *Compute Canada*.

CuteFlow est un code de résolution des équations de St-Venant par méthode volumes finis qui a été développé dans le groupe de recherche du GRANIT de l'ETS. La première version séquentielle sur CPU a été développée dans un premier temps par Azzeddine Soulaïmani et Youssef Loukili [**louki**] puis par Jean-Marie Zokagoa [**ZOKAGOA**]. Le code a par la suite été porté sur GPU en CUDA Fortran par Arun Kumar Suthar [**arun**]. Finalement le code a été porté sur une architecture multi-GPU par Vincent Delmas.

Pour tout ce qui concerne l'utilisation des clusters de calculs, l'utilisateur peut se référer au wiki officiel de *Compute Canada* [https://docs.computecanada.ca/wiki/Compute\\_Canada\\_Documentation](https://docs.computecanada.ca/wiki/Compute_Canada_Documentation). Sur ce wiki, on peut entre autres trouver les informations relatives à la connexion aux clusters de calculs, au stockage de données, au lancement de job et à l'allocation interactive de noeuds.

Pour plus de facilité d'utilisation des clusters de calculs, il est préférable d'être à l'aise avec un éditeur en ligne de commande comme vim, emacs ou nano et d'être capable de séparer un terminal en plusieurs terminaux, par exemple, à l'aide de tmux <https://docs.computecanada.ca/wiki/Tmux> (tout cela est installé par défaut sur les clusters de calculs).

## 2 Git pour la gestion des versions

On utilise le logiciel Git [**git**] pour gérer les versions du code. Un tutoriel est présent sur le [site officiel](#), un autre est présent sur [OpenClassrooms](#). On présente simplement comment cloner le dossier du code et comment le tenir à jour des modifications qui y sont faites. Git permet évidemment de faire bien plus de choses, entre autres, remonter dans l'historique des fichiers ou permettre à des personnes de travailler dans différentes branches du même code. Il serait d'ailleurs intéressant que chaque personne qui modifie le code travaille dans une branche spécifique.

Le dossier Git d'origine du code est présent sur CEDAR dans l'espace projets du groupe *def-soulaima*. Il est conseillé de cloner le code dans l'espace scratch des grappes de calculs pour avoir la place de stocker les fichiers résultats générés par les simulations.

Pour cloner le projet sur CEDAR dans l'espace scratch de l'utilisateur :

```
# Sur Cedar
cd ~/scratch
git clone ~/projects/def-soulaima/CUTEFLOW_CUDA_MPI.git
```

Les lignes précédées de # sont des commentaires.

Un dossier CUTEFLOW\_CUDA\_MPI sera créé contenant les fichiers importants du code, l'opération peut durer quelques minutes. On peut cloner ce projet sur d'autres grappes de calcul comme GRAHAM ou BELUGA ou encore sur son ordinateur personnel de la façon suivante,

```
# Sur une autre machine, remplacer username par nom d'utilisateur sur CEDAR
git clone ssh://username@cedar.computecanada.ca/~/projects/def-soulaima/CUTEFLOW_CUDA_MPI.git
```

Si l'utilisateur détruit par inattention certains fichiers du code, il peut toujours synchroniser son dossier le dossier source du code en faisant,

```
# Reset le dossier par rapport au dernier telechargement
git reset --hard

# Telecharge les changements depuis le dossier source sur CEDAR
git pull
```

Attention, de cette façon toutes les modifications apportées aux fichiers gérés par git par l'utilisateur seront écrasées. Cela n'aura pas d'impact sur les nouveaux fichiers et dossiers que l'utilisateur aura pu créer. Il y a d'autres façons de faire pour conserver une partie des modifications voir <https://git-scm.com/docs/gittutorial>.

La branche *master* du dossier concerne la version multi-GPU du code. Il existe aussi une branche *multi-cpu* qui concerne la version multi-cpu du code, l'utilisateur peut la charger en faisant,

```
git checkout multi-cpu
```

### 3 Structure des dossiers

On présente la structure du dossier du code, on peut l'obtenir avec la commande `tree -d CUTEFLOW_CUDA_MPI`. Dans ce dossier, on trouve le code de CUTEFLOW mais aussi tous les codes de pre/post-traitement ainsi que des scripts et des exemples de simulations et des fichiers de maillages.

```
CUTEFLOW_CUDA_MPI
├── bin
├── build
├── docs
│   └── guide_utilisation
├── examples
│   ├── examples1
│   ├── examples2
│   ├── examples3
│   └── base_files
├── meshes
│   ├── mille_iles_mesh_files
│   ├── RDP
│   ├── RDP_1M5
│   └── RDP_bris
├── scripts
└── src
    ├── conversion_maillage_JM
    ├── conversion_maillage_PD
    ├── cuteflow
    ├── merge_bluekenue
    ├── refine_mesh
    ├── split_mesh
    └── split_solution
```

Le dossier **src** contient les fichiers sources des différents programmes.

Le dossier **bin** contient les exécutables créés après la compilation.

Le dossier **build** ne contient rien au départ. Ce dossier est utilisé pour compiler le code et il contiendra les fichiers objets créés lors de la compilation.

Le dossier **example** contient des exemples de simulation pour tester le bon fonctionnement du code.

Le dossier **meshes** contient quelques fichiers de maillages classiques.

Le dossier **scripts** contient plusieurs scripts écrits en *bash* pour faire différentes choses.

Le dossier **docs** contient le guide d'utilisation du code.

### 4 Fichiers sources et compilation du code

Plusieurs codes sont présent dans le dossier **src**,

- `conversion_maillage_JM`
- `conversion_maillage_PD`
- `cuteflow`
- `merge_bluekenue`
- `merge_solutions`
- `refine_mesh`
- `split_mesh`
- `split_solution`

Un fichier *makefile* est présent dans le dossier **CUTEFLOW\_CUDA\_MPI**. Ce fichier permet de compiler tous les codes présents dans le dossier **src**. Pour compiler le code, la bonne façon de faire est de copier ce *makefile* dans le dossier **build** puis de compiler le code dans ce dossier. Les exécutables seront automatiquement placés dans le dossier **bin**

```
#Pour compiler tous les codes depuis le dossier CUTEFLOW_CUDA_MPI
```

```
cp makefile build/  
cd build/  
make
```

Il est possible selon le cluster de calcul que la compilation prenne plusieurs minutes.

On peut compiler les codes un par un en spécifiant leurs noms, par exemple pour compiler uniquement *cuteflow*,

```
make ../bin/cuteflow
```

ou encore pour le code *split\_mesh*,

```
make ../bin/split_mesh
```

L'utilisateur peut modifier les options de compilations dans le fichier *makefile* qu'il vient de copier dans le dossier **build**.

Pour la compilation du code *cuteflow* il faut spécifier la *Compute Capability* cible des GPU pour lesquels on compile. La *Compute Capability* par défaut dans le makefile est 6.0 ce qui fonctionne pour la compilation sur CEDAR et GRAHAM. Pour BELUGA, il faut spécifier une *Compute Capability* de 7.0. C'est possible en faisant,

```
make CC=70
```

Si on veut être sûr que le code est bien recompilé, on peut forcer la re-compilation de tous les fichiers avec l'option "-B" de la façon suivante,

```
#Sur CEDAR ou GRAHAM  
make -B
```

ou sur BELUGA,

```
#Sur BELUGA  
make -B CC=70
```

Les exécutables seront créés dans le dossier **bin**. A la suite de la compilation, le résultat de la commande *ls* dans le dossier **bin** devrait être le suivant,

```
#Resultat de la commande ls dans le dossier bin  
conversion_maillage_PD  
cuteflow  
merge_bluekenue  
merge_solutions  
refine_mesh  
split_mesh  
split_solution
```

On présente dans la suite l'utilisation de chacun des codes.

## 5 Découpe du maillage en sous domaines : *split\_mesh*

Le code **split\_mesh** permet de faire la décomposition d'un maillage en plusieurs sous domaines grâce à la bibliothèque METIS.

Avant de présenter le fonctionnement du code, il faut savoir qu'on a deux types de fichiers de maillages différents. Les anciens fichiers sont formatés pour une seule entrée et une seule sortie alors que les nouveaux peuvent prendre plusieurs entrées et sorties. Pour savoir si le maillage prends en compte plusieurs entrées et plusieurs sorties, il suffit de regarder dans les fichiers de maillages. Si le fichier ne prend en compte qu'une seule entrée, sous la ligne "Noeuds d'entrée" il n'y aura qu'une seule colonne avec les indices des noeuds d'entrée, si le fichier en prends en compte plusieurs, il y aura deux colonnes, la première pour les indices des noeuds d'entrée et la seconde pour le numéro de l'entrée à laquelle le noeud appartient. Il en va de même pour le traitement de plusieurs sorties.

Le code s'exécute avec plusieurs argument dans la ligne de commande de la façon suivante,

```
./split_mesh fichier_de_maillage nombre_de_sous_domaines multi_entrees multi_sorties
```

Où multi\_entrees et multi\_sorties prennent les valeur de 1 uniquement si le fichier de maillage est formaté pour avoir plusieurs entrées/sorties. On présente la procédure complète pour décomposer le fichier de maillage Mille\_Iles\_mesh\_481930\_elts.tx présent dans le dossier **meshes**. On va faire cette décomposition dans le dossier **examples/example1**.

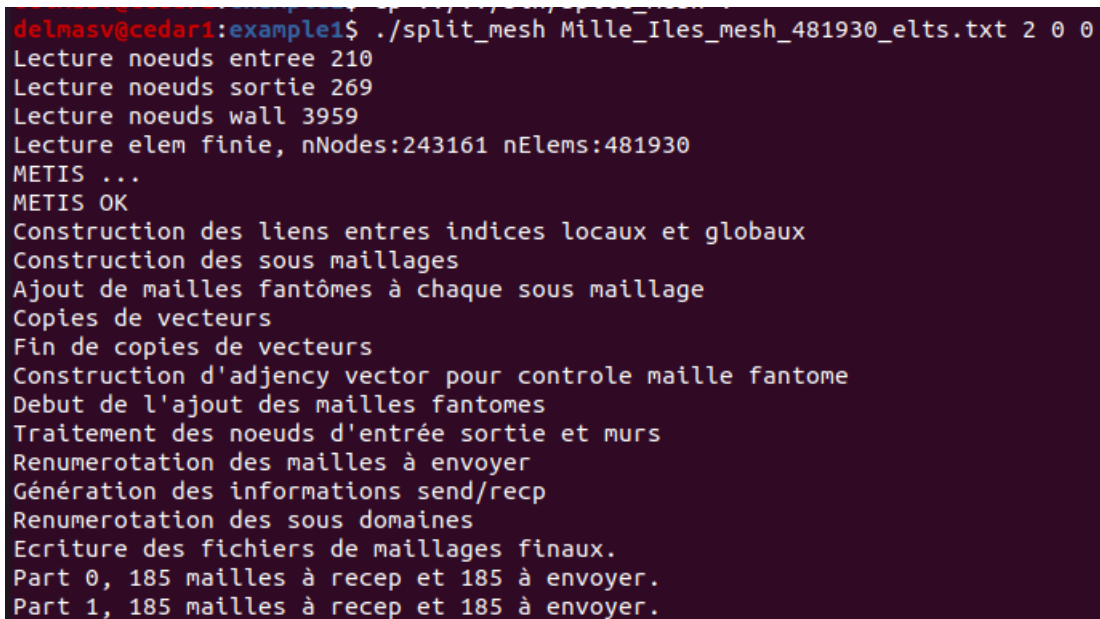
```
#Depuis le dossier CUTEFLOW_CUDA_MPI
cd exemples/example1

#Copie du fichier de maillage dans le dossier courant
cp ../../meshes/mille_iles_mesh_files/Mille_Iles_mesh_481930_elts.txt .

#Copie de l'executable split_mesh (prealablement compile) dans le dossier courant
cp ../../bin/split_mesh .

# Lancement du code avec comme argument:
# -le nom fichier de maillage a decouper
# -le nombre de sous domaine desires, ici 2
# -multi_entree=0, format pour une seule entree
# -multi_sorties=0, format pour une seule sortie
./split_mesh Mille_Iles_mesh_481930_elts.txt 2 0 0
```

Une fois ces commandes effectués la sortie dans le terminal devrait être conforme à la figure 1.



```
delmasv@cedari:example1$ ./split_mesh Mille_Iles_mesh_481930_elts.txt 2 0 0
Lecture noeuds entree 210
Lecture noeuds sortie 269
Lecture noeuds wall 3959
Lecture elem finie, nNodes:243161 nElems:481930
METIS ...
METIS OK
Construction des liens entres indices locaux et globaux
Construction des sous maillages
Ajout de mailles fantômes à chaque sous maillage
Copies de vecteurs
Fin de copies de vecteurs
Construction d'adjency vector pour controle maille fantome
Debut de l'ajout des mailles fantomes
Traitement des noeuds d'entrée sortie et murs
Renumerotation des mailles à envoyer
Génération des informations send/recp
Renumerotation des sous domaines
Ecriture des fichiers de maillages finaux.
Part 0, 185 mailles à recep et 185 à envoyer.
Part 1, 185 mailles à recep et 185 à envoyer.
```

FIGURE 1 – Exemple 1 : Décomposition du fichier Mille\_Iles\_mesh\_481930\_elts.txt

Si on utilise un fichier de maillage plus gros, il est possible d’avoir une segmentation fault à cause d’un manque de mémoire. Il y a plusieurs façon de régler ce problème.

Soit on demande un noeud de calcul interactivement avec la commande *salloc* comme indiqué sur le wiki [https://docs.computecanada.ca/wiki/Running\\_jobs](https://docs.computecanada.ca/wiki/Running_jobs), par exemple de la façon suivante,

```
salloc --time=0-02:00 --ntasks=1 --mem-per-cpu=32000M --account=def-soulaima
```

De cette façon un noeud de calcul nous sera alloué pour 2 heures avec 32 Go de mémoire ce qui suffit pour décomposer le plus grand maillage actuel de 11 millions d’éléments. La durée de la décomposition peut varier d’une trentaine de minutes lorsque l’on fait la décomposition en 32 à plus d’une heure si on fait une décomposition en 2 sous-domaines (plus on décompose en un grand nombre de sous domaines plus ce sera rapide). Une fois le noeud de calcul attribué, on peut relancer la commande

```
./split_mesh Mille_Iles_mesh_481930_elts.txt 2 0 0
```

et il ne devrait pas y avoir de problèmes.

Alternativement on peut utiliser un script de lancement présent dans le dossier **script** sous le nom de **decomp.sh**. C’est un script pour lancer de manière différée le code. Il suffit de le copier dans le dossier du maillage que l’on veut décomposer et de modifier les paramètres pour le travail que l’on veut mener.

```
cp ../../scripts/decomp.sh
```

```
#Après avoir modifier le fichier decomp.sh on peut lancer le job
```

```
sbatch decomp.sh
```

Avec cette façon de faire le job sera lancé quand l'ordonnanceur du cluster le voudra. On peut regarder les jobs en attente de lancement avec la commande

```
squeue -u $USER$
```

Pour plus d'informations, l'utilisateur peut se tourner vers le wiki officiel de *Compute Canada* [https://docs.compute-canada.ca/wiki/Compute\\_Canada\\_Documentation](https://docs.compute-canada.ca/wiki/Compute_Canada_Documentation).

Une fois que le code *split\_mesh* aura fini, plusieurs fichiers seront créés. En premier lieu, **les fichiers de maillage** de chaque sous domaine, dans l'exemple plus haut `0_Mille_Iles_mesh_481930_elts.txt` et `1_Mille_Iles_mesh_481930_elts.txt`. Ce sont ces fichiers qui seront lus par le code *cuteflow*. Il faut noter que dans le fichier *donnees.f* il suffit de donner le nom de base du fichier de maillage, ici `"Mille_Iles_mesh_481930_elts.txt"`, le code se chargera de détecter qu'il est lancé sur plusieurs GPU et chaque processus MPI lira le fichier de maillage qui lui correspond, dans l'exemple, le processus 0 lira le fichier `"0_Mille_Iles_mesh_481930_elts.txt"` et le processus 1 lira `"1_Mille_Iles_mesh_481930_elts.txt"`

Des **tables de correspondance** entre la numérotation globale et la numérotation locale de chaque sous domaine seront générées. Il y a deux types de fichiers, les fichiers `*_liens_elems.txt` et `*_liens_nodes.txt`. Comme leur noms l'indique les fichiers `*_liens_nodes.txt` contiennent sur chaque ligne deux entiers représentant en premier la numérotation locale et en second la numérotation globale d'un noeud, c'est le même principe pour les éléments dans les fichiers `*_liens_elems.txt`. Il y a un fichier de correspondance par sous domaine, dans l'exemple donné les fichiers `0_liens_nodes.txt`, `1_liens_nodes.txt`, `0_liens_elems.txt` et `1_liens_elems.txt` seront créés. **Ces fichiers sont d'une très grande importance pour plusieurs autres codes.**

Les autres fichiers servent à déboguer le code ou à visualiser les sous domaines avec Gnuplot.

## 6 Lancement d'une simulation : Cuteflow

Dans cette section on présente comment lancer une simulation depuis le dossier **examples/example2**. Cette simulation utilisera 2 GPU et s'effectuera sur les fichiers de maillages que l'utilisateur a déjà décomposé dans le dossier **examples/example1** voir section 5.

```
#Depuis le dossier CUTEFLOW_CUDA_MPI
cd examples/example2

# Copie des fichiers de maillages dans le dossier courant
cp ../example1/*_Mille_Iles_mesh_481930_elts.txt .

# Copie du fichier de donnees dans le dossier courant
cp ../../src/cuteflow/donnees.f .

# Copie de l'executable de CUTEFLOW dans le dossier courant
cp ../../bin/cuteflow .

# Copie du fichier de lancement du code dans le dossier courant
cp ../../scripts/2_gpu.sh .
```

A ce stade le résultat de la commande `ls` dans le dossier **example2** devrait être le suivant :

```
0_Mille_Iles_mesh_481930_elts.txt
1_Mille_Iles_mesh_481930_elts.txt
2_gpu.sh
donnees.f
cuteflow
```

L'utilisateur doit ensuite modifier les paramètres de simulation dans le fichier **donnees.f**. Il peut éditer ce fichier avec `vim`, `emacs`, `nano` ou autre. Ce fichier est largement commenté et les paramètres seront probablement facile à identifier.

```

1 0DONNEES_NAMELIST
2
3 ! Données du terrain
4 GP=9.81,
5 is_override_manning=1, override_manning=0.02200, ! nombre de manning qui override les autes si is_...=1
6
7 ! Données du maillage
8 meshfile='Mille_Iles_mesh_481930_elts.txt', ! Fichier de maillage
9 elt_bound=0, ! 1 si le fichier boundary_table existe déjà
10 multi_entree=0, multi_sortie=0, ! 0 si fichier non formaté pour plusieurs entrées/sorties
11
12 ! Initialisation avec un Barrage
13 eqlin_barrage=1, ! 1 pour initialiser avec un barrage
14 x1eqbar=274997.75521489076, x2eqbar=274756.1533974407, ! Abscisses des deux points du barrage
15 y1eqbar=5043578.489765059, y2eqbar=5043885.982987268, ! Ordonnées des deux points du barrage
16 H_AMONT=30.0, U_AMONT=0, V_AMONT=0, ! Valeur de l'init du coté - de la normale au barrage
17 H_AVAL=29.0, U_AVAL=0, V_AVAL=0, ! Valeur de l'init du coté + de la normale au barrage
18
19 ! Initialisation avec un plan
20 plan=0 ! 1 pour initialiser avec un plan
21 xplan=-0.0001, yplan=-0.0001, zplan=1 ! Vecteur normal au plan
22 xpoint=274956.783798834, ypoint=5043746.46205659, zpoint=31.35 ! Point appartenant au plan
23
24 ! Initialisation a partir d'un fichier, fichier avec solution ax elements
25 solinit=1, ! 1 pour initialiser a partir du fichier
26 fich_sol_init='sol_test.txt', ! nom du fichier d'initialisation
27
28 ! Conditions aux limites
29 inlet='inflow', ! Type d'entrée : {inflow,transm}
30 debitglob=800.0, ! Débit aux entrées, séparer les débits pas des ,
31 debit_var=0, loi_debitglob='Mille_iles_Qt.txt', ! Loi de debit fonctionne uniquement avec 1 entrée
32 H_sortie=29.0, ! Hauteur du niveau à la sortie du domaine
33
34 ! Paramètres des schémas numériques
35 IFLUX=2, ! 1 -> HLLC zoka, 2 -> HLLC Riadh
36 tolisec=1.0E-06, ! Tolérance sec/mouillé
37 timedisc='euler', ! Schéma en temps : {euler,second,runge}
38 friction=1, ! 1 pour prendre en compte la friction
39 fricimplic=1, ! 0 -> explicite, 1 -> I-dt/2*, 2 -> I-dt*B
40 is_dry_as_wall=0, ! 1 pour mettre les mailles seches comme des murs
41 local_time_step=0, ! 1 pour utiliser le local time step
42
43 TS=300.0, CFL=0.9, ! Temps maximal de simulation, nombre CFL
44 tol_reg_perm=1.0E-15, ! Tolérance relative entre débit entrée et débit sortie
45 freqaffich=5000, ! Frequence de print dans outfile.[0-9]
46
47 ! Sauvegarde de la solution
48 merged_solution=0, ! Ecriture de la solution recombinaées (lent)
49 nbrjauges=0, jauges_snapshots=100, ! Nombre de jauges, nombre de snapshots
50 xjauges = 272494.31, 274767.53, 279066.02, ! Abscisses des jauges
51 yjauges = 5042222.41, 5046515.26, 5050380.38, ! Ordonnées des jauges
52
53 nbrcoupes=0, coupes_snapshots=100, ! Nombre de coupes, nombre de snapshots
54 coupe_a = -0.7946, -2.6754, -1.4439 ! Coefficient a de ax+by
55 coupe_b = 5.2585e+06, 5.7768e+06, 5.4397e+06 ! Coefficient b de ax+by
56
57 solrestart=1, restart_snapshots=2, ! Sauvegarde en overwrite la solution pour restart
58 solsimple=3, simple_snapshots=10, ! 1 noeuds, 2 éléments, 3 noeuds+elements
59 solvtk=1, vtk_snapshots=10, ! 1 pour sauvegarder les fichiers vtk pour video
60 sortie_finale_bluekenue=1/ ! Sauvegarde fichiers T35 à la fin

```

FIGURE 2 – Exemple 2 : Aperçu du fichier **donnees.f** édité avec vim

La seule chose à noter ici par rapport au fichier **donnees.f** est que le nom du fichier de maillage doit être le nom de base du fichier, ici `Mille_Iles_mesh_481930_elts.txt`, le code se chargera d’aller lire les fichiers prefixés par le numéro du process MPI qui doit les traiter. On décrit chaque paramètre de ce fichier en annexe.

Il y a ensuite deux façon de lancer une simulation, soit en lancer un job, soit en demandant un noeud interactif.

On peut demander un noeud interactif de la façon suivante,

```
salloc --time=0-01:00 --nodes=1 --ntasks-per-node=2 --gres=gpu:2 --mem-per-cpu=4000M --account=rrg-soulaima-ac
```

On demande avec cette commande un noeud de calcul avec 2 GPU, 2 CPU et 4 Go de mémoire par CPU pour 1 heure. Ici on spécifie comme groupe `rrg-soulaima-ac` uniquement lorsque l’on est sur CEDAR pour utiliser des GPU comme c’est l’attribution de ressources qui a été donnée au groupe de recherche. On peut aussi utiliser le groupe `def-soulaima` mais on aura une priorité et une quantité de ressources alloués moins importante.

Il se peut qu’il faille attendre un petit moment pour que le noeud de calcul nous soit alloué. Une fois que le noeud de calcul nous a été alloué, on peut lancer la simulation avec les commandes suivantes,

```

#Chargement des modules
module load pg/19.4 cuda/10.0.130 openmpi/3.1.2

#Lancement de la simulation
mpirun --mca pml ob1 --mca btl openib -n 2 ./cuteflow

```

De cette façon tous les process MPI vont faire une sortie directement dans le terminal, ce n’est souvent pas ce qu’on veut, on peut alors remplacer la dernière commande par,

```
mpirun --mca pml ob1 --mca btl openib -n 2 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

De cette façon aucun process de fera de sortie dans la console, ils feront chacun leur sorties dans les fichiers **outfile** suffixé par leur numéro de process, par exemple **outfile.0** pour la sortie du process 0. On peut monitorer l’avancement de la simulation dans un de ses fichiers avec la commande

```
tail -f outfile.0
```

qui permettra de voir tous les ajouts qui sont faits dans le fichiers de manières interactive.

Si on suit cette façon de faire (demander un noeud interactif) on ne devrait rien voir dans le terminal une fois la commande

```
mpirun --mca pml ob1 --mca btl openib -n 2 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

lancée, si des erreurs sont présentes c'est ici qu'on les verra.

Une fois la simulation terminée on peut regarder avec un éditeur de texte le fichier **outfile.0** par exemple pour s'assurer que tout c'est bien passé. Dans ce cas, avec le fichier de donnée montré plus haut on obtient pour le fichier **outfile.0** de la figure 3

Alternativement, on peut soumettre le code à l'ordonnanceur, après avoir modifier le fichier **2\_gpu.sh** pour correspondre à ses besoins, avec la commande suivante :

```
# En étant dans le dossier 481930_2gpu
sbatch 2_gpu.sh
```

On peut regarder si le code est lancé et/ou à quel moment il le sera avec la commande

```
squeue -u $USER
```

De la même façon que précédemment, on peut suivre l'avancement en monitorant les fichiers outfile.[0-9] qui servent de stdout pour chaque process MPI.

Une fois la simulation terminée, selon les paramètres mis dans *donnees.f*, plusieurs fichiers résultats seront générés en particulier les fichiers \*.vtk si le paramètre video a été mis à 1 comme c'est le cas par défaut ici. Un fichier résultat est créé par sous domaine, le domaine 0 correspond aux fichiers 0\_FV-Paraview\_\*.vtk. Ces fichiers peuvent ensuite être visualisés à l'aide de Paraview voir section 10

## 6.1 Lancement de la simulation à partir d'une solution déjà générée

Une simulation peut être redémarrée à partir de n'importe quelle solution enregistrée sur les éléments. En particulier, si l'utilisateur ne veut pas générer ces solutions tout le long de la simulation, il peut mettre le paramètre *solrestart* à 1 dans le fichier *donnees.f*. Le paramètre *restart\_snapshot* sert à spécifier le nombre de fois où le fichier solution \*\_solution\_elements\_restart.txt sera re-écrit. Si on spécifie *restart\_snapshot* = 1 le fichier ne sera écrit qu'une fois à la fin de la simulation. Il peut être intéressant de spécifier un nombre plus grand que 1 au cas où la simulation plante en cours de route sans avoir encore atteint le temps final. Si on spécifie *restart\_snapshot* = 10 le fichier sera re-écrit 10 fois au cours de la simulation à intervalle de temps constant.

Une fois les fichiers \*\_solution\_elements\_restart.txt générés si on veut faire repartir la simulation sur ces fichiers il faudra les renommer puis mettre leur nom de base en paramètre dans *fich\_sol\_init* dans le fichier *donnees.f*.

Dans le cas de l'exemple 2, on peut par exemple mettre *solrestart* = 1 et *restart\_snapshots* = 1 dans le fichier *donnees.f*. Ensuite une fois une première simulation finie, les fichiers 0\_solution\_elements\_restart.txt et 1\_solution\_elements\_restart.txt seront générés. Il suffit ensuite de les renommer, en faisant par exemple

```
mv 0_solution_elements_restart.txt 0_sol_example2.txt
mv 1_solution_elements_restart.txt 1_sol_example2.txt
```

Il faudra alors ensuite mettre le paramètre *solinit* = 1 et *fich\_sol\_init* = 'sol\_example2.txt' pour initialiser la solution avec ces fichiers.

## 6.2 Format des fichiers solutions

Plusieurs fichiers solutions peuvent être générés pendant les simulations. Le premier type de fichier correspond à l'option *sol\_vtk* dans le fichier de donnée et permet d'écrire des fichiers au format .vtk pour la lecture avec Paraview. Ces fichiers contiennent la description du maillage sous forme d'une *UNSTRUCTURED\_GRID* et contiennent des inconnues aux points de ce maillage. En particulier, ces fichiers contiennent 3 scalaires, *h* la hauteur d'eau, *eta* la hauteur de la surface libre, *b* la hauteur de la bathymétrie (*eta* = *b* + *h*), et un vecteur *velocity* qui correspond à la vitesse (On parle bien de la vitesse  $U = (u,v)$  et non pas des flux ( $hu, hv$ )).

On peut en plus générer des fichiers correspondants à l'option *sol\_simple* dans le fichier de donnée. Ces fichiers contiennent à chaque fois 4 colonnes représentant dans l'ordre, *h* la hauteur d'eau, *eta* la hauteur de la surface libre, *hu* flux dans la direction x, *hv* flux dans la direction y. Selon l'option que l'on choisit pour *sol\_simple*, on peut avoir ces fichiers soit aux noeuds du maillage avec l'option 1, soit aux centre des mailles avec l'option 2, soit les deux avec l'option 3.

Les fichiers solutions liés à l'option *sol\_restart* ont exactement le même format que les fichiers de *sol\_simple* aux éléments. La différence est qu'ils sont écrasés et mis à jour durant la simulation plutôt que successivement écrits.



```

1 Device name: Tesla P100-PCIE-12GBthread id : 0
2 Compute capability : 6.0
3 0 lecture du maillage en cours
4 id 0 , 122191 nodes
5 id 0 , 241151 elms
6 id 0 , 210 noeuds d'entree
7 id 0 , 0 noeuds de sortie
8 id 0 , 2741 noeuds de murs
9 id 0 , 185 mailles fantomes a recep
10 id 0 , 185 mailles fantomes a envoyer
11 id 0 , 1 bloc fantomes a receptionner
12 240967 185 1
13 id 0 , 1 bloc fantomes a envoyer
14 240782 185 1
15 0 lecture du maillage reussie
16 gridx, gridy (for set_boundary) = 15072 15072
17 =====
18 Parameters
19 =====
20 meshfile =
21 0_Mille_Iles_mesh_481930_elts.txt
22 cotemin = 0.1529265774476539
23 elt_bound = 0
24 H_AMONT = 30.000000000000000
25 U_AMONT = 0.000000000000000
26 V_AMONT = 0.000000000000000
27 H_AVAL = 29.000000000000000
28 U_AVAL = 0.000000000000000
29 V_AVAL = 0.000000000000000
30 iflux = 2
31 Friction = 1
32 jauges_snapshots = 100
33 nbrjauges = 0
34 nbrcoupes = 0
35 solrestart = 0
36 restart_snapshots = 10
37 solvtk = 1
38 vtk_snapshots = 10
39 tol_reg_perm = 1.000000000000001E-015
40 tol = 9.9999999747524271E-007
41 tolisec = 1.000000000000000E-008
42 tolaffiche = 0.100000014901161
43 freqaffich = 1000
44 dry_as_wall = 0
45 local time step = 0
46 nombre entree = 1
47 debitglob 1 = 800.0000000000000
48 longueur entree 1 = 1672.08232559227
49 nombre sortie = 1
50 H_SORTIE 1 = 29.000000000000000
51 =====
52 ***** FULL_FV *****
53
54 FULL-ORDER : VOLUMES FINIS
55
56 Loop over time starts
57 -----
58 nt = 1000
59 prochain tsolvtk = 90.0000000000000 3
60 tc = 70.39981172647879 Secondes => 1.173330195441313
61 Minutes
62
63 debit_entree 1 = 799.9999999999994
64 debit_sorti = 0.000000000000000 M3/S
65 -----
66 nt (end of loop on time) = 1295
67 Time taken by time loop (parallel) = 3412.849 ms
68 =====
69
70 ===== FIN DE LA SIMULATION =====
71 =====
72
73 DUREE DU CALCUL : 3.420150041580200 Secondes
74 =====
75

```

**NORMAL** outfile.0

FIGURE 3 – Exemple 2 : fichier **outfile.0**

## 7 Lancement de plusieurs simulations

Dans la version précédente du code, on pouvait utiliser le paramètre `multi_simul` pour spécifier qu'il fallait faire plusieurs simulations. Les paramètres de chaque simulation étaient donnés dans un fichier parfois nommé `INPUT_MONTE_CARLO.dat` dans lequel on avait une ligne de paramètre pour chaque simulation. En particulier on devait spécifier, `debit_glob`, `hamont`, `haval` et le `manning`. Avec cette méthode les simulations se lançaient les unes à la suite des autres. L'objectif de cette section est de présenter la procédure pour lancer toutes ces simulations en même temps sur les clusters de calculs à l'aide de `job arrays` et d'un script créé pour l'occasion.

L'objectif de cette méthode est de lancer le script `multi.sh` qui va créer un dossier par simulation en y copiant tous les fichiers nécessaires à l'exécution du code et en modifiant le fichier de donnée pour correspondre au cas voulu pour chaque simulation. Ces modifications sont faites par le script `gen_donnees.sh` qui génère le fichier de donnée. Pour modifier les paramètres de la simulation il faut modifier le fichier `gen_donnees.sh`. Plus généralement l'utilisateur peut vouloir modifier d'autres paramètres de simulation et il devra modifier les scripts `multi.sh` et `gen_donnees.sh`.

Le long de la section on prend comme exemple le lancement de plusieurs simulations dans le dossier **examples/example3** en utilisant les fichiers de maillages qui ont été découpés en section 5.

### 7.1 Génération des fichiers de données

On a besoin d'un moyen de générer les fichiers de données automatiquement à partir des valeurs de `debitglob`, `hamont`, `haval` et `manning`. C'est le but du script `gen_donnees` qui est utilisé à l'intérieur du script `multi`. On n'a pas normalement à l'utiliser tout seul mais on décrit quand même son utilisation simple.

Pour générer les fichiers de données on fait,

```
#Pour debit_global=1000 hamont=31 haval=30 manning=0.04
./gen_donnees 1000 31 30 0.04
```

Le fichier de donnée `donnees.f` sera alors créé et il contiendra bien les paramètres passés en argument. Attention, dans la configuration actuelle la valeur de `haval` sera aussi utilisé comme `hsortie`. Ce script peut être trouvé dans le dossier **scripts** et l'utilisateur sera sûrement amené à le modifier pour l'utilisation qu'il désire en faire. Ce script sera appelé par le script `multi.sh` avec en argument chacune des lignes du fichier d'input, on explique cela dans la suite.

### 7.2 Préparation du dossier `base_files`

Avant de lancer le script `multi.sh` il faut préparer les fichiers dont le code aura besoin. Pour cela il faut créer un dossier `base_files` dans lequel on va mettre ces fichiers. Les fichiers nécessaires sont les fichiers de maillages, l'exécutable `cute_flow`, on peut en plus ajouter les fichiers de liens et l'exécutable `merge_solutions` si on veut pouvoir recombinaison les solutions dans la suite. On procèdera alors de la façon suivante,

```
#Depuis le dossier CUTEFLOW_CUDA_MPI
cd examples/example3

#Copie des scripts dans le dossier courant
cp ../../scripts/multi.sh .
cp ../../scripts/gen_donnees.sh .

#Creation du dossier base_files et copie des fichiers de maillages
mkdir base_files
cp ../example1/*_Mille* base_files/
cp ../../bin/cute_flow base_files/

#Pour regrouper les fichiers solutions on aura besoin de certains fichiers
cp ../example1/*_lien* base_files/
cp ../../bin/merge_solutions base_files/
```

Suite à ces opérations le résultat de la commande `tree` lancée dans le dossier **examples/example3** devrait être conforme à la figure 4. (On note que le dossier **example3** devait déjà contenir le fichier `INPUT_MONTE_CARLO.dat`)

```
delmasv@cedar1:example3$ tree
.
├── base_files
│   ├── 0_liens_elems.txt
│   ├── 0_liens_nodes.txt
│   ├── 0_Mille_Iles_mesh_481930_elts.txt
│   ├── 1_liens_elems.txt
│   ├── 1_liens_nodes.txt
│   ├── 1_Mille_Iles_mesh_481930_elts.txt
│   ├── cuteflow
│   └── merge_solutions
├── gen_donnees.sh
├── INPUT_MONTE_CARLO.dat
└── multi.sh

1 directory, 11 files
```

FIGURE 4 – Exemple 3 : Sortie de la commande *tree*

Seul le fichier *cuteflow* et les fichiers contenant le mot *resart* dans leur nom seront copié dans les dossier **multi**. L'utilisateur peut changer ce comportement en modifiant le script *multi.sh*.

### 7.3 Lancement du script *multi.sh*

Le script *multi.sh* lit un fichier d'input et crée pour chaque ligne (pour chaque cas de simulation) un dossier *multi\_\**[0-9]. Il copie dans ce dossier les fichiers du dossier *base\_files* et génère les fichiers de donnée grâce au script *gen\_donnees*. Le fichier *INPUT\_MONTE\_CARLO.dat* nous sert de fichier d'input pour lancer plusieurs simulations dans le dossier *examples/example3*.

```
1 5
2 800.00    29.000000    29.000000    0.020000
3 1000.00   30.000000    29.000000    0.020000
4 1200.00   31.000000    29.000000    0.020000
5 1400.00   32.000000    29.000000    0.020000
6 1600.00   33.000000    29.000000    0.020000
```

FIGURE 5 – Exemple 3 : Fichier *INPUT\_MONTE\_CARLO.dat*

On voit dans ce fichier que 5 simulations seront lancées, pour comprendre ce à quoi chaque colonne correspond il faut aller voir le script *gen\_donnees.sh* qui génère à partir de chaque ligne le fichier de donnée correspondant. Dans ce cas exemple, la première colonne correspond au débit d'entrée, la seconde à la hauteur amont, la troisième à la hauteur aval et à la hauteur de sortie, et la dernière correspond au nombre de Manning global dans le domaine.

Dans le dossier **examples/example3** on peut lancer ce script avec la commande,

```
./multi.sh INPUT_MONTE_CARLO.dat
```

```
delmasv@cedar1:example3$ ./multi.sh INPUT_MONTE_CARLO.dat
Cas 1 800.00 29.000000 29.000000 0.020000
Cas 2 1000.00 30.000000 29.000000 0.020000
Cas 3 1200.00 31.000000 29.000000 0.020000
Cas 4 1400.00 32.000000 29.000000 0.020000
Cas 5 1600.00 33.000000 29.000000 0.020000
```

FIGURE 6 – Exemple 3 : Lancement du script *multi.sh*

Les différents dossiers *multi\_\**[0-9] seront créés les uns à la suite des autres. On peut ensuite vérifier les paramètres de simulation par exemple dans *multi\_1*. Si un paramètre doit être changé, par exemple le temps final de simulation, plutôt

que de le changer à la main dans chacun des dossier de simulation, il suffit de le changer dans le script *gen\_donnees.sh* puis de relancer le script *multi.sh* comme indiqué précédemment. De cette façon la modification s'appliquera dans tous les dossiers.

On peut vérifier la bonne structure du dossier **examples/example3** suite à l'exécution du script *multi.sh* en executant la commande *tree* dans ce dossier, la sortie devrait être identique à la figure 7.

```
delmasv@cedar1:example3$ tree
.
├── array.sh
├── base_files
│   ├── 0_Mille_Iles_mesh_481930_elts.txt
│   └── 1_Mille_Iles_mesh_481930_elts.txt
├── cuteflow
├── donnees.f
├── gen_donnees.sh
├── INPUT_MONTE_CARLO.dat
├── multi_1
│   ├── cuteflow
│   └── donnees.f
├── multi_2
│   ├── cuteflow
│   └── donnees.f
├── multi_3
│   ├── cuteflow
│   └── donnees.f
├── multi_4
│   ├── cuteflow
│   └── donnees.f
├── multi_5
│   ├── cuteflow
│   └── donnees.f
└── multi.sh

6 directories, 18 files
```

FIGURE 7 – Exemple 3 : *tree* une fois le script *multi.sh* lancé.

## 7.4 Lancement du job array

On a créé dans la partie précédente la structure des dossiers pour lancer toutes les simulations spécifiées dans le fichier *INPUT\_MONTE\_CARLO.dat*, il faut maintenant lancer toutes ces simulations sur le cluster de calcul. Dans ce but on utilise un job array. Un job array est un moyen rapide de lancer plusieurs jobs avec les mêmes paramètres, la description en est fait sur le wiki de Compute Canada [https://docs.computecanada.ca/wiki/Job\\_arrays](https://docs.computecanada.ca/wiki/Job_arrays). Le fichier que nous allons utiliser dans ce cas est le fichier *array.sh* présent dans le dossier **script**.

```
1 #!/bin/bash
2 #SBATCH --nodes=1
3 #SBATCH --ntasks-per-node=2
4 #SBATCH --gres=gpu:2
5 #SBATCH --mem=4000M
6 #SBATCH --time=0-00:10
7 #SBATCH --account=def-soulaima
8 #SBATCH --array=1-5
9
10 module load pgi/19.4 cuda/10.0.130 openmpi/3.1.2
11
12 cd multi_${SLURM_ARRAY_TASK_ID}
13 mpirun --mca pm1 ob1 --mca btl openib -n 2 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

FIGURE 8 – Exemple 3 : Fichier *array.sh*

Alternativement sur CEDAR on peut utiliser *array\_cedar.sh* ou le nom du groupe à été changé pour utiliser l'attribution de ressources qui à été donné au groupe de recherche.

```

1 #!/bin/bash
2 #SBATCH --nodes=1
3 #SBATCH --ntasks-per-node=2
4 #SBATCH --gres=gpu:2
5 #SBATCH --mem=4000M
6 #SBATCH --time=0-00:10
7 #SBATCH --account=rrg-soulaima-ac
8 #SBATCH --array=1-5
9
10 module load pgi/19.4 cuda/10.0.130 openmpi/3.1.2
11
12 cd multi_${SLURM_ARRAY_TASK_ID}
13 mpirun --mca pml ob1 --mca btl openib -n 2 sh -c './cuteflow > outfile.${OMPI_COMM_WORLD_RANK}'

```

FIGURE 9 – Example 3 : Fichier *array\_cedar.sh*

Ce fichier décrit un job qui utilise un noeud de calcul avec 2 gpu pour une durée de 10mn. Un paramètre important est le paramètre array qui décrit le nombre de jobs à lancer. Dans cet exemple on lance 5 jobs qui seront numérotés de 1 à 5. On se sert de ce numéro qui sera stocké dans la variable d'environnement `SLURM_ARRAY_TASK_ID` pour se déplacer dans le dossier adéquat avant de lancer le code. Pour soumettre ce job array à l'ordonnanceur il suffit ensuite de faire

```
sbatch array.sh
```

Pour lancer les 5 simulations du dossier **examples/example3** il faut donc faire,

```

#Depuis le dossier CUTEFLOW_CUDA_MPI
cd examples/example3

#Copie du script array.sh ou array_cedar.sh
cp ../../scripts/array.sh .

#Lancement du job array
sbatch array.sh

```

Les 5 simulations seront alors lancées sur le cluster de calcul. On peut savoir si les simulations sont bien lancées en faisant la commande `squeue -u $USER`

```

delmasv@cedar1:example3$ sbatch array_cedar.sh
Submitted batch job 49026346
delmasv@cedar1:example3$ squeue -u $USER

```

JOBID	USER	ACCOUNT	NAME	ST	TIME_LEFT	NODES	CPUS	TRES_PER_N	MIN_MEM	ODELIST	(REASON)
49026346_1	delmasv	rrg-soulaima	array_cedar.sh	R	9:49	1	2	gpu:p100:2	4000M	cdr346	(None)
49026346_2	delmasv	rrg-soulaima	array_cedar.sh	R	9:49	1	2	gpu:p100:2	4000M	cdr385	(None)
49026346_3	delmasv	rrg-soulaima	array_cedar.sh	R	9:49	1	2	gpu:p100:2	4000M	cdr249	(None)
49026346_4	delmasv	rrg-soulaima	array_cedar.sh	R	9:49	1	2	gpu:p100:2	4000M	cdr252	(None)
49026346_5	delmasv	rrg-soulaima	array_cedar.sh	R	9:49	1	2	gpu:p100:2	4000M	cdr258	(None)

```

delmasv@cedar1:example3$

```

FIGURE 10 – Example 3 : Commande `squeue -u $USER`

Dans le cas de la figure 10 les 5 simulations se sont toutes lancées et il reste 9 min 49 s de temps maximal pour leur exécution. Si dans la colonne **ST** (Status) il est indiqué **PD** (Pending) c'est que les simulation ne se sont pas encore lancées et il faut simplement attendre. Encore une fois pour ce qui concerne l'utilisation des clusters de calcul l'utilisateur peut se référer au wiki officiel de *Compute Canada*.

## 7.5 Récupération des fichiers résultats dans un seul dossier (Optionnel)

Une fois le job array soumis et terminé, chaque simulation et chaque GPU va avoir généré ses propres fichiers de résultats dans son propre dossier. Le script `gen_results` utilise le code `merge_solutions` décrit en section 8 pour regrouper les fichiers solution de chaque gpu en un unique fichier résultat pour chaque simulation.

Dans le cas de l'exemple 3, pour regrouper les fichiers solutions générés il faut faire,

```

#Depuis le dossier CUTEFLOW_CUDA_MPI
cd examples/example3

#Copie du code merge solution et du script gen_result
cp ../../bin/merge_solutions

```

```
cp ../../scripts/gen_results.sh
```

```
#Lancement du script pour regrouper les fichiers generes avec 2 GPU
./gen_results INPUT_MONTE_CARLO.dat 2
```

En plus de créer un unique fichier pour chaque simulation, dans sa configuration actuelle, un dossier **results** va être créé. Dans ce dossier un fichier solution global GLOBAL\_SOL\_FV\_MULTISIM\_3D.txt va être créé, contenant les résultats de simulation SOL3D pour chaque simulation les uns à la suite des autres. Un nouveau fichier INPUT\_MONTE\_CARLO.dat va être créé correspondant au cas présents dans le fichier GLOBAL\_SOL\_FV\_MULTISIM\_3D.txt. C'est pour corriger le fait que parfois certaines simulations ne se finissent pas, certains résultats ne sont donc pas générés et ne sont pas au final dans GLOBAL\_SOL\_FV\_MULTISIM\_3D.txt. Le fichier INPUT\_MONTE\_CARLO.dat contenu dans le dossier **results** est donc emputé des cas qui ne se sont pas finis et correspond donc bien aux résultats présents dans le fichier GLOBAL\_SOL\_FV\_MULTISIM\_3D.txt.

## 8 Combinaison des fichiers de solution

Lorsque qu'une simulation se termine elle génère plusieurs fichiers qui peuvent pour simplifier le post-traitement être combiné en un unique fichier. Pour ce faire, il faut utiliser le code *merge\_solutions* présent dans le dossier **bin** après compilation. Il y a deux types de fichiers à recombinaison, ceux qui listent les inconnues par noeuds et ceux qui les listent par éléments. Le code *merge\_solution* peut donc être lancé soit avec le paramètre "noeuds" soit "elements" selon le type de fichier à recombinaison et nécessite les fichiers de lien entre les numérotation locales et globales correspondants.

On présente dans la suite comment recombinaison les fichiers résultats générés lors de l'exemple 2.

```
# Utilisation : ./merge_solutions {noeuds/elements} nom_de_base_fichiers nGPU
./merge_solutions noeuds solution_noeuds_simple_10.txt 2
./merge_solutions elements solution_elements_simple_10.txt 2
```

**ATTENTION IL FAUT QUE LES FICHIERS \*\_lien\_nodes.txt ET \*\_lien\_elems.txt SOIENT PRÉSENTS DANS LE DOSSIER COURANT**

La première commande va recombinaison les fichiers *0\_solution\_noeuds\_simple\_10.txt* et *1\_solution\_noeuds\_simple\_10.txt* en un unique fichier *solution\_noeuds\_simple\_10.txt* en utilisant les tables de correspondances *0\_liens\_nodes.txt* et *1\_liens\_nodes.txt*.

La seconde va faire la même chose pour les fichiers de solutions sur les éléments et en lisant les tables de correspondances pour les éléments.

## 9 Combinaison des fichiers de solution pour Bluekenue

Dans le dossier *bin*, on peut trouver après la compilation le code *merge\_bluekenue* qui permet de combiner des fichiers \*.T3S générés par CUTEFLOW en un unique fichier sur le domaine global. Au vu du temps pris par cette re-combinaison, ce code est uniquement utilisable si on veut combiner les résultats finaux d'une simulation et non re-combiner des fichiers générés à intervalle de temps régulier. Ce genre de fichiers sont générés si l'option *solution\_finale\_bluekenue* est mise à 1 dans le fichier *donnees.f*.

Pour fonctionner correctement le code a besoin des tables de correspondances entre la numérotation locale et la numérotation globale pour chaque sous domaine. **Ces fichiers sont créés lors de la découpe du fichier de maillage et il faudra les copier dans le dossier courant pour que merge\_bluekenue s'exécute correctement.** Voir section 5 pour plus d'information sur ces tables de correspondances, ce sont les fichiers nommés [0-9]\_liens\_nodes.txt et [0-9]\_liens\_elems.txt.

Le code est simple d'utilisation il suffit de lancer l'exécutable *merge\_bluekenue*, qui se trouve dans le dossier *merge\_bluekenue* après avoir compilé avec la commande *make*, avec comme premier argument le nom de base des fichiers T3S à combiner et en second argument le nombre de fichiers à combiner (qui correspond au nombre de GPU utilisés pour la simulation).

Par exemple pour combiner les fichiers *0\_solution-H-Bluekenue\_t\_245.759s.T3S* et *1\_solution-H-Bluekenue\_t\_245.759s.T3S* :

```
# Utilisation : ./merge_bluekenue nom_de_base_fichiers nGPU
./merge_bluekenue solution-H-Bluekenue_t_245.759s.T3S 2
```

Le fichier FV-H-Bluekenue\_t\_245.759s.T3S sera créé lors de l'opération. Il faut répéter l'opération pour les autres fichiers de bluekenue ETA, U et V.

**ATTENTION IL FAUT QUE LES FICHIERS \*\_lien\_nodes.txt ET \*\_lien\_elems.txt SOIENT PRÉSENTS DANS LE DOSSIER COURANT**

Si une erreur de Segmentation Fault se produit lors de l'exécution c'est probablement que les tables de correspondances ne correspondent pas au maillage utilisé lors de la simulation.



## 10 Traitement dans Paraview des fichiers \*.vtk

Le logiciel Paraview [10.5555/2789330] est installé par défaut sur les grappes de calcul de Compute Canada et permet une utilisation client/serveur ce qui permet de visualiser de très larges solutions sans avoir à télécharger les données sur son ordinateur personnel. Un tutoriel détaillé est présent sur le site officiel [https://www.paraview.org/Wiki/The\\_ParaView\\_Tutorial](https://www.paraview.org/Wiki/The_ParaView_Tutorial) et l'utilisation en mode client/serveur sur les grappes de calcul de Compute Canada est faite sur leur wiki officiel <https://docs.computeCanada.ca/wiki/ParaView>.

### 10.1 Tunnel SSH pour la visualisation avec Paraview sur les clusters de calcul en utilisant MobaXterm

Pour faire de la visualisation à distance avec Paraview, il va falloir lancer un serveur Paraview sur un noeud de calcul du cluster et s'y connecter depuis son ordinateur. Pour s'y connecter la seule façon de procéder et d'utiliser un tunnel SSH pour forwarder le port 11111 local sur le port 11111 du noeud de calcul sur lequel on a lancé le serveur Paraview. De cette façon en faisant se connecter Paraview sur le port 11111 de sa machine locale, on se connectera en fait sur le noeud de calcul voulu.

Avant de faire un tunnel SSH, il faut demander un noeud de calcul sur le cluster et lancer un serveur Paraview dessus. Pour ce faire, deux scripts sont présents dans le dossier **scripts/**. Le premier *request\_visu.sh* permet simplement de faire une commande qui demande un noeud de calcul avec un certain nombre de CPU et de mémoire, ici 16 CPU avec 12Go de mémoire. Le script *load\_para\_cpu.sh* permet ensuite de lancer un serveur Paraview utilisant les 16 CPU demandé lors de l'étape précédente (on peut évidemment demander plus de CPU ou de mémoire si besoin).

```
#Depuis le dossier CUTEFLOW_CUDA_MPI
```

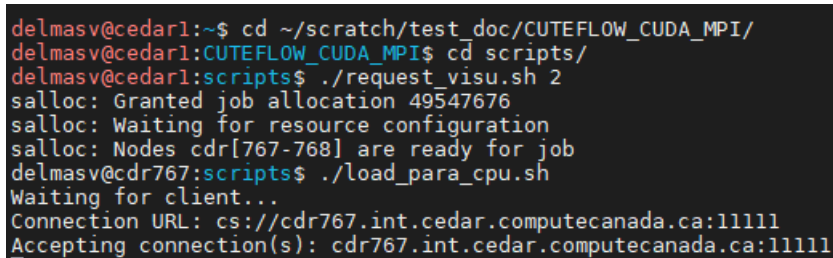
```
cd scripts/  
./request_visu.sh 2
```

Une fois le noeud de calcul attribué, faire

```
#Depuis le dossier CUTEFLOW_CUDA_MPI/scripts/
```

```
./load_para_cpu.sh
```

Cela peut prendre un peu de temps mais on devrait avoir un affichage proche de la figure suivante,



```
delmasv@cedar1:~$ cd ~/scratch/test_doc/CUTEFLOW_CUDA_MPI/  
delmasv@cedar1:CUTEFLOW_CUDA_MPI$ cd scripts/  
delmasv@cedar1:scripts$ ./request_visu.sh 2  
salloc: Granted job allocation 49547676  
salloc: Waiting for resource configuration  
salloc: Nodes cdr[767-768] are ready for job  
delmasv@cdr767:scripts$ ./load_para_cpu.sh  
Waiting for client...  
Connection URL: cs://cdr767.int.cedar.computecanada.ca:11111  
Accepting connection(s): cdr767.int.cedar.computecanada.ca:11111
```

FIGURE 11 – Lancement du serveur Paraview sur un noeud de calcul

Une fois arrivé au point de la figure 11 le serveur Paraview est lancé sur un noeud de calcul du cluster. L'important ici est de repérer que est le noeud de calcul sur lequel le serveur est lancé. On voit dans l'image que le noeud de calcul s'appelle *cdr767*, il est indiqué dans la partie droite de la variable PS1 (à droite du @) et c'est aussi la première partie de l'URL de connections que nous n'allons pas utiliser dans la suite. Sur CEDAR les noeuds s'appellent avec *cdr* en préfixe, sur BELUGA c'est *blg* et sur GRAHAM c'est *gra*. Une fois le noeud de calcul repéré on peut passer à l'étape suivante qui consiste à créer le tunnel SSH entre notre ordinateur et ce noeud sur le cluster de calcul. Pour ce faire on utilise MobaXterm.

MobaXterm propose une visuel qui montre très bien ce qu'il se passe voir figure suivante.

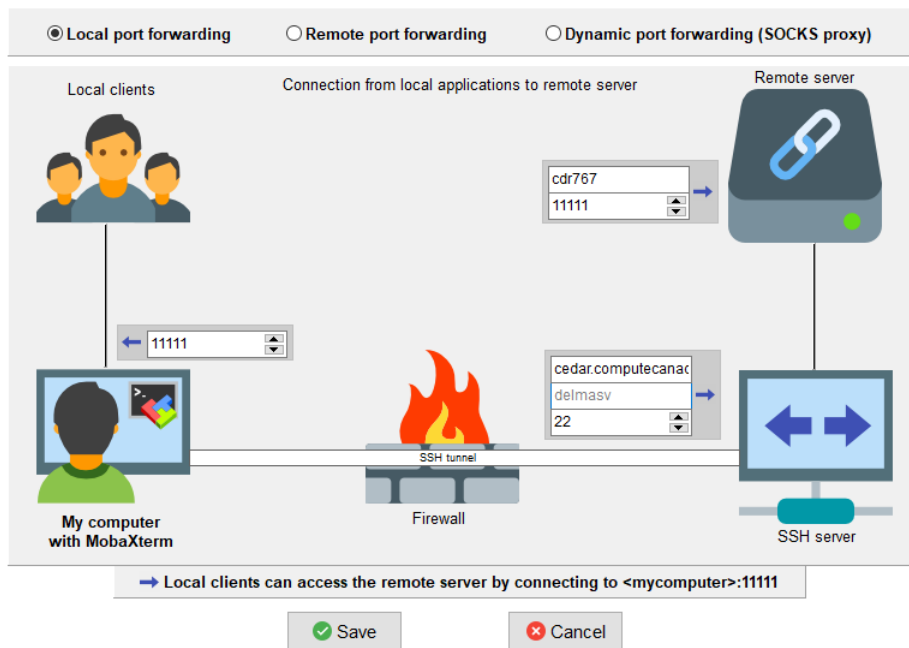


FIGURE 12 – MobaXterm tunnel SSH pour Paraview

Sur cette figure, on voit qu'on connecte le port 11111 de notre ordinateur via le serveur SSH *cedar.compute.canada.ca* sur le port 11111 du noeud de calcul *cdr767*. Il faut paramétrer cette fenêtre comme indiqué sur la figure en changeant simplement le nom du noeud de calcul sur lequel le serveur Paraview a été lancé. Une fois ces paramètres rentrés, il suffit de lancer le tunnel en cliquant sur le signe lecture dans MobaXterm voir figure suivante.

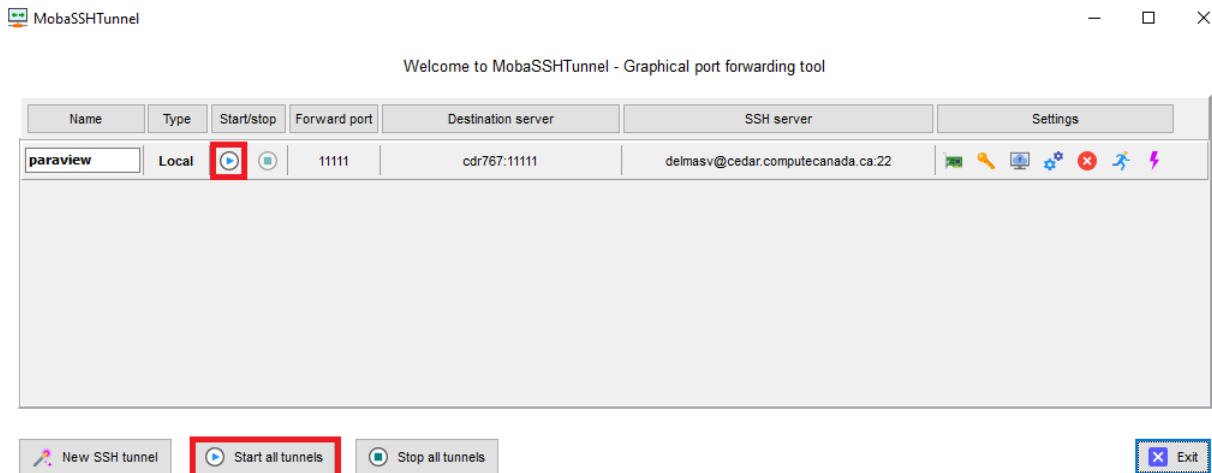


FIGURE 13 – Lancement du tunnel SSH dans MobaXterm

Une fois que le tunnel SSH a été lancé, il suffit de lancer Paraview sur son ordinateur et se connecter au port 11111 de son ordinateur local. Il faut cependant s'assurer d'utiliser la même version de Paraview que celle présente sur les clusters de calculs (loadé par le script *load\_para\_cpu.sh*) qui est la version 5.5.2 de Paraview téléchargeable gratuitement sur <https://www.paraview.org/download/>.

Une fois Paraview lancé, si c'est la première fois il faut créer la connexion à un serveur de la manière suivante.



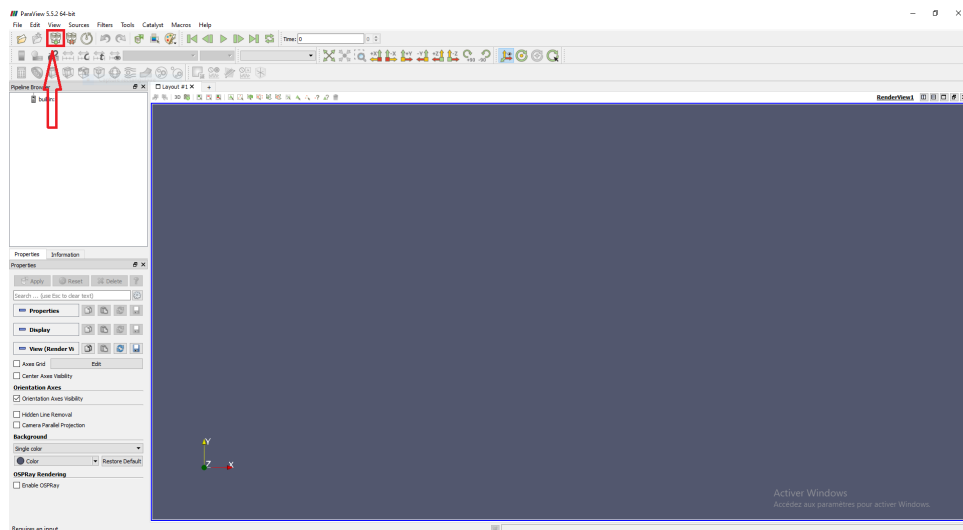


FIGURE 14 – Création de la connexion dans Paraview

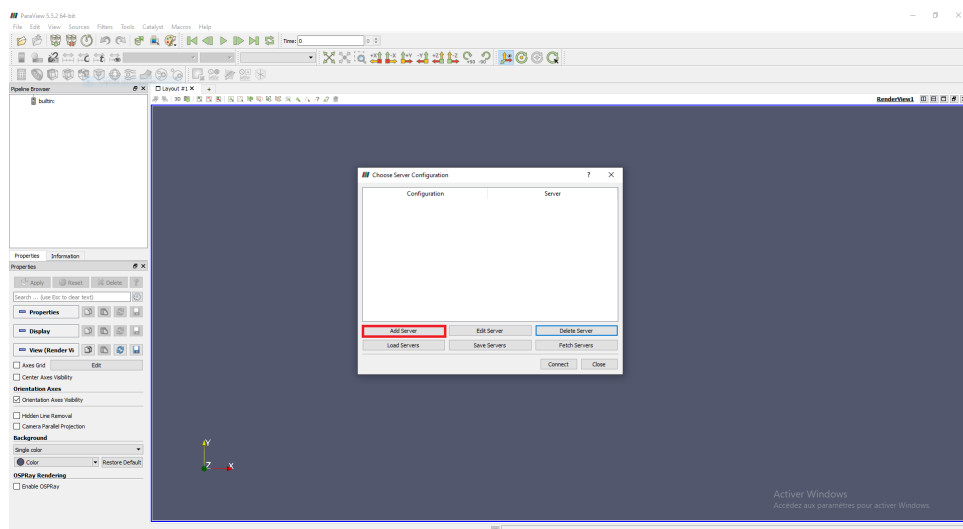


FIGURE 15 – Création de la connexion dans Paraview

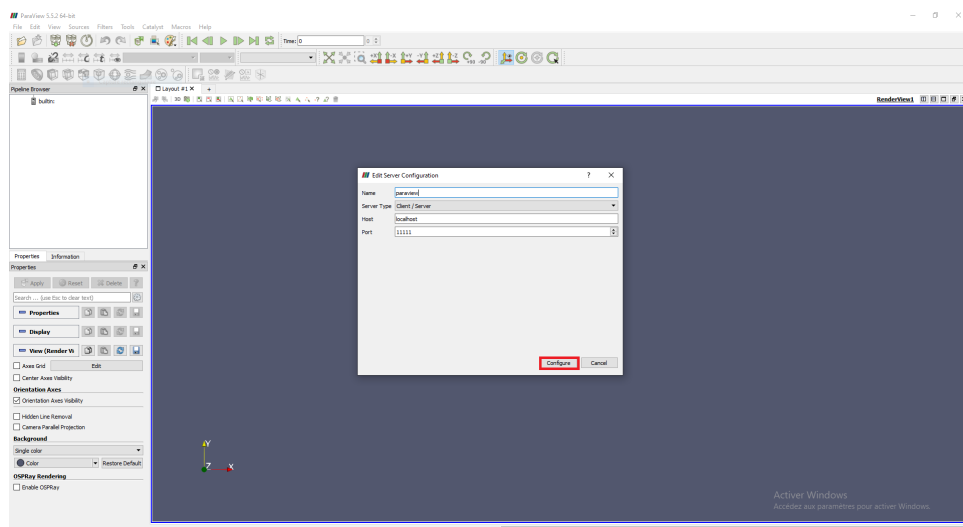


FIGURE 16 – Création de la connexion dans Paraview

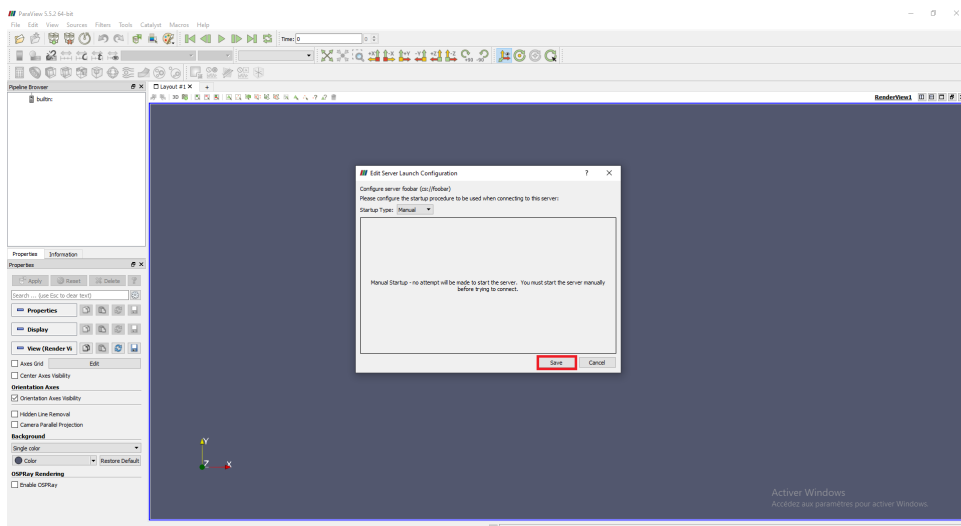


FIGURE 17 – Création de la connexion dans Paraview

A ce stade la configuration est finie, il suffit de cliquer sur connect pour se connecter a son port 11111 local qui est redirigé via le tunnel SSH sur le port 11111 du noeud de calcul sur le cluster sur lequel on a lancé le serveur Paraview un peu plus tôt.

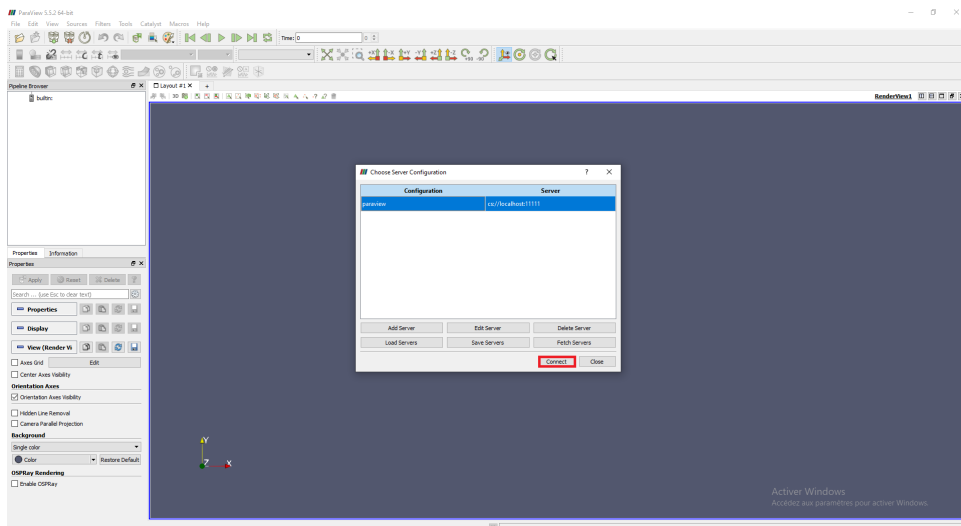


FIGURE 18 – Connexion au port 11111 de sa machine dans Paraview

Si tout ce passe bien il ne devrait pas y avoir d'erreurs une fois qu'on clique sur connect mais l'écran peut rester figer quelques instants. On devrait ensuite voir l'affichage changer dans la partie gauche de Paraview pour correspondre à l'image 20. On peut aussi vérifier que dans le terminal MobaXterm ou on avait lancé le serveur Paraview il devrait être indiqué *Client Connected*.

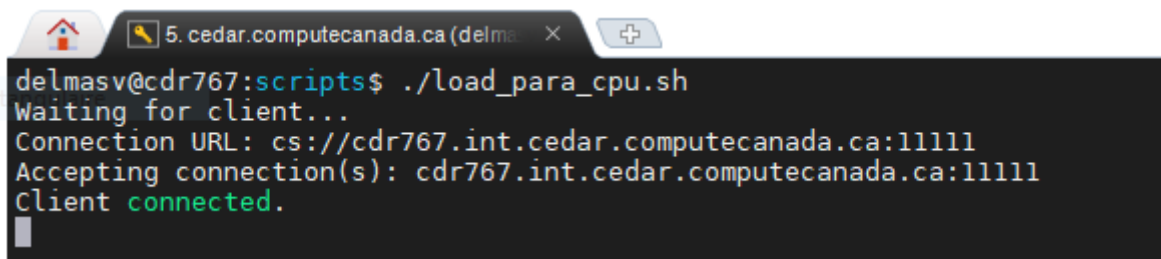


FIGURE 19 – Connexion au serveur Paraview réussie

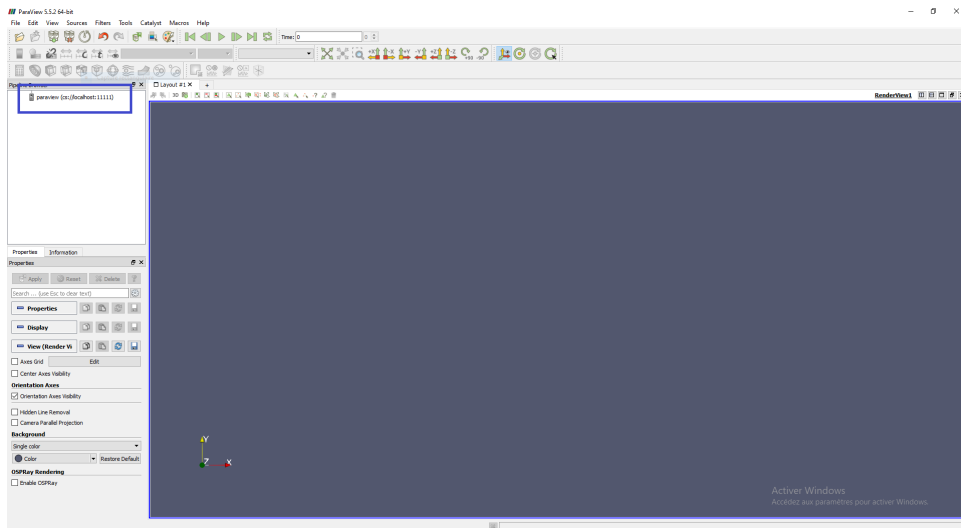


FIGURE 20 – Connexion réussie au serveur distant Paraview

## 10.2 Visualisation des fichiers résultats générés par CUTEFLOW sur Paraview

On présente ici une utilisation basique pour visualiser les fichiers \*.vtk produits par le code CUTEFLOW. On considère que les fichiers 0\_FV-Paraview\_9999.vtk et 1\_FV-Paraview\_9999.vtk ont déjà été téléchargés localement ou que l'utilisateur s'est déjà connecté au cluster de calcul via Paraview.

Dans Paraview sélectionner File > Open puis naviguer jusqu'aux fichiers, 0\_FV-Paraview\_9999.vtk et 1\_FV-Paraview\_9999.vtk les sélectionner puis cliquer sur ouvrir.

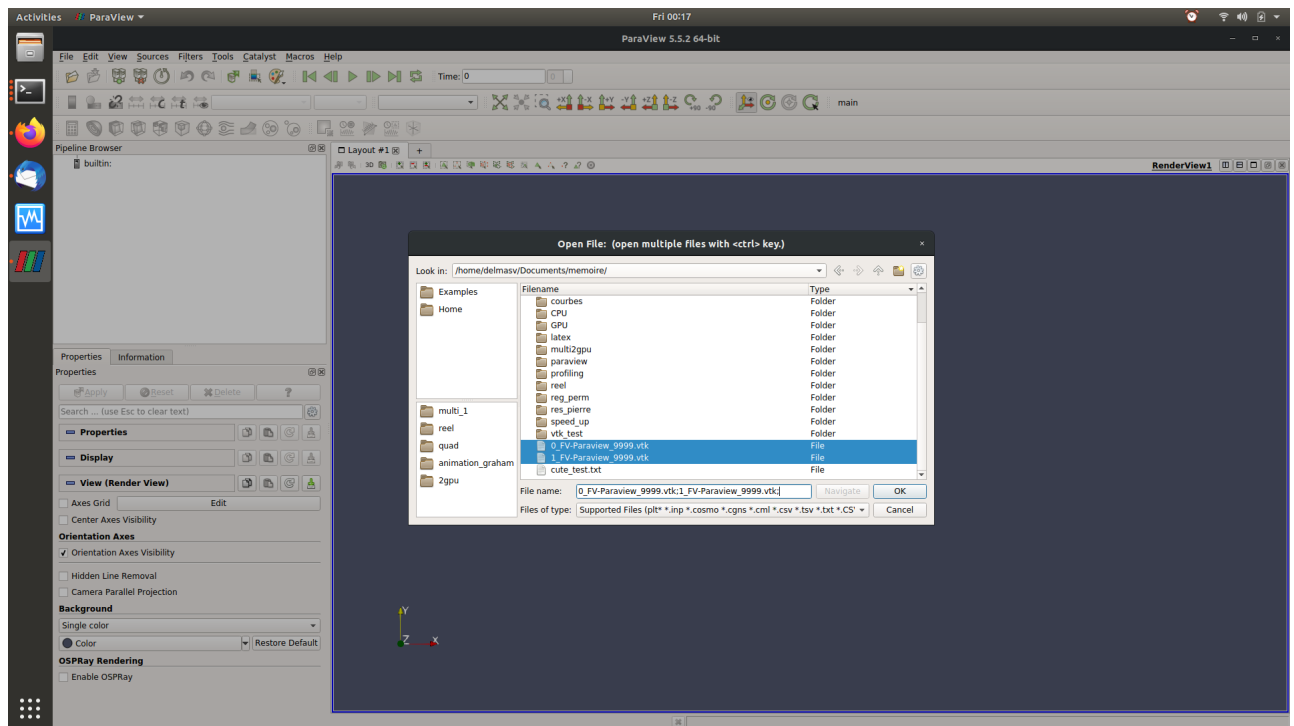


FIGURE 21 – Ouverture des fichiers

Une fois les fichiers présent dans la partie gauche de Paraview, il faut les sélectionner et cliquer sur Apply. On peut soit le faire une fois pour chaque fichier ou tous les sélectionner avec Ctrl-Clic puis cliquer sur Apply en les ayant tous sélectionner.

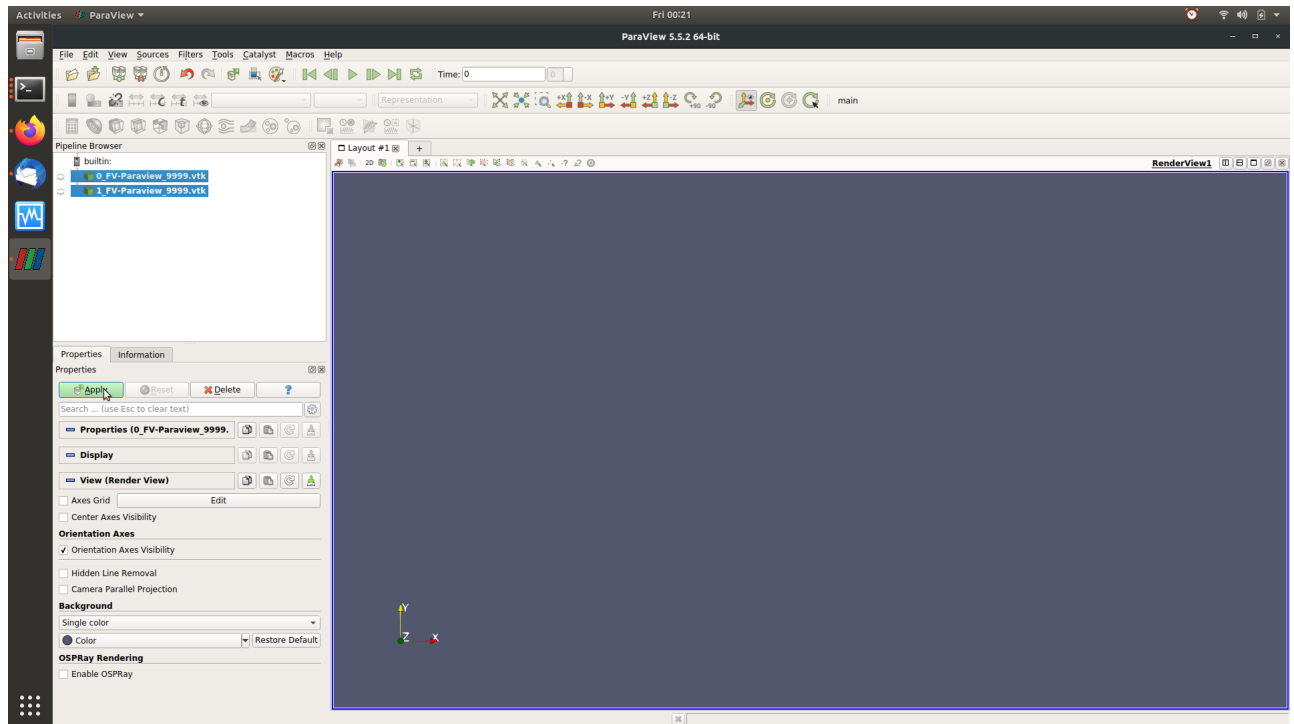


FIGURE 22 – Sélectionner les fichiers et cliquer sur Apply

Le domaine complet devrait apparaître dans la fenêtre de visualisation. On peut noter qu'on peut afficher ou cacher un domaine en cliquant sur l'oeil bleu à gauche du nom du fichier concerné dans la fenêtre de gauche.

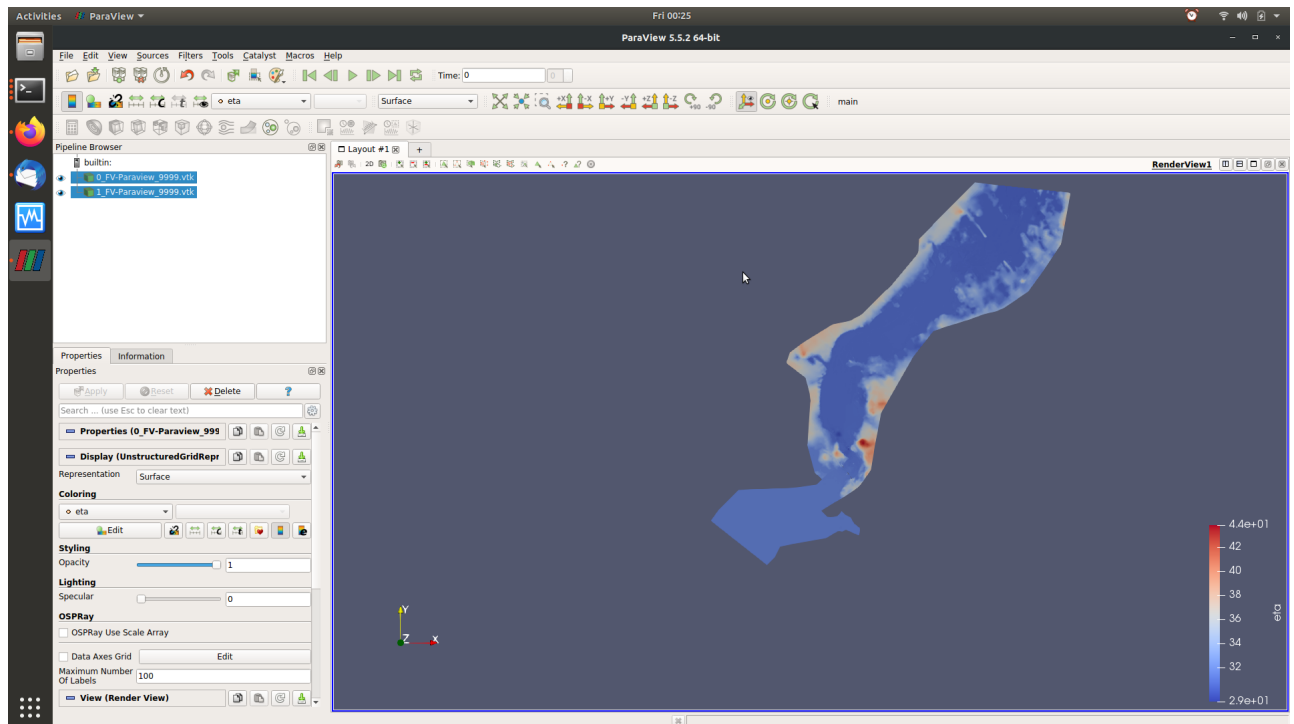


FIGURE 23 – Affichage du domaine

Avant de continuer le post-traitement il est pratique de fusionner les domaines en un seul grand domaine. Pour ce faire il suffit de sélectionner **les deux fichiers** dans la fenêtre de gauche puis d'aller dans Filters > Commons et de cliquer sur Group DataSets puis de cliquer sur Apply (comme après chaque modification).

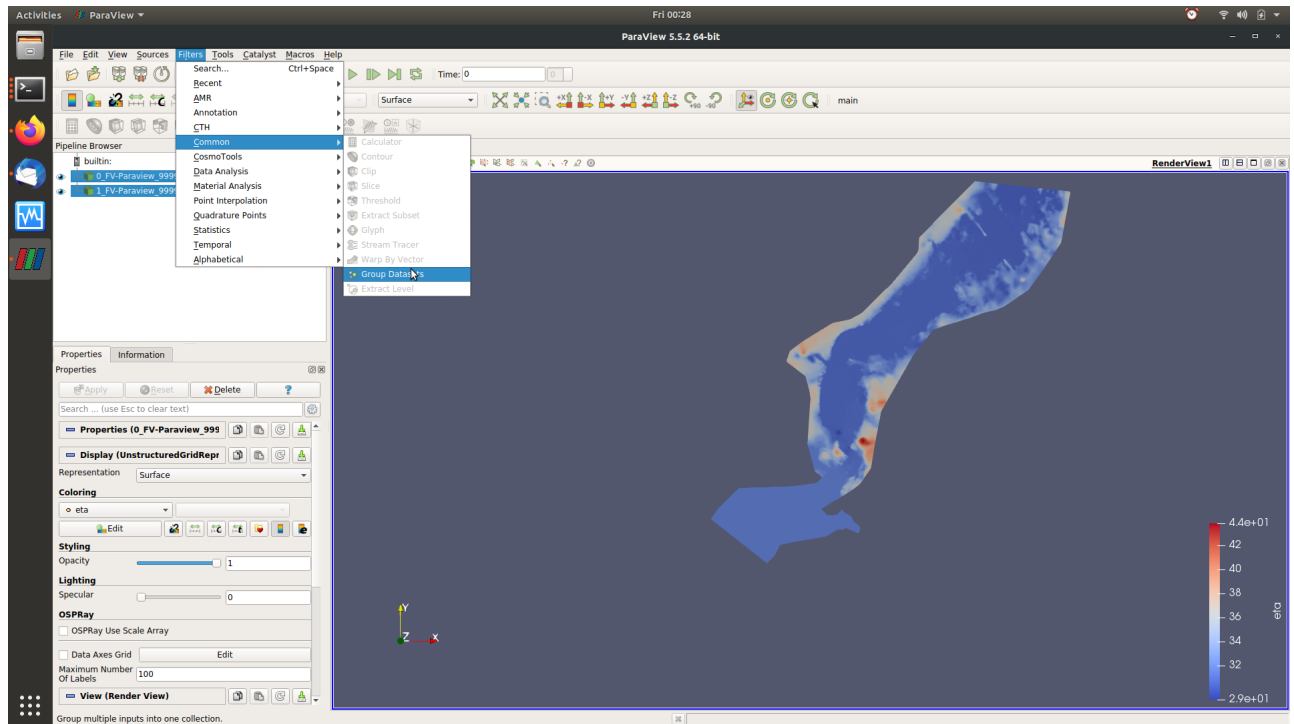


FIGURE 24 – Grouper les sous domaines

Cette opération va créer plusieurs fichiers dans la fenêtre de gauche, celui qui nous intéresse est le plus bas dans la liste, GroupDataSets1 à coté d'un cube. C'est le fichier qu'il faut sélectionner pour faire la suite du post-traitement. De cette façon le post-traitement sera effectué sur tout le domaine.

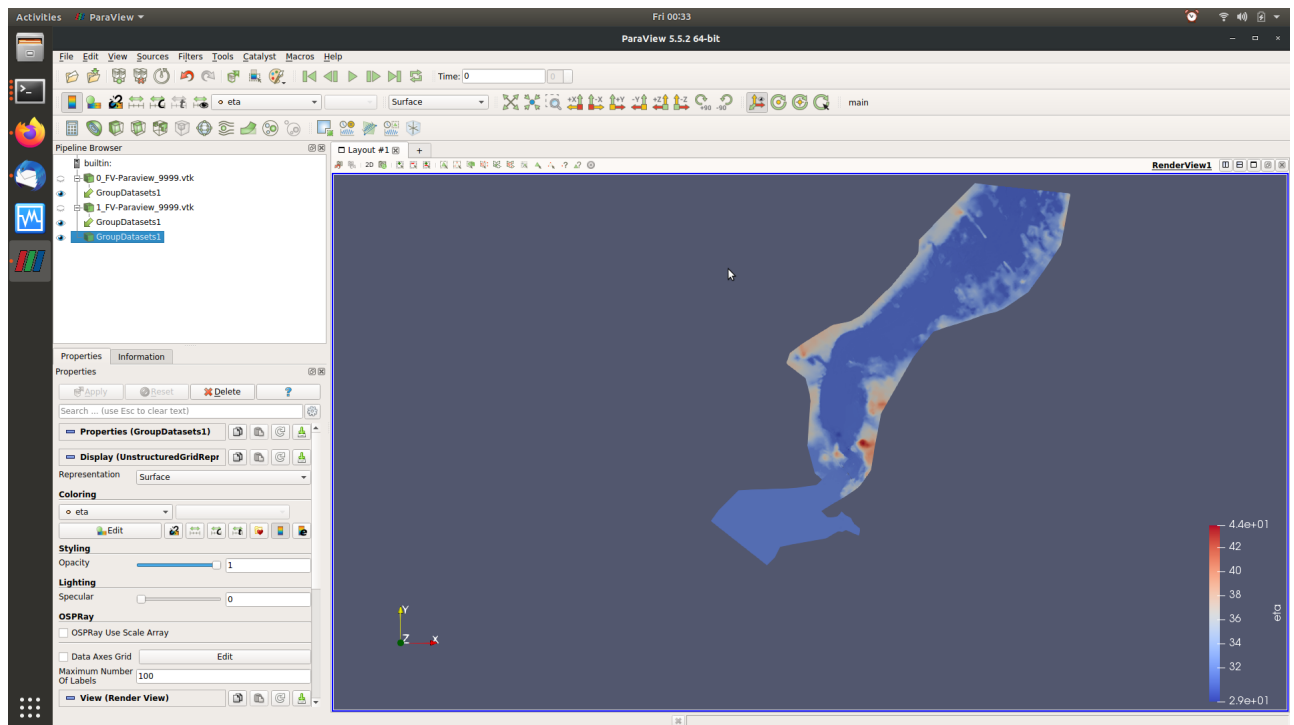


FIGURE 25 – Apply Group Dataset

Une fois les solutions chargées de cette façon le post-traitement peut commencer. On peut par exemple afficher un isovolume en allant dans Filters > Alphabetical et en cliquant sur isovolume puis en cliquant sur Apply. A noter que tous les filtres disponibles sont accessibles dans Filters > Alphabetical.

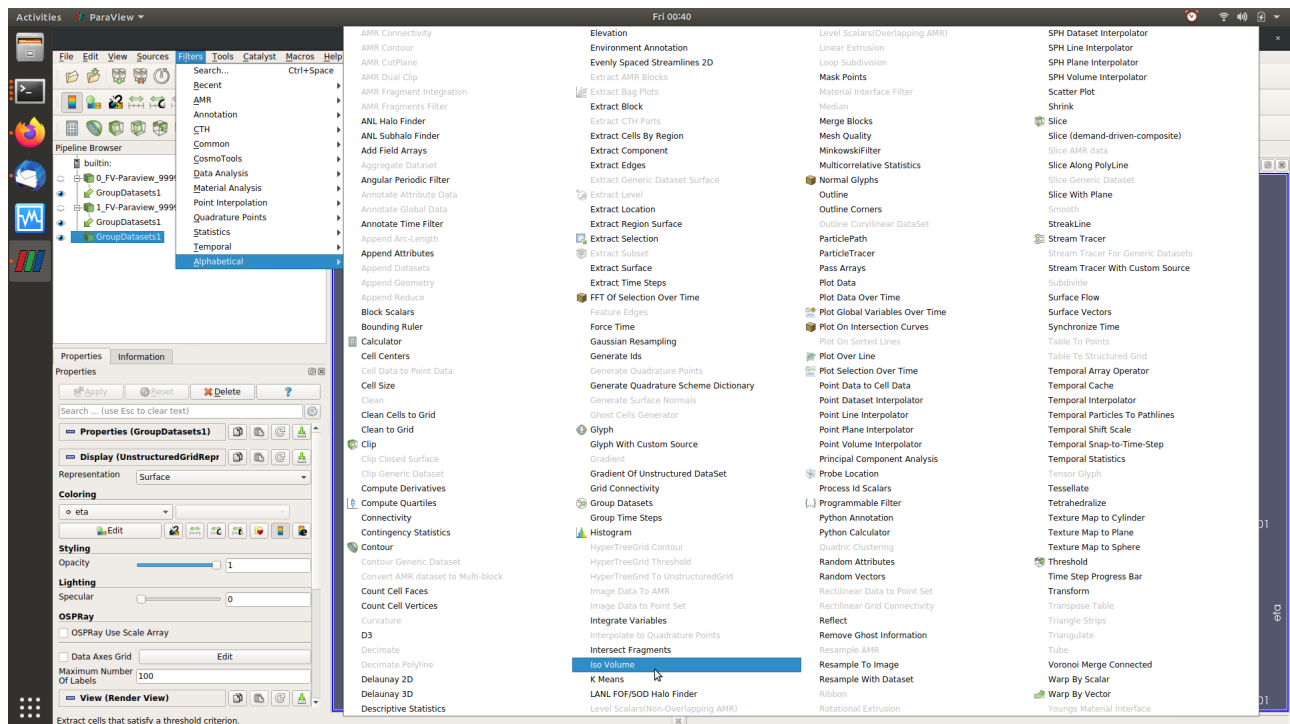


FIGURE 26 – Filters Alphabetical isovolume

On peut ensuite configurer la variable sur laquelle baser l'isovolume et les limites inférieures en supérieures dans la fenêtre en bas à gauche.

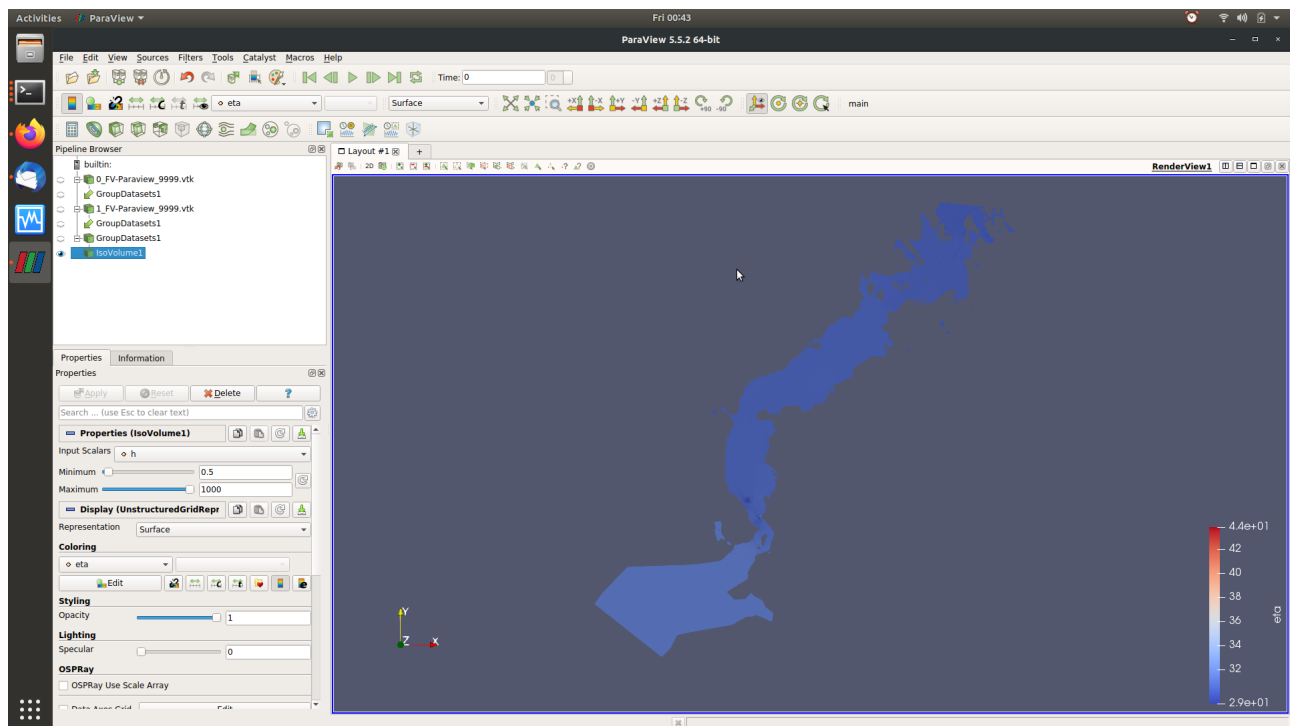


FIGURE 27 – Isovolum sur h entre 0.5 et 1000

On peut aussi afficher une ligne d'isovaleurs, en sélectionnant Filters > Alphabetical et en cliquant sur Contour puis en cliquant sur Apply. Comme pour l'isovolume on peut ensuite fixer les paramètres dans la fenêtre en bas à gauche. **Attention à bien re-sélectionner le GroupDataSet1 dans la fenêtre de gauche pour y appliquer les lignes d'isovaleurs.**

On a dans cet exemple caché l'isovolume en cliquant sur l'oeil à coté du fichier dans la fenêtre de gauche puis re-affiché le domaine complet de la même façon en cliquant sur l'oeil à coté de GroupDataSets1 dans la fenêtre de gauche. On a ensuite coloré le domaine en fonction de la norme de la vitesse, en sélectionnant velocity et magnitude en dessous de Coloring dans la fenêtre en bas à gauche lorsque GroupDataSet1 est sélectionné. On a ensuite ajouté les lignes d'iso

valeurs en selectionnant Filters > Alphabetical > Contour puis j'ai configuré deux isovaleurs sur la variable h à 0.5 et 0.05 voir la fenêtre en bas a gauche lorsque le fichier Contour est selectionné.

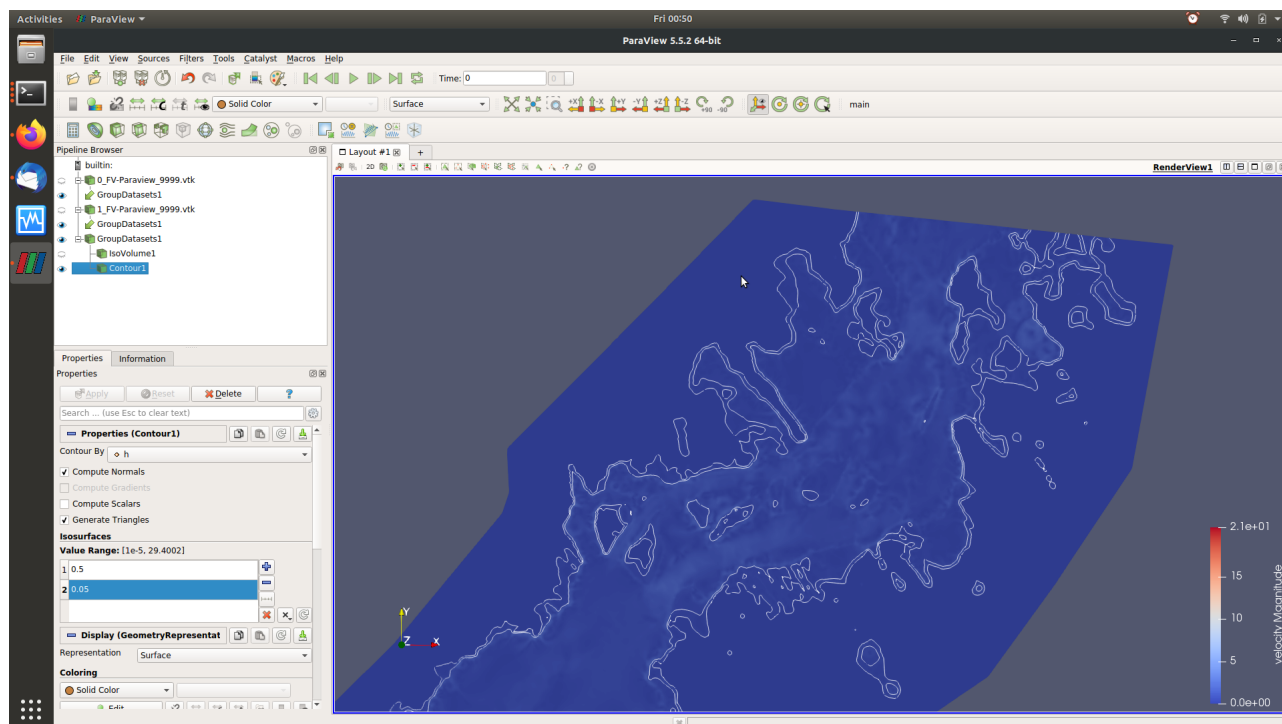


FIGURE 28 – Lignes d'isovaleurs  $h = 0.5$  et  $h = 0.05$  sur le domaine coloré par velocity magnitude