Documentation de la version multi-GPU de CUTEFLOW

Vincent Delmas

1er juillet 2020

Table des matières

1 Introduction à la version multi-GPU de CUTEFLOW

1.1 Git pour la gestion des versions

On utilise le logiciel Git [1] pour gérer les versions du code. Un tutoriel est présent sur le site officiel https://git-scm.com/docs/gittutorial. On présent simplement comment cloner le dossier du code et comment le tenir à jour des modifications qui y sont faites. Git permet évidement 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'ailleur intéressant que chaque personne qui modifie le code travaille dans une branche spécifique.

Le dossier Git d'origine du code est présent sur Cédar 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
```

Un dossier CUTEFLOW_CUDA_MPI sera créé contenant les fichiers important du code. On peut noter qu'on peut cloner ce projet sur d'autres grappes de calcul ou 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
```

Pour télécharger les modification faites au fichiers source des différents codes, on peut procéder de la manière suivante.

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

# Telecharge les changements
git pull
```

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

1.2 Structure des dossiers

On présent la structure des dossiers du code.

```
CUTEFLOW_CUDA_MPI
├── build
├── gpu_config
├── split_mesh
├── merge_solutions
├── merge_bluekenue
├── split_solution
└── simulations_mille_illes
    ├── 743968_2gpu
    └── multi_2gpu
        └── base_files
```

Le dossier **build** contient le coeur de la version multi-GPU du code CUTEFLOW. Voir section 3 pour l'utilisation.

Le dossier **gpu_config** contient des fichiers de configuration pour le lancement du code sur les grappes de calcul. Ces fichiers sont des exemples et sont destinés à être copié et modifié pour correspondre au besoin voulus. Ces fichiers doivent ensuite être soumis à l'ordonanceur à l'aide de la commande sbatch pour lancer une simulation voir section 4.

Le dossier **split_mesh** contient le code pour diviser un maillage en autant de sous domaines que voulus, c'est un code en C++, un fichier modules et un makefile sont présent de la même façon que dans le dossier build. Voir section 2 pour l'utilisation de ce code.

Le dossier **merge_solutions** contient un code qui permet de regrouper les fichiers de solution 2D et 3D générés dans le code pour le traitement par Monte Carlo. Voir section 5 pour le lancement de plusieurs simulations et section 6 pour la combinaison des fichiers résultats.

Le dossier **merge_bluekenue** contient un code pour regrouper les fichiers *.T3S générés à la fin des simulation. Voir section 7 pour son utilisation.

Le dossier **split_solutions** contient un code qui permet de spéparer un fichier de solution pour le restart en autant de sous domaines que voulus en fonction du maillage donné. Section de documentation à venir.

Le dossier **simulations_mille_iles** contient des exemples de simulations. Dans le dossier **743968_2gpu** un exemple simple d'une simulation sur un maillage de 743968 éléments. Dans le dossier **multi_2gpu** un exemple de 10 simulations pour un maillage de 481930 éléments.

2 Découpe du maillage en sous domaines

Le dossier **split_mesh** contient le nécessaire pour faire la découpe d'un maillage en sous domaines. Il est très fortement conseillé de créer un nouveau dossier dans lequel on découpe le maillage pour garder les fichiers organisés. On présente en exemple comment découper le fichier de maillage Mille_Iles_mesh_481930_elts.txt en 2 sous domaines. Il suffit simplement de faire :

```
#Utilisation : ./split_mesh fichier_de_maillage nombre_de_sous_domaines multi_entrees
    ↪ multi_sorties
./split_mesh Mille_Iles_mesh_481930_elts.txt 2 0 0
```

Ou 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 garder les fichiers organisés dans un nouveau sous dossier.

```
# En etant dans le dossier CUTEFLOW_CUDA_MPI

#Aller dans le dossier du code de decomposition du maillage
cd split_mesh

# Creation d'un dossier, le nom importe peu tant qu'on s'y retrouve
mkdir mille_iles_481930_2gpu

# Deplacement dans le dossier que l'on vient de creer
cd mille_iles_481930_2gpu

# Copie du fichier de maillage a decouper dans le dossier courant,
#remplacer **chemin** par le chemin d'access au fichier de maillage.
cp **chemin**/Mille_Iles_mesh_481930_elts.txt .

# Copie de l'executable de decomposition
cp ../split_mesh .

# Lancement du code avec comme argument le fichier de maillage a decouper
# et le nombre de sous domaine desires
./split_mesh Mille_Iles_mesh_481930_elts.txt 2 0 0
```

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 donneemai_miles_iles.txt 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 le code dans le dossier merge_solutions qui les utilise pour ré-assembler les solutions locales en solution globale.

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

3 Fichiers source et compilation du code

Les fichiers sources sont des fichiers *.cuf (CUDA Fortran). Dans ce dossier un fichier **modules** est présent. Il indique la commande à exécuter pour loader les modules nécessaire à la compilation du code. Un **makefile** est présent pour compiler le code avec la commande make et nettoyer le dossier de compilation avec la commande make clean. Une fois la compilation finie l'exécutable **runfile** est créé. A notre que make clean ne supprime pas l'exécutable créé lors de la compilation.

Exemples de lancement sur les cas tests et description du fichier d'input *donnees.f* à venir

4 Lancement d'une simulation

Le lancement d'une simulation se fait quasiment de la même façon que pour la version sur un seul GPU. Encore un fois l'important est d'organiser les simulation pour ne pas se retrouver avec des fichiers résultats dans tous les sens. Il est conseillé de créer un nouveau dossier pour chaque simulation, on présente la procédure en continuant sur l'exemple du fichier de maillage Mille_Iles_mesh_481930_elts.txt qui a été décomposé en section 2

```
# En etant dans le dossier CUTEFLOW_CUDA_MPI

# Creation et deplacement dans un dossier pour la simulation, le nom importe peu
mkdir 481930_2gpu
cd 481930_2gpu

# Copie des fichiers de maillages dans le dossier courant
```

```
cp ../split_mesh/mille_iles_481930_2gpu/^*[0-9]_Mille_Iles_mesh_48930_elts.txt .

# (Optionnel) Copie des tables de correspondances dans le dossier courant,
cp ../split_mesh/mille_iles_48930_2gpu/^*[0-9]_lien_nodes.txt .
cp ../split_mesh/mille_iles_48930_2gpu/^*[0-9]_lien_elems.txt .

# (Optionnel) Copie de l'executable merge_bluekenue
# Si fichier introuvable il faut aller compiler le code dans merge_bluekenue avec make
cp ../merge_bluekenue/merge_bluekenue .

# Copie des fichiers de donnees dans le dossier courant, remplacer **chemin**
cp ../build/donnees.f .

# Copie du runfile de CUTEFLOW dans le dossier courant
cp ../build/runfile .

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

# Copie du makefile du code dans le dossier courant,
# Inutile pour la compilation, utile pour nettoyer le dossier avec la commande "make
    ↪ clean"
cp ../build/makefile .
```

A ce stade le résultat de la commande ls dans le dossier 481930_2gpu devrait être le suivant :

```
0_Mille_Iles_mesh_481930_elts.txt
0_lien_elems.txt                    # Optionnel pour combinaison Bluekenue
0_lien_nodes.txt                    # Optionnel pour combinaison Bluekenue
1_Mille_Iles_mesh_481930_elts.txt
1_lien_elems.txt                    # Optionnel pour combinaison Bluekenue
1_lien_nodes.txt                    # Optionnel pour combinaison Bluekenue
2_gpu.sh
donnees.f                           # Nouveau fichier de donnees
makefile
merge_bluekenue                     # Optionnel pour combinaison Bluekenue
runfile
```

Il faut ensuite de vérifier les données de simulation dans le fichiers *donnees.f*. Il faut s'assurer que le nom du fichier de maillage est bien Mille_Iles_mesh_481930_elts.txt, le code se chargera tout seul de lire les fichiers de maillages préfixés par le numéro de sous domaine. Finalement il faut vérifier le temps maximal d'exécution du code dans le fichier 2_gpu.sh. On peut ensuite soumettre le code à l'ordonnanceur avec la commande suivante :

```
# En etant 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
```

Une fois le code lancé on peut suivre l'avancement en monitorant les fichiers outfile.[0-9] qui servent de stdout pour chaque process MPI, par exemple pour le stdout du process 0 :

```
# En etant dans le dossier 481930_2gpu, une fois le code lance
tail -f outfile.0
```

Cette commande permet de regarder la fin du fichier outfile.0 de manière dynamique, Ctrl-C pour en sortir.

Une fois la simulation terminée, selon les paramètres mis dans *donnees.f*, plusieurs fichiers résultats seront générés en particulier les fichiers *.vtk si le parametre video à été mis a 1. Un fichier résultat est créé par sous domaine, le domaine 0 correspond aux fichiers 0_FV-Paraview_*.vtk.

On laisse aussi la sortie de fichiers *.T3S pour Bluekenue uniquement pour la solution finale. On peut combiner ces fichiers avec le code présent dans le dossier merge_bluekenue voir section 7.

5 Lancement de plusieurs simulations

Dans le version précédente du code on pouvait utiliser le paramètre multi_simul pour spécifier au code qu'il fallait faire plusieurs simulations. Les paramètres de chaque simulation étaient donnés dans un fichier souvent 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. On peut faire en sorte de les lancer toutes en même temps en utilisant des job arrays sur les clusters de Compute Canada.

L'objectif de cette méthode est de lancer le script *multi* qui va créer un dossier par simulation en y copiant tous les fichiers nécessaires à l'exécution du code et en modifiant le fichier des données pour correspondre au cas voulu. **Ces modifications sont faites par le script gen_donnees qui génère les fichiers de donnée, pour modifier les paramètres de la simulation il faut modifier le fichier gen_donnees et pas les fichiers de donnée générés**.

5.1 Génération des fichiers de donnée

On a besoin d'un moyen de générer les fichiers de données automatiquement a 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'as pas normalement à l'utiliser tout seul mais on décrit quand même sont 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
```

Les différents fichiers de données sont alors générés. Attention, dans la configuration actuelle la valeur de haval sera aussi utilisé comme hsortie.

5.2 Préparation du dossier base_files

Avant de lancer le script *multi* 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, le runfile, on peut en plus ajouter les fichiers de liens et l'executable *merge_solutions* si on veut pouvoir recombiner les solutions dans la suite. La commande ls exécutée dans le dossier base_files devrait donner,

```
#Dans le dossier 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
2_gpu.sh
makefile
merge_solutions
runfile
```

Tous ces fichiers seront copiés dans chacun des dossiers créés pour chaque cas. On pourrait par exemple ajouter les fichiers de restart dans le dossier base_files pour commencer le code d'une solution déjà initialisés. Il faudrait alors changer le paramètre sol_init dans le script *gen_ donnees*.

5.3 Lancement du script *multi*

Le script *multi* 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*. Pour l'utiliser il faut avoir le dossier base_files ainsi qu'un fichier d'input comme INPUT_MONTE_CARLO.dat dans le répertoire courant, ensuite

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

Les différents dossiers multi_*[0-9] seront crées 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* puis de relancer le script *multi* comme indiqué précédemment. De cette façon la modification s'appliquera dans tous les dossiers.

5.4 Lancement du 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. On prends en exemple le fichier array.sh suivant,

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --gres=gpu:2
#SBATCH --mem=8000M
#SBATCH --time=0-00:40
#SBATCH --account=def-soulaima
#SBATCH --array=1-10

module load pgi/19.4 cuda/10.0.130 openmpi/3.1.2

cd cas_${SLURM_ARRAY_TASK_ID}
mpirun --mca pml ob1 --mca btl openib -n 2 sh -c './runfile > outfile.
    ↪ $OMPI_COMM_WORLD_RANK'
```

Ce fichier décrit un job qui utilise un noeud de calcul avec 2 gpu pour une durée de 40mn. Un paramètre important est le paramètre array qui décrit le nombre de jobs à lancer. Dans cet exemple on lance 10 jobs qui seront numéroté de 1 à 10. 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'ordonanceur il suffit ensuite de faire

```
sbatch array.sh
```

5.5 Récupération des fichiers résultats

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 6 pour regrouper les fichiers solution de chaque gpu en un unique fichier résultat pour chaque simulation. Il s'utilise de la façon suivante,

```
#Pour des simulations utilisant 2GPU
./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 crée, 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.

6 Combinaison des fichiers de solution restart, sol2D et sol3D

Lorsque qu'une simulation se termine elle génère plusieurs fichiers qui peuvent pour simplifier le post-traitement être combiné en un unique fichier. Le traitement de la re-combinaison des fichier *.T3S de Bluekenue est décrite à part en section 7. Dans le dossier **merge_solutions** on trouve le code qui permet de recombiner les fichiers de restart, sol2D et sol3D du code. Par exemple pour recombiner les fichiers 0_SOL_FV_MULTISIM_3D.txt et 1_SOL_FV_MULTISIM_3D.txt on fait,

```
# Utilisation : ./merge_solutions {restart|sol2d|sol3d} nom_de_base_fichiers nGPU
./merge_solutions sol3d SOL_FV_MULTISIM_3D.txt 2
```

ATTENTION IL FAUT QUE LES FICHIERS *_lien_nodes.txt ET *_lien_elems.txt SOIENT PRÉSENT DANS LE DOSSIER COURANT

7 Combinaison des fichiers de solution pour Bluekenue

Dans le dossier merge_bluekenue on peut trouver le code qui permet de combiner des fichiers *.T3S générés par CUTEFLOW en un unique fichier sur le domaine global. Au vue 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.

Pour fonctionner correctement le code à besoin des tables de correspondances entre la numérotation locale et la numérotation globale pour chaque sous domaine. **Ces fichiers sont crées lors de la découpe du fichiers de maillage et il faudra les copier dans le dossier courant pour que merge_bluekenue s'exécute correctement**. Voir section 2 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_FV-H-Bluekenue_t_245.759s.T3S et 1_FV-H-Bluekenue_t_245.759s.T3S :

```
# Utilisation : ./merge_bluekenue nom_de_base_fichiers nGPU
./merge_bluekenue FV-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 PRÉSENT 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.

8 Traitement dans Paraview des fichiers *.vtk

Le logiciel Paraview [2] 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 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.

On présente ici une utilisation basique pour visualiser les fichiers *.vtk produits par le code CUTEFLOW en prenant la suite de l'exemple en section 4. On considère que les fichiers 0_FV-Paraview_9999.vtk et 1_FV-Paraview_9999.vtk ont déjà été téléchargés localement.

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.

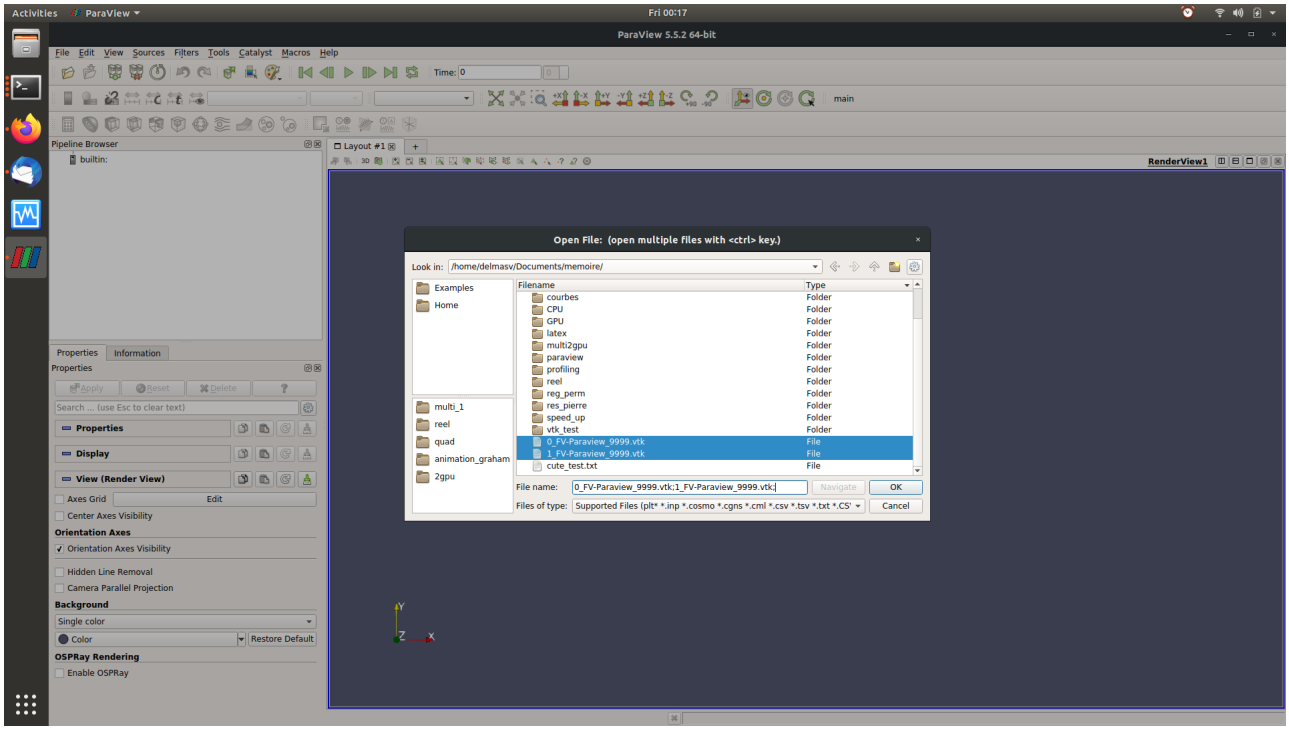

FIGURE 1 – Ouverture des fichiers

Une fois les fichiers présent dans la partie gauche de Paraview, il faut les sélectionner et cliquer sur Apply, le bouton vert juste en dessous de l'arbre des fichiers. 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.

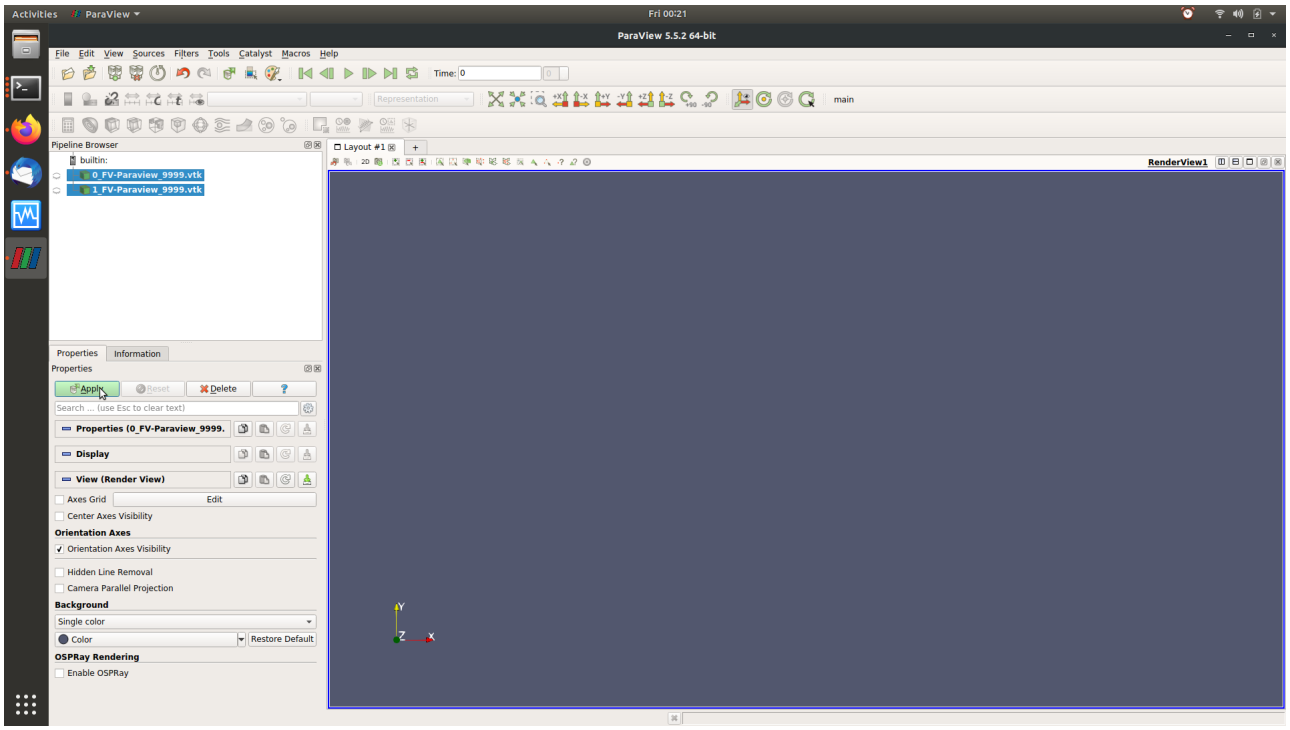

FIGURE 2 – Sélectionner les fichiers et cliquer sur Apply

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.

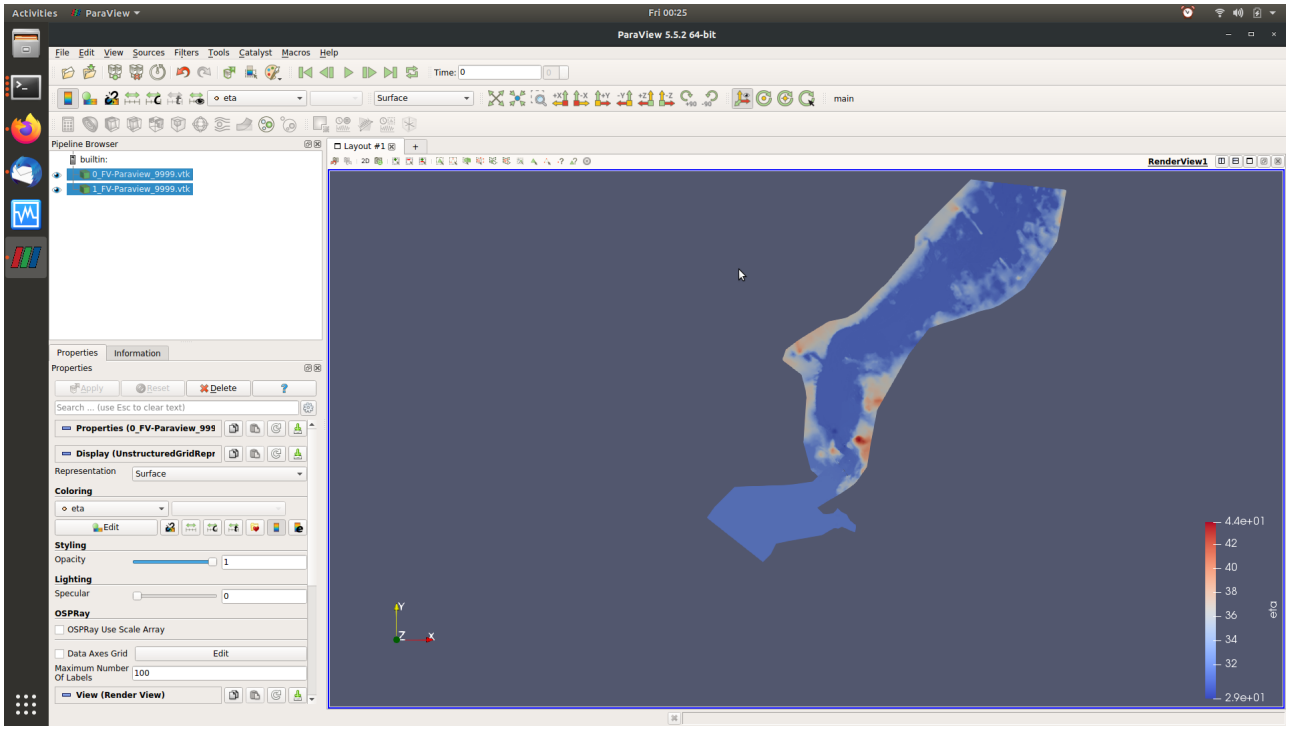

FIGURE 3 – Affichage du domaine

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).

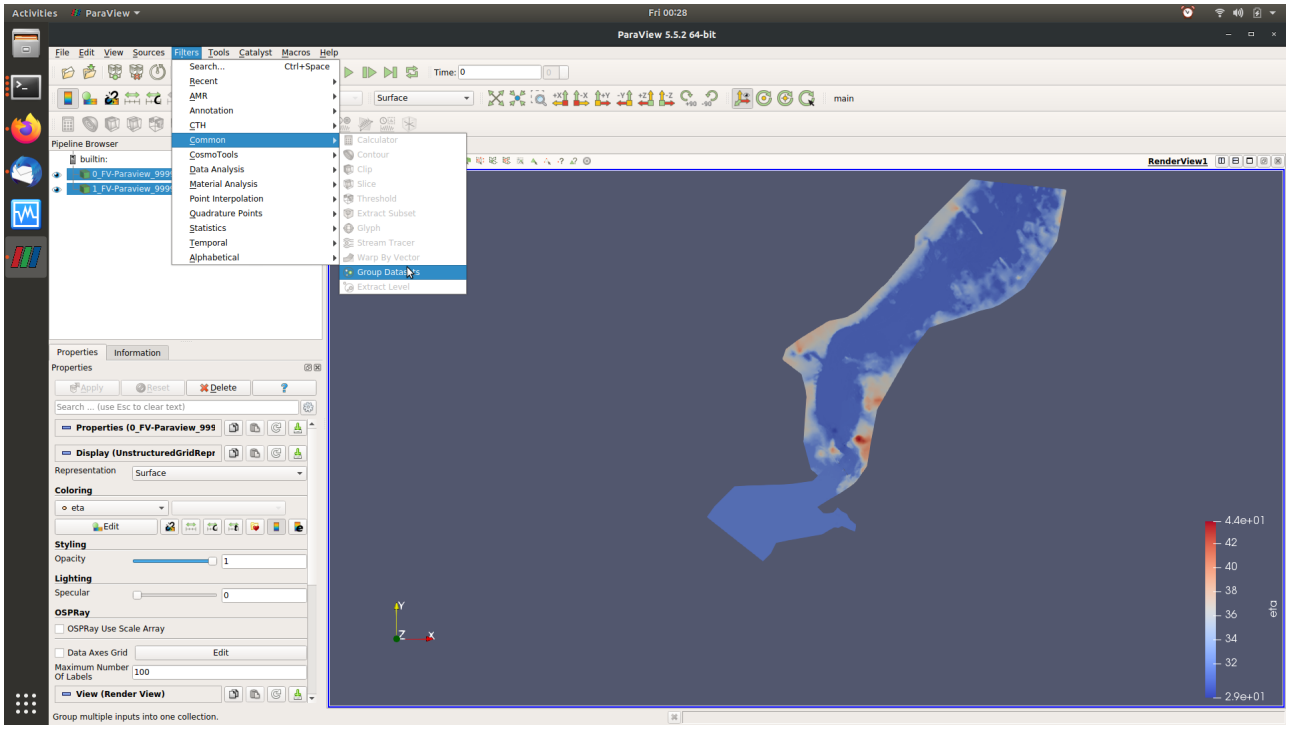

FIGURE 4 – Grouper les sous domaines

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.

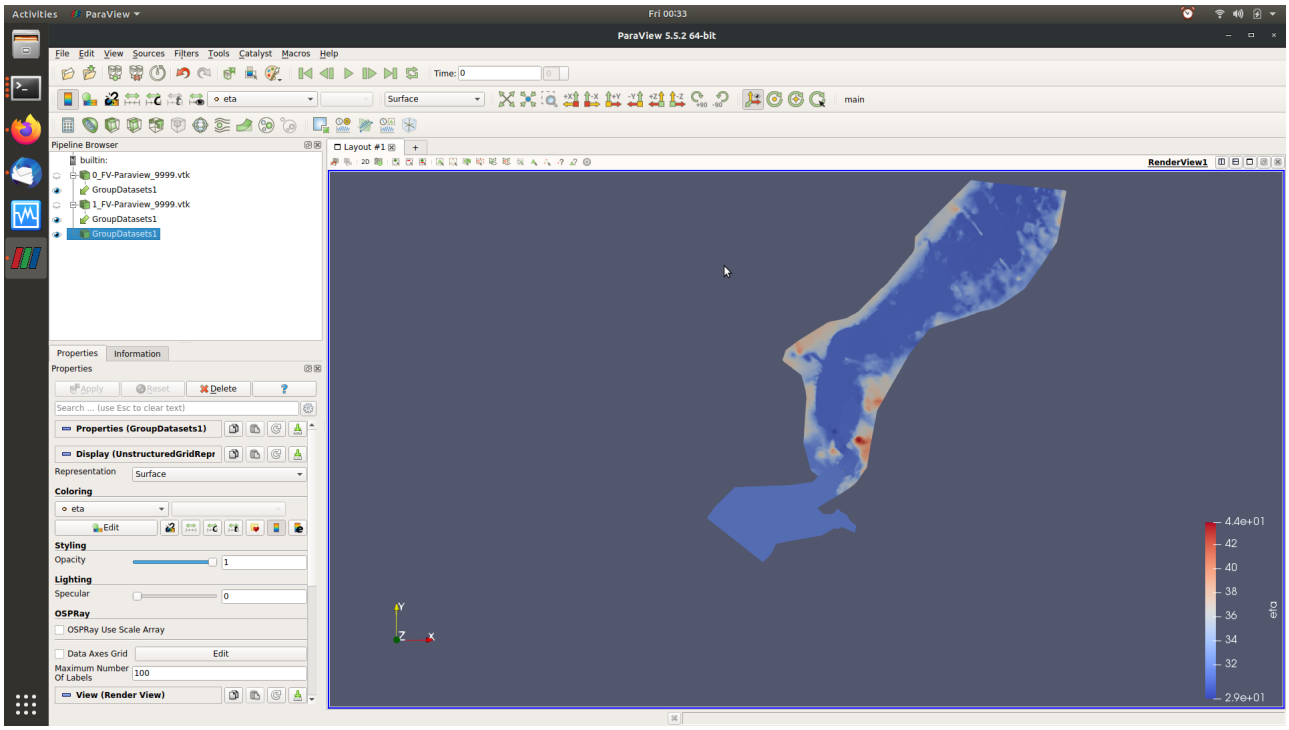

FIGURE 5 – Apply Group Dataset

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.

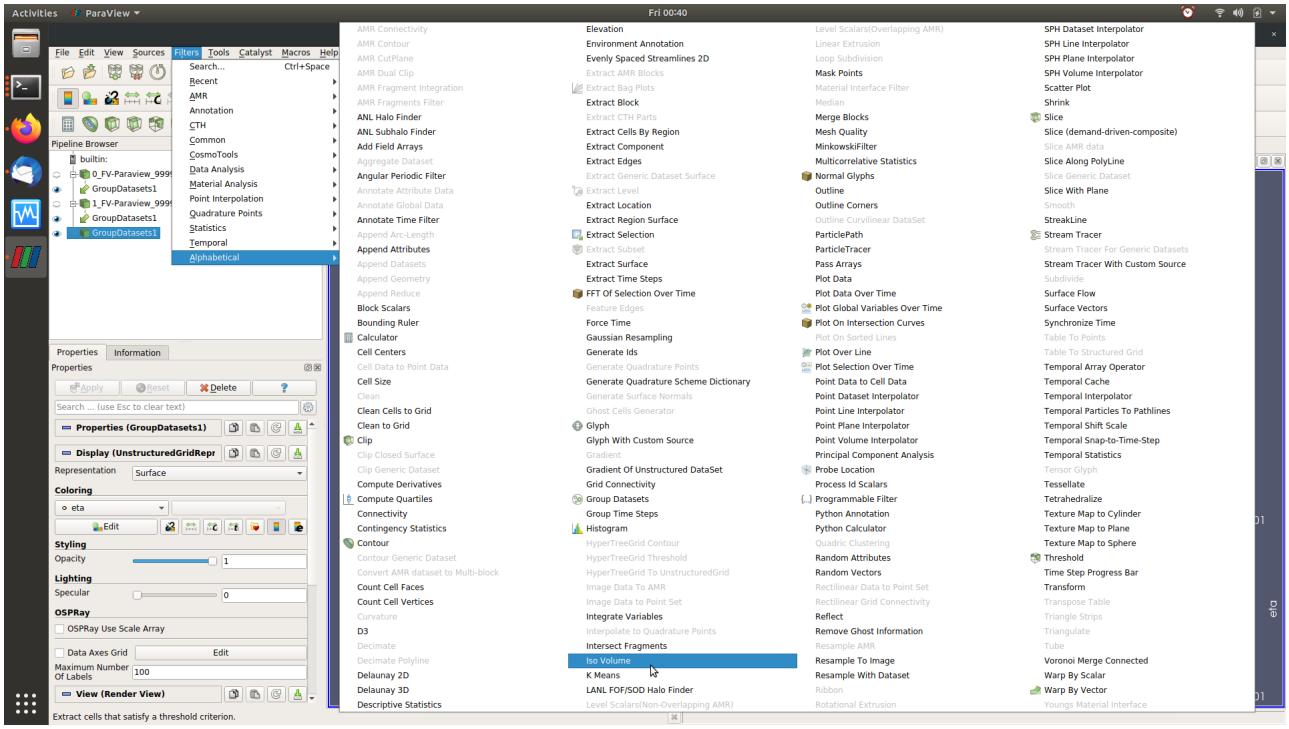

FIGURE 6 – Filters Alphabetical isovolume

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

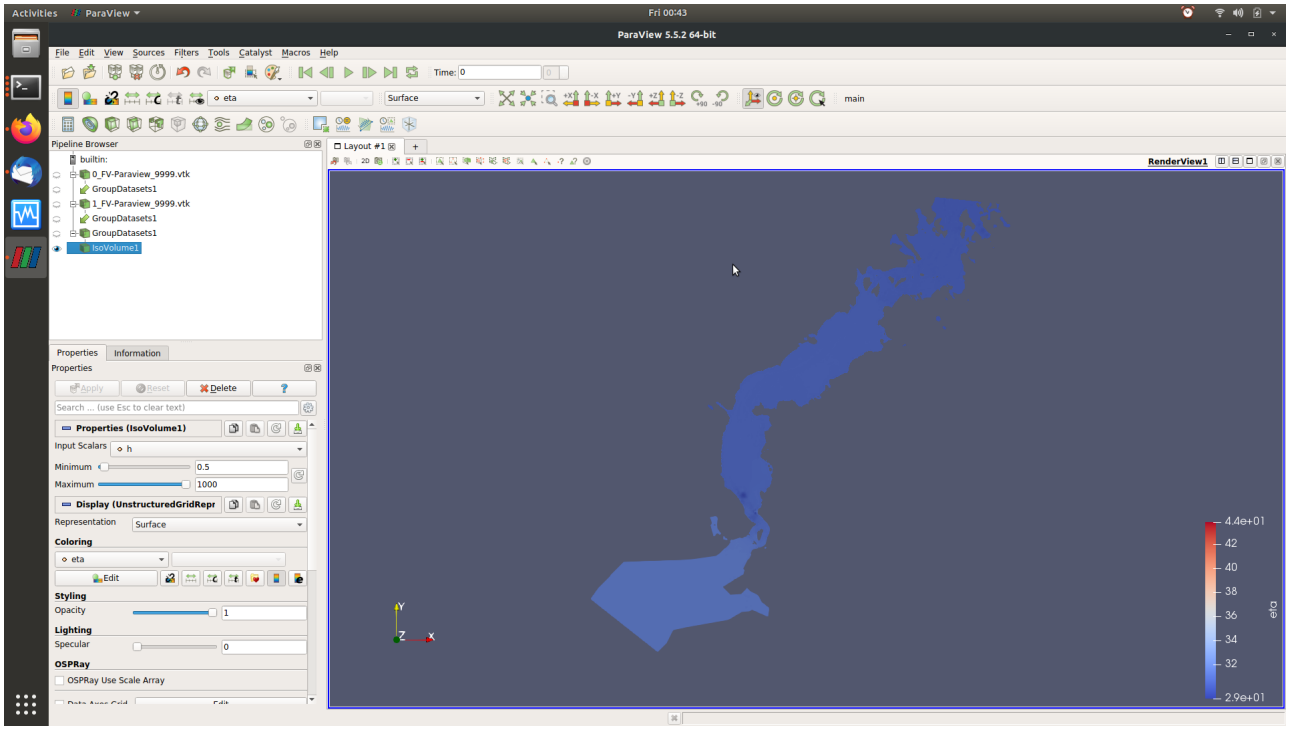

FIGURE 7 – Isovolume sur h entre 0.5 et 1000

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 a gauche. **Attention à bien re-selectionner 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 a 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.

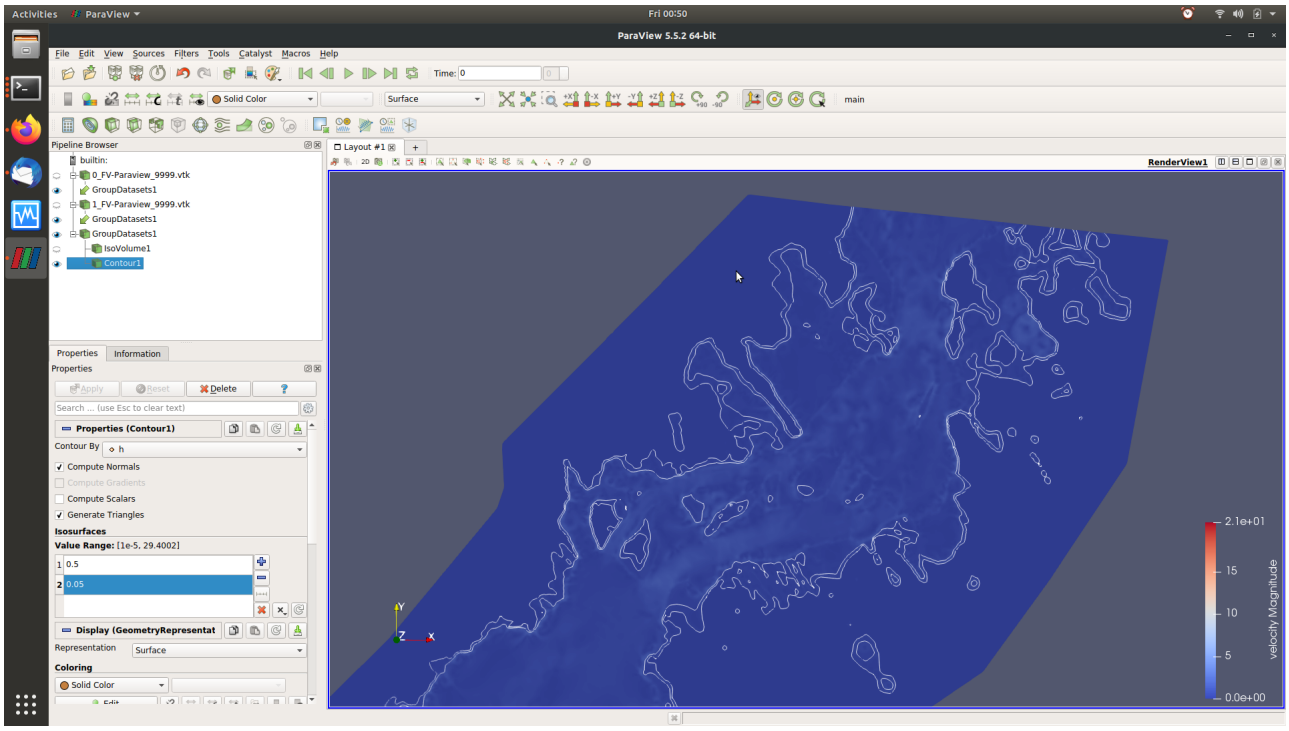

FIGURE 8 – Lignes d'isovaleurs $h = 0.5$ et $h = 0.05$ sur le domaine coloré par velocity magnitude

Références

[1] URL : https://git-scm.com/. (23/02/2020).

[2] Utkarsh AYACHIT. *The ParaView Guide : A Parallel Visualization Application*. Clifton Park, NY, USA : Kitware, Inc., 2015. ISBN : 1930934300.