



Emmanuel Courcelle  
Hervé Neau  
Annaïg Pedrono

## Atelier Visualisation in Situ : TP ParaView/Catalyst

Ce TP s'appuie sur l'exemple pjacobi fourni par J. M. Favre dans les exemples VisIt/libsim.

**Objectif** : connecter la bibliothèque Catalyst de ParaView avec un code de calcul

Prérequis :

- compilation de deux versions de ParaView (avec et sans interface graphique)
- make
- mpi
- un compilateur C ou Fortran ou Python selon le langage choisi

Le TP est prévu pour du bash. Modifier les chemins vers l'installation de ParaView dans les fichiers paraview\_GUI.sh et paraview\_batch.sh

### Partie I : Utilisation de ParaView/Catalyst in Situ pour définir la mise en page

a – Modifier le Makefile pour :

- Définir le chemin vers le répertoire d'installation de ParaView
- Définir le compilateur

```
cd TP_visu2017_Catalyst
source paraview_batch.sh
cd F90
```

Recompiler avec :

```
make clean; make
```

Tester :

```
./pjacobi
```

Le programme pjacobi est fonctionnel, mais *il est pour l'instant complètement indépendant de ParaView.*

Modifier le Makefile pour compiler avec les bibliothèques **ParaView/Catalyst**.

```
BUILD_CATALYST=1
make clean; make
```

Démarrer paraview :

```
source ../paraview_GUI.sh  
paraview &
```

Au premier lancement de ParaView, dans le menu :  
**tools/Manage Plugins.../CatalystScriptGeneratorPlugin**  
cliquez sur **Auto Load** puis cliquer **Load Selected**.

Puis, préparer ParaView à la connexion :  
**menu Catalyst -> Connect... -> OK -> OK**

**Attention** : Si vous travaillez sur un serveur, deux utilisateurs connectés sur la même machine ne peuvent utiliser le même numéro de port !

On peut vérifier que Paraview écoute effectivement avec la commande :  
`netstat -plet`

**b** – Lancer l'exécutable pour vérifier que tout se passe bien :

```
cp script/coproc.py .
```

**Si nécessaire**, modifier la dernière ligne du script pour ajuster la connexion à Paraview.

```
./pjacobi_catalyst coproc.py
```

**Dans ParaView,**  
# menu **Catalyst => Pause Simulation**  
# création de Extract: cliquer à gauche de « input » sur le petit triangle et le cercle  
=> **Extract : input** => rendre visible l'extract en cliquant sur l'oeil  
=> faire la mise en page adaptée pour sortir des images  
# menu **Coprocessing Export State -> Next**  
# **show all sources**  
# cliquer **Extract: input -> add -> Next -> Next**  
# cliquer **Output rendering components i.e. views**  
# adapter si besoin **Write Frequency = 1**  
# **Finish**  
# sauver le script python (ex **batch\_pjacobi.py**)  
  
# menu **Catalyst => Continue**  
# menu **Catalyst => Pause** (mise à jour du champ de temperature)  
# menu **Catalyst => Continue**  
# menu **file => exit**  
# menu **file => disconnect**

Tester ce script créé dans paraview avec la commande suivante :

```
./pjacobi_catalyst batch_pjacobi.py  
=> sortie des images à la fréquence définie dans le code (pjacobi.f90) et/ou dans le script python (ex  
batch_pjacobi.py)  
=> image_0.png image_5.png image_10.png ...
```

```
eog image_0.png
```

Modifier le code et/ou le script python pour changer la fréquence de sortie des images.

Trier les images et en faire un film avec le script fourni : tri\_png\_video.sh

```
# test sur 4 cores
```

```
mpirun -np 4 ./pjacobi_catalyst batch_pjacobi.py
```

## **Partie II : Utilisation d'un fichier pvd pour définir la mise en page**

La visualisation in situ sur ce code où aucun cœur ne détient les données de tout le domaine ne fonctionne pas en parallèle. La solution conseillée est d'utiliser le fichier de résultat pvd d'un premier calcul pour faire la mise en page.

```
mpirun -np 4 ./pjacobi
```

Ouvrir le fichier last\_step.pvd dans Paraview et refaire la même suite de commandes qu'au paragraphe I-b pour créer un script bach nommé *batch\_jacobi\_pvd.py* par exemple. Cliquer sur le bouton Camera Reset avant d'exporter le script python.

Ce bouton corrige des erreurs de position de caméra par défaut.



```
mpirun -np 4 ./pjacobi_catalyst batch_jacobi_pvd.py
```

Tester :

- Interpréter le message d'erreur.
- Commenter les commandes inconnues dans le script

Observer les images créées, trier les et faire des vidéos avec le script : tri\_png\_video.sh

Editer le script python et tester les options et commandes disponibles.

Comparer ce script python batch\_jacobi\_pvd.py avec celui précédent batch\_pjacobi.py.

## **Partie III : Modification du code source pour ajouter de nouvelles fonctionnalités**

a – Modifier les sources pour faire afficher la variable scalaire **par\_rank**

b - Modifier les sources pour faire afficher la variable vectorielle **gradTemp**

d - Créer de nouveaux scripts batch pour créer des images avec les vecteurs et le rang des processus en connectant paraview ou à partir du fichier pvd.

c – Trier les images par ordre chronologique et faire une vidéo

## Annexe à l'intention des personnes qui travaillent sur Calmip

Paraview 5.4.1 est installé sur Eos. Pas la peine d'utiliser les scripts d'initialisation de paraview, à la place utilisez les modules prédéfinis :

**module load paraview/5.4.1**

ou

**module load paraview/pvbatch-5.4.1**

Pour utiliser l'interface graphique de Paraview, utilisez les nœuds de visualisation graphique d'Eos. Pour un meilleur confort d'utilisation, nous vous recommandons d'utiliser une grande fenêtre d'affichage :

**runVisuSession.sh -g 1600x1000**

Bien sûr vous devrez avoir installé sur votre poste de travail le logiciel virtual VNC.

Lorsque la fenêtre graphique est lancée, vous pouvez appeler Paraview à partir du menu principal :

*Bouton de droite*

*Vizualisation applications / VisIt*

Vous avez également à votre disposition un gestionnaire de fichiers (pratique pour visualiser les fichiers jpg générés par le programme de calcul lors de la partie III).

Toutes les explications sur l'affichage graphique à distance se trouvent ici :

<https://www.calmip.univ-toulouse.fr/spip.php?article463>

Pour lancer les calculs il est **indispensable** d'utiliser **les nœuds de calcul**. Ceux-ci n'ont pas besoin de graphique grâce à la bibliothèque Mesa (module pvbatch-5.4.1). Un script de lancement utilisant slurm est fourni, donc pour lancer les calculs il suffit de taper :

**sbatch slurm.sh**

Vous pouvez lire en temps réel ce qui est envoyé sur la sortie standard avec la commande :

**tail -f slurm.<job-id>.out**

Toutes les explications sur slurm et la configuration des queues sur eos se trouvent ici :

<https://www.calmip.univ-toulouse.fr/spip.php?rubrique96>