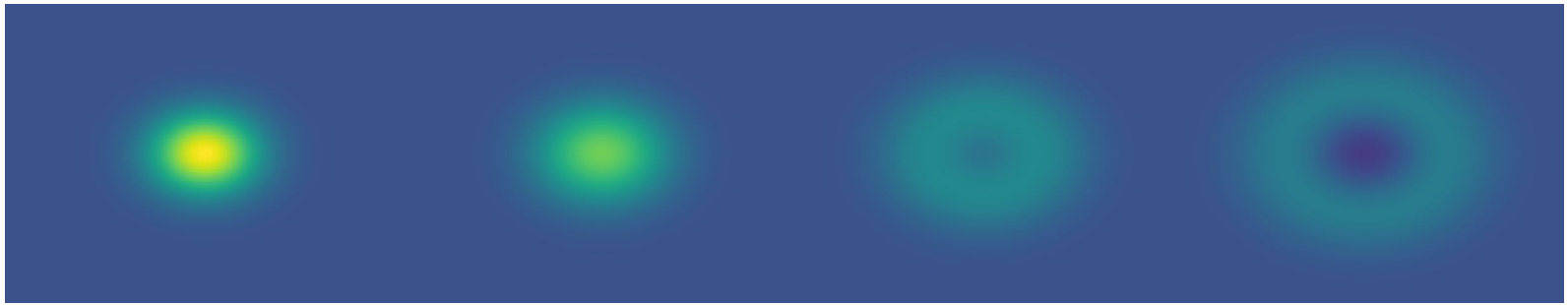
Projet HPC:
le modèle shallow water

Clément Chouteau
Romain Mekarni

Sommaire

Introduction du problème

Exemple de simulation de dimensions 256*256,



La précision de la simulation nécessite un pas de temps faible, mais elle n'évolue pas beaucoup, cela suggère l'enregistrement des grilles avec un pas de temps (paramètre export-step).

Analyse du code fourni (1)

- 6 tableaux de données: hFil, uFil, vFil, hPhy, uPhy, vPhy
- double buffering avec t modulo 2

On doit calculer hFil, uFil, vFil, hPhy, uPhy, vPhy, pour t à partir de ces tableaux à t-1

Remarquons que:

1. Chaque (H/U/V)Fil(t) dépend du calcul du (H/U/V)Phy(t) correspondant.
2. Les calculs de hPhy(t), uPhy(t), vPhy(t), sont indépendants entre eux.

Analyse du code fourni (2)

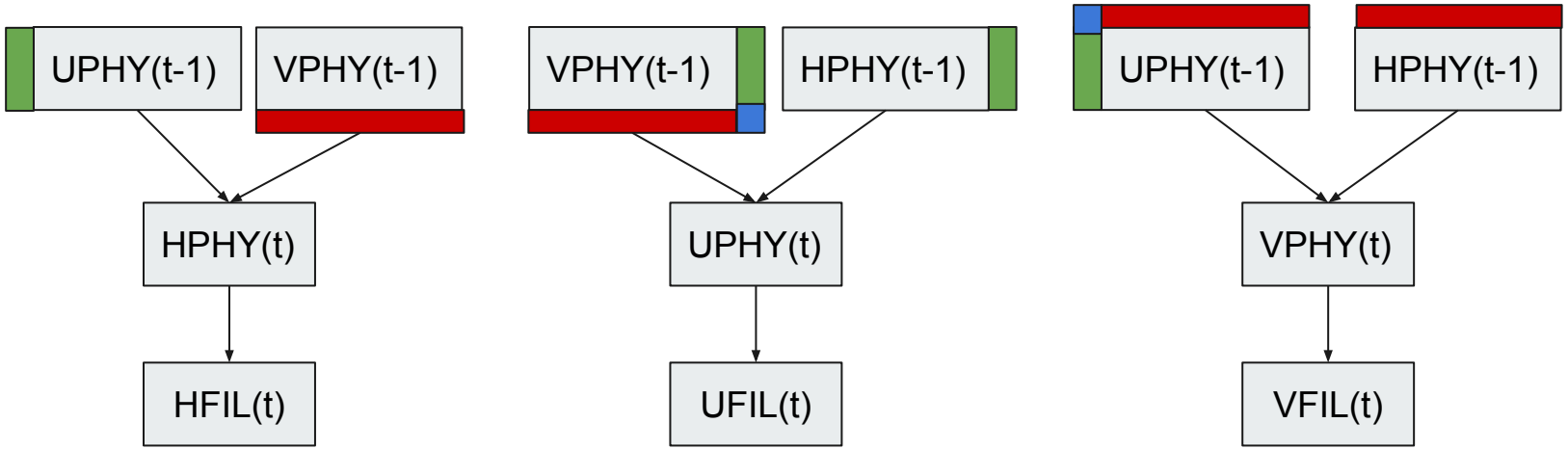
Initialement les matrices xPhy et xFil, sont stockées dans un tableau colonne par colonne puis ligne par ligne, et l'accès est ligne puis colonne, on a inversé le stockage pour un gain de performance important.

Gains en performances d'un facteur 2.29, (pour -x 2048 -y 2048 -t 20), sur le code séquentiel.

+ Consommation mémoire importante => 8096*8096 impossible sur 8Go de mémoire (4 threads)
+ En allouant seulement la partie nécessaire et les bords cela devient possible.
+ Permet de calculer des simulations dont les matrices sont trop grandes pour pouvoir être stockées en mémoire sur 1 seul ordinateur.

Analyse des dépendances (1)

Le calcul de chaque bloc dépend de (on a simplifié certaines dépendances sans débordements)



Analyse des dépendances (2)

Le calcul de chaque bloc (X)PHY demande de récupérer les bords de (Y)PHY, (Z)PHY

On a la circulation des (lignes, colonnes, coins) suivante:

1. UPHY ⇓, VPHY ⇑, HPHY ⇑
2. UPHY ⇒, VPHY ⇐, HPHY ⇐
3. UPHY ↘, VPHY ↖

Le version par bandes ne nécessite que la circulation des lignes.

Inutile de faire circuler les (X)FIL, car on n'a jamais besoin de points de (X)FIL en dehors de son bloc.

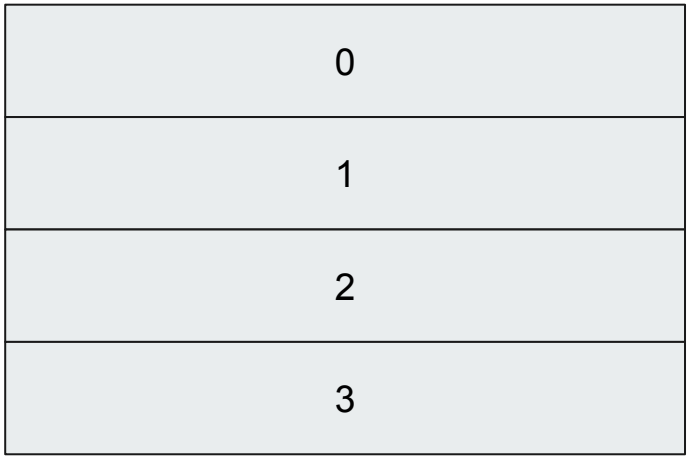
Découpage du travail (bandes) (1)

Coûts des échanges:

- données: X*(Y/p) + 2*X
- communications: 6
- transferts: 6*X

Inconvénients:

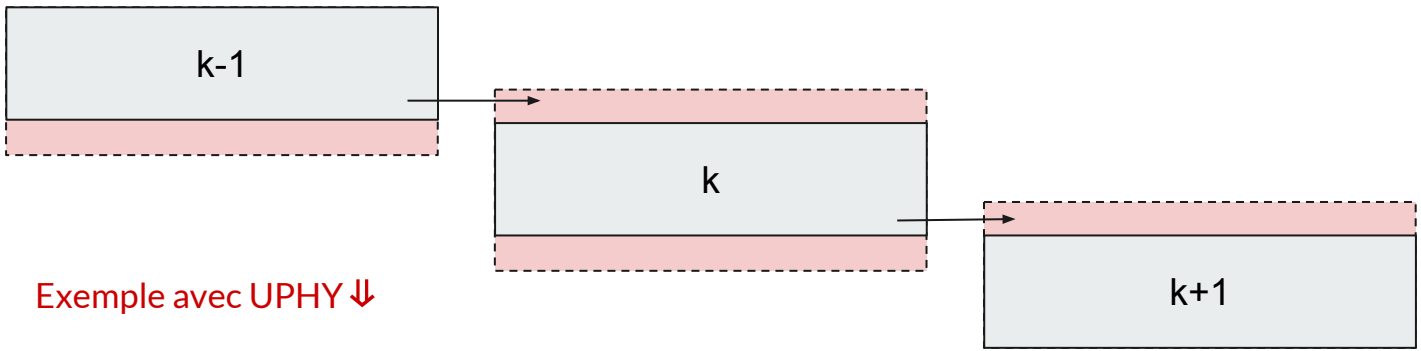
- taille des communications constante selon le nombre de processeurs, donc loi d'Amdahl.



Découpage du travail (bandes) (2)

Chaque processeur de calcul mémorise sa bande plus les lignes du haut et du bas.

Il calcul sa bande intérieure (t) à partir de sa bande (t-1) et des lignes et transmet une ligne au voisin ((k-1) ou bien (k+1) selon la donnée considérée), et récupère l'autre ligne avec l'autre voisin.



Exemple avec UPHY ⇓

Découpage du travail (blocs) (1)

Coûts des échanges:

- données: (X/q)*(Y/q) + 2*(X/q) + 2*(Y/q)
- communications: 16
- transferts: 6*(X/q) + 6*(Y/q) + 4

(0,0)	(0,1)
(1,0)	(1,1)

Inconvénients:

- un grand nombre de communications
- nécessite une recopie pour les colonnes
- on a quand même une loi d'Amdahl avec les temps de latence

Découpage du travail (blocs) (2)

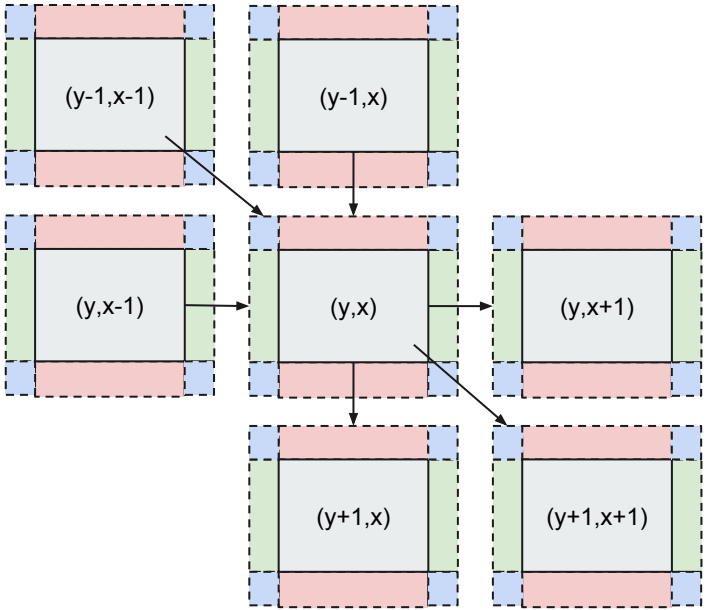
La version par blocs nécessite en plus la communication des colonnes et des coins.

Exemple avec:

UPHY ⇓

UPHY ⇒

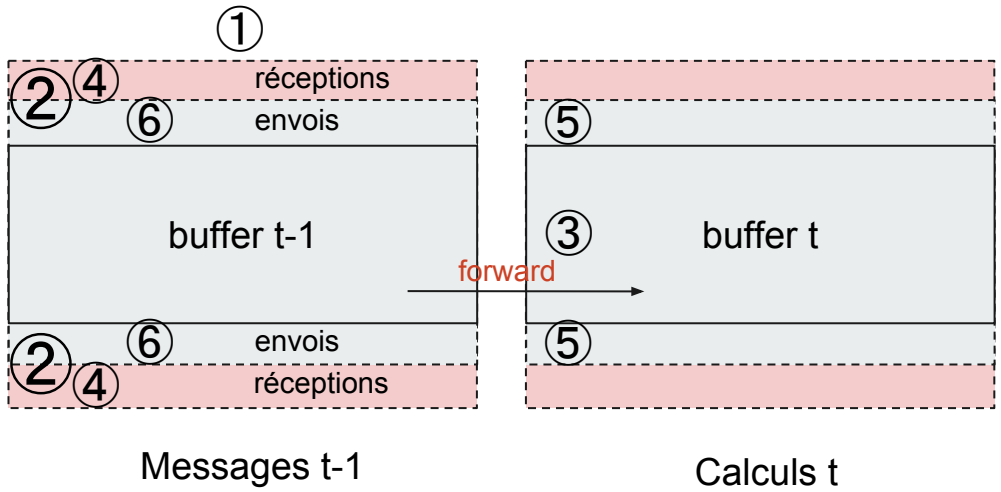
UPHY ↘



Recouvrement des communications

Recouvrement des communications:

1. Export de la grille t-1
2. Lancement échanges bords t-1
3. **Calcul du bloc intérieur t**
4. Attendre réception bords t-1
5. Calcul des bords t
6. Attendre envoi bords t-1



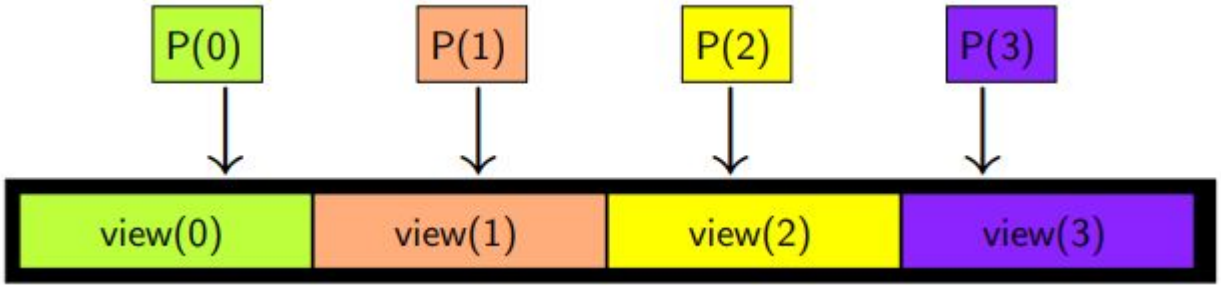
Tout t-1 échangé, et t calculé :

on peut passer à l'itération suivante t+1

Entrées sorties parallèles avec MPI I/O - bandes

`MPI_File_set_view`
accès binaire plus naturel
type émetteur -> un type fichier + offset

`MPI_File_iwrite`
écriture non bloquante
Une seule écriture à la fois, recouverte par calculs.



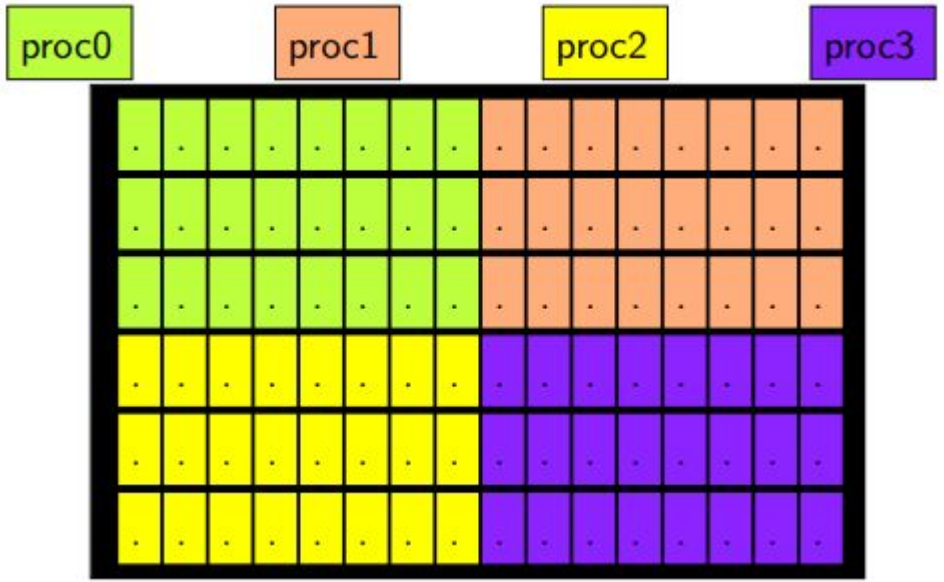
Entrées sorties parallèles avec MPI I/O - blocks

Données **non continues** dans le fichier : tableaux

Type spécialisé `MPI_Type_create_subarray` s'adapte à chaque processus

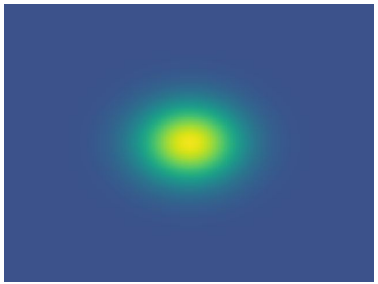
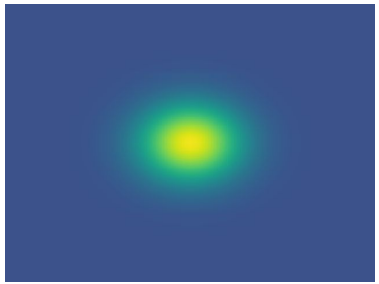
Données **non continues** en mémoire blocks :

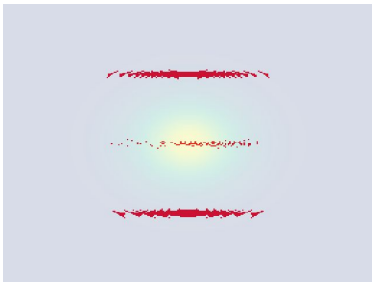
Type spécialisé `MPI_Type_contiguous` et redimensionné `MPI_Type_create_resized`.



Comment s'assurer les calculs restent corrects ?

- En utilisant MPI I/O, on peut écrire la simulation de façon distribuée
- on extrait la dernière image de la simulation
- on utilise la commande *compare* qui indique les différences en rouge.



Nous pouvons donc affirmer que notre implémentation est correcte

Résultats

Séquentiel : -x 8192 -y 8192 -t 20 : 47.827
Séquentiel : -x 32768 -y 32768 -t 40 : 1500 (extrapolé)

Pour le test de référence (cas 2) : -x 8192 -y 8192 -t 20 sur 16 noeuds :

```
3.99 (75%) make NODES=16 -C parallel test2
3.95 (75%) make NODES=16 MODE=--async -C parallel test2
3.09 (96%) make NODES=16 MODE=--block -C parallel test2
3.06 (98%) make NODES=16 MODE=--block --async -C parallel test2
```

Gros test : -x 32768 -y 32768 -t 40 sur 64 noeuds :

```
49.33 (47%) make NODES=64 -C parallel big_test
31.76 (74%) make NODES=64 MODE=--async -C parallel big_test
35.00 (64%) make NODES=64 MODE=--block -C parallel big_test
26.79 (87%) make NODES=64 MODE=--block --async -C parallel big_test
```

Résultats

Pour le test d'export (cas 3) : -x 512 -y 512 -t 40 sur 16 noeuds :

```
1.27 make NODES=16 -C parallel test3
1.25 make NODES=16 MODE=--async -C parallel test3
1.26 make NODES=16 MODE=--block -C parallel test3
1.26 make NODES=16 MODE=--block --async -C parallel test3
```

En faisant varier le pas de temps EXPORT_STEP=10 et -x 512 -y 512 -t 80 sur 16 noeuds :

```
0.48 make NODES=16 -C parallel test3
0.50 make NODES=16 MODE=--async -C parallel test3
0.47 make NODES=16 MODE=--block -C parallel test3
0.49 make NODES=16 MODE=--block --async -C parallel test3
```

Conclusion

- Messages non bloquants sur beaucoup noeuds
- Gains blocks : messages plus courts, optimisations cache
- Pas de temps petit pour la simulation, plus grand pour l'exportation
- Utiliser des types spéciaux MPI -> validation + performances
- Utiliser des variables explicites et des commentaires
- Décomposer le travail en sous objectifs et valider des portions