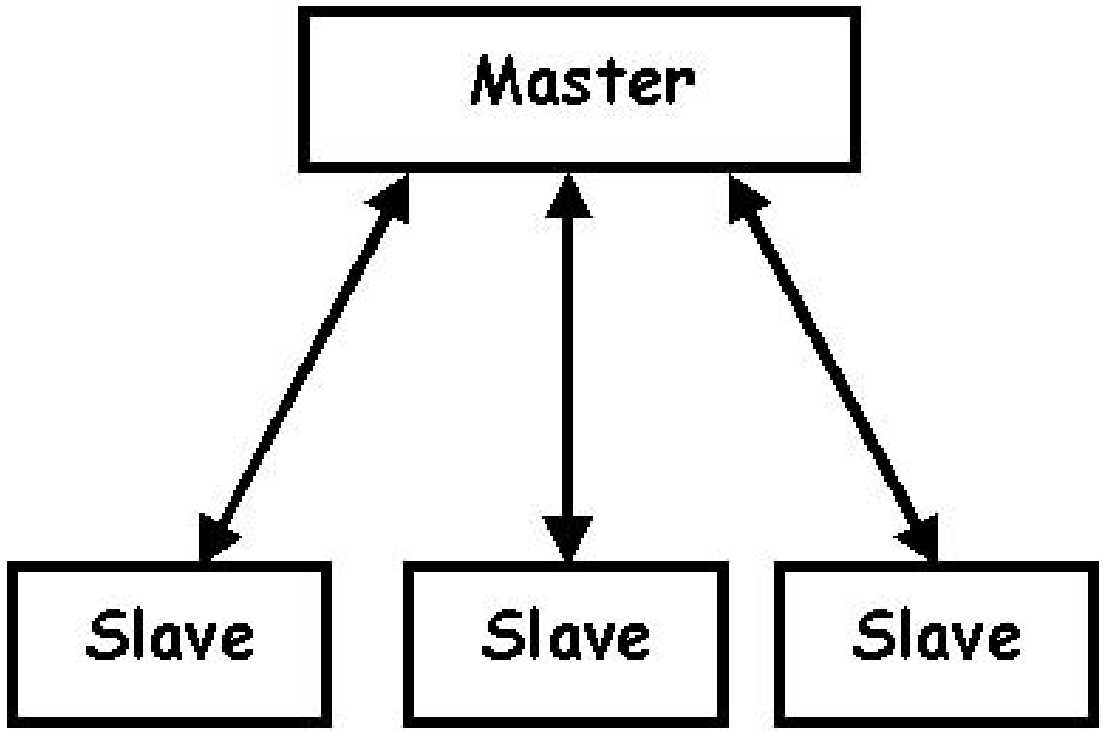
Projet de Parallélisme



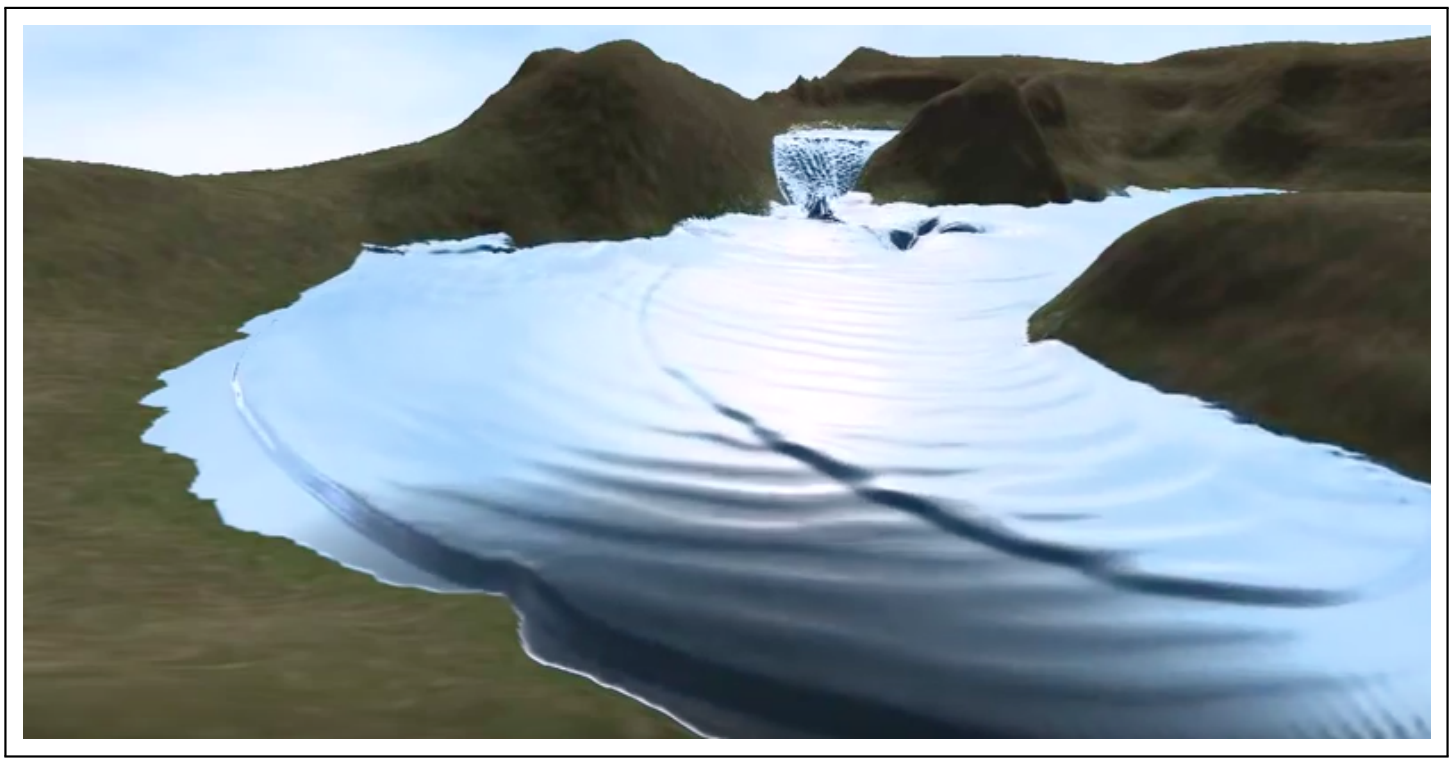
Polytech Sorbonne

09/04/2018

Fatine BENTIRES ALJ,
Alexia ZOUNIAS-SIRABELLA,

Présentation du problème

- Étude du modèle *shallow water*
- **But** : Écoulement d'un fluide sur la verticale
- **Exemples** : Ondes concentriques à la surface de l'eau, courants dans les bassins d'aquacultures



Le modèle séquentiel

- **But** : Calcule l'évolution de l'état d'une cloche centrée sur un nombre de pas de temps fixé par l'utilisateur.
- **Variables principales** : x, y les dimensions de la grille, t le nombre de pas de temps.

Agencement des dossiers

On a **plusieurs dossiers** :

- Fichiers *bin* : Bibliothèques
- Fichiers *inc* : Headers
- Fichiers *obj* : Objets
- Fichiers *src* : Fichiers sources

Dans le fichier **src**, on trouve :

- *export.c* : crée le fichier .sav
- *init.c* : initialise la grille
- *memory.c* : fait une allocation et libération de mémoire
- *parse_args.cpp* : récupère les éléments tapés au clavier
- *shalw.cpp* : fichier main

Problème

Nous allons voir deux méthodes de parallélisation :

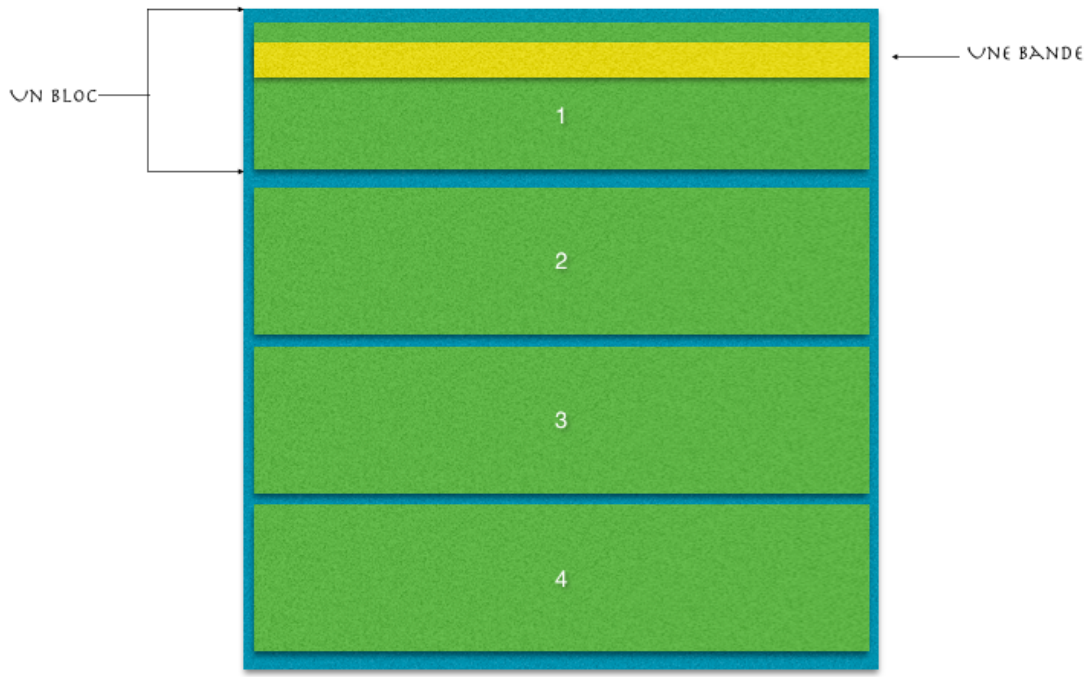
- La parallélisation par *bande*
- La parallélisation par *bloc*

$\longrightarrow$ **Quelle méthode est la meilleure ?**

Résolution par bande

Principe

- **Principe** : Découpage en *blocs*



- **Avantages** : Intuitif, "simple" à modéliser
- **Inconvénients** : Les résultats varient, les temps de calculs entre chaque processeur également

Résolution par bande

Ce que nous avons fait

- Implémentation de la méthode **bloquante** et **non bloquante**
- **Modification générale** : Définition de *g_sizex* et *g_sizey* comme la taille de la grille, sizex et sizey comme la taille de chaque bloc. Utilisation de Send, Recv, Isend et Irecv.

Résolution par bande

Communications bloquantes

- Un envoi : une réception
- Rang : permet de déterminer ce qu'on fait
- Deux MPI_Send ou MPI_Recv : **inter-blocage**
- **MPI_Scatter** pour couper et donner aux processeurs
- **MPI_Gather** pour regrouper

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

Résolution par bande

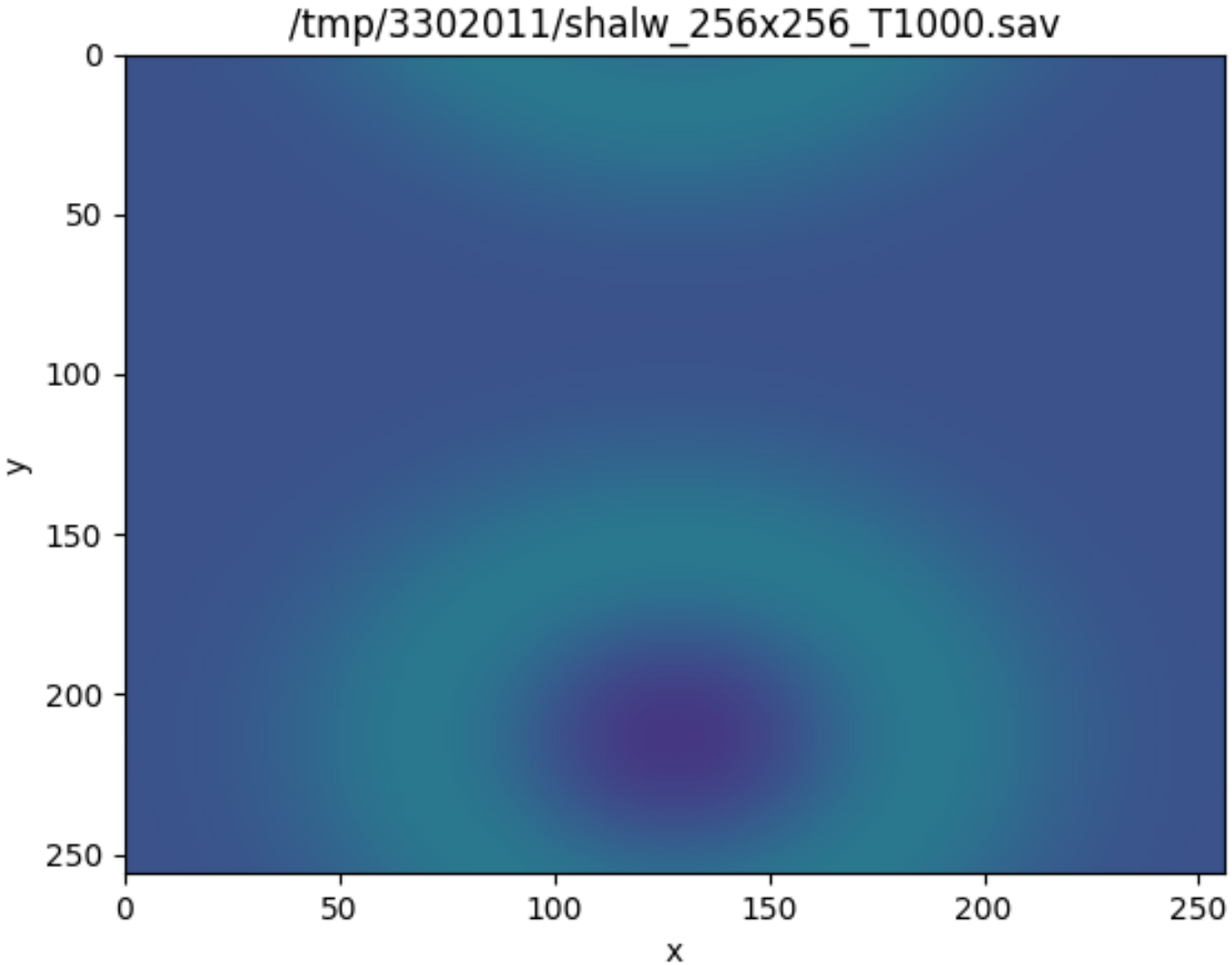
Performance

On obtient ainsi :

NP	Par défaut	N:512, t:40	N:8192, t:20
Séquentiel	18.2142 seconde(s)	3.22803 seconde(s)	961 seconde(s)
2	10.7929 seconde(s)	2.17436 seconde(s)	507.419 seconde(s)
4	12.3584 seconde(s)	2.37495 seconde(s)	605.162 seconde(s)
8	13.1102 seconde(s)	2.25222 seconde(s)	273.414 seconde(s)

Résolution par bande

Représentation communication bloquante



Résolution par bande

Communications non bloquantes

- Communication en parallèle d'autres traitements
- **Superposition** : plusieurs communications sans risque d'inter-blocage
- Besoin de l'utilisation d'un **MPI_Wait** bloquant ou MPI_Test non bloquant
- Prise des request données par MPI_Wait

```
int MPI_Isend(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int dest,
  int tag,
  MPI_Comm comm,
  MPI_Request *request
);
```

```
int MPI_Irecv(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Request *request
);
```

Résolution par bande

Performance

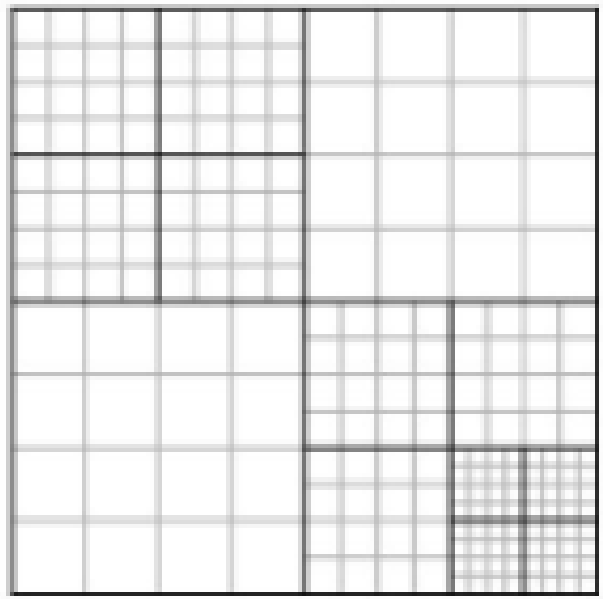
On obtient ainsi :

NP	Par défaut	N:512, t:40	N:8192, t:20
Séquentiel	1502.30 seconde(s)	302.508 seconde(s)	trop long

Résolution par bloc

Principe

- **Principe** : Découpage en *blocs* : **MPI_type_Vector**



- **Avantages :** Possiblement plus efficace
- **Inconvénients :** Plus compliqué à implémenter, temps de travail entre les processeurs peut être mal réparti

Résolution par bloc

Performance

On obtient ainsi :

NP	Par défaut	N:512, t:40	N:8192, t:20
Séquentiel	29.3671 seconde(s)	4.96369 seconde(s)	424.504 seconde(s)
2	5.54023 seconde(s)	1.65683 seconde(s)	226.798 seconde(s)
4	2.77082 seconde(s)	1.06849 seconde(s)	158.134 seconde(s)
8	6.19444 seconde(s)	1.13032 seconde(s)	161.128 seconde(s)

Problèmes rencontrés

Problèmes rencontrés

- Visualiser l'implémentation de la méthode
- **Fuites de mémoires** avec Desalloc()

- **L'exportation** d'image avec la commande export

Conclusion

$\longrightarrow$ **Quel est le meilleur choix à adopter ?**

Cela dépend clairement du type de problème. Ici, il semblerait que la parallélisation par bloc soit plus efficace...