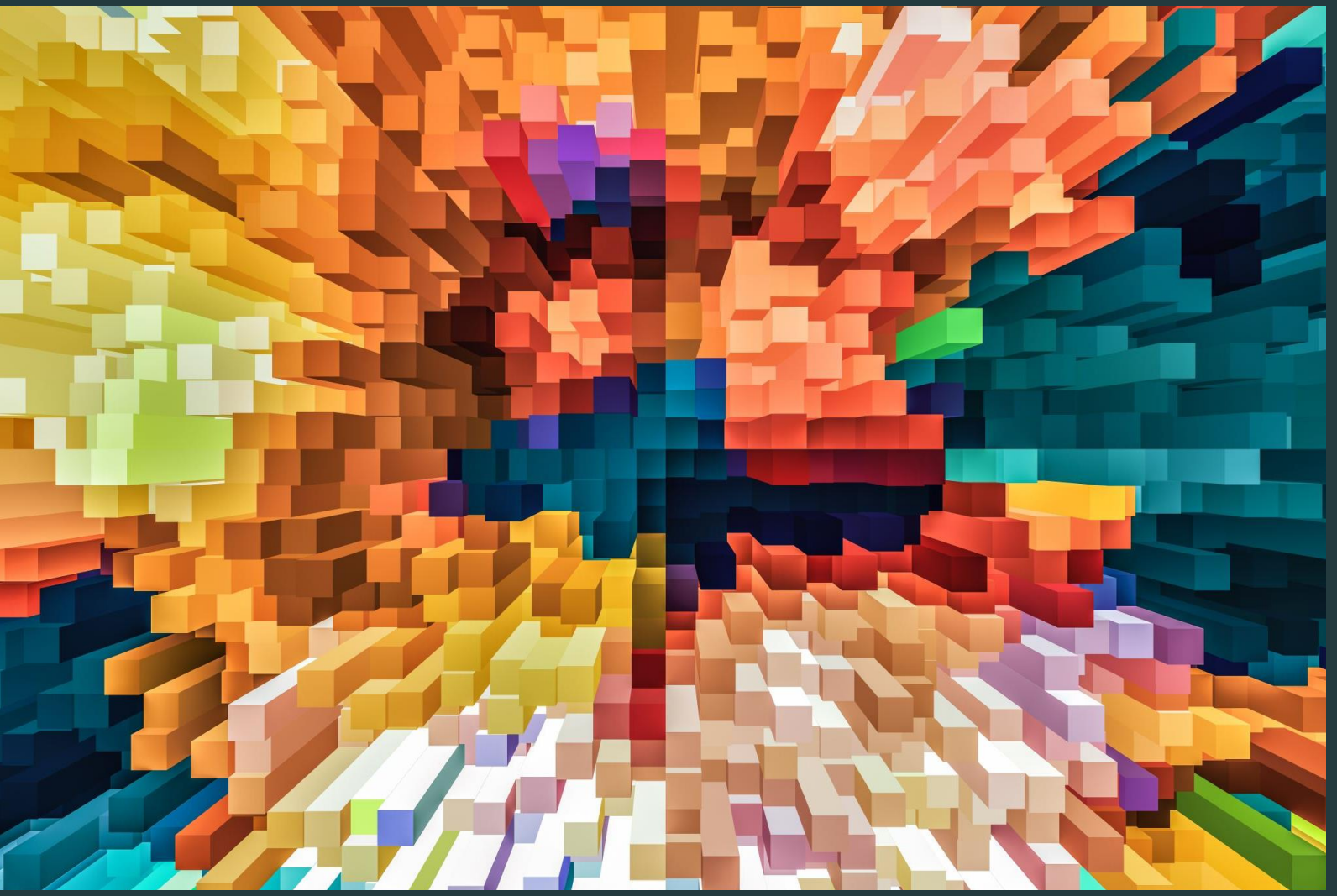


AIDES POUR LE PROJET IN203

Multithreading, MPI et cellules fantômes

Multithreading et tirage aléatoire

- La fonction rand() est une fonction donnant une valeur pseudo-aléatoire
- Utilise une graine servant à initialiser une suite ayant un comportement qui semble aléatoire
- La graine est une variable globale protégée pour le multi-threading
- Chaque accès de rand utilise des mutex → grosses pénalités pour le code multi-threadé (openMP ou Posix Thread)

Tirage aléatoire en C++ 11

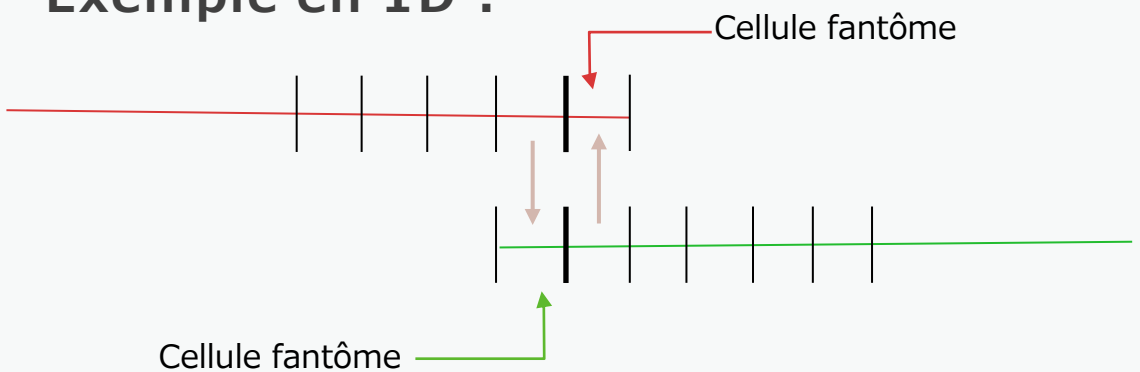
- En C++ 11 et supérieur, fonctions aléatoires proposées multithreadables
- Chaque thread utilise une graine aléaoire différente
- On en profite pour demander un tirage aléatoire uniforme
- `std::minstd_rand0` → même type de tirage aléatoire que rand()
- `std::uniform_real_distribution<double>` → permet un tirage uniforme
- Bon speed-up pour le projet !

Sous-grilles et cellules fantômes

- Pour paralléliser en MPI, on découpe la grille en sous-grille;
- Chaque cellule d'une sous-grille doit connaître l'état de ses cellules voisines (gauche, droite, haut, bas);
- Pour les cellules au bord d'une sous-grille, comment connaître les valeurs d'une cellule se trouvant sur une autre sous-grille ?
- Par raisonnable d'échanger cellule par cellule (vous chercherez pourquoi…)

Sous-grilles et cellules fantômes

- On va rajouter une rangée de cellule à gauche, à droite, en haut et en bas
- Ces cellules sont appelées cellules fantômes
- Servent de zone de réception des valeurs des cellules des sous-grilles voisines
- Exemple en 1D :



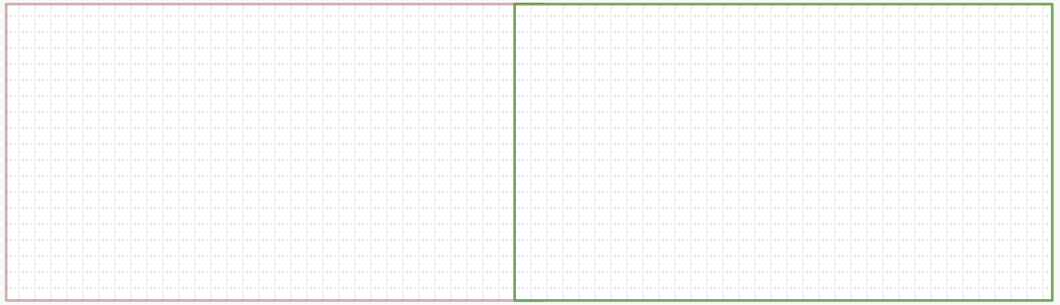
Sous-grilles et cellules fantômes

- Même principe en 2D, 3D…
- On regroupe les données à envoyer sur les cellules fantômes d’une autre grille puis on les envoie
- On reçoit ses données qu’on distribue sur les cellules fantômes
- Si l’ordonnancement déjà compatible, on peut directement recevoir ou envoyer les données sans les regrouper
- Mais ne pas faire d’envoie ou de réception cellules par cellules !

Sous-grilles et cellules fantômes

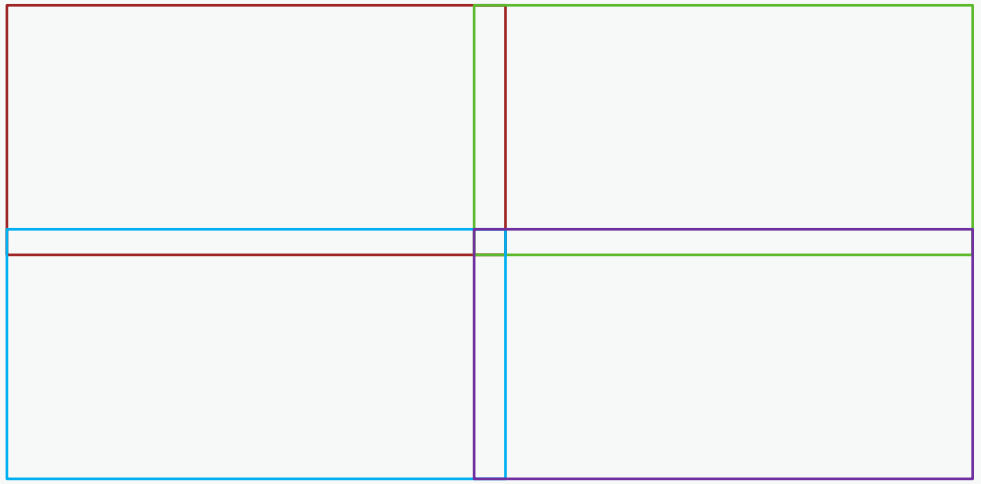
- **Découpage par tranche d'une grille NxN :**

- Semblable au cas 1D : si les données sont rangées par colonne, pas de buffer d'envoie/réception à créer
- On échange entre deux grilles N données et une grille a en général deux voisins (sauf au bord)
- Pour P grilles, on échange 2NP données



Sous-grilles et cellules fantômes

- Découpage cartésien : on découpe dans les deux directions du plan



- Pour chaque sous-grille, en général, quatre voisins
- On échange 2N/P données avec un voisin
- En tout, 4N données échangées au total
- Par contre, utilisation d’un buffer obligatoire pour au moins une direction

OpenMP, Multithread et MPI

- **Fonctions de MPI par défaut non multithreadable**
- **Si on remplace MPI_Init par MPI_Init_thread : permet d'utiliser MPI dans un contexte multithreadé**
- MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
- Argc, argv : pareil que pour MPI_Init
- Required : **MPI_THREAD_MULTIPLE (valeur la plus tolérente)**
- Provided : Donne le niveau multithreadable possible (MPI_FUNNELED : main thread only, MPI_THREAD_SERIALIZED : zone protégée, les threads passent un par un, MPI_THREAD_MULTIPLE : les threads peuvent exécuter la fonction en même temps)

OpenMP, Multithread et MPI

- Si plusieurs threads d’un seul nœud envoient sur le même nœud destinataire, attention aux conflits entre les messages
- Bien penser à utiliser les identificateurs de message (tag) pour différencier les messages