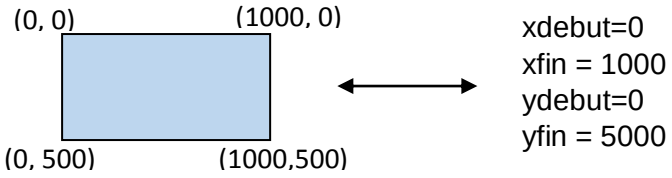
LIU Zhuying 11306849
FLORES Isabelle 11102272

Rapport Projet Content Addressable Network, Mif32 Parallélisme

Pour **stocker un espace**, nous avons simplement une structure qui contient quatre entiers:

```
typedefstruct espace {
        int xdebut;
        int xfin;
        int ydebut;
        int yfin;
}Espace;
```



Pour **stocker les voisins et les données**, nous avons hésité de passer en C++ pour utiliser la librairie STL. Finalement nous avons trouvé GSList qui est une librairie *Open-Source* en c. Celle-ci est une liste générique et simplement chaînée. Chaque voisin est un tableau de trois entiers qui représentent le numéro du processus et l'identifiant du nœud.

```
typedefstructvoisins{
        GSList *voisinsHaut;
        GSList *voisinsBas;
        GSList *voisinsGauche;
        GSList *voisinsDroite;
}Voisins;

typedefstruct donnee{
        int pos[2];
        int valeur;
} Donnee;

typedefstruct donnees {
        GSList *contenu;
}Donnees;
```

Insertion des nœuds:

Le processus 0 envoie l'autorisation d'insertion à chaque processus et attendre la fin de l'insertion, ensuite il envoie l'autorisation d'insertion au prochain processus.

Chaque processus attend l'invitation d'insertion et envoyer aux processus déjà inséré son identifiant, et le processus qui a la responsabilité de cet identifiant lui attribue un espace. Une fois un processus a eu son espace distribué, il annonce au processus 0 la fin d'insertion.

Connaissance des voisins:

Quand tous les processus sont insérés, chaque processus envoie son espace à tout le monde. Et chacun calcul si l'espace reçu est son voisin.

Calcul des voisins:

Pour le nœud rose, un de ses voisins en haut a trois situations.

rose.xdebut<= bleu.xdebut<rose.xfin

rose.xdebut<bleu.xfin<=rose.xfin

rose.xdebut>= bleu.xdebut&&rose.xfin<= bleu.xfin

Dans ces trois cas, *bleu.yfin == rose.ydebut – 1*.

Insertion des données:

Le processus 0 envoie chaque coordonnée à un processus aléatoire, et attendre la réponse du processus qui a la responsabilité de cette coordonnée, ensuite il lui envoie la donnée.

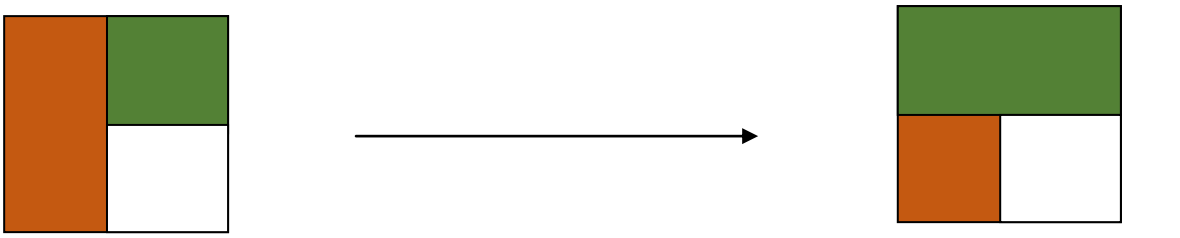
Chaque processus attend un message de deux entiers de n'importe quelle source. Si ces deux sont positifs, il s'agit d'une coordonnée. Le processus vérifie si cette coordonnée est dans son espace, si oui il répond au processus 0, sinon il trouve un voisin de la bonne direction et envoyer cette coordonnée à ce voisin. Si ces deux entiers sont (-1,-1), cela indique la fin d'insertion des données.

La recherche des données est similaire à l'insertion des données.

Suppression d'un nœud:

Le nœud en suppression parcourt un par un ses voisins. Il demande au voisin en cours d'envoyer son espace, si possible il découpe une partie de son espace et distribuer cet espace au voisin. Il trouve ensuite les données stockées dans cette partie et envoyer au voisin. Quand tout son espace est distribué et toutes les données sont envoyées, la suppression est terminée. Ce nœud vide ses données et ses voisins stockées. Les voisins qui ont reçu son espace et ses données en partie ou entièrement, ajoutent l'espace à l'existant, et ajoutent les données reçues dans son stockage de données. A la fin, tous les processus recalculent leurs voisins en *broadcast*. (On aurait pu faire la mise à jour des voisins avec juste les voisins du nœud supprimé. Cela serait plus efficace.)

Remplacement partiel du nœud en suppression:



(vert.yfin<=orange.fin) && (vert.ydebut==orange.ydebut)

vert.ydebut = orange.ydebut
orange.ydebut = vert.yfin

Transmission des données:

Le nœud en suppression envoie d'abord un entier pour prévenir la taille des données à venir, ensuite il envoie un tableau d'entier de la taille promise.

Fichier log:

Notre n'avons pas pu faire les fichiers de log demandés, mais nous avons donné un exemple d'exécution avec les affichages que nous jugeons utiles.