Ray Marcelin **RA-IL1**
Quilliou Erwan
Mougel Antonin

Consignes

Lien du dépôt git: https://github.com/Erwanquilliou/TpProgPartage/

Ce projet est à réaliser en groupes de 4. Vous devrez rédiger un rapport pour illustrer votre travail et fournir un lien vers un dépôt git public contenant votre code. Pour information, le DS contiendra quelques questions sur ce projet.

Votre rapport devra contenir :

- *les réponses aux questions du sujet*
- *une description de l'architecture complète (schéma) de votre application ainsi que les rôles de tous ses composants.*
- *le code des interfaces utilisées (le reste étant dans le dépôt)*
- *un diagramme de séquences incluant tous les acteurs (processus) qui illustre l'ordre des étapes lors d'une exécution complète. Indiquez le nom des méthodes distantes utilisées sur ce diagramme.*

Déposez sur votre GIT une animation (capture d'écran) qui montre les différentes parties de l'application en exécution. Pensez a faire des affichages sur les différents terminaux qui permettent de suivre cette exécution.

Préambule

> *Maintenant que vous avez développés différentes applications réparties dans différentes situations. Nous terminons par cet ultime projet qui illustrera un nouvel aspect de la programmation répartie : le calcul parallèle. Il se décline sous différentes formes, et nous verrons ici sa forme la plus simple, la parallélisation des données. Ici, on souhaite réaliser un traitement impossible à réaliser sur un ordinateur personnel pour manque de puissance de calcul. Ayant accès à un ensemble de machines, nous décidons de découper ce gros calcul en petits morceaux et de les transmettre à ces machines. Elles calculeront et renverront chacune une partie du résultat. Il faudra ensuite assembler les différents morceaux pour obtenir le résultat final.*

Question

> *En utilisant votre meilleur outil, votre imagination, décrivez et illustrez comment cela pourrait être réalisé, sans rentrer dans les détails JAVA, que vous n'allez pas tarder à mettre en œuvre.*

Nous découpons l'image en un nombre élevé de morceaux. L'objectif est de donner ces morceaux aux différentes machines disponibles qui vont les calculer. Pour cela, nous pouvons boucler le nombre de morceaux de l'image. Nous ne pouvons pas découper l'image avec le nombre de machines disponibles car ces machines peuvent s'ajouter ou se déconnecter.

Nous souhaitons exécuter les calculs de manière asynchrone donc en utilisant des threads. Il faut cependant faire attention à ce que plusieurs machines n'affichent pas la même image.

A la manière du tableau blanc, le service central permettrait d'inscrire un nouvelle utilisateur et de donner leurs listes tandis que le client appelerait l'action a réaliser (ici calculer l'image). Cela appellerait les machines "esclaves" qui calculent l'image.

Le tracé de rayon (raytracing)

Nous vous proposons un type de calcul très gourmand en cycles CPU : la synthèse d'images. Avant l'avènement de Dall.e ou de Stable Diffusion auxquelles on demande en quelques phrases de nous créer une œuvre sortie de leur imagination, synthétiser une image nécessitait un calcul fastidieux, qui a été plus tard délégué à des processeurs dédiés, les GPU. Il fallait souvent décrire une scène non pas en langage naturel mais sous la forme d'un ensemble de formes géométriques et de sources lumineuses. L'algorithme de synthèse d'image calculait la scène pixel par pixel, pour enfin produire une image. Voir ici pour les curieux.

L'archive fournie, est une implantation naïve en JAVA d'un tel algorithme. La scène calculée est décrite dans le fichier simple.txt.

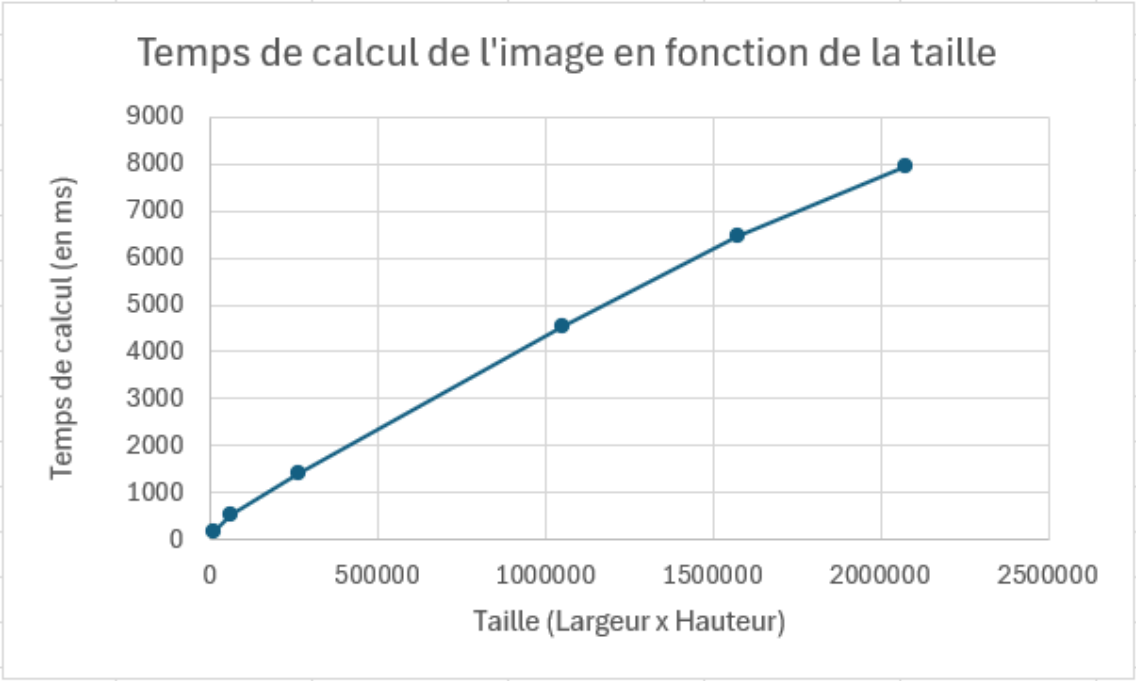
Questions

1. *Tester le programme en modifiant ses paramètres (sur la ligne de commande).*
2. *Observer le temps de d'exécution en fonction de la taille de l'image calculée. Vous pouvez faire une courbe (temps de calcul et tailles d'image).*
3. *En ne modifiant que le fichier LancerRaytracer.java, reproduire l'image suivante :*

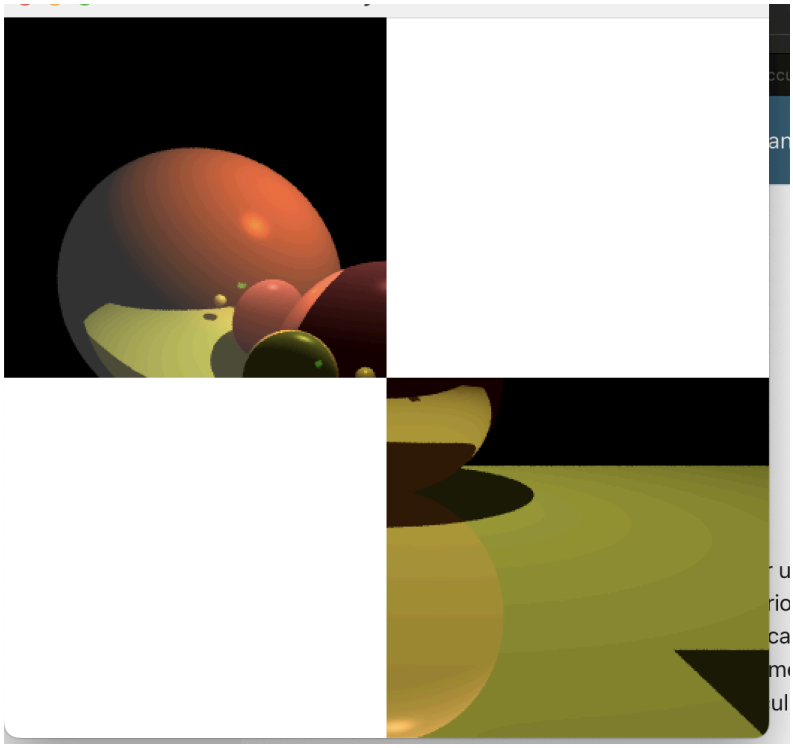
1: Le programme fonctionne correctement. Nous pouvons modifier différents paramètres comme la taille de l'image par exemple.

2: Toutes les données ont été prise sur le même ordinateur dans les mêmes conditions (l'une après l'autre) afin de garder les données les plus cohérentes possible

Largeur	Hauteur	Taille (Largeur	Temps de calcul (en ms)
100	100	10000	167
250	250	62500	512
512	512	262144	1396
1024	1024	1048576	4540
1024	1536	1572864	6468
1080	1920	2073600	7942



3:



Pour faire l'image:

```
int x0 = 0, y0 = 0;
int l = largeur/2, h = hauteur/2;
```

On divise la taille L/h en 2, et on garde la position 0, 0

```
//compute la deuxieme partie (bas droite)
int x02 = largeur/2;
int y02 = hauteur/2;
l = largeur/2;
h = hauteur/2;
```

On refait un calcul pour la partie en bas à droite (et on redéfini bien x0 et y0, car on les utilise pour dire où dessiner les dessins dans l'image)

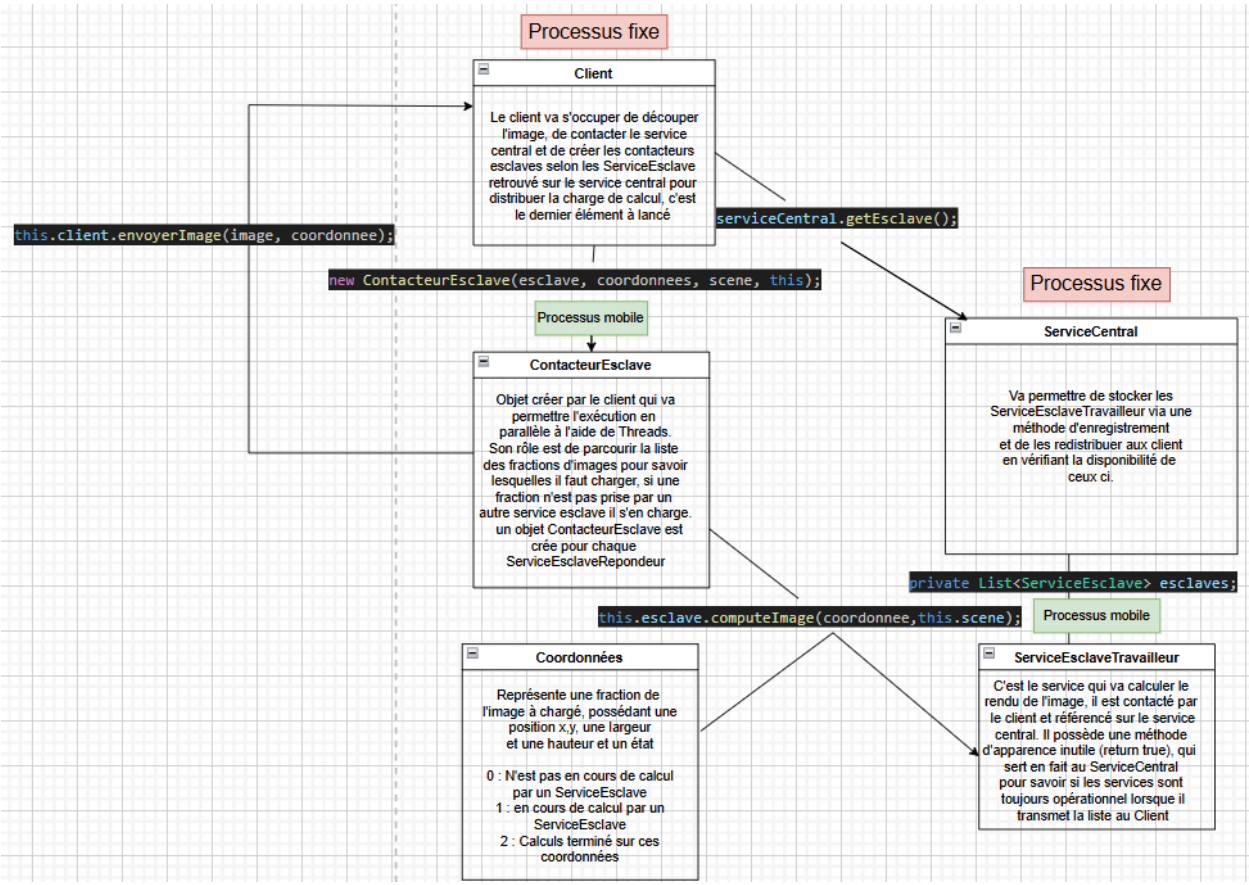
Accélérons les choses

Vous constatez rapidement que pour calculer une image de bonne résolution il faut s'armer de patience. Pour cela nous allons devoir paralléliser les calculs sur un ensemble de machines. Dans notre scénario nous disposons de :

- *Un ensemble de nœuds de calcul, capables de calculer un morceau d'une scène.*
- *Un serveur de nœuds qui nous permet de récupérer les coordonnées des nœuds de calcul.*
- *Un programme qui découpe le calcul, récupère les coordonnées des nœuds disponibles, envoie un calcul sur chacun et affiche le résultat.*

Questions

- *Faire un petit schéma de cette architecture en identifiant les choses suivantes :*
    1. *Le/les processus fixes (ceux qui écoutent sur un port choisi) et les processus mobiles ? (ceux qui rentrent et sortent a leur guise) ?*
    2. *Les types des données échangées entre les processus*



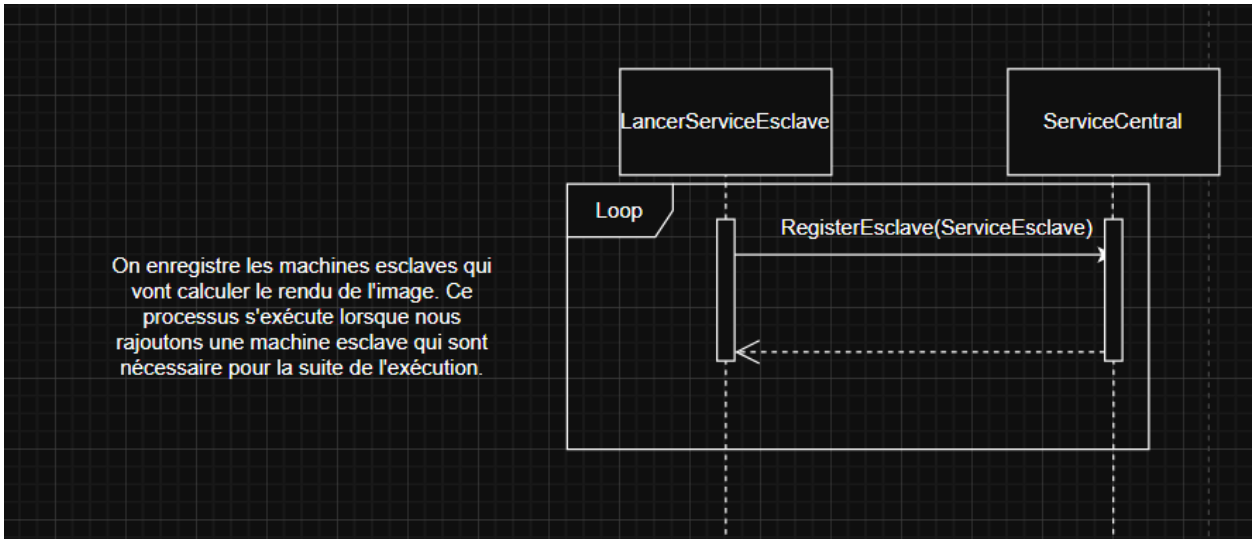
- *Si on veut que les calculs se fassent en parallèle, que faut-il faire ?*

  Pour que les calculs se fassent en parallèle, la première étape est de séparer tous les calculs à réaliser. Une fois cette séparation effectuée, le service central peut boucler par le nombre de séparation que nous voulons effectuer et les donner aux différents clients jusqu'à ce que l'image soit correctement affichée.
  À l'image du TP10, si nous voulons que ces calculs se fassent en parallèle, nous pouvons utiliser des threads. Nous aurions donc une exécution synchronisée.

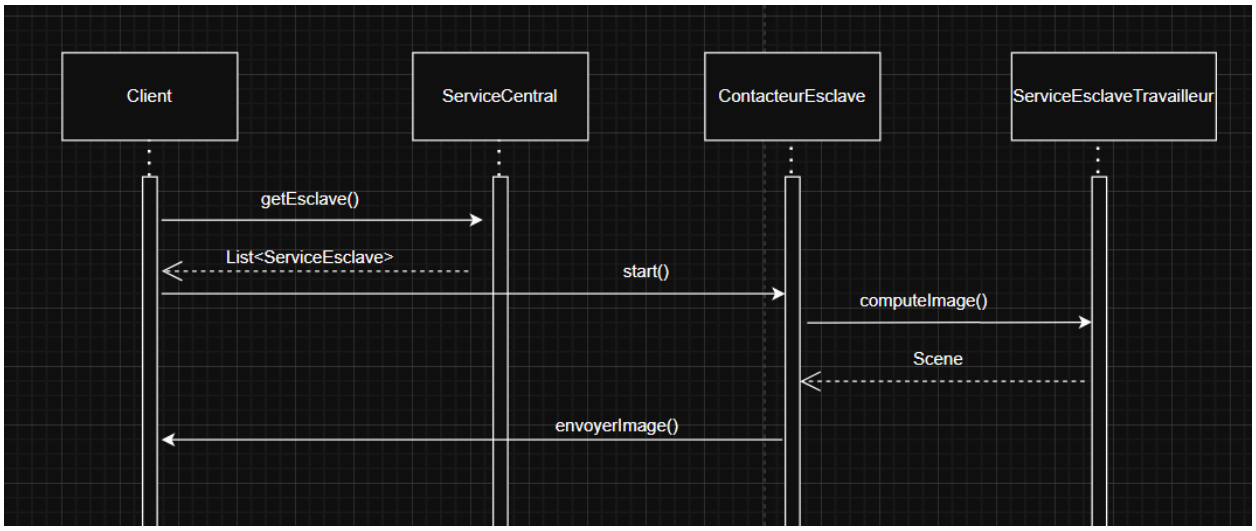
- *Lancez-vous et réalisez cette application répartie ! Vérifiez que le calcul est bien accéléré.*

DIAGRAMME DE SÉQUENCE:

Enregistrer une machine esclave:



Calculer une image:



CODE DES INTERFACES:

Interface ServiceEsclave :

```
public interface ServiceEsclave extends Remote{
    public boolean isAwake() throws RemoteException;
    public Image computeImage(int x, int y, int length, int height, Scene scene) throws
RemoteException;
}
```

Interface ServiceCentral :

```
public interface ServiceCentral extends Remote {
    public void registerEsclave(ServiceEsclave s) throws RemoteException;
    public List<ServiceEsclave> getEsclave() throws RemoteException;
}
```