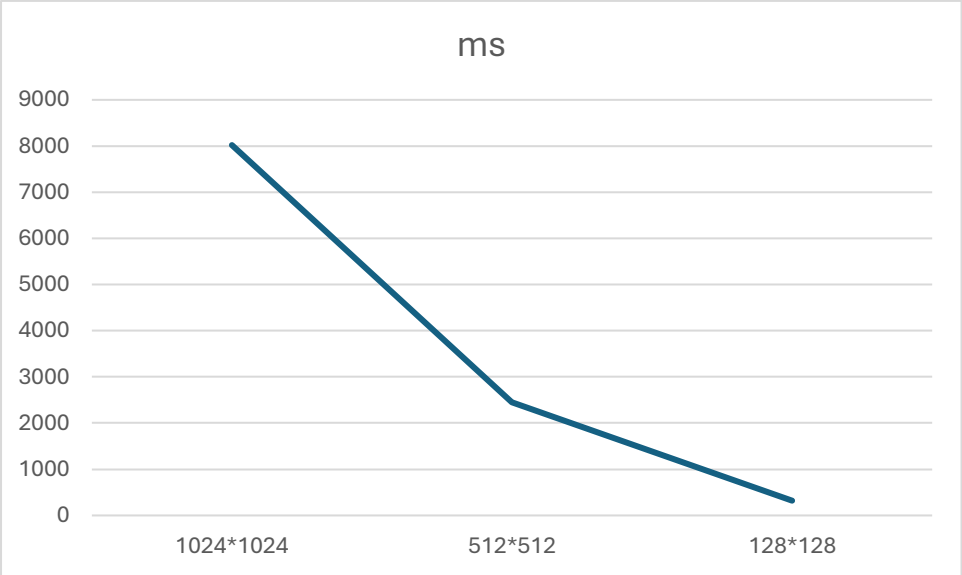
Projet notée calcul parallèles

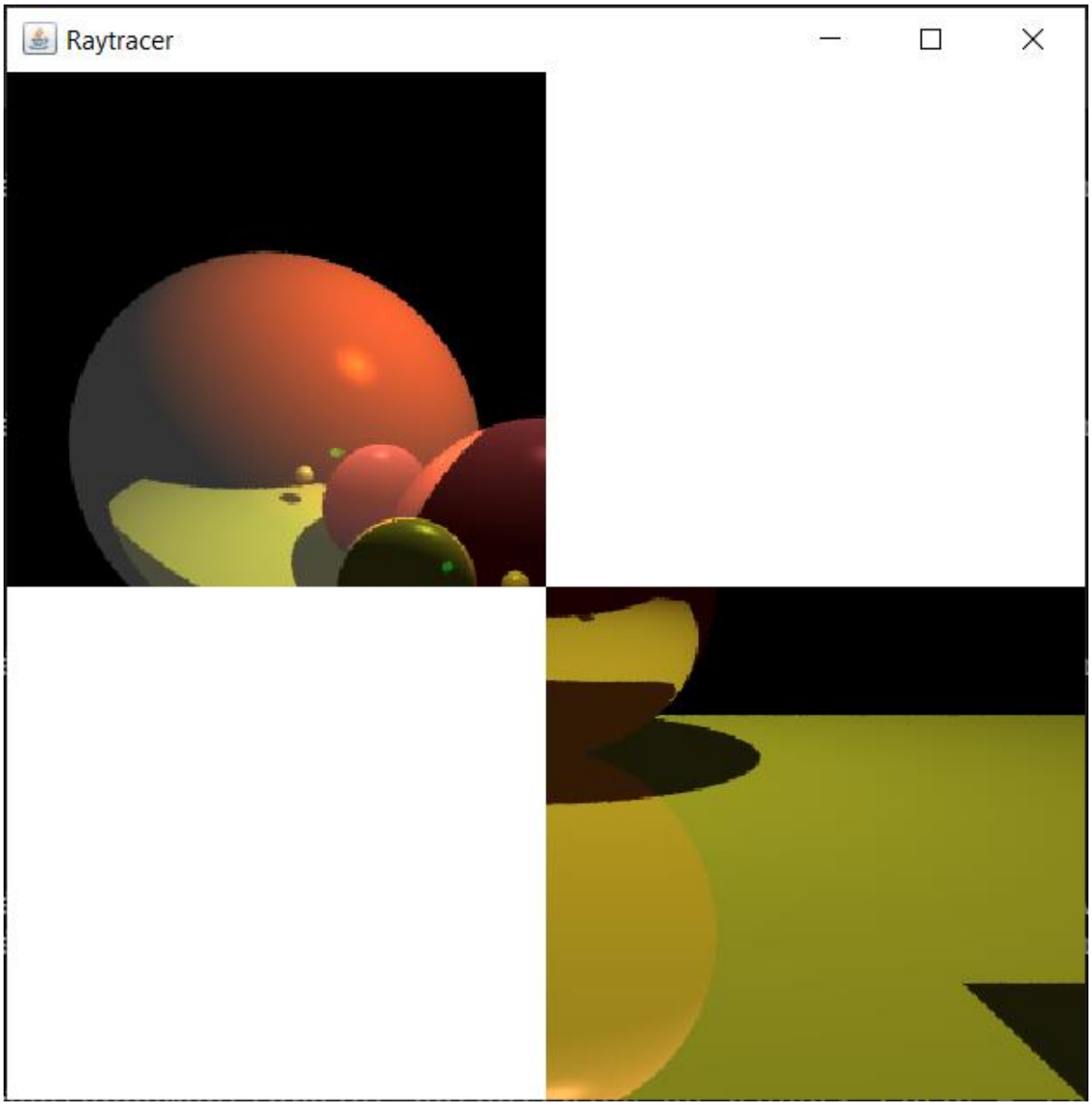
Préambule : Pour réaliser notre découpage on pourrait faire une architecture avec un cerveau principale et des « branches », c'est-à-dire que le cerveau va couper le calcul en plusieurs sous-calcul et les envoyés aux différentes machines, une fois cela fait les machines vont réaliser leurs calculs et renvoyé le résultat au cerveau qui va faire le résultat final.

Test du programme :



On remarque que plus l'image est grande plus le temps de calcul est long.

3.



J’ai dans un premier temps découper l’image en 4 partie égale :

```
int l = largeur / 2;
int h = hauteur / 2;
```

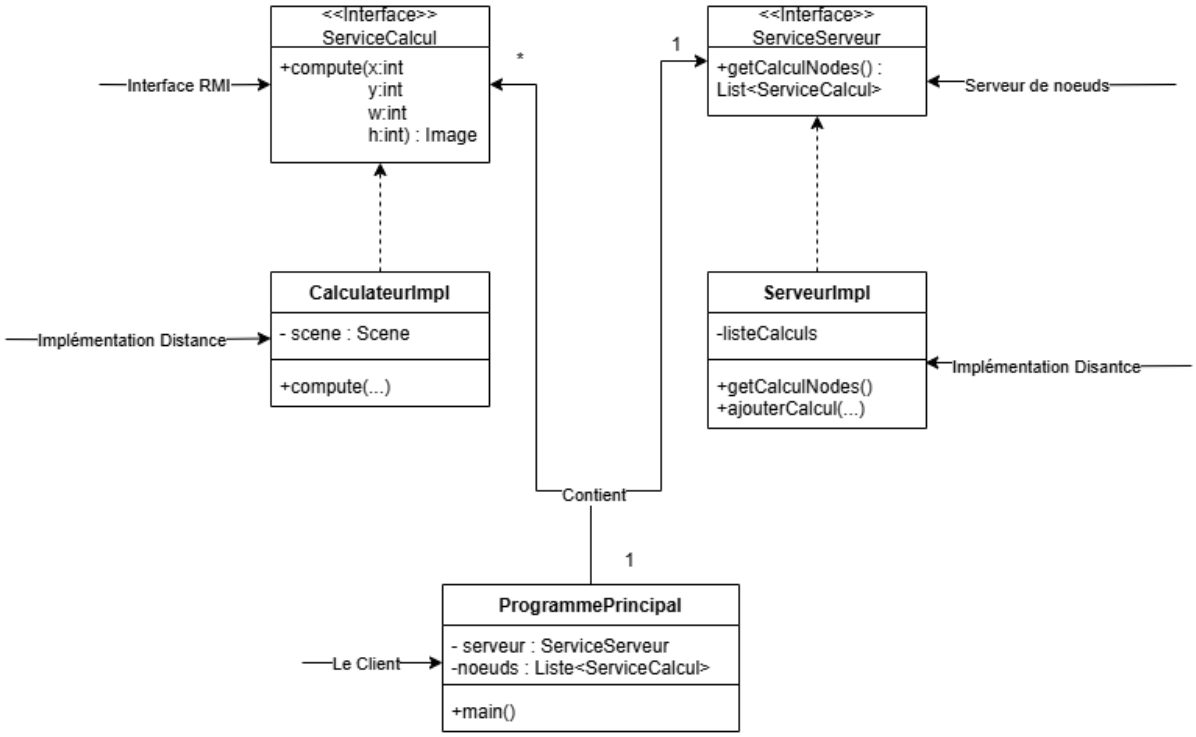
On fera une exécution en 512*512 qu’on transforme en deux blocs de 256*256.

Et ensuite on calcul les blocs qu’on souhaite afficher :

```java
// Bloc en haut à gauche
Image hautGauche = scene.compute( x0: 0,  y0: 0, l, h);
disp.setImage(hautGauche,  x0: 0,  y0: 0);

// Bloc en bas à droite
Image basDroit = scene.compute(l, h, l, h);
disp.setImage(basDroit, l, h);
```

Schéma



Raisonnement :

1. Quels sont les processus fixes et les processus mobiles ?

- **Processus fixes :**
    - Le **Serveur de nœuds (ServeurImpl)** : il reste actif en permanence, écoute sur un port RMI défini, et permet aux autres composants de s'y enregistrer ou de récupérer des adresses.
    - Les **Calculateurs (CalculateurImpl)** : bien qu'ils puissent être lancés dynamiquement, une fois démarrés, ils deviennent accessibles via RMI à un port connu. Ce sont donc eux aussi des processus fixes tant qu'ils sont disponibles.

- **Processus mobiles :**
    - Le **programme principal (client)** qui effectue le rendu final : il ne reste pas connecté de façon continue, mais utilise temporairement les services des autres. Il se lance, effectue ses appels à distance, récupère les données et s'arrête.

2. Types de données échangées entre les processus

- Entre le **client et le serveur de nœuds** :
    - Une **liste d'objets distants** List<ServiceCalcul> (références RMI) représentant les nœuds de calcul disponibles.
- Entre le **client et les nœuds de calcul** :
    - Appels de méthode compute(x, y, w, h) avec :
        - Coordonnées (int) : **x, y, w, h** définissant un bloc à calculer
        - En retour : un **objet Image** représentant une portion d'image calculée
- Eventuellement, entre les **nœuds de calcul et le serveur** :
    - Le **serveur reçoit des références RMI** (objets ServiceCalcul) lorsque les calculateurs s'enregistrent avec une méthode comme ajouterCalcul.

Chaque portion de l'image à calculer (un bloc) peut être envoyée à un nœud dans un thread distinct. Cela permet que tous les nœuds travaillent en même temps au lieu d'attendre les uns après les autres.

Exécution

Contexte de mon environnement, j'ai un pc portable pas très puissant qui va donc servir de serveur et d'affichage, j'ai un autre pc qui lui est beaucoup plus puissant c'est donc lui qui va faire les calculs.

Donc dans un premier temps sur le pc serveur on lance notre serveur de nœud :

J'utilise ceci : java -Djava.rmi.server.hostname=192.168.1.138 LancerServeur pour forcer la bonne adresse sinon ma machine de calcul reçois l'adresse de WSL c'est problème apparemment fréquent avec RMI sur des machines avec plusieurs interfaces réseau.

```
C:\Users\julie\OneDrive\Bureau\BUT2-S4\Qualité de développement\projet notee\tracé_de_rayon>java -Djava.rmi.server.hostname=192.168.1.138 LancerServeur
Serveur RMI prêt !
```

On peut voir que notre serveur est bien lancé et attend la connexion de client.

Ensuite côté de la machine de calcul dans un premier on lui donne les .class des fichiers suivants : tous les fichiers dans raytracer, les interfaces ServiceServeur et ServiceCalcul, CalculateurImpl et LancerCalculateur.

Ensuite on exécute LancerCalculateur :

```
C:\Users\julie\Desktop\calculateur>java LancerCalculateur
N?ud de calcul enregistré.
```

Et on peut apercevoir sur le terminal du serveur qu'un nouveau est bien présent :

```
Serveur RMI prêt !
N?ud enregistré, total : 1
```

Ensuite sur la machine Serveur on peut lancer le programme principal avec la taille qu'on veut :

```
C:\Users\julie\OneDrive\Bureau\BUT2-S4\Qualité de développement\projet notee\tracé_de_rayon>java ProgrammePrincipal simple.txt 512 512
Image calculée en parallèle en : 1501 ms
```

L'image s'affiche bien et si on compare avec la toute première exécution sur une seule machine on gagne 1000ms on passe d'environ 2500 ms a 1500 ms.

On peut rajouter des nœuds de calcul sur la même machine de calcul si on veut que ça aille encore plus vite, on peut même le faire avec plusieurs machines si on en a.

Pour éviter de faire à chaque fois toute les commandes on pourrais faire un script Bash coté serveur et client qui lance notre serveur avec simplement un double clic cela peut être pratique surtout que la commande pour lancer le serveur est longue.

Test quand on rajoute plusieurs nœuds avec la taille la plus grande que j'ai tester sur le graphique en haut :

Test avec 3 nœuds :

```
Serveur RMI prêt !
N?ud enregistré, total : 1
N?ud enregistré, total : 2
N?ud enregistré, total : 3
```

Avant d'après notre graphique on était à environ 8000ms, maintenant avec notre application répartie on peut calculer cette taille en :

```
C:\Users\julie\OneDrive\Bureau\BUT2-S4\Qua
le.txt 1024 1024
Image calculée en parallèle en : 3106 ms
```

Donc on peut voir que notre système est très efficace et permet de gagner beaucoup de temps surtout si les calculs sont très gourmands cette implémentation sera très efficace.

Le fichier calculateur est le fichier que tous les clients doivent avoir pour lancer les calculs et se connecter au serveur de nœuds.

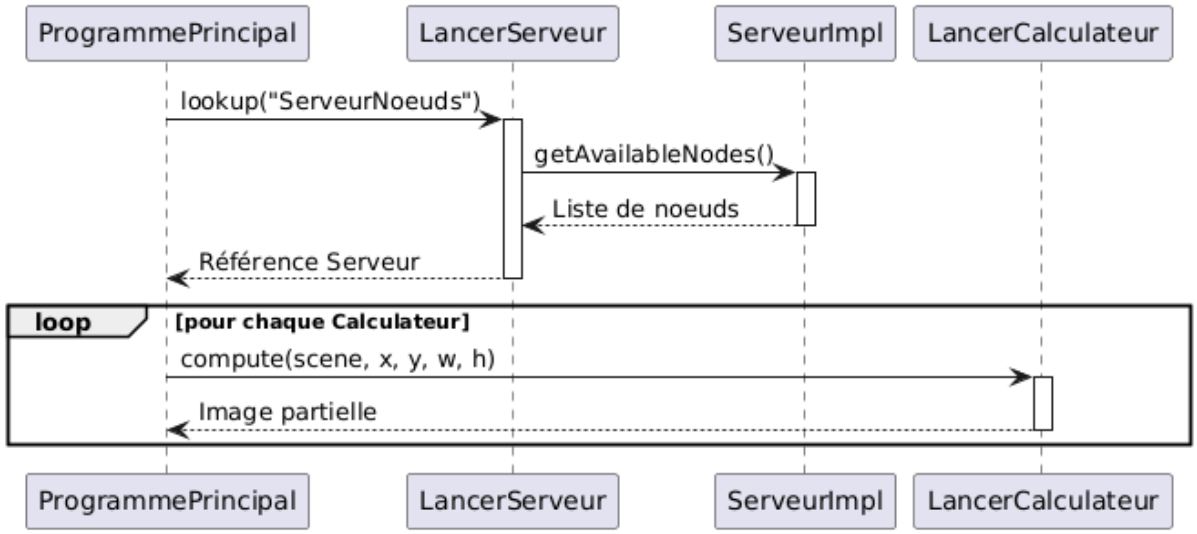
```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface ServiceServeur extends Remote {
    void registerNode(ServiceCalcul node) throws RemoteException;
    List<ServiceCalcul> getAvailableNodes() throws RemoteException;
}
```

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import raytracer.Image;
import raytracer.Scene;

public interface ServiceCalcul extends Remote {
    Image compute(Scene scene, int x, int y, int w, int h) throws RemoteException;
}
```

Diagramme de séquence pour le lancement du programme principale afficher une image de la taille voulu :



Explication retard :

Je tiens à mettre ceci pour m'excuser du retard de se devoir, j'ai eu des problèmes personnels et de santé durant les deux semaines du lundi 26 mai au vendredi 6 juin. Ces problèmes ont causé un grand retard sur ce projet en plus que je le réalise seul.