Rapport projet de MPI

Beuque Eric

29 janvier 2009

Table des matières

1 Introduction

Ce rapport fait partie du projet de MPI réalisé au sein du module de Systèmes Distribués, durant ma deuxième année de Master d'Informatique à l'UFR Sciences et Techniques de Besançon. Ce projet consistait à réaliser le calcul de l'ensemble de Mandelbrot en calcul distribué en utilisant la bibliothèque MPI en C++. Cet ensemble est un calcul permettant de dessiner des fractales (voir figure 1).

Ce calcul a été mis en oeuvre sous trois formats :

- Calcul séquentiel
- Calcul parallèle à équilibrage de charge statique
- Calcul parallèle à équilibrage de charge dynamique

A partir de là, une courte étude comparative sur les performances des trois méthodes nous étaient demandés.

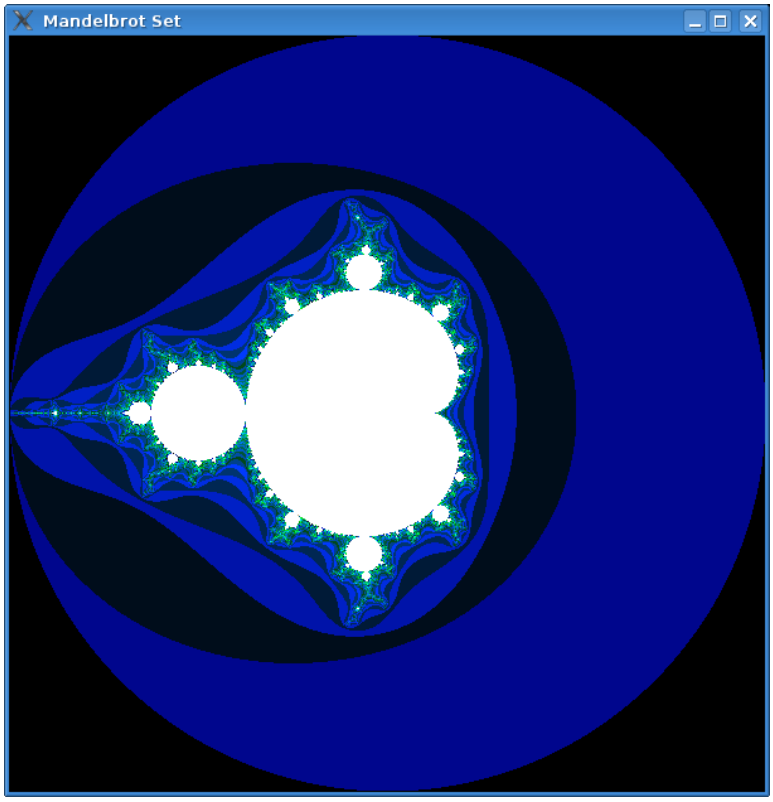

FIG. 1 – Fractale représentant l'ensemble de Mandelbrot

2 Calcul séquentiel

Le calcul séquentiel nous a été fourni et consiste juste à réaliser le calcul de chaque pixel de façon séquentiel sans utiliser MPI.

L'algorithme (voir algorithme 1) consiste en fait à imbriquer deux boucles sur la hauteur et la largeur de l'image et on calcul un par un les pixels de l'ensemble qu'on affiche directement dans l'interface.

Algorithm 1 Calcul de l'ensemble de Mandelbrot en séquentiel

Require: $largeur > 0$ and $hauteur > 0$ {Taille de l'image}
Require: $calculPixel(i, j)$ {Calcul la couleur d'affichage du pixel (i,j)}
  for $i = 0$ to $hauteur$ **do**
    for $j = 0$ to $largeur$ **do**
      $couleur \leftarrow calculPixel(i, j)$
      $afficherPoint(j, i, couleur)$
    end for
  end for

3 Calcul parallèle

Pour les calculs parallèles, on utilise donc la bibliothèque MPI qui permet de réaliser facilement un programme distribué qui peut utiliser plusieurs machines pour les calculs. MPI permet alors de se passer de la gestion de la couche réseaux du programme, et gère automatiquement la création des processus répartis sur les machines.

Pour simplifier les transmissions et les calculs, on attribue implicitement un numéro à chaque case de la zone de dessin.

- La case (0,0) sera la n^o 0
- La case (0, 1) sera la n^o 1
- ...
- La case (i, j) sera la n^o $(i \times largeur) + j$
- ...
- La case $(hauteur - 1, largeur - 1)$ sera la n^o $(hauteur \times largeur)$

3.1 Parallélisme à équilibrage de charge statique

L'équilibrage à charge statique consiste à ce qu'un processus maître recoive directement les pixels calculés par les processus esclave. Les esclaves calcule alors tous la même quantité d'informations.

Les esclaves (voir algorithme 2) peuvent determiner quels sont les pixels dont ils doivent faire le calcul, compte tenu du fait qu'ils connaissent les dimensions de l'image, le nombre de processus existant, leur rang dans l'environement et la taille de plage de pixels qu'ils doivent calculer en un coup.. Avec tous ces éléments, ils itèrent sur les cases dont ils doivent assurer le calcul. Pour chacune de ces cases, ils executent le calcul de la couleur qu'ils ajoutent dans le message. Une fois que la plage est entierement calculé, ils envoient le message au maître.

Algorithm 2 Calcul de l'ensemble de Mandelbrot en charge statique - esclave

Require: $largeur > 0$ and $hauteur > 0$ {Taille de l'image}
Require: $calculPixel(i, j)$ {Calcul la couleur d'affichage du pixel (i,j)}
Require: $size \geq 2$ {Nombre de processus}
Require: $rank > 0$ {Rang du processus courant}
Require: $root > 0$ {Rang du processus maître}
Require: $tailleplage > 0$ {Taille des plages de calcul}

```
case ← (rank − 1) × tailleplage
while case ≤ largeur × hauteur do
  for k = 0 to k < tailleplage do
    if case ≤ largeur × hauteur then
      i ← case ÷ hauteur
      j ← case mod hauteur
      couleur ← calculPixel(i, j)
      resultats(message)[k] ← couleur
      case ← case + 1
    end if
    envoyer(root, message)
  end for
end while
```

Quant au maître, il cherche à recevoir pour chaque case le traitement correspondant. Connaissant le nombre de processus dans l'environnement et la taille des plages de calculs, il peut déterminer quel processus lui envoie le résultat du calcul d'une série de pixels (voir algorithme 3). Dès qu'il reçoit un résultat d'un esclave, il affiche directement tous les points contenu dans le message dans l'image avec la couleur reçu pour chaque pixels.

Algorithm 3 Calcul de l'ensemble de Mandelbrot en charge statique - maître

Require: $largeur > 0$ and $hauteur > 0$ {Taille de l'image}
Require: $calculPixel(i, j)$ {Calcul la couleur d'affichage du pixel (i,j)}
Require: $size \geq 2$ {Nombre de processus}
Require: $tailleplage > 0$ {Taille des plages de calcul}

```
case ← 0
while case ≤ largeur × hauteur do
  emetteur ← ((case ÷ tailleplage) mod size − 1) + 1 {Determine le rang du processus emetteur}
  recevoir(emetteur, message)
  for k = 0 to k < tailleplage do
    if case ≤ largeur × hauteur then
      i ← case ÷ hauteur
      j ← case mod hauteur
      afficherPoint(j, i, resultats(message)[k])
      case ← case + 1
    end if
  end for
end while
```

3.2 Parallélisme à équilibrage de charge dynamique

L'équilibrage à charge dynamique consiste à ce qu'un esclave interroge à chaque fois le maître pour lui demander une liste des pixels à calculer. Les esclaves les plus rapide peuvent alors effectuer plus de traitement que les moins rapide, et l'ensemble du calcul de l'image est moins penalisé par les processus moins performants.

On voit sur l'algorithme 4 que l'esclave effectue un demande de traitement. Il recoit qu'il lui indique s'il y a un traitement à réaliser. Si oui, il recoit dans le message une plage de case pour lesquels il doit réaliser le calcul, représenté par les numéros de case de début et de fin. Pour chacune des case de la plage, il effectue le traitement et met le résultat dans la suite du message, qu'il envoi au maître. Ensuite il renvoi une demande de traitement, tant qu'il n'as pa reçu de message comem quoi il n'y a plus de traitement.

Algorithm 4 Calcul de l'ensemble de Mandelbrot en charge dynamique - esclave

Require: $largeur > 0$ and $hauteur > 0$ {Taille de l'image}
Require: $calculPixel(i, j)$ {Calcul la couleur d'affichage du pixel (i,j)}
Require: $size \geq 2$ {Nombre de processus}
Require: $rank > 0$ {Rang du processus courant}
Require: $root > 0$ {Rang du processus maître}

```
encore ← vrai
while encore do
  type(message) ← DEMANDE
  rang(message) ← rank
  envoyer(root, message) {Envoi un message de demande de traitement}
  recevoir(root, reponse) {Response du maître}
  if type(reponse) = TRAITEMENT then
    type(message) ← RESULTAT
    rang(message) ← rank
    casedebut(message) ← casedebut(reponse)
    casefin(message) ← casefin(reponse)
    for k = casedebut(reponse) to casefin(reponse) do
      i ← case ÷ hauteur
      j ← case  mod hauteur
      couleur ← calculPixel(i, j)
      id ← casefin(reponse) − casedebut(reponse) + k
      resultats(message)[id] ← couleur
    end for
    envoyer(root, message) {Envoi des résultats de traitement de la
    plage}
  else if type(reponse) = AUNCUN_TRAITEMENT then
    encore ← faux
  end if
end while
```

Pour ce qui est du maître (algorithme 5), au demarrage il attend un message d'un esclave quelconque. Le programme peut soit recevoir une demande de traitement ou soit des resultats. Dans le cas de la demande de traitement, on envoie une plage de case à calculer à l'esclave. S'il n'y a plus de case à calculer, on envoie un message un fin au processus. Dans le cas d'une réception de resultats, on récupère la liste des pixels et on affiche les affiche dans l'image. Ensuite, il se remet en attente d'un message d'un autre esclave, tant qu'il y a des calculs à effectuer, et qu'on a pas envoyé de message de fin à tous les processus.

Algorithm 5 Calcul de l'ensemble de Mandelbrot en charge dynamique - maître

Require: $largeur > 0$ and $hauteur > 0$ {Taille de l'image}
Require: $calculPixel(i, j)$ {Calcul la couleur d'affichage du pixel (i,j)}
Require: $size \geq 2$ {Nombre de processus}
Require: $rank > 0$ {Rang du processus courant}
Require: $root > 0$ {Rang du processus maître}
Require: $tailleplage > 0$ {Taille des plages de calcul}

```
nbactif ← size − 1
case ← 0
while nbactif ≠ 0 do
  recevoir(ANY_RANK, reponse) {Attente d'un message de n'importe qui}
  if type(reponse) = DEMANDE then
    if case ≤ largeur × hauteur then
      type(message) ← TRAITEMENT
      casedebut(message) ← case
      casefin(message) ← case + tailleplage − 1
      if casefin(message) > largeur × hauteur then
        casefin(message) ← largeur × hauteur
      end if
      case ← casefin(message) + 1
    else
      type(message) ← AUNCUN_TRAITEMENT
      nbactif ← nbactif − 1
    end if
    envoyer(rang(reponse), message) {Envoi de la response a l'esclave}
  else if type(reponse) = RESULTAT then
    for k = casedebut(reponse) to casefin(reponse) do {Affichage de la plage de pixels}
      i ← case ÷ hauteur
      j ← case  mod hauteur
      id ← casefin(reponse) − casedebut(reponse) + k
      couleur ← resultats(message)[id]
      afficherPoint(j, i, couleur)
    end for
  end if
end while
```

4 Performances

Les tests de performances ont été réalisé en lancant l'environnement MPI avec huits machines.

4.1 Test en fonction du nombre de processus

Nous voyons sur la figure 2 que le fait d'augmenter le nombre de processus reduit le temps de calcul sensiblement. En revanche ce temps diminue que lorsque le nombre de processus ne dépasse pas le nombre de machines. Par ailleurs, on voit que la charge dynamique est légèrement plus efficace que la charge statique. En revanche les deux sont bien meilleurs que le programme séquentiel.

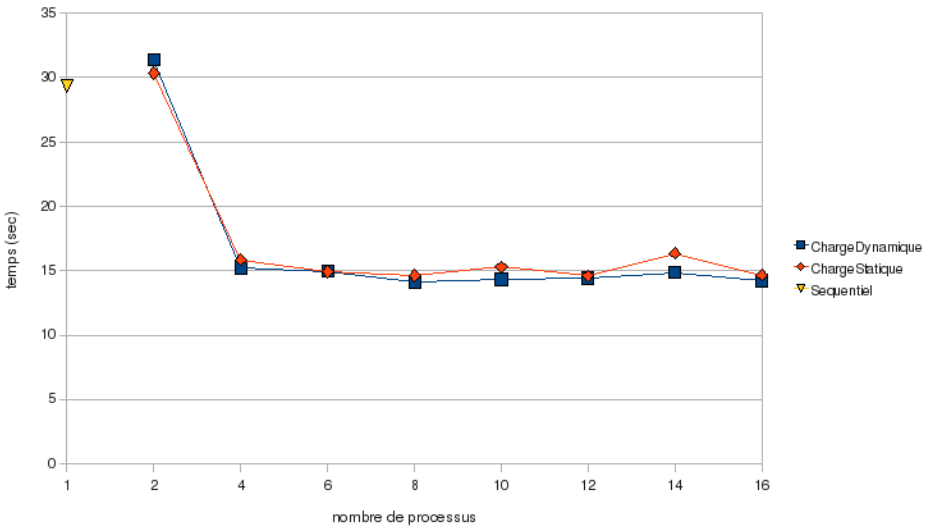

FIG. 2 – Test en fonction du nombre de processus esclave

4.2 Test en fonction de la taille des plages

Nous voyons sur la figure 3 que le fait d'augmenter la taille augmente le temps de calcul considérablement. Par ailleurs, on note toujours que la charge dynamique est légèrement plus efficace que la charge statique, et qu'il sont tous deux meilleurs que le programme séquentiel.

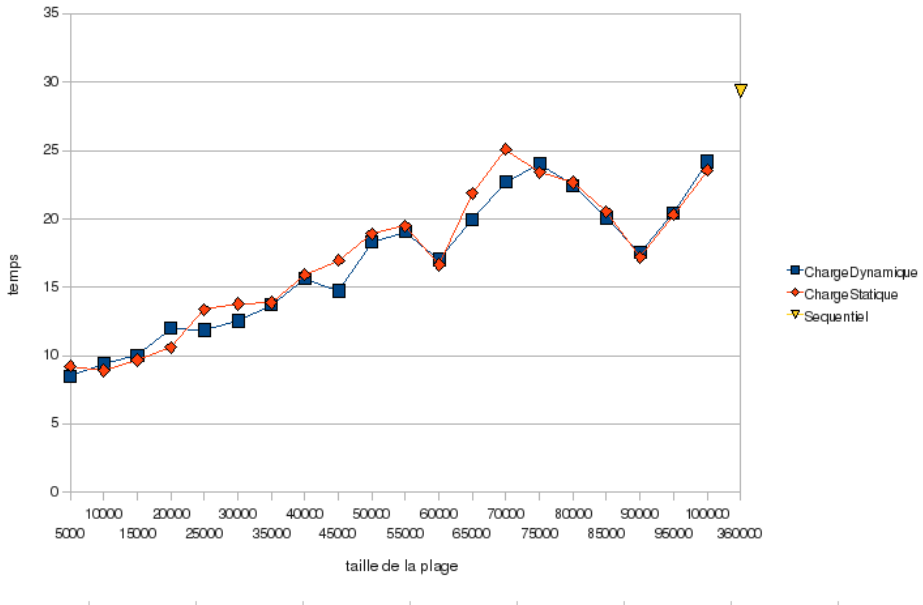

FIG. 3 – Test en fonction de la taille des plages

5 Conclusion

En conclusion, on pourra donc affirmer que la charge dynamique reste toujours la meilleure solution. De plus il faut limité le nombre de processus au nombre de machine que l'on utilise, et éviter de choisir une granularité trop grande.