UV F2B101

TP MPI – Image de Mandelbrot

Chloé Brouzes
Maxime Lasserre

Table des matières

Introduction

Ce TP a permis de calculer et effectuer le rendu de l'image de Mandelbrot. En analysant le programme et les résultats obtenus, nous avons pu mettre en évidence la performance d'une solution distribuée, et ce en 6 étapes définies ci-après.

Image de Mandelbrot

Cette première figure représente l'image de Mandelbrot avec une valeur maximale de x (MAXX) égale à 500. Notons que nous avons effectué ce calcul avec différentes valeurs de MAXX, l'unique différence au niveau du résultat étant la résolution de l'image.

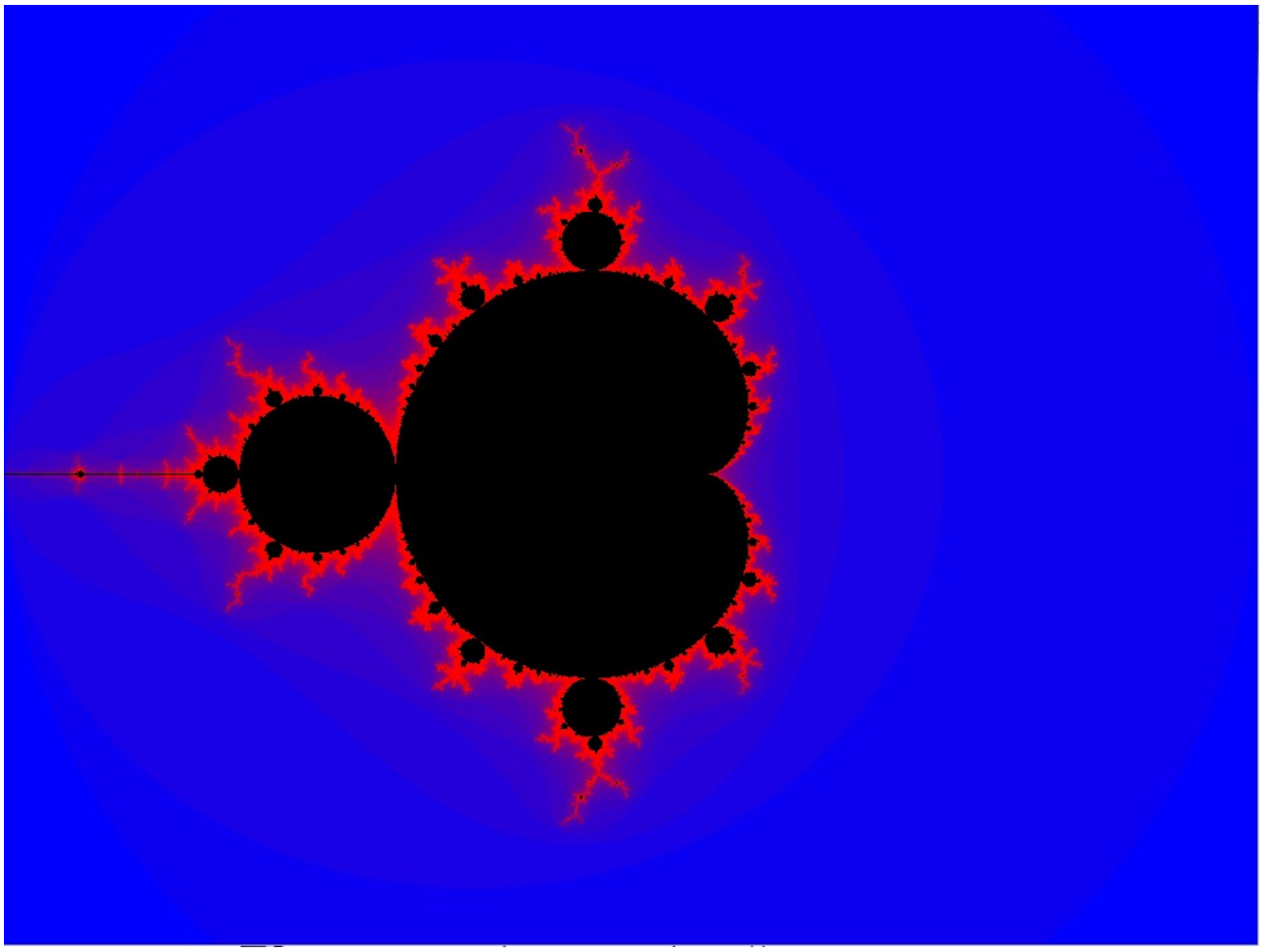


Figure 1 : Image de Mandelbrot MAXX=500

Mesures de temps de calcul

Des mesures de temps de calcul du programme ont été effectuées dans les différentes étapes. A noter que l'ensemble des calculs de ce rapport a été effectué avec une valeur de MAXX de 500.

Pour chaque valeur, le programme a été exécuté 10 fois afin de calculer la moyenne et vérifier l'écart-type des valeurs. Cela permet d'obtenir des résultats plus justes.

D'autre part, afin de limiter au maximum les perturbations et ainsi obtenir des résultats plus fiables, nous avons réalisé l'ensemble de ces mesures sur des machines non utilisées, ne comprenant pas le poste sur lequel nous travaillions. A noter qu'une nette diminution de l'écart-type de nos valeurs a été observée en arrêtant d'utiliser notre poste de travail.

Développement

L'analyse et l'amélioration de la performance du programme a été effectuée en 5 étapes, présentées ci-après.

Étape 1 : programme initial

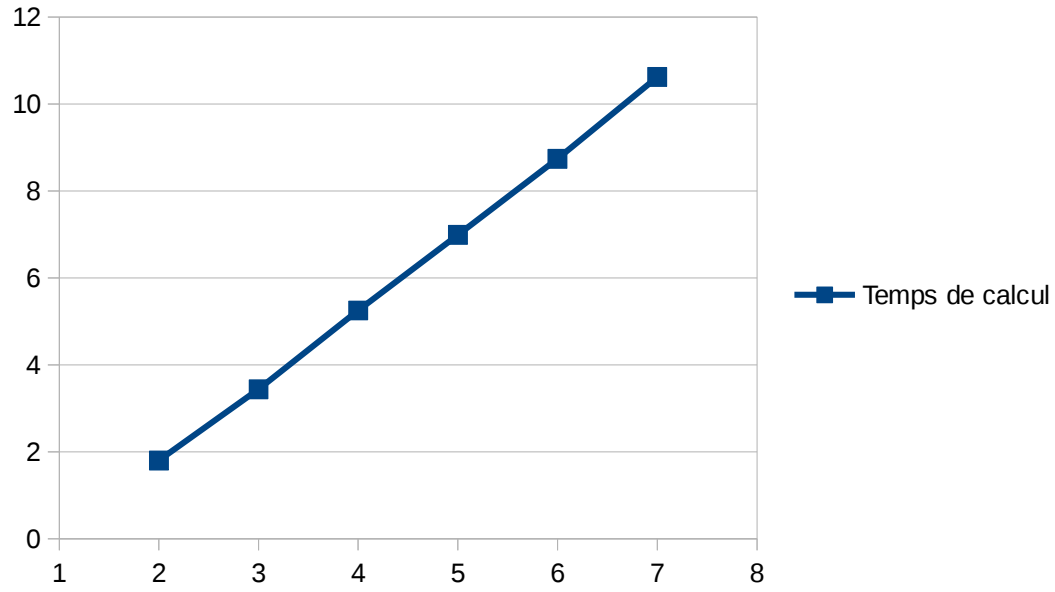
Cette première étape consiste à analyser et exécuter le programme fourni. Les temps de calcul ont été mesurés dans le code du maître entre la réception de la valeur de la première et la dernière case de l'image. Ils ne tiennent donc pas compte de la création de l'image (*dump*) qui n'a pas d'importance sur la performance du programme liée à la distribution du système. Pour cela, nous avons utilisé la commande MPI_Wtime comme suit :

```
if (rank == 0) {

 int res;

 // get the start time
 startwtime = MPI_Wtime();

 for(i = -MAXX; i <= MAXX; i++) {
  for(j = -MAXY; j <= MAXY; j++) {
   MPI_Recv(&res, 1, MPI_INT, 1, DATATAG, MPI_COMM_WORLD, &status);
   cases[i + MAXX][j + MAXY] = res;
  }
 }

 // get the end time
 endwtime = MPI_Wtime();
 printf("Temps ecoule = %f\n", endwtime-startwtime);

 dump_ppm("mandel.ppm", cases);
 printf("Fini.\n");
}
```

Le tableau suivant présente les temps de calcul du programme exécuté avec différents nombres de machines.

Nombre de machines	Temps de calcul	Écart-type
2	1,80 secondes	5e-04
3	3,44 secondes	4e-02
4	5,25 secondes	4e-02
5	6,99 secondes	5e-02
6	8,74 secondes	5e-02
7	10,62 secondes	3e-02

Les valeurs mesurées ont été représentées dans le graphe ci-dessous.



Comme le montrent les mesures et le graphe de cette première étape, l'ajout de machines augmente le temps de calcul du programme de façon linéaire. Une explication possible est que dans cette première version du programme, seul le premier esclave est écouté par le maître alors que l'ensemble des esclaves calculent l'image et envoient les résultats au maître. Ainsi, en ajoutant des machines, le nombre de messages échangés augmente, mais les esclaves ajoutés ne participent pas au résultat final de l'image. Le maître doit alors gérer un plus grand nombre de messages, inutiles au calcul de l'image.

D'autre part, l'image de Mandelbrot est, dans ce cas, calculé point par point. Autrement dit, l'esclave calcule et envoie la valeur de chaque point (x,y) un par un au maître. Cette procédure utilise de nombreux échanges maître/esclave, ce qui surcharge le réseau et ralenti le programme.

Deux améliorations sont alors possibles : permettre à l'ensemble des esclaves disponibles de participer au calcul de l'image, et calculer l'image groupe de points par groupe de points afin de réduire le nombre de messages envoyés.

Étape 2 : répartition statique et calcul point par point

Pour améliorer le programme précédent, nous avons corrigé le code afin de permettre au maître de recevoir des messages de n'importe quelle source. Ainsi, la machine maître peut recevoir des messages de tous les esclaves.

Dans cette étape, la distribution du travail, à savoir les points de l'image à calculer, est statique. L'image est découpée selon le nombre d'esclaves disponibles et chaque esclave possède un nombre fixe et défini de lignes à calculer. En exécutant le programme avec 4 machines, 3 esclaves seront disponibles et le calcul de l'image sera réparti de la façon suivante :

Esclave 1
Esclave 2
Esclave 3
Esclave 1
Esclave 2
Esclave 3

Figure 2 : Découpage statique du calcul de l'image avec 3 esclaves

Chaque esclave traite alors son bloc point par point en exécutant le code suivant :

```
double x, y;
int i, j, rc;
int res[3];

for (j = -MAXY; j <= MAXY; j++) {

 if (abs(j) % nbslaves != rank - 1) continue;

 for (i = -MAXX; i <= MAXX; i++) {
  x = 2 * i / (double)MAXX;
  y = 1.5 * j / (double)MAXY;

  res[0] = i;
  res[1] = j;
  res[2] = mandel(x, y);

  MPI_Send(&res, 3, MPI_INT, 0, DATATAG, MPI_COMM_WORLD);
 }
}
```

Le maître réceptionne les résultats, à savoir la valeur des différents points de l'image, et les stocke jusqu'à obtention de tous les points.

```
int res[3];
int count = 0;
```

```
while (count < NX * NY) {

 MPI_Recv(&res, 3, MPI_INT, MPI_ANY_SOURCE, DATATAG, MPI_COMM_WORLD,
&status);
 cases[res[0] + MAXX][res[1] + MAXY] = res[2];
 count++;
}
```

De même que dans la première étape, nous avons mesuré le temps de calcul de ce programme avec différents nombres de machines, en effectuant la moyenne de 10 mesures.

Nombre de machines	Temps de calcul	Écart-type
2	2,30 secondes	2e-06
3	2,30 secondes	3e-06
4	2,30 secondes	2e-05
5	2,30 secondes	6e-07
6	2,30 secondes	8e-05
7	2,30 secondes	9e-06
8	2,30 secondes	2e-05

D'après ces mesures, l'ajout de machines n'améliore pas la performance du programme : avec 1, 2 ou 6 esclaves le temps de calcul est le même. Cette limite est due au temps de traitement des messages par le maître. En effet, le calcul de l'image est effectué point par point. Ainsi, le maître reçoit les valeurs des points de l'image plus rapidement que dans l'étape 1 car plusieurs esclaves participent au calcul. Cependant, il doit traiter les messages un par un. Une queue se crée, et la rapidité de calcul liée à la multiplicité des esclaves est perdue du au temps de traitement des messages. Ce phénomène est visible sur le diagramme de type MSC suivant, réalisé avec 2 esclaves pour plus de lisibilité.

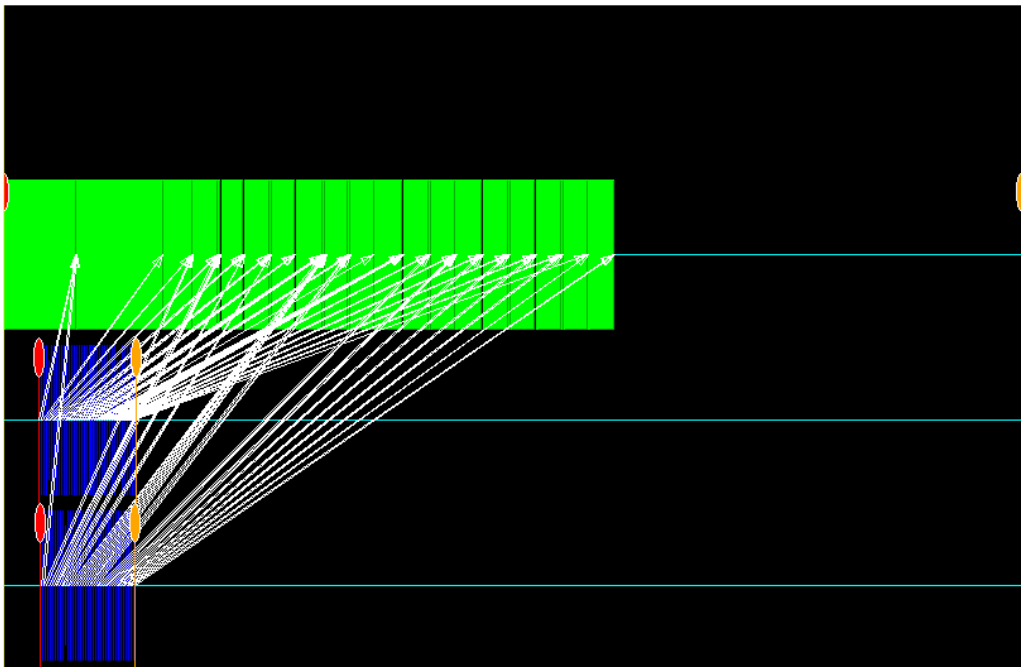


Figure 3 : Message Sequence Chart. Étape 2, 3 machines, MAXX=20

Étape 3 : répartition statique et calcul ligne par ligne

Afin de vérifier l'hypothèse faite à l'étape précédente, à savoir que le temps de traitement des messages par le maître limite la performance du programme, une troisième étape permet de calculer l'image de Mandelbrot ligne par ligne, tout en gardant une répartition du travail statique. Chaque esclave calcule donc les valeurs de l'ensemble des points d'une ligne avant d'envoyer le résultat au maître, ce qui diminue le nombre de messages échangés.

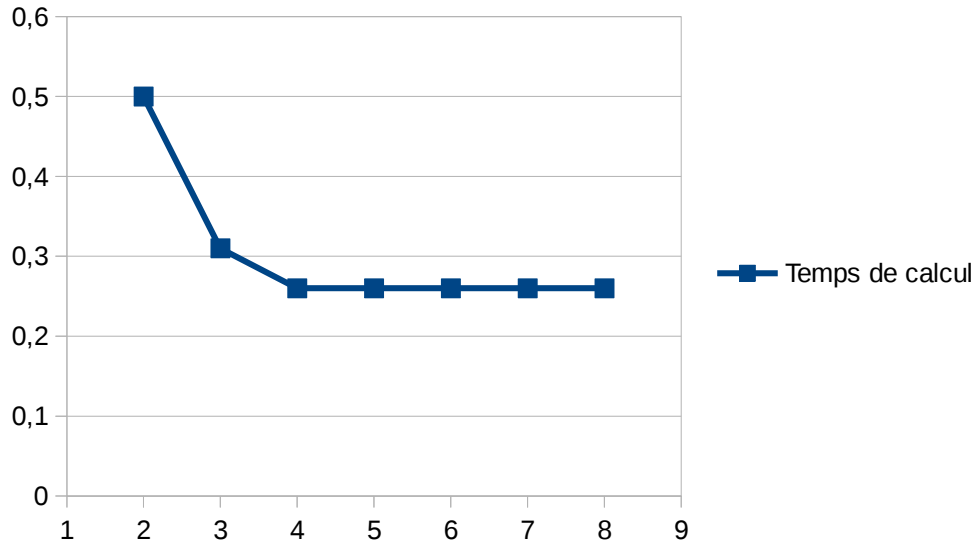
Le code esclave est donc transformé comme suit : chaque esclave calcule la valeur de l'ensemble des points d'une ligne, les stocke dans un tableau, puis envoie le tableau au maître.

```
for (j = -MAXY; j <= MAXY; j++) {

 if (abs(j) % nbslaves != rank - 1) continue;

 res[0] = j;
 for (i = -MAXX; i <= MAXX; i++) {

  x = 2 * i / (double)MAXX;
  y = 1.5 * j / (double)MAXY;
  res[i+1+MAXX] = mandel(x, y);
 }

 MPI_Send(&res, NX+1, MPI_INT, 0, DATATAG, MPI_COMM_WORLD);
 }
```

Les temps de calcul mesurés avec différents nombres de machines et en effectuant la moyenne de 10 mesures sont présentés dans le tableau suivant.

Nombre de machines	Temps de calcul	Écart-type
2	0,50 secondes	4e-04
3	0,31 secondes	2e-04
4	0,26 secondes	5e-05
5	0,26 secondes	2e-05
6	0,26 secondes	3e-03
7	0,26 secondes	2e-04
8	0,26 secondes	3e-02

Les valeurs mesurées ont été représentées dans le graphe ci-après.



Ces mesurent permettent de valider l'hypothèse de l'étape précédente.

Tout d'abord, à nombre de machines égal, le programme de calcul ligne par ligne est plus performant que celui point par point. Ceci s'explique par le fait que le nombre de messages est nettement diminué (divisé par le nombre de points par ligne, à savoir 2*MAXX+1=1001).

D'autre part, en ajoutant des machines, la performance du programme augmente car un plus grand nombre d'esclaves participe au calcul. Cependant, nous observons une stabilisation du temps de calcul à partir de 4 machines, soit 3 esclaves. Cela est probablement du au fait que le maître se retrouve surchargé par les messages entrants.

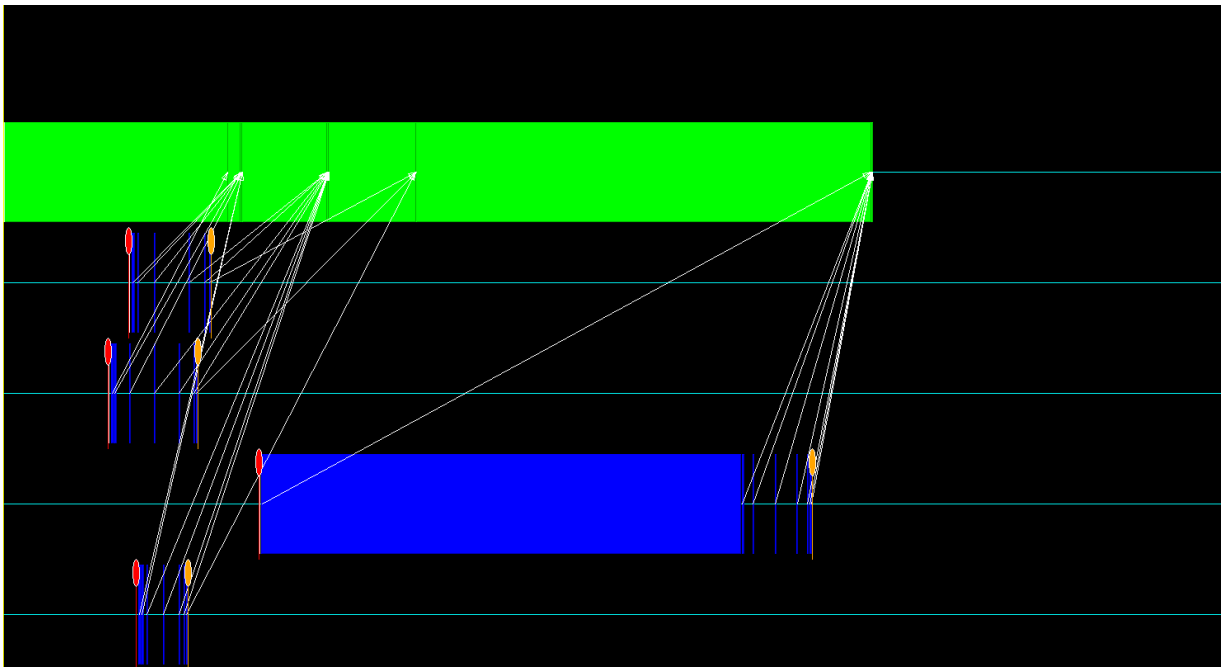


Figure 4 : Message Sequence Chart. Étape 3, 5 machines, MAXX=20

Ce diagramme MSC illustre le problème lié à la répartition de travail statique : si un esclave est plus lent que les autres, l'ensemble du calcul est ralenti. Une amélioration pourrait être de répartir les lignes à calculer de façon dynamique, c'est-à-dire en indiquant à chaque esclave la ligne suivante à traiter.

Étape 4 : répartition dynamique et calcul ligne par ligne

Cette quatrième étape résout le problème pointé précédemment : elle permet de passer d'une distribution du travail statique à une distribution dynamique. Comme précédemment, le calcul de l'image s'effectue ligne par ligne. L'amélioration consiste à indiquer aux esclaves la ligne suivante à traiter.

Pour cela, un tag *ORDERTAG* a été ajouté afin de repérer les messages indiquant à un esclave la ligne à calculer. Le code du maître est le suivant.

```
int res[NX+1];
int count = 0;
int i;

// initialisation - send first line to slaves
for (i = 1; i <= nbslaves; i++) {
 MPI_Send(&count, 1, MPI_INT, i, ORDERTAG, MPI_COMM_WORLD);
 count++;
}

while (count < NY) {

 // receive the result of a line computation
 MPI_Recv(&res, NX+1, MPI_INT, MPI_ANY_SOURCE, DATATAG,
MPI_COMM_WORLD, &status);
 slave = status.MPI_SOURCE;

 // store the result
 for (i = 0; i < NX; i++) {
  cases[i][res[0] + MAXY] = res[i+1];
 }

 // send a message with the next line to compute
 MPI_Send(&count, 1, MPI_INT, slave, ORDERTAG, MPI_COMM_WORLD);
 count++;
}
```

A la réception d'un message de type *ORDERTAG*, l'esclave récupère le numéro de ligne à traiter, calcule la valeur de l'ensemble des points de la ligne, et envoie le résultat au maître comme suit :

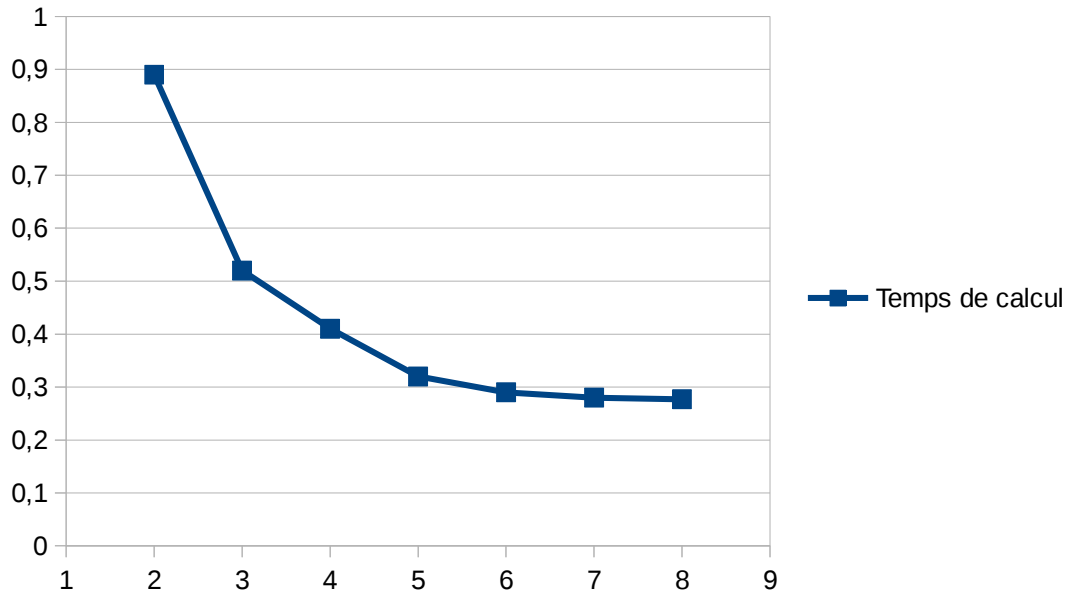
```
double x, y;
int i, j, rc;
int res[NY+1];

while (1) {
 // receive a message with the line to compute
 MPI_Recv(&j, 1, MPI_INT, 0, ORDERTAG, MPI_COMM_WORLD, &status);

 if (j >= NY) break;
```

```
 res[0] = j-MAXY;

// compute the image for the entire line
 for(i = -MAXX; i <= MAXX; i++) {
   x = 2 * i / (double)MAXX;
   y = 1.5 * (j-MAXY) / (double)MAXY;

   res[i+1+MAXX] = mandel(x, y);
 }

// send the result
 MPI_Send(&res, NX+1, MPI_INT, 0, DATATAG, MPI_COMM_WORLD);
}
```

Le temps de calcul du programme a été mesuré avec différents nombres de machines en prenant la moyenne sur 10 exécutions.

Nombre de machines	Temps de calcul	Écart-type
2	0,89 secondes	5e-05
3	0,52 secondes	2e-04
4	0,41 secondes	6e-04
5	0,32 secondes	1e-03
6	0,29 secondes	8e-04
7	0,28 secondes	4e-01
8	0,277 secondes	8e-01

Les valeurs mesurées ont été représentées dans le graphe ci-dessous.



En passant à une coordination dynamique des calculs, on constate que la vitesse d'exécution du programme est plus lente. Cela est du notamment au fait que l'on ajoute des communications

lorsque le maître coordonne le travail entre les esclaves. Le nombre de messages échangés est donc plus important et la durée d'exécution est augmentée.

Dans ce cas précis, cela semble être un inconvénient. Cependant dans un cas réel où une machine peut prendre plus de temps que les autres pour des raisons variées (machine avec une spécification plus faible, machine non dédiée au travail en cours, etc.) pouvoir allouer dynamiquement la charge vers d'autres esclaves plus véloces représente un avantage majeur.

Une nouvelle amélioration possible pour rendre cette version avec allocation dynamique plus rapide est alors d'augmenter le nombre de lignes calculées entre chaque envoie afin de diminuer le nombre de messages échangés. C'est l'objet de la partie suivante.

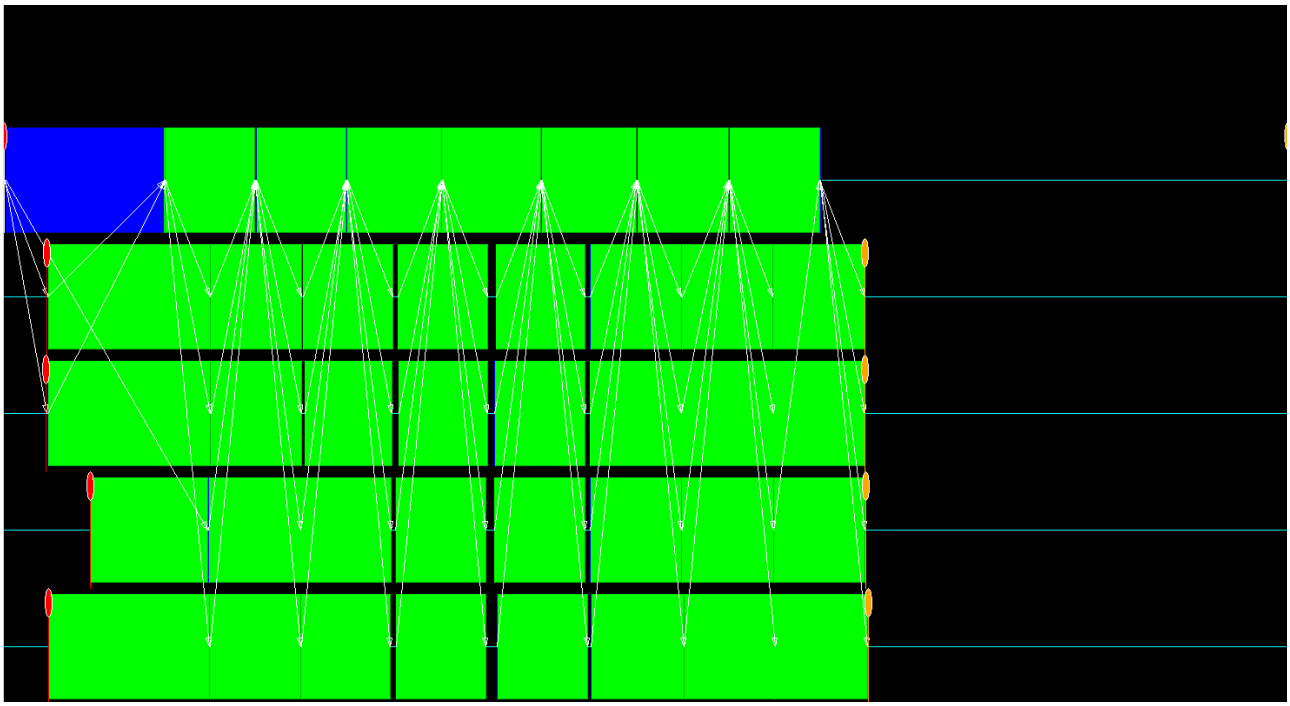


Figure 5 : Message Sequence Chart. Étape 4, 5 machines, MAXX=20

Ce diagramme illustre le conclusion précédente. On constate que le nombre de messages échangés est plus conséquent que dans l'étape 3, avec des communications plus nombreuses entre le maître et ses esclaves. Il est aussi intéressant de noter que les esclaves sont souvent en attente de réception d'un ordre de calcul.

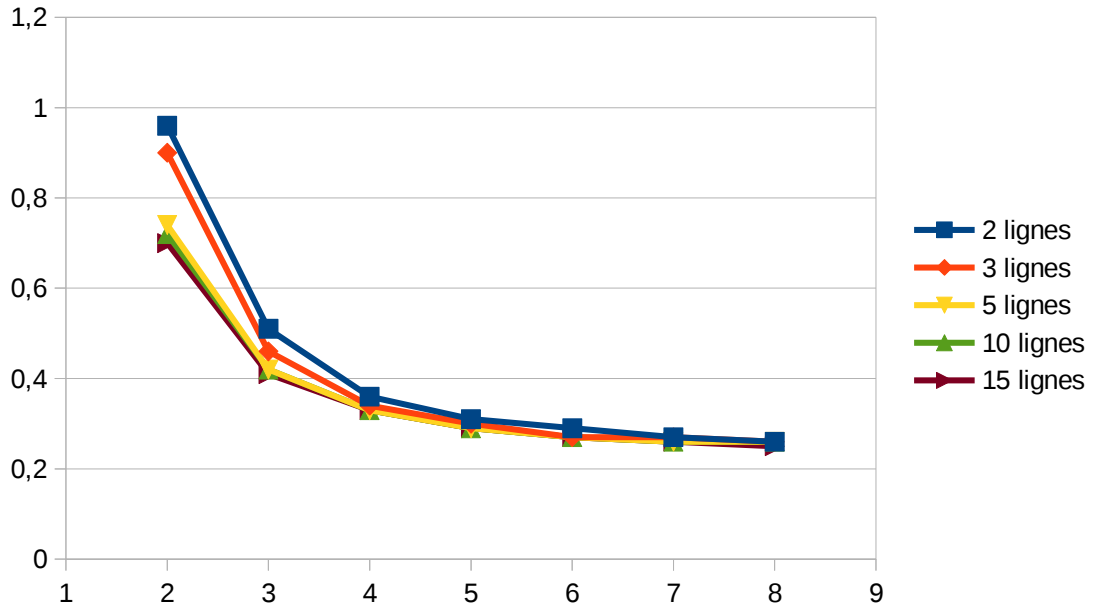
Étape 5 : répartition dynamique et calcul par groupe de lignes

La dernière étape de ce travail consiste à améliorer la distribution dynamique en découpant les calculs par bloc de lignes afin de diminuer le nombre de messages échangés.

Nous avons mesuré la performance du programme pour différents nombres de lignes : 2, 3, 5, 10 et 15. Chaque cas a été exécuté 10 fois afin de calculer la moyenne. Dans l'ensemble des cas, l'écart-type varie entre 1e-03 et 1e-05 et n'a donc pas été ajouté dans les tableaux suivants afin d'améliorer la visibilité. Les résultats sont indiqués en secondes.

Nombre de machines	2 lignes	3 lignes	5 lignes	10 lignes	15 lignes
2	0,96	0,90	0,74	0,72	0.70
3	0,51	0,46	0,42	0,42	0,41
4	0,36	0,34	0,33	0,33	0,33
5	0,31	0,30	0,29	0,29	0,29
6	0,29	0,27	0,27	0,27	0,27
7	0,27	0,27	0,26	0,26	0,26
8	0,26	0,26	0,26	0,26	0,25

Les valeurs mesurées ont été représentées dans le graphe ci-après.



Ce graphe valide l'hypothèse précédente : en groupant les calculs par ensemble de lignes, le temps d'exécution du programme est diminué. Ces résultats sont d'ailleurs cohérents avec les résultats de l'étape précédente, qui sont supérieurs.

D'autre part, à 8 machines, soit 7 esclaves, le temps de calcul est quasiment le même pour les différents groupes de lignes (2, 3, 5, 10 ou 15). Peut-être cela est-il, une fois de plus, due au temps de traitement des messages reçus par le maître. 0,26 secondes serait alors la durée minimale permettant de réaliser une image de Mandelbrot avec ce programme.

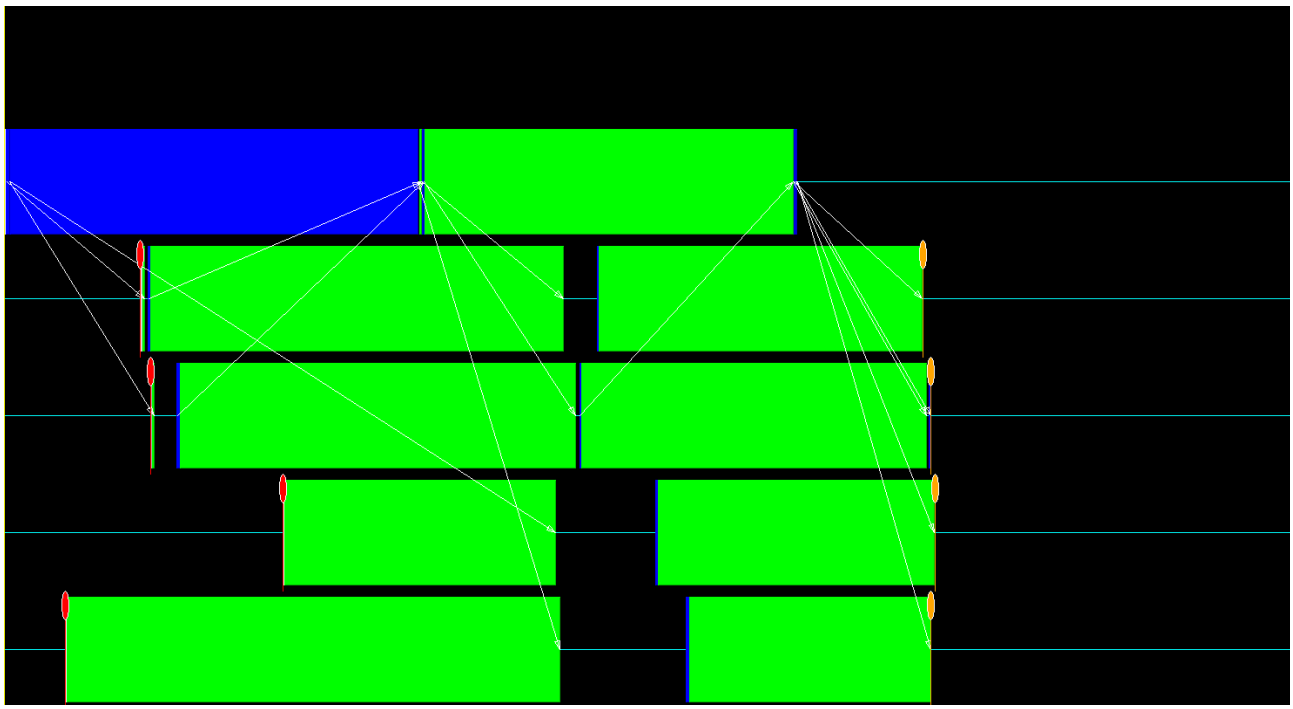


Figure 6 : Message Sequence Chart. Étape 5, 5 machines, groupes de 5 lignes, MAXX=20

Ce diagramme MSC, réalisé avec des groupes de 5 lignes, illustre la nette diminution du nombre de messages échangés, en comparaison avec l'étape précédente.

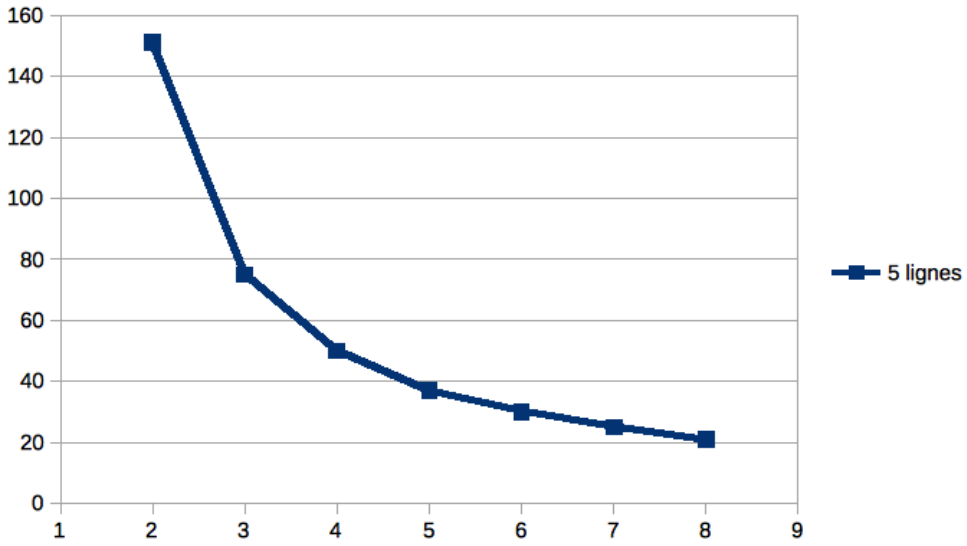
Étape 5 avec Sleep

Afin d'observer l'influence du temps de calcul, en comparaison au temps de traitement des messages, nous avons ajouté une temporisation du côté esclave. Après avoir calculé l'ensemble des valeurs des points d'un groupe de 5 lignes, les esclaves s'arrêtent une seconde avant d'envoyer les résultats au maître.

Les mesures effectuées sont présentées dans le tableau ci-dessous, en prenant la moyenne du temps de calcul sur 10 exécutions.

Nombre de machines	Temps de calcul
2	151 secondes
3	75 secondes
4	50 secondes
5	37 secondes
6	30 secondes
7	25 secondes
8	21 secondes

Les valeurs des mesures précédentes ont été représentées dans le graphe ci-dessous.



Ceci permet de mettre en évidence la part importante que peut prendre le calcul de la valeur des points dans le temps total d'exécution et de le rendre prépondérant devant le temps d'échange des messages. La courbe illustre alors le gain en temps disponible en effectuant le calcul de manière distribué.

Conclusion

Au cours de ce TP, nous avons analysé et essayé d'améliorer la performance du calcul de l'image de Mandelbrot.

Ce développement a été effectué en 6 étapes et nous a permis de mettre en évidence les deux sources du temps d'exécution :

- Le calcul de la valeur des points de l'image ;
- Et le traitement des messages échangés entre le maître et ses esclaves.

Nous avons alors pu identifier deux sources d'amélioration du temps d'exécution :

- Répartir le travail sur les différents esclaves de façon dynamique, en opposition à une répartition statique où chaque esclave se voit assigner un ensemble de points précis à traiter dès le début du programme. Pour une répartition dynamique, le maître doit garder en mémoire la dernière portion d'image demandée à un esclave afin de pouvoir indiquer au suivant les points à calculer.
- Calculer des portions d'images importantes afin de diminuer le nombre de messages échangés. Nous sommes alors passés d'un calcul point par point, avec envoie de message pour chaque point de l'image, à un calcul ligne par ligne, puis groupe de lignes par groupe de lignes. Une nette amélioration a été notée lors de ces changements.