Auteur: Pierre-Benjamin Monaco
Auteur: Lino Pascucci

MA-ParaDis Projet 1

N QUEENS PROBLEM
Rapport de projet

Pierre-Benjamin Monaco
Lino Pascucci

12 novembre 2022

Table des matières

Table des figures

Exemples de code

1 Introduction

Dans le cadre du cours MA-ParaDis, il nous a été demandé de résourdre le n-queens problem de manière distribuée

2 But

Le but de ce projet est de réussir une performance maximum pour le N le plus élevé avec une limite de temps d'exécution de 10 minutes.

3 Technologies utilisées

Pour atteidre notre but, les techonologies suivantes on été utilisée :

— C : Language de programmation.

— GMP : Librairie permettant de travailler avec des entiers de taille infinie.

— MPI : Libraire (framework) permettant la paralellisation d'un programme et l'échange de message entre les différentes instances de celui-ci.

4 Algorithme utilisé

L'algorithme utilisé est le suivant :

```
stack.push(empty board, row0, empty ignored_columns)
WHILE stack is not empty
    current_board, current_row, ignored_columns = stack.pop()
    IF current_row == N-1
        nbSolutions += 1
    ELSE
        LOOP current_col in not ignored columns
            new_queen = queenAt(current_row, current_col);
            IF new_queen is valid in current_board
                new_board = current_board + new_queen
                new_ignored_columns = ignored_columns + current_col
                stack.push(new_board, current_row + 1,new_ignored_columns)
            END IF
        END LOOP
    END IF
END WHILE
```

Code 1 – Algorithme utilisé pour résourde le N-Queens problem

Aucune optimisation n'a été faite pour éviter de calculer les symétries ou les rotations.

5 Structures de données utilisées

Les 3 piles utilisées (board, row, ignored columns) sont des tableaux de taille N^2 et un entier (stack_index) est utilisé pour placer les éléments. Le stack_index commence à 0 et augmente lors d'un pop et diminue lors d'un push.

Pour représenter un board, nous avons décidé d'utiliser des entiers (non signés) plutot que des tableaux.

```
1 board 4x4 :
 0 1 0 0
 0 0 0 1
 1 0 0 0
 0 0 1 0

en binaire    : 0b0100000110000010
en exadécimal : 0x4182
```

Code 2 – Exemple d'un tableau sous forme d'entier

Celà évite de devoir faire des boucles pour tester si la reine est valide mais te faire qu'une opération AND avec deux entiers.

```
Pour tester une reine en (3,4) dans un board 8x8 il faut :

1. Une reine et toutes ses attaques

 0 0 0 x 0 0 0 x
 x 0 0 x 0 0 x 0
 0 x 0 x 0 x 0 0
 0 0 x x x 0 0 0
 x x x Q x x x x
 0 0 x x x 0 0 0
 0 x 0 x 0 x 0 0
 x 0 0 x 0 0 x 0

cette reine vaut :
    0b0100100100101010000111001111111100011100001010100100100110001000
ou
    0x492A1CFF1C2A4988

2. Un tableau contenant des reines

 0 Q 0 0 0 0 0 0
 0 0 0 0 0 Q 0 0
 0 0 0 Q 0 0 0 0
 Q 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0

ce tableau vaut :
    0b0000000000000000000000000000000000000000100001000001000000000010
ou
    0x0000000000841002

3. Faire un AND binaire avec les deux tableaux

 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 Q 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0

    0x492A1CFF1C2A4988 & 0x0000000000841002 = 0x80000 != 0x0

si cet entier est différent de 0 alors la reine en (3,4) est invalide

```

Code 3 – Exemple de test de validité d'une reine

En C cela revient à faire :

```c
int isQueenValid(board, queen){
    return !(board & queen);
}
```

Code 4 – Exemple de fonction isQueenValid en C

6 Communication entre machines/processeurs

Il à été décidé que la communication entre machines serait faite selon le modèle **receiver initiated** car nous partions du principe que la charge allait être très haute pour toutes les machines à partir d'un N assez élevé. La première machine est lancée avec un board vide dans sa stack et les autres sont lancée avec une stack vide.

```
IF is first machine
    stack.push(empty board, row0, empty ignored_columns)
END IF

WHILE not terminated
    WHILE stack is not empty

        IF stack > 1
            Check for messages
            IF got message from X
                stack_data = stack.pop()
                Send ack = 1 to X
                Send stack_data to X
            END IF
        END IF

        current_board, current_row, ignored_columns = stack.pop()
        IF current_row == N-1
            nbSolutions += 1
        ELSE
            LOOP current_col in not ignored columns
                new_queen = queenAt(current_row, current_col);
                IF new_queen is valid in current_board
                    new_board = current_board + new_queen
                    new_ignored_columns = ignored_columns + current_col
                    stack.push(new_board, current_row + 1,new_ignored_columns)
                END IF
            END LOOP
        END IF
    END WHILE

    IF is first machine
        LOOP every messages
            Send ack = -1 to X
        END LOOP
    ELSE
        LOOP every machines not already contacted
            Send ask for stack
        END LOOP
        Check for messages
        IF got message from X
            IF message is ack = 1
                stack.push(data message)
            ELSE IF message is ack = -1
                terminate
            END IF
        END IF
    END IF

END WHILE
```

Code 5 – Algorithme utilisé pour résourde le N-Queens problem en parallèle

Les échanges entre machines peuvent être résumés comme ceci :

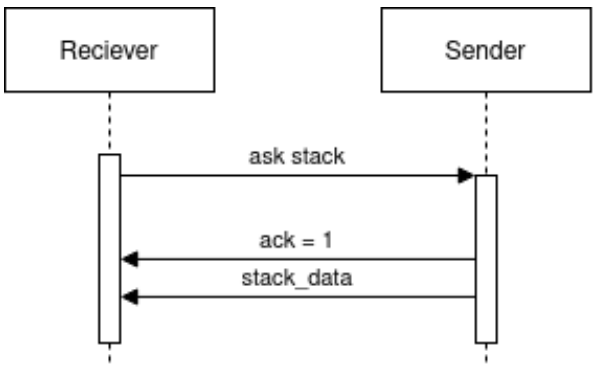

FIGURE 1 – Echange entre toutes les machines

Pour gérer la terminaison, les machines attendent sur la machine n°0. Si la machine 0 n'a plus de stack alors elle envoie un ack -1 à toutes les machines leur signifiant ainsi la fin du travail.

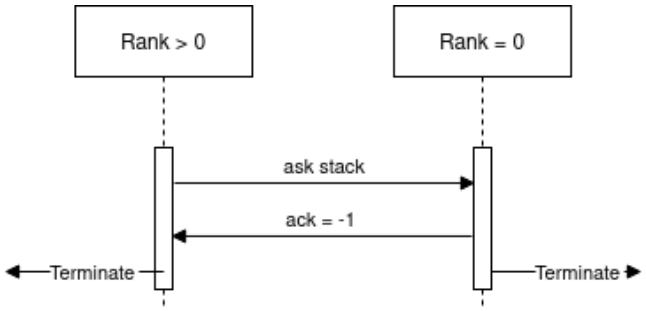

FIGURE 2 – Echange entre les machines et la machine n° 1

7 Performances

Voici un tableau et un graphique résumant les performance de notre code en séquentiel et en parallèle :

N	Sequential	Parallel (18p)	Sequential 2	Parallel 2 (18p)
1	0.00004	0.00890	0.00005	0.00920
2	0.00006	0.00730	0.00005	0.00320
3	0.00007	0.02320	0.00009	0.00730
4	0.00009	0.00760	0.00007	0.00560
5	0.00013	0.00400	0.00012	0.00470
6	0.00020	0.01070	0.00016	0.00400
7	0.00044	0.00480	0.00031	0.01150
8	0.00103	0.00710	0.00064	0.00950
9	0.00362	0.02370	0.00202	0.00560
10	0.01523	0.02100	0.00757	0.01270
11	0.07651	0.11090	0.03687	0.05170
12	0.40570	0.35810	0.19267	0.14740
13	2.37970	1.27270	1.10105	0.57610
14	14.65910	5.44470	6.59312	2.34380
15	94.85260	30.67670	41.39550	13.02170
16	657.69000	165.71350	285.53200	66.52760
17		857.07290		382.09950

FIGURE 3 – Tableau des performances

Le deux premières colonnes du tableau représentent les performances présentées pendant le cours. Après avoir comparé avec nos collègues, nous nous somme rendu compte qu'elle étaient plutot médiocre et nous nous sommes sentis obligé d'aller optimiser le code une dernière fois. Les deux dernières colonnes du tableau représentent les performances après les dernières modifications.

En relisant le code, il se trouve que nous avions laissé 4 mallocs dans du code utilisé en permanence. En faisant nos allocations au début et en libérant la mémoire à la fin, nous avons pu améliorer les performances du double en séquentiel

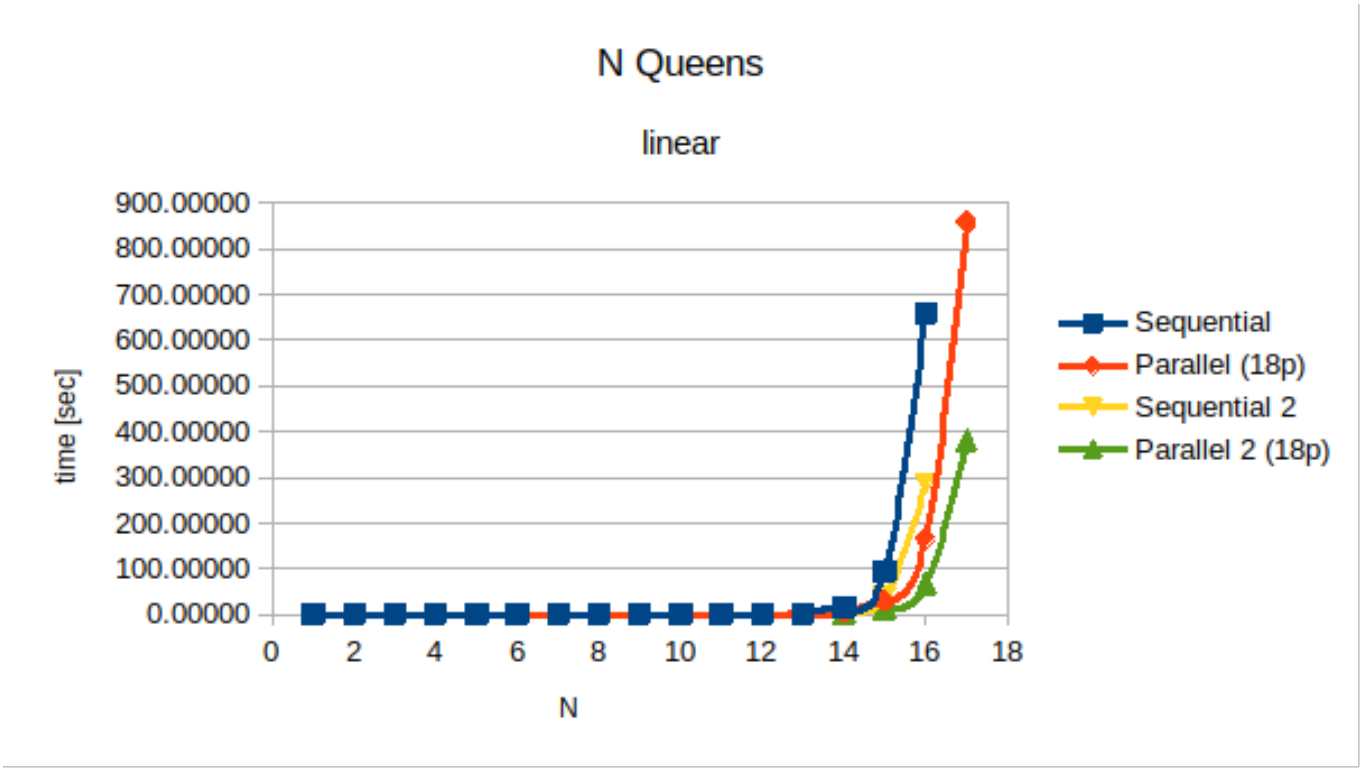

FIGURE 4 – Graphique des performances (linéaire)

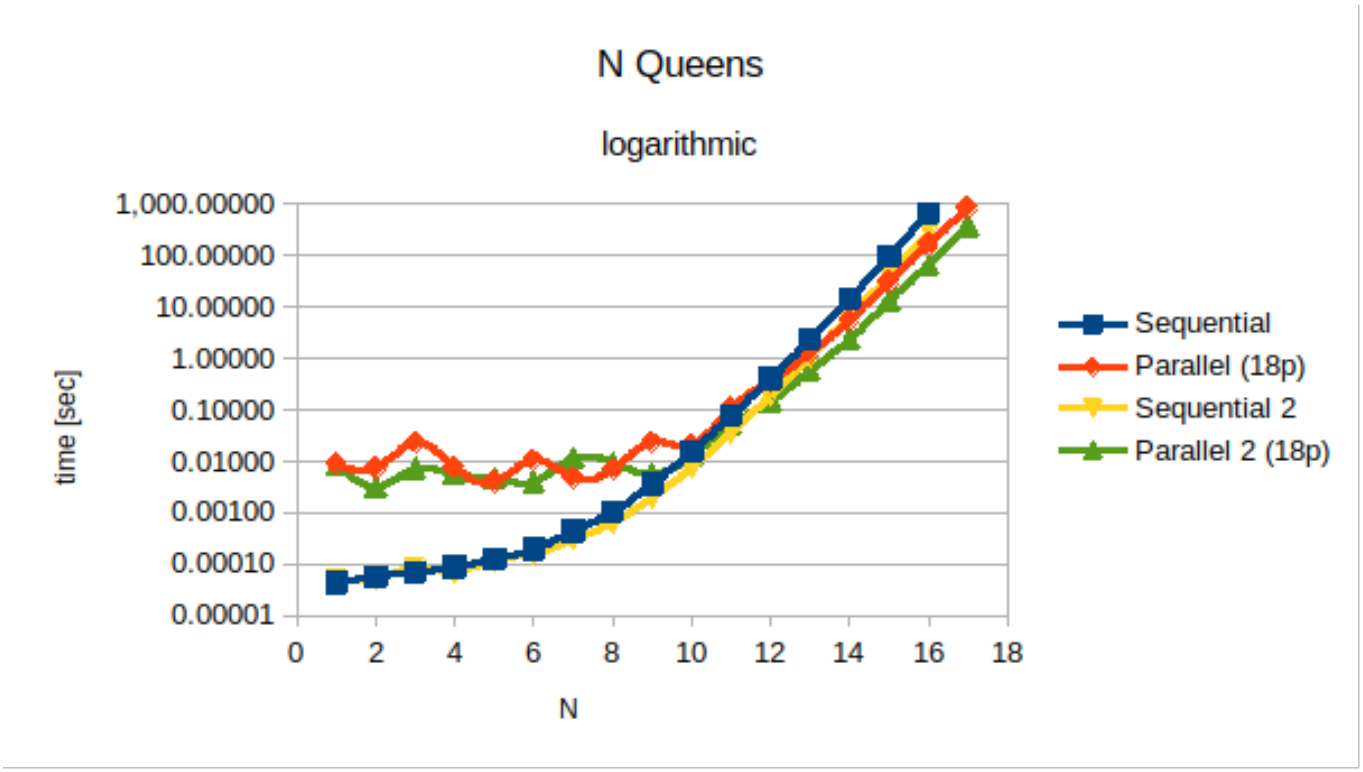

FIGURE 5 – Graphique des performances (logarithmique)

Il est aussi possible d'analyser le temps d'oisiveté moyen de chaque machine

Idle time and transaction per N

N	Idle time ratio	Transactions
1	99.50%	0
2	98.83%	0
3	98.83%	0
4	98.38%	0
5	97.48%	0
6	94.63%	0
7	96.24%	0
8	91.87%	0
9	90.53%	0
10	90.08%	0
11	93.90%	0
12	91.57%	4
13	87.95%	51
14	84.81%	424
15	79.20%	3476
16	67.69%	21965
17	66.85%	219907

FIGURE 6 – Tableau d'oisiveté

Le tableau ci-dessus montre que les machines sont sous-utilisées. Ceci est normal pour des petites valeurs de N mais si le code avait été bien fait, le taux d'oisiveté devrait être bien plus bas que celui qu'on observe ici.

Afin d'éviter de donner des trop petits travaux aux machines, la taille de l'arbre à envoyer à été réduite. Le sender ne partage sa stack que si l'arbre à une certaine taille. Le tableau suivant montre l'évolution de l'oisiveté en fonction de la taille minimale du tableau partagé.

Minimum tree size for transaction for N = 17

Tree size	Idle time	Transactions	Execution time
8	75.89%	3442792	685
9	65.16%	899513	391
10	66.85%	219907	390
11	68.29%	35638	407
12	73.61%	5162	488

FIGURE 7 – Tableau d'oisiveté en fonction de la taille minimum de l'arbre envoyé

8 Conclusion

Optimiser avec du bitwise c'est bien mais il faut savoir ce qu'on fait. Peut être que GMP est pas si puissant que ça ou alors a été mal utilisé ici.

Il faut Toujours faire très attention à la gestion de mémoire dynamique. Les mallocs prennent beacoup de temps et si ils sont fait dans la boucle principale alors on ajoute ce temps pour chaque calcul (voir Code 6 en annexe).

L'échange entre les machines est mal implémenté dans notre cas. Pour éviter de donner un travail trop petit, le sender n'envoie que des tableaux d'une certaine taille mais il doit attendre de résoudre la stack jusqu'a un certain niveau pour envoyer. Pendant ce temps tout le monde attend ce qui explique le taux d'oisiveté si haut.
Même en essayant de fine-tuner la taille minimum du tableau à partager, les performances n'augmentent pas (voir Fig 7). On se rend bien compte qu'il aurait mieux fallu penser les échanges et peut-être opter pour une aproche sender initiated, ce qui aurait ralenti un peu la machine avec le plus de taches mais aurait permi de partager plus rapidement et de manière plus optimale les calculs.
Une solution serait d'utiliser une stack qui peut être poppée depuis dessous car c'est tout en bas de la stack que l'on retrouve les jobs avec beaucoup de travail (le début de l'arbre). Le problème est qu'il faut aussi popper tous les niveaux dépendants de celui qu'on a popé pour ne pas faire les même calculs plusieurs fois ou alors s'assurer que l'arbre que l'on pop n'est pas encore utilisé.

On pourrait se demander si la solution n'était pas de découper de manière statique comme celà à été fait par d'autre groupes. En tout cas, dans notre cas cette solution aurait surement été plus performante. La solution d'avoir un master résoud les X premiers niveau d'un tableau et qui les passe aux autre aurait surement été plus optimale également.

Les echanges de messages entre processeurs ne prennent quasiment pas de temps. Quand on test MPI sur une machine mais avec plusieurs processeurs, les transactions sont instantanées. Pas quand il y a plusieurs machines et ceci nous a joué des tours quand nous avons testé le code pour la première fois sur le serveur. Peut-être qu'avec cette information nous nous serions tournés vers un autre algorithme d'échange de données entre machines.

Le fait d'avoir utilisé les fonctions asynchrones de MPI permettent de laisser travailler les machines plutot que de les faire attendre des messages. C'était intéressant de les découvrir et d'apprendre à les utiliser.
Au final, pleins de bonnes idées ont émergés de ce projet et il est clair que si l'on devait refaire un problème de parallelisation aujourd'hui nous ne le résoudrions pas de la même manière qu'ici. La question de la parallelisation resterai centrale et l'optimisation du code serait faite après avoir un pensé puis implémenté un partage optimal.

9 Annexes

9.1 Dernières modifications effectuées

```
diff --git a/n_queens_gmp.c b/n_queens_gmp.c
index bb356ba..ae5a5ca 100644
--- a/n_queens_gmp.c
+++ b/n_queens_gmp.c
@@ -29,6 +29,9 @@ mpz_t rows[NB_QUEENS];
 // merged columns, rows and diagonals in a single int
 mpz_t shots[NB_QUEENS][NB_QUEENS];

+mpz_t queen_manipulator;
+char serialized_board[MPI_SERIALIZED_BOARD_LENGTH];
+
 void makeColumn(mpz_t column, short startPos)
 {
     // set column to 0
@@ -388,31 +391,20 @@ short isQueenInDiagonal2(mpz_t board, int row, int column)

 short isQueenValid(mpz_t board, int row, int column)
 {
-    // init and set checker to 0
-    mpz_t checker;
-    mpz_init(checker);
-    mpz_set_ui(checker, 0);
-
     // compare line and board
-    mpz_and(checker, board, shots[row][column]);
-    short returnValue = mpz_cmp_ui(checker, 0);
+    mpz_and(queen_manipulator, board, shots[row][column]);
+    short returnValue = mpz_cmp_ui(queen_manipulator, 0);

-    // clear checker
-    mpz_clear(checker);
     return !returnValue;
 }

 void addQueenAt(mpz_t board, int row, int column)
 {
-    mpz_t new_queen;
-    mpz_init(new_queen);
-    mpz_set_ui(new_queen, 1);
+    mpz_set_ui(queen_manipulator, 1);

     // left shift queen to position
-    mpz_mul_2exp(new_queen, new_queen, (column + row * NB_QUEENS));
-    mpz_ior(board, board, new_queen);
-
-    mpz_clear(new_queen);
+    mpz_mul_2exp(queen_manipulator, queen_manipulator, (column + row * NB_QUEENS));
+    mpz_ior(board, board, queen_manipulator);
 }

 long long unsigned nb_solutions = 0;
@@ -478,7 +470,6 @@ void mpi_sender_routine(int *index_stack, mpz_t *boards_stack, int *rows_stack,
             int ack = 1;
             MPI_Send(&ack, 1, MPI_INT, rcv_rank, MPI_ACK_TAG, MPI_COMM_WORLD);

-            char *serialized_board = malloc(MPI_SERIALIZED_BOARD_LENGTH);
             mpz_get_str(serialized_board, MPI_GMP_N_BIT_SERIALIZING, boards_stack[(*index_stack)]);
 #if MPI_SHOW_DEBUG > 0
             printf("Proc %d send board %s\n", myrank, serialized_board);
@@ -493,7 +484,6 @@ void mpi_sender_routine(int *index_stack, mpz_t *boards_stack, int *rows_stack,
 #endif
             MPI_Send(&used_cols_stack[(*index_stack)], 1, MPI_UNSIGNED_LONG_LONG, rcv_rank, MPI_USEDCOL_TAG, MPI_COMM_WORLD);
             --(*index_stack);
-            free(serialized_board);
         }
     }
 }
@@ -512,7 +502,6 @@ void mpi_receiver_initiated_routine(int *index_stack, mpz_t *boards_stack, int *
     {
         (*index_stack) = 0;
-        char *serialized_board = malloc(MPI_SERIALIZED_BOARD_LENGTH);
         MPI_Recv(serialized_board, MPI_SERIALIZED_BOARD_LENGTH, MPI_BYTE, MPI_ANY_SOURCE, MPI_BOARD_TAG, MPI_COMM_WORLD, NULL);
 #if MPI_SHOW_DEBUG > 0
         printf("Proc %d recieved board : %s\n", myrank, serialized_board);
@@ -536,8 +525,6 @@ void mpi_receiver_initiated_routine(int *index_stack, mpz_t *boards_stack, int *
 #endif
         used_cols_stack[(*index_stack)] = used_cols;

-        free(serialized_board);
-
```

```
 #if MPI_STATS > 0
         num_transactions++;
 #endif
@@ -715,7 +702,7 @@ void checkAllQueensIt()

 int main()
 {
-
+    mpz_init(queen_manipulator);
 #if USE_MPI > 0
     double begin_build;
     double build_time;
@@ -774,6 +761,7 @@ int main()

     checkAllQueensIt();
     clearMasks();
+    mpz_clear(queen_manipulator);

 #if USE_MPI > 0
     total_time = MPI_Wtime() - begin_build;
```

Code 6 – Modifications effectuées pour améliorer les performances après la présentation