UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Rapport projet
Technique d'Optimisation et Parallélisation.
M1-CHPS

Nom:Lougani
Prénom:Faouzi
Numéro étudiant:**22003152**

Compilation et exécution du code :

*Il y a pas d'erreur de compilation,par contre l'exécution a échoué dans un premier temps car le nombre max de processus supporté par mon Ubuntu est dépassé (**mpirun -np 512 ./lbm**) .*

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$ mpirun -np 512 ./lbm
[proxy:0:0@louganifaouzi-ThinkPad-L520] HYDU_create_process (utils/launch/launch.c:2
2): pipe error (Too many open files)
[proxy:0:0@louganifaouzi-ThinkPad-L520] launch_procs (pm/pmiserv/pmip_cb.c:702): cre
ate process returned error
[proxy:0:0@louganifaouzi-ThinkPad-L520] HYD_pmcd_pmip_control_cmd_cb (pm/pmiserv/pmi
p_cb.c:885): launch_procs returned error
[proxy:0:0@louganifaouzi-ThinkPad-L520] HYDT_dmxu_poll_wait_for_event (tools/demux/d
emux_poll.c:77): callback returned error status
[proxy:0:0@louganifaouzi-ThinkPad-L520] main (pm/pmiserv/pmip.c:200): demux engine e
rror waiting for event
```

*On diminue le nombre de processus à **4** au lieu **de 512** par exemple et on aura ainsi une erreur de segmentation(**Segmentation fault).** Le **(signal 11)** veut dire que que le programme a accédé à un emplacement mémoire qui n'a pas été attribué.*

```
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM 1 CORNER -1, -1, -1, -1 ) ( POSITION 0
0 ) (WH 802 42 )

=================================================================================
===
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 15219 RUNNING AT louganifaouzi-ThinkPad-L520
=   EXIT CODE: 139
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
=================================================================================
===
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Segmentation fault (signal 11
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

Débogage du programme :

*Afin de déterminer l'origine de l'erreur et avoir plus de détails, un outil de débogage est indispensable ,j'ai opté pour **GDB**(GNU Project Debugger).*

*L'exécution du programme avec le débogueur ouvre un nouvel invite de commande, (gdb), on exécute le programme sur la console avec **Run**:*

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$ gdb ./lbm
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./lbm...
(gdb)
```

On voit plus de détails sur l'erreur :

```
Program received signal SIGSEGV, Segmentation fault.
0x0000555555556b45 in setup_init_state_global_poiseuille_profile (
    mesh=0x7fffffffde20, mesh_type=0x7fffffffde50, mesh_comm=0x7ff
    at lbm_init.c:85
85                                      Mesh_get_cell(mesh, i, j)[
equilibrium_profile(v,density,k);
(gdb)
```

*le bug vient de la fonction **Mesh_get_cell()** à la ligne **85** du fichier **lbm_init.c**.*
Donc la ligne de l'erreur est :

Mesh_get_cell(mesh, i, j)[k] = compute_equilibrium_profile(v,density,k);

*L'erreur effective viens de la structure **mesh** ,on fait un **print** afin de voir les adresses :*

```
(gdb) print mesh
$1 = (Mesh *) 0x7fffffffde20
(gdb) print (*mesh)
$2 = {cells = 0x0, width = 802, height = 162}
(gdb)
```

*L'une des cellules du maillage **n'est pas allouée** ,c'est a dire l'adresse **0x0**, une fois la fonction **Mesh_get_cell()** accède à cet emplacement un bug aura lieu.*

Si on fait un list de la fonction on deduit que la structure n'est pas allouée dans la fonction donc on cherche dans le fichier.c associé à la définition de la structure.

```
db) list Mesh_get_cell
0     /*******************  FUNCTION  *********************/
1     /**
2      * Fonction à utiliser pour récupérer une cellule du m
3     **/
4     static inline lbm_mesh_cell_t Mesh_get_cell( const Mes
5     {
6             return &mesh->cells[ (x * mesh->height + y) *
7     }
8
9     /*******************  FUNCTION  *********************/
db)
```

*Donc on ouvre le fichier **lbm_struct.c** ,l'erreur est clair dans la fonction Mesh_init()*

```
18        //alloc cells memory
19        //mesh->cells = malloc( width * height  * DIRECTIONS * sizeof( double ) );
20        mesh->cells = NULL;
```

Pour corriger cette erreur il suffit de :

```
18        //alloc cells memory
19        mesh->cells = NULL;
20        mesh->cells = malloc( width * height  * DIRECTIONS * sizeof( double ) );
```

*Afin d'exécuter le programme ,j'ai choisi **3 processus** par contre pour réduire le temps d'exécution ,j'ai réduit le nombre d'itération dans le fichier config.txt de 16000 à **5.***

config.txt

```
1 iterations          = 5
2 width               = 800
3 height              = 160
4 #obstacle_r          =
5 #obstacle_x          =
6 #obstacle_y          =
7 reynolds            = 100
8 inflow_max_velocity = 0.100000
9 inflow_max_velocity = 0.100000
10 output_filename     = resultat.raw
11 write_interval      = 50
```

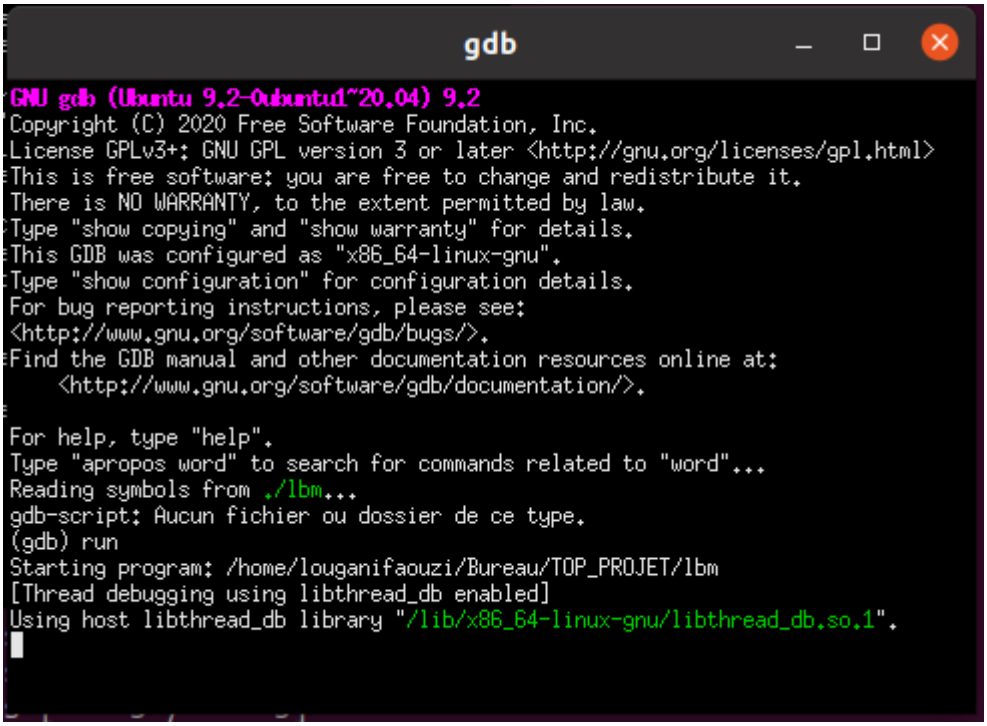
```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$ mpirun -np 3 ./lbm
 RANK 1 ( LEFT 0 RIGHT 2 TOP -1 BOTTOM -1 CORNER -1, -1, -1, -1 ) ( POSITION 266 0
 ) (WH 268 162 )
================== CONFIG ==================
iterations           = 5
width                = 800
height               = 160
obstacle_r           = 17.000000
obstacle_x           = 161.000000
obstacle_y           = 83.000000
reynolds             = 100.000000
reynolds             = 100.000000
inflow_max_velocity  = 0.100000
output_filename      = resultat.raw
write_interval       = 50
------------ Derived parameters -------------
kinetic_viscosity    = 0.034000
relax_parameter      = 1.661130
============================================
 RANK 0 ( LEFT -1 RIGHT 1 TOP -1 BOTTOM -1 CORNER -1, -1, -1, -1 ) ( POSITION 0 0
) (WH 268 162 )
 RANK 2 ( LEFT 1 RIGHT -1 TOP -1 BOTTOM -1 CORNER -1, -1, -1, -1 ) ( POSITION 533
0 ) (WH 268 162 )
Progress [    1 /     5]
Progress [    2 /     5]
Progress [    3 /     5]
Progress [    4 /     5]
^Z
[2]+  Arrêté                 mpirun -np 3 ./lbm
```

L'image à droite montre que programme ne veut pas terminer l'exécution, ce bug est nommé ***interblocage*** *(ou* ***deadlock*** *en anglais)*
Un interblocage se produit lorsque des processus concurrents s'attendent mutuellement .

Afin d'avoir plus d'information sur ce bug .On execute dans un terminal la commande suivante : ***mpirun -np 3 xterm -e gdb -command=gdb-script ./lbm***

Cette commande permet de de lancer le programme avec 3 processus ,chacun dans terminal xtern en lançant son propre gdb .

Une fois on a exécuter la commande on fait un ***Run*** *dans chaque terminal gdb on aura le résultat suivant :*



*avec la commande **backtrace** toujours on identifie la trace de l erreur.*

*L'interblocage est causé par la ligne **202** du fichier main.c.*

```
200         if( rank == RANK_MASTER && fp != NULL)
201         {
202                 close_file(fp);
203         }
```

*L'interblocage est causé par la fonction qui ferme le fichier **close_file()** si rank est égale à **RANK_MASTER**.*
*Si on analyse le code de la fonction **close_file() :***

```
void close_file(FILE* fp){
        //wait all before closing
        MPI_Barrier(MPI_COMM_WORLD);
        //close file
        fclose(fp);
}
```

*On trouve qu'elle a une barrière,donc ici **fclose(fp)** doit attendre tous les processus pour fermer le fichier **fp** et non pas juste celui du rang **RANK_MASTER .***

Afin de résoudre le problème on doit attendre tous les processus donc on modifie dans main.c:

```
199         //CORRECTION ICI ON attend RANK_MASTER +Les autres processus
200         if(fp != NULL)
201         {       if(rank)
202                 close_file(fp);
203         }
```

maintenant notre programme finie son exécution correctement :

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$ ./lbm
================== CONFIG ==================
iterations           = 5
width                = 800
height               = 160
obstacle_r           = 17.000000
obstacle_x           = 161.000000
obstacle_y           = 83.000000
reynolds             = 100.000000
reynolds             = 100.000000
inflow_max_velocity  = 0.100000
output_filename      = resultat.raw
write_interval       = 50
------------ Derived parameters -------------
kinetic_viscosity    = 0.034000
relax_parameter      = 1.661130
============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 CORNER -1, -1, -1, -1 ) (
 0 ) (WH 802 162 )
Progress [    1 /     5]
Progress [    2 /     5]
Progress [    3 /     5]
Progress [    4 /     5]
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

Scalabilité du code :

*Afin de déterminer si notre code possède une bonne **Scalabilité** ou non on va :*

- *Augmenter les données et la taille totale du problème.*
- *Augmenter les données pour une taille de problème fixe.*

Dans les 2 cas on observe le temps de calcul émit .

***Cas 1 :**Augmenter les données et la taille totale du problème.*

*Avant de commencer on doit avoir le temps de calcul dans l'état normal dans un premier temps j'ai essayé avec **rdtsc** qui récupère le temps en cycles du coups c'est difficile de voir la différence entre les valeurs,j'ai opter pour la fonction **clock** qui a une précision supérieure à une douzaine de microsecondes avec le code suivant :*

```
#include<time.h>
clock_t begin = clock();
{
/* notre code  */
}
clock_t end = clock();
double time_spent = (double)(end – begin)/CLOCKS_PER_SEC;
```

On aura le résultat suivant :

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$ mpirun -np 4
./lbm
================== CONFIG ==================
iterations            = 5
width                 = 800
height                = 160
obstacle_r            = 17.000000
obstacle_x            = 161.000000
obstacle_y            = 83.000000
reynolds              = 100.000000
reynolds              = 100.000000
inflow_max_velocity   = 0.100000
output_filename       = resultat.raw
write_interval        = 50
------------ Derived parameters --------------
kinetic_viscosity     = 0.034000
relax_parameter       = 1.661130
============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM 1 CORNER -1, -1, -1, -1 ) ( POSITIO
N 0 0 ) (WH 802 42 )
 RANK 1 ( LEFT -1 RIGHT -1 TOP 0 BOTTOM 2 CORNER -1, -1, -1, -1 ) ( POSITION
 0 40 ) (WH 802 42 )
 RANK 2 ( LEFT -1 RIGHT -1 TOP 1 BOTTOM 3 CORNER -1, -1, -1, -1 ) ( POSITION
 0 80 ) (WH 802 42 )
 RANK 3 ( LEFT -1 RIGHT -1 TOP 2 BOTTOM -1 CORNER -1, -1, -1, -1 ) ( POSITIO
N 0 120 ) (WH 802 42 )
Progress [     1 /      5]
Progress [     2 /      5]
Progress [     3 /      5]
Progress [     4 /      5]
time_spent:0.330319 SECONDES
time_spent:0.343059 SECONDES
time_spent:0.346194 SECONDES
time_spent:0.323781 SECONDES
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

*Le temps d'exécution total=**1.343353** **s***

On met une configuration de **3 fois** les données initiales dans le fichier **config.txt** sauf itérations.

```
1 iterations            = 5
2 width                 = 2400
3 height                = 480
4 #obstacle_r            =
5 #obstacle_x            =
6 #obstacle_y            =
7 reynolds              = 300
8 inflow_max_velocity   = 0.300000
9 inflow_max_velocity   = 0.300000
.0 output_filename      = resultat.raw
.1 write_interval       = 150
```

*Le temps d'exécution total=**9.765341 s =>le temps a augmenté** **considérablement***

***Cas 2:**Augmenter les données pour une taille de problème fixe*
*On met une configuration de **3 fois** les données initiales suivantes:*
*vitesse des données entrantes **reynolds,** le facteur d'échelle **inflow_max_velocity ,***
*l'intervalle d'écriture entre les sorties **write_interval.***

```
iterations           = 5
width                = 800
height               = 160
#obstacle_r          =
#obstacle_x          =
#obstacle_y          =
reynolds             = 300
inflow_max_velocity  = 0.300000
inflow_max_velocity  = 0.300000
output_filename      = resultat.raw
write_interval       = 150
```

*Le temps d'exécution total=***1.352412 s =>*<u>le temps reste presque inchangé</u> (1.343353 s)***

Conclusion scalabilité :

On déduit d'après les résultats des deux cas étudiés précédemment que la scalabilité de notre code est ***mauvaise.*** *Une optimisation doit être faite pour améliorer la scalabilité de l'application.*

Optimisation du code :

Afin de déterminer la partie du code à optimiser, il faut savoir les fonctions qui coûtent le plus en terme de temps,pour cela on doit faire <u>un profilage de code</u>,avec l outil ***gprof*** *comme suit :*

1. *Dans le makefile on ajoute le flag* ***-pg***
2. *On fait notre* ***make*** *et on exécute avec* ***./lbm***
3. ***./lbm > md.txt***
4. ***gprof lbm>lbm.gprof***

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 22.24      0.10     0.10                             compute_equilibrium_profile
 20.02      0.19     0.09                             get_vect_norme_2
 15.57      0.26     0.07                             get_cell_velocity
 11.12      0.31     0.05                             Mesh_get_cell
 11.12      0.36     0.05                             propagation
  6.67      0.39     0.03                             compute_cell_collision
  6.67      0.42     0.03                             helper_compute_poiseuille
  2.22      0.43     0.01                             Mesh_get_cell
  2.22      0.44     0.01                             main
  2.22      0.45     0.01                             setup_init_state_global_po

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.
```

On ouvre le fichier ***lbm.gprof :***

D'après le fichier les fonctions qui coûtent le plus de temps sont :

- ***compute_equilibrium_profile()***
- ***get_vect_norme_2()***
- ***get_cell_velocity()***
- ***Mesh_get_cell()***
- ***propagation()***

Pour réduire le temps d'exécution de ces fonctions on utilisera le système de parallélisation ***OpenMP****.*

Parallélisation OpenMP :

- *La fonction* ***compute_equilibrium_profile()*** *et* ***propagation()*** *ne possède pas de zone a paralléliser,donc pas de parallélisation nécessaire.*
- *La fonction* ***get_vect_norme_2()*** *et* ***get_cell_velocity():****la parallélisation ici provoque un* ***interblocage****,donc pas de parallélisation nécessaire.*
- *La fonction* ***Mesh_get_cell():*** *elle fait un simple* ***return*** *d'une cellule du maillage donc pas de parallélisation nécessaire.*

Comme aucune parallélisation ne peut se faire à l'intérieur des fonctions précédentes , on cherche la zone ou elles sont la plupart utilisées , avec la condition que la parallélisation soit possible .

La fonction ***save_frame()*** *qui fait appel au fonctions* ***get_cell_density()****,* ***get_cell_velocity()*** *et* ***get_vect_norme2()*** *peut être parallélisée,par contre comme on est dans un contexte de 2 boucles imbriquées pour éviter une erreur de segmentation type* ***Abandon (core dumped) ,****on doit déclarer une section critique pour gérer la variable* ***cnt*** *,malgré que les sections critiques réellement ralentissent l'exécution du programme, on est obligé de l'utiliser.*

```
cnt = 0;
        //debut de parallelistion de la boucle (region parallele)
        #pragma omp parallel for
        for ( i = 1 ; i < mesh->width - 1 ; i++)
        {
                for ( j = 1 ; j < mesh->height - 1 ; j++)
                {
                        //compute macrospic values
                        density = get_cell_density(Mesh_get_cell(mesh, i, j));
                        get_cell_velocity(v,Mesh_get_cell(mesh, i, j),density);
                        norm = sqrt(get_vect_norme_2(v,v));

                        // debut de la section critique
                        #pragma omp critical
                        {
                                //fill buffer
                                buffer[cnt].density = density;
                                buffer[cnt].v = norm;
                                cnt++;
                                // variable cnt (critique) est executée par un seul thread  a la fois

                                //errors
                                assert(cnt <= WRITE_BUFFER_ENTRIES);

                                //flush buffer if full
                                if (cnt == WRITE_BUFFER_ENTRIES)
                                {
                                fwrite(buffer,sizeof(lbm_file_entry_t),cnt,fp);
                                cnt = 0;
                                }
                        }
                }
```

On exécute le programme ,et on voit que le temps a <u>diminué</u> pour la configuration initiale avec un seule processus :

Avant Parallélisation

```
louganifaouzi@louganifaouzi-ThinkPad
================== CONFIG =========
iterations            = 5
width                 = 800
height                = 160
obstacle_r            = 17.000000
obstacle_x            = 161.000000
obstacle_y            = 83.000000
reynolds              = 100.000000
reynolds              = 100.000000
inflow_max_velocity   = 0.100000
output_filename       = resultat.raw
write_interval        = 50
------------ Derived parameters ----
kinetic_viscosity     = 0.034000
relax_parameter       = 1.661130
=====================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BO
Progress [     1 /      5]
Progress [     2 /      5]
Progress [     3 /      5]
Progress [     4 /      5]
time_spent:1.311583 SECONDES
```

Après parallélisation OpenMp

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/
================== CONFIG ==================
iterations            = 5
width                 = 800
height                = 160
obstacle_r            = 17.000000
obstacle_x            = 161.000000
obstacle_y            = 83.000000
reynolds              = 100.000000
reynolds              = 100.000000
inflow_max_velocity   = 0.100000
output_filename       = resultat.raw
write_interval        = 50
------------ Derived parameters ------------
kinetic_viscosity     = 0.034000
relax_parameter       = 1.661130
=============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 C
Progress [     1 /      5]
Progress [     2 /      5]
Progress [     3 /      5]
Progress [     4 /      5]
time_spent:1.272349 SECONDES
```

<u>Parallélisation MPI</u> :

*Une analyse de la fonction principale **main()** nous informe qu'une barrière est implémentée après chaque une des fonction suivantes :**save_frame_all_domain()**,**special_cells()**,**collision()** ,**propagation()**,**lbm_comm_ghost_exchange()** .Par contre si on analyse les communications **mpi** qui ont lieu ,on déduit que juste les deux fonctions **save_frame_all_domain()**,**lbm_comm_ghost_exchange()** y participent .*
*Du coups a quoi sert de ralentir le programme (attendre tous les processus)=>**solution:supprimer les barrières dans la fonction principale main()***
*On exécute le programme ,et on voit que le temps a **<u>diminué</u> encore après** avoir supprimer les barrières.*

Après parallélisation OpenMp

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/
================== CONFIG ==================
iterations            = 5
width                 = 800
height                = 160
obstacle_r            = 17.000000
obstacle_x            = 161.000000
obstacle_y            = 83.000000
reynolds              = 100.000000
reynolds              = 100.000000
inflow_max_velocity   = 0.100000
output_filename       = resultat.raw
write_interval        = 50
------------ Derived parameters ------------
kinetic_viscosity     = 0.034000
relax_parameter       = 1.661130
=============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 C
Progress [     1 /      5]
Progress [     2 /      5]
Progress [     3 /      5]
Progress [     4 /      5]
time_spent:1.272349 SECONDES
```

Après suppression barrières MPI

```
================== CONFIG ==================
iterations            = 5
width                 = 800
height                = 160
obstacle_r            = 17.000000
obstacle_x            = 161.000000
obstacle_y            = 83.000000
reynolds              = 100.000000
reynolds              = 100.000000
inflow_max_velocity   = 0.100000
output_filename       = resultat.raw
write_interval        = 50
------------ Derived parameters ------------
kinetic_viscosity     = 0.034000
relax_parameter       = 1.661130
=============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 CORNER -1, -1, -1,
1 ) ( POSITION 0 0 ) (WH 802 162 )
Progress [     1 /      5]
Progress [     2 /      5]
Progress [     3 /      5]
Progress [     4 /      5]
time_spent:1.106842 SECONDES
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

Fuite mémoire et accès :

J'ai utilisé l'outil Valgrind afin de voir si il y a des fuites mémoire en exécutant : ***valgrind ./lbm***

```
Progress [    1 /     5]
Progress [    2 /     5]
Progress [    3 /     5]
Progress [    4 /     5]
time_spent:28.681031 SECONDES
==98564==
==98564== HEAP SUMMARY:
==98564==     in use at exit: 4,317 bytes in 10 blocks
==98564==   total heap usage: 1,865 allocs, 1,855 frees, 37,70
==98564==
==98564== LEAK SUMMARY:
==98564==    definitely lost: 0 bytes in 0 blocks
==98564==    indirectly lost: 0 bytes in 0 blocks
==98564==      possibly lost: 912 bytes in 3 blocks
==98564==    still reachable: 3,405 bytes in 7 blocks
==98564==         suppressed: 0 bytes in 0 blocks
==98564== Rerun with --leak-check=full to see details of leake
==98564==
==98564== Use --track-origins=yes to see where uninitialised v
==98564== For lists of detected and suppressed errors, rerun w
==98564== ERROR SUMMARY: 1 errors from 1 contexts (suppressed:
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

L'outil valgrind a détecter une erreur et afin d'avoir plus de détails on exécute :
valgrind -s ./lbm

```
==98925==    indirectly lost: 0 bytes in 0 blocks
==98925==      possibly lost: 912 bytes in 3 blocks
==98925==    still reachable: 26,975 bytes in 8 blocks
==98925==         suppressed: 0 bytes in 0 blocks
==98925== Rerun with --leak-check=full to see details of leaked memory
==98925==
==98925== Use --track-origins=yes to see where uninitialised values come
==98925== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==98925==
==98925== 1 errors in context 1 of 1:
==98925== Conditional jump or move depends on uninitialised value(s)
==98925==    at 0x4BA53DA: ??? (in /usr/lib/x86_64-linux-gnu/libmpich.so
==98925==    by 0x4BA6B48: ??? (in /usr/lib/x86_64-linux-gnu/libmpich.so
==98925==    by 0x4B87EB2: ??? (in /usr/lib/x86_64-linux-gnu/libmpich.so
==98925==    by 0x4A3FC81: ??? (in /usr/lib/x86_64-linux-gnu/libmpich.so
==98925==    by 0x4A3F9B2: PMPI_Init (in /usr/lib/x86_64-linux-gnu/libmp
1.8)
==98925==    by 0x109915: main (main.c:146)
==98925==
==98925== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Expiration de la minuterie durant l'établissement du profile
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

L'erreur n'est pas méchante donc on déduit qu'on a pas de fuite mémoire.

Pour l'accès la fonction ***collision()*** *(fichier* ***l****bm_phys.c) on a un accès au données non contiguë en mémoire.Car ses deux boucles sont* ***inversées****,parcourir les colonnes* ***j*** *puis les lignes* ***i*** *dans une mémoire ralentit l'accès a la case mémoire.Aussi ça contredit le*

rôle de la fonction qui retourne l ensemble des cellules organisés après avoir récupérer leurs coordonnées (x,y) et non pas (y,x) avec ***Mesh_get_cell()****.*
Dans la fonction ***propagation()*** *on rencontre le même cas, donc on doit inverser aussi les boucles .*
=>solution:mettre les boucles sous la forme (i,j)

```
312 void collision(Mesh * mesh_out,const Mesh * mesh_in)
313 {
314         //vars
315         int i,j;
316 
317         //errors
318         assert(mesh_in->width == mesh_out->width);
319         assert(mesh_in->height == mesh_out->height);
320         /*on met les boucle (j,i) en forme de (i,j)  pour avoir un acces contigue a la memoire */
321         //loop on all inner cells
322         for( j = 1 ; j < mesh_in->height - 1 ; j++)
323                 for( i = 1 ; i < mesh_in->width - 1 ; i++ )
324                         compute_cell_collision(Mesh_get_cell(mesh_out, i, j),Mesh_get_cell(mesh_in, i, j));
325 }
```

```
333 void propagation(Mesh * mesh_out,const Mesh * mesh_in)
334 {
335         //vars
336         int i,j,k;
337         int ii,jj;
338         //loop on all cells
339         /*on met les boucle (j,i) en forme de (i,j)  pour avoir un acces contigue a la memoire */
340         for ( i = 0 ; i < mesh_out->width; i++)
341         {
342                 for ( j = 0 ; j < mesh_out->height ; j++)
343                 {
344                         //for all direction
345                         for ( k  = 0 ; k < DIRECTIONS ; k++)
346                         {
347                                 //compute destination point
348                                 ii = (i + direction_matrix[k][0]);
349                                 jj = (j + direction_matrix[k][1]);
350                                 //propagate to neighboor nodes
351                                 if ((ii >= 0 && ii < mesh_out->width) && (jj >= 0 && jj < mesh_out->height))
352                                         Mesh_get_cell(mesh_out, ii, jj)[k] = Mesh_get_cell(mesh_in, i, j)[k];
353                         }
354                 }
355         }
356 }
```

On exécute le programme avec ***./lbm*** *,et on voit que le temps a* ***<u>diminué</u> encore après*** *avoir changer l ordre des boucles .*

Après suppression barrières MPI

```
================== CONFIG ==================
iterations           = 5
width                = 800
height               = 160
obstacle_r           = 17.000000
obstacle_x           = 161.000000
obstacle_y           = 83.000000
reynolds             = 100.000000
reynolds             = 100.000000
inflow_max_velocity  = 0.100000
output_filename      = resultat.raw
write_interval       = 50
------------ Derived parameters -------------
kinetic_viscosity    = 0.034000
relax_parameter      = 1.661130
============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 CORNER -1, -1, -1,
1 ) ( POSITION 0 0 ) (WH 802 162 )
Progress [    1 /     5]
Progress [    2 /     5]
Progress [    3 /     5]
Progress [    4 /     5]
time_spent:1.106842 SECONDES
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

Après avoir changer l ordre des boucles

```
================== CONFIG ==================
iterations           = 5
width                = 800
height               = 160
obstacle_r           = 17.000000
obstacle_x           = 161.000000
obstacle_y           = 83.000000
reynolds             = 100.000000
reynolds             = 100.000000
inflow_max_velocity  = 0.100000
output_filename      = resultat.raw
write_interval       = 50
------------ Derived parameters -------------
kinetic_viscosity    = 0.034000
relax_parameter      = 1.661130
============================================
 RANK 0 ( LEFT -1 RIGHT -1 TOP -1 BOTTOM -1 CORNER -1, -1, -1,
1 ) ( POSITION 0 0 ) (WH 802 162 )
Progress [    1 /     5]
Progress [    2 /     5]
Progress [    3 /     5]
Progress [    4 /     5]
time_spent:0.994988 SECONDES
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Bureau/TOP_PROJET$
```

Effets de l'optimisation :

On commence par le temps émit dans chaque fonction par notre application , on fait la même étape de profilage de code, avec ***gprof*** *précédemment .on aura :*

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 37.07      0.10     0.10 14011498     0.00     0.00  get_vect_norme_2
 18.53      0.15     0.05  6975504     0.00     0.00  compute_equilibrium_profile
 11.12      0.18     0.03 10335892     0.00     0.00  Mesh_get_cell
 11.12      0.21     0.03   567296     0.00     0.00  get_cell_velocity
  7.41      0.23     0.02   575266     0.00     0.00  get_cell_density
  7.41      0.25     0.02   512000     0.00     0.00  compute_cell_collision
  3.71      0.26     0.01  2338632     0.00     0.00  helper_compute_poiseuille
  3.71      0.27     0.01        4     2.50    44.20  collision
  0.00      0.27     0.00   512000     0.00     0.00  lbm_cell_type_t_get_cell
  0.00      0.27     0.00   116141     0.00     0.00  Mesh_get_cell
  0.00      0.27     0.00     3604     0.00     0.00  compute_bounce_back
  0.00      0.27     0.00        4     0.00     0.00  lbm_comm_height
  0.00      0.27     0.00        4     0.00     0.00  lbm_comm_width
  0.00      0.27     0.00        4     0.00     6.76  propagation
  0.00      0.27     0.00        4     0.00     0.00  special_cells
  0.00      0.27     0.00        1     0.00     0.00  open_output_file
  0.00      0.27     0.00        1     0.00     0.00  save_frame
  0.00      0.27     0.00        1     0.00     0.00  write_file_header
```

On voit clairement que les nouvelles valeurs du temps passé dans chaque fonction a baissé, ce qui indique que l'optimisation a apporté des améliorations .

Mais coté scalabilité ?

On fait la même étude faite au début avec les mèmes configurations ***(cas 1 et 2)*** *ce tableau récapitule les résultats .*

	Cas 1 : *Augmenter les données et la taille totale du problème.*	***Cas 2:*** *Augmenter les données pour une taille de problème fixe*
Temps d'exécution total avant optimisation	***9.765341 s***	**1.352412 s**
Temps d'exécution total avant optimisation	**6,327456 s**	**0,9993s**

Les résultats sont beaucoup mieux après optimisation Donc, l'optimisation appliqué au code, a évolué la scalabilité de programme, malgré que on n'a pas encore atteint la scalabilité parfaite.

Problèmes non résolus :

Parallélisation OpenMP:*Le code est pleins de boucles et partie parallélisables mais le problème est dans les variables partagées , j'ai essayer de régler ce problème mais cela implique une grande modification du code initial.*

Gnuplot:*j'ai essayer de visualiser les résultats a chaque fois ,malgré que je maîtrise un peux mais mon Ubuntu refuse d'installer une version récente.*

Exécution du programme pour un nombre important de processus.

Conclusion:

Afin d'avoir une scalabilité parfaite il faut respecter certaines règles importantes lors de l'optimisation d'une application:

1. *Stabilité du système d'exploitation dans lequel le code ou l'application va s'exécuter.(processus en arrière plan, fréquence)*
2. *Correction des (bugs) dans le code.*
3. *Mesures après chaque modification de code pour voir l'impact .*
4. *Étude de scalabilité du code initial et final.*
5. *Parallélisation MPI et éviter l'usage excessif de barrières inutiles.*
6. *Parallélisation OpenMP pour les boucles .*
7. *Identifier les fuite mémoire ,chois de la meilleure méthode d'accès au données.*