```
# Included files:
 * julia.c


# Ignored C and H files
 * compute_julia_pixel.c
 * julia.h
 * CMakeCCompilerId.c
 * write_bmp_header.c




commit fcf03c70bdb5da664779b5660933f9fe1ec8d110
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:58:56 2023 +0200

    📝 README.md add cache optimisation description

 TP/TP2-julia/julia/README.md | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)

commit c96425e2e243838257e1236d84e72bf20a4858c0
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:56:01 2023 +0200

    📝 generate rendu pdf file

 TP/TP2-julia/julia/rendu.pdf | Bin 56374 -> 57155 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)

commit 7d65d526446498177b92b1a3c48dae6f506950f0
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:55:44 2023 +0200

    📝 README.md debrief TP

 TP/TP2-julia/julia/README.md | 12 ++++++------
 1 file changed, 6 insertions(+), 6 deletions(-)

commit 32853af2eccab9345adbac62a5085d14c0bc58cd
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:55:13 2023 +0200

    ⚰️ julia.c: delete useless macros

 TP/TP2-julia/julia/julia.c | 2 --
 1 file changed, 2 deletions(-)

commit 13c969a17687ad86f949774c17b95b4ab876fef3
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:42:45 2023 +0200

    ♻️ move README.md in order to proj2pdf to work

 TP/TP2-julia/{ => julia}/README.md |   0
 TP/TP2-julia/julia/proj2pdf        | 247 ++++++++++++++++++++++++++++++++++++++++
 TP/TP2-julia/julia/rendu.pdf       | Bin 0 -> 56374 bytes
 3 files changed, 247 insertions(+)

commit ddc4a71e028a0cd813dc25566bc28ac6a01ba0d2
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:31:40 2023 +0200

    💄 julia.c: fix clang-tidy

 TP/TP2-julia/julia/julia.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit cd04f367313b58b13ea14bd21418c3cbb01d5b77
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:27:13 2023 +0200

    🎨 julia.c clang-format

 TP/TP2-julia/julia/julia.c | 55 ++++++++++++++++++++++------------------------------
 1 file changed, 26 insertions(+), 29 deletions(-)

commit aa1cd168ed713270a4c385d396fd7841cd5b25dd
Author: AlexisHamon <alexis.hamon24@gmail.com>
Date:   Fri Sep 29 00:25:59 2023 +0200

    question 5

 TP/TP2-julia/julia/julia.c | 82 +++++++++++++++++++++++++++++++++++++++++++----------
 1 file changed, 67 insertions(+), 15 deletions(-)
```

TP2 : Julia, threads, et répartition de charge

Programmation multi-threadée

Exercice 1: La fractale de Julia

Nous nous retrouvons dans le cas classique du pavage d'un domaine sans vecteur de dépendance. On peut donc parraléliser à la volée comme bon nous le semble.

On définit une structure d'itérateur, `julia_it`, composée d'un mutex pour l'attribution des pavés.

Tant que le travail n'est pas fini `julia_it->i = 0`, chaque thread `threadCalc` va obtenir un numéro de pavé, récupérer les bornes de ce dernier grâce aux fonctions `juliaGetXmin`, `juliaGetXmax`, `juliaGetYmin` et `juliaGetYmax`.

Toute l'image est alors parcourue via une attribution dynamique des pavés aux différents threads.

Bien que l'affirmation de la question 5 est vraie, un autre facteur important est l'optimisation du cache propre à chaque machine. Le compromis souhaité se trouve valoir sur ma machine des pavés de taille 32.

```c
/*
 ********************************************************
 ********* Fichier julia.c *********
 ********************************************************
 */
#define _GNU_SOURCE

#define min(x, y) ((x) < (y) ? (x) : (y))

#include "julia.h"
#include <pthread.h>
#include <sched.h>
#include <stddef.h>
#include <stdlib.h>

struct julia_it {
  // Those are constants between threads
  int size_tile;
  int n_tile_height;
  // Those needs to be accessed threw a critical space
  pthread_mutex_t mutex;
  int i;
};

struct julia_it *juliaAlloc(int width, int height, int size_tile) {
  struct julia_it *it = malloc(sizeof(struct julia_it));
  it->size_tile = size_tile;
  int n_tile_width = width / size_tile + 1;
  it->n_tile_height = height / size_tile + 1;
  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
  it->mutex = mutex;
  it->i = n_tile_width * it->n_tile_height;
  return it;
}

int juliaGet(struct julia_it *it) {
  pthread_mutex_lock(&it->mutex);
  int i = (it->i ? (it->i--) : (0));
  pthread_mutex_unlock(&it->mutex);
  return i;
}

int juliaGetYmin(struct julia_it *it, int i) {
  return ((i - 1) % it->n_tile_height) * it->size_tile;
}
int juliaGetYmax(struct julia_it *it, int i) {
  return (((i - 1) % it->n_tile_height) + 1) * it->size_tile;
}

int juliaGetXmin(struct julia_it *it, int i) {
  return ((i - 1) / it->n_tile_height) * it->size_tile;
}
int juliaGetXmax(struct julia_it *it, int i) {
  return (((i - 1) / it->n_tile_height) + 1) * it->size_tile;
}

void juliaFree(struct julia_it *it) { free(it); }

int cpucount() {
  cpu_set_t cpuset;
  sched_getaffinity(0, sizeof(cpuset), &cpuset);
  return CPU_COUNT(&cpuset);
}

struct img {
  int width;
  int height;
  unsigned char *pixel;
};

struct thread_args {
  struct img *img;
  struct julia_it *it;
};

void *threadCalc(void *ptr) {
  struct thread_args *args = (struct thread_args *)ptr;
  int i;

  while ((i = juliaGet(args->it))) {
    int y_upb = min(args->img->height, juliaGetYmax(args->it, i));
    int x_upb = min(args->img->width, juliaGetXmax(args->it, i));
    for (int y = juliaGetYmin(args->it, i); y < y_upb; ++y) {
      for (int x = juliaGetXmin(args->it, i); x < x_upb; ++x) {
        compute_julia_pixel(x, y, args->img->width, args->img->height, 1.0,
                            &args->img->pixel[(y * args->img->width + x) * 3]);
      }
    }
  }

  return NULL;
}

int main(int argc, char **argv) {
  if (argc < 2)
    return 1;
  int n = atoi(argv[1]);

  if (n < 1)
    return 0;

  struct img *img = malloc(sizeof(struct img));
  img->width = 2 * n;
  img->height = n;
  img->pixel = malloc(img->width * img->height * 3 * sizeof(unsigned char));

  FILE *out = fopen("outfile.bmp", "w");
  if (out == NULL) {
    perror("Cannot open outfile");
    exit(1);
  }

  int cpu_count = cpucount();

  struct thread_args *args = malloc(sizeof(struct thread_args));
  args->img = img;
  args->it = juliaAlloc(img->width, img->height, 32);

  pthread_t *pth_t = malloc(cpu_count * sizeof(pthread_t));

  for (unsigned th = 0; th < cpu_count; ++th)
    pthread_create(&pth_t[th], NULL, threadCalc, (void *)args);

  for (unsigned th = 0; th < cpu_count; ++th)
    pthread_join(pth_t[th], NULL);

  free(args->it);
  free(args);
  free(pth_t);

  write_bmp_header(out, img->width, img->height);
  fwrite(img->pixel, img->width * img->height * 3, 1, out);
  fclose(out);

  free(img->pixel);
  free(img);
  return 0;
}


//////////////////////////////// clang-tidy output
```