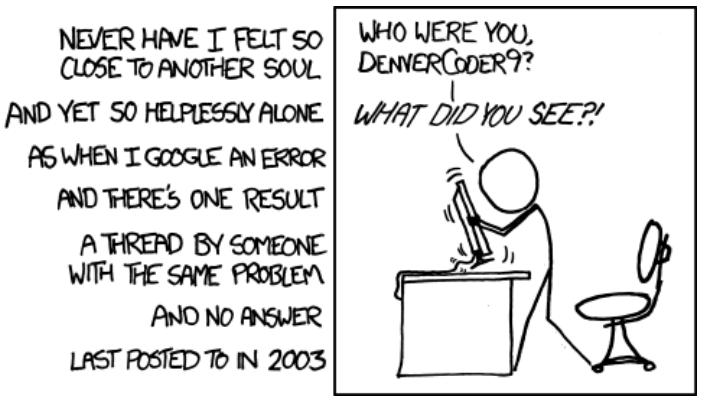
MPI* Info - TP10

Utilisation de threads



O. Caffier

1 Somme dans un tableau

```c
struct data_s {
    int debut;
    int fin;
    int* tab;
    int emplacement_tmp_res;
    int** tmp_res;
};

typedef struct data_s data;

void* mythread(void* arg) {
    data* arg_tmp = (data*)arg;
    int id_tab_tmp_res = arg_tmp->emplacement_tmp_res;
    for (int i = arg_tmp->debut; i < arg_tmp->fin + 1; i++) {
        *((arg_tmp->tmp_res)[id_tab_tmp_res]) += (arg_tmp->tab)[i];
    }
    return NULL;
}

int main(int argc, char* argv[]) {
    assert(argc == 3);
    int len = atoi(argv[1]);
    int n = atoi(argv[2]);
    int sub_div = len / n;
    int* tab = malloc(len * sizeof(int));
    for (int i = 0; i < len; i++) {
        tab[i] = i;
    }

    pthread_t* tab_thread = malloc(n * sizeof(pthread_t));
    int** tmp_res_main = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        tmp_res_main[i] = malloc(sizeof(int));
        *(tmp_res_main[i]) = 0;
    }

    data** tab_data = malloc(n * sizeof(data*));
    for (int i = 0; i < n; i++) {
        tab_data[i] = malloc(sizeof(data));
        tab_data[i]->debut = i * sub_div;
        tab_data[i]->fin = (i + 1) * sub_div - 1;
        tab_data[i]->emplacement_tmp_res = i;
        tab_data[i]->tmp_res = tmp_res_main;
        tab_data[i]->tab = tab;
    }

    // Debut du main
    for (int id_thread = 0; id_thread < n; id_thread++) {
        pthread_create(&(tab_thread[id_thread]), NULL, mythread, tab_data[id_thread]);
    }

    for (int id_thread = 0; id_thread < n; id_thread++) {
        pthread_join(tab_thread[id_thread], NULL);
    }
    int res = 0;
    for (int i = 0; i < n; i++) {

        res += *(tmp_res_main[i]);
    }

    printf("Somme : %d \n", res);

    return 0;
}
```

2 Automates cellulaires

2.1 Implémentation naïve

La version naïve de l'implémentation utilisera le type suivant pour le ruban :

```c
typedef short int* ruban;
```

```c
short int regle_evolution(short int left, short int center, short int right){
    if (left == 1){
        if (center == 1){
            if (right==1){
                return 0;
            }
            else{
                return 0;
            }
        }
        else{
            if (right==1){
                return 0;
            }
            else{
                return 1;
            }
        }
    }
    else{
        if (center==1){
            if (right==1){
                return 1;
            }
            else{
                return 1;
            }
        }
        else{
            if (right==1){
                return 1;
            }
            else{
                return 0;
            }
        }
    }
}

void affiche_automate(short int* ruban, int taille){
    for (int i =0 ; i<taille; i++){
        if (ruban[i] == 1){
            printf("X");
        }
        else{
            printf(" ");
        }
    }
    printf("\n");
}
```

```c
void naive_transitions(short int* ruban, int taille, int nb_tour){
    //affiche_automate(ruban,taille);
    for (int num_boucle = 0 ; num_boucle < nb_tour ; num_boucle ++){
        short int* ruban_tmp = malloc(taille*sizeof(short int));
        ruban_tmp[0] = regle_evolution(0,ruban[0],ruban[1]);
        for (int id_centre = 1; id_centre < taille-1; id_centre++){
            ruban_tmp[id_centre] = regle_evolution(ruban[id_centre-1],ruban[id_centre],ruban[id_centre+1]);
        }
        ruban_tmp[taille-1]=regle_evolution(ruban[taille-2],ruban[taille-1],0);

        for (int i = 0 ; i<taille; i++){
            ruban[i] = ruban_tmp[i];
        }
        free(ruban_tmp);
        //affiche_automate(ruban,taille);
    }
}
```

2.2 Implémentation multi-thread

```c
void init_ruban_1(short int* ruban, int len){
    int middle = len/2;
    for (int i =0; i<len; i++){
        ruban[i]=0;
    }
    ruban[middle] = 1;
}

short int evol(short int left, short int center, short int right){
    if (left == 1){
        if (center == 1){
            if (right==1){
                return 0;
            }
            else{
                return 0;
            }
        }
        else{
            if (right==1){
                return 0;
            }
            else{
                return 1;
            }
        }
    }
    else{
        if (center==1){
            if (right==1){
                return 1;
            }
            else{
                return 1;
            }
        }
        else{
            if (right==1){
                return 1;
            }
            else{
                return 0;
            }
        }
    }
}
```

```c

void affiche_ruban(short int* ruban, int taille){
    for (int i =0 ; i<taille; i++){
        if (ruban[i] == 1){
            printf("X");
        }
        else{
            printf("-");
        }
    }
    printf("\n");
}

void* mythread(void* arg){
    data_thread* arg_tmp = (data_thread*)arg;

    short int remember = arg_tmp->ruban[arg_tmp->debut];
    arg_tmp->ruban[arg_tmp->debut] = evol(arg_tmp->avant, arg_tmp->ruban[arg_tmp->debut], arg_tmp->ruban[
        arg_tmp->debut+1]);

    for (unsigned int id_centre = arg_tmp->debut +1 ; id_centre < arg_tmp->fin; id_centre++ ){
        short int remember_tampon = arg_tmp->ruban[id_centre];
        arg_tmp->ruban[id_centre] = evol(remember, arg_tmp->ruban[id_centre], arg_tmp->ruban[id_centre+1]);
        remember = remember_tampon;
    }

    arg_tmp->ruban[arg_tmp->fin] = evol(remember, arg_tmp->ruban[arg_tmp->fin], arg_tmp->apres);
    return NULL;
}

void transitions_tour(short int* ruban, int len, int nb_thread){
    int sub_div = len/nb_thread;

    pthread_t* tab_thread = malloc(nb_thread* sizeof(pthread_t));

    data_thread** tab_data = malloc(nb_thread * sizeof(data_thread*));

    // Cas id_thread == 0
    tab_data[0] = malloc(sizeof(data_thread));
    tab_data[0]->debut = 0;
    tab_data[0]->fin = sub_div-1;
    tab_data[0]->avant = 0;
    tab_data[0]->apres = ruban[sub_div];
    tab_data[0]->ruban = ruban;


    for (int id_thread = 1; id_thread <nb_thread-1 ; id_thread++){
        tab_data[id_thread] = malloc(sizeof(data_thread));
        tab_data[id_thread]->debut = id_thread*sub_div;
        tab_data[id_thread]->fin = (id_thread + 1)*sub_div -1 ;
        tab_data[id_thread]->avant = ruban[id_thread*sub_div -1];
        tab_data[id_thread]->apres = ruban[(id_thread+1)*sub_div];
        tab_data[id_thread]->ruban = ruban;

    }



    // Cas id_thread == nb_thread - 1
    tab_data[nb_thread -1] = malloc(sizeof(data_thread));
    tab_data[nb_thread -1]->debut = (nb_thread -1)*sub_div;
    tab_data[nb_thread -1]->fin = nb_thread*sub_div -1 ;
    tab_data[nb_thread -1]->avant = ruban[(nb_thread-1)*sub_div -1];
    tab_data[nb_thread -1]->apres = 0;
    tab_data[nb_thread -1]->ruban = ruban;

```

```
112
113     for (int id_thread =0 ; id_thread<nb_thread; id_thread++){
114         pthread_create(&(tab_thread[id_thread]),NULL,mythread,tab_data[id_thread]);
115     }
116
117     for (int id_thread = 0; id_thread < nb_thread; id_thread++) {
118         pthread_join(tab_thread[id_thread], NULL);
119     }
120
121 }
122
123
124 void transitions(short int* ruban, int len, int nb, int nb_thread){
125    //affiche_ruban(ruban,len);
126     for (int i=0; i<nb; i++){
127         transitions_tour(ruban,len,nb_thread);
128         //affiche_ruban(ruban,len);
129     }
130 }
131
132
133 int main(){
134     clock_t t;
135     t = clock();
136
137     int len = 10000;
138     int nb_thread = 50;
139     int nb_tours = 5000;
140
141
142     short int *ruban = malloc(sizeof(short int)*len);
143     init_ruban_1(ruban, len);
144
145
146     transitions(ruban, len, nb_tours, nb_thread);
147     t = clock() - t;
148     printf(" \n =====> Perf = %f seconds", ((float)t) / CLOCKS_PER_SEC);
149     return 0;
150 }
```