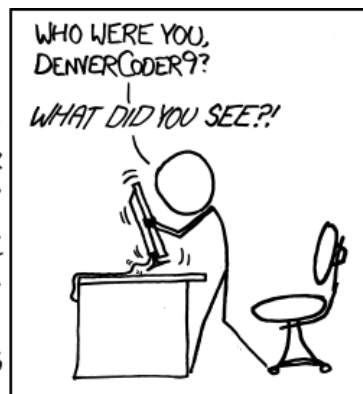


MPI* Info - TP10

Utilisation de threads

NEVER HAVE I FELT SO
CLOSE TO ANOTHER SOUL
AND YET SO HELPLESSLY ALONE
AS WHEN I GOOGLE AN ERROR
AND THERE'S ONE RESULT
A THREAD BY SOMEONE
WITH THE SAME PROBLEM
AND NO ANSWER
LAST POSTED TO IN 2003



1 Somme dans un tableau

```
1 struct data_s {
2     int debut;
3     int fin;
4     int* tab;
5     int emplacement_tmp_res;
6     int** tmp_res;
7 };
8
9 typedef struct data_s data;
10
11 void* mythread(void* arg) {
12     data* arg_tmp = (data*)arg;
13     int id_tab_tmp_res = arg_tmp->emplacement_tmp_res;
14     for (int i = arg_tmp->debut; i < arg_tmp->fin + 1; i++) {
15         *((arg_tmp->tmp_res)[id_tab_tmp_res]) += (arg_tmp->tab)[i];
16     }
17     return NULL;
18 }
19
20 int main(int argc, char* argv[]) {
21     assert(argc == 3);
22     int len = atoi(argv[1]);
23     int n = atoi(argv[2]);
24     int sub_div = len / n;
25     int* tab = malloc(len * sizeof(int));
26     for (int i = 0; i < len; i++) {
27         tab[i] = i;
28     }
29
30     pthread_t* tab_thread = malloc(n * sizeof(pthread_t));
31     int** tmp_res_main = malloc(n * sizeof(int*));
32     for (int i = 0; i < n; i++) {
33         tmp_res_main[i] = malloc(sizeof(int));
34         *(tmp_res_main[i]) = 0;
35     }
36
37     data** tab_data = malloc(n * sizeof(data*));
38     for (int i = 0; i < n; i++) {
39         tab_data[i] = malloc(sizeof(data));
40         tab_data[i]->debut = i * sub_div;
41         tab_data[i]->fin = (i + 1) * sub_div - 1;
42         tab_data[i]->emplacement_tmp_res = i;
43         tab_data[i]->tmp_res = tmp_res_main;
44         tab_data[i]->tab = tab;
45     }
46
47     // Debut du main
48     for (int id_thread = 0; id_thread < n; id_thread++) {
49         pthread_create(&(tab_thread[id_thread]), NULL, mythread, tab_data[id_thread]);
50     }
51
52     for (int id_thread = 0; id_thread < n; id_thread++) {
53         pthread_join(tab_thread[id_thread], NULL);
54     }
55     int res = 0;
56     for (int i = 0; i < n; i++) {
57
58         res += *(tmp_res_main[i]);
59     }
60
61     printf("Somme : %d \n", res);
62
63     return 0;
64 }
```

2 Automates cellulaires

2.1 Implémentation naïve

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

```
1 typedef short int* ruban;

1 short int regle_evolution(short int left, short int center, short int right){
2     if (left == 1){
3         if (center == 1){
4             if (right==1){
5                 return 0;
6             }
7             else{
8                 return 0;
9             }
10        }
11        else{
12            if (right==1){
13                return 0;
14            }
15            else{
16                return 1;
17            }
18        }
19    }
20    else{
21        if (center==1){
22            if (right==1){
23                return 1;
24            }
25            else{
26                return 1;
27            }
28        }
29        else{
30            if (right==1){
31                return 1;
32            }
33            else{
34                return 0;
35            }
36        }
37    }
38 }

39
40 void affiche_automate(short int* ruban, int taille){
41     for (int i =0 ; i<taille; i++){
42         if (ruban[i] == 1){
43             printf("X");
44         }
45         else{
46             printf(" ");
47         }
48     }
49     printf("\n");
50 }
```

```

1 void naive_transitions(short int* ruban, int taille, int nb_tour){
2     //affiche_automate(ruban,taille);
3     for (int num_boucle = 0 ; num_boucle < nb_tour ; num_boucle ++){
4         short int* ruban_tmp = malloc(taille*sizeof(short int));
5         ruban_tmp[0] = regle_evolution(0,ruban[0],ruban[1]);
6         for (int id_centre = 1; id_centre < taille-1; id_centre++){
7             ruban_tmp[id_centre] = regle_evolution(ruban[id_centre-1],ruban[id_centre],ruban[id_centre+1]);
8         }
9         ruban_tmp[taille-1]=regle_evolution(ruban[taille-2],ruban[taille-1],0);
10
11         for (int i = 0 ; i<taille; i++){
12             ruban[i] = ruban_tmp[i];
13         }
14         free(ruban_tmp);
15         //affiche_automate(ruban,taille);
16     }
17 }

```

2.2 Implémentation multi-thread

```

1 void init_ruban_1(short int* ruban, int len){
2     int middle = len/2;
3     for (int i =0; i<len; i++){
4         ruban[i]=0;
5     }
6     ruban[middle] = 1;
7 }
8
9 short int evol(short int left, short int center, short int right){
10     if (left == 1){
11         if (center == 1){
12             if (right==1){
13                 return 0;
14             }
15             else{
16                 return 0;
17             }
18         }
19         else{
20             if (right==1){
21                 return 0;
22             }
23             else{
24                 return 1;
25             }
26         }
27     }
28     else{
29         if (center==1){
30             if (right==1){
31                 return 1;
32             }
33             else{
34                 return 1;
35             }
36         }
37         else{
38             if (right==1){
39                 return 1;
40             }
41             else{
42                 return 0;
43             }
44         }
45     }
46 }

```

```

47
48 void affiche_ruban(short int* ruban, int taille){
49     for (int i = 0 ; i < taille; i++){
50         if (ruban[i] == 1){
51             printf("X");
52         }
53         else{
54             printf("-");
55         }
56     }
57     printf("\n");
58 }
59
60 void* mythread(void* arg){
61     data_thread* arg_tmp = (data_thread*)arg;
62
63     short int remember = arg_tmp->ruban[arg_tmp->debut];
64     arg_tmp->ruban[arg_tmp->debut] = evol(arg_tmp->avant, arg_tmp->ruban[arg_tmp->debut], arg_tmp->ruban[
        arg_tmp->debut+1]);
65
66     for (unsigned int id_centre = arg_tmp->debut + 1 ; id_centre < arg_tmp->fin; id_centre++){
67         short int remember_tampon = arg_tmp->ruban[id_centre];
68         arg_tmp->ruban[id_centre] = evol(remember, arg_tmp->ruban[id_centre], arg_tmp->ruban[id_centre+1]);
69         remember = remember_tampon;
70     }
71
72     arg_tmp->ruban[arg_tmp->fin] = evol(remember, arg_tmp->ruban[arg_tmp->fin], arg_tmp->apres);
73     return NULL;
74 }
75
76 void transitions_tour(short int* ruban, int len, int nb_thread){
77     int sub_div = len/nb_thread;
78
79     pthread_t* tab_thread = malloc(nb_thread* sizeof(pthread_t));
80
81     data_thread** tab_data = malloc(nb_thread * sizeof(data_thread*));
82
83     // Cas id_thread == 0
84     tab_data[0] = malloc(sizeof(data_thread));
85     tab_data[0]->debut = 0;
86     tab_data[0]->fin = sub_div-1;
87     tab_data[0]->avant = 0;
88     tab_data[0]->apres = ruban[sub_div];
89     tab_data[0]->ruban = ruban;
90
91
92     for (int id_thread = 1; id_thread < nb_thread-1 ; id_thread++){
93         tab_data[id_thread] = malloc(sizeof(data_thread));
94         tab_data[id_thread]->debut = id_thread*sub_div;
95         tab_data[id_thread]->fin = (id_thread + 1)*sub_div - 1 ;
96         tab_data[id_thread]->avant = ruban[id_thread*sub_div - 1];
97         tab_data[id_thread]->apres = ruban[(id_thread+1)*sub_div];
98         tab_data[id_thread]->ruban = ruban;
99     }
100
101
102
103
104     // Cas id_thread == nb_thread - 1
105     tab_data[nb_thread - 1] = malloc(sizeof(data_thread));
106     tab_data[nb_thread - 1]->debut = (nb_thread - 1)*sub_div;
107     tab_data[nb_thread - 1]->fin = nb_thread*sub_div - 1 ;
108     tab_data[nb_thread - 1]->avant = ruban[(nb_thread-1)*sub_div - 1];
109     tab_data[nb_thread - 1]->apres = 0;
110     tab_data[nb_thread - 1]->ruban = ruban;
111

```

```

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 }

```