Informatique Embarquée
M2 / 2017

Colimaçon

Remarques

- Respectez les consignes
  - Page de garde
  - Votre nom
  - PDF (un rapport, pas un roman, pas de txt, pas de .docx, .PPTX) vérifiez qu'il est lisible.
  - Arborescence
  - Ne m'envoyez pas vos « cochonneries »
    - *~ *.o *.a executables, __MACOSX, .nfsxxx, ._toto
    - .cbp

TEMPS

- Estimés :
  - De 1H à 12H (1/2h À 42h, 3H à 12H, 1H à 10H!)
  - Moyenne 2016 : 4H30 / Médiane 4H
  - Moyenne 2015 : 5H30 / Médiane 5H
  - Moyenne 2014: 5H45 / Médiane : 5H
- Réels :
  - De 1H à 144H (1H30 à 28H, 3H à 32H , 3H30 à 20H)
  - Moyenne : 8H30 / Médiane : 6H
  - (2016 8h) (2015 10H) (2014 : 7H) (2013 :11H) / (2016 8h) (2015 : 9H) (2014 : 7H50) (2013 : 8H)
- Ratio :
  - De 0,7 à 36 Moyenne : 2,2 - Médiane : 1,5

NOTES

Moyenne	13,2
Médiane	14
Max	19
Min	5
NB soumissions	58

Remarques

- Propriétés du Makefile
  - Il en faut un !
  - make => fait ce qu'il faut
  - make ne refait rien, pas d'erreur !
  - make clean ; make ne devrait rien refaire
  - make mrproper ; make doit tout refaire
  - touch toto.h ; make doit refaire le minimum !
  - touch main.c ; make doit refaire le minimum !

Remarques

- Testez ! Vérifiez votre programme
  - 1x1, 1x10, 10x1
  - 2x2, ... carrés
  - Rectangles allongés 4x8
  - Rectangles verticaux 12x6
  - Idem en pair/impair et impair/pair
  - Grandes tailles
  - Allocations mémoires impossibles
  - Arguments invalides -1 -1 ! a b !
  - 0 9 (division par zéro!)

Remarques

- Algorithme
  - KISS !
  - En général : 1st to last
    - 4 boucles
    - 1 boucle et test de direction
    - Test de direction sur indices ou case déjà remplie
  - En moyenne 226 lignes en 8H30
    - De 13 à 725

Remarques

- Attention !
  - Si on a libcoli.a et libcoli.so dans le même répertoire, l'éditions de liens se fait le plus souvent avec libcoli.a
  - Se vérifie avec ldd

Remarques

- Formule val = f(i,j)
  - Permet un balayage linéaire => utilisation du cache
  - Et parallélisation triviale... indépendamment de son intérêt.
- Tests automatiques
  - Doivent valider le résultat (remplissage correct du tableau)

Simple ? :-) !

```
 Cmp = 1;              /* la valeur a inserer */
 top = 0;              /* indice de la ligne du haut a remplir */
 bottom = rows - 1;    /* indice de la ligne du bas a remplir */
 left = 0;             /* indice de la colonne de gauche a remplir */
 right = columns - 1; /* indice de la colonne de droite a remplir */

 while (cmp <= rows * columns) {
     for (j=left; j<=right && cmp<=rows*columns; j++)   /* haut */
            (*array)[top * columns + j] = cmp++;
     for (i=top+1; i<=bottom && cmp<=rows*columns; i++) /* droite */
            (*array)[i * columns + right] = cmp++;
     for (j=right-1; j>=left && cmp<=rows*columns; j--) /* bas */
            (*array)[bottom * columns + j] = cmp++;
     for (i=bottom-1; i>top && cmp<=rows*columns; i--) /* gauche */
            (*array)[i * columns + left] = cmp++;

     /* decalage des lignes et des colonnes a remplir */
     top++; bottom--; left++; right--;
}
```

Remplissage « linéaire »

```
for (row = 0; row < HEIGHT; row++) {
    for (col = 0; col < WIDTH; col++) {

      // find the rectangle this cell resides in.
      depth = DEPTH(row,col);

      // find relative x, y for this cell in the rectangle it belongs to
      rel_row = row - depth;
      rel_col = col - depth;

      if ((rel_row == 0) || (rel_col == rect[depth].width-1)) {
           rel_rank = rel_row+ rel_col;
      } else {
           rel_rank = (2 * (rect[depth].width-1)) - rel_col +
                      (2 * (rect[depth].height-1)) -rel_row;
      }

      grid[(row * WIDTH) + col] = rect[depth].start_count + rel_rank;
    }
  }
```

Perf stat spirale

```
perf stat ./colimacon 10000 10000 >/dev/null

 Performance counter stats for './colimacon 10000 10000':

      30565,377276 task-clock                #    1,000 CPUs utilized
               121 context-switches          #    0,000 M/sec
                17 CPU-migrations            #    0,000 M/sec
            97 798 page-faults               #    0,003 M/sec
    91 337 272 427 cycles                    #    2,988 GHz                      [99,96%]
             0 stalled-cycles-frontend   #    0,00% frontend cycles idle    [99,97%]
             0 stalled-cycles-backend    #    0,00% backend  cycles idle
[99,94%]
    75 156 283 311 instructions              #    0,82  insns per cycle          [99,99%]
    12 978 938 192 branches                  #  424,629 M/sec                    [99,99%]
    75 175 958 781 branch-misses             #  579,22% of all branches
[99,96%]

      30,575889104 seconds time elapsed
```

Perf stat linéaire

```
perf stat ./colimacon 10000 10000 >/dev/null

 Performance counter stats for './colimacon 10000 10000':

      29896,864356 task-clock                #    1,000 CPUs utilized
               116 context-switches          #    0,000 M/sec
                 8 CPU-migrations            #    0,000 M/sec
            97 815 page-faults               #    0,003 M/sec
    81 976 270 972 cycles                    #    2,742 GHz                      [99,96%]
                 0 stalled-cycles-frontend   #    0,00% frontend cycles idle     [99,97%]
                 0 stalled-cycles-backend    #    0,00% backend  cycles idle
[99,97%]
    81 959 426 414 instructions              #    1,00  insns per cycle          [99,98%]
    13 252 574 828 branches                  #  443,276 M/sec                    [99,98%]
    81 981 350 687 branch-misses             #  618,61% of all branches
[99,95%]

      29,906344878 seconds time elapsed
```