Fachbereich
Angewandte Informatik

Hochschule Fulda
University of Applied Sciences

Dokumentationsvorlage für Parallelverarbeitung

Vorgelegt von: Sebastian Bittner 928895
Jürgen Göbel 822602
Mathias Jäger 930208

:

Inhaltsverzeichnis

1 Josephs Abschnitt

Lorem ipsum dolor [LK73] sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1.1 Josephs Unterabschnitt

```
1  /* This program computes a fibonacci sequence in two different
2   * approaches: iterative and recursive. It also determines the
3   * computing time for each approach.
4   * The program computes the sums for fibonacci numbers 'n' in the range
5   * 35 <= n <= 45. It prints the fibonacci solutions along with the
6   * determined computing times in standard output.
7   *
8   * File Name: exercise1-2.c         Author: Juergen Goebel
9   * Date: 09.04.2014
10  */
11
12 #include <stdlib.h>
13 #include <stdio.h>
14 #include <time.h>
15
16 #define F0 0                              /* first step of fib. seq.     */
17 #define F1 1                              /* second step of fib. seq.    */
18 #define MIN 35                            /* bottom limit of input range*/
19 #define MAX 45                            /* top limit of input range    */
20
21 struct Clocks
22 {
23   clock_t start_CPU,
24           end_CPU,
25           total_CPU;
26
27   time_t  start_Wall,
28           end_Wall,
29           total_Wall;
30 };
31
32 long long fibonacci_iterative (int);
33 long long fibonacci_recursive (int);
34 void print_time (struct Clocks);
35
36 int main (void)
37 {
38   long long result;
39   struct Clocks iter,
40                 recur;
41   int i;
42
43   /* Start block for Iterative Fibonacci */
44   printf ("Iterative Fibonacci\n====\n");
```

```
45
46    iter.start_CPU = clock ();
47    iter.start_Wall = time (NULL);
48
49    /* computes and prints all sums in
50     * range with MIN and MAX boundaries
51     */
52    for (i = MIN; i <= MAX; i++)
53    {
54      result = fibonacci_iterative (i);
55      printf ("%i: %lld\n", i, result);
56    }
57
58    iter.end_CPU = clock ();
59    iter.end_Wall = time (NULL);
60    iter.total_CPU = iter.end_CPU -
61                     iter.start_CPU;
62    iter.total_Wall = iter.end_Wall -
63                      iter.start_Wall;
64
65    print_time (iter);
66    /* End block for Iterative Fibonacci */
67
68    /* Start block for Recursive Fibonacci */
69    printf ("Recursive Fibonacci\n====\n");
70    recur.start_CPU = clock ();
71    recur.start_Wall = time (NULL);
72
73    for (i = MIN; i <= MAX; i++)
74    {
75      result = fibonacci_recursive (i);
76      printf ("%i: %lld\n", i, result);
77    }
78
79    recur.end_CPU = clock ();
80    recur.end_Wall = time (NULL);
81    recur.total_CPU = recur.end_CPU -
82                      recur.start_CPU;
83    recur.total_Wall = recur.end_Wall -
84                       recur.start_Wall;
85
86    print_time (recur);
87    /* End block of Recursive Fibonacci */
88
89    return EXIT_SUCCESS;
90  }
91
92  long long fibonacci_iterative (int n)
93  {
94    long long first,
95              second,
96              next;
97    int i;
98    first = F0;
99    second = F1;
100   next = 0;
```

```
101
102   for (i = F1; i <= n; i++)
103   {
104     if (i == F1)
105       next = i;
106     else
107     {
108       next = first + second;
109       first = second;
110       second = next;
111     }
112   }
113
114   return next;
115 }
116
117 long long fibonacci_recursive (int n)
118 {
119   switch (n)
120   {
121     case F0:
122       return F0;
123       break;
124     case F1:
125       return F1;
126       break;
127     default:
128       return
129         (fibonacci_recursive (n - 1) +
130          fibonacci_recursive (n - 2));
131       break;
132   }
133 }
134
135 void print_time (struct Clocks generic)
136 {
137   printf ("\nCPU clock\n");
138   printf ("START: \t%ld\n",
139             generic.start_CPU);
140   printf ("END: \t%ld\n",
141             generic.end_CPU);
142   printf ("TOTAL: \t%ld\n",
143             generic.total_CPU);
144   printf ("TOTAL IN SEC: \t%ld\n",
145             (generic.total_CPU /
146               CLOCKS_PER_SEC));
147
148   printf ("\nWall clock\n");
149   printf ("START: \t%ld\n",
150             generic.start_Wall);
151   printf ("END: \t%ld\n",
152             generic.end_Wall);
153   printf ("TOTAL: \t%ld\n",
154             generic.total_Wall);
155   printf ("\n\n");
```

```
156 || }
```

Listing 1: Quellcode von Aufgabe 1-2

Literaturverzeichnis

[LK73] Shen Lin und Brian W. Kernighan. „An Effective Heuristic Algorithm for the Travelling-Salesman Problem“. In: *Operations Research* 21 (1973), S. 498–516.