src/sequential.c

```c
#include <stdio.h>
#include <stdlib.h>

#define DEBUG 1
#define ARRAY_SIZE 40

void bs(int n, int * vetor)
{
    int c=0, d, troca, trocou =1;

    while ((c < (n-1)) & trocou )
        {
        trocou = 0;
        for (d = 0 ; d < n - c - 1; d++)
            if (vetor[d] > vetor[d+1])
                {
                troca      = vetor[d];
                vetor[d]   = vetor[d+1];
                vetor[d+1] = troca;
                trocou = 1;
                }
        c++;
        }
}

int main()
{
    int vetor[ARRAY_SIZE];
    int i;

    for (i=0 ; i<ARRAY_SIZE; i++)              /* init array with worst
       case for sorting */
        vetor[i] = ARRAY_SIZE-i;


    #ifdef DEBUG
    printf("\nVetor: ");
    for (i=0 ; i<ARRAY_SIZE; i++)              /* print unsorted array */
        printf("[%03d] ", vetor[i]);
    #endif

    bs(ARRAY_SIZE, vetor);                     /* sort array */

    #ifdef DEBUG
    printf("\nVetor: ");
    for (i=0 ; i<ARRAY_SIZE; i++)              /* print sorted array */
        printf("[%03d] ", vetor[i]);
    #endif

    return 0;
}
```

src/parallel.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define DEBUG 1
#define ARRAY_SIZE 100000

void
bs (int n, int* vetor)
{
  int c = 0;
  int d;
  int troca;
  int trocou = 1;

  while ((c < (n-1)) & trocou )
    {
      trocou = 0;
      for (d = 0 ; d < n - c - 1; d++)
    if (vetor[d] > vetor[d+1])
      {
        troca      = vetor[d];
        vetor[d]   = vetor[d+1];
        vetor[d+1] = troca;
        trocou = 1;
      }
      c++;
    }
}

int*
interleaving (int vetor[], int tam)
{
  int* vetor_auxiliar;
  int i1;
  int i2;
  int i_aux;

  vetor_auxiliar = malloc(tam * sizeof(int));

  i1 = 0;
  i2 = tam / 2;

  for (i_aux = 0; i_aux < tam; i_aux++) {
    if (((vetor[i1] <= vetor[i2]) && (i1 < (tam / 2))) || (i2 == tam))
      vetor_auxiliar[i_aux] = vetor[i1++];
    else
      vetor_auxiliar[i_aux] = vetor[i2++];
  }

  return vetor_auxiliar;
}
```

```
53
54  int
55  print_vec (int* vec, int size)
56  {
57    int i;
58
59    printf("[ ");
60    for (i = 0; i < size; i++)
61      printf("%d ", vec[i] );
62    printf("]\n");
63    return 0;
64  }
65
66  int
67  parent (int my_rank)
68  {
69    return (my_rank - 1) / 2;
70  }
71
72  int
73  left_child (int my_rank)
74  {
75    return 2 * my_rank + 1;
76  }
77
78  int
79  right_child (int my_rank)
80  {
81    return 2 * my_rank + 2;
82  }
83
84  int
85  root (void)
86  {
87    double t1,t2;
88    t1 = MPI_Wtime();
89
90    int i;
91    int j;
92    int vec[ARRAY_SIZE];
93
94    // Populate the vector
95    for (i = 0, j = ARRAY_SIZE - 1; i < ARRAY_SIZE; i++, j--)
96      vec[i] = j;
97
98    // MPI stuff
99    int proc_n;
100   int my_rank;
101   MPI_Status status;
102   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
103   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
104
105   // Set delta
106   int delta = ARRAY_SIZE / ((proc_n + 1) / 2);
```

```
107
108 #ifdef DEBUG
109   printf("Vector size: %d\n", ARRAY_SIZE);
110   printf("Delta: %d\n", delta);
111 #endif
112
113   if (ARRAY_SIZE <= delta)
114     {
115       bs(ARRAY_SIZE, vec);
116 #ifdef DEBUG
117       print_vec(vec, ARRAY_SIZE);
118 #endif
119     }
120   else
121     {
122       int size = ARRAY_SIZE / 2;
123
124       // Sending message to the children
125       MPI_Send(&size,    1, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
126       MPI_Send(  vec, size, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
127
128       MPI_Send(     &size,    1, MPI_INT, right_child(my_rank), 1,
              MPI_COMM_WORLD);
129       MPI_Send(vec + size, size, MPI_INT, right_child(my_rank), 1,
              MPI_COMM_WORLD);
130
131       // Receiving message from the children
132       MPI_Recv(       vec, size, MPI_INT,  left_child(my_rank),
              MPI_ANY_TAG, MPI_COMM_WORLD, &status);
133       MPI_Recv(vec + size, size, MPI_INT, right_child(my_rank),
              MPI_ANY_TAG, MPI_COMM_WORLD, &status);
134
135       int* ans = interleaving(vec, ARRAY_SIZE);
136
137 #ifdef DEBUG
138       print_vec(ans, ARRAY_SIZE);
139 #endif
140
141       free(ans);
142     }
143
144   t2 = MPI_Wtime();
145   fprintf(stderr, "Time: %fs\n\n", t2-t1);
146
147   return 0;
148 }
149
150 int
151 child (void)
152 {
153   // MPI stuff
154   int proc_n;
```

```
155   int my_rank;
156   MPI_Status status;
157   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
158   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
159
160   int size;
161   MPI_Recv(&size, 1, MPI_INT, parent(my_rank), MPI_ANY_TAG, MPI_COMM_WORLD
          , &status);
162
163   int* vec = malloc(size * sizeof(int));
164
165   if (!vec)
166     return EXIT_FAILURE;
167
168   MPI_Recv(vec, size, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD
          , &status);
169
170 #ifdef DEBUG
171   printf("My rank is: %d and my vec size is: %d\n", my_rank, size);
172 #endif
173
174   // Set delta
175   int delta = ARRAY_SIZE / ((proc_n + 1) / 2);
176
177   if (size <= delta)
178     {
179       bs(size, vec);
180       MPI_Send(vec, size, MPI_INT, parent(my_rank), 1, MPI_COMM_WORLD);
181     }
182   else
183     {
184       int child_size = size / 2;
185
186       // Sending message to the children
187       MPI_Send(&child_size,          1, MPI_INT, left_child(my_rank), 1,
             MPI_COMM_WORLD);
188       MPI_Send(        vec, child_size, MPI_INT, left_child(my_rank), 1,
             MPI_COMM_WORLD);
189
190       MPI_Send(     &child_size,          1, MPI_INT, right_child(my_rank)
             , 1, MPI_COMM_WORLD);
191       MPI_Send(vec + child_size, child_size, MPI_INT, right_child(my_rank)
             , 1, MPI_COMM_WORLD);
192
193       // Receiving message from the children
194       MPI_Recv(             vec, child_size, MPI_INT,  left_child(my_rank)
             , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
195       MPI_Recv(vec + child_size, child_size, MPI_INT, right_child(my_rank)
             , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
196
197       int* ans = interleaving(vec, size);
198
199       MPI_Send(ans, size, MPI_INT, parent(my_rank), 1, MPI_COMM_WORLD);
200
```

```
201         free(ans);
202       }
203
204     free(vec);
205
206     return 0;
207   }
208
209   int
210   main (int argc, char** argv)
211   {
212     int my_rank;
213     int proc_n;
214
215     MPI_Init(&argc, &argv);
216
217     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
218     MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
219
220     if (my_rank == 0)
221       root();
222     else
223       child();
224
225     MPI_Finalize();
226
227     return 0;
228   }
```

src/optimized.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4
5  #define DEBUG 1
6  #define ARRAY_SIZE 100000
7  #define DELTA(proc_n) (ARRAY_SIZE / proc_n)
8  #define VALID(i, lim_i) (i < lim_i)
9  #define HOI(array, i, lim_i, next_i) (array[i] > array[next_i] || !VALID(i
       , lim_i))
10
11 void
12 bs (int n, int* vetor)
13 {
14   int c = 0;
15   int d;
16   int troca;
17   int trocou = 1;
18
19   while ((c < (n-1)) & trocou )
20     {
21       trocou = 0;
22       for (d = 0 ; d < n - c - 1; d++)
23     if (vetor[d] > vetor[d+1])
24       {
25         troca      = vetor[d];
26         vetor[d]   = vetor[d+1];
27         vetor[d+1] = troca;
28         trocou     = 1;
29       }
30       c++;
31     }
32 }
33
34 int*
35 interleaving (int vetor[], int tam, int delta)
36 {
37   int* vetor_auxiliar;
38   int i1;
39   int lim_i1;
40   int i2;
41   int lim_i2;
42   int i3;
43   int i_aux;
44
45   vetor_auxiliar = malloc(tam * sizeof(int));
46
47   int child_size = (tam - delta) / 2;
48
49   i1     = 0;
50   lim_i1 = child_size;
51   i2     = child_size;
```

```
52    lim_i2 = child_size * 2;
53    i3     = child_size * 2;
54
55    for (i_aux = 0; i_aux < tam; i_aux++) {
56      if ((VALID(i1, lim_i1)) && HOI(vetor, i2, lim_i2, i1) &&  HOI(vetor,
          i3, tam, i1))
57        {
58      vetor_auxiliar[i_aux] = vetor[i1++];
59        }
60      else
61        {
62      if (VALID(i2, lim_i2) &&  HOI(vetor, i3, tam, i2))
63        {
64          vetor_auxiliar[i_aux] = vetor[i2++];
65        }
66      else
67        {
68          vetor_auxiliar[i_aux] = vetor[i3++];
69        }
70        }
71    }
72
73    return vetor_auxiliar;
74  }
75
76  int
77  print_vec (int* vec, int size)
78  {
79    int i;
80
81    printf("[ ");
82    for (i = 0; i < size; i++)
83      printf("%d ", vec[i] );
84    printf("]\n");
85    return 0;
86  }
87
88  int
89  parent (int my_rank)
90  {
91    return (my_rank - 1) / 2;
92  }
93
94  int
95  left_child (int my_rank)
96  {
97    return 2 * my_rank + 1;
98  }
99
100 int
101 right_child (int my_rank)
102 {
103   return 2 * my_rank + 2;
104 }
```

```
105
106 int
107 root (void)
108 {
109   double t1,t2;
110   t1 = MPI_Wtime();
111
112   int i;
113   int j;
114   int vec[ARRAY_SIZE];
115
116   // Populate the vector
117   for (i = 0, j = ARRAY_SIZE - 1; i < ARRAY_SIZE; i++, j--)
118     vec[i] = j;
119
120   // MPI stuff
121   int proc_n;
122   int my_rank;
123   MPI_Status status;
124   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
125   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
126
127   // Set delta
128   int delta = DELTA(proc_n);
129
130 #ifdef DEBUG
131   printf("Vector size: %d\n", ARRAY_SIZE);
132   printf("Delta: %d\n", delta);
133   printf("My rank is: %d and my vec size is: %d\n", my_rank, ARRAY_SIZE);
134 #endif
135
136   if (ARRAY_SIZE < 2 * delta)
137     {
138       bs(ARRAY_SIZE, vec);
139 #ifdef DEBUG
140       print_vec(vec, ARRAY_SIZE);
141 #endif
142     }
143   else
144     {
145       int child_size = (ARRAY_SIZE - delta) / 2;
146
147       // Sending message to the children
148       MPI_Send(&child_size,    1, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
149       MPI_Send(  vec, child_size, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
150
151       MPI_Send(     &child_size,          1, MPI_INT, right_child(my_rank)
              , 1, MPI_COMM_WORLD);
152       MPI_Send(vec + child_size, child_size, MPI_INT, right_child(my_rank)
              , 1, MPI_COMM_WORLD);
153
154       bs(ARRAY_SIZE - 2 * child_size, vec + 2 * child_size);
```

```
155
156         // Receiving message from the children
157         MPI_Recv(              vec, child_size, MPI_INT,  left_child(my_rank)
                , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
158         MPI_Recv(vec + child_size, child_size, MPI_INT, right_child(my_rank)
                , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
159
160 #ifdef DEBUG
161         printf("Rank %d: Before interleaving\n", my_rank);
162         print_vec(vec, ARRAY_SIZE);
163 #endif
164
165         int* ans = interleaving(vec, ARRAY_SIZE, delta);
166
167 #ifdef DEBUG
168         printf("Rank %d: After interleaving\n", my_rank);
169         print_vec(ans, ARRAY_SIZE);
170 #endif
171
172         free(ans);
173       }
174
175   t2 = MPI_Wtime();
176   fprintf(stderr, "Time: %fs\n\n", t2-t1);
177
178   return 0;
179 }
180
181 int
182 child (void)
183 {
184   // MPI stuff
185   int proc_n;
186   int my_rank;
187   MPI_Status status;
188   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
189   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
190
191   int size;
192   MPI_Recv(&size, 1, MPI_INT, parent(my_rank), MPI_ANY_TAG, MPI_COMM_WORLD
          , &status);
193
194   int* vec = malloc(size * sizeof(int));
195
196   if (!vec)
197     return EXIT_FAILURE;
198
199   MPI_Recv(vec, size, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD
          , &status);
200
201 #ifdef DEBUG
202   printf("My rank is: %d and my vec size is: %d\n", my_rank, size);
203 #endif
204
```

```
205   // Set delta
206   int delta = DELTA(proc_n);
207
208   if (size <= 2 * delta)
209     {
210       bs(size, vec);
211       MPI_Send(vec, size, MPI_INT, parent(my_rank), 1, MPI_COMM_WORLD);
212     }
213   else
214     {
215       int child_size = (size - delta) / 2;
216
217       // Sending message to the children
218       MPI_Send(&child_size,          1, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
219       MPI_Send(        vec, child_size, MPI_INT, left_child(my_rank), 1,
              MPI_COMM_WORLD);
220
221       MPI_Send(     &child_size,          1, MPI_INT, right_child(my_rank)
              , 1, MPI_COMM_WORLD);
222       MPI_Send(vec + child_size, child_size, MPI_INT, right_child(my_rank)
              , 1, MPI_COMM_WORLD);
223
224       bs(size - 2 * child_size, vec + 2 * child_size);
225
226       // Receiving message from the children
227       MPI_Recv(             vec, child_size, MPI_INT,  left_child(my_rank)
              , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
228       MPI_Recv(vec + child_size, child_size, MPI_INT, right_child(my_rank)
              , MPI_ANY_TAG, MPI_COMM_WORLD, &status);
229
230 #ifdef DEBUG
231       printf("Rank %d: Before interleaving\n", my_rank);
232       print_vec(vec, size);
233 #endif
234
235       int* ans = interleaving(vec, size, delta);
236
237 #ifdef DEBUG
238       printf("Rank %d: After interleaving\n", my_rank);
239       print_vec(vec, size);
240 #endif
241
242       MPI_Send(ans, size, MPI_INT, parent(my_rank), 1, MPI_COMM_WORLD);
243
244       free(ans);
245     }
246
247   free(vec);
248
249   return 0;
250 }
251
252 int
```

```
253 main (int argc, char** argv)
254 {
255   int my_rank;
256   int proc_n;
257
258   MPI_Init(&argc, &argv);
259
260   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
261   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
262
263   if (my_rank == 0)
264     root();
265   else
266     child();
267
268   MPI_Finalize();
269
270   return 0;
271 }
```