Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – Programação OMP

Fernando W Cruz

Soma básica... Posso paralelizar?

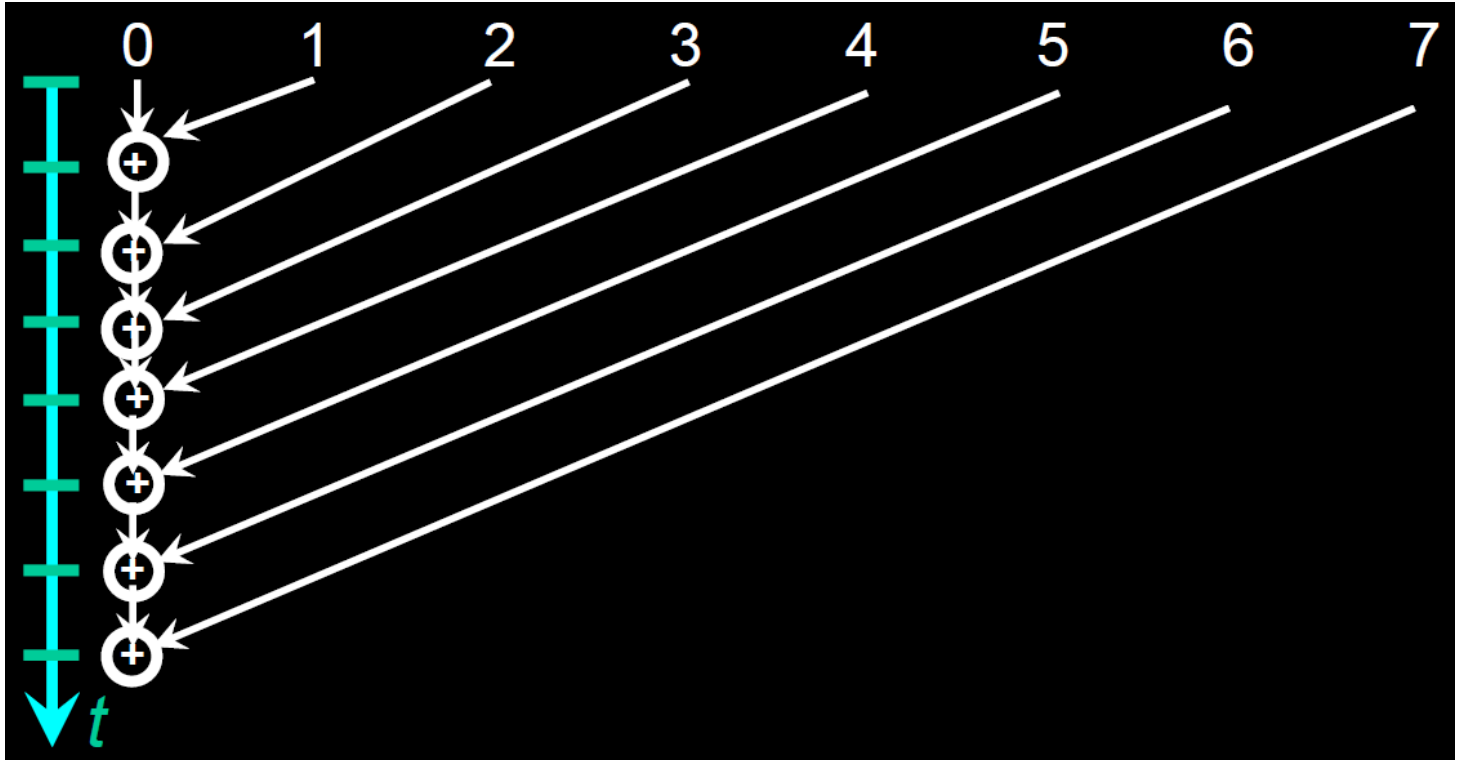
- Dados N inteiros, Ai, i=0, ..., N-1, onde N é uma potência de 2, calcule

$$S = \sum_{i=0}^{n-1} A_i$$

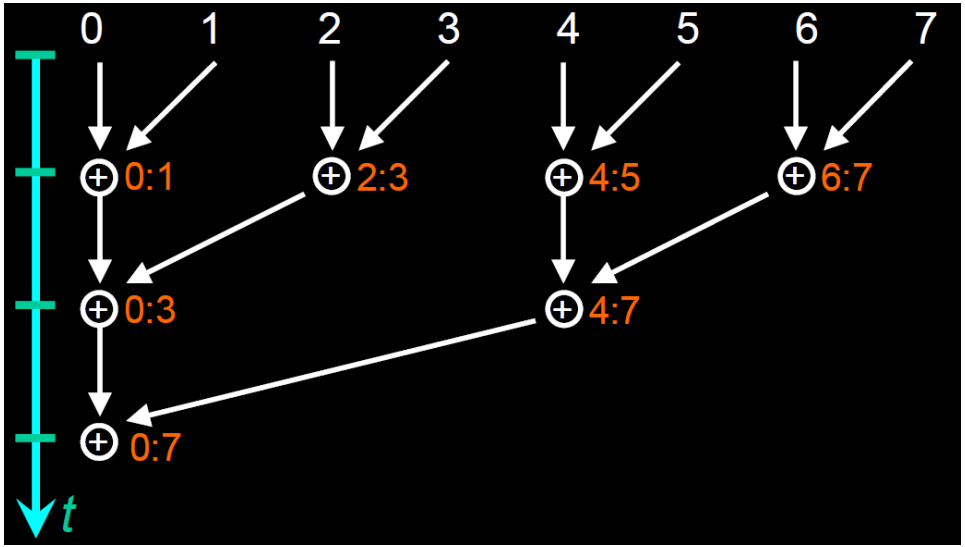
- Algoritmo usual (não é paralelizável, pois uma iteração depende do resultado da iteração anterior (recursão)

```
s = 0;
for (i = 0; i<n; i++)
   s = s + A[i];
```

Soma sequencial (supondo N = 8 =>(2^3))

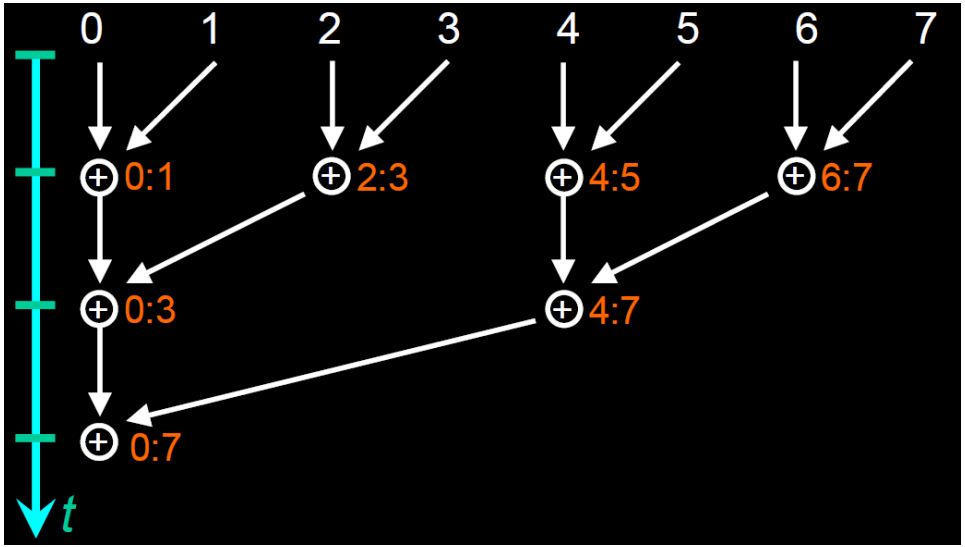


Soma paralela (“*cascade sum*” ou “*recursive doubling*”)



- Mesma quantidade de contas do algoritmo sequencial
- Complexidade paralela, com n processadores: O(log n)
- Todas as iterações de cada instância do laço são independentes
- Cada iteração do laço em shift é dependente da anterior

Soma paralela (“*cascade sum*” ou “*recursive doubling*”)




```
Codificação:
    for (shift=1; shift<n; shift=2*shift) {
        for (i=0; i<n; i+=2*shift)
            a[i] = a[i] + a[i+shift];
    }
```

- Mesma quantidade de contas do algoritmo sequencial
- Complexidade paralela, com n processadores: O(log n)
- Cada iteração do laço em shift é dependente da anterior
- Todas as iterações de cada instância do laço são independentes

Soma paralela

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <omp.h>
4
5   int main(int argc, char *argv[]) {
6       long long N=atoi(argv[1]);
7       int *v = (int *) malloc(sizeof(int)*N);
8
9       for (int i=0;i<N;i++)
10          v[i]=1;
11
12
13      for (int shift=1; shift<N; shift*=2)
14
15          for (int i=0; i<N; i=2*shift+i)
16              v[i]+=v[i+shift];
17
18      printf("soma = %d\n", v[0]);
19      return 0;
20  } /*fim-main */
```

Soma paralela

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <omp.h>
4
5   int main(int argc, char *argv[]) {
6       long long N=atoi(argv[1]);
7       int *v = (int *) malloc(sizeof(int)*N);
8
9       for (int i=0;i<N;i++)
10          v[i]=1;
11
12
13      for (int shift=1; shift<N; shift*=2)
14
15          for (int i=0; i<N; i=2*shift+i)
16              v[i]+=v[i+shift];
17
18      printf("soma = %d\n", v[0]);
19      return 0;
20  } /*fim-main */
```

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <omp.h>
4
5   int main(int argc, char *argv[]) {
6       long long N=atoi(argv[1]);
7       int *v = (int *) malloc(sizeof(int)*N);
8
9       for (int i=0;i<N;i++)
10          v[i]=1;
11      #pragma omp parallel
12      {
13      for (int shift=1; shift<N; shift*=2)
14          #pragma omp for
15          for (int i=0; i<N; i=2*shift+i)
16              v[i]+=v[i+shift];
17      }
18      printf("soma = %d\n", v[0]);
19      return 0;
20  } /*fim-main */
```

Cada thread executa todas as iterações

Iterações independentes divididas entre as threads

Paralelização de algoritmos de ordenação

- Exercício: Paralelizar o algoritmo *bubble sort*, considerando a seguinte lógica:

```
void bubble (int a[], int n) {
                int i, j;
                for (i=n-1; i>0; i--) {
                  for (j=0; j<i; j++) {
                    ce(&a[j], &a[j+1]);
                  }
                }
              }
```

```
void ce(int *a, int *b) {
  int t;
  if (*a > *b) {t=*a; *a=*b; *b=t;}
 }
```

Paralelização de algoritmos de ordenação

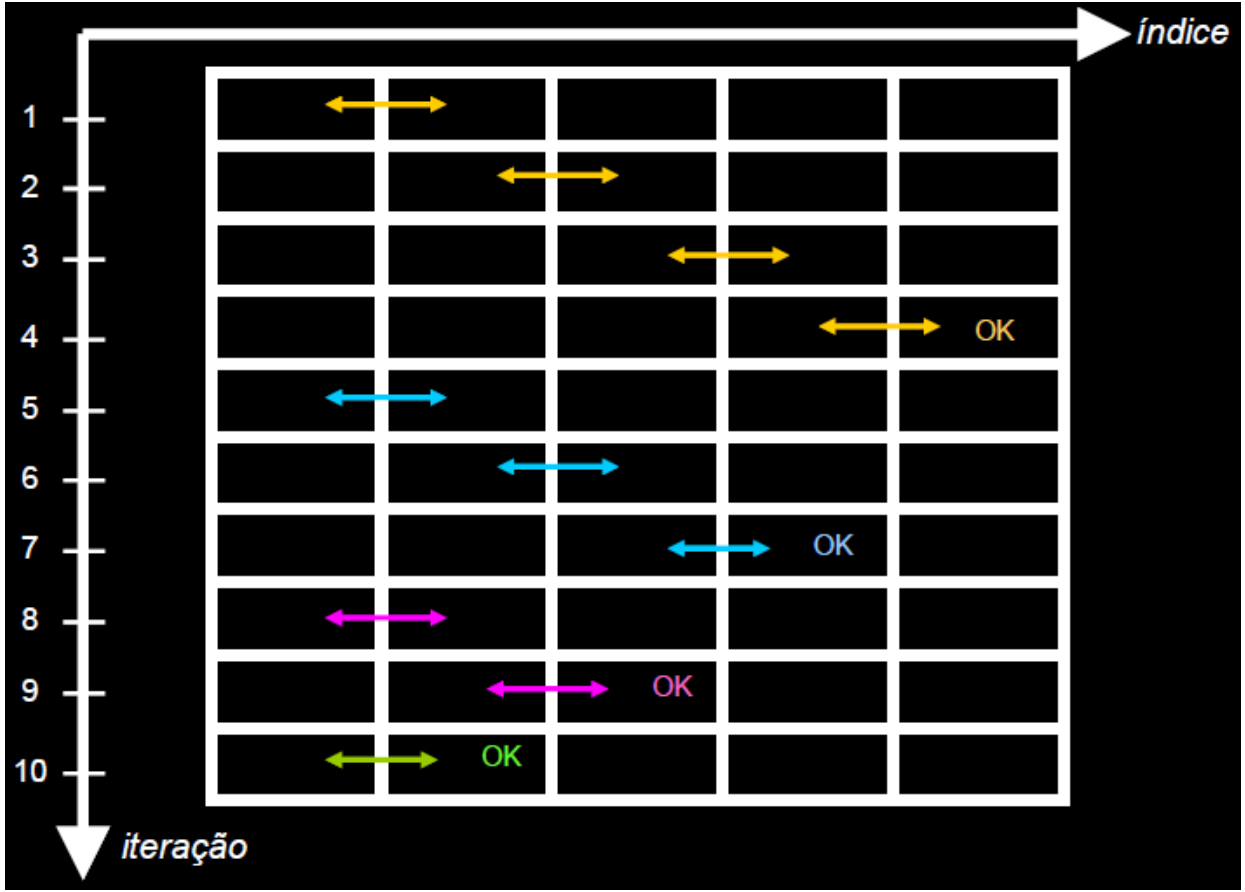
- Perceba que o algoritmo é inerentemente sequencial
- As iterações dos laços interno e externo são dependentes

Paralelização de algoritmos de ordenação

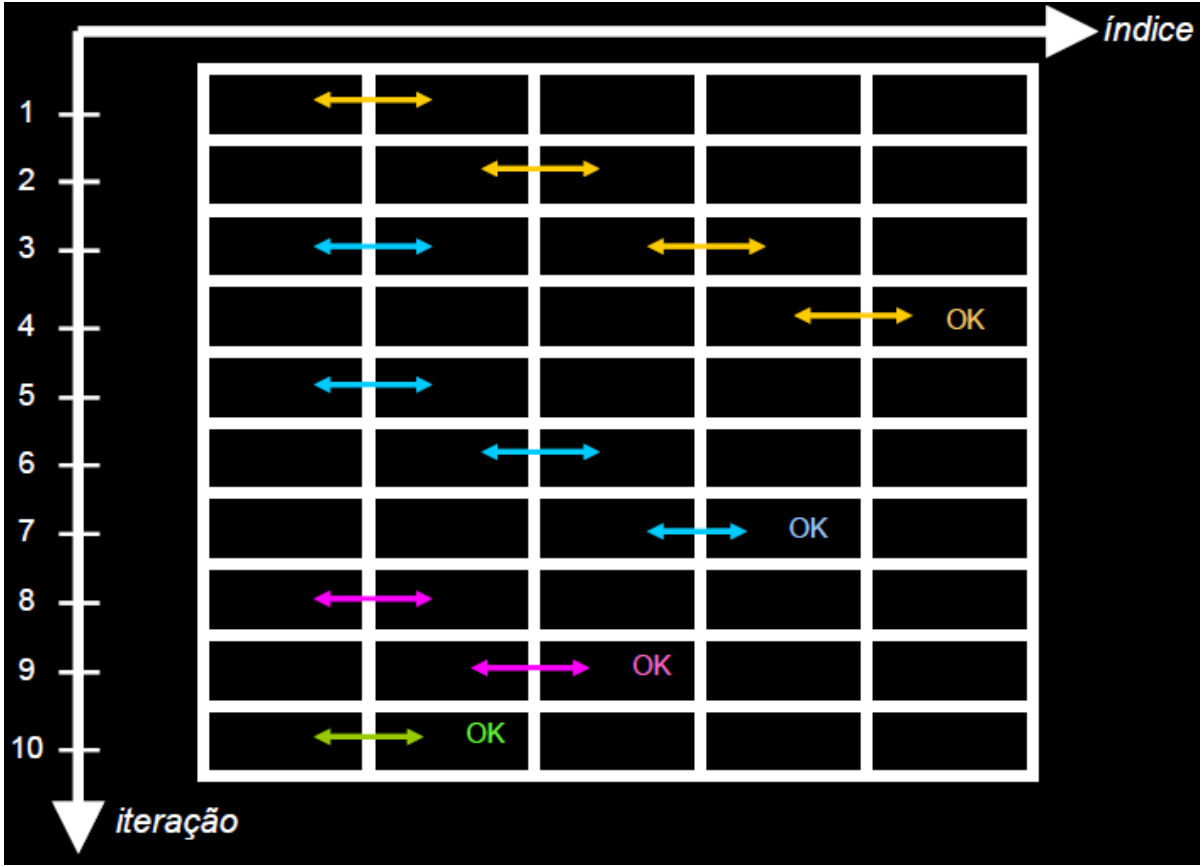
- Perceba que o algoritmo é inerentemente sequencial
- As iterações dos laços interno e externo são dependentes

```
5   void comparaTroca(int *a, int *b) {
6       int t;
7       if (*a > *b){
8           t = *a; *a = *b; *b =  t;
9       } /*fim-if */
10  } /* fim-comparaTroca */
11
12  void bubbleSort(int a[], int n) {
13      int i, j;
14      for (int i=n-1; i>0; i--)
15          for (int j=0; j<i; j++)
16              comparaTroca(&a[j], &a[j+1]);
17  } /* fim-bubleSort */
18
19  int main(int argc, char *argv[]) {
20      int N=atoi(argv[1]);
21      int a[N];
22      for (int i=0;i<N;i++)
23          a[i]=rand()/10000;
24      bubbleSort(a, N);
25      for (int i=0;i<N;i++)
26          printf("%d\n", a[i]);
27      return 0;
28  } /* fim-main */
```

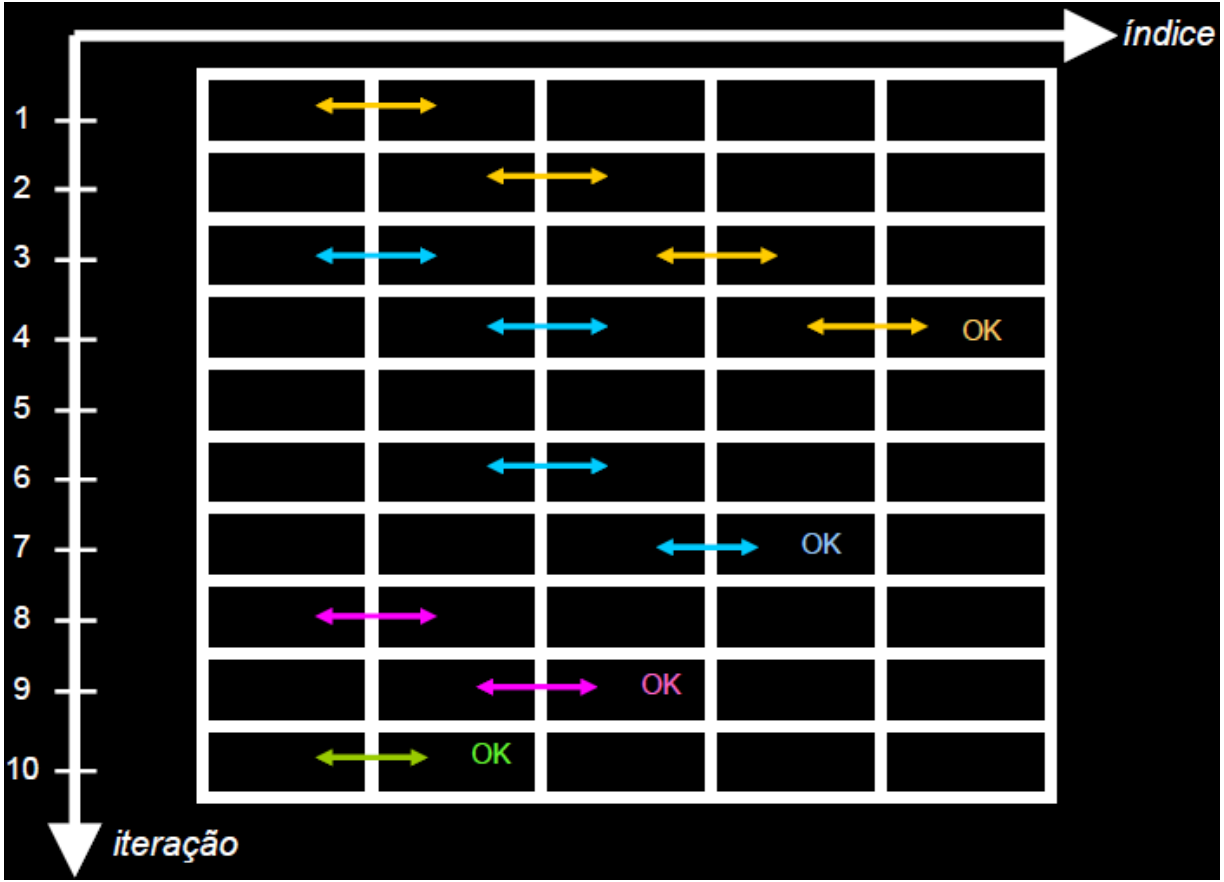
Fluxo do *bubble sort*



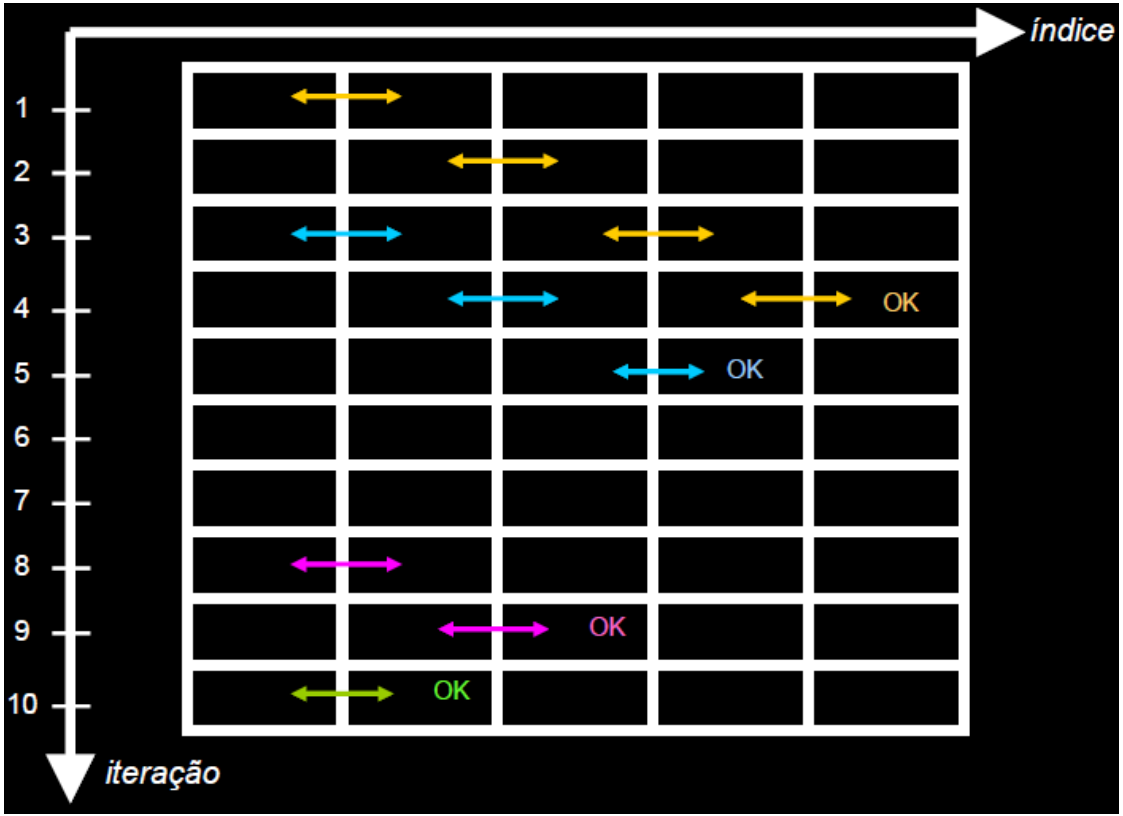
Solução *OddEven transposition sort*



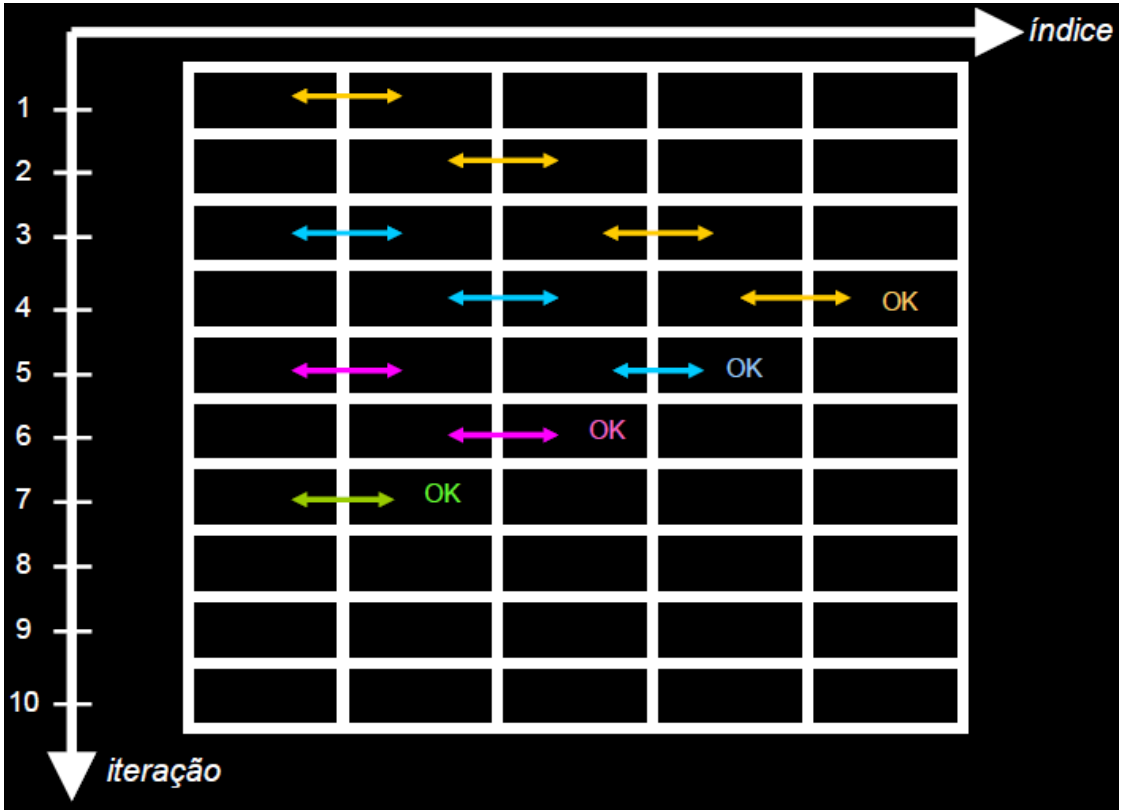
Solução *OddEven transposition sort*



Solução *OddEven transposition sort*



Solução *OddEven transposition sort*



Paralelização de algoritmos de ordenação

```
5   void comparaTroca(int *a, int *b) {
6       int t;
7
8       if (*a> *b) {
9           t=*a; *a = *b; *b = t;
10      } /*fim-if */
11  } /* fim-comparaTroca */
12
13  void ParImpar(int a[], int n) {
14      int i, j, nHalf, ultPar, ultImpar;
15
16      nHalf=n/2; ultPar=nHalf-1;
17      if ((n%2) == 0)
18          ultImpar = nHalf-2;
19      else
20          ultImpar = nHalf-1;
21      for (int i=0; i<n-1; i++) {
22
23          for (int j=0; j<=ultImpar; j++)
24              comparaTroca(&a[2*j+1], &a[2*j+2]);
25
26          for (int j=0; j<=ultPar; j++)
27              comparaTroca(&a[2*j], &a[2*j+1]);
28      } /*fim-for */
29
30  } /* fim-ParImpar */
```

```
32  int main(int argc, char *argv[]) {
33      int N=atoi(argv[1]);
34      int a[N];
35
36      for (int i=0;i<N;i++)
37          a[i]=rand()/10000;
38      ParImpar(a, N);
39      for (int i=0;i<N;i++)
40          printf("%d\n", a[i]);
41      return 0;
42  } /* fim-main */
```

Solução *OddEven transposition sort*

```
 5  void comparaTroca(int *a, int *b) {
 6      int t;
 7
 8      if (*a> *b) {
 9          t=*a; *a = *b; *b = t;
10      } /*fim-if */
11  } /* fim-comparaTroca */
12
13  void ParImpar(int a[], int n) {
14      int i, j, nHalf, ultPar, ultImpar;
15
16      nHalf=n/2; ultPar=nHalf-1;
17      if ((n%2) == 0)
18          ultImpar = nHalf-2;
19      else
20          ultImpar = nHalf-1;
21      #pragma omp parallel
22      {
23          for (int i=0; i<n-1; i++) {
24          #pragma omp for
25          for (int j=0; j<=ultImpar; j++)
26              comparaTroca(&a[2*j+1], &a[2*j+2]);
27          #pragma omp for
28          for (int j=0; j<=ultPar; j++)
29              comparaTroca(&a[2*j], &a[2*j+1]);
30          } /*fim-for */
31      } /*fim-pragma */
32  } /* fim-ParImpar */
```

```
34  int main(int argc, char *argv[]) {
35      int N=atoi(argv[1]);
36      int a[N];
37
38      for (int i=0;i<N;i++)
39          a[i]=rand()/10000;
40      ParImpar(a, N);
41      for (int i=0;i<N;i++)
42          printf("%d\n", a[i]);
43      return 0;
44  } /* fim-main */
45
```

Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – OMP

Fernando W Cruz