Nome: Leonardo Rothier Soares Cardoso

# Tarefa 15 – Análise de Complexidade

## Problema escolhido:

```
[BUBLESORT]
for(i = 0; i < n-1; i++)
  for(j = 0; j < n-i-1; j++)
    if(arr[j] > arr[j+1])
      swap(&arr[j], &arr[j+1]);
```

### RAM

Operação relevante: nº de movimentações com os elementos do vetor (swap – 3 operações)

Pior caso: vetor está em ordem decrescente

T(n) = $\sum_{i=0}^{n-2}\sum_{j=0}^{n-i-2} 3$

O($n^2$)

### PRAM

Paralelizando o for externo

Teremos executando em paralelo até então ( um núcleo para cada coluna):

| 1-> J = 0 | 2-> J = 0 | 3-> J = 0 | ... | n-2-> J = 0 |
|---|---|---|---|---|
| If(arr[0] > arr[1]) swap(&arr[0], &arr[1]); | If(arr[0] > arr[1]) swap(&arr[0], &arr[1]); | If(arr[0] > arr[1]) swap(&arr[0], &arr[1]); | | If(arr[0] > arr[1]) swap(&arr[0], &arr[1]); |

Exemplo : Arranjo[] = {8,7,6,5,4,3,2}

step

0 8 <-> 7 8 <-> 7 8 <-> 7 ... 8 <-> 7 (não faria nem sentido ne) .

Exploraremos então um paralelização só interna

| 1-> J = 0 | 2-> J = 1 | 3-> J = 2 | ... | n-i-1-> J =n-i-2 |
|---|---|---|---|---|
| If(arr[0] > arr[1]) swap(&arr[0], &arr[1]); | If(arr[1] > arr[2]) swap(&arr[1], &arr[2]); | If(arr[2] > arr[3]) swap(&arr[2], &arr[3]); | | If(arr[n-i-2] > arr[n-i-1]) swap(&arr[n-i-2], &arr[n-i-1]); |

Exemplo : Arranjo[] = {8,7,6,5,4,3,2}

step(time)

0 8 <-> 7 <-> 6 <-> 5 <-> 4 <-> 3 <-> 2

Ainda também não daria certo, pois eles estariam  fazendo operações(swap) ao mesmo tempo em uma mesma posição do array(a um mesmo elemento do array), e cada operação dessa muda como o array vai estar preenchido e isso influencia no resultado e no desempenho do nosso algoritmo, então há meio que uma dependência aí que eu pude localizar.

Vamos ter que fazer uma mudança no código para resolver esse problema. Onde a ideia ficaria mais ou menos assim:

step

0  8 <-> 7  6 <-> 5  4 <-> 3  2

1  8  7 <-> 6  5 <-> 4  3 <-> 2

Onde que antes a primeira passada levaria n-2, fazemos em apenas 2 step, já uma constante, logo nosso for interno teria uma análise de complexidade de O(1) tendo a nosso dispor claro a máquina PRAM com infinito  processadores, precisaríamos sempre de n/2 processadores, não importando o tamanho de n.

Para isso usaremos a ideia de par e ímpar(mais a ideia de intercalar), de for e paralelizaremos eles separadamente:

**[BUBLESORT]**

```
for (int i = 0; i < n-1; i++) {
    for (int j = 0; j < (n/2); j++) {
        if (array[2*j] > array[2*j + 1]) {
            aux            = array[2*j];
            array[2*j]     = array[2*j + 1];
            array[2*j + 1] = aux;
        }
    }
    for (int j = 0; j < (n/2)-1; j++) {
        if (array[2*j + 1] > array[2*j + 2]) {
            aux            = array[2*j + 1];
            array[2*j + 1] = array[2*j + 2];
            array[2*j + 2] = aux;
        }
    }

}
```

No final ainda teríamos o O(n) do for externo, então não sairíamos do O(n) infelizmente, mas já estaria melhor, mesmo que injustamente, do nosso amiguinho nlogn. E para um algoritmo que era considerado um dos piores para este tipo de aplicação, já está muito bom.

## Extra

Não vou desperdiçar essa chance de ouro, em que consegui destrinchar e em entender a fundo como a paralelização do bubble sort poderia funcionar, pensei, porque não mostrar(não dei mole)

Primeira vez paralelizando, a partir da teoria:

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

int array [100000];
void bubble_sort(){
    int n = sizeof(array)/sizeof(array[0]);
    int aux;
    for (int i = 0; i < n-1; i++) {
        #pragma omp parallel for schedule(dynamic, 1000) shared (n, array
) private(aux)
        for (int j = 0; j < (n/2); j++) {
            if (array[2*j] > array[2*j + 1]) {
                aux            = array[2*j];
                array[2*j]     = array[2*j + 1];
                array[2*j + 1] = aux;
            }
        }
        #pragma omp parallel for schedule(dynamic, 100) shared (n, array)
 private(aux)
        for (int j = 0; j < (n/2)-1; j++) {
            if (array[2*j + 1] > array[2*j + 2]) {
                aux            = array[2*j + 1];
                array[2*j + 1] = array[2*j + 2];
                array[2*j + 2] = aux;
            }
        }

    }
}
void fill_array(){
    int n = sizeof(array)/sizeof(array[0]);
    for(int i = 0; i < n; i++){
        array[i] = n - i; // array decrescente
    }
}
void print_array(){
    int n = sizeof(array)/sizeof(array[0]);
    for(int i = 0; i < n; i++){
        printf("%d ", array[i]);
    }
}
int main(){

    fill_array();
    bubble_sort();
    print_array();
    return 0;
}
```

## Executando

Mesmo minha máquina(carroçinha) não sendo lá nem próxima de uma máquina PRAM conseguimos mesmo assim um desempenho muito bom

Tendo um sequencial aí de:

real 0m24.943s

user 0m24.000s

sys 0m0.031s

e um paralelo :

real 0m5.779s

user 0m44.438s

sys 0m0.188s

com as saídas certinhas, um speedup de aproximadamente 4,3161