Sort Parallel

Henrique Mendonça

Leonardo Viana Won Dollinger

Luiz Junio Veloso Dos Santos

Maria Luisa de Oliveira Rodrigues

QuickSort (sequencial)

```
Menu
1128781@PMG34INFLL21412: ~/Downloads
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
1128781@PMG34INFLL21412:~/Downloads$ time ./quickNP.exe < Random.txt > saida.out

real    0m0,309s
user    0m0,289s
sys     0m0,020s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickNP.exe < Random.txt > saida.out

real    0m0,310s
user    0m0,299s
sys     0m0,008s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickNP.exe < Random.txt > saida.out

real    0m0,305s
user    0m0,279s
sys     0m0,024s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickNP.exe < Random.txt > saida.out

real    0m0,309s
user    0m0,305s
sys     0m0,005s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickNP.exe < Random.txt > saida.out

real    0m0,310s
user    0m0,298s
sys     0m0,012s
1128781@PMG34INFLL21412:~/Downloads$
```

Média de tempo sequencial:

Real: 0m0,308
User: 0m0,294
Sys: 0m0,13

Máquina:

PC DELL PUC-MG

QuickSort (paralelo)

```
1128781@PMG34INFLL21412:~/Downloads$ time ./quickP.exe < Random.txt > saida.out

real    0m0,260s
user    0m0,324s
sys     0m0,008s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickP.exe < Random.txt > saida.out

real    0m0,264s
user    0m0,320s
sys     0m0,016s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickP.exe < Random.txt > saida.out

real    0m0,268s
user    0m0,329s
sys     0m0,013s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickP.exe < Random.txt > saida.out

real    0m0,266s
user    0m0,319s
sys     0m0,020s
1128781@PMG34INFLL21412:~/Downloads$ time ./quickP.exe < Random.txt > saida.out

real    0m0,266s
user    0m0,329s
sys     0m0,008s
```

Média de tempo paralelo:

Real: 0m0,264
User: 0m0,324
Sys: 0m0,13

Melhora de Aprox. 14%

QuickSort

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

int arr[1000000];
int Nthreads = 4, n = 0;

int main() {

    int i;
    omp_set_num_threads(4);
    omp_set_nested(1);

    for(i = 0; i < 1000000; i++){
     scanf ("%d", &arr[i]);
    }
    quicksort();
    for(i = 0; i < 1000000; i++){
     printf ("%d\n", arr[i]);
    }
}
```

```c
void quickSort(int esq, int dir)
{
    int pi;
    n++;
    if (esq < dir)
    {
        pi = particao( esq, dir);

        if(n > 20){
            quickSort(esq, pi - 1);
            quickSort( pi + 1, dir);
        }
        else{
            #pragma omp parallel sections num_threads(Nthreads)
            {
                #pragma omp section
                quickSort(esq, pi - 1);
                #pragma omp section
                quickSort( pi + 1, dir);
            }
        }
    }
}

void quicksort(){
    quickSort(0, 999999);
}
```

QuickSort

```c
/*
    Essa função usa o último elemento como pivô,
    posiciona o elemento pivô na posição correta no
    array ordenado e coloca todos os menores que o
    pivô à esquerda (do pivô) e todos os elementos
    maiores à direita
*/
int particao (int esq, int dir)
{
    int pivo = arr[dir];
    int i = (esq - 1);  //menor elemento

    for (int j = esq; j <= dir- 1; j++)
    {
        if (arr[j] <= pivo)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[dir]);
    return (i + 1);
}
```

Merge Sort (sequencial)

Terminal ter, 30 de out, 22:56

marialuisa@marialuisa: ~/Área de Trabalho/seminarios

Arquivo Editar Ver Pesquisar Terminal Ajuda

```
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./nparalelo.exe<in1milhao.txt>respostanp.out

real    0m0,354s
user    0m0,337s
sys     0m0,017s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./nparalelo.exe<in1milhao.txt>respostanp.out

real    0m0,353s
user    0m0,333s
sys     0m0,021s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./nparalelo.exe<in1milhao.txt>respostanp.out

real    0m0,383s
user    0m0,366s
sys     0m0,017s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./nparalelo.exe<in1milhao.txt>respostanp.out

real    0m0,381s
user    0m0,349s
sys     0m0,032s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./nparalelo.exe<in1milhao.txt>respostanp.out

real    0m0,373s
user    0m0,357s
sys     0m0,016s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$
```

Média de tempo Sequencial:

Real: 0m0,368
User: 0m0,348
Sys: 0m0,21

Máquina:

OS: Ubuntu 18.04

CPU: Intel i5 7200U

Merge Sort (paralelo)

Terminal ter, 30 de out, 22:55

marialuisa@marialuisa: ~/Área de Trabalho/seminarios

Arquivo Editar Ver Pesquisar Terminal Ajuda

```
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./paralelo.exe<in1milhao.txt >respostap.out

real    0m0,265s
user    0m0,512s
sys     0m0,032s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./paralelo.exe<in1milhao.txt >respostap.out

real    0m0,265s
user    0m0,528s
sys     0m0,012s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./paralelo.exe<in1milhao.txt >respostap.out

real    0m0,261s
user    0m0,522s
sys     0m0,012s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./paralelo.exe<in1milhao.txt >respostap.out

real    0m0,269s
user    0m0,546s
sys     0m0,012s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$ time ./paralelo.exe<in1milhao.txt >respostap.out

real    0m0,275s
user    0m0,507s
sys     0m0,040s
marialuisa@marialuisa:~/Área de Trabalho/seminarios$
```

Média de tempo paralelo

Real: 0m0,267
User: 0m0,523
Sys: 0m0,20

Máquina:

OS: Ubuntu 18.04
CPU: Intel i5 7200U
Melhora de 28%

Merge Sort:

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

void mergesort(int a[], int i, int j, int proc);
void merge(int a[], int i1, int j1, int i2, int j2);
int a[1000000];
int Nthreads = 0, n = 0;
```

```c
int main(){
    int i;
    int proc = omp_get_num_procs();
    omp_set_num_threads(proc);
    omp_set_nested(1);

    for(i = 0;i < 1000000; i++){
        scanf("%d",&a[i]);
    }

    mergesort(a, 0, 999999, proc);

    for(i= 0;i < 1000000; i++){
        printf ("%d\n", a[i]);
    }
}
```

Merge Sort:

```c
void merge(int a[], int i1, int j1, int i2, int j2){
    int temp[1000000];
    int i, j, k;
    i = i1;
    j = i2;
    k = 0;

    while(i <= j1 && j <= j2){
        if(a[i] < a[j]){
            temp[k++]=a[i++];
        }
        else{
            temp[k++]=a[j++];
        }
    }
    while(i <= j1){
        temp[k++]=a[i++];
    }
    while(j <= j2){
        temp[k++]=a[j++];
    }

    for(i = i1, j = 0; i <= j2; i++,j++){
        a[i]=temp[j];
    }
}
```

```c
void mergesort(int a[],int i,int j, int proc){
    int mid;
    n++;

    if(i < j){
        mid = (i + j)/2;

        if(n > 20){
            mergesort(a, i, mid, proc);
            mergesort(a, mid + 1, j, proc);
        }
        else{
            #pragma omp parallel sections num_threads(proc)
            {
                #pragma omp section
                {
                    mergesort(a, i, mid, proc);
                }
                #pragma omp section
                {
                    mergesort(a, mid+1, j, proc);
                }
            }
        }
        merge(a, i, mid, mid+1, j);
    }
}
```

Counting Sort:

```
henrique@HENRIQUE-PC: /media/henrique/DADOS/Usuarios/Henrique/workspac...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Tempo de execucao do CountingSort Paralelo:

real    0m0,502s
user    0m0,412s
sys     0m0,030s
Deseva comparar os dados da saida? n
Fim dos testes
Apagando os arquivos inuteis...
henrique@HENRIQUE-PC:/media/henrique/DADOS/Usuarios/Henrique/workspace/CountingS
ort$ ./testar.sh
Compilando os programas...
Deseja testar a velociadade de execucao do programa? sim
Testando a velociade de execucao do programa:
Tempo de execucao do CountingSort Não Paralelo:

real    0m0,515s
user    0m0,404s
sys     0m0,023s
Tempo de execucao do CountingSort Paralelo:

real    0m0,502s
user    0m0,402s
sys     0m0,038s
Deseva comparar os dados da saida?
```

Máquina:

OS: Ubuntu 18.04

CPU: Intel i5-3210M

Counting Sort:

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

#define n 1000000
#define bool short
#define true 1
#define false 0

//Variaveis globais
int array[n+1];
int i;
int j;
```

```c
int leArraySystemIn()
{
    for(int indice=0;indice<n;indice++)
    {
        scanf("%d%*c", &array[indice]);
    }//fim for
    return 0;
}//fim de leArraySystemIn

//Retorna o maior elemento do arranjo.
int getMaior()
{
    int maior = array[0];
    #pragma omp parallel for shared(maior)
    for(i=0;i<n;i++)
    {
        if(maior<array[i])
        {
            maior=array[i];
        }
    }
    return maior;
}//fim getMaior
```

Counting Sort:

```c
int main()
{
    //Lendo o arranjo
    printf("\nIniciando o programa\n");//teste
    leArraySystemIn();
    printf("\nArray lido com sucesso\n");//Teste
    countingSort();
    printf("\nArray ordenado com sucesso\n");//Teste
    imprimeArranjo();
    printf("\nArray impresso com sucesso\n");//Teste
    printf("\nFim do programa\n");
    return 0;
}//fim main
```

```c
// Algoritmo de ordenacao
int countingSort()
{

    //Array para contar o numero de ocorrencias de cada elemento
    int tamCount = getMaior() + 1;
    int* count = (int*) malloc(tamCount * sizeof(int));
    int ordenado[n];

    //Inicializar cada posicao do array de contagem
    #pragma omp parallel for shared(count, tamCount)
    for(i=0;i<tamCount;i++)
    {
        count[i]=0;
    }//fim for

    //Agora, o count[i] contem o numero de elemento iguais a i
    //#pragma omp parallel for shared(array, count)
    for(int i=0;i<n;i++)
    {
        int tmp=count[array[i]];
        count[array[i]]=tmp+1;
    }//fim for

    //Agora, o count[i] contem o numero de elemento menores ou iguais a i
    for(i=1;i<tamCount;i++)
    {
        count[i]+=count[i-1];
    }//fim for

    //Ordenando
    for(i=n-1;i>=0;i--)
    {
        ordenado[count[array[i]]-1]=array[i];
        count[array[i]]--;
    }//fim for

    //Copiando para o array original
    #pragma omp parallel for shared(array, ordenado)
    for(i=0;i<n;i++)
    {
        array[i]=ordenado[i]
    }//fim for

    return 0;
}//fim countingSort
```

Selection Sort($O(n^2)$)

Média de tempo sequencial:

Real: 25m21,390s
User: 25m20,890s
Sys: 0m0,200s

Média de tempo paralelo

Real: 11m9,790s
User: 39m26,302s
Sys: 0m5,423s

Máquina:

OS: Funtoo/Gentoo

CPU: Intel i7 4500U

Obs: Não use o selection sort para uma quantidade grande de elementos.... como 1.000.000

Selection Sort:

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

// Structure for enabling reduction on the index of elements
struct Compare { int val; int index; };
// Custom reduction for finding the index of the max element.
#pragma omp declare reduction(maximum : struct Compare : omp_out = omp_in.val > omp_out.val ? omp_in : omp_out)

int main()
{//Programa principal
    int array[1000000];
    for(int i = 0; i < 1000000; i++){
        scanf("%d", &array[i]);
    }

    imprimeArray(array, 1000000);
    selectionSort(array, 1000000);
    imprimeArray(array, 1000000);

    return 0;
}//Programa principal
```

Selection Sort:

```
1 void selectionSort(int* array, int numElementos)
2 {//Inicio selectionSort
3     int i, j;
4
5     for(i = 0; i < numElementos - 1; i++)
6     {//Inicio for i
7         // Declare the structure required for reduction
8         struct Compare max;
9
10        // Initialize the variables
11        max.val = array[i];
12        max.index = i;
13
14        // Parallel for loop with custom reduction, at the end of the loop,
15        //max will have the max element and its index.
16        #pragma omp parallel for reduction(maximum:max)
17        for(j = i + 1; j < numElementos; j++)
18        {//Inicio for j
19            if(array[j] > max.val){
20                max.index = j;
21                max.val = array[j];
22            }
23        }//Fim for j
24        swap(&array[max.index], &array[i]);
25    }//Fim for i
26 }//Fim selectionSort
```