Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente

Relatório de Entrega de Atividades

Aluno(s): Amanda Oliveira Alves e Fillype Alves do Nascimento
Matrícula: 15/0116276 e 16/0070431
Atividade: Aula Prática 07 - OpenMP

1. 0,740 segundos

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.1.c
// atividade: 1.1

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    for (int i = 0; i < n; i++) {
        soma = soma + valores[i];
    }

    return soma;
}

int main(){


    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
```

```
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");

    free(valores);

    return 0;
}
```

2. 1° - 0.275051 segundos
2° - 0.276095 segundos
3° - 0.275498 segundos
4° - 0.277346 segundos
5° - 0.275858 segundos
Média - 0.2759696 segundos

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.2.c
// atividade: 1.2

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    for (int i = 0; i < n; i++) {
        soma = soma + valores[i];
```

```
    }

    return soma;
}

int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();

    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);

    return 0;
}
```

3. 1 thread - Tempo: 1.658383 segundos Speedup: 0.166408845
2 threads - Tempo: 3.787002 segundos Speedup: 0.072872842
4 threads - Tempo: 5.036278 segundos Speedup: 0.05479634

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.3.c
// atividade: 1.3

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    omp_set_num_threads(1);

    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        printf("thread number: %d\n", id);
        for (int i = id; i < n; i+=1) {
            #pragma omp critical
            {
                soma = soma + valores[i];
            }
        }

    }
    return soma;

}

int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));
```

```
    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();
    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);

    return 0;
}
```

4. 1 thread - Tempo: 0.833068 segundos Speedup: 0.331268996
2 threads - Tempo: 2.793510 segundos Speedup: 0.098789551
4 threads - Tempo: 2.479831 segundos Speedup: 0.111285648

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.4.c
// atividade: 1.4

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    omp_set_num_threads(1);
```

```
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        printf("thread number: %d\n", id);
        for (int i = id; i < n; i+=1) {
            #pragma omp atomic
                soma = soma + valores[i];
        }

    }
    return soma;

}

int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();
    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);
```

```
    return 0;
}
```

5. 1 thread - Tempo: 0.291191 segundos Speedup: 0.947727093
2 threads - Tempo: 0.169315 segundos Speedup: 1.6299182
4 threads - Tempo: 0.163667 segundos Speedup: 1.686165201

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.5.c
// atividade: 1.5

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    omp_set_num_threads(4);

    #pragma omp parallel
    {
        long long int soma_parcial = 0;
        int id = omp_get_thread_num();
        printf("thread number: %d\n", id);
        for (int i = id; i < n; i+=4) {
            soma_parcial = soma_parcial + valores[i];
        }
        #pragma omp atomic
        soma+=soma_parcial;
    }
    return soma;

}
```

```
int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();
    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);

    return 0;
}
```

6. 1 thread - Tempo: 0.259813 segundos Speedup: 1.062185495
2 threads - Tempo: 0.143571 segundos Speedup: 1.922182056
4 threads - Tempo: 0.133893 segundos Speedup: 2.061120447

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.6.c
// atividade: 1.6

#include <stdio.h>
```

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    int i;

    omp_set_num_threads(2);

    #pragma omp parallel private(i)
    {
        long long int soma_parcial = 0;
        int id = omp_get_thread_num();
        printf("thread number: %d\n", id);
        #pragma omp for private(i)
        for (i = 0; i < n; i++) {
            soma_parcial = soma_parcial + valores[i];
        }
        #pragma omp atomic
        soma+=soma_parcial;
    }
    return soma;

}

int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
```

```
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();
    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);

    return 0;
}
```

7. Tempo: 0.125814 segundos Speedup: 2.193472904

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.7.c
// atividade: 1.7

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

long long int somavalores(int *valores, int n){

    long long int soma = 0;
    int i;

    #pragma omp parallel for private(i) reduction(+:soma)
    for (i = 0; i < n; i++) {
        soma += valores[i];
    }

    return soma;
```

```
}

int main(){

    long long int i, n, soma;
    int *valores;

    // scanf("%lld", &n);
    n = 100000000;
    valores = (int *)malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        valores[i] = 1;
    }

    double initialTime = omp_get_wtime();

    soma = somavalores(valores, n);
    printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
    free(valores);

    double finalTime = omp_get_wtime();
    printf("Tempo de processamento: %lf segundos.\n", finalTime -
initialTime);

    return 0;
}
```