Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente - 2019/2

Relatório de Entrega de Atividades

Aluno(s): Ana Luísa
Gabriel Sylar
Atividade: Laboratório 7 - OpenMP

1.1. Mensure o tempo gasto pelo programa base, utilizando o comando time.

```
#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 = 1000000000;
	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");

	return 0;
}
```

Resposta:
O código fornecido (acima), foi executado utilizando o comando time, fornecendo os seguintes tempos:

```
time ./11

Soma: 1000000000 - ok

real  0m6,597s
user  0m5,601s
sys   0m0,996s
```

1.2. Mensure o tempo gasto pelo programa base, utilizando a função omp_get_w_time, medindo desde o início do programa até o término (liberação da memória alocada) e medindo apenas o tempo de processamento. Como a intenção é reduzir o tempo de interferência do SO (principalmente no segundo caso), realize a medição no mínimo três vezes para verificar a média dos tempos decorridos.

Resposta: Dividimos a atividade em 1.2a, que mensura do início do programa até o término e 1.2b, que mensura somente o tempo de processamento.

Para a 1.2a, utilizamos o seguinte código:

```
#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;

	double inicio = omp_get_wtime();
	// scanf("%lld", &n);
	n = 1000000000;
	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");

	double fim = omp_get_wtime();
	printf ("Tempo total da execução: %lf\n", fim-inicio);

	return 0;
}
```

Obtivemos os seguintes resultados:

```
for i in {1..3}; do ./12a; done

Soma: 1000000000 - ok
Tempo total da execução: 6.445593

Soma: 1000000000 - ok
Tempo total da execução: 6.444732

Soma: 1000000000 - ok
Tempo total da execução: 6.441025
```

A média para o tempo total da execução foi 6,4438.

Para a 1.2b, utilizamos o seguinte código:

```
#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 = 1000000000;
	valores = (int *)malloc(n * sizeof(int));

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

	double inicio = omp_get_wtime();
	soma = somavalores(valores, n);
	double fim = omp_get_wtime();
```

```
	printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
	printf ("Tempo da soma somente: %lf\n", fim-inicio);

	return 0;
}
```

Obtivemos os seguintes resultados:

```
for i in {1..3}; do ./12b; done

Soma: 1000000000 - ok
Tempo da soma somente: 2.577480

Soma: 1000000000 - ok
Tempo da soma somente: 2.578567

Soma: 1000000000 - ok
Tempo da soma somente: 2.575945
```

A média do tempo de processamento foi 2,577330667.

1.3. Atividade - Utilize a diretiva pragma omp parallel para paralelizar o trecho do código base que realiza as somas. O laço deve ser modificado de forma a variável i iterar em cima da quantidade de threads, ao invés de apenas incrementar. É a diretiva critical do OpenMP para sanar a condição de corrida envolvendo a variável soma. Mensure os tempos utilizando 1, 2 e 4 threads, que coincidem com a quantidade de núcleos dos computadores utilizados no laboratório.

Resposta: Para realizar essa atividade, utilizamos o seguinte código, alterando somente o número de threads (NT):

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

#define NT 4

long long int somavalores (int *valores, int n) {
	long long int soma = 0;
	int mt;
	omp_set_num_threads (NT);

	#pragma omp parallel
	{
		int i, id;
		mt = omp_get_num_threads();
		id =  omp_get_thread_num();
		for (i = id; i < n; i=i+mt) {
```

```
                    #pragma omp critical
                    soma = soma + valores[i];
                }
        }
        return soma;
}

int main() {
        long long int i, n, soma;
        int *valores;

        // scanf("%lld", &n);
        n = 1000000000;
        valores = (int *)malloc(n * sizeof(int));

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

        double inicio = omp_get_wtime();
        soma = somavalores(valores, n);
        double fim = omp_get_wtime();

        printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
        printf ("Tempo de processamento: %lf\n", fim-inicio);

        return 0;
}
```

Para 1 thread, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./13; done

Soma: 1000000000 - ok
Tempo de processamento: 16.620279

Soma: 1000000000 - ok
Tempo de processamento: 16.744026

Soma: 1000000000 - ok
Tempo de processamento: 16.901299
```

A média foi 16,755201333.

Para 2 threads, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./13; done

Soma: 1000000000 - ok
Tempo de processamento: 35.777798

Soma: 1000000000 - ok
Tempo de processamento: 34.247624
```

```
Soma: 1000000000 - ok
Tempo de processamento: 32.845985
```

A média foi 34,290469.

Para 4 threads, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./13; done

Soma: 1000000000 - ok
Tempo de processamento: 49.323579

Soma: 1000000000 - ok
Tempo de processamento: 49.931614

Soma: 1000000000 - ok
Tempo de processamento: 45.382910
```

A média foi 48,212701.

1.4. Atividade - Troque a diretiva critical por atomic e mensure a diferença entre os tempos de ambos.

Resposta:

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

#define NT 4

long long int somavalores (int *valores, int n) {
	long long int soma = 0;
	int mt;
	omp_set_num_threads (NT);

	#pragma omp parallel
	{
		int i, id;
		mt = omp_get_num_threads();
		id =  omp_get_thread_num();
		for (i = id; i < n; i=i+mt) {
			#pragma omp atomic
			soma = soma + valores[i];
		}
	}
	return soma;
}

int main() {
```

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

      // scanf("%lld", &n);
      n = 1000000000;
      valores = (int *)malloc(n * sizeof(int));

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

      double inicio = omp_get_wtime();
      soma = somavalores(valores, n);
      double fim = omp_get_wtime();

      printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
      printf ("Tempo de processamento: %lf\n", fim-inicio);

      return 0;
}
```

Para 1 thread, usando atomic, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./14; done

Soma: 1000000000 - ok
Tempo de processamento: 8.140362

Soma: 1000000000 - ok
Tempo de processamento: 8.139267

Soma: 1000000000 - ok
Tempo de processamento: 8.136253
```

A média foi 8,1386273.

Para 2 threads, usando atomic, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./14; done

Soma: 1000000000 - ok
Tempo de processamento: 28.676331

Soma: 1000000000 - ok
Tempo de processamento: 34.491020

Soma: 1000000000 - ok
Tempo de processamento: 33.758130
```

A média foi 32,3084936.

Para 4 threads, usando atomic, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./14; done
```

```
Soma: 1000000000 - ok
Tempo de processamento: 26.989878

Soma: 1000000000 - ok
Tempo de processamento: 25.750194

Soma: 1000000000 - ok
Tempo de processamento: 27.365877
```

A média foi 26,701983.

1.5. Altere o programa base para utilizar somas locais por thread antes de acumular na variável soma os resultados do trabalho individual de cada uma. Lembre-se que acumular as somas parciais na soma total é uma região critica e deve ser tratada (critical ou atomic) para evitar condições de corrida. Mensure os tempos e compare com o código base.

Resposta:

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

#define NT 4

long long int somavalores (int *valores, int n) {
	long long int soma = 0;
	int mt;
	omp_set_num_threads (NT);

	#pragma omp parallel
	{
		long long int soma_parcial = 0;
		int i;
		for (i = 0; i < n; i++) {
			soma_parcial = soma_parcial + valores[i];
		}
		#pragma omp atomic
		soma = soma + soma_parcial;
	}
	return soma;
}

int main() {
	long long int i, n, soma;
	int *valores;

	// scanf("%lld", &n);
	n = 1000000000;
```

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

      int n_parcial = n / NT;
      for (i = 0; i < n_parcial; i++)
      valores[i] = 1;

      double inicio = omp_get_wtime();
      soma = somavalores(valores, n_parcial);
      double fim = omp_get_wtime();

      printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
      printf ("Tempo de processamento: %lf\n", fim-inicio);

      return 0;
}
```

Para 1 thread, usando somas parciais, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./15; done

Soma: 1000000000 - ok
Tempo de processamento: 3.197666

Soma: 1000000000 - ok
Tempo de processamento: 3.196728

Soma: 1000000000 - ok
Tempo de processamento: 3.228015
```

A média foi 3,207469667.

Para 2 threads, usando somas parciais, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./15; done

Soma: 1000000000 - ok
Tempo de processamento: 1.606616

Soma: 1000000000 - ok
Tempo de processamento: 1.608764

Soma: 1000000000 - ok
Tempo de processamento: 1.606490
```

A média foi 1,60729.

Para 4 threads, usando somas parciais, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./15; done

Soma: 1000000000 - ok
Tempo de processamento: 1.176983
```

```
Soma: 1000000000 - ok
Tempo de processamento: 1.161521

Soma: 1000000000 - ok
Tempo de processamento: 1.166281
```

A média foi 1,168261667.

1.6. Atividade - A partir do código da atividade anterior (somas parciais separadas e em seguida acúmulo na soma total), altere o código para utilizar a diretiva pragma omp for (lembre-se que a região crítica ainda precisa ser tratada).

Resposta:

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

#define NT 4

long long int somavalores (int *valores, int n) {
	long long int soma = 0;
	int mt;
	omp_set_num_threads (NT);

	#pragma omp parallel
	{
		long long int soma_parcial = 0;
		int i;
		#pragma omp for
		for (i = 0; i < n; i++) {
			soma_parcial = soma_parcial + valores[i];
		}
		#pragma omp atomic
		soma = soma + soma_parcial;
	}
	return soma;
}

int main() {
	long long int i, n, soma;
	int *valores;

	// scanf("%lld", &n);
	n = 1000000000;
	valores = (int *)malloc(n * sizeof(int));

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

	double inicio = omp_get_wtime();
```

```
	soma = somavalores(valores, n);
	double fim = omp_get_wtime();

	printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
	printf ("Tempo de processamento: %lf\n", fim-inicio);

	return 0;
}
```

Para 1 thread, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./16; done

Soma: 1000000000 - ok
Tempo de processamento: 2.270009

Soma: 1000000000 - ok
Tempo de processamento: 2.267783

Soma: 1000000000 - ok
Tempo de processamento: 2.264670
```

A média foi 2,267487333.

Para 2 threads, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./16; done

Soma: 1000000000 - ok
Tempo de processamento: 1.137704

Soma: 1000000000 - ok
Tempo de processamento: 1.139439

Soma: 1000000000 - ok
Tempo de processamento: 1.130278
```

A média foi 1,135807.

Para 4 threads, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./16; done

Soma: 1000000000 - ok
Tempo de processamento: 0.869599

Soma: 1000000000 - ok
Tempo de processamento: 0.876155

Soma: 1000000000 - ok
Tempo de processamento: 0.865649
```

A média foi 0,870467667.

1.7. Atividade - Altere o programa base para utilizar a redução provida pelo OpenMP e mensure os tempos comparando com o código base.

Resposta:

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

#define NT 4

long long int somavalores (int *valores, int n) {
      long long int soma = 0;
      int mt;
      omp_set_num_threads (NT);

      int i;
      #pragma omp parallel for reduction (+:soma)
      for (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 = 1000000000;
      valores = (int *)malloc(n * sizeof(int));

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

      double inicio = omp_get_wtime();
      soma = somavalores(valores, n);
      double fim = omp_get_wtime();

      printf("Soma: %lld - %s\n", soma, soma == n ? "ok" : "falhou");
      printf ("Tempo de processamento: %lf\n", fim-inicio);

      return 0;
}
```

Para 1 thread, usando reduction, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./17; done

Soma: 1000000000 - ok
Tempo de processamento: 2.266838
```

```
Soma: 1000000000 - ok
Tempo de processamento: 2.274488

Soma: 1000000000 - ok
Tempo de processamento: 2.268480
```

A média foi 2,269935333.

Para 2 threads, usando reduction, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./17; done

Soma: 1000000000 - ok
Tempo de processamento: 1.135416

Soma: 1000000000 - ok
Tempo de processamento: 1.136759

Soma: 1000000000 - ok
Tempo de processamento: 1.139616
```

A média foi 1,137263667.

Para 4 threads, usando reduction, obtivemos os seguintes resultados:

```
for i in {1..3}; do ./17; done

Soma: 1000000000 - ok
Tempo de processamento: 0.876085

Soma: 1000000000 - ok
Tempo de processamento: 0.868409

Soma: 1000000000 - ok
Tempo de processamento: 0.879577
```

A média foi 0,874690333.

Fim.