Implementações de algorítmos de contagem de números primos naive e Bag of tasks em OpenMP.

Victor Abi-Ramia Antonio Rachide

July 1, 2024

1 Introdução

Este trabalho tem como objetivo estudar as métricas de speedup e eficiência para implementações dos algorítmos de contagem de números primos usando abordagem naive e Bag of tasks em OpenMP. Os dois casos estudados neste trabalho utilizam diferentes opções de escalonamento: "static" e "dynamic", para criar implementações correspondentes do trabalho anterior, feito em MPI, para os algorítmos de contagem de números primos naive e Bag of tasks. No caso do algoritmo análogo ao bag of tasks do último trabalho, diferentes tamanhos de "chunksize" foram investigados: 100, 1000 e 10000 com o intuito de avaliar possível variações de peformace. A descrição dos algoritmos Naive e Bag of tasks para o problema de contagem de números primos é apresentada na seção 2. Na seção 3 as configurações usadas são apresentadas. Na seção 4, os resultados são demonstrados.

2 Algoritmos de contagem de números primos

2.1 Abordagem Naive

O algoritmo baseado em método "Naive" para contagem de números primos envolve na divisão de um intervalo de números a serem verificados entre diferentes processos. Nesta versão do algoritmo baseado em método "Naive" implementada usando OpenMP, o processo mestre sinaliza para API do OpenMP a necessidade de criar outras threads que terão intervalos de tamanho aproximadamente iguais para realizar a contagem de números primos. Depois que cada thread tiver suas contagens parciais, um processo de "reduce" é realizado, obtendo a contagem total de números primos.

```
total = 0;  // Reinicia o total para cada número de threads

t_inicio = omp_get_wtime();
#pragma omp parallel for reduction(+:total) schedule(static) num_threads(numberOfthreads[j])
for (long int i = 3; i <= n; i += 2) {
    if (primo(i) == 1) {
        total++;
    }
}

total += 1;  // Acrescenta o número 2, que também é primo

t_fim = omp_get_wtime();

#pragma omp single
{
    printf( "Quant. Threads: %ld \n", numberOfthreads[j]);
    printf( "Quant. de primos entre 1 e %ld: %ld \n", n, total);
    printf( "Tempo de execução: %3.10f segundos\n", t_fim - t_inicio);
    timematrix[m][j][k] = t_fim - t_inicio;
}
```

2.2 Abordagem Bag of tasks

O algoritmo baseado no método "Bag of tasks" para contagem de números primos consiste na partilha do intervalo total de números a serem verificados em intervalos menores, na implementação em MPI por um processo mestre. Contudo, nesta implementação em openMP, o "OpenMP runtime system" é responsável pela criação dos processos subordinados e para distribuição dos seus intervalos de números a serem verificados. Diferentemente da versão "Naive", o escalonamento implementado divide a tarefa em intervalos do tamanho especificado na variável "chunksize" e que são distribuídos para as processos usando com uma política "first-come-fisrt-served".Depois que cada thread tiver suas contagens parciais e todos intervalos tiverem seus números primos contados, um processo de "reduce" é realizado, obtendo a contagem total de números primos.

```
total = 0;  // Reinicia o total para cada número de threads

t_inicio = omp_get_wtime();
#pragma omp parallel for reduction(+:total) schedule(dynamic, 1000) num_threads(numberOfthreads[j])
for (long int i = 3; i <= n; i += 2) {
    if (primo(i) == 1) {
        total++;
    }
}

total += 1;  // Acrescenta o número 2, que também é primo

t_fim = omp_get_wtime();

#pragma omp single
{
    printf( "Quant. Threads: %ld \n", numberOfthreads[j]);
    printf( "Quant. de primos entre 1 e %ld: %ld \n", n, total);
    printf( "Tempo de execução: %3.10f segundos\n", t_fim - t_inicio);
    timematrix[m][j][k] = t_fim - t_inicio;
}
```

3 Computador e compilador:

Neste trabalho, usei as seguintes configurações:

Compilador	gcc $(\text{x86}_6 4 - win32 - seh - rev0, MinGW - W64) 8.1.0$
Número de processadores	4
Processador	Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz 2.00 GHz
Arquitetura do processador	AMD64
Memória RAM	4 GB
Sistama Operacional	Windows 64 bits

4 Resultados:

As seguintes execuções dos programas foram realizadas, usei 5 execuções para obter a média e o desvio padrão dos tempos:

n	$[10^6, 10^7, 10^8]$
Número de threads	1-7
Número de execuções	5
Schedule(dynamic)	$[10^2, 10^3, 10^4]$

4.1 $\mathbf{n} = 10^6$

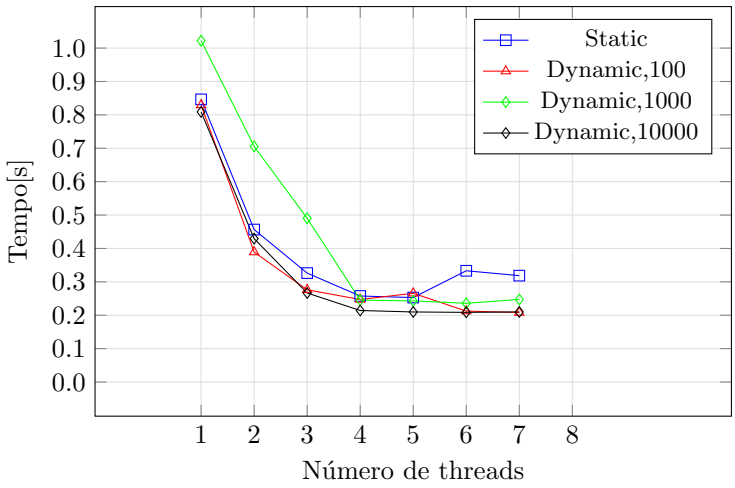

Figure 1: Média dos Tempos de execução vs. Número de threads, n = 10^6

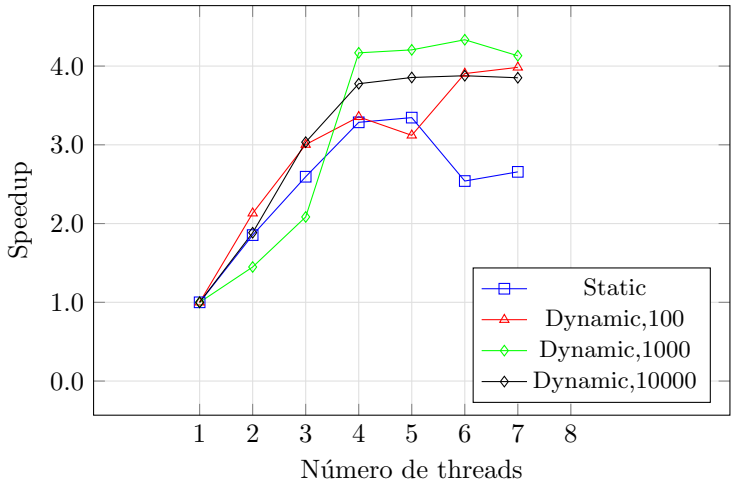

Figure 2: Média dos Speedup vs. Número de threads, n = 10^6

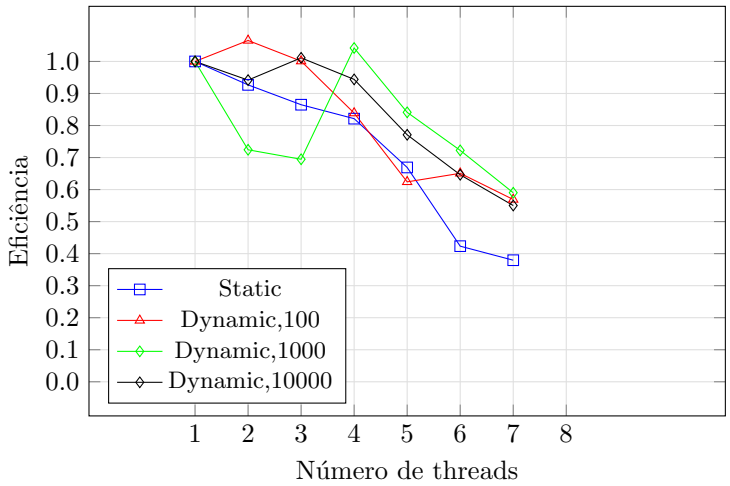

Figure 3: Média das Eficiências vs. Número de threads, n = 10^6

4.2 n = 10^7

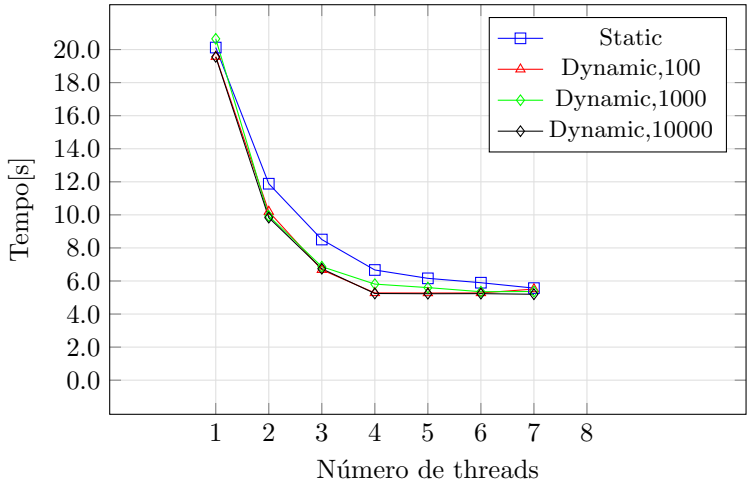

Figure 4: Média dos Tempos de execução vs. Número de threads, n = 10^7

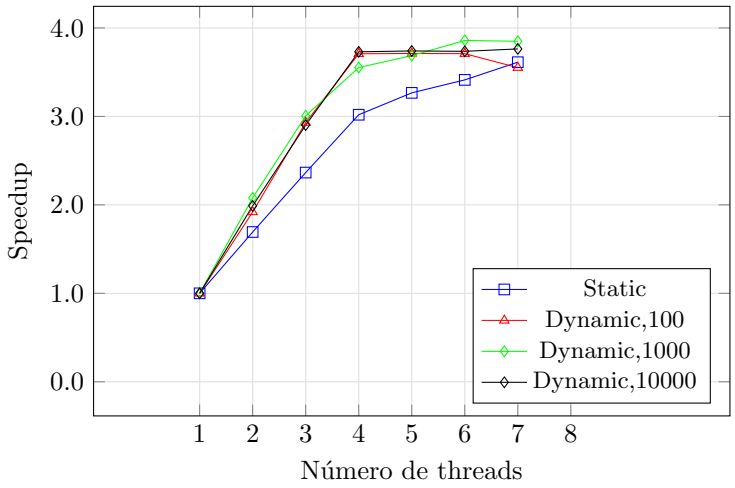

Figure 5: Média dos Speedup vs. Número de threads, n = 10^7

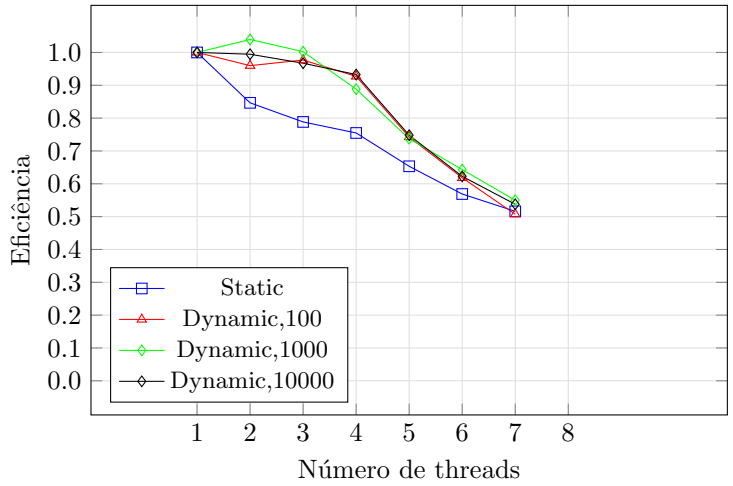

Figure 6: Média das Eficiências vs. Número de threads, n = 10^7

4.3 $\mathbf{n = 10^8}$

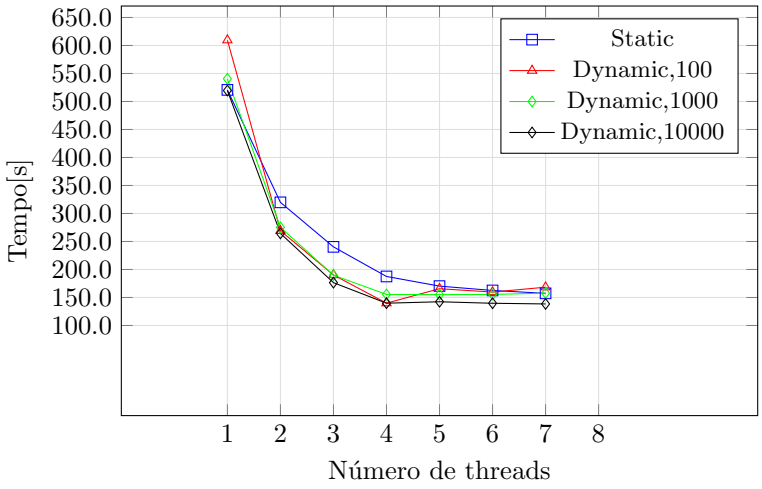

Figure 7: Média dos Tempos de execução vs. Número de threads, n = 10^8

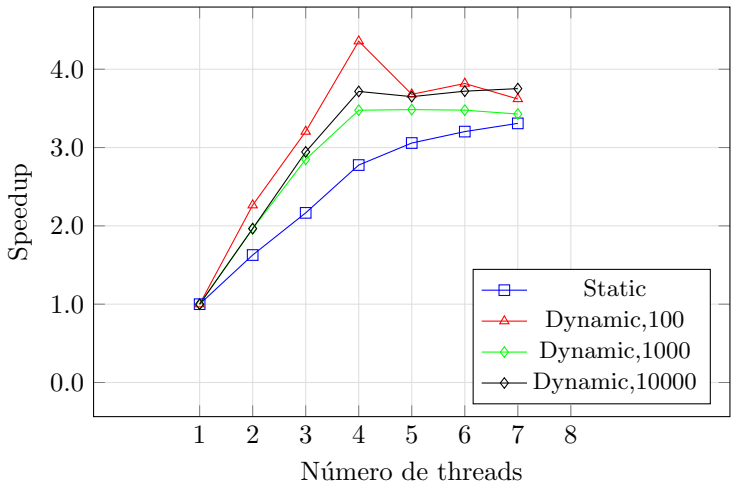

Figure 8: Média dos Speedup vs. Número de threads, n = 10^8

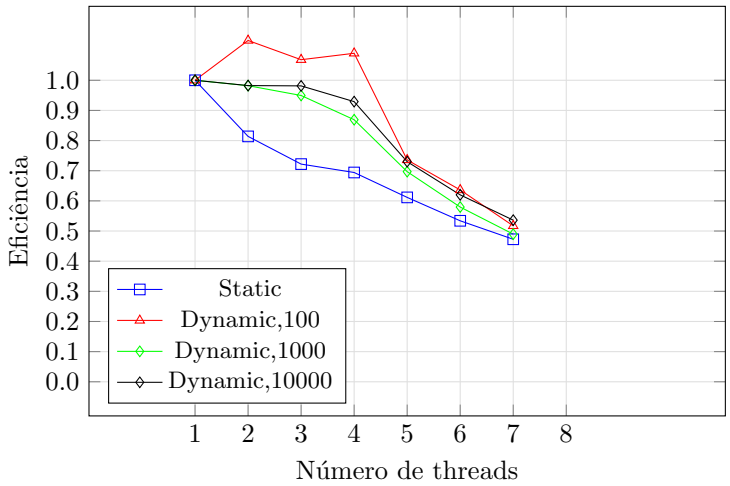

Figure 9: Média das Eficiências vs. Número de threads, n = 10^8