

Simulador de Propagação de Vírus com OpenMP


Crístian Marcos Weber
João Victor Bolsson

Estratégias adotadas:

- Solução 1: Paralelizar o for de 'n_trials' iterações em virusim.cpp utilizando schedule static (os loops apresentam carga de trabalho equivalente), definindo a região de soma parcial como região crítica.
- Solução 2: Paralelizar o for mais externo em virusim.cpp utilizando schedule dynamic (apresentou melhor resultado), sem necessidade de definir região crítica.

Para as duas soluções foi criado um objeto da classe Population para cada thread.

Solução 1:


```
#pragma omp parallel for schedule(static) num_threads(nthreads)   
for (int it = 0; it < n_trials; it++) {  
    populations[omp_get_thread_num()->propagateUntilOut(populations[omp_get_thread_num()->centralPerson(),  
                                                         prob_spread[ip], rand)  
  
    double aux = populations[omp_get_thread_num()->getPercentInfected();  
    #pragma omp critical  
    {  
        percent_infected[ip] += aux;  
    }  
}
```

Solução 2:

```
//para cada probabilidade, calcula o percentual de pessoas infectadas
#pragma omp parallel for schedule(dynamic) num_threads(nthreads)
for (int ip = 0; ip < n_probs; ip++) {

    prob_spread[ip] = prob_min + (double) ip * prob_step;
    percent_infected[ip] = 0.0;
    rand.setSeed(base_seed+ip); // nova sequencia de numeros aleatorios

    // executa varios experimentos para esta probabilidade
    for (int it = 0; it < n_trials; it++) {
        // queima floresta ate o fogo apagar
        populations[omp_get_thread_num()->propagateUntilOut(populations[omp_get_thread_num()->centralPerson(), prob_spread[ip], rand)
        percent_infected[ip] += populations[omp_get_thread_num()->getPercentInfected();
    }
}
```



Testes Realizados

Todos os resultados apresentados são referentes a execuções com valores de <nro. experimentos> e <probab. maxima> iguais à 5000 e 101, respectivamente.

Os testes feitos abrangeram a mudança de complexidade do problema a partir da variação do tamanho da população. Para este parâmetro, foram testados os valores 20, 40 e 80 tanto para execução sequencial quanto para execução paralelizada (em 2 threads) das duas soluções apresentadas.

Os experimentos foram realizados em um ambiente com SO Ubuntu e uma arquitetura de 2 núcleos de 2.30GHz.

Resultados:

Execução	Tamanho da População	Tempo de execução (segundos)
Sequencial	20	71.73
Solução 1 (2 threads)	20	63.35
Solução 2 (2 threads)	20	62.21
Sequencial	40	421.66
Solução 1 (2 threads)	40	326.58
Solução 2 (2 threads)	40	328.29
Sequencial	80	28.2526
Solução 1 (2 threads)	80	1916.82
Solução 2 (2 threads)	80	1928.19

Resultados:

Execução	Tamanho da População	Speedup
Solução 1 (2 threads)	20	1.1323
Solução 2 (2 threads)	20	1.1530
Solução 1 (2 threads)	40	1.2911
Solução 2 (2 threads)	40	1.2844
Solução 1 (2 threads)	80	1.4739
Solução 2 (2 threads)	80	1.4652

Observações:

Apesar de não ter sido possível explorar a fundo a paralelização pela limitação do número de núcleos disponíveis, foi possível notar um speedup crescente com o aumento da carga do problema, que apesar de ter sido pouco significativo para execuções com baixa população, se mostrou considerável nas execuções de maior complexidade.