T4: Simulador de Propagação de Vírus com OpenMP

Cadeira: Programação Paralela - elc139

Integrantes: René Gargano Ferrari e Thiago Pavin

Estratégias de Paralelização Adotadas

Versão 1

```
#pragma omp parallel for schedule(runtime)
for (int ip = 0; ip < n_probs; ip++)
```

Na Versão 1 a paralelização foi feita a partir do loop que varia a probabilidade, englobando também o loop de testes. Fazendo com que ‘n_probs’ seja distribuída entre as threads.

Versão 2

```
#pragma omp parallel for schedule(runtime)
for (int it = 0; it < n_trials; it++)
```

Na Versão 2 a paralelização foi feita apenas no loop de testes. Distribuindo assim ‘n_trials’ entre as threads.

Experimentos Realizados

Os experimentos foram feitos a partir das duas versões apresentadas, utilizando o Sistema operacional ‘Ubuntu 18.04’ o hardware utilizado foi, CPU Intel Core i7-6500U 3.053 GHz e 7.89GB de memória.

Escalonamento:

- Static
- Dynamic
- Guided

Configurações do Problema:

- Pequena (10 100 101)
- Média (30 1000 101)
- Grande (60 3000 101)

Threads:

- 1
- 2
- 3
- 4

Escolha do Método de Escalonamento

Dos três métodos de escalonamento testados, notou-se uma melhora maior na média de tempo de execução no método *dynamic*. Logo o escolhemos como método principal para a realização dos testes.

Quando o escalonamento é definido para *dynamic*, o OpenMP dividirá as iterações em chunks com tamanho definido pela variável chunk_size. As threads serão executadas por ordem de chegada. A primeira thread que terminar de executar seu chunk irá entrar na fila para executar o próximo chunk. Caso o chunk_size não seja definido, ele será 1 por default.

Escolha do Método de Escalonamento

Método Static

O método *static* executa suas threads de forma cíclica. Como há um aumento de trabalho conforme as iterações do programa, julgamos que isso é algo ruim para o desempenho, já que threads que terminarem seu trabalho antes terão que esperar threads que receberam mais trabalho para fazer.

Escolha do Método de Escalonamento

Guided vs Dynamic

O método *guided* se mostra melhor que o *dynamic* em alguns casos, porém dependendo da escolha do *chunk-size* ele também pode ter um desempenho pior.

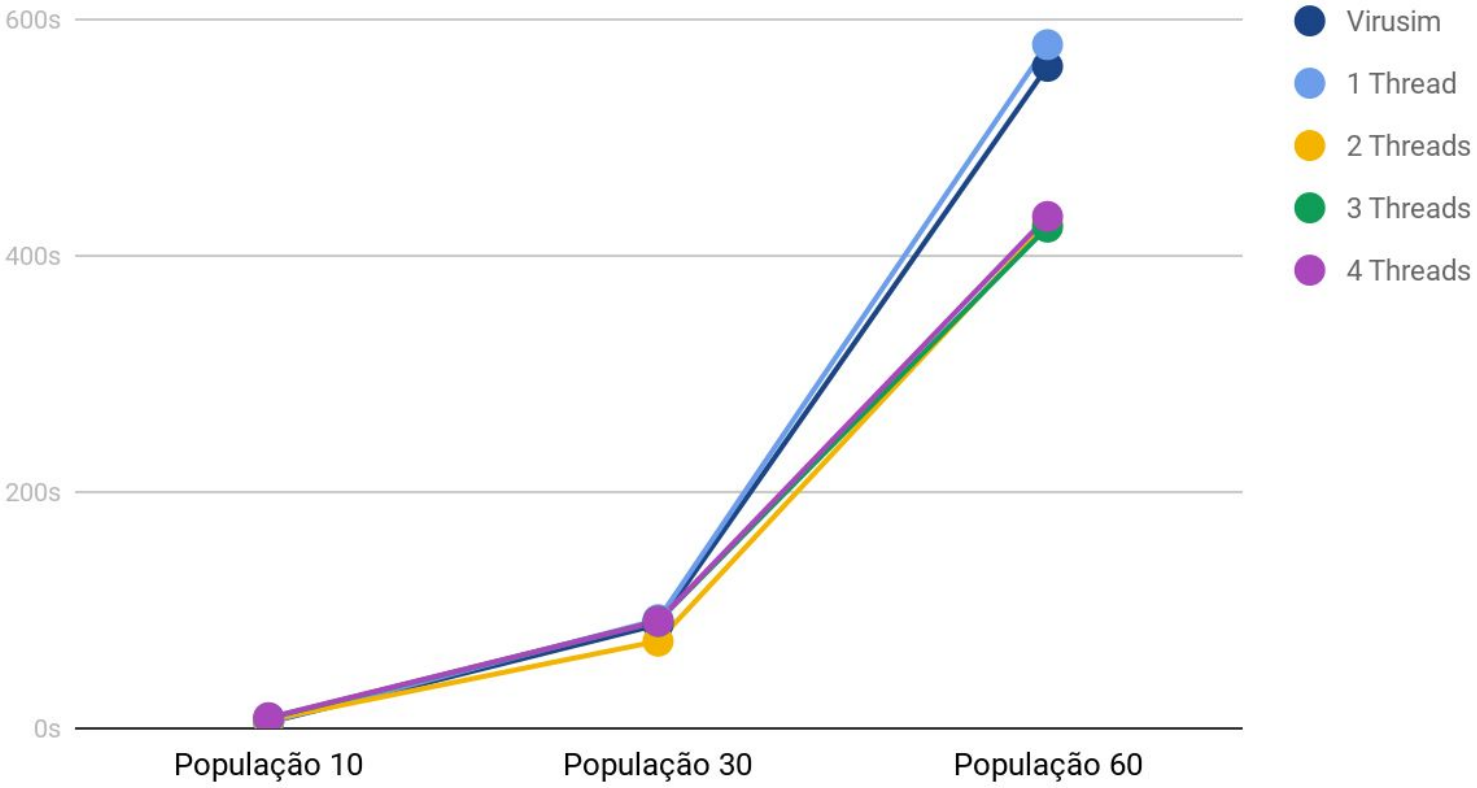
Dada a dificuldade de prever a eficiência do método *guided* em diferentes situações, optamos por escolher o método *dynamic* por ter maior constância a partir de uma boa escolha do *chunk-size.*

Experimentos

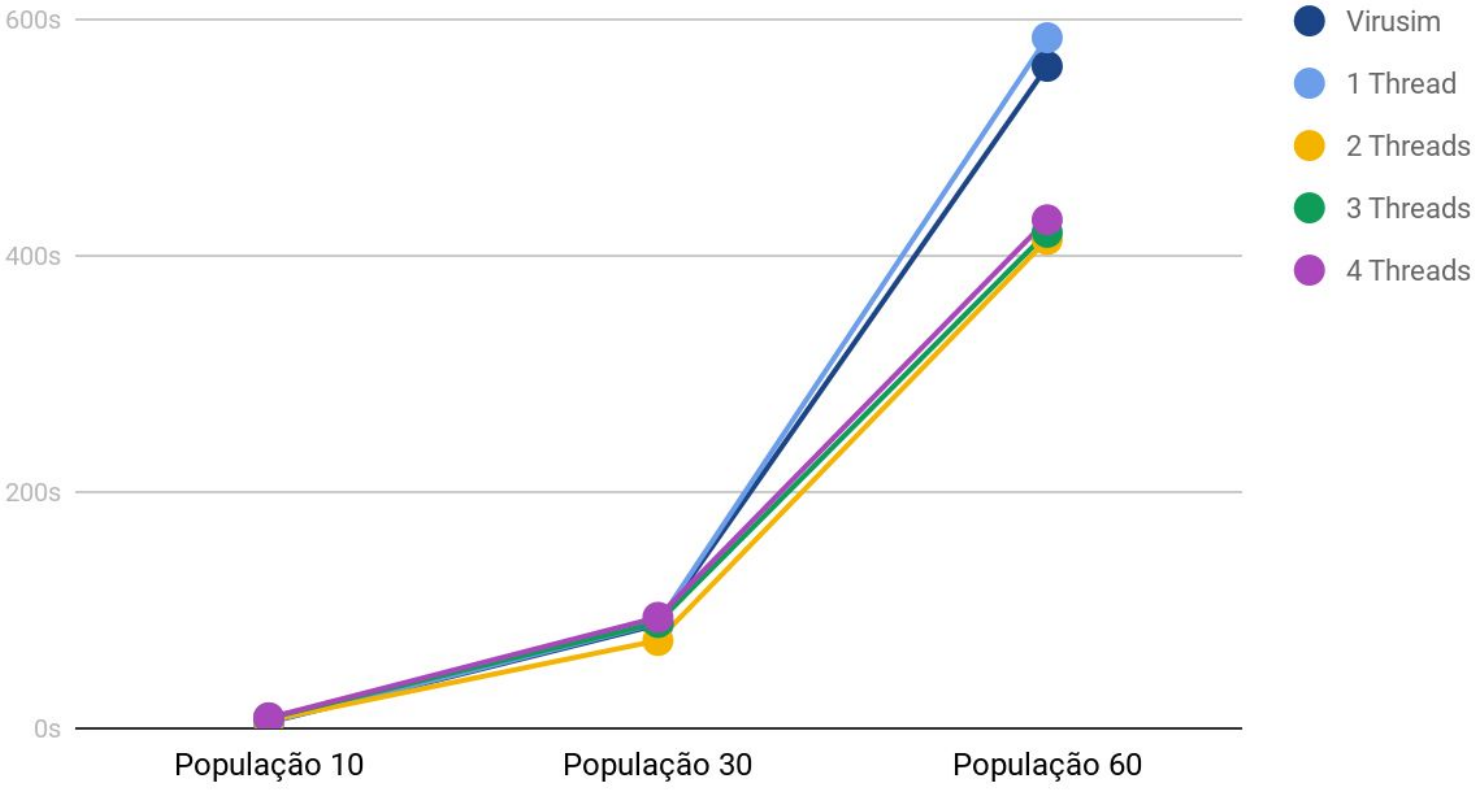
A partir da realização de inúmeros testes, notou-se que a influência das threads na redução do tempo de execução da aplicação aumentou com o incremento do tamanho da população.

É possível notar essa diferença nos gráficos a seguir, onde a população é variada e o número de trials é fixo em 3000.

Version 1 vs Virusim



Version 2 vs Virusim

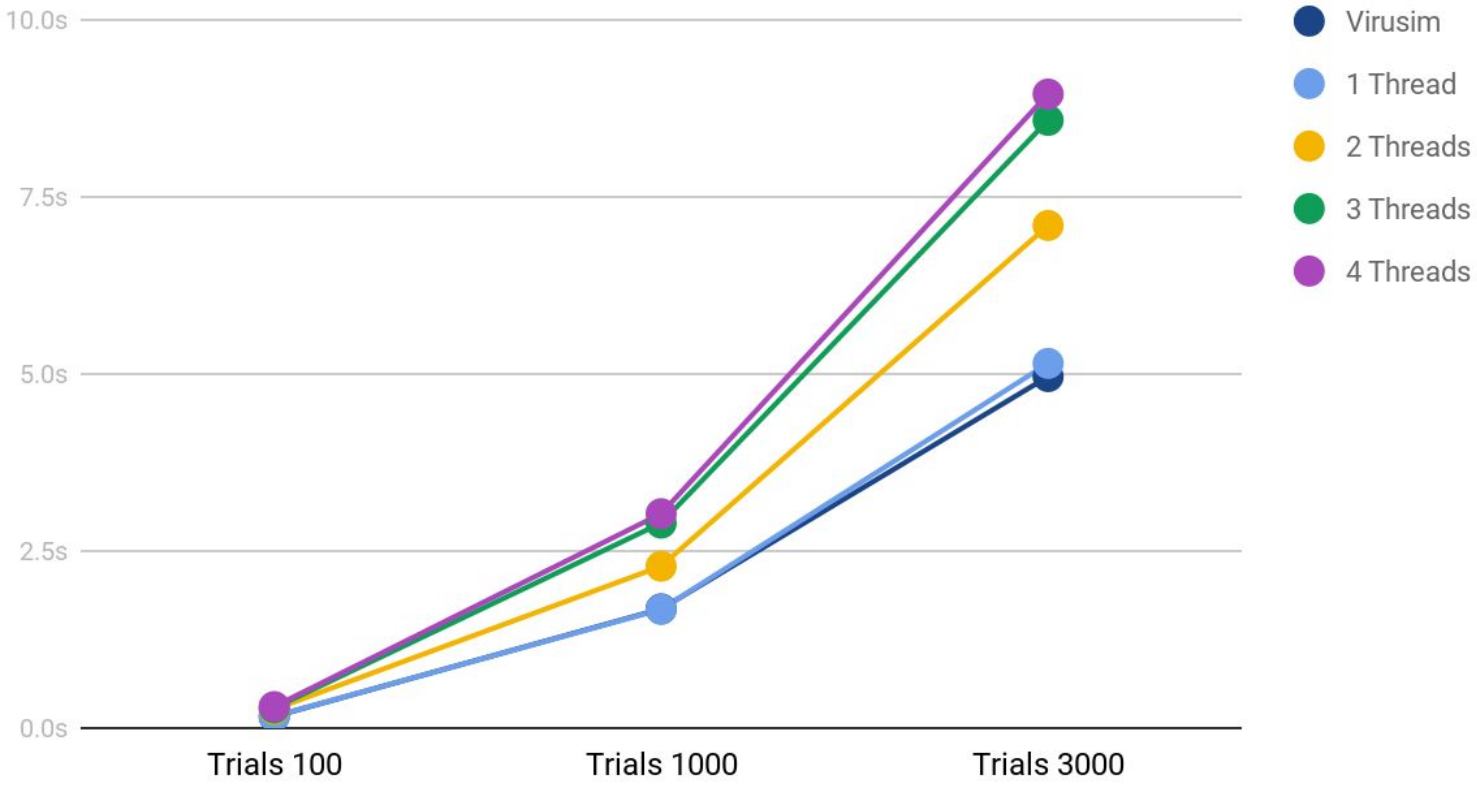


Experimentos

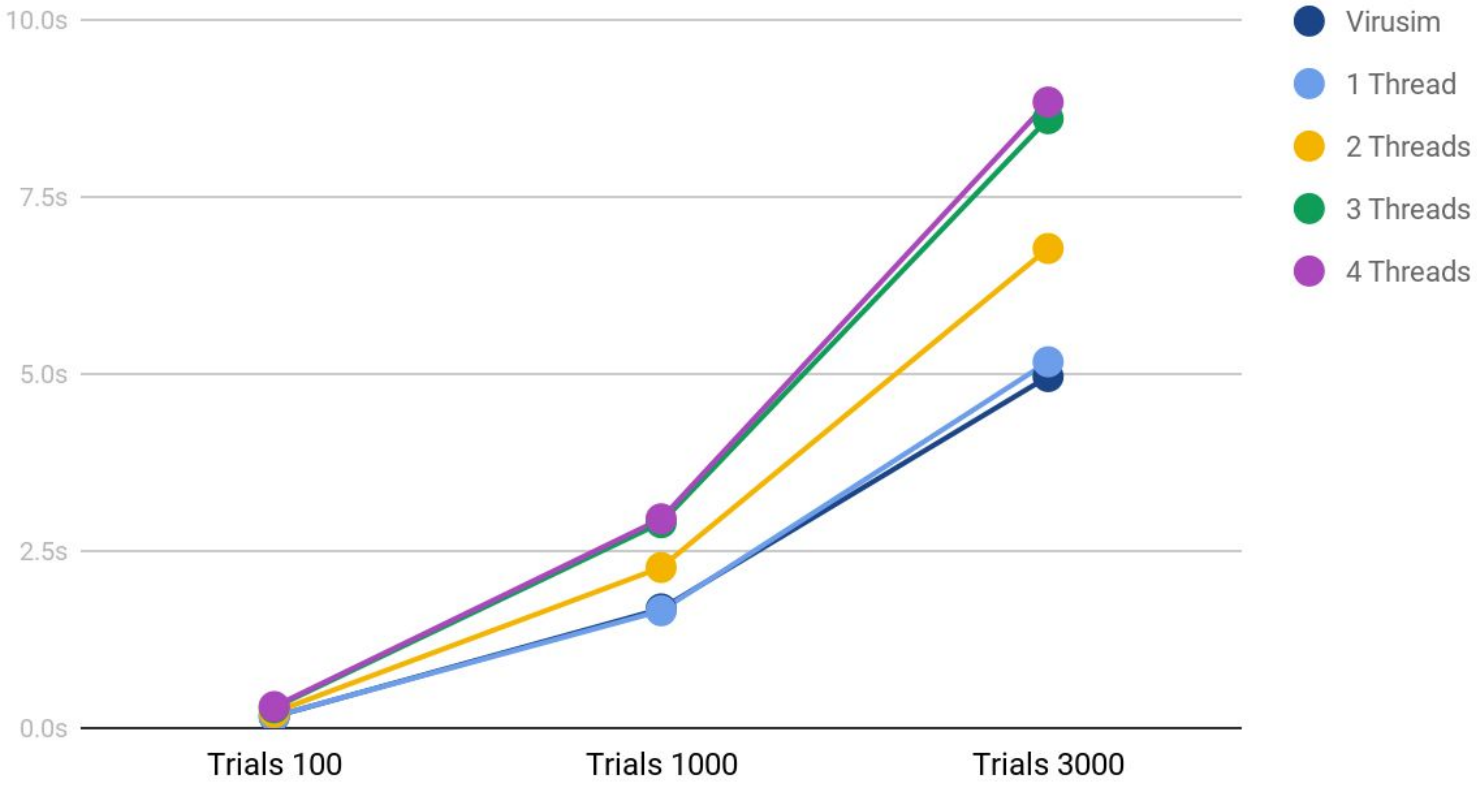
Diferentemente dos testes anteriores, notou-se que há uma influência negativa das threads no tempo de execução da aplicação com o incremento da quantidade de trials.

É possível notar essa diferença nos gráficos a seguir, onde a população é fixa em 10 e o número de trials é variado.

Version 1 vs Virusim



Version 2 vs Virusim



Considerações Finais

- Enquanto a Versão 1 divide ‘n_probs’ de trabalho entre as threads, a Versão 2 divide ‘n_trials’ de trabalho entre as threads. Notou-se que ambas as abordagens obtiveram resultados parecidos.

- O método *dynamic* mostrou-se mais constante que o *guided*.

- Notou-se que as threads têm mais eficiência com o aumento da população e menos eficiência com aumento de *trials*.