

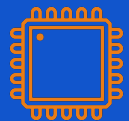
# Programação Paralela



T4

Simulador de Propagação de Vírus  
com OpenMP

Mariano Dorneles de Freitas – Talles Siqueira Ceolin



# Hardware

Especificações do computador utilizado nos testes:

<b>TÍTULO</b>	Acer Aspire A515-51G-58VH
<b>SISTEMA OPERACIONAL</b>	Ubuntu 18.04.4 LTS
<b>PROCESSADOR</b>	Intel Core i5-7200U 2.5 - 3.1 GHz
<b>MEMÓRIA</b>	8 GB



# Paralelismo

## Versão 1

```
[...]  
#pragma omp parallel for schedule(runtime)  
    for (int it = 0; it < n_trials; it++) {  
        int thread_num = omp_get_thread_num();  
        population[thread_num]->propagateUntilOut(population[thread_num]->centralPerson(),  
                                                    prob_spread[ip], rand);  
        percent_infected[ip] += population[thread_num]->getPercentInfected();  
    }  
[...]
```



# Paralelismo

## Versão 1

Na primeira versão, decidiu-se paralelizar o número de trials, uma vez que, no teste de maior volume, este número seria potencialmente alto. Foram utilizados schedule e chunks automáticos.

Então, criou-se um vetor para separar a população de cada uma das threads e, por fim, executá-las. Também foi modificada a forma de conferir a progressão do programa, economizando tempo de processamento.



# Paralelismo

## Versão 2

```
[...]  
#pragma omp parallel for schedule(runtime)  
    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);  
    }  
[...]
```



# Paralelismo

## Versão 2

Já na segunda versão do programa, utilizou-se a paralelização nas probabilidades, com intuito de dividir de maneira mais proporcional a quantidade de processamento para cada thread.

Assim como na primeira versão, foram utilizados `schedule` e `chunks` automáticos (decididos pelo compilador). Os resultados deste teste são dados de maneira não ordenada, uma vez que as threads trabalham de forma simultânea e os prints estão logo após o processamento.



# Resultados

\*Tempo calculado pelo comando *time* do terminal

\*\*Aceleração em relação ao original

Versão	População	Experimentos	Tempo*	Aceleração**
Original	30	5000	2m14s	-
Original	50	500	55s	-
Original	100	50	38s	-
Versão 1	30	5000	2m21s	0.95
Versão 1	50	500	43s	1.27
Versão 1	100	50	24s	1.58
Versão 2	30	5000	2m15s	0.99
Versão 2	50	500	42s	1.30
Versão 2	100	50	23s	1.65



# Conclusão

A partir dos resultados, podemos concluir que:

- Para testes de grande volume, a maneira com que foi implementada a paralelização não foi eficiente, retornando um resultado muito próximo (por vezes pior) da mesma operação serializada;
- Conforme os teste foram diminuindo, o desempenho do processamento paralelo foi se tornando mais evidente. A segunda versão mostrou ser uma solução melhor à versão 1, uma vez que todos seus speedups foram minimamente superiores.