Processos e Threads

Tabela de Resultados dos Experimentos

Nome do Programa	Mediana (mean)	Desvio padrão (σ)	User
contagem_sequencial	**4.268 s**	**0.314 s**	**4.356 s**
contagem_com_2processos	**1.291 s**	**0.055 s**	**2.593 s**
contagem_com_2threads	**1.739 s**	**0.262 s**	**3.465 s**
contagem_com_10threads	**1.532 s**	**0.368 s**	**6.078 s**

Configurações do computador onde os experimentos foram executados

Sistema Operacional Host:

Windows 11

Subsistema Linux utilizado:

WSL 2 — Ubuntu 22.04 LTS

Arquitetura x86_64

Hardware

Processador (CPU)

11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz

- 2 núcleos físicos
- 4 threads (processadores lógicos)
- Frequência base: 2,19 GHz
- Frequência durante a execução: ~2,44 GHz
- Cache L1: 160 KB
- Cache L2: 2,5 MB

- Cache L3: 6 MB
- Virtualização habilitada (necessário para WSL2)

Memória RAM

16 GB DDR4 (SODIMM)

- Velocidade: 3200 MT/s
- Disponível no momento do teste: ~4 GB
- Em cache: 4 GB
- Em uso (compactada): 11,6 GB (1,4 GB compactado)
- Reservada para hardware: 261 MB
- Slots usados: 2 de 2

Armazenamento

SSD NVMe

Modelo: SM2P41C3 NVMe ADATA 256GB

- Tipo: SSD NVMe
- Capacidade: 239 GB
- Disco do sistema: Sim
- Arquivo de paginação: Ativo
- Tempo médio de resposta: 4,6 ms

Ferramentas Utilizadas

Compilador:

GCC (build-essential) instalado via Ubuntu/WSL2

Ferramenta de benchmark:

hyperfine 1.18.0

Conclusão

A partir dos testes, ficou claro como o paralelismo influencia o desempenho da tarefa. A solução iterativa foi a mais lenta, o que já era esperado, porque ela percorre todo o arquivo usando apenas um fluxo de execução.Com as versões paralelas, o tempo caiu bastante. Entre todas, a solução com 2 processos foi a mais rápida. Isso aconteceu porque cada processo trabalha separado, com seu próprio espaço de memória, o que reduz disputa por recursos e permite que o sistema operacional distribua melhor a carga entre os núcleos da CPU.

A versão com 2 threads também melhorou o tempo em relação à sequencial, mas ainda ficou abaixo dos 2 processos. Isso faz sentido, já que as threads compartilham o mesmo espaço de memória e acabam competindo mais por cache e barramentos internos da máquina.

Já a versão com 10 threads teve um desempenho intermediário. Em teoria, mais threads deveriam acelerar o processo, mas na prática aparecem alguns limites. Um deles é o acesso ao arquivo, que não escala junto com o número de threads. Outro é a competição entre as próprias threads, que aumenta quando há muitas delas rodando ao mesmo tempo.

No geral, os resultados não batem exatamente com o cenário teórico ideal, mas isso é normal, porque vários fatores do hardware e do sistema influenciam: acesso ao SSD, escalonamento, uso de cache, tempo gasto criando e sincronizando threads, entre outros.

Mesmo assim, o comportamento geral foi o esperado. As soluções paralelas foram bem mais rápidas que a iterativa, e usar processos trouxe o melhor desempenho nessa tarefa. O trabalho mostrou que, dependendo do tipo de atividade, aumentar demais o número de threads pode até atrapalhar, e que é importante considerar as características do hardware antes de escolher como paralelizar.