Comparing Concurrency Models in Python: Threading vs Multiprocessing

Felipe Campelo Sabbado - 240000183
Rodrigo Paiva Calado – 250001513

Programação Avançada
Professora Maryam Abbasi

Escola Superior de Gestão
Instituto Politécnico de Santarém

MOTIVAÇÃO → DESAFIO DA PROGRAMAÇÃO CONCORRENTE MODERNA

↪ **Desempenho** → A programação concorrente é fundamental para que múltiplas tarefas sejam executadas simultaneamente, aumentando a eficiência de sistemas modernos com múltiplos núcleos.

↪ **Coordenação** → quando múltiplos threads interagem com recursos partilhados, são necessários mecanismos de sincronização para evitar inconsistências e garantir a correção dos dados.

↪ **Abordagem Tradicional** → o método mais comum utiliza *locks* para garantir a exclusão mútua, permitindo que apenas uma *thread* aceda ao recurso por vez.

↪ **Alternativas** → devido às limitações dos *locks*, algoritmos *lock-free* e não bloqueantes têm sido estudados para oferecer maior escalabilidade através de operações atômicas de hardware (como o *CAS – Compare-and-Swap*).

PROBLEMA → LIMITAÇÕES NOS MECANISMOS DE SINCRONIZAÇÃO

↪ **Limitações dos *Locks*** → Apesar de intuitivos, podem introduzir gargalos de performance e problemas críticos.

⚠ Estima-se que 30% dos bugs em programação paralela sejam ***deadlocks***.

⚠ **Inversão de Prioridade** e ***Race Conditions*** também são problemas comuns que dificultam a deteção e correção de erros em grandes sistemas.

↪ **Limitações do *Lock-Free*** → Embora eliminem o bloqueio explícito, são complexas de desenhar, validar e implementar corretamente.

↪ **Questão de investigação** → Determinar se estruturas *lock-free* são realmente mais rápidas e escaláveis que as baseadas em *locks* em cenários de alta competitividade.

METHODOLOGY → EXPERIMENTAL

Hardwares:

- PAIVA-DESKTOP: Windows 11 (AMD Ryzen 7 7700, 32GB RAM DDR5)
- FELIPE-LAPTOP: Windows 11 (AMD Ryzen 7 5700U, 12GB RAM DDR4)
- PAIVA-LAPTOP: Windows 10 (AMD Ryzen 7 5800H, 16GB RAM DDR4)

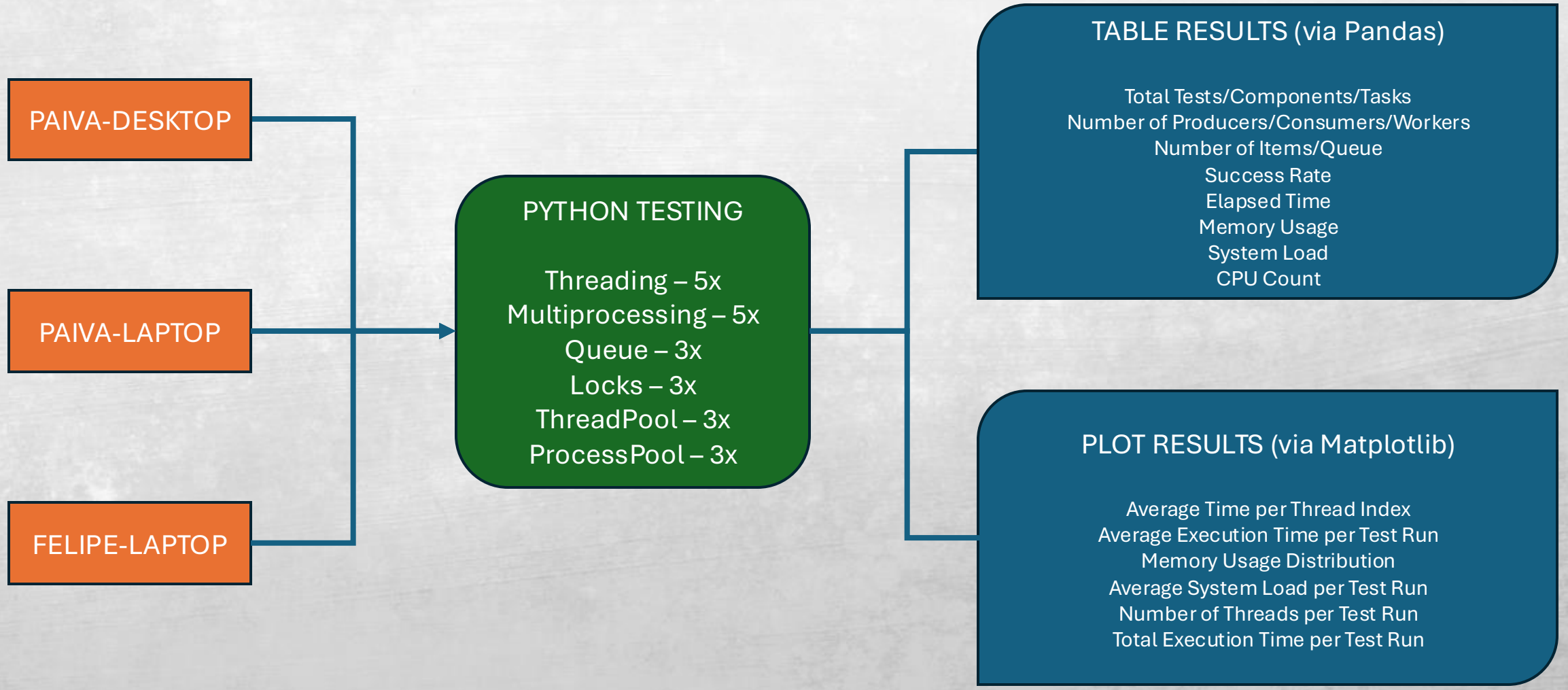
Python Libraries Used:

- Pandas (dataframe)
- Matplotlib (pyplot)
- Threading (Thread, current_thread, Lock, Semaphore)
- Multiprocessing (freeze_support, cpu_count, current_process)
- Queue (queue, empty)
- Concurrent.futures (ThreadPoolExecutor, ProcessPoolExecutor)
- Numpy (mean, std, array, number, min, max)
- Psutil (Process().memory info().rss)

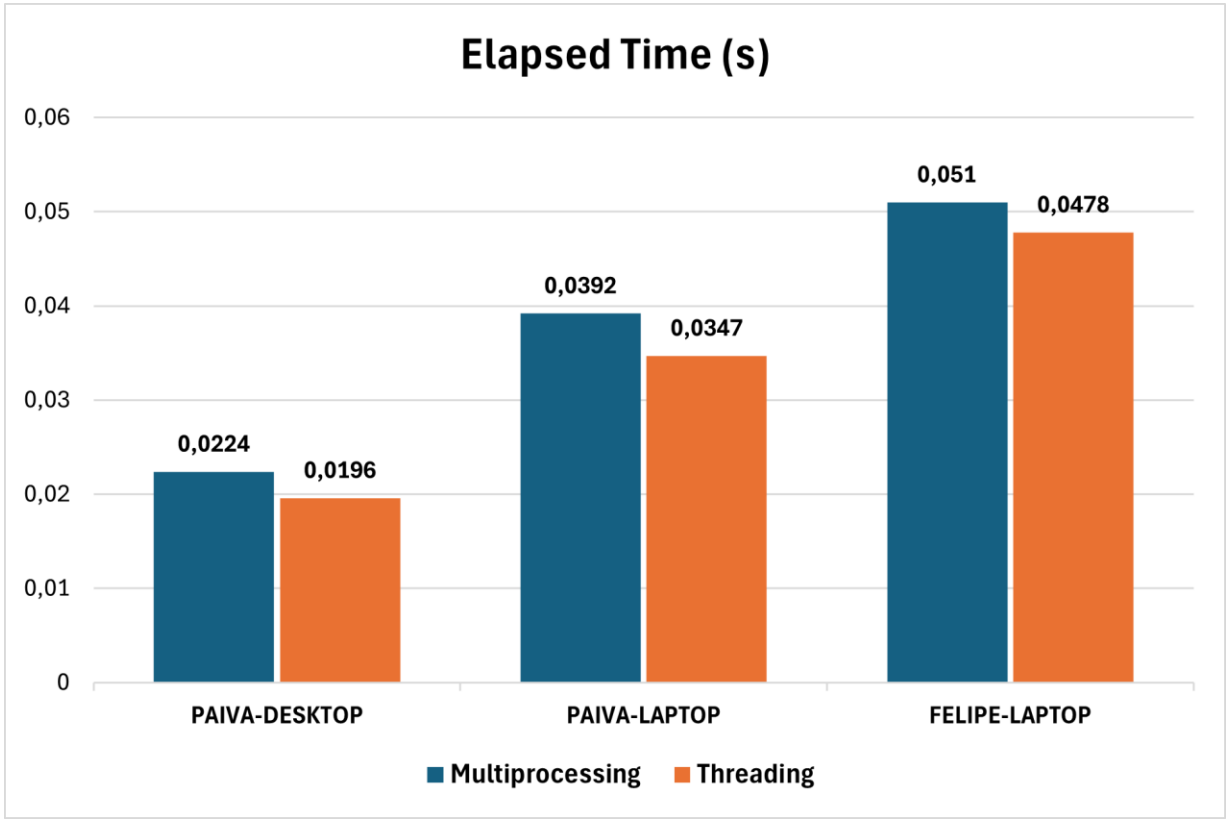
Realized Tests:

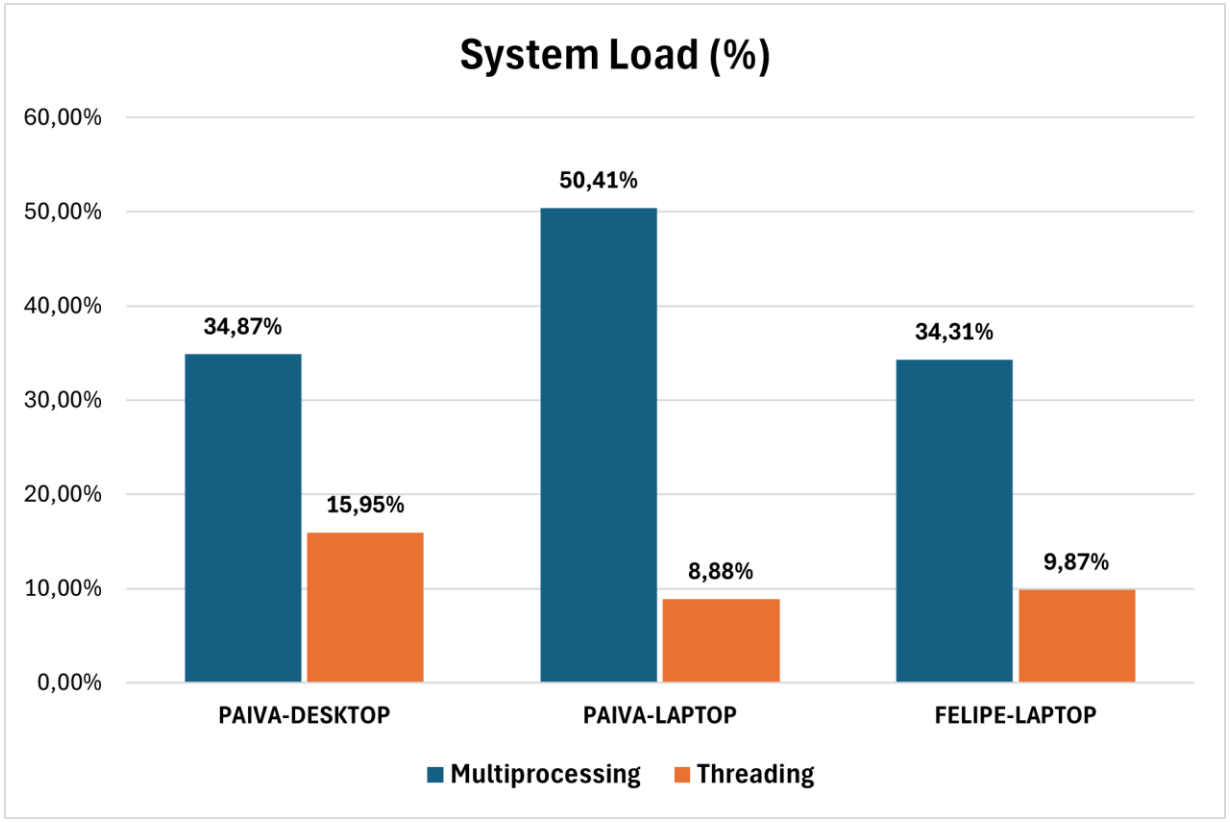
- Threading - 5x - Scalable threading intensive calculation tests (summing from 1 to a large random number), using different numbers of threads
- Multiprocessing - 5x - CPU-bound tests, multiple processes, intensive calculation (summing from 1 to a large random number) by each process, exploiting multiple CPU cores
- Queue - 3x - Comparative tests of threaded and multiprocessing queue systems, including producer-consumer, workflows, and CPU-intensive tasks
- Locks - 3x - Thread and multiprocess synchronization tests, including race conditions, locks, semaphores, condition variables, and multiprocessing
- ThreadPool - 3x - Comparative tests of ThreadPoolExecutor with multiple executions of simple and I/O-bound tasks
- ProcessPool - 3x - Comparative tests of ProcessPoolExecutor with CPU-intensive tasks, parallel processing, and distributed search

METHODOLOGY DIAGRAM EXPLANATION



Results – Multiprocessing & Threading





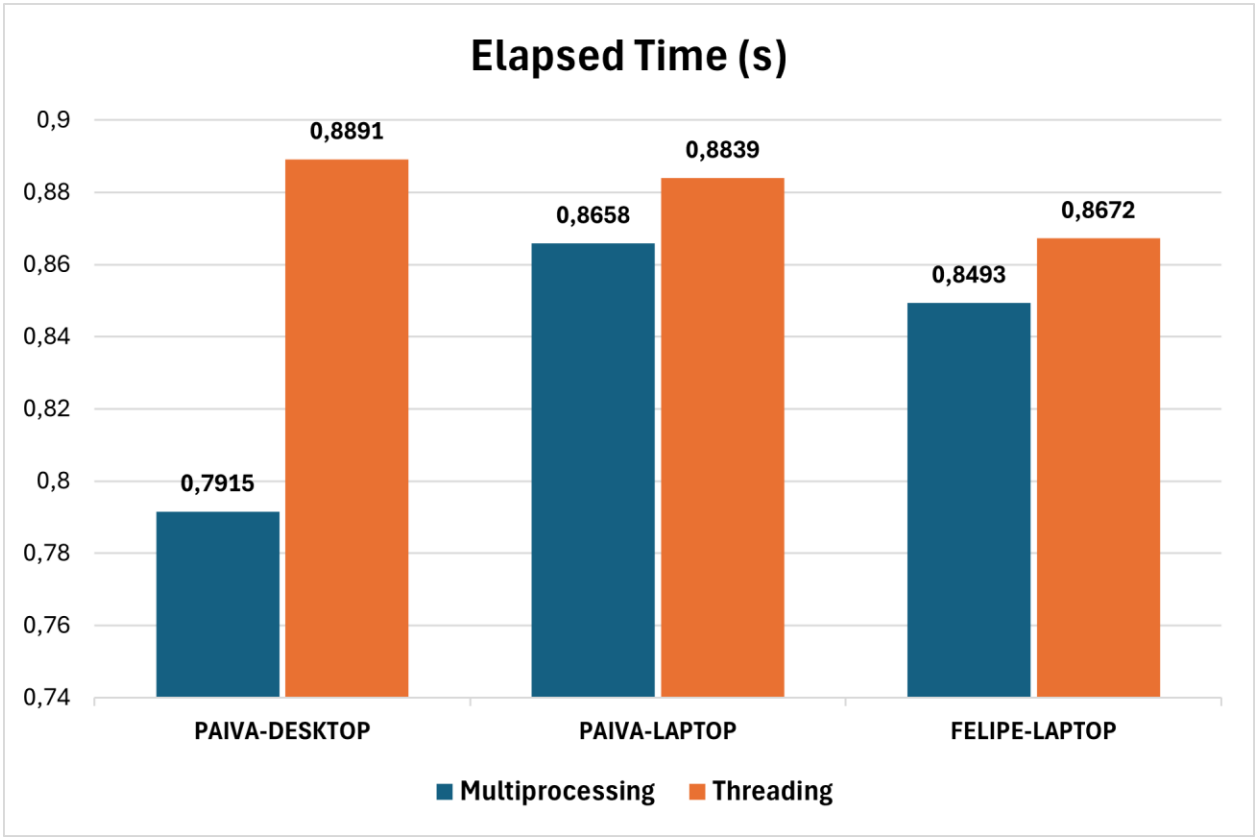
Computer	Total Tests	Elapsed Time	Memory Usage	System Load	CPU Count
PAIVA-DESKTOP	28	0.0224 s	0.01 GB	34.87%	16
PAIVA-LAPTOP	28	0.0392 s	0.01 GB	50.41%	16
FELIPE-LAPTOP	27	0.0510 s	0.01 GB	34.31%	16

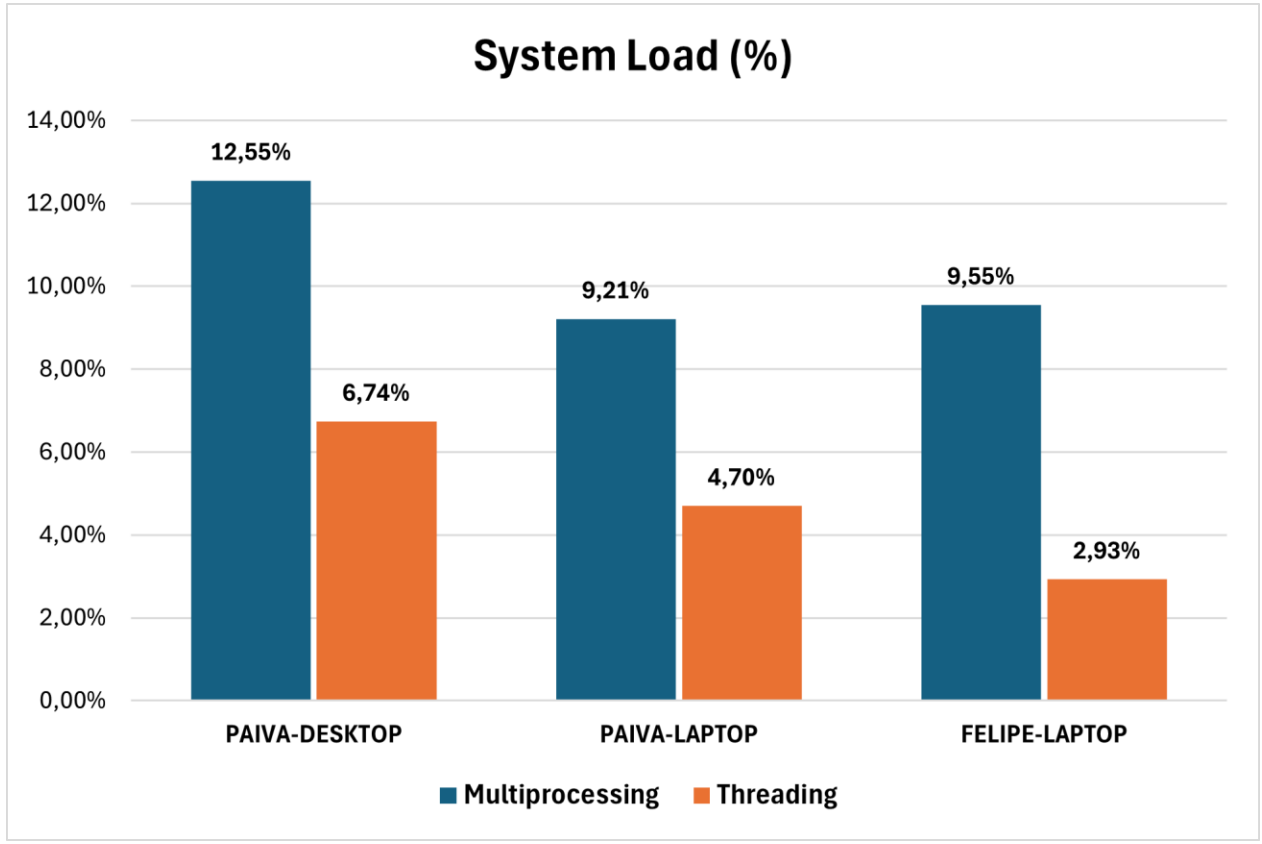
Multiprocessing Table Results on All Hardwares

Computer	Total Tests	Elapsed Time	Memory Usage	System Load	CPU Count
PAIVA-DESKTOP	215	0.0196 s	0.01 GB	15.95%	16
PAIVA-LAPTOP	220	0.0347 s	0.01 GB	8.88%	16
FELIPE-LAPTOP	215	0.0478 s	0.01 GB	9.87%	16

Theading Table Results on All Hardwares

Results – Queues





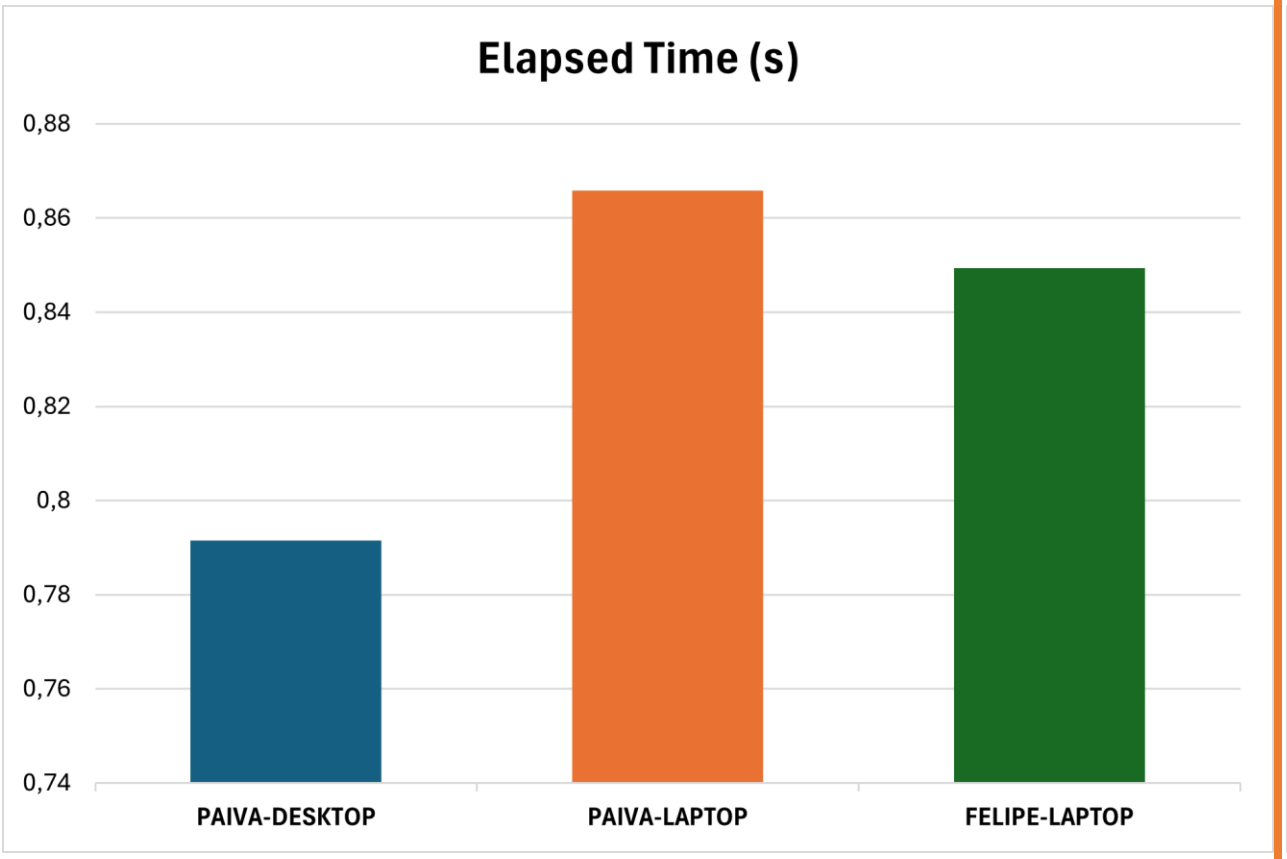
Computer	Total Components	Prod./Cons./Work.	Items/Queue	Elapsed Time	Memory Usage	System Load	CPU Count
PAIVA-DESKTOP	42	4/4/0	84/5	0.7915 s	0.01 GB	12.55%	16
PAIVA-LAPTOP	42	4/4/0	84/5	0.8658 s	0.01 GB	9.21%	16
FELIPE-LAPTOP	42	4/4/0	84/5	0.8493 s	0.01 GB	9.55%	16

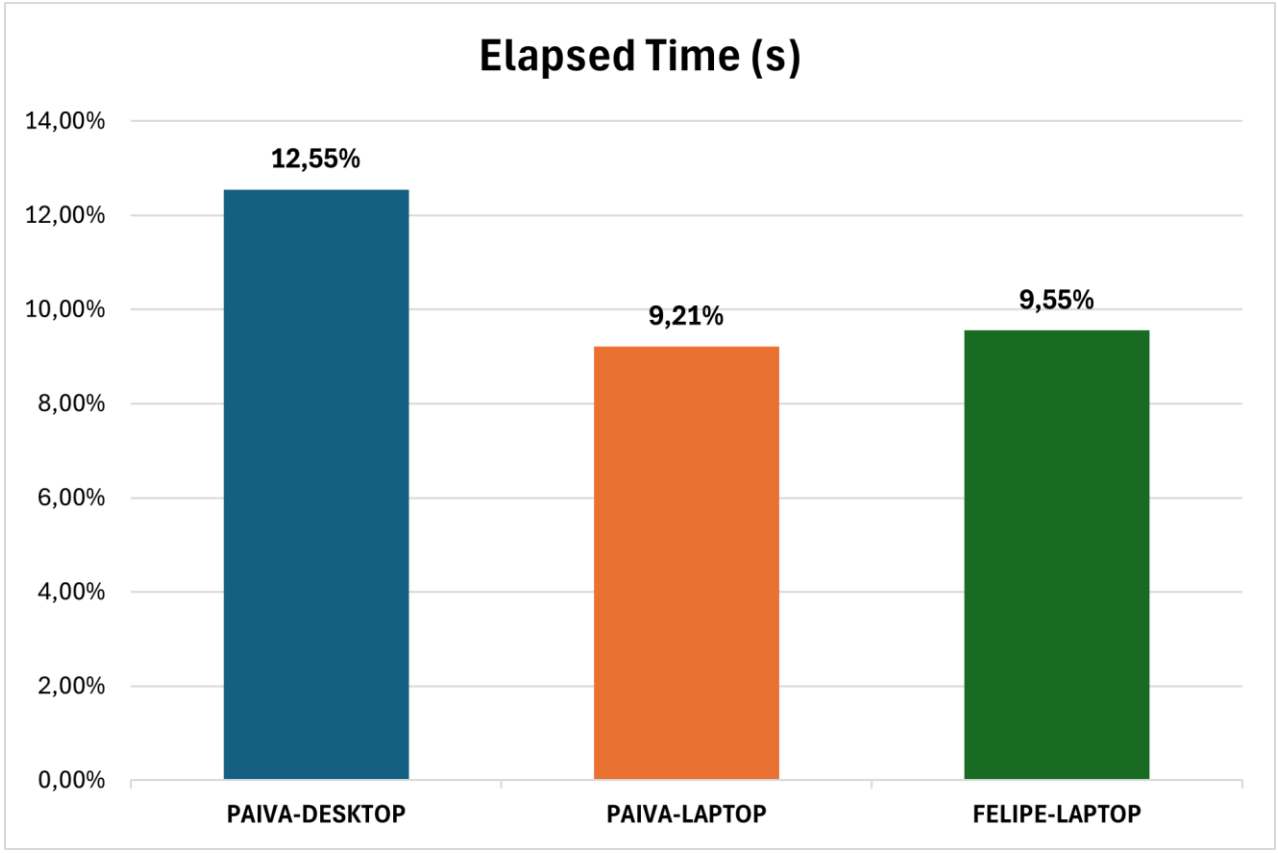
Queue (Multiprocessing) Table Results on All Hardwares

Computer	Total Components	Prod./Cons./Work.	Items/Queue	Elapsed Time	Memory Usage	System Load	CPU Count
PAIVA-DESKTOP	42	4/5/5	108/5	0.8891 s	0.01 GB	6.74%	16
PAIVA-LAPTOP	42	4/5/5	108/5	0.8839 s	0.01 GB	4.70%	16
FELIPE-LAPTOP	42	4/5/5	108/5	0.8672 s	0.01 GB	2.93%	16

Queue (Threading) Table Results on All Hardwares

Results – Locks

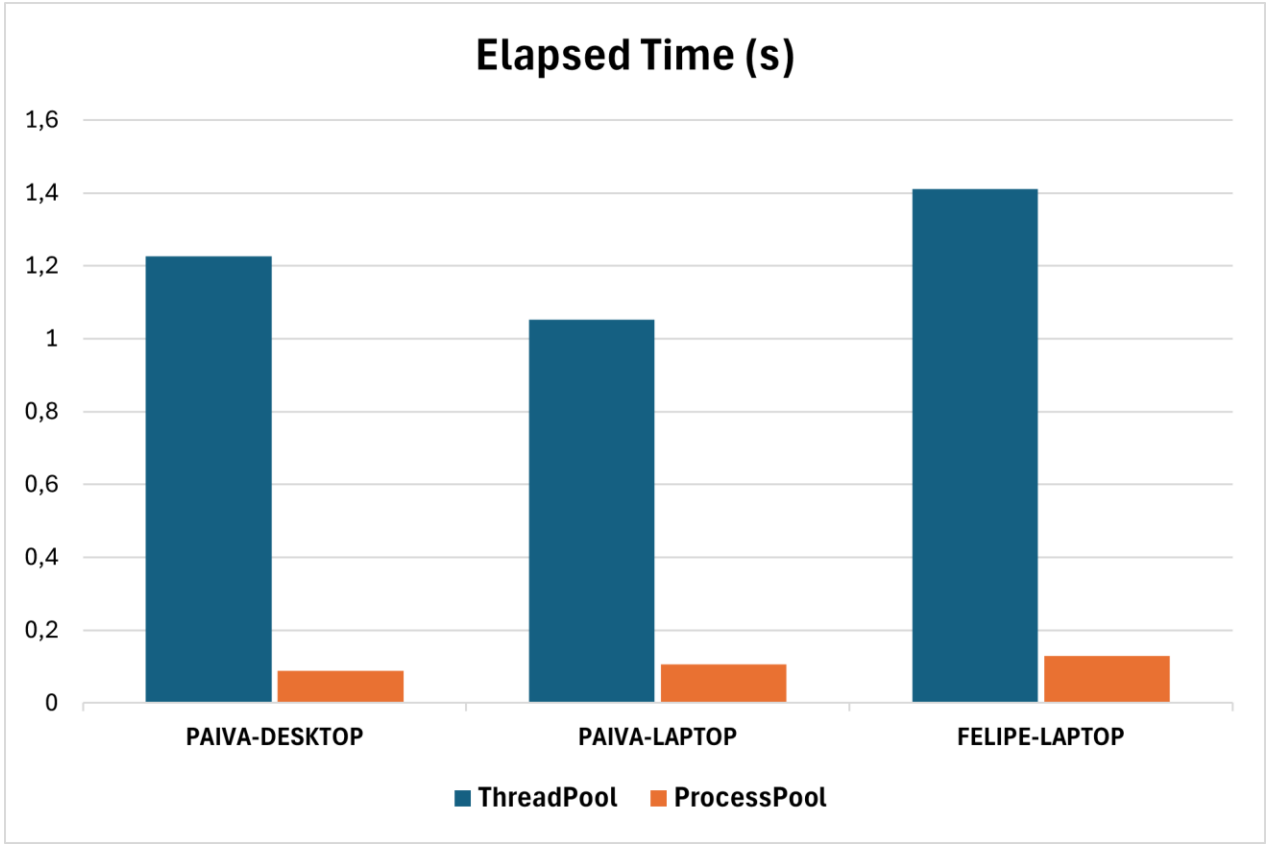


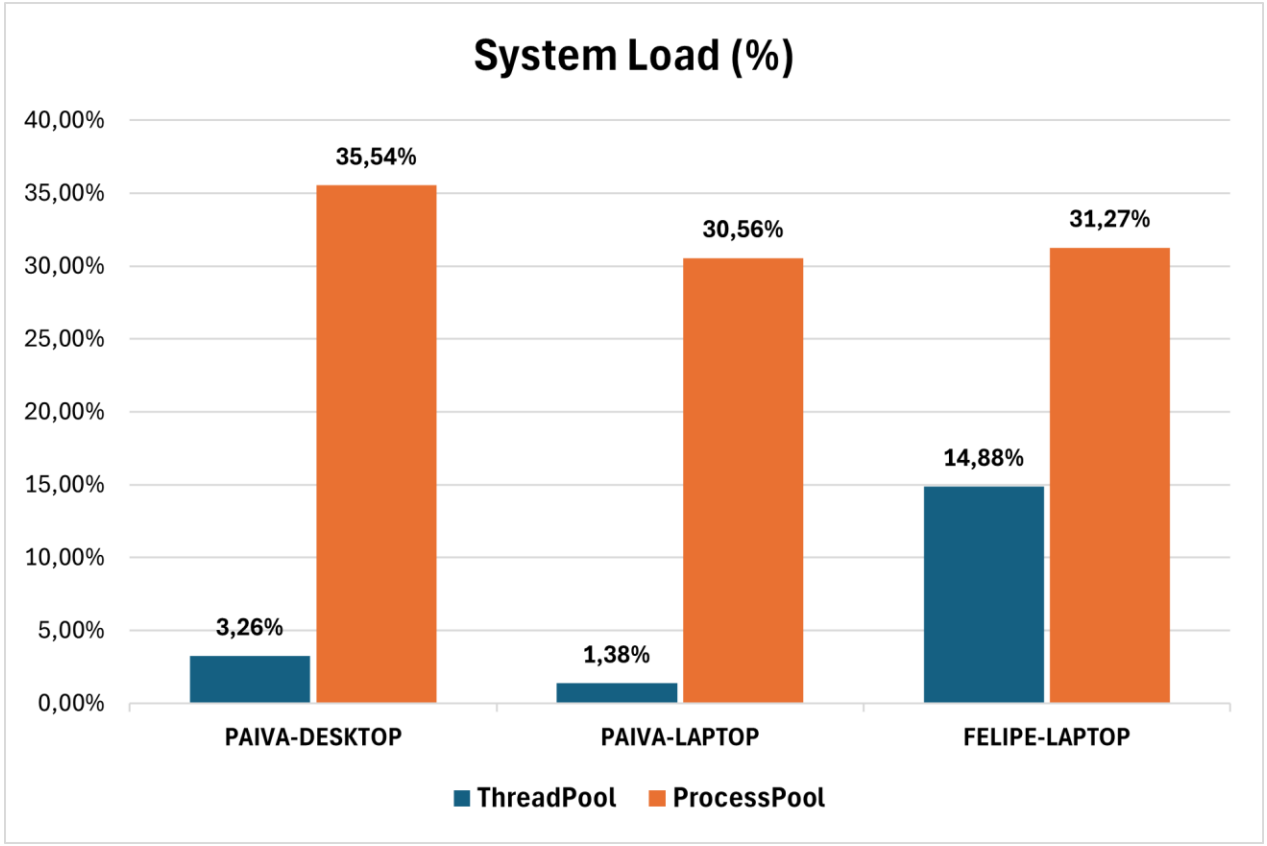


Computer	Total Tests	Success Rate	Elapsed Time	Memory Usage	System Load
PAIVA-DESKTOP	15	80%	0.9763 s	0.10 GB	9.55%
PAIVA-LAPTOP	15	80%	1.0597 s	0.095 GB	3.43%
FELIPE-LAPTOP	15	80%	1.2342 s	0.092 GB	5.81%

Locks Table Results on All Hardwares

Results – ThreadPool & ProcessPool





Computer	Total Tasks	Elapsed Time	Memory Usage	System Load
PAIVA-DESKTOP	11	1.2254 s	0.098 GB	3.26%
PAIVA-LAPTOP	11	1.0527 s	0.093 GB	1.38%
FELIPE-LAPTOP	11	1.4099 s	0.091 GB	14.88%

ThreadPool Table Results on All Hardwares

Computer	Total Tasks	Elapsed Time	Memory Usage	System Load
PAIVA-DESKTOP	9	0.0884 s	0.098 GB	35.54%
PAIVA-LAPTOP	9	0.1065 s	0.087 GB	30.56%
FELIPE-LAPTOP	9	0.1302 s	0.090 GB	31.27%

ProcessPool Table Results on All Hardwares

Conclusão » Principais Constatações

***Hardware* vs. Integridade:** Mais potência de CPU (frequência ou núcleos) não resolve *race conditions*.

Superioridade do *Multiprocessing*: Para tarefas *CPU-bound*, o modelo de multiprocessamento escalou melhor que *Threading*, superando as limitações do GIL. GIL.

Eficiência de recursos: Uso de *Pools* foi a abordagem mais estável, mantendo um consumo de memória entre 90~98 MB.

Limitações Python: Estruturas puramente *lock-free* são limitadas ao nível do interpretador pois ele atua como um mecanismo mestre baseado em *locks*.

Trabalho Futuro

Exploração do Python 3.12+: Investigar se o "*Per-Interpreter GIL*" consegue reduzir a diferença de desempenho entre threads e processos.

Computação distribuída: Expandir os testes para sistemas distribuídos.

Programação assíncrona: Integrar bibliotecas para comparar o *multitasking* cooperativo com os modelos preemptivos analisados.