Trabalho 2

Alunos: Caio Henrique Ramos Rufino (GRR20224386) e
Frank Wolff Hannemann (GRR20224758)

O código implementa o algoritmo **`multi_partition`**, que realiza o particionamento paralelo de um vetor de entrada (`Input`) em múltiplas faixas determinadas por um vetor de partições (`P`). A lógica é dividida em duas fases principais, ambas otimizadas com o uso de threads.

Contagem de Elementos nas Faixas

- **Objetivo**: Contar quantos elementos de `Input` pertencem a cada faixa definida por `P`.
- **Detalhes**:
    - Cada thread trabalha sobre um subconjunto do vetor `Input` e utiliza a função de busca binária (`binary_search`) para determinar a faixa correspondente de cada elemento.
    - Os resultados são acumulados no vetor `range_count`, onde cada posição representa o número de elementos na faixa correspondente à posição do vetor.
- **Segurança**: Para evitar condições de corrida, o incremento de valores no vetor `range_count` é realizado utilizando operações atômicas (`__atomic_fetch_add`).

Construção do Vetor `Output`

- **Objetivo**: Preencher o vetor `Output` com os elementos de `Input`, particionados corretamente em faixas consecutivas.
- **Detalhes**:
    - As threads novamente processam subconjuntos de `Input`, usando `binary_search` para determinar a faixa de cada elemento.
    - Os elementos são inseridos no vetor `Output` nas posições apropriadas, com base nos índices de inserção armazenados no vetor atômico `range_index`.
- **Segurança**: As inserções no vetor `Output` utilizam índices gerenciados com operações atômicas (`atomic_fetch_add`), garantindo que múltiplas threads insiram elementos simultaneamente sem sobrescrever valores.

Uso de Pool de Threads

O programa utiliza um **pool de threads** para executar a função `multi_partition` de forma eficiente:

1. **Inicialização**:

    - As threads são criadas apenas uma vez e reutilizadas durante toda a execução, reduzindo a sobrecarga associada à criação/destruição de threads.
2. **Distribuição do Trabalho**:
    - Cada thread recebe um intervalo específico de elementos de `Input` para processar, garantindo uma divisão equilibrada do trabalho.
3. **Sincronização**:
    - Uma barreira (`pthread_barrier_t`) é usada para sincronizar todas as threads ao final de cada fase (contagem e construção), garantindo que todas concluam suas tarefas antes de passar para a próxima fase.

Características de Arquitetura

1. **Cores Físicos**:
    - Possui **4 núcleos físicos**, que operam de forma independente e permitem o processamento simultâneo de múltiplas threads.
2. **Threads por Núcleo**:
    - Cada núcleo suporta **1 thread** (sem hyper-threading), garantindo que cada thread tenha acesso total aos recursos do núcleo.
3. **Frequência de Operação**:
    - A frequência base é de **2.80 GHz**, adequada para computações intensivas.

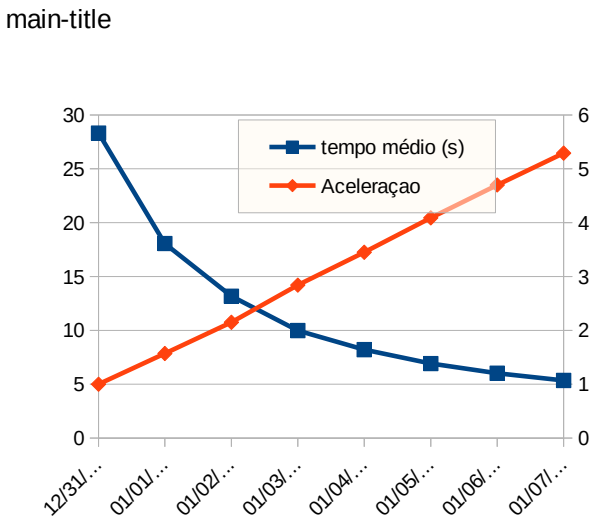
Limitações

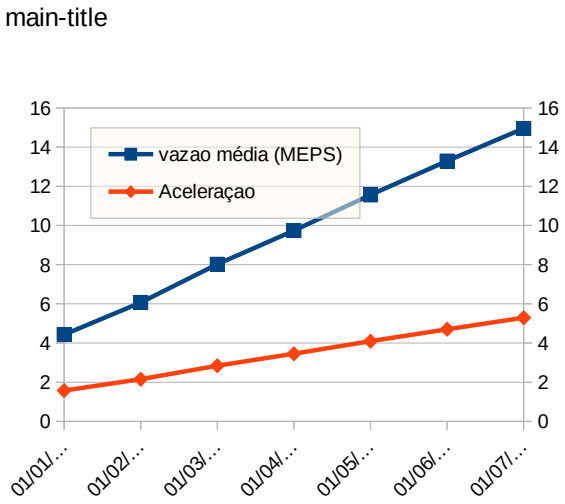
1. **Falta de Hyper-Threading**:
    - Cada núcleo processa uma única thread por vez, limitando o potencial de paralelismo em cenários com mais de 4 threads.
2. **Idade da Arquitetura**:
    - Lançada em 2007, pertence à geração Penryn, ficando atrás em desempenho e eficiência energética em comparação com CPUs modernas.
3. **Ausência de Cache L3**:
    - Pode impactar negativamente o desempenho em programas com acesso intensivo à memória compartilhada entre núcleos.

Nota sobre o desempenho:

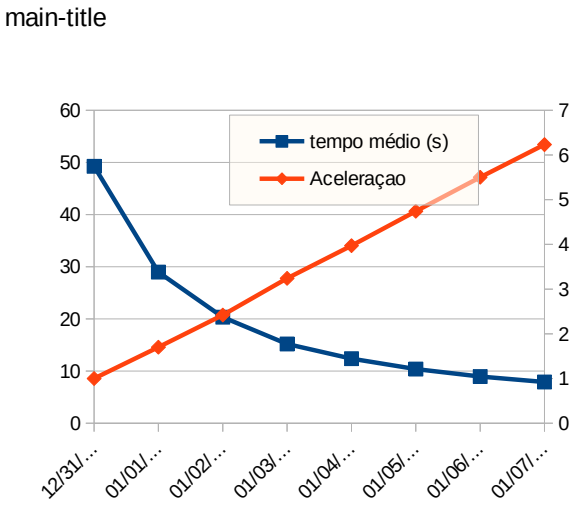
Durante a implementação desse programa, fizemos todos os testes no meu notebook e tivemos uma performance MUITO superior a que conseguimos rodando no cluster, que chegou relativamente perto da implementação do professor. Não sei ao certo o motivo disso, talvez a diferença de hardware simplesmente. De qualquer forma, as tabelas contém os valores das execuções nesse cluster.

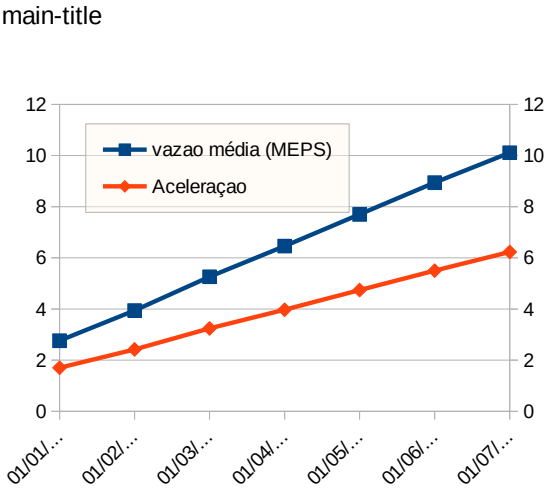
Gráficos Parte A





Gráficos Parte B





LSCPU

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 38 bits physical, 48 bits virtual
CPU(s): 8
On-line CPU(s) list: 0-7
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s): 2
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 23
Model name: Intel(R) Xeon(R) CPU E5462 @ 2.80GHz
Stepping: 6

CPU MHz: 2792.839
BogoMIPS: 5585.67
Virtualization: VT-x
L1d cache: 256 KiB
L1i cache: 256 KiB
L2 cache: 24 MiB
NUMA node0 CPU(s): 0-7
Vulnerability Itlb multihit: KVM: Mitigation: VMX disabled
Vulnerability L1tf: Mitigation; PTE Inversion; VMX EPT disabled
Vulnerability Mds: Vulnerable: Clear CPU buffers attempted, no microcode; SMT disabled
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Full generic retpoline, STIBP disabled, RSB filling
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Not affected
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl cpuid aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 lahf_lm pti tpr_shadow vnmi flexpriority vpid dtherm

Topology