# Trabalho 3

Lucas Emanuel de Oliveira Santos – GRR20224379
CI1316 – Programação Paralela
*Universidade Federal do Paraná – UFPR*
*Curitiba, novembro de 2024*

## 1. Implementação

O código implementa um algoritmo de multi-partição distribuído utilizando a biblioteca MPI (Message Passing Interface) para realizar uma tarefa de particionamento de um conjunto de números de forma eficiente. A partir de um vetor de entrada de números aleatórios (Input), o código divide os elementos em diferentes partições com base nos pontos de partição fornecidos no vetor P. O objetivo é distribuir essas partições entre os processos do MPI para que cada processo trate uma parte da entrada, realizando uma operação de comparação para alocar cada elemento na partição correta.

O código começa com a inicialização do ambiente MPI e a obtenção da posição de cada processo no comunicador MPI (rank) e o número total de processos (size). Cada processo gera uma parte dos números aleatórios que compõem o vetor de entrada. O vetor P contém os pontos de divisão das partições, que são transmitidos de forma eficiente entre os processos através da operação de MPI_Bcast. Cada processo então utiliza a função binary_search para alocar seus elementos nas partições apropriadas. A distribuição das partições é feita de maneira balanceada entre os processos, garantindo uma carga de trabalho equitativa.

A função central deste código é multi_partition_mpi, que realiza o particionamento propriamente dito. Ela usa contadores auxiliares para armazenar quantos elementos pertencem a cada partição e, em seguida, distribui esses elementos entre os processos com o uso de MPI_Alltoall e MPI_Alltoallv. Esses métodos permitem que os processos enviem e recebam dados entre si, de maneira eficiente, sem sobrecarregar nenhum processo individualmente. Após o particionamento e a distribuição dos dados, o código verifica se a alocação foi feita corretamente utilizando a função verifica_particoes, que valida se os elementos foram alocados nas partições corretas.

Por fim, o código calcula o tempo de execução e o throughput da aplicação. O tempo total de execução é medido utilizando um cronômetro, e o throughput é calculado como o número total de operações divididas pelo tempo total em segundos. Esse valor de throughput fornece uma medida de desempenho da aplicação, indicando a quantidade de operações realizadas por segundo. Esse tipo de medição é fundamental para avaliar a eficiência do algoritmo em um ambiente distribuído e verificar o impacto da paralelização sobre a performance geral da aplicação.

## 2. Apêndice

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: 2792.819

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

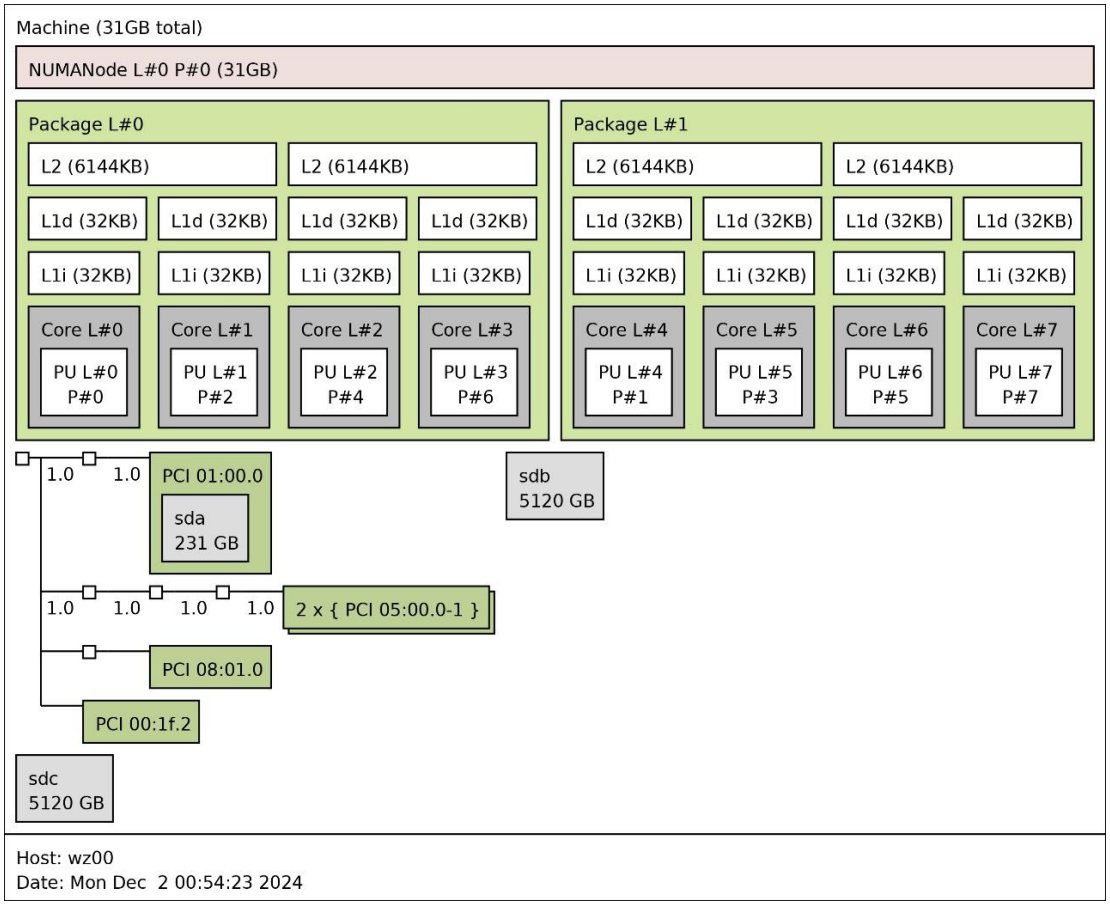
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization

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


Machine (31GB total)
NUMANode L#0 P#0 (31GB)
Package L#0
L2 (6144KB)
L2 (6144KB)
L1d (32KB)
L1d (32KB)
L1d (32KB)
L1d (32KB)
L1i (32KB)
L1i (32KB)
L1i (32KB)
L1i (32KB)
Core L#0
PU L#0 P#0
Core L#1
PU L#1 P#2
Core L#2
PU L#2 P#4
Core L#3
PU L#3 P#6
Package L#1
L2 (6144KB)
L2 (6144KB)
L1d (32KB)
L1d (32KB)
L1d (32KB)
L1d (32KB)
L1i (32KB)
L1i (32KB)
L1i (32KB)
L1i (32KB)
Core L#4
PU L#4 P#1
Core L#5
PU L#5 P#3
Core L#6
PU L#6 P#5
Core L#7
PU L#7 P#7
1.0
1.0
PCI 01:00.0
sda
231 GB
sdb
5120 GB
1.0
1.0
1.0
1.0
2 x { PCI 05:00.0-1 }
PCI 08:01.0
PCI 00:1f.2
sdc
5120 GB
Host: wz00
Date: Mon Dec 2 00:54:23 2024