Trabalho 2 - Programação Paralela

Bianca Mendes Francisco
bmf23@inf.ufpr.br

Nathália Nogueira Alves
nna23@inf.ufpr.br

Universidade Federal do Paraná
Curitiba, Paraná, Brasil

O objetivo deste relatório é detalhar a implementação do algoritmo KNN utilizando MPI, assim como tempos coletados e resultados concluídos. O código-fonte e resultados completos estão disponíveis no repositório do projeto.

1. IMPLEMENTAÇÃO DO ALGORITMO

O código-fonte está dividido em duas partes principais: o algoritmo KNN e a função principal, responsável por gerenciar os processos MPI.

O KNN foi implementado com base em conceitos discutidos em aula. O algoritmo utiliza uma *max heap* para armazenar as menores distâncias encontradas. A estrutura é atualizada por meio das operações *insert* (inserção padrão na heap) e *decreaseMax* (substitui a raiz quando um valor menor é recebido). O cálculo das distâncias é realizado por meio de dois loops aninhados: o primeiro percorre os pontos de interesse e o segundo, para cada um deles, percorre os pontos de consulta.

A função principal é responsável por:

1. Gerar aleatoriamente os pontos utilizados no cálculo
2. Distribuir os dados entre os processos MPI a partir do processo principal
3. Executar o KNN em cada processo sobre sua respectiva seção de pontos
4. Reunir os resultados parciais no processo principal

2. DESCRIÇÃO DO PROCESSADOR

A figura 1 apresenta a topologia do sistema obtida por meio do comando *lstopo*. O processador possui um único nó *NUMA* com 63GB de memória distribuídos entre dois pacotes físicos.

Cada pacote contém 4 núcleos físicos, totalizando 8 núcleos no sistema. Cada núcleo dispõe de caches L1 e um cache L2 compartilhado.

Além disso, a figura mostra o mapeamento das unidades de processamento lógico, que correspondem aos *threads* disponíveis para execução paralela.

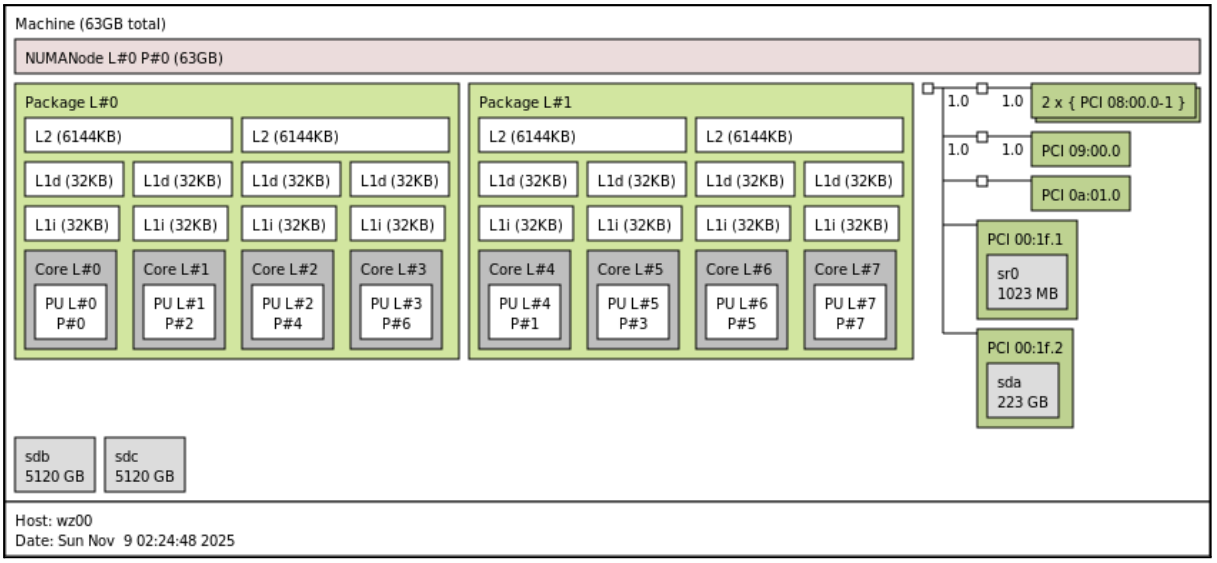

Figura 1: saída do comando lstopo.

3. EXPERIÊNCIAS REALIZADAS

Cada experimento foi realizado com os parâmetros:

- Conjunto principal (Q): 128 pontos
- Conjunto de consulta (P): 400 000 pontos
- Dimensionalidade: 300 dimensões
- Número de vizinhos (d): 1024 pontos mais próximos de P para cada ponto de Q

Foram conduzidas três configuraçòes de execução:

1. Apenas 1 processo MPI
2. 8 processos MPI no mesmo host
3. 8 processos MPI em 4 hosts diferentes

A. RESULTADOS

Para cada configuração foram medidos:

- o tempo de envio de dados a partir do processo principal
- o tempo de computação do KNN
- o tempo de união dos resultados no processo principal

Os valores completos estão disponíveis na pasta /resultados/ do repositório do projeto. Cada experimento foi executado 10 vezes e o valor final de cada métrica é a média dos tempos obtidos. Nos testes paralelos, o tempo total de execução foi definido como o maior tempo observado entre os processos MPI.

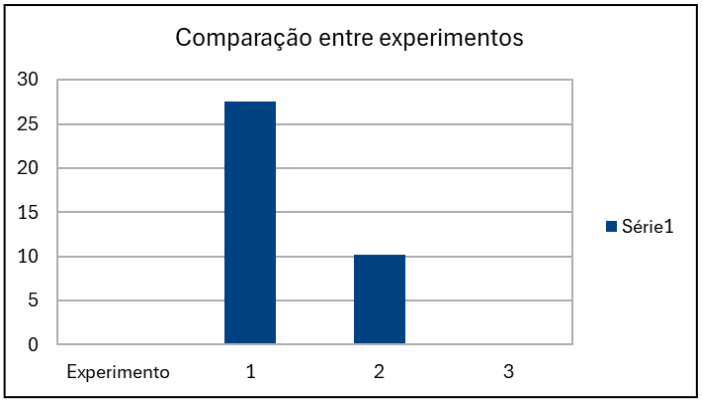

Figura 2: comparação entre tempos de execução.

4. CONCLUSÃO

Com base nos resultados obtidos, observa-se uma redução significativa no tempo de execução à medida que o número de processos MPI aumenta. O Experimento 1, executado de forma sequencial, apresentou o maior tempo médio de processamento, enquanto o Experimento 2, com oito processos no mesmo host, reduziu consideravelmente esse tempo devido à paralelização local. Por fim, o Experimento 3, distribuindo os oito processos entre quatro hosts distintos, apresentou o comportamento esperado, com desempenho ainda melhor em razão da divisão equilibrada da carga de trabalho entre diferentes máquinas. Esses resultados confirmam a eficácia da abordagem paralela e distribuída na aceleração do cálculo do KNN em grandes volumes de dados.

APÊNDICE A) Saída do comando 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:                 2793.307
BogoMIPS:                5586.61
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: Split huge pages
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