UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO

Programação Paralela e Distribuída

Exercício Programa - EP4

Professor: Hermes Senger

Aluno: Ivan Duarte Calvo, 790739

São Carlos, SP, 2023

1 Metodologia

Inicialmente, foi realizada a paralelização utilizando MPI. A parte mais importante da implementação utilizada é:

```
1  MPI_Scatter(a, num_local_size, MPI_DOUBLE, ap, num_local_size,
    MPI_DOUBLE, 0, MPI_COMM_WORLD);
2    MPI_Scatter(b, num_local_size, MPI_DOUBLE, bp, num_local_size,
    MPI_DOUBLE, 0, MPI_COMM_WORLD);
3
4    for(int i=0;i<num_local_size;i++)
5    cp[i] = ap[i]+bp[i];
6
7    MPI_Gather(cp, num_local_size, MPI_DOUBLE, c, num_local_size,
    MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

No código exibido, existem os vetores a e b que serão somados e os resultados atribuídos ao vetor c. Há também os vetores auxiliares, para serem calculados em paralelo, são os vetores: ap, bp e cp.

A função *MPI_Scatter* divide os vetores principais para os vetores auxiliares em cada processador, após isso o laço *for* principal é executado e são feitas as somas para cada posição do vetor. Finalmente, é chamada a função *MPI_Gather* que é responsável por juntar os valores nos vetores auxiliares cp em cada processador e finalmente atribuir para o vetor principal c.

2 Hardware

O código paralelizado foi rodado localmente em um computador pessoal, com a utilização de 1, 2, e 4*threads*. Por fim, os respectivos tempos de execução são utilizados para a composição dos gráficos de *Speedup* e Eficiência, e para as comparações necessárias.

Para avaliar o hardware em que o código foi submetido foi utilizado o comando *lscpu* que retorna informações sobre a CPU utilizada no problema. Neste exercício, a CPU utilizada foi:

```
1 Architecture:            x86_64
2   CPU op-mode(s):        32-bit, 64-bit
3   Address sizes:         43 bits physical, 48 bits virtual
4   Byte Order:            Little Endian
```

```
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx
    CPU family:          23
    Model:               24
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            1
    Frequency boost:     enabled
    CPU max MHz:         2100.0000
    CPU min MHz:         1400.0000
    BogoMIPS:            4192.22
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep
   mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
   mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_g
                         ood nopl nonstop_tsc cpuid extd_apicid
   aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2
   movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy
                         svm extapic cr8_legacy abm sse4a misalignsse 3
   dnowprefetch osvw skinit wdt tce topoext perfctr_core perfctr_nb
   bpext perfctr_llc mwaitx cpb hw_pstate ssbd ibpb vmmc
                         all fsgsbase bmi1 avx2 smep bmi2 rdseed adx
   smap clflushopt sha_ni xsaveopt xsavec xgetbv1 xsaves clzero irperf
   xsaveerptr arat npt lbrv svm_lock nrip_save tsc_scal
                         e vmcb_clean flushbyasid decodeassists
   pausefilter pfthreshold avic v_vmsave_vmload vgif overflow_recov
   succor smca sev sev_es
Virtualization features:
  Virtualization:        AMD-V
Caches (sum of all):
  L1d:                   128 KiB (4 instances)
  L1i:                   256 KiB (4 instances)
  L2:                    2 MiB (4 instances)
  L3:                    4 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-7
```

Como pode ser observado, a CPU utilizada possui um total de 4 núcleos físicos, portanto, esse foi o máximo a ser utilizado no problema.

3 Resultados

O algoritmo foi submetido para execução no computador pessoal, com o *hardware* informado previamente, utilizando 1, 2 e 4 processadores e com vetores no tamanho 80000. Os resultados obtidos foram os seguintes:

Tabela 1: Execução no computador pessoal

nº de *threads*	Tempo de Execução	*Speedup*	Eficiência
1	0.259589	1	1
2	0.271993	0.954396	0.477198
4	0.304110	0.853602	0.213401

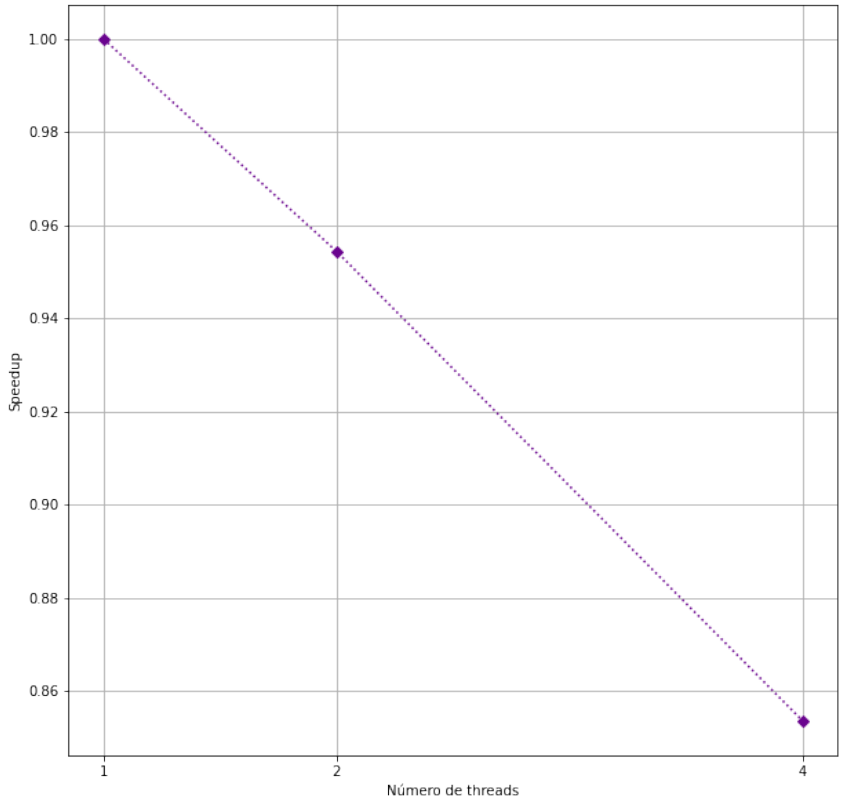

Figura 1: Gráfico de *Speedup*.

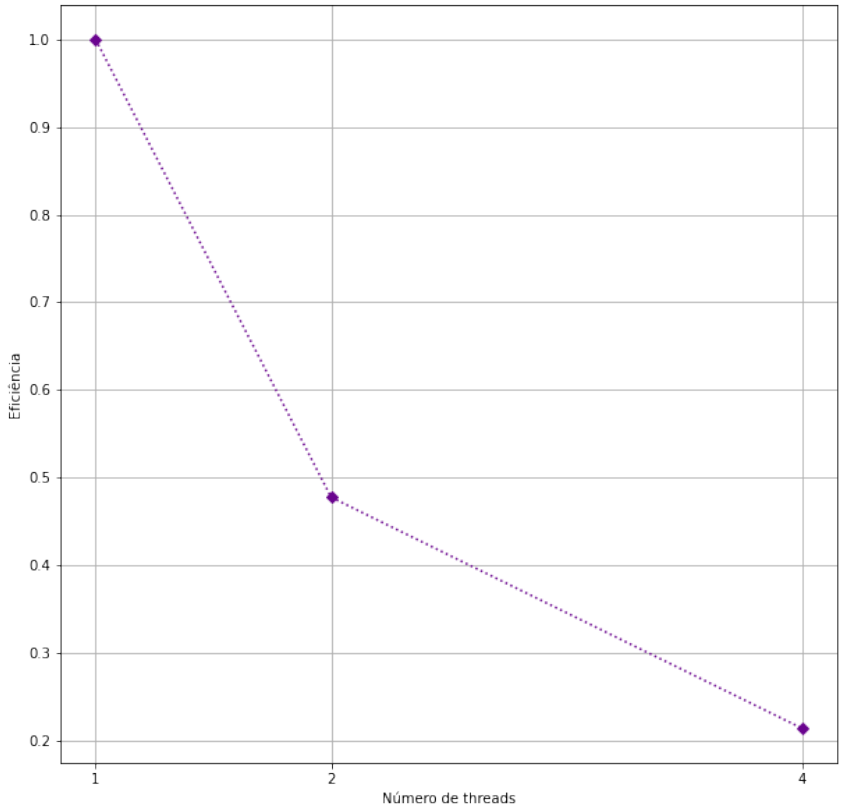

Figura 2: Gráfico de eficiência.

4 Conclusão

Como pode ser visto tanto nos dados numéricos quanto nos gráficos, os resultados obtidos em *speedup* e eficiência não foram satisfatórios e portanto é importante analisar e tentar compreender o que levou a obtenção de tais resultados.

É sabido que implementações de programação paralela possuem *overhead*, isto é, um excesso de tempo em relação aos programas sequenciais devido aos custos computacionais realizados exclusivamente à paralelização, como por exemplo a comuniação entre os diferentes processadores e os cálculos para divisões de vetores.

A partir disso, é interessante analisar cada problema para avaliar o quão vantajoso a paralelização pode ser. Neste caso abordado, a paralelização foi prejudicial, não trazendo nenhum ganho de velocidade. O principal motivo para tal acontecimento é o tamanho do vetor utilizado ser relativamente pequeno, fazendo com que o ganho de tempo obtido nos cálculos paralizados seja perdido através dos tempos para comunicação entre processadores. Infelizmente não foi possível realizar testes com vetores em escalas muito maiores

pois a memória do computador pessoal é bastante limitada.

Portanto, pode-se concluir que a paralelização através da MPI pode trazer um ganho de tempo e *speedup* para o algoritmo utilizado, mas para isso é necessário que o tamanho dos vetores a ser somado seja muito grande e devido à capacidade limitada de memória no computador utilizado para os testes os vetores nos tamanhos necessários não puderam ser devidamente testados.