Relatorio: Algoritmo Ping Pong

Ovidio Jose da Silva Junior-GRR20182667

24 de dezembro de 2022

0.1. Resumo

O trabalho tem como objetivo a experimentação de latência entre envio de mensagens utilizando o MPI entre 2 processos, também houve a utilização da biblioteca Chronos, ambas bibliotecas foram utilizadas em um código na linguagem C, dada a sua compatibilidade.

0.2. Metodologia

Para testes foi utilizado uma maquina que possui 12 *threads*, para coleta dessa informação foi utilizado o comando *lscpu* descrito no *listing* 3, e com as seguintes características gerais da CPU obtidas com o comando *lstopo*:

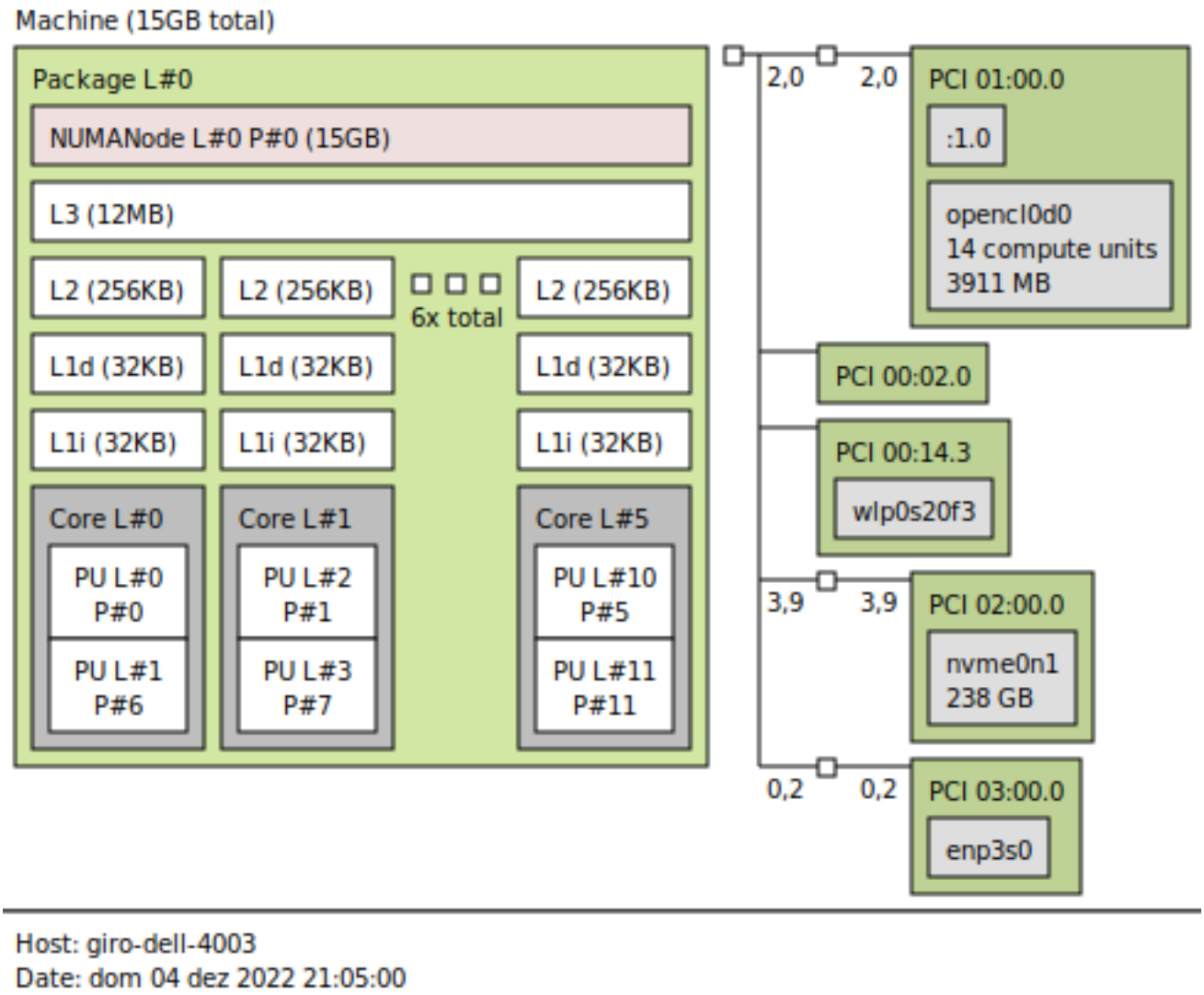

Figura 1. Topologia da CPU

Foi executado testes de envio de mensagens onde o tempo era medido entre essas trocas, afim de calcular a latência e a quantidade de MB/s transferidos por segundo.

0.3. Implementação

Para implementação foi utilizado 1 barreira e 2 processos, separando o algoritmo em 2 etapas, as etapas com uso de mensagens bloqueantes e seguido pelas não bloqueantes, além disso a execução do código paralelo funcionou atribuindo a cada processos 2 vetores nomeados consecutivamente de ping e pong. Cada processo possui uma copia desses vetores, onde o processo 0 preenchia o vetor ping e o enviava para o processo 1, e o processo 1 preenchia o vetor pong e o mandava para o processo 0. Com isso o intuito de no final do experimento ambos os vetores possuírem ambos os vetores completos da maneira correta.
Na linha de comando é passada o numero de processos além da quantidade de mensagens e seus tamanhos, após receber os parâmetros o programa calcula e gera as mensagens encima dos vetores ping e pong e envia de acordo com a modalidade

```
usage: %s <quantidade de mensagens> <tamanho> <Opicional: bloqueantes|nao
    bloqueantes>
```

Listing 1. exemplo de entrada

0.3.1. Bloqueantes

Para as mensagens Bloqueantes foi utilizado

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
    int tag, MPI_Comm comm)
```

Listing 2. envio de mensagens bloqueantes

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int
    tag,
             MPI_Comm comm, MPI_Status *status)
```

Listing 3. recebimento de mensagens bloqueantes

0.3.2. Não bloqueantes

Para as mensagens Bloqueantes foi utilizado

```
int MPI_Isend(const void *buf, int count, MPI_Datatype datatype, int dest
   , int tag,
              MPI_Comm comm, MPI_Request *request)
```

Listing 4. envio de mensagens não bloqueantes

```
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Request * request)
```

Listing 5. recebimento de mensagens não bloqueantes

0.4. Resultados e Conclusões

No método não bloqueante o envio a vazão é maior como pode ser visto nos gráficos abaixo, o desempenho de latência das mensagens entre os processo também é maior nas mesnagens nao bloqueantes, isso se deve ao fato do método bloqueante travar a execução do programa no envio e no recebimento de mensagens, tendo como consequência um desempenho um pouco pior.

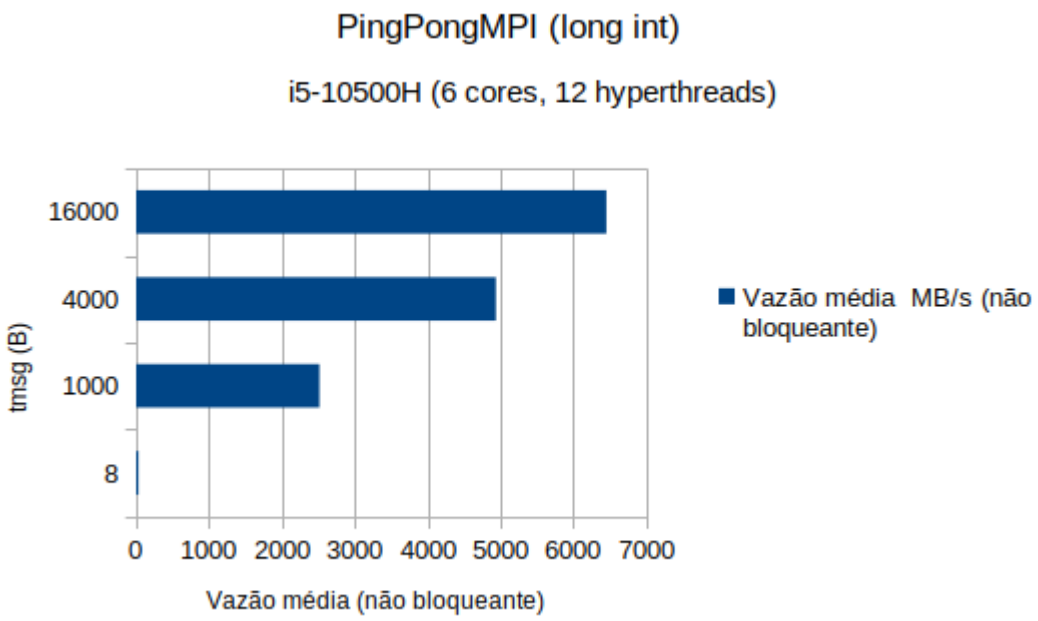

Figura 2. Grafico do experimento mensagens não bloqueantes

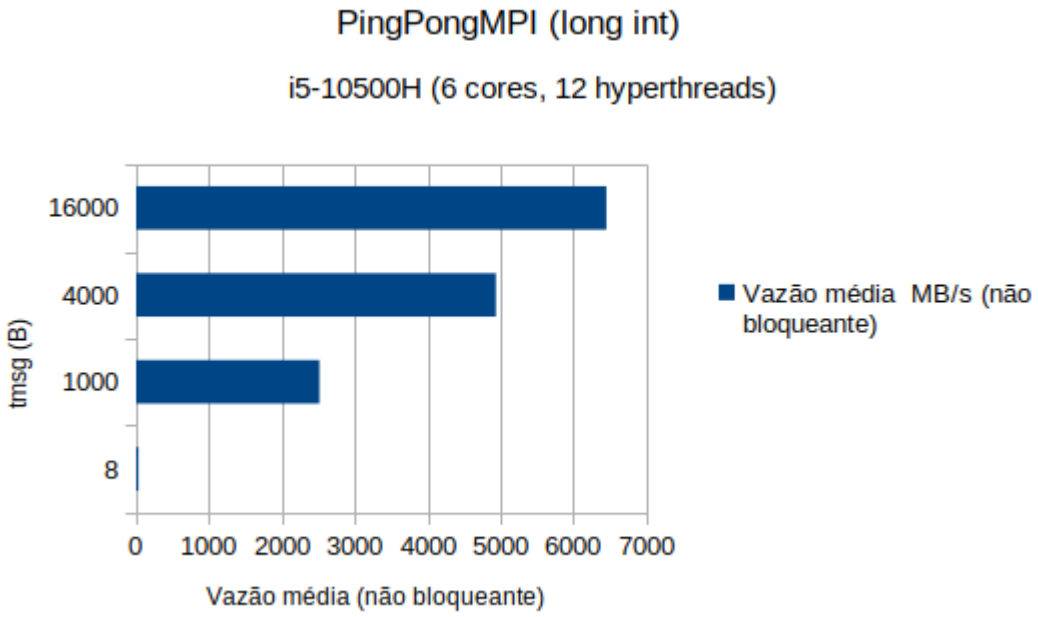

Figura 3. Grafico do experimento mensagens bloqueantes

```
Arquitetura:                         x86_64
Modo(s) operacional da CPU:          32-bit, 64-bit
Ordem dos bytes:                     Little Endian
Address sizes:                       39 bits physical, 48 bits virtual
CPU(s):                              12
Lista de CPU(s) on-line:             0-11
Thread(s) per n cleo:                 2
```

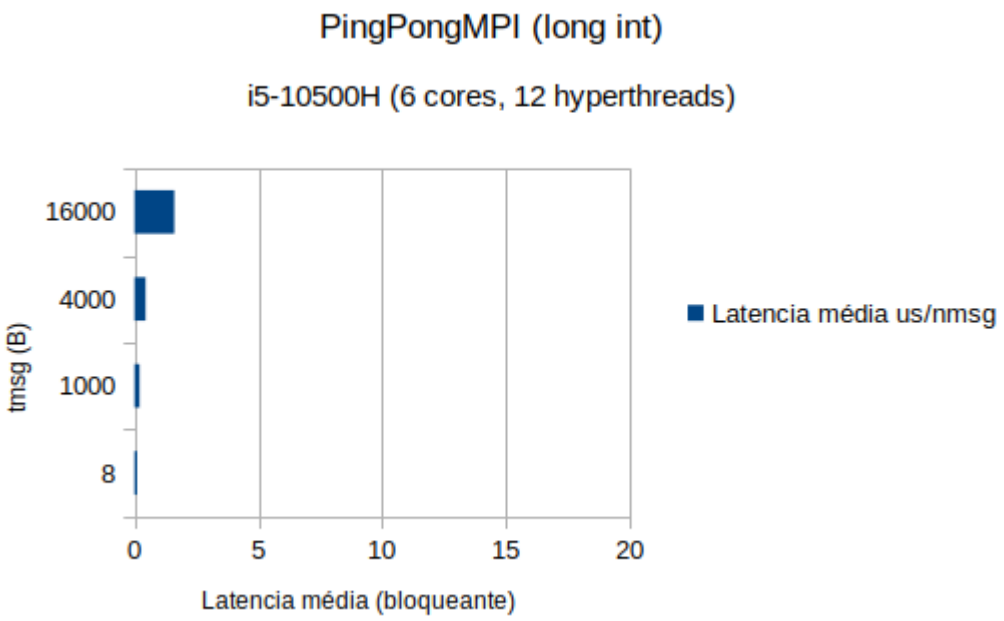

Figura 4. Grafico do experimento mensagens não bloqueantes

```
N cleo(s) por soquete:           6
Soquete(s):                      1
N  (s) de NUMA:                   1
ID de fornecedor:                GenuineIntel
Fam lia da CPU:                   6
Modelo:                          165
Nome do modelo:                  Intel(R) Core(TM) i5-10500H CPU @ 2.50
   GHz
Step:                            2
CPU MHz:                         1136.906
CPU MHz m x.:                     4500,0000
CPU MHz m n.:                     800,0000
BogoMIPS:                        4999.90
Virtualiza o:                     VT-x
cache de L1d:                    192 KiB
cache de L1i:                    192 KiB
cache de L2:                     1,5 MiB
cache de L3:                     12 MiB
CPU(s) de n 0 NUMA:               0-11
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Mitigation; Clear CPU buffers; SMT
   vulnerable
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass
   disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and
    __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB
   conditional, RSB filling
```

```
Vulnerability Srbds:             Mitigation; Microcode
Vulnerability Tsx async abort:   Not affected
Op  es:                            fpu vme de pse tsc msr pae mce cx8
   apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
   sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art
   arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid
   aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg
   fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
   tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch
    cpuid_fault epb invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced
   tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1
   avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt
   xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify
   hwp_act_window hwp_epp pku ospke md_clear flush_l1d arch_capabilities
```

Listing 6. saida lscpu