Relatório
Trabalho 3 de Programação Paralela

GRR20190427 - João Lucas Cordeiro

1 Introdução

Este é um relatório do terceiro trabalho de Programação Paralela, onde implementamos uma troca de mensagens por meio do MPI, de forma, tanto bloqueante, quanto não-bloqueante. O algoritmo desenvolvido será detalhado, assim como o resultado dos experimentos usando o código.

2 O Algoritmo

O algoritmo começa tratando a entrada dos argumentos que o usuário colocou no terminal. Caso os argumentos não estejam de acordo com as regras exigidas pro funcionamento, o programa fecha com um erro. Caso esteja tudo correto, os argumentos são guardados em variáveis.

Apóis isso, iniciamos o MPI e guardamos o número de processos MPI em *nProc* e o rank do processo atual em *rankProc*. Depois, guardamos em *ni* quantos números terão em cada mensagem e o *rank* do outro processo em *otherProc*.

Então, criamos a mensagem de acordo com o *rank* do processo na função *CriaMsg* e a guardamos em *buffMsgSend*. Também allocamos espaço para as mensagens que serão recebidas em *buffMsgRecv* e criamos uma variável do tipo **MPI_STATUS** para usar na função de recebimento de mensagens.

Aqui, os vetores usados de *buffer* são verificados pela primeira vez. Caso tenha sido chamado com a opção *não-bloqueante*, uma barreira é usada aqui para sincronizar os processos. Então, caso seja o processo 0, o cronômetro é ligado.

No caso da chamada para mensagens *bloqueantes*, usamos a função **MPI_Ssend** para enviar mensagens com sincronia e depois a função **MPI_Recv** para receber uma mensagem caso sejamos o processo com *rank* igual à 0. Caso o *rank* seja 1, fazemos a mesma coisa mas na ordem invertida.

Para a chamada para mensagens *não-bloqueantes*, usamos a função **MPI_Bsend** para mandarmos mensagens sem sincronia. Usamos a mesma função para receber e a ordem é igual, tanto pro $rank = 0$, quanto para o $rank = 1$.

Caso tenha sido chamado com a opção *não-bloqueante*, usamos novamente uma barreira para sincronizar os processos. Caso seja o processo 0, paramos o cronômetro, guardamos o tempo total na variável *tempo* em segundos e a imprimimos na tela.

Então, os vetores de *buffer* são verificados novamente. Por fim, o **MPI** é finalizado.

3 Descrição do Processador

Os resultados citados neste relatório foram obtidos na máquina orval do DINF. Usando o comando *lscpu* temos estas informações do processador:

Arquitetura: x86_64
Modo(s) operacional da CPU: 32-bit, 64-bit
Ordem dos bytes: Little Endian
Tamanhos de endereço: 40 bits physical, 48 bits virtual
CPU(s): 16
Lista de CPU(s) on-line: 0-15
Thread(s) per núcleo: 2
Núcleo(s) por soquete: 4
Soquete(s): 2
Nó(s) de NUMA: 2
ID de fornecedor: GenuineIntel
Família da CPU: 6
Modelo: 44
Nome do modelo: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
Step: 2
Aumento de frequência: habilitado
CPU MHz: 1600.000
CPU MHz máx.: 2401,0000
CPU MHz mín.: 1600,0000
BogoMIPS: 4799.75
Virtualização: VT-x
cache de L1d: 256 KiB
cache de L1i: 256 KiB
cache de L2: 2 MiB
cache de L3: 24 MiB
CPU(s) de nó0 NUMA: 0-3,8-11
CPU(s) de nó1 NUMA: 4-7,12-15
Vulnerability Itlb multihit: KVM: Mitigation: VMX disabled
Vulnerability L1tf: Mitigation; PTE Inversion; VMX conditional cache flushes, SMT vulnerable
Vulnerability Mds: Vulnerable: Clear CPU buffers attempted, no microcode; SMT vulnerable
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Mmio stale data: Unknown: No mitigations
Vulnerability Retbleed: Not affected
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Retpolines, STIBP disabled, RSB filling, PBRSB-eIBRS Not affected
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Not affected
Opções: 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 pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 popcnt lahf_lm epb pti tpr_shadow vnmi flexpriority ept vpid dtherm ida arat

4 Resultados

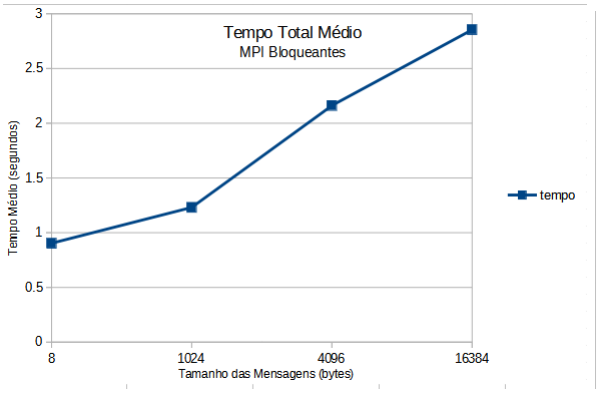
Os experimentos são rodados usando os scripts que o professor disponibilizou para a realização do trabalho. Aqui, mostramos os resultados de uma execução do experimento:

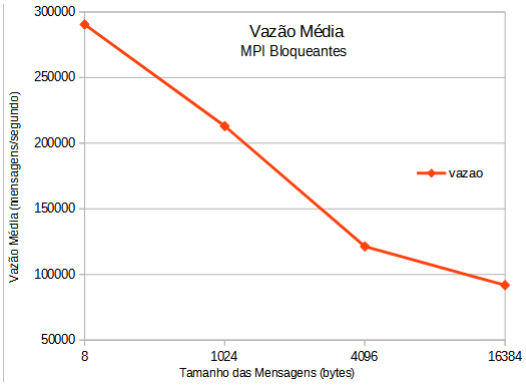
Tamanho das Mensagens	Tempo Med. Bloq.	Vazão Med. Bloq.	Vazão Med. Não Bloq.
8	0.9024532	290479.32	280834.36
1024	1.2304195	213052.54	204148.14
4096	2.1612161	121294.67	121734.96
16384	2.854452	91836.89	94462.35

Sobre a versão bloqueante, seu tempo total aumenta pois o tamanho das mensagens também aumentam, o que acontece de forma inversamente proporcional com a vazão também. Já na versão não-bloqueante a tendência da vazão não muda muito também, diminuindo com o aumento do tamanho das mensagens.

E aqui, tamos os gráficos gerados:

Gráficos para a versão bloqueante:





Gráficos para a versão não-bloqueante: