Relatório
Trabalho 1 de Programação Paralela

GRR20190427 - João Lucas Cordeiro

1 Introdução

Este é um relatório do primeiro trabalho de Programação Paralela, onde implementamos uma versão da soma de prefixos que roda com várias threads. O algoritmo desenvolvido será detalhado, assim como o resultado dos experimentos usando o código.

2 O Algoritmo

O algoritmo consiste inicialmente em fazer a soma de prefixos parcialmente em partes do vetor de entrada. O tamanho das partes é o numero total de elementos no vetor de entrada divido pelo número de threads requeridas.

Cada parte é feita por uma thread, que é criada no começo do *PtPrefixSum*, usando a função *PrefixSumPart*. Usamos um vetor de argumentos para passar: o intervalo do vetor de entrada, o vetor de entrada, o de saída e uma posição do vetor de máximos. Aqui, temos uma barreira e o algoritmo só continua depois que todas as threads fizeram sua parte.

Nisso, separamos o último termo de cada parte e guardamos em um vetor, o *vMaximos*. Depois, fazemos uma soma de prefixos nesse vetor. Aqui não fazemos de forma paralela, já que o vetor é pequeno, com o tamanho = número de threads.

Depois, fazemos threads para somar os termos do *vMaximos* nos mesmos intervalos usados anteriormente, usando a função *sumMaxPrefix*. Aqui, passamos os argumentos: intervalo do vetor de saída, número da thread, vetor de saída e de máximos. Também temos uma barreira, fazendo o algoritmo continuar apenas quando todas as threads terminarem.

Com isso, temos o vetor resultante com a saída desejada. Como todas as operações foram feitas na função *PtPrefixSum*, calculamos o tempo baseado no tempo que essa função rodou. Então, imprimimos o tempo total e as operações por segundo. Assim, o algoritmo termina.

3 Script

O script utilizado roda o programa para 1073741824 (2^{30}) elementos, quantia que pode ser alterada no valor **ARGELEM**. Então entramos em um *for* que roda o programa de 1 a 8 threads. Dentro deste for, rodamos o programa 10 vezes, pegamos seus resultados e geramos os dados desejados: tempo médio e mínimo, OP/s médio e máximo, e aceleração.

Com isso, geramos um csv com os dados desejados. Usando este documento, criamos um gráfico para ilustrar os resultados do experimento.

4 Descrição do Processador

Os resultados citados neste relatório foram obtidos na máquina cpu1 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: 48 bits physical, 48 bits virtual
CPU(s): 32
Lista de CPU(s) on-line: 0-31
Thread(s) per núcleo: 1
Núcleo(s) por soquete: 8
Soquete(s): 4
Nó(s) de NUMA: 1
ID de fornecedor: AuthenticAMD
Família da CPU: 23
Modelo: 1
Nome do modelo: AMD EPYC 7401 24-Core Processor
Step: 2
CPU MHz: 1999.999
BogoMIPS: 3999.99
Virtualização: AMD-V
Fabricante do hipervisor: KVM
Tipo de virtualização: completo
cache de L1d: 2 MiB
cache de L1i: 2 MiB
cache de L2: 16 MiB
cache de L3: 64 MiB
CPU(s) de nó0 NUMA: 0-31
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf: Not affected
Vulnerability Mds: Not affected
Vulnerability Meltdown: Not affected
Vulnerability Mmio stale data: Not affected
Vulnerability Retbleed: Mitigation; untrained return thunk; SMT disabled
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Retpolines, IBPB conditional, 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 mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm rep_good no pl cpuid extd_apicid tsc_known_freq pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor la hf_lm svm cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw perfctr_core ssbd ibpb vmmcall fsgsbase tsc_adjust bmi1 avx2 smep bmi2 rdseed adx sm ap clflushopt sha_ni xsaveopt xsavec xgetbv1 xsaves clzero xsaveerptr virt_ssbd arat npt nrip_save arch_capabilities

5 Resultados

Aqui, mostramos os resultados de uma execução do experimento:

Threads	Tempo Min.	OP/s Max.	Tempo Med.	OP/s Med.	Aceleracao
1	6.328968	169655112.335144	6.701284	160503466.555295	1.000000
2	6.315697	170011623.096399	6.670299	161149355.582668	1.004024
3	4.217674	254581491.044643	4.565511	235853884.744733	1.463573
4	3.367576	318847124.400207	3.517798	305391637.303104	1.294834
5	2.797691	383795665.535985	2.957082	363502706.001406	1.190283
6	2.319597	462900093.023430	2.414418	444994885.965153	1.224185
7	1.980507	542155012.273246	2.137764	502978258.589937	1.130301
8	1.785363	601413787.130838	1.884859	570345091.883962	1.133935

E aqui, temos o gráfico ilustrando as OP/s e a aceleração dependendo do número de threads: