Programação Paralela - Trabalho 2

Fabiano A. de Sá Filho
Departamento de Informática
Universidade Federal do Paraná – UFPR
GRR20223831
fabiano.filho@ufpr.br

***Resumo*—Este trabalho teve como objetivo a paralelização de um programa de particionamento de vetores em C. Para tal foi utilizado um pool de threads com a biblioteca pthreads. Foram rodados dois experimentos, o primeiro com mil elementos no vetor de posições, e o segundo com 100 mil. Conforme planilhas em anexo, a paralelização foi um sucesso, de forma que ao aumentar o número de threads, diminui-se o tempo de execução e aumenta-se o throughput.**

I. Observações sobre o trabalho

O arquivo enviado contém os códigos fonte e makefile. O README contém as instruções para compilar e rodar o programa. Também tem a saída que foi rodada no cluster w00 e que foi utilizada na planilha para os dois experimentos, que também é inclusa. Vale pontuar algumas ressalvas sobre o desenvolvimento do trabalho:

- Impactos da Cache: Modifique o script slurm fornecido e fiz meu proprio script run.sh, que produzem os resultados utilizados pela planilha. Dessa forma, o script slurm chama o executável novamente cada uma das NTIMES vezes, o que significa que a cada chamada do executavel os vetores são gerados novamente, não havendo portanto impacto da Cache nas medições de tempo.
- Apesar do meu algoritmo escalar bem com paralelismo, eu acredito que minha solução tem complexidade alta, talvez $O(n * np)$. Não pude rodar os experimentos com 100 mil elementos no cluster w00, em todas as tentativas, ultrapassei o limite de tempo.

II. Discussão de Resultados

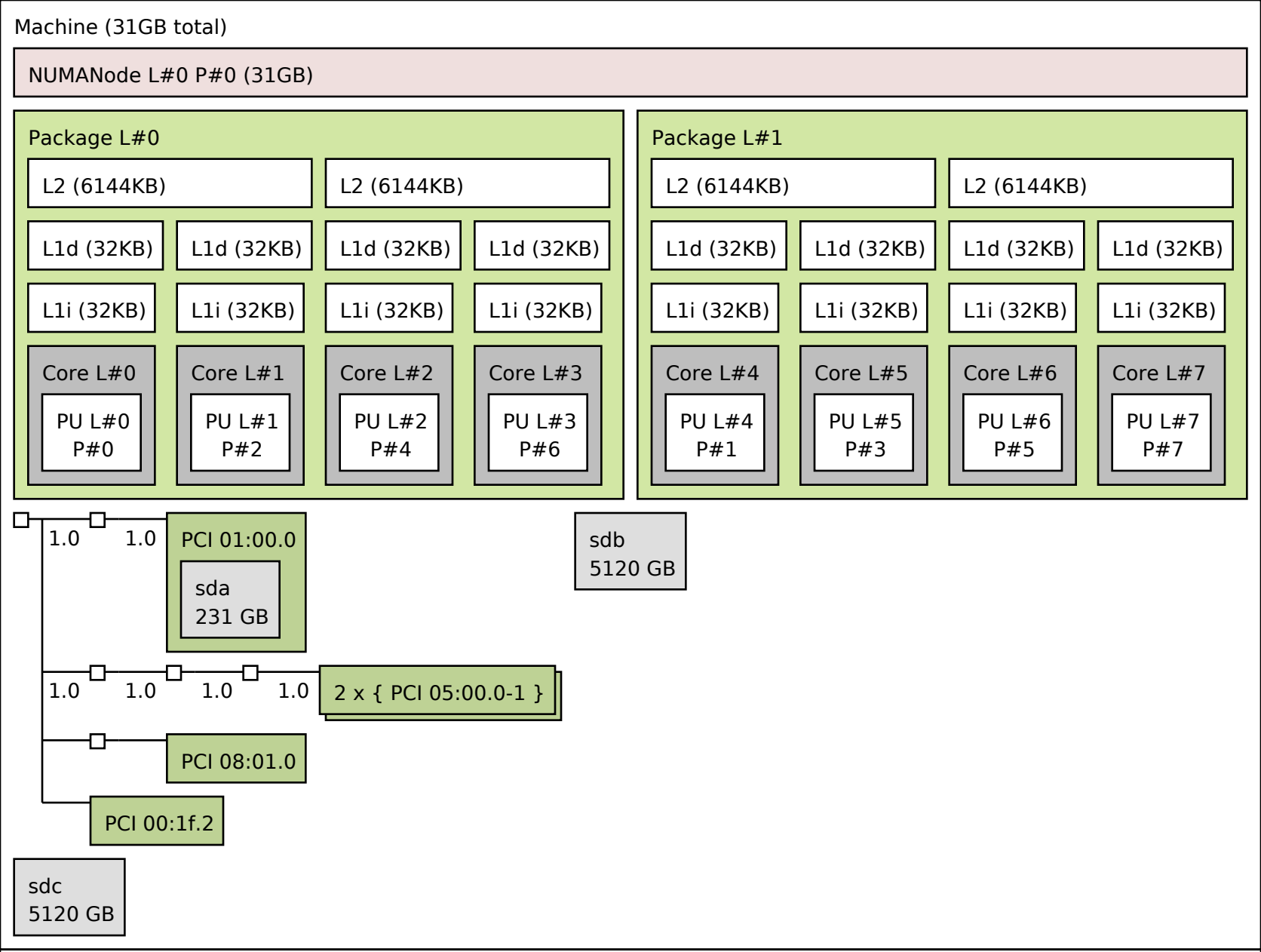
Como pode ser observado pela planilha, o algoritmo de partição proposto obteve sucesso ao ser paralelizado. Por exemplo, para 1000 posições (parte A), obtivemos um tempo médio de 11,1 segundos sem threads, mas com 8 threads, o tempo cai para 1,6 segundos, representando uma aceleração de 7,1x. O throughput obtido com 8 threads foi de 6,41 MEPS, contra somente 0,91 MEPS com apenas uma thread.

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  @ 2.80
   GHz
Stepping:                        6
CPU MHz:
   2792.839
BogoMIPS:                        5585.67
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: VMX disabled
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
```

III. SAÍDA DO PROGRAMA TOPOLOGY



Host: wz00
Date: Mon Nov 25 21:21:15 2024