"Nothing in life is to be feared, it is only to be understood" (Marie Curie).

Prática - Problemas de Cache

Paulo Lisboa de Almeida

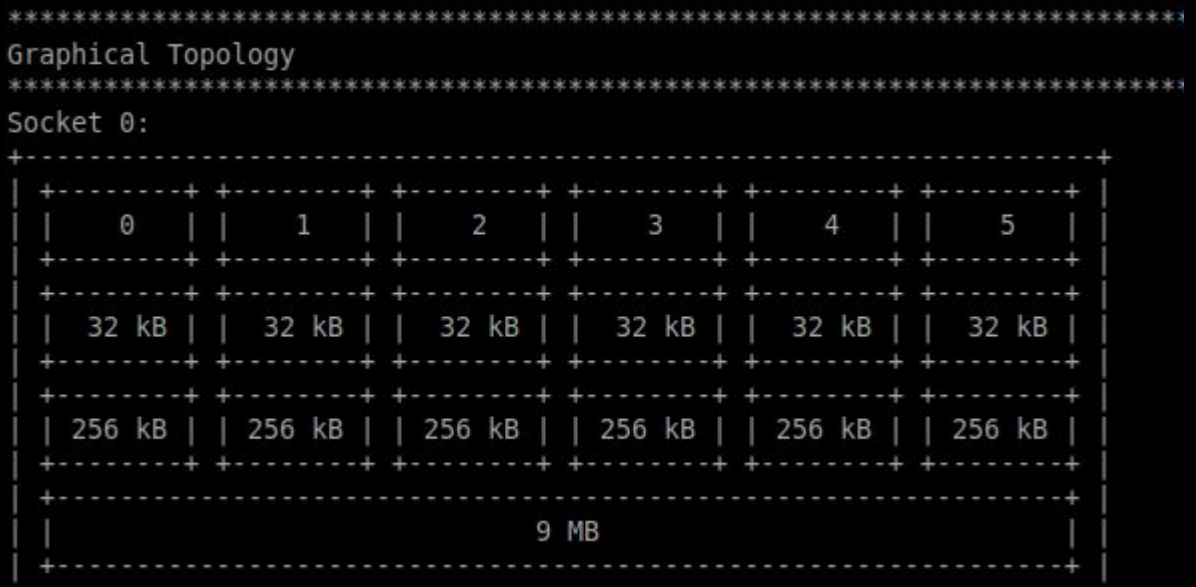
DSBD

UFPR

Topologia da cache

Verifique a topologia das caches de sua máquina.

likwid-topology -c -g



Contadores

Para verificar os grupos dos contadores de performance em sua CPU, execute o comando.

```
likwid-perfctr -a
```

Vamos assumir que os grupos para medição de miss da caches L2 e L3, Barramento de Memória, e compartilhamento falso estão disponíveis.

```
FALSE_SHARE, L2CACHE, L3CACHE, MEM
```

Caso não existam esses grupos, tente encontrar outros grupos similares.

Compilando o programa Matriz

Baixe o programa Matriz disponibilizado.

Entenda o que esse programa faz, e como.

Compile o programa.

```
gcc matriz.c -o matriz
```

Executando

Para executar, utilize o seguinte comando:

```
likwid-perfctr -C 0 -g L2CACHE ./matriz
```

Exercício

Feche todos os demais programas.

- Durante o teste não deixe sua máquina gastar tempo com outras coisas.

Abra dois terminais.

- Em um, execute o programa `htop` para verificar a carga nas CPUs.
- No outro terminal execute o programa `matriz` com a linha de comando do slide anterior.

Faça isso 3x.

Anote para cada rodada:

- O tempo real.
- Os misses na cache para cada rodada nas CPUs envolvidas.
- Nesse teste, somente o Core 0 está envolvido.

Exercício

Modifique o Loop, de i,j para j,i.

O programa ainda faz a mesma coisa, mas agora ele salta de linha em linha no loop mais interno, ao invés de fazer coluna a coluna.

Repita os testes.

Qual é mais rápido, e o que o programa mais lento está fazendo de errado na cache para gerar mais misses?

Programa Loop

Abra o programa `Loop` em um editor de texto.

Note as duas versões comentadas, uma que executa com uma única thread, e uma que executa em duas threads.

Programa Loop

Remova o comentário do programa para executar o loop em **uma única thread**.

Para compilar o programa em um computador com OpenMP instalado.

```
gcc loop.c -o loop -fopenmp
```

Execute

Para executar, utilize o seguinte comando:

```
likwid-perfctr -C 0 -g FALSE_SHARE ./loop
```

Exercício

Feche todos os demais programas.

Abra dois terminais.

- Em um, execute o programa `htop` para verificar a carga nas CPUs.
- No outro terminal rode o programa `loop` com a linha de comando do slide anterior.

Faça isso 3x.

Anote para cada rodada.

- O tempo real.
- A quantidade de compartilhamento falso para cada rodada em MB nas CPUs envolvidas.
- Nesse teste, somente o Core 0 está envolvido.

Exercício

Comente o trecho do programa que executa em uma única thread.

Remova o comentário do programa que executa em duas threads.

Compile novamente.

Execute os testes novamente, tomando cuidado para executar em duas CPUS.

Exemplo de comando:

```
time likwid-perfctr -C 0,2 -g FALSE_SHARE ./loop
```

Conclusões

O que aconteceu com as medidas, quando comparadas com a versão de uma única thread?

Você esperava que seu programa executasse duas vezes mais rápido? Isso realmente aconteceu?

Quais problemas estamos causando na cache? Por quê?

Exercício

Baseado na discussão do slide anterior, crie uma nova versão do programa que executa em duas threads.

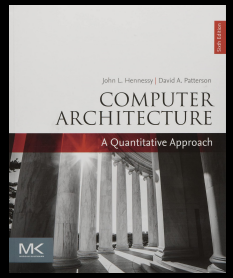
Elimine os problemas sendo causados na cache que você detectou.

Execute novamente as medições e compare.

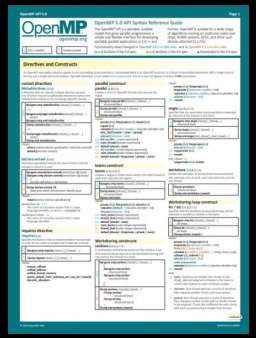
Referências

Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.

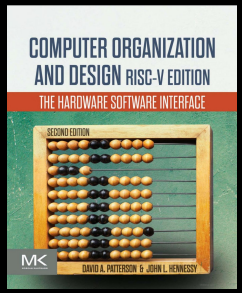
Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Manual OpenMP 5.2. 2022. www.openmp.org/resources/refguides



Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Licença