Análise de código e eficiência do método do Gradiente

Aryane Ast dos Santos
Kevin Katzer

23 de novembro de 2014

Sumário

1 Introdução **1**

2 Verificação de uso de memória com Valgrind **2**

3 Arquitetura do computador **2**

4 Análise do cálculo do fator lambda **3**

4.1 flop operations 3

4.2 mem utilization 3

4.3 explicar graficos flops_dp, cache, mem 3

4.4 melhoria 3

5 Análise do cálculo do resíduo **3**

5.1 flop operations 3

5.2 mem utilization 3

5.3 explicar graficos flops_dp, cache, mem 3

5.4 melhoria 3

1 Introdução

Motivação...

2 Verificação de uso de memória com Valgrind

Ao executar a ferramenta Valgrind para se obter informações sobre vazamento de memória no programa gradSolver, foi possível observar 5 erros, todos em contextos diferentes, além de 16 allocações e apenas 2 liberações de memória.

Os resultados da execução do programa são parcialmente apresentados na figura 1.

```
==29599== Command:  ./gradSolver -r 5
==28949== HEAP SUMMARY:
==28949== in use at exit:  560 bytes in 14 blocks
==28949== total heap usage:  16 allocs, 2 frees, 800 bytes allocated
==28949== LEAK SUMMARY:
==28949== definitely lost:  560 bytes in 14 blocks
==28949== ERROR SUMMARY: 5 errors from 5 contexts (suppressed:  0 from 0)
```

Figura 1: Saída do Valgrind

Aqui o gradSolver foi executado com uma matriz quadrada de ordem 5, porém os mesmos problemas listados na figura 1 são encontrados em execuções de matrizes de qualquer dimensão. E de maneira análoga, ao resolver os problemas apresentados, numa execução com matriz maior, eles ficam também automaticamente resolvidos.

Para contornar os vazamentos de memória encontrados, foi necessário liberar a memória dos vetores alocados explicitamente como o vetor x na função main, o vetor aux em calcGrad e o vetor r de resíduo na função gradSolver. Além disso, no main, foram adicionados frees para os ponteiros para char das flags do getopt.

3 Arquitetura do computador

Utilizando a ferramenta likwid-topology, é possível obter as seguintes informações sobre a arquitetura do computador utilizado para os testes de performance.

Como pode-se notar na figura 2, nas servidoras do DInf, há uma CPU Interlagos, fabricada pela AMD, com 1 socket e 6 cores.

Existem 3 níveis de cache, sendo o primeiro (L1) com 16kB de memória, o segundo (L2) com 2MB e o terceiro (L3) com 8MB. A cache L1 é separada em 6 grupos, sendo que cada grupo é destinado a um core diferente, a cache L2 é separada em 3, tendo um par de cores ligados a cada grupo, e o último nível de cache, L3, não é separado, sendo toda a sua memória disponível a todos os cores da CPU.

Por fim, a memória RAM totaliza 3678.97MB, sendo que desse tanto, apenas 152.301MB estavam livres no momento de execução do likwid-topology.

4 Análise do cálculo do fator lambda

4.1 flop operations

4.2 mem utilization

4.3 explicar graficos flops_dp, cache, mem

4.4 melhoria

5 Análise do cálculo do resíduo

5.1 flop operations

5.2 mem utilization

5.3 explicar graficos flops_dp, cache, mem

5.4 melhoria

```
CPU type:  AMD Interlagos processor

Hardware Thread Topology
Sockets:  1
Cores per socket:  6
Threads per core:  1
HWThread Thread Core Socket
0 0 0 0
1 0 1 0
2 0 2 0
3 0 3 0
4 0 4 0
5 0 5 0
Socket 0:  ( 0 1 2 3 4 5 )

Cache Topology
Level:  1
Size:  16 kB
Cache groups:  ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 )
Level:  2
Size:  2 MB
Cache groups:  ( 0 1 ) ( 2 3 ) ( 4 5 )
Level:  3
Size:  8 MB
Cache groups:  ( 0 1 2 3 4 5 )

NUMA Topology
NUMA domains:  1
Domain 0:
Processors:  0 1 2 3 4 5
Relative distance to nodes:  10
Memory:  152.301 MB free of total 3678.97 MB
```

Figura 2: Saída do likwid-topology