FSO - Trabalho 1, Questão 1

> Alunos: Dylan Guedes e Artur Bersan
>
> Matrículas: 12/0115727 e 14/0016813

Sistema Operacional Utilizado

> Ubuntu Linux - x86_64 Kernel 3.13.0-32-generic
>
> Linux Fedora 24 - x86_64 Kernel 4.6.3-300

Ambiente de Desenvolvimento

> Computadores pessoais. Utilizamos o Linux, Makefile, git, valgrind e vim.

Instruções

Compilando

Cada item tem sua instrução no makefile. Para compilar o item 3 da questão 3 (parte C da questão), escreva:

```
$ make c
```

Utilização

Um arquivo de input pronto está localizado na mesma pasta do Makefile,

e tem o nome de "in". Para utilizar, passe o conteúdo do arquivo "in" para o binário gerado:

```
$ cat in | ./a.out
```

O binário será então utilizado com o conteúdo do arquivo "in". Também é possível utilizar o arquivo "large_input", que apresenta uma matriz maior que a presente no arquivo "in".

Caso seja necessário a utilização de input via IO, os inputs esperados são os seguintes: > Número de linhas da matriz A Número de colunas da matriz A Número de linhas da matriz B Número de colunas da matriz B Itens da matriz A Itens da matriz B

O programa responderá então com as matrizes entradas (A e B) e o resultado de AxB.

Output - valgrind (usando os inputs presentes em "in")

Utilização

1. Compile a questão:

   ```
   $ make a
   ```

2. Utilize o binário no valgrind, utilizando o arquivo "in" de input:

   ```
   $ valgrind ./a.out < in
   ```

   Também é possível utilizar o arquivo "large_input", que apresenta

uma matriz maior que a presente no arquivo "in".

Item A - Resultado

```
==14122== HEAP SUMMARY:
==14122==     in use at exit: 0 bytes in 0 blocks
==14122==   total heap usage: 16 allocs, 16 frees, 5,392 bytes a
==14122==
==14122== All heap blocks were freed -- no leaks are possible
```

Item B - Resultado

```
==10623== LEAK SUMMARY:
==10623==    definitely lost: 0 bytes in 0 blocks
==10623==    indirectly lost: 0 bytes in 0 blocks
==10623==      possibly lost: 0 bytes in 0 blocks
==10623==    still reachable: 1,584 bytes in 4 blocks
==10623==         suppressed: 0 bytes in 0 blocks
```

Item C - Resultado

```
==10644== LEAK SUMMARY:
==10644==    definitely lost: 0 bytes in 0 blocks
==10644==    indirectly lost: 0 bytes in 0 blocks
==10644==      possibly lost: 1,632 bytes in 6 blocks
==10644==    still reachable: 0 bytes in 0 blocks
==10644==         suppressed: 0 bytes in 0 blocks
```

Limitações conhecidas

- O valgrind alerta que talvez alguns blocos de memória tenham sido

perdidos na solução C
- O input não é grande o suficiente para fazer com que as soluções que usem thread tenham chance de ter melhor desempenho que a solução que não use.
- Na solução C, as threads utilizadas sempre vão "em batch". Exemplo: no caso de 4 threads, são lançadas 4 threads. A quinta thread não é liberada até que as 4 sejam finalizadas.

Análise

Resultados obtidos (utilizando "time", e o arquivo "in" como input)

Item A

```
real    0m0.003s
user    0m0.002s
sys     0m0.003s
```

Item B

```
real    0m0.005s
user    0m0.002s
sys     0m0.006s
```

Item C

```
real    0m0.009s
user    0m0.001s
sys     0m0.010s
```

Resultados obtidos (utilizando "time", e o arquivo "large_input" como input)

Item A

```
real    0m0.006s
user    0m0.004s
sys     0m0.005s
```

Item B

```
real    0m0.025s
user    0m0.007s
sys     0m0.026s
```

Item C

```
real    0m0.029s
user    0m0.011s
sys     0m0.027s
```

Análise

A não utilização de threads teve um desempenho melhor tanto com pequeno input (3x2 e 2x3) quanto com um input um pouco maior (20x20 e 20x20). Isto se deve ao fato de que as operações realizadas no problema são relativamente simples e que o input ainda não era grande o suficiente para fazer com que a utilização de threads valesse a pena. O overhead de criar threads, esperar, dar join em cada uma, dentre outras coisas, faz com que um tempo relativamente grande seja gasto gerenciando esse recurso.

A respeito da utilização de diversas threads X a utilização de threads num limite dado pelo /proc/cpuinfo, o desempenho da questão B ficou inferior para o arquivo "in" no que tange o tempo em que o binário estava executando (0.002s vs 0.001s). Isto se deve ao fato de uma menor alocação de threads. Por outro lado, o tempo em que o SO estava gerenciando as threads (sys) foi bem maior (0.006s vs 0.010s). Descobrir o número de núcleos do processador em tempo de execução pode ter tido um grande peso nesse resultado.

Ainda, para um maior input, a solução B teve desempenho quase igual em relação ao item sys (ou seja, a diferença diminuiu, provavelmente pela solução C usar menos threads), mas em relação ao user a solução B teve desempenho superior (0.007s vs 0.011s).

Logo, conclui-se que para o problema dado e para o conjunto de inputs utilizados a solução A embora seja a mais simples foi a solução mais rápida, a solução B a segunda mais rápida, e a solução C a de pior desempenho.