Mini EP3

Analisando o Desempenho de um Programa com *Profiling*

Motivação

Em computação paralela e distribuída, otimizar o código é crucial para maximizar o desempenho de aplicações complexas. Antes de paralelizar, é essencial garantir que o código serial esteja bem otimizado. Nos Mini EPs anteriores, usamos o tempo de execução para avaliar o desempenho e otimizamos o código sem ao menos pensar em paralelizá-lo. Uma abordagem mais precisa é o ***profiling***, que coleta informações detalhadas sobre o comportamento do programa, como uso de recursos e tempo de execução das rotinas.

O *profiling* permite identificar os *hotspots*, as partes do código que mais demandam tempo de execução. Ao focar nesses pontos críticos, otimizações específicas podem resultar em melhorias significativas no desempenho geral. No entanto, nem sempre é possível ou necessário otimizar todas as partes do código; alguns gargalos de desempenho são inerentes à natureza da tarefa.

Duas técnicas principais de *profiling* são usadas: **instrumentação** e **amostragem**. A instrumentação modifica o código-fonte para incluir logs detalhados em cada chamada de função, o que pode gerar sobrecarga de processamento. A amostragem é menos invasiva, interrompendo o programa periodicamente para registrar o estado de execução, fornecendo uma visão estatística dos pontos críticos.

Com ferramentas como o `gprof` do Linux, é possível analisar o arquivo gerado pelo *profiling* para identificar quais partes do código requerem otimização ou se a paralelização seria eficaz. Assim, o *profiling* se torna um passo fundamental na preparação de um código para execução em ambientes paralelos e distribuídos.

Além do `gprof`, outra ferramenta poderosa para *profiling* em sistemas Linux é o `perf`. O `perf` oferece recursos avançados de análise de desempenho, permitindo coletar dados sobre eventos de hardware, como ciclos de CPU, *cache misses* e *branch mispredictions*, além de informações sobre o *software*, como chamadas de função e tempo de execução.

Enunciado

Neste terceiro Mini EP, você deverá usar a técnica de *profiling* para melhorar o código fornecido. Você estará recebendo o código a ser otimizado `mini_ep3.c` e um `Makefile` com instruções de como compilar o código. Utilize ferramentas como `perf` ou `gprof` para avaliar as regiões mais demoradas do código e identificar os gargalos de desempenho. Analise os resultados do *profiling* e proponha otimizações

para o código, justificando suas escolhas. Implemente as otimizações e compare o desempenho antes e depois das mudanças, utilizando novamente as ferramentas de *profiling* para medir o impacto das otimizações.

Você deverá entregar:

- Um arquivo compactado contendo o código-fonte otimizado e o `Makefile`.
- Um pequeno relatório em PDF descrevendo os resultados do *profiling* antes e depois das otimizações, as otimizações implementadas e sua justificativa.

Algumas Dicas

Para realizar o *profiling* com `gprof` é necessário fornecer algumas flags de compilação. Em compiladores como o GCC, isso pode ser feito com o comando `gcc -pg -g <program.c> -o <exec>` para códigos em C. Esse comando instrui o compilador a gerar informações de *profiling* durante a execução do programa. Após a execução, um arquivo chamado `gmon.out` será gerado, contendo os dados coletados. Para analisar esses resultados, é necessário utilizar o programa `gprof`, invocando-o com o comando `gprof <exec>`. Essa análise permite identificar as rotinas que mais consomem tempo e os potenciais pontos de otimização do código.

Se estiver usando `gprof`, analise as relações entre as funções (quem chama quem) para entender o fluxo de execução e identificar chamadas recursivas ou cadeias de chamadas longas que podem ser otimizadas. O campo `% time` indica a porcentagem do tempo total de execução gasto em cada função. Concentre-se nas funções com maior porcentagem, pois são potenciais *hotspots*. O campo `self seconds` indica o tempo gasto dentro da própria função, sem incluir chamadas a outras funções. `calls` número de vezes que a função foi chamada.

Se estiver usando `perf`, utilize os comandos `perf record -g <exec>` e `perf report` para realizar o `profile` e avaliar os resultados, respectivamente. O campo `Overhead` indica a porcentagem de tempo gasto em cada função ou linha de código. Já o campo `Samples` indica o número de vezes que o contador de desempenho foi interrompido em uma determinada função ou linha de código.

Links relevantes:

- https://realpython.com/python-profiling/#perf-count-hardware-and-system-events-on-linux
- https://perf.wiki.kernel.org/index.php/Tutorial
- https://web.eecs.umich.edu/~sugih/pointers/gprof_quick.html
- https://www.math.utah.edu/docs/info/gprof_6.html