Introdução à Programação Multithread com PThreads

Francisco José da Silva e Silva

Universidade Federal do Maranhão

Objetivos, Datas e Observações

- Introduzir o conceito de programação concorrente com múltiplas threads e seções críticas a serem tratadas.
- Analisar comparativamente o desempenho do algoritmo com diferentes números de threads.
- Data limite para envio: **4 de Junho de 2019**
- **2 pontos em cima da segunda nota (8 pontos prova escrita + 2 pontos trabalhos de implementação)**
- Envio feito SIGAA (códigos, como executar seu trabalho e um pequeno relatório)
- Dúvidas podem e devem ser retiradas com o monitor: **Rodolfo Sobreira Alves / rodolfosalves@lsdi.ufma.br / 981578730 (WhatsApp/Telegram)**. Se sintam a vontade para entrar em contato. Além disso, o monitor se encontrará no Laboratório de Sistemas Distribuídos Inteligentes - LSDI (prédio anexo de pesquisa, atrás do CT, do lado esquerdo, segundo andar) nos horários e dias abaixo para tirar dúvidas relacionadas ao trabalho.

  Quarta Feira (22/05/19) das 8:00 as 11:00

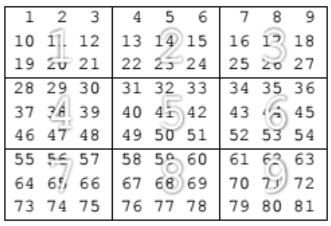
  Sexta Feira (24/05/19) das 14:00 as 18:00

  Sexta Feira (31/05/19) das 14:00 as 18:00

1 Implementação

O objetivo final do algoritmo é: dada uma matriz de números naturais aleatórios (intervalo 0 a 29999) contabilizar quantos números primos existem e o tempo necessário para isso. No entanto, isso será feito de duas formas:

- De modo serial, ou seja, a contagem dos primos será feita um a um, um após o outro. Esse será o seu tempo de referência.
- De modo paralelo. Para tanto, o trabalho de verificar cada número e se for primo contabilizá-lo consistirá na subdivisão da matriz em "macroblocos" (submatrizes), sem qualquer cópia, baseando-se apenas nos índices. Como no exemplo abaixo:



- Ou seja, a matriz acima é 9 x 9 e cada macrobloco é composto por 9 elementos. O macrobloco 1 vai da coluna 0 a 2 e da linha 0 a 2, e assim sucessivamente. Os macroblocos serão as unidades de trabalho de cada thread. Atenção: Nem a matriz nem os macroblocos deverão ser obrigatoriamente quadradas. A única exigência é que todos os macroblocos tenham o mesmo tamanho. Além disso, você deve encontrar alguma forma de PARAMETRIZAR essa divisão (usando a diretiva define, por exemplo) a fim de poder efetuar os testes para diferentes tamanhos de macroblocos. Os macroblocos terão tamanhos que podem variar de desde um único elemento até a matriz toda (equivalente ao caso serial).

- **Essa matriz e a variável que contabiliza o número de números primos encontrados na matriz deverão ser globais (logo, compartilhadas) e únicas.**

Dito qual é o objetivo final da implementação, segue um passo a passo das diretrizes básicas a serem seguidas pelo algoritmo:

- Geração de uma matriz de números naturais aleatórios (intervalo 0 a 29999) usando uma semente pré-definida no código, a fim de sempre ter a mesma "matriz aleatória" para todos os testes. A geração de números aleatórios em C se dá com o uso das funções srand() e rand().

- O tamanho da matriz deverá ser consideravelmente grande a fim de que possam ser efetuadas medidas de desempenho consistentes. Para efeito de comparação, num desktop com 8 GB de RAM e CPU core i5 com 4

  núcleos reais, a verificação serial de uma matriz de 20000 x 20000 levou aproximadamente 42 segundos.

- **Atenção**: Muito cuidado na escolha desse tamanho! Nos testes efetuados , o processo que efetuou o teste com a matriz desse tamanho ocupou aproximadamente 1,5 GB de RAM. No entanto, visto ter o computador uma quantidade de RAM expressivamente maior que isso, pode-se concluir com alguma certeza que a memória secundaria (HD ou SSD, que é muito mais lento que a RAM, obviamente) não fora usada. A escolha das dimensões da matriz deverá levar em conta a configuração do computador onde os testes serão executados a fim de evitar medidas incorretas devido ao uso da memória virtual. Faça testes de olho no consumo de memória ao rodar o executável (use o **Gerenciador de Tarefas**, se estiver no Windows ou os comandos "**top**", "**htop**", "**vmstat**") a fim de definir um tamanho razoável para a matriz. Um tamanho razoável é algum que leve a um tempo de busca para o caso serial superior a 20 segundos.

- **Busca serial** pela quantidade de números primos da matriz gerada acima. Exiba a quantidade de números primos encontrados e o tempo decorrido nessa busca.

- **Busca paralela** pela quantidade de números primos na mesma matriz usada na busca serial. Cabe aqui, algumas dicas:

  A escolha do número de threads para o teste principal deverá levar em conta o número de núcleos reais da CPU que equipa o computador. Se a CPU tiver N >= 2 núcleos reais, crie N threads. Se não souber dessa informação, procure saber para evitar sobrecarregar seu computador desnecessáriamente.

  A atribuição de qual macrobloco será processado em cada momento deverá ser da seguinte forma: Suponha que foram criadas 4 threads. A thread 1 deverá começar a busca no macrobloco 1, a thread 2 no macrobloco 2, a thread 3 no macrobloco 3 e a thread 4 no macrobloco 4. Devido à natureza aleatória dos números (consequentemente do tempo de verificação se o número é primo ou não depender da magnitude do número) e do escalonador do sistema operacional, nada se pode afirmar sobre que thread terminará sua busca primeiro. No entanto, digamos que a thread 2 termine sua busca primeiro. Ela deverá: somar o número de primos encontrado à variável que esteja contabilizando o número total de primos da matriz (ou seja, a thread usará uma variável temporária para contar o número de primos e, após terminada a busca no macrobloco, somará esse valor à variável global) e reiniciar a busca no próximo macrobloco que ainda não foi atribuído a nenhuma thread. A variável que controla quais macroblocos estão livres/alocados deverá ser global (compartilhada). 1. Exemplo: Thread 2 termina. Logo, ela verificará se o macrobloco 5 já foi atribuído a alguma thread. Se não, ela "marca-o" como já atribuído e reinicia seus trabalhos nele. Caso contrário, busca

pelo próximo macrobloco “livre”, até que por fim se esgote os macroblocos a serem buscados. Neste ponto, a thread termina e a thread principal fica esperando que as demais threads terminem.

Atenção: É proibido criar um vetor para armazenar o número de primos encontrados em cada macrobloco e depois somá-los. Como já mencionado, as variáveis que armazenam o número de primos total, a que controla a alocação do macroblocos e matriz principal deverão ser globais e o acesso compartilhado deverá ser controlado.

2 O que enviar ?

Pelo SIGAA, deve ser enviado:

- Código
- Relatório, que deve conter:

  Como executar seu código;

  Bugs encontrados;

  Dificuldades enfrentadas;

  Avaliação do seu Trabalho: Seja Criativo :)

Avalie/critique o os resultados dos testes, faça análises relativas ao tempo de processamento versus diferentes números de threads (Aumente muito, algumas centenas ou mais, o número de threads a fim de que o overhead possa realmente ficar crítico e analise os resultados) e tamanhos de macroblocos. Você encontrará resultados bem interessantes quando o tamanho dos macroblocos for muito grande ou muito pequenos! Em resumo, brique com seu trabalho, e analise: o que você pôde aprender com esse trabalho?