Universidade Federal
de São João del-Rei

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático 1 - Computação Paralela - Identificação de números primos

Amanda de Araújo Morato - 152050038
Guilherme Morbeck Rodrigues - 182050054

São João del-Rei
Outubro de 2022

Sumário

1 Introdução

Um dos grandes desafios atuais na Ciência da Computação é tornar viáveis soluções que forneçam respostas com mais exatidão e que reduzam o tempo de processamento de algoritmos. Uma das áreas que se dedica a propor tal viabilidade é a Computação Paralela.

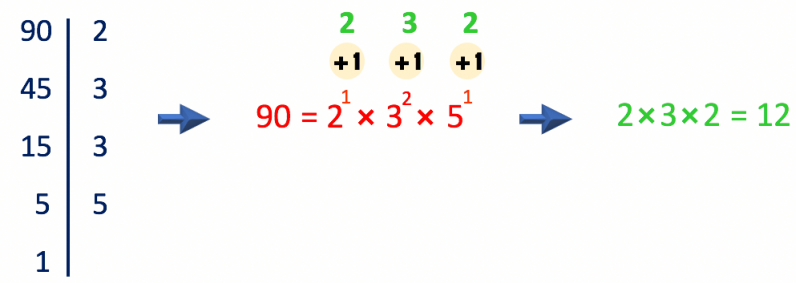
A Computação Paralela é uma forma de computação que opera sob o princípio de que grandes problemas geralmente podem ser divididos em problemas menores, e que esses subproblemas podem ser resolvidos simultaneamente por diferentes processadores. Tal divisão pode ocorrer entre os dados e/ou entre tarefas, tendo se tornado ainda mais viável para o público geral após a criação de processadores multinúcleo.

Para que tenhamos um contato com a Programação Paralela, esse trabalho prático propõe uma implementação mestre-escravo de um algoritmo que identifica a quantidade de divisores de cada valor em um vetor de inteiros. Para essa implementação, foi utilizada a linguagem de programação C e o conjunto de bibliotecas OpenMPI.

2 Perfil de desempenho sequencial

Como ponto de partida, desenvolvemos um algoritmo sequencial para resolver o problema em questão. Nesse algoritmo, o arquivo de entrada é lido e os valores contidos nele são carregados para a memória principal em um vetor de tamanho S, onde S é o número de linhas do arquivo. Em seguida, é chamada uma função onde, para cada valor N do vetor, os números de 1 a N/2 são percorridos e verifica-se quais deles são divisores de N. Essa verificação é salva em um contador inicializado como 1, já que todo número também é divisível por ele mesmo.

Como esperado, a execução do programa é bastante lenta. Tendo isso em vista, antes de realizar a paralelização, foi implementada também uma função dinâmica para fins comparativos. Nessa função, tomamos a fatoração de N em primos e calculamos o produto dos expoentes destes primos adicionados em 1, assim como é feito no exemplo da figura abaixo, onde 90 possui 12 divisores.



3 Paralelização

3.1 Identificação das oportunidades de paralelização

A oportunidade de paralelização mais evidente é a execução das funções que realizam as divisões e fatoração. Essas funções são as mais demandantes computacionalmente, enquanto as outras partes do programa, como leitura e escrita no arquivo, não possuem tempo de execução considerável, correspondendo a menos de 2% do código.

3.2 Algoritmo paralelo

A paralelização do programa foi feita através da implementação da estratégia mestre-escravo em um único programa para todos os processadores, ou seja, com a técnica de implementação Single Program, Multiple Data (SPMD).

Após a leitura do arquivo pelo processo mestre, os dados são carregados em um vetor (nums_global). O tamanho desse vetor é enviado para todos os processos escravos, utilizando o MPI_Bcast para que esse valor seja utilizado por todos ao longo da execução. Em seguida, cada processo escravo aloca memória suficiente para receber o subvetor nums_local e o MPI_Scatter é usado para enviar os dados do vetor nums_global para todos os processos.

```
//Enviando o tamanho do vetor de elementos para todos os
   processos.
MPI_Bcast(&num_elementos_global, 1, MPI_INT, 0, MPI_COMM_WORLD);

//Alocando os vetores de elementos locais.
int num_elementos_local = (num_elementos_global/size_of_cluster);
nums_local = malloc(sizeof(int) * num_elementos_local);

//Enviando uma parte do vetor global para cada processo.
MPI_Scatter(nums_global, num_elementos_local, MPI_INT, nums_local
   , num_elementos_local, MPI_INT, MESTRE, MPI_COMM_WORLD);
```

Os processos escravos chamam a função para calcular a quantidade de divisores e salvam os resultados em um vetor (divisores_local). É então utilizado o MPI_Gather para enviar ao mestre todos os vetores divisores_local na mesma ordem em que os vetores nums_local foram recebidos. O processo mestre salva todos os dados recebidos no MPI_Gather e o programa é finalizado.

```
//Calculando os divisores de cada elemento dos vetores locais
   paralelamente.
int divisores_local[num_elementos_local];
for (int i = 0; i < num_elementos_local; i++){
    divisores_local[i] = Calcula_Divisores(nums_local[i]);
}

//Recebendo os vetores locais de todos os processos.
MPI_Gather(&divisores_local, num_elementos_local, MPI_INT,
   nums_global, num_elementos_local, MPI_INT, MESTRE,
   MPI_COMM_WORLD);

if (process_rank == MESTRE) { ... //Salva todos os dados no
   arquivo de saida.
```

3.3 Avaliação dos ganhos com a paralelização

As execuções para avaliação do algoritmo sequencial e paralelo foram realizadas em um computador com processador Intel Core I5, 4 núcleos de processamento de 2.50GHz, sistema operacional de 64 bits e memória RAM de 8 GB.

Após múltiplas execuções do algoritmo sequencial, os seguintes resultados foram obtidos:

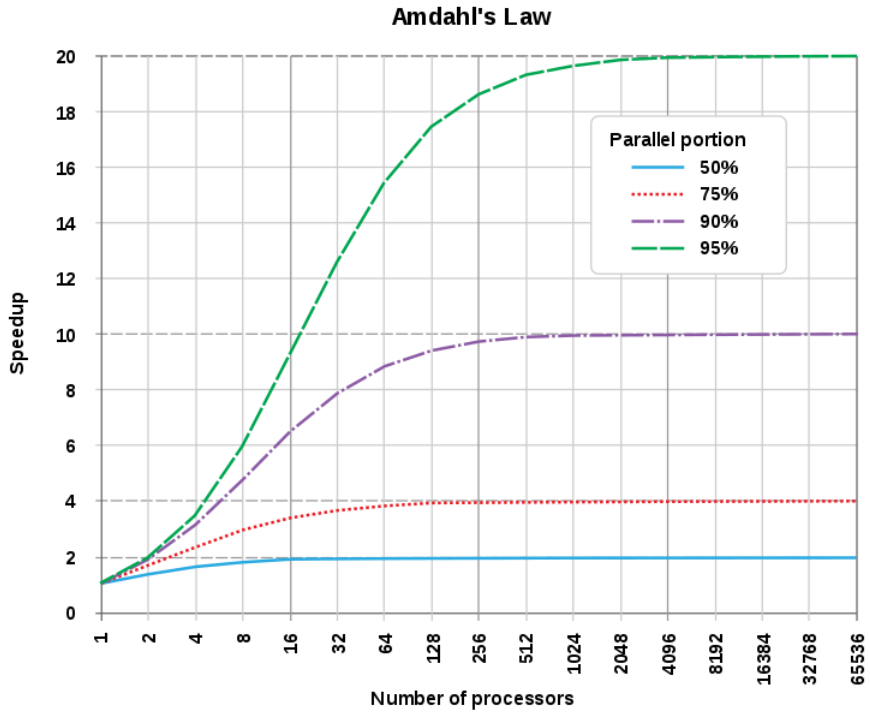
ALGORITMO SEQUENCIAL	
FUNÇÃO	TEMPO DE EXECUÇÃO
Calcula_Divisores	312,922811
Calcula_Divisores_Dinâmico	45,175951

Houve um ganho considerável de tempo ao comparar os dois algoritmos sequenciais, sendo o algoritmo dinâmico aproximadamente 6,93 vezes mais rápido que o trivial. Já para as execuções do algoritmo paralelo, tivemos os resultados seguintes:

ALGORITMO PARALELIZADO			
FUNÇÃO	Nº DE PROCESSADORES	TEMPO DE EXECUÇÃO	SPEEDUP
Calcula_Divisores	1	269,420883	1,161464574
	2	139,440523	2,244131077
	3	128,531284	2,434604256
	4	115,494513	2,709417122
Calcula_Divisores_Dinâmico	1	45,323595	0,996742447
	2	24,400784	1,851413914
	3	20,920086	2,159453408
	4	19,286425	2,342370398

Podemos perceber que o tempo de execução à medida que aumentamos a quantidade de processadores não aumentou linearmente. Após análise e pesquisa, notamos que os valores de speedup obtidos seguem a lei de Amdahl. Essa lei é utilizada para encontrar a melhora máxima esperada para um sistema quando apenas uma parte dele é melhorada. Quando são utilizados múltiplos processadores, o speedup de um programa fica limitado pelo tempo necessário para executar a parte sequencial do código.

Como exemplo temos que, se um programa precisasse de 20 horas para executar e a parte que não pode ser paralelizada levasse uma hora usando um núcleo de processamento, não é possível haver um aumento de velocidade maior que 20x, independente do número de processadores dedicados à execução desse programa. Ou seja, ao utilizar múltiplos processadores, o speedup de um programa é limitado pela fração sequencial do programa, como pode ser visto no gráfico abaixo.



Considerando P como a parte paralelizável de um programa e 1-P como a parte não paralelizável, a lei de Amdahl afirma que o speedup máximo que pode ser encontrado utilizando N processadores é $S(N) = \frac{1}{1-P+\frac{P}{N}}$.

4 Conclusão

A implementação de códigos em paralelo tem como grande desafio a quebra do paradigma sequencial que é utilizado na grande maioria dos programas. A realização do trabalho forneceu maior contato com a computação paralela e nos mostrou como sua implementação pode influenciar na execução de programas, forçando a quebra de tal paradigma.

Visando melhorar o tempo de execução, notamos que o mapeamento de pontos paralelizáveis de um programa é um ponto crucial para obter melhorias de desempenho. Esse mapeamento passa por diversas análises em questão de como a memória e dados serão compartilhados pelos processadores e em como manobrar as dependências das tarefas.

Como apresentado nos resultados dos testes, a utilização de técnicas de paralelização pode trazer melhorias de desempenho no programa à medida que as tarefas mais custosas computacionalmente possam ser paralelizáveis, respeitando as restrições impostas pela comunicação de processadores e pontos sequenciais do código. Essas restrições fazem com que a curva de melhoria de desempenho de um determinado programa seja sigmoidal, tendendo a diminuir o ganho a partir de uma quantidade de processadores. Existe um valor máximo de quanto o custo da comunicação e paralelização sobrepõe o custo de processar e executar sequencialmente.

Portanto, a utilização de técnicas de paralelização pode trazer grandes melhorias de desempenho aos algoritmos, principalmente em códigos que não exista grande dependência de comunicação entre os processos, podendo então tratar grandes quantidades de dados em um menor tempo.

5 Referências

- https://pt.wikipedia.org/wiki/Lei_de_Amdahl
- https://www.open-mpi.org/
- https://mpitutorial.com/
- https://www.codingame.com/playgrounds/54443/openmp/hello-openmp
- https://blog.professorferretto.com.br/como-determinar-a-quantidade-de-divisores-de-um-numero/