Geração de Números Perfeitos

Rafael Tenfen[1]

[1] Universidade do Estado de Santa Catarina (UDESC)

rafaeltenfen.rt@gmail.com

Abstract. *Perfect numbers are positive integer that is equal to the sum of its positive divisors, excluding the number itself. There are some ways to verify if a number is perfect, like by example, using brute force trying to find one by one the predecessor divisors, or using Euclid's formula. This article will approach the generating of perfect numbers sequentially and the refactory for the same algorithm using techniques of advanced parallel programming.*

Resumo. *Números perfeitos, são números inteiros positivos onde a soma de seus divisores positivos excluindo o próprio número, é igual ao seu valor. Há várias formas utilizadas para verificar se um número é perfeito, como por exemplo encontrar um a um os divisores dos antecessores, ou utilizando a formula de Euclid. Nesse trabalho, sera abordado a questão de geração de números perfeitos de modo sequencial, e como realizar a refatoração para o algoritmo utilizando técnicas de programação paralela avançada.*

1. Introdução

Um número natural é dito perfeito se é igual a soma de seus divisores, excluindo o próprio número. Assim, n é perfeito se e somente se

$$\sigma(n) = \sum_{d|n} d = 2n$$

Matemáticos tem estudado esses números por aproximadamente dois milênios, no entanto, o mistério de suas propriedades e sua existência permanece não solucionado, levantando algumas questões. Há infinitos números perfeitos? Existe algum número perfeito ímpar? Todas essas questões, resistiram ao teste do tempo e permanecem em aberto [Holdener 2002].

Utilizando a técnica de *Mersenne prime* em que $(2^p - 1)$ talvez seja primo. Um número primo p é um número inteiro positivo que contém somente dois positivos divisores, 1 e o próprio número p [Crandall and Pomerance 2006]. Até o momento, foram descobertos cinquenta e um números perfeitos, o último por sinal em sete de dezembro de dois mil e dezenove, pode ser representado da seguinte forma: $2^{82.589.932}(2^{82.589.933} - 1)$, nenhum dos números perfeitos encontrados é impar [Mersenne Research 2019]. Contudo, ninguém pode garantir que não existam números perfeitos ímpares e nem a sua finitude.

Euclid foi bem sucedido em utilizar números primos e *Mersenne primes* para estabelecer o **Teorema A**. Utilizando o teorema de Euclid, foi possível encontrar os quatro primeiros números perfeitos - 6, 28, 496, 8128. Logo após, Euler em 1747 com o **Teorema B**. mostrou que todo número perfeito que é par pode ser representado pela aplicação da regra de Euclid [Pollack and Shevelev 2012].

Teorema A (Euclid). Se p é um número primo e $2^p - 1$ também é um número primo. Então $n = 2^{p-1}(2^p - 1)$ é um número perfeito.

Teorema B (Euler). Todo número perfeito par têm forma $2^{p-1}(2^p - 1)$, onde p e $2^p - 1$ são primos.

2. Problema

A implementação de geração de números perfeitos, assim como inúmeros outros problemas de programação pode ser feita de modo sequencial, mas preferencialmente deve ser utilizado técnicas de programação paralela para o seu desenvolvimento, utilizando essas técnicas espera-se uma melhora na performance em relação ao mesmo problema escrito de modo sequencial.

Para aplicar a paralelização geração de números perfeitos, será utilizado um arquitetura que pode executar, de forma paralela, fluxos de instruções independentes, sendo que cada fluxo tem seu próprio fluxo de dados, chamada de MIMD (*multiple instruction, multiple data*) em conjunto com o OpenMP. O OpenMP é definido como uma API (*Application Programming Interface*) para programação paralela de memória compartilhada. Ele foi projetado para que toda *thread* existente na solução tenha possibilidade de acessar toda a memória disponível [Bianchini et al.].

3. Proposta de Implementação

Nessa seção, sera abordado a metodologia **PCAM** que define quatro etapas do processo de paralelização para o problema de geração de números perfeitos. Essa separação de 4 etapas, apresentada na Figura 1, reflete uma forma que induz a pensar nos aspectos fundamentais de paralelização, permitindo a quebra de um problema em pedaços pequenos que possam ser justamente calculados ao mesmo tempo por diferentes núcleos de processamento [Schnorr and Nesi].

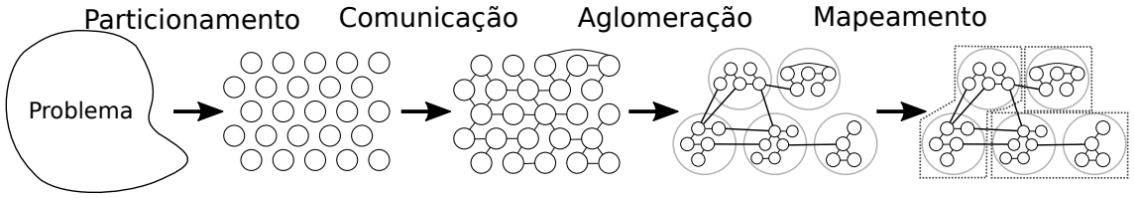

Figure 1. Modelo metodológico PCAM com suas quatro fases: particionamento, para quebrar o problema em pedaços menores; comunicação, para definir quais pedaços devem comunicar-se entre si; aglomeração, para agrupar pedaços com critérios de localidade; e mapeamento, para atribuir a unidades computacionais os grupos que devem ser processados [Schnorr and Nesi]

Utilizando programação sequencial foi implementado duas formas de geração de números perfeitos. A primeira forma denominada de *"brute force"*, utiliza-se apenas da técnica de dividir números antecessores um a um para verificar seus possíveis divisores, realizar a soma desses divisores e verificar se o resultado da soma é igual ao número, definindo assim o número como perfeito.

A segunda forma denominada de *"Euclid"*, utiliza-se da fórmula de Euclid, apresentada no **Teorema A** da seção 1 que verifica se um número é perfeito utilizando a

verificação de números primos. Dessa forma, é utilizado a geração de números primos, aplicados a fórmula de Euclid para geração dos números perfeitos.

3.1. Particionamento

O particionamento é uma abordagem significativa para uma eficiente atribuição de executável a *tasks*, a fim de utilizar computação paralela no processamento de instruções. O particionamento é uma divisão de partes independentes para resolvê-las em paralelo. Desse modo, pequenas tarefas devem ser definidas, para serem processadas de forma otimizada e evitem dados e cálculos duplicados. Quanto menores as partições ficam, maior o potencial do paralelismo [Hoettger et al. 2013].

Na paralelização do código sequencial criado, sera utilizado um conjunto de particionamento de dados e também o particionamento das operações, também chamadados de decomposição de domínio e decomposição funcional. A fim de paralelizar a geração de números perfeitos, deve-se verificar se um número é primo em concorrência para a forma de *"Euclid"*, ou de fato executar em paralelo a verificação do número perfeito na forma de *"brute force"*.

3.2. Comunicação

Em um programa paralelo, é comum que as tarefas necessitem trocar informações para realizar suas operações. Em **PCAM**, a fase de comunicação envolve justamente o projeto destas atividades de troca de dados. Nos cenários onde inexiste a necessidade de comunicação [Schnorr and Nesi].

Para o problema abordado, será utilizado comunicação local em que a operação de computação de uma determinada tarefa (um ponto no domínio de problema discretizado) necessita dados de um pequeno número de tarefas (pontos) vizinhas. Nesse contexto, as tarefas tem vizinhos claramente definidos e imutáveis em função do particionamento estabelecido anteriormente, dessa forma, as comunicações são frequentemente fixas, ditas estruturadas. A discretização a ser realizada de maneira estática é fixa desde a concepção na fase de particionamento do projeto até a execução do código.

3.3. Aglomeração

A fase de aglomeração tem por objetivo tornar o algoritmo abstrato das fases precedentes em algo mais realista, de acordo com os limites impostos pela configuração da plataforma de execução alvo. O objetivo principal é obter um programa eficiente, além de que nessa fase, deve ser definido a granularidade não fixa das tarefas [Schnorr and Nesi].

A aglomeração na geração de números perfeitos deve acontecer ao realizar a soma dos possíveis divisores na forma *"brute force"* a fim de verificar o perfeccionismo do número. Além disso, na forma de *"Euclid"* a aglomeração toma papel ao agrupar os números perfeitos gerados.

3.4. Mapeamento

O quarto e ultimo estágio da metodologia **PCAM**, chamado de mapeamento, consiste em definir onde cada tarefa será executada. Os requisitos fundamentais na atividade explícita de mapeamento envolvem colocar tarefas concorrentes em unidades de processamento

diferentes, de forma que tais tarefas sejam de fato executadas em paralelo, além de alocar tarefas que se comunicam frequentemente em locais próximos da topologia de interconexão, tanto física quanto lógica [Schnorr and Nesi].

No problema apresentado nesse trabalho, com base nos demais estágios do **PCAM** em que o particionamento utilizará um conjunto de particionamento de dados e também particionamento das operações, a comunicação definida é estática, estruturada e local. Então, o mapeamento a ser utilizado é de uma *thread* por processador.

4. Conclusão

As fases precedentes da metodologia **PCAM** permitem o particionamento e a definição das comunicações necessárias para a resolução paralela de um problema. O resultado destas fases é um algoritmo abstrato que contém potencialmente muitas tarefas, tendo em vista que o objetivo é identificar a menor operação possível que possa ser executada concorrentemente com as demais [Schnorr and Nesi].

Enfim, tem-se por objetivo aplicar as técnicas da metodologia **PCAM** apresentadas na seção 3 que teve como base o livro *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering* [Foster], com o auxílio do OpenMP para aprimorar a performance da geração de números perfeitos utilizando as técnicas paralelas avançadas.

References

Bianchini, C. P., Vilabôas, F. G., and Castro, L. N. Paralelismo de tarefas utilizando openmp 4.5.

Crandall, R. and Pomerance, C. B. (2006). *Prime numbers: a computational perspective*, volume 182. Springer Science & Business Media.

Foster, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA.

Hoettger, R., Igel, B., and Kamsties, E. (2013). A novel partitioning and tracing approach for distributed systems based on vector clocks. In *2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, volume 2, pages 670–675. IEEE.

Holdener, J. A. (2002). A theorem of touchard on the form of odd perfect numbers. *The American mathematical monthly*, 109(7):661–663.

Mersenne Research, I. (2019). List of known mersenne prime numbers - primenet.

Pollack, P. and Shevelev, V. (2012). On perfect and near-perfect numbers. *Journal of Number Theory*, 132(12):3037–3046.

Schnorr, L. and Nesi, L. *Projetando e Construindo Programas Paralelos. Anais da Escola Regional de Alto Desempenho da Região Sul*, volume v. 19ed.