Cálculo de Primos de Mersenne com o Teste de Lucas-Lehmer

Este programa foi projetado para identificar **primos de Mersenne**, que são números na forma $M_p = 2^p - 1$, onde p é um número primo. Para realizar essa tarefa, ele utiliza o **Teste de Lucas-Lehmer**, um método eficiente para verificar a primalidade de números de Mersenne.

---

1. Estrutura Geral do Programa

O programa é dividido em três partes principais:

1. **Função de Lucas-Lehmer**:
    - A função verifica se um número M_p (número de Mersenne) é primo.
    - Inicialmente, calcula-se $M_p = 2^p - 1$.
    - Em seguida, a sequência de Lucas-Lehmer é gerada e verificada. Começa-se com $S_0 = 4$ e, para cada iteração $S_{i+1} = (S_i^2 - 2) \bmod M_p$ Após $p-2$ iterações, se o valor final for 0, M_p é primo.
2. **Função de Verificação de Primos**:
    - Antes de verificar se M_p é primo, o programa verifica se p é primo.
    - A função utiliza o método de tentativa e erro (divisores até a raiz quadrada de p) para determinar a primalidade de p.
    - Esta função existe pois é necessário que o p para o cálculo do primo de Mersenne seja também um primo
3. **Função Principal (Main)**:
    - A função principal itera sobre todos os números ímpares até um limite predefinido (LIM), verificando se são primos.
    - Para cada número primo p, calcula $M_p = 2^p - 1$.
    - Em seguida, o teste de Lucas-Lehmer é aplicado a M_p . Se M_p for primo, o número de Mersenne é exibido.

---

2. Algoritmo do Teste de Lucas-Lehmer

O teste baseia-se em uma sequência matemática específica que só retorna zero no final quando MpM_pMp é primo:

- **Passo 1**: Calcular $M_p = 2p - 1$.
- **Passo 2**: Iniciar $S_0 = 4$.
- **Passo 3**: Iterar $S_{i+1} = (S_i^2 - 2) \bmod M_p$ para $p-2$ passos
- **Passo 4**: Verificar se $S_{p-2} = 0$. Se sim, M_p é primo.

---

3. Uso da Biblioteca GMP

O programa utiliza a biblioteca GMP para manipulação de números grandes. Isto é essencial porque os números de Mersenne crescem exponencialmente à medida que p aumenta. A biblioteca permite:

- Operações como potência (mpz_ui_pow_ui), subtração (mpz_sub_ui), módulo (mpz_mod) e comparação (mpz_cmp_ui).
- Representação e impressão de grandes números (mpz_out_str).

---

4. Limitação e Escalabilidade

- **Limitação**: O limite (LIM) restringe os valores de p verificados. Se o limite for aumentado, o programa poderá identificar primos de Mersenne maiores, mas demandará mais tempo e memória.
- **Escalabilidade**: Graças à GMP, a manipulação de números grandes é eficiente, permitindo expandir os cálculos para valores altos de p, desde que os recursos do sistema suportem.

---

5. Saída do Programa

Para cada p que gera um número de Mersenne primo, o programa exibe o seguinte:

- O valor de p.
- O número primo de Mersenne.

---

6. Uso do paralelismo

a) Organização do Algoritmo

- Na versão paralela, o código é dividido em três tarefas principais, realizadas simultaneamente:
    1. **Busca por números primos (checar_primo):**
        - Um thread identifica números primos e os armazena em um array.
    2. **Cálculo dos números de Mersenne ($2^p - 1$):**
        - Outro thread utiliza os números primos encontrados para calcular os números de Mersenne.
    3. **Teste de Lucas-Lehmer:**
        - As demais threads executam o teste de Lucas-Lehmer nos números de Mersenne calculados.

7.Comparação de Desempenho: Speedup

Para medir o desempenho das versões sequencial e paralela do código, realizamos uma análise de **speedup**, definida como a razão entre o tempo de execução da versão sequencial (T_s) e o tempo de execução da versão paralela (T_p):

Speedup = T_s/T_p

Resultados:

- LIM=10000, NUM_THREADS=16:
    - T_s=235602 ms
    - T_p=79130 ms
    - Speedup = 2,97x
- LIM=10000, NUM_THREADS=4:
    - T_s=235602 ms
    - T_p= 164301 ms
    - Speedup = 1,43x