Trabalho Final de Programação Paralela - Paralelizando Quebra de Chaves do Algoritmo RSA

Miguel Alfredo Nunes

miguel.alfredo.nunes@gmail.com

07/12/2022

Sumário

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**
- História do algoritmo, segundo a Wikipedia:

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**
- História do algoritmo, segundo a Wikipedia:
  - Entre 1976 e 1977 Rivest, Shamir e Adleman tentaram desenvolver uma função que calculasse números, mas que era difícil de inverter;

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**
- História do algoritmo, segundo a Wikipedia:
  - Entre 1976 e 1977 Rivest, Shamir e Adleman tentaram desenvolver uma função que calculasse números, mas que era difícil de inverter;
  - Em abril de 1977 eles passaram o feriado de Páscoa Judaica na casa de um aluno;

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**
- História do algoritmo, segundo a Wikipedia:
  - Entre 1976 e 1977 Rivest, Shamir e Adleman tentaram desenvolver uma função que calculasse números, mas que era difícil de inverter;
  - Em abril de 1977 eles passaram o feriado de Páscoa Judaica na casa de um aluno;
  - Lá, eles consumiram "uma quantidade generosa de vinho" antes de voltar para casa;

Algoritmo RSA

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **Rivest**, Adi **Shamir**, e Leonard **Adleman**
- História do algoritmo, segundo a Wikipedia:
  - Entre 1976 e 1977 Rivest, Shamir e Adleman tentaram desenvolver uma função que calculasse números, mas que era difícil de inverter;
  - Em abril de 1977 eles passaram o feriado de Páscoa Judaica na casa de um aluno;
  - Lá, eles consumiram "uma quantidade generosa de vinho" antes de voltar para casa;
  - Rivest, sem conseguir dormir, deitou no seu sofá com um livro de matemática - pela manhã ele tinha formalizado o algoritmo.

Algoritmo RSA

1. Gerar aleatoriamente números primos P e Q, diferentes entre si;
2. Calcular $N = P \times Q$;
3. Calcular um número E que não tenha divisores em comum com $(P-1) \times (Q-1)$;
4. Calcular um número D tal que o resto da divisão de $E \times D$ por $(P-1) \times (Q-1)$ é igual a 1
   - D é dito inverso modular de E modulo $(P-1) \times (Q-1)$

O par $\langle E, N \rangle$ compõe a chave pública, $\langle D, N \rangle$ compõe a chave privada.

Algoritmo RSA

- Sendo m uma mensagem representada numericamente (por exemplo o código ASCII de uma letra).
  - Cifrar: m^E;
  - Decifrar: m^D;
  - $(m^E)^D \equiv (m^D)^E \equiv m \pmod{N}$;
  - $(m^D)^E \equiv m \pmod{N}$ quer dizer que o resto da divisão de $(m^D)^E$ por N é igual a m.
- Fatorando N nos seus componente P e Q e tendo o valor E, é fácil obter D;
- Segurança da criptografia se dá pela dificuldade em fazer isso - problema da fatoração de primos;
- Trabalho final da matéria de Complexidade de Algoritmos (CAL).

Implementação

- C++ usando OpenMP, MPI e Gnu Multi Precision (GMP) - biblioteca para trabalhar com números de tamanho arbitrário;
- Um processo gera as chaves serialmente, são salvas em arquivos que são lidos pelo processo que vai fatorar paralelamente;
- Ideia principal é, dado N, calcular os intervalos: $\left[3, \left(\frac{\sqrt{N}}{x}\right)\right), \left[\left(\frac{\sqrt{N}}{x}\right), 2 \times \left(\frac{\sqrt{N}}{x}\right)\right), \ldots, \left[\left(\frac{\sqrt{N}}{x}\right) \times (x-1), \sqrt{N}\right)$ onde x é o número de tarefas;

Implementação

- Para obter os resultados aqui apresentados, foi executado em 8 processos que se comunicam pelo MPI, cada um com 8 threads do OpenMP;
- Tomando $x = \text{processos} \times \text{threads} \times 8$, algoritmo executou sobre 512 intervalos;
- OpenMP e MPI não sabem como tratar os tipos do GMP, logo tive que encapsular os intervalos em uma struct e passar essa struct para funções que lidam com números do GMP;
- Não foi possível utilizar MPI para fazer comunicação entre diversos computadores, 8 processos rodando em `localhost`.

Implementação

```
typedef struct
{
    std::string lower;
    std::string upper;
    std::string valorP;
    std::string valorQ;
}block;
```

- GMP tem funções que traduzem números de/para strings de C, tipo `string` do C++ tem métodos para converter de/para strings de C;

Implementação

- Ideia original era calcular os intervalos e comunicar um array de `blocks` por MPI para todos os processos;
- Isso gerou muitos problemas, solução alternativa foi paralelizar a computação dos intervalos – cada processo calcula apenas uma porção dos intervalos:

```
int turn = 0;
for(int i = 1; i < total_blocos; i++){
    if(turn == PROCS)
        turn = 0;
    if(rank != turn){
        turn++;
        continue;
    }
    // calcula o intervalo
    turn++;
}
```

Implementação

```
if(rank == 0)
    inicio = clock();
#pragma omp parallel
{// REGIAO PARALELA
    #pragma omp single
    {// criando tasks
        for(auto & bloco: dados){
        #pragma omp task
        {
            forcabruta_paralelo(&bloco,N, rank);
        }
        }
    }// fim do single, executando as tasks
}// FIM DA REGIAO PARALELA
if(rank == 0)
    termino = clock();
```

Implementação

```
int forcabruta_paralelo(block * bloco, mpz_t N,
    int rank){

// tira os dados do bloco

// testa um por um os numeros do limite
// inferior ao limite superior do bloco
    if(mpz_divisible_p(N, FatorTestado) != 0){
        // calcula outro fator

        DONE = 1; // global
        MPI_Bcast(&DONE, 1, MPI_INT, rank,
            MPI_COMM_WORLD);
        // comunica todos os processos que terminou
    }
}
```

Tempos de Execução Serial

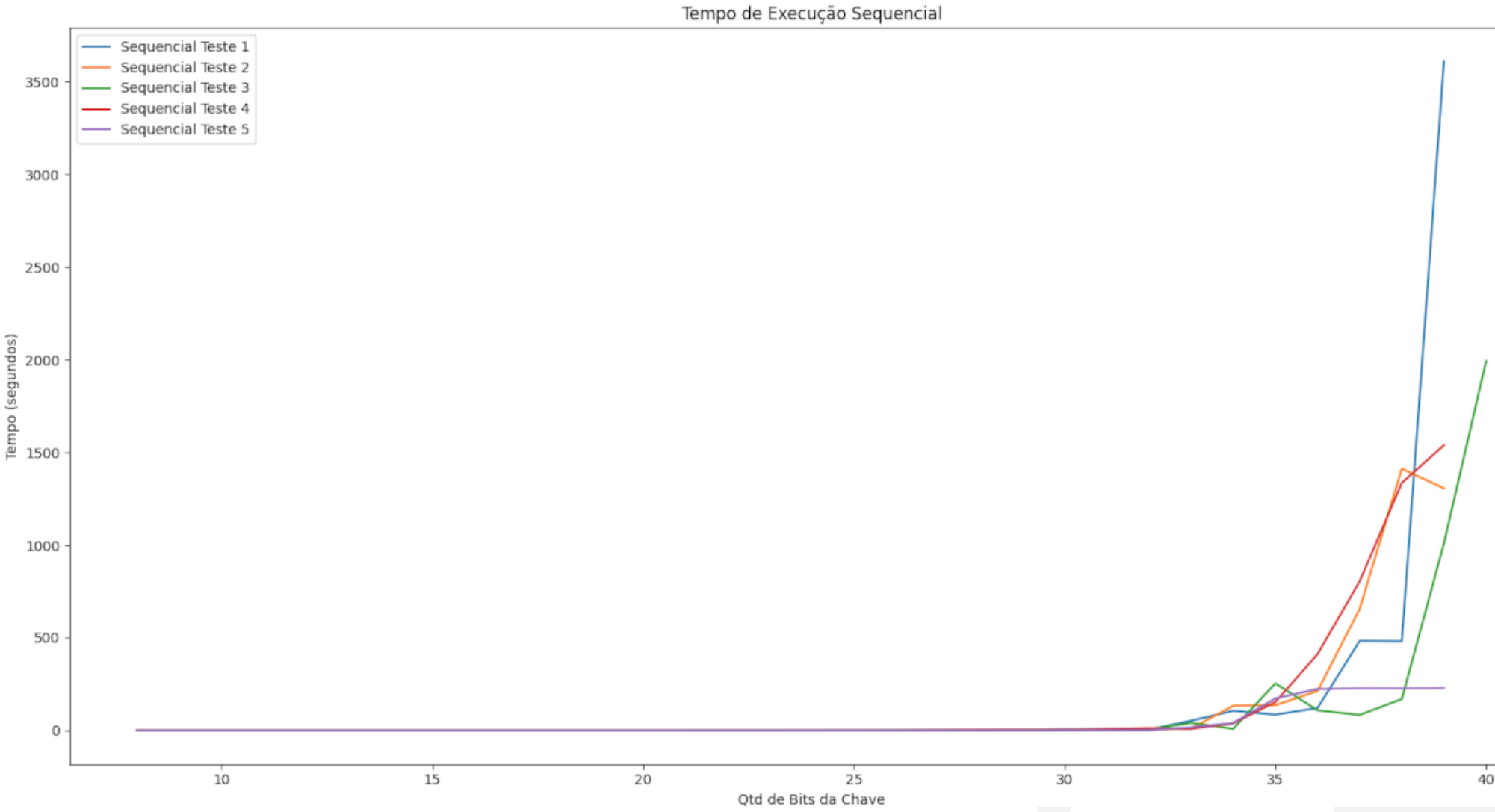

Figura: Gráfico do Tempo de Execução Serial

Tempos de Execução Paralelos

Tempos Serial e Paralelo

Speed Up e Eficiência

Gráficos