

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

Miguel Alfredo Nunes

`miguel.alfredo.nunes@gmail.com`

08/12/2022

1 Algoritmo RSA

2 Implementação

3 Desempenho



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

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **R**ivest, Adi **S**hamir, e Leonard **A**dleman

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **R**ivest, Adi **S**hamir, e Leonard **A**dleman
- História do algoritmo, segundo a Wikipedia:

- Sistema criptográfico de chave pública, qualquer um com a chave pública consegue cifrar, porém apenas a chave privada consegue decifrar;
- RSA: Ron **R**ivest, Adi **S**hamir, e Leonard **A**dleman
- 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;

- 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;

- 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;

- 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.

- 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.

- 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).

- 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:
 $\left[3, \left(\frac{\sqrt{N}}{x}\right)\right), \left[\left(\frac{\sqrt{N}}{x}\right), 2 \times \left(\frac{\sqrt{N}}{x}\right)\right), \dots, \left[\left(\frac{\sqrt{N}}{x}\right) \times (x-1), \sqrt{N}\right)$
onde x é o número de tarefas;

- 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.

```
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){

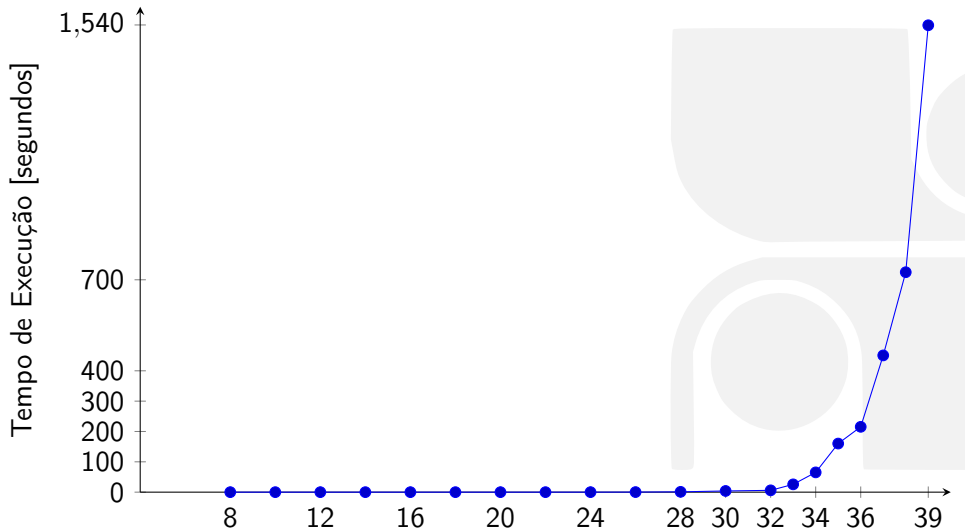
// tira os dados do bloco

// 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 Sequenciais

Média do Tempo de 5 Execuções Sequenciais



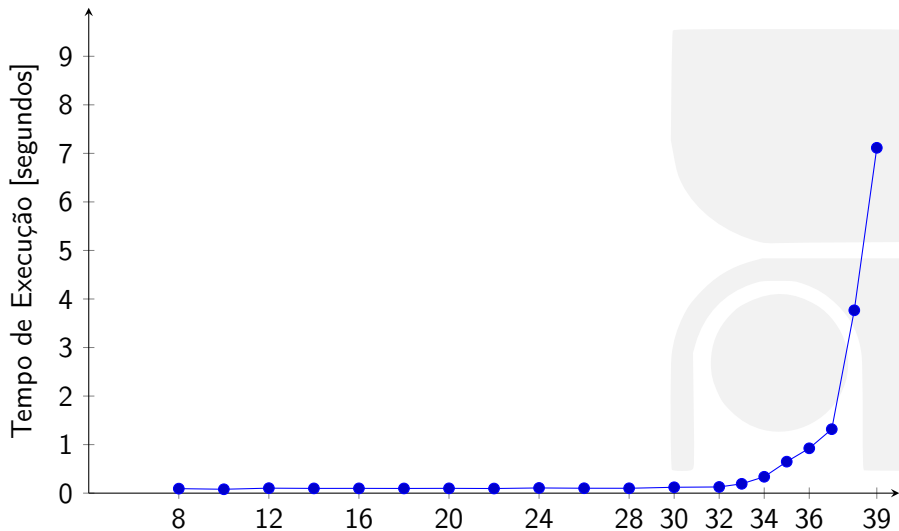
Tempo de Execução Sequenciais

| Bits | Tempos (s) | Desvio Padrão | Bits | Tempo (s) | Desvio Padrão |
|------|------------|---------------|------|-----------|---------------|
| 8 | 9.8e-06 | 0.000001 | 28 | 1.040 | 0.811 |
| 10 | 9.4e-06 | 0.000003 | 30 | 3.924 | 1.681 |
| 12 | 1.56e-05 | 0.000005 | 32 | 5.983 | 3.844 |
| 14 | 6.05e-05 | 0.000065 | 33 | 25.687 | 19.796 |
| 16 | 0.0001 | 0.000222 | 34 | 65.173 | 52.038 |
| 18 | 0.0009 | 0.000484 | 35 | 160.161 | 61.661 |
| 20 | 0.004 | 0.002550 | 36 | 215.341 | 121.509 |
| 22 | 0.012 | 0.017561 | 37 | 450.763 | 297.031 |
| 24 | 0.0629 | 0.052313 | 38 | 724.968 | 604.576 |
| 26 | 0.228 | 0.187669 | 39 | 1539.012 | 1259.526 |

Tabela: Tempos de Execução Sequencial

Tempos de Execução Paralelos

Média do Tempo de 5 Execuções Paralelas



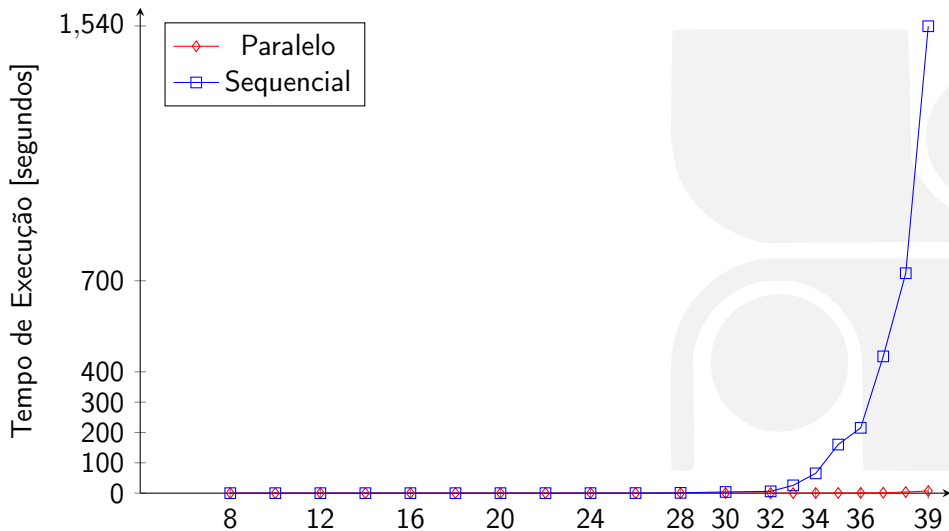
Tempo de Execução Paralelo

| Bits | Tempos (s) | Desvio Padrão | Bits | Tempo (s) | Desvio Padrão |
|------|------------|---------------|------|-----------|---------------|
| 8 | 0.092975 | 0.010403 | 28 | 0.100403 | 0.019439 |
| 10 | 0.080127 | 0.012217 | 30 | 0.121074 | 0.016820 |
| 12 | 0.102825 | 0.014666 | 32 | 0.127594 | 0.037898 |
| 14 | 0.097613 | 0.011124 | 33 | 0.192118 | 0.060779 |
| 16 | 0.098501 | 0.009293 | 34 | 0.337218 | 0.124848 |
| 18 | 0.096620 | 0.012456 | 35 | 0.648707 | 0.214186 |
| 20 | 0.099038 | 0.021616 | 36 | 0.923999 | 0.217719 |
| 22 | 0.095208 | 0.013887 | 37 | 1.317791 | 0.606575 |
| 24 | 0.107163 | 0.033128 | 38 | 3.767034 | 1.718947 |
| 26 | 0.101060 | 0.018860 | 39 | 7.114548 | 4.065866 |

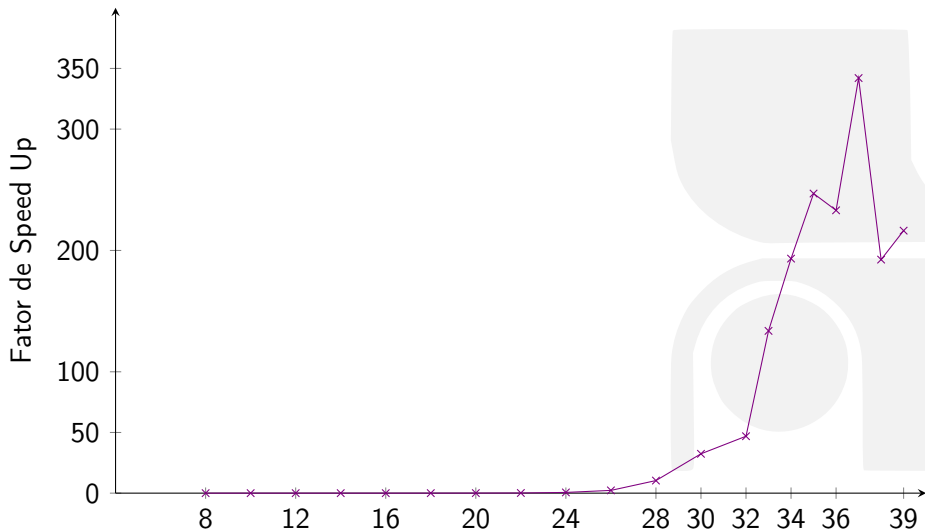
Tabela: Tempos de Execução Paralelo

Tempos Sequencial e Paralelo

Média do Tempo de 5 Execuções Sequenciais e Paralelas



Speed Up em Função do Número de Bits de uma Chave



| Bits | SpeedUp | Eficiência | Bits | SpeedUp | Eficiência |
|------|---------|------------|------|---------|------------|
| 8 | 0.00 | 0.00 % | 28 | 10.36 | 16.19 % |
| 10 | 0.00 | 0.00 % | 30 | 32.42 | 50.65 % |
| 12 | 0.00 | 0.00 % | 32 | 46.89 | 73.27 % |
| 14 | 0.00 | 0.00 % | 33 | 133.71 | 208.92 % |
| 16 | 0.00 | 0.00 % | 34 | 193.27 | 301.98 % |
| 18 | 0.01 | 0.02 % | 35 | 246.89 | 385.77 % |
| 20 | 0.04 | 0.07 % | 36 | 233.05 | 364.15 % |
| 22 | 0.13 | 0.20 % | 37 | 342.06 | 534.47 % |
| 24 | 0.59 | 0.92 % | 38 | 192.45 | 300.70 % |
| 26 | 2.26 | 3.54 % | 39 | 216.32 | 338.00 % |

Tabela: Speed Up e Eficiência

Execuções Paralelas

