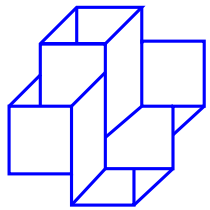
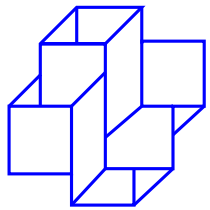
Paralelização Eficiente para o Algoritmo Binário de Exponenciação Modular

Pedro Carlos da Silva Lara
Fábio Borges de Oliveira
Renato Portugal

Laboratório Nacional de Computação Científica

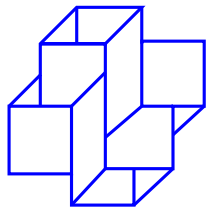


Índice



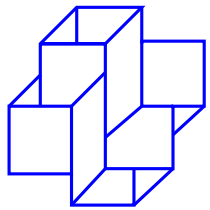
Introdução

- Neste trabalho é considerado o problema de calcular g^e mod m, onde $g, e \in \mathbb{Z}_m$.
- A exponenciação modular é utilizada em grande parte dos algoritmos de criptografia assimétrica e testes de primalidade.
- O desempenho do algoritmo de exponenciação modular possui forte impacto no desempenho de muitos criptossistemas assimétricos.



Introdução

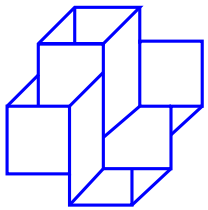
- Foi utilizado o conceito de paralelismo de tarefas para acelerar o calculo de $g^e \mod m$ sendo $g, e \in \mathbb{Z}_m$.
- A técnica proposta está baseada no algoritmo binário de exponenciação modular.



Algoritmo Binário

Também referenciado com *square-and-multiply* o algoritmo binário de exponenciação modular pode ser descrito recursivamente como:

$$g^e = \begin{cases} 1 & \text{se } e = 0 \\ (g^{e/2})^2 & \text{se } e \text{ é par} \\ g^{e-1}g & \text{se } e \text{ é ímpar} \end{cases}$$



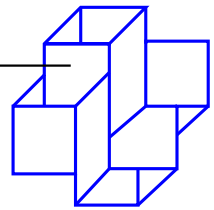
Algoritmo Binário

Algoritmo 1: Algoritmo binário versão esquerda para direita.

Entrada: Inteiro $g \in \mathbb{Z}_m$ e $e = \sum_{i=0}^{k} 2^i b_i$ onde $b_i \in \{0, 1\}$ (representação em base binária).

Saída: $g^e \mod m$.

```
1 início
2     a ← 1;
3     para i = k até 0 faça
4         a ← a^2 mod m;
5         se b_i = 1 então
6             a ← a · g mod m;
7     retorna a;
8 fim
```



Complexidade

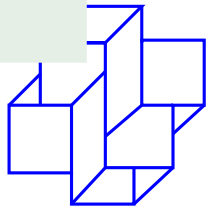
Custo Computacional

O algoritmo anterior, no caso médio, executa $\lfloor \log_2 e \rfloor + 1$ elevações ao quadrado e $\frac{\lfloor \log_2 e \rfloor + 1}{2}$ multiplicações modulares.

Custo Computacional

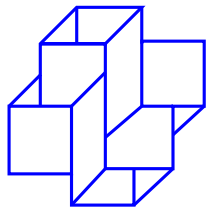
Sendo E o custo para uma elevação ao quadrado, M o custo para uma multiplicação e $j = \lfloor \log_2 e \rfloor + 1$

$$T(j) = jE + \frac{j}{2}M$$



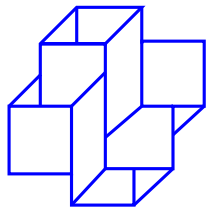
Paralelização do Algoritmo Binário

- $e = (b_k b_{k-1} b_{k-2} \ldots b_{k/2} b_{k/2-1} \ldots b_2 b_1 b_0), b_i \in \{0,1\}, i \in \{0,\ldots,k\}$
- Fazendo $e = 2^r e_1 + e_0$ com $r = k/2$.
- $e = (\overbrace{b_k b_{k-1} b_{k-2} \ldots b_{k/2}}^{e_1} \overbrace{b_{k/2-1} \ldots b_2 b_1 b_0}^{e_0})$
- $e_0 = (b_{k/2-1} \ldots b_1 b_0)$ e $e_1 = (b_k b_{k-1} \ldots b_{k/2})$

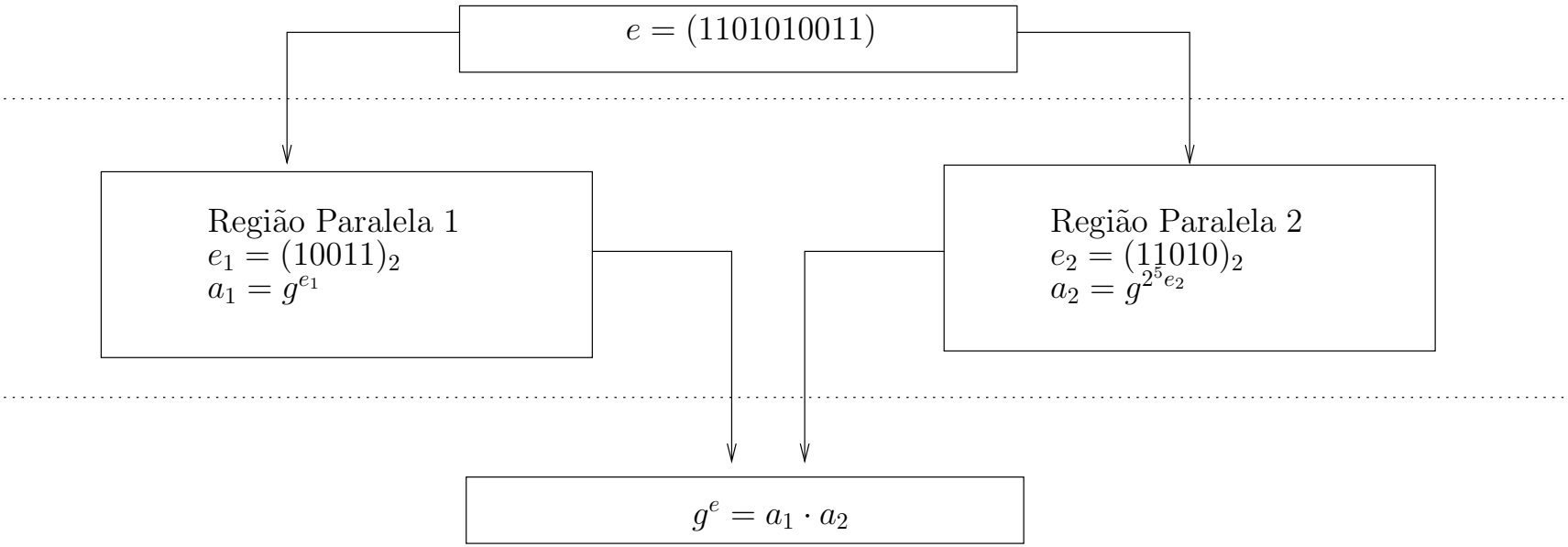


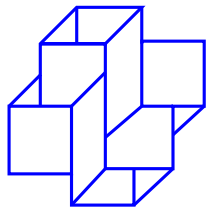
Paralelização do Algoritmo Binário

- Desta forma
$$g^e = g^{2^r e_1 + e_0} = g^{2^r e_1} g^{e_0}.$$
- O cálculo de
$$g^e = g^{2^r e_1} g^{e_0}$$
poderá ser executado em paralelo.
- $\log_2 e_0 \approx \frac{\log_2 e}{2}$ e $\log_2 e_1 \approx \frac{\log_2 e}{2}$



Paralelização do Algoritmo Binário





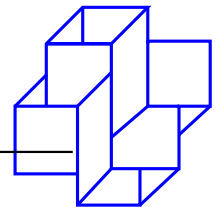
Paralelização do Algoritmo Binário

Algoritmo 2: Paralelização do algoritmo binário.

Entrada: Inteiro $g \in \mathbb{Z}_m$ e $e = \sum_{i=0}^{k} 2^i b_i$ onde $b_i \in \{0, 1\}$ (representação em base binária).

Saída: $g^e \mod m$.

1 **início**
2 $\quad e_0 \leftarrow (b_{r+1} \ldots b_1 b_0)_2$;
3 $\quad e_1 \leftarrow (b_k \ldots b_{r-1} b_r)_2$;
4 $\quad$ **início**
5 $\quad\quad$ **[Região Paralela 1]**
6 $\quad\quad a_0 \leftarrow g^{e_0} \mod m$;
7 $\quad$ **fim**
8 $\quad$ **início**
9 $\quad\quad$ **[Região Paralela 2]**
10 $\quad\quad a_1 \leftarrow g^{2^r}$;
11 $\quad\quad a_1 \leftarrow a_1^{e_1} \mod m$;
12 $\quad$ **fim**
13 $\quad$ **retorna** $a_0 \cdot a_1 \mod m$;
14 **fim**



Generalização

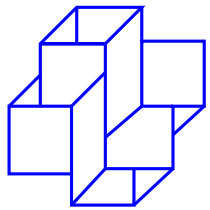
- Poderiamos fazer

$$e = 2^{r_n} e_n + 2^{r_{n-1}} e_{n-1} + \ldots + 2^{r_1} e_1 + e_0.$$

- Com $\log_2 e_i = \frac{\log_2 e}{n+1}$ e $r_i = \frac{i \log e}{n+1}$ para $i \in \{0, \ldots, n\}$.
- Logo

$$g^e = g^{2^{r_n} e_n + 2^{r_{n-1}} e_{n-1} + \ldots + 2^{r_1} e_1 + e_0}$$

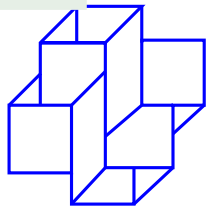
$$g^e = (g^{e_n})^{2^{r_n}} (g^{e_{n-1}})^{2^{r_{n-1}}} \ldots g^{e_0}.$$



Generalização (Exemplo)

Exemplo

- $e = (10100111111010011001101001001001)$
- $j = \lfloor \log_2 e \rfloor + 1 = 32$
- $e = (\underbrace{10100110}_{e_3}\underbrace{11101001}_{e_2}\underbrace{10011010}_{e_1}\underbrace{01001001}_{e_0})$
- $e = (10100111^{r_3}11101001^{r_2}10011010^{r_1}01001001)$
- $r_1 = 8, r_2 = 16, r_3 = 24$
- $e = 2^{r_3}e_3 + 2^{r_2}e_2 + 2^{r_1}e_1 + e_0$

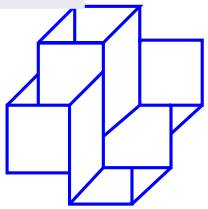


Custo Computacional

- Para duas linhas o custo computacional é determinado pelo cálculo da expressão $(g^{e_1})^{2^r}$ sendo $r = \frac{\lfloor \log_2 e \rfloor + 1}{2}$.
- Como $\log_2 e_1 \approx \frac{\log_2 e}{2}$ e fazendo $j = \lfloor \log_2 e \rfloor + 1$, para calcular $(g^{e_1})^{2^r}$ é necessário:

Complexidade

$$T_2(j) = jE + \left(\frac{j}{4} + 1\right) M$$

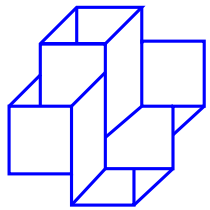


Custo Computacional (Caso Geral)

- Neste caso a complexidade fica determinada pelo termo onde r_i é máximo, ou seja, $(g^{e_k})^{2^{rn}}$.
- Como $\log_2 e_i = \frac{\log_2 e}{n+1}$, $i \in \{0, \ldots, n\}$ e $r_n = \frac{n \log_2 e}{n+1}$.

Complexidade

$$T_n(j) = jE + \left(\frac{j}{2n} + n - 1\right) M$$



Análise de Complexidade

Comparação

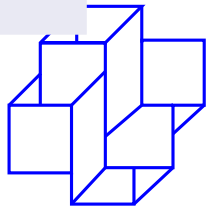
- Sequencial

$$T(j) = jE + \frac{j}{2}M$$

- Paralelo, usando n linhas paralelas

$$T_n(j) = jE + \left(\frac{j}{2n} + n - 1\right)M$$

$$j = \lfloor \log_2 e \rfloor + 1$$



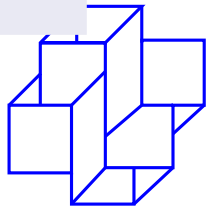
Análise de Complexidade

Considerações

- Não houve redução no número de elevações ao quadrado.
- É necessário estudar o comportamento da função

$$\gamma(n, j) = \frac{j}{2n} + n - 1$$

  a medida que n cresce.
- Esta função está associada ao número de multiplicações modulares, no caso médio.



Análise de Complexidade

- Considerando j constante, a fim de minimizar o número de multiplicações, considere a derivada parcial

$$\frac{\partial \gamma}{\partial n} = -\frac{j}{2n^2} + 1 \qquad (1)$$

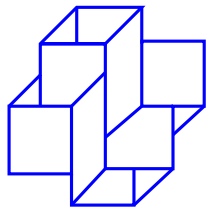
- Igualando (1) a zero e resolvendo na variável n, temos a seguinte relação entre o número de linhas paralelas n e o tamanho do expoente j.

$$n = \sqrt{\frac{j}{2}} \qquad (2)$$

- Como

$$\frac{\partial^2 \gamma}{\partial n^2} = \frac{j}{n^3} > 0$$

a expressão (2) é um mínimo global.



Acelerando o Algoritmo

Algoritmo 3: Paralelização da multiplicação final.

Entrada: Inteiros $a_1, a_2, \ldots, a_n \in \mathbb{Z}_p$

Saída: O produtório $\prod_{i=1}^{n} a_i \in \mod p$.

início

  enquanto $n \neq 1$ **faça**

    para *cada processador* $i = 1$ **até** $n/2$ **faça em paralelo**

      $A_i \leftarrow a_{2i-1} \cdot a_{2i} \mod p$;

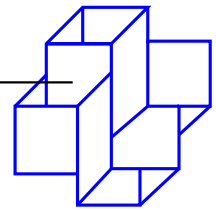
    $n \leftarrow n/2$;

    para $i = 1$ **até** n **faça**

      $a_i \leftarrow A_i$;

  retorna A_1;

fim



Análise de Complexidade

- Este algoritmo computa $\gamma = \lceil \log_2 n \rceil$ multiplicações. Assim o custo computacional total fica

$$T(j) = (j+1)E + \left(\frac{j+1}{2n} + \lceil \log_2 n \rceil \right) M.$$

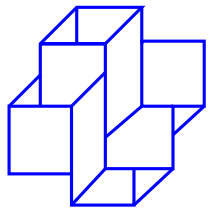
- Neste caso, o número de multiplicações modulares é

$$\mu(j, n) = \frac{j+1}{2n} + \lceil \log_2 n \rceil.$$

- Desta forma, resolvendo

$$\frac{\partial \mu(j, n)}{\partial n} = 0$$

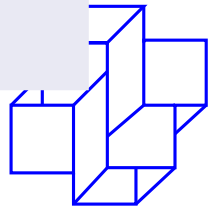
- $n = \frac{\ln 2}{2}(j+1)$.



Resultados Experimentais

Recursos Utilizados

- Sistema operacional Open SuSE Linux 8.14.2 kernel 2.6.25.18 x86/64.
- O *hardware* utilizado foram 8 nós Sun Blade x6250.
- Cada nó com 2 processadores Intel Xeon E5440 Quad Core 3GHz e 16GB de memória física interligados por um barramento *InfiniBand*.
- GMP (GNU Multiple Precision), aritmética de precisão múltipla.
- OpenMPI, paralelismo usando memória compartilhada.



Resultados Experimentais (512 bits)

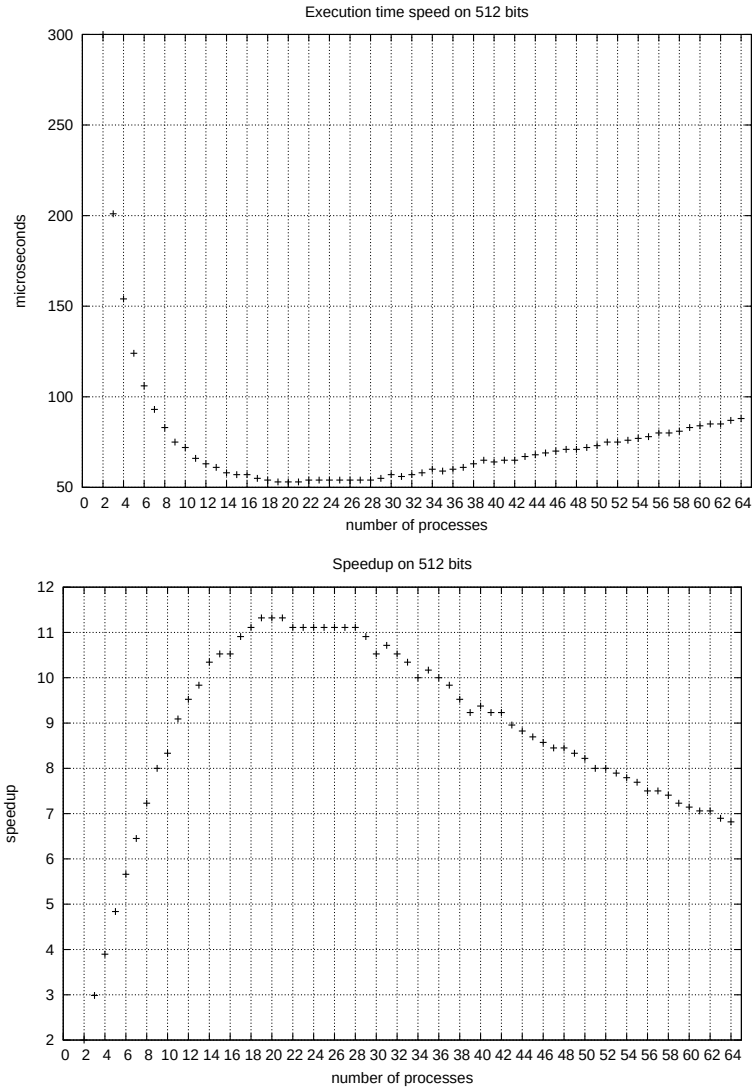


Figura: Tempo de execução e *speedup* para até 64 unsado 512 bits

Resultados Experimentais (1024 bits)

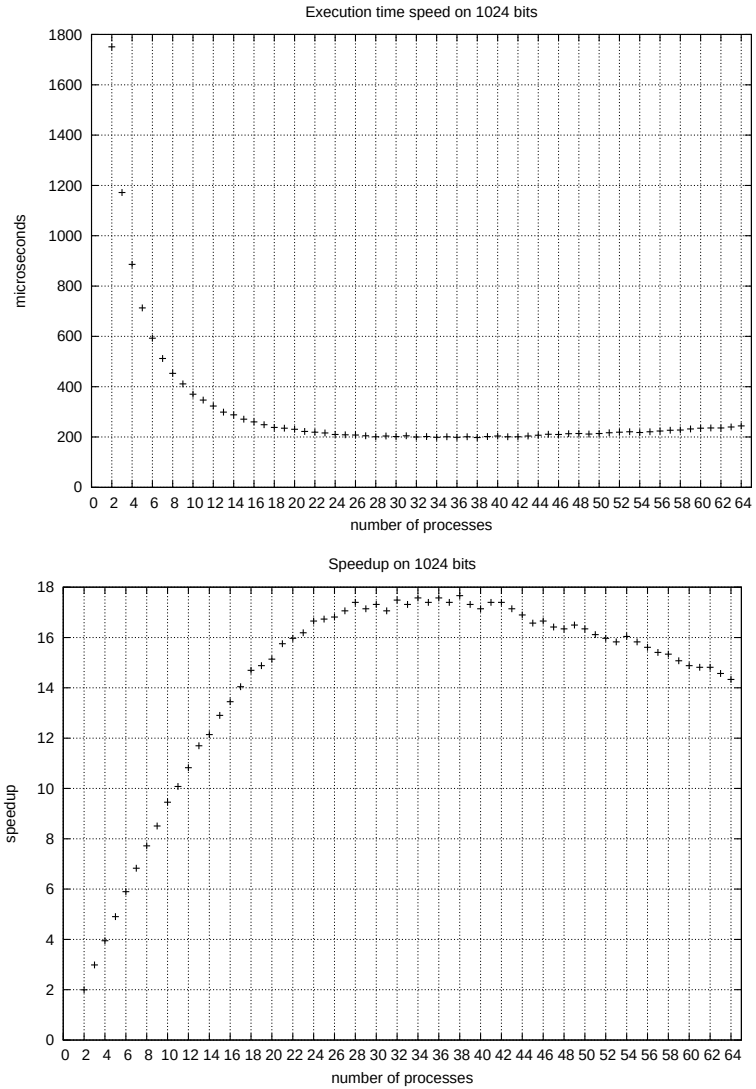


Figura: Tempo de execução e *speedup* para até 64 unsado 1024 bits

Resultados Experimentais (2048 bits)

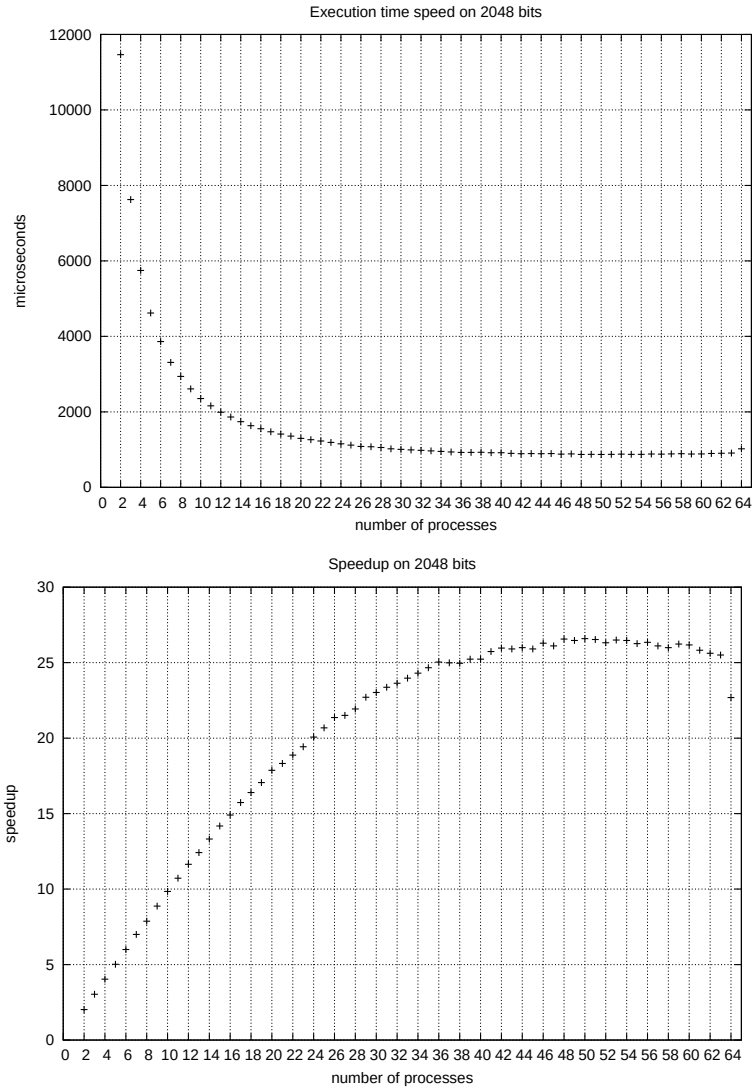


Figura: Tempo de execução e *speedup* para até 64 unsado 2048 bits

Resultados Experimentais (Comparação)

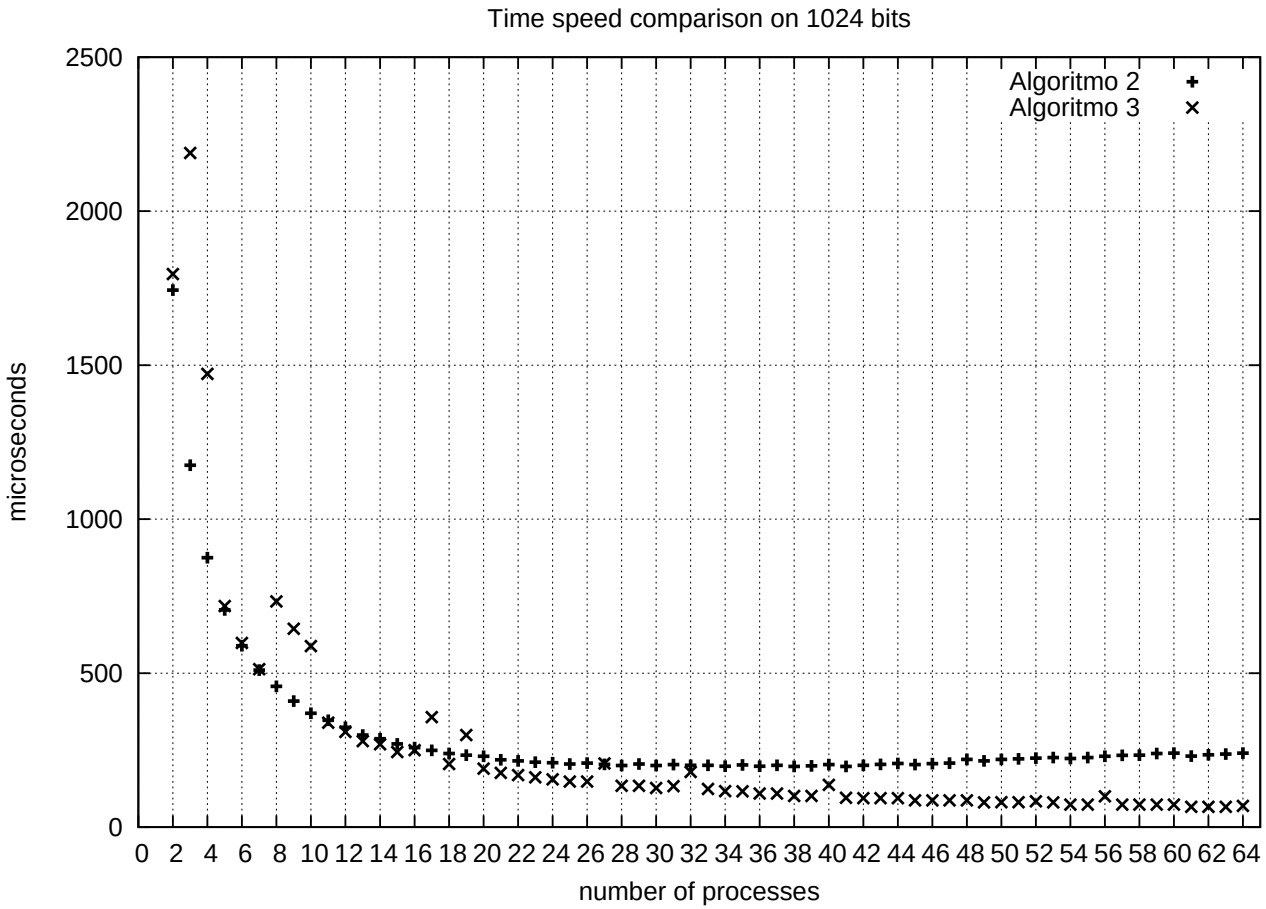
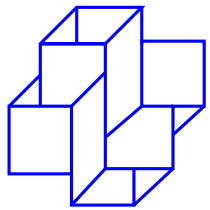


Figura: Comparativo entre os algoritmo de paralelização apresentados.

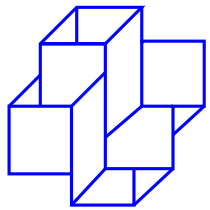
Trabalhos Correlatos

- Grande parte dos autores posicionam seu trabalho em problemas de base-fixa, *e. g.*, *Diffie-Hellman*.
- A versão *multi-prima* do RSA junto com o Teorema do Resto Chinês possui uma paralelização direta e eficiente.



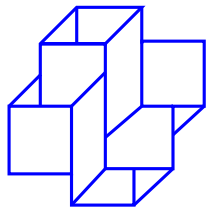
Trabalhos Futuros

- Utilizar GPGPU (General-Purpose computing on Graphics Processing Units).
- Trabalhar com algoritmos mais interessantes do ponto de vista de desempenho, *e. g.*, Janela Deslizante.
- Ajustar o ponto de partição, de forma a balancear a carga entre os processadores.



Conclusões

- Este trabalho descreveu uma técnica de paralelismo para o cálculo da exponenciação modular para o caso geral.
- A paralelização final do algoritmo mostrou-se mais escalável e como menor custo computacional que o algoritmo original.
- Somente foi possível diminuir o número de multiplicações modulares. O número de elevações ao quadrado foi mantido inalterado.



Último Slide

- Obrigado pela atenção!
- Email para contato: borges@lncc.br.
- Perguntas?