# ANÁLISE DE DESEMPENHO DA BIBLIOTECA GMP: UM ESTUDO DE CASO COM A FATORAÇÃO DE NÚMEROS

# ANALYSIS OF THE LIBRARY GMP PERFORMANCE: A CASE STUDY WITH NUMBERS FACTORING

Rafael Ayres Claudino*
Antonio Marcos Neves Esteca**

## RESUMO

Uma das mais completas bibliotecas disponíveis no mercado para o trabalho com aritmética de precisão arbitrária em linguagem C é a GNU Multiple-Precision Library, também conhecida como GMP, a qual é bastante utilizada na área de criptografia e em pesquisas sobre Teoria dos Números. Essa biblioteca permite trabalhar sem limites práticos de precisão, sendo a memória da máquina a única barreira existente. Apesar de seu grande potencial, poucos trabalhos abordam a GMP na literatura. Diante disso, o objetivo desse artigo é apresentar uma análise mais profunda dessa biblioteca, de modo a avaliar seu desempenho, por meio de um estudo de caso com a fatoração de números.

**Palavras-chave:** Números gigantes. Linguagem C. GMP. Análise de desempenho.

## ABSTRACT

One of the most complete libraries available on the market to work with arbitrary precision arithmetic in C is the GNU Multiple Precision-Library, also known as GMP, which is widely used in the encryption area and research on Number Theory. This library allows you to work without practical limits of accuracy, being the memory of the machine the only existing barreria. Despite its great potential, few works approach the GMP in the literature. Thus, the aim of this paper is to present a deeper analysis of this library in order to evaluate their performance through a case study with the factorization of numbers.

**Keywords:** Big numbers. C language. GMP. Performance analysis.

* Bolsista de Iniciação Científica da FATECE-Pirassununga/SP, graduando em Ciência da Computação. rafael-claudino@hotmail.com
** Coordenador do Curso de Ciência da Computação da FATECE-Pirassununga/SP. am.esteca@sjrp.unesp.br

## Introdução

O trabalho com números gigantes (do inglês *big numbers*) é algo de crucial importância em algumas áreas da ciência, como a Criptografia e Teoria dos Números (SILVEIRA; FALEIROS, 2010). Uma das linguagens de programação mais empregadas atualmente, especialmente em projetos científicos, é a linguagem C.

Apesar de seu grande potencial, a linguagem C possui recursos muito limitados para lidar com números gigantes, o que levou ao surgimento de bibliotecas específicas para este fim, sendo a GNU Multiple-Precision Library, conhecida como GMP, a principal delas (SILVEIRA; FALEIROS, 2010).

Sabe-se que a biblioteca GMP tem como vantagem em relação às bibliotecas da própria linguagem C sua capacidade de trabalhar sem limites práticos de precisão, sendo a memória da máquina a única barreira existente (GRANLUND, 2002). No entanto, até o momento não foram realizados estudos sobre o impacto da biblioteca no desempenho dos programas construídos. Neste contexto, este trabalho tem como objetivo aprofundar o conhecimento sobre a biblioteca GMP, bem como verificar suas vantagens e desvantagens em relação às bibliotecas da própria linguagem C, com ênfase na comparação do desempenho dos programas construídos com e sem o uso da biblioteca.

Para atender ao objetivo da pesquisa, como estudo de caso, foram construídas duas versões de um programa que realiza a fatoração de números retornando como resposta os fatores primos e seus expoentes. Uma das versões foi escrita utilizando-se apenas os recursos e bibliotecas da própria linguagem C, enquanto a outra empregou exatamente o mesmo algoritmo, mas agora utilizando a biblioteca GMP. A definição desse estudo de caso justifica-se pelo fato de que o algoritmo empregado demanda uma elevada carga de processamento, permitindo exercitar os programas construídos por mais tempo.

O restante desse artigo é organizado como segue: na seção 2 é apresentada uma visão geral da biblioteca GMP, abordando sua integração ao compilador de linguagem C utilizado no trabalho; em seguida, na seção 3, é apresentada uma visão geral sobre os números primos, de modo a contextualizar o estudo de caso abordado; na seção 4 são apresentadas as duas versões construídas do programa de fatoração em números primos; na seção 5 são revelados os resultados obtidos a partir da análise comparativa dos dois programas; por fim, na seção 6 são apresentadas as considerações finais e propostas para trabalhos futuros.

## 1 Biblioteca GMP

GMP é uma biblioteca livre para aritmética de precisão arbitrária, operando com números inteiros, racionais e de ponto flutuante (SNIR et al., 1996). Não há limite prático para a precisão, exceto os implicados pela memória disponível na máquina. Essa biblioteca possui um rico conjunto de funções com uma interface regular. A GMP é aplicada principalmente em aplicações de criptografia, em pesquisas científicas, em aplicações de segurança na Internet, em sistemas de álgebra e em álgebra computacional (SANCHES; NOGUEIRA, 2011). O projeto da GMP foi desenvolvido cuidadosamente para que ela seja o mais rápido possível, tanto para pequenas como para grandes operações, de modo que suas instruções são altamente otimizadas.

O lançamento da primeira versão da GMP foi feito em 1991, mas seu desenvolvimento é contínuo, sendo implementadas melhorias a cada ano. Desde a versão 6, GMP é distribuída como uma biblioteca livre para uso, compartilhamento e aprimoramento.

### 1.1 Categorias funcionais da GMP

Na biblioteca GMP existem cinco categorias de funções, a saber:

- Funções aritméticas inteiras de alto nível de precisão (MPZ) - Existem cerca de 150 funções aritméticas e lógicas nesta categoria;
- Funções aritméticas racionais de alto nível de precisão (MPQ) - É composta por cerca de 35 funções;
- Funções aritméticas de ponto flutuante de alto nível de precisão (MPF) - Esta é a categoria funcional GMP a ser usada se o tipo “double” em C não oferecer a precisão suficiente para uma aplicação. Há cerca de 70 funções nesta categoria;
- Inteiros positivos de baixo nível (MPN) - Esta categoria contém funções de alto desempenho, pois nenhum gerenciamento de memória é empregado. Entretanto, o uso de suas funções é mais difícil, uma vez que nem o conjunto de funções é regular, nem a interface de chamadas.
- Classe em C++ para uso de todas as funções acima.

Dentre as funções apresentadas acima, foram utilizadas nessa pesquisa apenas as da categoria MPZ, pois serão processados somente números inteiros. Em todo o código-

fonte escrito por essa categoria é necessário colocar a palavra “mpz” antes de todas as funções do algoritmo.

É importante observar que nem todo compilador aceita a biblioteca GMP. Neste trabalho foi utilizado o compilador Code Blocks 13.12 em ambiente Windows. A seguir, são descritos os passos para a integração da GMP ao compilador:

1° passo) efetuar o download da biblioteca GMP em http://cs.nyu.edu/exact/core/gmp/gmp-static mingw-4.1.tar.gz;
2° passo) extrair o conteúdo do arquivo gmp-static-mingw-4.1;
3° passo) criar uma pasta chamada com o nome "gmp" na raiz do disco rígido onde está instalado o sistema operacional. Dentro dessa pasta, inserir o conteúdo da pasta descompactada no passo anterior;
4° passo) criar um novo projeto no Code Blocks. Esse projeto deve conter um arquivo chamado main.c;
5° passo) configurar as opções de compilação. Para tanto, basta clicar com o botão direito do *mouse* sobre o arquivo main.c e, em seguida, clicar em "Build Options". Feito isso, realize as seguintes configurações:

- na aba "Linker settings", vá em "Link libraries", clique em "Add" e selecione o arquivo libgmp.a na pasta "lib", interna à pasta "gmp" criada no passo 3;
- na aba "Search directories", vá em "Compiler", clique em "Add" e selecione a pasta "include", interna à pasta "gmp" criada no passo 3.
- ainda na aba "Linker settings", vá em "Linker", clique em "Add" e selecione a pasta "lib", interna à pasta "gmp" criada no passo 3.

## 2 Números Primos

Os números primos levaram a vários resultados importantes apesar de sua simples definição: “Um número natural $p \neq 1$ é *primo* se os seus únicos divisores são 1 e $p$”. Um número natural que não é primo é chamado de número *composto*. A unidade 1 não é considerada nem primo e nem composto. Um número primário é aquele que não pode ser gerado, através da operação de multiplicação, por outro número primário.

Desde a remota época de **Pitágoras** (Samos, 570 a.C. - Metaponto, 497 a.C.), passando pelos mais renomados matemáticos da história e ainda nos dias atuais, com o desenvolvimento da computação algébrica e das técnicas de criptografia, a pesquisa

sobre propriedades e aplicações dos primos sempre esteve em um patamar de destaque (BURTON, 1989).

A escolha da fatoração de números em seus elementos primos como o estudo de caso dessa pesquisa deve-se à alta complexidade envolvida no processo e, consequentemente, à elevada carga de processamento demandada, gerando com isso a possibilidade de analisar o comportamento da biblioteca GMP.

## 3 Códigos e Métodos

Para realizar a análise da biblioteca GMP foi desenvolvido um algoritmo simples para a decomposição de números em seus fatores primos. É importante observar que o método utilizado não emprega as técnicas mais otimizadas disponíveis na literatura, uma vez que o objetivo do trabalho é gerar elevada carga de processamento.

Após a elaboração do algoritmo foi realizada sua implementação em duas versões. Na primeira delas (Versão A), o código-fonte é construído com base apenas nas bibliotecas originalmente presentes na linguagem C, utilizando sintaxe padrão, sendo possível manipular até o número inteiro 18446744073709551615, dada a limitação da linguagem. Já na segunda versão (Versão B), o código-fonte é completamente escrito com base na sintaxe da biblioteca GMP, não havendo limite para o número inteiro a ser tratado. Cabe salientar que os programas seguem exatamente a mesma lógica, sem nenhuma diferença.

A seguir são apresentadas as duas versões implementadas do código-fonte.

- **Versão A:**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <windows.h>

int main(int argc, char *argv[])
{
   unsigned long long int x,n,i,d,t,u,f,z,a,b,e,r,o;
   int y,c,inicio, fim, tmili;
   c=0;
   u=1;
   t=1;
   y=1;
   f=1;
  while(c==0){
      printf("DIGITE UM NUMERO PARA SABER SEUS FATORES PRIMOS\n");
      scanf("%I64d",&n);// recebe o número para a decomposição
      b=sqrt(n);//extrai a raiz quadrada do numero na variável B
      inicio = GetTickCount();
      for(x=2;x<=b;x++){//loop de repetição
```

```
        if(n%x==0){ //comparação para saber quais os números que são divisores de N
          if(x==2){ //se for divisível por 2 ele faz um processo especial.
            d=x*x; //coloca na variável D o quadrado de 2
            for(y=1;n%d==0;y++){//loop de repetição para saber qual o grau do expoente da base 2
                d=d*x;//cada vez que executa o for acima, aumenta uma potência de 2 na variável D
            }
           printf("\n1---o numero primo divisor e: %I64d elevado a: %d\n\n",x,y);//imprimindo a potência de 2 que
divide o número
           t=pow(x,y);//coloco na variável t a potência de 2
           if(t==n)// se o número for um quadrado perfeito ele encerra a busca
               {
                 x=b+1;
               }
           f=n/t;
           e=sqrt(f);
           for(i=2;i<=e;i++){//loop de repetição para verificar se a variável f é primo
               if(f%i==0){
                 i=e+1;
               }
               if(i==e){
                 printf("\n2---o numero primo divisor e:%I64d elevado a:%d\n\n",f,1);//se f for primo ele imprime
na tela
                 u=f*t;
                 if(u==n)//se a variável u for igual ao número digitado encerra a busca
                    {
                    x=b+1;
                    }
               }
           }
             u=1;
             f=1;
        }
        else{
             for(i=2;i<x;i++){//loop de repetição para verificar se a variável x divisor de n é primo
                 if(x%i==0){//se x não for um primo então para o loop
                      i=x;
                 }
                 if(i==x-1){ //se for primo, entra nesse if
                   d=x*x;//coloco na variavel D o quadrado de x
                   for(y=1;n%d==0;y++){//loop de repetição para saber qual o grau do expoente da base x
                        d=d*x;//cada vez que executa o for acima, aumenta uma potência de x na variável D
                   }
                   printf("\n3---o numero primo divisor e:%I64d elevado a:%d\n\n",x,y);//imprimindo a potência de
base x e expoente y
                   for(a=1;a<=y;a++)
                          {
                            u=u*x;
                          }
                    f=t*u;  //coloca na variável f a multiplicação dos números encontrados que são divisores de n
                    if(u==n)
                       {
                         x=b+1;
                          i=x;
                       }
                 }
             }
             if(f!=n)// se esse número não for igual a n, executa essa condição
               {
                  a=n/f;//coloca na variável a o outro numero que é divisor de n
                  e=sqrt(a);//coloca em e a raiz quadrada desse número
                  for(i=2;i<=e;i++){//loop de repetição para verificar se a variável a é primo
                       if(a%i==0){//se não for primo acaba a busca
                           i=e+1;
                       }
                      if(i==e){//se for primo entra nessa condição
                          printf("\n4---o segundo primo divisor e:%I64d elevado a:%d\n\n",a,1);//imprimindo o
numero primo a elevado ao expoente 1
```

```
                        x=b+1;//encerra a busca
                    }
                }
            }
        }
    }
    if(f==n){//se f for igual a n achou todos os primos divisores de n
        printf("acabo a variavel f e:%I64d\n\n",f);//imprime f
        x=b+1;//encerra a busca
    }
    if(x==b){//se o número digitado para decompor em fatores primos for um número primo, então ele entra
nessa condição
        printf("O NUMERO E PRIMO\n\n");
    }
  }
        fim = GetTickCount();
        tmili = fim - inicio;
        printf("TEMPO DECORRIDO EM MILISSEGUNDOS: %d\n\n\n\n", tmili);
        u=1;
        t=1;
        f=1;
        a=1;
        e=1;
       }
  c=0;
  system("PAUSE");
  return 0;
}
```

- **Versão B:**

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include <windows.h>

intmain(void)
{
mpz_tx,n,i,d,t,u,f,z,a,b,e,r,y,c,m, resp, aux, aux2,inicio,fim, tmili;//criando variáveis para armazenar dados
mpz_init(x);//inicializando as variáveis
mpz_init(n);
mpz_init(i);
mpz_init(d);
mpz_init(t);
mpz_init(u);
mpz_init(f);
mpz_init(z);
mpz_init(a);
mpz_init(b);
mpz_init(e);
mpz_init(m);
mpz_init(r);
mpz_init(y);
mpz_init(c);
mpz_init(resp);
mpz_init(aux);
mpz_init(aux2);
mpz_init(inicio);
mpz_init(fim);
mpz_init(tmili);
mpz_set_ui(c,0);//atribuindo valores às variáveis
mpz_set_ui(u,1);
mpz_set_ui(t,1);
mpz_set_ui(y,1);
mpz_set_ui(f,1);
```

```
mpz_set_ui(resp,1);
mpz_set_ui(m,0);

while(mpz_sgn(c) == 0){//função de repetição
printf("DIGITE UM NUMERO PARA SABER SEUS FATORES PRIMOS\n"); //imprime na tela
gmp_scanf("%Zd", n);//recebe os dados digitados
mpz_sqrt(b,n);//calcula a raiz quadrada do valor digitado e armazena na variável b

mpz_set_ui(inicio,GetTickCount());//registrar tempo inicial
for(mpz_set_ui(x,2);mpz_cmp(x,b)<=0;mpz_add_ui(x, x, 1)){ //loop de repetição
    mpz_mod(m,n,x);//a variável m recebe o resto da divisão de n por x
    if (mpz_sgn(m)==0){//verifica se a variável m é igual a zero
      if(mpz_cmp_ui(x,2)==0){//compara a variável x com 2
        mpz_mul(d, x, x);//multiplica x por x e armazena na variável d
        mpz_mod(m,n,d);//a variável m recebe o resto da divisão de n por d
        for(mpz_set_ui(y,1);mpz_sgn(m)==0;mpz_add_ui(y,y,1)){//loop de repetição
             mpz_mul(d, d, x);//multiplica x por d e armazena na variável d
             mpz_mod(m,n,d);//a variável m recebe o resto da divisão de n por d
        }
        gmp_printf("1---o numero primo divisor e: %Zd elevado a: %Zd\n\n",x,y);//imprime na tela
        for(mpz_set_ui(aux,1);mpz_cmp(aux,y)<=0;mpz_add_ui(aux, aux, 1)){//loop de repetição
              mpz_mul(resp, resp, x);//multiplica x por resp e armazena na variavelresp
        }
        mpz_set(t,resp);//armazena na variável t o conteúdo da variavelresp
        mpz_set_ui(resp,1);

        if(mpz_cmp(t,n)==0)//compara a variável t com n
           {
            mpz_add_ui(x,b,1);//armazena na variável x, a variável b+1
           }
           mpz_cdiv_q(f,n,t);// variável f recebe n dividido por t
           mpz_sqrt(e,f);// variável e recebe a raiz quadrada de f

        for(mpz_set_ui(i,2);mpz_cmp(i,e)<=0;mpz_add_ui(i,i, 1)){//loop de repetição
             mpz_mod(m,f,i);//a variável m recebe o resto da divisão de f por i
             if(mpz_sgn(m)==0){//verifica se a variável m é igual a zero
                 mpz_add_ui(i,e,1);//armazena na variável i a variável e +1
             }
             if(mpz_cmp(i,e)==0){//compara i com e
                 gmp_printf("\n2---o numero primo divisor e:%Zd elevado a:%d\n\n",f,1);//imprime na tela
                 mpz_mul(u, f, t);//multiplica t por f e armazena na variável u
                 if(mpz_cmp(u,n)==0)//compara u com n
                   {
                    mpz_add_ui(x,b,1);//armazena na variável x, a variável b+1
                    }
             }
           }
         }

else{

    for(mpz_set_ui(i,2);mpz_cmp(i,x)<0;mpz_add_ui(i,i,1)){//loop de repetição
      mpz_mod(m,x,i);//a variável m recebe o resto da divisão de x por i
      if(mpz_sgn(m)==0){//verifica se a variável m é igual a zero
        mpz_set(i,x);//armazena na variável i o conteúdo da variável x
      }
      mpz_sub_ui(aux2,x, 1);//armazena na variável aux2 a subtração de x por 1
      if(mpz_cmp(i,aux2)==0){//compara a variável i com a variável aux2
        mpz_mul(d, x, x);//multiplica x por x e armazena na variável d
        mpz_mod(m,n,d);//a variável m recebe o resto da divisão de n por d
        for(mpz_set_ui(y,1);mpz_sgn(m)==0;mpz_add_ui(y,y,1)){//loop de repetição
          mpz_mul(d, d, x);//multiplica x por d e armazena na variável d
          mpz_mod(m,n,d);//a variável m recebe o resto da divisão de n por d
        }
       gmp_printf("\n3---o numero primo divisor e:%Zd elevado a:%Zd\n\n",x,y);//imprime na tela
       for(mpz_set_ui(aux,1);mpz_cmp(aux,y)<=0;mpz_add_ui(aux, aux, 1)){//loop de repetição
          mpz_mul(resp, resp, x);//multiplica x por resp e armazena na variavelresp
```

```
            }
       mpz_mul(u, resp, u);//multiplica u por resp e armazena na variável u
       mpz_set_ui(resp,1);//armazena 1 na variavelresp
       mpz_mul(f, t, u);//multiplica u por t e armazena na variável f
       if(mpz_cmp(u,n)==0)//compara a variável u com a variável n
            {
                 mpz_add_ui(x,b,1);//armazena na variável x, a variável b+1
                 mpz_set_ui(i,x);//armazena o conteúdo da variável x na variável i
            }
       }
      }
       if(mpz_cmp(f,n)!=0)//verifica se a variável f é diferente da variável n
       {
            mpz_cdiv_q(a,n,f);//a variável a recebe a divisão da variável n pela f
            mpz_sqrt(e,a);// a variável e recebe a raiz quadrada da variável a

            for(mpz_set_ui(i,2);mpz_cmp(i,e)<=0;mpz_add_ui(i,i,1)){//loop de repetição
                 mpz_mod(m,a,i);//a variável m recebe o resto da divisão de a por i
                 if(mpz_sgn(m)==0){//compara m com 0
                      mpz_add_ui(i,e,1);}//armazena na variável i o valor de e+1
                 if(mpz_cmp(i,e)==0){//compara a variável i com a variável e
                      gmp_printf("\n4---o segundo primo divisor e:%Zd elevado a:%d\n\n",a,1);//imprime na tela
                      mpz_add_ui(x,b,1);//armazena na variável x o valor de b+1
                 }
            }
        }
}
}
if((mpz_cmp(f,n)==0)){
     gmp_printf("acabo a variavel f e:%Zd\n\n",f);//imprime na tela
     mpz_add_ui(x,b,1);//armazena na variável x o valor de b+1
     }
if((mpz_cmp(x,b)==0)){//compara a variável x com a variável b
          printf("O NUMERO E PRIMO\n\n");//imprime na tela
     }
}
mpz_set_ui(fim,GetTickCount());
mpz_sub(tmili,fim,inicio);
gmp_printf("TEMPO DECORRIDO EM MILISSEGUNDOS: %Zd\n\n\n\n", tmili);
 //registrar tempo final
mpz_set_ui(u,1);
mpz_set_ui(t,1);
mpz_set_ui(e,1);
mpz_set_ui(f,1);
mpz_set_ui(a,1);
}
mpz_set_ui(c,0);
return EXIT_SUCCESS;
}
```

## 4 Resultados

Após a implementação, as duas versões do algoritmo foram submetidas a medições de tempo de execução mediante a entrada dos mesmos valores, ou seja, as versões A e B do programa fatoraram os mesmos números, sendo aferido, em milissegundos, o tempo necessário a cada uma, conforme apresentado na Tabela 1, que aponta também a razão do tempo levado pela versão B em relação à versão A do

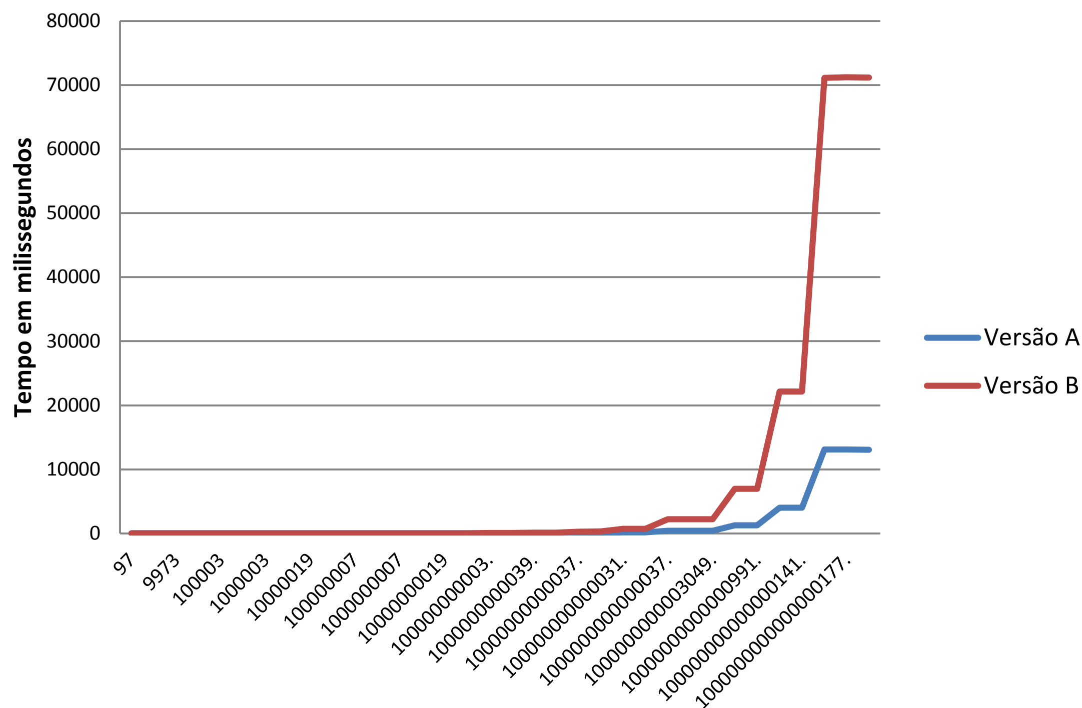
programa. Observa-se que todos os testes foram executados sobre a mesma plataforma computacional, empregando também os mesmos recursos de hardware.

Tabela 1 - Comparativo do tempo de fatoração (Versão A x Versão B)

| | **Tempo em milissegundos** | | |
|---|---|---|---|
| **Número a ser fatorado** | **Versão A** | **Versão B** | **B/A** |
| 97 | 0 | 0 | - |
| 997 | 0 | 0 | - |
| 9973 | 0 | 0 | - |
| 99991 | 0 | 0 | - |
| 100003 | 0 | 0 | - |
| 121067 | 0 | 0 | - |
| 1000003 | 0 | 0 | - |
| 1031003 | 0 | 0 | - |
| 10000019 | 0 | 0 | - |
| 10042327 | 0 | 0 | - |
| 100000007 | 0 | 0 | - |
| 100037293 | 0 | 0 | - |
| 1000000007 | 0 | 0 | - |
| 1000027013 | 0 | 16 | - |
| 10000000019 | 0 | 16 | - |
| 10000021363 | 0 | 16 | - |
| 100000000003 | 16 | 46 | 2,9 |
| 100000010549 | 16 | 47 | 2,9 |
| 1000000000039 | 32 | 93 | 2,9 |
| 1000000008673 | 32 | 124 | 3,9 |
| 10000000000037 | 46 | 265 | 5,8 |
| 10000000003549 | 47 | 296 | 6,3 |
| 100000000000031 | 141 | 733 | 5,2 |
| 100000000005629 | 141 | 733 | 5,2 |
| 1000000000000037 | 421 | 2215 | 5,3 |
| 1000000000002551 | 421 | 2215 | 5,3 |
| 1000000000003049 | 421 | 2230 | 5,3 |
| 10000000000000061 | 1263 | 6973 | 5,5 |
| 10000000000000991 | 1264 | 6958 | 5,5 |
| 100000000000000003 | 3994 | 22136 | 5,5 |
| 100000000000000141 | 3994 | 22152 | 5,5 |
| 1000000000000000003 | 13104 | 71120 | 5,4 |
| 1000000000000000177 | 13105 | 71214 | 5,4 |
| 1000000000000000201 | 13073 | 71167 | 5,4 |

Por meio do gráfico 1 é possível notar com clareza o comportamento das duas versões do algoritmo. Apesar da versão B do programa ter a vantagem de poder trabalhar com números muito maiores que a versão A, o seu desempenho começa a cair bruscamente com o aumento dos números. Observando-se a Tabela 1, constata-se que até os maiores números suportados pela versão A do programa, a versão B estava levando aproximadamente 5,5 vezes o tempo necessário pela versão A.

Gráfico 1 - Comparativo do tempo de fatoração (Versão A x Versão B).




A partir dos resultados mensurados é possível afirmar que o emprego da biblioteca GMP eleva sobremaneira a capacidade de processamento, mas afeta consideravelmente o desempenho do programa. Com isso, o uso dessa biblioteca deve ser feito somente para tarefas que demandem o processamento de números não suportados pela versão escrita com as bibliotecas da própria linguagem C.

## Considerações Finais e Trabalhos Futuros

Esse trabalho realizou uma análise da biblioteca GMP, uma das mais poderosas ferramentas para trabalho com números gigantes disponíveis no mercado. Para tanto, foi definido um estudo de caso com a fatoração de números primos, a partir do qual foram construídas duas versões de um mesmo algoritmo de fatoração. Uma das versões

construídas empregou apenas os recursos padrão da linguagem C, enquanto a outra incorporou a biblioteca GMP. Após a implementação, ambos os programas foram submetidos a testes que permitiram aferir o desempenho de cada um.

A análise dos resultados permite constatar que a principal vantagem da biblioteca GMP é ampliar sobremaneira o tamanho dos números inteiros processáveis. Por outro lado, verificou-se que essa biblioteca degrada muito o desempenho do programa. Desse modo, pode-se concluir que a biblioteca deve ser empregada apenas quando o problema lidar com números maiores que a solução em C puro pode atender.

Por fim, sugere-se que trabalhos futuros avaliem a biblioteca GMP em relação a outros aspectos, em especial aos propostos por Sebesta (2011), sendo eles: Legibilidade, Capacidade de Escrita e Confiabilidade. A partir desse estudo, melhorias poderiam ser apontadas de modo a aprimorar a qualidade dos recursos oferecidos pela biblioteca GMP.

## Referências

BURTON, D. **Elementary Number Theory.** University of New Hampshire, Wm C. Brown, Durham, 1989.

GRANLUND, T. **The GNU MP. The Multiple Precision Arithmetic Library**. Edição 4.1.2, Free Software Foundation, 2002.

SANCHES, A. P.; NOGUEIRA, K. L. **Um estudo comparativo dos Métodos QS e MPQS**. 2011. 65 p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual de Mato Grosso do Sul, Dourados, 2011.

SEBESTA, R. W. **Conceitos de Linguagem de Programação**. 9. ed. São Paulo: Bookman, 2011.

SILVEIRA, A. S.; FALEIROS, A. C. **Criptografia de chave pública. O papel da Aritmética em precisão múltipla**. São José dos Campos: Instituto Tecnológico de Aeronáutica (ITA), 2010. Relatório de Iniciação Científica. 6 páginas.

SNIR, M. et al. **MPI:** The Complete Reference. Cambridge: The MIT Press, 1996.