Experimentos de Cálculo com Precisão Numérica Arbitrária

Experiments in Calculation with Arbitrary Numerical Precision

Nathan Machado dos Santos[1], José Lucas Carvalho Silva[1], Rogério Azevedo Rocha[1], Tanilson Dias dos Santos[1] e Mirray Victor Lima Oliveira[1]

[1] *Universidade Federal do Tocantins*

Data de recebimento do manuscrito: 30/08/2023
Data de aceitação do manuscrito: 25/09/2023
Data de publicação: 16/10/2023

Resumo—No projeto de linguagens de programação muitas vezes o tamanho máximo alocado para alguns tipos de variáveis é limitado. Isso prejudica alguns tipos de aplicação que carecem de uma maior acurácia quanto a representação de números com uma quantidade de dígitos além da capacidade de representação dessa linguagem. Nesta pesquisa investigamos o problema de representação numérica e para contornar este problema nos valemos de artifícios computacionais como o uso de bibliotecas de precisão arbitrária. Este artigo apresenta os resultados preliminares desta pesquisa para cálculo dos valores de constantes como Fibonacci e π, onde obteve-se um sucesso parcial, além de uma possível solução para os problemas encontrados.

Palavras-chave—Imprecisão númerica, problemas matemáticos, números irracionais, números transcendentais, número π, fibonacci

Abstract—*In programming language design projects, there is often a limitation on the maximum size allocated for certain variable types. This constraint poses challenges for applications that require more precise representation of numbers with a higher number of digits than the language's capacity allows. In this research, we delve into the issue of numeric representation and, in order to address this concern, we employ computational techniques such as the utilization of arbitrary precision libraries. This article presents the initial findings of our research in calculating constants like Fibonacci and π, where partial success was achieved, and a potential solution to the encountered issues is proposed.*

Keywords—*Numerical imprecision, mathematical problems, irrational numbers, transcendental numbers, number π, fibonacci*

I. Introdução

Os computadores enxergam e lidam com os números os representando numa base binária, isto é, com bits, unidades que podem ser 0 ou 1. Os números inteiros, mais comumente utilizados, são representados por grupos de bits, onde o valor ocupa a totalidade desse grupo. Números muito extensos, sejam fracionais ou não, como números reais, irracionais, dentre outros, são representados pelo que são chamados de números de pontos flutuantes (ou em alguns casos números de pontos fixos), onde os bits são divididos em 3 partes distintas, o sinal, o expoente e a mantissa.

Os números de ponto flutuante são padronizados e definidos pela IEEE 754 (Instituto de Engenheiros Elétricos e Eletrônicos), que foi estabelecida em 1985, contando com grandes revisões de tempos e tempos, e adotada por todas as grandes empresas produtoras de microchips e seus softwares/compiladores associados. Essa representação nada mais é do que uma expressão de notação científica, onde cada parte dessa expressão possui um número predeterminado de bits, baseado na variável. Nos padrões da IEEE 754, Temos que para float, 1 bit se desgina para o sinal, 8 para o expoente e 23 para a mantissa. Já para double, têm-se 1 bit para o sinal, 11 para o expoente e 52 para a mantissa, segundo [1].

Na definição de linguagens de programação, geralmente, são especificadas, no seu projeto, limites para o tamanho das variáveis utilizadas. Essas especificações são intrísecas ao propósito de cada linguagem e agem como um gargalo em cálculos que exigem grande precisão e/ou muitas casas decimais para representar números com infinitas casas decimais (ou simplesmente com muitas casas), e.g. os valores de e e π.

A Tabela 1 apresenta o tamanho de alguns tipos de variáveis em linguagem de programação C[1], considerando uma arquitetura e compilador de 32 bits, segundo [2] e [3]. Esses valores seguem o padrão da IEEE 754, e o intervalo apresentado demonstra o poder de representação dessa configuração de variável.

Observando a Tabela 1, é possível notar que há uma

Dados de contato: Tanilson Dias dos Santos, tanilson.dias@uft.edu.br

[1]Em linguagem C, os tamanhos das variáveis podem ser distintos dos apresentados, a depender da máquina e do compilador utilizado.

TABELA 1: VARIÁVEIS EM C

Tipo de variável	Tamanho em bits	Intervalo	
signed int	32	-2.147.483.648	2.147.483.647
unsigned int	32	0	4.294.967.295
float	32	3,4E-38	3,4E+38
double	64	1,7E-308	1,7E+308

limitação para o tamanho de uma variável tipo double, que no caso apresentado, para a linguagem C que é o valor $\pm 1.7^{308}$. Quando um valor escapa do limite de uma variável, a IEEE 754 dita alguns tratamentos de exceções que devem ser aplicados nestes casos, a depender das configurações utilizadas, com a exceção mais comum no caso de um overflow se arredondar o resultado para infinito, ou o valor máximo possível.

Além destes limites existentes, outro fator extremamente importante é o fato de que existem erros de precisão relacionados aos números de ponto flutuante. É inerente a este tipo de representação a existência de erros de arredondamento, tendo em vista que a proposta é armazenar números infinitos ou simplesmente muito grandes, em um espaço finito de armazenamento.

Há na literatura registros de diversos trabalhos que investigam ou utilizam bibliotecas de precisão arbitrária em seu escopo, podemos citar por exemplo as pesquisas desenvolvidas em [4], [5], [6] e [7].

Nesta pesquisa efetuamos uma breve comparação de alguns cálculos de números com muitas casas decimais e fazemos uma breve comparação dos resultados retornados de vários tipos de variáveis com os resultados retornados pelo cálculo utilizando variáveis da biblioteca GMP (GNU Multiple Precision Arithmetic Library), que permite configurar a precisão da variável do modo desejado, a fim de verificar os casos de uso e o benefício dos aumentos de precisão e consequente poder de representação.

II. METODOLOGIA E FERRAMENTAS

Este projeto aborda uma pesquisa de natureza aplicada, investigativa e experimental. O objetivo central é explorar a biblioteca GMP (GNU Multiple Precision Arithmetic Library) em cálculos que exigem precisão elevada. A pesquisa busca soluções práticas para problemas específicos de representação numérica, enquanto aprofunda a compreensão de problemas de interesse da comunidade científica em geral. A abordagem é também experimental, pois envolve identificar ferramentas e técnicas para calcular variáveis numéricas nos problema selecionados, e avaliar a performance da biblioteca GMP. Assim, esta pesquisa contribui para a compreensão dos limites de representação de linguagens e programação e identifica cenários onde o uso de outros artifícios computacionais podem ser úteis.

Os experimentos desta pesquisa foram conduzidos em uma máquina Acer Aspire 5 A515-45-R4ZF AMD, equipado com 8GB de RAM e um SSD de 256GB, baseado na arquitetura x64. A configuração incluiu ainda uma máquina virtual com o sistema operacional Debian GNU/Linux de 64 bits, com 4 GB de memória, 20 GB de capacidade de armazenamento e 5 Unidades Centrais de Processamento (CPUs). Nessa máquina virtual, foram empregados os compiladores GNU GCC (GNU Compiler Collection), Portugol Studio e Replit para compilar em linguagem Python. Para viabilizar a análise abrangente, os códigos foram implementados em três linguagens distintas: C, Python e Portugol. Para garantir que os resultados fossem precisos e não suscetíveis a erros de arredondamento, a precisão dos cálculos foi configurada, com a biblioteca GMP, para para 128 bits, portanto o dobro de precisão definido para a variável do tipo double, ver Tabela 1.

Existem diversos métodos para calcular o valor de π, que variam desde uso de séries infinitas à algoritmos iterativos mais simples. O método selecionado para o cálculo efetuado nesta pesquisa foi o Algoritmo de Chudnovsky, que é considerado o estado da arte atual para o cálculo de novas casas decimais em π, segundo [8]. O algoritmo utilizado tem boa eficiência e detem todos os recordes recentes de novas casas decimais para representação de π. O recorde mais recente, utiliza esse algoritmo, consegue representar o número transcedental com 100 trilhões de casas decimais, feito alcançado pela Google em um projeto sob comando da desenvolvedora Emma Haruka Iwao, com o intuito de demonstrar o poder computacional da Google Cloud, utilizando o programa Y-Cruncher (ferramenta de computação de constantes), que utiliza o algoritmo de Chudnovsky em uma versão super otimizada e de forma escalável, como explicado por [7].

Para os experimentos foi utilizada uma versão simples do algoritmo em C, em versões com e sem uso da biblioteca GMP, e em Python.

$$\frac{1}{\pi} = 12 \sum_{k=0}^{\infty} \frac{(-1)^k (6k)!(545140134k + 13591409)}{(3k)!(k!)^3 (640320^3)^{k+1/2}} \qquad (1)$$

A seguir descrevemos os resultados preliminares da pesquisa.

III. RESULTADOS

Nesta seção apresentamos os resultados obtidos nos experimentos realizados, identificando alguns erros de precisão notados em cada linguagem para diferentes tipos de dados. Além disso, efetuamos uma análise comparativa do desempenho da biblioteca GMP em relação às linguagens avaliadas.

a. Cálculo de Fibonacci

Números em pontos flutuantes são representados utilizando a notação científica de base 2. Como foi dito anteriormente, eles possuem uma mantissa e um expoente, ver[1]. No entanto, a precisão é limitada, o que pode acarretar em arredondamentos indevidos de números muito extensos ou muito pequenos. Isso ocorre devido à limitação de bits disponíveis para a mantissa e ao alcance do expoente.

Nos experimentos, foram utilizados os 100 primeiros números da sequência de Fibonacci como base. Observamos que, durante os testes, os dados do tipo float em linguagem C foram os primeiros a apresentar erros de representação. Esses erros tornaram-se evidentes a partir do 37º termo. O número 24157817 é o 37º termo da sequência e possui 8 casas decimais, ver [9]. No entanto, foi notado que a

 ISSN: 2675-3588

representação do tipo float foi arredondada para 24157816, resultando em uma imprecisão númerica. Até esse ponto, todos os outros tipos de dados conseguiram reproduzir com precisão o valor exato dos números já conhecidos da sequência, exceto pelo tipo float.

Um tipo de dado inteiro em linguagem C e Portugol possui 32 bits. Para esse tipo, o intervalo de representação é de -2147483648 a 2147483647 como mostra a Tabela 1. Uma operação que resulte em um valor maior que 2147483647 ou menor que -2147483648 resultará em um overflow caso ultrapasse o limite máximo, ou underflow se for menor que o limite mínimo. Portanto, quando um número inteiro ultrapassa o valor máximo que pode ser representado com o número de bits disponíveis, ocorre um rollover (estouro de representação com mudança no sinal), onde o bit mais significativo é usado para continuar contando a partir do valor mínimo. Ao lidar com tipos inteiros nas linguagens C e Portugol, observou-se um início de divergências a partir do 47º termo na sequência. O número conhecido nessa sequência é 2971215073, ver[9]. No entanto, em ambas as linguagens, os números foram representados de forma imprecisa, sendo este -1323752223, indicando um overflow.

O próximo tipo a demonstrar imprecisão em sua representação foi o tipo Double, em linguagem C, esse tipo de dado representa números de ponto flutuante usando uma quantidade finita de bits para representar a mantissa e o expoente. O arredondamento ocorre porque a representação binária de números fracionários não pode ser exata para todos os valores possíveis. Algumas frações decimais não têm uma representação binária precisa e isso pode levar a arredondamentos. Com isso, a partir do 79º termo, começaram a surgir divergências. Para ilustrar, o número correspondente ao 79º termo da sequência é 14472334024676221, ver[9]. No entanto, a representação em double foi 14472334024676220, ocorrendo um erro no último dígito, onde houve arredondamento de 1 para 0.

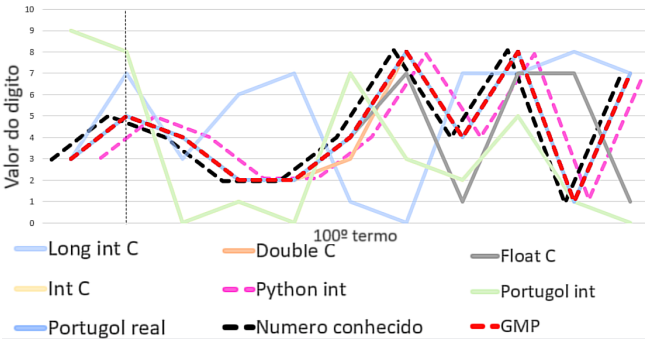

Figura 1: Gráfico representativo do 100º termo de Fibonacci

A última variável a apresentar falhas de representação durante o cálculo dos primeiros 100 números foi o tipo Unsigned Long Int, em linguagem C. Essa imprecisão surgiu no 94º termo. Para contextualizar, o valor correto para o 94º termo é 19740274219868223167 [9]. Entretanto, a representação do tipo Unsigned Long Int foi 1293530146158671551.

Na Figura 1, é apresentado um gráfico que ilustra a variação de desempenho entre os diferentes tipos de variáveis no cálculo do 100º termo da sequência de Fibonacci até o 11º dígito.Na Figura 2, essa análise é estendida até os últimos

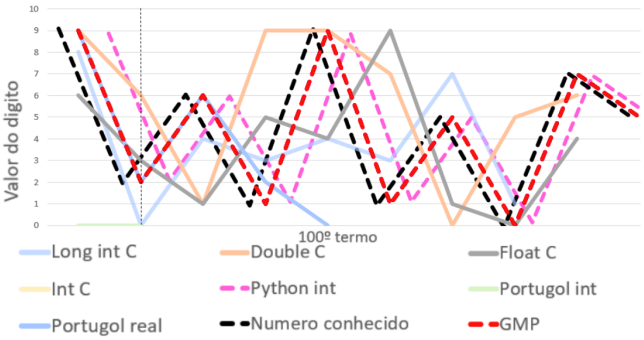

Figura 2: Continuação do gráfico representativo do 100º termo de Fibonacci

10 dígitos do mesmo termo. Este termo é representado pelo número identificado como 354224848179261915075, ver[9].O eixo das abscissas (x) denota o i-ésimo dígito do 100º termo de Fibonacci, enquanto o eixo das ordenadas (y) representa o valor do correspondente dígito no 100º termo. A análise gráfica evidencia os comportamentos distintos exibidos por cada tipo de variável em relação à representação deste número. Importante destacar que o tipo inteiro em Python não se vê limitado em termos de tamanho. Em virtude desta particularidade da linguagem Python, foi possível o cálculo preciso de todos os termos da sequência até o limiar dos 100 números. Com isso novos experimentos foram conduzidos, abrangendo os primeiros 2000 números da sequência. Tanto em Python quanto com o uso da biblioteca GMP, foi possível representar com precisão esses 2000 números, uma vez que não houve perda de exatidão devido a limitações de tamanho de variáveis.

No contexto da biblioteca GMP, cumpre salientar que a consecução do cálculo e a representação integral do número composto por 21 dígitos ocorreu sem manifestações de erros de arredondamento ou perda de acurácia. Este desfecho evidencia a proficiência da biblioteca GMP em lidar com cálculos que requerem altos níveis de exatidão numérica, mantendo a integridade da precisão mesmo ao lidar com valores numericamente extensos.

Os resultados iniciais dos experimentos evidenciam que o uso da biblioteca GMP é útil, por exemplo, para o cálculo da sequência de Fibonacci a partir do 79º termo da sequência na maioria das linguagens e tipos de variáveis utilizadas. No caso da linguagem Python há uma peculiaridade que o tamanho do inteiro possui como única restrição a capacidade de memória disponível, ver[10]. Esses foram os principais resultados inclusos neste lote de experimentos.

b. Cálculo de π

O algoritmo de Chudnovsky proporciona uma convergência em π extremamente rápida, com cerca de 14 dígitos por iteração, segundo [8], o suficiente para estourar rapidamente o espaço de memória das variáveis comuns. Foram utilizadas 5 iterações do algoritmo, com o intuito de obter 50 casas decimais com uma relativa margem de segurança, para que o algoritmo não interfira negativamente na demonstração dos tipos de variáveis.

No tipo Float, para a linguagem C, o tamanho da variável não consegue comportar partes do cálculo do algoritmo

de Chudnovksy, o que fez com que o último dígito (7º dígito) recebesse um valor errado, mesmo ainda estando nos limites estabelecidos para essa variável, resultando no valor 3.141592502...[2]

A implementação do algoritmo de Chudnovksy utilizando o tipo Double para o cálculo de π é menos prejudicada pela precisão da variável. É possível obter exatamente o valor conhecido até 15º dígito, o último dígito comportado pela especificação do tipo Double, alcançando o valor de 3.141592653589793560...

Em Python, existe uma tipagem dinâmica, onde não se faz necessário especificar o tipo da variável, apenas inicializar um valor sobre ela, ver [10]. Porém, pela documentação da linguagem, é possível observar que os números de ponto flutuante são armazenados em um tipo denominado Float, mas que possui a mesma precisão do Double em C. Como o tamanho da variável Float em Python é o mesmo da variável Double em C, espera-se que a precisão da mesma seja exatamente igual, que é o resultado obtido. O valor resultante do cálculo em Python é 3.141592653589793560... onde é possível notar que até o erro resultante é o mesmo.

Por último, utilizando a biblioteca GMP em linguagem C, ainda com o algoritmo de Chudnovsky, porém, adaptado para a biblioteca. Foi utilizada primeiramente uma precisão de 128 bits, o dobro de uma variável Double comum, para efeitos de comparação. O valor resultante possui 39 casas decimais corretas, como pode ser observado abaixo:

3.14159265358979323846264338327950288419500...

Uma nova rodada de testes foi realizada, desta vez com 192 bits de precisão, para comprovar que o único fator limitante para o algoritmo retornar os 50 dígitos solicitados foi a precisão configurada, e o número calculado foi:

3.14159265358979323846264338327950288419716939937511

Contendo 49 casas decimais com o valor correto. Para fins de comparação, o valor exato de π, ver [11], com 50 dígitos é:

31415926535897932384626433832795028841971693993751 0

Percebe-se então que há uma falha na computação do último dígito solicitado utilizando a biblioteca GMP. Após mais alguns testes, percebeu-se que ao aumentar o número de casas decimais exibidas com 192 bits de precisão, o resultado voltou a se apresentar corretamente, e novamente o último dígito estava incorreto. Após algumas pesquisas, foi possível confirmar que tal falha se trata de um erro de arredondamento da própria biblioteca, que não apresenta uma solução de arredondamento exato, problema que foi corrigido por uma biblioteca de extensão/melhoria da própria GMP, a MPFR (Multiple-Precision Binary Floating-Point Library), ver [12].

IV. Considerações Finais

Nesta pesquisa investigamos características de precisão e intervalo de representação de variáveis de algumas linguagens de programação e efetuamos uma comparação com a biblioteca de precisão arbitrária conhecida como GMP.

Algumas curiosidades interessantes merecem ser destacadas. Primeiro, notamos que em termos das linguagens Portugol, C e Python, todas elas possuem limitações parecidas, com a exceção do tipo inteiro da linguagem Python, que possui uma precisão dinâmica conforme a necessidade e o armazenamento disponível [10]. Fora essa exceção, as linguagens possuem um intervalo de representação relativamente próximo, o que era o resultado esperado, afinal todas seguem as especificações da IEEE 754 em suas variáveis de ponto flutuante.

Ademais, é possível observar que o uso da GMP ou de qualquer outra biblioteca de precisão arbitrária é útil quando necessário efetuar cálculos envolvendo muitas casas decimais e/ou inteiros muito extensos, proporcionando resultados mais precisos, sendo possível ajustar a precisão conforme necessidade. Um detalhe negativo sobre o uso da GMP é que devido ser uma biblioteca antiga, as referências encontradas estão, na maioria das vezes, defasadas.

Como trabalhos futuros pretendemos dar continuidade aos experimentos comparando resultados de cálculos de constantes reais, irracionais e transcendentais, como o número de euler, dentre vários outros, por diversos métodos, a fim de comparar quando é mais interessante o uso de cada método de cálculo e qual precisão necessária para utilizar cada um. Também é de nosso interesse investigar e comparar a eficiência de outras bibliotecas de precisão arbitrária como a MPFR, que além de permitir alterações de precisão, proporciona um arredondamento correto do número, ver [12]

[2]Os dígitos sublinhados representam os valores incorretos da sequência. Apenas os 3 primeiros dígitos incorretos foram apresentados, os demais valores foram ocultados para evitar poluição textual.

Agradecimentos

O presente trabalho foi realizado com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – Brasil.

Referências

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *Association for Computing Machinery, Inc.*, 1991. [Online]. Available: https://pages.cs.wisc.edu/~david/courses/cs552/S12/handouts/goldberg-floating-point.pdf

[2] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Prentice Hall Professional Technical Reference, 1988.

[3] R. Dawson, "Programming in ANSI C," 7 2012. [Online]. Available: https://repository.lboro.ac.uk/articles/book/Programming_in_ANSI_C/9405848

[4] D. H. Bailey, R. Barrio, and J. M. Borwein, "High-precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218, no. 20, pp. 10 106–10 121, 2012.

[5] G. Yarmish and J. Yarmish, "Finding large primes," *Utilitas Mathematica*, vol. 114, Mar. 2020. [Online]. Available: https://utilitasmathematica.com/index.php/Index/article/view/1510

[6] D. W. Brzeziński and P. Ostalczyk, "Numerical calculations accuracy comparison of the inverse laplace transform algorithms for solutions of fractional order differential equations," *Nonlinear dynamics*, vol. 84, pp. 65–77, 2016.

[7] E. H. Iwao, "Pi in the sky: Calculating a record-breaking 31.4 trillion digits of archimedes' constant on google cloud." [Online]. Available: https://cloud.google.com/blog/products/compute/calculating-31-4-trillion-digits-of-archimedes-constant-on-google-cloud

 ISSN: 2675-3588

[8] J. Guillera, "History of the formulas and algorithms for pi," *arXiv: History and Overview*, 2009. [Online]. Available: https://doi.org/10.48550/arXiv.0807.0872

[9] "Oeis sequence a000045: Fibonacci numbers," Online Encyclopedia of Integer Sequences. [Online]. Available: https://oeis.org/A000045/b000045.txt

[10] P. S. Foundation, "Python 3.11.4 documentation," 2023. [Online]. Available: https://docs.python.org/3/library/stdtypes.html#typesnumeric

[11] R. Sedgewick and K. Wayne, "first 1 million digits of pi." [Online]. Available: https://introcs.cs.princeton.edu/java/data/pi-1million.txt

[12] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, p. 13–es, jun 2007. [Online]. Available: https://doi.org/10.1145/1236463.1236468

ISSN: 2675-3588