UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

Pitágoras de Azevedo Alves Sobrinho

ULP-CALC

Ferramenta para calculo de *Unit Last Place* e analise de precisão de ponto flutuante

Natal, Rio Grande do Norte.
4 de março de 2015

1. INTRODUÇÃO

O *machine epsilon*, ou simplesmente ε, é na aritmética de ponto flutuante o menor número x tal que:

$$x + 1 > 1$$

Assim o *machine epsilon* nos fornece um valor para o erro relativo máximo numa representação de ponto flutuante. O *machine epsilon* pode ser calculado da seguinte forma, numa representação de base numérica binaria:

$$\varepsilon = 2^{-(p-1)}$$

Onde **p** é a precisão da representação em ponto flutuante. Como estamos lidando apenas com números binários, **p** é o numero de bits da mantissa mais o bit implícito **1**.

Porém, ao tentarmos calcular o erro absoluto numa determinada precisão (o espaçamento entre os números) a tarefa pode se tornar mais complexa devido a própria natureza do ponto flutuante. Pois devido ao valor representado depender tanto da mantissa quanto do expoente, valores maiores precisam de um valor maior no expoente. Com o valor no expoente maior, o valor do bit menos significativo da mantissa acaba se tornando cada vez mais, aumentando assim o erro absoluto. Assim, o erro absoluto máximo para diferentes números em ponto flutuante armazenados no computador pode ser diferente.

Este erro absoluto chama-se *Unit in the Last Place* ou ULP. Ele pode ser calculado através da seguinte função:

$$ULP(x) = \begin{cases} epsilon, & \text{se } |x| = 1 \\ epsilon \times 2^{\lfloor log_2 x \rfloor}, & \text{se } |x| \neq 1 \end{cases}$$

Sendo assim, o valor de ULP de x aumenta, conforme o valor de x também aumenta.

Dado que o valor de ULP varia de acordo com o numero representado e de acordo pela precisão da mantissa, a ferramenta **ulp-calc** tem o objetivo de simplificar a analise do erro absoluto e relativo em representações de ponto flutuante que utilizam quantidades diferentes de bits.

2. METODOLOGIA

Esta tarefa foi desenvolvida como um projeto de software na linguagem C++ que não utiliza conceitos próprios do paradigma da programação orientada a objetos. Foi estabelecido um diretório compartilhado em nuvem através do software *Dropbox* para cooperação e edição simultânea dos códigos entre a dupla que o estava desenvolvendo.

A compilação dos códigos fonte foi feita inteiramente no ambiente *Linux* com o compilador G++.

A ferramenta para plotagem de gráficos escolhida é o *gnuplot*. Este é capaz de gerar gráficos a partir de dados coletados e funções matemáticas e salvar-lhos em arquivos de imagem.

Foram definidas como funcionalidades básicas finais as expressas no seguinte fluxograma:

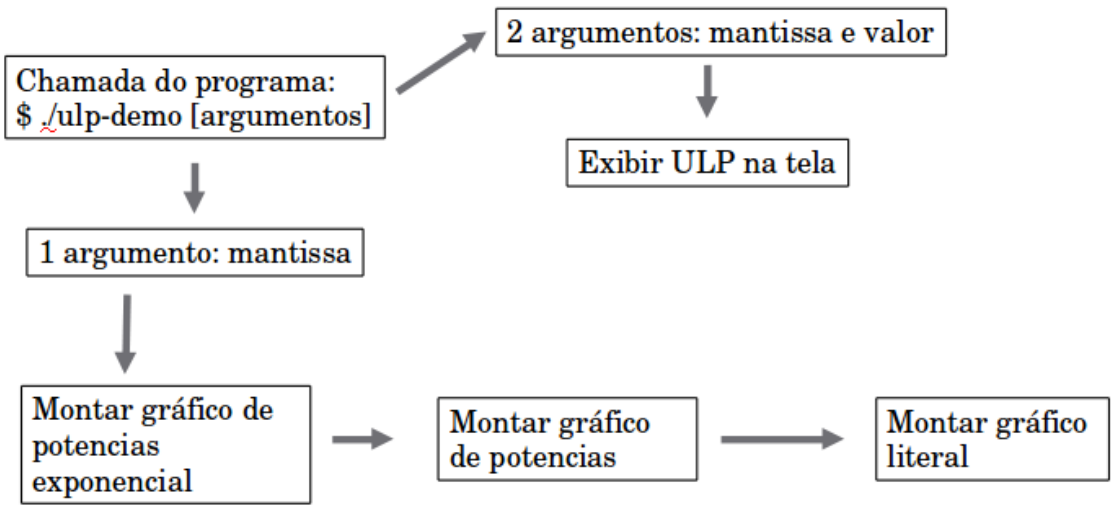

Figura 01: Fluxograma

3. DESENVOLVIMENTO

O software em si consiste das bibliotecas **multi** e **ulp-calc** que implementam as funções necessárias, do arquivo **main.cpp** que utiliza as funções de **multi** e **ulp-calc** para implementar as funcionalidades mostradas na Figura 01, e do *script* chamado **build** que, quando executado, constrói um executável chamado **ulp-demo** que pode ser utilizado pelo usuário.

3.1 As bibliotecas desenvolvidas

3.1.1 Biblioteca multi

Ela consiste de dois arquivos: **multi.h** onde estão declarados os protótipos das funções e **multi.cpp** onde estão suas implementações. Ela foi construída como um conjunto de funções para que se tornasse possível a execução de tarefas diferentes além desta, por isso apenas algumas destas funções definidas na biblioteca em questão foram utilizadas para esta tarefa. As principais funções utilizadas foram:

- **int** piso(**float** x): Faz o arredondamento para baixo do numero passado como parâmetro:

```
int piso(double x){
    return (int) x;
}
```

- **float** potencia(**int** x, **int** y): Eleva o numero x à potencia y e retorna o resultado:

```
float potencia(int x, int y){
    if(y < 0){
        return 1 / (potencia(x, -y));
    }else if(y == 0){
        return 1;
    }else{
        float result = x;
        for(int i = 1; i < y; i++){
            result = result*x;
        }
        return result;
    }
}
```

- **float** baseToBin(**string** x, **int** base): Recebe um numero escrito numa

sequencia de caracteres (string) chamado “x” numa base numérica determinada pelo argumento “base” e retorna seu valor em ponto flutuante:

```
float baseToBin(std::string x, int base){
    //point é a posição, na string, do ponto que separaria
    //a parte inteira da parte fracionaria do numero
    int point = 0;
    std::string parteInteira = "", parteFracionaria = "";
    int inteira = 0;
    float fracionaria = 0;

    //localiza o ponto
    for(int i = 0; i < x.size(); i++){
        if(x[i] == '.'){
            point = i;
            break;
        }
    }

    if(point == 0){
        //se point ainda é 0, então o numero
        //não deve ter parte fracionaria
        parteInteira = x;
        reverseString(parteInteira);
        parteFracionaria = "0";
    }else{
        //divide a string em duas
        parteInteira = x.substr(0, point);
        reverseString(parteInteira);
        parteFracionaria = x.substr(point+1, x.size()-1);
    }

    //agora com a string da parte inteira(invertida), faz-se
    //a soma de potencias.
    for(int i = 0; i <= parteInteira.size()-1; i++){
        inteira += charToInt(parteInteira[i]) * potencia(base, i);
    }

    //é feita a soma de potencias novamente, agora para a parte
    //fracionaria
    for(int i = 0; i <= parteFracionaria.size()-1; i++){
        fracionaria += charToInt(parteFracionaria[i]) * potencia(base, -i-1);
    }

    return inteira + fracionaria;
}
```

3.1.2 Biblioteca ulp-calc

Ela consiste de dois arquivos: **ulp-calc.h** onde estão declarados os protótipos das funções e **ulp-calc.cpp** onde estão suas implementações.

A principal função aqui é a função “**double** ulp(**int** p, **double** x)” onde “p”

é a precisão da representação numérica (largura da mantissa + 1) e “x” é o valor cujo ULP deve ser calculado. A função é a seguinte:

```
double ulp(int p, double x){
    if(x == 1){
        //ulp de 1
        return potencia(2, -(p-1)); //2^-(p-1)
    }else{

        return ulp(p, 1) * potencia(2, piso(log2(x)));
        //ulp(x) = ulp(1) * 2^(funçãoPiso(logx))
    }
}
```

Aqui vemos um caso da única função neste software que não foi implementada por nós, a função “**float** log2(**float** x)”. Ela vem da biblioteca “math.h” do C++.

Além da função para o calculo do ULP em si há 3 funções para criação de gráficos. Os algoritmos deles são muito semelhantes, com algumas diferenças no *script* de definição do gráfico ou nos dados que estão sendo armazenados. A função para criar o terceiro gráfico, o gráfico literal de ULP, é o seguinte:

```
/**
Esta função cria um grafico com os valores de ulp não apenas para as potencias de 2.
por isso ela adquire a aparencia de "degraus".
*/
std::string trueGraph(int mant){
    //stream (fluxo) de dados para o arquivo do grafico
    std::ofstream graph;
    //esse numero "-3" no fim é importante para não sobrescrever outros graficos
    std::string archiveName = "output-" + std::to_string(mant) + "-3";
    //definie o valor maximo até onde serão calculados os ulps
    float maxValue = potencia(2, mant-1);
    //o arquivo com o script e os dados terá a extensão ".data"
    graph.open(archiveName + ".data");

    //comando de plotagem para ser interpretado pelo software "gnuplot"
    std::string plotDeclaration = "set title \"" + std::to_string(mant) + " PRECISION ULP - GRAPH 3\"\n" +
        "set xrange[1:" + std::to_string(maxValue) + "]\n" +
        "set xlabel \"x\"\n" +
        "set yrange[0:1]\n" +
        "set ylabel \"ULP(x)\"\n" +
        "set terminal pngcairo size 900,700 enhanced font 'Verdana,10'\n" +
        "set output \"" + archiveName + ".png\"\n" +
        "plot \"-\" using 1:2 title 'ULP(X)' with linespoints\n";
    graph << plotDeclaration;

    //ticks existe porque não podemos simplesmente testar todos os valores de x possiveis
    float ticks = maxValue / 10000; //o grafico terá 10000 "pontos"
    for (double i = 1; i <= maxValue; i+=ticks) {
        graph << i << "\t" << ulp(mant, i) << "\n";
    }

    //fecha arquivo
    graph.close();
    std::string plotCommand = "gnuplot " + archiveName + ".data";
    //converte a string plotCommand para char[] e passa para o prompt do sistema
    system(plotCommand.c_str());

    //retorna o nome do arquivo
    return archiveName + ".data";
}
```

3.2 Os gráficos gerados

Os gráficos gerados com o auxilio do software *gnuplot* demonstram graficamente a variação do espaçamento entre os números em ponto flutuante, o ULP. Cada um tem o objetivo de demonstrar essas variações de uma forma ligeiramente diferente.

3.2.1 Gráfico das potencias de 2 exponencial

Como na função ULP(x) só é levado em consideração o piso do $\log_2 x$, é um comportamento característico dessa função o valor do ULP(x) só variar a cada potencia de 2. Para a construção deste primeiro gráfico são armazenados os dados do ulp de cada potencia de 2 até que o ulp seja igual a 1.

Estes dados são então plotados num gráfico em que cada espaçamento entre os valores no eixo x não progride de forma constante, mas de acordo com o $\log_2$, para que cada potencia possa ser analisada separadamente. Isso dá ao gráfico uma aparência de uma função exponencial. A partir desse gráfico é possível ver intuitivamente as diferenças em magnitude dos ulp's nas diferentes potencias de 2.

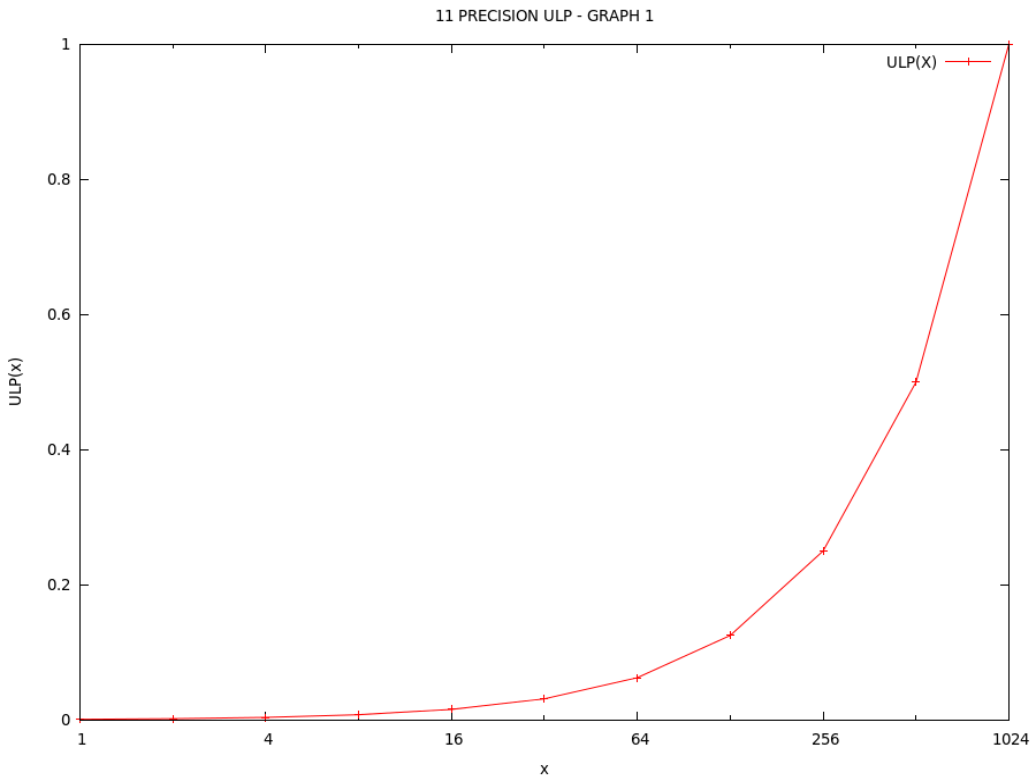

Figura 02: Gráfico exponencial das potencias do IEEE-754 Half-Precision

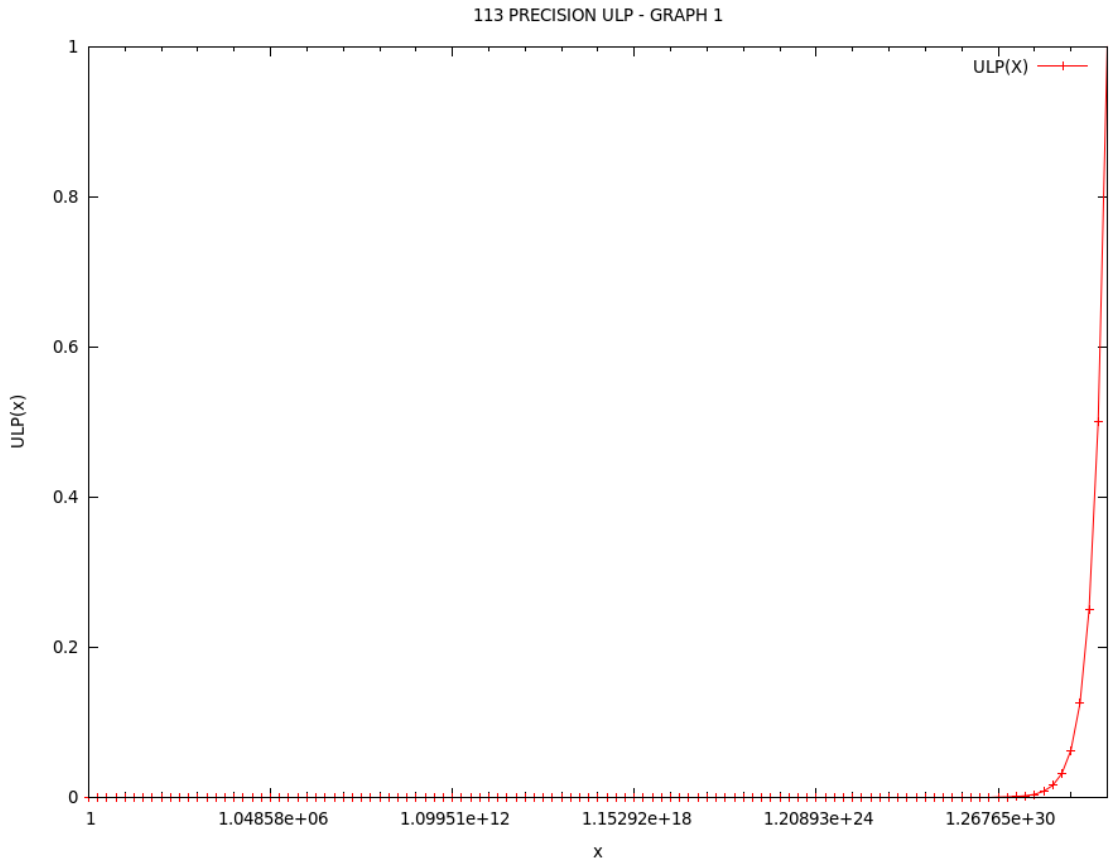

Figura 03: Gráfico exponencial das potencias do IEEE-754 Quad-Precision

3.2.2 Gráfico das potencias de dois

Este gráfico é muito semelhante ao gráfico descrito em 3.2.1. A única diferença é o eixo x, que aqui não foi alterado. Este gráfico toma então a forma de uma função de primeiro grau, uma reta crescente. Este gráfico mostra como para todas as potencias de 2, mesmo com ulp's muito diferentes, o erro relativo (a razão entre ULP(x) e x) é o mesmo: o *machine epsilon.*

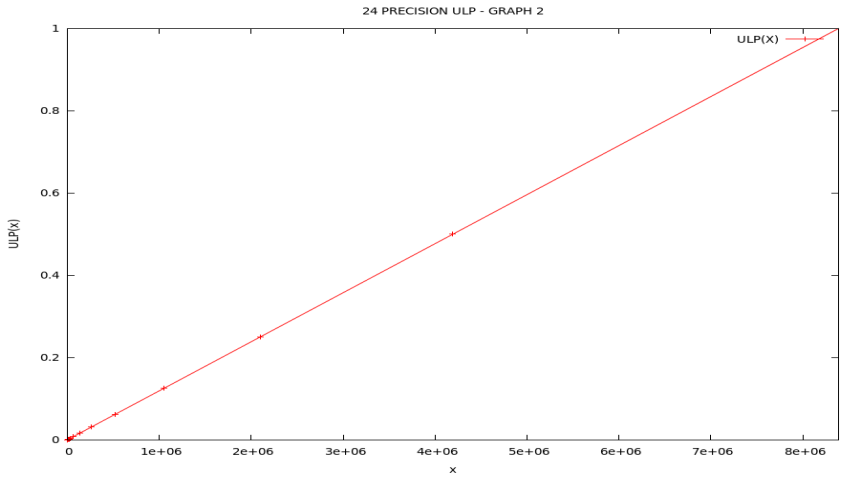

Figura 04: Gráfico das potencias de IEEE-754 Single-Precision

3.2.3 Gráfico literal de ULP

Este é o menos abstrato dos gráficos gerados nesta tarefa. Aqui são armazenados os resultados de ULP(x) para um grande numero de valores a partir de x = 1 até quando ULP(x) = 1. Quanto maior o numero de resultados armazenados, maior é a precisão deste gráfico, porém mais lento se torna o software é mais memoria é gasta.

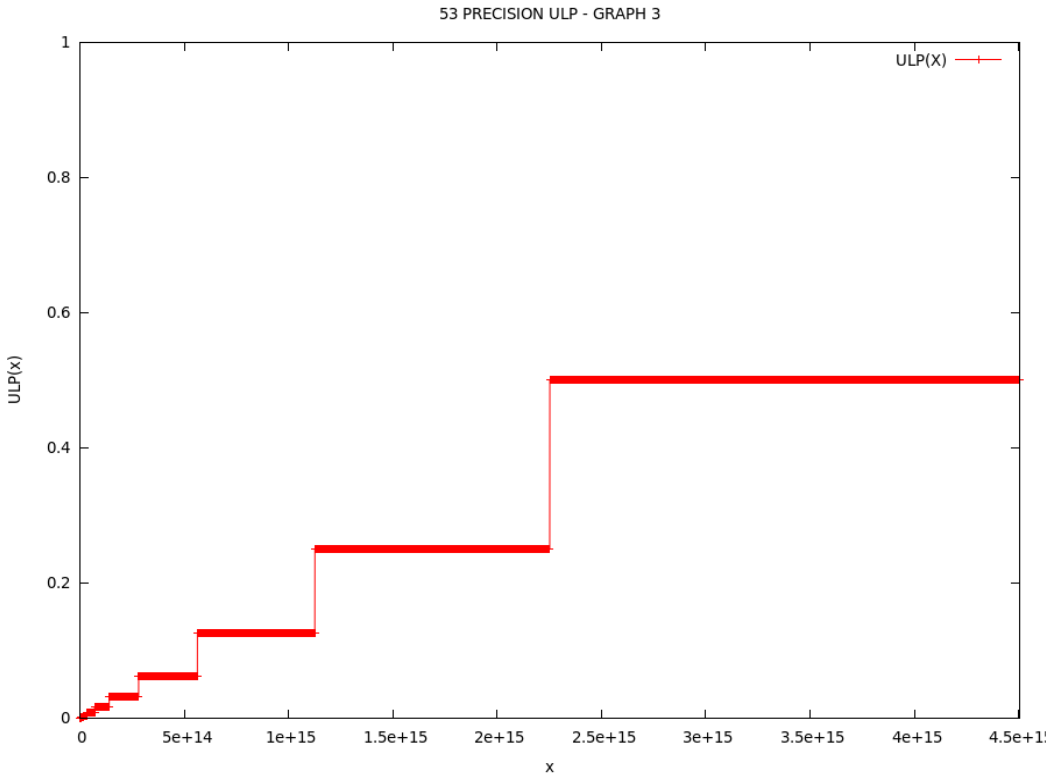

Figura 05: Gráfico para ULP de IEEE-754 Double-Precision

4. REFERÊNCIAS

- Épsilon de Maquina, Wikipedia. Acessado em 04/04/2015. http://pt.wikipedia.org/wiki/%C3%89psilon_de_m%C3%A1quina;
- Unit last Place, Wikipedia. Acessado em 04/04/2015. http://en.wikipedia.org/wiki/Unit_in_the_last_place;
- Compilador G++, do GNU Compiler Collection: https://gcc.gnu.org/;
- Gnuplot: http://www.gnuplot.info;
- Dropbox: https://www.dropbox.com;