UNIVERSIDADE FEDERAL DA BAHIA

DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO

LUCAS MENEZES PEREIRA

MATHEUS OLIVER DE CARVALHO CERQUEIRA

RELATÓRIO DE COMPARAÇÃO DE PROCEDIMENTOS C/ASSEMBLY/JAVA PARA OPERAÇÕES MATEMÁTICAS COM NOTAÇÃO DE PONTO FLUTUANTE

Salvador

2017

1. APRESENTAÇÃO

C é uma das linguagens de programação mais populares, sendo predominante no ensino básico de programação devido à semelhança semântica e sintática com a língua inglesa, o que torna seu entendimento acessível a grande parte da população (linguagem de alto nível). Assim, a maioria das arquiteturas prevê um compilador C.

Assembly, no entanto é uma notação voltada ao comando direto à máquina, que é utilizado por arquiteturas específicas de computador, para programar códigos entendidos por dispositivos computacionais, como microprocessadores e microcontroladores. Nesse sentido, linguagens de alto nível surgem como uma camada, em que o programador utiliza procedimentos, que são traduzidos pelo compilador em linguagem de máquina, que é interpretada pelo dispositivo.

Portanto, como a linguagem C tem, internamente em suas funções, comandos em Assembly, os programas podem ser construídos mesclando as linguagens, com intenção de otimizar o código em trechos específicos. A otimização é muito útil para processamento de quantidades extraordinárias de dados, ou quando a precisão das medidas de entradas e saída deve ser máxima.

A representação de números em ponto flutuante surgiu como alternativa para que métodos computacionais pudessem abrangir essa precisão e extensão que diversas aplicações como Astronomia e Astronáutica, Meteorologia e Computação gráfica requerem. Essa notação será utilizada para expressar resultados das operações.

A linguagem JAVA, baseada em C, é uma das linguagens de programação mais utilizadas, possuindo diversas bibliotecas e interface que facilitam o trabalho dos programadores. Seu ponto forte é a facilidade de execução de um mesmo programa em diversos sistemas operacionais.

Essas três linguagens serão analisadas neste relatório do ponto de vista de tempo de execução.

2. MATERIAIS UTILIZADOS

2.1. Sistema Operacional:

- Windows 10 64 bits, com processador em x64.

2.2. Hardware:

- 4GB de memória RAM (Utilizável: 3,89GB);
- Intel Core i3-3010M CPU @2.40GHz.

2.3. Software:

- Netbeans.

3. OBJETIVOS

3.1. Geral

Comparar as execuções de operações matemáticas entre números em notação de Ponto Flutuante para cada linguagem (C, Assembly e JAVA) e perceber como o código pode ser modificado, a partir da integração de linguagens, para deixar a execução mais rápida.

3.2. Específicos

Considerando as funções abaixo, para um número x real:

- Módulo;
- Exponencial Natural;
- Logaritmo Natural;
- Raiz Quadrada;
- Cosseno;
- Seno;
- Tangente;
- Arco Cosseno;
- Arco Seno;
- Arco Tangente.

Deve-se:

- Desenvolver programa em C para o cálculo de cada função matemática;
- Desenvolver programa em JAVA para o cálculo de cada função matemática;
- Desenvolver programa em C/ASSEMBLY para o cálculo de cada função. Neste caso, procedures em ASSEMBLY que realizam o cálculo das funções serão desenvolvidas e chamadas no programa em C;
- Comparar o tempo de execução dos programas desenvolvidos na linguagem C e na linguagem JAVA e indicando o ganho de desempenho, expressando os motivos para tal. Considera-se o ganho como a relação entre o tempo de execução em C e o tempo de execução em C/ASSEMBLY.

4. METODOLOGIA

Cada objetivo será cumprido individual e sequencialmente. Como padrão para os programas, a seguinte estrutura será aplicada:

- Inclusão de bibliotecas;
- Declaração de variáveis globais, sendo um contador inteiro para iteração, que compreenderá o intervalo de [-100000;100000[(Duzentas mil iterações) e servirá como entrada para as funções; e uma variável do tipo “double” (dupla precisão, ou precisão estendida), para receber os resultados das funções para cada iteração. Sabe-se que esses valores não funcionarão integralmente para todas as funções, mas, sendo a comparação entre linguagens, o número de iterações bem-sucedidas é o mesmo.
- Execução do programa contendo a iteração e a contagem dos tempos. O princípio da cronometragem será utilizar funções nativas para pegar o instante de tempo imediatamente antes da primeira iteração, o imediatamente após a última, realizar a subtração desses instantes e exibir o intervalo de tempo em milissegundos. Tal precisão facilitará o processo de comparar os tempos, sendo suficiente para as duzentas mil contagens a cada operação matemática.

A execução da mesma estrutura para os programas de linguagens diferentes é a terceira garantia de um julgamento justo, sendo a primeira o fato de serem rodados na mesma máquina e a segunda é que a execução dos programas será feita na mesma IDE (Netbeans). Pode-se adicionar um quarto critério, que foi garantir que nenhum outro programa, além da IDE estivesse em primeiro ou segundo plano de execução na máquina, havendo a tendência de utilização quase semelhante da CPU, de disco e de memória para todas as linguagens.

Dada a estrutura de programação, foi estabelecido um total de cinco execuções de cada operação, de cada linguagem, com objetivo de estimar melhor o tempo de execução médio (o que não aconteceria se os valores de tempo de cada função fossem constantes).

5. OPERACIONAL

5.1. Linguagem C

A inclusão das bibliotecas foi dada por:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <sys\timeb.h>
```

A fim de tornar a execução mais rápida, cada função foi transformada eu um programa (seria o mesmo se as partes não utilizadas em cada análise de função fosse comentada). Assim, como cada função corresponde a um programa, as operações foram construídas internamente ao programa, de forma que a única diferença entre as

funções está nas duas linhas internas ao for(), e, consequentemente, no nome das variáveis que guardam o valor de retorno das funções. Essa diferença será explicitada nos códigos a seguir, com as palavras em **negrito**.

Função Módulo de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double MODULO;
	for(i=-100000;i<100000;i++){
		MODULO = abs(i);
		printf("\n%lf", MODULO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Exponencial Natural de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double EXPONENCIAL;
	for(i=-100000;i<100000;i++){
		EXPONENCIAL = exp(i);
		printf("\n%lf", EXPONENCIAL);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Logaritmo Natural de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double LOGARITMO;
	for(i=-100000;i<100000;i++){
		LOGARITMO = log(i);
		printf("\n%lf", LOGARITMO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
```

```
        system("pause");
}
```

Função Raiz de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double RAIZ;
	for(i=-100000;i<100000;i++){
		RAIZ = sqrt(i);
		printf("\n%lf", RAIZ);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Cosseno de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double COSSENO;
	for(i=-100000;i<100000;i++){
		COSSENO = cos(i);
		printf("\n%lf", COSSENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Seno de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double SENO;
	for(i=-100000;i<100000;i++){
		SENO = sen(i);
		printf("\n%lf", SENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Tangente de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double TANGENTE;
	for(i=-100000;i<100000;i++){
		TANGENTE = tan(i);
		printf("\n%lf", TANGENTE);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Arco Cosseno de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double ARCOCOSSENO;
	for(i=-100000;i<100000;i++){
		ARCOCOSSENO = acos(i);
		printf("\n%lf", ARCOCOSSENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Arco Seno de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double ARCOSENO;
	for(i=-100000;i<100000;i++){
		ARCOSENO = asen(i);
		printf("\n%lf", ARCOSENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Arco Tangente de X:

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double ARCOTANGENTE;
	for(i=-100000;i<100000;i++){
		ARCOTANGENTE = atan(i);
		printf("\n%lf", ARCOTANGENTE);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Para esses códigos, foram repetidas 5 execuções para cada, e os resultados serão apresentados posteriormente, na seção RESULTADOS E DISCUSSÕES

5.2. Linguagem JAVA

No caso da linguagem JAVA, não foram incluídas bibliotecas no início do programa, mas as chamadas das funções ocorreram dentro das iterações (como em $abs = Math.abs(i)$;), devido à própria estrutura de JAVA. Excetuando isso, os programas seguem a mesma estrutura dos criados em C, desde a ordem de declaração de variáveis, até a contagem de tempo. O destaque em negrito, utilizado na seção de C, foi mantido aqui, para, novamente, evidenciar as semelhanças estruturais dos programas, entre si, e entre linguagens distintas.

Função Módulo de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double abs;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			abs = Math.abs(i);
			System.out.println(abs);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}
```

```
}
```

Função Exponencial Natural de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double exp;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			exp = Math.exp(i);
			System.out.println(exp);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}

}
```

Função Logaritmo Natural de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double log;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			log = Math.log(i);
			System.out.println(log);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}

}
```

Função Raiz de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double sqrt;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			sqrt = Math.sqrt(i);
			System.out.println(sqrt);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));

	}
```

```
}
```

Função Cosseno de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double cos;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			cos = Math.cos(i);
			System.out.println(cos);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}

}
```

Função Seno de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
		int i;
		double sin;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			sin = Math.sin(i);
			System.out.println(sin);
		}
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	 }
}
```

Função Tangente de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
	int i;
	double tan;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			tan = Math.tan(i);
			System.out.println(tan);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}
}
```

Função Arco Cosseno de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
	int i;
	double acos;
	long initialTime = System.currentTimeMillis();
	for(i=-100000;i<100000;i++){
		acos = Math.acos(i);
		System.out.println(acos);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	 }

}
```

Função Arco Seno de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
	int i;
	double asin;
		long initialTime = System.currentTimeMillis();
		for(i=-100000;i<100000;i++){
			asin = Math.asin(i);
			System.out.println(asin);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	 }
}
```

Função Arco Tangente de X:

```
package pontoflutuante;
public class Pontoflutuante{
	public static void main(String[] args) {
	int i;
	double atan;
	long initialTime = System.currentTimeMillis();
	for(i=-100000;i<100000;i++){
		atan = Math.atan(i);
		System.out.println(atan);
		 }
		long endTime = System.currentTimeMillis();
		System.out.println("Tempo total em milissegundos: "
+(endTime-initialTime));
	}

}
```

No caso do ASSEMBLY, foram elaborados dois métodos de associação à linguagem C, sendo Inline (In) o método de utilizar ASEMBLY no próprio programa em C, e Módulos separados (Sep) a chamada de procedures externas ao programa principal em C.

O método Inline é preferível quando se tem pouca instrução ASSEMBLY para embarcar. A sintaxe de ASSEMBLY para In, diferentemente do método Sep, que utiliza notação INTEL, é a arquitetura AT&T, que possui algumas diferenças. No código, essas diferenças podem influenciar no tempo de execução.

Para o método de módulos separados, mais comum quando há muitas instruções, é preciso passar o .asm (extensão do arquivo ASSEMBLY embarcado) por um montador, antes de compilar o arquivo.C. Ao utilizar as funções embarcadas, é necessário declarar "extern" antes da função.

5.3. Linguagem ASSEMBLY associada a programa em C

Como será utilizada a linguagem C, serão mantidas as bibliotecas utilizadas anteriormente:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <sys\timeb.h>
```

Entretanto, em vez de utilizar funções internas à main(), os programas foram escritos com chamadas a funções fora do main(), mas pertencentes ao arquivo C. Isso gera um gasto de poucos milissegundos em relação à técnica anterior, pois, quando os comandos são internos ao main(), o compilador "lê" o código sequencialmente, enquanto a chamada da função vai fazer a busca para o endereço em que a função foi alocada, e depois retornar o valor ao último endereço antes da execução.

Função Módulo de X:

```
double ASMabs(double x) {
        double ans;
        asm(
            "fldl %1;" //carrega x em st0
            "fabs;" //abs
            :"=st"(ans) //saida: st0
            :"m"(x) //entrada: x (na memoria)
            );
       return ans;
}
```

```
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double MODULO;
	for(i=-100000;i<100000;i++){
		MODULO = ASMabs(i);
		printf("\n%lf", MODULO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Exponencial Natural de X:

```
double ASMexp(double x) {
	double ans;
	double e = 2.71828182846;
	asm(
		"fldl %1;"
		"fldl %2;"
		"fyl2x;"
		"fld1;"
		"fld %%st(1);"
		"fprem;"
		"f2xm1;"
		"faddp;"
		"fscale;"
		"fxch %%st(1);"
		"fstp %%st"
		:"=st"(ans)
		:"m"(x), "m"(e)
		);
	return ans;
}
void main(){
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double EXPONENCIAL;
	for(i=-100000;i<100000;i++){
		EXPONENCIAL = ASMexp(i);
		printf("\n%lf", EXPONENCIAL);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
	system("pause");
}
```

Função Logaritmo Natural de X:

```
double ASMln(double x) {
	double ans;
	asm("fldl %1;"
		"fld1;"
		"fxch;"
		"fyl2x;"
		"fldl2e;"
		"fxch;"
		"fdivp;"
		:"=st"(ans)
		:"m"(x)
		);
	return ans;
}
void main() {
	struct timeb start, end;
	int diff;
	ftime(&start);
	int i;
	double LOGARITMO;
	for(i=-100000;i<100000;i++){
		LOGARITMO = ASMln(i);
		printf("\n%lf", LOGARITMO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
   printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Raiz de X:

```
double ASMsqrt(double x){
	double ans;
	asm(
		"fldl %1;"
		"fsqrt"
		:"=st"(ans)
		:"m"(x)
		);
	return ans;

}

void main() {

	struct timeb start, end;

	int diff;

	ftime(&start);

	int i;

	double RAIZ;

	for(i=-100000;i<100000;i++){

		RAIZ = ASMsqrt(i);
```

```
                printf("\n%lf", RAIZ);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
        printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Cosseno de X:

```
double ASMcos(double x) {
        double ans;
        asm("fldl %1;"
                "fcos"
                :"=st"(ans)
                :"m"(x)
        );
        return ans;
}
void main() {
        struct timeb start, end;
    int diff;
    ftime(&start);
        int     i;
        double COSSENO;
        for(i=-100000;i<100000;i++){
                COSSENO=ASMcos(i);
                printf("\n%lf", COSSENO);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
        printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Seno de X:

```
double ASMsin(double x) {//WORKING
        double ans;
        asm("fldl %1;"
                "fsin"
                :"=st"(ans)
                :"m"(x)
        );
        return ans;
}

void main() {
        struct timeb start, end;
    int diff;
    ftime(&start);
```

```
	int	i;
	double SENO;
	for(i=-100000;i<100000;i++){
		SENO = ASMsin(i);
		printf("\n%lf", SENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Tangente de X:

```
double ASMtan(double x) {//WORKING
	double ans;
	asm("fldl %1;"
		"fptan;"
		"fstp %%st;"
		:"=st"(ans)
		:"m"(x)
	);
	return ans;
}

void main() {
	struct timeb start, end;
   int diff;
   ftime(&start);
	int	i;
	double TANGENTE;
	for(i=-100000;i<100000;i++){
		TANGENTE = ASMtan(i);
		printf("\n%lf", TANGENTE);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Arco Cosseno de X:

```
double ASMacos(double x) {//WORKING

	double ans;

	asm("fldl %1;"

		"fldl %1;"

		"fmulp;"

		"fld1;"

		"fsubp;"
```

```
                "fsqrt;"
                "fldl %1;"
                "fpatan;"
                :"=st"(ans)
                :"m"(x)
        );
        return ans;
}
void main() {
        struct timeb start, end;
    int diff;
    ftime(&start);
        int        i;
        double ARCOCOSSENO;
        for(i=-100000;i<100000;i++){
                ARCOCOSSENO = ASMacos(i);
                printf("\n%lf", ARCOCOSSENO);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
        printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Arco Seno de X:

```
double ASMasin(double x) {
        double ans;
        asm(
                "fldl %1;"
                "fldl %1;"
                "fldl %1;"
                "fmulp;"
                "fld1;"
                "fsubp;"
                "fsqrt;"
                "fpatan;"
                :"=st"(ans)
                :"m"(x)
                );
          return ans;
```

```
}


void main() {
	struct timeb start, end;
   int diff;
   ftime(&start);
	int	i;
	double ARCOSENO;
	for(i=-100000;i<100000;i++){
		ARCOSENO = ASMasin(i);
		printf("\n%lf", ARCOSENO);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Arco Tangente de X:

```
double ASMatan(double x) {//WORKING
	double ans;
	asm("fldl %1;"
		"fld1;"
		"fpatan;"
		:"=st"(ans)
		:"m"(x)
	);
	return ans;
}


void main() {
	struct timeb start, end;
   int diff;
```

```
    ftime(&start);
        int         i;
        double ARCOTANGENTE;
        for(i=-100000;i<100000;i++){
                ARCOTANGENTE = ASMatan(i);
                printf("\n%lf", ARCOTANGENTE);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
        printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Para o ASSEMBLY (Sep)

//ASSEMBLY EXTERNO

```
;       double ans;
;       asm("fldl %1;" //carrega x em st0
;               "fabs;" //abs
;               :"=st"(ans) //saida: st0
;               :"m"(x) //entrada: x (na memoria)
;       );
;       return ans;

section .text

global _ASMabs

_ASMabs:
        enter   0,0
        fld     qword [ebp + 8]
        fabs
        leave
        ret
```

```
//C
extern double ASMabs(double x);


void main() {
      struct timeb start, end;
   int diff;
   ftime(&start);
      int i;
      double MODULO;
      for(i=-100000;i<100000;i++){
                  MODULO = ASMabs(i);
                  printf("\n%lf", MODULO);
      }
      ftime(&end);
      diff = (int) (1000.0 * (end.time - start.time)+(end.millitm -
start.millitm));
 printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Exponencial Natural de X:

//ASSEMBLY EXTERNO

```
;double ASMexp(double x) {//WORKING
;     double ans;
;     double e = 2.71828182846;
;     asm("fldl %1;"
;                 "fldl %2;"
;                 "fyl2x;"
;                 "fld1;"
;                 "fld %%st(1);"
;                 "fprem;"
```

```
;               "f2xm1;"
;               "faddp;"
;               "fscale;"
;               "fxch %%st(1);"
;               "fstp %%st"
;               :"=st"(ans)
;               :"m"(x), "m"(e)
;     );
;     return ans;
;}
```

//C

```
extern double ASMexp(double x);

void main() {
      struct timeb start, end;
   int diff;
   ftime(&start);
      int       i;
      double EXPONENCIAL;
      for(i=-100000;i<100000;i++){
                EXPONENCIAL = ASMexp(i);
                printf("\n%lf", EXPONENCIAL);
      }
      ftime(&end);
      diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
   printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Logaritmo Natural de X:

//ASSEMBLY EXTERNO

```
;double ASMln(double x) {//WORKING
```

```
;	double ans;
;	asm("fldl %1;"
;		"fld1;"
;		"fxch;"
;		"fyl2x;"
;		"fldl2e;"
;		"fxch;"
;		"fdivp;"
;		:"=st"(ans)
;		:"m"(x)
;	);
;	return ans;
;}

section .text

global _ASMln

_ASMln:
	enter 0, 0
	fld qword [ebp + 8]
	fld1
	fxch
	fyl2x
	fldl2e
	fxch
	fdivp
	leave
	ret
//C
extern double ASMln(double x);
```

```
void main() {
    struct timeb start, end;
   int diff;
   ftime(&start);
    int       i;
    double LOGARITMO;
    for(i=-100000;i<100000;i++){
            LOGARITMO = ASMln(i);
            printf("\n%lf", LOGARITMO);
    }
    ftime(&end);
    diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
   printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

```
package pontoflutuante;
    public class Pontoflutuante{
            public static void main(String[] args) {
                    int i;
double log;
                    long initialTime = System.currentTimeMillis();
                    for(i=-100000;i<100000;i++){
                    log = Math.log(i);
                            System.out.println(log);
                     }
                long endTime = System.currentTimeMillis();
                System.out.println("Tempo total em milissegundos: " +(endTime-initialTime));
 }
}
```

Função Raiz de X:

//ASSEMBLY EXTERNO

```
;double ASMsqrt(double x) {//WORKING
;    double ans;
```

```
;	asm("fldl %1;"
;		"fsqrt"
;		:"=st"(ans)
;		:"m"(x)
;	);
;	return ans;
;}

section .text

global _ASMsqrt

_ASMsqrt:
	enter 0,0
	fld  qword[ebp + 8]
	fsqrt
	leave
	ret
//C
.
extern double ASMsqrt(double x);

void main() {
	struct timeb start, end;
   int diff;
   ftime(&start);
	int i;
	double RAIZ;
	for(i=-100000;i<100000;i++){
		RAIZ = ASMsqrt(i);
		printf("\n%lf", RAIZ);
	}
	ftime(&end);
```

```
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));

	 printf("\n\nTempo total em milissegundos: %u\n", diff);

   system("pause");

}
```

Função Cosseno de X:

```
//ASSEMBLY EXTERNO

;double ASMcos(double x) {//WORKING

;	double ans;

;	asm("fldl %1;"

;		"fcos"

;		:"=st"(ans)

;		:"m"(x)

;	);

;	return ans;

;}


section .text


global _ASMcos


_ASMcos:

	enter 0, 0

	fld qword [ebp + 8]

	fcos

	leave

	ret


//C

extern double ASMcos(double x);

void main() {
	struct timeb start, end;
   int diff;
```

```
    ftime(&start);
      int       i;
      double COSSENO;
      for(i=-100000;i<100000;i++){
                COSSENO=ASMcos(i);
                printf("\n%lf", COSSENO);
      }
      ftime(&end);
      diff = (int) (1000.0 * (end.time - start.time)+(end.millitm -
start.millitm));
       printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Seno de X:

//ASEMBLY EXTERNO

```
;double ASMsin(double x) {//WORKING
;     double ans;
;     asm("fldl %1;"
;               "fsin"
;               :"=st"(ans)
;               :"m"(x)
;     );
;     return ans;
;}


section .text


global _ASMsin


_ASMsin:
      enter 0,0
      fld qword [ebp + 8]
      fsin
      leave
      ret
```

//C

```
extern double ASMsin(double x);

void main() {
      struct timeb start, end;
    int diff;
    ftime(&start);
      int       i;
      double SENO;
      for(i=-100000;i<100000;i++){
                SENO = ASMsin(i);
                printf("\n%lf", SENO);
      }
      ftime(&end);
      diff = (int) (1000.0 * (end.time - start.time)+(end.millitm -
start.millitm));
       printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Tangente de X:

//ASSEMBLY EXTERNO

```
;double ASMtan(double x) {//WORKING
;     double ans;
;     asm("fldl %1;"
;               "fptan;"
;               "fstp %%st;"
;               :"=st"(ans)
;               :"m"(x)
;     );
;     return ans;
;}

section .text

global _ASMtan

_ASMtan:
      enter 0, 0
```

```
        fld qword [ebp + 8]
        fptan
        fstp ST0
        leave
        ret
```

```
//C
extern double ASMtan(double x);

void main() {
        struct timeb start, end;
    int diff;
    ftime(&start);
        int       i;
        double TANGENTE;
        for(i=-100000;i<100000;i++){
                TANGENTE = ASMtan(i);
                printf("\n%lf", TANGENTE);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm -
start.millitm));
         printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Arco Cosseno de X:

```
//ASSEMBLY EXTERNO
;double ASMacos(double x) {//WORKING
;       double ans;
;       asm("fldl %1;"
```

```
;               "fldl %1;"
;               "fmulp;"
;               "fld1;"
;               "fsubp;"
;               "fsqrt;"
;               "fldl %1;"
;               "fpatan;"
;               :"=st"(ans)
;               :"m"(x)
;     );
;     return ans;
;}

section .text

global _ASMacos

_ASMacos:
      enter 0, 0
      fld       qword [ebp + 8]
      fld       qword [ebp + 8]
      fmulp
      fld1
      fsubp
      fsqrt
      fld       qword [ebp + 8]
      fpatan
      leave
      ret

//C
extern double ASMacos(double x);
```

```
void main() {
      struct timeb start, end;
   int diff;
   ftime(&start);
      int       i;
      double ARCOCOSSENO;
      for(i=-100000;i<100000;i++){
                ARCOCOSSENO = ASMacos(i);
                printf("\n%lf", ARCOCOSSENO);
      }
      ftime(&end);
      diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
       printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

Função Arco Seno de X:

//ASSEMBLY EXTERNO

```
;double ASMasin(double x) {//WORKING
;     double ans;
;     asm("fldl %1;"
;               "fldl %1;"
;               "fldl %1;"
;               "fmulp;"
;               "fld1;"
;               "fsubp;"
;               "fsqrt;"
;               "fpatan;"
;               :"=st"(ans)
```

```
;                :"m"(x)
;     );
;     return ans;
;}

section .text

global _ASMasin

_ASMasin:
      enter 0, 0
      fld qword [ebp + 8]
      fld qword [ebp + 8]
      fld qword [ebp + 8]
      fmulp
      fld1
      fsubp
      fsqrt
      fpatan
      leave
      ret

//C
extern double ASMasin(double x);

void main() {
      struct timeb start, end;
   int diff;
   ftime(&start);
      int       i;
      double ARCOSENO;
      for(i=-100000;i<100000;i++){
```

```
            ARCOSENO = ASMasin(i);
            printf("\n%lf", ARCOSENO);
        }
        ftime(&end);
        diff = (int) (1000.0 * (end.time - start.time)+(end.millitm - start.millitm));
         printf("\n\nTempo total em milissegundos: %u\n", diff);
    system("pause");
}
```

Função Arco Tangente de X:

//ASSEMBLY EXTERNO

```
;double ASMatan(double x) {//WORKING
;       double ans;
;       asm("fldl %1;"
;               "fld1;"
;               "fpatan;"
;               :"=st"(ans)
;               :"m"(x)
;       );
;       return ans;
;}

section .text

global _ASMatan

_ASMatan:
        enter 0, 0
        fld qword [ebp + 8]
        fld1
        fpatan
```

```
	leave
	ret

//C
extern double ASMatan(double x);


void main() {
	struct timeb start, end;
   int diff;
   ftime(&start);
	int	i;
	double ARCOTANGENTE;
	for(i=-100000;i<100000;i++){
		ARCOTANGENTE = ASMatan(i);
		printf("\n%lf", ARCOTANGENTE);
	}
	ftime(&end);
	diff = (int) (1000.0 * (end.time - start.time)+(end.millitm -
start.millitm));
	 printf("\n\nTempo total em milissegundos: %u\n", diff);
   system("pause");
}
```

6. RESULTADOS E DISCUSSÕES

Para a linguagem C, foram obtidos os seguintes valores:

Tabela de Tempos de Execução em Linguagem C (ms)						
	1	2	3	4	5	Média
abs()	3300	3308	3282	4270	3320	3496
exp()	2650	2550	2578	2591	2560	2585,8
log()	2630	2501	2610	2549	2580	2574
sqrt()	2800	2710	2780	2801	2752	2768,6
cos()	2580	2610	2519	2530	2651	2578
sin()	2780	2600	2550	2520	2570	2604
tan()	2841	2670	2580	2630	2590	2662,2
acos()	2720	2900	2980	2891	3030	2904,2
asin()	2900	2950	3070	2890	2970	2956
atan()	2841	2880	2720	2790	2830	2812,2

Tabela 1 – Tempos de execução em milissegundos para linguagem C.

Para a linguagem JAVA, foram obtidos os seguintes valores:

Tabela de Tempos de Execução em Linguagem JAVA (ms)						
	1	2	3	4	5	Média
abs()	12684	12207	12709	12202	11980	12356
exp()	12153	12299	12118	12265	12562	12279
log()	13541	13696	13568	13650	13998	13691
sqrt()	13730	13495	13564	13639	14531	13792
cos()	18102	17185	17223	17824	17259	17519
sin()	18446	17605	17518	17588	18005	17832
tan()	17309	18148	17365	17266	17725	17563
acos()	10240	10701	10210	10223	9994	10274
asin()	10773	10270	10270	10672	10309	10459
atan()	17537	17765	17721	17315	17498	17567

Tabela 2 – Tempos de execução em milissegundos para linguagem JAVA.

Para o Programa C, com procedures elaboradas em ASSEMBLY(In), foram obtidos os seguintes valores:

Tabela de Tempos de Execução em C/Assemby(In) (ms)						
	1	2	3	4	5	Média
abs()	5400	5440	5410	5430	5447	5425,4
exp()	4140	4201	4190	4226	4141	4179,6
log()	4300	4331	4230	4250	4200	4262,2
sqrt()	4453	4452	4456	4439	4460	4452
cos()	4070	4111	4241	4080	4122	4124,8
sin()	4100	4180	4142	4090	4081	4118,6
tan()	4135	4120	4110	4100	4121	4117,2
acos()	4332	4280	4270	4320	4260	4292,4
asin()	4320	4303	4300	4276	4311	4302
atan()	4160	4073	4090	4100	4061	4096,8

Tabela 3 – Tempos de execução em milissegundos para ASSEMBLY (In).

Para o Programa C, com procedures elaboradas em ASSEMBLY (Sep), foram obtidos os seguintes valores:

Tabela de Tempos de Execução em C/Assemby(Sep) (ms)						
	1	2	3	4	5	Média
abs()	24223	24070	24995	24575	27221	25016,8
exp()	26667	26119	26604	13599	26601	23918,0
log()	29221	33622	20953	28909	28722	28285,4
sqrt()	28166	32098	27272	30141	26687	28872,8
cos()	27123	26353	27006	24966	28253	26740,2
sin()	19657	25268	25842	24891	32746	25680,8
tan()	25660	16372	34252	27942	16013	24047,8
acos()	29654	26960	25512	27805	28648	27715,8
asin()	25989	26759	32552	31902	21016	27643,6
atan()	20039	20689	25225	24367	25780	23220,0

Tabela 4 – Tempos de execução em milissegundos para ASSEMBLY (Sep).

De acordo com os resultados explicitados na tabela de valores médios e gráfico correspondente abaixo:

Tabela de Tempos de Execução em Linguagem C (ms)				
	C	JAVA	C/Assembly (In)	C/Assembly (Sep)
abs()	3496,0	12356,0	5425,4	25016,8
exp()	2585,8	12279,0	4179,6	23918,0
log()	2574,0	13691,0	4262,2	28285,4
sqrt()	2768,6	13792,0	4452,0	28872,8
cos()	2578,0	17519,0	4124,8	26740,2
sin()	2604,0	17832,0	4118,6	25680,8
tan()	2662,2	17563,0	4117,2	24047,8
acos()	2904,2	10274,0	4292,4	27715,8
asin()	2956,0	10459,0	4302,0	27643,6
atan()	2812,2	17567,0	4096,8	23220,0

Tabela 5 – Tempos médios das linguagens.

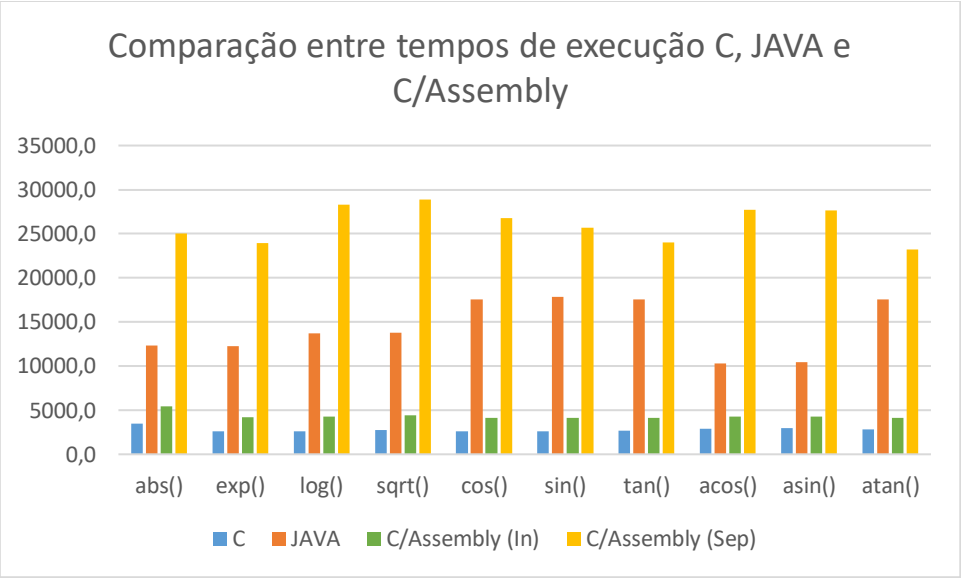

Gráfico 1 – Comparação de tempos médios de execução das linguagens observadas.

É possível perceber a regularidade dos intervalos. Em todas as funções, a ordem crescente de tempos foi: C < C/ASSEMBLY (In) < JAVA < C/ASSEMBLY (Sep). Os resultados das operações dentro das linguagens, no caso de C e C/ASSEMBLY (In), também foram bem homogêneos.

Os Ganhos considerados ente C e C/ASSEMBLY (In) foram como se pode observar na tabela:

Tabela de Ganhhos em execução	
	Ganho (C / C+Assembly (In))
abs()	0,64437645
exp()	0,61867164
log()	0,60391347
sqrt()	0,62187781
cos()	0,62500000
sin()	0,63225368
tan()	0,64660449
acos()	0,67659118
asin()	0,68712227
atan()	0,68643820
Média	0,64428492

Tabela 6 – Ganhos em Tempo de Execução.

Antes de elucidar sobre os resultados, é preciso conhecer um pouco mais as linguagens.

C é uma linguagem imperativa e procedural, para implementação de sistemas. Foi desenvolvida para ser compilada, fornecendo acesso de baixo nível à memória e baixos requerimentos do hardware. Também foi desenvolvida para ser uma linguagem de alto nível, para maior reaproveitamento do código. C foi útil para muitas aplicações que foram codificadas originalmente em Assembly. De forma sucinta, a linguagem C já utiliza internamente rotinas em Assembly e o compilador utiliza o código-fonte em C para gerar um executável em código nativo.

A linguagem JAVA, entretanto, tem seu código fonte compilada para um Bytecode, que, em vez de passar diretamente ao Sistema Operacional, é enviado para a JVM (Java Virtual Machine), que garante a leitura de um mesmo programa JAVA para diversos sistemas. É a JVM quem passa o programa para o sistema para ser lido pela máquina.

Assim, o JAVA possui uma etapa a mais, tornando o processo de compilação e execução maior que o realizado na linguagem C. Assim, era de se esperar que o tempo gasto pelo programa em C fosse bem menor que o necessário para o JAVA. Essa diferença se torna irrisória em máquinas com maior capacidade de memória RAM, pois, devido à própria evolução da linguagem, o JAVA é capaz de executar mais processos em paralelo (programação multi-thread) sendo, neste caso, necessário um maior intervalo de iteração para que o C volte a ser mais rápido.

Para o ASSEMBLY, percebe-se que o Inline resultou em tempos muito próximos (pouco maiores que C puro), o que era de se esperar, pois a linguagem C, por si só, é construída em ASSEMBLY, e totalmente Otimizada, sendo a melhor possibilidade dentro dos limites da máquina utilizada. Quanto ao ASSEMBLY em módulos separados, percebe-se que as medidas de tempo superaram os valores obtidos para a linguagem JAVA, o que pode indicar que os testes não foram adequados.

O primeiro ponto a ser considerado é que as funções matemáticas para o ASSEMBLY foram implementadas diretamente, enquanto para C e JAVA, foram utilizadas funções prontas. A justificativa para tal é que as funções “acos”, “asin”, e “atan” não são nativas de ASSEMBLY e, portanto, não seria possível analisá-las diretamente, sendo necessário misturar os métodos, o que tornaria o método de comparação injusto. Logo, A comparação foi específica entre as formas In e Sep, que, como é possível observar, resultou em grandezas bem distintas.

Outra consideração a ser feita é que o instante final de tempo foi medido somente ao final do loop. Sabendo que o sistema operacional tende a manipular e evitar loops, sendo a primeira forma de contornar isso, a contagem acumulada de instantes finais dentro de cada iteração. Há ainda a possibilidade de medir diretamente a frequência do clock da máquina, que toma o número de ciclos por operação, que resultaria em um valor mais verdadeira.

Portanto, a comparação do ganho da linguagem C foi feita apenas para o ASSEMBLY Sep e o resultado foi menor que 100%, indicando o o tempo gasto no puro C é menor.

7. CONCLUSÃO

Tem-se que o ganho é de cerca de 64%, o que torna mais prático utilizar operações diretamente em Linguagem C, dispensando a necessidade de elaborar um programa “misto” (utilizando funções matemáticas C e embarcando ASSEMBLY simultaneamente). Diante dos resultados apresentados e métodos de medição utilizados, C tem um tempo menor, sendo a opção mais considerável para cálculos matemáticos extensos utilizando notação de ponto flutuante.

8. REFERÊNCIAS

IBM. Disponível em: <https://www.ibm.com/developerworks/br/java/tutorials/j-introtojava1/index.html>. Acesso em 16 de Dezembro de 2017.

Tutorials Point. Disponível em: <https://www.tutorialspoint.com/c_standard_library/math_h.htm/>. Acesso em 10 de Dezembro de 2017

Microsoft. Disponível em: < https://msdn.microsoft.com/pt-br/library/windows/desktop/ms724390(v=vs.85).aspx>. Acesso em 13 de Dezembro de 2017.

Oracle. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html>. Acessado em 12 de Dazembro de 2017.