



**UNIVERSIDADE ESTADUAL DE MARINGÁ**

DEPARTAMENTO DE INFORMÁTICA  
IMPL. DE LINGUAGENS DE PROGRAMAÇÃO

PROF. DR. ANDERSON FAUSTINO DA SILVA



**Análise do Experimento em Otimização nas Compilações do  
GCC e CLANG**

**Alunos:**

Filipe da Silva Carvalho  
José Gabriel Junior

RA: 93306  
RA: 54011

**MARINGÁ  
2022**

# Sumário

<b>1. Introdução .....</b>	<b>3</b>
<b>2. Objetivos .....</b>	<b>4</b>
<b>3. Desenvolvimento .....</b>	<b>4</b>
<b>3.1. Compiladores e Otimização .....</b>	<b>4</b>
<b>3.2. Ambiente .....</b>	<b>5</b>
<b>3.3. Metodologia .....</b>	<b>6</b>
<b>4. Resultados .....</b>	<b>9</b>
<b>4.1. Algoritmo n-body .....</b>	<b>9</b>
<b>4.2. Algoritmo spectral-norm .....</b>	<b>10</b>
<b>4.3. Algoritmo mandelbrot .....</b>	<b>11</b>
<b>4.4. Algoritmo binary-trees .....</b>	<b>12</b>
<b>4.5. Algoritmo pidigits .....</b>	<b>13</b>
<b>5. Conclusão .....</b>	<b>14</b>
<b>Referências .....</b>	<b>15</b>

## 1. Introdução

Na programação existem diferentes tipos de linguagem de códigos fontes, esses por sua vez na maioria dos casos devem ser compilados para assim possibilitar a sua execução pela máquina. Uma dessas linguagem é a C, que dentro dos vários compiladores podemos citar o GCC (GNU) e Clang.

Na verdade, o GCC é um conjunto de compiladores definidos no projeto GNU, possibilitando não só a compilação da linguagem C, mas de várias outras. Já o Clang é um front-end que funciona em conjunto com a LLVM para efetuar a compilação. Nas duas situações são possíveis a definição de níveis de otimização, que tirando algumas variações, são os mesmos para os dois.

Esse experimento foi realizado a simulação com diferentes níveis de otimização para um conjunto de algoritmo, sendo apresentado no final os resultados em tempo de execução para cada nível de otimização e compilador adotado.

## 2. Objetivos

Nesse experimento será realizado uma análise nas compilações de códigos escritos em C com os compiladores GCC e CLANG, para diferentes níveis de otimização. Assim serão armazenados os tempos de 10 execuções para cada nível de otimização e no final será plotado o gráfico da média, comparando as compilações em GCC com as em CLANG.

## 3. Desenvolvimento

### 3.1. Compiladores e Otimização

No desenvolvimento de software temos um processo chamado de compilação, que visa a transformação de um determinado código fonte para um código de máquina, que posteriormente será possível a sua execução.

Para a linguagem C temos um potente compilador chamado **GCC**, que na verdade é conjunto de compiladores definidos no projeto GNU. Seu processo de desenvolvimento começou em 1987 por Richard Stallman, mas só foi concluído em 1999.

Ainda sobre a manipulação em código escritos em C, existe uma outra opção chamada **CLANG** que é um front-end para o processo de compilação. A sua principal diferença do GCC é que ele utiliza a LLVM como back-end, enquanto o GCC é um pacote completo.

No momento da compilação são possíveis definir vários parâmetros, como nome do código fonte, nome do arquivo gerado, bibliotecas utilizadas e também o nível de otimização, esse por sua vez pode melhorar o desempenho do programa influenciando diretamente no tamanho do código de máquina gerado. Abaixo segue uma tabela com os possíveis níveis de otimização, por padrão o GCC e CLANG não utilizam nenhuma otimização por padrão.

**Quadro 1 – Níveis de Otimização**

Níveis	Descrição
<b>-O0</b>	Sem otimização, este nível compila o mais rápido e gera o código mais depurável, podemos descrever como sendo o Default
<b>-O1</b>	Nível inicial de otimização, aplica apenas algumas, mas já começa a influenciar no tamanho do arquivo
<b>-O2</b>	Nível moderado de otimização que permite a maioria das otimizações.
<b>-O3</b>	Como o -O2, exceto que permite otimizações que demoram mais para serem executadas ou que podem gerar código maior, na tentativa de fazer o programa rodar mais rápido
<b>-Ofast</b>	Permite todas as otimizações de -O3 junto com outras otimizações agressivas que podem violar a estrita conformidade com os padrões da linguagem.
<b>-Os</b>	Como o -O2 somado a otimizações extras para reduzir o tamanho do código.
<b>-Oz</b>	Como o -Os, mas reduz ainda mais o tamanho do código
<b>-Og</b>	Como o -O1, mas esta opção pode desabilitar diferentes otimizações para melhorar a depuração.

**Fonte:** Documentação do GCC/GNU.

### 3.2. Ambiente

Os testes foram efetuados a partir de uma VM rodando em um PC com seguintes especificações:

**Quadro 2 – Configuração da máquina.**

<b>PROCESSADOR</b>	Ryzen 7 2700x 3.7GHz 16MB L3
<b>RAM</b>	16Gb (2 x 8Gb 2400Mhz)
<b>ROM</b>	SSD 420Gb
<b>SO</b>	Windows 10 Enterprise
<b>RAM-VM</b>	2Gb
<b>ROM-VM</b>	40Gb
<b>SO-VM</b>	Linux Mint 20 Ulyana (base Ubuntu 20.04 focal)

Para efetuar os testes foram necessários também a instalação de alguns pacotes, como o CLANG e o Samba. Referente a algumas bibliotecas para a linguagem C, também foram instalados os pacotes Build-Essential e libgmp3-dev. Abaixo seguem algumas versões dos serviços e pacotes utilizados:

**Quadro 3 - Versões**

<b>GCC</b>	9.4.0
<b>CLANG</b>	10.0.0
<b>Samba</b>	4.13.17
<b>Build-Essential</b>	12.8
<b>Bash</b>	5.0.17(1)
<b>LibGmp3-Dev</b>	2:6.2.0

### 3.3. Metodologia

Inicialmente foram escolhidos 5 algoritmos definidos no site especificado pelo professor. Abaixo segue uma relação dos algoritmos selecionados:

**Quadro 4 - Algoritmos**

<b>Algoritmo</b>	<b>Versão</b>	<b>Link</b>
n-body	1	<a href="https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gcc-1.html">https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gcc-1.html</a>
spectral-norm	1	<a href="https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/spectralnorm-gcc-1.html">https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/spectralnorm-gcc-1.html</a>
mandelbrot	1	<a href="https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/mandelbrot-gcc-1.html">https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/mandelbrot-gcc-1.html</a>
binary-trees	1	<a href="https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-gcc-1.html">https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/binarytrees-gcc-1.html</a>
pidigits	1	<a href="https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/pidigits-gcc-1.html">https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/pidigits-gcc-1.html</a>

Após a escolha dos algoritmos, foi utilizado uma VM com o SO Mint para efetuar as simulações. Também foi utilizado o Samba para facilitar a movimentação de arquivos.

Para realizar as simulações foi utilizado um script em Bash para automatizar o processo.

**Figura 1** – Script **executa.sh**.

```
#!/bin/bash
TIMEFORMAT='Tempos[ Real: %3R  User: %3U  Sys: %3S ]'
for comp in gcc clang;
do
    for i in 0 1 2 3;
    do
        $comp -O$i $1 $3 $4;

        for j in $(seq 10);
        do
            echo "-----";
            echo "Compilacao: $comp -O$i $1 $3 $4  Parametro: $2  Execucao: $j";
            echo "Tamanho executavel:" $(du -h a.out)
            time ./a.out $2 > /etc/null
            echo
        done
    done
done
```

Esse script funcionara da seguinte forma:

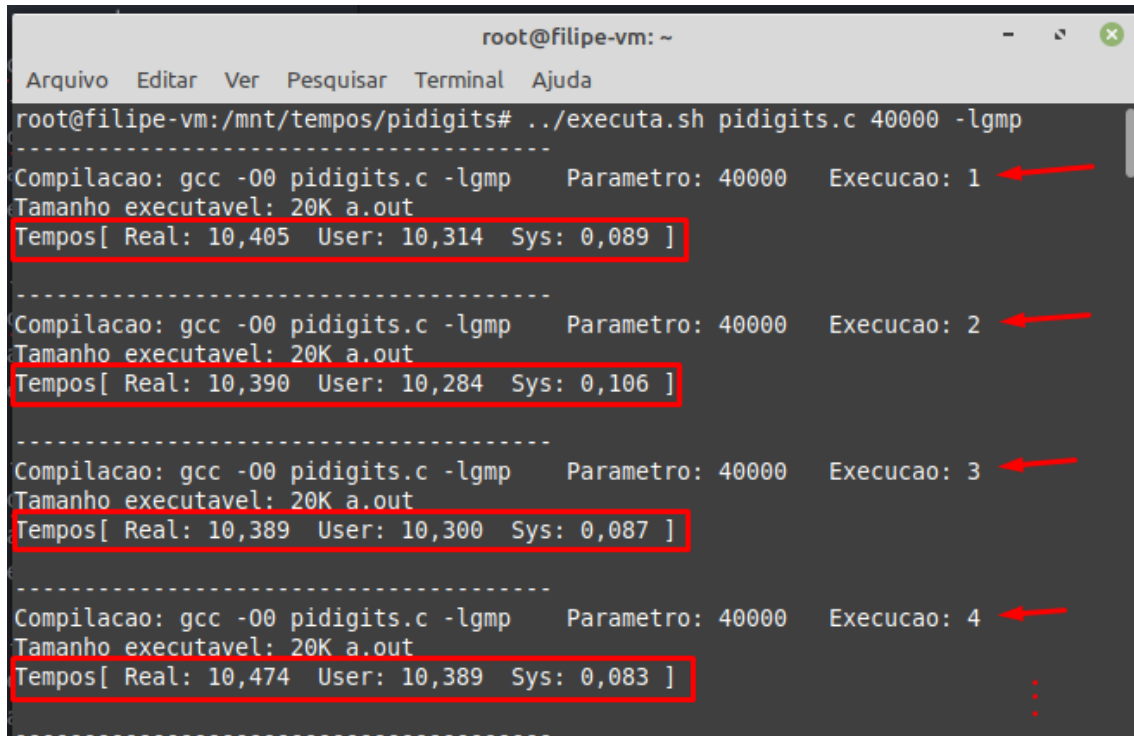
- Inicialmente é modificado a variável de ambiente TIMEFORMAT para formatar a exibição das informações de tempo exibidas pelo comando time;
- O primeiro laço será rodado por duas vezes, umas para compilação com o GCC e outra para o CLANG;
- Nesse segundo laço, será rodado por 4 vezes, referente as otimizações utilizadas nesse experimento (O0, O1, O2 e O3);
- Após realizado a compilação, entra o terceiro laço que se repetira por 10 vezes, efetuando a execução do programa e apresentado o log implementado, contendo tempo de execução e tamanho do executável.

Para utilizar o script a sintaxe ficou da seguinte forma:

```
../executa.sh arquivo.c paramExecucao paramComp1 paramComp2
```

Foi adicionado apenas dois parâmetros de compilação pois conforme os algoritmos escolhidos o máximo de bibliotecas utilizadas eram de duas simultâneas.

Figura 2 – Exemplo de log gerado na execução do script.



```
root@filipe-vm: ~
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
root@filipe-vm:/mnt/tempos/pidigits# ../executa.sh pidigits.c 40000 -lgmp
-----
Compilacao: gcc -O0 pidigits.c -lgmp    Parametro: 40000    Execucao: 1
Tamanho executavel: 20K a.out
Tempos[ Real: 10,405  User: 10,314  Sys: 0,089 ]
-----
Compilacao: gcc -O0 pidigits.c -lgmp    Parametro: 40000    Execucao: 2
Tamanho executavel: 20K a.out
Tempos[ Real: 10,390  User: 10,284  Sys: 0,106 ]
-----
Compilacao: gcc -O0 pidigits.c -lgmp    Parametro: 40000    Execucao: 3
Tamanho executavel: 20K a.out
Tempos[ Real: 10,389  User: 10,300  Sys: 0,087 ]
-----
Compilacao: gcc -O0 pidigits.c -lgmp    Parametro: 40000    Execucao: 4
Tamanho executavel: 20K a.out
Tempos[ Real: 10,474  User: 10,389  Sys: 0,083 ]
-----
...
```

Assim esse Script foi executado para cada algoritmo escolhido, os parâmetros do programa foram utilizados os próprios citados no site de referência, conforme comando abaixo:

```
../executa.sh nbody.c 50000000 -lm
../executa.sh spectral-norm.c 5500 -lm
../executa.sh mandelbrot.c 16000 -lm -pthread
../executa.sh binary_trees.c 21 -lm
../executa.sh pidigits.c 40000 -lgmp
```

Após essas execuções foram anotados todos os valores de tempo e adicionado a planilhas, gerando médias e gráficos, conforme será apresentado na próxima sessão.



## 4. Resultados

### 4.1. Algoritmo n-body

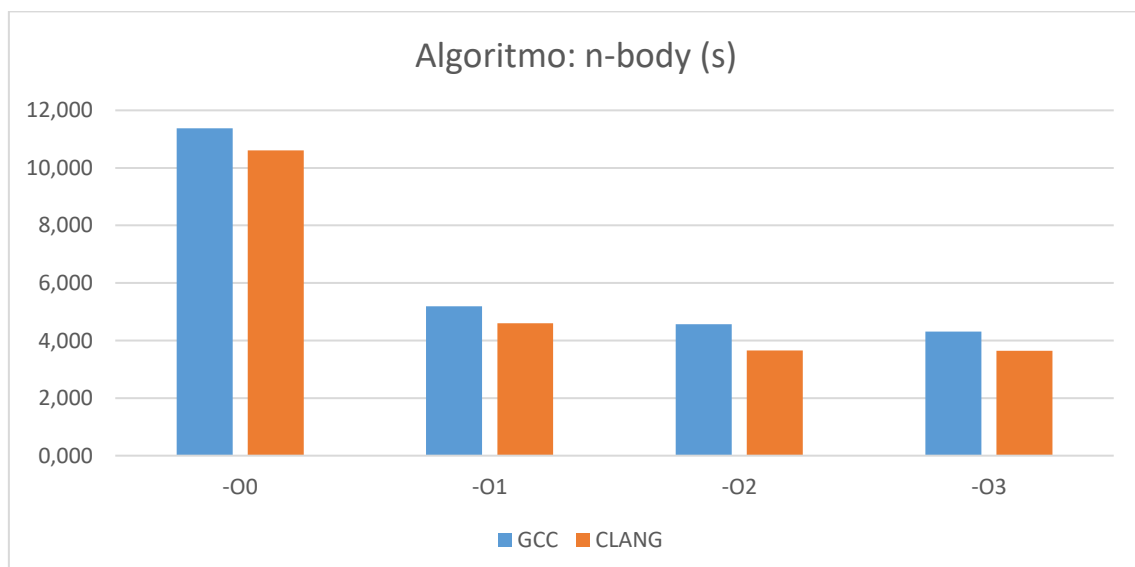
**Quadro 5** – Tempos de execução em segundos do algoritmo n-body

n	GCC				CLANG			
	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3
1	11,281	5,179	4,567	4,314	10,616	4,602	3,646	3,626
2	11,455	5,163	4,594	4,302	10,592	4,596	3,647	3,662
3	11,353	5,178	4,547	4,316	10,562	4,621	3,705	3,659
4	11,484	5,252	4,568	4,367	10,554	4,565	3,677	3,631
5	11,270	5,191	4,547	4,291	10,709	4,579	3,630	3,670
6	11,374	5,185	4,633	4,329	10,617	4,593	3,653	3,652
7	11,441	5,249	4,542	4,339	10,599	4,608	3,635	3,656
8	11,372	5,442	4,569	4,306	10,780	4,669	3,683	3,631
9	11,335	5,185	4,554	4,330	10,630	4,605	3,660	3,629
10	11,387	5,266	4,561	4,302	10,531	4,634	3,666	3,633
<b>Médias</b>	<b>11,373</b>	<b>5,188</b>	<b>4,564</b>	<b>4,315</b>	<b>10,608</b>	<b>4,604</b>	<b>3,657</b>	<b>3,643</b>

**Quadro 6** – Tamanho dos executáveis do algoritmo n-body

Tamanhos dos Executáveis (k)		
Otimização	GCC	CLANG
-O0	20	20
-O1	20	20
-O2	20	20
-O3	20	24

**Gráfico 1** – Comparação entre as médias das execuções do algoritmo n-body



## 4.2. Algoritmo spectral-norm

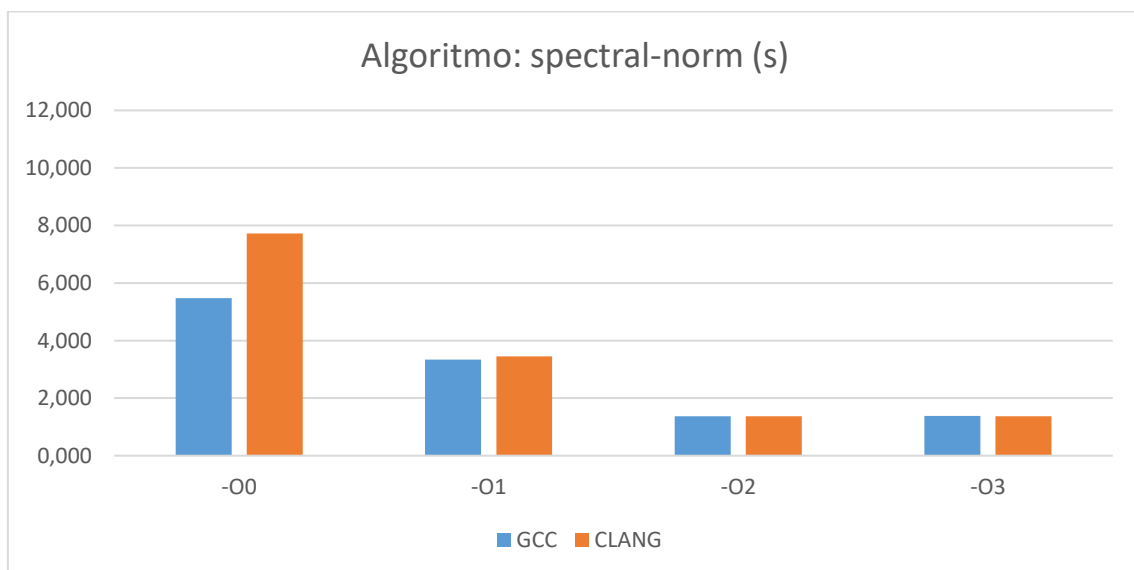
**Quadro 7** – Tempos de execução em segundos do algoritmo spectral-norm.

n	GCC				CLANG			
	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3
1	5,453	3,340	1,376	1,377	7,656	3,520	1,400	1,375
2	5,475	3,314	1,381	1,366	7,808	3,454	1,375	1,368
3	5,469	3,289	1,372	1,373	7,794	3,418	1,369	1,369
4	5,477	3,368	1,370	1,363	7,621	3,486	1,374	1,376
5	5,493	3,331	1,398	1,375	7,761	3,451	1,371	1,379
6	5,469	3,329	1,374	1,381	7,535	3,441	1,370	1,370
7	5,470	3,359	1,370	1,381	7,595	3,451	1,372	1,366
8	5,547	3,316	1,368	1,431	7,743	3,435	1,379	1,364
9	5,464	3,395	1,371	1,385	7,793	3,455	1,376	1,381
10	5,497	3,399	1,365	1,367	7,712	3,476	1,375	1,377
<b>Médias</b>	<b>5,473</b>	<b>3,336</b>	<b>1,372</b>	<b>1,376</b>	<b>7,728</b>	<b>3,453</b>	<b>1,375</b>	<b>1,373</b>

**Quadro 8** – Tamanho dos executáveis do algoritmo spectral-norm.

Tamanhos dos Executáveis (k)		
Otimização	GCC	CLANG
-O0	20	20
-O1	20	20
-O2	20	20
-O3	20	20

**Gráfico 2** – Comparação entre as médias das execuções do algoritmo spectral-norm.



### 4.3. Algoritmo mandelbrot

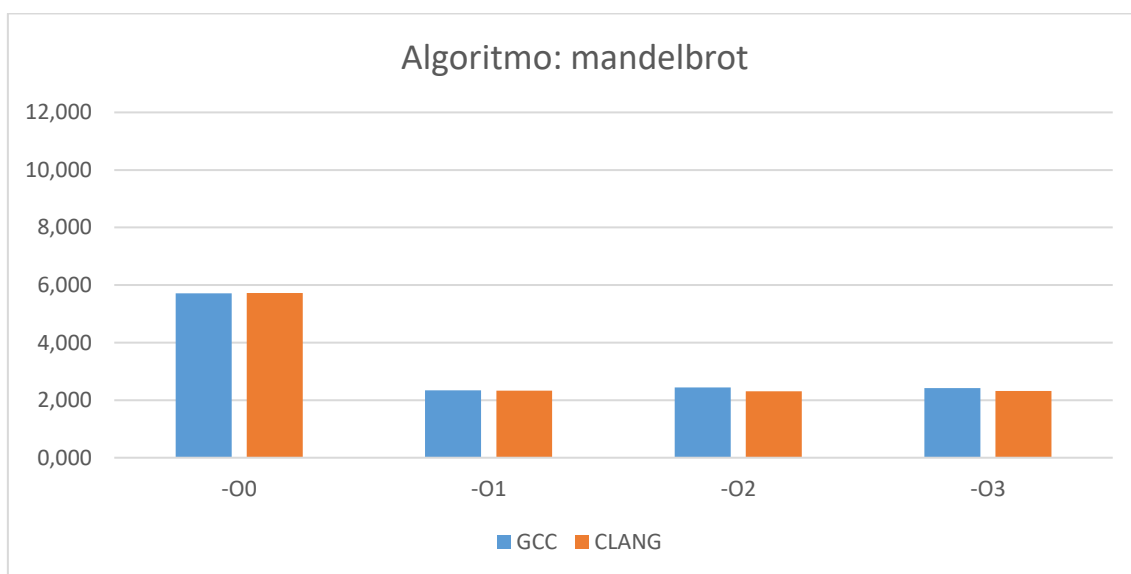
**Quadro 9** – Tempos de execução em segundos do algoritmo mandelbrot.

n	GCC				CLANG			
	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3
1	5,787	2,338	2,442	2,427	5,689	2,331	2,323	2,318
2	5,716	2,320	2,412	2,446	5,693	2,317	2,313	2,374
3	5,645	2,311	2,444	2,440	5,729	2,332	2,298	2,350
4	5,728	2,353	2,446	2,421	5,725	2,348	2,331	2,326
5	5,717	2,332	2,428	2,433	5,711	2,330	2,300	2,342
6	5,714	2,338	2,413	2,418	5,728	2,336	2,308	2,285
7	5,692	2,354	2,432	2,414	5,759	2,331	2,306	2,299
8	5,732	2,343	2,438	2,420	5,746	2,321	2,314	2,323
9	5,698	2,346	2,439	2,415	5,741	2,317	2,324	2,304
10	5,727	2,366	2,442	2,410	5,738	2,334	2,330	2,320
<b>Médias</b>	<b>5,717</b>	<b>2,341</b>	<b>2,439</b>	<b>2,421</b>	<b>5,729</b>	<b>2,331</b>	<b>2,314</b>	<b>2,322</b>

**Quadro 10** – Tamanho dos executáveis do algoritmo mandelbrot.

Tamanhos dos Executáveis (k)		
Otimização	GCC	CLANG
-O0	20	20
-O1	20	20
-O2	20	20
-O3	20	20

**Gráfico 3** – Comparação entre as médias das execuções do algoritmo mandelbrot.



#### 4.4. Algoritmo binary-trees

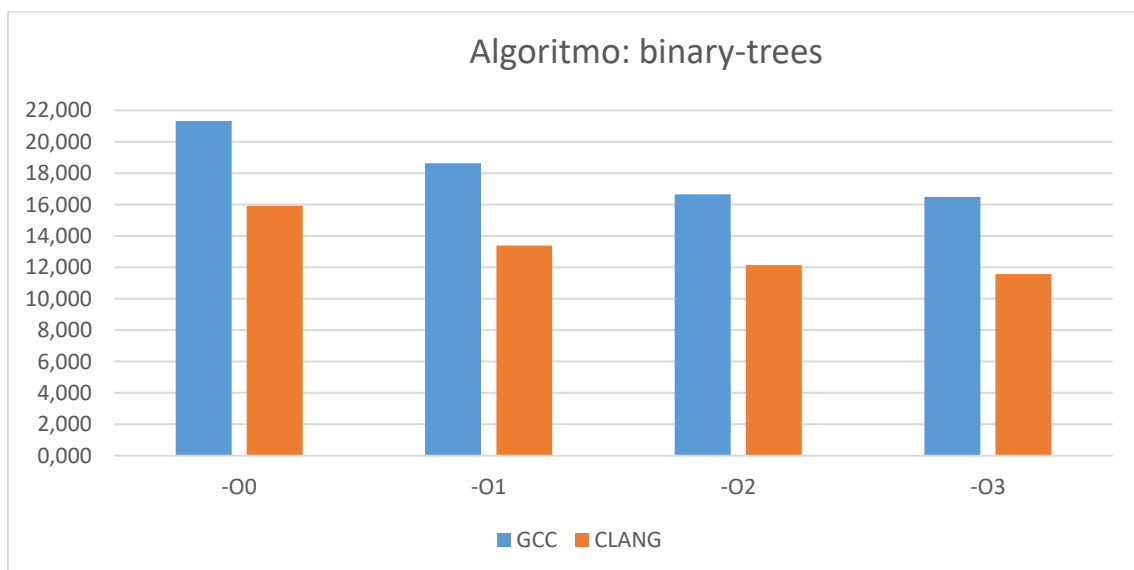
**Quadro 11** – Tempos de execução em segundos do algoritmo binary-trees.

n	GCC				CLANG			
	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3
1	21,245	18,716	17,005	16,495	15,969	13,417	12,075	11,591
2	21,433	18,731	16,793	16,468	16,018	13,760	12,202	11,624
3	21,074	18,546	16,609	16,329	15,830	13,407	11,979	11,517
4	21,741	18,338	16,459	17,073	16,039	13,286	12,102	11,392
5	21,321	18,401	16,867	17,068	15,813	13,303	12,235	11,403
6	21,332	18,915	16,983	16,770	15,937	13,220	12,258	11,525
7	21,499	18,479	16,696	17,148	15,895	13,379	12,307	11,563
8	21,362	18,763	16,613	16,230	15,828	13,390	12,271	11,624
9	21,276	18,881	16,395	16,404	15,996	13,405	12,110	11,640
10	21,253	18,490	16,321	16,232	15,692	13,362	12,036	11,753
<b>Médias</b>	<b>21,327</b>	<b>18,631</b>	<b>16,655</b>	<b>16,482</b>	<b>15,916</b>	<b>13,385</b>	<b>12,156</b>	<b>11,577</b>

**Quadro 12** – Tamanho dos executáveis do algoritmo binary-trees.

Tamanhos dos Executáveis (k)		
Otimização	GCC	CLANG
-O0	20	20
-O1	20	20
-O2	20	20
-O3	24	20

**Gráfico 4** – Comparação entre as médias das execuções do algoritmo binary-trees.



#### 4.5. Algoritmo pidigits

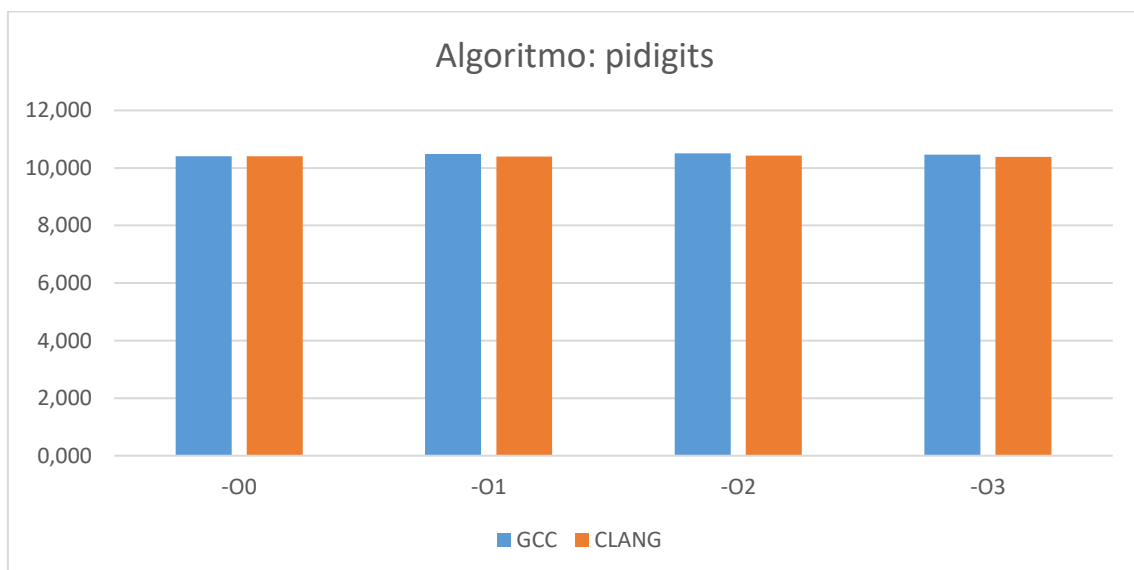
**Quadro 13** – Tempos de execução em segundos do algoritmo pidigits.

n	GCC				CLANG			
	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3
1	10,405	10,477	10,574	10,383	10,428	10,390	10,402	10,462
2	10,390	10,545	10,516	10,491	10,374	10,538	10,399	10,368
3	10,389	10,695	10,395	10,506	10,368	10,352	10,556	10,355
4	10,474	10,441	10,535	10,483	10,447	10,449	10,521	10,365
5	10,358	10,489	10,423	10,520	10,389	10,491	10,437	10,313
6	10,334	10,690	10,543	10,308	10,351	10,394	10,303	10,461
7	10,421	10,393	10,430	10,436	10,524	10,497	10,426	10,403
8	10,440	10,421	10,444	10,442	10,446	10,319	10,417	10,370
9	10,524	10,429	10,496	10,495	10,538	10,374	10,552	10,483
10	10,571	10,657	10,546	10,428	10,362	10,396	10,431	10,471
<b>Médias</b>	<b>10,413</b>	<b>10,483</b>	<b>10,506</b>	<b>10,463</b>	<b>10,409</b>	<b>10,395</b>	<b>10,429</b>	<b>10,387</b>

**Quadro 14** – Tamanho dos executáveis do algoritmo pidigits.

Tamanhos dos Executáveis (k)		
Otimização	GCC	CLANG
-O0	20	20
-O1	20	20
-O2	20	20
-O3	20	20

**Gráfico 5** – Comparação entre as médias das execuções do algoritmo pidigits.



## 5. Conclusão

Após realizarmos os testes, utilizando os 5 algoritmos e executarmos por 10 vezes em cada nível de otimização, revezando entre as compilações GCC e CLANG, notamos que na maioria dos casos quanto maior o nível da otimização, mais rápido será o processamento. Mas aconteceu de alguns casos que não houveram melhora do nível -O2 para -O3, acreditamos que isso ocorreu pois já havia atingido a otimização máxima possível.

Também foi notado uma diferença nas execuções com nível de otimização default (-O0), alguns foram mais rápidos em GCC enquanto outros foram mais rápido com CLANG.

Essas variações de tempo foram constatadas em quase todos os algoritmos, exceto o **pidigits**, que acredito não ser possível sua otimização, pois os tempos se mantiveram independentemente do nível de otimização.

No quesito tamanho dos executáveis, houveram apenas dois casos de variação, possivelmente isso se deve a serem códigos pequenos, talvez com códigos maiores isso seria mais perceptível.

## Referências

Which programming language is fastest? Disponível em: <<https://benchmarksgame-team.pages.debian.net/benchmarksgame/>>. Acesso em: 22 abr. 2022.

GCC online documentation - GNU Project. Disponível em: <<https://gcc.gnu.org/onlinedocs/>>. Acesso em: 22 abr. 2022.