Universidade de São Paulo
Instituto de Física de São Carlos

Prática 1: Complexidade

Stefan Taiguara Couperus Leal 10414866

02 de Agosto de 2019

Contents

1 Programas sem a otimização

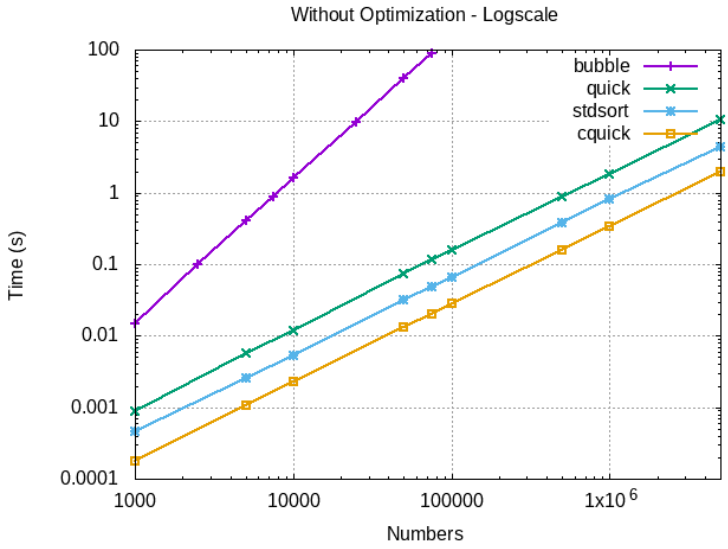

Figure 1:
Comparação de tempo de execução para cada método de sort sem a utilização de otimização

2 Programas com a otimização

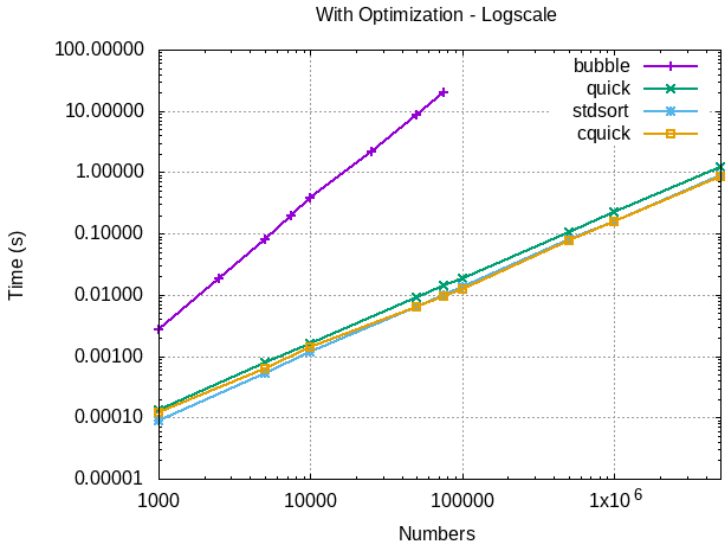

Figure 2:
Comparação de tempo de execução para cada método de sort com a utilização de otimização

3 Otimizações vistas individualmente

3.1 Bubble Sort

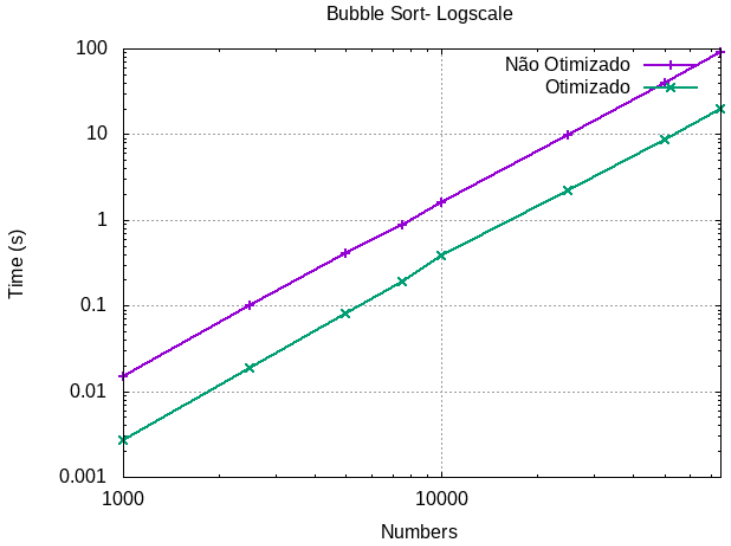

Figure 3:
Comparação de tempo de execução com e sem a otimização

bubble.cpp	Nao otimizado	Otimizado	Diferenca
N	t(s)	t(s)	%
1000	0.0150798	0.0026974	559.04945503077
2500	0.100568	0.0188995	532.119897351782
5000	0.403481	0.0815709	494.638406588624
7500	0.874547	0.192306	454.768441962289
10000	1.60068	0.38566	415.049525488772
25000	9.89544	2.17505	454.952299947128
50000	40.173	8.75064	459.086421107485
75000	90.0311	19.822	454.197860962567

Table 1: Comparação entre o uso do bubble sort com e sem a otimização

$$M = (480 \pm 80)\%$$

3.2 Quick Sort

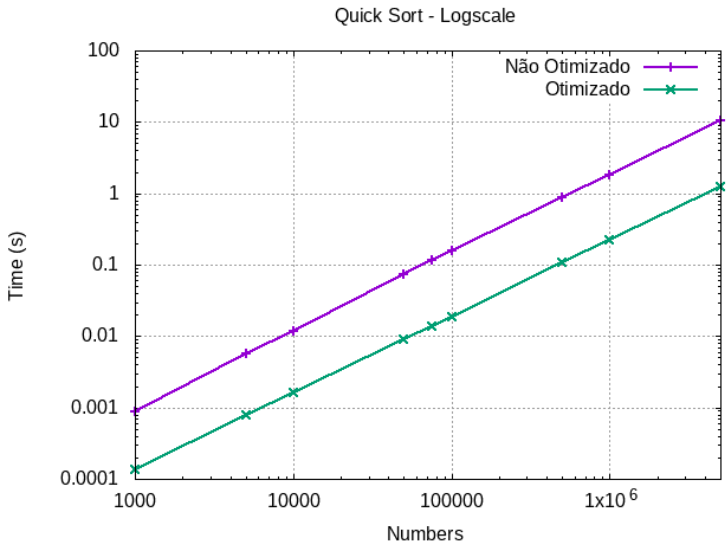

Figure 4:
Comparação de tempo de execução com e sem a otimização

quick.cpp	Não Otimizado	Otimizado	Diferença
N	t(s)	t(s)	%
1000	0.00088135	0.0001349	653.335804299481
5000	0.005691	0.0007757	733.659920072193
10000	0.012008	0.00159235	754.105567243382
50000	0.0728277	0.00894055	814.577402956194
75000	0.117504	0.0139109	844.690135073935
100000	0.159358	0.0188964	843.324654431532
500000	0.871283	0.107163	813.044614279182
1000000	1.85177	0.225075	822.734644007553
5000000	10.6437	1.24485	855.01867694903

Table 2: Comparação entre o uso do quick sort com e sem a otimização

$$M = (800 \pm 70)\%$$

3.3 cQuick Sort

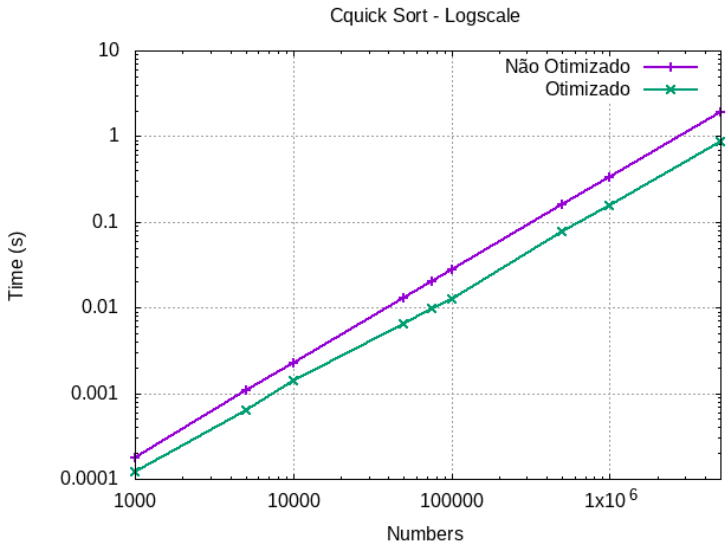

Figure 5:
Comparação de tempo de execução com e sem a otimização

cquick.cpp	Não	Otimizado	Diferença
N	t(s)	t(s)	%
1000	0.0001796	0.00012165	147.636662556515
5000	0.00108565	0.0006263	173.343445633083
10000	0.00226915	0.0013973	162.395333858155
50000	0.0131055	0.006372	205.673258003766
75000	0.0204937	0.00963175	212.77234147481
100000	0.0278923	0.0124632	223.797259130881
500000	0.16071	0.0776592	206.942641696026
1000000	0.336523	0.156496	215.036167058583
5000000	1.94205	0.85818	226.298678598895

Table 3: Comparação entre o uso do cquick com e sem a otimização

$$M = (200 \pm 30)\%$$

3.4 Stdsort

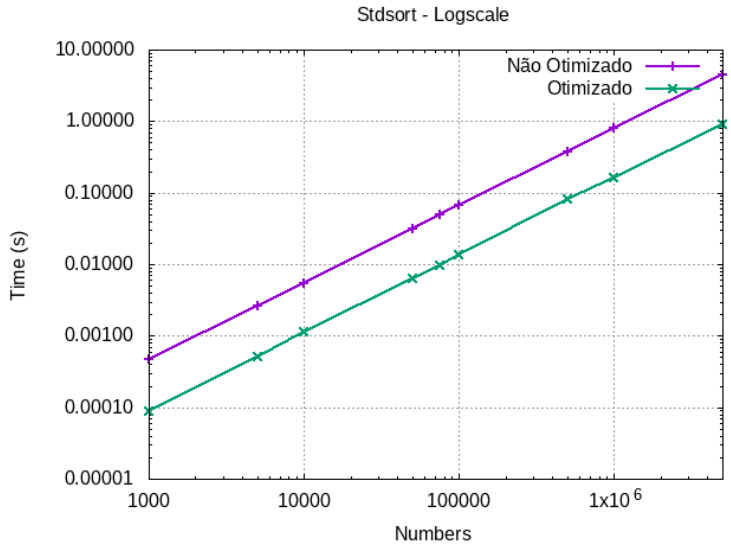

Figure 6:
Comparação de tempo de execução com e sem a otimização

stdsort.cpp	Não Otimizado	Otimizado	Diferença
N	t(s)	t(s)	%
1000	0.0004638	8.8E-005	527.045454545454
5000	0.0025799	0.0005184	497.665895061728
10000	0.0053947	0.00114835	469.778377672312
50000	0.0313781	0.00631355	496.996143215782
75000	0.0488633	0.00975975	500.661389892159
100000	0.0664287	0.0133985	495.792066276076
500000	0.379537	0.080877	469.276803046602
1000000	0.809606	0.160195	505.387808608259
5000000	4.44977	0.888075	501.057906145314

Table 4:

$$M = (500 \pm 20)\%$$

4 Conclusão

Qual a complexidade de cada programa?

Algoritmo	Complexidade
bubble.cpp	O^2
quick.cpp	$O(n \log n)$
cquick.cpp	$O(n \log n)$
stdsort.cpp	$O(n \log n)$

Table 5:

Essa complexidade se reflete nos resultados?

O que você pode concluir sobre a importância da complexidade do programa para seu desempenho?

Quanto maior for a complexidade pior o desempenho que vai ser apresentado pelo programa, portanto sempre deve se ter em mente a complexidade do algoritmo utilizado.

O que você pode concluir sobre a importância de otimizações no código?

Algoritmo	Diferença (%)
bubble.cpp	$(480 \pm 80)\%$
quick.cpp	$(800 \pm 70)\%$
cquick.cpp	$(500 \pm 20)\%$
stdsort.cpp	$(200 \pm 30)\%$

Table 6: Tabela com as diferenças de performance quando há o uso do -O2

Como pode ser visto pela tabela 6 a otimização apresenta uma mudança drástica de velocidade de processamento.

Os diversos codigo são afetados da mesma forma pelas otimizações?

Como pode ser visto pela tabela acima percebe-se que para o stdsort.cpp a diferença foi de apenas 200%, mas para o quick.cpp a diferença foi de 800%, sendo a média para todos os algoritmos 500%. Conclui-se que a otimização difere para cada tipo de algoritmo.

Como você compara o impacto da complexidade e o impacto das otimizações?

Para 75000 elementos, quantas vezes mais rápido é o código de ordenação do std::sort em relação bubble sort?

o std::sort é 2031 vezes mais rápido do que o bubble sort para 75000 elementos.

Você consegue explicar a diferença entre os tempos do stdsort.cpp e do cquick.cpp otimizados?

stdsort.cpp		cquick.cpp		Diferença
N	t(s)	N	t(s)	t(s)
1000	8.8E-005	1000	0.00012165	-3.365E-005
5000	0.0005184	5000	0.0006263	-0.0001079
10000	0.00114835	10000	0.0013973	-0.00024895
50000	0.00631355	50000	0.006372	-5.84499999999998E-005
75000	0.00975975	75000	0.00963175	0.00012800000000000001
100000	0.0133985	100000	0.0124632	0.0009353
500000	0.080877	500000	0.0776592	0.00321779999999999
1000000	0.160195	1000000	0.156496	0.00369900000000001
5000000	0.888075	5000000	0.85818	

Table 7:

Portanto conclui-se que não há uma diferença significativa de tempo entre os dois métodos utilizados, dado que a diferença média deu:

$$M = (0.000 \pm 0.010)s$$

Comparando quick.cpp e stdsort.cpp, você recomenda o uso de algoritmos da STL?

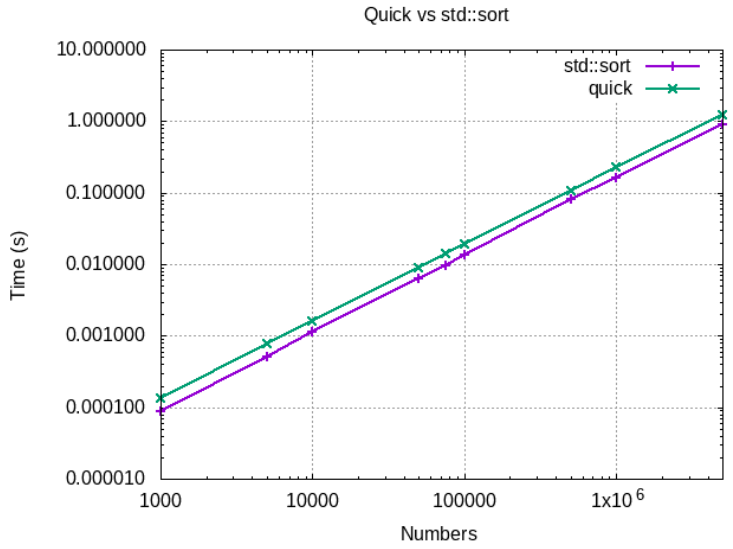

Figure 7:
Gráfico de Quick sort vs std::sort

5 Bibliografia

References

[1] Laboratório de Física II - Livro de Práticas;

[2] Tipler, P. A., Mosca, G.. *Física para Cientistas e Engenheiros.* Vol.1. LTC

[3] Nussenzveig, H. M.. *Curso de Física Básica.* Vol. 2. Editora Blucher