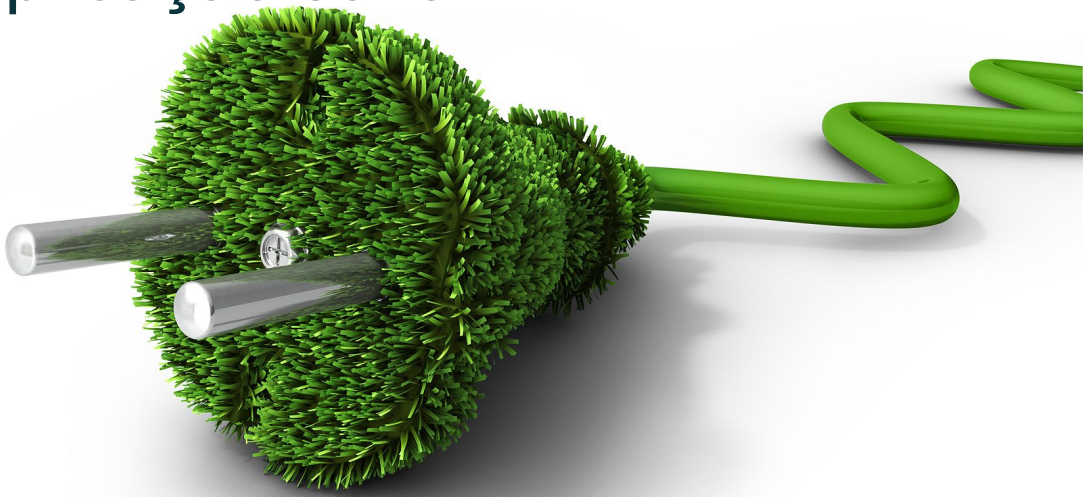
CaparicaPost

Análise e Otimização de Consumo de Energia de uma Aplicação Java

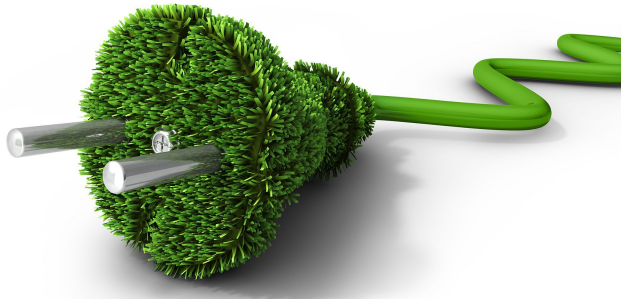


Universidade do Minho
Escola de Engenharia

Trabalho realizado por:
A71604 - Diogo Couto
A67738 - Gil Gonçalves
A67751 - Pedro Silva

Conteúdos

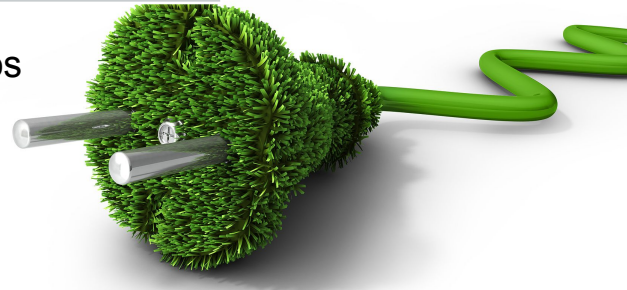
- Introdução
- Metodologia de Teste
- Primeira fase - Otimizações
- Segunda fase - Programação Verde
- Conclusões



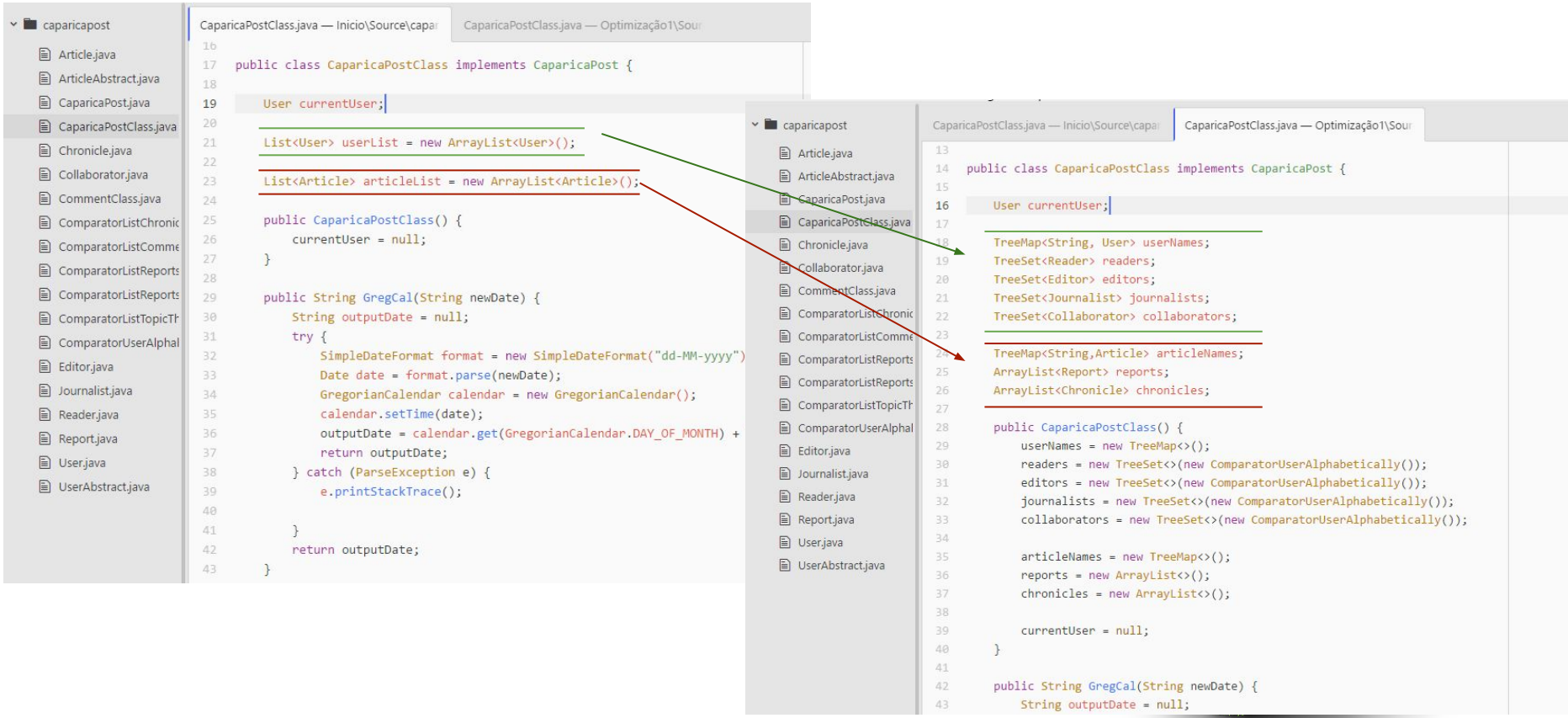
Metodologia de Teste

	PC - 1	PC - 2
Processador	Intel® Core™ i7 - 4750HQ 2,0GHz	Intel® Core™ i5 Dual Core - 3210M 2,5 GHZ
Placa Gráfica	GeForce950m 4GB	GeForce640m 2GB
Memória RAM	8.00GB	6.00GB

Tabela 1 - Specs dos Computadores Utilizados



Primeira Fase - Otimizações



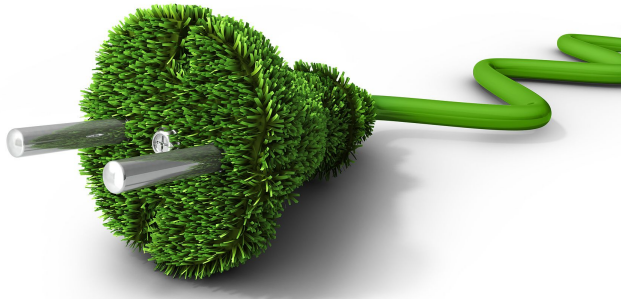
Primeira Fase - Resultados

	Inicio	Otimizado	Ganho%
GPU (J)	0.09271	0.0792	1.1705
CPU (J)	27.4065	17.8698	1.5336
Package (J)	46.2936	31.0919	1.4889
Time (Seconds)	1.9608	1.4439	1.3579

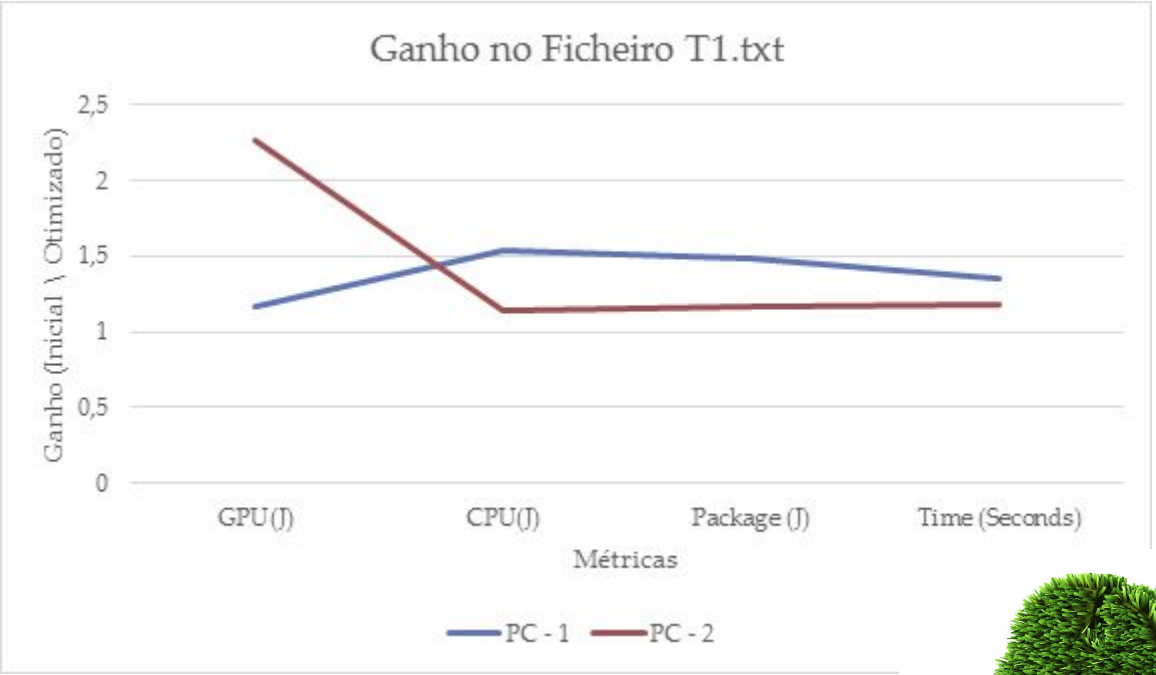
Tabela 2 - T1.txt - PC 1

	Inicio	Otimizado	Ganho%
GPU (J)	1.42538	0.6301	2.2621
CPU (J)	38.8563	33.9977	1.1429
Package (J)	52.6736	45.0657	1.1688
Time (Seconds)	3.2022	2.7017	1.1852

Tabela 3- T1.txt - PC 2



Primeira Fase - Resultados



Primeira Fase - Resultados

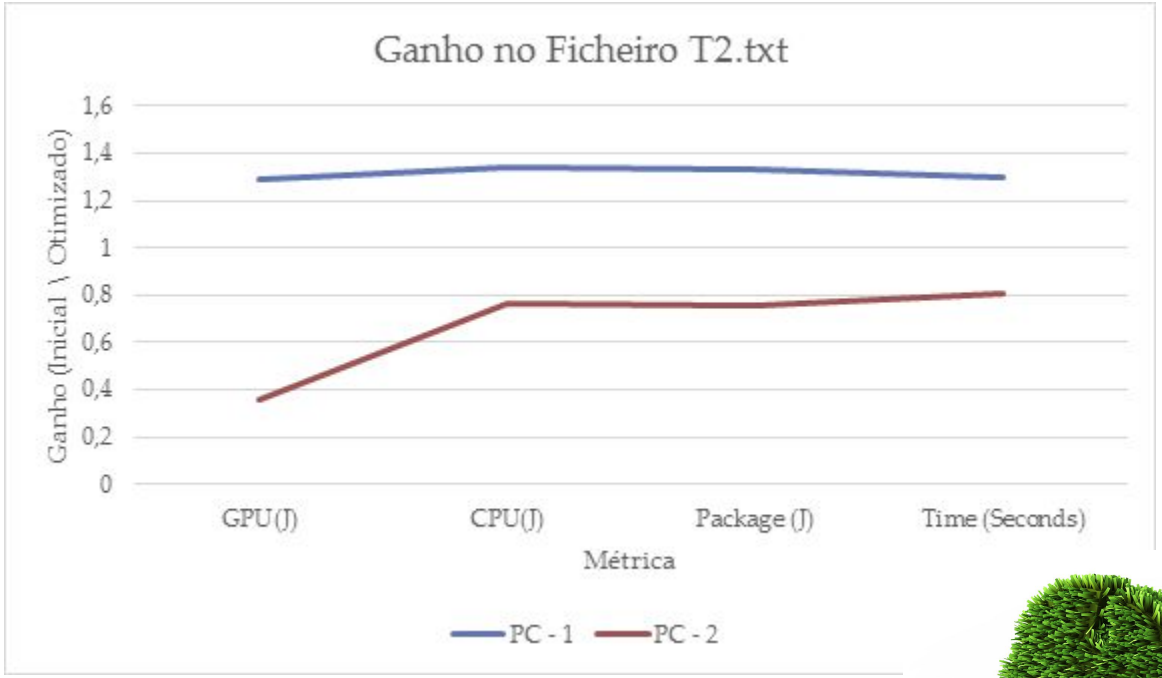
	Inicio	Otimizado	Ganho%
GPU (J)	0.074414	0.0578	1.2874
CPU (J)	13.7276	10.2255	1.3424
Package (J)	21.7071	16.2525	1.3356
Time (Seconds)	0.7928	0.6094	1.3356

Tabela 4 - T2.txt - PC 1

	Inicio	Otimizado	Ganho%
GPU (J)	0.3957	1.1026	0.3588
CPU (J)	17.5818	22.8754	0.7685
Package (J)	23.2189	30.7739	0.7545
Time (Seconds)	1.3721	1.6972	0.8084

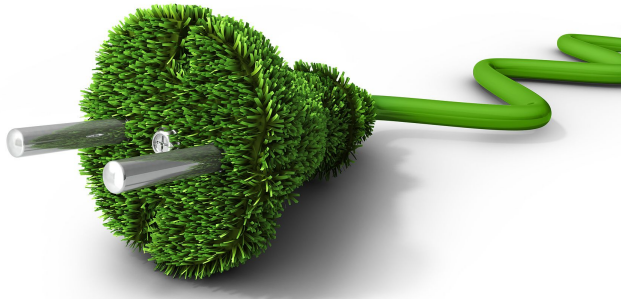
Tabela 5 - T2.txt - PC 2

Primeira Fase - Resultados



Segunda fase - Programação Verde

- Três estratégias utilizadas:
  - Trocar operador “+” por StringBuilder
  - Trocar Switch Statements por Classe Comando
  - Substituição das estruturas de dados



StringBuilder porquê usar?

- É mais rápido que o operador “+”;

- O compilador o que faz quando é usado o operador “+” é chamar o *StringBuilder* ;

- Aloca muito menos espaço de memória;

```java
public void PrintComments() {
    StringBuilder sb= new StringBuilder();
    System.out.println("Chronicle comments:");
    Collections.sort(commentList, new ComparatorListComments());
    for (int i = 0; i < commentList.size(); i++) {
        System.out.println(commentList.get(i).GetAuthor() + "; " + commentList.get(i).GetText());
    }
}
```

```java
public void PrintComments() {
    StringBuilder sb= new StringBuilder();
    System.out.println("Chronicle comments:");
    Collections.sort(commentList, new ComparatorListComments());
    for (int i = 0; i < commentList.size(); i++) {
        sb.setLength(0);
        sb.append(commentList.get(i).GetAuthor());
        sb.append("; ");
        sb.append(commentList.get(i).GetText());
        System.out.println(sb.toString());
    }
}
```

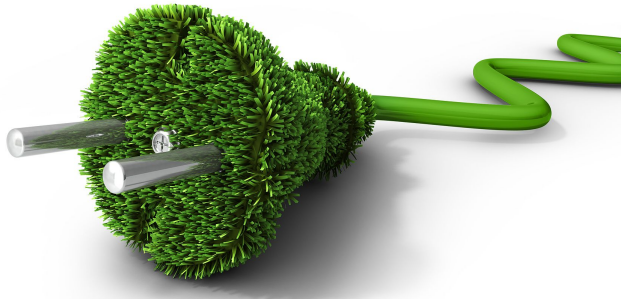
StringBuilder - Resultados

	Inicio	Otimizado	Ganho%
GPU (J)	0.0792	0.0511	1.5494
CPU (J)	17.8698	13.3211	1.3414
Package (J)	31.0919	27.8751	1.1153
Time (Seconds)	1.4439	1.7998	0.802247

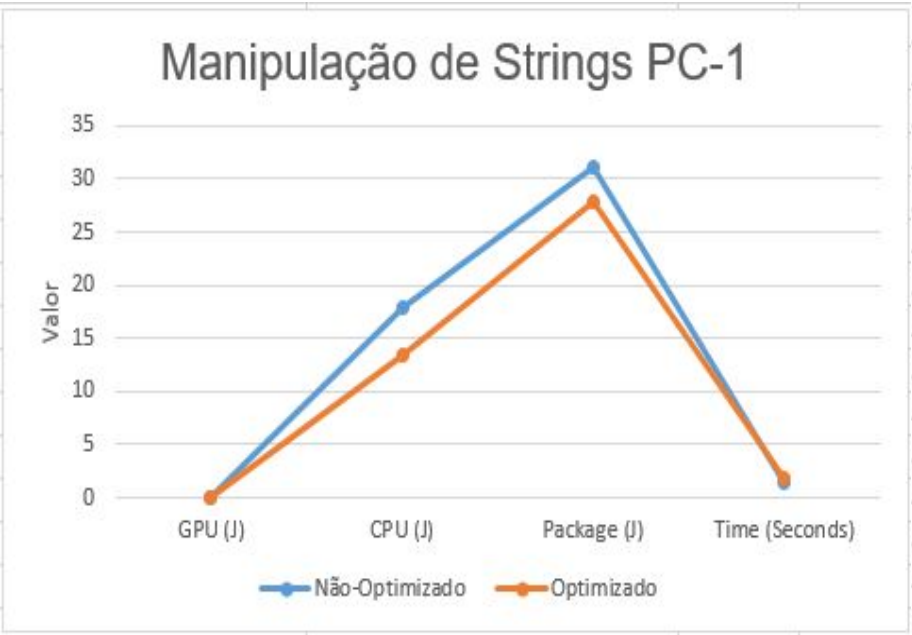
Tabela 3 - T1.txt - PC 1

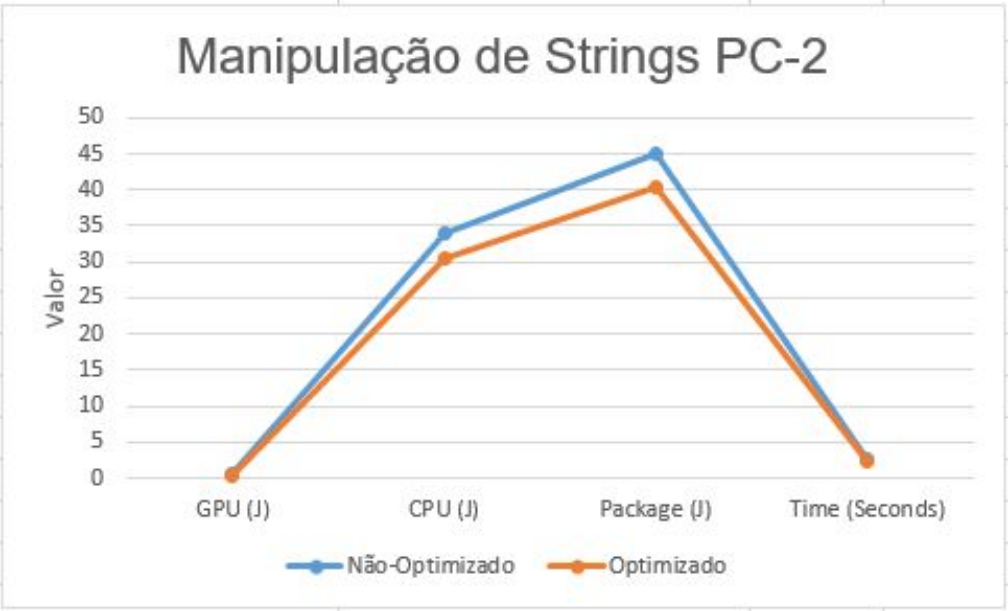
	Inicio	Otimizado	Ganho%
GPU (J)	0.6301	0.423004	1.4895
CPU (J)	33.9977	30.4897	1.1150
Package (J)	45.0657	40.4313	1.1146
Time (Seconds)	2.7017	2.4403	1.1071

Tabela 4 - T1.txt - PC 2



StringBuilder - Resultados





StringBuilder - Resultados

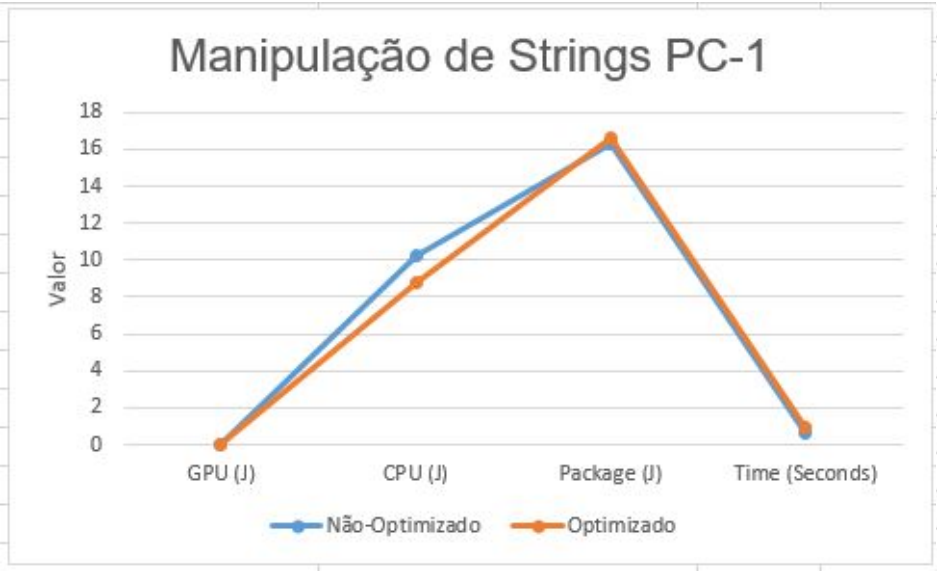
	Inicio	Otimizado	Ganho%
GPU (J)	0.0578	0.0129	4.4696
CPU (J)	10.2253	8.8202	1.1593
Package (J)	16.2525	16.6256	0.6150
Time (Seconds)	0.6094	0.9185	0.0663

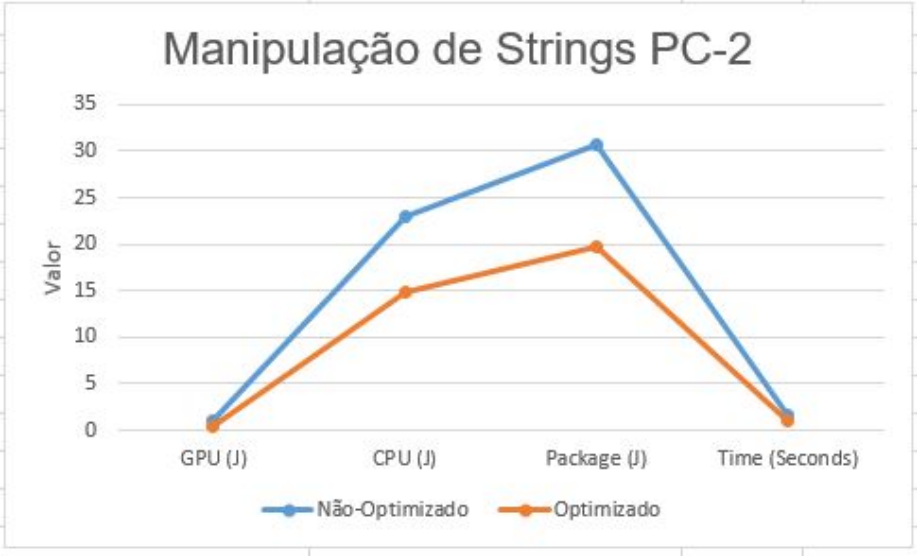
Tabela 5 - T2.txt - PC 1

	Inicio	Otimizado	Ganho%
GPU (J)	1.1025	0.4754	2.3189
CPU (J)	22.8754	14.9098	1.5342
Package (J)	30.7739	19.7599	1.5579
Time (Seconds)	1.6971	1.1157	1.5210

Tabela 6 - T2.txt - PC 2

StringBuilder - Resultados





Classe Comando

```
public static final String ADD="ADD";
public static final String EXIT="EXIT";
public static final String LOGIN="LOGIN";
public static final String LOGOUT="LOGOUT";
public static final String REGISTER="REGISTER";
public static final String TOP="TOP";
public static final String LIST="LIST";
public static final String LIKE="LIKE";
public static final String ASSIGN="ASSIGN";
public static final String APPROVE="APPROVE";
public static final String REPORT = "REPORT";
public static final String CHRONICLE= "CHRONICLE";
public static final String COMMENT ="COMMENT";
public static final String EDITOR = "EDITOR";
public static final String JOURNALIST ="JOURNALIST";
public static final String ALL = "ALL";
public static final String TOPIC = "TOPIC";
public static final String THEME = "THEME";
public static final String COMMENTS ="COMMENTS";
public static final String CHRONICLES ="CHRONICLES";
public static final String REPORTS = "REPORTS";
public static final String READER = "READER";
public static final String COLLABORATOR= "COLLABORATOR";
```

- Criação de variáveis globais evitando assim o desperdício de memória, ou seja, a mesma variável é apenas declarada uma vez;

- Código mais suscetível a mudanças.

Alteração da classe comando

```
public final class Comando {

    private static  HashMap<String,String> comandoClasse = new HashMap<String,String>();

    public static final String ADD="ADD";
    public static final String EXIT="EXIT";
    public static final String LOGIN="LOGIN";
    public static final String LOGOUT="LOGOUT";
    public static final String REGISTER="REGISTER";
    public static final String TOP="TOP";
    public static final String LIST="LIST";
```

Acrescentou-se um objeto *Hashmap* que irá conter todas as operações a serem efectuadas pelo utilizador.

Switch Statements

```
while (!cmd.equalsIgnoreCase("EXIT")){
    if (cmd.equalsIgnoreCase("LOGIN"))
        Login(cp, in);
    else if (cmd.equalsIgnoreCase("LOGOUT"))
        Logout(cp);
    else if (cmd.equalsIgnoreCase("REGISTER"))
        Register(cp, in);
    else if (cmd.equalsIgnoreCase("TOP"))
        Top(cp, in);
    else if (cmd.equalsIgnoreCase("LIST"))
        List(cp, in);
    else if (cmd.equalsIgnoreCase("LIKE"))
        Like(cp, in);
    else if (cmd.equalsIgnoreCase("ASSIGN"))
        Assign(cp, in);
    else if (cmd.equalsIgnoreCase("APPROVE"))
        Approve(cp, in);
    else if (cmd.equalsIgnoreCase("ADD"))
        Add(cp, in);
    cmd = in.next();
```

```
while(comando[0].equals(Comando.EXIT)==false) {

        try{


            String classeComando = Comando.getClasseFromComando(comando[0]);



            if(classeComando==null) {

                throw new ComandoIlegalException("Comando não encontrado");



            }

        Class c = Class.forName(classeComando);
        Constructor cons = c.getConstructor(Scanner.class,String[].class,CaparicaPost.class);
        cons.newInstance(a,comando,cp);
```

Switch Statements - Resultados

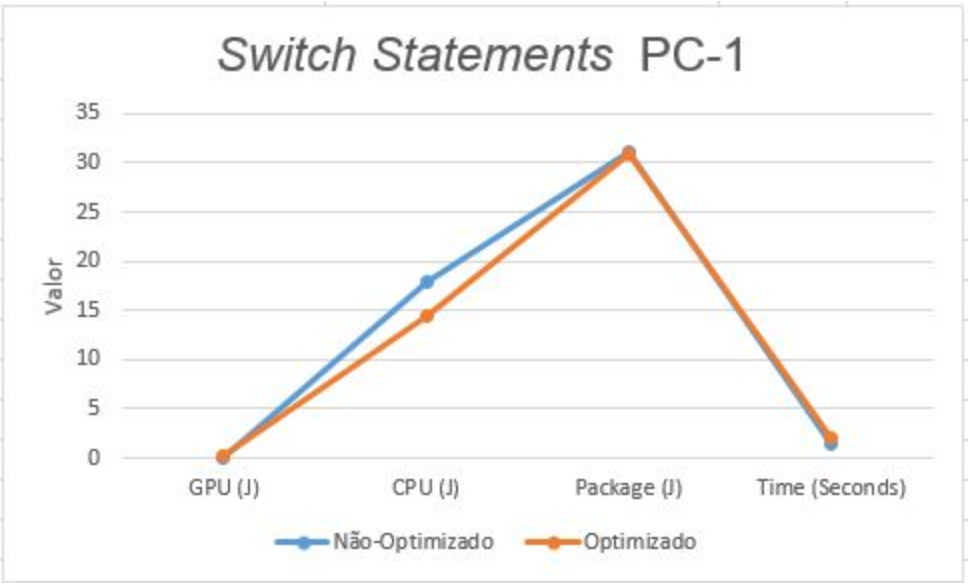
	Inicio	Otimizado	Ganho%
GPU (J)	0.0792	0.2496	0.3173
CPU (J)	17.8698	14.4057	1.2404
Package (J)	31.0919	30.8269	1.0085
Time (Seconds)	1.4439	1.9812	0.7287

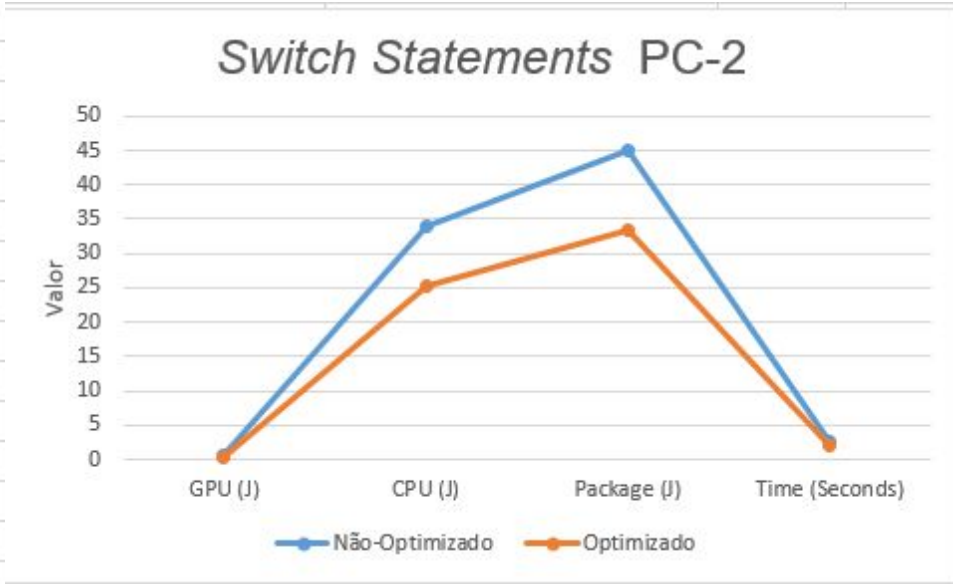
Tabela 7 - T1.txt - PC 1

	Inicio	Otimizado	Ganho%
GPU (J)	0.6301	0.3744	1.6828
CPU (J)	33.9977	25.1598	1.3512
Package (J)	45.0657	33.2955	1.3535
Time (Seconds)	2.7017	2.0599	1.311557

Tabela 8 - T1.txt - PC 2

Switch Statements - Resultados





Switch Statements - Resultados

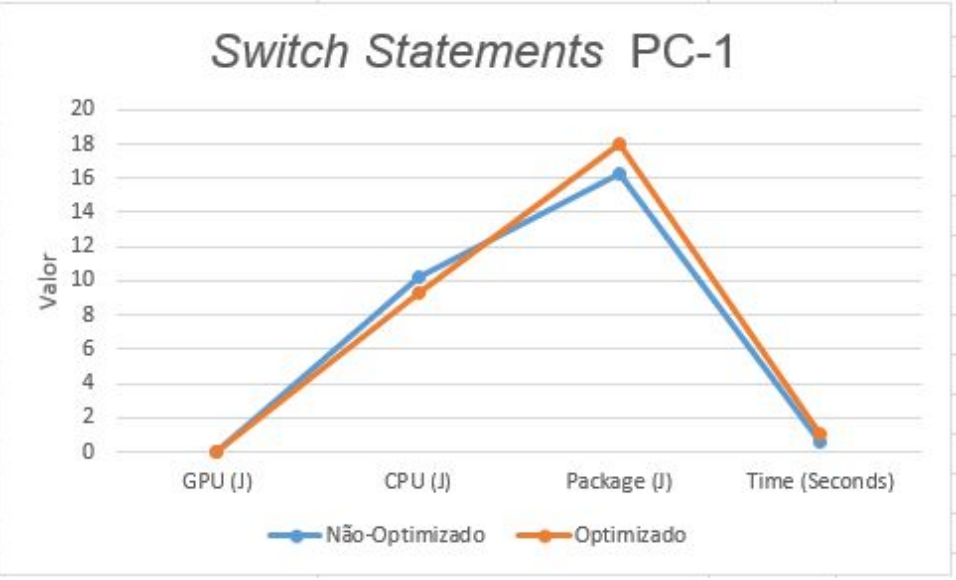
	Inicio	Otimizado	Ganho%
GPU (J)	0.0578	0.0342	1.6891
CPU (J)	10.2253	9.3083	1.0985
Package (J)	16.2525	17.9495	0.9054
Time (Seconds)	0.6094	1.0145	0.6007

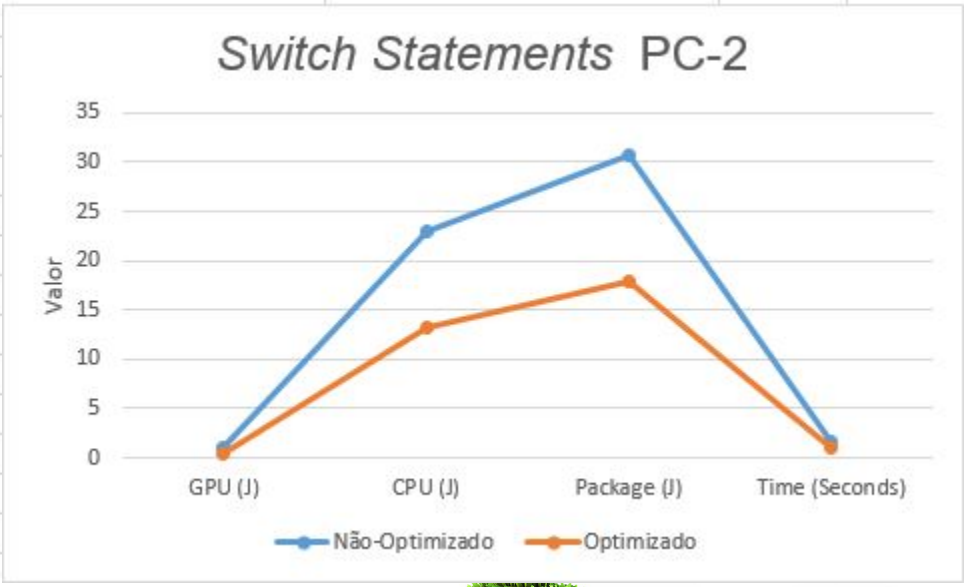
Tabela 9 - T2.txt - PC 1

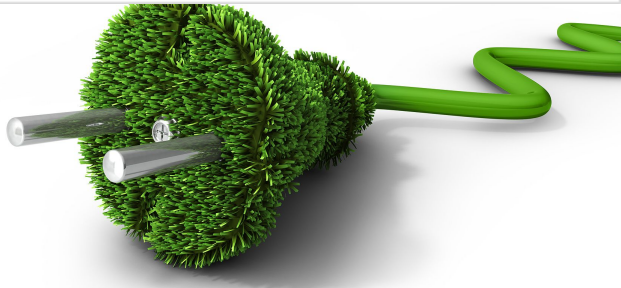
	Inicio	Otimizado	Ganho%
GPU (J)	1.1025	0.5037	2.1887
CPU (J)	22.8754	13.2326	1.7287
Package (J)	30.7739	17.7887	1.7299
Time (Seconds)	1.6971	1.0563	1.6065

Tabela 10 - T2.txt - PC 2

Switch Statements - Resultados







Alteração da Estrutura de dados - Profiling dos dados

	T1.TXT	T2.TXT
REGISTER	551	757
LOGOUT	1009	1053
ADD	4436	1587
LIKE	5321	1456
LOGIN	1009	1053
APPROVE	626	305
ASSIGN	73	38
LIST	2693	997

Depois de efectuar as várias alterações fizemos um profiling aos dados para averiguar quais as operações mais frequentes de modo a escolher as estruturas de dados apropriadas.

Alteração da estrutura de dados - Resultados

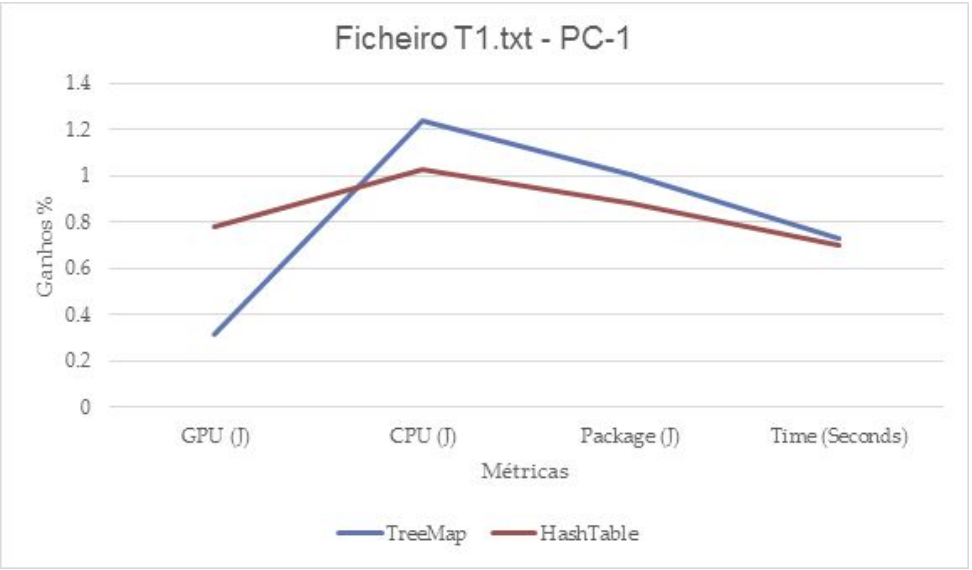
	TreeMap	HashTable	Ganho% TM	Ganho % HT
GPU (J)	0.2496	0.1012	0.3173	0.7826
CPU (J)	14.4057	17.4382	1.2404	1.0247
Package (J)	30.8269	35.1810	1.0085	0.8837
Time (Seconds)	1.9812	2.0666	0.7287	0.6986

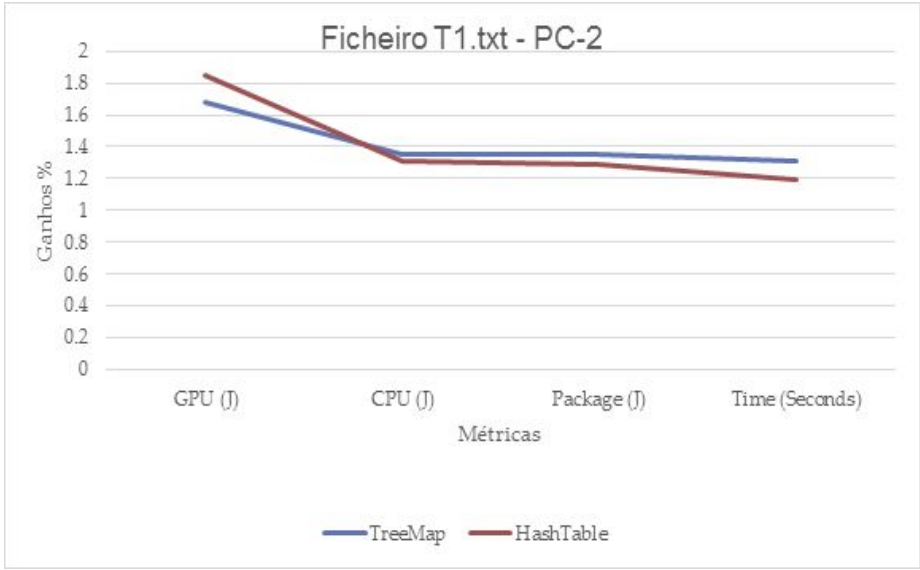
Tabela 11 - T1.txt - PC 1

	TreeMap	HashTable	Ganho% TM	Ganho% HT
GPU (J)	0.3744	0.3409	1.6828	1.8481
CPU (J)	25.1598	26.0101	1.3512	1.3070
Package (J)	33.2955	35.0502	1.3535	1.2857
Time (Seconds)	2.0599	2.2741	1.3115	1.1880

Tabela 12 - T1.txt - PC 2

Alteração da estrutura de dados - Resultados





Alteração da estrutura de dados - Resultados

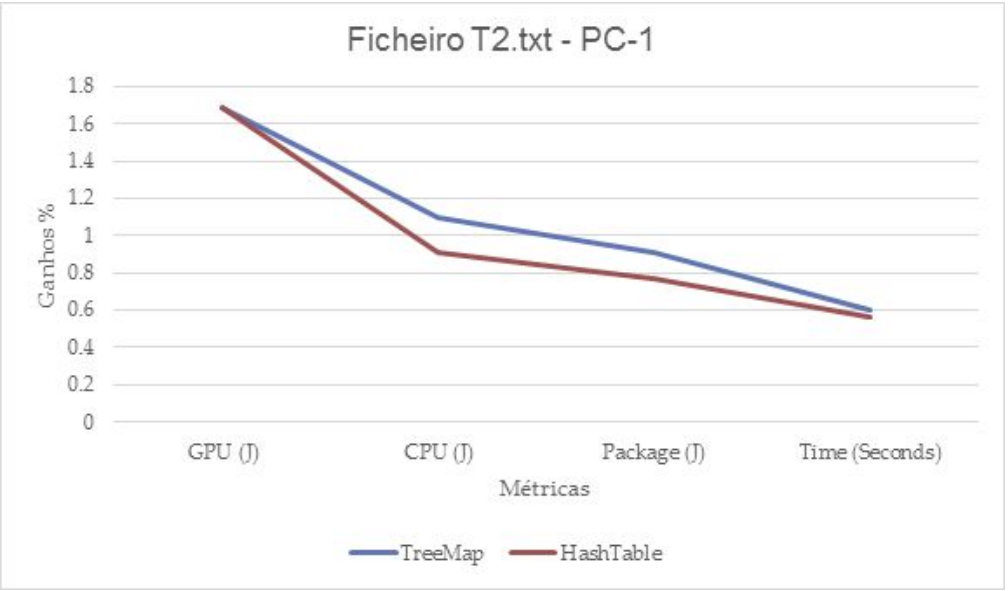
	TreeMap	HashTable	Ganho% TM	Ganho % HT
GPU (J)	0.0342	0.0342	1.6891	1.6891
CPU (J)	9.3083	11.2356	1.0985	0.9100
Package (J)	17.9495	21.0875	0.9054	0.7707
Time (Seconds)	1.0145	1.0900	0.6007	0.559

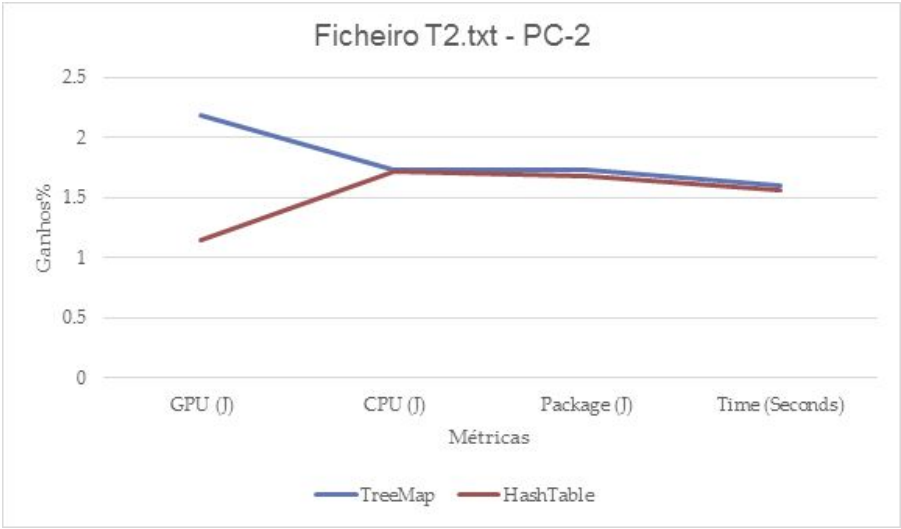
Tabela 13 - T2.txt - PC 1

	TreeMap	HashTable	Ganho% TM	Ganho% HT
GPU (J)	0.5037	0.9655	2.1887	1.1418
CPU (J)	13.2326	13.3032	1.7287	1.7195
Package (J)	17.7887	18.3768	1.7299	1.6746
Time (Seconds)	1.0563	1.0853	1.6065	1.5637

Tabela 14 - T2.txt - PC 2

Alteração da estrutura de dados - Resultados

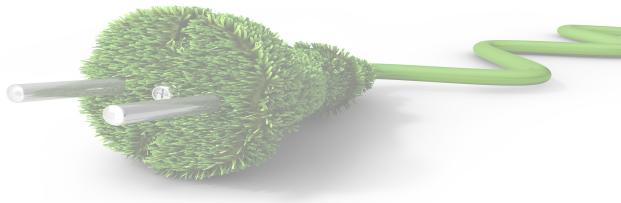




Conclusões

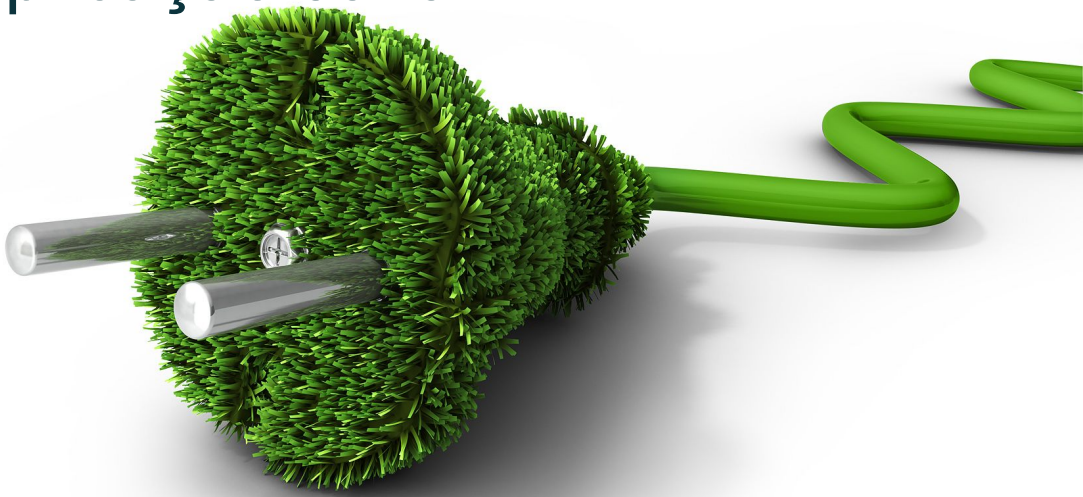
- O consumo de energia é importante para um empresa, porque se uma *query* for executada várias vezes conseguir reduzir o consumo dessa *query* a empresa conseguirá reduzir os custos associados a essa *query*;

- Criar código suscetível a mudanças é importante, porque mais tarde esse código poderá ser reutilizado bastando fazer uma pequenas mudanças



CaparicaPost

Análise e Otimização de Consumo de Energia de uma Aplicação Java



Universidade do Minho
Escola de Engenharia

Trabalho realizado por:
A71604 - Diogo Couto
A67738 - Gil Gonçalves
A67751 - Pedro Silva