



**Universidade do Minho**  
Escola de Engenharia  
Mestrado Integrado em Engenharia Informática

## **Unidade Curricular de Análise e Testes de Software**

### **MIEI 4.º ANO**

Ano Letivo de 2015/2016  
Trabalho Prático - Parte1

## **GREEN PROFILING getEnergy**

### **“Projeto 4: Green Profiling**

Neste projeto pretende-se construir um profiler de energia para C++. A ideia é este analisador produzir um gráfico com a energia que vai sendo usada (energia em Joules no eixo do y) durante a execução do programa (tempo de execução em ms no eixo do x). A equipa docente desenvolveu já mecanismos para ligar programas em Java e Haskell à framework RAPL da intel, que estima o consumo de energia em CPUs daquele fabricante. Usando esta ligação à framework RAPL, pretende-se que durante a execução de um programa C++, logo durante a execução da máquina virtual MSP, e a cada X milissegundos se invoque a API RAPL para sabermos que energia está a ser consumida nesse momento. Após a máquina virtual terminar de executar, será produzido um gráfico com o profile do consumo de energia (eventualmente exportando a informação para csv e usando um sistema de folhas de cálculo). De modo a termos uma leitura da energia em intervalos de tempo fixos, a leitura de energia terá de ser feita por uma thread/processo diferente daquele que está a executar a máquina virtual. “

*fonte: página oficial da disciplina*

David Miguel Alves (a53791)  
Nélson Emanuel Pereira Torres (pg31063)  
Pedro Duarte Cardoso Lopes (a32652)

Dezembro, 2015

## Proposta de Trabalho:

O objectivo da primeira parte do trabalho prático de Análise e Teste de Software passou por implementar uma solução de software para medição de consumos energéticos e comparação de consumos entre soluções alternativas computacionais.

Foi nos disponibilizada pela equipa docente uma maquina virtual e um processador para a linguagem c++. A ideia é implementar uma instrução “energy” para medição de consumos energéticos das instruções dos programas em C++, ou seja, na sua execução na máquina virtual MSP.

Para o efeito recorreremos à utilização da framework Rapl da Intel.

Numa fase inicial, pretendemos fazer a captação de consumos através da invocação de uma função ou expressão que, sendo acrescentada à linguagem C++ ( no código fonte dos ficheiros a serem processados) nos permita obter, em tempo de execução, uma consulta de valores energéticos por parte da máquina virtual.

Introdução da instrução “energia()” no código fonte do ficheiro **maiorDeDoisNumeros.i** :

```
void main() {  
  
    energia();  
    int a;  
    int b;  
    int res;  
    a = input(int);  
    b = input(int);  
    res = max(a,b);  
    print('=');  
    print(res);  
}  
  
int max(int a, int b){  
    int res;  
    if (a > b) {  
        res = a;  
    }  
    else {  
        res = b;  
    }  
    return res;  
}
```

## Utilização do Software:

Para executar a máquina virtual utilizando a framework Rapl deveremos executar o comando `modprobe` com direitos de administrador. O "modprobe" é um programa utilizado para adicionar módulos de kernel ao kernel do sistema operativo e que possibilita, no caso do Rapl, aceder aos valores de consumos do CPU.

Quando executamos a partir da máquina virtual um programa fonte C--, caso seja encontrada uma entrada da instrução "energia()", será então efectuada uma consulta de consumos de CPU e o registo devolvido será guardado numa pasta "logs" na raiz do projecto. Este registo será nomeado com o timestamp do momento de acesso e poderá ser assim identificado.

Exemplo da Execução com o ficheiro `maiorDeDoisNumeros.i` modificado:

```
pedro@pedro-Yoga: ~/greenmspFinal1/Tools/i2msp/Tom
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$ sudo modprobe msr
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$ sh exec.sh
3
4
0
#
=
4

pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$ date
Qui Dez  3 01:36:22 WET 2015
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$ ls -l logs
total 48
-rw-rw-r-- 1 pedro pedro 77 Dez  2 04:11 energy_2015-12-02_04:11:26.log
-rw-rw-r-- 1 pedro pedro 77 Dez  2 04:13 energy_2015-12-02_04:13:16.log
-rw-rw-r-- 1 pedro pedro 77 Dez  2 04:17 energy_2015-12-02_04:17:41.log
-rw-rw-r-- 1 pedro pedro 78 Dez  2 14:21 energy_2015-12-02_14:21:26.log
-rw-rw-r-- 1 pedro pedro 78 Dez  2 21:30 energy_2015-12-02_21:30:39.log
-rw-rw-r-- 1 pedro pedro 77 Dez  2 22:03 energy_2015-12-02_22:03:24.log
-rw-rw-r-- 1 pedro pedro 78 Dez  2 22:05 energy_2015-12-02_22:05:06.log
-rw-rw-r-- 1 pedro pedro 76 Dez  2 22:06 energy_2015-12-02_22:05:56.log
-rw-rw-r-- 1 pedro pedro  0 Dez  2 23:14 energy_2015-12-02_23:14:52.log
-rw-rw-r-- 1 pedro pedro  0 Dez  2 23:15 energy_2015-12-02_23:15:46.log
-rw-rw-r-- 1 pedro pedro  0 Dez  3 01:11 energy_2015-12-03_01:11:34.log
-rw-rw-r-- 1 pedro pedro 77 Dez  3 01:12 energy_2015-12-03_01:12:38.log
-rw-rw-r-- 1 pedro pedro 78 Dez  3 01:17 energy_2015-12-03_01:17:20.log
-rw-rw-r-- 1 pedro pedro 77 Dez  3 01:35 energy_2015-12-03_01:35:32.log
-rw-rw-r-- 1 pedro pedro 75 Dez  3 01:36 energy_2015-12-03_01:36:03.log
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$ more logs/energy_2015-12-03_0
1:36:03.log
ram: 0.091858000000002 cpu: 10.537108999997145 package: 41.43383800001175
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$
```

## Implementação:

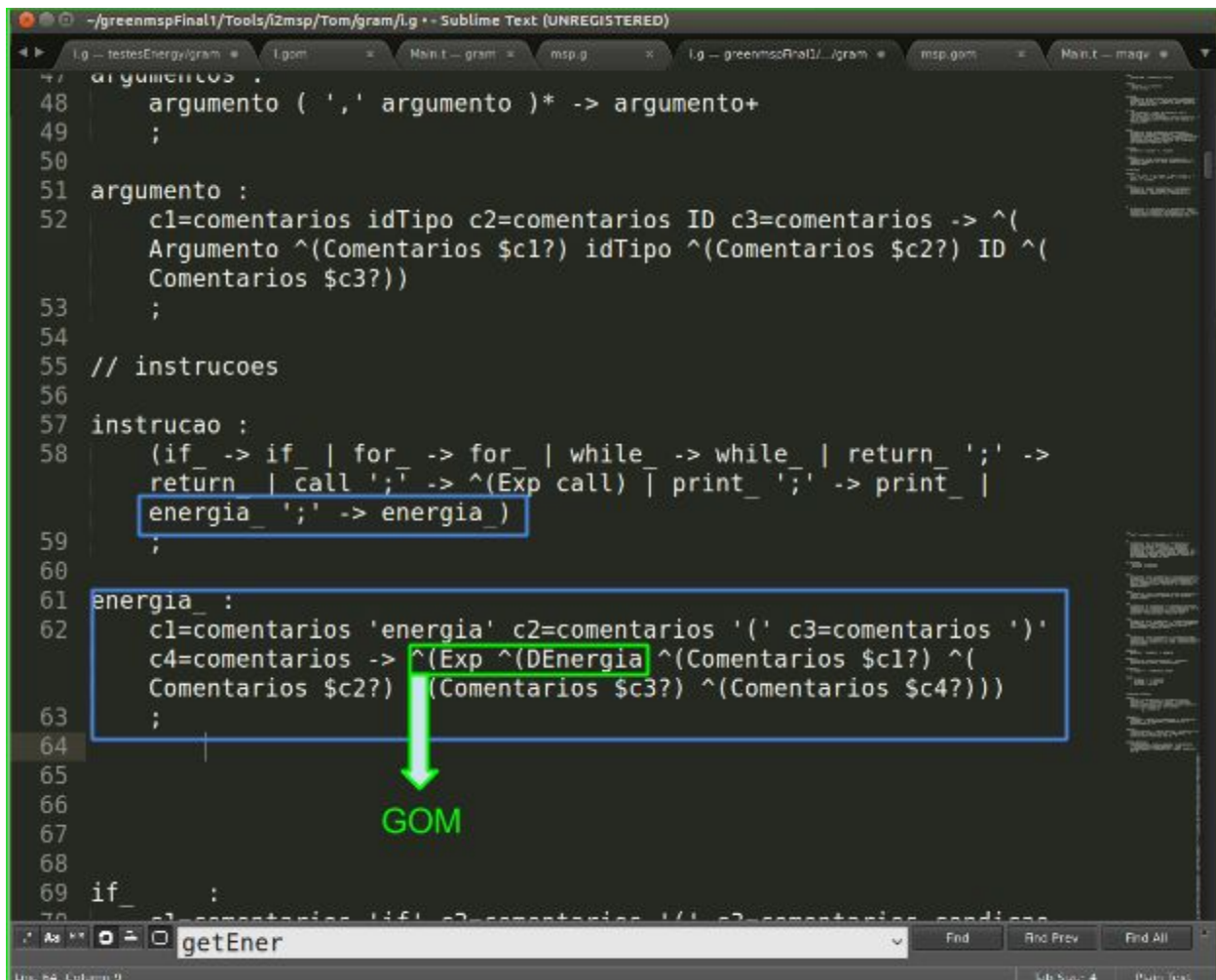
A introdução da instrução “energy” na gramática do C-- e na MSP exigiu uma série de alterações em vários ficheiros do programa, cuja descrição se segue:

## Alterações no Processador da Linguagem C--

### ANTLR

Adicionamos à gramática do C-- um novo símbolo **Não Terminal energia\_**.

No ficheiro **i.g** (Gramática do ANTLR com os construtores para GOM-TOM) que se encontra na pasta **gram** do projecto:



```
48 argumento ( ',' argumento )* -> argumento+
49 ;
50
51 argumento :
52   c1=comentarios idTipo c2=comentarios ID c3=comentarios -> ^(
53     Argumento ^(Comentarios $c1?) idTipo ^(Comentarios $c2?) ID ^(
54       Comentarios $c3?)
55   );
56
57 // instrucoes
58
59 instrucao :
60   (if_ -> if_ | for_ -> for_ | while_ -> while_ | return_ ';' ->
61     return_ | call ';' -> ^(Exp call) | print_ ';' -> print_ |
62     energia_ ';' -> energia_)
63   ;
64
65 energia_ :
66   c1=comentarios 'energia' c2=comentarios '(' c3=comentarios ')'
67   c4=comentarios -> ^(Exp ^(Denergia ^(Comentarios $c1?) ^(
68     Comentarios $c2?) (Comentarios $c3?) ^(Comentarios $c4?)))
69   ;
70
71 if_ :
```

GOM

getEner

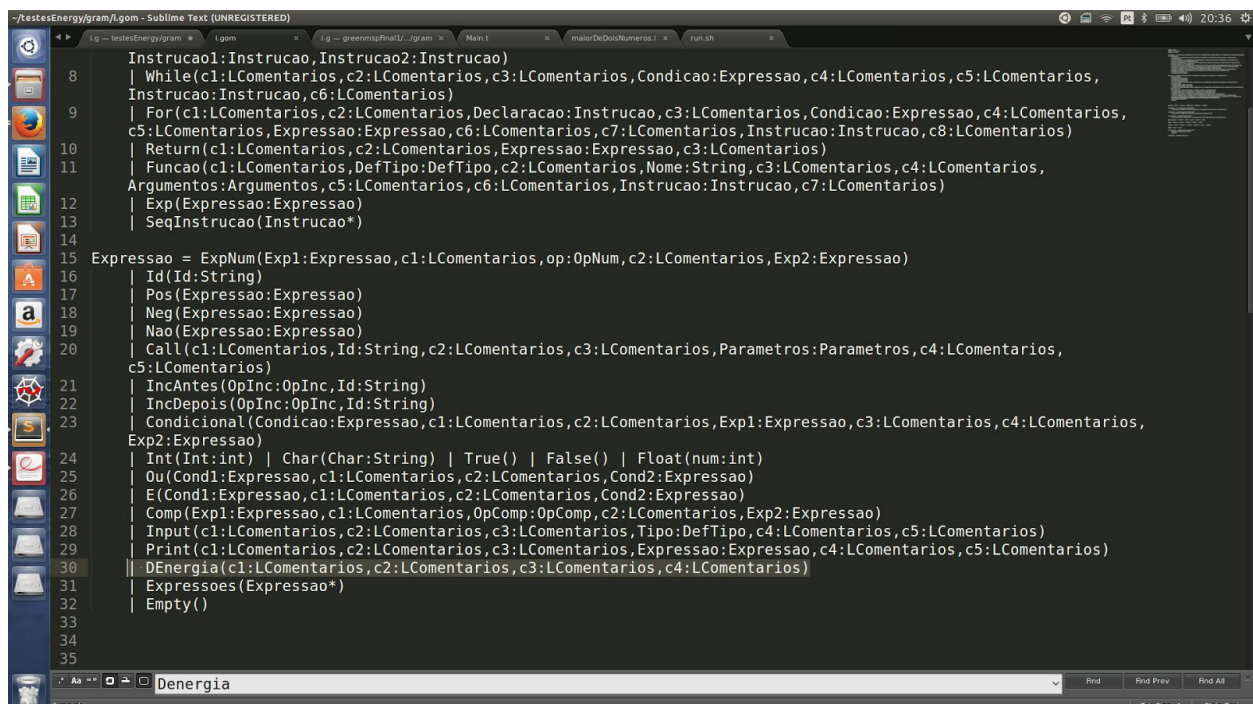
Line 64, Column 9

## Nota:

Na análise lexical poderíamos ter feito match de “energia” apenas e não “energia()”, interpretando como sendo uma palavra reservada da linguagem c--, contudo como pretendemos deixar em aberto a possibilidade de na segunda parte do projecto implementar a função energia(“tempos”,...) para poder utilizar argumentos começamos desde já a preparar a componente lexical nesse sentido.

## Estrutura de Dados do GOM

No ficheiro **i.gom**, o texto seleccionado no editor:



```
Instrucao1:Instrucao,Instrucao2:Instrucao)
| While(c1:LComentarios,c2:LComentarios,c3:LComentarios,Condicao:Expressao,c4:LComentarios,c5:LComentarios,
Instrucao:Instrucao,c6:LComentarios)
| For(c1:LComentarios,c2:LComentarios,Declaracao:Instrucao,c3:LComentarios,Condicao:Expressao,c4:LComentarios,
c5:LComentarios,Expressao:Expressao,c6:LComentarios,c7:LComentarios,Instrucao:Instrucao,c8:LComentarios)
| Return(c1:LComentarios,c2:LComentarios,Expressao:Expressao,c3:LComentarios)
| Funcao(c1:LComentarios,DefTipo:DefTipo,c2:LComentarios,Nome:String,c3:LComentarios,c4:LComentarios,
Argumentos:Argumentos,c5:LComentarios,c6:LComentarios,Instrucao:Instrucao,c7:LComentarios)
| Exp(Expressao:Expressao)
| SeqInstrucao(Instrucao*)

15 Expressao = ExpNum(Exp1:Expressao,c1:LComentarios,op:OpNum,c2:LComentarios,Exp2:Expressao)
16 | Id(Id:String)
17 | Pos(Expressao:Expressao)
18 | Neg(Expressao:Expressao)
19 | Nao(Expressao:Expressao)
20 | Call(c1:LComentarios,Id:String,c2:LComentarios,c3:LComentarios,Parametros:Parametros,c4:LComentarios,
c5:LComentarios)
21 | IncAntes(OpInc:OpInc,Id:String)
22 | IncDepois(OpInc:OpInc,Id:String)
23 | Condicional(Condicao:Expressao,c1:LComentarios,c2:LComentarios,Exp1:Expressao,c3:LComentarios,c4:LComentarios,
Exp2:Expressao)
24 | Int(Int:int) | Char(Char:String) | True() | False() | Float(num:int)
25 | Ou(Cond1:Expressao,c1:LComentarios,c2:LComentarios,Cond2:Expressao)
26 | E(Cond1:Expressao,c1:LComentarios,c2:LComentarios,Cond2:Expressao)
27 | Comp(Exp1:Expressao,c1:LComentarios,OpComp:OpComp,c2:LComentarios,Exp2:Expressao)
28 | Input(c1:LComentarios,c2:LComentarios,c3:LComentarios,Tipo:DefTipo,c4:LComentarios,c5:LComentarios)
29 | Print(c1:LComentarios,c2:LComentarios,c3:LComentarios,Expressao:Expressao,c4:LComentarios,c5:LComentarios)
30 | DEnergia(c1:LComentarios,c2:LComentarios,c3:LComentarios,c4:LComentarios)
31 | Expresoes(Expressao*)
32 | Empty()
33
34
35
```

Note-se no construtor da gramática (ANTLR) no slide anterior e a correspondência na estrutura de dados do GOM:

```
energia_ :
    c1=comentarios 'energia' c2=comentarios '(' c3=comentarios ')' c4=comentarios -> ^(Exp
    ^DEnergia ^Comentarios $c1?) ^Comentarios $c2?) ^Comentarios $c3?) ^Comentarios
    $c4?));
```



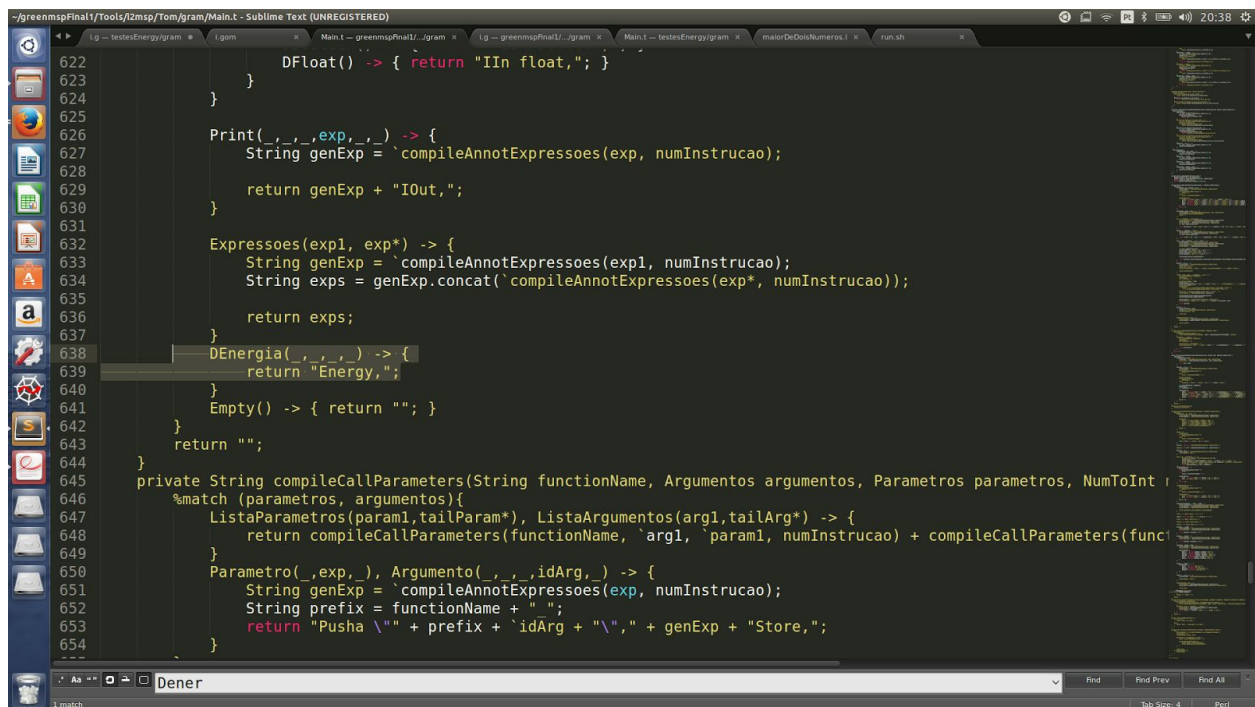
A regra **Exp ^ (DEnergia)** do ANTLR vai gerar instrução na estrutura de dados do gom:

```
Instrução = ....  
|....  
|Exp(Expressao: Expressao)  
|...
```

## Gerador TOM para a Linguagem do MSP

O ficheiro **Main.t** é o responsável por gerar o código legível para a Máquina virtual e adicionamos portanto um método para tratamento da instrução “energy()”, no gom **DEnergia()**.

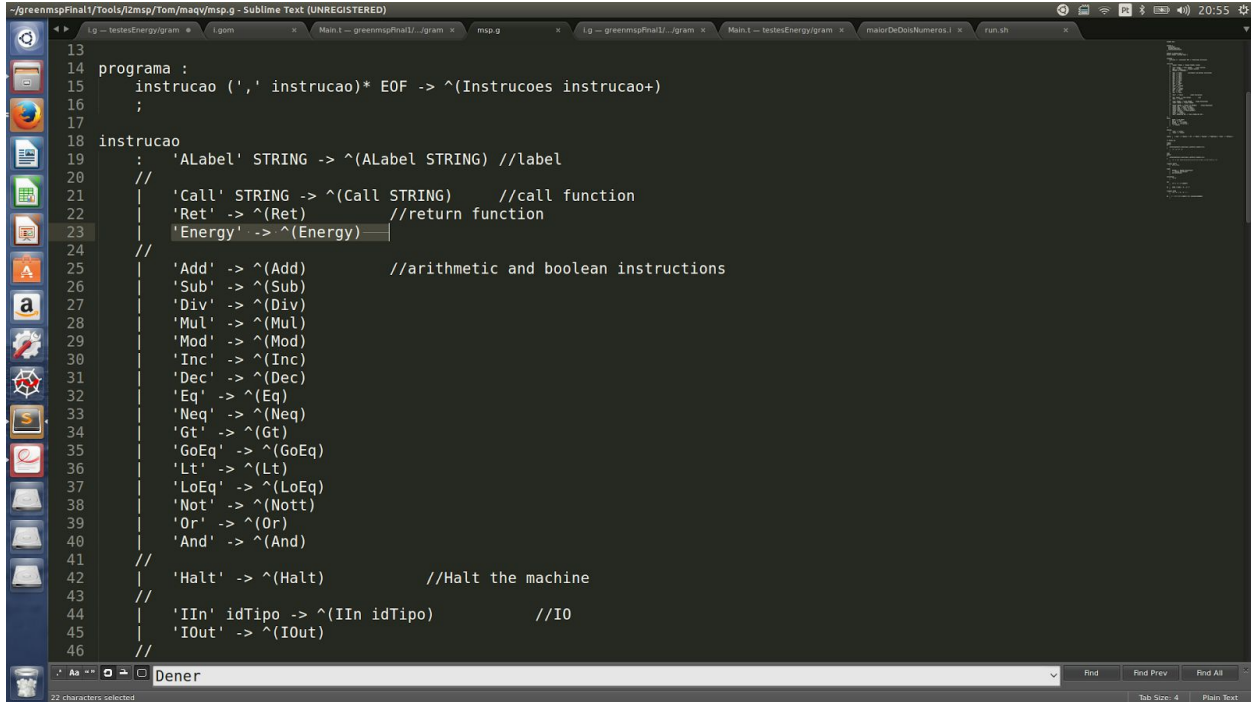
Vai ser enviado para a máquina virtual para posterior processamento o texto “Energy,”.



```
622         DFloat() -> { return "IIn float;"; }  
623     }  
624 }  
625  
626 Print(_,_,exp,_) -> {  
627     String genExp = `compileAnnotExpressoes(exp, numInstrucao);  
628  
629     return genExp + "IOut;";  
630 }  
631  
632 Expressoes(exp1, exp*) -> {  
633     String genExp = `compileAnnotExpressoes(exp1, numInstrucao);  
634     String exps = genExp.concat(`compileAnnotExpressoes(exp*, numInstrucao));  
635  
636     return exps;  
637 }  
638 DEnergia(_,_,_) -> {  
639     return "Energy;";  
640 }  
641 Empty() -> { return ""; }  
642 }  
643 return "";  
644 }  
645 private String compileCallParameters(String functionName, Argumentos argumentos, Parametros parametros, NumToInt numToInt) {  
646     %match (parametros, argumentos){  
647         ListaParametros(param1,tailParam*), ListaArgumentos(arg1,tailArg*) -> {  
648             return compileCallParameters(functionName, `arg1, `param1, numInstrucao) + compileCallParameters(functionName, `arg1, `tailArg*, numToInt);  
649         }  
650     }  
651     Parametro(_,exp,_, Argumento(_,_,idArg,_) -> {  
652         String genExp = `compileAnnotExpressoes(exp, numInstrucao);  
653         String prefix = functionName + " ";  
654         return "Pusha \"\" + prefix + `idArg + "\",\" + genExp + "Store;";  
655     }  
656 }
```

## Alterações na Máquina Virtual MSP:

Ficheiro **misp.g** (texto seleccionado no editor):



```
13
14 programa :
15   instrucao (',' instrucao)* EOF -> ^(Instrucoes instrucao+)
16   ;
17
18 instrucao
19   : 'ALabel' STRING -> ^(ALabel STRING) //label
20   //
21   | 'Call' STRING -> ^(Call STRING) //call function
22   | 'Ret' -> ^(Ret) //return function
23   | 'Energy' -> ^(Energy)
24   //
25   | 'Add' -> ^(Add) //arithmetic and boolean instructions
26   | 'Sub' -> ^(Sub)
27   | 'Div' -> ^(Div)
28   | 'Mul' -> ^(Mul)
29   | 'Mod' -> ^(Mod)
30   | 'Inc' -> ^(Inc)
31   | 'Dec' -> ^(Dec)
32   | 'Eq' -> ^(Eq)
33   | 'Neq' -> ^(Neq)
34   | 'Gt' -> ^(Gt)
35   | 'GoEq' -> ^(GoEq)
36   | 'Lt' -> ^(Lt)
37   | 'LoEq' -> ^(LoEq)
38   | 'Not' -> ^(Nott)
39   | 'Or' -> ^(Or)
40   | 'And' -> ^(And)
41   //
42   | 'Halt' -> ^(Halt) //Halt the machine
43   //
44   | 'IIn' idTipo -> ^(IIn idTipo) //IO
45   | 'IOut' -> ^(IOut)
46   //
```

Na página anterior verificamos que no caso de ser encontrada a instrução “energia()” no ficheiro fonte da linguagem C--, seria enviado pelo Processador da Linguagem para a máquina virtual MSP a expressão “Energia,”

No ficheiro **misp.gom** (texto seleccionado no editor de texto):

```
1 module maqv.msp
2 imports int String
3 abstract syntax
4
5 Instrucoes = Instrucoes(Instrucao*)
6
7 Instrucao = ALabel(id:String)
8   Call(id:String)
9   Ret()
10  Energy()
11  Add()
12  Sub()
13  Div()
14  Mul()
15  Mod()
16  Inc()
17  Dec()
18  Eq()
19  Neq()
20  Gt()
21  GoEq()
22  Lt()
23  LoEq()
24  Nott()
25  Or()
26  And()
27  Halt()
28  IIn(tipo:DefTipo)
29  IOut()
30  Jump(id:String)
31  Jumpf(id:String)
32  Push(t:Termo)
33  Pusha(t:Termo)
34  Load()
```

No ficheiro **Main.t** (texto selecionado no editor de texto):

```
275
276
277 public String run(Instrucoes prog) throws IOException{
278   pc++;
279   Instrucoes orig = this.original;
280   %match (prog){
281     Instrucoes(inst,instrs*) -> {
282       %match(inst) {
283         ALabel(id) -> {
284           if(`id.startsWith("f:")){
285             actualFuncName=`id;
286           }
287           return `run(instrs*);
288         }
289         Call(id) -> {
290           `pushFuncs(S(actualFuncName));
291           `pushFuncs(S(id));
292           prog = `jmp(orig,id);
293           return `run(prog);
294         }
295         Energy() ->{
296           getEnergia();
297           return `run(instrs*);
298         }
299       }
300     }
301     Ret() -> {
302       Termo calledLabel = `topFuncs();
303       `popFuncs();
304       Termo callerFLabel = `topFuncs();
305       `popFuncs();
306     }
307   }
```

É executado o método `getEnergia()` desenvolvido pelo grupo que invoca uma instância da classe “EnergyExample” presente na API do RAPL fornecido pelos docentes. Em seguida é retomada a seguinte instrução do programa a ser lida.



## Thread `getEnergia` (*Prova de Conceito*)

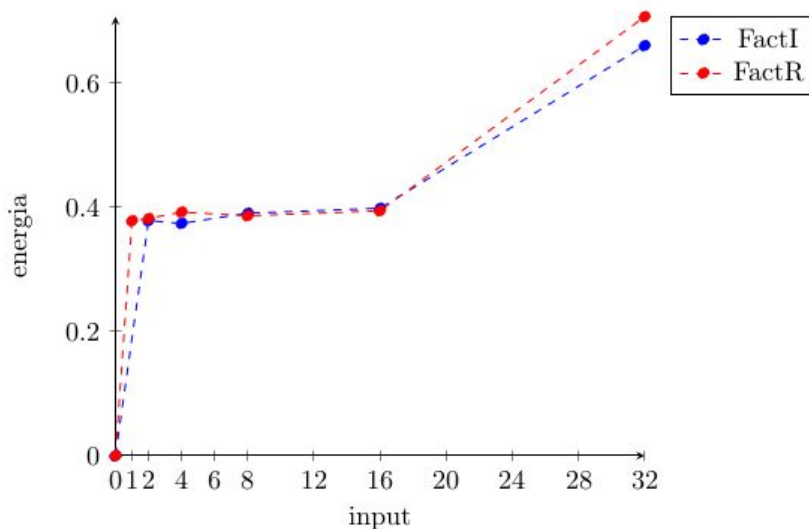
Nesta segunda fase já alteramos a thread de forma a que seja possível notar alguns consumos viáveis, para isso tomamos algumas decisões tal como invocar a thread apenas na primeira chamada e reutilizá-la nas chamadas seguintes através de modo a diminuir o overhead; A thread passou a escrever no log só quando a função termina e não a cada chamada da função, para isso a thread tem um estado que vai sendo atualizado a cada chamada e no final da captura escreve e desliga-se.

Também foi decidido não implementar a captura de 10 em 10 segundos, como foi comentado no início dos projectos, uma vez que a captura pode ser feita a pedido do utilizador, não com base de tempo, mas com base em ações, tornando os resultados assim mais viáveis.

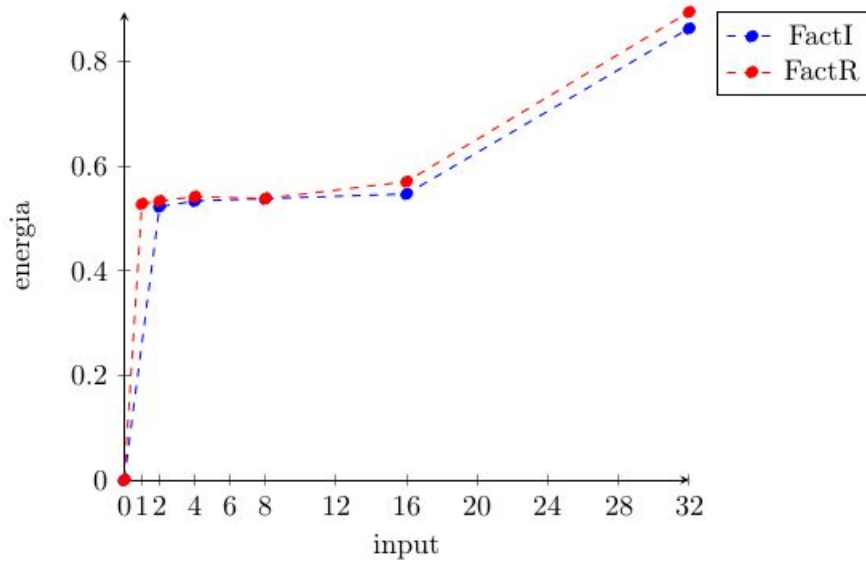
Na fase final fizemos as alterações que nos foram propostas e temos 2 métodos de captura de energia, o manual e o automático. O automático calcula a energia no início e fim da chamada de uma função e o manual é igual ao sistema que tínhamos implementado antes, sendo assim, agora é possível correr um programa invocando funções e sem “`getEnergy()`” e obter energia à mesma.

Com o trabalho realizá-lo foi possível obter gráficos como:

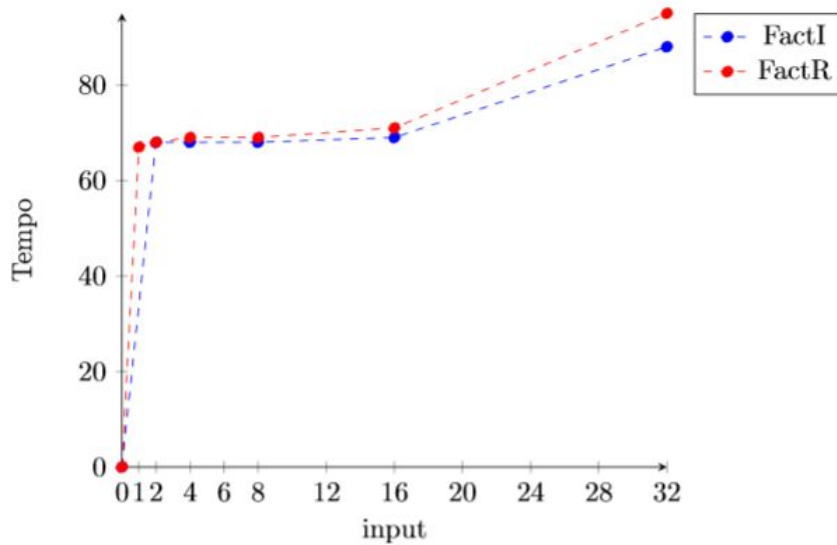
### 1 CPU



## 2 Package



## 3 Time



## Dificuldades

Nesta fase tivemos maior dificuldade na configuração das ferramentas, o que fez com que a maior fatia de tempo dispendida no projecto fosse gasta na percepção de erros de configuração do que propriamente no desenvolvimento da componente computacional. Como

por exemplos poderemos referir o facto de termos encontrado dificuldades decidir a versão do java mais apropriada (1.7 ou 1.8) uma vez que os erros de compilação inicialmente eram variados e alternados entre ambas versões, também erros de linkagem de bibliotecas e variadas dificuldades com a definição e obtenção do classpath (reconhecimento do ficheiro mas não reconhecimento das funções que esta suporta através do JNI), especificamente no caso do Tom quando executado em modo de administrador.

Estes problemas foram contornados recorrendo à utilização de uma thread para fazer o trabalho relacionado com a medição de consumos, afastando-nos assim da maior complexidade de ter que lidar com compatibilidades entre as variadas ferramentas utilizadas na composição da máquina virtual.

## Conclusão

Conseguimos completar aquilo que nos foi proposto, implementar a instrução `getEnergy` na máquina virtual msp. Tivemos alguma dificuldade em conseguir trabalhar com o parser e a gramática e obtivemos também algumas dificuldades durante a compilação que aos poucos foram resolvidas. No final do trabalho já nos foi possível de comparar e analisar funções o objectivo pretendido.