



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

Implementação de um método energia em c--

“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 milisegundos 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)

Janeiro, 2016

Proposta de Trabalho:

O objectivo 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** :

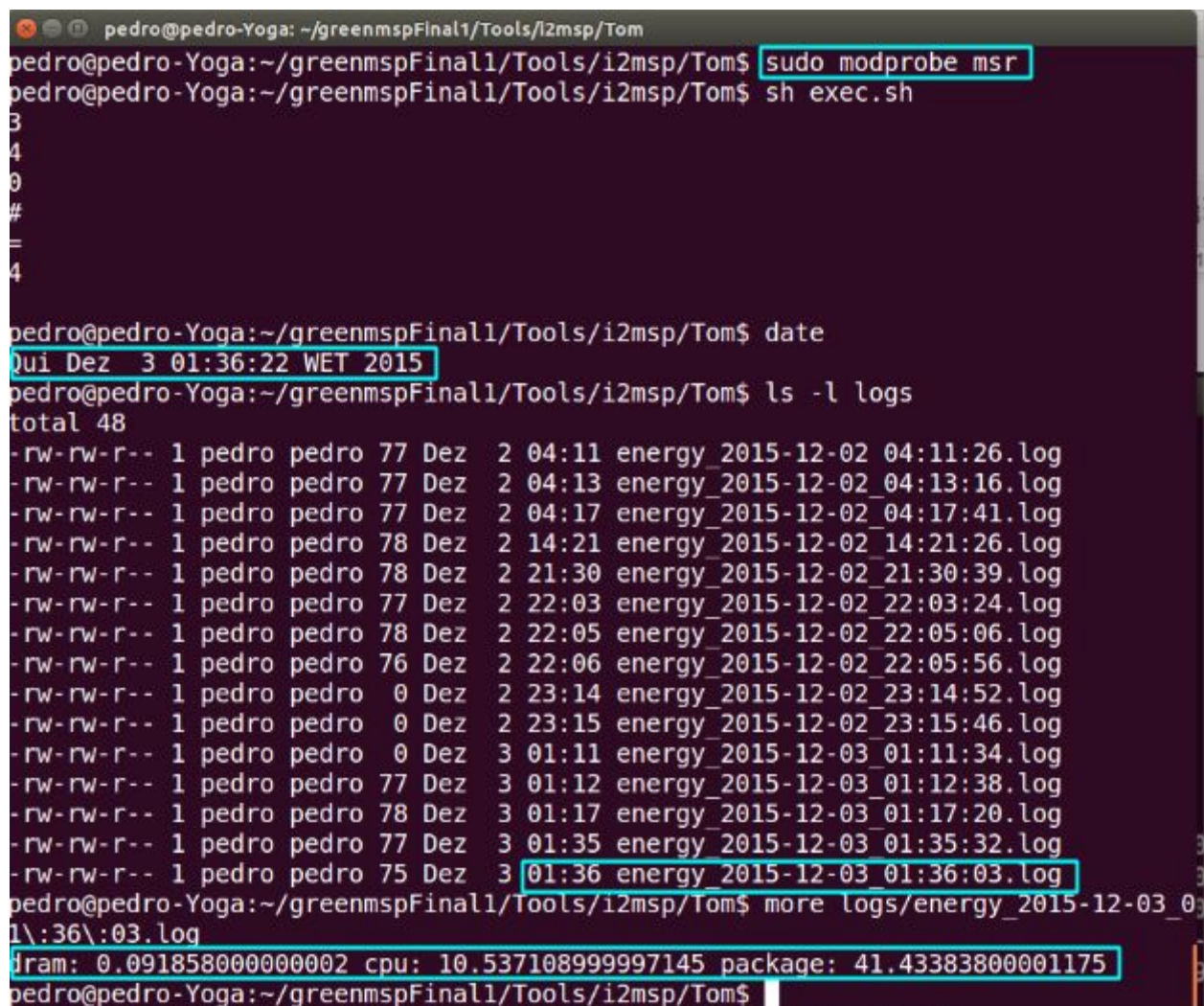
```
void main() {  
  
    energia();  
    int a;  
    int b;  
    int res;  
    a = input(int);  
    b = input(int);  
    res = max(a,b);  
    print('=');  
    print(res);  
}  
  
int max(int a, int b){  
    int res;  
    if (a > b) {  
        res = a;  
    }  
    else {  
        res = 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_01:36:03.log
ram: 0.091858000000002 cpu: 10.537108999997145 package: 41.43383800001175
pedro@pedro-Yoga:~/greenmspFinal1/Tools/i2msp/Tom$
```

Implementação:

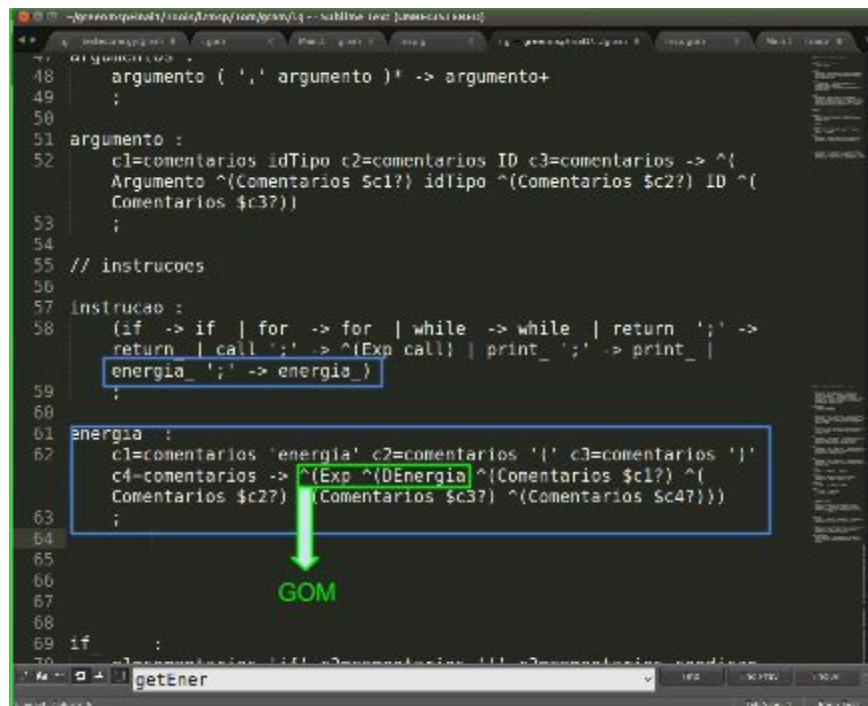
A introdução da instrução “energia” na gramática do C-- e na MSP exigiu uma série de alterações em vários ficheiros do programa, cuja descrição se segue:

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:



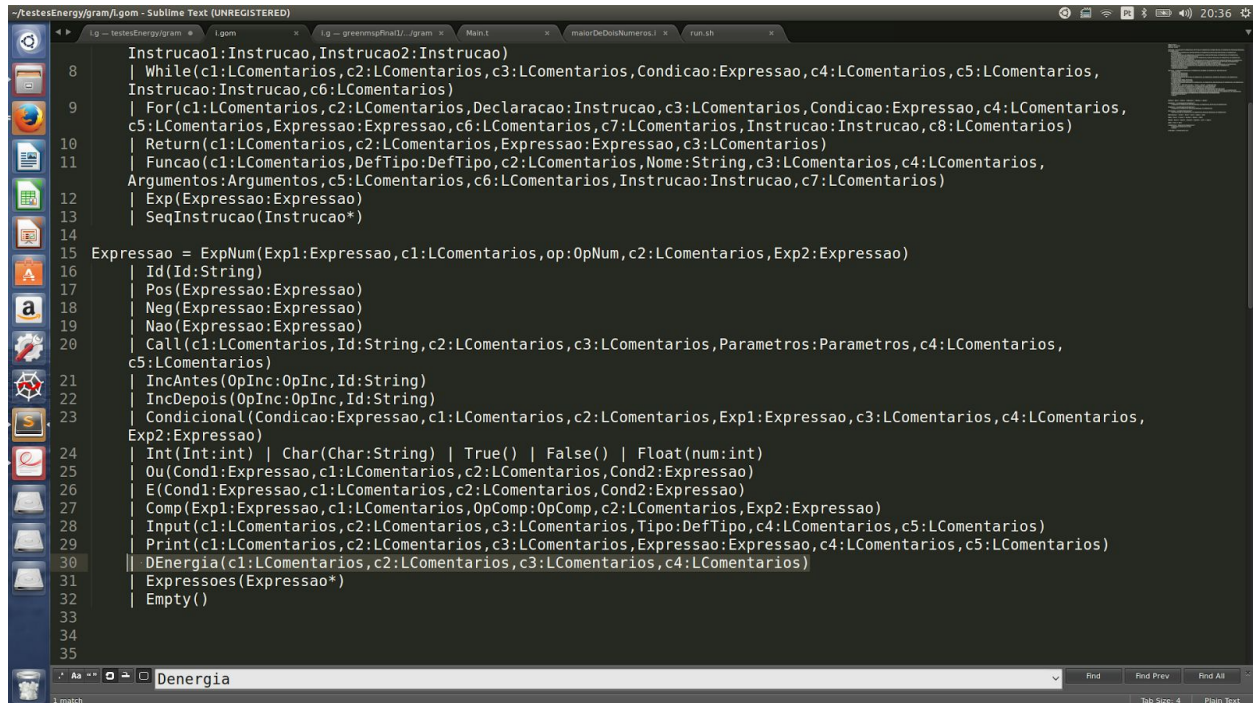
```
48     argumento ( ',' argumento )+ -> argumento+
49     ;
50
51     argumento :
52     c1=comentarios idTipo c2=comentarios ID c3=comentarios -> ^(
53     Argumento ^(Comentarios $c1?) idTipo ^(Comentarios $c2?) ID ^(
54     Comentarios $c3?)
55     ;
56
57     // instrucoes
58     instrucao :
59     (if -> if | for -> for | while -> while | return ';' ->
60     return | call ';' -> ^(Exp call) | print ';' -> print |
61     energia_ ';' -> energia_)
62     ;
63
64     energia_ :
65     c1=comentarios 'energia' c2=comentarios '(' c3=comentarios ')'
66     c4=comentarios -> ^(Exp ^(DEnergia ^(Comentarios $c1?) ^(
67     Comentarios $c2?) (Comentarios $c3?) ^(Comentarios $c4?)))
68     ;
69
70     if :
```

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:



```
8 Instrucao1:Instrucao,Instrucao2:Instrucao)
9 | While(c1:LComentarios,c2:LComentarios,c3:LComentarios,Condicao:Expressao,c4:LComentarios,c5:LComentarios,
10 Instrucao:Instrucao,c6:LComentarios)
11 | For(c1:LComentarios,c2:LComentarios,Declaracao:Instrucao,c3:LComentarios,Condicao:Expressao,c4:LComentarios,
12 c5:LComentarios,Expressao:Expressao,c6:LComentarios,c7:LComentarios,Instrucao:Instrucao,c8:LComentarios)
13 | Return(c1:LComentarios,c2:LComentarios,Expressao:Expressao,c3:LComentarios)
14 | Funcao(c1:LComentarios,DefTipo:DefTipo,c2:LComentarios,Nome:String,c3:LComentarios,c4:LComentarios,
15 Argumentos:Argumentos,c5:LComentarios,c6:LComentarios,Instrucao:Instrucao,c7:LComentarios)
16 | Exp(Expressao:Expressao)
17 | SeqInstrucao(Instrucao*)
18
19 Expressao = ExpNum(Exp1:Expressao,c1:LComentarios,op:OpNum,c2:LComentarios,Exp2:Expressao)
20 | Id(Id:String)
21 | Pos(Expressao:Expressao)
22 | Neg(Expressao:Expressao)
23 | Nao(Expressao:Expressao)
24 | Call(c1:LComentarios,Id:String,c2:LComentarios,c3:LComentarios,Parametros:Parametros,c4:LComentarios,
25 c5:LComentarios)
26 | IncAntes(OpInc:OpInc,Id:String)
27 | IncDepois(OpInc:OpInc,Id:String)
28 | Condicional(Condicao:Expressao,c1:LComentarios,c2:LComentarios,Exp1:Expressao,c3:LComentarios,c4:LComentarios,
29 Exp2:Expressao)
30 | Int(Int:int) | Char(Char:String) | True() | False() | Float(num:int)
31 | Ou(Cond1:Expressao,c1:LComentarios,c2:LComentarios,Cond2:Expressao)
32 | E(Cond1:Expressao,c1:LComentarios,c2:LComentarios,Cond2:Expressao)
33 | Comp(Exp1:Expressao,c1:LComentarios,OpComp:OpComp,c2:LComentarios,Exp2:Expressao)
34 | Input(c1:LComentarios,c2:LComentarios,c3:LComentarios,Tipo:DefTipo,c4:LComentarios,c5:LComentarios)
35 | Print(c1:LComentarios,c2:LComentarios,c3:LComentarios,Expressao:Expressao,c4:LComentarios,c5:LComentarios)
36 | DEnergia(c1:LComentarios,c2:LComentarios,c3:LComentarios,c4:LComentarios)
37 | Expressoes(Expressao*)
38 | Empty()
39
```

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?))) ;
```

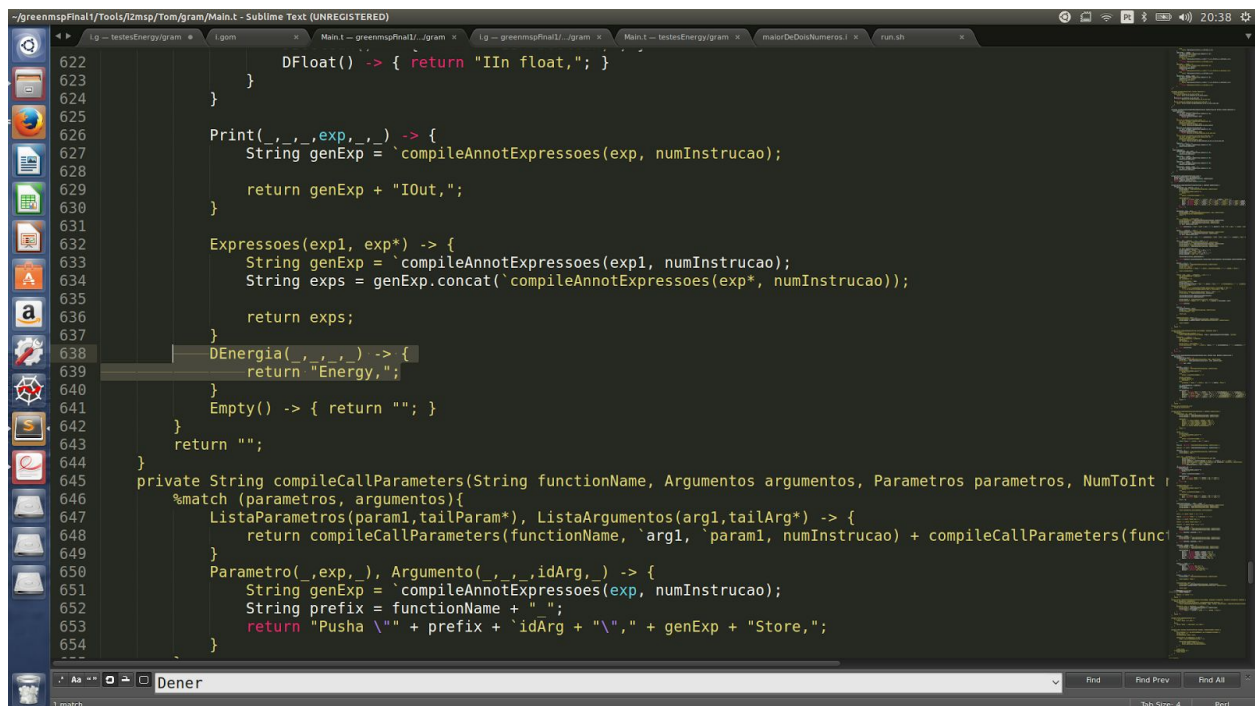
A regra **Exp ^(DEnergia)** do ANTLR vai gerar instrução na estrutura de dados do gom:

```
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()**.

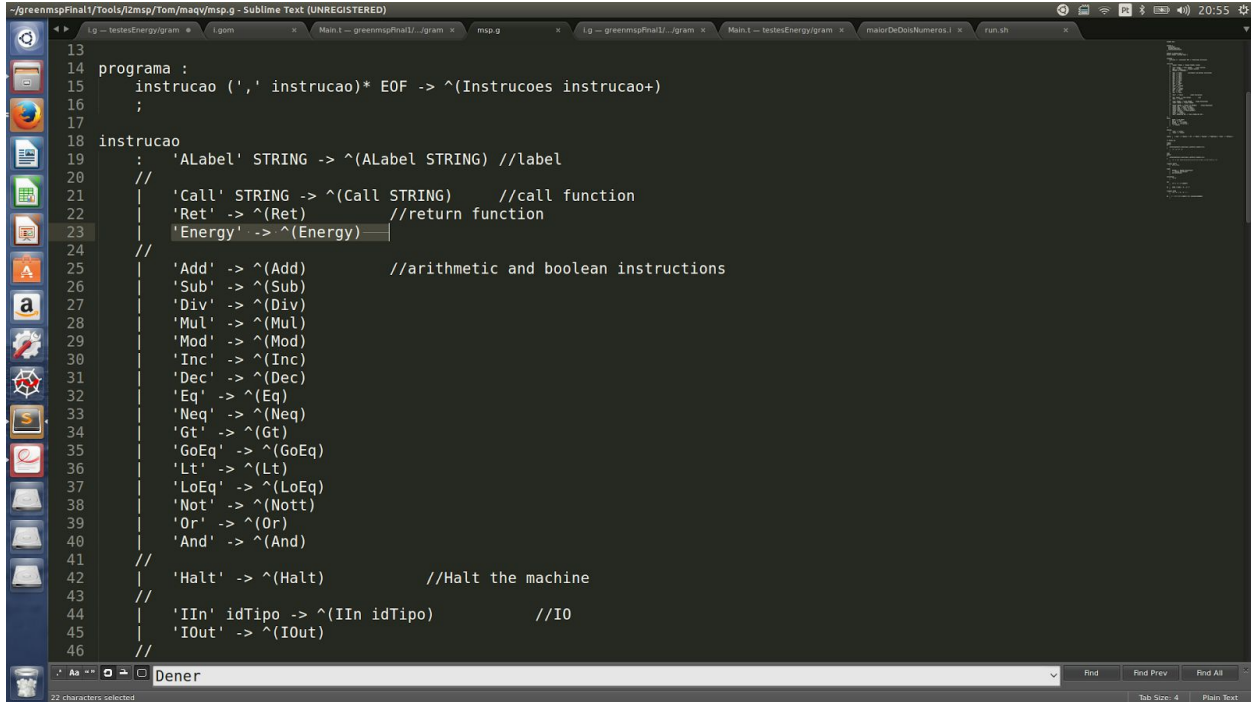
Vai ser enviado para a máquina virtual para posterior processamento o texto “Energy,”.



```
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 i
646 %match (parametros, argumentos){
647     ListaParametros(param1,tailParam*), ListaArgumentos(arg1,tailArg*) -> {
648         return compileCallParameters(functionName, `arg1, `param1, numInstrucao) + compileCallParameters(funci
649     }
650     Parametro(,exp,,) , Argumento(,,idArg,) -> {
651         String genExp = `compileAnnotExpressoes(exp, numInstrucao);
652         String prefix = functionName + "_";
653         return "Pusha \" + prefix + `idArg + "\",\" + genExp + "Store,;";
654     }
655 }
```

Alterações na Máquina Virtual MSP:

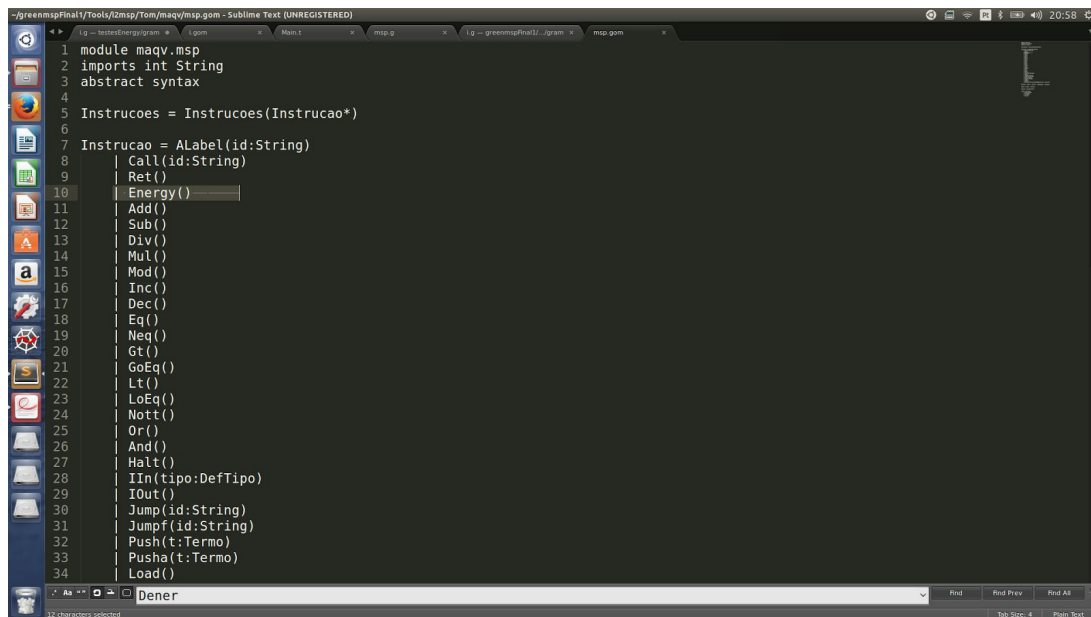
Ficheiro **msp.g** (texto seleccionado no editor):



```
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   //
```

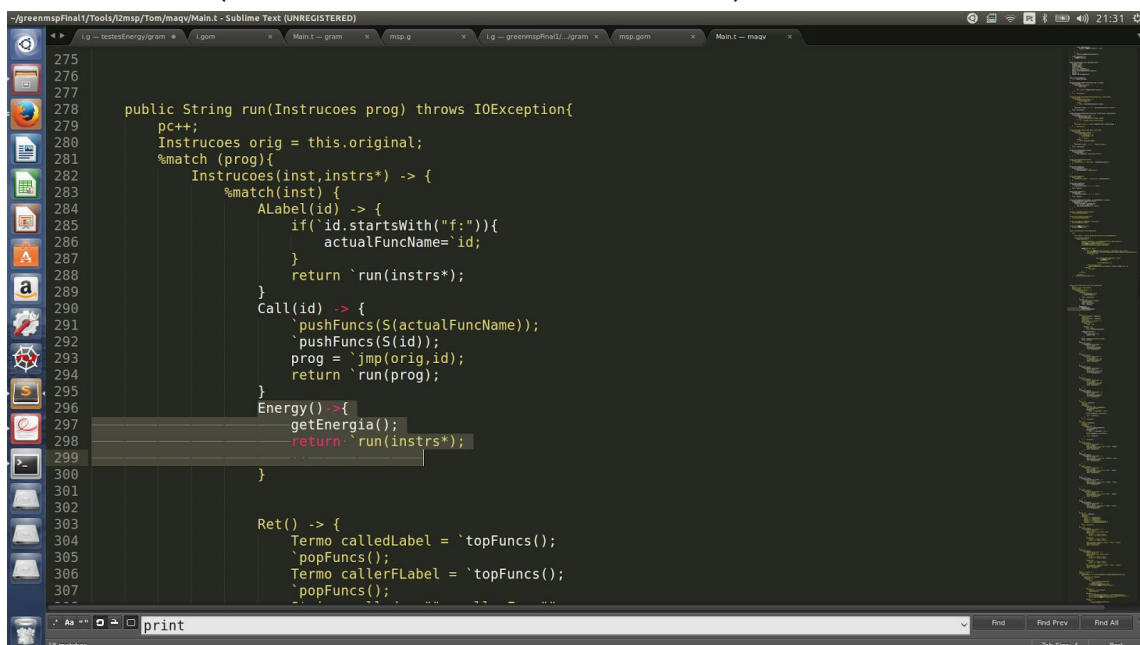
Na página anterior verificamos que no caso de ser encontrada a instrução “energia()” no ficheiro fonte da linguagem C--, seria enviado pelo Processador da Linguagem para a máquina virtual MSP a expressão “Energia,”

No ficheiro **mvp.gom** (texto selecionado no editor de texto):



```
1 module mvp
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  | IOIn()
30  | Jump(id:String)
31  | Jumpf(id:String)
32  | Push(t:Termo)
33  | Pusha(t:Termo)
34  | Load()
35
36 Dener
```

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



```
275
276
277
278 public String run(Instrucoes prog) throws IOException{
279   pc++;
280   Instrucoes orig = this.original;
281   %match (prog){
282     Instrucoes(inst,instrs*) -> {
283       %match(inst) {
284         ALabel(id) -> {
285           if('id.startsWith("f:"){
286             actualFuncName=id;
287           }
288           return `run(instrs*);
289         }
290         Call(id) -> {
291           `pushFuncs(S(actualFuncName));
292           `pushFuncs(S(id));
293           prog = `jmp(orig,id);
294           return `run(prog);
295         }
296         Energy()->{
297           getEnergia();
298           return `run(instrs*);
299         }
300       }
301     }
302   }
303   Ret() -> {
304     Termo calledLabel = `topFuncs();
305     `popFuncs();
306     Termo callerLabel = `topFuncs();
307     `popFuncs();
308   }
309 }
```

É executado o método `energia()` 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*)

Na 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 (na linguagem c--) é igual ao sistema que tínhamos implementado antes, sendo assim, agora é possível correr um programa invocando funções e sem “energia()” e obter energia à mesma. Quando se corre a thread esta também vai fazer a leitura do tempo em ms, para se saber quanto tempo demorou a função a ser executada.

```
class EnergyThread implements Runnable {
    private Thread t;
    private ArrayList<String> arrayEnergy = new ArrayList();

    public EnergyThread( ArrayList<String> array){
        this.arrayEnergy = array;
    }
    public void run() {
        try{
            final Process x = Runtime.getRuntime().exec("sudo java EnergyExample");
            //x.waitFor();
            BufferedReader input = new BufferedReader(new InputStreamReader(x.getInputStream()));
            String line = null;
            try {
                while ((line = input.readLine()) != null){
                    arrayEnergy.add(line);
                }
            } catch (IOException e) {
            }
        } catch (IOException e) {
        }
    }

    public void start () {
        if (t == null){
            t = new Thread (this);
            t.start ();
            while (t.isAlive()) {
                try {
                    t.join();
                } catch (InterruptedException ex) {
                    System.out.println("InterruptedException error");
                }
            }
        }
    }
}
```

Resultados:

Manual:

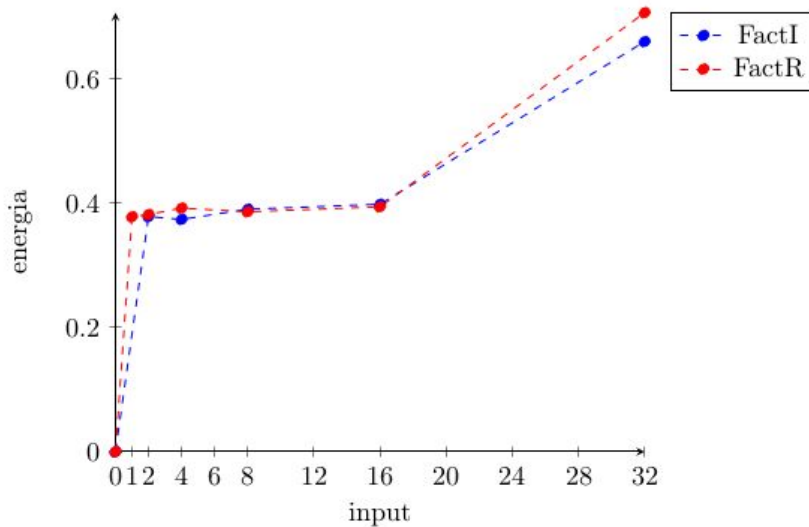
```
dram: 2141.540131 cpu: 24900.20079 package: 43571.289398 time: 1453221348770  
dram: 2141.541397 cpu: 24903.525375 package: 43575.301727 time: 1453221348978  
dram: 2141.541397 cpu: 24907.579102 package: 43580.133789 time: 1453221349197
```

Automático:

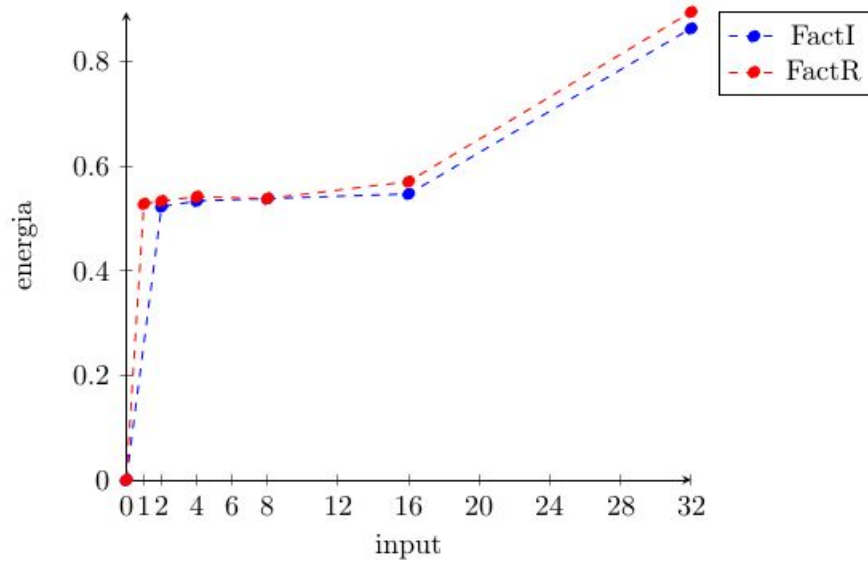
```
f:fatorialI  
dram: 2141.540131 cpu: 24901.218887 package: 43572.546906 time: 1453221348844  
dram: 2141.540756 cpu: 24902.361221 package: 43573.916656 time: 1453221348912  
f:fatorialR  
dram: 2141.541397 cpu: 24904.864822 package: 43576.883484 time: 1453221349046  
dram: 2141.541397 cpu: 24906.377457 package: 43578.69606 time: 1453221349130
```

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

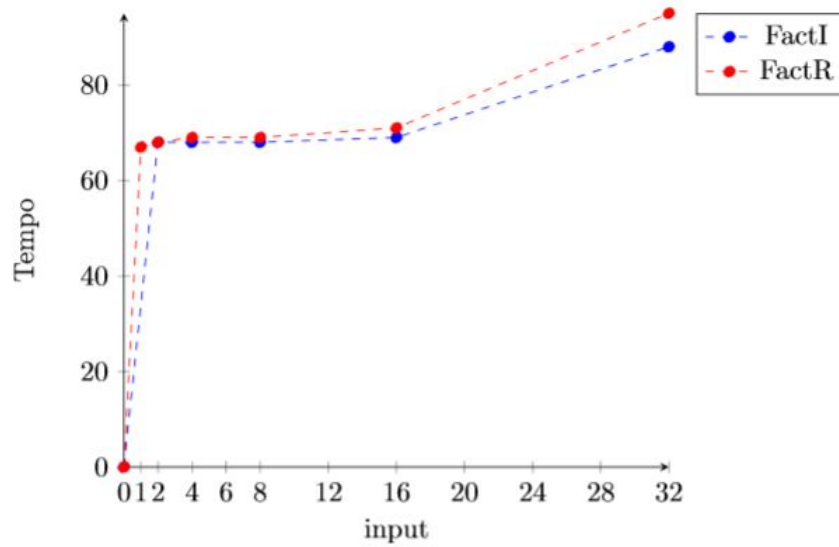
1 CPU



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 energia 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.