Roskilde University and the ENTRA project

John Gallagher

ICT-Energy Community
Workshop
Barcelona
23-25 April, 2014

ENTRAproject.eu
FET MINECC

Roskilde University

- Danish public university, founded 1972
- Consists of six large multi-disciplinary departments
- Our research group is located in CBIT.
  - the department of Communication, Business and Information Technologies
  - PLIS = Programming, Logic and Intelligent Systems (http://plis.ruc.dk).

PLIS projects

- ENTRA: Whole Systems Energy Transparency
- NUSA: Numeric and Symbolic Abstractions for Software Model Checking
- HYLOCORE - Hybrid Logic, Computation and Reasoning
- LoST: Logic-statistic modelling and analysis of biological sequence data (recently ended)
- SIABO: Semantic Information Access through Biomedical Ontologies (recently ended)

ENTRA Partners and Roles

Roskilde University Denmark	Bristol University UK	IMDEA Software Inst., Spain	XMOS Ltd. UK
Coordinator, Static analysis, optimisation	Modelling, toolchains, system design	Static analysis, optimisation, verification	Modelling, HW platform, benchmarking

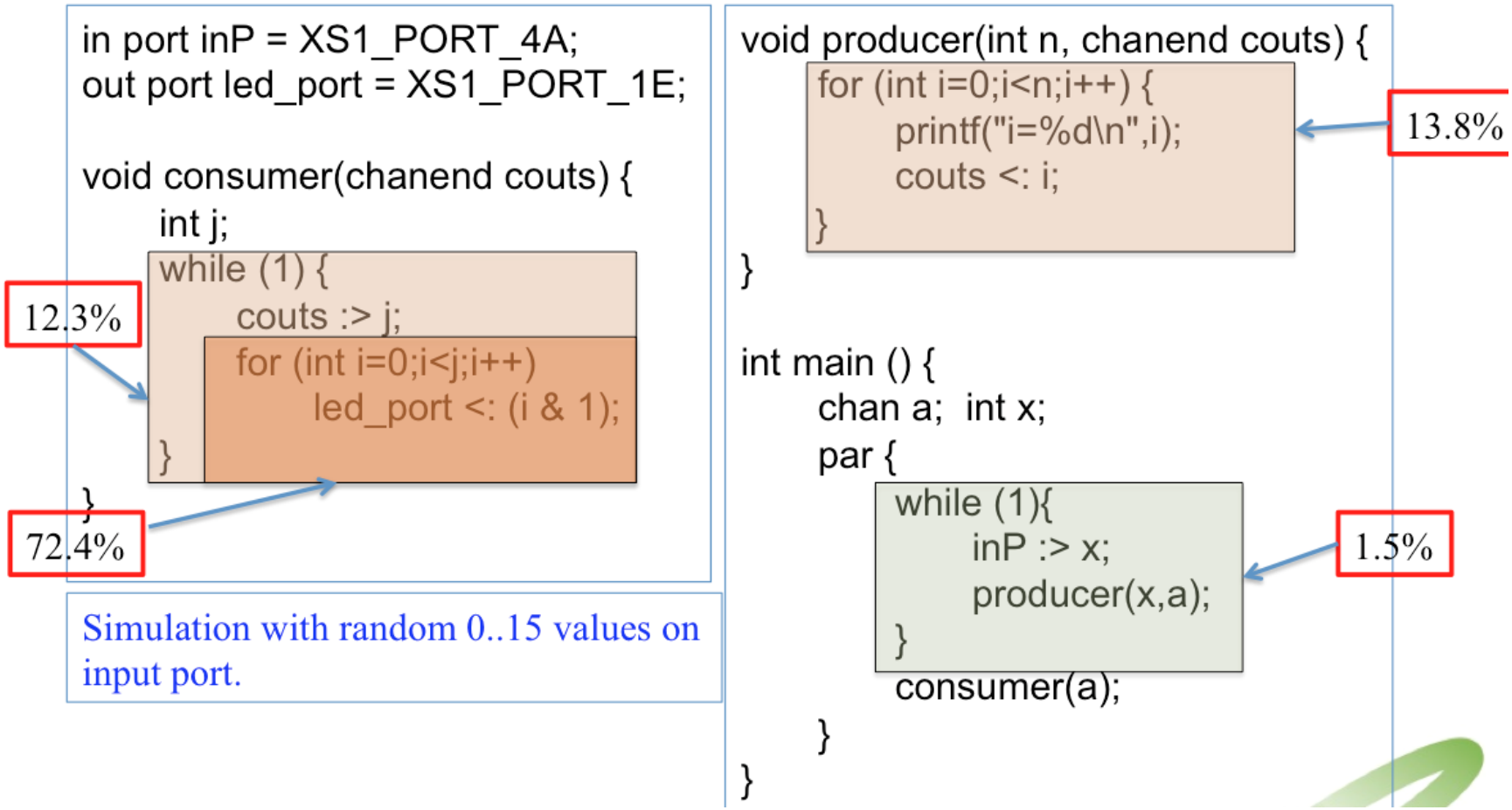
What is Energy Transparency?

· · · information on energy usage is available for programs, without executing them, at all levels from machine code to high-level application code.

Visibility through the system layers

Energy is consumed by the lowest layers of the system stack (hardware)– but is visible to the developer in the higher layers.

Visualizing energy usage



State analysis techniques

Formal techniques for control flow analysis of programs.

- Discover program invariants;
- Verify assertions about the program state.

We model resources (energy, time, statement counts) as "extra variables" in the program state.

Logical modelling of states and transitions

Key issue is a flexible representation of program execution semantics that can link to energy models.

Program state at program point j represented by a predicate $state_j$.

$$state_j(\overbrace{X_1, \ldots, X_{j_n}}^{\text{state variables}}).$$

State transition from program point j to program point k.

$$state_j(\overbrace{X_1, \ldots, X_{j_n}}^{\text{state variables}}) \leftarrow state_k(\overbrace{X'_1, \ldots, X'_{k_n}}^{\text{state variables}}) \wedge c(X_1, \ldots, X_{j_n}, X'_1, \ldots, X'_{k_n}).$$

Extra variables representing resources can be added to the state representation.

$$state_j(\overbrace{X_1, \ldots, X_{j_n}}^{\text{state variables}}, \overbrace{E_0, \ldots, E_k}^{\text{resource variables}}).$$

Example - internal representation

```
int fib(int n){
   int i, Fnew, Fold, temp, ans;
   Fnew = 1; Fold = 0;
11 for (i = 2;i <= n;i++) {
     temp = Fnew; Fnew = Fnew + Fold; Fold = temp;
18 }
   ans = Fnew;
   return ans;
}
```

```
state__18_1(A,B,C,D,Resource1) :-
  A>0,
  E = D+B,
  F = C+1,
  Resource1 = Resource+28
  state__11_1(D,C,E,F,Resource).
```

Internal semantic representation - ISA level code

Translation of xCORE assembly programs to internal constraint logic program form (Horn clauses)

```
1   <fact>:
2   0x01: entsp 0x2
3   0x02: stw   r0, sp[0x1]
4   0x03: ldw   r1, sp[0x1]
5   0x04: ldc   r0, 0x0
6   0x05: lss   r0, r0, r1
7   0x06: bf    r0, <0x08>



11  0x07: bu    <0x10>
12  0x10: ldw   r0, sp[0x1]
13  0x11: sub   r0, r0, 0x1
14  0x12: bl    <fact>

16  0x13: ldw   r1, sp[0x1]
17  0x14: mul   r0, r1, r0
18  0x15: retsp 0x2


21  0x08: mkmsk r0, 0x1
22  0x09: retsp 0x2
```

```
1   fact(R0,R0_3):-
2       entsp(0x2),
3       stw(R0,Sp0x1),
4       ldw(R1,Sp0x1),
5       ldc(R0_1,b0x0),
6       lss(R0_2,bR0_1,R1),
7a      bf(R0_2,0x8),
7b      fact_aux(R0_2,Sp0x1,R0_3,
            R1_1).

10  fact_aux(1,Sp0x1,R0_4,R1):-
11      bu(0x10),
12      ldw(R0_1,Sp0x1),
13      sub(R0_2,R0_1,0x1),
14a     bl(fact),
14b     fact(R0_2,R0_3),
16      ldw(R1,Sp0x1),
17      mul(R0_4,R1,R0_3),
18      retsp(0x2).

20  fact_aux(0,Sp0x1,R0,R1):-
21      mkmsk(R0,0x1),
22      retsp(0x2).
```

Big-step vs. small-step semantics

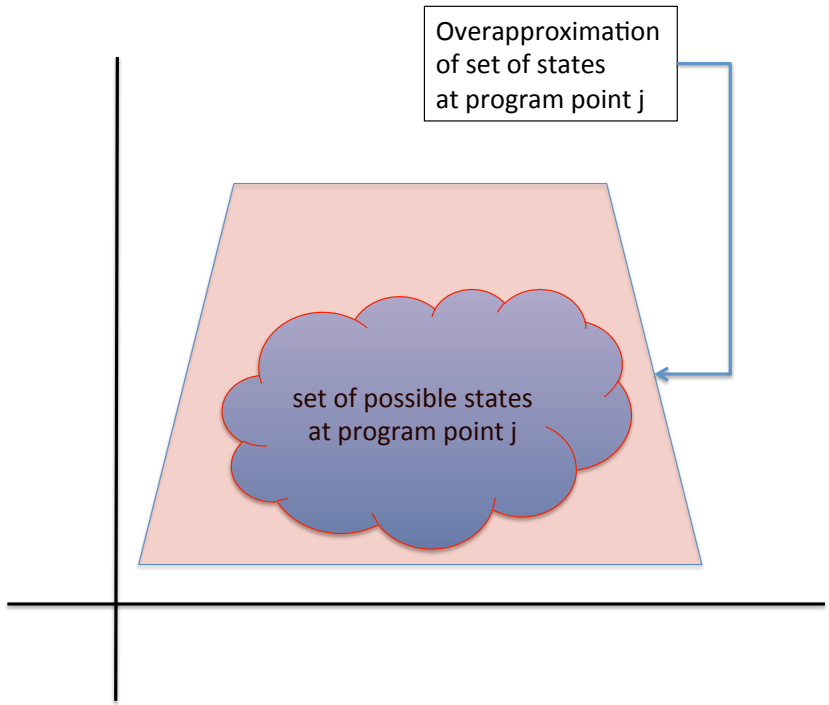
```
big-step
block(S0,Sn) :-
   stmt1(S0,S1),
   stmt2(S1,S2),
   .   .   .
   stmtn(Sn−1,Sn).
```

```
small-step
block(S0) :-
   stmt1(S0).
stm1(S0) :-
   stmt2(S1).
   .   .   .
stmtn-1(Sn−1) :-
   stmtn(Sn).
stmtn(Sn) :-
   .   .   .
```

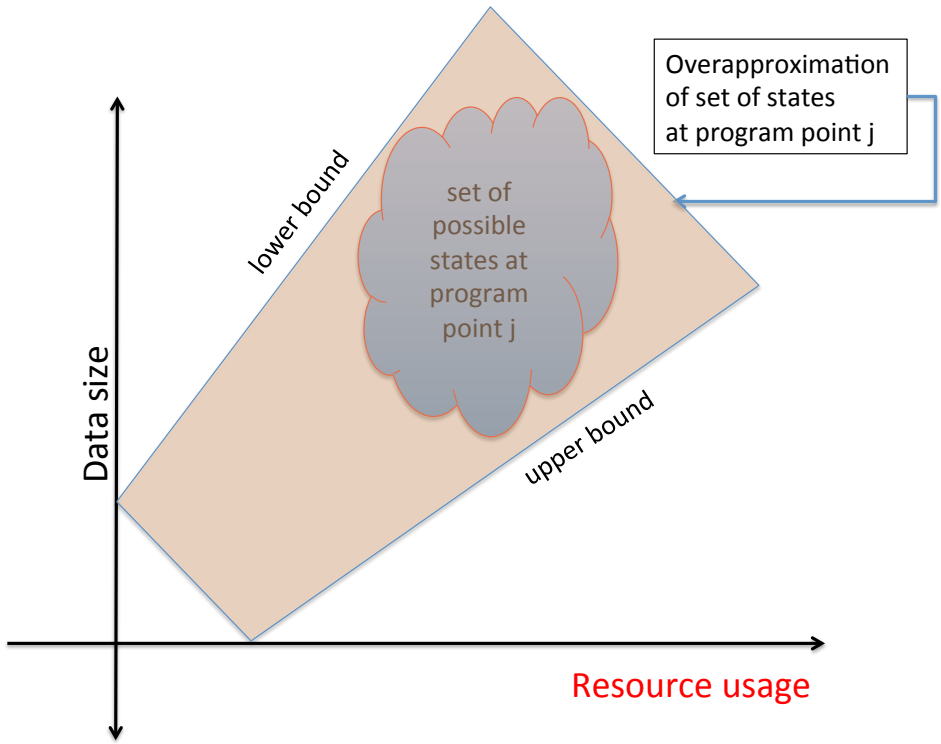
Semantics for concurrency is a particular challenge.

Approximating the run-time state space

Invariants that hold in the over-approximation also hold in the actual states.



Approximating resource usage



Techniques for deriving approximations

- Abstract interpretation. A mathematical framework for systematic abstraction of program semantics. E.g. abstract domain of convex polyhedra (Halbwachs and Cousot). Refinement techniques for iteratively improving approximations.
- Approximate solution of cost relations (e.g. LOPSTR 2013 article by Liqat *et al.*)
- Approximation of probabilistic relationships between data size and resource usage, given a probability distribution P_X on the input data size (Rosendahl & Kirkeby 2014).

$$P_{\mathrm{p}}(z) = \sum_{x \in X \wedge \mathrm{p}(x) = z} P_X(x)$$

Key research areas in ENTRA

Hardware-software energy modelling

- Kerstin's talk yesterday

Program analysis

- Static analysis of code to infer information about its energy/power usage.

Optimization

- program transformations to reduce energy.