Energie analyse

Bernard van Gastel

Open Universiteit
www.ou.nl

Blind spot for software energy consumption

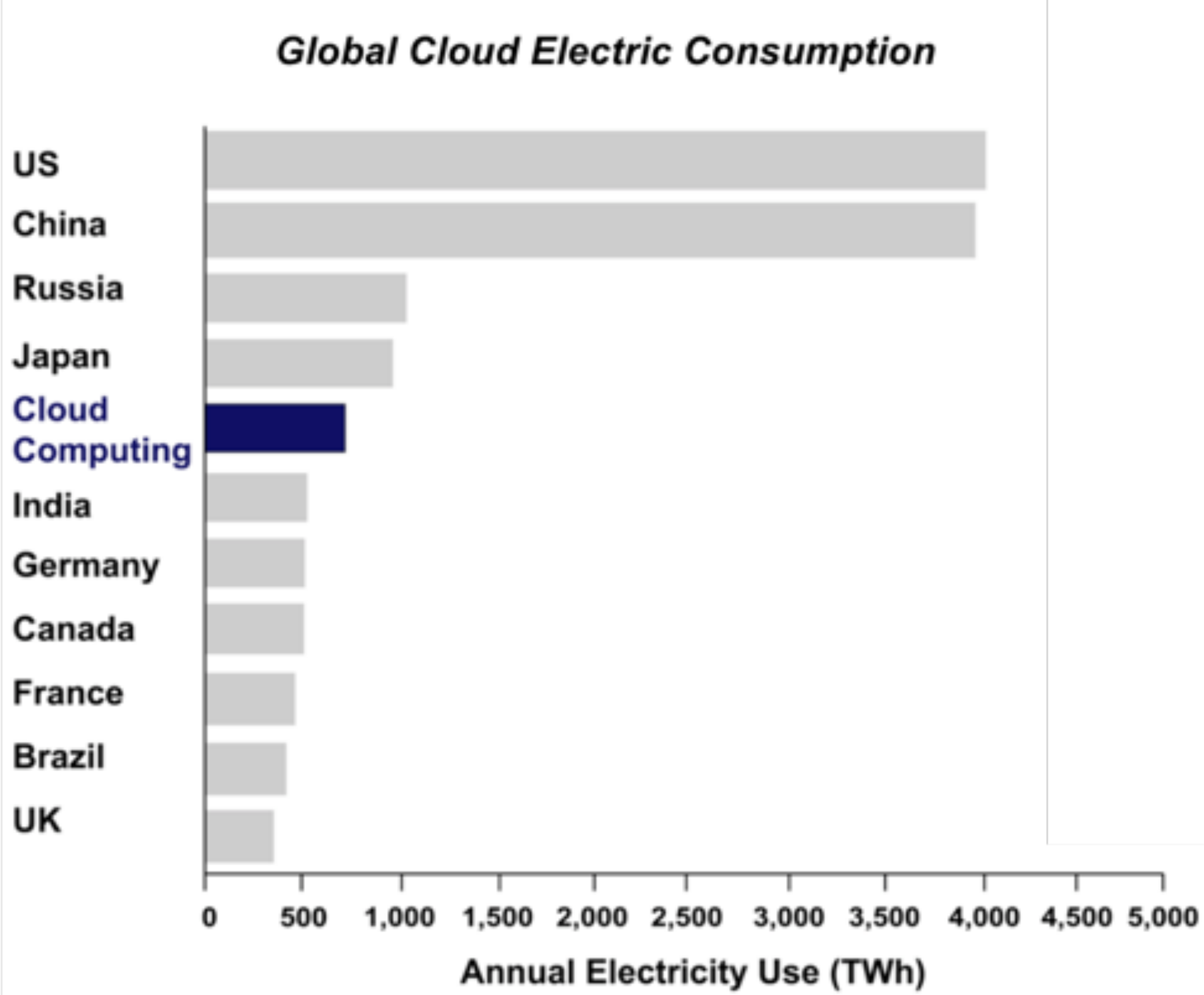
- How much energy does it cost?
  - a Google search
  - your personal cloud
  - big data calculations
  - website
  - BitCoin mining
  - ....
- Did you learn how to write energy efficient programs?
- Did you teach how to write energy efficient programs?

Energy is of primary importance in IT

- Mobile phone models that last longer get better reviews
- Data centres are located where energy is cheap
- ARM market share grew with rising energy prices
- Due to laws and regulations, long-term plan, etc
- Due to hardware limits (to avoid meltdowns, …)

Energy consumption of IT is increasing

- 10% of world-wide energy production........ in 2012!



Source: Greenpeace International, How Clean is Your Cloud, April 2012
Note: Cloud consumption here includes telecommunications infrastructure, but not the entire ICT ecosystem.

Software controls hardware

- Processor accounts for around 5% on modern phones
- Take into account all the components:
  - WiFi, Screen, Audio
  - 3G connection
  - Industrial Motors, Car engines, Auto pilot of airplanes
  - Heating using smart thermostats
  - Robot vacuum cleaner

Relevance

- Computers control the world and hence software controls a large fraction of the energy used in the world!
- Energy is a resource: energy consumption analysis is a form of resource consumption analysis
- Analysis of complete systems is needed:
  software+hardware = control+machine

All kinds of hardware can be controlled

- Many approaches target specific (class of) hardware
- Infeasible to develop analysis for each (class of) component(s)
- We need a **generic approach**

Evaluating energy consumption is time consuming…

- Developing software is an iterative process
- Developing energy efficient software is an iterative process
- Evaluating software for *n* devices, requires *n* test setups
  - e.g. there are over 10000 Android models supported by Google Play
- It is important to get feedback **quickly**….

Our proposed approach

- Start with defining an explicit interface in software
- Define parametric hardware models which are controlled by the interface in the software
- Define exact semantics for input language
- Define type system deriving a the exact energy consumption

Our previous approach

- Used a Hoare Calculus
- Used an over approximation
- Limited applicability of the approach:
  - Problems with reusing derived bounds (e.g. function call)
  - Recursion is not supported
- Needed a pre-analysis
- A new system taking these improvements into account was more natural as a type system, and could derive exact energy consumption (but probably can retain the flexibility to over approximate)

Start with defining an explicit interface in software

- Define a simple language (to start with)
- Explicit control of hardware components

```
⟨Const⟩      ::= '0' | '1' | ... | '9' | ⟨Const⟩⟨Const⟩ | '-' ⟨Const⟩
⟨Id⟩         ::= 'A' | 'a' | 'B' | 'b' | ... | 'Z' | 'z' | ⟨Id⟩⟨Id⟩
⟨Input⟩      ::= '#' ⟨Id⟩
⟨Var,FuncName,Component⟩ ::= ⟨Id⟩
⟨FuncDef⟩    ::= 'function' ⟨FuncName⟩ '(' ⟨Var⟩ ')' 'begin' ⟨Expr⟩ 'end'
⟨Bin-Op⟩     ::= '+' | '-' | '*' | '>' | '>=' | '==' | '!= '| '<=' | '<' | 'and' | 'or'
⟨Expr⟩       ::= ⟨Const⟩ | ⟨Input⟩ | ⟨Var⟩
               | ⟨Var⟩ ':=' ⟨Expr⟩ | ⟨Expr⟩ ⟨Bin-Op⟩ ⟨Expr⟩
               | ⟨Component⟩ '::' ⟨FuncName⟩ '(' ⟨Expr⟩ ')'
               | ⟨FuncName⟩ '(' ⟨Expr⟩ ')'
               | ⟨Statement⟩ ',' ⟨Expr⟩
⟨Statement⟩  ::= 'skip' | ⟨Statement⟩ ';' ⟨Statement⟩ | ⟨Expr⟩
               | 'if' ⟨Expr⟩ 'then' ⟨Statement⟩ 'else' ⟨Statement⟩ 'end'
               | 'repeat' ⟨Expr⟩ 'begin' ⟨Statement⟩ 'end'
               | ⟨FuncDef⟩ ⟨Statement⟩
```

Hardware models

- Each component is modelled by a Finite State Machine
- Models with the same interface can be exchanged
- Each component has functions that operate on the component
- Each component function can modify the state
- Each state has an associated power draw and incidental energy usage
- Can be created using specs of manufacturer and/or measured **dynamically**

Define type system

- Deriving a higher order function that computes the energy consumption
- Depends on input variables, so we use a dependent type system to express all variables in terms of the input
- Because of the nature of a type system, function signatures can be reused.

Example

```
C::on();
repeat #n begin
  C::transmit(#n)
end;
C::off()
```

- Switches a radio device on
- Transmit *n* pieces of information
- Switches the radio off

- Energy model:
  - while switched on the radio has a power draw of `u`
  - each transmit call cost `i` energy, and will take time `t`

- Intuitive energy bound of:

$$n \times (t \times u + i)$$

Example (2)

```
C::on();
repeat #n begin
  C::transmit(#n)
end;
C::off()
```

- The analysis rules derive two bounds: a energy type and a state change type
- Both are higher order functions, having a concrete state as input.
- These results can be combined with $\oplus, \otimes, \ggg$
- The energy bound of $\mathbf{C::transmit(\#n)}$ is:

$$E^{\#n} \oplus (\Sigma^{\#n} \ggg (td_t \oplus i_i)) = td_t \oplus i_i$$

  (the energy costs E of evaluating the argument *plus* the cost of the time dependent energy consumption)

- The resulting state change function is the function composition of the state change function of evaluating the argument and the delta function of C::transmit

Example (3)

```
C::on();
repeat #n begin
  C::transmit(#n)
end;
C::off()
```

- The general approach to analysing a loop is:

$$repeat(T_{bound}, E_{loop}, \Sigma_{loop})$$

- In this case:

$$repeat(n, td_t \oplus i_i, id)$$

- Eliminating the recursive definition of '$repeat$':

$$n \otimes (td_t \oplus i_i)$$

- We can eliminate the higher order functions:

$$n \times (e \times t \times u + i)$$

- Because we know the component is on, we can eliminate e:

$$n \times (t \times u + i)$$

Flexibility

- In principle the type system is exact (assuming exact hardware models). The **repeat** and **if** functions are precise, there is no over approximation. The type system returns a function operating on an environment returning a bound.
- These **repeat** and **if** functions can be given another meaning, together with the basic blocks of the type system, so the type system returns an over approximating function (which yields results quicker) (future work)
- Instead of executing the resulting function on a concrete environment, one can also interpret the result symbolically. This will result in a symbolic bound. Over approximation is a requirement for this. (future work)

Future work

- Add recursion
  - Using a cost-relation (instance of recurrence relation) to solve recursion (and improve the loop)
- Evaluate applicability on large systems
- Ability to analyse `structured parallel programs' (also using dependent types)
- Apply on a real programming language (e.g. by using LLVM IR as analysis input)
- Implement the type system in our prototype

Conclusion

- Presented a type system deriving an energy consumption function with parametric hardware models
- Scalable, by permitting reuse of analysis results.
- Flexible, potentially deriving more bounds
- More elegant *and* concise presentation
- Now self-contained