# Towards Energy Consumption Analysis of Programs at the LLVM IR Layer

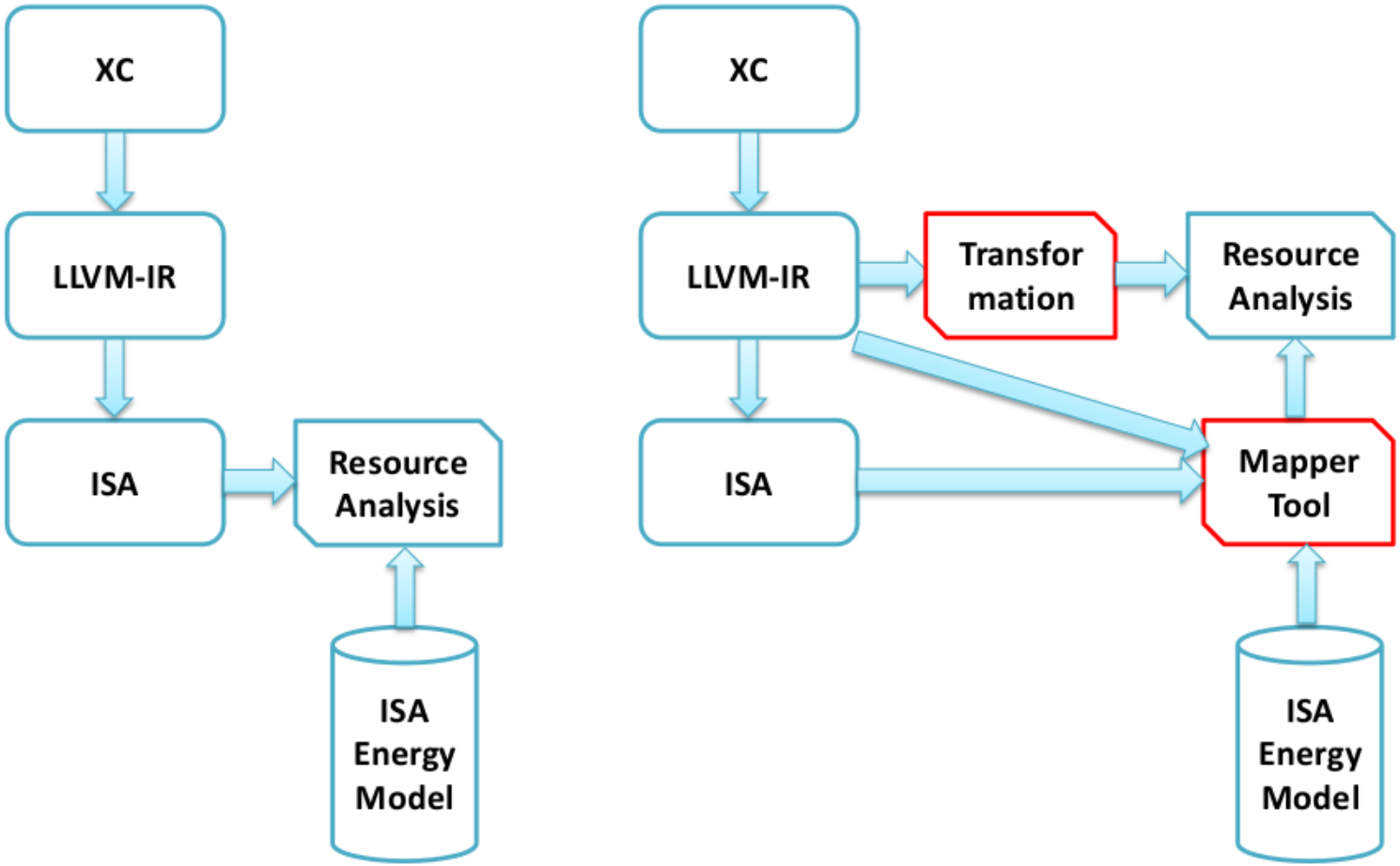
**K. Georgiou[1], U. Liqat[2]**
P. Lopez-Garcia[2], M.V. Hermenegildo[2], K. Eder[1],

[1]University of Bristol
[2]IMDEA Software Institute

April 24, 2014

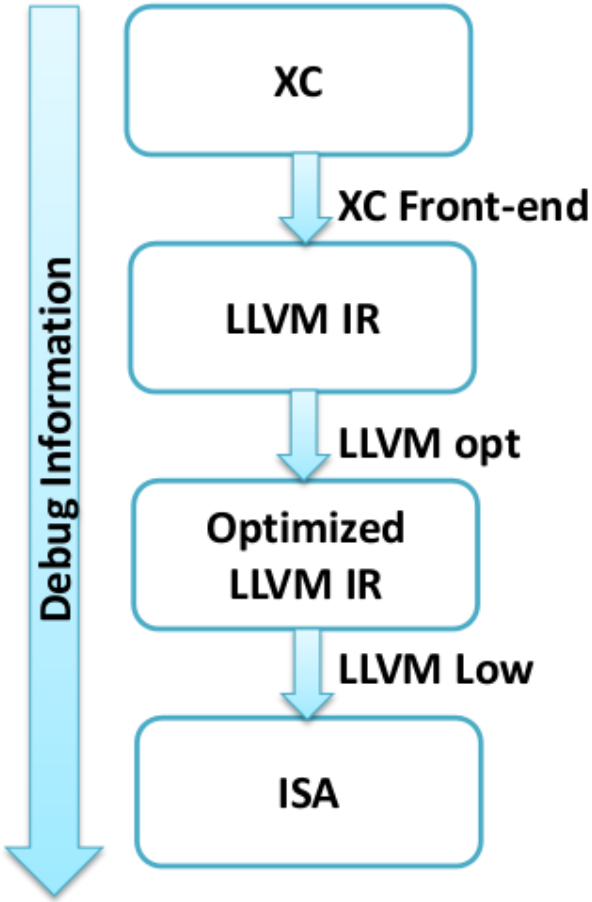
# Overview

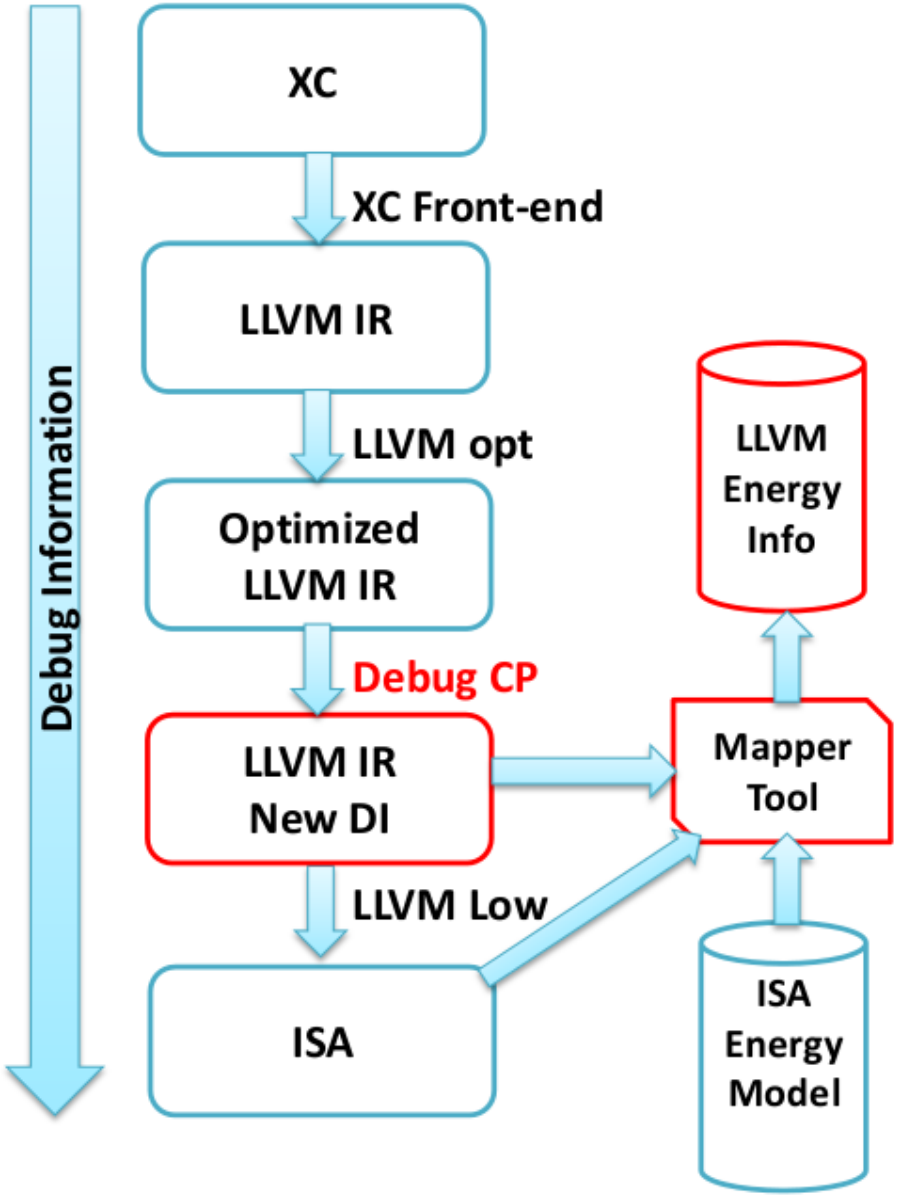
# Motivation



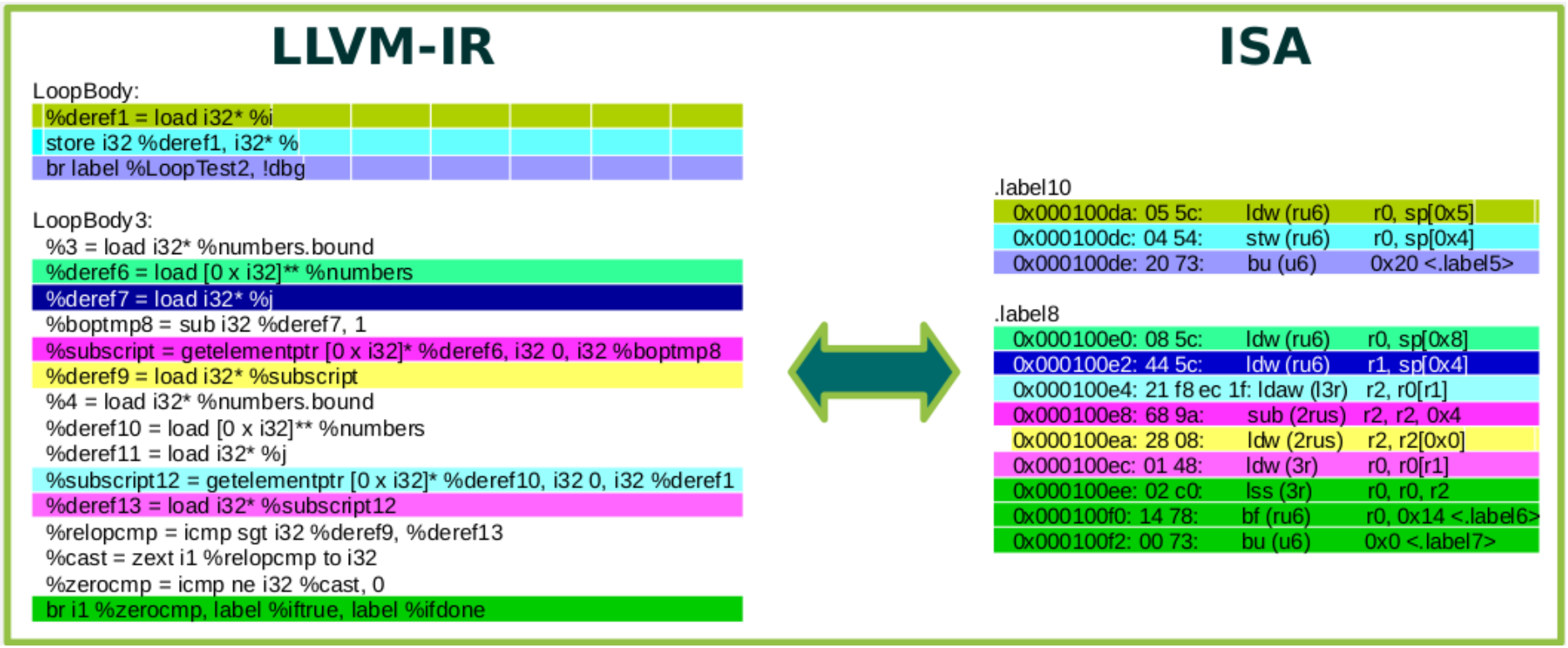

[1] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. Lopez-Garcia, N. Grech, M. Hermenegildo, and K. Eder. Energy Consumption Analysis of Programs based on XMOS ISA-level Models. In Pre-proceedings of the 23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR13), Sept. 2013.

# Mapping Technique



# LLVM-IR/ ISA Mapping Example



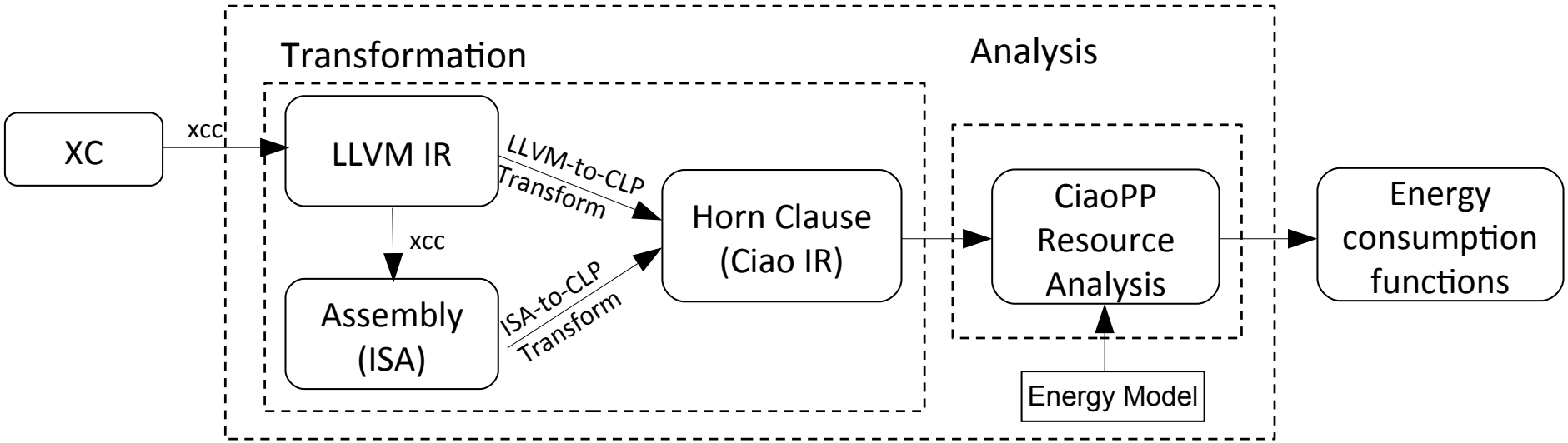
# Mapper Output

```
fibonacci_body {212000} :

    %zerocmp = icmp ne i32 %cast: { } {0}
    br i1 %zerocmp: { bt_ru6, ldc_ru6, bu_u6 } {159000}
    %deref = load i32* %n: { ldw_ru6 } {53000}
    %relopcmp = icmp eq i32 %deref: { } {0}
    %cast = zext i1 %relopcmp to i32: { } {0}
```

```
"fibonacci_body": {
  "instructions": [
    "bt_ru6",
    "ldc_ru6",
    "bu_u6",
    "ldw_ru6"
  ],
  "lineEnds": "13",
  "lineStarts": "13",
  "energy": 212000
},
```

# Transformation based analysis framework




## CiaoPP Resource Usage Analysis

Static analysis that automatically infers both upper and lower bounds on the usage that a program makes of a very general notion of user-definable resource, in this case *energy*.

# Transforming LLVM IR to Horn Clauses

Each LLVM IR block is transformed to *Horn clause* (in Ciao language):

$$< block\_id > (< params >) \ :- \ S_1, \ \ldots \ , S_n.$$

which is the Intermediate representation on which the CiaoPP analysis operates.

- Each LLVM IR instruction is represented by a literal $S_i$.
- Input/Ouput parameters (*params*) to each block are inferred via a fixpoint analysis on the CFG of the program.
- Types are transformed into their counterparts in Ciao language.
- Branching instructions are transformed to calls to other blocks.
  - Branching to multiple blocks is treated as call to a predicate with multiple clauses.

# Example

```
struct mystruct
{
  int x;
  int arr[5];
};
```

```
void print(struct mystruct Arg0[7])
{
  // print contents of input argument
  ...
  ...
}
```

Cost function for the function *print*? :

# Example

```
struct mystruct
{
  int x;
  int arr[5];
};
```

```
void print(struct mystruct Arg0[7])
{
  // print contents of input argument
  ...
  ...
}
```

Cost function for the function *print*? :

$$C_s * length(s) \times C_{s.arr} * length(s.arr) + C_{rest}$$

# Example

```
struct mystruct
{
  int x;
  int arr[5];
};
```

```
void print(struct mystruct Arg0[7])
{
  // print contents of input argument
  ...
  ...
}
```

Cost function for the function *print*? :

$$C_s * length(s) \times C_{s.arr} * length(s.arr) + C_{rest}$$

LLVM IR representation of function *print*:

*define void @print(* $[7 \times \{i32, [5 \times i32]\}]$* * *noalias nocapture) nounwind*

# Example

*define void @print( [7 × {i32, [5 × i32]}]* noalias nocapture) nounwind*

```
:- regtype array1/1.
array1([]).
array1([Elem|T]):-
  struct(Elem),
  array1(T).
```

```
:- regtype struct/1.
struct(mystruct(X,Y)):-
  num(X),
  array2(Y).

:- regtype array2/1.
array2([]).
array2([Elem|T]):-
  num(Elem),
  array2(T).
```

Ciao language declaration of function print:

:- entry print(*Arg*0) : array1(Arg0).
print(*Arg*0):- ...

# Accuracy of Energy Analyses: LLVM vs. ISA layer

| Program | Error vs. HW | | ISA/LLVM |
|---|---|---|---|
| | llvm | isa | |
| fact | 4.5% | 2.86% | 0.94 |
| fibonacci | 11.94% | 5.41% | 0.92 |
| sqr | 9.31% | 1.49% | 0.91 |
| power_of_two | 11.15% | 4.26% | 0.93 |
| reverse | 2.18% | N/A | N/A |
| concat | 8.71% | N/A | N/A |
| mat_mult | 1.47% | N/A | N/A |
| sum_facts | 2.42% | N/A | N/A |
| Average | **6.46%** | **3.50%** | **0.92** |

- ISA analysis estimations are reasonably accurate.
- LLVM estimations are close to ISA estimations.
- A bigger class of programs are analyzable at LLVM-IR layer.

# Future Work

Mapping

- Minimize mapping error
- Obtain an LLVM-IR stand alone energy model
- Map timing information

Resource Analysis

- Extend the analysis for concurrent programs
- Better modeling of memory, e.g Hashed SSA form

# Thank you!

**kyriakos.georgiou@bristol.ac.uk**

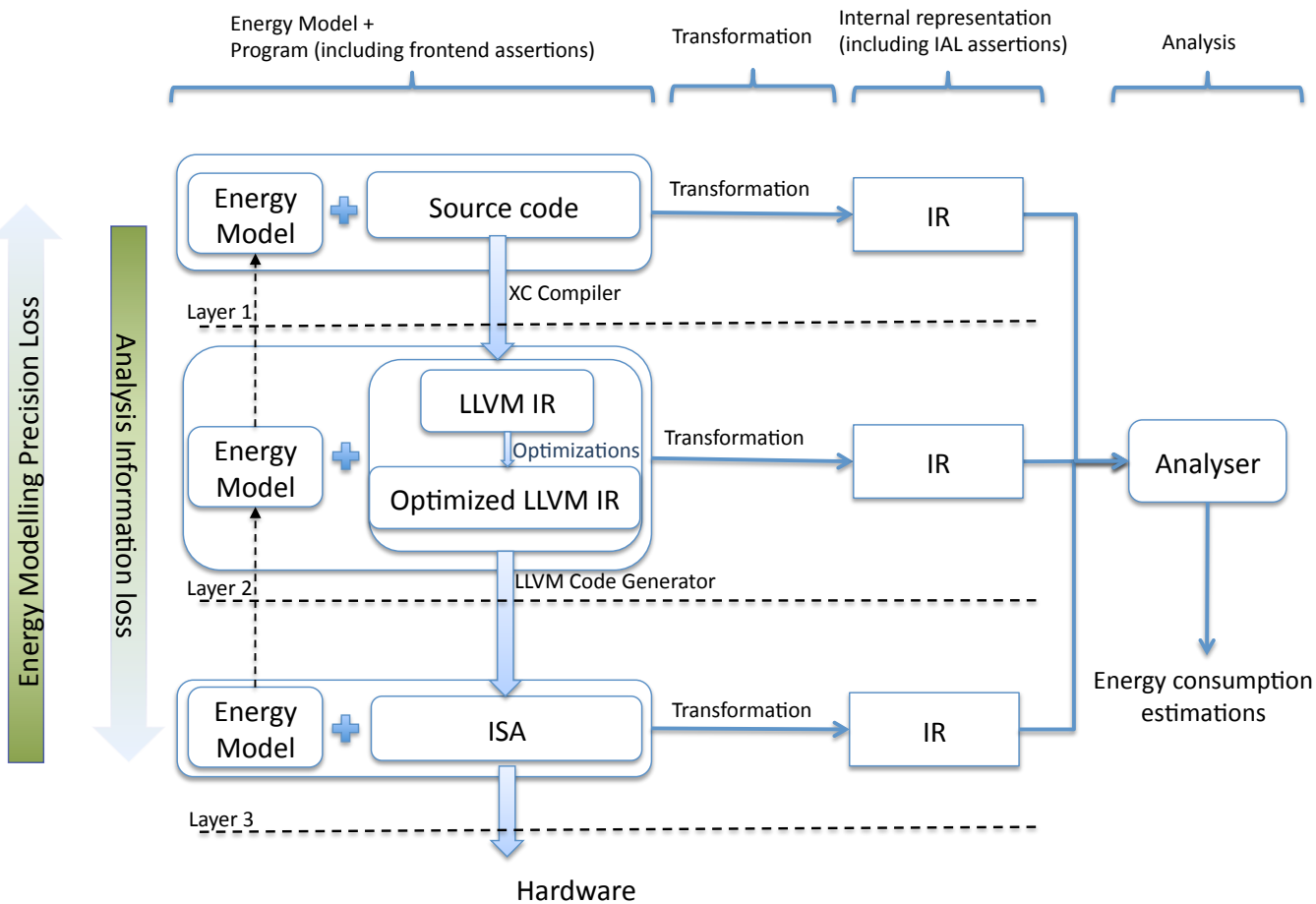
**umer.liqat@imdea.org**

pedro.lopez@imdea.org

manuel.hermenegildo@imdea.org

kerstin.eder@bristol.ac.uk

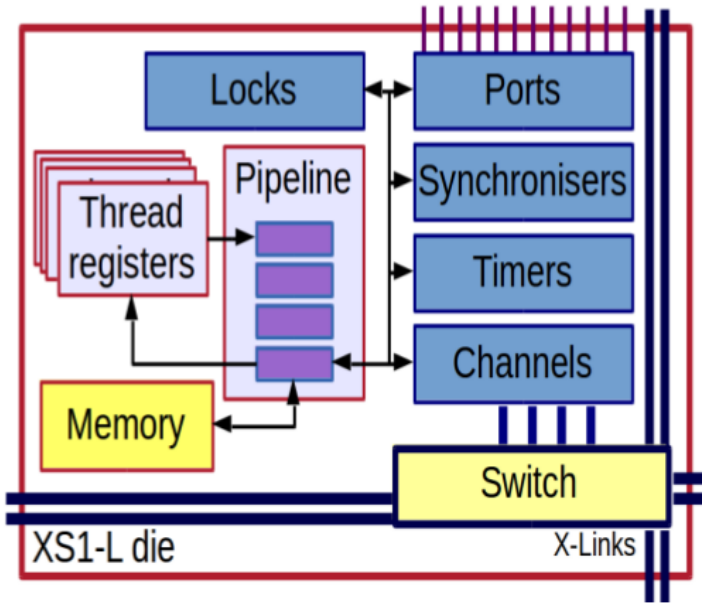
# Summary


Energy Model +
Program (including frontend assertions)
Transformation
Internal representation
(including IAL assertions)
Analysis
Energy Modelling Precision Loss
Analysis Information loss
Energy Model
Source code
Transformation
IR
XC Compiler
Layer 1
LLVM IR
Optimizations
Optimized LLVM IR
Energy Model
Transformation
IR
Analyser
Layer 2
LLVM Code Generator
Energy Model
ISA
Transformation
IR
Energy consumption estimations
Layer 3
Hardware

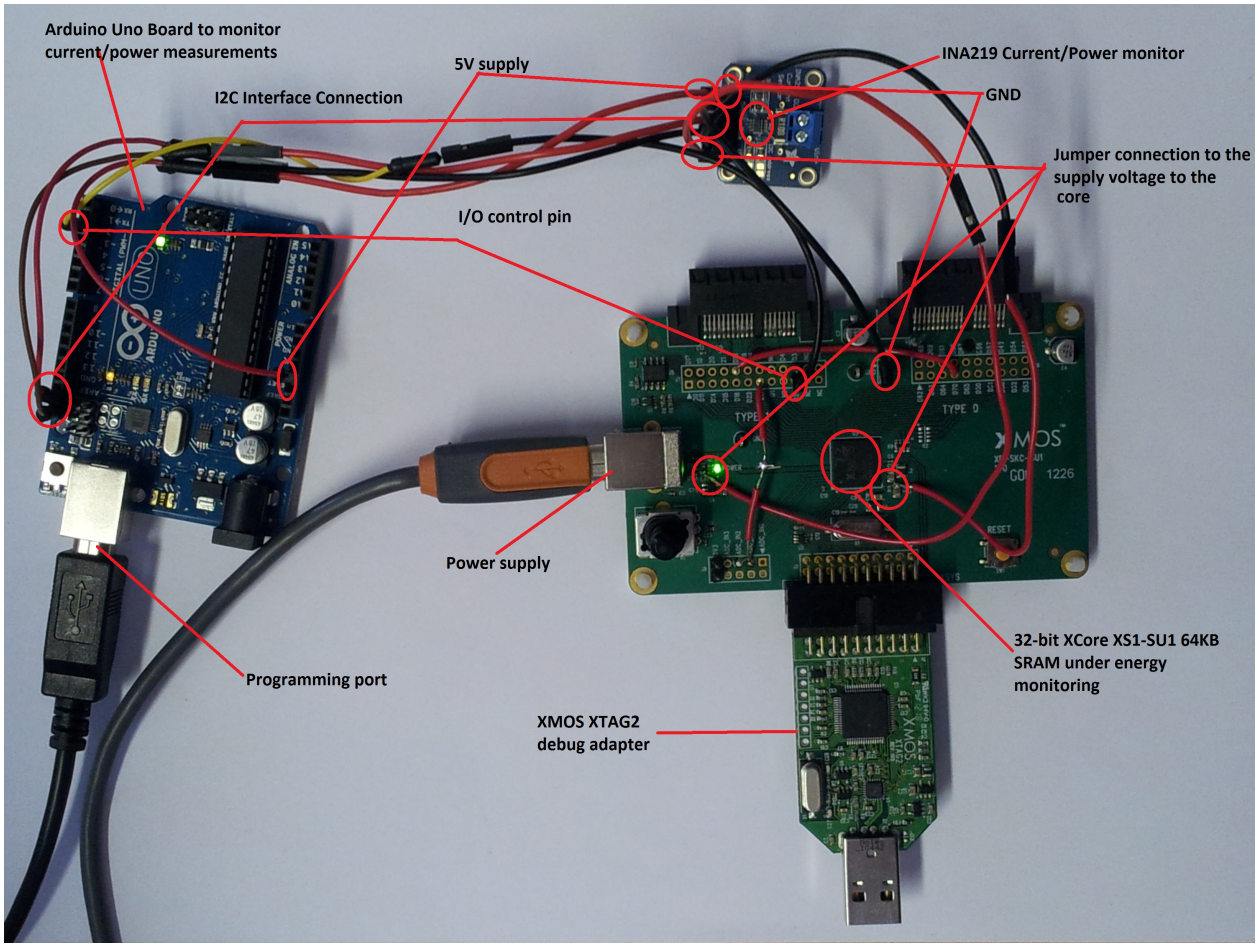
# XMOS Architecture

We focus on the XCORE processor, a 32bit multicore microcontroller designed by XMOS.

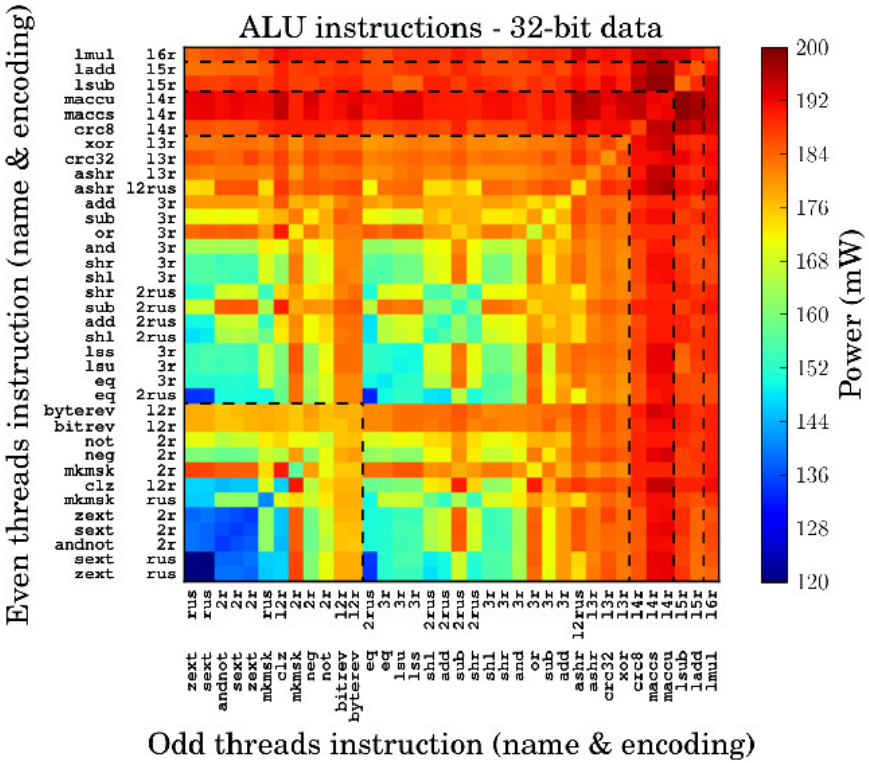
- 64KiB SRAM
- No Cache hierarchies
- Channel based communication between threads and cores
- Up to eight threads per core
- Four stage pipeline
- Simple scheduling (no branch prediction)

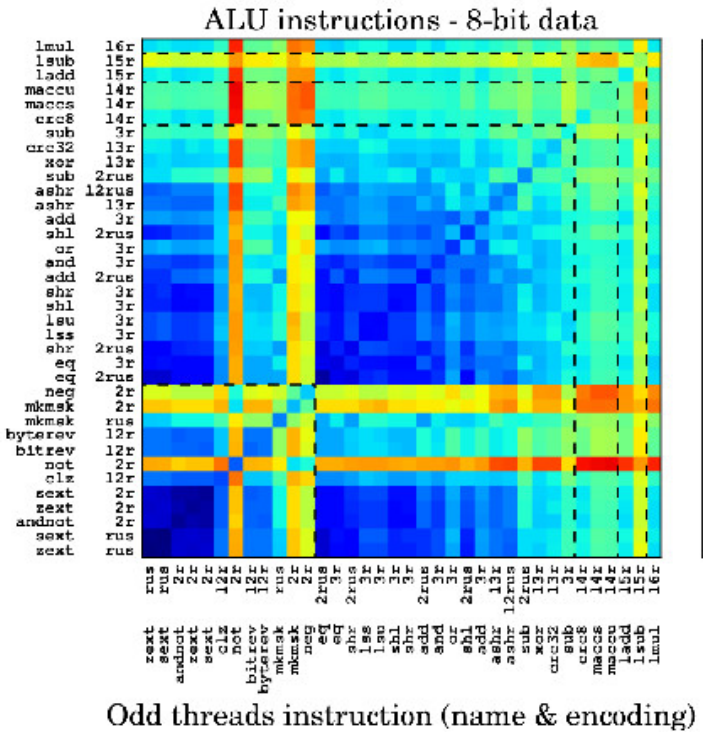


# Energy Measuring Set Up



# ISA Characterization



# Analysing with CiaoPP: Example

```
int sumFact(int Arr[], int NoOfElem) {
    if(size==0)
        return 0;
    return fact(Arr[NoOfElem-1])+
           sumFact(Arr, NoOfElem-1);
}

int fact(int i) {
    if(i<=0)
        return 1;
    return i*fact(i-1);
}
```

# Analysing with CiaoPP: Example

```
int sumFact(int Arr[], int NoOfElem) {
    if(size==0)
       return 0;
    return fact(Arr[NoOfElem-1])+
           sumFact(Arr, NoOfElem-1);
}

int fact(int i) {
    if(i<=0)
       return 1;
    return i*fact(i-1);
}
```

$$Sz^{ub}_{Ret}(Arr, NoOfElem) = Factorial(Sz^{ub}_{Elem}) * Sz^{ub}_{NoOfElem}$$

$$Cost_{arraySum}(Arr, NoOfElem) =$$
$$10 * \underline{Sz^{ub}_{NoOfElem} * Sz^{ub}_{Elem}} + 16 * Sz^{ub}_{NoOfElem} + 67$$

where $Elem \in Arr$