A Multi-level Energy Consumption Static Analysis for Single and Multi-threaded Embedded Programs

Kyriakos Georgiou, Steve Kerrison, Kerstin Eder

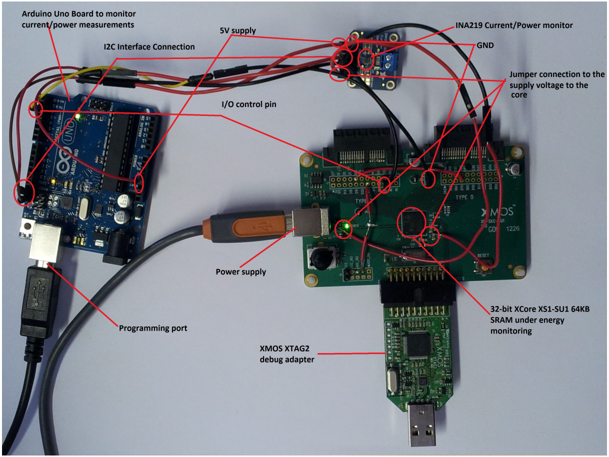
University of Bristol

ENTRA workoshop, Malaga, May 7 2015

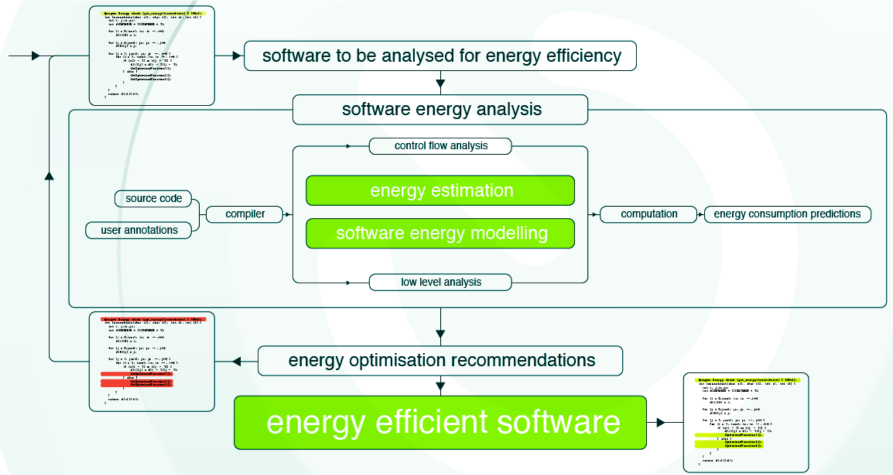
Why is ECSA Important

- Currently, the only way to estimate energy consumption of applications is through physical measurements or simulation.
- Physical energy measurements not accessible to every software developer.
  - Special equipment needed.
  - Advance hardware knowledge needed.
- Simulation can be time consuming.
- Energy estimations during development time key element to energy efficient/ greener software.
  - Developers, will be aware of their code energy efficiency,
  - Compare the energy efficiency between different version of their codes,
  - Meet the targeted energy budget.

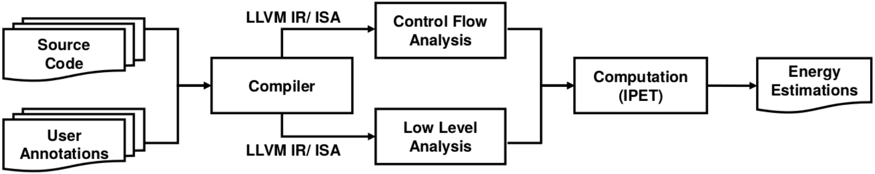
Energy measuring set up.



ECSA vision for Energy Aware Software Development



ECSA Static Analysis in a nutshell



- **Low Level Analysis:** Captures the processor behavior to associate energy costs to atomic units in a program CFG (e.g. ISA, LLVM-IR instructions)
- **Control Flow Analysis:** Captures the dynamic behavior of the program (e.g. loop, recursion detection and bounding)
- **Computation:** Use of Implicit Path Enumeration Technique. The task of retrieving energy consumption estimations is mapped to an ILP system. Solving the ILP system to retrieve the bounds.

What is currently available?

State of the art:

- **Average Case Energy Models**
- **Worst Case Resource Static Analysis**

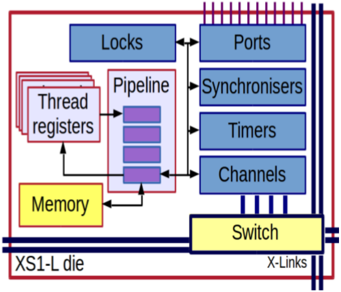
What can we get out of this???

Low Level Analysis

XMOS XS1 Architecture

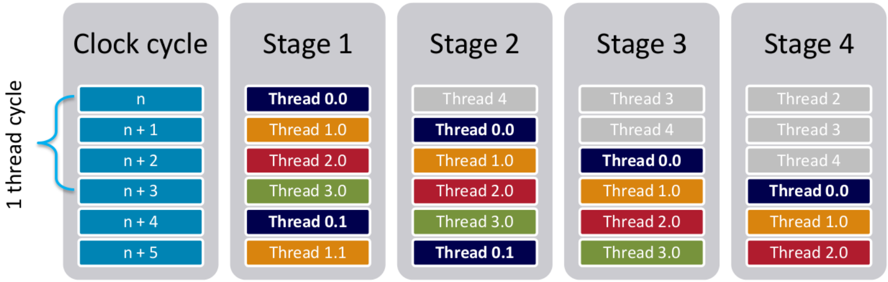
We focus on the XCORE processor, a 32bit multicore microcontroller designed by XMOS.

- 64KiB SRAM
- No Cache hierarchies
- Channel based communication between threads and cores
- Instructions dedicated to comms & I/O
  - Not memory mapped
- Peripherals: Software defined interfaces
- Event driven, no idle loops



XMOS XS1 threads/pipeline (1/2)

- Up to eight threads per core
- Four stage pipeline
- Simple scheduling (no branch prediction)
- At 500MHz, 125MIPS per thread for ¡= 4 threads

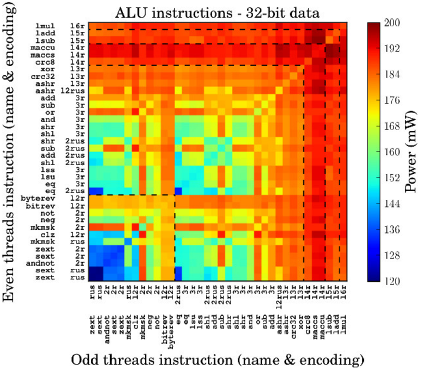


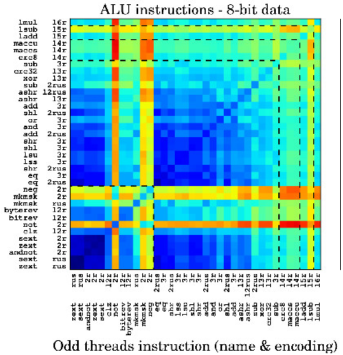
ISA Energy Characterization

$$P_{dyn} = (C_{idle} + (C_{instr} \times S_N \times O)) \times V^2 \times F$$

- ISA based characterization
- Multi-threaded energy model
- Complete instruction set
  - With regression-tree capturing harder to reach instructions
- Voltage/frequency parameterization

ISA Energy Characterization





ISA Instruction Time Cost

- One ISA instruction needs 4 clock cycles to complete$*$
- Cycle time

$$T_{clk} = 1/F$$

- Instruction Time

$$I_t = T_{clk} * 4$$

- e.g. 400 MHz $\Rightarrow I_t = 10ns$

$*$**up to four threads**

ISA Instruction Time Cost Exceptions

- Division
- Communication Time is constant on the same core
- Communication Time btwn cores
- Input output on ports time may vary

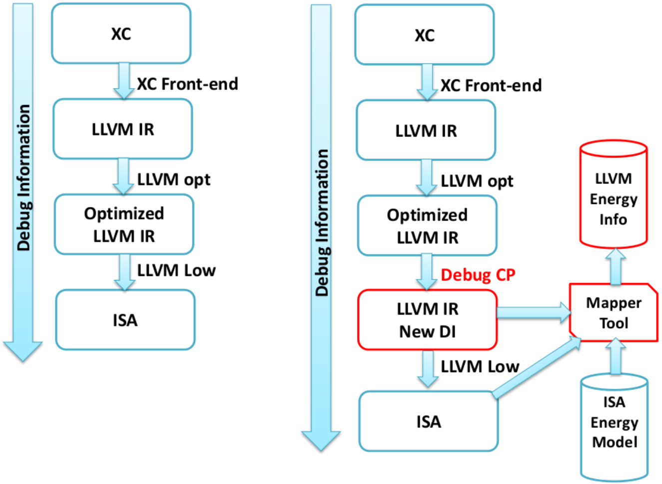
ECSA on a higher Level than ISA

Our existing energy consumption model is on the ISA level.

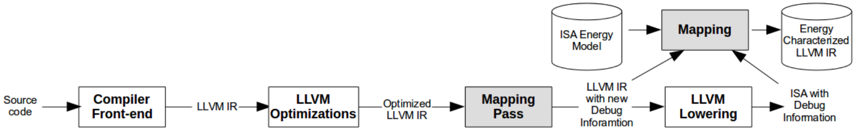
LLVM is a common optimizer and code emitter.

- Need of a good correlation btwn LLVMIR /ISA to be able to transfer info btwn them
- LLVMIR is the optimum place for resource analysis and energy optimizations
- Applicable to many architectures
- All the information needed for the resource analysis are preserved
- LLVMIR is closer to the source code than the ISA level

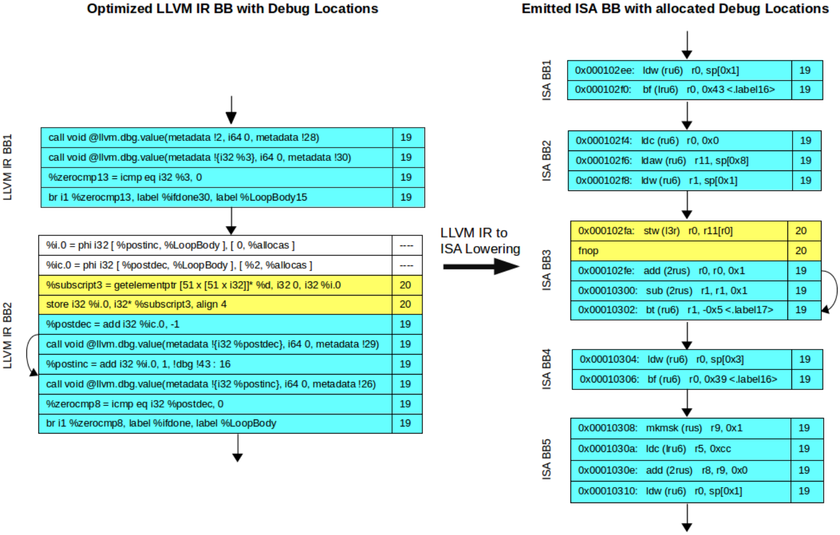
Mapping Technique



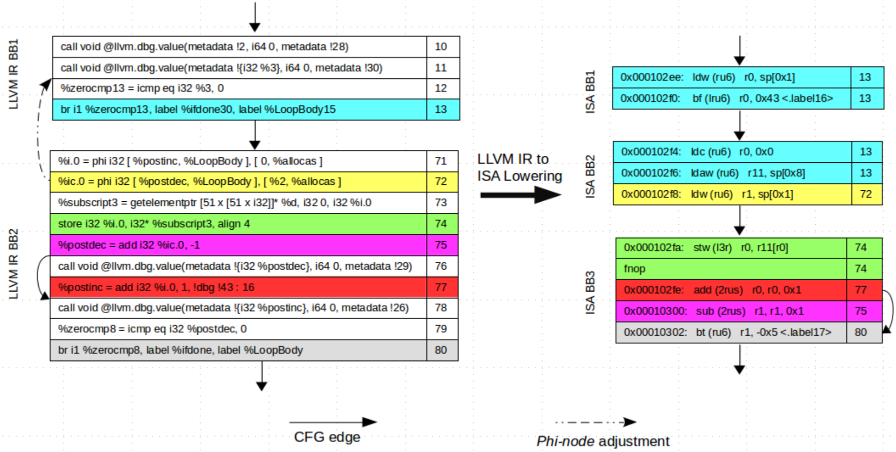
Mapping Technique



LLVM-IR/ ISA Mapping Example



LLVM-IR/ ISA Mapping Example



Control Flow Analysisn and Computation

Implicit Path Enumeration (IPE)

- Very popular technique for WCET calculation
- **It expresses the search of the WCET as an Integer Linear Programming problem where the execution time is to be maximized under some constraints on the execution counts of the basic blocks**
- **The worst case execution path is defined by the set of blocks with their respective execution counts but not the order which they are executed**

Integer Linear Programming Formulation(ILP) 1

Objective Function:

- Let x_i be the number of times the basic block B_i is executed when the program takes the maximum time to complete
- Let c_i be the time cost of the basic block B_i
- If N is the number of basic blocks in the program the WCET is given by the max value of the expression:

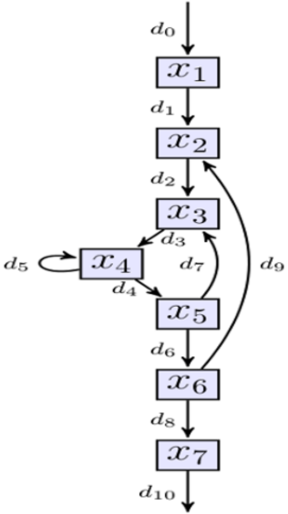
$$\sum_{i=1}^{N} c_i x_i \tag{1}$$

*Note: For Xcore, c_i is constant over all possible times the B_i is executed assumed no unpredictable ports IO or communication happening

ILP Program Structural Constraints

- Can be extracted automatically from the program's Control Flow Graph (CFG)
- In the CFG we label the edges with variables d_i and basic blocks with x_i variables
- This variables represent the times those edges and basics blocks are exercised during the program execution
- The constraints can be deduced from the CFG as follows: **At each node, the execution count of the basic block must be equal to both the sum of the control flow going into it and the sum of the control flow going out from it**

ILP Program Structural Constraints Example



$$d_0 = 1 \quad (2)$$

$$x_1 = d_1 = d_0 \quad (3)$$

$$x_2 = d_1 + d_9 = d_2 \quad (4)$$

$$x_3 = d_2 + d_7 = d_3 \quad (5)$$

$$x_4 = d_3 + d_5 = d_4 + d_5 \quad (6)$$

$$x_5 = d_4 = d_6 + d_7 \quad (7)$$

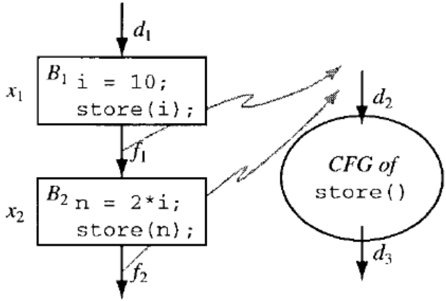
$$x_6 = d_6 = d_8 + d_9 \quad (8)$$

$$x_7 = d_8 = d_{10} \quad (9)$$

$$d_{10} = 1 \quad (10)$$

ILP Program S. Constraints Function Call Example

f-edges treated similar to d-edges



$$x_1 = d_1 = f_1$$

$$x_2 = f_1 = f_2$$

$$d_2 = f_1 + f_2$$

ILP Program Functional Constraints

- Constraints used to denote loop bounds and other path information that depend on the functionality of the program (Data Flow Analysis can minimize user input)

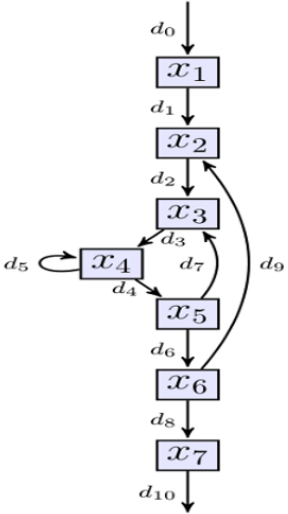
- Minimum requirement from programmer to set the loop bounds

- More functional constraints from the user can help to get tighter bounds

JpegDCT Code

```
void jpegdct(short d[], short r[])
{
    long int t[12];
    int v=0;
    short i, j, k, m, n, p, ic, ik;
    for (ik=2; ik; ik--) {
      for (i = 8; i; i--, v+=8) {
           for (j = 3; j>=0; j--) {
                // some code
           }
         // some code
      }
    }
}
```

ILP Program F. Constraints Loop Bounds Example



$$1x1 <= x2 \quad (11)$$

$$x2 <= 2x1 \quad (12)$$

$$1x2 <= x3 \quad (13)$$

$$x3 <= 8x2 \quad (14)$$

$$1x3 <= x4 \quad (15)$$

$$x4 <= 4x3 \quad (16)$$

ILP Solving Example

$max : b_1 * x1 + b_2 * x2 + b_3 * x3 + b_4 * x4 + b_5 * x5 + b_6 * x_6 + b_7 * x_7$

$$
\begin{aligned}
&d_0 = 1 \\
&x_1 = d_1 = d_0 \\
&x_2 = d_1 + d_9 = d_2 \\
&x_3 = d_2 + d_7 = d_3 \\
&x_4 = d_3 + d_5 = d_4 + d_5 \\
&x_5 = d_4 = d_6 + d_7 \\
&x_6 = d_6 = d_8 + d_9 \\
&x_7 = d_8 = d_{10} \\
&d_{10} = 1
\end{aligned}
$$

$$
\begin{aligned}
&1x1 <= x2 \\
&x2 <= 2x1 \\
&1x2 <= x3 \\
&x3 <= 8x2 \\
&1x3 <= x4 \\
&x4 <= 4x3
\end{aligned}
$$

- Solve this by lp_solver: standard linux pri-installed package
- Complexity: NP complete, although most of the cases it collapses to LP which can be solved in polynomial time

Results

Single-threaded Benchmarks Results

Benchmark	T vs HW	ISA SA vs HW	LLVM IR SA vs HW	ISA SA vs T	P. EC
Base64	-2.67%	-2.52%	16.69%	0.15%	✓
Mac	-3.38%	-3.26%	-3.26%	0.12%	✓
Levenshtein	-1.87%	-0.83%	2.24%	1.04%	✓
Radix4Div	-7.50%	57.89%	60.39%	65.39%	
B. Radix4Div	-7.99%	33.44%	34.84%	41.43%	
Cnt	14.31%	14.23%	14.55%	0.08%	✓
Dijkstra	-4.24%	34.55%	38.36%	38.79%	
Statistics	-2.98%	-2.79%	-5.93%	0.19%	
Fir	-16.00%	-12.17%	-10.24%	3.83%	
SFloatAdd32bit	-7.59%	29.33%	29.42%	36.92%	
SFloatSub32bit	-7.54%	35.58%	36.36%	43.12	
MatMul	-1.28%	-0.88%	-1.21%	0.41%	✓
Biquad	-3.61%	8.69%	7.53%	12.3%	
Jpegdct	-2.61%	-2.11%	-2.40%	0.50%	

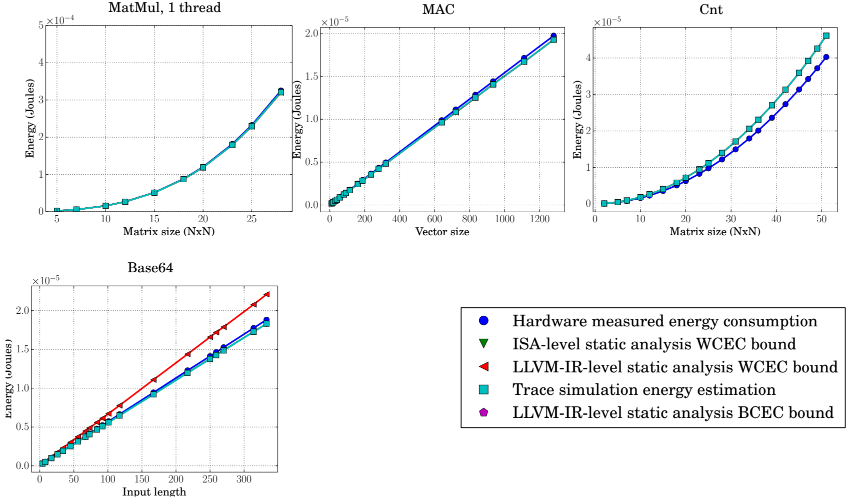
T: Trace Simulation **HW**: Hardware Measurements **ISA SA**: ISA EC Static Analysis **LLVM IR SA**: LLVM IR EC Static Analysis.

Multi-threaded Benchmarks Results

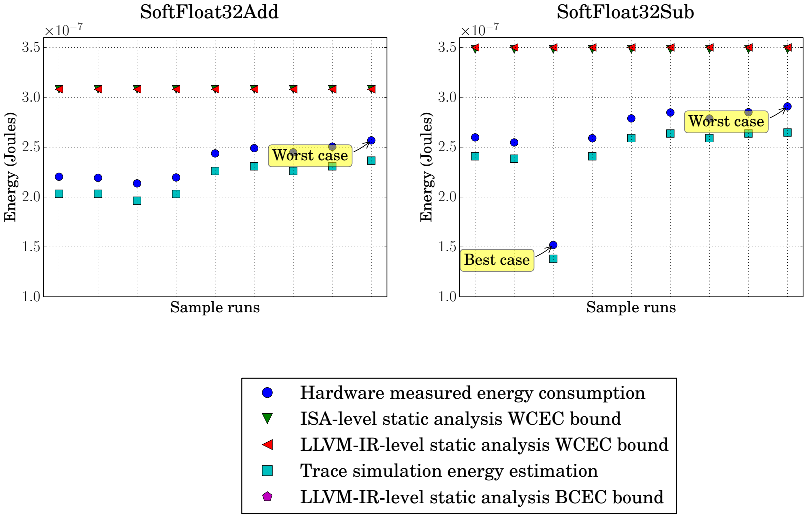
Benchmark	T vs HW	ISA SA vs HW	LLVM IR SA vs HW	ISA SA vs T	P. EC
MatMul	-1.28%	-0.88%	-1.21%	0.41%	✓
MatMul_2T	-12.88%	-1.16%	-0.59%	11.72%	✓
MatMul_4T	-0.84%	11.14%	11.77%	12.09%	✓
Biquad	-3.61%	8.69%	7.53%	12.3%	
Biquad_2T	-9.31%	0.47%	0.41%	9.78%	
Biquad_4T	-5.60%	3.88%	4.35%	9.48%	
Jpegdct	-2.61%	-2.11%	-2.40%	0.50%	
Jpegdct_2T	-6.18%	-5.37%	-6.70%	0.81%	
Jpegdct_4T	-1.06%	-0.03%	-1.97%	1.03%	

T: Trace Simulation **HW**: Hardware Measurements **ISA SA**: ISA EC Static Analysis **LLVM IR SA**: LLVM IR EC Static Analysis.

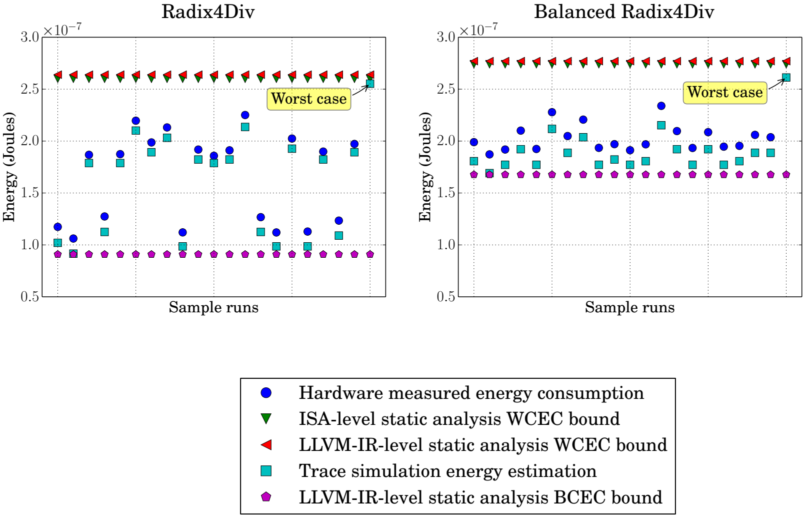
Single-threaded Benchmarks



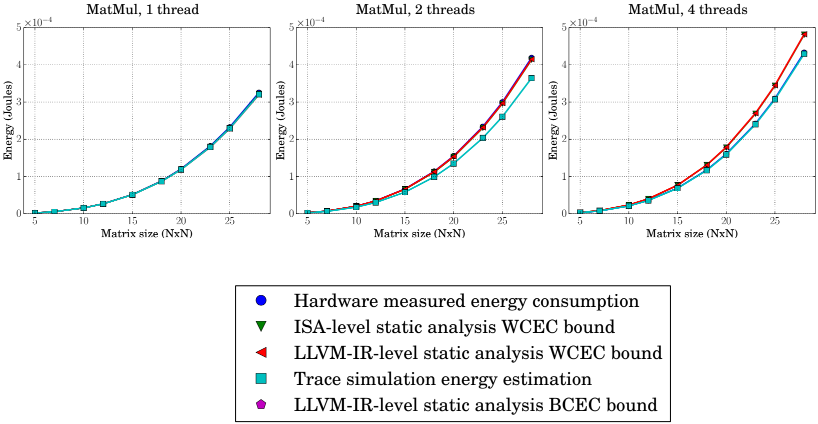
Single-threaded Benchmarks



Single-threaded Benchmarks



Multi-threaded Benchmarks



Energy consumption trends for parametric benchmarks, using regression analysis

Benchmark	Regression Analysis (nJ)	x
Base64	$f(x) = 54.9x + 62.3$	string length
Mac	$f(x) = 15x + 21.1$	length of two vectors
Cnt	$f(x) = 2.4x^3 + 17.6x^2 + 5.7x + 34.5$	matrix size
MatMul	$f(x) = 14x^3 + 17.1x^2 + 4.3x + 34$	size of square matrices
MatMul_2T	$f(x) = 18.1x^3 + 20.3x^2 + 5.7x + 112$	size of square matrices
MatMul_4T	$f(x) = 21x^3 + 23.3x^2 + 7.1x + 213.1$	size of square matrices

- Programmers/ users can predict a program's energy consumption under specific parameter values
- Embedding such equations into an operating system (e.g. library function calls), can enable energy aware decisions:
  - for scheduling tasks
  - checking if the remaining energy budget is adequate to complete a task
  - downgrade the quality of service and complete the task with less energy

Future work

- Extend the analysis to programs with comms & I/O.
- Use more sophisticated path and data flow analysis to extract tighter bounds.
- Use the retrieved energy consumptions equations in an OS for real time energy aware decisions.
- Extend the analysis to other architectures (Cortex M series).

Thank you!

Questions?

kyriakos.georgiou@bristol.ac.uk
steve.kerrison@bristol.ac.uk
kerstin.eder@bristol.ac.uk