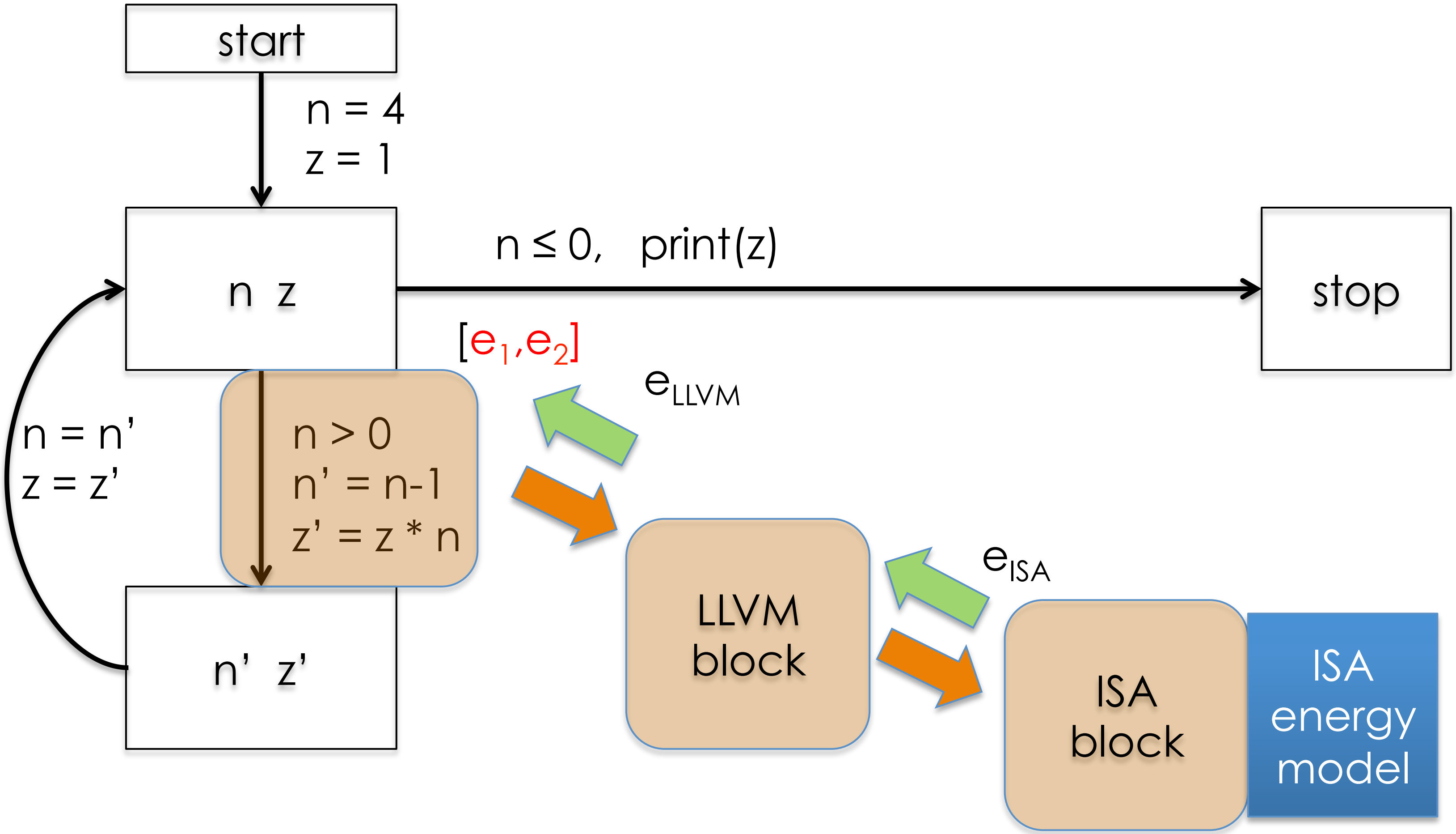
Software and Energy-aware Computing
Static analysis and optimization

John Gallagher
Roskilde University

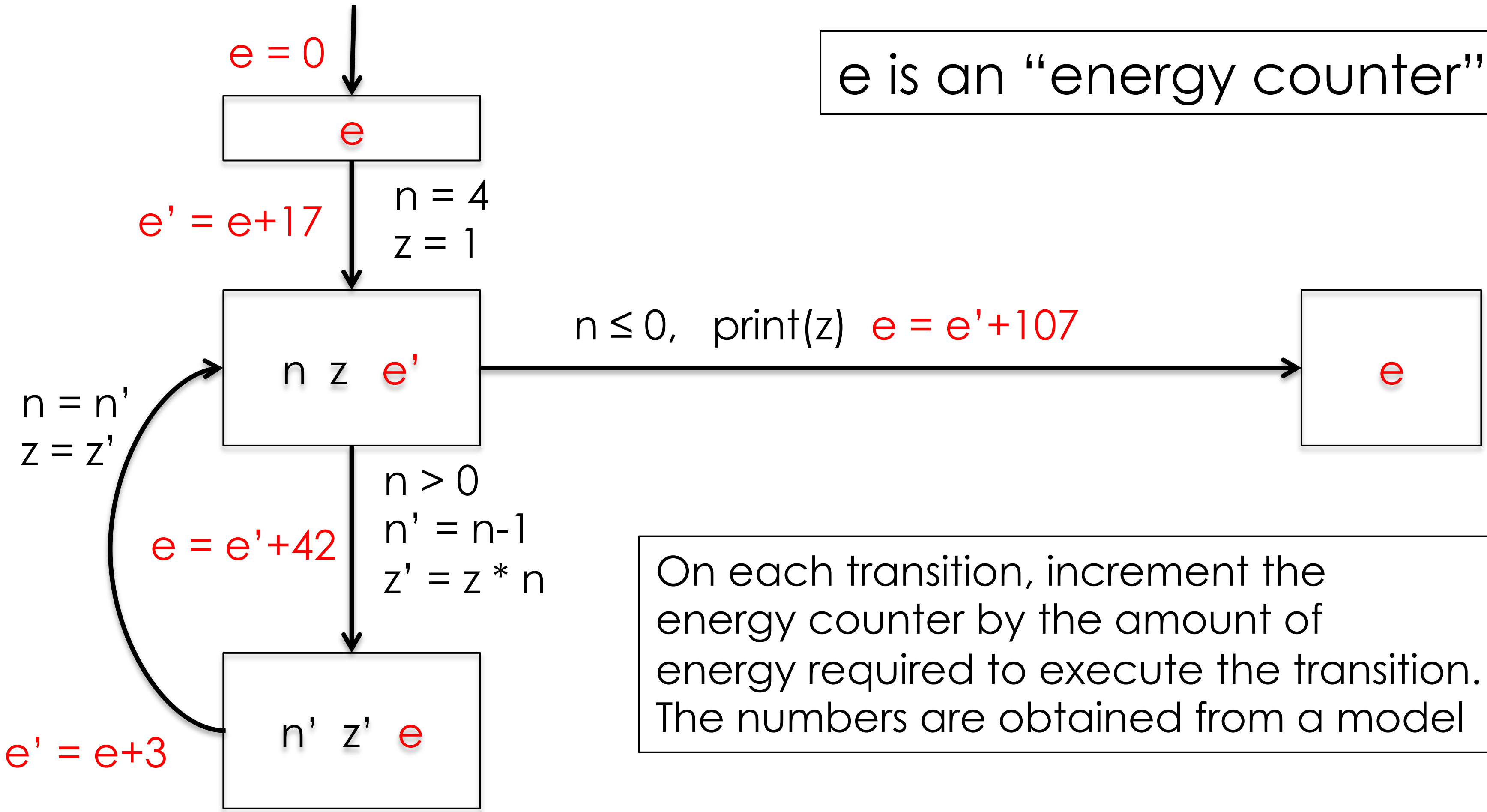
ICT-Energy: Energy consumption in future ICT devices

Summer School, Aalborg, Denmark, August 13-16, 2016

Energy models – block-based



Adding energy to the model



e is an "energy counter"

On each transition, increment the energy counter by the amount of energy required to execute the transition. The numbers are obtained from a model

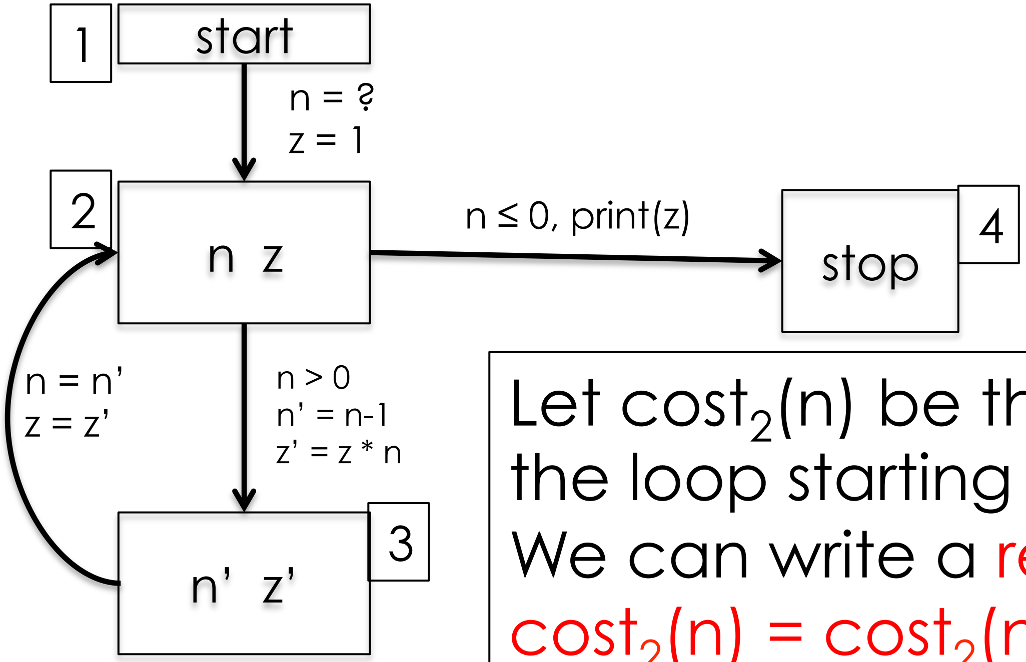
Estimating total energy

- The total energy consumed by the program is given by the energy counter in the reachable "stop" state.
- For this example, the analysis yields a value of 304 (initial value n=4)
- However if the input data is unknown, we would get a relationship between input value n and energy e.
- In the example, e = 17 + n*45 + 107

Beyond linear energy estimates

- With polyhedron or interval abstractions, we are limited to linear expressions.
- This is quite restrictive and approximate
- A better approach is given by deriving cost functions from the automaton, and solving them

Deriving cost functions



Let $cost_2(n)$ be the cost of the loop starting at 2.

We can write a recurrence relation

$cost_2(n) = cost_2(n-1) + 45$ (if $n > 0$)

$cost_2(n) = 0$ (if $n \leq 0$)

The cost of the whole computation for input n is $17 + cost_2(n) + 107$

Solving cost relations

- Tools like Mathematica are capable of solving many recurrence relations.

$cost_2(n) = cost_2(n-1) + 45$ (if $n > 0$)
$cost_2(n) = 0$ (if $n \leq 0$)

has a closed-form solution

$cost_2(n) = 45*n$

More complex cases

- By solving energy recurrence equations we can get non-linear energy functions
- E.g. a matrix multiplication program for matrices of size n

$42.47\ n^3 + 68.85\ n^2 + 49.9\ n + 24.22$ nJoules

Some available tools for cost analysis

- CiaoPP (IMDEA Software, Madrid)
  - a resource analysis tool based on solving cost relations (using Mathematica)
  - designed for Prolog programs, adapted to imperative languages
- COSTA (UCM, Madrid).
  - Can analyse resources such as time and energy for Java and Java bytecode (uses the PUBS solver)
- Termination analysis tools
  - several tools for proving termination of programs are being adapted for resource analysis

Trickier examples

- Loops counters can have inter-dependencies
- Complexity of example is O(2.m), not $O(m^2)$

```
void main(int m) {
    int i=m, n = 0; //stack = emptyStack();
l1 : while (i > 0) {
        i--;
        if (?) //push
          n++; //stack.push(element);
       else //popMany
l2 :      while (n > 0 && ?)
             n--; //element = stack.pop();
     }
}
```

Analysis of communication and timing

- We consider a language with synchronous channel communication
- Usually, threads enter some periodic behaviour, synchronising among themselves
- The programmer needs a model of how much work and time a thread uses between communications

Potential power optimisations (1)

- Sometimes, threads should run as slowly as possible, while still meeting deadlines from other threads
  - thus analysis of timing and synchronisation is critical
- Reducing clock frequency of cores saves power

Potential power optimisations (2)

- Threads that communicate a lot should be close (take account of communication infrastructure).
- Bottlenecks can be removed by shifting tasks or introducing more threads
- Very inactive threads can be merged with other threads.

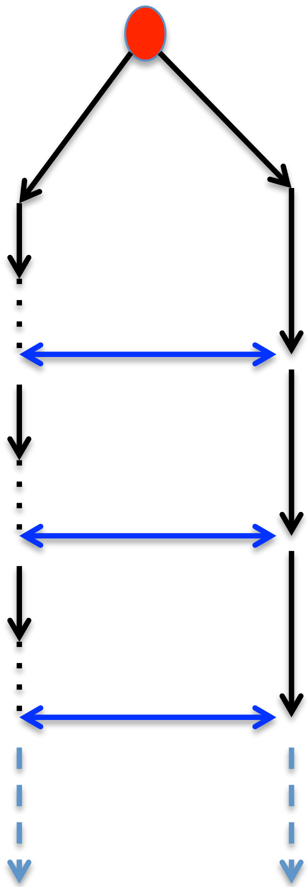
Parallel execution

Timing analysis is vital.

The left thread always waits for the other.

Possible energy optimisations:

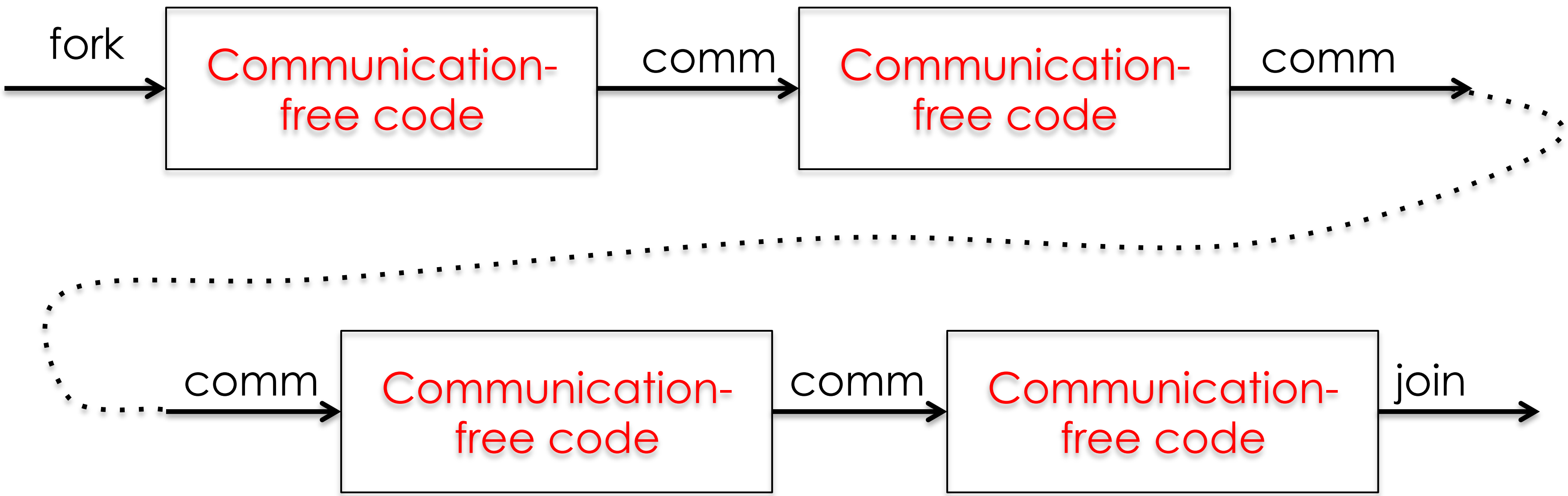
1. slow down the left thread
2. give it some more work to balance the load
3. put in power-saving mode while waiting



The threads run until they reach a synchronisation point.

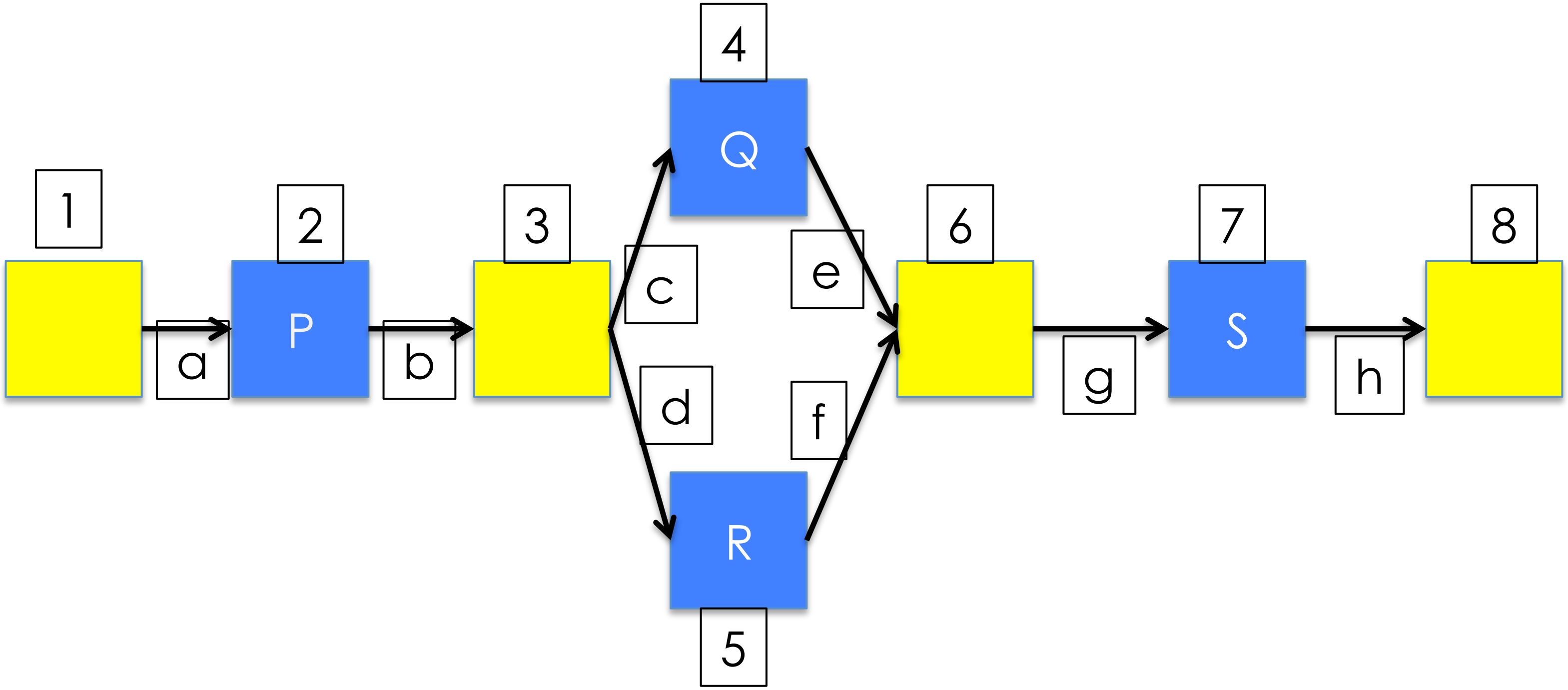
After synchronising, they continue to the next, etc.

Behaviour of a single thread



Each thread is parsed into blocks of communication-free code, separated by synchronous communications.
Assume that the communication channels are staticallly known.

Example thread behaviour

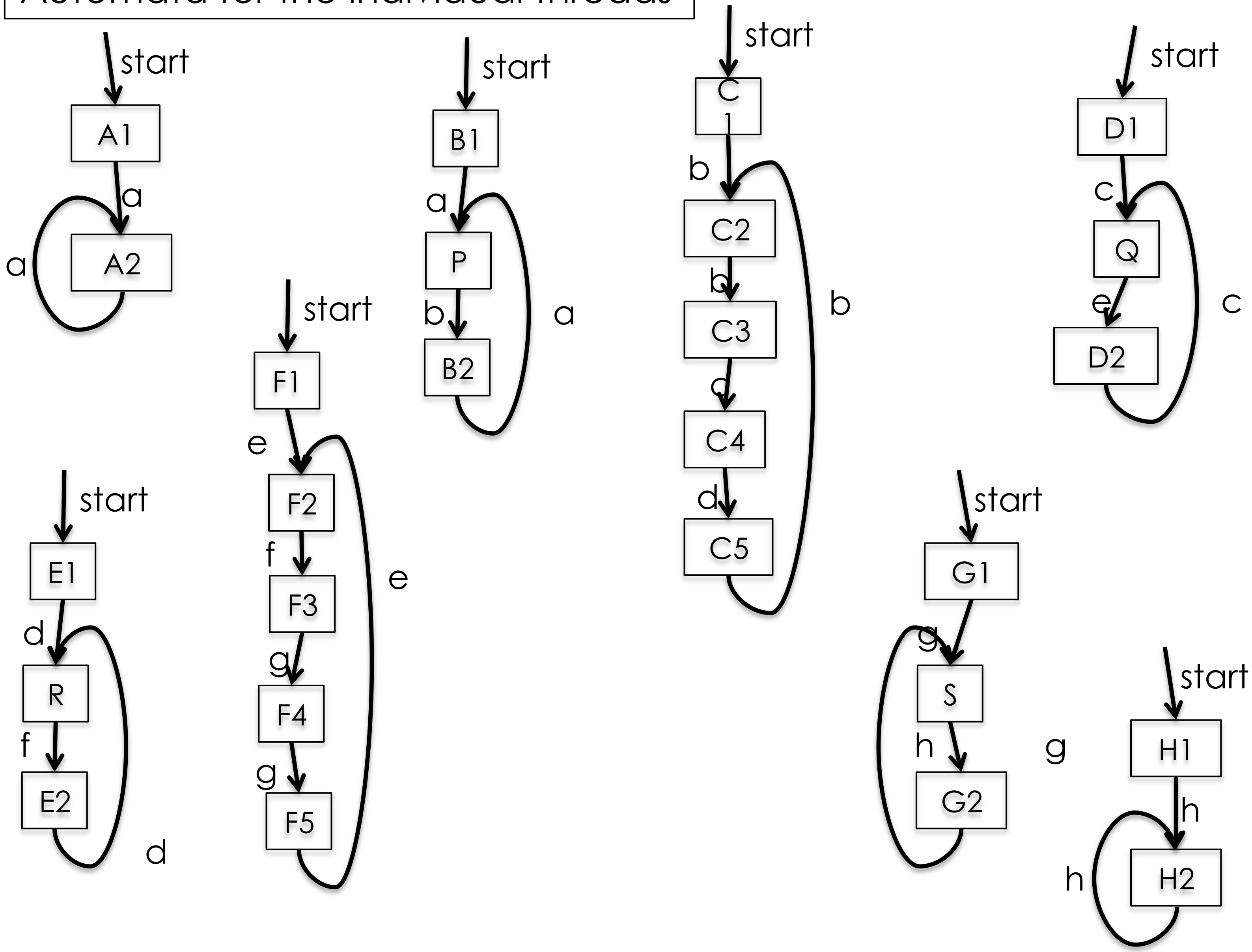


8 threads in a pipeline with a split in the middle.
P,Q,R and S are some functions on the values passed along.

Analysis of the sequential components

- We assume that we used the sequential techniques already mentioned
  - to get energy estimates for P,Q,R and S
  - to get execution time estimates for P,Q,R and S

Automata for the individual threads



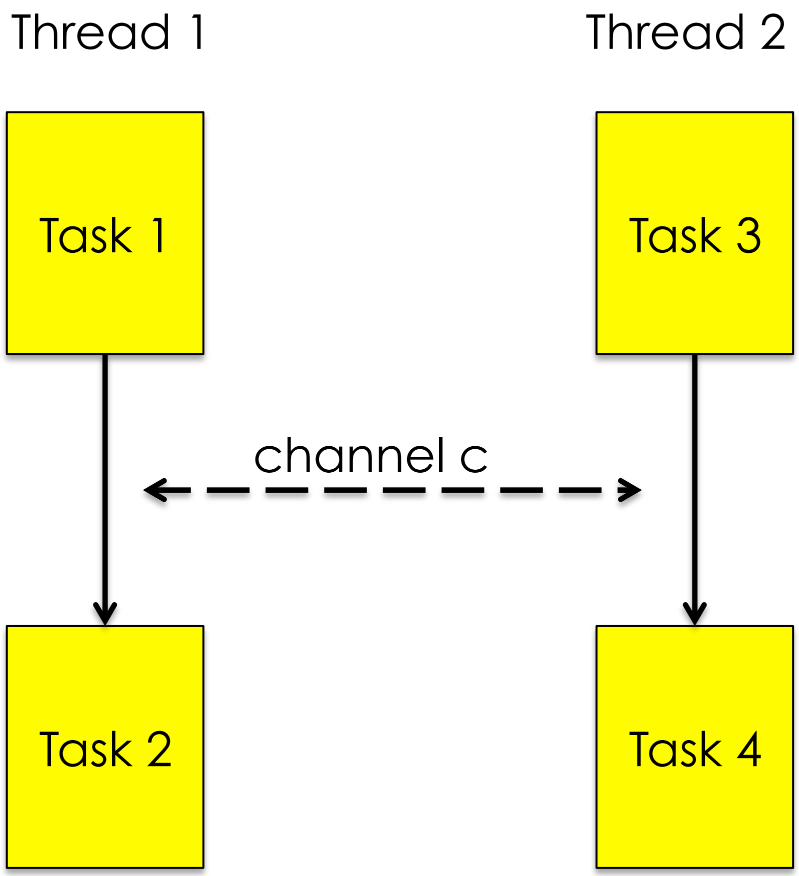
Energy and power estimates

- The energy of the whole cycle consists of
  - the total energy for the tasks in the cycle
  - an overhead for the number of active threads (obtained from the critical path)
  - an estimate of the energy used while idling
- The power (Watts) is E/T, where E is the energy and T is the time of the cycle

Task durations

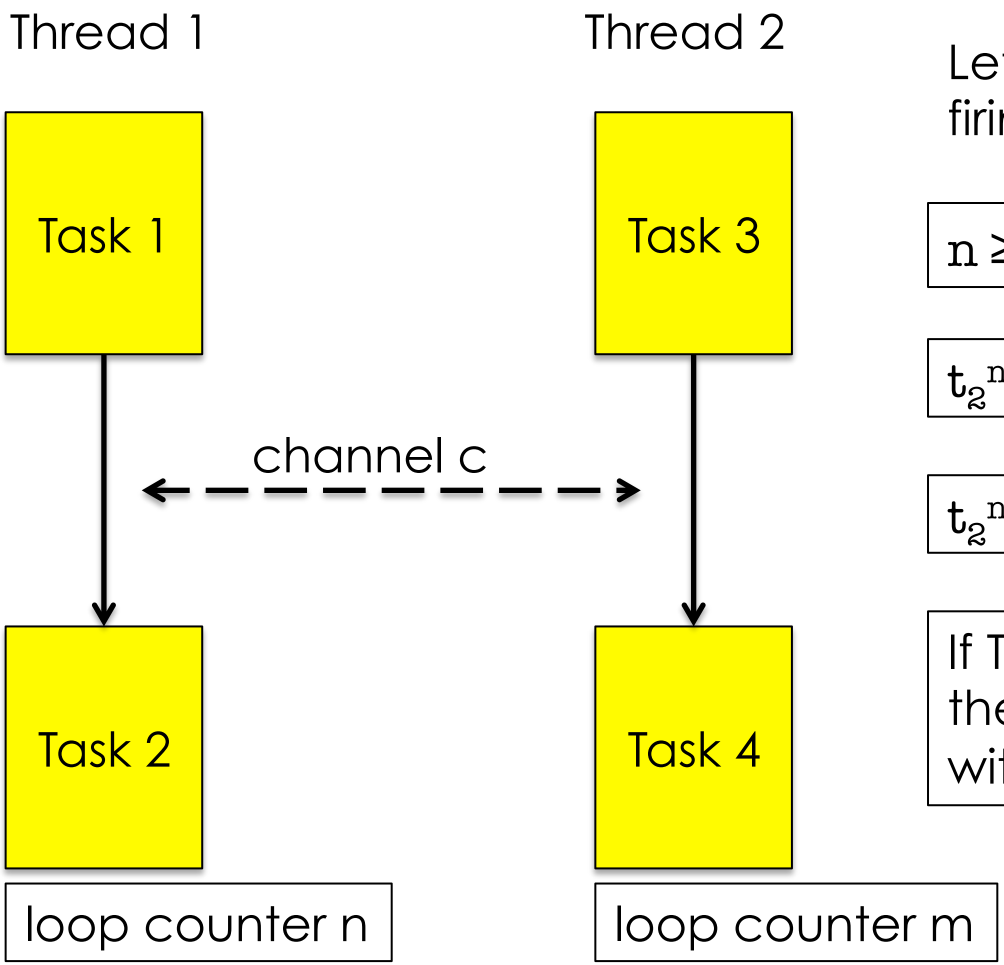
- Assume that each task has a duration
  - could be an interval [lower, upper]
  - or in general a constraint that could depend on data values
  - these can be obtained from a timing analyser and/or automatic complexity analysis
  - Let the duration of Task k be d_k

Synchronisation



Tasks 2 and 4 can start simultaneously as soon as both Tasks 1 and 3 have completed and the channel communication has been made.

Synchronisation constraints (1)



Let t_k^m be the time of the m[th] firing of task k.

$n \geq 0, m \geq 0$

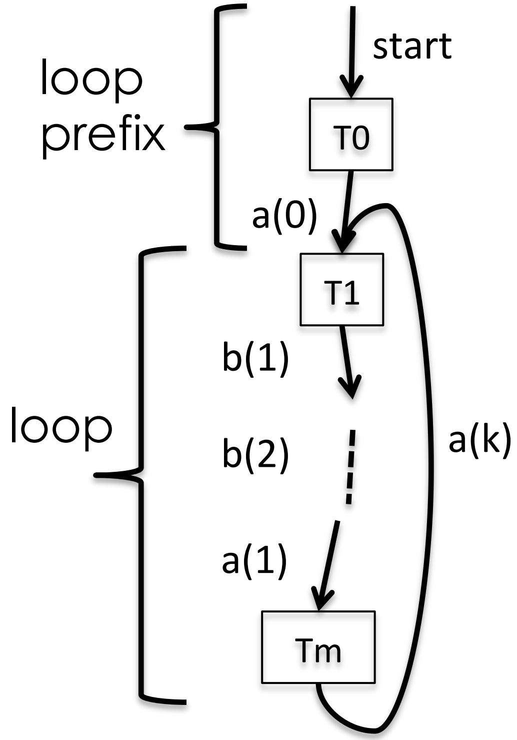
$t_2^n = \max(t_1^n + d_1, t_3^m + d_3)$

$t_2^n = t_4^m$

If Task 2 (or 4) is a loop header then replace t_2^n (or t_4^m) with t_2^{n+1} (or t_4^{m+1})

(Inspired by SDF graphs)

Counting communications



Annotate the channel communications so that they can be counted.

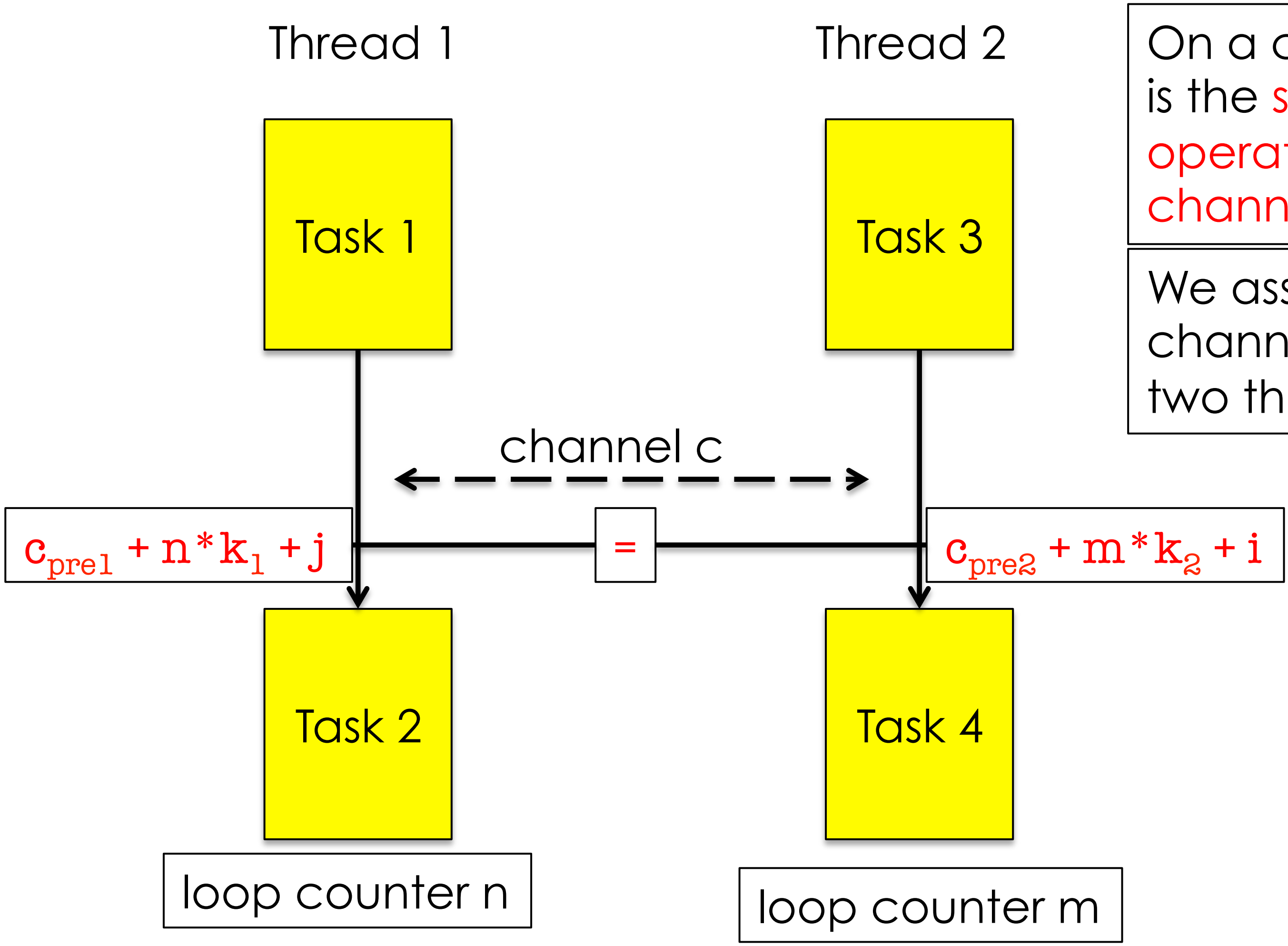
Let c_{pre} be the number of channel communications on c in the loop prefix.

Number every channel communication, for each channel c in the loop

c(1), ... c(k)

$\mathbf{c_{pre} + n * k + j}$ = the number of communications on c when **c(j)** in the loop is encountered, when **n** iterations of the loop are completed.

Synchronisation constraints (2)

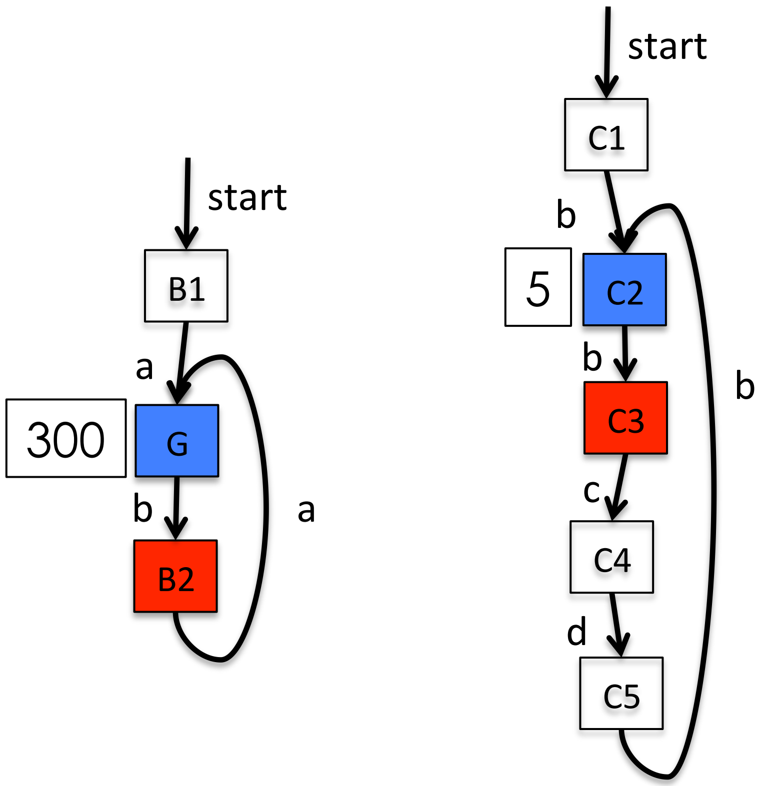


On a channel c, there is the same number of operations at each channel end.

We assume that each channel joins exactly two threads.

Example (logical encoding)

```
fires__B2(A,5+B) :-
        1+C*2+1=0+A*1+1,
        1+C*2+1>=0,
        C>=0,
        A>=0,
        fires__C2(C,B),
        fires__G(A,D),
        5+B>=300+D.
fires__B2(A,300+B) :-
        1+C*2+1=0+A*1+1,
        1+C*2+1>=0,
        C>=0,
        A>=0,
        fires__C2(C,D),
        fires__G(A,B),
       5+D<300+B.
```



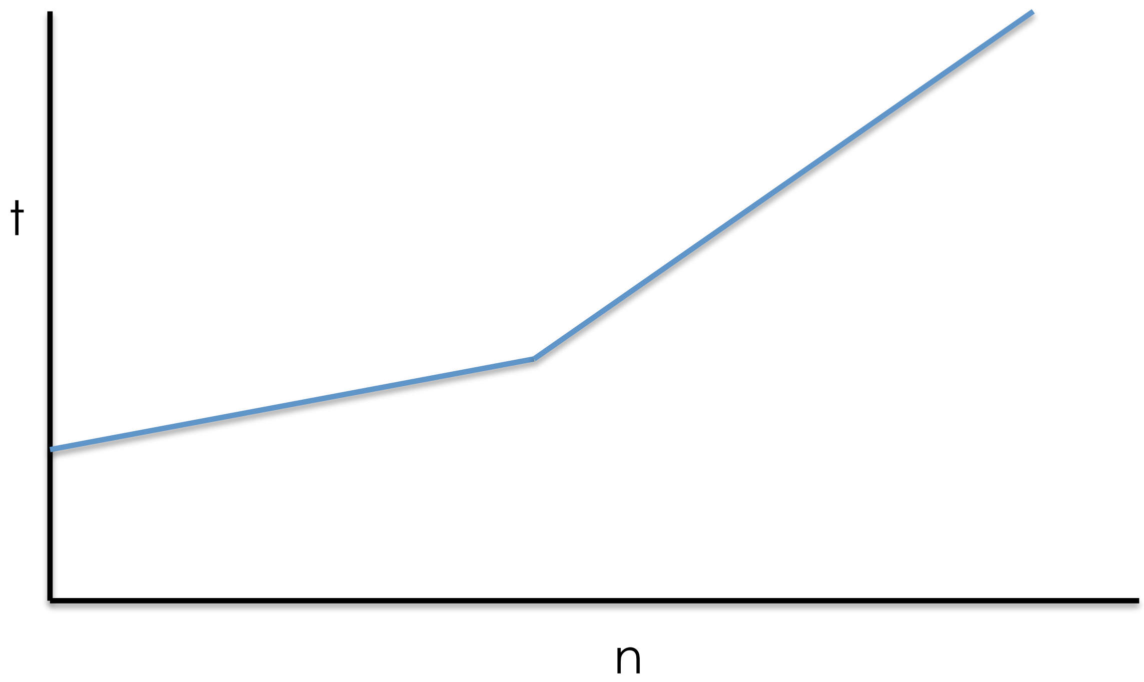
A and C are the loop counters og G and C2

Analysis of the constraints

- Generate the complete set of synchronisation constraints
- Solve them
  - more generally, obtain an approximate solution (abstract interpretation again!)
- For each task, derive a relationship between n and t, where t is the task's n^{th} firing time.

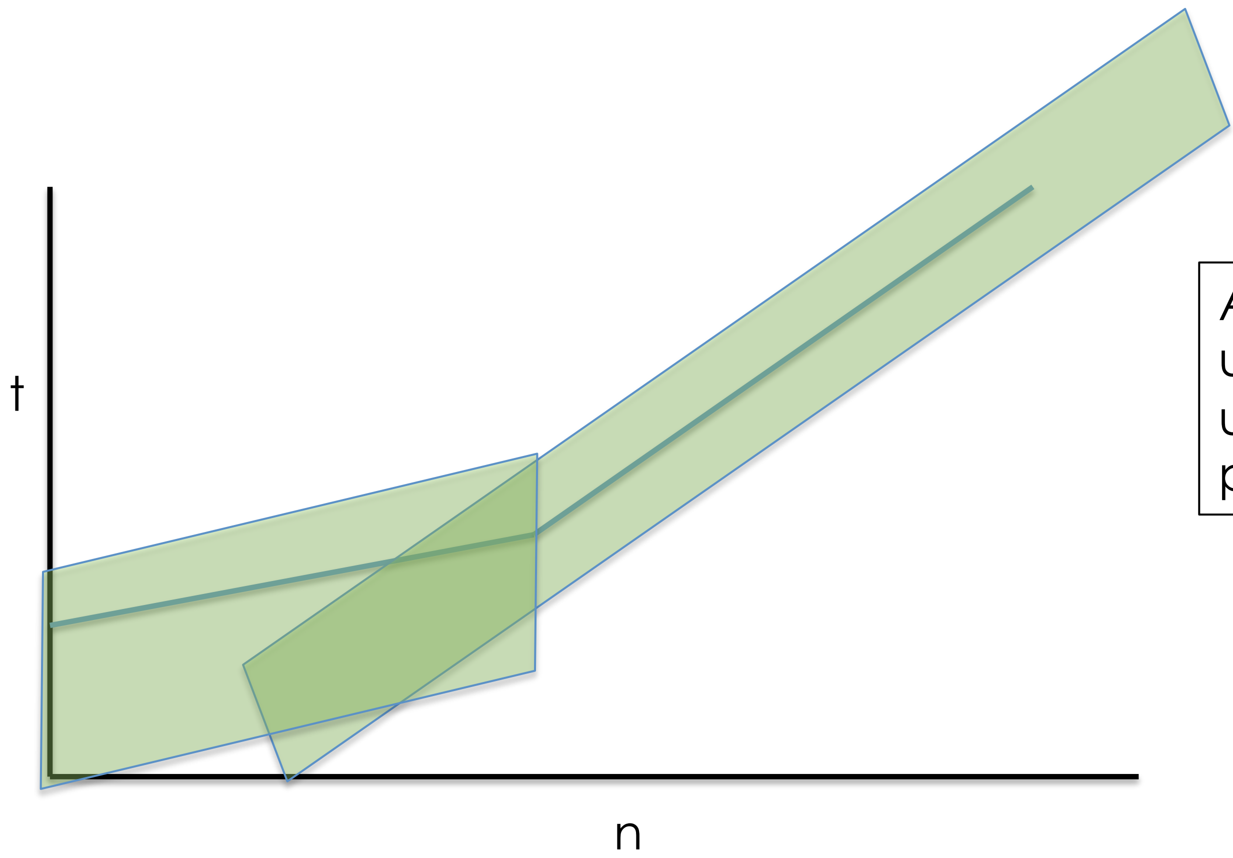
Transient and periodic behaviour

- Typically, threads take a few iterations to reach a steady state.



First few firings happen rapidly, then there is a slowdown as delays from other threads take effect.

Approximation of throughput



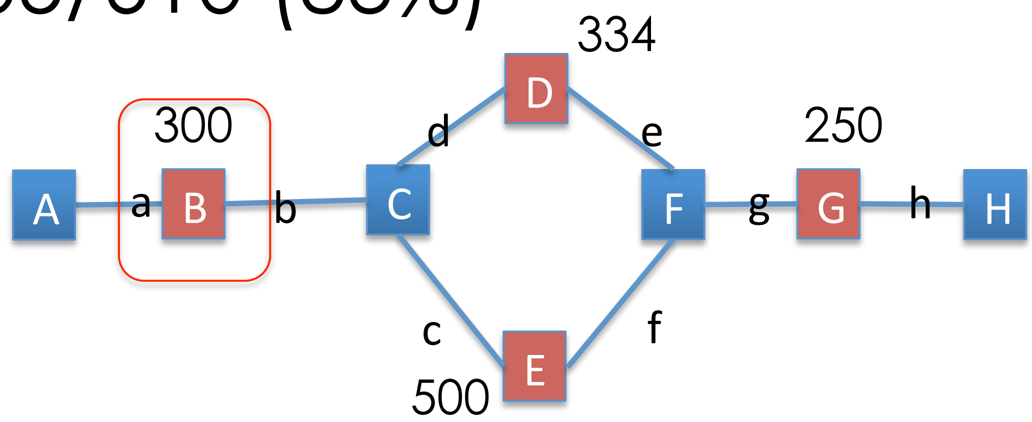
Approximation using a finite union of convex polyhedra

Analysis results

- For the 8-thread pipeline example
- Given task durations
  - G = 300
  - Q = 334
  - R = 500
  - S = 250
  - all other tasks = 5
- Derive period of threads = 610 or 305
- Some threads loop twice as fast as others

Thread activity

- Thread 1 = 5/305 (1.6%)
- Thread 2 = 305/305 (100%)
- Thread 3 = 20/610 (3.2%)
- Thread 4 = 339/610 (56%)
- Thread 5 = 505/610 (83%)
- etc.



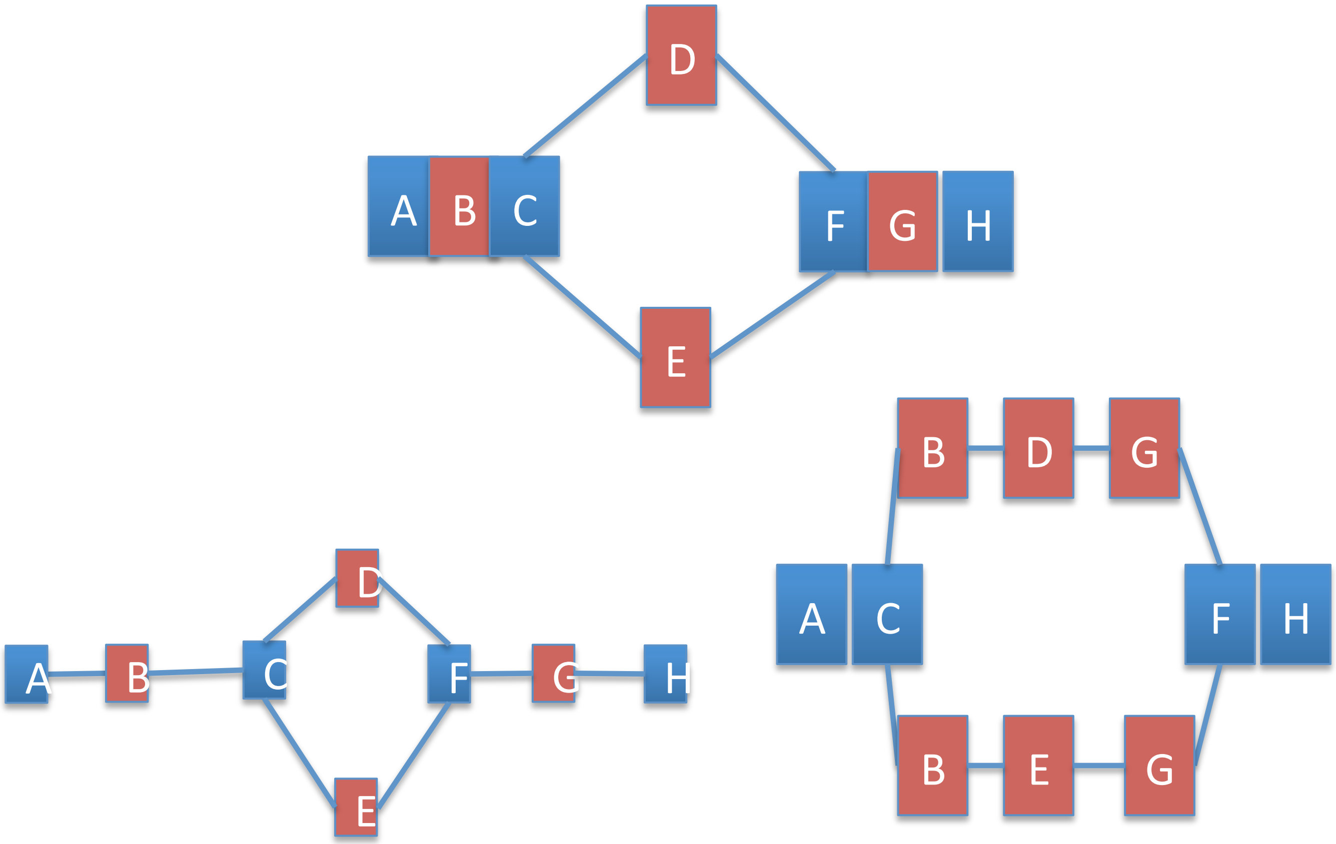
Other information

- Throughput and thread activity obtained directly from the solution to the constraints
- Other information that can be derived from earliest firing time includes
  - when one task definitely waits for another
  - which tasks can run simultaneously
  - which tasks on different threads do not run at the same time
  - frequency of each channel communication

Energy and power estimates

- The energy of the whole cycle consists of
  - the energy for each task in the cycle
  - an overhead for the number of active threads (obtained from the critical path)
  - an estimate of the energy used while idling
- The power (Watts) is E/T, where E is the energy and T is the time of the cycle

Possible transformations



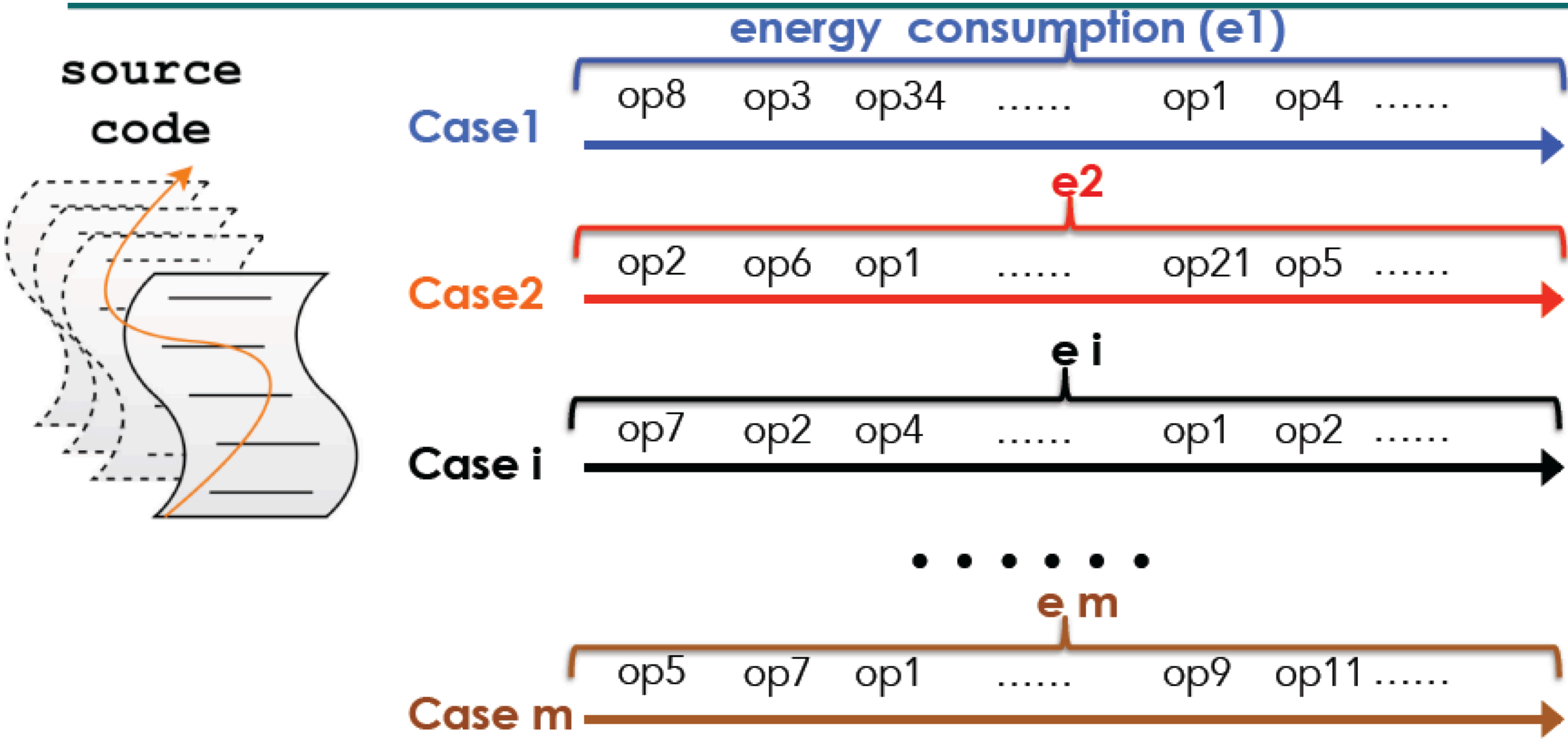
Energy optimisation for Android game code case study

- Work by Xueliang Li, Roskilde University (to appear in SCAM 2016)
- Energy of game code is highly dependent on user interaction
- We modelled the energy consumption the Cocos2d-Android game engine
- Energy consumption of operations in the source code was estimated using machine learning techniques
  - based on a large number of test cases for different interaction scenarios.

A Source Code energy model

- Android code is Java
- What is the code's energy cost? How can we measure it?
- The compiler produces Dalvik bytecode, which itself is interpreted by the Java virtual machine
- Is it realistic to attribute energy costs to source code?

Energy measurement of test cases



$n_j^{(i)}$ = # executions of op j in case i

Learning source-code operation costs

Numbers of executions of the energy operations in one test case

Energy costs of the operations

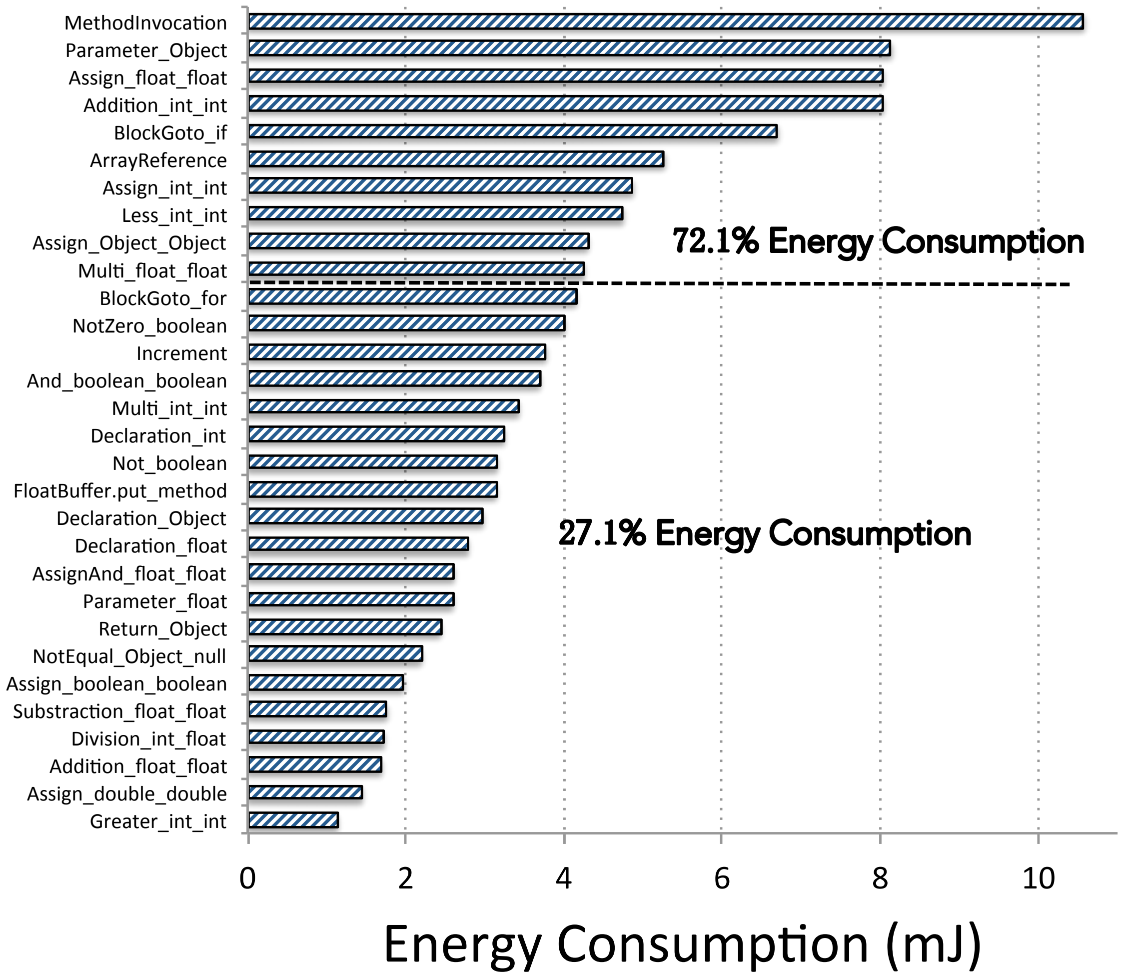
$$\begin{pmatrix} n_1^{(1)} & n_2^{(1)} & \dots & n_l^{(1)} \\ n_1^{(2)} & n_2^{(2)} & \dots & n_l^{(2)} \\ & \dots & \dots & \\ n_1^{(m-1)} & n_2^{(m-1)} & \dots & n_l^{(m-1)} \\ n_1^{(m)} & n_2^{(m)} & \dots & n_l^{(m)} \end{pmatrix} \times \begin{pmatrix} cost_1 \\ cost_2 \\ \dots \\ cost_l \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ \dots \\ e_{m-1} \\ e_m \end{pmatrix}$$

Acquired from log file

Aiming to obtain

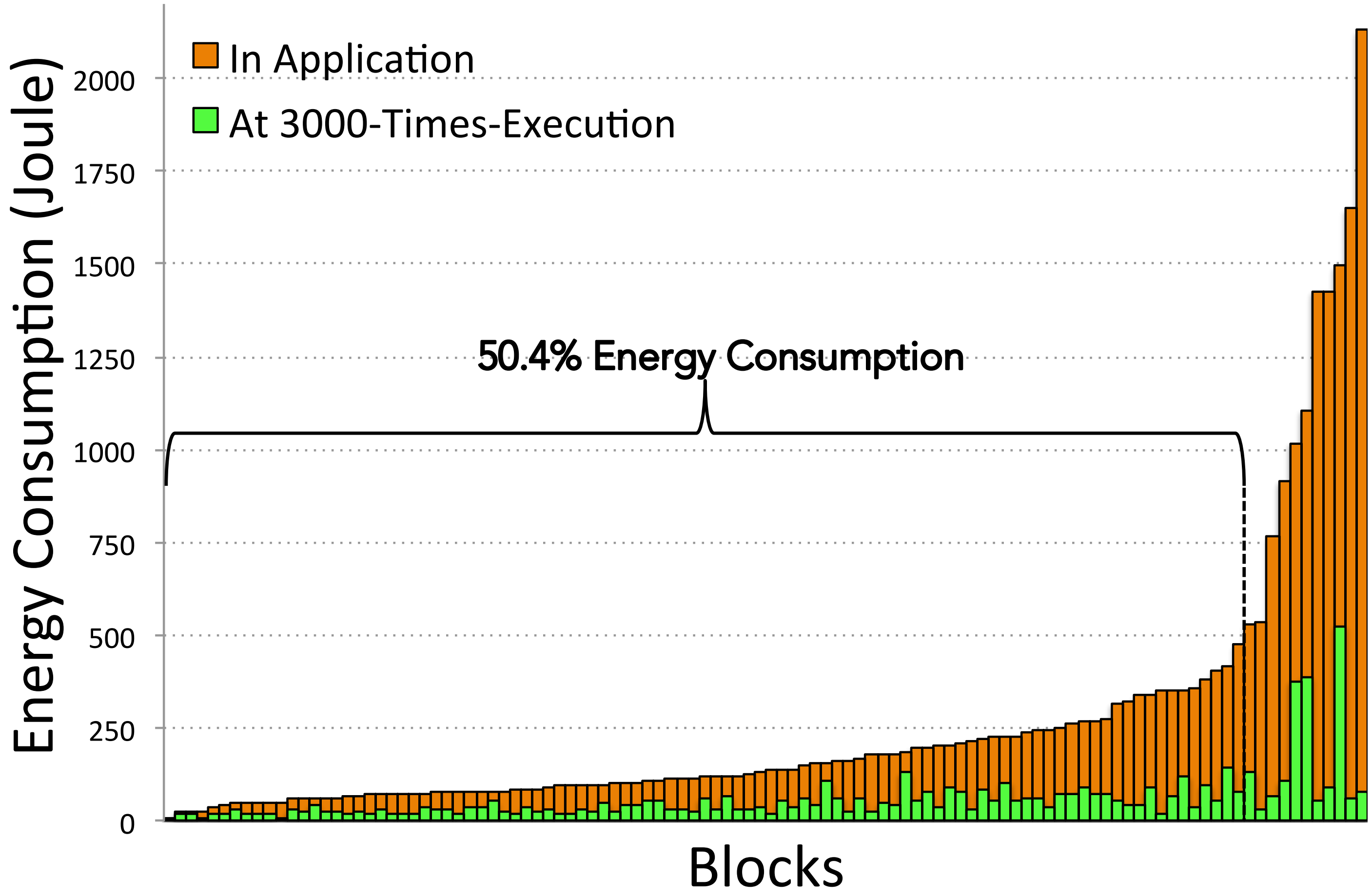
Measured

Identifying which ops use most energy



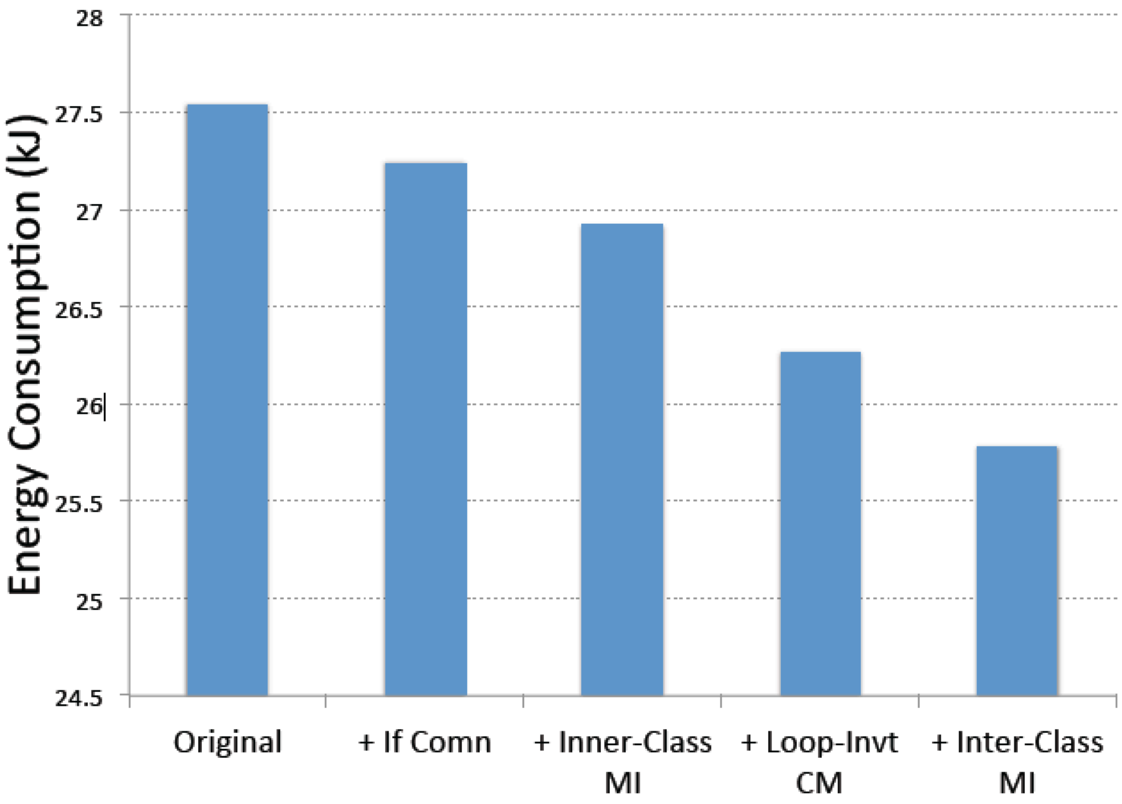
Top 10 ops account for 72.1% of energy usage

Which code blocks use most energy?



A few blocks dominate energy usage. These are targets for energy optimisation

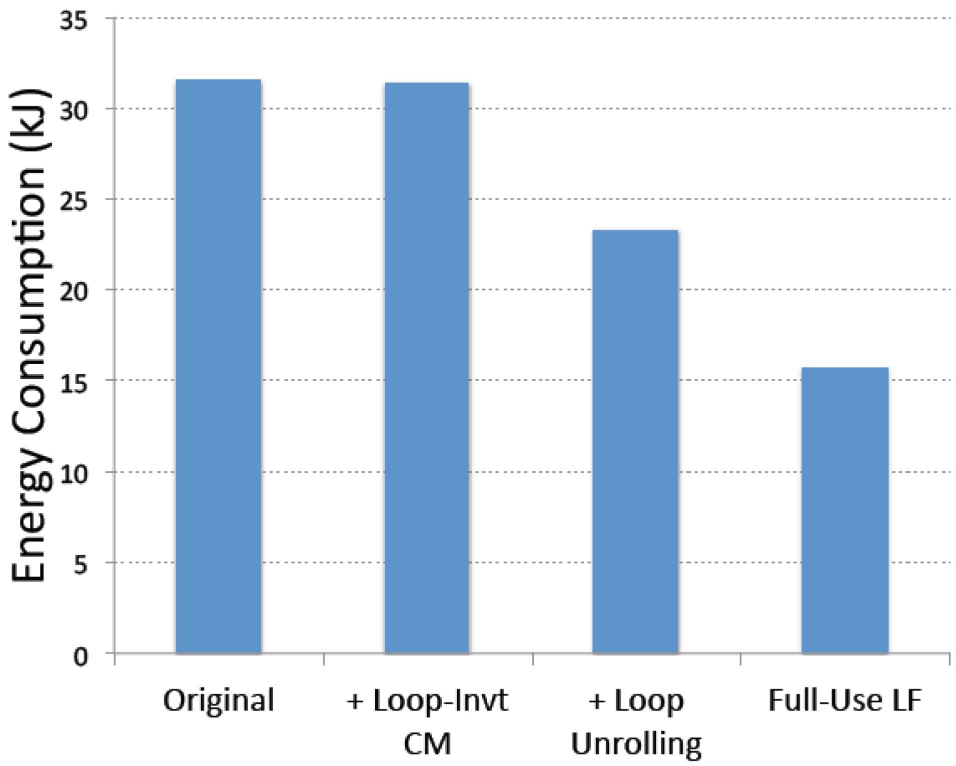
Optimisation. Example 1



Energy consumption of the code without and with the changes in Click & Move.

Overall saving: 6.4%

Optimisation. Example 2



Energy consumption of the code without and with the changes in Orbit.

Overall saving: 50.2%

Energy optimisations through energy transparency

- A thorough energy analysis of a suite of code enabled insight into where most energy was consumed
- This enabled source-code transformations to be focussed on the most effective areas.

Useful references

- B. Steigerwald and A. Agrawal. *Green software*. In San Murugesan and G. R. Gangadharan, editors, Harnessing Green IT : Principles and Practices, chapter 3. John Wiley & Sons, Hoboken, NJ, USA, 2012.
- U. Liqat, K. Georgiou, S. Kerrison, P. Lopez-Garcia, M. V. Hermenegildo, J. P. Gallagher, and K. Eder. *Inferring Parametric Energy Consumption Functions at Different Software Levels: ISA vs. LLVM IR*. In M. Van Eekelen and U. Dal Lago, editors, Foundational and Practical Aspects of Resource Analysis. Fourth International Workshop FOPARA 2015, Revised Selected Papers , Lecture Notes in Computer Science. Springer, 2016. http://arxiv.org/abs/1511.01413

Thank you