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

\[ n = 4 \]
\[ z = 1 \]

\[ n > 0 \]
\[ n' = n-1 \]
\[ z' = z \times n \]

\[ n \leq 0, \quad \text{print}(z) \]

 poop

LLVM block

ISA block

ISA energy model
Adding energy to the model

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

\[
\begin{align*}
    e &= 0 \\
    e' &= e + 17 \\
    n &= 4 \\
    z &= 1 \\
    n' &= n - 1 \\
    z' &= z \times n \\
    n &\leq 0, \quad \text{print}(z) \\
    e &= e' + 107 \\
    n &= n' \\
    z &= z' \\
    e &= e' + 42 \\
    e' &= e + 3
\end{align*}
\]
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, $e = 17 + n\times45 + 107$
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 $\text{cost}_2(n)$ be the cost of the loop starting at 2. We can write a recurrence relation:

$$\text{cost}_2(n) = \text{cost}_2(n-1) + 45 \text{ (if } n > 0\text{)}$$

$$\text{cost}_2(n) = 0 \text{ (if } n \leq 0\text{)}$$

The cost of the whole computation for input $n$ is $17 + \text{cost}_2(n) + 107$
Solving cost relations

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

\[ \text{cost}_2(n) = \text{cost}_2(n-1) + 45 \text{ (if } n > 0) \]
\[ \text{cost}_2(n) = 0 \text{ (if } n \leq 0) \]

has a closed-form solution
\[ \text{cost}_2(n) = 45\times n \]
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 $n$

  $42.47n^3 + 68.85n^2 + 49.9n + 24.22$ 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 $O(m^2)$

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

- **A1**
  - Start
  - **a** to **A2**
  - **a** loop

- **B1**
  - Start
  - **a** to **P**
  - **b** to **C2**
  - **b** loop

- **C**
  - Start
  - **b** to **C2**
  - **c** to **C5**
  - **e** loop

- **D**
  - Start
  - **c** to **Q**
  - **e** to **D2**
  - **c** loop

- **E1**
  - Start
  - **d** to **R**
  - **f** to **F3**
  - **g** to **F5**
  - **e** loop

- **F1**
  - Start
  - **e** to **F2**
  - **f** to **F3**
  - **g** to **F5**
  - **e** loop

- **F2**
  - **e** loop

- **F3**
  - **f** loop

- **F4**
  - **g** loop

- **F5**
  - **g** loop

- **G1**
  - Start
  - **g** to **S**
  - **h** to **G2**
  - **g** loop

- **G2**
  - **h** loop

- **H1**
  - Start
  - **g** to **S**
  - **h** to **H2**
  - **g** loop

- **H2**
  - **h** loop
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 \([\text{lower, upper}]\)
  – 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

Task 1

Task 2

channel c

Task 3

Task 4

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 $k$.

\[
\begin{align*}
n &\geq 0, \ m \geq 0 \\
\end{align*}
\]

\[
\begin{align*}
t_2^n &= \max(t_1^n + d_1, t_3^m + d_3) \\
\end{align*}
\]

\[
\begin{align*}
t_2^n &= t_4^m \\
\end{align*}
\]

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

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

Number every channel communication, for each channel \( c \) in the loop \( c(1), \ldots, c(k) \)

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

Annotate the channel communications so that they can be counted.
On a channel $c$, there is the same number of operations at each channel end.

We assume that each channel joins exactly two threads.

\[
c_{\text{pre1}} + n \cdot k_1 + j = c_{\text{pre2}} + m \cdot k_2 + i
\]

loop counter $n$

loop counter $m$
Example (logical encoding)

\[
\begin{align*}
\text{fires}_\text{B2}(A,5+B) &:- \\
1+C*2+1=0+A*1+1, & \\
1+C*2+1>=0, & \\
C>=0, & \\
A>=0, & \\
\text{fires}_\text{C2}(C,B), & \\
\text{fires}_\text{G}(A,D), & \\
5+B>=300+D. & \\
\text{fires}_\text{B2}(A,300+B) &:- \\
1+C*2+1=0+A*1+1, & \\
1+C*2+1>=0, & \\
C>=0, & \\
A>=0, & \\
\text{fires}_\text{C2}(C,D), & \\
\text{fires}_\text{G}(A,B), & \\
5+D<300+B. & \\
\end{align*}
\]

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.

![Graph showing transient and periodic behaviour](image)

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 of 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

\[ n_j^{(i)} = \# \text{ executions of op } j \text{ in case } i \]
Learning source-code operation costs

Numbers of executions of the energy operations in one test case

\[
\begin{pmatrix}
n_1^{(1)} & n_2^{(1)} & \ldots & n_l^{(1)} \\
n_1^{(2)} & n_2^{(2)} & \ldots & n_l^{(2)} \\
\vdots & \vdots & \ddots & \vdots \\
n_1^{(m-1)} & n_2^{(m-1)} & \ldots & n_l^{(m-1)} \\
n_1^{(m)} & n_2^{(m)} & \ldots & n_l^{(m)}
\end{pmatrix}
\]

Acquired from log file

Energy costs of the operations

\[
\begin{pmatrix}
cost_1 \\
cost_2 \\
\vdots \\
cost_l
\end{pmatrix}
\]

\[
\begin{pmatrix}
e_1 \\
e_2 \\
\vdots \\
e_{m-1} \\
e_m
\end{pmatrix}
\]

Aiming to obtain

\[
\begin{pmatrix}
cost_1 \\
cost_2 \\
\vdots \\
cost_l
\end{pmatrix}
\]

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


Thank you