Energy efficient HPC software

Learning from embedded systems software development

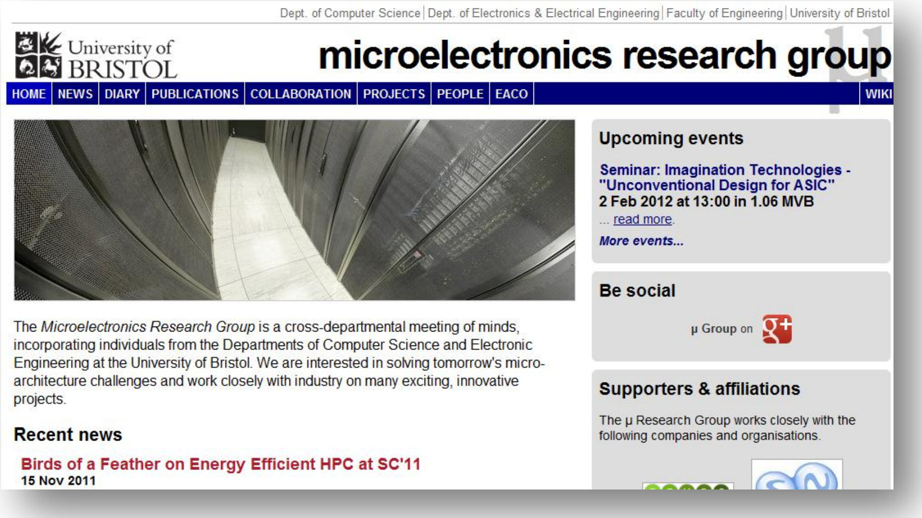
SIAM PP12

In this presentation

- University of Bristol Micro Research Group
- Energy challenges in HPC and embedded
- Solutions from the embedded world
- Latest research
- The future of energy efficient HPC software

About us

- Microelectronics Research Group, University of Bristol, UK
- Multi-disclipline
  - Design verification
  - Embedded systems
  - HPC
- **E**nergy **E**fficient **CO**mputing (EACO) is part of our core research.
- Dr. Kerstin Eder's research sponsored by the Royal Academy of Engineering.


Dept. of Computer Science | Dept. of Electronics & Electrical Engineering | Faculty of Engineering | University of Bristol
University of BRISTOL
microelectronics research group
HOME
NEWS
DIARY
PUBLICATIONS
COLLABORATION
PROJECTS
PEOPLE
EACO
WIKI

The *Microelectronics Research Group* is a cross-departmental meeting of minds, incorporating individuals from the Departments of Computer Science and Electronic Engineering at the University of Bristol. We are interested in solving tomorrow's micro-architecture challenges and work closely with industry on many exciting, innovative projects.

Recent news

Birds of a Feather on Energy Efficient HPC at SC'11
15 Nov 2011

Upcoming events

Seminar: Imagination Technologies - "Unconventional Design for ASIC"
2 Feb 2012 at 13:00 in 1.06 MVB
... read more.
More events...

Be social

μ Group on g+

Supporters & affiliations

The μ Research Group works closely with the following companies and organisations.

The Royal Academy of Engineering

About me

- MEng, Computer Systems Engineering, UoB
- PhD student in μ group
- Researching software energy modelling & optimisations
- Focus on embedded multi-core & multi-threaded systems.

In this presentation

- University of Bristol Micro Research Group
- **Energy challenges in HPC and embedded**
- Solutions from the embedded world
- Latest research
- The future of energy efficient HPC software

What are we trying to do?

HPC

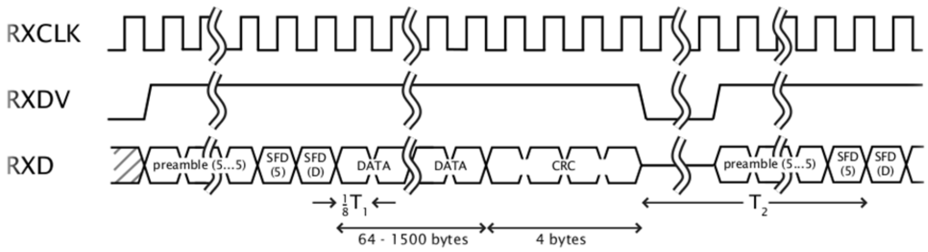
- Increase:
  - Density
    - More compute power per rack
  - Performance
  - Capability
- Manage:
  - Energy
    - Electricity bill
    - Cooling infrastructure

Embedded systems

- Increase:
  - Density
    - Smaller components
  - Performance
  - Available features on one chip
- Manage:
  - Energy
    - Battery life
    - Cooling requirements

The key difference?

- Embedded applications have **real-time constraints**
  - **Latency sensitive**
  - If a deadline is missed, something **breaks!**

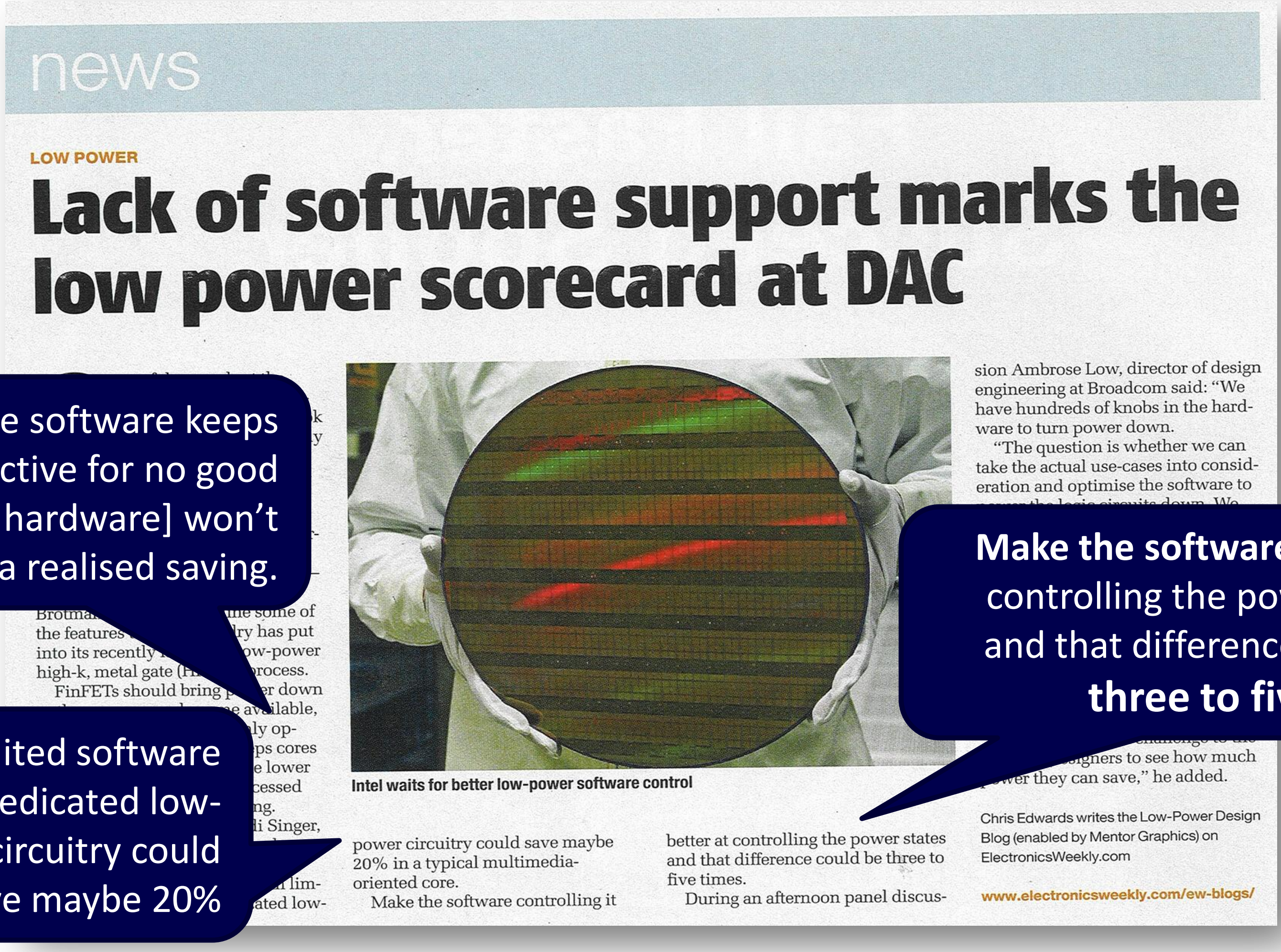


Ethernet MII RX timing

- HPC applications use **massive data-sets** and/or **many permutations**
  - Throughput oriented
  - Tolerate latency within acceptable limits
- The above describes general cases, not all cases!

Motivation

ElectronicsWeekly 15-21 June 2011 | No. 2472



news

LOW POWER

Lack of software support marks the low power scorecard at DAC

Brotma… …me some of the features … …ry has put into its recently … …ow-power high-k, metal gate (H… …process.

FinFETs should bring p… …er down … …e available, … …ly op- … …ps cores … …e lower … …cessed … …ng. … …i Singer, … …l lim- … …ated low-power circuitry could save maybe 20% in a typical multimedia-oriented core.

Make the software controlling it better at controlling the power states and that difference could be three to five times.

During an afternoon panel discussion Ambrose Low, director of design engineering at Broadcom said: “We have hundreds of knobs in the hardware to turn power down.

“The question is whether we can take the actual use-cases into consideration and optimise the software to … …signers to see how much power they can save,” he added.

Intel waits for better low-power software control

Chris Edwards writes the Low-Power Design Blog (enabled by Mentor Graphics) on ElectronicsWeekly.com

www.electronicsweekly.com/ew-blogs/

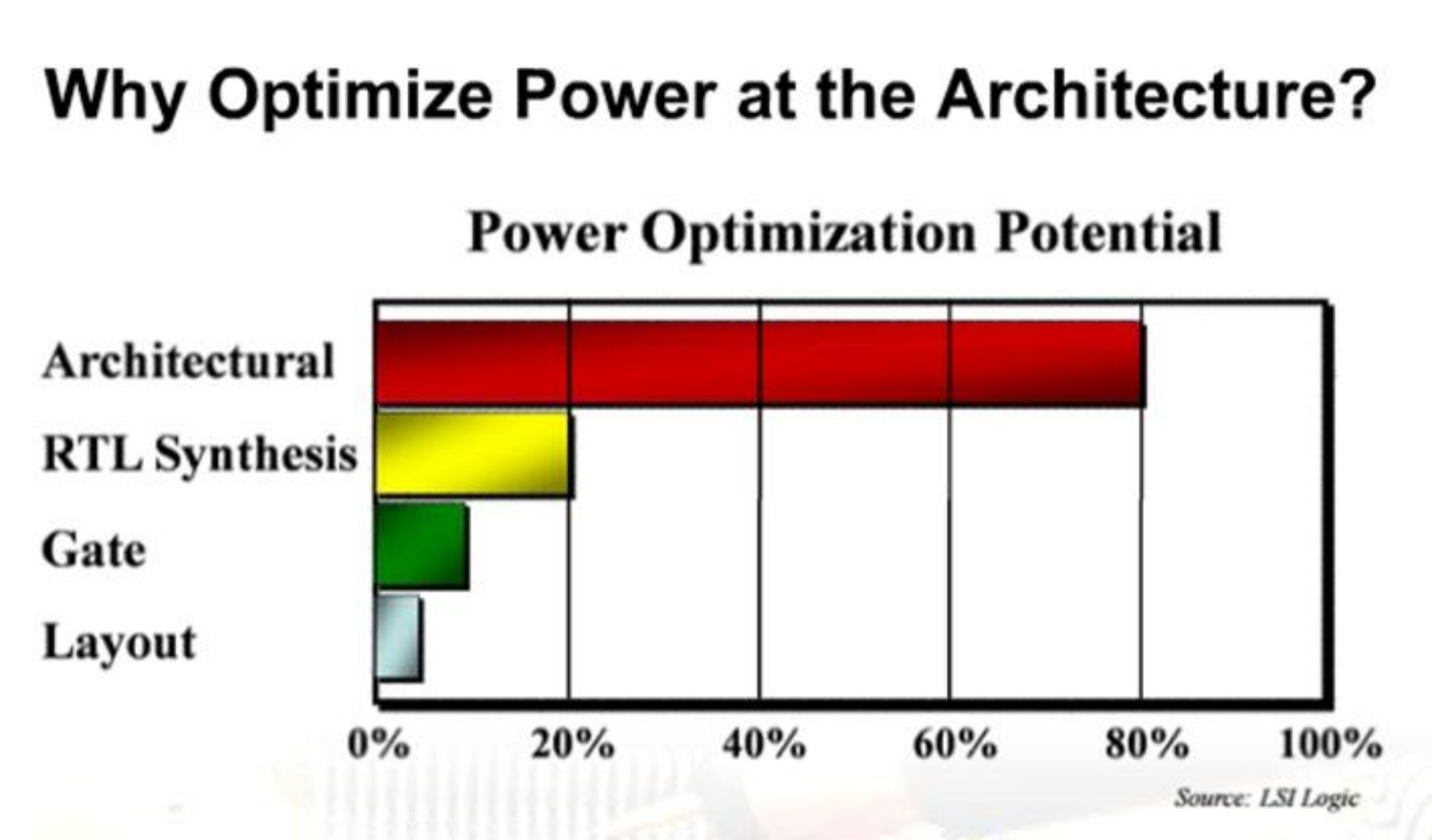
[…] if the software keeps cores active for no good reason [the hardware] won’t deliver a realised saving.

With limited software support, dedicated low-power circuitry could save maybe 20%

Make the software better at controlling the power states and that difference could be **three to five times.**

Greater Savings at Higher Levels

According to LSI Logic & Mentor Graphics…



In this presentation

- University of Bristol Micro Research Group
- Energy challenges in HPC and embedded
- **Solutions from the embedded world**
- Latest research
- The future of energy efficient HPC software

Aligning SW Design Decisions with Energy Efficiency Design Goal

Key steps:

- "Choose the **best algorithm** for the problem at hand and make sure it **fits well with** the computational **hardware**. Failure to do this can lead to costs far exceeding the benefit of more localized power optimizations.
- Minimize **memory size** and expensive **memory accesses** through algorithm transformations, efficient mapping of data into memory, and optimal use of memory bandwidth, registers and cache.
- Optimize the **performance** of the application , making **maximum use of available parallelism.**
- Take advantage of **hardware support for power management**.
- Finally, select instructions, sequence them, and order operations in a way that **minimizes switching** in the CPU and datapath."

Kaushik Roy and Mark C. Johnson. **1997**. Software design for low power. In *Low power design in deep submicron electronics*, Wolfgang Nebel and Jean Mermet (Eds.). Kluwer Nato Advanced Science Institutes Series, Vol. 337. Kluwer Academic Publishers, Norwell, MA, USA, pp 433-460.

Runtime -> energy?

- Can we use run-time to infer the energy consumed?
- A system's power profile is rarely flat over the run-time.
- To know energy without directly measuring it, we need to know *what* is consuming time, not just how much time has passed.
- Examples:
  - Memory accesses
  - ALU op
  - SIMD
  - I/O
  - The many forms of *"idle"*

Instruction-Level Power Analysis

Energy Cost *(E)* of a program *(P)*:

$$E_P = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k$$

Instruction base cost, B_i, of each instruction *i*, occurring N_i times

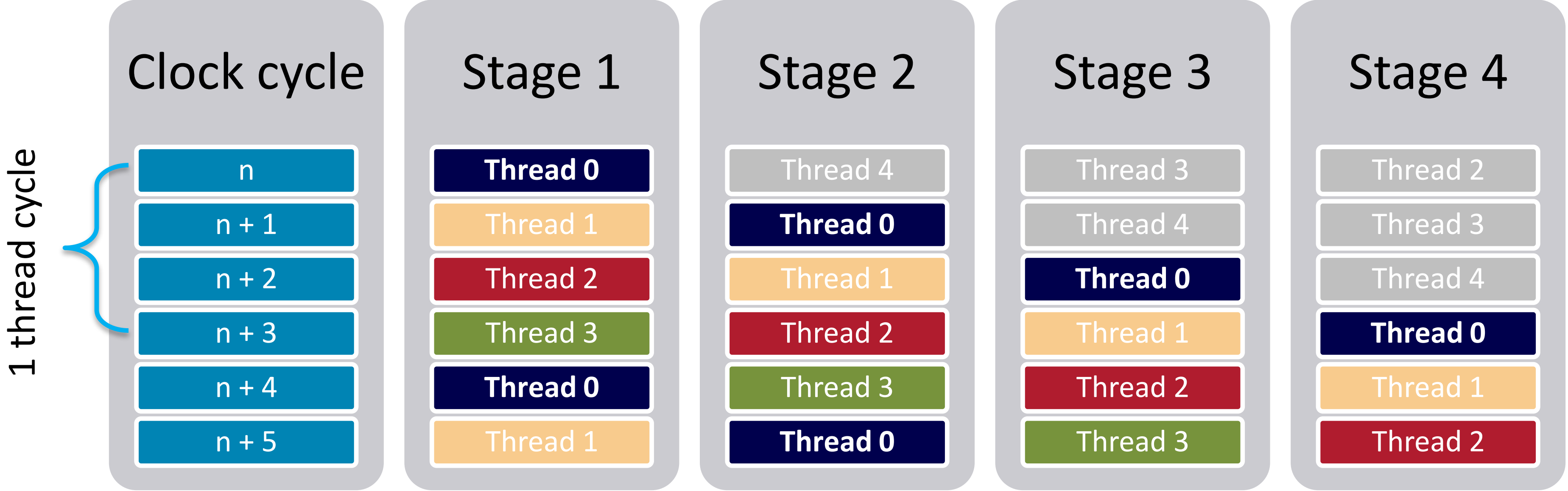
Circuit state overhead, $O_{i,j}$, for each instruction pair *i* and *j.*

Other instruction effects (stalls, cache misses, etc)

V. Tiwari, S. Malik and A. Wolfe, "Instruction Level Power Analysis and Optimization of Software", Journal of VLSI Signal Processing Systems, 13, pp 223-238, 1996.

ILPA: Doesn't always work

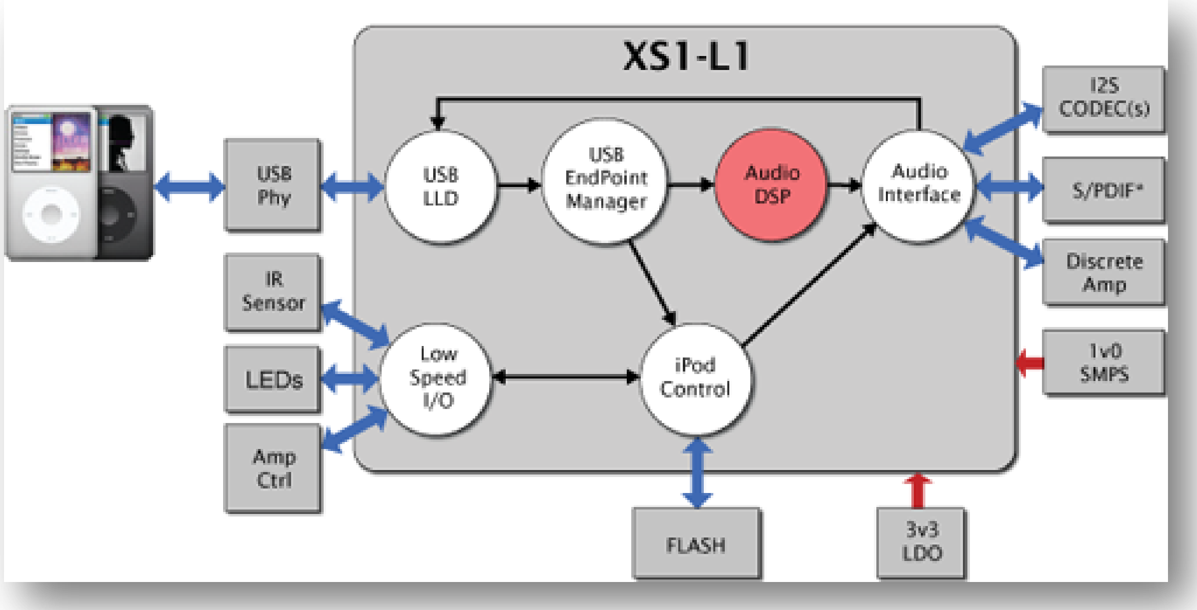
- XMOS XS1 embedded processor
- Up to eight threads per core
- Four stage pipeline
- Simple scheduling
- At 500MHz, 125MIPS per thread for <= 4 threads



Clock cycle	Stage 1	Stage 2	Stage 3	Stage 4
n	**Thread 0**	Thread 4	Thread 3	Thread 2
n + 1	Thread 1	**Thread 0**	Thread 4	Thread 3
n + 2	Thread 2	Thread 1	**Thread 0**	Thread 4
n + 3	Thread 3	Thread 2	Thread 1	**Thread 0**
n + 4	**Thread 0**	Thread 3	Thread 2	Thread 1
n + 5	Thread 1	**Thread 0**	Thread 3	Thread 2

Fixing the model

- Thread interaction example
  - I/O & **protocols** govern activity
  - More detail gives a **higher accuracy model.**
  - The more we can **predict**, the more **energy we can save**.



Doing *nothing* well

- One can spend a lot of time waiting for I/O
  - Sensors
  - Networks
  - Storage devices
- What do we do when waiting?
  - Early computation
  - Outstanding tasks
  - **Go to sleep**

Take a nap

- Frequency scaling *can* be fast (DFS)
  - Linear reduction in power
- Voltage **and** frequency scaling is slow (DVFS)
  - Quadratic reduction in power
- Turning off altogether is really slow
  - Well, turning *back on* is.
  - Best reduction in power

Formulae for calculating and controlling power/energy

$$P_{static} = V_{core} I_{leak}$$

$$P_{dynamic} = A_{\mu} C_{sw} V_{core}^{2} F$$

$$E = T(P_{static} + P_{dynamic})$$

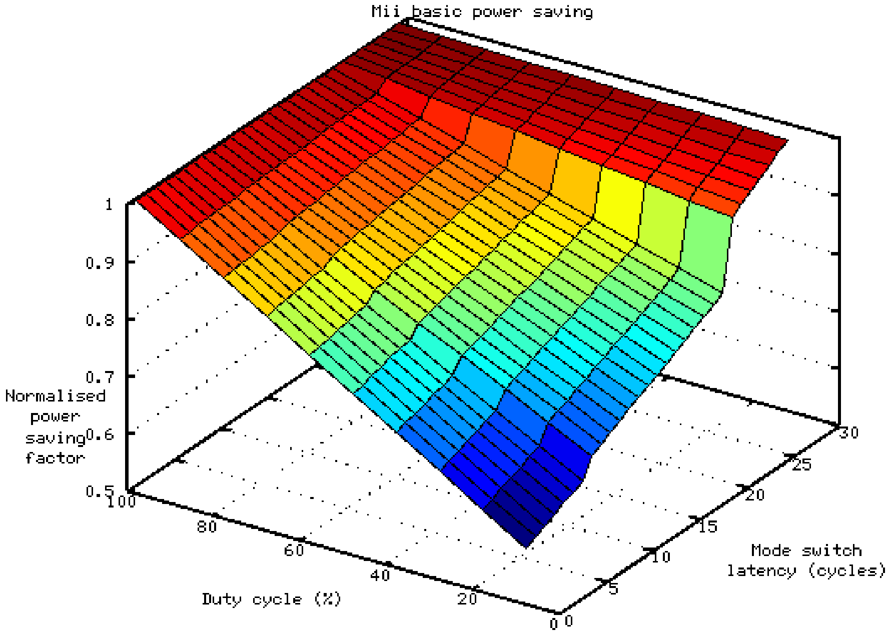
- We work within the tolerable latencies for the tasks and protocols we're using.
  - If we have 1ms to respond to the next signal, we can scale back the voltage while we wait for it.
  - If we have 1920ns, we have to re-think.

Software induced insomnia

- One of our biggest problems…
- HW has to guess when it can use low power states. **Why?**
- Because if we let SW do it, it takes too long. **WHY‽**

- HW & SW designers need to understand each other better!
- Knowing how the SW influences the HW's energy saving features is *key* to SW energy modelling and optimisation on modern processors.

The ideal strategy



Power saving vs. Ethernet duty cycle vs. Wake-up latency.

- Find a frequency & voltage at which we achieve 100% utilisation in time **and** meet all deadlines.
  - No slack
  - No extra latencies from DVFS
- This can be difficult.
  - So we're always searching for better task allocation strategies
  - And trying to make it quicker and easier to use energy saving features

In this presentation

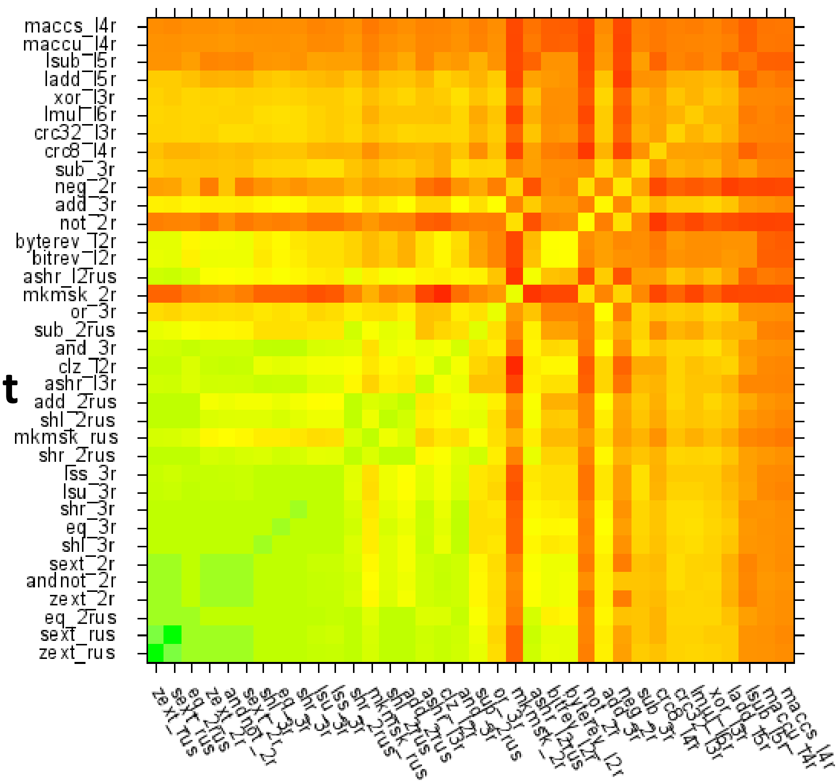
- University of Bristol Micro Research Group
- Energy challenges in HPC and embedded
- Solutions from the embedded world
- **Latest research**
- The future of energy efficient HPC software

Building a better model…

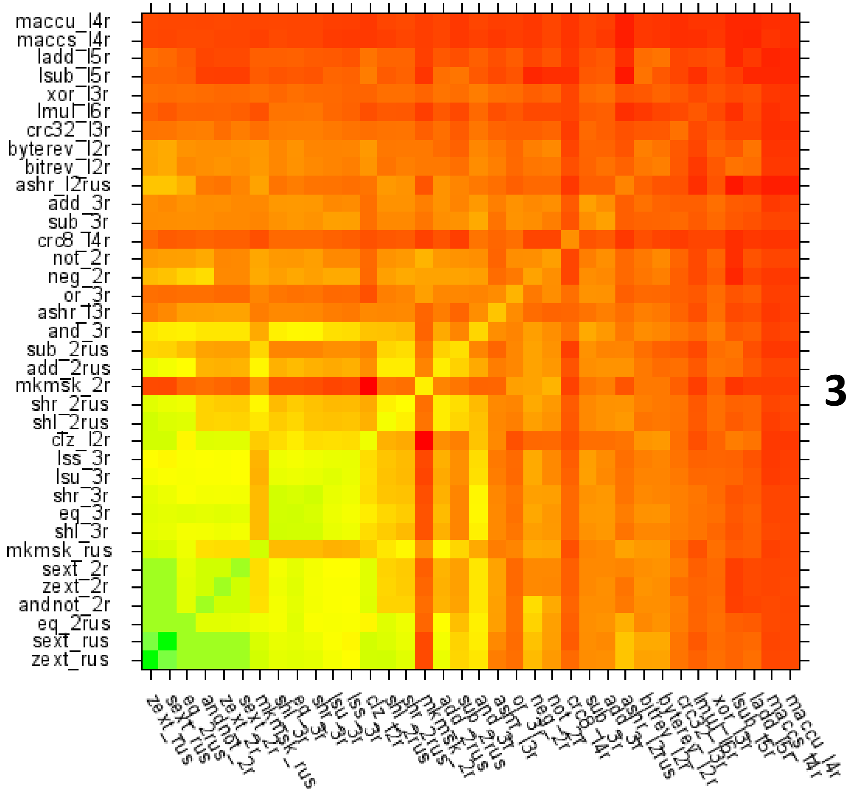
- Example: 32-bit XMOS hardware.
- The full width of the data path is **always active.**
  - No architectural specialisation for narrower data.
- Let’s compare the switching with 32-bit and 16-bit values.
- And let’s also look at the **operand count** for instructions.

Moving data matters

- The “hottest“ instructions tend to have more operands.
- 16-bit: **10-25%** improvement for some instruction pairings.



16-bit



32-bit

What does this tell us?

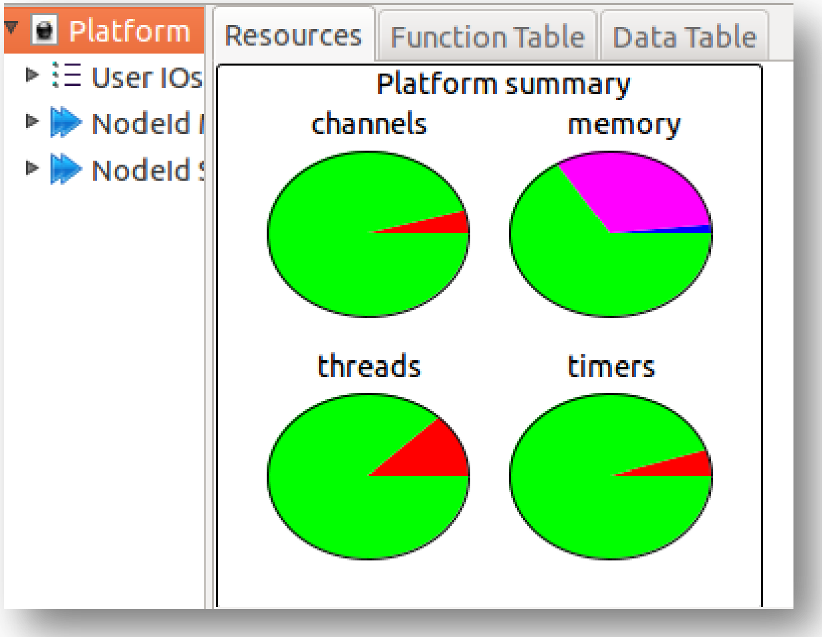
- A **cooperative** hardware/software optimisation.
- Possible base for a simple model.
- Needs expanding to deal with communication and other I/O.
  - Will give us the ability to model the energy trade-off between communication and computation.
- The challenge then becomes **communicating this data to the software engineer.**
  - Enable experimentation, *without* expensive re-runs.
  - Produce an algorithm that's a *good hardware fit.*

In this presentation

- University of Bristol Micro Research Group
- Energy challenges in HPC and embedded
- Solutions from the embedded world
- Latest research
- **The future of energy efficient HPC software**

Modelling the HPC software

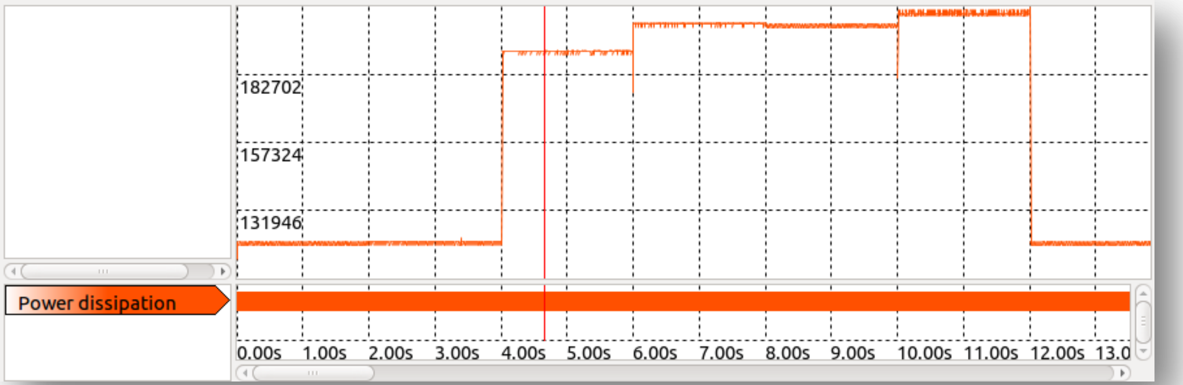
- Through static analysis of program code, determine our resource usage:
  - What components are we exercising?
  - FPU/SIMD/GPU/Disk/Comms
- Where is the energy going?



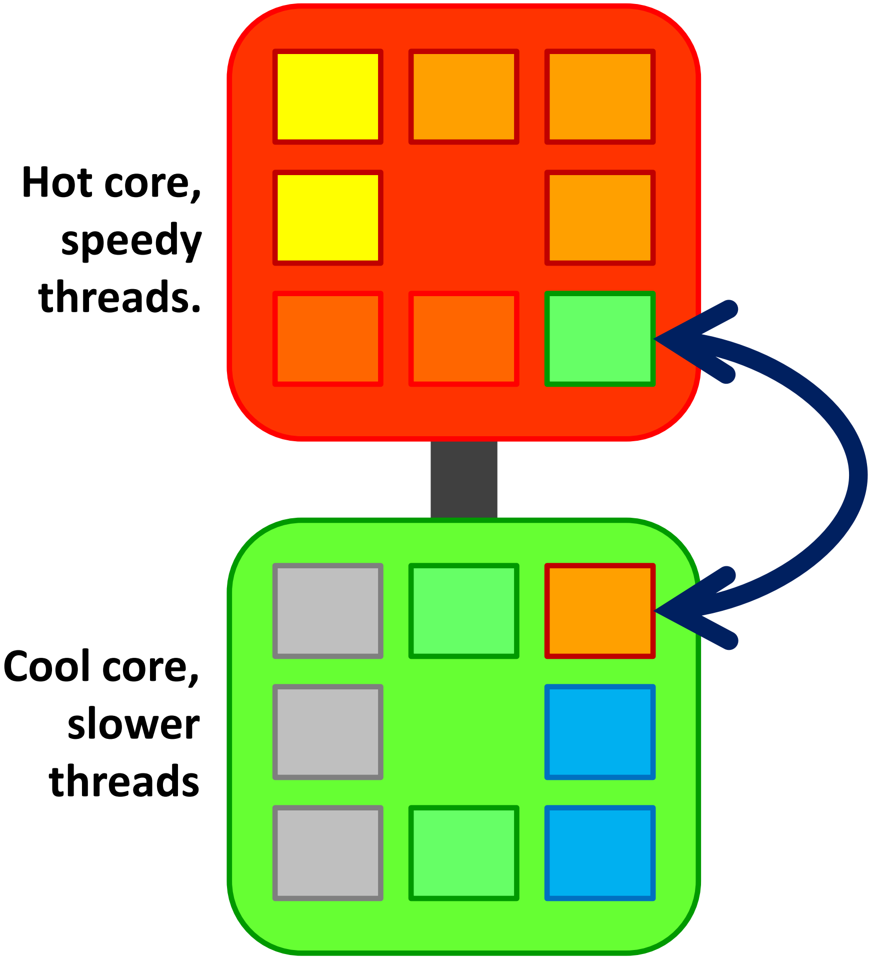
- What resources are we **not** using?
  - What can we slow down/put to sleep?

Profiling

- Run-time profiling is often used to guide performance optimisations.
- A workload that is predictable can help us with more than just performance.
- Profile resource usage and set our frequencies, voltages, link states and task allocations accordingly.



Utilising multi-core/multi-threaded



- Assign less demanding tasks to low frequency cores.
- Move tasks that interfere with other optimisation efforts onto a different core.
- Example: ARM's big.LITTLE SoC design strategy.
- We have to deal with the **timing & communication implications** of doing this.

Finding the limit

- For **existing hardware…**
- Analyse energy efficiency in commonly used applications & libraries.
- **A call to arms:**
  - Looking for examples that we can hand-optimize
  - Measure the gap between manual opt. & tools.
- We also want to know **how you value your tools**
  - What do they tell you about resource usage?

A new paradigm

- **Energy as a 1st class design goal…**

```
in 15ms do {...}
in 29000mJ do {...}
```

- Allocate compute resources in units of energy, not just time.
- Captures both execution duration and power efficiency.
- Forces HPC application developers to think about energy.
- Energy-aware tools become *necessary* for developing new, energy efficient HPC code.

Thank you!

Getting in touch:

steve.kerrison@bristol.ac.uk

http://www.cs.bris.ac.uk/Research/Micro/

http://gplus.to/uobmicro/