Power-Normalized Performance Optimization of Concurrent Many-Core Applications

Matthew Travers, Rishad Shafik, Fei Xia

ACSD2016, June 2016, Toruń

Newcastle University

PRiME
www.prime-project.org

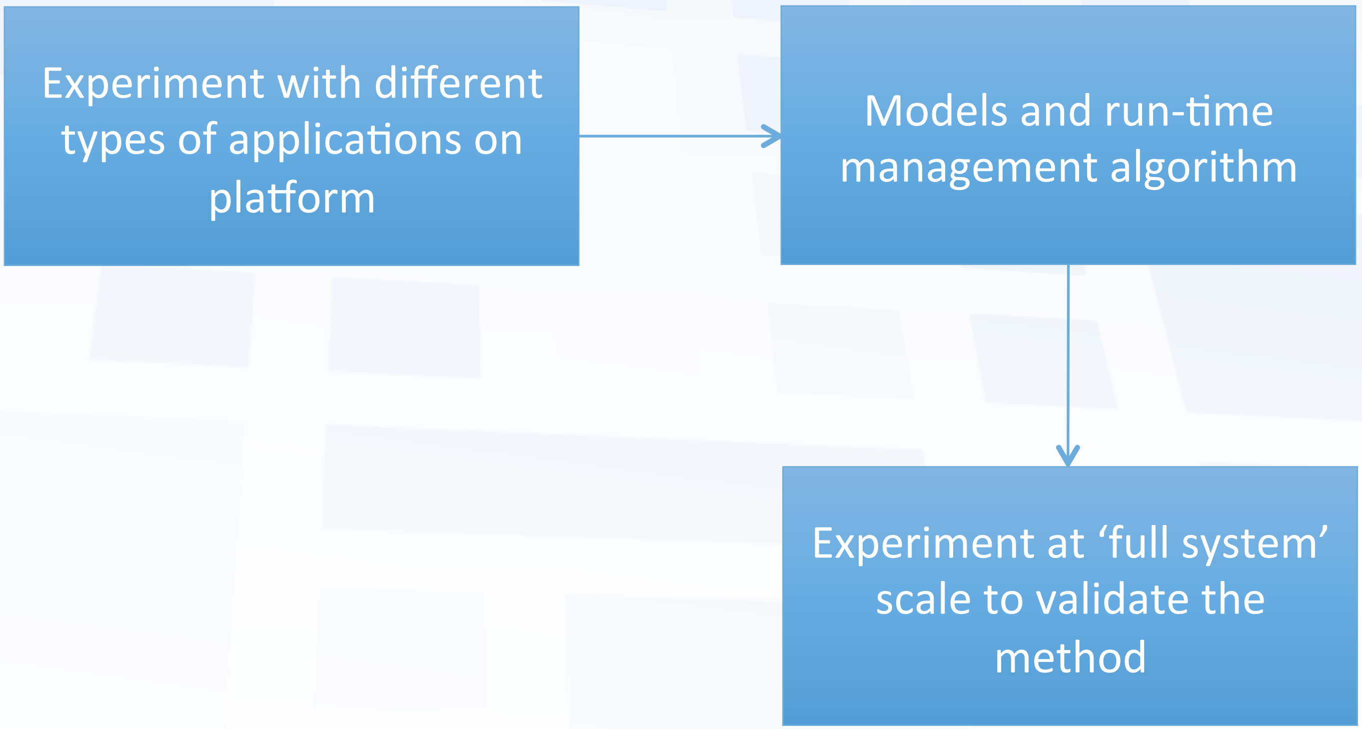
Motivation

- Complex platforms with multiple processing units (cores) facilitate the execution of concurrent programs
- Programs contain concurrency within or can be executed concurrently with other programs by the OS
- Great for performance, power, etc. but
- Run-time management is an important issue
- Existing OS management utilities are limited and lack sophistication

This work

- Experimentation targeting the running of multiple applications concurrently on multi-core systems
- Models targeting improved run-time management
- In order to improve efficiency
  – Measured in power-normalized performance (IPS/Watt) which is the same as the amount of computation per unit of energy

This work


Experiment with different types of applications on platform
Models and run-time management algorithm
Experiment at ‘full system’ scale to validate the method

Different types of applications

- Performance – power characteristics are related to multiple potentially independent variables
  - System platform architecture
  - Hardware control choices (voltage, frequency, clock gating, power gating, etc.)
  - Software control choices (types of tasks scheduled, types of instructions executed, etc.)
  - OS mapping choices (task-to-core scheduling decisions, etc.)
  - The problem is NP

Different types of applications

- In order to reduce the problem space, from experience it is conducive to divide applications (tasks) into different types
  - Computation (CPU) intensive
  - Memory intensive
  - A combination of both
- In this work we try to restrict ourselves to the three types listed above
  - Large reduction of decision space

Bespoke apps vs. established benchmarks

- Bespoke programs allow fine tuning in experimentation
  - You can set things in the program for various characteristics – no black box
  - Relation to ‘real world’ apps is relatively remote
  - Good for theoretical models at the beginning of study
- Established benchmarks connect to real world apps
  - ‘Standard’ group of apps that facilitate cross comparisons
  - Inevitable degree of black boxing – difficult to know what’s going on inside in certain cases in spite of open source
  - Distance between theory and experimental results

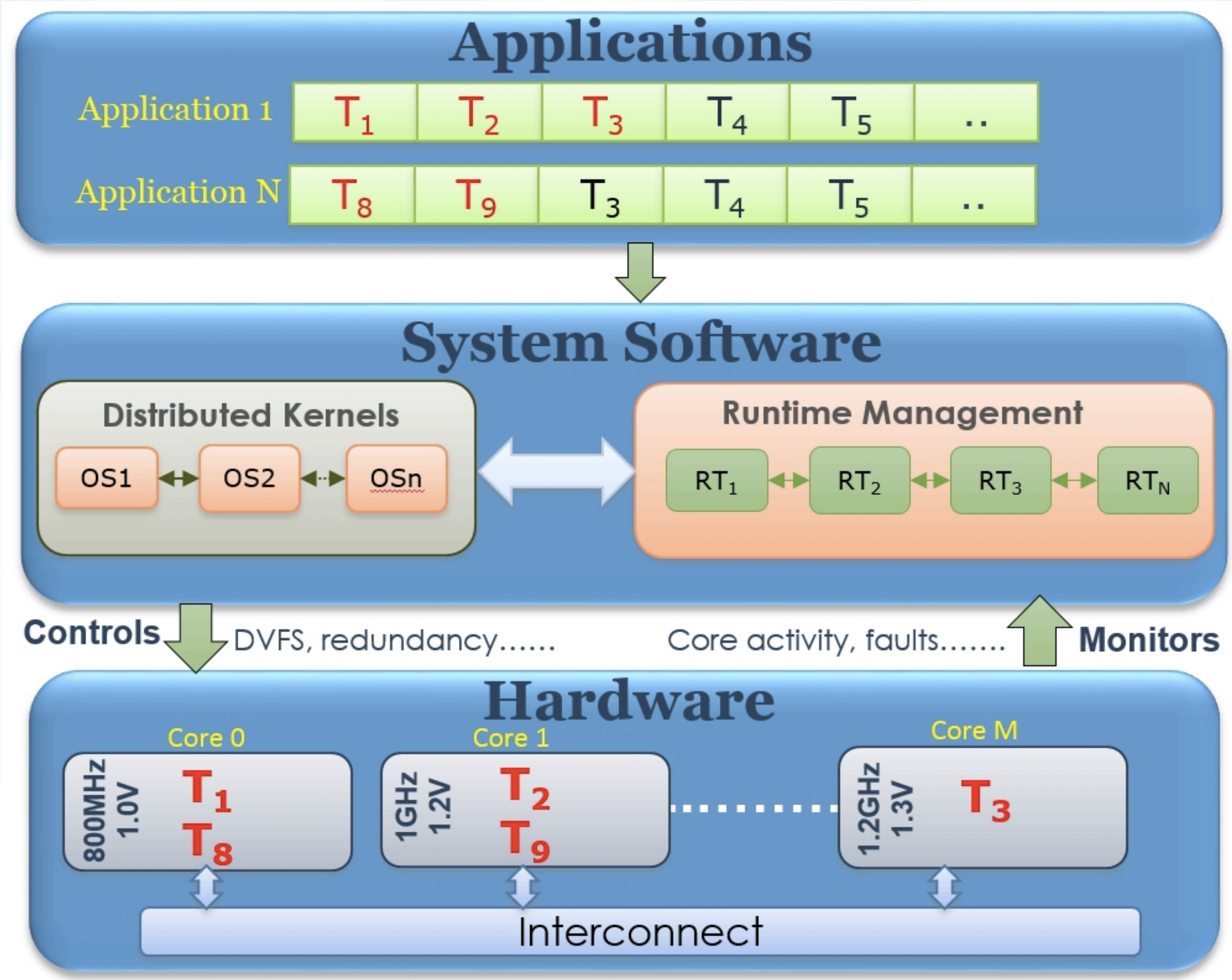
Modelling for RTM

- Runtime management is essentially a control system
  - Observability, controllability
- Observations are based on monitors
  - Run-time measurements for the parameters being controlled
  - Infer ‘efficiency’ from things that you can measure
  - IPS and Watt can both be measured through performance monitors (all modern CPUs have them)
  - Standard ones include ‘instructions retired’, ‘power consumption’, ‘cache misses’
  - External to CPUs (e.g. battery life, plugged-in vs. not, etc.)
  - Fault/error detection

Modelling for RTM

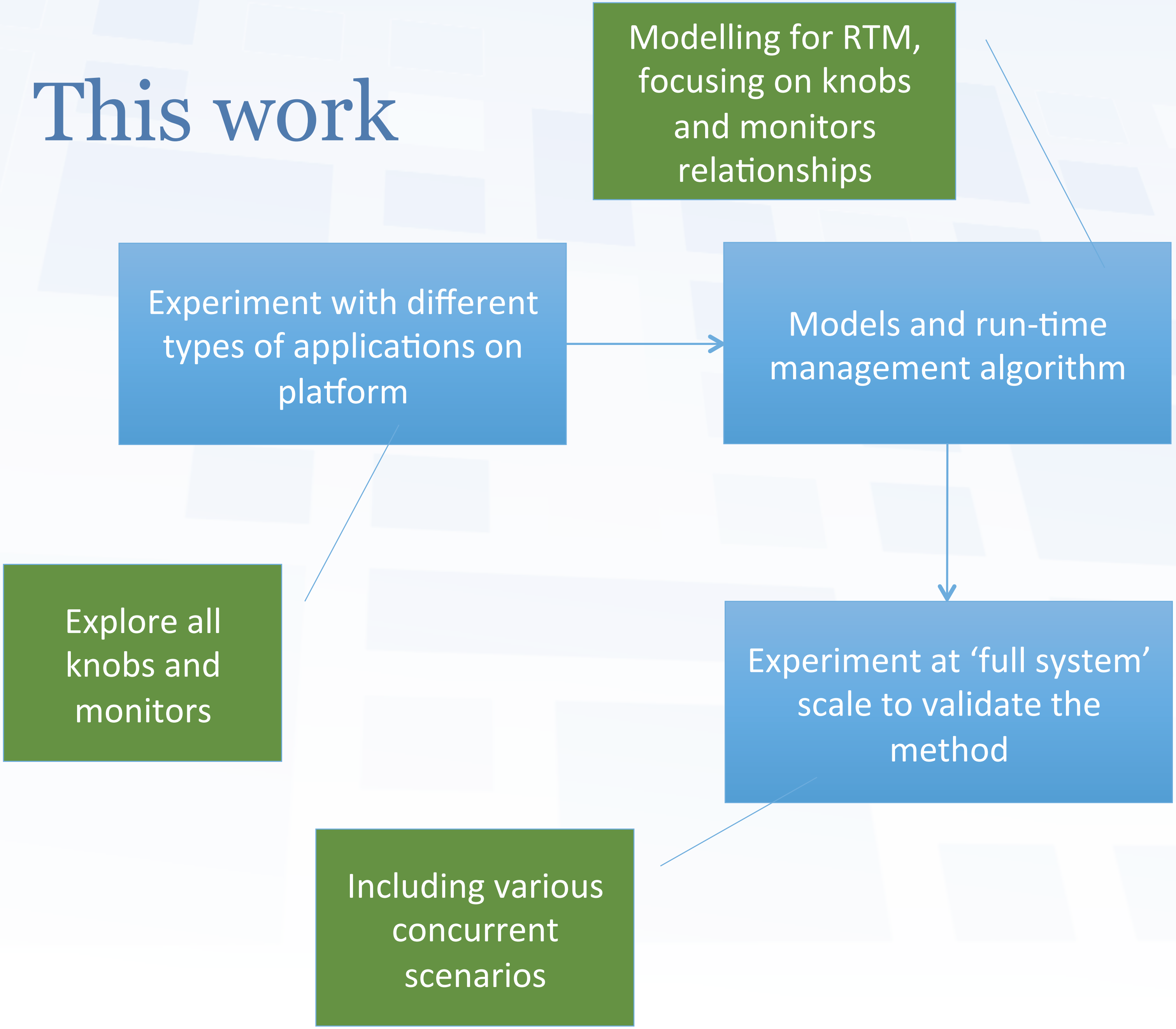
- Controls are implemented through ‘knobs’
  - Parameters that the RTM can directly tune
  - H/W provides power islands, whose V and f are tunable independently
  - H/W provides multiple compute elements (cores)
  - S/W has multiple threads
  - Thread-to-core mapping can also be a knob
  - Re-scheduling to combat faults and other scheduling decisions

RTM



Applications
Application 1
T1 T2 T3 T4 T5 ..
Application N
T8 T9 T3 T4 T5 ..
System Software
Distributed Kernels
OS1 OS2 OSn
Runtime Management
RT1 RT2 RT3 RTN
Controls
DVFS, redundancy......
Core activity, faults.......
Monitors
Hardware
Core 0
800MHz 1.0V
T1 T8
Core 1
1GHz 1.2V
T2 T9
Core M
1.2GHz 1.3V
T3
Interconnect

This work



Modelling for RTM, focusing on knobs and monitors relationships
Experiment with different types of applications on platform
Models and run-time management algorithm
Explore all knobs and monitors
Experiment at ‘full system’ scale to validate the method
Including various concurrent scenarios

Explorative experiments

- A desktop PC around an Intel Core i7
  - Sandy Bridge E with no GPU
  - Four physical cores
  - Hyperthreading for eight logical cores
- The best support (at the time of system design) for monitoring
  - On chip sensing for temperature, power, etc.
  - Full set of performance counters accessible through registers
  - Robust/stable support from tools such as Likwid

Explorative experiments (1)

- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?

PRiME
www.prime-project.org

Explorative experiments (1)

- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?

Measures the power of the entire PC

- Needs extensive gymnastics to relate to processor power consumption
- Found in some exercises
- We needed higher resolution/ precision for correlating to on-chip performance counters!

Explorative experiments (1)

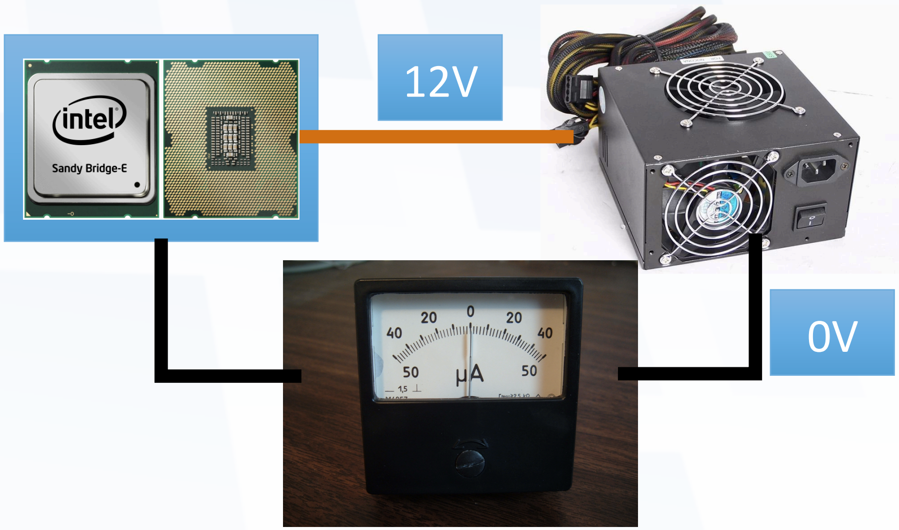
- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?

But what's the alternative?

- Directly measure current close to the CPU block?

Explorative experiments (1)

- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?

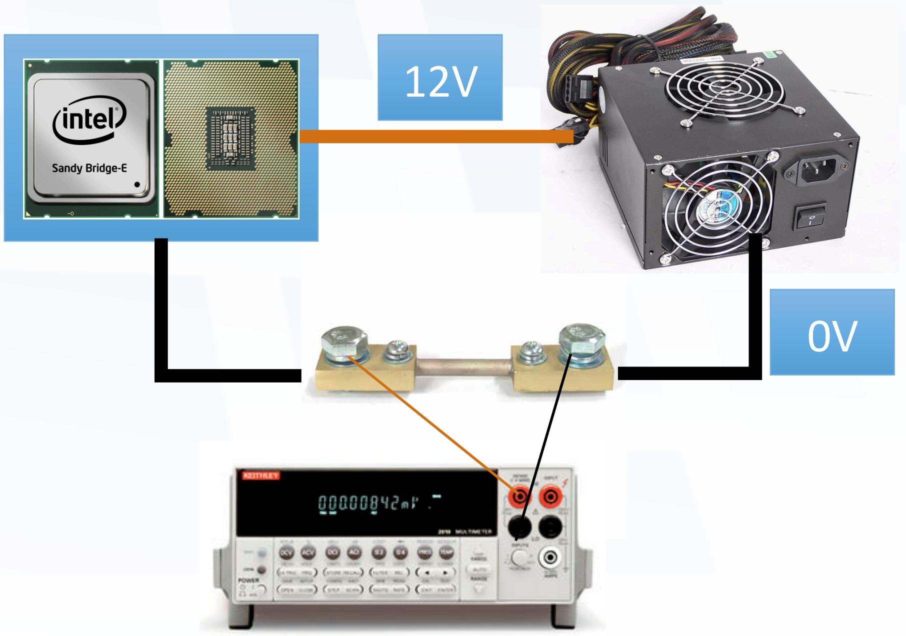


But high-precision digital multimeters are limited to 1A

- And i7 routinely exceed 12W

Explorative experiments (1)

- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?

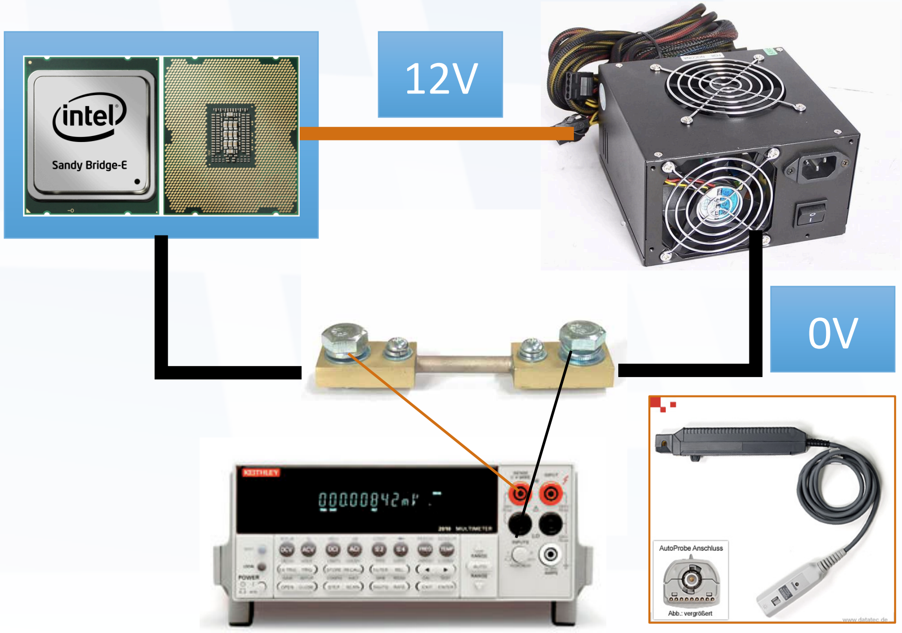


Use a shunt resister

- aka current sensing resister
- Changes current measurement to voltage measurement

Explorative experiments (1)

- H/W experimental support system
  - To improve our own confidence in the on-ship sensing
  - Measuring current through a CPU poses challenges
  - Measure at the CPU or at the wall?



Use a shunt resister

- aka current sensing resister
- Changes current measurement to voltage measurement

Alternative is the clamp probe

Explorative experiments (1-2)

- Direct h/w measurements are expensive
  - Extremely tedious compared to reading performance counter outputs (run experiment, download logged data through USB, collate and analyse data, etc.)
  - Not usable at run-time – we used these to check on Likwid
  - Conducted a number of basic experiments with bespoke benchmarks stressing different parts of the system (typically sqrt and sync as example CPU stress and memory interface stress tasks, also tried finding prime numbers and other more 'general' tasks)
  - Compared with what Likwid reported
  - Confirmed high confidence in Likwid for this system is warranted

Explorative experiments (2)

- Characterization experiments on the system platform with Likwid
  - Established benchmarks from the PARSEC suite
  - Tried benchmarks that are memory-intensive (canneal), cpu-intensive (freqmine) and both (streamcluster)
  - Data collected used in building our models

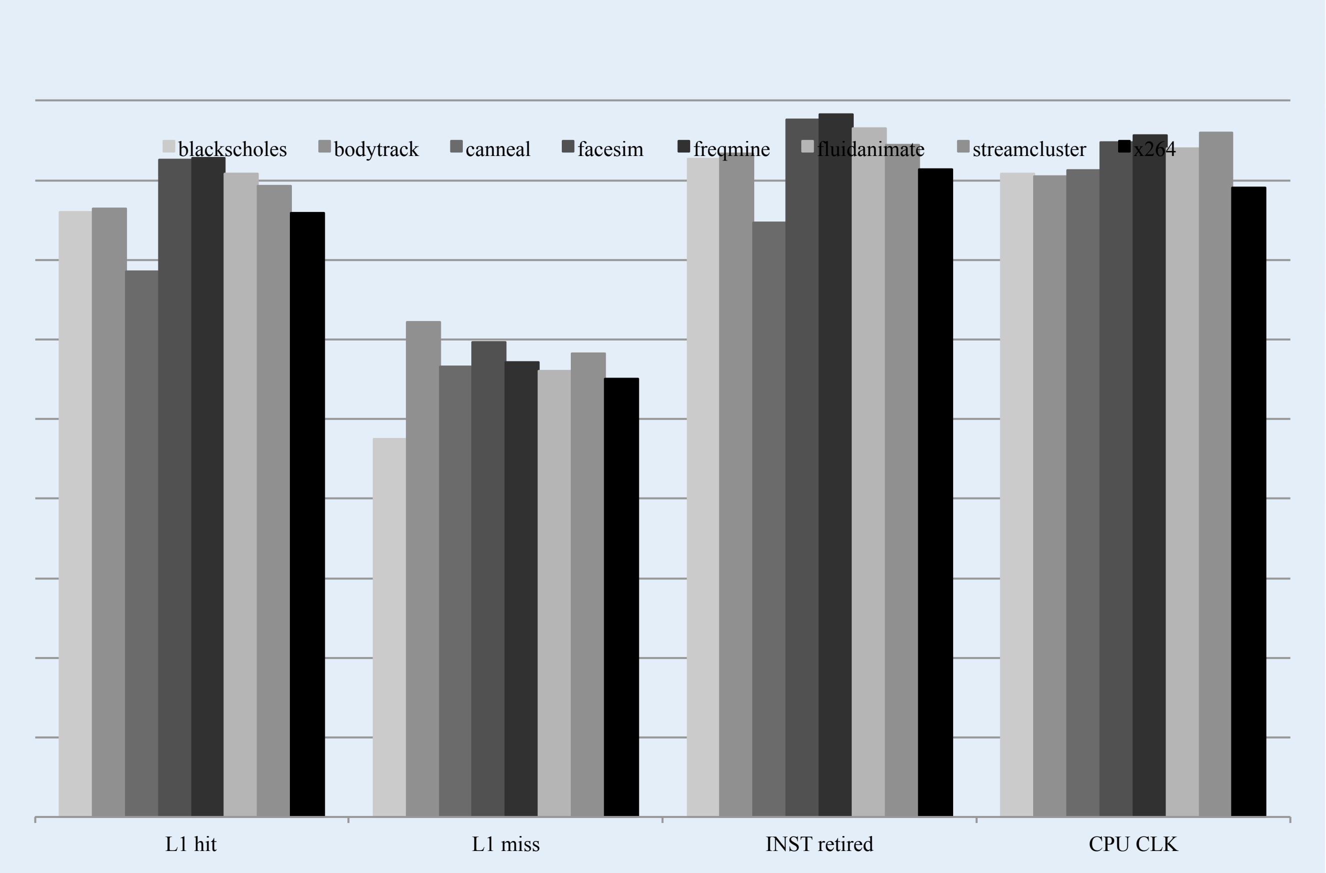
"The Princeton Application Repository for Shared-Memory Computers (PARSEC) is a benchmark suite composed of multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors."

- http://parsec.cs.princeton.edu/

Explorative experiments (2)

- Characterization experiments on the system platform with Likwid
  - Extablis
  - Tried b intensiv
  - Data co

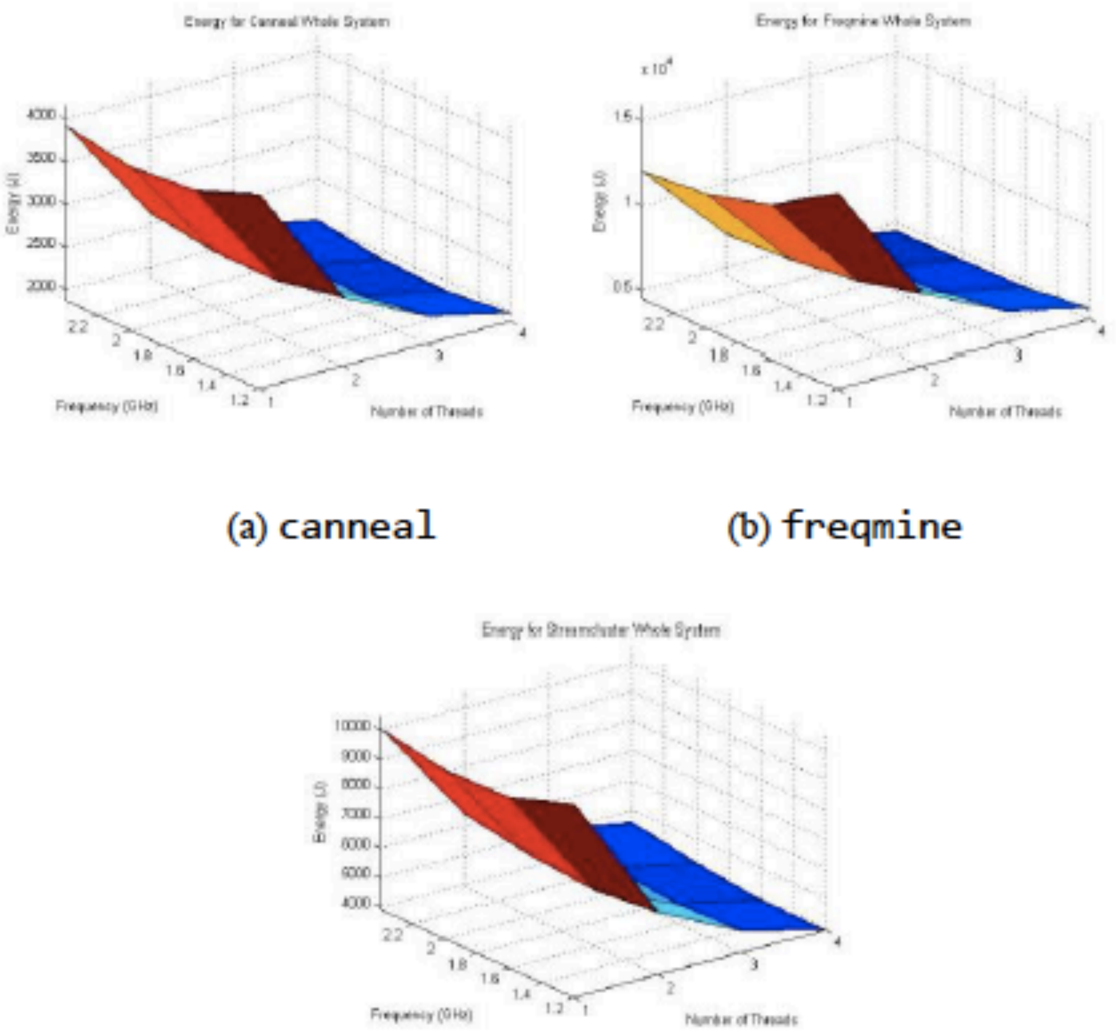
"The Pr (PARSEC suite fo represe multipro



– http://parsec.cs.princeton.edu/

2)

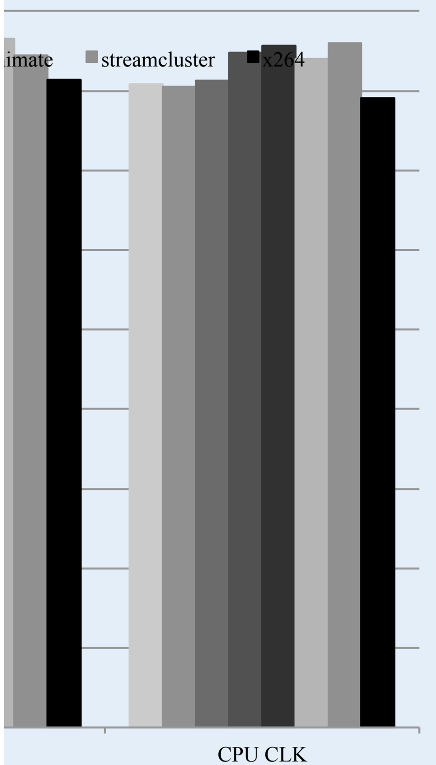
ystem



(a) canneal (b) freqmine

(c) streamcluster

Fig. 2. Energy used for a complete run of each application at different operating frequencies and number of cores allocated, data recorded with Likwid



http://parsec.cs.princeton.edu/

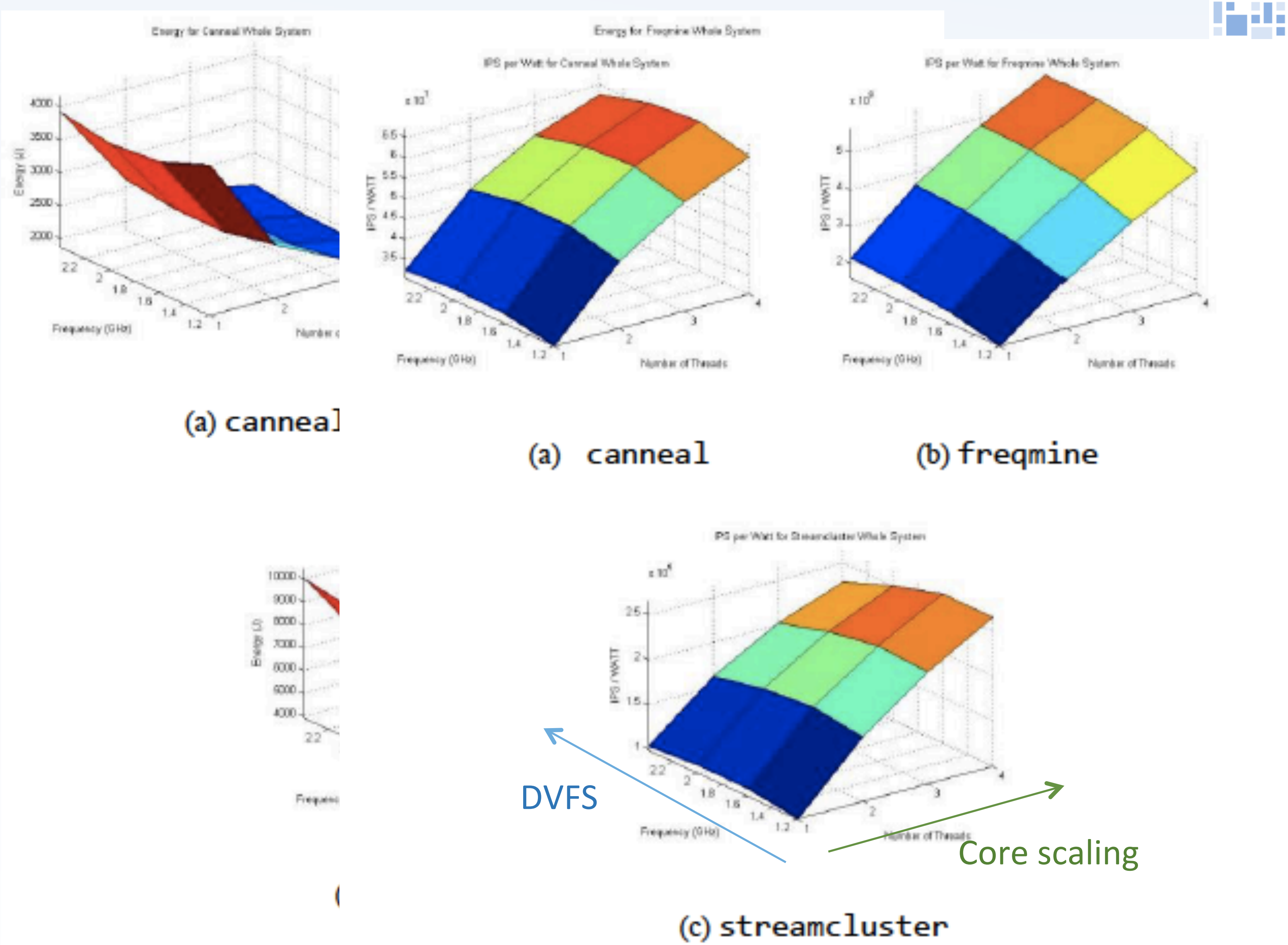


Fig. 2. Energy used for a co
frequencies and number of co

Fig. 3. Average IPS/Watt for a complete run of each application at different operating frequencies and number of cores allocated, data recorded with Likwid

http:

Explorative experiments (3)

- Investigate the behaviour of standard Linux governors as RTMs (benchmarks being run singularly or in parallel)
  - Lowest energy (highest IPS/Watt) always happens with 4 cores
  - IPS/Watt closely follows energy (benchmarks are each a fixed number of instructions so energy per benchmark is proportional to energy per instruction)
  - Mem+CPU and CPU+CPU much better than Mem+Mem
  - Running two copies of the same benchmark gets the same IPS/Watt for running one copy of it (does not modify parallelizability etc.) – no need to investigate this option for RTM

Modelling

- To establish a model linking monitor-able and knob-able parameters (operational state variables) with IPS/Watt
  - Formula for power + formula for throughput

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

Modelling

- To establish a model linking monitor-able and control-able parameters (operational state variables) with IPS/Watt
  - Formula for power + formula for throughput

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

Constants mean linear assumption for static power – one of the simplest approximations

Modelling

PRiME
www.prime-project.org

- To establish a model linking monitor-able and control-able parameters (operational state variables) with IPS/Watt
  - Formula for power + formula for throughput

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

Generally accepted accurate model for dynamic power

Modelling

- To establish a model linking monitor-able and control-able parameters (operational state variables) with IPS/Watt
  - Formula for power + formula for throughput

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

Frequency, number of cores, and cycles per instruction are easily obtainable operational state variables

Modelling

- To establish a model linking monitor-able and control-able parameters (operational state variables) with IPS/Watt
  - Formula for power + formula for throughput

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

State variables include f, V, N and CPI, coefficients include C, α, ϒ and ω

Modelling

- To establish the model we used linear regression
  - It is possible to use this method during run-time with learning

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X,$$

h is the hypothetical function

Modelling

- To establish the model we used linear regression
  - It is possible to use this method during run-time with learning

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X,$$

h is the hypothetical function

n is the number of predictors

Modelling

- To establish the model we used linear regression
  - It is possible to use this method during run-time with learning

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X,$$

h is the hypothetical function

n is the number of predictors

X are the predictors – a predictor may be a non-linear function of independent variable(s), e.g. $N*f*V^2$

Modelling

- To establish the model we used linear regression
  - It is possible to use this method during run-time with learning

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X,$$

Θ are the fitting coefficients – 'linear' means h is linear with Θ

h is the hypothetical function

n is the number of predictors

X are the predictors – a predictor may be a non-linear function of independent variable(s), e.g. $N*f*V^2$

Modelling

- To establish the model we used linear regression
  - It is possible to use this method during run-time with learning

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X,$$

Θ are the fitting coefficients – 'linear' means h is linear with Θ

h is the hypothetical function

n is the number of predictors

X are the predictors – a predictor may be a non-linear function of independent variable(s), e.g. $N*f*V^2$

Regression ➔ fitting to optimize some metric – min squared prediction error

Modelling

- Predictors from state variables f, V, N and CPI
  - h should be based on the following formulas
  - but IPS/Watt is not linear with an easy to find Θ
  - so we have h=Watt and then combine with IPS, which has no coefficients needing fitting

$$P_{total} = P_{static} + P_{dynamic}$$

$$P_{static} = \gamma V + \omega,$$

$$P_{dynamic} = \alpha C V^2 f,$$

$$IPS = \frac{fN}{CPI},$$

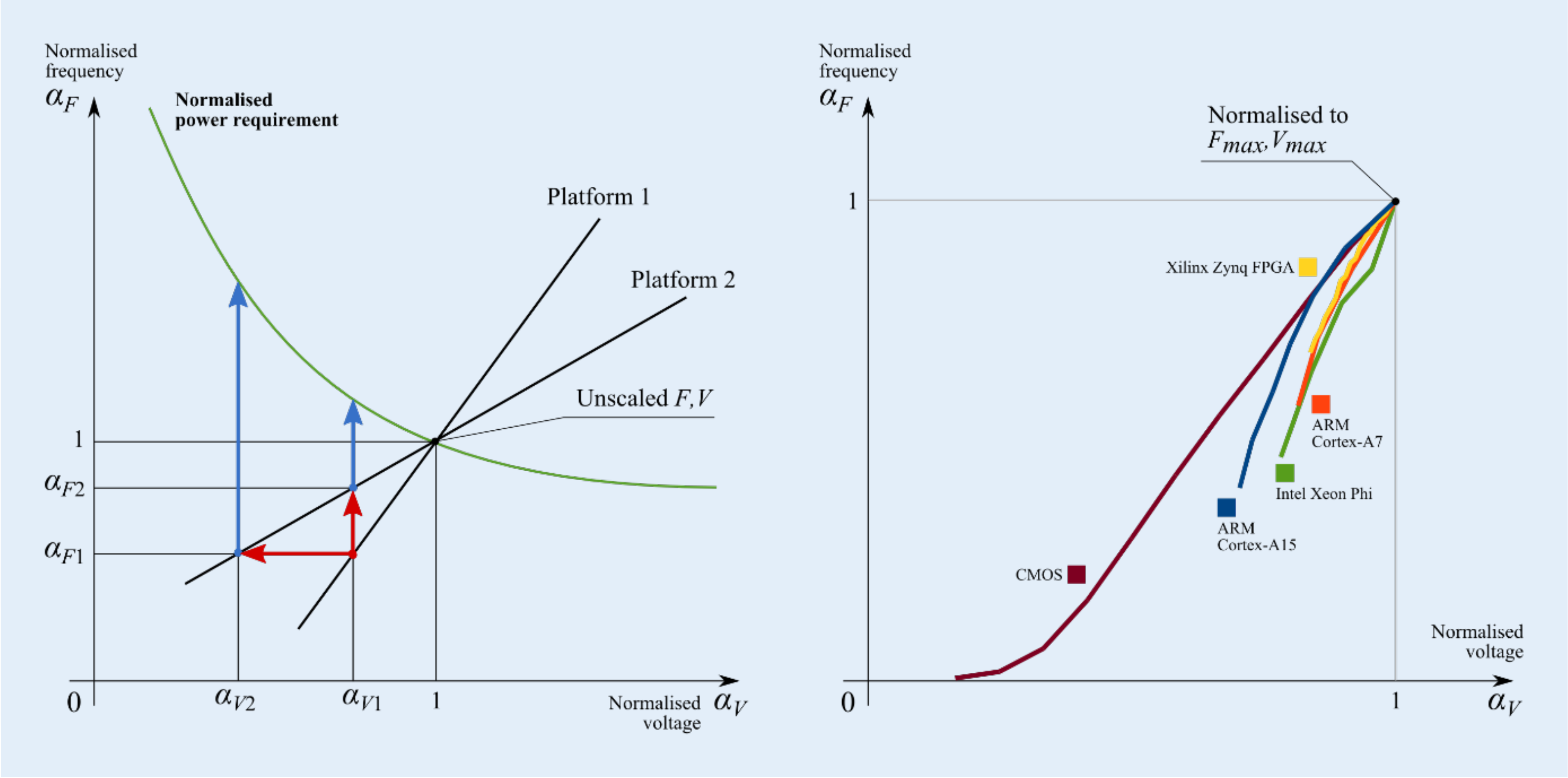
Modelling

- For commercially available platforms, it is usual that f and N are not independent with each other
  - So we can reduce the number of state variables by 1

$$V = \varphi f + \beta,$$

Modelling

- For commercially available platforms, it is usual



Modelling

- Predictors from state variables f, ~~V~~, N and CPI
  - h=Watt should then be

$$Watt = h_\theta = \theta_0 + \theta_1 \boxed{Nf} + \theta_2 \boxed{Nf^2} + \theta_3 \boxed{Nf^3}$$

$$x_1 = Nf, x_2 = Nf^2, x_3 = Nf^3$$

Modelling

- Hypothetical function for IPS/Watt

$$\frac{IPS}{Watt} = \frac{fN}{CPI(\theta_0 + \theta_1 Nf + \theta_2 Nf^2 + \theta_3 Nf^3)},$$

Modelling

- Model reduction

$$\frac{IPS}{Watt} \approx \frac{fN(1\times 10^9)}{CPI(11.061 + 0.645fN + 1.4351f^2)}$$

$$\frac{IPS}{Watt} = \frac{fN}{CPI(\theta_0 + \theta_1 Nf + \theta_2 Nf^2 + \theta_3 Nf^3)},$$

Modelling

- Model reduction

$$\frac{IPS}{Watt} \approx \frac{fN(1\times10^9)}{CPI(11.061 + 0.645fN + 1.4351f^2)}$$

$$\frac{IPS}{Watt} = \frac{fN}{CPI(\theta_0 + \theta_1 Nf + \theta_2 Nf^2 + \theta_3 Nf^3)},$$

$\theta_3 = 0$

Modelling

- Model reduction

$$\frac{IPS}{Watt} \approx \frac{fN(1 \times 10^9)}{CPI(11.061 + 0.645fN + 1.4351f^2)}$$

$$\frac{IPS}{Watt} = \frac{fN}{CPI(\theta_0 + \theta_1 Nf + \theta_2 Nf^2 + \theta_3 Nf^3)},$$

θ_3=0

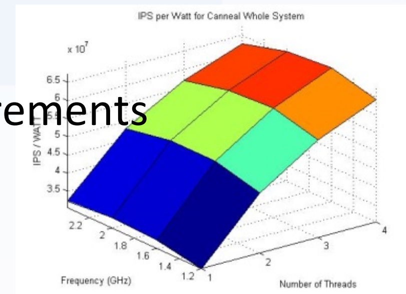
predictor change!

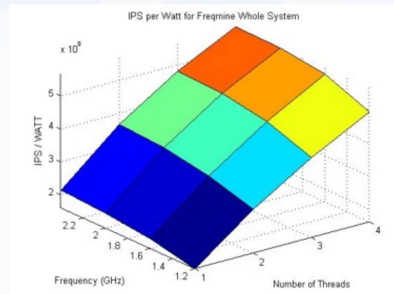
Modelling

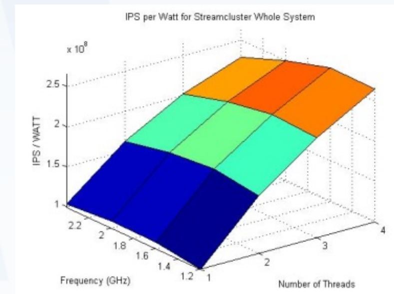
- This level of model reduction is justified by
  - Common practice in many branches of engineering (chemical, biological, materials, mechanical etc.)
  - The fitting quality is very high (going with the ‘right’ function does not further improve it significantly)
  - The model will be used in run-time optimization, which is a discrete-space programming problem (both models would give the same optimal operation points for all the examples we tried)
  - In future run-time use, simpler functions are easier for learning – lower overhead for the RTM (1 fewer predictor can cause a substantial reduction in learning overhead)

Model vs. measurements

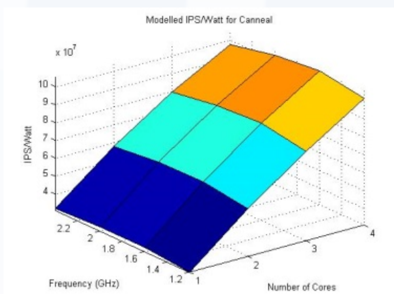
measurements

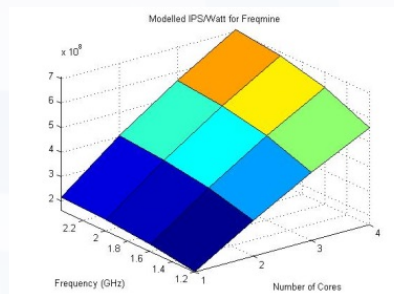


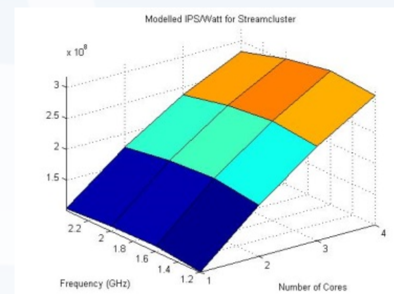




model







memory CPU memory+CPU

quantitatively close, qualitatively the same

RTM design

Algorithm 1: Run-time optimization

1. Check PID changes
2. **If** application scenario changed?
3. Obtain PID of new application
4. Calculate CPI of application
5. Calculate IPS/Watt using model (8)
6. Allocate cores to application
7. Change frequency for max(IPS/Watt)
8. **End if**
9. Wait for next activation

PID tracks processes being run or joining

RTM implementation

Python script running in conjunction with OS implementing the RTM

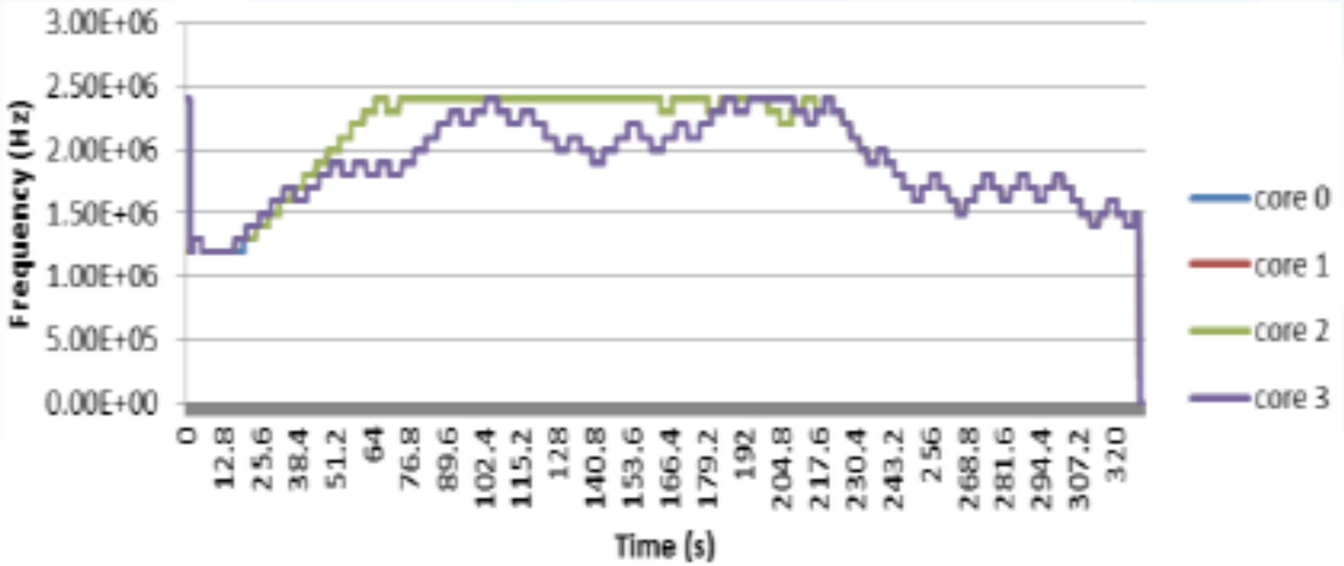
```
while true
do
    if [[ $PID -eq 0 ]]
    then
        governor="ondemand"
        if [[ $m -eq 1 ]]
        then
            set_governor
            m=0
        fi
        PID=$( cat PID.txt )
        echo "no program running"
        sleep 1
    else
        echo "new program added"
        governor="userspace"
        PID=$( cat PID.txt )
        j=0
        PID_CPI_array
        j=0
        if [[ $m -eq 0 ]]
        then
            set_governor
            m=1
        fi
        j=0
        CORE_Allocation
        j=0
        Linear_model
        j=0
        sleep 2
    fi
done
```

Results

- General non-trivial improvements on IPS/Watt over standard Linux governors recorded
  - Between a few % points and 23%

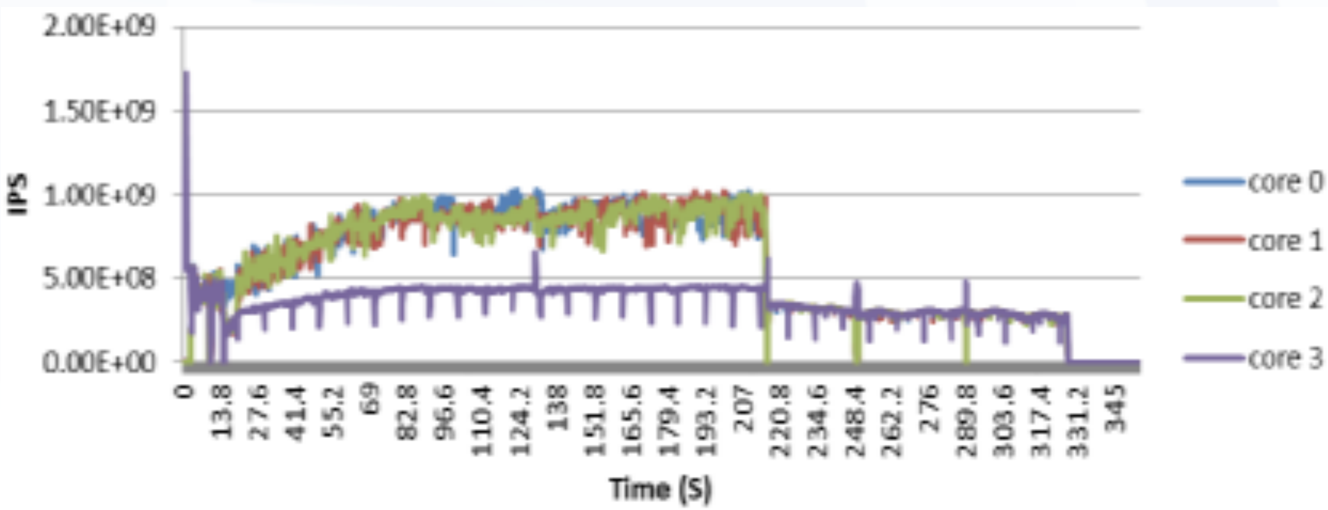
Results

- Traces show that the RTM changing frequency and task to core mapping frequently



Results

- IPS changes as a result of the RTM's decisions



This work

Modelling for RTM, focusing on knobs and monitors relationships

Experiment with different types of applications on platform

Models and run-time management algorithm

Explore all knobs and monitors

Experiment at 'full system' scale to validate the method

Including various concurrent scenarios

Need to try a lot more cases: system architectures, app. bechmarks, different CPI granularities, etc.