Power Management
Selected Issues

COMP9242 2007/S2 Week 8

Power Management Issues

Laptops

- Obvious objective: prolong disconnected operating
  ➜ maximise battery lifetime

Power Management Issues

Laptops

- Obvious objective: prolong disconnected operating
  - ➜ maximise battery lifetime
- Assume: fixed battery capacity C
  - ➜ smaller discharge current $I = \frac{dQ}{dt}$ increases usable time
  - ➜ objective is to reduce average power consumption $P = VI$
  - ➜ reality more complex: strong current drains battery more

Power Management Issues

Laptops

- Obvious objective: prolong disconnected operating
  - ➜ maximise battery lifetime
- Assume: fixed battery capacity C
  - ➜ smaller discharge current $I = \frac{dQ}{dt}$ increases usable time
  - ➜ objective is to reduce average power consumption $P = VI$
  - ➜ reality more complex: strong current drains battery more
- Approaches:
  - ➜ spin down disk when idle
    - ➜ trade-off with response time
  - ➜ put peripherals into low-power mode when idle
  - ➜ blank screen
  - ➜ reduce clock frequency

Power Management Issues

Laptops

- Obvious objective: prolong disconnected operating
  - ➜ maximise battery lifetime
- Assume: fixed battery capacity C
  - ➜ smaller discharge current $I = \frac{dQ}{dt}$ increases usable time
  - ➜ objective is to reduce average power consumption $P = VI$
  - ➜ reality more complex: strong current drains battery more
- Approaches:
  - ➜ spin down disk when idle
    - ➜ trade-off with response time
  - ➜ put peripherals into low-power mode when idle
  - ➜ blank screen
  - ➜ reduce clock frequency
    - why does this help?

Power Management Issues

Desktops

- Objectives: prevent overheating of CPU core

Power Management Issues

Desktops

- Objectives: prevent overheating of CPU core
- Approaches:
  - ➜ reduce clock frequency

Power Management Issues

Desktops

- Objectives: prevent overheating of CPU core
- Approaches:
  - ➜ reduce clock frequency
    - why does this help?

Power Management Issues

Servers

- Objectives:
  - prevent overheating of CPU core
  - reduce overall *power intake*
  - reduce overall *heat dissipation*

Power Management Issues

Servers

- Objectives:
  - prevent overheating of CPU core
  - reduce overall *power intake*
  - reduce overall *heat dissipation*
- Note: about 1/3 of lifetime cost of hosting centres is power!
  - ➜ for computer power supply
  - ➜ for air conditioning

Power Management Issues

Servers

- Objectives:
  - prevent overheating of CPU core
  - reduce overall *power intake*
  - reduce overall *heat dissipation*
- Note: about 1/3 of lifetime cost of hosting centres is power!
  - ➜ for computer power supply
  - ➜ for air conditioning
- Approaches:
  - ➜ reduce clock frequency
  - ➜ spin down disks
  - ➜ shut down nodes
    - use virtualization to migrate load

General Power Management Issues

- Power-performance tradeoff:
  - ➜ transitioning between power modes takes time
  - ➜ time lost for transition to high-power mode degrades performance

General Power Management Issues

- Power-performance tradeoff:
  - ➜ transitioning between power modes takes time
  - ➜ time lost for transition to high-power mode degrades performance
- Cost-savings tradeoff:
  - ➜ transitioning between power modes costs energy
  - ➜ if transitioning too aggressively may consume more energy

General Power Management Issues

- Power-performance tradeoff:
  - ➜ transitioning between power modes takes time
  - ➜ time lost for transition to high-power mode degrades performance
- Cost-savings tradeoff:
  - ➜ transitioning between power modes costs energy
  - ➜ if transitioning too aggressively may consume more energy
  - ➜ rule of thumb for transition to low-power mode:
    - transition when wasted energy exceeds transitioning cost

Low-power modes

- Disk:
  - ➜ spin down

Low-power modes

- Disk:
  - ➜ spin down
- Memory:
  - ➜ loss-free low-power state
    - content not accessible until return to normal state
    - fast transition
    - moderate savings
  - ➜ destructive low-power state
    - content lost
    - slow transition
    - high savings

Low-power modes

- Disk:
  - ➜ spin down
- Memory:
  - ➜ loss-free low-power state
    - content not accessible until return to normal state
    - fast transition
    - moderate savings
  - ➜ destructive low-power state
    - content lost
    - slow transition
    - high savings
- CPU:
  - ➜ dynamic voltage and frequency scaling (DVFS)

Dynamic Voltage and Frequency Scaling

Basis of DVFS:

1. DRAM circuits draw power only when switching
   - ➜ switching energy (approximately) constant per clock cycle:

$$P = VI \propto V^2$$

   - ➜ switches per second = core frequency:

$$P \propto f$$

   - ➜ overall:

$$P(f) \propto fV^2$$

Dynamic Voltage and Frequency Scaling

Basis of DVFS:

1. DRAM circuits draw power only when switching
   - ➜ switching energy (approximately) constant per clock cycle:

$$P = VI \propto V^2$$

   - ➜ switches per second = core frequency:

$$P \propto f$$

   - ➜ overall:

$$P(f) \propto fV^2$$

2. DRAM circuits switch faster under higher voltage
   - ➜ can reduce voltage at lower frequencies (to a limit):

$$V_{min} \text{ monotonic in } f$$

$$P_{min} \propto f^2 \cdots f^3$$

Dynamic Voltage and Frequency Scaling

Basis of DVFS:

3. Work performed per clock cycle is independent of frequency
   - ➜ computation takes fixed number of cycles N
   - ➜ total computation time is:

$$T = N/f$$

   - ➜ energy for computation is:

$$E = PT \propto V^2$$

   ... independent of f!

Dynamic Voltage and Frequency Scaling

Basis of DVFS:

3. Work performed per clock cycle is independent of frequency
   - ➜ computation takes fixed number of cycles N
   - ➜ total computation time is:

$$T = N/f$$

   - ➜ energy for computation is:

$$E = PT \propto V^2$$

     ... independent of f!
   - ➜ frequency scaling alone doesn't save energy
     - but reduces core temperature
   - ➜ energy saving comes from ability to reduce voltage with frequency

Dynamic Voltage and Frequency Scaling

Basis of DVFS:

3. Work performed per clock cycle is independent of frequency
   - ➜ computation takes fixed number of cycles N
   - ➜ total computation time is:

$$T = N/f$$

   - ➜ energy for computation is:

$$E = PT \propto V^2$$

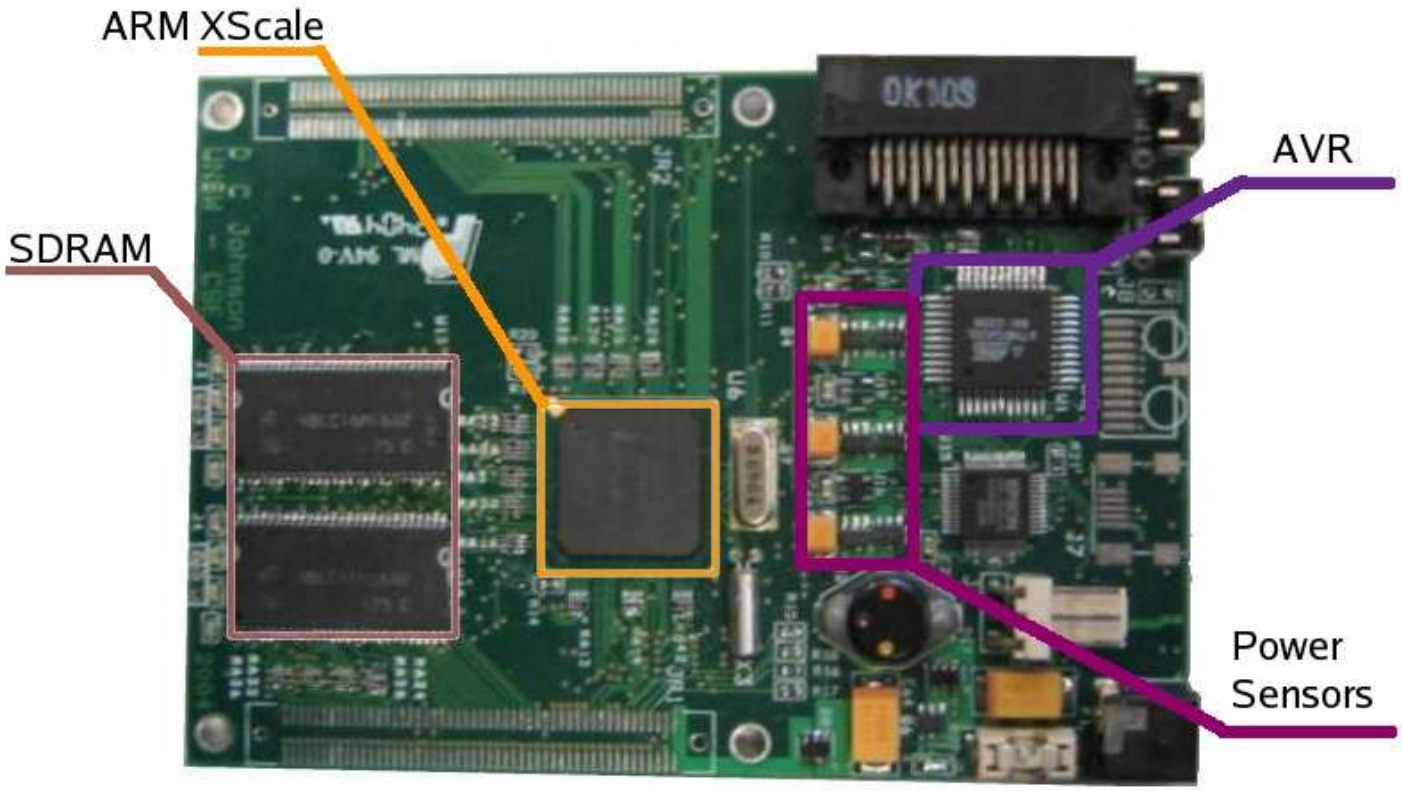
     ... independent of f!
   - ➜ frequency scaling alone doesn't save energy
     - but reduces core temperature
   - ➜ energy saving comes from ability to reduce voltage with frequency

- **Let's have a look at real hardware!**

Hardware Platform

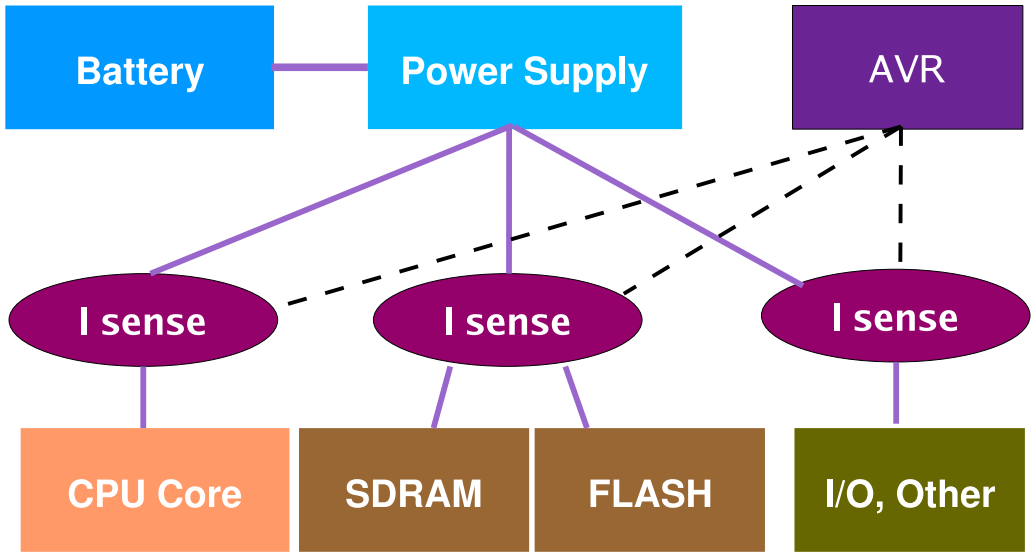
PLEB 2 (NICTA '05):

- Single board computer based on Intel XScale (PXA255)
- Representative of a typical embedded system



Hardware Platform

- Three power supplies: CPU, Memory and IO
- Each power supply's current is measured
- On-board micro-controller (AVR) for monitoring
- Samples taken periodically according to AVR clock
- $f_{\max}$ samples is 15 KHz, subdivided into three channels



PXA255 Voltage and Frequency Settings

Valid voltage/frequency configurations:

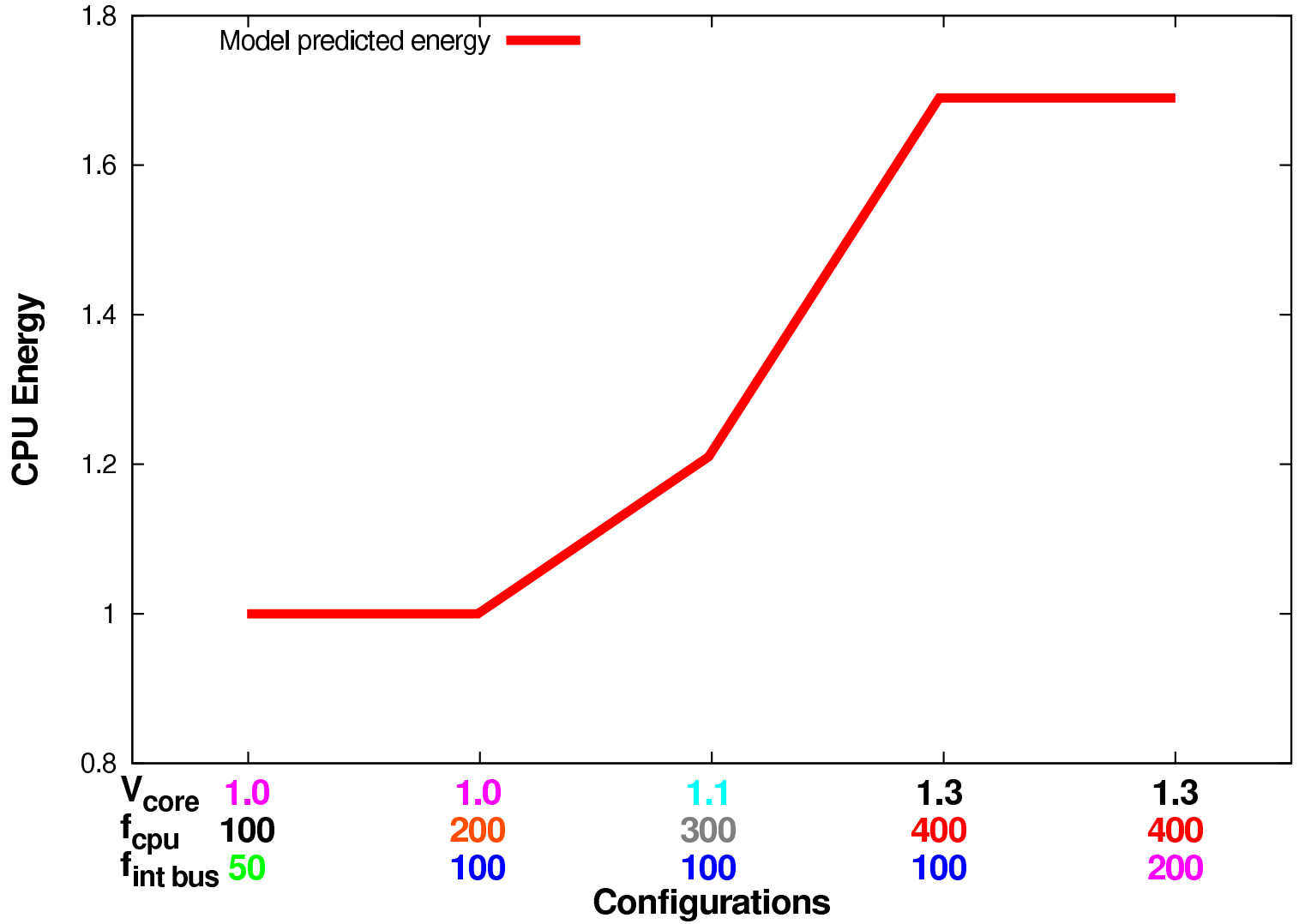
V_{core} (V)	f_{cpu} (MHz)	$f_{internalbus}$ (MHz)	f_{mem} (MHz)
1.0	100	50	100
1.0	200	100	100
1.1	300	100	100
1.3	400	100	100
1.3	400	200	100

Benchmarks

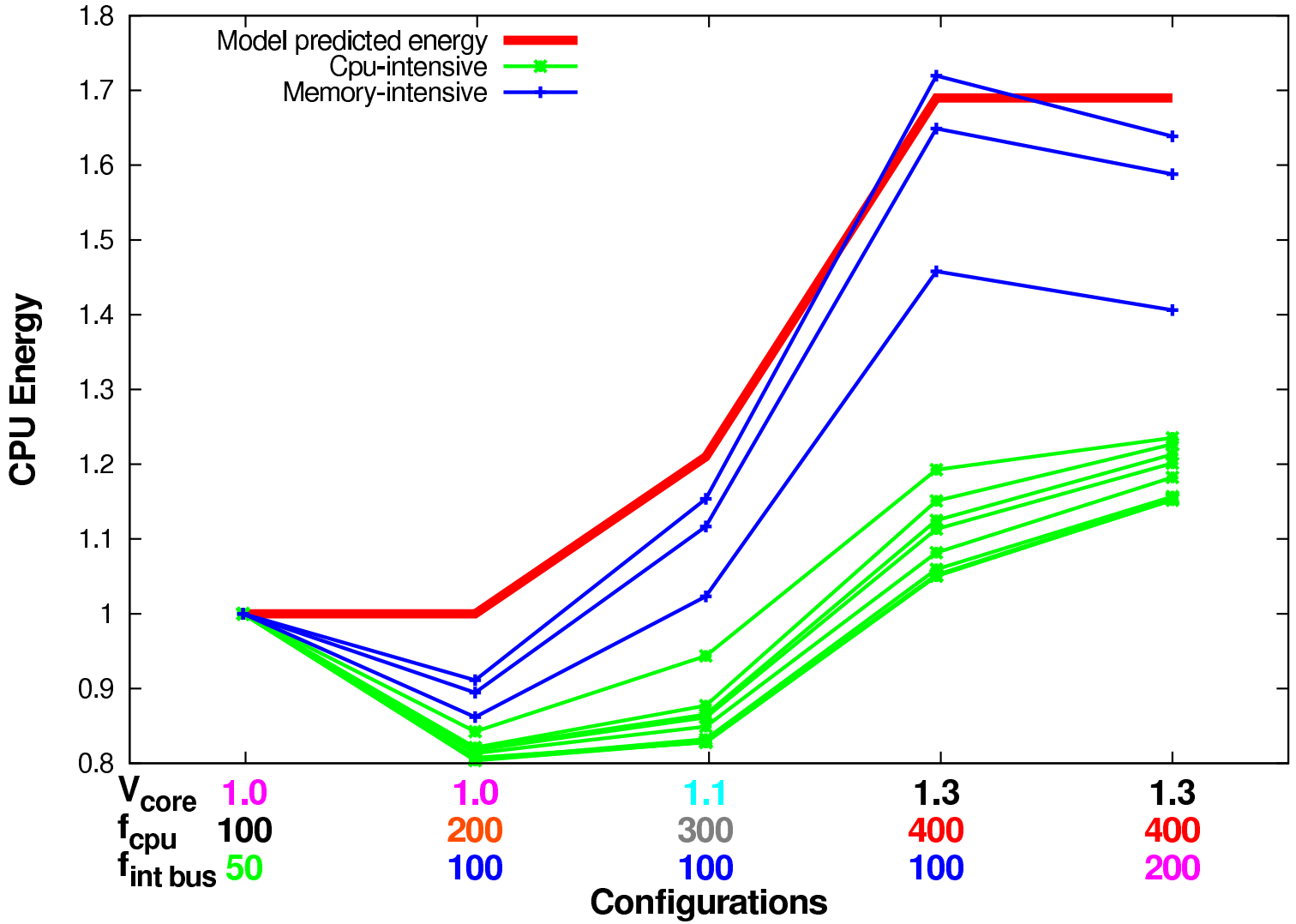
14 Macro benchmarks:

- ➜ most from MiBench, selected according to resource limitations
- ➜ automotive, industrial control, network, office, signal processing
- ➜ plus 4 more: gzip, MP3, image analysis, speech compression

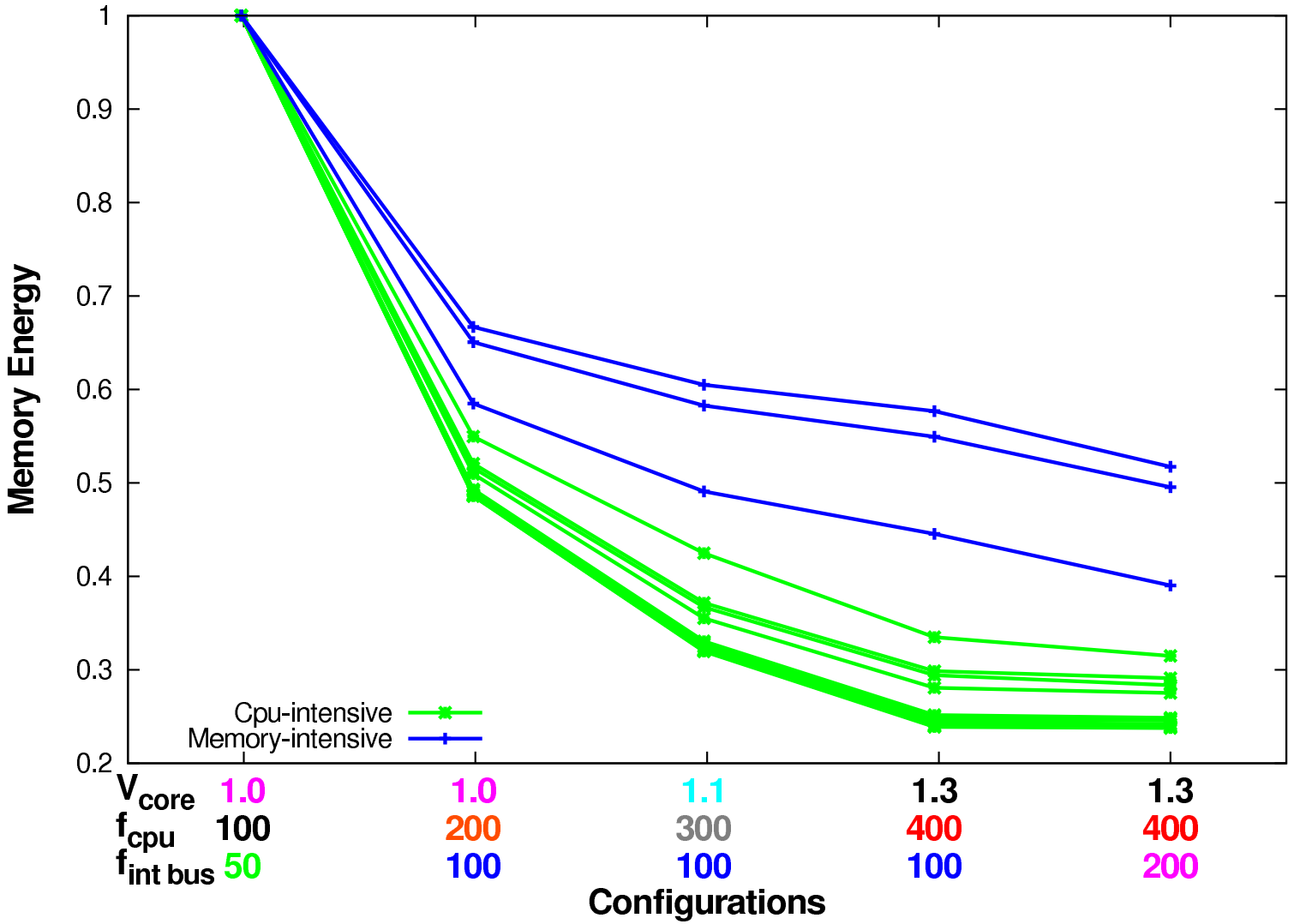
1. Normalized CPU Energy Model



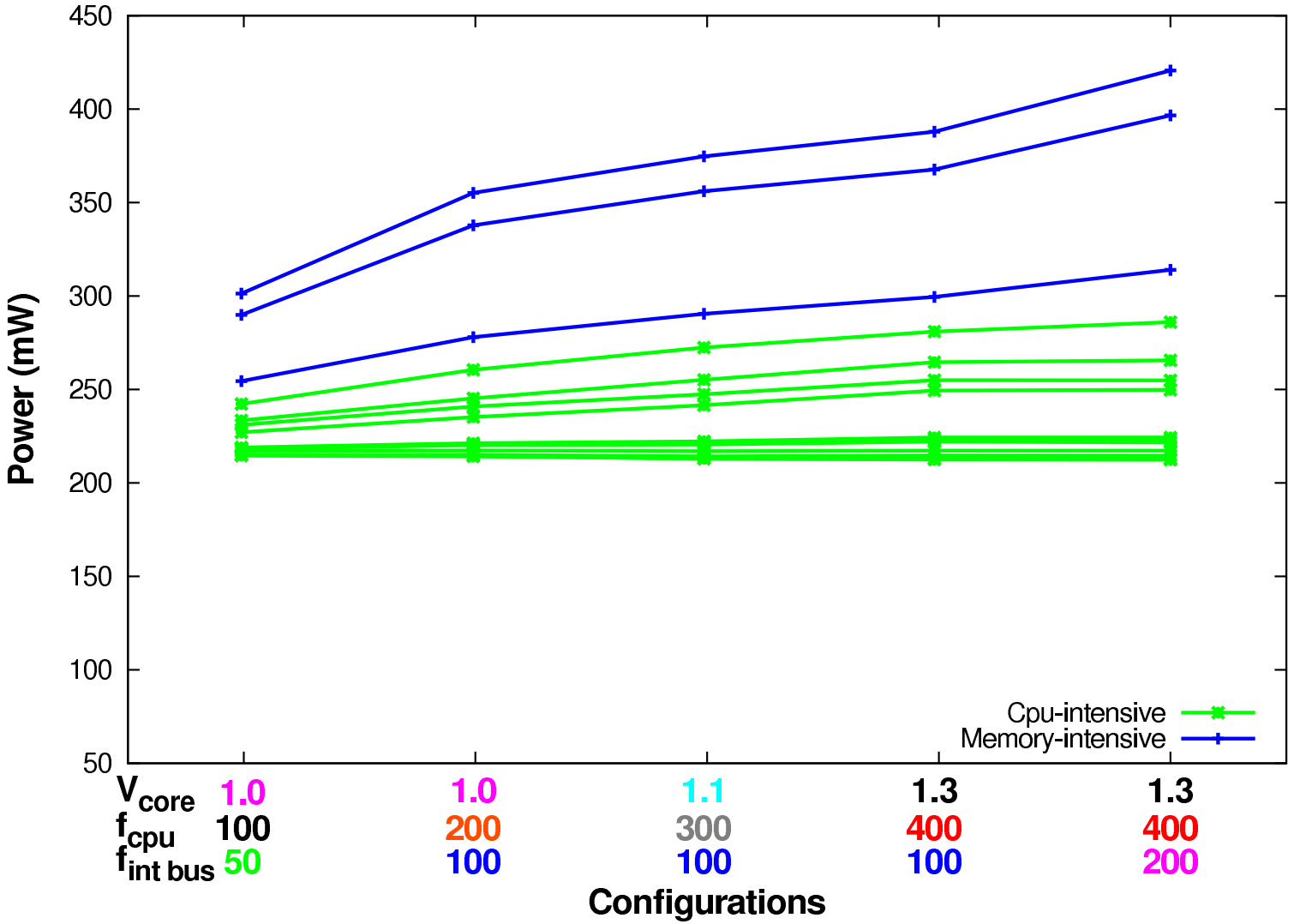
1. Normalized CPU Energy



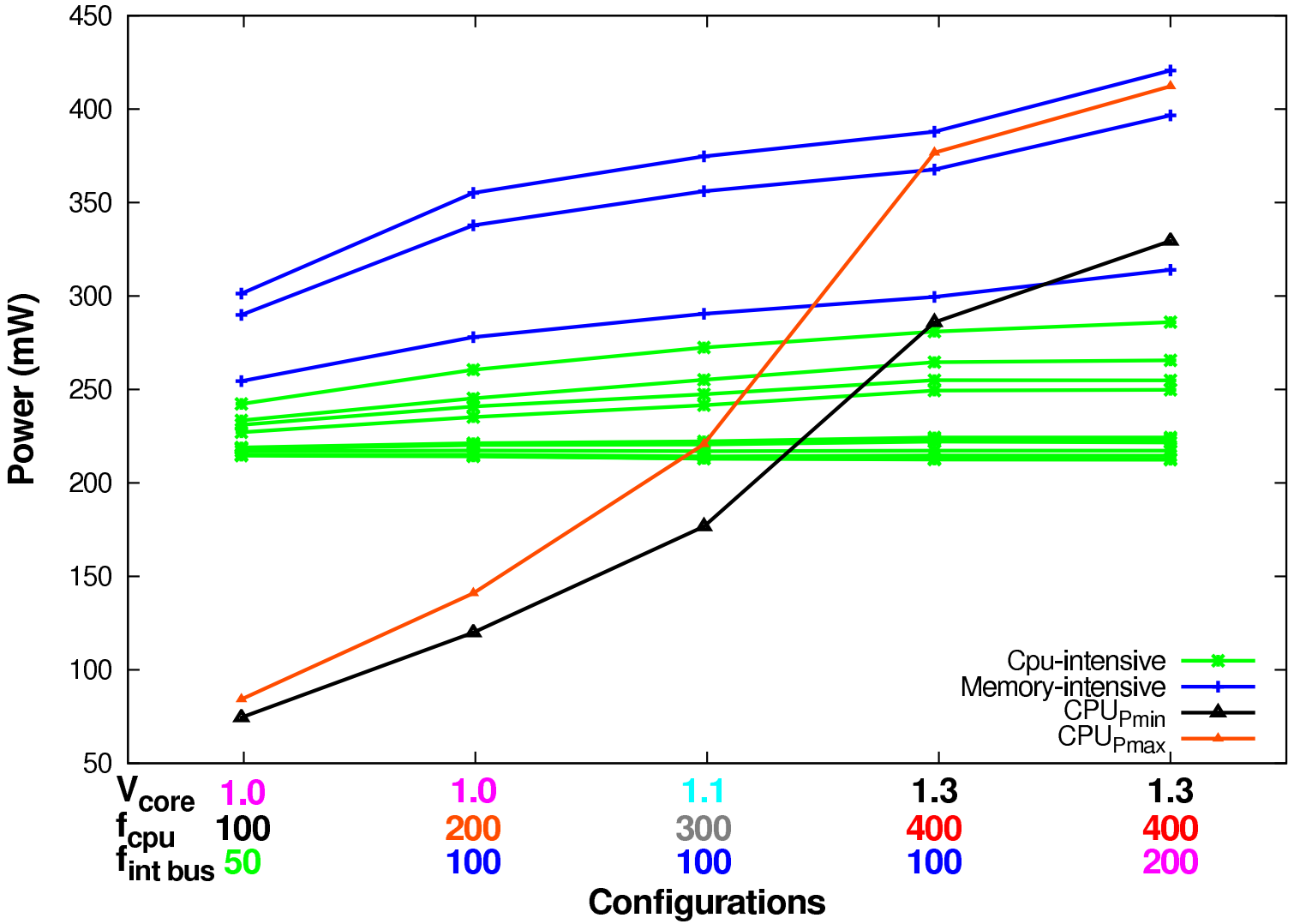
1.B Normalized Memory Energy



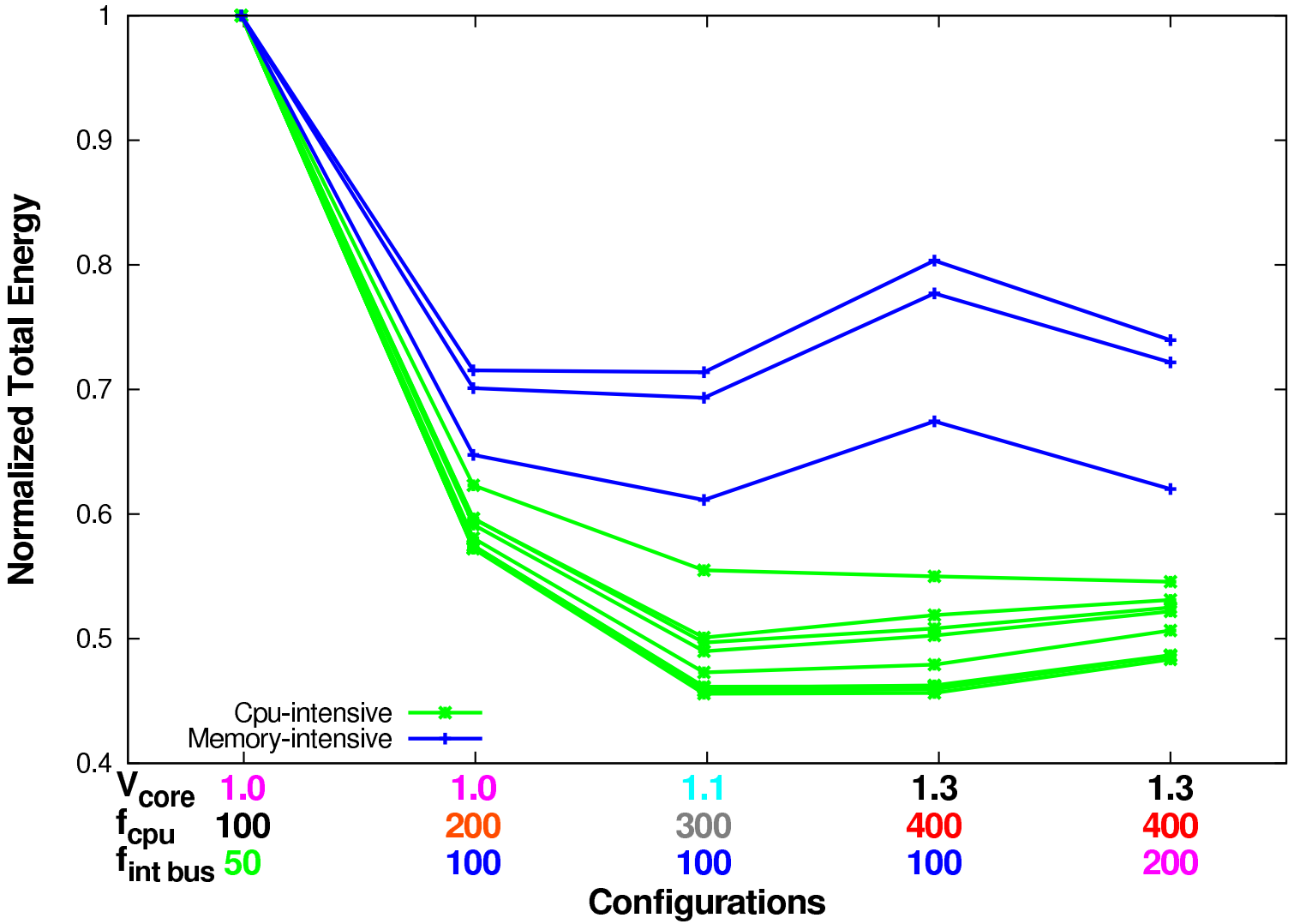
2. Memory Power



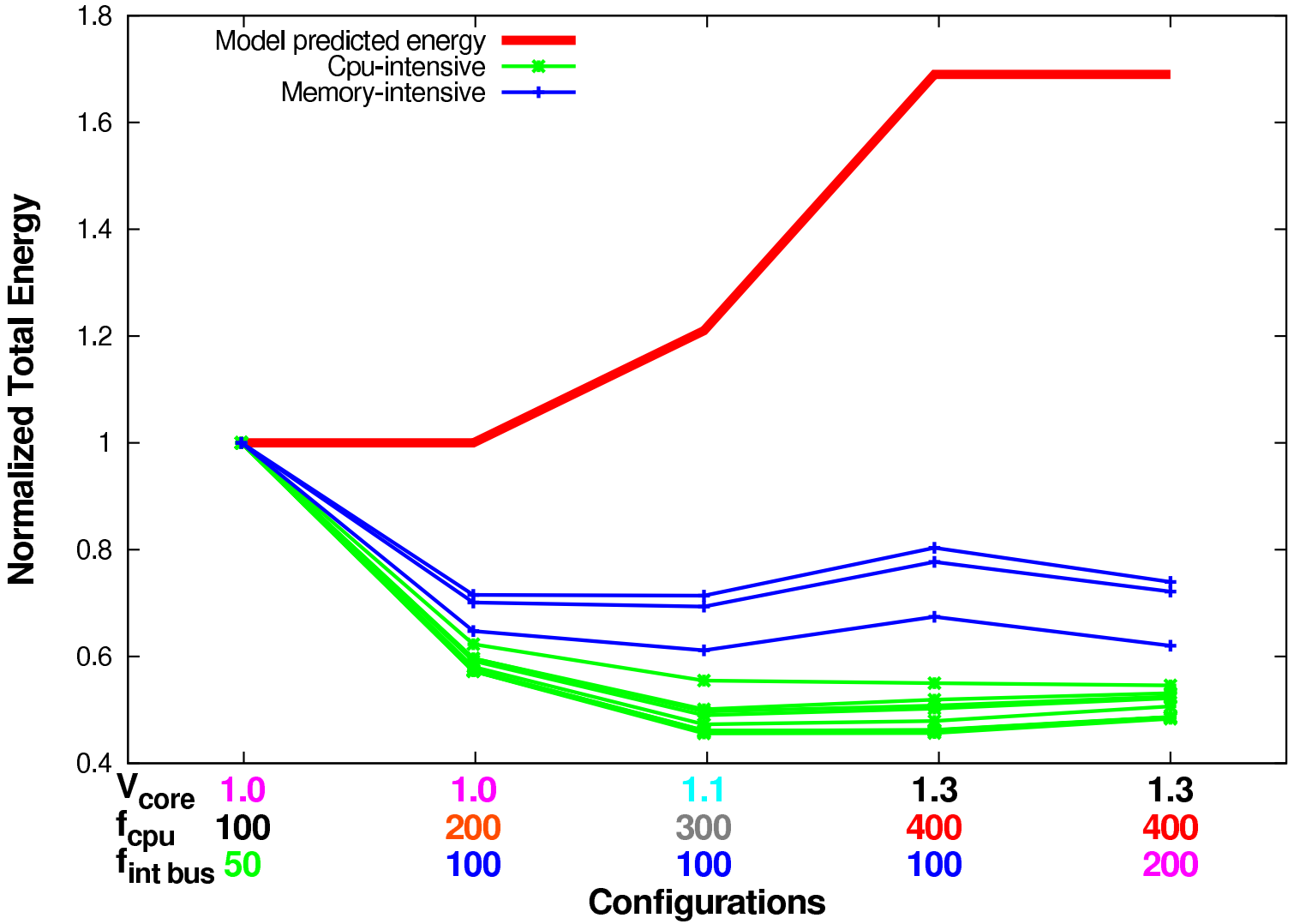
2. Memory Power vs CPU Power



3. Normalized Total Energy



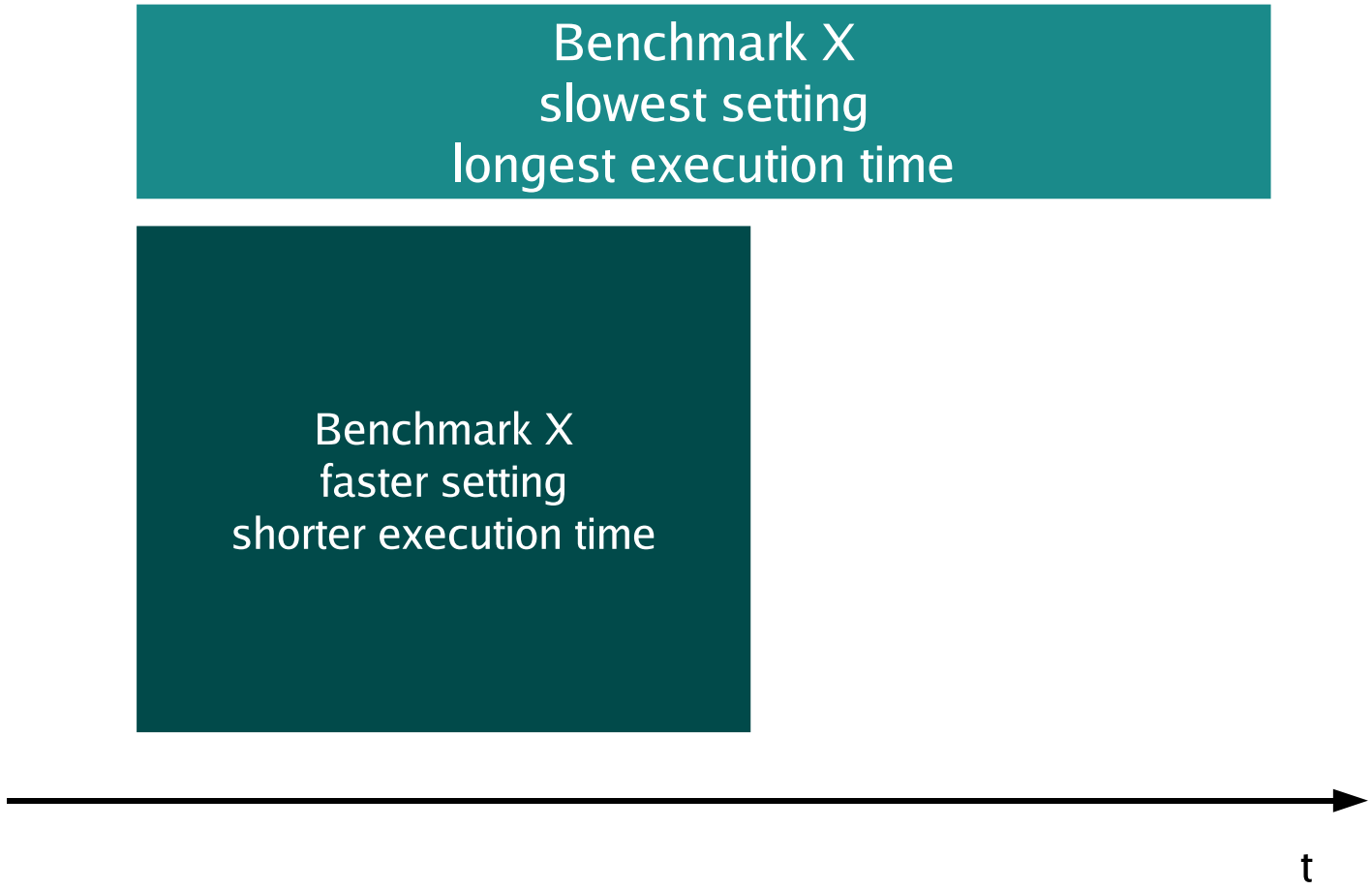
3. Normalized Total Energy vs Naive Model



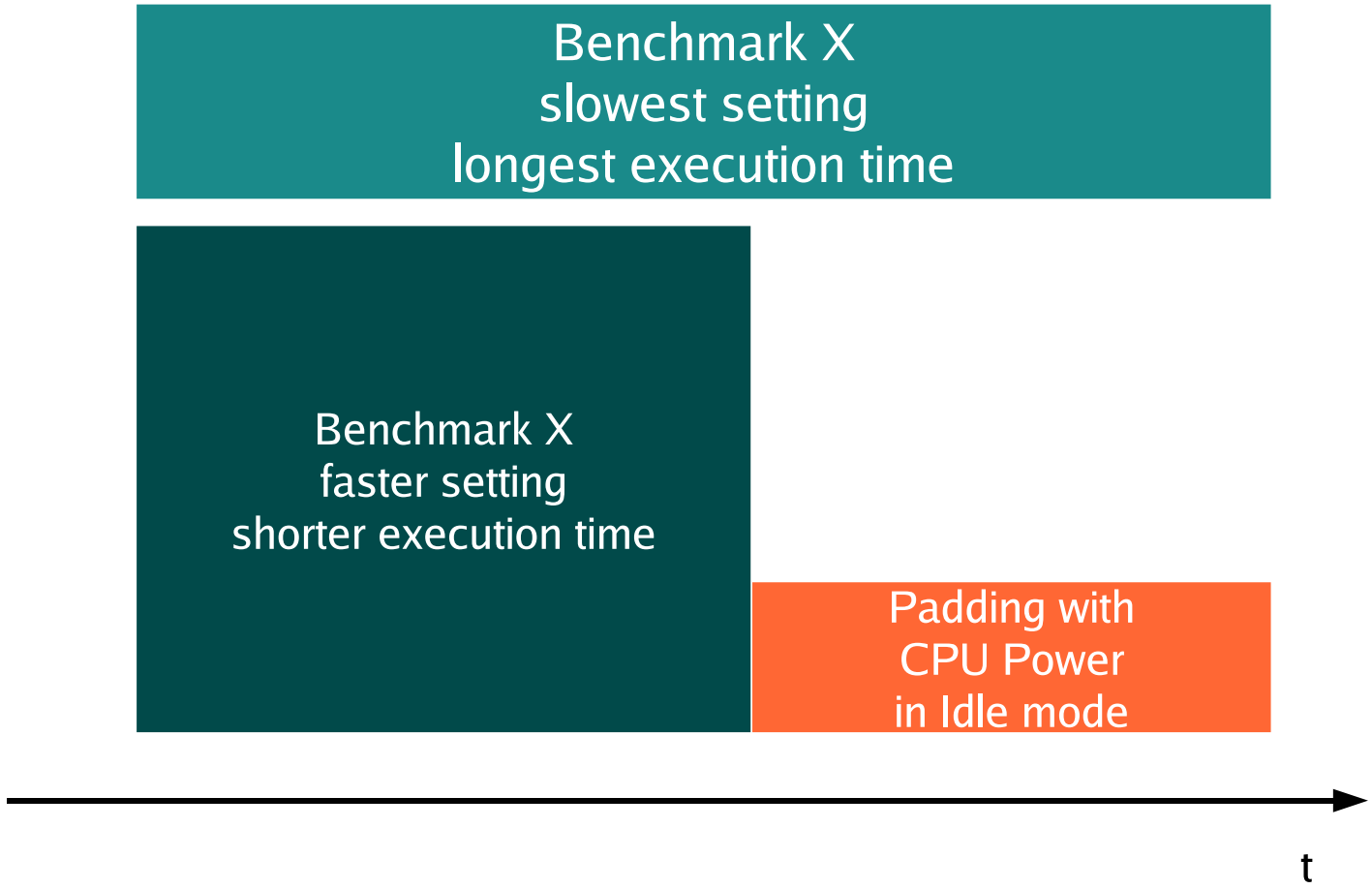
Padding with idle power to equal total time

Benchmark X
slowest setting
longest execution time

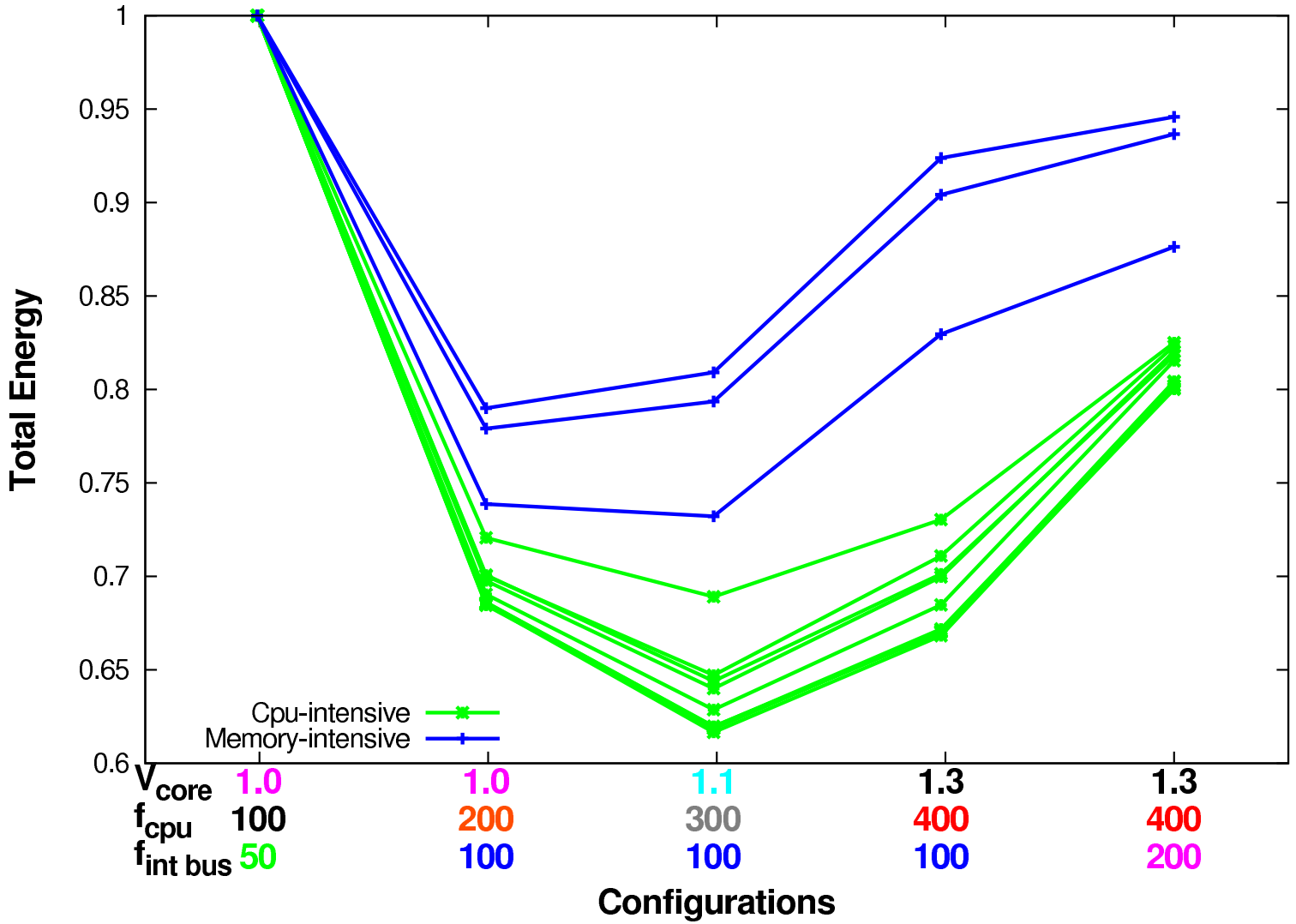
Padding with idle power to equal total time



Padding with idle power to equal total time



4. Normalised total energy padded to equal time



Memory Power Estimation

What if we cannot measure memory power?

Memory Power Estimation

What if we cannot measure memory power?

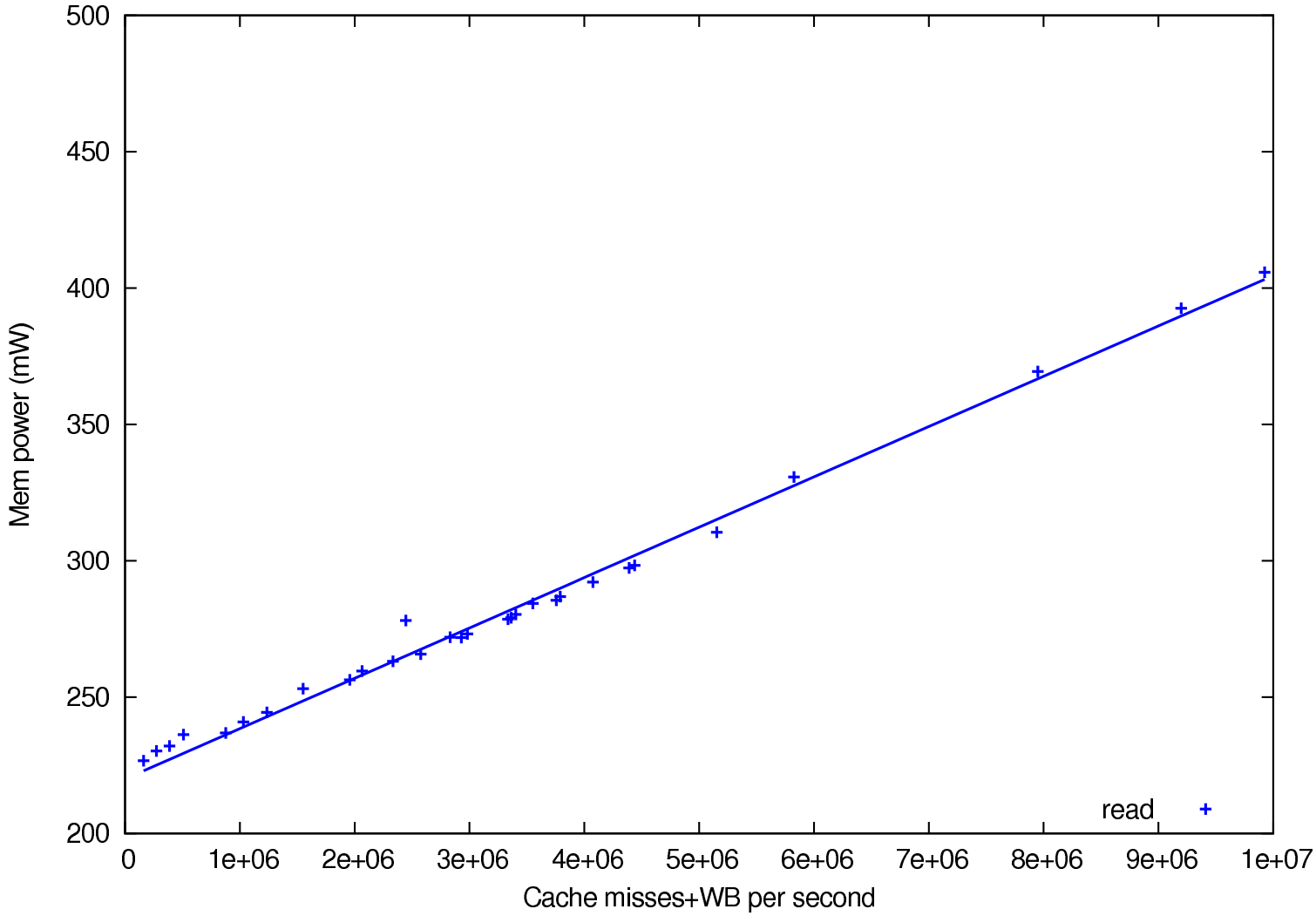
- Model memory power via performance counters.
- Counting memory accesses leads to good results (WB02)
- However, XScale doesn't have enough performance counters

Memory Power Estimation

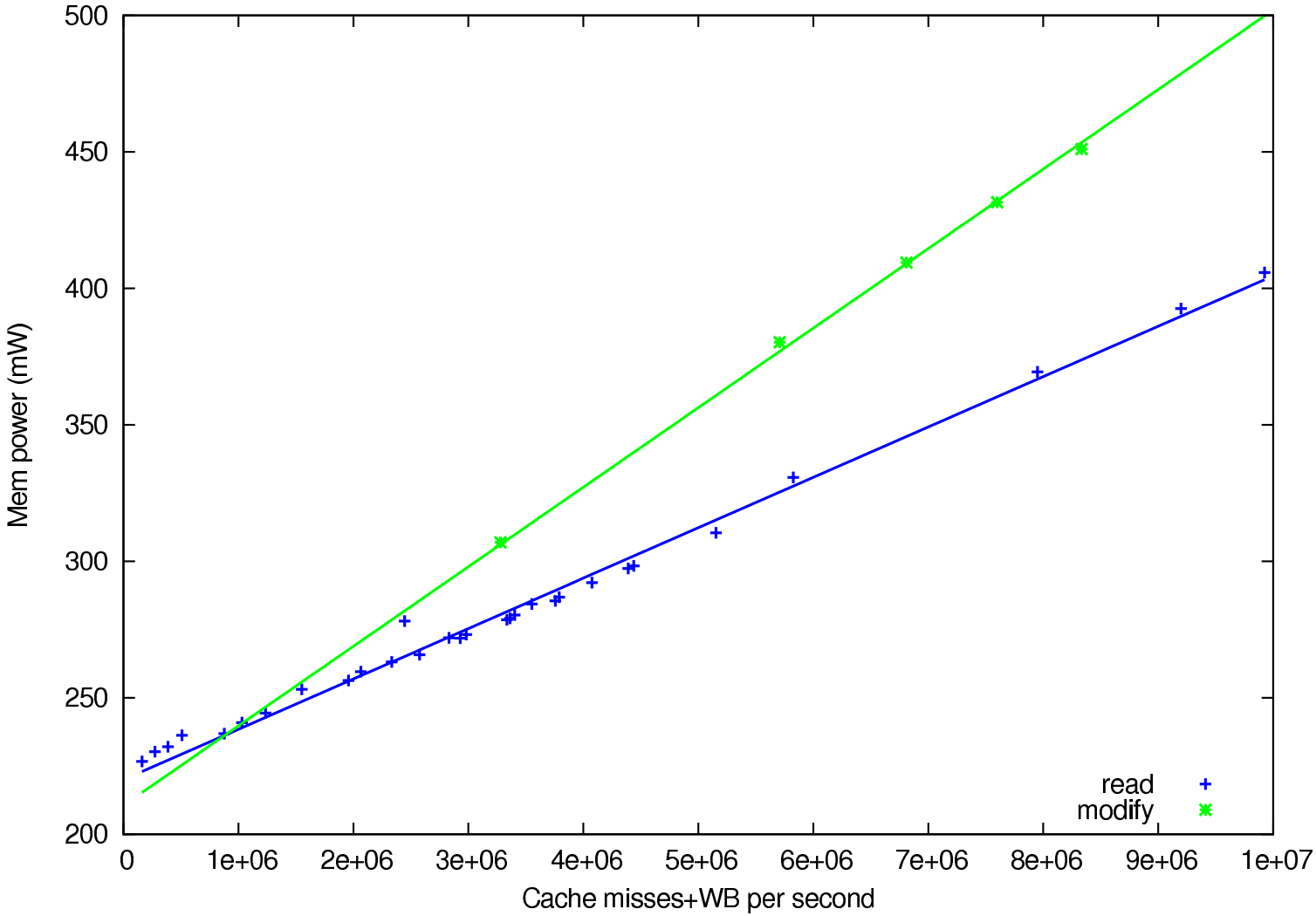
What if we cannot measure memory power?

- Model memory power via performance counters.
- Counting memory accesses leads to good results (WB02)
- However, XScale doesn't have enough performance counters
- Idea: count cache misses instead?
- Investigated via two micro-benchmarks
  - *read*: only load instructions
  - *modify*: load and then store to same location

5. Memory power vs cache miss rate



5. Memory power vs cache miss rate



What can we learn from this?

- The simple assumptions behind the common DVFS approach are *wrong*!

What can we learn from this?

- The simple assumptions behind the common DVFS approach are *wrong*!
  - ➜ memory stalls
  - ➜ static (leakage) power
  - ➜ memory power (and I/O devices)
  - ➜ multiple frequency/voltage domains

What can we learn from this?

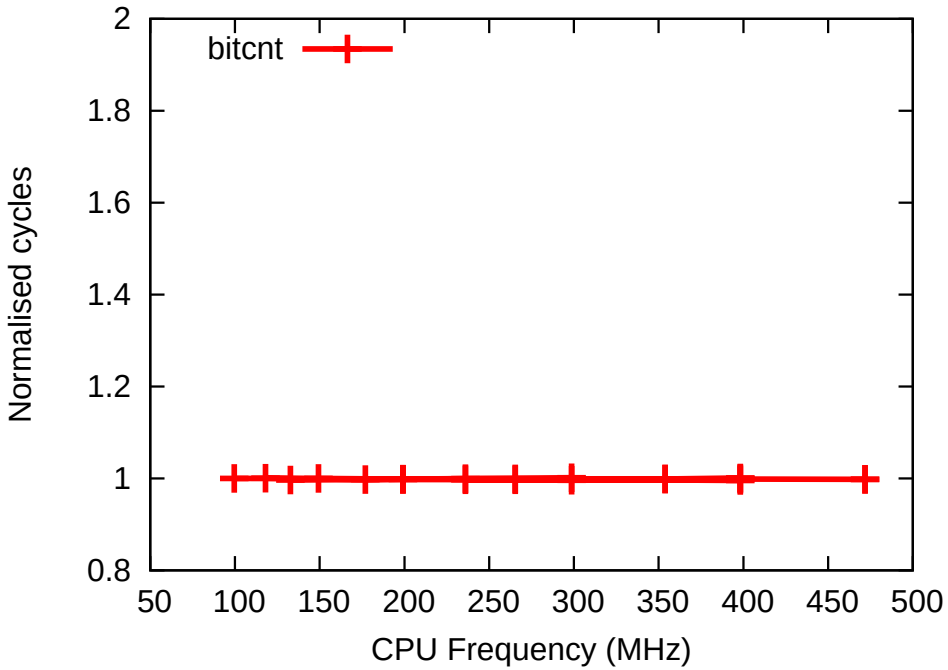
- The simple assumptions behind the common DVFS approach are *wrong*!
  - ➜ memory stalls
  - ➜ static (leakage) power
  - ➜ memory power (and I/O devices)
  - ➜ multiple frequency/voltage domains
- Things aren't as simple as some people think...
  - ➜ yet this model is used in many publications!

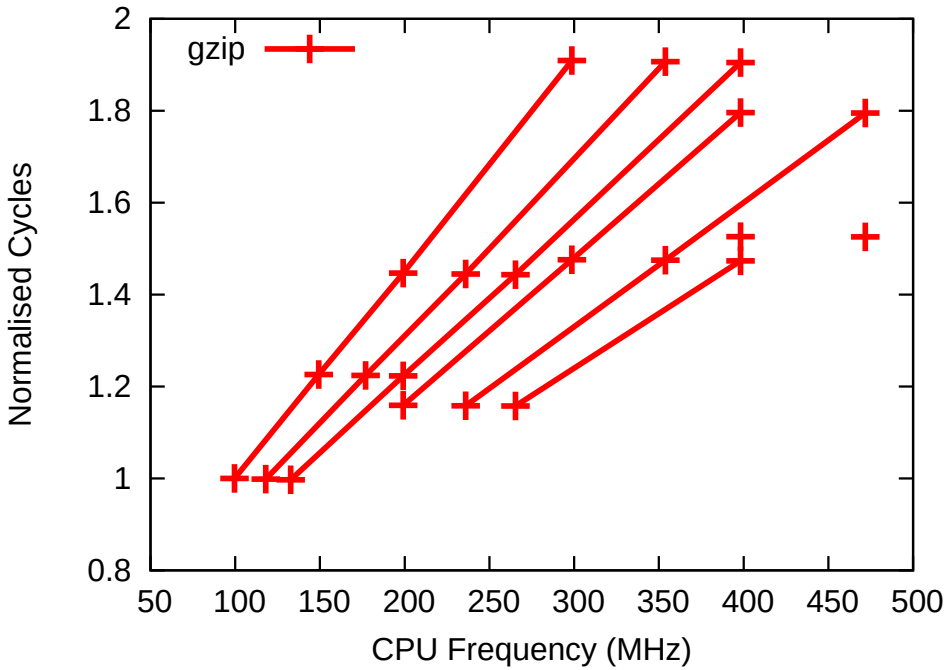
Memory stalls

- Total time of memory access is independent of core frequency
- Cycles for memory access are *proportional* to core frequency

Memory stalls

- Total time of memory access is independent of core frequency
- Cycles for memory access are *proportional* to core frequency



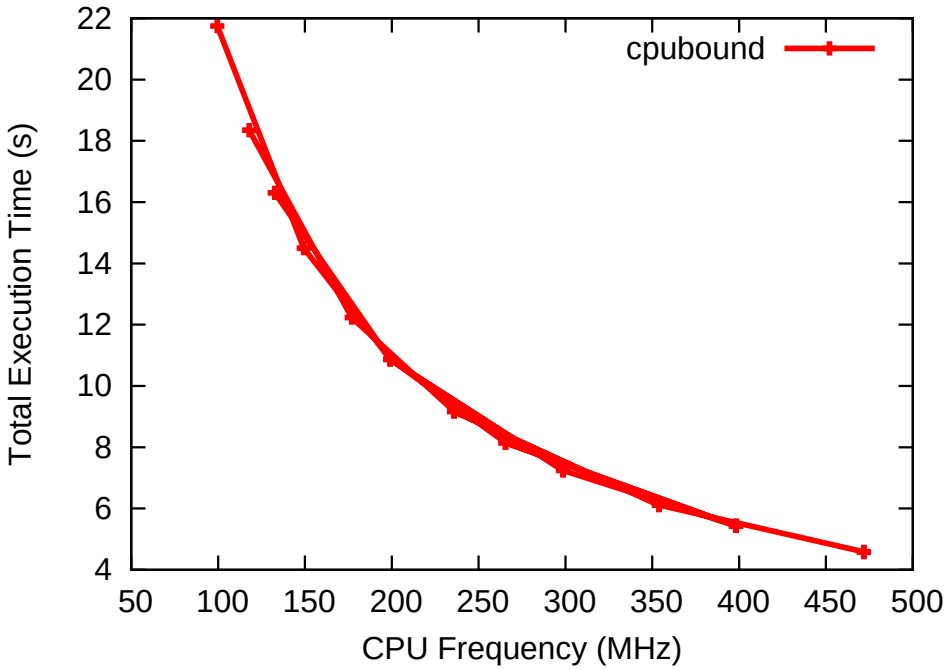


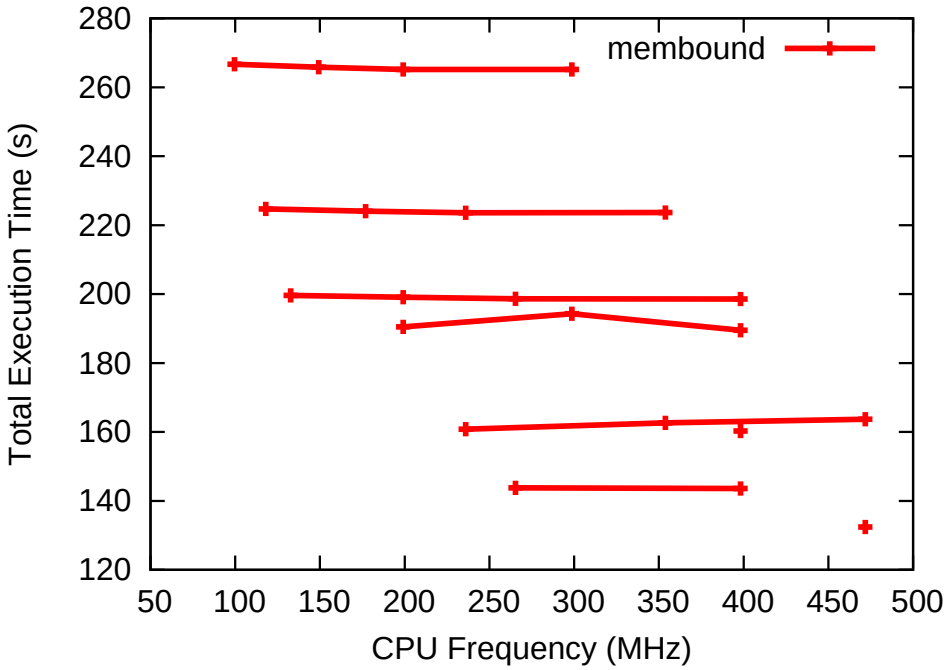
Memory stalls

- Hence, with increasing core frequency:
  - ➜ time for CPU instructions drops
  - ➜ time for memory accesses is unchanged

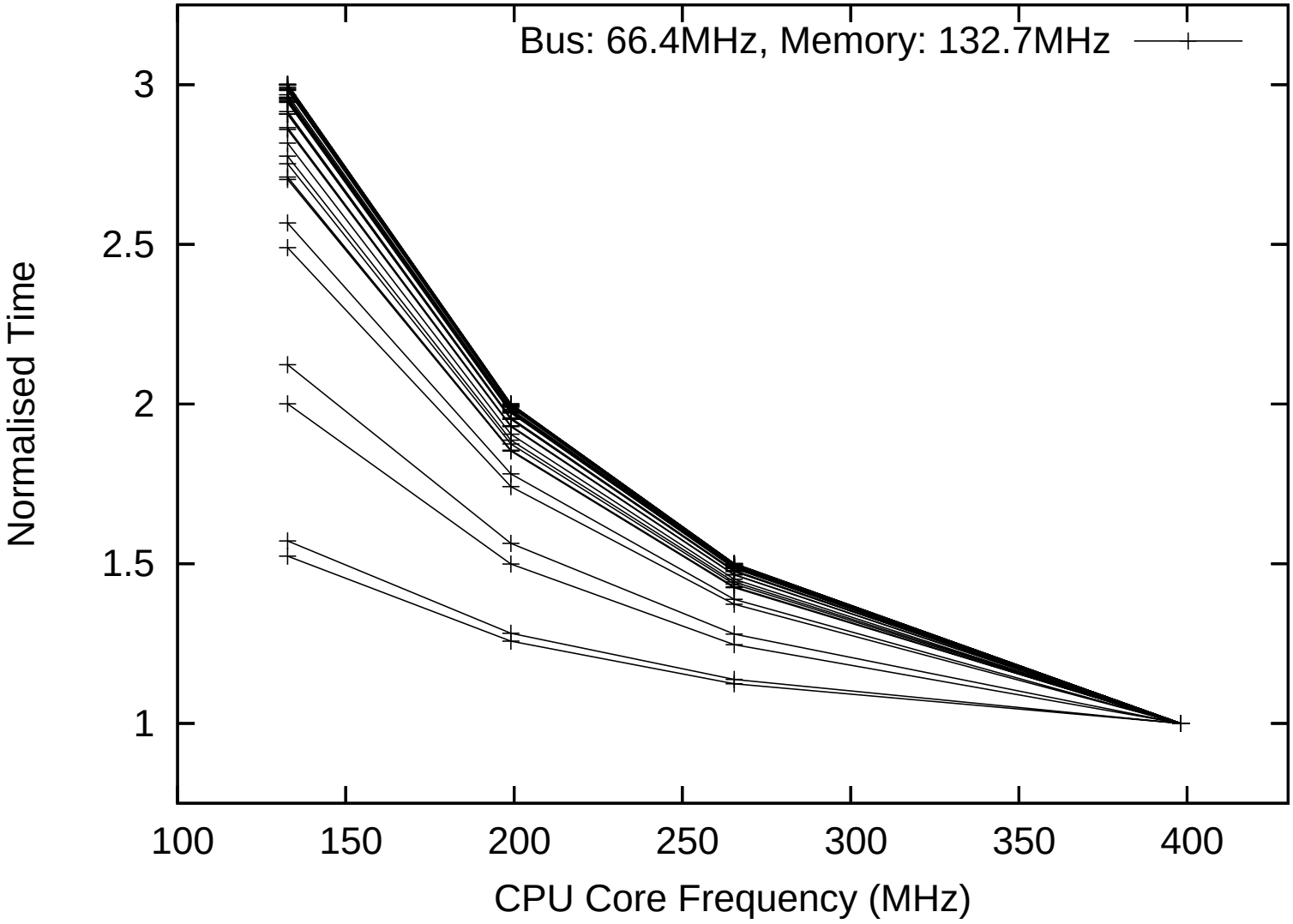
Memory stalls

- Hence, with increasing core frequency:
  - ➜ time for CPU instructions drops
  - ➜ time for memory accesses is unchanged





Real Program Behaviour

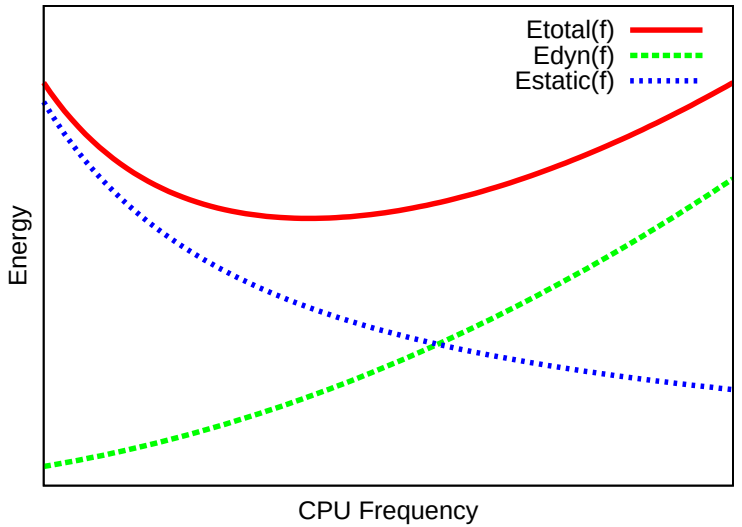


Static power

- DRAM consumes power also when **not** switching
  ➜ in modern processors static power is starting to dominate!
- Static *power* is independent of frequency
  ➜ static *energy* is proportional to execution time!

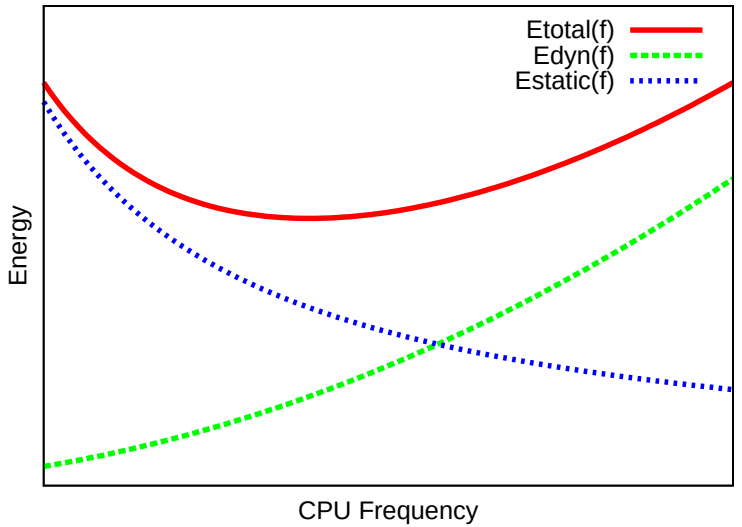
STATIC POWER

- DRAM consumes power also when **not** switching
  - ➜ in modern processors static power is starting to dominate!
- Static *power* is independent of frequency
  - ➜ static *energy* is proportional to execution time!
- Hence, with increasing core frequency:
  - ➜ dynamic energy decreases
  - ➜ static energy increases



Static power

- DRAM consumes power also when **not** switching
  - ➜ in modern processors static power is starting to dominate!
- Static *power* is independent of frequency
  - ➜ static *energy* is proportional to execution time!
- Hence, with increasing core frequency:
  - ➜ dynamic energy decreases
  - ➜ static energy increases
  - ➜ optimal frequency depends on relative size of static and dynamic power
  - ➜ lowest energy may be at intermediate frequency!



What is Needed for Energy Management?

- Effect of DVFS on energy consumption depends on workload characteristics
  - ➜ need individual settings for each application
  - ➜ requires *a-priori* characterisation or run-time adjustment

What is Needed for Energy Management?

- Effect of DVFS on energy consumption depends on workload characteristics
  - ➜ need individual settings for each application
  - ➜ requires *a-priori* characterisation or run-time adjustment
- Minimising energy with DVFS requires:
  - ➜ prediction of *performance* (execution time) under DVFS
  - ➜ prediction of *power consumption* under DVFS

What is Needed for Energy Management?

- Effect of DVFS on energy consumption depends on workload characteristics
  - ➜ need individual settings for each application
  - ➜ requires *a-priori* characterisation or run-time adjustment
- Minimising energy with DVFS requires:
  - ➜ prediction of *performance* (execution time) under DVFS
  - ➜ prediction of *power consumption* under DVFS
- Needs to happen in real time if *a-priori* characterisation is infeasible
  - ➜ can use performance counters (WB02)
  - ➜ needs to be done with low overhead

Realistic Execution Time Model

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \cdots$$

- C_x represents the number of clock cycles for f_x.
- C_x is workload-specific, independent of frequency.
- Assumes no dependences — works well in practice.

Realistic Execution Time Model

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \ldots$$

- C_x represents the number of clock cycles for f_x.
- C_x is workload-specific, independent of frequency.
- Assumes no dependences — works well in practice.

C_x: characterise online using performance counters

$$\begin{aligned} C_{bus} &= \alpha_1 PMC_1 + \alpha_2 PMC_2 + \ldots \\ C_{mem} &= \beta_1 PMC_1 + \beta_2 PMC_2 + \ldots \end{aligned}$$

(C_{cpu} inferred from the other results)

Realistic Execution Time Model

Estimating C_{cpu}:

➜ C_{cpu} is difficult to estimate using performance counters alone

Realistic Execution Time Model

Estimating C_{cpu}:

- ➜ C_{cpu} is difficult to estimate using performance counters alone
- ➜ But...

$$C_{cpu} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} + \cdots$$

- ➜ If we can measure cycles, we can estimate C_{cpu}.
- ➜ Or... If we can estimate the off-chip cycles, we can calculate the on-chip cycles.

Evaluation

The data:

- ➜ Typical embedded platform (PLEB 2, XScale based)
- ➜ Cycle counter, 2 performance counters, 13 events
- ➜ 22 frequency setpoints with different f_{cpu}, f_{bus} and f_{mem}
- ➜ 37 benchmarks run to completion at each setpoint for all frequency settings
- ➜ All performance counter events were measured end-to-end for each benchmark/frequency
- ➜ Benchmarks were partitioned for calibration and validation
- ➜ Measurements: Cycles, Frequencies, Performance counters

Evaluation

Which counters are the best to use?

Per workload:

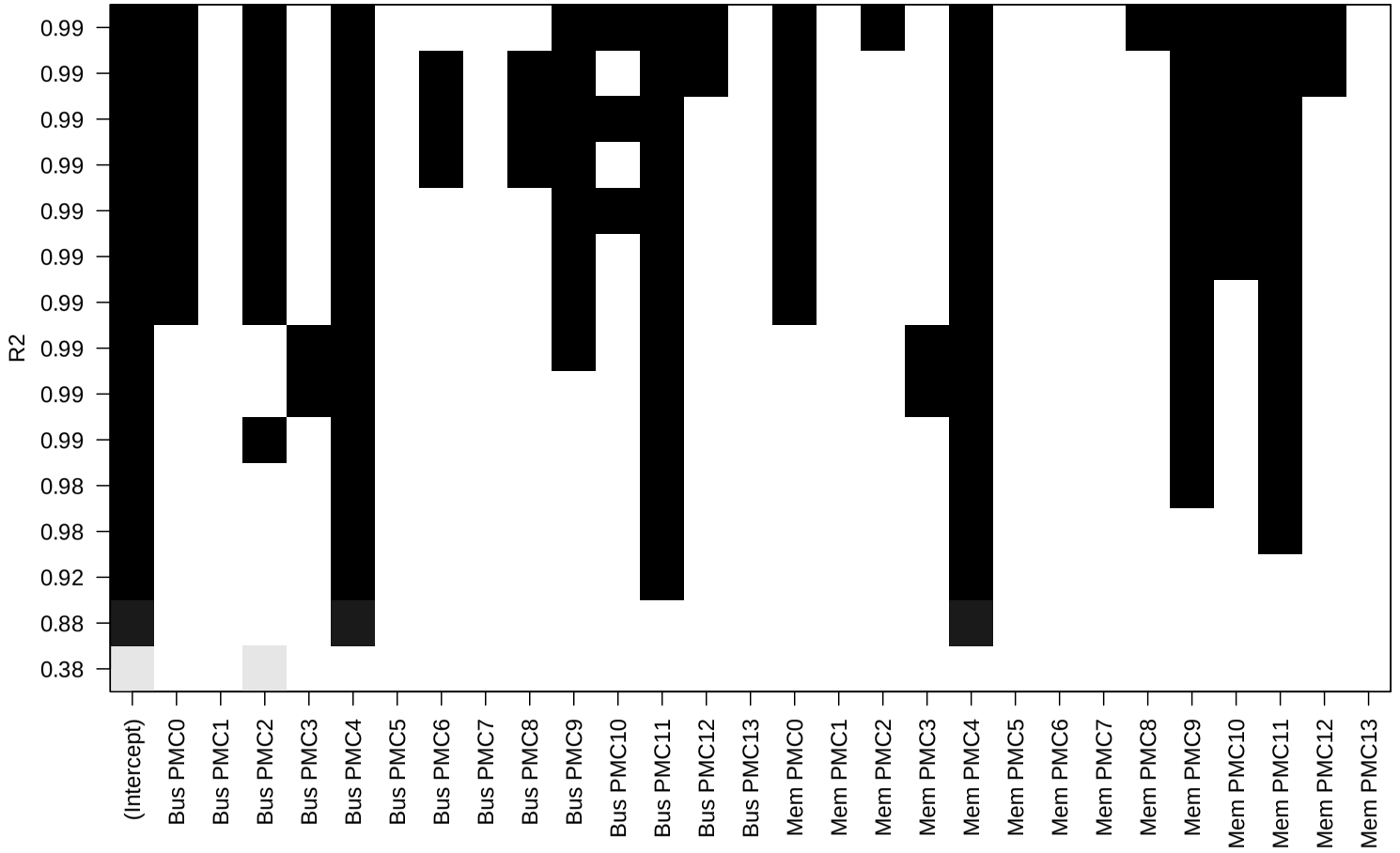
$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \ldots$$

Per platform:

$$C_{bus} = \alpha_1 PMC_1 + \alpha_2 PMC_2 + \ldots$$
$$C_{mem} = \beta_1 PMC_1 + \beta_2 PMC_2 + \ldots$$

- ➜ Fit for each benchmark for the average PMC_x
- ➜ Exhaustive parameter selection based on bic or R^2.

Picking Performance Counters



➜ Best correlation of power and PMC readings for different number of counters

USING THE MODEL

Making predictions:

How long would it have taken to run at XX MHz?

$$C'_{tot} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} + \frac{f'_{cpu}}{f'_{bus}} C_{bus} + \frac{f'_{cpu}}{f'_{mem}} C_{mem}$$

Errors:

- ➜ Predicting the performance at the maximum frequency
- ➜ 2-parameter: avg 1.7%, max 7%
- ➜ CPU frequency only: avg 10%, max 36%

USING THE MODEL

Relative performance:

➜ Often we want a normalised performance (e.g. 50% of maximum).

$$s = \frac{t^{max}}{t'} = \frac{f'_{cpu}}{f^{max}_{cpu}} \times \frac{C^{max}_{tot}}{C'_{tot}}$$

This is a linear equation in terms of the PMCs

Power/Energy model

Power/Energy model basics:

For our system:

Power/Energy model

Power/Energy model basics:

- ➜ *Events* each use an amount of energy
- ➜ An event may use energy in more than one voltage domain

For our system:

$$E_{events} = V_{cpu}^2(\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \beta_0 PMC_0 + \cdots + \beta_m PMC_m$$

Power/Energy model

Power/Energy model basics:

- ➜ *Events* each use an amount of energy
- ➜ An event may use energy in more than one voltage domain
- ➜ Clock cycles count as events

For our system:

$$E_{events} = V_{cpu}^2(\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \beta_0 PMC_0 + \cdots + \beta_m PMC_m$$

$$E_{freqs} = V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + (\gamma_4 f_{cpu} + \gamma_5 f_{bus} + \gamma_6 f_{mem})\Delta t$$

Power/Energy model

Power/Energy model basics:

- ➜ *Events* each use an amount of energy
- ➜ An event may use energy in more than one voltage domain
- ➜ Clock cycles count as events
- ➜ Static power models power not related to events or voltages.
- ➜ Constant IO power for the benchmarks tested.

For our system:

$$E_{events} = V_{cpu}^2(\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \beta_0 PMC_0 + \cdots + \beta_m PMC_m$$

$$E_{freqs} = V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + (\gamma_4 f_{cpu} + \gamma_5 f_{bus} + \gamma_6 f_{mem})\Delta t$$

$$E_{static} = P_{static}\Delta t$$

Power/Energy model

Power/Energy model basics:

- ➜ *Events* each use an amount of energy
- ➜ An event may use energy in more than one voltage domain
- ➜ Clock cycles count as events
- ➜ Static power models power not related to events or voltages.
- ➜ Constant IO power for the benchmarks tested.

For our system:

$$E_{events} = V_{cpu}^2(\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \beta_0 PMC_0 + \cdots + \beta_m PMC_m$$

$$E_{freqs} = V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + (\gamma_4 f_{cpu} + \gamma_5 f_{bus} + \gamma_6 f_{mem})\Delta t$$

$$E_{static} = P_{static}\Delta t$$

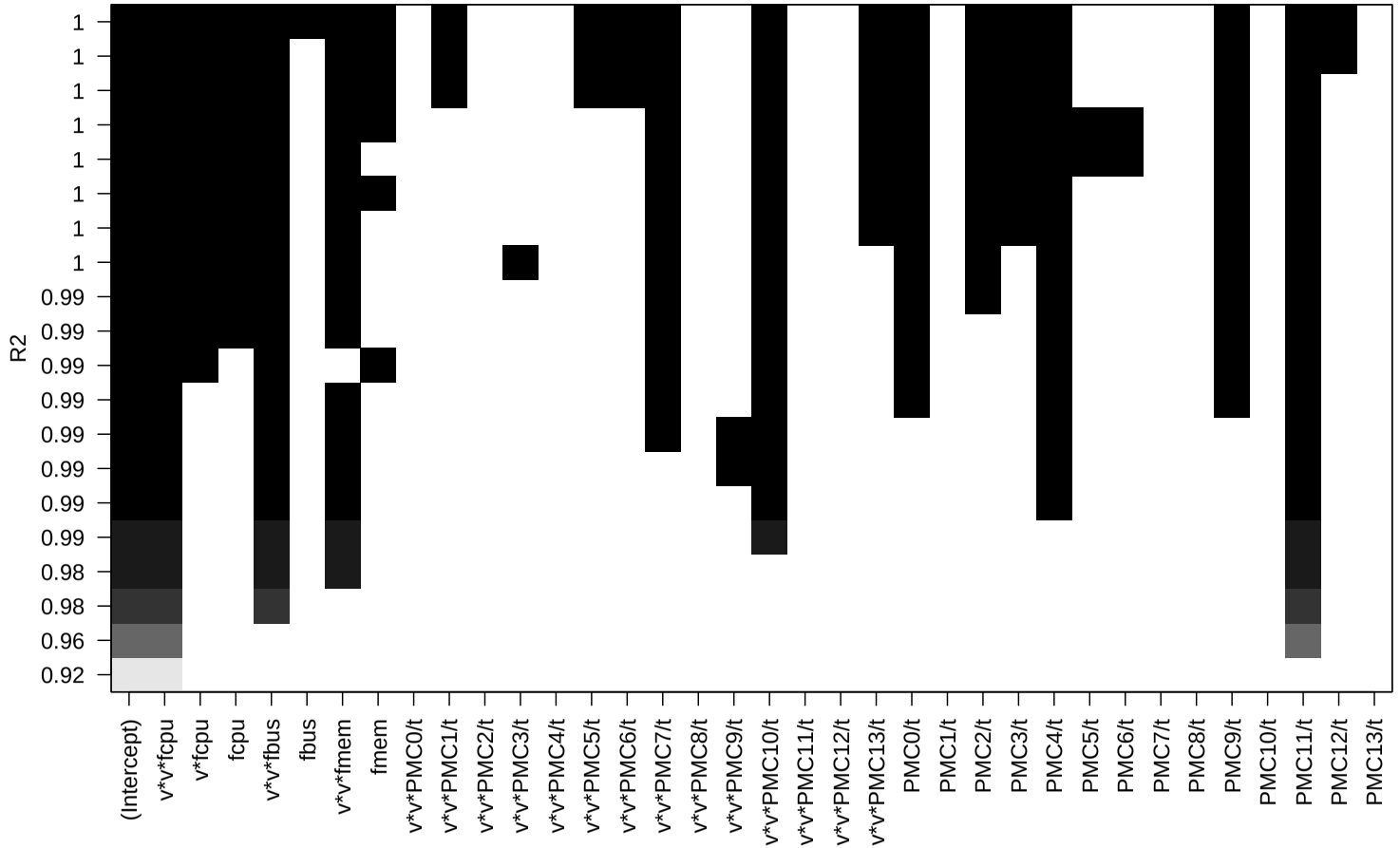
Time is essential for either power or energy estimation

Power/Energy model

The Data:

- Same as ETM data
- End-to-end energy measured using a shunt resistor
- Measurements triggered using a GPIO
- Each workload sample is used to predict the power for every other sample
- Frequency and voltage are varied independently

Power/Energy model



Power/Energy model

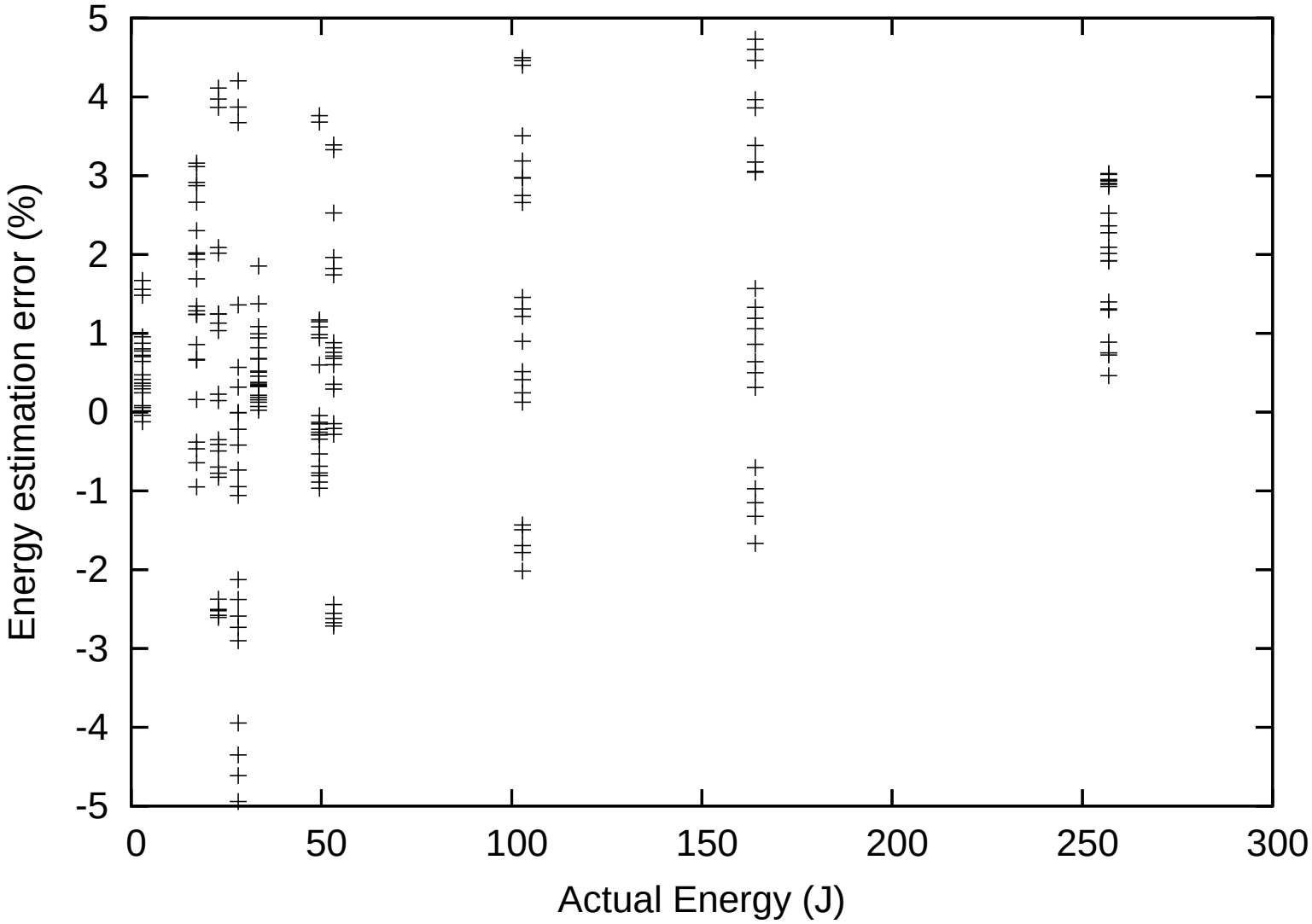
Power error data (using a measured run-time):

Counters	Param.	R^2	Max Err (%)	Avg Err (%)
1	4	0.9836	7.46	2.14
2	5	0.9871	6.94	2.31
3	6	0.9904	4.85	1.26
4	7	0.9922	3.78	1.16
5	8	0.993	3.68	0.92
6	9	0.9938	2.94	0.89
6	11	0.9947	2.75	0.77

Energy error data (estimated run-time — four counters):

➜ Max: 4.9%, Avg: 1.5%

Power/Energy model



Using the models

Sound approach:

- Determine appropriate PMC set and weights via systematic process
  - once per platform
- Use these to predict performance and power for actual load at run time
- Context-switch DVFS settings

Using the models

Sound approach:

- Determine appropriate PMC set and weights via systematic process
  - once per platform
- Use these to predict performance and power for actual load at run time
- Context-switch DVFS settings

Using these results:

- Thread-level DVFS.
- Identify appropriate frequency choices
- $\boldsymbol{Energy \cdot Delay}$
- Sleep mode vs. DVFS trade-offs

USING THE MODELS

Performance predictions at run-time:

- A toy system: frequency scaling in Linux
- Measures the performance counters after each time slice
- Predicts the performance at all of the other setpoints for that time-slice
- Assuming temporal locality, chooses the setpoint with the closest to a chosen performance
- Error: avg 1.9%, max 7%

Not useful in itself, but demonstrates the model