Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data

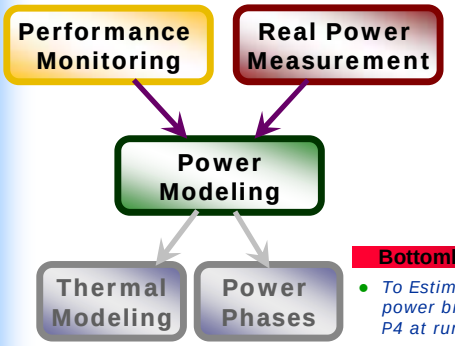
Canturk Isci & Margaret Martonosi
Princeton University

MICRO-36
12.03.2003
San Diego, CA

Motivation

- Power Matters!
- Need good Measurement/Modeling techniques for Power & Thermally aware/adaptive systems
- Need for Fast-Realtime Modeling and Measurement to observe long time periods
- Need live, run-time power/thermal measures

THE BIG PICTURE



Bottomline…

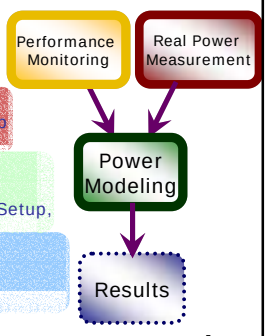
- *To Estimate component power breakdowns for P4 at runtime…*

Questions We Answer

- What kinds of techniques can we use beyond simulation in power research?
  - Power Measurements without interfering with the hardware
  - Power Estimation based on performance counters
- How does power behavior of programs change over their whole runtime?
  - (Power Phases further extend this analysis)
- How well such an estimation framework can perform in comparison to actual measurements
  - We show with several benchmarks, using our synchronized measurement and estimation setup

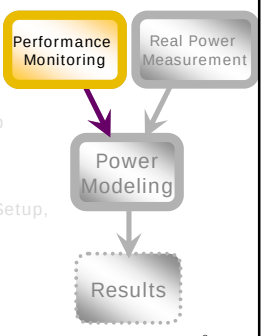
Remainder of Talk

- Performance Monitoring
  - P4 Performance Counters
  - Performance Reader LKM
- Real Power Measurement
  - P4 Power Measurement Setup
  - Examples
- Power Modeling
  - P4 Power Model
  - Model + Measurement Sync Setup, Verification
- Results
  - SPEC CPU2000
  - Desktop Applications



Performance Monitoring

- Performance Monitoring
  - P4 Performance Counters
  - Performance Reader LKM
- Real Power Measurement
  - P4 Power Measurement Setup
  - Examples
- Power Modeling
  - P4 Power Model
  - Model + Measurement Sync Setup, Verification
- Results
  - SPEC CPU2000
  - Desktop Applications



Live CPU Performance Monitoring with Hardware Counters

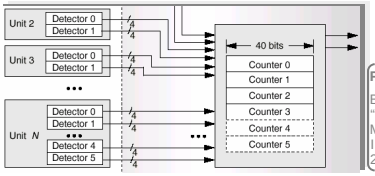
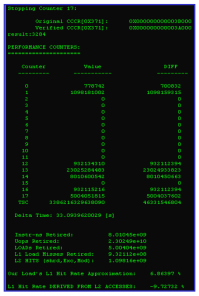


Figure from:
Brinkley Sprunt, "Pentium 4 Performance Monitoring Features", IEEE Micro, Jul-Aug 2002, pp. 72-82.

- Most CPUs have hardware performance counters
- P4 Performance Monitoring HW:
  - 18 Event Counters
  - 18 Counter Configuration Control Registers
    - Configure how to count
  - 45 Event Selection Control Registers
    - Configure what to count
  - Additional Control Registers

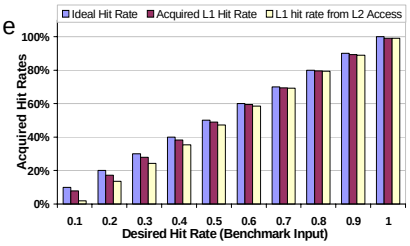
Our Event Counter: Performance Reader

- Performance Reader implemented as Linux Loadable Kernel Module
  - Implements 6 syscalls:
    - *select_events()*
    - *reset_event_counter()*
    - *start_event_counter()*
    - *stop_event_counter()*
    - *get_event_counts()*
    - *set_replay_MSRs()*
- User Level Interface:
  - Defines the events → Starts counters
  - Stops counters → Reads counters & TSC



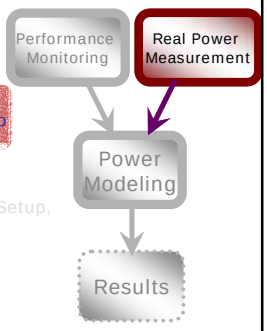
Performance Reader: Example Validation

- L1_Dcache benchmark
- Controls cache hit behavior
- Validated against measured cache events
- Vary hit rate from 0-100%



Processor Power Measurement

- Performance Monitoring
  - P4 Performance Counters
  - Performance Reader LKM
- Real Power Measurement
  - P4 Power Measurement Setup
  - Examples
- Power Modeling
  - P4 Power Model
  - Model + Measurement Sync Setup, Verification
- Results
  - SPEC CPU2000
  - Desktop Applications

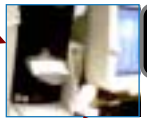


P4 Power Measurement Setup

Clamp ammeter on 12V lines on measured CPU

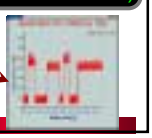
DMM reading clamp voltages

1mV/Adc conversion



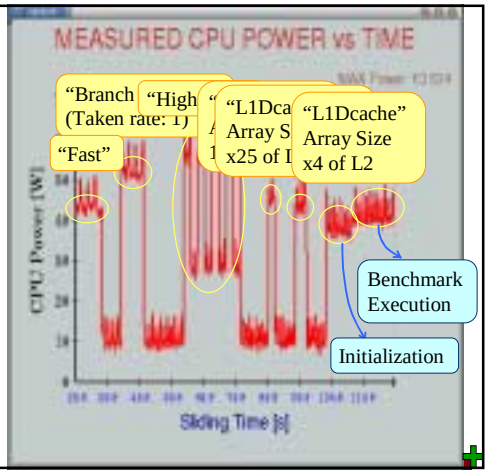
Serial Reader (PowerMeter) (PowerPlotter)

Voltage readings via RS232 to logging machine



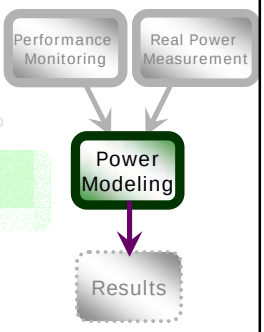
Convert to Power vs. time window

PowerPlotter: Example



Processor Power Modeling

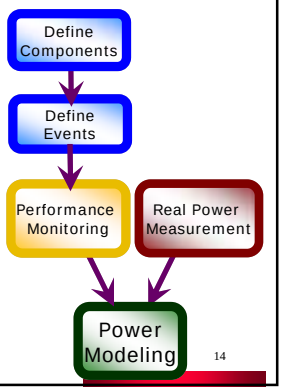
- Performance Monitoring
  - P4 Performance Counters
  - Performance Reader LKM
- Real Power Measurement
  - P4 Power Measurement Setup
  - Examples
- Power Modeling
  - P4 Power Model
  - Estimation + Measurement Sync. Setup, Verification
- Results
  - SPEC CPU2000
  - Desktop Applications



13

P4 POWER MODEL

- Define components (I.e. L1 cache, BPU, Regs, etc.), whose powers we'll model:
  - from annotated layout
- Determine combination of P4 events that represent component accesses best
- Gather counter info with minimal power overhead and program interruption
- Convert counter based access rates into component power breakdowns
- Verify total power against measured processor power



14

Defining Events → Access Rates

- We determined 24 events to approximate access rates for 22 components
- Used Several Heuristics to represent each access rate
- Ex: 2nd Level BPU:
  - Metric 1: Instructions fetched from L2 (predict)
    - Event: ITLB_Reference
      - Counts ITLB translations
    - Mask:
      - All hits
    - Expression: 8 x ITLB_Reference
      - Minimum 8 instructions per L2 line (128B / 16B)
  - Metric 2: Branches retired (history update)
    - Event: branch_retired
      - Counts branches retired
    - Mask:
      - Count all Taken/NT/Predicted/Mispredicted
- Need to rotate counters 4 times to collect all event data
  - Used 15 counters & 4 rotations to collect all event data

15

Access Rates → Component Powers

- Governing relation:

$$Power(C_i) = AccessRate(C_i) \cdot ArchitecturalScaling(C_i) \cdot MaxPower(C_i) + NonGatedClockPower(C_i)$$

From Performance Counters | From Microarchitectural Properties | Initially, area based estimates; Later modified by tuning benchmarks | Estimated by the large power jump during idle → low utilization (upc)

16

Access Rates → Component Powers

- Governing relation:

$$Power(C_i) = AccessRate(C_i) \cdot ArchitecturalScaling(C_i) \cdot MaxPower(C_i) + NonGatedClockPower(C_i)$$

- EX: Trace cache delivers 3 uops/cycle in deliver mode and 1 uop/cycle in build mode:

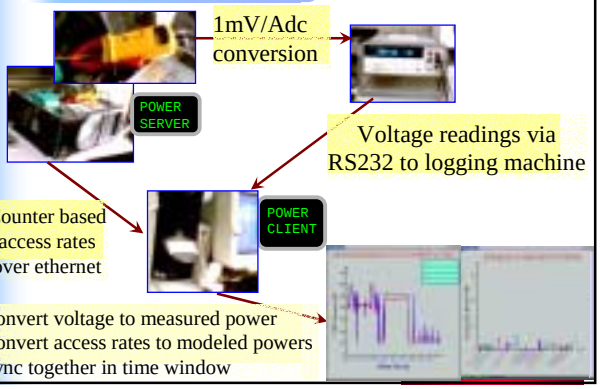
$$Power(TC) = \left[\frac{AccessRate(TC)}{3} + AccessRate(ID)\right] \bullet MaxPower(TC) + ClkPower(TC)$$

- Total power is computed as the sum of all 22 component powers + measured idle power (8W):

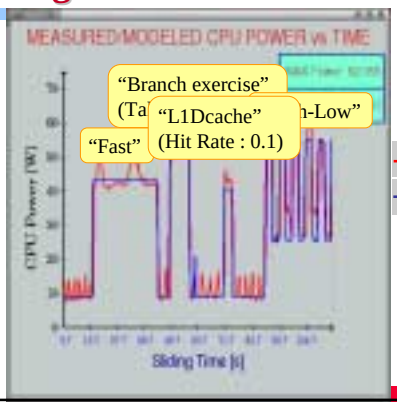
$$Total\ Power = \sum_{i=1}^{22} Power(C_i) + Idle\ Power$$

17

Experiment Setup



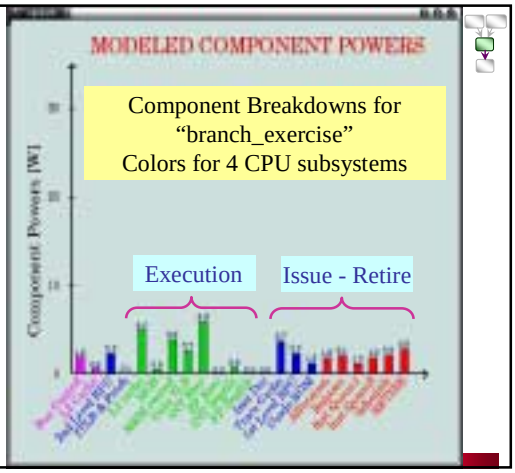
Tuning Benchmarks



Counter-based Power Estimation: Validation Step 2

Adjusting max power coeff-s on stressmarks

Component Breakdowns



Complete Example: Retirement Logic

- Retirement Logic defined from annotated die layout
- Access rate approximation based on performance counters: $\frac{UopsRetired}{\Delta Cycles}$
- Power relation for retirement logic:
  - Can retire at most 3 uops/cycle

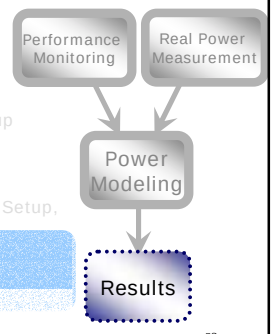
$$Power(Ret) = \left[\frac{AccessRate(Ret)}{3}\right] \bullet MaxPower(Ret) + ClkPower(Ret)$$

- Initial area based Max power estimation:
  - MaxPower = Area% x Max Processor Power ➔
  - MaxPower = 6.5% x 72W = 4.7W
- Max power & Clk power estimations after tuning:
  - MaxPower = 1.5W | ClkPower = 2W
- Final hardcoded power equation for retirement logic:

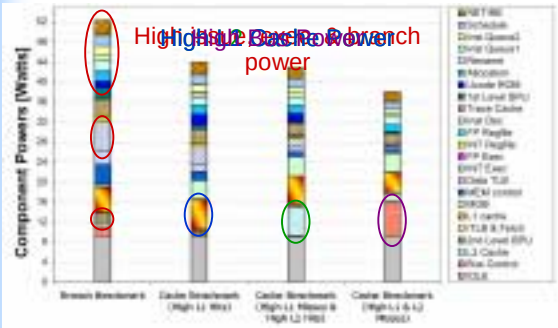
$$Power(Ret) = \frac{AccessRate(Ret)}{(3)} \bullet [0.5 \cdot (3)] + 2.0$$

Power Estimation Results

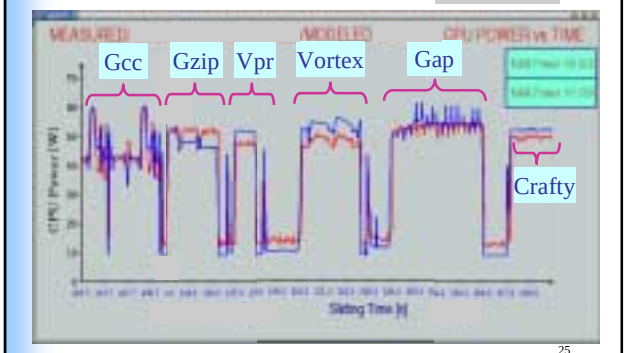
- Performance Monitoring
  - P4 Performance Counters
  - Performance Reader LKM
- Real Power Measurement
  - P4 Power Measurement Setup
  - Examples
- Power Modeling
  - P4 Power Model
  - Model + Measurement Sync Setup, Verification
- Results
  - SPEC CPU2000
  - Desktop Applications



Validation for Fidelity: Benchmark Power Breakdowns

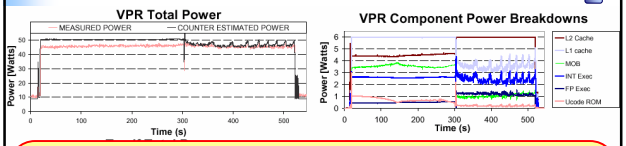


Validation for Accuracy: SPEC Results



25

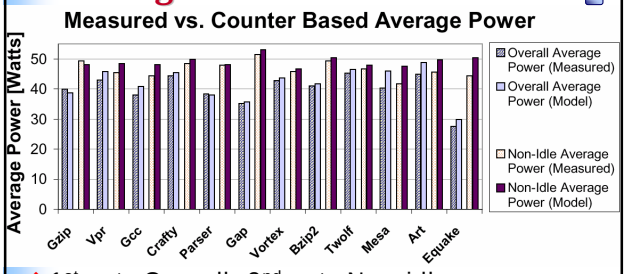
SPEC2000 Results



Equake Elaboration: (FP benchmark**)**
Initialization and computation phases
FP intensive mesh computation phase
Initialization with high complex IA32 instructions

Twolf Elaboration:(Integer benchmark**)**
Several loop computations traversing memory
<High Memory Power>
Although ~const. Total power, component powers have slight gradients

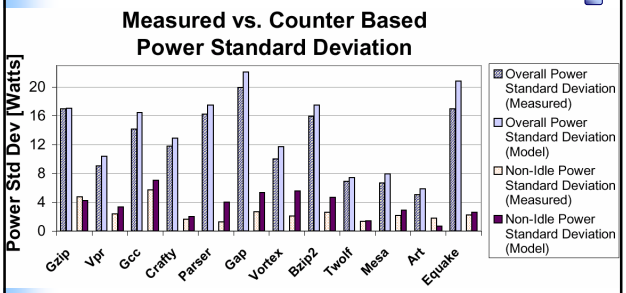
Average SPEC Total Powers



- 1st set: Overall, 2nd set: Non-idle power
- Average difference between measurement and estimation: 3W
- Worst case: Equake (5.8W)

27

Stdev of SPEC Total Powers

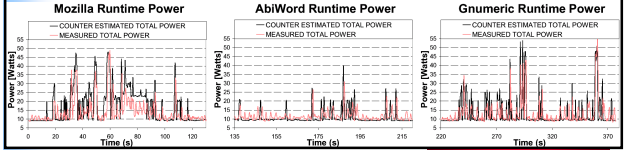


- 1st set: Overall, 2nd set: Non-idle power
- Average difference: 2W
- Worst case: Vortex (3.5W)

28

Desktop Applications

- We aim to track low power utilizations as well.
- Desktop applications are usually low power with intermittent power bursts
- 3 applications, with common operations such as open/close application, web, streaming media, text editing, save to disk, statistical computations.



Conclusions

- Contributions:
  - Portable runtime real power measurement system
  - Performance counter based runtime power model and runtime verification with synchronous real power measurement for arbitrarily long timescales!
  - Physical component based power estimates for processor, which can be used in power phase analyses and thermal modeling
- Outcomes:
  - We can do reasonably accurate real power measurements at runtime without interfering with HW
  - We can perform runtime power modeling, with the tiny performance reader without inducing any significant overhead to power profile
  - Component power breakdowns can be used to identify program power phases

30

Related Work

- **Implementing counter readers:**
  - PCL [Berrendorf 1998], Intel VTune, Brink & Abyss [Sprunt 2002]
- **Using counters for Power:**
  - CASTLE [Joseph 2001], power profilers
  - event driven OS/cruise control [Bellosa 2000,2002]
- **Real Power Measurement:**
  - Compiler Optimizations [Seng 2003]
  - Cycle-accurate measurement with switch caps [Chang 2002]
- **Power Management and Modeling Support:**
  - Instruction level energy [Tiwari 1994]
  - PowerScope: Procedure level energy [Flinn 1999]
  - Event counter driven energy coprocessor [Haid 2003]
  - Virtual Energy Counters for Mem. [Kadayif 2001]
  - ECOsystem: OS energy accounting [Ellis 2002]

Our Work in Comparison

- Power estimation for a complex, aggressively clock-gated processor
- Component power estimates with physical binding to die layout
  - Laying the groundwork for thermal modeling
- Portable implementation with current probe and power server LKM
- Power oriented phase analysis with acquired power vectors

EOP

Support/Detail Slides

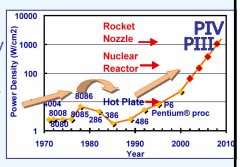
- **FOLLOWING SLIDES INCLUDE MORE DETAILS OR SUPPORT FOR THE 4 PARTS OF THE TALK, I HAVE THEM HERE FOR COMPLETENESS AND IF SOMEONE WONDERS STH THAT I HAVE THE ANSWERS HERE**

DETAILS for MOTIVATION

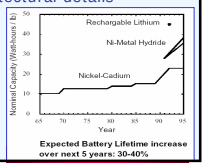
- Following (blue) slides are the details of the motivation slide I have

MOTIVATION

- Power Matters!
  - Performance improves exponentially → **SO DOES POWER DENSITY**
    - Chip areas increase 7%/year
  - Battery Life: Improves Much Slower
  - Thermal Issues
    - Follows power density
    - Packaging costs: +$1/W over ~40W



- Need good Measurement/Modeling techniques for Power & Thermally aware/adaptive systems
  - Using Measurement to probe microarchitectural details
    - CASTLE, data activity experiment
  - Compiler Level Power Optimizations
    - SW Power Profiling and Optimization
  - Power aware OS
    - power modeling for decision making
  - Dynamic thermal/power management
    - Thermal hotspots & Power threshold



MOTIVATION

- Power Models reflecting modern processors
  - Clock gating, power
  - Voltage regulation, di/dt
- Need for Fast-Realtime Modeling and Measurement to observe long time periods
  - Thermal time constants: O(s)
  - Not feasible even with architecural simulators
    - i.e.: 1s of real run ⇔ ~5 x IPC hrs of WATTCH simulation
- Need live, run-time power/thermal measures
  - Dynamic Thermal Management
  - Power-Aware OS & Systems control

Motivation

- Battery technology increases much slower
- Packaging costs: +$1/W over 35-40W [2]

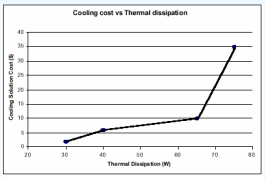
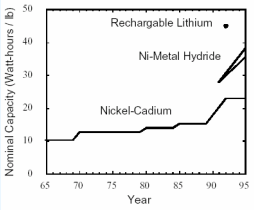


Figure 2: The cost of removing heat from a microprocessor



Expected Battery Lifetime increase over next 5 years: 30-40%

Back to slide

Details for Performance Counters

- These Slides are support and more detail slides related to:
  - performance counters
  - our LKM performance reader
  - Our L1Dcache and branch exercise benchmarks

P4 Detector - Counter Clusters

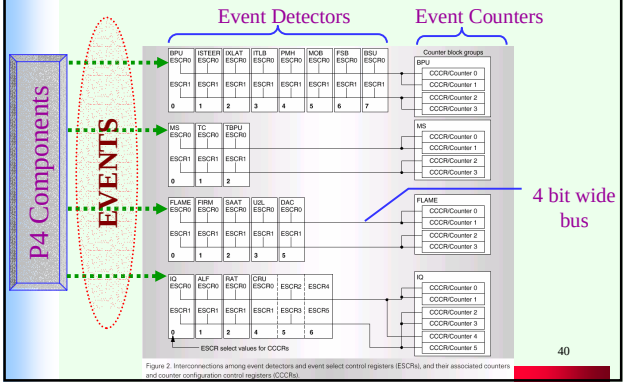
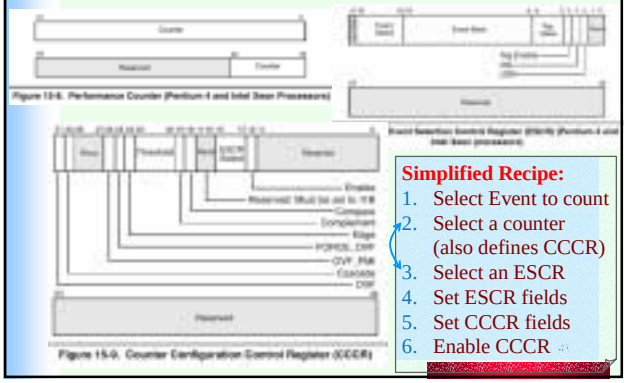


Figure 2. Interconnections among event detectors and event select control registers (ESCRs), and their associated counters and counter configuration control registers (CCCRs).

Counters, ESCRs & CCCRs



Simplified Recipe:

1. Select Event to count
2. Select a counter (also defines CCCR)
3. Select an ESCR
4. Set ESCR fields
5. Set CCCR fields
6. Enable CCCR

Counter Overview

- Event Types
  - 59 event classes
  - 100s of events to count
  - Metric Classifications:
    - General
      Ex: Speculative Uops retired
    - Branching
      Ex: Mispredicted conditionals
    - Trace Cache and Front End
      Ex: Processor N deliver mode
    - Memory
      Ex: MOB Load replays
    - Bus
      Ex: Prefetch bus accesses
    - Characterization
      Ex: Packed SP retired
    - Machine Clear
      Ex: Memory Order Machine Clear
- Counting Types
  - Non-retirement:
  - At-Retirement:
    - Can count BOGUS vs NBOGUS, Tag uops,etc.
    - Mechanisms:
      - Front end tagging
      - Execution tagging
      - Replay Tagging
      - No Tags
  - Also:
    - Event Counting
    - Event Based Sampling
    - Precise EBS

Counting Mechanisms

- Counting Types
  - Non-retirement:
    Events occur any time during execution
  - At-Retirement:
    Events at the retirement of instruction
    - Can count BOGUS vs NBOGUS, Tag uops to count, etc.
      - Terminology➔
    - Mechanisms:
      - Front end tagging (i.e. LD/ST retired)
      - Execution tagging (i.e. packed_DP_retired)
      - Replay Tagging (i.e. L1 misses)
      - No Tags (i.e. uops retired)
  - Also:
    - Event Counting | IEBS | PEBS

At Retirement Counting Terminology

- BOGUS/NBOGUS (speculative)
- Tagging (count uops that encounter event)
- Replay (Data speculation)

Verifying Counter Reader

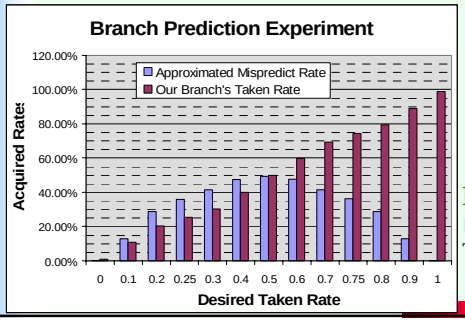
- 1) L1Dcache_exercise:
  - Uses pointer assignment
  - L1=8K, L2=256K
  - Array Size = (L1 Size/Hit Rate)
    - i.e. for 10% Hit rate: 80K ➔ 20K entries
    - Array Size < L2 size
  - Array elements PRBS of array indices
  - Bench loop:
    - new index ← array[old index]

…Verifying Counter Reader

- 2) branch_exercise:
  - Uses random number comparison
  - Assigns 400K PRBS array outside bench loop
    - To avoid rand() instructions in bench loop
  - bench loop:
    - Compares array index to threshod
      - Threshold = RAND_MAX*TakenRate
  - Repeats 1000 reseeding each time
  - However gcc adds 2 more branches into bench loop:
    - Loop exit condition (Prediction ~ 100%)
    - Unconditional JMP (Prediction ~ 100%)
  - Our Branch's Expected Mispredict Rate:
    - ~ (0.5 - |TakenRate – 0.5|)

…Verifying Counter Reader

- 2) branch_exercise results:



Back

Ex:

branch_exercise

Taken Rate=0.5

Details for Real Power Measurement

- These slides provide more detailed description of
  - Machine under test
  - real power measurement method
  - P4 power lines

P4 Details

- Karelian.ee:
  - P4 – 1.4GHz
  - 0.18µ, C4-FC-PGA-423
  - Heatsink → Folded Fin
  - M6, Al interconnect
  - Die Size: 217 mm^2
  - Package Size: 5.34cm x 5.17cm
  - Power: Idle/typ./max=??/51.8/71W
  - D$1&T$1/L2: 8K&12KUops/256K
  - Voltage: 1.7/1.75V

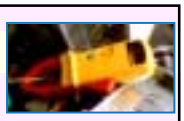
MEASUREMENT Method

- Select **Power lines** that reflect CPU power
  - P4 uses 12 V lines
- Clamp the current probe over the 12V lines
  - 1mV/Adc conversion
- Connect the clamp into DMM
- Send Voltage reading over serial
- Log the voltage readings
  - Convert to instantaneous power as:
    - 12 x V_{sample} x 1000
- Log Power values
- Plot Power values

MEASUREMENT Tools

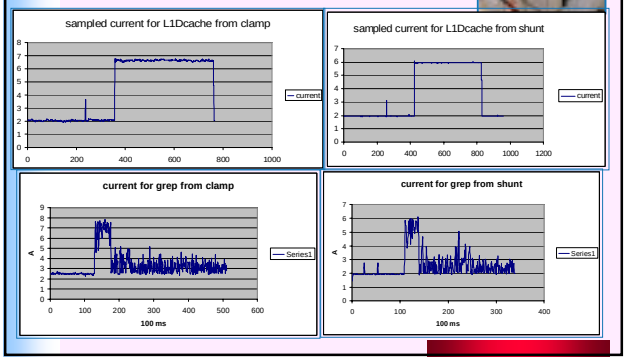
- Poll serial port ~20ms
  - quicker → overkill, slower → overlook
- Compute running average
- sample every Δt you select
  - Easier to sync with Power Model
- PowerMeter:
  - Convert voltage reading to power and log
    - P=12 x V_{read} x 1000
- PowerPlotter:
  - Plot Power samples over sliding time window
    - 100 s history with 1000 samples (Δt = 100ms)

Current Probe



- Fluke i410
- Uses Hall Voltage to measure current and convert to Voltage:
  - 1mV / Adc
- Range: 0.1 – 400A
- Accuracy: 3.5%
- Generated voltage is fed to DMM
- Compared against the **Ppro** Amoeba shunt setup for verification
  - → → → → → → → → → → → → → → →

Clamp vs Shunt

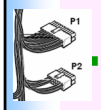


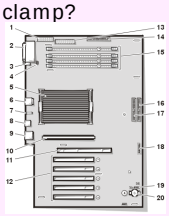
DMM

- Agilent 34401A
- Measurement Motive:
  - We should sample as quick as possible (grep case)
- Measurement Setup:
  - Fast 4 digit, Autozero OFF, Display OFF
    - From [8], 1000 readings/s (x150 faster than fast 6 digit)
- Serial Interface:
  - From [9] 55 ASCII readings /s
    - Polling serial port faster than 20ms is overkill

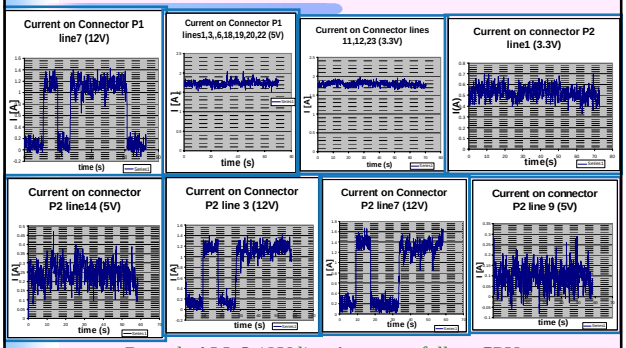
P4 Power Lines

- Which power lines should we cut / clamp?
  - [5] shows the power lines:
    - **1**-CPU power connector
    - **13**-System power connector
    - P1 → 13 & P2 → 1
  - [6],[7] say P4 uses 12V lines for CPU, rather than 5V lines
    - Both P1 & P2 have 12, 5 and 3.3 V lines
  - I run branch_exercise (takenRate=1) and gzip_static → obtain the current variation on the lines
    - → → → → → → → → → → → → → → → → →





Current on Power Lines



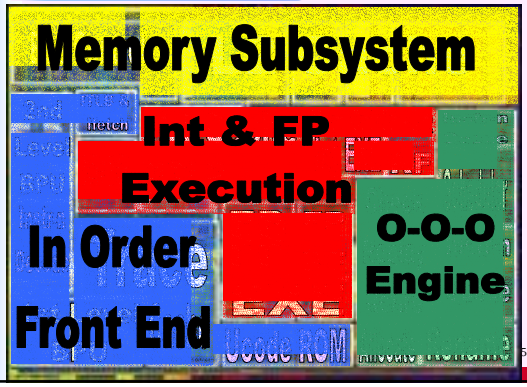
Reveals ALL 3 12V lines' currents follow CPU activity → All add to CPU Power!

Back

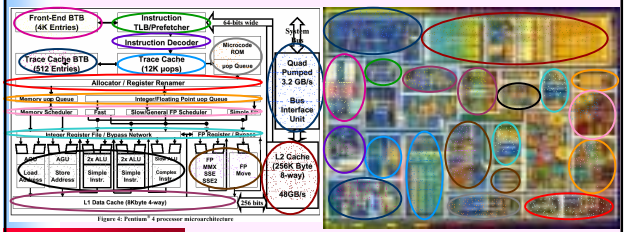
Details for Power Model

- These slides provide more detail for:
  - How we define the 22 components
  - Used counters and the rotation scheme
  - Tuning

Defining Components



P4 Architecture vs Layout



Components to Model:

1) Bus Control
2) L2 Cache
3) 2nd Level BPU
4) ITLB & Ifetch
5) L1 Cache
6) MOB
7) Mem Control
8) DTLB
9) Int EXE
10) FP EXE
11) Int RF
12) FP RF
13) Decode
14) Trace $
15) 1st Level BPU
16) Microcode ROM
17) Allocation
18) Rename
19) Inst-n Qs
20) Schedule
21) Inst-n Qs
22) Retirement

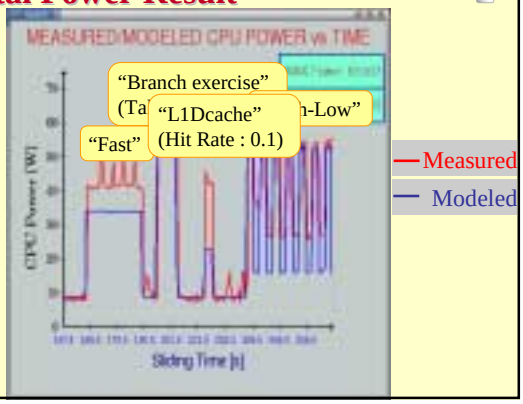
Back

Counter Rotations

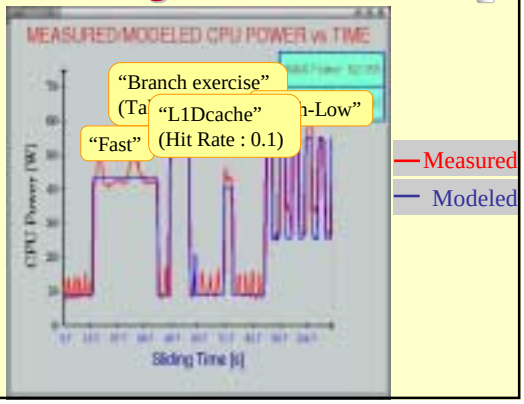
Counters	Rotation1	Rotation2	Rotation3	Rotation4
cntr0	IOQ_allocation	IOQ_allocation	FSB_data_activity	FSB_data_activity
cntr1	BSQ_cache_ref	BSQ_cache_ref	BSQ_cache_ref	BSQ_cache_ref
cntr2	BPU_fetch_rqsts	BPU_fetch_rqsts	MOB_ld_replay	MOB_ld_replay
cntr3	ITLB_reference	ITLB_reference	ITLB_reference	ITLB_reference
cntr4	uop_queue_writes(0x07)	uop_queue_writes(0x07)	uop_queue_writes(0x07)	uop_queue_writes(0x07)
cntr5	TC_deliver_mode	TC_deliver_mode	TC_deliver_mode	TC_deliver_mode
cntr6	uop_queue_writes(0x04)	uop_queue_writes(0x04)	uop_queue_writes(0x04)	uop_queue_writes(0x04)
cntr7	-	-	-	-
cntr8	packed_SP_uop	scalar_SP_uop	64bit_MMX_uop	x87_FP_uop
cntr9	LD_port_replay	LD_port_replay	LD_port_replay	LD_port_replay
cntr10	packed_DP_uop	scalar_DP_uop	128bit_MMX_uop	x87_SIMD_moves_uop
cntr11	ST_port_replay	ST_port_replay	ST_port_replay	ST_port_replay
cntr12	branch_retired	branch_retired	machine_clear	machine_clear
cntr13	uops_retired	uops_retired	uops_retired	uops_retired
cntr14	front_end_event	front_end_event	front_end_event	front_end_event
cntr15	uop_type	uop_type	uop_type	uop_type
cntr16	-	-	-	-
cntr17	-	-	-	-

Back

Area Based Power Estimate – Total Power Result



After Tuning?



Component Breakdowns