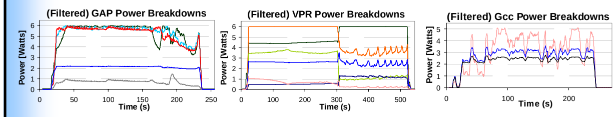
Identifying Program Power Phase Behavior Using Power Vectors

Canturk Isci & Margaret Martonosi
Princeton University

WWC-6
10.27.2003
Austin, TX

Power Phase Behavior

- Existence of distinguishable intervals during an application's execution lifetime such that:
  - They share significantly higher resemblance within themselves in terms of power behavior the application exhibits on a given processor
  - This similarity is carried out by not only the total processor power, but also the distribution of power into processor sub-units



Our Power Phase Analysis

❖**Goal:**
- Identify phases in program power behavior
- Determine execution points that correspond to these phases
- Define small set of power signatures that represent overall power behavior

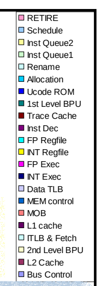
❖**Our Approach – Outline:**
- Collect samples of estimated power values for processor sub-units ***<Power Vectors>*** at application runtime
- Define a power vector similarity metric
- Group sampled program execution into phases
- Determine execution points and representative signature vectors for each phase group
- Analyze the accuracy of our approximation

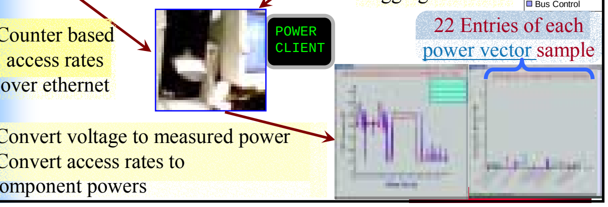
Motivation

❖**Characterizing power behavior:**
- Future power-aware architectures and applications
- Dynamic power/thermal management
- Architecture research

❖**Utilizing power vectors:**
- Direct relation to actual processor power consumption
- Acquired at runtime
- Identify program phases with no knowledge of application

Generating Power Vectors





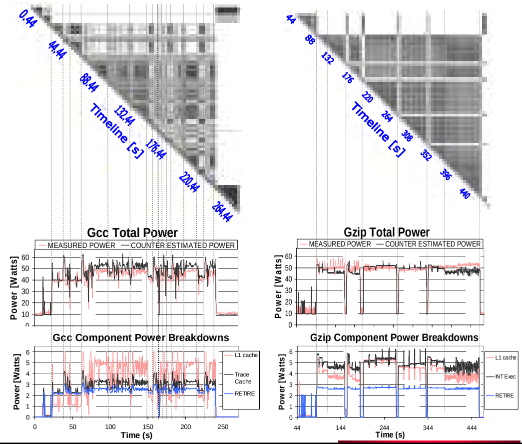
Power Vector Similarity Metric

- How to quantify the 'power behavior dissimilarity' between two execution points?
  1. Consider solely total power difference ☒ $|Total\ Power_r - Total\ Power_c|$
  2. Consider manhattan distance between the corresponding 2 vectors ☒ $\sum_{i=1}^{22} |PV_r(i) - PV_c(i)|$
  3. Consider manhattan distance between the corresponding 2 vectors normalized ☒ $\sum_{i=1}^{22} |NPV_r(i) - NPV_c(i)|$
  4. Consider a combination of (2) & (3) ☑
- Construct a "similarity matrix" to represent similarity among all pairs of execution points
  - Each entry in the similarity matrix:

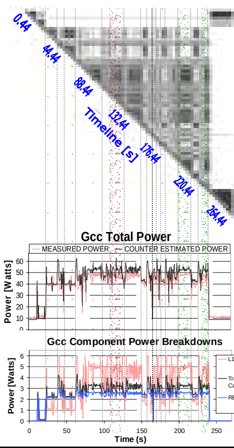
$$FM(r,c) = \min\left(\frac{OM(r,c)}{\max_{r',c'}(OM(r',c'))} + \frac{NM(r,c)}{\max_{r',c'}(NM(r',c'))}, 1\right)$$

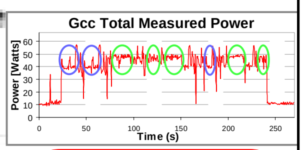
Gcc & Gzip Matrix Plots

(Similar to SimPoints work of Sherwood et al.)



Gcc & Gzip Matrix Plots



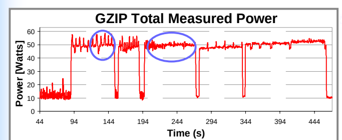


Gcc Elaboration:
Very variant power

Almost identical power behavior at 30, 50, 180s.

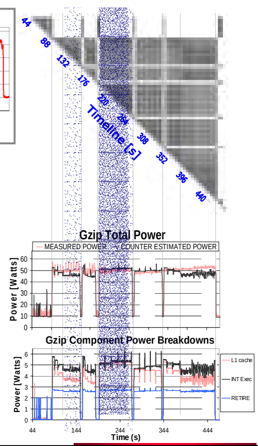
Although 88s, 110s, 140s, 210s and 230s show similar total power; 88, 210 and 230 share higher similarity.

Gcc & Gzip Matrix P



Gzip Elaboration:
Much regular power behavior

Spurious similarities are again distinguished by the similarity analysis



Grouping Execution Points

- **"Thresholding Algorithm":**
  - Define a threshold of similarity < % of max dissimilarity>
  - Start from first execution point (0,0) and identify ones in the fwd execution path that lie within threshold for both normalized and absolute metrics
  - Tag the corresponding execution points (j,j) as the same group
  - Find next untagged execution point (r,r) and do the same along forward path
  - Rule: A tagged execution point cannot add new elements to its group!

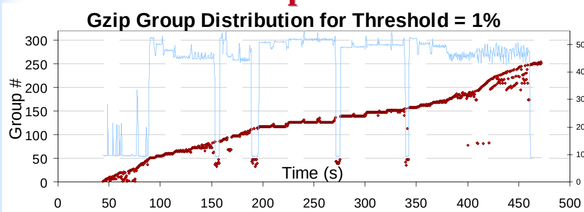
- We demonstrate the outcome of thresholding with **Grouping Matrices**

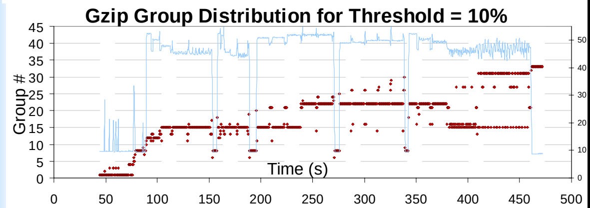
11

Gzip Grouping Matrices

- Gzip has 974 power vectors
- Cluster vectors based on similarity using "thresholding"
  - Max Gzip power dissimilarity: 47.35W

Generated Group Distributions



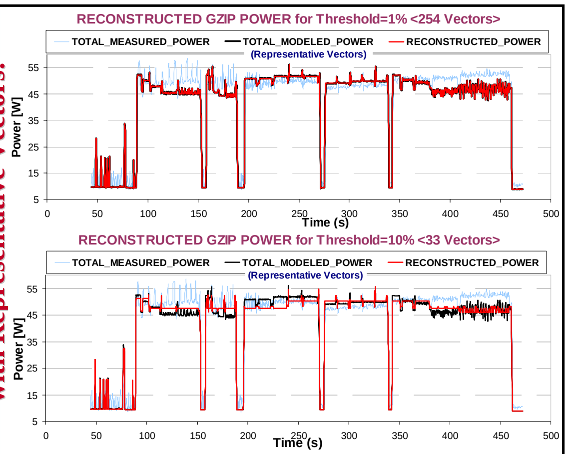


Representative Vectors & Execution Points

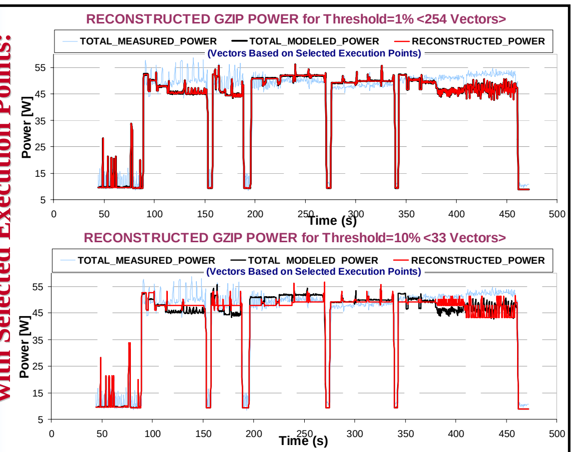
- We have each execution point assigned to a group
- For Each Group:
  - Define a representative vector as the average of all instances of that group
  - Select the execution point that started the group (The earliest point in each group)
- For Each Execution Point:
  - Assign the corresponding group's representative vector as that point's power vector
  - Assign the power vector of the selected execution point for that group as that point's power vector
- We can represent whole execution with as many power vectors as the number of generated groups

14

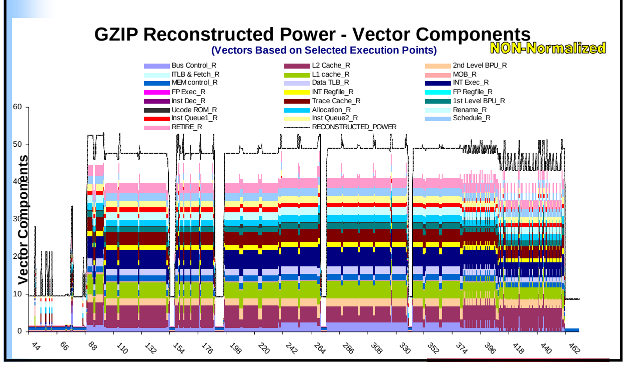
Reconstructing Power Trace with Representative Vectors:



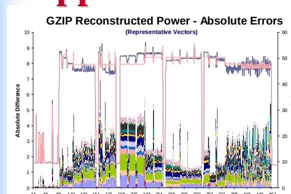
Reconstructing Power Trace with Selected Execution Points:

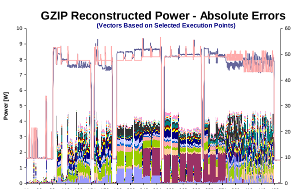


Component Power Characterizations



Approximation Error





- Due to thresholding algorithm → Errors for selected exec. points are bounded with the threshold
  - Max Error: 4.71W & RMS Error: 3.08W
- As representative vectors are group centroids → Cumulative errors for repr. vectors are lower
  - Max Error: 7.10W & RMS Error: 2.31W
- Error in total power< Σ(Component errors)

18

Conclusion

- Presented a power oriented methodology to identify program phases that uses power vectors generated during program runtime
- Provided a similarity metric to quantify power behavior similarity of different execution samples
- Demonstrated our representative sampling technique to characterize program power behavior
- Can be useful for power & characterization research:
  - Power Phase identification/prediction
  - Reduced power simulation
  - Dynamic power/thermal management

Related Work

- Dhodapkar and Smith [ISCA'02]
  - Working set signatures to detect phase changes
- Sherwood et. al. [PACT'01,ASPLOS'02,ISCA'30]
  - Similarity analysis based on program basic block profiles to identify phases
- Todi [WWC'01]
  - Clustering based on counter information to identify similar behavior
- Our work in comparison
  - Power oriented
  - Power behavior similarity metric
  - Runtime
  - No information about the application is required
  - Bounded approximation error with thresholding

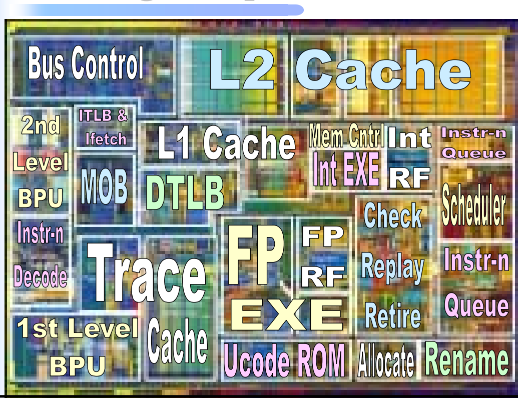
EOP

EXTRA SLIDES

- More detail on Power Vectors
- Similarity matrices and equations for all discussed techniques
- Exemplified description of the two matrices and plots
- Discussion of the ongoing and future research
- Includes also some new ideas and some things to do to make our current analysis solid
- Starts with the discussion of presented work
- Discusses some shortcomings, things that need to be done to improve and to verify that it is unique and solid
  - (Some parts of current work also discusses these issues)
- Also provides some answers to reviewers' questions
- Includes some possible new ideas

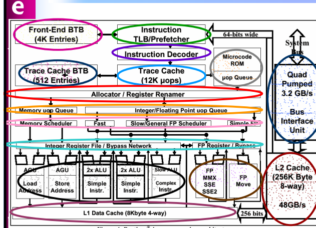
Power Vector Components

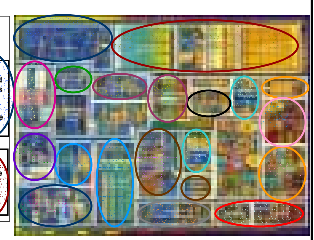
Defining Components



Power Vector 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

Power Vector Components

Defining Events → Access Rates

- We determined 24 events to approximate access rates for 22 components
- Used Several Heuristics to represent each access rate
- **Examples:**

	Access Heuristics
Bus Control	$\frac{IOQ\ Allocation}{\Delta Cycles_1} + \frac{Bus\ Ratio \cdot FSB\ Data\ Activity}{\Delta Cycles_2}$
Front End BPU	$\frac{8 \cdot ITLB\ Reference}{\Delta Cycles_1} + \frac{Branch\ Retired}{\Delta Cycles_2}$
L1 Cache	$\frac{Ld\ Port\ Replay + St\ Port\ Replay}{\Delta Cycles_1} + \frac{Front\ End\ Event}{\Delta Cycles_2}$
Trace Cache	$\frac{Uop\ Queue\ Writes}{\Delta Cycles_1}$
Integer Execution	$2 \cdot (\frac{Uop\ Queue\ Writes}{\Delta Cycles_1} - FP\ Exe.\ Access\ Rate) - L1\ Cache\ Access\ Rate - \frac{Branch\ Retired}{\Delta Cycles_2}$

- Need to rotate counters 4 times to collect all event data
  - Used 15 counters & 4 rotations to collect all event data

25

Power Vector Components

Access Rates → Component Powers

- "Performance Counter based Access Rate estimations are used as proxy for max component power weighting together with microarchitectural details in order to estimate processor sub-unit powers"

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

- EX: Trace cache delivers 3 uops/cycle in deliver mode and 1 uop/cycle in build mode:
  - Power(TC)=[Access-Rate(TC)/3 + Access-Rate(ID)] x MaxPower(TC) + Non-gated TC CLK power
- Total power is computed as the sum of all 22 component powers + measured idle power (8W):

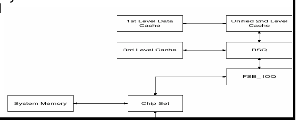
$$Total\ Power = \sum_{i=1}^{22} Power(C_i) + Idle\ Power$$

26

Power Vector Components

Counter Access Heuristics

- 1) BUS CONTROL:
  - No 3rd Level cache → BSQ allocations ~ IOQ allocations
  - Metric1: Bus accesses from all agents
    - Event: IOQ_allocation
      - Counts various types of bus transactions
        - Should account for BSQ as well
      - access based rather than duration
    - MASK:
      - Default req. type, all read (128B) and write (64B) types, include OWN,OTHER and PREFETCH
  - Metric2: Bus Utilization(The % of time Bus is utilized)
    - Event: FSB_data_activity
      - Counts DataReaDY and DataBuSY events on Bus
    - Mask:
      - Count when processor or other agents drive/read/reserve the bus
    - Expression: FSB_data_activity x BusRatio / Clocks Elapsed
      - To account for clock ratios



Power Vector Components

Counter Access Heuristics

- 2) L2 Cache:
  - Metric: 2nd Level cache references
    - Event: BSQ_cache_reference
      - Counts cache ref-s as seen by bus unit
    - MASK:
      - All MESI read misses (LD & RFO)
      - 2nd level WR misses
- 3) 2nd Level BPU:
  - Metric 1: Instructions fetched from L2 (predict)
    - Event: ITLB_Reference
      - Counts ITLB translations
    - Mask:
      - All hits, misses & UC hits
  - Metric 2: Branches retired (history update)
    - Event: branch_retired
      - Counts branches retired
    - Mask:
      - Count all Taken/NT/Predicted/MissP

28

Power Vector Components

Counter Access Heuristics

- 4) ITLB & I-Fetch:
  - etc.........
- 10) FP Execution:
  - Metric: FP instructions executed
    - event1: packed_SP_uop
      - counts packed single precision uops
    - event2: packed_DP_uop
      - counts packed single precision uops
    - event3: scalar_SP_uop
      - counts scalar double precision uops
    - event4: scalar_DP_uop
      - counts scalar double precision uops
    - event5: 64bit_MMX_uop
      - counts MMX uops with 64bit SIMD operands
    - event6: 128bit_MMX_uop
      - counts integer SSE2 uops with 128bit SIMD operands
    - event7: x87_FP_UOP
      - counts x87 FP uops
    - event8: x87_SIMD_moves_uop
      - counts x87, FP, MMX, SSE, SSE2 ld/st/mov uops

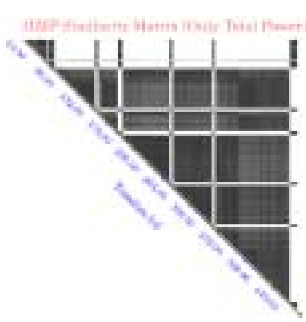
Back

29

Different Similarity Metrics

Similarity Based on Total Power

$$Total\ Similarity\ Matrix(r,c) = |Total\ Power_r - Total\ Power_c|$$

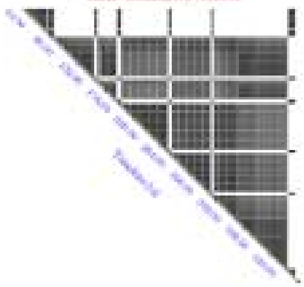


30

Similarity Based on Absolute Power Vectors

Different Similarity Metrics

$$Original\ Similarity\ Matrix(r,c) = \sum_{i=1}^{22} |PV_r(i) - PV_c(i)|$$

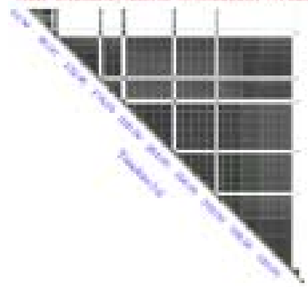


31

Similarity Based on Normalized Power Vectors

Different Similarity Metrics

$$Normalized\ Similarity\ Matrix(r,c) = \sum_{i=1}^{22} |NPV_r(i) - NPV_c(i)|$$

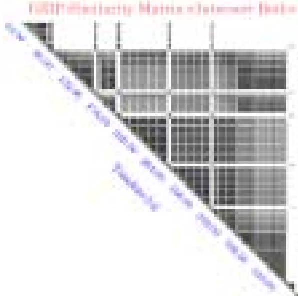


32

Similarity Based on Both Absolute and Normalized Power Vectors

Different Similarity Metrics

$$FM(r,c) = \min\left(\frac{OM(r,c)}{\max_{r',c'}(OM(r',c'))} + \frac{NM(r,c)}{\max_{r',c'}(NM(r',c'))}, 1\right)$$



Back

33

Similarity Matrix Example

Similarity & Grouping Matrices

- Consider 4 vectors, each with 4 dimensions:

0	1	2	3
5	3	4	1
3	5	4	5
2	1	2	3
1	2	2	4

Exemplary Manhattan Distance Calculation :
Distance Between Vectors 1 & 2 :
$|3-4|+|5-4|+|1-2|+|2-2| = 3$

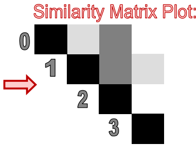
- Log all distances in the similarity matrix
- Color-scale from black to white (only for upper diagonal)

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

⇨

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

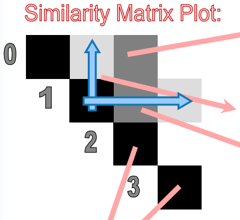
⇨ Similarity Matrix Plot:



Interpreting Similarity Matrix Plot

Similarity & Grouping Matrices

Similarity Matrix Plot:



- Level of darkness at any location (r,c) shows the amount of
- Vertically above the diagonal shows
- Horizontally right of the diagonal shows similarity of the sample at the diagonal to future samples
  - i.e. 1 vs. 2,3
- All samples are perfectly similar to themselves
  - All (r,r) are black

Back

35

Grouping Matrix Example

Similarity & Grouping Matrices

0	1	2	3
5	3	4	1
3	5	4	5
2	1	2	3
1	2	2	4

- Consider same 4 vectors:

Maximum Distance Between Vectors : 7
10% Threshod ⇒ 0.7, none can be grouped together
50% Threshold ⇒ Vectors with distance ≤ 3.5 can be grouped together

- Mark execution pairs with distance ≤ Threshold

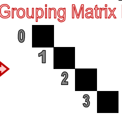
Grouping Matrix Plots:

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

⇨

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

⇨



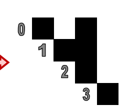
Threshold: 10%

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

⇨

	0	1	2	3
0	0	6	3	7
1	6	0	3	6
2	3	3	0	7
3	7	6	7	0

⇨



Threshold: 50%

Back

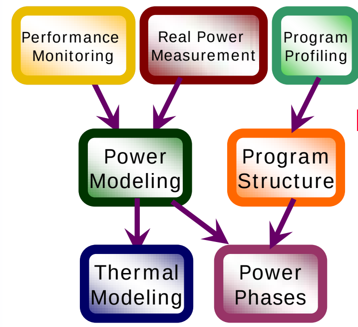
Current & Future Research

❖ FOLLOWING SLIDES DISCUSS ONGOING RESEARCH RELATED TO POWER PHASES. PLANS FOR FUTURE RESEARCH ARE ALSO DISCUSSED

Current & Future Research

37

THE BIG PICTURE



Bottom line…

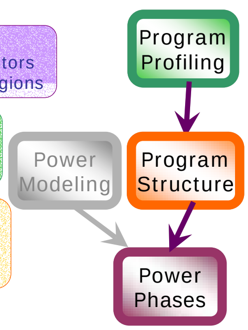
- *To Estimate component power & temperature breakdowns for P4 at runtime…*
- *To analyze how power phase behavior relates to program structure*

Current & Future Research

38

Phase Branch

- ❖ Power Phase Behavior
  - Similarity Based on Power Vectors
  - Identifying similar program regions
- ❖ Profiling Execution Flow
  - Sampling process' execution
  - "PCsampler" LKM
- ❖ Program Structure
  - Execution vs. Code space
  - Power Phases ⇔ Exec. Phases
    - NOT YET?



Current & Future Research

39

Discussion→From here on

- ❖ Generality of the technique
  - All Spec benchmarks show distinct phase behavior:
- ❖ Repeatability of the experiment
  - Need to be able to arrive at similar phase behavior in order to characterize an application
- ❖ Correlation between vector components
  - Inherent redundancy in power vectors
  - Could be removed with PCA
- ❖ Alternative norms for similarity
- ❖ Applicability of selected execution points

Questions & Rebuttals

40

Similarity Analysis for Other Applications?

- ❖ SPECs show similar applicable behavior
  - Not always phase-like, i.e. twolf has more like a power gradient
- ❖ Results for other benchmarks: <NOT READY>
  - Gcc & Twolf:
    - # of groups w.r.t. thresholds
    - Errors plots for reconstructed & selected vectors
- ❖ Apply to other applications:
  - Desktop applications
    - Will follow the bursty behavior, maybe determine action signatures??
      - Saving, computation, streaming, etc.
  - Ghostscript might be interesting
    - Correlation between phases vs. locations of images

Questions & Rebuttals

41

Dependence of Results to the Applied Power Model?

- ❖ The generic technique requires only sufficiently detailed power breakdowns that add up to total power
  - Doesn't matter how you acquire the 'power vectors' otherwise
- ❖ If you use some other characterization data other than power vectors
  - Can still perform phase analysis, but cannot provide a direct estimation for reconstructed power behavior
    - Maybe use some kind of mapping?
    - Log measured power data as well as characterization metrics?
      - Would still be unable to predict component-wise

Questions & Rebuttals

42

Questions & Rebuttals

Possibility for Other Processors?

- Most recent processors are keen on power management
  - There will be enough power variability to exploit for power phase analysis
- Porting the power estimation to other architectures
  - Requires significant effort to
    - Define power related metrics
    - Implement counter reader and power estimation user and kernel SW
- Porting to same architecture, different implementation
  - More straightforward
    - Reevaluate max/idle/gated power estimates
- Experiences with other architectures:
  - Castle project for Pentium Pro (P6)
    - Few watts of variation
    - Low dimensionality
  - IBM Power3 II
    - Very low measured power variation

43

Questions & Rebuttals

Other Statistical Techniques?

- Alternative measures of distance:
  - Different norms $L_{N_{r,c}} = \sqrt[\frac{1}{N}]{\sum_{i=1}^{22} \left|PV_r(i) - PV_c(i)\right|^N}$
  - Canberra Distance $C(r,c) = \sum_{i=1}^{22} \frac{\left|PV_r(i) - PV_c(i)\right|}{\left|PV_r(i) + PV_c(i)\right|}$
  - Squared Chi-squared distance, etc.
- Other similarity metrics:
  - Pearson's correlation coefficient $PC(r,c) = \sum_{i=1}^{22} \frac{\left|PV_r(i) \cdot PV_c(i)\right|}{22}$
  - Cosine similarity $CS(r,c) = \frac{\sum_{i=1}^{22} PV_r(i) \cdot PV_c(i)}{\sqrt{\sum_{i=1}^{22} PV_r(i)^2 \bullet \sum_{i=1}^{22} PV_c(i)^2}}$

Questions & Rebuttals

SPEClite vs. SimPoints vs. Power Vectors

- Approaches of 3 methods:
  - SPEClite:
    - Performance counts →
      (runtime) ➧ (Perf. Oriented) ➧ (k-means partitioning for vectors normalized to 0 mean and unit variance) ➧ (PCA to create reduced vectors) ➧ (selects vectors closest to centroids)
  - SimPoints:
    - Basic block accesses →
      (Simulation) ➧ (Perf. Oriented) ➧ (k-means partitioning for normalized basic block vectors) ➧ (selects vectors closest to centroids –except for 'Early' Simpoints)
  - Power Vectors:
    - Component power consumptions →
      (Runtime) ➧ (Power Oriented) ➧ (threshold based partitioning for 'normalized + absolute' vectors) ➧ (Selects earliest vector of each group)

45

Questions & Rebuttals

Why Power Vectors w.r.t. Others?

- Provides a direct interpretation for power consumption
  - Could be used to identify specific power behavior for dynamic power/thermal management
- Power phases might be not a perfect translation of performance phases
  - <CURRENT WORK INVESTIGATES>
    - I.e. same basic block accesses during different architectural states
- Generated at runtime
  - Easy repeatability, etc.
- Thresholding provides upper bound estimate for the power approximation with selected execution points

46

Modified Stuff

- **FOLLOWING SLIDES I MODIFIED FROM THE ORIGINAL, BUT STILL KEEP 'EM**

47

Our Power Phase Analysis

- **Goal:**
  - Identify phases in program power behavior
  - Determine execution points that correspond to these phases
  - Define small set of power signatures that represent overall power behavior
- **Our Approach:**
  - Collect samples of estimated power values for processor sub-units **<*Power Vectors*>** at application runtime
  - Define a similarity metric regarding these power vectors
  - Process the outcome of this similarity metric to group sampled execution points (/Power Vectors) into phases
  - Determine execution points and representative signature vectors for each phase group
  - Quantify the closeness of our approximation based on these vectors to original power behavior

Motivation

- **Characterizing power behavior:**
  - Future power-aware architectures and applications
  - Dynamic power/thermal management
    - Multiconfigurable hardware
    - Thread scheduling, DVS, DFS
    - Recurring phase prediction
  - Architecture research
    - Representative –reduced– simulation points
- **Utilizing power vectors:**
  - Direct relation to actual processor power consumption
  - Acquired at runtime →
    - Similarity relations generated quickly
    - Easy repeatability for different datasets/compilations
    - Identify (recurring) phases over large scales of execution
  - Identify program phases with no knowledge of application
    - (i.e. no basic block profile, PC sampling, code space info, etc.)

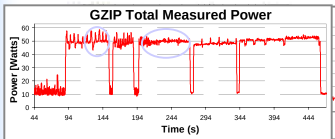
49

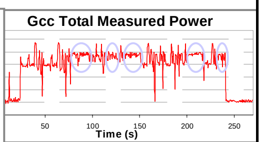
Power Vector Similarity Metric

- How to quantify the 'power behavior dissimilarity' between two execution points?
  1. Consider solely total power difference $|Total\ Power_r - Total\ Power_c|$
     - ☒ Conceals significant phase information
  2. Consider manhattan distance between the corresponding 2 vectors $\sum_{i=1}^{22} |PV_r(i) - PV_c(i)|$
     - ☒ Vectors with small magnitudes are inherently closer
  3. Consider manhattan distance between the corresponding 2 vectors normalized $\sum_{i=1}^{22} |NPV_r(i) - NPV_c(i)|$
     - ☒ Indifferent to magnitude of power consumption
  4. Consider a combination of (2) & (3)
     - ☑ Restricts us to both absolute and ratio-wise similarities
- Construct a "similarity matrix" to represent similarity among all pairs of execution points
  - Each entry in the similarity matrix:

$$FM(r,c) = \min\left(\frac{OM(r,c)}{\max_{r',c'}(OM(r',c'))} + \frac{NM(r,c)}{\max_{r',c'}(NM(r',c'))}, 1\right)$$

Gcc & Gzip Matrix P





Gzip Elaboration:
Much regular power behavior

Spurious similarities such as 100-150s and 200-280 are distinguished by the similarity analysis

Gcc Elaboration:
Very variant power

Almost identical power behavior at 30, 50, 180s.

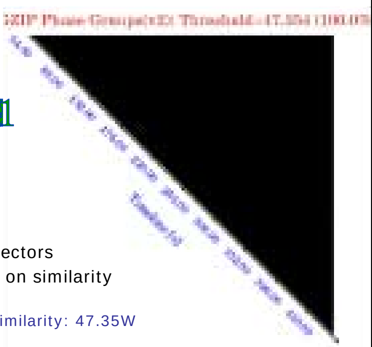
Although 88s, 110s, 140s, 210s and 230s show similar total power; 88, 210 and 230 share higher similarity.

Similarity for Simplicity?

- So, we can identify similar power phases:
  - I.e. informally: if similarity matrix(r,c) is DARK → Execution points r & c have similar power behavior
- **2 Questions:**
  - **1) How do we group the execution points (power vectors) based on their similarity?**
  - **2) Could we represent power behavior with reasonable accuracy, with a small number of 'signature' vectors?**
- Our answer to Q.1: **"Thresholding Algorithm":**
  - Define a threshold of similarity < % of max dissimilarity>
  - Start from first execution point (0,0) and identify ones in the fwd execution path that lie within threshold for both normalized and absolute metrics
  - Tag the corresponding execution points (j,j) as the same group
  - Find next untagged execution point (r,r,) and do the same along fwd path
  - Rule: A tagged execution point cannot add new elements to its group!
    - We demonstrate the outcome of thresholding with **Grouping Matrices**

52

Gzip Grouping Matrices



- Gzip has 974 power vectors
- Cluster vectors based on similarity using "thresholding"
  - Max Gzip power dissimilarity: 47.35W

Conclusion

- Presented a power oriented methodology to identify program phases that uses power vectors generated during program runtime
- Provided a similarity metric to quantify power behavior similarity of different execution samples
- Demonstrated our representative sampling technique to characterize program power behavior
  - Representative vectors for program power signatures
  - Execution points for representative simulation
- Can be useful for power & characterization research:
  - Power Phase identification/prediction
  - Reduced power simulation
  - Dynamic power/thermal management

54