

# Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache

Nathan Sadler and Daniel Sorin  
[sorin@ee.duke.edu](mailto:sorin@ee.duke.edu)

**Department of Electrical & Computer Engineering  
Duke University**



# Overview



- **Motivation: we must tolerate errors in caches**
  - We focus on L1 data cache (L1D)
- **Contributions**
  - *Compare existing schemes for protecting L1D*
  - *Develop new scheme for protecting L1D*
  - *Identify best option(s) for protecting L1D*

# Outline




- **Overview**
- **Existing Error Protection Schemes**
- **New Error Protection Scheme**
- **Experimental Evaluation**
- **Conclusions**


# Error Protection Codes

- **Add  $r$  check bits to each  $k$ -bit datum**
- **EDC = error detecting code**
  - E.g., parity bit
- **ECC = error correcting code**
  - E.g., Hamming codes
- **Hamming distance (HD)**
  - Can detect  $d$ -bit errors with  $HD = d+1$
  - Can correct  $c$ -bit errors with  $HD = 2c+1$


# Existing Scheme #1

- 
- **EDC/ECC**
    - EDC on L1D (generally one parity bit per word)
    - ECC on L2
  - **Detect error in L1D? Get correct data from L2**
  - **Costs**
    - Needs write-thru L1D → more L2 bandwidth & power
    - Extra parity bit in L1D → slightly slower, more power-hungry L1D
  - **Examples: Pentium4, UltraSPARC IV, Power4**


# Existing Scheme #2

- 
- **ECC/ECC**
    - ECC on L1D (at word or block granularity)
    - ECC on L2
  - **Detect error in L1D? Correct it in place**
  - **Costs**
    - Several extra bits in L1D → slower, more power-hungry L1D
  - **Examples: AMD K8, Alpha 21264**

# Qualitative Comparison

- 
- **EDC/ECC, as compared to ECC/ECC**
    - L1D is slightly faster and less power-hungry
    - Uses more L2 bandwidth and L2 power
  - **Goal #1: quantify these differences and identify which option is best**
  - **Goal #2: create scheme that achieves best of both EDC/ECC and ECC/ECC**

# Other Options

- 
- **Many other options for protecting L1D**
    - Keep replicas of blocks in L1D
    - Early writebacks of dirty data
    - Periodically refresh L1D with known-good L2 block
  - **Replication Cache [Zhang, ICS 2004]**
    - Keeps replicas in small dedicated R-cache
    - Protects all L1D blocks, but requires lots of writebacks from R-cache to L2

# Outline

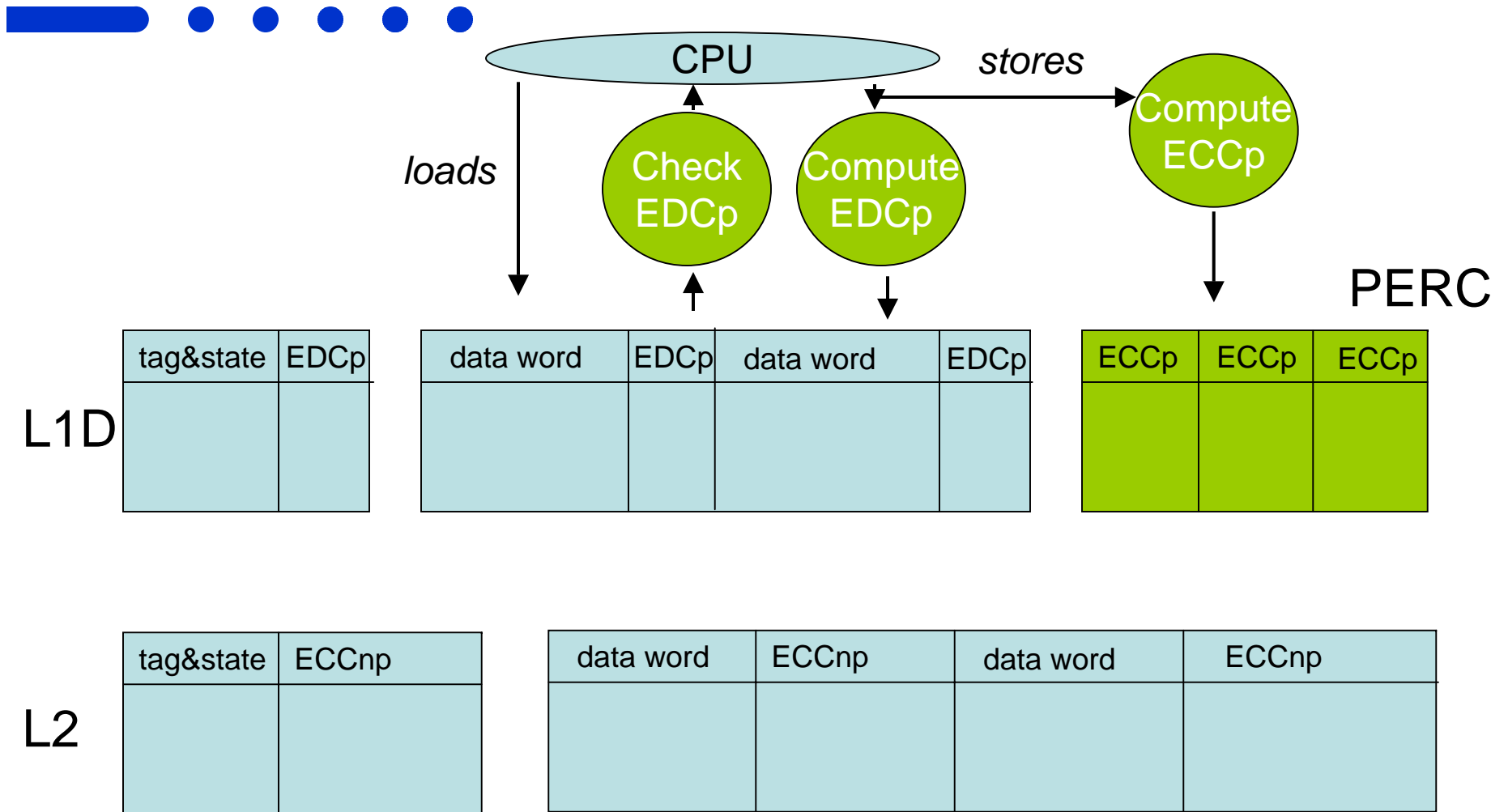


- **Overview**
- **Existing Error Protection Schemes**
- **New Error Protection Scheme**
- **Experimental Evaluation**
- **Conclusions**


# Observation

- **Ideally, we'd get best of both worlds**
  - Like EDC, keep only parity bit per word in L1D
  - Like ECC, avoid write-through traffic to L2
- **Key: use punctured error codes**
  - Can **separate**  $r$  check bits into  $r_d$  bits for detection and  $r_c$  bits for correction ( $r = r_d + r_c$ )
  - Keep just the  $r_d$  EDC bits in L1D (like EDC/ECC)
  - Keep the  $r_c$  ECC bits in dedicated **Punctured Error Recovery Cache (PERC)**


# Cache Hierarchy with PERC



# PERC Error-Free Operation

- 
- **Load hit in L1D**
    - Read data and EDCp from L1D
  - **Store hit in L1D**
    - Write data and EDCp to L1D; write ECCp to PERC
  - **Replacement from L1D to L2**
    - Read data and EDCp from L1D and ECCp from PERC → write this combined block into L2
  - **Fill from L2 to L1D**
    - Read block from L2 → write data and EDCp into L1 and write ECCp into PERC

# PERC Error Recovery

- 
- **Error detected when load hit in L1D obtains EDCp that indicates error**
  - **Recovery process**
    - Read ECCp from PERC and use it with data and EDCp (from L1D) to correct error
    - Provide correct data to processor
    - Write correct data and EDCp into L1D
    - Write correct ECCp into PERC

# Outline

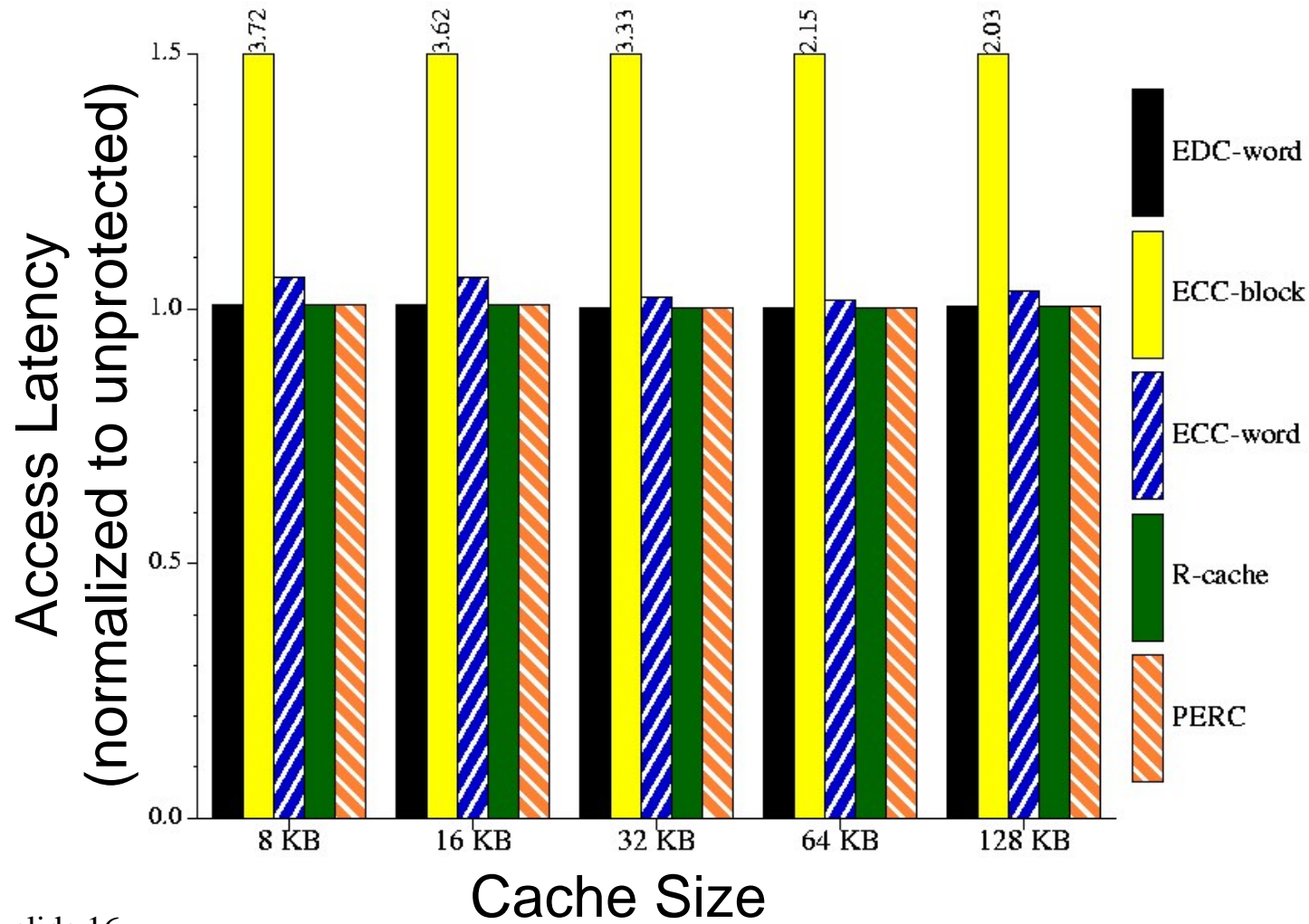


- **Overview**
- **Existing Error Protection Schemes**
- **New Error Protection Scheme**
- **Experimental Evaluation**
- **Conclusions**

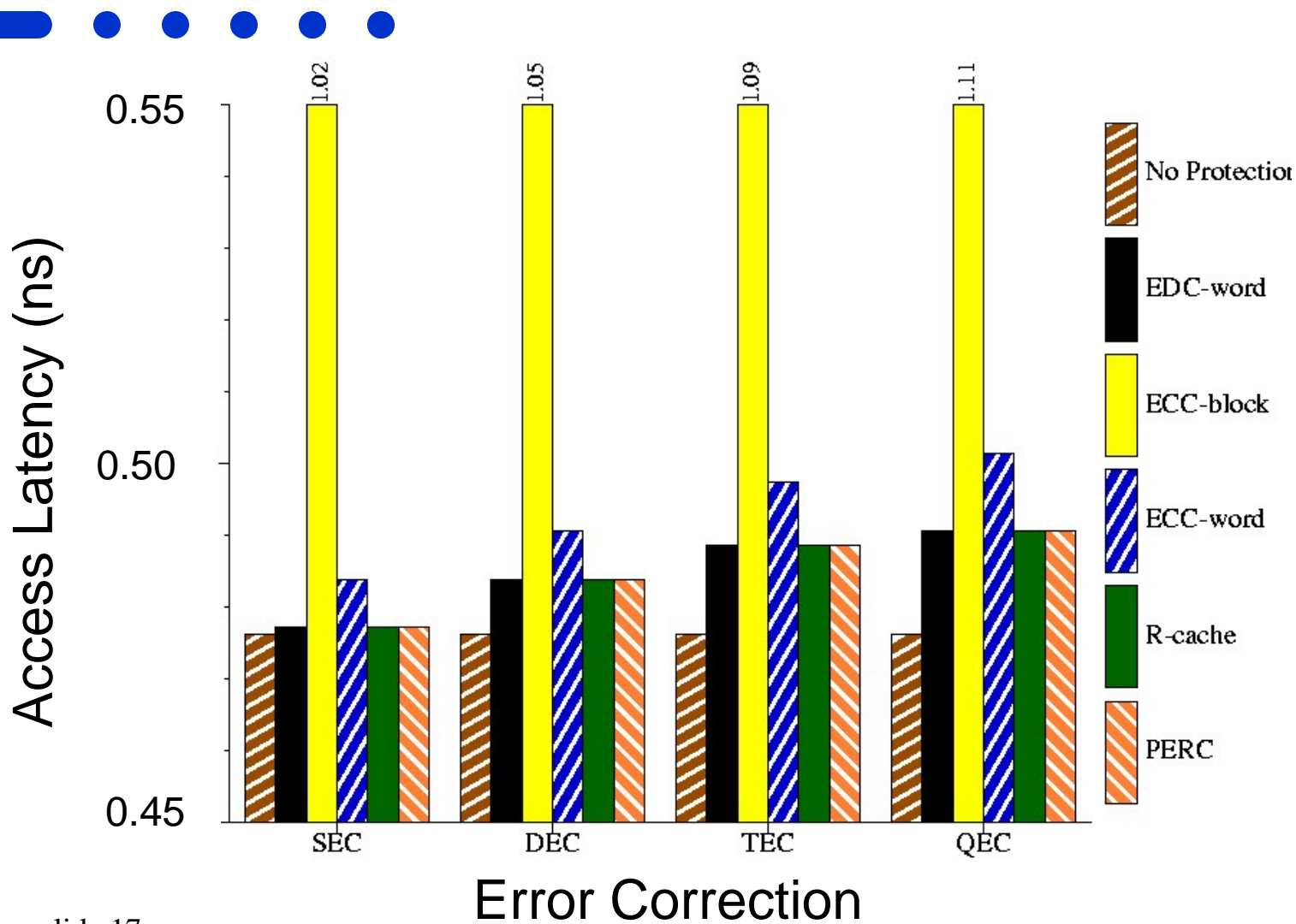
# Experimental Methodology

- **Cacti 3.0 for latency and power**
  - Modeling 90nm technology
- **SimpleScalar 4.0 for processor timing**
  - 12-stage, 4-wide superscalar processor
    - Core: 128-entry ROB, 64-entry LSQ
    - L1 I & L1D: 64KB, 2-way, 32B blocks, 3-cycles
    - L2: 1MB, 16-way, 64B blocks, 8-cycles
  - All SPEC CPU benchmarks
    - Will only show SPECint in this talk
  - SimPoints for sampling from benchmarks

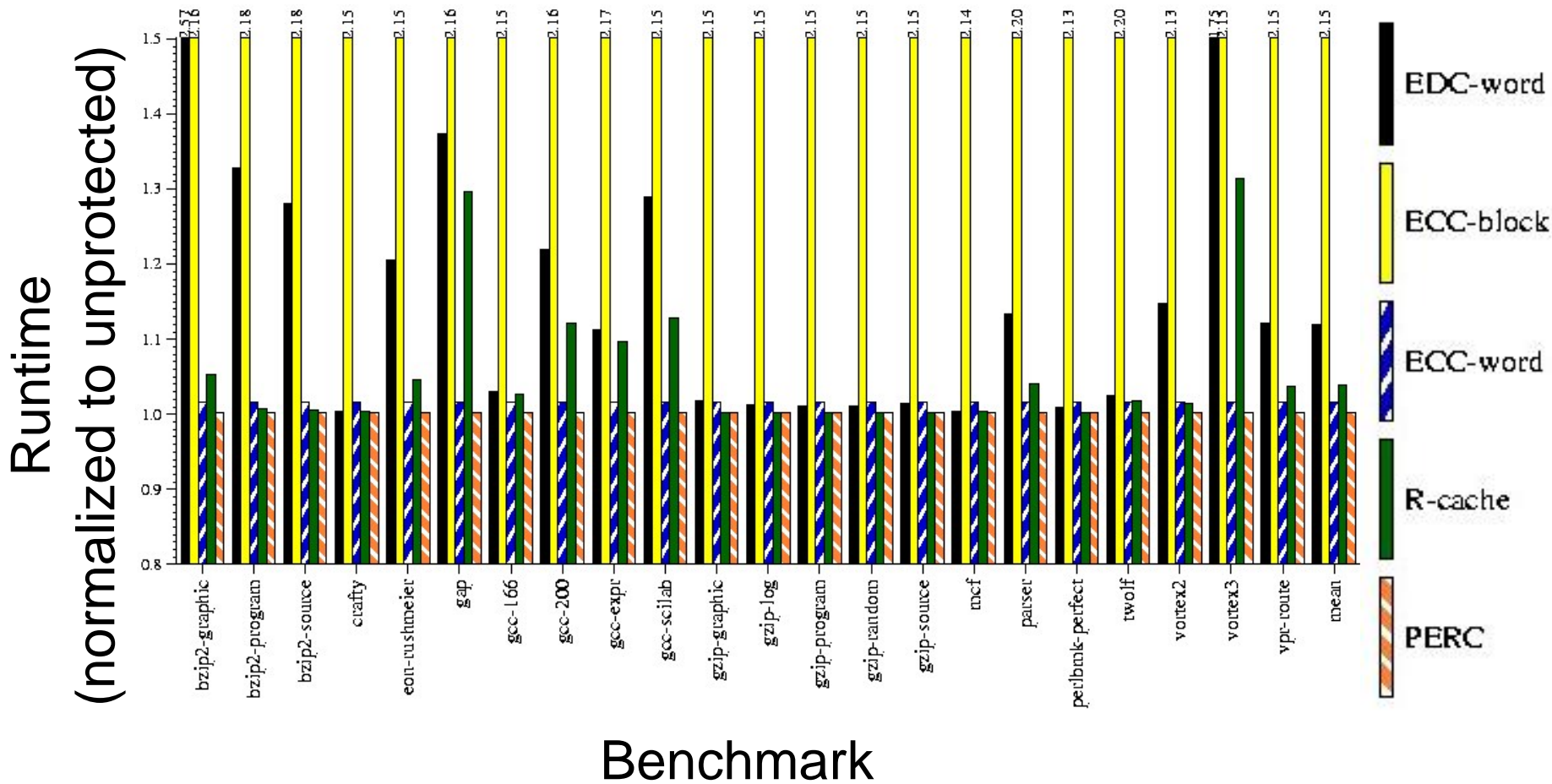
# L1D Access Latency



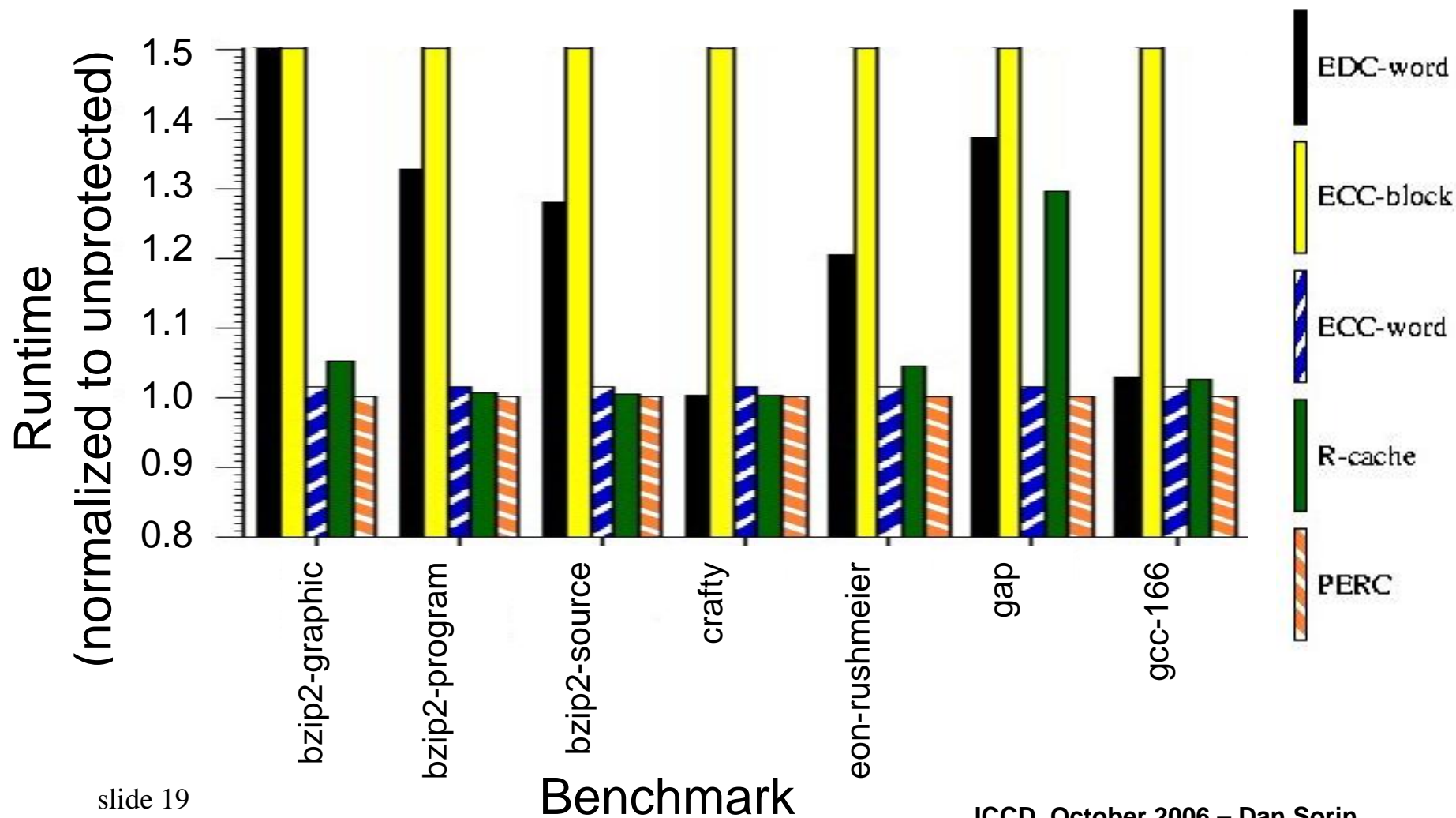
# L1D Latency vs. Error Coverage



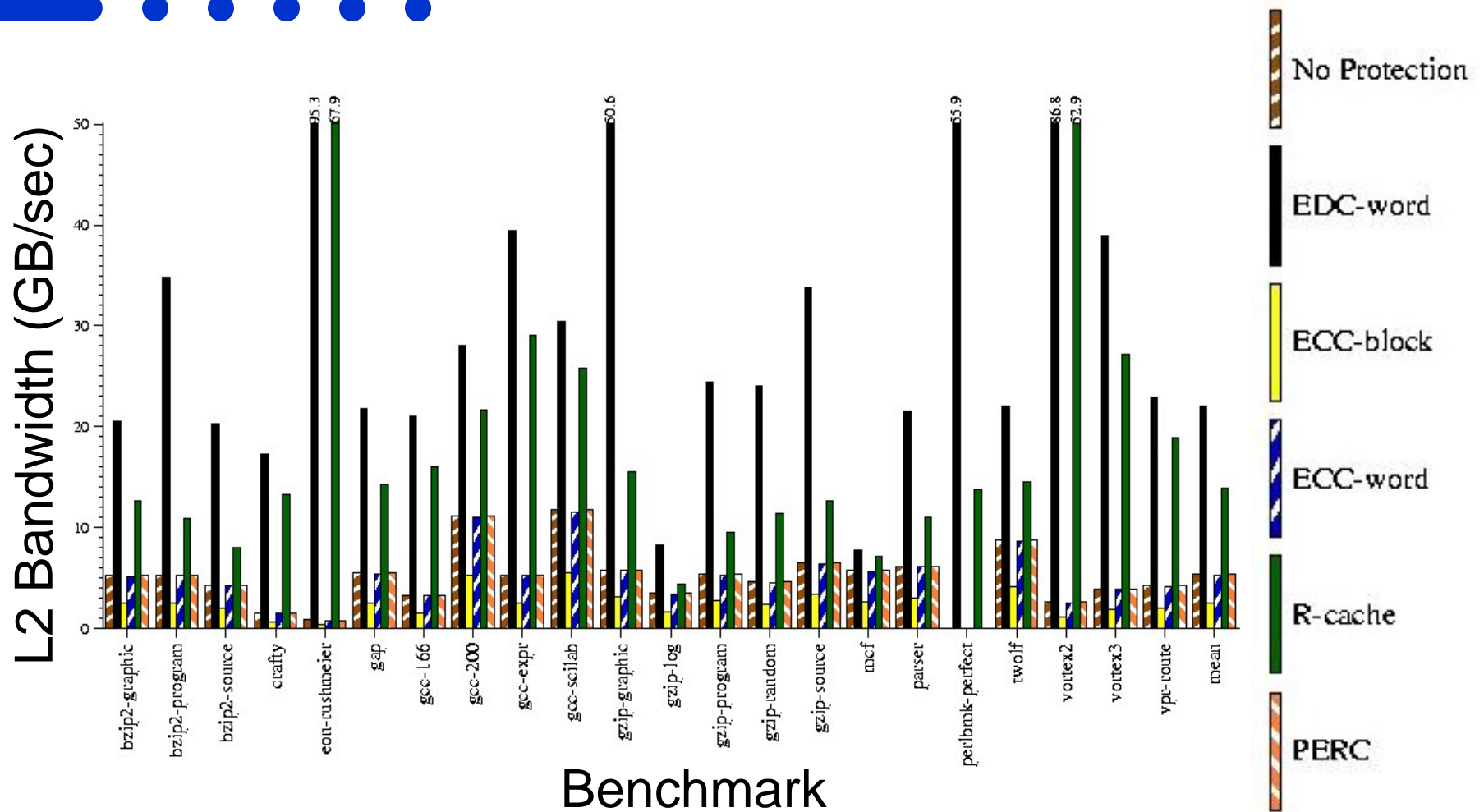
# Runtime



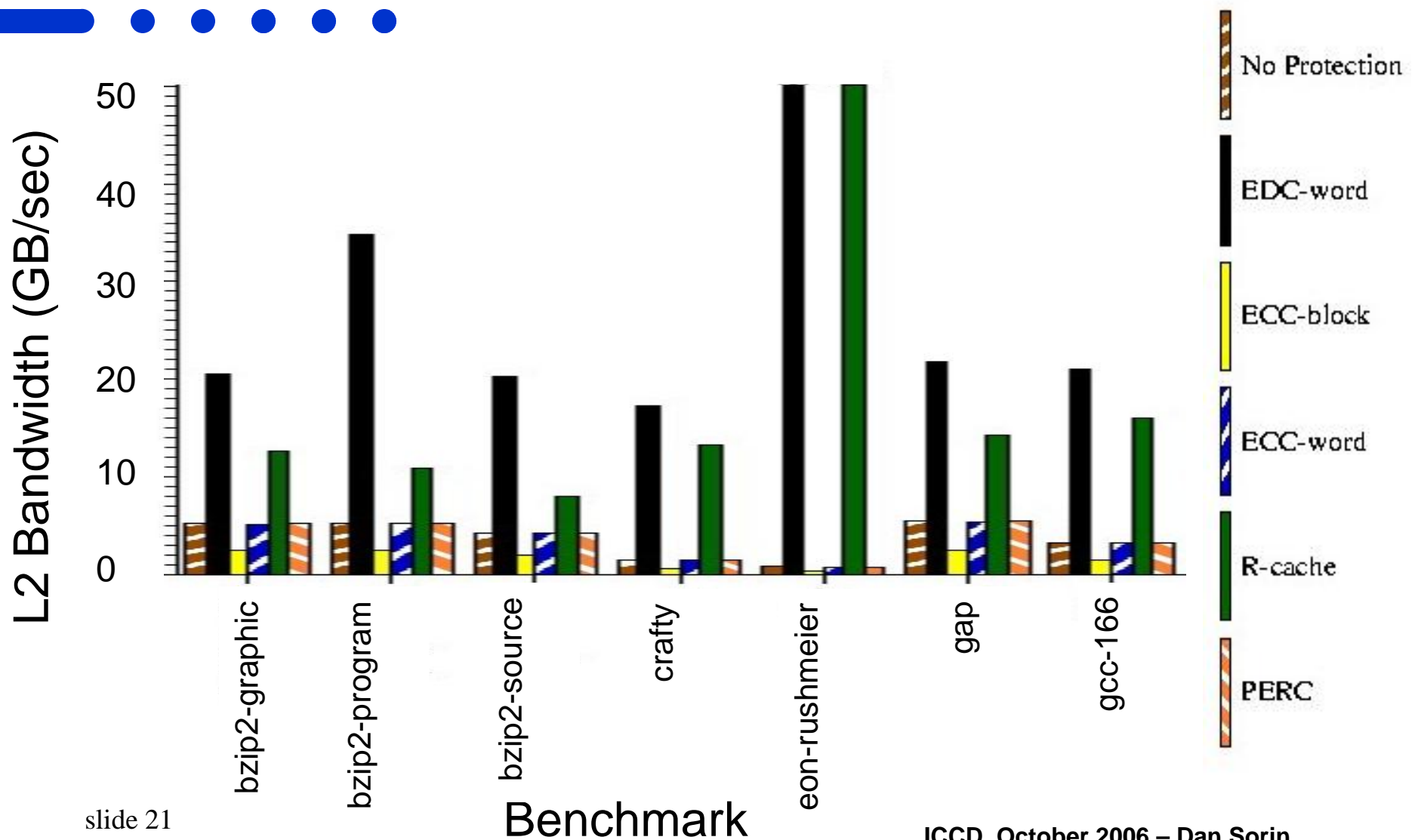
# Runtime



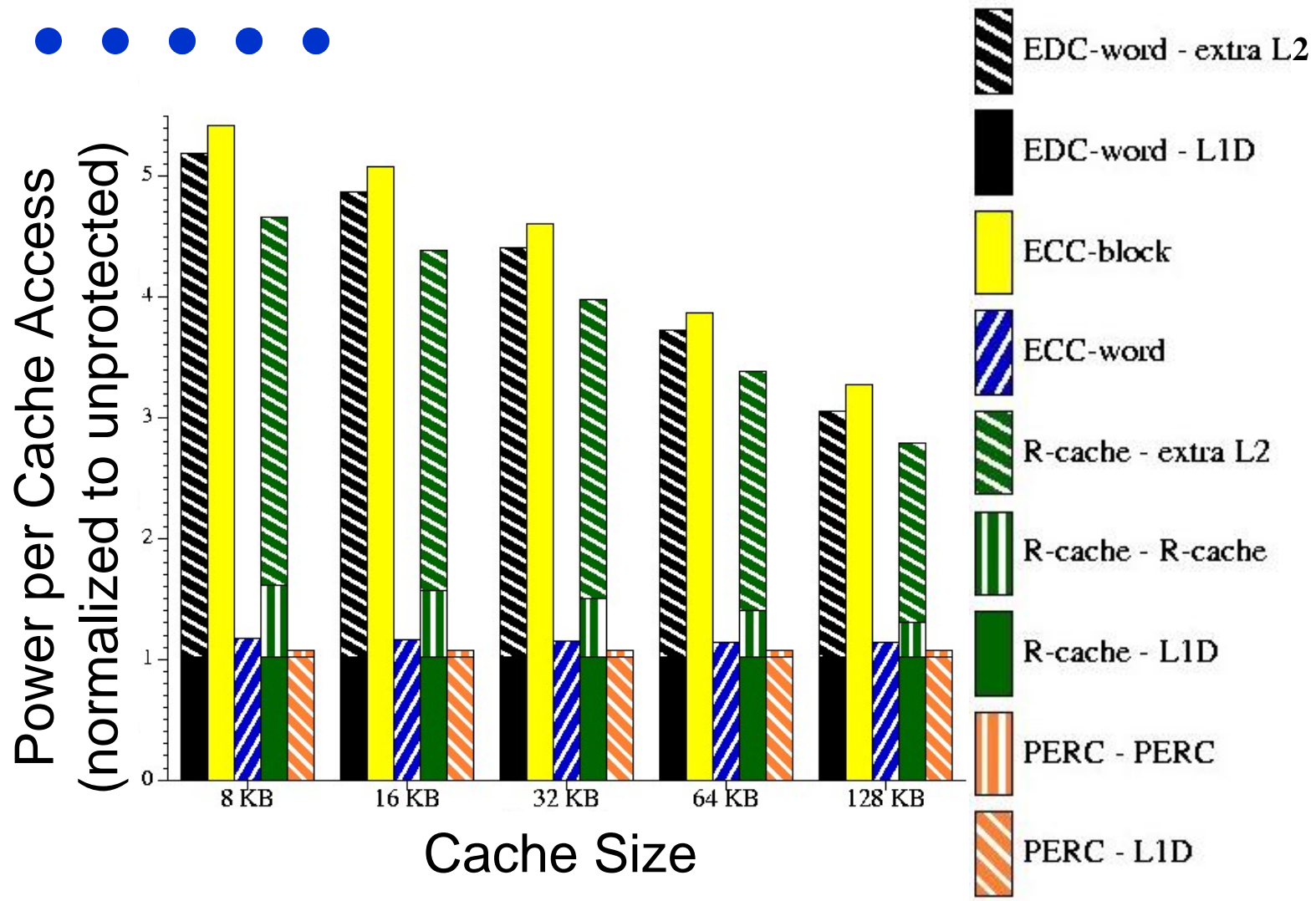
# L2 Cache Bandwidth




# L2 Cache Bandwidth



# Power Consumption



# Conclusions

- 
- **ECC/ECC (at word granularity) clearly better than EDC/ECC**
  - **PERC can achieve “best of both worlds”, although its advantage relative to ECC/ECC is often small**