Near-Threshold Computing: How Close Should We Get?

Alaa R. Alameldeen

Intel Labs

Workshop on Near-Threshold Computing

June 14, 2014

Overview

- High-level talk summarizing my architectural perspective on near-threshold computing
- Near-threshold computing has gained popularity recently
  - Mainly due to the quest for energy efficiency
- Is it really justified?
  + Reduces static and dynamic power
  - Reduces frequency, adds reliability overhead
- The case for selective near-threshold computing
  - Use it , but not everywhere
- Case Studies: VS-ECC and Mixed-Cell Cache Designs

Why Near-threshold Computing?

- Near-threshold computing has gained popularity recently. Why?
  - Mainly: Energy Efficiency
  - Running lots of cores with fixed power budget
  - Avoiding /delaying “dark silicon”
  - Spanning market segments from ultra-mobile to super computing
- Theory:
  - Dynamic power reduces quadratically with operating voltage
  - Static power reduces exponentially with operating voltage
  - The lower voltage we run, the less power we consume

But Obviously, It Is Not Free…

- Latency Cost:

  Lower voltage leads to lower frequency

  - Cores run slower, taking longer to run programs
  - Energy = Power x Time. Lower power doesn't always translate to lower energy
- Reliability Cost:

  Individual transistors and storage elements begin to fail due to smaller margins

  - Whole structures may fail
  - Lots of redundancy or other fault tolerance mechanisms needed (i.e., more area, power, complexity)

Latency Cost

- A lower voltage drives lower frequency
- To the first order, at low voltages, V α f
- Iron Law of processor performance:

$$\text{Program Runtime} = \frac{\textit{Instructions}}{\textit{Program}} \times \frac{\textit{Cycles}}{\textit{Instruction}} \times \frac{\textit{Time}}{\textit{Cycle}}$$

- Lower frequency increases Time/Cycle, therefore increases program runtime

Latency Impact on Energy Efficiency

- A program that runs longer consumes more energy

  Energy = Power x Time

  Program Energy = Average Power x Program Runtime

- Even if average power is lower, it's possible energy will be higher

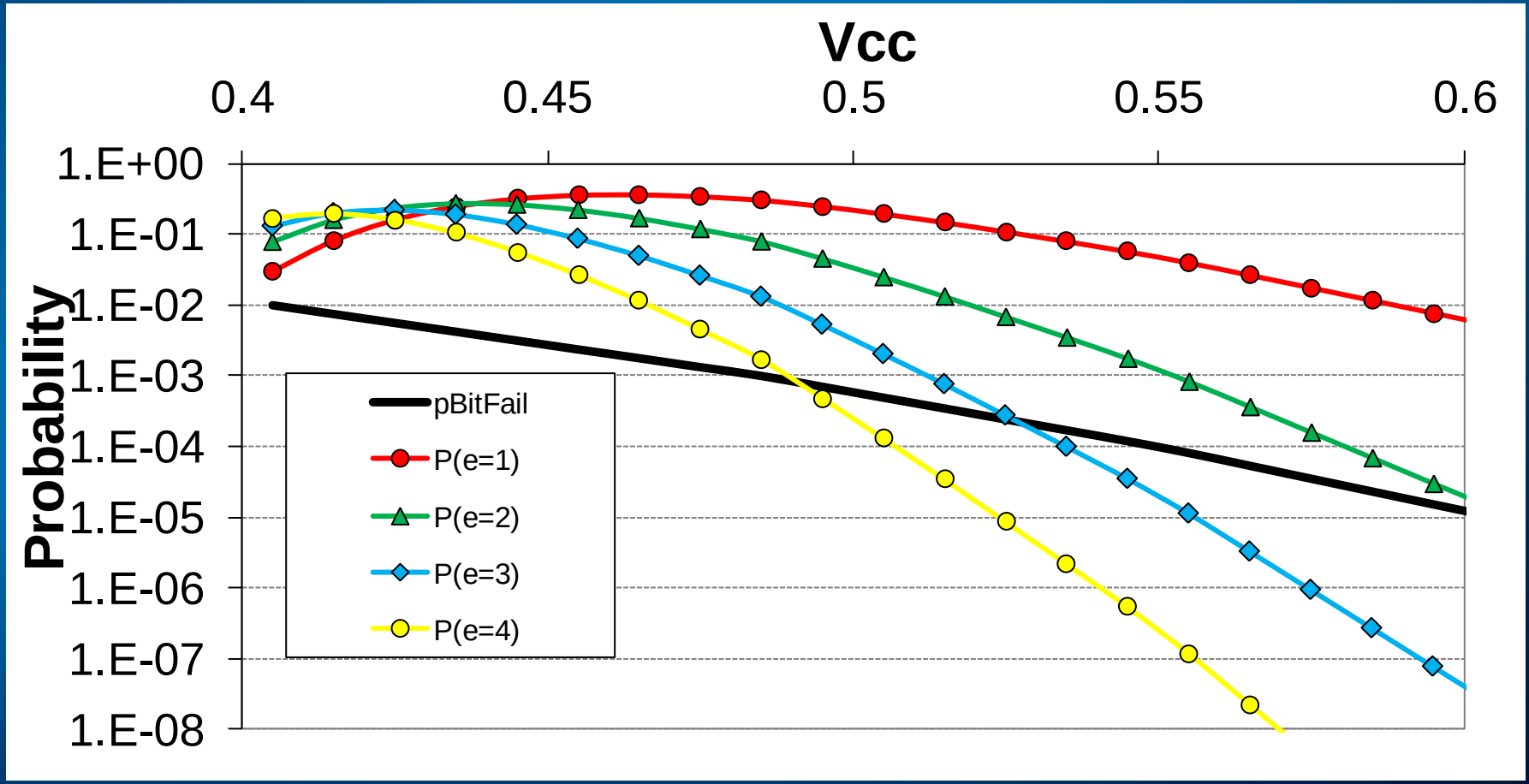
And There is Also User Experience…

- Not too many users will be happy with slower execution
- Mobile users like longer battery life, but they absolutely hate long wait times
  - Especially if the system is idle most of the time
  - Response time really matters when the system is active
- If voltage is too low, significant impact on user experience

Reliability Cost

- Getting too close to threshold significantly increases failures for individual transistors and storage elements
- Getting too close to tail of the distribution

Example: SRAM Bit and 64B Failures



Cost of Lower Reliability

- We need to make sure the whole chip works even if individual components fail
  - That is, we need to build reliable systems from unreliable components
- To improve reliability, we either increase redundancy or add other fault tolerance mechanisms
  - More power, area, $ cost

Simple Answer: TMR

- Basically, include three copies of everything, use majority vote
- Extremely high cost
  - More than 3x area increase
  - More than 3x power increase
- But even that might not be sufficient
  - Large structures may always fail, having three copies won't help
  - Need to do at transistor/cell level
  - Majority voting gets really expensive at that level

Another Answer: Error-Correcting Codes

- Applies only to storage or state elements
- At single-bit level, degenerates to TMR, but:
- Mostly area efficient if amortized across more bits
  - A small number of bits needed to detect/correct errors in large state elements
- But latency inefficient
  - Error correction requirements increase with larger blocks
  - SECDED on a 64B cache line may take a single cycle, but 4EC5ED might use ~ 15 cycles
- For logic elements, RAZOR-style circuits needed to reduce overhead

This Seems Too Hard…

- So why not relax our reliability requirements instead?

Approximate Computing to the Rescue

- If reliability is not absolutely required, then we can take a best-effort approach
- In other words
  - If something works correctly, great
  - If it doesn't, the incorrect outcome might be good enough
- Background:
  - Some applications don't care for 100% accurate computations
  - Example: Individual pixels on a large screen
  - We could take advantage by using NTC for them

But It Sounds Too Good To Be True…

- In reality, too many applications care about reliability
- And even applications that could tolerate errors need some code to be reliable
  - A pixel error on a bitmap is no big deal, but a pixel error in a compressed image (e.g., jpeg) causes too much noise
  - In a long sequence of computations, early computations need accuracy while later can tolerate errors
- Too much overhead to allow NTC selectively
  - Definitely needs programmer input
  - Could lead to too fine-grain control of reliability

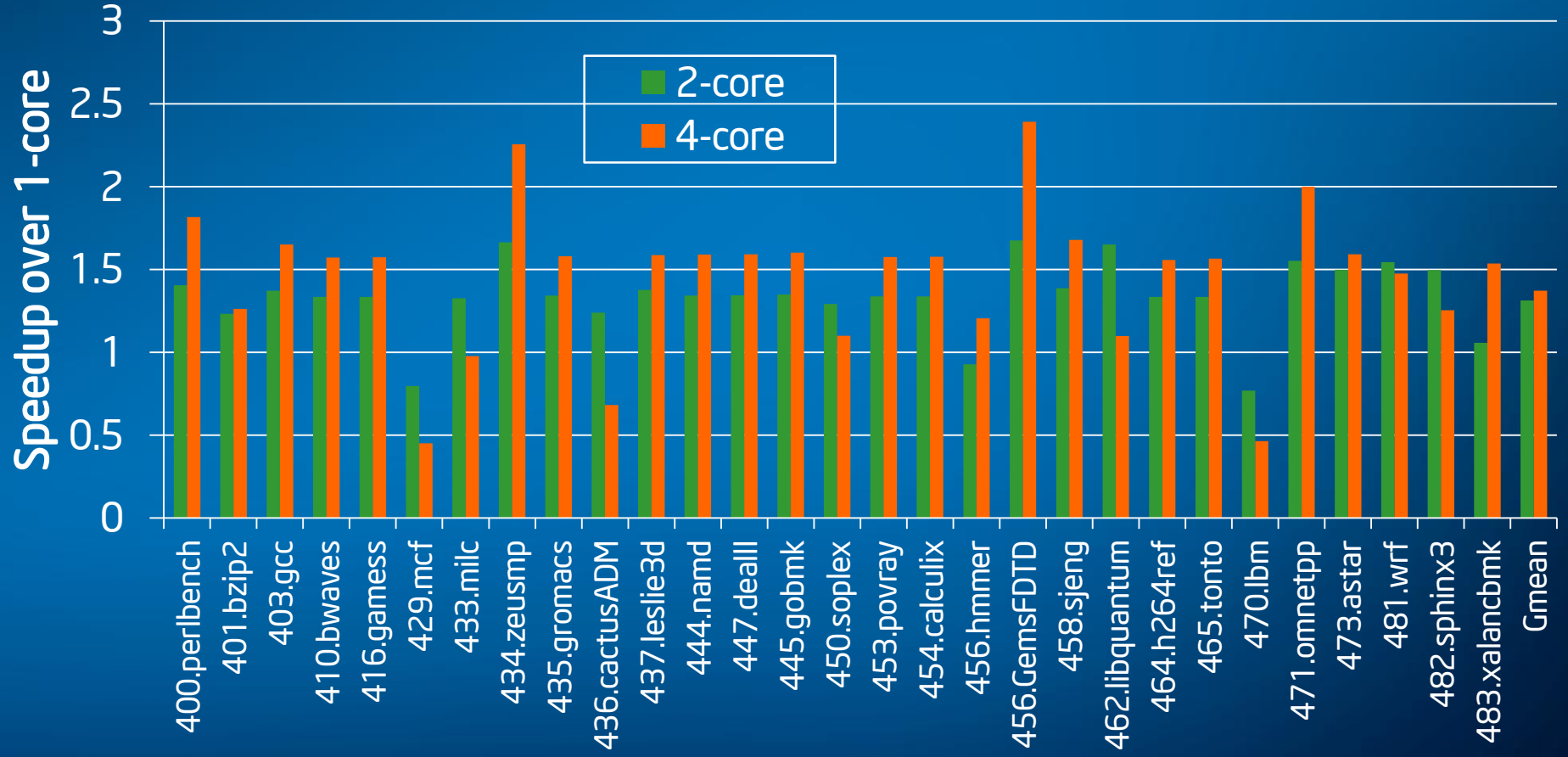
My Architectural Perspective

- Near-threshold computing is great if power savings outweigh latency and reliability cost
- But in many cases, cost is too great
- So we shouldn't give up on NTC, but only use it in places where it helps
- Or alternatively, we shouldn't get too close to threshold to the point where costs outweigh benefits
- Selective NTC requires architectural support

Case Study: Mixed-Cell Cache Design

- Optimize only part of cache for low (or near-threshold) voltage, using more reliable (bigger) cells
- Rest of cache uses normal cells
- During normal mode, all cache is active
- At low voltage, could only turn on reliable part
- Causes significant performance drawbacks

Speedup of Multi-Core over Single Core

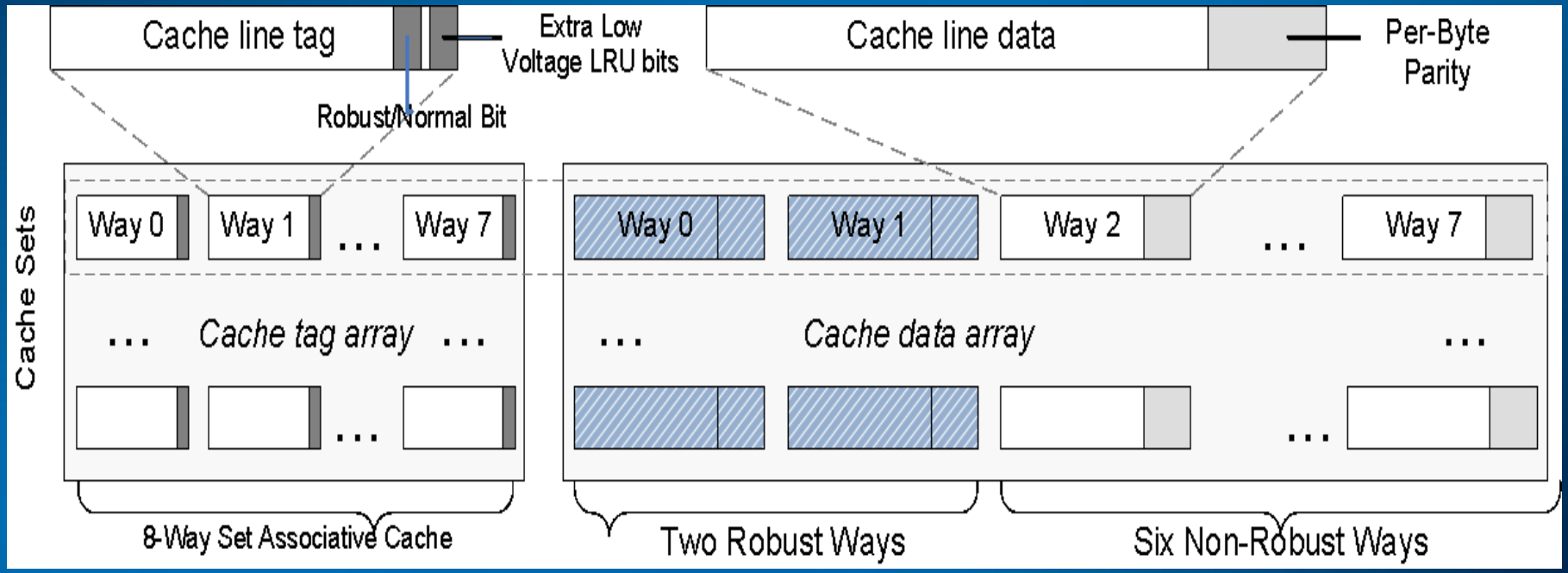


Compared to 1P, 2P is 31% better, 4P is 37% better

4P has Much Better Performance than 1P, But…

- Design is TDP-limited
  - To activate 4 cores, need to run at Vmin
  - Without separate power supplies, only robust cache lines will be active
  - 4P is where we really need the extra cache capacity for performance
- Mixed caches include robust cells that could run at low voltage, and regular cells that only work at high voltage
- Our Mixed-Cell Architecture:
  - All cache lines are active at Vmin
  - Architectural changes to ensure error-free execution

Mixed-Cell Cache Design

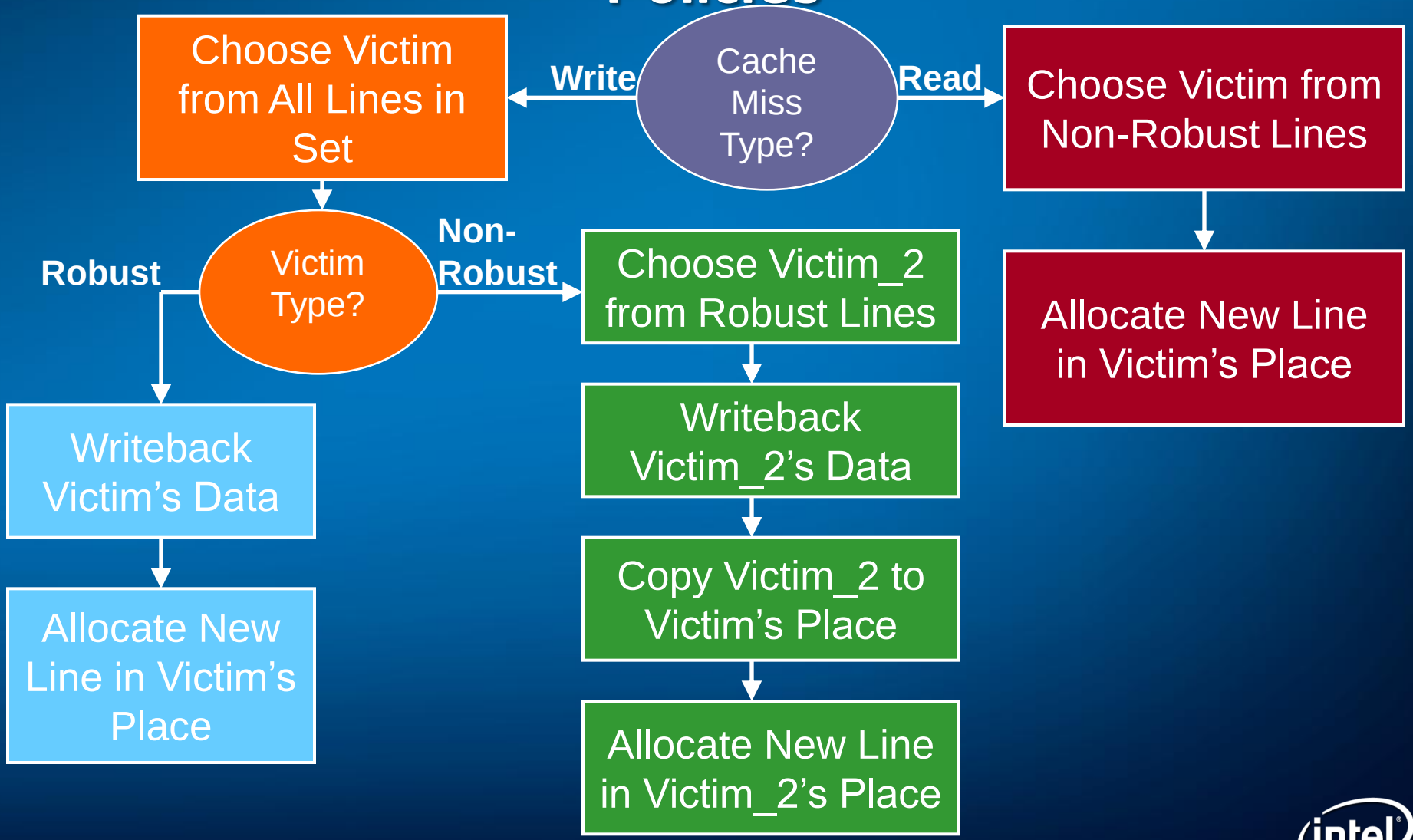


- Each cache set has two robust ways
- Modified data only stored in robust ways
- Clean data protected by parity

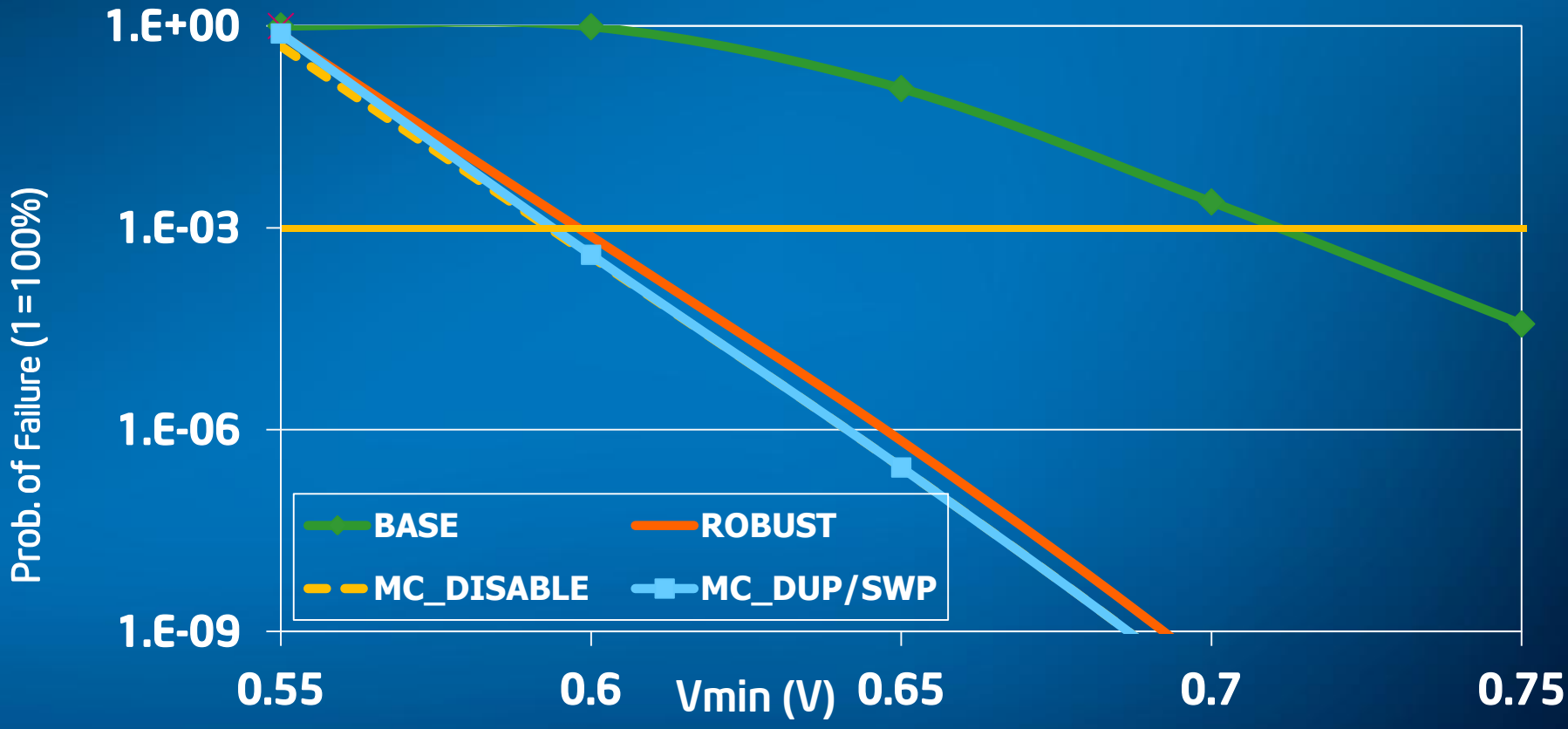
Mixed-Cell Architectural Changes

- Change cache insertion/replacement policy to allocate modified data only to robust ways
- What to do for Writes to a Clean Line?
  - **Writeback (MC_WB):** Convert dirty line to clean by writing back its data to the next cache level (all the way to memory)
  - **Swap (MC_SWP):** Swap newly-written line with the LRU robust line, and write back the data for victim line to next cache level
  - **Duplication (MC_DUP):** Duplicate modified line to another non-robust line by victimizing line in its partner way

Changes to Cache Insertion/Replacement Policies



Cache Vmin for Mixed-Cell Caches

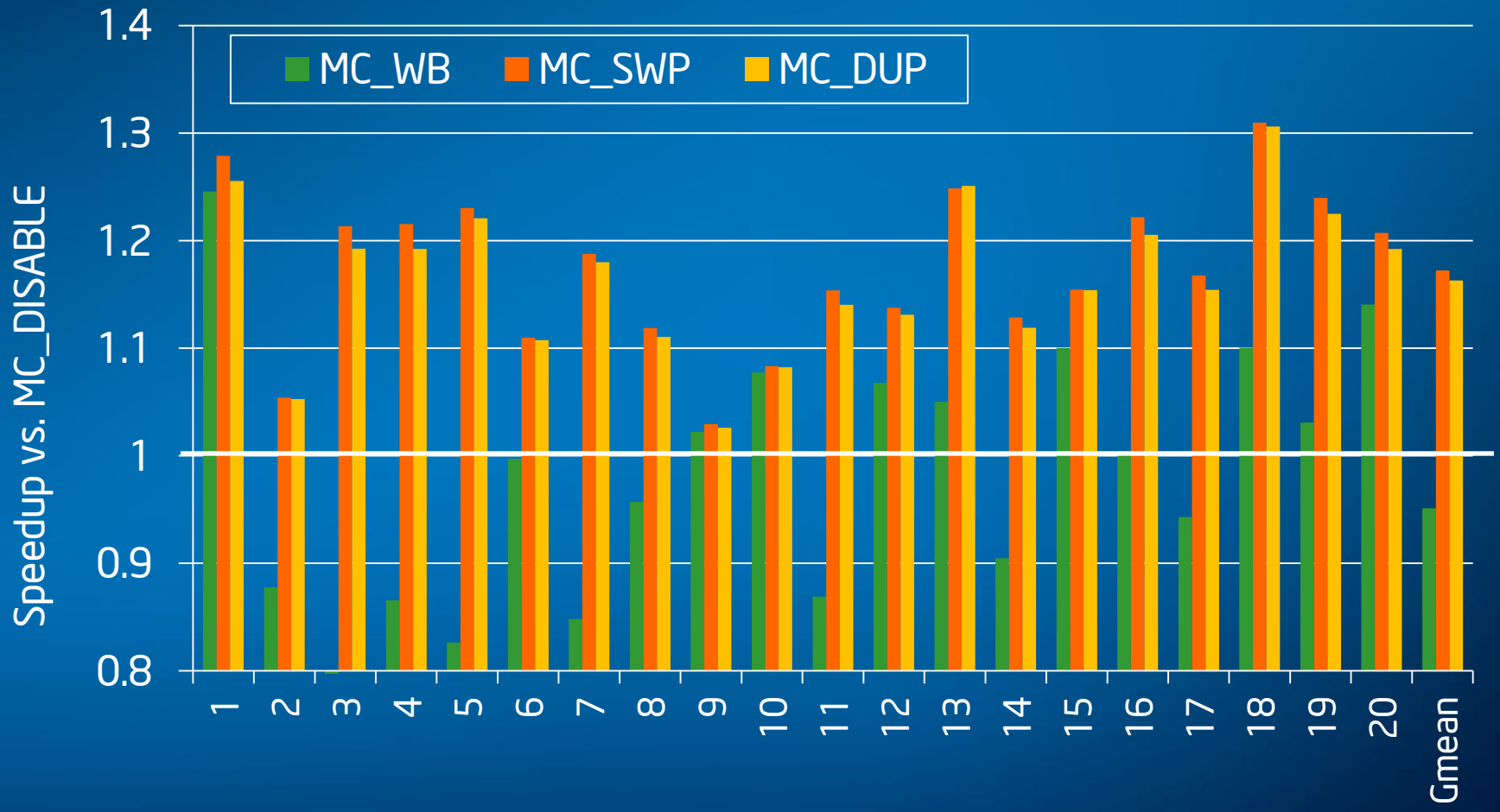


New MC_DUP and MC_SWP mechanisms are very close to building the cache with only robust cells (but much larger cache capacity)

Evaluation

- Used CMP$im
- Cache configuration based on current Intel mainline cores
- Compared our mechanisms to baseline and prior MC proposals
  - ROBUST: Cache only uses robust cells, much smaller capacity iso-area
  - MC_Disable: Only 1/4 of cache is operational at Vmin
- Used 4-program mixes from SPEC workloads

Multi-core (4P) Performance



Geomean 17% speedup for MC_SWP over MC_DISABLE

Mixed-Cell Cache Summary

- Philosophy: Only part of cache is reliable enough to operate at near-threshold
- A multi-core system (at Vmin) needs larger cache capacity
- Our mixed-cell architecture preserves cache capacity at Vmin
  - Improves performance
  - Reduces dynamic energy
- Could be extended to other parts of the memory hierarchy, and newer memory technologies

Case Study: VS-ECC

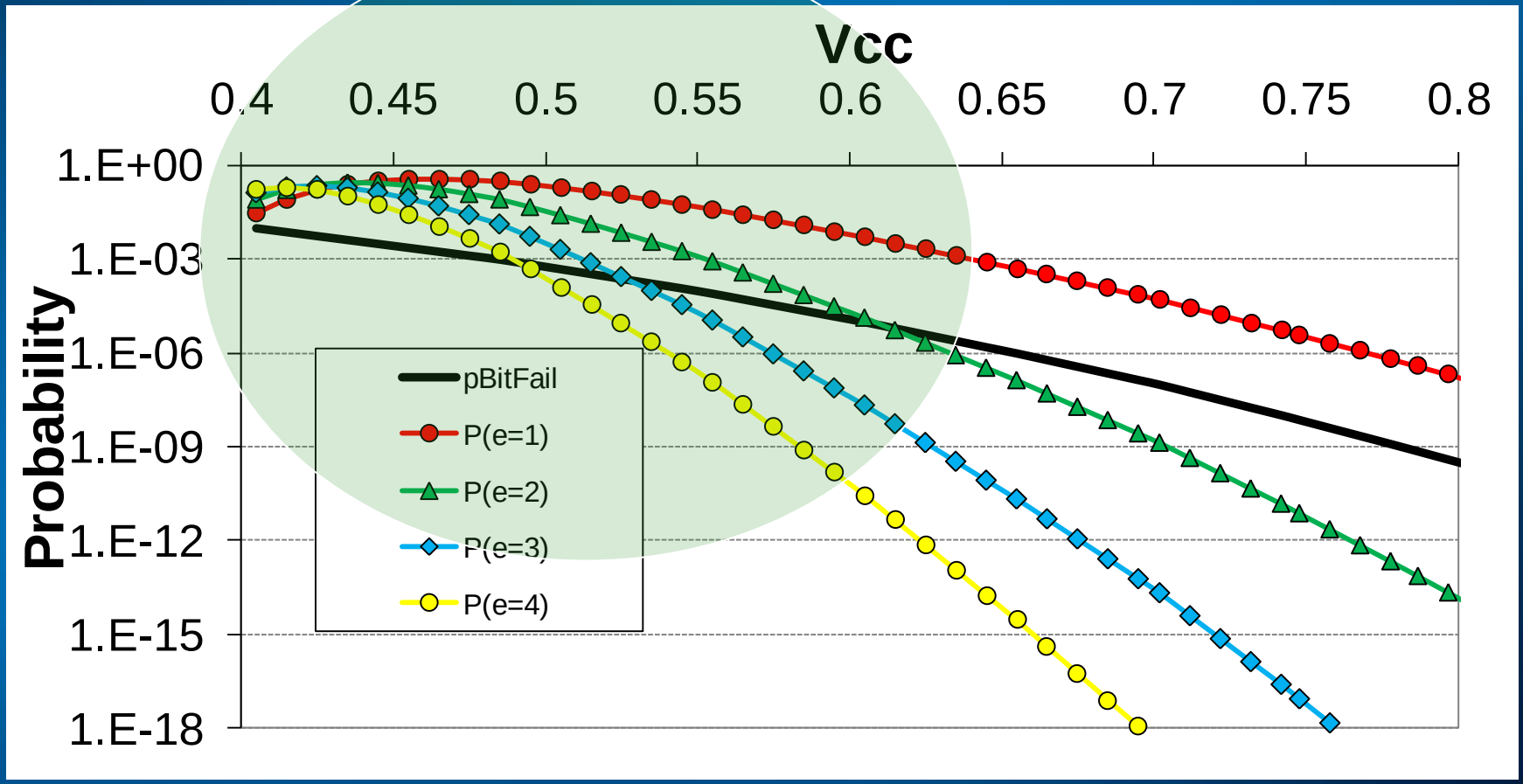
- Large caches and memories limit voltage scaling
  - Many cells fail at low voltages
  - Need to account for weakest cell
- Error-Correcting Codes (ECC) allow lower voltages by recovering from (multiple) failures
- Uniform ECC increases latency, power & area

➲ Our Proposal: Variable-Strength ECC (VS-ECC)

  - Better performance, power and area vs. uniform ECC
  - Allocates ECC budget to lines that need it
  - Online testing identifies lines needing more protection

VS-ECC Motivation

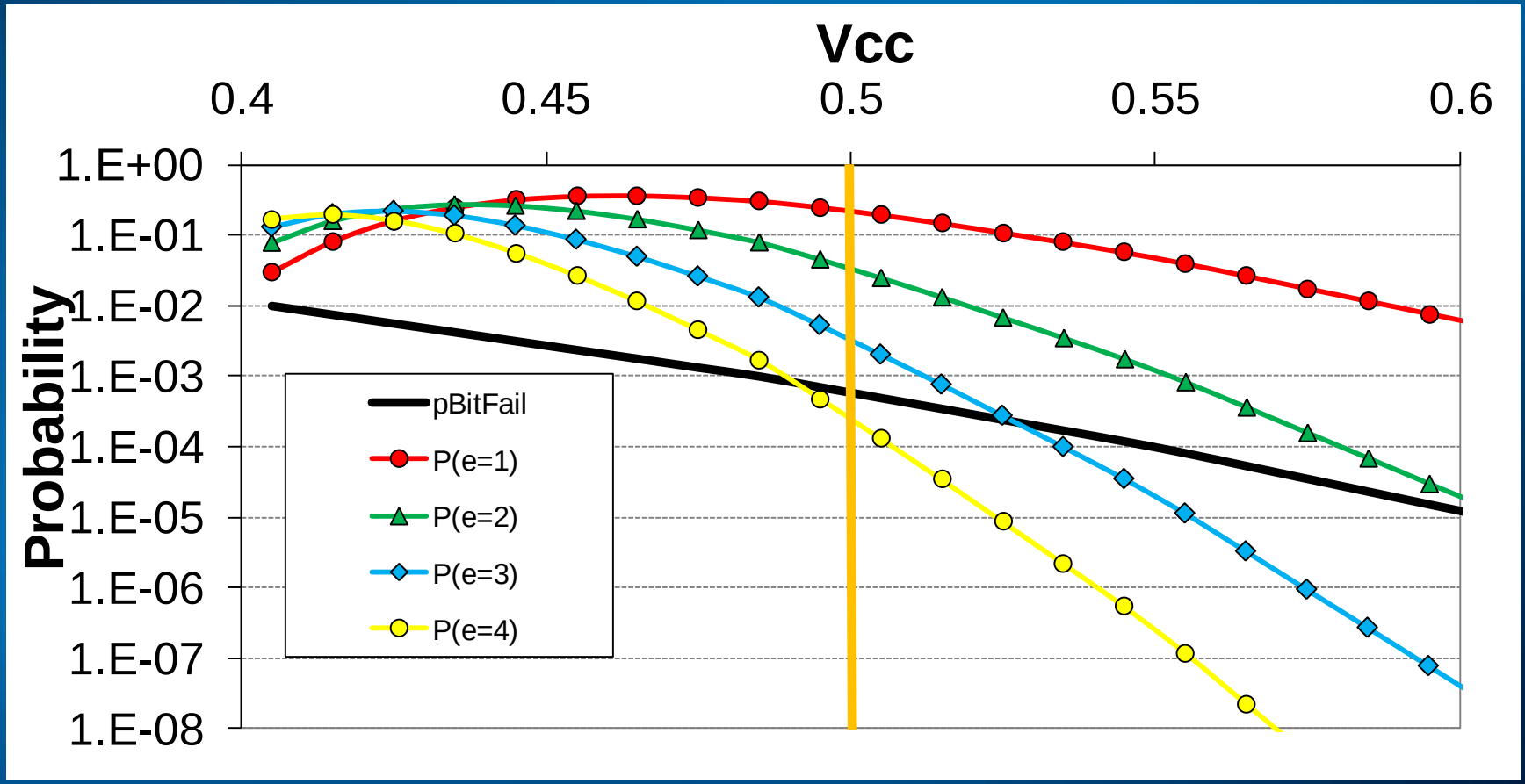
64B lines



- Most cache lines have 0-1 failures if we don't get too close to threshold
- But some lines (especially for large caches) have more failures

VS-ECC Motivation

64B lines



- Need a strong ECC code to protect worst lines
- Uniform ECC for all lines is expensive AND unnecessary

Prior Low Voltage Solutions

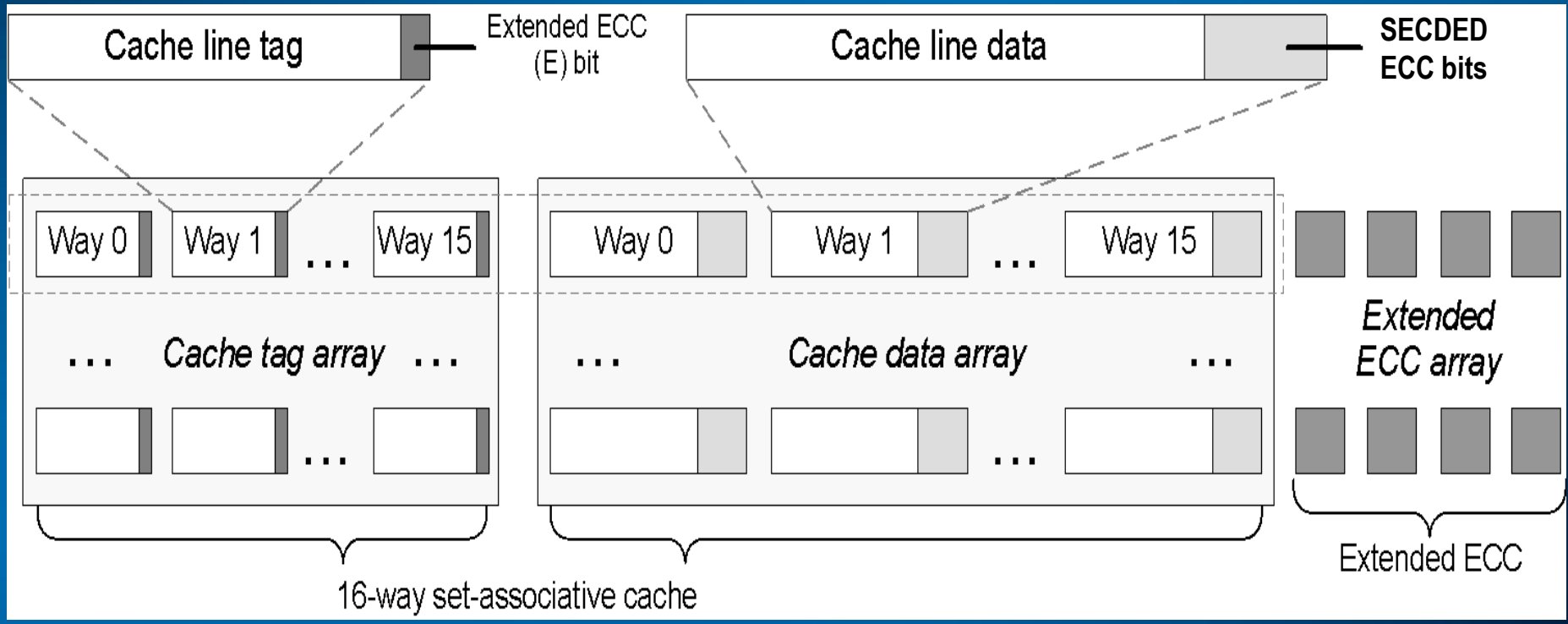
- Uniform-Strength Error Correction Codes
  - SECDED (Single Error Correction, Double Error Detection)
  - DECTED (Double Error Correction, Triple Error Detection)
  - Two-dimensional ECC: Kim et al., MICRO 07
  - Multi-bit segmented ECC (MS-ECC): Chishti et al., MICRO 09
- Architectural solutions for persistent failures
  - Word Disable: Wilkerson et al., ISCA 08, Roberts et al., DSD 07
  - Bit Fix: Wilkerson et al., ISCA 08
- Circuit Solutions: Larger cells, alternative cell designs

➲ All use same level of protection for all cache lines

Variable-Strength ECC (VS-ECC)

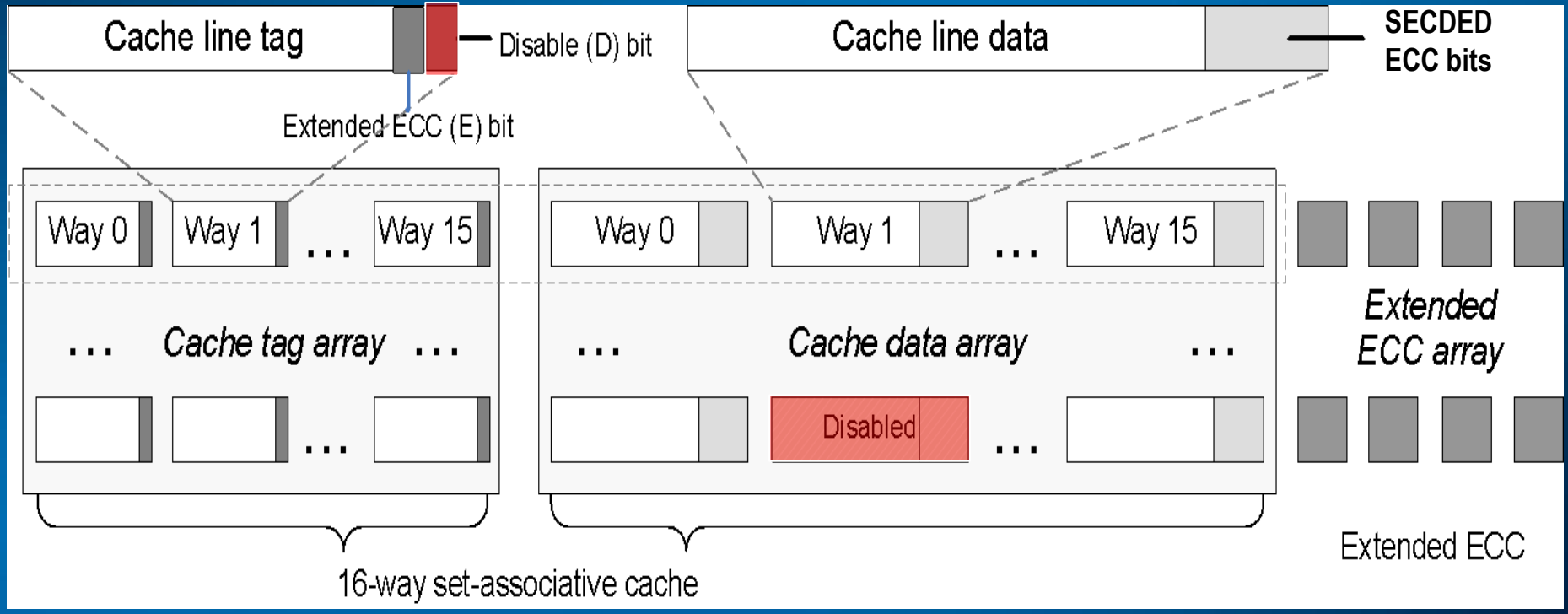
- Key idea: Provide strong ECC protection only for lines that need it
  - But still provide single-error correction for soft errors
- VS-ECC achieves lower voltage at minimum cost
- Three variations are explored
- Need to identify which lines need stronger protection

Design 1: VS-ECC-Fixed



- Fixed number of regular and extended ECC lines
- Regular lines protected by SECDED
- Extended ECC lines use 4-bit correction

Design 2: VS-ECC-Disable



- Add a disable bit to each line
- Lines with 3 or more errors are disabled
- Lines with zero errors use SECDED, 1-2 errors use 4-bit correction

Cache Characterization

- We need to classify cache lines based on their number of failures
- Manufacturing-time testing expensive & needs non-volatile on-die storage for fault map
- Proposal: Online testing on 1st transition to low voltage

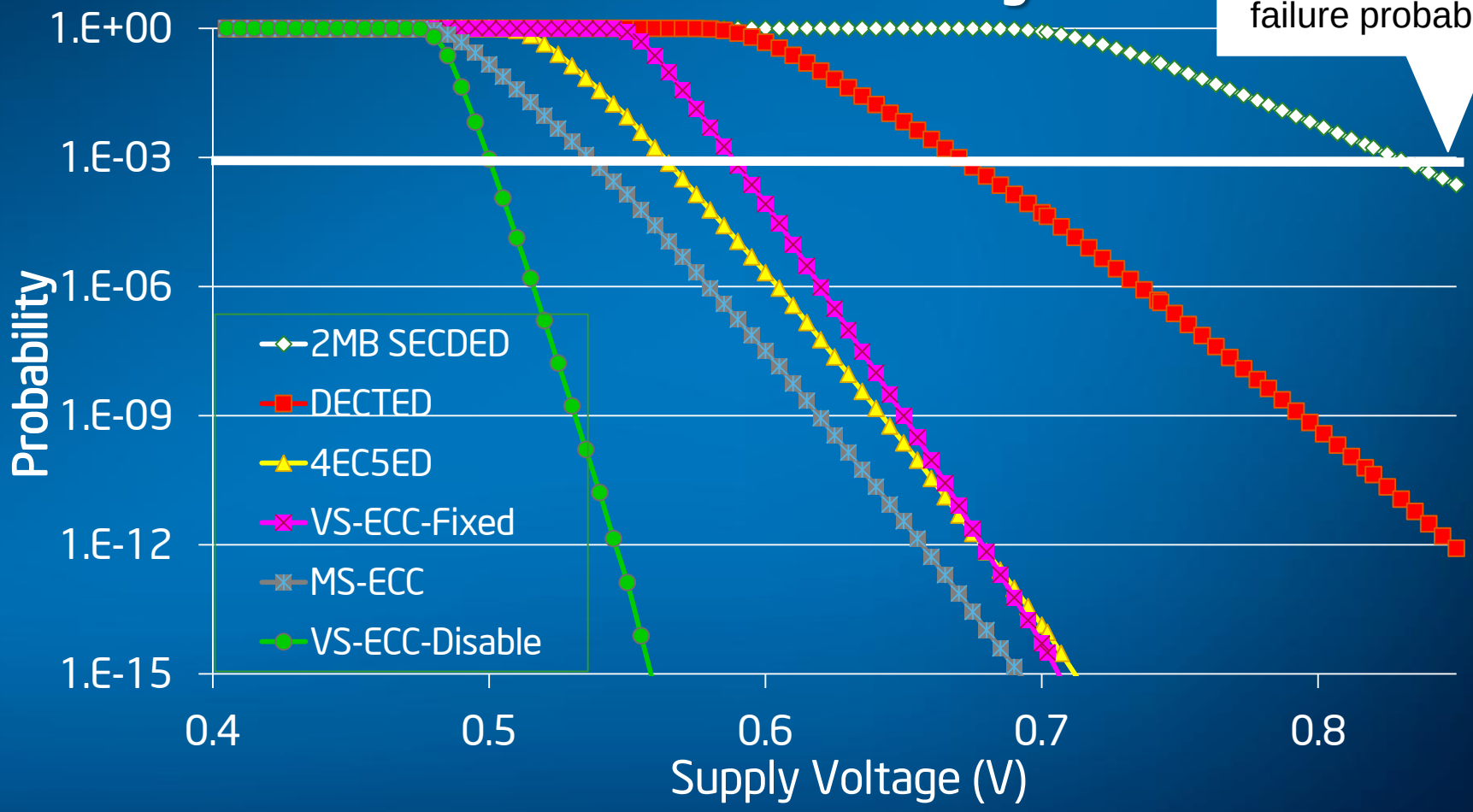
Online Testing at Low Voltage

- Cache is still functional during testing, but with reduced capacity
- Divide cache to working part (protected by 4-bit ECC) and part under test, then switch roles
- Use standard testing patterns, store error locations in tag
- Note: Not all VS-ECC designs require the same testing accuracy
  - Optimizing test time is an opportunity for future work

Simulated Configurations

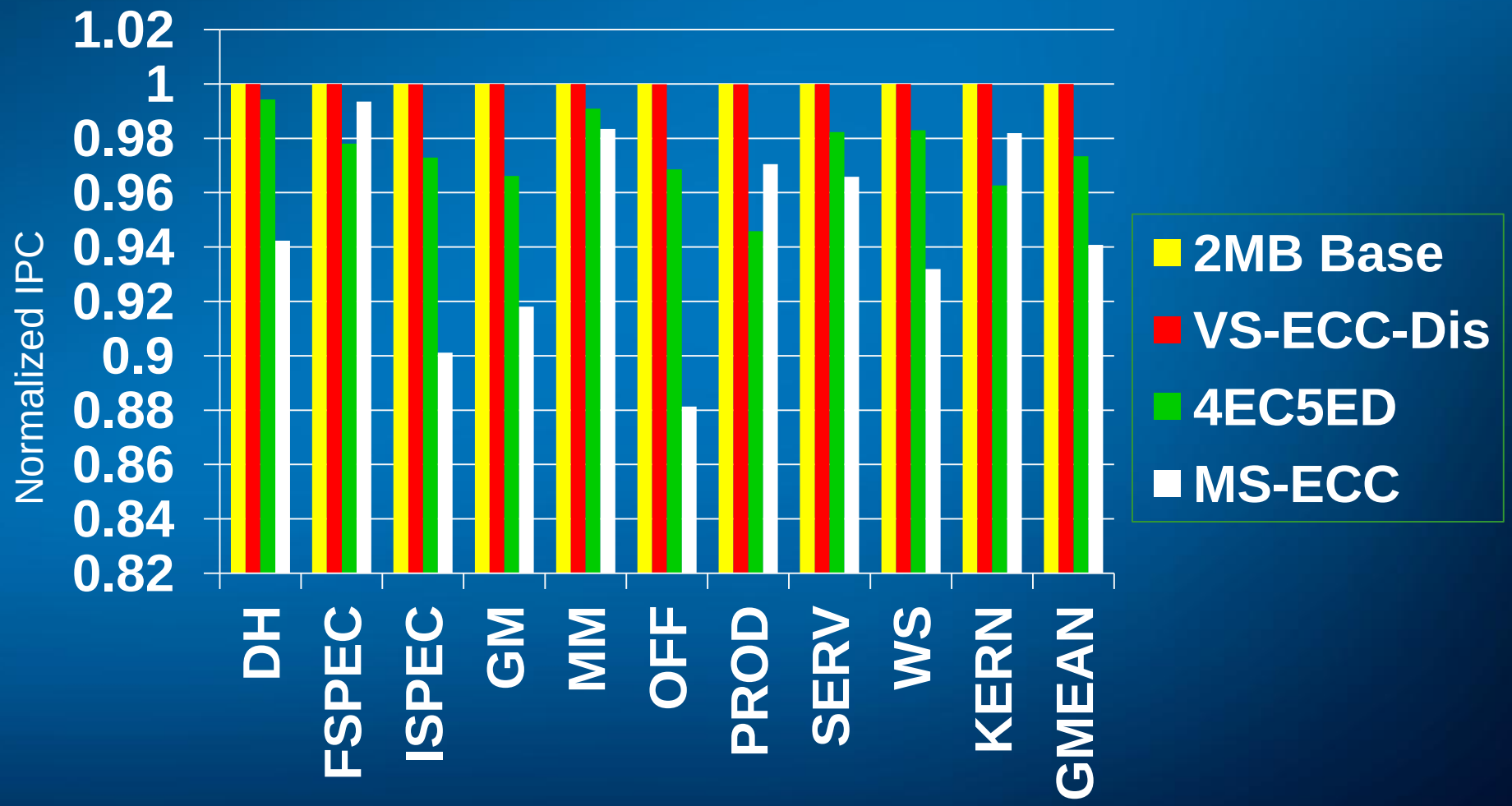
- Baseline
  - 2MB 16-way L2 (12 cycles), SECDED ECC to recover from non-persistent errors (1 cycle)
- Uniform-strength ECC
  - DECTED: 1 cycle, corrects one persistent error per line
  - 4EC5ED: 15 cycles, corrects up to three persistent errors per line
  - MS-ECC: 64-bit segments, 4 corrections/segment, corrects up to three persistent errors per segment, cache becomes 1MB 8-way
- Variable-strength ECC
  - VS-ECC-Fixed: 12 lines with SECDED (1 cycle), 4 with 4EC5ED (15 cycles)
  - VS-ECC-Disable: VS-ECC-Fixed+disable lines with ≥ 3 errors

Results: Reliability



- **VS-ECC has similar voltage scaling to 4EC5ED**
- **VS-ECC-Disable achieves lowest voltage**

Results: Performance at Low Voltage



- Similar IPC to baseline, better than uniform ECC

VS-ECC Summary

- Near-threshold computing needs strong ECC capability in large caches
- Uniform ECC techniques are expensive (performance, power, area)
- Variable-Strength ECC provides strong protection only to lines that need it
- VS-ECC + Line Disable is the most cost-effective mechanism
- But it really needs practical online testing mechanisms

Key Messages

- Near-threshold computing : Sometimes benefits outweigh costs, and some other times they don't
- It's better to use near-threshold computing selectively rather than for everything
- Alternatively, we should not get too close to threshold, only as long as benefits outweigh costs

Acknowledgments

- Samira Khan (Intel/CMU)
- Chris Wilkerson (Intel)
- Ilya Wagner (Intel)
- Zeshan Chishti (Intel)
- Jaydeep Kulkarni (Intel)
- Wei Wu (Intel)
- Shih-Lien Lu (Intel)
- Daniel Jiménez (Texas A&M)
- Nam S. Kim (Wisconsin)
- Hamid Ghasemi (Wisconsin)