# Modeling Bounds on Migration Overhead for a Traveling Thread Architecture

Patrick A. La Fratta, Peter M. Kogge
Department of Computer Science and Engineering
University of Notre Dame
April 23, 2010

# Outline

- Problem Description
- Background
  - Heterogeneous Multicore
  - Control Speculation
  - Migrant Computations
- Objectives
- Application Characterization
- Design: PAM Architecture & Thread Migration

# Outline

- Problem Description
- Background
  - Heterogeneous Multicore
  - Control Speculation
  - Migrant Computations
- Objectives
- Application Characterization
- Design: PAM Architecture & Thread Migration
  - **Parallelization ↔ Locality Exploitation**

Data access latency places bound on parallelization

# Outline

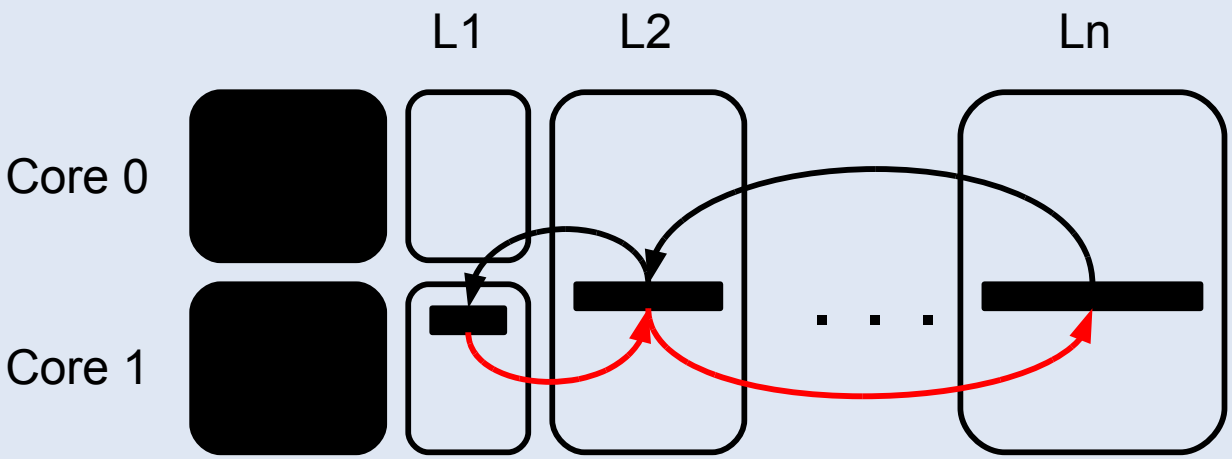
- Problem Description
- Background
  - Heterogeneous Multicore
  - Control Speculation
  - Migrant Computations
- Objectives
- Application Characterization
- Design: PAM Architecture & Thread Migration
  - **Parallelization ↔ Locality Exploitation**
- Evaluation: Analytical Modeling
- Results and Discussion

# Problem Description

- Conventional caching

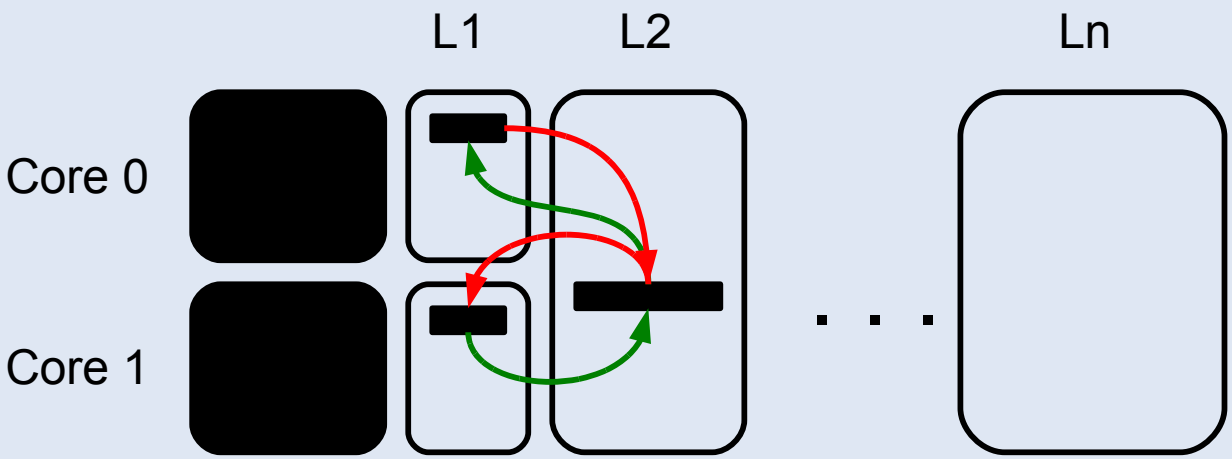
# Problem Description

- Conventional caching
  - Moving lines to/from upper/lower caches is expensive



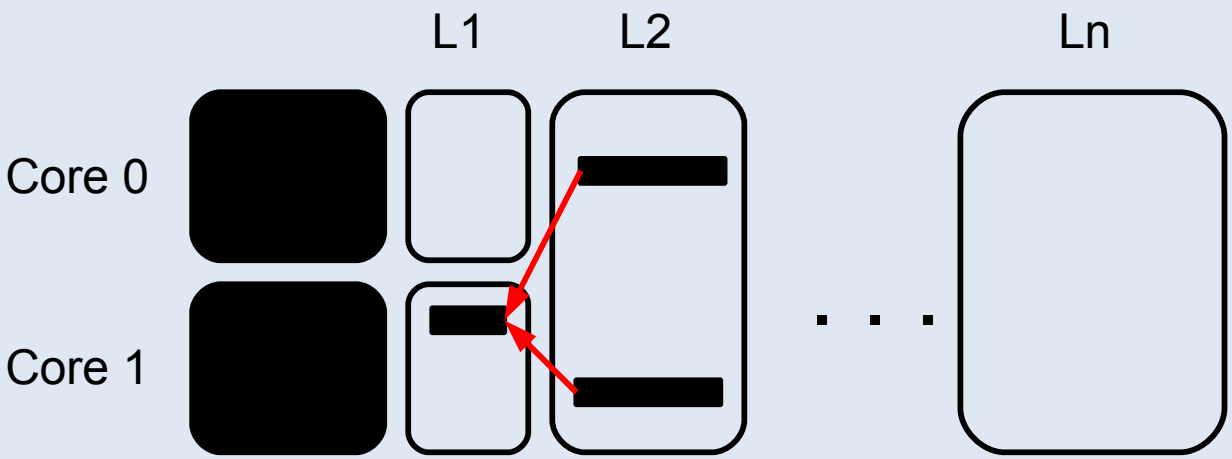
# Problem Description

- Conventional caching
  - Moving lines to/from upper/lower caches is expensive
  - High overhead of ”ping-ponging” to maintain coherence



# Problem Description

- Conventional caching
  - Moving lines to/from upper/lower caches is expensive
  - High overhead of ”ping-ponging” to maintain coherence
  - Cache pollution and conflicts



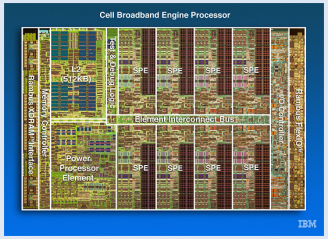
# Problem Description

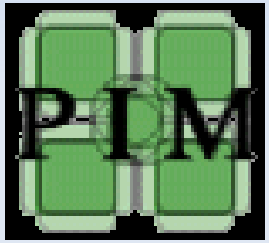
- Conventional caching
  - Moving lines to/from upper/lower caches is expensive
  - High overhead of ”ping-ponging” to maintain coherence
  - Cache pollution and conflicts
- Parallelization
  - Parallel programming has proven to be difficult
  - Compiler and µArch have increased responsibility

# Background

<u>Architecture</u>
Heterogeneous Multicore, PIM

Cell BE



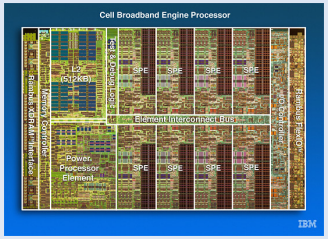



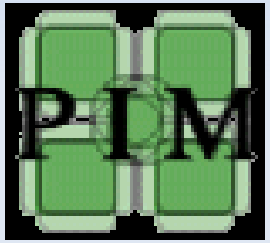

<u>Compilers</u>
Thread-level speculation

# Background

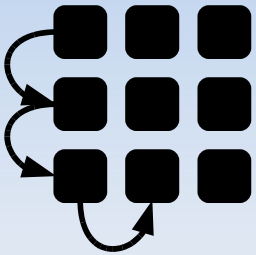
Architecture
Heterogeneous Multicore, PIM

Cell BE




Migrant Computation

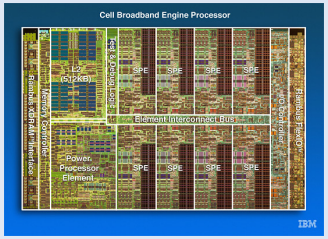


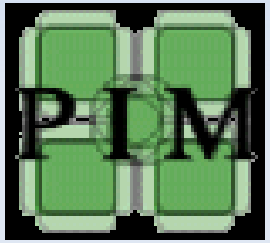
Compilers
Thread-level speculation

# Background

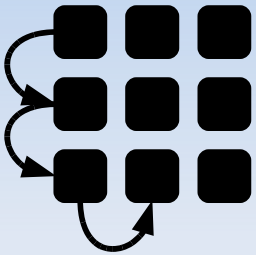
Architecture
Heterogeneous Multicore, PIM

Cell BE



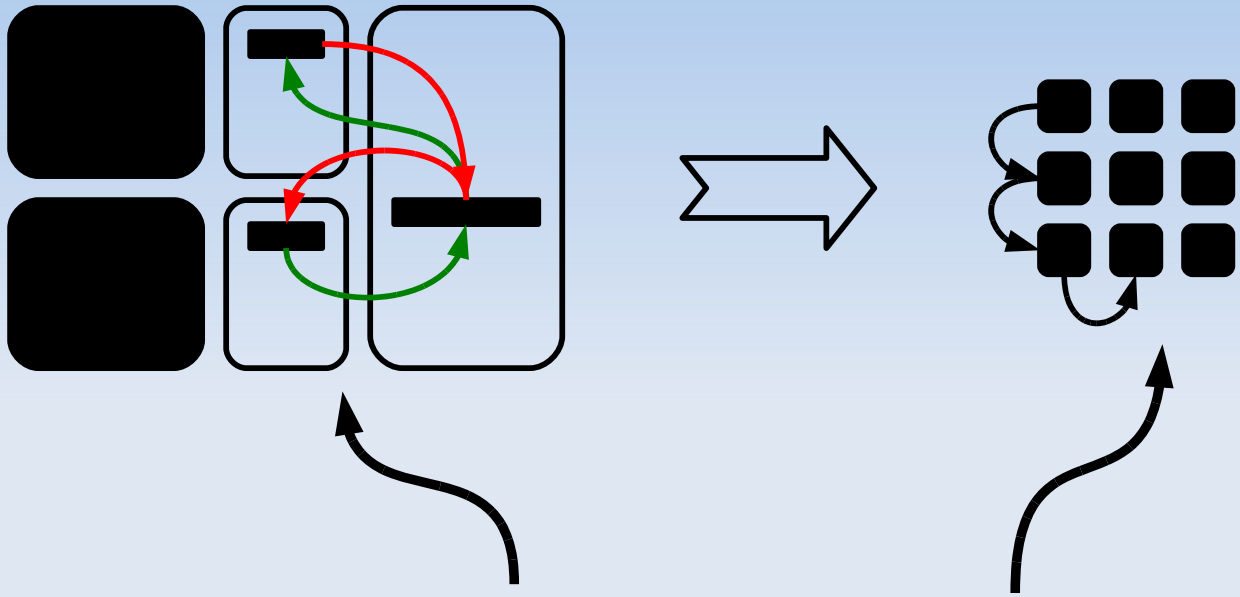

Migrant Computation



Compilers
Thread-level speculation

Passive/Active Multicore Architecture executing Traveling Threads

# Background: Migrant Computation



- Shifting traffic from cache line movement to thread migration.
- How much overhead due to computational migration can we tolerate to match performance of conventional architecture?

# Objectives

## I. Application Studies

- Parallelization mechanism: aggressive speculation
  - Motivation for design:
    - Critical Paths
      - Branch-critical path
      - ”Graph tails”
    - Memory Wall
  - Characterize parallelism available in applications

## II. Design Evaluation

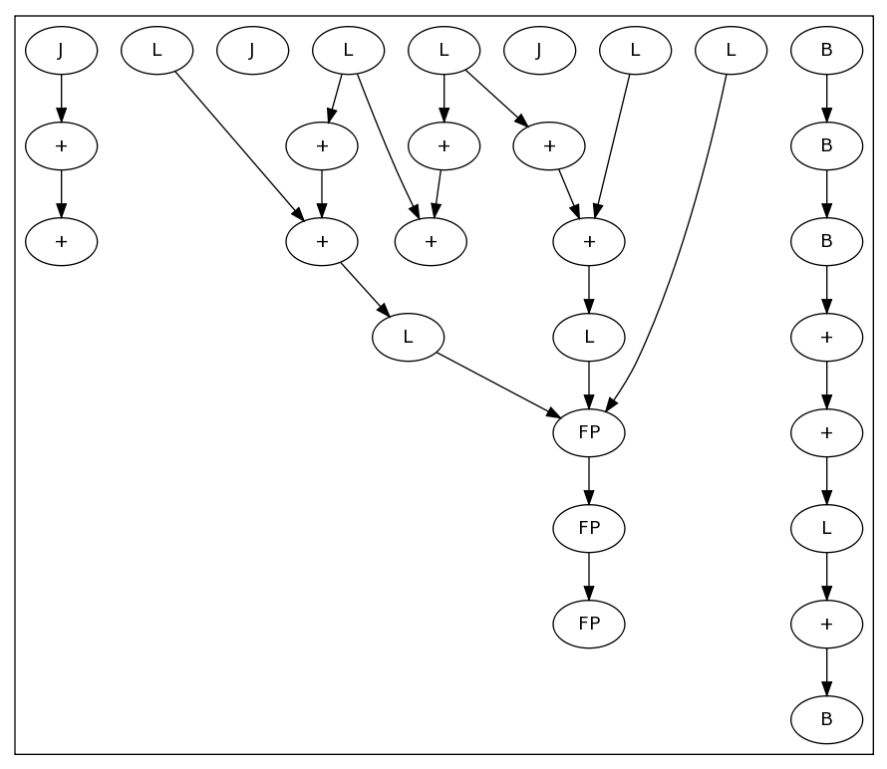
- Analytical performance models for comparative evaluation
  - Introduce metric of *instructions per migration* (IPM)
- **Break-even plane btw migration and conventional caching**

# Application Characterization

## ADAGs

BB 0-3

- Assume execution of only one speculative path at a time.




- Nodes/Edges
- Regions/Subregions
- ASAP Scheduling

Subregion Size = 4

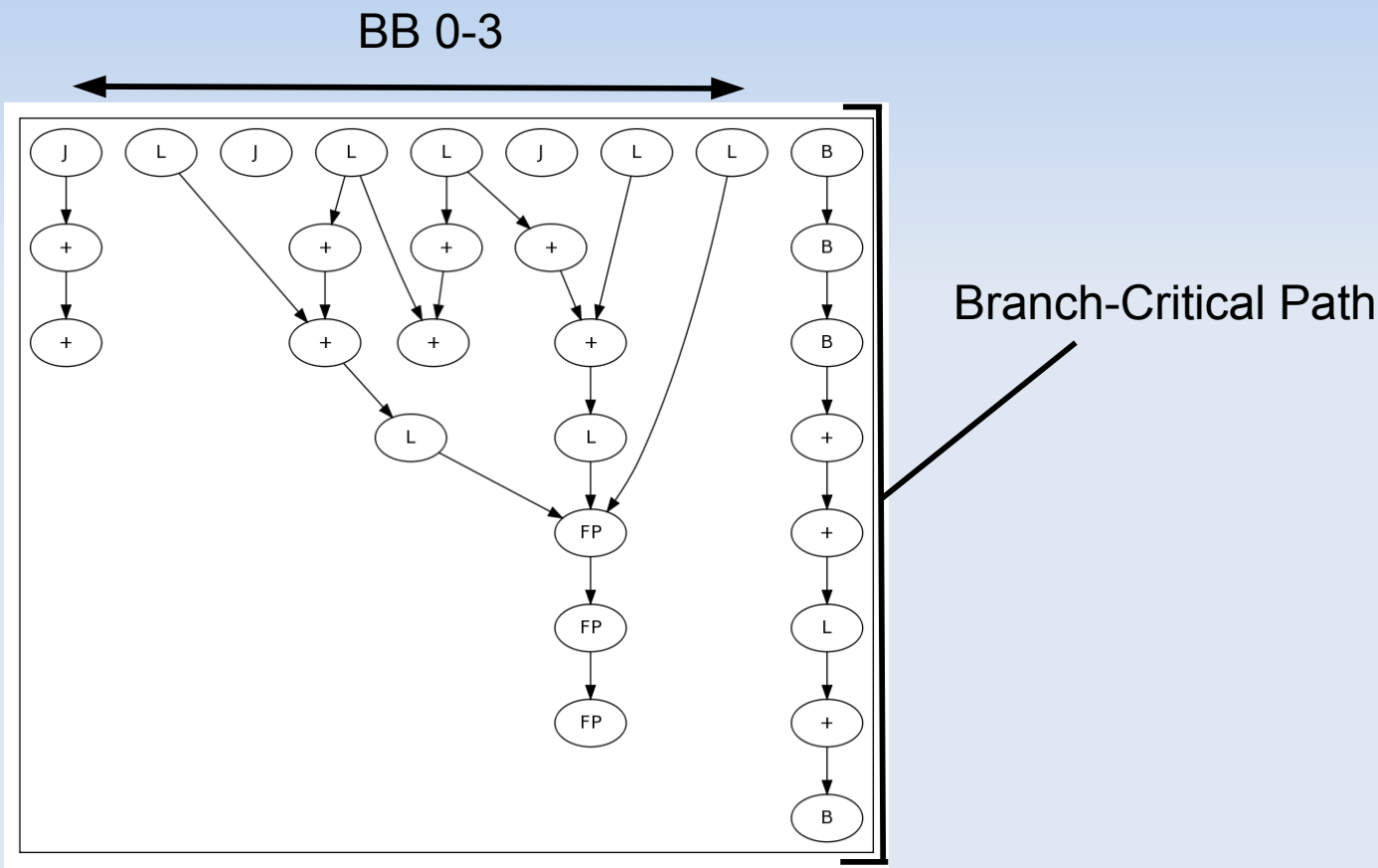
Key - B: Branch, L: Load, S: Store, +: ALU, FP: Floating Point, J: Jump

Example taken from alegra.4B.hemipen in Sandia-FP.

# Application Characterization: Critical Paths

## Branch-Critical Paths

- Assume execution of only one speculative path at a time.




Subregion Size = 4

Key - B: Branch, L: Load, S: Store, +: ALU, FP: Floating Point, J: Jump

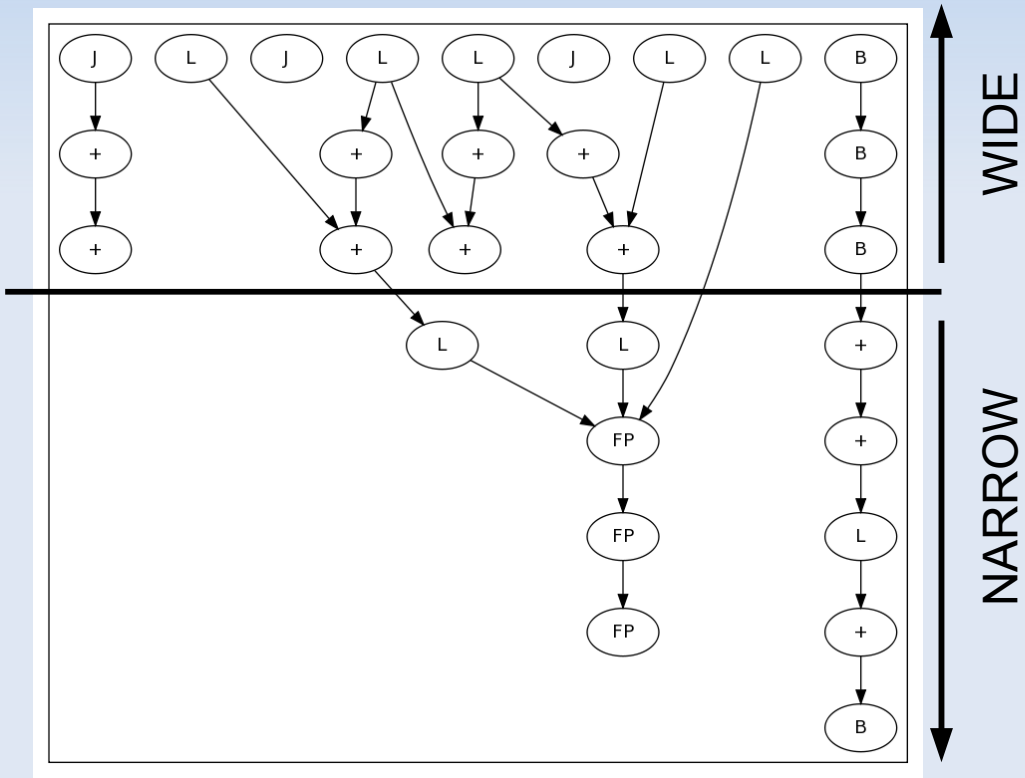
Example taken from alegra.4B.hemipen in Sandia-FP.

# Application Characterization

## ADAG Tails

BB 0-3

- Assume execution of only one speculative path at a time.



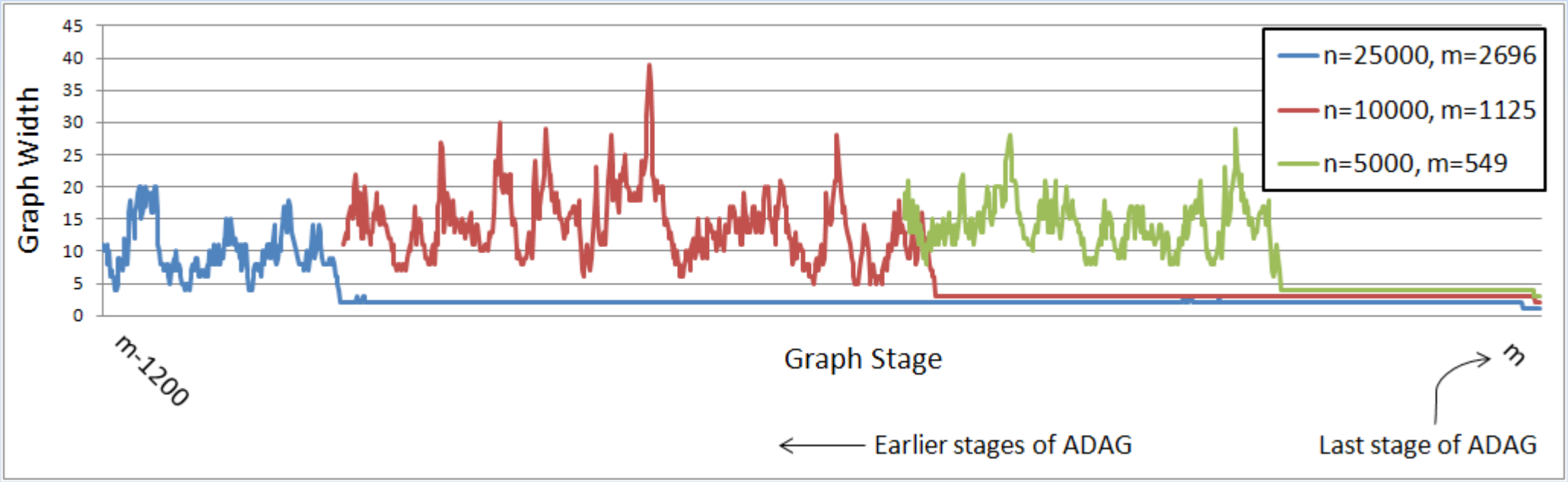

Subregion Size = 4

Key - B: Branch, L: Load, S: Store, +: ALU, FP: Floating Point, J: Jump

Example taken from alegra.4B.hemipen in Sandia-FP.

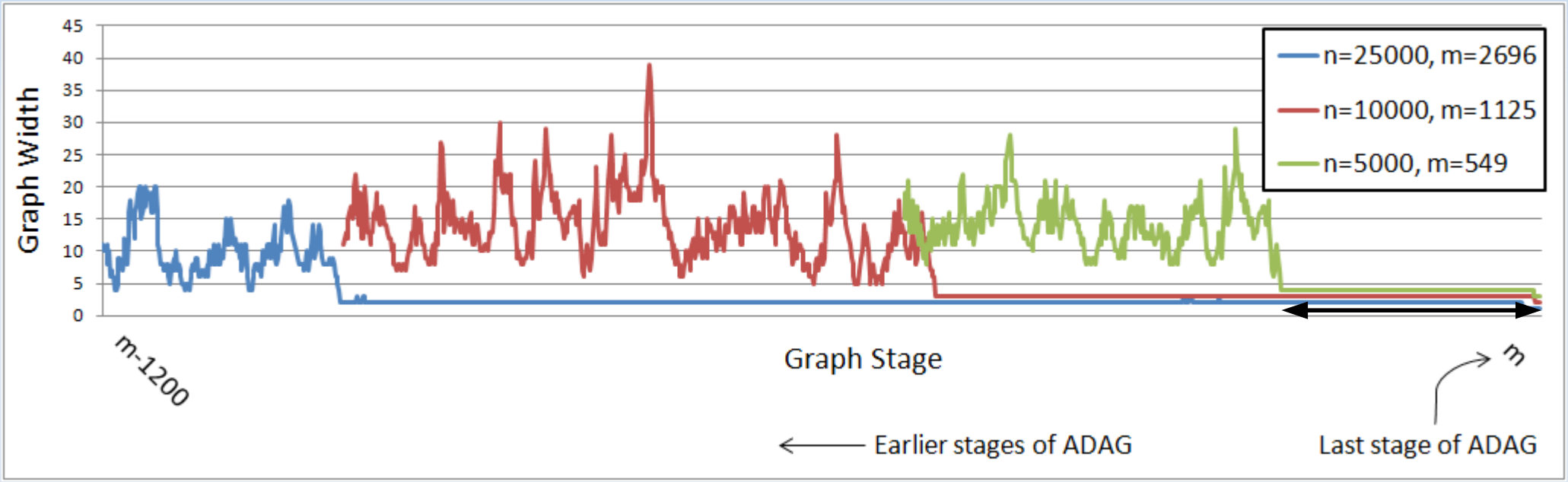
# Application Characterization: Critical Paths

## ADAG Tails



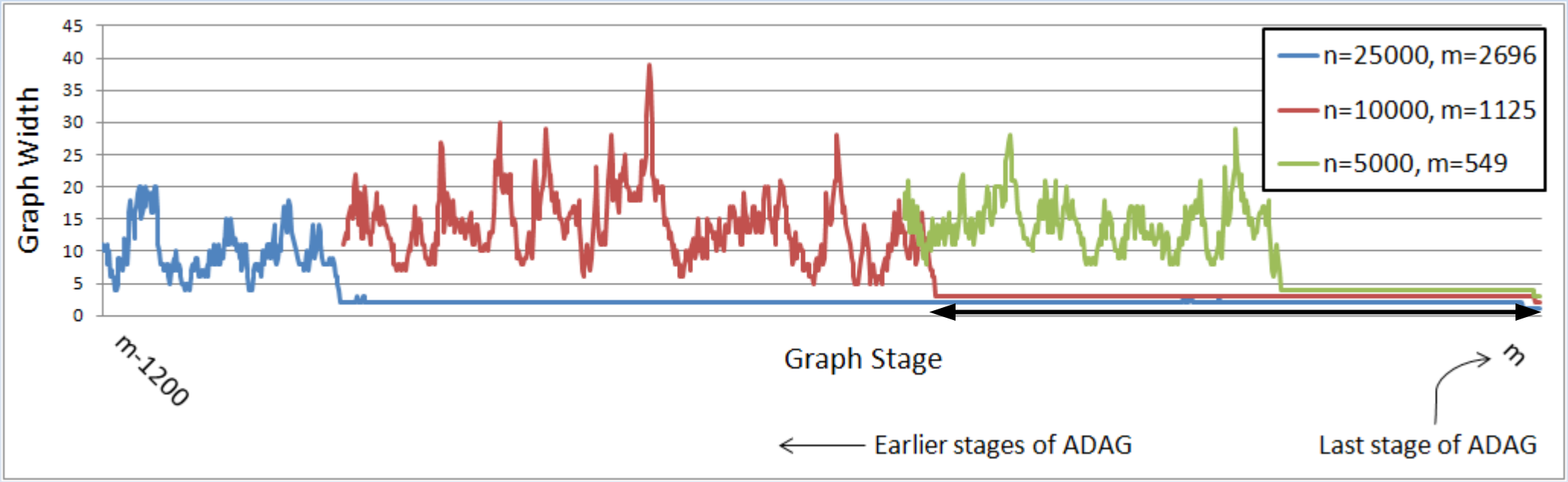
# Application Characterization: Critical Paths

## ADAG Tails



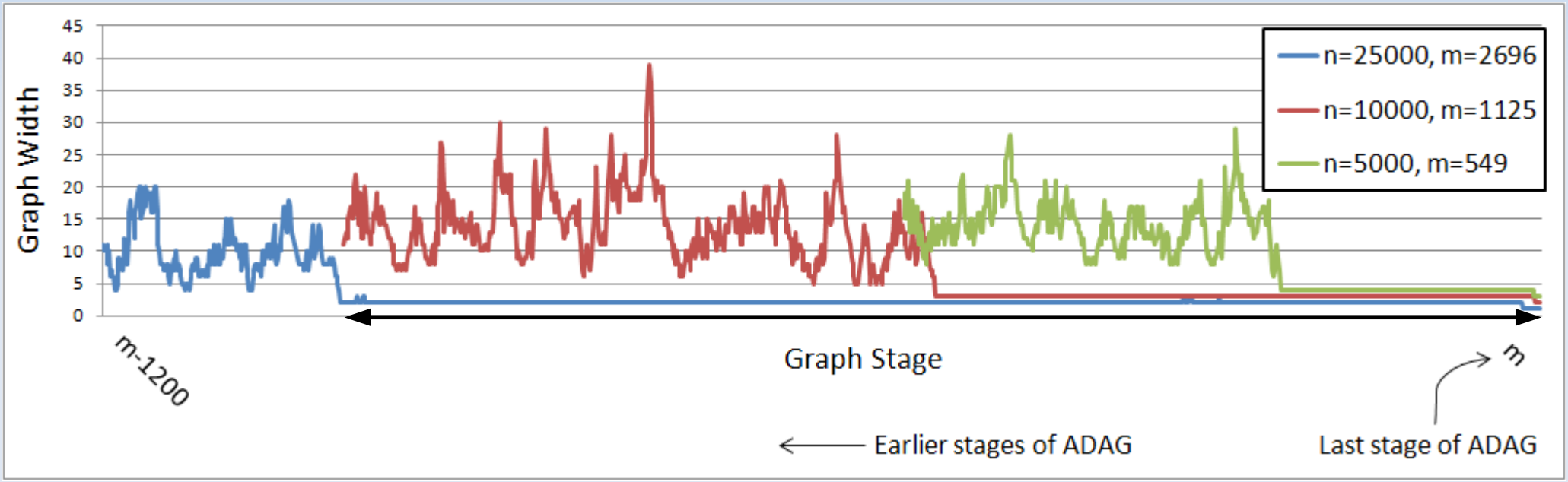
# Application Characterization: Critical Paths

## ADAG Tails



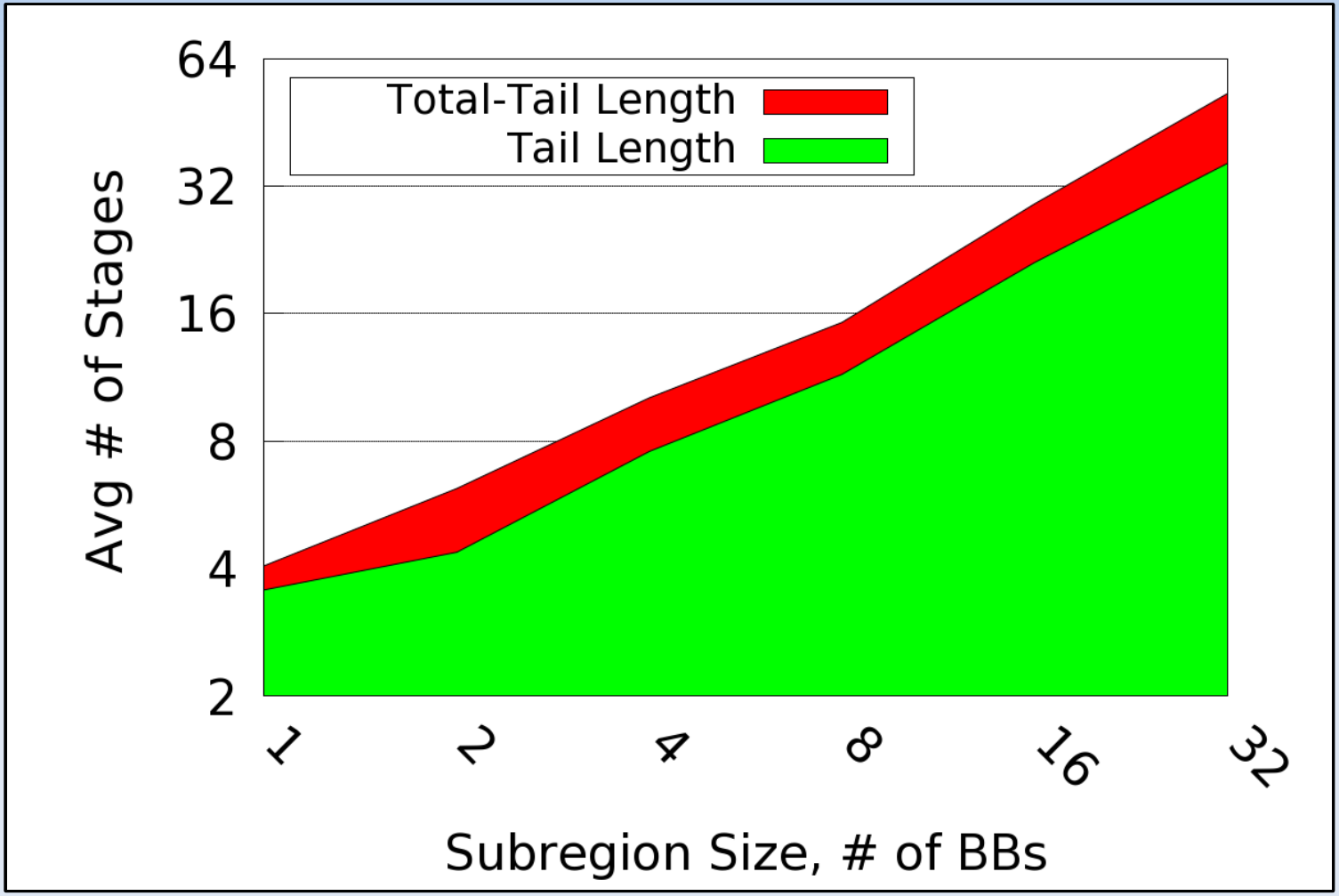
# Application Characterization: Critical Paths

## ADAG Tails



# Application Characterization: Critical Paths

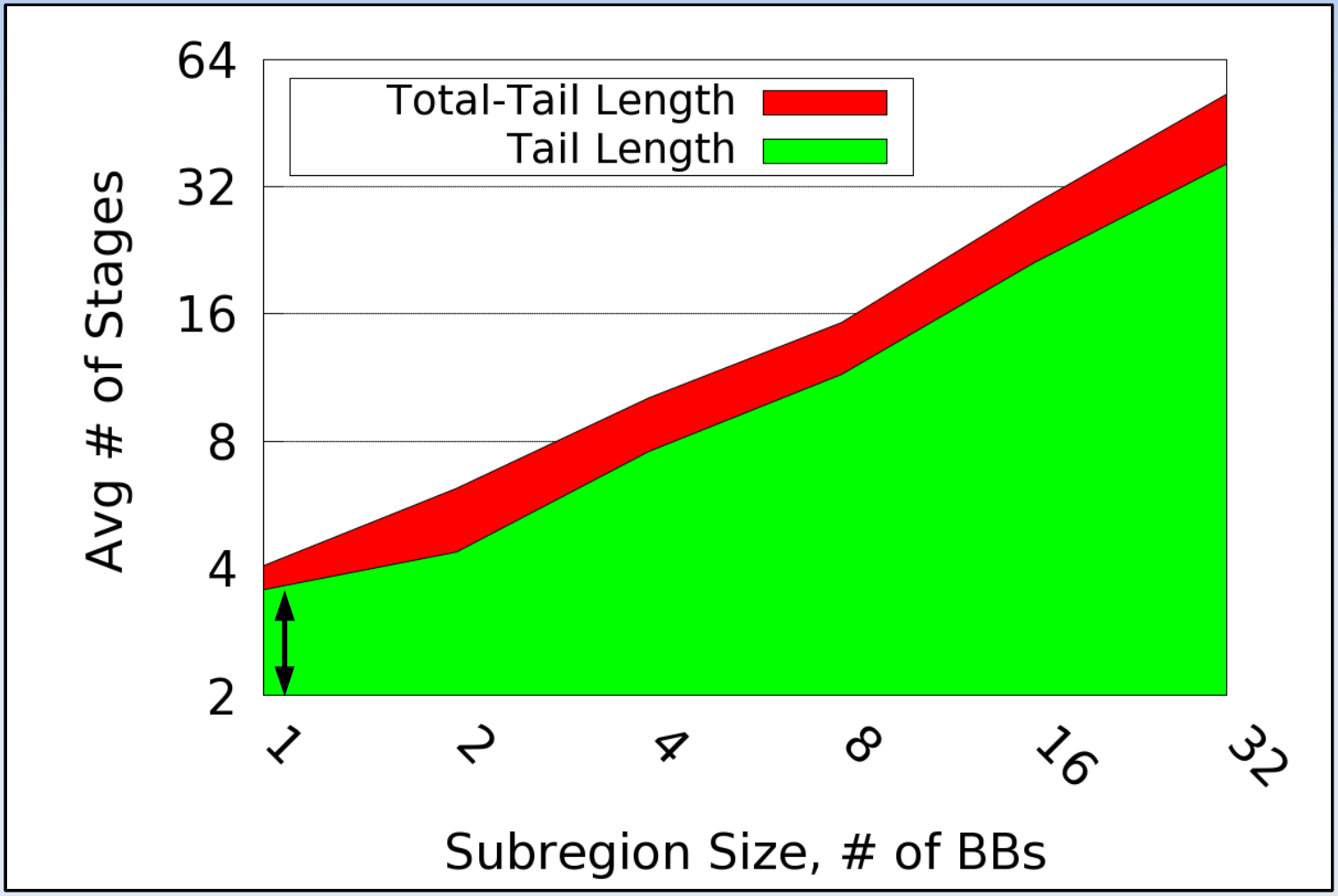
## ADAG Tails




179.art.1.1b
(SPEC-FP)

# Application Characterization: Critical Paths

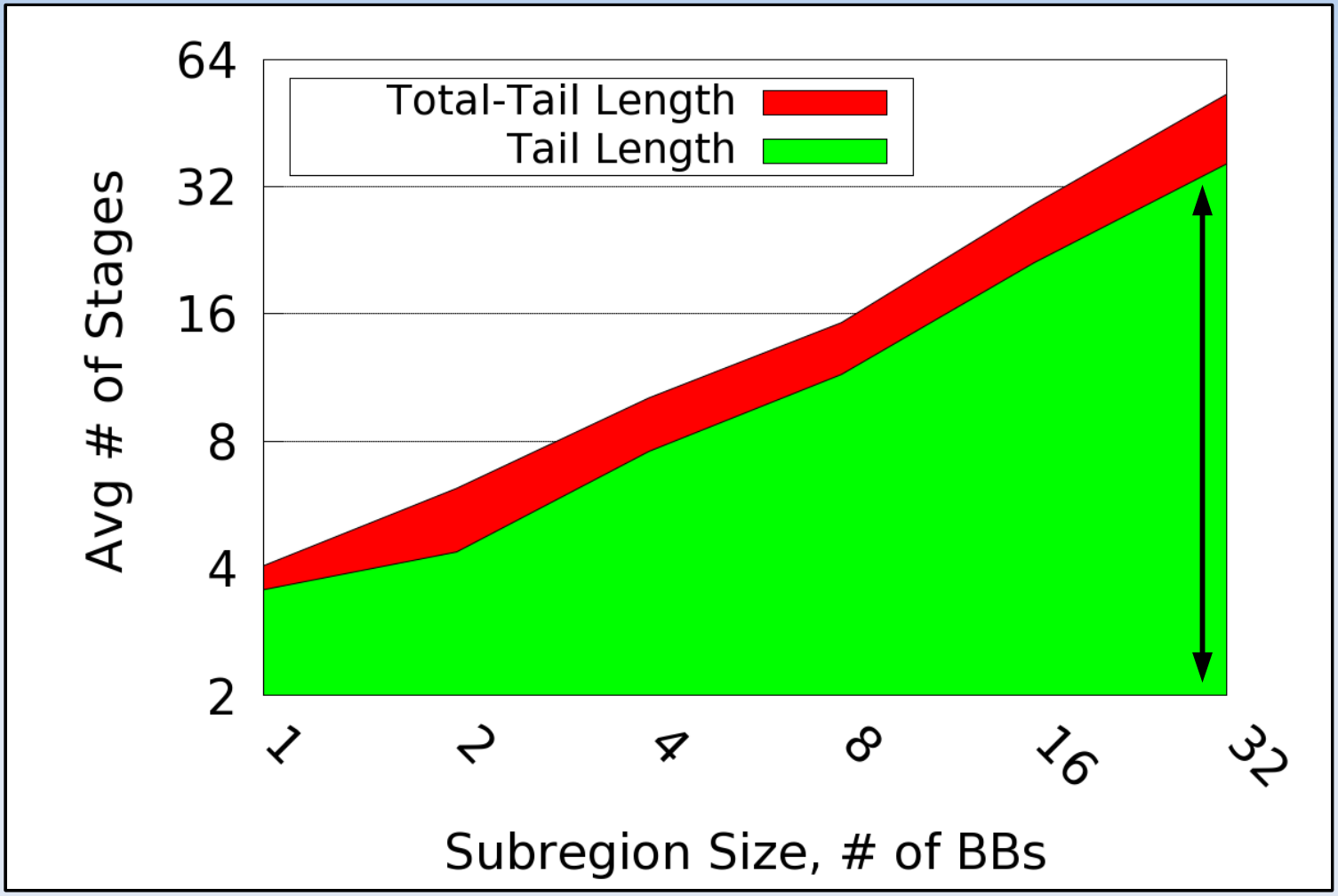
## ADAG Tails




179.art.1.1b
(SPEC-FP)

# Application Characterization: Critical Paths

## ADAG Tails




179.art.1.1b
(SPEC-FP)

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

$$IPC_{max} = \frac{I_{tot}}{L_{\text{asap}}}$$

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

Memory latency-bounded IPC.

$$IPC_{max} = \frac{I_{tot}}{L_{\text{asap}}}$$

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

Cycle Time
(0.25 ns)

$$IPC_{max} = \frac{I_{tot}}{L_{asap}}$$

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

Avg. Instructions per Access
(determined empirically for each app)

$$IPC_{max} = \frac{I_{tot}}{L_{asap}}$$

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

Limit on Avg. Memory Access Time (taken from CACTI)

$$IPC_{max} = \frac{I_{tot}}{L_{\text{asap}}}$$

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

$$IPC_{max} = \frac{I_{tot}}{L_{asap}}$$

Theoretical
”maximum” IPC

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

$$IPC_{max} = \frac{I_{tot}}{L_{asap}}$$

Total # of Instructions in ADAG

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

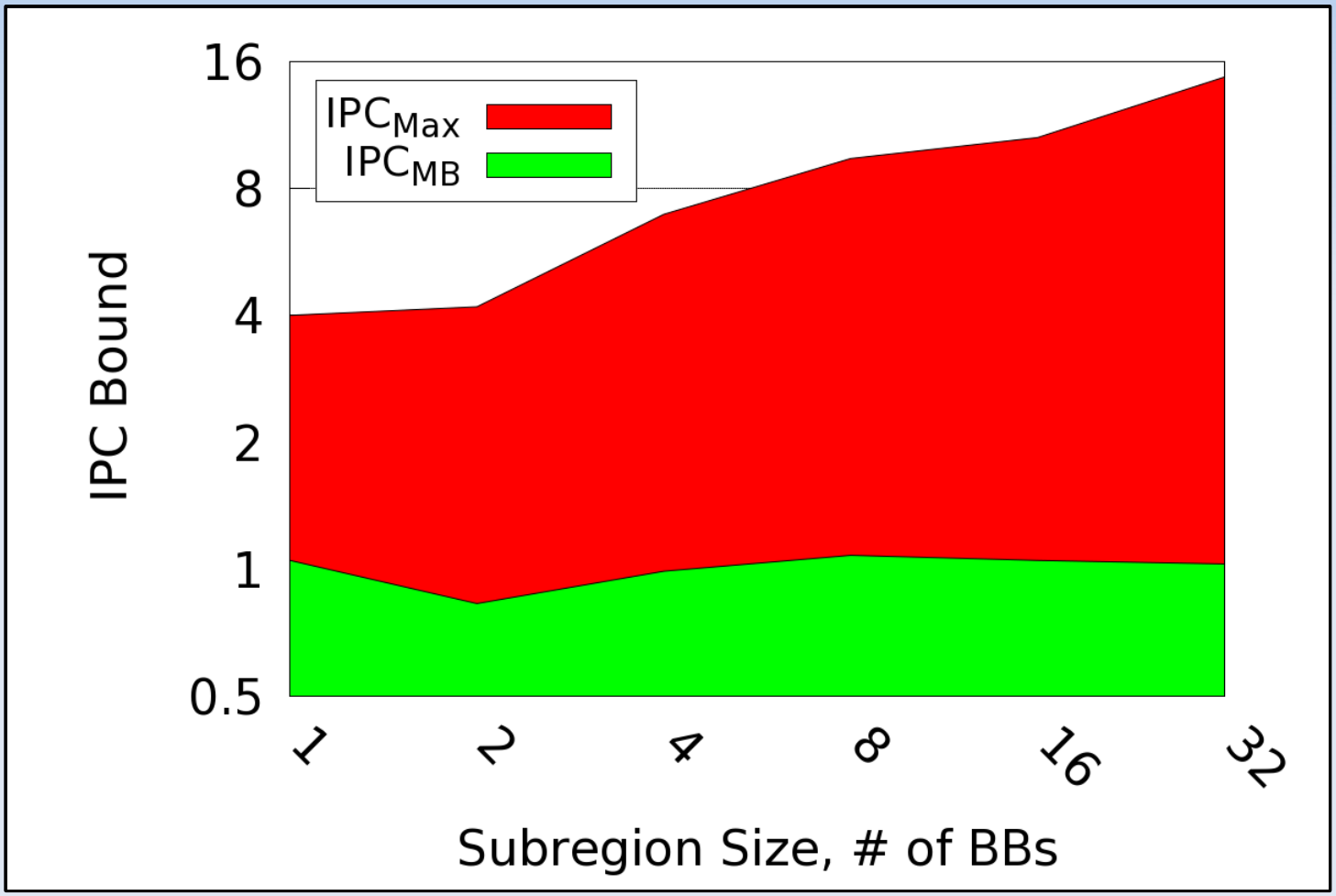
$$IPC_{MB} = \frac{T_C * IPA}{AMAT_{\lim}}$$

$$IPC_{max} = \frac{I_{tot}}{L_{asap}}$$

# of Stages in ADAG

# Application Characterization: Memory Wall

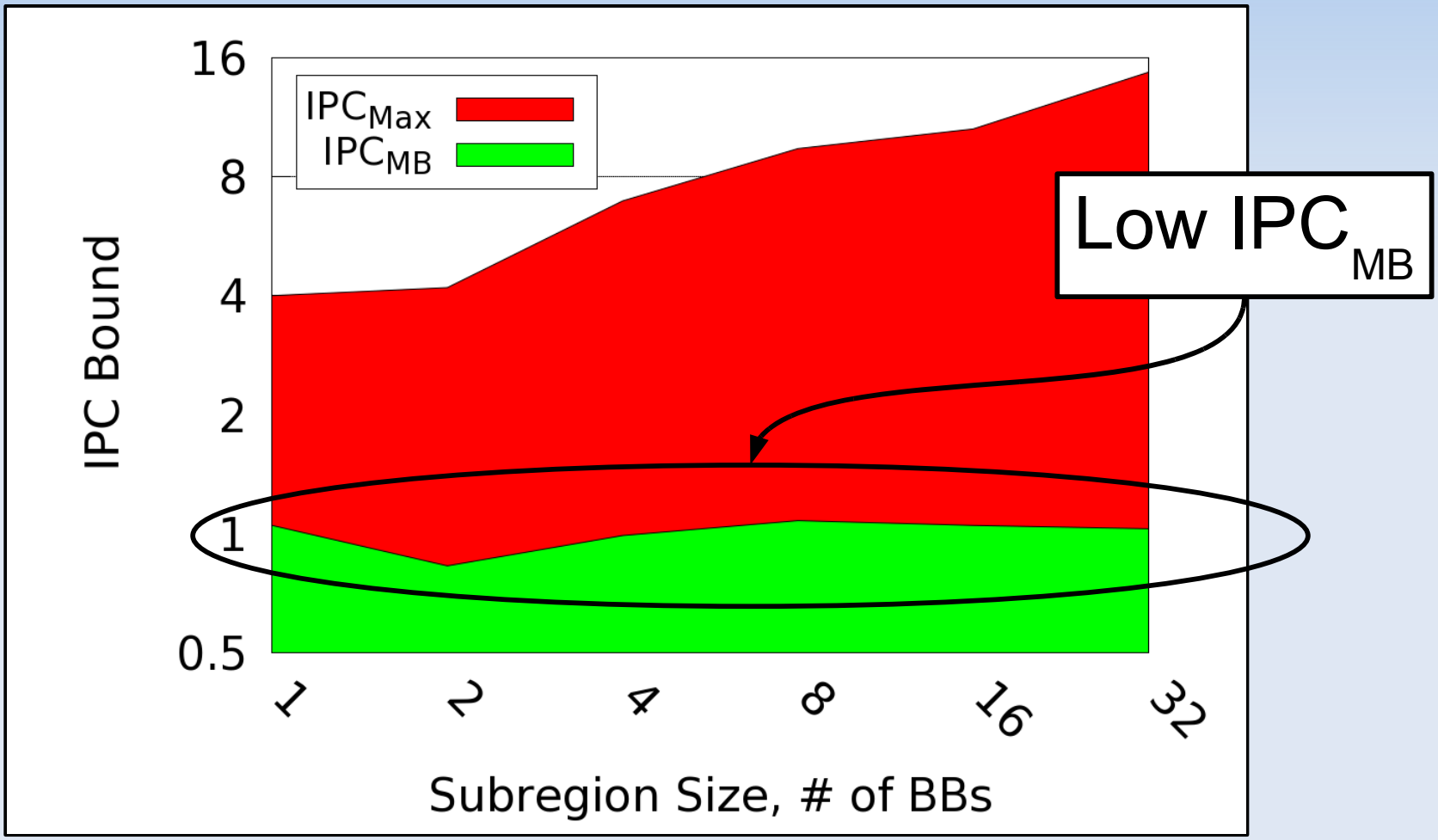
## Memory Latency-Bounded Speculation Depth




sp.4B
(Sandia-Int)

# Application Characterization: Memory Wall

## Memory Latency-Bounded Speculation Depth

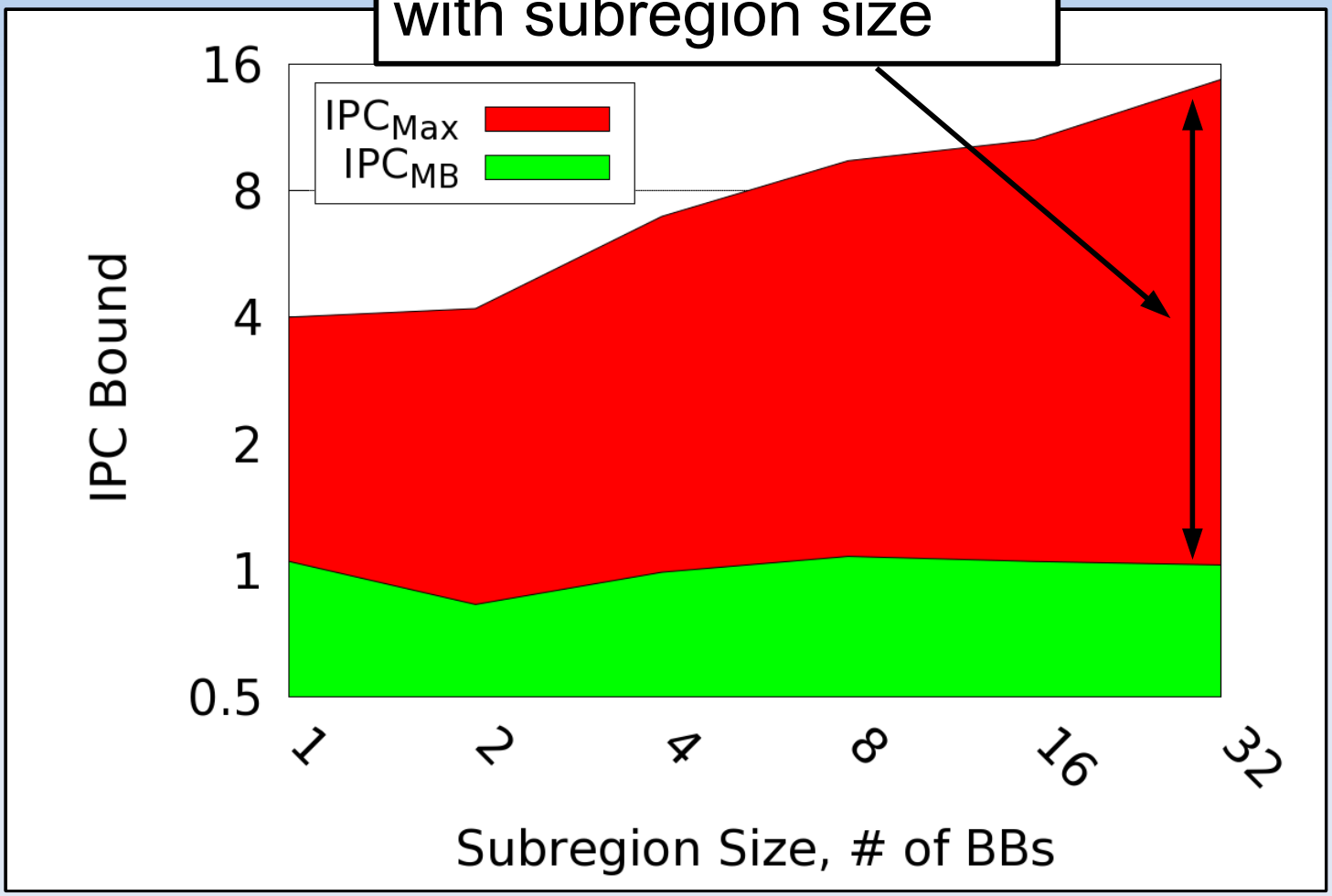



sp.4B
(Sandia-Int)

# Application Characterization: Memory Wall

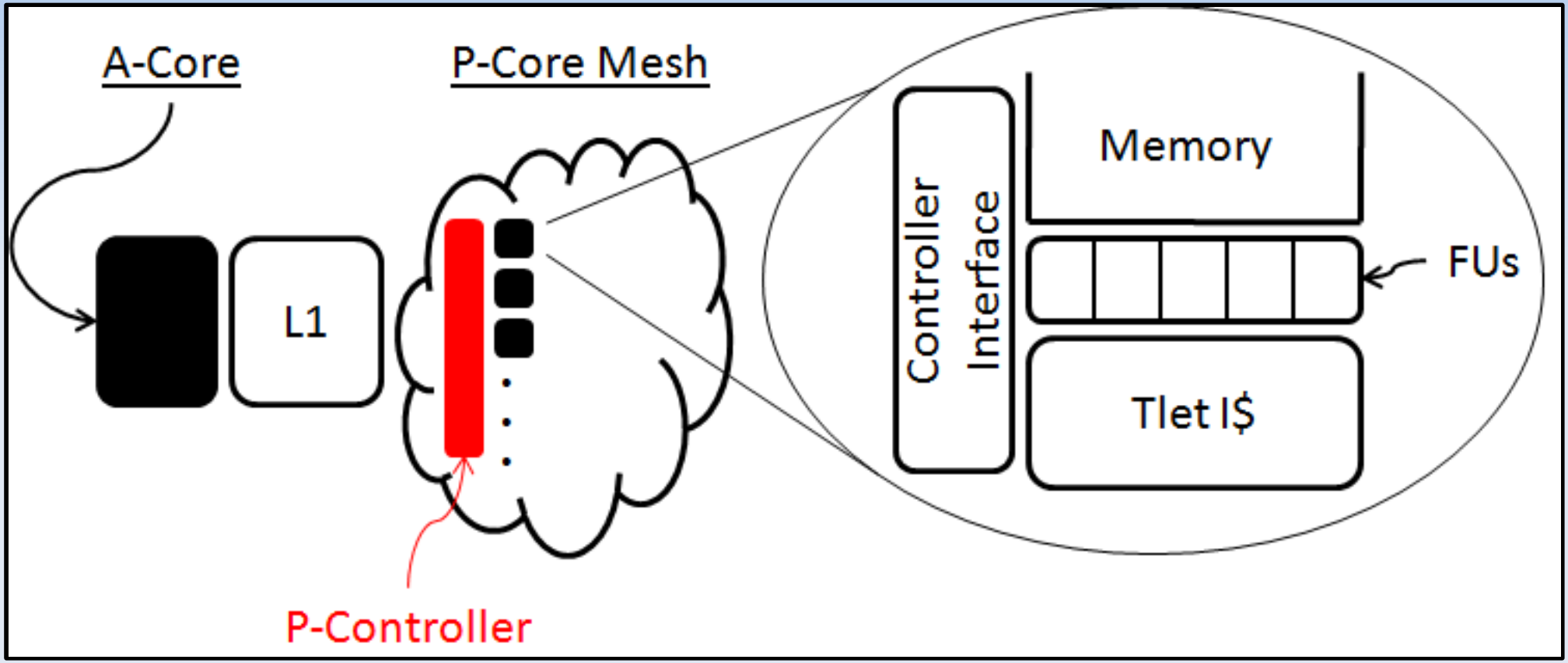
Memory Late ... lation Depth

Wasted "potential" performance increases with subregion size

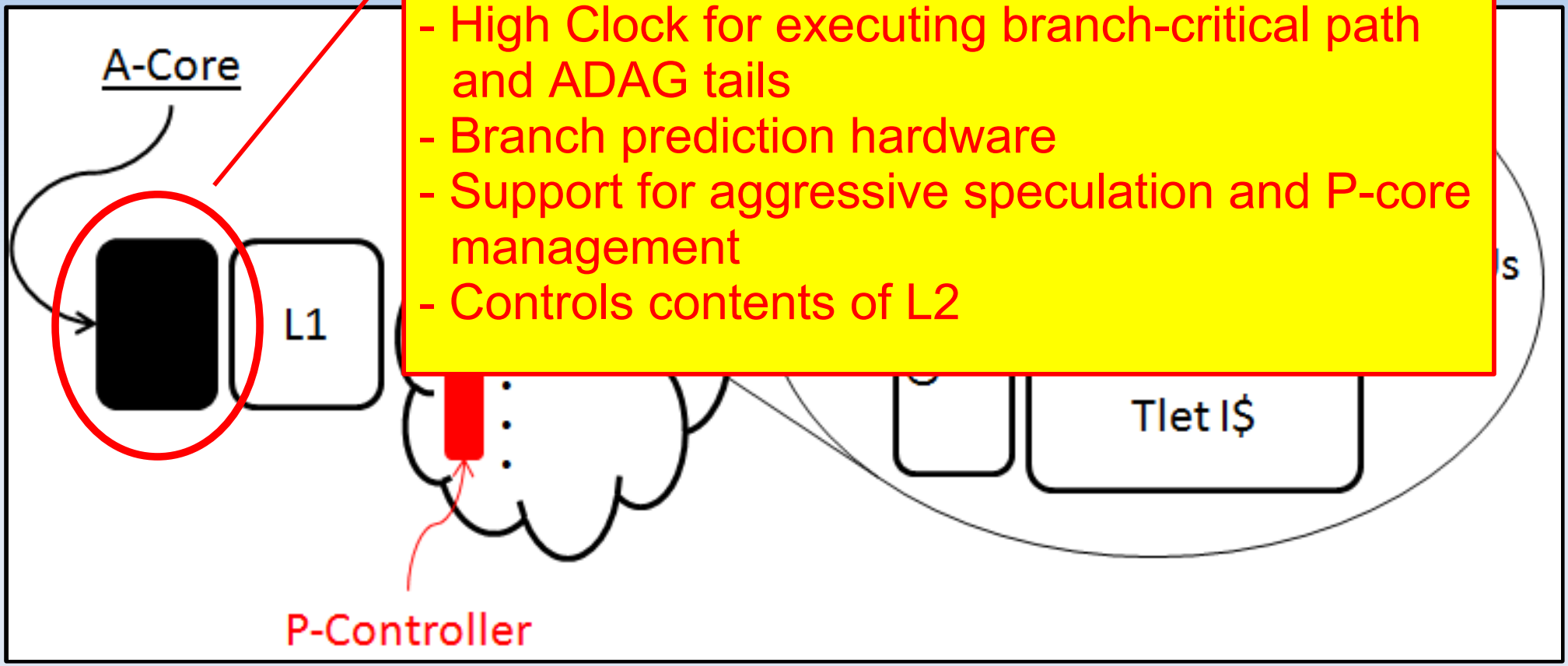



sp.4B
(Sandia-Int)

# PAM Architecture

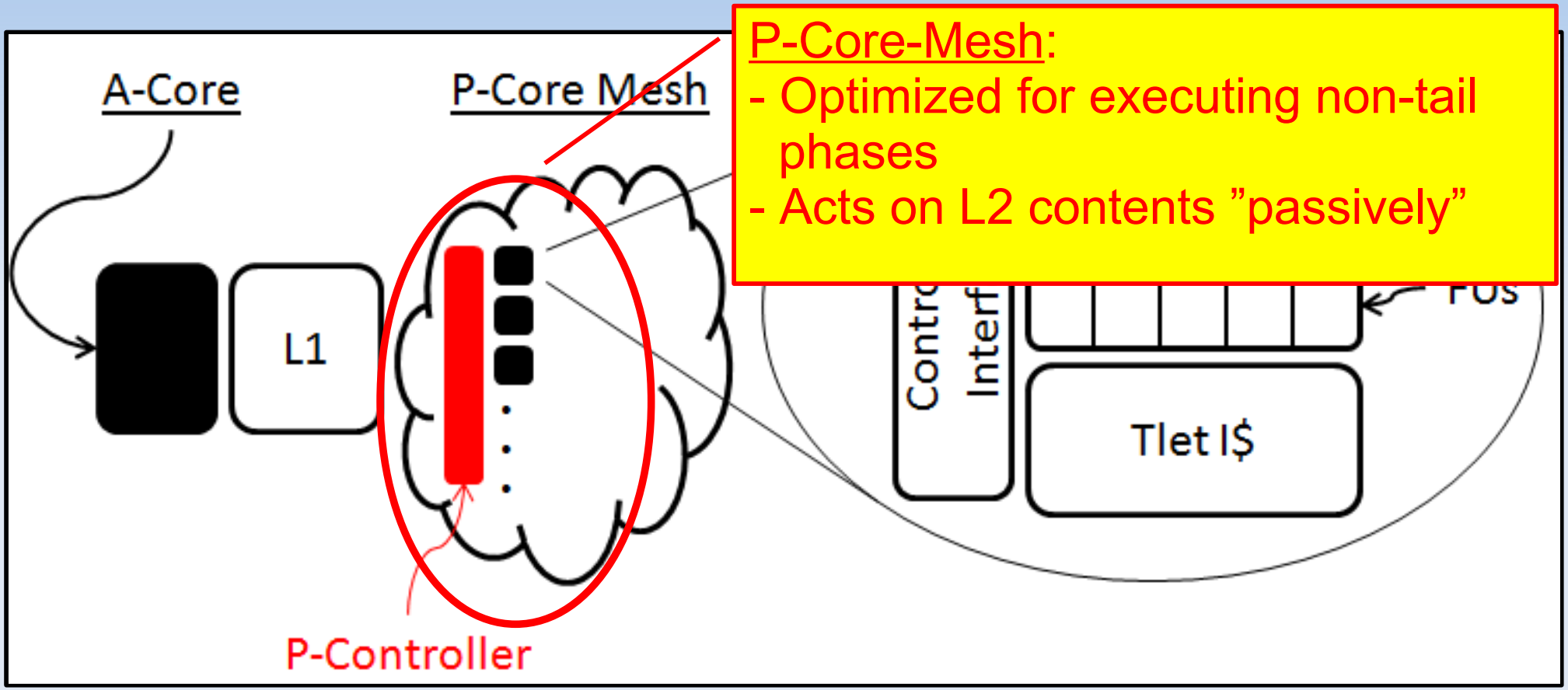


# PAM Architecture



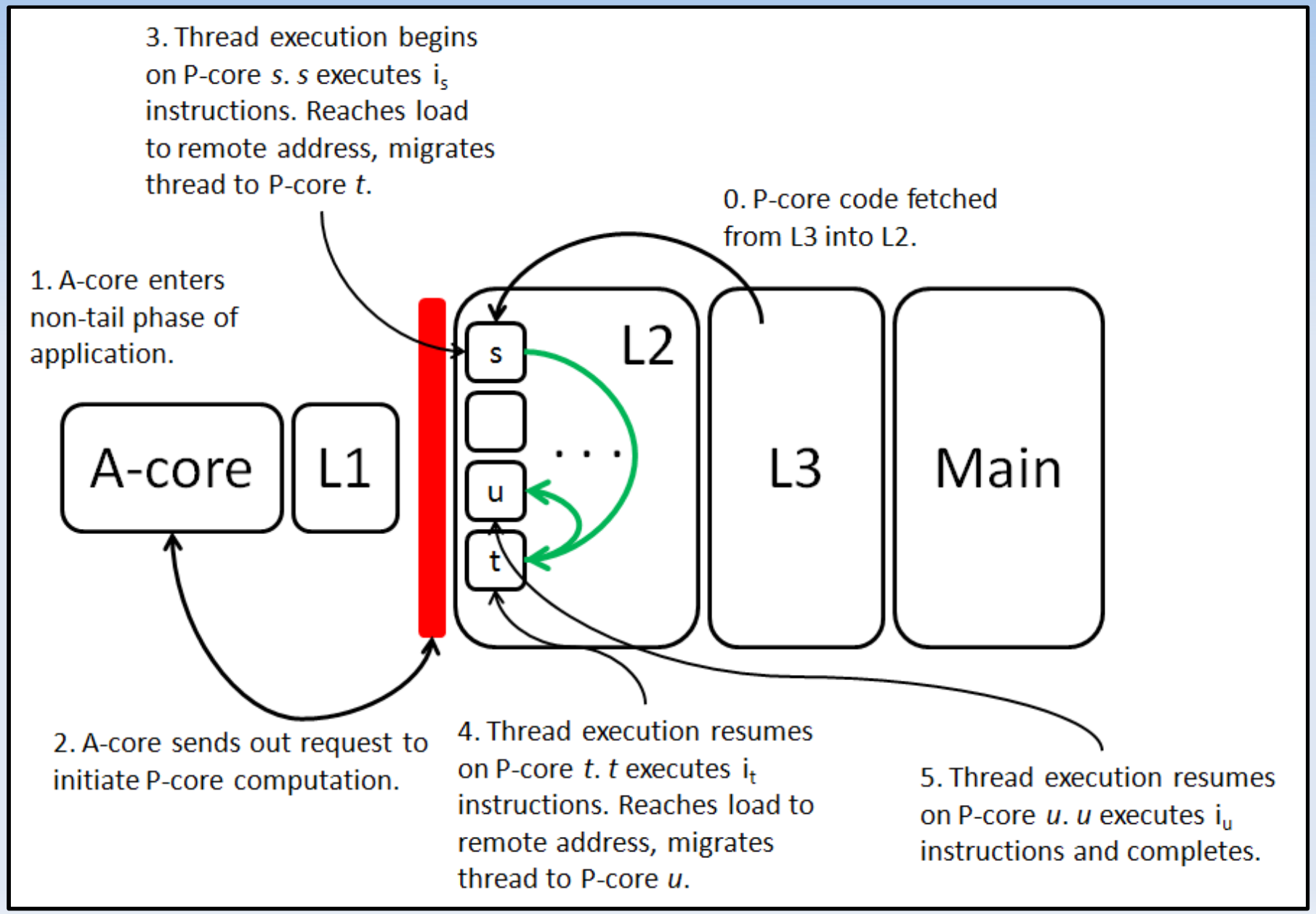

A-Core Features:
- High Clock for executing branch-critical path and ADAG tails
- Branch prediction hardware
- Support for aggressive speculation and P-core management
- Controls contents of L2

# PAM Architecture

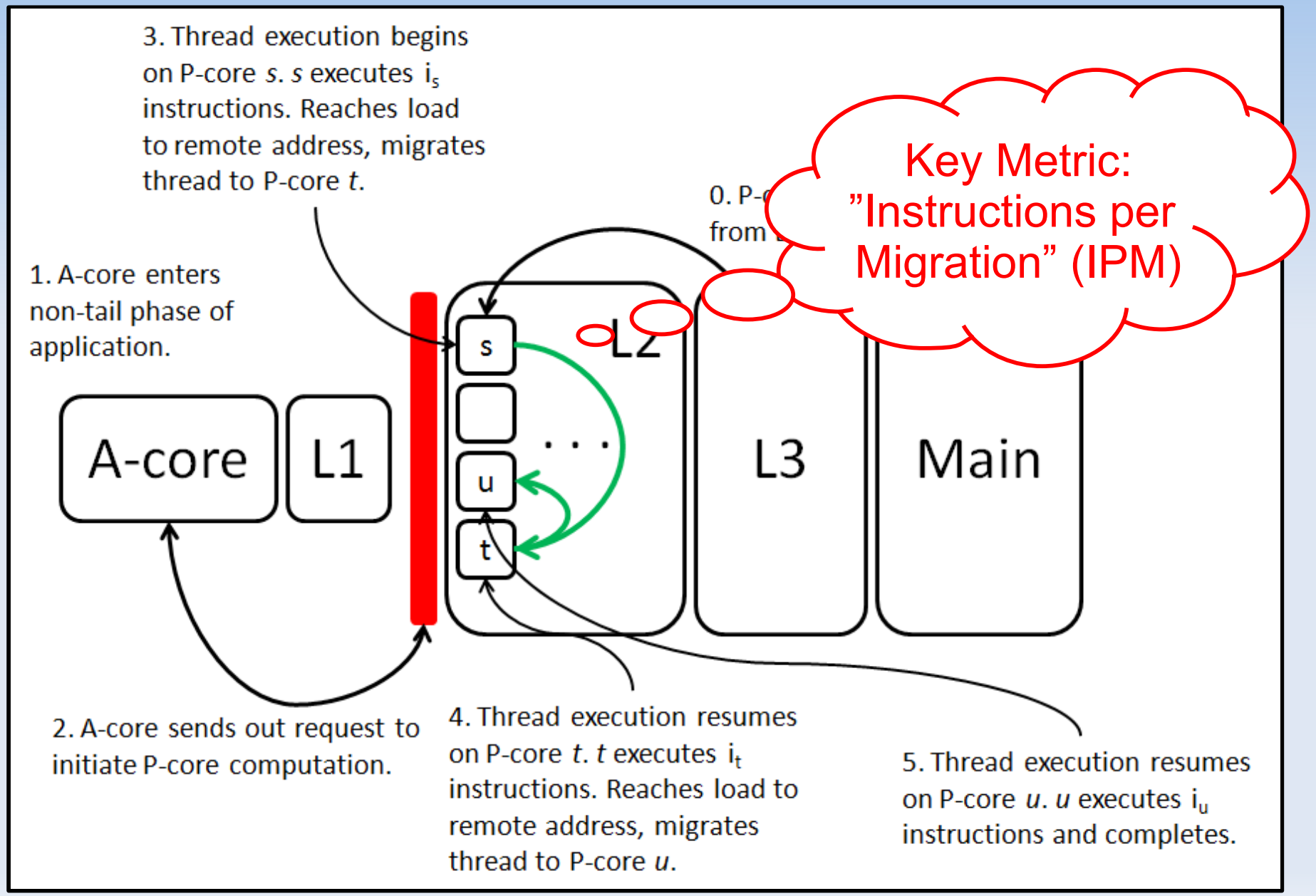



P-Core-Mesh:
- Optimized for executing non-tail phases
- Acts on L2 contents "passively"

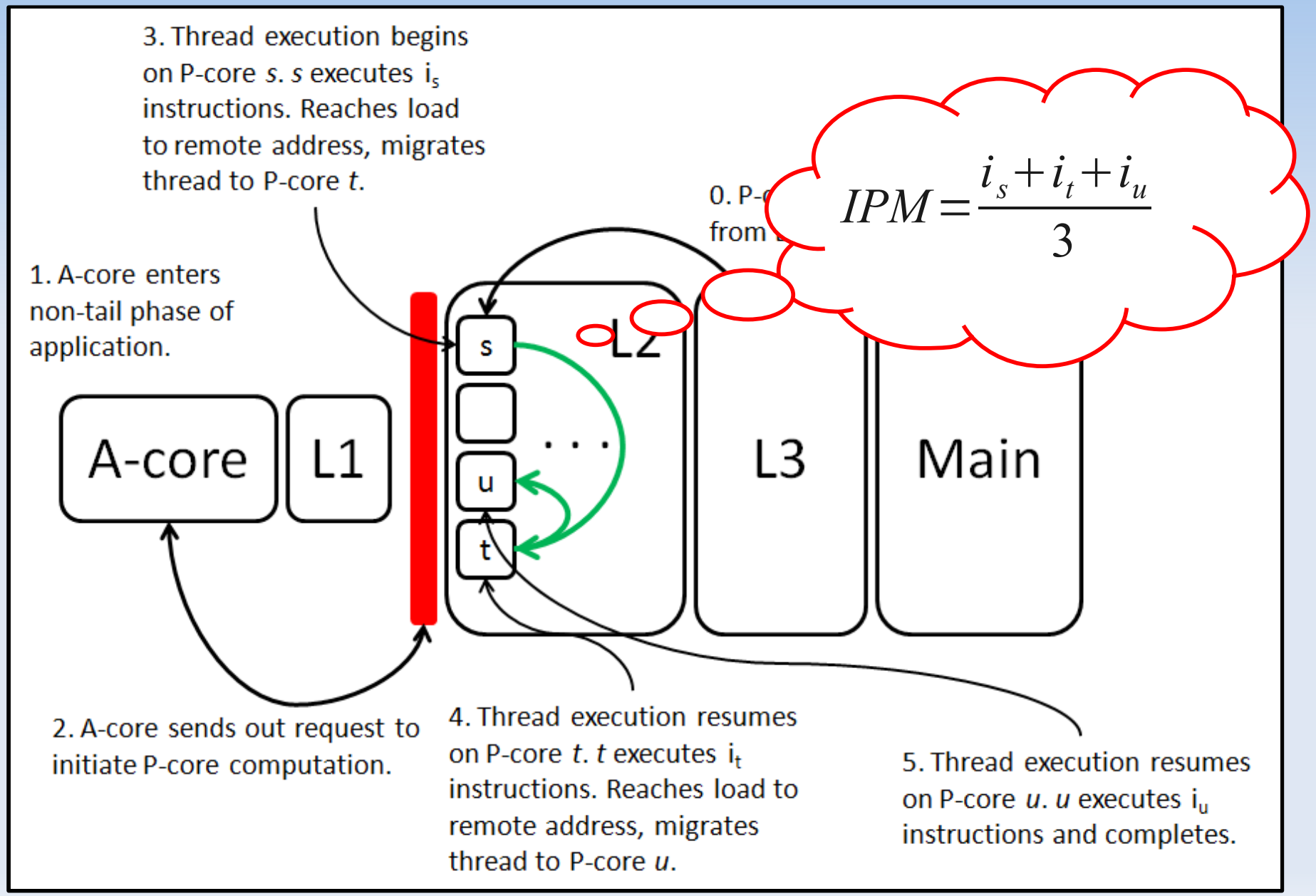
# PAM Architecture



# PAM Architecture



# PAM Architecture



# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\underbrace{\frac{T_C}{p * IPC_{\lim}} + MPI * T_M}_{} = \frac{T_C}{IPC_{\lim}}$$

Set average instruction execution time for baseline

equal to

Average instruction execution time with p-cores (includes migration overhead)

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

# of p-cores
(2 - 64)

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

Best-case $IPC_{MB}$ when speculating aggressively

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

Avg. migrations per instruction (calculated)

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

Migration time (taken from CACTI as latency for cache size 1 – 16MB)

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

Solve for IPM

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

At breakeven, call this *mIPM*, for *minimum instructions per migration* required to match baseline performance.

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

Solve for IPM

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

# Analytical Model

- Break-even plane btw migration and conventional caching

$$\frac{T_C}{p * IPC_{\lim}} + MPI * T_M = \frac{T_C}{IPC_{\lim}}$$

$$MPI = \frac{T_C}{IPC_{\lim} * T_M} * \frac{p-1}{p}$$

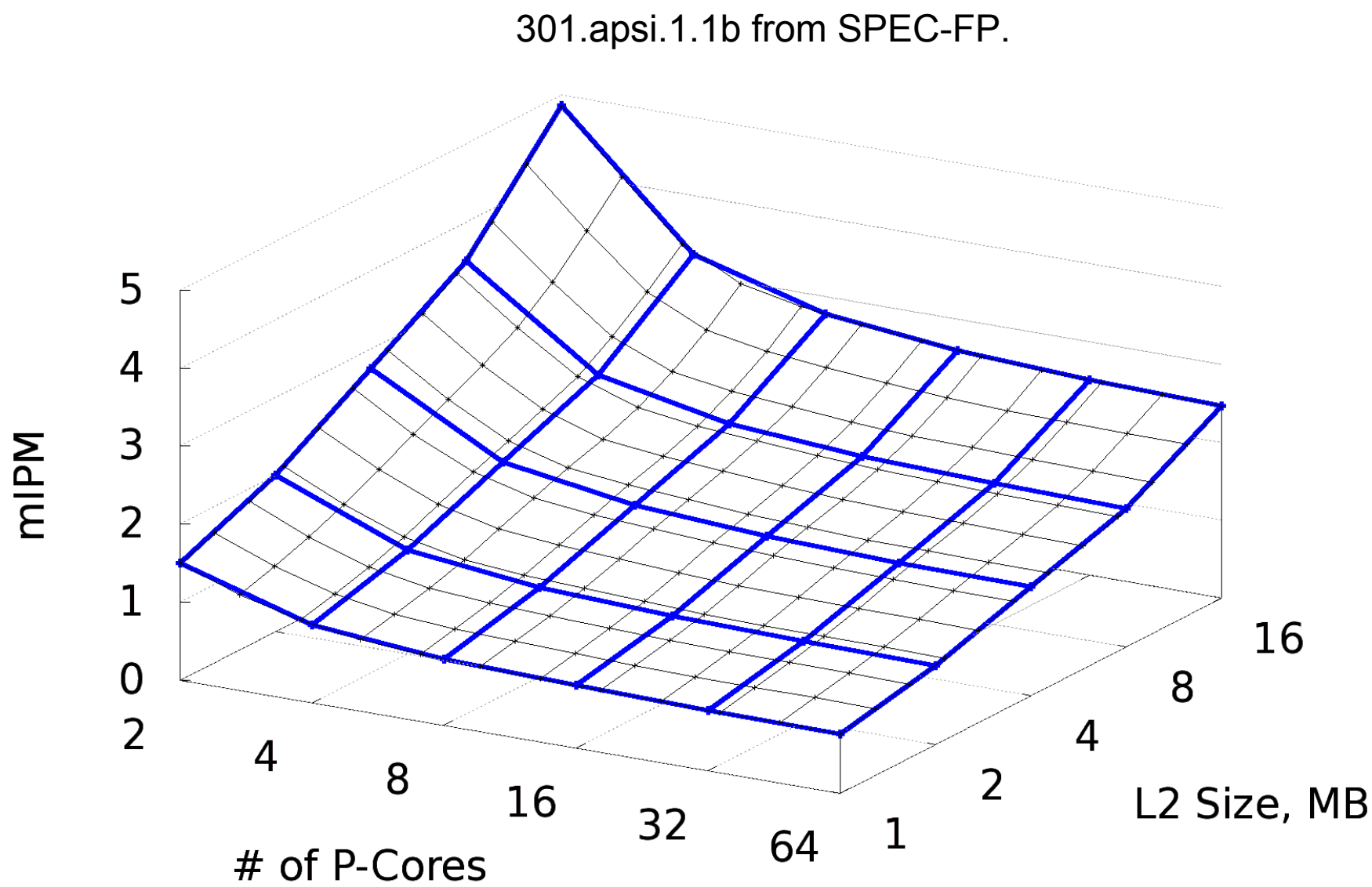
$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

Solve for IPM

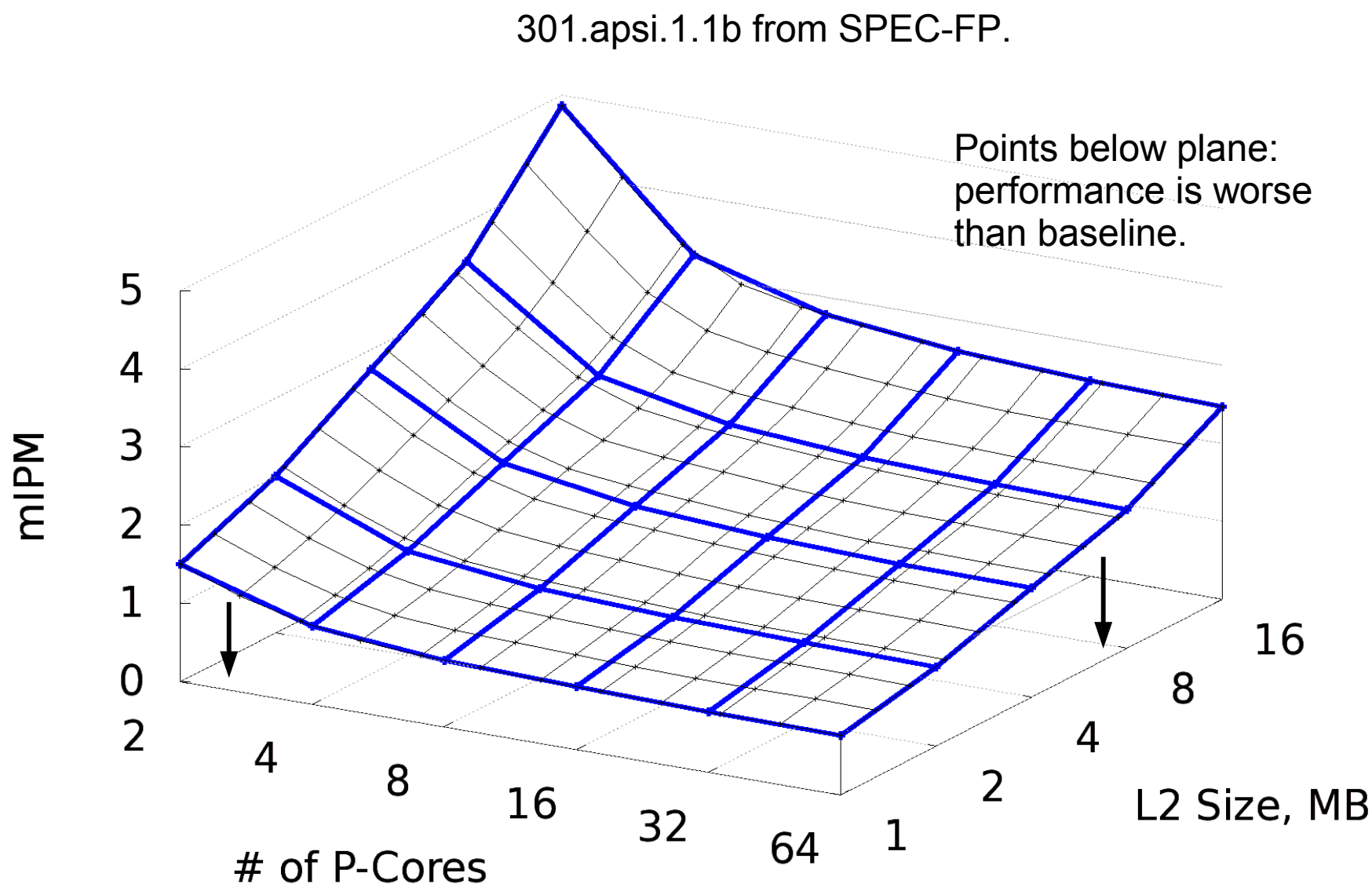
At breakeven, call this *mIPM*, for *minimum instructions per migration* required to match baseline performance.

(Lower is better.)

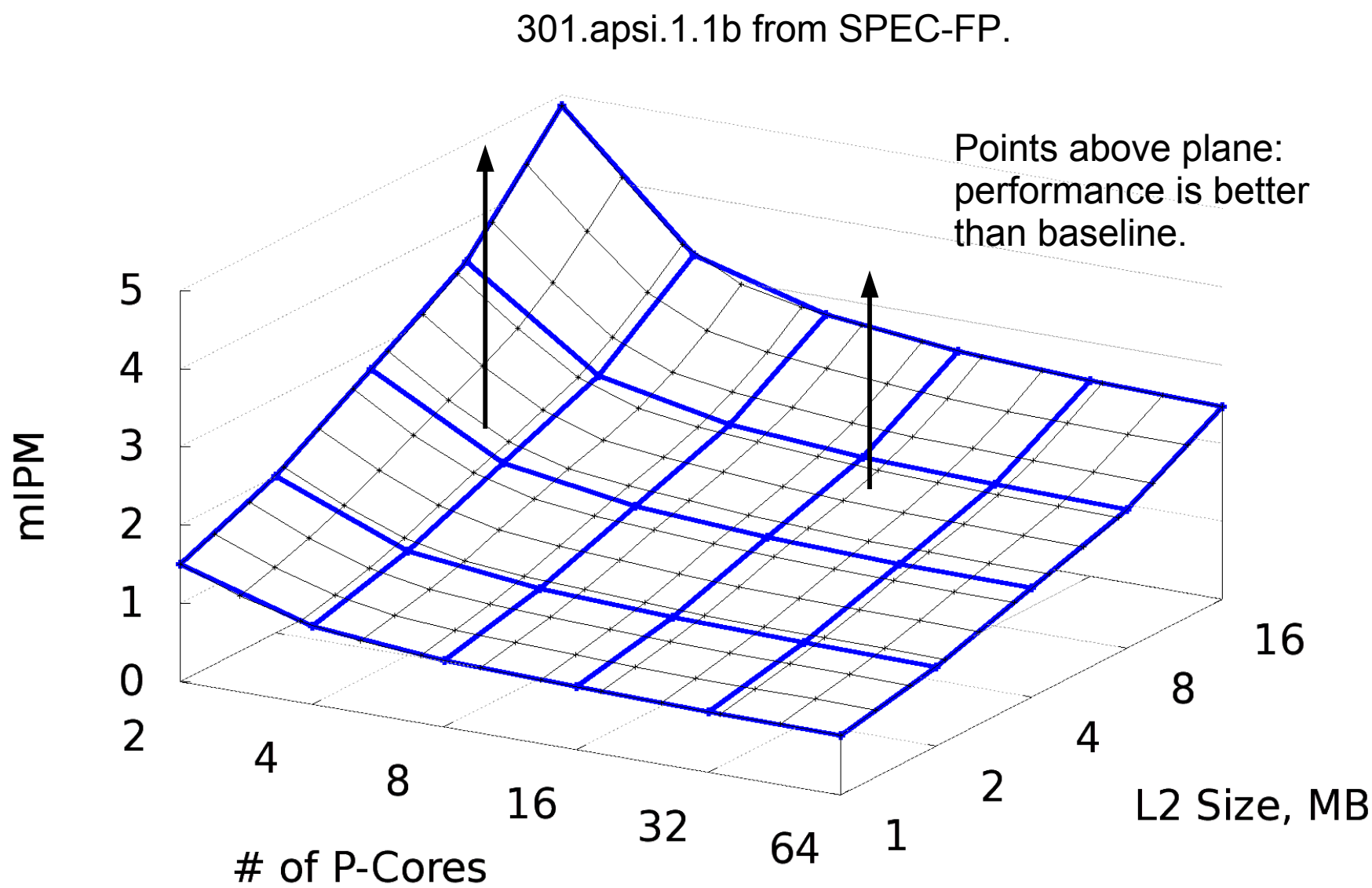
# Performance Results



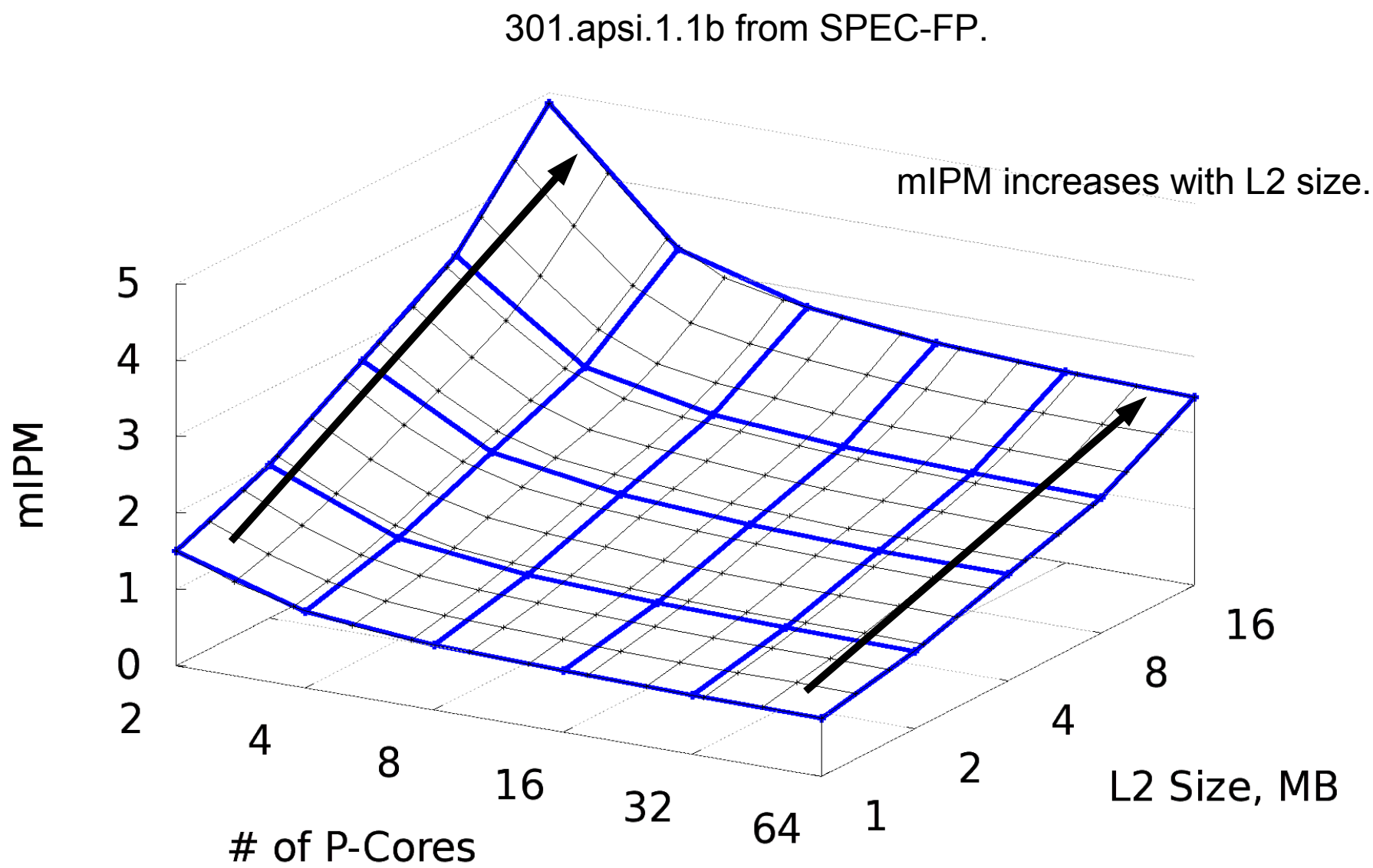
# Performance Results



# Performance Results



# Performance Results

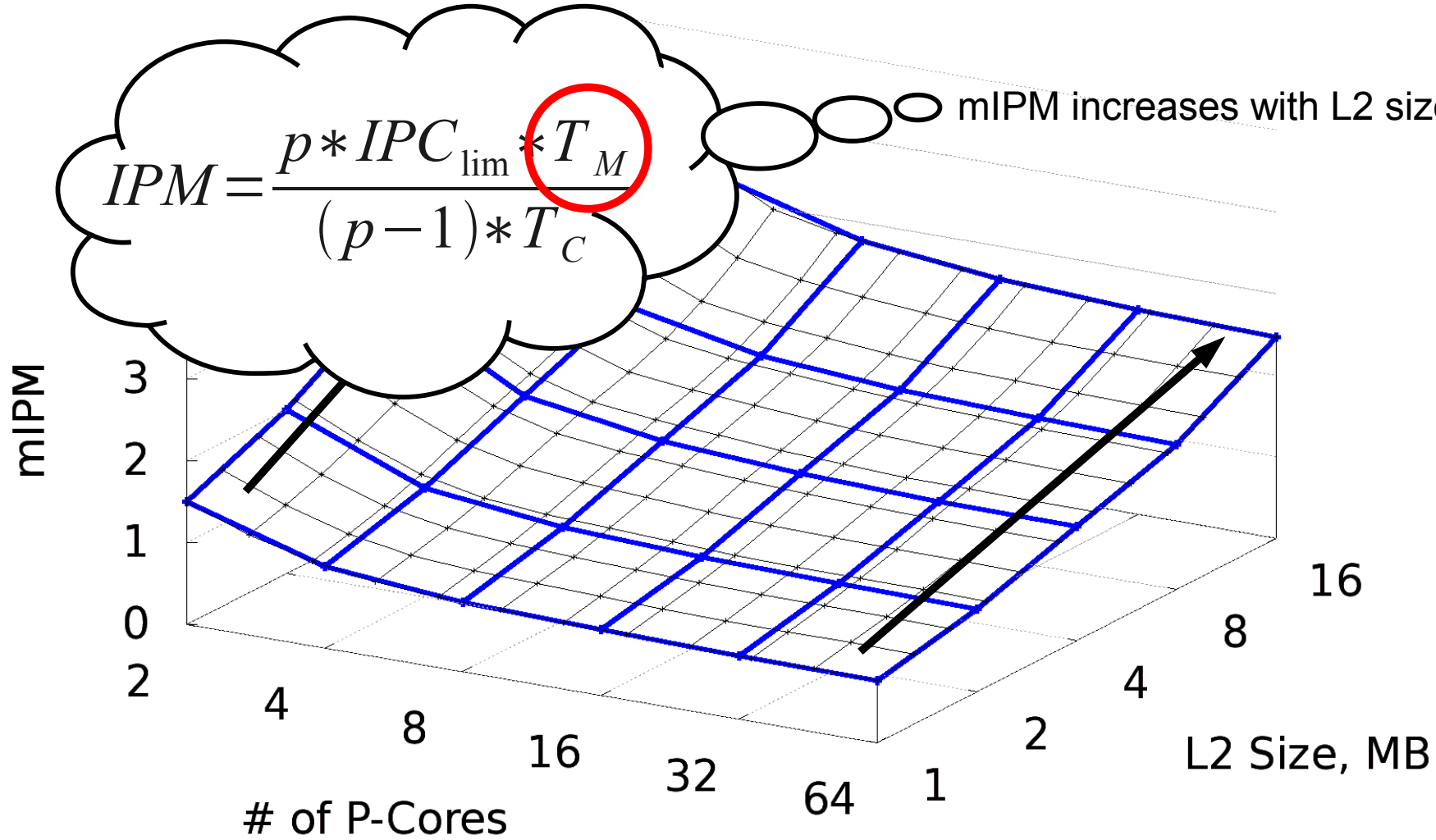


# Performance Results

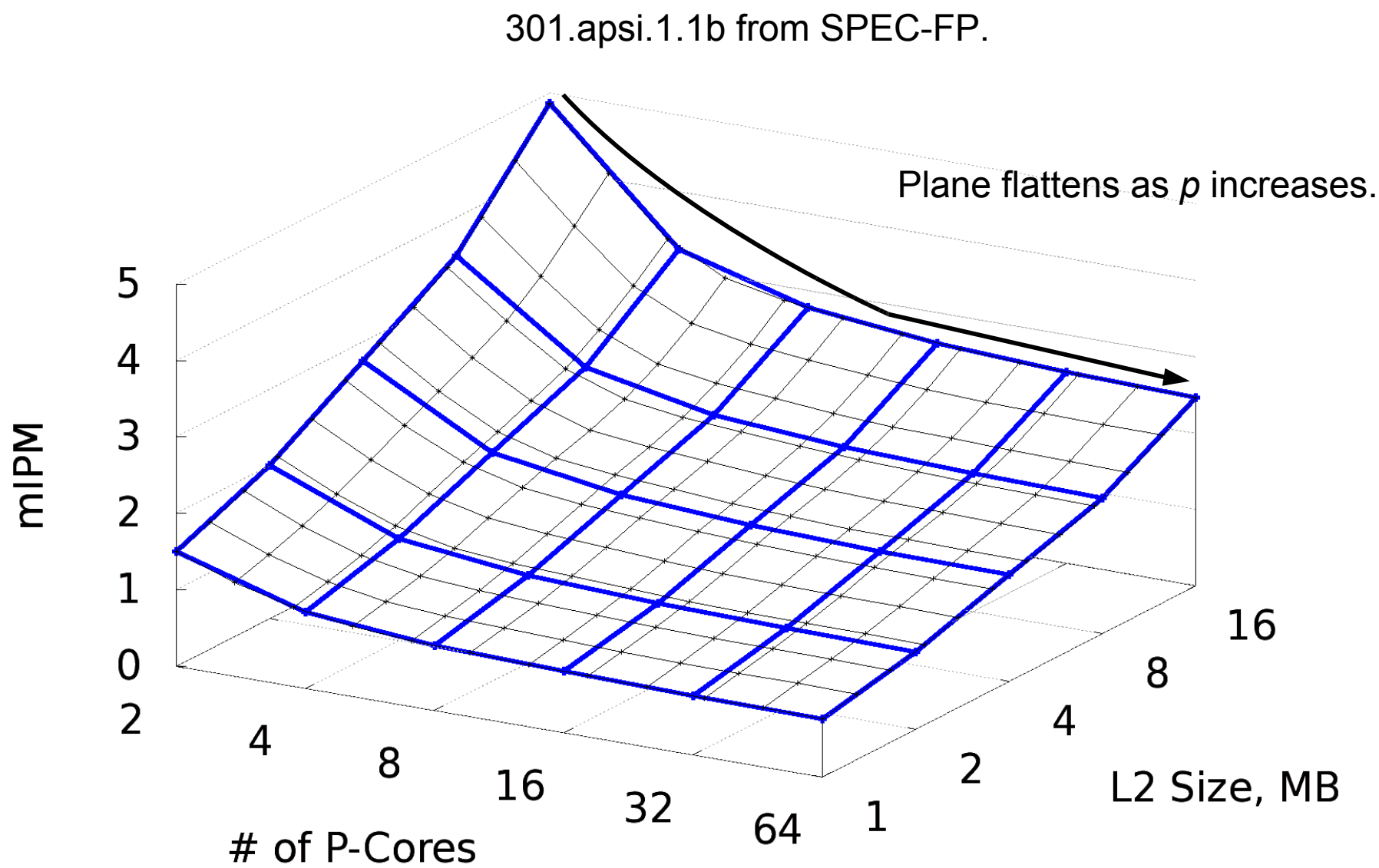
301.apsi.1.1b from SPEC-FP.

$$IPM = \frac{p * IPC_{\lim} * T_M}{(p-1) * T_C}$$

mIPM increases with L2 size.

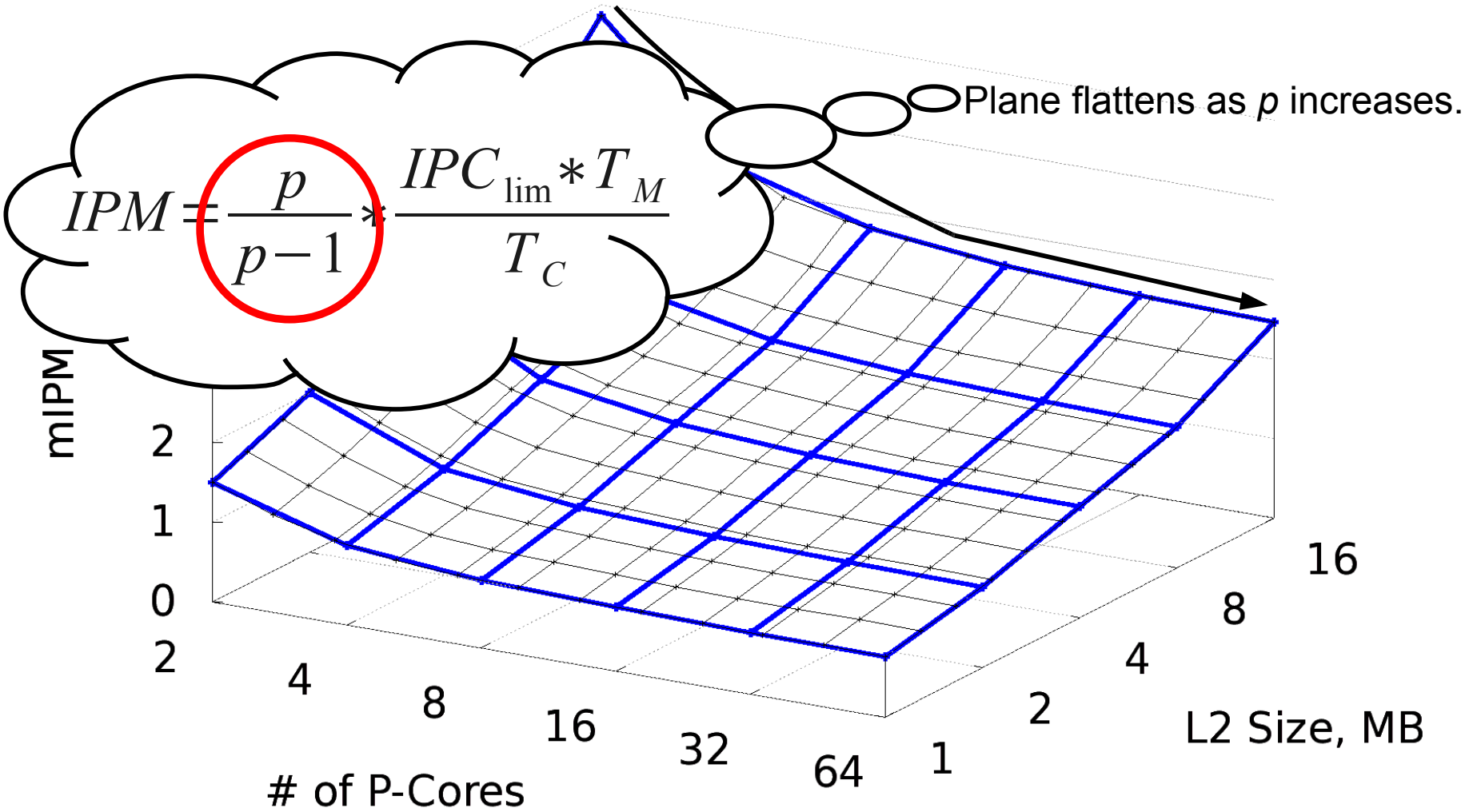


# Performance Results

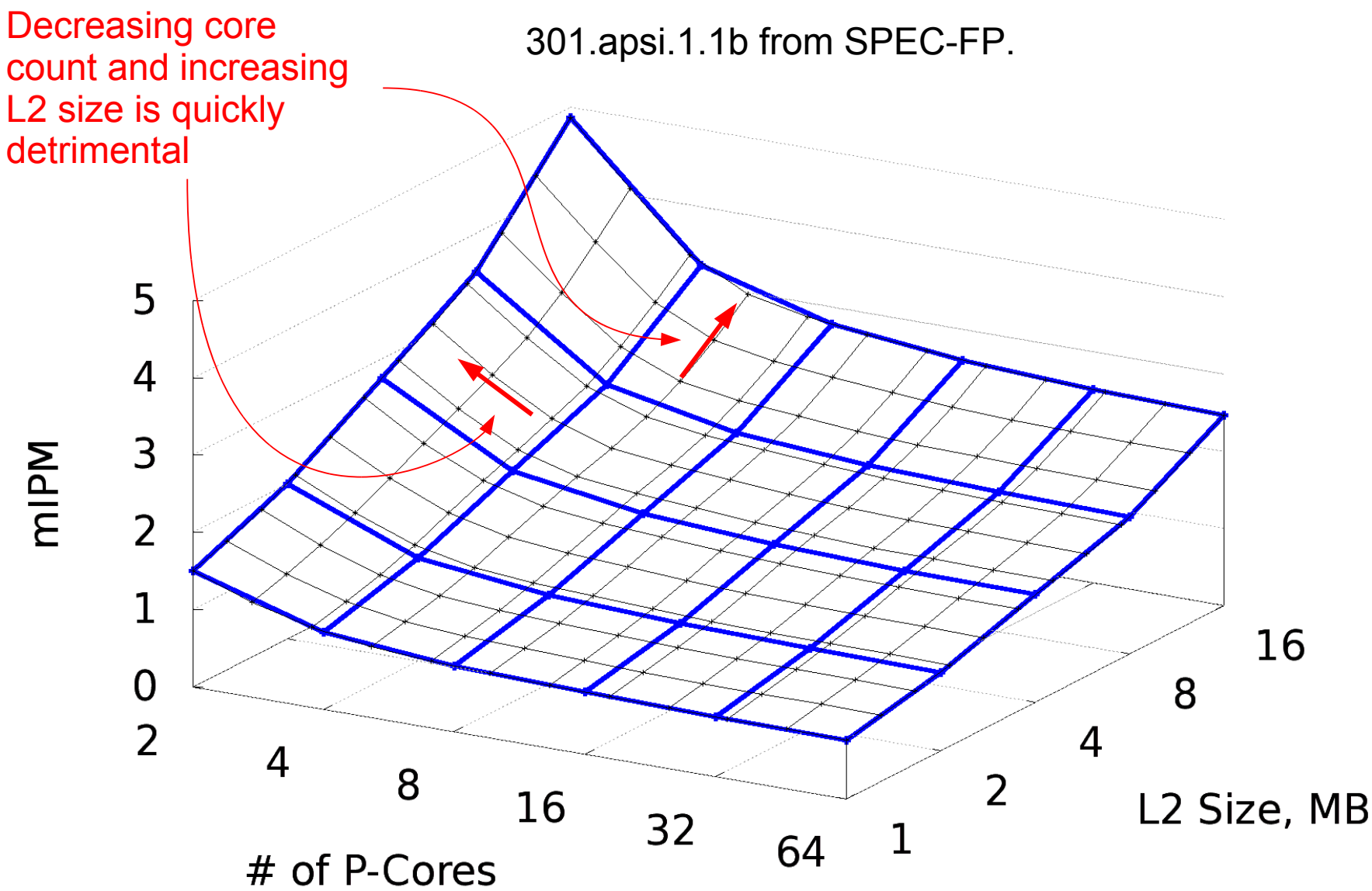


# Performance Results

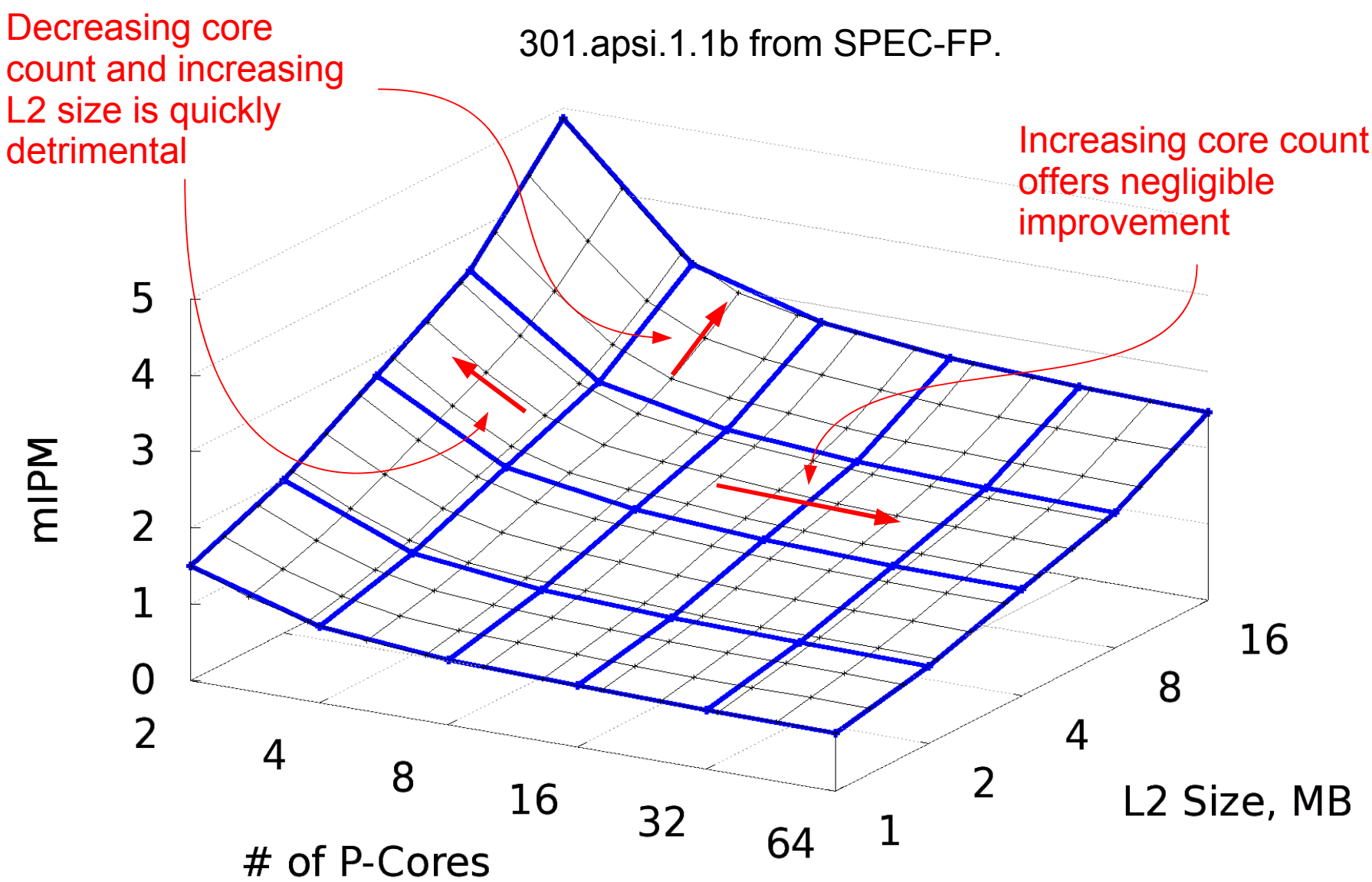
301.apsi.1.1b from SPEC-FP.



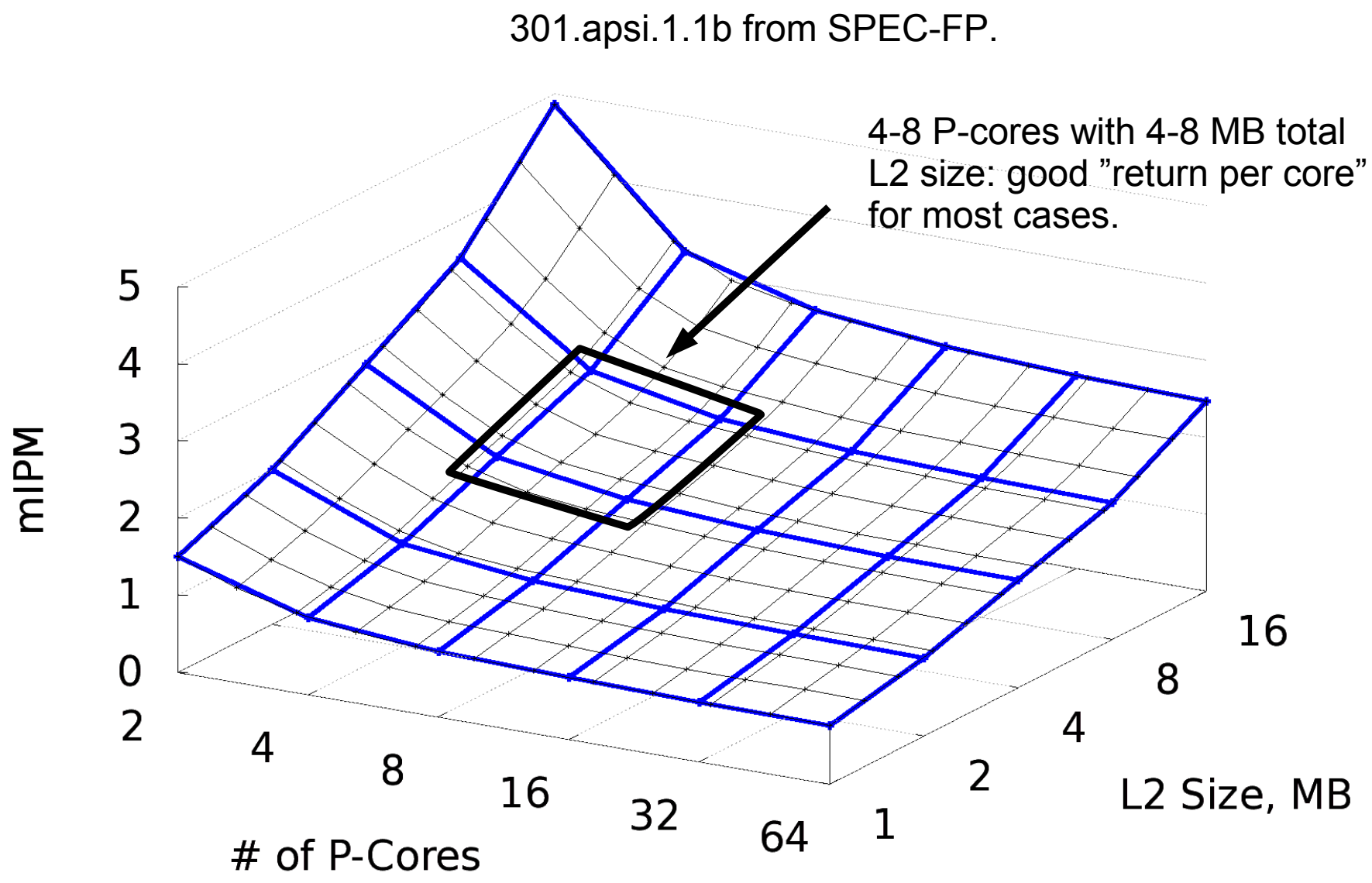
# Performance Results



# Performance Results

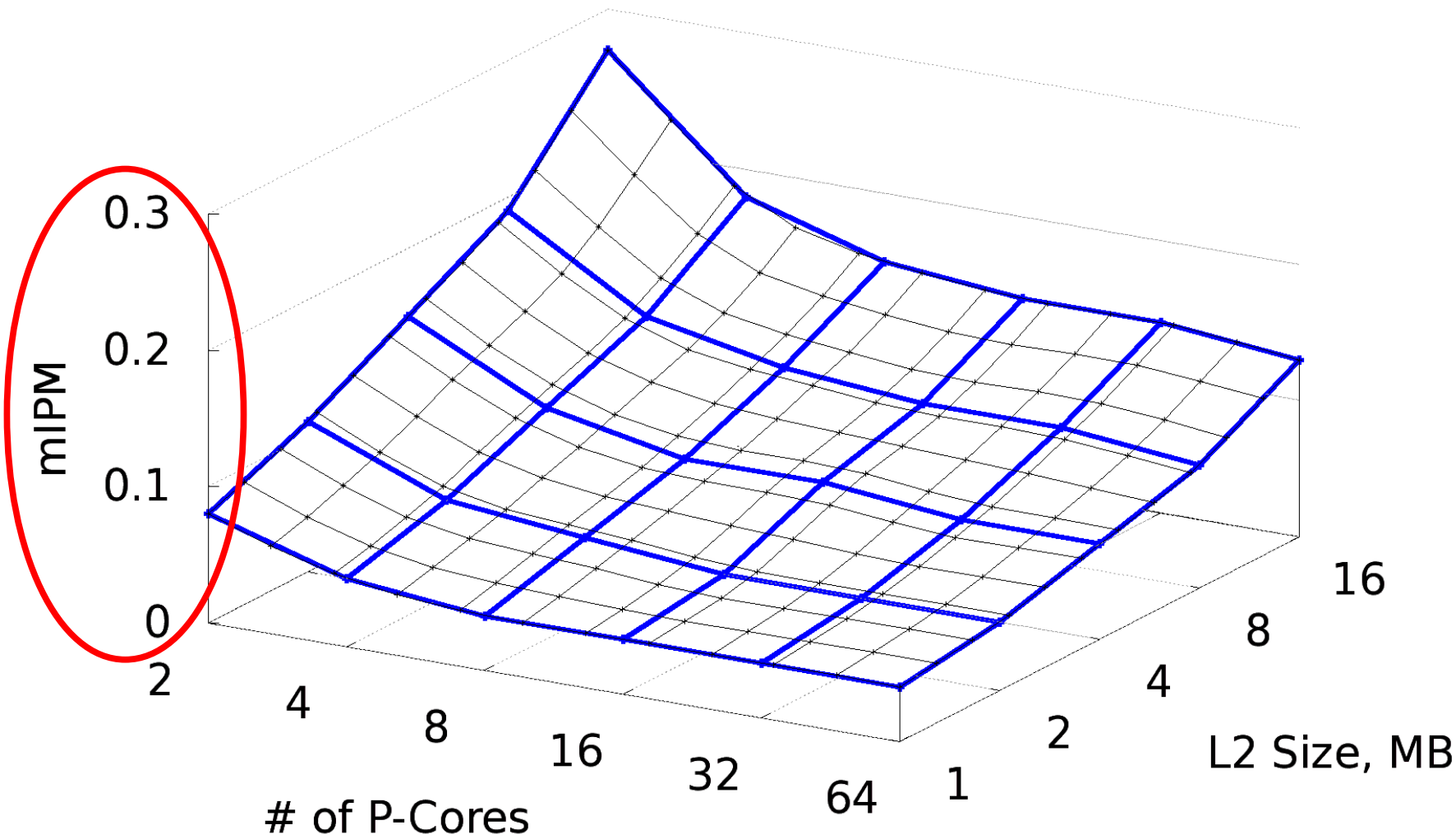


# Performance Results



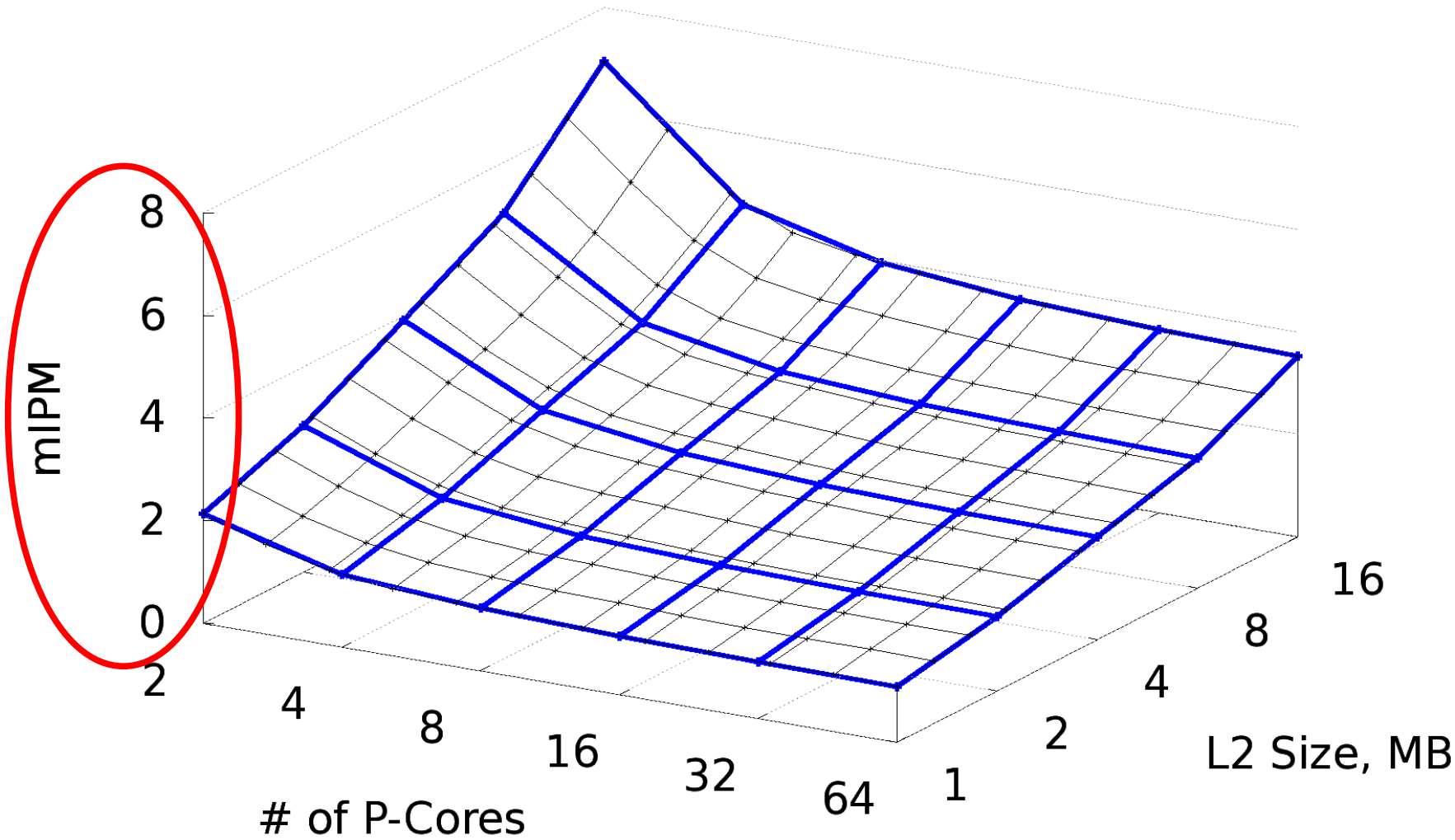
# Performance Results

Good case: 176.gcc.1.1b from SPEC-Int.



# Performance Results

Poor case: cube3.4b.vbr from Sandia-FP.



# Conclusions

- Asymmetry in PAM architecture is justified
  - Branch-critical paths
  - ADAG tails
- Sweet spot: 4 to 8 P-cores with 4 to 8 MB total L2 size
  - Equivalent to 1 or 2 MB per P-core/Bank
  - 2 – 4 mIPM
- mIPM across all cases:
  - Best case (min): 0.04 instructions
  - Worst Case (max): 7.09 instructions
  - Average: 1.88 instructions

# Summary

- Take-away: Why is 2 – 4 mIPM good?
- In on-going work
  - Observed 7.7 IPM on average across all traces
  - P-core thread cover applications up to 24%
- Other evidence that this is a good idea
  - Linpack experiments show up to 23% reduction in energy consumption (see Kogge, La Fratta, Vance, ”Facing the Exascale Energy Wall”, IWIA 2010.)

> <u>Moving computations into the cache hierarchy shows great potential</u> <

# Future Work

- PAM Cache Protocol
  - Handle case when P-core operands aren't in L2
  - Optimize locality between A-core and P-cores
  - Migration Conflicts: Multiple threads at same P-core
- Improve performance model
  - More accurate parallelization numbers
  - Model thread scheduling
  - Incorporate more memory hierarchy parameters
  - Interconnect design
- Verify accuracy of sampling methodology

# Future Work

Questions?

- PAM Cache Protocol
  - Handle case when P-core operands aren't in L2
  - Optimize locality between A-core and P-cores
  - Migration Conflicts: Multiple threads at same P-core
- Improve performance model
  - More accurate parallelization numbers
  - Model thread scheduling
  - Incorporate more memory hierarchy parameters
  - Interconnect design
- Verify accuracy of sampling methodology

# Supplementary Slides