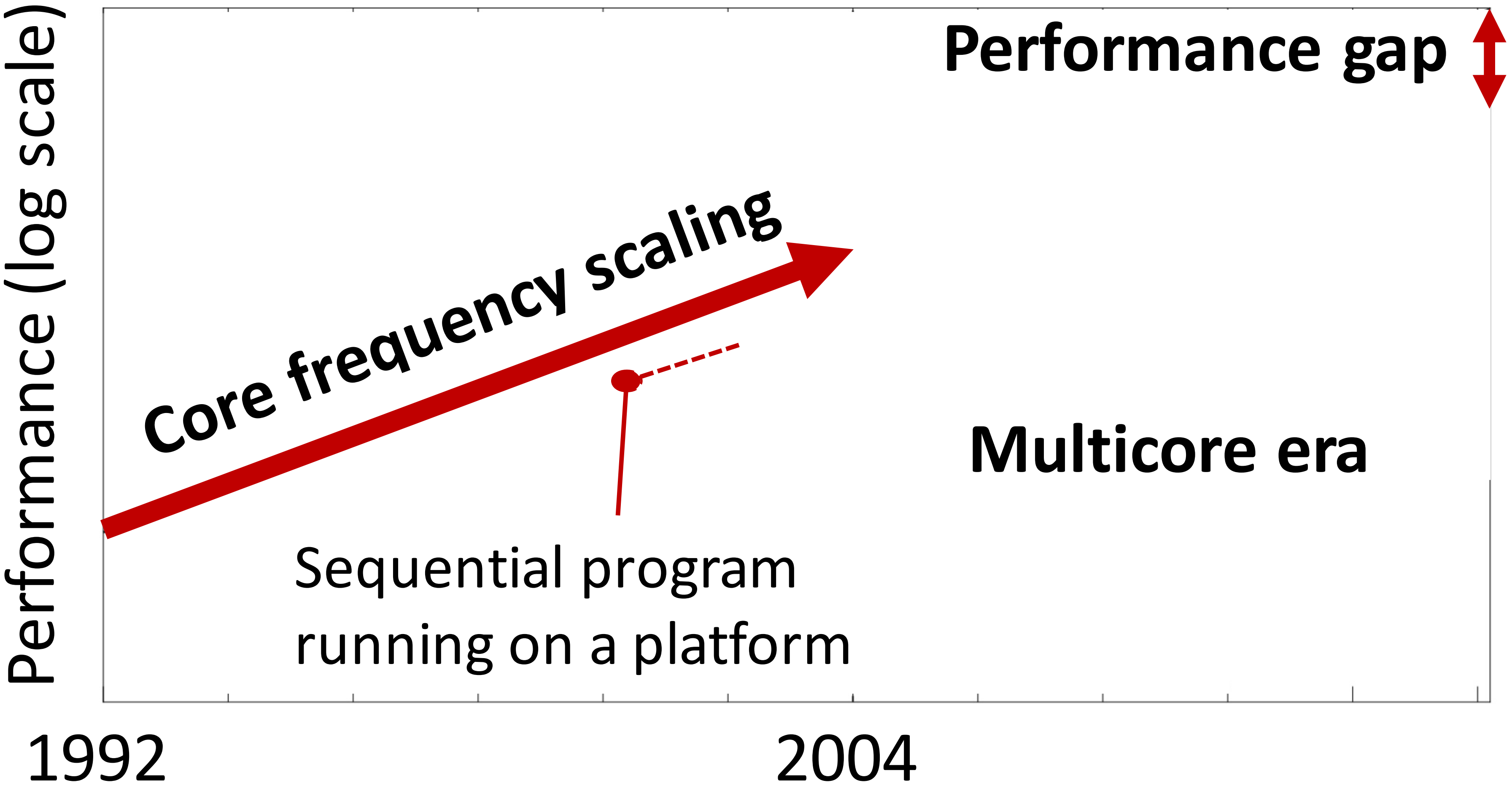
An example
# of a research compiler

EEGS 322: Compiler Construction

Simone Campanoni
Robby Findler

5/25/2016

Sequential programs are not accelerating like they used to



Multicores are underutilized

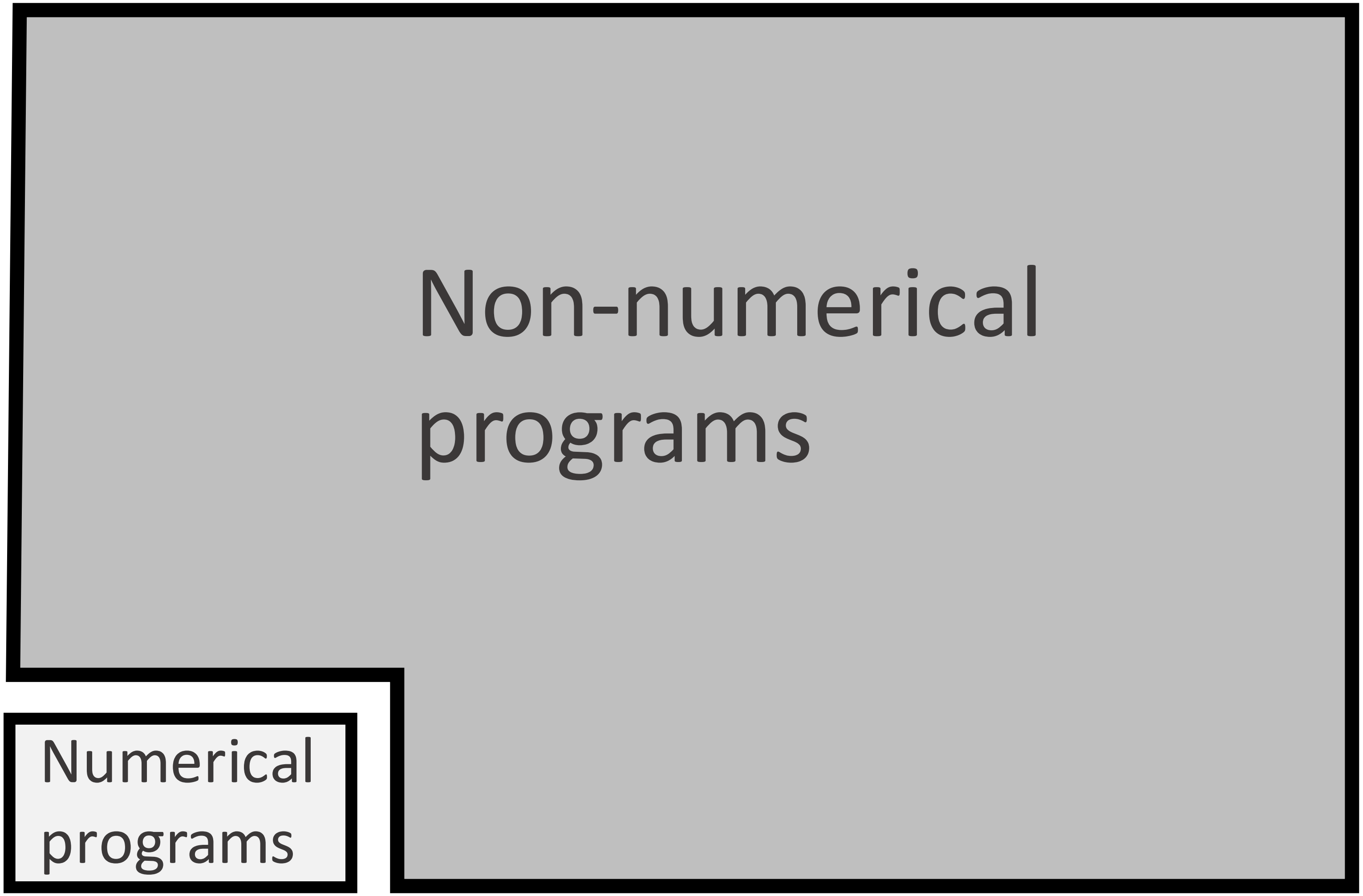
Single application:
Not enough explicit parallelism

- Developing parallel code is hard
- Sequentially-designed code is still ubiquitous

Multiple applications:
Only a few CPU-intensive applications running concurrently in client devices

Parallelizing compiler:
Exploit unused cores
to accelerate
sequential programs

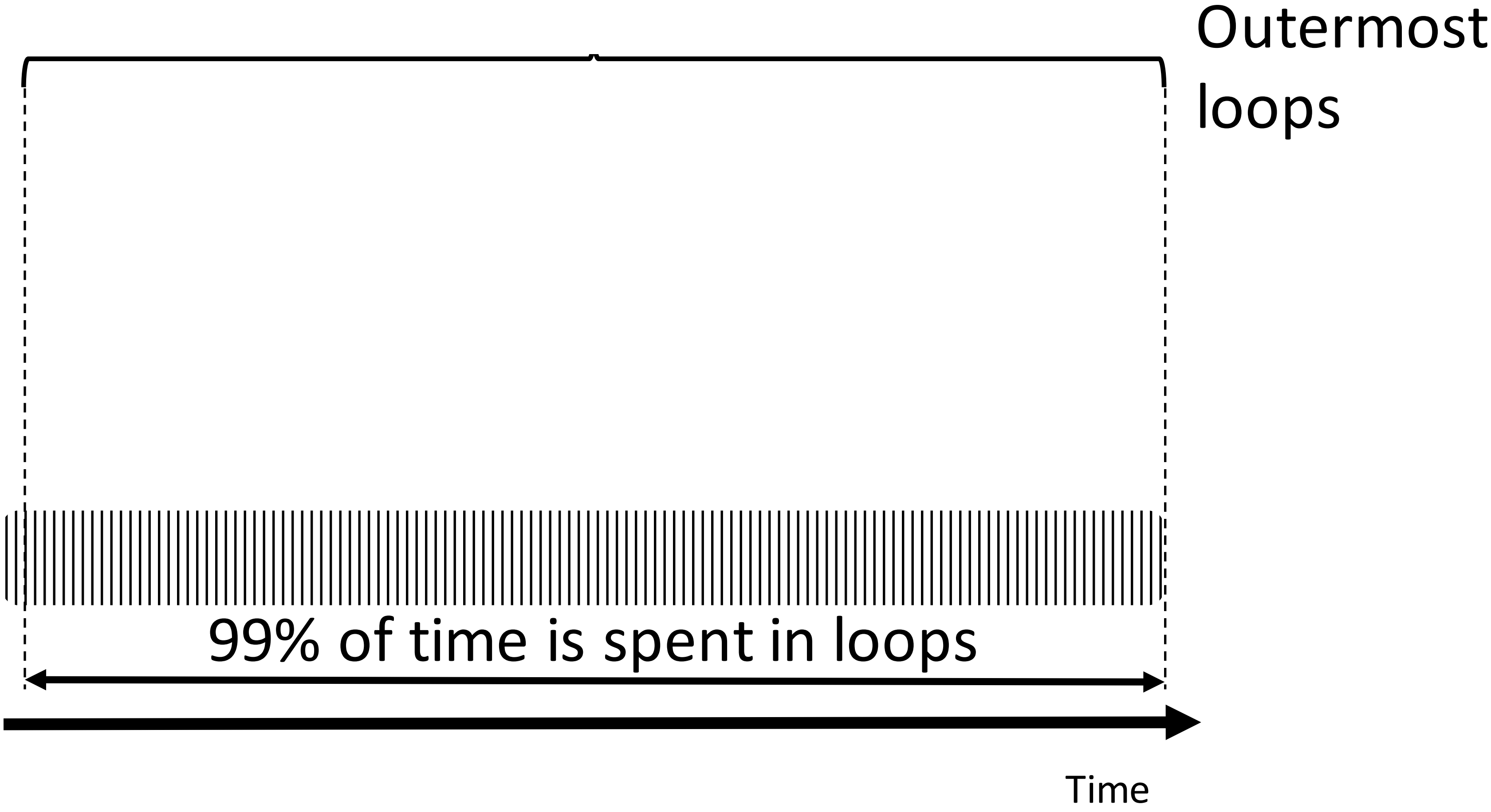
Non-numerical programs need to be parallelized



Non-numerical programs

Numerical programs

Parallelize loops to parallelize a program

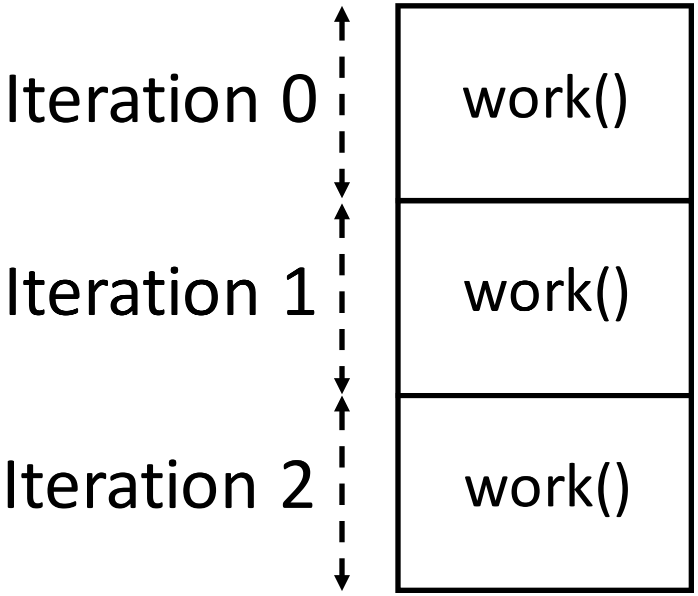


Outermost loops

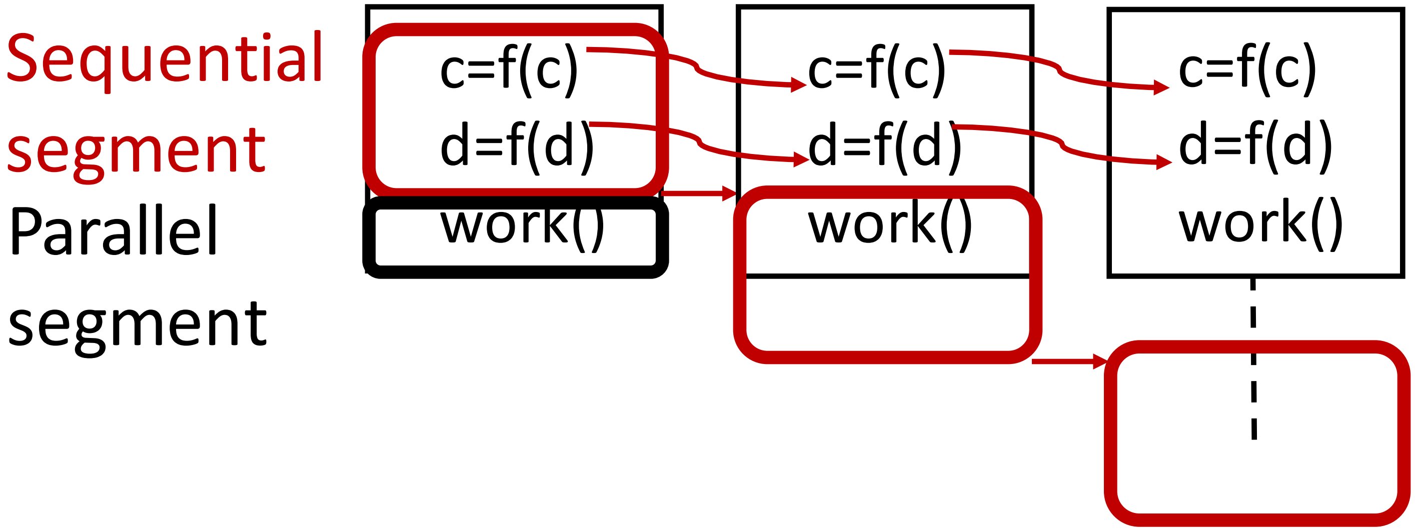
99% of time is spent in loops

Time

DOACROSS parallelism

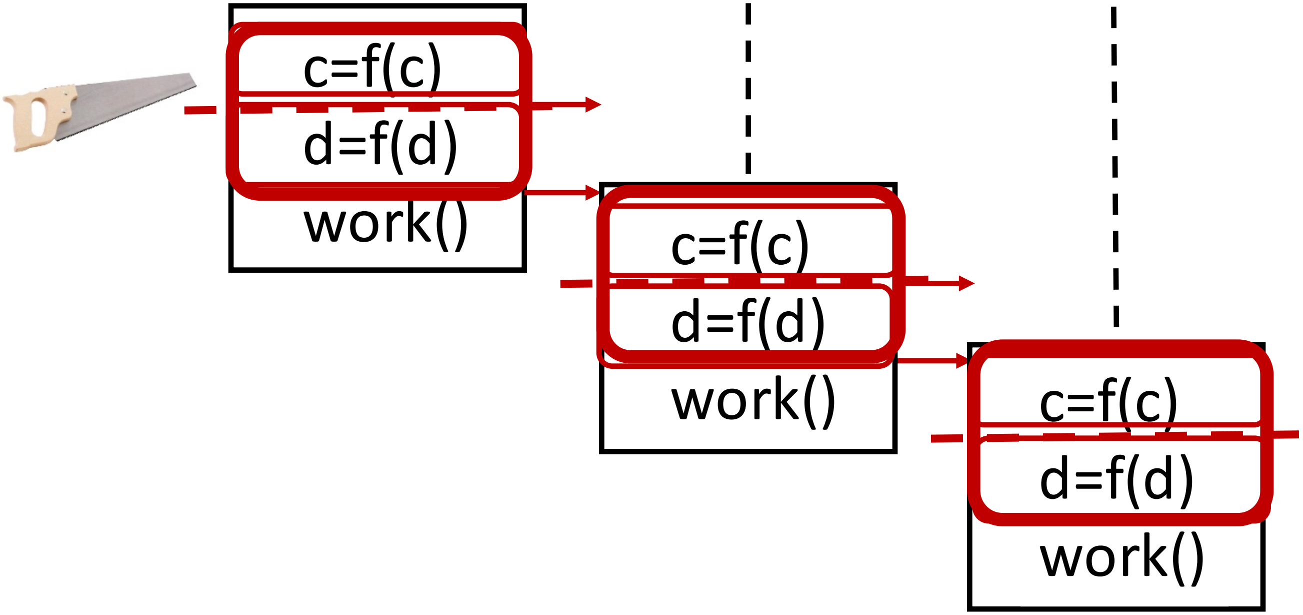


DOACROSS parallelism



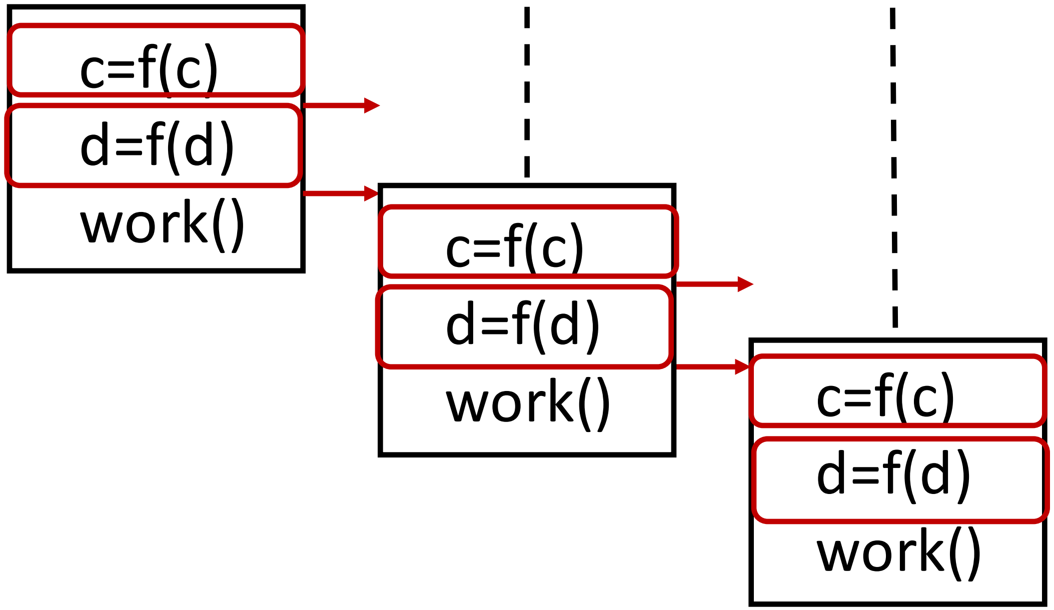
HELIX: DOACROSS for multicore

[*Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012*]



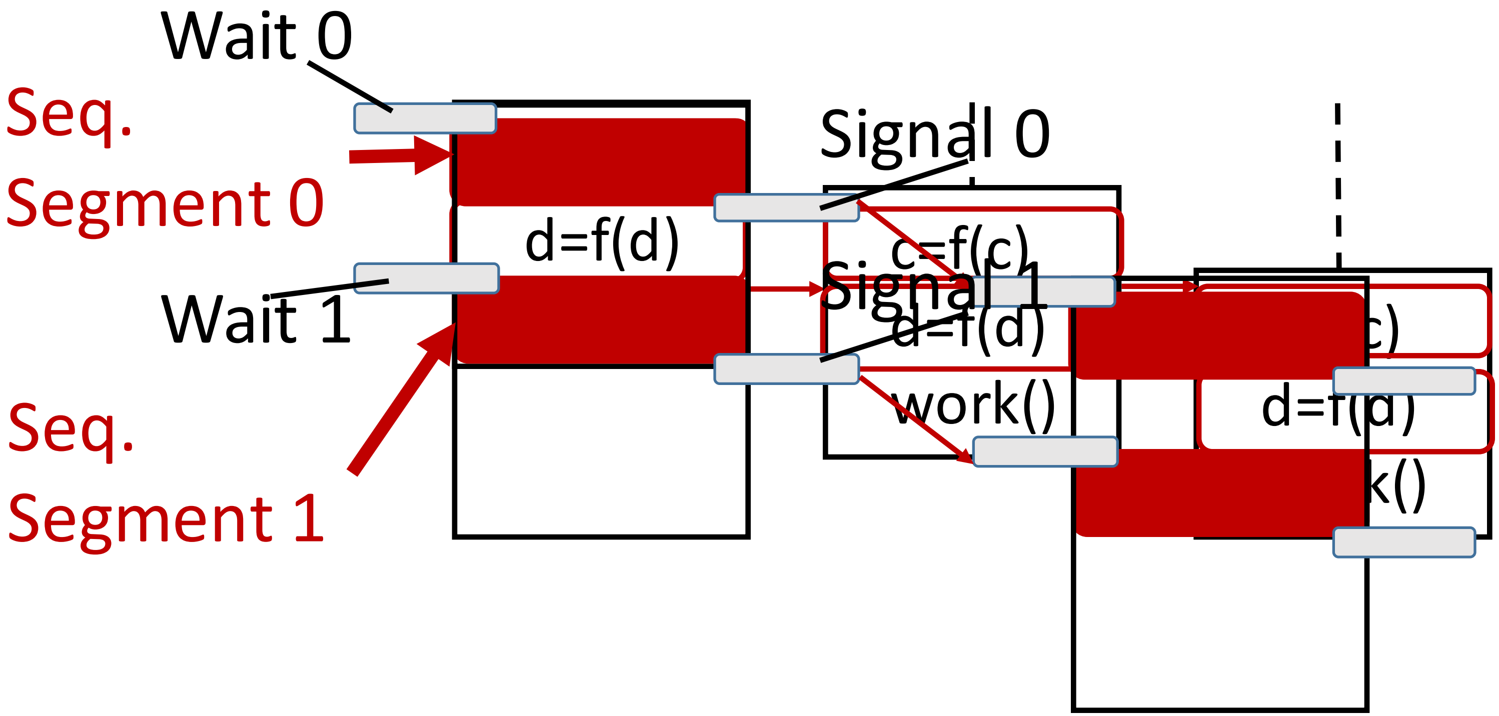
HELIX: DOACROSS for multicore

[Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012]



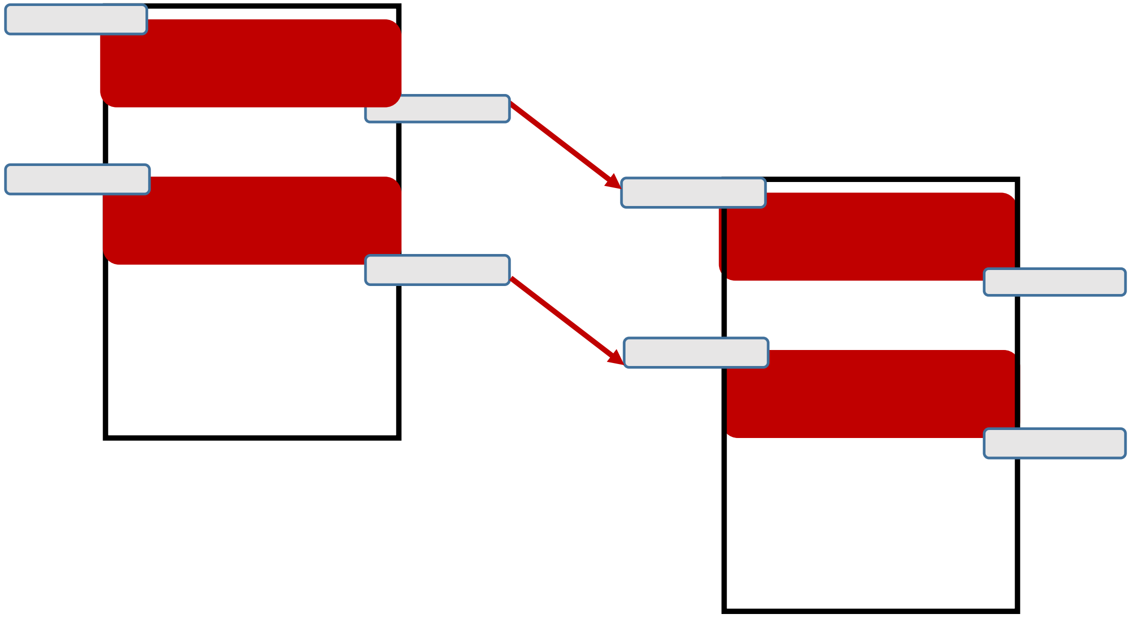
HELIX: DOACROSS for multicore

[*Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012*]



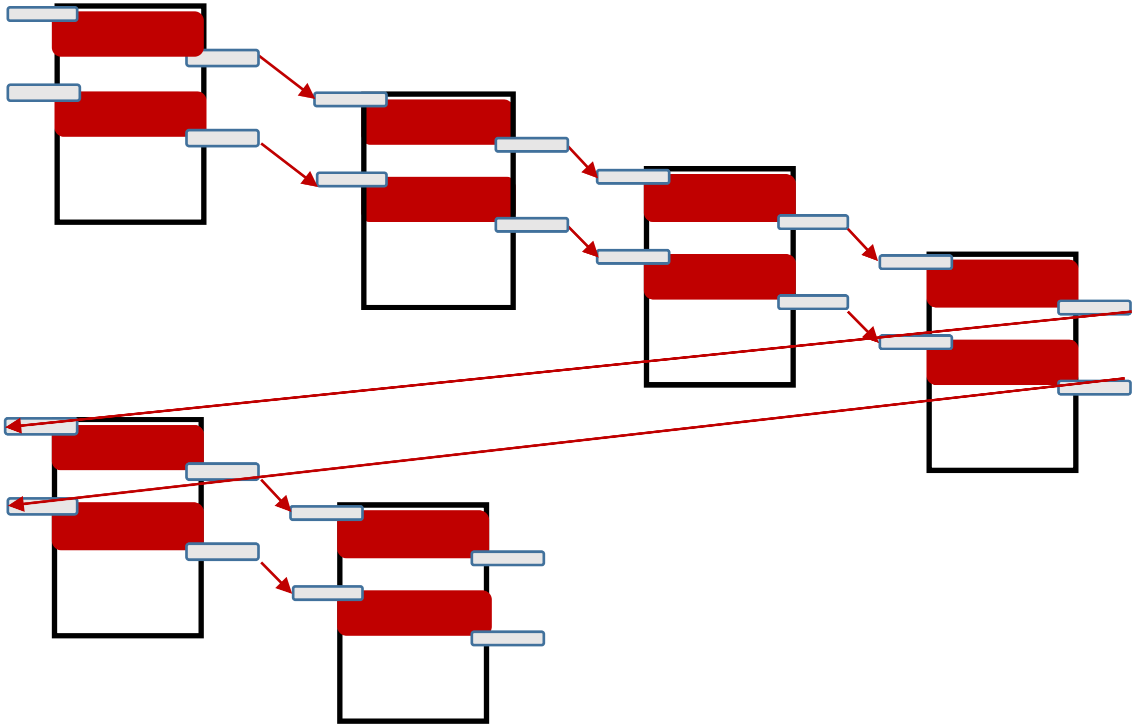
HELIX: DOACROSS for multicore

[*Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012*]

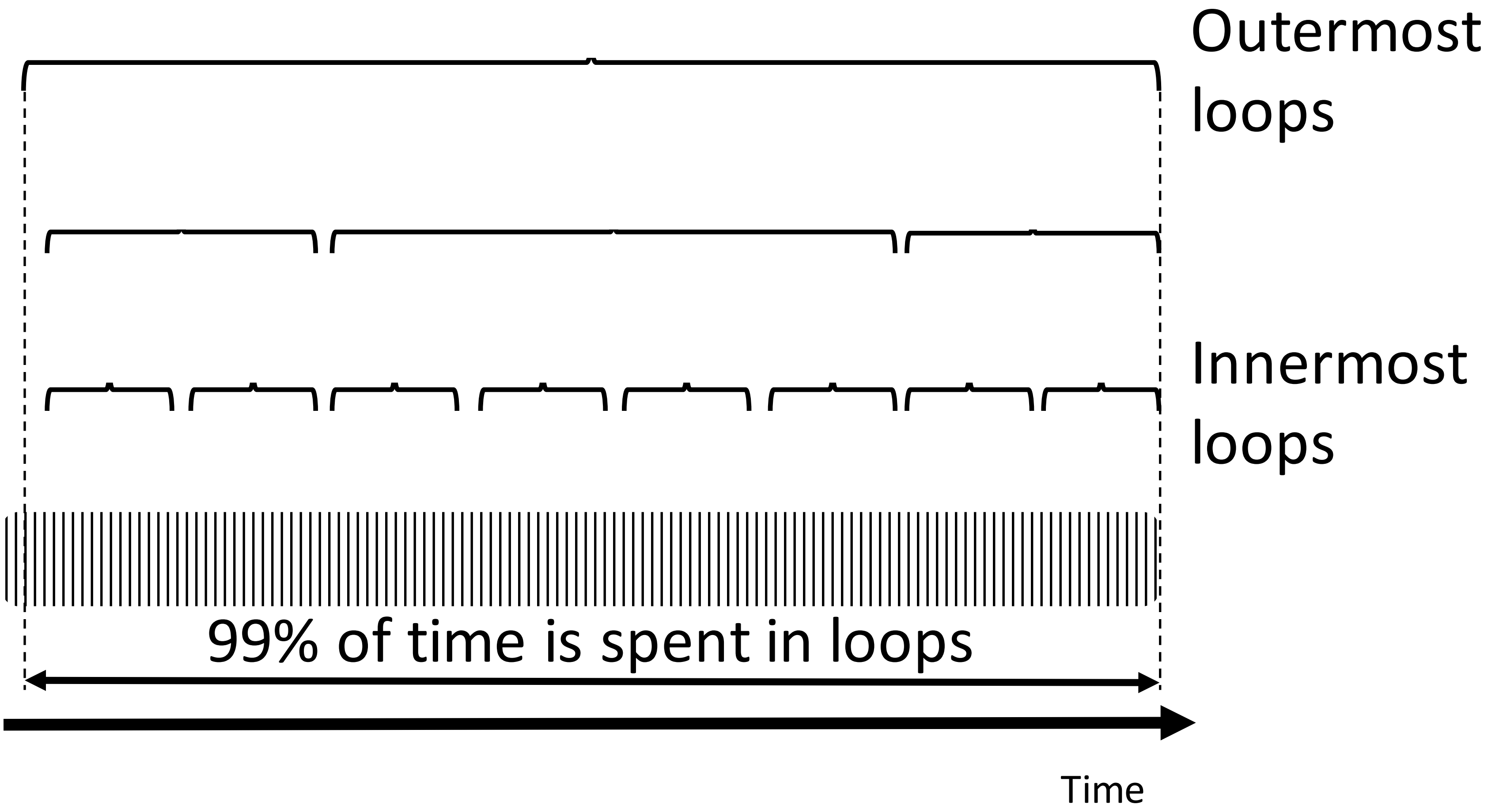


HELIX: DOACROSS for multicore

[*Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012*]



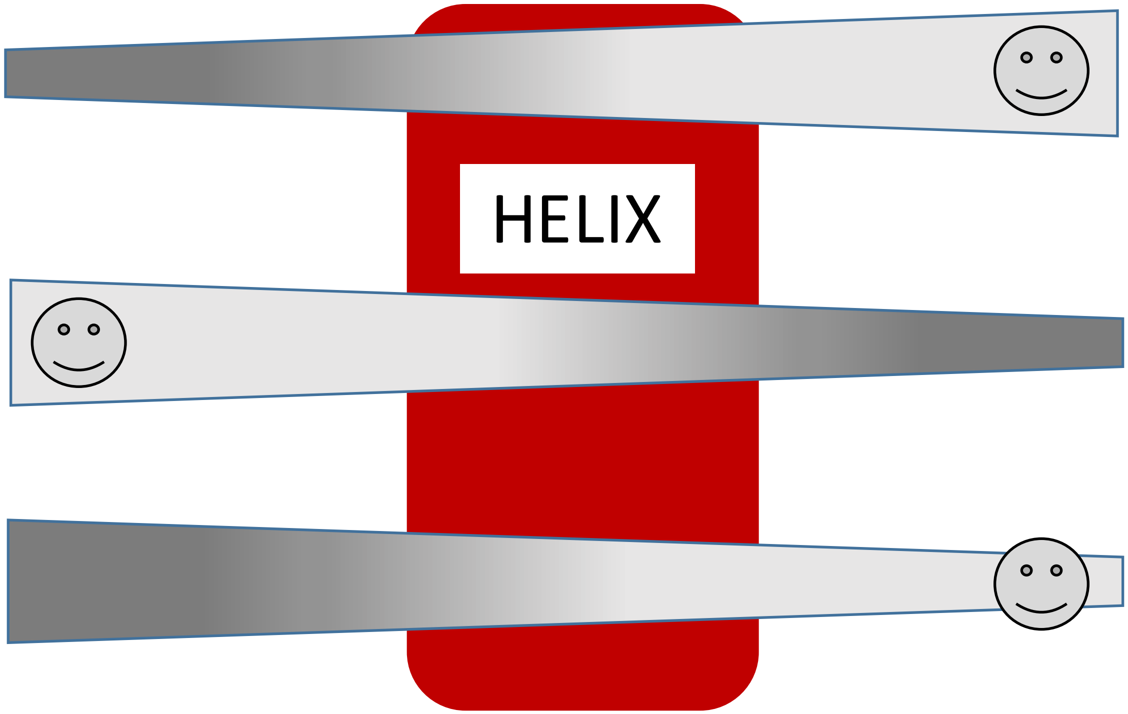
Parallelize loops
to parallelize a program



Parallelize loops to parallelize a program

Innermost loops

Outermost loops



HELIX

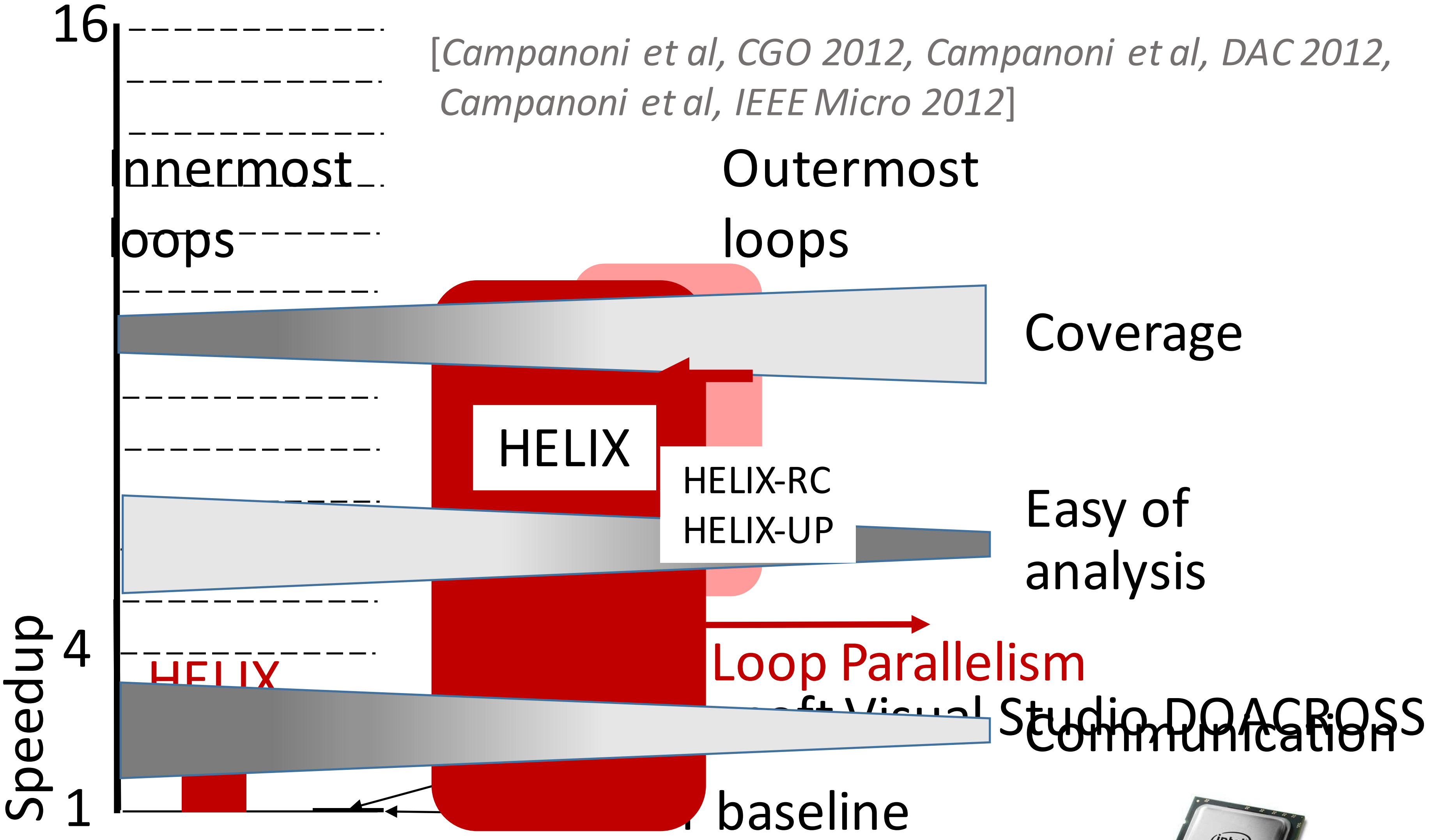
Coverage

Ease of analysis

Communication

HELIX: DOACROSS for multicore

[Campanoni et al, CGO 2012, Campanoni et al, DAC 2012, Campanoni et al, IEEE Micro 2012]

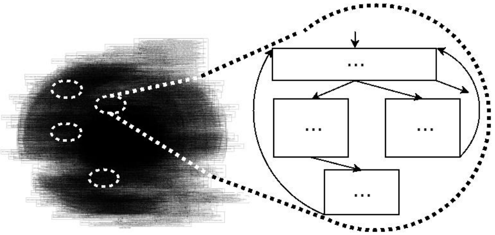


4-core Intel Nehalem

Outline

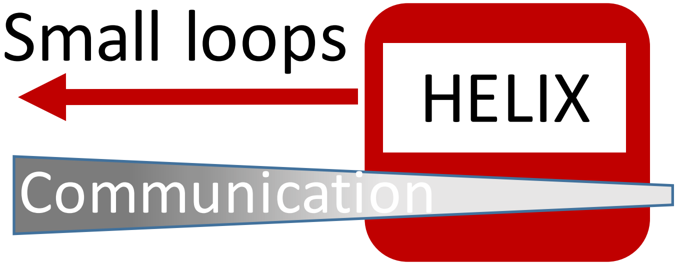
Small Loop Parallelism and HELIX

[CGO 2012
DAC 2012,
IEEE Micro 2012]



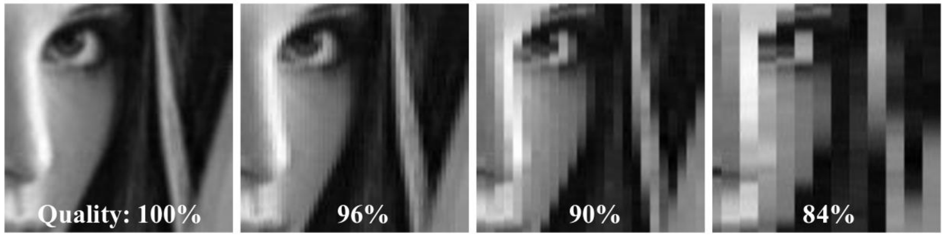
HELIX-RC: Architecture/Compiler Co-Design

[ISCA 2014]

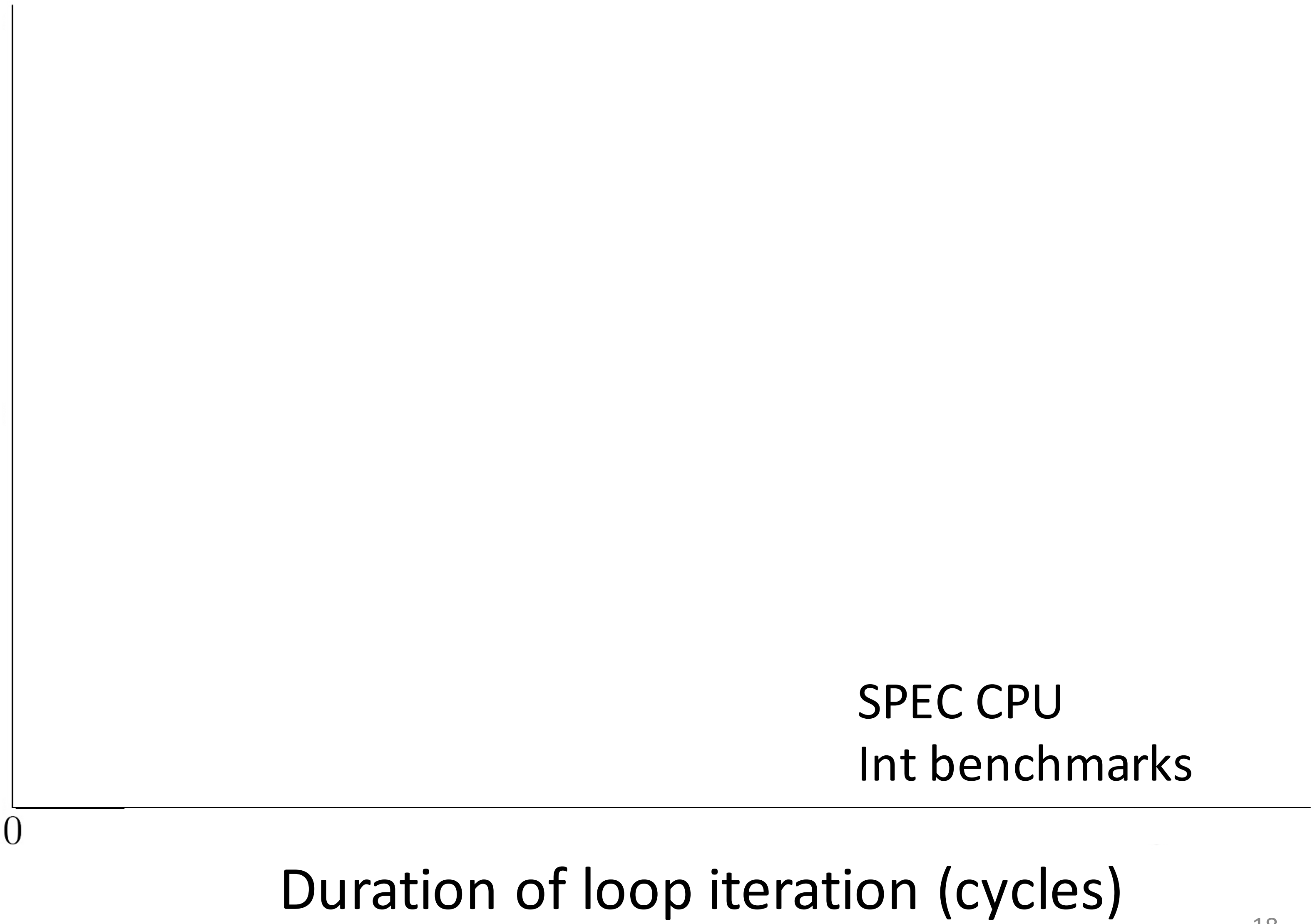


HELIX-UP: Unleash Parallelization

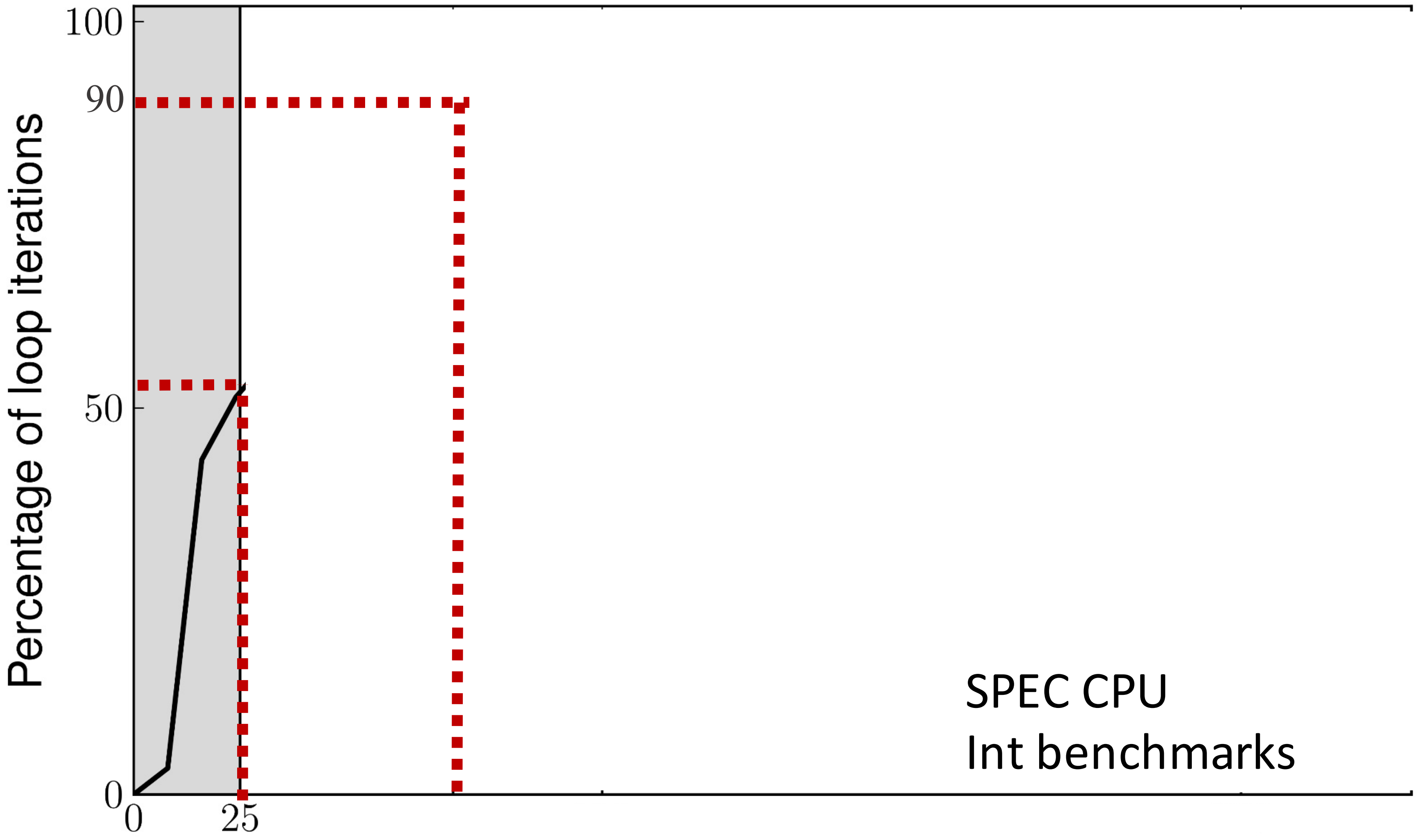
[CGO 2015]



SLP challenge: short loop iterations

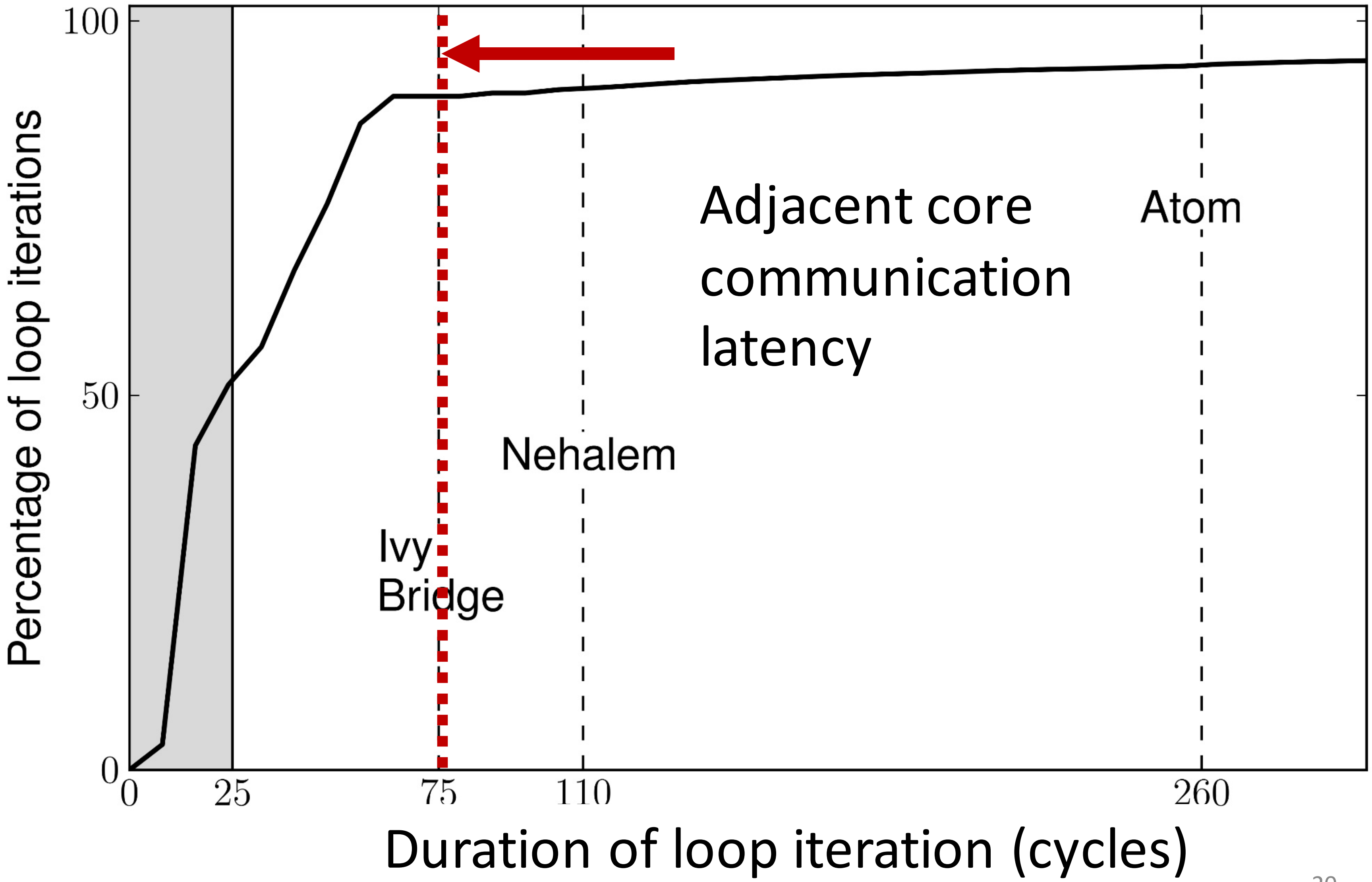


SLP challenge: short loop iterations



Duration of loop iteration (cycles)

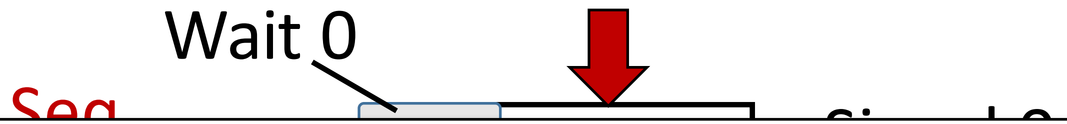
SLP challenge: short loop iterations



A compiler-architecture co-design to efficiently execute short iterations

Compiler

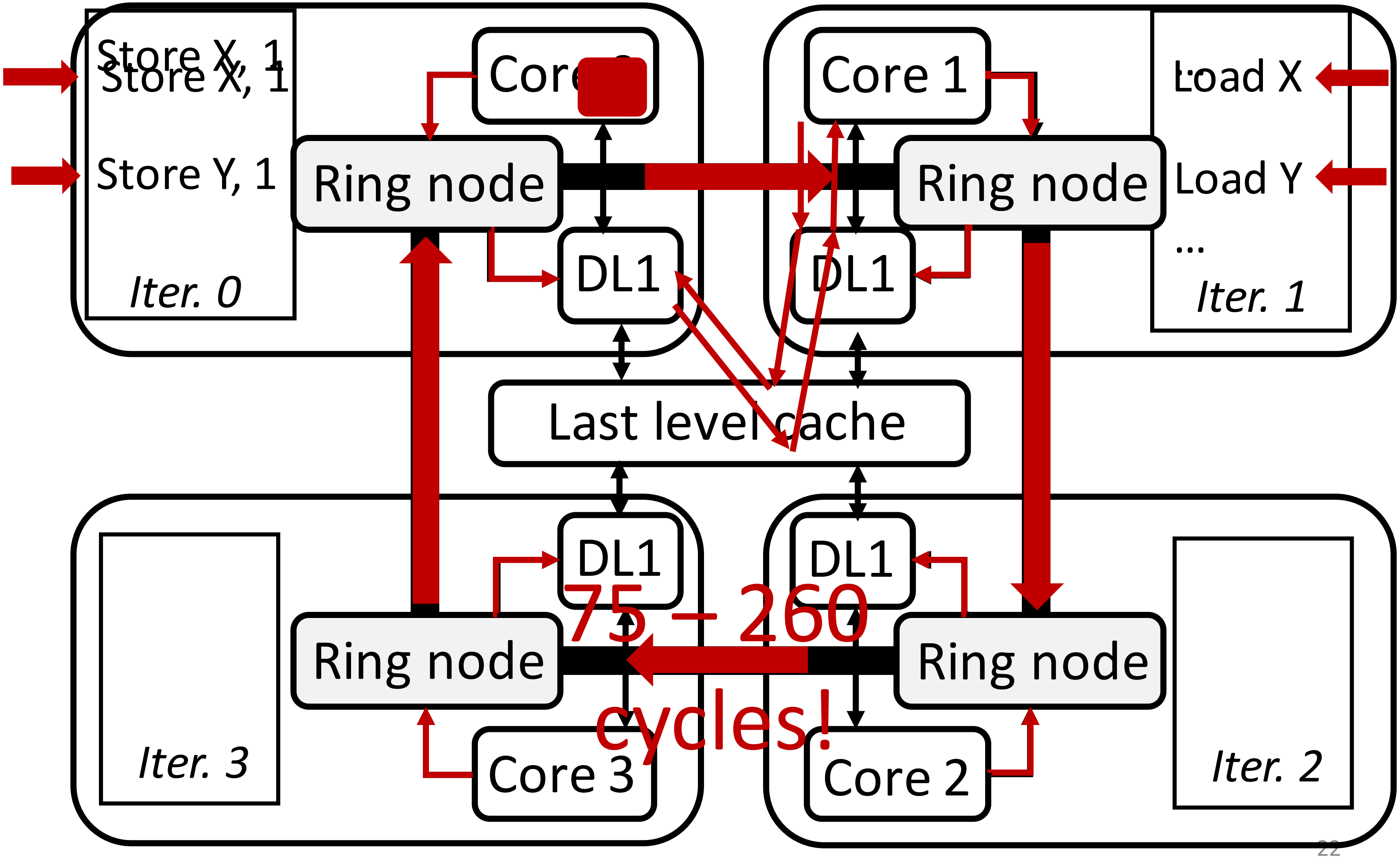
- Identify latency-critical code in each small loop
  - Code that generates shared data
- Expose information to the architecture



Architecture: Ring Cache

- Reduce the communication latency on the critical path

Light-weight enhancement of today's multicore architecture



Store X, 1

Store X, 1

Store Y, 1

Iter. 0

Core 0

Ring node

DL1

Core 1

Ring node

DL1

Load X

Load Y

…

Iter. 1

Last level cache

DL1

Ring node

Core 3

Iter. 3

DL1

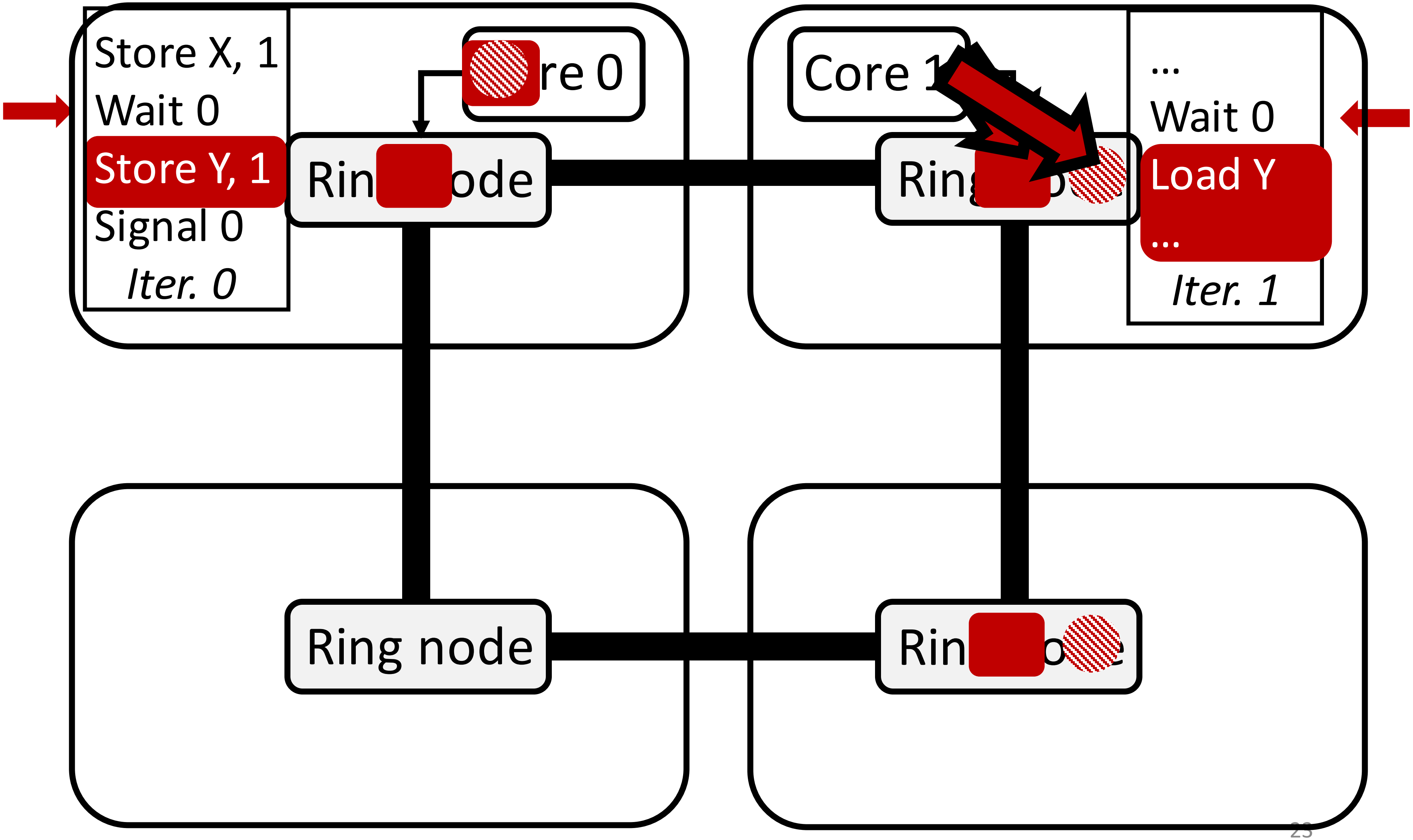
Ring node

Core 2

Iter. 2

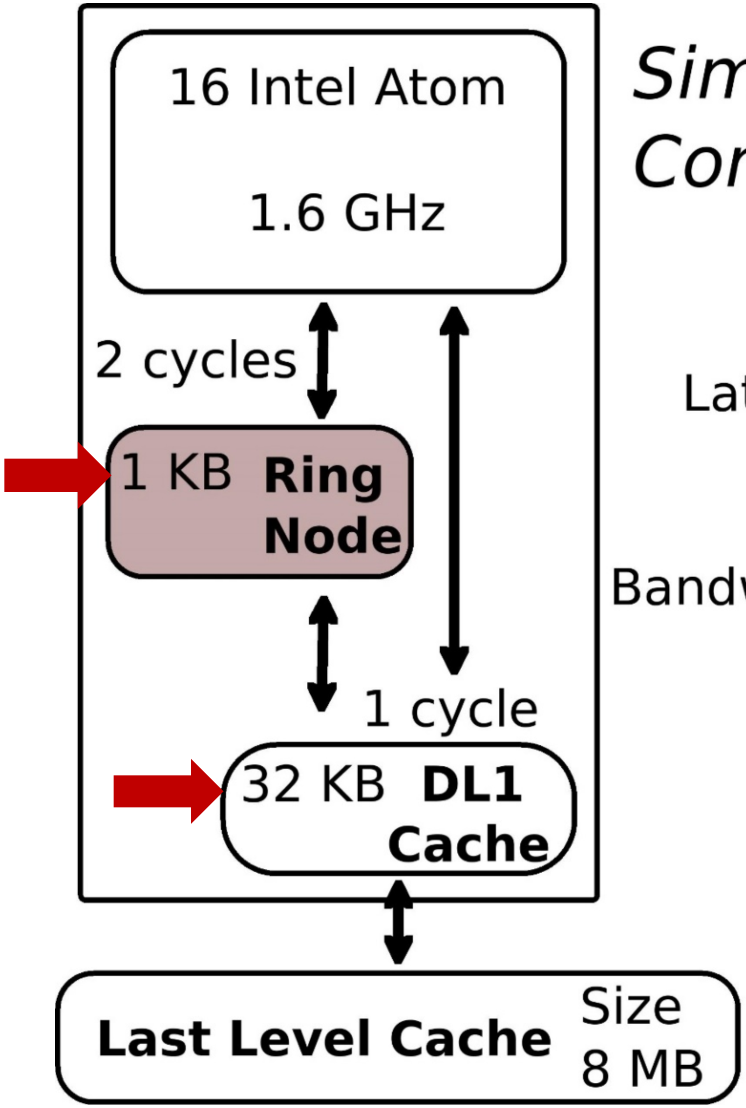
75 – 260 cycles!

Light-weight enhancement of today's multicore architecture



Simulator: XIOSim, DRAMSim

Compiler : ILDJIT (LLVM)

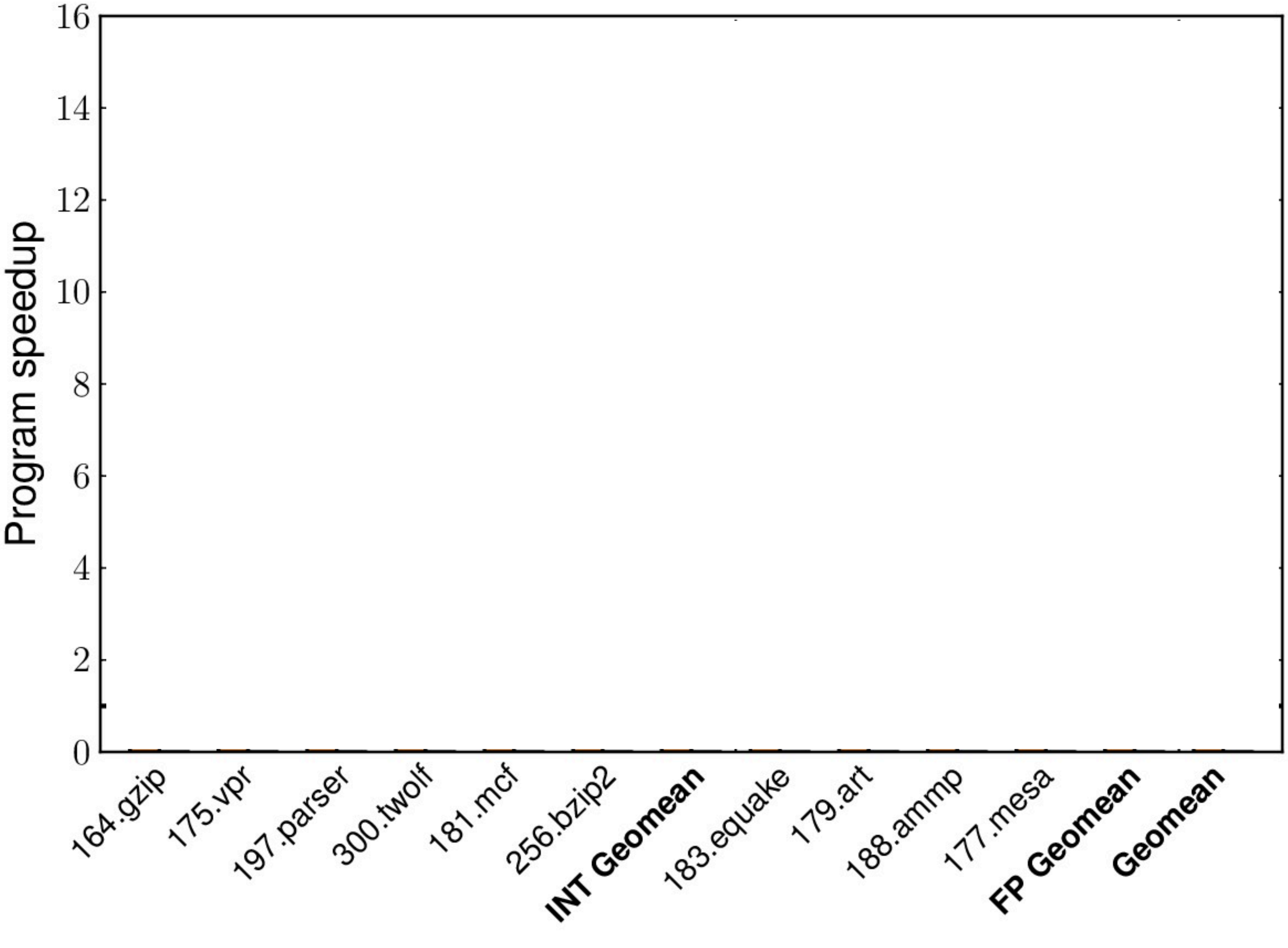


Latency: 1 cycle

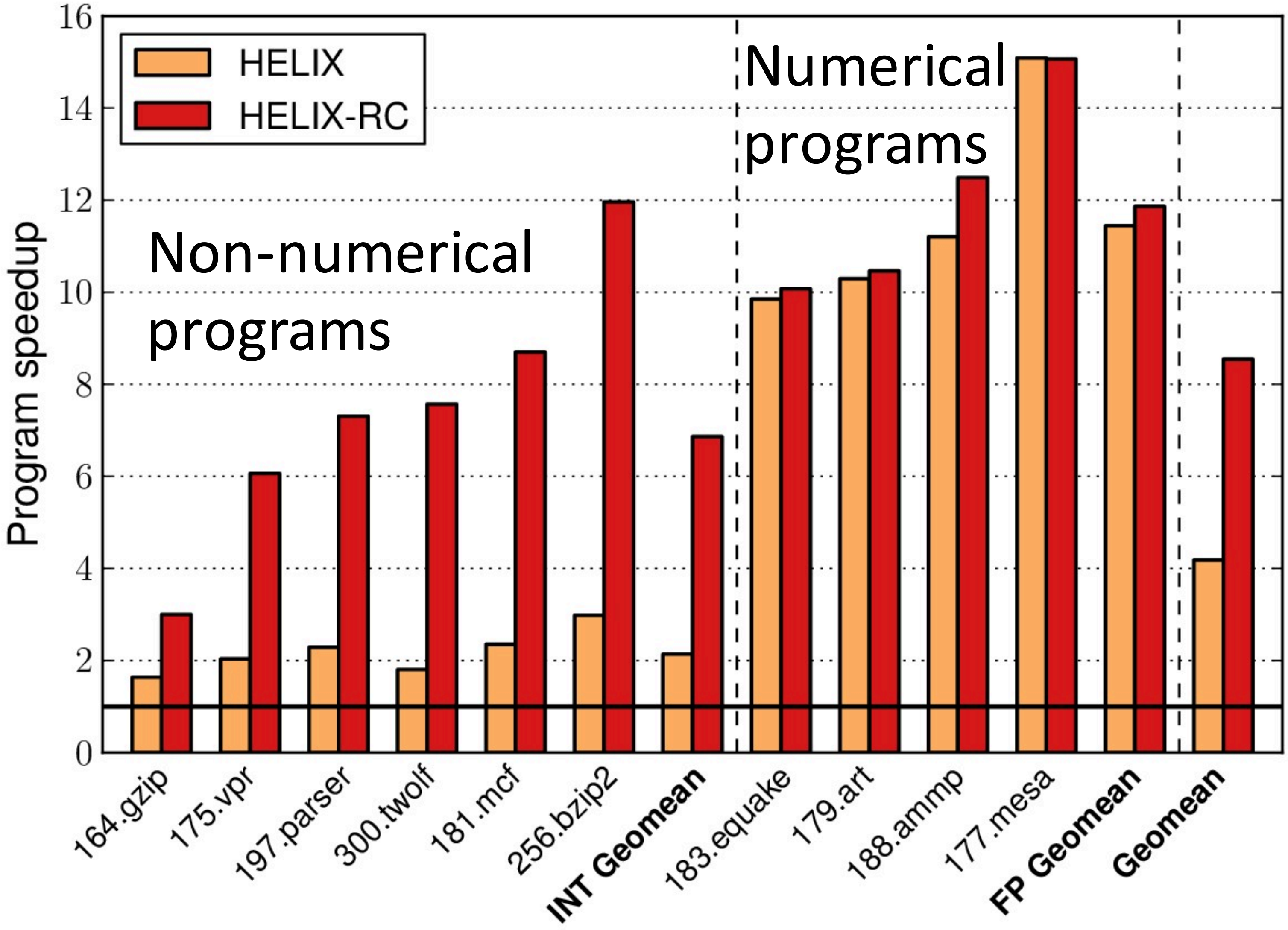
Bandwidth: 70 bits for signals
68 bits for data

98% hit rate

The importance of HELIX-RC



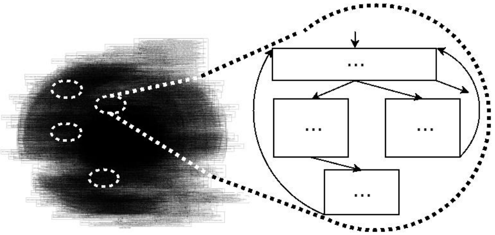
The importance of HELIX-RC



Outline

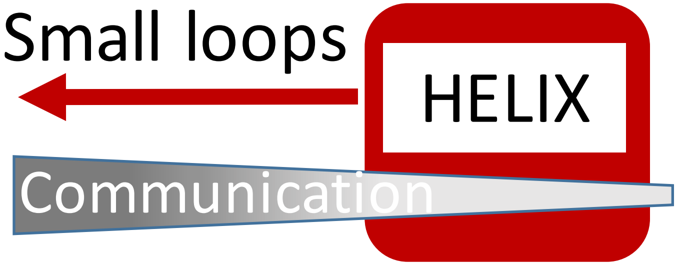
Small Loop Parallelism and HELIX

[CGO 2012
DAC 2012,
IEEE Micro 2012]



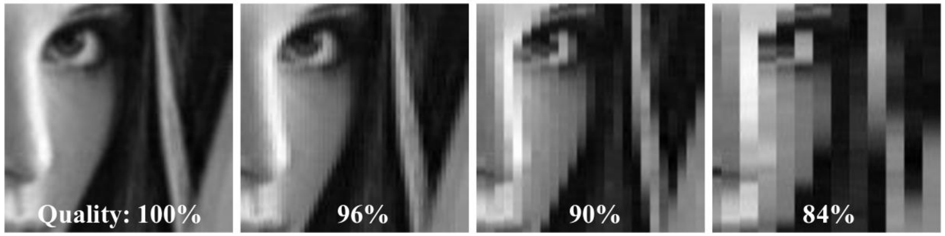
HELIX-RC: Architecture/Compiler Co-Design

[ISCA 2014]



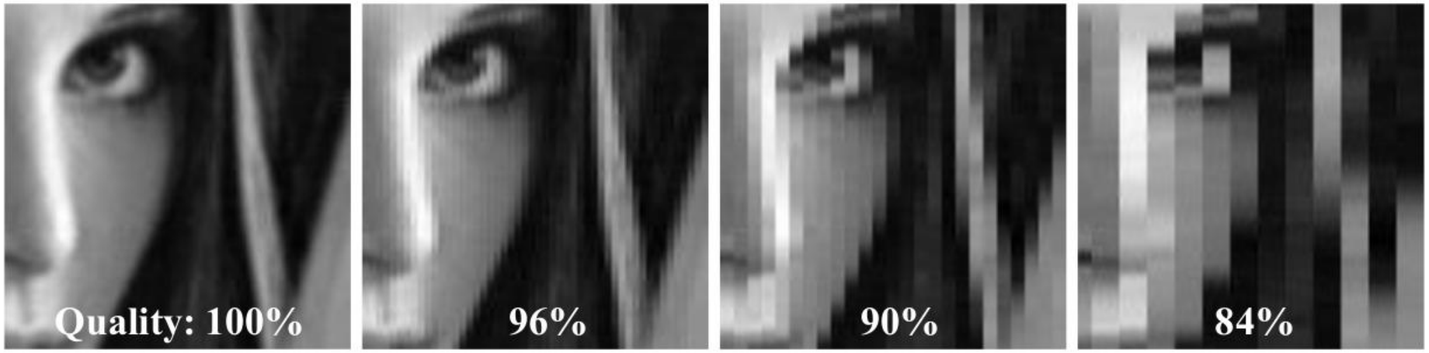
HELIX-UP: Unleash Parallelization

[CGO 2015]



Opportunity:
relax program semantics

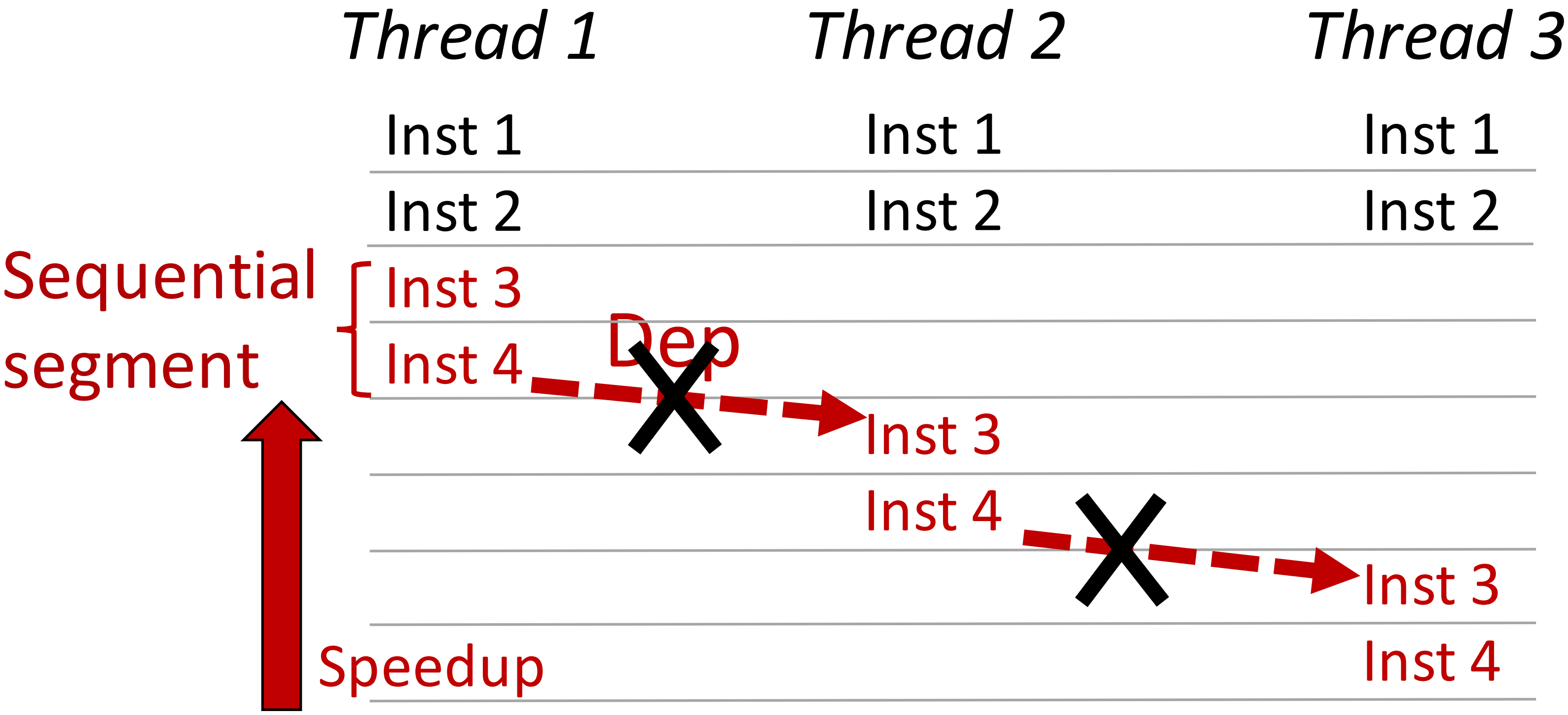
- Some workloads tolerate output distortion



- Output distortion is workload-dependent

Relaxing transformations remove performance bottlenecks

- Sequential bottleneck



Thread 1 *Thread 2* *Thread 3*

Inst 1 Inst 1 Inst 1

Inst 2 Inst 2 Inst 2

Sequential segment

Inst 3

Inst 4

Dep

Inst 3

Inst 4

Inst 3

Inst 4

Speedup

Relaxing transformations remove performance bottlenecks

- Sequential bottleneck

- Communication bottleneck

- Data locality bottleneck

Relaxing transformations remove performance bottlenecks

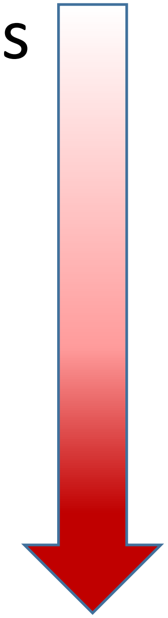
No relaxing transformations

Relaxing transformation 1

Relaxing transformation 2

…

Relaxing transformation k



No output distortion

Baseline performance

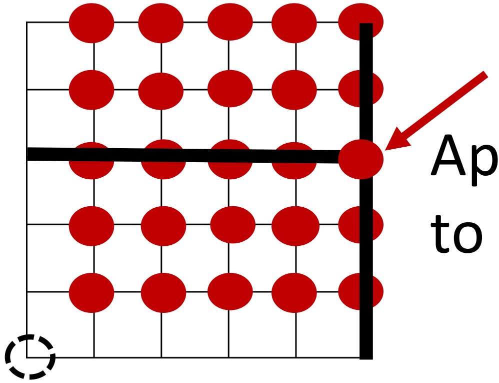
Max output distortion

Max performance

Design space of HELIX-UP

Apply relaxing transformation 5 to code region 2

- ○ Performance
- ○ Energy saved
- ○ Output distortion



Code region 1

Apply relaxing transformation 3 to code region 1

Code region 2

1) User provides output distortion limits
2) System finds the best configuration
3) Run parallelized code with that configuration

Pruning the design space

Empirical observation:
Transforming a code region
affects only the loop it belongs to

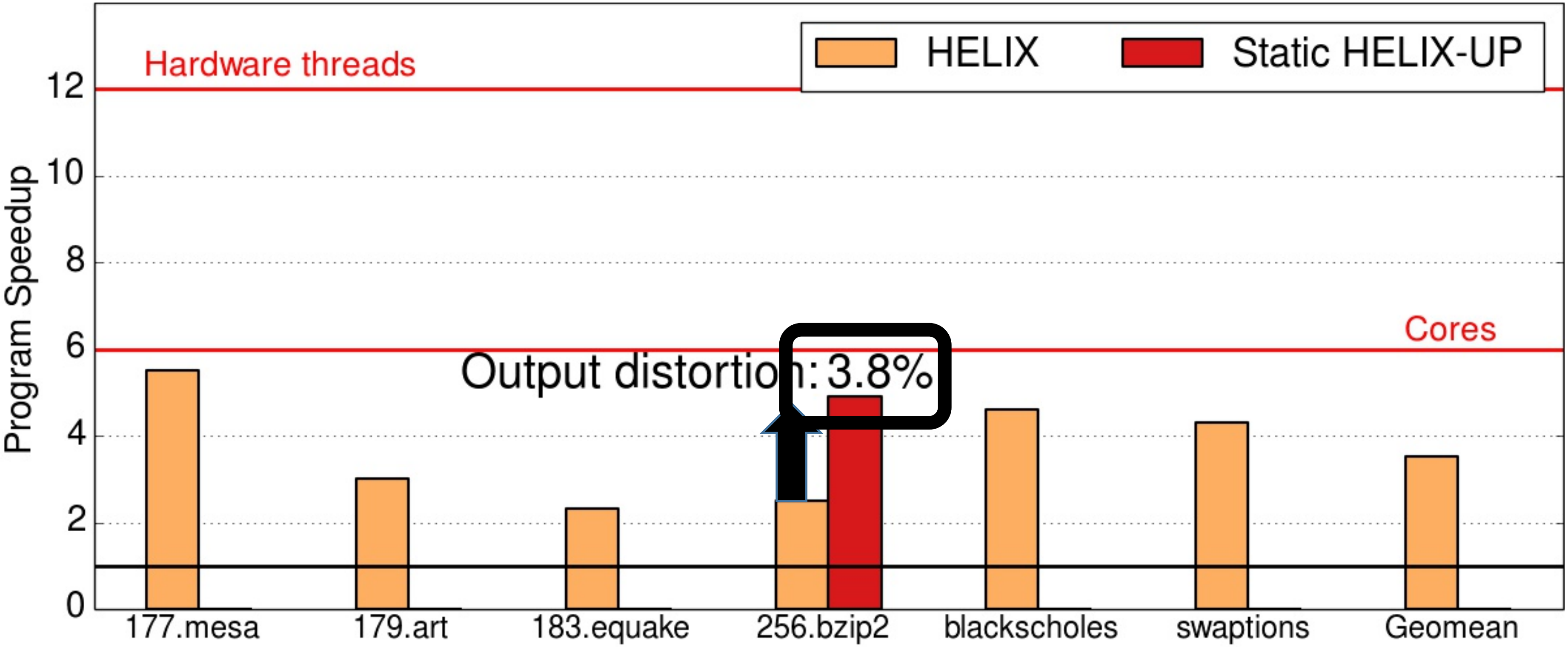
50 loops, 2 code regions per loop
2 transformations per code region

Complete space = 2^{100}
Pruned space = 50 * (2^2) = 200

How well does HELIX-UP perform?

HELIX-UP unblocks extra parallelism with small output distortions

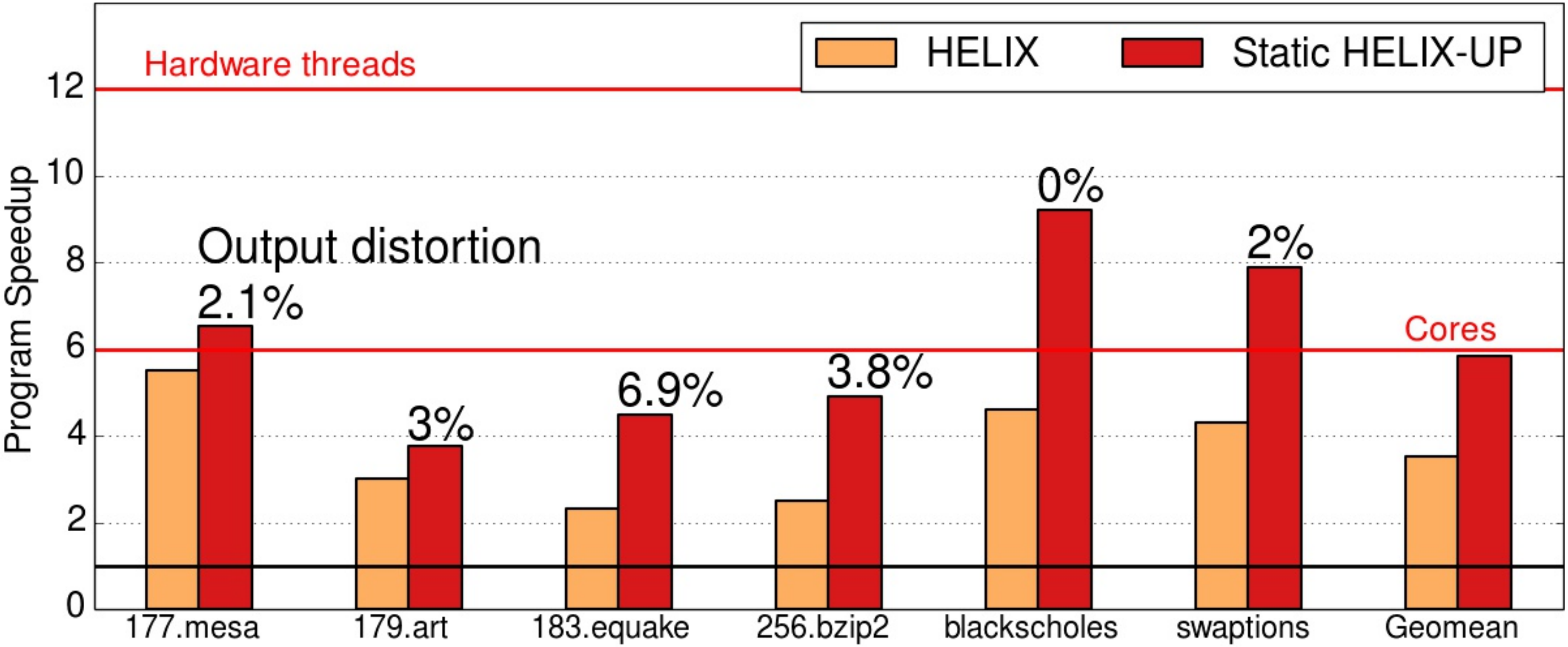
HELIX: no relaxing transformations



Output distortion: 3.8%

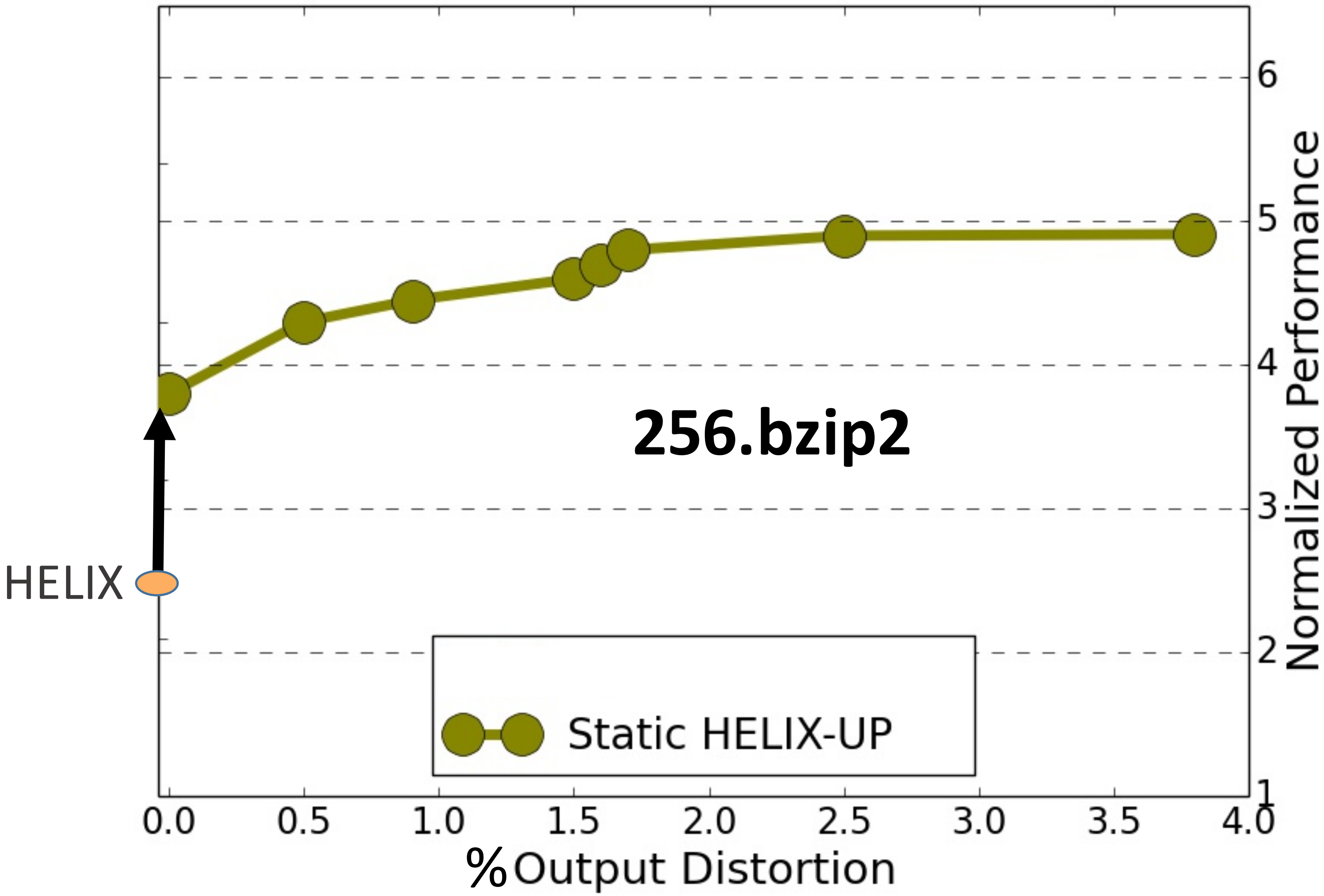
Nehalem 6 cores
2 threads per core

HELIX-UP unblocks extra parallelism with small output distortions



Nehalem 6 cores
2 threads per core

Performance/distortion tradeoff

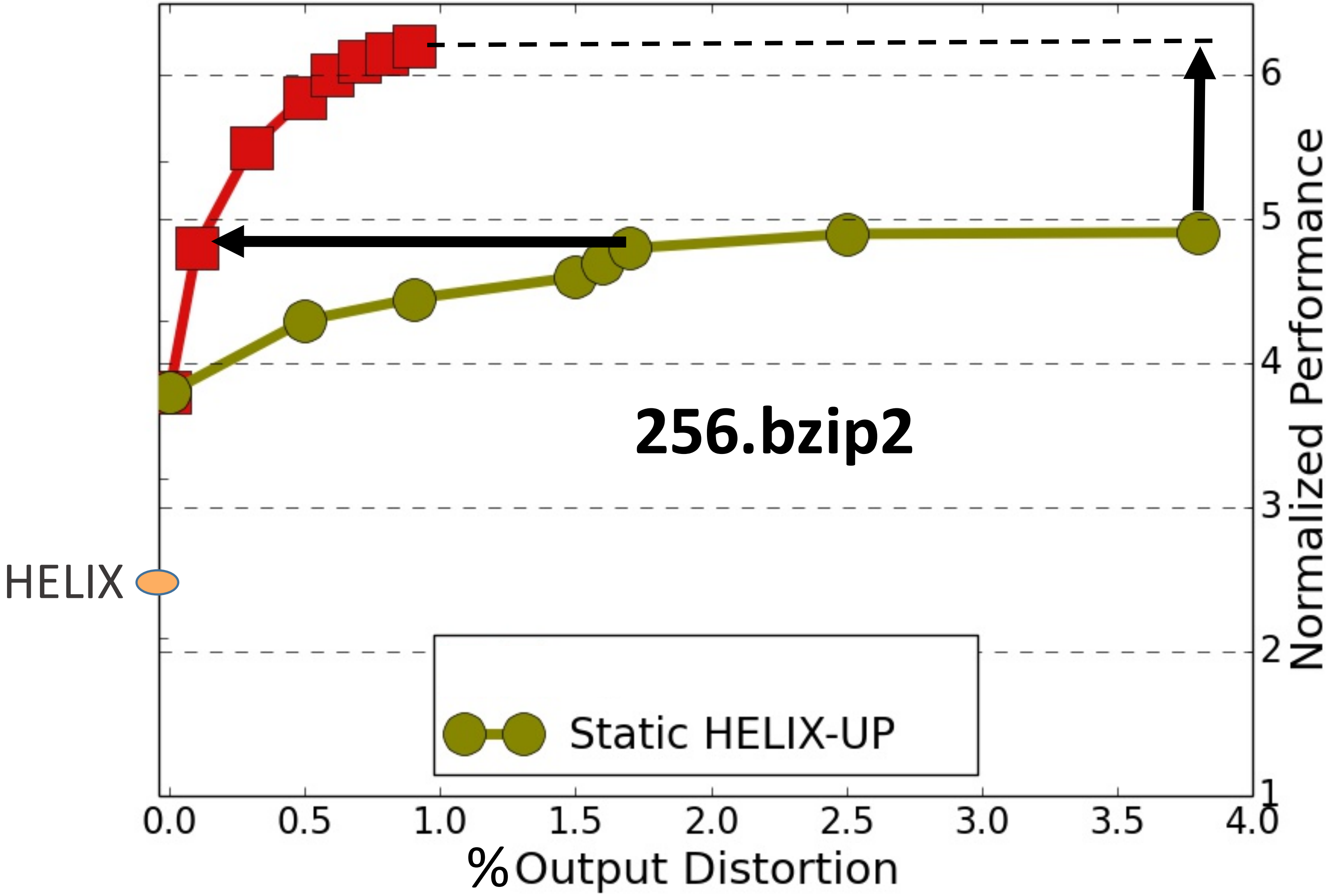


Run time code tuning

- Static HELIX-UP decides
  how to transform the code based on
  profile data averaged over inputs

- The runtime reacts to transient bottlenecks
  by adjusting code accordingly

Adapting code at run time unlocks more parallelism

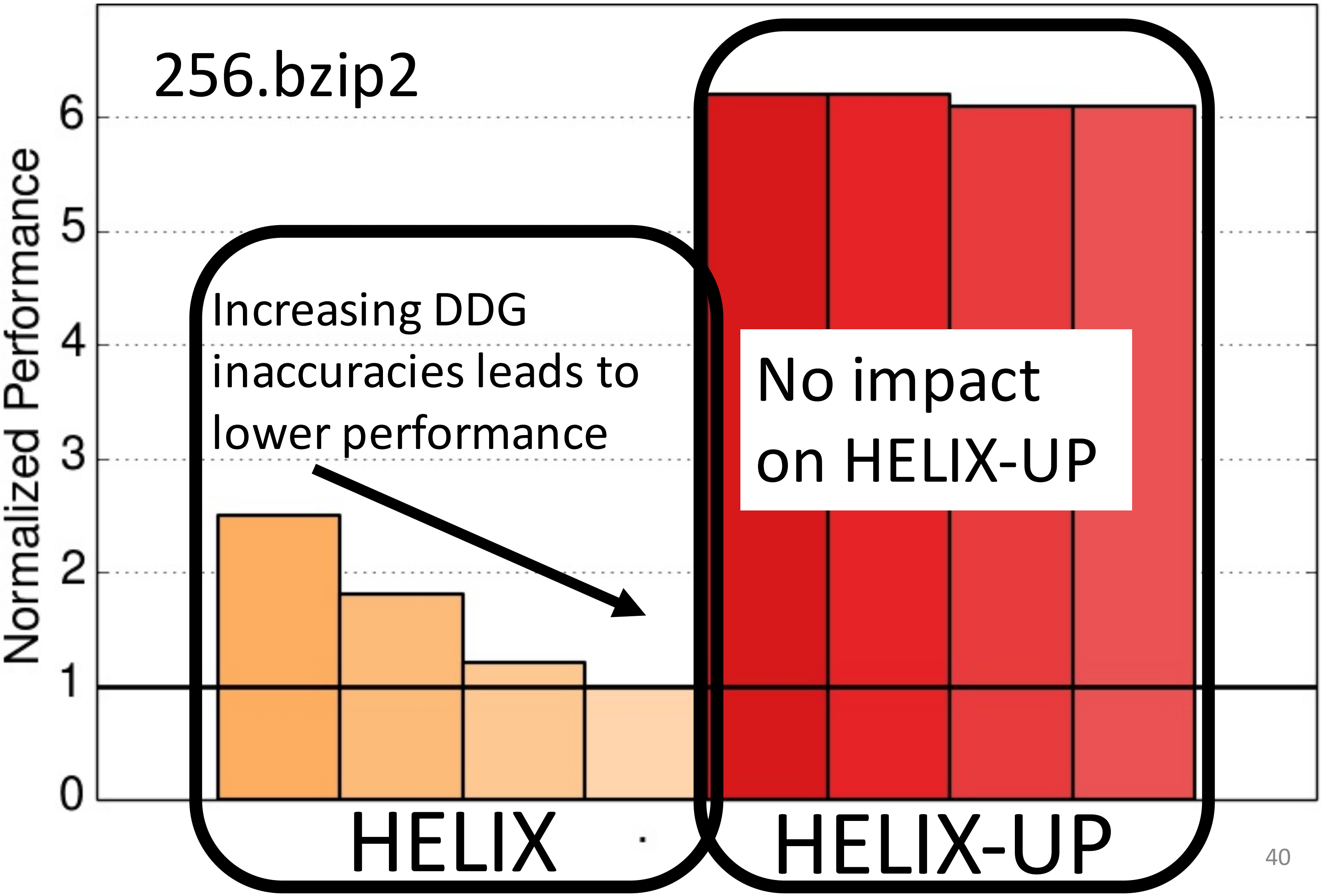


HELIX-UP improves more than just performance

- Robustness to DDG inaccuracies

- Consistent performance across platforms

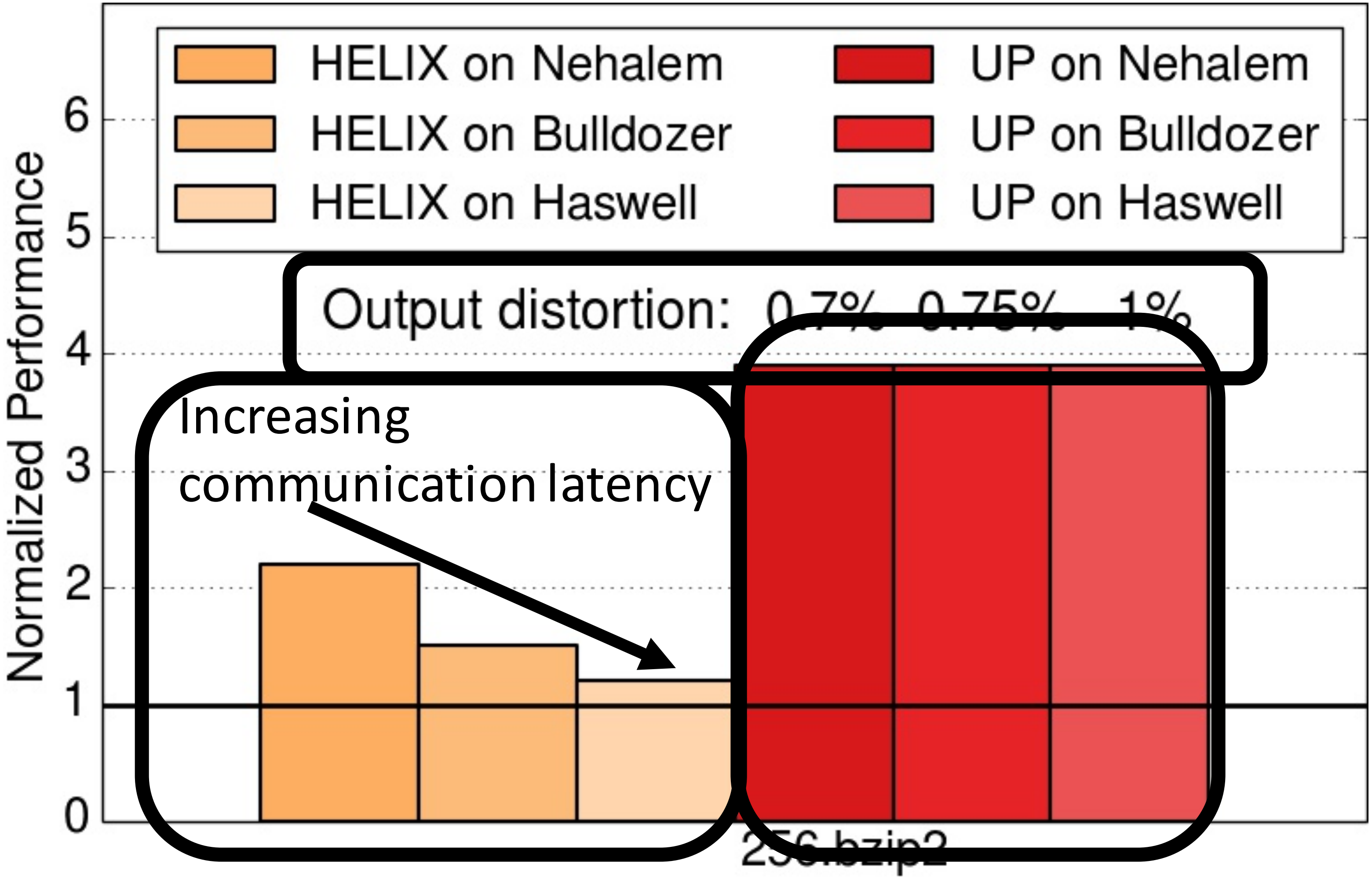
Relaxed transformations to be robust to DDG inaccuracies



Increasing DDG inaccuracies leads to lower performance

No impact on HELIX-UP

Relaxed transformations for consistent performance

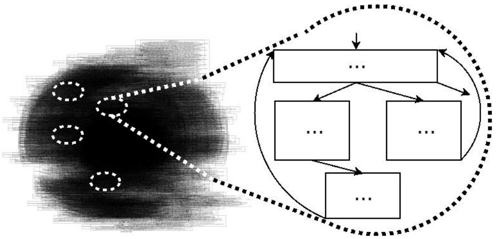


Output distortion: 0.7% 0.75% 1%

Increasing communication latency

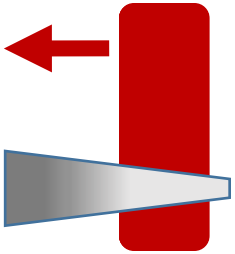
Small Loop Parallelism and HELIX

- *Parallelism hides in small loops*

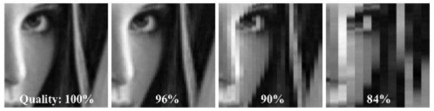


HELIX-RC: Architecture/Compiler Co-Design

- *Irregular programs require low latency*



HELIX-UP: Unleash Parallelization

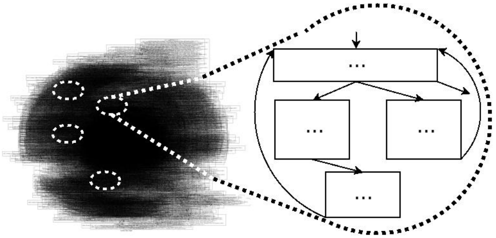


- *Tolerating distortions boosts parallelization*

Thank you!

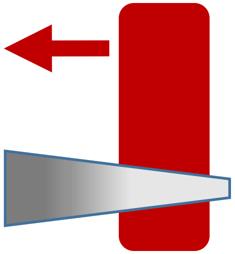
Small Loop Parallelism and HELIX

- *Parallelism hides in small loops*

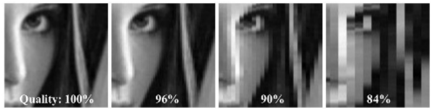


HELIX-RC: Architecture/Compiler Co-Design

- *Irregular programs require low latency*



HELIX-UP: Unleash Parallelization



- *Tolerating distortions boosts parallelization*