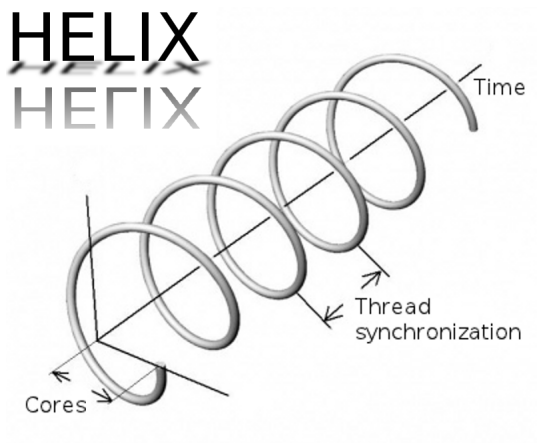
HELIX-RC
An Architecture-Compiler Co-Design for Automatic Parallelization of Irregular Programs

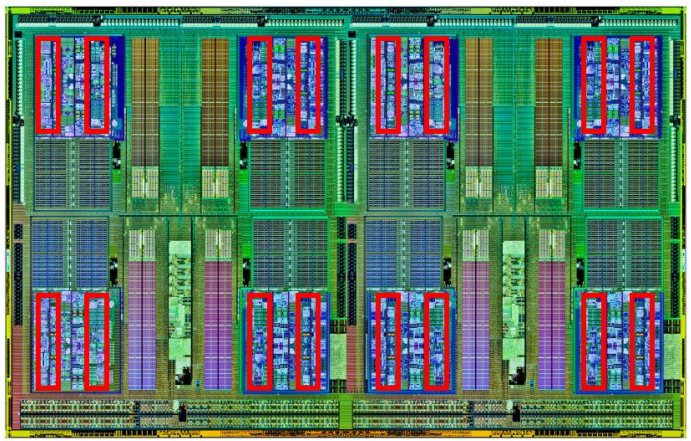
Simone Campanoni, Kevin Brownell, Svilen Kanev
Timothy M. Jones, Gu-Yeon Wei, David Brooks

HARVARD
School of Engineering and Applied Sciences



Motivation

Today's commodity platforms include multiple cores

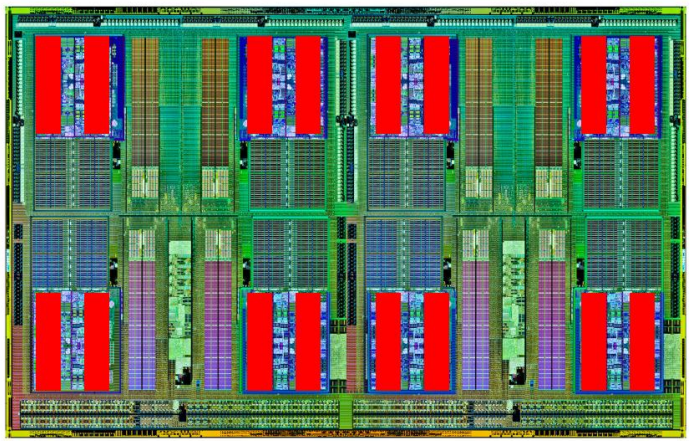


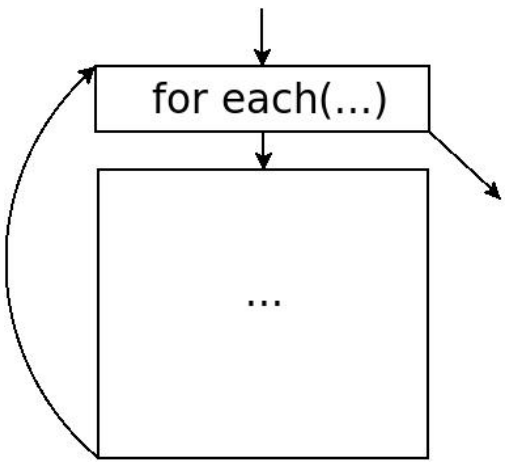
Use multiple cores for a **single** program

Distribute **loop iterations** among cores

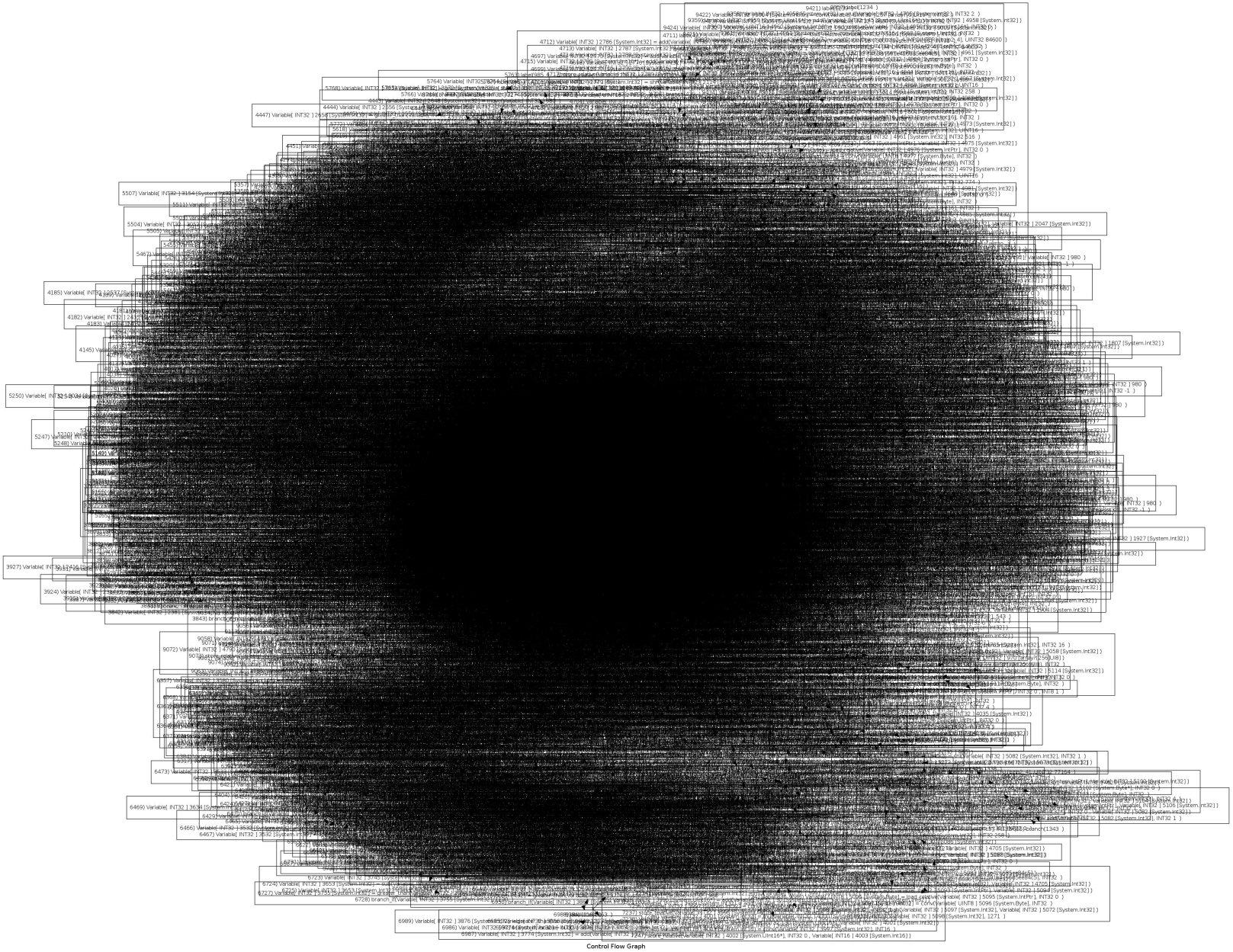
Motivation

Today's commodity platforms include multiple cores



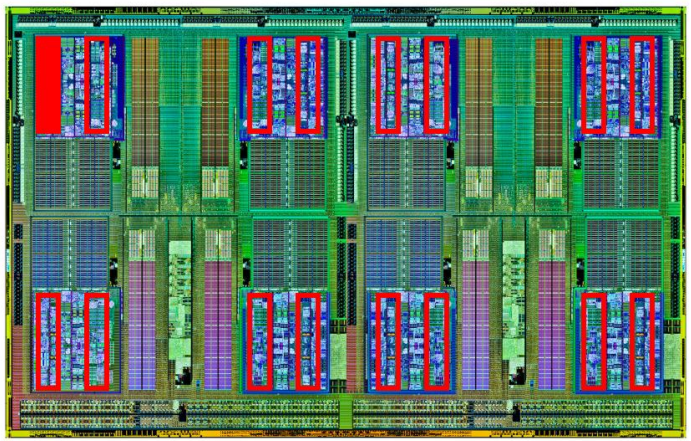


Motivation



Motivation

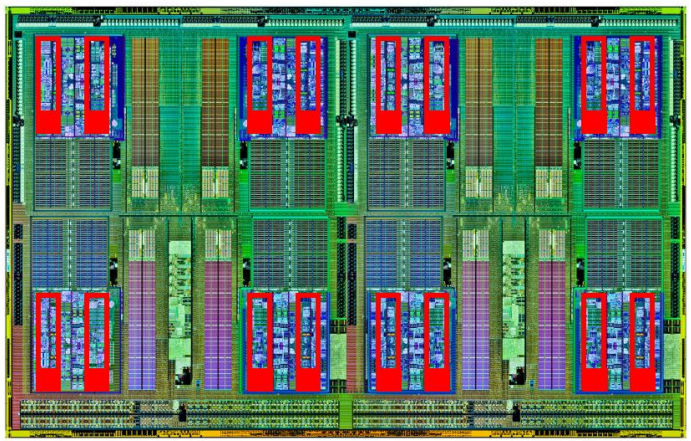
Today's commodity platforms include multiple cores

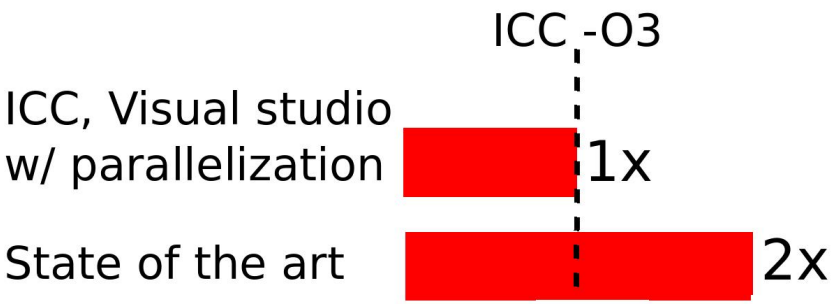


Program Speedup

Motivation

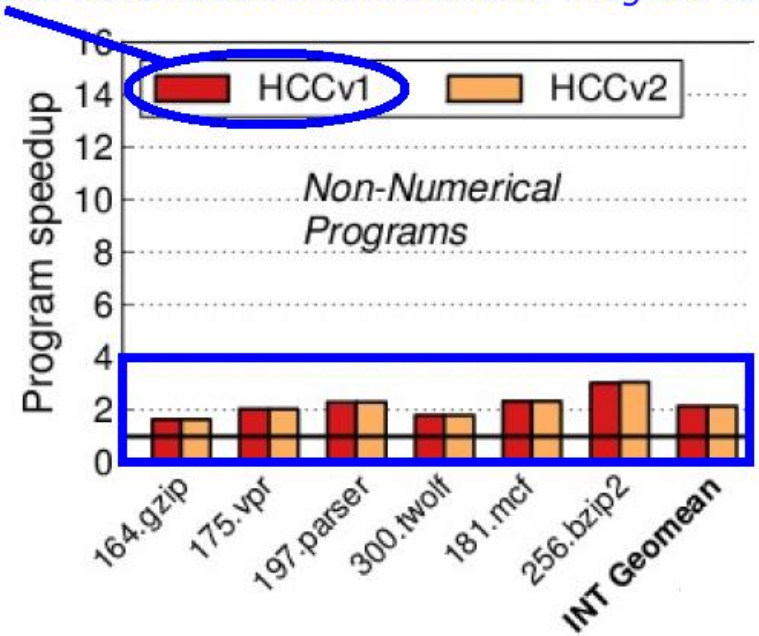
Today's commodity platforms include multiple cores

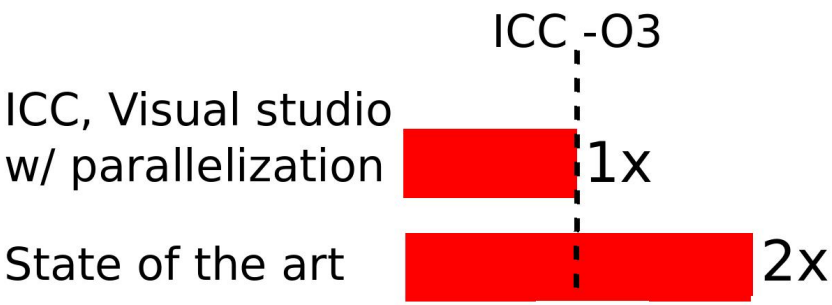




Motivation

CGO 2012, HELIX: Automatic Parallelization of Irregular Programs for Chip Multiprocessing

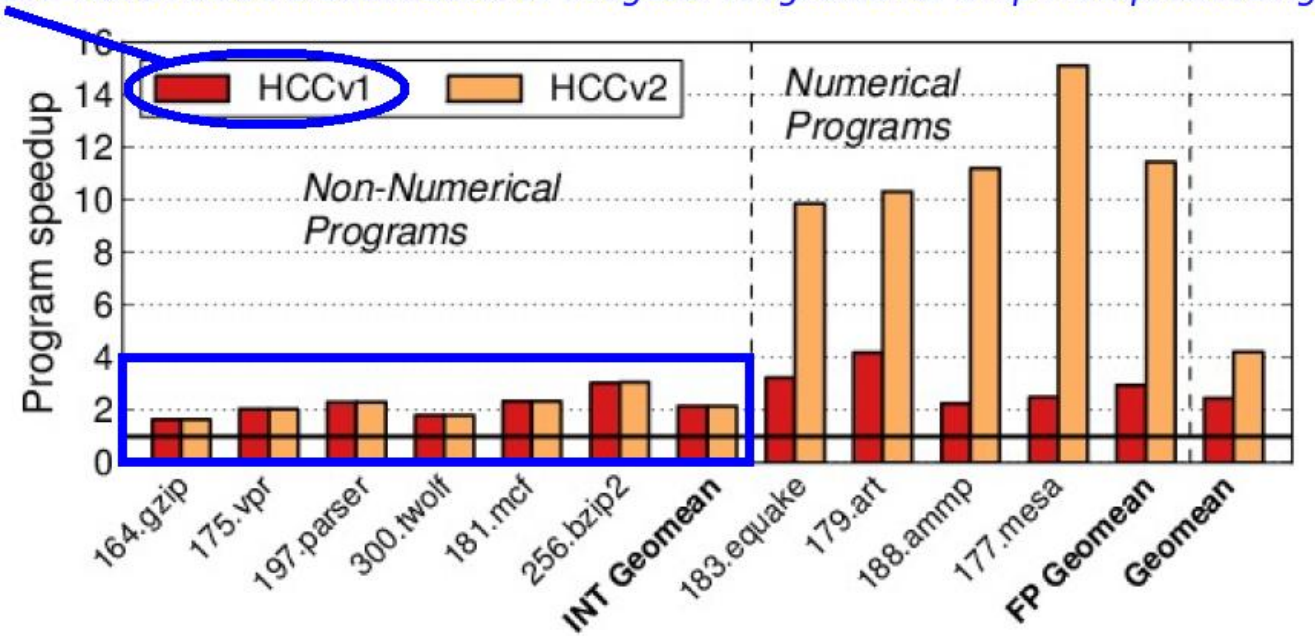


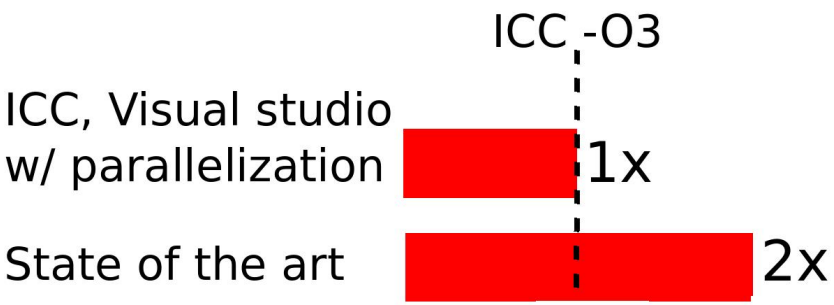


Program Speedup

Motivation

CGO 2012, HELIX: Automatic Parallelization of Irregular Programs for Chip Multiprocessing

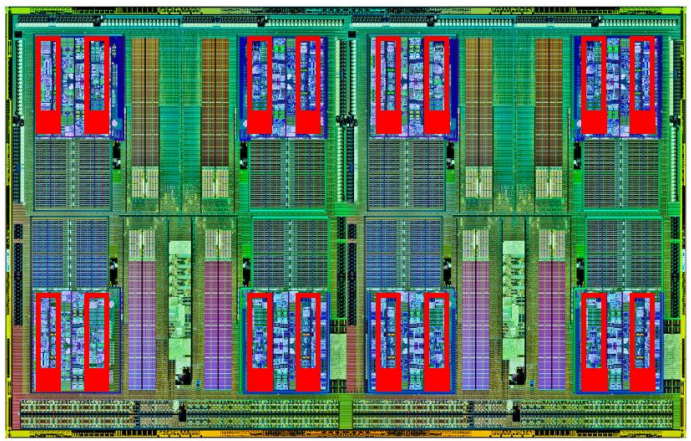


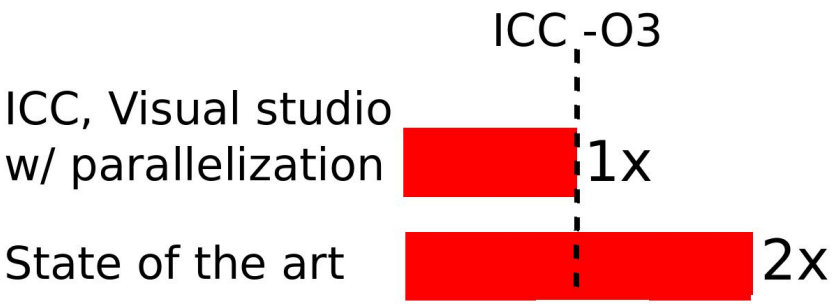


Program Speedup

Motivation

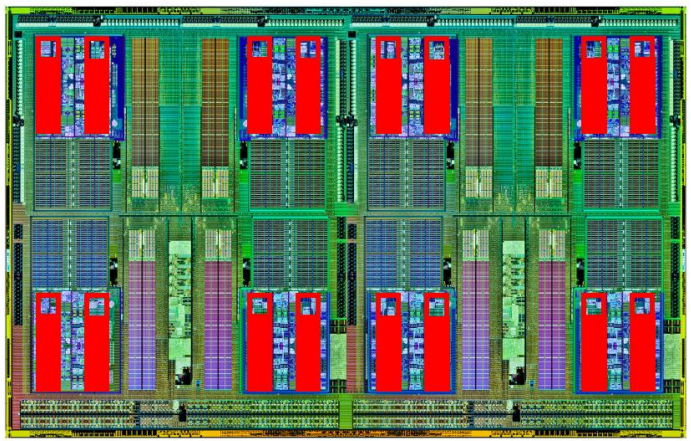
Today's commodity platforms include multiple cores

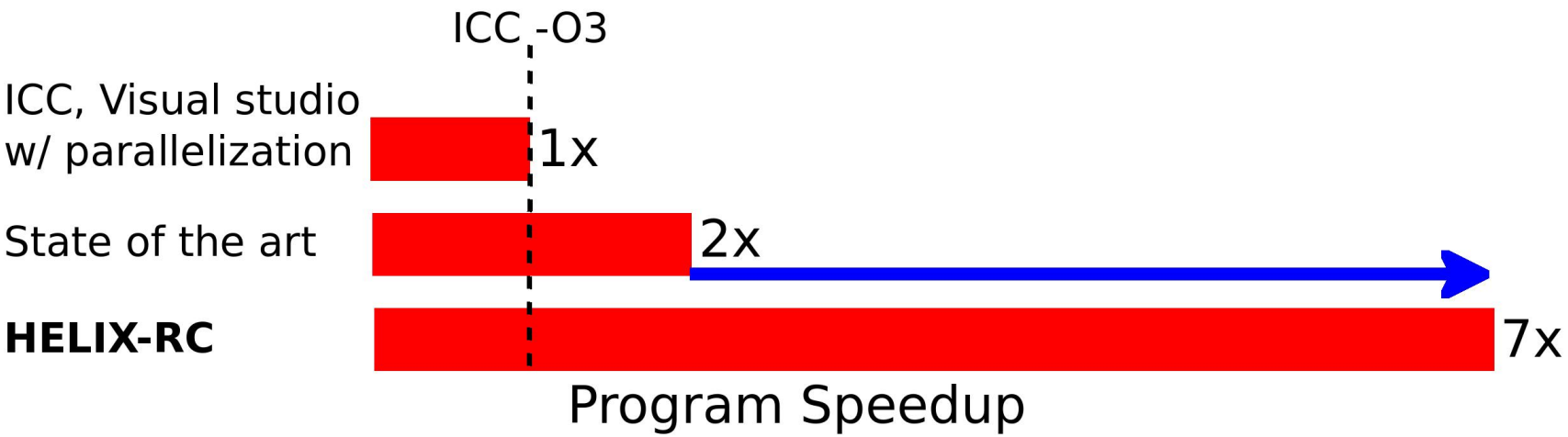




Motivation

Today's commodity platforms include multiple cores





Motivation

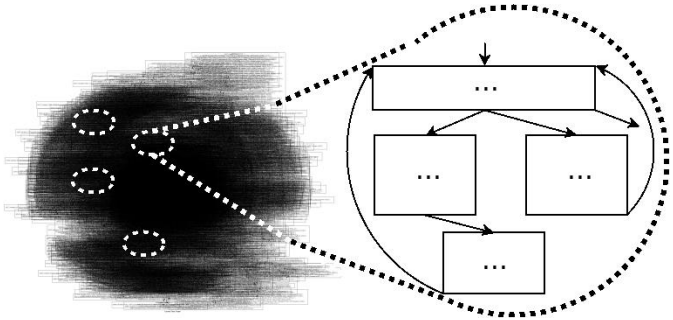
Today's commodity platforms include multiple cores

Ring Cache

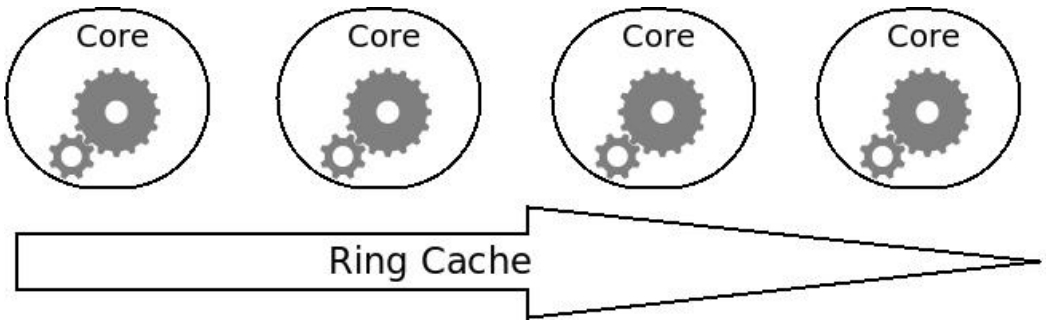
makes
automatic parallelization practical
for conventional multicores

Outline

- **Opportunity of small loops**



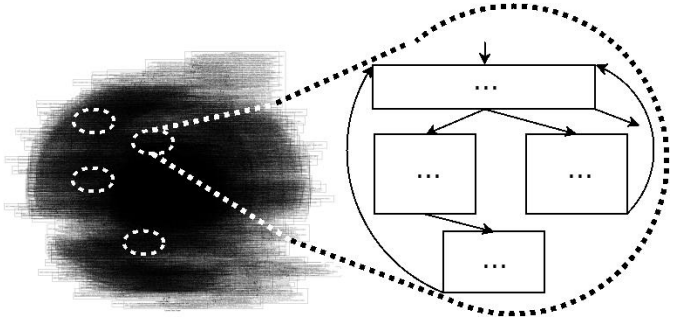
- **The HELIX-RC solution**



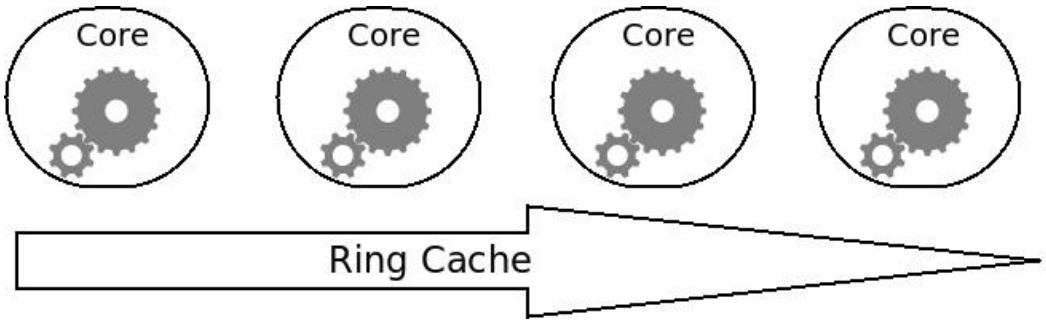
- **Evaluation of HELIX-RC**

Outline

- **Opportunity of small loops**



- **The HELIX-RC solution**



- **Evaluation of HELIX-RC**

Opportunity

Code complexity

- Control flow
- Data flow

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

Opportunity

⇑ Code complexity

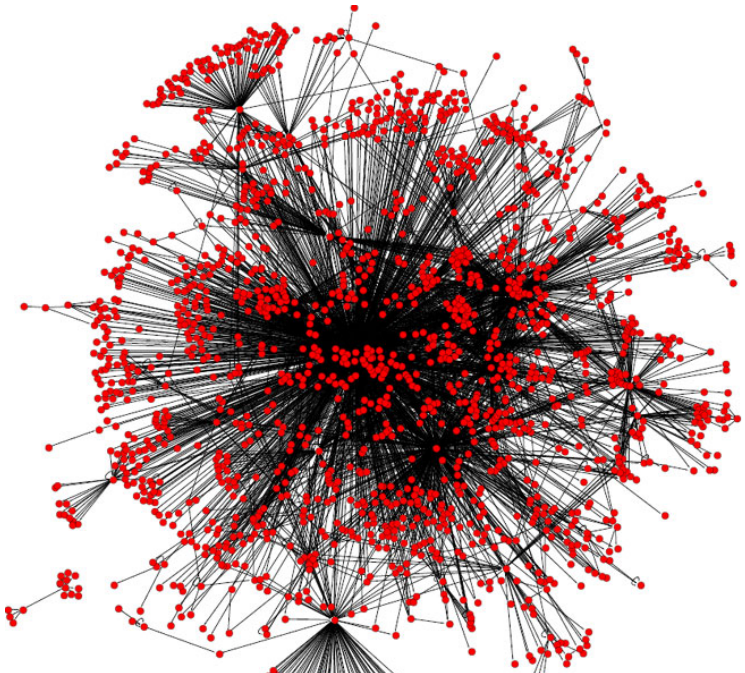
- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)



Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops

Opportunity

⇑ Code complexity

- Control flow
- Data flow

Dependences to satisfy ⇑

- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops

- ⇓ Code complexity

Opportunity

⇑ Code complexity
- Control flow
- Data flow

Dependences to satisfy ⇑
- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops

- ⇓ Code complexity
  - ⇓ *Apparent* (only 1.2× more dependences))

Opportunity

⇑ Code complexity
- Control flow
- Data flow

Dependences to satisfy ⇑
- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops — No TLS

- ⇓ Code complexity
  - ⇓ *Apparent* (only 1.2× more dependences))

Opportunity

⇑ Code complexity
- Control flow
- Data flow

Dependences to satisfy ⇑
- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops — No TLS

- ⇓ Code complexity
  - ⇓ *Apparent* (only 1.2× more dependences))
- Enable code transformations to recompute shared values

Opportunity

⇑ Code complexity
- Control flow
- Data flow

Dependences to satisfy ⇑
- *Actual*
- *Apparent* ⇑

Prior Works

- Thread-Level-Speculation (TLS)
  - ⇓ *Apparent*
- TLS overhead ⇒ big loops (10× more dependences!)

HELIX-RC targets small (hot) loops — No TLS

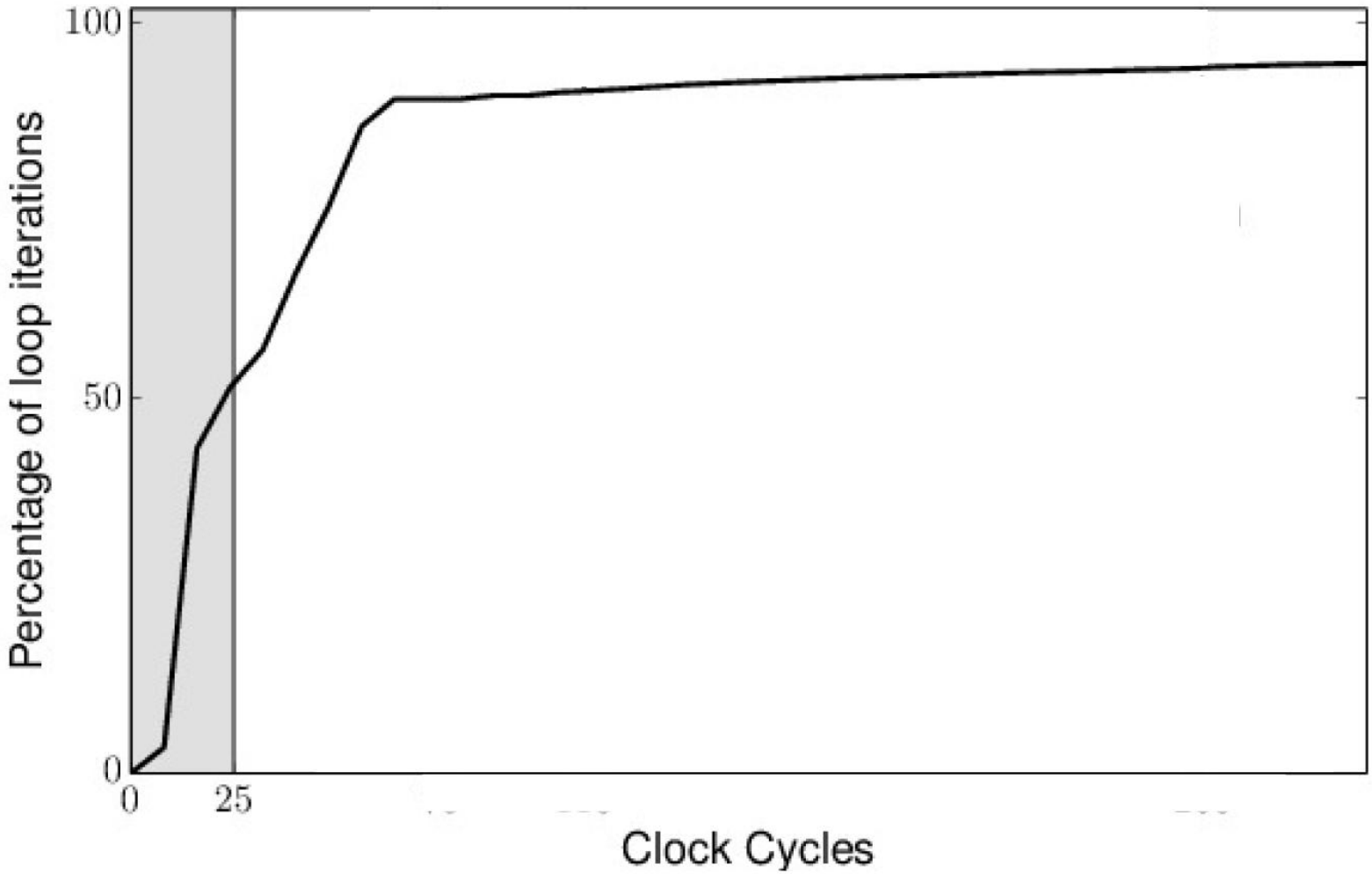
- ⇓ Code complexity
  - ⇓ *Apparent* (only 1.2× more dependences))
- Enable code transformations to recompute shared values
  - ⇓ *Actual*

Main Challenge for Small Loops

Very short loop iterations

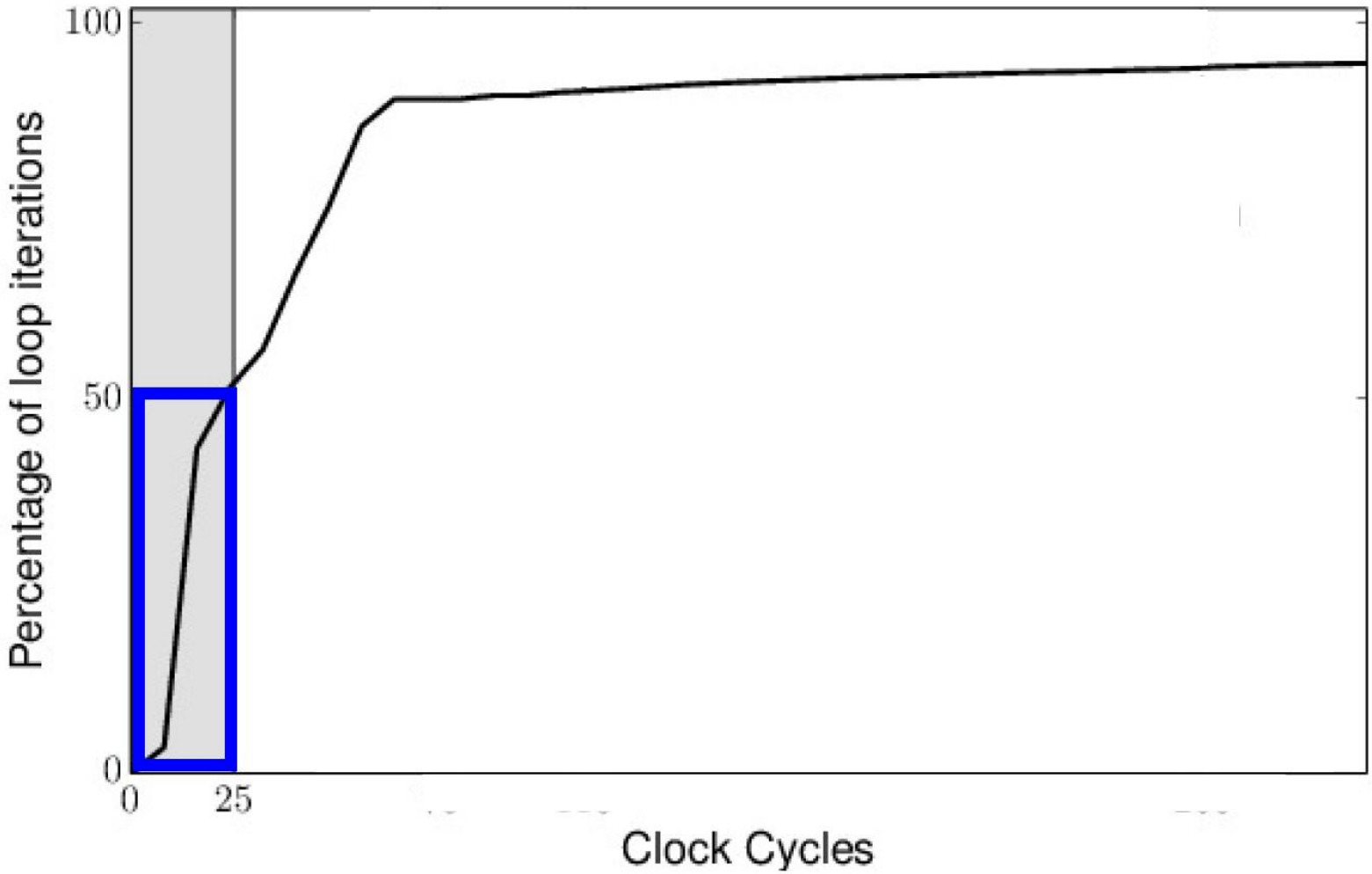
Main Challenge for Small Loops

Very short loop iterations



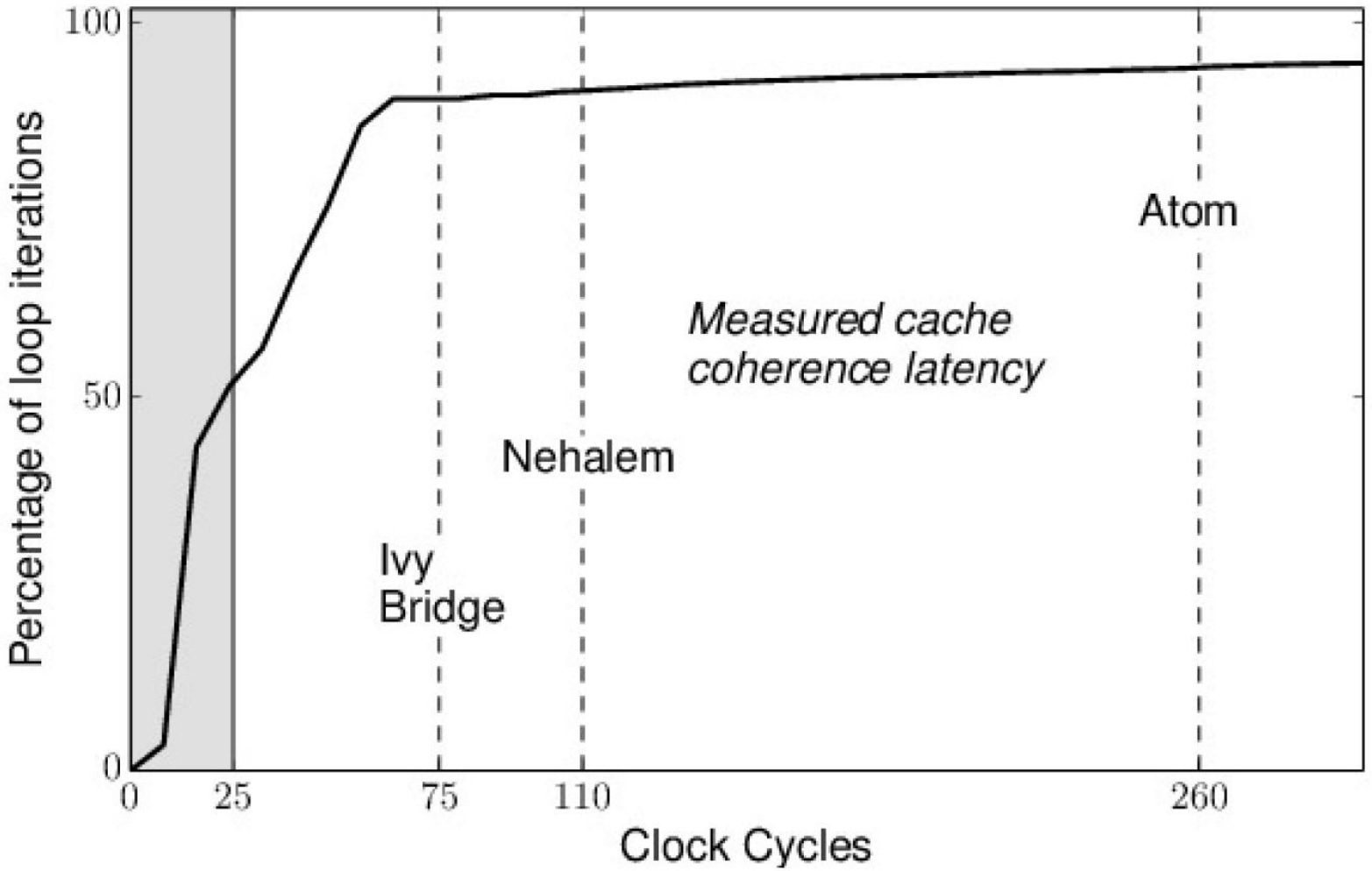
Main Challenge for Small Loops

Very short loop iterations



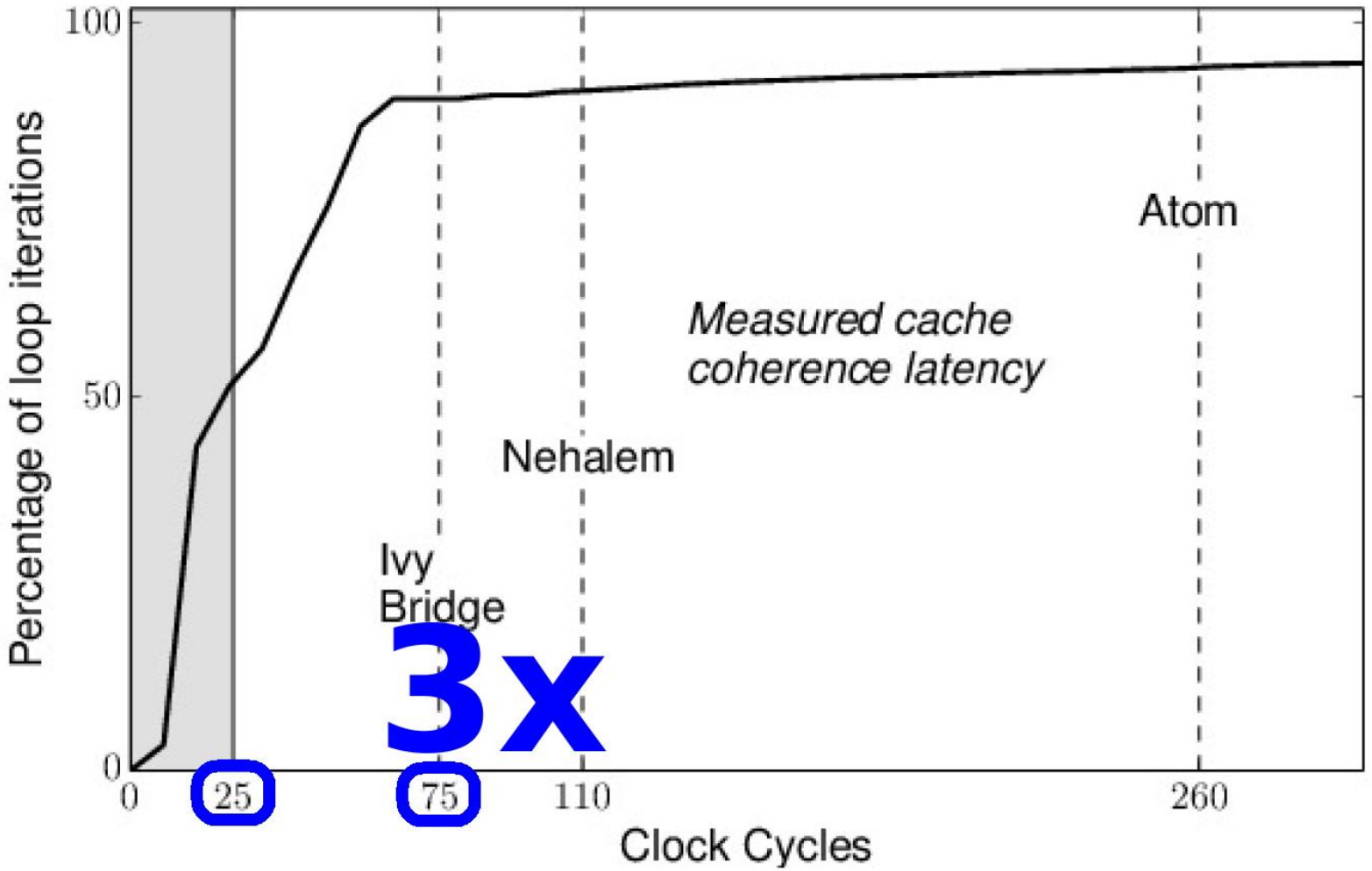
Main Challenge for Small Loops

Very short loop iterations



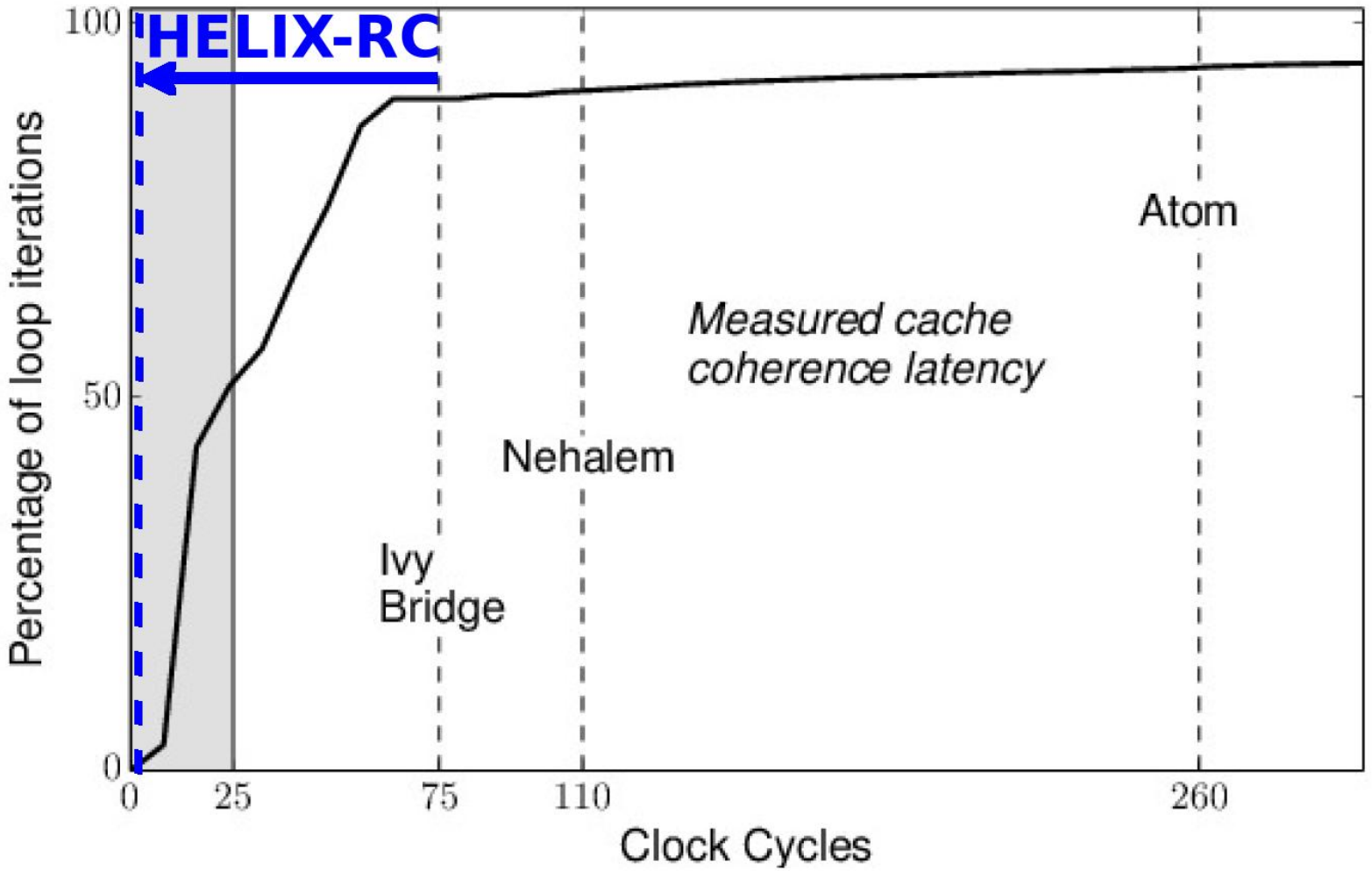
Main Challenge for Small Loops

Very short loop iterations



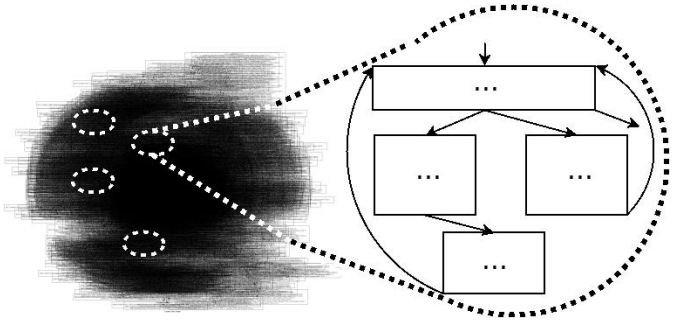
Main Challenge for Small Loops

Very short loop iterations

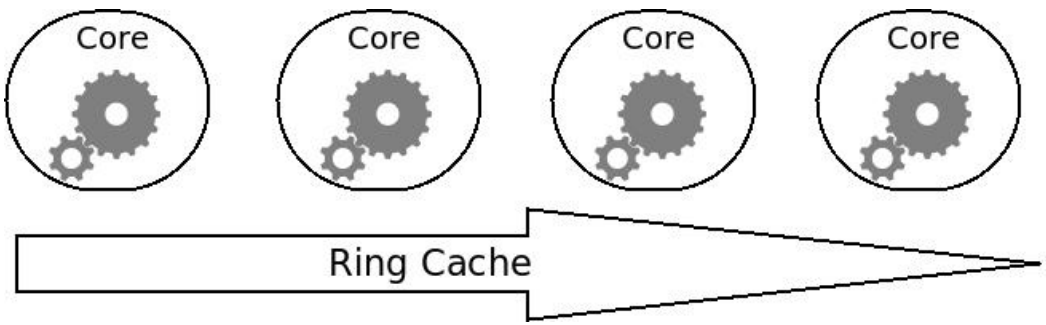


Outline

- **Opportunity of small loops**



- **The HELIX-RC solution**

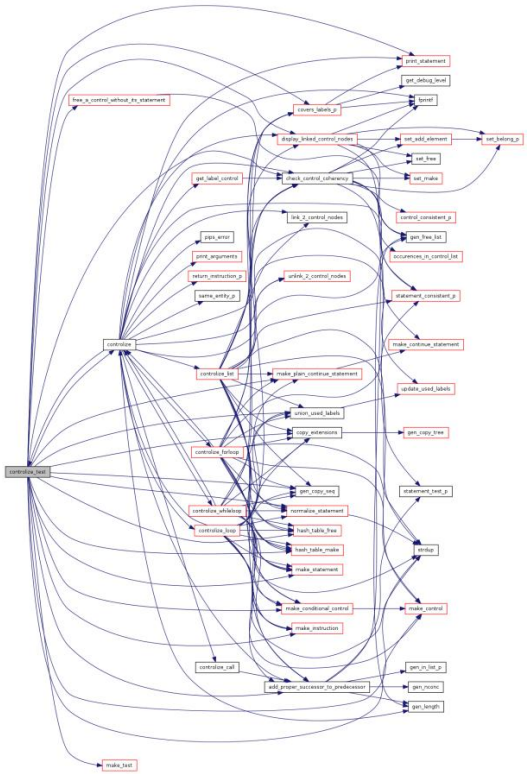


- **Evaluation of HELIX-RC**

Split the Work Among Compiler and Architecture

Compiler: HCCv3

Architecture: Ring Cache

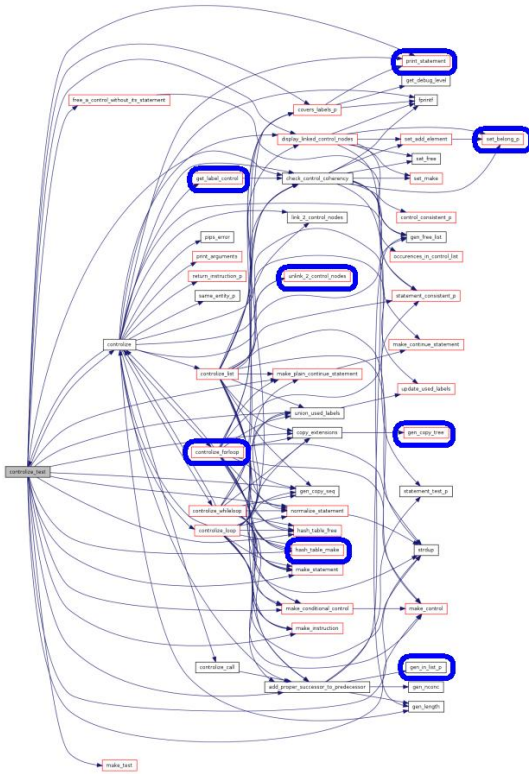


Split the Work Among Compiler and Architecture

Compiler: HCCv3

- Identify code that *may* generate shared data

Architecture: Ring Cache

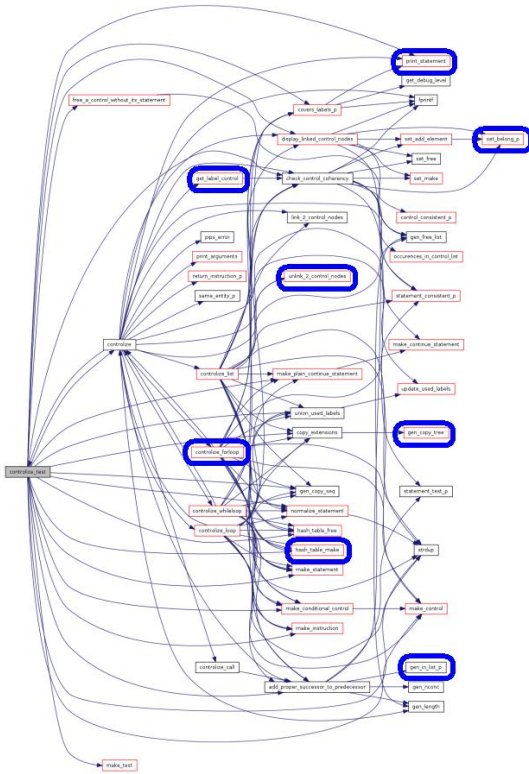


Split the Work Among Compiler and Architecture

Compiler: HCCv3

- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache



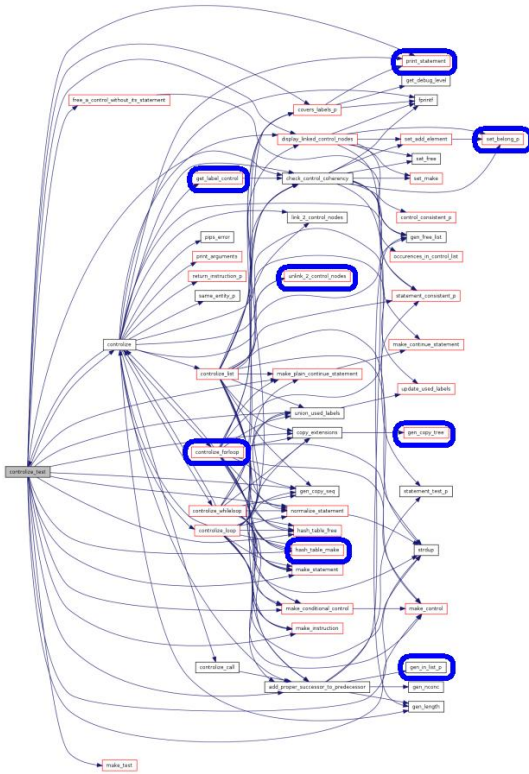
Split the Work Among Compiler and Architecture

Compiler: HCCv3

- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache

Drastically reduce the communication latency



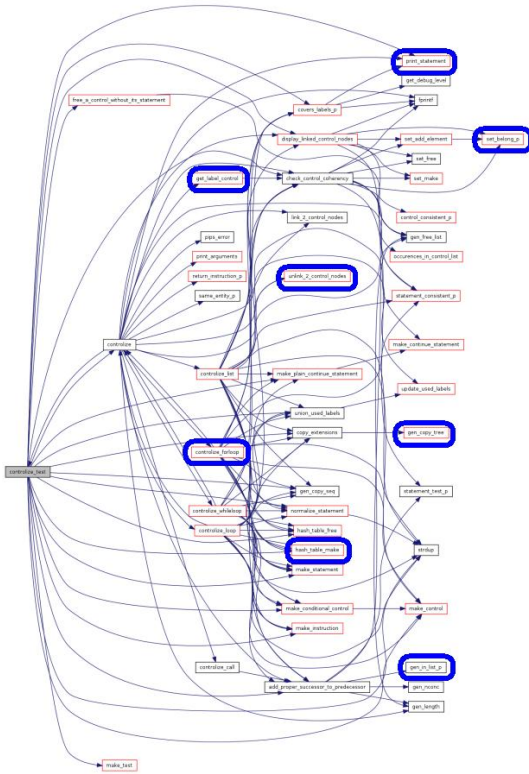
Split the Work Among Compiler and Architecture

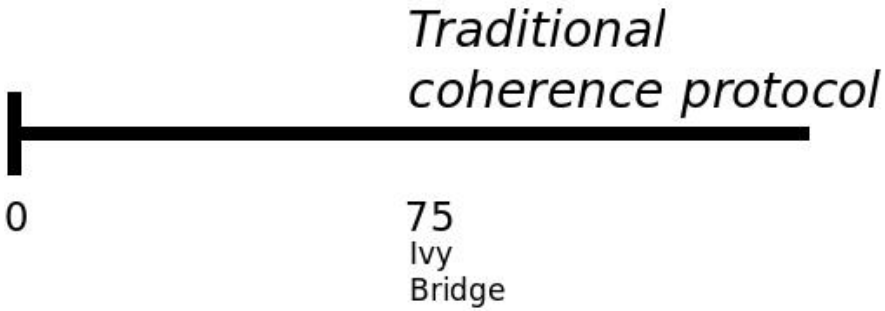
Compiler: HCCv3

- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache

Drastically reduce the communication latency





Split the Work Among Compiler and Architecture

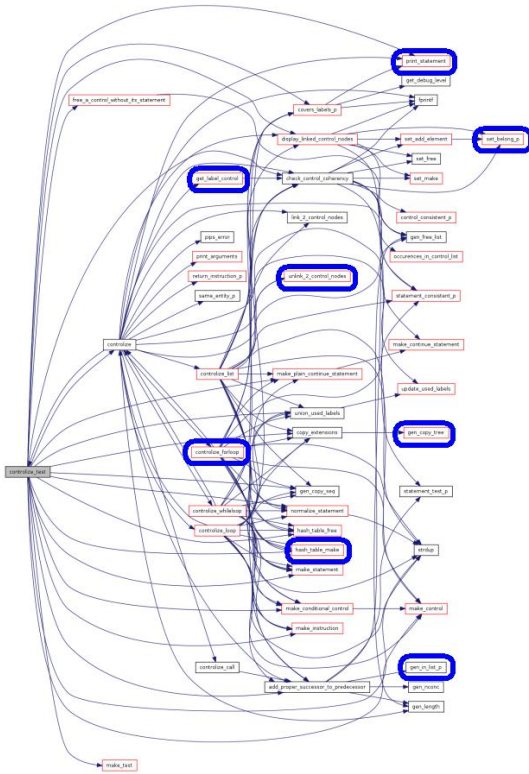
Compiler: HCCv3

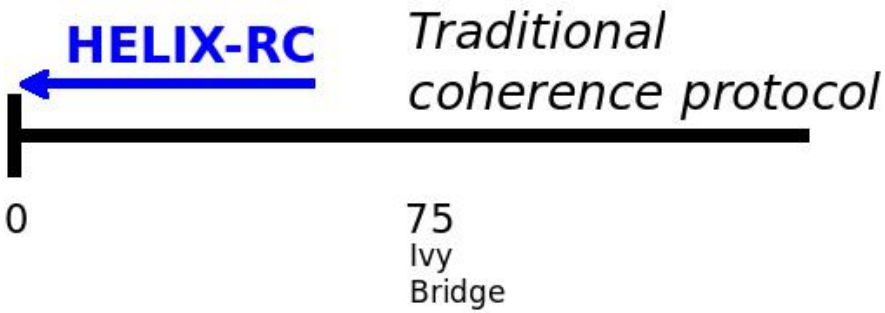
- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache

Drastically reduce the communication latency

- Proactive data distribution





Split the Work Among Compiler and Architecture

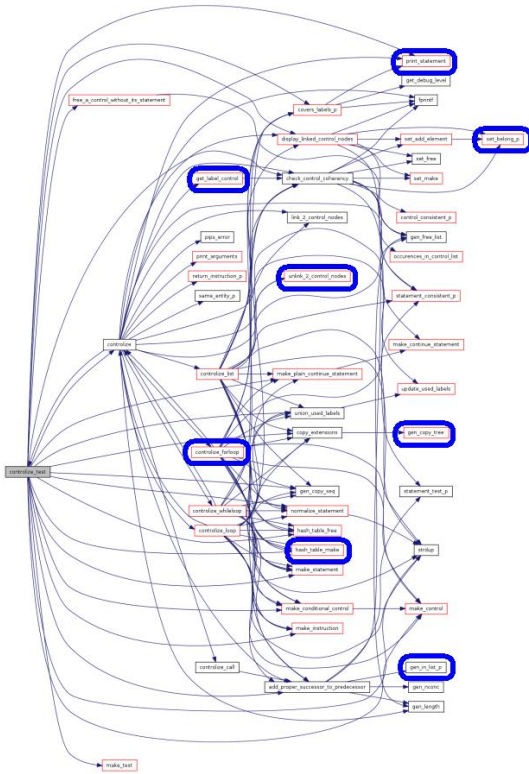
Compiler: HCCv3

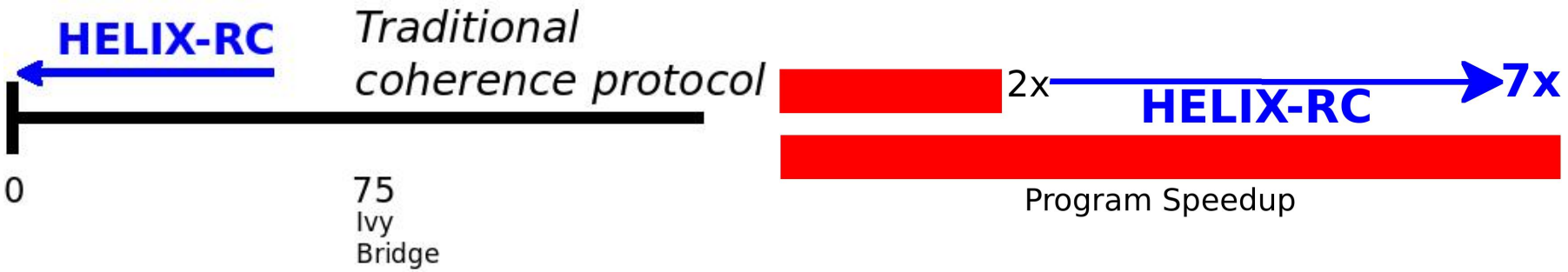
- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache

Drastically reduce the communication latency

- Proactive data distribution





Split the Work Among Compiler and Architecture

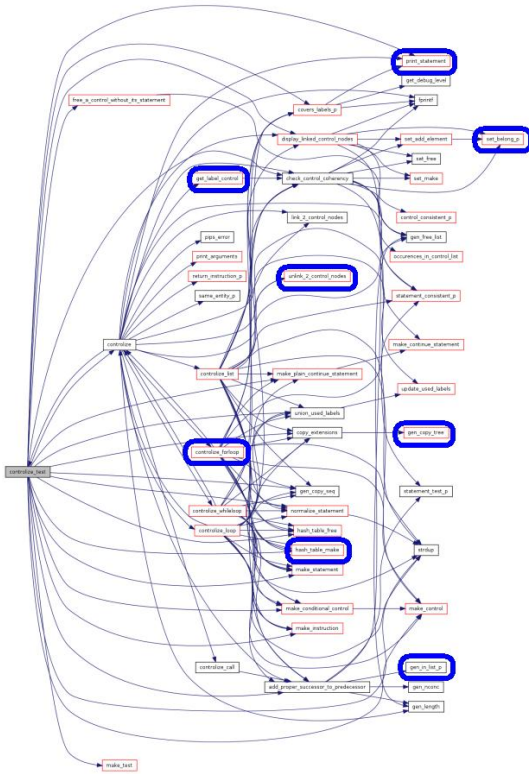
Compiler: HCCv3

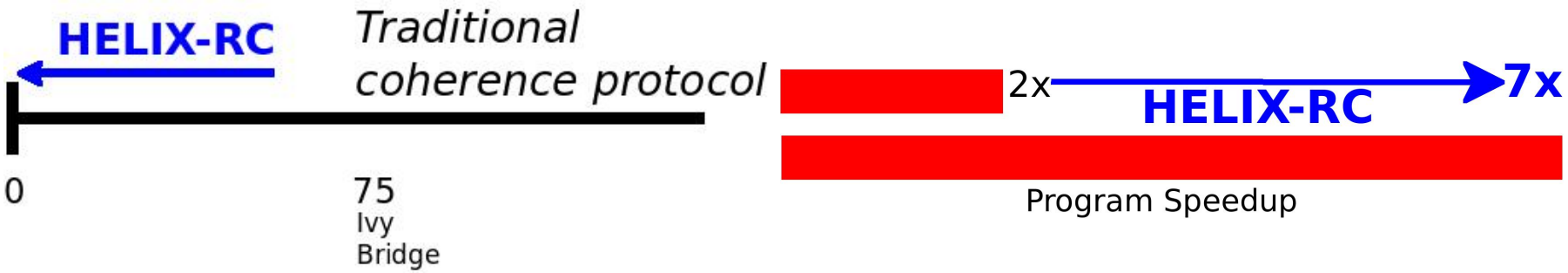
- Identify code that *may* generate shared data
- Expose information to architecture

Architecture: Ring Cache

Drastically reduce the communication latency

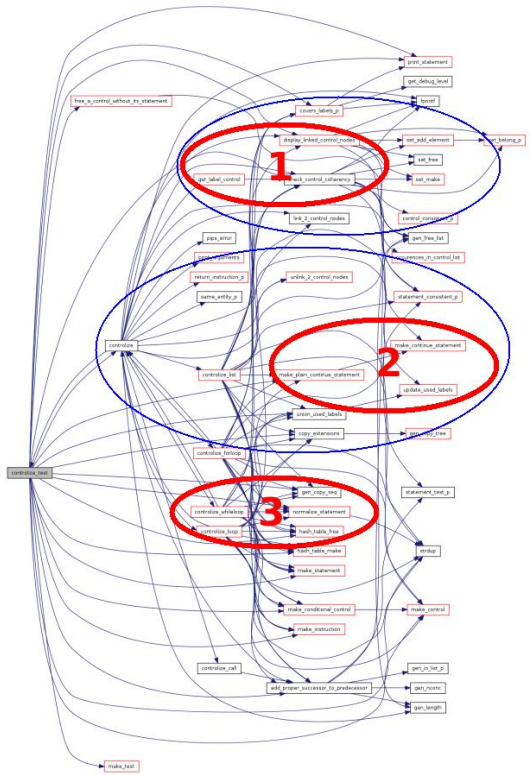
- Proactive data distribution





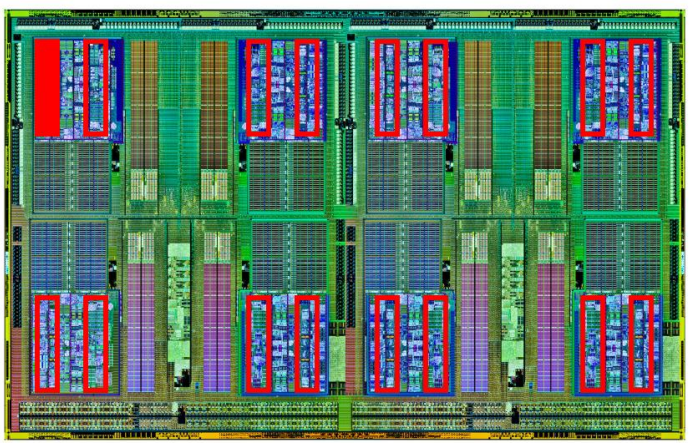
Compiler HCCv3

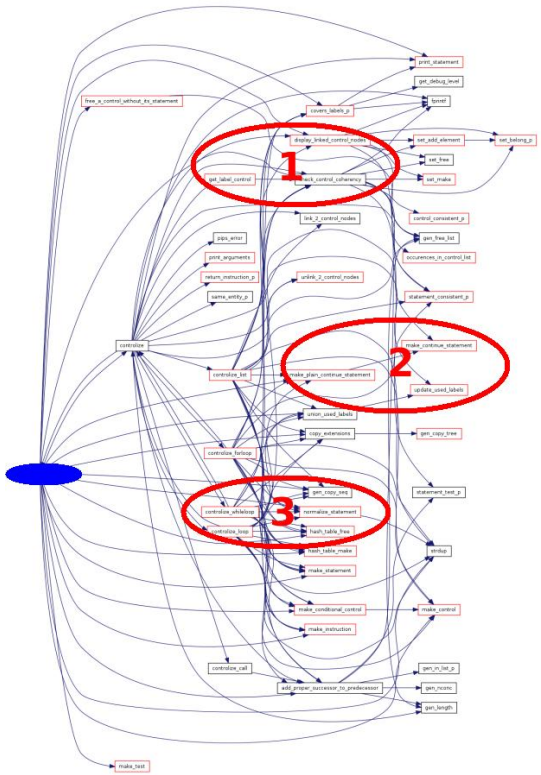
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory



Compiler HCCv3

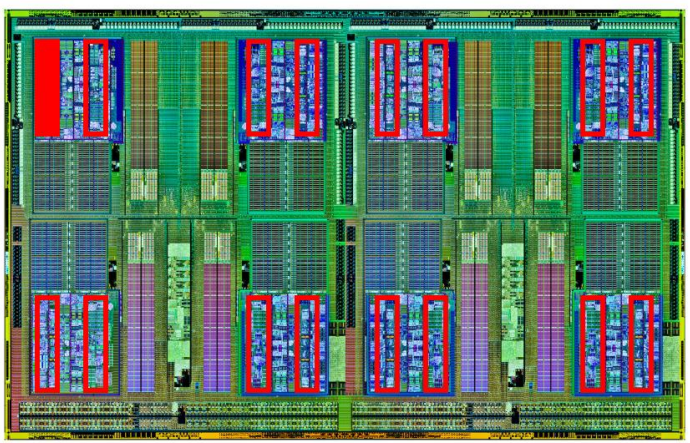
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory

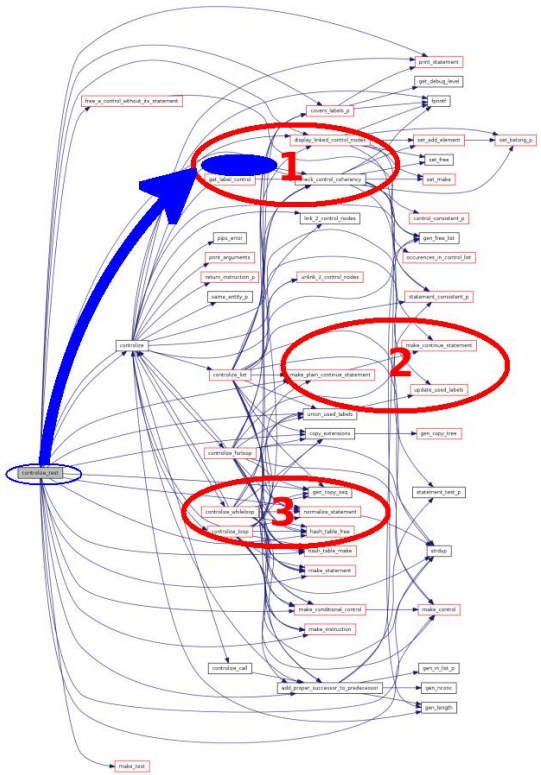




Compiler HCCv3

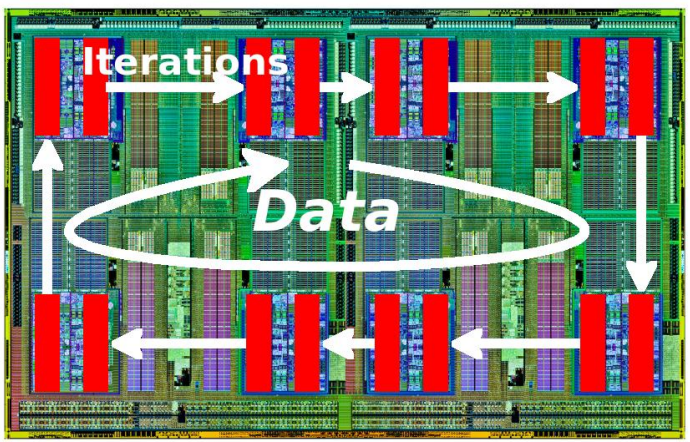
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory

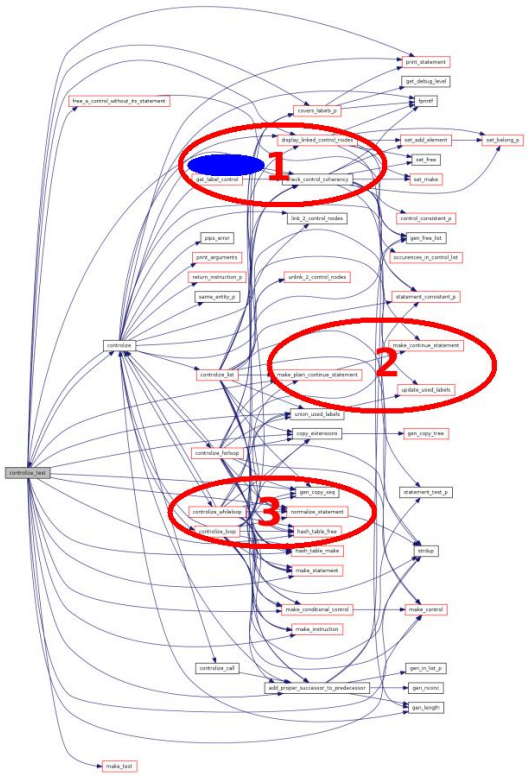




Compiler HCCv3

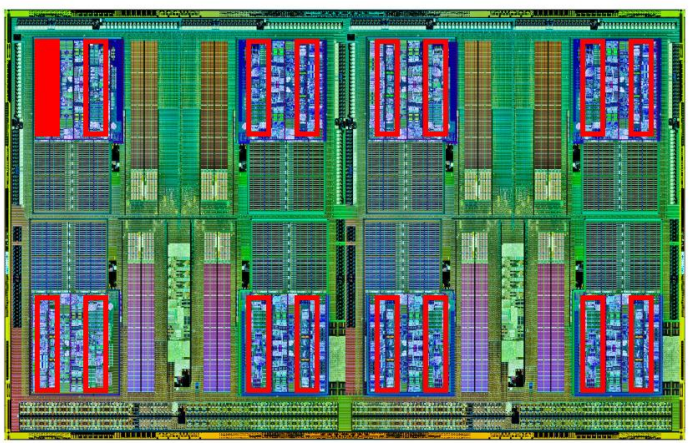
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory

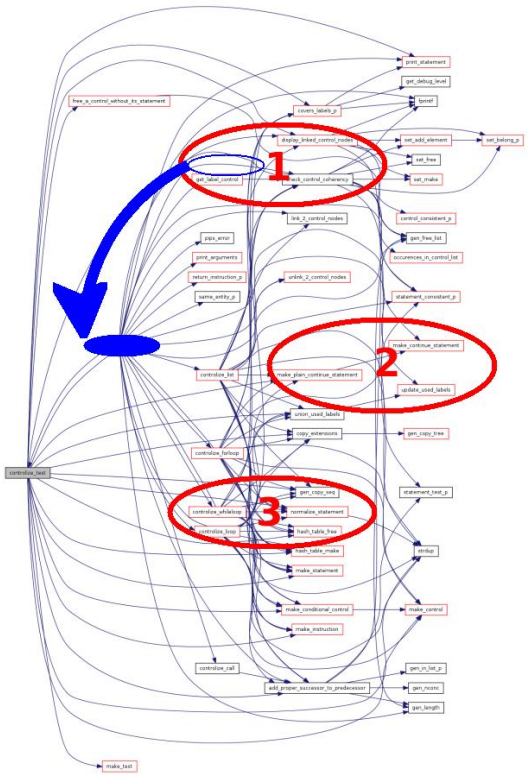




Compiler HCCv3

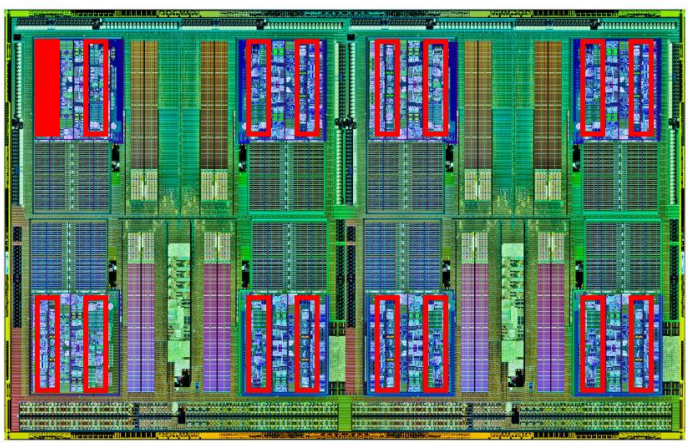
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory

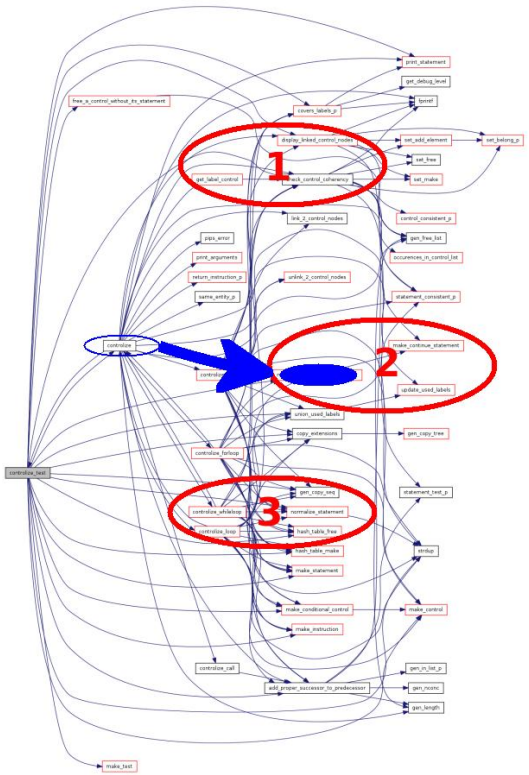




Compiler HCCv3

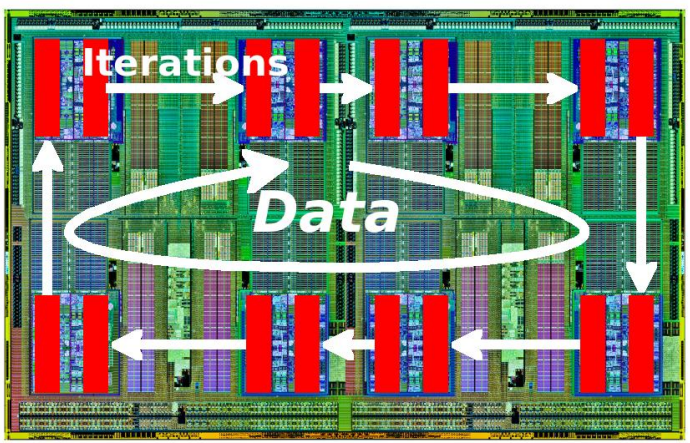
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory

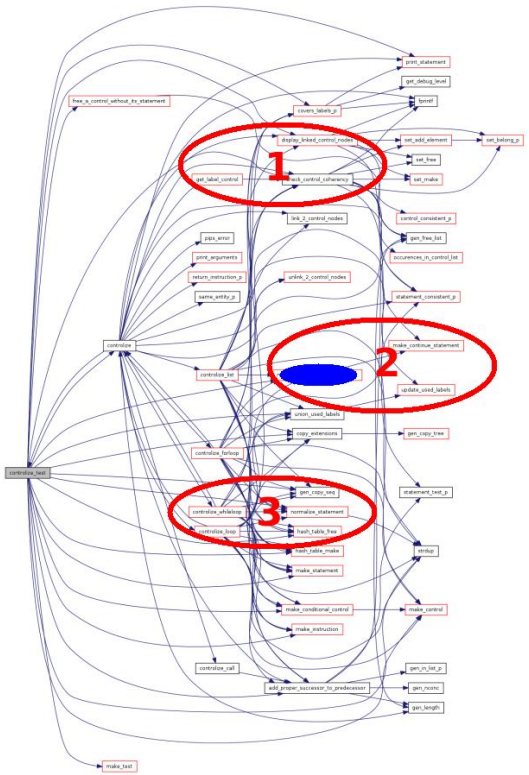




Compiler HCCv3

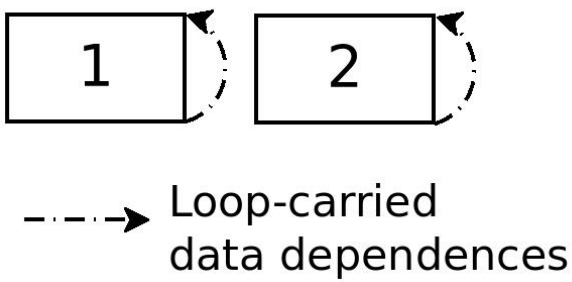
1. Parallelize most promising loops
   - Identify code that *may* generate shared loop iteration data
   - Keep shared data in memory





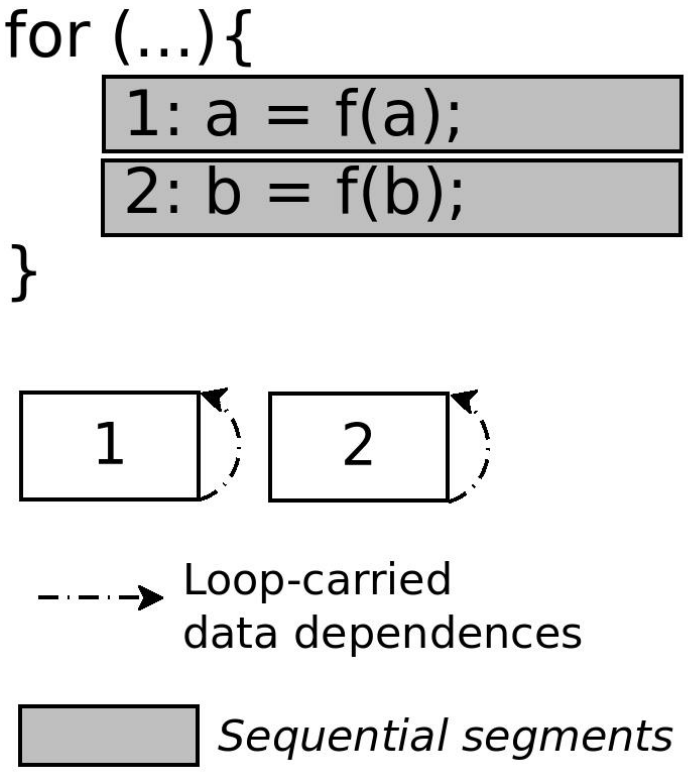
Parallelism Among Sequential Segments

```
for (...){
    1: a = f(a);
    2: b = f(b);
}
```



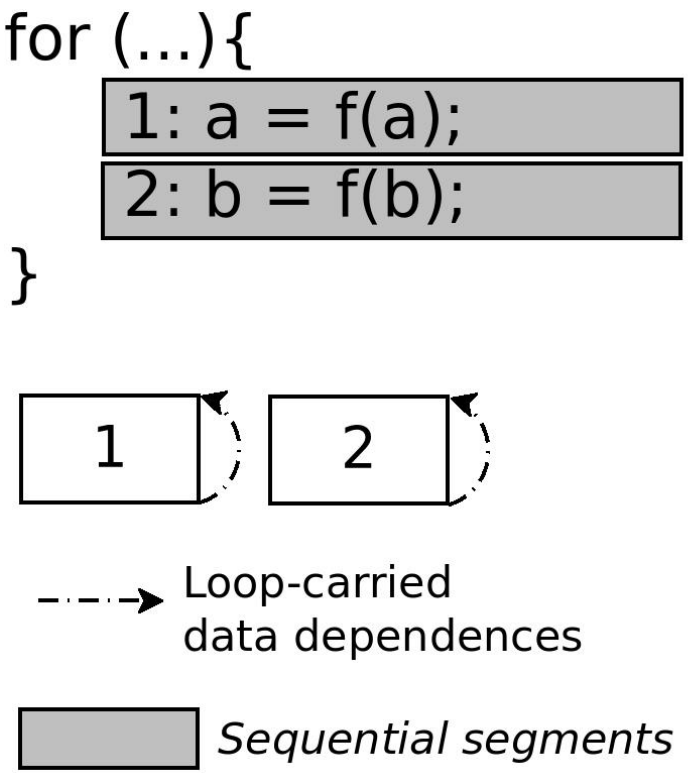
- A small loop

Parallelism Among Sequential Segments



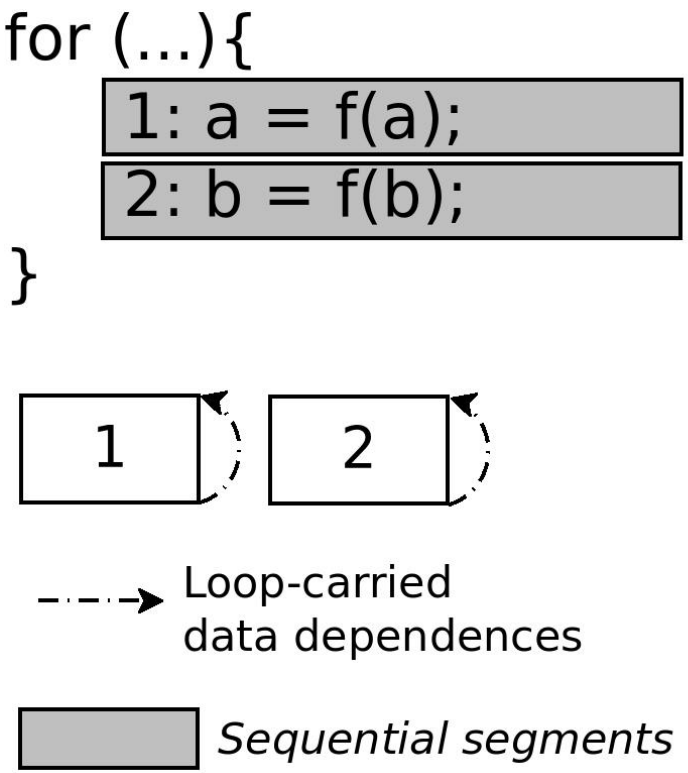
- Sequential segments 

Parallelism Among Sequential Segments



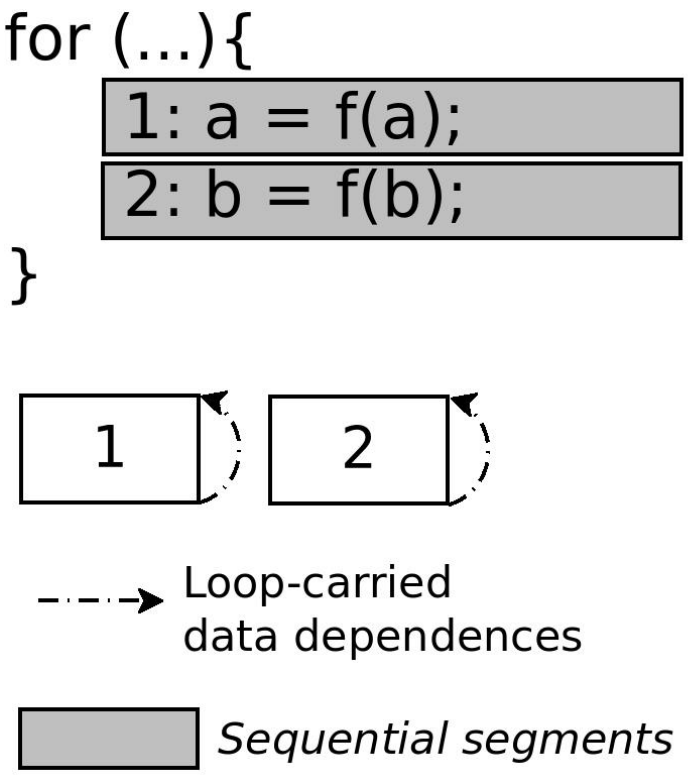
- Sequential segments *may* generate shared data

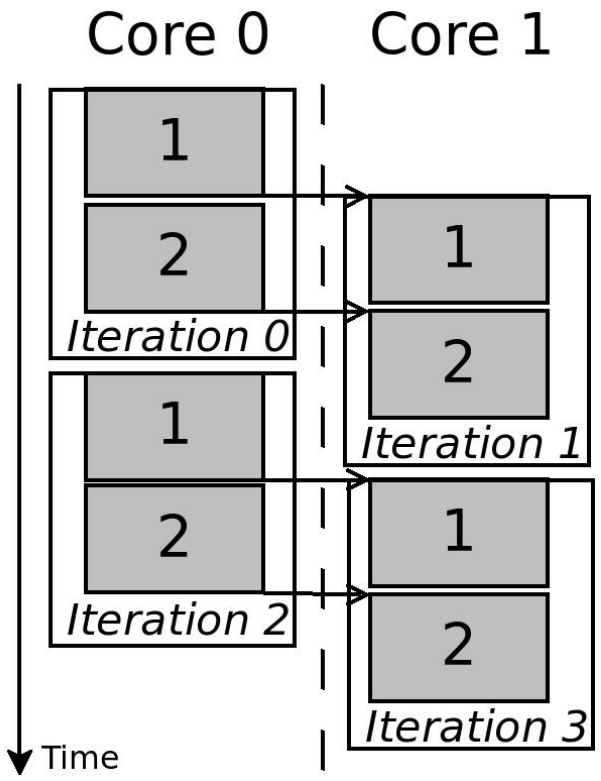
Parallelism Among Sequential Segments



- Sequential segments *may* generate shared data
  - Each sequential segment executes in loop-iteration order

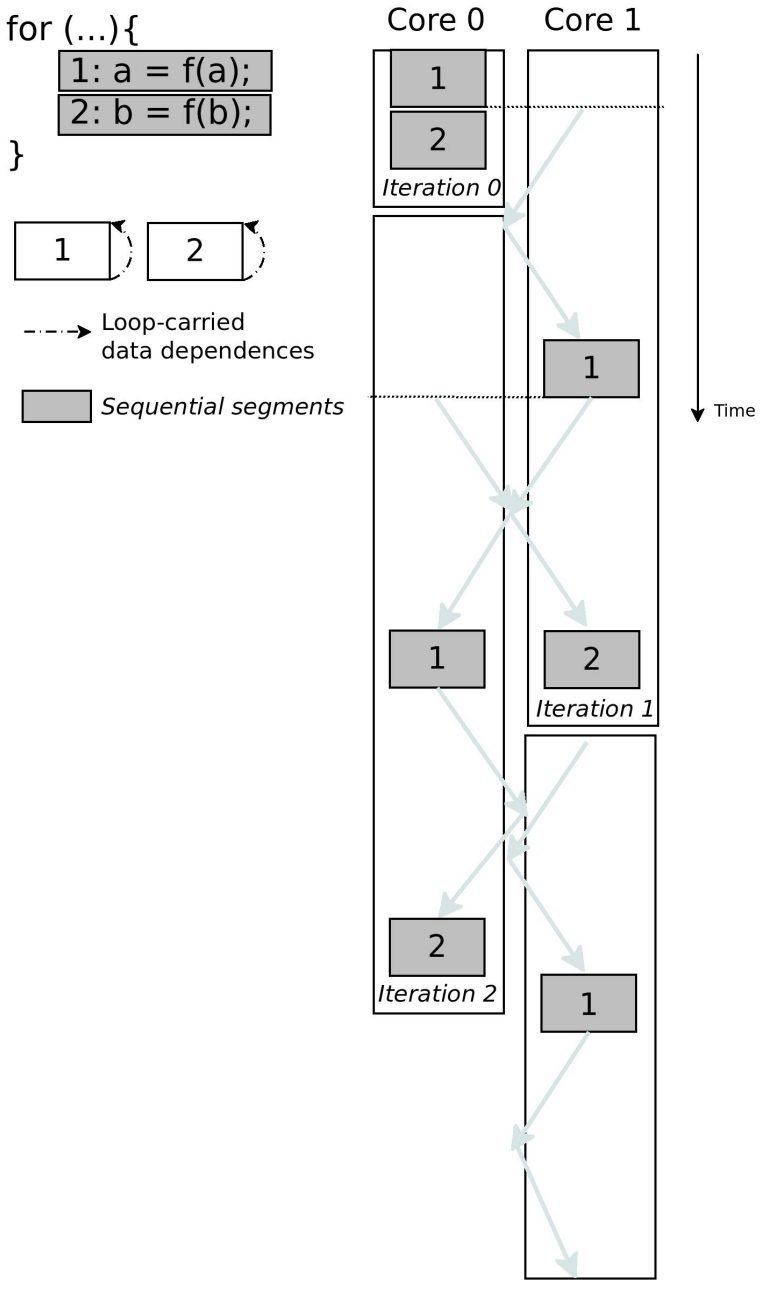
Parallelism Among Sequential Segments





- Sequential segments *may* generate shared data
  - Each sequential segment executes in loop-iteration order
- Parallelism among sequential segments

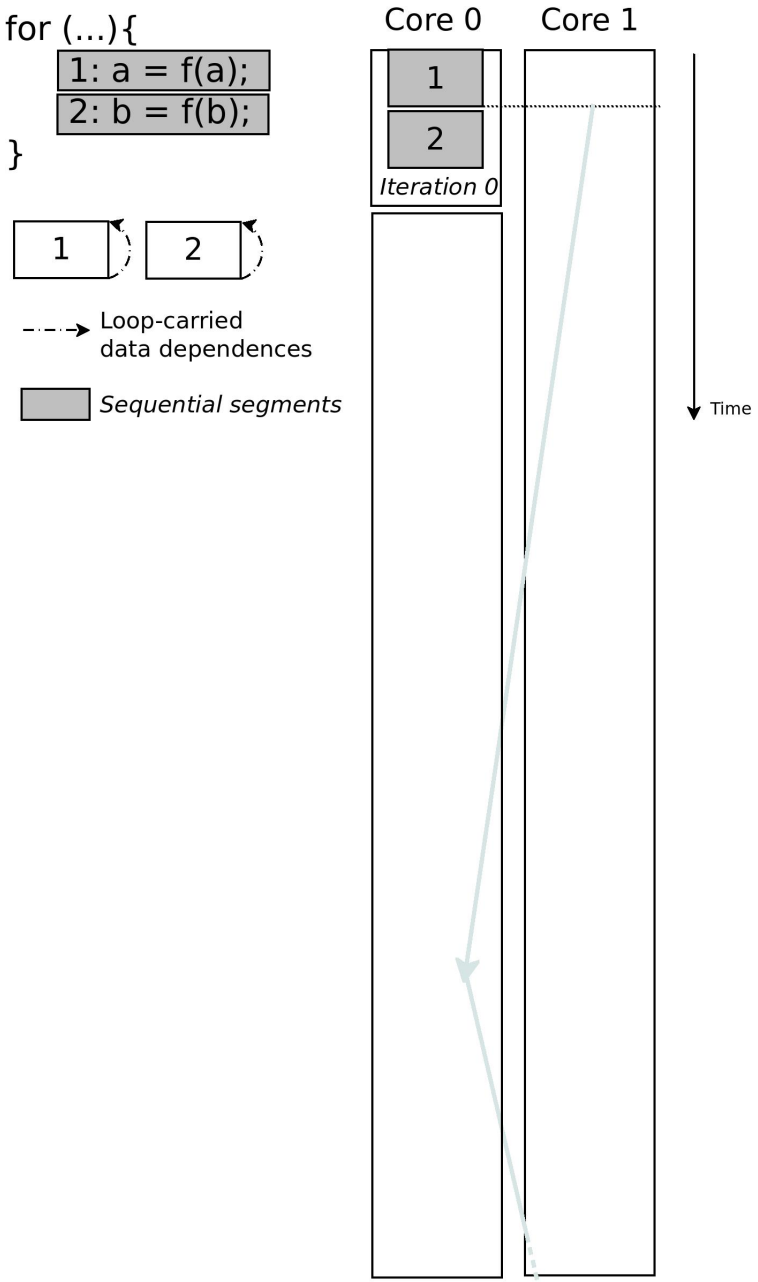
Small Loops Do Not Work On Commodity Multicore



Bottleneck

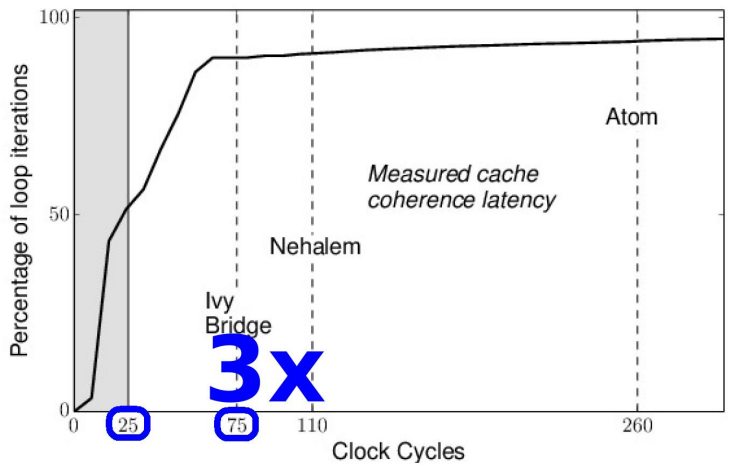
- Data movement

Small Loops Do Not Work On Commodity Multicore

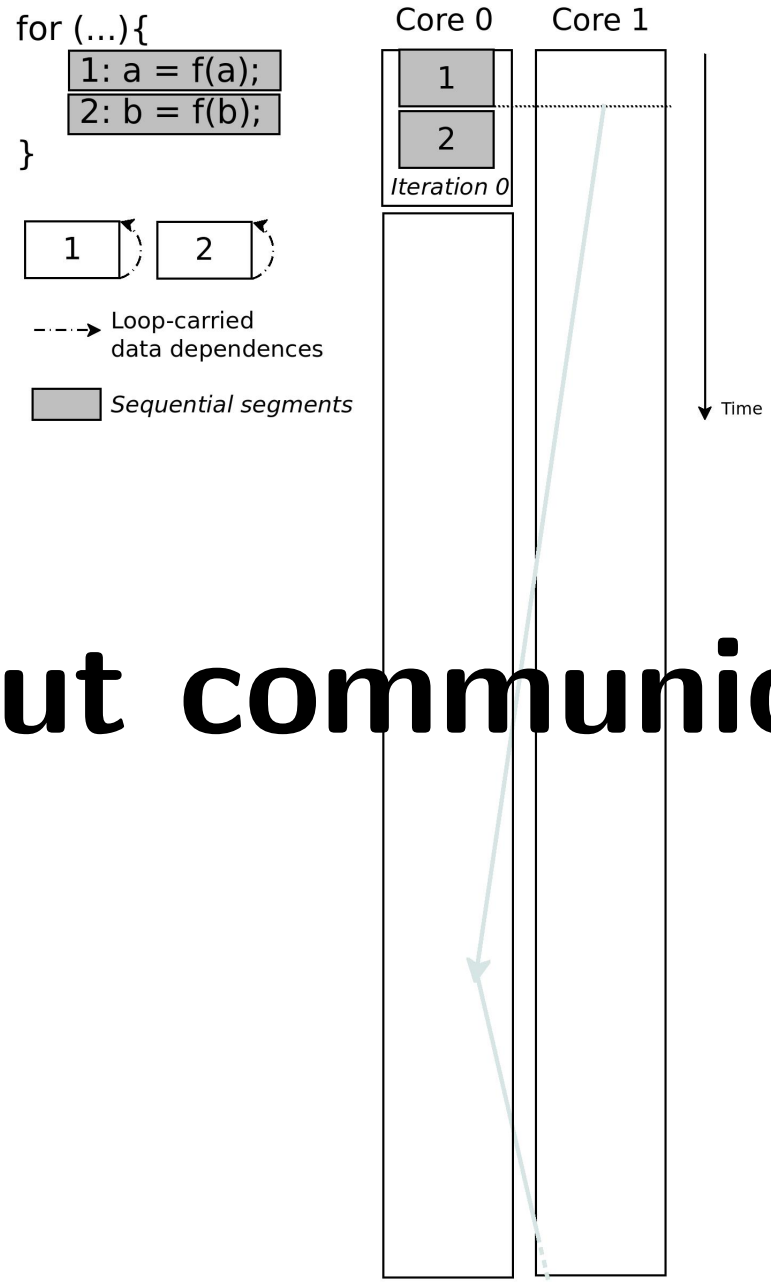


Bottleneck

- Data movement

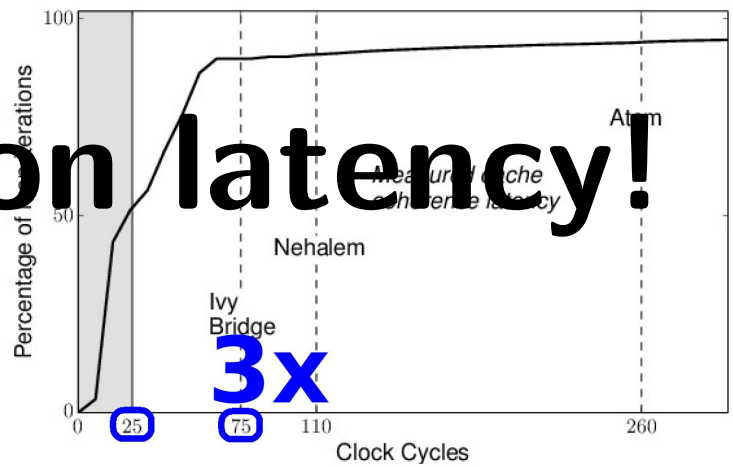


Small Loops Do Not Work On Commodity Multicore



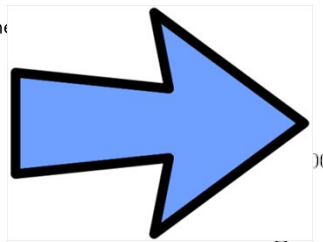
Bottleneck

- Data movement

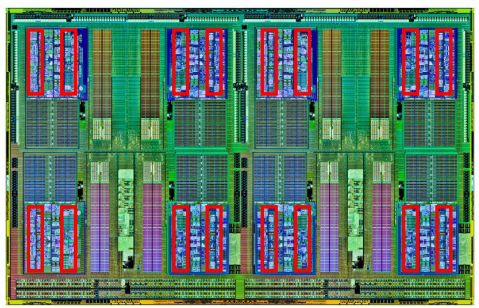


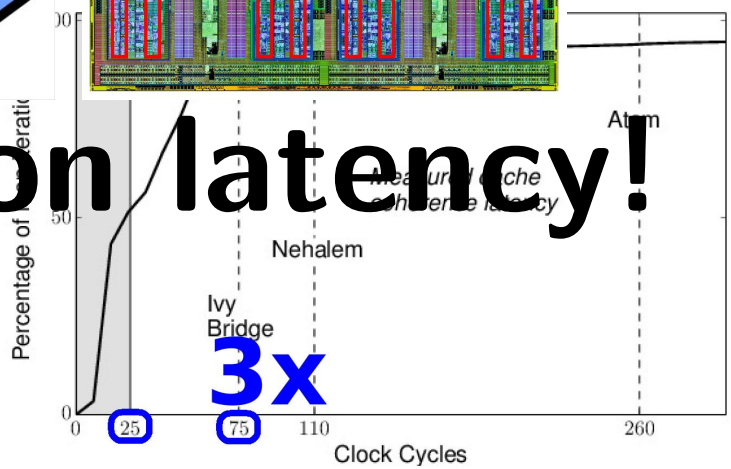
Cut communication latency!

Small Loops Do Not Work On Commodity Multicore



Bottleneck

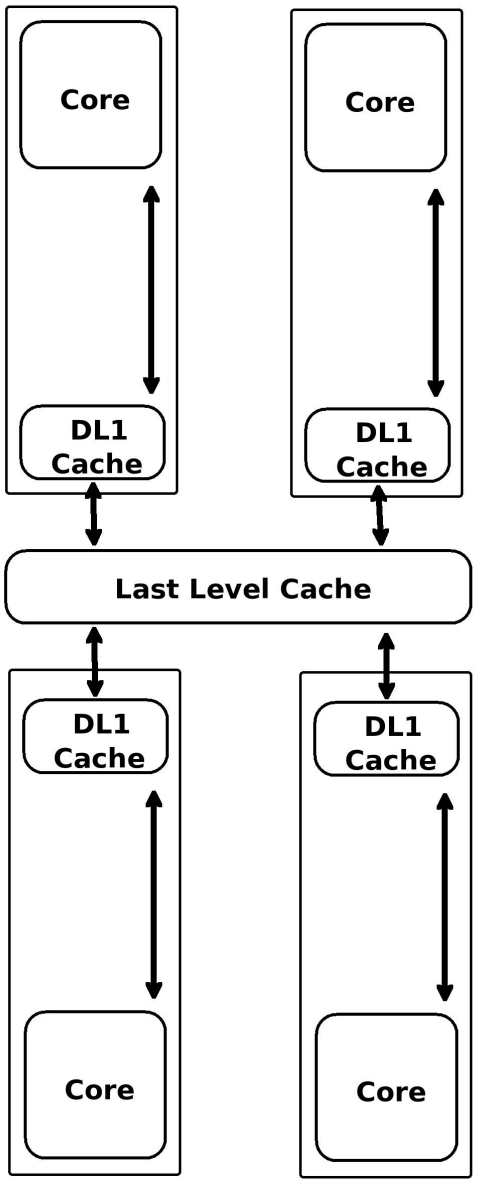




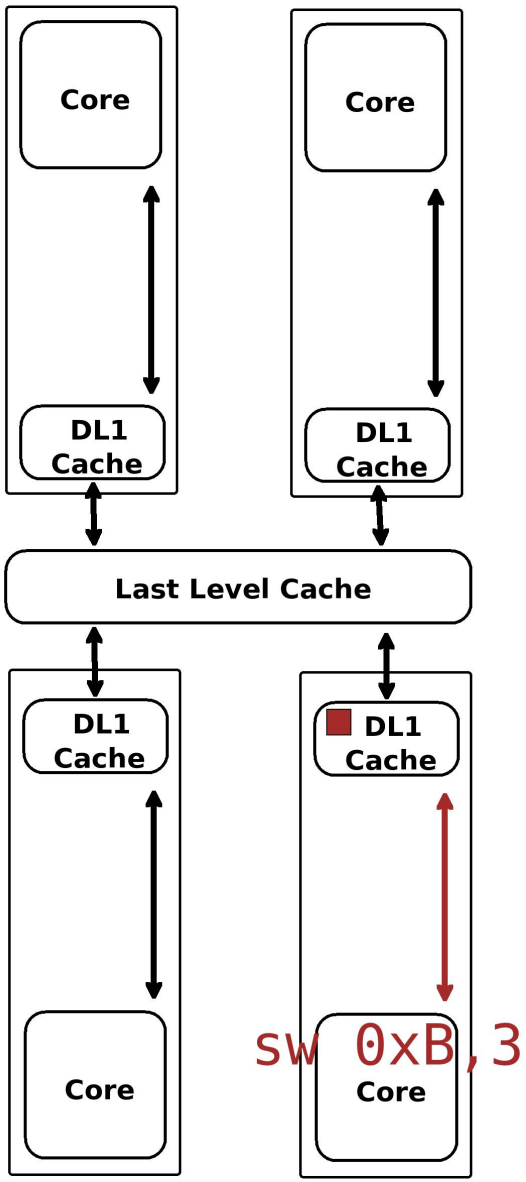
Cut communication latency!

Light Enhancement in Conventional Multicore Architecture

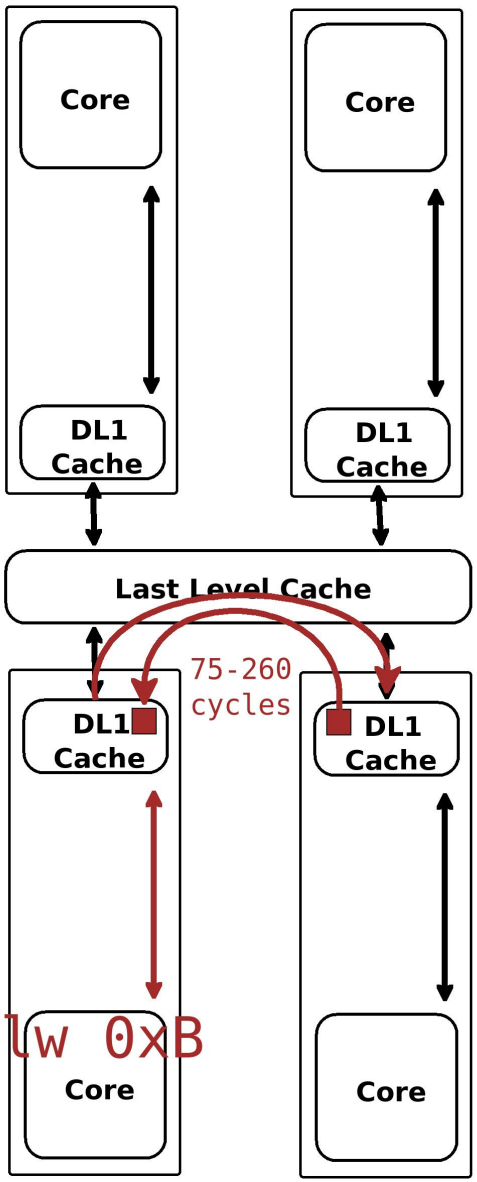
Light Enhancement in Conventional Multicore Architecture



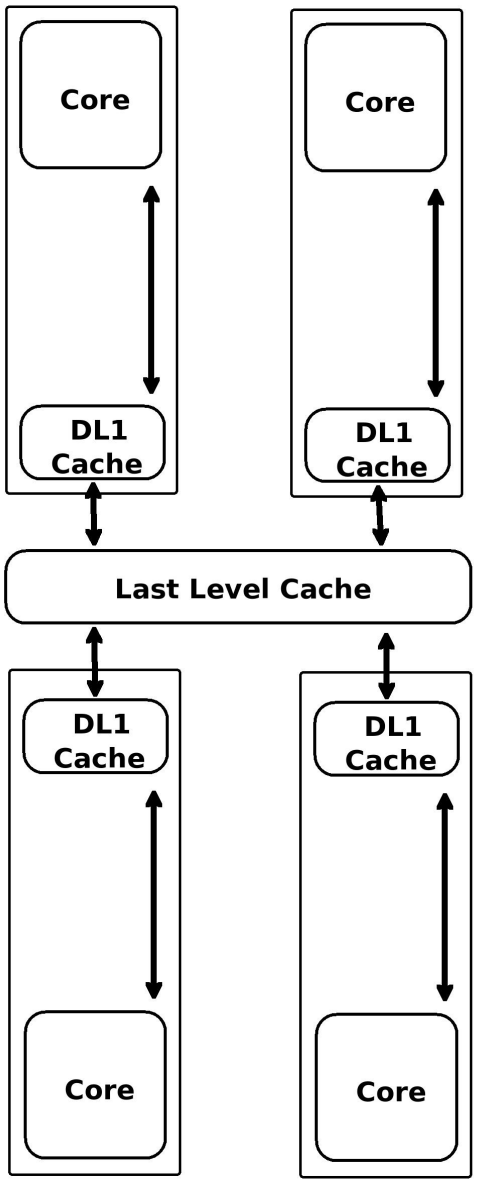
Light Enhancement in Conventional Multicore Architecture



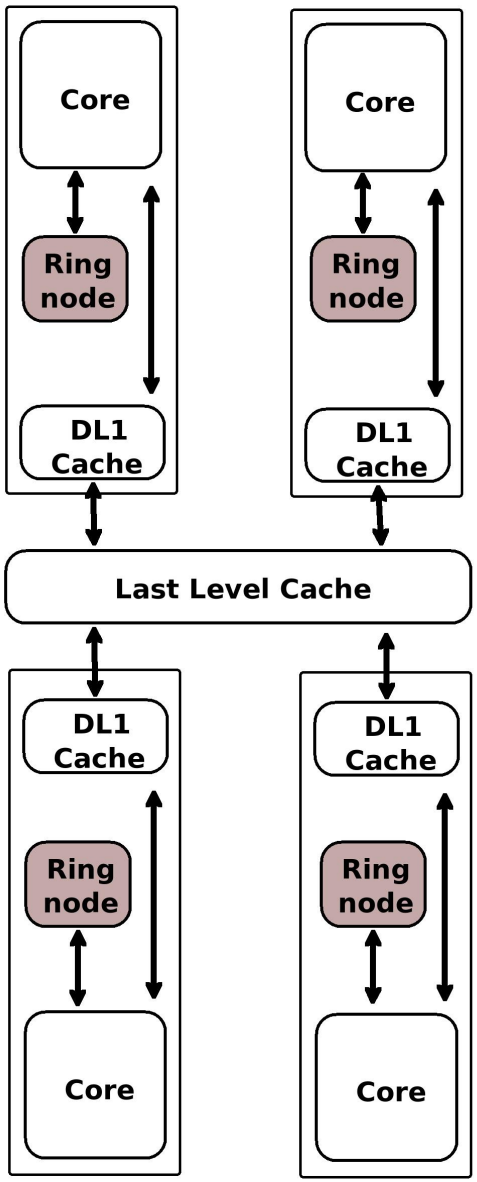
Light Enhancement in Conventional Multicore Architecture



Light Enhancement in Conventional Multicore Architecture

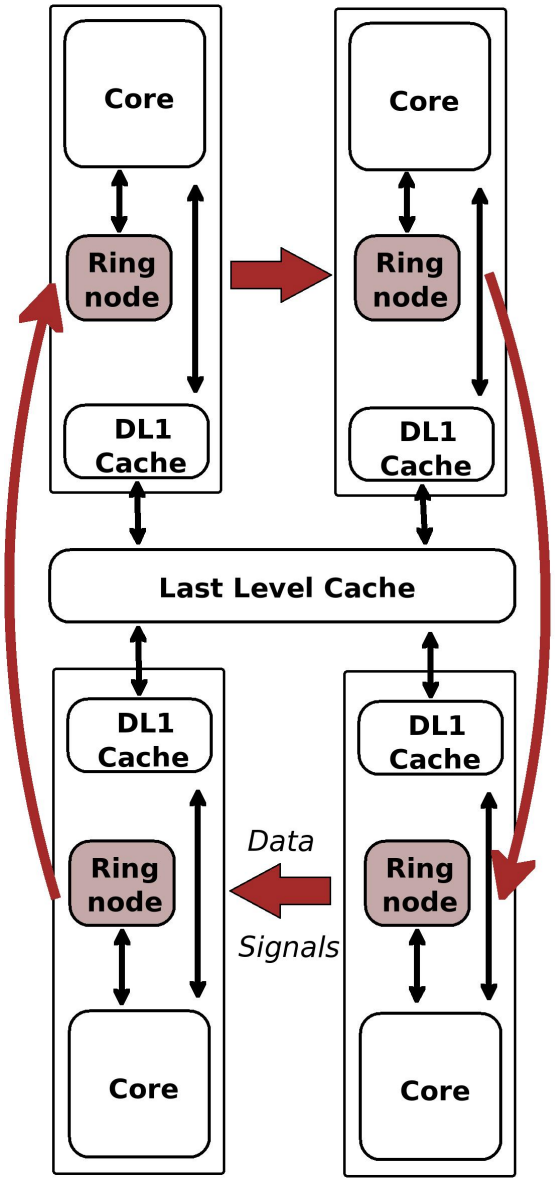


Light Enhancement in Conventional Multicore Architecture



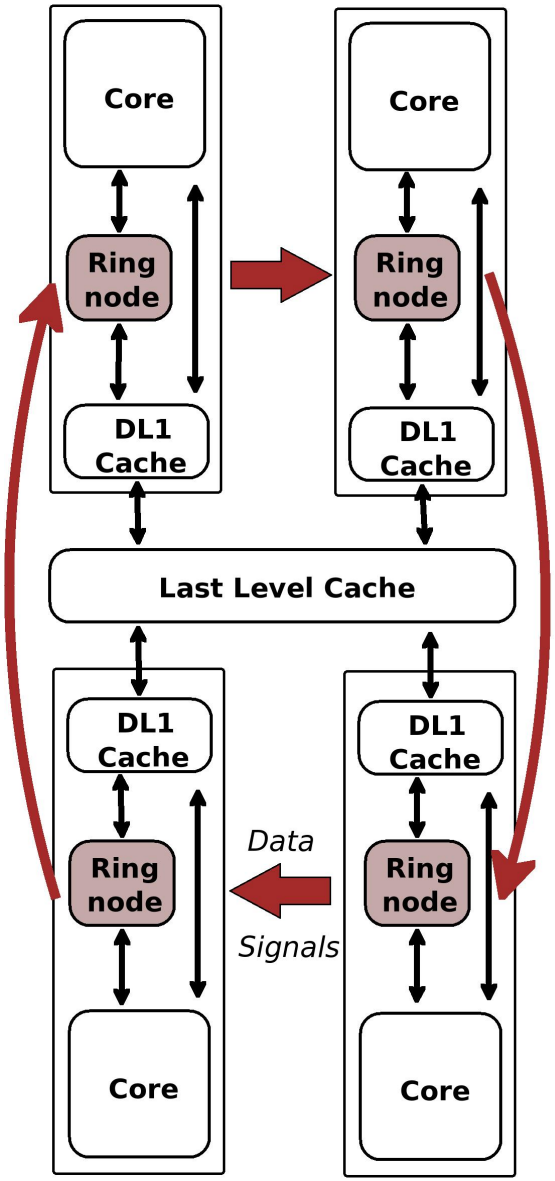
Ring nodes
cache shared data

Light Enhancement in Conventional Multicore Architecture

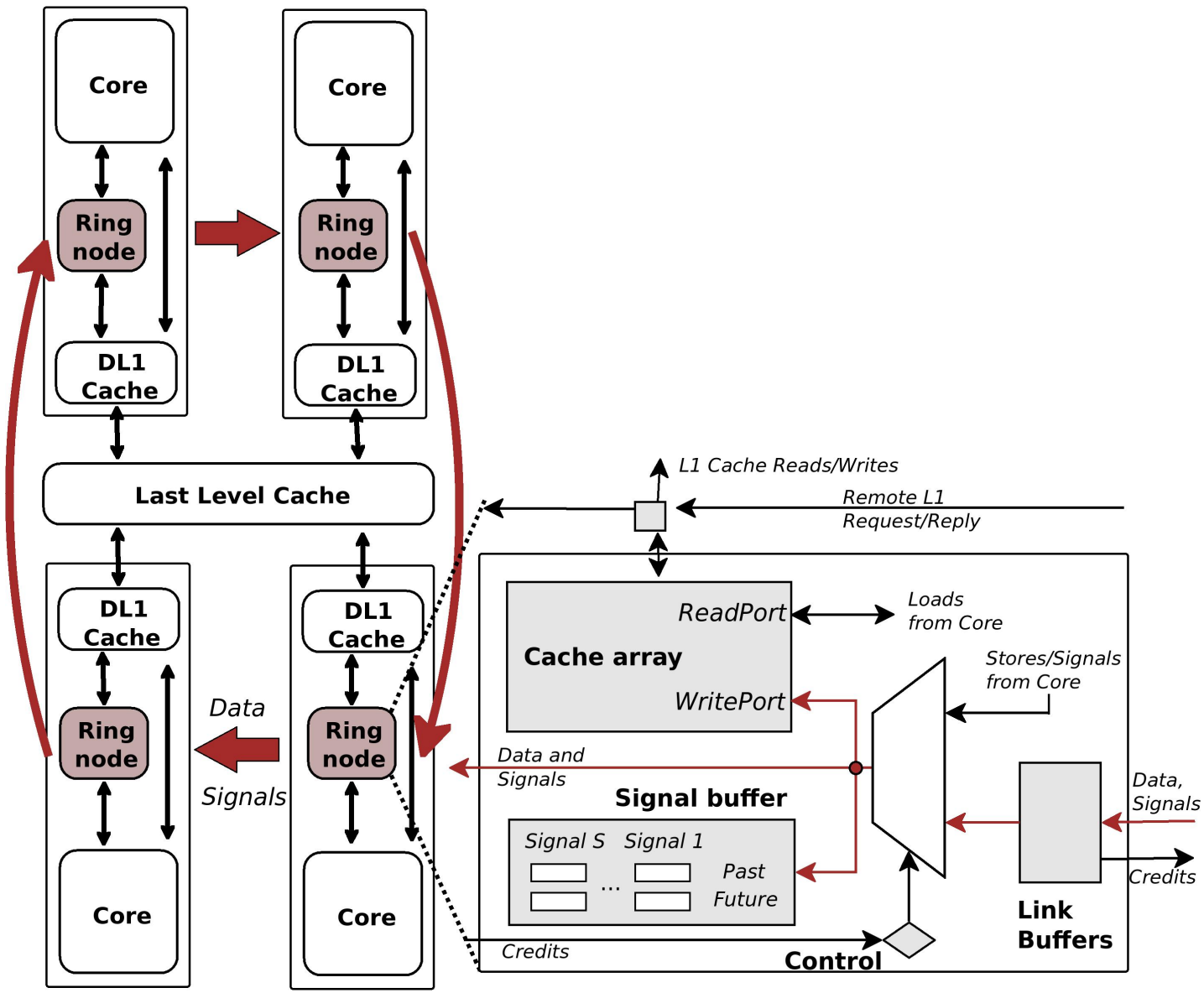


Accelerate communication shaped by the compiler

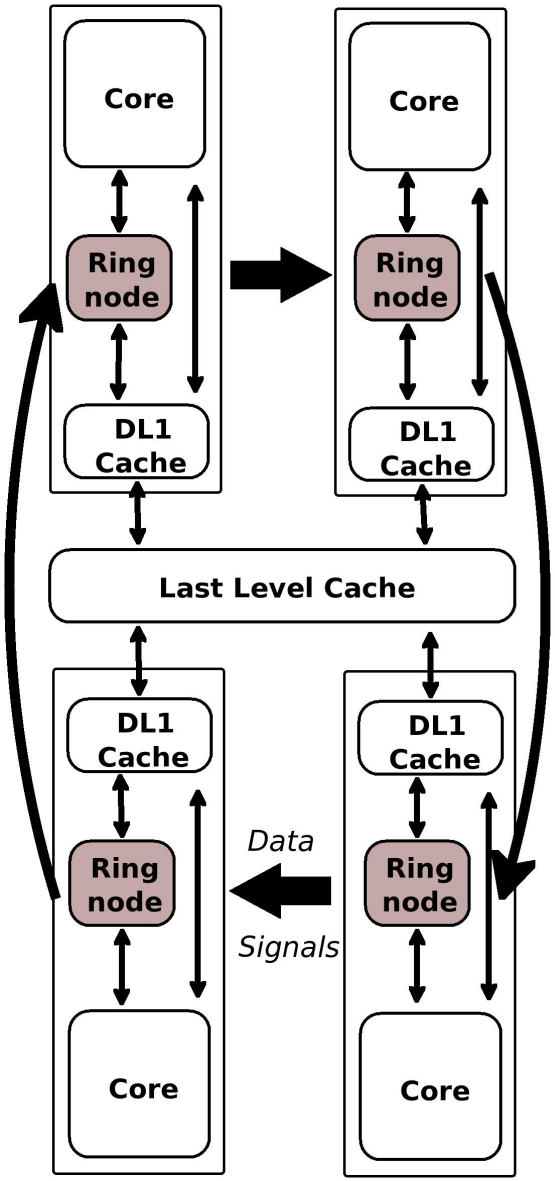
Light Enhancement in Conventional Multicore Architecture



Light Enhancement in Conventional Multicore Architecture



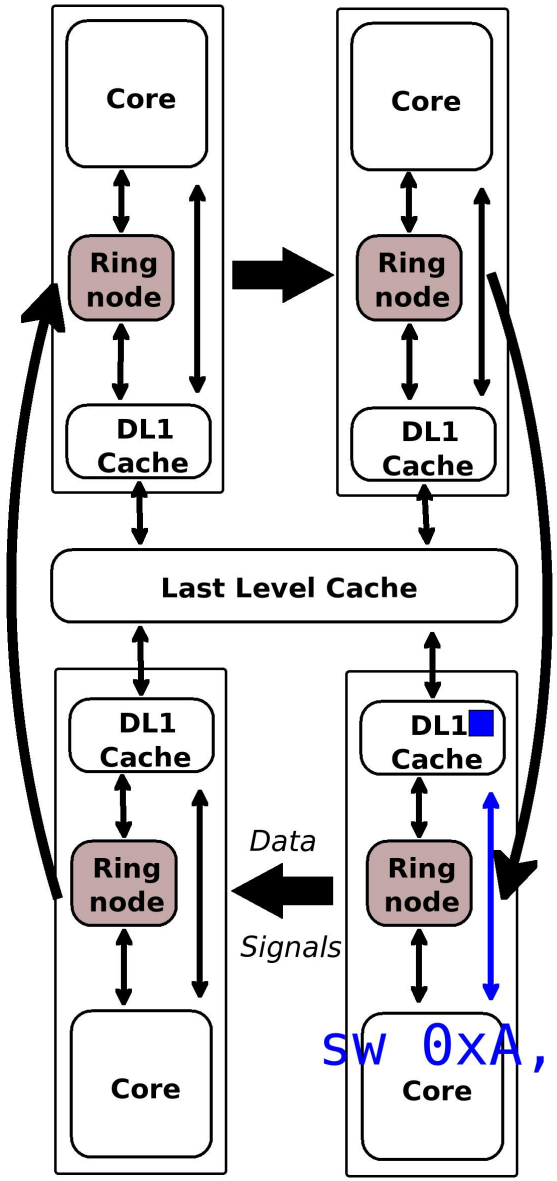
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

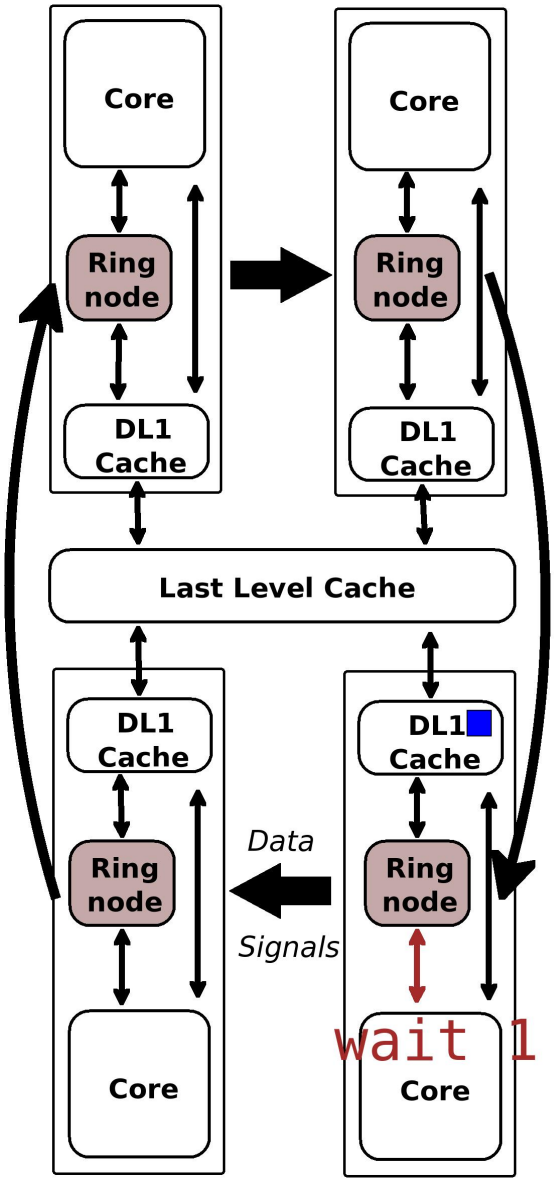
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

Light Enhancement in Conventional Multicore Architecture

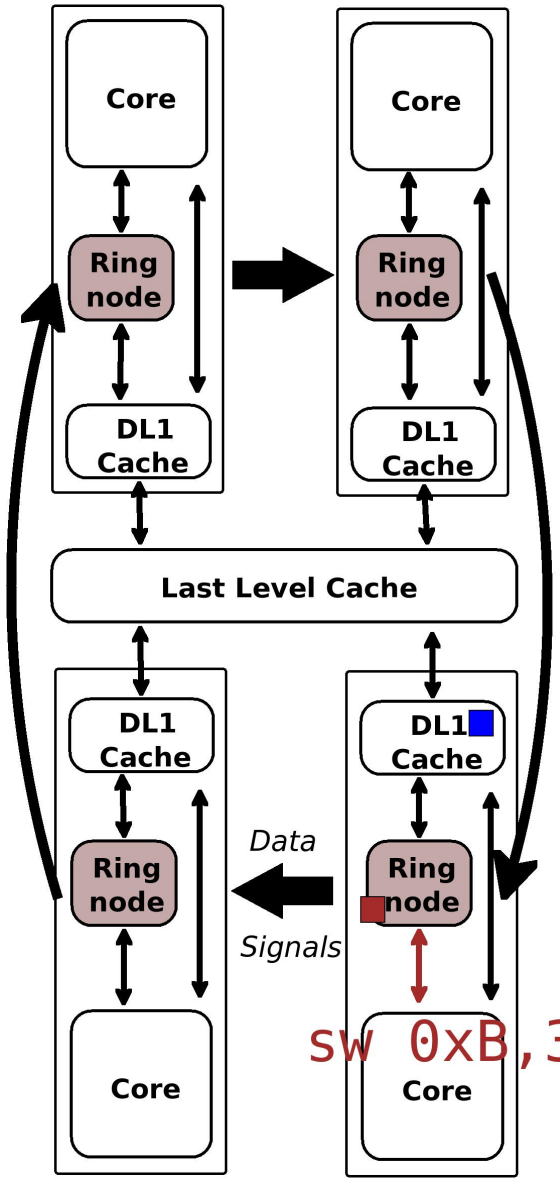



```
sw 0xA,5
```

```
wait 1
sw 0XB,3
signal 1
```

Sequential segment

Light Enhancement in Conventional Multicore Architecture

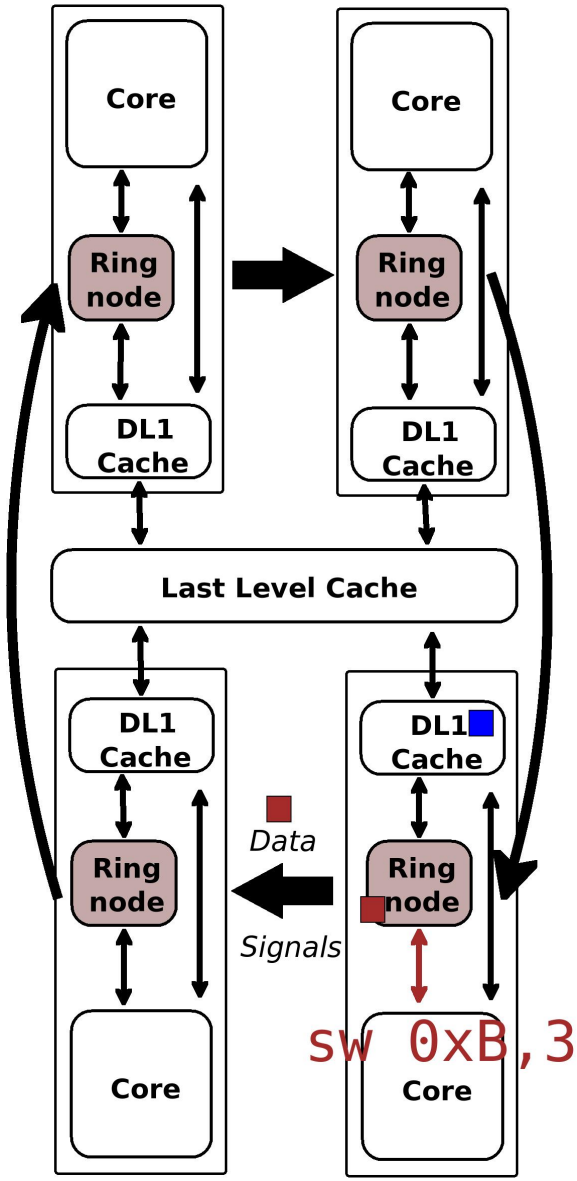


Delayed broadcast for unknown consumers

```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

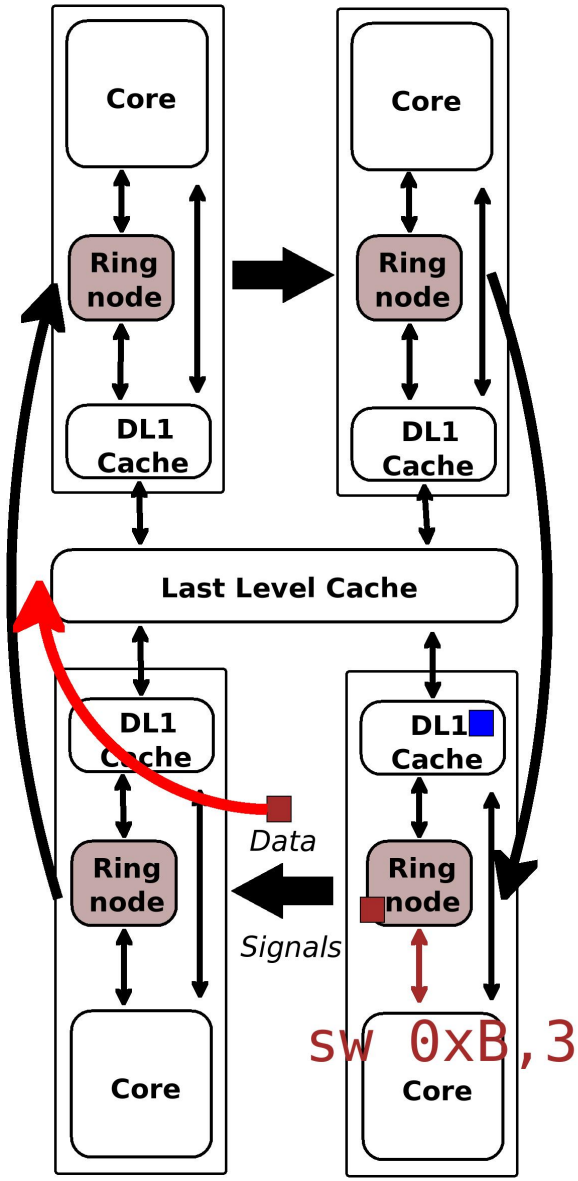
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

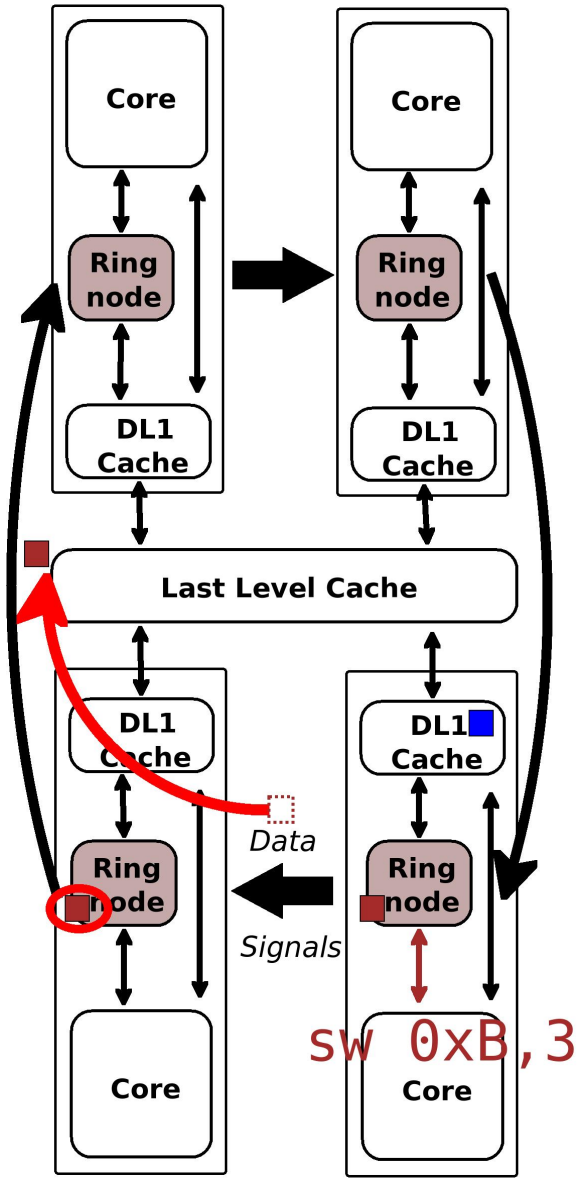
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

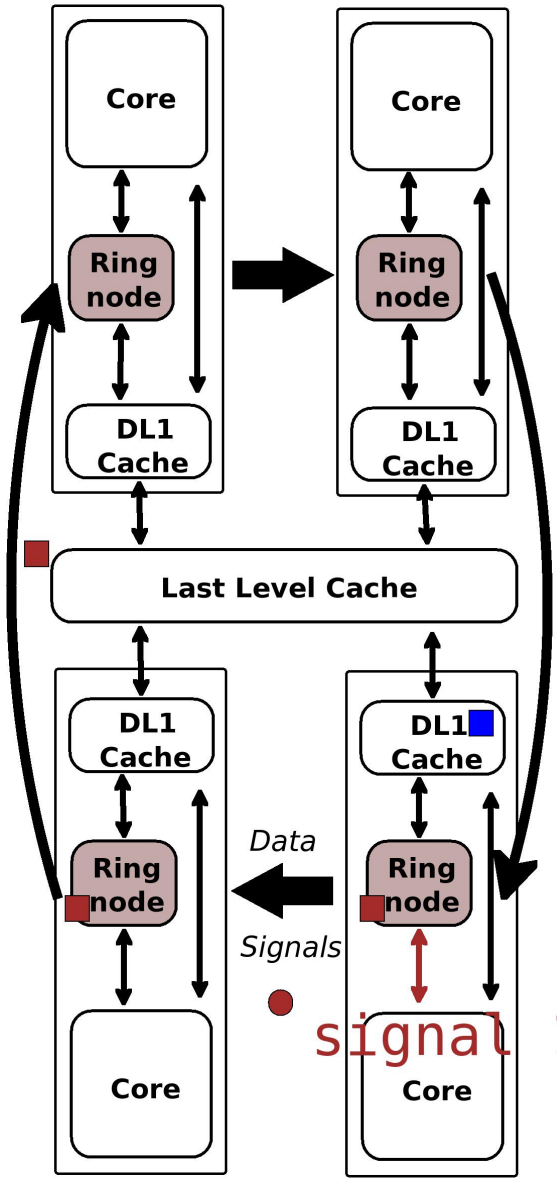
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

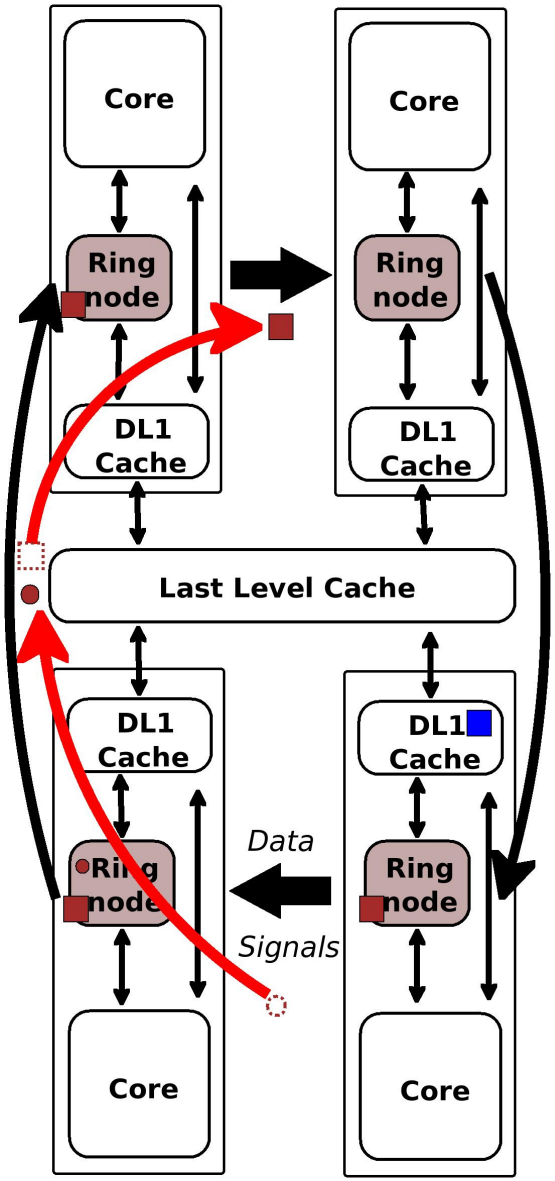
Light Enhancement in Conventional Multicore Architecture




```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

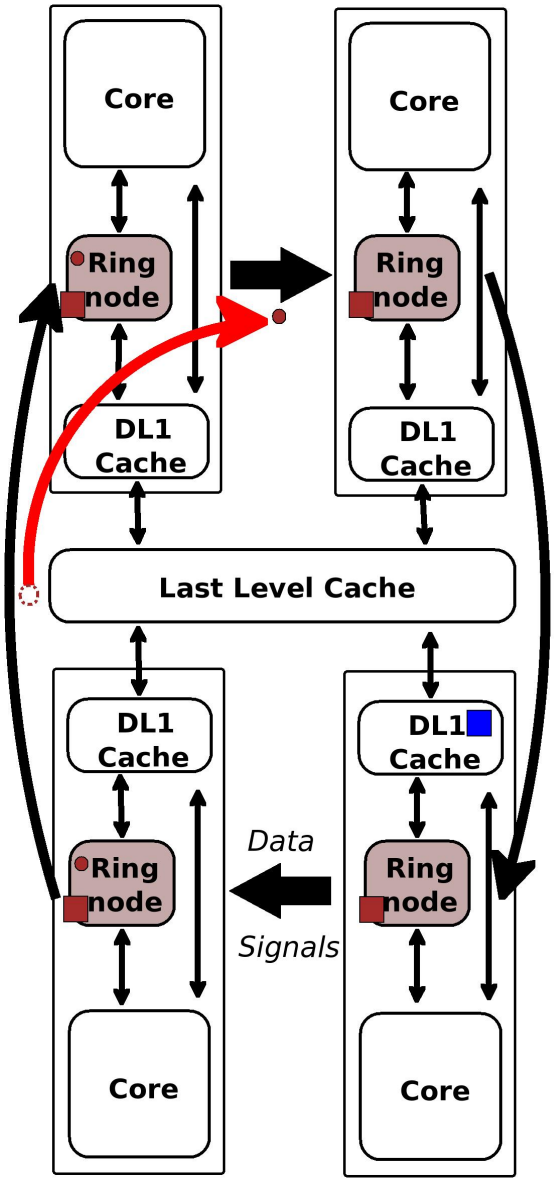
Light Enhancement in Conventional Multicore Architecture



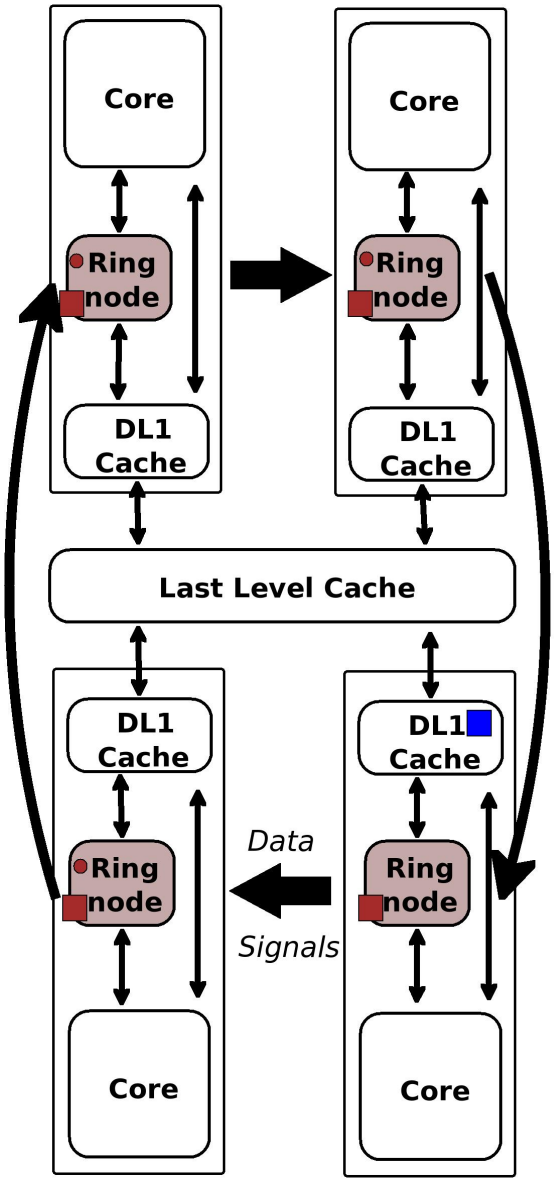

```
sw 0xA,5
wait 1
sw 0XB,3
signal 1
```

Sequential segment

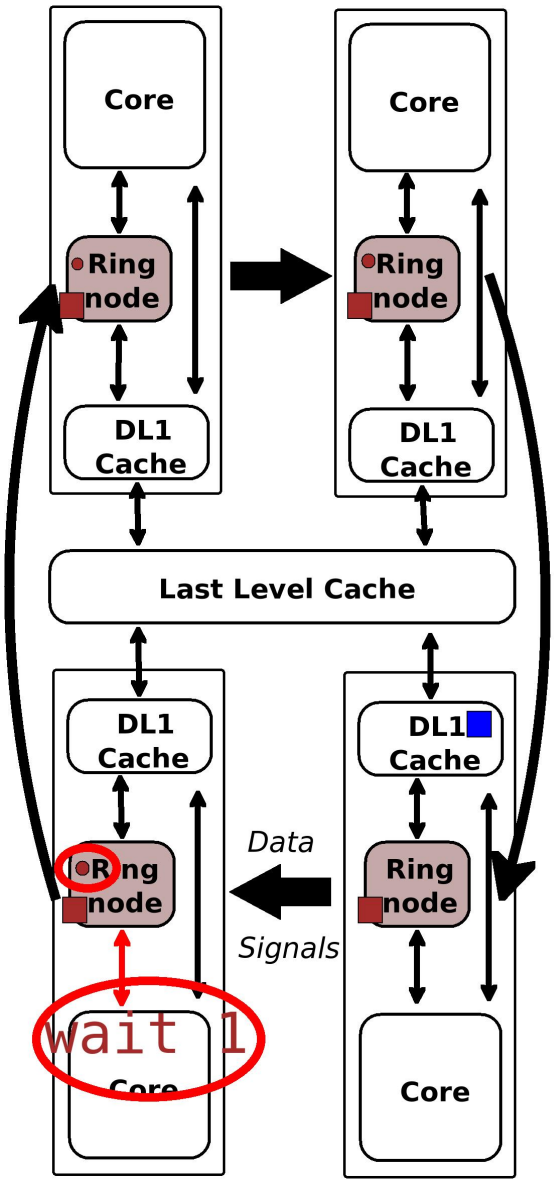
Light Enhancement in Conventional Multicore Architecture



Light Enhancement in Conventional Multicore Architecture



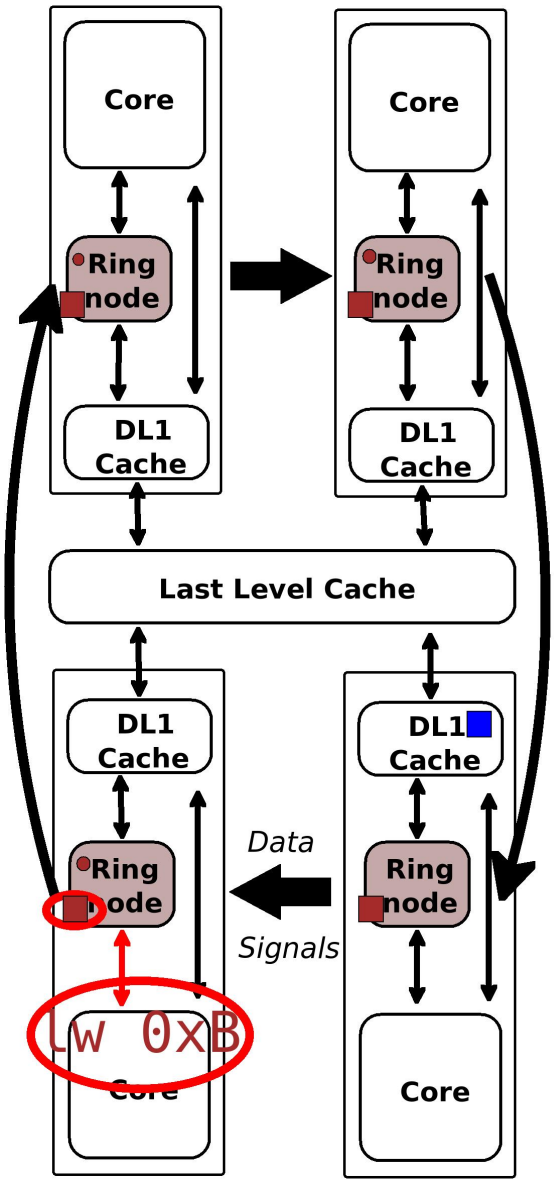
Light Enhancement in Conventional Multicore Architecture



Core synchronization

No communication cost!

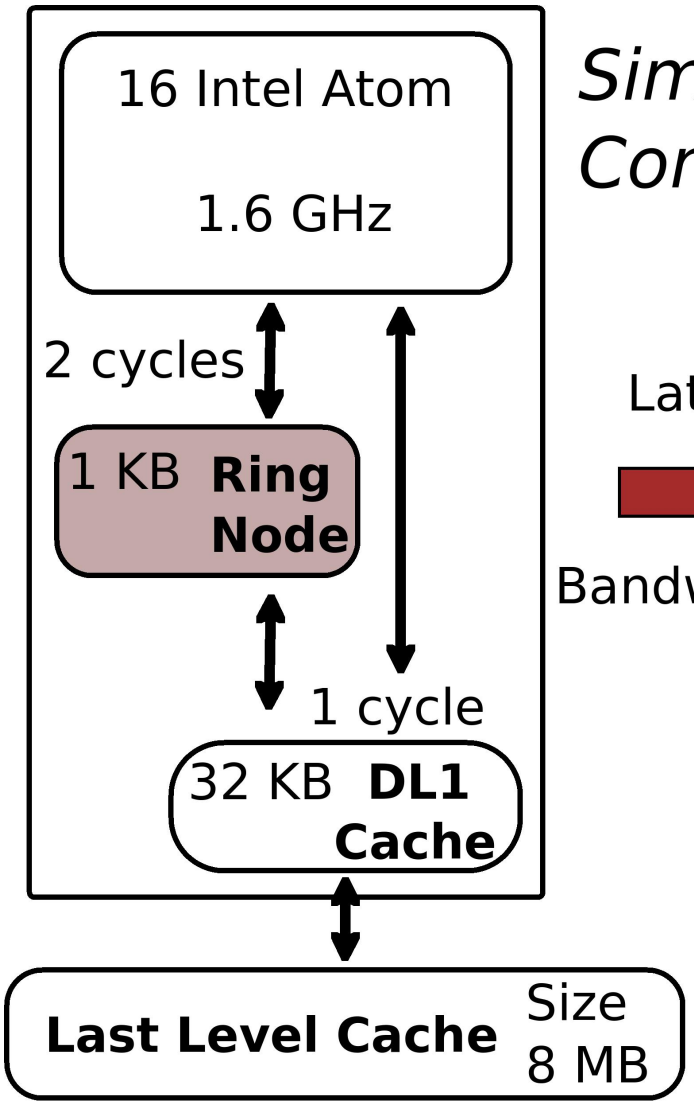
Light Enhancement in Conventional Multicore Architecture



Access to remote data locally

No communication cost!

Architecture Parameters



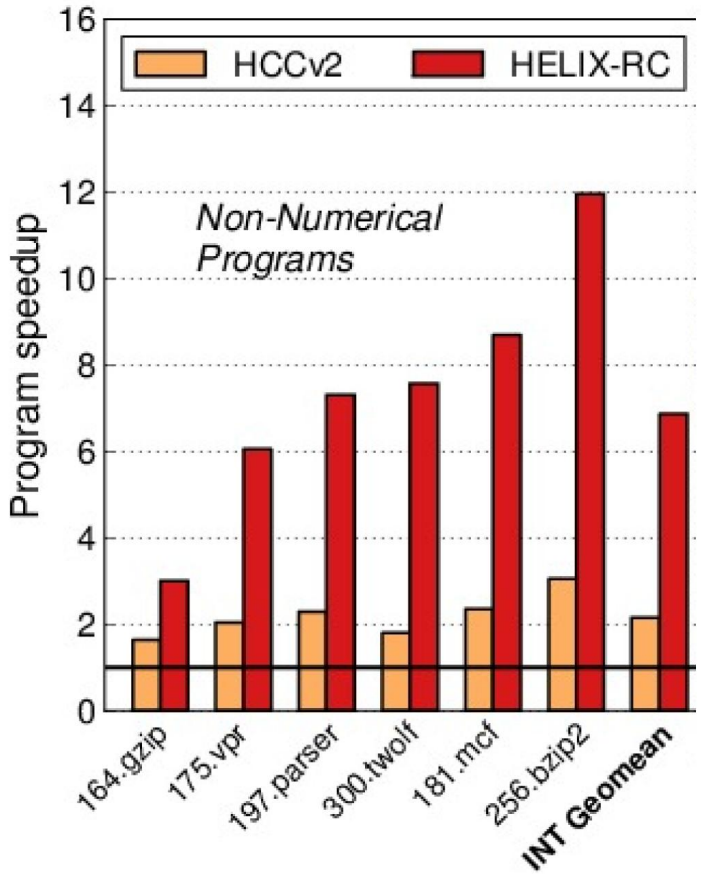
Simulator: XIOSim, DRAMSim
Compiler : ILDJIT (LLVM)

Latency: 1 cycle

Bandwidth: 70 bits for signals
68 bits for data

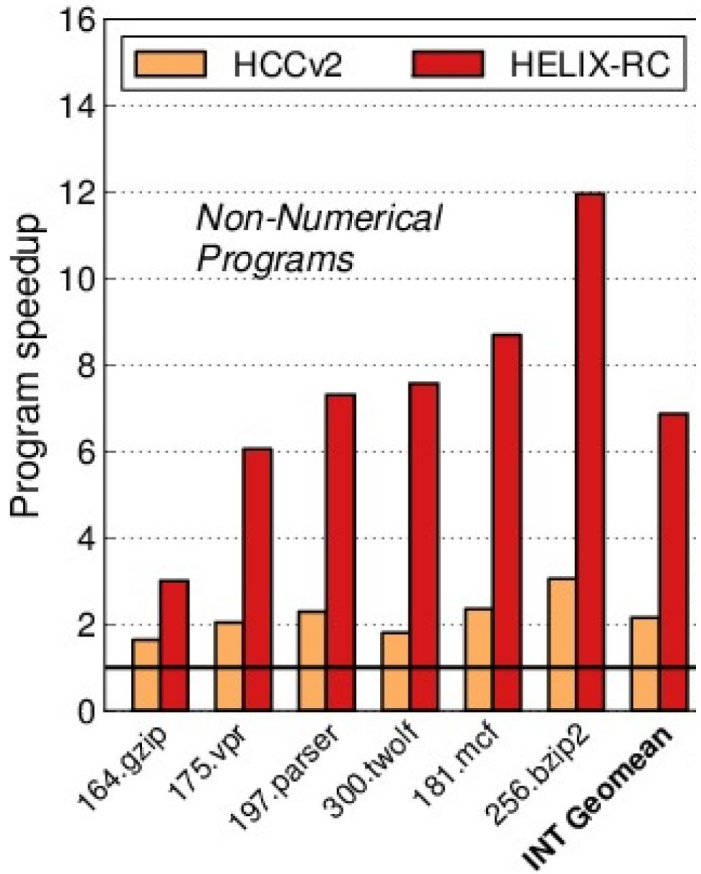
*["XIOSim:Power-performance Modeling of Mobile x86 Cores"
ISLPED 2012, Svilen Kanev et al.]*

The Importance of the Co-Design



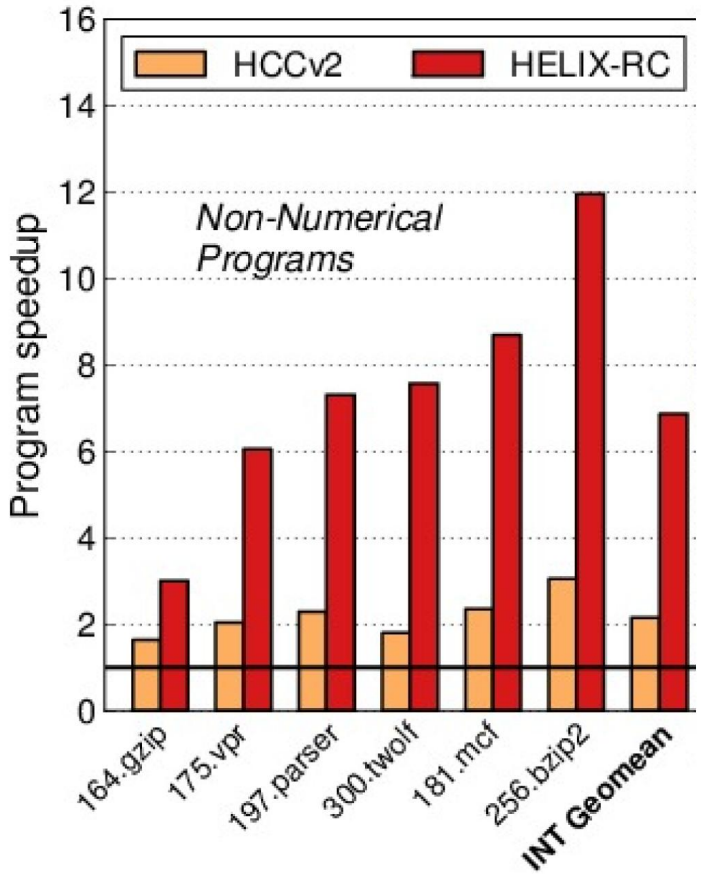
- Compiler-architecture co-design is effective for *non-numerical* workloads

The Importance of the Co-Design



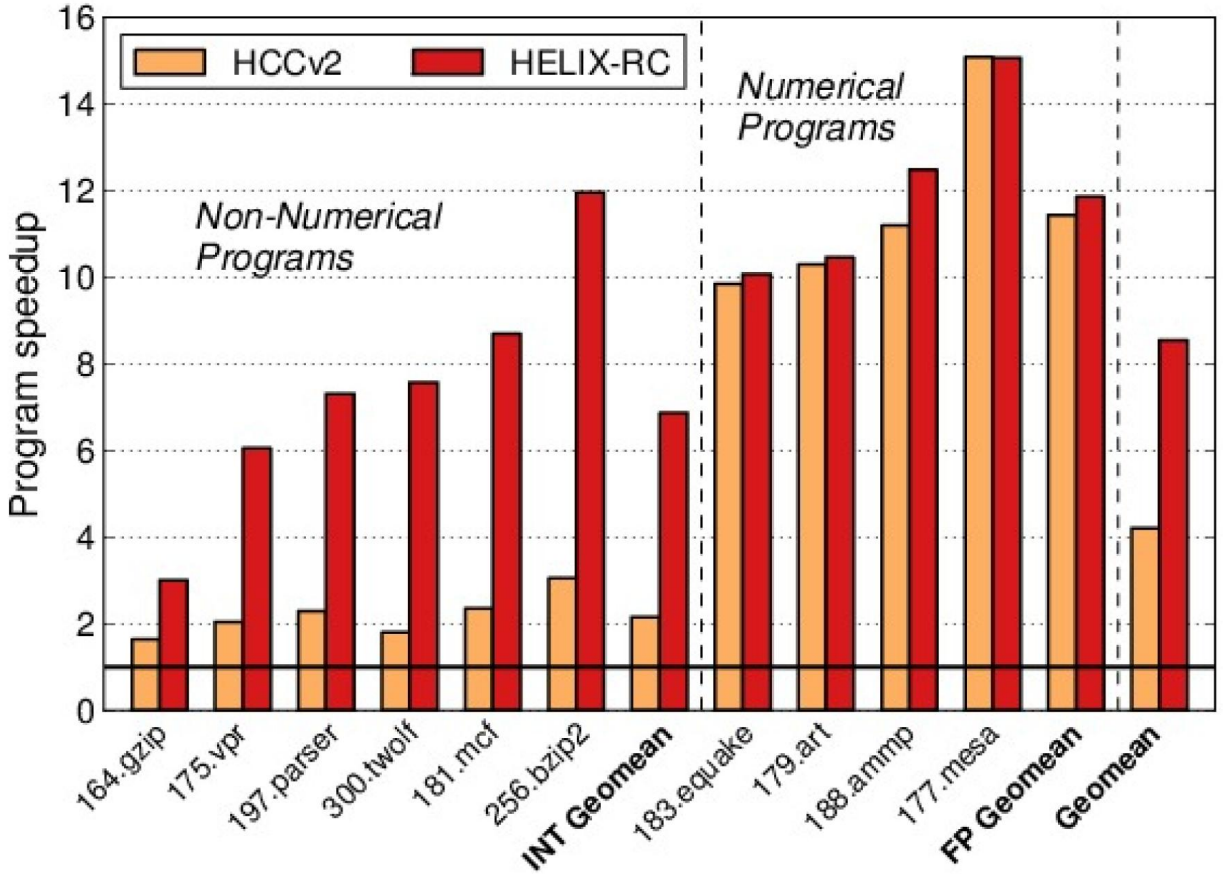
- Compiler-architecture co-design is effective for *non-numerical* workloads
- $3 <$ speedup < 12

The Importance of the Co-Design



- Compiler-architecture co-design is effective for *non-numerical* workloads
- $3 <$ speedup < 12
- No slowdown!

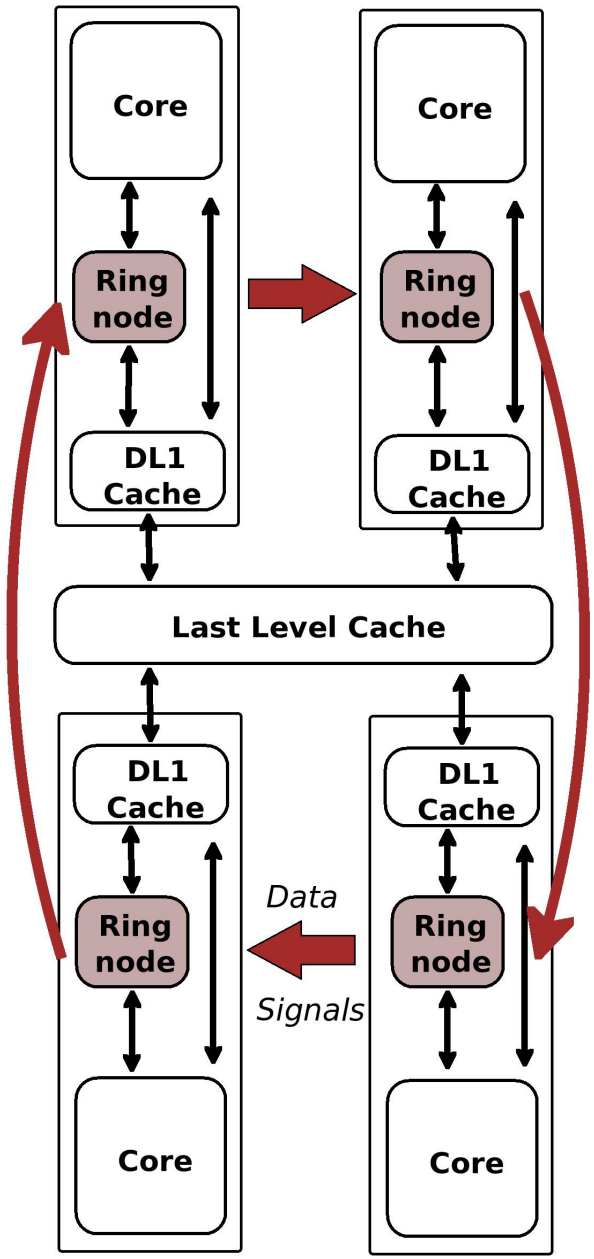
The Importance of the Co-Design



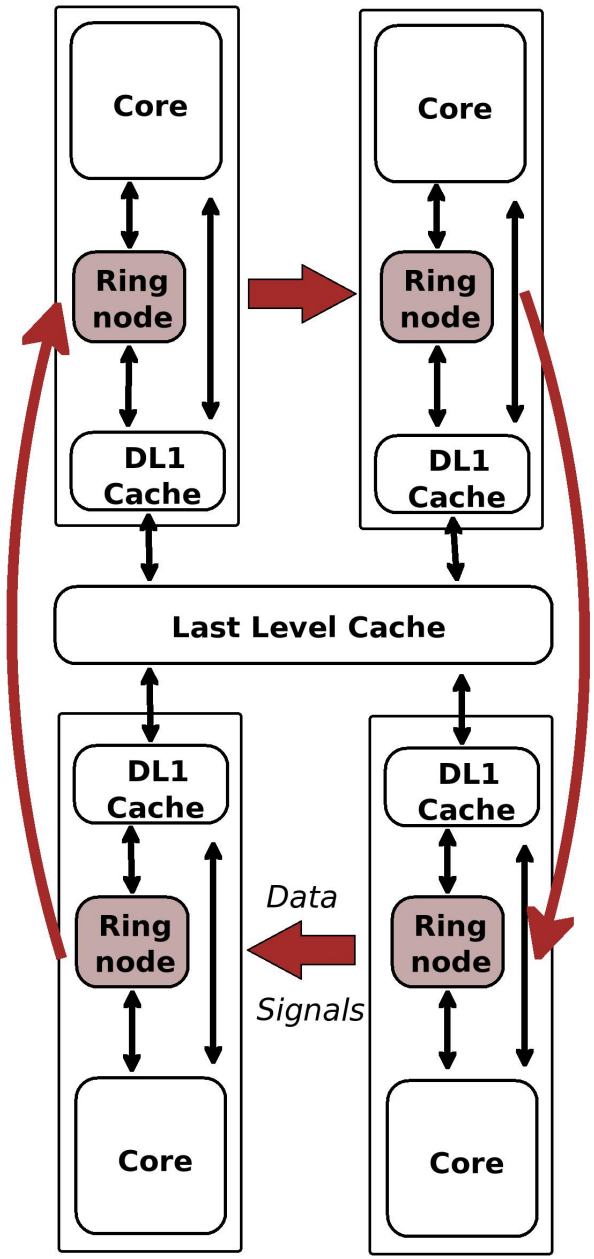
- Compiler-architecture co-design is effective for *non-numerical* workloads
- $3 <$ speedup < 12
- No slowdown!

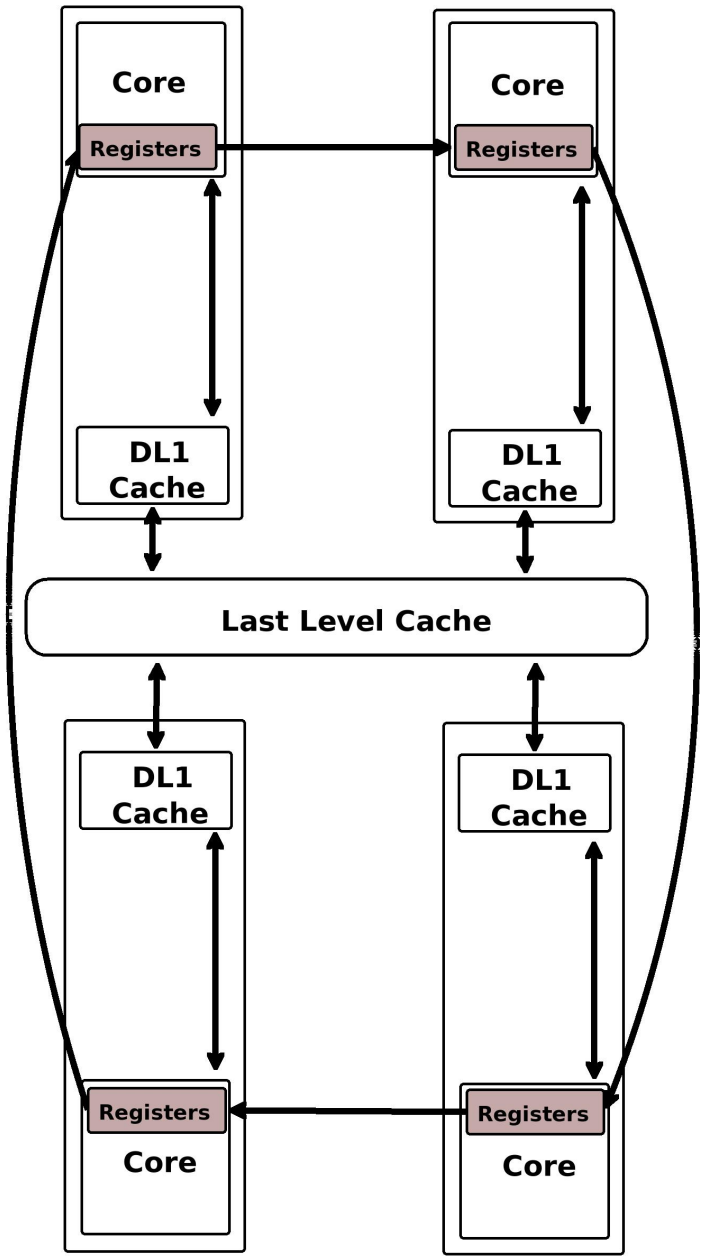
Ring Cache vs. Ring Register

Ring Cache vs. Ring Register

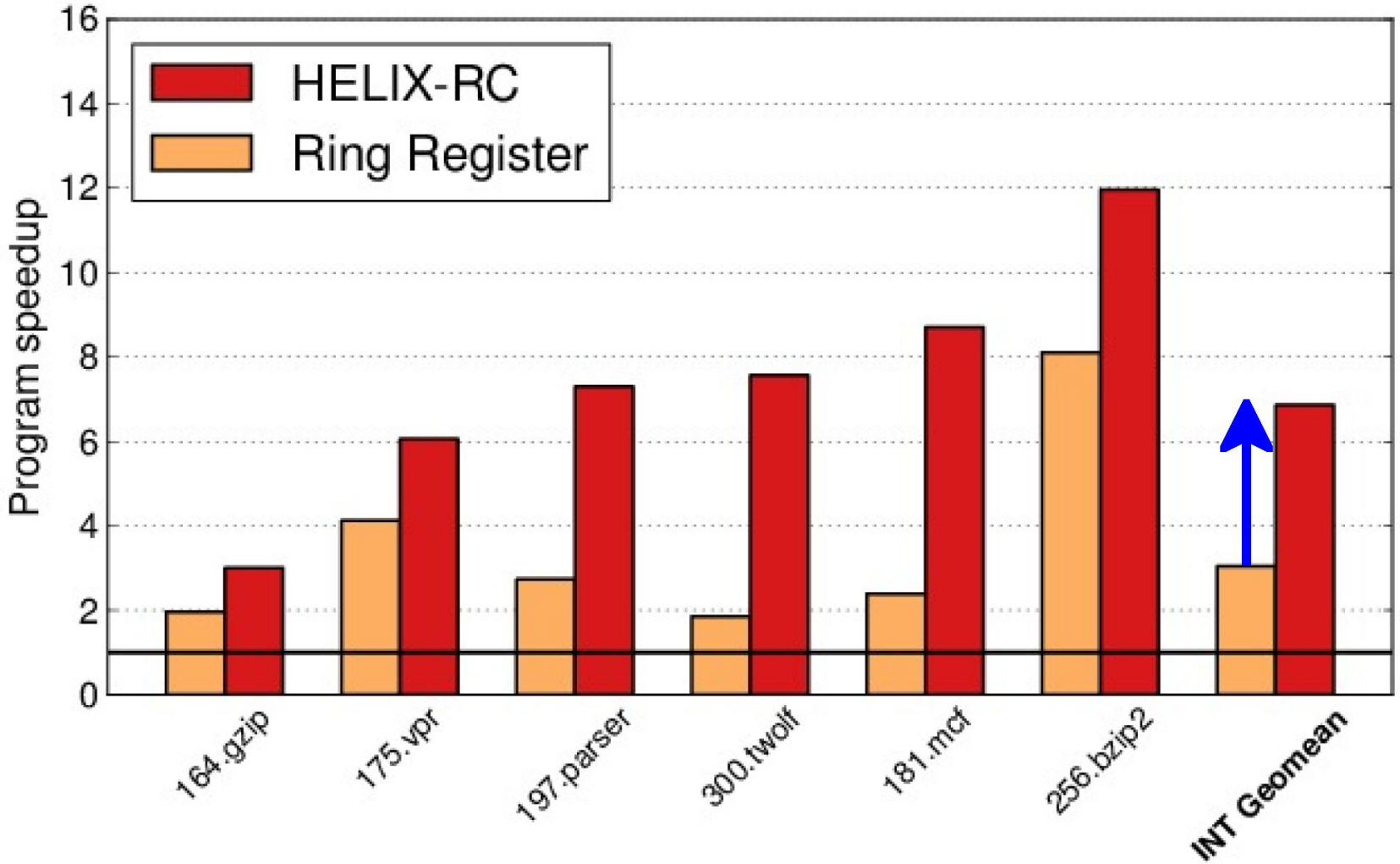


Ring Cache vs. Ring Register

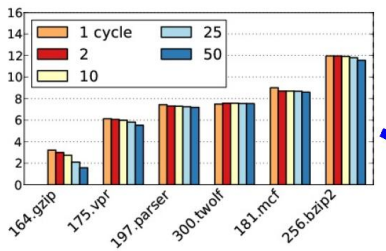


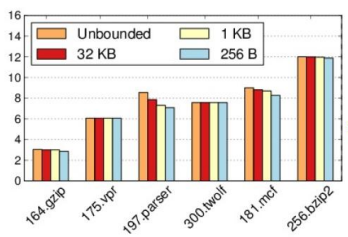


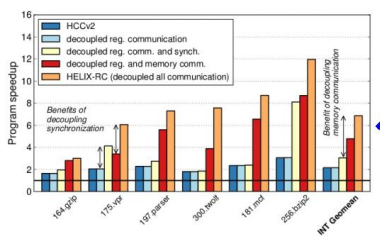
The Importance of a Cache-Based Scheme

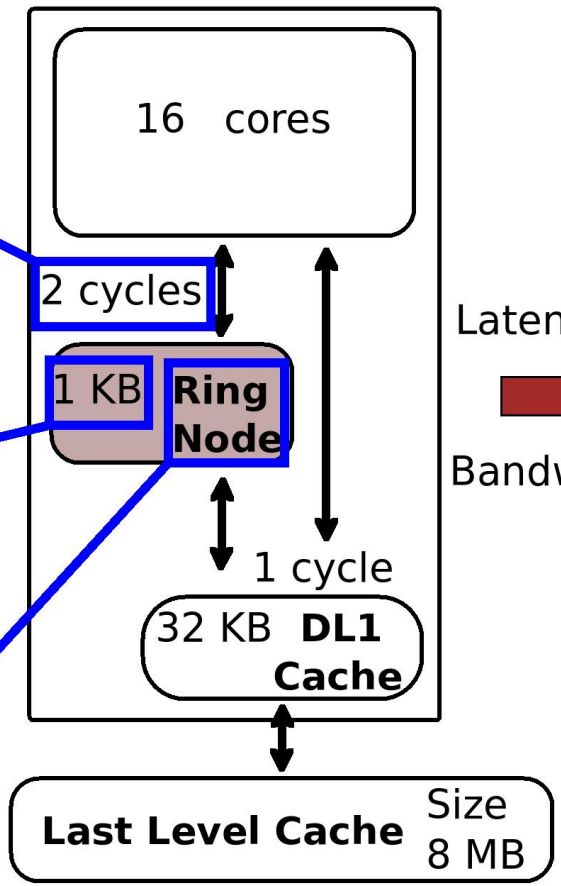


Ring Cache Parameter Analysis

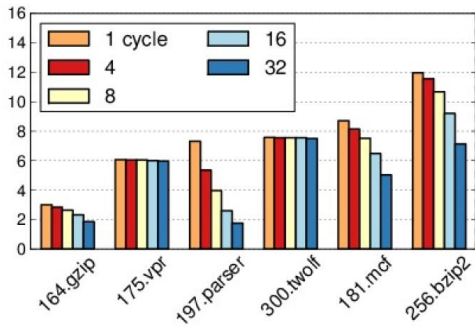




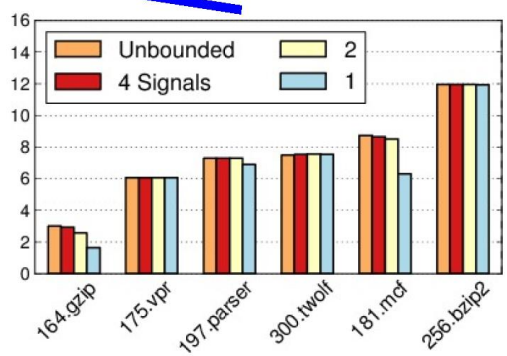




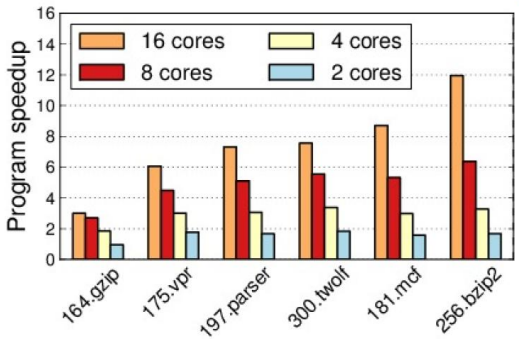
Latency: 1 cycle

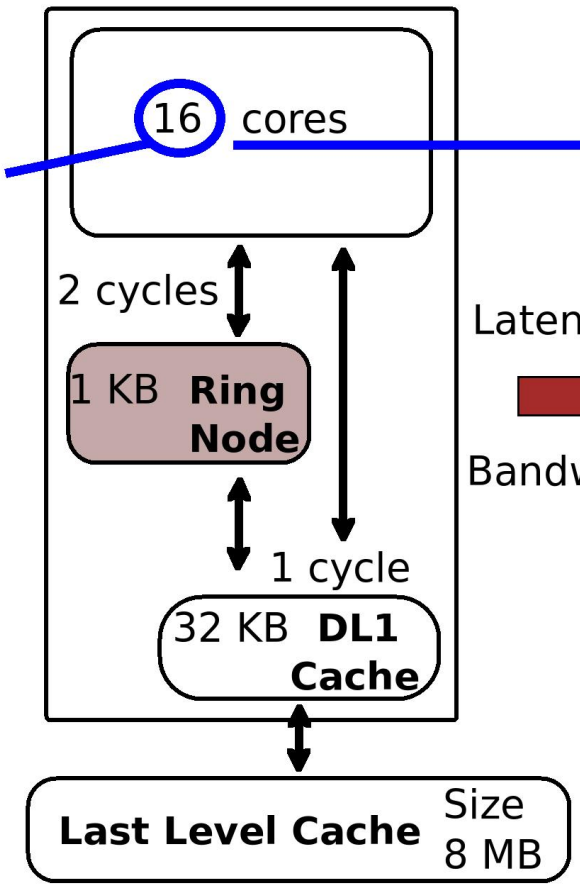


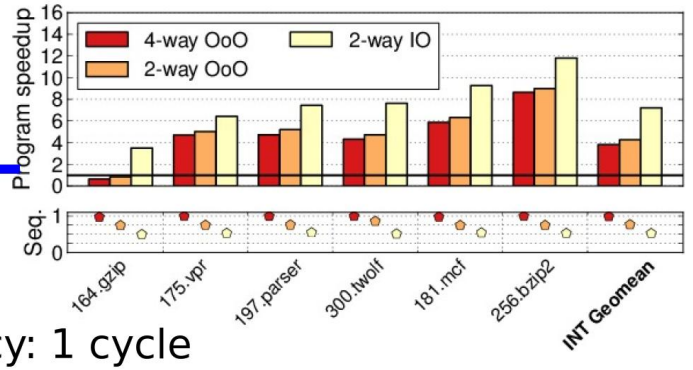
Bandwidth: 70 bits for signals



Core Parameters Analysis







Latency: 1 cycle

Bandwidth: 70 bits for signals

Conclusion

Ring Cache

makes
automatic parallelization practical
for conventional multicores

Conclusion

HELIX-RC

- Small loops $\Rightarrow$ few frequent dependences
  - Cut communication latency
  - Proactive data forwarding $\Rightarrow$ $\sim$0 communication latency

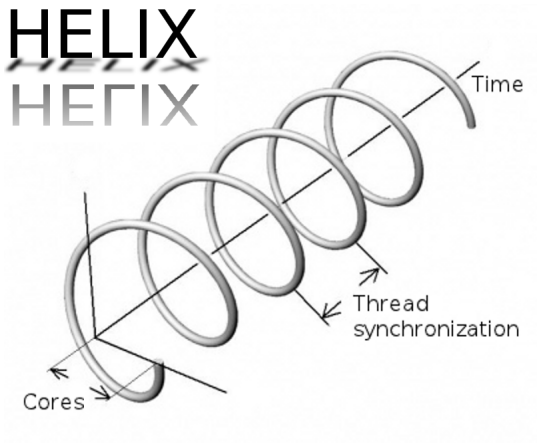
Conclusion

HELIX-RC

- Small loops ⇒ few frequent dependences
  - Cut communication latency
  - Proactive data forwarding ⇒ ∼0 communication latency

Questions?

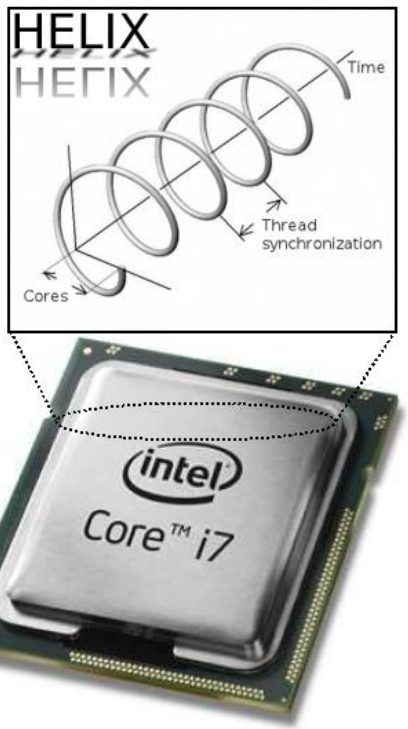
http://helix.eecs.harvard.edu



Conclusion

Conclusion

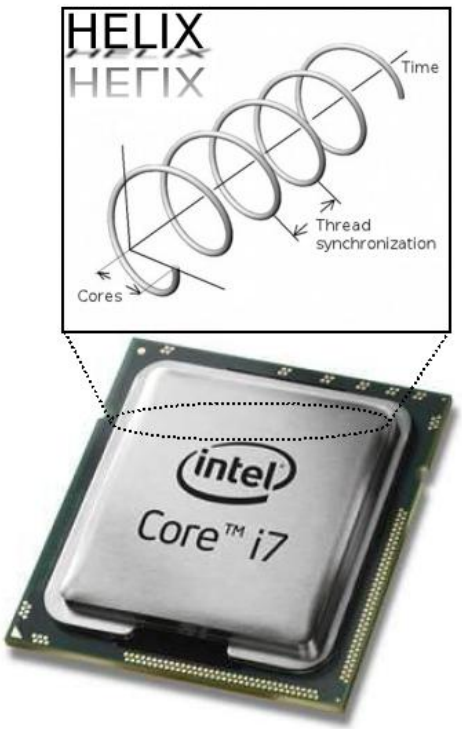
Real Platform → *Hardware support*



2x

Conclusion

Real platform ← → *Hardware support*

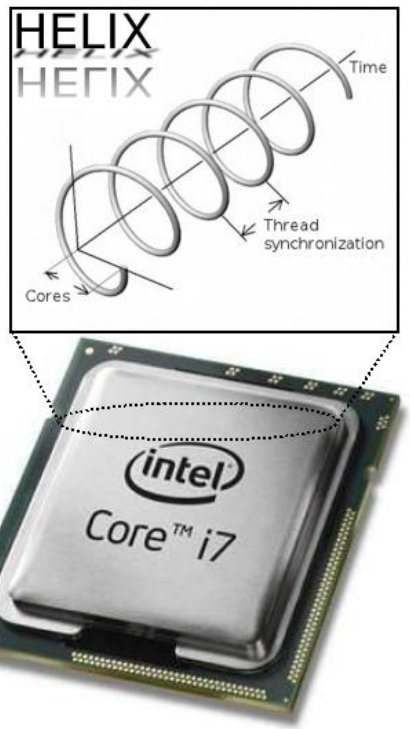


2x

HELIX-RC **7x**

Conclusion

Real platform ← → *Hardware support*



2x

HELIX-RC 7x

We will release binaries for both

http://helix.eecs.harvard.edu