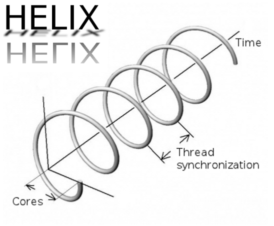
The HELIX Project: Overview and Directions

Simone Campanoni, Timothy M. Jones, Glenn Holloway
Gu-Yeon Wei, David Brooks

HARVARD
School of Engineering and Applied Sciences



Summary

- Motivation
- The HELIX Research Project
- HELIX on commodity processors
- Adaptive HELIX

Project Goal

Project Goal

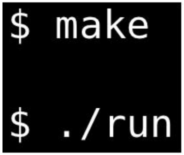
Making the extraction of thread-level parallelism mainstream

Project Goal

Making the extraction of thread-level parallelism mainstream

Project Goal

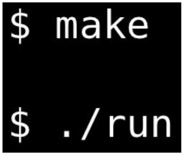
Making the extraction of thread-level parallelism mainstream



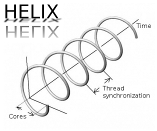
```
$ make

$ ./run
```

Project Goal

Making the extraction of thread-level parallelism mainstream

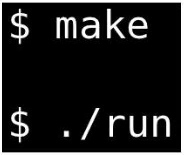


```
$ make

$ ./run
```

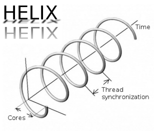


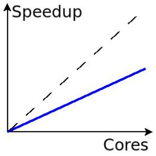
Project Goal

Making the extraction of thread-level parallelism mainstream



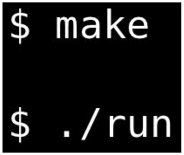
```
$ make

$ ./run
```



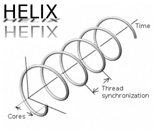


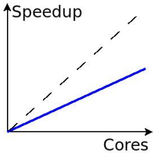
Project Goal

Making the extraction of thread-level parallelism mainstream



```
$ make

$ ./run
```

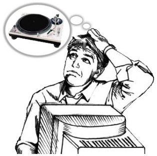


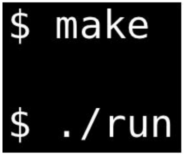


Instead of

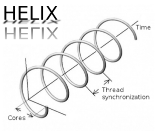
Project Goal

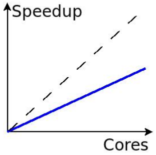
Making the extraction of thread-level parallelism mainstream





```
$ make

$ ./run
```



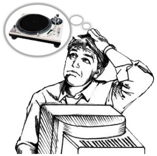


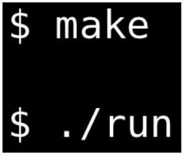
Instead of

- Software engineer

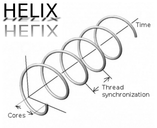
Project Goal

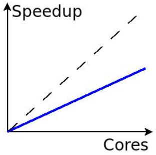
Making the extraction of thread-level parallelism mainstream





```
$ make

$ ./run
```



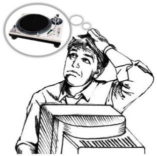


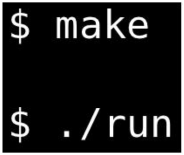
Instead of

- Software engineer
- Compilers

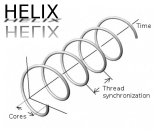
Project Goal

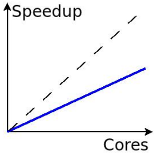
Making the extraction of thread-level parallelism mainstream





```
$ make

$ ./run
```



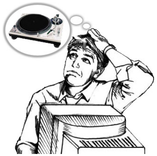
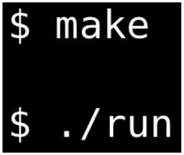


Instead of

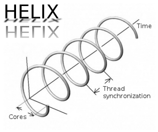
- Software engineer
- Compilers
- Computer architecture

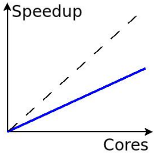
Project Goal

Making the extraction of thread-level parallelism mainstream

```
$ make

$ ./run
```



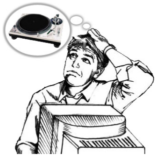


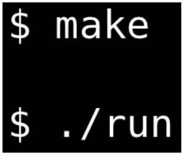
Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI

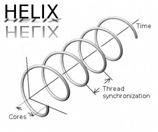
Project Goal

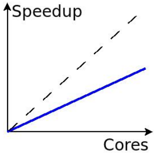
Making the extraction of thread-level parallelism mainstream





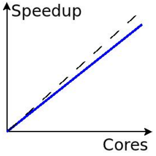
```
$ make

$ ./run
```





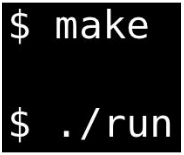
Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI

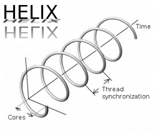


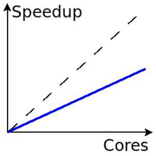
Project Goal

Making the extraction of thread-level parallelism mainstream



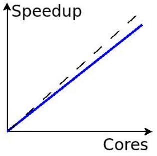
```
$ make

$ ./run
```

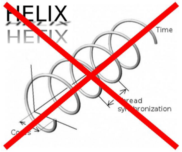




Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI





Motivation

Extraction of Thread-Level-Parallelism (TLP)

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

Provide more parallelism

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

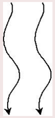
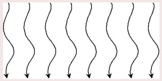
Provide more parallelism

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

Provide more parallelism

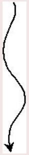
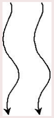
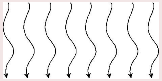



Motivation

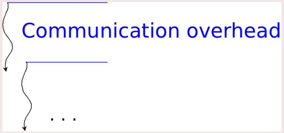
Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

Provide more parallelism

Reduce communication overhead

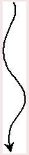


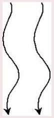
Motivation

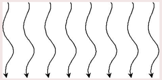
Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

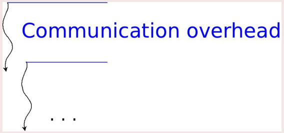
Provide more parallelism

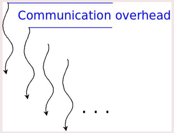






Reduce communication overhead



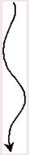
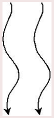
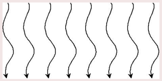


Motivation

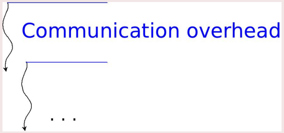
Extraction of Thread-Level-Parallelism (TLP)

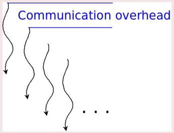
- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

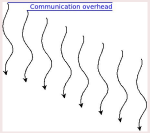
Provide more parallelism

Reduce communication overhead





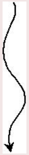
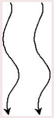
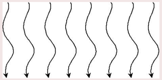


Motivation

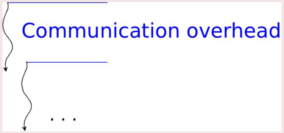
Extraction of Thread-Level-Parallelism (TLP)

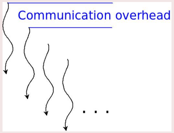
- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$

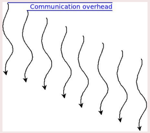
Provide more parallelism

Reduce communication overhead







Parallelism $\Leftrightarrow$ *communication*

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Automatic approaches target *loops*

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Automatic approaches target *loops*

- General rule:
  90% of the execution is spent in 10% of the code

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Automatic approaches target *loops*

- General rule:

more than 90% of the execution is spent in less than 10% of the code

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Automatic approaches target *loops*

- General rule:

more than 90% of the execution is spent in less than 10% of the code

- 10% of the code = hot *loops*

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Automatic approaches target *loops*

- General rule:

more than 90% of the execution is spent in less than 10% of the code

- 10% of the code = hot *loops*

- Our analysis:

Covering $\geq$ 98% of program by selecting loops properly is possible

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

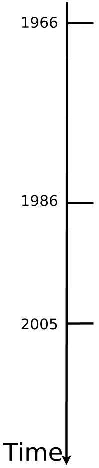
Main automatic approaches proposed:

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

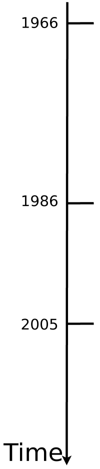


Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:



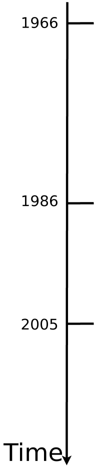
DOALL

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:



DOALL

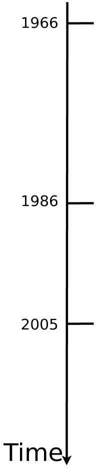
Speedup increases with number of cores

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:



DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

2005

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

2005

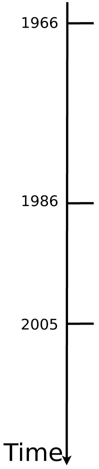
Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:



DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

2005

DSWP

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

2005

DSWP

Speedup are stable on inter-core communication delay

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

2005

DSWP

Speedup are stable on inter-core communication delay

Hard to predict speedup

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

2005

DSWP

Speedup are stable on inter-core communication delay

Hard to predict speedup

- Hard to avoid slowdown

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

1966

DOALL

Speedup increases with number of cores

Limited applicability

- Loop-carried dependences not handled

1986

DOACROSS

Applicable to a broader set of programs

Extremely sensitive to inter-core communication

2005

DSWP

Speedup are stable on inter-core communication delay

Hard to predict speedup

- Hard to avoid slowdown

Time

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

Is there a way to achieve all of these?

Speedup increases with number of cores

Applicable to a broader set of programs

Speedup are stable on inter-core communication delay

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

Is there a way to achieve all of these?

Speedup increases with number of cores

Applicable to a broader set of programs

Speedup are stable on inter-core communication delay

Produce predictable speedup

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

HELIX

Speedup increases with number of cores

Applicable to a broader set of programs

Speedup are stable on inter-core communication delay

Produce predictable speedup

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

HELIX

Speedup increases with number of cores

General purpose technique

Speedup are stable on inter-core communication delay

Produce predictable speedup

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

HELIX

Speedup increases with number of cores

General purpose technique

Speedup are stable on inter-core communication delay

Produce speedup predictable enough to avoid slowdown

Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: $\Uparrow$ performance $\Leftrightarrow$ TLP $\Uparrow$
- Manual approach: $\Uparrow$ software development time

Main automatic approaches proposed:

HELIX

Speedup increases with number of cores

General purpose technique

DOACROSS $<$ Stability of speedup $<$ DSWP

Produce speedup predictable enough to avoid slowdown

Motivation (2)

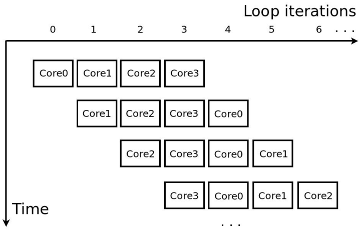
HELIX

- General purpose technique
- Avoid slowdown (always)
- $|\text{threads}| \leq |\text{loop iterations}|$
    - TLP extracted between loop iterations

Motivation (2)

HELIX

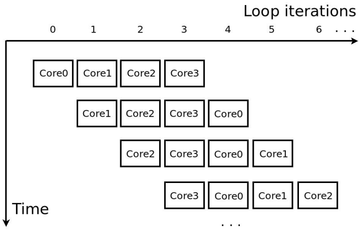
- General purpose technique
- Avoid slowdown (always)
- $|\text{threads}| \leq |\text{loop iterations}|$
    - TLP extracted between loop iterations
    - Iterations grouped on modular value

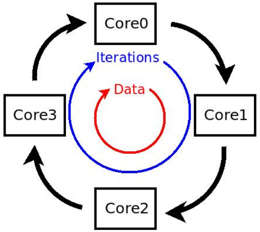


Motivation (2)

HELIX

- General purpose technique
- Avoid slowdown (always)
- $|\text{threads}| \leq |\text{loop iterations}|$
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
  - Cores organized as a ring

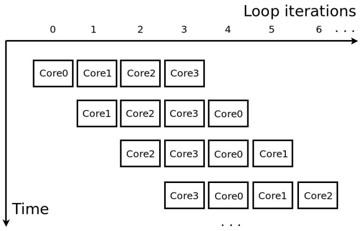


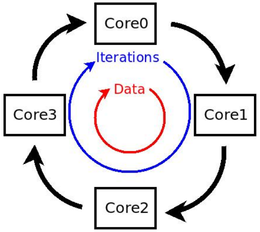


Motivation (2)

HELIX

- General purpose technique
- Avoid slowdown (always)
- $|\text{threads}| \leq |\text{loop iterations}|$
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
  - Cores organized as a ring
- Automatic selection of loops

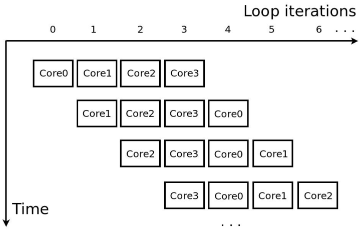


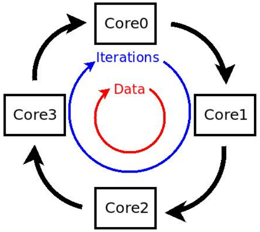


Motivation (2)

HELIX

- General purpose technique
- Avoid slowdown (always)
- $|\text{threads}| \leq |\text{loop iterations}|$
    - TLP extracted between loop iterations
    - Iterations grouped on modular value
    - Cores organized as a ring
- Automatic selection of loops
- *Easy* to implement

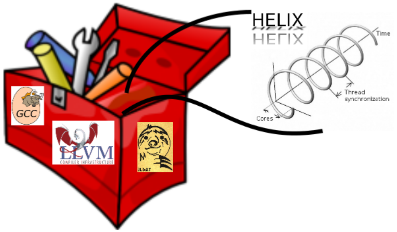




Motivation (2)

HELIX

- General purpose technique
- Avoid slowdown (always)
- |threads| $\leq$ |loop iterations|
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
  - Cores organized as a ring
- Automatic selection of loops
- *Easy* to implement



Summary

- Motivation
- The HELIX Research Project
- HELIX on commodity processors
- Adaptive HELIX

A Simple Idea

```
for (...){
    1: a = update(a);
    2: work1(a);
    3: b = update(b);
    4: work2();
}
```

- A simple program

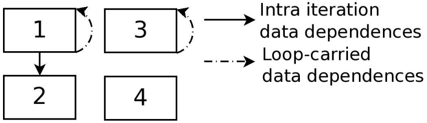
A Simple Idea

```
for (...){
    1: a = update(a);
    2: work1(a);
    3: b = update(b);
    4: work2();
}
```

- Loop-carried data dependences ▨

A Simple Idea

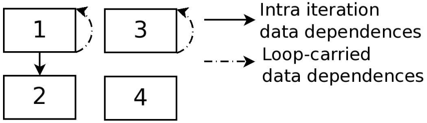
```
for (...){
      1: a = update(a);
      2: work1(a);
      3: b = update(b);
      4: work2();
}
```


1
2
3
4
Intra iteration data dependences
Loop-carried data dependences

- Idea: exploit independent instructions

A Simple Idea

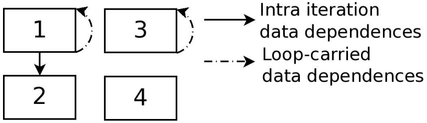
```
for (...){
      1: a = update(a);
      2: work1(a);
      3: b = update(b);
      4: work2();
}
```

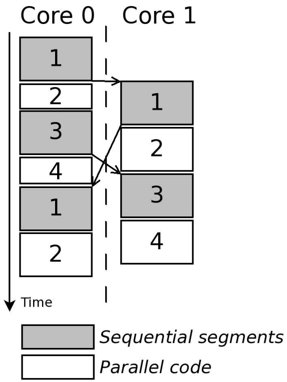

1
2
3
4
Intra iteration data dependences
Loop-carried data dependences

- Idea: exploit independent instructions *and*

A Simple Idea

```
for (...){
      1: a = update(a);
      2: work1(a);
      3: b = update(b);
      4: work2();
}
```

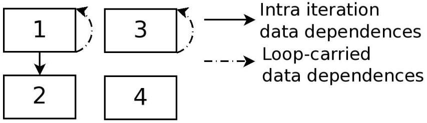

1
2
3
4
Intra iteration data dependences
Loop-carried data dependences



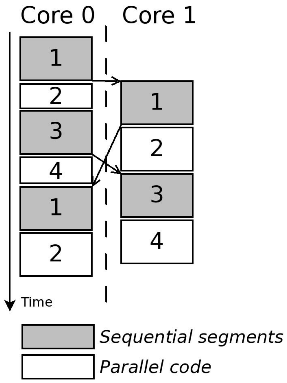
- Idea: exploit independent instructions *and* parallelism among sequential segments

A Simple Idea

```
for (...){
      1: a = update(a);
      2: work1(a);
      3: b = update(b);
      4: work2();
}
```



1
2
3
4
Intra iteration data dependences
Loop-carried data dependences



- Idea: exploit independent instructions *and*

  parallelism among sequential segments

Problem: amount of synchronization required increases drastically!

Status of HELIX

Prototype

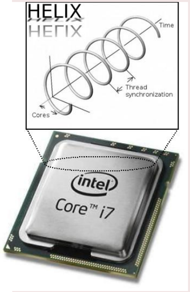
Status of HELIX

Prototype

- Target: commodity processors
  - Intel® Core™ i7-980X

Status of HELIX

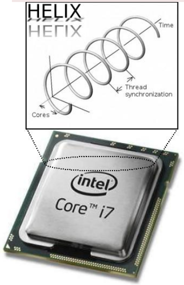
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X

Status of HELIX

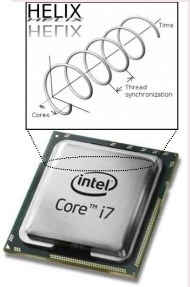
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation

Status of HELIX

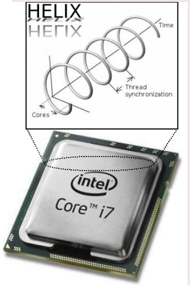
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time

Status of HELIX

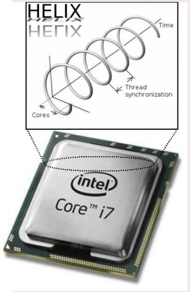
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**

Status of HELIX

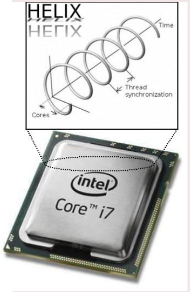
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - Constrain communication overhead

Status of HELIX

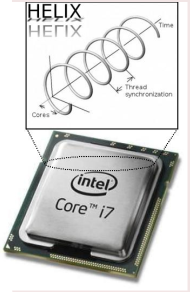
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

Status of HELIX

Prototype

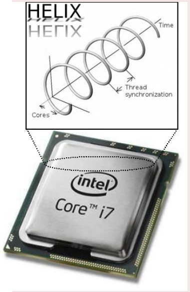


- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Status of HELIX

Prototype

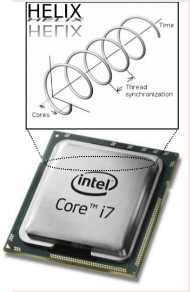


- Target: commodity processors
    - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
    - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Status of HELIX

Prototype



- Target: commodity processors
  - Intel® Core$^{\mathrm{TM}}$ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

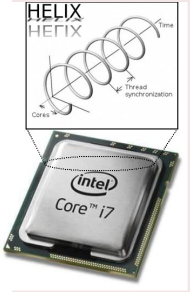
[CGO 2012, IEEE Micro 2012]

Hardware support

Hardware support

Status of HELIX

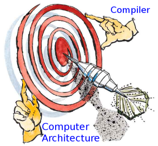
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

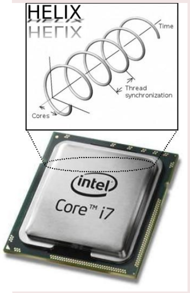
Hardware support



Hardware support

Status of HELIX

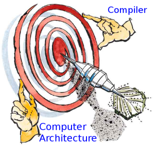
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Hardware support

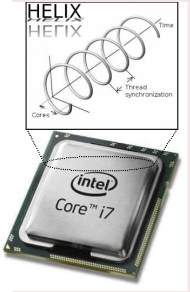


Hardware support

- Push HELIX to the limit

Status of HELIX

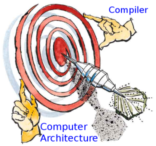
Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Hardware support

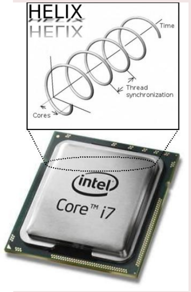


Hardware support

- Push HELIX to the limit
- Minor changes to commodity processors

Status of HELIX

Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

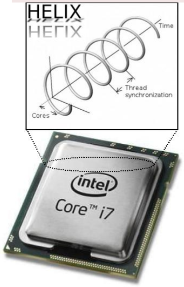
Adaptive HELIX

Hardware support

Adaptive HELIX

Status of HELIX

Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Adaptive HELIX

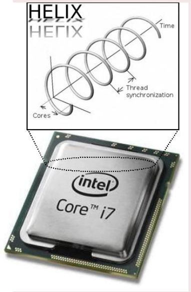
Hardware support

Adaptive HELIX

Adapt code at run time to:

Status of HELIX

Prototype



- Target: commodity processors
  - Intel® Core$^{\text{TM}}$ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Adaptive HELIX
Hardware support

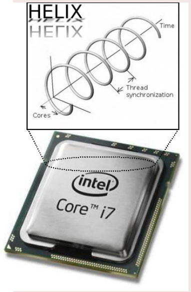
Adaptive HELIX

Adapt code at run time to:

- Parallel behavior

Status of HELIX

Prototype



- Target: commodity processors
  - Intel® Core™ i7-980X
- Static code generation
- Number of cores decided at compile time
- **Challenge: achieve speedup**
  - **Constrain communication overhead**

[CGO 2012, IEEE Micro 2012]

Adaptive HELIX
Hardware support

Adaptive HELIX

Adapt code at run time to:

- Parallel behavior
- System requirements

Summary

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Optimizations

Adopted solutions

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal
- Automatic selection of loops

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal
- Automatic selection of loops

Approach

- Select loops to parallelize

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal
- Automatic selection of loops

Approach

- Select loops to parallelize
  - Each loop $\in$ program is analyzed independently
  - These analysis are merged to identify the most profitable loops
  - Light off line profile based selection

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal
- Automatic selection of loops

Approach

- Select loops to parallelize
  - Each loop $\in$ program is analyzed independently
  - These analysis are merged to identify the most profitable loops
  - Light off line profile based selection
- Parallelize one loop at a time

HELIX on Commodity Processors

Overhead

Signalling

Notify threads

Data forwarding

Forward data between threads

Optimizations

Adopted solutions

- New code analysis to reduce the number of signals to send
- Code scheduling and use of SMT to reduce the delay per signal
- Automatic selection of loops

Approach

- Select loops to parallelize
  - Each loop $\in$ program is analyzed independently
  - These analysis are merged to identify the most profitable loops
  - Light off line profile based selection
- Parallelize one loop at a time
  - Each loop uses all cores decided at compile time

HELIX on Commodity Processors: Evaluation

Platform

- Intel$^{®}$ Core$^{\mathrm{TM}}$ i7-980X with six cores
  - Each operating at 3.33 GHz, with Turbo Boost disabled
- Three cache levels
  - The first two, 32KB and 256KB, are private to each core
  - All cores share the last level 12MB cache

HELIX on Commodity Processors: Evaluation

Platform

- Intel® Core™ i7-980X with six cores
  - Each operating at 3.33 GHz, with Turbo Boost disabled
- Three cache levels
  - The first two, 32KB and 256KB, are private to each core
  - All cores share the last level 12MB cache

Benchmarks

C benchmarks from SPEC CPU2000

HELIX on Commodity Processors: Evaluation

Platform

- Intel® Core™ i7-980X with six cores
  - Each operating at 3.33 GHz, with Turbo Boost disabled
- Three cache levels
  - The first two, 32KB and 256KB, are private to each core
  - All cores share the last level 12MB cache

Benchmarks

C benchmarks from SPEC CPU2000

Compiler

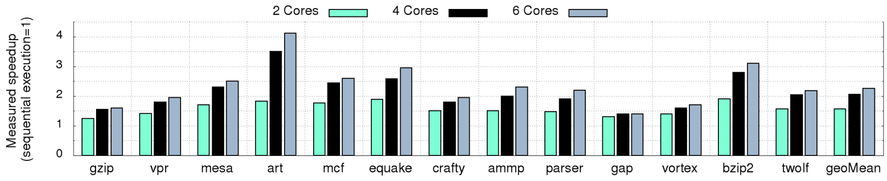
- HELIX has been implemented $\in$ static compiler ILDJIT

Speedup Obtained on a Real System

Overall program speedup

Speedup Obtained on a Real System

Overall program speedup



Notice: no slowdown

Summary

- Motivation
- The HELIX Research Project
- HELIX on commodity processors
- Adaptive HELIX

Adaptive HELIX

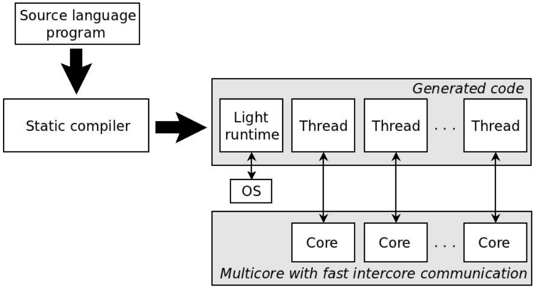
- Code produced for N cores

Adaptive HELIX

- Code produced for N cores
- The number of cores changes to M at run time
    - Performance
    - Multi-programs scenario

Adaptive HELIX

- Code produced for N cores
- The number of cores changes to M at run time
  - Performance
  - Multi-programs scenario

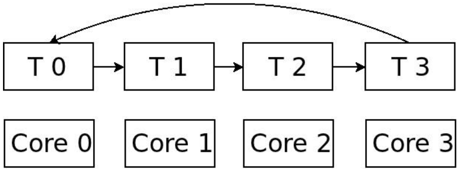


Adaptive HELIX: Cost

What is the cost of adapting the produced binary?

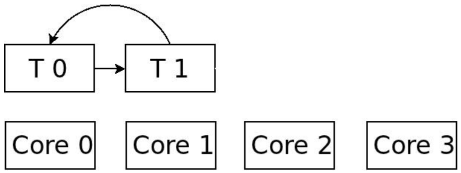
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



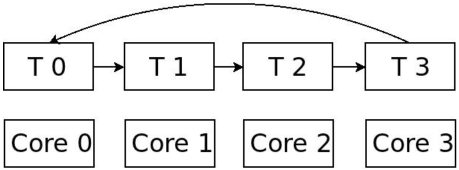
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



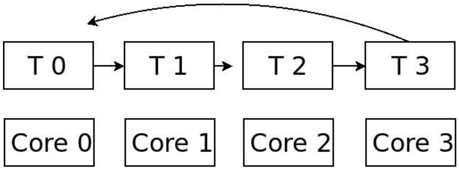
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



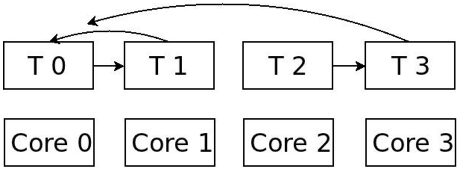
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



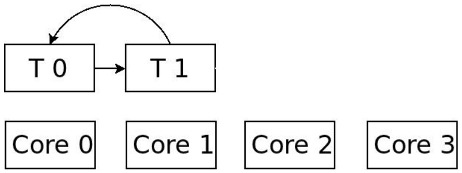
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



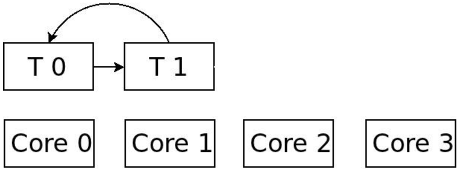
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



Adaptive HELIX: Cost

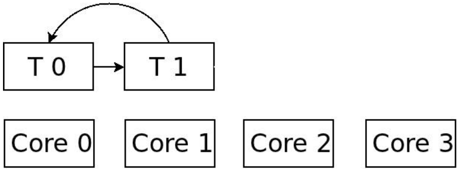
What is the cost of adapting the produced binary?



- Few store instructions

Adaptive HELIX: Cost

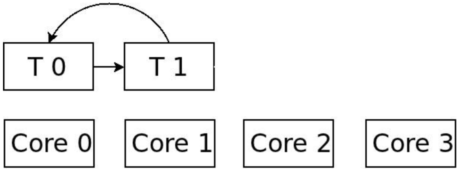
What is the cost of adapting the produced binary?



- Few store instructions
- Thread management

Adaptive HELIX: Cost

What is the cost of adapting the produced binary?



- Few store instructions
- Thread management
  - Thread pool

Adaptive HELIX: Performance

Adaptive HELIX: Performance

- Programs have execution phases

Adaptive HELIX: Performance

- Programs have execution phases
    - Different executed paths

Adaptive HELIX: Performance

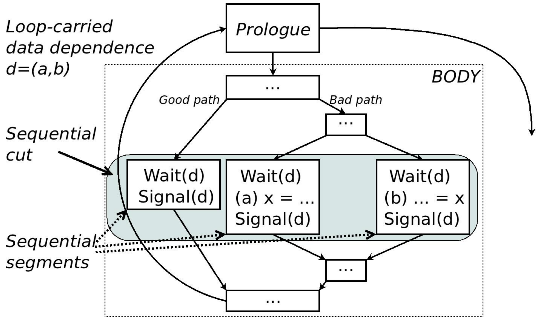
- Programs have execution phases
  - Different executed paths
- The amount of parallelism of a loop changes over time

Adaptive HELIX: Performance

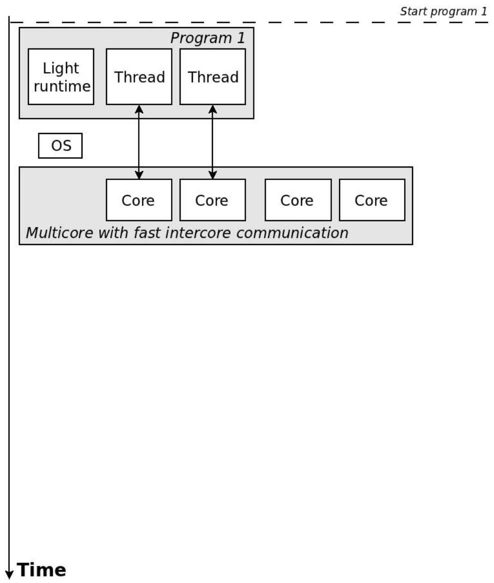
- Programs have execution phases
  - Different executed paths
- The amount of parallelism of a loop changes over time
  - Number of cores to target are adapted at run time

Adaptive HELIX: Performance

- Programs have execution phases
  - Different executed paths
- The amount of parallelism of a loop changes over time
  - Number of cores to target are adapted at run time

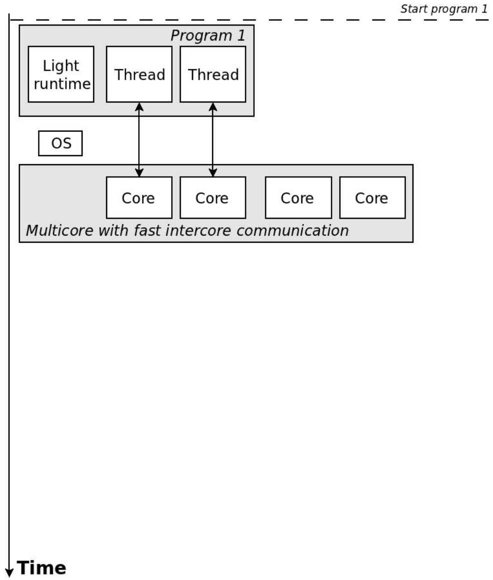

Loop-carried data dependence d=(a,b)
Prologue
BODY
...
Good path
Bad path
...
Sequential cut
Wait(d) Signal(d)
Wait(d) (a) x = ... Signal(d)
Wait(d) (b) ... = x Signal(d)
Sequential segments
...
...

Adaptive HELIX: Performance (2)



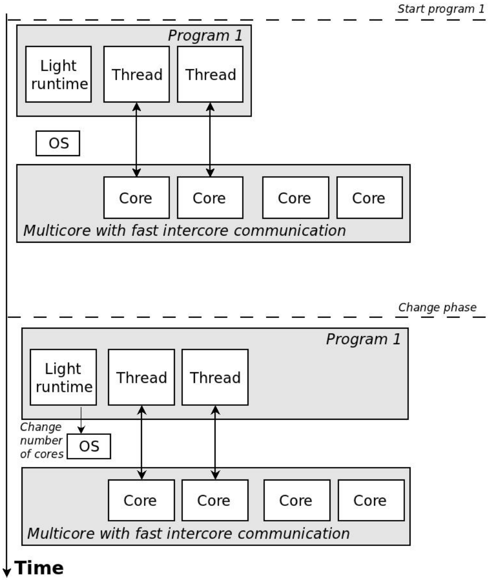
- Program 1 has been parallelized for 2 cores

Adaptive HELIX: Performance (2)



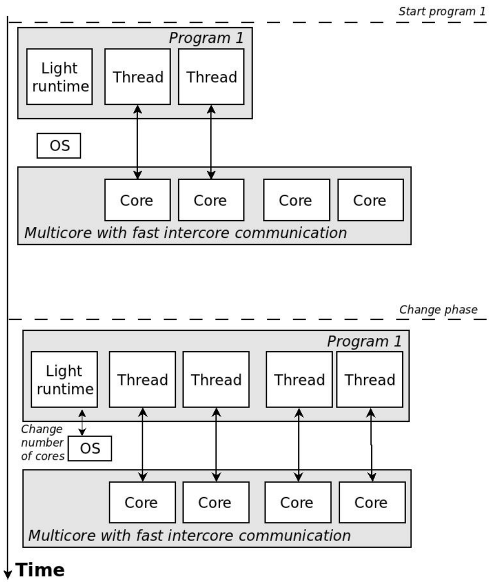
- Program 1 has been parallelized for 2 cores
- The program changes execution phase

Adaptive HELIX: Performance (2)



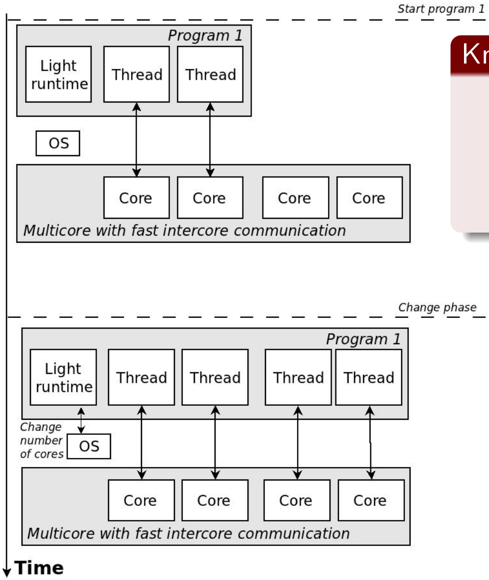
- Program 1 has been parallelized for 2 cores
- The program changes execution phase
- Light runtime starts the interaction with OS

Adaptive HELIX: Performance (2)



- Program 1 has been parallelized for 2 cores
- The program changes execution phase
- Light runtime starts the interaction with OS
- Program 1 increases the cores to 4

Adaptive HELIX: Performance (2)

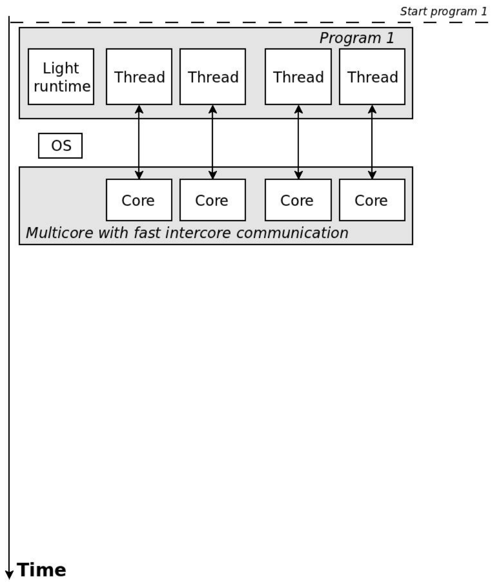


Knowledge

- Program parallelism: *Light runtime*
- Resources available: *OS*

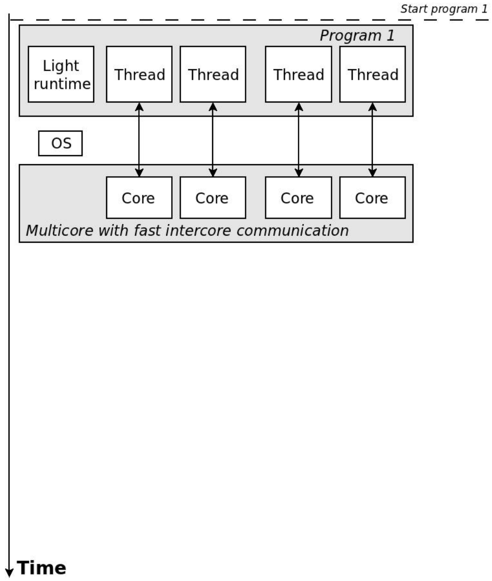
- Program 1 has been parallelized for 2 cores
- The program changes execution phase
- Light runtime starts the interaction with OS
- Program 1 increases the cores to 4

Adaptive HELIX: Multi-programs



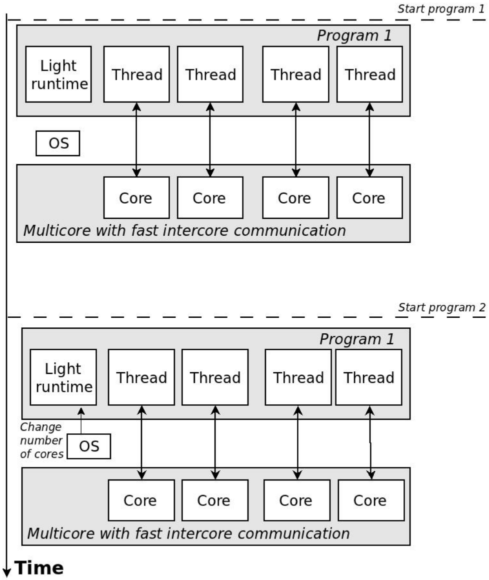
- Program 1 has been parallelized for 4 cores

Adaptive HELIX: Multi-programs



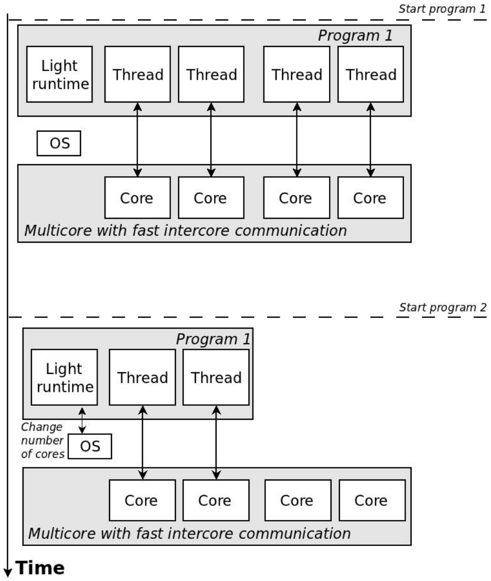
- Program 1 has been parallelized for 4 cores
- Program 2 starts running

Adaptive HELIX: Multi-programs



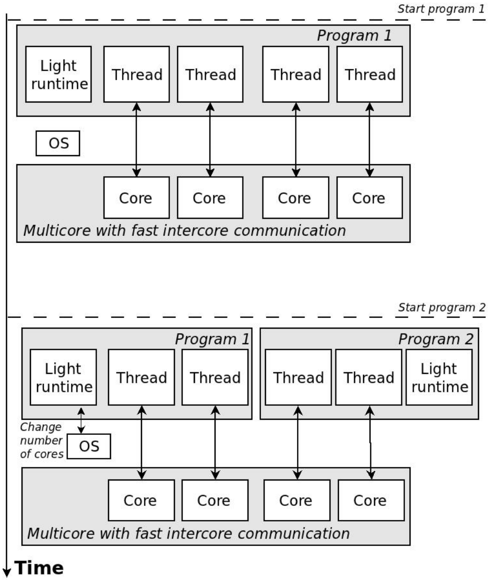
- Program 1 has been parallelized for 4 cores
- Program 2 starts running
- OS starts the interaction with Light runtime

Adaptive HELIX: Multi-programs



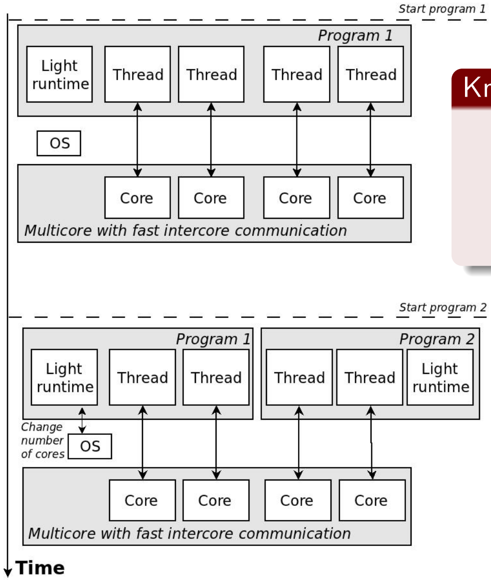
- Program 1 has been parallelized for 4 cores
- Program 2 starts running
- OS starts the interaction with Light runtime
- Program 1 reduces the cores

Adaptive HELIX: Multi-programs



- Program 1 has been parallelized for 4 cores
- Program 2 starts running
- OS starts the interaction with Light runtime
- Program 1 reduces the cores

Adaptive HELIX: Multi-programs



Knowledge

- How to adapt the code: *Light runtime*
- Running programs: *OS*

- Program 1 has been parallelized for 4 cores
- Program 2 starts running
- OS starts the interaction with Light runtime
- Program 1 reduces the cores

Conclusion

HELIX: a new general purpose technique to extract parallelism

Conclusion

HELIX: a new general purpose technique to extract parallelism

- Significant speedups can be achieved on current hardware

Conclusion

HELIX: a new general purpose technique to extract parallelism

- Significant speedups can be achieved on current hardware
  - Hardware not designed for this type of execution
  - Slowdowns are always avoided

Conclusion

HELIX: a new general purpose technique to extract parallelism

- Significant speedups can be achieved on current hardware
  - Hardware not designed for this type of execution
  - Slowdowns are always avoided
- HELIX is able to run both independent and most of dependent code in parallel

Conclusion

HELIX: a new general purpose technique to extract parallelism

- Significant speedups can be achieved on current hardware
    - Hardware not designed for this type of execution
    - Slowdowns are always avoided
- HELIX is able to run both independent and most of dependent code in parallel
- The HELIX code is adapted at run time
    - for performance
    - to handle multiple programs

Light runtime and OS extension is required

Team

References

Websites

- HELIX
  - http://helix.eecs.harvard.edu
  - http://twitter.com/#!/Helix_project
- ILDJIT
  - http://ildjit.sourceforge.net

Email

- xan@eecs.harvard.edu

Thanks for your attention!