The HELIX Project: Overview and Directions

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

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

Making the extraction of thread-level parallelism mainstream
Project Goal

Making the extraction of thread-level parallelism mainstream
Making the extraction of thread-level parallelism mainstream
Project Goal

Making the extraction of thread-level parallelism mainstream

```bash
$ 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

[Diagram showing speedup vs. cores]
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

Instead of

- Software engineer
- Compilers
Project Goal

Making the extraction of thread-level parallelism mainstream

Instead of

- Software engineer
- Compilers
- Computer architecture
Project Goal

Making the extraction of thread-level parallelism mainstream

Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI
Making the extraction of thread-level parallelism mainstream

Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI
Project Goal

Making the extraction of thread-level parallelism mainstream

Instead of

- Software engineer
- Compilers
- Computer architecture
- VLSI
Motivation

Extraction of Thread-Level-Parallelism (TLP)

In multicore era:

Manual approach:

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

DSWP
- Speedup are stable on inter-core communication delay
- Hard to predict speedup
- Hard to avoid slowdown
Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: \( \uparrow \) performance \( \Leftrightarrow \) TLP \( \uparrow \)

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

DSWP
- Speedup are stable on inter-core communication delay
- Hard to predict speedup
- Hard to avoid slowdown
Motivation

Extraction of Thread-Level-Parallelism (TLP)
- In multicore era: ↑ performance ⇔ TLP ↑

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 \( \iff \) TLP \( \uparrow \)

Provide more parallelism

Reduce communication overhead

Communication overhead
Motivation

Extraction of Thread-Level-Parallelism (TLP)
- In multicore era: \( \uparrow \) performance \( \Leftrightarrow \) TLP \( \uparrow \)

Provide more parallelism

Reduce communication overhead

Communication overhead

Communication overhead
Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: \( \uparrow \) performance \( \Leftrightarrow \) TLP \( \uparrow \)

Provide more parallelism

Reduce communication overhead

Communication overhead

Communication overhead
Motivation

Extraction of Thread-Level-Parallelism (TLP)

- In multicore era: \( \uparrow \) performance \( \Leftrightarrow \) TLP \( \uparrow \)

Provide more parallelism

Reduce communication overhead

\[ \text{Parallelism} \Leftrightarrow \text{communication} \]
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
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: ↑ performance ⇔ TLP ↑
- Manual approach: ↑ 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 ≥ 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

Time

Simone Campanoni
HELIX
4/23
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: ↑ performance ⇔ TLP ↑
- Manual approach: ↑ 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 ⇔ 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**

Time

1966

1986

2005
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
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 \( \iff \) 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

- **DSWP**
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
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
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

- **DSWP**
  - Speedup are stable on inter-core communication delay
  - Hard to predict speedup
  - Hard to avoid slowdown
Motivation

Extraction of Thread-Level-Parallelism (TLP)
- In multicore era: \(\uparrow\) performance \(\iff\) 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
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
HELIX

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

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

Idea: exploit independent instructions parallelism among sequential segments

Problem: amount of synchronization required increases drastically!

Simone Campanoni
HELIX 8/23

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

1 → 3  
2 → 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();
}

- Idea: exploit independent instructions and
A Simple Idea

Idea: exploit independent instructions \textit{and} parallelism among sequential segments.

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

Simone Campanoni

HELIX 8/23
for (...)
    1: a = update(a);
    2: work1(a);
    3: b = update(b);
    4: work2();
}

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

Problem: amount of synchronization required increases drastically!
Status of HELIX

Prototype

Static code generation
Number of cores decided at compile time
Challenge: achieve speedup
Constrain communication overhead

[ CGO 2012, IEEE Micro 2012 ]

Hardware support
Push HELIX to the limit
Minor changes to commodity processors
Prototype

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

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

Prototype

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

 Challenge: achieve speedup
Constrain communication overhead

[ CGO 2012, IEEE Micro 2012 ]

Hardware support
Push HELIX to the limit
Minor changes to commodity processors

Simone Campanoni
HELIX 9/23
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
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™ 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™ 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

- 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

- 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

- Adapt code at run time to:
  - Parallel behavior
  - System requirements
  - 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

Adapt code at run time to:

Adaptive HELIX

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 ]

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

Adapt code at run time to:

- Parallel behavior
- System requirements
Summary

- Motivation
- The HELIX Research Project
- HELIX on commodity processors
- Adaptive HELIX
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
  - Automatic selection of loops

**Approach**

- Select loops to parallelize
  - Each loop in the program is analyzed independently
  - These analyses 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

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

<table>
<thead>
<tr>
<th>Signalling</th>
<th>Data forwarding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify threads</td>
<td>Forward data between threads</td>
</tr>
</tbody>
</table>

## 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

<table>
<thead>
<tr>
<th>Signalling</th>
<th>Data forwarding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify threads</td>
<td>Forward data between threads</td>
</tr>
</tbody>
</table>

### 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 ∈ 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 ∈ 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
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
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
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 ∈ static compiler ILDJIT
Overall program speedup
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
Code produced for N cores
The number of cores changes to M at run time
  - Performance
  - Multi-programs scenario
What is the cost of adapting the produced binary?
What is the cost of adapting the produced binary?
What is the cost of adapting the produced binary?
What is the cost of adapting the produced binary?

![Diagram showing T0, T1, T2, T3, Core 0, Core 1, Core 2, Core 3 connections.]

Adaptive HELIX: Cost
Adaptive HELIX: Cost

What is the cost of adapting the produced binary?

T0 → T1 → T2 → T3

Core 0 → Core 1 → Core 2 → Core 3
What is the cost of adapting the produced binary?
What is the cost of adapting the produced binary?

Diagram:

```
T0 ----> T1

Core 0   Core 1   Core 2   Core 3
```
What is the cost of adapting the produced binary?

- Few store instructions
What is the cost of adapting the produced binary?

- Few store instructions
- Thread management
What is the cost of adapting the produced binary?

- Few store instructions
- Thread management
  - Thread pool
Adaptive HELIX: Performance

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
Adaptive HELIX: Performance

- Programs have execution phases
  - Different executed paths
Programs have execution phases
  - Different executed paths
- The amount of parallelism of a loop changes over time
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
Program 1 has been parallelized for 2 cores.
Program 1 has been parallelized for 2 cores
The program changes execution phase
Program 1 has been parallelized for 2 cores
- The program changes execution phase
- Light runtime starts the interaction with 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: 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

Diagram:
- Start program 1
- Light runtime
- Program 1 (Threads): Thread, Thread, Thread
- OS
- Multicore with fast intercore communication
- Time
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

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
HELIX: a new general purpose technique to extract parallelism

- Significant speedups can be achieved on current hardware
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: 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
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

Simone Campanoni
## References

### Websites
- **HELIX**
  - [http://helix.eecs.harvard.edu](http://helix.eecs.harvard.edu)
  - [http://twitter.com/#!/Helix_project](http://twitter.com/#!/Helix_project)
- **ILDJIT**
  - [http://ildjit.sourceforge.net](http://ildjit.sourceforge.net)

### Email
- xan@eecs.harvard.edu
Thanks for your attention!