HELIX: Automatic Parallelization of Irregular Programs for Chip Multiprocessing

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

- Motivation
- A simple idea
- Single loop parallelization
- Loop selection
- Evaluation
- Conclusion
Motivation

Extraction of Thread-Level-Parallelism (TLP)

In multicore era:

\[
\uparrow \quad \text{performance} \quad \uparrow \quad \text{TLP}
\]

Manual approach:

\[
\uparrow \quad \text{software development time}
\]

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

Simone Campanoni
HELIX 3/26
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 \(\uparrow\)
- 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

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**
  - 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

Simone Campanoni | HELIX | 3/26
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
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**

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

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
- 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:

<table>
<thead>
<tr>
<th>Year</th>
<th>Approach</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>DOALL</td>
<td>Speedup increases with number of cores, Limited applicability, Loop-carried dependences not handled</td>
</tr>
<tr>
<td>1986</td>
<td>DOACROSS</td>
<td>Applicable to a broader set of programs, Extremely sensitive to inter-core communication</td>
</tr>
<tr>
<td>2005</td>
<td>DSWP</td>
<td>Hard to predict speedup, Hard to avoid slowdown</td>
</tr>
</tbody>
</table>
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
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

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:

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
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
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 \( \prec \) Stability of speedup \( \prec \) DSWP

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

### 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
- Inter-core communication \( \Rightarrow \)
- Produce speedup predictable enough 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:

**HELIX**

- Speedup increases with number of cores
- General purpose technique

**DOACROSS** < Stability of speedup < DSWP

- Inter-core communication \( \Rightarrow \) private cache access hit
- Produce speedup predictable enough to avoid slowdown
Motivation (2)

HELIX

- General purpose technique
- Predictable speedup
  - Avoid slowdown
- $|\text{threads}| \leq |\text{loop iterations}|$
Motivation (2)

HELIX

- General purpose technique
- Predictable speedup
  - Avoid slowdown
- $|\text{threads}| \leq |\text{loop iterations}|$
- TLP extracted between loop iterations
- Iterations grouped on modular value
Motivation (2)

**HELIX**
- General purpose technique
- Predictable speedup
  - Avoid slowdown
- $|\text{threads}| \leq |\text{loop iterations}|$
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
- Automatic selection of loops

Diagram:
- Loop iterations: 0, 1, 2, 3, 4, 5, 6, ...
- Time: Core0, Core1, Core2, Core3, Core4...
- Tasks assigned to cores at different times.
Motivation (2)

HELIX

- General purpose technique
- Predictable speedup
  - Avoid slowdown
- $|\text{threads}| \leq |\text{loop iterations}|$
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
- Automatic selection of loops
- Easy to implement
Motivation (2)

HELIX

- General purpose technique
- Predictable speedup
  - Avoid slowdown
- $|\text{threads}| \leq |\text{loop iterations}|$
  - TLP extracted between loop iterations
  - Iterations grouped on modular value
- Automatic selection of loops
- *Easy* to implement
Summary

- Motivation
- A simple idea
- Single loop parallelization
- Loop selection
- Evaluation
- Conclusion
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();
}

- **Idea:** exploit independent instructions

Diagram:

- Intra iteration data dependences
- Loop-carried data dependences
A Simple Idea

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

- Idea: exploit independent instructions *and*

![Diagram showing data dependencies and loop-carried data dependencies]
A Simple Idea

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

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

Idea: exploit independent instructions and parallelism among sequential segments

Problem: amount of synchronization required increases drastically!

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

![Diagram showing parallel execution on two cores]

- Idea: exploit independent instructions \textit{and} parallelism among sequential segments
- Problem: amount of synchronization required increases drastically!
<table>
<thead>
<tr>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signalling</strong></td>
</tr>
<tr>
<td>Notify threads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Adopted solutions</strong></td>
</tr>
</tbody>
</table>
Overhead

**Signalling**
Notify threads

**Optimizations**

**Adopted solutions**
- New code analysis to reduce the number of signals to send
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
<table>
<thead>
<tr>
<th>HELIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead</td>
</tr>
<tr>
<td>Signalling</td>
</tr>
<tr>
<td>Notify threads</td>
</tr>
<tr>
<td>Sequential code</td>
</tr>
<tr>
<td>Code that must execute in loop-iteration order</td>
</tr>
</tbody>
</table>

<p>| Optimizations |</p>
<table>
<thead>
<tr>
<th>Adopted solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>New code analysis to reduce the number of signals to send</td>
</tr>
<tr>
<td>Code scheduling and use of SMT to reduce the delay per signal</td>
</tr>
</tbody>
</table>
## Overhead

### Signalling
- Notify threads

### Sequential code
- Code that must execute in loop-iteration order

## 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
- Code scheduling
Overhead

Signalling
Notify threads

Sequential code
Code that must execute in loop-iteration order

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
- Code scheduling
- Execution of $\neq$ segments in parallel
Overhead

Signalling
Notify threads

Sequential code
Code that must execute in loop-iteration order

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
- Code scheduling
- Execution of $\neq$ segments in parallel
### Overhead
- **Signalling**
  - Notify threads

- **Sequential code**
  - Code that must execute in loop-iteration order

- **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
- Code scheduling
- Execution of ≠ segments in parallel
- Automatic selection of loops
### Overhead

<table>
<thead>
<tr>
<th>Signalling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify threads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequential code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code that must execute in loop-iteration order</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data forwarding</th>
</tr>
</thead>
<tbody>
<tr>
<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
- Code scheduling
- Execution of \( \neq \) segments in parallel
- Automatic selection of loops

### Approach

- Select loops to parallelize
Overhead

Signalling
Notify threads

Sequential code
Code that must execute in loop-iteration order

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
- Code scheduling
- Execution of $\neq$ segments in parallel
- Automatic selection of loops

Approach
- Select loops to parallelize
  - Light profile based selection
**Overhead**

**Signalling**
- Notify threads

**Sequential code**
- Code that must execute in loop-iteration order

**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
- Code scheduling
- Execution of $\neq$ segments in parallel
- Automatic selection of loops

**Approach**
- Select loops to parallelize
  - Light profile based selection
- Parallelize one loop at a time
Overhead

Signalling
- Notify threads

Sequential code
- Code that must execute in loop-iteration order

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
- Code scheduling
- Execution of $\neq$ segments in parallel
- Automatic selection of loops

Approach
- Select loops to parallelize
  - Light profile based selection
- Parallelize one loop at a time
  - Each loop uses all cores decided at compile time
Summary

Motivation
A simple idea
Single loop parallelization
Loop selection
Evaluation
Conclusion
Step 1: Normalizing the Loop

Reason: prologue is executed in loop-iteration order

Best case: single exit controlled by an induction variable
Step 1: Normalizing the Loop

The code is scheduled to minimize time spent in the prologue. Reason: prologue is executed in loop-iteration order.

Best case: single exit controlled by an induction variable.
Step 1: Normalizing the Loop

The code is scheduled to minimize time spent in the loop.

Reason: prologue is executed in loop-iteration order.

Best case: single exit controlled by an induction variable.
Step 1: Normalizing the Loop

- The code is scheduled to minimize time spent ∈ prologue
  - Reason: prologue is executed in loop-iteration order
  - Best case: single exit controlled by an induction variable
Step 2: Identifying data dependences to satisfy

- The code is scheduled to minimize time spent ∈ prologue
  - Reason: prologue is executed in loop-iteration order
- Best case: single exit controlled by an induction variable
The code is scheduled to minimize time spent ∈ prologue
- Reason: prologue is executed in loop-iteration order
- Best case: single exit controlled by an induction variable
Step 3: Starting next iterations

- The code is scheduled to minimize time spent ∈ prologue
  - Reason: prologue is executed in loop-iteration order
- Best case: single exit controlled by an induction variable
Step 4: Computing Sequential Segments
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{Data}$:

- 
- 
-
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{Data}$:

- Instructions $Wait(d)$ are inserted as late as possible
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{Data}$:

- Instructions $Wait(d)$ are inserted as late as possible
- Instructions $Signal(d)$ are inserted as early as possible
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{\text{Data}}$:

- Instructions $\text{Wait}(d)$ are inserted as late as possible
- Instructions $\text{Signal}(d)$ are inserted as early as possible
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{Data}$:

- Instructions $Wait(d)$ are inserted as late as possible
- Instructions $Signal(d)$ are inserted as early as possible
Step 4: Computing Sequential Segments

For every $d = (a, b) \in D_{Data}$:

- Instructions $Wait(d)$ are inserted as late as possible
- Instructions $Signal(d)$ are inserted as early as possible
Step 5: Minimizing Sequential Segments

Method inlining and code scheduling applied.

Loop-carried
data dependence
\(d=(a,b)\)

Sequential
segments

Prologue

Wait(d)
Signal(d)
(a) \(x = \ldots\)  
Signal(d)
(b) \(\ldots = x\)  
Signal(d)

BODY

Sequential segment 1

Sequential segment 2

Sequential segment 3

Core 0

Core 1

Time

TLP among segments
Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals

80% – 98% of signals sent removed
Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences

Theorem

Let $G = (N, E)$ be a data dependence redundancy graph and let $N_{to-synch} \subseteq N$ be the set of dependences that includes every node without incoming edges and one node per cycle of $G$. Synchronizing the set $N_{to-synch}$ synchronizes the entire set of dependences $N$. 
Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences

\[(a, b) \rightarrow (c, d)\]
Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences

(a,b) → (c,d) → ... → ... → ...

Fully connected component
Steps 6 and 7

Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences

(a,b) → (c,d) → ...

Simone Campanoni
HELIX 13/26
Steps 6 and 7

Step 6: Minimizing Signals

New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences

(a,b) → (c,d) → ...

Simone Campanoni
HELIX 13/26
**Step 6: Minimizing Signals**

New analysis developed to minimize redundancy of signals

- intra- and inter-data dependences
- 80% – 98% of signals sent removed
Steps 6 and 7

Step 6: Minimizing Signals
New analysis developed to minimize redundancy of signals
- intra- and inter-data dependences
- 80% – 98% of signals sent removed

Step 7: Inserting Inter-Thread Communication
New analysis to minimize loads and stores of shared locations
Step 8: Coupling with Helper Threads

Cache memories are pull systems

Solution: couple helper threads for signal prefetching

Observation: sequence of sequential segments predictable
Step 8: Coupling with Helper Threads

Cache memories are pull systems. Solution: couple helper threads for signal prefetching.

Observation: sequence of sequential segments predictable.
Step 8: Coupling with Helper Threads

Cache memories are pull systems

No prefetching
Core 0
SS 1
A
SS 2
B
SS 3
Core 1
C

Observation: sequence of sequential segments predictable

Solution: couple helper threads for signal prefetching
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching

Observation: sequence of sequential segments predictable
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
- Observation: sequence of sequential segments predictable
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
- Observation: sequence of sequential segments predictable

![Diagram showing no prefetching vs. prefetching with and without balancing](image-url)
Step 8: Coupling with Helper Threads

- Cache memories are pull systems
- Solution: couple helper threads for signal prefetching
- Observation: sequence of sequential segments predictable
Summary

Motivation
A simple idea
Single loop parallelization
Loop selection
Evaluation
Conclusion
Loop Selection

HELIX approach
 HELIX approach

- Each loop in the program is analyzed independently.
Loop Selection

HELIX approach

- Each loop $\in$ program is analyzed independently
- The program is analyzed to identify the most profitable loops
Single Loop Analysis

Assumption

Time spent to send a signal is always \(\in\) critical path constant.

Speedup = \(\frac{Seq}{Seq + Par}\)  

\(\text{Overhead} \approx \text{Sig} \times S + \left\lceil \frac{\text{Bytes}}{\text{CPU word}} \right\rceil \times M\)

Thanks to characteristic of the produced code:

\(\text{Sig} = |\text{loop iterations}| \times |\text{sequential segments}|\)
Single Loop Analysis

Assumption
- Time spent to send a signal is *always* ∈ critical path
Assumption

- Time spent to send a signal is
  - always $\in$ critical path
  - constant
Assumption

- Time spent to send a signal is
  - always ∈ critical path
  - constant

\[
\text{Speedup} = \frac{\text{Seq} + \frac{\text{Par}}{N}}{\text{Seq} + \frac{\text{Par}}{N} + O}
\]
**Assumption**

- Time spent to send a signal is
  - always $\in$ critical path
  - constant

$$\text{Speedup} = \frac{\text{Seq} + \text{Par}}{\text{Seq} + \frac{\text{Par}}{N} + O}$$

**Overhead**

$$O \approx \text{Sig} \times S + \left\lceil \frac{\text{Bytes}}{\text{CPU}_{\text{word}}} \right\rceil \times M$$
Single Loop Analysis

**Assumption**

- Time spent to send a signal is
  - always ∈ critical path
  - constant

\[
\text{Speedup} = \frac{\text{Seq} + \frac{\text{Par}}{N}}{\text{Seq} + \frac{\text{Par}}{N} + O}
\]

where

**Overhead**

\[
O \approx \text{Sig} \times S + \left[ \frac{\text{Bytes}}{\text{CPU}_{\text{word}}} \right] \times M
\]

Thanks to characteristic of the produced code:

\[
\text{Sig} = |\text{loop iterations}| \times |\text{sequential segments}|
\]
Identify loops to parallelize

Propagate parallel code information

Loop1
- $T = 0.5$
- $\text{max}T = 0.5$

Loop2
- $T = 0.3$
- $\text{max}T = 0.3$

Loop3
- $T = 0.4$
- $\text{max}T = 0.4$

Loop4
- $T = 0.4$
- $\text{max}T = 0.4$
Identify loops to parallelize

Propagate parallel code information

- **Loop1**: \( T = 0.5 \), \( \text{maxT} = 0.8 \)
- **Loop2**: \( T = 0.3 \), \( \text{maxT} = 0.8 \)
- **Loop3**: \( T = 0.4 \), \( \text{maxT} = 0.4 \)
- **Loop4**: \( T = 0.4 \), \( \text{maxT} = 0.4 \)
Identify loops to parallelize

Propagate parallel code information

Notice: only max parallel is propagated
Identify loops to parallelize

Exploit parallel code information

Notice: only max parallel is propagated
Identify loops to parallelize

Exploit parallel code information

Notice: only max parallel is propagated
Identify loops to parallelize

Exploit parallel code information

Notice: only max parallel is propagated

Is this an heuristic?
Loop Selection for 179.art

Nesting level

1

2

match

3

T = 1
maxT = 121000

L3

T = 2
maxT = 120000

L6

T = 70000
maxT = 70000

L8

T = 5000
maxT = 5000

L10

T = 2000
maxT = 5000

L9

T = 50000
maxT = 50000

L11

scan_recognize

T = 21000
maxT = 123000

L1

main

T = 500
maxT = 2000

L2

T = 1000
maxT = 1000

L4

T = 1000
maxT = 1000

L5

reset_nodes

Search in a deeper level
Selected loops
Loops not reached
Motivation
A simple idea
Single loop parallelization
Loop selection
Evaluation
Conclusion
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
## 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
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 ∈ static compiler ILDJIT
- C benchmarks are first translated to CIL bytecode by GCC4CLI
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 ∈ static compiler ILDJIT
- C benchmarks are first translated to CIL bytecode by GCC4CLI

Evaluation
- The input \textit{train} is used to select loops
- The input \textit{ref} is used to compute the speedups
Speedup Obtained on a Real System

Overall program speedup
Overall program speedup
Speedup Obtained on a Real System

Overall program speedup

Most significant contributions
Speedup Obtained on a Real System

Overall program speedup

Most significant contributions
Speedup Obtained on a Real System

Overall program speedup

Most significant contributions

Notice: no slowdown
Chosen Loops

Most of the time is spent inside parallel code.
Chosen Loops

Most of the time is spent inside parallel code
Most of the time is spent inside parallel code
Chosen Loops

Most of the time is spent inside parallel code

Loops $\in$ single nesting level is a poor solution
Chosen Loops

Most of the time is spent inside parallel code

Loops ∈ single nesting level is a poor solution
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
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
- 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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
  - Successfully identify the most profitable loops
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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
  - Successfully identify the most profitable loops
  - Avoid slowdowns
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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
  - Successfully identify the most profitable loops
  - Avoid slowdowns
  - Reduce delay per signal
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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
  - Successfully identify the most profitable loops
  - Avoid slowdowns
  - Reduce delay per signal
- How the hardware can be designed to improve HELIX code?
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
- HELIX is able to run both independent and most of dependent code in parallel
- Thanks to the code predictability, HELIX is able to
  - Successfully identify the most profitable loops
  - Avoid slowdowns
  - Reduce delay per signal
- How the hardware can be designed to improve HELIX code?
- What are the limits of HELIX?
References

Websites

- HELIX
  - http://helix.eecs.harvard.edu
- ILDJIT

Email

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