An example of a research compiler
Sequential programs are not accelerating like they used to
Multicores are underutilized

Single application:
Not enough explicit parallelism
  • Developing parallel code is hard
  • Sequentially-designed code is still ubiquitous

Multiple applications:
Only a few CPU-intensive applications running concurrently in client devices
Parallelizing compiler:
Exploit unused cores to accelerate sequential programs
Non-numerical programs need to be parallelized.
Parallelize loops to parallelize a program

99% of time is spent in loops

Outermost loops
DOACROSS parallelism

Iteration 0

Iteration 1

Iteration 2

work()

work()

work()
DOACROSS parallelism

Sequential segment

Parallel segment

\[
c = f(c) \\
d = f(d) \\
\text{work()}
\]
HELIX: DOACROSS for multicore

HELIX: DOACROSS for multicore

HELIX: DOACROSS for multicore

HELIX: DOACROSS for multicore

HELIX: DOACROSS for multicore

Parallelize loops to parallelize a program

99% of time is spent in loops

Outermost loops

Innermost loops

Time
Parallelize loops to parallelize a program

Innermost loops

Outermost loops

Coverage

Ease of analysis

Communication

HELIX
HELIX: DOACROSS for multicore

Outline

Small Loop Parallelism and HELIX

[CGO 2012
DAC 2012,
IEEE Micro 2012]

HELIX-RC: Architecture/Compiler Co-Design

[ISCA 2014]

HELIX-UP: Unleash Parallelization

[CGO 2015]
SLP challenge: short loop iterations

Duration of loop iteration (cycles)

SPEC CPU
Int benchmarks
SLP challenge: short loop iterations

Percentage of loop iterations vs. Duration of loop iteration (cycles)

SPEC CPU
Int benchmarks
SLP challenge: short loop iterations

Percentage of loop iterations

Duration of loop iteration (cycles)

Adjacent core communication latency

Nehalem

Ivy Bridge

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

**Compiler**

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

**Architecture: Ring Cache**

- Reduce the communication latency on the critical path
Light-weight enhancement of today’s multicore architecture

```
Store X, 1
Store Y, 1
Iter. 0

Ring node

Core 0

DL1

Ring node

Core 1

DL1

Ring node

Last level cache

75 – 260 cycles!

Iter. 1

Load X

Load Y

Iter. 2

Core 2

Iter. 3

Core 3

Store X, 1

Store Y, 1
```
Light-weight enhancement of today’s multicore architecture
Simulator: XIOSim, DRAMSim
Compiler: ILDJIT (LLVM)

Latency: 1 cycle
Bandwidth: 70 bits for signals
          68 bits for data

98% hit rate
The importance of HELIX-RC
The importance of HELIX-RC

Non-numerical programs

Numerical programs

Program speedup

HELIX
HELIX-RC

164.gzip 175.vpr 197.parser 300.twolf 181.mcf 256.bzip2 INT Geomean 183.equake 179.art 188.ammp 177.mesa FP Geomean Geomean
Outline

Small Loop Parallelism and HELIX

HELIX-RC: Architecture/Compiler Co-Design
[ISCA 2014]

Small loops

HELIX

Communication

HELIX-UP: Unleash Parallelization
[CGO 2015]

Quality: 100% 96% 90% 84%
Opportunity:
relax program semantics

• Some workloads tolerate output distortion

• Output distortion is workload-dependent
Relaxing transformations remove performance bottlenecks

• Sequential bottleneck

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst 1</td>
<td>Inst 1</td>
<td>Inst 1</td>
</tr>
<tr>
<td>Inst 2</td>
<td>Inst 2</td>
<td>Inst 2</td>
</tr>
<tr>
<td>Inst 3</td>
<td>Inst 3</td>
<td>Inst 3</td>
</tr>
<tr>
<td>Inst 4</td>
<td>Inst 4</td>
<td>Inst 4</td>
</tr>
</tbody>
</table>

Sequential segment

Speedup
Relaxing transformations remove performance bottlenecks

• Sequential bottleneck

• Communication bottleneck

• Data locality bottleneck
Relaxing transformations remove performance bottlenecks

No relaxing transformations
Relaxing transformation 1
Relaxing transformation 2
...
Relaxing transformation k

No output distortion
Baseline performance
Max output distortion
Max performance
Design space of HELIX-UP

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

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

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

Complete space = $2^{100}$
Pruned space = $50 \times (2^2) = 200$

How well does HELIX-UP perform?
HELIX: no relaxing transformations with small output distortions

HELIX-UP unblocks extra parallelism

Nehalem 6 cores
2 threads per core
HELIX-UP unblocks extra parallelism with small output distortions

Nehalem 6 cores
2 threads per core
Performance/distortion tradeoff

256.bzip2

HELIX

%Output Distortion

Normalized Performance

Static HELIX-UP

36
Run time code tuning

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

• The runtime reacts to transient bottlenecks by adjusting code accordingly
Adapting code at run time unlocks more parallelism

256.bzip2

HELIX

Normalized Performance

% Output Distortion

Static HELIX-UP
HELIX-UP improves more than just performance

- Robustness to DDG inaccuracies
- Consistent performance across platforms
Relaxed transformations to be robust to DDG inaccuracies

Increasing DDG inaccuracies leads to lower performance

No impact on HELIX-UP
Relaxed transformations for consistent performance
Small Loop Parallelism and HELIX

- *Parallelism hides in small loops*

HELIX-RC: Architecture/Compiler Co-Design

- *Irregular programs require low latency*

HELIX-UP: Unleash Parallelization

- *Tolerating distortions boosts parallelization*
Thank you!
Small Loop Parallelism and HELIX

• *Parallelism hides in small loops*

HELIX-RC: Architecture/Compiler Co-Design

• *Irregular programs require low latency*

HELIX-UP: Unleash Parallelization

• *Tolerating distortions boosts parallelization*