An example of a research compiler

Simone Campanoni
simonec@eecs.northwestern.edu
Sequential programs are not accelerating like they used to.

Core frequency scaling

Performance gap

Multicore era

Sequential program running on a platform

1992

2004
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
DOALL parallelism

Iteration 0

Iteration 1

Iteration 2

work()
DOACROSS parallelism

Sequential segment

Parallel segment
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

Time

Outermost loops

Innermost loops
Parallelize loops to parallelize a program

- Innermost loops
- Outermost loops

- Coverage
- Ease of analysis
- Communication
HELIX: DOACROSS for multicore


Speedup

SPEC INT baseline

ICC, Microsoft Visual Studio, DOACROSS

HELIX

4-core Intel Nehalem

Coverage

Easy of analysis

Loop Parallelism

Communication

VENN

Innermost loops

Outermost loops

HELIX-RC
HELIX-UP

Small Loop Parallelism
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

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

SPEC CPU Int benchmarks
SLP challenge: short loop iterations

Duration of loop iteration (cycles)

Percentage of loop iterations

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

Core 0

Core 1

Load X
Load Y

... Iter. 1

Ring node

DL1

Last level cache

Ring node

Core 3

Core 2

Ring node

75 – 260 cycles!

Iter. 3

Iter. 2
Light-weight enhancement of today’s multicore architecture

Store X, 1
Wait 0
Store Y, 1
Signal 0

Iter. 0

Core 0

Core 1

Wait 0
Load Y

Iter. 1

Iter. 0
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

Program speedup

Non-numerical programs

Numerical programs

HELIX
HELIX-RC

164.gzip 175.vpr 197.parser 300.twolf 181.mcf 256.bzip2 INT Geomean 189.equake 179.art 188.ammp 177.mesa FP Geomean Geomean
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]
Opportunity: relax program semantics

• Some workloads tolerate output distortion

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

- Sequential bottleneck
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

![Graph showing performance vs. distortion for HELIX and 256.bzip2]
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

% Output Distortion

Normalized Performance

256.bzip2

HELIX

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

![Graph showing normalized performance with increasing communication latency. The graph compares HELIX on Nehalem, HELIX on Bulldozer, HELIX on Haswell, UP on Nehalem, UP on Bulldozer, and UP on Haswell. The output distortion varies from 0.7% to 1%.](image)
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*