Meet the Walkers
Accelerating Index Traversals for In-Memory Databases

Onur Kocberber
Boris Grot, Javier Picorel, Babak Falsafi,
Kevin Lim, Parthasarathy Ranganathan
Our World is Data-Driven!

Data resides in huge databases
   – Most frequent task: find data

Indexes used for fast data lookup
   – Rely on pointer-intensive data structures

Indexing efficiency is critical
   – Many requests, abundant parallelism
   – Power-limited hardware

Need high-throughput and energy-efficient index lookups
Index Lookups on General-Purpose Cores

Index Lookups
• Data in memory
• Inherent parallelism

OoO Cores
• Pointer-chasing → Low MLP
• Limited OoO inst. window
  – One lookup at a time

OoO cores are ill-matched to indexing
Widx: an Indexing Widget

Specialized: Custom HW for index lookups
   – Switch fewer transistors per lookup

Parallel: Multiple lookups at once
   – Extract parallelism beyond the OoO exec. window

Programmable: Simple RISC cores
   – Target a wide range of DBMSs

3x higher throughput, 5.5x energy reduction vs. OoO
Outline

• Introduction
• Indexing in database systems
• Indexing inefficiencies in modern processors
• Widx
• Evaluation highlights
• Summary
Modern Databases & Indexing

Indexes are essential for all database operators
– Data structures for fast data lookup

Hash index: fundamental index structure

Dominant operation: join via hash index
Join via Hash Index

Join up to the index for every value in A and B

A

B

Hash Index on B

Join via Hash Index
How Much Time is Spent Indexing?
Measurement on Xeon 5670 CPU with 100GB Dataset

Indexing is the biggest contributor to execution time
Dissecting Index Lookups

**Hash:** Avg. 30% time of each lookup
- Computationally intensive, high cache locality

**Walk:** Avg. 70% time of each lookup
- Trivial computation, low cache locality

Next lookup: Inherently parallel
- Beyond the inst. window capacity
Outline

- Introduction
- Indexing in Database Systems
- Indexing inefficiencies in modern processors
- Widx
- Evaluation highlights
- Summary
Roadmap for Efficient and High-Throughput Indexing

1. Specialize
   – Customize hardware for hashing and walking

2. Parallelize
   – Perform multiple index lookups at a time

3. Generalize
   – Use a programmable building block
Step 1: Specialize

Design a dedicated unit for hash and walk
- **Hash**: compute hash values from a key list
- **Walk**: access the hash index and follow pointers
Step 2: Parallelize

- Serial
- Decoupled
- Decoupled & Parallel
Step 3: Generalize

Widx unit: common building block for hash and walk
  – Two-stage RISC core
  – Custom ISA

Widx units are programmable
  – Execute functions written in Widx ISA
  – Support limitless number of data structure layouts
Putting it all together: Widx

When Widx runs, core goes idle

Simple, parallel hardware
Programming Model

Development
Write code for each unit and compile for Widx ISA

Execution
Communicate query-specific inputs

DBMS
Index
Code

Hash
hash (arg1, arg2, …)
{…….}

Walk
walk (arg1, arg2, …)
{…….}

Res. Produce
emit (arg1, arg2, …)
{…….}

Load the code
Outline

• Introduction
• Indexing in Database Systems
• Indexing inefficiencies in modern processors
• Widx
• Evaluation highlights
• Summary
Methodology

Flexus simulation infrastructure [Wenisch '06]

**Benchmarks**
- TPC-H on MonetDB
- TPC-DS on MonetDB
- Dataset: 100GB

**uArch Parameters**
- Core Types
  - OoO: 4-wide, 128-entry ROB
  - In-order: 2-wide
- Frequency: 2GHz
- L1 (I & D): 32KB
- LLC: 4MB

**Area and Power**
- Synopsys Design Compiler
- Technology node: TSMC 40 nm, std. cell
- Frequency: 2GHz
- Widx Area: 0.24mm²
- Widx Power: 0.3W
Widx Performance

3x higher indexing throughput
Widx Efficiency

5.5x reduction in indexing energy vs. OoO
Conclusions

Indexing is essential in modern DBMSs

Modern CPUs spend significant time in index lookups
   – Not efficient & fall short of extracting parallelism

Widx: Specialized widget for index lookups
   – Efficient, parallel & programmable

3x higher throughput, 5.5x energy reduction vs. OoO
Thanks!