Rethinking Databases Design with Smart Memory Systems
Databases and Memory Systems

**On-disk** (1960s –)
- Complex buffer management to reduce disk access
- Finer grain concurrency control to hide I/O overhead
- B-Tree index with (possibly) duplicate data
- ...

**In-memory** (2000s -)
- No buffer – simpler design and more efficient
- Coarser locks or no locks due to higher processing speed
- Memory friendly index, or even no index
- ...

**Smart memory** (now –)
- Smart memories can further change how databases are built
  - HICAMP memory
  - Transactional memory
  - Non-volatile memory
  - Shiftable memory (HP Labs)
  - ...

...
Two Case Studies on HICAMP Memory

- Enable efficient *update* and *scan* on compressed bitmap index

- Hardware-assisted multi-version concurrency control
Efficient Update on Bitmap Index

• Advantages of bitmap index
  – High query performance
  – Space efficient after compression
  – Simpler locking scheme compared to B-Tree

• Commonly used in OLAP, can bitmap be used for OLTP as well?
  – Problem: *updating compressed bitmap is inefficient*
    • decompress – update – recompress
    • variable length due to run-length encoding
  – Solution: *HICAMP memory enables efficient updates on bitmap with compression*
HICAMP Deduplication and DAG Structure

- HICAMP memory manages each object as a DAG
- The same content is stored only once
- Duplicate data are referenced to the same physical location by pointers
- Zero lines are referred by zero pointers

Store two integer arrays $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 5, 6, 7, 8\}$ and $\{9, 10, 11, 12, 0, 0, 0, 0, 0, 0, 0, 13, 14, 15, 16\}$ in HICAMP
Efficient Bitmap Update on HICAMP

• Store as *raw* format
  – Efficient update on bitmap without de-/recompression

• Hardware compression
  – Identical bit sequences are stored only once
  – Zero sequences are simply referred by zero pointers

• Efficient lookup with hardware support
  – Scan for next non-zero bit is $O(1)$
    1. Find next non-zero leaf in the DAG
    2. Find the first non-zero bit in the leaf
Multi-Version Concurrency Control

• Typically, MVCC is implemented by saving multiple versions of a data item with timestamps
• Timestamp checking is required when serving requests
• Pros
  – Never aborts read-only transactions
  – Allows higher concurrency
• Cons
  – Checking timestamps hurts performance
  – Space overhead for saving timestamps per version
  – Headache of garbage collection
Lightweight MVCC on HICAMP

- HICAMP copy-on-write
  - Writes are not executed in-place
  - Instead, a new copy is created
- Each transaction generates a new snapshot at low cost
- Old versions are automatically released once the reference counts reach zero
- Optimistic concurrency control with multiversion serial validation

Change array \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 5, 6, 7, 8\} to \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\} in HICAMP
Conclusions

• Novel memory systems make previous “bad solutions” applicable.

• Smart memories are taking over certain functionalities. Databases should take full advantage of them.

• Hardware innovation is the place where new database research opportunities are generated.