Transactional Memory

Concurrency unlocked Programming

Outline

- Background
- Motivation
- Database Transaction
- Transactional Memory History
- Transactional Memory
  - **Example**
  - **Mechanisms**
- Software Transactional Memory
- Hardware Transactional Memory
- Application

Background

- Around 2004, 50 years of exponential improvement in the performance of sequential computers end, due to **power and cooling concerns**
- In the terminology of Intel's founder, Andrew Grove, this is an inflection point—a "**time in the life of a business when its fundamentals are about to change**"
- **Multiprocessor**

Motivation

- Difficulty of Multiprocessor Programming: **Parallelism** and **Nondeterminacy**
  - **Prune nondeterminism**
  - **Violation(atomicity-violation, order-violation)**
  - **Deadlock**
- Parallelism programming (Multi-thread programming) lacks comparable **abstraction mechanisms**.

Database Transaction

- A **transaction** specifies a program semantics in which a computation **executes as if it was the only computation accessing the database**. Other computations may execute simultaneously, but the model **restricts the allowable interactions among the transactions**.
- Offer an **abstraction mechanism** in database systems for reusable parallel computations.
- **ACID** (Atomic, Consistent, Isolated and Durable)

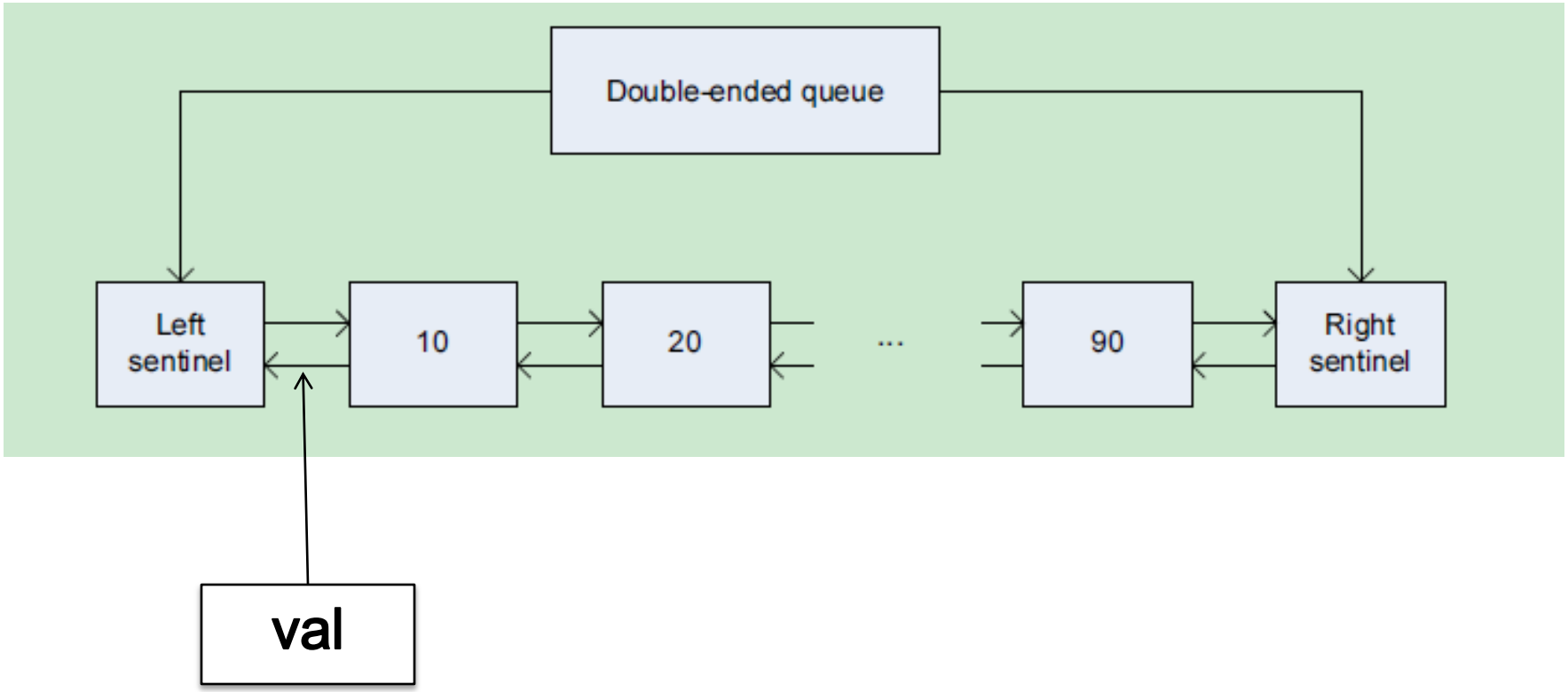
Transactional Memory History

- In 1977, Lomet proposed the idea to **map database transaction into programming language**, but no implementation
- In 1993, Herlihy and Moss proposed **hardware supported transactional memory**
- In 1993, Stone et al. proposed an atomic multi-word operation known as **"Oklahoma Update"**
- In recent years, a huge ground swell of interest in both hardware and software systems for implementing transactional memory.

Transactional Memory

- **"Steal" ideas from database transaction**. (A transaction is a sequence of actions that appears **indivisible** and **instantaneous** to an outside observer.)
- The properties of transactions provide a convenient abstraction for **coordinating concurrent reads and writes of shared data** in a concurrent or parallel system.

Example


Double-ended queue
Left sentinel
10
20
...
90
Right sentinel
val

Example

```
void PushLeft(DQueue *q, int val) {
  QNode *qn = malloc(sizeof(QNode));
  qn->val = val;
  do {
    StartTx();
    QNode *leftSentinel = ReadTx(&(q->left));
    QNode *oldLeftNode = ReadTx(&(leftSentinel->right));
    WriteTx(&(qn->left), leftSentinel);
    WriteTx(&(qn->right), oldLeftNode);
    WriteTx(&(leftSentinel->right), qn);
    WriteTx(&(oldLeftNode->left), qn);
  } while (!CommitTx());
}
```

Mechanisms of Transactional Memory

- Manage the **tentative work** that a transaction does while it executes, akin the `left` and `right` fields of the `qn` object.
- Ensure **isolation** between transactions, **detect and resolve conflicts**
  - **"Eager conflict detection": identifies conflicts when running.**
  - **"Lazy conflict detection": detect only when they try to commit.**

Integrated with high-level programming language

```
void PushLeft(DQueue *q, int val) {
  QNode *qn = malloc(sizeof(QNode));
  qn->val = val;
  atomic {
    QNode *leftSentinel = q->left;
    QNode *oldLeftNode = leftSentinel->right;
    qn->left = leftSentinel;
    qn->right = oldLeftNode;
    leftSentinel->right = qn;
    oldLeftNode->left = qn;
  }
}
```

Software Transactional Memory

- A rich vein of research over the last decade, focusing on approaches that can be implemented on current mainstream processors.
- There is a very large space of different possible STM designs, and simple **version-number-based approach** is by no means the state of the art.

Software Transactional Memory

Record Version number into **read-log**

Record the change into **write-log** and **undo-log**

```
void PushLeft(DQueue *q, int val) {
  QN                                    ode));
  qr
  do
    StartTx();
    QNode *leftSentinel = ReadTx(&(q->left));
    QNode *oldLeftNode = ReadTx(&(leftSentinel->right));
    WriteTx(&(qn->left), leftSentinel);
    WriteTx(&(qn->right), oldLeftNode);
    WriteTx(&(leftSentinel->right), qn);
    WriteTx(&(oldLeftNode->left), qn);
  } while (!CommitTx());
}
```

Check conflicts based on version number and decide commit or abort. If commit, increase the version number.

Hardware Transactional Memory

- Provide advantages over STM
  - **Lower overheads**
  - **Better power and energy profiles**
  - **Less invasive**
  - **Strong isolation**
- Identifying Transactional Locations: Via extensions to instruction set
  - **Explicitly transactional HTMs: more flexible**
  - **Implicitly transactional HTMs: reuse**
- Tracking Read-Sets and Managing Write-Sets
  - **Extend the mechanisms of current caches and buffers to track read-sets and manage write-sets**

Hardware Transactional Memory

- Detecting Data Conflicts: Cache coherence protocol
  - **Cache state: MESI (modified/exclusive/shared/invalid)**
- Resolving Data Conflicts
  - **Eager conflict detection: aborting the transaction on the processor that receives the conflicting request, and transferring control to software following such an abort.**
- Managing Architectural Register State
  - **Create a shadow copy of architectural registers at the start of a transaction for restoration.**

Hardware Transactional Memory

- Committing and Aborting HTM Transactions: making all transactional updates visible to other processors instantaneously
  - **Obtain the write permissions for all the transactional addresses**
  - **Block any subsequent requests from other processors**
  - **Drain the store buffer into the data cache**

Application

- Manage **shared-memory data structures** in which scalability is difficult to achieve via lock-based synchronization
- Some Examples
  - **Double-ended queue operations (**`PushLeft`**)**
  - **Graph algorithms**
  - **Quake game server**
- Limitations
  - **The performance of code executing within a transaction can be markedly slower than the performance of normal code**
  - **Parallel/Semantic errors**

Overall Conclusion

- Transactional Memory model provides an unlocked solution and an abstraction mechanism to parallel programming.
- Software Transactional Memory: coexist with other programming abstractions and depends on the environment.
- Hardware Transactional Memory: Efficient but application sensitive.
- Transactional Memory is not a panacea.

Reference

1. Transactional Memory (Second Edition) Tim Harris, James Larus, Ravi Rajwar, Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2010
2. Kunle Olukotun and Lance Hammond. The future of microprocessors. Queue, 3(7):26–29, 2005. DOI: 10.1145/1095408.1095418 1
3. A. S. Grove. Only the paranoid survive. Doubleday, 1996. 1
4. E. A. Lee, "The Problem with Threads," in IEEE Computer, 39(5):33-42, May 2006 as well as an EECS Technical Report, UCB/EECS-2006-1, January 2006.
5. Learning from Mistakes --- A Comprehensive Study on Real World Concurrency Bug Characteristics. Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. 13th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'08).
6. Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lockfree data structures. In ISCA ’93: Proc. 20th Annual International Symposium on Computer Architecture, pages 289–300, May 1993. DOI: 10.1145/165123.165164 6, 17, 149, 155
7. David B. Lomet. Process structuring, synchronization, and recovery using atomic actions. In ACM Conference on Language Design for Reliable Software, pages 128–137, March 1977. DOI: 10.1145/800022.808319 6, 62
8. Janice M. Stone, Harold S. Stone, Phil Heidelberger, and John Turek. Multiple reservations and the Oklahoma update. IEEE Parallel & Distributed Technology, 1(4):58–71, November 1993. DOI: 10.1109/88.260295 6, 149, 150, 158