Optimistic Synchronization

...and the Natural Degree of Parallelism of Concurrent Applications

Prof. P. Felber
Pascal.Felber@unine.ch
http://iiun.unine.ch/

Based on work with D. Didona, D. Harmanci, P. Romano, J. Schenker
Why TM?

- Multi-cores are here, the “free ride” is over
  - Cannot rely on ever increasing CPU frequency
- Concurrent programming necessary...
  ...but hard to get right and efficient
  - Coarse-grained locking
    - Easy but will not scale
  - Fine-grained locking
    - Can scale, but hard to compose, debug, reason about (deadlocks, process crashes, priority inversions, ...)
- Paradigms to simplify parallel programming?
TM in a Nutshell

- TM can simplify concurrent programming
  - Sequence of instructions executed atomically
    - BEGIN ... READ / WRITE ... COMMIT
  - Alleviates problems of locks: both safe and scalable
  - Optimistic CC: upon conflict, rollback & restart

```
T1: B R W C B W R C B W W C B R R C
T2: B W R→A W R W→A W R W C→A W R W C
```
VELOX Stack Overview
Concurrent Programming for Multi-core Architectures

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
When to use TM

- TM is not good for all applications
- There should be **some conflicts**…
  (otherwise no synchronization necessary)
- …but **not too many**…
  (otherwise pessimistic CC is better)
- …with **not-too-long** transactions…
  (to keep the cost of aborts reasonable)
- …involving data **not known statically**
  (otherwise no simpler than locks)
Which TM Algorithm?

- Many algorithms and design choices
  - Lock-free vs. obstruction-free vs. lock-based
  - Object- vs. word-based
  - Visible vs. invisible reads
  - Encounter- vs. commit-time locking
  - Write-through vs. write-back
  - Other implementation parameters...
- Performance of TM also depends on workload

![Graph showing throughput vs. configuration]
Another Dimension

- Most existing TM benchmarks have a variable number of threads
  - Should we have as many threads as cores?
  - More cores = more processing power...
    = more concurrency = more potential conflicts...
    ¿=? less performance
  - Is there an “optimal” number of threads for a given workload (and a given TM algorithm/configuration)?
  ⇒ “Natural degree of parallelism” of the workload
Natural Degree of Parallelism

STAMP

vacation-high

vacation-low

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Natural Degree of Parallelism

STAMP

kmeans-high

kmeans-low

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Natural Degree of Parallelism

STAMP

intruder

ETLmodular
ETLsuicide
CTLsuicide

yada

ETLmodular
ETLsuicide
CTLsuicide

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Natural Degree of Parallelism

STAMP

ssca2

labyrinth

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
A wide variety of transactions!
Note on Contention Management

- Proper contention management can avoid degradation of throughout for high thread counts
  - E.g., serialize conflicting transactions
  - ... but it does not improve performance...
  - Typically remains flat
- ... and it wastes resources
  - More threads, same performance!
**STM Bench7**

- Strive to avoid performance degradation
  - Serialization with TM-aware scheduler helps
  - Performance still degrades with high thread counts
Adapting Concurrency

- **Goal**: dynamic adaptation of the degree of parallelism in TM applications
- **Requires support** for variable number of threads
- **Refactoring** of application/benchmark code
  - Split work into small pieces
  - Assign pieces to tasks
  - Use thread pool to execute tasks
- **Introduction of a new “main” loop**
  - 3 phases: measurement, decision making, adaptation
  - Threads run tasks that process pieces
Importance of Number of Pieces

- Too many pieces ⇒ extra overhead
- Too few pieces ⇒ less even load balancing

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Thread Management

- Exploration-based scaling
  - Measure performance of application threads
    - Period duration adjusted according to throughout
  - Explore neighboring/random configurations
    - Adjust the number of threads
  - Upon improvement continue, otherwise revert
Hill Climbing Algorithm

- Simple hill-climbing algorithm
  - Increase #threads
    - If performance improves: keep on adding threads
    - If performance degrades: revert #threads
  - Decrease #threads
    - If performance improves: keep on removing threads
    - If performance degrades: revert #threads
  - Periodically choose random #threads
    - If performance improves: keep #threads
    - If performance degrades: revert #threads
  - Step sizes: linear or logarithmic
Exploration-Based Scaling

STAMP

vacation-high

vacation-low
Exploration-Based Scaling

STAMP

Performance is better than with any static number of threads!
How can that be?
Usually we slowly converge toward an optimal value.
Exploration-Based Scaling

STAMP

With intruder, the number of threads continuously evolves (no optimal value?)
Exploration-Based Scaling

- The intruder benchmark models an intrusion detection system
  - Processes fragments of messages
- Three types of transactions
  - **T1** — Get network packet
  - **T2** — Find position in data structure
    - If message complete, search for intrusions
  - **T3** — Store information about detected intrusions
Exploration-Based Scaling

STAMP

- $2^{nd}$/3rd transactions grow
- Cost of 2nd transaction dominates
- More conflicts near the end of the execution (larger data structures)
Exploration-Based Scaling

Synthetic benchmark

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Dynamic Workloads

- Some workloads have dynamically-changing properties
  - Varying transaction lengths (e.g., as data structures get populated)
  - More conflicts at certain times
  - Different mixes of transactions
- The optimal degree of parallelism is not always a constant value
- Likely to happen in real-world applications (?)

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Exploration vs. Modeling

● Alternative: predictive models
  ● Forecast the impact of adding/removing threads
  ● Decision based on model

● Pros
  ● No overhead from testing (trial and error)
  ● Good for replicated/distributed TM
    ● Exploration does not scale well across nodes

● Cons
  ● Accurate models hard to get right
Challenges of DTM

- Increase in the dimensionality of the problem
  - \#threads per node × \#nodes in the system
  - Quadratic growth of the solution space
- Must also account for contention on network
  - The cost of distributed synchronization dominates
- Costs of on-line exploration are asymmetric
  - Changing local \#threads
    - Cheap: like in centralized case
  - Changing \#nodes
    - Expensive: global synchronization & state
Transactional Auto-Scaler (TAS)

- Performance forecasting model for DTM
  - Predict the throughput of a DTM application when deployed over different scales

![TPC-C throughput and abort graphs](image-url)
Exploration + Modeling

- Idea: combine both approaches
  - Online exploration can improve model’s accuracy
  - Model can improve scalability of exploration

- Exploration gathers feedback from the system
  - Vary the local degree of concurrency: avoid expensive global system’s reconfigurations
  - Use ML to learn a corrective factor, function of
    - Platform’s scale
    - Application’s workload
Self-Correcting TAS

- Adjust #nodes and #threads per node based on model
- Model is corrected using local exploration
Self-Correcting TAS

- Correction factor improves accuracy of model
  - Average relative absolute error decreases from 103% to 5%

TPC-C throughput

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Throughput (tx/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 th real</td>
<td>1000</td>
</tr>
<tr>
<td>2 th TAS</td>
<td>1500</td>
</tr>
<tr>
<td>2 th SCTAS</td>
<td>2000</td>
</tr>
<tr>
<td>4 th real</td>
<td>1500</td>
</tr>
<tr>
<td>4 th TAS</td>
<td>2000</td>
</tr>
<tr>
<td>4 th SCTAS</td>
<td>2500</td>
</tr>
<tr>
<td>8 th real</td>
<td>1500</td>
</tr>
<tr>
<td>8 th TAS</td>
<td>2000</td>
</tr>
<tr>
<td>8 th SCTAS</td>
<td>2500</td>
</tr>
</tbody>
</table>

Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications
Summary

- Performance depends on workload
  - Driven mainly by conflicts
  - Long transactions, large write sets, etc.
- There is no one-size-fits-all TM
  - Many dimensions for adaptation
  - Degree of concurrency particularly important
  - More knob: (opt|pess)imistic, (H|S)TM, etc.
- Some workloads evolve at runtime
  - Dynamic adaptation techniques can help adjust TM
- Algorithm effectiveness depends on context