Autonomic Parallelism Adaptation on Software Transactional Memory

Naweiluo Zhou – Grenoble University/INRIA, France
Gwenael Delaval – Grenoble University, France
Bogdan Robu – Grenoble University, France
Eric Rutten – INRIA, France
Jean-Francois Mehaut – Grenoble University/INRIA, France

Naweiluo.Zhou@inria.fr

October 7th, 2015
Overview

1 Introduction

2 Background on TM and Control Techniques
   - Transactional Memory
   - Autonomic Computing Techniques

3 Autonomic Parallelism Adaptation
   - Profiling procedure
   - feedback control loop

4 Performance Evaluation

5 Discussion

6 Conclusion and Future Work
   - Conclusion
   - Future Work
1. Introduction

Multi-core Processor

- Multi-core processors are ubiquitous, more parallelisms/concurrency levels give higher performance?
- Many threads execute concurrently. Threads share data. More threads maybe more conflict!

Synchronization VS Computation

A high concurrency level may decline computing time, but increase synchronization time. How to handle the trade-off between synchronization and computation?
1. Introduction

**Locks**

A traditional way for synchronization. But:
- Deadlocks, vulnerability to failures, faults...
- Difficult to detect deadlocks
- Hard to figure out the interaction among concurrent operations

**Transactional Memory**

Lock-free, therefore no deadlocks! HTM, STM and HyTM.
1. Introduction

Runtime Parallelism Adaptation:

- choice of parallelism significantly impacts on performance.
- onerous to set a parallelism offline, especially for a program with online behaviour fluctuation.
- feedback control loops to manipulate parallelism autonomously.

Contribution of this paper

- An adaptive profiling framework for searching and applying optimal parallelism online
- a feedback control loop to enable autonomy and reduce overhead
2. Transactional Memory

**Concepts**

- Shared variables are wrapped by **transactions** (atomic blocks)
- Concurrent accesses are performed inside transactions
- Transactions are executed speculatively and can either commit or abort. No other intermediate status
- Can be implemented in STM (e.g., TinySTM, SwissTM...), HTM and HyTM
2. Transactional Memory

Three concepts

1. Commit: a transaction succeeds—changes are made
2. Abort: a transaction has a conflict — changes are discarded
3. Rollback: re-execute the aborted transactions
2. Transactional Memory

Example

consider three threads read/write data from/to the objects of different memory locations. Access occur inside transactions

Case 1
thread 1 reads object 3
thread 2 reads object 3

Case 2
thread 2 reads object 7
thread 3 reads object 7
Autonomic Computing

A system is regarded as an autonomic control system if it has one of the features:

- **Self-optimization**: seek to improve performance & efficiency
- **Self-configuration**: a new component learns the system configurations
- **Self-healing**: recover from failures
- **Self-protection**: defend against attacks
Autonomic Computing

Elements of a feedback control loop:

1. Managed element: any software or hardware resource
2. Autonomic manager—a software component: monitor, plan, knowledge
3. Sensor: collect information
4. Effector: carry out changes

Figure: A feedback control loop
Autonomic Computing

Components of the autonomic manager:

1. Monitor: sampling
2. Analyser:
3. Knowledge
4. Plan: use the knowledge of the system to do computation
5. Execute: make changes

Figure: A feedback control loop
Autonomic Computing

Heptagon Programming Language

1. straightforward for programming control loops.
2. composed of different states.
3. values in the input flows are used to compute the outputs which decides the next state.
Autonomic Computing

Heptagon Programming Language

```plaintext
1 node delayable(r,c,e:bool) returns (a,s:bool)
2   let
3     automaton
4       state Idle
5         do a = false ; s = r and c
6         until r and c then Active
7           | r and not c then Wait
8       state Wait
9         do a = false ; s = c
10        until c then Active
11       state Active
12         do a = true ; s=false
13        until e then Idle
14     end
15   end
```

Grenoble University/INRIA, France
Zhou, Robu, Delaval, Rutten, Mehaut
Naweiluo.Zhou@inria.fr
3. Profiling Algorithm

What we measure

1. parameters: commits, aborts, time
2. commit ratio = commits/(commits + aborts), throughput = commits/time
3. CR thresholds

Figure: Periodical profiling procedure
3. Profiling Algorithm

1. Profiling starts once a program starts. Two threads are active, others are suspended.

2. First profiling interval, parallelism is only increased until throughout significantly falls.

3. At non-action interval, only check CR.

4. Increase or decrease parallelism until throughput shows significantly drop.

5. Set parallelism and CR range.

Figure: Periodical profiling procedure.
3. The feedback control loop

**Figure:** The feedback control loop on adjusting parallelism

**Figure:** Periodical profiling procedure
3. The feedback control loop

Figure: The feedback control loop on adjusting parallelism

Figure: Periodical profiling procedure
3. Feedback Control Loop

1. Control objective: maintain the maximum throughput and keep global CR staying in a certain range within which the conflicts are minimized.

2. Inputs: commits, aborts, time.

3. Outputs: optimum parallelism, next parallelism, CR up_threshold, CR low_threshold, profile_flag.

Figure: The feedback control loop on adjusting parallelism.
Two decision functions

1. for parallelism: increase or decrease parallelism based on CR and throughput

2. for CR thresholds:

\[ CR_{\text{UP}} = CR_{\text{optimum}} \times 1.1 \]

\[ CR_{\text{LOW}} = CR_{\text{optimum}} \times 0.9 \]

**Figure:** The feedback control loop on adjusting parallelism
Two decision functions

**Increase State**

1. CR=1;
2. throughput increase &&
   \( tn < max \ tn \)

---

**Figure**: The feedback control loop on adjusting parallelism
3. Implementation

A monitor to control parallelism: three entry points

```
stm_commit() { // when a tx commits
    ...
}
stm_thread_init() { when a tx thread init
    ...
}
stm_thread_exit() { when a tx thread terminates
    ...
}
```
Implementation

- balance threads execution time, avoid threads starvation
  - First In First Out queues
  - round-robin rotate
Experimental Platform

Hardware

- 4 processors with 2.4 GHz frequency, 32 cores and 128 GB RAM.

Software

- STM: tinySTM
- Control: Heptagon
- Benchmarks: EigenBench, STAMP
Benchmark setting

### EigenBench
- stable behaviour
- online fluctuation: minor modification to Eigenbench to enable online changes

### STAMP
- labyrinth
- genome
- intruder
- vacation
Online throughput variation

Figure: EigenBench with online behaviour

Figure: genome
Runtime thread number change

Figure: thread number change on EigenBench with online behaviour
Time comparison for **EigenBench** on static and adaptive parallelism

**Figure**: the data set with stable online behaviour

**Figure**: the data set with online variation
Time comparison for **STAMP**

- **intruder**

- **vacation**

- **genome**
# Time comparison

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>best case</th>
<th>median value</th>
<th>worse case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>eigenBench</strong> (stable)</td>
<td>-5%</td>
<td>+8%</td>
<td>+46%</td>
</tr>
<tr>
<td><strong>eigenBench</strong> (online variation)</td>
<td>0%</td>
<td>+10%</td>
<td>+54%</td>
</tr>
<tr>
<td>genome</td>
<td>+19%</td>
<td>+95%</td>
<td>+99%</td>
</tr>
<tr>
<td>vacation</td>
<td>-23%</td>
<td>+62%</td>
<td>+84%</td>
</tr>
<tr>
<td>labyrinth</td>
<td>-71%</td>
<td>-33%</td>
<td>+61%</td>
</tr>
<tr>
<td>intruder</td>
<td>-6%</td>
<td>+41%</td>
<td>+43%</td>
</tr>
</tbody>
</table>

**Table:** Performance comparison on difference benchmarks. The performance of adaptive parallelism is compared with the minimum value, median value and the maximum value of the static parallelism.
Overhead analysis

- incorrect parallelism.
- thread migration.
- the choice of profiling length and non-profiling length.
- the choice of the thread number to manipulate at each parallelism profiling length.
- the choice of throughput variation rate

Figure: throughput fluctuation
Discussion

pros and cons

- demonstrate an effective way to obtain the optimum parallelism online.
- short-size transaction suffers from overhead by calling the monitor (e.g., intruder); reduce frequency of calling the monitor.
- long-size transaction: too much time spent for profiling (e.g., labyrinth).
- thread migration issues (e.g., vacation, genome)
Conclusion

- investigate an autonomic parallelism adaptation approach on a STM system
- examined the performance of different static parallelism and concludes that runtime regulation of parallelism is crucial to performance
- introduce a feedback control loop to automate the choice of parallelism at runtime
- analyse the implementation overhead and discussed the advantages and limitation of our work
Future Work

- investigate thread migration issues.
- design more loops to control thread affinity and further enhance performance.
- ...
Questions?