Predicting the cost of lock contention in parallel applications on multicores using analytic modeling

Xiaoyue Pan  Jonatan Linden  Bengt Jonsson

Uppsala University, Sweden

November 23, 2012
Scalability of parallel applications

- Ideal vs. measured speedup

Gap may be composed of
- Work load imbalance
- Hardware contention, e.g., cache, memory
- Software contention, e.g., lock contention
Predict the cost of lock contention

- How much of the gap is caused by lock contention?
Problem to solve

- A multi-threaded program with a shared lock
  - Critical section
  - Lock contention

- Predict the cost of lock contention when scaling up to a certain number of threads (e.g., 64)
Target applications

- Constant number of threads
- Different threads may have different locking behaviors
- Locks not nested
- Lock holding time does not change with more threads
- Repetitive locking behavior
Our approach

Different approaches:

- Run the application directly
- Simulation
- Analytical modeling
Our approach

- Different approaches:
  - Run the application directly
  - Simulation
  - Analytical modeling

- Our approach - analytical modeling with queueing networks
  - Profile the application to get
    - Lock access patterns
    - Time spent in/out of critical sections
  - Use queueing network models to predict the lock contention
An example queueing network

- **Node 0**: Infinite-server node
- **Node 1**: One-server node
- **Node 2**: One-server node

Routing Probabilities:
- From Node 0 to Node 1: 0.4
- From Node 0 to Node 2: 0.6
An example queueing network

Infinite-server node

One-server node

Routing Probability: $\frac{9}{24}$
An example queueing network

Infinite-server node

One-server node

Routing Probability $\frac{10}{24}$
An example queueing network

Infinite-server node

One-server node

Routing Probability

node_0

node_1

node_2

\[ t_1 \]

\[ t_2 \]

0.4

0.6

1

m

\[ \frac{11}{24} \]
**Example**

**Skeleton Program**

<table>
<thead>
<tr>
<th>loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute for $t_P$</td>
</tr>
<tr>
<td>if $&lt;\text{condition}&gt;$</td>
</tr>
<tr>
<td>Acquire $\text{lock}_1$</td>
</tr>
<tr>
<td>Compute for $t_1$</td>
</tr>
<tr>
<td>Release $\text{lock}_1$</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>Acquire $\text{lock}_2$</td>
</tr>
<tr>
<td>Compute for $t_2$</td>
</tr>
<tr>
<td>Release $\text{lock}_2$</td>
</tr>
<tr>
<td>endloop</td>
</tr>
</tbody>
</table>

**Figure:** Example program skeleton: executed by all threads
Model lock behavior with closed queueing networks

- Threads as customers
- Local computation as \textit{infinite-server nodes}
- Lock access as \textit{one-server nodes}
- \textit{Routing probabilities} describes the program flow
- Average time in/out of critical sections as service times of the queues
- Account for lock overhead in the service times

![Diagram showing lock contention and holding times](image)

**Figure:** Lock holding time and lock contention
What can we do with the model

- Solve the model with standard methods (e.g., mean value analysis (MVA))
- Output from the model
  - Average queue length at each node - *Average number of threads waiting for the lock*
  - Average waiting time at each node - *Average lock contention*
Evaluation

- Two target multi-core machines have been used in our experiments:
  - a 8-core machine (Intel Nehalem E5520 with QPI)
  - a 64-core machine (Intel Nehalem X6550 CPUs with QPI).
- CLH locks (a low-overhead spin lock) and Pthread mutex locks are used.
- Benchmarks
  - Simple microbenchmark
  - Microbenchmark emulating real programs
  - A real benchmark from the PARSEC benchmark suite - dedup
Microbenchmarks

- Simple microbenchmark

  loop
  Compute for $t_c$
  Acquire lock
  Compute for $t_l$
  Release lock

  end loop

  Figure: Microbenchmark compute and lock

- All threads execute the same loop
- In the loop, first local computation, then access the shared lock
Simple benchmark with exponential distribution

- Time in/out the critical sections exponentially distributed
- Model has 3% average relative error

![Figure: measured vs. modeled lock contention, Legends shows the different lock holding time](image-url)
Simple benchmark with deterministic distribution

- Time in/out the critical sections are deterministic
- Model has 3.8% average relative error

![Graph showing lock contention vs. number of threads with measured and modeled lock holding times for different lock holding durations.]
Simple benchmark with uniform distribution

- Exponential and deterministic distributions not realistic
- Uniformly distributed in/out of the locks time with difference variance
- Exponential distribution provides an overestimation
- The model has average relative error below 1% for 1 – 8 threads and 4.8% for 16 – 64 threads
Evaluating the lock contention of *dedup*

- The *dedup* benchmark
  - Pipelined parallelism (5 stages in total)
  - 2 stages (ChunkProcess and Compress) share a hashtable (each bucket protected by a lock)

*Figure:* Structure of the dedup benchmark: figure taken from [1]
Modeling *dedup*

- Profile a run with 1 thread in each stage
  - extract the parameters for the queueing network
- Each pipeline stage is modeled as a class of customers
- Service times measured directly from the profile run
- Construct the queueing network
- Solve the queueing network to get the lock contention
Evaluation result of *dedup*

**Figure:** *dedup* measured and modeled contention for the *ChunkProcess* and *Compress* stages
Conclusions

- Lock contention prevents scaling
- Lock access patterns and lock holding times affect lock contention
- Lock overhead need to be considered
- Our method can be used to predict the cost of lock contention