Universität Heidelberg
Institut für Technische Informatik

Prof. Dr. Robert Strzodka
Steffen Lammel

Advanced Parallel Computing
Term 2015 (Summer)

Exercise 9

- **Return electronically (MOODLE) until Tuesday, 23.06.2015 14:00**
- **Include name on the top sheet. Stitch several sheets together.**
- **A maximum of two students is allowed to work jointly on the exercises.**

9.1 Reading

Read the following paper and provide a review as explained in the first lecture (see slides):

- Milo M K Martin, Mark Hill, David Wood, Token Coherence: A New Framework for Scalable Directory-Based Cache Coherence Schemes, IEEE Micro 2003.
- Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. 2010. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. IEEE Micro 30, 2 (March 2010), 16-29.

(25 points)

9.2 Red-Black Tree – Fine grain locking

Based on the versions developed in the previous exercises, we will now start implementing a fine-grain locking scheme. Implement a version that uses a shared read lock and exclusive write locks.

Resources:

- http://en.wikipedia.org/wiki/Red-black_tree (I)
- http://www.slideshare.net/piotrszymanski/red-black-trees (II)

- Implement a red-black tree with fine grain locking. It should support the following two operations: `add, search`. **Use the top-down versions, not the bottom-up ones!**
- Ensure that the RBTs your code generates fulfill the 5 requirements stated in the link (I) above. Opportunities to check the consistency would be after initialization and after the experiment (maybe after each operation for verification purposes). Obviously, do not include these checks in your time measurements.
- For insert, ensure that all elements are unique.

(50 points)

9.3 Red-Black Tree – Performance of fine-grain locking

- Perform a performance evaluation of the sequential RBT function.
- Initialize the RBT with 10M elements before starting the measurement (i.e., perform 10M insert operations).
- Generate a stream of update actions, with each action including the command (add, search) and an associated data (index to be inserted or searched). Generate this update

stream using random values. Use different ratios between search and insert operations as shown below!

- Measure the performance in terms of operations per second (OPS) for an update stream with varying lengths like shown below. Report also graphically and interpret your results! Use `moore` for this performance assessment.

	Operations per second (90/10 search/insert ratio)									
Operations	**Sequential**	**Best coarse grain**	**Fine-grain locking**							
Threads	**1**		**2**	**4**	**8**	**12**	**16**	**24**	**32**	**48**
100k										
500k										
1M										
5M										
10M										
25M										
50M										

	Operations per second (50/50 search/insert ratio)									
Operations	**Sequential**	**Best coarse grain**	**Fine-grain locking**							
Threads	**1**		**2**	**4**	**8**	**12**	**16**	**24**	**32**	**48**
100k										
500k										
1M										
5M										
10M										
25M										
50M										

	Operations per second (10/90 search/insert ratio)									
Operations	**Sequential**	**Best coarse grain**	**Fine-grain locking**							
Threads	**1**		**2**	**4**	**8**	**12**	**16**	**24**	**32**	**48**
100k										
500k										
1M										
5M										
10M										
25M										
50M										

Also, report your results graphically and interpret your results!

(25 points)

Note: RBT Visualization

The working material includes a python script (rbtv.py) to visualize your RBTs and is supposed to help you with debugging your code.

In order to use the visualization you have to dump your tree in a specific (but simple) format. The source code of the script contains a syntax description, example C code to produce output in the desired format and usage information.

Obviously, subsequent exercises will continue to use RBTs. So ensure a nice coding and plenty of meaningful comments! Make use of header files, and structure your code in functions!

Total: 100 points