Ruprecht-Karls University of Heidelberg
Institute of Computer Engineering

Holger Fröning
Bernhard Klein

Advanced Parallel Computing
Summer term 2019

Exercise 6

- **Return electronically until Tuesday, June 11, 9:00 am**
- **Include name on the top sheet.**
- **A maximum of two students is allowed to work jointly on the exercises.**
- **Moore is offline from Mai 31, 12:00 am to June 4, 9:00 am**

6.1 Reading

Read the following two papers and provide reviews as explained in the first lecture (see slides):

- Calin Cascaval, Colin Blundell, Maged Michael, Harold W. Cain, Peng Wu, Stefanie Chiras, and Siddhartha Chatterjee. 2008. Software Transactional Memory: Why Is It Only a Research Toy?. Queue 6, 5 (September 2008), 46-58.
- Aleksandar Dragojević, Pascal Felber, Vincent Gramoli, and Rachid Guerraoui. 2011. Why STM can be more than a research toy. Commun. ACM 54, 4 (April 2011), 70-77.

(25 points)

6.2 Parallel Prefix Sum - Development

Develop a multi-threaded program that performs a parallel prefix sum operation. A detailed description of this operation can be found in this reference: Mark Harris, Parallel Prefix Sum (Scan) with CUDA, NVidia Whitepaper, 2007 (provided with this exercise). Use Blelloch's algorithm for your implementation.

- Use random data as input array, which in combination with a seed parameter can be made deterministic.
- Your program should accept two parameters, one for the length of the array and one for the number of threads working cooperatively on the operation.
- You can limit the set of allowed array sizes to power-of-two values.

(35 points)

6.3 Parallel Prefix Sum - Analysis

Based on the program developed in 6.2, perform a performance analysis on *moore*. Report scan times (in ms, or s) together with the number of threads in the following table. The scan time should not include the time required to initialize the input array, only the time required to perform the parallel prefix scan operation. Choose a single input array size that yields reasonable run times for all thread counts. Feel free to make use of compiler optimizations.

Furthermore, use the *numactl* tool to control data placement. First, make yourself familiar with this tool (e.g., using man-pages). *moore* has four nodes (i.e.: sockets), each with 12 CPUs (i.e.: cores). Then, perform three sets of experiments in total:

1. Execute your program with default *numactl* parameters.
2. Manually create contention by forcing memory allocations to a single node. (*membind* parameter)
3. Now, distribute the input array over all nodes using the *interleave* parameter.

	Parallel Prefix Scan I	**Parallel Prefix Scan II**	**Parallel Prefix Scan III**
Thread Count	*No NUMA optimizations (default)*	*Single NUMA memory (membind=...)*	*Interleaved NUMA memory (interleave=...)*
1			
2			
4			
8			
12			
16			
24			
32			
40			
48			

Also include a graphical representation. Try to interpret your results! Good luck!

(40 points)

Total: 100 points