Ruprecht-Karls University of Heidelberg
Institute of Computer Engineering

Holger Fröning
Bernhard Klein

Advanced Parallel Computing
Summer term 2019

Exercise 2

- **Return electronically until Tuesday, May 7, 9:00 am**
- **Include name on the top sheet.**
- **A maximum of two students is allowed to work jointly on the exercises.**

2.1 Reading

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

- Christoph Lameter. 2013. An overview of non-uniform memory access. Commun. ACM 56, 9 (September 2013), 59-54. DOI=http://dx.doi.org/10.1145/2500468.2500477
- Fabien Gaud, Baptiste Lepers, Justin Funston, Mohammad Dashti, Alexandra Fedorova, Vivien Quéma, Renaud Lachaize, and Mark Roth. 2015. Challenges of memory management on modern NUMA systems. Commun. ACM 58, 12 (November 2015), 59-66. DOI: https://doi.org/10.1145/2814328

(25 points)

2.2 Pointer Chasing Benchmark

The *Pointer Chasing Benchmark* plots memory load latencies as a function of an array size and a stride when walking through the array. The various latencies reported in combination with the array size and the stride give a nice impression about the memory hierarchy, i.e. if the array fits into a certain cache level, what latency penalties are associated with cache misses or TLB misses, and what the latency of main memory is. Also, as the array size divided by the stride reduces the amount of touched elements, conclusions about associativity can be made.

The *Pointer Chasing Benchmark* is also known as *Memory Mountain* if bandwidth is plotted instead of latency. In this exercise, use the source code provided to assess the performance of a multi-core system. Ensure that the parameters `ARRAY_MIN`, `ARRAY_MAX` are set correctly.

Use *moore* for these experiments, but ensure that you're not disturbing others. That means, before you start your test program, ensure that the system is idle. Also, report the results from another system that is available for you, for instance one of your computers at home. Include a description of the system.

For both experiments, besides reporting the results graphically (array sizes as series, stride on x-axis, and latency on y-axis) include a sophisticated interpretation of the results, including:

- Details of the memory hierarchy (cache levels, associated latency)
- Size of the caches
- Main memory access latency
- TLB miss penalty
- Associativity of the caches and the TLB

(30 points)

2.3 Multi-threaded load bandwidth

Write a multi-threaded test program that loads collaboratively an array with a given size from memory. Collaboratively means that each thread is assigned a memory region that it is sequentially loading from memory. Ensure the use of the keyword *volatile* to prevent the compiler from performing optimizations with the associated load operations. Measure the time required (using a high-precision timer) and report bandwidth for a varying thread count. Ensure that the array has a size that prevents scheduling effects in the time measurement and yields stable results.

Perform this experiment on *moore* and another computer of your choice. Report bandwidth graphically over the number of threads. Compare against the memory bandwidth stated in the datasheet of the processors! Interpret the results!

(20 points)

Total: 75 points

Additional Information

Pthread introduction

For development and experiments with Pthread programs, you can use the computers of the terminal room of the institute. For remote access, follow this procedure:

1. Connect to the Uni Heidelberg network using the VPN client.
2. Open an ssh connection to *moore*:
   `ssh apcXX@ceg-moore.ziti.uni-heidelberg.de`
3. Immediately after the first login, change the password using the `passwd` command.
4. It is highly recommended to use SSH-keys to login internally without passwords.

If you're not familiar (any more) with Pthreads, you might want to read this tutorial:
`https://computing.llnl.gov/tutorials/pthreads/`