CHALMERS
UNIVERSITY OF TECHNOLOGY

EDA284 Lab 3: Simulation of Cache Coherence Protocols

Contacts: Mustafa Abduljabbar, Madhavan Manivannan and Miquel Pericàs,
Emails: musabdu, madhavan, miquelp@chalmers.se

March 13, 2020

We assume a cache coherent multi-core system as the baseline for this lab. Figure 1 shows the organization of the baseline system. Each core has access to a private L1 cache and a Last Level Cache (LLC) that is shared across multiple cores. The private L1 caches communicate with the LLC using a simple bus. Coherence is maintained across the multiple private L1 caches using a snooping based write invalidate cache coherence protocol. This simple baseline will be used to understand different cache coherence protocols. The baseline system is modeled using MultiCacheSim https://github.com/blucia0a/MultiCacheSim and Pin tool by Intel https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

Background

How are memory accesses handled? Memory accesses (issued by threads) in a multi-threaded application are captured by Pin. Each request is then handled by the pin tool that models the baseline cache hierarchy shown in Figure 1. The access is sent to the core's private L1 cache to check if the data are available in the cache. In case the data are found in the L1 cache with suitable permissions, the access is complete and control is returned back to Pin. If the request misses at the private L1 cache, the private caches of the other cores and the LLC are snooped. If the requested data resides in a remote cache the appropriate cache responds with the data. In case the requested

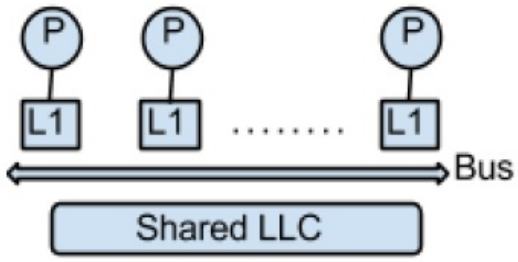

Figure 1: Baseline Organization - L1 DCache 64K (8way,64byte blocks) LLC 4MB (16way, 64byte blocks)

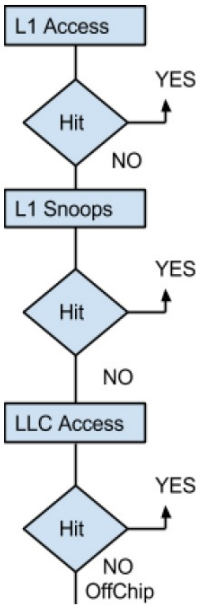

Figure 2: Memory Access Flowchart

data are not available in the on-chip hierarchy the request is considered an LLC miss. Figure 2 shows a generic flowchart of how memory accesses are handled in the baseline system. Please note that this only provides a simplified view of the actions involved at each level of the on-chip cache hierarchy.

Cache Coherence Protocols

The cache coherence protocols we use for evaluation are:

(1) MSI protocol.

(2) MESI protocol.

(3) MESI protocol optimized for migratory sharing.

For details regarding how MSI and MESI protocols work, please refer to the textbook. Protocol 3 (MESI + migratory) includes a small modification to the baseline MESI protocol. In this protocol, when a Read request for a modified block is received by a remote cache, the line is forwarded with exclusive permissions to the requester and the copy in the remote cache is invalidated. Interested readers may refer to the source code of MultiCacheSim for the cache coherence protocol.

Microbenchmarks

We use three microbenchmarks for comparing different coherence protocols. These microbenchmarks have been designed to stress certain communication patterns so that they can be used for

comparative evaluation of the different protocols.

Running the Benchmarks

The pin tarball can be found here https://www.dropbox.com/s/8tguhwmuwzeorb7/eda284_lab3.tar.gz?dl=0. The benchmark tarball contains microbenchmark sources. Start by first building the microbenchmarks and the pin tool (found inside ./source/tools/MutiCacheSim/). The generic command for invoking pin and using a pin tool is as follows.

```
./pin -t <tool name> <tool options> -- ./app_binary <app options>
```

To simulate the micro-benchmark on a certain configuration use the following command:

```
./pin -ifeellucky -mt -t ./source/tools/MultiCacheSim/obj-intel64/mcs.so \
-csize 65536 -bsize 64 -assoc 8 -llc -llcsize 1048576 -llcbsize 64 \
-llcassoc 16 -numcaches 4 -protos \
./source/tools/MultiCacheSim/obj-intel64/mcs_mesi.so \
-- benchmark_path <benchmark options> \
```

The parameters used in the commandline (listed above) configure the baseline system.

```
./pin -> Pin Binary
ifeellucky -> fixes compatibility issues
mt -> multithreaded mode
t ./source/tools/MultiCacheSim/objintel64/mcs.so > MultiCacheSim pin tool
csize -> L1 cache size
bsize -> L1 block size
assoc -> L1 block associativity
llc -> enable LLC
llccsize -> LLC cache size
llcbsize -> LLC block size
llcassoc -> LLC block associativity
protos -> coherence protocol of interest
numcaches -> Number of Private Caches (Should match the number of application threads)
```

Interpreting the Results

The simulator only reports cache access statistics (local accesses, remote accesses, hits, misses, etc). For this lab you can estimate the execution time of microbenchmarks based on these cache access statistics. To estimate the execution time you should assume that a private L1 access (hit) takes 1 cycle, an LLC access (hit) takes 20 cycles, and a remote L1 access (hit) takes 30 cycles. Finally assume that offchip accesses take 50 cycles each.

Tasks

Notes

- Do not use results from cpuid0 in your analysis as it includes statistics for the serial section of the program (which is not of interest).

- An int is four bytes and a cache line is 64 bytes.
- To evaluate the effect of working set size, modify the number of counters used by the benchmark.
- Evaluate benchmarks with three Working Set Sizes (WSS smaller than L1 cache capacity, WSS larger than L1 capacity and WSS larger than LLC capacity)

Benchmark Analysis

(a) Analyze the source code of the microbenchmarks 1 and 2, and provide a brief description of how each of the microbenchmarks work.

(b) Comment on the prominent communication patterns in each of the microbenchmarks.

MSI vs. MESI

(a) Evaluate MSI and MESI in the context of microbench1 and microbench2 using 8 threads.

(b) Compare MSI and MESI.

(c) When is having the E state beneficial (over baseline MSI)?

(d) How does the behavior change when the dataset size changes?

MESI vs MESI+migratory

(a) Evaluate MESI and MESI+migratory optimizations in the context of microbench1 and microbench3 using 8 threads.

(b) Compare MESI and MESI+migratory.

(c) Why is one better than the other?

(d) How does the behavior change when the dataset size changes?

Analyzing Cache Access

(a) What is the point of enabling `ALIGN_ALLOC` in microbenchmark3.c (i.e., what are we avoiding)?

(b) Show an example run to explain the difference between microbenchmark3 and microbenchmark4 using pin. Also, reflect on the effect of cache coherence protocols in such scenarios.