CCE3013 – Computer Architecture
Assignment 2012-2013

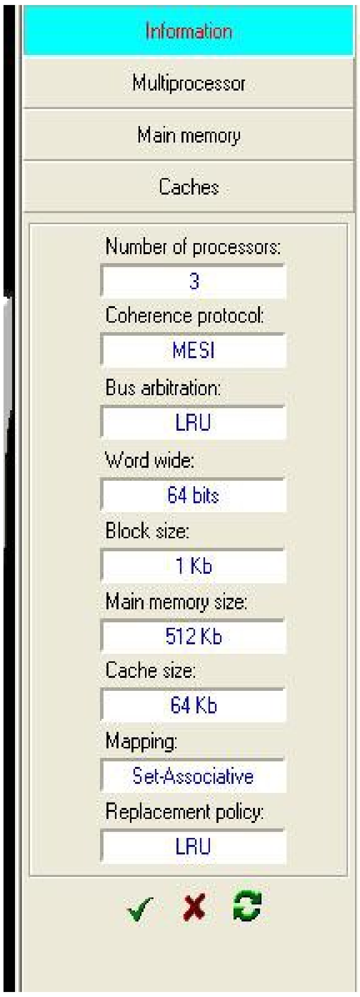
Use of SMPCache software for modelling Cache in a multiprocessor environment.

Requirements: The SMPCache software is available at the Computer Lab. Together with installing this software there is a need for multiprocessor traces. These traces are also available at the lab. The manual “Getting strarted with SMPCache 2.0 is also necessary to understand the installation, operation, experimental settings and analysis of the results.

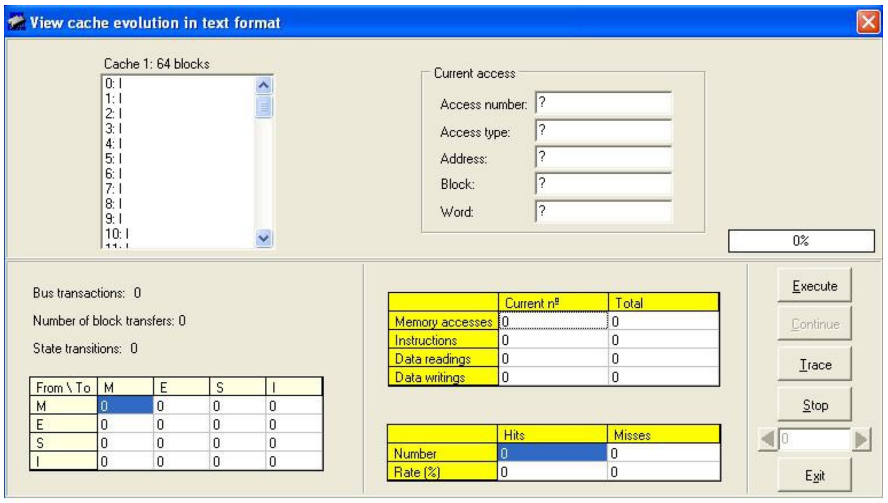
Purpose: The work is a demonstration of how the theoretical concepts associated with cache, cache coherence, multiprocessor bus sharing operates.

Demo 1: To get acquainted with the operation I am suggesting that you look at the step by step trace of one application. The basic information is in Section 6.4.2 of the Manual. I am highlighting certain necessary steps.

1. Firstly you must configure the system. See page 3 configuration files.
2. Set up the configuration as in Figure 1 at the side. This uses 3 processors, MESI for cache coherence, and bocks of size 1 kB, on words that are 8 bytes wide.
3. Use from the sample traces – from FILE, Open Memory Traces – and the appropriate subdirectory - the trace Cexp.prg.
4. From the View Menu set the Cache evolution, and use the text button in the subsequent window. Eventually you should get Figure 2 below



Number of processors: 3
Coherence protocol: MESI
Bus arbitration: LRU
Word wide: 64 bits
Block size: 1 Kb
Main memory size: 512 Kb
Cache size: 64 Kb
Mapping: Set-Associative
Replacement policy: LRU



Cache 1: 64 blocks

0: I
1: I
2: I
3: I
4: I
5: I
6: I
7: I
8: I
9: I
10: I

Current access

Access number: ?
Access type: ?
Address: ?
Block: ?
Word: ?

0%

Bus transactions: 0
Number of block transfers: 0
State transitions: 0

From \ To	M	E	S	I
M	0	0	0	0
E	0	0	0	0
S	0	0	0	0
I	0	0	0	0

	Current nº	Total
Memory accesses	0	0
Instructions	0	0
Data readings	0	0
Data writings	0	0

	Hits	Misses
Number	0	0
Rate (%)	0	0

Figure 2

Note that initially none of the subwindows have any entries. At this point, use execute to start the first instruction from the Cexp.prg trace. You should have a text copy of the Cexp.prg file via Wordpad.

The first few instructions are reproduced here.

```
0 00000170
2 00003600
0 00000174
0 00000178
0 0000017c
0 00000180
3 00008d84
```

Note that the first digit is 0 for instruction, 2 for memory read, 3 for memory write (see page 5 of tutorial). At this point view the changes after EACH execution to the MESI table and to the percentage rate. Continue with the step-by-step for **twenty five** entries of the trace.

From the results obtained in seeing the change in the MESI table for EACH instruction execution, deduce how the cache coherence protocol is working.

Demo 2. Repeat using the same trace program, but this time looking at the Memory Block evolution.
Repeat looking at the Multiprocessor evolution .
Again look at the changes at EACH step and comment on how the different memory usage impacts on the bus.

At this point you will do two of the Projects involving Multiprocessor Traces. These traces are (FFT, Simple, Speech and Weather – again available from the laboratory. The two projects are :

Influence of the Cache Size on the Miss Rate

Influence of the cache size on the Bus Traffic.

The two projects are reproduced below, taken from "Student Projects using SMPCache2.0" - Projects 7 and 8.

The projects are self explanatory. In your report you should show some of the results using sinfle step and breakpoints or using full execution, depending on what you want to highlight.

Name	References	Language	Comments
FFT	7,451,717	Fortran	Parallel application that simulates the fluid dynamics with FFT
Simple	27,030,092	Fortran	Parallel version of the SIMPLE application
Speech	11,771,664	---	Kirk Johnson and David Kranz (both at MIT) are responsible for this trace
Weather	31,764,036	Fortran	Parallel version of the WEATHER application, which is used for weather forecasting. The serial version is from NASA Space Flight Center, Greenbelt, Md.

Table 2: Multiprocessor traces.

3.1. Project 7: Influence of the Cache Size on the Miss Rate

Purpose

Study the influence of the cache size on the miss rate during the execution of a parallel program in a SMP (symmetric multiprocessor). This project also allows us to show that all the previous uniprocessor projects can be performed in a similar way for multiprocessor systems (multiprocessor traces).

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 8.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = LRU.
- Word wide (bits) = 16.
- Words by block = 32 (block size = 64 bytes).
- Blocks in main memory = 524288 (main memory size = 32 MB).
- Mapping = Set-Associative.
- Cache sets = They will vary depending on the number of blocks in cache, but you must always have four-way set associative caches (remember: Number_of_ways = Number_of_blocks_in_cache / Number_of_cache_sets).
- Replacement policy = LRU.

Configure the blocks in cache using the following configurations: 16 (cache size = 1 KB), 32, 64, 128, 256, 512, 1024, and 2048 (cache size = 128 KB). For each of the configurations, obtain the global miss rate for the system using the trace files: *FFT*, *Simple*, *Speech* and *Weather*.

Does the global miss rate increase or decrease as the cache size increases? Why? Does this increment or decrement happen for all the benchmarks or does it depend on the different locality grades? What does it happen with the capacity and conflict (collision) misses when you enlarge the caches? And, what does it happen with the compulsory and coherence misses when you enlarge the caches? Are there conflict misses in these experiments? Why?

In these experiments, it may be observed that for great cache sizes, the miss rate is stabilized. Why? We can also see great differences of miss rate for a concrete increment of cache size. What do these great differences indicate? Do these great differences of miss rate appear at the same point for all the programs? Why?

Compare these results with the results obtained in the project 2 (section 2.2). You can observe that the miss rates are higher for multiprocessor traces than for uniprocessor traces. Do you think that, in general, parallel programs exhibit more or less spatial and temporal locality than serial programs? Why? Is it due to the shared data?

In conclusion, does the increase of cache size improve the multiprocessor system performance?

3.2. Project 8: Influence of the Cache Size on the Bus Traffic

Purpose

Show the influence of the cache size on the bus traffic during the execution of a parallel program in a SMP.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 8.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = LRU.
- Word wide (bits) = 16.
- Words by block = 32 (block size = 64 bytes).
- Blocks in main memory = 524288 (main memory size = 32 MB).
- Mapping = Set-Associative.
- Cache sets = They will vary depending on the number of blocks in cache, but you must always have four-way set associative caches (remember: Number_of_ways = Number_of_blocks_in_cache / Number_of_cache_sets).
- Replacement policy = LRU.

Configure the blocks in cache using the following configurations: 16 (cache size = 1 KB), 32, 64, 128, 256, 512, 1024, and 2048 (cache size = 128 KB). For each of the configurations, obtain the bus traffic (in bytes per memory access) for the system using the trace files: *FFT*, *Simple*, *Speech* and *Weather*. In order to compute the bus traffic, assume that cache block transfers move 64 bytes (the block size) on the bus data lines, and that each bus transaction involves six bytes of command and address on the bus address lines. Therefore, you can compute the address traffic (including command) by multiplying the obtained bus transactions by the traffic per transaction (6 bytes). In the same way, you can compute the data traffic by multiplying the number of block transfers by the traffic per transfer (64 bytes). The total bus traffic, in bytes per memory access, will be the addition of these two quantities divided by the number of memory accesses (references) in the trace (see Table 2).

Does the global bus traffic increase or decrease as the cache size increases? Why (give two reasons, one for the data traffic and another for the address+command bus traffic)? Does this increment or decrement happen for all the benchmarks?

In these experiments, it may be observed that for great cache sizes, the bus traffic is stabilized. Why? We can also see great differences of bus traffic for a concrete increment of cache size. What do these great differences indicate? Do these great differences of bus traffic appear at the same point for all the programs? Why?

In conclusion, does the increase in the cache size improve the multiprocessor system performance? Are the conclusions you obtain similar to the previous ones for the miss rate project?