Student Projects using SMPCache 2.0

1. Introduction

This document contains some ideas for student projects using SMPCache. These idea descriptions are intended as starting point from which other many project assignments could be designed. Students should be familiar with the simulator to carry out any of the projects. We have developed the *"Getting Started with SMPCache"* manual with this aim.

If you have comments about this document or the simulator, please contact Miguel A. Vega at mavega@unex.es (Fax: +34-927-257202) or at the following address:

Miguel A. Vega-Rodríguez

Dept. de Informática, Univ. de Extremadura, Escuela Politécnica

Campus Universitario s/n. 10071. Cáceres. España (Spain)

2. Uniprocessor Traces

We will first study the basic algorithms and concepts that are present in every cache memory system, uniprocessor or multiprocessor. We will consequently configure the SMPCache simulator with a single processor, and we will use uniprocessor traces. For this first set of projects we will consider traces of some SPEC'92 benchmarks (*Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*), according to real tests performed on a

MIPS R2000 system. The traces used represent a wide variety of "real" application programs. These traces come from the Parallel Architecture Research Laboratory (PARL), New Mexico State University (NMSU), and they are available by anonymous ftp to *tracebase.nmsu.edu*. The traces had different formats, like Dinero or PDATS, and they have been changed to the SMPCache trace format (see *Getting Started with SMPCache 2.0*, section 4). These traces, with the correct format for SMPCache, are included in your copy of the simulator. A summary of the traces is given in Table 1.

Name	Classification	Language	Comments
Hydro	Floating point	---	Astrophysics: Hydrodynamic Naiver Stokes equations
Nasa7	Floating point	Fortran	A collection of 7 kernels. For each kernel, the program generates its own input data, performs the kernel and compares the result against an expected result
Cexp	Integer	C	Portion of a Gnu C compiler that exhibits strong random behaviour
Mdljd	Floating point	Fortran	Solves the equations of motion for a model of 500 atoms interacting through the idealized Lennard-Jones potential. It is a numerical program that exhibits mixed looping and random behaviour
Ear	Floating point	---	This trace, the same as the rest, was provided by Nadeem Malik of IBM
Comp	Integer	C	Uses Lempel-Ziv coding for data compression. Compresses an 1 MB file 20 times
Wave	Floating point	Fortran	Solves Maxwell's equations and electromagnetic particle equations of motion
Swm	Floating point	Fortran	Solves a system of shallow water equations using finite difference approximations on a 256*256 grid
UComp	Integer	C	The uncompress version of *Comp*

Table 1: Uniprocessor traces.

☞**Remember**: All these uniprocessor projects can be performed in a similar way with multiprocessor traces.

2.1. Project 1: Locality of Different Programs

Purpose

Show that the programs have different locality, and there are programs with "good" or "bad" locality.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Blocks in cache = 128 (cache size = 4 KB).

- Mapping = Fully-Associative.
- Replacement policy = LRU.

Obtain the miss rate using the memory traces: *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp* (trace files with the same name and extension ".prg").

Do all the programs have the same locality grade? Which is the program with the best locality? And which does it have the worst? Do you think that the design of memory systems that exploit the locality of certain kind of programs (which will be the most common in a system) can increase the system performance? Why?

During the development of the experiments, you can observe graphically how, in general, the miss rate decreases as the execution of the program goes forward. Why? Which is the reason?

2.2. Project 2: Influence of the Cache Size

Purpose

Show the influence of the cache size on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the blocks in cache using the following configurations: 1 (cache size = 0,03 KB), 2, 4, 8, 16, 32, 64, 128, 256, and 512 (cache size = 16 KB). For each of the configurations, obtain the miss rate using the trace files (extension ".prg"): *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

Does the 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 cache? 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?

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

2.3. Project 3: Influence of the Block Size

Purpose

Study the influence of the block size on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Main memory size = 256 KB (the number of blocks in main memory will vary).
- Cache size = 4 KB (the number of blocks in cache will vary).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the words by block using the following configurations: 4 (block size = 8 bytes), 8, 16, 32, 64, 128, 256, 512, and 1024 (block size = 2048 bytes). For each of the configurations, obtain the miss rate using the trace files: *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

Does the miss rate increase or decrease as the block 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 compulsory misses when you enlarge the block size? What is the pollution point? Does it appear in these experiments?

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

2.4. Project 4: Influence of the Block Size for Different Cache Sizes

Purpose

Show the influence of the block size on the miss rate, but in this case, for several cache sizes.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 32.
- Main memory size = 1024 KB (the number of blocks in main memory will vary).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the words by block using the following configurations: 8 (block size = 32 bytes), 16, 32, 64, 128, 256, 512, and 1024 (block size = 4096 bytes). For each of the configurations of words by block, configure the number of blocks in cache in order to get the following cache sizes: 4 KB, 8 KB, 16 KB, and 32 KB. For each configuration obtain the miss rate using the memory trace: *Ear*.

We are first going to ask you the same questions as in the previous project: Does the miss rate increase or decrease as the block size increases? Why? What does it happen with the compulsory misses when you enlarge the block size? Does the pollution point appear in these experiments?

Does the influence of the pollution point increase or decrease as the cache size increases? Why?

2.5. Project 5: Influence of the Mapping for Different Cache Sizes

Purpose

Analyse the influence of the mapping on the miss rate for several cache sizes.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 32.
- Words by block = 64 (block size = 256 bytes).
- Blocks in main memory = 4096 (main memory size = 1024 KB).
- Replacement policy = LRU.

Configure the mapping using the following configurations: Direct, two-way set associative, four-way set associative, eight-way set associative, and fully-associative (remember: Number_of_ways = Number_of_blocks_in_cache / Number_of_cache_sets). For each of the configurations of mapping, configure the number of blocks in cache in order to get the following cache sizes: 4 KB (16 blocks in cache), 8 KB, 16 KB, and 32 KB (128 blocks in cache). For each configuration obtain the miss rate using the memory trace: *Ear*.

Does the miss rate increase or decrease as the associativity increases? Why? What does it happen with the conflict misses when you enlarge the associativity grade?

Does the influence of the associativity grade increase or decrease as the cache size increases? Why?

In conclusion, does the increase of associativity improve the system performance? If the answer is yes, in general, which is the step with more benefits: from direct to 2-way, from 2-way to 4-way, from 4-way to 8-way, or from 8-way to fully-associative?

2.6. Project 6: Influence of the Replacement Policy

Purpose

Show the influence of the replacement policy on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Blocks in cache = 128 (cache size = 4 KB).
- Mapping = 8-way set-associative (cache sets = 16).

Configure the replacement policy using the following configurations: Random, LRU, LFU, and FIFO. For each of the configurations, obtain the miss rate using the trace files (extension ".prg"): *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

In general, which is the replacement policy with the best miss rate? And which does it have the worst? Do the benefits of LFU and FIFO policies happen for all the benchmarks or do they depend on the different locality grades?

For a direct-mapped cache, would you expect the results for the different replacement policies to be different? Why or why not?

In conclusion, does the use of a concrete replacement policy improve the system performance? If the answer is yes, in general, which is the step with more benefits: from Random to LRU, from Random to LFU, or from Random to FIFO? Why (consider the cost/performance aspect)?

3. Multiprocessor Traces

After analysing the basic algorithms and concepts that are present in every cache memory system (uniprocessor or multiprocessor), we study some theoretical issues related with multiprocessor systems. In these projects, we will consequently configure the SMPCache simulator with more than one processor, and we will use multiprocessor traces with tens of millions of memory accesses (references) for four benchmarks (*FFT*, *Simple*, *Speech* and *Weather*). These traces were provided by David Chaiken (then of MIT) for NMSU PARL, and they are available by anonymous ftp to *tracebase.nmsu.edu*. The traces represent several real parallel applications (*FFT*, *Simple* and *Weather* traces were generated using the post-mortem scheme implemented by Mathews Cherian with Kimming So at IBM). The traces had different formats, like the canonical format for multiprocessor traces developed by Anant Agarwal, and they have been changed to the SMPCache trace format (see *Getting Started with SMPCache 2.0*, section 4). These traces, with the correct format for SMPCache, can be obtained at the address http://atc.unex.es/mavega/SMPCache/SMPCacheEnglish.htm or http://atc.unex.es/smpcache. A summary of the traces is shown in Table 2.

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 of cache size improve the multiprocessor system performance? Are the conclusions you obtain similar to the previous ones for the miss rate (project 7)?

3.3. Project 9: Influence of the Cache Coherence Protocol on the Miss Rate

Purpose

Study the influence of the cache coherence protocol on the miss rate during the execution of a parallel program in a symmetric multiprocessor.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 8.
- 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).
- Blocks in cache = 256 (cache size = 16 KB).
- Mapping = Set-Associative.
- Cache sets = 64 (four-way set associative caches).
- Replacement policy = LRU.

Configure the cache coherence protocol using the following configurations: MSI, MESI, and DRAGON. For each of the configurations, obtain the global miss rate for the system using the memory traces: *FFT*, *Simple*, *Speech* and *Weather*.

Do all the protocols have the same miss rate? Which is the coherence protocol with the best miss rate? And which does it have the worst? In particular, is the miss rate the same for the MSI and MESI protocols? Why?

Do you observe any difference between the update-based protocol and the invalidation-based protocols? Which? Why? Are the coherence misses the same for these two kinds of protocols?

Do you think that the results and conclusions obtained with these experiments are of general application or they may change depending on the used benchmarks?

In conclusion, does the use of a concrete cache coherence protocol improve the multiprocessor system performance? Why?

3.4. Project 10: Influence of the Cache Coherence Protocol on the Bus Traffic

Purpose

Analyse the influence of the cache coherence protocol 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.
- 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).
- Blocks in cache = 256 (cache size = 16 KB).
- Mapping = Set-Associative.
- Cache sets = 64 (four-way set associative caches).
- Replacement policy = LRU.

Configure the cache coherence protocol using the following configurations: MSI, MESI, and DRAGON. 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).

Do all the protocols have the same bus traffic? Which is the coherence protocol with the best bus traffic? And which does it have the worst? In particular, is the bus traffic the same for the MSI and MESI protocols? Why (for this answer, remember how the miss rate was for these two protocols –project 9)?

Do you observe any difference between the update-based protocol and the invalidation-based protocols? Which? Why (give at least two reasons)?

Do you think that the results and conclusions obtained with these experiments are of general application or they may change depending on the used benchmarks? Indicate other scenario in which the invalidation protocol does much better than the update protocol.

In conclusion, does the use of a concrete cache coherence protocol improve the multiprocessor system performance? Why? Are the conclusions you obtain similar to the previous ones for the miss rate (project 9)?

3.5. Project 11: Influence of the Number of Processors on the Miss Rate

Purpose

Study the influence of the number of processors on the miss rate during the execution of a parallel program in a symmetric multiprocessor.

Development

Configure a system with the following architectural characteristics:

- 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).
- Blocks in cache = 256 (cache size = 16 KB).
- Mapping = Set-Associative.
- Cache sets = 64 (four-way set associative caches).
- Replacement policy = LRU.

Configure the number of processors in the SMP using the following configurations: 1, 2, 4, and 8. For each of the configurations, obtain the global miss rate for the system using the three traces that were generated by the post-mortem scheme: *FFT*, *Simple* and *Weather*.

Does the global miss rate increase or decrease as the number of processors increases? Why? Does this increment or decrement happen for all the benchmarks or does it depend on the different locality grades, shared data,...? What does it happen with the capacity and coherence misses when you enlarge the number of processors? Are there conflict misses in these experiments? Why?

Do you think that the results and conclusions obtained with these experiments are of general application or they may change depending on the cache coherence protocol? Why?

In conclusion, does the increase of the number of processors improve the multiprocessor system performance? Why and in what sense?

3.6. Project 12: Influence of the Number of Processors on the Bus Traffic

Purpose

Evaluate the influence of the number of processors on the bus traffic during the execution of a parallel program in a symmetric multiprocessor.

Development

Configure a system with the following architectural characteristics:

- 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).
- Blocks in cache = 256 (cache size = 16 KB).
- Mapping = Set-Associative.
- Cache sets = 64 (four-way set associative caches).
- Replacement policy = LRU.

Configure the number of processors in the SMP using the following configurations: 1, 2, 4, and 8. For each of the configurations, obtain the bus traffic (in bytes per memory access) for the system using the three traces that were generated by the post-mortem scheme: *FFT*, *Simple* and *Weather*. Remember: 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 number of processors 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 or does it depend on the different locality grades, shared data,...?

In general, which is the step with more bus traffic: from 1 to 2 processors, from 2 to 4 processors, or from 4 to 8 processors? Why?

Do you think that the results and conclusions obtained with these experiments are of general application or they may change depending on the cache coherence protocol? Why?

Change the results obtained with these experiments in order to measure the bus traffic in bytes per instruction (you must divide by the number of instruction captures -not memory accesses- in the traces). Supposing that these applications run at a sustained 100 MIPS per processor (and they are integer-intensive applications), what is the address and data bus bandwidth requirement for each application and configuration of the number of processors?

The computation in the question above gives the average bandwidth requirement under the assumption that the bus bandwidth is enough to allow the processors to execute at full speed. This computation provides a useful basis for sizing the system. For example, on a machine with 4.8 GB/s of data bandwidth, how many processors will the bus be able to support without saturating for each benchmark? Remember: If the bandwidth is not sufficient to support the application, the application will slow down.

In conclusion, does the increase of the number of processors improve the multiprocessor system performance? Why and in what sense? Are the conclusions you obtain similar to the previous ones for the miss rate (project 11)?