**CSCI 6461 Computer Architecture II Fall 2022**

Project 1 (100 pts) 09/27/22

Due 10/15/2022

This project is to help you get familiar with the software SMPCache (<http://arco.unex.es/smpcache/>). The following tasks provide a powerful tool for understanding the complex design features of a modern computer system.

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 tasks 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

**Task table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tasks | Purpose | Configuration | Experiment | Questions |
| 1. Locality of Different Programs | Show that the programs have different locality, and there are programs with “good” or “bad” locality. | 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. Influence of the Cache Size | Show the influence of the cache size on the miss rate. | 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? |
| 3. Influence of the Block Size | Study the influence of the block size on the miss rate. | 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? |
| 4. Influence of the Block Size for Different Cache Sizes | Show the influence of the block size on the miss rate, but in this case, for several cache sizes. | 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? |
| 5. Influence of the Mapping for Different Cache Sizes | Analyse the influence of the mapping on the miss rate for several cache sizes. | 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? |
| 6. Influence of the Replacement Policy | Show the influence of the replacement policy on the miss rate. | 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)? |