ECE669: Parallel Computer Architecture
Fall 2004 **Handout #5**

Homework # 5 **Due: November 17**

Cache Systems

In this laboratory you will familiarize yourself with the cache and directory simulator, and learn about design tradeoffs in cache coherence schemes for scalable multiprocessors. This lab will lead you through the following sequence of experiments:

1. Familiarization with the cache and directory simulators and trace driven simulation. You will refer to the cache and directory simulator appendix (Appendix A) for information on using the various simulators and data manipulation scripts.

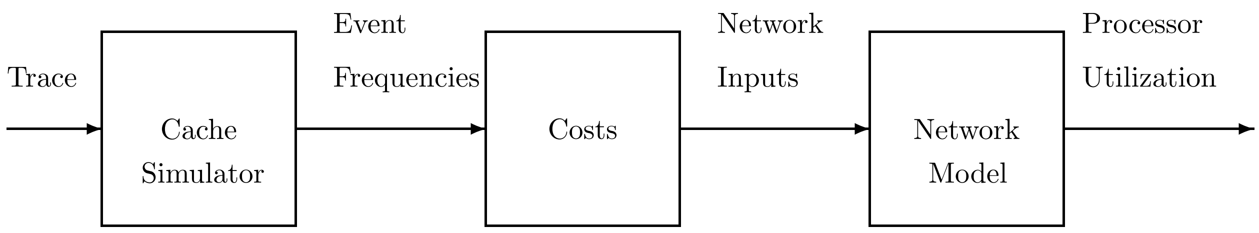
2. Obtaining event frequencies such as cache miss rates and invalidation rates through cache simulations. Comparing the various event rates for the different cache coherence protocols and cache parameters. This comparison should yield a qualitative understanding of cache behavior in multiprocessors. These event frequencies will help determine overall system performance.

   Overall system performance is given by the effective processor utilization, which is computed in the following two steps.

3. Deriving the network traffic parameters. Combine the cache event statistics obtained through simulations with their associated costs (such as block size) and derive the network input parameters: message probability on a given cycle, and average message size.

4. Deriving the processor utilization, by plugging the message probability and message size parameters into the network model.

These steps are depicted in the following figure.



1 Familiarization with the Trace

The trace is a sequence of records. Each record consists of an address and two additional pieces of information: the processor number and the address type. Let us denote a record as a triple *(cpunum, type, address)*. The processor number specifies the processor that made this request, and the address type (e.g., *rl*, *wl*) denotes, for instance, whether the reference was a read or write. To see what the beginning of the trace looks like type:

```
unix-prompt% vsee 200 < brief.ref | more
```

The line "2 wl 48443c" corresponds to a write by processor # 2 to memory address 48443c. Convince yourselves that the trace represents 64 processors.

The simulator program reads records from the trace file sequentially, simulating the behavior of the caches and directories for each input address. For example, on reading the record (2, wl, 48443c) from the trace, the simulator executes a processor write to address 48443c in processor 2's cache.

Q 1: If the j^{th} entry in the trace is the record (2, rl, 59778), what record in the $j+1^{th}$ position in the trace will result in an invalidation.

2 Cache and Directory Simulation

Simulate (using *dirsim*) a full-map directory with 64 processors, 64K-byte caches, and cache block size of 16 bytes. Condense the large output file produced using the event-dirn.awk file. Refer to Appendix A for directions on running the directory simulator, *dirsim*, and the awk programs.

Refer to the condensed output produced by eventdirn.awk and verify that you can explain each of the components of listed in the event column. You might find the hints provided in Section 6 of this handout useful in interpreting the output.

Simulate single link chain and double link chain directory schemes with 64 processors, 64K-byte cache, and cache block size 16 bytes and condense the large output files using the awk scripts eventschain.awk and eventdchain.awk respectively. Also simulate a four-pointer limited directory scheme with 64 processors, 64K-byte cache, and cache block size 16 bytes and condense the output file using the eventdirn.awk script.

Q 2: Compare the outputs of the these schemes with the full map scheme. Suggest as many reasons as you can explaining why the event frequencies for the various schemes differ. For example, why is the read miss number for single link chain higher than the corresponding numbers for the other schemes? Why does the limited directory scheme have a higher invalidation rate? (The "Other Messages" category largely comprises invalidations).

Q 3: Contrast the memory requirements required to house the pointers as a function of the number of processing nodes N for the four-pointer limited directory scheme, the full-map scheme, the single link chain, and the double link chain schemes. Include the pointer space associated with both main memory and the cache. Assume that each processing node has M blocks of memory (each block is 16 bytes) and a cache of size C blocks.

3 Deriving Network Input Parameters

Now, use the event frequencies for the four schemes from the condensed output files to derive the input parameters to feed the network model. Note that the necessary formulae are provided at the end of each condensed output file.

Q 4: Compute the network request probability (m) for each scheme by weighting the frequencies of the individual events by the number of messages generated by each, and summing them up.

Q 5: Similarly compute the average message size (B) by weighting the frequencies of the individual events by the number of flits transferred for each event and adding them up. Assume that flits are 16 bits wide.

Q 6: The network request rate and the message size are themselves a good indicator of the relative program execution times with the various caching schemes. Can you predict the relative performance of the four schemes using m and B? Clearly state how you used these two values in estimating the relative performance.

4 Deriving Processor Utilization

The processor utilization can be derived by plugging the network request rate and message size into the network model discussed in class.

We will use the following notation:

U	processor utilization
T	network latency
m	network request rate
B	message size
ρ	network channel utilization
k	network radix
k_d	distance traveled in a dimension ($\frac{k-1}{2}$)
n	network dimension
M	memory latency (assume $M = 0$)
N	number of processors

For a k-ary n-cube the processor utilization can be derived as shown below. For more details refer to the handout for the interconnection network lab. The model shown here assumes unidirectional channels. (For all your computations assume $M = 0$ for simplicity.)

$$Channel\ Utilization\ \rho = mB\frac{k}{2} = mBk_d$$

$$Network\ latency\ T = \frac{T_0}{2} + \frac{Bk_d}{4} - \frac{1}{2m} + \frac{1}{2}\sqrt{\left(T_0 - \frac{Bk_d}{2} + \frac{1}{m}\right)^2 + 2B^2(k_d - 1)(n+1)}$$

where $T_0 = nk_d + M + B$ is the unloaded network latency. The corresponding utilization is computed as:

$$Processor\ utilization\ U = \frac{1}{1 + mT}$$

Q 7: Compute the processor utilization for the four coherence schemes for a two dimensional network of 64 processors.

5 Comparison Shopping

Now that you are familiar with the process of simulating a cache coherence strategy, and obtaining resulting processor utilization using the network model, let us evaluate the effect of varying several system parameters.

Q 8: What is the performance of a full-map scheme for 16K-byte caches? How does it compare to the performance of the 64K byte cache from your previous simulations? Do you expect this result to change with a longer trace?

The cache miss rate in multiprocessors has the following components:

1. Cache misses caused due to first time accesses.
2. Cache interference caused when one block replaces another in the cache.
3. Cache misses caused due to invalidations resulting from coherence transactions.

Q 9: How do each of these three components change with cache size? Using the variation in the cache miss rate for 16K-byte and 64K-byte caches to guide your intuition, can you say which of the above components might be dominant as multiprocessor caches become very large?

Q 10: How do you expect the above three components to change as we increase the cache block size, for the same total cache size?

Q 11: How do you expect the above three components to change as we increase the problem size, keeping the number of processors and other machine parameters constant? Assume the problem on which you are working is the ubiqitous Jacobi relaxation.

6 Awk Output Files

These hints are meant to help you interpret results obtained using the directory-based cache simulator and the awk filters. Below is a brief discussion of the various cache-access cases in which network traffic caseoccurs.

1. Read Miss in Write Mode – This case occurs when a processor attempts to read a value from its cache, a cache miss occurs, and the sole owner of a dirty copy of the data is a cache at another processor in the system. This case will require two network messages: one to request the data and another to send the data to the requesting cache.

   This case actually requires two additional messages: one message to ask the processor with the dirty copy to write it back, and the second message corresonding to the writeback itself. However, for simplicy we report these writeback messages in the "Other Messages category.

2. Read Miss Not in Write Mode – This case occurs when a processor attempts to read a value from its cache, a cache miss occurs, and zero or more caches in the system contain clean copies of the data. This case will require two network messages: one to request the data and another to send the data to the requesting cache.

3. Write Miss in Write Mode – This case occurs when a cache attempts to write a value in its cache but the sole owner of a dirty copy of the data is a cache at another processor. This case will require two messages to request and receive ownership of the data.

   This case requires two additional messages to maintain cache coherence: one to invalidate the current cache owner and a second to receive an acknowledgement. These additional invalidation messages have been lumped into the "Other Messages" section of the results and *not* into the statistics for this case.

4. Write Miss Not in Write Mode – This case occurs when a cache attempts to write into a block, and zero or more other caches contain read-only copies of that block. This case will require two messages to acquire ownership of the data.

   This case also requires additional messages to invalidate the copies in the other caches. Once again invalidation is included in "Other Messages."

5. Write Hit Not in Write Mode – This case occurs when a cache attempts to write to a block for which it has read-only access, and other caches potentially have read-only copies of the block as well. This case will involve two messages to receive ownership of the data.

   This case also requires a number of invalidation messages for other data copies. Once again invalidation is included in "Other Messages."

Each event reported in the "Other Messages" category requires two messages.

The message rate for the protocol may be determined by summing up the number of messages in each case multiplied by the probability of that case occuring. There should six terms to add; one for each of the six rows (or cases) shown in the awk output file.

Average message size may be determined by summing message sizes multiplied by probability of occurance for each case outlined above and then dividing by the message rate.

A Using the Directory Simulator

A.1 General Information

This section documents data and programs to be used in cache and directory simulations.

The 64 processor address trace is available in `brief64.ref`. Each record in the trace contains the identifier of one of 64 processors, a type code (shared read, shared write, test-and-set, ...), and the address associated with this code. The trace was gathered using *T-Mul-T* (written by David Kranz) on a Speech recognition program (Viterbi search algorithm) written by Kirk Johnson, both of MIT.

One of the more difficult (and tedious) problems with trace-driven simulation involves manipulating very large quantities of data. Even though this trace is relatively short compared to other multiprocessor address traces, it is still 10 Mbytes long. Since this file is so long, do not copy it to other directories or to other file systems.

A.2 Command-Line Interface to Directory Simulators

The directory simulator has a command-line interface. Use `dirsim` to run simulations. The parameters for simulations are specified via command line switches as follows:

- **-s** l Number of lines (blocks) in each processor's cache.
- **-b** b Size of each cache block. This is the unit of cache coherence.
- **-c** n_p Number of processors. Use 64 for this parameter.
- **-d** s Coherence directory scheme. Full-map = 64. Limited = 1, 2, 3, or 4.
  Single link chain = s. Double link chain = d.
- **-f <filename>** Output file. Default output file is statZZZ.out.

The directory simulator output must be post-processed to calculate the probabilities of different types of cache events. The simulator output is post-processed with the `awk` program, a `UNIX` hack that was written by Aho, Kernighan, and Weinberger. `awk` reads a script file (written by the authors of the directory simulator) and an output file from the directory simulator. There is one script file for each type of directory protocol. `awk` outputs event probabilities and other information that may be used to calculate the system's average network request rate and average network message size.

Watch out! The directory simulator appends to the file specified with the `-f` parameter (instead of just wiping out the file and rewriting it on each run). This feature reduces the chance that data from extremely long simulations will be lost. Before you run `awk`, you might want to make sure that there is only one set of results in the directory simulator output file.

A.2.1 Examples

1. The commands below run the directory simulator for a 64 processor system, with a 64K processor cache (4096 sets, 1 block per set, 16 bytes per block) and a 2 pointer limited directory. Note the link commands (ln -s) which provide a path to input files in the course locker. They should only be executed once per input file in the working directory.

   ```
   prompt% dirsim -s 4096 -b 16 -c 64 -d 2 -f dir2.out <brief64.ref
   ...
   prompt% awk -f eventdirn.awk dir2.out
   ...
   ```

2. The commands below run the directory simulator for a 64 processor system, with a 32K processor cache (2048 sets by 1 block per set by 16 bytes per block) and a single link chain directory.

   ```
   prompt% dirsim -s 2048 -b 16 -c 64 -d s -f schain.out <brief64.ref
   ...
   prompt% awk -f eventschain.awk schain.out
   ...
   ```

A.3 The `vsee` Program

The `vsee` program may be used to display trace data in human-readable format. Each record of the trace (as described) is printed to stdout. `vsee` requires one parameter: the number of records to be listed. The processor identifiers and the addresses are printed in hexadecimal, and the opcodes are printed according to the following scheme:

`r` = read; `m` = modify; `im` = interlocked modify; `w` = write; `fa` = fetch and add (usually signifies atomic synchronization operation)

`b` = byte; `w` = word; `l` = long; `p` = private;

The above are combined to indicate operation types; e.g., `wp` = write a word to private memory.

The other interesting instruction is `ifetch` = instruction fetch

The `filler` opcode indicates information from the tracing mechanism that does not fit into this format, but may be useful in some other context.

Operations named by their hex code are not currently in use.

A.3.1 Example

The command below lists 100 records of the brief64 trace to stdout.

```
prompt% vsee 100 < brief64.ref
```

A.4 Files

brief64.ref:	truncated speech trace
dirsim:	directory simulator
eventdir.awk:	`awk` script for reading dirsim output for limited/full map
eventdchain.awk:	`awk` script for reading dirsim output for double link chain
eventschain.awk:	`awk` script for reading dirsim output for single link chain
vsee:	program used to display trace data