Assignment 3: Cache and memory

November 2006

Introduction

In Chapter 5 of Hennessy & Patterson (3ed), dealing with memory hierarchies, several techniques are described to speed up the access of 'lower-level' memory (main memory). Examples include making the memory (and bus) wider, using interleaved memory, and using specific properties of DRAM.

In this assignment, we investigate what the effect of such techniques is in a number of realistic situations. For this purpose, we use the `Dinero` cache simulator. `Dinero` is a program that simulates the behaviour of all sorts of caches. The input consists of traces, i.e. a list of all memory references generated by a specific application (e.g. `gcc`) during its execution. Output is presented at various levels of details. We use option `-o2` to get 'bus output', which means that `Dinero` generates a list of all reads and writes from the cache to the underlying memory level, as well as some information about the number of accesses, hit percentage, etc.

In the `Dinero man` page you will find how the program works in more detail and how different caches can be simulated by using different options.

Assignment

The assignment is as follows: determine which of the given combinations of cache- and memory-types (15 in total) give the best results in terms of bandwidth and bank conflicts. You should do this by way of one or more C/C++ programs that use the `Dinero` output as input.

Two traces will be used for input for `Dinero`, both from popular performance benchmarks.. Het gaat om `lisp.002.din` and `spic.002.din`, which can be found in `/usr/local/edu/data/ca.opg4`. Note that these are large files. If you want to work at home, you may want to use the trace in `testfile` for testing purpose, as this is a much smaller file. However, eventually you *should* run your code with the large traces at the university.

Definitions

We use the following definitions:

The *bandwidth* of a cache/memory-system is *the total number of bytes sent between CPU and cache divided by the number of cycles.* We assume here that memory activity is the bottleneck for the entire system – in other words: that the cache is continuously busy processing requests.

A *bankconflict* is defined as follows:

For normal memory, a bankconflict is *a request that is sent to a bank that is still busy handling the previous request (either in the access phase or in the bus transfer phase).* The first memory access incurs no conflict.

In page-mode DRAM we do not use multiple banks, so the above definition is not so useful[1]. In this case, interpret the following as a bankconflict: *a request for a page (=column) in which the previous request was not for the same page.* The first access here also does not incur a conflict

Configurations

We use the following 5 memory configurations:

- standard memory with random access time of 8 clock cycles
- 4-bank word-interleaved memory with random access time of 8 clock cycles
- 8-bank word-interleaved memory with random access time of 8 clock cycles
- page-mode DRAM with a page-size of 64 words, a random access time of 8 clock cycles and a 'next access time' of 3 clock cycles.
- page-mode DRAM with a page-size of 1024 words, a random access time of 8 clock cycles and a 'next access time' of 2 clock cycles.

To clarify this even more: in page-mode DRAM with 64-word pages, adresses 0,4,8,...,FC (hex) are located on page 0, addresses 100, 104, ..., 1FC on page 1, etc. You do not need to take refreshing into consideration. See also *Computer Architecture: a Quantitative Approach, 2nd ed.* (the relevant pages are scanned and available from the assignments webpage).

Cache configurations

We use the following 3 cache configurations:

- no cache, a write buffer of 1 word deep
- a 64 KB, unified, direct-mapped, write-through, no write-allocate cache with 4 word blocks and a 1 word write buffer
- a 64 KB, unified, direct-mapped, write-back, write-allocate cache with 4 word blocks and a 1 word write buffer

[1] For information about page-mode memory see, for instance, the scanned pages from H&P, 2nd ed., that are available from the assignments web page.

Handling requests and other assumptions

All addresses in this assignment are word-aligned and all data-accesses to the cache are 1 word.

Per clock cycle, 1 request for 1 word can be handled. The banks can be used in parallel (the pages of the page-mode DRAM obviously cannot), but requests should remain *in order*. In other words, a request for a bank that is busy will block all subsequent requests. If a request going from the cache to memory concerns more than 1 word, this will be transformed by the cache logic in multiple requests for 1 word.

The time for submitting the requests does not have to be taken into consideration; access time is measured from the start of the request. A bus transfer takes 1 cycle. During both access time and bus transfer of a specific bank, no new request for this bank can be submitted.

There are many other design decisions (and assumptions) you can make in this assignment. Depending on your assumptions, results may be slightly different. So make all of your assumptions explicit and argue why you made them. This is important for the grading!

Output format of `Dinero`

As `Dinero`'s man page is not very elaborate, we will briefly talk about `Dinero`'s bus-output style. We are interested in the transfers and communication between CPU and cache. This is recorded in a BUS2-record by the *reference count* variable. This value is the number of 'demand references' (request for the cache) since the last 'bus transfer' (i.e. cache miss). According to the extra information in the file `doc.h` the current reference (the cache request that caused a cache miss and the current memory request) is included in this count. A write-back cache may require multiple memory requests to handle a single cache miss; however, the reference that led to the miss is only counted once and so the corresponding BUS2-records have a reference count of 0.

Submitting

a. Answer the following question: what influence has the assumption that cache/memory is the bottleneck of the system on the results (i.e. bandwidth and bank conflicts)?

b. Submit a number of *well-documented* C/C++ programs that calculate the number of bank conflicts and the memory bandwidth, for all combinations of memory and cache (15 in total)[2].

   Also provide a *shell-script* that calls both the cache simulator and your programs (for both traces), and writes the results in files. We prefer to receive all results in one file, but with clear indicators of which results belong to which configuration and which also mentions all `Dinero` options used. Example:

```
dinero -i16k -d16384 -b16 <
          /usr/local/edu/data/ca.opg4/lisp.002.din >tmp1
compu 1 <tmp1 >res1.1
compu 2 <tmp1 >res1.2
```

[2] A single program will do, although it may be handy to write so-called filters and have multiple programs cooperate in a so-called *pipe*.

```
compu 3 <tmp1 >res1.3
compu 4 <tmp1 >res1.4
compu 5 <tmp1 >res1.5
..
..
etc.
```

Submite all of the following: C/C++ programs, header files, Makefiles, shell script, program output. Submit this both electronically an on paper (listing). The executable should not be submitted (they should be easy to make). Do not include compiling commands in the shell script, use the Makefile for this instead.

c. Make a well-written presentation about your results in which you argue for the memory and cache configuration of your preference. For this purpose, also take costs into consideration.