EAC Practical 1a (Intro to Simplescalar and Wattch)

Deadline: 4pm 14 October 2010

Worth: 5% of course marks

Introduction

The purpose of this practical is to give you a hands-on introduction to Simplescalar and Wattch simulator, which will allow you to make a head-start in your course project later on.

Your task is to model a filter cache, by changing the code in the simulator and to evaluate the performance and power/energy of a processor equipped with a filter cache by simulating benchmarks and interpreting the results.

The filter cache was described in MICRO'97 (DOI:10.1109/MICRO.1997.645809), by Kin *et al.* It is a tiny cache placed ahead of the standard, level 1 D-cache with the intention of providing a large proportion of memory references with a lower energy cost than the standard D-cache. It will be covered in more detail later in the course and you shouldn't need to read the above paper to do this coursework; basic knowledge of caches from previous courses will suffice.

The following section provides detailed guidance on what you have to do.

You will need to submit the *modified source code files* electronically before the deadline. Do not send any files that you did not edit or any object files. It is best to make an archive of the files before submitting them. The command you use should look like (use the correct class, course name for you): `submit cs4|msc eac-4|eac-5|eac-4-v 1a <file1.tar>`

The assessment of this practical will be based mainly on the completeness of the model (both performance and power must be modelled) and how modular your approach is: filter-cache should be an option and the simulator should work as usual when it is disabled in the configuration file.

This practical is to be done individually. Please bear in mind the School of Informatics guidelines on plagiarism.

Details

You will need to make changes to files: `sim-outorder.c` `power.h` `power.c`

For `sim-outorder.c` you need to do the following:

- Declare variables for the filter cache itself, its configuration options and latency. Use the existing D-cache as a guide. Search for `cache_dl1_opt, cache_dl1_lat, cache_dl1`.

- Find function `sim_reg_options` and "register" options and latency for the filter cache. Again use the code for the D-cache as a guide. Note that the filter cache is small and highly associative, so use an appropriate default configuration, such as 1 set, associativity 32 and block size equal to that of the D-cache.

- Find function `sim_check_options` and create a filter cache using the code for D-cache as a guide. Consider that filter cache may be disabled in the configuration file and the processor will still have to work in this case.

- Find function `sim_reg_stats` and "register" the statistics for the filter cache.

- There are two macros `__READ_CACHE, __WRITE_CACHE` which access the D-cache in the fast-forwarding mode of the simulator if `ACCESS_CACHE_WHEN_FASTFWD` is defined. If you want to be thorough you can change these, but you do not have to as this option is not used and changing macros is tricky if you are not familiar with C.

- Find function `sim_main` and `clear_cache_stats` for filter cache before detailed simulation starts.

- Find function `ruu_commit` and look where/how the D-cache is accessed (for stores only). Change the code so that filter cache is accessed instead, if it is configured. At this point you need to decide which variable to use for counting filter cache accesses for Wattch (you can declare it later).

- Find function `ruu_issue` and do as above. This is for load accesses.

- Now it is time to consider what happens on filter cache misses. When you "created" the filter cache, you had to provide an access function (`access_fn`). This is called on a miss (details in function `cache_access` in `cache.c`) and you have to write it.

  Find `dl1_access_fn` and use it to create your filter cache access function. Make sure you count the number of D-cache accesses for Wattch.

- Finally declare a counter for filter accesses (your new variable mentioned earlier) for Wattch.

For `power.h` find the structure that keeps the power of all units and add declarations for filter cache power and decoder, wordline, etc. Use D-cache (search for `dcache`) as a guide.

For power.c you need to do the following:

- Declare the filter cache variable defined in `sim-outorder.c` as extern. Do the same for the counter of filter accesses. In addition, declare variables for `total` and `max` filter accesses. Also declare variables for `filter_power` and cc1-3.

- Find function `clear_access_stats` and clear the counter for filter accesses.

- Find function `update_power_stats` and using `dcache_access` as a guide write code to calculate filter power and other filter-related statistics. You also need to add filter power to the various power "totals". Don't forget that there are a number of `_cc` variables which need to be changed too.

  At this point you need to make a decision as to which cache the D-TLB power will be added to. Note that the D-TLB is accessed for every (data) memory access. Once you've decided make sure that D-TLB power is not added twice and be consistent by searching for all other places where it is referenced.

- Find function `power_reg_stats` and "register" the statistics for filter cache power. Search for `dcache` and do what is needed for filter cache. Note that some statistics need to be defined and others need to be changed to use filter-related statistics. There is quite a lot of editing to be done here.

- Find function `dump_power_stats` and using `dcache_power` as a guide, do what is needed for filter power.

- Find function `calculate_power` and modify it to calculate power for the various parts of filter cache and the total power per access. Use the D-cache as a guide, but beware that it is not very clear where the code for D-cache starts and ends. Later in the same function scale your filter power variables to account for short-circuit power as it is done for the others.

  If you code this part similar to the D-cache, you may encounter problems when you run this part of the code, which calls CACTI. CACTI cannot handle very small caches; it will give some error messages (look at the output file in detail), but will not exit the simulator and the power of the unit will not be accurate. I propose to use a CAM-like structure to model the filter cache, similar to how the D-TLB is modelled. You will need to change the D-TLB code substantially though to get it right. Also if you use a CAM for filter cache, the decoder, wordline etc will be undefined and you will need to remove them from your code.

Evaluation

In order to get more realistic results, you will need to run longer simulations. Skip the first billion instructions and simulate for 100 million instructions. Each simulation takes around 5 minutes on my DICE machine.

Run three experiments (per benchmark):

- using the original code,

- your modified version with filter cache disabled,

- your modified version with filter cache

There should be minimal differences in the results of the first two. Ideally they should be identical, but there's some randomness in simplescalar, so small differences are possible.

Is filter cache saving power? How is the speed affected? Does it save energy?