Abstract of Course Project: ECE 231 Advance System Software

Anticipatory Paging Algorithm Based on Next-Page Prediction

Jinfeng Liu  jinfengl@uci.edu  ID: 65547013

The objective of this project is to study the behavior of programs and suggest an anticipatory paging algorithm that can effectively reduce the page miss rate.

Demand paging and LRU-approximate page replacement algorithms are commonly accepted by existing systems. These algorithms are based on the “past” knowledge by looking back to the run-time history. On the other hand, the optimum paging scheme should make decisions based on the program behavior in the future. Unfortunately, such optimum algorithm is impossible to achieve. However, we can still try some heuristics to predict the near future of program execution. Therefore, we propose an anticipatory paging scheme based on prediction to next-page(s) of the current page.

Before the performance of paging mechanisms is measured, certain behaviors of program should be studied. As it is commonly known, the access to data in memory (data page) and the execution on instructions in memory (code page) have different behaviors. In general, the program behavior to access data pages is largely dependent on the way that the programmer writes the code. A good example is the question in midterm. Thus, the accesses to data pages are hardly predictable. Prediction-based anticipatory paging may not yield any performance speedup for data pages.

On the other hand, the behavior of accessing code pages is of more regularity. This is due to the locality brought by the sequential nature of the program. If all code within the current page consists of only sequential instructions, the next page is always the following logical code page. In this case, the anticipatory pager can bring the predicted next page into memory at an appropriate time before a page miss can occur. However, there are different types of non-sequential instructions, e.g., branch, jump, procedure call and return. In these cases, the next required page could be one of several specific pages -- we introduce the concept next page set that contains the code page numbers of those pages that could be the next code page to execute. Thus, the goal of anticipatory paging is to select the next code page of the current page from the next page set that is most likely to be “hit” by the actual next page.

We assume part of the program will be executed multiple times. Initially, the next page set of all pages only contains the following code page. This field is changed dynamically by the run-time history. When the same page is executed again, such knowledge will be helpful to predict the future. Each number in the next page set should be associated with a priority. The page with highest priority will be selected as the candidate for next page to bring in. The multiple page numbers and the corresponding priorities can be changed by different criteria adaptively during run-time execution.

If there is no free page available, page replacement takes place. Our next page set model can affect the existing LRU or its approximations. A general heuristic is not to swap out any pages that are among the next page set of the current page. In this approach, even if our prediction is wrong, the actual next page still has some chances to “hit” the existing pages in main memory.

There are a few issues that may be interesting to study. One question is what the appropriate page size can bring our new model maximum efficiency. Consider an example in Fig. 1.

We want the number of candidates in next page set to be as few as possible, thus high-accuracy prediction can be achieved. Assuming page 1, 2, 3, 4 are logically continuous pages. Consider Page 2, there are three possible candidates for its next page, 1, 3 and 4. If we use large page size, Page 1 and Page 2 are combined as a single page as Page A, so are Page 3 and 4 as Page B. Then the next page of Page A is always B, which is
100% predictable. This implies coarse granularity on page size. However, we can also make the page smaller. If we maintain the page size in a fine-grained manner so that each page only contains one branch instruction, e.g., page a, b, and c as shown in the figure, the next page should be either the page that contains the branch target or the following page. In this approach, we can achieve high prediction accuracy as well. Also, for the code pages that have many procedure calls, smaller page size may be desirable, regardless of more overheads. The size of the next page set is another issue. It can be related to the page size.

![Page A and Page B](image)

Fig. 1 Different page size can affect the size of next page set

The second issue is related to pages containing procedures. Since a procedure should return to its caller, the page containing its caller instruction should be assigned a high priority in the next page set. But this will incur a problem. A procedure can be called multiple times in anywhere of the program, this will cause the next page set to blow. A heuristic is to remove the caller’s page number from the set after the procedure returns. Another problem is that we only want to have the caller’s page number in the next page set of pages that contain the return instruction, but such knowledge cannot be known. We have to leave this decision on heuristics.

One more question is whether to contain data pages to the next page set. So far the discussion is based on code pages only, since code pages exhibit more locality. Considering the fact that same code accesses the same set of data, we may also want to bring in the predicted next data page, which is most likely to be accessed by the next code page, as we bring in the predicted next code page. The program behavior should be inspected first before we try any heuristics for data pages.

The study will be performed on SimpleScalar toolset to examine the program behavior of Spec95 Benchmarks. The new paging scheme will be simulated in comparison with demand paging. The performance measurements will only count times of page miss. We assume a perfect anticipatory pager that can always bring the predicted next page into main memory before the page miss can occur.

Keywords: anticipatory paging, data page, code page, prediction, next page set, page miss rate