

Run-Time Locality and Stride Prediction for Small Data Cache Management

Seongwoo Kim and Arun K. Somani
Dependable Computing and Networking Laboratory
Department of Electrical and Computer Engineering,
Iowa State University, Ames, IA 50011
{skim, arun}@iastate.edu

In this study, we present a hardware cache management mechanism for data caches, which intelligently determines the fetch size based on the locality of reference observed at run-time. This scheme targets the cases where static prediction is not possible or has a very low accuracy (Locality varies substantially across and within applications). On a miss in the data cache, a prediction table, which keeps track of data access pattern (temporal and spatial locality information), suggests how many data items to be fetched in what stride and direction. As a result, data items more likely to be used get into the data cache, reducing miss rate, cache pollution, and data fetch bandwidth.

Distinctive handling of temporal locality and spatial locality within each isolated-cache (e.g., data cache) has not been sufficiently exploited in the classic cache architecture research. Our scheme is to explore the potential of this idea. In the literature, several table-based schemes have been proposed for data prefetching and exploiting spatial locality. However, our approach differs from those schemes in the following aspects for better cost-performance benefit.

- We use a single modular cache rather than separate modules for temporal data and spatial data. Physical cache split (multiple modular caches) by locality types has inherent potential of low performance when a single type of locality is dominant in an application. A single modular cache of a minimum line size with decoupled tag can flexibly accommodate both small and large data block at a low cost.
- We use the size of data accessed (byte, halfword, word, and double-word), but not instruction address (IA), in indexing the prediction table. Data type is a part of instruction, giving us a selectivity, and the data of the same type within a small memory block may have the

same locality. In order to use IA, an access to the PC at the back-end of the pipeline is required. Using access data size eliminates this overhead while providing similar selectivity.

- We combine locality and stride information into a single level table. This enables efficient handling of both unit and non-unit stride accesses, in addition to the detection of temporal data. Selective caching based on run-time behavior can deliver a good performance in a range of applications including multimedia programs.

Figure 1 shows the fields of an entry in the prediction table. The combination of *stride* and *count* represents different locality status. For example, *stride* = 0 and *count* = 15 indicate a data item which has high temporal locality, while *stride* = 32 and *count* = 8 correspond to a data segment of a spatial locality and less frequent access with stride of 32. The access data type and the MSB part of effective address (EA) form *tag* field. The LSB part of EA is *addr*. This determines the size of memory block for which a table entry holds locality behavior.

Valid	Tag	Address	Direction	Stride	Count
--------------	------------	----------------	------------------	---------------	--------------

Figure 1: The contents of table entry.

Figure 2 shows the state transition diagram of an entry. Each state can be for data items of either spatial or temporal locality. The entry status is simply changed by *count* value. If *count* = 1, the entry is in *transient* state (T). If *count* \geq 2, it is in *steady* state (S).

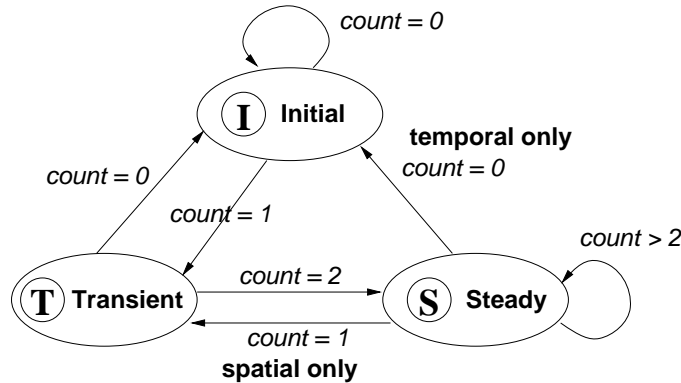


Figure 2: Entry state transition.

We have established detailed algorithms and implemented required simulators. From our simulations, we have so far found that the table is effective in capturing locality behavior. However, optimal parameters such as unit fetch size, widths of *addr*, and *stride* are still need to be found. Most importantly, for given table information, an effective strategy for varying fetch size is critical for the performance, but this is still under investigation.

We have obtained intermediate simulation results. Based on this, we intend to make a judgment whether we are going to further pursue this research or not. The results are shown in the following. Figure 3 shows the relative miss ratio compare to the base d-cache configuration. Our base system is 16KB direct-mapped D-cache with 32B line. Figure 4 shows the relative unused block ratio. The lower ratio, the more intelligent management our LSPT scheme provides. Figure 5 is to present how much data traffic is required by the D-cache. Lower ratio is desirable.

This research has been postponed due to student’s leave of absence for military service. It is scheduled to be completed by Fall, 2001.

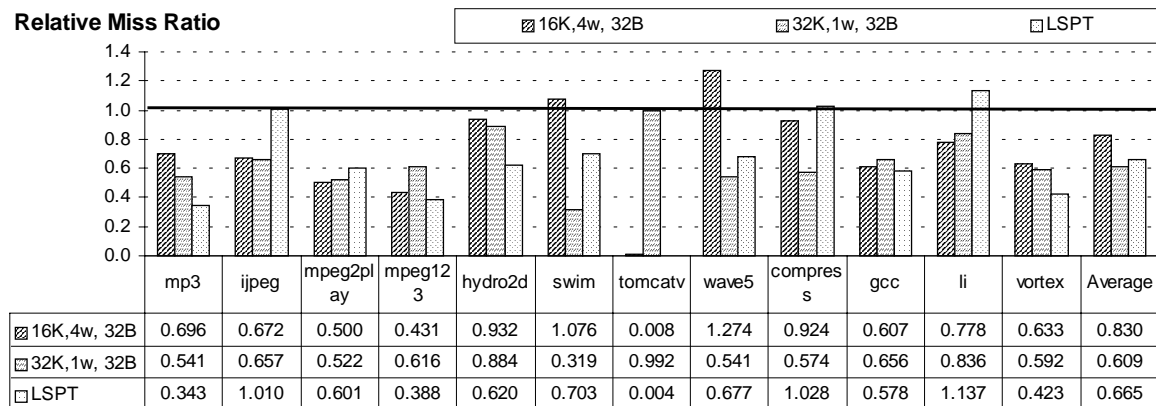


Figure 3: Relative miss ratio in D-cache.

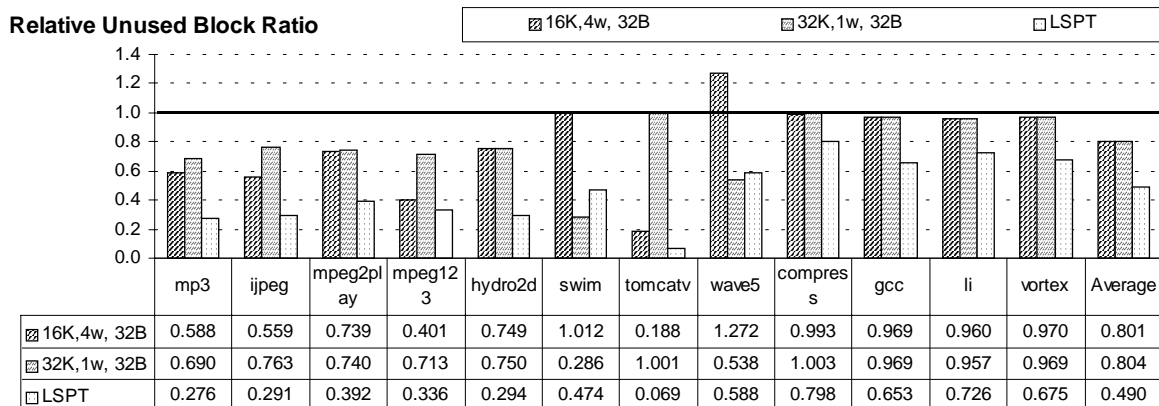


Figure 4: Relative ratio of unused block.

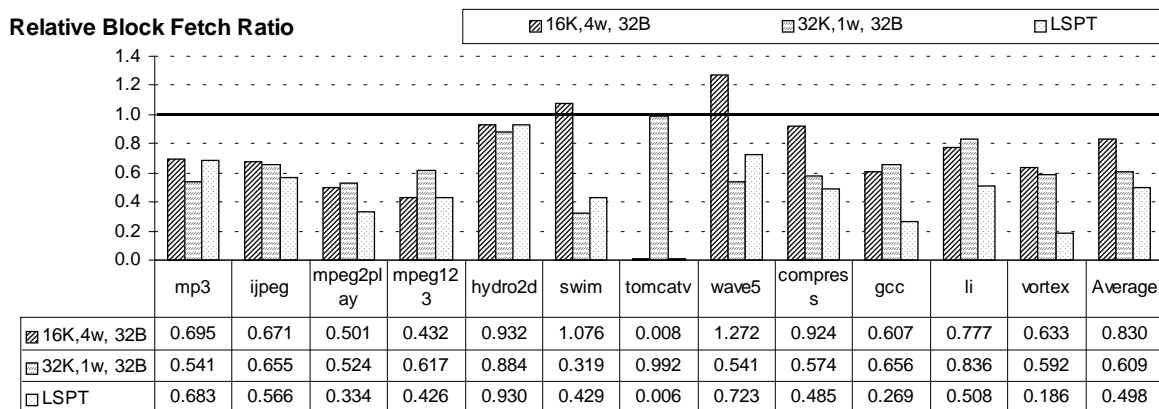


Figure 5: Relative ratio of data block fetched.