



























**Question (8) Exercise- 2.2**

**Assume you are designing a hardware prefetcher for the preceding unblocked matrix transposition code. The simplest type of hardware prefetcher only prefetches sequential cache blocks after a miss. More complicated “nonunit stride” hardware prefetchers can analyze a miss reference stream and detect and prefetch nonunit strides. In contrast, software prefetching can determine nonunit strides as eas- ily as it can determine unit strides. Assume prefetches write directly into the cache and that there is no “pollution” (overwriting data that must be used before the data that are prefetched). For best performance given a nonunit stride prefetcher, in the steady state of the inner loop, how many prefetches must be outstanding at a given time?**

* An alternative to hardware prefetching is for the compiler to insert prefetch instructions to request data before the processor needs it. There are two flavors of prefetch:
* Register prefetch loads the value into a register.
* Cache prefetch loads data only into the cache and not the register.
* Prefetching makes sense only if the processor can proceed while prefetching the data; that is, the caches do not stall but continue to supply instructions and data while waiting for the prefetched data to return. As you would expect, the data cache for such computers is normally nonblocking.
* Like hardware-controlled prefetching, the goal is to overlap execution with the prefetching of data. Loops are the important targets because they lend themselves to prefetch optimizations. If the miss penalty is small, the compiler just unrolls the loop once or twice, and it schedules the prefetches with the execution. If the miss penalty is large, it uses software pipelining or unrolls many times to prefetch data for a future iteration.

for (j 1⁄4 0; j < 100; j 1⁄4 j + 1)

{ prefetch(b[j + 7][0]);

/\* b(j,0) for 7 iterations later

\*/ prefetch(a[0][j + 7]);  
/\* a(0,j) for 7 iterations later \*/

a[0][j] 1⁄4 b[j][0] \* b[j + 1][0];}

This revised code prefetches a[i][7] through a[i][99] and b[7][0] through b[100][0], reducing the number of non prefetched misses

for (i 1⁄4 1; i < 3; i 1⁄4 i + 1)  
for (j 1⁄4 0; j < 100; j 1⁄4 j + 1) {

prefetch(a[i][j + 7]);  
/\* a(i,j) for + 7 iterations \*/

a[i][j] 1⁄4 b[j][0] \* b[j + 1][0];}

**Question (9) Exercise-2.19**  [10/12] <2.3> You have been asked to investigate the relative performance of a banked versus pipelined L1 data cache for a new microprocessor. Assume a 64 KB two-way set associative cache with 64-byte blocks. The pipelined cache would consist of three pipe stages, similar in capacity to the Alpha 21264 data cache. A banked implementation would consist of two 32 KB two-way set associative banks. Use CACTI and assume a 65 nm (0.065 m) technology to answer the fol- lowing questions. The cycle time output in the web version shows at what frequency a cache can operate without any bubbles in the pipeline.

* 1. [10]<2.3>What is the cycle time of the cache in comparison to its access time, and how many pipe stages will the cache take up (to two decimal places)?
  2. [12] <2.3> Compare the area and total dynamic read energy per access of the pipelined design versus the banked design. State which takes up less area and which requires more power, and explain why that might be.