L2 Cache Design for PowerPC™ -Based Systems

Allan R. Steel
IBM Corp.
11400 Burnet Rd.
Austin, TX 78758
asteel@vnet.ibm.com

August 1, 1994

1. Introduction

The PowerPC family of processors are bringing the power of scientific and technical workstations to the PC desktop market. The workstation and PC markets have until now been separated by processor architecture, typical workloads, and price. With the advent of low-cost PowerPC desktop systems come new issues to grapple with when designing or specifying components to be used in these systems. It is the intent of this paper to give some guidance in designing, specifying, or selecting L2 cache subsystems for PowerPC-based computers targeted by the PowerPC Reference Platform Specification. This specification covers a range of systems from laptops to low-end servers, including symmetric multiprocessors (SMP's).

It is worth spending some time contrasting the worlds whose union is enabled by PowerPC. The PC market is dominated by systems built around the Intel X86 processor architecture. Typical workloads are a variety of core desktop applications running under a graphical user interface (GUI) based operating system. This core of applications consists of word processing, spreadsheet, presentation graphics, and database. Recently, multimedia applications have become widely used. To a lesser extent, these systems are used as low end servers.
The technical workstation market generally uses RISC based processors. Typical workloads are scientific/engineering applications which tend to be numerically intensive. These systems are also used in commercial server environments, usually for transaction processing applications.

Some important characteristics of typical systems designed for these markets which affect L2 cache design are shown in Table 1-1. These attributes have played a large role in the development of L2 cache subsystems for the PC and technical workstation markets.

Attribute	PC Market	Workstation Market
Processor Architecture	CISC (X86)	RISC
Typical L1 Cache Type	Small, Write-through	Small to Medium, Copy-back
Typical Processor Bus Speed	≤33 Mhz	> 33 Mhz
Typical OS Utilization	High	Low for technical applications
Performance Modeling	Limited modeling	High for technical workloads

Table 1-1. System Characteristics

Although Pentium™ class systems are now coming to market, the experience base for developing PC L2 caches was built on 486 based systems. Important features of the 486 are its small, write-through L1 cache, and its typical 33 Mhz external bus speed. The small size (8 Kb instruction and data typical) obviously means increased average latency on memory accesses due to more L1 misses. Although L2 caches are generally thought of as improving performance by reducing this latency, another important attribute is the increase in processor bus through-put that they can provide. The small size and write-through policy of the 486 L1 cache create significantly more bus traffic than a larger, copy-back design. Finally, processor bus speeds of 33 Mhz or less indicates that reasonable L2 caches could be built using discrete asynchronous SRAM components.

Published results of Intel based PC performance modeling are rare. Perhaps it was not considered useful given the standardization of the systems and the ease in testing prototype caches. What is abundant are tests of actual systems running a variety of benchmark programs. What these tests show is that L2 caches up to 256 Kb offer significant performance improvements over systems without L2 cache, and that copy-back designs perform significantly better than write-through designs. While benchmarking actual systems offers more certainty in the result than performance modeling, it makes analysis of the result more difficult. However, given the small size and write-through policy of the 486 L1 cache, the benchmark results are hardly surprising. With the advent of graphical user interfaces and sophisticated application programming interfaces, PC workloads tend to have high operating system utilization and context switching. The small L1 cache simply cannot bear the load that this presents. Also, all processor writes use up memory bandwidth. In a typical workload, up to 20% of all memory accesses may be writes. A copy-back L2 design has a big advantage over a write-through design in this environment, since it frees up memory bandwidth for processor writes which hit in the L2.

Technical workstations were initially developed to serve the scientific and engineering communities. More recently they have played an important role as commercial servers. They typically use RISC processors with small to medium size (8 Kb to 32 Kb) copy-back L1 caches. A great deal of performance modeling has been done for technical workloads, much less so for commercial workloads. This modeling has indicated that only limited performance gain is achieved by adding an L2 cache for many technical benchmarks. This is presumably because these workloads tend to run for long periods of time within tight instruction loops which remain resident in the L1. The benefit of the L2 comes when these applications work with large amounts of data.

Modeling of commercial workloads suggests that L2 cache does add significantly to performance. These workloads have much higher OS utilization and context switching than do their technical counterparts. The typical design for these systems has been a relatively large (512 Kb - 1 Mb), direct-mapped, write-through L2 (at least for uniprocessor systems). The high processor bus speeds have meant that synchronous SRAM components are generally employed. Since the L1 caches are copy-back, not as many processor writes are seen at the processor bus, so write-through L2 caches are usually acceptable.

2. L2 Cache Considerations for PowerPC

The PowerPC architecture allows very high performance, superscalar processor implementations. It is the nature of RISC architectures that for a given function, more instruction fetches will be required than for a CISC counterpart. Add to that the very high speeds that these processors will run at (over 100 Mhz for currently announced processors) and the instruction fetch load alone will become a serious burden on the memory hierarchy. PowerPC implementations have alleviated this to some degree by increasing the L1 cache size as compared to the 8 Kb of the 486, and making the L1 copy-back (this makes more of the memory bandwidth available to service instruction fetches). Even with improved L1 caches, though, the raw performance of PowerPC processors will demand faster and more robust memory hierarchies than current desktop or workstation implementations provide.

This argument would hold true even when running traditional operating systems and applications on PowerPC machines. However, PowerPC performance will also enable new software technologies, some of which we can predict and some which will inevitably be invented. Multi-tasking OS's, and sophisticated application programming interfaces are already common in desktop systems. Multiprocessor support is coming on line, and object-oriented OS's will be available in the near future. This will likely result in much more frequent context switching and less locality of reference, which will reduce cache efficiency. Multimedia applications will likely have high peak memory bandwidth requirements. The net result of this is that the memory hierarchy will be stressed more than ever, and L2 cache will become a key factor in distinguishing systems on the basis of performance.

L2 cache designs have three acceptance measures: their ability to increase system performance, their cost, and their usability characteristics. A traditional performance metric for L2 caches is hit ratio, or the percentage of processor memory accesses which can be serviced by the L2. Another is its latency, or how quickly it can return data to the processor. A third measure which is not considered as often is the ability of the L2 to increase memory bandwidth from the processor's point of view. This is a function of hit ratio, latency, and also the L2's write strategy. Write strategy refers to whether the cache is write-through or copy-back. The cost is a combination of the L2 component cost, as well as the cost of integrating it into a system. Finally, usability characteristics refer to things like physical dimensions, power consumption, featurability, and upgradeability.

The three acceptance measures; performance, cost, and usability, will be prioritized differently for different segments of the market. In the portable market, L2 caches are not common. They use up space and power and are thus not used. This will likely change with PowerPC portables, as the performance implications of not having an L2 are greater. In this market, usablility and cost factors will be very important. L2 caches must be small in dimension, use little power, and be inexpensive. Featurablility, or the ability to make the L2 cache optional, will also be a likely requirement. In the desktop market, all three measures can be important, depending on where in the market spectrum a product is aimed. For low-end systems cost may have priority over performance, while the reverse would be true for the high end. In both cases featurability and upgradeability will likely be required. Finally, in the server market, performance will be most important.

3. Design Guidelines

This section will discuss various L2 cache design choices with regard to the considerations and acceptance measures discussed in section two.

Inline versus Look-aside

One of the basic of issues in designing a memory hierarchy is how an L2 cache will fit into the system topologically. The two choices are referred to as inline and look-aside. An inline L2 resides between the processor bus and system bus. A look-aside L2 shares the system bus with the processor. These two design points are shown in Figure 2-1. Another possibility exists, namely that the L2 reside between the memory and the memory controller (some might call this an L3 cache). This design will not be considered in this paper, but may become important if the processor and memory controller functions are integrated on a single chip.

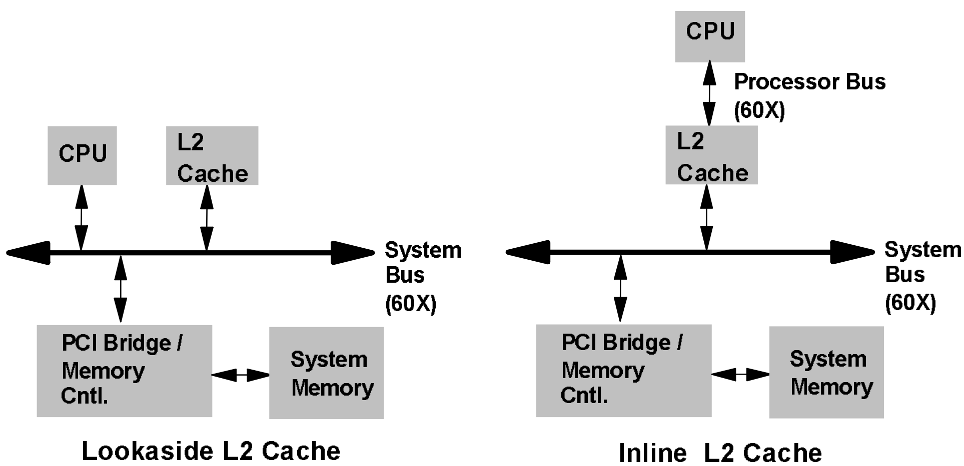

Figure 2-1. Look-aside and Inline L2 Cache Structures

An inline L2 cache has several positive attributes, resulting from the fact that it buffers the processor from the system bus. This means that L1 misses which hit L2 do not use the system bus, freeing up its bandwidth for other purposes. Likewise, the L2 cache may be able to filter system bus snoop requests from the processor, leaving the processor-L2 interface bus available to the CPU more often. This assumes that the L2 can determine whether the snoop address could not possibly be held in the L1. This would be true if the data in L1 is a subset of data in L2 (L1 inclusivity), a common design point. It would also be true if L2 maintained a copy of the L1 directory.

Inline L2 caches are generally more expensive than look-aside designs. This is largely due to the fact that two bus interfaces, one for the processor and one for the system, must be implemented. In the assumed topology this means two full 60X buses would be implemented, along with interfaces to the SRAM and tag RAM, unless these were embedded on the same chip as the controller. Another potential disadvantage of an inline L2 is that an extra penalty is paid on loads that miss L2, since the data must pass through the L2 before getting to the processor.

Look-aside L2 caches offer advantages due to their topology as well. First, a look-aside design can be made featurable, that is the system can run, albeit more slowly, without the L2. This is not feasible with an inline design, and is an important feature in the cost-competitive desktop and portable markets. Second, a look-aside design will likely be cheaper, as only one bus interface is required.

A disadvantage of the look-aside design is that it adds an extra load to the processor bus. This can become a problem as bus speeds exceed 50 Mhz. Another disadvantage is increased system bus traffic.

Look-aside L2 designs are the recommended approach for uniprocessor PowerPC systems. They can provide adequate performance for less cost than an inline, and are feasible for bus speeds up to 66 Mhz. It should be noted that this recommendation may not be true if the L2 controller is integrated either into the processor or the memory controller. Inline caches are recommended for commercial SMP server systems. Each processor in an SMP server running transaction processing software can put a heavy load on its bus, with bus utilization exceeding 50%. Inline L2 caches isolate the system bus from this traffic. Without inline L2 caches the processors would constantly stall while waiting for the system bus.

The question is less clear with regard to client SMP systems. Since little experience base exists regarding the use of these systems, it it difficult to predict whether an inline L2 per processor is generally required, or whether a shared look-aside L2 would sometimes suffice. It is fair to say, however, that a look-aside implementation could allow SMP-enabled systems to start as a uniprocessor system without L2 cache. The subsequent upgrade path would be to add the L2 and then a second processor. Given the likely demands to be placed on the memory hierarchy in this configuration, it is reasonable to suggest these systems would need a large, robust L2 cache (eg. copy-back with two or four -way associativity).

Write-through versus Copy-back

Another fundamental issue is whether the L2 cache is write-through (store-through) or copy-back (store-in). A write-through design caches processor writes, but passes them on to memory as well, assuring that memory always contains the same data for an address as the L2. A copy-back design caches processor writes, but does not pass them on to memory until either it needs to make room for data from a different address, or the latest copy of the data is required by another device on the system bus.

The main advantage of a write-through L2 cache is its simplicity, particularly if the cache will be look-aside. Cache coherence is simpler, since dirty (modified) lines wouldn't exist in the L2. This also means that store or fetch data never has to wait for a dirty line to be moved out of the L2 cache, simplifying buffering requirements.

The main disadvantage of write-through L2 caches is that updating memory on every write uses up memory bandwidth. This is particularly problematic in MP designs, but can also be a problem in memory bandwidth limited uniprocessors. As discussed earlier, this can be a result of the high demands that PowerPC processors place on the memory subsystem. It can also occur if devices with high memory bandwidth requirements are part of the system. In a look-aside configuration, performance can also be hurt if the memory controller does not perform write buffering because the writes to memory delay the processor from accessing the processor bus.

Copy-back caches minimize the write traffic on the system bus. In an inline design, writes that hit L2 do not use the system bus at all. In a look-aside design, writes that hit use the system bus for a fraction of the time compared to a write-through design. Since a main attraction of an inline L2 cache is to isolate the system and processor busses from each other, the obvious design point for inline L2's is copy-back. The recommended design for look-aside L2 caches depends on the system in which it will be used. It does seem likely that many PowerPC configurations will benefit from the increased bandwidth that a copy-back L2 cache can provide (section 4 of this paper will provide a sample analysis).

Associativity

A characteristic of an L2 cache is its line size, or the amount of data associated with a directory tag. When fetching from memory, at least one line of data is returned. Associativity determines the number of cache locations that a line of data from a particular memory address may occupy. A fully associative cache would allow a line to reside at any cache address. Direct mapped designs allow a line from memory to reside at one and only one address. A four way associative design would allow a line to reside at one of four locations in the cache.

The advantage of higher associativity is more efficient use of cache memory space. Direct mapped caches can "thrash" if two lines of data which map into the same cache address are being used at the same time. A general rule of thumb is that four way associativity provides a sufficient guard against thrashing. Another rule of thumb is that two or four way associative L2 caches in the 128 Kb to 512 Kb range will have the same hit ratio as direct mapped L2 designs twice their size.

The disadvantage of associativity is complexity and cost. In a four way associative design, four entries in tag ram must simultaneously be read and compared with the address of an incoming memory request. Then, one of four cache entries must be selected before returning data to the processor. This presents a major problem for L2 caches which employ discrete tag ram, cache ram, and L2 control components. The I/O requirements are extensive, and timing considerations may add extra wait cycles.

Two or four way associative L2 cache designs are recommended for PowerPC systems when feasible. Due to considerations discussed earlier, locality of reference may be less with emerging operating systems. This will cause a greater amount of thrashing in direct mapped designs. The cost of associativity may be offset by the ability to use a smaller cache. Associativity is a particularly attractive option for integrated L2 designs, where tag RAM, data RAM, and L2 control functions coexist on the same chip. This relieves many of the I/O and timing problems discussed earlier. If only tag ram and control functions can be integrated, two way associative would be possible. This would require interleaving the SRAM in two banks, with an address bit used to do the late select. A four way associative design could also be done using a "most recently used" algorithm. In this method, the most recently used 'way' on each SRAM interleave would be accessed. If the cache hit did not occur in the one of the assumed ways, an extra cycle would be used to read the correct way. This would not increase I/O counts and should have minimal extra timing delay.

Pipelining

Pipelining refers to the ability to accept a memory request while still processing a prior memory request. Memory accesses have an address and data phase. On a fetch request, the processor places an address on the bus and eventually receives data. PowerPC processors are capable of placing subsequent requests on the bus after receiving acknowledgment that its earlier requests were accepted but before the data phase for the first request has completed. This overlapping of address and data bus phases can significantly improve throughput.

L2 caches usually return data over multiple cycles. Latency for L2 caches is usually described by specifying how many cycles until the first data is returned from the transfer start signal on the 60x bus, followed by the number of cycles between subsequent 'data beats.' For instance, if a cache took four cycles from transfer start to first data, and then returned three more data beats in successive cycles, its latency would be described as 4-1-1-1. For this example, first data would arrive four cycles after transfer start, while the entire transfer would take 7 cycles. If no pipelining existed, then back to back fetches from L2 would take a total of 14 cycles. If pipelining were present, then while data for the first request was being returned, processing of the second request could begin. This might allow the data for the two requests to be completed in 11 cycles (4-1-1-1-1-1-1-1).

It is recommended that L2 caches support at least one level of pipelining, unless such support results in an additional wait state to a simple L2 access. Another consideration for L2 cache designs is that many memory controllers support pipelining as well. This can cause the following scenario for look-aside cache designs: Fetch A misses the L2 and is handled by the memory controller who acknowledges receipt of the address for the request to the processor. The processor then issues fetch B, which hits in the L2. The requirement on the memory subsystem is that data be returned in the same order that requests were issued. This means that the L2 cache has to hold off returning data for fetch B until the data phase for fetch A is complete. It should be assumed that memory controllers will support varying degrees of pipelining, from none to two requests deep. L2 caches must be compatible in this regard with its target memory controllers. To be most general, an L2 cache should work with different levels of memory controller pipelining.