

CS 838: Performance Analysis of TokenCMP

Mike Marty

1.0 Introduction

This paper is meant to be read as an addendum to work submitted to ISCA 2004 [1]. In this work, a TokenCMP protocol was developed that was simpler and faster than a 2-level directory protocol. Although we showed significant improvement in performance, we did not provide a detailed explanation nor analysis of the performance results. Furthermore, the 2-level directory protocol compared against did not implement similar coherence policies.

In this paper, I further analyze the performance of various TokenCMP protocols. They are compared against an improved 2-level directory implementation that has similar performance policies to the best-performing TokenCMP variants. To study the effect of filtering and retry timeout latency, eight additional TokenCMP variations are implemented and compared. The causes for performance differences among TokenCMP variations will become apparent and will serve as guidance towards developing further improvements.

2.0 TokenCMP Review

2.1 Description

TokenCMP is an adaptation of TokenB [4] designed for moderately sized systems built from multiple CMP chips. We target, and simulate, a system built from four, 4-way CMP nodes each with a shared L2 and private L1 caches. As in TokenB, broadcast is used to reduce the latency of cache-to-cache transfers both internally and externally. That is, on a L1 miss the request is broadcast to other on-chip caches including the L2 and other local L1s. L2 controllers use filters to determine whether the local miss should be broadcasted to other CMP nodes. L2 controllers receive external messages and use the same filter to determine if it should be broadcasted locally to the L1 caches.

TokenCMP extends token counting, used to enforce coherence at the processor endpoints, by distributing tokens to individual caches rather than to a single processing node. Forward progress is achieved by using the distributed persistent request scheme [2] with a table of outstanding persistent requests at each cache.

Three variations of TokenCMP were presented-- TokenCMP0, TokenCMP1, and TokenCMP2. These protocols differed on the number of transient requests (before going persistent), and the timeout latency for issuing a retry. The details can be found in Table 3

2.2 Performance (ISCA 2004 submission)

Results in [1] indicated a substantial performance difference between TokenCMP2 and TokenCMP1. The protocol differences between these two variations are summarized below:

- TokenCMP1 makes a single transient request, and then goes persistent immediately upon a timeout
- TokenCMP1 uses a fixed retry timeout of 250 cycles (roughly the time to satisfy an off-chip request). Whereas TokenCMP2 dynamically varies the retry timeout based on the measured latency of previously completed requests

TokenCMP2 computes a retry timeout latency based on the measured and averaged latencies of previously completed requests. In TokenB, token coherence was done at the node level such that the only requests included in the latency estimate were L2 misses requiring coherence transactions at the global level. In TokenCMP, L1 controllers are responsible for tracking outstanding coherence requests-- that is, token coherence is done at the L1 level rather than the L2 level. Therefore, L2 hits are included in the running latency estimate of completed requests. We believe that this causes the running average of completed requests to be driven to low levels. Thus when a miss does occur, a quick succession of requests is emitted from the L1 controllers.

We hypothesize that this “machine gun” effect may improve the performance caused by coherence races. First, if a request fails due to an in-flight message containing tokens, the “trail” of requests may then succeed in “finding” the tokens. Second, once a quick succession of requests have been emitted, the transient retry threshold is reached causing the request to go persistent. In cases of contention in which processors are competing over tokens, getting to the persistent request mechanism sooner is conceivably better than all competing processors incurring long timeouts before going persistent. Third, the bandwidth-reducing filters forward any retried transient requests. If these filters are often incorrect, then this “machine-gun” effect will cause the retry to be correctly broadcasted sooner.

3.0 Expanded Analysis

This goal of this performance analysis is to further understand the performance of TokenCMP. First, DirectoryCMP1 will be discussed which will be used alongside DirectoryCMP0 as comparison points. Then the various TokenCMP implementations will be used to test hypothesis.

3.1 DirectoryCMP1

DirectoryCMP1 is a different implementation of a 2-level directory protocol with similar policies to the best performing TokenCMP protocol. In particular, it implements the exclu-

sive state and migratory sharing for faster read-modify-write sequences for shared and non-shared blocks. The shared L2 cache is also non-inclusive whereas strict inclusion is maintained with DirectoryCMP0. Anecdotal evidence indicates that DirectoryCMP1 is much more complex than DirectoryCMP0 as the number of states in the L2 controller is nearly doubled. Non-inclusive caches is one reason but other contributors include no requirements on point-to-point network ordering and chip exclusiveness-- the protocol knows if all shared copies exist on-chip such that a local GETX does not need to be ordered at the global directory.

Table 1 gives measured latencies for certain classes of uncontended coherence transactions. Note that local cache-to-cache misses include GETS requests that can be satisfied by a local L1 sharer.

TABLE 1.

	TokenCMPx	DirectoryCMP1
miss to local memory	161	169
miss to global memory	219	227
local cache-to-cache miss	12	19
global cache-to-cache miss	82	243

TokenCMP is expected to outperform 2-level directory protocols by removing the costly indirection through the global directory/memory as shown in the measured latencies. On workloads that exhibit a large number of cache-to-cache transfers, I expect to observe a noticeable improvement in runtime. Profiling the occurrence of these transaction classes will be left for future work. However in a system fixed at 16 processors, as I assume throughout, increased CMP integration will result in fast on-chip cache-to-cache transfers. Therefore, the expected improvement will be less than observed in a 16-processor SMP in which all cache-to-cache transfers incur similar penalties which are larger due to going off-chip.

Also note that our simulation infrastructure interleaves memory on the lower order bits of the cache block index. Furthermore, even though Solaris supports NUMA memory allocation techniques, our simulator does not provide the OS with necessary information required to do so. The ramifications for our simulation of TokenCMP is that the number of misses to global memory is dilated and will result in extra traffic on the global interconnect. I expect this effect to be minor as the bandwidth is not limited in our simulations.

3.2 TokenCMP Variations

Table 2 summarizes the variations of TokenCMP that will be examined.

TABLE 2.

	transient requests	retry latency	E-state/migratory	notes
TokenCMP0	1	250	no	
TokenCMP1	1	250	yes	
TokenCMP2	3	dynamic	yes	
TokenCMP3	3	25	yes	
TokenCMP3B	3	25/delay	yes	persistent request delayed by 100 cyc
TokenCMP4	1	15/250	yes	retry depends on L2's filtering action
TokenCMP5	3	dynamic	yes	latency estimates do not include L2 hits
TokenCMP6	1	500	yes	no filter
TokenCMP6B	1	500	yes	
TokenCMPNull	0	n/a	yes	all requests persistent

3.3 Updated Simulation Parameters

- L2 capacity was increased from 4 MB to 8 MB per chip. The higher capacity should result in more sharing misses and will reflect future silicon trends
- Bandwidth is effectively increased to infinity so that network contention, due to a possible simulator bug, doesn't affect the performance comparisons

3.4 Hypothesis #1: TokenCMP2 overcomes incorrect filtering

TokenCMP uses filters at the L2 cache to drop initial outgoing and incoming transient requests (persistent and retried requests are not filtered). The primary purpose of filtering outgoing requests is to relieve the bandwidth pressure of the global interconnect as well as the pin I/O-- arguably a more critical resource as it is limited by technology rather than monetary cost. Filtering incoming requests, from the global interconnect, will also reduce the snoop bandwidth required of the L1 caches.

The filter maintains a limited copy of the L1 tags that is non-atomically and speculatively updated such that it may be incorrect. A filter entry consists of a bit for each on-chip processor which indicates that a copy exists at that processor. In addition, a "chip exclusive" bit is maintained to denote that no other external sharers exist. This bit is used to filter out GETX requests in which all of the sharers are on-chip.

When a local L1 broadcasts its request to all on-chip caches, the L2 first accesses the filter to determine if the request should sent off-chip and carries out the appropriate action. Then, it sets a bit in the filter indicating that the requestor is now a sharer *before* the request has been satisfied¹. The bit is cleared on an L1 writeback. L1 caches also explicitly

inform the L2 when it has lost all of its tokens for a block due to an external transient request.

The chip-exclusive bit is set when the L2 observes a local GETX request and is cleared when a external request is observed/forwarded. This action may cause an initial request, made by another on-chip processor, to be incorrectly filtered. However this may result in a request-combining effect where only a single initial request is propagated off the chip.

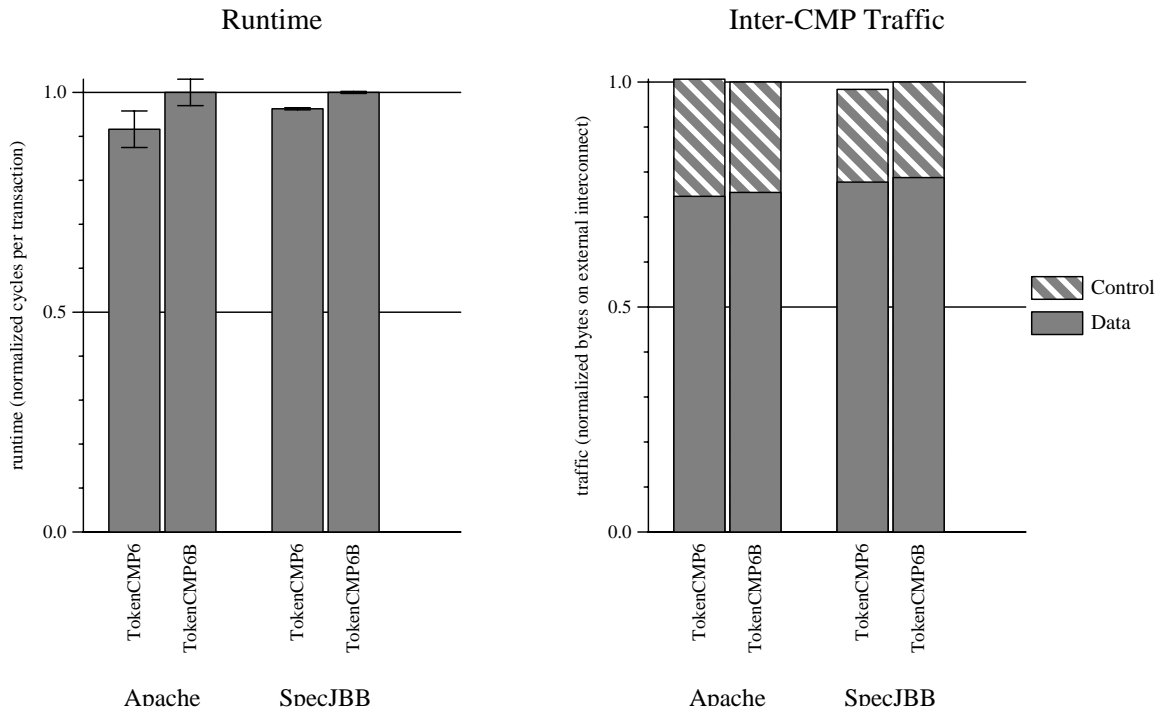
In TokenCMP3 and TokenCMP2, the initial transient request is quickly followed by a retry that bypasses filtering. To eliminate the possibility that incorrect filtering is the cause for superior performance of these protocols, TokenCMP6 and TokenCMP6B are compared. These variations make a single transient request before going persistent. The timeout is set at 500 cycles to exaggerate the effect of the transient request failing. The only variable in this experiment is that TokenCMP6 disables the filter such that all requests are broadcast.

-
1. This characteristic could be useful for future work as competing requests from external processors will then be forwarded by the L2 filter such that “collision detection” could be performed.

TABLE 3. Number of messages observed on external interconnect. A single JBB run is shown for each protocol

	External Requests	Requests filtered
TokenCMP6	1267737	1158143
TokenCMP6B	1301154	0

FIGURE 1. Filter Performance Comparison



As shown in Figure 1, the filters hurt performance by 2-9%. With a more reasonable time-out latency, I expect the performance penalty to be even smaller. However Figure 1 also shows that the filters are not effectively reducing traffic on the external interconnect. In Table 3, the external requests received by the L2 controller is shown for a single run of a single JBB workload for both TokenCMP6 and TokenCMP6B. The filter is able to drop 91% of external requests which will substantially relieve the L1 snoop bandwidth. Unfortunately, direct statistics are not available for the outgoing filter. Instead, we can compare the number of external requests seen by the L2 controllers for TokenCMP6 and TokenCMP6B. With outgoing filtering enabled, only a 2.5% reduction in messages on the external interconnect is observed¹. The filters are clearly not effective and they will be further studied.

1. The variability of JBB is low enough that workload perturbations should not have a substantial effect on the number of messages.

3.5 Hypothesis #2: A quick succession of requests improves performance

TokenCMP3 is similar to TokenCMP2 except that it puts out a burst of transient requests at a fixed latency, and then goes persistent on the fourth request.

Figure N shows that Token3 performs similarly to Token2. This does indeed confirm a burst of requests improves performance. However it is still possible that the reason for improved performance is the reduced time until the request goes persistent.

3.6 Hypothesis #2B: TokenCMP2 and TokenCMP3 perform well because requests go persistent quickly

TokenCMP3B is the same as TokenCMP3 except for one key difference: the issuing of a persistent request is delayed by 100 cycles. Therefore, if the performance improvement observed with TokenCMP2 and TokenCMP3 is due to the avoidance of very small coherence races, performance should be similar between TokenCMP3B and TokenCMP3.

[RESULTS UNAVAILABLE AT 9AM SUBMISSION DEADLINES]

4.0 Other TokenCMP Variations

4.1 TokenCMP5 and TokenNULL

[RESULTS NOT AVAILABLE FOR 9AM SUBMISSION]

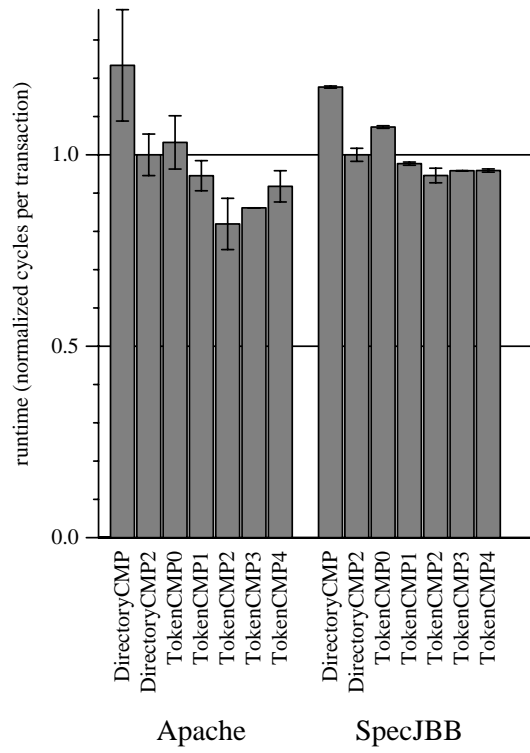
4.2 TokenCMP4

Ideally the timeout latency should be slightly longer than the time expected to complete the request. If a race condition occurred on-chip, then it would be advantageous to retry sooner than if the race condition occurred off-chip. In other words, the retry latency should not be significantly longer than the round-trip time required to communicate with every relevant cache.

We've also seen that the persistent request mechanism is an efficient way to deal with contention. If it is inevitable that a request will go persistent, then it should not be delayed by futile transient attempts.

TokenCMP4 attempts to achieve these goals. On every transient request made by the L1 cache, the L2 filter sends an explicit reply informing the controller if the request was broadcasted off-chip or dropped. If sent off-chip, a longer fixed timeout latency is used. If

Runtime



Inter-CMP Traffic

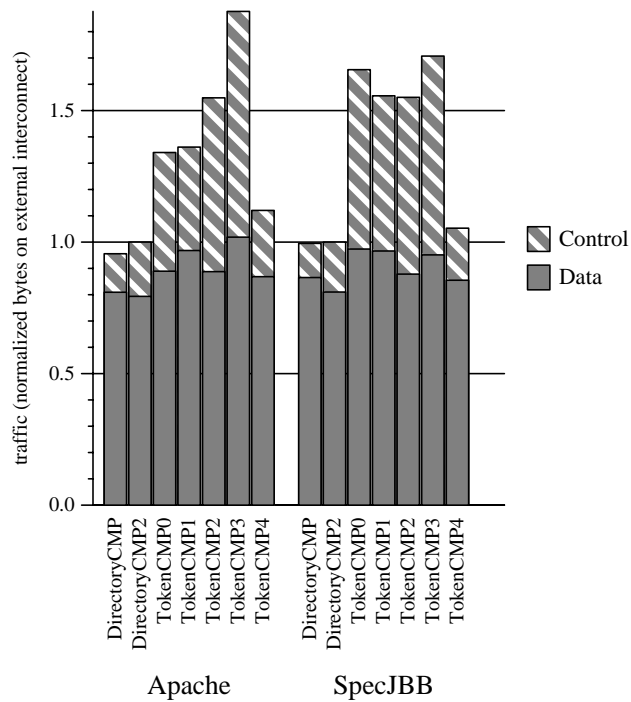


FIGURE 2 Performance of TokenCMP variations compared to DirectoryCMP

the request was filtered, the L1 controller times out very quickly. On the first timeout, the persistent request mechanism is immediately invoked.

As shown in Figure 2, the performance of TokenCMP4 almost approaches that of TokenCMP2 and TokenCMP3 yet it uses far less traffic.

Table 4 shows that the first transient request succeeds 98.3% of the time which is significantly better than the other TokenCMP variations. This proves that the timeout scheme chosen provides ample time for the transient request to be fulfilled when possible.

TABLE 4.

	transient requests	1 issue	2 issues	3 issues	4 issues
TokenCMP0	1	86%	14.3%	~ 0%	~ 0%
TokenCMP1	1	86%	14%	~ 0%	~ 0%
TokenCMP2	3	87%	1.4%	2.8%	9%
TokenCMP3	3	86%	0.1%	2%	11%
TokenCMP4	1	98.3%	1.7%	~ 0%	~ 0%

5.0 Conclusions and Future Work

This analysis has shown the importance of the retry timeout latency in hierarchical Token Coherence protocols. Taking into account of the hierarchical cache and network has proved to be an effective means to choosing a timeout latency that allows transient requests to be fulfilled yet doesn't waste time by waiting. However work still needs to be done in order to approach the performance of TokenCMP2 and TokenCMP3.

Furthermore, with the constraints on off-chip bandwidth and the ineffectiveness of request filtering, it is clear that other mechanisms must be explored. Destination-set prediction [3] may be the key to achieving these goals and will be explored for future implementations of TokenCMP.

6.0 References

- [1] M. R. Marty, M.D. Hill, M. M. K. Martin, D.A. Wood. *Implementing Faster and Simpler Multiple-CMP Systems using Token Coherence*. Internal Document, University of Wisconsin, 2003.
- [2] M. M. K. Martin. *Token Coherence*. PhD thesis, University of Wisconsin, 2003.
- [3] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 206–217, June 2003.
- [4] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.