From the 18th International Symposium on Computer Architecture, 1991

Race-free Interconnection Networks and Multiprocessor Consistency

Anders Landin, Erik Hagersten and Seif Haridi

Swedish Institute of Computer Science[1]

Abstract

Modern shared-memory multiprocessors require complex interconnection networks to provide sufficient communication bandwidth between processors. They also rely on advanced memory systems that allow multiple memory operations to be made in parallel. It is expensive to maintain a high consistency level in a machine based on a general network, but for special interconnection topologies, some of these costs can be reduced.

We define and study one class of interconnection networks, *race-free networks*. New conditions for sequential consistency are presented which show that sequential consistency can be maintained if all accesses in a multiprocessor can be ordered in an acyclic graph. We show that this can be done in race-free networks without the need for a transaction to be globally performed before the next transaction can be issued.

We also investigate what is required to maintain processor consistency in race-free networks. In a race-free network which maintains processor consistency, writes may be pipelined, and reads may bypass writes.

The proposed methods reduce the latencies associated with processor write-misses to shared data.

1 Introduction

The presence of data in multiple copies in a machine and the possibility of performing memory accesses in parallel have effects on which synchronization techniques can be used in a multiprocessor. Trade-offs have to be made between system performance versus hardware cost and generality of synchronization and inter-processor communication models. Several memory-access models have been proposed. They differ in the degree of consistency offered by the multiprocessor. Stronger access models are more expensive to maintain in that they require some kind of serialization between accesses, while looser models extensively can utilize techniques such as lockup-free caches [14] to improve system performance.

Several network topologies have been proposed, ranging from the single shared bus to complex networks such as meshes or multistage networks. This paper identifies one class of interconnection networks, *race-free* networks, and shows that the use of such networks can reduce cost in terms of traffic overhead and wait times for maintaining different memory access-order models. In section 2 of this paper we investigate what conditions are needed to maintain sequential consistency in a system. In section 3 the race-free network is defined. Section 4 deals with the conditions needed to maintain sequential consistency in race-free networks. In section 5 we relax the scheme introduced to maintain sequential consistency, and study two access-order models that lie close to processor consistency. Section 6 proposes different methods for improving the bandwidth of a race-free network. A summary of presented results is found in section 7.

1.1 Memory access models

Sequential consistency was introduced by Lamport in [11]. This is the highest consistency level, and guarantees that the result of an execution of a program on a multiprocessor is the same as the execution on a single processor with multitasking.

Processor consistency, introduced by Goodman [7], offers a lower level of consistency. It guarantees that the result of any execution is the same as if the operations of each individual processor appear in the sequential order specified by its program, but the order in which writes from two processors occur need not be observed in the same order by all processors.

Looser access models have been proposed by several groups (e.g. *weak ordering* [4, 2], *release consistency* [5]). They have in common that processors only may communicate under explicit synchronization with primitives that are recognized by hardware.

In a recent paper [6], Gharachorloo et al. present performance evaluation results for an architecture based on a general network showing that the looser access models for several applications show results comparable to processor

[1] SICS, Po. Box 1263, S-164 28 KISTA, Sweden. Email: landin@sics.se, hag@sics.se, seif@sics.se.

consistency, while sequential consistency always performs worse. They advocate that processor consistency is a suitable trade-off between demands on performance and implementation cost for systems with general networks.

1.2 General assumptions

In the following discussions, it is assumed that a multiprocessor may maintain multiple copies of data, for example, by the use of caches. Caches may be multilevel, and can be local to a processor or used by several processors in a part of a system. By *global* data we mean data that is common to several processors. By *shared* data, we mean data that for the moment exists in several copies and is directly accessible to a processor by reads. By the term *write-miss* we mean a write to data that is not exclusively owned by the writing processor. Data is allowed to reside in different parts of the system, for example in distributed memory blocks as in a NUMA (Non-Uniform Memory Access) architecture. It is also assumed that the system uses some kind of coherence protocol that is based on a write-invalidate policy.

Furthermore, for the sake of simplicity, we assume that the size of a coherence unit (here called *item*) is equal to the line size of the caches, and that all processor communication is done through memory operations (inter-processor interrupts propagated outside the race-free network for example are not allowed for communication here).

2 Sequential consistency

In this section we will show that sequential consistency can be maintained in a multiprocessor if certain conditions on the ordering of accesses are satisfied. These conditions are less restrictive than those presented earlier.

Lamport defined sequential consistency in [11]:

> [A system is sequentially consistent if] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

It is normally expensive to maintain sequential consistency in cache-based multiprocessors. Scheurich and Dubois formulated in [15] a condition that is sufficient but not necessary to maintain sequential consistency in a cache-based machine.

> Condition A (Sequential consistency): "Sequential consistency is satisfied in any system if an access may not be performed with respect to any processor until the previous access by the same processor has been globally performed and if accesses of each individual processor are globally performed in program order."

The key terms *perform*, *perform with respect to* and *perform globally* were defined by Scheurich and Dubois. We formulate them as:

Definition 1 (Perform with respect to)
A LOAD by processor i is considered performed with respect to processor k, at a point in time when a subsequently issued STORE to the same address by processor k cannot affect the value returned by the LOAD.

A STORE by processor i is considered performed with respect to processor k at a point in time when a subsequently issued LOAD to the same address by processor k returns the value defined by this STORE (or a subsequent STORE to the same location).

Definition 2 (Performing globally)
A STORE is globally performed when it is performed with respect to all processors. A LOAD is globally performed if it is performed with respect to all processors and if the STORE which is the source of the returned value has been globally performed.

Further, for simplicity we formulate:

Definition 3 (Perform)
A STORE by processor P is considered performed when is is performed with respect to P.

A LOAD is considered performed when it is performed with respect to all processors.

As have been pointed out before [1, 3], Condition A is unnecessarily restrictive since it requires that each access is globally performed before the following access may start.

We investigate the conditions for sequential consistency based on general relations between accesses in a multiprocessor. The reasoning here uses basically the same strategy as Lamport used in [11].

Definition 4 (Access graph)
The Access graph is a directed graph of accesses in a multiprocessor with the arcs defined by the following relations:

- *Accesses from the same processor are ordered in program order.*

- *Two writes to the same address from different processors are ordered.*
- *A read R is preceded by the write that is source of the value read by R.*
- *A write W, that follows another write W_s to the same address, is ordered after a read R if W_s is source of R.*

Theorem 5 (Sequential consistency)
Sequential consistency is maintained in a multiprocessor if the Access graph is an acyclic graph.

Proof:

If the Access graph is an acyclic graph, then all accesses in the multiprocessor can be ordered in a sequential order. Since the accesses of each processor appear in the access graph in program order, this implies that sequential consistency is maintained.

It is interesting to note that from the Access graph, the data-flow relations between a read and its preceding write can be extracted as a partial order. This gives the minimal conditions required to maintain sequential consistency in a program if the definition by Lamport is interpreted so that the result of the execution of a program is the set of data that the reads in the program return. The implications of this go beyond the scope of this paper.

The following conditions are sufficient to obtain an acyclic access graph, and thus sufficient for sequential consistency:

Condition 6 (Sequential consistency)
If the following conditions are satisfied in a multiprocessor, sequential consistency is maintained.

[w-w] Two consecutive writes from the same processor must be performed with respect to any processor in the order specified by the program.

[w-r] A read that follows a write W in the program of a processor, may not be performed until all writes before W also have been performed with respect to the processor.

[r-r] A read that follows another read R in the program of a processor, may not be performed until all writes before the source of R also have been performed with respect to the processor.

[r-w] A write that follows a read in the program of a processor may not be performed with respect to any processor until the read has been performed.

[p-p] Two writes from different processors to the same address, must be performed in the same order with respect to any processor.

Proof:

We will show that with these conditions, only acyclic access graphs can be built.

[w-w], [w-r], [r-r] and [r-w] guarantee that the accesses of each processor appear as ordered in program order.

[p-p] assures that writes from different processors to the same address are ordered.

[w-r] and [r-r] guarantee that the source of a read is ordered before the read.

[p-p], [w-w] and [r-w] assures that a write that follows a source to a read are seen in this order by all processors.

If the Access graph is not acyclic, then this implies that a cyclic relation exists between accesses. Program order is by definition acyclic, thus a cycle in the graph must involve at least two processors. All access relations between two processors relate accesses to the same address. The following transitions are then possible: w$\mapsto$w, w$\rightarrowtail$r and r$\rightarrowtail$w. If we have a cycle with two writes to the same address, condition [p-p] has been violated. If the cycle includes a write and a read, this implies that the write is the source of the read and also that is follows the source of the read. This violates [w-w] or [w-r] since two writes from a processor must be performed in program order with respect to any processor therefore it must be well defined which write is source and which follows the source to the read, and further [w-r] and [r-r] assures that the read may not be performed until all accesses ordered before the read in the access graph also are performed with respect to the reading processor. Thus the access graph must be acyclic, and sequential consistency is guaranteed

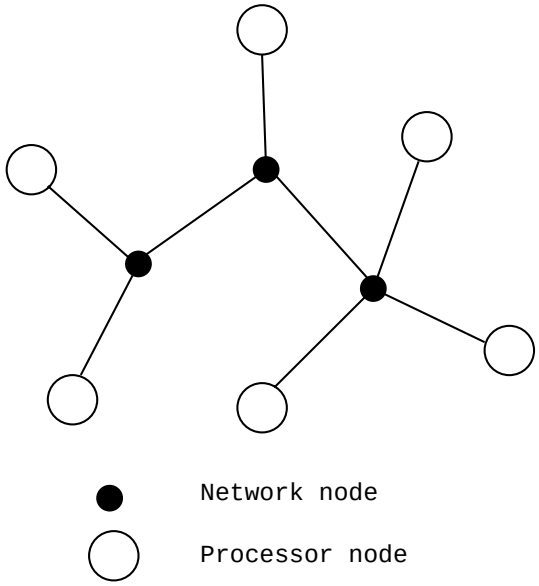

Figure 1: An example of a race-free network

3 Race-free networks

The fact that the cost of maintaining some kind of consistency in a multiprocessor varies for different interconnection network topologies has been briefly mentioned in [7]. We will identify one class of networks, *race-free networks* (RFN), that shows interesting topological properties and allows for more efficient implementations of different access-order models than those based on general networks.

A network graph consists of processor nodes, network nodes and arcs connecting the nodes. Processor nodes are in the tips of the network. By a *path* from a node X to another node Y we mean the sequence of network nodes that a transaction has to pass to travel from X to Y. Processor nodes contain the processors and possibly distributed memory and/or local caches. Network nodes may contain distributed memory or caches as well.

Definition 7 (Race-free network)

- *A race-free network is a network with the topology of any acyclic undirected network graph.*
- *Transactions propagate on the arcs in the network without the possibility of overtaking each other.*
- *Transactions may be buffered in the network nodes but buffers must maintain a strict FIFO order between transactions.*

For simplicity, the buffering in a node is considered to be done so that when a transaction arrives at a node, the transaction is immediately handled. The resulting transactions (normally a forwarding of the old transaction) are put in a FIFO buffer common to all transactions about to leave the node. A node has *issued* a transaction when it is buffered in the FIFO buffer.

The race-free network has properties that facilitate the support of consistency in a system:

Theorem 8 (Transaction ordering)
In a race-free network, transactions issued in a specific order from one node are observed in the same order by all other nodes.

Proof:

Transaction ordering follows from the inability of transactions to overtake each other in the network, and because all transactions have to propagate along the same path, since the network is acyclic.

Causal correctness was formulated by Scheurich in [14]. We formulate the following theorem for race-free networks:

Theorem 9 (Causal correctness)
Any transaction A that has reached a node X in a race-free network before X issues a new *transaction B will also have reached any other node Y before it receives B.*

Proof:

The transaction A can be one of the following three types: 1) Transactions that first reach Y and then propagate to X; 2) transactions that first reach X and then propagate in the network towards Y; 3) transactions that reach a node Z which is the first node they reach that is in the path from X to Y. The casual correctness (CC) is obviously satisfied for a transaction A from group 1. That transactions from group 2 satisfy CC follows from the transaction ordering. Transactions from group 3 have all passed Z before B reaches Z, thus the transaction ordering from Z to Y gives that CC is true.

That this is satisfied in any acyclic undirected graph was pointed out by Dahlgren in [16].

The properties of a race-free network also help support coherence. To handle coherence, a node in a network that receives an invalidation of an item and has a buffered data transaction for the same item or receives a data transaction for that item from another arc before the invalidation has been further propagated must invalidate the data transaction and may not propagate it to any other node in the system. We call this property *local coherence*. Similar functionality may also be required for example to detect and solve write races on an item from two or more processors. Since this paper focuses on access ordering such mechanisms are assumed without explicit statement in the following.

The nodes in a race-free network may be ordered in a tree. In some tree-ordered directory- or cache-based systems, the network keeps state information about the locations of datum copies. If such knowledge is present and a tree-ordering exists, the item is said to be *exclusive* in the *subsystem* which is the smallest subtree that contains all copies of the item. The network node that is the root in the subtree is called the *root node* in the subsystem or for that item. Note that this is a dynamic relation. If such information is not kept in the network or no tree-ordering exists, a centralized node in the network is regarded as root node for all items, and all items are exclusive to the whole system. In systems without tree ordering, by subsystems to a node, we mean nodes in the direction from the central root and towards the tips of the network.

4 Sequential consistency with race-free networks

In this section we will investigate which demands are required for maintaining the highest consistency level, *sequential consistency* in a race-free network, and how this differs from the case of a general network.

4.1 General networks

The normal action taken to ensure sequential consistency is to suspend the processor (or at least prevent it from accessing shared data) upon a write-miss to global data. An invalidation or write request of the item is then sent out to all processors that share copies of the item. The processor can continue to execute when an invalidation acknowledge has been received from all other processors indicating that all other copies have been destroyed. This follows Condition A by Scheurich and Dubois presented above. The time the processor has to be suspended is then equal to the longest time it takes for request and acknowledge messages to propagate.

4.2 Race-free networks

An informal argument why all invalidations must be acknowledged from all the processors in a general system before the processor may continue execution is that otherwise a processor in a general network may continue execution and perform a read without knowledge of all the writes made before its own write and thus precede its source. With a race-free network, a better scheme can be used. Here, it is not necessary to let the processors acknowledge invalidations; instead, an invalidation acknowledge can be sent out directly from a network node if it can guarantee that the acknowledge will arrive at the processor after invalidations from all writes that precede this write.

The following conditions are sufficient for maintaining sequential consistency in a race-free network:

Definition 10
A write-access is considered performed with respect to a node when the node has buffered the write-access for propagation to all other nodes in its subsystem.

Condition 11 (Sequential consistency, RFN)
In a system with a race-free network sequential consistency is maintained if:

- *A reading processor may continue execution when the read has been performed.*
- *A writing processor may continue execution when the write has been performed with respect to its root node, and when all writes performed with respect to the root of this item previous to and including this write, also have been performed with respect to all nodes in the path from the root downto the processor.*

Proof:

Sequential consistency is maintained if condition 6 is satisfied.

Paragraph [w-w] is satisfied since: 1) The first write is performed with respect to all nodes in the path from the root to the processor before the second write may be issued, thus all processors with a path to the writing processor that does not include the root will receive this write before the second due to theorem 8; 2) all other nodes that share copies of both items will observe the first write before the second write due to theorem 9.

[w-r] and [r-r] are satisfied since an item read by a processor either is cached at the processor or not. If it is cached we know that the source of the read can not be ordered before the processors last write, since when the processor was allowed to continue execution, all previous writes had been performed with respect to the processor. If the item not is cached, before it can be propagated to the processor, all writes previous to the source have also been propagated. Since an access after a read is not performed until the read is performed, [r-r] and [r-w] are obviously satisfied.

[p-p] is satisfied because all writes to the same item are sequentialized to an order by the root node in the subsystem.

4.3 Scheme 1

We will here sketch the fundamental mechanisms of a protocol that fulfills the condition 11 in a race-free network. We call this scheme *Scheme 1*:

Scheme 1

- *At a write-miss to global data, the processor sends a write request to the network and suspends execution.*
- *The network node that is root node for that item sends invalidations to other copies of the item and returns a write acknowledge to the writing processor.*
- *A node that receives an acknowledge forwards the acknowledge to the writing processor and sends invalidations to other subsystems.*
- *The processor may continue to access global data when the write acknowledge is received.*
- *At a read-miss the processor sends a read request to the network and suspends execution until data is received from the network.*

This scheme satisfies condition 11. Since reads are blocking, the condition for reads is satisfied. Write-misses meet the condition since the acknowledge transaction is preceded by all other transactions that arrived at the root before the write-request; when a suspended processor receives an acknowledge, all transactions that reached the root node before the write also will reach the processor before it receives the acknowledge. Further, due to the transaction ordering properties of the network, all processors with shared data will see invalidations due to writes in the same order as in which they appear in the access graph.

An example can be studied in figure 2. Note that this only describes the fundamental mechanisms of the access ordering in the network. A scheme similar to this was introduced by Warren and Haridi in [17] and a full implementation is described in [10, 9].

In systems where the size of the smallest writeable unit (write unit size) is less than the item size (which is the common, general case), a processor must have a copy of the item it writes to before it performs the write.[2]

In [1] Adve and Hill propose new optimizations for use in general networks. The optimizations allow a processor to continue execution after a write has been issued as soon as the processor has a copy of the item written to. The restriction is that the data the processor may access before it has received the write acknowledge must be exclusive to that processor (i.e., it must suspend accesses to shared data). This method is possible to introduce in a race-free network as well. It requires that no processor performs a write-access on data that someone else has exclusively (this is satisfied if a write requires that the writer first gets a copy of the item). A processor may continue execution after a write has been issued as long as it only accesses data exclusive to the processor. The accessed data may not be distributed to any other processor until the outstanding write acknowledge has been received. If an invalidation is received instead of an acknowledge, the write must be retried. Incoming requests to items accessed during a pending write may not be served until the write has been acknowledged.

[2] Or in an extreme case, keep information about each item exactly which parts are modified. Extra care must also be taken when an invalidation is performed so that no parts of an item is lost.

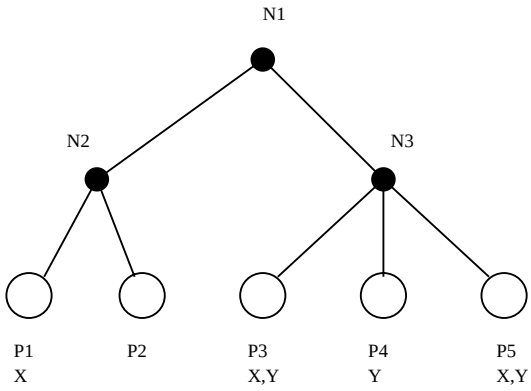

Figure 2: A hierarchical system with five processors P1 – P5. P1, P3 and P5 have copies of X. Y is shared by P3, P4 and P5. In this situation N3 is root node for Y, while N1 is root node for X. If P3 issues a write to X at the same time as P5 issues a write to Y, both write requests will reach node N3. N3 sends the request for X to N1, an invalidation of Y to P3, and an acknowledge for Y to P5. N1 returns an acknowledge for X to N3 and an invalidation to P1. When N3 receives the acknowledge for X, it forwards the acknowledge to P3 and sends an invalidation of X to P5.

4.4 Comparison between race-free and general networks

The race-free network scheme contributes in two ways to improve performance compared to the general case.

- The time a processor has to be suspended is reduced since the write acknowledge is returned directly from the network. In the general case, the suspension time is: $max(t_{inv-to-proc_i} + t_{ack-from-proc_i})$, and with the new scheme the processor is suspended: $t_{inv-to-root} + t_{ack-from-root}$.
- The total network load is reduced with the new scheme since the number of transactions needed to maintain consistency is reduced. For the general scheme, $2 * N$ transactions are required if the item written is shared by N processors. In a race-free network, $2 + N$ transactions are sent. Note however that most of the transactions in the latter case only are propagated in a small part of the network.

In tree-structured networks, the delay for writes to shared global data can be reduced by about 50 % with the new scheme. How this effects system performance varies greatly depending on the fraction writes to global data vs other memory accesses, the amount of sharing and cache sizes.

5 Lower consistency levels

It is possible to relax a few of the conditions for sequential consistency described above and thereby achieve new access-order models close to processor consistency. The models described here are basically processor consistent. It may however be interesting to note that they are somewhat stronger than that.

Processor consistency was defined by Goodman in [7]:

> A multiprocessor is said to be *processor consistent* if the result of any execution is the same as if the operations of each individual processor appear in the sequential order specified by its program.

Gharachorloo et al. formulated in [5] the following access restrictions needed to obtain processor consistency in a system with a general interconnection network:

> (A) before a load is allowed to perform with respect to any other processor, all previous LOAD accesses must be performed, and
>
> (B) before a STORE is allowed to perform with respect to any other processor, all previous accesses (LOADs and STOREs) must be performed.
>
> The above conditions allow reads following a write to bypass the write.

For systems based on race-free networks, processor consistency can however be obtained under less restrictive conditions. In addition to allowing reads to bypass writes, pipelining of writes is possible. The following conditions are sufficient for processor consistency:

Condition 12 (Processor consistency)
Processor consistency is satisfied in any system if:

[w-w] Two consecutive writes from the same processor must be performed with respect to any processor in the order specified by the program.

[r-r] A read that follows another read R in the program of a processor, may not be performed until all writes before the source of R also have been performed with respect to the processor.

[r-w] A write that follows a read in the program of a processor may not be performed with respect to any processor until the read has been performed.

Condition [w-w] guarantees that all processors observe the same write-sequence from any other processor. The conditions [r-r] and [r-w] require that the reads of the processor appear in this sequence in program order.

These conditions allow reads following a write to bypass the write and subsequent writes to be pipelined.

5.1 Scheme 2

```
Initially: X=Y=0 in processor caches.

P1:                      P2:

X=1;                     Y=1;
if (Y=0) then            if (X=0) then
   unique;                  unique;
```

Figure 3: Two processors execute the programs P1 and P2. In sequential consistency it is guaranteed that maximally one processor enters the unique region. In Scheme 2, it may very well happen that both processors issue their writes and perform their reads before the invalidations have propagated. Note the condition that they already share copies of X and Y.

Let us start by relaxing the demand in Scheme 1 that a processor must be suspended from execution from when it has issued a write request until it receives the acknowledge. If we prevent the processor from reading the value written and prevent information about the value of the item to propagate to any other node in the network until the acknowledge has been received an interesting situation arises.

Now, the way invalidations propagate in the network is modified. In the previous scheme, invalidations were sent out from the root node. If instead the write request propagating in the network towards the root directly causes invalidations to be sent into the subsystems it passes, an interesting access-order model is achieved. We call this new scheme *Scheme 2*. If the size of an item is larger than the smallest writeable unit additional care must be taken. In [8] a protocol with similarities to this is shown in detail, which handles such false sharing.

Scheme 2

- *At a write-miss to global data, the processor sends a write request for the item to the network.*
- *A network node that receives a write request from a subsystem sends invalidations of the item to other subsystems. If it is the root of the item an acknowledge is returned; otherwise, the request is propagated towards the root.*
- *The processor may continue to access global data as soon as the request has been issued, but may not read or propagate the value of the item until the write acknowledge has been received.*
- *At a read-miss the processor sends a read request to the network and suspends execution until data is received.*

This scheme meets the requirements of processor consistency because of the transaction ordering properties of the network; all other processors receive invalidations of items in the same order as the requests are issued by the writing node. Since the writes appear ordered and reads are blocking the scheme satisfies the requirements for processor consistency. In addition to this, the following condition is satisfied:

[p-p, other] Two writes to the same item from different processors are performed in the same order with respect to any *other* processor.

This is clear since writes still are ordered by the root node so that no information of a new item may be distributed until all other writes before that write will be known by any processor when it receives the new value.

Sequential consistency however is no longer maintained. Since the [w-r] condition of sequential consistency not is fulfilled a read that follows a write may appear to be done before the write. A simple example shows this in figure 3.

Scheme 2 differs from sequential consistency in that no conclusion of the ordering of accesses can be drawn from that a write has been executed. Note however that processors observing accesses by reads, all see the same interleaving. It might be interesting to note, that if the algorithm in figure 3 is modified so that the variable written is read again by the writer, correct operation is achieved.

5.2 Scheme 3

Scheme 2 differs from scheme 1 in that a processor may continue execution directly after a write has been issued. It may however neither read the item written nor propagate its value until the acknowledge has been received. If further relaxations are introduced we get a lower consistency level.

We introduce a third scheme, *Scheme 3*, which to a writing processor is equal to scheme 2 except that now the processor may read the new value, as well as distribute it as soon as an invalidation of the item has been issued. Here we assume that an item is the smallest writeable unit.

Now, no ordering of writes exists. ([p-p, other] is not satisfied). The access order from each processor is nonetheless maintained. This scheme is close to processor consistency, but maintains in addition still the causal correctness offered by the network topology.

Scheme 3

- *At a write-miss to global data the processor sends an invalidation of the item to the network.*
- *The invalidation is propagated in the subsystem where the item is exclusive.*
- *The processor may continue to access global data as well as read and propagate the new value of the item as soon as the invalidation has been issued.*

5.3 Comparison with general networks

We have shown two schemes that maintain access-order models that are weaker than sequential consistency but slightly stronger than processor consistency. Implementations of processor consistency in general networks allow reads following a write to bypass the write. Writes can be buffered but must be performed in program order and may only be considered performed after receiving an acknowledge that all other copies are invalidated.

The race-free network allows the processor to have several outstanding writes. That is, the processor never needs to stall on a write operation. The network topology guarantees that the write operations are seen in correct sequential order by all other nodes.

6 Improving performance

A major drawback of the race-free network is that it has a possible bottleneck – the root. A general network can obtain arbitrarily high bandwidth by introduction of redundance (i.e. several parallel paths between two processor nodes). A race-free network may especially have bottlenecks that limit the bandwidth between distant processors since this communication must pass the root node. These drawbacks can be reduced by *splitting* presented below.

6.1 Splitting paths

In a race-free network, (higher) levels in the hierarchy may be *split*. The splitting introduces multiple slices of a path between network nodes to increase communication bandwidth. This was first suggested for a shared bus by Rudolph and Segall in [13]. In a race-free network splitting must be done with caution. With multiple paths, transactions propagating in the network may overtake each other, and the fundamental properties of race-free networks may be lost.

1. Splitting is done so that a unique mapping (interleaving) of transactions is established that assigns different parts of the item space to different *slices* of a path in the network (e.g. transactions associated with items with even addresses take one slice, and transactions associated with odd addresses the other). In this manner, all items see a network with only one single path.

2. The transaction sequence must be maintained so that all transactions issued from one node in the system are received by all other nodes in the same issuance order, or so that the sequence between transactions can be restored to the correct order by the receiving nodes.

The first condition is needed to maintain local coherence. The second is needed to keep the race-free properties of the network. To restore the sequence between transactions if, for example, several transactions are sent in parallel, there must be a unique prioritization between the different slices in the split path so that the order between parallel transactions on different slices can be restored.

A node in a split network can split caches, directory memories, protocol machines etc which may be present in the node. Note however that the FIFO buffering in the node must maintain a FIFO order common to all slices in the node.

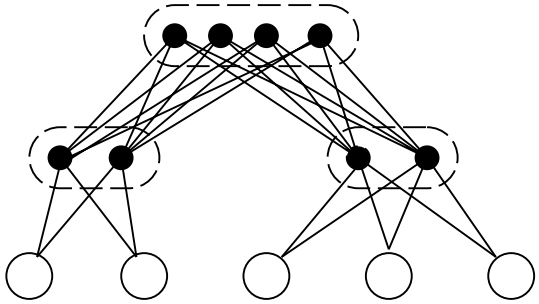

Figure 4: A race-free network with two level splitting. Note that in the split node, the transaction sequence must be maintained, so that the race-free properties not are lost.

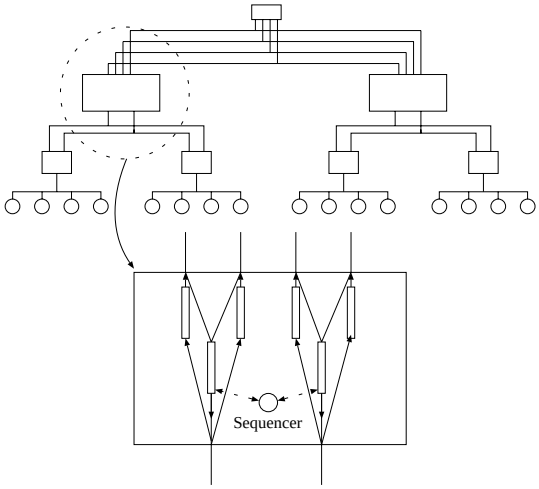

Figure 5: A race-free, hierarchical bus network with splitting for relaxed transaction ordering.

6.2 Relaxing the transaction ordering in Scheme 1

In this section we introduce a way to further improve performance in a split network. The schemes, 1, 2 and 3 rely on three main properties: transaction ordering, causal correctness and local coherence. It turns out that only a part of all transactions need strict ordering.

Here we assume that a coherence protocol is assumed to be based on transactions from the following categories: {Read, Data, Write, Acknowledge, Invalidate}. A Read transaction is a request to get information about the value of an item. A Data transaction is a transaction that carries the value of an item, possibly sent as a reply to a Read. A Write transaction is a request to write an item. An Acknowledge transaction is the reply that grants permission to write. An Invalidate transaction is a demand to erase the information about the value of an item.

In Scheme 1, the critical transactions Invalidate and Acknowledge are propagated only from the root node towards the processors, while Write requests are propagated only in the opposite direction.

If memory is distributed among the processors, transactions Read and Data are propagated in the network in both directions. Transactions propagating up the network, towards the root node can thus be of types {Read, Data, Write}, and transactions propagating downwards can be {Read, Data, Acknowledge, Invalidate}.

To maintain Scheme 1, the requirement that transactions may not overtake each other can be relaxed. In fact all transactions propagating upwards in the network, {R, D, W}, bound for different items can be propagated in an arbitrary order. This requires some justification. Since a processor is suspended during read and write misses, each processor can only have one outstanding R or W transaction. Data transactions are propagated as replies to Read requests, and may be sent out at any time, requests to writes waiting for acknowledge may however not be served until the acknowledge has been received. Two Write requests propagating in the network originate from different processors. The order of these writes is a result of a race condition which is solved by their root nodes. Therefore the ordering of two W transactions is arbitrary and may be altered by the network. The order of transactions of the other types does not change the access ordering as long as they are made to different items, and may thus also be altered by the network. Note though, that all transactions for the same item still must be kept in order. This is not a problem since they propagate along the same slice in a split path.

The relaxed transaction sequence facilitates implementations of split nodes. Now, only transactions sent from a root node to its children need to be ordered. All the transactions from the sons and upwards can be sent in

an arbitrary order. This results in that the available bandwidth can be utilized to a large extent, even with the limitation that each transaction only can use one of the slices to propagate.

Note that this does not apply to schemes 2 and 3, where for example invalidation transactions must be ordered even when propagated upwards.

6.3 Heterogeneous networks

A race-free network may be a part of a larger network. A simple example of this is architectures like for example the DASH [12, 5]. In the DASH architecture a cluster of processors share a bus (which is a very simple race-free network). Such clusters are interconnected with a general network (a pair of meshes).

It is however possible to use larger systems based on race-free networks as subsystems with a general network on top. In this way the benefits of the two strategies can be combined. The number of transactions needed, and the delay time in the general network can be drastically reduced while the power of the high bandwidth network can be fully utilized.

The schemes presented above can be used locally in the race-free subsystems. The top node of each race-free system must then implement a more general scheme to handle the general network. This is particularly useful in Scheme 1 since minimal functionality needs to be added. If the subsystems follow Scheme 1 (sequential consistency) the top-level protocol can be based on common principles with write/acknowledge messages sent between the root nodes. A root node that issues a write must then wait for all other acknowledges to be able to return an acknowledge down its own network. A root node that receives an invalidation from the general network may respond with an acknowledge as soon as it has the invalidations due to the write buffered bound downwards.

Note that schemes 2 and 3 requires buffering of accesses by the root node in the race-free subsystems, which is probably not worth while if global networks accesses are frequent. In Scheme 2 for write ordering to be global, all writes must be buffered at the root node, and need acknowledge messages from all other roots to be performed. In Scheme 3 an optimization may be done, as correct ordering of writes only need to exist between writes from the same processor. If this is done the causal properties are lost.

7 Summary

In this paper, we have defined and studied one class of interconnection networks, race-free networks. We have shown that for different access-order models, such networks require less restrictive conditions on memory accesses than have been used for general networks. These reduce latencies associated with writes to shared global data.

New conditions for sequential consistency have been presented which show that sequential consistency can be maintained if all accesses can be ordered in an acyclic graph. We have shown how this can be done in a race-free network without the need for a transaction to be globally performed before the processor can continue execution of next instruction.

General networks require invalidation acknowledge messages from all other processors that share copies of the item written to allow a writing processor to continue (to access shared global data). In race-free networks, such an acknowledge can be sent out from a network node, and the writing processor can continue execution before all other copies are invalidated. The time a processor is suspended in a general network is $max_i(t_{inv-to-proc_i} + t_{ack-from-proc_i})$. For race-free networks the corresponding time is $t_{inv-to-root} + t_{ack-from-root}$. In tree-structured networks, $t_{inv-to-root} \approx t_{inv-to-proc}/2$, and $t_{ack-from-root} \approx t_{ack-from-proc}/2$. The network traffic is also reduced.

We have also presented two schemes that are close to processor consistency. One guarantees processor consistency and, in addition, that a global write-order can be observed by all *reading* processors. The other guarantees processor consistency and causal correctness. These two models allow reads to bypass writes, and writes to be pipelined (retired from the write buffer as soon as they are issued into the network). This has previously only been achievable for lower consistency levels.

8 Related work

A multiprocessor with a race-free interconnection network is under implementation at the Swedish Institute of Computer Science. The machine is called the Data Diffusion Machine, DDM [9] and has a network based on hierarchical buses. A full protocol for sequential consistency has been presented in [10, 9]. An implementation of a protocol similar to Scheme 2 has been presented in [8].

9 Acknowledgments

We especially want to thank Per Stenström and Fredrik Dahlgren, Lund University, Sweden for many interesting discussions on these topics. Fredrik Dahlgren showed us that a race-free network may be any acyclic graph, which is a wider definition than our previous hierarchical network definition. Per Stenström has also provided us with constructive advice and comments in the process of completing this paper.

SICS is a non-profit research foundation sponsored by the Swedish National Board for Technical Development (STU), Swedish Telecom, LM Ericsson, ASEA Brown Boveri, IBM Sweden, Bofors Electronics and the Swedish Defence Material Administration (FMV).

References

[1] Sarita Adve and Mark Hill, Implementing Sequential Consistency in Cache-Based Systems, In Proc. of the 1990 International Conference on Parallel Processing, August 1990

[2] Sarita Adve and Mark Hill, Weak Ordering – A New Definition, In Proc. of 17th International Symposium on Computer Architecture, May 1990

[3] Sarita Adve and Mark Hill, Weak Ordering – A New Definition And Some Implications, Computer Sciences Technical report #902, University of Wisconsin, December 1989

[4] Michel Dubois, Christoph Scheurich, Faye Briggs, Memory Access Buffering in Multiprocessors, In Proc. of the International Symposium on Computer Architecture, May 1986

[5] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Philip Gibbons, Anoop Gupta and John Hennessy, Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, In Proc. of 17th International Symposium on Computer Architecture, May 1990

[6] Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors, In Proc. of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, April 1991

[7] James R. Goodman, Cache Consistency and Sequential Consistency, Technical Report 61, SCI Committee, March 1989

[8] Erik Hagersten, Anders Landin and Seif Haridi, Multiprocessor Consistency and Synchronization thru Transient Cache States, In Proc. of the Workshop on Scalable Shared-Memory Architectures, Kluwer Academic Publisher, Norwell, Mass, 1991

[9] Erik Hagersten, Anders Landin, Seif Haridi and David Warren, Moving the Shared Memory Closer to the Processors - DDM, Swedish Institute of Computer Science, Technical Report, November 1990

[10] Erik Hagersten, Seif Haridi and David Warren, The Cache-Coherence Protocol of the Data Diffusion Machine, In Proc. of the Cache and Interconnect Workshop, Kluwer Academic Publisher, Norwell, Mass, 1990

[11] Leslie Lamport, How to Make a Multiprocessor Computer That Correctly Executes Multiprocessor Programs, IEEE Transactions on Computers C-28 (1979), 690-691

[12] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anop Gupta and John Hennessy, The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor, In Proc. of 17th International Symposium on Computer Architecture, May 1990

[13] Larry Rudolph and Zary Segall, Dynamic Decentralized Cache Schemes for MIMD Parallel Processors, In Proc. of the 11th International Symposium on Computer Architecture, 1984

[14] Christoph Scheurich, Access Ordering and Coherence in Shared Memory Multiprocessors, PhD thesis, University of Southern California, Technical Report no CENG 89-19. USC, May 1989

[15] Christoph Scheurich and Michel Dubois, Correct Memory Operation of Cache-based Multiprocessors, In Proc. of the 14th International Symposium on Computer Architecture, June 1987

[16] Per Stenström and Fredrik Dahlgren, Personal communication. November 1990

[17] David H. D. Warren and Seif Haridi, Data Diffusion Machine – A Scalable Shared Virtual Memory Multiprocessor. In Proc. of the International Conference on Fifth Generation Computer Systems 1988. ICOT, 1988