

SEQUENTIAL CONSISTENCY REVISIT: THE SUFFICIENT CONDITION AND METHOD TO REASON THE CONSISTENCY MODEL OF A MULTIPROCESSOR-ON-A-CHIP ARCHITECTURE

Yuan Zhang Weirong Zhu Fei Chen Ziang Hu Guang R. Gao
Department of Electrical & Computer Engineering, University of Delaware
Newark, Delaware 19716, U.S.A
{zhangy,weirong,fchen,hu,ggao}@capsl.udel.edu

ABSTRACT

In his seminal paper in 1979 [1] on memory consistency, Lamport proposed two requirements for a multiprocessor system to be sequentially consistent. The second condition stated that memory “requests from all processors issued to an individual memory module are serviced from a single FIFO queue. Issuing a memory request consists of entering the request on this queue”. Recently, the authors have the opportunity to revisit Lamport’s conditions in the context of a design study of the IBM Cyclops multiprocessor-on-a-chip architecture (known as BG/C) from the system software angle. We find that when a multiprocessor system employs a network to communicate with its shared memory modules – such as in the BG/C architecture - we need to carefully elaborate Lamport’s requirements to cover the network. Thus we have refined the Lamport’s second requirement along this line and demonstrated that the revised conditions are sufficient for ensuring the sequentially consistent behaviors for a class of BG/C-like architectures.

KEY WORDS

Sequential consistency, multiprocessor-on-a-chip, memory model

1 Introduction

A memory model defines how memory system behaves in a computer system. It specifies the semantics of memory operations in a program, such as read, write and synchronization operations.

The most widely accepted memory model for the shared memory machine is Lamport’s *sequential consistency* (SC) model, as described in the following statement [1]:

[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 order.

In his 1979 paper cited above, Lamport also illustrated how to satisfy SC in a simple hypothetical shared-

memory multiprocessor system [1]. Lamport’s hypothetical multiprocessor system will be described in more detail in Section 2, but briefly, it consists of a collection of processors and shared memory modules, and the processors communicate with each other only through shared memory read and write operations. Lamport posted the following two requirements as sufficient for ensuring this system to be sequentially consistent:

1. R1: Each processor issues memory requests in the order specified by its program.
2. R2: Memory requests from all processors issued to an individual memory module are serviced from a single FIFO queue. Issuing a memory request consists of entering the request on this queue.

Our work described below is mostly inspired by a suggestion from Henry Warren from IBM T.J. Watson Center [2] to consider a proof of sequential consistency for the BG/C multiprocessor machine designed by Monty Denneau [3]. Denneau’s multiprocessor-on-a-chip architecture employs a crossbar network between the processors and memory modules in a so-called dance-hall configuration [4]. Denneau believes that the base BG/C chip architecture is sequentially consistent and there is no need to issue fence-like instructions after each memory operation to ensure SC.

Although Denneau’s conjecture is, at first glance, quite obvious, it took us much longer to construct a proof to be presented in this paper. A major issue is: a memory operation needs to go through the network between it is “issued” by the processor and it enters the corresponding memory module, instead of as in condition R2 “... Issuing a memory request consists of entering the request on this queue”. Therefore, the proof process needs to show what features of the BG/C on-chip network make it satisfy the requirement of the original Lamport’s R2 condition. We end up with a refined R2 condition that will be illustrated in Section 2.

Another challenge in the proof is to show that the condition R2 (or the refined R2) is sufficient to lead the validity of the widely referred Lamport’s SC definition. In a real multiprocessor system, the lifetimes of memory operations may actually be overlapped - a fact also pointed out

by Lamport. Thus the completion order of memory operations, which determines the system consistency model, may be different from the strict Lamport order. To this end, our proof shows that in the context of the BG/C chip architecture, which satisfies our refined R2 condition, and begin with a total order based on the completion time, we can actually perform a series of “result-preserving” reordering transformations back to a total order based on the issuing order. Therefore, we can show that the base BG/C architecture is sequentially consistent. This method is also applicable to reason the sequentially consistent behaviors of other architectures.

Let us put this paper in a broader context. We have witnessed the emerging technology trend on multiprocessor-on-a-chip architecture with 10s-100s processing/thread units. In the general-purpose parallel computing arena, a representative multiprocessor-on-a-chip architecture is the class of cellular architectures (IBM Cyclops architecture [5, 6, 7] is one example of this class). At the chip level, the cellular architecture employs a decentralized microprocessor design – interconnecting a large number of very light-weight processors called processing cells by a large on-chip network. Such network provides rich interconnection and sufficient bandwidth for interprocessor communication among the processing cells and their shared memory. Another class, in the domain of application-specific architectures, is called *extreme chips* [8]. As many as 300+ RISC processors are interconnected within one chip to perform process communication tasks. Therefore, designing such a chip and establishing its memory consistency requirement have a practical significance.

This paper is organized as follows. Section 2 first revisits Lamport’s two requirements in the context of a multiprocessor with a dance-hall organization, then proposes our refined requirements. In section 3 we first introduce the base IBM BG/C multiprocessor-on-a-chip architecture, then discuss the method to reason its memory model. The proof method is presented in section 4. The related work is listed in section 5. Section 6 summarizes the whole paper’s work and gives the conclusions.

2 R2 Condition Refinement

The architecture model we concern, as shown in Figure 1(a), consists of a collection of processors and memory models, and the processors communicate with one another only through reading or writing shared data in memory modules. Processors and memory modules are connected by a single bus or general network. We called it a “dance-hall” configuration (much inspired by past literature).

Let us first examine the sufficient condition for this system to be sequentially consistent. As mentioned in the previous section, Lamport has proposed two requirements in 1979, and the second requirement is “R2: Memory requests from all processors issued to an individual memory

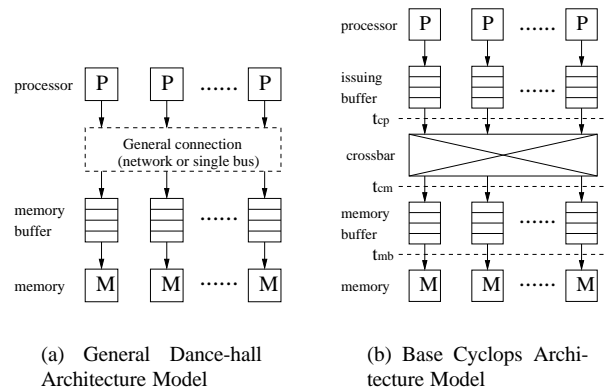


Figure 1.

module are serviced from a single FIFO queue. Issuing a memory request consists of entering the request on this queue.” R2 implies that an operation is *issued* only when it has reached the memory buffer. When the R2 condition was explained using his hypothetical multiprocessor system, Lamport did not elaborate what requirements the network should follow in order to ensure the sequential consistency.

In real architecture models, networks cause some delay in transmitting a memory request, i.e. a delay from the time it is issued by the processor to the time it arrives at the destination memory queue after traveling through the network. If no restriction on the distribution of such delay is imposed, a network design may violate the R2 condition. Let us consider the following example:

```
Initially x = flag = 0

P1:                                     P2:

i1: x = 1;                               i3: while(flag==0) {;
i2: flag = 1;                             i4: read x;
```

We expect that in most “reasonable” memory models *i4* returns $x = 1$. Programmers may reason this example (say, under Lamport’s hypothetical multiprocessor model) as follows. We know that *i1* is issued before *i2*, and *i3* is issued before *i4* - from the condition R1. Since the *while* loop in *i3* exits only when the check of *flag* returns value 1, *i2* must have been performed at that time. Consequently, when *i4* is issued, *i1* has already been issued. Thus *i4* will be performed after *i1* and return the value written by *i1*.

However, suppose in a real dance-hall system x and *flag* are in different memory modules, and the delay occurs in the network such that although *i1* is issued to the network before *i4*, it may arrive at the memory buffer for x later than *i4*. If this happens, then *i4* returns the wrong value 0 - a contradiction to sequential consistency!

Since Lamport’s R2 condition does not rule out this possibility, we are inspired to re-consider the sufficient requirements for a general dance-hall system to be sequen-

tially consistent, and propose the following refined R2 requirement:

R2-refined: Two operations designated to the same memory model M will be delivered to M 's FIFO queue in the same order as they enter into the network.

We will prove later that, in the context of a class of multiprocessor-on-a-chip architecture such as the IBM BG/C architecture, R1 and R2-refined together are sufficient to ensure the sequentially consistent behavior of a general dance-hall system.

3 Cyclops Consistency Model

3.1 Target Architecture Model

The base Cyclops multiprocessor-on-a-chip architecture model is illustrated in Figure 1(b). The extended model, as well as its design and implementation details, can be found at [7, 9].

The base model is composed of a collection of processors and memory banks connected by a crossbar. Each processor issues its memory operations in program order into a FIFO queue called the “issuing buffer”.¹

The crossbar provides the following nice property, which we name as the “equal latency” property. For all memory operations targeted to the same memory bank M , regardless their issuing processors, the time they take to travel the network and reach M 's memory buffer is a constant - hence the term “equal latency” property². If two memory requests having the same memory destination are ready to be issued at the same time, one of them, determined by arbitration, will be stalled at its issuing buffer until the conflict is resolved.

The programming model is rather simple – only “read” and “write” operations are considered.

3.2 Problem Formulation

As described in the introduction, it has been conjectured that the base Cyclops architecture *obeys* the sequential consistency, and this is true even without using an explicit “sync” or “fence” [10] instruction after each read or write instruction. The problem is: *can we establish the validity of this conjecture and how?* Equivalently, since Cyclops base architecture model satisfies both the R1 and R2-refined conditions - from the above specifications, *can we prove, using Cyclops as an example, that R1 and R2-refined*

¹In fact, a memory request cannot be issued unless its uniprocessor data dependence and control dependence are satisfied.

²The implemented Cyclops hardware guarantees that it takes the same amount of time for all memory operations, whatever their source processors and target memory models, to travel across the crossbar. This is a special case of the Cyclops base model, therefore all the conclusions still hold.

together constitute a sufficient condition for a system to be sequentially consistent?

Before we proceed to present our proof, we wish to discuss why the answer, although might look intuitive, is not trivial.

There are two conditions indicated in the *classical* sequential consistency (SC) definition cited in section 1:

- (1) Operations of all processors are executed in some sequential order (i.e., a total order).
- (2) In that total order, operations from an individual processor are executed in the program order.

We call the total order which satisfies condition (2) the “Lamport order”. Thus the most straightforward way to prove a system to be sequentially consistent is to find out this Lamport order.

In our Cyclops base system, it is natural to build a Lamport order λ based on all memory operations' issuing time t_{cp} (please refer to Figure 1(b) for the definition of the t_{cp}). Representing all operations executed in a parallel program $Prog$ as a multi-set Ψ , the issuing order is mathematically defined as:

$$\lambda = \{ \langle i, j \rangle \mid i \in \Psi \wedge j \in \Psi \wedge (t_{cp}(i) < t_{cp}(j) \vee (t_{cp}(i) = t_{cp}(j) \wedge Mem(i) \leq Mem(j))) \}$$

where $Mem(i)$ shows which memory bank the operation i accesses, assume every memory bank has a unique ID³.

It is clear that a system obeying this order is sequentially consistent. However, because the memory operations begin to take effect only when they are performed by memory modules, and because the crossbar will “reorder” the memory operations going to the the different memory modules by different delays, we have another total order based on memory operations' performing time t_{mb} , mathematically defined as:

$$\chi = \{ \langle i, j \rangle \mid i \in \Psi \wedge j \in \Psi \wedge (t_{mb}(i) < t_{mb}(j) \vee (t_{mb}(i) = t_{mb}(j) \wedge Mem(i) \leq Mem(j))) \}$$

In Theorem I of section 4 we will prove that those two orders - issuing order λ and performing order χ are different, and a strict program order is not guaranteed in χ . Thus χ is not a Lamport order.

It is our belief that it is often easier for a programmer to think about the order of memory operations characterized by the time when they leave a processor, while it is the performing order that determines the system consistency model. Thus our job is to prove that in the context of Cyclops base architecture, although the performing order is different from the issuing order, it generates the same execution results, thus preserves the sequentially consistent behavior of the system. For this purpose, we introduce the following concepts:

Definition 1: An **execution** of a parallel program $Prog$ at system S is an (M, O) pair, where M is the multi-set of operations executed, and O is an order on M .

³Although in a real system two operations with the same t_{cp} may be executed simultaneously (by different memory banks), forcing an order here neither loses any generality nor influences the validity of the proof and conclusions.

Definition 2: An execution (M, O) is a **strict sequentially consistent execution** if O is a Lamport order.

Definition 3 Two executions (M_1, O_1) and (M_2, O_2) of a parallel program $Prog$ are **equivalent** if:

- (1) They have the same set of memory operations, *i.e.*, $M_1 = M_2$ ¹;
- (2) Any memory read operation in (M_1, O_1) returns the same value as (M_2, O_2) , and vice versa;

It is obvious that an execution is equivalent to itself.

Based on above definitions, sequential consistency can be equivalently defined as:

Definition 4: A shared-memory system S is **sequentially consistent** if any execution of a program $Prog$ on S is equivalent to a strict sequentially consistent execution of $Prog$.

From **Definition 2** we know (Ψ, λ) is a strict sequentially consistent execution. In the next section, we will begin from the total order χ , and perform a series of “result-preserving” reordering transformations back to the total order λ . Thus by **Definition 3** we can prove that the base Cyclops architecture is sequentially consistent.

4 Proof

The main body of this section consists of **Theorem I**, **Theorem II** and their proofs. **Theorem I** and **Theorem II** are listed as following:

Theorem I The performing order χ is not a Lamport order.

Theorem II Cyclops architecture is sequentially consistent.

Theorem I: The performing order χ is not a Lamport order.

Proof:

Suppose two memory operations i and j issued from the same processor and $i \prec j$, where \prec denotes the program order, then $t_{cp}(i) < t_{cp}(j)$. i targets the memory location x in memory bank M_x , and j targets the memory location y in memory bank M_y . Also suppose:

1. the crossbar delay to memory bank M_x and M_y is $C(M_x)$ and $C(M_y)$, respectively;

¹The need for this condition may not be obvious. It is needed when there are some reads whose return values determine whether an operation is executed or not, or what memory address will be accessed. Let us look at the following example:

```

r1 = read x;
if(r1 == 0)
    write y = 1;
else
    write z = 1;

```

Here the return value of “read x” determines whether then-branch or else-branch is executed. If two executions have different return values of “read x”, they are not equivalent because they have different *execution paths*.

2. there are already W_x operations waiting at M_x 's buffer, and W_y operations waiting at M_y 's buffer;
3. in average it takes S_x units of time and S_y units of time for M_x and M_y , respectively, to process a memory request;

then:

$$t_{mb}(i) = t_{cm}(i) + W_x * S_x = t_{cp}(i) + C(M_x) + W_x * S_x,$$

and

$$t_{mb}(j) = t_{cm}(j) + W_y * S_y = t_{cp}(j) + C(M_y) + W_y * S_y.$$

Suppose $W_x = W_y$, $S_x = S_y$ and $C(M_x) > C(M_y)$ (this probably happens in real machine), we probably have $t_{mb}(i1) > t_{mb}(i2)$. That means, i will be performed after j in χ , and the original program order is broken. Therefore χ is not a Lamport order.

Proof Done.

Our proof of **Theorem II** proceeds as follows: According to **Definition I** in section 3, the “execution” is conceptually represented as the (Ψ, χ) pair. We also know the “pseudo” execution (Ψ, λ) is a strict sequentially consistent execution. In **Lemma 2**, we show that the order χ can be transformed – through a series of reordering steps that preserve “equivalence” (based on **Lemma 1**) under the guidance of order λ – into a Lamport order, hence (Ψ, χ) is equivalent to a strict sequentially consistent execution.

Before reading the proof, readers can think of the execution (Ψ, λ) as executing Ψ on a “ideal Lamport machine”, in which operations are executed one by one in a sequential order. For each operation its issuing time is its performing time, therefore there is no lifetime overlapping between any pair of operations. This case is illustrated in Figure 2(a). However, in Cyclops, memory operations may be out-of-order executed, because of network delay; their lifetimes are overlapped with each other. The order based on their performing time is χ , which is different from λ .

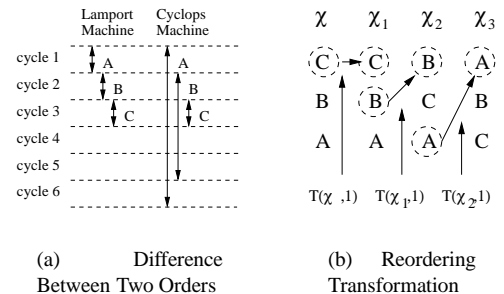


Figure 2.

In order to prove (Ψ, λ) and (Ψ, χ) are equivalent, we introduce a reordering transformation T . $T(\chi, i)$ transforms χ into χ' by “percolating” the i th operation $O(i)$ in χ

upward and insert it in a position that do not violate the corresponding ordering specified by λ . Set $\chi_0 = \chi$ and begin from $i = 1$, we apply a sequence of T s on χ with increasing values of i , *i.e.*, $\chi_1 = T(\chi_0, 1)$, $\chi_2 = T(\chi_1, 2)$, \dots , $\chi_n = T(\chi_{n-1}, n)$. Finally we reach the order $\chi_n = \lambda$. We will proof that $\chi = \chi_1 = \dots = \chi_n = \lambda$ (where “=” denotes equivalence).

Readers will notice that after $\chi_i = T(\chi_{i-1}, i)$, the order of the first i operations in χ_i are consistent with λ , and the rest operations are ordered in the same way as χ . Figure 2(b) gives a simple example of this series of transformations. We also assume that the multi-set Ψ is changed to Ψ' due to T . We will proof later that $\Psi = \Psi'$.

Lemma 1: Execution (Ψ', χ_i) is equivalent to (Ψ, χ_{i-1}) , after the transformation $\chi_i = T(\chi_{i-1}, i)$, which reorders $O(i)$ in χ_{i-1} .

Briefly, this proof is proceeded by contradiction. We first assume that reordering operation $O(i)$ in χ_{i-1} , which is either read or write, will cause at least one read operation to return different value. Then a conflict with an obvious fact will occur. Readers can skip the following part if they have no interest in the proof details.

Proof:

let us first check the second condition in **Definition 3** – all read operations in χ_i return the same results as in χ_{i-1} . Suppose the i th operation $O(i)$ in χ_{i-1} is moved backward to position j in χ_i , $0 < j \leq i$. We consider two cases:

Case #1: $O(i)$ is a read operation;

Case #2: $O(i)$ is a write operation;

Case #1 is proved by contradiction. We assume that $O(i)$ reads a different value after it is reordered. Suppose in χ_{i-1} , $O(i)$ returns the value written by a write operation $O(k)$, as shown in Figure 3, then $k < i$. It means:

$$t_{mb}(O(k)) < t_{mb}(O(i)) \text{ or}$$

$$(t_{mb}(O(k)) = t_{mb}(O(i)) \wedge Mem(O(k)) \leq Mem(O(i)))$$

Since $O(k)$ and $O(i)$ access the same memory bank, $O(k)$ must arrive earlier than $O(i)$ at the memory bank, we have:

$$t_{mb}(O(k)) < t_{mb}(O(i)) \quad (1)$$

And only when $O(i)$ is moved to a position before $O(k)$ in χ_i can $O(i)$'s value be changed. “ $O(i)$ is moved to a position before $O(k)$ ” means:

$$t_{cp}(O(i)) < t_{cp}(O(k))$$

therefore,

$$t_{cp}(O(i)) + C = t_{cm}(O(i)) < t_{cp}(O(k)) + C = t_{cm}(O(k))$$

Because each memory bank has a single buffer, $t_{cm}(O(i)) < t_{cm}(O(k))$ is equivalent to

$$t_{mb}(O(i)) < t_{mb}(O(k)) \quad (2)$$

But, (1) and (2) are in conflict, a contradiction. Thus $O(i)$ still returns the same result. Note other read operations'

values cannot be changed by reordering $O(i)$, all read operations return the same values before and after the reordering.

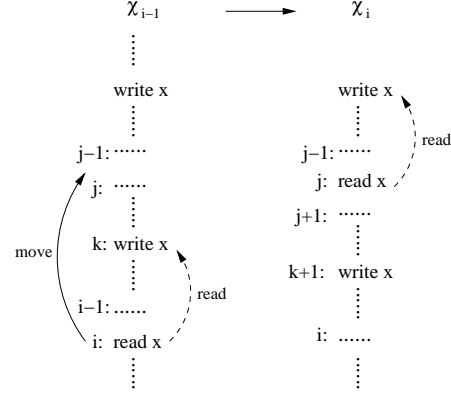


Figure 3. Case #1

For Case #2, $O(i)$ is a write. Again, proof proceeds by contradiction. We consider two sub-cases below.

Sub-Case #1: Assuming that there is a read $O(q)$ later than $O(i)$ ($q > i$) in χ_{i-1} that will read a different value after the reordering. It implies the existence of a write $O(p)$, $i > p \geq j$, as shown in figure 4(a). By reordering $O(i)$ to position j in χ_i , $O(i)$'s value will be killed by $O(p)$ and then $O(q)$ returns $O(p)$'s value, instead of $O(i)$. The proof of a contradiction can follow a similar argument as in Case #1 by comparing t_{mb} and t_{cp} of $O(p)$ and $O(i)$.

Sub-Case #2: Assuming that there is a read $O(q)$ earlier than $O(i)$ ($q < i$) in χ_{i-1} that will read a different value after the reordering. It implies that on χ_{i-1} , there is a write $O(p)$, $p < j$ and $j \leq q < i$, as shown in figure 4(b), and $O(q)$ returns the value written by $O(p)$. By reordering $O(i)$ to position j in χ_i , $O(i)$ will kill $O(p)$ so that $O(q)$ will return $O(i)$'s value. Again and similarly, this can generate the contradiction.

Now we can draw the conclusion that in χ_i all reads return the same values as in χ_{i-1} .

Now let us look at the condition 1 in **Definition 3**. Because all reads on the first i operations in χ_i return the same values as in χ_{i-1} , any if branch after i operations which depends on the read values in the first i operations will have the same execution path in χ_{i-1} . This guarantees that χ_i and χ_{i-1} execute the same operations, *i.e.*, $\Psi' = \Psi$.

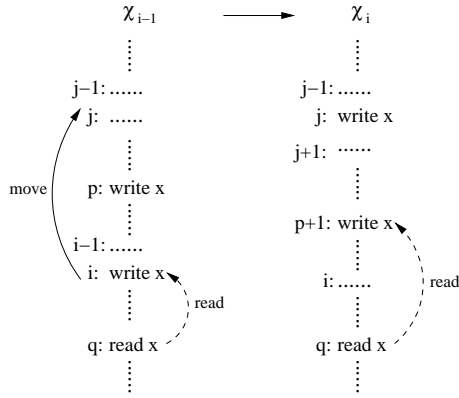
Therefore both conditions in **Definition 3** are satisfied, thus (Ψ', χ_i) is equivalent to (Ψ, χ_{i-1}) .

Proof Done.

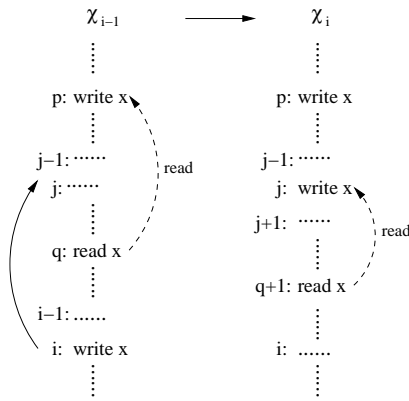
Lemma 2 (Ψ, χ) is equivalent to a strict sequentially consistent execution.

Proof:

Because $\chi = \chi_1 = \chi_2 = \dots = \chi_{n-1} = \chi_n = \lambda$, then $\chi = \lambda$. We know (Ψ, λ) is a strict sequentially consistent execution. From **Definition 3**, we know that (Ψ, χ) is equivalent to a strict sequentially consistent execution.



(a) Case #2 - Subcase #1



(b) Case #2 - Subcase #2

Figure 4.

Proof Done.

Theorem II: Cyclops is sequentially consistent.

Proof:

Lemma 2 can be applied to every execution of a parallel program *Prog* at Cyclops. According to **Definition 4**, Cyclops is sequentially consistent.

Proof Done.

This proof method is not restricted to reason the memory model of BG/C architecture, but also suitable for other multi-processor-on-a-chip system. As a summary, we outline the main steps to prove a shared memory system to be sequentially consistent:

1. Figure out the total order which determines the property of the memory model;
2. Find out a Lamport order, hence a strict sequentially consistent execution. We believe there is always such a Lamport order in the system if all memory operations are issued from processors in program orders;

3. Prove the total order in step 1 is equivalent to the Lamport order in Step 2, using the above reordering transformation.

5 Related Work

Memory consistency model is an intensively studied field in parallel computer architecture, with a large amount of research work published in the literature. Sequential consistency was first defined by Lamport [1, 11] for shared-memory multiprocessor system with network but no cache. Afterwards, Scheurich and Dubois proposed a sufficient condition for sequential consistency at a cache-based system [12, 13]. Shasha and Snir also proposed a software algorithm to ensure sequential consistency [14].

Besides, a lot of research work were conducted to relax the strong condition of sequential consistency to allow more performance optimization. They include processor consistency [15], weak consistency [16], release consistency [10], etc. All of the above discuss the memory model from the system point of view. Adve proposed two memory models from the programmer's point of view [17, 18], and stated that if software obeys the synchronization model defined by the memory model, then the hardware appears sequentially consistent. Gao and Sarkar proposed the location consistency model (LC) and a cache coherence protocol in 2000 [19], which does not rely on the memory coherence assumption. Almost every work mentioned above takes a shared memory system as an example to the presented memory model, thus distinct from our job - exploit a existing system's consistency model.

At the context of memory consistency model proof, Lamport proposed a method based on logical clock and time [20]. Based on Lamport's work, Plakal proposed a reasoning technique to verify a directory cache coherence protocol [21, 22].

6 Conclusions

In this paper we specify the memory model of Cyclops multiprocessor-on-a-chip architecture (known as BG/C). We first check Lamport's two requirements for a dance-hall architecture to be sequentially consistent. We find that Lamport's two requirements R1 and R2 need to be carefully elaborated when we take into account the network delay. Then we proposed the revised requirement - R2-refined. We informally prove that the base Cyclops architecture, which satisfies both R1 and R2-refined conditions, obeys sequential consistency. The proof method is general for all BG/C-like architectures.

7 Acknowledgments

We would like to acknowledge Monty Denneau and Henry S. Warren from IBM T.J. Watson Research Center for their suggestions on this topic and the many communications

without which the progress of this research would not be possible. We also thank useful discussions from members of the CAPSL group at the University of Delaware, in particular Hongbo Rong, Juan del Cuvillo and Andres Marquez. Finally, the last author wish to acknowledge the support in part by NSF, under the NGS grant 0103723, DOE, under grant number DE-FC02-01ER25503, DARPA, under the HPCS program, and other funding agencies.

References

- [1] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, September 1979.
- [2] Henry S. Warren. Personal communication, February 2004.
- [3] Monty Denneau. Personal communication, February 2004.
- [4] Jaswinder Pal Singh David E. Culler and Anoop Gupta. *Parallel Computer Architecture, a Hardware/Software Approach*. Morgan Kaufmann, 1998.
- [5] George S. Almasi, Călin Cașcaval, José G. Castañón, Monty Denneau, Wilm Donath, Maria Eleftheriou, Mark Giampapa, Howard Ho, Derek Lieber, José E. Moreira, Dennis News, Marc Snir, and Henry S. Warren, Jr. Demonstrating the scalability of a molecular dynamics application on a petaflops computer. *International Journal of Parallel Programming*, 30(4):317–351, August 2002.
- [6] George S. Almasi, Daniel K. Beece, Ralph Bellofatto, Gyan Bhanot, and EtAl. Blue gene/l, a system-on-a-chip. In *2002 IEEE International Conference on Cluster Computing (CLUSTER 2002)*, Chicago, IL, U.S.A., 2002. IEEE Computer Society.
- [7] Calin Cascaval, Jos G. Castaos, Luis Ceze, Monty Denneau, Manish Gupta, Derek Lieber, Jos E. Moreira, Karin Strauss, and Henry S. Warren Jr. Evaluation of a multithreaded architecture for cellular computing. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, Boston, Massachusetts, USA, 2002.
- [8] Steven J. Vaughan-Nichols. Vendors go to extreme lengths for new chips. *Computer*, pages 18–20, Jan 2004.
- [9] Juan B. del Cuvillo, Ziang Hu, Weirong Zhu, Fei Chen, and Guang R. Gao. *Toward a Software Infrastructure for the Cyclops64 Cellular Architecture*. Department of Electrical and Computer Engineering, University of Delaware, Newark, Delaware 19716, caps technical memo 55 edition, April 2004.
- [10] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, Seattle, Washington, May 1990.
- [11] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.
- [12] Christoph Scheurich and Michel Dubois. Correct memory operation of cache-based multiprocessors. In *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pages 234–243, Pittsburgh, Pennsylvania, June 1987.
- [13] C. E. Scheurich. *Access Ordering and Coherence in Shared Memory Multiprocessors*. PhD thesis, University of Southern California, 1989.
- [14] Dennis Shasha and Marc Snir. Efficient and correct execution of parallel programs that share memory. *ACM TOPLAS*, 10(2):282–312, April 1988.
- [15] J. R. Goodman. Cache consistency and sequential consistency. Technical Report 1006, Department of Computer Science, University of Wisconsin, Madison, February 1991.
- [16] Michel Dubois, Christoph Scheurich, and Faye Briggs. Memory access buffering in multiprocessors. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 434–442, Tokyo, Japan, June 1986.
- [17] Sarita V. Adve and Mark D. Hill. Weak ordering—a new definition. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 2–14, Seattle, Washington, May 1990.
- [18] Sarita V. Adve and Mark D. Hill. A unified formalization of four shared-memory models. *IEEE Transactions on Parallel and Distributed Systems*, pages 613–624, June 1993.
- [19] Guang R. Gao and Vivek Sarkar. Location consistency - a new memory model and cache consistency protocol. *IEEE Trans. on Computers*, 49(8):798–813, August 2000.
- [20] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [21] Anne Condon, Mark D. Hill, Manoj Plakal, and Daniel J. Sorin. Using lamport clocks to reason about relaxed memory models. In *Proceedings of the Fifth International Symposium on High-Performance Computer Architecture (HPCA1999)*, pages 270–278, Orlando, FL, USA, January 1999.
- [22] Manoj Plakal, Daniel J. Sorin, Anne Condon, and Mark D. Hill. Lamport clocks: Verifying a directory cache-coherence protocol. In *Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA98)*, pages 67–76, Puerto Vallarta, Mexico, June 1998.