Hardware barrier synchronization overview

Elma Hurtic, Swann Levasseur
EDA282, Parallel Computer Organization and Design
Chalmers University of Technology, Göteborg, Sweden
March 2013

Abstract

In this paper, we will first introduce the motivation behind hardware barrier. Second, we will describe different ways of implementing hardware barrier synchronizations, according to the network topology and hardware support. Then we will present some examples of machines using hardware barrier synchronization (e.g. Cray TE3 [14], IBM BlueGene/Q [6]) and finally we will give our opinion on what could be the future of barrier synchronization in massively parallel computers.

Introduction

Over the past decades, parallel machines never ceased to grow and becoming nowadays supercomputers. Massively parallel computers, integrate thousands of Chip MultiProcessor (CMP), in order to execute parallel applications and solve complex scientific problems, orders of magnitude faster than classic computers. Such parallel applications are based on a multitude of concurrent threads, each executing a small part of the total algorithm. To be viable, parallel algorithms need to be sequentially consistent, this means that even if operations are performed concurrently, the program will keep the same execution order and the same atomicity. One way to ensure such condition is to use barrier synchronization. A barrier synchronization can be describe as a logical controller that delimits two parallel code sections and thus controls the algorithm flow. Theoretically, a barrier synchronization stops all concerned threads that enter the barrier until all threads have reached it. In MIMD supercomputers, barrier synchronization tends to suffer form the scalability problem, since it needs to synchronize thousands of threads spread across even more Processing Units. CMP also starts to suffer from the scale problem, chips with dozens of cores are designed and need to be synchronized at high speed. Since programmers are likely to use fine grained parallelism and designers tend to put more and more cores in chips, barrier operations become progressively critical in parallel operations.

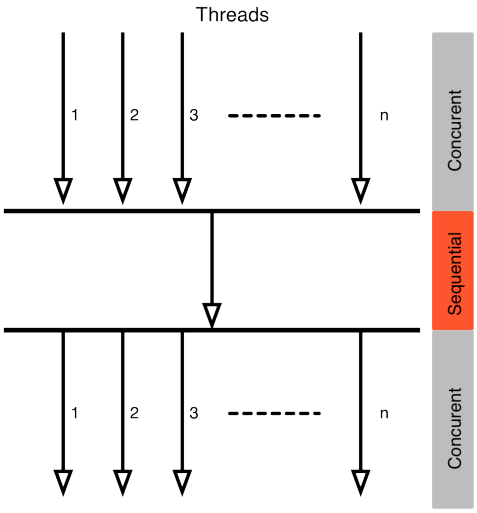


Figure 1: Barrier theory

Practically a barrier synchronization comprises three phases: configuration, reduction and distribution. The configuration phase consist in setting which processor will be involved in a specific barrier and when, this phase can be dynamic (executed while executing the program) or static (executed during the initialization of the program).
The reduction is seen as the action of waiting until all threads have entered the barrier, while the distribution is seen as the action of signaling to each thread that the barrier is completed. In large computers, a barrier synchronization requires to use a complex network and one or more of the participating processing units needed to arbitrate the barrier.
There are three main types of barrier [13], SoftWare (SW) or memory based barriers [9], Hybrid (H) or network based barriers [8], [11] and HardWare (HW) or global line based barriers [10], [15], [17], [18]. Software based barriers involve a shared variable that is atomically incremented until it reaches the "barrier number" (number of participating threads). This kind of barrier is fairly easy to implement (since it is implemented in software), it is inexpensive and has no limitation to the number of barrier executed simultaneously. However, it requires the use of the processor runtime to access the shared memory through the data network, this requires coherence mechanisms and creates more bus traffic. Those factors can lead to a synchronization time an order of magnitude larger than other barrier types.
A hybrid barrier can be described as a barrier which uses hardware supports but still uses the computer data network by adding some features to the network nodes. The hardware support is seen as a peripheral unit which proceed the barrier concurrently to the processor, it can by configurable or fixed. Hybrid barriers have the advantage to unload the processor and to process the barrier slightly faster. Nevertheless, it still creates more bus traffic which will slow down the barrier operation and since the barrier is in hardware, it results in less flexibility and scalability.
Hardware barriers can be described as barriers that use hardware supports and a specific network (or control network) to proceed. Such barriers are the subject of this paper and are discussed in the coming sections.

1 Motivation

An increasing number of parallel applications extensively use barrier synchronizations. Furthermore, in MIMD machines those synchronizations do not involve all threads each time and several synchronizations need to be issued concurrently. Consequently the synchronization overhead induced by each barrier's latency could represent a significant part of the total execution time. If an application strongly relies on synchronization, it will tend to be critical. Thus, using a software based barrier would strongly decrease the performance of the application due to the great memory operation latency. Using a hybrid based barrier would tend to increase the congestion overhead on the bus, which could also result in a global performance reduction. José L. Abellán et al. have shown that in certain cases the overhead can be up to 20% of the total application runtime [1].
Regarding this first observation and comparing it to the two previously cited barrier mechanisms (hybrid & software), we can identify an important bottleneck, the network. A preferred barrier mechanism, in this case, would reduces the traffic and processor overhead.
A second bottleneck would be caused by CMPs, Moore's law tells us that the number of transistors in a chip double each 18 months, and thus designers be likely to create many core architectures with several dozens of them. The arise of many core chips, expresses the problem of the size of such architecture.
In nowadays chips, wires physical properties dictate the speed of a given system, and it becomes fairly hard to reach one side of the chip from another in one clock cycle. Moreover, the silicon real estate is an important limit here. Accordingly to those extents a software based barrier would have been the best choice here since it is fully synchronized (no wire delay problems) and it is not hardware dependent (no real estate needed). However, the software based barrier still suffers from its long latency and could result in a great performances loss. The preferred synchronization mechanism would tend to be fairly simple (small footprint in the system size) and be able to handle on-chip fast long distance communication.
It is to solve the previous problems that hardware based barriers have been created. A hardware based barrier would use a specific separated network and will therefore not influence the data network congestion. Since this network would only be devoted to the barrier mechanism, no complex logic would be needed to arbitrate the communication.
In the next section we will describe and compare different ways of implementing hardware barrier synchronization according to their intrinsic parameters.

2 Implementation of HW barrier

2.1 Classic HW barrier

Starting from the previous definition of a hardware based barrier, it is easy to draw a classic scheme of what a HW support would be. The easiest way to implement a HW barrier is a *wired AND* or *wired OR*. The *wired AND* barrier consist of an AND gate and wires connecting the gate with the involved threads. Each thread has a line connected to one of the entries of the gate, it would be the reduction phase, and the output signal would be the distribution phase. When all threads have set their lines to *high,* to signify that they have reached the barrier, the gate generates the distribution signal and makes threads pass the barrier. The *wired OR* barrier is fairly similar to the *wired AND* except that the steady state of each line of the reduction phase is *high*, and the distribution signal is

also inverted such that a *high* signal signifies the steady state and a *low* signal signifies that the barrier is done. Figure 2 shows schematics of *wired AND* and *Wired OR*.

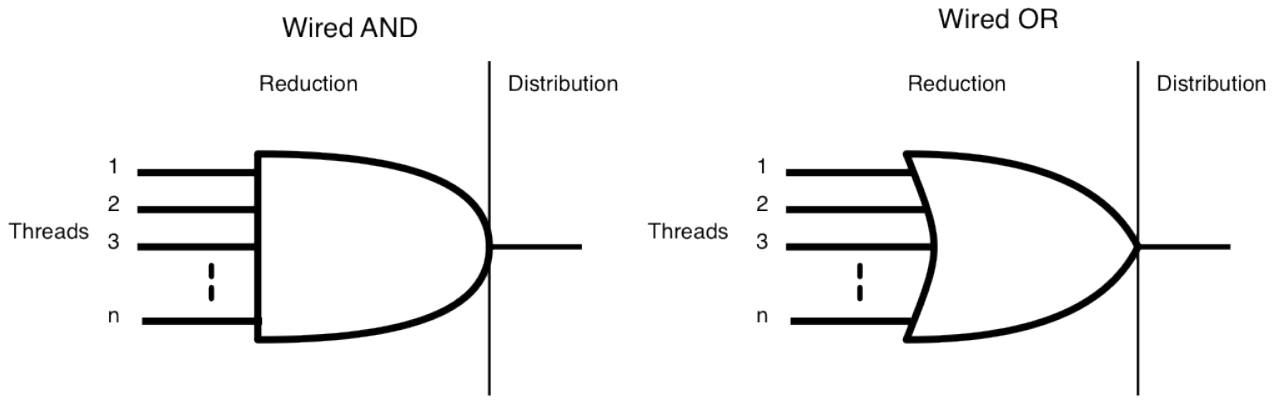


Figure 2: Wired AND, OR

This type of barrier is absurdly simple, it only needs few electronics and if we consider a small system (n < 10 processors) it can be fairly fast. Now, regarding those basic HW barrier we can directly identify some obvious limitations. This HW barrier is hardwired, it is impossible to change the topology and the repartition of the barrier, which means that this barrier type is fully static. All threads are involved in each barrier synchronization, that would create overhead in non directly involved threads. It is not deadlock-free, if a processor stalls and never enters the barrier, all the system will crash. Since it is hardwired, each single barrier needs its own wired network, an *AND* or *OR* gate. This is a major limitation, since the system needs to support multiple concurrent barrier synchronizations. If we consider the gate's output itself, it is impossible to deal with a line loaded by thousands of processor lines, it would take ages, even with a "super" buffered gate, to load or unload it.
To finish on the classic HW support, it does not scale to large computers, and does not improve the existing software or hybrid barrier. It would only have the advantage of being fairly simple and being really fast with a few threads to synchronize. This kind of support has been used in the first implementation of HW barrier, when limitations were not a bottleneck. Since processors were fairly slow, even a "slow" HW barrier was still faster than a SW barrier. The next sections of this paper will show how we can improve the classic barrier by using different network topologies, strategies, and improved hardware support.

2.2 Nowadays hardware support

The first piece of the hardware barrier is the hardware component added to the processor. Hardware supports, as described in the previous section have been as simple as an *AND* gate but to overcome their obvious limitations, nowadays hardware support integrates lots of features. Fundamentally, the "soul" of the *wired AND* or *wired OR* is still here, but it has been upgraded to meet the new requirements that supercomputers impose. The hardware support of today has to be fast, reconfigurable at anytime to support the dynamicity of the network, avoid deadlocks, and it also has to be sufficiently simple to be used extensively on CMP. In this subsection we will describe three examples of hardware support that shows the evolution of the simple *wired AND*. The Cyclical Cascade Chains (CCC) [2], the Distributed Virtual Bit-Slice Synchronizer (DVBSS) [3], and TLSync [5].

The Cyclical Cascade Chains [2] has been designed by T. A. Johnson and R. R. Hoare in 2001. It offers a simple evolution of the *wired AND* and solves some classic issues. The CCC is based on the *wired AND* and thus still uses an AND gate to proceed to the barrier synchronization.

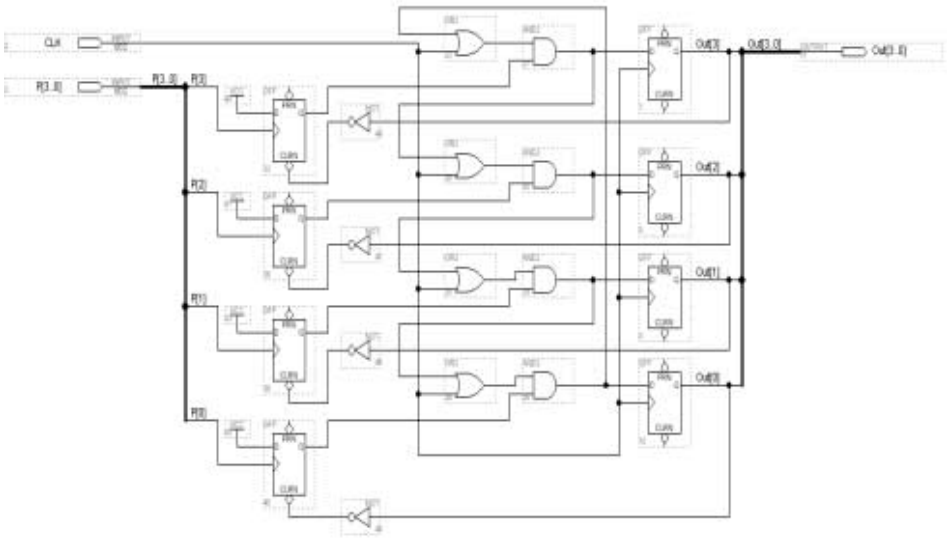


Figure 3: Static CCC [2]

As you can see on Figure 3, each AND gate first input is preceded by an OR gate, its first input is connected to the barrier clock and the second one is connected to the previous AND gate's output. This is called the cyclical chain, it ensures the intercorrelation of all AND gates and ensures that the barrier will be passed at the same time by all threads. The FlipFlops (FF) ensure the stability of signals in the barrier block, and make it synchronous with the falling edge of the clock. Each barrier output is connected to the reset of the FF through an inverter, it is an automatic reset mechanism. When a thread commits to the barrier, it sends a pulse on its line, the pulse is saved in the corresponding flipflop, then at the next clock cycle the corresponding AND gate's output is raised. If all threads have committed, the cyclical chain is completed, the barrier is passed, and the automatic reset cleans the FF. You may have noticed that in this example all threads have to participate in each barrier, this is a serious handicap, but it can be fixed by changing the chain from static to dynamic.

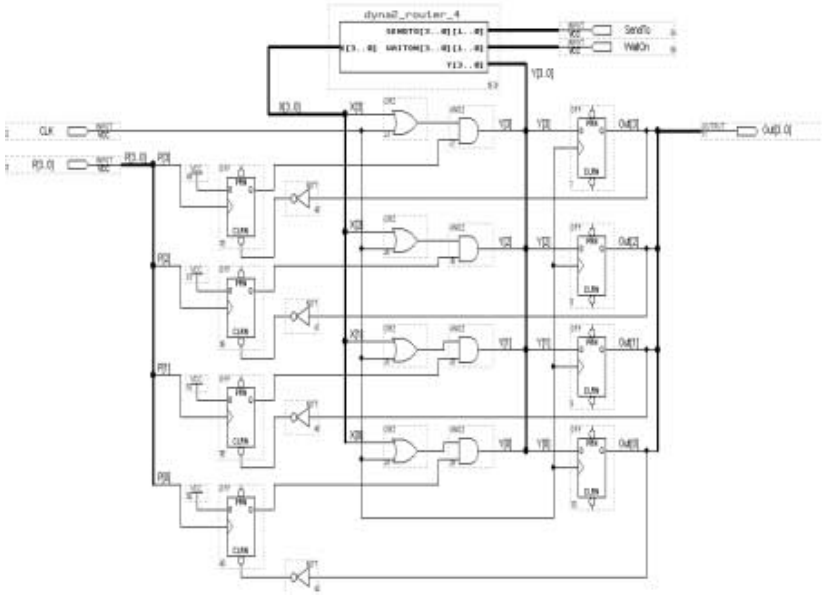


Figure 4: Dynamic CCC [2]

Figure 4 shows the dynamic version of the CCC. This version uses a small crossbar controlled by two input vectors used to choose which thread participates in which barrier, thus the chain can be dynamically wired. By supporting dynamic reconfiguration this HW barrier can be partitioned in several sub-barriers and therefore support multiple concurrent barrier synchronizations. By using a binomial tree, this barrier can be easily scaled up to 128 threads and be relatively simple and fast. The biggest drawback would be that this barrier does not fully support concurrent barrier, it can be partitioned but it cannot overlap barrier on the same HW. The CCC is to be reserved for small architectures with a low number of threads (≈ less than 512), however it proposes a simple optimized alternative to the *wired AND* by making it more flexible.

The Distributed Virtual Bit Slice Synchronizer [3] has been designed by Igor Valerievich Zotov in 2010, it provides an evolved HW barrier that strongly improves the possibilities of the classic *wired AND*. We will present this barrier in context of a 2D mesh control network (to make it simpler) but it can fully support N dimensions configuration.

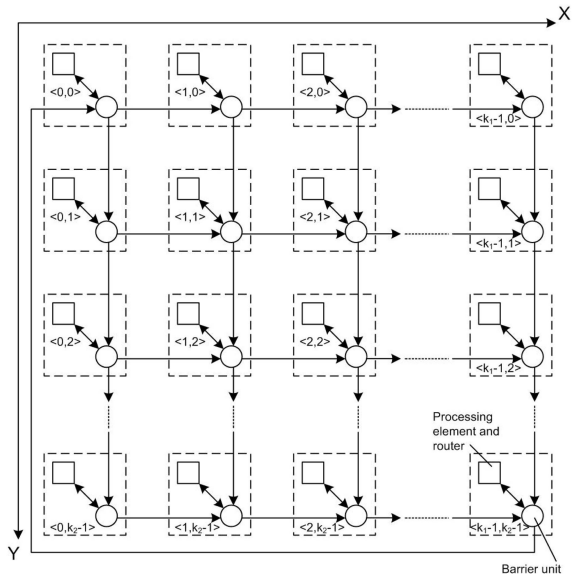


Figure 5: DVBSS network topology [3]

Figure 5 shows the example network topology, notice that the last node of the mesh (k1-1, k2-1) needs to be connected to the first node (0,0), and that the linking in unidirectional it only increases on X and Y axis and never goes back. Each node is connected to its four neighbors, the two previous neighbors give inputs to the concerned node and the two following get inputs from the concerned node.

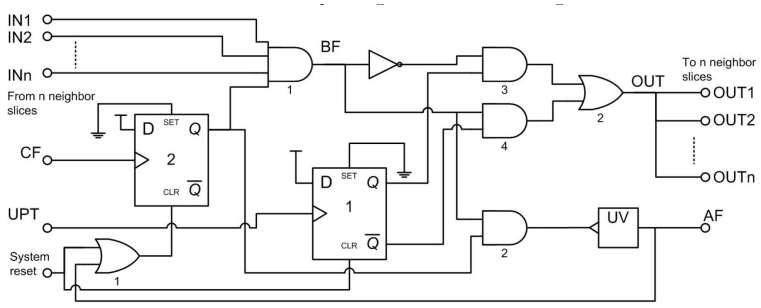


Figure 6: DBSS schematic [3]

As shown in Figure 6, the simple Distributed Bit Slice Synchronizer uses three signals to communicate with the Processing Element (PE) of the node. Completion Flag (CF), indicates that

the PE is entering the barrier, Unit Position Flag (UPF) indicates that this node is the last of the array, those two signals are inputed by the PE, and Activation Flag (AF) indicates to the PE that the barrier is ongoing (high) or passed (low). Two FlipFlops (FF1, FF2) are used to copy the input from the PE. At the beginning of a barrier, UPF is set to high in the last node (k1-1, k2-1), this operation sets to high AND 3 and OR 2, and since this node is only connected to the first node, the first nodes AND gate is unlocked. In all other nodes, UPF is set to low, which means that AND gate 3 is locked. When a PE reaches the barrier, it sets CF to high, consequently the AND gate 1 is unlocked. As soon as the first node reaches the barrier and sets FF1 to high, its AF signal goes high, AND gate 4 goes high and therefore OR gate 2 goes high, transmitting the barrier signal to the two following nodes. If the following nodes have reached the barrier, the signal continues to propagate until it reaches the last node. When the AND gate 1 of the last node is set to high, it means that the reduction phase is completed, AND gates 3, 4 are consequently set to low and therefore the output of the last node goes low.
This event starts the distribution phase since all the following nodes output (starting from (0,0)) will go low. The AF signal will also go low, signifying to the PE that the barrier is passed, and reseting the FF2 through OR gate 1. We can see this barrier like a "wave" propagating high level during the reduction phase, and low level during the distribution phase.
Now, this simple DBSS can be upgraded to a DVBSS, 'V' standing for virtual. In this case, the virtualization means supporting concurrent barrier groups all across the network. This upgrade is provided by the adding of four elements, a N entry latch using an input multiplexer (MUX), an output de-multiplexer (DMX), and a barrier group mapper. Figure 7 shows the DVBSS.

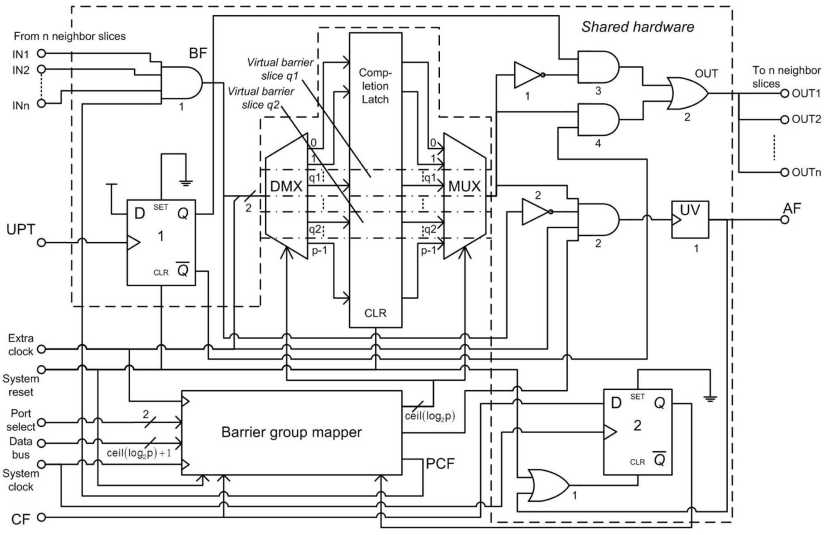


Figure 7: DVBSS schematic [3]

The MUX, DMX and N entry latch are interposed at the output of FF2 and ensure the memorization of the current state of each ongoing barrier. The barrier group mapper is the logic which control the MUX, latch and DMX. It takes the barrier group number to know which context corresponds to which entries of the latch and is synchronized to the main clock. The barrier group mapper also uses another clock to decide when to switch from one barrier group to another. This clock is called Distributed Circulating Wave Clock (DCWC), Figure 8 shows the propagation of the clock on the 2D mesh network.

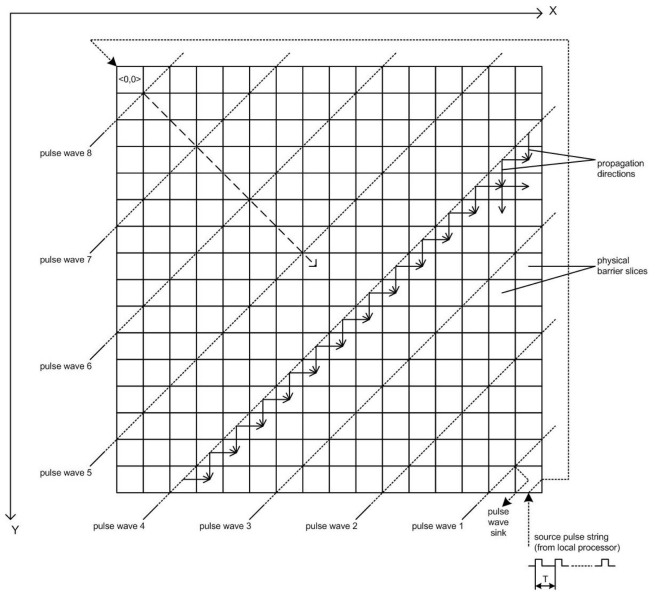


Figure 8: DVBSS clock propagation [3]

When a node receives a pulse from this clock, it switches its latched group context to another and then sends a pulse to the following neighbors. This operation makes the virtualization active by making the contexts switch at a regular rhythm, the size of the network determines the number of overlapping group contexts.
The DVBSS has the advantage of being fast and simple. It supports concurrent barrier synchronizations in groups of different sizes and shapes and thus overcomes the last limitation of the HW barriers. By its dynamic nature, the DVBSS is deadlock free. The DVBSS is scalable to multidimensional meshes network with a great number of nodes, making it optimal for a large scale computer use and since it does not require a complex logic it can be easily implemented in CMP.

Transmission Line Sync is a barrier scheme developed by Jungju Oh et al. in 2011 [5]. This barrier is specially designed for large MCPs and combines the classic *wired AND* with the transmission lines technology [19]. Transmission Lines (TL) are a way to use Solid State Radio Frequency Interconnect (SSRFI) to provide near speed of light communications. A transmission line comprises a medium (special metallic interconnect using three layers of the chip), transmitters and receivers. Figure 9 shows the schematic of a receiver and a transceiver.

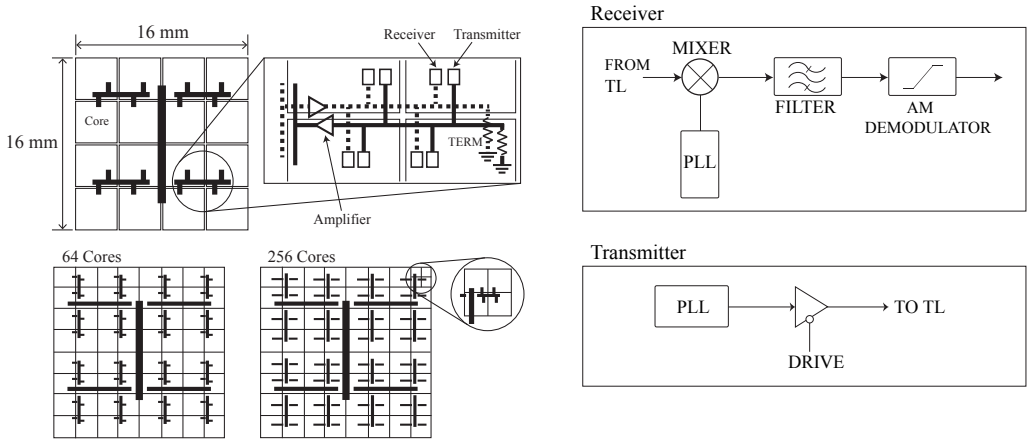


Figure 9: TLSync receiver / transmitter / Lines [5]

In short the transmission line is permanently loaded by a carrier sine wave at a high frequency. To transmit a message the transmitter adds a modulated harmonic on top of the carrier, while the receiver filters the carrier and demodulates the harmonic to obtain the message. This is widely used in all the classic RF communications. The great advantage of the transmission line is that as long as harmonics bandwidths do not overlap, the line can transmit messages concurrently.
TLSync uses this property to realize concurrent *wired AND* barrier. During the reduction phase, each participating thread leaves a "tone" on the carrier, using the binary amplitude modulation. This "tone" tell us that the thread has not entered the barrier yet. When a thread enters the barrier, it stops its "tone", the information is traveling at the speed of light, all threads are notified immediately. When all threads have reached the barrier, the carrier is empty of harmonics and all threads can pass the barrier. Now, by allocating a frequency bandwidth per barrier, TLSync can support many concurrent barriers on the same line at the same time. Each transceiver is tunable, thus TLSync does not require more transceivers to handle concurrent barrier synchronizations. TLSync is one of the fastest HW barrier since its latency time is inferior at 4 ns, it supports many concurrent barrier synchronizations and needs only one set of receiver / transmitter per PE, therefore it is a very promising HW barriers. However, TL still suffers from several drawbacks, they can only be used in ASICs, since it requires specific mixed signal RF electronics. It also requires specific lines, designed to support high frequencies. The RF filters, modulator and demodulator take real estate and therefore cannot be used extensively. TLSync has to be improved to be widely used but it promises great improvements in HW barrier synchronization.

It exists plenty of different approaches that improve the performances of the classic *wired AND* and we chosen to present these three because they were the more characteristic ones. Those barrier schemes show that it is possible to achieve dynamic, and fast HW barrier that overcome the software and hybrid barrier schemes. The next section presents some examples of network topology that are used to improve the performances of HW based barriers.

2.3 Network topology

The point of a hardware based barrier is that it uses a separate control network instead of the classic data network. Since the hardwired network can be different from the barrier "network". Network topology means how we interconnect all the different participating threads of a HW barrier, it could for instance, be a virtual route mapped on top of the hardwired network. This makes sense for dynamic topologies. The barrier network topology is one of the most important features of a HW based barrier, it basically determines the scalability of the barrier. Since electronics are fairly fast, the network has started to become a bottleneck.
Lots of effort have been pushed into this subject over the past decades, and three main topologies show up to be the more efficient ones: *Tree* [7], *single line* [1], and *Parallel lines* [4]. To provide a good support, the network has to be dynamically reconfigurable, deadlock-free, adaptable to different hardwired topologies and fairly fast to ensure an improvement compared to the classic data network.

2.3.1 Trees

Trees have been widely used in many systems [7]. We can distinguish four types of trees: Binary trees, Balanced Binomial trees, Linear trees, and Star trees. In a tree topology, the first node is called a *root*, the middle nodes are called *branches*, and the last node are leaves.

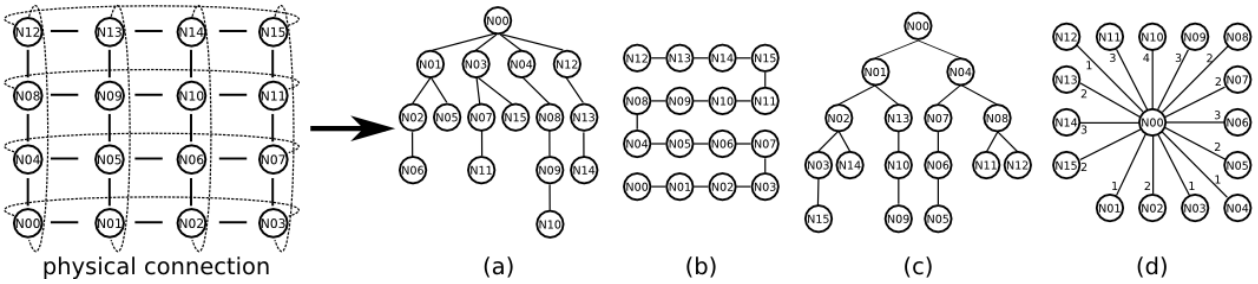


Figure 10: Tree topologies [7]

The Linear tree, is maybe the simplest one, as you can see on Figure 10.b, it consists of nodes interconnected linearly. Each node is only connected to another, creating a chain. We can notice that there is only one leaf in this topology. The map of this topology is usually defined using Hamiltonian paths. The depth of this kind of tree is defined by the number of participating nodes "n". Linear trees have the advantage of being simple and not to require complex control logic. But, they suffer from several problems, by having a "n" proportional depth its latency is very high, and to be optimum their mapping need to be well calculated and thus add overhead to the barrier synchronization time. A last drawback would be that this topology is not deadlock free. We could consider to use it when the network constraints are high and the required performances are low. But, since HW barriers are used to provide high speed features, and are implemented on a separate network we would never see such topology used for a HW barrier.

T**he Binary tree**, shown on Figure 10.c, consist of nodes connected in pair. Each node is connected to two or less other nodes in order to create a wide and distributed network. The depth of such tree is usually proportional to the log2 of "n", where n is the number of participating nodes. We can notice that this type of tree has one root and two to the power of "n" leaves. The Binary tree is fairly fast compared to previously cited linear trees, its depth attenuates the latency, and its parallelism greatly accelerates the barrier process. However, its overhead is still consequent and by only accepting two child nodes per parent node, it may lead to situations where a node is isolated or distant and therefore increase the latency. Binary trees have been used in several implementations, they are a good option in many cases.

The Balanced Binomial tree, is structurally very close to the binary tree, however its nodes repartition is different. We can see on Figure 10.a that each node may support more than two other nodes. The tree's order "k" defining its depth, related to the binomial coefficients. Advantages of the Balanced Binomial tree are almost the same to that of the binary tree, except that it supports multiple child nodes per parent node. This improvement leads to latency reduction compared to the binary tree. This topology provides the lowest latency of all trees and is widely used in many systems.

The Start tree [12], is a pretty straight forward approach, like Figure 10.d shows all the child nodes are connected to one parent node. This tree has always a depth of 1, 1 root and "n-1" leaves. This tree is in theory the fastest since its depth is only one. Nonetheless, Its performances are strongly dependent to the capacity of the root to handle the numerous parallel lines. This topology also needs one mapped path per participating node. This is the main reason why we do not see this topology in many systems.

Tree topologies can be implemented in multidimensional meshes and torus, can support virtualization, and can be dynamic [18] or static. In nowadays systems it is almost always dynamic, but in the first implementations of HW barriers, binary trees were hardwired and mostly static [14]. Hardwired non dynamic topologies are non scalable to large architecture and have thus disappeared from those. The CM-5 is an example of hardwired topology which is partially dynamic, by "cutting" the branches to create sub-trees. Dynamic topologies are needed since barriers need to support thread groups of different size and repartition. Dynamicity also allows, in certain cases, to be fault tolerant [15], such feature is truly appreciable.

2.3.2 Single or global line

The single line topology [1], is based on only one communication medium interfacing several nodes. This approach is interesting since we only need one line all across the system to process to the barrier. We do not have a cascade of overhead caused by the network switches and therefore the latency of such topology is fairly low. This topology is obviously deadlock-free and dynamic, it could also support virtualization and we describe later how this topology can exploit "physical parallelism". It has unfortunately one major drawback, it needs its very own hardware and thus involves large implementation costs. BlueGene/Q implement such tree [20]

2.3.3 Parallel lines

In the parallel lines topology, many lines are used in parallel to connect all threads, instead of only one. This topology directly exploits one kind of "physical parallelism" and it has to be differentiated from just a hardwired tree that would have parallel copies of itself. In The Parallel lines topology each line correspond to a thread, each thread can write only on its own line, but it can read all the other lines. Therefore, the barrier process can occur in parallel. This has the advantage of limiting the overhead, and consequently increases the barrier speed. Like the single line, it is deadlock free and dynamic. However, it needs one line per thread and thus it is not easily scalable. "people" describe an implementation using optical interconnect [4].

Those two last topologies are usually used in CMPs since the number of threads to synchronize is low, while the tree topologies are mainly used in supercomputers due to they ability to scale. However, since the technology evolve and make global line scalable, we will see it more and more in large scale system [21]. The barrier network topology is an important part of hardware based barrier synchronization, it requires careful attention to ensure full performance.

3 Examples of actual implementation

There are three ways of implementing barrier synchronization, there is the *Network-based* approach, the *Memory-based* approach and the *Global Lines* approach.
The Network-based approach is implemented in Message-passing multiprocessor systems in which data sharing is done via message-exchanging, an example would be **Cray T3E** [14].
The Memory-based approach is implemented in Shared-memory multiprocessor systems in which data sharing is done via load and store instructions, an example would be **Thinking machine CM-5** [16]. Global lines approach is implemented in IBM's **BlueGene/Q** [20][21].

3.1 Cray T3E

Cray T3E was developed by Cray Research and was released for sale in 1995, it was the second generation of the massively parallel supercomputer (predecessor was Cray T3D). In a Cray T3E (up

to) 2048 Processing Elements (PE) can be found, each of those will occupy one node in a bidirectional 3D torus network. Each PE consists of DEC Alpha 21164 processor, a control chip, a router chip and a local memory, a block diagram of the Processing Element can be seen in Figure 11.

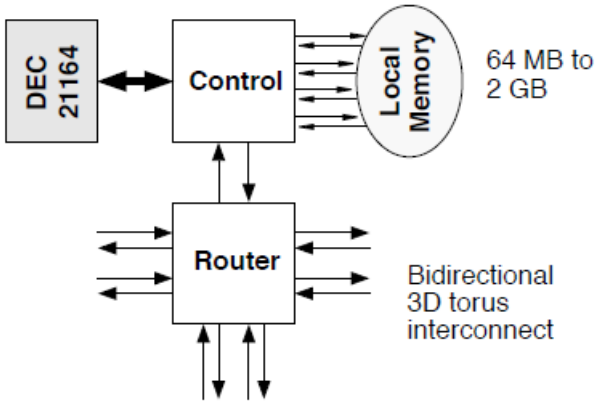
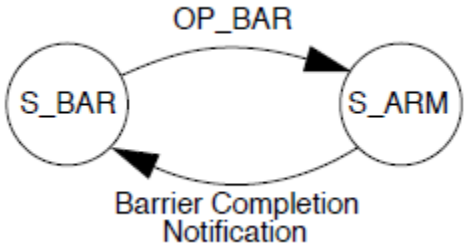


Figure 11: T3E PE block diagram [14]

The memory of the DEC 21164 processor is augmented with a set of explicitly managed external registers (E-registers). This is interesting because all remote communication and synchronization is done between the E-registers and the memory. This is not all, to be able to execute barrier and eureka synchronization Cray T3E provides (per processor) a set of 32 Barrier/eureka Synchronization Units (BSUs) that are accessible as memory mapped registers.
This is how it works: A set of processors will be given access to a BSU, now depending on the state the BSU is in the processor will perform fitting operation with load and store operations. When a processor enters a barrier an *Arm Barrier* operation will change the BSU state from *Barrier just completed* to *Barrier is armed* and once all the participating processors have reached the barrier the network will deliver notification about the barrier completion and BSU will change state back to *Barrier just completed*, see Figure 12. The barrier events are signaled with packets which are sent over a virtual channel. Barrier/Eureka synchronization tree is embedded in the 3D torus network.



State	Operation	Description
S_BAR		Barrier just completed
	OP_BAR	Arm Barrier
S_ARM		Barrier is armed

Figure 12: Barrier state diagram 3.2 Thinking machine CM-5

3.2 Thinking machine CM-5

Connection Machine-5 was developed by Thinking Machines Corporation and was released in 1993. It is a synchronized MIMD machine that can contain from 32 to 16384 processing nodes, each processing node consists of: 32MHz SPARC Processor, 32MB Memory and 64-bit floating-point and integer Vector Processing Unit. CM-5 has three networks: data, control and diagnostic.

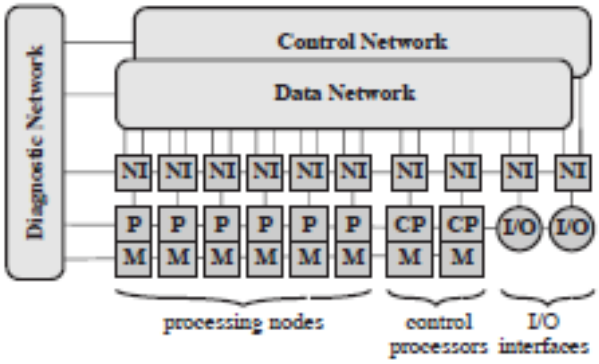


Figure 13: CM-5 organization

The control network is a tree-like network that was implemented for easier synchronization of processors and includes functions such as broadcasting of data, barrier synchronization and parallel prefix. Because of our interest in the barrier synchronization the emphasis will be on the combining operation of the control network, which supports four types of operations, one being the router done operation, which is used to notify the processors when the communication involving the data network is completed. The router done operation is based on the Kirchoff's current law, meaning that the number of messages in must equal the number of messages out, this is done by keeping track of the number of messages that enter versus leaves the network. This is how it works: Whenever a processors is done sending messages it will push a message into its outgoing router-done FIFO, which simply means that it has entered the barrier. When all other participating processors have entered the barrier, i.e. pushed a message into their outgoing router-done FIFO, the control network that all this time has been keeping track of the number of messages going in and number of messages going out of the network and checking the difference, will push a message into all of the participating processors ingoing router-done FIFO if the result is zero, indicating that the barrier has been completed.

3.3 IBM BlueGene/Q

BlueGene/Q (BG/Q) is the third generation of the massively parallel supercomputer developed by IBM. Each BG/Q rack has 1024 compute nodes, where one compute node consist of sixteen PowerPC A2 processors together with 16GB SDRAM-DDR3 memory. The main network that is used is the 5D torus in which the Global Barrier (GBarrier) is implemented. The functionality of the GBarrier will be described using the 2D-mesh network and 16-core CMP for the sake of clarity.

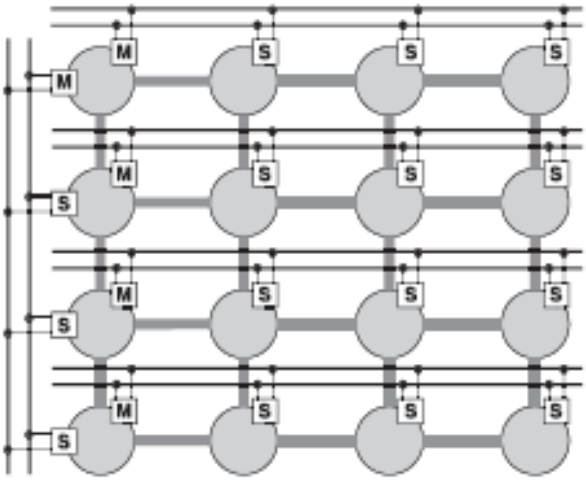


Figure14: GBarrier architecture for a 16 CMP on a 2D mesh network

This is how it works: GBarrier mechanism uses a G-line based network which consists of G-lines that are used to transmit signals (1-bit messages), and controllers (master and slave) that implement the synchronization protocol. In the Figure 14 the G-lines are the 10 black lines (in reality wires) and the controllers are the boxes tagged with either M (Master) or S (Slave). One line is used for transmitting and other for receiving signals. The synchronization protocol is all about exchange of signals between the master and slave controllers and implementation of S-CSMA technique that in short counts number of signals sent across every G-line.

When a processors enters a barrier the account phase, which is one of two phases of the synchronization protocol, will be started. It is finished when last participating processor has entered the barrier. When this has happened the second phase of the synchronization protocol, the release phase will command the processors to resume execution. This process is implemented in the master and slave controllers and the state diagram can be seen in Figure 15.

The expression in the left bracket signifies an event and the expression in the right bracket signifies an action, [EVENT]/[ACTION]. Bar reg is a 1-bit register used for the barrier arrival notification, it is set to one when the processor has entered the barrier.

In the XglineY X identifies the type of controller, M is for Master and S is for Slave, Y is either H for Horizontal or V for Vertical lines. Each Master has registers ScntH and ScntV that can be updated with value MAXH and MAXV, i.e. the sum of the participating horizontal and vertical slave controllers.

Vflag is used for the local synchronization of the horizontal master and the corresponding (master or slave) controller with whom they share the core.

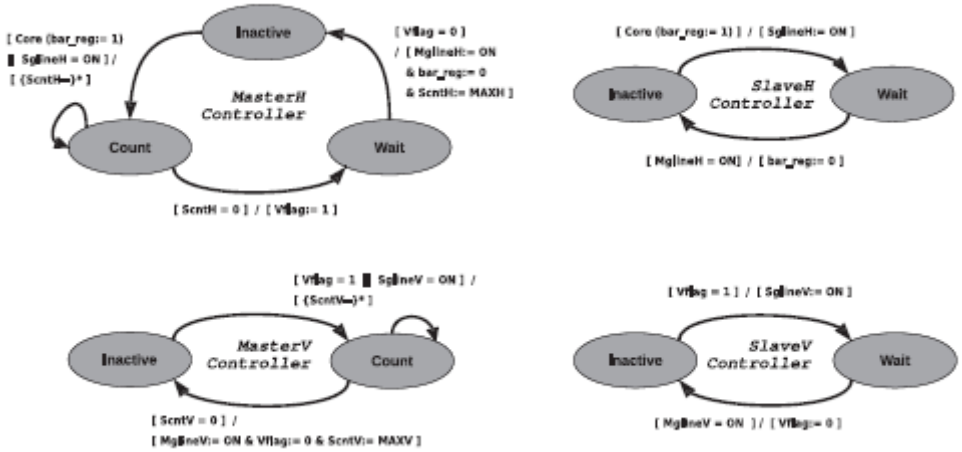


Figure 15: Gline controlers Implementation

4 Our opinion regarding the future HW barriers

As previously shown, there are many different network topologies and hardware supports that is possible to mix. This allows designers to choose which barrier to use depending on their needs, and thus open a wide range of possible applications. The number of core in Chip Multi-Processors tend to grow continuously since the past decade and it will soon lead to massive many-cores chips. The need of a fast on-chip synchronization system is real and several research, such as TLSync [5] or the Glines [1] are on the good way to fix it. On the supercomputer side, the race to the Exa-scale has started and those computers will integrate millions of parallel processing units. The size of such computers is getting bigger, the barrier latency will become increasingly critical. It is really important that efficient hardware barriers keep this latency low to avoid to face a barrier synchronization bottleneck. Regarding the previous conclusions, future HW barrier will probably be multileveled. Using ultra high speed HW barrier on chip, interfaced with a modular and hierarchical control network using fiber optics mediums. Those interfaces would be realized on specific network controller chips, which would constitute the second level of the barrier. Each interface would regroup a set of CMP and probably act as a crossbar switch, allowing to directly interconnect all CMPs of a node. Virtual channels would not be used for very high performance since the number of concurrent barrier groups will grow accordingly to the number of cores, leading to a ridiculously high number of channels, which would exponentially increase the latency of each particular barrier group. Using laser technologies with parallel light modulation in a fiber optic global line network will solve this concurrent barrier group problem. Figure 16 shows a "picture" of what we think the future would be. Each PE would uses transmission lines for the lower level of the hardware barrier, each group of PE would be linked through a second level of hardware barrier, a group of PE continuing a node of the global barrier network, it would be connected through laser high speed interconnect to the other nodes, ensuring the global barrier synchronization. This figure is just an example and many other multilevel barrier schemes are possible.

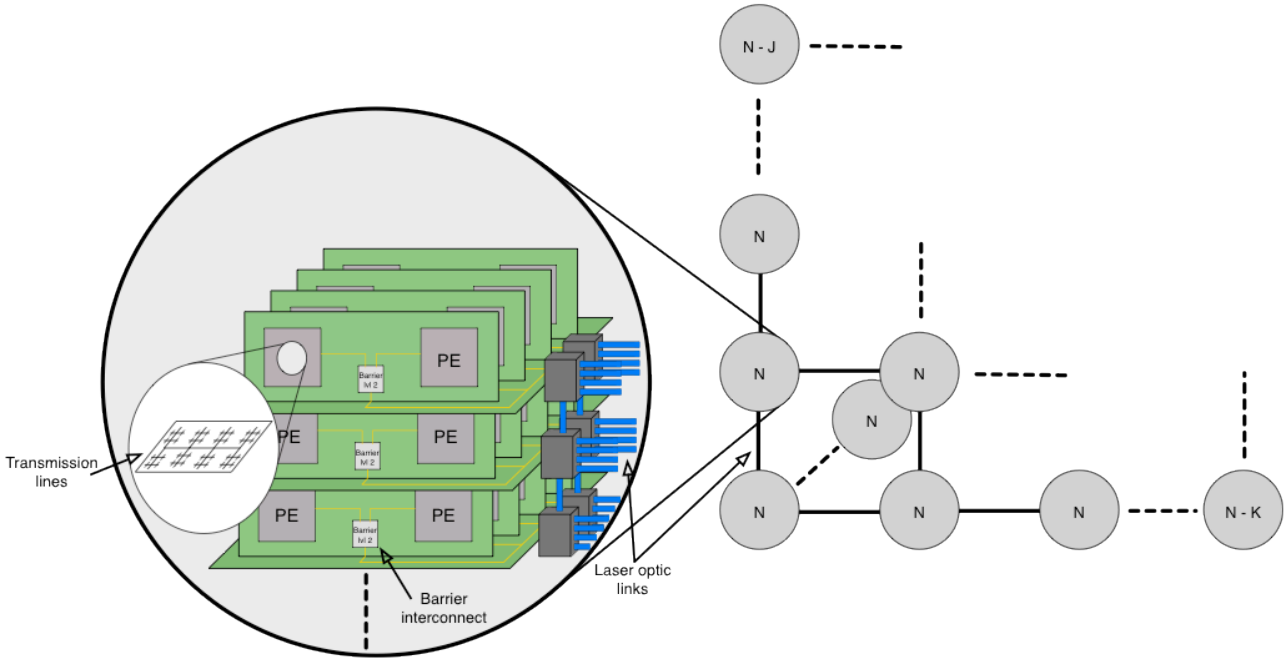


Figure 16: Barrier future

5 Conclusion

In the coming supercomputer generation, and in order to achieve exa-scale computing, designers will have to use an increasing number of cores. The threads synchronization is becoming a bottleneck and classic synchronization schemes are insufficient to solve the problem in the long term. Hardware barrier synchronization bring a valuable solution to this problem. It is the most promising synchronization method, with high performances, low power consumption and low latency, it overcomes all other synchronization schemes.

References

1. José L. Adellán et al., "Efficient Hardware Barrier Synchronization in Many-Core CMPs", *IEEE Transaction on parallel and distributed systems*, Vol 23, No 8, August 2012.

2. T. A. Johnson and R. R. Hoare, "Cyclical Cascade Chains: A dynamic barrier synchronization mechanism for multiprocessor systems", *IEEE Parallel and Distributed Processing Symposium*, Proceedings 15th International, 2001.

3. Igor Valerievich Zotov, "Distributed Virtual Bit Slice Synchronizer: A Scalable Hardware Barrier Mechanism for n-Dimensional Meshes", IEEE Transaction on Computer, vol 59, issue 9, 2010.

4. William E. Cohen et. al, "An Optical Bus-Based Distributed Dynamic Barrier Mechanism", I*EEE Transaction on Computer*, vol 49, issue 12, 2000.

5. Jungju Oh et. al, "TLSync: Support for Multiple Fast Barriers Using On-Chip Transmission Lines", *Computer Architecture (ISCA), 2011 38th Annual International Symposium*.

6. Goerge Almási et. al, "Optimization of MPI Collective Communication on BlueGene/L Systems", 2005

7. Shanyan Gao et. al, "Hardware Implementation of MPI_Barrier on an FPGA Cluster", *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference*, 2009.

8. Xiaowen Chen and Shuming Chen, "DSBS: Distributed and Scalable Barrier Synchronization in Many-core Network-on-Chips", *international joint Conference of IEEE,* 2011.

9. Oreste Villa et. al, "Efficiency and Scalability of Barrier Synchronization on Noc Based Many-core Architectures", *CASES '08 Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems.*

10. Martha Ximena Torres Delgado and Sergio Takeo Kofuji, "A Distributed Barrier Synchronization Solution in Hardware 2D-Mesh Multicomputer", *High Performance Computing, 1996. Proceedings. 3rd International Conference.*

11. Donald Johnson et. al, "Low-Cost, High-Performance Barrier Synchronization on Networks of Workstations", *Journal of parallel and distributed computing,* vol 40, 1997.

12. Elizabeth Whitaker Lynch, George F. Riley, "Hardware Supported Time Synchronization in Multi-Core Architectures", *23rd Workshop on Principles of Advanced and Distributed Simulation,* ACM/IEEE/SCS, 2009.

13. Vara Ramakrishnan et. al, "Efficient Techniques for Nested and Disjoint Barrier Synchronization", *University of California, Dept. of Info. & Computer Science, 1996.*

14. Steven L. Scott, "Synchronization and Communication in the T3E Multiprocessor", *Cray Research,* 1996.

15. Rajeev Sivaram et. al, "A Reliable Hardware Barrier Synchronization Scheme", *Parallel Processing Symposium, 1997. Proceedings, 11th International.*

16. Charles E. Leiserson et. al, "The Network Architecture of the Connection Machine CM-5", *Thinking Machines Corporation,* 1996.

17. W. E. Cohen et. al, "Dynamic Barrier Architecture for Multi-Mode Fine-Grain Parallelism using Conventional Processors", Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference.

18. Sangman Moh et. al, "Four-Ary Tree Based Barrier Synchronization for 2D Meshes without Nonmember Involvement", *IEEE Transaction on Computer*, vol 50, No 8, 2001.

19. T. Krishna, A. Kumar, P. Chiang, M. Erez, and L.-S. Peh. "NoC with near-ideal express virtual channels using global-line communication", *Symp. on High-Performance Interconnects*, pages 11–20, 2008.

20. IBM Corporation, "IBM System Blue Gene/Q datasheet", november 2011.

21. Dong Chen et al, "The IBM Blue Gene/Q interconnection Network and Message Unit", *High Performance Computing, Networking, Storage and Analysis* (SC), 2011 International Conference.