Research Journal of Applied Sciences, Engineering and Technology 11(8): 902-909, 2015
DOI: 10.19026/rjaset.11.2102
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: June 24, 2015 Accepted: August 15, 2015 Published: November 15, 2015

**Research Article**

# Bidirectional Virtual Bit-slice Synchronizer: A Scalable Solution for Hardware-level Barrier Synchronization

Jamil Al Azzeh
Department of Computer Engineering, Al Balqa Applied University, Amman, 11134, Jordan

**Abstract:** In the study, a new distributed hardware-level method for barrier synchronization of parallel processes in a mesh-connected parallel system is presented, which is based on the use of a virtually sliced barrier control network timed by two bidirectional clock pulse waves originating from the corner processors of the mesh.

**Keywords:** Barrier synchronization, hardware-level barrier, mesh-connected parallel system, parallel processes

## INTRODUCTION

Barrier synchronization is known to be a specific form of massive interprocessor communication in multicomputer and multiprocessor systems which guarantees a given precedence relation between code sections. It involves no data transfer, but highly affects the computer system performance.

A barrier, a cornerstone entity of barrier synchronization, is typically defined as a logical delimiter in the control flow of a parallel program, at which all or some processes (threads) must wait for their peers to proceed simultaneously (Axelrod, 1986). When a barrier is executed, two phases take place. During the first phase (known as "reduction"), each participating process reports of its arrival and starts waiting for the barrier to complete. The second phase (known as "distribution") begins as soon as all the participants have reached the barrier and it goes on until all the peers are notified that they can resume. Barrier is a fundamental collective communication procedure in parallel programs developed using the MPI (Forum, 2012) and OpenMP (OpenMP Architecture Review Board, 2011) parallel programming standards. According to these standards, specific syntax to support barrier synchronization on a process group or a thread team is used (MPI_Barrier() and #pragma omp barrier in C/C++, respectively).

In the four past decades, there have been developed a wide range of methods for barrier synchronization. Depending on the implementation level existing methods can be divided into three classes: software (Tsafrir and Feitelson, 2002; Li *et al*., 2004; Tzeng *et al*., 2005), hybrid (Moh *et al*., 2001; Hindam, 2004; Sampson *et al*., 2006) and hardware (Thinking Machines, 1992; Delgado and Kofuji, 1996; Ramakrishnan *et al*., 1999; Cohen *et al*., 2000; Johnson and Hoare, 2001; Zotov, 2010; Ashraf *et al*., 2012; Al-Azzeh, 2013). Hardware barriers have shown to be a better solution in general because they are order-of-magnitude faster than software methods and produce no extra message traffic compared to hybrid solutions. At the same time, the lack of flexibility is the main problem of hardware-level methods. In most cases, hardware barriers impose stringent limitations on the barrier group configuration and/or the number of barriers in an application and, therefore, the effective use of these methods in practice is not possible.

In Zotov (2010), a new hardware-level distributed barrier mechanism for mesh-connected parallel systems has been presented to solve some flexibility issues. This approach, called the Distributed Virtual Bit-Slice Synchronizer (DVBSS), puts away the restrictions on the configuration of barrier groups and, in theory, it eliminates the limitations on the number of barriers in the system. However, DVBSS requires multi-bit "long" wraparound connections between corner processor units of the mesh which are rather complicated to be implemented and bring extra delay in the synchronization latency. Yet, the addition of new processor units to the mesh becomes an issue because of the necessity to physically reconnect the wraparound multi-bit channels.

In the study, we extend the DVBSS approach presented in Zotov (2010). The objective of the study is to present a modified distributed hardware barrier architecture making it possible to transfer barrier state signals through the mesh in two opposite directions between two corner units thus eliminating the wrap around connections of the DVBSS network and providing better flexibility and scalability. We demonstrate that our extended scheme accepts dynamically defined (possibly overlapping) barrier groups of arbitrary size and shape, allowing noncontiguous group member allocations. Our

This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

simulation study shows that barrier synchronization duration in most cases is kept as low as O (10) μsec depending on the peak number of instantiated barrier groups which corresponds to DVBSS and existing hybrid methods.

## MATERIALS AND METHODS

In the study, we consider *d*-dimensional mesh-connected distributed memory parallel systems. A target system is supposed to consist of $k_1 \times k_2 \times \ldots \times k_d$ processing units, where $k_i$ denotes the width of an *i*th dimension of the mesh. Each unit $\langle x_1, x_2, \ldots, x_d \rangle$ is connected to the corresponding neighbors by at most 2D bidirectional links. Each processing unit is supposed to be capable of executing a single process of the application, but there can be several applications running in parallel in separate partitions of the system. It is assumed that each processor is connected to a router to send and receive messages. Two types of messages are considered-barrier and non-barrier. Non-barrier messages are those that perform all communication operations, except barrier synchronization. Barrier messages are those that implement barrier synchronization and are transferred to/from dedicated control network (synchronizer) superposed on the main communication network of the system.

We suppose that the parallel system under consideration is programmed using the extended MPI standard, version 3.x (Forum, 2012), but we do not restrict our barrier mechanism to MPI-based environment only. We assume that barrier groups are instantiated and released at runtime and there is at least one global barrier group in the application at any given moment. It is supposed that processes can be dynamically created and destroyed and that any process may be a participant of several barrier groups at the same time.

It is assumed that a collection of parallel applications are running in parallel in the system, each occupying a separate partition and having its own set of barrier groups. No limitations on the size and shape of partitions are set, yet partitions may be noncontiguous. It is supposed that partitioning is dynamically defined, meaning that a partition may change as new processes are created or terminated. Each barrier group can be synchronized at any number of barriers and any barrier can be inside a loop which in turn can be nested. Thus, multiple simultaneous barriers are allowed.

We suppose that a separate control network consisting of identical barrier units is superposed on the

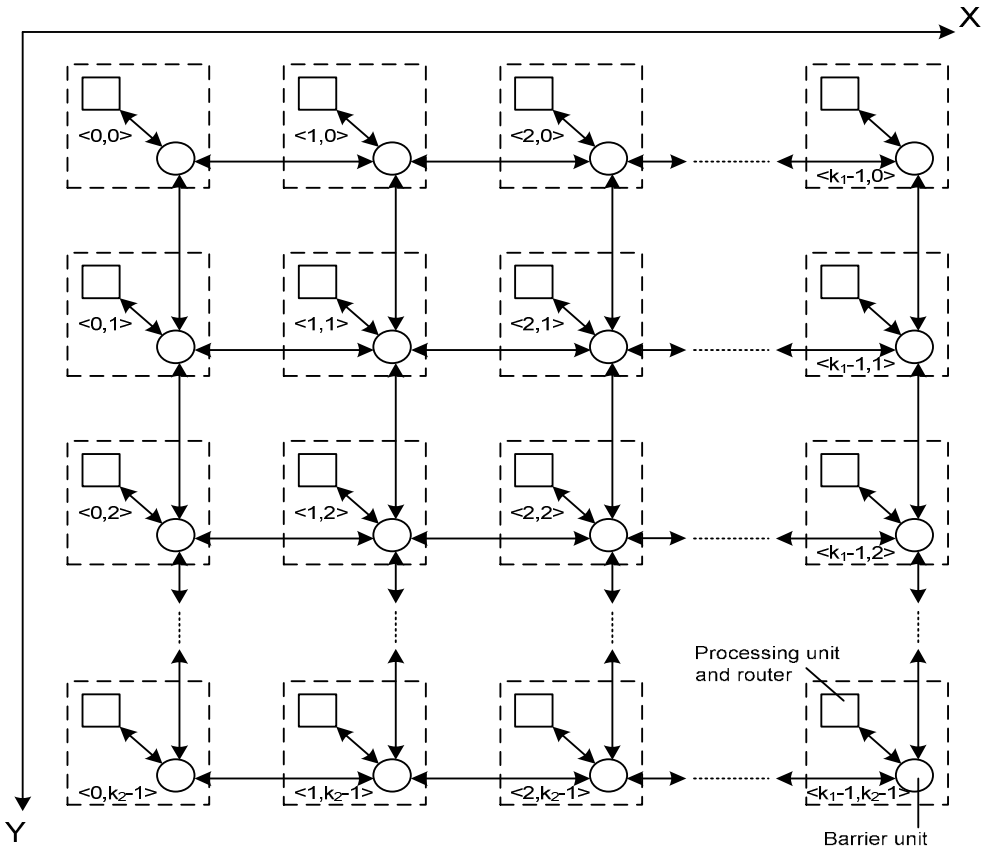


Fig. 1: Directed graph representing the topology of the control network (2D case)

system. Each barrier unit (which we denote $\langle x_1, x_2, \ldots, x_d \rangle$) is connected to the corresponding router with a separate channel and can receive/transmit messages from/to the local processor. Barrier units are connected to each other with multi-bit channels in such a way that any unit has the same set of neighbors as that of the corresponding local processor. Bidirectional connections are used, therefore, any neighbor of a given barrier unit can be both a receiver and a transmitter.

Figure 1 shows a directed graph representing the topology of the control network constructed for a 2D mesh parallel system. Circles denote barrier units, while solid rectangles denote processing elements together with their routers.

Each barrier unit is comprised of slices. A slice is a single-bit "section" of the corresponding barrier unit. The slices of a barrier unit are numbered consecutively 0, 1, 2, …, $m$ and operate in parallel. Each slice of a barrier unit is connected to the corresponding slices of the neighbor barrier units according to the network topology shown in Fig. 1. The set of slices having the same number $i$ in all barrier units together with their connections make a single-bit physical control network (which can also be understood as $i^{th}$ control network slice).

To make a single-bit physical control network capable of performing concurrent synchronizations, we apply a virtualization scheme based on time-division channeling. We assume that there exists a set of $p$ single-bit virtual control networks in a physical one. Also it is supposed that there is a distributed switching mechanism capable of activating the virtual networks one after another in a pipeline fashion. Barrier groups are mapped onto virtual control networks in such a way that parallel groups are assigned to different networks. As a result of such virtualization, each physical slice can be a participant of multiple barrier episodes associated with appropriate virtual networks allocated to different barrier groups. A physical slice includes $p$ virtual slices numbered 0, 1, 2, …, $p$, where $p$ is understood as the upper bound on the number of barrier groups assignable onto a physical control network (i.e., the virtualization "depth" of the synchronization mechanism).

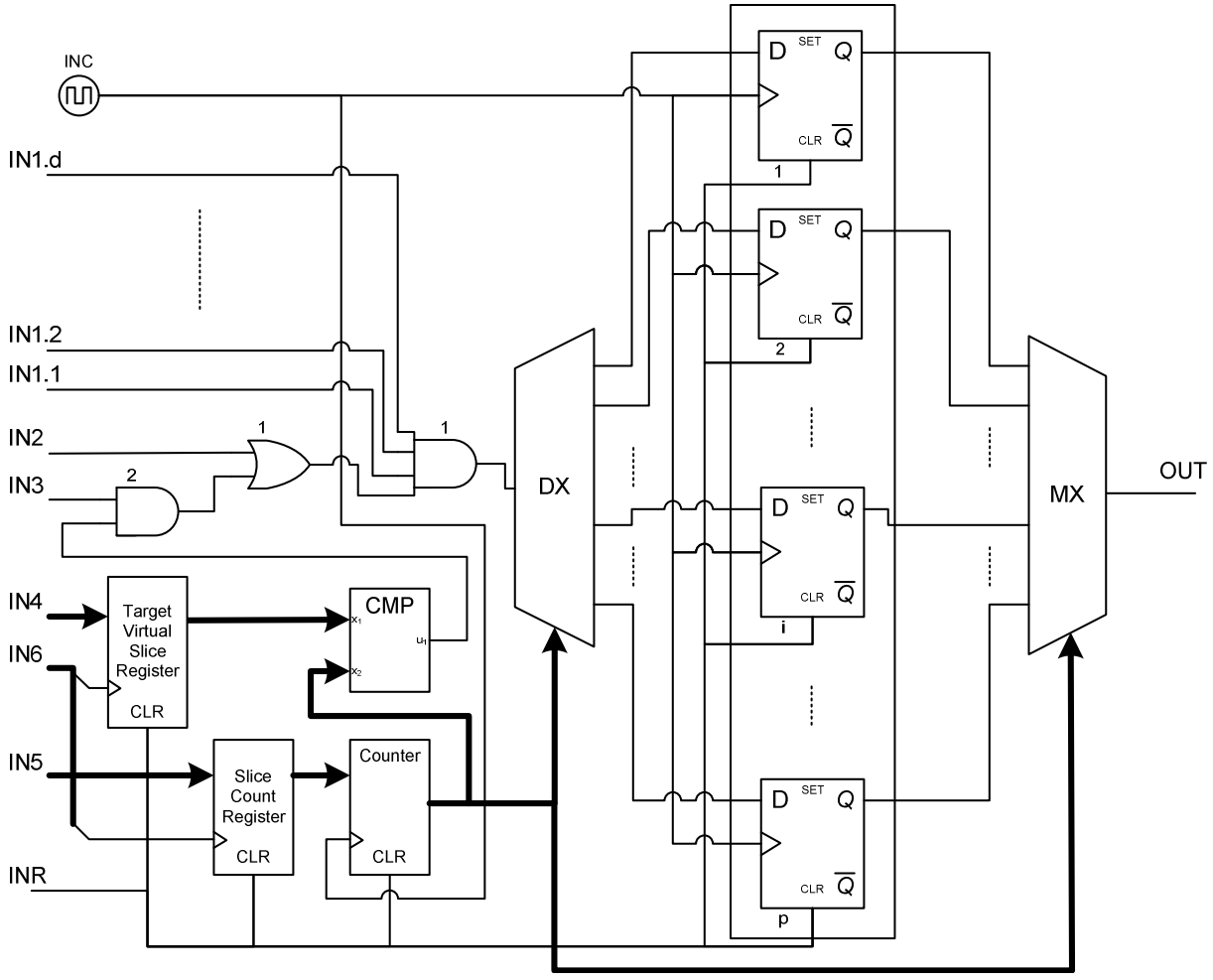


Fig. 2: Logical configuration of a physical slice for the reduction phase (d-dimensional case)

Figure 2 shows the logical configuration of a physical slice necessary to implement the reduction phase of barrier synchronization according to our method (reduction slice).

Hardwired inputs IN1.1, IN1.2, …, IN1.*d* are used to connect the slice to the neighbor transmitter slices, i.e. those slices whose coordinates are equal to or less than those of the current unit by one at the only position (note that if there is no neighbor along a certain dimension *j*, e.g., node <1, 0> in Fig. 1 has no neighbor above, input IN1.*j* should be constant high). The only fan-out connection OUT is employed for wiring the slice to the neighbor receiver slices, i.e., those slices whose coordinates are equal to or greater than those of the current unit by one at the only position (again, there may be no neighbor along a certain dimension *j*, e.g., node <0, $k_2$-1> in Fig. 1 has no neighbor below). Inputs IN1.*i* and output OUT are necessary to receive barrier state signals indicating the completion of the reduction phase for different barrier groups.

The functions of the remaining terminals are as follows. Single-bit input INC is used to receive clock pulses from the distributed clocking mechanism providing coordinated operation of the physical slices across the *d*-dimensional mesh (detailed below). The system reset input terminal INR allows to initially clear the flip-flops and registers. Single-bit input IN2 is required to check out a mask bit indicating if the current processor unit is a participant of the current barrier group (logical "0" stands for "yes" and logical "1" - for "no"). Single-bit input IN3 is necessary to check out a state bit indicating if the current processor has reached the current barrier (taken into account if IN2 equals "0"). [$\log_2$ p] -bit-wide input channel IN4 is introduced to receive the target virtual slice numbers from the local processor to indicate which slice is allocated to the next barrier to arrive at. [$\log_2$ p] -bit-wide input channel IN5 is necessary to receive the virtual slice count, i.e., how many virtual slices are currently allocated to barrier groups in the entire system. The system clock input bus IN6 is required to supply clock pulses from the local processor.

In addition to the input and output terminals, the slice in Fig. 2 contains a collection of *p* flip-flops, a *p*-to-1 multiplexer MX, a 1-to-*p* demultiplexer DX, a target virtual slice register, a slice count register, a counter, a

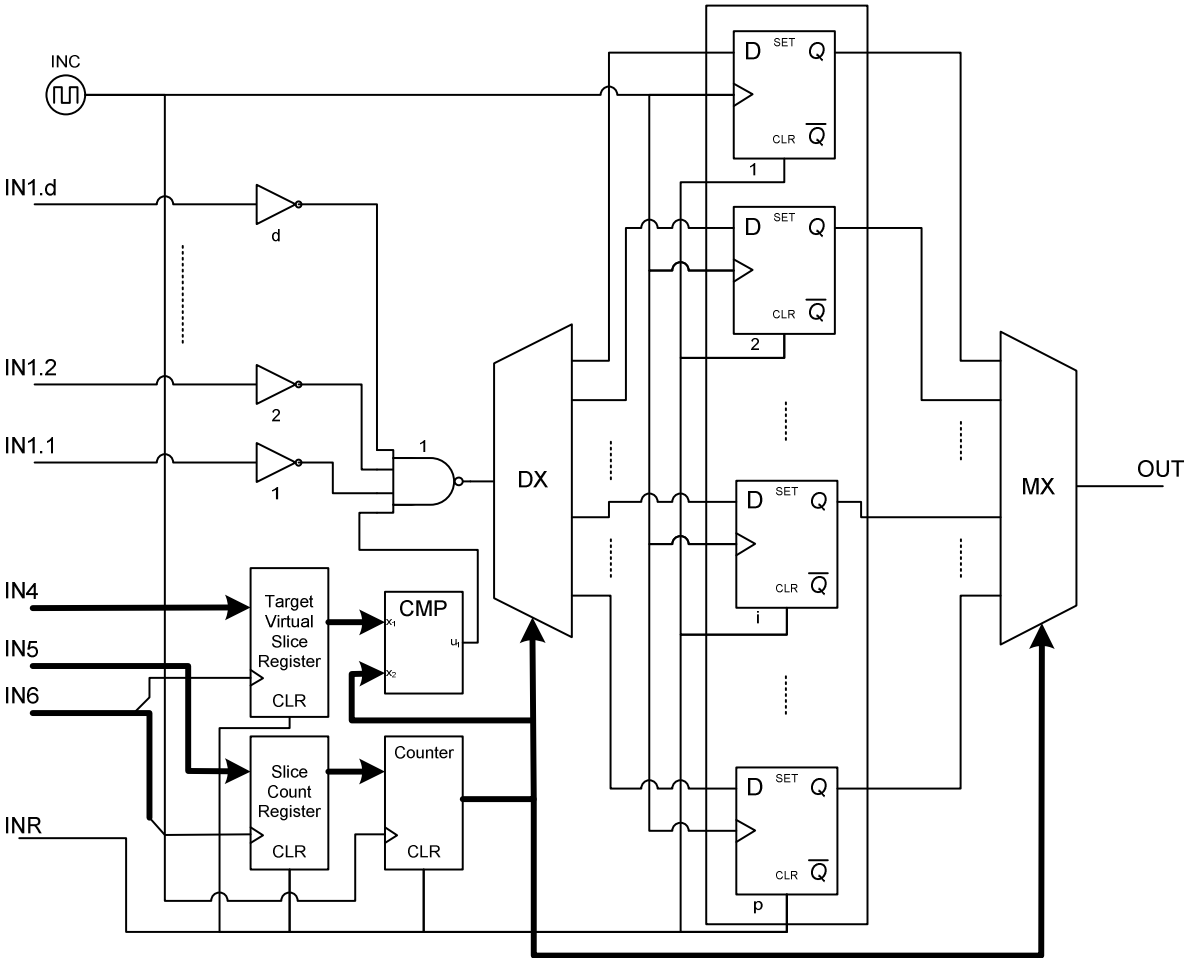


Fig. 3: Logical configuration of a physical slice for the distribution phase (d-dimensional case)

comparator CMP and gates 1 and 2 and an OR gate 1. The flip-flops together with the counter, demultiplexer DX and multiplexer MX are used to implement the proposed virtualization scheme. Flip-flop *i* (1≤i≤p) contains the state of the reduction phase for the current barrier *b* ($f_i$) of barrier group $f_i$ currently mapped onto virtual slice *i*. If all the participants of group $f_i$ whose coordinates are less or equal to those of the current one have reached barrier *b* ($f_i$), then flip-flop *i* is set to logical "1". Otherwise it is reset to "0". Which state should be the next is determined by the output of AND gate 1.

AND gate 1 together with OR gate 1 AND gate 2 and comparator CMP are used to produce the state signal for the current barrier b ($f_i$). The output of AND gate 1 may go high only if all the transmitter neighbors of the current unit supply the signals of logical "1" to inputs IN1.1, IN1.2, …, IN1.*d*. If the above condition does hold, then the output of AND gate 1 starts depending on the output of OR gate 1. OR gate 1 issues "1", if the current processor is not a participant of barrier group $f_i$ (the mask bit IN2 = "1") or if it is a participant and AND gate 1 issues the signal of logical "1". Otherwise the OR gate's output is held low meaning that the current processor hasn't yet arrived at the current barrier. AND gate 1 produces high output level, if the current processor has reached the current barrier (the state bit IN3 = "1") and the comparator's output is high which implies that the reduction phase for the target virtual slice is in progress (i.e., the counter's content is the same as the target virtual slice read from the corresponding register). If CMP issues low logical level, then the state bit is not taken into account because the current virtual slice is different from the target virtual slice. Note that the counter is zeroed as soon as it reaches the maximum virtual slice number contained by the slice count register.

Figure 3 shows the logical configuration of a physical slice necessary to implement the distribution phase of barrier synchronization according to the proposed method (distribution slice).

The diagram in Fig. 3 is similar to that in Fig. 2, except that it contains *d* input invertors, a NAND gate 1 and it has no inputs IN2 and IN3. The function of its elements and terminals is the same as that of the above diagram. The only difference is that negated barrier state signals are transferred and processed by this unit. The negate operator is performed in hardware by units $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ and $\langle 0,0,\ldots,0\rangle$ as follows. Unit $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ directly transfers output OUT of its reduction slices (Fig. 2) to all inputs IN1.1, IN1.2, …, IN1.*d* of its distribution slices (Fig. 3), respectively. Unit $\langle 0,0,\ldots,0\rangle$, in turn, directly transfers output OUT of its distribution slices (Fig. 3) to all inputs IN1.1, IN1.2, …, IN1.*d* of its reduction slices (Fig. 2), respectively. This means that the reduction slices of unit $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ are connected to its distribution slices and the distribution slices of unit $\langle 0,0,\ldots,0\rangle$ are connected to its reduction slices, respectively (i.e., the consecutive numbering of slices is strictly adhered to). Note that the reduction slice of Fig. 2 and the distribution slice of Fig. 3 may use the same slice count and target virtual slice registers, while the counters and the comparators must be separate because of the distributed clocking scheme adopted.

To guarantee correct operation of the proposed barrier synchronization scheme, a suitable clocking mechanism is required. For this purpose we adopt the Distributed Circulating Wave clocking (DCW-clocking) technique described in paper (Zotov, 2010). However, the usage of the DCW-clocking "as is" isn't possible because of the difference in the control network logic, topology and schemata. For this reason we build up an extended distributed clocking mechanism applicable to meshes of any dimension on the basis of the DCW-clocking scheme. We call the new clocking mechanism the Bidirectional Distributed Wave clocking (BDW-clocking).

The basic idea of the BDW-clocking technique is that two trains of clock pulses are simultaneously injected into the control network at the opposite "corners" of the mesh $\langle 0,0,\ldots,0\rangle$ and $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ .The distribution of clock pulses in the barrier slices is synchronized by adding the pulses that arrive from the transmitter neighbors and issuing replica pulses to the receiving neighbors of the current unit. This guarantees synchronized virtual slice switching across the entire mesh. The distribution of clock pulses through the control network can be imagined as the propagation of two series of waves of timed pulses. The first series (forward pulse wave) is transferred from unit $\langle 0,0,\ldots,0\rangle$ to unit $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ to control the reduction phase and the second series (backward pulse wave) propagates from unit $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ to unit $\langle 0,0,\ldots,0\rangle$ to control the distribution phase. The frequency of clock waves can be set by dividing the system clock frequency of processor units $\langle 0,0,\ldots,0\rangle$ and $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$.

Figure 4 schematically illustrates the clock pulse distribution order for the control network of a 2D mesh system according to the BDW-clocking technique. To make the figure clear, separate units are indicated with squares and linkages between them are not shown. Note that the *d*-dimensional case can be understood in the same way.

To implement the BDW-clocking technique, two clocking networks are superposed on the control network, one for the reduction phase and the other for the distribution phase. The reduction and distribution clocking networks are responsible for transferring forward and backward pulse waves, respectively (Fig. 4).

Both these networks consist of identical cells, each cell corresponding to a particular barrier unit.

A cell of a clocking network is diagrammed in Fig. 5. The cell consists of a position D-type flip-flop and gates 1 and 2, an OR gate 1 and a univibrator UV 1. AND gate 1 has $d+1$ inputs at all and its first $d$ inputs are connected to the outputs of OR gates 1 of the transmitter neighbors. The output of OR gate 1 is wired to the corresponding inputs of AND gates 1 of the receiver neighbors. The univibrator's output is connected to input INC of the local reduction and the distributed slices (Fig. 2 and 3). The input terminals of the position flip-flop and AND gate 2 are connected to the local processor.

The position flip-flops remain clear across the entire mesh, except that of unit $\langle 0,0,\ldots,0\rangle$ in the forward clocking network and that of unit $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$ in the backward clocking network; these two flip-flops are set by unit position pulses received from the corresponding local processors. Such configuration of the position flip-flops in the forward clocking network guarantees that clock pulses are injected into this network by local processor $\langle 0,0,\ldots,0\rangle$ (AND gate 1 is blocked and gate 2 is open). Analogously, in the backward clocking network it is guaranteed that clock pulses are injected into this network by local processor $\langle k_1-1,k_2-1,\ldots,k_d-1\rangle$. At the same time, in the other cells

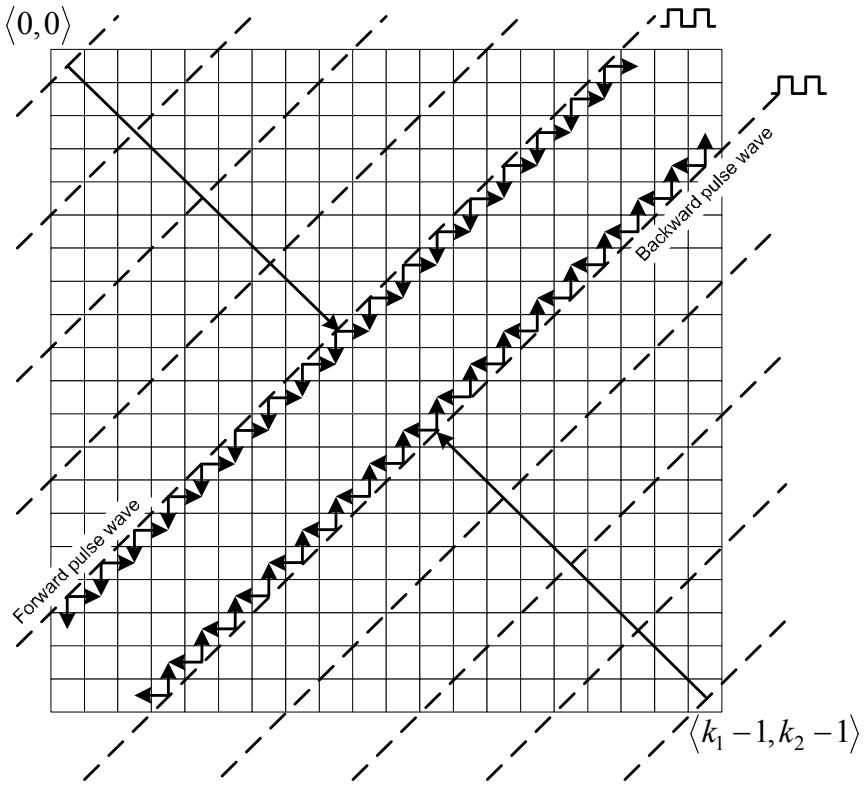


Fig. 4: The clock pulse distribution order for the control network of a 2D mesh system

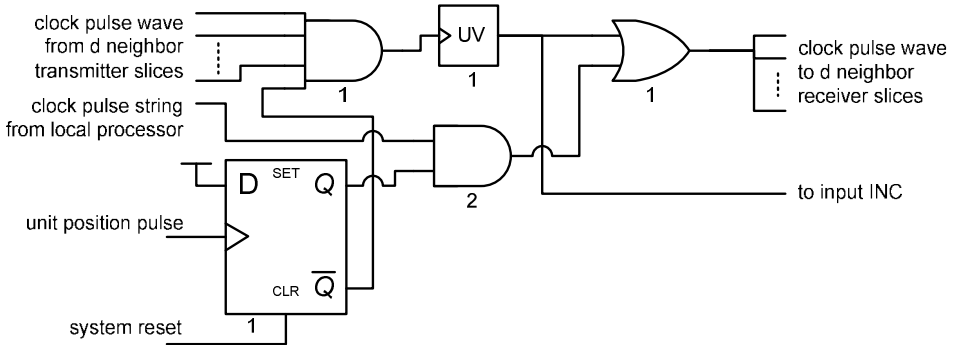


Fig. 5: The logical diagram of a cell of a clocking network (d-dimensional case)

Table 1: Comparison results

| Barrier method | Flexibility | Scalability | Applicability |
|---|---|---|---|
| The CM-5 barrier network (Thinking Machines, 1992) | Low (barrier groups of powers of 2 in size are acceptable only) | **Poor:** Physical reconfiguration of the barrier network is necessary when new processors are added to the system. | Not applicable to mesh topologies |
| Delgado-Kofuji's distributed hardware synchronizer (Delgado and Kofuji, 1996) | Low (barrier groups must be of rectangular shape) | **Fair:** The number of concurrent barriers is the key limitation. | Applicable to 2D meshes only |
| The MDBS network (Ramakrishnan *et al*., 1999) | Low (barrier groups must be mapped onto neighboring processors only) | **Good:** The number of barriers in the system is a limiting factor. | Applicable to 2D meshes only |
| The BTM network (Moh *et al*., 2001) | High (no limitations on barrier group configuration) | **Good:** The number of barrier registers is a limiting factor. | Applicable to 2D meshes only |
| DVBSS (Zotov, 2010) | High (no limitations on barrier group configuration) | **Fair:** Hardwired wraparound connections between corner units of the mesh need reconnection when new units are added. | Applicable to d-dimensional meshes |
| BVBSS (the proposed solution) | High (no limitations on barrier group configuration) | **Good:** The number of physical slices is a limiting factor. The local memory mapping scheme can alleviate the problem. | Applicable to d-dimensional meshes |

AND gate 2 remains blocked and the pulses issued by the local processors are ignored. Oppositely, in these cells AND gate 1 is maintained open, therefore the clock pulses from the transmitter neighbors are Added, the output of AND gate 1 eventually goes high and makes the univibrator UV produce a replica pulse. Thus the clock pulse wave is relayed to reach the next receiver neighbors in the forward/backward clocking network and affect the corresponding barrier units. This provides the clock pulse distribution order illustrated in Fig. 4.

## RESULTS AND DISCUSSION

The results of comparison of the proposed method to well-known hardware-level barriers are summarized in Table 1.

Table 1 shows that the proposed method is a more flexible alternative compared to the peers to barrier synchronize mesh-connected parallel systems of any dimension.

Taking into account the hardware-level implementation of the proposed method, we have estimated the BVBSS hardware-level complexity. We calculated the minimum number of generic logic gates (AND, OR, NOR, NAND) necessary to build up the barrier synchronization hardware depending on $d$, $m$, $p$, $k_1, k_2, \ldots, k_d$. Table 2 summarizes some calculation results regarding the 2D case (d = 2). To simplify calculations (with no loss of generality) we assumed that $k_1 = k_2$ and $m$ = p. Based on Table 2, one can see that hardware complexity limitations (VLSI limitations, in particular) are satisfied even in large meshes with high $m$ and $p$. For example, in a 32×32 mesh with m = p = 32 the hardware-level complexity of a barrier unit will be as low as 26024 logic gates (including the complexity of the clocking networks).

The synchronization latency of the proposed barrier synchronization method was evaluated through simulation. We conducted a series of simulation studies on a 32×32 mesh synthetic parallel system. The examined system was supposed to run a number of parallel applications, each having a dynamically changing set of processes. We assumed the "one-process-to-a-processor" allocation model during our experiments. During our simulation, barrier groups were instantiated and released randomly (the participants of barrier groups were picked at random). Synchronized groups were also randomly picked from the set of existing ones. Several synchronized groups were possible at the same time. The number of barrier episodes for each group was also taken at random. In our experiments, we measured the time required for a barrier state signal to travel from the last completed process of a barrier group to the last resumed participant of the same group. By summing up the startup and wakeup delays and average signal travel time, we calculated the synchronization latency of the proposed barrier mechanism. Resulted from our simulation, we found out that the synchronization latency stays as short as O (10) μsec depending on the number of instantiated barrier groups and slightly increasing with the virtual slice count $p$ (subject to the average gate delay is 5 nsec). This is known to be not worse than the latency of known hardware barriers.

Table 2: Hardware-level complexity of a 2D control network

| | $k_1 = k_2$ | | | |
|---|---|---|---|---|
| m, p | 4 | 8 | 16 | 32 |
| 4, 4 | 9792 | 39168 | 156672 | 626688 |
| 8, 8 | 31872 | 127488 | 509952 | 2039808 |
| 16, 16 | 112192 | 448768 | 1795072 | 7180288 |
| 32, 32 | 416384 | 1665536 | 6662144 | 26648576 |

## CONCLUSION

In the present study, we have presented a distributed hardware-level barrier synchronization method for $d$-dimensional mesh-connected parallel systems, the Bidirectional Virtual Bit-Slice Synchronizer (BVBSS). The proposed method is based upon a specific virtualization scheme making it possible to have $p$ virtual control networks (slices) superposed on a physical one, while there may be up to $m$ physical slices operating in parallel. Each virtual slice can be

dynamically allocated to any barrier group. Our method employs a specific wave clocking technique to switch between virtual slices in a parallel pipeline fashion, sending two series of pulse waves originating from two corner processors across the mesh to provide faster barrier operation.

The BVBSS scheme has been shown to be more flexible than the existing hardware barriers; it accepts dynamically defined (possibly overlapping) barrier groups of arbitrary size and shape and noncontiguous group member allocation is possible. Our simulation study has proved the BVBSS to be as fast as the other hardware barrier synchronization models making it possible to synchronize arbitrary processes in O (10) μsec in a 32×32 mesh parallel system.

## NOMENCLATURE

| | | |
|---|---|---|
| $d$ | : | The number of dimensions in the mesh |
| $k_i$ | : | The width of an $i^{th}$ dimension of the mesh |
| $\langle x_1, x_2, \ldots, x_d \rangle$ | : | The coordinates of a unit along the dimensions |
| $m$ | : | The number of physical barrier slices in a unit |
| $p$ | : | The number of virtual slices in each physical slice |
| $i, j$ | : | Indices |
| $f_i$ | : | The current barrier group for virtual slice $i$ |
| $b(f_i)$ | : | The current barrier for barrier group $f_i$ |
| IN1.$i$, IN2 | : | The input terminals of the barrier unit |
| OUT | : | The fan-out output of the barrier unit |

## REFERENCES


Al-Azzeh, J.S., 2013. Review of methods of distributed barrier synchronization of parallel processes in matrix VLSI systems. Int. Rev. Comput. Softw., 8(4): 927-932.

Ashraf, A.K., I.V. Zotov, A.M. Hazem and D.E. Skopin, 2012. Distributed barrier synchronization procedure with the dynamic limitation of the coordinating signal propagation area. Int. Rev. Comput. Softw., 7(3): 991-995.

Axelrod, T.S., 1986. Effects of synchronization barriers on multiprocessor performance. Parallel Comput., 3: 129-140.

Cohen, W.E., D.W. Hyde and R.K. Gaede, 2000. An optical bus-based distributed dynamic barrier mechanism. IEEE T. Comput., 49(12): 1354-1365.

Delgado, M. and S.T. Kofuji, 1996. A distributed barrier synchronization solution in hardware for 2D-mesh multicomputers. Proceeding of the 3rd International Conference on High Performance Computing, pp: 368-373.

Forum, M.P.I., 2012. MPI: A Message-passing Interface Standard. Version 3.0, Message Passing Interface Forum. Retrieved from: //http://www.mpi-forum.org/docs/docs.html.

Hindam, T., 2004. Connecting the distributed hardware agents for barrier synchronization operation. Proceeding of the International Conference on Electrical, Electronic and Computer Engineering, pp: 261-264.

Johnson, T.A. and R.R. Hoare, 2001. Cyclical cascade chains: A dynamic barrier synchronization mechanism for multiprocessor systems. Proceeding of the 15th International Parallel Distributed Processing Symposium, pp: 2061-2068.

Li, J., J.F. Martinez and M.C. Huang, 2004. The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors. Proceeding of the 10th International Symposium on High Performance Computer Architecture, pp: 14-23.

Moh, S., C. Yu, D. Lee, H.Y. Youn, D. Han and D. Lee, 2001. Four-ary tree-based barrier synchronization for 2D meshes without nonmember involvement. IEEE T. Comput., 50(8): 811-823.

OpenMP Architecture Review Board, 2011. OpenMP Application Program Interface Version 3.1. Retrieved from: //http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

Ramakrishnan, V., I.D. Scherson and R. Subramanian, 1999. Efficient techniques for nested and disjoint barrier synchronization. J. Parallel Distr. Com., 58(8): 333-356.

Sampson, J., R. González, J.F. Collard, N.P. Jouppi, M. Schlansker and B. Calder, 2006. Exploiting fine-grained data parallelism with chip multiprocessors and fast barriers. Proceeding of the 39th Annual IEEE/ACM International Symposium Microarchitecture, pp: 235-246.

Thinking Machines, 1992. The Connection Machine CM-5 Technical Summary. Thinking Machines Corporation, Cambridge, MA.

Tsafrir, D. and D.G. Feitelson, 2002. Barrier synchronization on a loaded SMP using two-phase waiting algorithms. Proceeding of the International Parallel Distributed Processing Symposium, pp: 80-87.

Tzeng, N.F., B. Kasula and H. Wu, 2005. Efficient barrier synchronization on wireless computing systems. Proceeding of the 11th International Conference on Parallel Distributed Systems, pp: 782-788.

Zotov, I.V., 2010. Distributed virtual bit-slice synchronizer: A scalable hardware barrier mechanism for n-dimensional meshes. IEEE T. Comput., 59(9): 1187-1199.