

# ClusterNet: An Object-Oriented Cluster Network

Raymond R. Hoare

Department of Electrical Engineering, University of Pittsburgh  
Pittsburgh, PA 15261  
hoare@pitt.edu

**Abstract.** Parallel processing is based on utilizing a group of processors to efficiently solve large problems faster than is possible on a single processor. To accomplish this, the processors must communicate and coordinate with each other through some type of network. However, the only function that most networks support is message routing. Consequently, functions that involve data from a group of processors must be implemented on top of message routing. We propose treating the network switch as a function unit that can receive data from a group of processors, execute operations, and return the result(s) to the appropriate processors. This paper describes how each of the architectural resources that are typically found in a network switch can be better utilized as a centralized function unit. A proof-of-concept prototype called ClusterNet<sub>4EPP</sub> has been implemented to demonstrate feasibility of this concept.

## 1 Introduction

In the not-so-distant past, it was common for groups of people to pool their resources to invest in a single, high-performance processor. The processors used in desktop machines were inferior to the mainframes and supercomputers of that time. However, the market for desktop computers has since superseded that of mainframes and supercomputers combined. Now the fastest processors are first designed for the desktop and are then incorporated into supercomputers. Consequently, the fastest processors, memory systems, and disk controllers are packaged as a single circuit board. Thus, the highest performance "processing element" is a personal computer (PC).

Almost every individual and company uses computers to help them be more efficient. Networks enable seemingly random connections of computers to communicate with each other and share resources. Computational and data intensive applications can utilize the resources of a cluster of computers if the network is efficient enough. If the network is inefficient, the added communication and coordination cost reduces, or even removes, the benefit of using multiple computers. As more computers are used to execute an application in parallel, the extra overhead eventually removes the performance benefit of the additional resources. Thus, for a cluster of computers to be

used as a single parallel processing machine, they must be able to efficiently communicate and coordinate with each other.

Stone, in his popular book on high-performance computer architecture [1], states that peak performance of a parallel machine is rarely achieved. The five issues he cited are:

1. Delays introduced by interprocessor communications
2. Overhead in synchronizing the work of one processor with another
3. Lost efficiency when one or more processors run out of tasks
4. Lost efficiency due to wasted effort by one or more processors
5. Processing costs for controlling the system and scheduling operations

These issues are relevant for all computer architectures but are particularly troublesome for clusters. Clusters typically use commodity network switches that have been designed for random configurations of computers and routers. Thus, packets are used to encapsulate data and provide routing information. This software overhead accounts for 40-50% of the total communication time[2].

Network switches are designed for typical network traffic. Rarely will every incoming packet be destined for the same output port for a sustained period of time. The outgoing network link would become saturated, its buffers will become full, the switch will have to drop incoming packets, and the packets will have to be resent. While this is extremely rare for a typical network, the *gather* and *all-to-all* communication operations require this type of communication pattern[3].

Processor coordination and synchronization are group operations that require information from each processor. While such operations can be executed using message passing communications, a total of  $N$  messages must be sent through the network to gather the data and broadcast the result(s). These communication operations can ideally be overlapped to require only  $(\log_2 N + 1)$  communication time steps.

Processor scheduling and control are also operations that require data from each of the processors. However, this information must be maintained to ensure even load distribution. The algorithms used for scheduling and controlling clusters are not computationally intensive but require efficient access to every computer's status. Operations that involve data from a collection of computers are defined as *aggregate operations* [4]. Ironically, the architectural characteristics of a typical switch are well suited for executing aggregate operations. A typical modern switch interconnects 8 to 80 computers, contains a processor (or routing logic), and stores routing information in a lookup table. For a 32 or 64-processor cluster, a single switch is capable directly interconnecting all of the computers.

Rather than assuming that a cluster is a purely distributed memory architecture that communicates through a point-to-point network, this paper examines the entire cluster architecture, including the network switch to demonstrate how a better cluster architecture can be created. Specifically, the architectural resources contained in a typical switch will be examined, reallocated and/or changed, to facilitate efficient communication and control of the entire cluster. As shown in the following table, the architectural features of a network switch and a cluster are almost exact opposites. In fact, the architectural characteristics are almost exact opposites. The proposed ClusterNet

architecture utilizes these differences to form a new architecture that is a complement of both distributed and shared memory, as well as a complement of parallel and serial program execution.

**Table 1.** Architectural features of network switches and a cluster of computers.

Architectural Feature	Network Switch	Cluster
Number Of Processors	1	16, 32, 64
I/O Ports Per Processor	8-80	1-3
Memory Architecture	Shared	Distributed
Storage	Lookup Table	RAM & Disk
Functionality	Fixed	Programmable
Execution Model	Serial	Parallel
Topology	Unknown	Star Topology
Performance Criteria	Packets per Second	Seconds per Communication
Communication Pattern	Point-To-Point	Point-To-Point & Collective

## 2 ClusterNet

While network switches can be used to facilitate cluster communications, there are a number of architectural differences between the network switch and the rest of the cluster. By combining these two architectures, a more efficient cluster architecture called ClusterNet, can be built.

Rather than limiting the network switch to routing packets, we propose expanding the role of the switch to execute functions on data gathered from a group of processors. Furthermore, because of the switch's memory architecture, it should also be able to store data. Thus, by combining data storage with computation, an *object-oriented* cluster network can be created. To simplify our discussion, our new object-oriented switch will be labeled an *aggregate function unit (AFU)* and the unmodified network switch will just be called a *switch*.

The goal of this paper is to demonstrate how the resources of a switch can be more efficiently utilized when placed within the context of a cluster architecture. Table 2 shows how ClusterNet's usage of architectural resources differs from a switch.

**Table 2.** ClusterNet's usage of architectural resources.

Architectural Resource	Switch Usage	AFU Usage
Routing Logic	Route Messages	Execute Functions
Switch Memory	Address Lookup Table	Data Structures
Switch Port	Input/Output Packet Queue	Register Interface
Physical Link	Send/Receive Packets	Send/Receive Data
Software Interface	Send/Receive Messages	Access To AFU Port
Application Interface	MPI	Aggregate Functions

The remainder of this section will discuss each of the resources listed above and how they can be used to provide a more robust cluster architecture called ClusterNet. Section 3 describes a proof-of-concept four-processor prototype that was built. Section 4 describes related work and section 5 offers conclusions and future directions.

## 2.1 Functionality: Router vs. Aggregate Function Execution

The routing logic (or processor) can collect and distributed information from every processor because most network switches interconnect between 4 and 80 computers. However, cluster implementations have maintained a distributed-memory architecture in which the processors communicate through message passing.

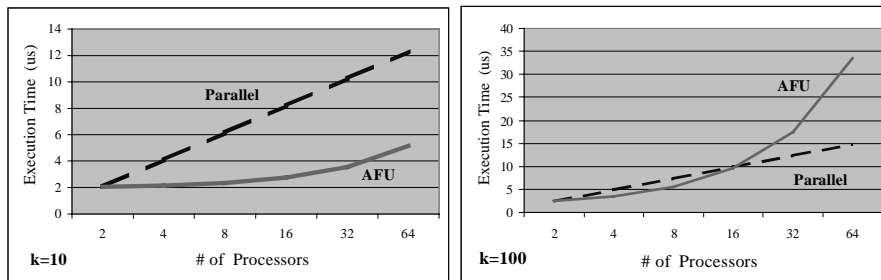
Ironically, group operations such as *Global Sum* are implemented by sending  $N$  messages through the same switch in  $\log_2 N$  time steps. Each time step requires a minimum of a few microseconds, over 1000 processor cycles. Rather than performing the computation, the network switch is busy routing packets.

Instead of using the network switch's processor to route messages, the processor can be used to execute functions within the network. Because the switch is directly connected to each of the processors, data from every processor can be simultaneously sent into the network switch. Upon arrival, the specified function is computed and the result is returned to each processor.

To quantify this proposition, we define the following variables:

- $N$  - The number of processors in the cluster (2 - 64).
- $\alpha$  - The communication time between a processor and the switch (1 $\mu$ s).
- $k * (N-1)$  - The number of instructions to be executed.
- $\epsilon$  - The amount of time required to compute a single instruction (5 ns).

If an associative computation is executed using  $N$  processors and a point-to-point network, the amount of time required is approximately  $(2\alpha + k\epsilon) * \log_2 N$  because computation can be overlapped. If the switch's processor is used to execute the same function, the amount of time required is  $(2\alpha + (N-1)k\epsilon)$ . From an asymptotic perspective, it is better to use all  $N$  processors rather than the AFU's processor. However, when typical values are used ( $\alpha = 1\mu$ s,  $\epsilon = 5$ ns) the resulting graphs show the performance tradeoffs as we change  $k$  and  $N$ , shown in Fig. 1.



**Fig. 1** Collective computation using the AFU verses using all  $N$  processors for  $k=10$  and  $100$ ,  $\alpha = 1\mu$ s,  $\epsilon = 5$ ns.

## 2.2 Network Storage: Routing Tables vs. Network-Embedded Data Structures

To enable a switch to be used for any network topology, it must be able to change how it routes different packets. This is typically implemented through a lookup table. When a packet is received, its destination address is used as an index into the lookup table to determine which port the packet should be routed to. This information can also be changed because network configurations change.

In a cluster architecture, the routing lookup table is of minimal use because each processor is directly connected to the switch. If we require that processor  $i$  be attached to port  $i$  then there is no need for a routing table. The network-embedded memory can then be used as a cluster resource.

For example, the lookup table could be used to track cluster-wide resources. If a resource is needed, the lookup table could be used to determine where the resource is located. This concept can be used to implement a dynamically-distributed shared-memory cluster architecture. In a distributed shared-memory architecture (i.e. Cray T3D) there is a single memory address range that is distributed across all of the processors. Each processor can access any portion in memory by simply specifying a memory address. However, this results in non-uniform memory access times. Direct memory access was not built into the Cray T3E. A *dynamically*-distributed shared-memory still uses a single address range but allows blocks of memory to migrate to the processor that needs them. When a memory request is made, the entire block of memory is relocated and placed in the local memory of the requesting processor. For regular access patterns, this drastically improves performance.

However, there is an inherent contradiction within the dynamically-distributed shared-memory architecture. A shared resource table is needed to determine where each block is located. To share this location table, it too must be placed in shared memory. The location table can be distributed across the processors but requires two requests for every memory access. If the switch's lookup table is used for the location table, memory requests could be sent to the network and the network could forward them to the processor that currently owns the block.

In addition to a lookup table, the network-embedded memory can be used to represent any number of useful data structures. Synchronization data structures can be used to implement static, dynamic and directed synchronization. A processor load table can be kept in the network to facilitate dynamic task allocation to the least loaded processor. Queues and priority queues can also be used for task allocation and load balancing. Even shared linked-lists can be implemented with a small amount of additional control logic.

## 2.3 Network Port Interface: I/O Queues vs. Register Interface

Because all networks use packets, they also contain I/O queues to store the packets until the router logic is able to handle them. The drawback to this is that the queues

become full and overflow. Our design does not require queues because it does not route packets.

The AFU does, however, execute functions and does transmit data. As shown in Fig. 4, the interface to the AFU appears as four registers. The OpCode register is used to specify which function is to be executed. The Data registers are used to move data between the PC and the AFU. Function parameters and function results are passed through these registers using the full/empty bits to indicate valid data. The Counter register can be used as a function parameter and is useful when accessing the network-embedded data structures described earlier. The Counter is particularly useful when accessing adjacent locations in memory. When a word has been read from memory the counter automatically increments. In this way, streaming read and write operations can be implemented easily by setting the appropriate OpCode and sending/receiving an entire block of data.

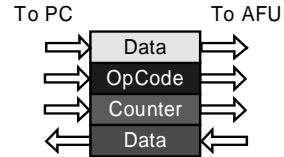


Fig. 3. The AFU Interface Port

Table 3. Latencies (in us) for point-to-point messages for several architectures.

Platform	Latency (in $\mu$ s)	Send Overhead	Receive Overhead	Ref.
IBM SP2	39	-	-	[5]
Intel Paragon	6.5	21.5%	33.8%	[2]
Meiko 7CS-2	7.5	22.7%	21.3%	[2]
Cray T3D	2.2	-	-	[6]
Memory Channel	5-20	-	-	[7]
Myrinet	11.2	17.9%	23.2%	[2]
SHRIMP	10+	-	-	[8]
ParaStation	5+	-	-	[9]
PAPERS	3-5	-	-	[10]
ClusterNet <sub>JEP</sub>	1.7-5.2	-	-	[11]

## 2.4 Software Interface: Packet vs. Direct Read and Write

As was shown in the Table 3, the software overhead for sending and receiving a message consumes 40-50% of the overall message latency. This is due to the time spent encoding and decoding packets. Rather than accepting this overhead, we propose expanding the functionality of the network and simplifying the network interface. Most architectures layer their communication libraries on top of point-to-point primitives that encode, send and decode packets. ClusterNet executes functions within the network and can be used to execute collective communications within the network.

As a result of executing functions within the network hardware, the software interface is very simple and only requires seven assembly-level instructions listed below. Lines 1 and 2 are used to set the OpCode and Counter registers. The OpCode is used to specify which function should be executed. If the OpCode has not changed, these

registers do not need to be set. After data is placed into the network, the function is executed and the results are returned. This architecture relies on the fact that the network link between the processor and the AFU perform error detection and correction.

```

1. I/O Write (OpCode) /* Optional */
2. I/O Write (Count) /* Optional */
3. I/O Write (Data)
4. I/O Read (Result)
5. if ( Result == NOT_A_NUMBER) goto line 4
6. if ( Result != PREFIX_TOKEN ) goto line 8
7. I/O "Data" Read (Result)
8. /* The Aggregate Function has completed. */

```

### 3 The ClusterNet<sub>4EPP</sub> Proof-of-Concept Prototype

The four-processor Object-Oriented Aggregate Network[11], called ClusterNet<sub>4EPP</sub>, demonstrates that the simplified network interface is feasible and performs very well using a small FPGA (Altera 10K20). The PCs' parallel ports were used as the network interface and require approximately 1μs to access. Experimental results were performed and a PCI device was accessible in approximately 450 ns. For ClusterNet<sub>4EPP</sub>, read and write access time to each of the four registers (Data In, OpCode, Counter, Data Out) was found to be 1.7 μs. IEEE 1284 in EPP mode was used for cable signaling.

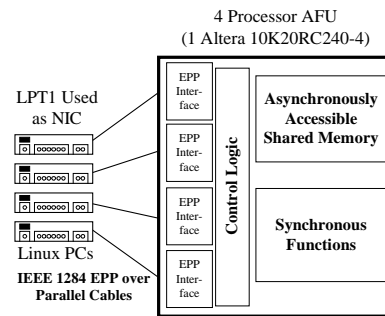


Fig. 4. ClusterNet<sub>4EPP</sub>

To demonstrate that network-embedded data structures are feasible and beneficial, an embedded RAM block was placed inside the FPGA. The control logic for the RAM block was modified and the synchronization/ arithmetic operations shown below were implemented. Each operation is executed on a single memory location. While a processor was not placed in the network, these operations can be used to perform simple global operations.

Experimentation was performed to determine the effect of memory contention but due to the small number of processors and a 120 ns memory-access time, no effect could be detected. All memory accesses required approximately 1.7 μs. If the Op-Code and the Counter need to be set, the total execution time is 5.2 μs. All of the memory operations can be executed on any word in the embedded memory. In addition to memory operations, barrier synchronization and a number of reduction operations were implemented. These operations are described in Table 4.

**Table 4.** Memory operations for the RAM embedded within ClusterNet<sub>JEPF</sub>

Memory Operations
Non-blocking Memory Read / Exchange
Wait for Lock=1 (or 0 ) then Read
Wait for Lock=0 (or 1), Exchange and set Lock=1
Non-blocking Write and Unlock/Lock
Wait for Lock=0, OR with RAM
Wait for Lock=0, XOR with RAM
Wait for Lock=0, Decrement/ Decrement RAM
Wait for Lock=0, RAM = RAM -/+ Data

## 4. Related Research

The NYU Ultracomputer [12, 13] and the IBM RP3 [14] are both dance-hall shared memory architectures. The Ultracomputer was the first architecture to propose that combining be performed within the processor-to-memory interconnection network. Messages that reference identical memory locations are combined if both messages are buffered within the same switch at the same time. The computations that can be performed within the interconnection network are Fetch-and-Add, Fetch-and-Increment, and other Fetch-and-Op functions, where Op is associative.

Active Messages from Berkeley [2] allow functions to be executed at the network interface on the local or remote node. Active Networks perform operations on data values that are passed through the network [15, 16]. Fast Messages[17] modify the network interface drivers to reduce the overhead for sending and receiving messages. Sorting networks were introduced in [18] and have continued to remain a topic of interest [19, 20]. Multistage data manipulation networks are discussed in [21].

A number of commercial architectures have included direct support for various associative aggregate computations. The Cray T3D directly supports, through Cray-designed logic circuits, barrier synchronization, swap, and Fetch-and-Increment [22]. The TMC Connection Machine CM-5 has a control network that supports reduction operations, prefix operations, maximum, logical OR and XOR [22]. These architectures can be considered aggregate networks but they are very specific in the functions that they are designed to execute.

PAPERS, Purdue's Adapter for Parallel Execution and Rapid Synchronization, is a network that allows a number of aggregate computations to be performed within a custom network hub that is attached to a cluster of Linux PCs [23-25]. This design uses a combination of barrier synchronization with a four-bit wide global NAND to construct a robust library of aggregate computations and communications.

A number of cluster projects have employed different approaches to reduce the communication cost of point-to-point and broadcast messages. SHRIMP [8, 26] uses memory bus snooping to implement a virtual memory interface. Point-to-point mes-

sages in SHRIMP have a 10+  $\mu\text{s}$  latency and remote procedure calls have a 3+  $\mu\text{s}$  latency. Myrinet [27] provides gigabit bandwidth with a 0.55  $\mu\text{s}$  worst-case latency through its pipelined crossbar switch.

## 5. Conclusions and Future Directions

This paper has proposed the concept of combining the architectural characteristics of a network switch with that of a cluster of desktop computers. Rather than using the resources of the switch for message routing, this paper has proposed using them to create a function unit that is capable of performing computations on data that is aggregated from a group of processors. Specifically, the following switch resources can better serve the architectural needs of a cluster in the following way:

- The switch lookup table should be used as network-embedded shared memory.
- The functionality of the switch should be expanded to include aggregate functions. This reduces the total amount of time required for group computations.
- The functionality of the switch should be configurable. This will enable greater utilization of the architectural resources of the entire cluster rather than just the processors.
- Packets are not needed if each processor has direct access to a set of registers within the "switch". This remove the need to encode and decode packets and reduces the software overhead to less than ten assembly-level instructions.

ClusterNet<sub>4EPP</sub> was described and implements numerous instructions that access the shared memory in as little as 1.7 $\mu\text{s}$ . A number of functions were implemented that involved data from all of the processors. These functions included OR, XOR, AND and ADD. While ClusterNet<sub>4EPP</sub> has demonstrated that it is possible to implement functions within the network, there are still a number of issue that have not been addressed. Scalability to large systems has not been demonstrated and the performance of complex functions is still unknown.

Scalability and function performance are currently being examine using an Altera 10K100 that is five times larger that the 10K20 and is currently able to interconnect 8 processors. The figure to the left shows the 10K100 prototype in the left portion of the picture and four of the connectors in the right portion of the picture. The cable in the middle of the picture with the Altera label is the FPGA configuration cable. The EPP in currently working and the remainder of the design is expected to be completed by SuperComputing '99 in the middle of November.

Future directions include using a higher bandwidth physical layer and a PCI network interface card. Each of these areas are under development but experimental results have not been obtained yet. Additionally, embedding a DSP or RISC processor into the network would enable rapid experimentation with system-level resource management. After that is achieved, user-level programmability of the AFU will be approached.

## References

1. H. S. Stone, *High-Performance Computer Architecture*, Third ed. Reading, MA: Addison-Wesley Publishing Company, 1993.
2. D. Culler, L. Liu, R. Martin, and C. Yoshikawa, "Assessing Fast Network Interfaces," *IEEE Micro*, vol. 16, pp. 35-43, 1996.
3. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI, The Complete Reference*. Cambridge, Massachusetts: The MIT Press, 1996.
4. R. Hoare and H. Dietz, "A Case for Aggregate Networks," *Proceedings of the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, Orlando, FL, 1998.
5. C. Stunkel and e. al., "The SP2 High-Performance Switch," *IBM Systems Journal*, vol. 34, pp. 185-204, 1994.
6. R. Kessler and J. Schwarzmeier, "Cray T3D: a New Dimension for Cray Research," *Proceedings of the In Digest of Papers. COMPCON Spring '93*, San Francisco, CA, 1993.
7. M. Fillo and R. Gillett, "Architecture and Implementation of Memory Channel 2," *Digital Equipment Corporation High Performance Technical Computing*, pp. 34-48, 1997.
8. M. Blumrich and e. al., "Virtual Memory Mapped Network Interfaces for the SHRIMP Multicomputer," *Proceedings of the The 21st Annual International Symposium on Computer Architecture*, 1994.
9. T. Warshko, W. Tichy, and C. Herter, "Efficient Parallel Computing on Workstation Clusters," *University of Karlsruhe, Dept. of Informatics, Karlsruhe, Germany Technical Report 21/95*, 1995.
10. R. Hoare, T. Mattox, and H. Dietz, "TTL-PAPERS 960801, The Modularly Scalable, Field Upgradable, Implementation of Purdue's Adapter for Parallel Execution and Rapid Synchronization," *Purdue University, W. Lafayette, Internet On-line Tech Report*, 1996.
11. R. Hoare, "Object-Oriented Aggregate Networks," in *School of Electrical Engineering. W. Lafayette: Purdue University*, 1999.
12. A. Gottlieb and e. al., "The NYU Ultracomputer, Designing a MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers*, pp. 175-189, 1983.
13. R. Bianchini, S. Dickey, J. Edler, G. Goodman, A. Gottlieb, R. Kenner, and J. Wang, "The Ultra III Prototype," *Proceedings of the Parallel Systems Fair*, 1993.
14. G. Pfister and V. Norton, "'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *Proceedings of the 1985 International Conference on Parallel Processing*, 1985.
15. D. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," *Computer Communications Review*, vol. 26, 1996.
16. D. Tennenhouse and e. al., "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, pp. 80-86, 1997.
17. H. Bal, R. Hofman, and K. Verstoep, "A Comparison of Three High Speed Networks for Parallel Cluster Computing," *Proceedings of the First International Workstion on Communication and Architectural Support for Network-Based Parallel Computing*, San Antonio, TX, 1997.
18. K. Batcher, "Sorting Networks and Their Applicaitons," *Proceedings of the Spring Joint Computer Conference*, 1968.
19. J. Lee and K. Batcher, "Minimizing Communication of a Recirculating Bitonic Sorting Network," *Proceedings of the the 1996 International Conference on Parallel Processing*, 1996.

20. Z. Wen, "Multiway Merging in Parallel," IEEE Transactions on Parallel and Distributed Systems, vol. 7, pp. 11-17, 1996.
21. H. J. Siegel, Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Edition ed. New York, NY: McGraw-Hill, 1990.
22. G. Almasi and A. Gottlieb, Highly Parallel Computing, Second Edition. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1994.
23. H. Dietz, R. Hoare, and T. Mattox, "A Fine-Grain Parallel Architecture Based on Barrier Synchronization," Proceedings of the International Conference on Parallel Processing, Bloomington, IL, 1996.
24. R. Hoare, H. Dietz, T. Mattox, and S. Kim, "Bitwise Aggregate Networks," Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing, New Orleans, LA, 1996.
25. T. Mattox, "Synchronous Aggregate Communication Architecture for MIMD Parallel Processing," in School of Electrical and Computer Engineering. W. Lafayette, IN: Purdue University, 1997.
26. E. Felten and e. al., "Early Experience with Message-Passing on the SHRIMP Multicomputer," Proceedings of the The 23rd Annual International Symposium on Computer Architecture, Philadelphia, PA, 1996.
27. N. Boden and e. al., "Myrinet: A Gigabit per Second Local Area Network," in IEEE-Micro, vol. 15, 1995, pp. 29-36.
28. R. Brouwer, "Parallel algorithms for placement and routing in VLSI design", Ph. D. Thesis, University of Illinois, Urbana-Champaign, 1991.
29. J. Chandy, et. al. "Parallel Simulated Annealing Strategies for VLSI Cell Placement", in Proceedings of the 1996 International Conference on VLSI Design, Bangalore, India, January 1996.
30. T. Stornetta, et. al., "Implementation of an Efficient Parallel BDD Package", Proc. 33rd ACM/IEEE Design Automation Conference, 1996.
31. R. Ranjan, et. al., "Binary Decision Diagrams on Network of Workstations", In Proceedings of the International Conference on Computer-Aided Design, pp. 358-364, 1996.