Routing Algorithms for IBM SP1

Bülent Abali[1] and Cevdet Aykanat[2]

[1] IBM Thomas J. Watson Research Center,
P.O.Box 218, Yorktown Heights, NY 10598
[2] Bilkent University, 06533 Ankara, Turkey

Abstract. Scalable multicomputers are based upon interconnection networks that typically provide multiple communication routes between any given pair of processor nodes. Routes must be selected for communication so that the load is distributed evenly among the links and switches to prevent congestion in the network. We describe the route selection algorithm used in the IBM 9076 SP1 multicomputer. We then describe a new algorithm for reducing network congestion and compare the two algorithms.

1 Introduction

Scalable multicomputers are based upon interconnection networks that typically provide multiple communication routes between any given pair of processor nodes. Multiple routes provide low latency, high bandwidth, and reliable interprocessor communication. In such networks, the selection of the routes is important because of its impact on the communication performance. Routes must be selected so that the communication load is distributed evenly among the links and switches to prevent congestion in the network. In this paper we describe the route selection algorithm used in the IBM 9076 SP1 multicomputer. We then describe an experimental algorithm for reducing network congestion and compare the two algorithms. In the next section we give an overview of the SP1 network architecture. In Section 2 we describe the SP1 routing algorithm. In Section 3 we describe the experimental routing algorithm and compare the two algorithms in Section 4.

1.1 The SP1 Network Architecture

The 9076 SP1 is a commercially available multicomputer whose communication architecture is based upon the Vulcan architecture [1]. The SP1 processor nodes attach to a multistage interconnection network consisting of 8 input 8 output non-blocking switches [1]. The switch chip shown in Fig.3 consists of 8 receiver and 8 transmitter modules, an unbuffered 8×8 crossbar, and a 1-KByte large central queue. Each input and output port consists of 8 data lines and 2 control lines. Processor nodes communicate by sending and receiving message packets. Packets are of variable length with up to 255 bytes in size. The method of packet transfer is similar to *wormhole routing* [2], with the difference in that when a packet is blocked the packet bytes are not buffered in place but they are temporarily transferred to the central queue until the blocked output port is cleared up. The method of packet transfer also differs from *virtual cut-through* technique [3, 2] in that flow control is byte based, not packet based. When there is no output contention packet bytes pass through the switch chip via crossbar in 5 clock cycles. Packets are formatted such that the first byte of each packet indicates the packet length, followed by a number of routing bytes, followed by data. The *source routing* technique is used for routing packets [4]. In this technique, the source processor node determines the complete route and puts the respective route bytes in the packet. As the packet proceeds to its destination, each switch chip examines the first route byte of the packet and determines the destination output port. The switch chip also strips off the portion of the routing information pertaining to itself. The packet has no route bytes remaining upon arriving at the destination node. In the SP1 implementation, the switch chip operates at 40 MHz, resulting in a peak bandwidth of 40 MB/s per port and ports may be interconnected with links over 100 feet in length enabling the easy construction of large networks.

In the network implementations, the switch chip input port i and output port i are paired together to form a full duplex bidirectional channel. The resulting 4×4 bidirectional switch element can forward a packet to any of the 8 output ports, including the output ports on the same side with the input port (called "turn-around routing"). In this respect, the SP1 network topologies differ from more commonly known unidirectional multistage interconnection networks (MIN) such as the Omega and indirect binary n-cube [5, 6]. Bidirectionality enhances the modularity, fault-tolerance, and diagnosis of the network as described in [1]. Eight switches placed in a 2-stage configuration interconnected with a shuffle form the switch board as shown in Fig. 4. The switch board provides full connectivity; it can route a packet from any 32 input ports to any 32 output ports. Switch boards may be interconnected in various ways to construct larger networks. A 16 node network is constructed using only one switch board with the 16 processor nodes attached to the left hand side of the board and the 16 ports on the right hand side unused. A 32 node network is constructed using two switch boards whose right hand sides are interconnected with straight wires. Examples of 128 node and 256 node networks are shown in Figs. 5 and 6. Custom network topologies of any size can be easily constructed due to the link technology used.

2 The SP1 Routing Algorithm

We developed the SP1 routing algorithm originally for the Vulcan prototype [1]. A modified version of the algorithm is also being used in IBM's recently announced SP2 multicomputer. The SP1 routing algorithm is a simple algorithm that selects a single shortest path between each pair of processor nodes, although multiple shortest paths may exist. In this respect, the SP1 routing algorithm is comparable to the commonly known XY routing algorithm for 2-

dimensional meshes and the *e-cube* routing algorithm for hypercubes [2]. In a 2-dimensional mesh, the XY routing algorithm uses the single route that goes along the X dimension first and then along the Y dimension, although two nodes have $(hx + h_y)!/h_x!h_y!$ different shortest paths from one to another, where h_x and h_y are the internode distances in the X and Y dimensions, respectively. In the hypercube topology, the e-cube algorithm uses the single route that goes along the increasing order of dimension, although two nodes with a Hamming distance of k have $k!$ shortest paths from one to another.

The shortest path routing is not necessarily the best choice for all communication patterns [7]. However, in the absence of any information on communication patterns, we decided to use the shortest paths since fewer switches and links would be used. We use the modified *Breadth-First Search* algorithm shown in Fig. 1 for building a breadth-first spanning (BFS) tree rooted at each source node (src), and then we follow the spanning tree paths to find the shortest paths from the source node to the rest of the processor nodes. The algorithm is originally due to [8] and uses a first-in, first-out (FIFO) queue Q for the breadth-first search. We added a simple static load balancing strategy to ensure that links are included in the selected routes in a balanced manner. The network graph $G = (V, E)$ is represented by a linked list of vertices. Each vertex $v \in V$ represents a processor node or a switch, and each arc $e \in E$ represents a half duplex link. Only the non-faulty links and switches are represented in G. The direction of an arc indicates the direction of message transmission. Each switch vertex has a maximum in-degree of 8 and an out-degree of 8, and each processor vertex has an in-degree of 1 and an out-degree of 1. The $u.parent$ field indicates the parent of vertex u in the spanning tree, and $u.distance$ indicates the distance of vertex u to the root (the source node) of the tree. The $u.port[i]$ field represents the vertex attached to the output port i of vertex u, and hence also represents the arc from vertex u to vertex $u.port[i]$.

Load balancing is facilitated by the $u.portusage[i]$ field which indicates how many times an output port has been used during route generation. While building a spanning tree from a given source node, each time a source–destination path is found the *portusage* field is incremented for each output port in that path. Usage count of the ports determine the order of breadth-first search from the next source node, such that from a given vertex v we first visit the vertices adjacent to the least frequently used output ports (i.e. with the smallest counts), which is accomplished by sorting the port usage counts in lines 10–17.

The routes are stored in a route table in each processor's memory. The route table approach enables routing to be done in a topology independent fashion. Note also that by design the SP1 routing algorithm does not assume a topology, whereas the e-cube and the XY routing algorithms assume hypercube and 2-dimensional mesh topologies, respectively. Topology independence is important for fault-tolerance and scalability; missing links and switches are handled properly by the SP1 routing algorithm, and larger networks of different topological properties can be implemented easily without having to change the routing hardware or the algorithm. Although, the SP1 routing algorithm attemps to in-

```
RTG(G) /* Route Table Generator */
    for each vertex u ∈ V[G]
        for i = 0 to 7
            u.portusage[i] ← 0
    for each vertex src ∈ V[G]
        BFS_RTG(G, src)

BFS_RTG(G, src)
    for each vertex u ∈ V[G]
        u.visit ← WHITE
        u.distance ← 0
        u.parent ← NIL
    src.visit ← GRAY
    ENQUEUE(Q, src)
    while Q ≠ ∅
        u ← head[Q]
        if u.type = SWITCH then
            for i = 0 to 7
                index[i] ← i
            for j = 7 to 1
                for i = 0 to j − 1
                    if u.portusage[i] > u.portusage[i + 1] then
                        tmp ← index[i]
                        index[i] ← index[i + 1]
                        index[i + 1] ← tmp
            for j = 0 to 7
                i ← index[j]
                v ← u.port[i]
                if v ≠ NIL AND v.visit = WHITE then
                    v.visit ← GRAY
                    v.distance ← u.distance + 1
                    v.parent ← u
                    v.parentport ← i
                    ENQUEUE(Q, v)
        if u.type = PROCESSOR then
            TRACEBACK(G, u)
        DEQUEUE(Q)
        u.visit ← BLACK

TRACEBACK(G, u)
    while u.distance ≠ 0
        v ← u.parent
        i ← u.parentport
        v.portusage[i] ← v.portusage[i] + 1
        u ← v
```

Fig. 1. The SP1 algorithm for route selection

clude the links in the routes in a balanced manner, it does not base the routing decisions on any measured or estimated network traffic. Therefore, it is a *non-adaptive* algorithm. Adaptive switches are known to perform better than non-adaptive switches in general with somewhat increased switch complexity [9, 7]. However, in the experiments we observed that the SP1 routing algorithm realizes many commonly used communication patterns without link conflicts for some network topologies.

In the SP1 multicomputer, network topologies are generally designed to be deadlock-free [2] with shortest path routes. For example, all the topologies used in the experiments reported in this paper are deadlock-free with shortest path routes. However, we have some experimental topologies that may cause deadlock cycles due to "turn-around routing" where a packet enters and leaves a switch from the same side. In such cases we eliminate the deadlock causing routes by putting routing restrictions on some switches while generating the routes.

3 An Experimental Routing Algorithm

We developed an experimental algorithm for adaptive route selection in SP1 like networks. We were motivated by the fact that although the SP1 switch chip was not designed for adaptive routing, software control of route selection and existence of multiple routes between any pair of nodes would allow us to make better routing decisions if estimates of the network traffic were available. We assumed that the network traffic is represented by a Node Interaction Graph (NIG). NIG is a directed graph whose vertices represent the processor nodes, while arcs represent interprocessor communication. NIG arcs may have weights indicating the amount of information transmitted from the source node to the destination node. The NIG model may appear unrealistic for general applications since it does not model the temporal interactions between the processor nodes. However, a large class of applications such as iterative solution of systems of equations that arise in numerical computing may be represented with NIGs (see [10, 11] for examples). When all vertices of an NIG have an in-degree and out-degree of 1, then it is a *permutation routing* problem. NIGs may be obtained in several ways, such as the users or compilers supplying NIGs based on the expected program behavior, or the operating system supplying NIGs based on the history of system workload. However, these are system level issues that we will not attempt to solve, but briefly discuss here. It is probably difficult for a user to generate NIGs unless communication patterns of the parallel program are well known or unless parallel program profiling tools are available. Compiler generated NIGs would be more convenient. It was suggested that NIGs may be derived during the data partitioning phase of a parallelizing compiler. The issue of multiple NIGs and their respective route tables may be dealt with easily. In SP1 route tables are stored in processor nodes memory. When the routing problem (NIG) changes, switching to a new route table is simply a matter of changing a pointer to the new route table base. Another issue is when to run the experimental routing algorithm described: in SP1, the operating system may perform a logical to physical node number mapping before loading programs to the nodes (described in Section 4.2). Therefore, to get good results the experimental routing algorithm must be executed using the physical node numbers obtained from the mapping.

In the experimental routing algorithm, we formulate the route selection problem as minimization of the cost function

$$\text{cost} = \sum_{\ell \in L} W_\ell^2 + K \sum_{s \in S} W_s^2 \tag{1}$$

where L is the set of all links, W_ℓ is the total flow through link ℓ, S is the set of all switches, and W_s is the total flow through switch s. The nonlinear cost function penalizes the links and switches with higher flow. For example, n messages each with a unit flow routed over one link will contribute n^2 units to the cost, whereas the n messages routed over n different links will contribute n units to the cost. We used an exponent of 2 merely for convenience. An exponent of 3 made no appreciable difference in results. $K \geq 0$ is the weight of the total switch penalty and it is a hardware dependent constant. $K \neq 0$ is used to minimize switch sharing. In some switch designs, messages sharing the switch resources such as a central queue may impact the performance and this may be taken into account in the cost function by a nonzero constant K that is derived empirically or by analysis. A cost function similar to Eq. 1 was used in [12] for routing in networks with virtual cut-through capability. However, Eq. 1 differs from that of [12] such that the second term due to switch sharing does not exist in [12]. Furthermore, the distance metric that we use in our algorithm is based on the number of network hops, whereas in [12] it is based on the link utilization.

A brief sketch of the adaptive algorithm is given in Fig. 2. The objective is to minimize the cost. For each communication arc $(s, d, f) \in \mathrm{E(NIG)}$ an initial route is selected, where f is the required amount of flow from node s to node d. The total cost is calculated after the initial selection of routes. Then, sequentially for each arc $(s, d, f) \in \mathrm{E(NIG)}$, the previously selected route is ripped up and a new route with smaller incremental cost is selected from the set of routes $R[s][d]$. The procedure is repeated iteratively until the cost converges to a local minimum. The algorithm is guaranteed to converge because the cost is monotonically non-increasing. If the cost from previous iteration does not change, the algorithm does not terminate immediately but allows a different set of routes with the same cost to be tried a bounded number of times ($NTRY = 2$ in this case) in anticipation of further cost reduction in the next iteration. For the topologies we used, the route set $R[s][d]$ consists of all deadlock-free shortest-path routes from node s to node d. However, in richer topologies a restricted subset of the routes between nodes s, d may also be considered, because the number of routes may get quite large increasing the execution time.

4 Results and Conclusions

We have implemented the experimental route selection algorithm and compared its performance to the SP1 routing algorithm using a set of communication work-

```
ROUTER(NIG, G)
    Let R be the set of all routes, where R[i][j]
           is a set of all possible routes from node i to j
    for each arc e = (src, dst, flow) ∈ E(NIG)
           Select an initial route r ∈ R[e.src][e.dst]
           Add e.flow to the links and switches on the path of route r
           Update cost
    previous_cost ← ∞
    n_trials ← NTRY ← 2
           (to try the same cost a number of times)
    while previous_cost > cost OR n_trials ≠ 0
           previous_cost ← cost
           for each arc e = (src, dst, flow) ∈ E(NIG)
                  cost ← ROUTE_ONE_EDGE(e, G, R)
                  if cost = previous_cost then
                         n_trials ← n_trials − 1
                  else
                         n_trials ← NTRY

ROUTE_ONE_EDGE(e, G, R)
    Rip up previously selected route for e and update cost
    Find a route r ∈ R[e.src][e.dst] with the smallest incremental cost.
    If there are multiple such routes, then select one randomly
    Update G and cost
```

Fig. 2. The adaptive algorithm for route selection

loads. Results given in Tables 1 through 3 show how well the two algorithms deal with network congestion. Note that results are based on simulations. They are not based on actual measurements on SP1. The FLOW metric in the tables denotes the maximally loaded link in the network for given workload and network size. Figs. 7-9 summarize the relative performance of the experimental routing algorithm over the SP1 routing algorithm based on the FLOW metric, as described in more detail in Section 4.2.

4.1 Workloads and Methods

In the simulations, we used standard network topologies available from IBM for 16, 32, 64 node systems. For 128, 256, and 512 node networks we used topologies shown in Figs. 5 and 6. The 256 node topology has all the nodes connected to the left hand side of the network with the right hand side ports remaining unconnected. The 512 node topology is constructed from two 256 node networks shown in Fig. 6 whose right hand sides are interconnected with straight wires. Not shown in the figures is the 256-A topology which consists of 8 second stage boards instead of the 16 used in Fig. 6.

We used a number of communication workloads (NIGs): in the RANDOM-F workload each node i sends a unit size message to a randomly selected node j. RANDOM-V is similar except that message sizes randomly vary between 1 and 10 units. DOLOOP refers to a commonly used communication pattern in parallel programs coded in Fortran. Each node executes

DO $I = 1, N-1$
  each node $J = 0 \ldots N-1$ sends message to node $(I+J)(mod N)$
  where N is the number of processors
CONTINUE

Note that each iteration of the loop corresponds to one NIG graph. EXOR refers to a communication pattern that provides conflict free routing in hypercubes as shown in [13]. It is similar to the DOLOOP, except that the order of exchange is different

for $i = 1$ to $N-1$
  each node $j = 0 \ldots N-1$ sends message to node i EXOR j
  where N is the number of processors

NCUBE refers to a commonly used communication pattern in divide and conquer type algorithms. Given 2^n processor nodes, each node sends to n other nodes

for $i = 0$ to $n-1$
  each node $(j_{n-1} \ldots j_i \ldots j_0)$ sends to node $(j_{n-1} \ldots \overline{j_i} \ldots j_0)$

where $(j_{n-1} \ldots j_i \ldots j_0)$ is the binary representation of a processor's node number and $(j_{n-1} \ldots \overline{j_i} \ldots j_0)$ is the i-th bit complemented. We derived the remaining workload from BCSSTK9, BCSPWR10, BLCKHOLE, and JAGMESH9 matrices of Harwell-Boeing sparse matrix collection. We mapped task graphs obtained from the sparse matrices to processor graphs using Kernighan-Lin heuristic to minimize communication [14]. We then assumed that the processors communicate using the DOLOOP algorithm. The resulting NIGs resemble the NIGs for DOLOOP with the exception that the arcs have variable weights. We report here only the results for BCSSTK9. Others were similar.

In the tables, the COST column refers to the minimum cost obtained by the algorithms as given by Eq. 1. We set the constant $K = 0$ in the simulations since switch sharing does not incur any penalty in the SP1 switch. As a performance metric we also included the maximally loaded link in the network given in the FLOW column. Note that smaller cost does not necessarily mean smaller maximum link flow. However, in practice we have not observed a case of maximum link flow increasing with decreasing cost.

4.2 Results

The main result of the paper regarding the SP1 algorithm is shown in Table 1 which indicates that the SP1 routing algorithm generates conflict free routes for 16, 32, and 512 node topologies for DOLOOP, EXOR, and NCUBE workloads. FLOW columns show that the maximum link load is 1.0 indicating conflict free routing. The experimental routing algorithm was most effective with the RANDOM workloads; the maximum link load was smaller by a factor of 2 to 3 compared to the SP1 routing algorithm. For the BCSSTK9 workload the difference between the two algorithms was negligible, most probably due to the fact that the NIGs were sparse and used the DOLOOP algorithm. Thus messages rarely shared any links or switches. The results in Tbl. 1 are summarized in Fig. 7 by dividing the SP1 algorithm max. link load by the experimental algorithm max. link load averaged over the six workloads. A ratio of 1.0 or greater in Fig. 7 indicates that the experimental algorithm performs better. Going back to Tbl. 1, we see that for the 16 and 32 node DOLOOP cases the experimental algorithm performed worse than the SP1 routing algorithm suggesting a weakness of the experimental algorithm: since it is a local minimization heuristic, the quality of results depend very much on the initial selection of the routes. To fix this problem we modified the experimental routing algorithm such that instead of selecting the initial routes randomly in Line 3 of Fig. 2, we selected the routes generated by the SP1 routing algorithm as the initial routes. Results of this experiment are reported in Tbl. 2 which indicate that the experimental routing algorithm always performs better than or equal to the SP1 routing algorithm for individual cases. However, the averages did not change significantly except for the 256-C and 512 node cases as shown in Fig. 8.

We performed a third set of simulations reported in Table 3 and Fig. 9 to test the effects logical to physical node mapping. In SP1, each user is presented with a logical sequence of node numbers from 0 to N-1. The logical node number observed by a user program is not necessarily equal to the physical node number of the underlying node. A logical to physical node number mapping is performed by the system. This is necessary because some nodes may already have been allocated to other users, and some nodes may be down, and therefore cannot be allocated. To test the effects of mapping on routing we randomly interchanged the node numbers. Although random mapping will not happen in practice, we tried it as a worse case for the SP1 routing algorithm and to see how well the experimental algorithm adapts to the mapping. Results show that the DOLOOP, EXOR, and NCUBE communication patterns could not use the conflict-free routes generated with the SP1 algorithm anymore and the experimental routing algorithm performed better.

4.3 Conclusions

Simplicity is the main advantage of the SP1 routing algorithm. It selects the routes in a balanced manner and in a topology independent fashion which is useful in large networks possibly with broken links and switches. Among the workloads that we used, the experimental algorithm improved the performance over the SP1 algorithm most significantly when node interactions were random. In general the experimental algorithm will be useful for networks in which the route selection is under software control such as in the SP1 network and when the problem has known routing loads in advance. Compilers appear to be good candidates for producing that information. For future work it would be interesting to try the experimental algorithm not only by simulations but also by actual implementation on SP1 for real problems. Also interesting to see is how well the experimental algorithm performs compared to a network with adaptive switch hardware.

Acknowledgements

We wish to thank Craig Stunkel of IBM Research for his valuable discussions. We also would like to thank D.G. Shea and C.J. Tan of IBM Research for their support, and finally the anonymous reviewers for their suggestions.

References

1. C. B. Stunkel, D. G. Shea, B. Abali, M. M. Denneau, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker, "Architecture and Implementation of Vulcan," in *Proc. Int. Parallel Processing Symp.*, pp. 268–274, April 1994. An extended version is also available as Research Report RC19492 from the IBM T.J. Watson Research Center.
2. W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, May 1987.
3. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A new computer communications switching technique," *Computer Networks*, vol. 3, pp. 267–286, September 1979.
4. A. S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.
5. D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, vol. C-24, pp. 1145–1155, December 1975.
6. M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, vol. C-26, pp. 458–473, May 1977.
7. S. A. Felperin, L. Gravano, G. D. Pifarre, and J. L. C. Sanz, "Routing Techniques for Massively Parallel Communication," *Proceedings of the IEEE*, vol. 79, pp. 488–503, April 1991.
8. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* NY: McGraw-Hill, 1990.
9. R. V. Boppana and S. Chalasani, "A Comparison of Adaptive Wormhole Routing Algorithms," in *Proceedings of the 20th Ann. Int. Symp. on Computer Architecture*, pp. 351–360, May 1993.
10. T. Bultan and C. Aykanat, "A New Mapping Heuristic Based on Mean Field Annealing," *J. Parallel and Distributed Comput.*, vol. 16, pp. 292–305, 1992.
11. C. Aykanat, F. Ozguner, F. Ercal, and P. Sadayappan, "Iterative Algorithms for Solution of Large Sparse Systems of Linear Equations on Hypercubes," *IEEE Trans. Comput*, vol. 37, no. 12, pp. 1554–1567, 1988.
12. D. D. Kandlur and K. G. Shin, "Traffic Routing for Multicomputer Networks with Virtual Cut-Through Capability," *IEEE Transactions on Computers*, vol. 41, pp. 1257–1270, October 1992.
13. B. Abali, F. Ozguner, and A. Bataineh, "Balanced Parallel Sort on Hypercube Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 572–581, May 1993.
14. B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Tech. J.*, vol. 49, pp. 291–307, 1970.

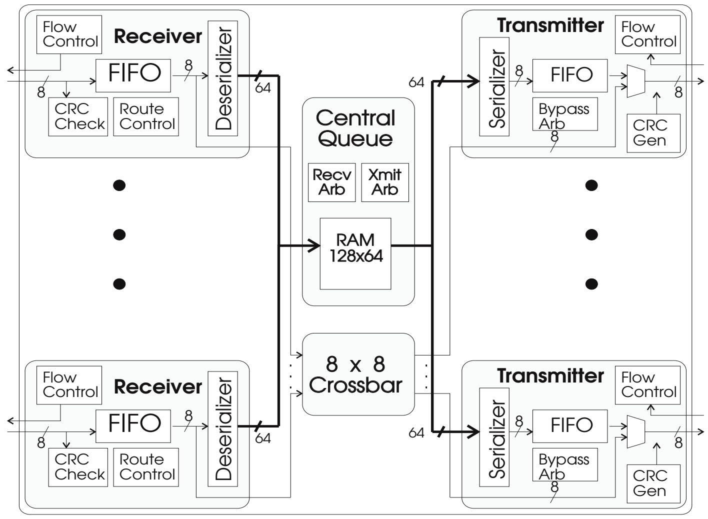

Fig. 3. The Switch Chip Block Diagram.

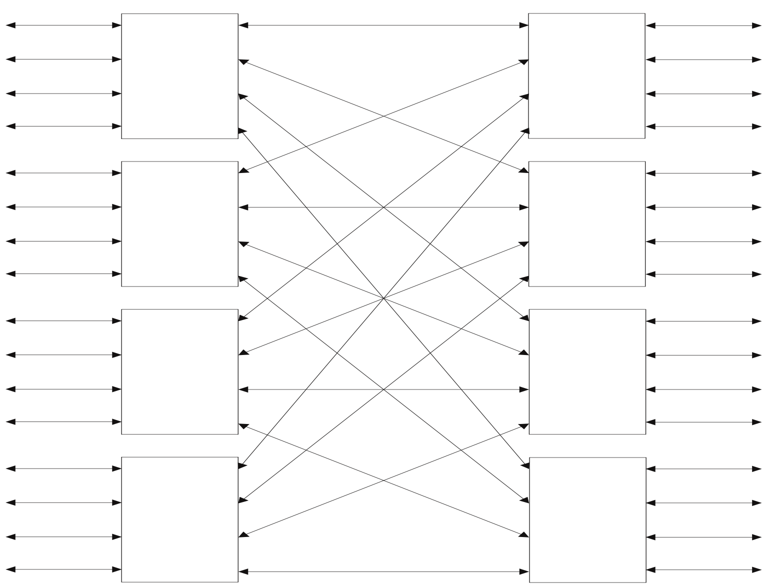

Fig. 4. The Switch Board consisting of 8 Switch Chips

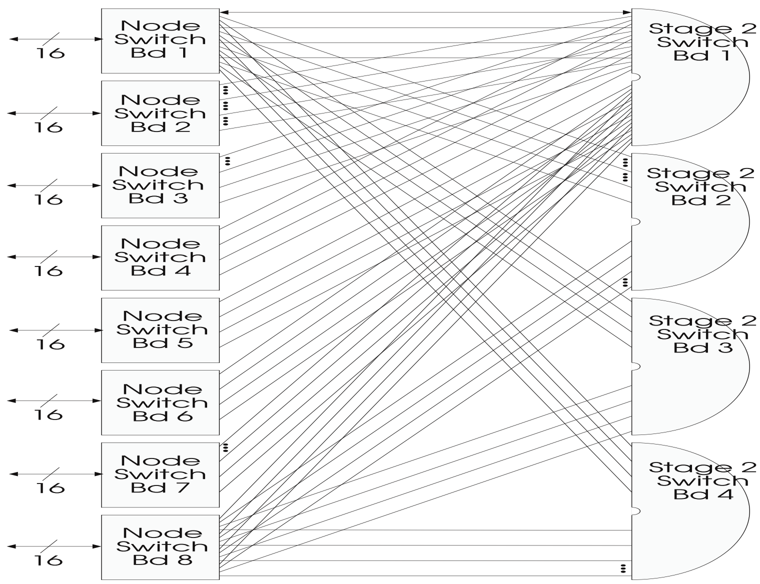

Fig. 5. A 128 node network consisting of 8 first stage and 4 second stage switch boards.

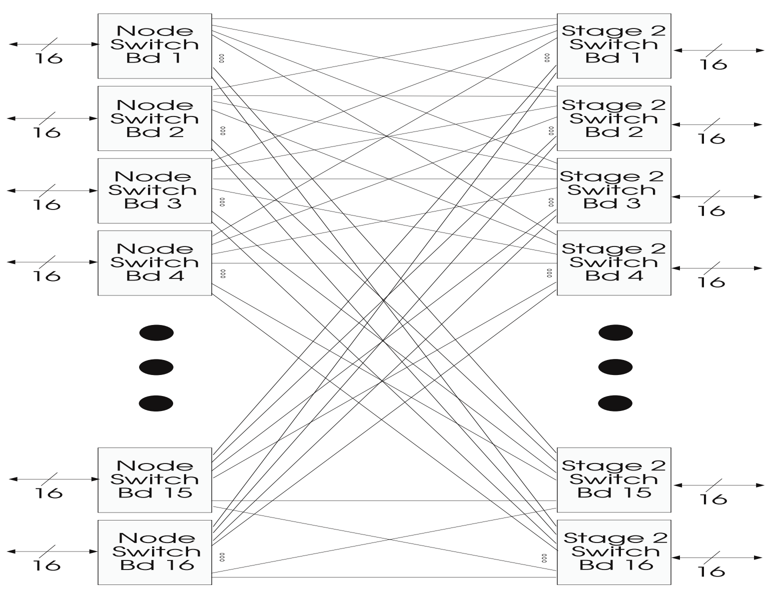

Fig. 6. A 256 node network consisting of 16 first stage and 16 second stage switch boards.

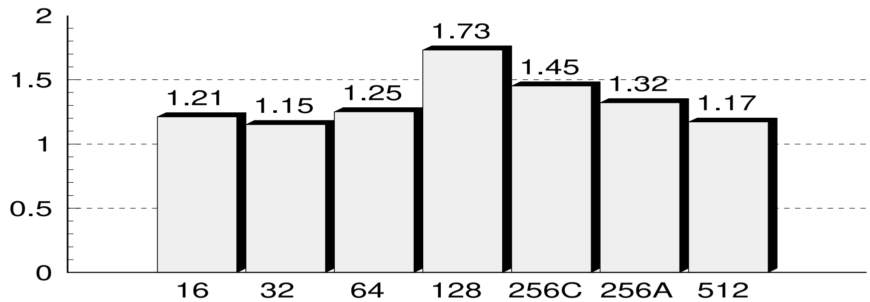

Fig. 7. Ratio of SP1 FLOW over EXP FLOW averaged over the six workloads given in Table 1

# NODES		16 NODES		32 NODES		64 NODES		128 NODES	
WORKLOAD		FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
RANDOM-F	SP1	2.10	30.4	2.50	86.0	3.20	250.2	3.50	712.0
RANDOM-F	EXP	1.20	25.2	1.70	73.2	2.00	198.0	2.00	571.6
RANDOM-V	SP1	15.30	1169.8	18.50	3348.7	22.00	9469.4	27.50	26914.1
RANDOM-V	EXP	9.30	947.8	10.20	2836.3	13.00	7384.2	10.90	21566.7
DOLOOP	SP1	1.00	25.6	1.00	74.3	3.19	247.7	3.87	863.2
DOLOOP	EXP	1.13	26.1	1.55	77.5	2.73	244.6	1.93	575.4
EXOR	SP1	1.00	32.0	1.00	82.3	3.40	307.2	2.55	949.7
EXOR	EXP	1.00	32.0	1.00	82.3	3.40	302.9	1.52	571.9
NCUBE	SP1	1.00	32.0	1.00	74.7	2.50	236.0	1.60	588.8
NCUBE	EXP	1.00	32.0	1.00	74.7	2.50	234.0	1.20	442.4
BCSSTK9	SP1	61.71	21521.7	36.31	11841.4	28.67	6225.3	13.37	2183.5
BCSSTK9	EXP	61.71	21521.7	36.31	11841.4	26.88	6077.8	12.15	2080.8
# NODES		256 NODES-C		256 NODES-A		512 NODES			
WORKLOAD		FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
RANDOM-F	SP1	4.50	1874.8	4.20	1741.0	4.80	4348.8		
RANDOM-F	EXP	2.00	1503.8	2.00	1374.0	2.00	3504.2		
RANDOM-V	SP1	33.80	67670.2	30.00	62225.3	32.30	158898.9		
RANDOM-V	EXP	14.00	53969.6	16.20	47799.3	17.20	126871.5		
DOLOOP	SP1	1.96	1421.3	3.03	1521.3	1.00	3166.2		
DOLOOP	EXP	1.95	1503.8	3.07	1633.7	2.01	3498.2		
EXOR	SP1	1.76	1438.5	3.29	1779.8	1.00	3184.9		
EXOR	EXP	1.76	1406.0	3.29	1804.2	1.88	3222.2		
NCUBE	SP1	1.33	1045.3	2.00	1194.7	1.00	2267.4		
NCUBE	EXP	1.33	1032.0	2.00	1205.3	1.43	2282.3		
BCSSTK9	SP1	5.00	584.2	4.96	561.2	2.38	219.5		
BCSSTK9	EXP	4.96	583.7	4.96	561.2	2.38	219.6		

Table 1. Comparison of the SP1 routing algorithm and the experimental routing algorithm which uses random initial routes.

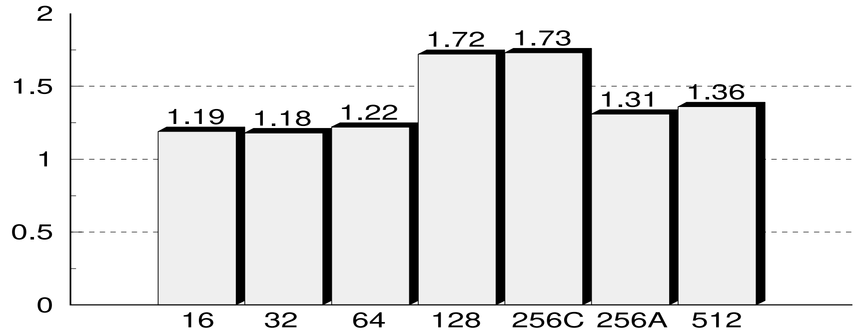

Fig. 8. Ratio of SP1 FLOW over EXP FLOW averaged over the six workloads given in Table 2

# NODES	16 NODES		32 NODES		64 NODES		128 NODES	
WORKLOAD	FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
RANDOM-F SP1	2.10	30.4	2.50	86.0	3.20	250.2	3.50	712.0
RANDOM-F EXP	1.30	26.2	1.90	75.4	2.00	199.0	2.00	572.6
RANDOM-V SP1	15.30	1169.8	18.50	3348.7	22.00	9469.4	27.50	26914.1
RANDOM-V EXP	9.90	957.0	10.50	2839.7	14.00	7437.0	11.70	21584.7
DOLOOP SP1	1.00	25.6	1.00	74.3	3.19	247.7	3.87	863.2
DOLOOP EXP	1.00	25.6	1.00	74.3	2.92	241.5	1.83	552.6
EXOR SP1	1.00	32.0	1.00	82.3	3.40	307.2	2.55	949.7
EXOR EXP	1.00	32.0	1.00	82.3	3.40	302.9	1.52	560.5
NCUBE SP1	1.00	32.0	1.00	74.7	2.50	236.0	1.60	588.8
NCUBE EXP	1.00	32.0	1.00	74.7	2.50	234.0	1.20	438.0
BCSSTK9 SP1	61.71	21521.7	36.31	11841.4	28.67	6225.3	13.37	2183.5
BCSSTK9 EXP	61.71	21521.7	36.31	11841.4	26.88	6077.8	12.15	2080.8

# NODES	256 NODES-C		256 NODES-A		512 NODES			
WORKLOAD	FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
RANDOM-F SP1	4.50	1874.8	4.20	1741.0	4.80	4348.8		
RANDOM-F EXP	2.00	1503.8	2.00	1376.8	2.10	3512.6		
RANDOM-V SP1	33.80	67670.2	30.00	62225.3	32.30	158898.9		
RANDOM-V EXP	15.00	54114.4	17.00	47788.7	17.10	126901.1		
DOLOOP SP1	1.96	1421.3	3.03	1521.3	1.00	3166.2		
DOLOOP EXP	1.09	1373.8	3.02	1521.3	1.00	3166.2		
EXOR SP1	1.76	1438.5	3.29	1779.8	1.00	3184.9		
EXOR EXP	1.00	1389.7	3.29	1779.8	1.00	3184.9		
NCUBE SP1	1.33	1045.3	2.00	1194.7	1.00	2267.4		
NCUBE EXP	1.00	1024.0	2.00	1194.7	1.00	2267.4		
BCSSTK9 SP1	5.00	584.2	4.96	561.2	2.38	219.5		
BCSSTK9 EXP	4.96	583.7	4.96	561.2	2.38	219.5		

Table 2. Comparison of the SP1 routing algorithm and the experimental routing algorithm which uses the initial routes selected by the SP1 routing algorithm.

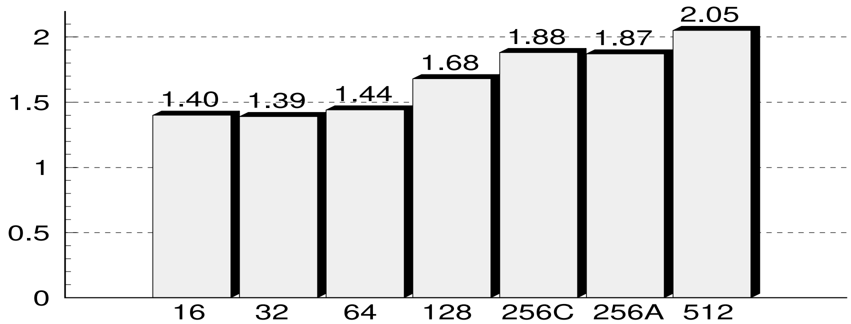

Fig. 9. Ratio of SP1 FLOW over EXP FLOW averaged over the four workloads given in Table 3

# NODES		16 NODES		32 NODES		64 NODES		128 NODES	
WORKLOAD		FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
DOLOOP	SP1	2.00	30.4	2.74	92.5	3.21	240.3	3.74	712.3
DOLOOP	EXP	1.13	25.9	1.87	78.1	2.00	191.5	2.00	565.0
EXOR	SP1	2.13	30.9	2.81	92.0	3.13	240.2	3.63	714.1
EXOR	EXP	1.67	27.3	1.94	80.1	2.00	193.2	2.00	566.5
NCUBE	SP1	2.25	30.5	3.00	92.0	3.00	253.3	3.71	728.6
NCUBE	EXP	1.50	26.0	2.00	78.0	2.00	201.3	2.00	572.6
BCSSTK9	SP1	82.08	41564.3	51.46	30824.4	30.25	15574.1	14.49	6979.4
BCSSTK9	EXP	76.83	40468.7	44.21	28000.8	28.02	13817.9	12.38	6158.5
# NODES		256 NODES-C		256 NODES-A		512 NODES			
WORKLOAD		FLOW	COST	FLOW	COST	FLOW	COST	FLOW	COST
DOLOOP	SP1	4.26	1850.6	4.21	1705.7	4.76	4319.3		
DOLOOP	EXP	2.00	1488.5	2.02	1359.2	2.03	3487.1		
EXOR	SP1	4.24	1850.0	4.24	1706.6	4.73	4311.7		
EXOR	EXP	2.01	1493.1	2.01	1362.1	2.03	3495.4		
NCUBE	SP1	4.25	1862.5	4.25	1719.0	4.78	4322.0		
NCUBE	EXP	2.00	1496.2	2.00	1375.5	2.00	3487.8		
BCSSTK9	SP1	5.87	2791.8	5.88	2477.2	2.69	646.2		
BCSSTK9	EXP	5.10	2396.8	5.10	2075.1	2.39	564.0		

Table 3. Comparison of the SP1 routing algorithm and the experimental routing algorithm with random mapping of tasks onto processors