Static Allocation of Multirail Networks*

Salvador Coll, Eitan Frachtenberg, Fabrizio Petrini,
Adolfy Hoisie and Leonid Gurvits

CCS-3 Modeling, Algorithms, & Informatics Group
Computer & Computational Sciences Division
Los Alamos National Laboratory
{scoll,eitanf,fabrizio,hoisie,gurvits}@lanl.gov

Technical Report

Abstract

Using multiple independent networks (also known as rails) is an emerging technique to overcome bandwidth limitations and enhance fault-tolerance of current high-performance clusters. This report presents the limitations and performance of static rail-allocation approaches, where each rail is pre-assigned a direction for communication. An analytical lower bound on the number of networks required for rail allocation is shown. We present an extensive experimental comparison of the behavior of various allocation schemes in terms of bandwidth and latency, compared to static rail allocation. We also compare the ability of static and dynamic rail-allocation mechanism to stripe messages over multiple rails. Scalability issues of static and dynamic rail allocation are also compared. We find that not only static rail allocation necessarily consumes many resources, it also performs poorly compared to dynamic rail allocation schemes, in all the tested aspects.

1 Introduction

System interconnection networks have become a critical component of the computing technology with a direct impact on the design, architecture, and use of high-performance parallel computers. Indeed, not only the sheer computational speed distinguishes high-performance computers from desktop systems, but the efficient integration of the computing nodes into tightly coupled multiprocessor systems. Network adapters, switches, device driver software and communication libraries are increasingly becoming performance critical components in modern supercomputers.

One approach to build large scale supercomputers, with as many as thousands of processors, is to use shared memory multiprocessors (SMPs) as building blocks. In such

*The work was supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36

machines it is very important to keep the ratio between computing power and communication capability properly balanced.

A natural solution to the problems of limited bandwidth availability in network connections and of fault tolerance is the use multiple parallel networks, or "rails". To the best of our knowledge, very little attention has been given so far in the literature in studying communication protocols, performance characteristics, fault-tolerance, implementation of system software and communication libraries for multiple rails.

Besides from being a challenging scientific endeavor, the analysis of multi-railed networks has direct practical implications too. Los Alamos National Laboratory and Compaq are currently developing an extreme-scale, multi-railed cluster of SMPs, the 30Tops ASCI Q machine[1]The Q-machine is based on the Quadrics network (QsNet)[2], which consists of two building blocks, a 64bit/66MHz PCI card with a programmable network interface called Elan [7] and a low-latency high-bandwidth communication switch called Elite [8]. Elites can be interconnected in a fat-tree topology [3]. A recent performance evaluation of the QsNet, shows that the network performance is seriously limited by the PCI bus [5]. In fact, the network can deliver almost 340 MB/sec at user-level (400MB/sec of raw bandwidth), but the PCI implementation can sustain only 300 MB/sec, using the most efficient PCI chipset on the market. A further performance degradation in the presence of bidirectional traffic, limits the aggregate communication bandwidth to 80% of the unidirectional bandwidth on most PCI chipsets (Intel 840, Serverworks He and LE, Compaq Wildfire). Though the next generation of the PCI interface, called PCI-X, will double the nominal performance, the new generation of QsNet will also double its performance, so this design issue won't disappear. One important constraint that impacts all communication strategies proposed is that bidirectional traffic cannot be efficiently supported by the I/O interfaces.

This technical report complements our study in [1] of dynamic rail allocation schemes. In this report we present the basic properties of a multi-railed network, and analyze the static rail allocation approach to communication over multiple rails. This approach allocates each network interface to unidirectional traffic. That is, each network interface can either send or receive messages, and its allocation is determined at boot/initialization time. Static allocation poses the problem of reachability between nodes: we want to have a direct path in the network between any possible pair of nodes. The use of intermediate nodes could seriously degrade the latency achieved by zero-copy, user-level communication protocols, a key feature of most high-performance networks.

We compare the "optimal" static approach (see next section) with the basic, local-dynamic and dynamic approaches in [1]. In a nutshell, the basic approach uses rails in round-robin fashion, using one rail for every send while disregarding if it is busy; the dynamic approach reserves rails on both sides before sending a message, and can stripe messages over several rails.

The experimental results, obtained using a circuit-level simulator of the network and network interface, explore the performance of the static rail allocation and compare it the the performance of the dynamic methods under several traffic loads and message sizes. These results shed new light into the benefit of using multiple network rails and expose several trade-offs in the design of the allocation algorithms.

[1]http://www5.compaq.com/alphaserver/news/supercomputer_0822.html

[2]http://www.quadrics.com

The rest of this report is organized as follows. Section 2 proposes two static allocation algorithms and presents a formal analysis on the rail requirement limits. One of the proposed algorithms is optimal in terms of the required number of rails. The details of the experimental evaluation carried out and the results are described in Section3. Finally, we conclude in Section 4

2 Static Allocation

In this section the static allocation of network interfaces, in which each rail is exclusively a transmitter or receiver, is analyzed. We obtain theoretically the best allocation pattern and the appropriate algorithm to generate it. We will use the terms network interface and rail interchangeably throughout this paper .

2.1 Theoretical bound

The answer we are seeking in this section is: what is the maximum number of processing nodes that we can use for a given number of rails, under the following constraints:

- Each node can only transmit or receive on a given rail but not both. This ensures unidirectional access to the I/O bus.
- Full connectivity among the nodes, i.e. each node can transmit to every other node without passing through intermediate nodes.
- Rails are independent: messages cannot pass from one rail to another.

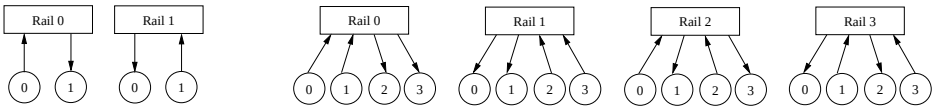
We can describe a static allocation using a binary matrix where columns represent nodes and rows represent rails, so that a value of '1' in the A_{ij} entry means that node *j* transmits on rail *i*. Figure 1 depicts static allocations examples and their equivalent allocation matrices. In the example shown in Figure 1(a), rail 0 can be used for sending by node 0 and receiving by node 1. Since the allocation is static, one more rail is required to allow communication from node 1 to node 0. As can be seen two rails are sufficient to ensure full connectivity between two nodes. When considering four nodes, at least four rails are required to ensure full connectivity. Figure 1(b) shows one possible allocation.

One simple bound of $n \leq 2^{\frac{r}{2}}$, where *n* is the number of nodes and *r* is the number of rails can be obtained with the static allocation described in Algorithm 1. While this allocation is simple, and clearly satisfies the constraints, it is not optimal. The optimality is described by the following theorem:

Theorem 1. *Given r network rails the number of nodes n that can be statically allocated to these rails with unidirectional communication in the network interface card (NIC) and full node connectivity cannot exceed*

$$n \leq \begin{pmatrix} r \\ \lfloor \frac{r}{2} \rfloor \end{pmatrix} \tag{1}$$

Proof. Each node can use any given rail for either transmitting or receiving, but not both (unidirectional requirement). Let a binary vector represent the static allocation of nodes on a rail: the vector's *i*th entry is 0 if the *i*th node receives on this rail and 1 if it



(a) Two rail allocation for two nodes

(b) Four rail allocation for four nodes

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Figure 1: Simple static allocation examples for 2 and 4 nodes. Rectangles denote networks (rails); circles represent nodes and arrows denote the allocation of each rail to each node as either transmitting or receiving.

Algorithm 1 : Static rail allocation with $2\log_2 n$ rails.

```
procedure log_rail_alloc
begin
    for i = 0 to log2 n − 1 do
    begin
        allocate nodes on rail in consecutive groups of 2^i, alternating
        between transmitters and receivers, starting with transmitters.
    end
    for i = 0 to log2 n − 1 do
    begin
        allocate nodes on rail in consecutive groups of 2^i, alternating
        between transmitters and receivers, starting with receivers.
    end
end
```

transmits on it. We can represent the static allocation of the entire system as a binary matrix *A* with *r* rows, each representing one rail, and *n* columns, each representing one node. Let A_{ij} denote the value at row *i* and column *j* of *A*, that is, the role allocated to the *j*th node on the *i*th rail. The problem can thus be formalized as determining the maximum number of columns *n* of a binary matrix with *r* rows for which the following property holds:

$$\forall\, x, y \in \{1..n\},\, x \neq y:\ \exists\, \rho \in \{1..r\}\ \ s.t.\ A_{\rho x} = 0,\ A_{\rho y} = 1 \tag{2}$$

For each matrix column *j* let S_j be the set of indexes *i* for which $A_{ij} = 1$:
$S_j = \{1 \leq i \leq r \mid A_{ij} = 1\}$. Note that the property (2) of a matrix *A* is equivalent to the following property:

$$\forall\, x, y \in \{1..n\},\, x \neq y:\ S_x \nsubseteq S_y \tag{3}$$

The equivalence stems from the fact that if (3) doesn't hold, i.e.

$$\exists\, x, y \in \{1..n\},\, x \neq y\ s.t\ S_x \subset S_y$$

then for every row $\rho \in \{1..r\}$ for which $A_{\rho x} = 1$ we havealso $A_{\rho y} = 1$ so (2) cannot hold. In the other direction, if (3) holds then for every two columns $x, y \in \{1..n\}$, $x \neq y$ there would have to be at least one row $\rho \in \{1..r\}$ for which $A_{\rho x} = 0$, $A_{\rho y} = 1$, or else either $S_x \subset S_y$ or $S_y \subset S_x$. The maximum number of columns *n* for a matrix *A* with the property (3) is given by Sperner's lemma to be (1). A short proof of this lemma can be found in [4]. □

2.2 Allocation algorithm

We propose an algorithm to allocate *r* rails to *n* nodes for any given *r* and *n* that satisfies (1). This algorithm is simple to implement and is optimal in the sense that it can allocate rails for all the nodes even when the bound is tight. The procedure is to define *n* binary vectors (each representing the rail transmit/receive allocation for a single node), each having exactly $\left[\frac{r}{2}\right]$ 1's in them. The number of distinct vectors with this property is

$$\left(\begin{array}{c} r \\ \left[\frac{r}{2}\right] \end{array} \right)$$

so there is a sufficient number of vectors to allocate for *n* nodes. Also note that any two different vectors containing the same number of 1's satisfy condition (3), so by inference these vectors satisfy the requirement (2). Any enumeration that produces the different vectors can provide this vectors. For example, strings can be enumerated by lexicographic order (for *r=4* we could have 0011, 0101, 0110, 1001, 1010, 1100). Another simple procedure to define these vectors is described in Algorithm 2.

2.3 Examples

Figure 2 shows the relationship between the number of nodes and the number of rails required to support them according to our requirements, using the two allocation algorithms described. An example allocation using Algorithm1 is depicted in Figure 3. We note that a maximum of 8 nodes can be allocated using 6 rails. Figure 4 is an example of an optimal allocation matrix created by Algorithm 2 for 20 nodes on 6 rails (20 nodes is the maximum for 6 rails).

Algorithm 2 : Optimal static rail allocation.

```
{ build_rail_vectors is a recursive procedure that runs until n binary
  vectors of length r are output (n is the number of nodes and r is
  the number of rails), each representing an allocation of a single
  node. The procedure tries to allocate a 1 and then 0 for each vector
  location, and backtracks whenever a vector is completed. It should
  be first called from outside with the following parameters:
  build_rail_vectors (empty_vector, r, int(r/2))
}
Procedure build_rail_vectors
Input: vector being built (current_vector),
       rails left to allocate (rails_left),
       ones left for this vector (ones_left)
begin
  if n vectors were output then return  { Ending condition met -
                                          allocated for all nodes }

  if rails_left <= 0 then  { No. more rails to allocate means that - }
     output current_vector { the current vector (node) is completed. }
  else
  begin                          { Still have rails to allocate }
   if ones_left > 0 then    { Try to allocate a 1 if any left }
      build_rail_vectors (current_vector appended with 1,
                          rails_left - 1,
                          ones_left - 1)
   if (rails_left - ones_left) > 0 then    { Try to allocate a 0 -
                                             if any left }
       build_rail_vectors (current_vector appended with 0,
                           rails_left - 1,
                           ones_left)
  end
end
```

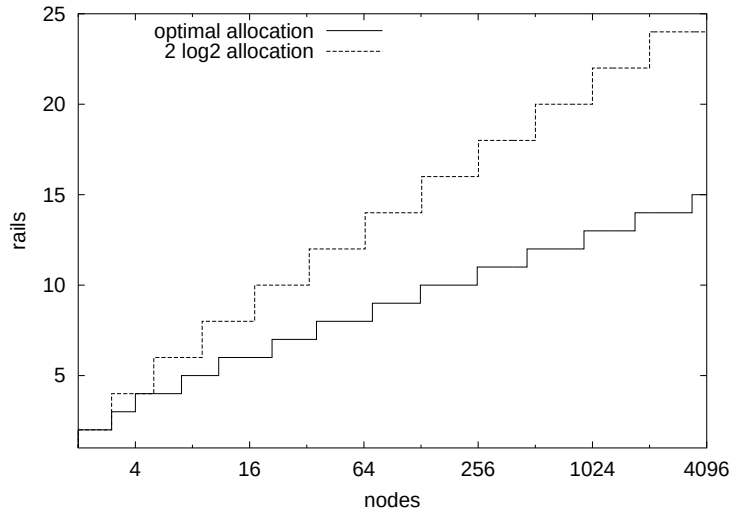


Figure 2: Required rails as a function of the number of nodes for both static allocation algorithms

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 3: Example allocation for 6 rails and 8 nodes using Algorithm 1.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Figure 4: Optimal allocation matrix for 6 rails and 20 nodes created using Algorithm 2.

3 Experimental Framework and Results

3.1 Simulation Model

In the experimental evaluation we focus our attention on a family of fat-tree interconnection networks, 32 SMP nodes, each with four processors. Using the optimal allocation described in 2.2, it can be seen that a minimum of seven rails is required to statically allocate 32 nodes with the optimal allocation. The simulation model tries to capture the most important characteristics of the QsNet at the granularity of the clock cycle. The simulator models wormhole flow-control, with two virtual channels on each physical channel. The input buffers on each virtual channel can contain up to 128 flits [2], each consisting of two bytes. A flit can be transmitted over a physical channel in a single clock cycle, while a packet can be routed through an Elite switch in six clock cycles.

The simulator also models a thread processor in the NIC, which can process incoming control and data packets and can send a reply in few hundreds of clock cycles. Another important characteristic is the unidirectionality of the I/O bus, which can transmit data in one direction. We also assume that the bus bandwidth is equalized with the external network bandwidth (an optimistic set of assumptions, given the current state of the art).

This model is evaluated in the SMART (Simulator of Multiprocessor ARchitectures and Topologies) environment [6]. Implemented in C++, SMART is an object-oriented discrete-event simulation tool for evaluating parallel architectures and high performance interconnection networks.

3.2 Communication Patterns

In our model each process generates packets independently, using three random variables: (1) The message size, which is exponentially distributed with a given mean value. The average message size we used is 32KB;[3] (2) the inter-arrival time is also exponentially distributed around a given mean value; and the destinations are randomly chosen with equal probability between the processes.

3.3 Metrics

The performance of an interconnection network under dynamic load is usually assessed by two quantitative parameters, the*accepted bandwidth* or *throughput* and the*latency*. Accepted bandwidth is defined as the sustained data delivery rate given some offered bandwidth at the network input. Two important characteristics are the saturation point and the sustained rate after saturation. Saturation is defined as the minimum offered bandwidth where the accepted bandwidth is lower than the global packet creation rate at the source nodes. It is worth noting that, before saturation, offered and accepted bandwidth are the same. The behavior above saturation is important because the network and/or the allocation algorithms can become unstable, leading to a sharp performance degradation. We usually expect the accepted bandwidth to remain stable after saturation, for example in the presence of bursty applications that require peak performance for a short period of time.

The experimental results of each traffic are presented using two graphs, one to display the accepted bandwidth and the other to display the network latency. In both graphs the x-axis corresponds to the offered bandwidth normalized with the unidirectional bandwidth of the links connecting the processing nodes to the network switches. This makes the analysis independent from the link bandwidth and the flit size.

We report the latency in cycles rather than absolute time, in order to make our analysis insensitive to technological changes. Given that the I/O bus in the network interface can only allow unidirectional traffic, the maximum achievable throughput under uniform traffic is only 50% of the nominal injection bandwidth. The intuition behind this limitation is the following: for example, let's consider a cluster with only two nodes and single network rail. Under uniform traffic only one SMP can send to another at any given time, due to the unidirectionality constraint in the endpoints.

3.4 Experimental results

Figure 6 depicts the bandwidth and latency results for the different algorithms. Only the optimal static allocation is compared because the resources required for simulating enough rails for the $2logn$ allocation were prohibitive. We expect the optimal allocation to perform better than the $2logn$ allocation, so these results serve to demonstrate the weaknesses of the static allocation.

One trend arising from these data is that the dynamic algorithm outperforms all the other schemes in terms of bandwidth after the saturation point. It is interesting to note that the basic algorithm outperforms the static approach. This can be explained by the fact that the probability of finding a free rail is higher in the basic approach

[3] A study of the effect of the message size and the motivation for choosing 32KB can be found in [1]

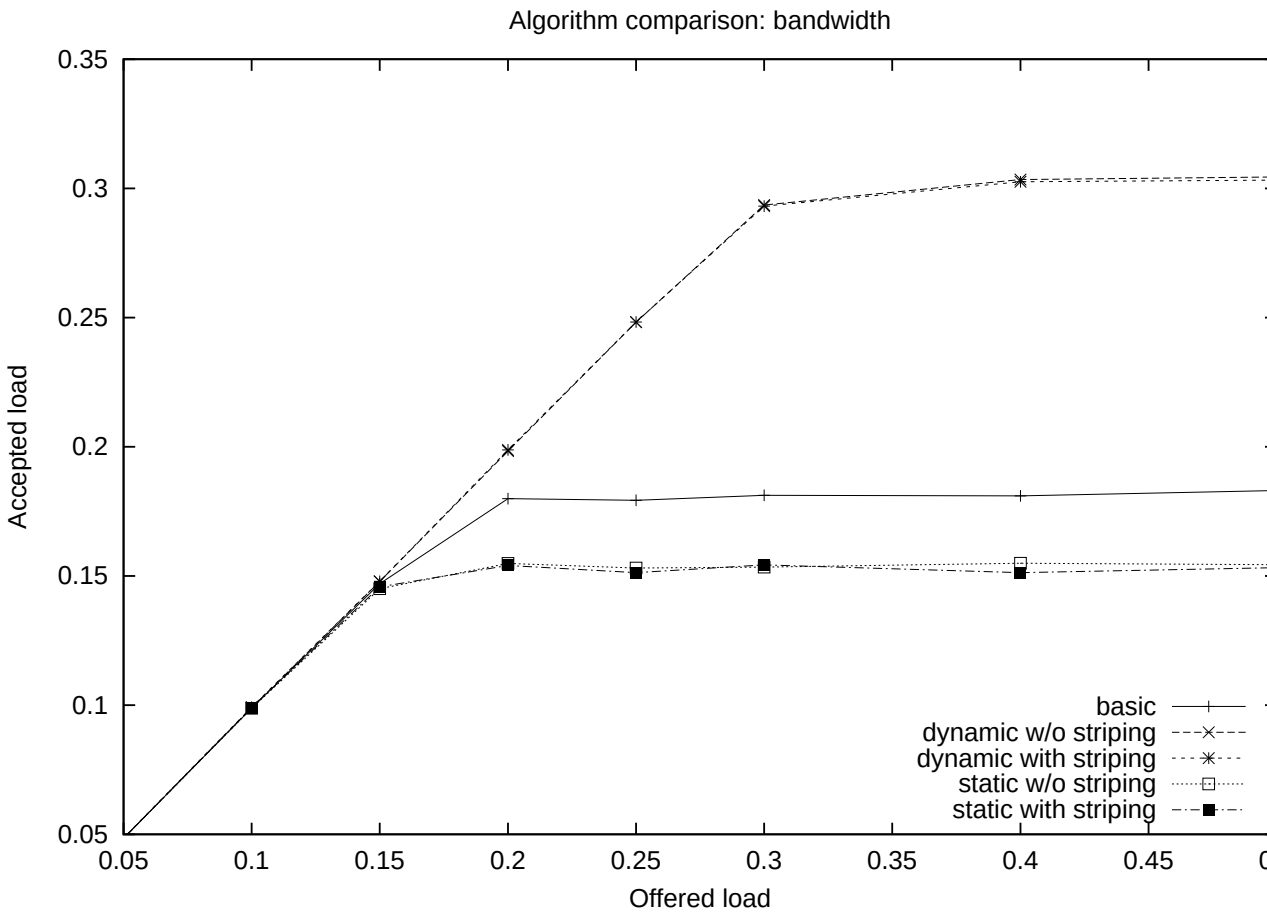


Figure 5: Bandwidth comparison

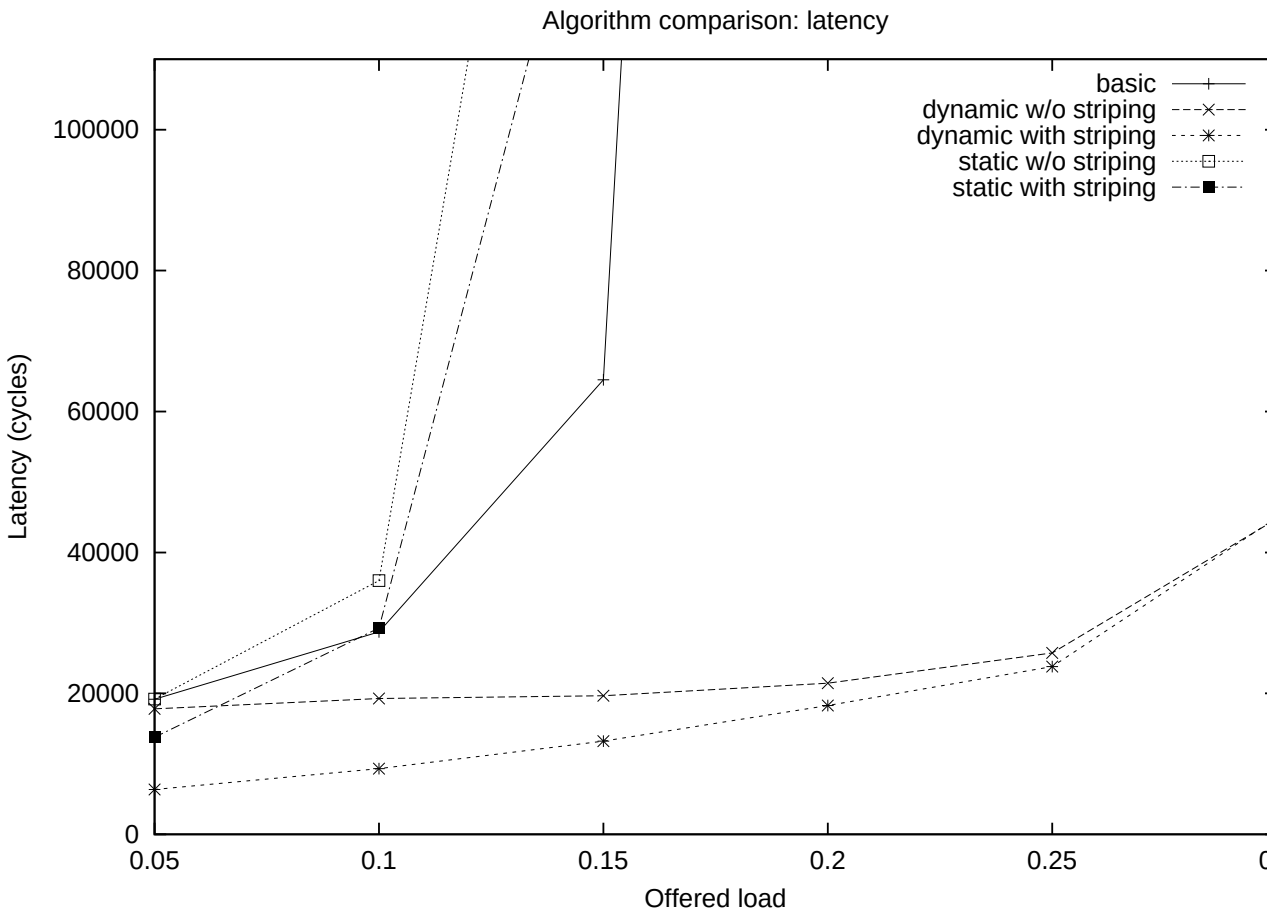


Figure 6: Latency comparison

than in the static approach, where the number of outgoing channels is severely limited. This restriction also penalizes the message latency. The basic algorithm has a higher probability of using a free rail because it has seven rails to pick from, albeit bidirectional, while the static algorithm has fewer rails to choose from, even if unidirectional. This suggests that the static approach may not be worthwhile to implement, even if enough rails are available for it. Its only merit is a simple implementation and lack of protocol overhead, but this can be obtained with the basic approach as well, with higher or better performance.

4 Conclusions

One of the novel methods that can be used to increase communication performance is using redundant networks (rails). In this report we explored various aspects of statically allocating multiple rails for unidirectional communication. We have shown that not only is static allocation very demanding in resources, it also performs quite poorly. In fact, we found it performs even worse than the basic algorithm, which uses only one rail in a round-robin fashion, both in terms of bandwidth and latency. This remains true even if we stripe messages over several rails, an approach which can lead to significant bandwidth improvment in dynamic allocation methods. We conclude that while static allocation may be simple to implement, it has no real advantage, and only dynamic allocation methods should be used on a multirail network for good unidirectional performance.

Acknowledgments

We thank José Duato for spearheading the project and for pointing out the limitations of the static approach.

References

[1] Salvador Coll, Eitan Frachtenberg, Fabrizio Petrini, Adolfy Hoisie, and Leonid Gurvits. Using Multirail Networks in High Performance Clusters. In *Third IEEE International Conference on Cluster Computing (Cluster'01)*, Newport Beach, CA, USA, October 2001.

[2] William J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[3] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[4] D. Lubell. A short proof of Sperner's theorem. *Journal of Combinatory Theory*, 1(299), 1966.

[5] Fabrizio Petrini, Adolfy Hoisie, Wu chun Feng, and Richard Graham. Performance Evaluation of the Quadrics Interconnection Network. In *Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.

[6] Fabrizio Petrini and Marco Vanneschi. SMART: a Simulator of Massive ARchitectures and Topologies. In *International Conference on Parallel and Distributed Systems Euro-PDS'97*, Barcelona, Spain, June 1997.

[7] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, January 1999.

[8] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, November 1999.