Communicating Process Architectures 2004
Ian East, Jeremy Martin, Peter Welch, David Duce, and Mark Green (Eds.)
IOS Press, 2004

An Evaluation of Inter-Switch Connections

Brian VINTER and Hans Henrik HAPPE
University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark

Abstract. In very large clusters it is not possible to get Ethernet-switches that are large enough to support the whole cluster, thus a configuration with multiple switches are needed. This work seeks to evaluate the interconnection strategies for a new 300+ CPU cluster at the University of Southern Denmark. The focal point is a very inexpensive switch from D-Link which unfortunately offers only 24 Gb ports. We investigate different inter-switch connections and their impact at application level.

1. Introduction

Cluster computing is today the most widespread architecture for supercomputing for the obvious reason that they offer inexpensive scalable solutions to most computing needs. We have earlier designed and build a 512 CPU cluster[1] based on 100Mb Ethernet. When designing with 100Mb networks a balanced interconnect could be obtained using 24 port 100Mb switches connected through a 1Gb switch, thus getting a 2.4:1 ratio on the internal to external available bandwidth. Today the standard PC is equipped with 1 Gb network interface and thus needs a 10Gb connection for a 24 port switch to keep the same ratio as we achieved on the previous 100Mb system. Unfortunately 10Gb Ethernet is currently targeted for long haul network backbones and thus priced far beyond what is available for cluster computer production.

Ethernet switches are time consuming[8] and the time of passing an Ethernet package through one switch is defined by:

$$T(1) = T(\text{Overhead}) + T(\text{Channel}) + T(\text{Routing})$$

where T(Overhead) is usually small, less than 2 us, and T(Routing) is less than 0.5 us on most modern switches. That leaves the T(Channel) as the major problem and this is heavily dependent on whether the switch is a 'store-and-forward' type, where the complete package is received at the switch before it is sent on to the receiver, or a 'worm-hole' switch, which forwards the data as they arrive. For a 'store-and-forward' switch we actually get:

$$T(1) = T(\text{Overhead}) + 2*T(\text{Channel}) + T(\text{Routing})$$

As we expand the number of switches we get:

$$T_{\textit{worm-hole}}(n) = T(\text{Overhead}) + T(\text{Channel}) + n*T(\text{Routing})$$
$$T_{\textit{store-and-forward}}(n) = T(\text{Overhead}) + n*(T(\text{Channel}) + T(\text{Routing})) + T(\text{Channel})$$

Unfortunately, Ethernet is mainly used for office use where store-and-forward has its advantages; thus worm-hole switches are not available and we have to use store-and-forward switches.

Now another opportunity has presented itself with the introduction of the D-Link 3324 series switch that are 24 port Gb switches with dedicated 10Gb uplink ports. The switch comes in two flavors: 3324sr with two 10Gb uplink ports and 3324sri with six uplinks running at 10Gb. The 3324 series can either be stacked to a height of 12 using only the 3324sr switches to a total of 288 Gb ports. Alternatively a 3324sri can be connected with up to six 3324sr switches to a total of 168 available Gb ports.

We are looking for at solution that scales beyond 300 CPUs – thus none of the solutions intended by D-Link are sufficient for our needs. Fortunately the 3324 is a layer 3 switch that offers link trunking and we thus imagine a system with two times 168 CPUs connected by an 8 Gb trunk. The question then is what is the impact of this?

In section 2, we describe the system used for the experiments. In section 3, the benchmarks are described and, then, the results are analyzed in section 4. Finally, the conclusions are summarized in section 5.

2. Test System

2.1 Cluster

The experiments are carried out on the SDU Roadrunner cluster, which consists of 48 nodes each with 1.7 GHz Intel Celeron CPU, i845 chipset and 256MB memory. Half the nodes (24) have an Intel Ethernet Express/Pro 1000 gigabit Ethernet adapter in addition to the onboard 100Mb NIC. The experiments will thus be carried out using these machines.

2.2 Switches

The experiments are centered on two D-Link 3324 switches, a 3324sr and a 3324sri. Since the purpose of the experiment is to investigate the effects of using two interconnected switches we choose a scenario where we use 24 nodes which can then be connected to one or two switches. Thus the results of the two-switch experiments can be compared to the performance when running on only one switch.

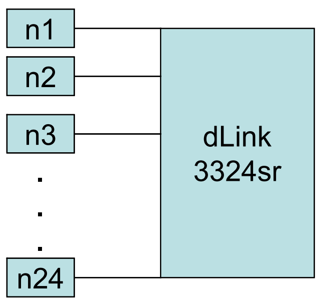

Figure 1: the “flat” model where all nodes are connected to one switch.

An identical experiment will be performed with the regular Gbit switch in Roadrunner, a HP2824, to verify the performance of the switches themselves.

The switches are intended to be interfaced through the dedicated 10Gb link which naturally will be the first alternative to running on one switch. The dedicated 10Gb link maintains the 2.4:1 ratio we have in our existing system which we know to be well balanced for most applications.

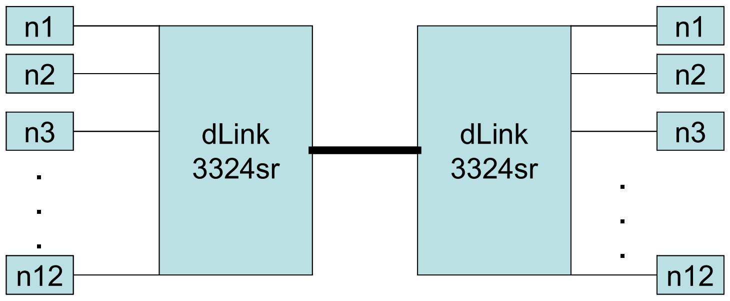

Figure 2: the “10G” model where the nodes are connected to two switches interconnected by a dedicated 10Gb link.

Unfortunately we cannot base the complete system on the 10Gb connections since these are limited to 288 ports in the stacking model and 168 in the star configuration. The 3324 however allows trunking of multiple links between two switches, thus we will also experiment with trunks from eight links, which is the maximum, down to a single link connecting the machines.

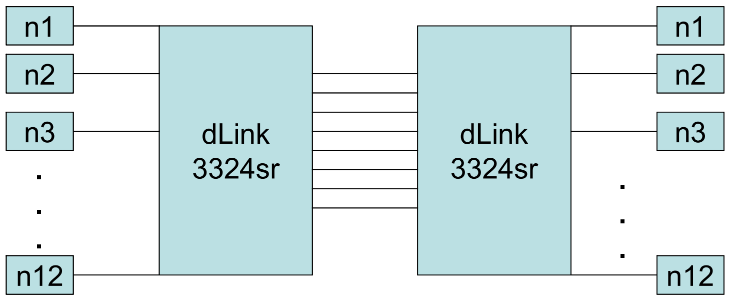

Figure 3: the “8” through “1” model where the nodes are connected to two switches interconnected by a trunk of width 1-8 links.

2.3 *Trunking Algorithm*

Trunking of ports between Ethernet switches means that the load is distributed across multiple links in order to obtain a higher bandwidth, in essence parallel paths from one switch to another. The 3324 comes with six algorithms for distributing the load over the trunked links. These algorithms really are just hash-keys to map incoming packages to ports. The available keys are IP source, IP destination, IP source and destination, MAC source, MAC destination and finally MAC source and destination. The switch comes with default setting of IP source, which makes good sense since nodes will often have a uniform distribution of IP addresses.

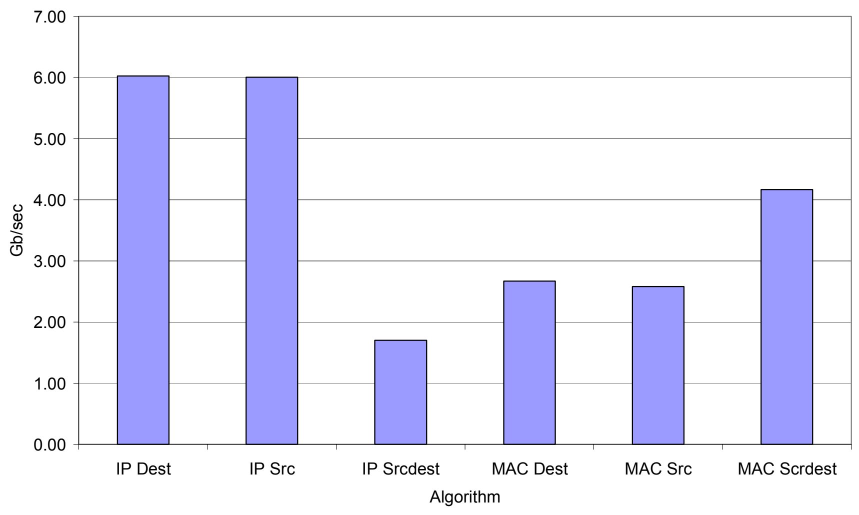

Figure 4: the performance of the different trunking algorithms.

The algorithms were tested with the code shown in Figure 5. The nodes in Roadrunner have contiguous IP addresses and the IP source model seems to fit the performance well, although the absolute result of 6 Gb/sec seems somewhat low. Thus for the remaining experiments we keep the IP source algorithm.

3. Benchmarks

Many benchmarks exists for testing cluster-computer performance, including micro-benchmarks such as Pallas[3], NAS[4] and HPL[5]. We have been designing and building clusters for a while, including the largest cluster within the Nordic countries, Horseshoe[6][7] and have developed a set of benchmarks that represents the type of code that our users run. The user base is 45% chemistry, 25% bio-science, 15% plasma-physics and the remaining 15% is distributed across a quite diverse group of researchers. The benchmark suite is used for testing all aspects of our system including CPU's, compute-nodes, compilers and in this case networking.

3.1 Bandwidth

We test the available bandwidth in two ways, a 'raw' test where a set of NPtcp tests are run in parallel and a structured test where 24 MPI processes exchange 10 messages of each 10MB.

```
partner = (rank+size+size/2)%size;
start=second();
for(i=0;i<10;i++){
  MPI_Irecv(in, dsize, MPI_BYTE, partner, 42, MPI_COMM_WORLD, &flag);
  MPI_Send(out,dsize, MPI_BYTE, partner, 42, MPI_COMM_WORLD);
  MPI_Wait(&flag, (void *)0);
}
stop=second();
if(rank==0)printf("%lf sec\n",stop-start);
```

Figure 5: the simple bandwidth test.

3.2 Monte Carlo Estimate of π

Monte Carlo methods are a widely used group of numerical methods, which involve sampling from random numbers. The Monte Carlo method can be used to solve otherwise intractable problems. Since Monte Carlo applications are based on random events a large number of such events must typically be processed to ensure a realistic result. The Monte Carlo method we use here, Monte Carlo π, is utterly uninteresting in and of itself but exhibits the same behavior as real world Monte Carlo applications as well as the related Las Vegas methods and random walks, all of which are frequently used in physics, biology and finance.

The Monte Carlo estimation of π is embarrassingly parallel, and is included as a sanity-check, if we do not get perfect speedup on this application there is an error somewhere.

3.3 Lower Upper Factorization

To perform the LU decomposition in parallel we basically have two choices, either divide the matrix by rows or by columns. If we divide the system by columns then one processor alone can decide the pivot row and broadcast the pivot to all processors. After this an all-to-all communication phase is needed to create a copy of the active row at all nodes. If the matrix is distributed row-wise then an actual election of the best pivot value is needed. After the election the process that won the election broadcasts its row.

We chose the row-wise solution in order to reduce the large messages to a one-to-all broadcast. Including the partial pivot this means that each iteration use two synchronous group operations, one election of the best divisor and a broadcast of the 'winning' row.

3.4 SOR

Successive Over-Relaxation, SOR, is a frequently used technique for solving very large systems of partial differential equations by successive approximations. The general idea is to approximate each element in a matrix to its neighbors until the sum of all changes within one iteration converges below a given value. The Red-Black checker pointing version of SOR, returns identical results for the same system of equations; independent of the actual computing environment, while at the same time providing sufficient parallelism that real speedup can be achieved. With Red-Black checker pointing, the equation system is divided into alternating red and black points in a chess-board fashion. Updating a red point depends only on black neighboring points and vice versa. Using this, an algorithm is derived where each worker-process updates all its red points and then exchanges red border point values with its neighbors. Each worker then updates its black points and repeats the communication for the black points. At the end of each iteration the global change in the system is calculated and the process continues until the change in the system is below a given threshold.

3.5 WATOR

The WAter TORus world, WATOR, is a classic discrete event simulation[1], and while it provides valuable information in itself we chose to introduce it here for reasons similar to the Monte Carlo π example, namely that it is simple and easy to understand, while still being typical for the class of applications that it represents. Discrete event simulations are widely used to model everything from digital systems to financial forecasts, logistics and traffic simulations.

WATOR is the simulation of a very special world; first of all the planet is not a sphere as most planets we know, rather the planet is shaped as a doughnut, or a torus, which greatly simplifies mapping the world into discrete blocks. As the whole surface of WATOR is covered with water there are only two types of life that we are interested in: fish and sharks.

Fish are simple organisms that move around at random, and at some point when a fish comes of age is will have two children and die itself. A fish can move into any of its neighboring eight squares, given that the square is empty, if all eight neighboring squares are occupied the fish remains in place. At any discrete time step a fish becomes one time-unit older, and once it has come of age it will split into two fish, one of which will go to a neighbor cell while the other stays in the cell where it was born, both new fish is age zero.

Sharks are similar simple creatures, however sharks also need to eat fish in order to survive. At each time-step a shark moves to a random neighboring cell which holds a fish, if there are no fish which neighbors the shark it moves to a random neighboring cell and increases its hunger index, if the hunger index reaches a starvation limit the shark dies, when the shark eats it resets the hunger index to zero. Similar to fish, sharks will at some point get old enough to breed and once this age is reached the shark is replaces by two new sharks, both with age and hunger index zero.

Thus the simulation of one time-step consists of two steps, first all fish are moved, then the sharks, each step issues four asynchronous MPI operations, two send and two receive. The randomness of the application allows us to ignore further synchronization issues.

3.6 Traveling Salesman Problem

The Traveling Salesman Problem, TSP, is a classic representative for the class of global optimization problems. The TSP solution we use in this work is a depth-first branch-and-bound algorithm which makes the parallel version different from the other applications we use by the fact that a static division of the work would result in a highly unbalanced execution. Thus the parallel TSP is implemented as a bag-of-tasks application, a paradigm that does not come natural to the SPMD programming paradigm that MPI is designed around.

The parallel TSP is implemented as a global master process and set of worker-threads on each node, each thread communicates with the master to retrieve jobs and submit results. A job is represented as a set of cities that have already been placed and a set that need to be placed, i.e. a sub-tree. Each job that is sent from the master has the length of the shortest known route piggy-backed and each node maintains one shared instance of the bound value. Since the application is so highly unbalanced in the workload this application use synchronous messages for communication. A scheme for applying asynchronous messages could be developed, but two things talk against it, first of all an asynchronous scheme would place an extra job at each node, which is likely to increase the load-unbalance.

3.7 Matrix Multiplication

Matrix multiplication is frequently used in scientific applications. Although several concurrent and parallel algorithms exist that require extensive communications during the calculation, an alternative approach is to distribute the matrixes coarsely amongst the nodes and broadcast one matrix to all nodes one block at a time.. This approach is similar to the one used in PBLAS [2].

We have chosen a less generic, but more efficient, algorithm where we maintain both A and B in distributed state and thus only need to broadcast one matrix to the other processes. Also, the matrix is not broadcast column by column, but the entire block at once. Each node uses an efficient sub-blocked matrix-multiplication to take advantage of cache-memory.

The MPI solution results in one synchronous broadcast per node in the overall execution. The broadcasts are rather large however and will stress the switch significantly.

4. Results

4.1 Bandwidth

The first bandwidth test consists of twelve parallel NPtcp runs where nodes 1-12 transmits to nodes 13-24, the test is not synchronized and from the results it is apparent that some of the tests terminate before the last starts. This can be seen from the fact that using one Gb link the test yields 250Mb/sec which cannot be offered to 12 nodes at a time since this would be 3 Gb/sec.[1]

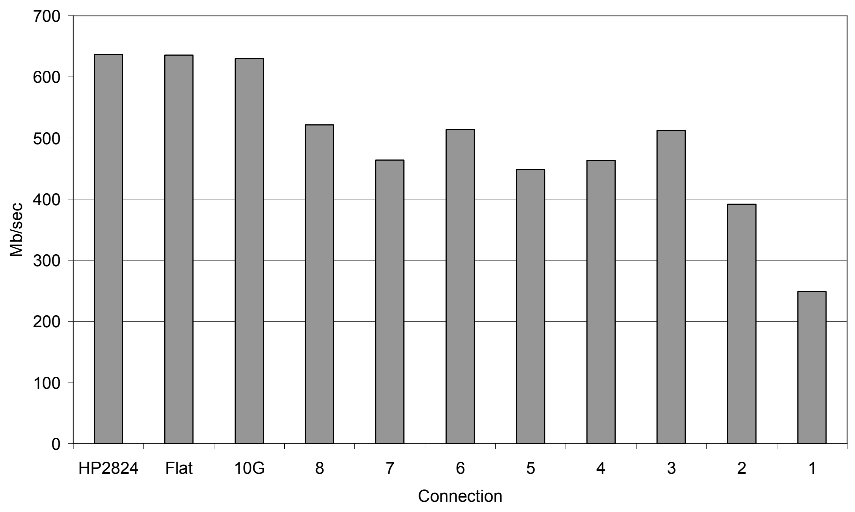

Figure 6: the bandwidth process measured with NPtcp.

The bandwidth test in figure 6 shows a somewhat lower bandwidth but it seems that this is due to the performance of the nodes and the MPI implementation. We are using LAM MPI for these experiments and this can be the reason for the difference in bandwidth between figures 4 and 6. As a sanity check we have run the experiments on the HP2824 switch also and that performs exactly as the D-Link 3324. Thus we can assume that the limited bandwidth is not a result of the switch. This has been verified by running the MPI-bandwidth test on a faster cluster using the lower grade HP2724 switch. In this scenario 8.2 Gb/s was achieved with only 16 nodes.

[1] The PCI bus unfortunately has problems breaking the 500Mb barrier on our system.

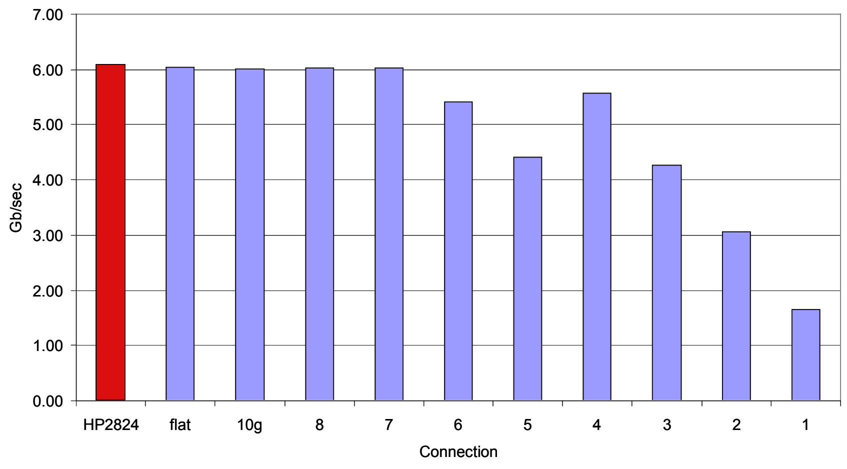

Figure 7: the bandwidth measured with the MPI program from Figure 5.

From the “4-trunk” down to the “1-trunk” we see a linear decrease in throughput which is what one would expect. With one Gb link the throughput is down to 1.7 Gb, less than the maximum of two Gb but still a reasonable performance.

4.2 Applications

The applications all run quite well and only when the bandwidth between the node becomes very low do two of the applications exhibit an actual performance decrease. Figure 8 shows the increased runtime for all the applications when the two switches are connected by a single link. Obviously only the LU and Matrix Multiplication applications are influenced by the limited bandwidth. Interestingly both of these applications depend heavily on broadcasts.

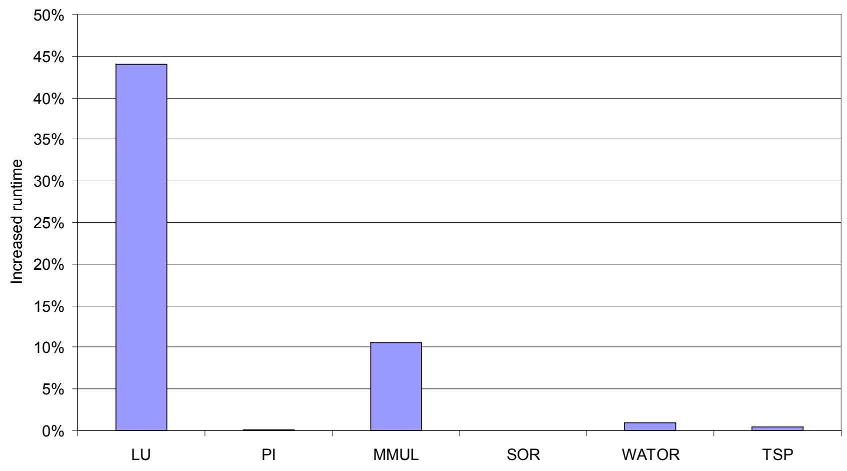

Figure 8: the increased runtime when running on a single Gb link between the switches.

The experiments below were run in two setups; one where MPI was given the nodes in sequence (“linear”) and one where they were set up on each side of the switches (“cross”).

4.2.1 LU

The LU test fluctuates quite a lot in its execution times, however it is evident that the "linear" boot-sequence performs worse than the "cross" sequence. The poor performance for the "cross" model on the "2-trunk" is hard to explain, but otherwise the "linear" is much worse and when only one link is available runtime increases by almost 45%.

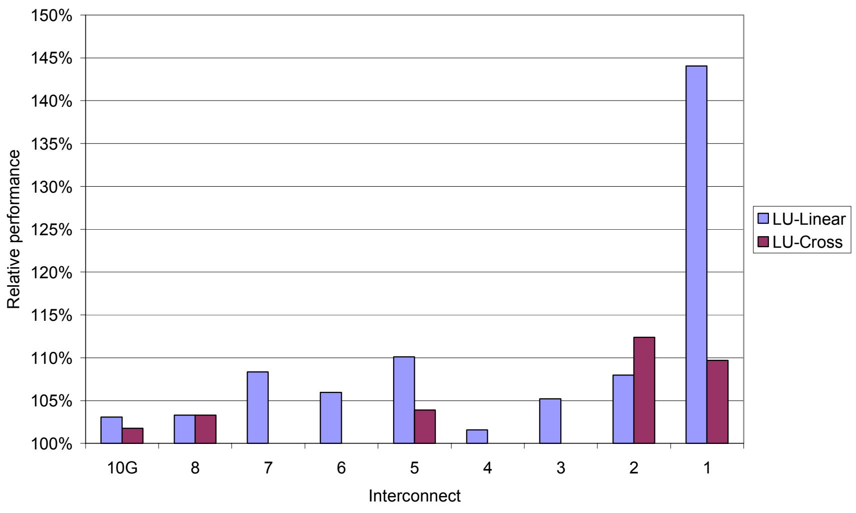

Figure 9: the impact of the interconnect on the LU application.

4.2.2 Matrix Multiplication

The Matrix Multiplication test is not as sensitive to the bandwidth as the LU test but in this experiment it is significant that the "cross" boot-schema performs much worse than the "linear". Once again the "2-trunk" model performs worse than the others but otherwise it is clear that even a 4 Gb connection between the switches is sufficient to satisfy this application.

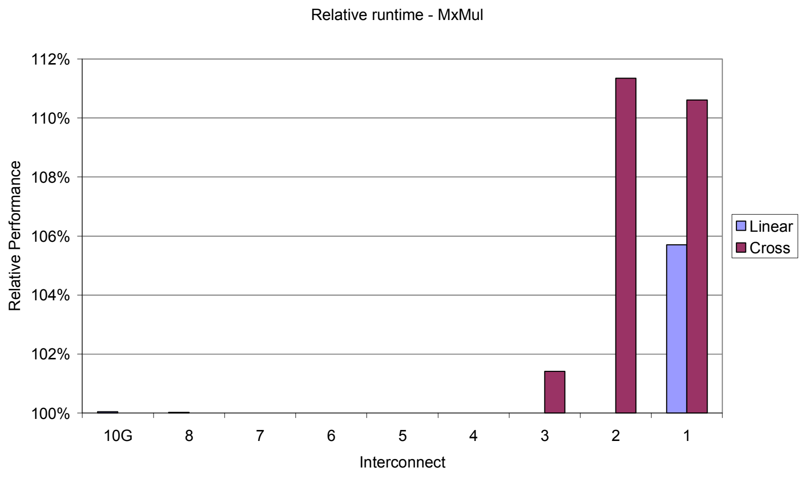

Figure 10: the impact of the interconnect on the Matrix Multiplication.

5. Conclusions

None of the applications exhibit performance degradation above 5% as long as the connection ratio does not drop below 1:6 and most of the applications don't reach any level of degradation at all.

Thus we conclude at we can base our large installation on these cheap switches and propose the layout shown in figure 11. For most jobs we can maintain the 2.4:1 ratio and only when more than 152 CPU's are used will we be forced to attempt the 19:1 ratio. If an application suffers from this ratio it will have to be forced to exist with one of the two "wings" in figure 11 where the 2.4:1 ratio is maintained.

The 10Gb uplinks deliver what was promised and this allows us to maintain the existing 2.4:1 ratio, if we force an application to stay within one 152 CPU segment of the cluster, that we know scales to 652 CPU's today. Thus we dare conclude that since this existing ratio can also be meet here we dare base our new cluster on the D-Link switches.

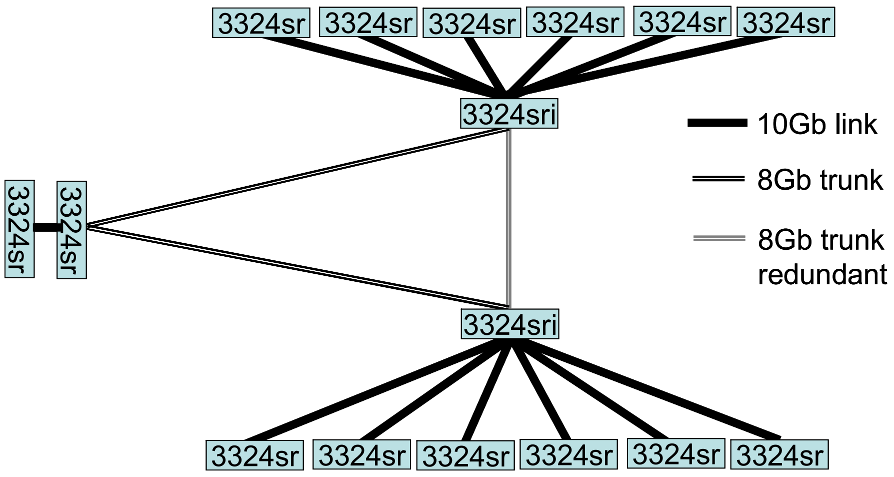

Figure 11: the final 304 port switch design.

Acknowledgements

We would like to thank D-Link for lending us the switches for these experiments.

References

[1] A.K. Dewdney. *Computer recreations.* Scientific American, 250:22–34, 1984.

[2] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert A. van de Geijn. *Parallel implementation of BLAS: General techniques for level 3 BLAS.* Technical report, The University of Texas at Austin Austin, Texas 78712, 1995.

[3] `http://www.pallas.com/e/products/pmb/index.htm`

[4] `http://www.nas.nasa.gov/Software/NPB/`

[5] `http://www.netlib.org/benchmark/hpl/`

[6] Brian Vinter, *Design and implementation of a 512 CPU cluster as a general purpose SuperComputer.* Proceedings of ParCo 2003, Dresden University of Technology.

[7] `http://www.dcsc.sdu.dk/`

[8] Cost/Performance evaluation of Gigabit ethernet and Myrinet as Cluster Interconnect, Proceedings of Opnetwork 2000, Washington, DC, August 2000.