2.10 Clos-networks

Theory of Clos networks came into existence with telephone networks – paper : : Charles Clos, – “*A Study of Non-Blocking Switching Networks*”, 1952.

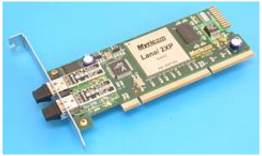
2.10.1 Myrinet

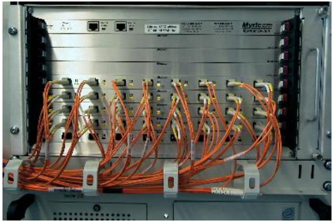
Myrinet (`http://www.myri.com`)
Standardised network architecture
Aims:

- Max throughput with scalability to large number of processors
- Low latency
- eliminate the need to allocate processors according to network topology
  - (historical remark: in the same reasons, RAM replaced *sequential access memory* (disk, trumble)

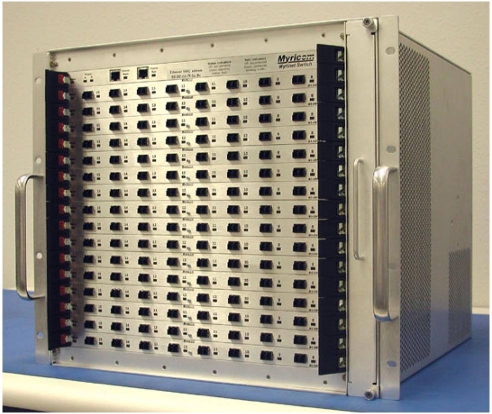
In general, good network makes $1/2$ cluster cost!

Myrinet card and switch



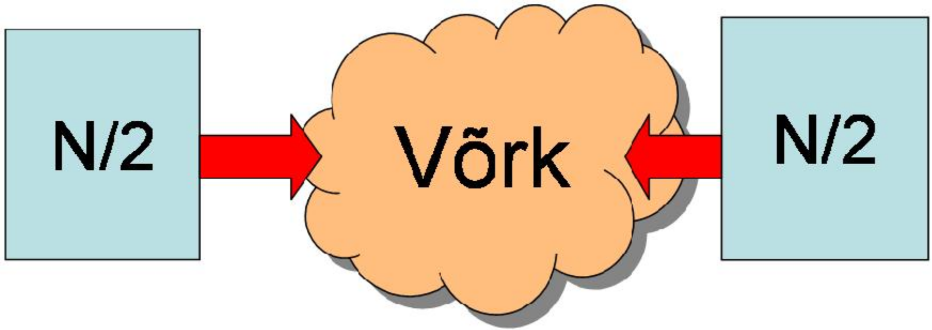


128-port Myrinet Clos network switch



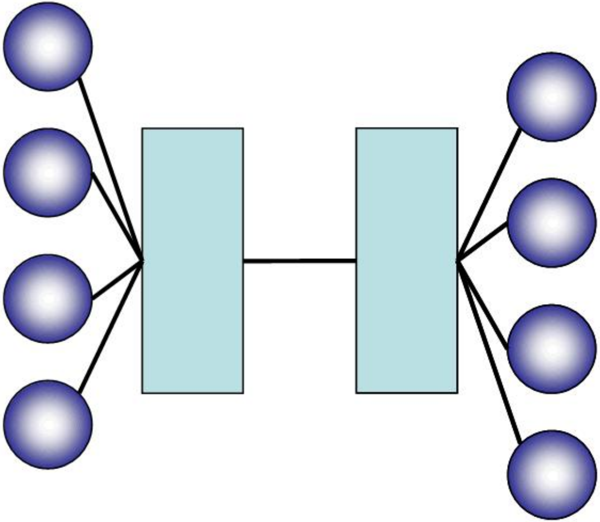
2.10.2 Topological concepts

- **Minimum bisection** (*minimaalne poolitus*)
  Definition. Minimal number of connections between arbitrary two equally sized sets of nodes in the network



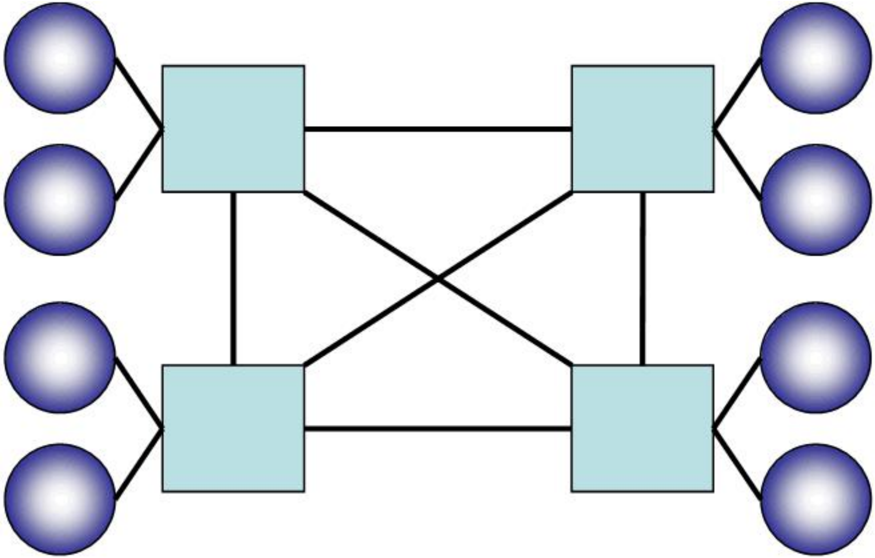
- Describes minimal throughput, independently from communication pattern
- Upper bound in case of N-node network is N/2 connections
- Network topology achieving this upper bound is called ***full bisection*** (*täispoolitus*)

Example. 8 nodes, network switch has 5 ports:



Minimal bisection is 1 – i.e. very bad

But, if to set the network up as follows:



Question: Does there exist full bisection? If yes, how?

Answer: minimal bisection: 4. But not in case of each possible communication pattern! (find!) => not rearrageable network

2.10.3 Rearrangeable networks (*ümberjärjestatavad võrgud*)

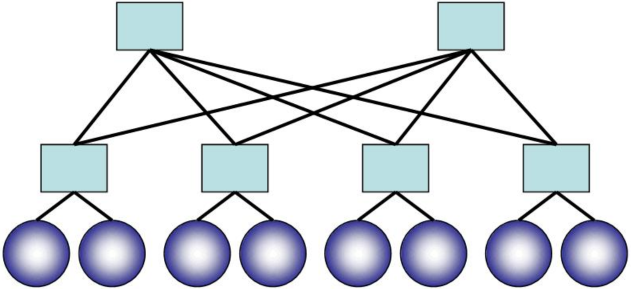
Definition. Network is called rearrangeable, if it can find a route in case of arbitrary permutation of communication patterns (or, whichever node wants to talk to with whichever other node, there is always a way to do it simultaneously)

Theorem. Whichever rearrangeable network has full bisection.

The opposite is not true. Network can have full bisection without being rearrangeable.

(Proof is based on Hall's Theorem)

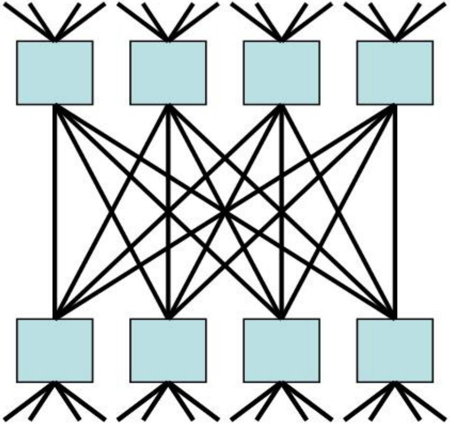
Example: 8-node clos network with 4-port switches.
"Leaf"-switches
"Spine"-switches

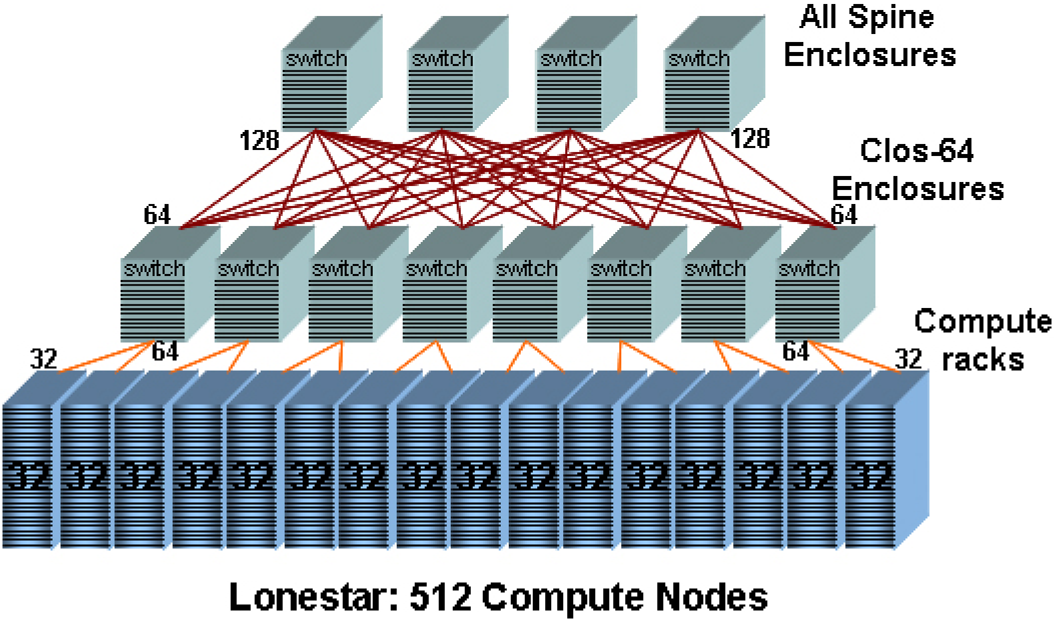


Main rule: As many connections to the leaves as many there are to the spine-switches (enables rearangeability) **Therefore:** Clos networks are:

- scalable upto a large number of nodes
- rearrangeable (proved, that in each permutation there exists a routing table)
- There exist multiple routes

In case of too few ports on a switch, it is possible to compose a larger one:




All Spine
Enclosures
switch
switch
switch
switch
128
128
Clos-64
Enclosures
64
64
switch
switch
switch
switch
switch
switch
switch
switch
32
64
64
32
Compute
racks
32
32
32
32
32
32
32
32
32
32
32
32
32
32
32
32
Lonestar: 512 Compute Nodes

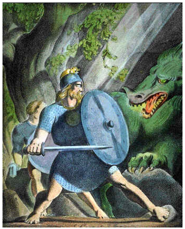
2.11 Beowulf clusters

- History

- 1993-1994 Donald Becker (NASA) – 16 486 DX4 66MHz processors, ETHERNET-channel bonding, COTS (*Commodity Off The Shelf*) – created in CESDIS (*The Center of Excellence in Space Data and Information Sciences*), 74 MFlops (4,6 MFlops/node).

- ***Beowulf*** – formerly project name, not the system
  Dr Thomas Sterling: "*What the hell, call it 'Beowulf.' No one will ever hear of it anyway.*"

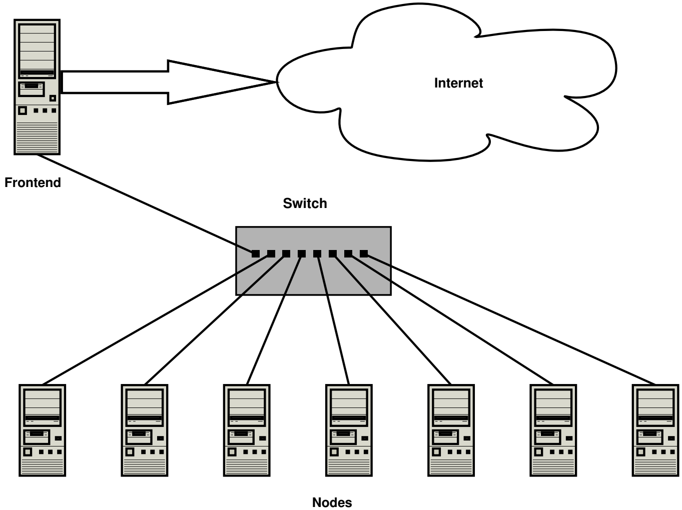
- **Beowulf** - first poem in English based on Scandinavian legend about hero who defeated the monster Grendel with his courage and strength



- Network choice
  Ethernet 100Mbps, Gigabit Ethernet, Infiniband, Myrinet (*Scali, pathscale, numalink, etc*)

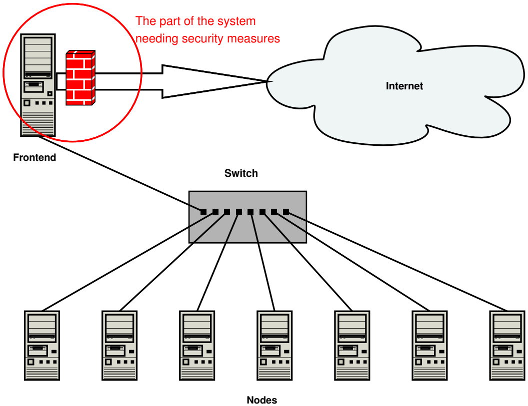
- OS – most often linux

- Typical Beowulf cluster



- Building a cluster (rack/shelf)?

- console switch?
- Security



- which software.
- Cluster administration
  - ClusterMonkey
  - Rocks
  - Clustermagic
  - etc.

3 Parallel Performance

How to measure parallel performance?

3.1 Speedup

$$S(N,P) := \frac{t_{seq}(N)}{t_{par}(N,P)}$$

$t_{seq}(N)$ – time for solving given problem with **best known** sequential algorithm

- In general, different algorithm than the parallel one
  - If same (and there exist a faster sequential one):

Relative Speedup

- $0 < S(N,P) \le P$

- If $S(N,P) = P$, – **linear speedup** or **optimal speedup** (Example, embarrassingly parallel algorithms)
- May happen that $S(N,P) > P$, (swapping; cache effect) – **superlinear speedup**
- Often $S(N,P) < 1$, – slowdown

3.2 Efficiency

$$E(N,P) := \frac{t_{seq}(N)}{P \cdot t_{par}(N,P)}.$$

Presumably, $0 < E(N,P) \leq 1$.

3.3 Amdahl's law

In each algorithm $\exists$ parts that cannot be parallelised

- Let σ $(0 < \sigma \leq 1)$ – sequential part
- Assume that the rest $1-\sigma$ parallelised optimally

Then, in best case:

$$S(N,P) = \frac{t_{seq}}{\left(\sigma + \frac{1-\sigma}{P}\right) t_{seq}} = \frac{1}{\sigma + \frac{1-\sigma}{P}} \leq \frac{1}{\sigma}.$$

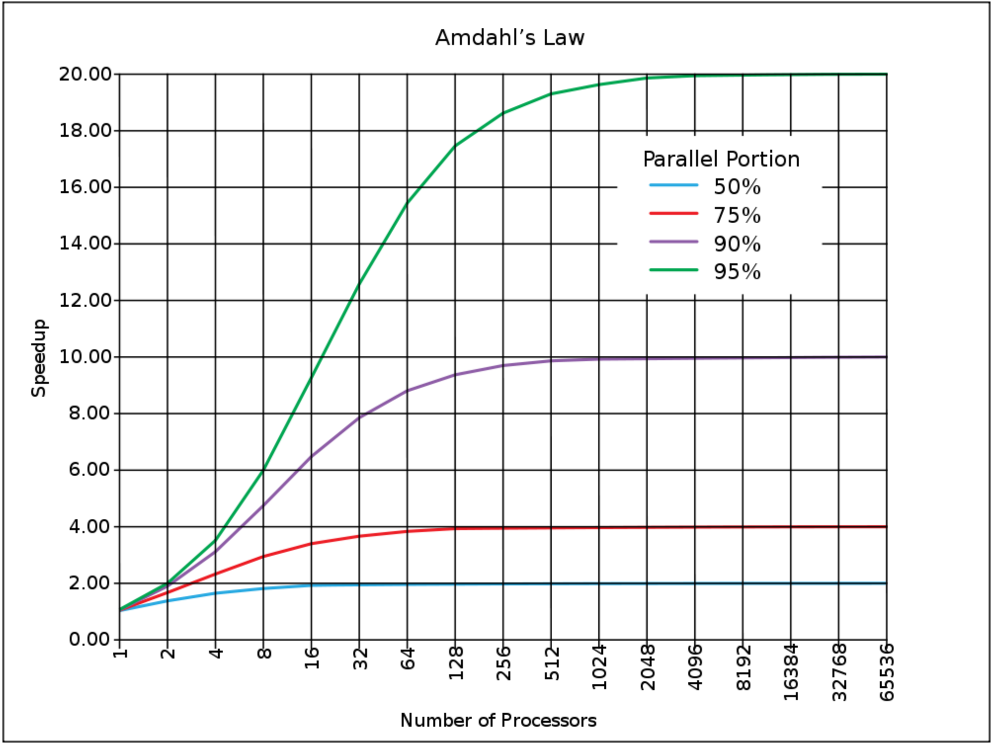
Example 1. Assume 5% of the algorithm is not parallelisable (ie. $\sigma = 0.05$) => :

P	max $S(N,P)$
2	1.9
4	3.5
10	6.9
20	10.3
100	16.8
∞	20

Therefore, not much to gain at all with huge number of processors!

Example 2. If $\sigma = 0.67$ (33% parallelisable), with $P = 10$:

$$S(N,10) = \frac{1}{0.67 + 0.33/10} = 1.42$$


Amdahl's Law
Parallel Portion
50%
75%
90%
95%
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

3.4 Gustafson-Barsis' law

John Gustafson & Ed Barsis (Scania Laboratory) 1988:

- 1024-processor nCube/10 claimed: they bet Amdahl's law!
- Their $\sigma \approx 0.004...0.008$
- but got $S \approx 1000$
- (Acording to Amdahl's S might be 125...250)

How was it possible?

Does Amdahl's law hold?

Mathematically – yes. But in practice – not very good idea to solve a problem with fixed size N on whatever number of processors!

In general, $\sigma = \sigma(N) \neq \text{const}$

Usually, σ **decreases with** N **growing!**

Algorithm is said to be **effectively parallel** if $\sigma \to 0$ with $N \to \infty$

Scaled efficiency to avoid misunderstandings:

3.5 Scaled efficiency

$$E_S(N,P) := \frac{t_{seq}(N)}{t_{par}(P \cdot N, P)}$$

- Problem size increasing accordingly with adding new processors – does time remain the same?
- $0 < E_S(N,P) \leq 1$
- If $E_S(N,P) = 1$ – linear speedup

3.6 Methods to increase efficiency

Factors influencing efficiency:

- communication time
- waiting time
- additional computations
- changing/improving algorithm

Profiling parallel programs

- MPE - jumpshot, LMPI, MpiP
- Totalview, Vampir, Allinea OPT
- Linux - gprof (compiler switch -pg)
- SUN - prof, **gprof**, prism
- Many other commercial applications

3.6.1 Overlapping communication and computations

Example: Parallel Ax-operation for sparse matrices

par_sparse_mat.f90 (http://www.ut.ee/~eero/F95jaMPI/Kood/mpi_CG/par_sparse_mat.f90.html) ("Fortran95 ja MPI" pp. 119-120)

Matrix partitioned, divided between processors. Starting communication (non-blocking); calculations at inside parts of the region => economy in waiting times.

3.6.2 Extra computations instead of communication

- Computations in place instead of importing the results over the network
- Sometimes it pays off!

Example: Random number generation. *Broadcast*ing only seed and generate in parallel (deterministic algorithm)

3.7 Benchmarks

5 main HPC (*High Performance Computing*) benchmarks:

- NPB
- Linpack
- HINT
- Perf
- IOzone
- Graph 500

3.7.1 Numerical Aerodynamic Simulation (NAS) Parallel Benchmarks (NPB)

MPI, 8 programs (IS,FT,MG,CG,LU,SP,BT,EP) from Computational Fluid Dynamics (CFD) code

- **Integer Sort** (**IS**) - integer operations and communication speed. Latency of critical importance
- **Fast Fourier Transform** (**FT**). Remote communication performance; 3D ODV solution

- **Multigrid** (**MG**). Well-structured communication pattern test. 3D Poisson problem with constant coefficients
- **Conjugate Gradient** (**CG**). Irregular communication on unstructured discretisation grid. Finding smallest eigenvalue of a large positive definite matrix
- **Lower-Upper diagonal** (**LU**). Blocking communication operations with small granularity, using SSOR (Symmetric Successive Over-Relaxation) to solve sparse 5x5 lower and upper diagonal systems. Length of the message varying a lot
- **Scalar pentadiagonal (SP) and block tridiagonal (BT)**. Balance test between computations and communication. Needs even number of processes. Solving a set of diagonally not dominant SP and BT systems. (Although similar, difference in the ratio of communication and computations – SP having more communication than BT
- **Embarrassingly Parallel (EP)**. Gauss residual method with some specifics; showing the performance of the slowest processor.
- + some new tests in version 3, good with networked multicore processors

3.7.2 Linpack

Jack Dongarra. HPL - *High Performance Linpack*, using MPI and BLAS. Solving systems of linear equations with dense matrices. The aim is to fit a problem with maximal size (advisably, utilising 80% of memory).

- Used for http://www.top500.org
  - Also, http://www.bbc.co.uk/news/10187248

- R_{peak} - peak performance in Gflops
- R_{max} - maximal achieved performance in Gflops
- N- size of matrix giving peak performance in Gflops (usually <80% memory size)
- NB - blocksize. In general, the smaller the better, but usually in range 32...256.

3.7.3 HINT benchmark

The HINT (Hierarchical INTegration). Quite popular benchmark. Graphical view of:

- floating point performance
- integer operation performance
- performances with different memory hierarchies

3.7.4 Perf

Measures network latency and banwidth between two nodes

3.7.5 IOzone

Tests I/O performance

3.7.6 Graph 500

- Graph 500 benchmark initiative (http://www.graph500.org/specifications)