Performance-Engineered Computer Networks

**INTERCONNECTION NETWORKS**, sometimes called **SYSTEM AREA NETWORKS** (SANs), play a critical role in all types of parallel computers – be they clusters spanning many racks or many cores on the same chip. A well-engineered network can dramatically outperform the obvious alternatives. At Aggregate.Org, we have created **public domain technologies and tools for performance-engineered networks.**

**AGGREGATE FUNCTION NETWORKS (AFNs).** An AFN is a parallel function unit that collects global state information and computes functions of that state. In 1994, we built our **PAPERS AFN** (Purdue’s Adapter for Parallel Execution and Rapid Synchronization); using secure OS-bypass I/O, it delivered 3μs barrier synchronization. Other AFN operations include broadcast & multicast, putget, reductions, scans, searches (e.g., first or count), voting & scheduling, and parallel signaling (eurekas). A 2017 software-implemented AFN uses shared memory and TCP.

**FLAT NEIGHBORHOOD NETWORKS (FNNs).** FNNs provide single-switch latency, and better bisection bandwidth than a fat tree with comparable hardware complexity, by using multiple network interfaces per node. As shown above for 4-port switches connecting 8 nodes, an FNN connects nodes to switches such that each node pair has at least one switch in common. The best wiring pattern usually is asymmetric; the design is evolved from random wiring patterns using a genetic algorithm (GA).

FNN design problems and solution quality are summarized by square maps in which the darkness of each point indicates how many single-switch paths exist between the corresponding pair of nodes; the lower left triangle is the minimum design requirement, the upper right is what the FNN design actually provides. The photo and map above are for the world’s first FNN, KLAT2 (Kentucky Linux Athlon Testbed 2), built in 2000.

**SPARSE and FRACTIONAL FLAT NEIGHBORHOOD NETWORKS (SFNNs and FFNNs).** Surprisingly, very few parallel programs depend on every node talking to every other; usually, each node talks to at most \(O(\log(N))\) other nodes. This is true even using personalized all-to-all as MPI typically implements it.

SFNNs ensure single-switch latency only for all node pairs that are expected to communicate, thus requiring shockingly few cheap, narrow, switches even for systems having many thousands of nodes. The first SFNN was KASY0 (Kentucky Asymmetric Zero) in 2003 (above). FFNNs flip priorities, finding the best coverage possible with a fixed network cost. For example, using far less hardware than KASY0, the above map shows coverage of an FFNN in green – only the red spots deliver poorer latency.

**KENTUCKY’S NETWORK IMPLEMENTATION TOPOLOGY TOOL (KNITT).** Knitting converts 1D material, such as yarn or network cable, into a 2D or 3D structure. KNITT uses genetic algorithm (GA) evolutionary search technology to create the physical placement structure that will most efficiently implement the wiring of a given logical interconnection pattern. Above is partitioning of a fat tree with 64 nodes and 16-port switches into 3 racks; only 26 of the 96 cables cross racks instead of the expected 40.

**NETWires and NodeScape.** As networks become more complex, visualizing networked systems becomes more important. NETWires draws static diagrams of networks, like the fat tree above. NodeScape colors a diagram or actual photo of a parallel computer according to properties sampled from the running system, such as node temperature (below) or load average – which is very useful for KNITT-scrambled physical node placements.

@techreport{sc17pecn, author={Henry Dietz}, title={Performance-Engineered Computer Networks}, institution={University of Kentucky}, address={http://aggregate.org/WHITE/sc17pecn.pdf}, month={Nov}, year={2017}}