INCREASING HYPERCUBE COMMUNICATIONS ON LOW-DIMENSIONAL PROBLEMS

John L. Gustafson
Floating Point Systems, Inc.

September 26, 1986

ABSTRACT

We present a simple but powerful hardware technique for greatly increasing interprocessor bandwidth on an arbitrary hypercube of processors. If each processing node has N links to other nodes, then a configuration with only N/M links in use is capable of increasing bandwidth by M-fold for applications requiring only (N/M)-dimensional interconnect. This allows hypercubes to combine the advantages of parallel buses and robust serial interconnections. This is particularly important on hypercubes favoring operations on contiguous vectors, since such machines encourage domain decomposition along “pencils” occupying an entire dimension rather than subregions which minimize the ratio of surface area to volume. A specific example is given for the FPS T Series, which presents the additional challenge of having multiplexed links. T Series configurations up to 128 processors can elect to use a planar (toroidal) interconnect as a linear (ring) interconnect operating at double the communications rate. Examples of increased speed on applications are given.

Introduction

Multiprocessors based on the binary N-cube interconnect contain nearest-neighbor meshes of dimension less than or equal to N. This can be demonstrated by *partitioning* the N-digit binary (Gray code) numbering of each node. Within a partition of n bits, the Gray-code changes in bits represent linear numbering of 2^n processors along a particular dimension. Different partitions represent different dimensions. For example, suppose processors in a 6-dimensional binary cube have numbers represented $b_1b_2b_3b_4b_5b_6$, where b_i is a binary digit. Then the partitioning $(b_1)(b_2)(b_3b_4b_5b_6)$ represents a three-dimensional domain of size $2{\times}2{\times}16$ (with wraparound). Thus, the extreme cases are N partitions with one bit each (representing an N-dimensional domain with two processors on every edge) and a single partition representing a 1-dimensional domain of 2^N processors.

Physical implementations of hypercube computers require the existence of N physical links out of each processor, but do *not* specify how those physical links are selected or switched out of the CPU. For example, one might be restricted to the use of a single link at any time, and communication across that link might not be able to overlap other CPU activities. This is the situation in the Caltech Cosmic Cube, and in the Intel iPSC productization of that hypercube. The NCUBE design allows simultaneous operation of many DMA links; roughly, 9 of the 22 links (11 input, 11 output) can be active at any time before main memory bandwidth is exhausted. The FPS T Series processors are intermediate between these two, in that 4 out of 15 bidirectional links can be active at any time, and can overlap other CPU activities.

The two preceding paragraphs point to the conclusion that *communications bandwidth is wasted when running problems of low dimensionality* on machines that have the capacity for using several links simultaneously. Interprocessor communication is typically the most precious resource in a hypercube and is often the most important factor in application efficiency. In principle, the hardware connections in a hypercube are entirely point-to-point, implying that there is really no way to recoup the unused links without changing the system framework. In practice, there is a solution that exploits the fact that some systems have a *maximum* configuration which is much larger than the *typical* configuration.

Link Doubling

The FPS T Series provides an example of using link doubling to improve effective communication speed, although the technique is by no means confined to that version of the hypercube. The four bidirectional links embodied in the Inmos Transputer in a node are each software-switchable to one of four interprocessor links. The former are referred to herein as "hard links" whereas the latter are referred to as "soft links." They are organized as shown in Figure 1.

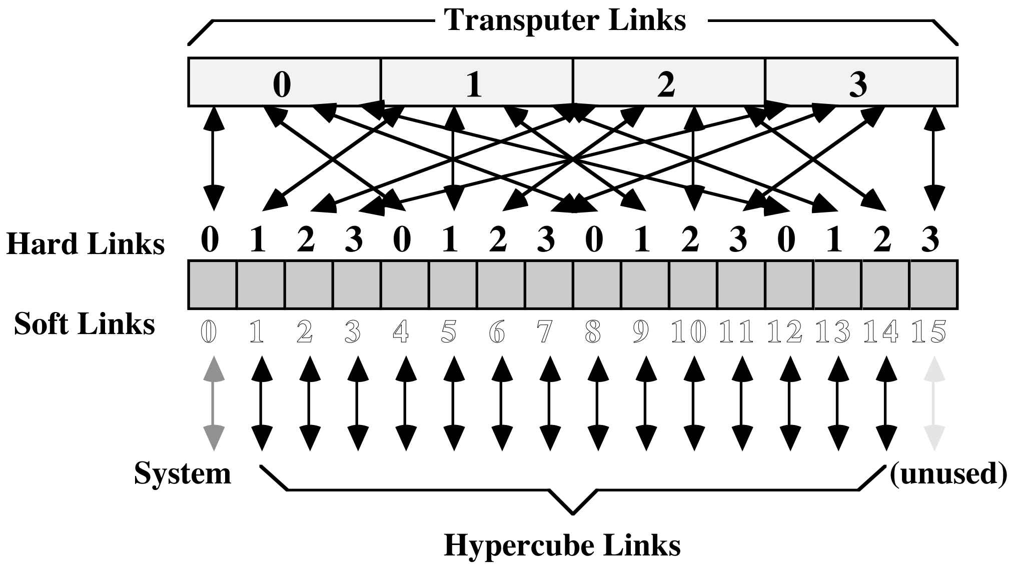

Figure 1. T Series Link Organization

If i is the number of a soft link and j is the number of the hard link to which it maps, then

$$j = i \bmod 4$$

so that the burden on the hard links increases evenly as the system grows. The "system ring" uses soft link 0, the one-dimensional interconnect used for such functions as uploading or downloading data and programs. Soft link 15 is not driven. The remaining 14 are available for hypercube interconnections.

Each processor occupies one circuit board, and eight boards are combined with system facilities to form a “module” which consumes the system soft link 0 and the first three hypercube links. Note that there is thus no need to ever multiplex the links for the module, or 3-cube. The packaging is such that all external cabling is module-to-module, using ordinary D-connectors with 8 bidirectional channels (32 wires). Two modules are mounted in a cabinet connected with such a cable combining soft link 4 from every node. At this point, there is contention between soft link 4 and soft link 0, since both map to hard link 0. To reassign all four hard links to a different set of soft links takes about 320 µseconds. If no reassignment is needed, message startup time is quite low… approximately 9 µseconds.

Larger systems further increase the number of roles played by the hard links. The largest system so far configured uses up to soft link 6 (64 processors). At any time, the hard links constitute a two-dimensional (toroidal) domain capable of fine-grained parallelism. Applications requiring simultaneous use of more than four soft links tend to be restricted to coarser-grained parallelism because of the time to reconfigure links.

Less than half of the soft links have been used, suggesting that in fact the remaining links can be *doubled up* by adding cables to the system. This need have no impact whatsoever on system software which already ignores those channels on small systems, but provides the applications programmer with additional paths for communication. When links are doubled, the two-dimensional hard interconnect becomes a one-dimensional hard interconnect with *twice the bandwidth* between nearest-neighbor processors. A doubling scheme is shown in Figure 2.

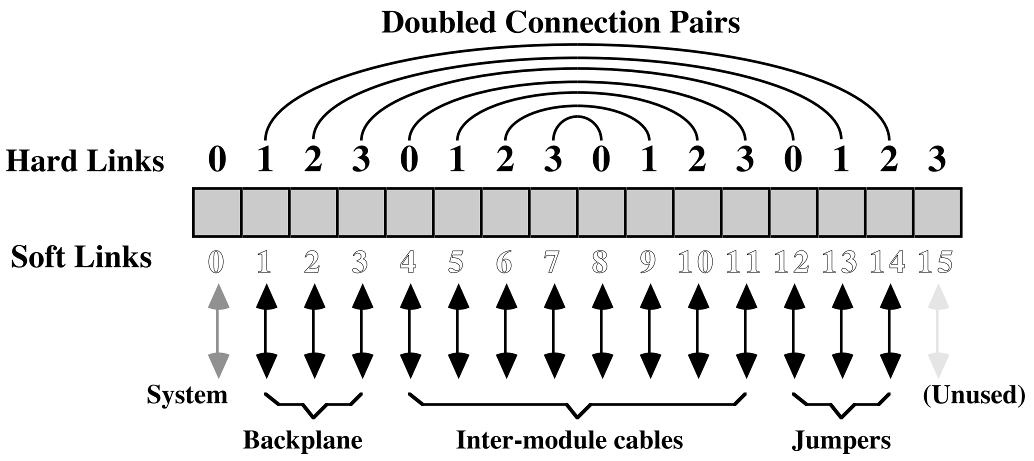

Figure 2. Link Doubling on the T Series

Soft link *j* is cabled the same as soft link 15–*j*. This means that hard link 0 is always doubled by hard link 3, and hard link 1 is always doubled by hard link 2. This is essential for one-dimensional interconnection subsets to have full use of all four links, since hard links 0 and 3 will always connect one side of every processing node to a neighbor and hard links 1 and 2 will always connect the other side. Since soft links 1 to 3 are actually connected across the backplane of a module, the easiest way to double them is to place a *jumper* across corresponding cable sockets for soft links 14 to 12. Schematically, such jumpers can be pictured as shown in Figure 3.

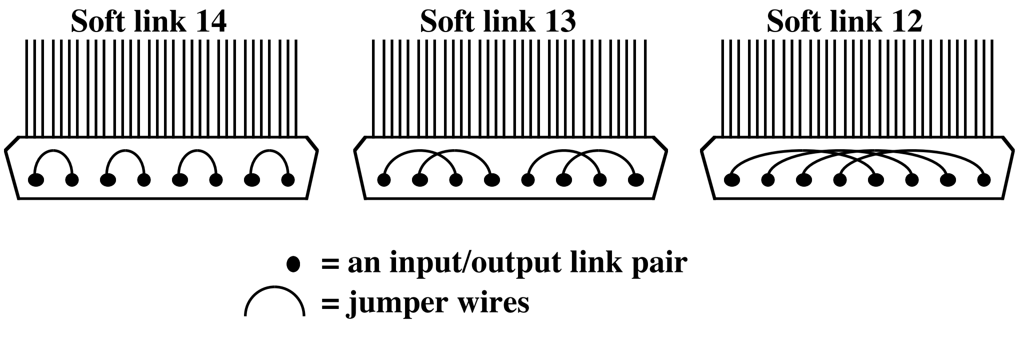

Figure 3. Jumpers to Duplicate Backplane Connections

For a 64-processor system, say, the additional hardware required is simply four cables and 12 jumpers, and insignificant cost compared to the rest of the system. Yet the effect is to double the performance of problems which are badly communication bound! The scheme works up to a 128-processor system, which is a large (and fairly expensive) configuration of the T Series. For larger configurations, one can gradually remove the doubling, resulting in loss of double bandwidth across certain paths resembling “fault lines” in the domain decomposition.

Example: 2D Fourier Transform

On a 16-processor system, the four hard links of a T Series node provide the two-dimensional point-to-point topology shown in Figure 4, where the hollow typeface denotes the soft link numbering.

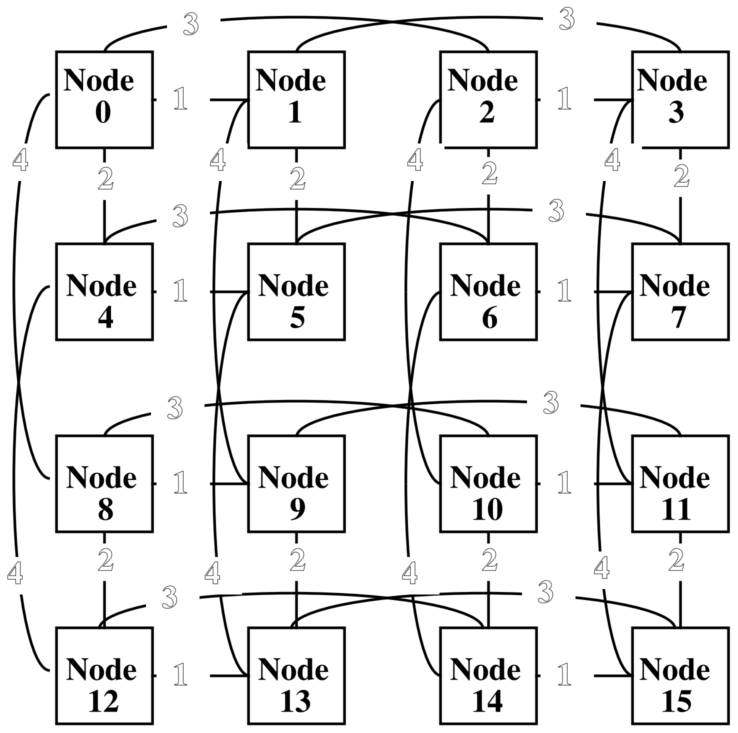

Figure 4. 16-Processor Topology for 2D FFT

However, when performing a radix 2 FFT, only *one dimension at a time* is exercised within the network; in fact, only one of the four possible directions out of a processor is needed to perform the data exchange for the “butterfly” computations which span processors. Now supposed that the application programmer recognizes that there are two paths between each processor pair; one could communicate the real data and the other path the imaginary data, for example. On one *x*-directional pass of the 2D FFT, soft link 12 is used to attach the same processors as soft link 3, and since they map to hard link 0 and hard link 3 respectively, they can operate simultaneously. On the other *x*-direction pass, soft link 14 supplements soft link 1 simultaneously.

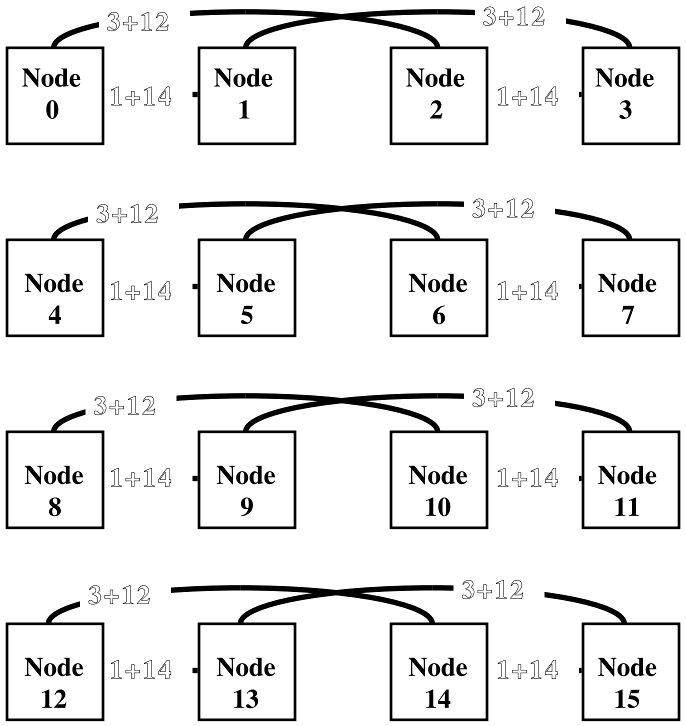

Figure 5. Doubled Links for x-Direction Passes of 2D FFT

Then the links must be reconfigured to perform the *y*-direction passes, but the amount of computation is large relative to the time to reconfigure the soft links.

Tests have shown this algorithm, without link doubling, to perform at about 12 MFLOPS on a 512×512 FFT, where each of the processors contains a 128×128 point subdomain of complex 64-bit numbers. Communication easily dominates the computation in this case. The link doubling is expected to increase the performance to about 20 MFLOPS.

Significance for Vector Processing

Several hypercubes offer or plan to offer some form of vector arithmetic capability. This affects the domain decomposition in a profound way, as illustrated in Figure 6.

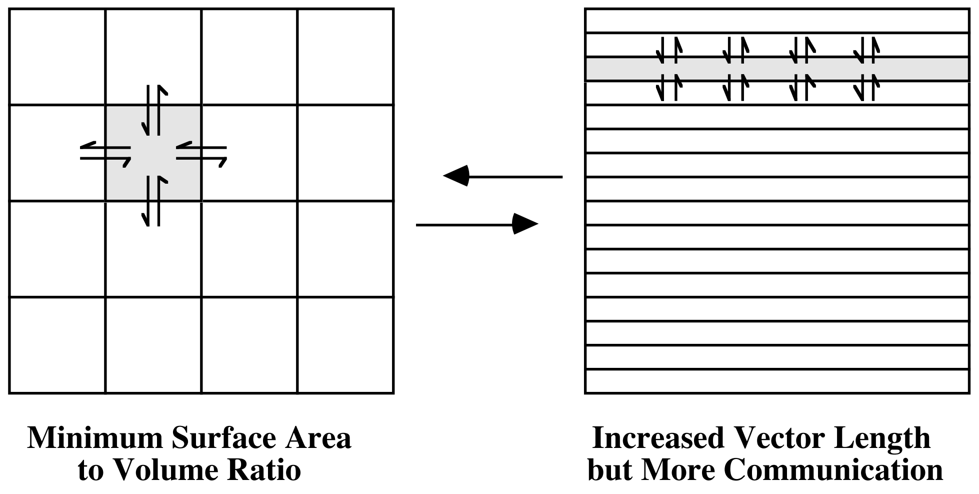
Minimum Surface Area to Volume Ratio
Increased Vector Length but More Communication

Figure 6. Domain Decomposition Tradeoffs

The Caltech paradigm is to minimize the ratio of surface area to volume to minimize the penalty for communication; this implies the use of squares or cubes as optimal subdivisions of 2-space and 3-space. However, this can make vectors short enough that the startup time for the vector operation reduces the performance. On a vector hypercube, one attempts to make the domains as long and thin as possible, with the optimal ratio being where the computation on interior points *just barely* overlaps communication of the exterior points.

With simultaneous link operation, the communication time is the *maximum* of the directions communicated, not the sum. In the example in Figure 6, the communication time will be increased four times by using the method on the right. However, the horizontal communication is eliminated.

The implication of link doubling for vector hypercubes is simply to reduce the penalty for such long vectors, permitting a higher fraction of peak theoretical speed to be attained without degradation by non-overlapped communication. In the case of the T Series, many two-dimensional problems appear to be optimally solved by making the subdomains long enough to span the entire rectangle and thus eliminating the need to communicate at all across the ends. With link doubling, the two-dimensional problem can be treated with a one-dimensional interconnect as in Figure 6, where each nearest-neighbor connection is twice as fast. A unidirectional transfer currently has an asymptotic speed of 0.691 MBytes/s on the T Series, with half speed achieved for transfers of length 26 bytes. With link doubling, the asymptotic speed on one-dimensional topologies is 1.382 MBytes/s, which brings the ratio of compute speed: link speed down to about 35 : 1 for 64-bit computations.

Generalization to *M*-Fold Increases

Suppose the existence of a hypercube node capable of N link communications, of which K can be simultaneous, and an application requiring a (K/M)-dimensional nearest-neighbor ensemble, where M divides K evenly and $1 < M \leq K$. Then a configuration of dimension (N/M) or less can be configured so that there are M links for every nearest-neighbor connection, effectively increasing communication speed M-fold in any *single* dimension at a time.

The idea of having to change cabling for every application is obviously not appealing, but that is not the intent here. Hypercubes with many links per node have *typical* configurations that use only a fraction of that maximum number of links. It appears quite practical, therefore, to replicate link connections to the maximum degree permitted by a configuration; cabling changes are made when the hypercube is upgraded to more processors, not from one application to another.

In the case of the T Series, $N = 14$ and $K = 4$, and implying $M = 2$ or $M = 4$. The latter case is restricted to the case of a single module (8 processors), but in fact, one could cable to allow 2.7 MByte/s coupling as shown in Figure 7.

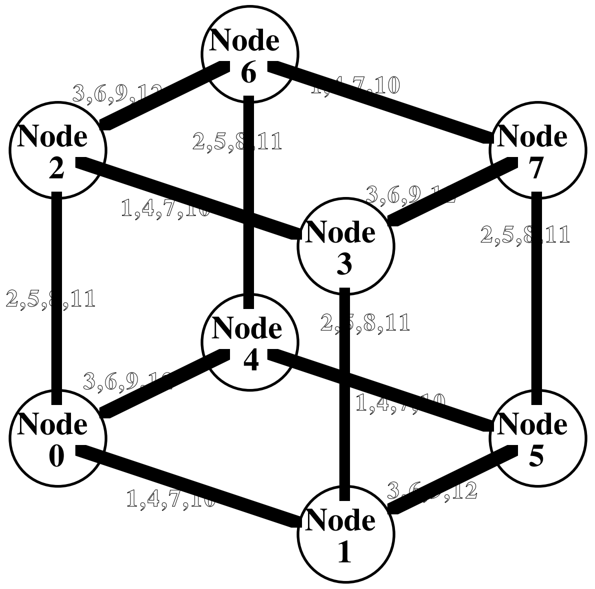

Figure 7. Link Quadrupling on a 3-Cube

Conclusions

The technique of link doubling on hypercubes allows one to recoup one of the features of a bus architecture: smaller configurations can have larger communication bandwidths. Note that this goes completely against the theology of ensemble architectures, since link doubling does not scale! It is difficult to resist such increases in bandwidth, however, since the impact on software appears to be so minor and the additional hardware is very inexpensive compared to the rest of the system. Link doubling could become a standard technique for performance improvement on any commercial hypercube that is available in a wide range of sizes. ■