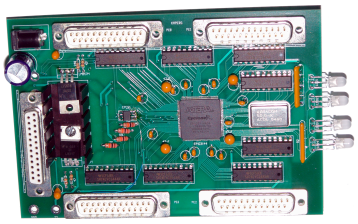
Come Together Right Now Over Me

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 multiple cores on the same chip. Although commodity hardware and straightforward topologies are sometimes effective, communications within parallel programs tend to have specific properties that allow a well-engineered network to dramatically outperform the obvious alternatives.

Aggregate Functions. In parallel computing systems, it is very common that the global state of a computation must be sampled – which is not an efficient operation when synthesized as "collective communications" using point-to-point network hardware. In 1994, we invented AGGREGATE FUNCTION NETWORKS (AFNs) as an extension of the fast barrier synchronization hardware we had developed earlier. An AFN doesn't route messages; rather, an AFN is really a simple parallel computer dedicated to computing functions of global state. A typical aggregate function communication is implemented by each processor placing its data and an opcode in its dedicated interface to the AFN and then reading the AFN-computed result back. Thus, many operations sampling global state can be implemented in *essentially constant time independent of the number of nodes*. Such aggregate functions include:

- Confirmation of hardware reliability
- Barrier synchronization
- VLIW multiway branch support
- SIMD any and all tests
- Broadcast & multicast
- PutGet (conflict-free reverse-routed messages)
- Reductions
- Scans (parallel prefix operations)
- Searches (first, count, & quantify)
- Voting & scheduling operations
- Ranking (sorting)
- Parallel signaling ("Eurekas")



Cluster AFNs. Using simple custom hardware, most basic aggregate operations are accomplished with just $3\mu s$ total latency. We have placed various AFN hardware designs and support software in the public domain. The simplest design is WAPERS (Wired-and Adapter for Parallel Execution and Repid Synchronization), which uses wired-AND logic implemented by wiring parallel ports together without any active components. High-performance cluster AFNs, such as the 2006 KAPERS (Kentucky's Adapter for Parallel Execution and Rapid Synchronization) AFN shown above, can be implemented for less than $25/node.

Although we have been building high-performance AFN hardware since 1994, significant improvements continue to be made, and the 2006 KAPERS AFN incorporates a number of firsts. In addition to the basic functions, this AFN supports a very general form of shared memory. The memory is nybble-oriented, but efficiently supports various operations on data objects of any multiple of 4 bits in length. This AFN also supports reduce multiply operations of any precision, implementing multiplication using addition of values represented in a logarithmic number system.

On-Chip AFNs For Multi-Core Processors. As multi-core processors have become common, there has been much talk of scaling to huge numbers of cores on a single chip. However, shared memory communication does not scale well to large numbers of cores due to a combination of competition for shared resources and the overhead of dynamic arbitration. By tightly integrating an AFN on chip, an alternative, more efficient, path is provided for coordination and communication.

Although such tight integration always had been our goal for AFNs, it is only over the past year that we have become focussed on determining precisely how an on-chip AFN would work in a modern multi-core processor. Working with Sam Midkiff at Purdue, Soohong Kim has defined a detailed structure for a CMP-AFN to be integrated with IA32 cores. This cycle-accurate AFN model is incorporated in a multi-core simulation implemented using Ruby, and performance was measured substituting AFN operations for the methods normally used by OpenMP. Early results are very encouraging; an OpenMP barrier synchronization across 4 cores obtains more than 6X speedup, while 8 and 16 core systems achieve 11X and 12X respectively.

Perhaps the most difficult issue for the CMP-AFN is the fact that multi-core programming models like OpenMP do not directly match the number of active threads to the number of cores, neither do operating systems like Linux force every process to be bound to the same core throughout its lifetime (i.e., processes can migrate across cores). Thus, the CMP-AFN incorporates new mechanisms that virtualize groups and allow more than one group to be simultaneously active.

This document should be cited as:

```
@techreport{sc07afn,
author={Henry Dietz, William Dieter, and Sam Midkiff},
title={Come Together Right Now Over Me},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc07afn.pdf},
month={Nov}, year={2007}}
```


Aggregate.Org UNBRIDLED COMPUTING

SC07