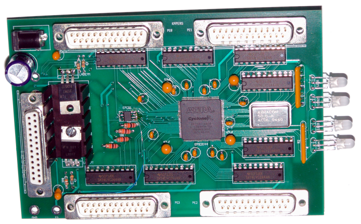
Come Together Right Now Over Me

INTERCONNECTION NETWORKS, sometimes called SYSTEM AREA NETWORKS (SANs), play a critical role in all types of parallel computers – be they clusters spanning many racks, multiple cores on a processor chip, or a massively-parallel GPU. 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*. The AFAPI (Aggregate Function Application Program Interface) includes:

- 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. With Sam Midkiff at Purdue, simulation of a detailed structure for a CMP-AFN to be integrated with IA32 cores sped-up OpenMP barrier synchronization by 6X on 4 cores and 12X on 16 cores.

AFN concepts for GPUs. To hide memory latency, Graphics Processing Units (GPUs) sacrifice many of the timing properties of traditional SIMD, making the obvious implementation methods problematic. We are working on efficient hardware and software implementations of the AFAPI. The hardware changes generally require redesign of GPU chips, but significant improvements can be made by software leveraging some of the more obscure properties of existing GPU hardware.

For example, AFAPI operations can be implemented on compute capability 1.0 NVIDA CUDA systems – which do not have any of the "atomic operation" hardware support present in later models – by using the fact that multiple "simultaneous" maskable stores to the same memory cell are processed efficiently. AFAPI's `p_any(flag)` operation within a block can be coded as:

```
if (flag) sharedtemp = serial; /* maskable store */
__syncthreads();
p_any = (sharedtemp == (serial++));
```

This document should be cited as:

```
@techreport{sc08afn,
author={Henry Dietz and Diego Rivera},
title={Come Together Right Now Over Me},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc08afn.pdf},
month={Nov}, year={2008}}
```

Aggregate.Org
UNBRIDLED COMPUTING

SC08 AUSTIN, TX