All Together Now

When we built the world's first Linux PC cluster supercomputer in February 1994, we did it to test the little box shown above. That box, the first PURDUE'S ADAPTER FOR PARALLEL EXECUTION AND RAPID SYNCHRONIZATION (PAPERS) unit, implemented a new communication model that we call AGGREGATE FUNCTION COMMUNICATION. This model, perhaps best described by the AGGREGATE FUNCTION APPLICATION PROGRAM INTERFACE (AFAPI) specification, centers on **N-way communications** like:

- 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")

We have built over two dozen types of custom AGGREGATE FUNCTION NETWORK (AFN) hardware for clusters since 1994. However, our work in this area now centers on ***how to make these operations efficient on all machines***... from chip multiprocessors to GRAPHICS PROCESSING UNITS (GPUS).

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. 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 [1].

AFN concepts for GPUs. To improve scalability, GPUs sacrifice many of the timing properties of traditional SIMD, making the obvious implementation methods problematic. Despite that, by 2009 we had a complete set of efficient primitives for all compute capability levels of CUDA devices [2,3]. For example, here's the logic for `p_any(flag)` within a block on CUDA with any compute capability level:

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

Here is the basic algorithm for barrier synchronization across blocks, with *O(1)* time when blocks have significant time skew:

```
__syncthreads(); /* Sync within each Block */
/* Pick a representative from each block */
if (threadIdx.x == 0) {
  /* Get my barrier number */
  int barno = barnos[blockIdx.x] + 1;
  int hisbarno;
  int who = (blockIdx.x + 1) % gridDim.x;
  /* Check in at barrier */
  barnos[blockIdx.x] = barno;
  do { /* Scan for all here or somebody passed */
    /* Wait for who */
    do {
      hisbarno = barnos[who];
    } while (hisbarno < barno);
    /* Bump to next who */
    if (++who >= gridDim.x) who = 0;
  } while ((hisbarno == barno) && (who != blockIdx.x));
  /* Tell others we are all here */
  barnos[blockIdx.x] = barno + 1;
}
__syncthreads(); /* Sync within each Block */
```

There are a number of variants on most algorithms. Throughout 2010, we have been developing versions using fully portable OPENCL. Algorithms like barrier synchronization have been ported and tested in OPENCL on both NVIDIA and AMD/ATI targets. All are/will be freely available via `Aggregate.Org`.

References:

[1] S. P. Kim, *Chip Multiprocessors with On-Chip Aggregate Function Network*, `http://docs.lib.purdue.edu/dissertations/AAI3379419`, 2009

[2] B. D. Young, *MPI Within A GPU*, `http://hdl.handle.net/10225/1085`, July 2009

[3] D. A. Rivera-Polanco, *Collective Communication and Barrier Synchronization on NVIDIA CUDA GPU*, `http://hdl.handle.net/10225/1158`, Sept. 2009

This document should be cited as:

```
@techreport{sc10afapi,
author={Henry Dietz and Frank Roberts},
title={All Together Now},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc10afapi.pdf},
month={Nov}, year={2010}}
```

Aggregate.Org UNBRIDLED COMPUTING

SC10 New Orleans, LA