

Mike Houston, Kayvon Fatahalian, Jeremy Sugerman, Ian Buck, and Pat Hanrahan

## SPIRE

### An Infiniband Interconnected Graphics Cluster

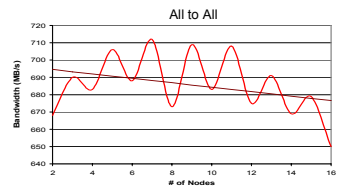
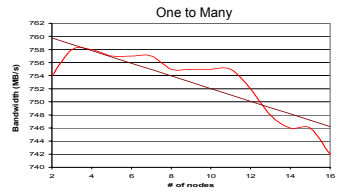
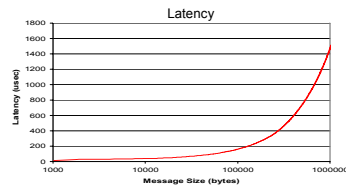
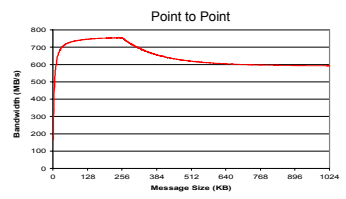
<http://spire.stanford.edu>

#### System Summary

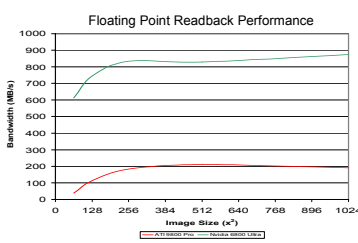
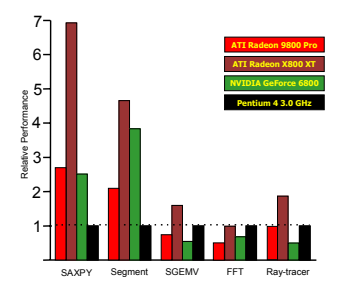
- 16 node cluster
- 32 2.4GHz P4 Xeons
- 16GB DDR
- 1.2TB IDE storage
- Mellanox 16 Infiniband switch
- Dlink 24-port GigE switch
- Linux – Fedora Core 2



#### Infiniband Performance



#### Single GPU Performance



#### Future Cluster Plans

- Dual/Quad AMD Opteron
- PCI Express
- Multiple graphics boards
- Infiniband 4X/12X
- Infiniband attached storage
- ATI X800XT PE PCIe or Nvidia 6800 Ultra PCIe

Prototype in development with GraphStream:



#### Peak Bandwidth

- 755MB/s
- 256KB message size

#### Cross sectional bandwidth:

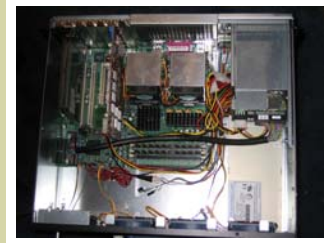
- 10.8 GB/s

#### Latency:

- 6usec for small messages
- Linear scaling for large messages

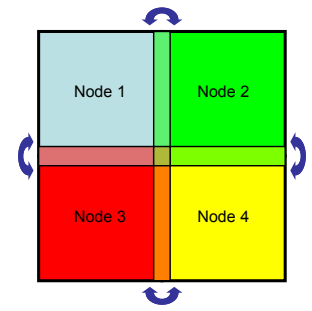
#### Scalability

- Shown to scale to >128 nodes
- Example: "Big Mac" cluster



## Brook and MPI

2D Block Decomposition



Brook is an open source programming environment for streaming computation on GPUs. For more information and downloads, see: <http://graphics.stanford.edu/projects/brookgpu>

Computation on GPUs has been shown to provide competitive performance compared to optimized code running on Intel P4 processors. Using Brook, a user can rewrite their computation loops as kernels, and execute their core computation on the GPU. However, available onboard memory limits the size of computational problems GPUs can efficiently tackle.

Traditionally, these resource constraints have been solved by distributing computation across a cluster using systems like MPI. MPI programs use different forms of data decomposition to accomplish their compute tasks across a cluster. As a simple example, 2D image processing can be distributed across a cluster using block-block decomposition (as shown to the right). Each node is responsible for a subsection of the data, and during the computation, overlapping regions on different nodes need to be updated/synchronized after a certain number of iterations.

Although the computation in an MPI application can be ported to GPUs using Brook, efficiently dealing with the region updates becomes important to scalability and application performance. Getting data to and from the GPU is a very expensive operation, so we want to limit the amount of data we push across the bus. We have added *stream domain* operators to Brook to allow the programmer to specify subsections of a stream to read and write, enabling the Brook runtime system to handle data transfers as efficiently as possible.

```
// Update overlapping region pseudo code
for(int i = 0; i < num_iterations; i++){
  // Send data off to neighbors
  for(int j = 0; j < num_neighbors; j++){
    // Get data from a section of the stream (data<y,x>)
    StreamRead( data.domain( float2( start_x[j], start_y[j] ),
                                float2( end_x[j], end_y[j] ) ),
              data2send);
    MPI_Send(data2send, ...); // Send data to neighbors
  }
  // Get data from neighbors
  for(int k = 0; k < num_neighbors; k++){
    MPI_Recv(updatedata, ...);
    // Update a section of the stream (data<y,x>)
    StreamRead( data.domain( float2( start_x[k], start_y[k] ),
                                float2( end_x[k], end_y[k] ) ),
              updatedata);
  }
  // Execute kernel over updated stream
  myKernel(data);
}
```

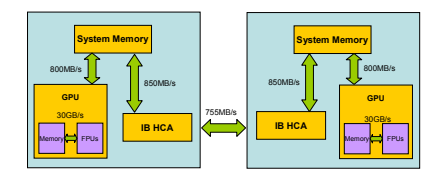
## Automatically Distributing Streaming Computation

As can be seen from the MPI example above, parallelizing an application across a cluster often centers around handling data distribution and communication. For the best performance, a user must map their application onto the cluster's memory and communication hierarchy efficiently. The figure to the right shows a simplified version of these hierarchies on our graphics cluster, SPIRE, running with Nvidia Geforce 6800 Ultras.

To take full advantage of the GPU, the application should constrain memory access to the fast local memory of the GPU as much as possible. Going off chip to the node's system memory is an order of magnitude slower. Accessing memory on another node incurs multiple bus transfers, causing further slowdowns. The last level of the hierarchy is accessing a remote node's GPU memory. This becomes problematic for scalability since recent GPUs can calculate at >60GFlops, and yet are constrained by slow access to system memory, and even slower remote memory access. It becomes difficult to keep the GPUs fed with data, and the memory access patterns must be carefully programmed to achieve good performance.

If the programmer has knowledge of both the locality in the application and the topology of the cluster, he can carefully write his MPI code to take full advantage of the computational power and memory subsystems of the graphics cluster. The difficulty in automatically distributing streaming computation is defining language constructs to help the compiler and runtime environment make these decisions.

We will be researching using a combination of language additions and features, compiler analysis, and runtime support to enable Brook applications to run efficiently across clusters of GPUs.



#### Language additions:

- Stream Affinity – define streams that will be accessed together
- Stream Domain – define a subset of a larger stream to access
- Stream Group – define access to multiple substreams
- Stream Stencil – define the access neighborhood of a stream element

#### Runtime support:

- Cluster definition file – provide information about memory hierarchies and interconnect/bus performance.
- Heuristics – use configuration file to perform data distribution based on cluster characteristics

