Outline

- Programming models
- Partitioning in Streams
- Partitioning information
  - Includes irregular grid support
- Open Issues
  - How is node-parallelism expressed?
  - What should be automatic?
  - What else must we have to support partitioning
Programming model

How is multi-node parallelism expressed?

- **MPI**
  - Stream programs are run independently on the nodes
  - Nodes synchronize and transfer data using MPI calls

- **Stream parallelism**
  - Streams represent shared data
  - Streams are partitioned over the entire machine
  - Kernels are run on all nodes (according to the data partition)
MPI

“StreamC” + MPI

- Similar to Imagine’s programming model
- Programmer partitions the data and computation
  - Same as for supercomputers today

**Pros**
- We know how to do it
- Will run legacy code

**Cons**
- Cumbersome
- No abstraction
- Not optimal
Stream node-parallelism

- Serial code + parallel kernels
  - The same serial code is run on all nodes
    - Serial data is replicated on all nodes
    - Writes are committed from a single master or owner node
  - Kernels run in parallel on all nodes
    - Streams are partitioned across the nodes
    - Shared data in memory is also partitioned
- Blind partitioning
  - Each stream (or shared memory) is split into $N_{nodes}$ contiguous parts of size $(Stream \ Length \ / \ N_{nodes})$

Should we also express “scalar” multi-node parallelism?
Stream Partitioning - preliminaries

- Streams are not memory
  - Streams represent transient data
    - Dependencies between kernels
    - Stream properties (shape, stencils,...)
  - Persistent data is in memory
    - `MemBuffer` type represents shared data
    - Uses a Brook allocator
      - Specifies size
      - Specifies memory layout (segment registers)
  - StreamLoad / StreamStore transfer data between Streams and MemBuffers

- SVM Functions
  - Special functions in Brook (like kernels)
  - Coded in SVM language and not Brook
    - Like `asm()` in C
  - Have access to all SVM parameters
Stream Partitioning

- Load data into MemBuffer
  - Data is blindly spread across the nodes
- Calculate structures for partitioning
  - Adjacency matrix ...
- Call a SVM_function for partitioning
  - User supplied or from a library
  - The partition function modifies the data layout (moves data around)
- StreamLoad reads the data into a stream
  - The stream “inherits” the partition from the MemBuffer
- Can be done dynamically as well
struct Node : public AbstractData {
    node data;
    neighbor list;
};

#define N num_of_data_nodes;

main() {
    MemBuffer Node data;
    stream Node s_data;
    int[N][N] adj_matrix;
...
    LoadDataFromFile(file, data);
    kernel_CalcAdj(adj_matrix, data, N);
    SVM_AdjPart(data, adj_matrix);
    streamLoad(s_data, data);
    kernel_Compute(s_result, s_data);
...
}

void SVM_AdjPart(MemBuffer AbstractData,
                 int** adj_matrix,
                 int n_elements) {
    int part[n_elements];

    MetisPart(part, adj_matrix, NUM_NODES,
              other parameters);

    // part contains the “partition
    // permutation”

    data = data[part];

    return;
}

This assumes equal number of elements
per node
Automatic Partitioning

- Express partitioning information in Brook
  - Regular grids
  - Neighbor lists (adjacency matrices)
  - Spatial coordinates
  - Acceleration structures?
Regular Grids

- Regular grids are already in Brook
  - Shape and stencil information define the neighbors
- Not enough for partitioning
  - streamFlo for example uses a long stream as the multi-grid data structure
Irregular grids

- Complex Brook data type: \textit{AdjMatrix} - \texttt{(int[][][])}
  - defined when giving a shape to a stream
- \textit{streamStencilAdj} operator
  - Like stencil but based on the neighbor list

\begin{center}
\begin{tikzpicture}
  \node[circle,fill=blue!20] (A) at (0,1) {A};
  \node[circle,fill=blue!20] (B) at (1,2) {B};
  \node[circle,fill=blue!20] (C) at (2,1) {C};
  \node[circle,fill=blue!20] (D) at (1,0) {D};
  \node[circle,fill=blue!20] (E) at (-1,0) {E};
  \node[circle,fill=blue!20] (F) at (-1,1) {F};
  \draw (A) -- (B);
  \draw (B) -- (C);
  \draw (C) -- (D);
  \draw (D) -- (E);
  \draw (E) -- (F);
  \draw (F) -- (A);
\end{tikzpicture}
\end{center}

\texttt{streamGroupAdj? (ABEF)(CD)}
Spatial Coordinates

- Complex Brook type: $Coord - \text{(int[])}$
  - Associate with every stream element
    - Can be done in shape or as part of the data structure
- Acceleration structures
  - Octrees, Grids (StreamMD)
  - Usually based on spatial coordinates
“Conclusions”

- MPI will work but not the way to go
  - Too cumbersome
  - Doesn’t take advantage of shared-memory
  - Can’t utilize low-level knowledge
- Blind will work but not efficiently
  - Need smart partitioning (even if just for stripmining)
- SVM partition function approach should work
  - Programmer must still do most of the work
  - Low level details are exposed
Issues

- How do we make things better?
  - Automatic partitioning
- Present an abstract interface to several predefined partition algorithms
- How do we deal with problem-specific partitioning
- Does this handle the *algorithmic partitioning*?
- Do we need scalar multi-node parallelism?