Graph Streams

Bill Dally
October 10, 2001

1 Summary

This memo explores some possible ways to describe and implement graph streams - streams whose neighbors are represented by an arbitrary graph structure. General graph streams are handled by pre-processing them into fixed (or bounded) degree graph streams. The resulting fixed-degree streams are then traversed to build constant-sized index streams. The index streams are dereferenced to load the neighbors into the SRF which allows a kernel to stream in each element and all of its neighbors in an aligned manner.

2 Fixed degree graphs

The case where each element of a stream has a fixed number of neighbors is easy to declare and is easily handled by a cache on the memory side of a stream register file. Consider for example a stream of type `foo`

```
Stream foo {
  ... payload ...
  foo *north
  foo *south
  foo *east
  foo *west
}
```

If we declare a kernel in 'brook' style as

```
kernel bar(foo x {north, south, east, west}, baz out y) {
  y = x.north->v + x.south->v + ...
}
```

We can execute this code on a stream processor by performing the following steps on each strip of stream x.

1. Run a kernel that traverses a strip of x and generates four output index streams one each for north, south, east, and west.

2. Perform four indexed stream load operations one on each of these index streams - note that this does mean that we make multiple copies of shared records in the SRF, but a cache would prevent multiple references to main memory[1].

3. Finally, run kernel bar rewritten to take five input streams - one each for x and its four neighbors.

This is equivalent to the approach John Owens takes in the Imagine implementation of RTSL where he splits a shader kernel at each texture reference. In effect step 1 is the portion of kernel bar before the reference - which generates the addresses - and step 3 is the portion after the references.

3 Variable degree graphs

A harder case is when we have a graph where the number of neighbors varies from element to element. For example:

```
Stream vardeg {
  ... payload ...
  int nr_neighbors ;
  vardeg *neighbor[] ;
}

kernel sumvardeg(vardeg x {neighbor[0:nr_neighbors]}, double out sum) {
  ...
  for(i=0;i<x.nr_neighbors;i++) y += x.neighbor[i]->v ;
  sum = y ;
}
```

We can convert this variable degree case to a fixed degree case by preprocessing vardeg to split elements with degree greater than some fixed threshold, say 4, into a main element and one or more continuation elements each of which has degree less than or equal to four. An auxiliary field is added to each element indicating whether or not it is a continuation. With this approach, the sequence of operations becomes.

1. Traverse stream `x` generating output stream `x_prime` where each output element with degree greater than four is split into multiple elements each with degree ≤ 4.

[1]To make effective use of the cache requires that we traverse the graph in strips smaller than the cache size that cover a partition of the graph with a minimum number of external edges. A preprocessor (graph partitioner) should compute the index streams for these strips before the main computation starts.

2. Now traverse `x_prime` and generate four index streams for the four neighbors of each element or continuation element - insert NULL addresses when the number is less than four. (This corresponds to step 1 above).

3. Perform four indexed stream loads to load the four index streams into the SRF. Note that the memory system must interpret a NULL not as an error but as something generating a NULL value. This can be accomplished without special hardware by using a special pointer to a NULL record as the NULL address. (This corresponds to step 2 above).

4. Finally run the kernel on five streams - `x_prime` and the four neighbor streams. This requires that the kernel have retained state so it can accumulate the partial sums across continuation elements. (This corresponds to step 3 above).