# Brook QuickSpec

I. Buck

7 December 2001

## 0.1 Stream Declaration

The stream modifier is used to specify a pointer as streaming. Follows the C declaration rules (pg 122 K&R). Dereferencing the contents of a stream pointer or array explicitly is not allowed. Stream functions are the only way to gain access to the values within a stream.

Declaration examples:

```
stream float s;
  Illegal.  s is not a pointer.

stream float *s;
  A stream of floats.

stream float s[100]
  A stream of 100 floats

stream float s[100][200];
  A streaming 2D array of floats.

stream float *s[100];
  An array of 100 streams of floats

stream float (*s)[100]
  A stream of float[100] arrays.

stream float (*s)[100][200];
  A 2D array of float streams.

stream float (s[100])[200];
  An array of 200 streams of float[100] arrays.
```

## 0.2 Stream Manipulation Functions/Operators

(Similar to IMAGINE StreamC)

```
s = t;
  s is a reference to stream t.

s = t(start, end);
  s is a reference to elements of t starting at offset start and up
  to, but not including, end.

s = t+u;

  s is a reference to t concatenated with u. Types of s, t, u must
  match.

s = t >> 1;
  s is a reference to t rotated right by 1 element.

s = t << 8;
  s is a reference to t rotated left by 8 elements.

stream <type> (*s) [2] = t*u;

  Returns a stream of <type>[2] which is a reference of all
  combinations of each element of t with each element of u.

stream <type> (*s) [2] = streamSelfproduct(t);
  Returns a stream of <type>[2] consisting of all combinations of
  each element of t with all other elements of t. No repeats: set of
  t[i],t[j] (i!=j)

stream <type> *s = streamCopy(t);
  s is a copy of the data in t.  s must have same type as t.

stream <type> s = streamGather(addrs);
  Returns a stream of elements of type by dereferencing a stream of
  memory addresses.

stream <type> s = streamGather(t, indicies);
  Returns a stream of references to the elements of t in the order
  given by the stream of indicies.

void streamScatter(s, addrs);
  Stores elements of stream s to the memory addresses contained in
  stream addrs.

stream <type> t = streamScatter(s, indicies);
  Returs a stream of references to the elements of s in the order
  specified by the stream indicies.
```

```
  stream <type> s = streamStride(t, 16);
    Returns a reference to every 16th element from t.

  stream <type> s = streamReverse(t);
    Returns a reference which is the reverse order of t.

  stream <type> s = streamBitReverse(t, 2);
    Returns a reference to a stream consisting of elements of t in
with bit 2 reversed order.

  int len = streamLength(s);
    Returns the number of element in s.
```

## 0.3 Stream functions

Used to manipulate stream elements. Runs on scalar processor (not parallel). Anything goes code. Arguments read/write.

Example:

```
  void printfloats (stream float *s) {
    printf ("%f\n", *s);
  }
```

## 0.4 Kernel functions

Kernel functions are special type of stream functions which have certain restrictions to ensure parallel execution.

Restrictions

1. No global memory reads or writes.
2. Arguments can be read only, write only, or reduce.
3. No stream function calls.

## 0.5 Kernel Arguments

Arguments considered read only unless we use one of the following keywords:

1. out: Specifies argument write only.
2. mout: Specifies multiple writes. Outputs must be explicitly issued using the **push** command which produces another element on this element stream.
3. reduce: Specifies reducable argument. Only reducable operators permitted.

Reducable operators:

```
+=  Sum
*=  Product
<?= Minimum
>?= Maximum
|=  Logical OR
&=  Logical AND
.=  Logical XOR
-=  Negative of sum
/=  Reciprocal of product
```

Return values of kernel function can be used to create new streams.

```
kernel float mykernel (stream float *s) {
  ...
}

stream float *t = mykernel(s);
```

## 0.6 Stencils

Some kernel need access to neighbors. To specify the range of neighbors use .. specifier in the argument list:

```
kernel void func (stream float s[-1..1]) {
  float a = s[-1];
  float b = s[0];
  float c = s[1];

}

Notes:
  [] is equal to [0..0]

  For access to the index for that stream element, prefix the range
  with a variable, i.e. [i:-1..1]
```

## 0.7 Reduction functions

These are user defined reduction functions for more control over the reduction tree. A reduction function must takes at least two arguments which are the two elements to reduce. Additional arguments cannot be input streams. The return value is the reduced value.

```
reduce stream <type> func (stream <type> *a, stream <type> *b, ...)
{
  ...
```

```
}

<type> *p = func(s);
```

Likewise the reduction operators, listed above, can be used on streams as predefined reduction functions.

```
stream float *t = mykernel(...);
float max >?= t;
```

## 0.8 Special Streams

**FileStream**: Just like a FILE type in C. Usage:

```
void loaddata (stream float *a, FileStream fp) {
  fscanf (fp, "%f", a);
}

loaddata(a, "myfile.txt");
```