Stream Programming Languages/Brook Tutorial

Lecture #9:	Thursday, 2 May 2002
Lecturer:	Prof. Bill Dally
Scribe:	Alex Solomatnikov and Jae-Wook Lee
Reviewer:	Mattan Erez

There is one handout for StreaMIT today.
Opinions about programming in Stream C/Kernel C:
1) Too many ”undocumented features”.
2) No constant folding. Need better C front-end.
Project proposal is due on Tuesday, May 7. Keep it short into the point.

1 What is a Stream Programming Language?

Stream programming language targets to achieve two goals at the same time - efficiency and convinience in writing a streaming application. Since communication overhead dominates over that of computation in a streaming program, a stream processor exposes the communcation explicitly into the upper layer so that software handles it to maximize performance.

There are some examples of stream programming languates that have been developed and/or are being developed.

- **StreamC/KernelC** A language specific to a single machine - Imagine at Stanford
- **StreaMIT** A language intended for the MIT RAW machine but not machine specific
- **Brook** 2^{nd} generation language based on the Imagine concept of streams, but machine-independent

2 Issues on Kernels

2.1 Implicit vs Explicit

In C code of FIR filter it is not easy to figure out what is the input stream and what is the output stream. On the other hand, position of elements within the stream should be specified explicitly. Here is C code:

```
for(i=0; i<MAX-FIRLEN; i++) {
    s = 0;
    for (j=0;j<FIRLEN;j++)
        s += a[i+FIRLEN-1-j]*h[j];
    b[i-FIRLEN+1] = s;
}
```

In Brook, input and output streams should be explicitly declared but an element within a stream is determined by context implicitly, but not pointed by an absolute index of position explicitly. Here is a kernel declaration in Brook.

```
kernel fir(floatsa[i:0,FIRLEN-1], float h[FIRLEN], out floats b)  {
    s = 0;
    for (j=0;j<FIRLEN;j++)
        s += a[FIRLEN-1-j]*h[j];

    b = s;
}
```

The index for stream `a, (FIRLEN-1-j),` is an offset from the current pointer of streams denoted by `i`, rather than one from the first element of the stream. This kernel is called by a caller function like this: `fir(a, h, b)`. Neither in the caller or in the callee function is an absolute value of the index.

The code below shows how a stream is derived in Brook using stencil.

```
typedef stream float floats;
typedef stream float floatws[FIRLEN];

floats a, b;
floatws aa;
streamSetLength(a, 1024); streamSetLength(b,1024);
streamStencil(aa, a, STREAM_STENCIL_CLAMP, 1, 0, FIRLEN-1);
```

From programmer point of view the kernel is applied to stream of stencils.

2.2 Retained State vs Functional

StreamC/KernelC manually manages communication between clusters, which violates abstraction. It adopts ”retained state”, which keeps the values of variables across iteration boundary of kernel execution. On the other hand, Brook does not retain states of kernel execution. A good aspect of this scheme is that there is no data dependency so that we can exploit more data-level parallelism (DLP). In order to communicate across iterations, Brook uses ”reduction” variable. A reduction variable can take any type.

```
// sum pairs of input stream
// in Brook
// it is an old style Brook, though
kernel sumpair(floats a[i:-1,0], out floats b) {
    b = a + a[-1];
}
```

Figure 1: Access Across Input Stream in Brook

2.3 Access Across Input Streams

2.3.1 Brook

As shown in figure 1, kernel declaration in Brook specifies the range of data across input stream it needs.

How expensive is it to implement this mechanism? It should not be more expensive than any other way, because it tells a communication pattern to compiler, and the compiler should be clever enough to know how to implement this. Currently, Brook does not have a cycle-accurate ”isim” simulator, but ”idebug” to check output result only; so we don’t have to worry too much about the cost in doing the second assignment.

2.3.2 KernelC

In KernelC it is pretty ugly to access stream elements in a neighboring cluster; especially, the cluster at the end should get the variable back in time across the machine from the other side to feed in its functional unit.

2.3.3 StreaMIT

StreaMIT is based on JAVA, and its kernel, or ”filter” in StreaMIT terminology, is defined as a class. Each kernel has two essential member functions; *init* for initialization and *work* for main kernel operation.

In StreaMIT, an arbitrary element in a stream can be accessed using peek(); it retrieves the element of a given offset from the current stream pointer. In figure 2, the first input.peak(6) returns ’g’, and does not move the current pointer. On the other hand, input.pop() returns ’a’ which is indicated by the current pointer, and increments the value of the pointer.

2.4 Access To Global Data

In designing a kernel of FIR filter in KernelC, the kernel probably need to read in a coefficient stream first before taking a input data stream, which corresponds to *init* function in StreaMIT. In Brook, you can declare global variables for the table of coeffiecients so that a kernel can access them for table lookup.

Then values of the table can be written in a kernel? The answer is ”yes” for KernelC and ”no” for Brook - the global variables are read-only in the kernel of Brook. It is not clear how to implement it: register files - small capacity, local memory/scratchpad - small bandwidth.

If global data is too big, then the only option is global memory. But how can we hide memory access latency? Maybe, it's better to split into 2 kernels and generate intermidiate stream of indeces/addresses to amortize latency. In such case a lot of intermidiate data might need to be stored in other streams.

3 Issues on Streams

3.1 Stream Declarations and Derivations

Figure 3 shows how to declare a stream and derive one from an existing stream both in StreamC and Brook. All of the derived streams are not a copy of the original stream but a reference to it. However, StreamC has ”multiple of 8 problem”. In contrast, StreaMIT never sees a stream, because stream declaration is all implicit.

3.2 Communication Pattern

Connecting multiple kernels results in a communication pattern. In StreamC/KernelC as well as Brook, we get together different kernels freely as shown in figure 4.

In StreaMIT, kernel is a filter, basically; it takes only one input stream and generates only one output stream. You can do three things in connecting these filters.

- **Pipeline** one kernel follows another, or ”pipeline” them
- **Split/Join** takes one stream input and split it to feed multiple kernels and join their outputs into one stream
- **Feedback Loop** reverse positions of a splitter and a joiner

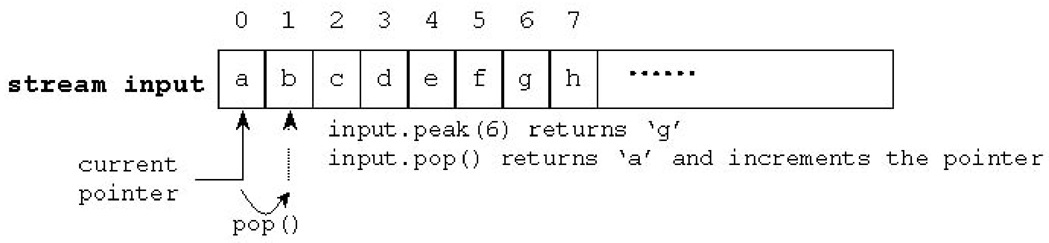
0 1 2 3 4 5 6 7
stream input
a b c d e f g h ......
input.peak(6) returns 'g'
input.pop() returns 'a' and increments the pointer
current pointer
pop()

Figure 2: Access Across Input Streams in StreaMIT

```
/* StreamC */
// a stream of 1024 "foo" records
im_stream x = newStreamData<foo>(1024);

// every third record from stream x
y = x(0, 1024, im_fixed, im_acc_stride, 3);

// these are "references"
// if you change y, x is changed as well


/* Brook */
typedef stream foo foos;
foos x,y;
streamSetSize(x, 1024);

streamStride(y, x, 1, 3); // y is "references"
```

Figure 3: Stream Declations and Derivations

4 Brook

4.1 What is the purpose of Brook?

Brook is designed to achieve the following goals:

- **Machine Independent**
- **More Suitable for Parallel Implementation** ”functional” kernel with no retained states; reduction mechanism that is converted to tree in log time

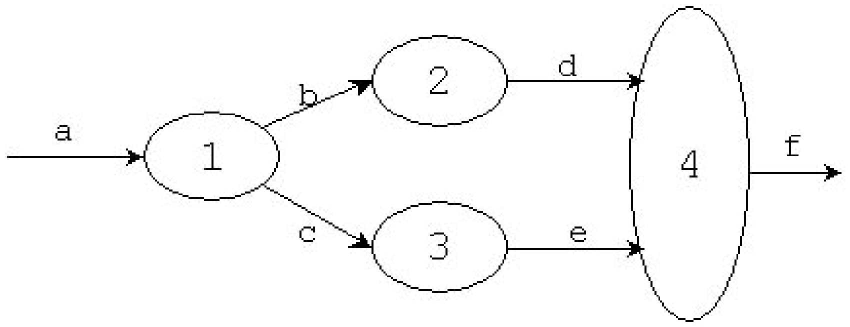

Figure 4: An Example for Communication between Kernels in Brook

- **Multidimensional Arrays** Stencil(figure 6) with decoration for boundary
- **Irregular Data Structures** Currently under development

Unlike StreamC/KernelC Brook does not allow conditional input streams, only conditional output.

4.2 2-D Array Access

An old version of Brook used streamShape() in order to access a 2-D array, which defines ”shape” of the stream as follows:

```
typedef stream float floats;
floats x[1024];
streamShape(x, 2, 32, 32);

kernel neighborAvg(floats a[x:-1,1][y:-1,1], out floats b) {
    int i, j;
    float s = 0;
    b = 0.25*(a[-1,0]+a[1,0]+a[0,-1]+a[0,1]);
}
```

A new version of Brook uses stencils to do the same thing:

```
typedef stream float floats;
typedef stream float floats2[3][3];
floats x;
floats2 y;
streamShape(x,2,32,32);
```

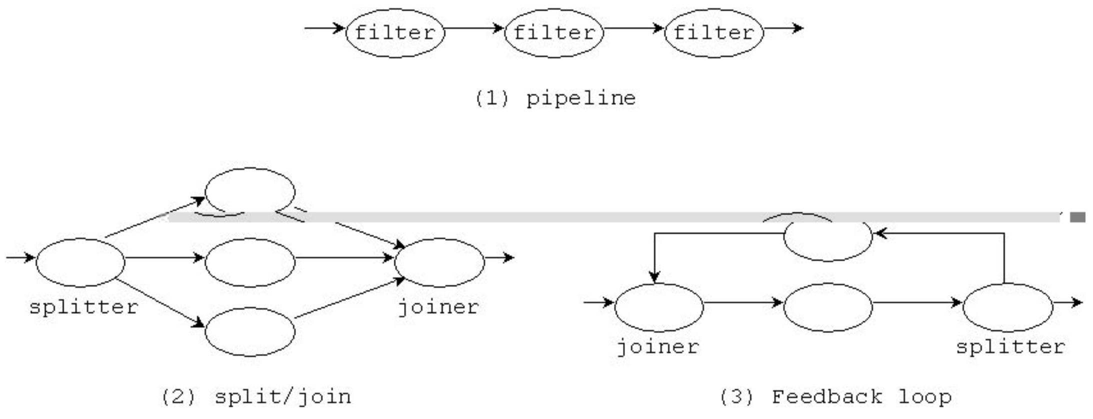

Figure 5: Elementary Communication Patterns in StreaMIT

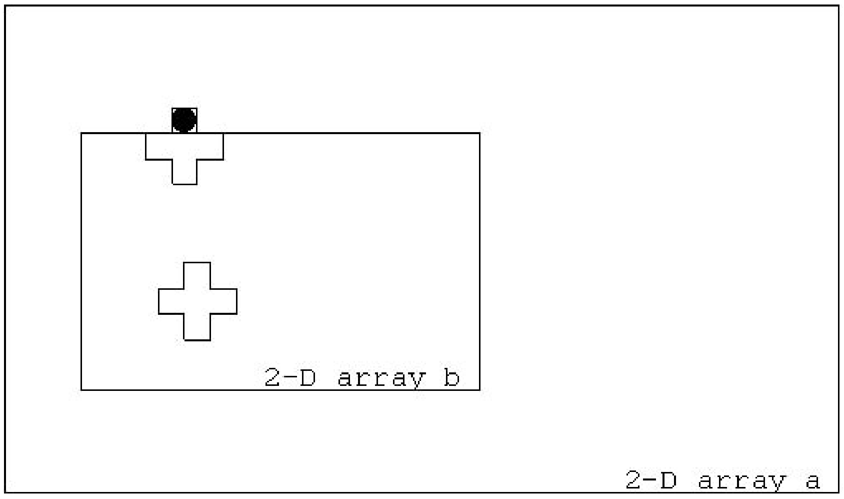

Figure 6: Stencil

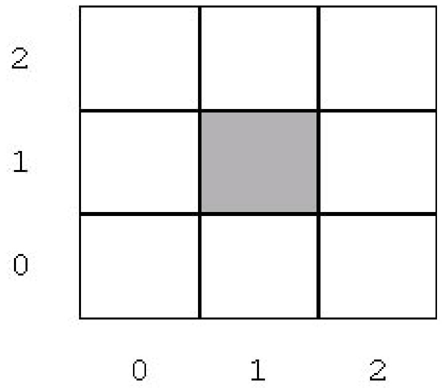

Figure 7: Stencil of a 3x3 Rectangle

```
streamStencil(y, x, STREAM_STENCIL_CLAMP, 2, 1, -1, 1, -1);

kernel void neighborAvg(floats2 a, out floats b) {
    b = 0.25*(a[0][1]+a[2][1]+a[1][0]+a[1][2]);
}
```

The stencil in the code above is defined by 3-by-3 array; currently, only rectangular shape of stencil is supported, and it yanks 9 elements centered around an original stream element, shaded in figure 7. In many cases, we only need 4 elements in the east, west, south, and north of a given element, but not those in the diagnal location. How we can manage the necessary data is a question.

In dealing with the elements beyond boundary of 2-D array like the shaded element in figure 6, Brook supports the following three ”decorations” for stencil. (Assume that the 2-D array b is derived from the array a in the figure.)

- **HALO** the array is a subset of a larger array which contains all of the elements in question
- **CLAMP** uses a specified value(e.g. 0) for the boundary values
- **PERIODIC** periodic boundary condition – like torus or donuts

It is possible to use variable as an index in the stencil.

4.3 Reduction

Reduction is another major mechanism to communicate between elements of a kernel. Unlike a stream which is either read-only or write-only, a reduction variable is both readable and writable. However, use of reduction variable assumes that computation is associative.

Restricting a stream to either read-only or write-only is critical to ease discovery of communication between streams.

4.4 Irregular Structures

To handle an irregular structure in Brook is still under development.

4.4.1 An Example

For example, let's think of the following problem – for each node in the graph summing the values of all of the neighboring nodes in each iteration; in this case, every vertex has a different number of neighboring vertices, and it takes a different amount of time to execute the kernel depending on the number of nodes, correspondingly.

One way to solve the problem is to generate a stream of indeces of neighbours from stream of nodes, then fetch the stream of neighbours from the memory and sum (slide 18).

Cleaner approach is to explicitly specify what you want and allow the compiler to decompose and optimize it (slide 19).

Figure 9 shows an examplar structure of the vertices and a possible data structure to handle this problem.

5 Summary and Open Questions

5.1 Summary

1) Communication restriction vs. ease of use.

2) No input-output streams are allowed to avoid dependencies/serilization. The only way to communicate between iterations is through reduction variables.

3) Handling complex structures.

```
struct node {
    float value;
    float old_value;
    int nr_neighbors;
    struct node *neighbors;
}

For each node, *node
    node->old_value = node->value;

For each node, *node
    node->value = 0;
    for each neighbor, *neighbor
        node->value += neighbor->old_value;
```

Figure 8: An Example of Irregular Structure Problem

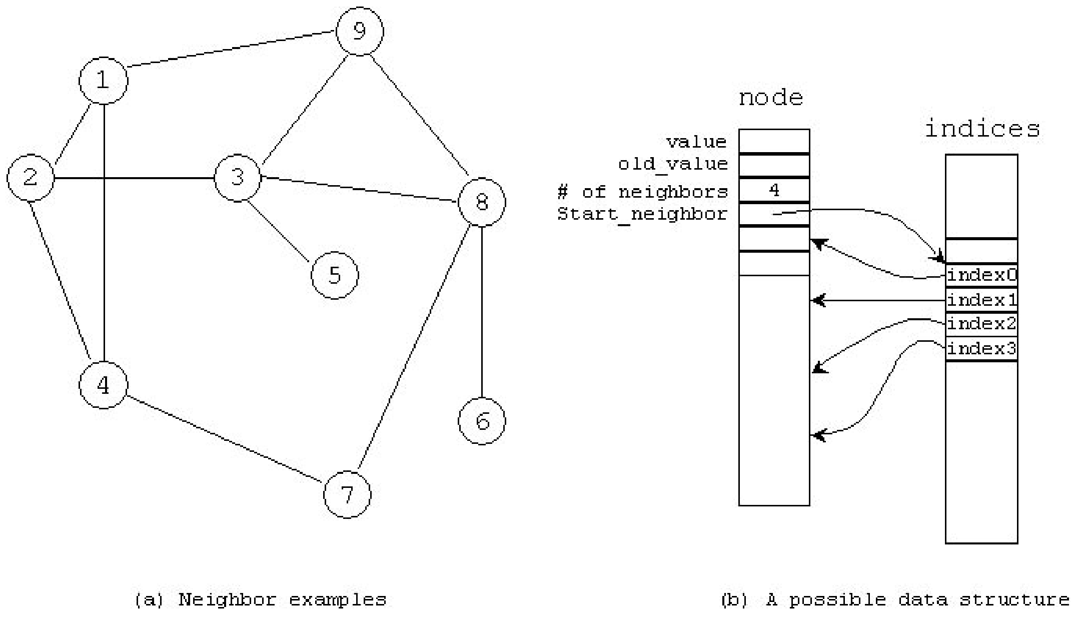

Figure 9: Irregular Structure of Vertices(a) and One Possible Data Structure(b)

5.2 Questions

Q. Is it possible to do graph algorithms in StreamC/KernelC?

A. Yes, first way shown in slide 18.

Q. Is cache useful in streaming computer? Where to place it?

A. Cache can be used between stream register file and main memory to reduce memory bandwidth requirements. Variable latency does not matter.

If separate cache is connected directly to each cluster, then miss in any of them will stall all clusters.

Q. What does StremProduct do?

A. StreamProduct(a, b, c) generates all combinations of elements of streams a and b:

```
       a    b    c    d

  p   a,p  b,p  c,p  d,p
  q   a,q  b,q  c,p  d,p
  r   a,r  b,r  c,p  d,p
```

StreamProduct has not been implemented yet. Efficient implementation should not generate whole product since it may be too large.

Q. Is Brook environment is the same as StreamC/KernelC?

A. No, Brook is implemented on Linux.

Q. Is there link between Brook and Imagine?

A. No, there is only converter from Brook to C, which allows to run Brook programs on standard PC/workstation.