GStream: A General-Purpose Data Streaming Framework on GPU Clusters

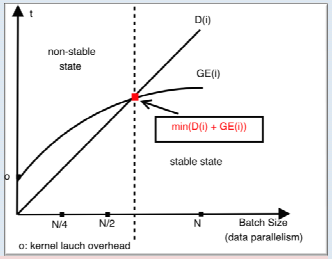
Yongpeng Zhang, Frank Mueller

NC State University

Motivation

- GPU's viability to operate on general streaming data is still unknown.
- Data streaming processing and data-parallelism are sometimes conflicting
  - — Stream processing favors smaller response time
  - — Massive data-parallelism tends to increase response time
  - — Existing streaming abstraction fails to consider this trade-off

• Suppose inputs arrive at a steady speed of N/T
• GE(i): GPU execution time for batch size i
• D(i): Response time for batch size i



The larger the batch size, the worse the average and maximum response time

- GPU cluster
  - — Handle different layers of memory

Design Goal

- Scalability
  - — No restriction on the size of the GPU cluster
- Transparency
  - — Task scheduling and GPU/host memory management handled by run-time .
- Extendability
  - — Easy to extend to customized need
- Programmability
  - — Syntax should be concise and provide compile time type-checking
- Flexibility
  - — Easy to switch b/w CPU and GPU execution
  - — Allows fast prototyping and debugging on CPU
- Reusability
  - — GPU kernels more expensive to develop
  - — Reusing existing CUDA libraries a plus

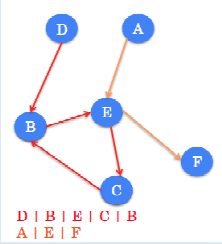
Gstream: System Model

- ***Filter***
  - — Encapsulate data processing; consume and/or produce data
  - — Main body of a filter execution is generalized into three-phase pattern:

```
void Filter::run() {
   start();
   while (!isDone())
      kernel();
   finish();
}
```

- ***Channel***
  - — One-way links b/w filters

- Operator "|" to concatenate filters using channel
  - — Concise, yet powerful to express complicated filter mapping



Gstream APIs

StreamSystem APIs:

void addFilter(FilterBase *filter)
void run();

Filter Functions to be Overridden:

void kernel() * (GPU kernels are launched inside)
void start() + (empty by default)
void finish() + (empty by default)
int getMinDegree(int portId)
int getMaxDegree(int portId)
*: must overridden; +: has default behavior

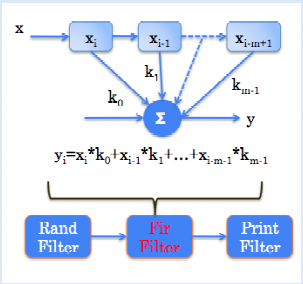
Channel Push APIs:

void reserve (StreamChannelBuffer &buffer, int size)

Channel Pop APIs:

void pop (StreamChannelBuffer &buffer, int min, int max)
void pop_finalize (int size)

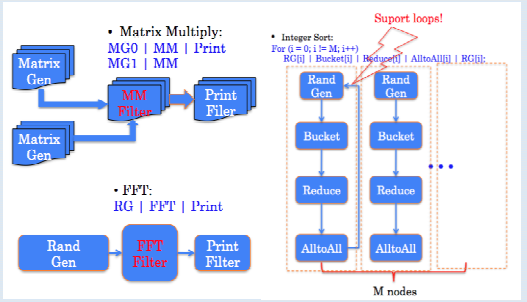
Case Study: Finite Impulse Response Filter




```
int main()
{
    StreamSystem ss;
    RandomFilter<float>rf;
    FirFilter<float, 100> firf;
    OutputFilter<float> pf;

    /* add filters to system */
    ss.addFilter(&rf);
    ss.addFilter(&firf);
    ss.addFilter(&pf);

    /* construct the channel*/
    rf | firf | pf;

    /* ready to run */
    ss.run();
    return 0;
}
```

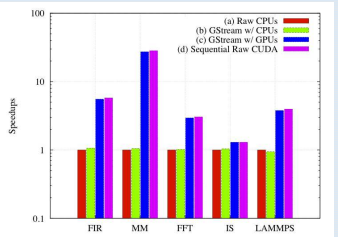
```
template<class T, int m>
class FifFilter: public Filter<typelist1<T>,
typelist1<T> > {
public:
virtual void start() {
    … /* setup coefficients array k[m] */
}
virtual int getMinDegreee(int portId) {
    return m;
}
virtual void kernel(){
    ChannelBuffer<T> input;
    ChannelBuffer<T> output;
    int batch = inPort[0]->pop(&input,
getMinDegree(0), getMaxDegree(0));
    if (batch != -1) {
        outPort[0]->reserve(&output, batch –m +
1);
        /* computation*/
        fir_kernel<<<…>>>(input, output, m);

        /* output ready, finalize the reserve*/
        outPort[0]->reserve_finalize();
        /* input port only consumes batch –m +1*/
        inPort[0]->pop_finalize(batch-m+1);
    }
    else { /* terminate condition */
        setDone();
    }
Private:
    float k[m];
};
```

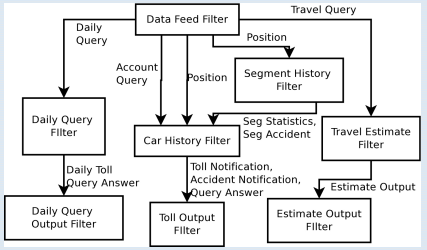
Experimental Results



- FIR: degree 100
- Matrix Multiply: matrix size 512 X 512
- FFT: 2D 512 X 512
- Integer Sort (IS)
- LAMMPS benchmark

- All running on a GPU cluster with 16 nodes
- Speedups up to 30X over CPU cluster with the same # of nodes





- Linear Road Benchmark: Original designed to provide scalable and fair benchmark for Stream Data Management Systems (SDMS)
- Performance measured by L-rating (# of express ways supported w/o breaking response time constraint)
- GStream achieves L= 40, in contrast to L=2.5 in Aurora and SPC

Conclusion and Future Work

- GStream is a general-purpose, scalable data streaming framework designed for GPU clusters
- We present a novel and concise, yet powerful streaming abstraction amenable to GPUs
- Gstream is easy to use, applicable to a variety of domains not constrained to traditional streaming problems
- Our future work includes:
  - Expand GStream to NAS benchmarks, making GPU cluster an attractive platform for high-performance computing