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

Yongpeng Zhang & Frank Mueller

---

**Motivation**
- GPU’s viability to operate on general streaming data is still unknown.
- Data streaming processing and data-parallelism are sometimes conflicting.
  - Streaming processing favors smaller response time – less communication.
  - Massive data-parallelism tends to increase response time.
  - Existing streaming abstraction fails to consider this trade-off.

- Suppose input tuples arrive at a steady speed of N/T.
  - GE(i): GPU execution time for batch size i.
  - Dj: 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 GPU and CPU 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-step pattern:
    - void Filter::run();
      - start();
      - while (!isDone())
        - kernel();
      - finish();

- **Channel**
  - One-way Links b/w filters.
  - Two types of channels:
    - p2p channel: links a predecessor and a successor filter.
    - group channel: links array of filters, well-defined communication pattern (broadcast, reduce, all-to-all etc.).
  - 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) { ...

- **Channel Push APIs**:
  - void pop (StreamChannelBuffer &buffer, int size)
  - void reserve (int size)

- **Channel Pop APIs**:
  - void pop (StreamChannelBuffer &buffer, int min, int max)
  - void pop_finalize (int size)

**Case Study: Finite Impulse Response Filter**

```cpp
template<class T, int m, int n class FifFilter: public Filter<Typelist1<T>, Typelist1<T> { > }
public:
  virtual void start() { ...
    ... /* setup coefficients array [m] */ ...
  }
  virtual int getMinDegree(int portId) { return m; }
  virtual void kernel()[
    StreamChannelBuffer<T> input;
    StreamChannelBuffer<T> output;
    int batch = inputPort[0]->pop() & inputPort[0]->getMaxDegree();
    if (batch == -1) {
      /* terminate condition */
      setDone();
    } else {
      /* must overridden; */
      output.buffer_finalize(batch-m+1);
    }
  }
private:
  float k[m];
};
```

---

**Experimental Results**
- **FIR**: degree 100
  - Matrix Multiply: matrix size 512 X 512
  - FFT: 20 512 X 512
  - Integer Sort (IS)
  - LAMMPS benchmark

- **GStream** achieve L=35, 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, adaptable 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.

---

**Experimental Setup**
- **Linear Road Benchmark**: Original designed to provide scalable and fair benchmark for Stream Data Management Systems (SDBM).
- **Performance measured by L-rating (# of express ways supported w/o breaking response time constraint)**
- **GStream** achieve L=35, in contrast to L=2.5 in Aurora and SPC.