**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
  - Stream processing favors smaller response time
  - Massive data-parallelism tends to increase response time
  - Existing streaming abstraction fails to consider this trade-off

**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();
      }
    }
    ``
  - Operator "$\odot$" to concatenate filters using channel
  - Concise, yet powerful to express complicated filter mapping

---

**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

---

**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_final(int size)

---

**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

---

**Case Study: Finite Impulse Response Filter**

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

---

**Operator "$\odot$" to concatenate filters using channel**

```c
int main() {
  StreamSystem ss;
  RandomFilter<float> F;
  F.filter<float, 100> f;
  OutputFilter<float> p;
  /* add filters to system */
  ss.addFilter(&f);
  ss.addFilter(&p);
  /* construct the channel */
  if (f) {
    /* computation */
    f.run<<<...>>>(input, output, m);
  }
  /* output ready, finalize the reserve */
  outPort[0] = reserve();
  /* input port only consumes batch = m + 1 */
  inPort[0] = pop_finalize(batch = m + 1);
  else /* terminate condition */
    clearDone();

  Private:
    float k[m];
}
```