# Compiler & Runtime Techniques for Adaptive Stream Processing

**Ryan Newton, Joel Svensson, Trevor McDonell,**
**Michael Vollmer, Ömer Sinan Ağacan, Buddhika Chamith**

**{rrnewton, joelsven, mcdonelt, vollmerm, oagacan, budkahaw}@indiana.edu**

## Introduction

Distributed stream processing requires a combination of technologies to monitor workloads and map them onto the local resources of worker machines. Here we highlight three sub-projects that address distinct aspects of this problem:

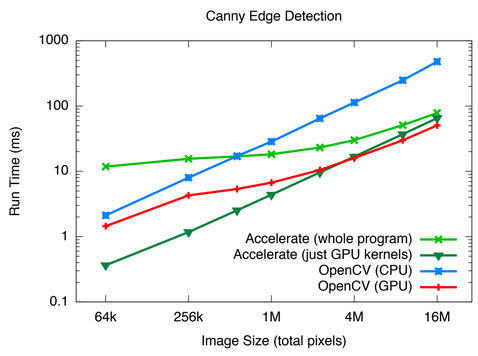
- ***Compile:*** JIT compiling dataflow graphs for available parallel architectures (Accelerate)
- ***Communicate:*** Sending irregular data efficiently between nodes (Compact Normal Form)
- ***Monitor:*** Profiling native-code programs based on binary self-modification and cross-modification (Ubiprof)

## Accelerate

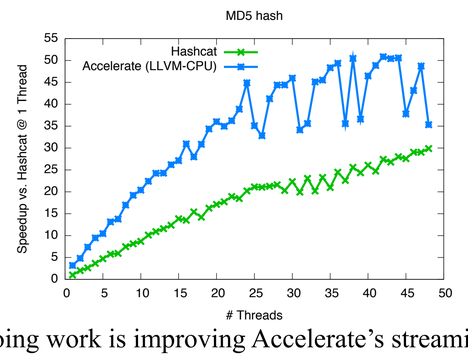
In a distributed execution plan, a subgraph of a stream dataflow graph must map onto the hardware of a worker node and achieve throughput. Our approach is a DSL JIT compiler called *Accelerate*.

- Accelerate [6] takes a graph of data-transformations and generates CUDA or LLVM code to run on CPU or GPU.
- Accelerate can launch concurrent GPU kernels on each new input (stream element) that arrives
- Accelerate is a (partially) formally verified compiler [4]

Currently, we apply Accelerate to streaming problems in the video domain:




Via it's LLVM backend, Accelerate can run data-processing algorithms on the CPU too




Ongoing work is improving Accelerate's streaming [7]. At IU, we are adding the ability to dynamically handle multiple devices, such as two GPUs [2], and add auto-tuning to the high-level compiler [3].

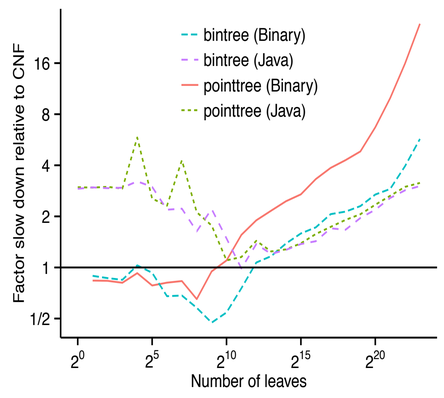
## Compact Normal Form (CNF)

Distributed stream processing systems (like Twitter Storm) must route streams over network links. Data (de)serialization consumes significant time, especially for irregular and pointer-based data structures.

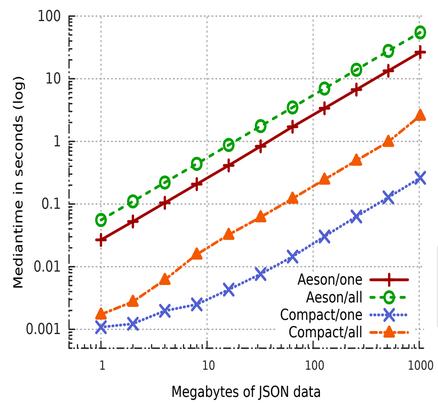
For immutable data in high-level languages, we explore an alternate heap representation: *Compact Normal Form* **[1]**. CNF allows regions of the heap to directly be:

- sent over the network (including RDMA)
- stored to disk
- skipped over by GC as *one* object

We have implemented CNF for the Glasgow Haskell Compiler (GHC). Our total speedup for sending, e.g., large binary trees through the network with CNF can exceed 16X:

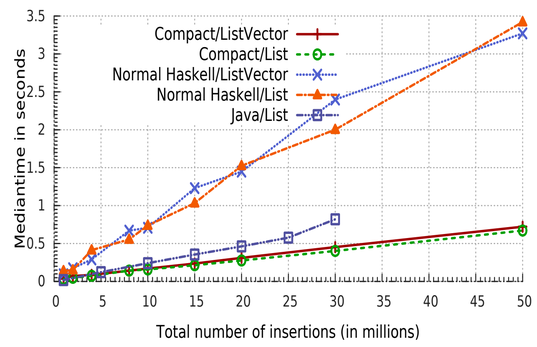



Likewise, reading twitter data from disk is faster when it can be mmap'd directly into GHC's heap, whether reading one record or all of them:




*Note: "Aeson" is a High-performance JSON-parsing library*

Finally, if inserting frequently into a large data structure, it being compact can actually help performance:




We are combining CNF with the HPX runtime (hpx.crest.iu.edu) for distributed execution.

## UbiProf

Profiling streaming dataflow graphs for scheduling purposes has different requirements than traditional time profiling.

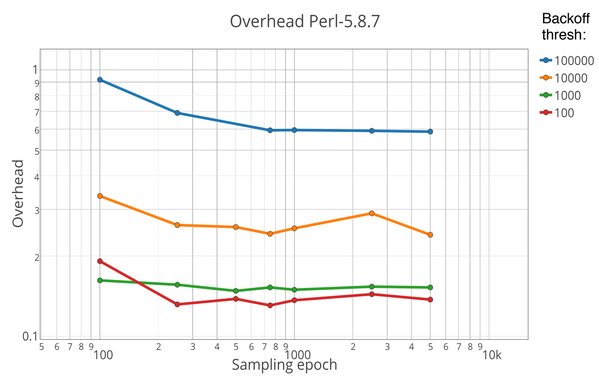

Capturing processing latency and data size requires sampling *intervals* rather than *instants*, and injecting custom code to compute data structure sizes. Thus dynamic probes, such as in DTrace, are appropriate.

Yet overheads for these probes are high. Thus we are exploring new approaches with an order of magnitude improvement in probe cost, and a probing infrastructure that is:

- Intra-process, user space
- Multicore scalable
- Free when not engaged

We have also used these probes to build a general profiling tool, UbiProf **[5]**. UbiProf dynamically activates and deactivates probes to stay under a given allowable overhed, using a *backoff* approach for hot functions to disable their own probes.

UbiProf controls a backoff threshold and the frequency with which deactivated probes are reactivated (sampling epoch).




Here, even on a workload with 9531 different functions and 18M calls/sec (Perl spec benchmark), it is possible to take up to 1000 samples per function per 100ms sampling epoch, for less than 2% overhead.

One consequence of capturing function start/end intervals is that UbiProf can flag functions with *multimodal* cost, which gprof cannot.

| | gzip | grep | bzip | perl | h264 |
|---|---|---|---|---|---|
| severe MM | 1 | 3 | 6 | 67 | 42 |
| called funs | 36 | 66 | 60 | 415 | 317 |
| gprof missed | 0 | 0 | 0 | 42 | 42 |

## Further information

PI Website: www.cs.indiana.edu/~rrnewton
Github: github.com/iu-parfunc
github.com/rrnewton
CREST: crest.iu.edu
PL@IU: lambda.soic.indiana.edu

## Paper links for phones

[1] Efficient Communication and Collection with Compact Normal Forms. *ICFP'15* tinyurl.com/compactnf

[2] Converting Data-Parallelism to Task-Parallelism by Rewrites, in submission. tinyurl.com/acc-multidev

[3] Meta-Programming and Auto-Tuning in the Search for High Perf. GPU Code, in sub. tinyurl.com/autotune-gpu

[4] Type-safe Runtime Code Generation: Accelerate to LLVM, in submission. tinyurl.com/acc-llvm

[5] Ubiprof: Towards Always-On Profiling of Native code, working draft. tinyurl.com/ubiprof

[6] Optimising Purely Functional GPU Programs, tinyurl.com/acc-optim

[7] Functional Array Streams, Madsen et al. FHPC 2015, tinyurl.com/acc-streaming

## Acknowledgments

This work is supported by NSF XPS award #1337242,
"XPS: DSD: Adaptive Stream-Processing Compilers for a Messy World"

INDIANA UNIVERSITY