

PHISH framework for Streaming Graph Algorithms

Steve Plimpton
Sandia National Labs

CERI/DIMACS Workshop on Streaming Graph Algorithms
Oct 2014 - Sandia

Collaborators: Tim Shead, Jon Berry, Cindy Phillips,
Todd Plantenga, Karl Anderson



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



PHISH is a simple library for stream processing

- Parallel Harness for Informatic Stream Hashing
 - phish swim in a stream
- Open source, BSD license

<http://www.sandia.gov/~sjplimp/phish.html>



PHISH is a simple library for stream processing

- Parallel Harness for Informatic Stream Hashing
 - phish swim in a stream
- Open source, BSD license

<http://www.sandia.gov/~sjplimp/phish.html>



- Commercial: IBM InfoSphere, Esper, SQLstream
- Open source: Twitter Storm, S4 (Yahoo!)
- Other ?

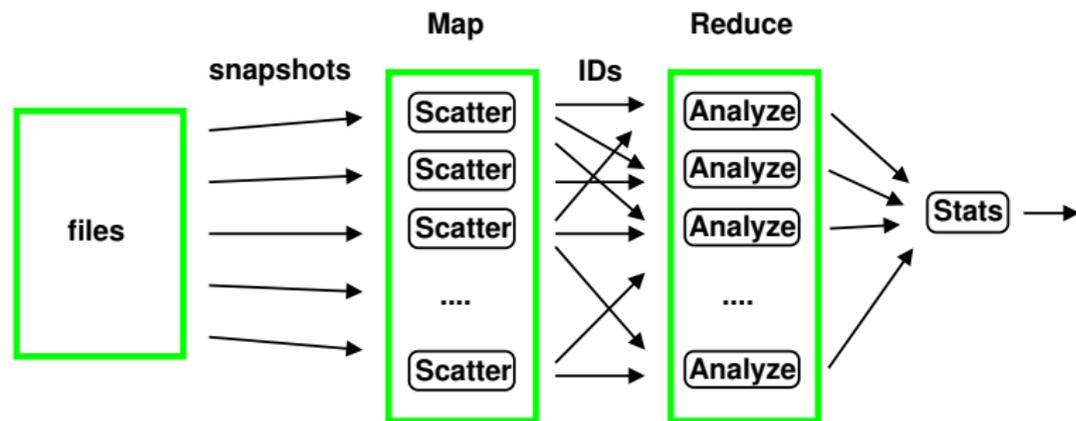
PHISH pheatures

- Based on **pipeline model**:
 - datums flow thru compute processes (not threads)
 - move datums between processes via message passing
 - multiple processes work together to perform analysis
 - split stream to enable **parallelism** & store **more state**
- Lightweight, portable **C library**
- Message passing via **MPI** or **sockets** (zeroMQ)
- Write compute kernels in C, C++, Fortran, Python, ...
- No HDFS (parallel file system with data redundancy)
- No fault-tolerance (blame it on MPI)

PHISH pheatures

- Based on **pipeline model**:
 - datums flow thru compute processes (not threads)
 - move datums between processes via message passing
 - multiple processes work together to perform analysis
 - split stream to enable **parallelism** & store **more state**
- Lightweight, portable **C library**
- Message passing via **MPI** or **sockets** (zeroMQ)
- Write compute kernels in C, C++, Fortran, Python, ...
- No HDFS (parallel file system with data redundancy)
- No fault-tolerance (blame it on MPI)
- PHISH lingo:
 - **minnow** = stand-alone program, link to PHISH lib
 - **school** = set of identical minnows (for parallelism)
 - **hook** = connection between 2 schools
 - PHISH **net**(work) = multiple schools, hooked together
 - PHISH **tales** = the manual

PHISH net for traditional MapReduce



- Scatter & analyze minnow schools for parallelism
- Convert data from **per-snapshot** to **per-particle**
- Enables trajectory analysis of individual particles

PHISH script for traditional MapReduce

minnow m scatter file1.dump file2.dump ...

minnow r analyze -dist 3.0

minnow s stats 10000

hook m hashed r

hook r single s

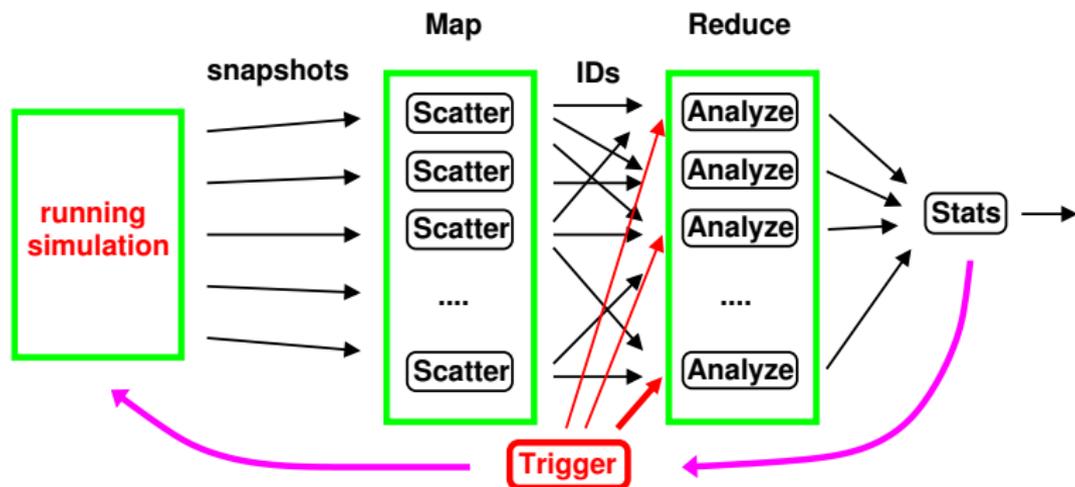
school m 16

school r 16

school s 1 invoke python

- Input script \Rightarrow **bait.py** \Rightarrow mpiexec or shell-script

PHISH net for streaming MapReduce



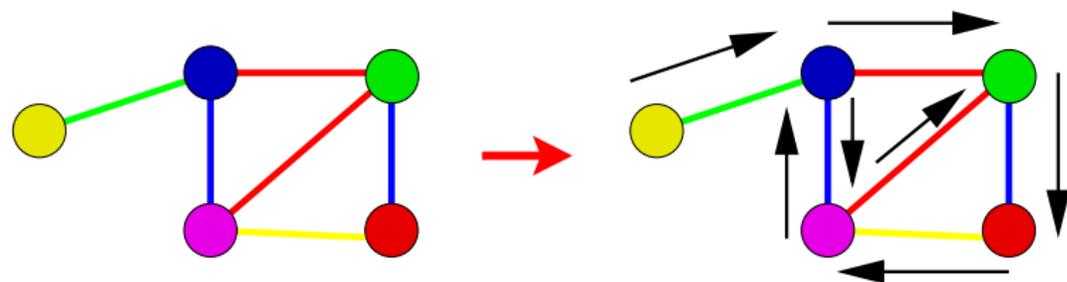
- **Streaming version:**
 - snapshot data processed on-the-fly
 - user interaction & simulation “steering”
- MapReduce is now **fine-grained** and **continuous**

Parallel streaming graph algorithms using PHISH

- ① Sub-graph isomorphism
 - ② Triangle enumeration
 - ③ Connected components
- All 3 algorithms store entire graph (in distributed memory)
 - SGI and TriEnum are **incremental**, not true streaming
 - CC is **constant-time**, can keep-up with a stream rate

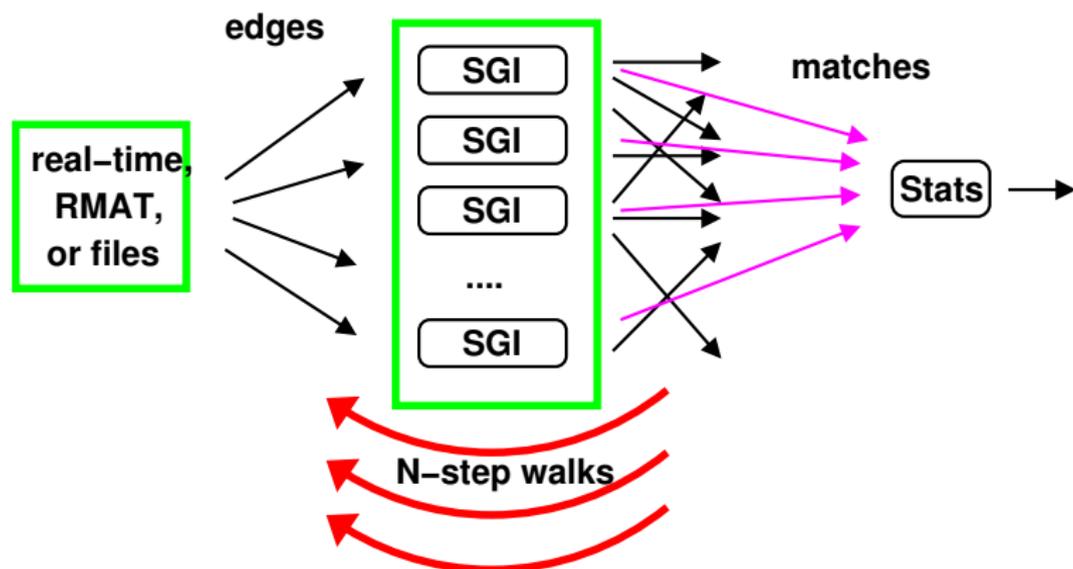
Sub-graph isomorphism

- Data mining, **needle-in-haystack** anomaly search
- Huge semantic graph with **colored vertices, edges** (labels)
- **SGI** = find all occurrences of small target graph
- Shared-memory algorithm by J Berry, MR with T Plantenga



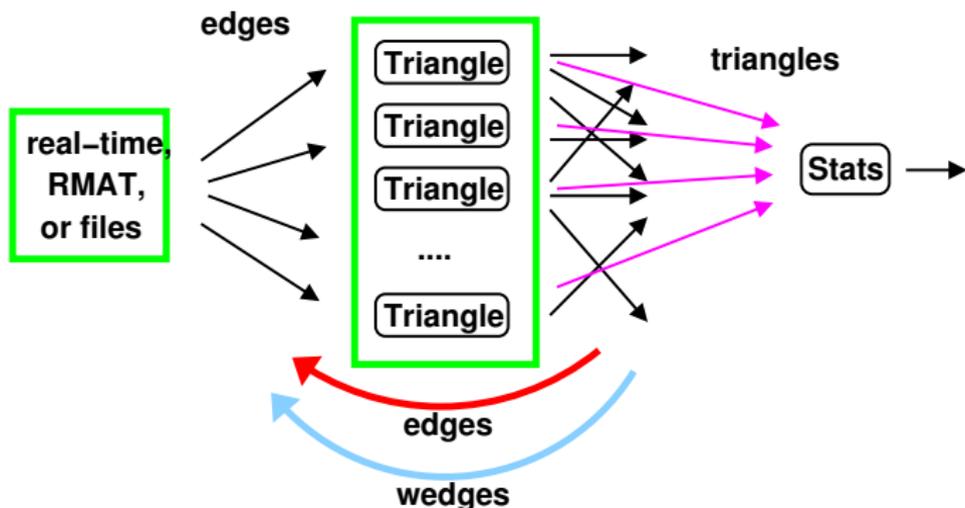
L															
F	0		2		5		3								

Streaming sub-graph isomorphism



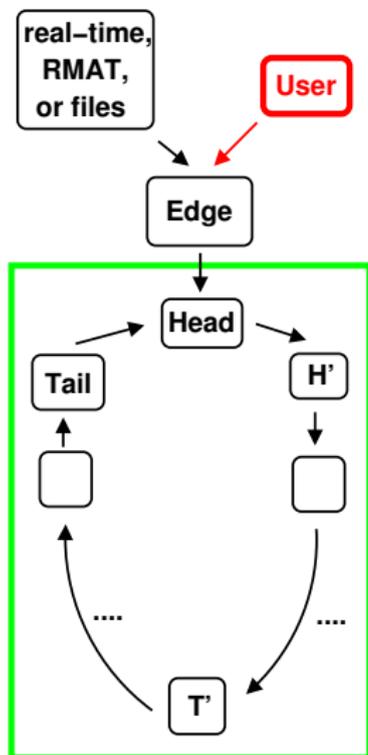
- Each edge stored **twice**, once by each vertex
- SGI minnows generate walk 1 step at time \Rightarrow **N iterations**
- Individual walks dropped if **constraint** not met
- **140 lines** of Python!

Streaming triangle enumeration

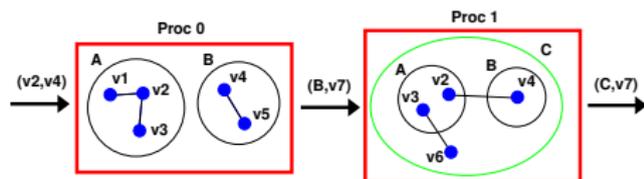


- Streaming version of *J Cohen, CS&E, 11, 29-41 (2009)*
- Owners of (I, J) vertices exchange degree/neighbor info
- **Low-degree vertex** does neighbor send
- Nth new edge triggers N-1 **wedge** messages
- 90 lines of Python

Streaming connected components



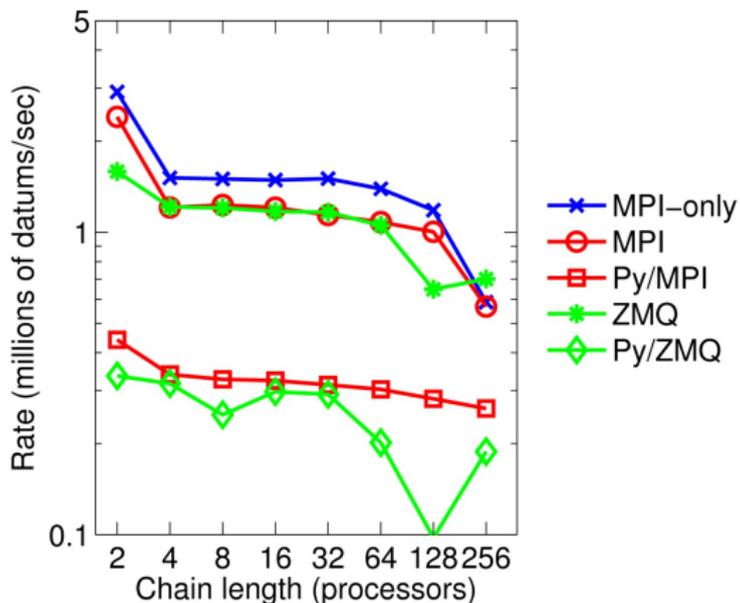
- Work with C Phillips & J Berry
- Details from Cindy tomorrow



- Graph stored **hierarchically**
edges once, vertices duplicated
- Embed **queries** in stream of edges
 - are V_i and V_j in same CC
 - what are components of size $< N$ and age $< A$
- PHISH supports mixing, ring, permutation, aging

Throughput benchmark with PHISH

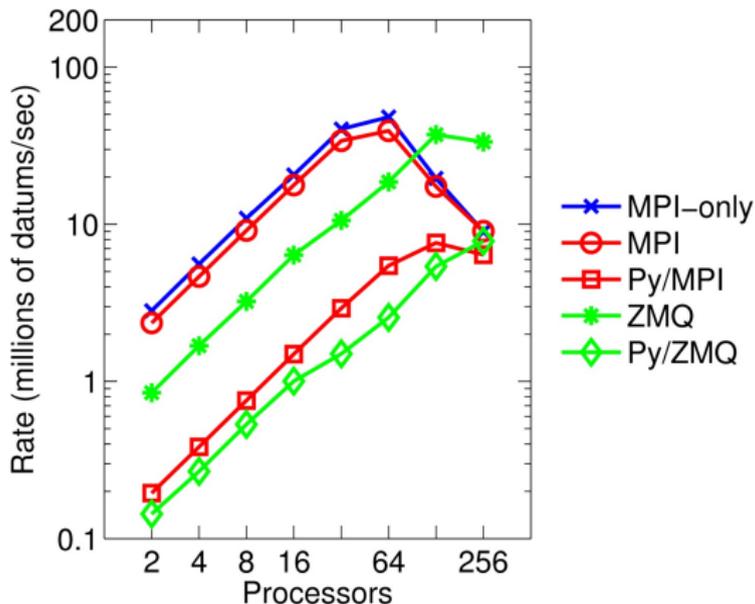
- Send zero-length datums along linear chain of processors



- Send 1K-length datums at 400K/sec (bandwidth limited)

Hashed all-to-all benchmark with PHISH

- Send zero-length datums from $P/2$ procs to $P/2$ procs



- Roll-over due to using multiple cores/node
- 147M datums/sec on 1024 procs (1 core/node)

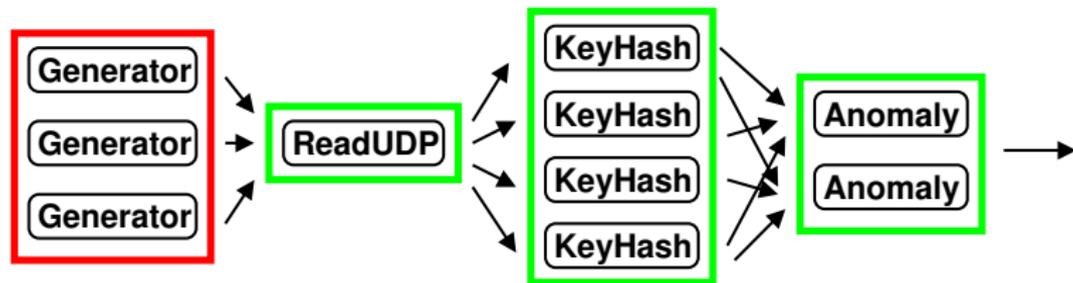
FireHose benchmarks via PHISH



FireHose benchmarks via PHISH



- Optimized C versions of anomaly detection kernels
- Parallelism via hashing to split stream across processes



serial PHISH = 1/0/1 processes, parallel PHISH = 1/4/2 processes

FireHose benchmark results

- Dell box: dual hex-core 3.47 GHz Intel Xeons (X5690)
- Maximum stream rates with no dropped packets
- #1 = fixed, #2 = unbounded, #3 = two-level keys

Implementation	Bench	# Gen	Rate (M/sec)	LOC
C++	#1	2	5.6	275
Python		1	0.45	190
PHISH (serial)		2	5.5	390
PHISH (parallel)		4	10.0	525
C++	#2	1	1.9	415
Python		1	0.14	290
PHISH (serial)		1	1.9	525
PHISH (parallel)		2	3.4	665
C++	#3	1	1.5	495

Thanks & links & papers

Open-source packages (BSD license):

- <http://www.sandia.gov/~sjplimp/phish.html> (PHISH)
- <http://mapreduce.sandia.gov> (MapReduce on top of MPI)
- <http://firehose.sandia.gov> (FireHose - soon)

Papers:

- Plimpton & Shead, *"Streaming data analytics via message passing with application to graph algorithms"*, JPDC, 74, 2687 (2014).
- Plantenga, *"Inexact subgraph isomorphism in MapReduce"*, JPDC, 73, 164 (2013).
- Berry, et al, *"Maintaining CC for infinite graph streams"*, BigMine '13, 95 (2013).
- Plimpton & Devine, *"MapReduce in MPI for large-scale graph algorithms"*, Parallel Computing, 37, 610 (2011).