Naiad: Iterative and Incremental Data-Parallel Computation
Frank McSherry  Rebecca Isaacs  Michael Isard  Derek G. Murray

Background
Data-parallel computation frameworks
• Process lots of data on many computers
• Hide low-level details from the programmer, like data distribution, scheduling and fault tolerance

The three basic techniques of data-parallel computation:
1. Express the program as a dataflow graph
2. Partition the data into many parts, across processors.
3. Each processor applies operators to its part of the data.

Payload  Time  Weight
*Brouwer* 1966  -1
*Erdős* 1913  +1
*Brouwer* 1881  +1

Typically, the dataflow graph is **acyclic** and the input data are **immutable** and **finite**.

But:
Many important algorithms contain loops
Many interesting problems are streaming
Leads to redundant computation and data movement
(and often hard to write the programs too)

Hypothesis: **Incremental** computation allows efficient iteration and recomputation when the input changes.

For example
• Declarative iteration using a **fixed-point operator**
• Each iteration produces a collection of increments
• Reaches fixed-point iff no increments produced

Naiad recasts parallel dataflow as computation over **increments**

Worked Example: Shortest Paths
1. Programmer writes declarative program with loops
2. C# Program is rendered to a cyclic dataflow graph
3. The dataflow graph is replicated for each independent thread, process, computer.
4. Computation proceeds with the shards exchanging increments on dataflow edges.
5. When no unprocessed increments remain, the computation quiesces and returns.

Future Research Directions
Scale up distributed implementation
Use dataflow graph to inform memory management
Explore trade-off between eager and lazy processing
Optimistic: may do unnecessary work; may scale.
Conservative: synchronizes to optimize work done.