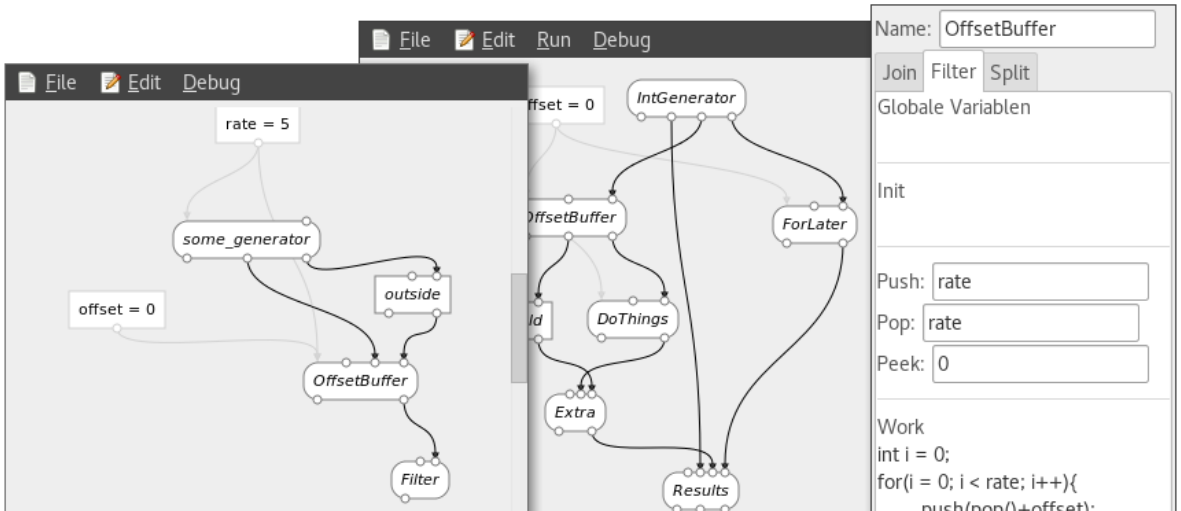
StreamGraph

Florian Ziesche, Jakob Karge and Boris Graf

Technische Universität Berlin

October 17, 2016



Overview

Idea & Goals

- Visualize parallel programming
- Graphical layer on top of a textual language
- StreamIt: subset representation

StreamIt

- Developed for streaming applications
- Pipeline-based structure

Base constructs

- Filter: "The Workhorse of StreamIt"; *work*-, *init*-functions
- Pipeline: linear succession of StreamIt constructs
- Split: split single data stream into multiple data streams
- Join: join previously splitted streams together

For further information on StreamIt see [1].

Parallelism in StreamIt

Split-Join

Splitting (or duplicating) of data:

- Paths independent between *split* and *join*

As result paths may be processed in parallel

Pipeline

Continuous data stream through filters:

- *Work*-functions are constantly recalculated.
- Filters can be processed independently.

Therefore filters in pipelines can be parallelised.

StreamGraph GUI

- Minimalistic and clear design
- Mouse control via pop-up menus
- Intuitive control

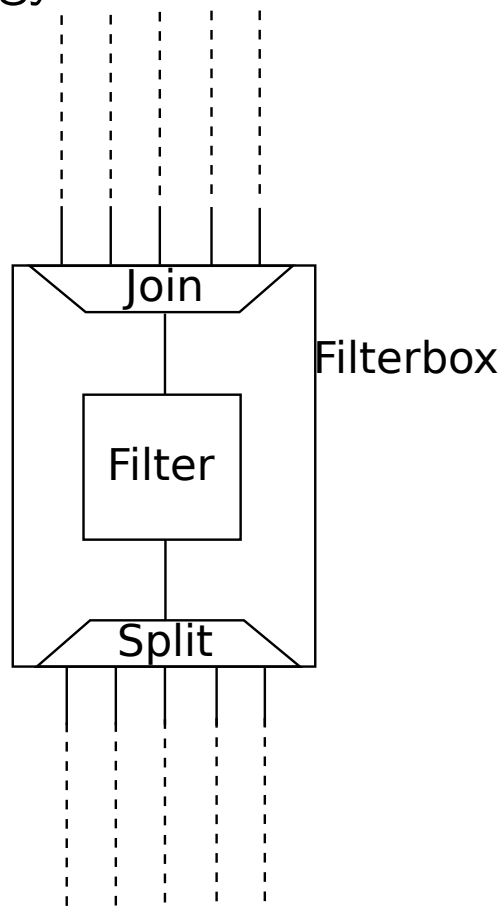
Elements

- Menu bar
- Working surface
- Notification bar
- Property window

StreamGraph GUI

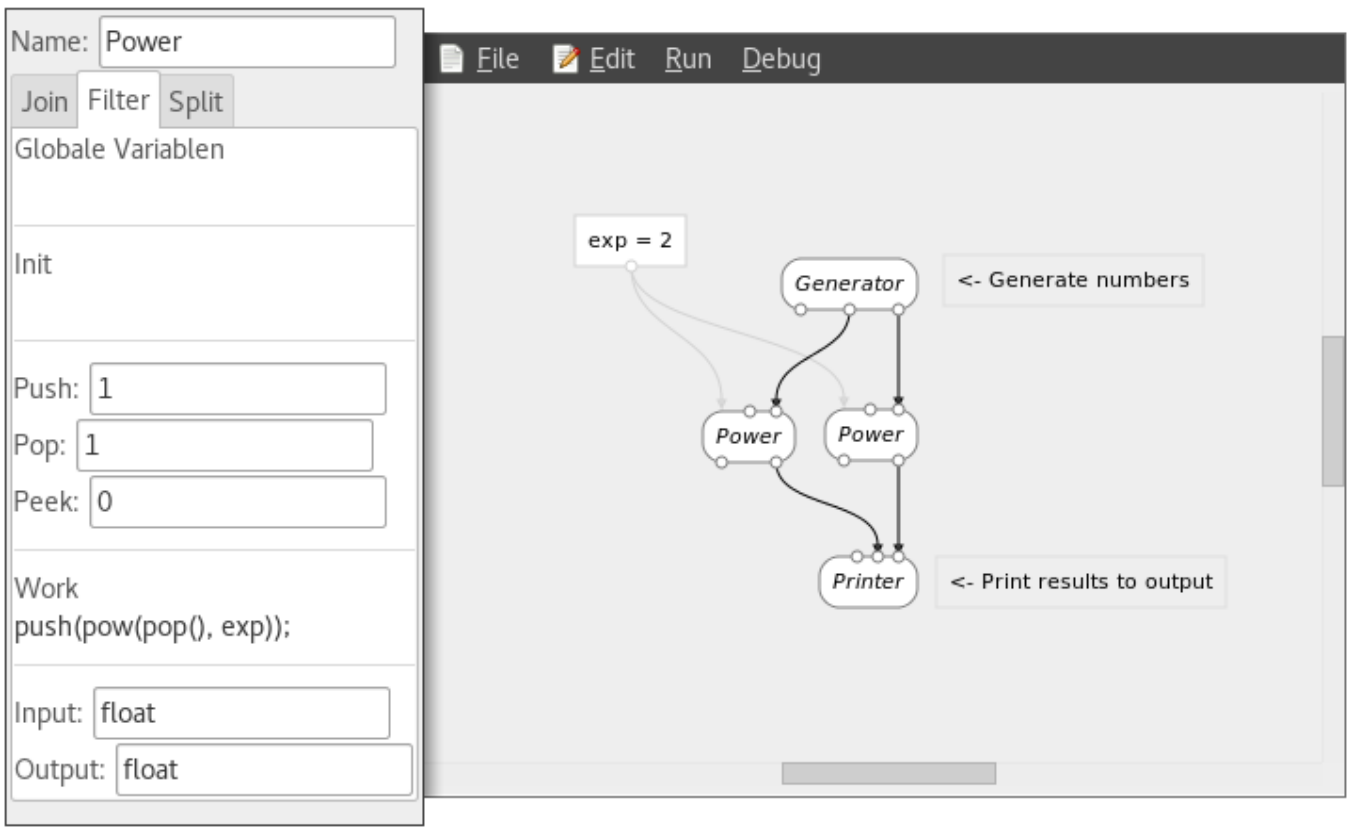
Visual abstractions of StreamIt toplogy

- Split & join integrated into filters
- Implicit pipelines



Demo

DEMO: `demoPower.str`



StreamGraph View

- Based on Gtk2::Ex::MindMapView [2]
- Gnome Canvas used as engine
- Item $\rightarrow$ border $\rightarrow$ content
- Handle keyboard and mouse events
- Many other changes

Model

Program structures modeled in two ways:

Graphical: *Node*

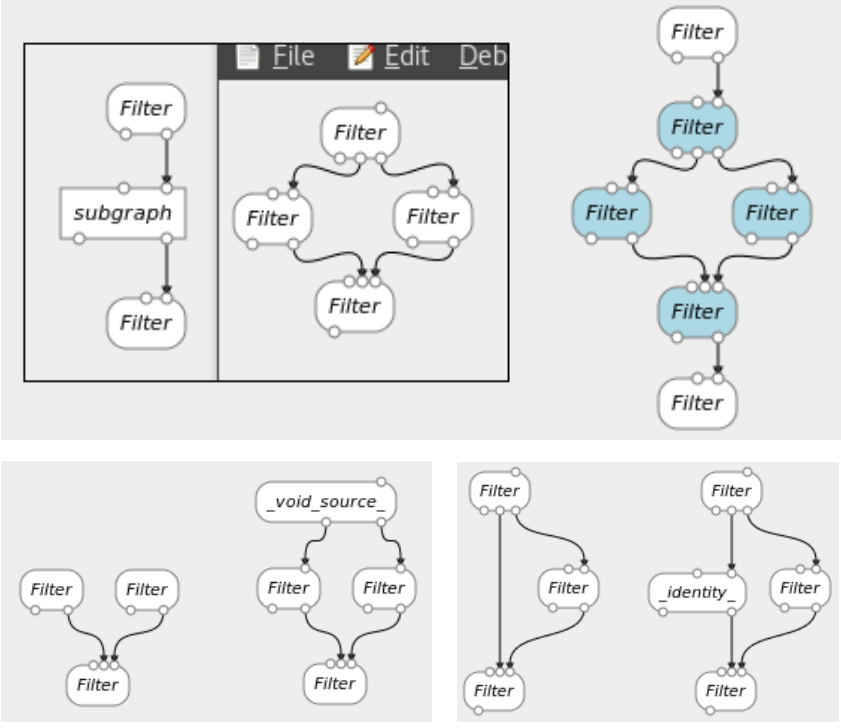
- Node types (*Filter*, *Subgraph*, *Parameter*, *Comment*)
- Factory

Textual: *CodeObject*

- StreamIt types (*Pipeline*, *Splitjoin*, *Parameter*)
- *CodeObject*s derived from *Node*s

Graph Transformation

- Load subgraphs
- Add void sources and -sinks
- Add identities for empty pipelines
- Test for graph errors



Code Generation

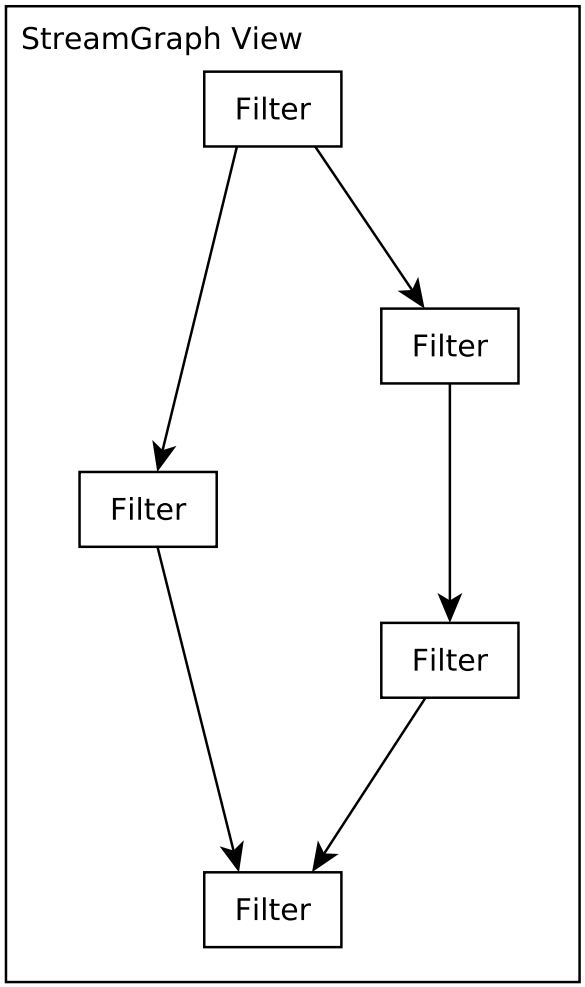
General procedure

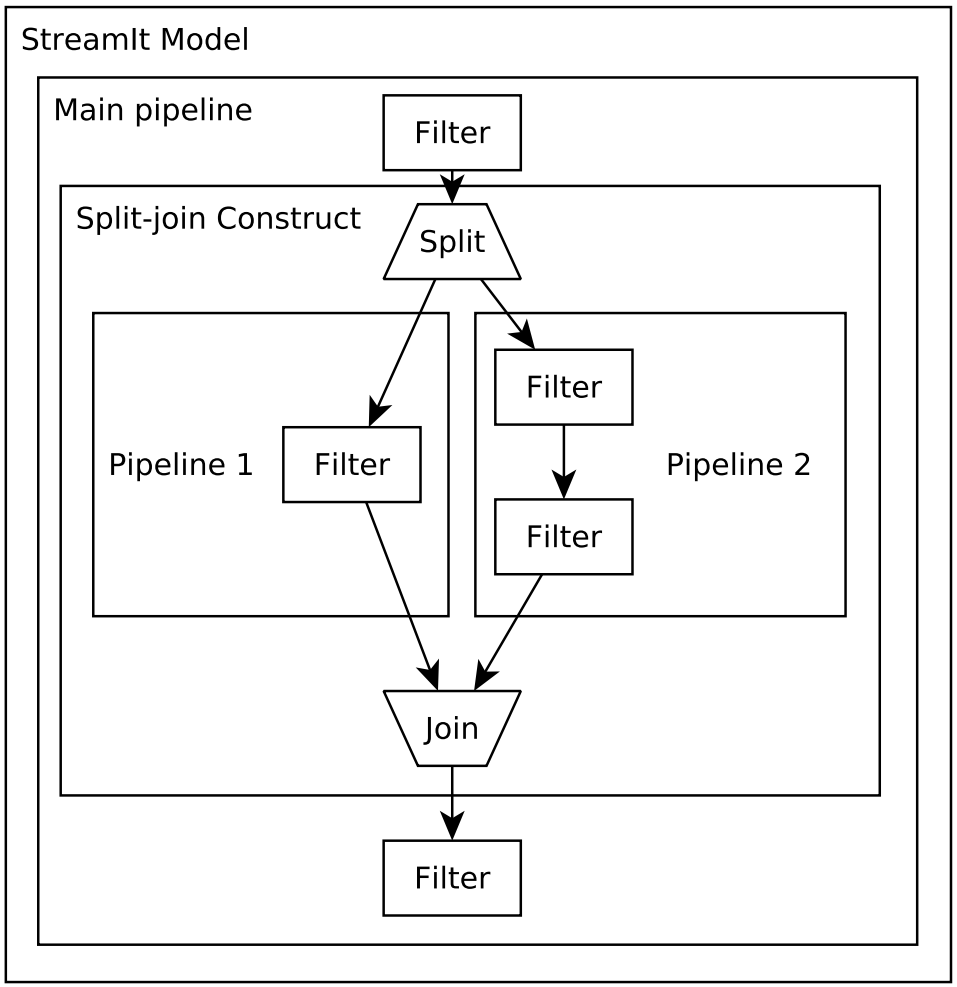
- Generate code for all filters
- Build pipeline-based structure
- Generate code for topology in recursive manner

Filter generation

- Make name unique
- Get parameters
- Build *init*- and *work*- function

StreamIt pipeline-based structure





Code Generation

Building the structure

- Begin with pipeline (main)
- If split is detected start building of split-join
  Build a pipeline for every Path.
- Continue with generation of pipeline.

Indirect recursive method guarantees StreamIt-type hierarchical structure

Get the code by generating from inside out.

Results

- Working prototype
- Limited StreamIt representation
- Complete toolchain from "empty" to "running"

Improvements

- Ordered splitjoin (currently only commutative use of results)
- More StreamIt constructs (e.g. message passing, feedback-loop)
- UI
  - Group-to-subgraph
  - In-graph error highlighting
  - Data visualization
  - ...
- More automatic compatability
- StreamIt+Java1.5+Perl-packages auto setup
- Code quality & style

Future

Possible research and development directions

- Different background language
- More general graphical environment

Bibliography


William Thies, Michael Karczmarek, and Saman Amarasinghe, *StreamIt: A Language for Streaming Applications*, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.


`http://search.cpan.org/~hemlock/Gtk2-Ex-MindMapView-0.000001/lib/Gtk2/Ex/MindMapView.pm`