P4: Portable Parallel Processing Pipelines
for Interactive Information Visualization

Kelvin Li
University of California, Davis

BayVAST, 2018

Goals

- Combine declarative visualization design and GPU Computing
- Provide both performance and flexibility
- Support developing high-performance visualization systems for big data applications

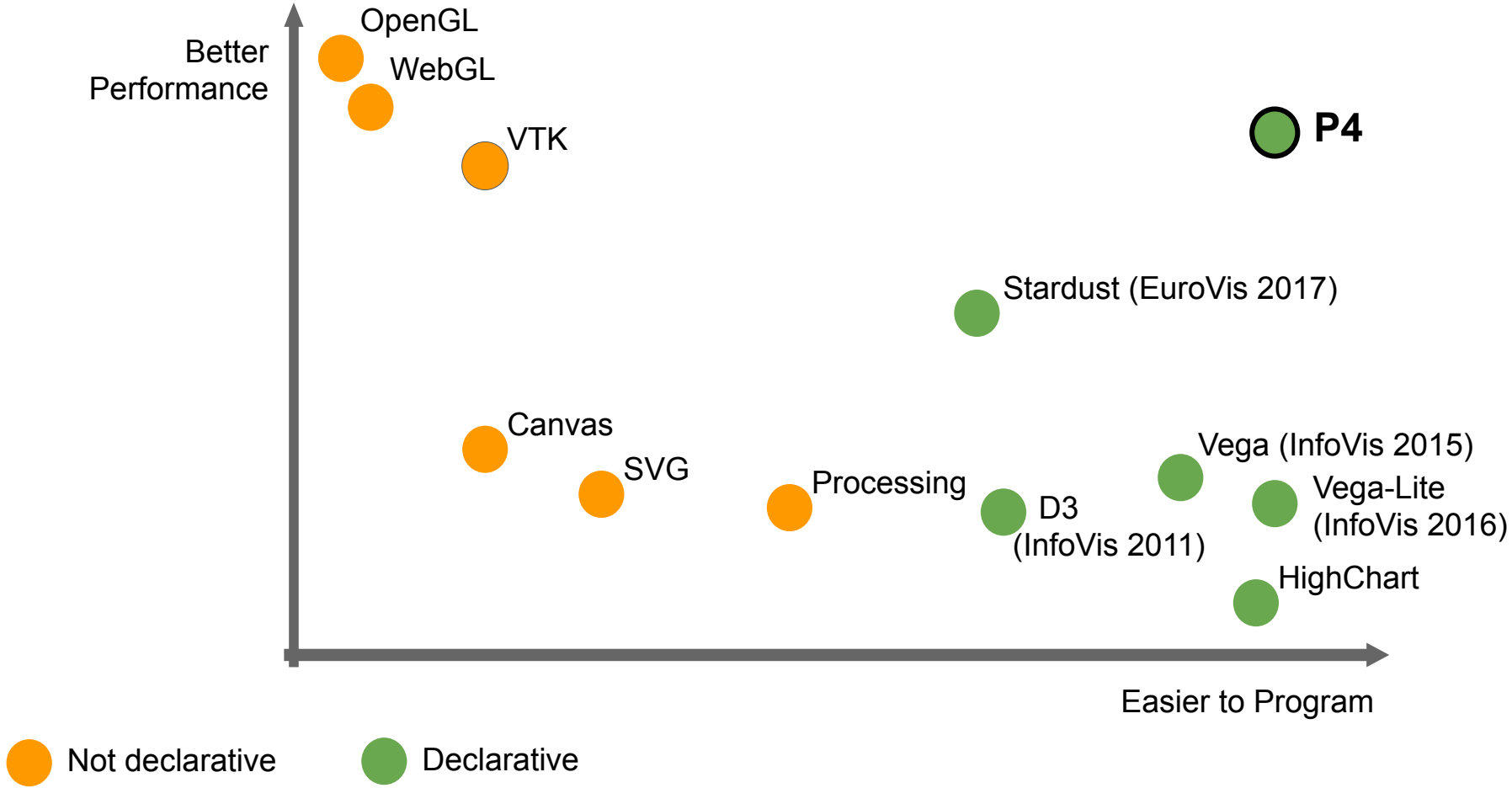
Background: Declarative Visualization Language

Declarative visualization languages specify ***what*** should be visualized instead of ***how*** to visualize.

- Make visualization systems much easier to develop
- Become popular for creating visualization applications
    - Examples: ggplot and D3

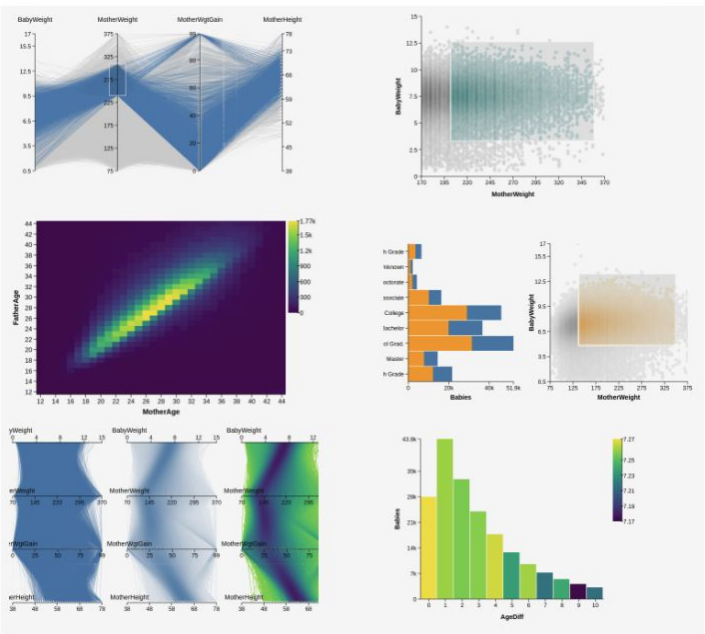
Visualization Toolkits and Libraries

Declarative visualization languages (e.g., D3, ggplot, Vega/Vega-Lite)	Easy to use but Low performance
Low-level graphics and visualization libraries (e.g., OpenGL/WebGL, VTK)	High performance but Difficult to program


Better
Performance
OpenGL
WebGL
VTK
P4
Stardust (EuroVis 2017)
Canvas
SVG
Processing
D3
(InfoVis 2011)
Vega (InfoVis 2015)
Vega-Lite
(InfoVis 2016)
HighChart
Easier to Program
Not declarative
Declarative

Current Version of P4

- Implemented based on WebGL 1.0
- Provided most common data transformations (e.g., ***deriving new values, filtering, binning, and group-by aggregations***)
- Supported common visualization designs (e.g., ***bar charts, heat maps, scatter plots, parallel coordinates)***



InfoVis Reference Model

Source Data

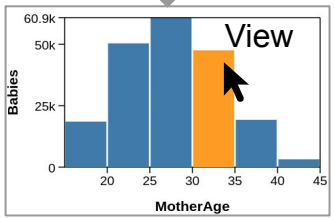
Data Transformation

Analytical Abstraction

Visual Mapping

Visualization Abstraction

View Transformation

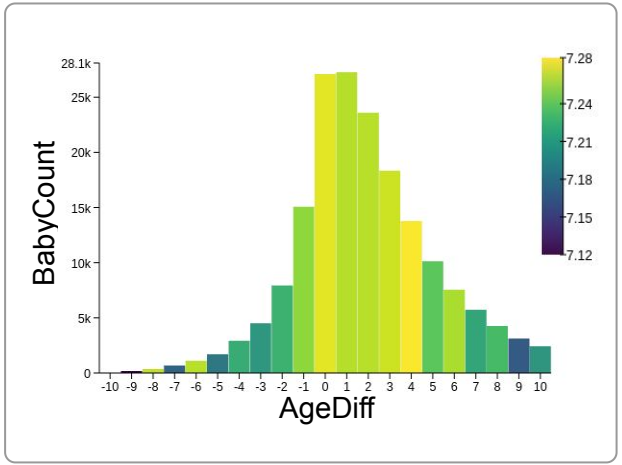


Interaction

P4 Declarative Grammar

```
{
  "data": {
    "source": {...},
    "scheme": {
      "BabyWeight" : "float",
      "BabyGender" : "string",
      "MotherAge"  : "int",
      …,
      "FatherAge"  : "int"
    }
  },
  "operations": [
    { "$match": {"MotherAge": [15, 46]} },
    {
      "$aggregate": {
        "$bin": {"MotherAge": 6},
        "$collect": {"Babies": {"$count": "*"}}
      }
    },
    {
      "$visualize": {
        "id": "chart",
        "mark": "bar",
        "x": "MotherAge",
        "y": "Babies",
        "color": "steelblue"
      }
    },
    {
      "$interact": {
        "event": "click", "from": "chart",
        "response": {
          "chart": {
            "selected": {"color": "orange"}
          }
        }
      }
    }
  ],
  "views": [
    {"id": "chart", "width": 600, "height": 400}
  ]
}
```

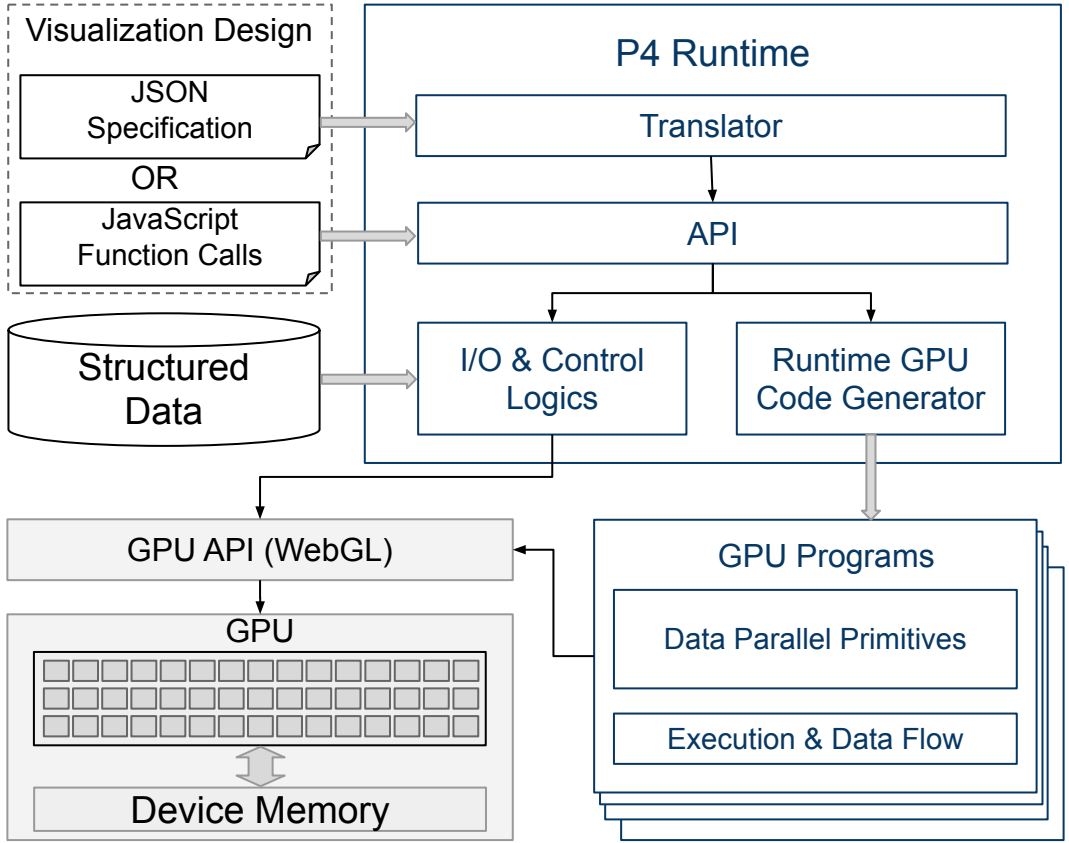
JavaScript API




```
pipeline.data(...)

.derive({ AgeDiff: “FatherAge - MotherAge”})

.match({ AgeDiff: [-10, 10]})

.aggregate({
  $group: "AgeDiff",
  $collect: {
    BabyCount: { $count: "*" },
    AvgBabyWeight: { $avg: "BabyWeight" }
  }
})
.visualize({
    mark: "bar",
    x: "AgeDiff",
    y: "BabyCount",
    color: {
      field: “AvgBabyWeight”,
      scheme: “viridis”
    }
})
```

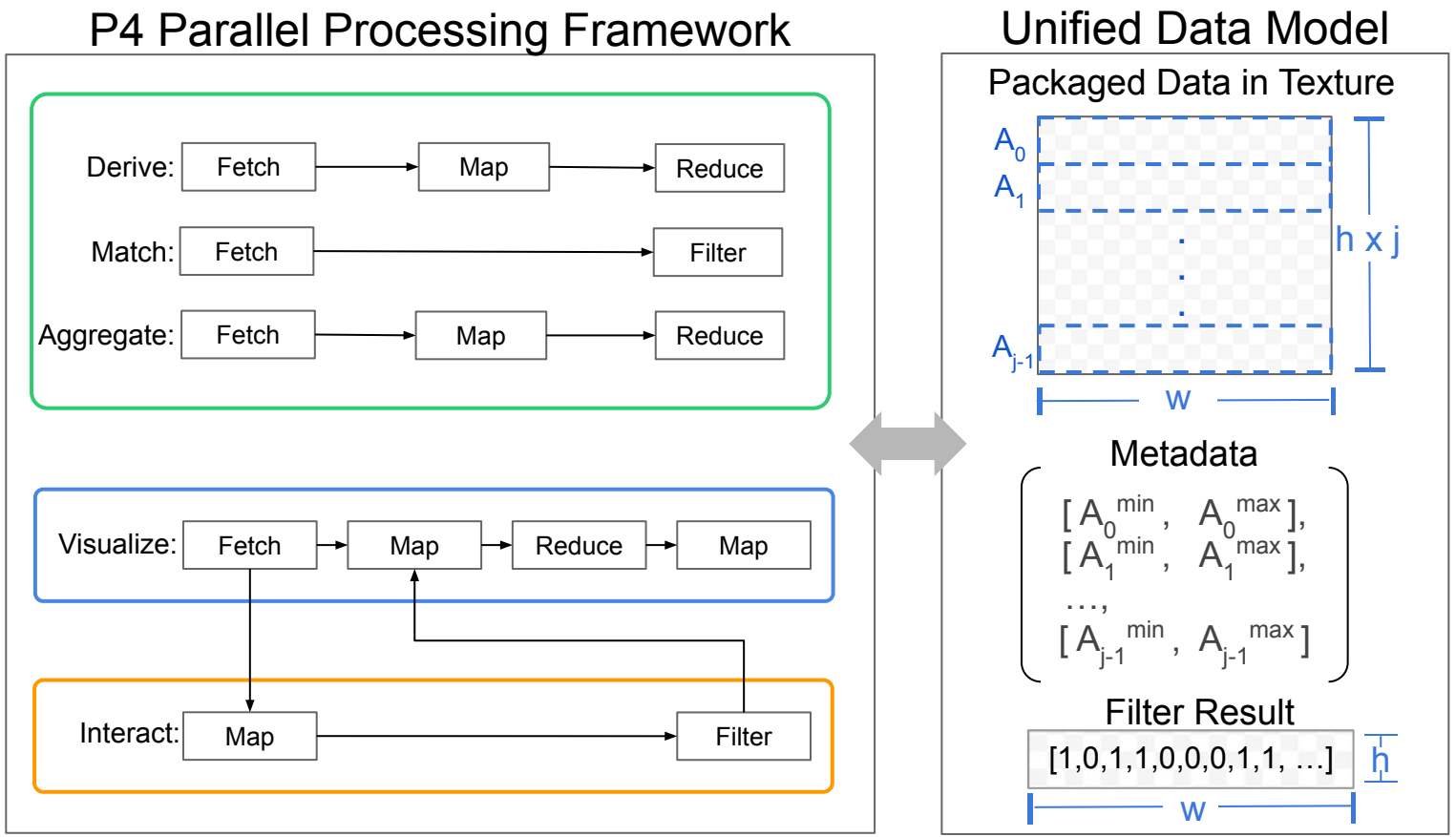
Dataflow and Workflow

- Cache all data in GPU memory
- Accelerate both data transformation and visualization operations using the GPU



Runtime GPU Code Generator

- Create GPU programs based on user specifications at runtime
- Compose all data processing and visualization operations by using four ***data-parallel primitives*** *(**Fetch, Map, Filter, and Reduce**)*


P4 Parallel Processing Framework
Derive:
Fetch
Map
Reduce
Match:
Fetch
Filter
Aggregate:
Fetch
Map
Reduce
Visualize:
Fetch
Map
Reduce
Map
Interact:
Map
Filter
Unified Data Model
Packaged Data in Texture
A0
A1
Aj-1
h x j
w
Metadata
[A0min , A0max],
[A1min , A1max],
…,
[Aj-1min , Aj-1max]
Filter Result
[1,0,1,1,0,0,0,1,1, …]
h
w

Aggregation Query

```
var aggr = P4.pipeline().data(...)
  .aggregate({
    $group: "MotherRace",
    $collect: {
        AvgWeight: {
            $avg: "BabyWeight"
        }
    }
})
```

WebGL Pipeline

GPU Memory

Textures

Generated WebGL 1.0 Code

c1

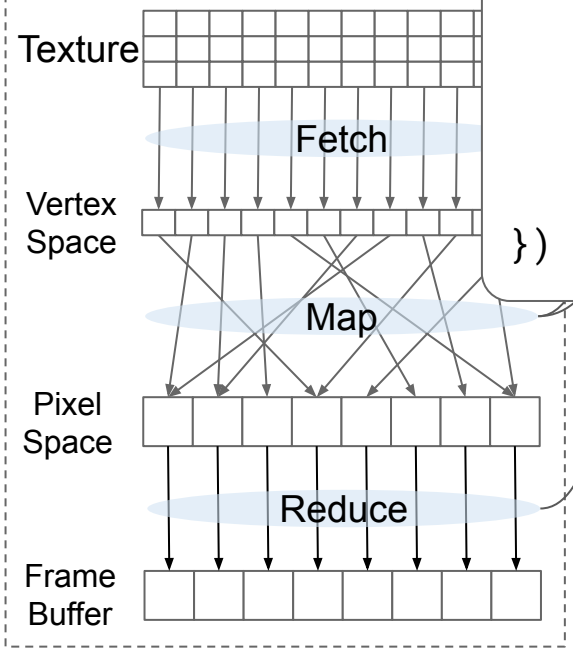
```
varying float value;
uniform int p4_aggr_key;
uniform int p4_field_id;
….
float p4_dpp_fetch(int key) {...};
float p4_dpp_map(int key) {...};

t(p4_dpp_fetch(p4_aggr_key));
p_fetch(p4_field_id);
_dpp_map(p4_aggr_key);
 vec4(ox*2.-1., 0., 0., 1.);
```

c2

```
or = vec4(0., 0., 1., value);
```

```
P4.pipeline().data(...)
  .aggregate({
    $group: "MotherRace",
    $collect: {
        AvgWeight: {
            $avg: "BabyWeight"
        }
    }
})
```

Aggregation Workflow



Blender

Offscreen FBO

Views

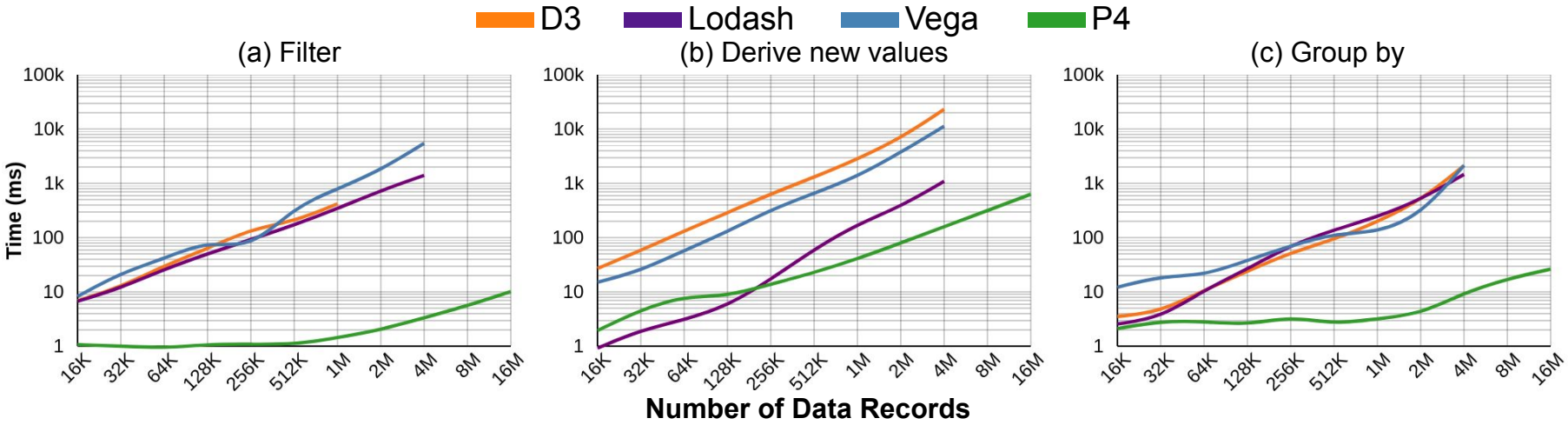
c3

```
var gl = aggr.ctx;
gl.enable(gl.BLEND);
gl.blendFunc(gl.ONE, gl.ONE);
gl.blendEquation(reduce_method); //gl.ADD
```

c4

```
gl.drawArraysInstanced(gl.POINTS,….);
```

Benchmarks: Data Transformation



Benchmarks performed using Google Chrome on a system with a Intel i7-4790 CPU and Nvidia GTX Titan GPU

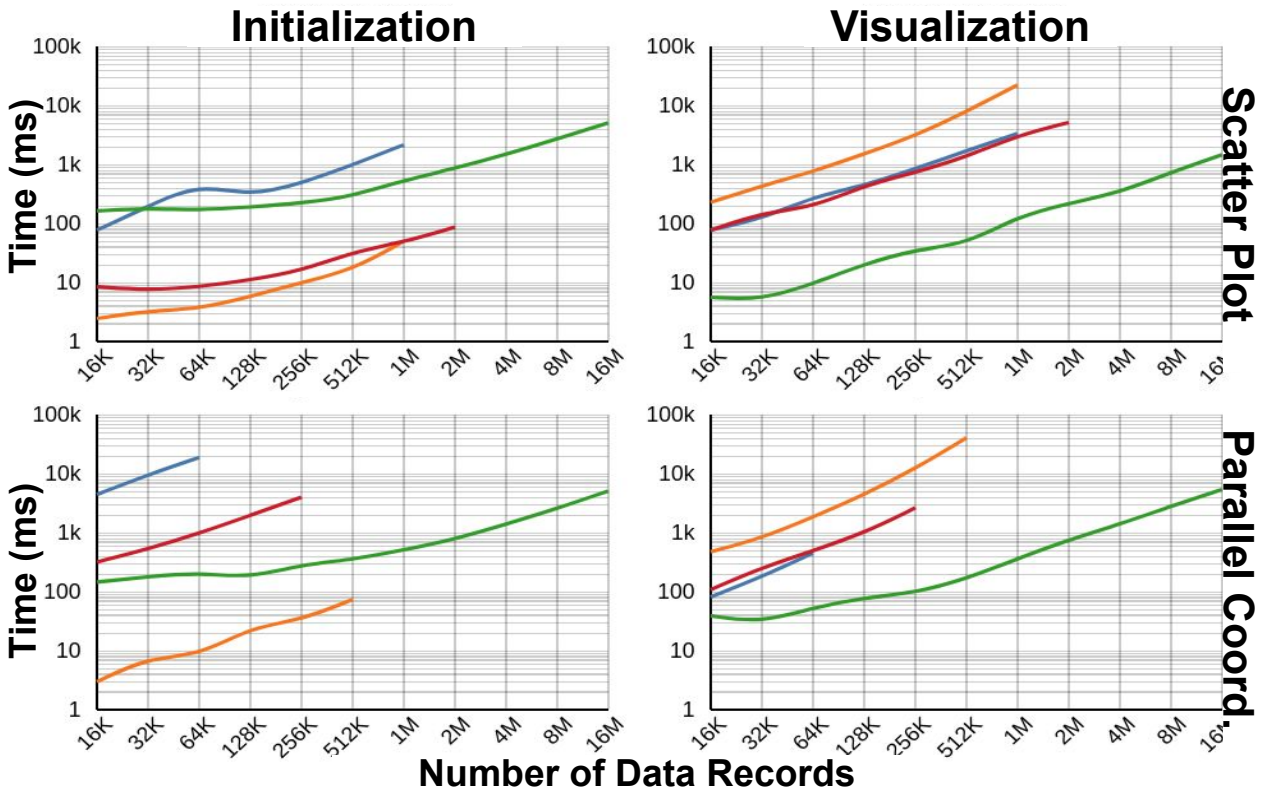
Benchmarks: Visualization

D3

Vega

Stardust

P4



Limitation and Future Work

- Data size must be less than GPU memory limit
- Current version only supports a small set of data transformation and visualization operations
- Future work:
    - Support progressive visual analytics to allow data size larger than memory
    - Add more functionalities for supporting visual analytics
    - Allow extensions to be added by others

Conclusion

- P4 combines declarative visualization design and GPU computing
- P4's novel framework makes this possible:
  - Formulate all InfoVis operations using only four data-parallel primitives (DPP)
  - Leverage ***DPP*** to generate GPU code based on user specification at runtime
- Our framework can be extended and applied to:
  - Server backends and desktop environments
  - Distributed computing on GPU clusters

Thank You !

Email: kelli@ucdavis.edu

Project Homepage: https://jpkli.github.io/p4

Source Code: https://github.com/jpkli/p4

TVCG Paper: https://ieeexplore.ieee.org/document/8468065