

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

## Mineria de Datos

### P4: Portable Parallel Processing Pipelines for Interactive Information Visualization

MSc. Vicente Machaca Arceda

29 de mayo de 2020



## Introducción

### P4

Arquitectura

Interfaces de programación

Framework de procesamiento paralelo

## Resultados

## Conclusiones



La visualización de datos a probado ser efectiva para el razonamiento de grandes volúmenes de datos [1].

**Figura:** Ejemplo de visualización de datos.

# Introducción

Gramáticas declarativas para visualización de datos



- ▶ **D<sup>3</sup>** data-driven documents [2].
- ▶ **ggplot2**: elegant graphics for data analysis [3].
- ▶ **Reactive vega**: A streaming dataflow architecture for declarative interactive visualization [4].
- ▶ **Vega-lite**: A grammar of interactive graphics [5].

# Introducción

Gramáticas declarativas para visualización de datos

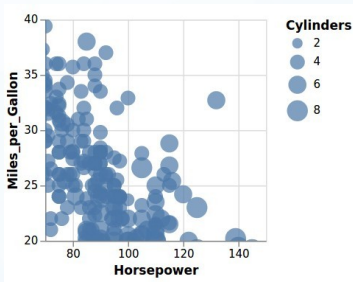


Figura: Scatterplot en Vega-lite.

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v4.json",
  "data": {"url": "data/cars.json"},
  "selection": {
    "grid": {
      "type": "interval", "bind": "scales"
    }
  },
  "mark": "circle",
  "encoding": {
    "x": {
      "field": "Horsepower", "type": "quantitative",
      "scale": {"domain": [75, 150]}
    },
    "y": {
      "field": "Miles_per_Gallon", "type": "quantitative",
      "scale": {"domain": [20, 40]}
    },
    "size": {"field": "Cylinders", "type": "quantitative"}
  }
}
```



### Problema

La mayoría de software de visualización de datos, no hacen uso del procesamiento paralelo para construir sistemas interactivos de alto desempeño [6].



### Problema

La mayoría de software de visualización de datos, no hacen uso del procesamiento paralelo para construir sistemas interactivos de alto desempeño [6].

### Propuesta

El paper presenta **P4**, un software de visualización. P4 hace uso de un lenguaje declarativo y aprovecha el poder computacional del GPU.



Introducción

P4

Arquitectura

Interfaces de programación

Framework de procesamiento paralelo

Resultados

Conclusiones



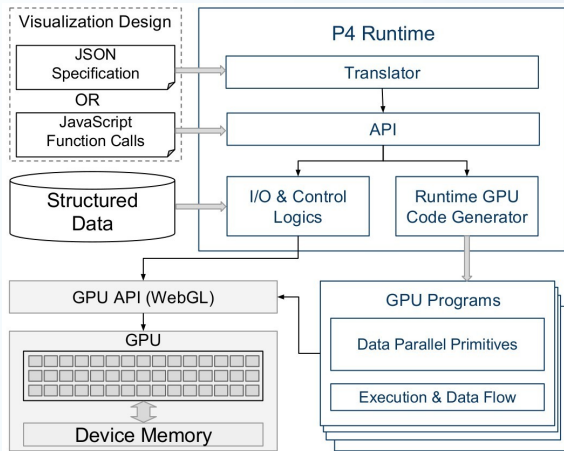


Figura: Arquitectura de P4. Fuente: [6]



Introducción

P4

Arquitectura

Interfaces de programación

Framework de procesamiento paralelo

Resultados

Conclusiones



Se utilizará una base de datos de nacimientos del 2015 (200000 muestras). Los atributos son:

- ▶ Mes de nacimiento.
- ▶ Genero del bebe.
- ▶ Peso del bebe.
- ▶ Edad de la madre.
- ▶ Raza de la madre.
- ▶ Estado civil de la madre.
- ▶ Grado de educación de la madre.
- ▶ Altura de la madre.
- ▶ Peso de la madre.
- ▶ Aumento de peso de la madre durante el embarazo.
- ▶ Edad del padre
- ▶ Raza del padre.
- ▶ Grado de educación del padre.



### Transformación de datos

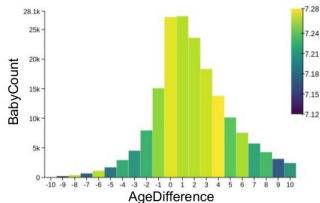
- ▶ **Derive.**- Genera nuevos atributos.
- ▶ **Match.**- Filtra los atributos.
- ▶ **Aggregate.**- Agrupa atributos.

# Interfaces de programación de P4

## Transformación de datos



```
var pipeline = P4.pipeline().data({
  method: "http",
  path: "data/Natality.csv",
  attributes: {
    BabyWeight: "float",
    BabyGender: "string",
    MotherAge: "int",
    ...,
    FatherAge: "int"
  }
});
```



```
pipeline
  .derive({
    AgeDifference: function(d) {
      return d.FatherAge - d.MotherAge;
    }
  })
  .match({
    AgeDifference: [-10, 10]
  })
  .aggregate({
    $group: "AgeDifference",
    $collect: {
      BabyCount: { $count: "*" },
      AvgBabyWeight: { $avg: "BabyWeight" }
    }
  })
  .visualize({
    mark: "bar",
    x: "AgeDifference",
    y: "BabyCount",
    color: {
      field: "AvgBabyWeight",
      scheme: "viridis"
    }
  })
});
```

Figura: Ejemplo de transformación de datos con P4. Fuente: [6]



### Mapeo visual

Permite que los atributos se renderizen de manera facil a canales visuales como: color, opacidad, ancho, altura, posición del eje x y eje y.

# Interfaces de programación de P4

## Mapeo visual



```
.data({...})  
.match({  
  MotherAge: [12, 50],  
  FatherAge: [12, 50]  
})  
.aggregate: ({  
  $group: ["FatherAge", "MotherAge"],  
  $collect: {  
    Babies: { $count: "*" }  
  }  
})  
.visualize({  
  mark: "rect",  
  x: "FatherAge",  
  y: "MotherAge",  
  color: "Babies"  
})
```

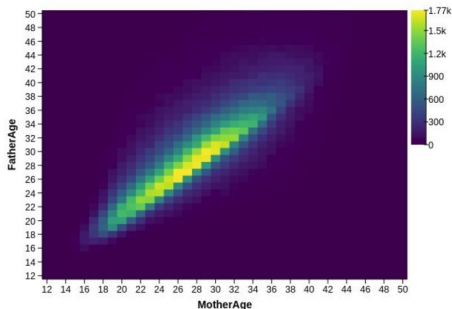


Figura: Ejemplo de visualización de datos con P4. Fuente: [6]

# Interfaces de programación de P4

## Mapeo visual



```
{
  "$visualize": {
    "mark": "line",
    "color": "steelblue",
    "opacity": 0.1,
    "y": [
      "BabyWeight",
      "MotherWeight",
      "MotherWgtGain",
      "MotherHeight"
    ],
    "brush": {
      "unselected": {
        "color": "lightgrey"
      }
    }
  }
}
```

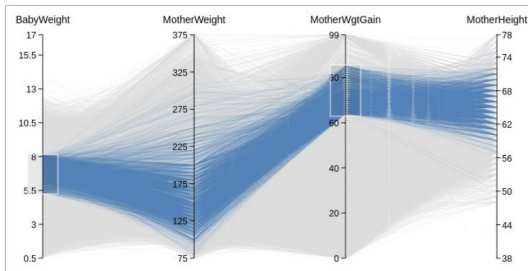


Figura: Ejemplo de visualización de datos con P4. Fuente: [6]



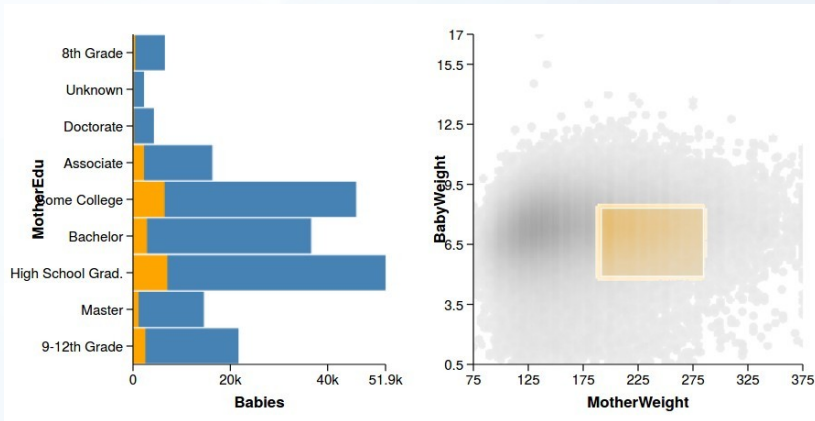


Figura: Ejemplo de interacciones con P4. Fuente: [6]



P4, mejora la reducción del ruido al mostrar grandes volúmenes de datos. Esto lo logra procesando la opacidad en cada pixel  $\hat{\alpha}$  con:

$$\hat{\alpha} = (1 - \alpha_{min})\left(\frac{p}{p_{max}}\right)^{\gamma} + \alpha_{min}$$

Donde:

- ▶  $p$ : Número de marcas superpuestas en pixel actual.
- ▶  $p_{max}$ : Máximo número de marcas superpuestas en toda la imagen.
- ▶  $\gamma = 1/3$
- ▶  $\alpha_{min} = 0,1$

# Interfaces de programación de P4

## Mejora de percepción

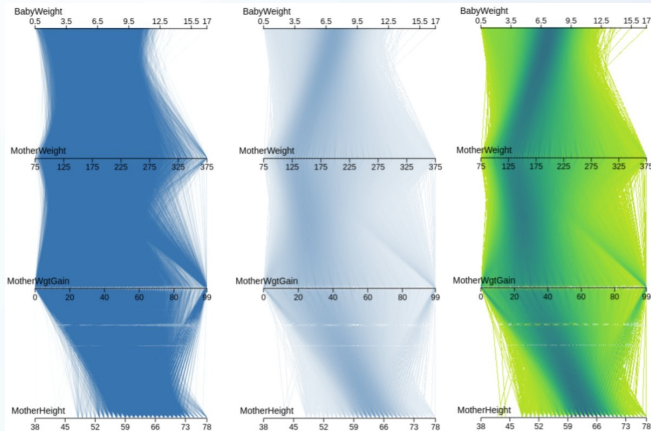


Figura: Izquierda: Alpha blending de WebGL, Centro: Ajuste de opacidad con P4, Derecha: Mapeo de color con P4. Fuente: [6]



Introducción

P4

Arquitectura

Interfaces de programación

Framework de procesamiento paralelo

Resultados

Conclusiones



### Primitivas de paralelización

Las operaciones de P4, pueden ser implementadas como las primitivas de programación funcional: *map*, *filter* y *reduce*. Entonces, si se implementa dichas primitivas funcionales en GPU, se soluciona el problema de paralelización de P4.

P4 tiene cuatro primitivas:

- ▶ Fetch.
- ▶ Map.
- ▶ Filter.
- ▶ Reduce.

# Framework de procesamiento paralelo

*map, filter y reduce*

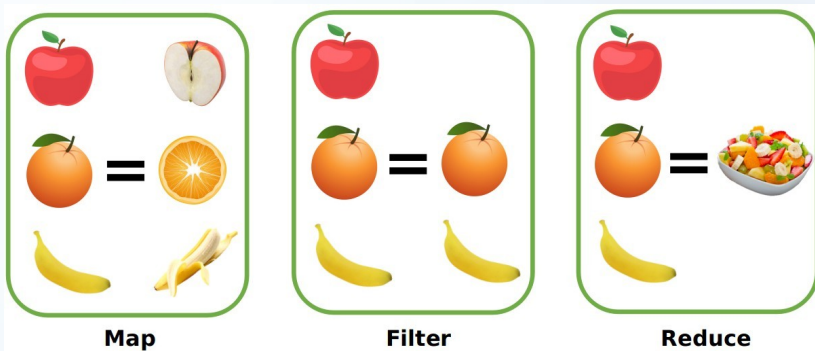


Figura: Ejemplo de *map*, *filter* y *reduce* utilizados en programación funcional.

# Framework de procesamiento paralelo

## Primitivas

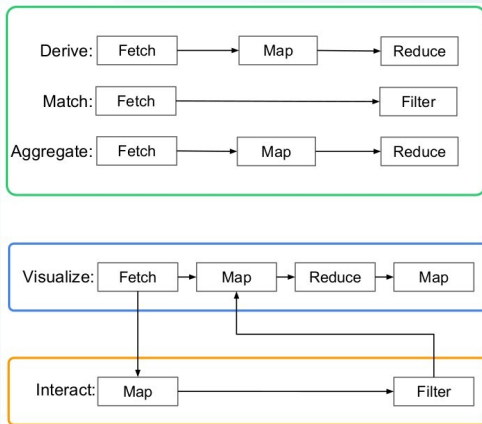
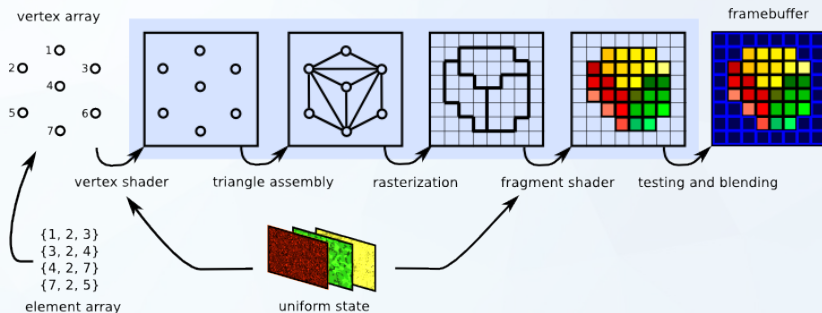


Figura: Framework para el procesamiento paralelo de P4. Fuente: [6]

# Framework de procesamiento paralelo

*Vertex shader y Fragment shader*



**Figura:** Ejemplo de *Vertex shader* y *Fragment shader* en WebGL.





- **Fetch.**- La información se almacena como texturas y se usan los shader para acceder a la información.



- ▶ **Fetch.**- La información se almacena como texturas y se usan los shader para acceder a la información.
- ▶ **Map.**- *Vertex shader* para procesar resultados intermedios.  
*Fragment shader* para escribir los resultados.



- ▶ **Fetch.**- La información se almacena como texturas y se usan los shader para acceder a la información.
- ▶ **Map.**- *Vertex shader* para procesar resultados intermedios. *Fragment shader* para escribir los resultados.
- ▶ **Filter.**- *Vertex shader* para el filtro. *Fragment shader* para guardar los resultados.



- ▶ **Fetch.**- La información se almacena como texturas y se usan los shader para acceder a la información.
- ▶ **Map.**- *Vertex shader* para procesar resultados intermedios. *Fragment shader* para escribir los resultados.
- ▶ **Filter.**- *Vertex shader* para el filtro. *Fragment shader* para guardar los resultados.
- ▶ **Reduce.**- Utiliza *blending* para obtener los máximos, mínimos, etc.



**Cuadro:** Librerías utilizadas para la comparación con P4.

Librería	Versión
<i>D</i> <sup>3</sup>	3.5.17
Vega	3.0.2
Lodash	4.17.4
Startdust	0.1.1

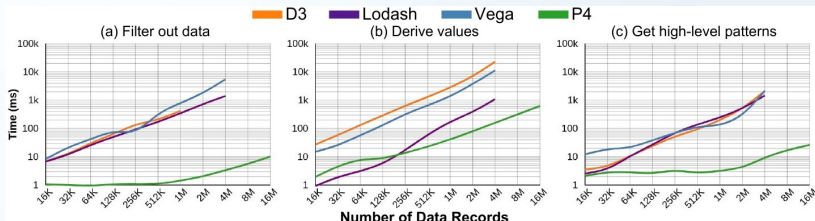


**Cuadro:** Tarjetas GPU utilizadas en los experimentos.

Fabricante	Modelo	Nucleos	Memoria
Intel	HD520	192	1GB DDR3
Nvidia	GTX940m	384	2GB DDR3
AMD	HD7970	2048	3GB DDR5
Nvidia	GTXTitan	2688	6GB DDR5

# Benchmark

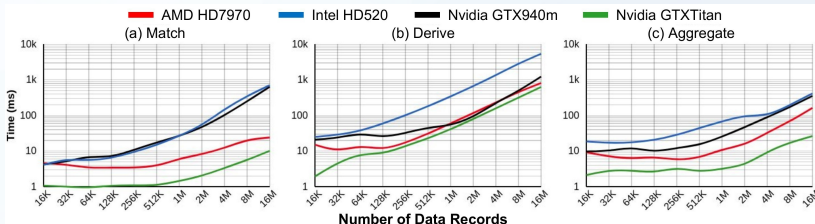
Desempeño en la transformación de datos



**Figura:** Tiempo promedio de P4 y las demás librerías para procesar las tareas de *Match*, *Derive* y *Aggregate*. Fuente: [6]

# Benchmark

Desempeño en la transformación de datos

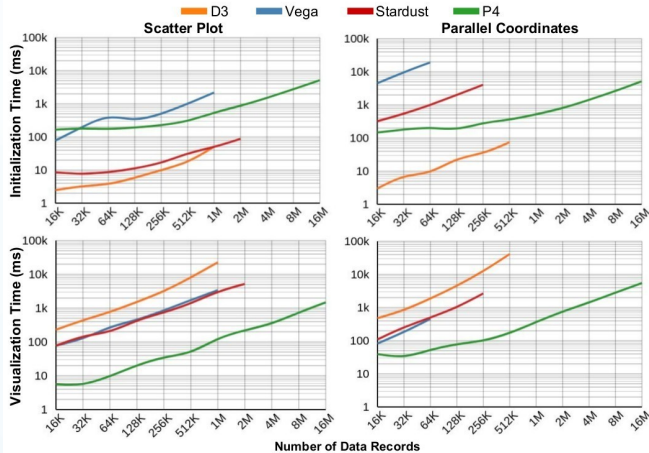


**Figura:** Tiempo promedio de P4 y las demas librerías para procesar las tareas de *Match*, *Derive* y *Aggregate*. Fuente: [6]



# Benchmark

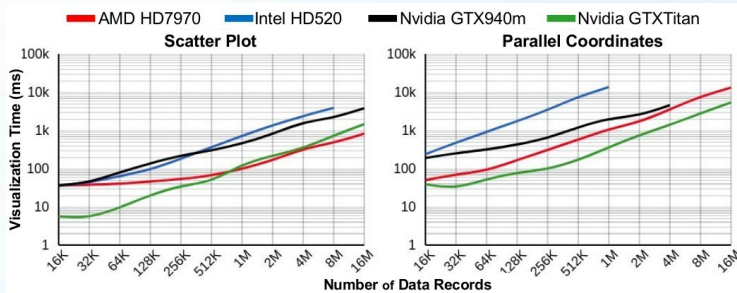
Desempeño en la transformación de datos



**Figura:** Comparación del tiempo promedio de inicialización y tiempo de visualización (tiempo por fotograma) con las demás librerías. Fuente: [6]

# Benchmark

Desempeño en la transformación de datos



**Figura:** Comparación del tiempo promedio de inicialización y tiempo de visualización (tiempo por fotograma en varios GPUs. Fuente: [6])



- ▶ Mientras que la mayoría de herramientas se enfocan en la expresividad, P4 combina el procesamiento con GPU y gramáticas declarativas.
- ▶ P4 es mas rapido que otras herramientas de transformación y visualización de datos.
- ▶ El desempeño de P4 depende fuertemente de la capacidad del GPU.
- ▶ Comparado con P4,  $D^3$  tiene mejor expresividad para la visualización de datos.



- [1] M. Card, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.
- [2] M. Bostock, V. Ogievetsky, and J. Heer, “D<sup>3</sup> data-driven documents,” *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.
- [3] H. Wickham, *ggplot2: elegant graphics for data analysis*. Springer, 2016.
- [4] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, “Reactive vega: A streaming dataflow architecture for declarative interactive visualization,” *IEEE transactions on visualization and computer graphics*, vol. 22, no. 1, pp. 659–668, 2015.



- [5] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, “Vega-lite: A grammar of interactive graphics,” *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 341–350, 2016.
- [6] J. K. Li and K.-L. Ma, “P4: Portable parallel processing pipelines for interactive information visualization,” *IEEE transactions on visualization and computer graphics*, 2018.

