# Introduction to Parallel and Distributed Processing

## Introduction to Apache GraphX

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

# Suggested Reading

- GraphX Programming Guide
  http://spark.apache.org/docs/latest/graphx-programming-guide.html/

# GraphX

- Library on top of Spark for large graph processing
- Pregel API plus extra functionality (e.g. subgraphs)
- Currently only Scala support (native to Spark)

# GraphX Idea

- User describes types associated with vertices and edges
- The resulting multigraph is stored as a pair of dedicated RDDs
- Basic storage optimization for basic data-types

# GraphX Operators

- GraphX provides a set of basic operators implementing operations on graphs (e.g. `inDegree`, `mapVertices`, `connectedComponents`)
- These implementations are highly optimized

# GraphX Graph

```
1  import org.apache.spark._
2  import org.apache.spark.graphx._
3
4  object SimpleApp {
5    def main(args: Array[String]) {
6      val conf = new SparkConf().setAppName("GraphX App")
7      val sc = new SparkContext(conf)
8
9      val V: RDD[(VertexId, (Int, Int))] =
10            sc.parallelize(Array((1L, (7, -1)), (2L, (3, -1)),
11                                 (3L, (2, -1)), (4L, (6, -1))))
12     val E: RDD[Edge[Boolean]] =
13           sc.parallelize(Array(Edge(1L, 2L, true),
14                                Edge(1L, 4L, true),
15                                Edge(2L, 4L, true),
16                                Edge(3L, 1L, true),
17                                Edge(3L, 4L, true)))
18
19     val G = Graph(V, E)
20   }
21 }
```

# GraphX Pregel API

- Implemented as an operator:

```
def pregel[A](initialMsg: A,
              maxIterations: Int = Int.MaxValue,
              activeDirection: EdgeDirection = EdgeDirection.Either)
             (vprog: (VertexId, VD, A) => VD,
              sendMsg: (EdgeTriplet[VD, ED]) => Iterator[(VertexId, A)],
              mergeMsg: (A, A) => A): Graph[VD, ED]
```

- Takes two lists of arguments: first with configuration, second with the actual function implementations
- `EdgeTriplet` view: edge and attributes of adjacent vertices:

```
var attr: ED
var dstAttr: VD
var dstId: VertexId
var srcAttr: VD
var srcId: VertexId
```

- Executes for the fixed number of steps or no messages

# Pregel API Example

```
1  val initMsg = Int.MaxValue
2
3  def vprog(id: VertexId, attr: (Int, Int), msg: Int):
4      (Int, Int) = {
5    if (msg == initMsg)
6      attr
7    else
8      (msg min val._1, attr._1)
9  }
10
11 def sendMsg(tri: EdgeTriplet[(Int, Int), Boolean]):
12     Iterator[(VertexId, Int)] = {
13   val src = tri.srcAttr
14
15   if (src._1 == src._2)
16     Iterator.empty
17   else
18     Iterator((tri.dstId, src._1))
19 }
20
21 def mergeMsg(msg0: Int, msg1: Int): Int = msg0 min msg1
```

# Pregel API Example

```
1  // ...
2
3  object SimpleApp {
4    def main(args: Array[String]) {
5      // ...
6      val G = Graph(V, E)
7
8      val Gm = G.pregel(initMsg,
9                        Int.MaxValue,
10                       EdgeDirection.Out)(
11                       vprog,
12                       sendMsg,
13                       mergeMsg)
14   }
15 }
```

# Pregel SSSP

```
1  val G: Graph[Long, Double] = ...
2  val sId: VertexId = 0
3
4  val iG = graph.mapVertices((id, _) =>
5           if (id == sId) 0.0 else Double.PositiveInfinity)
6
7  val sssp =
8    iG.pregel(Double.PositiveInfinity)(
9              (id, dist, newDist) => dist min newDist,
10             tri => {
11               if (tri.srcAttr + tri.attr < triplet.dstAttr)
12                 Iterator((tri.dstId, tri.srcAttr + tri.attr))
13               else
14                 Iterator.empty },
15             (msg0, msg1) => msg0 min msg1)
```

# For Fun

- Implement your own version of connected components using GraphX.