A Numeric Parallel DSL in Scala

Combining Concise Syntax and Parallel Performance

Eivind Barstad Waaler

Bekk/UiO

October 1, 2009

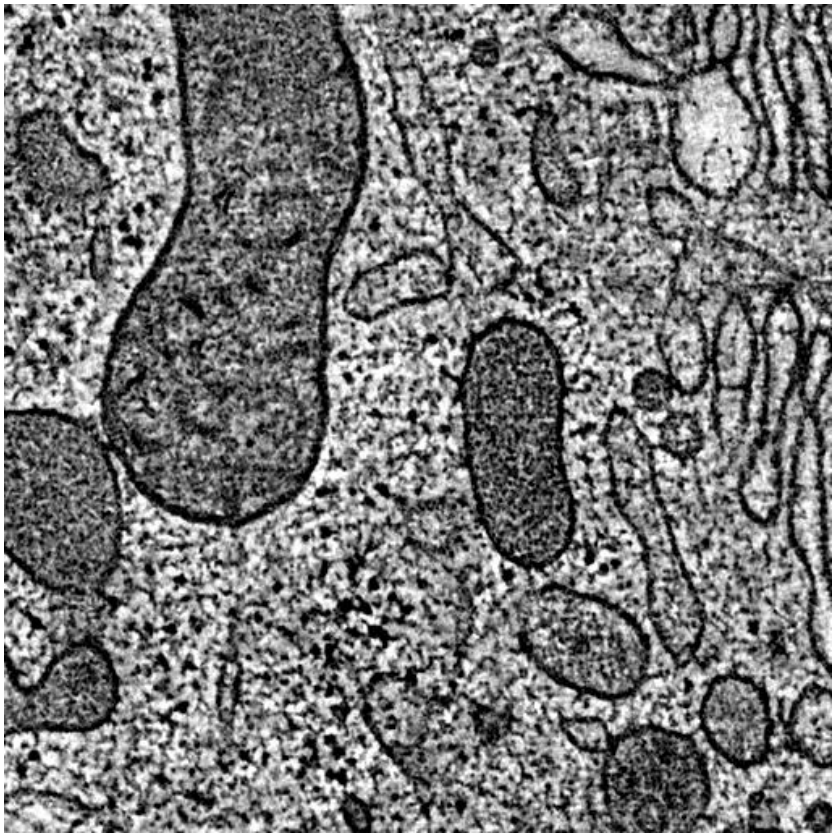
Outline

DSL for Image Processing

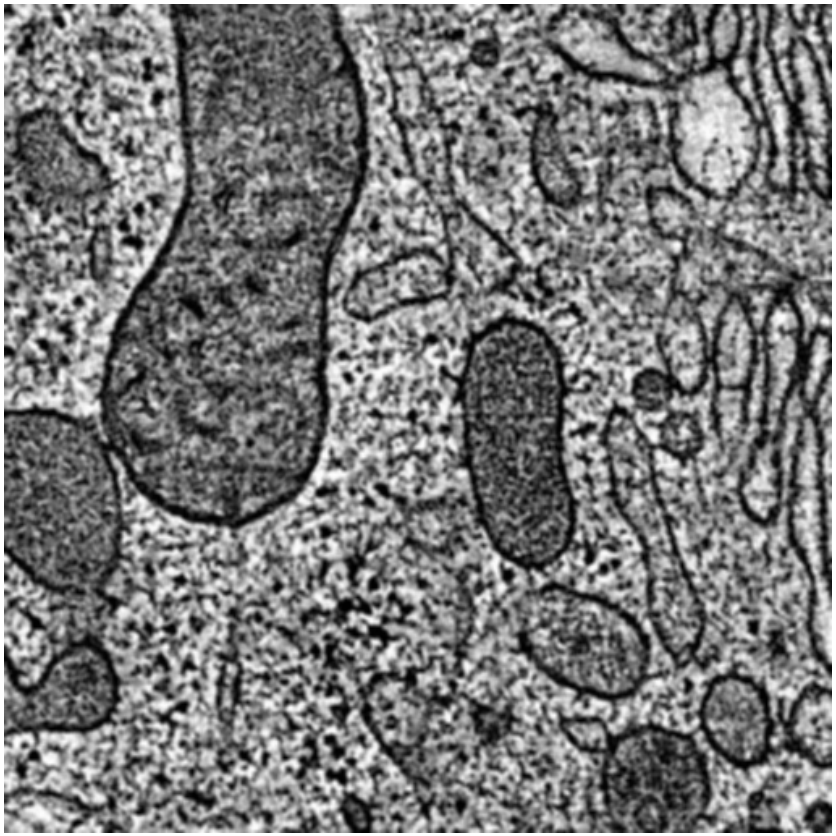
- Simple image processing examples
- Simple DSL Syntax
- Multi-threaded performance
- Inspired by MATLAB:

```
img = imread('cell.jpg');
imgAvg = medfilt2(img);
imwrite(imgAvg, 'cell_avg.jpg');
```

Example – Blur by Average Filter

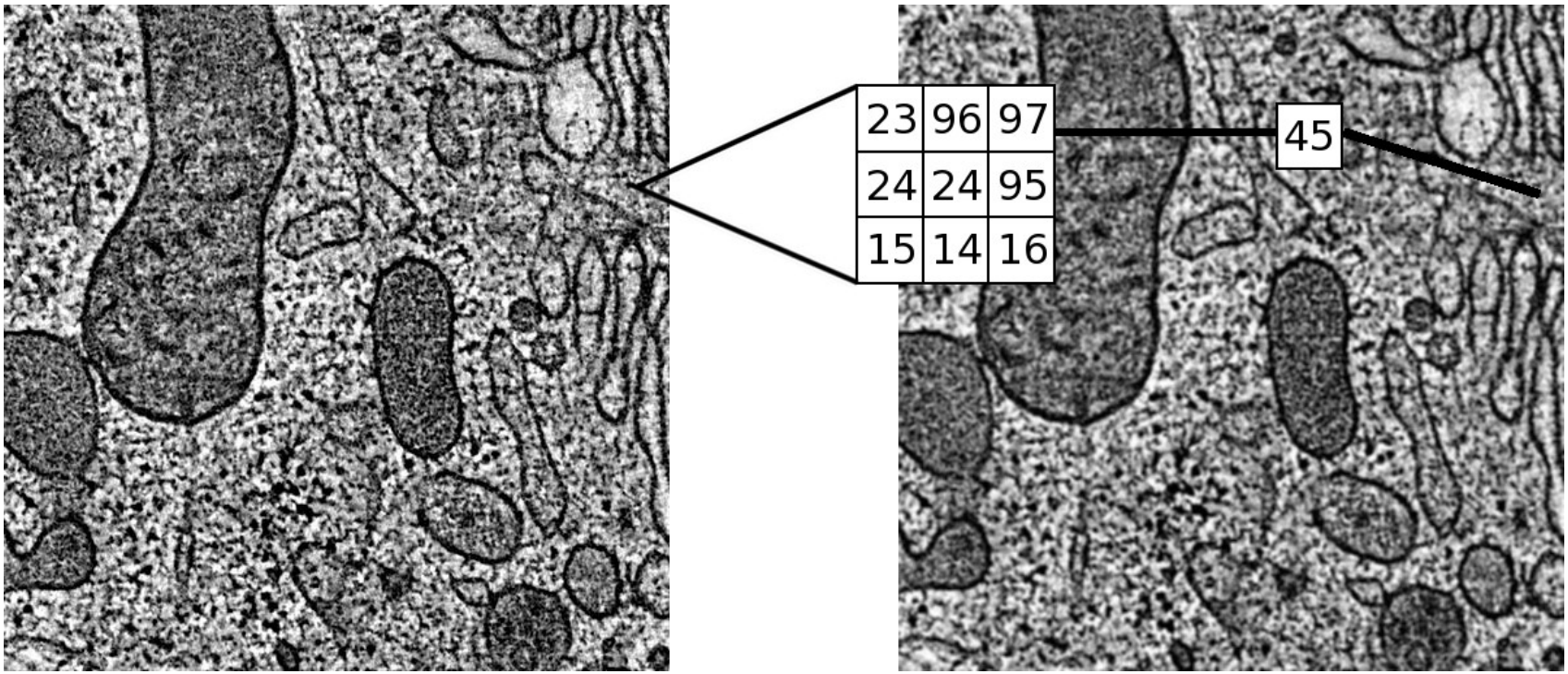


(a) Original



(b) Blurred

3x3 Average Filter



Scala Code Example

- MATLAB:

```
img = imread('cell.jpg');
% 3x3 neighborhood default
imgAvg = medfilt2(img);
imwrite(imgAvg, 'cell_avg.jpg');
```

- Scala:

```
// Imports...
val img = loadImageCP("/cell.jpg")
val se = StrEl(Square, 3)
val imgAvg = avg(img, se)
saveImage(imgAvg, "/cell_avg.jpg")
```

Loading Image – Java Integration

- Using `ImageIO` and `File` from Java

```
import java.io.File
import javax.imageio.ImageIO
import simage.structs.{Image, Matrix}
...
def loadImageCP(n: String): Image = {
  val img = ImageIO.read(getClass.getResourceAsStream(n))
  val w = img.getWidth
  val db = img.getRaster.getDataBuffer
  val data =
    for (i <- 0 to db.getSize - 1) yield db.getElem(i)
  Image(Matrix(w, data.toList))
}
```

Creating a Structuring Element

- Magic apply method
- Pattern matching

```
object StrEl {
  import StrElType._

  def apply(t: StrElType, num: Int): StrEl[Int] = {
    def ones(n: Int) = (for(i <- 1 to n) yield 1).toList
    t match {
      case HLine => new StrEl(num, ones(num))
      case VLine => new StrEl(1, ones(num))
      case Square => new StrEl(num, ones(num * num))
    }
  }
} // val se = StrEl(Square, 3)
```

Computing Average for Filter

```
def avg(img: Image, se: StrEl[Int]): Image =
  img.seOp(se, (seq) => seq.reduceLeft(_ + _) / seq.size)

protected def seOp(se: StrEl[Int], op: (Seq[Int]) => Int,
  fillMissing: Boolean, fill: Int, region: ImagePart) = {
  def pointOp(col: Int, row: Int) = {
    ...
    op(tmp)
  }
  val win = region.win
  val imgVals = for (row <- ...; col <- ...) yield {
    pointOp(col, row)
  }
  Image(Matrix(width, imgVals.toList))
}
```

Run in Parallel

```
def parallel[T](obj: Splittable[T],
  op: (T) => Splittable[T]): Splittable[T] = {
  obj.split match {
    case Array(region) => op(region)
    case regions: Array[T] => {
      val futures = for(region <- regions) yield future {
        op(region)
      }
      val results = awaitAll(5000, futures: _*)
      results.reduceLeft(_ merge _)
    }
  }
}
```

Parallel Average Method

```
class ImagePart ...

class Image extends Splittable[ImagePart] ...
  def split: Array[ImagePart] = ...
  def merge(o: Splittable[ImagePart]): ...

def avg(img: Image, se: StrEl[Int]): Image = {
  val splittable = parallel(
    img,
    img.partialSeOp(
      se,
      (seq) => seq.reduceLeft(_ + _) / seq.size,
      _: ImagePart))
  splittable.asInstanceOf[Image]
}
```

Conclusion and Future Thoughts

- Conclusion:
  - Many cool DSL features
  - Parallel computing with futures

Swarm:

- "Move the computation, not the data"
- Scala 2.8 – Delimited Continuations Plugin

OpenCL:

- Utilize GPU for parallel programming
- OpenCL4Java $\longrightarrow$ ScalaCL

Q & A

Questions, comments, examples?

- eivindw [@] gmail
- http://twitter.com/eivindw
- http://github.com/eivindw/simage