Bulk Synchronous Visualization

Lars Ailo Bongo (larsab@cs.uit.no)

Department of Computer Science, University of Tromsø, Norway

Background

- Data processing pipeline for integrating expression data
  - Dataset processed independently (integration at end)
  - How does each pipeline stage change the data?
- Need for per dataset visualizations
  - Often simple visualizations
  - Histograms, heatmaps, line charts…
  - Should be fast to implement

Goals

- MATLAB-like plotting
- Utilize high resolution
- Utilize multi-core CPUs
- Sequential programming
- Portability
- Interactive performance

Supported libraries

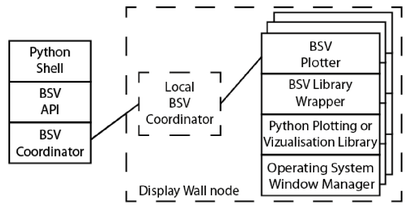
- Pyplot/ matplotlib
- Pyglet
- Tkinter/ PIL

Approach

1. Split data into independent parts
2. Visualize each part in a separate window
3. Write visualization functions at run-time
4. Synchronize all visualizations from a coordinator
   - All visualizations are clones of coordinator
5. Control which windows are visible
   - Very fast window management
   - Predict which are to be shown

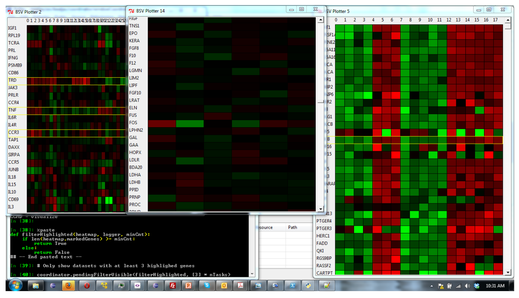
Design and Implementation

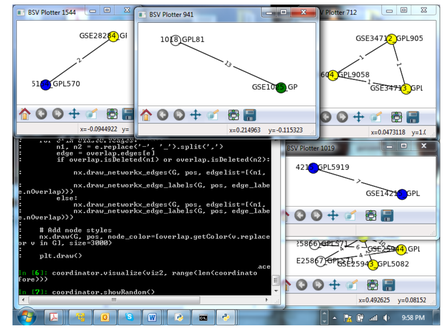
- Interactive shell → programmatic control over visualization
- One process with one window per part
  - Few are visible
  - Many are running
  - Most are not running
  - All are clones of coordinator process
- Schedule running processes for efficient window switching
- Implemented in Python
  - Can write new functions at runtime and send to running visualization processes
  - Can save a list of executed functions

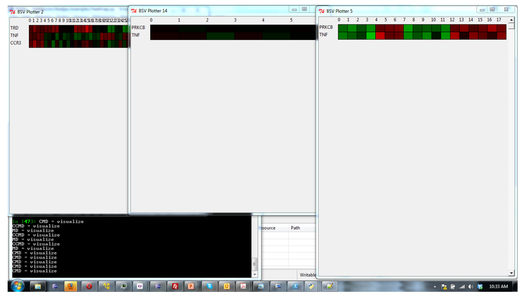


Assumptions

- Many biological visualizations can (and must) be split into independents parts
  - For example expression values for each dataset
- Visualizations must be synchronized
  - For example by highlighting certain genes
- Modern hardware and operating systems provide
  - Lots of DRAM
  - Lots of CPU cycles
  - Copy-on-write address space cloning
  - Low overhead process creation







Code examples

Start coordinator and plotters

```
# Create data structure for holding expression values and
# window state
heatmap = HeatmapWindow()
# Read a gene ID to gene name mapping file
heatmap.geneId2Name = loadMapFile(mapFilename)
# Create list with expression value filenames
filenames = os.listdir(dataDir).sort()
# Import coordinator class
from bsv.coordinator.coordinator import BSVCoordinator
# Import Tkinter wrapper functions
from bsv.utils.bsv2tkinter import createWindow, guiLoop,\
    closeWindow, setWindowPosSize, getWindowPosSize,\
    setWindowVisible
# Start coordinator with 20 windows, and 60 running processes
coordinator = BSVCoordinator(20, 60, createWindow, guiLoop,\
    closeWindow, setWindowVisible, getWindowPosSize,\
    setWindowPosSize, heatmap)
# Start the 60 plotter processes
coordinator.startPlotters()
# Load data on each plotter (using a dummy visualization
# function)
coordinator.visualize(loadData, filenames)
# Create heatmap visualization on each plotter
coordinator.visualize(vizHeatmap, [None] * len(filenames))
```

Visualization function

```
def highlightGene(heatmap, logger, geneID):
    # Get row index of gene to highlight
    rowIndex = heatmap.geneIDs.index(geneID)
    # Draw yellow rectangle around highlighted row in heatmap
    y = heatmap.rowHeight * (rowIndex + 1)
    heatmap.canvas.create_rectangle(0, y,
        heatmap.canvasWidth, y + canvas.rowHeight,
        outline='yellow')
    heatmap.canvas.update()
    # Maintain list of highlighted genes
    heatmap.highlightedGenes.append(geneID)
```

Filter function

```
def filterHighlightGenes(heatmap, logger, minCnt):
    if len(heatmap.highlightedGenes) >= minCnt:
        # This window can be shown
        return True
    else:
        # This window should be hidden
        return False
```

APIs

Coordinator interface:

- [start, stop]Plotters
- visualize
- layoutGrid, layoutParameter
- show[Next, Previous, Random, First, Index, Indexes]
- filterVisible, pendingFilterVisible, filterOff
- filterKill
- gather, gatherLogFiles
- executeLocally

Library wrapper interface:

- createWindow
- guiLoop
- closeWindow
- setWindowVisible
- getWindowSizePos
- setWindowSizePos

For source code and more information:

http://www.cs.uit.no/~larsab/bsv/