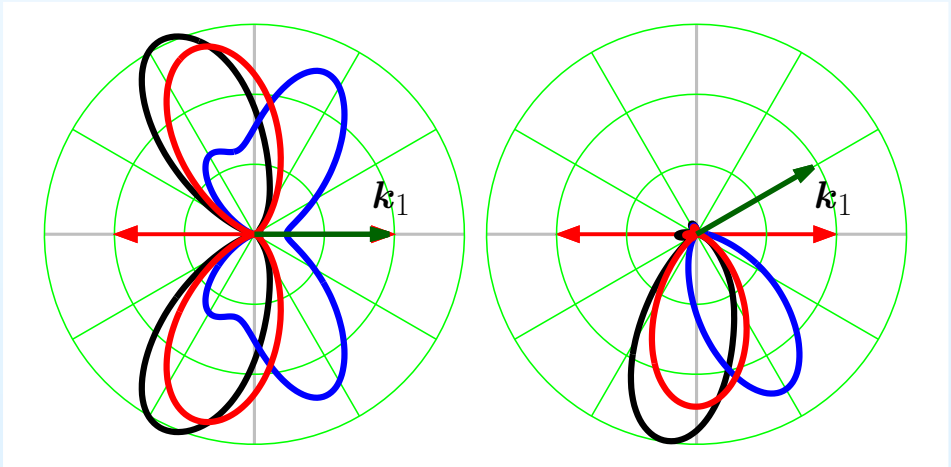
Plotting with gnuplot

Xiaoxu Guan

High Performance Computing, LSU

September 28, 2016



Overview

- What is **gnuplot** and what can **gnuplot** do for us?
- Fundamental concepts in gnuplot
- Plot functions, curves, points, . . .
- Built-in functions and functions user defined
- Customize axes, labels, and legends
- Other types of 2D plots
- Gnuplot with data files
- Borders and multiple plots
- Customize postscript plots
- Three-dimensional and contour plots
- Further reading

What is gnuplot?

- Gnuplot is one of the open-source cross-platform plotting systems;
- Note **gnuplot** is NOT a **GNU** project;
- It was written in C/C++, and other languages;
- Gnuplot is mostly a **command-line** driven plotting program;
- It supports many output formats of figure: (E)PS, PDF, SVG, LaTeX, PNG, ...
- Gnuplot is an implementation and a **scripting** system as well;
- It supports both batch and interactive modes;
- Gnuplot is a **backbone** of many other GUIs tools: Maxima, GNU Octave, Kayali, ...;
- It can be called by Perl, Python, Ruby, Fortran 95, ...;
- It has widely been used to create publication-quality figures;

Fundamental concepts in gnuplot

- Remember that gnuplot is a **command-line** driven program. It can also be thought as a scripting system;
- All the commands are case sensitive. They may be abbreviated as **short** as they are not ambiguous;
- Like shell scripting, gnuplot uses **#** as comments;
- Gnuplot reserves the **backslash** (\) as the character of line continuation;
- Plot on a **canvas**:

```
set terminal { <type of terminal> <options> }
```

```
set size x,y
```

```
set output "file.name"
```

- Type of **terminals**: png, jpeg, postscript, pdf, latex, pslatex, epslatex, svg, x11, and many others;

Fundamental concepts in gnuplot

- Different terminals may have different settings;

```
set terminal postscript portrait size 6, 4 \
    enhanced monochrome "Helvetica" 16
set output "figure1.ps"              # size in inch
```

```
set term png size 600, 400    # absolute size
set size 0.5, 0.4             # scale factors
set output "figure2.png"      # size in pixel
```

```
set term pslatex color dashed dl 4 15
set size 1.0, 2.0
set output "figure3.tex"             # size in inch
```

```
set term pdf color rounded linewidth 4
set output "figure4.pdf"
# size in 5 inches by 3 inches
```

- Use **help** command within gnuplot;

Plot functions, curves, points, ...

- **plot** is the primary command for 2D figures;

```
plot <ranges> <iteration>   \
     <function> | "<datafile>" datafile-modifiers \
     axes <axes> <title-spec> with <style> ...
```

```
set xrange [a:b]
set yrange [c:d]
```

- Note that functions can be built-in or user defined (`lt=linetype` and `lw=linewidth`);

```
a = 4.0 ; b = 5.2
my_fun(x) = exp(-a*x) * sin(b*x)
plot [-2:2] [-5:5] my_fun title "This my plot" \
                   lt 1 lw 4
```

- Define two parameters `a`, `b`, and the function `my_fun(x)`;

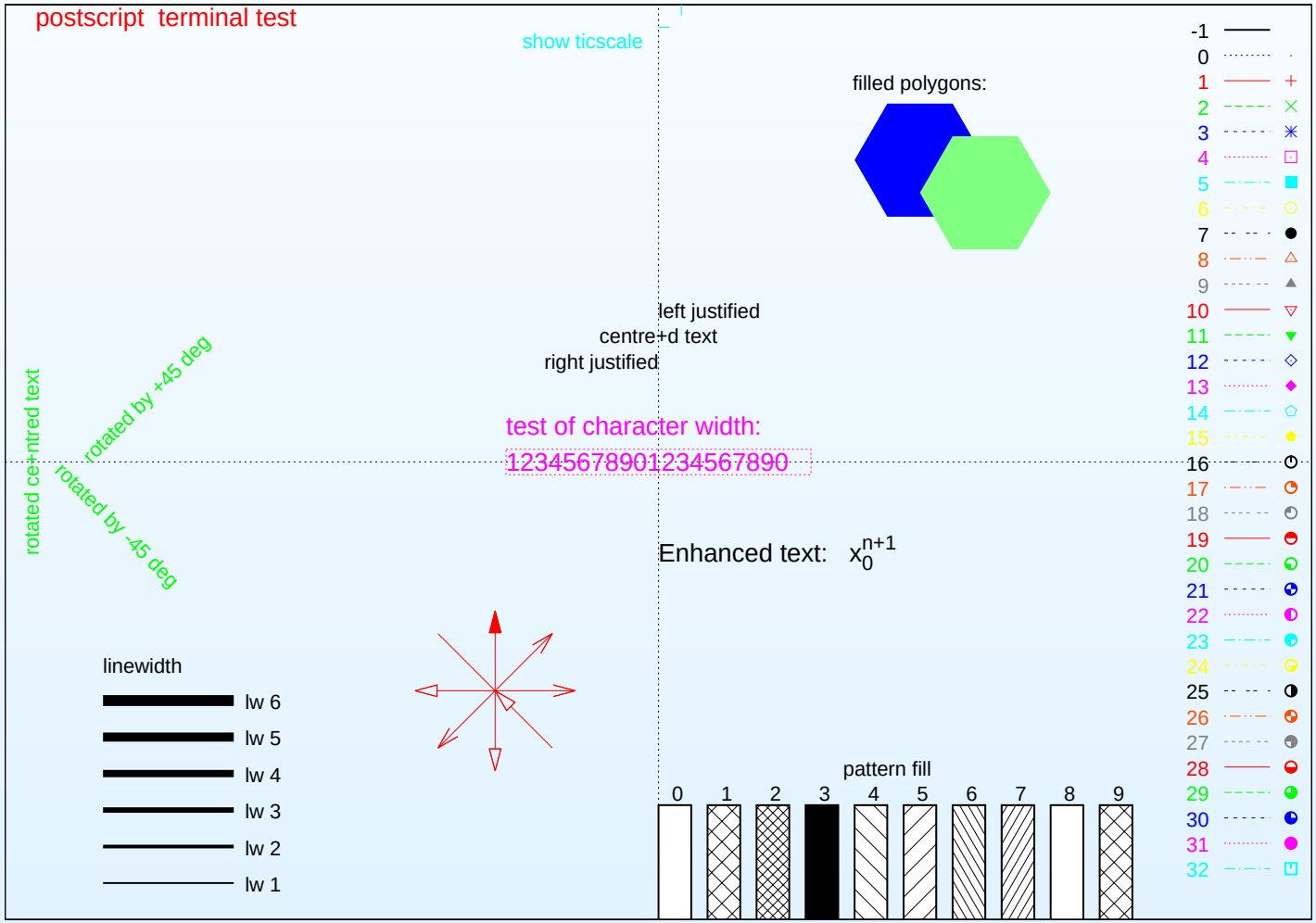
Built-in functions

- Gnuplot supports many built-in functions:

Function	Description
$\log(x)$	log function of x in natural base (e)
$\log 10(x)$	like the above, but in base 10
rand(0)	generator of pseudo random numbers (real)
sqrt(x)	$\sqrt{x}$
a**b	a^b
exp(x)	e^x
sin(x)	$\sin(x)$
tan(x)	$\tan(x)$
abs(x)	$\mathrm{abs}(x)$
sgn(x)	$+1$ if $x > 0$, or -1 if $x < 0$
gamma(x)	$\mathrm{gamma}(x)$

Plot functions, curves, points, . . .

- How to control **linetype**, **linewidth**, or **linecolor**, . . .:
- Again, this is terminal dependent: run command **test**;

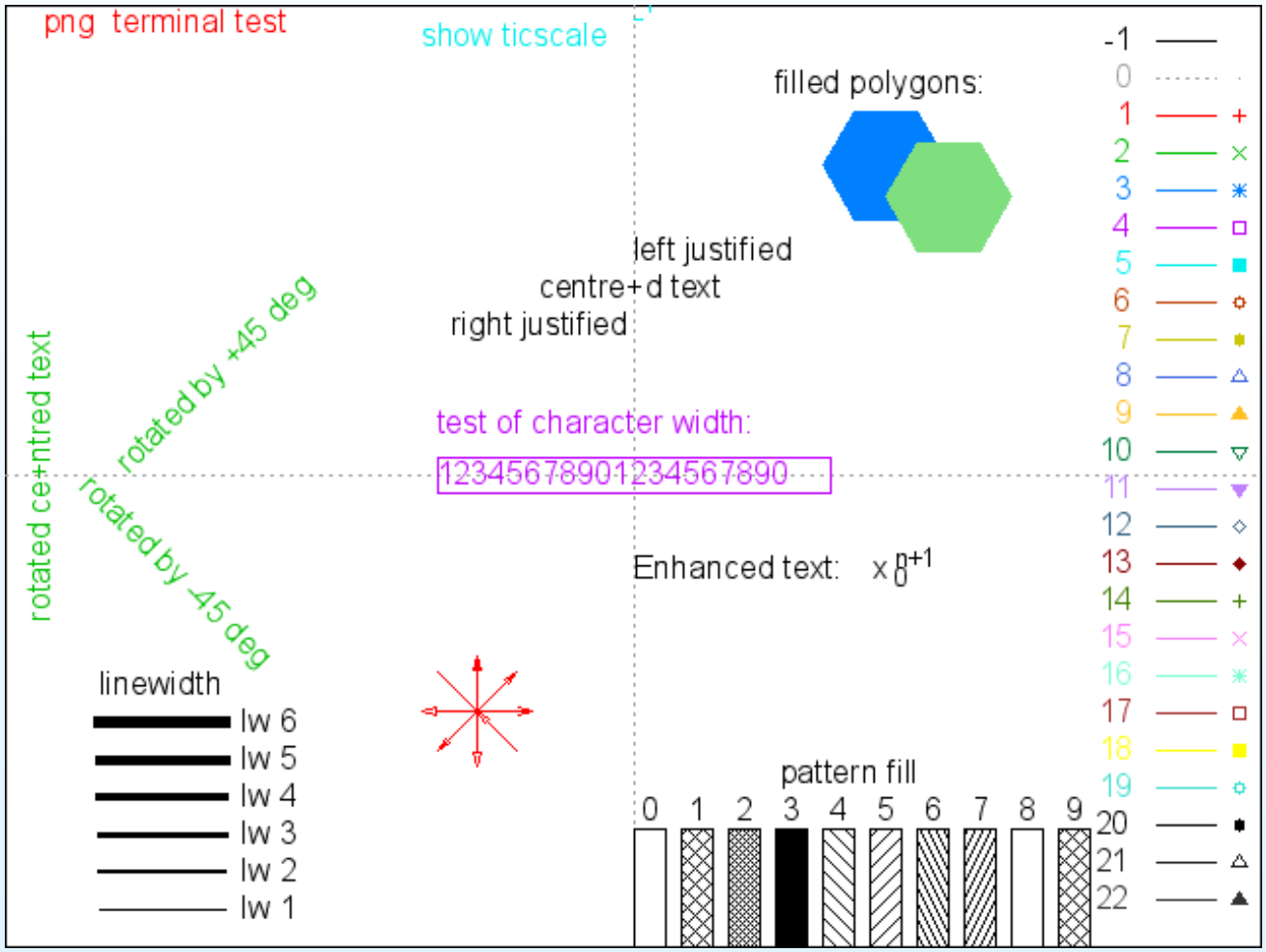


postscript terminal

It ranges from -1 to 32

Plot functions, curves, points, . . .

- How to control **linetype**, **linewidth**, or **linecolor**, . . .:
- Again, this is terminal dependent: run command **test**;



png terminal

It ranges from -1 to 22

Plot functions, curves, points, . . .

- How to control **linetype**, **linewidth**, or **linecolor**, . . .:
- Line colors can be represented either in name or in hex;

```
plot sin(x) lc rgb "blue" lt 1 lw 4, \
     tan(x) lc rgb "#800000" lt 2 lw 2 # dark brown
```

- Gnuplot allows us to define user-specified line style (`ls`): it can be reused once defined;

```
set style line 1 lt -1 lc rgb "orange" lw 2
set style line 2 lt 0 lc rgb "red" lw 3
set style line 3 lt 1 lc rgb "cyan" lw 4
plot sin(x) ls 1 title "orange", \
     tan(x) ls 2 title "red",\
     tan(x)*sin(x) ls 3 title "cyan" \
     (x*x-x+2)*sin(x) ls 3 title "cyan"
```

- Note line color can also be a variable;

Plot functions, curves, points, ...

- What about **point type** or **size**?
- Again, it depends on terminals: run **test**;

```
plot sin(x) with points pt 2 ps 3
plot sin(x) with p pt 4 ps 3
```

- How to add **points** on **lines**?

```
plot cos(x) with linespoints lt 1 lw 2 \
     lc rgb "blue" pt 2 ps 3
plot cos(x) with lp lt 1 lw 2 \
     lc rgb "green" pt 4 ps 2
```

- Generalize the definition of **line style**:

```
set style line 1 lt -1 lc rgb "orange" lw 2 \
    pt 10 ps 1                          # define ls 1
```

Customize axes, labels, and legends

- By default, gnuplot automatically draws axes, tics, labels, **legends** (also known as **keys**), etc;
- Gnuplot also allows us to customize them;

```
set xrange [1:200] ; set yrange [-1:1]
```

- Control the axis tics:

```
set xtics 50 ; set ytics 0.2      # main tics
set mxtics 5 ; set mytics 2       # minor tics
set xtics add (1)
# manually add label 1 on the x axis
```

- Control the length of tics:

```
set tics scale 2.5      # main x and y tics scales
set xtics scale 2.5     # main xtics scale
set ytics scale 2.5     # main ytics scale
```

Customize axes, labels, and legends

- Sometimes we might want to draw tic marks outwards;

```
set xtics scale 1.5 out     # default is in
```

- Control the axis labels (note **offset**):

```
set xtics font "Times New Roman,12"   # font and size
set xlabel "Number of cores" offset 0.0, 0.3
# specify the x-axis label
set ylabel "Speedup"                  # specify the y-axis label
```

- Control the formats of labels:

```
set format y "%3.1f"                  # format of y labels
set log y                             # y tics in log10
set format y "$10^{%01T}$"            # y in 10^k
```

- Cancel it? (using **unset**)

```
unset log y      # cancel the log setting of the y axis
```

Customize axes, labels, and legends

- How to control **keys** (**legends**):

```
unset key                    # don't draw the keys
set key at -5, 0.8           # draw the keys at the coordinates
set key at 0, 1.5 spacing 0.9 samplen 2
# draw the keys with specified line properties
```

- Add **labels**:

```
set label 1 at -1,0.2 "This is the first lobe"
set label 2 at 2,0.8 "T=0.4 a.u."
# define labels 1 and 2
```

- Draw **arrows**:

```
set arrow 1 from 1,0.4 to 3.5, 2.9 \
nohead filled lt 2 lw 1 lc rgb "blue" back
set arrow 2 from -1,0.0.2 to 0.5, 4.0 \
head filled lt 2 lw 1 lc rgb "dark-green" back
```

Other types of 2D plots

- Gnuplot supports many other types of 2D plots;
- Let's consider the **polar** plot:

```
set dummy t
set trange [0:pi*0.5]
set polar
plot 2-sin(t) lw 4
# define a dummy variable t and plot a function y(t)
```

- Gnuplot also supports bar plots, pie plots, error bars, loading binary figures, multiple axes, filling space between lines and other features;

```
plot "lsutiger.png" binary array=(128,128) ...
plot "myfigure" binary ...
```

- Let's see how gnuplot can handle data files;

Gnuplot with data files

- For 2D plots, the **textual** data file can be in the format of **multiple** columns. Both floating numbers and scientific notation (1.4E-2 or 2.6e+10) are recognizable;
- Add **#** in your data files whenever necessary for comments;

```
# Intel MPI lib on SuperMIC.
# 16 MPI tasks on 16 nodes.  Each node has one MPI
# task either in offload or no-offload mode.
# offload mode   NON-offload (only on host)
# column_1    column_2     column_3    column_4
      1         1710.53        1         133.06
      5          358.89        2          69.99
     10          192.72        3          48.79
     ...
    240           22.17
```

Gnuplot with data files

- What we want is to plot the speedup ($S_n = T_1/T_n$) against the number of threads on MIC and CPU;
- Column index starts from 1;

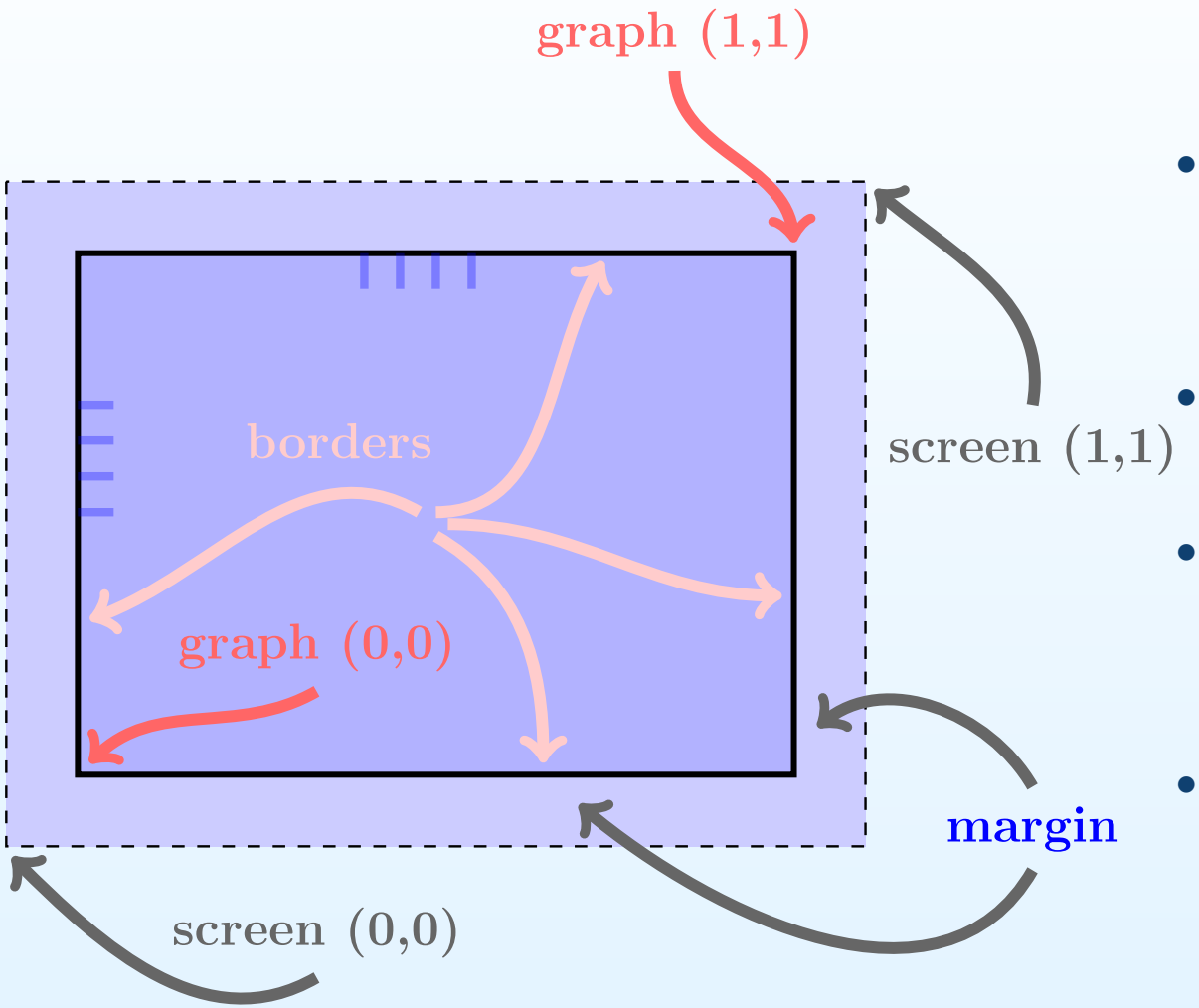
```
tmic = 1710.53 ; tcpu = 133.06
plot "timing.dat" using 1:2 w l lt 2 lw 3
plot "timing.dat" u 1:(tmic/$2) w l lt 2 lw 3
```

- **Multiple** lines on the same plot. If necessary, we can use the same or other data files;
- Gnuplot supports the **path** in filename as well;

```
tmic = 1710.53 ; tcpu = 133.06
plot "timing.dat" u 1:(tmic/$2) w l lt 2 lw 3, \
"" u 3:(tcpu/$4) w l lt 3 lw 2 \
pt 2 ps 3              # add points
```

Borders and multiple plots

- More terminologies: **screen**, **graph**, **origin**, **margin**, etc;

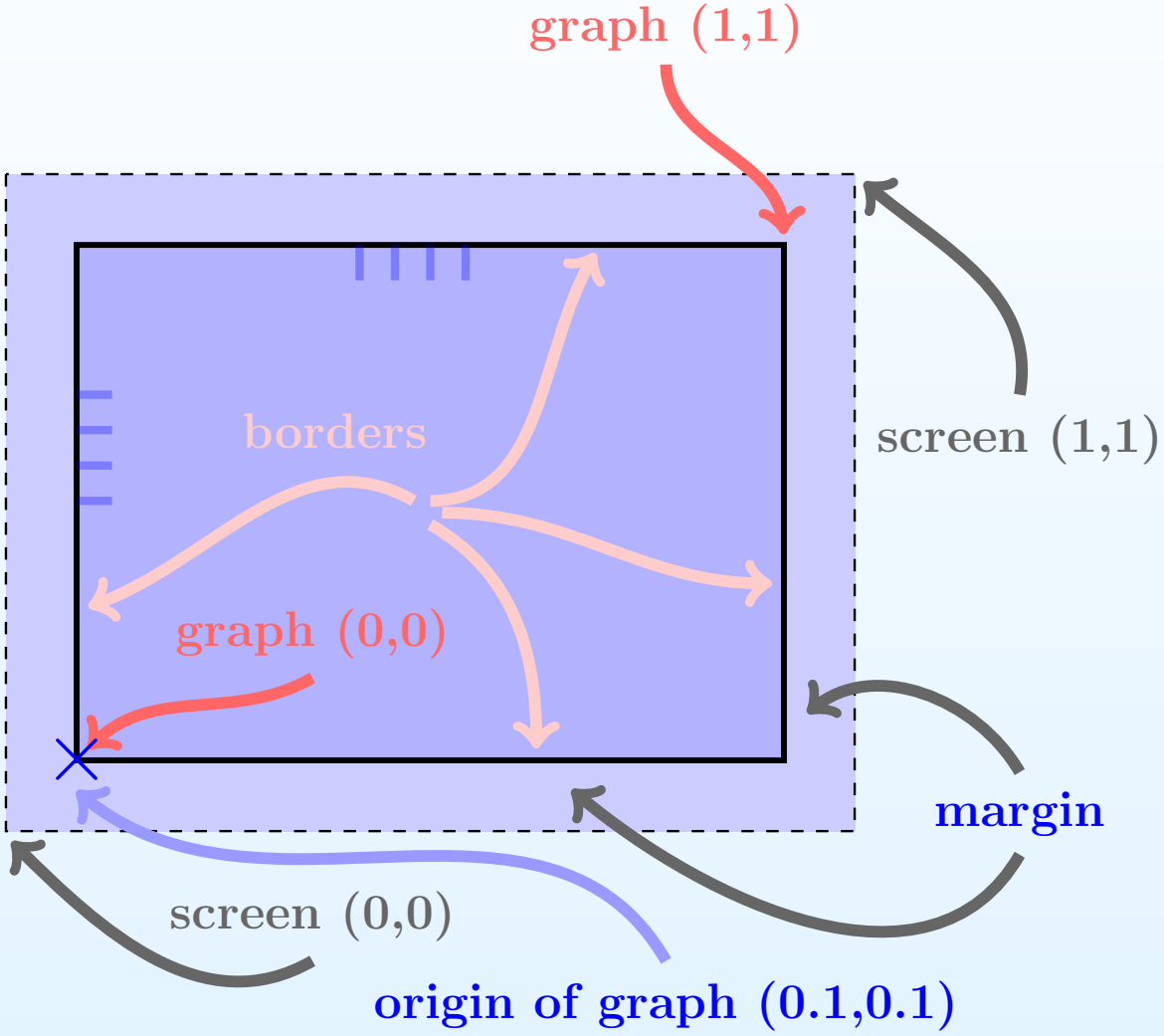


- **Screen** (0,0), (1,1) is the entire area (**canvas**).
- Borders mean four axes with tics.
- Borders define the **graph** area: (0,0),(1,1).
- **Margin** area is for axis labels and tics.

- What about the origin?

Borders and multiple plots

- More terminologies: **screen**, **graph**, **origin**, **margin**, etc;



- **set origin (0.1,0.1)**
- In the **screen** coordinates, set the **origin** of **graph**.
- Origin is useful to **multiple** plots.
- Each panel can have its own **origin**.

Borders and multiple plots

- **Borders** in gnuplot mean four **axes**;
- Gnuplot allows us to control four borders using **mask** values;

```
set border <options>          # define borders
```

- **Mask** values: bottom is 1, left 2, top 4, and right 8.

```
unset border    # important!
set border 3    # draw bottom and left axes (3=1+2)
set border 7    # draw bottom, left, and top axes (7=1+2+4)
set border 15   # same as the default
show border
```

- Note the above setting has nothing to do with tic marks;

```
set xtics nomirror    # turn off x tics mirror
set ytics nomirror    # turn off y tics mirror
```

- Borders are drawn on the **top** of all plot elements (default);

Borders and multiple plots

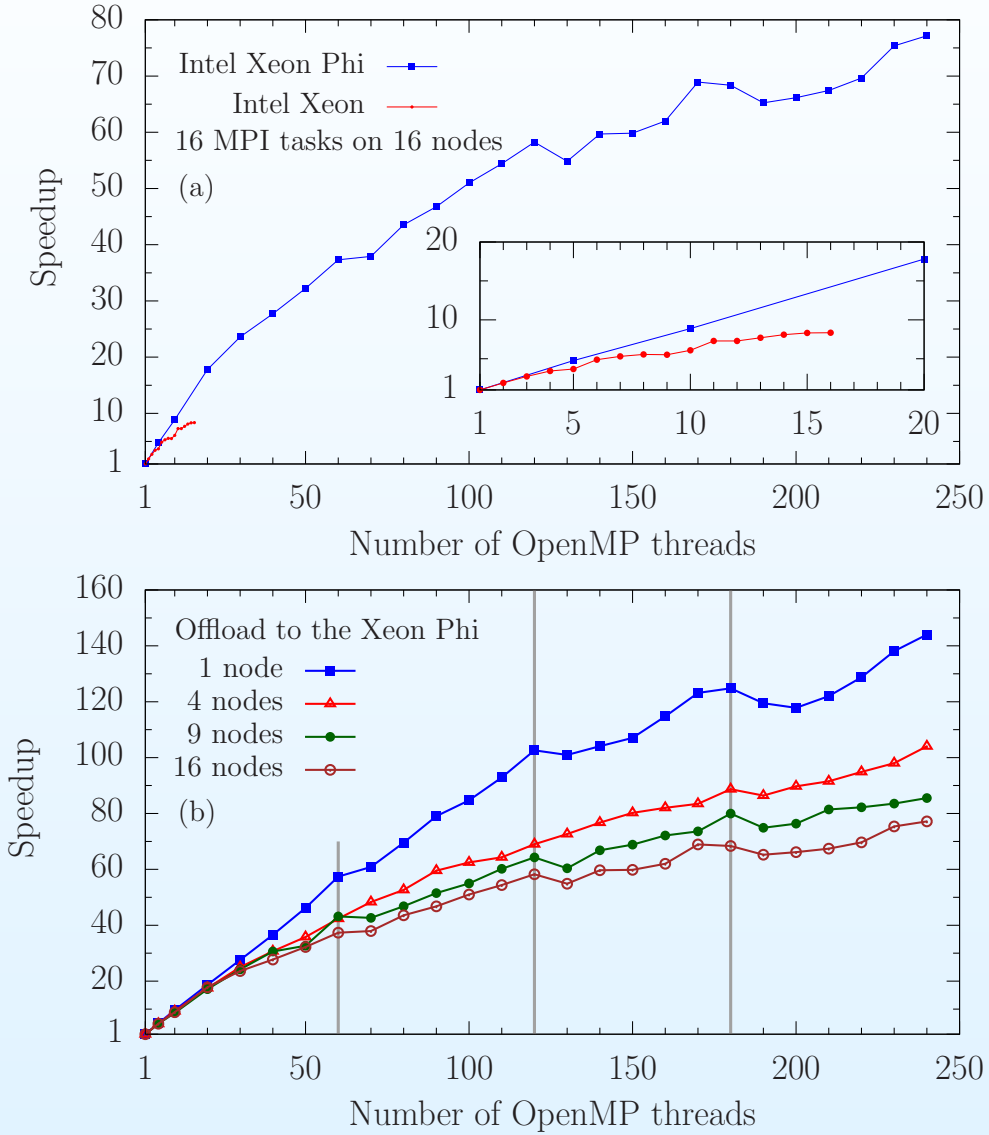
- On the same **canvas**, draw **multiple** plots on several panels;
- We may organize plots in terms of $m \times n$ (# rows $\times$ # cols) panels. Each panel may be customized separately;

```
set multiplot          # only once
set origin 0,0         # for the canvas or a panel
# the first panel
set origin 0.5/5.0, -1.5/3.5
set lmargin 0          # left margin
set rmargin 0          # right margin
set tmargin 0          # top margin
set bmargin 0          # bottom margin

# the second panel
set origin 0.5/5.0, 0.0/3.5
...
```

Borders and multiple plots

- The output of the script **multiplot.gpl**:



- The terminal is **pslatex**.
- Define a **canvas** first on which all panels will be drawn.
- Generally, we can use **set multiplot** to organize multiple panels in any way we want.
- Each panel can have its **own origin**, **size**, and **margins**, etc.

3D surfaces and contours

- The data file needs to be **blocked** and each block separates by a blank line:

$x_1 \quad y_1 \quad z_{11}$
$x_1 \quad y_2 \quad z_{12}$
$x_1 \quad y_3 \quad z_{13}$
$\ldots$
$x_1 \quad y_{\max} \quad z_{1\max}$
[blank line]
$x_2 \quad y_1 \quad z_{21}$
$x_2 \quad y_2 \quad z_{22}$
$x_2 \quad y_3 \quad z_{23}$
$\ldots$
$x_2 \quad y_{\max} \quad z_{2\max}$
[blank line]
$\ldots$

- A **blank line** between blocks.
- **Whitespace** between columns.
- The sequence of columns is **irrelevant**.
- It is better to use the terminal of **postscript** (or other vector terms).
- In most cases, **contour** plots are more useful.
- The gnuplot command is **splot** <options>

3D surfaces and contours

- Gnuplot supports "**heat map**" plot;
- Color mapping is used to represent one of the columns;

```
set term postscript enhanced color rounded 21
set pm3d map
set size square
```

- The **splot** syntax:

```
splot "myfile_1.dat" u 1:2:3
splot "myfile_2.dat" u 2:1:3
splot "myfile_3.dat" u 2:1:$3/1.e-4
splot "myfile_4.dat" every 2 u 2:1:($3)/1.e-4
```

- Control **colorbox**:

```
set colorbox horizontal user \
origin 0.2,0.11 size 0.62,0.015     # also vertical
```

3D surfaces and contours

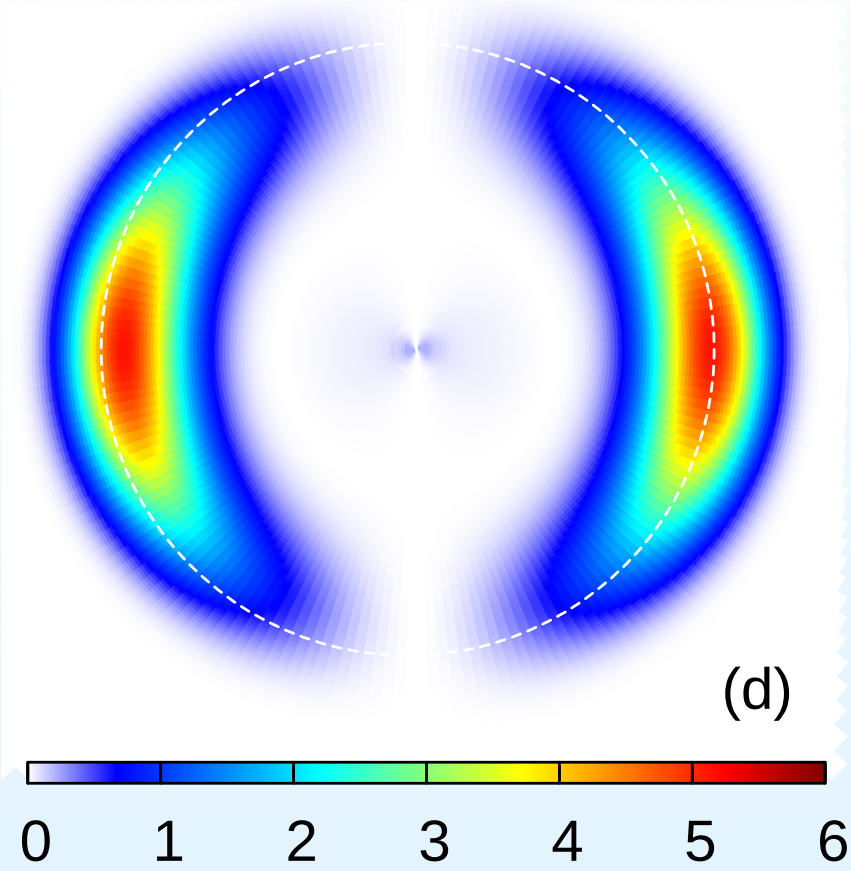
- We now know how to control the position and orientation of **colorbox**, etc;
- Let's consider the color scheme: **palette**;
- Some useful examples of palette:

```
set palette rgb 21,22,23  # hot (black-red-yellow-white)
set palette negative rgb 21,22,23
set palette rgb 33,13,10
    # rainbow (blue-green-yellow-red)
set palette rgb 34,35,36
    # AFM hot (black-red-yellow-white)
```

```
set palette defined ( 0 "#000090",1 "#000fff",\
        2 "#0090ff", 3 "#0fffee", \
        4 "#90ff70", 5 "#ffee00", \  # default matlab
        6 "#ff7000", 7 "#ee0000", 8 "#7f0000" )
```

3D surfaces and contours

- The output of **map3d.gpl**:



- `set multiplot`
- `set size square`
- `set pm3d map`
- `unset border`
- `set colorbox ...`
- `set palette defined ...`
- `splot 'map3d.dat' every 2 u ($1)*sin($2) : ($1)*cos($2) : ($3)/($1)/($1)/1.e-4`
- `...`

Further reading

- **Gnuplot homepage:** http://www.gnuplot.info
- **Not So FAQs:** http://lowrank.net/gnuplot/index-e.html
- **Also a good one:** http://www.gnuplotting.org

Questions?

sys-help@loni.org

Graphics in R

- There are three plotting systems in R
  - base
    - Convenient, but hard to adjust after the plot is created
  - lattice
    - Good for creating conditioning plot
  - ggplot2
    - Powerful and flexible, many tunable feature, may require some time to master
- We will focus on the `ggplot2` package today

Basic Concepts of `ggplot2`

- An implementation of Grammar of Graphics by Leland Wilkinson, which represents abstraction of graphic objects and ideas
- Two ways of plotting
  - The `qplot` function: similar to the base plotting system with more features
  - The `ggplot` function: the engine of `qplot` - powerful, flexible and full featured
    - Make the plots in layers
      - Plot the data
      - Overlay a summary
      - Metadata and annotation

ggplot2 Components

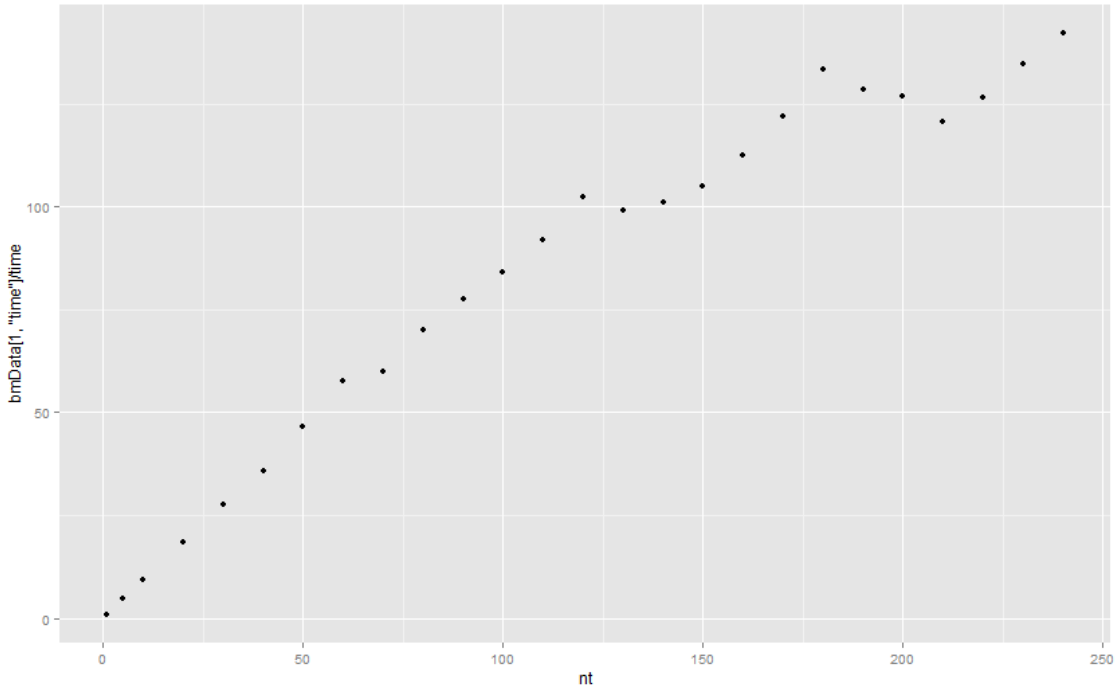
- **Data frame**
- **Aesthetic mappings**: how data are mapped to color, size
- **Geoms**: geometric objects like points, lines, shapes.
- **Facets**: for conditional plots.
- **Stats**: statistical transformations like binning and smoothing.
- **Scales**: what scale an aesthetic map uses (example: male = red, female = blue).
- **Coordinate system**

Example Data Frame

- Benchmark data on the SuperMIC cluster
  - Four columns: nt (number of threads), time, device ("mic" or "cpu"), node (number of nodes)

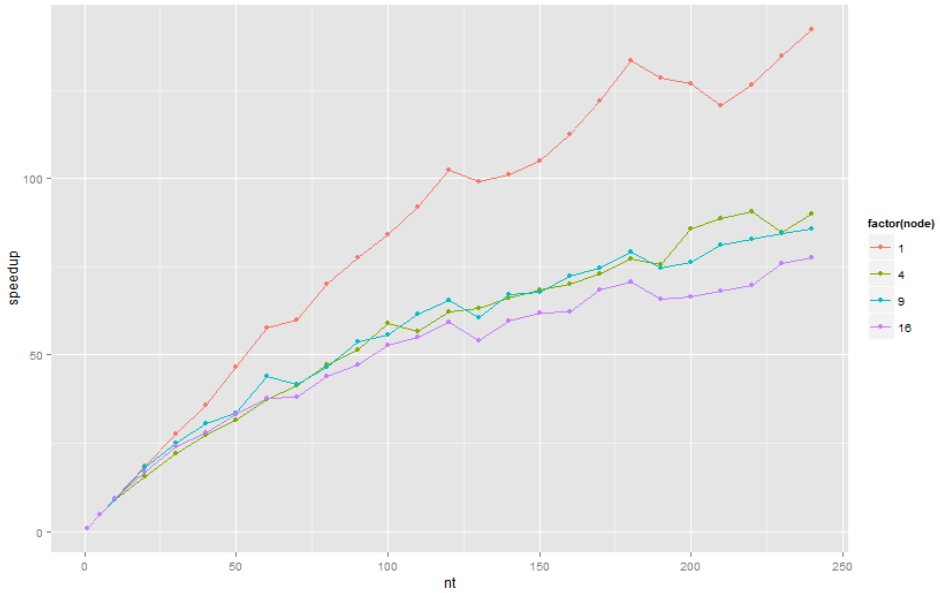
```
> head(bmData,3)
  nt       time device node
1  1 33358.562    mic    1
2  5  6894.032    mic    1
3 10  3566.916    mic    1
> tail(bmData,3)
    nt       time device node
182 18 10.237771    cpu   16
183 19  9.936815    cpu   16
184 20  9.452416    cpu   16
```

qplot Function

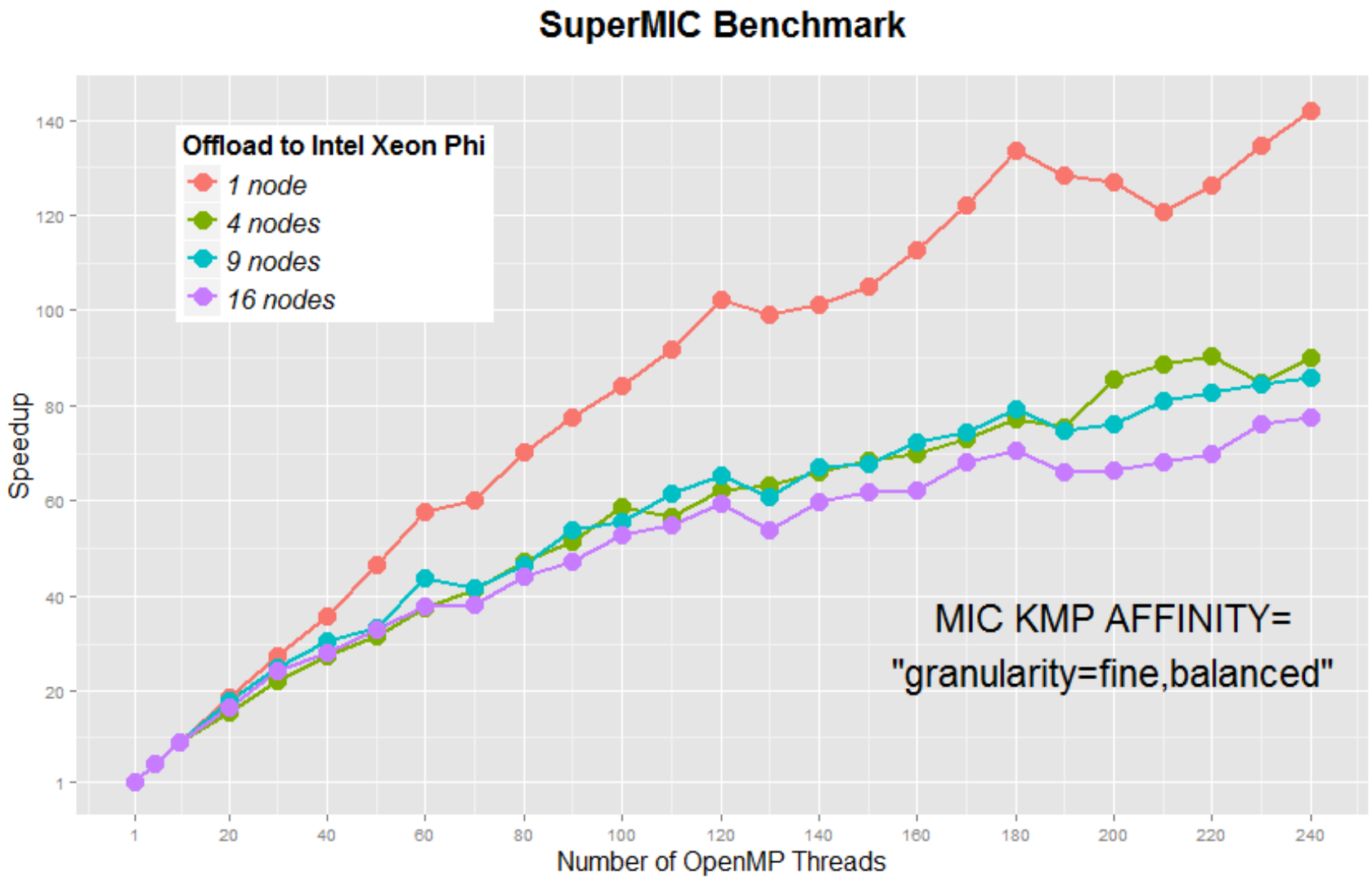



```
> qplot(nt,bmData[1,"time"]/time,
        data=subset(bmData,node==1 & device=="mic"))
```

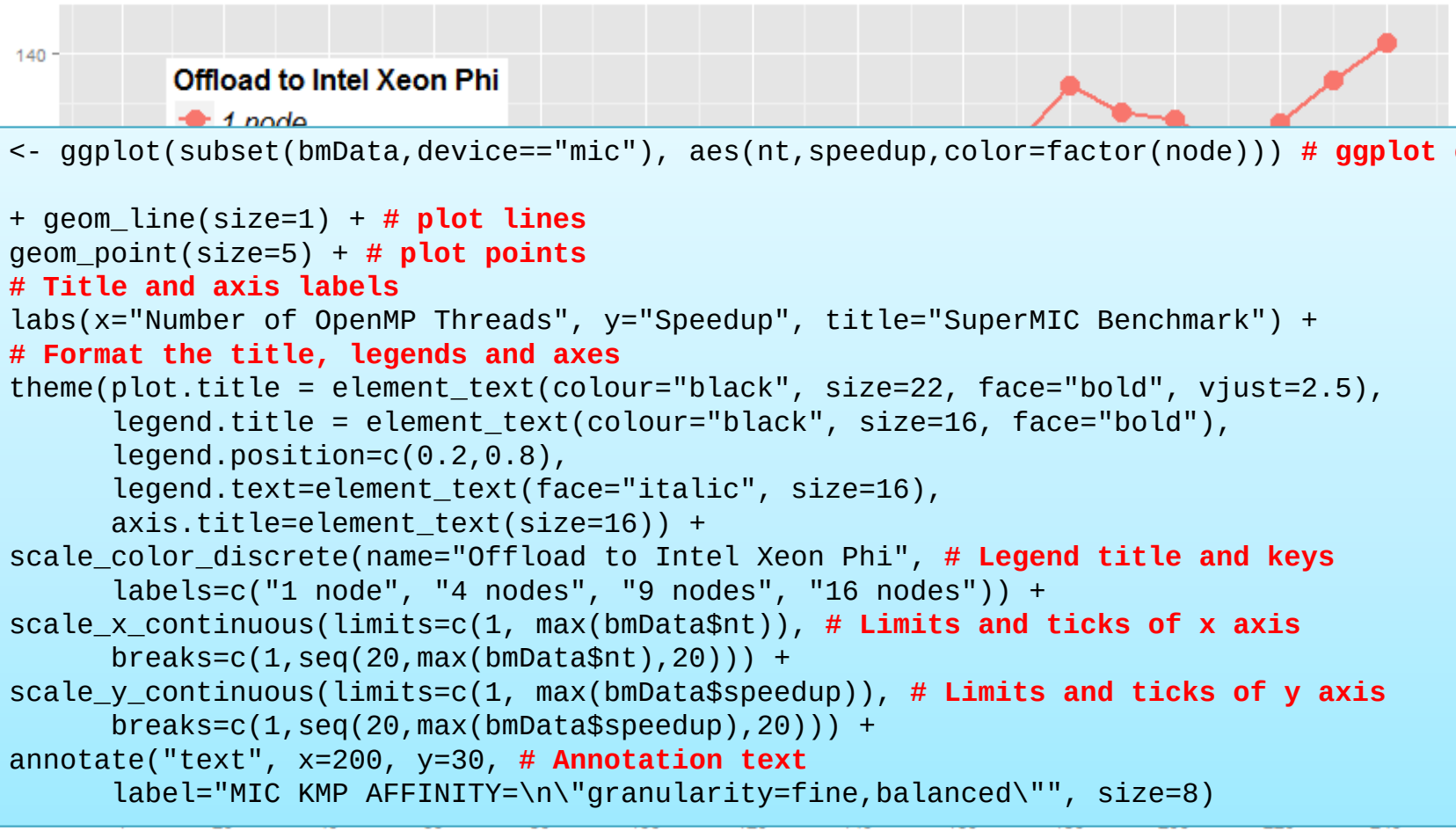
Using ggplot2




```
> g <- ggplot(subset(bmData,device=="mic"),
          aes(nt,speedup,color=factor(node)))
> summary(g)
data: nt, time, device, speedup, node [104x5]
mapping:  x = nt, y = speedup, colour = factor(node)
faceting: facet_null()
> g # No plot here yet
Error: No layers in plot
> g + geom_line() + geom_point() # Add geometric objects
```



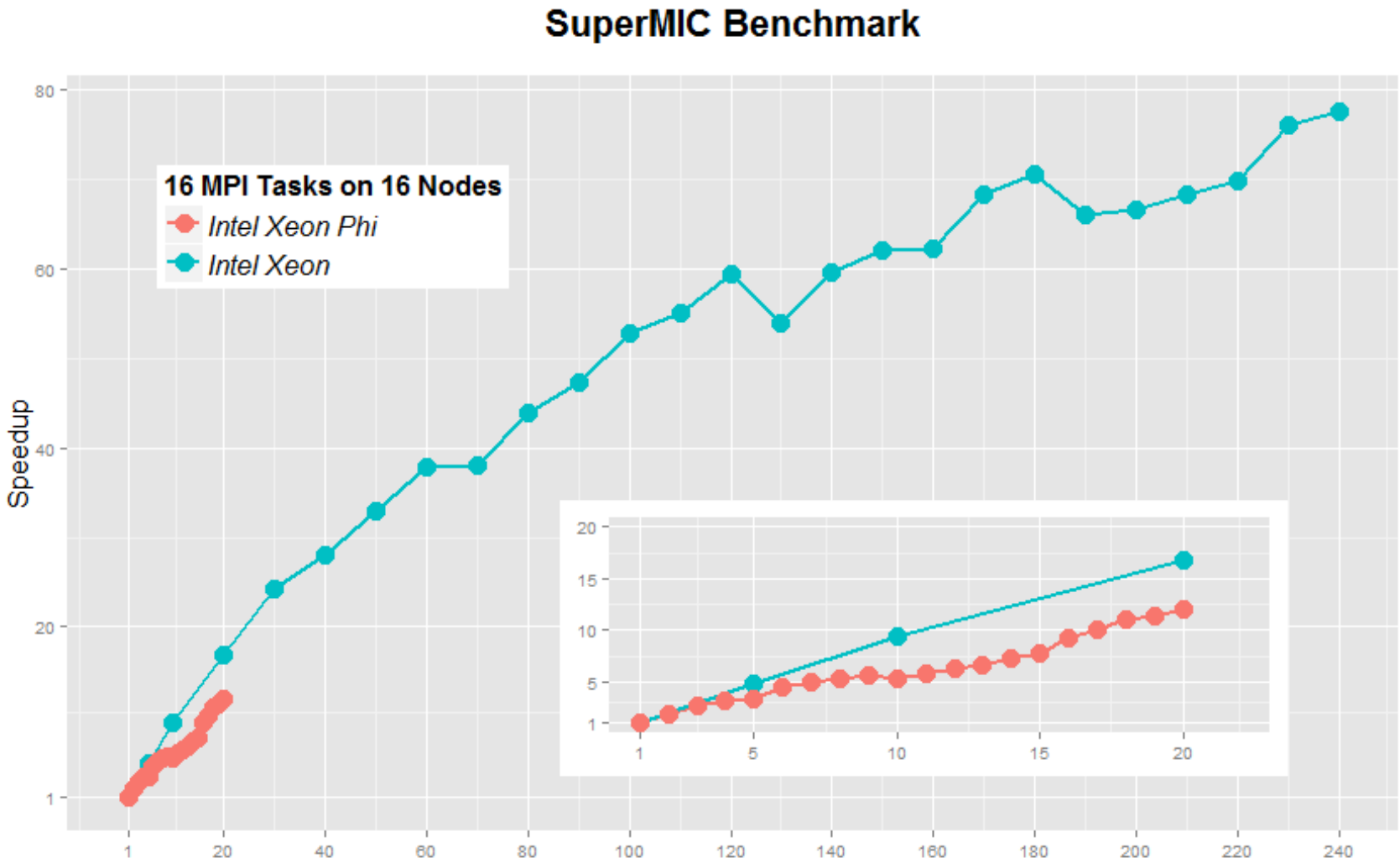
The same plot with decorations



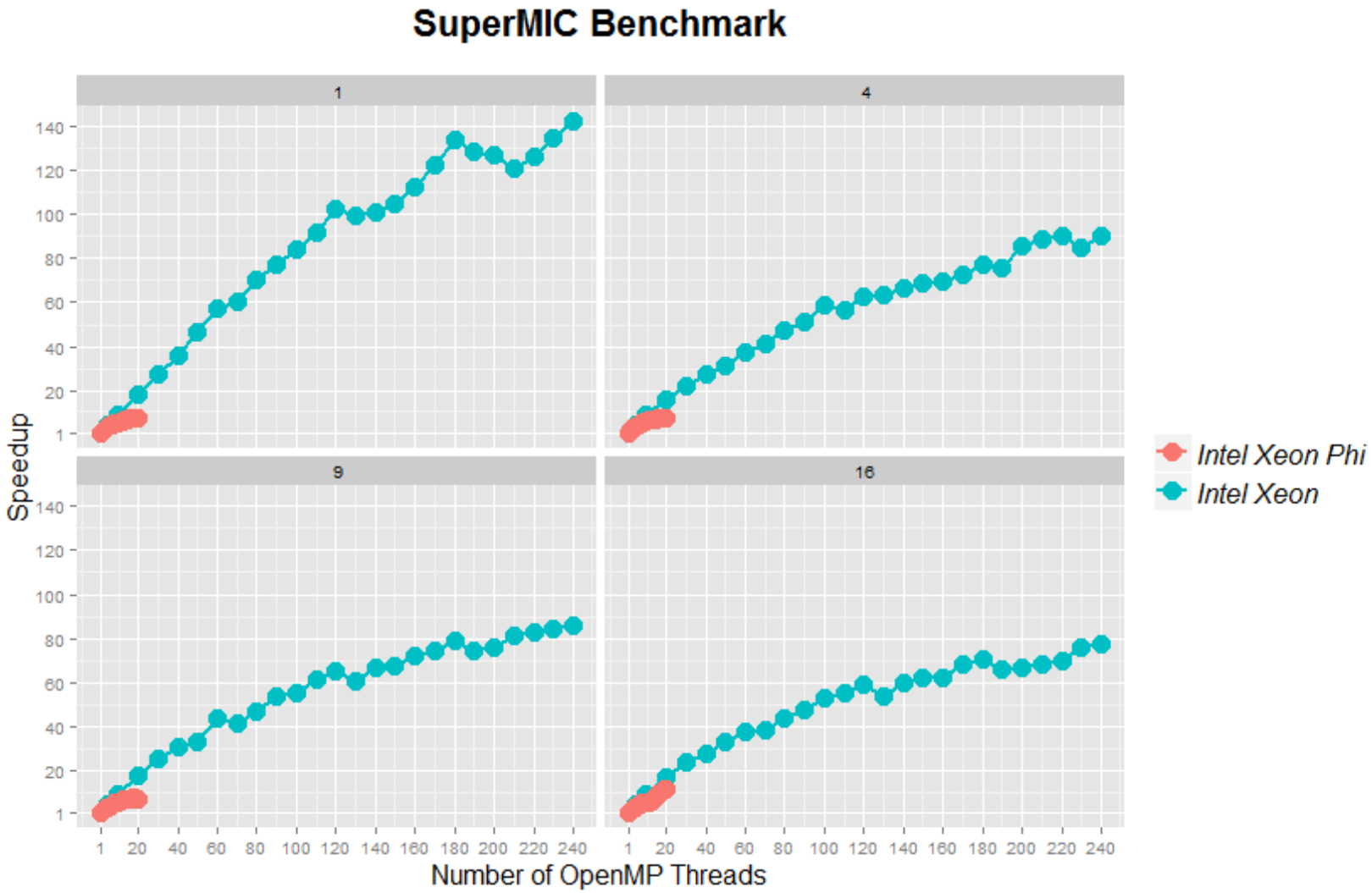

```
g <- ggplot(subset(bmData,device=="mic"), aes(nt,speedup,color=factor(node))) # ggplot call

g + geom_line(size=1) + # plot lines
  geom_point(size=5) + # plot points
  # Title and axis labels
  labs(x="Number of OpenMP Threads", y="Speedup", title="SuperMIC Benchmark") +
  # Format the title, legends and axes
  theme(plot.title = element_text(colour="black", size=22, face="bold", vjust=2.5),
        legend.title = element_text(colour="black", size=16, face="bold"),
        legend.position=c(0.2,0.8),
        legend.text=element_text(face="italic", size=16),
        axis.title=element_text(size=16)) +
  scale_color_discrete(name="Offload to Intel Xeon Phi", # Legend title and keys
        labels=c("1 node", "4 nodes", "9 nodes", "16 nodes")) +
  scale_x_continuous(limits=c(1, max(bmData$nt)), # Limits and ticks of x axis
        breaks=c(1,seq(20,max(bmData$nt),20))) +
  scale_y_continuous(limits=c(1, max(bmData$speedup)), # Limits and ticks of y axis
        breaks=c(1,seq(20,max(bmData$speedup),20))) +
  annotate("text", x=200, y=30, # Annotation text
        label="MIC KMP AFFINITY=\n\"granularity=fine,balanced\"", size=8)
```

Here's the code



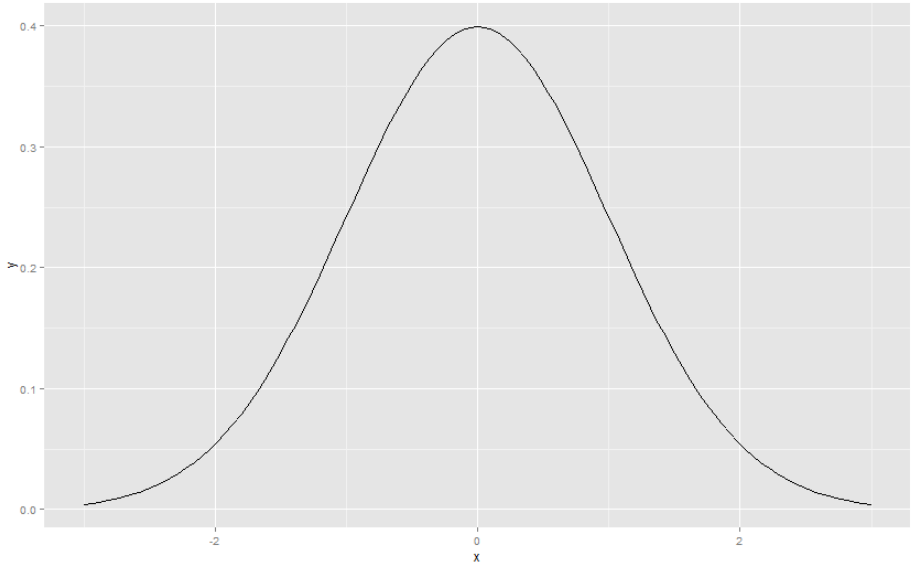
Plot with inset
(The code can be found in the script)



Plot with a 2x2 facet
(The code can be found in the script)

Plot Functions

- Plot a math function by using the `stat_function()` function




```
p <- ggplot(data.frame(x=c(-3,3)), aes(x=x))
p + stat_function(fun = dnorm)
```

Control The Output Format

- Graphic Devices
  - File devices
    - Bitmap: png, jpeg, etc.
    - Vector: pdf, svg, etc.
  - Screen devices (quartz, X11, etc.)
- Steps
  - Launch a graphic device
    - Functions: `png()`, `pdf()`, `svg()`, etc.
  - Make the plot
  - Close the device with `dev.off()` function

R Scripts

- We can put all the commands in a script
- Run R scripts with the `Rscript` command

```
[lyan1@qb4 Gnuplot-training]$ ls
benchmark.R  map3d.eps  map3d.pdf      multiplot.gpl  time-01-node-smic.dat  time-09-node-smic.dat
input3d.dat  map3d.gpl  multiplot.eps  multiplot.pdf  time-04-node-smic.dat  time-16-node-smic.dat
[lyan1@qb4 Gnuplot-training]$ Rscript benchmark.R
Warning messages:
1: Removed 22 rows containing missing values (geom_path).
2: Removed 22 rows containing missing values (geom_point).
[lyan1@qb4 Gnuplot-training]$ ls -tr
input3d.dat  map3d.pdf             time-04-node-smic.dat  multiplot.eps  smic_benchmark_1.pdf
map3d.gpl    multiplot.gpl         time-09-node-smic.dat  multiplot.pdf  smic_benchmark_2.svg
map3d.eps    time-01-node-smic.dat time-16-node-smic.dat  benchmark.R    smic_benchmark_3.png
```