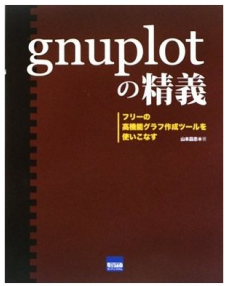
Brainwashing session : How to make TEX docs with tons of graphs (part II)?



TEX

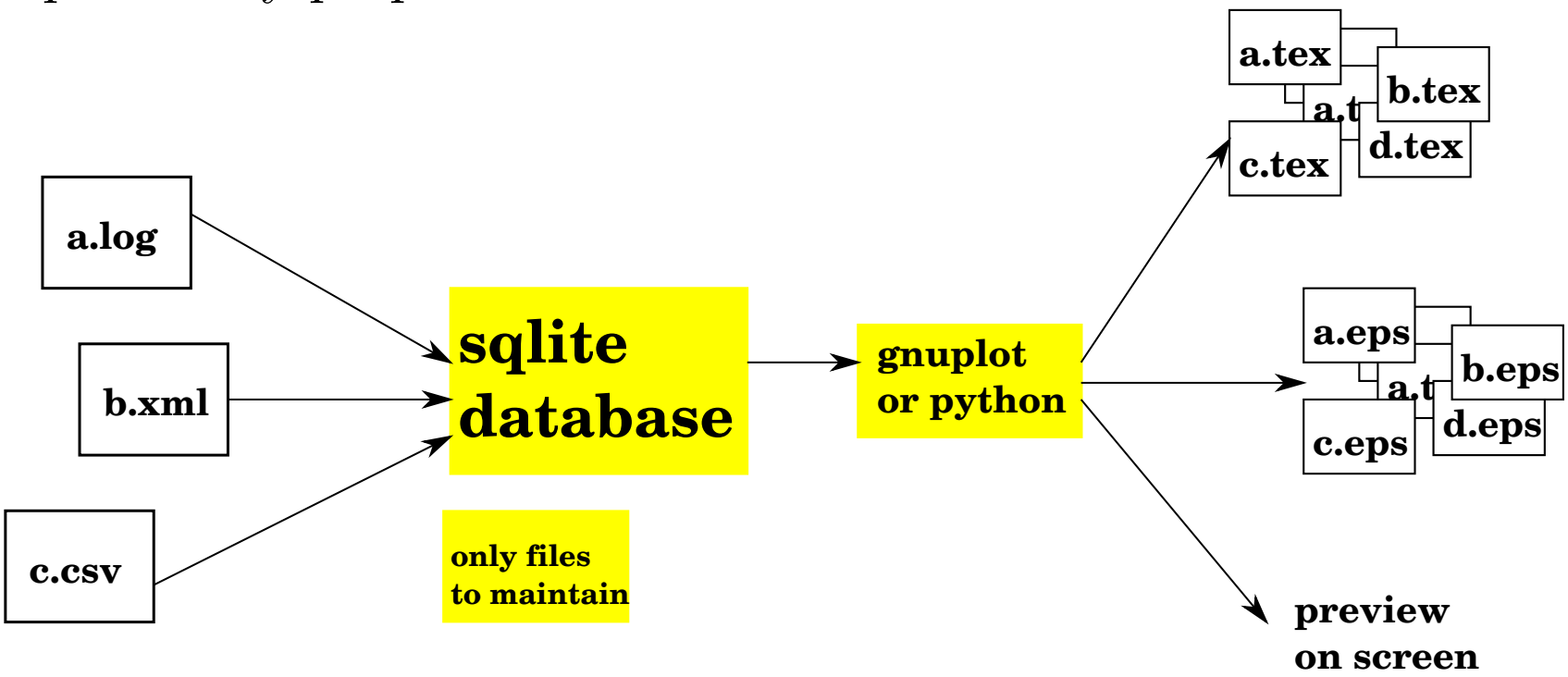
Introduction

Context

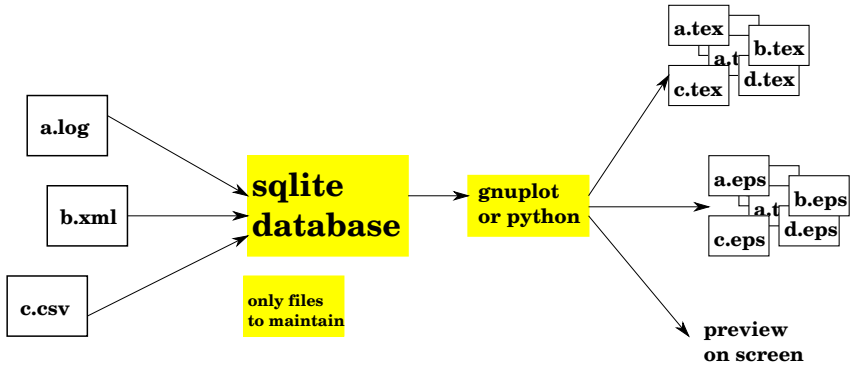
- You want to create a TEX document with lots of graphs easily
- You want to *automate the entire process*, from running experiments to producing TEX document with graphs, so you can *painlessly repeat the experiment* and update data in the document
- With ad-hoc solutions, your directory easily screws up with too many data files and scripts you never understand a week later

The practice

I previously proposed:



The practice



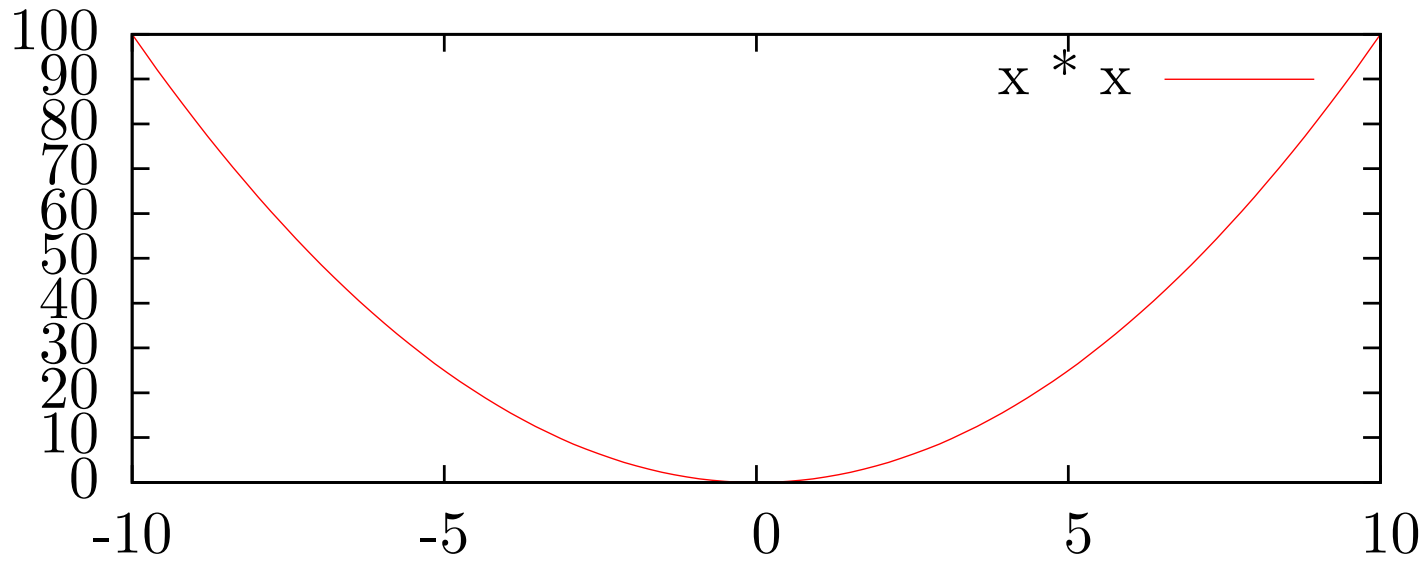
- You maintain only 2 files for all graphs
  - an sqlite3 database that has all raw data
  - a gnuplot or a script file that generates data to plot (by gnuplot)
- I talked about a command, txt2sql, which makes it straightforward to convert text files (log) to sqlite3 database
- *Generating lots of graphs from a database was still painful*, and I now address it

What I made this time?

- A small python library to interface with gnuplot command
- There is python-gnuplot package, but I do not rely on it
- A tentative name: `smart_gnuplotter.py`
- Code is available at:
  `http://code.google.com/p/smart-gnuplotter/`

A simplest example

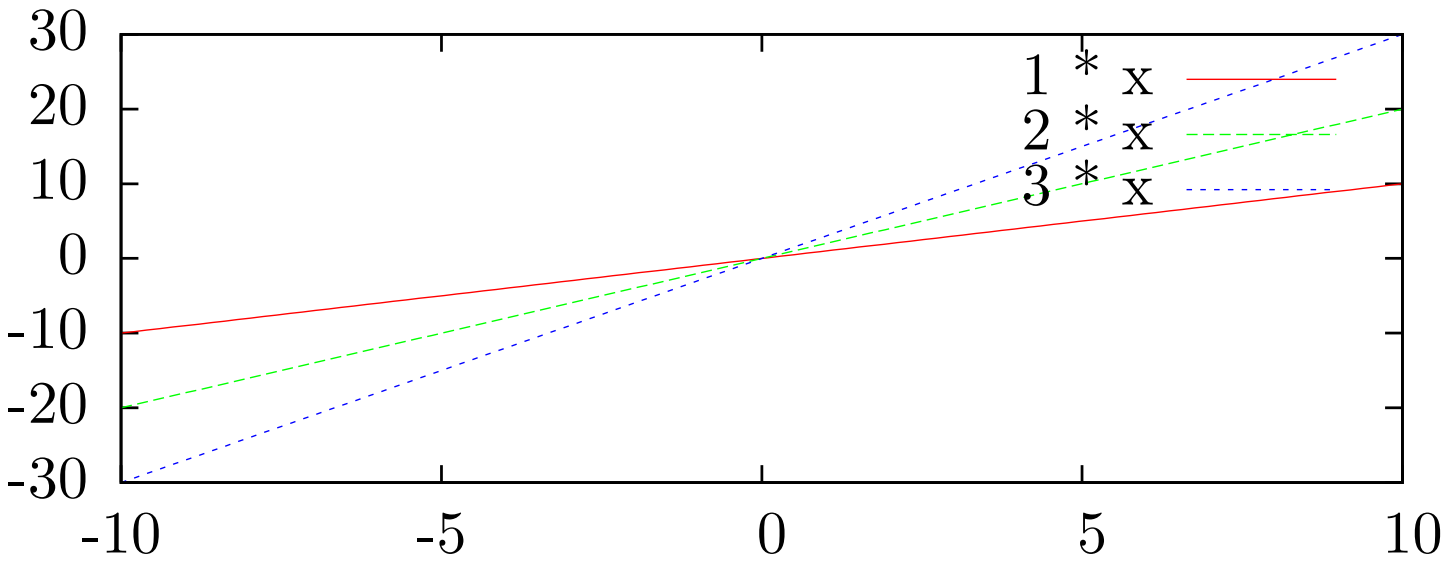
```
1 import smart_gnuplotter
2 g = smart_gnuplotter.smart_gnuplotter()
3 g.graphs("x*x")
```



Writing multiple plots in a single graph

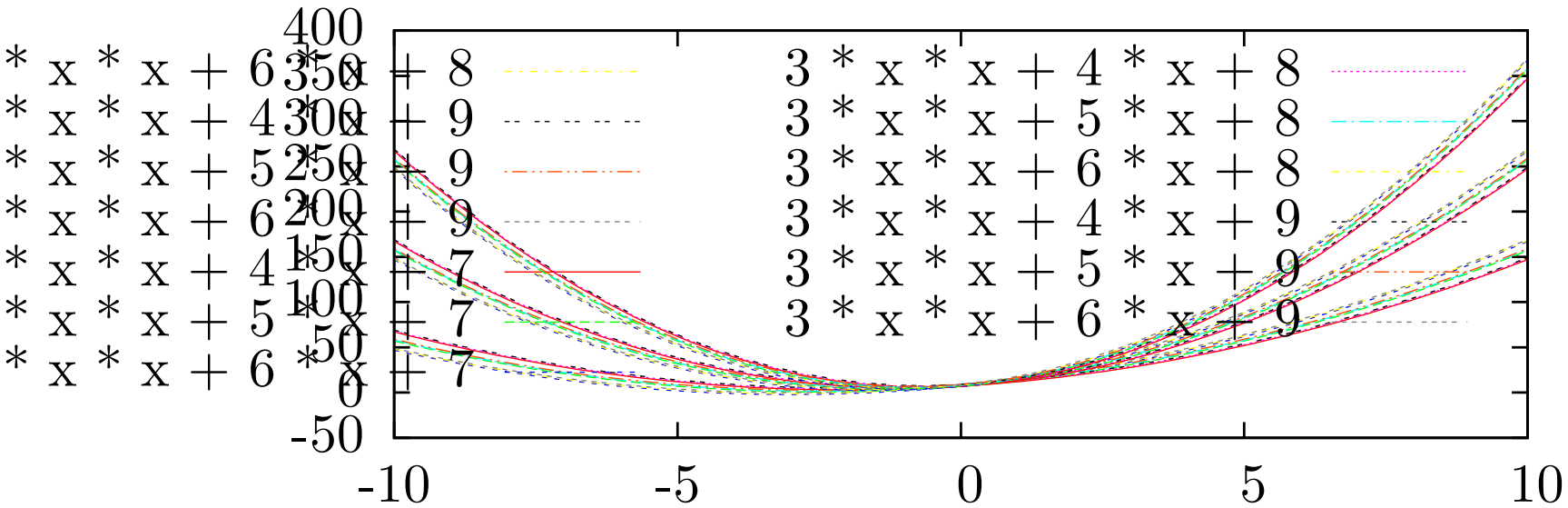
```
1 g.graphs("%(a)s * x", a=[1,2,3])
```

- The basic form is to parameterize the expression with *%(var)s* and supply its values with *var=list*



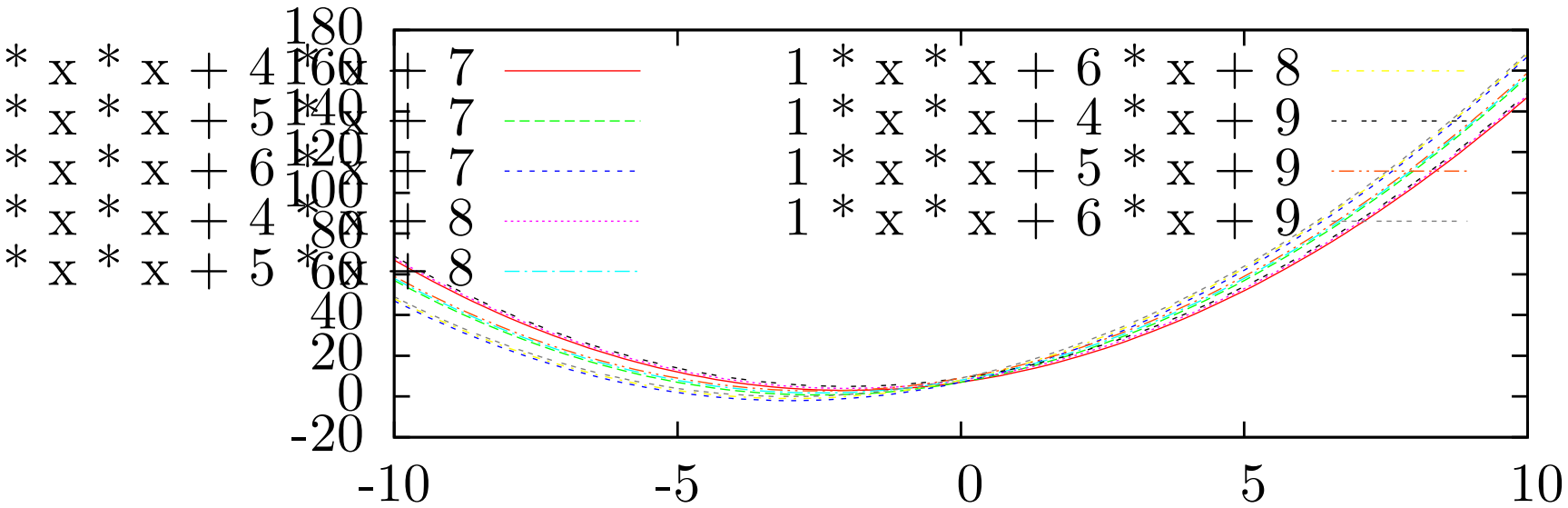
More plots in a single graph ...

```
1 g.graphs("%(a)s*x*x+%(b)s*x+%(c)", a=[1,2,3], b=[4,5,6], c=[7,8,9])
```

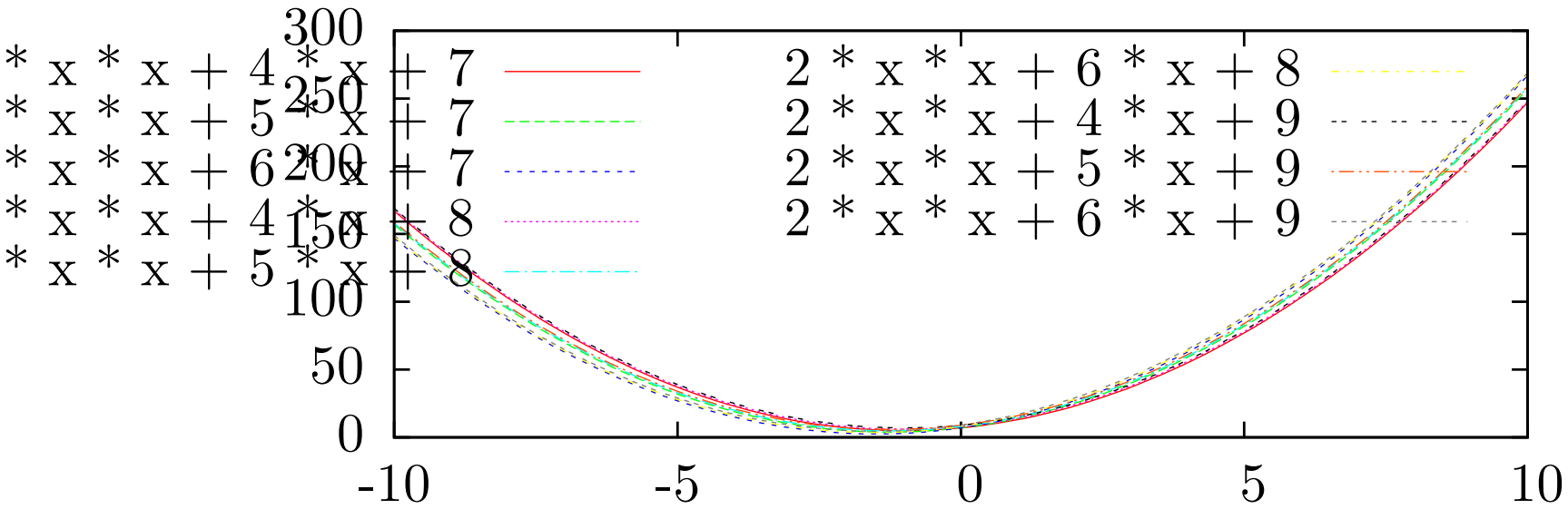


Writing multiple graphs in a single shot (1)

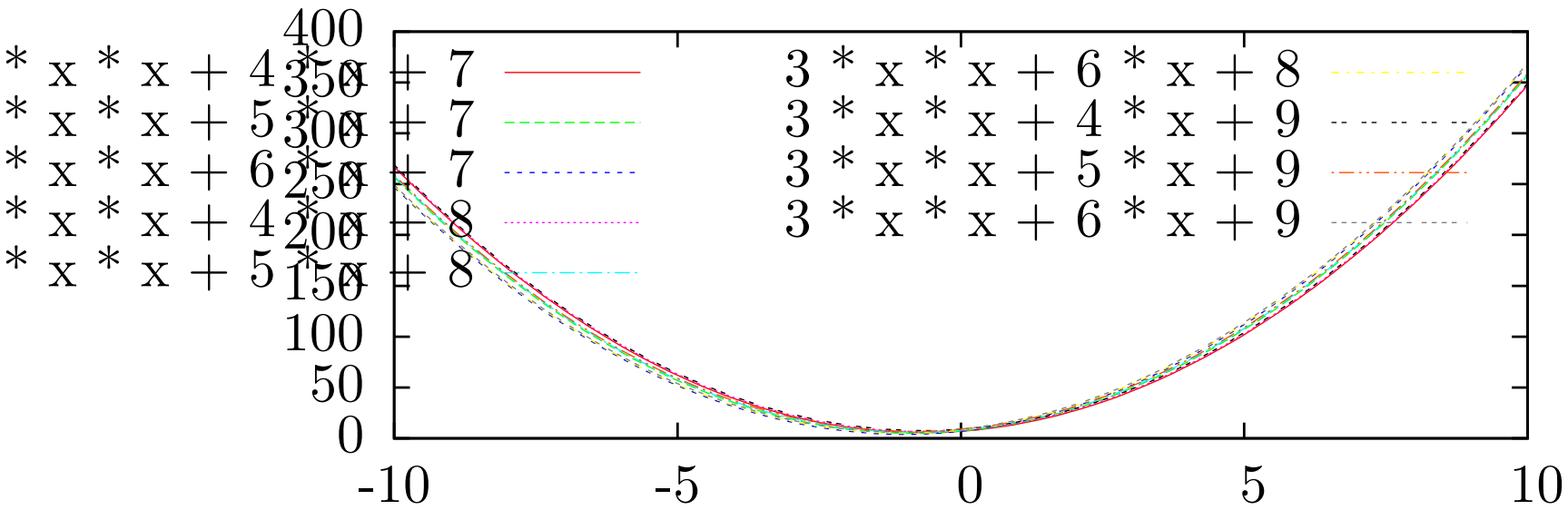
```
1 g.graphs("%(a)s*x*x+%(b)s*x+%(c)", a=[1,2,3], b=[4,5,6], c=[7,8,9],
2           graph_vars=["a"])
```



Writing multiple graphs in a single shot (2)



Writing multiple graphs in a single shot (3)



Changing terminals and writing to files

```
1 g.graphs("x*x",
2          terminal="epslatex size 10cm,5cm",
3          output="xx")
```

- will use epslatex terminal and generate output to `xx.tex`
- for some commonly used terminal types, extensions are automatically attached, so you don't have to change output names every time you change terminal types

Changing default terminals

Alternatively, you may set the default terminal.

```
1 g.default_terminal = "epslatex size 10cm,5cm"
2 g.graphs("x*x", output="xx")
```

Terminals and extensions currently automatically attached

terminal type	extension
epslatex	.tex
latex	.tex
pslatex	.tex
pstex	.tex
fig	.tex
texdraw	.tex
postscript	.eps
jpeg	.jpg
svg	.svg
gif	.gif
png	.png

Pausing behavior

- When terminal is a display (wxt, x11, or xterm), smart_gnuplotter generates "pause -1" so gnuplot waits until you enter a newline.
- If that happens, it then asks what to do for graphs that follow:

```
's' to suppress future prompts, 'q' to quit now, a <number>
to set pause to it, or else to continue [s/q/<number>/other]?
```

- Alternatively, you may change it directly by:

```
g.graphs(..., pause=x)
```

- or set the default by:

```
g.default_pause=x
```

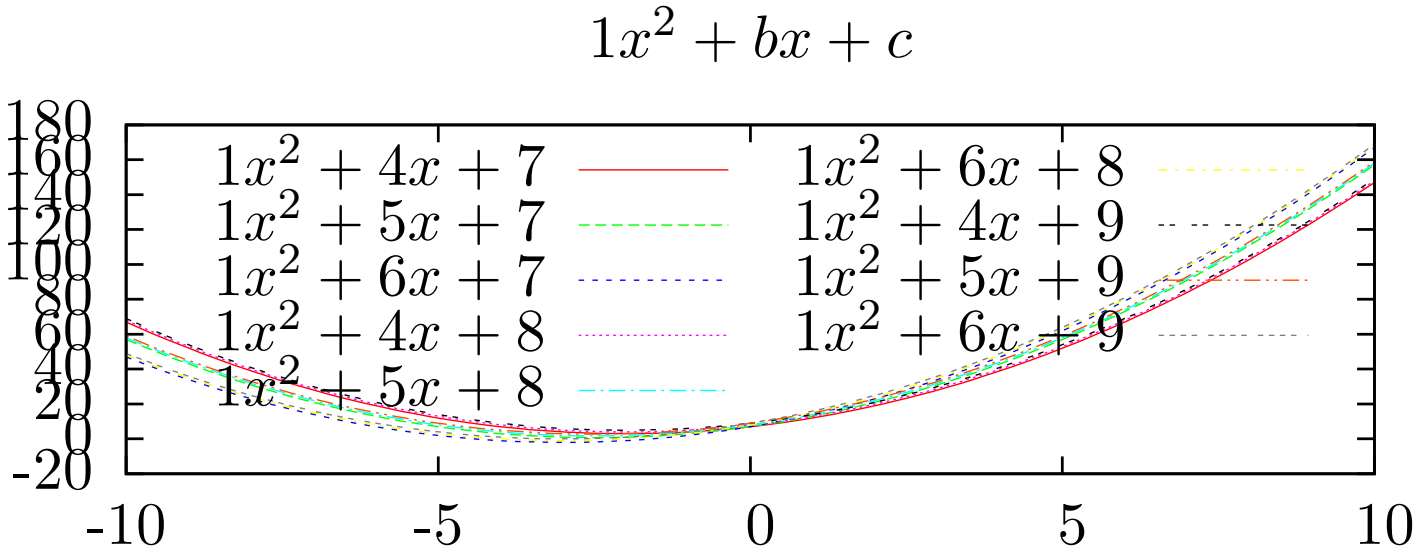
When gnuplot went wrong

- It leaves the file that fed gnuplot, so you may inspect it
- You may specify the filename by gpl_file

```
1 g.graphs(..., gpl_file="hoge.gpl")
```

Specifying various attributes

```
1 g.graphs("%(a)s*x*x+%(b)s*x+%(c)s", a=[1,2,3], b=[4,5,6], c=[7,8,9],
2           graph_vars=["a"], graph_attr=r'''
3 set title "$%(a)sx^2+bx+c$"
4 ''',
5           plot_attr=r'''title "$%(a)sx^2+%(b)sx+%(c)s$"''')
```



Specifying various attributes

- graph_attr modifies an entire graph; it is whatever comes before the 'plot' command
- plot_attr modifies each plot in a graph; it is whatever comes after each plot

```
g.graphs(E,
         graph_attr=graph_attr,
         plot_attr=plot_attr)
```

$\approx$

```
graph_attr
plot E plot_attr, E plot_attr, ...
```

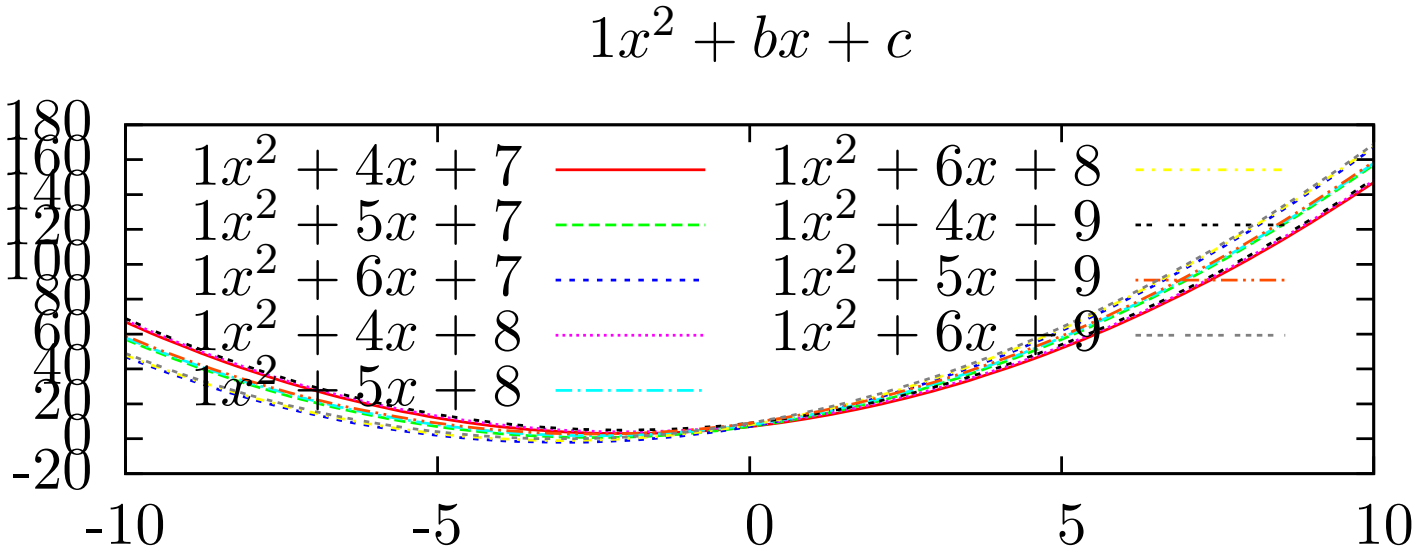
Shortcuts for frequently used attributes

Frequently used attributes can be directly set by keyword arguments. e.g.,

```
1  g.graphs("%(a)s * x * x + %(b)s * x + %(c)s",
2           a=[1,2,3], b=[4,5,6], c=[7,8,9],
3           output="graphs/mod_%(a)s_xx_bx_c",
4           graph_title="$%(a)sx^2+bx+c$",
5           graph_attr=r'''set key left
6  ''',
7           plot_title="$%(a)sx^2+%(b)sx+%(c)s$",
8           plot_with="lines linewidth 2",
9           graph_vars=["a"])
```

Shortcuts for frequently used attributes

Then you get:



Supported graph attributes

name	description	example
terminal	terminal used	"epslatex size 10cm,5cm"
output	file(∗) to output	"my_graph_%(a)s"
graph_title	graph title	"speedup_%(a)s"
xrange	xrange	"[0:10]"
yrange	yrange	"[0:10]"
boxwidth	set boxwidth(∗∗)	"0.9 relative"

- ∗ : it may be extended based on terminal type.
- ∗∗ : effective only when you plot with boxes

The effect ≈

```
1 set attr value
```

Supported plot attributes

name	description	example
plot_title	plot title	"%(a)s"
plot_with	style to plot plots	"boxes"
using	columns to plot plots	"1:2"

The effect ≈

```
1 plot ... using using with plot_with title "plot_title"
```

Other things to plot

- As it is normal in gnuplot, you may plot
  - datafile

```
1 g.graphs('"filename"', ...)
```

  - output of a command

```
1 g.graphs('">cmd"', ...)
```

- Besides, you may plot
  - data in python list

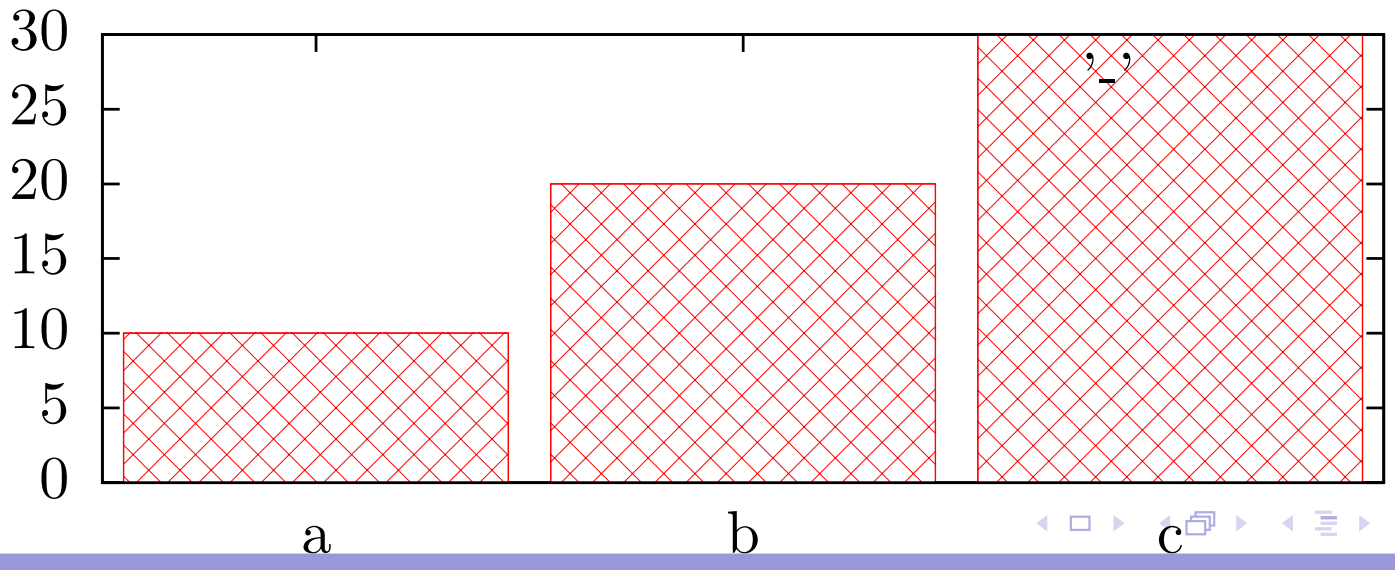
```
1 g.graphs([(1,2),(3,4),...], ...)
```

  - output of an SQLite query (later)

Bonus : writing graphs with symbolic x-axis

- With only gnuplot, it's tedious to show a graph with symbolic x-axis
- smart_gnuplotter does all the work you need to do for it

```
1 g.graphs([("a",10), ("b",20), ("c", 30)],
2          yrange="[0:]", boxwidth="0.9 relative",
3          plot_with="boxes fs pattern 1")
```



sqlite3: where the real power comes from

```
1 g.graphs((db, query, init_string, init_file, udfs, udas, udcs), ...)
```

- When you give a tuple as the first argument, a database query is executed and the result treated as data
  - init_string, init_file, udfs, udas, udcs are optional
- The above ≈

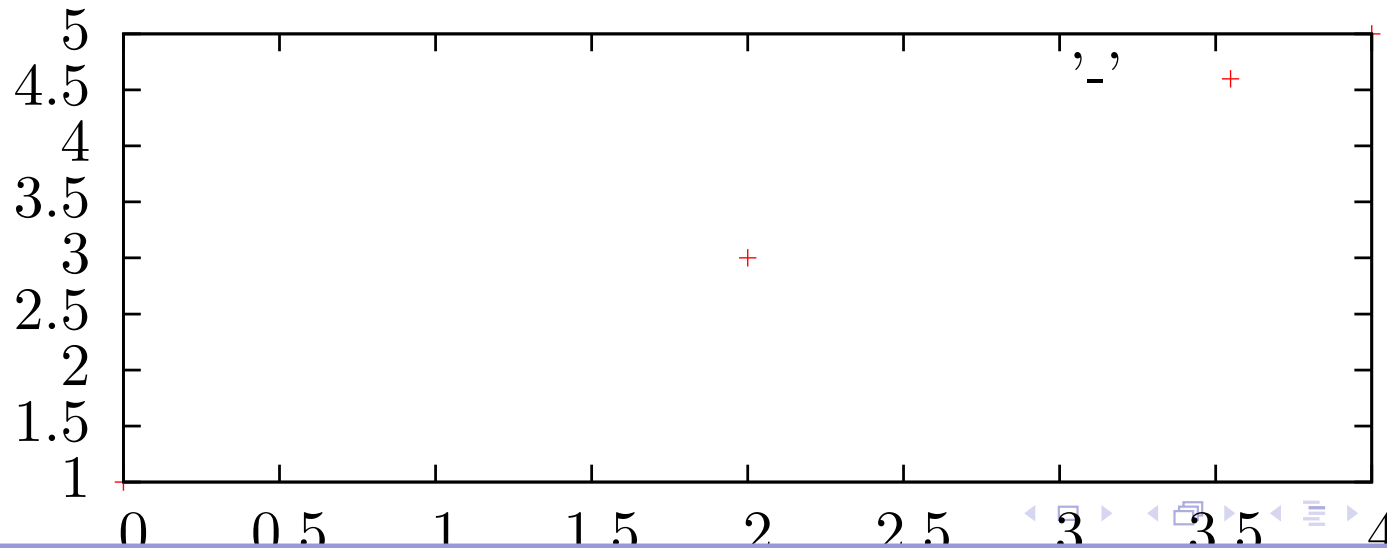
```
1 co = sqlite3.connect(db)
2 for name,arity,f in udfs: co.create_functions(name,arity,f)
3 for name,arity,f in udas: co.create_aggregates(name,arity,f)
4 for name,f in udcs:       co.create_collations(name,f)
5 co.execscript(init_string)
6 co.execscript(content of init_file)
7 co.execute(query)
8 g.graphs(co.fetchall(), ...)
```

A basic example with sqlite3

Say a database example.sqlite contains the following table

```
1 select * from t
2 0|1
3 2|3
4 4|5
```

```
1 g.graphs(("example.sqlite", "select * from t"))
```



Document

```
1 cd where you can load smart-gnuplotter
2 pydoc -p port smart-gnuplotter
```

and open localhost:*port* with your browser

Are You Brainwashed?

A real example : database schema

Database 'a' contains all results from matrix multiply

```
1 sqlite> .schema
2 CREATE TABLE a (arch, type, ppn, M, gflops_per_sec, ...);
```

- arch : architecture (barcelona, nehalem, sandybridge)
- type : program type (serial, MassiveThreads, Cilk)
- ppn : number of cores (1, 2, 4, 6, ...)
- M : matrix size
- gflops_per_sec : performance (GFLOPS)

Experiments are repeated, so there are many data of the same (arch, type, ppn, M)

A real example : graphs we want to show

Let's say we would like to show three graphs:

- serial : compares serial performance among program types, for each architecture
- gflops : shows GFLOPS with cores, for each architecture and program type
- speedup : shows speedup with cores, for each architecture and program type

Serial

- STEP 1: take some time interacting with sqlite3 to come up with a right query showing data for a single graph

```
1 select type,avg(gflops_per_sec) from a
2 where ppn=1 and arch="nehalem" and M=704
3 group by type
```

- STEP 2: parameterize it:

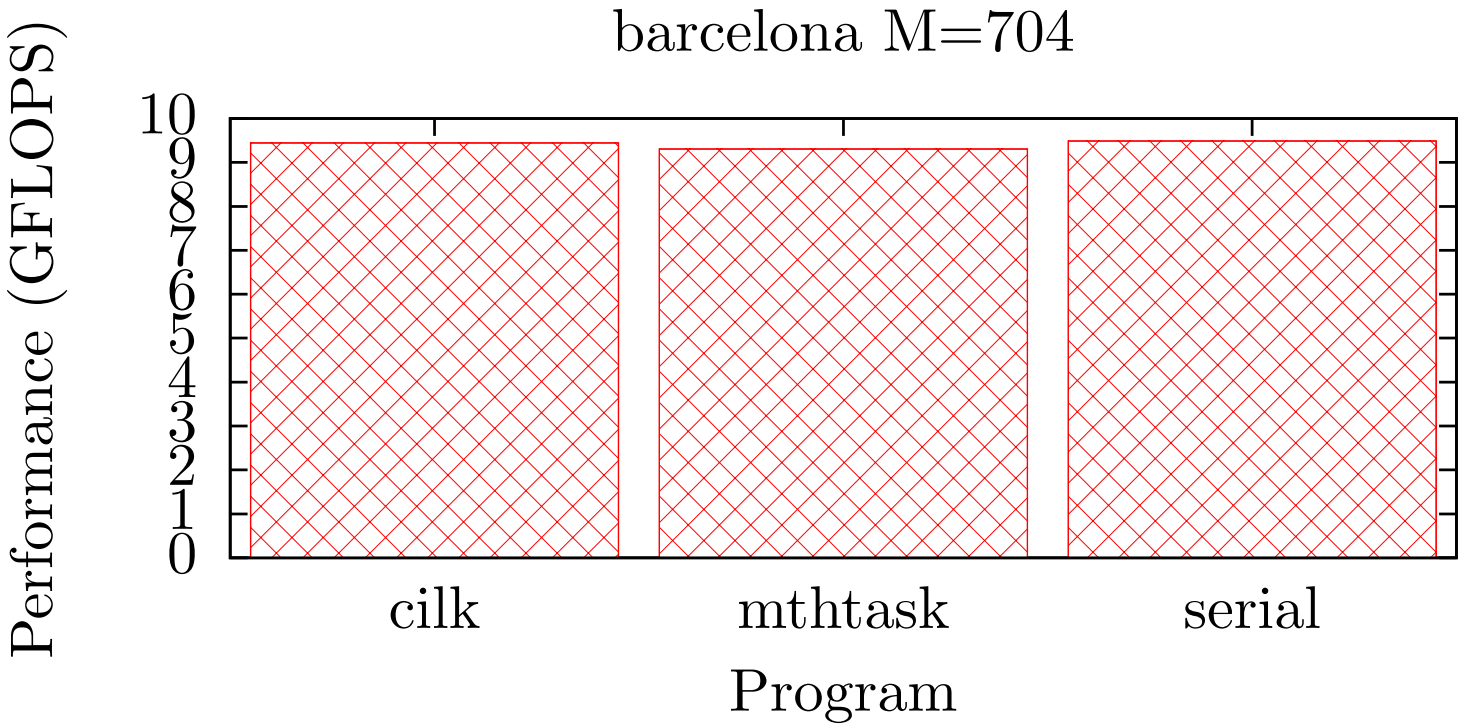
```
1 g.graphs(("matrix.sqlite",
2           r'''select type,avg(gflops_per_sec) from a
3 where ppn=1 and arch="%(arch)s" and M=%(M)s
4 group by type'''),
5          arch=["barcelona", "nehalem", "sandybridge"],
6          M=[704, 1088])
```

do_sql method for a smarter parameterization

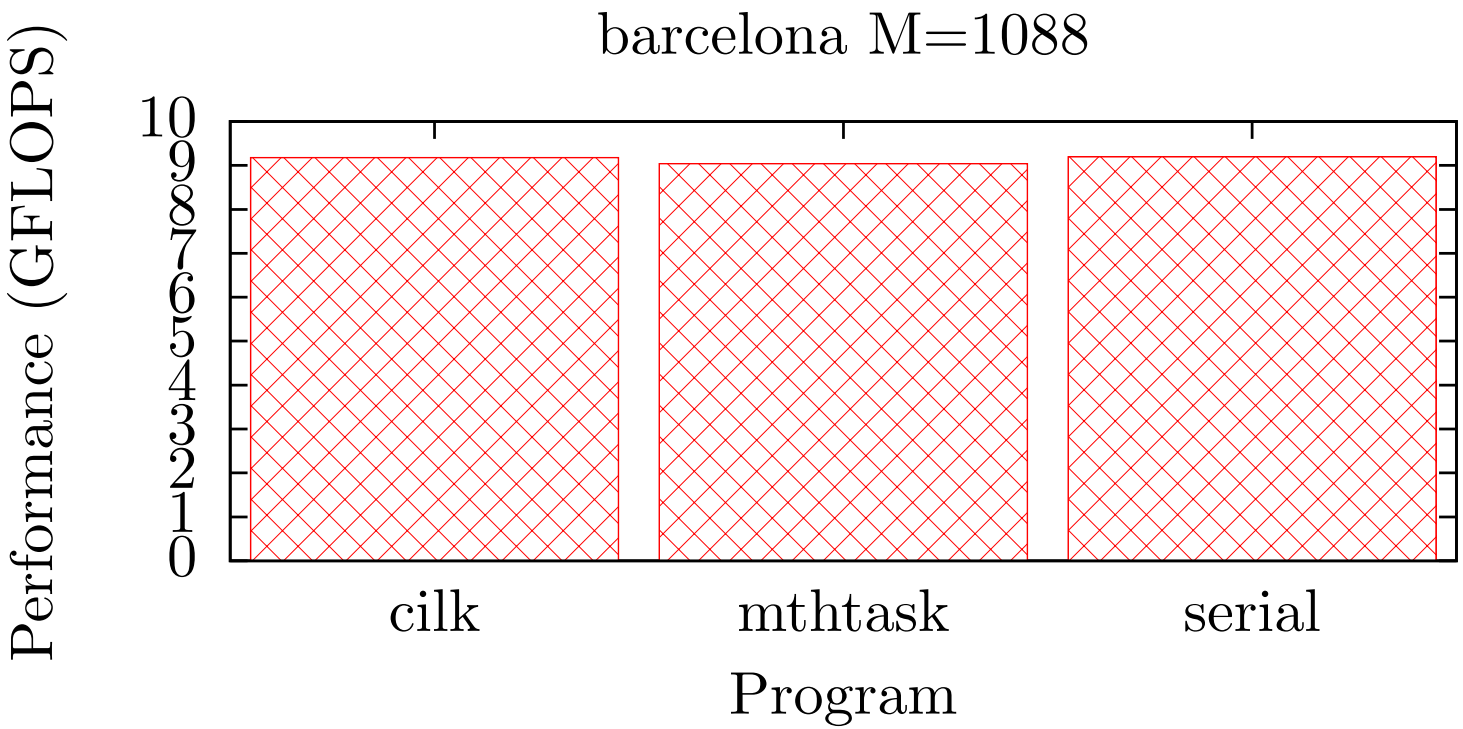
- You often want to ask database to determine parameters
- do_sql method just does that

```
1 Ms=g.do_sql(db,"select distinct M from a where M > 500",single_col=1)
2 archs=g.do_sql(db,"select distinct arch from a",single_col=1)
3 g.graphs(("matrix.sqlite",
4           r'''select type,avg(gflops_per_sec) from a
5 where ppn=1 and arch="%(arch)s" and M=%(M)s
6 group by type'''),
7          arch=archs, M=Ms, graph_vars=["arch", "M"])
```

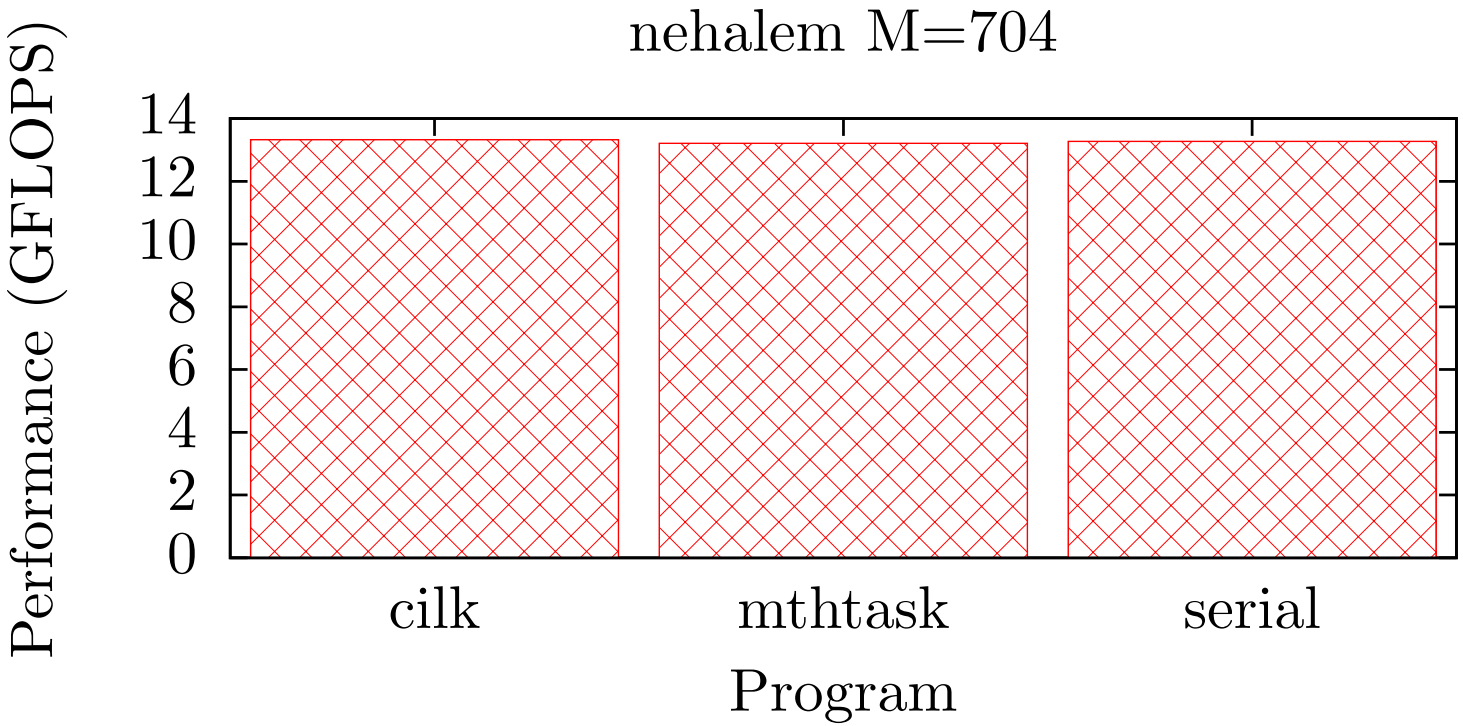
Barcelona, $M = 704$



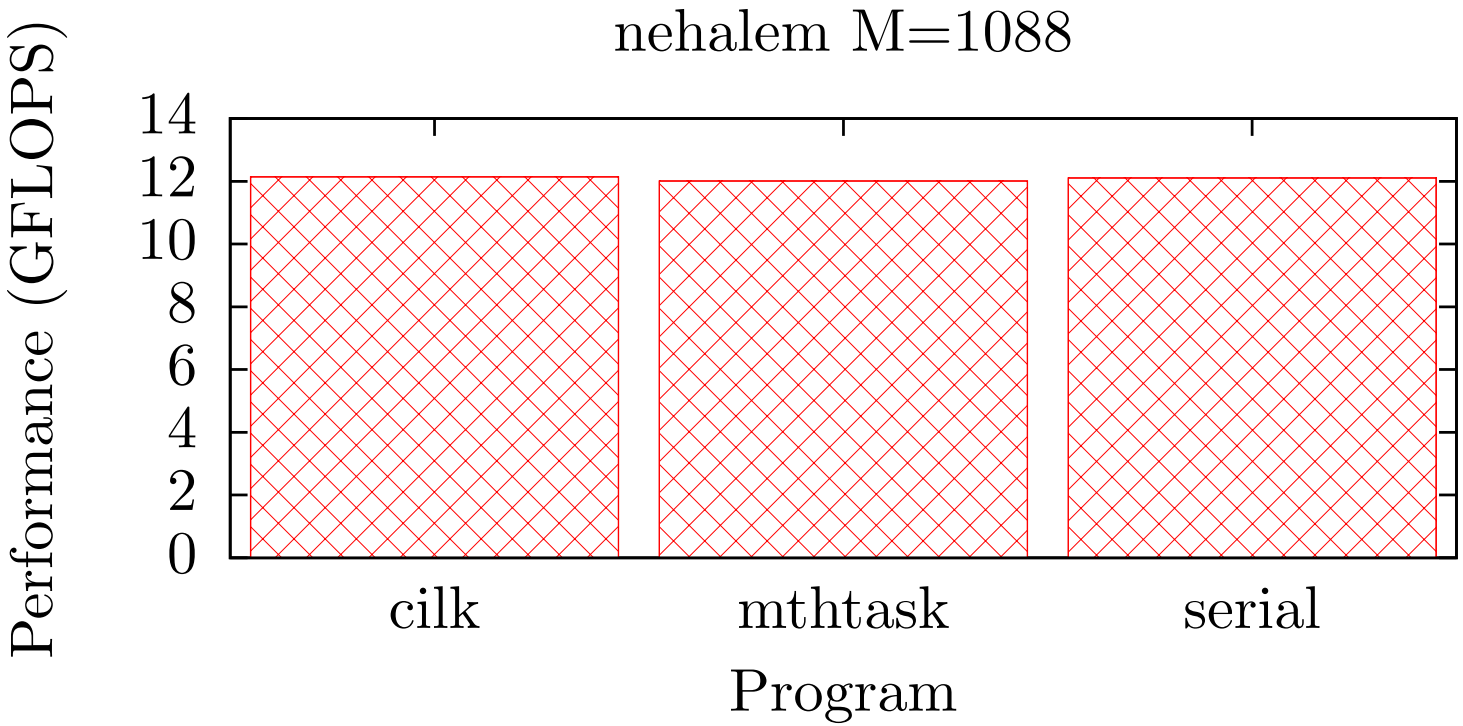
Barcelona, $M = 1088$



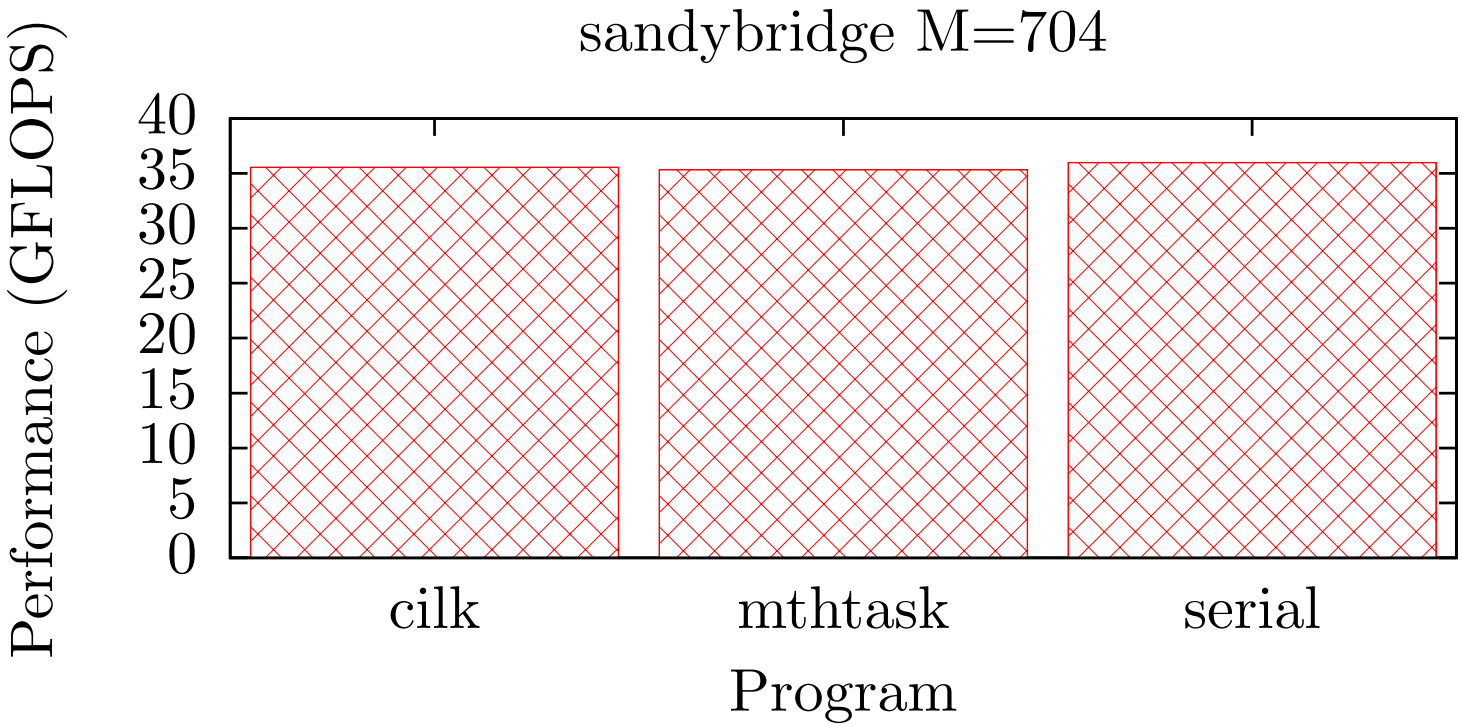
Nehalem, $M = 704$



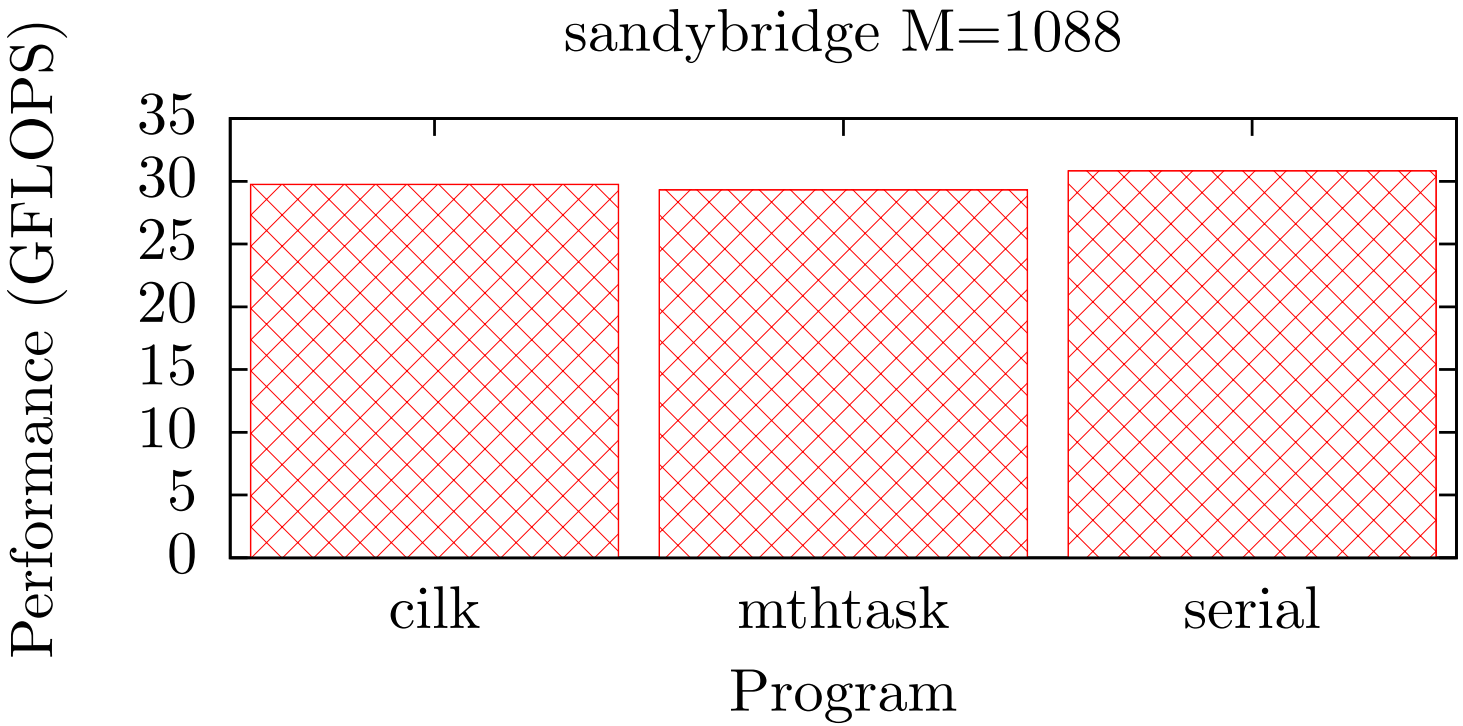
Barcelona, $M = 1088$



Sandy Bridge, $M = 704$



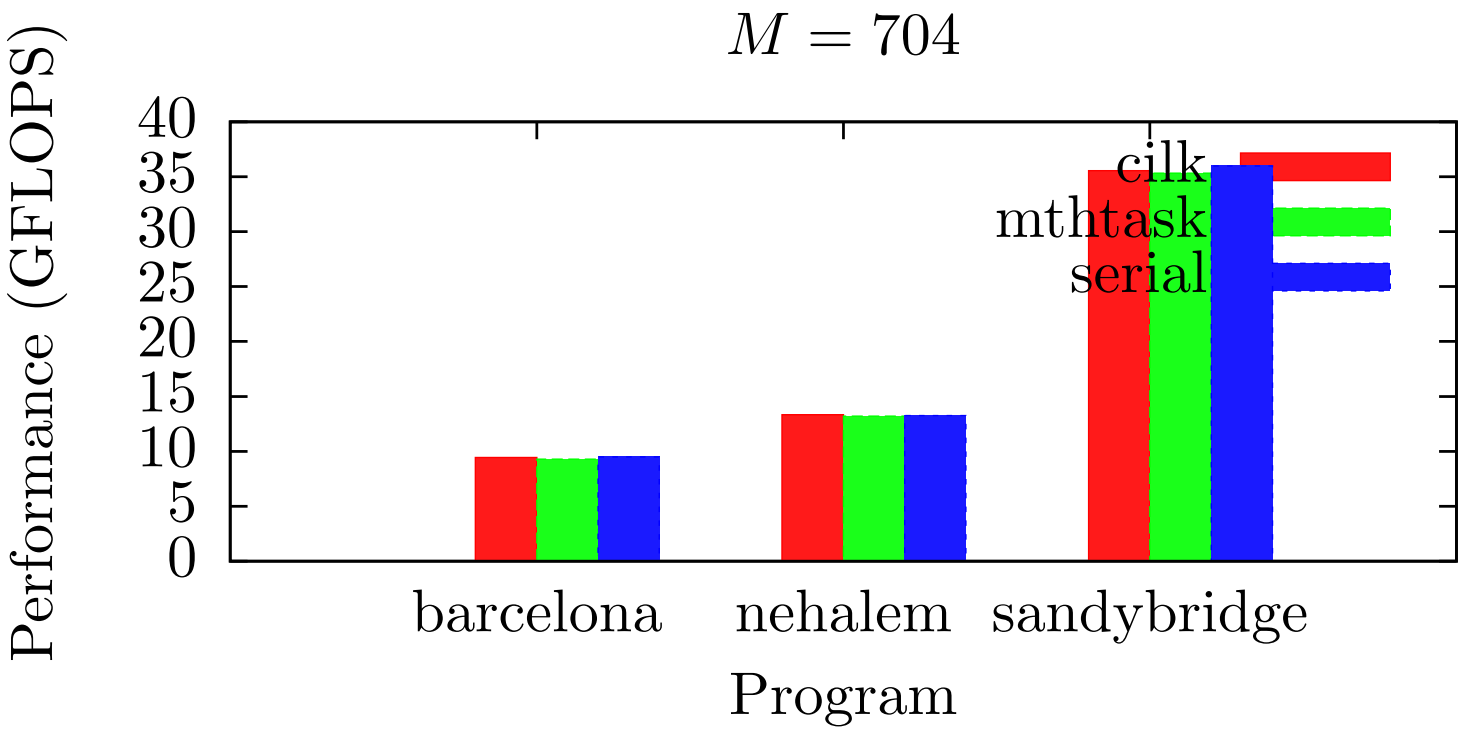
Sandy Bridge, $M = 1088$



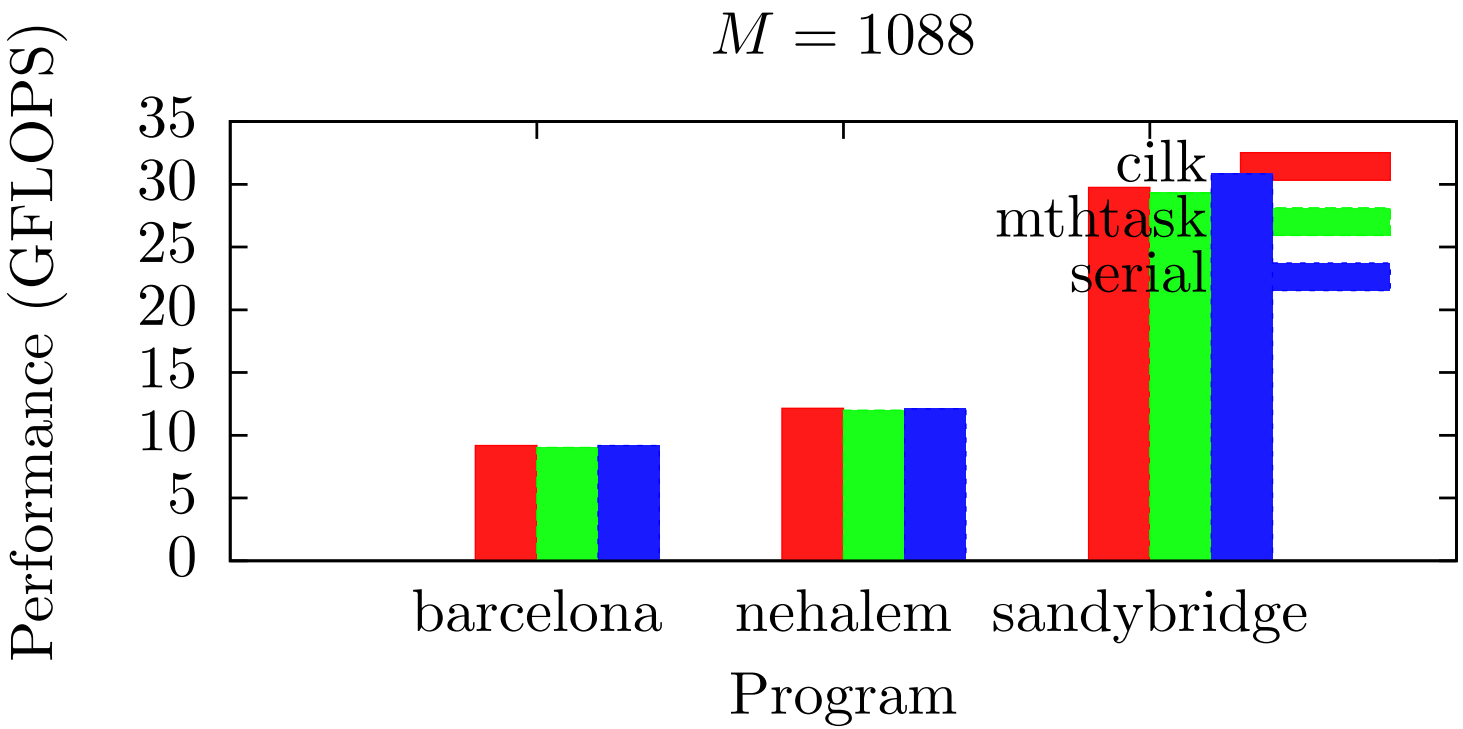
The real code that saves the 6 graphs

```
1  query = r'''select type,avg(gflops_per_sec) from a
2  where M = %(M)s and ppn = 1 and arch = "%(arch)s"
3  group by type'''
4
5  g.graphs(("matrix.sqlite", query),
6           output="graphs/serial_%(arch)s_%(M)s.tex",
7           plot_title="",
8           plot_with="boxes fs pattern 1",
9           boxwidth="0.9 relative",
10          graph_title="%(arch)s M=%(M)s",
11          yrange="[0:]",
12          xlabel="Program",
13          ylabel="Performance (GFLOPS)",
14          M=Ms, arch=archs, graph_vars=[ "arch", "M" ])
```

Putting three in a single graph with histogram : $M = 704$



$M = 1088$


$M = 1088$
Performance (GFLOPS)
35
30
25
20
15
10
5
0
cilk
mthtask
serial
barcelona
nehalem
sandybridge
Program

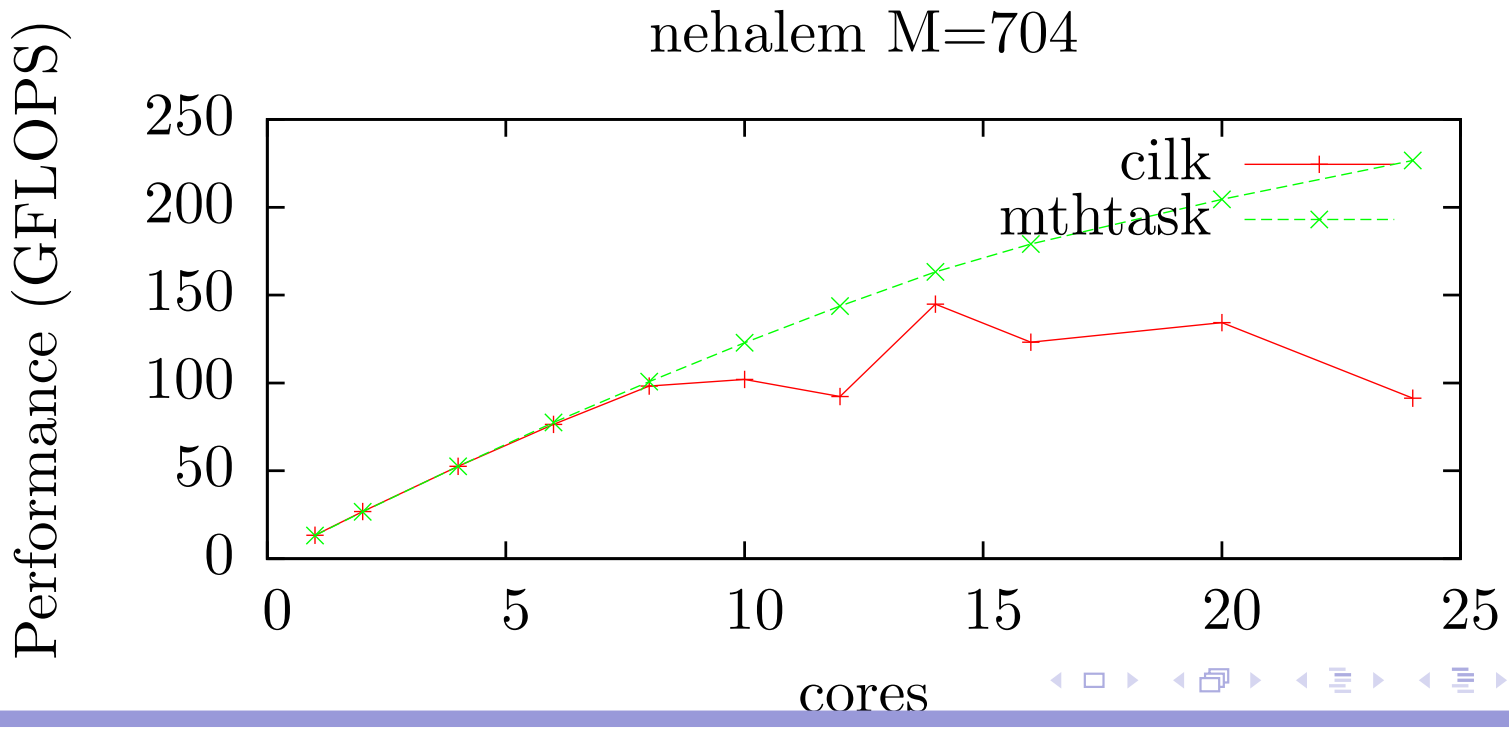
The real code that saves the 2 graphs

```
1  query = r'''select arch,avg(gflops_per_sec) from a
2  where M = %(M)s and ppn = 1 and type = "%(typ)s"
3  group by arch
4  order by arch'''
5  g.graphs(("matrix.sqlite", query),
6           output="graphs/serial_%(M)s",
7           graph_title="$M=%(M)s$",
8           plot_title="%(typ)s",
9           plot_with="histogram fs solid 0.9",
10          using="2",
11          yrange="[0:]",
12          xlabel="Program",
13          ylabel="Performance (GFLOPS)",
14          M=Ms, typ=all_types, graph_vars=[ "M" ])
```

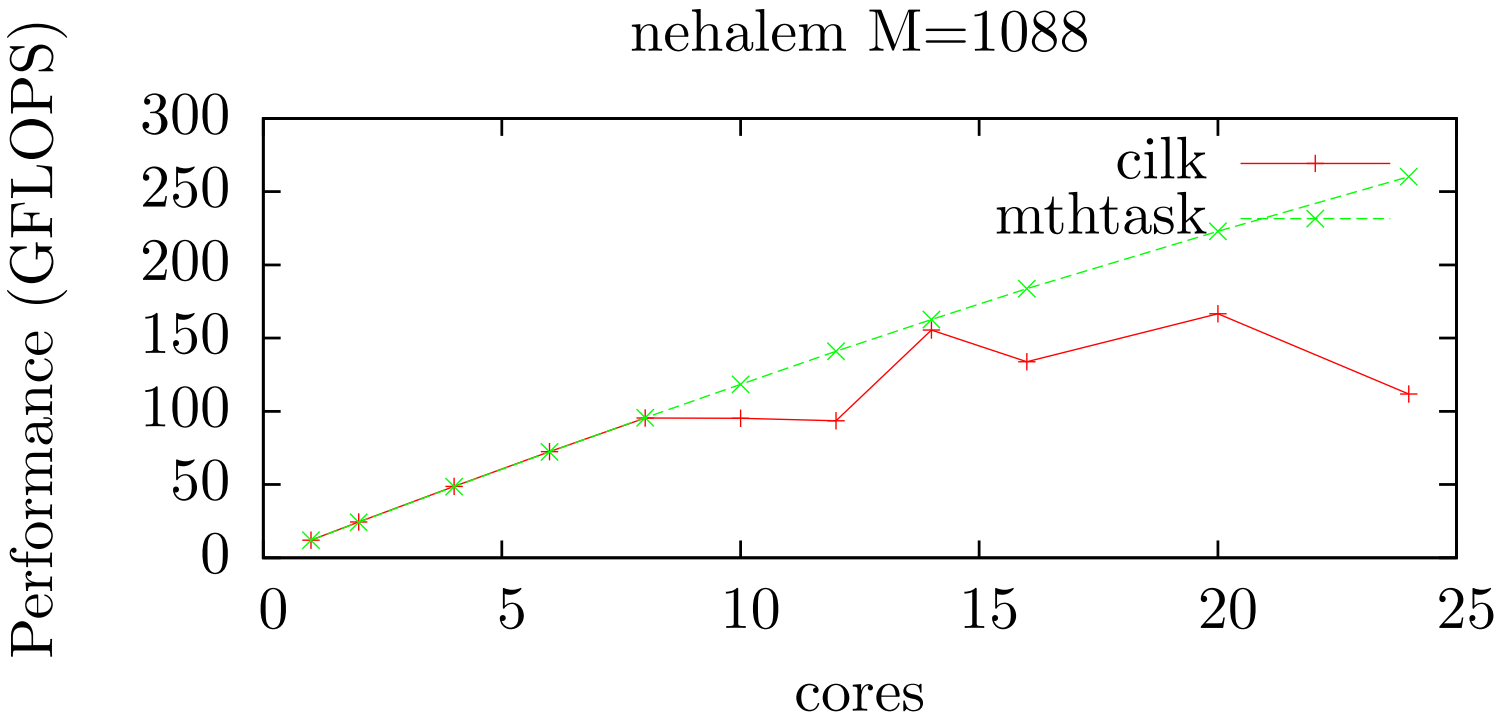
GFLOPS with cores

```
1 select ppn,avg(gflops_per_sec) from a
2 where type="%(typ)s" and M=%(M)s and arch="%(arch)s"
3 group by ppn
```

Note: I didn't bind workers to cores for Cilk

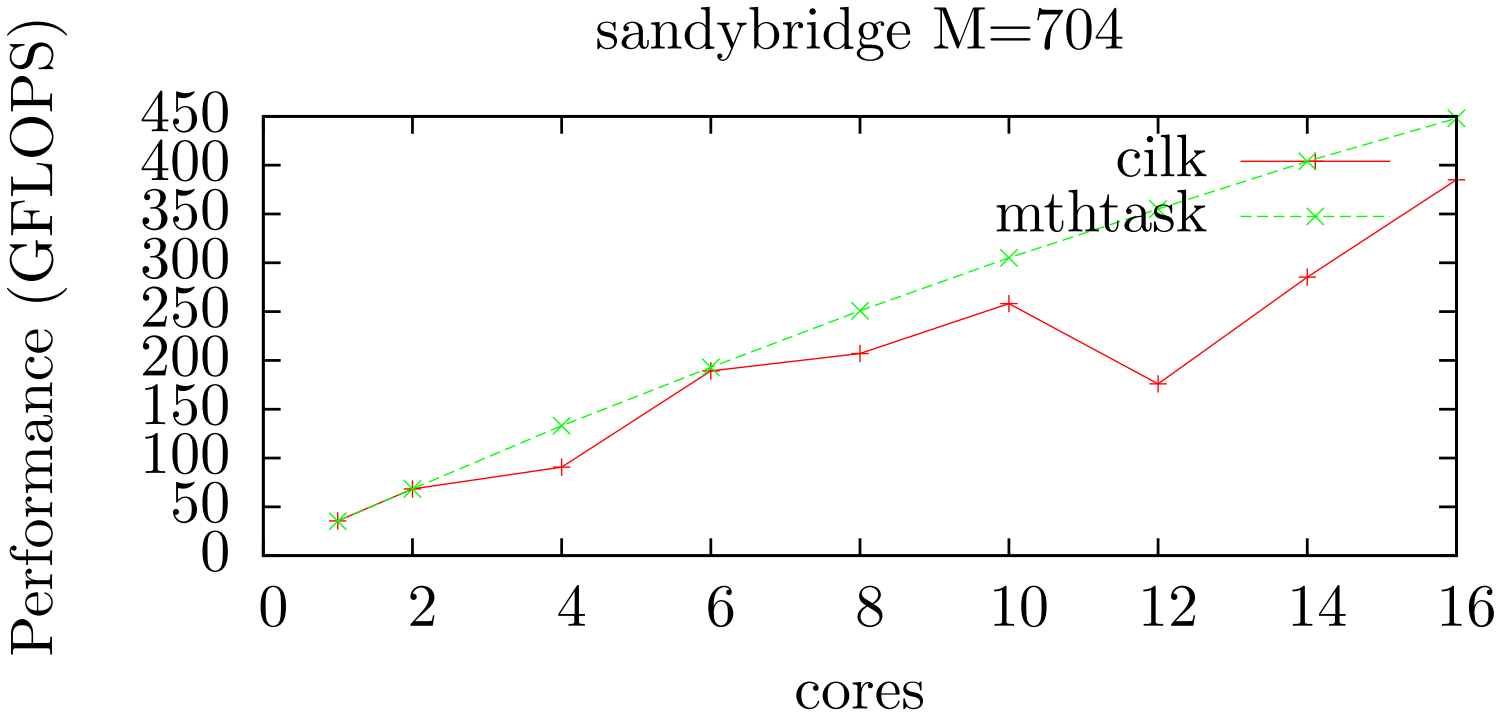


Nehalem, $M = 1088$



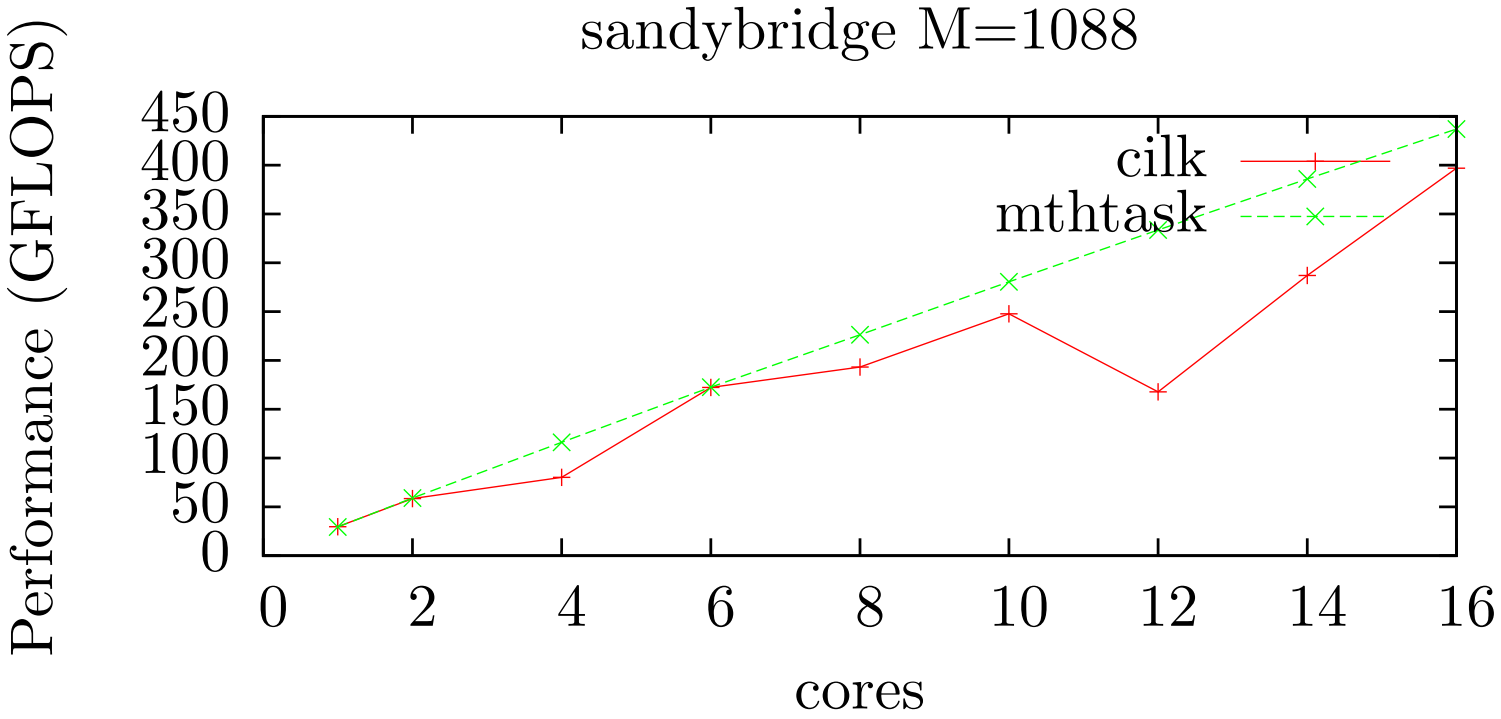
Sandy Bridge, $M = 704$

Note: TurboBoost was on

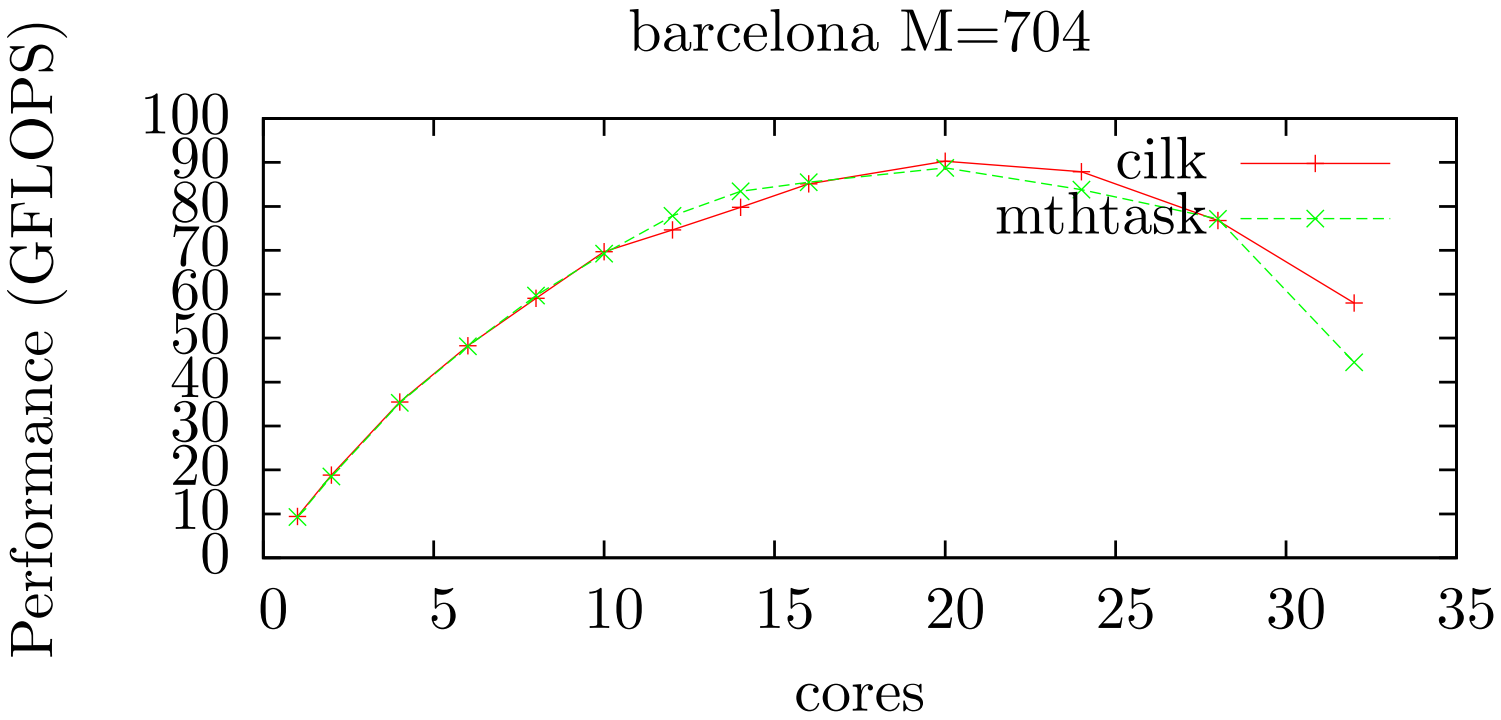


Sandy Bridge, $M = 1088$

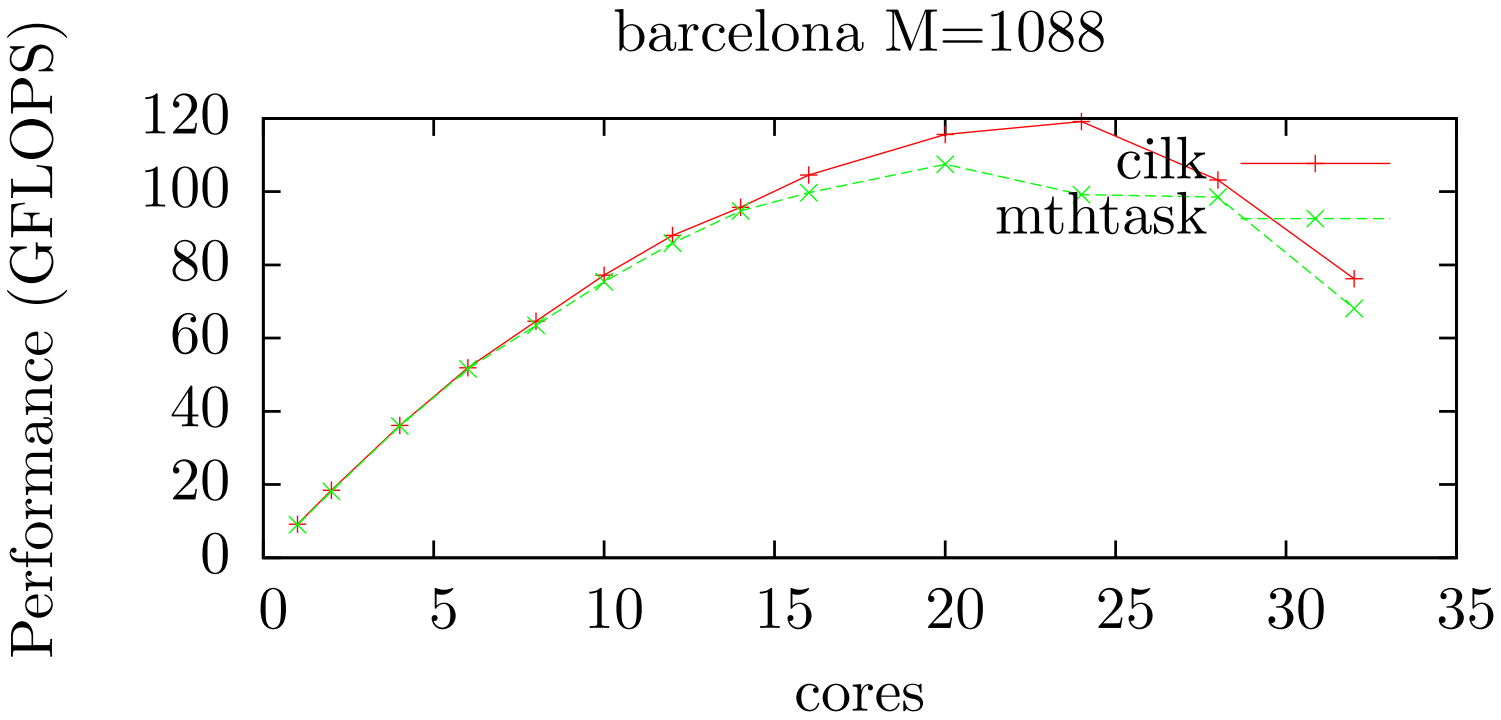
Note: TurboBoost was on



Barcelona, $M = 704$



Barcelona, $M = 1088$



The real code that shows the 6 graphs and save them

```
1  Ms    = g.do_sql(db, "select distinct M from a where M > 500",
           single_col=1)
2  archs = g.do_sql(db, "select distinct arch from a", single_col=1)
3  para_types = g.do_sql(db, "select distinct type from a where ppn > 1",
4                        single_col=1)
5
6  query = r'''select ppn,avg(gflops_per_sec) from a
7  where type="%(typ)s" and M=%(M)s and arch="%(arch)s"
8  group by ppn'''
9
10 g.graphs(("matrix.sqlite", query),
11           output="graphs/gflops_%(arch)s_%(M)s.tex",
12           plot_title="%(typ)s",
13           plot_with="linespoints",
14           graph_title="%(arch)s M=%(M)s",
15           xrange="[0:]",
16           yrange="[0:]",
17           xlabel="cores",
18           ylabel="GFLOPS",
19           typ=para_types, M=Ms, arch=archs, graph_vars=[ "arch", "M" ])
```

Translating it into speedup

- Speedup is:

$$\frac{\text{GFLOPS of a program}}{\text{GFLOPS of the serial program for the same parameter}}$$

- So the job is to augment the table with a column of the "GFLOPS of the serial program for the same parameter,"

arch	type	ppn	M	gflops_per_sec	serial_gflops_per_sec
nehalem	serial	1	704	50	50
nehalem	mthtask	1	704	48	50
nehalem	cilk	1	704	49	50
sandybridge	serial	1	704	90	90
sandybridge	mthtask	1	704	88	90
sandybridge	cilk	1	704	89	90
...	...	...	...	...	...

Preprocessing

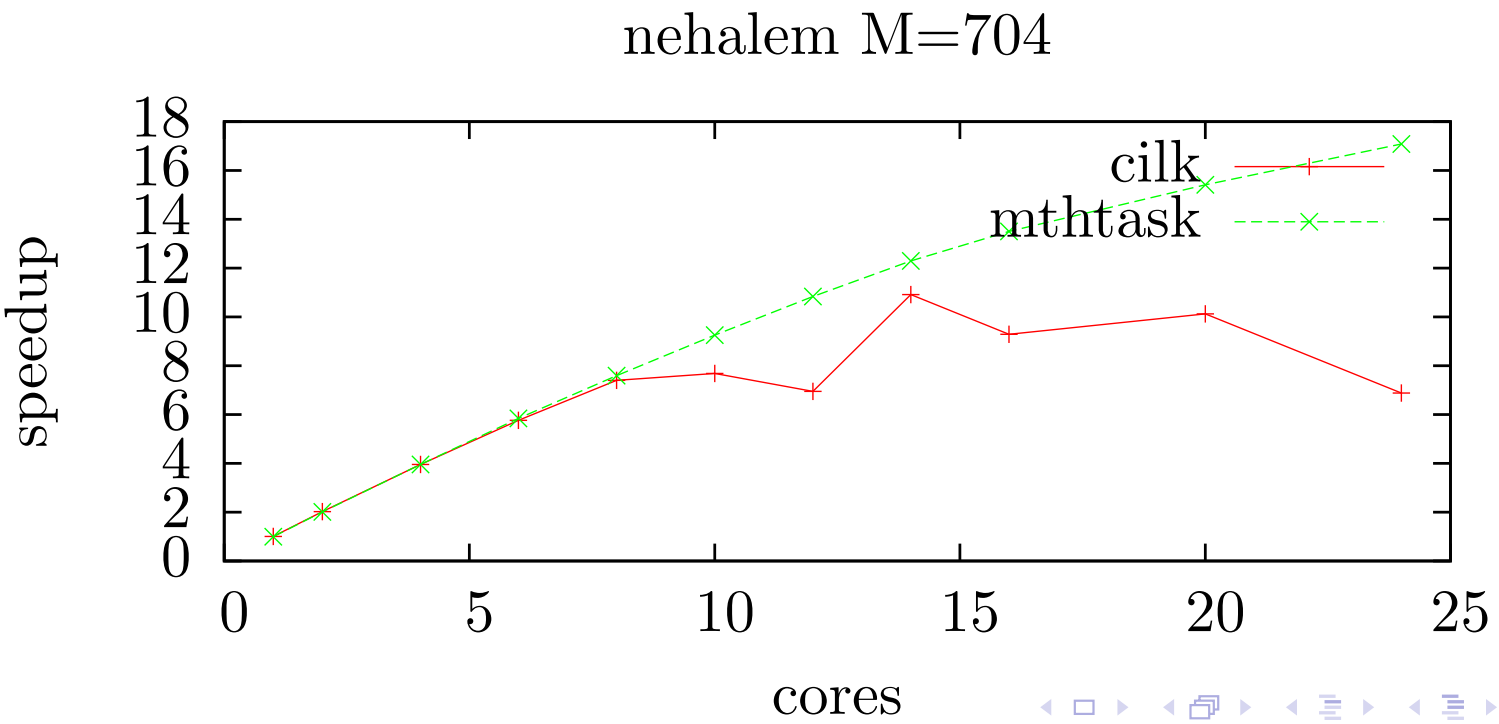
```
1 create temp table serial as
2 select arch,M,avg(gflops_per_sec) as serial_gflops_per_sec
3 from a where type = "serial"
4 group by arch,M;
5
6 create temp table b as select * from serial natural join a;
```

Note: you'd better create *temporary* tables only, so you may freely repeat it

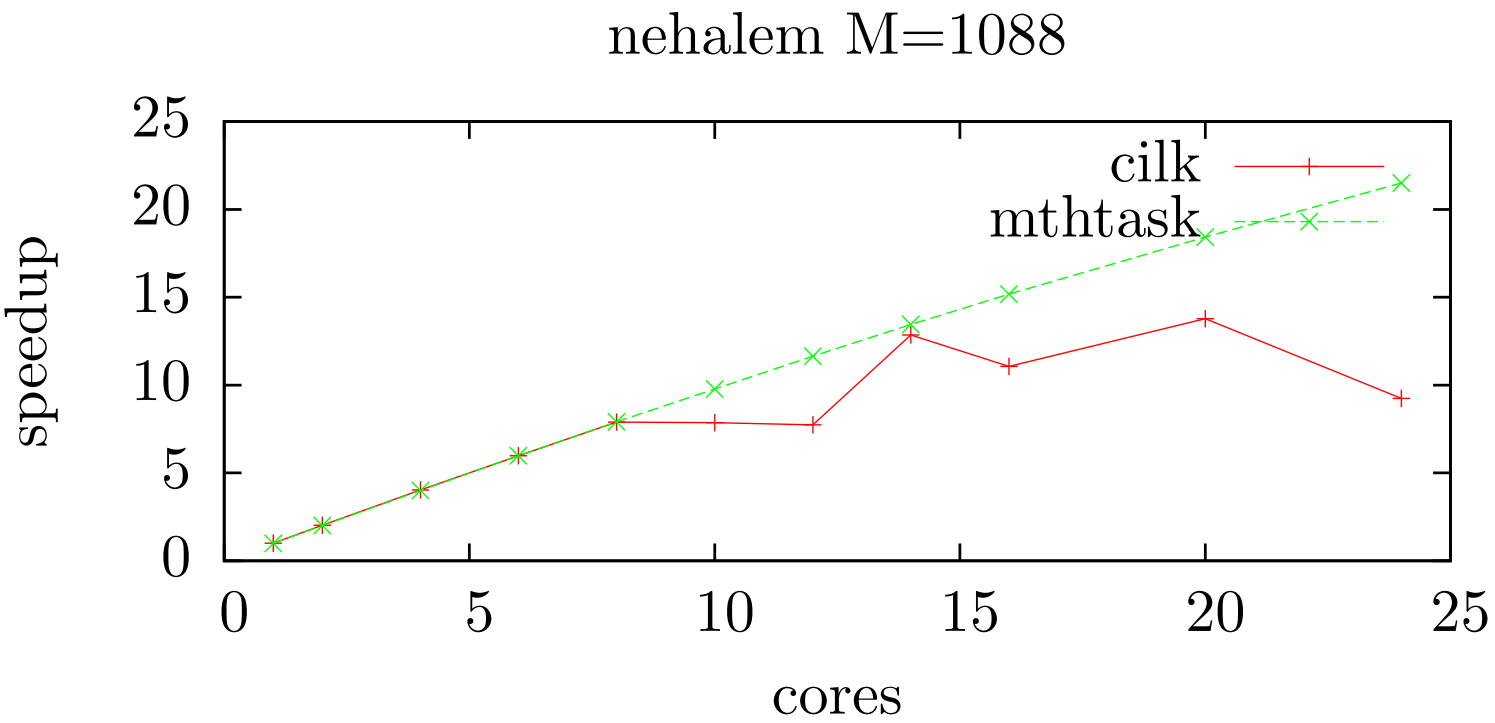
The query

The query itself is easy:

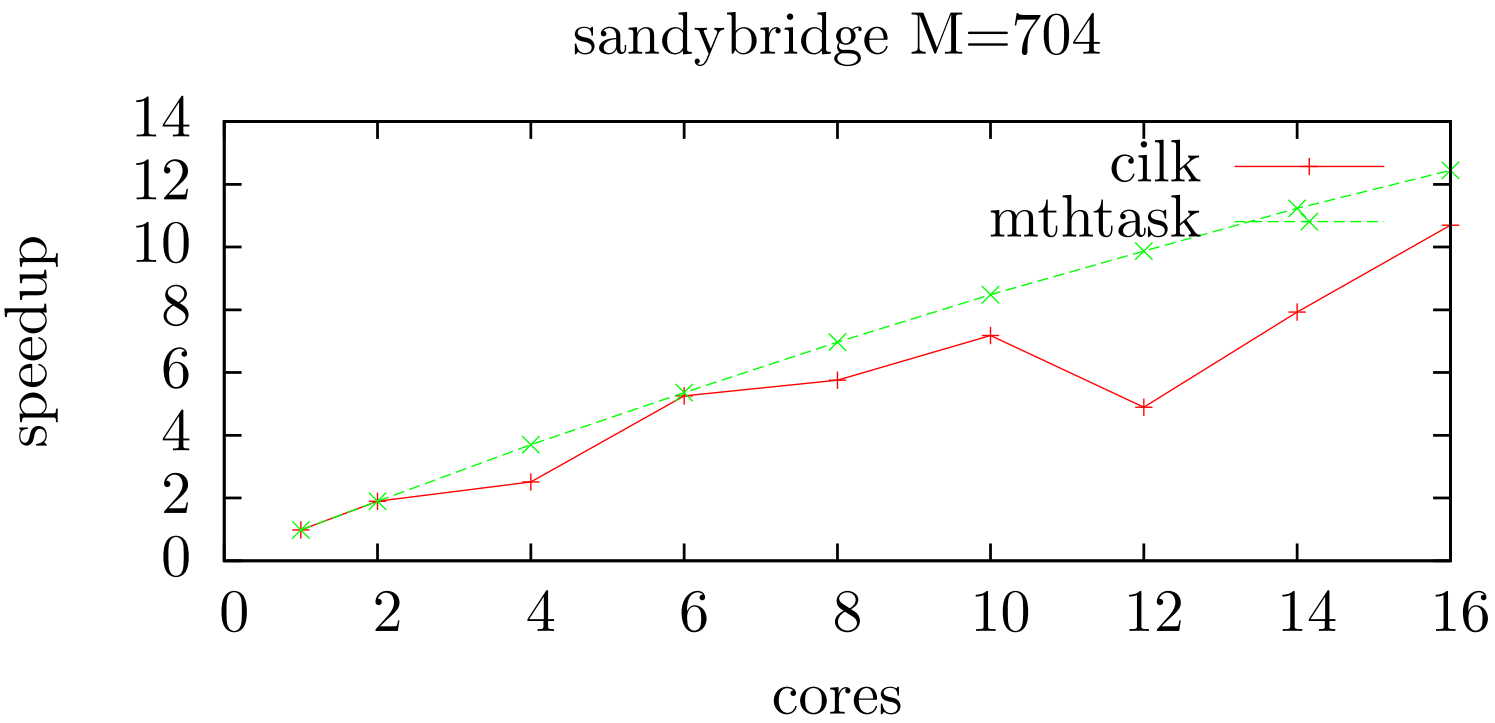
```
1 select ppn,avg(gflops_per_sec / serial_gflops_per_sec) from b
2 where type="%(typ)s" and M=%(M)s and arch="%(arch)s" group by ppn
```



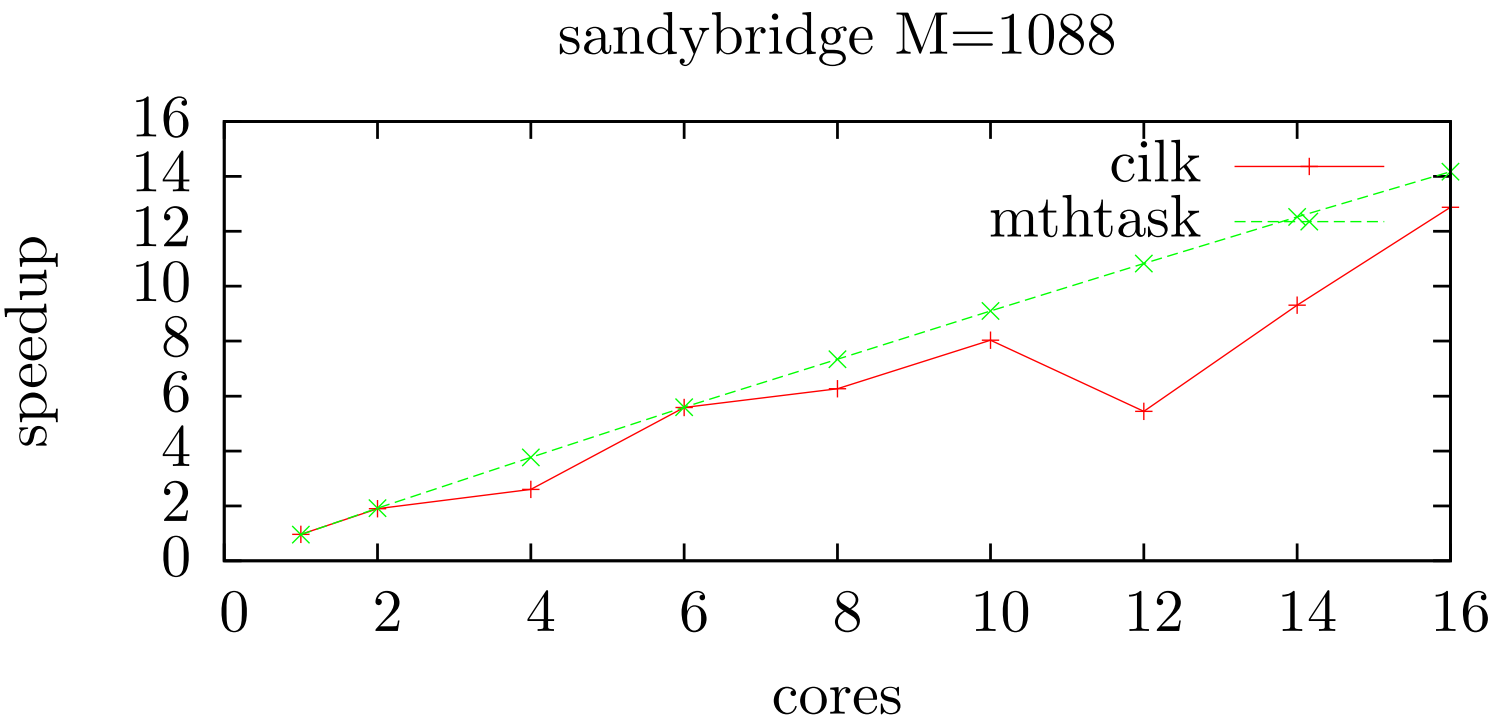
Nehalem, $M = 1088$



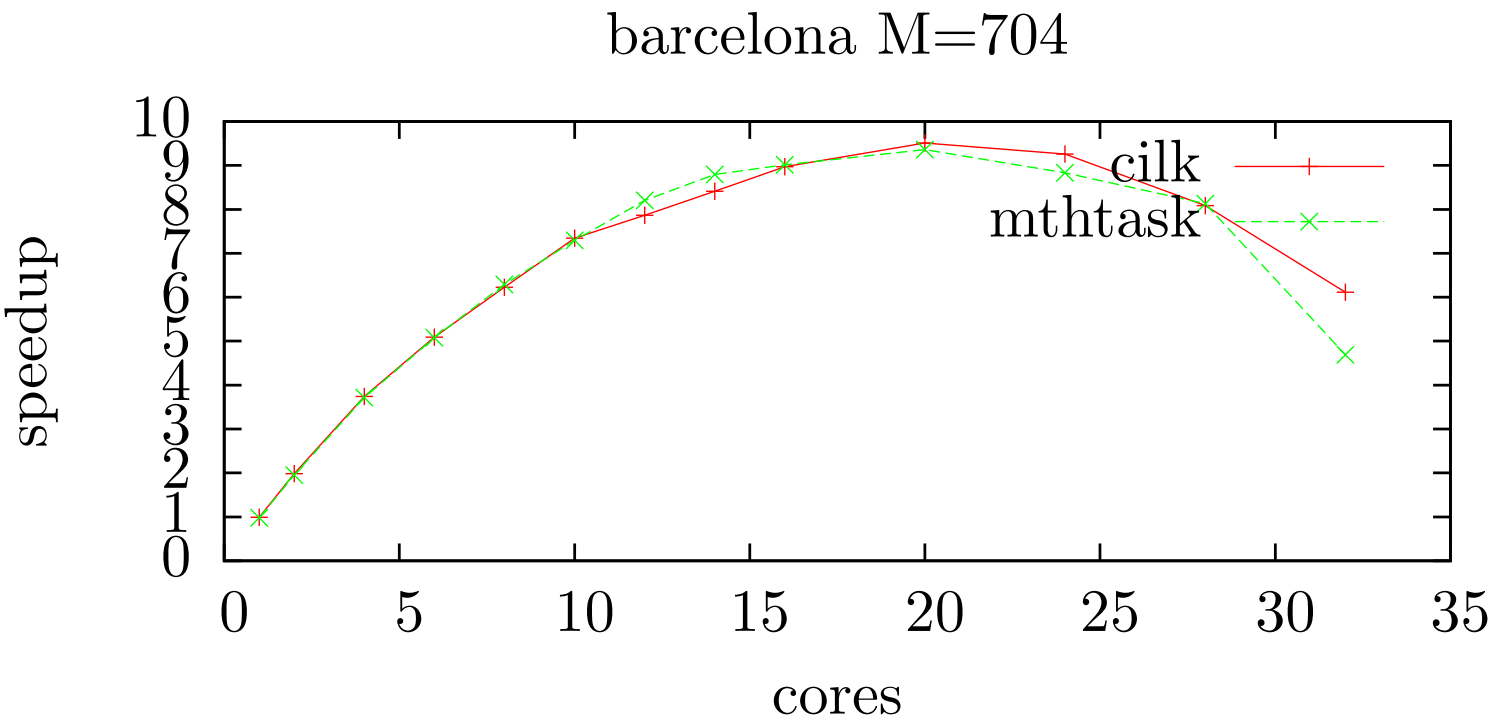
Sandy Bridge, $M = 704$



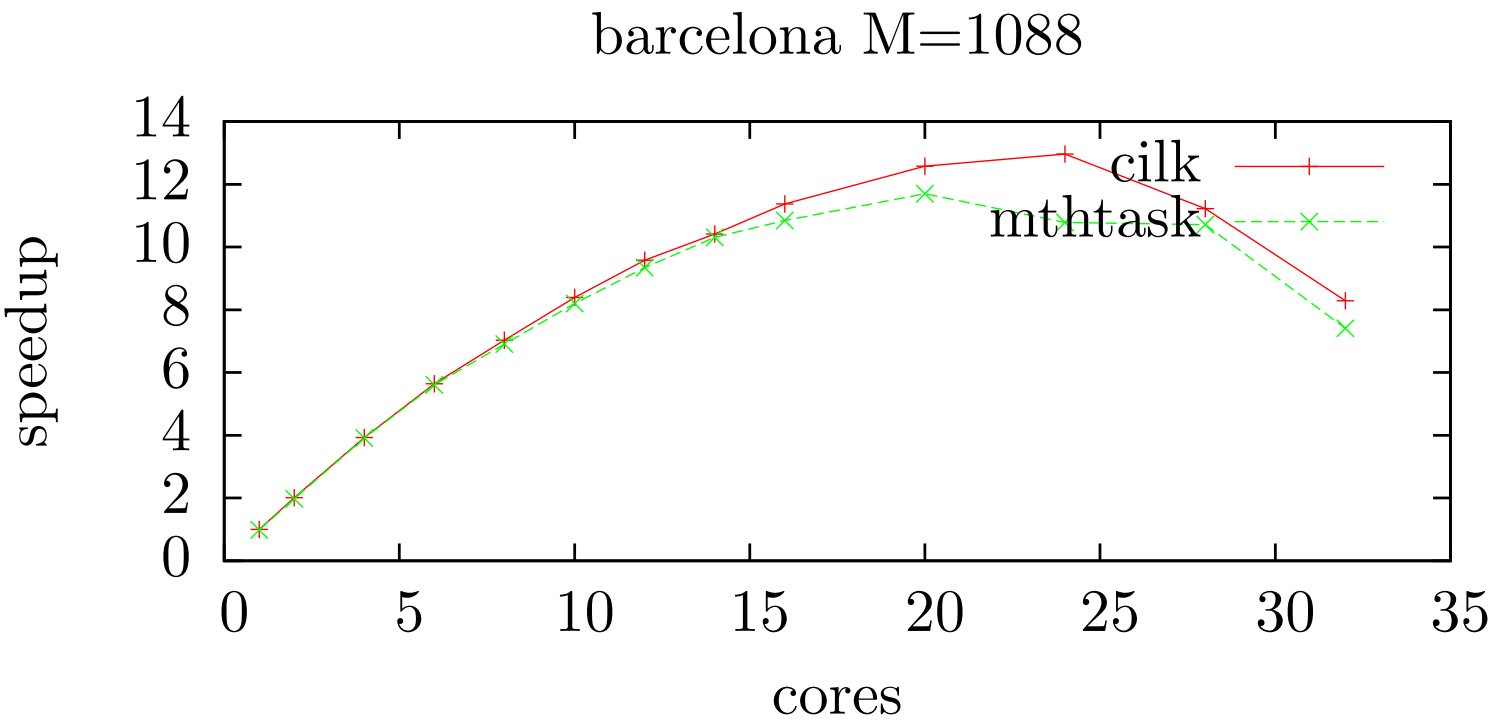
Sandy Bridge, $M = 1088$



Barcelona, $M = 704$



Barcelona, $M = 1088$

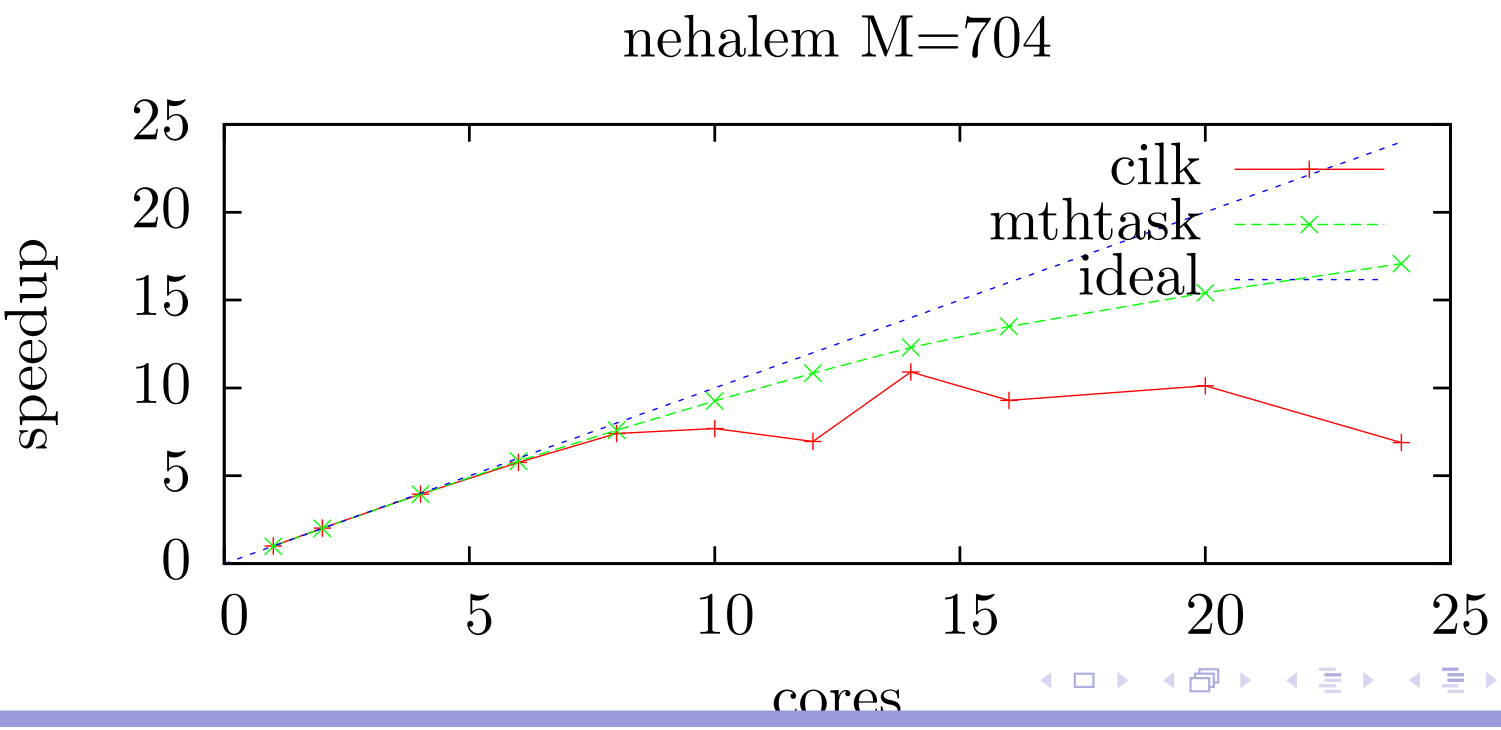


The real code that shows the 6 graphs and save them

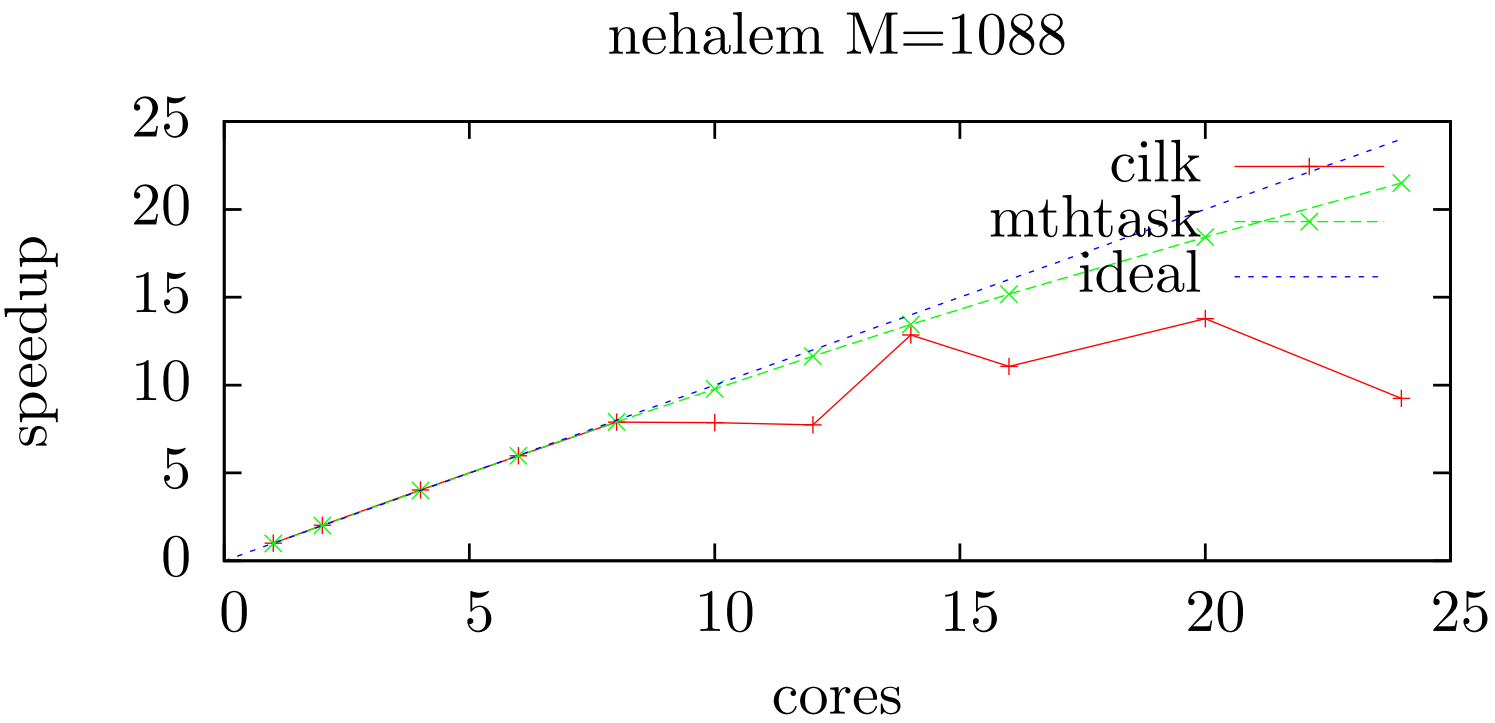
```
1  init = r'''create temp table serial as
2  select arch,M,avg(gflops_per_sec) serial_gflops_per_sec
3  from a where type = "serial" group by arch,M;
4
5  create temp table b as select * from serial natural join a;
6  '''
7  query=r'''select ppn,avg(gflops_per_sec / serial_gflops_per_sec) from b
8  where type="%(typ)s" and M=%(M)s and arch="%(arch)s" group by ppn'''
9
10 g.graphs(("matrix.sqlite", query, init),
11           output="graphs/speedup_%(arch)s_%(M)s.tex",
12           plot_title="%(typ)s",
13           plot_with="linespoints",
14           graph_title="%(arch)s M=%(M)s",
15           xrange="[0:]",
16           yrange="[0:]",
17           xlabel="cores",
18           ylabel="speedup",
19           typ=para_types, M=Ms, arch=archs, graph_vars=[ "arch", "M" ])
```

Overlaying "ideal" speedup

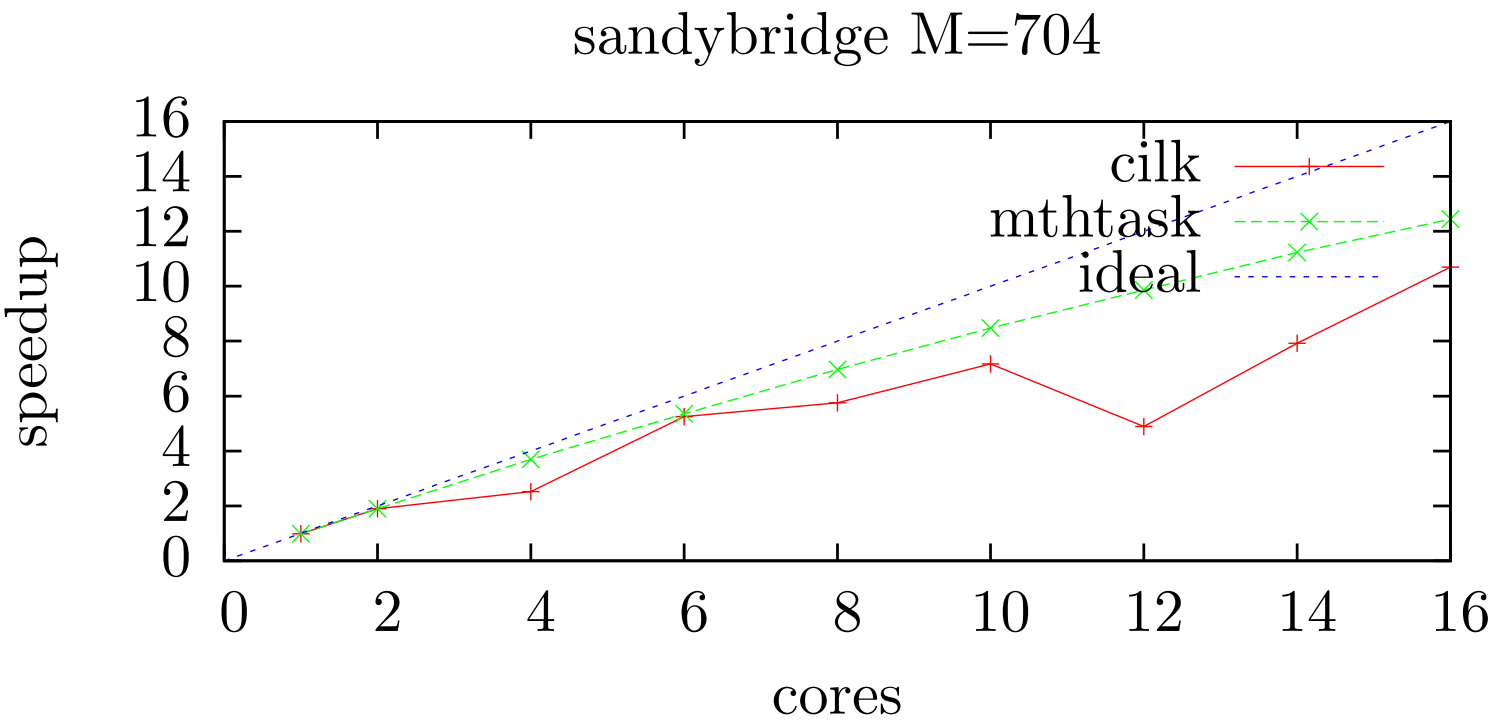
```
1  g.graphs(("matrix.sqlite", query, init),
2           output="graphs/speedup_%(arch)s_%(M)s.tex",
3           overlays=[("x",  "plot_title" :  "ideal" )],
4           ...)
```



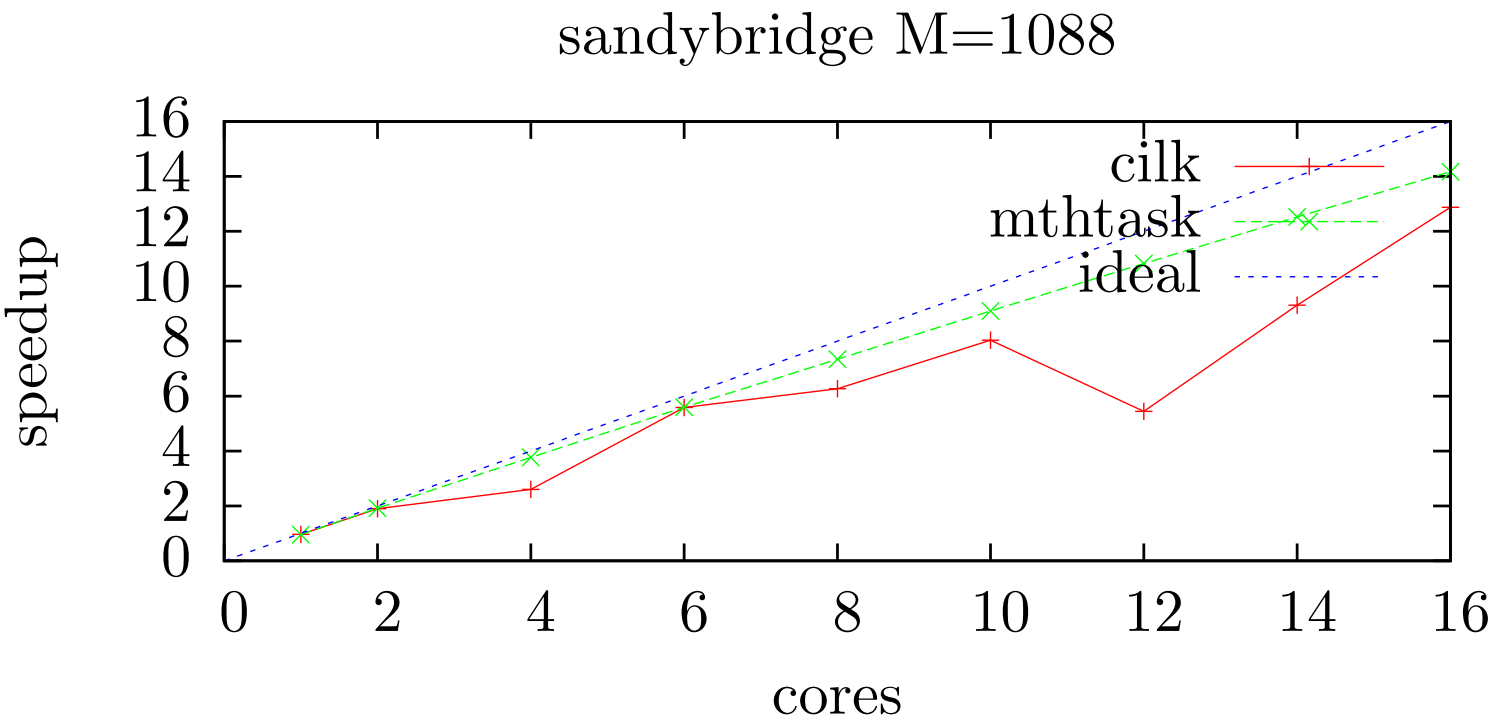
Nehalem, $M = 1088$



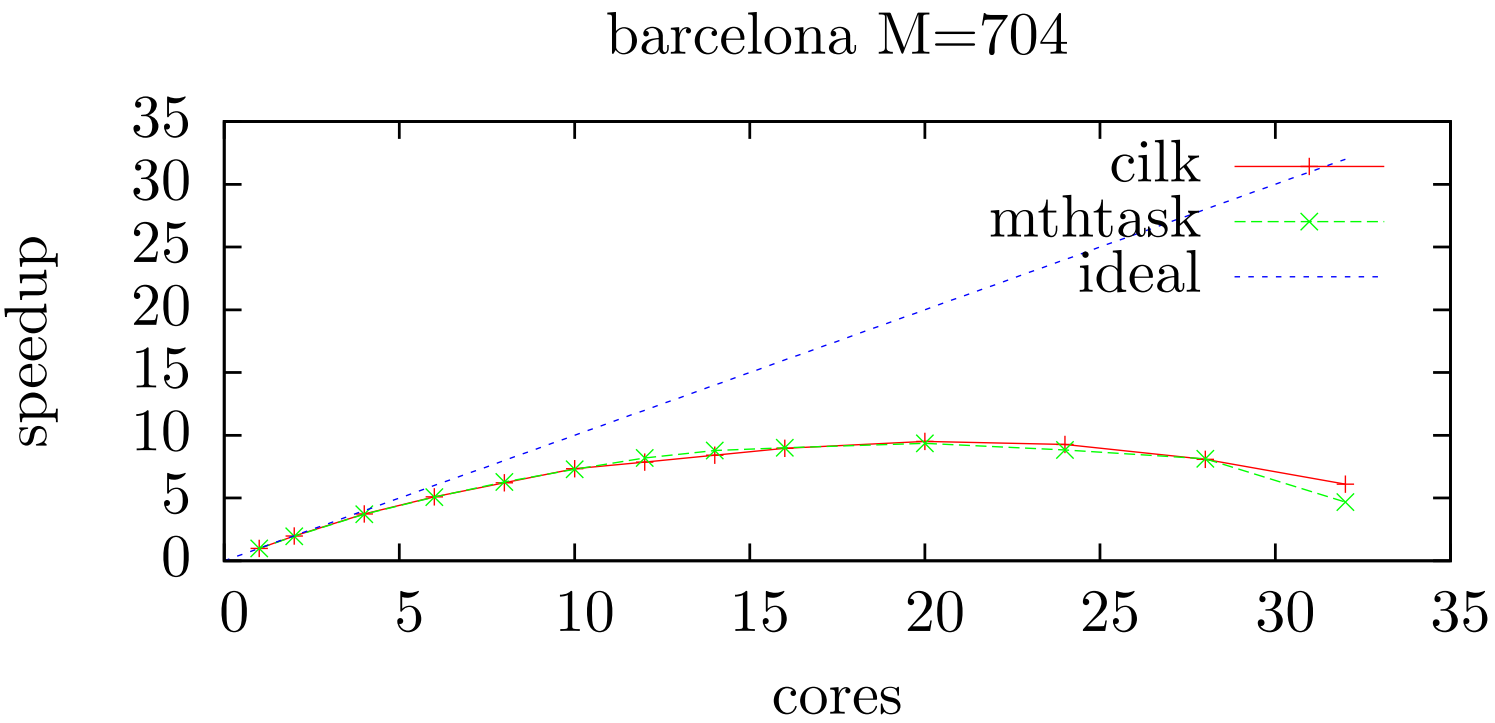
Sandy Bridge, $M = 704$



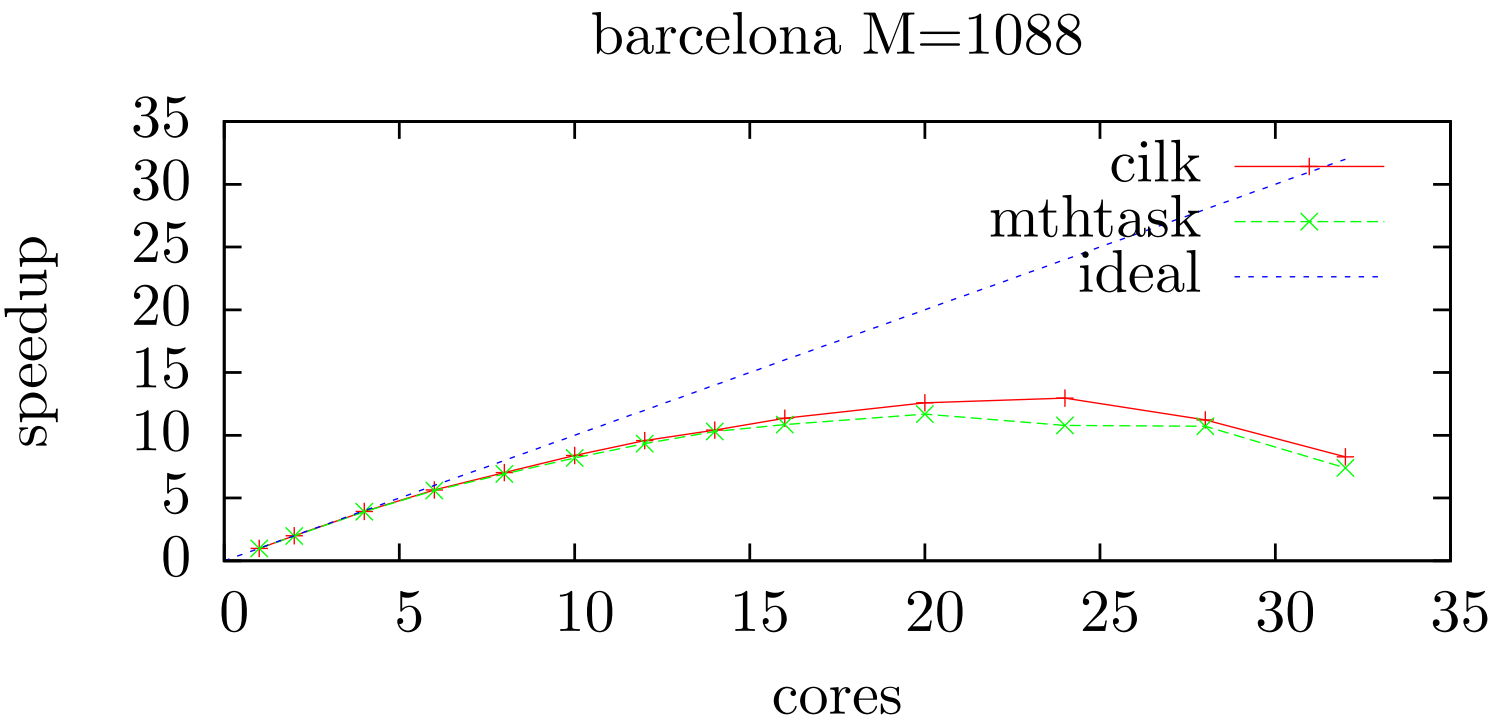
Sandy Bridge, $M = 1088$



Barcelona, $M = 704$



Barcelona, $M = 1088$



Adding confidence interval

- smart_gnuplotter extends sqlite3 with two user defined aggregates cimin and cimax
- you may also define your own user defined functions/aggregates/collations
- seeing them is important especially when your results do not seem very stable

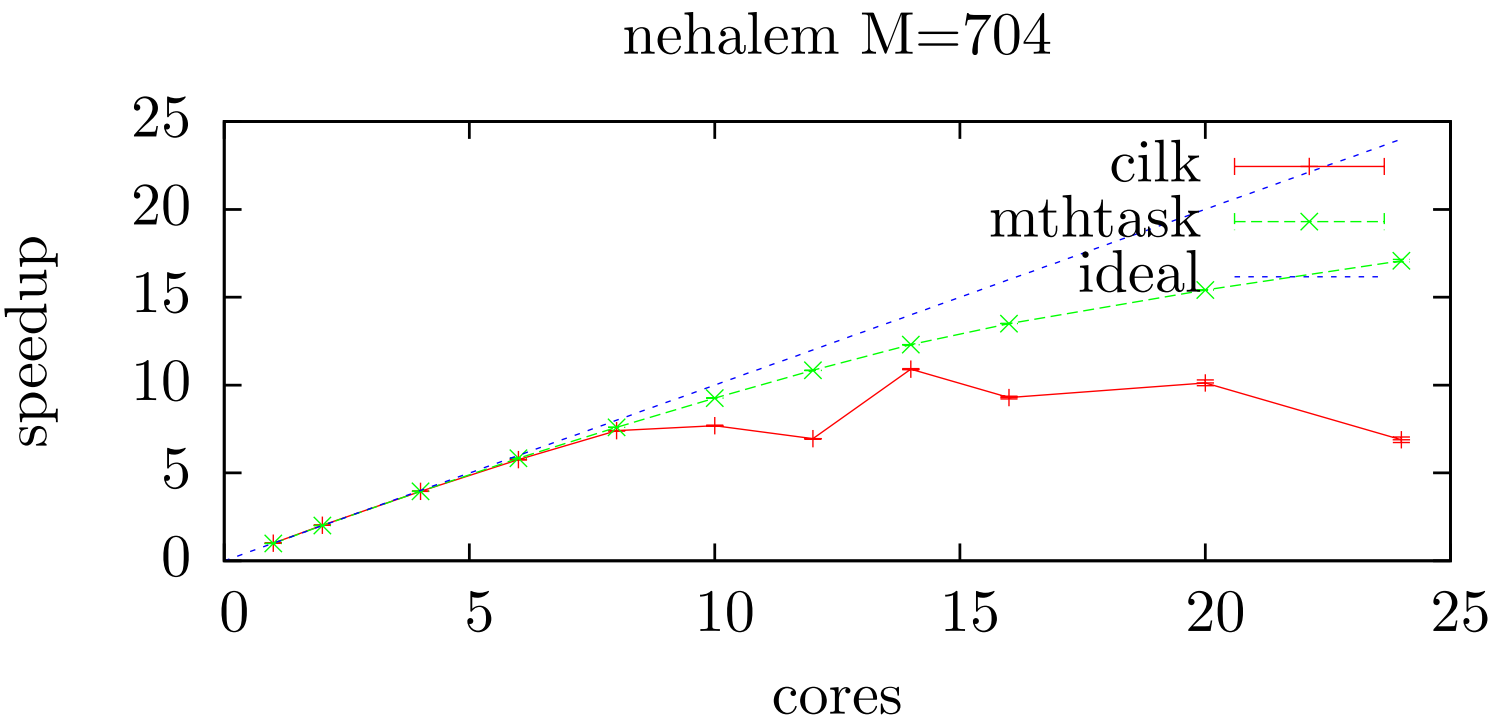
The basic form is to generate 4-columns data:

```
1 select X,avg(Y),cimin(Y, 0.05),cimax(Y, 0.05) from
```

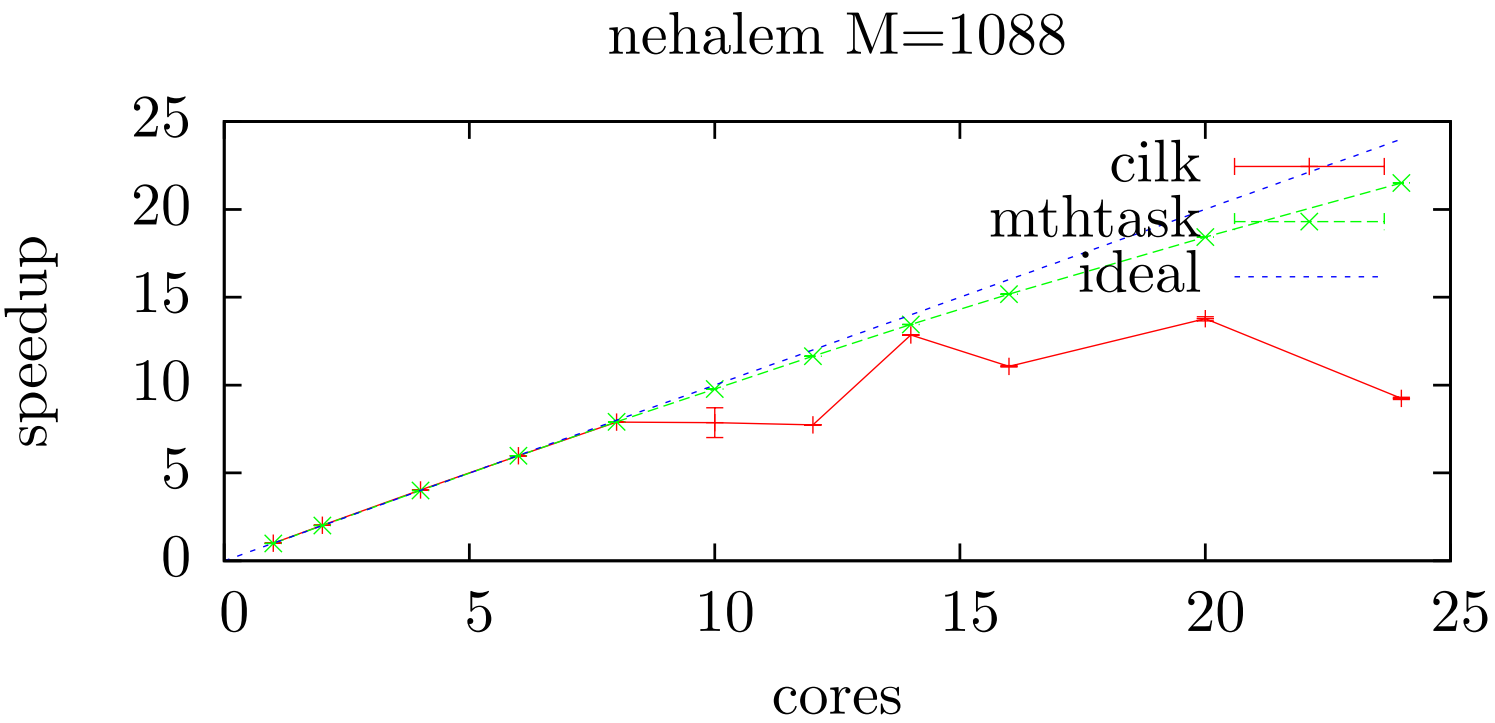
and plot them with yerrorlines style.

```
1 g.graphs(...,
2          plot_with="yerrorlines",
3          ...)
```

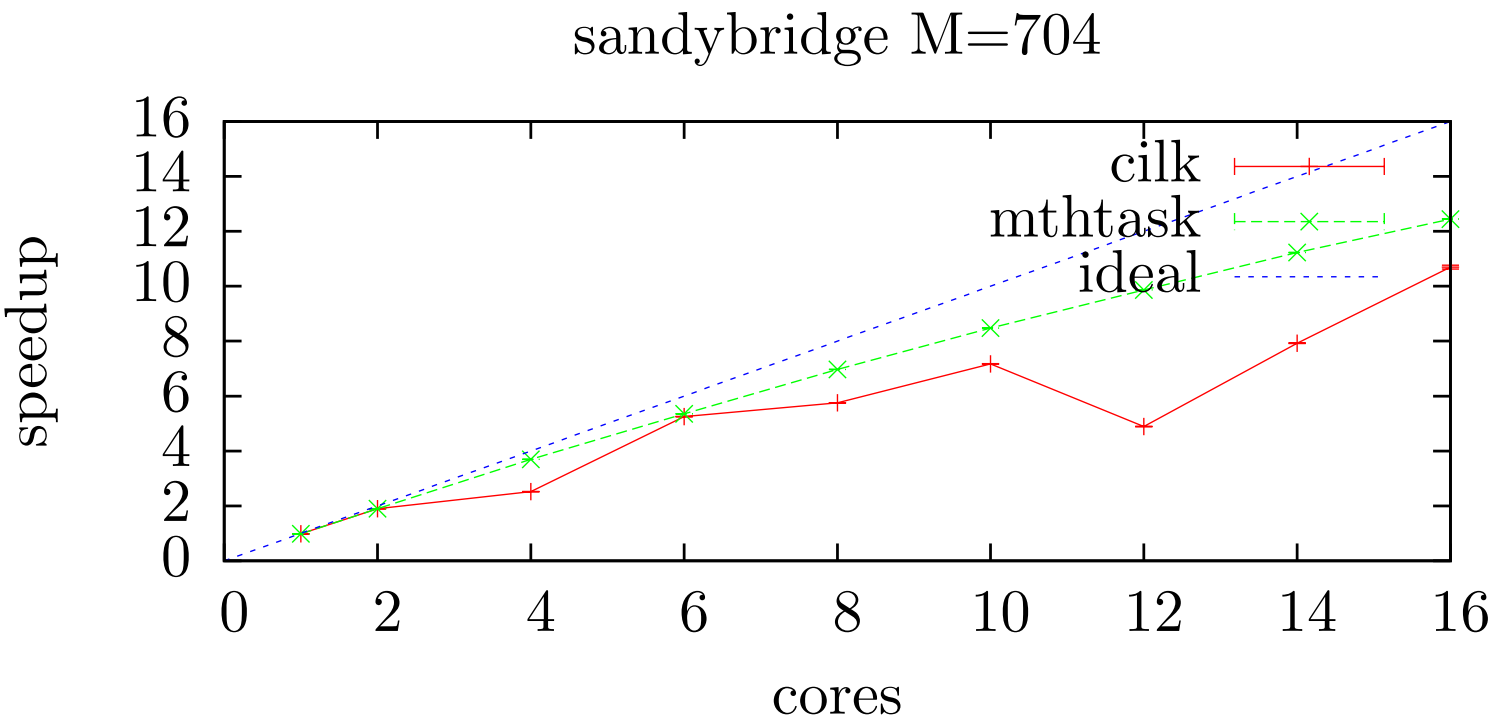
Nehalem, $M = 704$



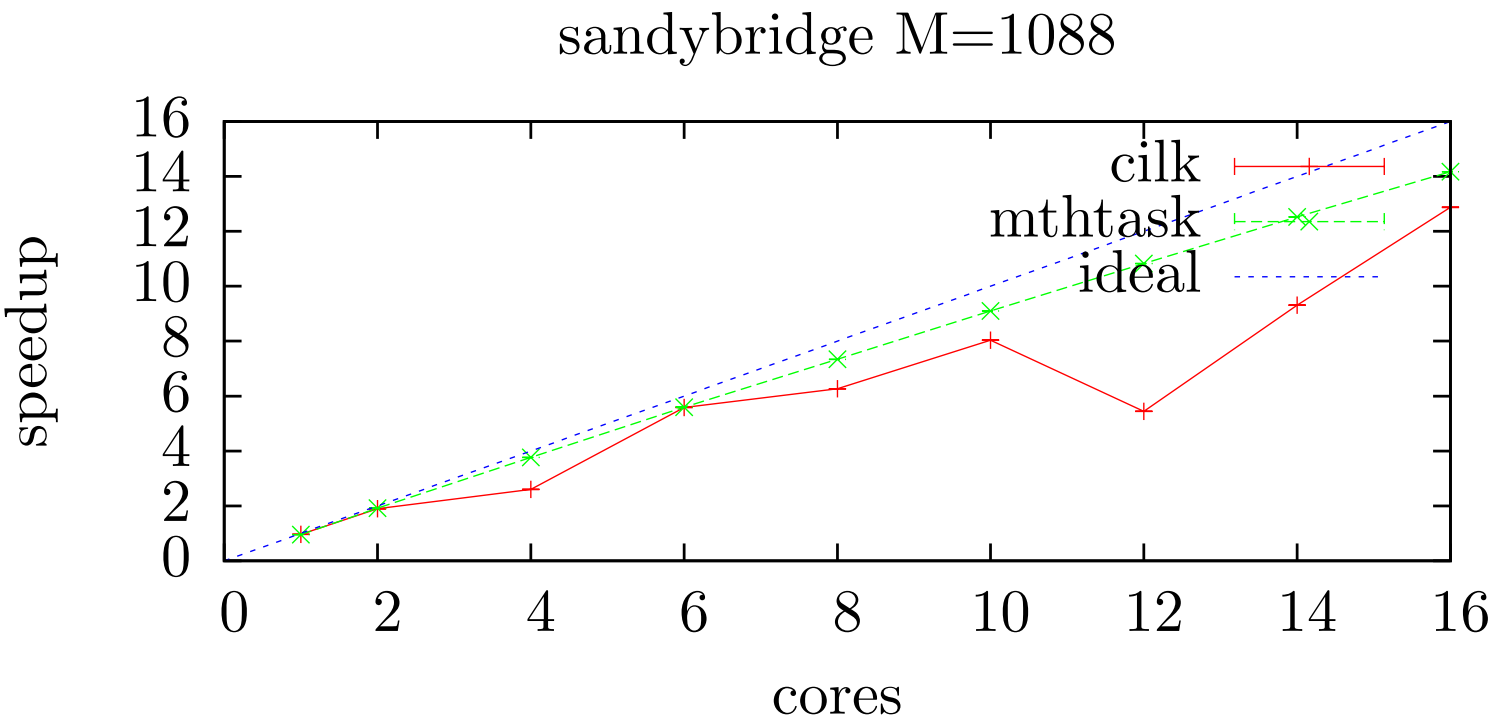
Nehalem, $M = 1088$



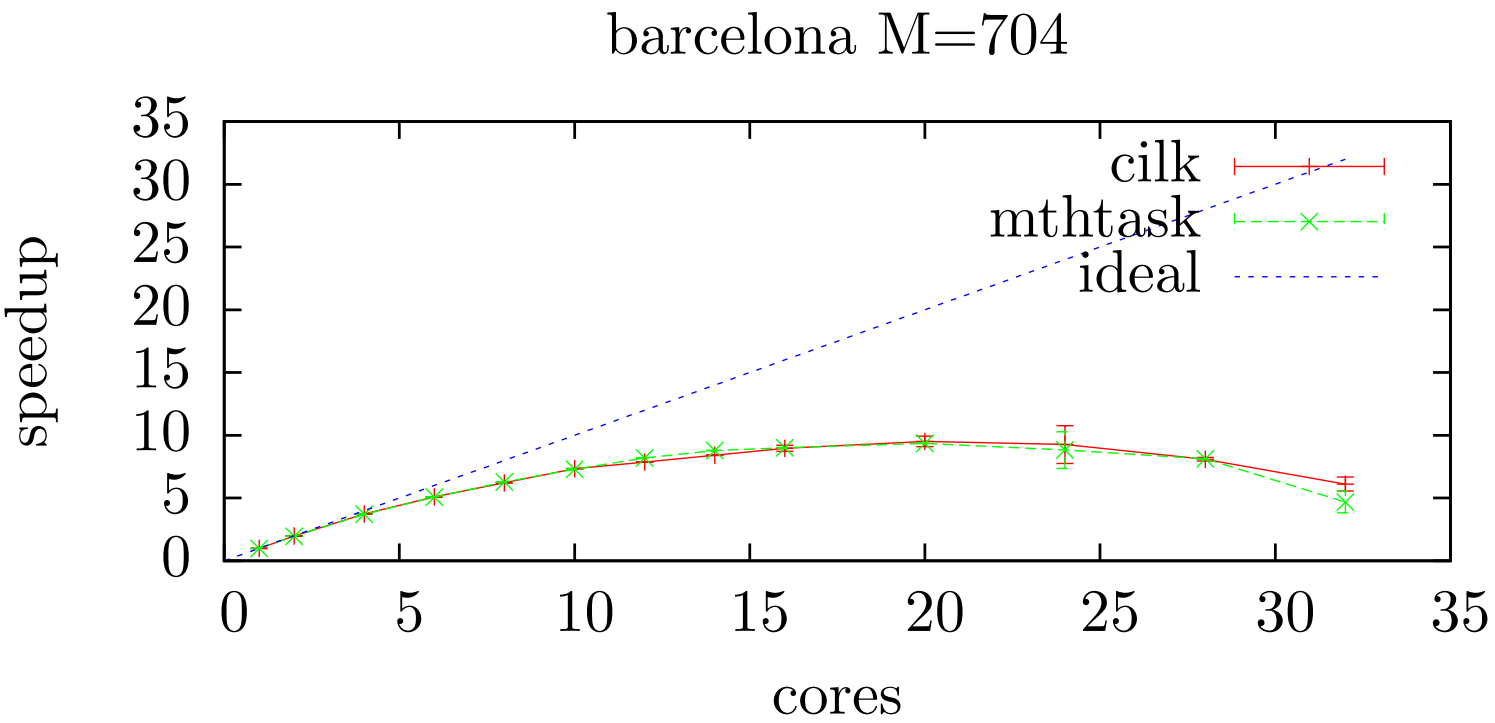
SandyBridge, $M = 704$



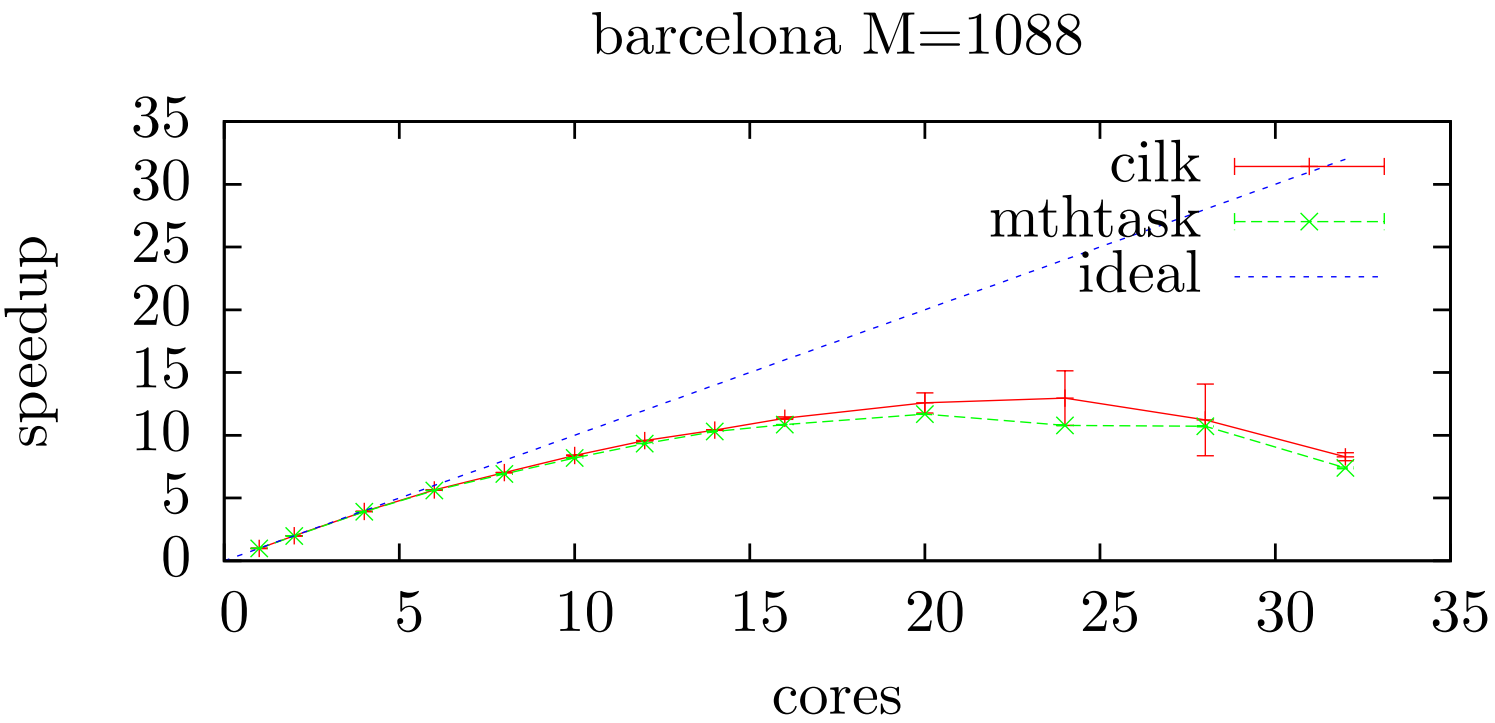
SandyBridge, $M = 1088$



Barcelona, $M = 704$



Barcelona, $M = 1088$



Notes on the experimental results

- Do not trust Cilk results. It will become significantly better by binding workers to cores
  - Now I proved the importance of automating the process!
- Results on Sandy Bridge (hongo600) will have been affected by TurboBoost, which boosts performance on a single core
  - We should confirm it by turning TurboBoost off on hongo6xx, but utilizing these dynamic behaviors increasingly sophisticated may be an interesting direction
- Results on Barcelona is miserable and investigation will lead to a better work stealer

Parameters that change together

- So the basic form is:

```
1 g.graphs("expression", var=values, var=values, ...)
```

- But you might not want to generate all combinations
- For example, you may want to have $(a, b) = (”hoge”, 10), (”bar”, 20)$, not $(”hoge”, 20)$ or $(”bar”, 10)$
- You may do so by having a parameter assuming tuple values:

```
1 X=[("hoge",10),("bar",20)]
```

and refer to them as %(X[0])s, %(X[1])s, etc.

Are You Brainwashed?

An exclusive offer for you!

- I put a template directory so you can get started now!

```
1 cp -r labdocs/common/tau/new_doc your_new_document
2 cd your_new_document
3 make
```

- It has a structure putting figures, images, databases, python scripts, and Makefile to compile everything into pdf.

The document template

- Just 'make' and you get two pdf files
  - a document
  - a beamer slide
- It uses the following programs and TEX packages.
  - platex .tex → .dvi
  - dvipdfmx .dvi → .pdf
  - inkscape .svg → .eps
  - convert and ebb to get bounding box of them
  - convert .jpg/.png/.gif → .eps
  - graphicx, listings, dvipdfm TEX packages
  - beamer TEX package (for slide)

Other things that will be useful

- generate TEX table (should be easy)
- generate Excel file (useful when you prefer/need PowerPoint)