Some hints to improve compilation time and execution performance

HEPIX Spring 2006

April 3-7, 2006

René Brun
CERN

Plan of talk

- Time to compile
  - May be a problem in some experiments
  - Some recipes for improvement
- Shared libs
- Improving the execution time
  - Code inlining (good and bad aspects)
  - Using the right collection classes
  - Profiling tools
- Differences between compilers or compiler versions
- Ready for Multithreading

Compilation time

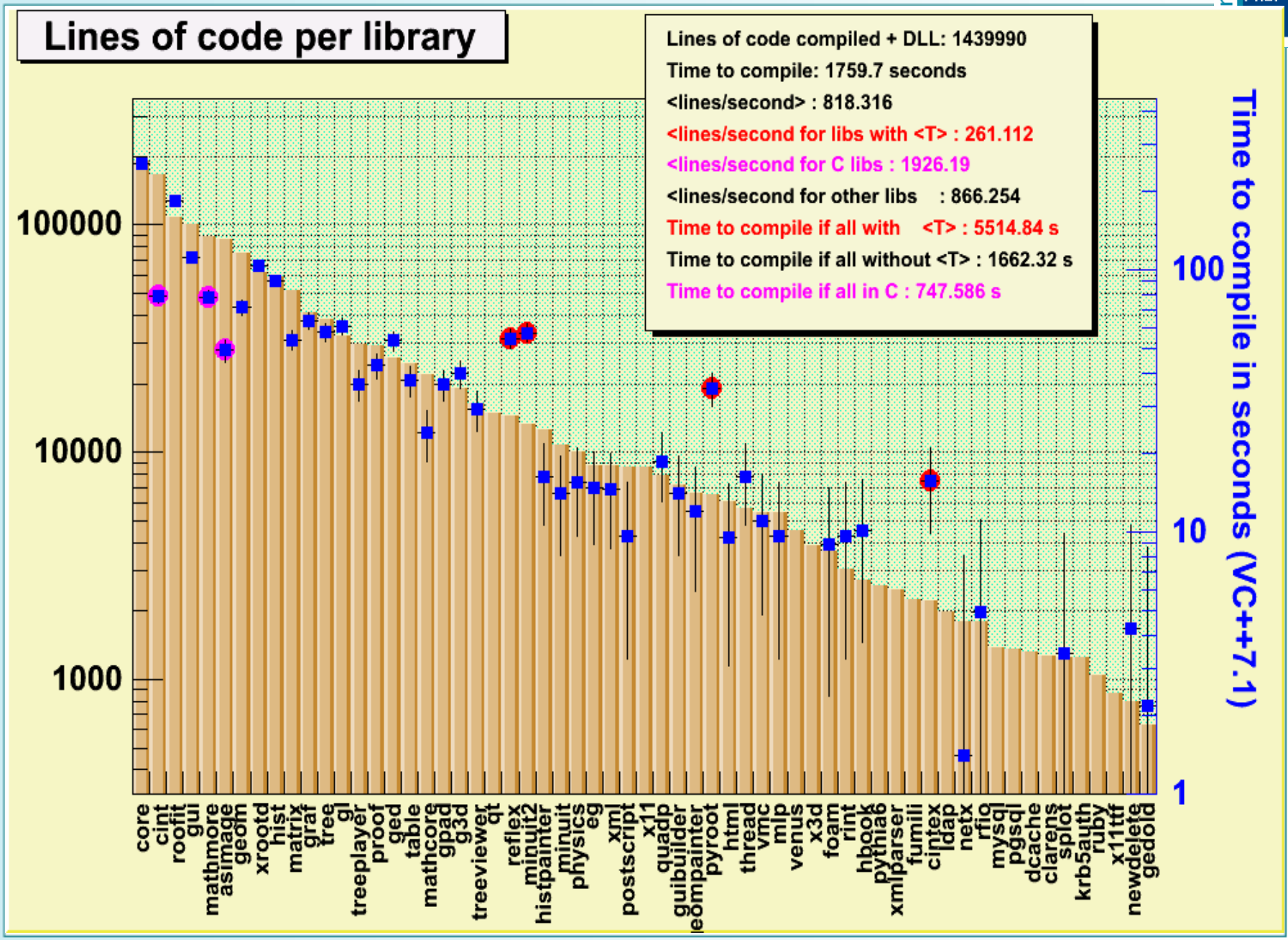
becoming a problem (Atlas, CMS)

Many causes, in particular abuse of templated code

LHC software

	Alice	Atlas	CMS	ROOT
number of lines in header files	102282	698208	104923	153775
classes total	1815	8910	???	1500
classes in dict	1669	>4120 2140	835	1422
lines in dict	479849	455705	103057	698000
classes c++ lines	577882	1524866	277923	857390
total lines Classes+dict	1057731	???	380980	1553390
total f77 lines	736751	928574	???	3000
directories	540	19522	<500	958
comp time	25'	750'	90'	30'
lines compiled/s	1196	50 (70)	71	863

Lines of code per library



Problem with STL Inlining

- STL containers are very nice. However they have a very high cost in a real large environment.
- Compiling code with STL is much much slower because of inlining (STL is only in header files). The situation improves with precompiled headers (eg in gcc4 and vc++).
- Object modules are bigger
- Compiler or linker is able to eliminate duplicate code in ONE object file or shared lib, not across libraries.
- If you have 100 shared libs, it is likely that you have the code for std:vector push_back or iterators 100 times!
- In-lining is nice if used with care (or toy benchmarks). It may have an opposite effect, generating more cache misses in a real application.
- Templates are statically defined and difficult to use in an dynamic interactive environment.

Example with include <string>

- This includes more than 20000 lines of C++ code!!!
- <string>, and also <vector>, <list> is used by nearly every C++ file in Atlas and CMS
- On many systems (eg Solaris/CC) <string> includes many other includes, in turn including other includes!!

```
/opt/SUNWspro/WS6U1/include/CC/std/stdio.h
        /usr/include/sys/feature_tests.h
            /usr/include/sys/isa_defs.h
        /usr/include/stdio.h
            /usr/include/iso/stdio_iso.h

/usr/include/sys/feature_tests.h

/usr/include/sys/va_list.h

/usr/include/stdio_tag.h

/usr/include/stdio_impl.h

/usr/include/sys/isa_defs.h

/opt/SUNWspro/WS6U1/include/CC/std/string.h
        /usr/include/sys/feature_tests.h
        /usr/include/string.h

/usr/include/iso/string_iso.h
```

```
usr/include/sys/feature_tests.h

/opt/SUNWspro/WS6U1/include/CC/std/ctype.h

/usr/include/sys/feature_tests.h
        /usr/include/ctype.h

/usr/include/iso/ctype_iso.h

/usr/include/sys/feature_tests.h
usr/include/sys/types.h
        /usr/include/sys/isa_defs.h
        /usr/include/sys/feature_tests.h
        /usr/include/sys/machtypes.h

/usr/include/sys/feature_tests.h
        /usr/include/sys/int_types.h
            /usr/include/sys/isa_defs.h
```

```
/usr/include/fcntl.h

/usr/include/sys/feature_tests.h
        /usr/include/sys/types.h
        /usr/include/sys/fcntl.h

/usr/include/sys/feature_tests.h

    /usr/include/sys/types.h
/usr/include/sys/stat.h
        /usr/include/sys/feature_tests.h
        /usr/include/sys/types.h
        /usr/include/sys/time_std_impl.h

/usr/include/sys/feature_tests.h
        /usr/include/sys/stat_impl.h

/usr/include/sys/feature_tests.h
            /usr/include/sys/types.h
    ……
```

The case of ROOT::smatrix

- smatrix is a small package in ROOT. All the code is in include smatrix.h (like STL headers).
- The package is designed to be efficient for small matrices.
- However, the experience shows that extensive use of smatrix increases the application compilation time by huge factors,
- eg smatrix/test/testOperations (about 1400 lines) compiles in less than 2 seconds with the standard TMatrix and takes 1 minute with smatrix calls.
- The object file testOperations.o is also very large (800 Kbytes) where the TMatrix version is only 30 Kbytes.

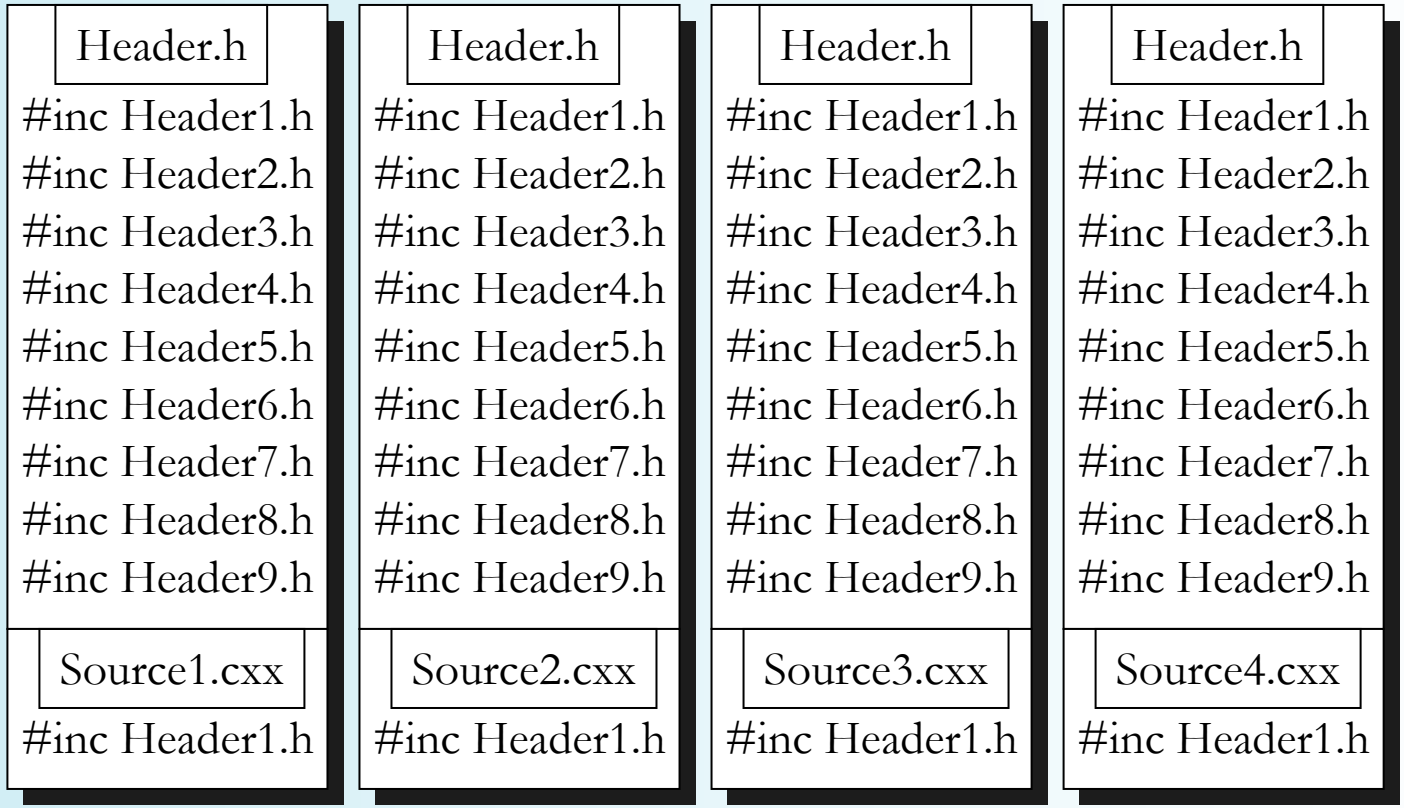
Precompiled headers

Reduce compilation time

But tricky to implement

Headers

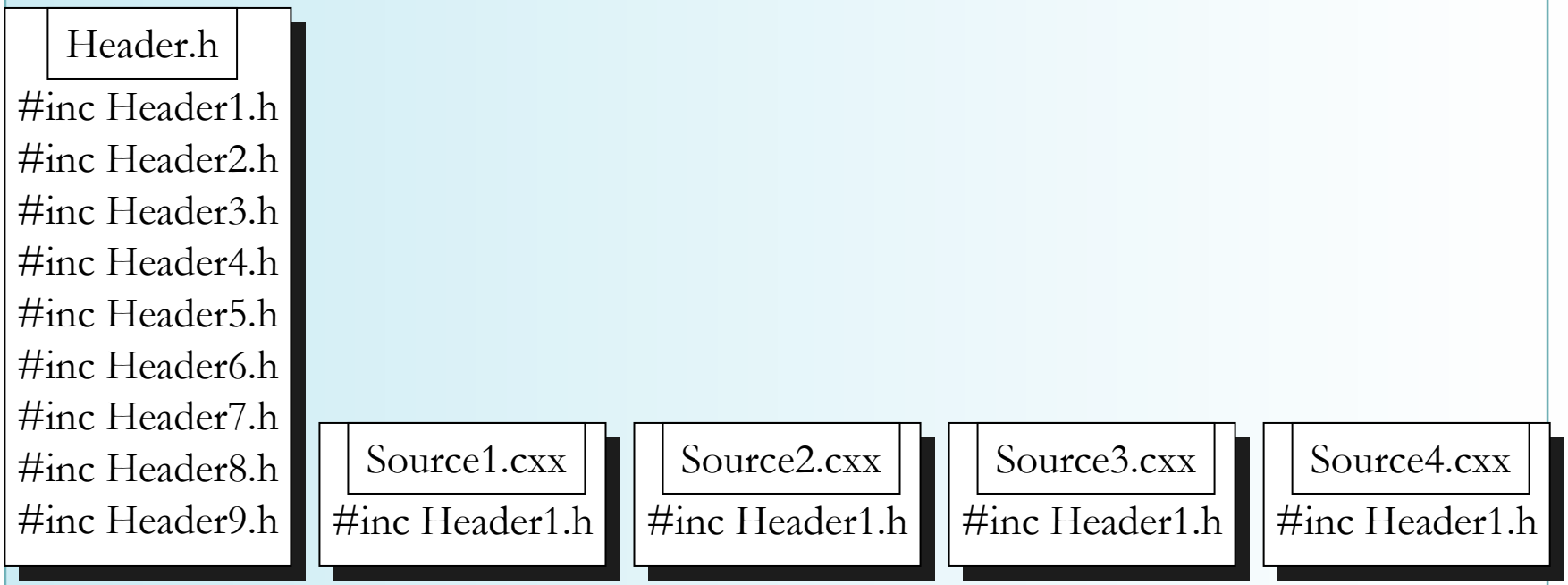
- #include is copy & paste headers into sources:



- The compiler compiles everything that's white

Precompiled Headers

- #include of a precompiled header: first compile header, then sources that include it



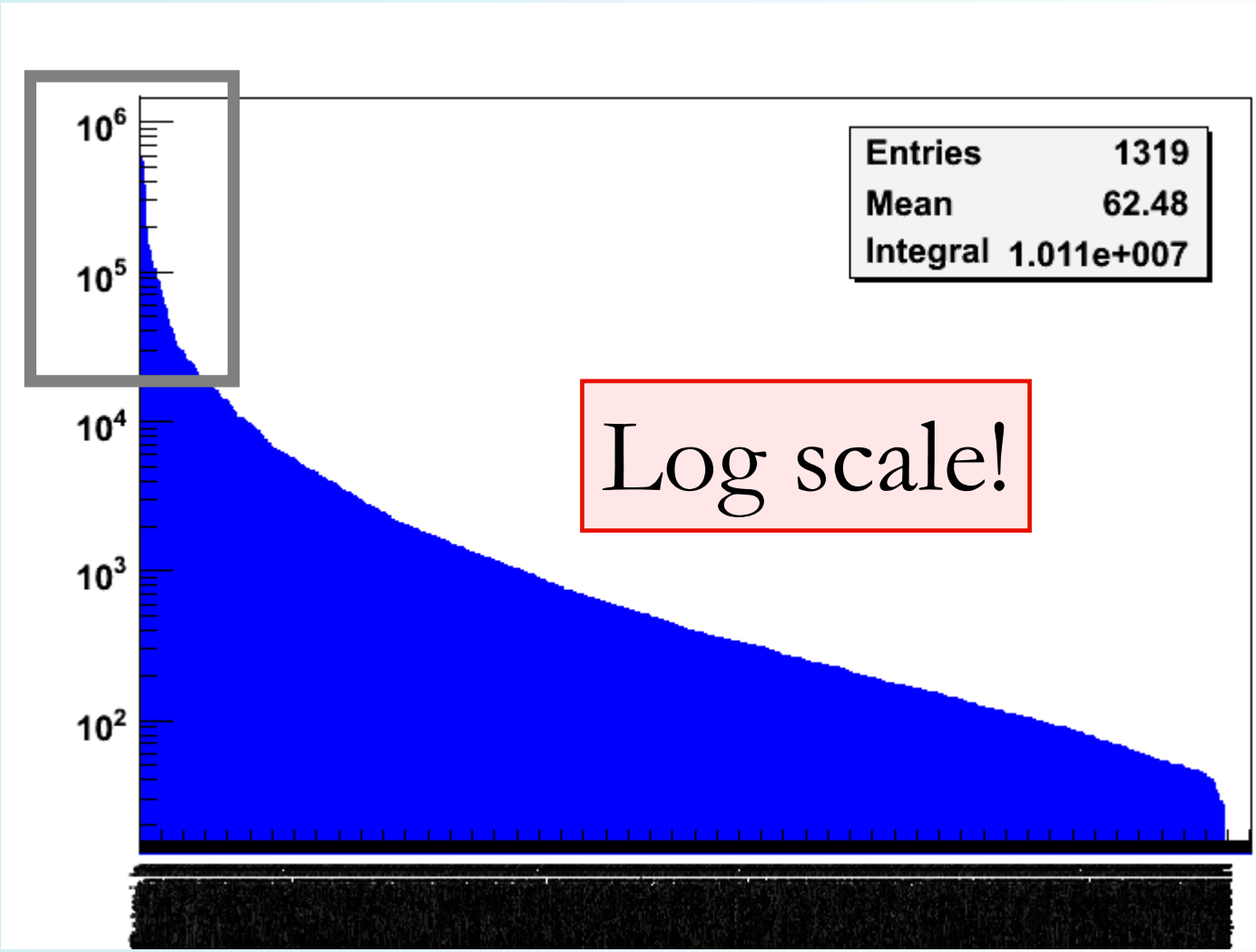
- A lot less to compile!

Precompiled Headers – Statistics in ROOT

- Number #inc:
- 955 RConfig.h
- 939 RVersion.h
- 935 DllImport.h
- 934 Rtypes.h
- 934 Rtypeinfo.h
- 915 TGenericClassInfo.h
- 890 Varargs.h
- 888 Riosfwd.h
- 882 TStorage.h
- 882 TObject.h

- Number #inc * lines:
- 758254 G__ci.h
- 580957 TMath.h
- 573705 TString.h
- 541548 TBuffer.h
- 534060 Bytes.h
- 448850 RConfig.h
- 378270 Rtypes.h
- 206856 TClass.h
- 193011 TROOT.h
- 185220 TObject.h

Precompiled Headers – Statistics



Precompiled Headers - ROOT

- All headers compiled as included: 10M lines
- All headers compiled once: 0.27M lines = 3%!

- Context (compiler flags, #included files before) has to be fixed for compiled headers to work
- Consequence: always #include the same set of headers for all sources!

Precompiled Headers - Optimum

- Optimum between precompile all / no headers.
- Current “set”: precompile only TH1.h and its #includes
- = 49 ROOT header files
- = 156 headers incl. GCC system headers

Precompiled Headers - ROOT

- Current CVS: enabled by default on GCC > 4.0.0, Windows MSVC ≥ 7.1, ICC .

- touch Rtypes.h && make
- -30% on SL4 GCC 4.0.2 debug (11 vs. 16min)
- -35% on Win MSVC8 debug (22 vs. 34mins)

- Full rebuilds benefit, too.

Problems with Shared libraries

Large applications have 10’,100’ shared libs

Increased startup time (interactive)

Use plug-in manager

Minimize exported symbols

Shared libs

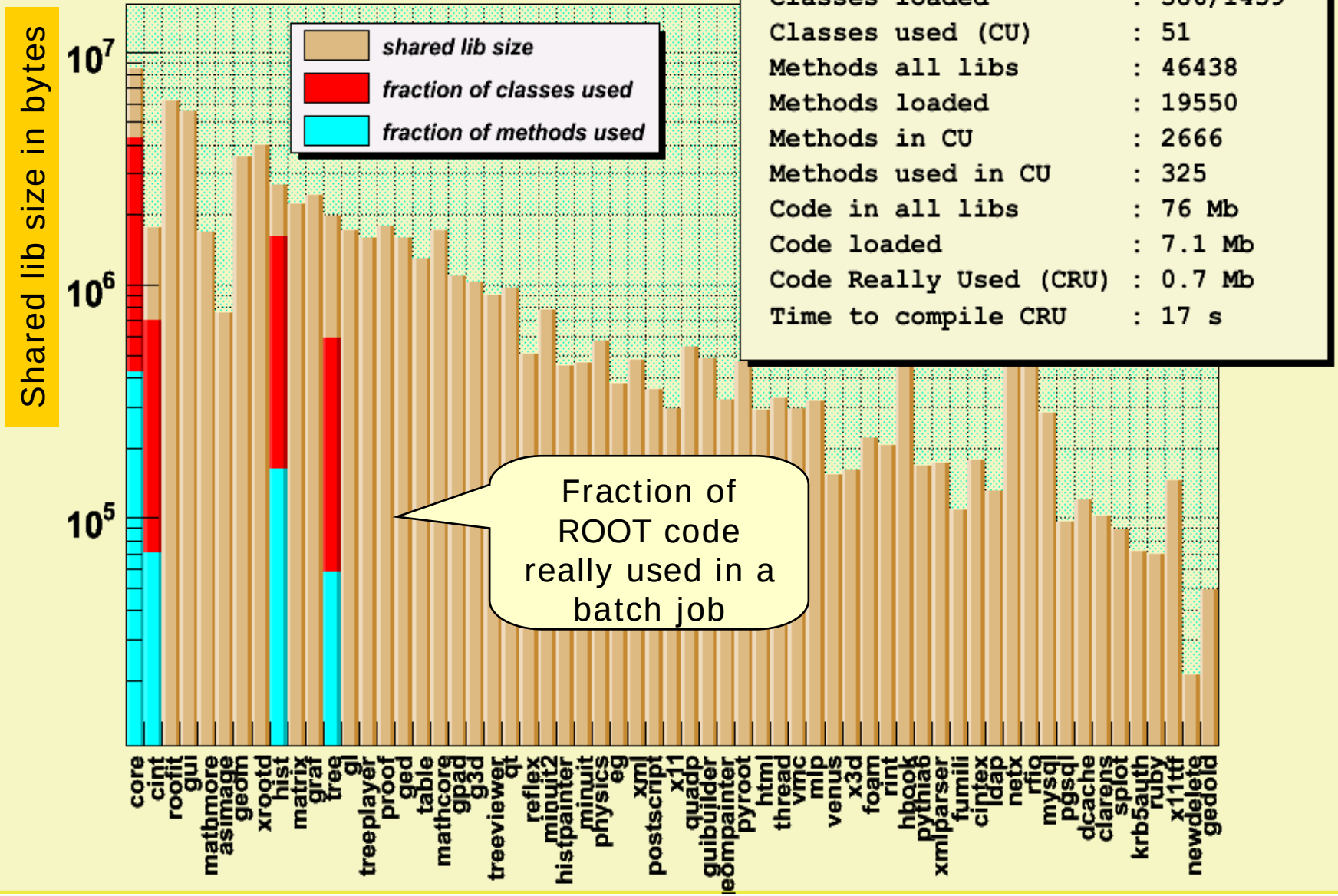
- Shared libs are essential for today large applications.
- They optimize the development time if inter-library dependencies is correctly managed.
- The plug-in manager is an essential component that minimizes the number of libraries linked at the start of an application.
- However, a large number of libs may be a killer, in particular for interactive applications.
- Because of large compilation times, most experiments export pre-compiled shared libs.
- These libs are compiled for maximum portability and do not always use efficiently local processors capabilities.

Exported Symbols

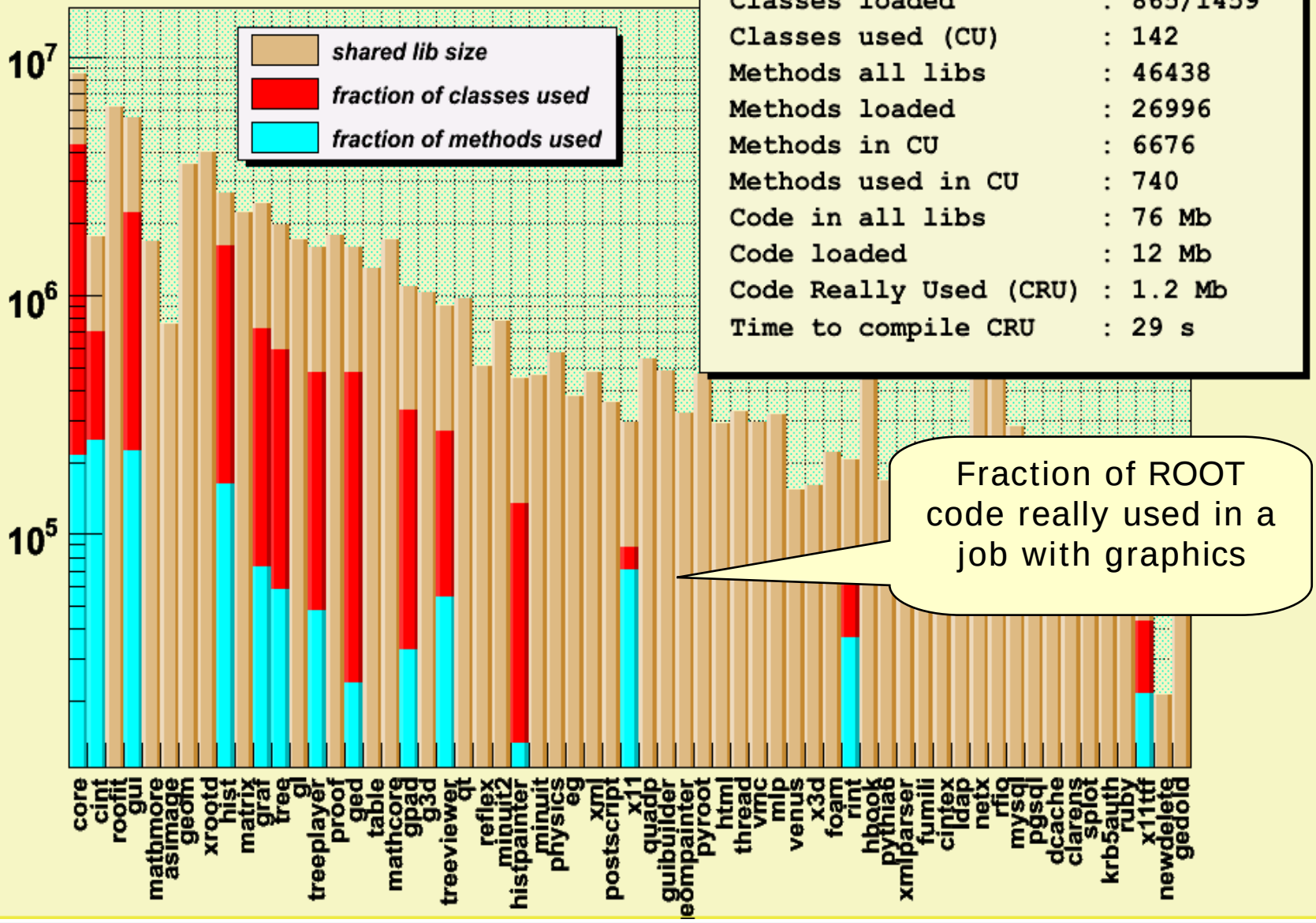
- Time to load a shared lib is grosso modo
- time = size * n * log(N)
  - size = shared lib size in bytes (mapped I/O)
  - n = number of exported symbols in lib
  - N = number of existing exported symbols in previously loaded shared libs

- A good compromise must be found between the number of libraries and their size (modularity vs performance)
- GCC4 & Windows allow selection of symbols accessible from outside shared lib (“exported”) .
- Currently most applications export all C++ symbols !

code used in a batch use case



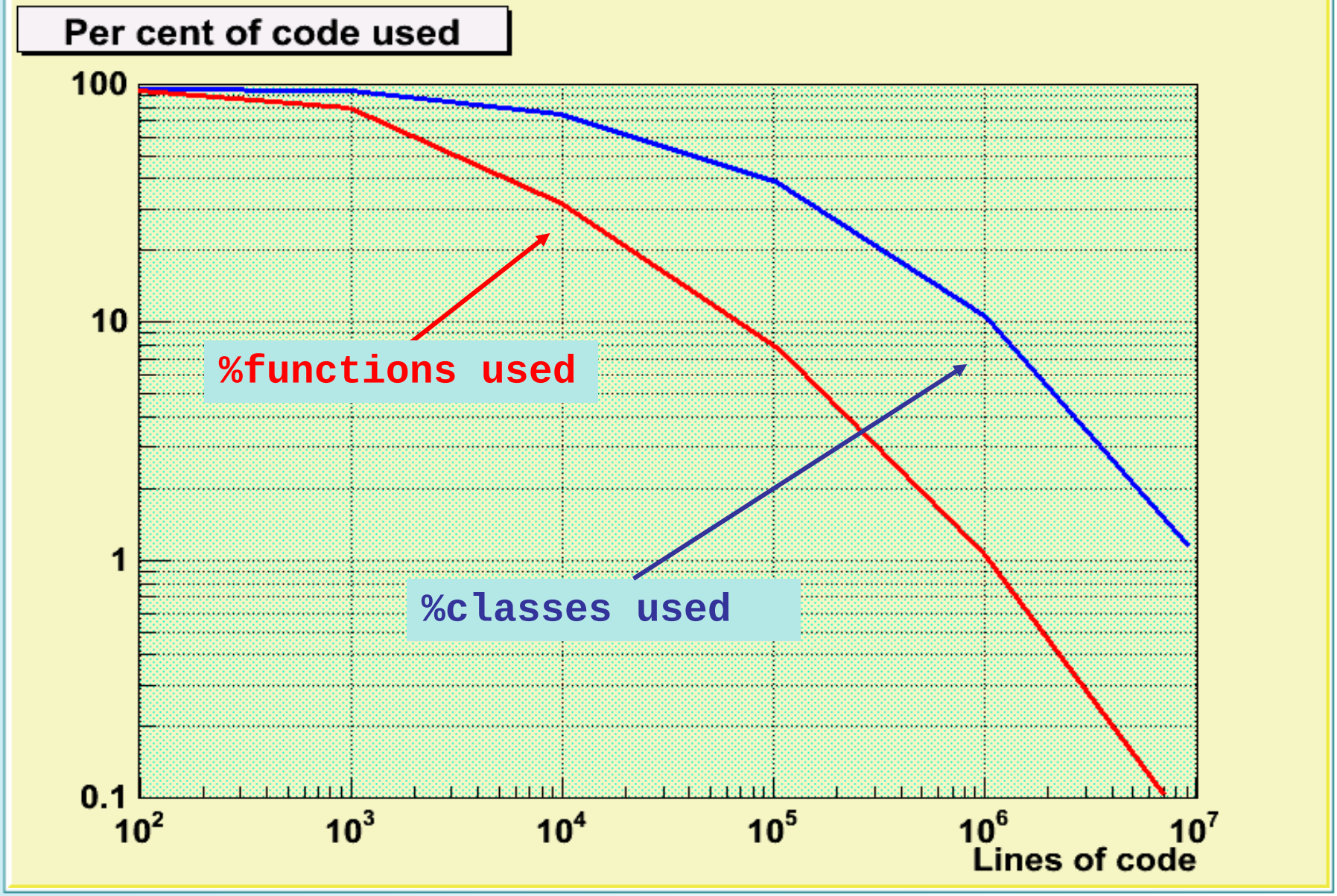
code used in a graphics use case

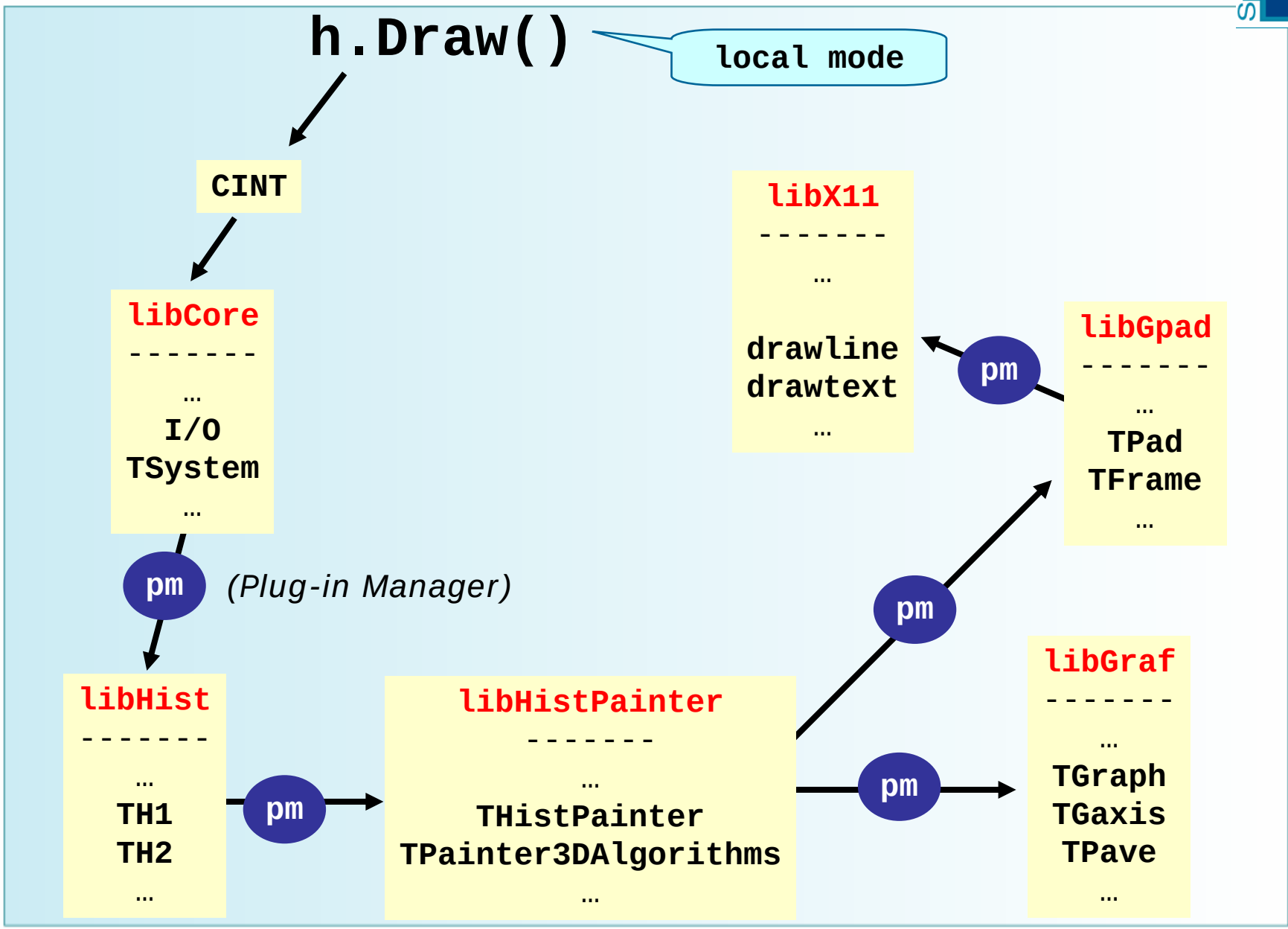


Libs used	: 14/86
Classes loaded	: 865/1459
Classes used (CU)	: 142
Methods all libs	: 46438
Methods loaded	: 26996
Methods in CU	: 6676
Methods used in CU	: 740
Code in all libs	: 76 Mb
Code loaded	: 12 Mb
Code Really Used (CRU)	: 1.2 Mb
Time to compile CRU	: 29 s

Fraction of ROOT code really used in a job with graphics

Fraction of code really used in one program




h.Draw()
local mode
CINT
libCore
I/O
TSystem
pm
(Plug-in Manager)
libHist
TH1
TH2
pm
libHistPainter
THistPainter
TPainter3DAlgorithms
pm
libX11
drawline
drawtext
pm
libGpad
TPad
TFrame
pm
libGraf
TGraph
TGaxis
TPave

Improving execution time

Use profiling tools

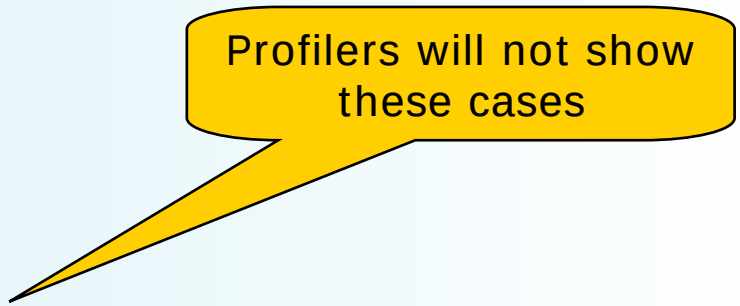
Run your application on many systems

Use the best collection class for your problem

Finding hot spots

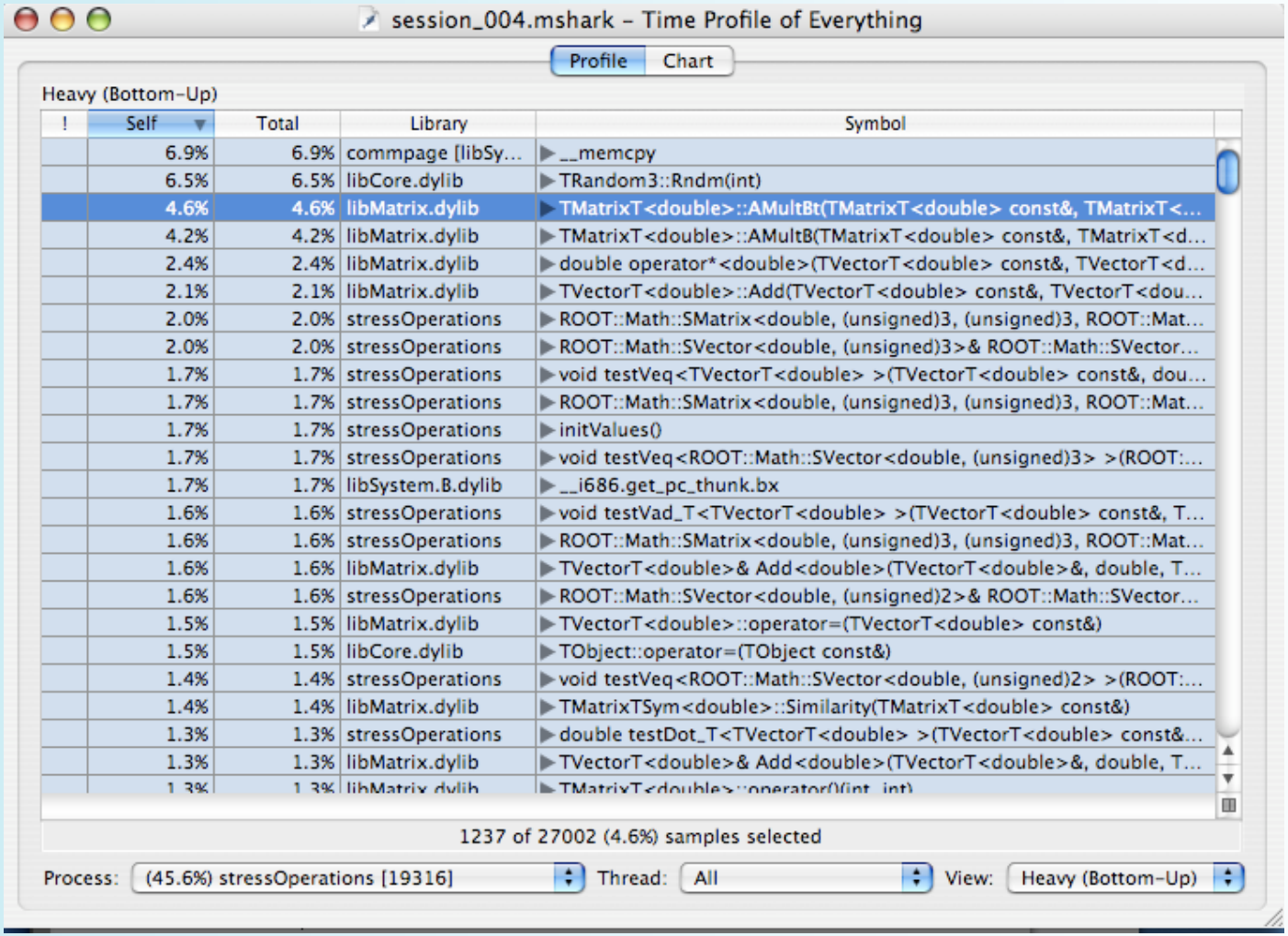
- The main source of inefficiency is bad coding,legacy algorithms (fortran like),bad use (or ignorance) of collection classes, etc.. eg

```
for (int i=0;i<nDetectors;i++) {
   Detector *det = ListOfDetectors->get(i);
   if (det->name() == myDetector->name()) {
      ... some code
   }
}
```

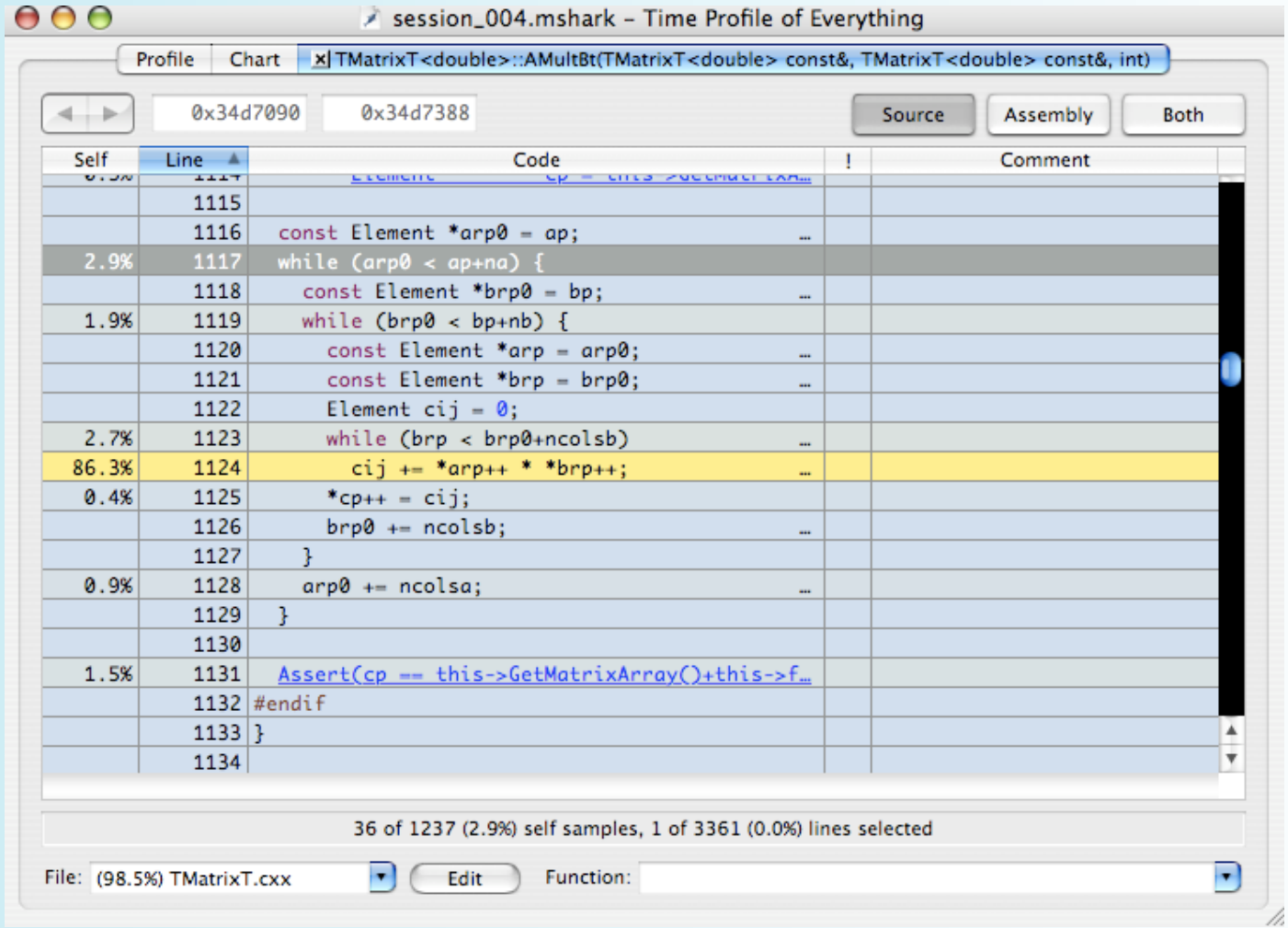



```
for (int i=0;i<N;i++) {
   for (int j=0;j<M;j++) {
      if (myArray[j] == 0) continue;
      // do some work
   }
}
```

Profiling tools: eg Shark on Macosx



Profiling tools: eg Shark on Macosx (2)



Example with smatrix

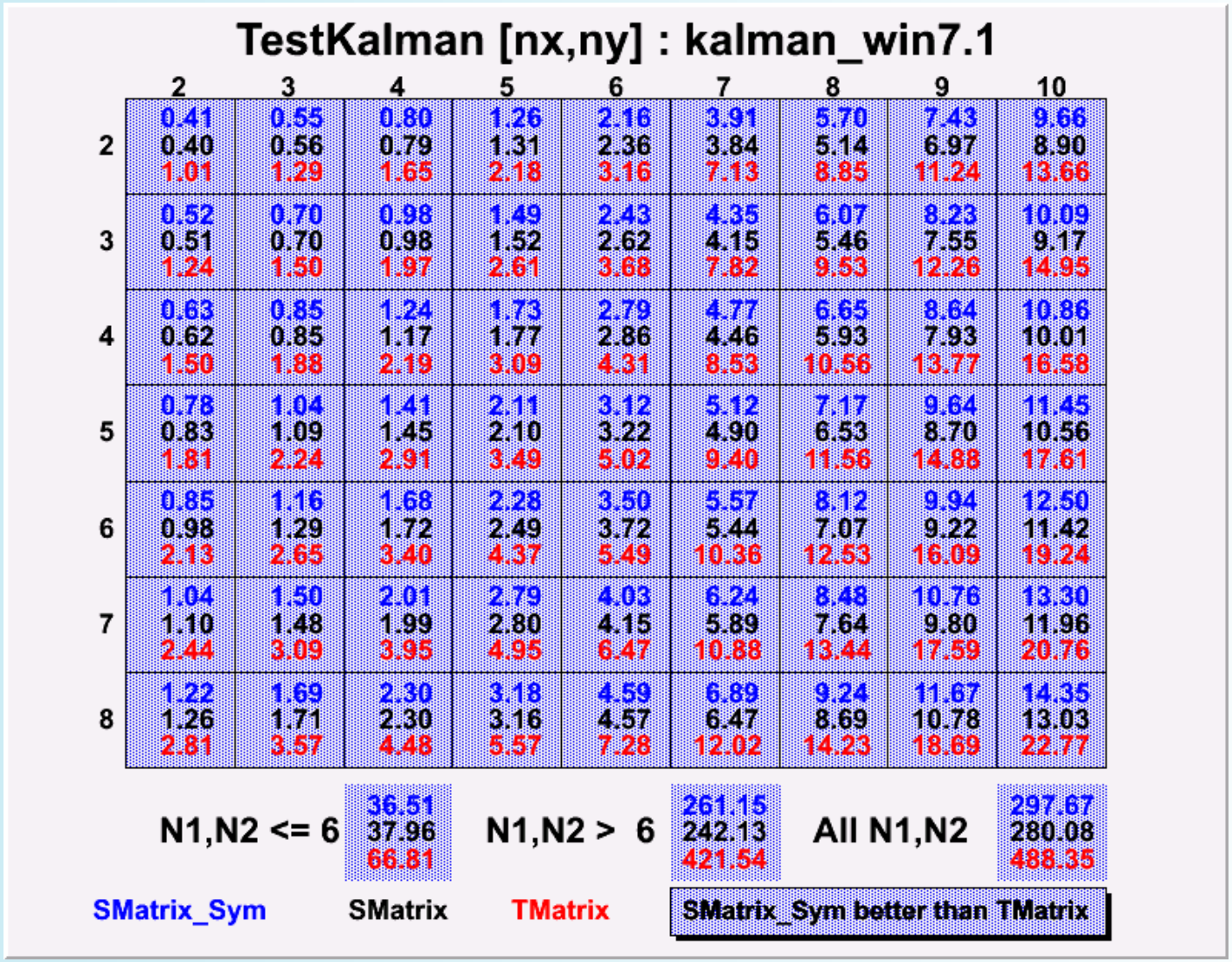
TestKalman [nx,ny] : kalman_slc3_gcc323

	2	3	4	5	6	7	8	9	10
2	0.30 0.33 0.86	0.38 0.44 1.00	0.56 0.64 1.39	0.88 0.98 1.69	1.54 1.64 2.96	5.77 5.84 5.22	8.00 7.81 6.06	10.41 10.23 7.41	14.36 14.58 9.03
3	0.37 0.46 1.01	0.56 0.60 1.16	0.72 0.99 1.59	1.10 1.29 1.96	1.84 2.03 3.40	6.24 6.33 5.48	8.19 8.17 6.64	10.97 10.76 7.61	14.37 14.02 9.86
4	0.47 0.61 1.16	0.63 0.76 1.42	0.89 1.03 1.72	1.39 1.48 2.48	2.16 2.27 3.67	6.71 6.43 6.14	9.04 8.92 6.95	11.71 11.37 8.85	15.07 14.69 10.33
5	0.60 0.78 1.28	0.85 1.03 1.55	1.19 1.28 2.35	1.71 1.80 2.69	2.58 2.70 4.44	7.03 6.79 6.60	9.52 9.18 8.34	12.41 12.03 9.44	15.74 16.00 12.16
6	0.77 0.96 1.59	1.26 1.22 2.09	1.49 1.60 2.42	2.13 2.17 3.58	3.06 3.12 4.61	7.81 7.39 7.55	10.19 9.90 8.09	12.98 12.53 10.36	17.56 16.92 11.58
7	0.96 1.25 1.75	1.33 1.49 2.17	1.77 1.99 3.03	2.46 2.57 3.53	3.47 3.62 5.48	8.24 8.08 7.90	10.56 10.13 9.50	13.08 12.72 10.62	18.03 16.96 14.14
8	1.14 1.48 2.05	1.68 1.79 2.81	2.15 2.33 3.02	2.95 3.14 4.67	4.07 4.27 5.59	8.99 8.79 8.69	11.47 11.37 9.34	14.48 14.36 12.29	19.15 18.07 13.71

N1,N2 <= 6		N1,N2 > 6		All N1,N2	
29.45	SMatrix_Sym	340.08	SMatrix_Sym	369.53	SMatrix_Sym
32.23	SMatrix	334.29	SMatrix	366.52	SMatrix
54.07	TMatrix	283.99	TMatrix	338.05	TMatrix

SMatrix_Sym better than TMatrix

Example with smatrix (2)



	2	3	4	5	6	7	8	9	10
2	0.41 0.40 1.01	0.55 0.56 1.29	0.80 0.79 1.65	1.26 1.31 2.18	2.16 2.36 3.16	3.91 3.84 7.13	5.70 5.14 8.85	7.43 6.97 11.24	9.66 8.90 13.66
3	0.52 0.51 1.24	0.70 0.70 1.50	0.98 0.98 1.97	1.49 1.52 2.61	2.43 2.62 3.68	4.35 4.15 7.82	6.07 5.46 9.53	8.23 7.55 12.26	10.09 9.17 14.95
4	0.63 0.62 1.50	0.85 0.85 1.88	1.24 1.17 2.19	1.73 1.77 3.09	2.79 2.86 4.31	4.77 4.46 8.53	6.65 5.93 10.56	8.64 7.93 13.77	10.86 10.01 16.58
5	0.78 0.83 1.81	1.04 1.09 2.24	1.41 1.45 2.91	2.11 2.10 3.49	3.12 3.22 5.02	5.12 4.90 9.40	7.17 6.53 11.56	9.64 8.70 14.88	11.45 10.56 17.61
6	0.85 0.98 2.13	1.16 1.29 2.65	1.68 1.72 3.40	2.28 2.49 4.37	3.50 3.72 5.49	5.57 5.44 10.36	8.12 7.07 12.53	9.94 9.22 16.09	12.50 11.42 19.24
7	1.04 1.10 2.44	1.50 1.48 3.09	2.01 1.99 3.95	2.79 2.80 4.95	4.03 4.15 6.47	6.24 5.89 10.88	8.48 7.64 13.44	10.76 9.80 17.59	13.30 11.96 20.76
8	1.22 1.26 2.81	1.69 1.71 3.57	2.30 2.30 4.48	3.18 3.16 5.57	4.59 4.57 7.28	6.89 6.47 12.02	9.24 8.69 14.23	11.67 10.78 18.69	14.35 13.03 22.77

N1,N2 <= 6	36.51 37.96 66.81	N1,N2 > 6	261.15 242.13 421.54	All N1,N2	297.67 280.08 488.35

SMatrix_Sym SMatrix TMatrix SMatrix_Sym better than TMatrix

Example with smatrix (3)

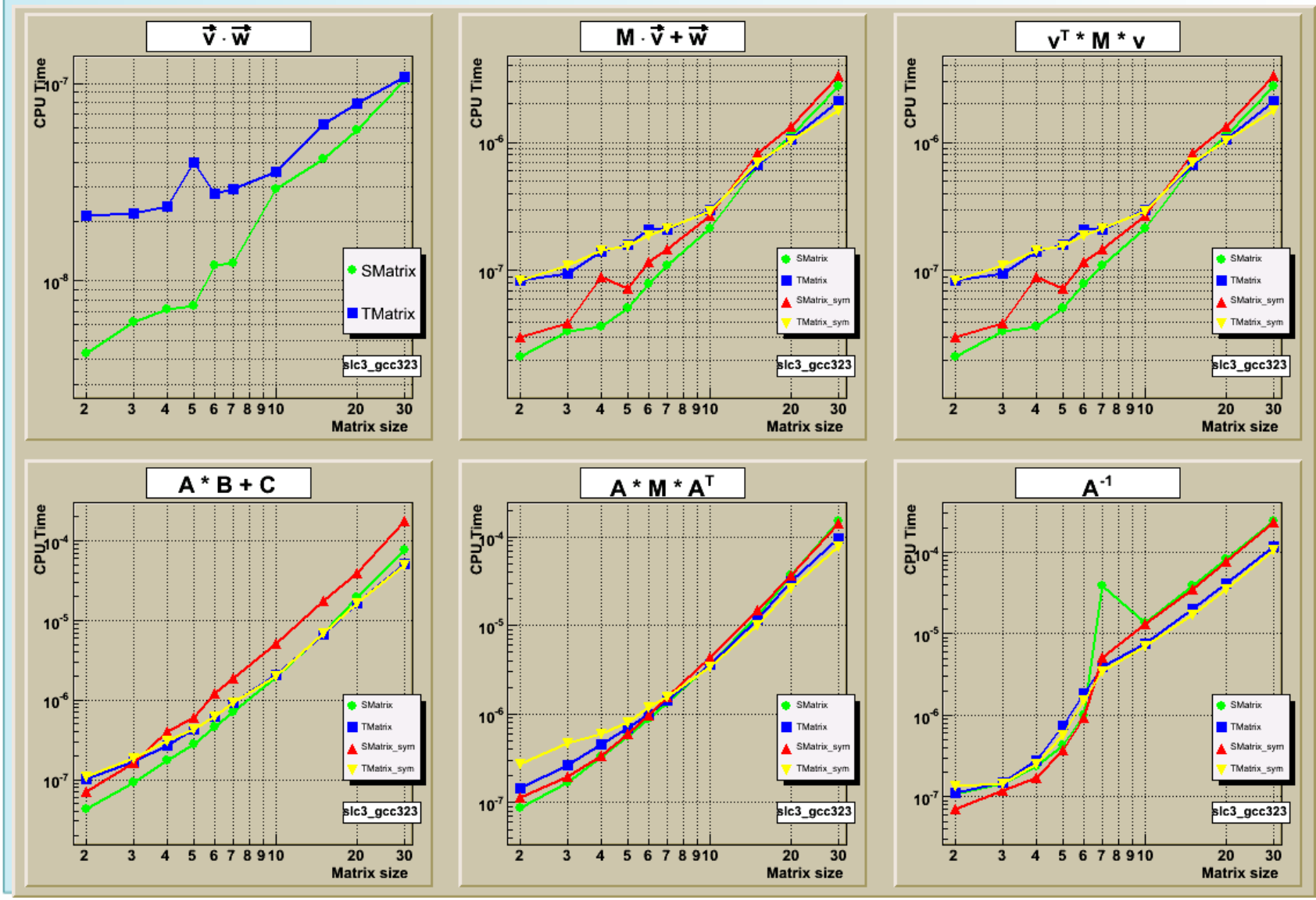
TestKalman [nx,ny] : kalman_solaris.5.9

	2	3	4	5	6	7	8	9	10
2	2.29 1.39 2.49	4.53 2.41 2.95	7.49 3.52 3.84	13.36 5.57 5.15	27.92 9.88 8.18	30.68 20.13 34.74	42.12 29.90 42.52	56.27 41.89 51.08	74.79 56.08 61.38
3	3.36 2.05 2.83	6.28 3.43 3.49	9.88 5.09 4.74	17.60 7.79 6.23	33.32 12.81 9.61	37.58 24.26 36.25	51.27 35.10 44.61	69.29 50.11 53.69	88.88 66.09 63.78
4	4.70 2.92 3.50	8.39 4.82 4.41	13.02 7.16 5.68	21.30 10.45 7.46	38.09 16.27 11.42	44.86 28.23 38.35	62.55 43.15 47.23	83.96 60.94 56.35	108.25 79.72 67.32
5	6.45 3.84 3.87	11.09 6.42 5.10	16.75 9.42 6.78	26.01 13.43 8.88	45.35 20.22 12.96	52.92 32.57 40.86	73.96 50.84 49.51	100.57 72.06 59.60	127.69 94.24 70.80
6	8.77 5.36 4.58	14.55 8.67 6.12	21.27 12.45 8.33	32.27 17.49 10.55	51.35 25.37 14.90	63.23 39.55 43.39	87.57 60.54 52.76	118.44 84.29 63.18	152.52 112.31 75.01
7	12.58 6.85 5.27	20.21 10.88 7.13	29.16 15.45 9.41	42.12 21.34 12.36	64.82 29.96 17.47	78.81 44.96 45.91	107.49 68.27 55.53	142.03 96.24 66.99	183.05 128.52 79.38
8	17.68 10.79 6.08	28.33 17.19 8.30	40.40 24.55 10.98	57.23 33.55 14.26	84.57 46.08 19.58	103.45 64.54 48.60	139.67 95.46 58.98	184.39 132.12 70.57	232.32 170.91 83.94

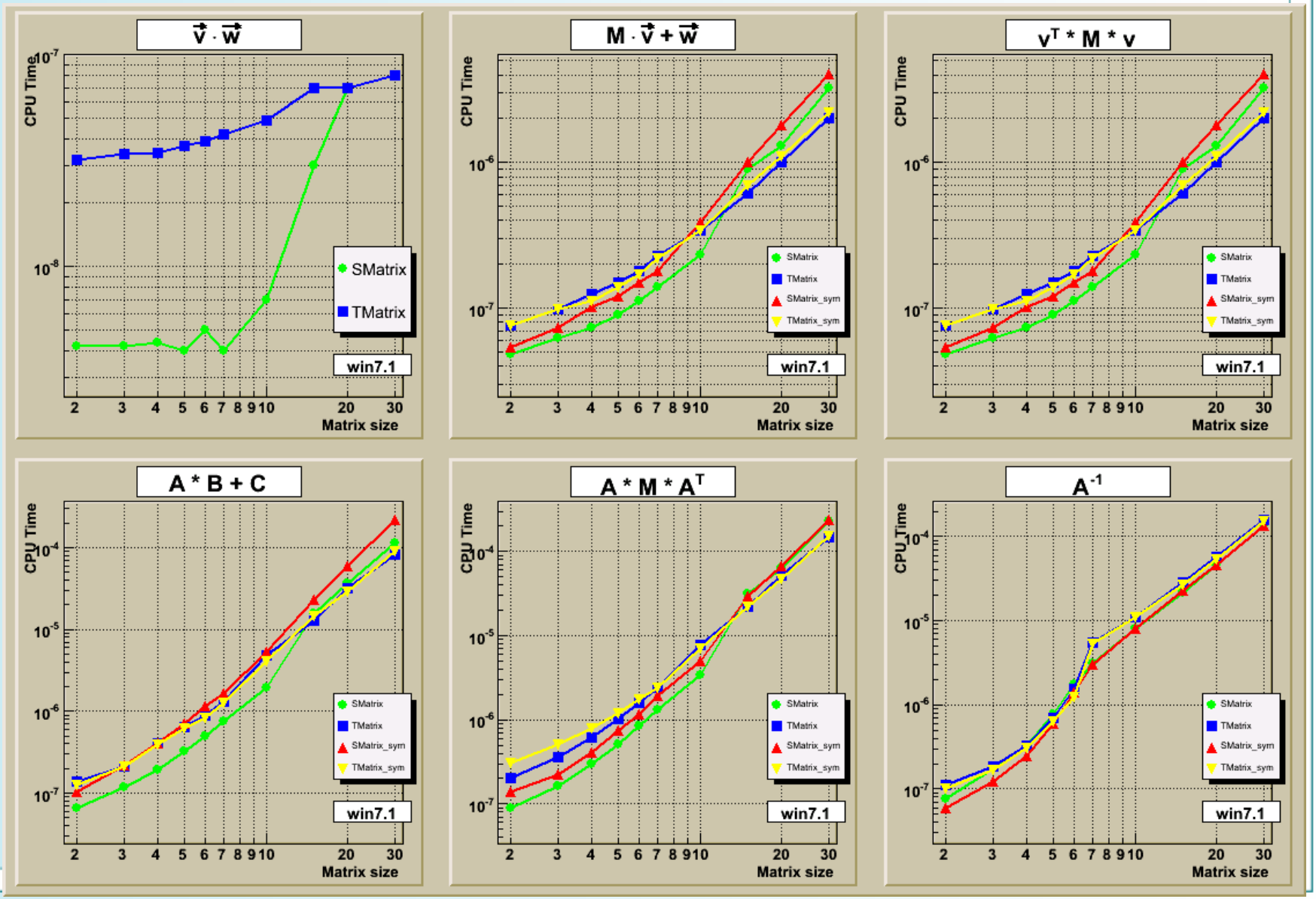
N1,N2 <= 6	445.38 218.23 164.08	N1,N2 > 6	3095.72 2099.65 1673.16	All N1,N2	3541.10 2317.89 1837.24

SMatrix_Sym SMatrix TMatrix SMatrix_Sym better than TMatrix

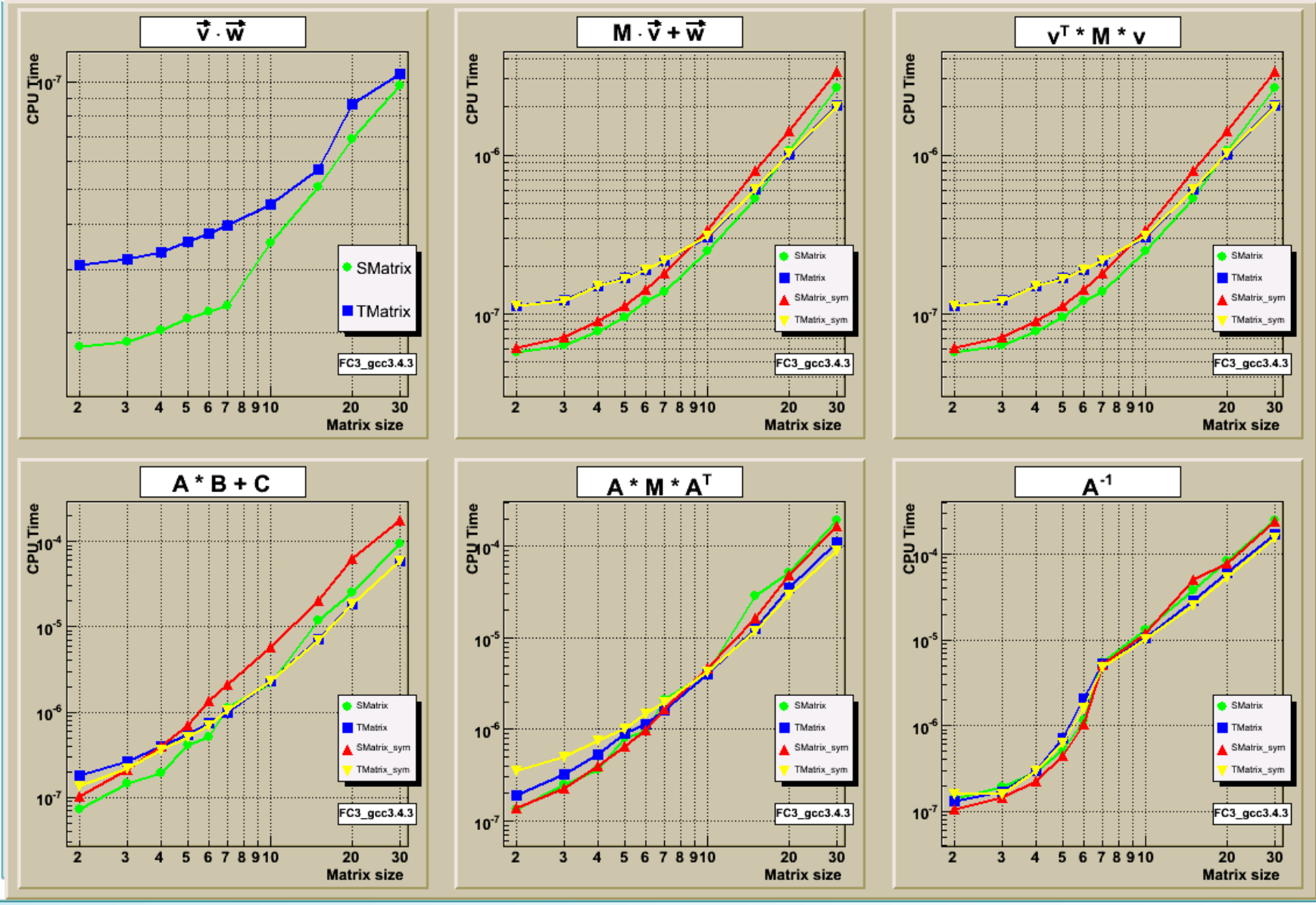
Example with smatrix (gcc3.2.3/slc3)



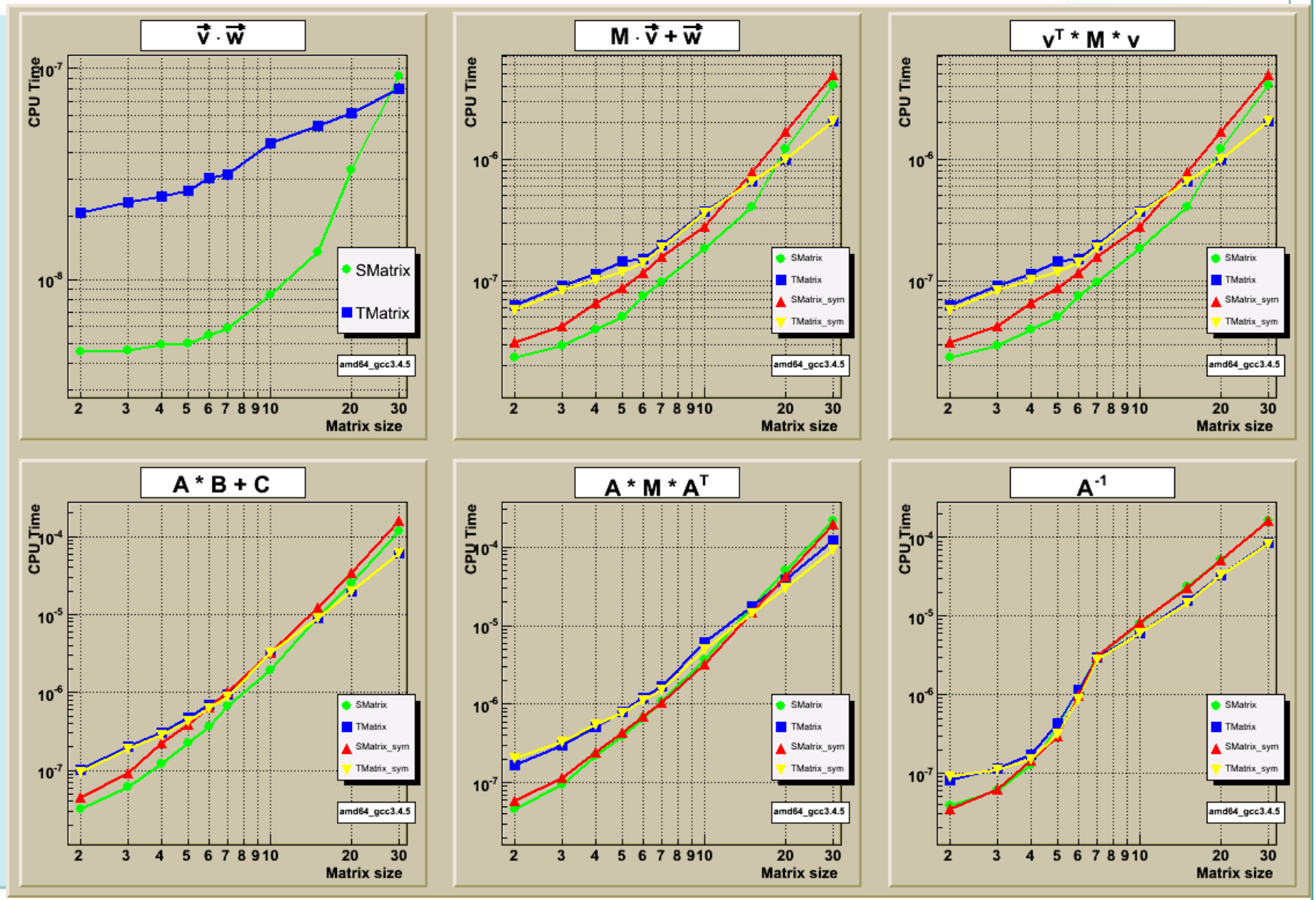
Example with smatrix (vc++7.1/Windows)



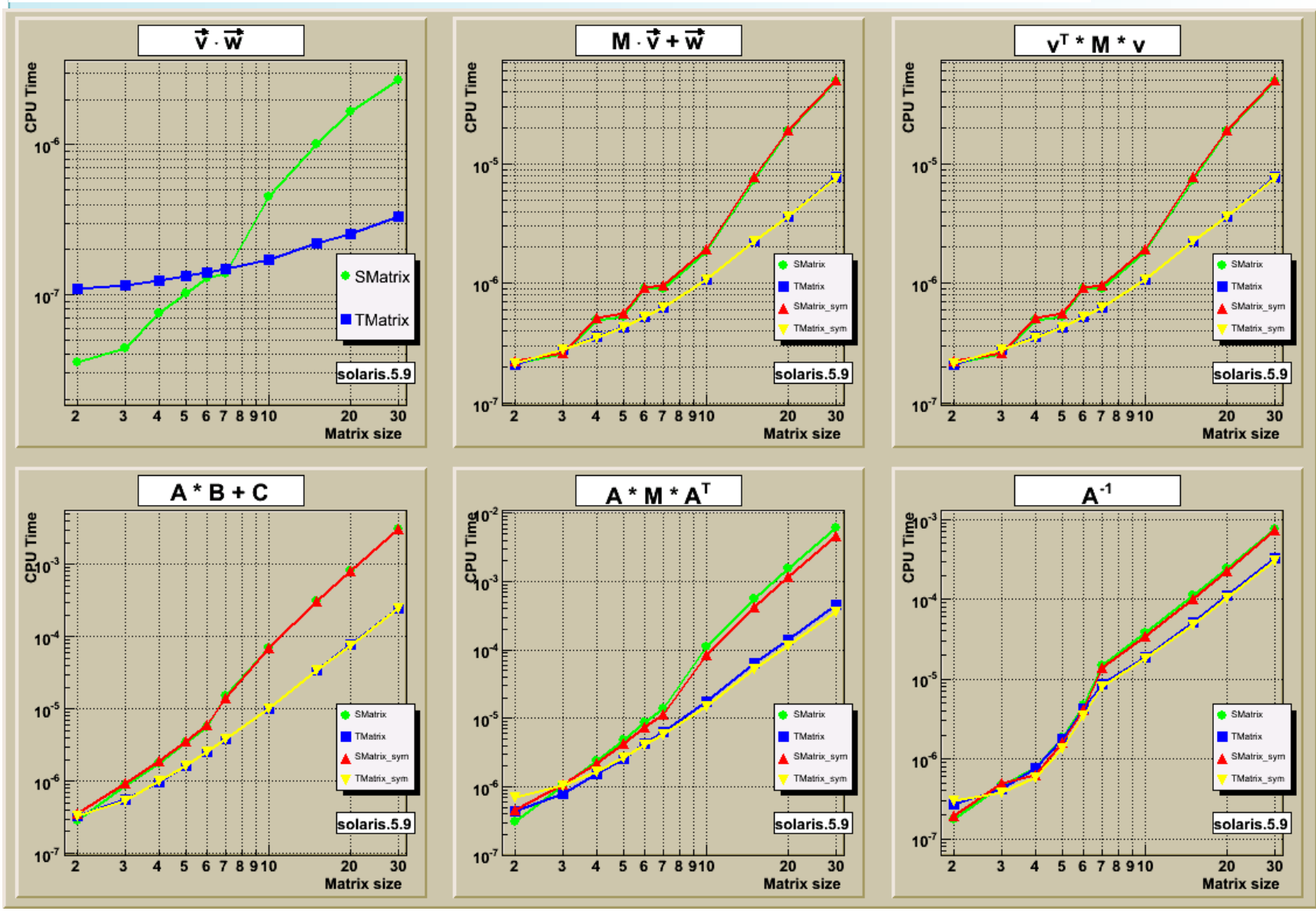
Example with smatrix (gcc3.4.5 Fedora3)



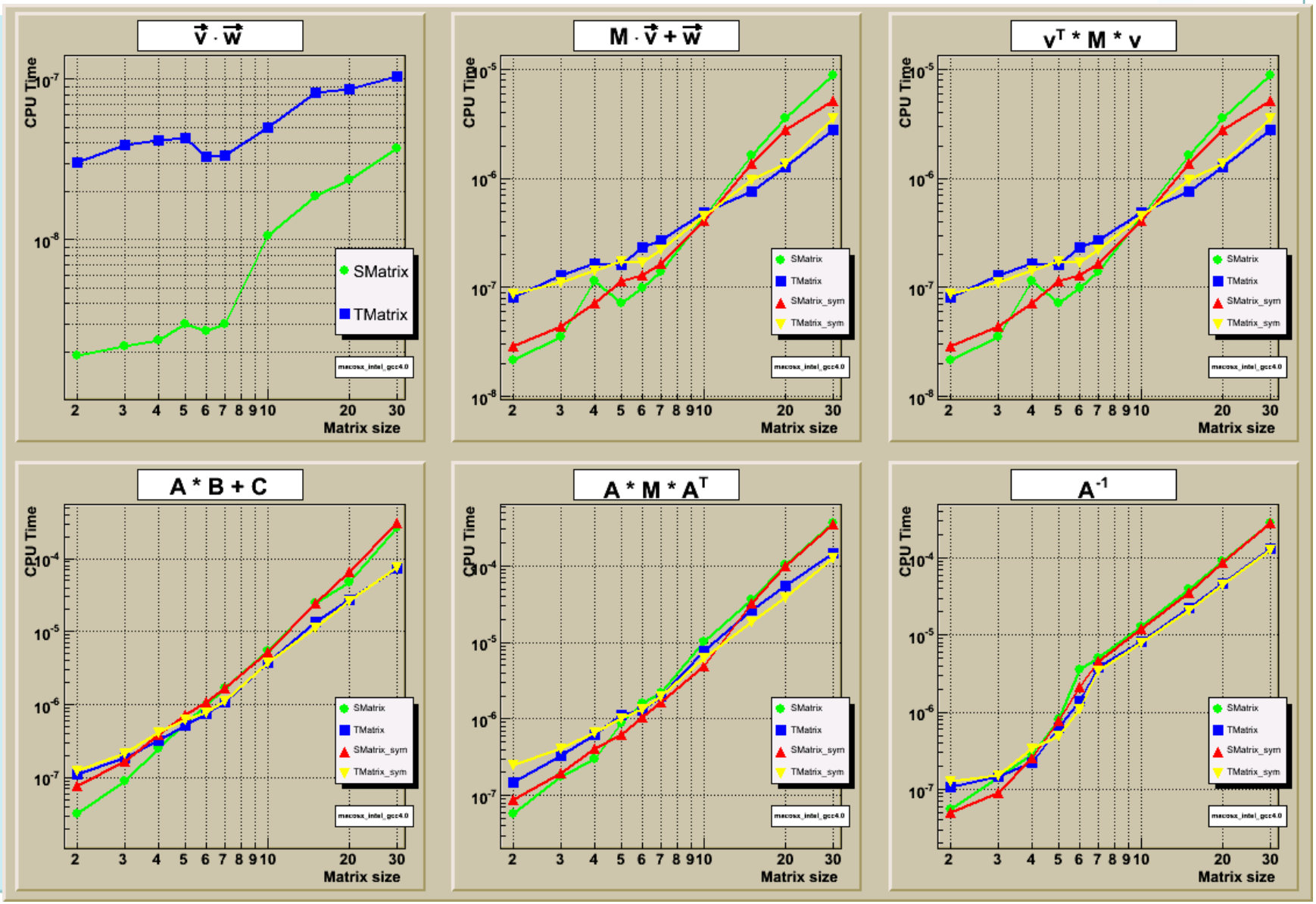
Example with smatrix (gcc3.4.5/amd64)



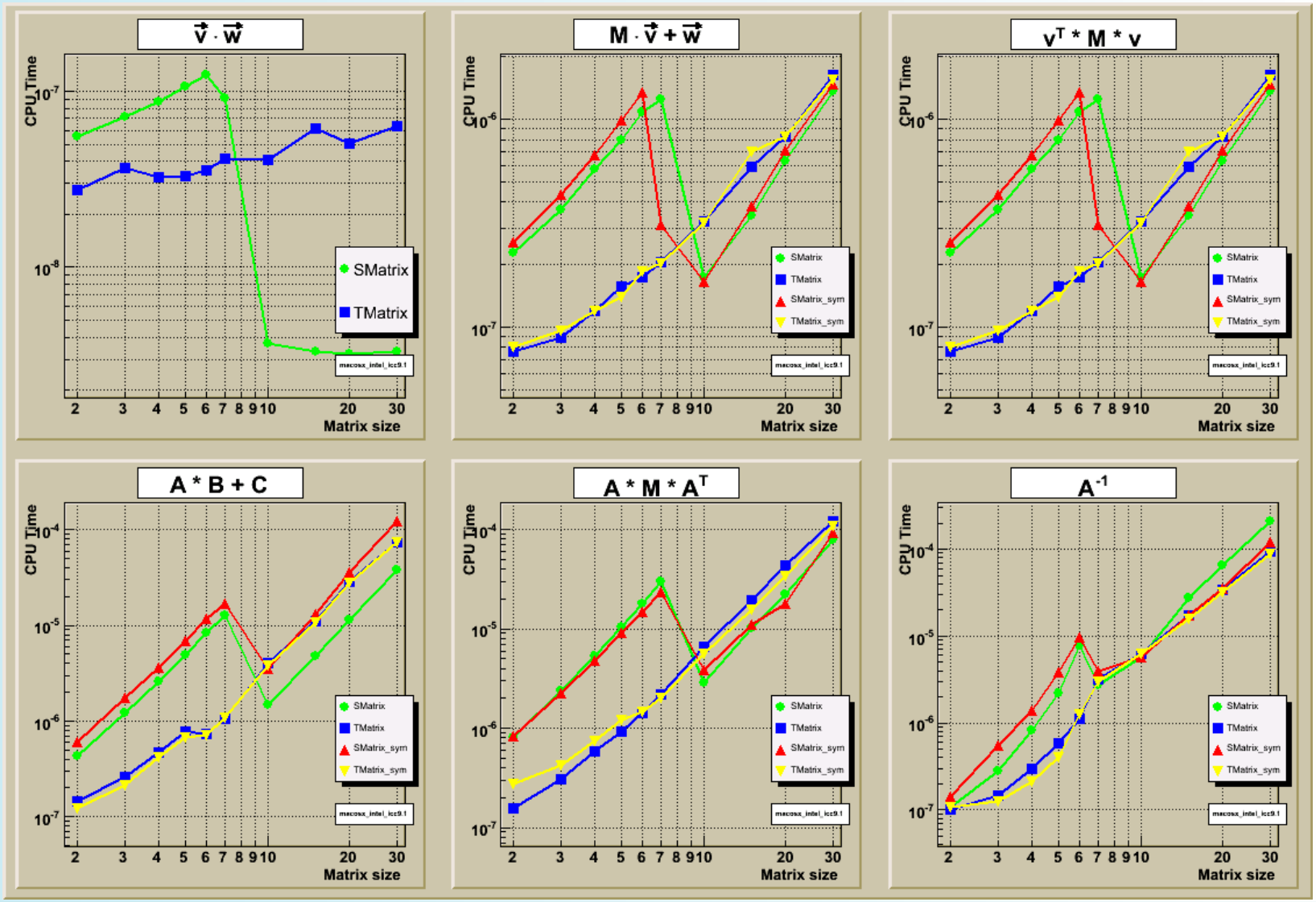
Example with smatrix (CC/solaris5.9)



Example with smatrix (gcc4.0.1/MacIntel)



Example with smatrix (icc9.1/MacIntel)



Choose the right collection classes

Type of collection	Fill memory	Write to disk	Read from disk	File size
TClonesArray(TObjHit)	1	1	1	1
vector<THit>	1.10	1.71	1.19	1.00
list<THit>	1.33	1.68	1.96	1.00
deque<THit>	1.27	1.51	1.53	1.00
set<THit>	1.47	1.57	2.46	1.00
multiset<THit>	1.41	1.65	2.42	1.00
map<int,THit>	1.70	1.68	2.61	1.04
multimap<int,THit>	1.66	1.68	2.54	1.04
vector<int,THit*>	1.04	2.71	3.77	1.14
list<THit*>	1.29	3.03	4.50	1.14
deque<THit*>	1.11	2.88	3.96	1.14
set<THit*>	1.41	3.14	4.77	1.14
multiset<THit*>	1.39	3.08	4.65	1.14
map<int,THit*>	1.39	3.37	4.92	1.18
multimap<int,THit*>	1.39	3.31	4.81	1.18

Different compilers , OS, machines

Benchmark (rootmarks)	centrino 1.5 Ghz Windows Vc++7.1	P IV 3 Ghz SLC3 gcc3.2.3	MacBook 2 Ghz Tiger gcc4.0.1	MacBook 2 Ghz Tiger icc9.1
stress –b 30	809	766	865	970
stress –b 1000	817	706	839	972
stressKalman	1301,1415 1490	942,1087 2212	960,848 1861	277,368 1798
stressLinear	534	807	707	849
stressGeom	681	734	889	1158
stressSpectrum	553	821	1295	1576
bench	849	881	901	989

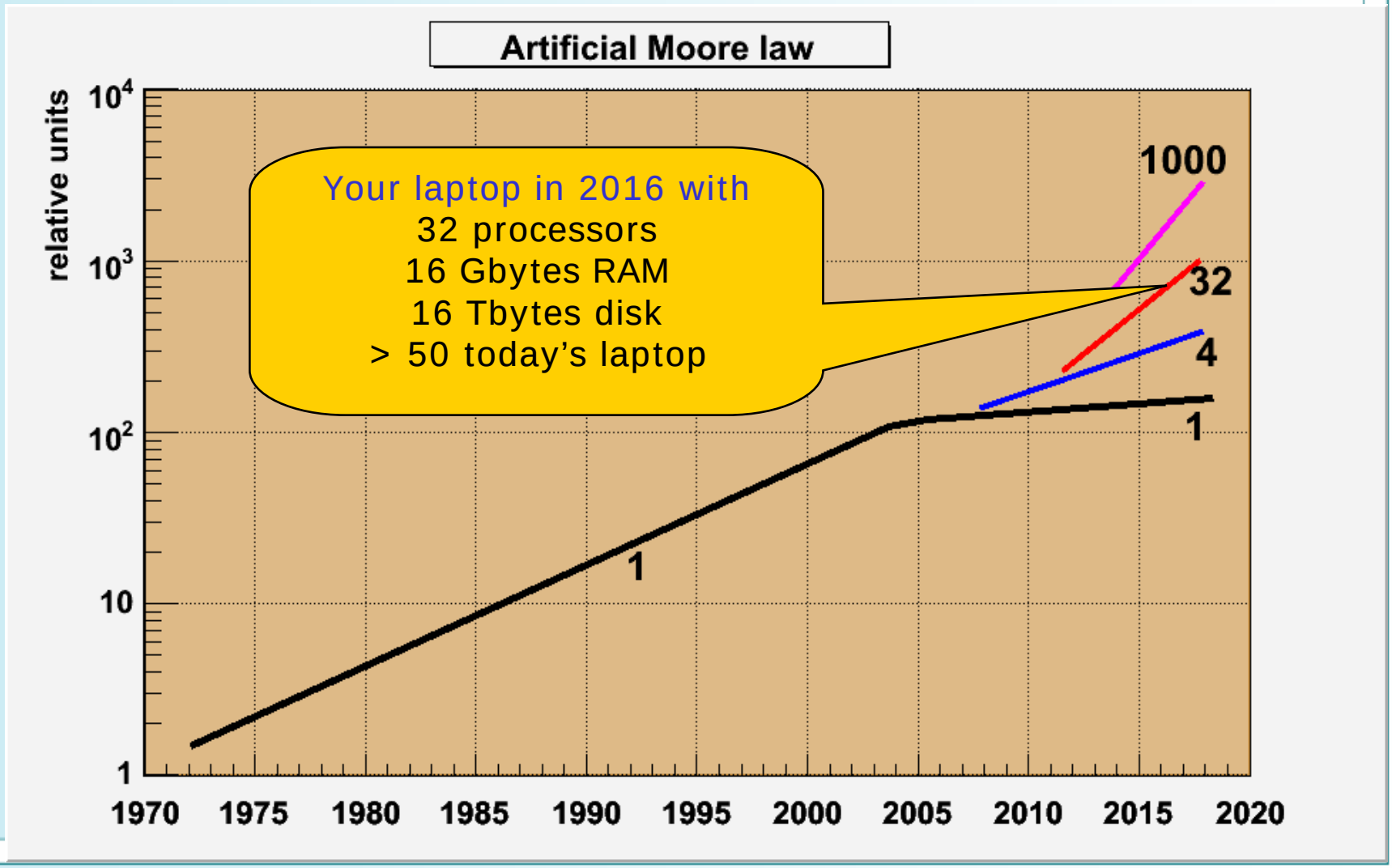
Multithreading

A requirement for multi-core CPUs

Making a program thread-safe is non trivial

Making a program thread-aware is complex

Moore's law revisited



Example: Impact on ROOT

- There are many areas in ROOT that can benefit from a multi core architecture. Because the hardware is becoming available on commodity laptops, it is urgent to implement the most obvious asap.
- Multi-Core often implies multi-threading. There are several areas to be made not only thread-safe but also thread aware.
  - PROOF obvious candidate. By default a ROOT interactive session should run in PROOF mode. It would be nice if this could be made totally transparent to a user.
  - Speed-up I/O with multi-threaded I/O and read-ahead
  - Buffer compression in parallel
  - Minimization function in parallel
  - Interactive compilation with ACLIC in parallel
  - etc..

Example: Impact on user applications

- Probably limited for simulation and reconstruction
- Huge impact for data analysis. Analysis code must take advantage of multi-core CPUs (PROOF selectors or like)

Summary

- Compilation time becoming a serious problem for large applications using inlined classes.
- Precompiled headers facility becoming available.
- Reduce application startup time by a careful organization of shared libs and minimization of exported symbols
- Be careful with templated code. Performance may vary a lot with the application size, compilers and OS.
- Use profiling tools as much as possible. Run your application on as many platforms as possible. This is good for performance and also for improving the quality of the code.