Evaluation of x32-ABI in the context of CERN applications

Nathalie Rauschmayr

Computing group of LHCb

October 24, 2012

Outline

Outline

x32-ABI - A new 32-bit ABI for x86-64

- Application Binary Interface based on 64-bit x86 architecture
- Uses 32-bit pointers instead of 64-bit
- Takes advantage of many x64-features
  - Larger number of CPU registers
  - Better floating-point performance
  - Faster position-independent code shared libraries
  - Function parameters passed via registers
  - Faster syscall instruction
  - ...
- Avoids overhead of 64-bit pointers

x32-ABI - A new 32-bit ABI for x86-64

- Developed by H.J. Lu (Intel)
- Introduced in Linux-Kernel 3.4 (released in summer 2012)
- http://www.linuxplumbersconf.org/2011/ocw/sessions/531
- Opinions in Linux-community differ quite a lot
  - 32-bit time values will overflow
  - adressable memory
  - ...

Outline

How CERN applications can profit

- Since the change from 32-bit to 64-bit LHCb application increased memory consumption by factor ≈ 1.6

  Brunel-application 700 MB $\rightarrow$ 1.2 GB

- Application uses millions of pointers
  - Transient-Event-Store: stores all events as pointers
  - Many virtual functions (virtual tables: function pointers and hidden pointers)
  - ROOT (histogram as a tree of pointers ...)
  - ...

How CERN applications can profit (2)

Impressive results from x32-developers:

181.mcf from SPEC CPU 2000 (memory bound):

Intel Core i7

$\sim$ *40% faster than x86-64*

$\sim$ *2% slower than ia32*

Intel Atom

$\sim$ *40% faster than x86-64*

$\sim$ *1% faster than ia32*

How CERN applications can profit (3)

186.crafty from SPEC CPU 2000 (64bit integer):

Intel Core i7

$\sim$ *3 % faster than x86-64*

$\sim$ *40% faster than ia32*

Intel Atom

$\sim$ *4 % faster than x86-64*

$\sim$ *26% faster than ia32*

$\Longrightarrow$ CERN applications will definitly gain in memory and very likely in CPU-time

Outline

Preconditions

x32-ABI requires:

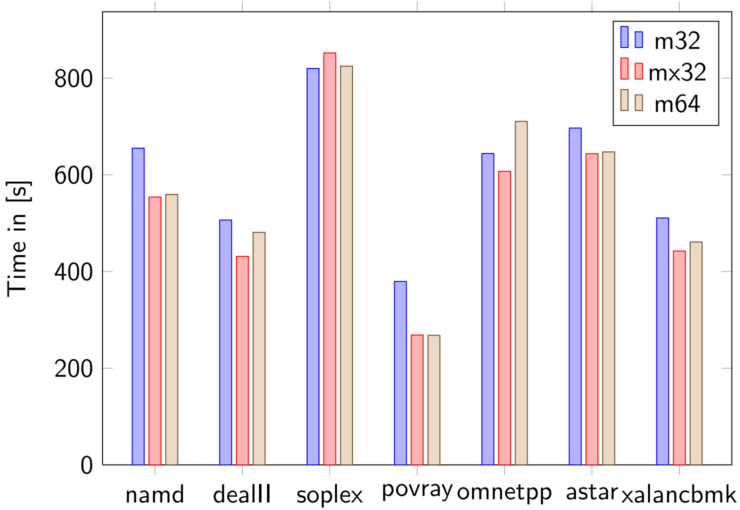
- Linux Kernel 3.4 compiled with CONFIG_X86_X32=y
- Gcc 4.7
- Binutils 2.22
- Glibc 2.16
- Recompiling all system libraries, required by an application, with gcc -mx32

```
ELF 32-bit LSB shared object, x86-64, version 1 (SYSV)
```

Outline

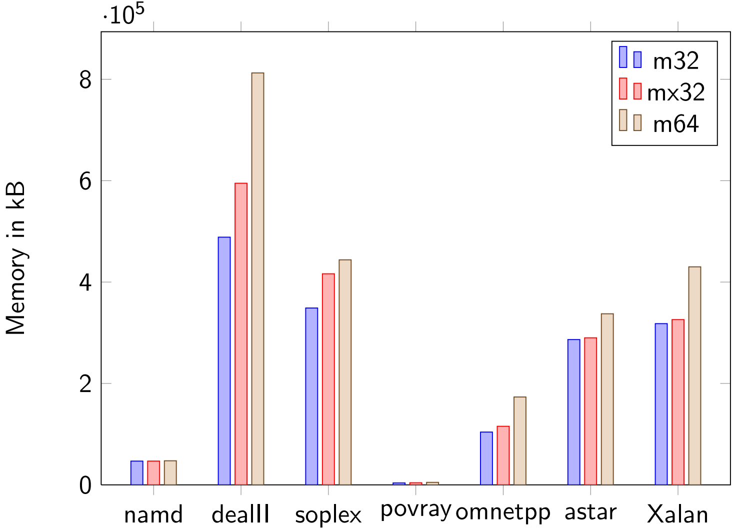
Results within HEPSPEC-benchmarks (SPEC2006)

Comparison of time:



Results within HEPSPEC-benchmarks (SPEC2006) (2)

Comparison of memory consumption:



Results within HEPSPEC-benchmarks (SPEC2006) (3)

Reasons for performance difference:

- code and data alignment
- x32 loop unrolling needs some tuning
- zero-extensions for pointer conversion $\rightarrow$ instruction growth

$\Longrightarrow$ x32-ABI still under development and there are possiblities for optimization

Compared to 32-bit tests:

- better 64-bit integer arithmetic due to 64-registers
- function calls are passed via registers and not via stack memory
- floating point values are returned via SSE registers

Outline

Results within LHCb-Applications

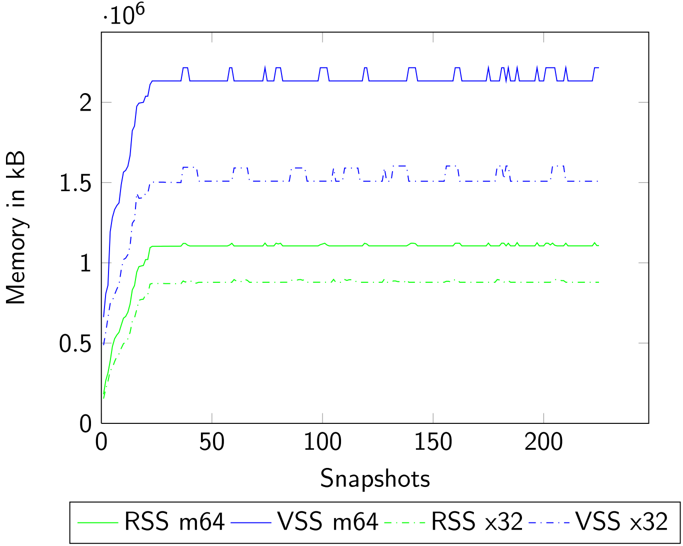
Brunel with Gaudiv23r3 and ROOT 5.34.00

Reconstruction of 1000 Events:

- During main loop: average reduction of 20 % physical and 28 % virtual memory
- User time: 3 % reduction
- System time: 5 % increase
- Total elapsed: 2 % reduction

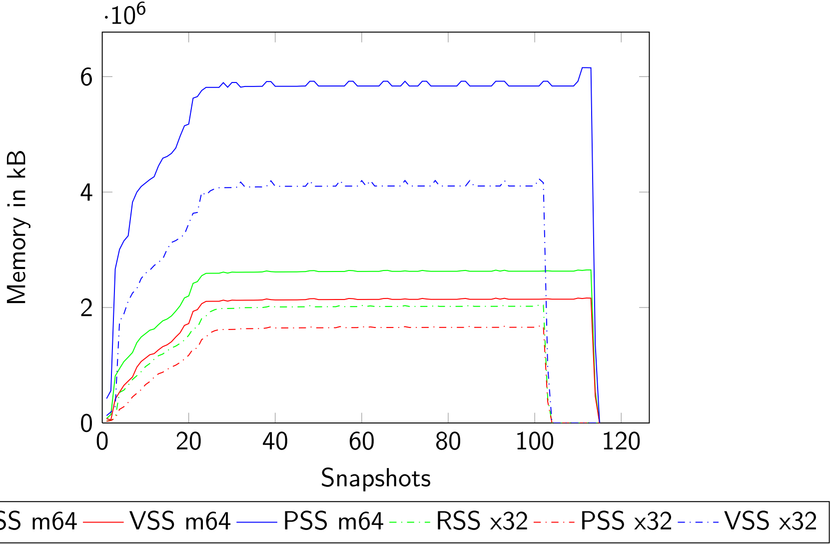
Results within LHCb-Applications (2)

Comparison of memory consumption:



Results within LHCb-Applications (3)

Running Brunel in parallel with two workers:



Results within LHCb-Applications (4)

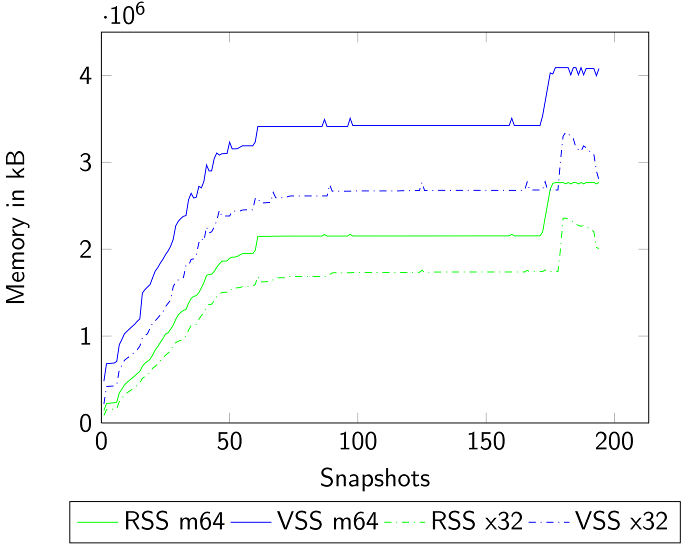
DaVinci with Gaudiv23r3 and ROOT 5.34.00

Analyse of 10000 Events:

- During main loop: average reduction of 21 % physical and 22 % virtual memory
- User time: 1.3 % increase
- System time: 6.7 % increase
- Total elapsed: 1.5 % increase

Results within LHCb-Applications (5)

Comparison of memory consumption:



Outline

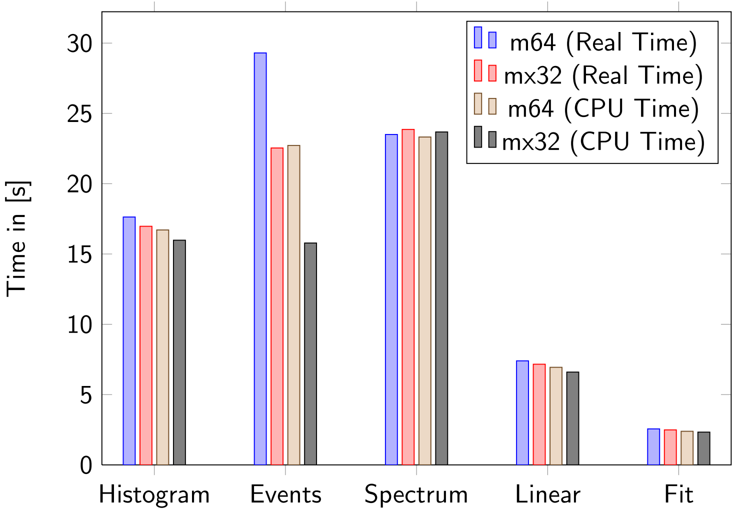
Results within ROOT-Benchmarks

Evaluations has been executed on the following ROOT-benchmarks

- stressHistogram
- stress 1000
- stressHepix
  - IO
  - linear algebra
  - vector
  - sparse matrix

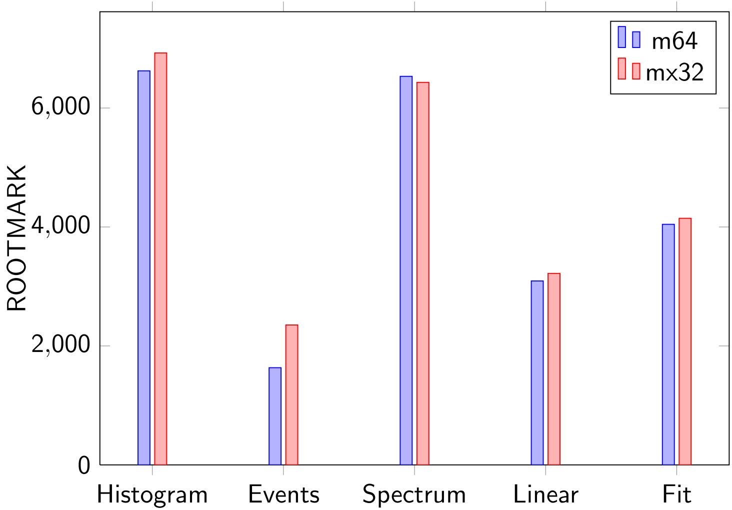
Results within ROOT-Benchmarks

Comparison of time:



Results within ROOT-Benchmarks

Comparison of ROOTMARKS:



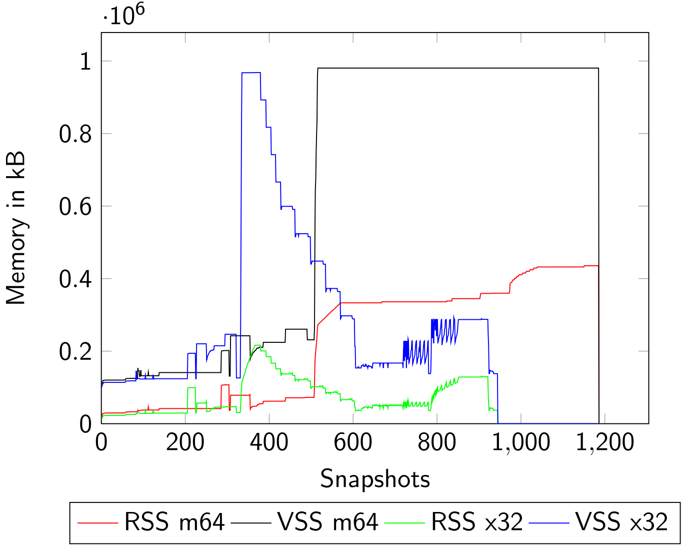
Results within ROOT-Benchmarks

Results in percentage:

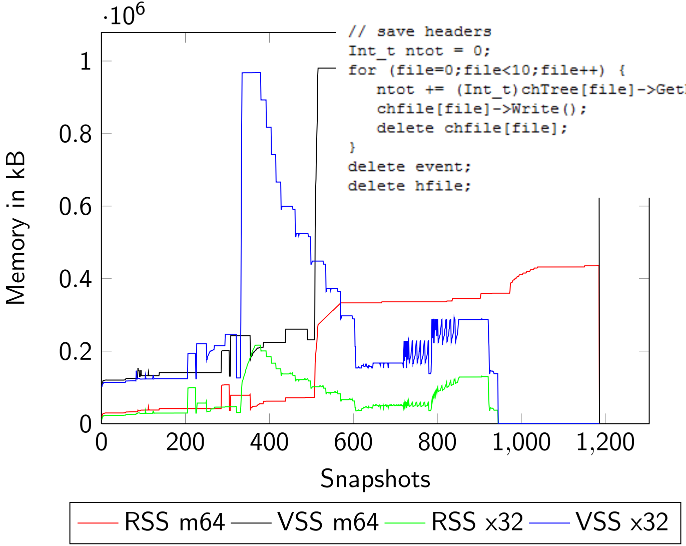
	ROOTMARK	Real Time	CPU Time
stressHistogram	+ 4.5 %	- 3.8 %	- 4.3 %
stress 1000	+ 44 %	- 23.1 %	- 30.5 %
stressSpectrum	- 1.5 %	+ 1.5 %	+ 1.5 %
stressLinear	+ 4.1 %	- 3.2 %	- 3.9 %
stressFit	+ 2.5 %	- 2.8 %	- 2.4 %

Results within ROOT-Benchmarks

stress 1000:



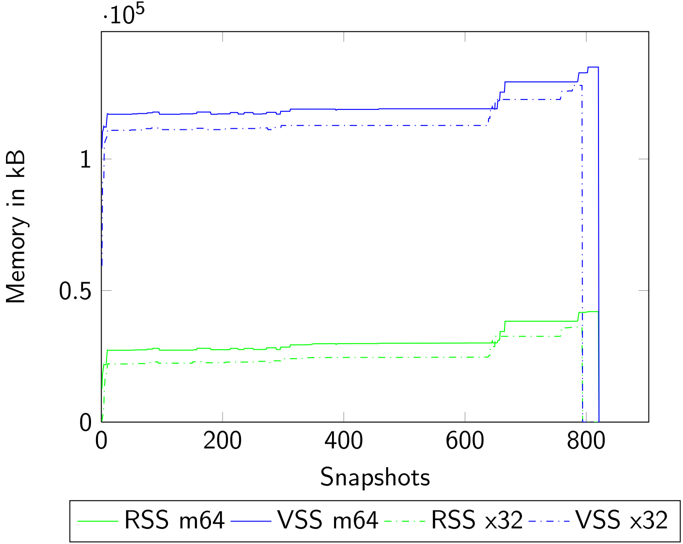
stress 1000:




```
// save headers
Int_t ntot = 0;
for (file=0;file<10;file++) {
   ntot += (Int_t)chTree[file]->GetEntries();
   chfile[file]->Write();
   delete chfile[file];
}
delete event;
delete hfile;
```

Results within ROOT-Benchmarks

stressHistogram:



Outline

Conclusion

- New room for improvement, (CERN) applications can profit substantially
- Computing Grid limits memory anyway to 4 GB per process
- In the context of multicore: each process can reduce memory consumption
- Gain in performance is for **FREE**

Conclusion (2)

Biggest drawback was the recompilation:

- Gaudi requires a lot of external packages
- CMT very inflexible
- Cast from pointer to long (...) will produce wrong results (xrootd)
- New pointer size required modifications in CINT (function and data pattern)

Getting a working environment:

- does not work out of the box
- building gcc fails due to missing glibc x32
- building glibc x32 with a partly built gcc

`http://www.gentoo.org/news/20120608-x32_abi.xml`

Any questions?

Thanks to: Axel Naumann, Ben Couturier, Benedikt Hegner and Marco Clemencic for their support