Myth and facts about 64-bit Linux®

Andreas Herrmann
André Przywara
AMD
Operating System Research Center

March 2nd, 2008

Myths...

You don't need 64-bit software with less than 3 GB RAM.

There are less drivers for 64-bit OS.

You will need all new software, all 64-bit.

... 64-bit software being twice as fast ...

640K ought to be enough for everybody ;-)

… and facts: The agenda

- 64-bit x86 hardware
  - Availability
  - Architecture extensions
- Software support
  - ABI extensions and GCC compiler
  - Linux kernel
- 32-bit compatibility
  - Hardware support
  - Linux® compat layer
- 32 => 64-bit porting issues
- Benchmarks and performance considerations

64-bit Hardware

64-bit x86 Hardware Availability

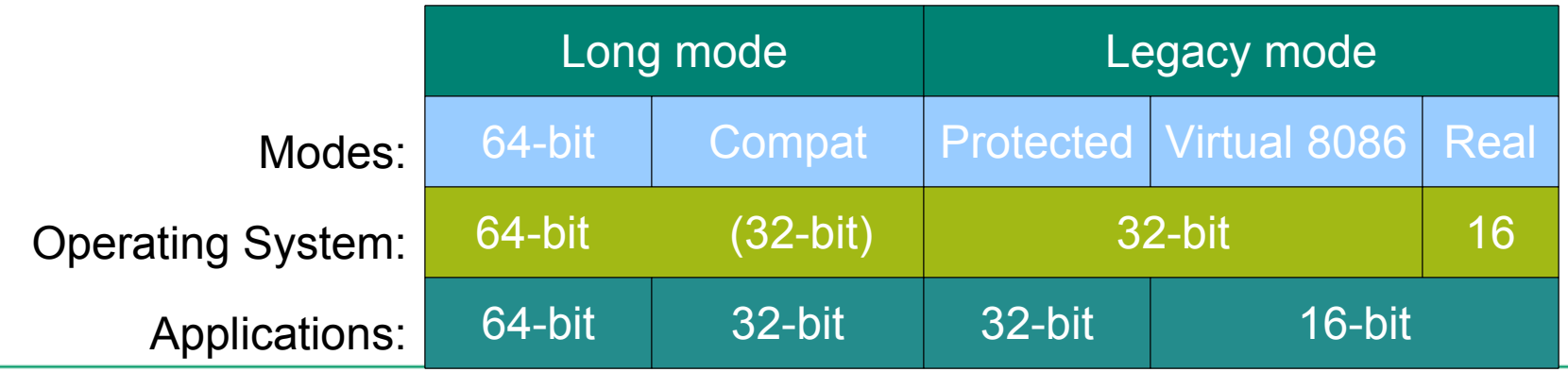
- Every AMD Opteron™ processor
- Every AMD Athlon™ 64 processor
- Every AMD Turion™ 64 processor
- Every AMD Phenom™ processor
- Newer AMD Sempron™ processors (since about 2005)

- Every Intel Core2™ processor
- Newer Intel Pentium™ 4 processors
- Newer Intel Xeon™ processors
- Some Intel Celeron™ processors

=> almost every nowadays sold x86 PC processor

Architecture extension: Operation modes

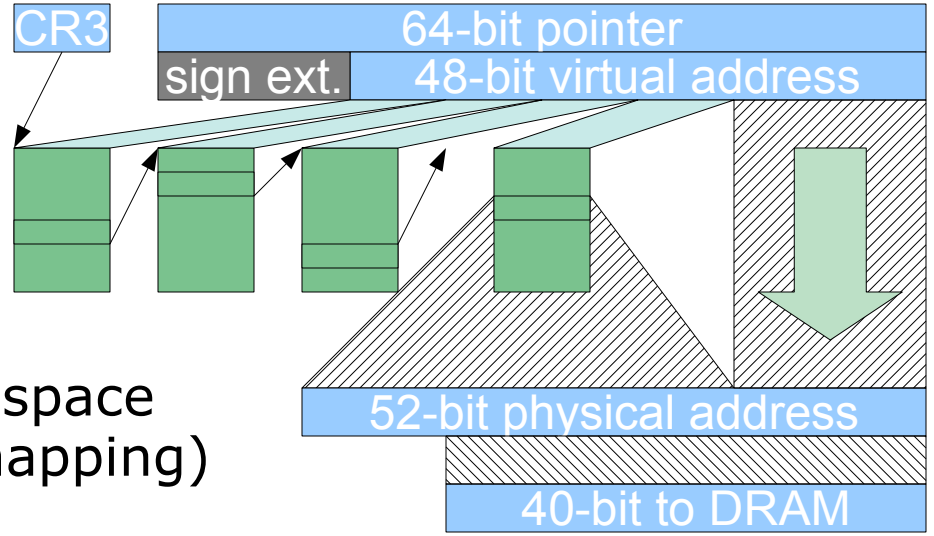
- Long mode introduced
- Two sub-modes:
  - Compatibility mode: similar to protected mode
  - 64-bit mode: full 64-bit capability
- Sub-modes are noted in one bit of CS descriptor
- Allows to run 32-bit binaries within a 64-bit environment
  - but requires explicit gateway to 64-bit code parts
- *Legacy* mode maintains 100% compatibility for 32-bit OSes



Architecture extensions: Address space

- Pointers (in registers) are always 64-bit
- 48 bits are used for virtual addressing
- Virtual address translated to physical address with paging
- Paging compatible to PAE mode (but extending to 4 levels)
- Restricts physical address to 52bit
- Current CPUs have a limited address bus anyway:
  - AMD Athlon™64: 40bits
  - AMD Phenom™: 48bits
  - Intel Core2™: 36bits
  - Intel Xeon™: 40bits

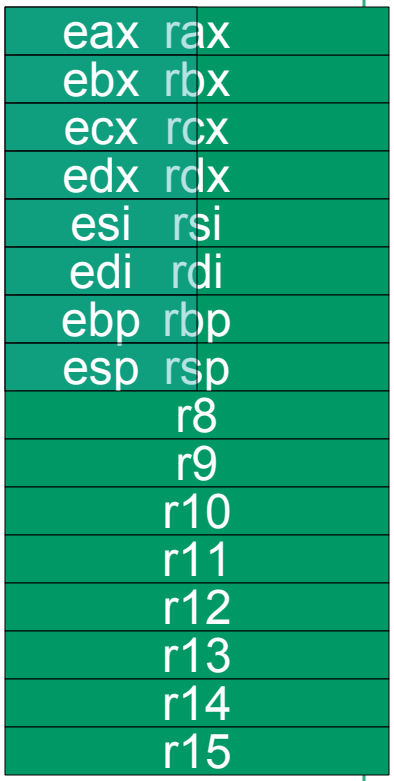
Extended virtual address space helps much (memory mapping)

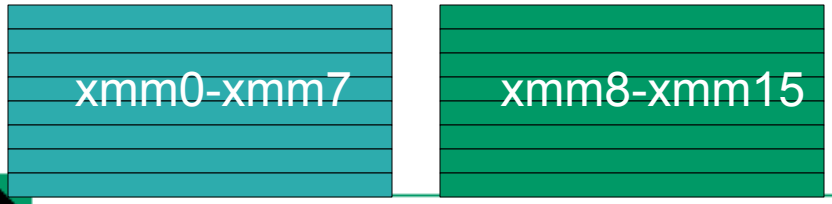


Architecture extensions: Register set

- General purpose registers extended to 64-bits
  - Replacing the "e" with an "r": eax -> rax, esp -> rsp, ...
  - 32-bit parts still available, simply use the "e"-name
- *Number* of registers doubled: r8 – r15 introduced
  - reason for performance improvements of many applications
  - Lower 32-bit can be accessed separately

SSE2 is the new natural floating point unit
Number of SSE registers doubled: xmm8-xmm15
SSE registers stay at 128bits (2*64-bit, 4*32-bit, 16*8bit)





Architecture extensions: Instruction set

- In long mode some obsolete x86 features have been removed:
  - Segmentation: base and limit are not checked
  - Hardware task switching
  - Call gates
- New addressing mode:
  - RIP relative
- Default operand size and immediates stay at 32-bits
- Special opcodes for loading 64-bit values (movabs)
- REX-Prefix byte for overriding default operand size (replacing one-byte inc and dec opcodes)

Software support for AMD64

gcc – Compiler support for AMD64

- most work done by SuSE engineers
- ABI describes conventions for binaries, compiler complies to
- Data types: int=32-bit, long int=64-bit, pointer=64-bit
- Parameter are passed in registers (6 integer, 8 floating point)
- Stack frame layout is simplified (-fomit-frame-pointers)
- Takes advantage of extended number of registers
- Uses 32-bit operands when possible to save space
- Contains cross-compiler functionality for i386 (-m32)

Linux x86-64 architecture

- Officially introduced in Linux 2.6 as arch/x86-64
- Merged with i386 in 2.6.24 (now: x86)
- No legacy code for older processors
- Introduces compat layer for i386 binaries

32-bit compatibility

32-bit compat: Hardware support

- **Compatibility** mode allows to run **unmodified** 32-bit binaries in a 64-bit environment (no 16-bit binaries!)
- Almost equal to 32-bit protected mode (including segmentation!)
- Syscalls switch between 32 and 64-bit
- Addresses get zero-extended to 64-bit
- Applications can use the lower 4 GB of **virtual** address space
  - Which can be mapped to any physical address range!

Linux compat layer

- Linux kernel maintains compat layer
- Allows user-space programs to be 32-bit
- Kernel cares about bitness
- Invokes appropriate /lib{32,64}/ld.so
- Booting a 32-bit installation with a 64-bit kernel works!
- Allows for several 32-bit apps to take more than 4GB
- Syscalls get translated to match size, pointers, structure layout and numbering

Linux compat in real life

- Switching between 32-bit and 64-bit applications is automatic
- 32-bit apps look for libraries in a different directory
- Actual algorithm depends on distribution
  - SuSE, RedHat: 32-bit: /lib, 64-bit: /lib64
  - Debian, Ubuntu: 32-bit: /lib32, 64-bit: /lib64 (symlinked to /lib)
- 32-bit apps need separate libraries, often called lib32*
- This applies to **all** libraries used (dependency chain!)
- Results in growing size of lib[32] directory
- linux32 tool to switch uname output for easier configuration (uses Linux personality feature)

Porting overview

- Many programs just need to be recompiled...
- ...but some may create problems
- Programmer visible changes
  - sizeof(long)==8
  - sizeof(void*)==8
- Look for
  - Usage of type *long*
  - Assignment of pointers to ints and vice versa
  - Dumping of data structures to disk / network
  - printf/scanf format specifiers
- Care about all warnings!

Porting issues: practical advices

- Don't assign pointers to ints and vice versa! (size mismatch)
  - use [u]intptr_t instead
- Don't dump structures to disk or over network (padding!)
  - use packed structs or better: write every single member
  - use explicit types: [u]int32_t, [u]int64_t, [u]int16_t
  - don't dump system structures like “struct stat” or “struct tm”!
- Use printf format specifiers from <inttypes.h> or casts
  - `int64_t foo; printf (“Number: %”PRId64”\n”, foo);`
  - `off_t ofs; printf (“Pos: %lli\n”, (long long) ofs);`

Benchmarks

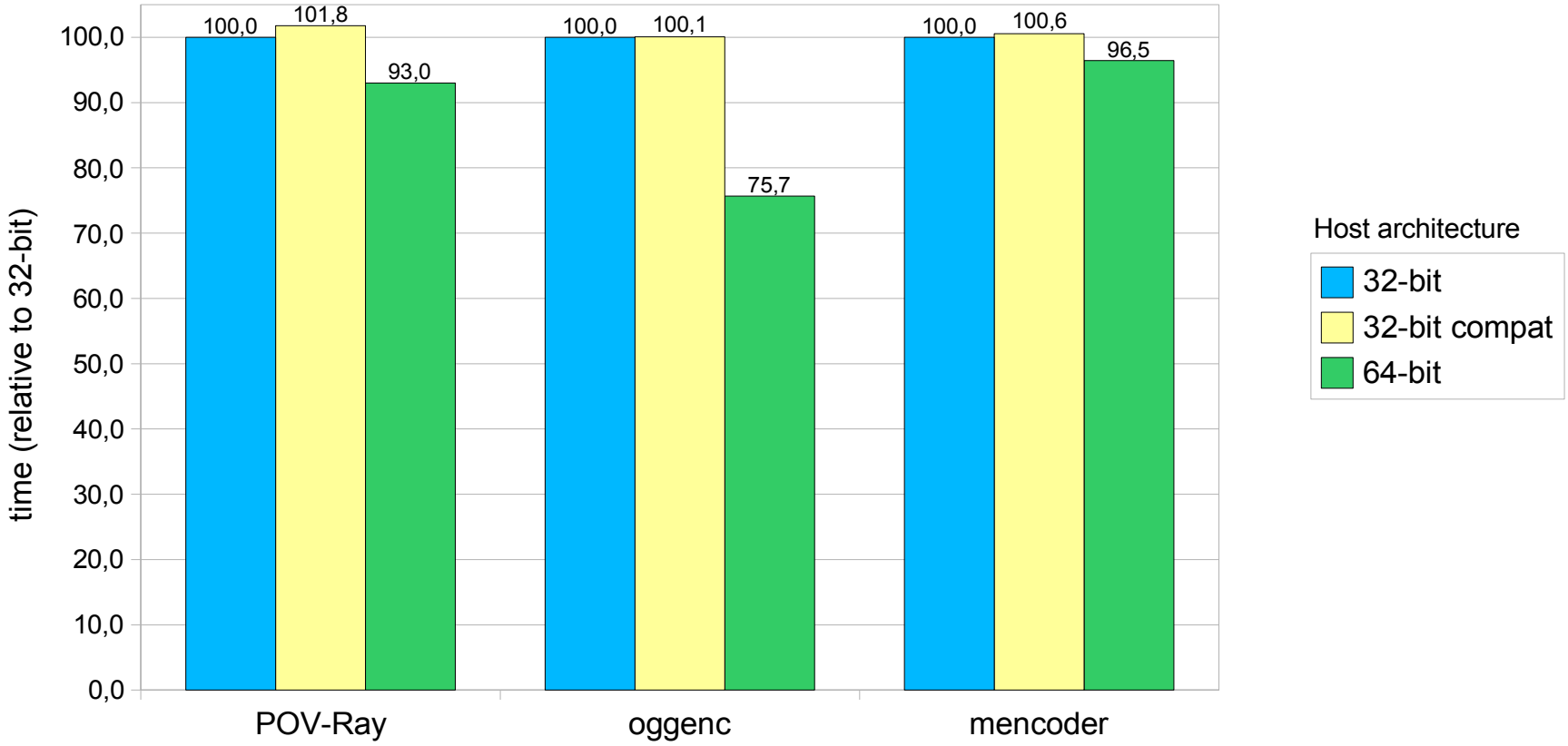
Benchmarks

- Real world benchmarks:
  - povray, oggenc, mencoder
  - kernel compilation
  - Comparing apples and oranges
    - compiling mozilla-firefox
    - compiling gtk+
- Microbenchmarks:
  - Syscall performance
  - Function call and 64-bit arithmetics

Benchmark setup

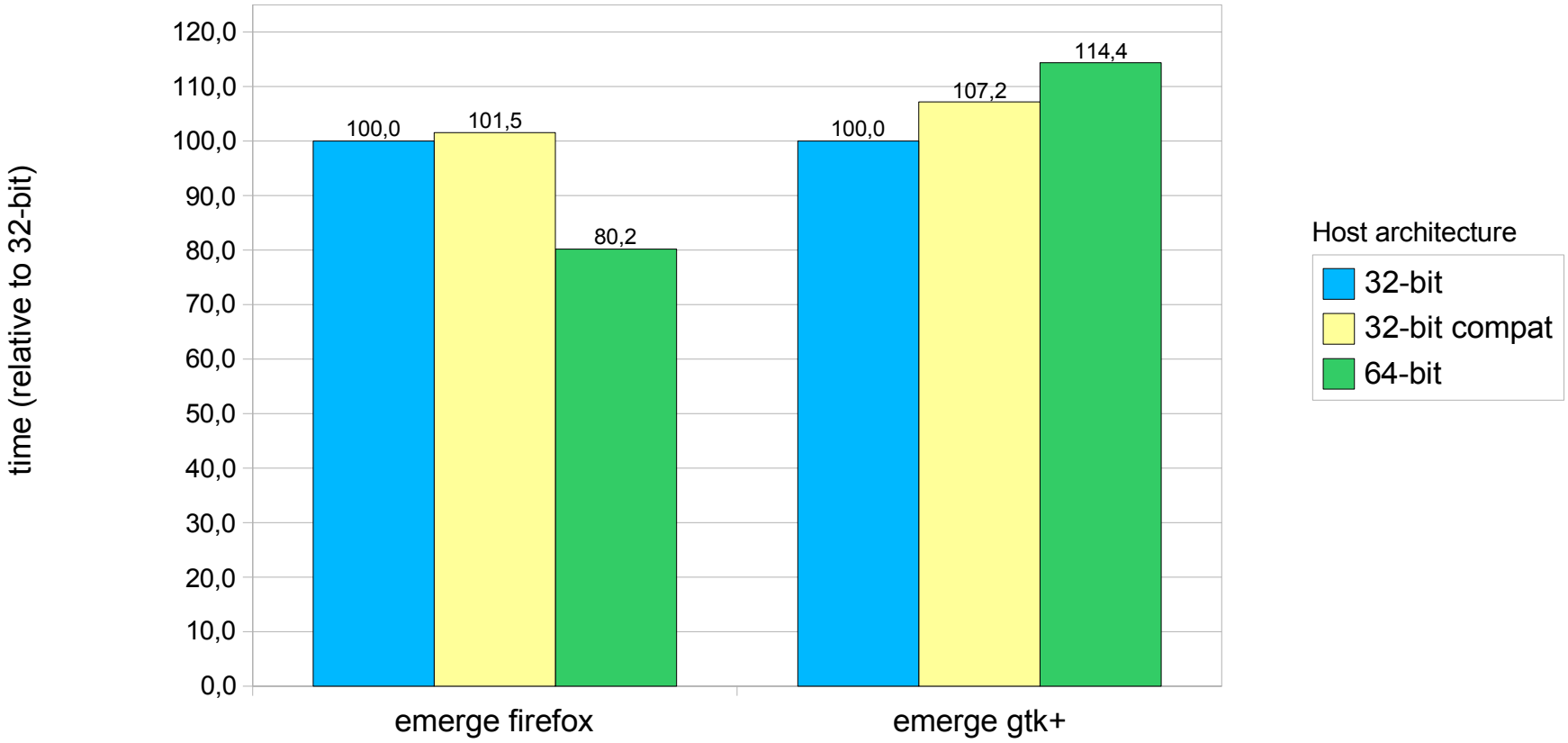
- Same hardware for all tests
  - Dual-core K8, 2x1024 kByte L2 cache, 1024 MByte RAM
- 2 Gentoo Linux installations (32-bit, 64-bit):
  - Identical USE flags, same packages installed
  - 64-bit: `CFLAGS="-march=k8 -O2 -pipe"`
  - 32-bit: `CFLAGS="-march=k8 -O2 -pipe -fomit-frame-pointer"`
- 3 “host architectures” as test environments:
  - 64-bit installation
  - 32-bit installation
  - 32-bit compat: 64-bit kernel with 32-bit installation and `linux32`
- Cross-compilers created using “`crossdev --stable --target ...`”

Benchmarks: povray, oggenc, mencoder



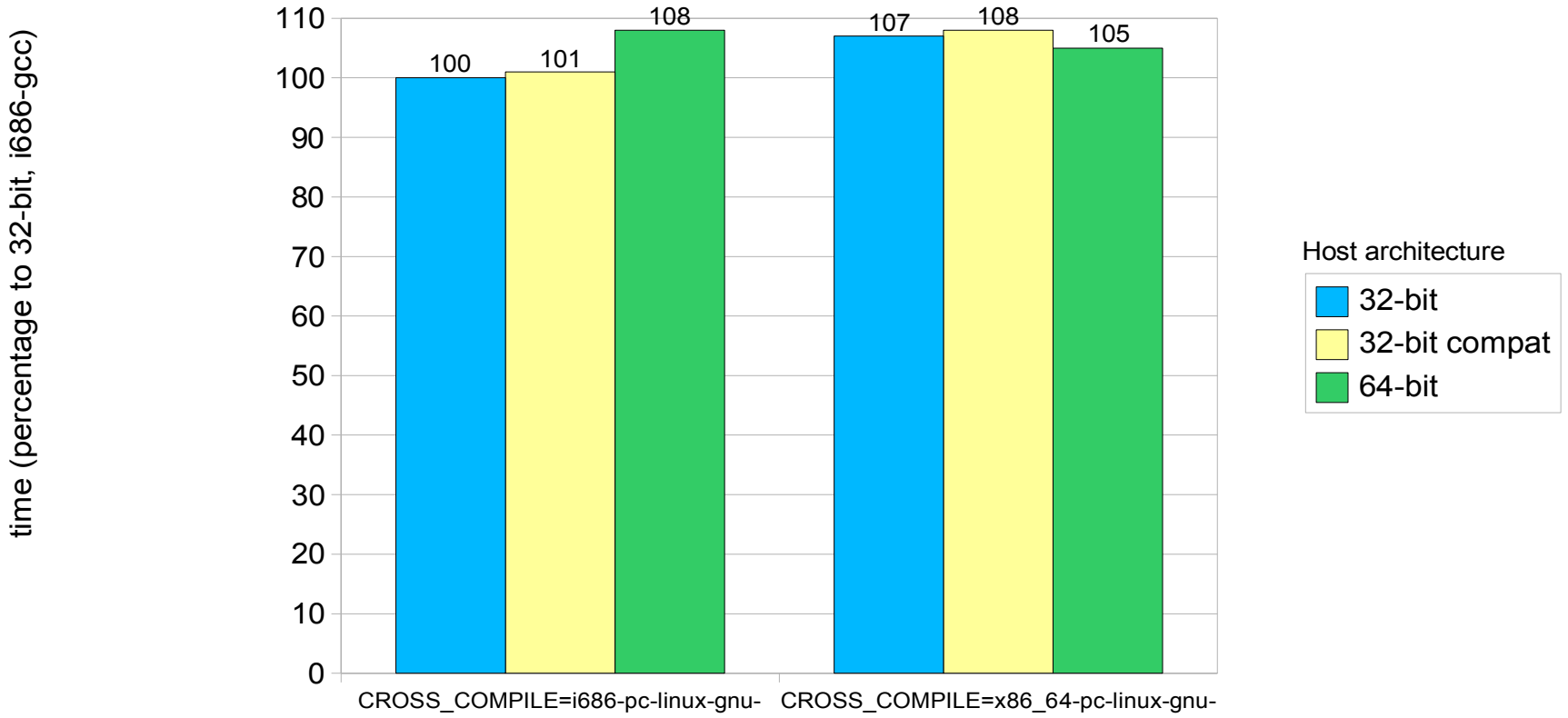
Time for a benchmark run, less is better

Benchmarks: “apples and oranges”



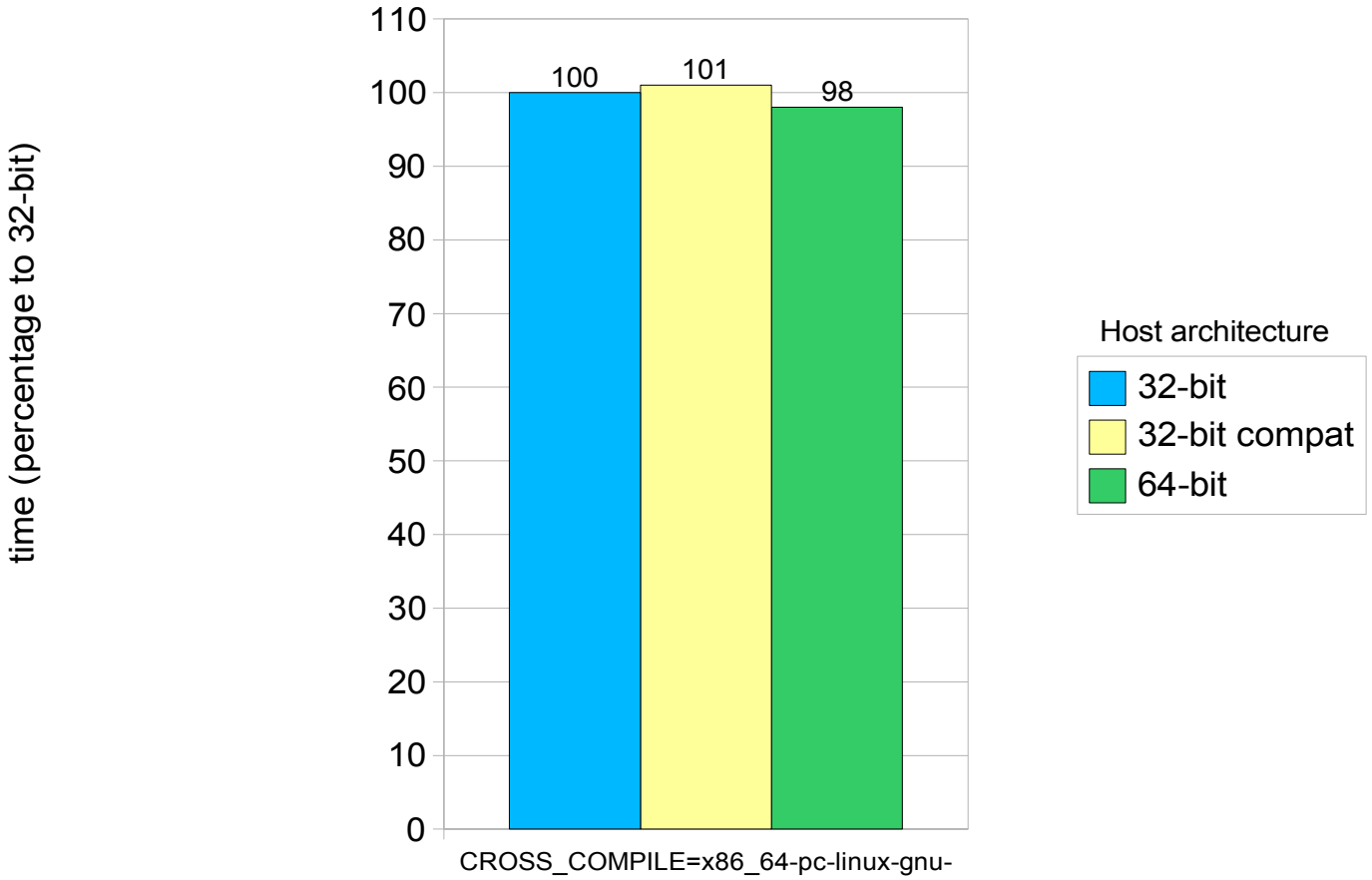
Time for a benchmark run, less is better

Benchmarks: Kernel compile (ARCH=i386)



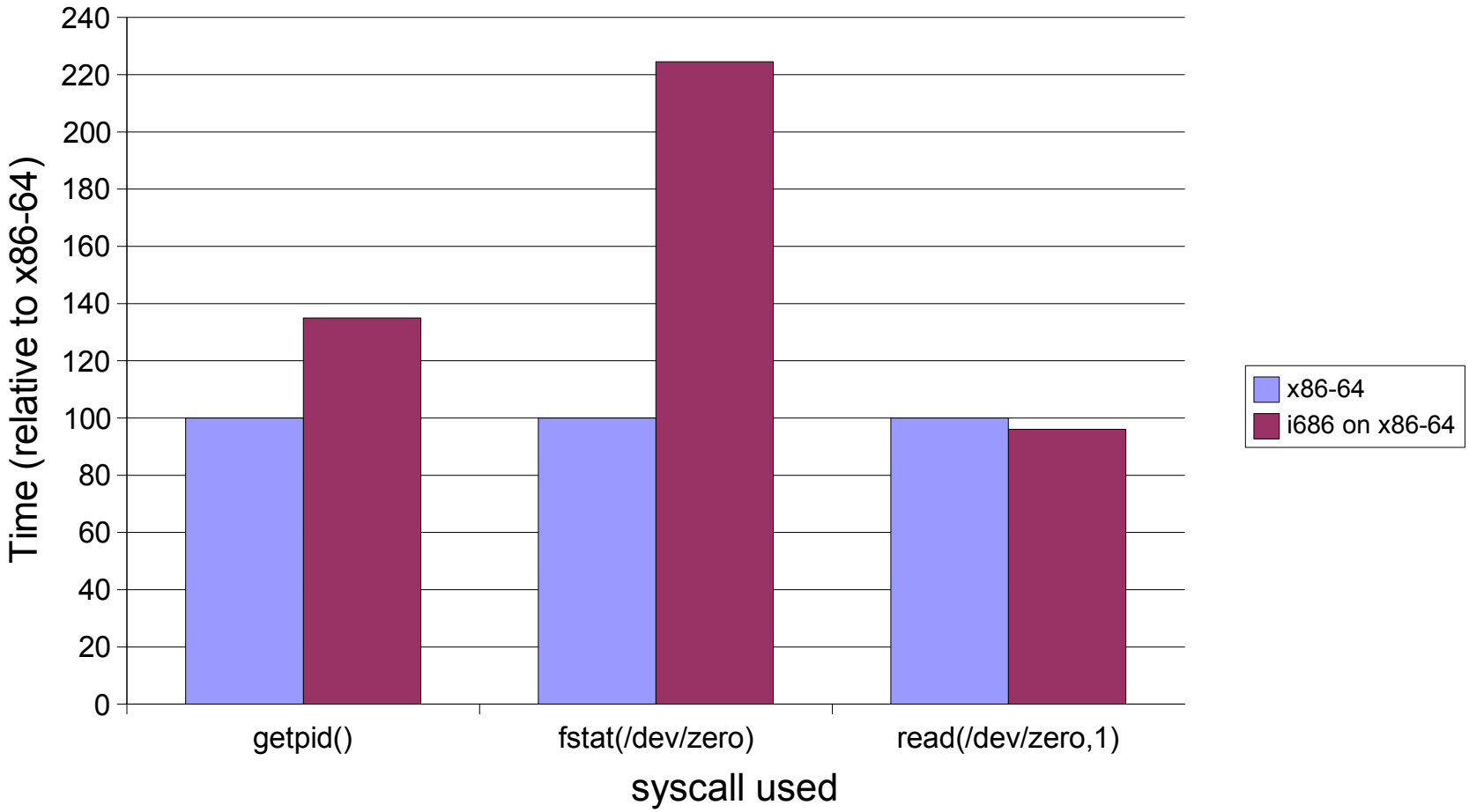
Time for a (cross) kernel compile, less is better

Benchmarks: Kernel compile (ARCH=x86_64)



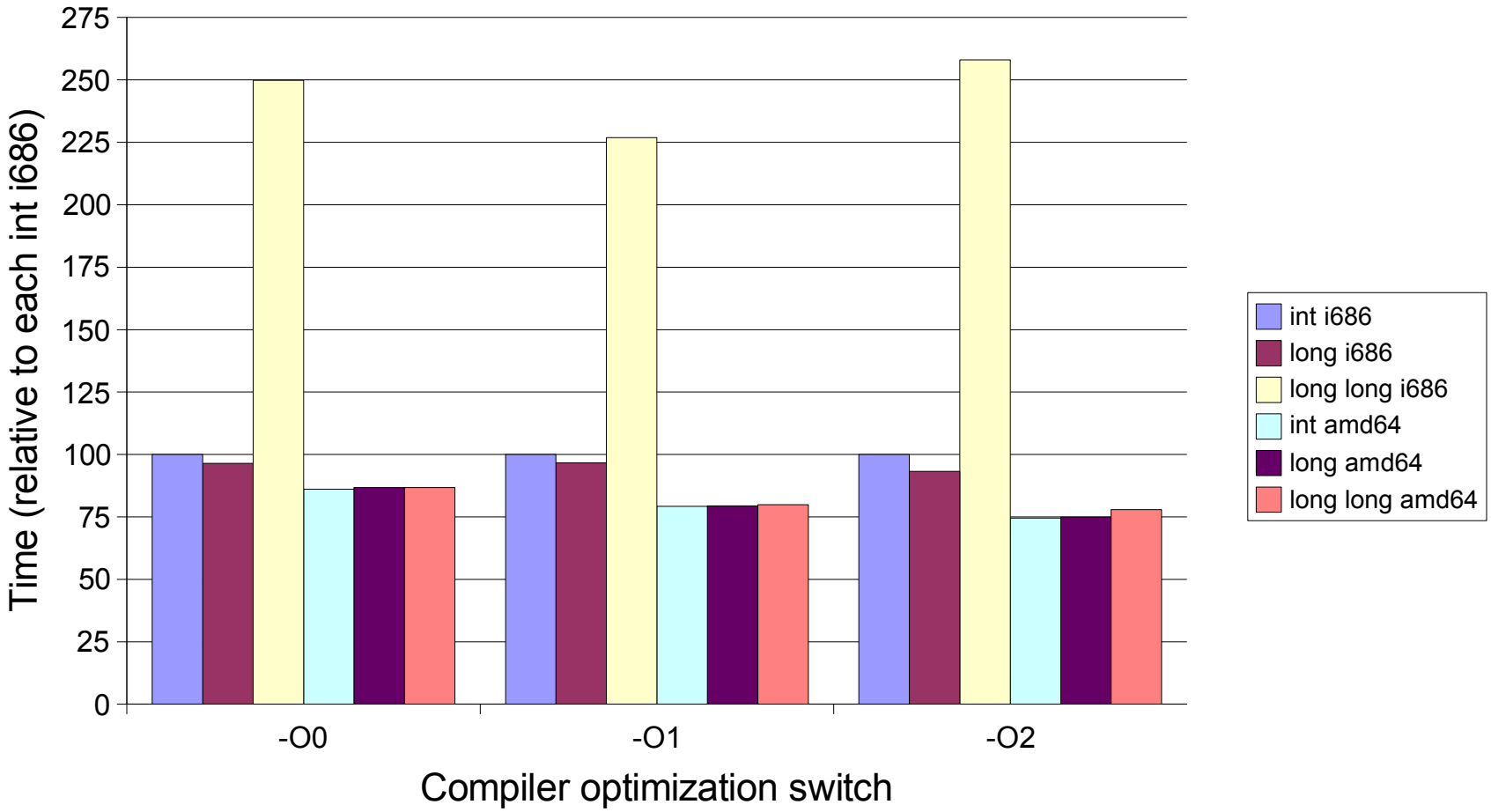
Time for a (cross) kernel compile, less is better

Microbenchmarks: Syscalls on compat



Time for calling 10 million syscalls in a loop, less is better

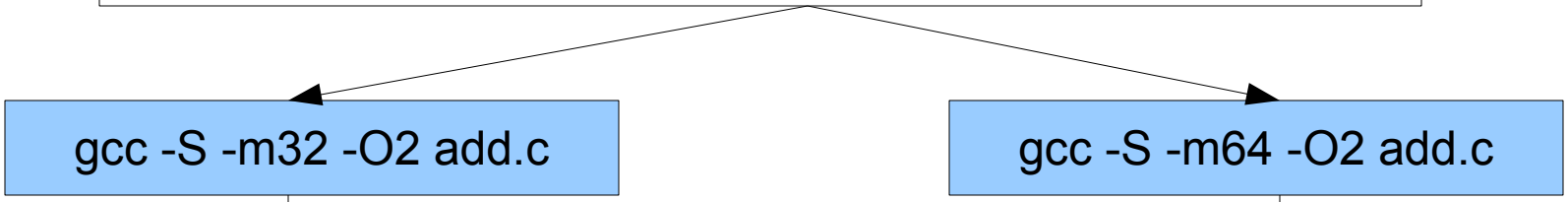
Microbenchmarks: Arithmetics



Doing additions of several numbers in a function called in a loop

Example: 64-bit arithmetics and function calls

```
uint64_t do_add (uint64_t a, uint64_t b)
{
    return a+b;
}
```




```
do_add:
        pushl   %ebp
        movl    %esp, %ebp

        movl    8(%ebp), %eax
        addl    16(%ebp), %eax
        movl    12(%ebp), %edx
        adcl    20(%ebp), %edx

        leave
        ret
```

instruction size: 17 bytes

```
do_add:
        leaq (%rdi,%rsi), %rax
        ret
```

instruction size: 5 bytes

Performance aspects

- Pro x86 64-bit Linux
  - Extended register set
  - Parameter passing in registers
  - Native 64-bit arithmetics
  - Enhanced virtual address space
- Contra x86 64-bit Linux
  - Larger memory footprint (larger binaries, 64-bit operands)
  - Cache utilization

Conclusions

Myths revisited

AMD Smarter Choice

You don't need 64-bit software with less than 3 GB RAM.

Performance advantages even on lesser equipped machines.

There are less drivers for 64-bit OS.

Mostly irrelevant for Linux (hail Open Source).

You will need all new software, all 64-bit.

32-bit compat mode performs very well and is transparent.

... 64-bit software being twice as fast ...

Only in very rare cases. (Lots of software is optimized for 32-bit.)

640K ought to be enough for everybody ;-)

2 GB or 4GB barrier is just around the corner.

Conclusion

Use a 64-bit system and use 32-bit apps in compat mode if necessary.

References

- Comparing 32-bit vs. 64-bit performance
  - Phoronix: "AMD Phenom 32-bit vs. 64-bit Performance" (http://www.phoronix.com/scan.php?page=article&item=998&num=1)
  - Zdnet: “Vista 32-bit vs. 64-bit & RTM vs. SP1” (http://blogs.zdnet.com/hardware/?p=1367)
  - Geekpatrol: “32-bit vs 64-bit Performance Under Mac OS X” (http://www.geekpatrol.ca/2006/09/32-bit-vs-64-bit-performance/)
- Jan Hubicka: Porting GCC to the AMD64 architecture (http://www.ucw.cz/~hubicka/papers/amd64/index.html)
- System V Application Binary Interface for AMD64 (http://www.x86-64.org/documentation/abi.pdf)
- Which **safe** compile flags to use for your Gentoo installation: (http://gentoo-wiki.com/Safe_Cflags)

Trademark Attribution

Porting: Usage of long

- Explicit usage of *long* is rare
- ANSI-C 90: sizeof(long)==sizeof(void*)
- This is still mostly true, but C99 drops this *assurance*!
- Better replace *long* by *int* or by *intptr_t*
- Think about hidden longs (typedefs like *size_t*, *off_t*)!
- Avoid unnecessary broadening to 64-bit
- Windows has complete different understanding of this!

- Advice: Track usage of long in program! Better replace it!

Porting: Assignment of pointers to ints

- Many programs use void* for opaque types
- Passing int types into those variables
- Compiler warns: incompatible size!
- Use [u]intptr_t (C99 type)
- #include <stdint.h>

Porting: Dumping to disk

- Writing *long*s and *pointer*s to disk or network breaks protocol
- Protocol assumes 32-bit size, but variable is 64-bit!
- Hardcoded size (4) works with write(little endian!), but is ugly
- Hardcoded size may work on reading, upper 32-bit undefined
- Portable size (sizeof(long)) will break it
- Replace types in structures with explicit types
  - [u]int32_t, [u]int64_t, [u]int16_t, [u]int8_t
- Then both hardcoded size and sizeof() work
- Alignment may change!
- Use __attribute__ ((__packed__)) on gcc
- Better avoid dumping or mapping of structs or variables at all

Porting: printf/scanf issues

- Source of many warnings:

```
typedef long int64_t;  // typedef long long int64_t for 32-bit
typedef int64_t off_t;
off_t filepos;
filepos=lseek (fd, SEEK_CUR, 0);
printf ("currently at %lli bytes\n", filepos);
```

- Warning on 64-bit: long long int specifier, long int argument!
- Solutions:
  - use "%z" for size_t, "%p" for pointers
  - cast arguments to long long and use "%lli"
- Specify explicit types with macros (#include <stdint.h>)
  - printf ("%"PRIi64"\n", myint64);