

# A look at 32- vs 64-bit userspace

**Olof Johansson**

*[olof@lixom.net](mailto:olof@lixom.net)*  
*[olof@pasemi.com](mailto:olof@pasemi.com)*

# The need for 64 bit computing

- Enables larger memory configurations
  - Without previous software complications
- Bigger memory address ranges
  - Databases, HPC, other large applications
- Simplifies some software
  - No need for manual memory management

# 64 bits -- costs?

- Code and data sizes
  - Cache, TLB footprints
  - Larger binaries = more disk space = more I/O
- Operating system overhead
  - More context to save (larger registers)
- Increased MMU complexity
- ...

# Implementations

- IA64, alpha
  - All 64-bit
  - No legacy compatibility to worry about
- AMD Opteron, IBM PPC, SPARC
  - Builds on 32-bit architectures
  - Offers mix/match of 32/64 bit operation
  - PPC: First OS support was 32-bit kernel, 64-bit processes

# AMD Opteron (x86\_64)

- 64-bit extensions to x86
- Adds new instructions, registers
  - Only accessible in 64-bit mode
- Similar memory management to x86
  - In-memory page tables with hardware walk
- New Linux/SysV ELF ABI
  - Exploits additions: register usage, calling conventions
- Intel support

# IBM PowerPC/POWER

- Additions to the PPC architecture
- No new general purpose registers
- MMU changes
  - Bridge architecture: table walk + hash lookup in hardware
  - Later: software managed SLB + hash lookup in hardware
- New Linux/SysV ELF ABI
  - New stack formats, calling conventions

# OS support

- Both x86\_64 and PPC64 allows for 32- or 64-bit userspace
- Distro support
  - x86\_64: 64-bit userspace with 32-bit add-ons
  - PPC64: 32-bit userspace with 64-bit add-ons
    - Exception: Gentoo ppc64
- 32-bit Linux kernels can not run 64-bit binaries

# OS support, cont

- Is a 64-bit /bin/lis or /bin/bash really needed?
- Cost/benefit of going to 64-bit
  - Performance differences
  - Two sets of libraries
    - Maintenance
    - Resource usage
- Needs a closer look to make a decision

# Benchmarks, cont

- Kernbench
  - Kernel compilation
- Imbench
  - Microbenchmark (processor, OS)
- OpenSSL
  - Encryption, hashes
- AIM
  - Multi-user server test

# Benchmarks, cont

- Hardware:  
PPC: Apple PowerMac G5, 2 x 2.0GHz PPC970  
AMD: Whitebox nForce4 planar, Athlon64 x2-3800
- Software:  
Gentoo 2005.1  
Mainline 2.6.14 kernel, 64-bit  
Base 64-bit installs, 32-bit chroots

# Kernbench

- Measures time to build a kernel
  - Primes filesystem caches, several runs
- Same kernel version, same toolchain
- crosstool: <http://kegel.com/crosstool>
  - Built/installed on all 4 environments
  - Build time observations:
    - x86/x86\_64: largely equivalent
    - PPC/PPC64: 64-bit 10% slower

# Kernbench results

## PPC:

Elapsed Time **208.69**

User Time 378.95

System Time 34.812

## x86:

Elapsed Time **162.612**

User Time 293.356

System Time 29.108

## PPC64:

Elapsed Time **253.47**

User Time 452.784

System Time 49.934

## x86\_64:

Elapsed Time **126.67**

User Time 219.252

System Time 27.288

- **PPC64 20% slower!**

- **x86\_64 27% faster!**

# Performance differences

- Why would x86 gain so much from going 64-bit?
- And why does PPC lose so much?
  
- Closer look at generated code
  - Pick one binary: gcc's x86-target cc1

# PPC vs PPC64

- Binary sizes: 6110158 vs 5273384
  - Some text, some data, some TOC
  - Text: 5020366 vs 4759812
- Fixed instruction size
  - More text = longer path lengths
- Two main sources:
  - Constants
  - ABI changes

# Constants on PPC

- Fixed 32-bit instructions
- Immediate values limited to 16 bits per instruction
- Loading 32-bit constants is two instructions
  - Load 16 bits, insert the other
- 64-bit is five instructions
  - Load 16 bits, insert 16 more, shift 32 bits, insert 16, insert 16
- Amount of constants is usually limited

# ABI changes on PPC

- Larger stack frames: 112 bytes
- Function descriptors
  - 24 bytes: entry point, TOC ptr, env ptr
- TOC pointer management
  - Trampoline functions to load new TOC pointers
  - NOP slot to restore old TOC on return

# x86 vs x86\_64

- More general purpose registers
- ABI changes
  - register parameter passing
- Binary sizes (cc1): 64-bit is 13% more text
- Variable length instructions
  - 13% more text, but 5% (13%) fewer instructions
  - Can contain 64-bit constants

# ABI changes on x86

- x86 is traditionally register starved
  - Register renaming helps, but not perfect
- New parameter passing conventions
  - Use registers instead of stack where possible
  - Stack parameter passing is fairly efficient, but uses more instructions

# AIM benchmark

- Simulates large chat server workload
- Several versions
  - OSDL version least stale: <http://reaim7.sf.net>
- Smaller differences:
  - PPC: 32-bit 2-6% faster
  - x86: 64-bit 2-2.5% faster

# Imbench

- Microbenchmark
- OS / hardware measurements
  - Cache/memory latencies
  - Networking
  - Filesystems
  - Fork/exec
- PPC: Most penalties in process handling, as expected
- x86: <xxx>

# OpenSSL

- Encryption library
  - Most popular encryptions and hashes
- Built-in performance measurement functions
- Caveats:
  - Not built with optimization on ppc64
  - Hand tuned loops for some functions/archs, not all

# OpenSSL, cont

- Results between x86 and PPC fairly similar
  - Most hashes faster on 32-bit
  - RSA, DSA faster on 64-bit
- Not all hashes/functions can make use of 64-bit registers
- ...or at least they haven't been tuned to do so

# Conclusions

- Bottom line:
  - PPC: 64-bit userspace: **BAD**
  - x86: 64-bit userspace: **GOOD**

... at least in the cases I have looked at

# Further work

- Some unexplained observations
  - crosstool build time on x86\_64
  - Context switch rates on kernbench/ppc64
- Optimization work
  - OpenSSL tunings for PPC, x86\_64?
- More performance measurements
  - Databases, JVM, desktop workloads

# Questions?