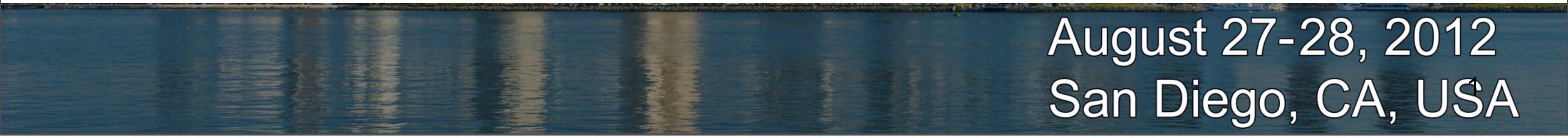
Evaluation of X32 ABI for Virtualization and Cloud

Jun Nakajima

Intel Corporation



August 27-28, 2012
San Diego, CA, USA

Agenda

- What is X32 ABI?
- Benefits for Virtualization & Cloud
- Evaluation of X32
- Summary

x32 ABI Basics

- x86-64 ABI but with 32-bit longs and pointers
- 64-bit arithmetic
- Fully utilize modern x86
- 8 additional integer registers (16 total)
- 8 additional SSE registers
- SSE for FP math
- 64-bit kernel is required
  - Linux kernel 3.4 has support for x32

Same memory footprint as x86 with advantages of x86-64.
No hardware changes are required.

ABI Comparison

	i386	x86-64	x32
Integer registers	6	15	15
FP registers	8	16	16
Pointers	4 bytes	8 bytes	4 bytes
64-bit arithmetic	No	Yes	Yes
Floating point	x87	SSE	SSE
Calling convention	Memory	Registers	Registers
PIC prologue	2-3 insn	None	None

The x32 Performance Advantage

- In the order of Expected Contribution:
  - Efficient Position Independent Code
  - Efficient Function Parameter Passing
  - Efficient 64-bit Arithmetic
  - Efficient Floating Point Operations

X32 is expected to give a 10-20% performance boost for C, C++

Efficient Position Independent Code

```
extern int x, y, z;
void foo () {  z = x * y; }
```

i386 psABI

```
    call __i686.get_pc_thunk.cx
    addl $_GLOBAL_OFFSET_TABLE_, %ecx
    movl y@GOT(%ecx), %eax
    movl x@GOT(%ecx), %edx
    movl (%eax), %eax
    imull        (%edx), %eax
    movl z@GOT(%ecx), %edx
    movl %eax, (%edx)
    ret

__i686.get_pc_thunk.cx:
    movl (%esp), %ecx
    ret
```

x32 psABI

```
     movl     x@GOTPCREL(%rip), %edx
     movl     y@GOTPCREL(%rip), %eax
     movl     (%rax), %rax
     imull     (%rdx), %rax
     movl     z@GOTPCREL(%rip), %edx
     movl     %rax, (%rdx)
     ret
```

X32 PIC code is shorter and faster

Efficient Function Parameter Passing

```
void bar (int x, int y, int z);
void foo (int x, int y, int z) { bar (y, z, x); }
```

i386 psABI

```
    subl    $28, %esp
    movl    32(%esp), %eax
    movl    %eax, 8(%esp)
    movl    40(%esp), %eax
    movl    %eax, 4(%esp)
    movl    36(%esp), %eax
    movl    %eax, (%esp)
    call    bar
    addl    $28, %esp
    ret
```

x32 psABI

```
    subl    $8, %esp
    movl    %edi, %eax
    movl    %esi, %edi
    movl    %edx, %esi
    movl    %eax, %edx
    call    bar
    addl    $8, %esp
    ret
```

X32 passes parameters in registers

Efficient 64-bit Integer Arithmetic

```
extern long long x, y, z;

void foo () {  z = x * y; }
```

i386 psABI

```
movl x+4, %edx
movl y+4, %eax
imull    y, %edx
imull    x, %eax
leal (%edx,%eax), %ecx
movl y, %eax
mull x
addl %ecx, %edx
movl %eax, z
movl %edx, z+4
ret
```

x32 psABI

```
movq x(%rip), %rax
imulq    y(%rip), %rax
movq %rax, z(%rip)
ret
```

X32 provides very efficient 64-bit integer support
(3 instructions vs. 10 instructions).

Efficient Floating Point Operation

```
extern double bar;
float foo () { return bar * bar; }
```

i386 psABI

```
        subl    $12, %esp
        movsd   bar, %xmm0
        mulsd   %xmm0, %xmm0
        movsd   %xmm0, (%esp)
        fldl    (%esp)
        addl    $12, %esp
        ret
```

x32 psABI

```
        movsd   bar(%rip), %xmm0
        mulsd   %xmm0, %xmm0
        ret
```

X32 doesn't use X87 to return FP values

Agenda

- What is X32 ABI?
- Benefits for Virtualization & Cloud
- Evaluation of X32
- Summary

Characteristics of Virtualization

- TLB misses in guests can be more expensive
  - Need to walk through both guest page tables and HAP (Hardware Assisted Paging) page tables
- Utilization of cache can be higher because of additional components
  - Hypervisor, driver domains
  - Other VMs

Advantage of x32 in Virtualization

- Over x86
  - Generic performance advantages of x32
  - Better cache utilization
    - fewer instructions, passing arguments in registers
- Over x86-64
  - Efficient guest page table structures
    - Only 4GB virtual address space
      - Single PML4 and 4 PDP directory entries in a row
  - Better cache utilization
    - 4-byte pointers

Advantage of x32 in Cloud

- Performance/memory is crucial in Cloud
  - 32-bit address space is still sufficient for many apps
  - Use memory for data, not for 8-byte pointers
- Use unified (64-bit) kernel
  - Use x32 ABI for 32-bit apps
  - Traditional x86 and x86-64 apps can run as well

Add x32 support to the list of OS images for Cloud

How should we use x32 in Xen?

- x32 apps in HVM Linux
  - Xen doesn’t need to know about x32
- x32 apps in PV Linux
  - Performance with x32 can regress because of 64-bit PV issues
  - Should be run in HVM container
- x32 PV Linux
  - 32-bit PV Linux with more registers used
  - Practically insignificant

Run x32 apps in HVM

Agenda

- What is X32 ABI?
- Benefits for Virtualization & Cloud
- Evaluation of X32
- Summary

Configurations

- VM
  - HVM (Linux) with 1GB/4GM memory
  - VCPUs (1, 2)
  - i386, x32, x86-64 apps
- Simple Web Applications
  - mysql-server (5.1.52) (run remotely, 64-bit)
  - apache (2.2.22), php (5.4.4), varnish
    - Compiled for i386, x32, x86-64 ABI

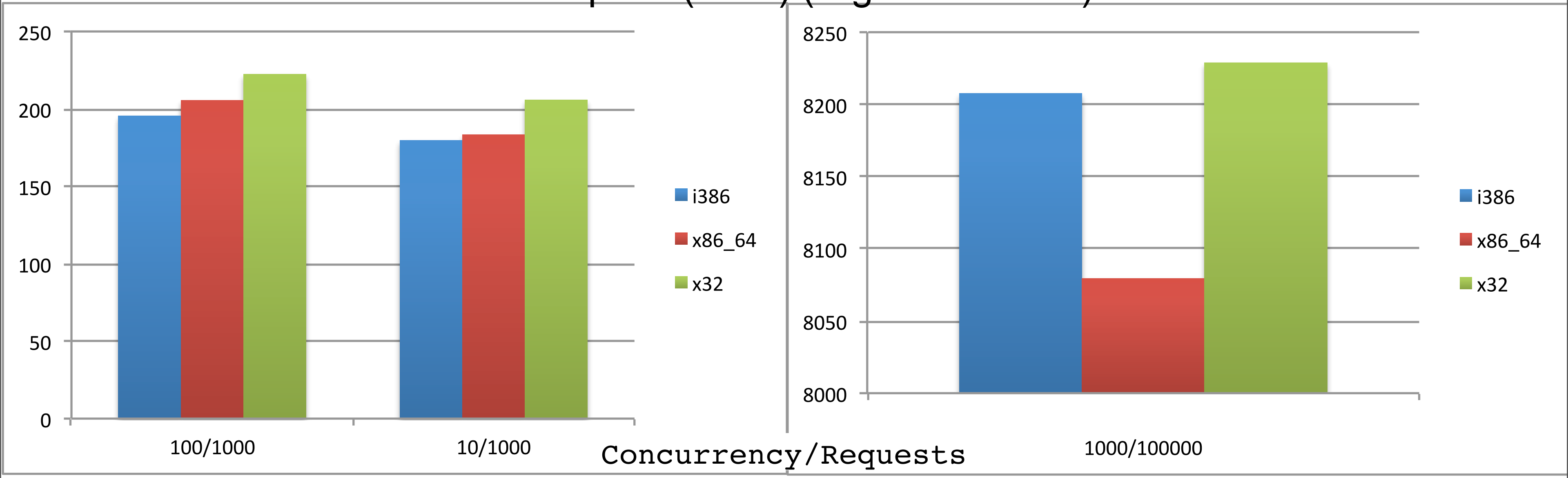
Simple Web Apps (1)

```
$ ab -c $concurrency -n $requests http://$guestip/drupal/
index.php
```

1GB, 2 VCPUs

1GB, 1 VCPU

Request (/sec.) (Higher is better)



No Varnish Enable X-Drupal-Cache

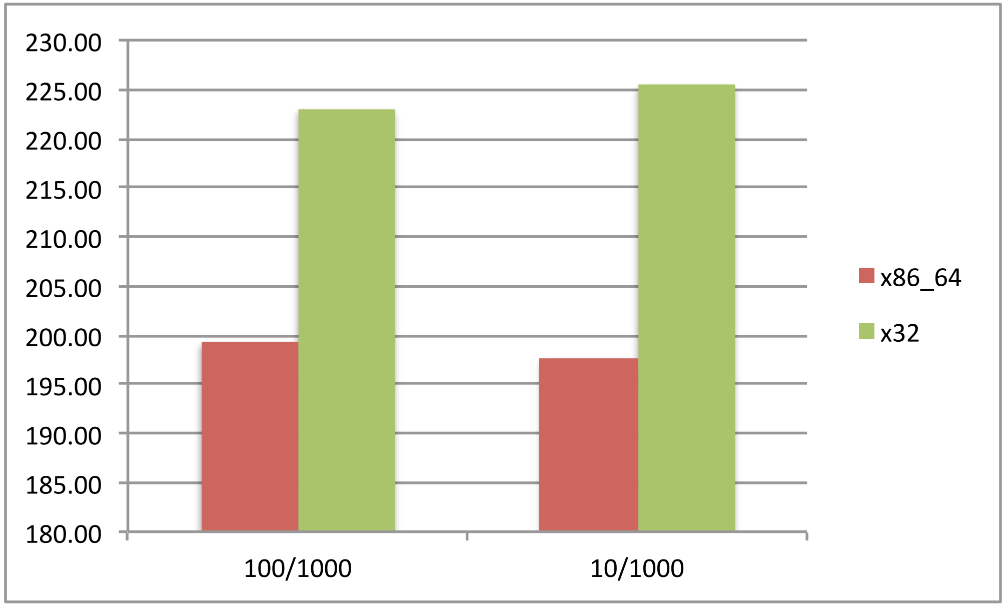
Enable Varnish Enable X-Drupal-Cache

Simple Web Apps (2)

```
$ ab -c $concurrency -n $requests http://$guestip/drupal/
index.php
```

Request (/sec.) (Higher is better)

4GB, 2 VCPUs



Concurrency/Requests

No Varnish Enable X-Drupal-Cache

Issues Found

- Some apps are not x32 clean
  - Code assumes 64-bit long and pointers using `__amd64__` or `__x86_64__`
  - Assembly code that assumes 64-bit long and pointers

Agenda

- What is X32 ABI?
- Benefits for Virtualization & Cloud
- Evaluation of X32
- Summary

Summary

- Generic advantages of x32 apps
  - See for details and status of x32 ABI:
    - https://sites.google.com/site/x32abi/
- Virtualization and Cloud can take more advantage of x32 apps
- Preliminary results look promising
  - x32 was always better than i386 or x86-64 with simple Web apps*
- Build and test more Web apps

*: Intel internal measurements