[v4,1/2] i386: Add Intel Processor Trace feature support

Message ID	1520182116-16485-1-git-send-email-luwei.kang@intel.com
State	New
Headers	show
Series	[v4,1/2] i386: Add Intel Processor Trace feature support
Related	show

diff (/patch/881307/raw/) | mbox (/patch/881307/mbox/) | series (/series/31864/mbox/)

Commit Message

Kang, Luwei (/project/qemu-devel/list/?submitter=69591) March 4, 2018, 4:48 p.m.

```
From: Chao Peng <chao.p.peng@linux.intel.com>

Expose Intel Processor Trace feature to guest.

To make Intel PT live migration safe and get same CPUID information
with same CPU model on diffrent host. CPUID[14] is constant in this
patch. Intel PT use EPT is first supported in IceLake, the CPUID[14]
get on this machine as default value. Intel PT would be disabled
if any machine don't support this minial feature list.

Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
Signed-off-by: Luwei Kang <luwei.kang@intel.com>
---
From V3:
 - fix some typo;
 - add some comments and safty check.

---
 target/i386/cpu.c | 78 ++++++++++++++++++++++++++++++++++++++++++++++++++++--
 target/i386/cpu.h |  1 +
 target/i386/kvm.c | 23 ++++++++++++++++
 3 files changed, 100 insertions(+), 2 deletions(-)
```

Comments

Eduardo Habkost (/project/qemu-devel/list/?submitter=195) March 9, 2018, 7:10 p.m. | #1 (/comment/1872631/)

```
On Mon, Mar 05, 2018 at 12:48:35AM +0800, Luwei Kang wrote:
> From: Chao Peng <chao.p.peng@linux.intel.com>
>
> Expose Intel Processor Trace feature to guest.
>
> To make Intel PT live migration safe and get same CPUID information
> with same CPU model on diffrent host. CPUID[14] is constant in this
> patch. Intel PT use EPT is first supported in IceLake, the CPUID[14]
> get on this machine as default value. Intel PT would be disabled
> if any machine don't support this minial feature list.
>
> Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
> Signed-off-by: Luwei Kang <luwei.kang@intel.com>
> ---
> From V3:
>  - fix some typo;
>  - add some comments and safty check.
>
> ---
>  target/i386/cpu.c | 78 +++++++++++++++++++++++++++++++++++++++++++++++++++--
>  target/i386/cpu.h |  1 +
>  target/i386/kvm.c | 23 +++++++++++++++++
>  3 files changed, 100 insertions(+), 2 deletions(-)
>
> diff --git a/target/i386/cpu.c b/target/i386/cpu.c
> index b5e431e..24e1693 100644
> --- a/target/i386/cpu.c
> +++ b/target/i386/cpu.c
> @@ -173,7 +173,32 @@
>  #define L2_ITLB_4K_ASSOC       4
>  #define L2_ITLB_4K_ENTRIES   512
>
> -
> +/* CPUID Leaf 0x14 constants: */
> +#define INTEL_PT_MAX_SUBLEAF     0x1
> +/*
> + * bit[00]: IA32_RTIT_CTL.CR3 filter can be set to 1 and IA32_RTIT_CR3_MATCH
> + *          MSR can be accessed;
> + * bit[01]: Support Configurable PSB and Cycle-Accurate Mode;
> + * bit[02]: Support IP Filtering, TraceStop filtering, and preservation
> + *          of Intel PT MSRs across warm reset;
> + * bit[03]: Support MTC timing packet and suppression of COFI-based packets;
> + */
> +#define INTEL_PT_MINIMAL_EBX     0xf

Thanks!  I didn't expect a detailed description of each bit.  I
thought that just adding macros for each bit instead of
hardcoding 0xf would be enough.

But after reading the docs, I understand it could be difficult to
```

```
choose a macro name for something like "support of IP Filtering,
TraceStop filtering, and preservation of Intel PT MSRs across
warm reset", so this description looks like the best we can do.
:)


I only see a problem below:

> +/*
> + * bit[00]: Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1 and
> + *          IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be
> + *          accessed;
> + * bit[01]: ToPA tables can hold any number of output entries, up to the
> + *          maximum allowed by the MaskOrTableOffset field of
> + *          IA32_RTIT_OUTPUT_MASK_PTRS;
> + * bit[02]: Support Single-Range Output scheme;
> + */
> +#define INTEL_PT_MINIMAL_ECX     0x7
> +#define INTEL_PT_ADDR_RANGES_NUM 0x2 /* Number of configurable address ranges */
> +#define INTEL_PT_ADDR_RANGES_NUM_MASK 0x3
> +#define INTEL_PT_MTC_BITMAP      (0x0249 << 16) /* Support ART(0,3,6,9) */
> +#define INTEL_PT_CYCLE_BITMAP    0x1fff         /* Support 0,2^(0~11) */
> +#define INTEL_PT_PSB_BITMAP      (0x003f << 16) /* Support 2K,4K,8K,16K,32K,64K */
>
>  static void x86_cpu_vendor_words2str(char *dst, uint32_t vendor1,
>                                       uint32_t vendor2, uint32_t vendor3)
[...]
> @@ -4083,6 +4129,34 @@ static int x86_cpu_filter_features(X86CPU *cpu)
>          }
>      }
>
> +    if ((env->features[FEAT_7_0_EBX] & CPUID_7_0_EBX_INTEL_PT) &&
> +        kvm_enabled()) {
> +        KVMState *s = CPU(cpu)->kvm_state;
> +        uint32_t eax_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_EAX);
> +        uint32_t ebx_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_EBX);
> +        uint32_t ecx_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_ECX);
> +        uint32_t eax_1 = kvm_arch_get_supported_cpuid(s, 0x14, 1, R_EAX);
> +        uint32_t ebx_1 = kvm_arch_get_supported_cpuid(s, 0x14, 1, R_EBX);
> +
> +        if (!eax_0 ||
> +           ((ebx_0 & INTEL_PT_MINIMAL_EBX) != INTEL_PT_MINIMAL_EBX) ||
> +           ((ecx_0 & INTEL_PT_MINIMAL_ECX) != INTEL_PT_MINIMAL_ECX) ||
> +           ((eax_1 & INTEL_PT_MTC_BITMAP) != INTEL_PT_MTC_BITMAP) ||
> +           ((eax_1 & INTEL_PT_ADDR_RANGES_NUM_MASK) <
> +                                        INTEL_PT_ADDR_RANGES_NUM) ||
> +           ((ebx_1 & (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP)) !=
> +                (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP))) {

I still don't see a check to ensure the host has bit 31 on ecx_0
set to 0, as I mentioned when reviewing v3.
```

```
The rest of the patch looks good.

> +            /*
> +             * Processor Trace capabilities aren't configurable, so if the
> +             * host can't emulate the capabilities we report on
> +             * cpu_x86_cpuid(), intel-pt can't be enabled on the current host.
> +             */
> +            env->features[FEAT_7_0_EBX] &= ~CPUID_7_0_EBX_INTEL_PT;
> +            cpu->filtered_features[FEAT_7_0_EBX] |= CPUID_7_0_EBX_INTEL_PT;
> +            rv = 1;
> +        }
> +    }
> +
>      return rv;
>  }
>
[...]
```

Kang, Luwei (/project/qemu-devel/list/?submitter=69591) March 12, 2018, 9:07 a.m. | #2 (/comment/1873254/)

```
> > +
> > +        if (!eax_0 ||
> > +           ((ebx_0 & INTEL_PT_MINIMAL_EBX) != INTEL_PT_MINIMAL_EBX) ||
> > +           ((ecx_0 & INTEL_PT_MINIMAL_ECX) != INTEL_PT_MINIMAL_ECX) ||
> > +           ((eax_1 & INTEL_PT_MTC_BITMAP) != INTEL_PT_MTC_BITMAP) ||
> > +           ((eax_1 & INTEL_PT_ADDR_RANGES_NUM_MASK) <
> > +                                        INTEL_PT_ADDR_RANGES_NUM) ||
> > +           ((ebx_1 & (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP)) !=
> > +                (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP))) {
> >
> I still don't see a check to ensure the host has bit 31 on ecx_0 set to 0, as I mentioned when reviewing v3.

Hi Eduardo,
    Thanks for the code review. I don't quite understand here why bit31 must same with host (meaning we must reject a host
where ecx_0 & (1 << 31) is set).
    Do you mean PT must be disabled in guest when host bit31 is set?
    Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
    I can't find any special on this bit. Could you help clarify?


Thanks,
Luwei Kang

>
> The rest of the patch looks good.
>
> > +            /*
> > +             * Processor Trace capabilities aren't configurable, so if the
> > +             * host can't emulate the capabilities we report on
> > +             * cpu_x86_cpuid(), intel-pt can't be enabled on the current host.
> > +             */
> > +            env->features[FEAT_7_0_EBX] &= ~CPUID_7_0_EBX_INTEL_PT;
> > +            cpu->filtered_features[FEAT_7_0_EBX] |= CPUID_7_0_EBX_INTEL_PT;
> > +            rv = 1;
> > +        }
> > +    }
> > +
> >      return rv;
> >  }
> >
> [...]
>
> --
> Eduardo
```

Eduardo Habkost (/project/qemu-devel/list/?submitter=195) March 12, 2018, 4:45 p.m. | #3 (/comment/1873667/)

```
On Mon, Mar 12, 2018 at 09:07:41AM +0000, Kang, Luwei wrote:
> > > +
> > > +        if (!eax_0 ||
> > > +           ((ebx_0 & INTEL_PT_MINIMAL_EBX) != INTEL_PT_MINIMAL_EBX) ||
> > > +           ((ecx_0 & INTEL_PT_MINIMAL_ECX) != INTEL_PT_MINIMAL_ECX) ||
> > > +           ((eax_1 & INTEL_PT_MTC_BITMAP) != INTEL_PT_MTC_BITMAP) ||
> > > +           ((eax_1 & INTEL_PT_ADDR_RANGES_NUM_MASK) <
> > > +                                          INTEL_PT_ADDR_RANGES_NUM) ||
> > > +           ((ebx_1 & (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP)) !=
> > > +                (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP))) {
> >
> > I still don't see a check to ensure the host has bit 31 on ecx_0 set to 0, as I mentioned when reviewing v3.
>
> Hi Eduardo,
>     Thanks for the code review. I don't quite understand here why bit31 must same with host (meaning we must reject a host
> where ecx_0 & (1 << 31) is set).

If the guest sees the bit set to 0, it will expect IP payloads
with RIP values, but the host CPU will generate IP payloads with
LIP values.  I assume KVM won't do RIP<->LIP translation on the
packets generated by the host before the guest sees them, will
it?


>     Do you mean PT must be disabled in guest when host bit31 is set?
>     Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
>     I can't find any special on this bit. Could you help clarify?

As far as I understand, this bit is special because KVM can't
emulate a value that's different from the host.
```

Kang, Luwei (/project/qemu-devel/list/?submitter=69591) March 13, 2018, 11:16 a.m. | #4 (/comment/1874193/)

```
> > > > +        if (!eax_0 ||
> > > > +           ((ebx_0 & INTEL_PT_MINIMAL_EBX) != INTEL_PT_MINIMAL_EBX) ||
> > > > +           ((ecx_0 & INTEL_PT_MINIMAL_ECX) != INTEL_PT_MINIMAL_ECX) ||
> > > > +           ((eax_1 & INTEL_PT_MTC_BITMAP) != INTEL_PT_MTC_BITMAP) ||
> > > > +           ((eax_1 & INTEL_PT_ADDR_RANGES_NUM_MASK) <
> > > > +                                          INTEL_PT_ADDR_RANGES_NUM) ||
> > > > +           ((ebx_1 & (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP)) !=
> > > > +                (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP))) {
> > >
> > > I still don't see a check to ensure the host has bit 31 on ecx_0 set to 0, as I mentioned when reviewing v3.
> >
> > Hi Eduardo,
> >     Thanks for the code review. I don't quite understand here why
> > bit31 must same with host (meaning we must reject a host where ecx_0 & (1 << 31) is set).
>
> If the guest sees the bit set to 0, it will expect IP payloads with RIP values, but the host CPU will generate IP payloads with LIP values.
> I assume KVM won't do RIP<->LIP translation on the packets generated by the host before the guest sees them, will it?

Fully understand. Will make a separate patch on this.

Thanks,
Luwei Kang

>
>
> >     Do you mean PT must be disabled in guest when host bit31 is set?
> >     Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
> >     I can't find any special on this bit. Could you help clarify?
>
> As far as I understand, this bit is special because KVM can't emulate a value that's different from the host.
>
> --
> Eduardo
```

Patch

diff (/patch/881307/raw/)	mbox (/patch/881307/mbox/)	series (/series/31864/mbox/)

```
diff --git a/target/i386/cpu.c b/target/i386/cpu.c
index b5e431e..24e1693 100644
--- a/target/i386/cpu.c
+++ b/target/i386/cpu.c
@@ -173,7 +173,32 @@
 #define L2_ITLB_4K_ASSOC       4
 #define L2_ITLB_4K_ENTRIES   512

-
+/* CPUID Leaf 0x14 constants: */
+#define INTEL_PT_MAX_SUBLEAF     0x1
+/*
+ * bit[00]: IA32_RTIT_CTL.CR3 filter can be set to 1 and IA32_RTIT_CR3_MATCH
+ *          MSR can be accessed;
+ * bit[01]: Support Configurable PSB and Cycle-Accurate Mode;
+ * bit[02]: Support IP Filtering, TraceStop filtering, and preservation
+ *          of Intel PT MSRs across warm reset;
+ * bit[03]: Support MTC timing packet and suppression of COFI-based packets;
+ */
+#define INTEL_PT_MINIMAL_EBX     0xf
+/*
+ * bit[00]: Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1 and
+ *          IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be
+ *          accessed;
+ * bit[01]: ToPA tables can hold any number of output entries, up to the
+ *          maximum allowed by the MaskOrTableOffset field of
+ *          IA32_RTIT_OUTPUT_MASK_PTRS;
+ * bit[02]: Support Single-Range Output scheme;
+ */
+#define INTEL_PT_MINIMAL_ECX     0x7
+#define INTEL_PT_ADDR_RANGES_NUM 0x2 /* Number of configurable address ranges */
+#define INTEL_PT_ADDR_RANGES_NUM_MASK 0x3
+#define INTEL_PT_MTC_BITMAP      (0x0249 << 16) /* Support ART(0,3,6,9) */
+#define INTEL_PT_CYCLE_BITMAP    0x1fff         /* Support 0,2^(0~11) */
+#define INTEL_PT_PSB_BITMAP      (0x003f << 16) /* Support 2K,4K,8K,16K,32K,64K */

 static void x86_cpu_vendor_words2str(char *dst, uint32_t vendor1,
                                      uint32_t vendor2, uint32_t vendor3)
@@ -428,7 +453,7 @@  static FeatureWordInfo feature_word_info[FEATURE_WORDS] = {
             NULL, NULL, "mpx", NULL,
             "avx512f", "avx512dq", "rdseed", "adx",
             "smap", "avx512ifma", "pcommit", "clflushopt",
-            "clwb", NULL, "avx512pf", "avx512er",
+            "clwb", "intel-pt", "avx512pf", "avx512er",
             "avx512cd", "sha-ni", "avx512bw", "avx512vl",
         },
         .cpuid_eax = 7,
@@ -3453,6 +3478,27 @@  void cpu_x86_cpuid(CPUX86State *env, uint32_t index, uint32_t count,
         }
         break;
```

```
      }
+     case 0x14: {
+         /* Intel Processor Trace Enumeration */
+         *eax = 0;
+         *ebx = 0;
+         *ecx = 0;
+         *edx = 0;
+         if (!(env->features[FEAT_7_0_EBX] & CPUID_7_0_EBX_INTEL_PT) ||
+             !kvm_enabled()) {
+             break;
+         }
+
+         if (count == 0) {
+             *eax = INTEL_PT_MAX_SUBLEAF;
+             *ebx = INTEL_PT_MINIMAL_EBX;
+             *ecx = INTEL_PT_MINIMAL_ECX;
+         } else if (count == 1) {
+             *eax = INTEL_PT_MTC_BITMAP | INTEL_PT_ADDR_RANGES_NUM;
+             *ebx = INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP;
+         }
+         break;
+     }
      case 0x40000000:
          /*
           * CPUID code in kvm_arch_init_vcpu() ignores stuff
@@ -4083,6 +4129,34 @@  static int x86_cpu_filter_features(X86CPU *cpu)
          }
      }

+     if ((env->features[FEAT_7_0_EBX] & CPUID_7_0_EBX_INTEL_PT) &&
+         kvm_enabled()) {
+         KVMState *s = CPU(cpu)->kvm_state;
+         uint32_t eax_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_EAX);
+         uint32_t ebx_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_EBX);
+         uint32_t ecx_0 = kvm_arch_get_supported_cpuid(s, 0x14, 0, R_ECX);
+         uint32_t eax_1 = kvm_arch_get_supported_cpuid(s, 0x14, 1, R_EAX);
+         uint32_t ebx_1 = kvm_arch_get_supported_cpuid(s, 0x14, 1, R_EBX);
+
+         if (!eax_0 ||
+            ((ebx_0 & INTEL_PT_MINIMAL_EBX) != INTEL_PT_MINIMAL_EBX) ||
+            ((ecx_0 & INTEL_PT_MINIMAL_ECX) != INTEL_PT_MINIMAL_ECX) ||
+            ((eax_1 & INTEL_PT_MTC_BITMAP) != INTEL_PT_MTC_BITMAP) ||
+            ((eax_1 & INTEL_PT_ADDR_RANGES_NUM_MASK) <
+                                        INTEL_PT_ADDR_RANGES_NUM) ||
+            ((ebx_1 & (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP)) !=
+                 (INTEL_PT_PSB_BITMAP | INTEL_PT_CYCLE_BITMAP))) {
+             /*
+              * Processor Trace capabilities aren't configurable, so if the
+              * host can't emulate the capabilities we report on
+              * cpu_x86_cpuid(), intel-pt can't be enabled on the current host.
+              */
```

```
+            env->features[FEAT_7_0_EBX] &= ~CPUID_7_0_EBX_INTEL_PT;
+            cpu->filtered_features[FEAT_7_0_EBX] |= CPUID_7_0_EBX_INTEL_PT;
+            rv = 1;
+        }
+    }
+
     return rv;
 }

diff --git a/target/i386/cpu.h b/target/i386/cpu.h
index faf39ec..f5b9ecb 100644
--- a/target/i386/cpu.h
+++ b/target/i386/cpu.h
@@ -640,6 +640,7 @@  typedef uint32_t FeatureWordArray[FEATURE_WORDS];
 #define CPUID_7_0_EBX_PCOMMIT  (1U << 22) /* Persistent Commit */
 #define CPUID_7_0_EBX_CLFLUSHOPT (1U << 23) /* Flush a Cache Line Optimized */
 #define CPUID_7_0_EBX_CLWB     (1U << 24) /* Cache Line Write Back */
+#define CPUID_7_0_EBX_INTEL_PT (1U << 25) /* Intel Processor Trace */
 #define CPUID_7_0_EBX_AVX512PF (1U << 26) /* AVX-512 Prefetch */
 #define CPUID_7_0_EBX_AVX512ER (1U << 27) /* AVX-512 Exponential and Reciprocal */
 #define CPUID_7_0_EBX_AVX512CD (1U << 28) /* AVX-512 Conflict Detection */
diff --git a/target/i386/kvm.c b/target/i386/kvm.c
index ad4b159..f9f4cd1 100644
--- a/target/i386/kvm.c
+++ b/target/i386/kvm.c
@@ -865,6 +865,29 @@  int kvm_arch_init_vcpu(CPUState *cs)
                 c = &cpuid_data.entries[cpuid_i++];
             }
             break;
+        case 0x14: {
+            uint32_t times;
+
+            c->function = i;
+            c->index = 0;
+            c->flags = KVM_CPUID_FLAG_SIGNIFCANT_INDEX;
+            cpu_x86_cpuid(env, i, 0, &c->eax, &c->ebx, &c->ecx, &c->edx);
+            times = c->eax;
+
+            for (j = 1; j <= times; ++j) {
+                if (cpuid_i == KVM_MAX_CPUID_ENTRIES) {
+                    fprintf(stderr, "cpuid_data is full, no space for "
+                                "cpuid(eax:0x14,ecx:0x%x)\n", j);
+                    abort();
+                }
+                c = &cpuid_data.entries[cpuid_i++];
+                c->function = i;
+                c->index = j;
+                c->flags = KVM_CPUID_FLAG_SIGNIFCANT_INDEX;
+                cpu_x86_cpuid(env, i, j, &c->eax, &c->ebx, &c->ecx, &c->edx);
+            }
+            break;
```

```
+        }
         default:
             c->function = i;
             c->flags = 0;
```