HQEMU v2.5.1
Technical Report

Virtualization Development Team

October 25, 2017

Contents

1 Introduction

HQEMU (Hybrid QEMU) is a cross-ISA, retargetable and multi-threaded dynamic binary translator on multicores. HQEMU integrates QEMU and LLVM as its building blocks, and supports process-level emulation and full-system virtualization. Figure 1 shows the organization of HQEMU. For more design details, please refer to the paper on http://dl.acm.org/citation.cfm?id=2259016.2259030.

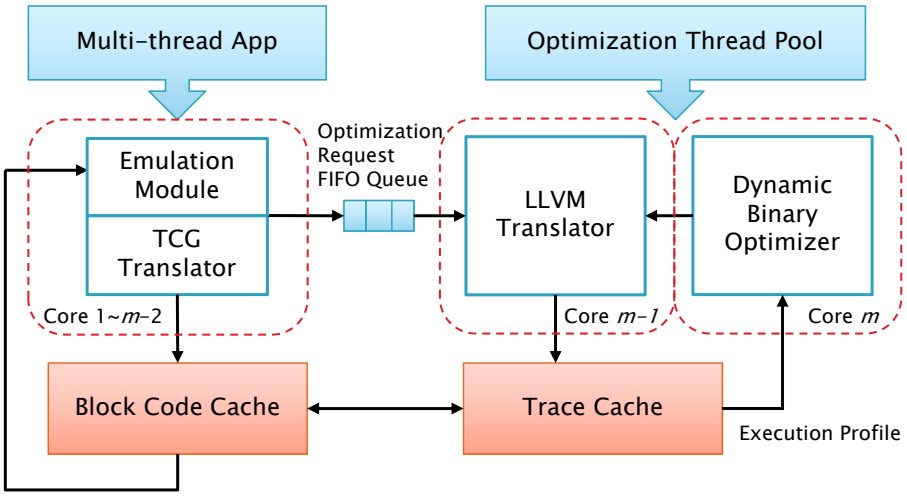

Figure 1: Major components of HQEMU on an m-core platform.

1.1 ISAs Supported in HQEMU v2.5.1

Guest ISA: x86, x86-64, arm, aarch64.
Host ISA: x86, x86-64.

1.2 Requirements

- Linux with recent kernel version. We use 2.6.30 and later, but earlier versions should also work.
- LLVM version 3.5 and later. Earlier versions are not supported.

HQEMU v2.5.1 is based on QEMU version 2.5. It has been tested in several Linux distributions, such as CentOS/x86, Debian/x86 and Gentoo/x86 Linux. Other Linux distributions are not tested but may also work. Table 1 lists the LLVM versions that are successfully integrated in HQEMU for the supported host ISAs. Please select the correct LLVM version when building HQEMU. (Note that HQEMU supports both MCJIT and the legacy JIT with LLVM v3.5. MCJIT is only supported with all other LLVM versions.)

Table 1: Supported LLVM versions for the host ISA.

Host ISA	**LLVM Version**		
	3.5	3.8	3.9
x86/x86-64	Yes	Yes	Yes

2 Installing and running HQEMU on the x86 host

In the following instructions, we use $HQEMU to denote the top directory of the HQEMU source tree.

2.1 Installing dependent packages

The following software packages are currently required by HQEMU:

- LLVM 3.5 or newer
- Clang 3.5 or newer
- GCC 4.x or newer

You can download prebuilt x86 binary of Clang from the official LLVM website. Since patching LLVM[1][2] is required to work with HQEMU, you **MUST** build LLVM from source.
The following example assumes that you will be building LLVM version 3.9 and Clang 3.9 from source and installing them in the /usr/local directory. x86-64 machine is used in this example.

2.1.1 Building LLVM and Clang

```
$ ls
cfe-3.9.0.src.tar.xz  compiler-rt-3.9.0.src.tar.xz
llvm-3.9.0.src.tar.xz

$ tar -xf llvm-3.9.0.src.tar.xz
$ cd llvm-3.9.0.src
$ patch -p1 < $HQEMU/patch/llvm/llvm-3.9.patch
$ mkdir build

$ cd tools
$ tar -xf ../../cfe-3.9.0.src.tar.xz
$ mv cfe-3.9.0.src clang
$ cd ../projects
$ tar -xf ../../compiler-rt-3.9.0.src.tar.xz
$ mv compiler-rt-3.9.0.src compiler-rt
$ cd ../build

# for llvm using cmake
$ cmake -G "Unix Makefiles" \
        -DCMAKE_INSTALL_PREFIX=/usr/local/llvm-3.9 \
        -DCMAKE_BUILD_TYPE=Release ..
```

[1] patch to reserve AREG0 register required by QEMU
[2] patch file 'llvm-${VERSION}.patch' is placed in the $HQEMU/patch/llvm directory

```
# for llvm using configure
$ ../configure --prefix=/usr/local/llvm-3.9 \
               --enable-optimized \
               --disable-assertions

$ make && make install
$ export PATH=/usr/local/llvm-3.9/bin:$PATH
```

2.2 Building and installing HQEMU

The building of HQEMU requires LLVM tool *llvm-config* to locate the LLVM header files and libraries, and *clang* to generate required LLVM bitcode file. Thus, make sure that paths of these tools have been added to your PATH environment variable before the building process (see Section 2.1.1). The default steps for building and installing HQEMU are as follows:

```
$ tar -zxf hqemu-2.5.1.tar.gz
$ mkdir build
$ cd build
$ ../hqemu-2.5.1/configure --prefix=`pwd` \
                           --target-list=i386-linux-user \
                           --enable-llvm
$ make && make install
```

Here are some configure arguments which may be of interest:

- --prefix=<path>: install all files in specified directory (/usr/local/ is the default if --prefix is not specified)
- --target-list=<list>: set target lists separated by comma (e.g., --target-list=i386-linux-user,arm-linux-user for process-level emulation; --target-list=i386-softmmu,arm-softmmu for full-system emulation)

2.3 Running HQEMU

You can download user mode test programs 'linux-user-test-0.3.tar.gz' from QEMU official website http://wiki.qemu.org/Testing. This tarball consists of test programs for Linux user-mode emulation and their associated runtime libraries for different architectures.

```
$ cd build
$ tar -zxf linux-user-test-0.3.tar.gz
$ ./bin/qemu-i386 -L./linux-user-test-0.3/gnemul/qemu-i386 \
                    ./linux-user-test-0.3/i386/sha1

SHA1=15dd99a1991e0b3826fede3deffc1feba42278e6
```

If error message 'Error: Bitcode file <path>/llvm_helper_i386.bc: No such file or directory' shows up, you need to manually create directory <path> and move bitcode file `llvm_helper_i386.bc` from your HQEMU installation path to directory <path>.

Users can use the environment variable, LLVM_MODE, to control HQEMU's execution mode, e.g.,

```
$ export LLVM_MODE=<mode>
```

The available mode values are:

- `none`: use the QEMU/TCG translator only. No LLVM optimization is used.
- `block`: use the LLVM translator only, with block-level translation.
- `hybrids`: HQEMU running execution and LLVM optimizations with one single thread.
- `hybridm`: HQEMU running execution and LLVM optimizations with different threads. (same as LLVM_MODE unset)

2.4 Debugging HQEMU

Developers who wish to find more verbose information about how the LLVM translator is working can use the debug masks to select different debugging level. The debugging level can be assigned by setting the '`-debuglv=`' option via the LLVM_CMD environment variable before running a program. The default debugging level is set to `none`. You can enable multiple debugging levels at the same time with desired debug masks separated by comma. The available debug mask values are listed below:

- `none`: no debug message will be displayed. This is the default debugging level.
- `llvm`: status of the LLVM translator.
- `in_asm`: display assembly codes of guest program.
- `op`: display TCG IRs of guest program.
- `out_asm`: display generated host assembly codes.
- `ir`: LLVM IRs before HQEMU specific optimizations.
- `ir_opt`: LLVM IRs after HQEMU specific optimizations.
- `entry`: translation functions that are called.
- `verify`: check whether the LLVM function is well-formed.
- `asm`: turn on debugging levels of `in_asm`, `op` and `out_asm`.
- `debug`: turn on debugging levels of `llvm`, `ir_opt` and `out_asm`.
- `all`: turn on all debugging levels

NOTE: debug messages are directed to 'stderr' and will be displayed on the screen.

The example below shows parts of the messages with debug masks, `llvm` and `ir_opt`, being set:

```
$ export LLVM_CMD='-debuglv=llvm,ir_opt'
$ ./bin/qemu-i386 -L./linux-user-test-0.3/gnemul/qemu-i386/ \
                    ./linux-user-test-0.3/i386/sha1

[00:00:00.001080] Initializing LLVM Environment.
[00:00:00.001309] Creating 1 translator(s).
[00:00:00.001322] Starting LLVM Translator 0.
[00:00:00.014509] LLVM initialized for target (x86_64-linux-gnu).
[00:00:00.019975] Use LLVM disassembler for guest (i386).
[00:00:00.019985] Use LLVM disassembler for host (x86_64).
[00:00:00.020002] LLVM IR Factory initialized.
[00:00:00.020070] LLVM Translator 0 initialized.
[00:00:00.042496] LLVM MCJIT initialized.
[00:00:00.042519] Requested trace info: pc 0x40802a46 length 1
[00:00:00.049942]     - Found a direct branch to pc 0x40802a53
[00:00:00.050006]     - Found a direct branch to pc 0x40802a46
[00:00:00.050026] Compile: start...
[00:00:00.050054] PostProcess: pc 0x40802a46 length 1 is_loop 1

define void @"40802a46"() noreturn nounwind naked {
init:
  %cpu = call i8* asm sideeffect "", "={r14}"() nounwind
  %cpu.struct = bitcast i8* %cpu to %struct.CPUX86State*
  %0 = getelementptr i8* %cpu, i32 8
  %1 = getelementptr i8* %cpu, i32 0
  %2 = getelementptr i8* %cpu, i32 48
  %3 = getelementptr i8* %cpu, i32 40
  %4 = getelementptr i8* %cpu, i32 44
  %edx = bitcast i8* %0 to i32*
  %eax = bitcast i8* %1 to i32*
  %cc_op = bitcast i8* %2 to i32*
  %cc_src = bitcast i8* %3 to i32*
  %cc_dst = bitcast i8* %4 to i32*
  %env = bitcast i8* %1 to i64*
  (......)
  br label %entry

entry:                                ; preds = %loopback, %init
  %eax.a.0 = phi i32 [ %5, %init ], [ %13, %loopback ]
  %10 = shl i32 %eax.a.0, 2
  %11 = add i32 %6, %10
```

```
  %12 = inttoptr i32 %11 to i32*
  store volatile i32 0, i32* %12
  %13 = add i32 %eax.a.0, 1
  %14 = sub i32 %13, 97
  %15 = add i32 %14, 97
  %16 = icmp ule i32 %15, 97
  br i1 %16, label %loopback, label %false_dest.block

loopback:                                 ; preds = %entry
  br label %entry, !exit !0
  (......)
}
```

2.5 Profiling HQEMU

Users who wish to profile a program's execution behavior can use the '`-profile=`' option via the LLVM_CMD environment variable to control the profiling levels of a target program. The default profiling level is set to `none`. You can enable multiple profiling levels at the same time with desired values separated by comma. The available profiling mask values are listed below:

- `none`: no profiling message will be displayed. This is the default profiling level.
- `basic`: static information. This level includes amount of basic blocks, guest and host assembly code size, guest instruction count and translation time; traces' amount, thresholds of trace generation, expected length, the number of trace exits, the number of indirect branch in a trace, guest and host assembly code size, translation time.
- `trace`: trace's runtime information. This level includes the values of trace counters.
- `cache`: addresses/sizes of basic block cache and trace caches.
- `all`: turn on all profiling levels.

NOTE: profiling messages are directed to 'stderr' and will be displayed on the screen.

The following example demonstrates the profiling results of program 'sha1'.

```
$ export LLVM_CMD='-profile=basic,trace'
$ ./bin/qemu-i386 -L./linux-user-test-0.3/gnemul/qemu-i386/ \
                    ./linux-user-test-0.3/i386/sha1

SHA1=15dd99a1991e0b3826fede3deffc1feba42278e6
-------------------------
Block statistic:
Num of Blocks  : 1856
G/H Code size  : 40378/422012 bytes
```

```
Guest iCount   : 11435
Trans. Cycles  : 179473023 cycles

Trace statistic:
Num of Traces  : 32
Profile Thres. : 50
Predict Thres. : 16
G/H Code size  : 7499/27626 bytes
Trans. Cycles  : 605679246 cycles
Average Len    : 4.1 (max=16)
Average # Exit : 3.5 (max=12)
Average # IB   : 0.4 (max=2)
TB covered     : 118/131 (Duplicate 13)
Guest Size     : 7269/7499 (Duplicate 230) bytes
Guest iCount   : 2143/2203 (Duplicate 60)
------------------------
Trace information:
------------------------
Thread 0:
Trace used: 17/32
  Id (C:Ex)  LoopCnt/  Count: ( Len) Accumulated Exit Cnt
   2 (1: 4)        0/     12: ( 1.0) 12 0 0 0 0
   3 (1: 3)       31/     35: (16.5) 4 0
     (......)
  24 (0: 1)        0/  35832: (16.0) 0 0 0 0 0 0 0 0 0 0 0 0 0 35832
  25 (0: 3)        0/  22661: (15.0) 0 0 0 0 0 0 0 0 0 355 355 21951
  26 (1: 2)     1736/   2013: ( 7.3) 277
  27 (0: 4)        0/     93: ( 2.4) 0 56 37
```

2.6 HQEMU Options

HQEMU provides several options for users to control the execution of HQEMU, such as printing debug information, profiling, and turning optimizations on/off. Options are passed via the LLVM_CMD environment variable. Multiple options can also be passed at the same time by separating them with space. For example:

```
$ export LLVM_CMD='option1=val option2=val'
$ export LLVM_CMD='-debuglv=debug -profile=basic,trace -threads=4'
```

To display HQEMU specific options, use the command:

```
$ export LLVM_CMD='-help'
```

To display all available options including HQEMU and LLVM builtin options, use the command:

```
$ export LLVM_CMD='-help-hidden'
```