Update on the BPF support in the GNU Toolchain

Jose E. Marchesi <jose.marchesi@oracle.com>
David Faust <david.faust@oracle.com>
Guillermo E. Martinez <guillermo.e.martinez@oracle.com>
Summary

• The port: bpf-unknown-none
• Miscellaneaus updates
• BTF support
• BPF CO-RE support
• New options for BPF features and ISA selection
• Support for BPF atomics
bpf-unknown-none

- Binutils port
- GCC backend
- GDB port
- Simulator
- Dejagnu board
Miscellaneous updates

- New xBPF instructions: SDIV and SMOD
- The GCC BPF backend can now emit DWARF.
- Adjustments to pacify the kernel verifier.
- Many bug fixes.
- We are finally on par with LLVM!
BTF support in GCC

- BPF Type Format: debug info format for BPF programs
- Now fully implemented in GCC
- Supported for any target via `-g btf`
  - Could be used to e.g. generate BTF for kernel directly
- Less general purpose than CTF
  - Tailored to BPF programs
Problem: BPF Portability

• BPF programs generally include kernel headers
  − Definitions of data structures
  − Not limited to userspace API, may be internal

• Two problems:
  − Internal kernel data structures regularly change
  − Might include other, target-specific headers

• Not portable between kernel versions!
Solution: BPF CO-RE

- Compile Once – Run Everywhere
- Designed by kernel and LLVM BPF folks
- Emit relocations annotating accesses to data structures:
  - Which instructions need adjusting
  - How to adjust according to structure definitions of local kernel
struct S *foo = ...;
char x = foo->data[3].c

/* do stuff with x */

ldxb %r3, [%r2+0x20]

...
BPF CO-RE in GCC

- Needs type information of records (BTF)
- ... also some unfortunate coupling with .BTF
- Implemented in the BPF backend
- CO-RE relocations emitted in .BTF.ext
- -mco-re
BPF CO-RE in GCC

Built-in marker for accesses to record

```c
#define _(x) __builtin_preserve_access_index (x)
char x = _(foo->data[3].x);
```

- Transparent: does not change semantics of program
- Overloaded: return type is type of expression
- Accepts statement-expressions...

```c
_((({ int x = foo->a;
    foo->b = x + foo->c;}));
```
BPF CO-RE in GCC

Attribute: applies to structs/unions

```c
struct S {
    int a;
    char c;
    struct T tea;
} __attribute__((preserve_access_index));
```

Any access to an attributed type will produce a CO-RE relocation.
GCC - BPF feature selection

• BPF has gained new instructions over time
  – Extra compare-and-branch operations (\(<, \leq\))
  – 32-bit ALU and 32-bit jump operations
  – New atomic operations

• New options in GCC to enable/disable generation of these instructions
  – -mjmpext, -mjmp32, -malu32

• Use to match features implemented by target kernel
GCC - BPF ISA version

- `mcpu=\{v1, v2, v3, latest\}`
  - V1: base eBPF with no extensions
  - V2: adds extra jump operations
  - V3: adds 32-bit ALU/jump operations
  - Default: latest, synonym for v3
- May combine with feature selectors
  - e.g. `-mcpu=v3 -mno-jmp32`
BPF Atomics
Contents

- Concurrency in eBPF.
- Linux Kernel: Atomics for eBPF.
- Changes in CGEN Framework.
- binutils-gdb: Atomic eBPF instructions.
- GCC: Atomic built-in functions.
More than one process/threads can access the \texttt{eBPF} store data to be processes by an \texttt{eBPF} program (map).

Synchronization mechanisms:

- Spinlocks ([https://lwn.net/Articles/779120/](https://lwn.net/Articles/779120/))
  - `bpf_spin_{lock,unlock}`
  - Reentrancy Pattern: Fine granularity!!
- Atomics instructions: ([https://lwn.net/Articles/838884/](https://lwn.net/Articles/838884/))
Linux Kernel: Atomics for eBPF 1/2

➢ Linux Kernel 2020.
  • https://www.kernel.org/doc/Documentation/networking/filter.rst

➢ eBPF Atomic operations:

\[
\text{BPF}\{\text{ADD, AND, OR, XOR, XCHG, CMPXCHG}\}
\]

"use the **immediate field** of the existing STX XADD instruction to encode the operation. This works nicely, **without breaking existing programs**, because the immediate field is currently reserved-must-be-zero, and extra-nicely because BPF_ADD happens to be zero"
Is this an issue in the original eBPF ISA design?

So we are expecting an OPERATOR!! (in the instruction's immediate field), but now we have the "extra encoding" i.e an OPCODE!! it represent what kind of atomic operation the VM will be execute.!!
Changes in **CGEN** Framework 1/2

➢ **Opcodes generation:** C structures for a particular target.
  - opcodes/bpf-opc.c
  - opcodes/bpf-desc.c
  - opcodes/bpf-desc.h
  - ...

➢ Make sure that **CGEN** is able to compute right structures for the "**eBPF** atomics encoding":

```c
(define-normal-insn-enum insn-atomic-op-le "eBPF atomic insn"
  ((ISA ebpfle xbpfle)) OP_ATOMIC_LE_ f-op-atomic
  ((ADD #x01)
   (OR  #x41)
   ...
   (CMP #xf1))

(define-normal-insn-enum insn-atomic-op-be "eBPF atomic insn"
  ((ISA ebpfbe xbpfbe)) OP_ATOMIC_BE_ f-op-atomic
  ((ADD #x01000000)
   (OR  #x41000000)
   ...
   (CMP #xf1000000)))
```
Changes in **cGEN** Framework 2/2

- Provides support to compute the mask and get values for constant fields where we placed the opcode:

  ```plaintext
  mask: 0xffffffff000000ff
  value: 0x000000e1000000db
  ;; xchg dw in L.E
  ```

- Fix a wrong calculation in *mask and value* when a field is defined in a long form with *an offset* is different to 0:

  ```plaintext
  bitrange: #(object <bitrange> 29 32 31 32 64 #t)
  mask: 0xffffffff
  value: simplify.inc:173:3: Error: Instruction has opcode bits outside of its mask.

  Fixed to:

  bitrange: #(object <bitrange> 29 32 31 32 64 #t)
  mask: 0xffffffff00000000
  value: 0xaa00000000
  ```

binutils-gdb: Atomic eBPF instructions 1/3

➢ Add atomic instructions: (dni) in CPU Description Language: bpf.cpu

(dais add (.if (.eq x-endian le) OP_ATOMIC_LE_ADD OP_ATOMIC_BE_ADD) ...) ... 
(dais cmp (.if (.eq x-endian le) OP_ATOMIC_LE_CMP OP_ATOMIC_BE_CMP) ...) 

➢ Add atomic instructions to simulator.

➢ Carefully internal revision and exhaustive regression test.
  • arm-linuxeabi, aarch64-linux, alpha-dec-vms, am33_2.0-linux, arc-linux-uclibc, avr-elf, cr16-elf, cris-elf, crisv32-linux, crx-elf, d10v-elf, d30v-elf, dlx-elf, epiphany-elf, fr30-elf, x86_64-linux, x86_64-w64-mingw32, x86_64-nacl, xgate-elf, xstormy16-elf, xtensa-elf, z8k-coff, z80-coff, ...

https://sourceware.org/pipermail/binutils/2021-August/117798.html
**binutils-gdb:** Atomic eBPF instructions 2/3

<table>
<thead>
<tr>
<th>Linux Kernel</th>
<th>Mnemonic</th>
<th>eBPF assembly Statements</th>
<th>eBPF decode Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPF_ADD</td>
<td>xadd(dw,w)</td>
<td>xaddw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 01</td>
</tr>
<tr>
<td>BPF_AND</td>
<td>xand(dw,w)</td>
<td>xandw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 51</td>
</tr>
<tr>
<td>BPF_OR</td>
<td>xor(dw,w)</td>
<td>xor dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 41</td>
</tr>
<tr>
<td>BPF_XOR</td>
<td>xxor(dw,w)</td>
<td>xxorw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 01</td>
</tr>
<tr>
<td>BPF_XCHG</td>
<td>xchg(dw,w)</td>
<td>xchg dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 01</td>
</tr>
<tr>
<td>BPF_CMPXCHG</td>
<td>xcmpeq(dw,w)</td>
<td>xcmpeq dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 00 00 00 01</td>
</tr>
</tbody>
</table>

Table 1. eBPF Atomic instructions in B.E.

<table>
<thead>
<tr>
<th>Linux Kernel</th>
<th>Mnemonic</th>
<th>eBPF assembly Statements</th>
<th>eBPF decode Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPF_ADD</td>
<td>xadd(dw,w)</td>
<td>xaddw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 01 00 00 00</td>
</tr>
<tr>
<td>BPF_AND</td>
<td>xand(dw,w)</td>
<td>xandw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 51 00 00 00</td>
</tr>
<tr>
<td>BPF_OR</td>
<td>xor(dw,w)</td>
<td>xor dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 41 00 00 00</td>
</tr>
<tr>
<td>BPF_XOR</td>
<td>xxor(dw,w)</td>
<td>xxorw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 01 00 00 00</td>
</tr>
<tr>
<td>BPF_XCHG</td>
<td>xchg(dw,w)</td>
<td>xchg dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 01 00 00 00</td>
</tr>
<tr>
<td>BPF_CMPXCHG</td>
<td>xcmpeq(dw,w)</td>
<td>xcmpeq dw [%r1+0x1eef],%r2</td>
<td>db 12 le ef 01 00 00 00</td>
</tr>
</tbody>
</table>

Table 2. eBPF Atomic instructions in L.E.
binutils-gdb: Atomic eBPF instructions 3/3

$ bpf-as --EB gas/testsuite/gas/bpf/atomic.s

$ bpf-objdump -dr a.out

a.out: file format elf64-bpfbe

Disassembly of section .text:

0000000000000000 <.text>:
  0: db 12 le ef 00 00 00 00 01 xaddw [%r1+0x1eef],%r2
  8: c3 12 le ef 00 00 00 00 01 xaddw [%r1+0x1eef],%r2
 10: db 12 le ef 00 00 00 00 41 xorw [%r1+0x1eef],%r2
 16: c3 12 le ef 00 00 00 00 41 xorw [%r1+0x1eef],%r2
 26: db 12 le ef 00 00 00 00 51 xandw [%r1+0x1eef],%r2
 28: c3 12 le ef 00 00 00 00 51 xandw [%r1+0x1eef],%r2
 30: db 12 le ef 00 00 00 00 a1 xxordw [%r1+0x1eef],%r2
 38: c3 12 le ef 00 00 00 00 a1 xxorw [%r1+0x1eef],%r2
 40: db 12 le ef 00 00 00 00 e1 xchgw [%r1+0x1eef],%r2
 48: c3 12 le ef 00 00 00 00 e1 xchgw [%r1+0x1eef],%r2
 50: db 12 le ef 00 00 00 00 f1 xcmpw [%r1+0x1eef],%r2
 58: c3 12 le ef 00 00 00 00 f1 xcmpw [%r1+0x1eef],%r2
GCC: Atomic built-in functions 1/4

➢ Fully compliant with:
  • https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html
  • https://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Atomic-Builtins.html

➢ Support to:
  • __atomic_fetch_\{add, sub, and, xor, or\}
  • __atomic_exchange
  • __atomic_compare_exchange_n
  • __atomic_<operation>_fetch
  • __sync_fetch_and_<operation>
  • __sync_<operation>_and_fetch
Defining atomics instructions in bpf machine description file:

- gcc/config/bpf/bpf.md:

```c
(define_insn "atomic_fetch_add<AMO:mode>"
  [(set (match_operand:AMO 0 "register_operand" ";=r")
    (unspec_volatile:AMO
     [(match_operand:AMO 1 "memory_operand" ";+o")
      (match_operand:AMO 2 "nonmemory_operand" ";0")
      (match_operand:AMO 3 "const_int_operand")]) ;; Memory model UNSPEC_XADD])

"" "xadd<mop>\t%1,%0"
```
Add constrains to place operands in specific register:

```
(define_register_constraint "t" "R0"
  "Register r0")
...
(define_insn "atomic_compare_and_swap<AMO:mode>_1"
  [(set (match_operand:AMO 0 "register_operand" "=t") ;; must be r0
      (unspec_volatile:AMO
        [(match_operand:AMO 1 "memory_operand" "+o") ;; memory
          (match_operand:AMO 2 "register_operand" "0") ;; expected
          (match_operand:AMO 3 "register_operand" "r") ;; desired
          (match_operand:SI 4 "const_int_operand")
          ;; success
          (unused)
          UNSPEC_CMPXCHG)])
  ""
  "\texttt{xcmp<mop>\textbackslash t,1,%3}\)"
```

 GCC: Atomic built-in functions 3/4
$ bpf-gcc -O2 /home/byby/oracle/src/gcc/gcc/testsuite/gcc.target/bpf/atomic-compare-exchange.c

$ bpf-objdump -dr a.out

```
a.out:  file format elf64-bpfle
Disassembly of section .text:

0000000000000000 <foo>:
  400900:  18 02 00 00 f8 11 00 00  lddw %r2,0x0011f8
  400908:  00 00 00 00 00 00 00 00  1
  400910:  18 03 00 00 d8 11 00 00  1
  400918:  00 00 00 00 00 00 00 00  1
  400920:  18 01 00 00 e0 11 00 00  1
  400928:  00 00 00 00 00 00 00 00  1
  400930:  79 15 00 00 00 00 00 00  1
  400938:  79 34 00 00 00 00 00 00  1
  400940:  b1 50 00 00 00 00 00 00  1
  400948:  db 42 00 00 00 00 00 00  1
  400950:  b7 06 00 00 00 00 00 00  1
  400958:  1d 50 01 00 00 00 00 00  1
  400960:  b7 06 00 00 00 00 00 00  1
  400968:  57 06 00 00 00 00 00 00  1
  400970:  5b 06 01 00 00 00 00 00  1
  400978:  7b 01 00 00 00 00 00 00  1
  400980:  79 35 00 00 00 00 00 00  1
  400988:  61 14 00 00 00 00 00 00  1
  400990:  b7 40 00 00 00 00 00 00  1
  400998:  c3 52 00 00 01 00 00 00  1
```
printf ("Thanks a lot for coming!!");