Adventures with LLVM in a magical land where pointers are not integers

David Chisnall

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 and FA8750-11-C-0249. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

What is a pointer?

- Conventional flat-memory architectures: a number indicating an address

- C requires: An value indicating an object and an offset that permits arithmetic

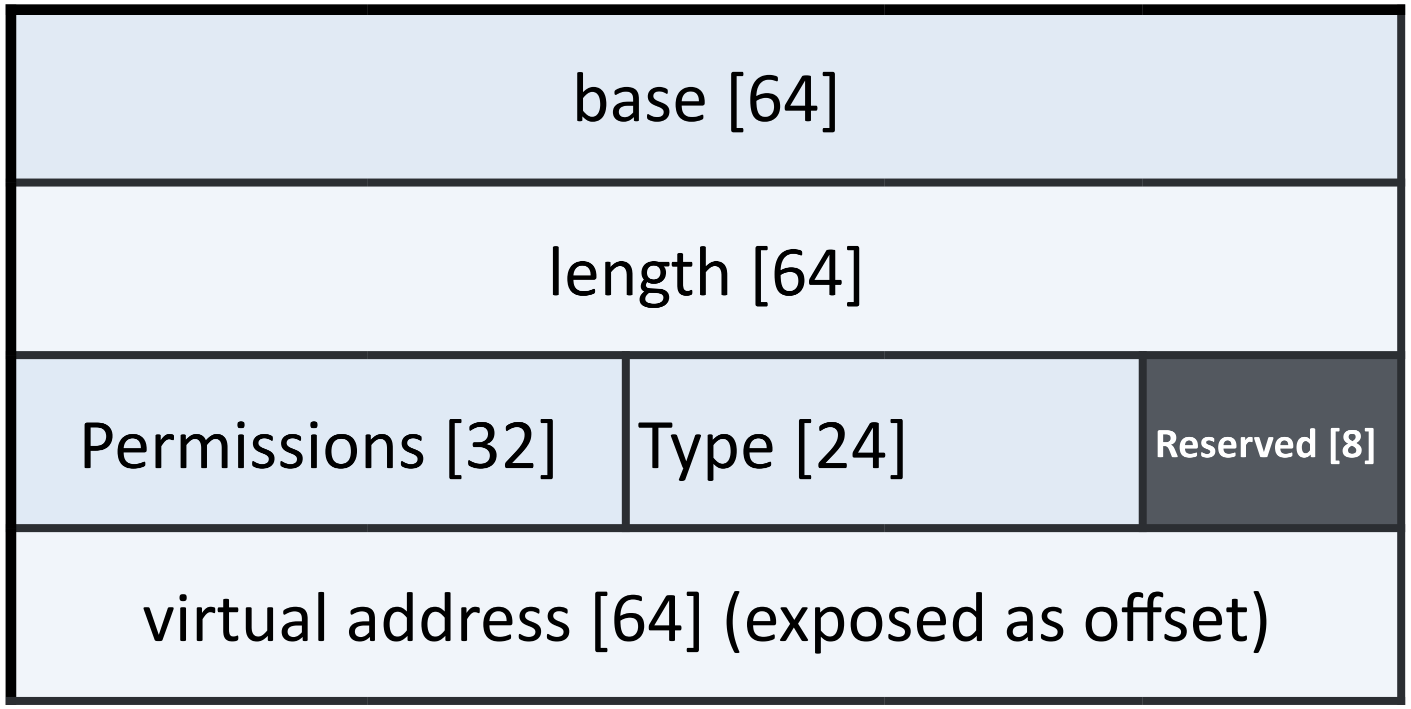
- People who write C require: Stable comparisons between pointers to different objects, unions of integers and pointers, other crazy stuff…

Fat pointers

- Fat pointers are pointers plus bounds information.
- Often implemented in software (e.g. Cyclone)
- Ours also have permissions.

Pointers in our processor

Memory capabilities: Atomic values identifying *and granting rights to* a region of memory.



base [64]

length [64]

Permissions [32]

Type [24]

Reserved [8]

virtual address [64] (exposed as offset)

Actually, it’s a bit more complicated…

- Some pointers are 64-bit integers (implicitly capability-relative).
- Some are memory capabilities.
- Some compilation units use both!
- Some want the stack to be a capability!

CHERI pointers in LLVM

	Conventional	Capability
Address space	0	200
Size	64 bits	256 bits
Round-trips via integer?	Yes	Sometimes…

Pointers in LLVM

- Strongly typed in IR.
- Can be converted (possibly lossily) to and from integers with `inttoptr` / `ptrtoint`
- All typesafe arithmetic should be done with GEPs
- Casts between address spaces with `addrspacecast` (added after we started, made life a lot easier!)

Except in the back end…

- `iPTR` is the value type for pointers.
- Back ends tell `SelectionDAG` which integer type should be used for pointers (oops!)
- Lots of pointer arithmetic done in `SelectionDAG` using normal arithmetic nodes

And a bit in the middle…

- Some optimisers assume that pointers are integers.
- Some assume that they know the representation of pointers.
- Most of these are easy to fix
  - Some by not running them
  - Some by teaching them that $2^{\mathrm{sizeof(ptr)}}$ does not give the size of the address space!

LLVM for CHERI

- Lots of changes throughout.
- Currently 13K lines of diff (4K more in clang).
- Includes 5K in the MIPS back end.
- Includes changes to allow `alloca`s in non-zero AS (only one stack AS per module!).

Size doesn’t imply range!

- Added methods to DataLayout that expose the *range* of a value separate from its size.
- CHERI pointers are 256-bits, with a 64-bit range.
- Call these in 20 places in optimisations (more on every merge from upstream)

Fixing SelectionDAG

- Added three new DAG nodes: `PTRTOINT`, `INTTOPTR`, `PTRADD`
- Added `iFATPTR` value type
- Added new `SelectionDAG` method
- Made 40 places use it! (also simplified a load of copy-and-pasted code

Some issues

- `PTRADD` is not symmetrical (pointer on left, integer on right)
- Existing DAG folding doesn’t handle it
- Works, but generates some inefficient code

Fixing pointer adds

```
SDValue SelectionDAG::getPointerAdd(SDLoc dl, SDValue Ptr, int64_t Offset) {
  EVT BasePtrVT = Ptr.getValueType();
  if (BasePtrVT == MVT::iFATPTR) {
    const TargetLowering *TLI = TM.getSubtargetImpl()->getTargetLowering();
    // Assume that address space 0 has the range of any pointer.
    MVT IntPtrTy = MVT::getIntegerVT(
        TLI->getDataLayout()->getPointerSizeInBits(0));
    return getNode(ISD::PTRADD, dl, BasePtrVT, Ptr, getConstant(Offset,
          IntPtrTy));
  }
  return getNode(ISD::ADD, dl, BasePtrVT, Ptr,
                 getConstant(Offset, BasePtrVT));
}
```

```
-  Ptr = DAG.getNode(ISD::ADD, dl, Ptr.getValueType(), Ptr,
-                    DAG.getConstant(IncrementSize, Ptr.getValueType()));
+  Ptr = DAG.getPointerAdd(dl, Ptr, IncrementSize);
```

Silly fixes

- AsmPrinter uses `EmitIntValue()` instead of `EmitZeros()` to write constant null pointers.

- `IRBuilder::getCastedInt8PtrValue()` needs a version that takes an address space.

- Lots of code in clang thinks `i8*` in AS 0 is a generic pointer type.

Conclusion

- LLVM IR is perfectly happy with fat pointers.
- LLVM code… nearly is.
- Needs an in-tree target with regression tests.