When 3 Memory Models Aren’t Enough
Porting VMS to x86 using LLVM

• Began two years ago
• Host LLVM on OpenVMS Itanium
• Using older 3.4.2 due to Itanium C++
• Convert our backend’s IR to LLVM IR
• Reuse existing frontends with almost no change
• Currently booting on VirtualBox & KVM
OpenVMS Cross-compilers

- Continue with current GEM-based frontends
- Use open source LLVM for backend code generation
- Create internal representation (IR) translator
Calling Standard & Memory Model

• Based on the AMD ABI
• Add OpenVMS changes
  − Arg count on every routine
    • Added new intrinsic and function attribute
    • Passed via AH
  − Exception handling / unwind information
    • Asynch unwind directives for prologue/epilogue
    • Others are doing current work and we’ll piggyback
  − ELF section vendor-specific flags
  − OpenVMS memory model
OpenVMS Memory Model

- Lots of legacy 32-bit VAX interfaces in OS
- Two sizes of pointers (32 & 64)
- Stack resides in 32-bit address space
- Static data resides in 32-bit address space
- Heap either in 32 or 64 bit address space
- Code in 64-bit address space by default
- But “routine addresses” must be 32-bit
OpenVMS Memory Model

• PIC only
• Since code may be very far from any static data, all data lots must be through the GOT. No PC-relative offsets allowed including literal pool.
• Since routines in other sections may be very far way, all routine calls must be through the GOT
• To achieve 32-bit routine values, the linker creates trampoline routines in 32-bit space
Current status

- Most GEM IR maps easily to LLVM
- G2L 26,000 lines of C++
- Static variable initialization is very different
- Aliasing variables for BLISS is a challenge
- DWARF is partially done, waiting for update to native LLVM
- Continue to use GEM’s util routines for command line, listing files, etc
- All LLVM changes total about 500 lines
- No optimizers for cross-compilers
- Work underway to native bootstrap to current LLVM by cross-compiling on Linux and cross-linking on OpenVMS Itanium for eventual execution on OpenVMS x86
- Followed by a VMS-ification of clang to use as our C++ compiler