vms Software

State of the Port to x86_64 July 2017

July 7, 2017

Update Topics

- Executive Summary
  - Development Plan
  - Release Plan
- Engineering Details
  - Compilers
  - Objects & Images
  - Binary Translator
  - Early Boot Path
    - Boot Manager
    - SYSBOOT
    - Virtual Machines
    - Condition Handling
  - Dump Kernel
  - Software Interrupt Services (SWIS)
  - Drivers and Device Management
  - XDELTA-lite
  - Cross Build
  - Conditional Code Verification

Executive Summary - Development

The Development Plan consists of five strategic work areas for porting the operating system to the x86_64 architecture.

- OpenVMS supports nine programming languages, six of which use a DEC-developed, proprietary back-end code generator on both Alpha and Itanium. We are creating a converter to internally connect these compiler front-ends to the open source LLVM back-end code generator, which targets x86_64 as well as many other architectures. (The other three compilers have their own individual pathways to the new architecture.) LLVM implements the most current compiler technology and associated development tools and it provides a direct path for porting to other architectures in the future.
- Operating system components are being modified to implement the industry standard x86_64 calling conventions and executable image format. We moved away from proprietary conventions and format in porting from Alpha to Itanium so there is much less work in these areas in porting to x86_64.
- As in any port to a new architecture we are implementing a number of architecture-defined interfaces that are critical to the inner workings of the operating system.
- OpenVMS is currently built for Alpha and Itanium from common source code. We are adding support for x86_64 so all conditional assembly directives must be verified.
- The boot manager for x86_64 has been upgraded to take advantage of new features and methods which did not exist when our previous implementation was created for Itanium. This will streamline the entire boot path and make it more flexible and maintainable.

Executive Summary - Release

The Release Plan includes multiple stages.

- Once we achieve First Boot we will work towards an Early Adopter Release. We define First Boot as being able to bring the system up to the point of logging in and successfully executing a DIRECTORY command. That may not sound like much but most of the architecture-specific aspects of the operating system must be working as well as being able to compile, link, load, and execute non-privileged code.
- The Early Adopter Release is for our partners. There will be some system components, most layered products, and analysis tools not in place or not yet complete but it will be enough for partners to start verifying their code on some x86 systems.
- There may be another less-than-complete system release depending on our own testing and feedback from our partners.
- The full production General Release will be the complete system as it ships today on Alpha and Itanium.

Compilers

Reminder: The **GEM-to-LLVM** (G2L) converter translates GEM tuples (GEM's internal language) into LLVM's internal language.

As work progresses on both C and BLISS compilers, the converter becomes more streamlined and functional. The GEM and LLVM internal representations are not one-to-one mappings so there is a fair amount of invention in the converter to make some translations. G2L generates LLVM bitcodes which, if needed, can now call GEM's internal library routines. This will make many things easier as development proceeds.

Some argument-passing constructs are particularly challenging. For the **VSI C** compiler changes are being made to the front-end in order to get the correct information in place to pass along to LLVM. We periodically compare VSI C with Clang since we intend to offer both on x86. For a given source, the two must produce functionally equivalent objects.

There is now G2L support for string and bit manipulation needed by **BLISS.** A little more work in these areas and we will be ready for the next round of serious BLISS testing in the system build.

XMACRO compiler work continues on 1) translating operand formats and 2) instruction substitution. Condition code handling is complete. Next up is PSECT mapping and label processing.

Objects & Images

Linker – The linker now creates OpenVMS x86 ELF images loadable by the Boot Manager. Many details are in place compared to just a few weeks ago. Also, we will not need “the big SYSBOOT single image” we used in the Itanium port for many months. The linker will be able to handle execlets and the base image from the beginning when we are in position to build the entire system. A remaining category of work involves the list of data items the compiler frontends must pass through LLVM to get into the object file for the linker, for example source module name and version number.

Librarian – complete

ANALYZE / OBJECT and /IMAGE – complete

Image Activator – Design changes are complete and implementation details are in progress.

Stack Unwinding – Investigation is underway. We will be using new support routines since x86 is sufficiently different from Itanium.

Binary Translator

Reminder: The Alpha-to-x86 binary translator parses an Alpha image and generates x86 instructions via LLVM. The prototype translator runs on linux and Windows/cygwin.

Many experiments have been conducted to evaluate memory mapping alternatives and RTL jacketing techniques.

Standard benchmark tools such as dhrystone and primes have been used along with a variety of C, FORTRAN, BASIC, and COBOL programs, for example a FORTRAN version of Adventure. Each test provides more insight into the variety of information the translator must support - image initialization, string operations, floating point, use of math libraries and language RTLs, to name just a few.

The work to date has focused on analyzing the images running in emulation, using simh as a base. Work recently started on creating LLVM internal representation to generate the x86 code.

NOTE: The Itanium-to-x86 translator work will follow the Alpha-to-x86 work.

Early Boot Path

Reminder: The early boot path is being streamlined and modernized to be more suitable for the UEFI / ACPI environment. The functions of the former OpenVMS primary bootstraps (VMB / APB / IPB) have been merged into the **boot manager** VMS_BOOTMGR and the OpenVMS **loader** SYSBOOT. The major goals of the new work for x86 are:

- Always boot using memory disk
- Never write another boot driver
- Eliminate the need for updating the primitive file system

Many I/O improvements have been made to enhance the Boot Manager, SYSBOOT, and debugger interfaces, both command line and graphical interfaces.

Refinement of network booting continues and we always test all aspects of the boot process on physical systems and **virtual machines.**

We discovered that virtual machines vary in their emulation of environment variables so we created a file-based implementation of our own to guarantee uniform persistence over reboots in all environments.

We are organizing the boot flags into phases (SYSBOOT, EXEC_INIT, SYSINIT), each with verbosity level options, to better manage the informational output during normal system startup and system debugging.

Early Boot Path, continued

Reminder: The x86 **SYSBOOT** code is initially being developed like the Boot Manager - in Microsoft C, built with Windows Visual Studio, then transferred to the appropriate platform for debugging. We have also occasionally compiled the SYSBOOT code on OpenVMS with the new x86 LLVM-based VSI C compiler so there are no surprises when we eventually move to building everything on OpenVMS.

Work continues on filling in the infrastructure needed later on in the system.

Work has started on moving the current SYSBOOT work from Visual Studio to the OpenVMS cross build. This is a big job but given the advancing states of the VSI C compiler, linker and librarian, we are now in position to start the transition.

The lack of “load/store physical” on x86 caused some of the traditional OpenVMS techniques for memory management to be very awkward, so there was some rethinking and prototyping of a slightly different approach for x86. By taking advantage of the boot-time UEFI physical memory map, we were able to reduce the complexity and likely improve performance a little.

Next up for SYSBOOT is loading and slicing x86 execlet images.

Early Boot Path, continued

Reminder: Most of the code in the x86 **condition handler** EXCEPTION.EXE will be in C. Since this code has few dependencies on the rest of the system, it presents another opportunity to take advantage of Visual Studio and the boot environment. Normally, interrupts are fielded by SWIS and sent off to the appropriate handlers, many of which go through EXCEPTION. We are “borrowing” that initial vectoring from SWIS and temporarily putting it in the new x86 EXCEPTION to facilitate some early debugging. This image can be loaded and started directly by the boot manager. With a little creativity we can force the EXCEPTION code into the various condition handlers and do a lot of debugging natively on x86 now.

The design is complete and has been reviewed. Data structures definitions are now in place and implementation is in progress. We have simplified the OpenVMS dispatch table to be more in line with the architectural definition.

Dump Kernel

Reminder: During normal system boot a second, minimalist OS instance is loaded into memory but not started. When the system goes down the Primary Kernel gathers and stores information; BUGCHECK then notifies the Boot Manager to boot the Dump Kernel which writes the crash dump using the runtime driver and finally initiates a shutdown.

The Dump Kernel is nearing completion. In addition to standard system disk testing, shadowed system disks and Dump Off System Disk (DOSD) have also been tested. Error log access and the dump contents have been verified. In fact, it has been used to analyze a real problem!

The Dump Kernel now boots entirely from the memory disk that is read in when the Primary Kernel boots, thus it boots very quickly.

Debugging of the Dump Kernel has been done on Itanium. The final verification will wait until we have other needed system components running on x86, for example run-time device drivers.

Software Interrupt Services (SWIS)

Reminder: SWIS is the component that provides a VAX/Alpha-like interface based on the current platform's features, so that the rest of the system continues with its assumptions in certain areas, such as

- 4 modes, each with its own stack
- per-process ASTs in each mode
- Software interrupts for the lower 16 IPLs
- context switching
- system service dispatching

The detailed design was done in multiple iterations, is now complete, and has had extensive review.

Implementation is in progress and the new code is included in the cross builds.

Much of the work requires detailed coordination with other areas of the system, such as memory management, condition handling, and image activation.

Drivers and Device Management

For the second time we upgraded our sources to the latest **ACPI** Component Architecture version. In addition to eventually being on x86, the new ACPI code will be in the next Itanium release.

We are currently evaluating all existing OpenVMS device management code to see how best to improve it in the x86 environment, especially in the areas of USB support and device recognition.

Paravirtualization: The VIRTIO protocol is used by multiple hypervisors for enhanced performance.

The design and implementation of the OpenVMS driver continues.

- SCSI/FIBRE IOGEN configuration code - complete
- Initialization routines - complete
- Run-time routines implementation - in progress

XDELTA-lite (a.k.a. XLDELTA)

In the two previous OpenVMS ports we did not have a debugger until very far along in the project. Nevertheless, 'print statements' took us a long way in the early system work.

For x86, we decided to evaluate how much we could do earlier, even in a stripped down fashion. This experiment has been a great success. We created a prototype, from scratch, that we call XDELTA-lite. Written in C and a little x86 assembler, it can

- set, stop at, and continue from breakpoints
- single step instructions
- examine/deposit memory
- examine/deposit registers

That may not sound like much but it is way ahead of where we have been in the past at the same relative point in the project. XLDELTA is built with the VSI C compiler and the linker and will be linked into SYSBOOT. We also created a primitive exception handler which is needed by the debugger. Note: XLDELTA itself was debugged in user mode on linux.

XLDELTA will be used in debugging the remaining work in SYSBOOT as well as some of the early work in exception handling and SWIS.

Cross Build

The X-Build environment (building x86 on Itanium) is rapidly taking shape as we create the procedures, integrate the necessary tools, verify the conditionalized code and data structures, and add code specific to x86.

Integrating something new at this early stage, for example an updated compiler, usually depends on “valuable error messages outweighing the noise” (those things we already know are incomplete). The “signal to noise” ratio is gradually increasing as we address the things the x-build reveals are missing or wrong.

We recently added the first x86 assembler source modules and the processing of header files they need, a major step in advancing the overall process.

Conditional Code Verification

Reminder: There are roughly 800 code modules needed for First Boot that contain conditionalized code, that is code compiled differently for one or more platforms. Each of these source code modules must be checked to verify the conditionals produce the correct code for all three architectures – Alpha, Itanium and x86. It involves MACRO-32 source, BLISS source, C source, structure definition language (SDL) source, C header files, BLISS require files, and MACRO include files.

This work is 85% complete; verification of the remaining modules will happen as part of the architecture-specific development work in areas such as memory management, SWIS, conditional handling, and device management.

Note that since this work immediately becomes part of the Alpha and Itanium system builds, the conditionals affecting those two platforms get tested in the normal regression testing now.

Once we get beyond First Boot, there are another 2000+ modules that need conditional verification.

For more information, please contact us at:

RnD@vmssoftware.com

VMS Software, Inc. • 580 Main Street • Bolton MA 01740 • +1 978 451 0110