OpenVMS Technical Journal V7

The Development Of A High Performance VAX 6000 Emulator

Dr. Robert Boers, CEO, CTO, Software Resources International S.A.

Overview

Software Resources International (SRI) develops commercial emulators for VAX hardware. Designed as a hardware abstraction layer (HAL), a VAX emulator is essentially a software mathematical model of VAX hardware. If the HAL is accurate enough, the original VAX operating systems and applications can be executed on it. This enables the use of unmodified VAX software on any platform for which such a HAL is available, thereby avoiding the cost and risks of using aged VAX hardware. This article describes the development of a HAL for the large VAX SMP systems, the ultimate performance step in replacing existing VAX systems.

Introduction

Since the advent of commercial computing, users have sought to simultaneously take advantage of new hardware advances while preserving their existing applications. Computer manufacturers like Digital Equipment Corporation (DEC) strove for backwards compatibility, but keeping systems compatible at the hardware level limited the innovation that could be applied as larger word lengths, more address space, and more sophisticated operating systems became available. When applications are written in a higher-level language, the amount of required changes can be limited by compiler compatibility, but the source code has to be available. Translating binary application code requires translated system calls[1] and is limited to application code.

Hardware Emulation

The ultimate way to support legacy software is hardware emulation. The computer's CPU interprets binary instructions. It does not matter whether this interpretation is done directly by hardware or by software routines, as long as the ultimate result is same. This is simple in principle, but in reality this solution is complex.

[1] Examples of such translators are the FX32 I86-to-Alpha converter or the VAX-to-Alpha and Alpha-to-Itanium converters for OpenVMS applications.

A computer contains not only a CPU, but also peripherals, interconnect hardware, clocks, and many other elements, each of which require a precise representation. The exact hardware component functionality must be recreated in software models, as well as the correct interaction in time between those component models. The result is an accurate software model of computer hardware, or a hardware abstraction layer (HAL), perceived by the legacy software as the original hardware system.

While exact hardware emulation is difficult to achieve, the rewards are significant. If the HAL is accurate enough, the hardware diagnostics and hardware verification tools can be used for testing. The emulator becomes independent of a particular legacy operating system; hence in principle it can run any code that ran on the original hardware. When the HAL is structured as a library of components representing hardware elements, it can be configured in real time into any system configuration for which the emulated elements are available.

Note that HALs can be found in most operating systems, where they make the work of an operating system designer easier by masking variations of the underlying CPUs, minor hardware variations, and so forth. In the case of a hardware system emulator, the HAL is not a thin layer of code requiring few system resources, but a collection of emulated system components with much higher complexity. Such components — and in particular the emulated CPU — require a large amount of computer horsepower. Usable emulators of computer hardware are only feasible due to the much higher performance of the systems on which such emulators are executed, as compared to the original hardware.

The emulation ratio (that is, the number of instructions of the host system required to implement one instruction of the simulated system), is a good metric by which to understand the cost of system emulation. Each emulated component contributes in its own way to the overall performance, so there is no uniform ratio for all components. In practice, we use VUPs (VAX Units of Processing) to compare the performance of our CHARON-VAX emulator products. Tests show that the average emulation ratio measured this way is determined mainly by the efficiency of the simulated components and the host CPU instruction set. The compatibility of host system floating point formats with the legacy hardware influences mathematical performance. A fast thread switching capability in the host system and low memory latency are also important, since emulated CPU components are usually represented as host memory locations.

For example, a VAX 4000 model 90 uses a master clock of 71 MHz for 32 VUPs, while a basic version of CHARON-VAX on a 3 GHz P4 yields approximately 20 VUPs, resulting in an emulation ratio of about 60. Unfortunately, because of the sequential nature of the CPU emulation, executing a HAL on a host system with a larger word length (for instance, emulating a 32-bit VAX system on a 64-bit host) does not provide additional performance, while a shorter host word length carries a heavy performance penalty.

20 VUPs is an acceptable performance for home use and low-end commercial VAX emulation, and in addition to several freeware and hobbyist implementations, SRI's low-end CHARON-VAX/XM product is a bestseller. However, the majority of the business-critical VAX systems still in operation have much higher performance requirements. Single-CPU VAX systems deliver up to 50 VUPs and VAX SMP systems (such as the VAX 7860) can deliver close to 300 VUPs. Replacing such systems with a simple interpretive emulator would require a host system with a clock frequency of 40–50 GHz, which is simply not feasible.

Advanced CPU Emulation (ACE)

In 2002, Software Resources International developed advanced CPU emulation (ACE), which allows us to break the 20 VUPs barrier on currently available I86 architecture using a method similar to the way hardware CPUs use multiple pipelines and look-ahead optimization to improve performance.

Each CPU is represented by two process threads. The first thread analyzes a VAX page of instructions and data, calculates potential future page references, and reorders execution instructions to optimize the use of the host system.[2] As a result, a much larger VAX page is created and buffered.

The second thread executes the processed page, which requires several refinements. Page processing involves a delay that can be intolerably long when driver code is executed, so ACE includes the original VAX page code in the extended page and uses it until the other thread has completed processing. Self-modifying code (for instance, in Oracle RdB) is handled by trapping write operations to VAX instructions, and the instruction sequence is reoptimized. It took a year to tune the prefetch and buffering mechanisms to reach consistently high performance for all VAX system architectures. AXE (the original VAX CPU verification test suite) was used to verify the correct operation of ACE.

The ACE technology allowed us to develop a high performance single-CPU VAX emulator that can deliver up to 80 VUPs on I86 platforms. It is interesting to note that, in spite of their lower CPU frequency, the AMD Opteron-based systems perform better than the Intel Xeon platforms. Opteron CPUs have an on-chip memory controller running at the CPU clock speed, which provides a very low memory latency. They also excel in fast floating point processing. The ACE- based products (sold under the CHARON-VAX *Plus* family name) range from the MicroVAX 3600 (running at 30 times its original speed) to the 512 MB VAX 4100-108, thereby covering most single-CPU VAX systems that are still in operation.

We had exceeded single VAX CPU performance by a big margin. The last hurdle was to reach performance sufficient to replace VAX hardware of any performance range. Following the VAX hardware development history, we had to emulate an SMP VAX system. This has implications for the emulator host system specifications, as SMP system emulation requires running a CPU emulator "*N*" times. The CPU emulation component, driven by the operating system that it is executing, takes all the resources it can get. Hence each emulated CPU requires a dedicated host CPU. The other emulator components cause a much lower load.[3] Therefore, emulating for instance a three-CPU VAX system requires a four-CPU host system.

We started developing a VAX SMP emulator at the end of 2003. At that time, there were not many four-way or eight-way CPU systems on the market, and nearly all had clock frequencies far below that of a common 1 GHz single-CPU system. We needed to emulate at least 3–6 VAX CPUs to make a significant step forward.

Modifying the ACE Design

The functionality of a VAX emulator depends on the VAX model that it represents, but its performance depends on the host system. As shown in the MicroVAX 3600 example, emulating a VAX does not mean it can only run at its "native" speed. For our VAX SMP implementation, we narrowed our choice to either the VAX 6000 or the VAX 7000 family. The VAX 7000 LSB backbone and its attached buses would be much more complex to emulate, so we chose the VAX 6000, which has a well-documented XMI bus. Its synchronous operation and its fixed number of slots (14, of which 10 are available for CPUs or peripheral controllers) allows straightforward configuration. While one team developed the SMP CPU implementation on a skeleton XMI bus, another team focused on the memory subsystem, Ethernet, and disk/tape controllers.

The original ACE implementation was synchronized with its VAX code execution, but with multiple CPUs the page reordering delays were too long to be acceptable. Properly synchronizing the emulated CPUs with the XMI bus involves strict timing constraints. The solution was the implementation of an asynchronous ACE mechanism. As a nice side effect, this new mechanism reduced emulated VAX interrupt latency. The field test experience was so positive that our single-CPU emulator products adopted this method as well.

[2] This rather complex part is host system–specific; the rest of the emulator is fully portable.

[3] Consequently, our high performance single-CPU VAX emulators alsorequire a dual-CPU host system.

The VAX 6000 emulator presented unique challenges. For example, every node on the XMI bus is capable of setting up an interrupt request, and each of them is able to respond to that interrupt request, becoming an interrupt server. Unlike single-CPU systems, where the CPU is mostly in control, peripheral nodes can specify which nodes are eligible to become interrupt servers.

The SMP VAX emulator project borrowed components from the existing products but produced many improvements as well. Asynchronous ACE, the most complex component, took only a year to develop by testing on an existing MicroVAX emulator. The XMI MSCP controller was developed the same way. In the SMP project we rewrote most of the emulator core, establishing a new code base for all our emulator products.

Implementing Multi-CPU Emulation

Once the XMI bus behaved properly, the VAX 6610 (single-CPU) emulator posed few problems. However, the step up to multi-CPU emulation required more design work. When multiple CPUs execute the same code, they can share the same VAX page that is currently being analyzed for reordering. For efficiency, each CPU has its own page processor thread. When a thread starts a page-reordering, other threads should back off. If a location is written in a page, its modified version should be invalidated, but another CPU might still be working on it. This required the development of an additional level of synchronization for the VAX page processing.

The solution results in heavily-threaded application code,[4] so the host operating system must be capable of efficient thread switching. To avoid catastrophic failure, CHARON-VAX constantly watches its resources. If there are not enough system resources available to run the VAX page analysis process, it slows down in a ”safe” mode to prevent a brutal interruption of the services that the VAX operating system needs to keep running.

A well-functioning VAX 6000 SMP emulator prototype was not yet the result we wanted. Our goal was to create a product family, CHARON-VAX/66x0, capable of replacing all large single-CPU and SMP VAX systems, including large configurations with multiple Ethernet controllers, disks, and several gigabytes of memory, with higher performance.

Our emulation of the standard VAX 6000 hardware was too limited. The largest VAX 6000 memory module was 128 MB, and the KDM70 disk controller supported only eight devices. With only 14 slots to use, and using up one for each VAX CPU, memory board, disk controller, or Ethernet adapter, we could not create, for instance, a six-CPU, 2 GB VAX with 200 disks.

Designing New “Hardware”

Initially we emulated the VAX XMI-to-BI adapter for additional peripheral support (we actually built a prototype). But a more elegant solution emerged. We decided to become “hardware engineers” and design higher-density memory boards and larger disk controllers than the VAX 6000 hardware ever had. We estimated how 256-, 512- or 1024-MB VAX 6000 memory boards would have looked if Digital Equipment Corporation had designed them[5] and we emulated the boards.

We designed the XMI KDM70 disk controller the same way, implementing the MSCP protocol, through which it communicates with the VAX operating system. MSCP devices are autonomous units that inform the operating system of their capabilities. By modifying its protocol responses, we made the controller capable of supporting several thousand disk drives, but because each drive requires a certain amount of buffer space, we limited the number to a more practical 256. Also, we added support for SCSI drives, which the VAX sees as MSCP drives. Similarly, we added support for SCSI tape drives to the modified KDM70.[6]

[4] The CHARON-VAX/6660 emulator uses about 30 parallel threads, including two for each emulated CPU.

[5] If they were designed, they never became products.

[6] Data-only, because the VAX 6000 does not know how to boot from a TA tape drive.

To our delight, OpenVMS accepted our modifications without complaining or requiring new drivers. In this way, we produced an SMP VAX emulator that behaves like a 3.5 GB VAX 6000. (The last 0.5 GB is occupied by the XMI I/O space.) For the current products, the emulated memory is limited to 2 GB, and it is easy to configure. You specify the memory size, then the emulator calculates the number and size of the memory boards for the four preallocated XMI memory slots. Even in a six-CPU VAX configuration with four memory boards, four XMI slots remain. These are typically used for one KDM70 and three Ethernet adapters. Each drive can be 8 GB or larger, and we have not seen a user exceed the limit of 256 drives, but a second controller could be configured at the expense of one Ethernet adapter.

The result is a flexible product that can provide fast one-to-six CPU VAX emulation on suitable hardware. Using four 275 dual-core Opteron CPUs in a four-way DL585 (effectively eight cores), the emulator delivers nearly 500 VUPs. We tried to emulate a seven-CPU VAX 6000-670 on an eight-way server, but there was no LMF key for that unusual number of VAX CPUs!

Transfer Licenses

A non-technical aspect of VAX emulation should be noted, involving the legal right to run a licensed VAX operating system and layered products. A few years ago, after we passed the original hardware certification tests, Compaq established transfer licenses for CHARON-VAX, which authorize the transfer of an existing OpenVMS version and specific listed layered software products to the CHARON-VAX emulator, whereby the existing LMF keys can be copied. The transfer licenses have order numbers and can be obtained from HP.

If a user upgrades from a small hardware VAX to a CHARON-VAX system providing a larger VAX system, the OpenVMS or layered products license units copied from the original system are not sufficient. As more customers take the opportunity to consolidate several hardware systems in one powerful emulator, this problem will become more common. To resolve this, we have an agreement with HP to provide the base transfer licenses and the operational licenses for OpenVMS, DECnet, and clustering, depending on the configuration of CHARON-VAX/66x0 products.

The development of three generations of VAX emulators has given us insight into how to design commercial emulators. The CHARON emulator core and our approach to emulator design are not restricted to VAX emulation. Our legacy product, CHARON-11, is selling in increasing numbers. We also implemented a prototype of an HP 3000 emulator, but we have not yet pursued product development. The emulation technology can also be used to economically replace embedded custom computers, and we have been approached by companies to do so. Concerning the CHARON product family, after our successful implementation of 16- and 32-bit systems emulation, we have started work on emulating 64-bit systems, and a prototype booted OpenVMS/Alpha succesfully. Stay tuned.

For more information:

For more information on Software Resources International or the CHARON-VAX family, visit our website at www.softresint.com or contact us at the address shown below. Dr. Boers can be reached via e-mail at: r_boers@softresint.com.

Software Resources International S.A.
Ch. DuPont-Du-Centenaire 109
1228 Plan-les-Ouates
Switzerland
Telephone: (41) 22 794 1070
FAX: (41) 22 794 1073
www.softresint.com