OpenVMS Technical Journal V10

Strategies for migrating from Alpha and VAX systems to HP Integrity server systems on OpenVMS

Robert Gezelter, CDP, CSA, CSE, Software Consultant

Overview

The ideal method for migration to OpenVMS on the Intel® Itanium® architecture is to recompile, relink, and requalify. For many users, this strategy has been highly successful. Many applications, each comprised of hundreds of thousands of lines of source code, have been ported virtually overnight.

Other organizations face more challenging circumstances. Restricted budgets, operational commitments, dependencies on layered products, staff shortages, the sheer number of applications, and the size of the aggregate source bases make the "all at once" strategy infeasible.

OpenVMS provides unique capabilities that allow an organization to assimilate HP Integrity servers in a phased, structured, extremely low-risk approach. This strategy decouples the different phases to the maximum extent possible, permitting the assimilation process to proceed transparently to users, with minimal disruption to operations and minimal business risk.

OpenVMS has clustering and image translation capabilities that offer end-user management and technical staff a unique degree of flexibility. In the past, this flexibility has enabled OpenVMS to confound critics by enabling the assimilation of Alpha processors and, more recently, Intel Itanium processors. The same capabilities that enabled the rapid assimilation of radically different hardware architectures by the OpenVMS Engineering organization itself are fully available to independent software vendors and end user

Introduction

Corporate Information Technology (IT) is a complex choreography of many interacting assets, including people, hardware, packaged software, and locally developed or maintained software. In today's environment, where IT has become the backbone of the enterprise, corporate IT is, in effect, committed to providing access to a wide variety of systems on a continuous basis, 24 hours a day, 7 days a week, 365 days a year, without interruption.

Changing hardware platforms is one of the most anxiety-provoking transitions in IT. The mere mention of a change of hardware platform brings forth images of long, uncertain migration plans and large numbers of modifications. The resources required for such projects are substantial, and the resulting budgets are commensurate.

While these reasons for concern are reasonable in a general sense, they do not apply to OpenVMS. This article examines the issues involved in the migration of applications on OpenVMS from computing systems based on the reduced instruction set (RISC) Alpha architecture to HP Integrity servers based on the Intel Itanium architecture. OpenVMS provides a set of unique facilities to make the transition from the Alpha architecture to the Itanium architecture a controlled and predictable evolution, with low technical risks. As a result, the transition can be managed with a minimum of business risk. OpenVMS on VAX, Alpha, and HP Integrity systems offers several facilities that allow the management of platform change on an incremental basis. When used properly, these facilities reduce transitions to a technical exercise with controlled steps. They eliminate the largest risk in IT management: the cutover. Instead of choreographing a cutover that affects all users simultaneously, the problem is reduced to transitioning users from the old to the new in gradual incremental groups, commensurate with the available support.

The nature of transition risk

Platform changes have been one of the classic nightmares of IT, [1] resulting in missed commitments, expensive overtime, interruptions of availability, and other dramatic side effects. It is no wonder that IT executives look upon such changes with all the enthusiasm of facing Dracula in person.

Different risks for different organizations

First, it is necessary to fully appreciate the process of platform transition in an end-user environment. Transitions in end-user environments are fundamentally different from transitions in product development environments, such as an independent software vendor (ISV) or original equipment manufacturer (OEM). In an ISV or OEM environment, the workload is skewed toward technical and product issues. Quality control and testing are important, but they are qualitatively different from their counterparts in an end-user organization. By contrast, an end-user organization has more pressing commitments to operations, as well as a lower ratio of engineering resources to programs.

Software development organizations often have extensive source control and rebuild processes, as well they must. Rebuilding products en masse is simply a fact of life. By contrast, end-user organizations are, even in these days of Sarbanes-Oxley,[2] far less likely to have the controls and processes in place to rebuild all used programs from their respective source components. This fact was encountered repeatedly during the remediation of Year 2000-related problems.

In reality, all computing facilities combine aspects of development and production (Figure 1). From a high-level perspective, all facilities can be categorized as an amalgam of the two extreme cases. In reality, it is not unusual to find that the exact categorization of an application, or a component of an application, depends on the role played by the particular component. For example, a compiler is a production component in an organization that is devoted to development, and is just as critical as a production application in an installation that does not undertake software development.

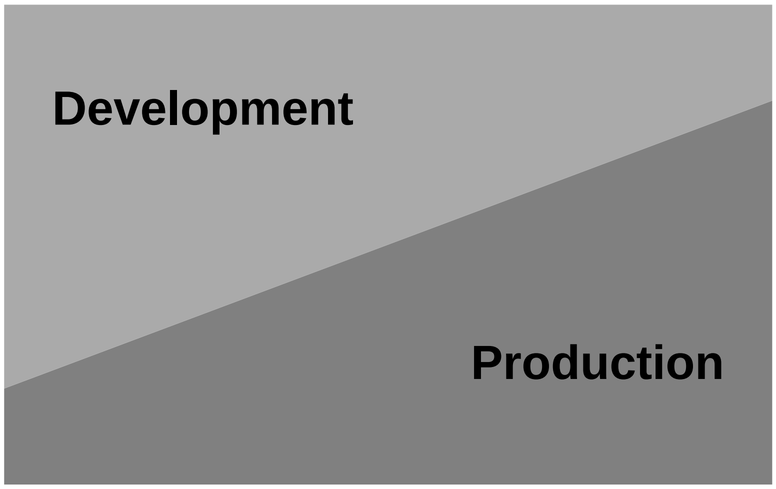


Figure 1 – All computing facilities are some combination of development and production.

As noted, platform replacement is as much a management exercise as a technical exercise.
In end-user environments, tasks are weighted toward management and qualification issues. In companies with hundreds or thousands of users, even a minor problem during a cutover can be

[1] The worst nightmare is generally considered to be a total "loss of data" event, such as that experienced by many small and medium businesses in the aftermath of some disaster, such as the World Trade Center attack (September 2001) or Hurricane Katrina in New Orleans (September 2005).

[2] In the United States of America, the Public Company Accounting Reform and Investor Protection Act of 2002 (Public Law 107-204), generally referred to as Sarbanes-Oxley, after its authors, provides the legal basis by which the Chief Executive Officer of a publicly owned corporation must personally certify the accuracy of all financial results on a regular basis.

catastrophic. In concept, the issue is simple: user support is a finite resource. Ideally, one should not cut over more users than can be supported in the event of a problem. The challenge is how to realize this management imperative within the realm of technical feasibility.

The strategic approach advocated by this article is, in turn, useful to OEMs and ISVs, in the context of shortened time and cost to market of products for use on HP Integrity servers running OpenVMS. The same concepts that apply to end users permit ISVs and OEMs to achieve delivery schedules with a low cost, low risk, and high certainty.

Management is rightly concerned with equipment costs and availability, additional problems that are often associated with the acquisition of a new hardware platform and the retirement of older hardware. Once schedule commitments are made, changes are often difficult and expensive. Older equipment might be nearing end of lease, and lease extensions can be expensive. A small slip in migration schedule can be disproportionately expensive. There is often a financial flashpoint between technical and cost considerations, with the technical team eager to defer migration to reduce technical risk, and the management team eager to retire older equipment to reduce financial exposure.

The general concept that operations must be sustained during a transition is far older than general-purpose computing.[3] Business crossed the line into continuous availability more than a century ago, with the advent of the telephone. In 1930, the 12,000-ton Indiana Bell Telephone Building in Indianapolis, Indiana, was relocated to make room for new construction while the full complement of 600 employees continued normal operations, including the central telephone switchboard. The Eichleay Company performed this move in two stages, an initial move of 52 feet, and then a rotation of 90°.[4]

Today, telephone switching centers are clearly computer applications. The move of the Indiana Bell Telephone Building was undertaken precisely because an interruption of telephone service while changing buildings was unacceptable. IT-dependent enterprises might differ in the details, but the concept remains unchanged.

[3] "Business as Usual While Moving an Eight-Story Steel-Frame Building," *Engineering News-Record*, July 2, 1931, page 8.
[4] "Business as Usual," p. 4.

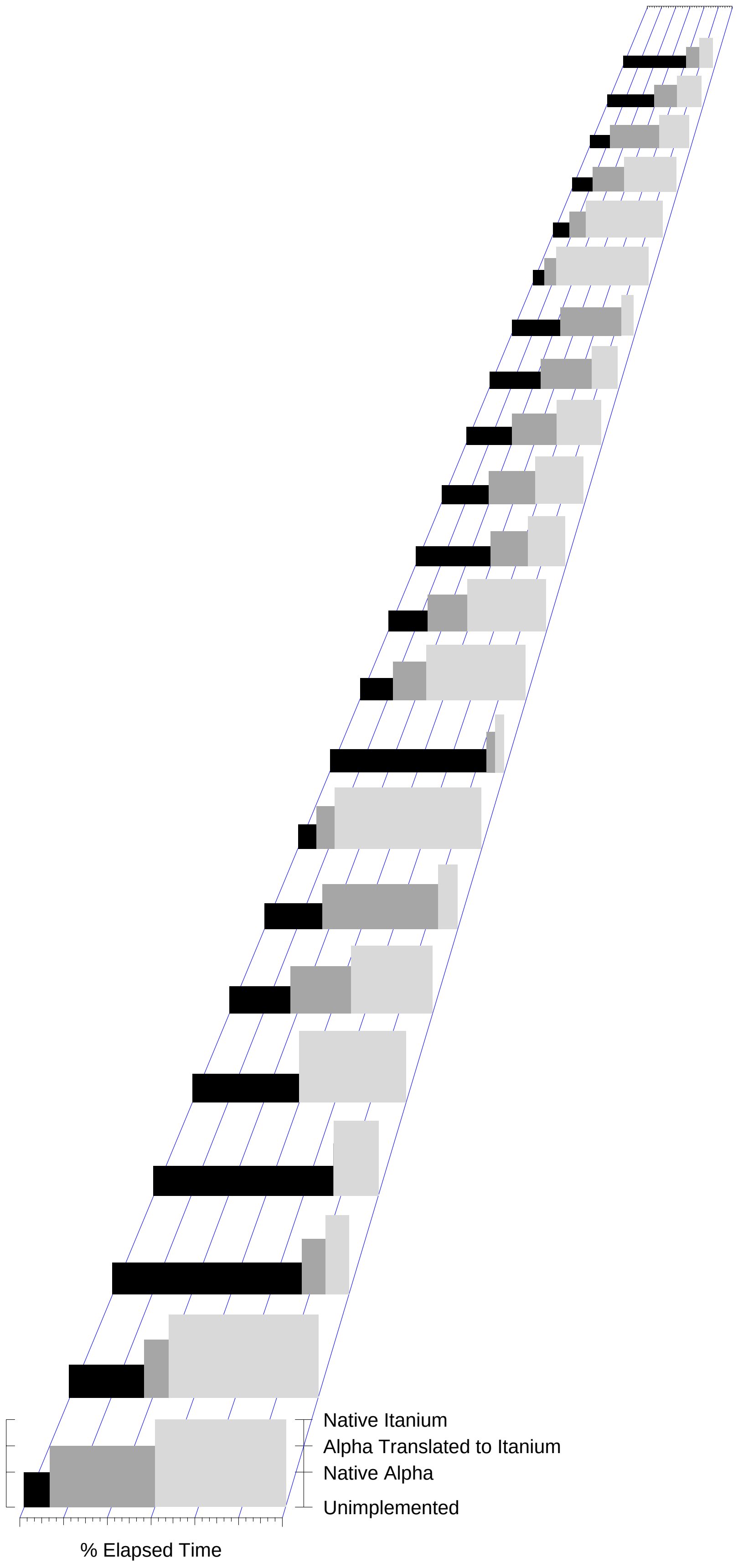


Figure 2 – Each application on an OpenVMS system can be migrated to HP Integrity servers as needed. An initial version might be composed of translated binary Alpha executable images, with the individual component images replaced by native Itanium-based executable images as the native images become available for production use. This can be done on a user-by-user basis or a component-by-component basis. Decoupling of the recompilation and requalification effort from the hardware changeover gives management flexibility in hardware deployments and retirements, without impacting production or functionality. Each software component follows its own transition path from native Alpha image to (optionally) a translated Itanium-based image and, finally, to a native Itanium-based image.

Software qualification and operational commitments

Software qualification is a major hurdle in many enterprises. End-user organizations must qualify software for several reasons, some of which are internal business processes, and some of which are required by external regulations. As compared with ISVs, software rebuilding and requalification are substantially more difficult problems in the end-user environment.

In an ISV or OEM environment, two factors make rebuilding and requalification a simpler problem. First, an ISV or OEM is ultimately in the business of building software. Simply put, this means that the build and qualification process is an integral part of their mainstream business. In contrast, the end-user environment is typically not focused on rebuilding and requalifying software. Another difference is that end-user environments tend to have a much larger number of programs. Although the ISV or OEM can control the scope of the migration, the end-user is faced with the reality of moving hundreds or thousands of applications from one platform to another – truly a daunting task.

Project management and scheduling are also major concerns in an end-user environment. While the ISV or OEM is typically in an engineering environment without day-to-day operational requirements, the end-user must organize rehosting and simultaneously provide uninterrupted support of operational commitments.

OpenVMS provides technologies that allow IT managers to manage change in incremental steps. One example is OpenVMS clusters. When used properly, OpenVMS clusters, particularly mixed-architecture OpenVMS clusters, provide precisely this functionality (Figure 3).

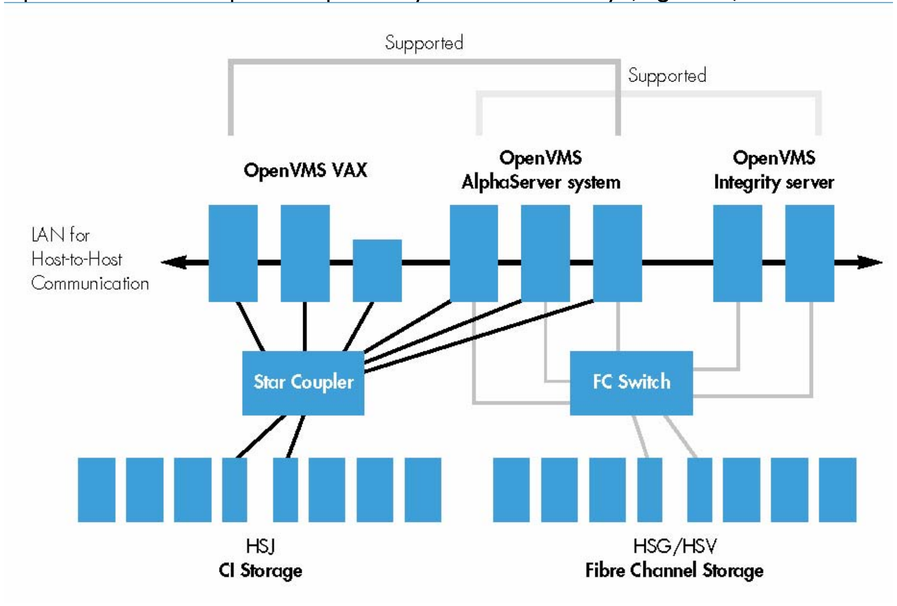

Supported
Supported
OpenVMS VAX
OpenVMS AlphaServer system
OpenVMS Integrity server
LAN for Host-to-Host Communication
Star Coupler
FC Switch
HSJ CI Storage
HSG/HSV Fibre Channel Storage

Figure 3 – OpenVMS cluster with VAX, Alpha, and HP Integrity servers (from *The HP OpenVMS Approach to High Availability Computing*, July 2004)

Engineering the details of conversion

The key to successfully moving OpenVMS environments from Alpha systems to Itanium-based systems is understanding and employing OpenVMS facilities that virtually eliminate the risks normally associated with changes in hardware platform.

For simple programs, the task of recompiling, relinking, and requalifying a program is straightforward. The suite of applications software that is at the center of modern data center operations is far more complex than a simple one-page program. The aggregate source base for an enterprise's locally developed applications programs often can be numbered in the thousands or millions of lines of code. For many application systems, the quality assurance process required of an applications system[5] is a more laborious component than the actual modification and testing of the code itself. ISVs and organizations that regularly rebuild their products are well prepared to perform the steps needed to migrate to Integrity server systems.

The hardware

For much of the last 50 years, migration from one platform to another has been considered a high-risk activity. We sometimes forget that, in the past, changing hardware platforms inevitably involved changing both the hardware platform and the operating system, often involving major changes in both applications programs and the scaffolding of command procedures that surrounded them. Such migrations also involve a "point of no return," after which reverting to the predecessor system is not easily accomplished. Indeed, these migrations, which require significant changes to all aspects of the computing environment, are high risk.

Carefully employed, the OpenVMS environment almost completely eliminates such business risks. Unlike many other hardware transitions, the transition of an OpenVMS Alpha system to an Integrity system is purely a question of changing hardware platform. With rare exception,[6] differences occur only in application programming interfaces (APIs), which are not used by most programmers.

It is important to note that OpenVMS was the first operating system for which a processor architecture was designed, rather than the converse of an operating system designed for a particular processor architecture. This fundamental strength has allowed OpenVMS to assimilate two additional processor architectures, each one radically different from the other.[7]

VAX architecture

The first member of the VAX family, the VAX-11/780, debuted in 1977. The architecture was a byte-oriented, CISC computer with 32-bit arithmetic, 16 registers, integral floating point, virtual memory, and other features. [8]

The VAX-11/780 was the first of a series of systems dubbed "superminicomputers," which provided many features previously available only in high-end mainframes in a price range more familiar to minicomputer users.

Unlike many past computer architectures, the VAX architecture was designed by a combined group of hardware and software engineers. It featured integral support for many software requirements, ranging from instructions for complex array indexing (used by compilers) to instructions for enhancing operating system constructs, such as asynchronous system traps (ASTs).[9]

Alpha architecture

The Alpha architecture, released in 1992,[10] is a 64-bit CPU architecture designed around the concepts of RISC. Originally used in the IBM 801,[11,12] RISC optimizes performance by simplifying the

---

[5] Levine, Diane E., "Software Development and Quality Assurance," Chapter 25, *Computer Security Handbook,* 4th Edition, Bosworth, 2002.

[6] There are differences in the application programming interfaces (APIs) used to access hardware-specific elements (such as arithmetic fault processing).

[7] The original 32-bit VAX architecture was an example of complex instruction set computing (CISC). In 1992, OpenVMS assimilated the Digital-designed, 64-bit Alpha processor, an example of reduced instruction set computing (RISC). In 2005, OpenVMS Version 8.2 was released, with support for both the Alpha RISC processor and systems based on the Itanium-based Explicitly Parallel Instruction Computing (EPIC) architecture, developed jointly by HP and Intel.

[8] Strecker, W.D., "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family," *Proceedings of the National Computer Conference,* Strecker,1978.

[9] See *VAX11 Architecture Handbook,* Digital 1978, and *VAX11 Software Handbook* [Digital1978a].

[10] See *Alpha Architecture Reference Manual* [Sites1992].

processor's instruction set. Ideally, each instruction in a RISC CPU requires a single cycle to execute. The Alpha processor has a larger register set (64 registers) than the original VAX but is culturally compatible with its CISC predecessor.[13]

One of the hallmarks of the later VAX processors, and of the entire Alpha series of processors, was an increasing level of hardware-managed parallelism, which was designed to increase the effective speed of the central processor.

Itanium architecture

HP Integrity servers are based on the Itanium architecture, which was jointly developed by HP and Intel. The Itanium instruction set is known as Explicitly Parallel Instruction Computing (EPIC).[14]

The Itanium architecture is different from the VAX and Alpha architectures in a number of ways. The VAX and Alpha designs are characterized by a separation between software optimizations and hardware optimizations. While there was coordination at the design and architectural level, there was no way for compilers on VAX and Alpha systems to signal information for code optimization to the hardware. VAX and Alpha were both designed so that coding in the actual machine code was viable.

This is not the case with the Itanium architecture, which presumes that Itanium machine code is compiler generated. The Itanium architecture removes hardware-imposed conflict resolution for registers. Instead, the compiler code generators are responsible for all such scheduling. The Itanium architecture also has facilities for predication, which reduces the need for branches, and cues for speculation, which indicate the probability of those branches that do exist.

Another difference between the earlier processor architectures used with OpenVMS and the Itanium architecture is the handling of registers on subroutine call. The earlier architectures performed at least a partial register save at each subroutine call. In modern high-level languages with call-intensive paradigms (such as C/C++), this leads to a high number of memory references at each call, resulting in a memory bottleneck. The Itanium architecture implements a register windowing scheme that allows the elimination of these register context operations.

The concepts behind the Itanium architecture appear to be dramatically different from the instruction set architectures that have been the mainstay of computing since the mid-1950s. However, examined in detail, this difference is far less important than it seems. Architectures that are generally similar to EPIC have been part of computing for more than two decades, through the mechanisms used to implement the familiar von Neumann architectures (for example, IBM System/360, DIGITAL PDP-11, VAX, and Alpha).

OpenVMS application programming interfaces

The true test of portability is whether an unchanged source program can be compiled for two different architectures.[15] This has been feasible for OpenVMS users since the debut of VAX/VMS in 1977. OpenVMS has been deliberately implemented on each of the architectures selected so that it is source compatible with its predecessors. It is not uncommon for programmers to successfully use 20-year-old documentation for most of their development work without ill effect. This ability to

[11] Cocke, J., and Markstein, V., "The Evolution of RISC Technology at IBM," *IBM Journal of Research and Development*, Volume 34, Number 1, 1990 [Cocke 1990].
[12] Rodin, G., "The 801 Minicomputer," *IBM Journal of Research and Development*, Volume 27, pp. 237–46, 1983, [Rodin 1983].
[13] Dobberpubl, et al., "The MicroVAX 78032 Chip, A 32-bit Microprocessor," *Digital Technical Journal,* Volume 2 (March 1986), pp. 13-14, [Dobberpubl 1986].
[14] Schlansker, M.S., and Rau, B.R., "EPIC Explicitly Parallel Instruction Computing," *Computer* (February 2000) [Schlansker 2000].
[15] Or with minimal changes.

assimilate dramatic changes in underlying technology without changing the API has been one of the historic strengths of the OpenVMS operating system.

For this reason, many programs written for early versions of VAX/VMS continue to run unchanged over 25 years later. In many cases, even recompilation has proven unnecessary. This consistency is not an accident. It is the result of conscious engineering decisions

.

Nonprivileged code

Central to the sequential migration from VAX to Alpha and from Alpha to HP Integrity is the fact the each pair of architectures fully supports the same data types. Thus, with appropriate care, it is possible to perform precisely the same computations on Alpha as on VAX and, in turn, on HP Integrity as on Alpha. Table 1 shows the compatibility of these formats.

		Architecture	
Characteristic	VAX	Alpha	Itanium
Character	ASCII	ASCII	ASCII
Floating-point format	VAX	VAX, IEEE	IEEE
Integer	32-bit	32/64-bit	32/64-bit
Number format		Two's complement	
Address size	32	64	64
Architecture type	CISC	RISC	EPIC

Table 1 - Data formats native to each processor type[16]

Correctness should not be confused with efficiency. When the VAX architecture was originally implemented, memory sizes were dramatically smaller, and packed data structures were considered standard practice. The more recent Alpha and Integrity platforms have dramatically larger basic memory sizes and correspondingly larger performance penalties for accessing unaligned data. Therefore, it is now preferable to use data structures whose elements are aligned on their natural boundaries.

In the past, differing hardware architectures directly impacted many issues visible to programmers, even applications programmers working many levels above the actual hardware. In those days, switching hardware platforms meant changes in integer representations, character sets, floating-point formats, and operating system interfaces. Each of these changes wreaks havoc on an installation's inventory of existing programs.

Today, programmers working at levels above the actual machine language -- even third-generation languages such as Fortran, PL/I, COBOL, BASIC, C/C++) – are little affected by the mechanics of the platform's instruction set, whether it is CISC, RISC, or EPIC. For that matter, programmers using Macro-32[17] on processors other than VAX are, in effect, using a higher-level language compiler rather than the assembler used on the VAX architecture.

The underlying processor architecture does not become visible until the programmer uses either the option to display the actual generated code (for example, the `/MACHINE_CODE` option in the OpenVMS C/C++ compiler), or the options in the symbolic debugger to display the actual machine-language code. Both of these generally are rare events. It is possible to work for years, or even decades, without exercising either of these options.

[16] Gezelter, R., "The Third Porting: Applying Past Lessons to the Alpha/Itanium Transition," CETS, September 2001 [Gezelter2001].

[17] Macro-32 is the assembler language for the VAX-11/780. See [HPMacro32].

By contrast, the data formats, character codes, and operating system interfaces are directly visible to programmers. On the three architectures supported by OpenVMS, all the available interfaces are the same across VAX, Alpha, and Itanium processors.[18]

Floating-point representations

Floating-point formats are another potential flashpoint. Alpha supports both the original VAX format for floating-point numbers and the IEEE standard representation.[19] The default on Alpha systems is VAX floating-point format. Figure 4 illustrates the differences in the layouts and details of the different floating- point formats.

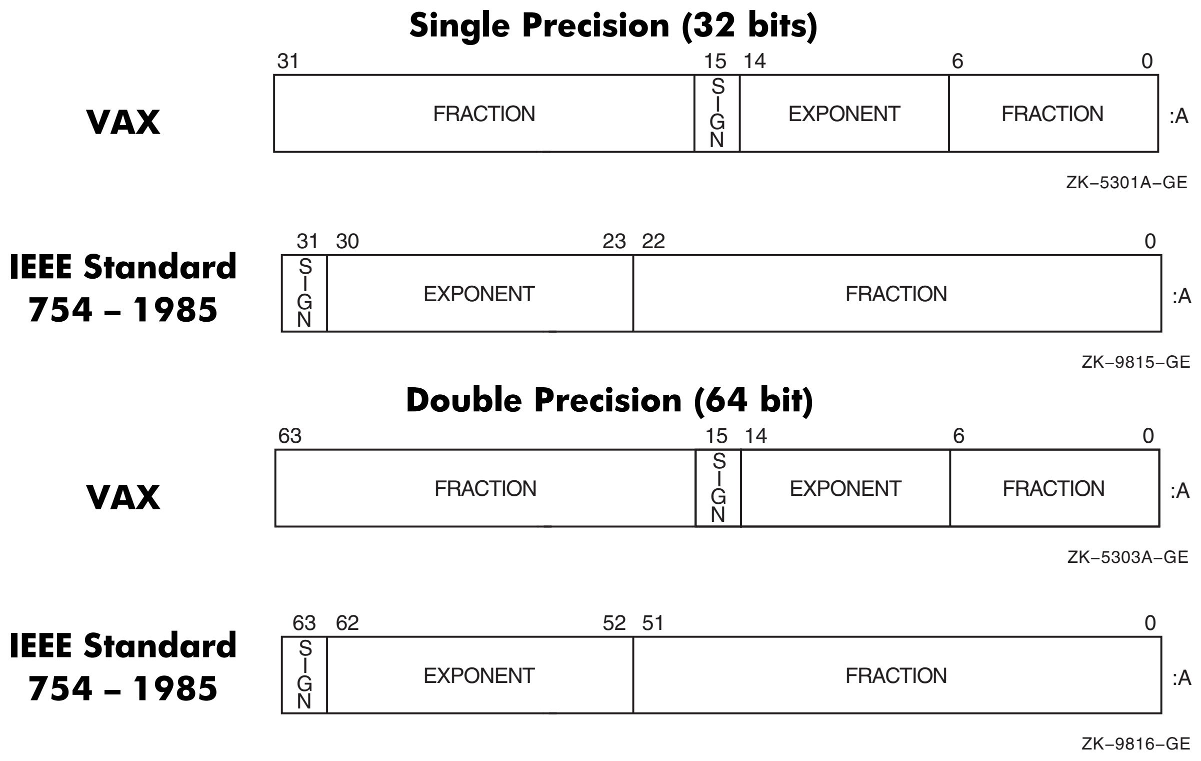


Figure 4 – Single- and double-precision floating-point formats for VAX and IEEE Standard 754-1985 (from *HP FORTRAN for OpenVMS User's Guide*)

Unlike Alpha processors, Itanium processors do not natively support the VAX floating-point formats. For most programs the differences are minor, and for most purposes the results will be the same. The difference in floating-point formats and the resulting small differences in computations is an obstacle in two situations: binary comparison of outputs between Alpha and HP Integrity systems, and some scientific computations.

One standard technique is to compare the output from one system with the output of another system. This technique is often used as a regression test or when migrating code to a different platform. Presumably, if the program produces an identical output for a reasonably sized dataset, it is

[18] Interfaces that are directly used to manage and address the peculiarities of each architecture, such as processing related to processor traps and similar operations, are, of course, different on each of the architectures. The implementation mechanics of subroutine calls are also different on each of the architectures. The mechanics of CALL operations are generally only the responsibility of the various compilers. In the vast majority of code, particularly applications code, these areas are tangential.

[19] *IEEE Standard for Binary Floating Point Arithmetic, ANSI IEEE Standard 754–1985* [IEEE1985].

functioning correctly.[20] For this reason, it is a good idea to recompile and retest programs using IEEE floating point on Alpha as a first step. Once the results of the program are determined to be correct, the same program on an Integrity system should produce the same results.[21]

Privileged code

Programs operating at inner modes, particularly programs that directly manage processor components, are inherently more dependent on the differences between the architectures. These differences matter to developers working at inner access modes and employing hardware-specific features.

Privileged code that is dependent on OpenVMS privileges are generally unaffected by changes in architecture. Such code does not rely upon operations that directly affect the processor or other hardware. As an example, a program that enters kernel mode to call a system component, but that does not otherwise affect processor hardware, is likely to be unaffected by changes in the underlying architecture.

Hardware-specific functionality

At the device and processor levels, the three architectures differ from each other just as processors and systems differ within each of the individual architectural families. These differences are generally invisible, particularly in end-user environments.

Conversion options

In a conventional, non-OpenVMS environment, there is little choice but to do a so-called “cold turkey” cutover of applications from one processor and operating system to another. Such a cutover requires migration of data en masse from one system to another, often requiring reformatting of data for the new system. This is an all-or-nothing approach. Once the process starts, there is often no way to simply reverse the process if something goes awry.

This approach is heavily laden with risk. In today’s environment, down time is prohibitively costly to the organization. A study by TechWise[22] showed that even modest down time frequently costs in excess of $10,000 USD/hour. Major system conversions that have gone awry have even impacted publicly reported profits.

The facilities of OpenVMS provide a uniquely malleable environment for changing hardware platforms. Specifically, many strategic and tactical choices are available that allow decoupling of the hardware cutover from the actual applications migration.

An OpenVMS migration from one processor architecture to another is a far lower-risk scenario. Disk data formats are consistent across OpenVMS systems running on different architectures. In fact, OpenVMS clusters provide the mechanisms to decouple data migration from application migration, whether or not an installation uses a Storage Area Network (SAN). Properly managed, it is quite possible to grow OpenVMS storage environments significantly and to change their physical realization without ever interrupting user access to information.[23]

---

[20] Or at least as correctly as the original program was functioning. This technique is popular for writing compilers because it produces a high degree of assurance that the compiler can be used to compile itself. Therefore it can be used to compile its own bug fixes.

[21] In fact, the Java® standard requires the use of the IEEE floating-point formats for precisely this reason. “4.2.3 Floating Point Types and Values”, p. 33 [Gosling1996].

[22] TechWise Research, Inc., *Quantifying the Value of Availability: A Detailed Comparison of Four Different RISC-Based Cluster Solutions Designed to Provide High Availability*, Version 1.1a (June 2000) [TechWise2000].

[23] See Gezelter, R., “Migrating OpenVMS Storage Environments without Interruption/Disruption,” HP Technology Forum, 2005 [Gezelter2005].

Managing change – the OpenVMS approach

OpenVMS, with its support for mixed-architecture OpenVMS clusters and its provisions for binary translation,[24] provides a way to assimilate Itanium-based servers into a production environment without requiring a "cold turkey" transition.

The combination of image translation and mixed-architecture OpenVMS clusters allows computing capacity and hardware inventory to be transitioned in the same orderly fashion as the software inventory. Computing capacity can be transitioned from one architecture to another as dictated by the changing needs of the applications mix (Figure 5). This allows cost-effective management of the hardware inventory separate from the tempo of the migration effort. Such a strategy allows users to be migrated to the new architecture in a seamless manner, thereby optimizing financial, technical, and support expenses and risks.

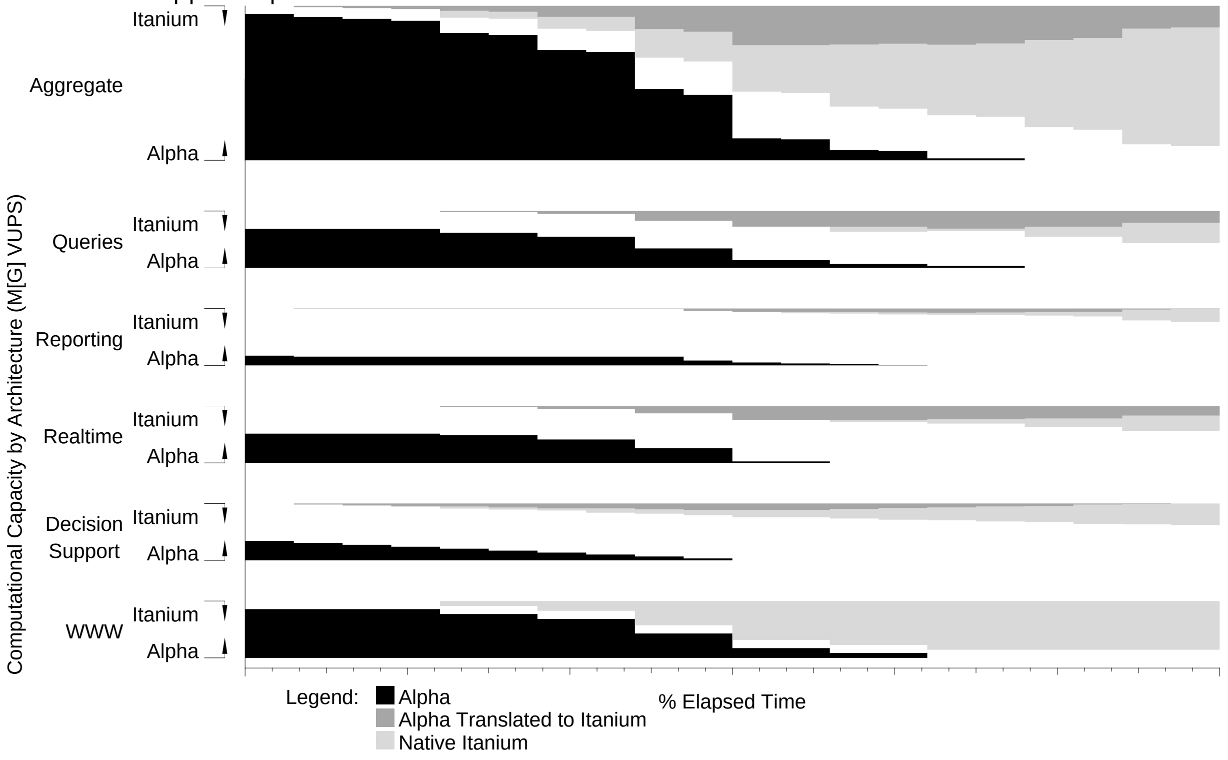


Figure 5 –Mixed-architecture OpenVMS clusters capacity transition from Alpha to HP Integrity servers over time.

Conventional changes in processor architecture give rise to another problem: interdependencies between applications and underlying software components. Modern software applications are not freestanding entities. Each one depends on an extensive collection of supporting software, each of which might depend on other software components. These interdependent components are often maintained by different groups, each with their own priorities, responsibilities, and commitments. Adding to the complexity, some components might be developed and maintained in house, while others might be licensed products.

These interdependencies, which can be devilishly difficult to ferret out, have been the cause of more than one migration setback. Concern and respect for the complexity of this problem has been part of the OpenVMS engineering philosophy since the initial design of the VAX-11/780. These concerns are at the heart of why such transitions are qualitatively safer on OpenVMS.

[24] On Alpha, for code translated from binary VAX executables; on Itanium, for code translated from Alpha executables (which, in turn, might have been translated from VAX binary images).

The solution to this problem of interdependencies was to implement many noncore parts of the operating system and its utilities as PDP-11 images, running under the RSX-11 Applications Migration Executive (AME).[25] This allowed engineering resources to be concentrated on components that had to operate in native mode (for example, the executive, RMS). It also allowed development of supporting components (such as compilers, editors, and tools) on existing, readily available PDP-11 systems. This approach allowed the OpenVMS Engineering organization to ship the first VAX-11/780 in October 1977, only 18 months after finalizing the initial design. The tools and utilities running under the emulation mode were gradually replaced in subsequent releases of the VAX/VMS operating system.[26] The common data formats and character sets between the different architectures mean that data translation was and is rarely required.[27]

In fact, the image-translation support allows native Itanium binaries to invoke translated images, and vice versa. It is such a fundamental part of the operating system implementation that it happens without conscious thought on the part of most users and system managers. Translated images, through transfer vectors, invoke entry points in the normal common run-time library when required.

Translated images have been part of OpenVMS production systems since the advent of Alpha and VAX. Perhaps the most commonly used translated component in recent history has been the `Monitor` utility (MONITOR) on the Alpha platform. Unheralded and virtually unnoticed, the Alpha version of `MONITOR` has been translated from successive VAX images since the advent of the Alpha in 1992 through 2005.[28] Similarly, the `TECO` text editor has made use of emulation and translation since the advent of the VAX in 1977.

The preceding examples suggest a prudent, time-tested strategy – one that has been used three times by the OpenVMS Engineering organization itself to move a development and production environment from one hardware architecture to another without interruption.[29] This strategy is characterized by an approach that dramatically reduces the Gordian knot[30] of components that must be migrated in order to have an operational production system.

Environments have a range of different applications (Figure 6). Core applications are critical to daily operations; other applications might be critical, but on a far longer time scale. Still others might be important but are not used on a regular basis.

[25] [Digital1978a], pp. 11-1 – 11-8.

[26] The last mainstream component to run in emulation mode was the SOS editor, retired eight years later with release of VAX/VMS Version 4.0. This coincided with the release of the MicroVAX I, the first VAX-family processor that did not include native hardware support for the PDP-11 instruction set. At that time, the AME, now a separate product, was enhanced with the addition of software instruction set emulation, allowing customers to continue to run PDP-11 images on VAX systems.

[27] The data types used in VAX translated images are available on Alpha. The data types used on Integrity are common with Alpha. All of the data types, with the exception of the floating point formats, are the same across all three architectures. The Alpha Environment Software Translator (AEST) has provisions for calling software routines to process VAX floating-point formats transparently, albeit with some loss of efficiency.

[28] The `MONITOR` utility on Alpha, through OpenVMS Version 7.3-2 – a period of over a decade – was translated from the VAX executable by the VAX-to-Alpha binary translator. The majority of the user community was not aware of this difference until the native mode MONITOR was announced with the release of Version 7.3-2.

[29] PDP-11 to VAX (1977 et seq.); VAX to Alpha (1992, et seq.); and Alpha to Integrity (2005, et seq.)

[30] The Gordian knot was a complicated knot that could not be untied. It was cut by Alexander the Great.

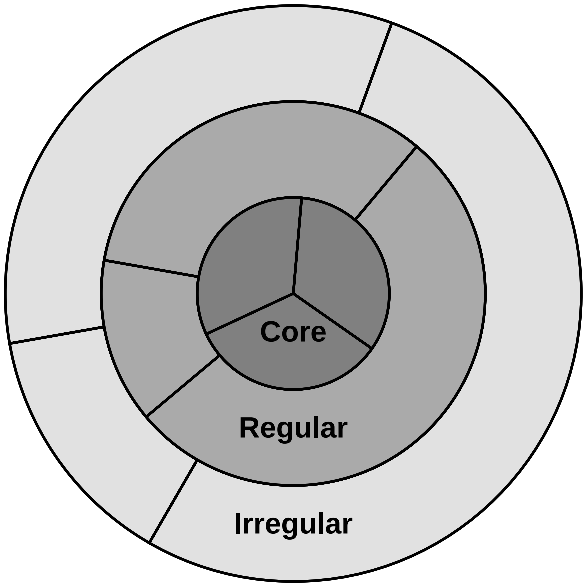
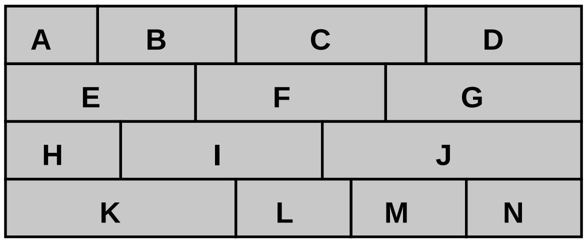


Figure 6 – Typical environments include a range of applications, from core to irregular utilities and reports.

Each end-user environment is unique and has its own issues — technical, political, and financial. In some environments, it is desirable to migrate core applications first. In others, the best candidates for early migration are the rarely used utilities that are not critical to day-to-day operations. Even two organizations in similar niches in the same industry can have different preferences for the order in which workload is moved from the old platform to the new platform. One of the strengths of OpenVMS is that this decision, with all its nuances, is driven by end-user business issues on a case-by-case basis.

Workload migration is also not limited to application-by-application migration. OpenVMS clusters might include systems with differing architectures transparently accessing the same files. Once an application is believed to be ready for production, it is possible to rehost the user community in stages that are calculated to minimize risk and strain on user support resources. Thus, in a well-executed OpenVMS migration, few situations require the transition to be made all at once.[31]

This capability of mixing and matching components allows a wide degree of flexibility when scheduling transitions of components from translated to native compiled forms (Figure 7). Initially, production might use an application entirely composed of translated images. As images become qualified for production use, it is a simple matter to integrate them into the production profiles. When this iterative process is completed, all of the images are natively compiled for the Itanium architecture.



[31] That is, changes made purely to the architecture of the underlying system. Pure changes to the hardware realization of resources can also be accomplished without disruption. Some changes, such as overall changes to the security scheme, might require "cold turkey" transitions.

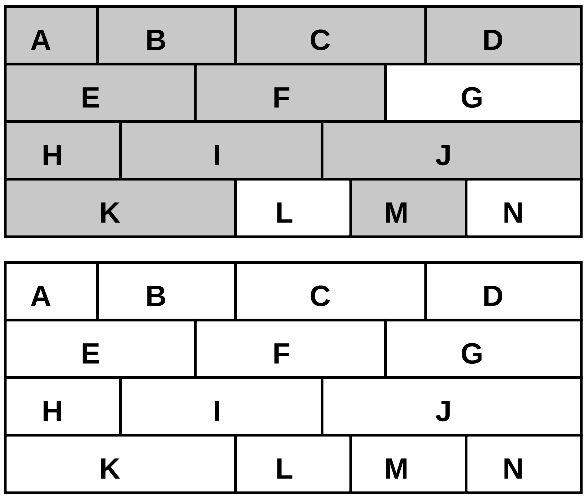


Figure 7 – Translated images allow the migration to be achieved one component at time.

Mixed-architecture clusters, together with binary translation, allow workloads to be moved to newer platforms in an orderly fashion that is compatible with all the requirements of business: financial, operational commitments, and technical resource availability. The same flexibility that is available with different processes is also usable on a shareable library-by-shareable library basis. Applications rarely exist as singletons. In the majority of cases, end-user environments have many applications, often grouped into families that use common supporting libraries.

The translated-image facilities enable individual libraries to be used in translated or native environments, as appropriate. Additionally, since the choice of library is controlled through the use of logical names,[32] different libraries (one native, one translated) can be used on the same cluster member, depending on some combination of user, application, and other criteria.[33]

Compiling natively for the Itanium architecture

Actively maintained applications are far easier to migrate. Active maintenance means that the code base[34] is recompiled and relinked regularly. Current code bases are less likely to use obsolescent or variant language features that are not available on the Itanium compilers. In the best case, these code bases can be migrated quickly to a native Itanium environment simply by recompiling, relinking, and requalifying.

In many environments, this smooth state of affairs is not the case. It is common to find executable images and shareable images that have not been rebuilt or recompiled for over 10 years. It is also common to find a fairly small staff responsible for literally hundreds of programs. In such an environment, the sheer volume of code makes even modest changes a daunting, drawn-out project.

Recompiling

Theoretically, recompilation is a routine process. The challenge is caused by the changes in the underlying language or options that are supported by the compiler. These can be extensions

[32] Section 8.3.2 *Activation of Shareable Images*, pp 336 [Goldenberg1997]
[33] Gezelter, R., "Inheritance Based Environments for OpenVMS Systems and OpenVMS Clusters," *OpenVMS Technical Journal*, Volume 3, February 2004 [Gezelter2004].
[34] A code base is the collection of source files that are compiled to produce an image. This includes source, header, include, and similar files. External libraries used by the resulting image are not generally considered part of the code base for an individual application.

available with the VAX or Alpha compilers that are not available on Itanium-based systems, [35] or they can be intervening changes in the level of the language.[36] In either case, source changes must be made before the program can be recompiled, and the recompiled program must be qualified to ensure that the changes have been made correctly.

Recompilation on Itanium-based systems also changes the default floating-point format from VAX to IEEE, except in the case of Java, whose specification already requires the use of IEEE floating point.

Linking

At the user level, there are few differences when linking on any of the architectures supported by OpenVMS.[37] Each architecture has its own object and executable file formats. On HP Integrity systems, the System V Executable and Linkable Format (ELF) is used for object files, shareable images, and executable images.[38]
The different formats are not an issue for users. Compilers targeting each architecture generate the correct object format for that architecture, and the Linker processes these object modules to create executable image files. Linker command files used on Alpha systems should not need changes to operate correctly on HP Integrity systems.

Translation versus emulation

Translation is far different from emulation. Emulation reinterprets the actual binary image of the instructions used by an earlier (or different) architecture. In the classic OpenVMS example, the original VAX-11 series processors[39] had integral hardware to emulate the nonprivileged aspects of the PDP-11[40] instruction set. Later VAX products used software emulation, which allowed them to execute unchanged binary images from RSX-11M/M-PLUS systems.[41]

The penalty for the flexibility of emulation is that each instruction must be reinterpreted each time it is used. Depending on a variety of factors,[42] software emulation produces extra overhead that reduces performance, often by a factor of 5 to 10.

Translation is inherently different. Translation uses an offline utility, the translator, to convert a binary executable image from the instruction set and format of one architecture to the instruction set and format of a different architecture. This process is akin to using a compiler in that it is run a single time. Thereafter, the resulting image executes without extensive supervision.[43]

Performance

The performance of translated binary images depends greatly on what actual processing is performed. Later in this article, some of the case studies show how the OpenVMS shareable library

[35] As an example, the OpenVMS C compiler supports the non-ANSI C Language extensions provided by the earlier VAX C compiler. The C compiler for Itanium-based processors does not support these extensions. The changes to bring code into compliance are not large, but in a large source base the sheer scale of the effort is a challenge.
[36] The Fortran compiler for Itanium-based processors supports Fortran-90. Many programs were written using earlier versions of the Fortran standard.
[37] VAX uses a different mechanism for declaring entry points to shareable libraries. Additionally, support related to 64-bit addresses is limited to the Alpha and Itanium implementations.
[38] See [TIS1995].
[39] VAX-11/780, VAX-11/750, VAX-11/730, VAX-11/782, VAX-11/785, VAX-11/725, VAX 8600, and VAX 8650.
[40] See [Digital1978c].
[41] Because the operating system API for the RSX-11 family differed from that provided by OpenVMS, a software library referred to as the Applications Migration Executive (AME), provided the mapping between the RSX-11 executive directives and the system service calls used by OpenVMS.
[42] See "Binary Translation," *Digital Technical Journal,* Volume 4, Number 4, 1992 [Sites1992a].
[43] The Translated Image Environment (TIE) for Alpha images executing on an Itanium processor includes an Alpha instruction set emulator. This emulator is invoked automatically when a fault caused by an attempt to transfer control to a block of instructions not identified as executable code by the binary-image translation utility, the Alpha Environment Software Translator (AEST). Therefore, each translated Alpha executable image contains a complete copy of the original Alpha executable.

facility and the image translation utility can be used in concert to optimize the timeline. This optimization is achieved by concentrating efforts where the most performance can be gained, and by translating other parts of the application that are not as performance sensitive.

Appropriate choices

Modern business must be agile, quickly adapting to changes in the business environment. This need occurs at all levels, including business strategy, IT provisioning, and IT operations. In IT provisioning, OpenVMS provides developers and maintainers with a range of choices for how to proceed with migration to a new hardware platform. In the simplest case, with plentiful resources and fully compliant sources, it is simply a matter of recompiling, rebuilding, and requalifying. This is generally the choice of software developers and of those with isolated programs. The problem of large, interrelated applications environments is a different world. Here, the conflicting requirements of different obligations and costs can easily create a situation of tangled interdependencies.

When resources or schedules are less favorable, OpenVMS features provide the ability to divide the migration into manageable, low-risk segments (Figure 8). These steps can be performed incrementally, with the end result of total conversion to native operation on Integrity servers, but without the risks associated with an instantaneous cutover.

Users are often required to log out of applications at various times in the work day (for example, at breaks, at meals, or at end of shift). These natural operational boundaries can be harnessed as points at which users, either as individuals or as groups, can be transitioned to changes in the underlying environment. Replacements for each component or group of components can be accomplished at the proper time, as a side effect of routine operations. This approach results in an orderly transition that does not impact users.

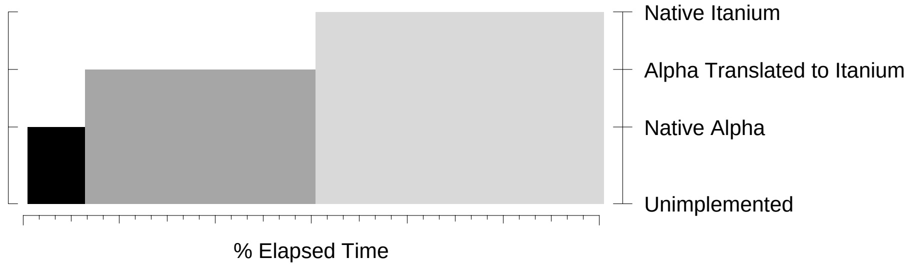


Figure 8 – An individual application or shareable image component of an application can be transitioned from Alpha to HP Integrity systems in stages, from image translation to native execution. This allows users to be transitioned from Alpha servers to HP Integrity servers without the need to recompile code.

Mixed-architecture OpenVMS clusters can be used in the same fashion. The ideal of transferring an OpenVMS cluster composed of VAX and Alpha nodes and affiliated storage to a pure OpenVMS cluster with only HP Integrity systems and current-generation storage arrays is achievable. When done carefully, it can be accomplished without any total interruptions of overall system availability. No changes to production routines other than the logins and logouts routinely done during the workday are required.

The procedures of cutting over OpenVMS cluster members without shutting down the entire cluster are well understood, as are the techniques that enable the migration of storage without interrupting user or application access.[44] The following sections focus on the software management steps that allow these goals to be achieved.

[44] Gezelter, R., "Migrating OpenVMS Storage Environments without Interruption/Disruption", HP Technology Forum 2005, Orlando, Florida, October 20, 2005 [Gezelter2005].

For example, it is not generally appreciated that, until the release of OpenVMS Alpha Version 7.3-2 in 2005, the Monitor utility, which monitors system performance, was a binary translation of the image used on the VAX version of OpenVMS. The entire OpenVMS universe has been using `MONITOR` and other translated images on Alpha for more than a decade without incident or, for that matter, awareness.[45] The same technology alternative is available for translating Alpha binary images for use on HP Integrity servers.

Five-step migration process

As shown earlier, there are many options for migrating a base of applications from an older OpenVMS architecture to the Itanium architecture. The important issue is how to accomplish the technical migration with minimal risk and maximal efficiency.

The steps in the transition process are:

Step 1	Establishing automatic qualification procedures.
Step 2	Translating all binary images to Itanium images; requalifying.
Step 3	Normalizing data formats.
Step 4	Updating source files to latest version of OpenVMS (Version 7.3 on VAX, Version 8.2 on Alpha); requalifying.
Step 5	Recompiling sources for HP Integrity, producing native Itanium images; requalifying.

Step 1 – Establishing qualification procedures

During the transition process, the team must verify proper functioning of the applications multiple times. Standardizing the qualification procedures for each application ensures that these tests are done in a consistent manner at each point.

These tests fall into two categories: functionality tests and regression tests. Tests of functionality guarantee that software can perform its intended functions. Regression tests are a far different category. The collection of regression tests for a program is the collection of test cases for problems that have been resolved. Running these tests as part of the qualification process ensures that problems do not reappear once they are corrected.

OpenVMS features such as the Logical Disk driver (`LDDRV`),[46] the Backup utility, and the logical name facility make it straightforward to implement re-creatable test environments with low risk and a high degree of reproducibility.

Step 2 – Translating binary images and requalifying

En masse binary translation of all images at this point, followed by requalification, provides an assurance that the application as a whole can be run on HP Integrity servers.

An application is often not a single image. It is common for a business application to consist of many images, some directly executable, others used as shareable libraries. Creating translated, qualified versions of these images ensures that developers, each working on their own part of the HP Integrity server system, can be assured that the software surrounding their individual piece is sound. It is not necessary to use the translated software in end-user production, although as `MONITOR` has demonstrated, it is an appropriate option.[47]

[45] There are often concerns about introducing new technologies during an architectural transition. The fact that the technology has been in widespread, everyday use is often a source of re-assurance.

[46] Originally (and still available as) freeware, LD is integrated with the standard OpenVMS distribution as of Release 8.2-1. See [vanderBurg2005].

[47] The choice depends upon the economic and hardware issues referenced earlier in this paper.

Step 3 – Updating source files to current language level

Many programs have been run for years or decades without recompilation. In the intervening time, language standards have changed. For example, many users have source programs that date to the Fortran-66 or VAX C standards.

Compiling these programs with current-generation compilers often results in errors and warnings. It is important to resolve these issues before attempting to transition the software. This step isolates these changes in compilers from changes brought on by the new architecture.

Step 4 – Normalizing data formats

The default[48] floating-point formats on Alpha have historically been the floating-point formats used on the VAX[49] and PDP-11.[50] The IEEE floating-point standard is supported by the Alpha compilers and is the default on HP Integrity servers. The results produced by equivalent computations using each of the two formats differ slightly, generally in an insignificant way.[51] For this reason, the format change is listed as a separate step. The other steps should produce the identical results on a bit-for-bit basis.

Step 5 – Recompiling sources for HP Integrity systems and requalifying

Once the source base is up to current language standards, and no issues have been identified relating to the change in floating-point format from VAX to IEEE, the source base can be compiled natively on the HP Integrity system.

Rationale

It might appear that the translation and recompilation steps are redundant, but this is far from the case. Some locally developed applications might consist of a single executable with a small source base. However, an application environment consisting of hundreds or thousands of executable images is much different. The dependency diagram of such an application collection illustrates why the multiple stages are needed. Simply put, at the end of step 4, we can be certain that the majority of older hardware can be retired.[52] This development disconnects the hardware retirement from total conversion of the applications base.

Example scenario

This scenario is one of multiple, interwoven applications and libraries, where the network of interdependencies virtually guarantees conflicts and problems. The complexity of each case study illustrates choices in the process that would not be apparent in simpler situations.

The case involves a large data-processing organization with a number of semiautonomous development groups, each responsible for some segment of the applications that support the overall business. The organization might be a financial institution, a manufacturing operation, a utility, a research project, or some other enterprise.

[48] Java is the exception. Compliance with the Java specification requires the use of IEEE floating-point format. It is possible to support Java calling of external routines using VAX floating point, but a small jacket transfer vector is required.

[49] See [Digital1978].

[50] See [Digital1978c].

[51] Complex numerical calculations (such as fluid dynamics, Monte Carlo methods) take into account and depend upon the behavior of the lowest-order bits of the computational results. These computations might be affected by the change in floating-point formats.

[52] Until it is possible to successfully natively recompile and requalify *all* of the executable images, it is unwise to lose the ability to recompile on the original platform, whether VAX or Alpha. Two options for retaining this capacity are available: workstations and emulators. Workstations can be an inexpensive hardware option to retaining the ability to compile and test VAX or Alpha programs in their native architectures. A second option is the use of a VAX or Alpha emulator, such as the Charon series of emulators available from Software Resources International (`http://www.softresint.com`).

The hardware environment is currently a moderate to large cluster of AlphaServer systems. Much of the hardware is leased, and the leases expire in a relatively short time. A large number of the applications are developed in house. The staff is confident that they can perform the transition, given the time to do the work of upgrading many old sources, which have been in use for years without recompilation. Many of the images depend on a series of underlying shareable images that are maintained by several of the development groups.

Management is concerned about finances and availability. A brief review of the finances, personnel, commitments, and dependencies shows a complex series of connections which, at first glance, seems irresolvable. In short, there appears to be no way, without dramatic increments in budget and personnel, that the transition can be accomplished without extending the leases on the existing AlphaServer systems, the cost of which is prohibitive.

This is not a far-fetched scenario.[53] What is unusual is that image translation, mixed-architecture OpenVMS clusters, and host-based volume shadowing provide the enabling technologies to transition this admittedly complex environment without ever interrupting production use of the system.[54] If problems do occur, the transition can be undone in seconds or minutes, at any point in the process.

The first two steps, which can be done somewhat in parallel, are to integrate one or two Integrity servers into the existing OpenVMS cluster, with access to all of the required data volumes, and to develop qualification criteria for the various components which comprise the applications base.

The next step, image translation, is a brute force process. Each image, whether directly executable or shareable, is translated for use on Integrity servers and then requalified. The end result of this process is a full applications environment running on the Integrity platform.

How the resulting translated environment is employed is a business decision that depends primarily on cost and risk. The environment can be used as one or both of the following:

- Allow the different development teams to work in parallel, each on its own sections of the applications base, using the translated images as a framework or scaffold.
- Shift production use to the Integrity platform using the translated images, redoing the translation process for images affected by patches.

This is not an all-or-nothing decision. The translated environments can be used for production use in some applications and not in others. In many environments, the performance of the translated images is adequate for transitional production use and possibly for long-term use.[55]

Once the translation has been qualified, it is possible to retire and remove the majority of the AlphaServer systems from the OpenVMS cluster and replace their computing capacity with HP Integrity servers, as required. It is recommended that at least one AlphaStation be retained as a resource for the interim rebuilding of executable images for translation, but that is a nominal expense.

The development teams can now work independently on the language and other minor changes needed to natively recompile their components on Integrity systems. As each image is recompiled and qualified for native use on HP Integrity systems, it can replace its translated version without disruption. At all times, the full functionality of the system is available to users and developers.

[53] In fact, this is essentially the scenario addressed successfully by the OpenVMS Engineering organization three times: initial release, the transition to Alpha, and the transition to Integrity servers.
[54] Assumes that the user data is already stored on volumes configured as extendable shadow set members under host-based volume shadowing.
[55] The OpenVMS Monitor utility is a living example of the utility of this approach to technical management.

Operational issues

Operational commitments greatly complicate the situation. Cold cutovers are an acceptable strategic approach only if extended interruptions in system availability are not an issue. The recommended approach is a phased, incremental cutover. If a problem occurs while moving a group of users to the newer version of the application, it is a straightforward matter to revert to the earlier application and undertake analysis of what transpired.

Case study 1: A simple case of a common library

The customer application is one step more complex than a simple, self-contained program. Each of the five applications in this case study uses a common set of underlying utility routines that are coded in Macro-32 (the assembler language for the VAX-11/780) and ANSI C.

The first step is to rebuild the original application on the Alpha by using a shareable library form of the utility routines rather than a set of object modules statically included at link time.

The applications are maintained by a different staff than the maintainers of the underlying utility library. Each group has its own schedules, budgets, and commitments. The goal is to transition this code base, which was originally designed for a VAX system more than 15 years ago, into a form that executes natively (without translation) on the HP Integrity platform. Of prime importance is that the transition to HP Integrity systems does not affect the use of the application for production on a daily basis.

The first substantive step is for the two groups to run their respective images through the Alpha Environment Software Translator (AEST), and to verify that the translated images function correctly. This step serves three purposes:

- To provide a baseline of functionality for the converted images.
- To permit production use to be transitioned to Integrity servers while the actual migration work is underway.
- To keep the system as a whole available throughout the course of the source modifications, thereby allowing both development groups to work in parallel without cross-interference.[56]

First, the utility library is translated using the AEST command:

```
$ AEST UTILITY_LIBRARY
```

Next, each application that is linked against the library is translated:

```
$ AEST APPLICATION1
$ AEST APPLICATION2
$ AEST APPLICATION3
$ AEST APPLICATION4
$ AEST APPLICATION5
```

The translated versions of both the library and the applications mutually reinforce the abilities of their developers to migrate the entire environment to the HP Integrity servers. It is also possible for the native versions of the executables to be integrated into the production environment on a step-by-step basis. If necessary, this integration can be accomplished on a group-by-group or individual basis.[57]

[56] Application developers can use the translated version of the utility routines to support their migration and testing. Utility developers can use the translated versions of the applications as the necessary foils for the native version of the utility library.

[57] The use of group-specific or job-specific logical names provides the foundation of this capability. See [Gezelter2004].

Case study 2: A large, well-maintained source collection

C-Kermit[58] is a good example of a program with a long, productive life. C-Kermit has been through several versions (Version 8.0 being the latest) and has been maintained by a loose confederation of developers. As an example, C-Kermit consists of a collection of source files, as shown in Table 2.

	Files	Lines	Blocks
*.c	41	277,376	18,327
*.h	22	17,409	1,018
Total	63	294,785	19,345

Table 2 - C-Kermit Version 8.0 Source Base

A casual review of the sources reveals a common problem with older source collections: the extensive use of variables with global scope.[59] These pervasive global variables make it difficult to divide the program into a series of self-contained shareable libraries. Fortunately for the C-Kermit maintainers, the conventions of ANSI C are well enforced, and the source was relatively straightforward to natively recompile for the Itanium architecture.

Suppose the source base had not been strictly ANSI C, and quick recompilation was not an option. Would translation have resulted in a production-usable utility?

A straightforward binary translation of the Alpha executable into an Itanium executable was done as an experiment. The performance of that executable was approximately 50% of the performance of a native compiled image (0.29 seconds versus 0.11 seconds; see Table 3) for the transfer of the distribution of INFO-ZIP (6,874 blocks); but that is quite acceptable.

Case study 3: Multiple, interdependent underlying libraries

Case study 1 illustrates the ease with which this approach decouples the physical aspects of the migration from the software engineering effort. The case of C-Kermit (case study 2) is a relatively simple one and almost does not seem worth the effort. Expanding case study 1 to include a larger set of libraries and multiple applications, with varying sets of the libraries begins to illustrate the complexities that can occur.

The apparent simplicity of case study 1 becomes a far more complex scheduling problem when one considers these issues.

More commonly, large applications environments are far more complex with many more interconnections and interdependencies.

Case study 4: A gradual transition of elements

Some applications contain single executable images whose source modules comprise aggregates of tens of thousands of lines in one or more languages. These applications are often maintained by teams of maintainers. The challenge in such a situation is to find a task sequence which allows all of the teams to meet their objectives.

The combination of the image translation facility, the Translated Image Environment, and the user-by-user (or finer) control over which shareable library is used uniquely provide significant leverage for

[58] Maintained by the Kermit Project located at Columbia University. See `http://www.columbia.edu/kermit`.

[59] Variables with a global scope are defined in C/C++ outside the scope of a single procedure. Within the compilation unit, they are visible to all modules. Outside of the compilation unit, they are referenced using the *extern* keyword. The references are then resolved by the Linker utility. Fortran, COBOL, and other languages have mechanisms with similar effects but different syntax.

OpenVMS users in managing and controlling this process. An example from a recent client project illustrates the mechanics of how this can be accomplished.

The `ITEMCHECK` program was implemented to cross-check and validate data contained in a series of RMS files containing data from a client relational database. Each RMS file contains data[60] from one table of a PC-resident relational database.

The sources needed to build `ITEMCHECK` are:

- The actual source for the `ITEMCHECK` program
- Some library collections developed for this particular project
- Some library collections developed for other projects

The individual libraries were implemented to be independent of one another. While each library does make use of variables with global scope, the variables are referenced only from within each library. Some interfaces, reminiscent of the C++ object-oriented programming style, are used to set and get these variables when necessary. Other static variables are used internally within libraries and routines and are not exposed directly to outside interfaces.

It is a simple matter to break the different components into a main image and eight shareable images. Each of the images can then be translated individually. The source base for `ITEMCHECK` and its underlying utilities is not particularly large, but it is large enough to be illustrative.

```
$ AEST ITEMCHECK
```

Each shareable image can now be translated individually:

```
$ AEST ITEMCHECK_SHRLIB
$ AEST DEBUGTRACESHR
$ AEST GRAPHICSSHR
$ AEST INDEXSHR
$ AEST ITEMSHR
$ AEST ORDERSHR
$ AEST OUTLINESHR
$ AEST PRODUCTSHR
$ AEST READKEYEDSHR
$ AEST RMSSEQUENTIALSHR
$ AEST UTILITIESSHR
```

The resulting collection of shareable images can be executed on an HP Integrity server without native recompilation. On an individual basis, each library can be transitioned from its translated version to its natively compiled version, if need be, on a user-by-user or application-by-application basis, until the entire assemblage has been compiled natively and fully qualified.

On a project basis, this provides a large number of alternative paths for project management.[61] Thus, a wide variety of benchmarks can be run in this case.[62] A sampling of these benchmarks (Table 3) shows the performance of this program when processing a representative dataset containing 1,000,000 line items comprising 500,000 orders.

[60] Because of nondisclosure agreements, the low-level details of the code cannot be released. The performance numbers contained in this paper represent runs with synthetic data comparable in character to the actual client data.

[61] Since there are a total of nine images (one main image and eight shareable libraries), there are a total of 512 (2^9) different combinations. This represents a dramatic increase in management flexibility and project scheduling.

[62] To be precise, 516: 512 combinations of the translated/natively compiled shareable libraries, the shareable image on Integrity servers without support for translated images, the single native image on Integrity, the original single native image on Alpha, and the shareable image component on Alpha.

AlphaStation 200 4/233; 384 MB, 2 RZ-29 SPECint92: 157.7

	CPU	Buffered IO	Direct IO	Virtual Size	Page Faults
Tare	0.16	58	8	171,584	173
C-Kermit 8.0 (Client side)[63]	17.73	19,327	40	184,512	749
ITEMCHECK					
Single Image	2,305.88	167	59,551	175,200	462
Shareable Images	2,462.81	197	59,582	175,200	490

Integrity rx2600; 1.4 GHz/1.5MB Cache; 2 GB; 2 36G SPECint2000 1170+[64]

	CPU	Buffered IO	Direct IO	Virtual Size	Page Faults
Tare	0.01	48	15	176,624	180
C-Kermit 8.0 (Client side) [65]					
Translated	0.29	19,868	57	198,288	764
Native	0.11	20,240	48	198,288	764
ITEMCHECK					
Single Translated Image	5,978.47	147	59,500	198,944	714
Translated Shareable Images	3,501.47	170	59,497	201,264	805
Assortments of Native/Translated Images					
Translated Main Program					
Mask 194	735.37	173	59,599	198,944	752
Mask 38	2,262.38	182	59,501	198,944	753
Mask 9	1,365.09	175	59,500	198,944	782
Native Main Program					
Mask194	179.41	180	59,501	198,944	750
Mask 38	2,016.53	177	59,500	198,944	752
Mask 9	823.49	170	59,499	198,944	779
Native Shareable Images	175.49	151	59,491	198,944	570
Single Native Image	163.01	121	59,482	198,944	533

Table 3 - Performance comparison of translated and native images[66]

As can be seen by the sampling of benchmarks, performance varies over a significant range depending on the usage of the various libraries. While the best performance is undoubtedly achieved by fully native images, all performance tests are within an acceptable band. The use of shareable images does not make a significant difference as compared with the use of single images. Additionally, the identification of the actual source of high CPU utilization allows the conversion of that aspect of the program to natively compiled code on a priority basis, allowing engineering effort to be focused where it produces the largest payoff.

Summary

OpenVMS permits the staged conversion of production environments from the Alpha or VAX architectures to the Integrity platform without the need for interruptions, cold cutovers, or other high-risk strategies.

[63] Transfer of INFO-ZIP.ZIP (Distribution file for INFO-ZIP; 6,874 blocks)
[64] SPECint2000 result for rx 2620-2 (1.3GHz/2MB Itanium 2) from [SPEC2004]
[65] Transfer of INFO-ZIP.ZIP (Distribution file for INFO-ZIP; 6,874 blocks)
[66] For complete datasets from the test series see `http://www.rlgsc.com/publications/vmstechjournal/migrationstrategies.html`.

The features of OpenVMS allow the transition from older architectures to the Itanium architecture in a well-controlled fashion, consistent with technical requirements and corporate resources, without compromising ongoing 24 x 7 operations.

References

Bosworth, S., and Kabay, M., *Computer Security Handbook,* 4th Edition, John Wiley and Sons, 2002 [Bosworth2002].

"Business as Usual While Moving an Eight-Story Steel-Frame Building," *Engineering News-Record,* July 2, 1931.

Cocke, J., and Markstein, V., "The Evolution of RISC Technology at IBM," *IBM Journal of Research and Development,* Volume 34, Number 1 [Cocke1990].

da Cruz, F., and Gianone, C., *Using C-Kermit Communication Software, Second Edition*, Digital Press, 1997 [daCruz1997].

D'Antoni, G., "Porting OpenVMS Applications to Itanium," Compaq Enterprise Technology Symposium (CETS), 2001.

Dobberpubl, et al., "The MicroVAX 78032 Chip: A 32-bit Microprocessor," *Digital Technical Journal,* Volume 2, March 1986 [Dobberpubl1986].

Gezelter, R., "Alpha/Itanium Issues," July 2001; see `http://www.rlgsc.com/alphaitanium.html`.

Gezelter, R., "The Third Porting: Applying Past Lessons to the Alpha/Itanium Transition," Compaq Enterprise Technology Symposium (CETS), September 2001 [Gezelter2001].

Gezelter, R., "Inheritance Based Environments for OpenVMS Systems and OpenVMS Clusters," *OpenVMS Technical Journal,* Volume 2, February 2004 [Gezelter2004].

Gezelter, R., "Migrating OpenVMS Storage Environments without Interruption/Disruption," HP Enterprise Technology Forum, September 2005 [Gezelter2005].

Gezelter, R., "Code Portability and Related Issues for EPIC," IEEE Distinguished Visitor Vermont Speaking Tour, Dartmouth College, September 2005 [Gezelter2005a].

Gezelter, R., "Strategies for Enterprise Migration from Alpha and VAX to HP Integrity," Encompass Canada Local User Group and National Research Council (Canada), Institute for Information Technology, November 2006 [Gezelter2006].

Goldenberg, R., Kenah, L., *VAX/VMS Internals and Data Structures, Version 5.2*, Digital Press, 1991.

Goldenberg, R., Saravanan, S., *VMS for Alpha Platforms: Internals and Data Structures*, Preliminary Edition, Volume 3, Digital Press, 1993.

Goldenberg, R., Dumas, D., Saravanan, S., *OpenVMS Alpha Internals: Scheduling and Process Control,* Digital Press, 1997.

Gosling, J., Joy, B., Steele, G., *The Java Language Specification,* Addison-Wesley, 1996 [Gosling1996].

Grant, C., "Porting OpenVMS to HP Integrity Servers," *OpenVMS Technical Journal,* Volume 5, June 2005.

Gursha, J., *High Performance Cluster Configuration System Management,* Digital Press, 1997

IEEE Standards Committee 754, *IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Standard 754-1985,* IEEE, New York, 1985 (reprinted in SIGPLAN Notes, 22(2):9-25, 1987.

Intel® Itanium® Architecture Software Developer's Manual: Application Architecture," Revision 2.2, Volume 1, Intel Corporation, July 2006.

Intel® Itanium® Architecture Software Developer's Manual: Instruction Set Reference," Revision 2.2, Volume 3, Intel Corporation, July 2006.

Levine, D., *Software Development and Quality Assurance,* Chapter 25, [Bosworth2002].

Microcomputer Processors, Digital Equipment Corporation, 1978 [Digital1978c].

Reagan, J., "Porting the MACRO-32 Compiler to OpenVMS I64," *OpenVMS Technical Journal*, Volume 6, February 2005.

Rav, B., and Schlansker, M.S., "Explicitly Parallel Instruction Computing," *Computer,* February 2000 [Schlansker2000].

Rodin, G., "The 801 Minicomputer," *IBM Journal of Research and Development,* Volume 27, 1983 [Rodin1983].

Rusu, S., and Singer, G., "The First IA64 Microprocessor", *IEEE Journal of Solid State Circuits,* Volume 35, Number 11, November 2000.

Sites, R. (ed.), *Alpha Architecture Reference Manual,* Digital Press, 1992 [Sites1992].

Sites, R., "Binary Translation," *Digital Technical Journal,* Volume 4, Number 4, 1992 [Sites1992].

Strecker, W., "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family," *Proceedings of the National Computer Conference, 1978* [Strecker1978].

TechWise Research, Inc., *Quantifying the Value of Availability: A Detailed Comparison of Four Different RISK-Based Cluster Solutions Designed to Provide High Availability,* Version 1.1a, June 2000 [TechWise2000].

TIS Committee, *Toolkit Interface Standards (TIS) Executable and Linking Format (ELF) Version 1.2*, May 1995; from http://www.x86.org/ftp/manuals/tools/elf.pdf, December 2006.

van der Burg, J., "Disk Partitioning on OpenVMS: LDDRIVER," *OpenVMS Technical Journal,* Volume 6, June 2005.

VAX-11 Architecture Handbook, Digital Equipment Corporation, 1978 [Digital1978].

VAX-11 Software Handbook, Digital Equipment Corporation,1978 [Digital1978b].

Zahir, R., et al., "OS and Compiler Considerations in the Design of the IA-64 Architecture," *Ninth International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS-IX),* Association for Computing Machinery.

OpenVMS manuals

HP OpenVMS DCL Dictionary: A–M (September 2003, Order number AA-PV5KJ-TK)

HP OpenVMS DCL Dictionary: N–Z (September 2003, Order number AA-PV5LJ-TK)

HP OpenVMS Systems Manager's Manual, Volume 1: Essentials (September 2003, Order number AA-PV5MH-TK)

HP OpenVMS Systems Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems (September 2003, Order number AA-PV5NH-TK)

OpenVMS Guide to System Security (June 2002, Order number AA-Q2HLF-TE)

OpenVMS User Manual (January 1999, Order number AA-PV5JD-TK)

OpenVMS Version 7.2 New Features Manual (January 1999, Order number AA-QSBFC-TE)

HP OpenVMS Programming Concepts Manual, Volume I (January 2005, Order number AA-RNSHD-TE)

HP OpenVMS Programming Concepts Manual, Volume II (January 2005, Order number AA-PV67H-TE)

HP OpenVMS LINKER Manual (July 2006, Order number BA554-90004)

HP OpenVMS Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers (January 2005, Order number BA-442-90001)

HP OpenVMS Migration Software for Alpha to Integrity Servers: Guide to Translating Images (February 2005)

HP OpenVMS Migration Software for Alpha to Integrity Servers: Release Notes (February 2005)

VAX MACRO and Instruction Set Reference Manual (April 2001, Order number AA-PS6GD-TE)

Further reading

Davis, R., *VAXcluster Principles*, Digital Press, 1993.

Acknowledgments

I would like to thank the numerous people who generously took the time to speak with me about the technologies involved in the architectural transitions in the OpenVMS community; the initial transition from the 16-bit PDP-11 RSX-family to VAX/VMS; the transition from the CISC VAX architecture to the RISC Alpha architecture; and the transition from the RISC Alpha to the EPIC Itanium architecture. I would like to thank members of OpenVMS Engineering, including Andrew Goldstein, Gaitan D'Antoni, and others who contributed recollections and pointers to information. I would also like to thank John Streiff, James Gursha, Jerrold Leichter, Gideon Eisenstadter, and others who read drafts and contributed their comments. I would also like to thank Fern Hertzberg for her assistance in organizing and copyediting this paper.

Biography

Robert Gezelter, CDP, CSA, CSE, Software Consultant, guest lecturer and technical facilitator has more than 29 years of international consulting experience in private and public sectors. He has worked with OpenVMS since the initial release of VMS in 1978, and with OpenVMS Cluster systems since their announcement in 1982. He has worked with portable software, translation, and cross compilers since 1975.

Mr. Gezelter received his BA and MS degrees in Computer Science from New York University. He also holds Hewlett-Packard's CSA and CSE accreditations relating to OpenVMS.
Mr. Gezelter is a regular guest speaker at technical conferences worldwide such as the HP Technology Forum and Encompass (formerly DECUS) events. His articles have appeared in the Network World, Open Systems Today, Digital Systems Journal, Digital News, and Hardcopy. He is also a Contributing Editor to the Computer Security Handbook, 4th Edition (Wiley, 2002) and the author of two chapters in the Handbook of Information Security (Wiley, 2005), including the chapter on *OpenVMS Security*. Many of his publications and speeches are available through his firm's www site at http://www.rlgsc.com.

He is a Senior Member of the IEEE, and a member of Infragard, Encompass, and International Association of Software Architects. He is an alumnus of the IEEE Computer Society's Distinguished Visitors Program and is a Director of the New York City chapter of IASA.
His firm's consulting practice emphasizes in-depth technical expertise in computer architectures, operating systems, networks, security, APIs, and related matters.
His clients range from the Fortune 10 to small businesses, locally, nationally, and internationally on matters spanning the range from individual questions to major projects.
He can be reached via his firm's www site at http://www.rlgsc.com.