Resume of Alice Gebura – Writing Sample

OpenVMS floating-point arithmetic on the Intel Itanium architecture — White Paper

Audience:

Programmers porting OpenVMS applications from the Alpha platform to the Itanium platform.

Project Description:

This white paper discusses how floating point formats differ between the Alpha and Itanium platforms and the options programmers have for reconciling these differences. HP/Compaq provided the information to third party software companies porting their OpenVMS applications.

OpenVMS floating-point arithmetic on the Itanium® architecture

executive summary

hp is bringing its *OpenVMS* operating system, middleware, and application portfolio to the Intel® Itanium® architecture. The Itanium architecture has a 64-bit model and basic system functions similar to the Alpha chip. However, there are some implementation differences between the two platforms that might affect user-written applications.

One of the differences is the availability of hardware-supported floating-point formats. The Itanium architecture implements floating-point arithmetic in hardware using the IEEE floating-point formats, including IEEE single and IEEE double. The Alpha architecture supports both IEEE and VAX floating-point formats in hardware, and *OpenVMS* compilers generate code using the VAX formats by default, with options (on Alpha) to use IEEE formats. Irrespective of whether it was originally written for VAX or Alpha, an *OpenVMS* application that uses the default VAX floating-point formats needs to produce equivalent behavior on the Itanium architecture using IEEE formats at the lowest level.

This white paper is for developers moving *OpenVMS* applications that use VAX floating-point formats. It describes how VAX floating-point formats will be supported on the Itanium architecture, as well as how and why data might be affected.

hp solution

On *OpenVMS* VAX and *OpenVMS* Alpha, VAX float is the default. VAX format data is assumed and VAX floating instructions are used.

On *OpenVMS* Alpha, you can specify the compiler option /FLOAT=IEEE. In this case, IEEE format data is assumed and IEEE floating instructions are used.

On *OpenVMS* Itanium, IEEE float is the default. IEEE format data is assumed and IEEE floating instructions are used.

On *OpenVMS* Itanium, you can specify the compiler option /FLOAT=D_FLOAT or /FLOAT=G_FLOAT.

VAX floating-point formats are supported on the Itanium architecture by converting them to IEEE single and IEEE double floating types. By default, this is a transparent process that will not impact most applications. All you need to do is recompile your application. hp is providing compilers for C, C++, Fortran, BASIC, PASCAL, and COBOL, all with the same floating-point options. The compiler-specific documentation will describe all the floating-point options in detail. Because IEEE floating-point format will be the default, unless your build explicitly specifies VAX floating-point format options, a simple rebuild for Itanium will use the native IEEE formats directly. For the large class of programs that do not directly depend on the VAX formats for correct operation, this is the most desirable way to build for Itanium.

When you compile an *OpenVMS* application that specifies an option to use VAX floating-point on the Itanium architecture, the compiler automatically generates code for converting floating-point formats. Whenever the application performs a sequence of arithmetic operations, this code does the following:

1. Converts VAX floating-point formats to either IEEE single or IEEE double floating-point formats.
2. Performs arithmetic operations in IEEE floating-point arithmetic.
3. Converts the resulting data from IEEE formats back to VAX formats.

Note that where no arithmetic operations are performed (VAX float fetches followed by stores), no conversion will occur. The code handles such situations as moves.

VAX floating-point formats have the same number of bits and precision as their equivalent IEEE floating-point formats. For most applications the conversion process will be transparent and thus a non-issue.

In a few cases, arithmetic calculations might have different results because of the following differences between VAX and IEEE formats (described in detail later in this paper):

- Values of numbers represented
- Rounding rules
- Exception behavior

These differences might cause problems for applications that do any of the following:

- Depend on exception behavior
- Measure the limits of floating-point behaviors
- Implement algorithms at maximal processor-specific accuracy
- Perform low-level emulations of other floating-point processors
- Use direct equality comparisons between floating-point values, instead of appropriately ranged comparisons (a practice that is extremely vulnerable to changes in compiler version or compiler options, as well as architecture)

You can test an application's behavior with IEEE floating-point values today by compiling it on an *OpenVMS* Alpha system with an IEEE qualifier. If that produces acceptable results, you should simply build the application on the Itanium-based platform (and Alpha, if you wish) using the same qualifier.

If you determine that simply recompiling with an IEEE qualifier is not sufficient because your application depends on the binary representation of floating point values, then you should first try building for Itanium by specifying the VAX floating-point option that was in effect for your VAX or Alpha build. This causes the representation seen by your code and on disk to remain unchanged, with some additional run-time cost for the conversions generated by the compiler. If this is not an efficient approach for your application, you can convert VAX floating-point binary data in disk files to IEEE floating-point formats prior to moving the application to the Itanium platform. This entails writing

and running a data conversion utility. To support customers who choose this approach, hp will provide algorithms and routines to make this task easier.

values of numbers represented

The set of numbers exactly represented by VAX and IEEE floating-point formats is different because of the different ways in which each represents the exponent. Table 1 shows the range and precision for each floating-point format.

floating-point format		lowest normalized value[1]	highest normalized value[2]
Single precision	IEEE single	$1.17549435E^{-38}$	$3.402823E^{38}$
	VAX F	$2.9387359E^{-39}$	$1.701411E^{38}$
Double precision	IEEE double	$2.2250738585072014E^{-308}$	$1.797693134862316E^{308}$
	VAX G	$5.562684646268003E^{-309}$	$8.988465674311579E^{307}$
	VAX D	$2.9387358705571877E^{-39}$	$1.70141183460469229E^{38}$

1. Smallest normalized value greater than zero
2. Maximum value before overflow

table 1: range matrix

Floating-point source code using VAX format should produce nearly equivalent results, within the accuracy of the types and language rules, when compiled for the Itanium architecture using IEEE floating-point format. If the operands and result of a VAX floating-point operation fall between the lowest and highest normalized IEEE values, the corresponding IEEE float result will usually be identical to the VAX result. In the case of a difference, the IEEE float result will have at least the same accuracy as the VAX float result. The situation is more complicated for VAX float operations involving values less than the corresponding lowest normalized IEEE value. Most programs will not be sensitive to these differences involving arithmetic with underflows.

An application that is extremely sensitive to the last bits of precision produced by individual VAX floating-point operations might exhibit unexpected behavior. VAX instructions that result in values too small to be represented with normalization, produce a zero result. The default IEEE emulation of VAX instructions preserves denormal intermediate results instead of forcing them to zero immediately. It is only when the final result of a computation is converted to VAX format for storage in a variable, that the VAX emulation forces an IEEE denormal to zero.

If subsequent operations in a computation multiply an intermediate denormal result by a very large value, the emulation might produce a non-zero VAX floating result, where the same computation with VAX instructions would have produced zero. The worst case is that you will lose two bits of precision for numbers near the VAX underflow threshold.

rounding rules

Rounding might sometimes produce different results. This is caused by the differences in biased rounding used by VAX floating-point and unbiased rounding used by IEEE floating-point.

In VAX floating-point operations, the result half way between two representable values is always rounded up (i.e. by adding half the value of the least significant bit retained to the magnitude of the result). In IEEE floating-point operations, the result half way between two representable values is rounded to the nearest even value (i.e. when exactly halfway between two representable values, the low order bit of the result is forced to zero).

Programs might see different results, but they are equally correct.

exception behavior

Because the underflow threshold is four times larger with IEEE than it is with VAX, some operations that currently do not underflow with VAX floating-point might underflow when the operation is done in IEEE floating-point. The VAX, Alpha, and Itanium architectures behave differently in the presence of floating underflow. For the few programs that depend on operations that underflow, *OpenVMS* provides several options to control underflow behavior. An *OpenVMS* programmer working with VAX formats can choose to have operations that underflow generate one of the following:

- An exception trap

  The underflow exception option lets you detect when an underflow might cause a difference in behavior. On a VAX or Alpha system, underflow exceptions are generated when a result is less than the lowest normalized VAX float result. On an Itanium system, underflow exceptions are generated when a result is less than the lowest normalized IEEE float result.

- A zero result

  The flush-to-zero option makes all operations with an underflow produce a zero result. On a VAX or Alpha system, flush-to-zero happens when a result is less than the lowest normalized VAX float value. On an Itanium-based system, flush-to-zero happens when a result is less than the lowest normalized IEEE float value.

- A denormalized intermediate result

  The denormalized intermediate option is available only on *OpenVMS* Itanium-based systems. This option lets *OpenVMS* Itanium systems produce intermediate expression values smaller than the lowest normalized IEEE float value. These denormalized intermediate values are less precise. Denormalized intermediate values smaller than the lowest normalized VAX float value are flushed-to-zero when the value is stored in a VAX float variable. Intermediate denormalized values larger than the lowest normalized VAX value will lose no more than two low-order bits of precision when compared to the corresponding VAX float intermediate value.

Overflow traps will occur *less* frequently on the Itanium architecture than on Alpha because the IEEE floating-point format supports values two times larger than the VAX floating-point format. An intermediate result that would overflow on VAX might not overflow on IEEE. If the final result of that computation results in a value that can be represented in VAX format, the computation will not raise an exception on IEEE, where it would have raised an overflow on VAX. If the final result is too large for VAX, the overflow will be raised in IEEE when the conversion to VAX takes place.

These differences are similar to those seen in VAX applications that depended on VAX D floating-point when they were ported to Alpha D-float implementation.

If an application is sensitive to the exact point in a computation at which an exception will be raised, it might exhibit unexpected behavior. Otherwise, floating-point codes using VAX format that do not depend upon the raising of an exception at a specific point in a computation to produce a correct result should produce nearly equivalent results.

performance considerations

For applications using VAX floating-point formats, performance will vary from application to application. Run-time performance could be affected by data conversion that occurs “on the fly” for an arithmetic operation because conversion requires several additional instructions. There is a run-time cost to each conversion that occurs when a variable’s value is fetched from memory or a computed result is stored back to memory.

Reading and writing to disk is slow relative to the computations done on the high-speed Itanium processor. Applications that require VAX floating-point format for on-disk representation are less likely to notice the cost of converting VAX floating-point data than those that are compute-bound.

Commercial applications that are I/O intensive or scientific applications that are computation intensive should be aware of these factors. If there are continual manipulations or calculations on floating-point data after reading in external VAX floating-point data, you might want to modify such an application to do the following:

- Convert external VAX floating-point data to IEEE format when read in
- Use native IEEE representation for variables and computations within the program
- Convert back to VAX floating-point when the data is written back out

An applications that is performance critical should definitely be converted to use IEEE formats.

summary

If you are moving an *OpenVMS* application to Itanium-based systems and the application performs floating-point arithmetic, you need to know that VAX floating-point formats are not directly supported by the Itanium architecture. In most cases, simply recompiling using the default IEEE floating point options for *OpenVMS* Itanium-based systems will produce equivalent results to using VAX formats on *OpenVMS* VAX or Alpha systems, and at maximum efficiency.

For programs that directly depend on the VAX formats, explicitly specifying the same VAX format option on Itanium that was in effect for the VAX or Alpha build of the program will usually produce functionally equivalent results, with some amount of overhead caused by on-the-fly conversions. If the overhead of on-the-fly conversions proves to be too high, data can be converted in bulk before the programs starts, the program run using native IEEE formats, and the results converted back when the program completes. Or a one-time application data migration might be most appropriate. Only those rare programs that heavily rely on the limits or detailed exception behaviors of the VAX formats are likely to require pervasive source code changes in moving to the Itanium architecture.

for more information

resource	**Location**
IA-64 Software Conventions and Runtime Architecture Guide	http://www.intel.com/design/Itanium/Downloads/24535802.pdf
IEEE Standard 754-1985 for Floating-point Computations	http://grouper.ieee.org/groups/754/faq.html#obtaining
Intel® Itanium® Architecture Software Developer's Manual	http://www.intel.com/design/itanium/manuals/
Itanium® Processor Floating-point Software Assistance Handler	http://www.intel.com/design/itanium/downloads/245415.htm
Compaq Standard 032-0 VAX Architecture Standard	Order Number: EL-00032-00-0000
Alpha Architecture Handbook	http://www.support.compaq.com/alpha-tools/documentation/current/alpha-archt/alpha-architecture.pdf