Porting to HP OpenVMS Integrity

Christian Moser
OpenVMS Engineering, Hewlett-Packard
Finland Development Center
cmos@hp.com

Agenda

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics

Porting Goals

- Provide an operating system environment, development tools, and documentation to make porting as easy as possible
  - Full port of the Operating System, Runtime Libraries, development tools and most layered products
  - Recompile, relink, requalify
- Use our experiences porting the operating system to make it easier for others to port their applications
  - Internal layered product groups, partners, and customers

Porting to OpenVMS I64

- Porting applications to I64 is easy
- Usually all that is required is to recompile/ relink and requalify the application.
  - Privileged code may require more effort
  - Porting 100,000 lines of C code did not require even one change

HOWEVER

Porting to OpenVMS I64

- MANY!!! Things have changed in the O/S
  - Different primitives
  - Different default floating point standard
  - New compilers
  - New image format
  - New calling standard
  - No console/PAL code

Most changes are transparent but these changes might affect your application

Porting to OpenVMS I64

- The purpose of this presentation is to use the experience we gained porting the base O/S, to ease the porting of your application.
  - It is a non-goal to discuss changes made to the O/S (like set predicate register x before calling the system service dispatcher)
  - We are trying to cover the most common issues. It is possible that you will encountered something not covered here.
- Might be used as a cookbook of things to look for before starting the port of your application.
- Again for most of the people….recompile and relink will do……
- All of the coding examples are working programs. We encourage you to test them yourself and get real experience…

How do I start?

- How do I start porting my application?
  - There are several approaches:
    - Re-examine the application for potential “hot spots”
    - Compile/link and see what’s broken
    - Compile and examine new messages
- There is no right approach, take the one that you feel most comfortable with

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics
- Using the XDELTA debugger
- Next steps…

Conditionalized code

- Macro

```
.if DF ALPHA          .if DF IA64
.endc                 .endc
```

- C

```
#ifdef __alpha        #ifdef __ia64
#endif                #endif
```

- Bliss

```
%IF ALPHA %THEN   %IF IA64 %THEN
 %FI               %FI
```

Conditionalized code – example

```
IPL31> type arch_test.c

#include <stdio.h>
#include <arch_defs.h>
void main()
{
#ifdef __vax
        printf("This will be printed on VAX\n");
#endif
#ifdef ALPHA
        printf("This will be printed on Alpha\n");
#endif
#ifdef __ia64
        printf("This will be printed on IA64\n");
#endif
#ifndef __vax
        printf("This program is not running on VAX");
#endif
}
```

Conditionalized code

Executed on IA64 system

```
IPL31> write sys$output f$getsyi("arch_name")
IA64
IPL31> r arch_test
This will be printed on IA64
This program is not running on VAX
IPL31>
```

Executed on Alpha system

```
MIKAXP> write sys$output f$getsyi("arch_name")
Alpha
MIKAXP> r arch_test
This will be printed on Alpha
This program is not running on VAX
```

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics

IEEE floating- point

- This is one of the biggest porting issues.
- Itanium supports only IEEE floating-point in hardware
- We have updated the appropriate Runtime Libraries to add IEEE interfaces where needed
- White Paper with technical details about the differences between VAX Float and IEEE Float is available at http://www.hp.com/products1/evolution/alpha_retaintrust/openvms/resources.html

IEEE floating point

- All compilers that currently support F, D, G, S, T, and X (S and T are native IEEE formats) will continue to do so on Itanium architecture
  - IEEE is the default
  - VAX floating point formats will be supported when specified as a switch to the compilers
  - The compilers generate code to call conversion routines (performance hit)

IEEE floating- point

- To use IEEE on both Alpha and IPF, developers have to change the source to use S & T versions of the APIs.
  - Some functions (like sin, cos,….) already know how to handle IEEE and require no changes to the application.

Q: My application uses F-float. I'm currently porting it to I64, I'm excited enough about the new architecture and I don't want to make any source changes for now. What can I do?

A: Have no fear….HP OpenVMS engineering is here…

Let's take a look at a real example……

```
$ ty float_test.c
#include <stdio.h>
#include <libdtdef.h>
#include <descrip>
#include <ssdef>

// prototypes
int lib$cvtf_to_internal_time();
int sys$fao();
int lib$put_output();
int lib$signal();

static $DESCRIPTOR (fao_desc, "Converted delta time: !%T");

main () {

 float  f1;
 unsigned long long int delta1;
 int    status;
 char   output[50]={0};
 struct dsc$descriptor_s outdesc;
 short int length;

 //init the descriptor
 outdesc.dsc$w_length  = sizeof(output);
 outdesc.dsc$a_pointer = (char *)&output;
 outdesc.dsc$b_class   = DSC$K_CLASS_S;
 outdesc.dsc$b_dtype   = DSC$K_DTYPE_T;

 f1 = 156.45;

 printf("This program converts %f seconds to delta time\n", f1);

 status = lib$cvtf_to_internal_time(&LIB$K_DELTA_SECONDS_F, &f1, &delta1);

 if (!(status&1))
        lib$signal(status);

 if ((sys$fao(&fao_desc,&length,&outdesc,&delta1))&1)
        lib$put_output(&outdesc);
}
```

Convert seconds to delta time

Executed on Alpha:

```
Alpha> cc float_test
Alpha> link float_test
Alpha> r float_test
This program converts 156.449997 seconds to delta time
Converted delta time: 00:02:36.44
```

Executed on IA64:

```
I64> cc float_test
I64> link float_test
I64> r float_test
This program converts 156.449997 seconds to delta time
%LIB-F-IVTIME, invalid time passed in, or computed
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine              line      rel PC           abs PC
FLOAT_TEST                                     0000000000010240 0000000000010240
FLOAT_TEST                                     00000000000100D0 00000000000100D0
                                               0000000000000000 FFFFFFFF80B1A030
                                               0000000000000000 000000007AE1BEE0
```

Compiled again with the /FLOAT qualifier

```
I64> cc float_test/float=g_float
I64> link float_test
I64> r float_test
This program converts 156.449997 seconds to delta time
Converted delta time: 00:02:36.44
IPL31>
```

Note the program would fail on Alpha as well if compiled with ieee_float

```
AXP> cc float_test/float=ieee
AXP> link float_test
AXP> r float_test
This program converts 156.449997 seconds to delta time
%LIB-F-IVTIME, invalid time passed in, or computed
%TRACE-F-TRACEBACK, symbolic stack dump follows
```

image	module	routine	line	rel PC	abs PC
FLOAT_TEST	FLOAT_TEST	main	4514	0000000000000174	0000000000030174
FLOAT_TEST	FLOAT_TEST	__main	0	0000000000000064	0000000000030064
			0	FFFFFFFF8025DE94	FFFFFFFF8025DE94

IEEE floating point example

- This program relies on the binary representation of the floating point value and therefore it fails on IA64.
- Compiled on IA64 with /FLOAT=G_FLOAT forced the compiler to use the default Alpha representation. No code changes are required in this case but there is some runtime cost.
- To use IEEE floating point representation, this program should be modified to use LIB$CVTS_TO_INTERNAL_TIME
- LIB$WAIT is another common example where floating point conversion may become an issue…let's take a look….

```
Alpha> ty wait.c
#include <stdio.h>
main()
{
float wait=7.0;

        printf("Waiting 7 seconds\n");
        lib$wait(&wait,0,0);
        printf("I'm done wainting..ciao...\n");

        return 0;
}
```

Executed on Alpha:

```
Alpha> cc wait
Alpha> link wait
Alpha> r wait
Waiting 7 seconds
I'm done wainting..ciao...
```

Executed on I64:

```
I64> cc wait
I64> link wait
I64> r wait
Waiting 7 seconds
%SYSTEM-F-FLTINV, floating invalid operation, PC=FFFFFFFF82142760, PS=0000001B
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine                line      rel PC           abs PC
LIBRTL                                              000000000016C752 FFFFFFFF82142752
LIBRTL                                              000000000020F430 FFFFFFFF821E5430
WAIT                                                0000000000010250 0000000000010250
WAIT                                                0000000000010180 0000000000010180
                                                    0000000000000000 FFFFFFFF80B1A030
                                                    0000000000000000 000000007AE1BEE0
```

The default floating point format used by LIB$WAIT is F_FLOAT, which does not match the default floating point format used on I64 (S_FLOAT)

Here is a modified version that will work on both platforms, using the native floating point formats


```
I64> ty wait_common.c
#include <stdio.h>
#include <arch_defs>
#include <libwaitdef>
main()
{
float wait=7.0;
#ifdef __alpha
        int mask = LIB$K_VAX_F;
#endif
#ifdef __ia64
        int mask = LIB$K_IEEE_S;
#endif
        printf("Waiting 7 seconds\n");
        lib$wait(&wait,0,&mask);
        printf("I'm done wainting..ciao...\n");

        return 0;
}
```

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics

Build tools

- The port to Itanium requires that applications will be built using the latest versions of the compilers
  - Some applications being built with older versions might see some issues introduced by changes to the compilers and even bugfixes within the compilers.
  - For example - Older versions of Bliss used to return a value for functions defined NOVALUE (similar to C void)
    - On I64 this has been fixed and some utilities failed
  - You might get away with relying on uninitialized variables, but on I64 you will be severely punished
- It is recommended to build your application on Alpha, using the latest version of the compilers you are using to uncover any hidden bugs/changes. This will make the move to the new platform easier.

Alpha Compilers

- HP recommends that you build your applications on OpenVMS Alpha using the latest versions of the compilers prior to starting your port to OpenVMS I64
- Latest/Next Releases on Alpha Platform
  - C V6.5, C++ V6.5
  - Fortran V7.5 (F90)
  - Basic V1.5
  - COBOL V2.8
  - Java 1.4.2-3
  - Pascal V5.9

OpenVMS on Integrity Server Compilers

- C
  - Itanium® architecture implementation of OpenVMS HP C V6.5 compiler
- C++
  - Based on the same front end compiler technology as HP C++
  - This is not a port of HP C++ V6.5 but it will be able to compile most of the same source code as HP C++ V6.5
- COBOL, BASIC, PASCAL, BLISS
  - Itanium architecture implementations of the current OpenVMS compilers

OpenVMS on Integrity Servers Compiler Plans

- FORTRAN
  - Itanium® architecture implementation of the current OpenVMS Fortran 90 compiler
- Java
  - Itanium architecture implementation of J2SE V1.4.2
- IMACRO
  - Compiles ported VAX Macro-32 code for Itanium architecture
  - Itanium architecture equivalent of AMACRO
- ADA
  - We will be providing an Ada-95 compiler
  - We did not port the existing Ada-83 compiler

Example – Moving from F77 to F90

- When using double precision float (REAL*8) for doing direct assignment (a=5.3)

F77 uses double precision

F90 uses single precision. The result is slightly further away from the real 5.3 value.

- A computation will produce a different result with no error signaled.
- Possible solutions:
  - Fix the coding bug, as the assignment is wrong.
    - Change the assignment to a=5.3D0 or a=5.3_8
    - 5.3D0 works for both F77 and F90
  - Compile using the /ASSUME=FP_CONSTANT switch

Example – Moving from F77 to F90

```
IPL31> ty float.for
       REAL*8              TEST

       TEST = 5.3
       PRINT 100,TEST
 100   FORMAT(F,' assigned as TEST = 5.3 ')

       TEST = 5.3D0
       PRINT 200,TEST
 200   FORMAT(F,' assigned as TEST = 5.3D0')

       END
IPL31> for float
IPL31> link float
IPL31> r float
      5.3000001907348633 assigned as TEST = 5.3
      5.2999999999999998 assigned as TEST = 5.3D0
IPL31> for/assume=fp_const float
IPL31> link float
IPL31> r float
      5.2999999999999998 assigned as TEST = 5.3
      5.2999999999999998 assigned as TEST = 5.3D0
```

Linker

- New /BASE_ADDRESS qualifier
  - Replaces the CLUSTER=,[base-address] option on Alpha
- New SHORT_DATA option
  - Allows you to combine read-only and read-write short data sections into a single segment
  - Eliminates unused space (up to 65,535 bytes) between two (read-only and read-write) segments
- New Alignments for PSECT_ATTRIBUTE option
  - Integer values of 5, 6, 7, and 8 representing the byte alignment indicated as a power of 2. e.g. 2**6 = 64-bit alignment
- It is highly recommended that data reduced object libraries are not referenced in Link command on Itanium® architecture
  - It is necessary for the linker to expand the entire module in memory which can result in BYTLM or PGFLQUOTA issues

- Porting Overview
- Conditionalized code
- IEEE Floating-Point
- Build tools
- Miscellaneous topics

Improperly declared functions and data

- C function declarations that points to objects that are not functions, may work on Alpha but these declarations will not work on IA64
  - Also seen with Bliss and Fortran
- The problem may manifest itself in many ways
  - For whatever reason, the most common symptom is a call to routine CLI$DCL_PARSE that fails with CLI-E-INVTAB

```
external int my_cld();
  status = cli$dcl_parse(&cmdstr, my_cld,
                         lib$get_input, lib$get_input);
external int my_cld;
  status = cli$dcl_parse(&cmdstr, &my_cld,
                         lib$get_input, lib$get_input);
```

Source Code that May Need to Change

- Architecture Specific code
  - All Alpha assembler code must be rewritten
- Conditionalized code
  - Build command files
    - **`$ if .not. Alpha  ! Assumes VAX`**
  - Application source code
    - **`#ifndef (alpha)  // Assumes VAX`**
    - C asm code
- SYS$GOTO_UNWIND system service must be replaced by SYS$GOTO_UNWIND_64
  - OpenVMS I64 requires a 64-bit invocation context
  - SYS$GOTO_UNWIND_64 can be used on Alpha to maintain common source code

Source Code that May Need to Change

- SYS$LKWSET and SYS$LKWSET_64 system services runtime behavior has been modified
  - The entire image, not the specified range of pages, is locked
  - Consider using LIB$LOCK_IMAGE and LIB$UNLOCK_IMAGE for simplicity
- SS$_HPARITH (high performance arithmetic trap) is replaced by SS$_FLTINV (floating point invalid) and SS$_FLTDIV (floating divide by zero)
  - To maintain common code use:
    if ((sigargs[1] == SS$_HPARITH) II (sigargs[1] == SS$_FLTINV) II (sigargs[1] == SS$_FLTDIV))
- Mechanism Array data structure has been changed
  - Standard calling interfaces have not changed
- The Porting Guide contains all of the details

Hardware Model

- The hardware model for all IA64 systems is set to 4096
  - HW model is set to 0 in E8.1 and E8.2

```
I64> write sys$output f$getsyi("hw_name")
HP rx2600  (900MHz/1.5MB)
I64> write sys$output f$getsyi("hw_model")
4096
```

- Any code relying on the hardware model of the system has to change.
- SHOW MEMORY used to determine the page size based on the following algorithm:

```
if (hardware_model>=1024)
        page_size=8192;
else page_size=512;
```

- SHOW MEMORY has been modified to use the page_size item code of $GETSYI system service on all three architectures.

IMACRO

- On I64 the calling standard changed
  - We now use Intel's software conventions
  - IA64 only preserves register R4-R7 across routine calls (Alpha preserves R2-R15)
  - For programs written in high-level languages (like C,Bliss) this difference is not visible.
  - When MACRO-32 programs added you have to start worrying about how to pass register parameters between languages.
  - High-level languages might trash a register IMACRO assumed to be preserved (and vice versa)
  - Please reference the IMACRO porting guide for more details

IMACRO - coding changes

- JSB_ENTRYs that reference R16-R21should be changed to .CALL_ENTRYs that reference n(AP)

```
XFC_COMPARE_QIOS:
          .JSB_ENTRY
          MOVL     (R16),R0
          MOVL     (R17),R1
          EVAX_SUBQ CFS$Q_TOTALQIOS(R0),-
                    CFS$Q_TOTALQIOS(R1),R0
          RSB

XFC_COMPARE_QIOS:
          .CALL_ENTRY
          MOVL     @4(ap),R0
          MOVL     @8(ap),R1
          EVAX_SUBQ CFS$Q_TOTALQIOS(R0),-
                    CFS$Q_TOTALQIOS(R1),R0
          RET
```

- Code that is moving data in R16-R21 and then perform a JSB should be modified to use $SETUP_CALL64 and $CALL64

IMACRO - coding changes

- MACRO32 JSBs to any other language (Bliss/C) routines
  - If IMACRO can't determine the language of a target JSB, the following message will be generated:

```
JSB     (R6)   ;ALLOCATE AND INSERT ENTRY IN SYMBOL TABLE
%IMAC-I-CODGENINF, (1) Indirect JSB with default linkage
```

- .USE_LINKAGE directive with the LANGUAGE=OTHER option tells IMACRO that the target routine is written in a language other than IMACRO. IMACRO will than save and restore R2,R3,R8-R15 around the JSB except for registers in the output mask.

It is recommended to add a USE_LINKAGE statement prior to the JSB call

.use_linkage language=other (or language=macro if the target routine is in MACRO)
JSB (R6)

IMACRO - coding changes

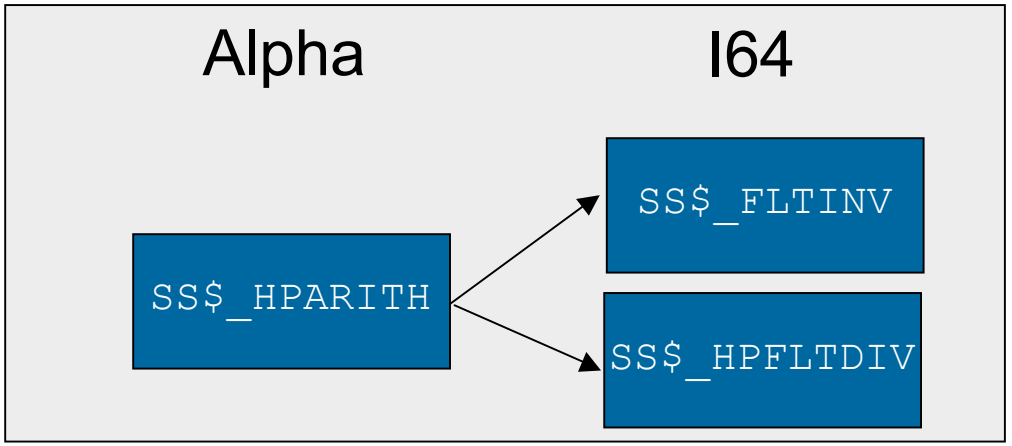
- MACRO32 CALL/CALLG to non-standard routines
  - A non standard routine (written in Bliss C or MACRO) returns a value in a register other than R0 or R1
  - Since IMACRO saves and restores R2,R3,R8-R15, the returned value may be overridden
  - .CALL_LINKAGE or .USE_LINKAGE should be used in every module that calls the non standard routine.
  - For example,
    .CALL_LINKAGE RTN_NAME=FOO$BAR, OUTPUT=<R3,R8,R10>
  - The call_linkage needs only to appear once in every module
  - The .USE_LINKAGE directive will be used only once
  - Here is a small example from DCL, where a MACRO routine is calling a C routine.

```
.IF DF IA64
.use_linkage input=<r0,r1,r2,r3,r8,r9>, output=<r0,r1,r2>,
             language=other
.ENDC
 JSB     DCL$FID_TO_NAME            ; dispatch to the action
                                          ; routine
```

Condition Handlers Use of SS$_HPARITH

On OpenVMS Alpha, SS$_HPARITH is signaled for a number of arithmetic error conditions. On OpenVMS I64, SS$_HPARITH is never signaled for arithmetic error conditions; instead, the more specialized SS$_FLTINV and SS$_FLTDIV error codes are signaled on OpenVMS I64.

Update condition handlers to detect these more specialized error codes. In order to keep code common for both architectures, wherever the code refers to SS$_HPARITH, extend it for OpenVMS I64 to also consider SS$_FLTINV and SS$_FLTDIV.



Up yours!

Quotas and process settings

- OpenVMS I64 images are much larger, sometimes 3x-4x!
- Ensure your PGFLQUOTA and BYTLM are (at least)
  4x-10x your Alpha settings.
  - $ set default sys$system
  - $ run authorize
  - UAF> mod your_account/pgflquota=nnnnnn
  - UAF> mod your_account/bytlm=nnnnnn

THREADS

- THREADCP tool was not ported to OpenVMS I64
  - Relink to change threads related characteristics of an image
  - Use the new SET IMAGE command

- If your application increases the stack size for a thread from the default size, you should increase it more

  HP recommends starting with an increase of three 8-Kb pages (24576 bytes).

New Calling Standard

- Publicly available today at http://www.hp.com/products1/evolution/alpha_retaintrust/openvms/resources.html
- Intel® calling standard with OpenVMS modifications
  - No frame pointer (FP)
  - Multiple stacks
  - only 4 preserved registers across calls
  - Register numbers you're familiar with will change
- All OpenVMS provided tools "know" about these changes
- Most user applications are not affected
- Your code that "knows" about the Alpha standard will almost certainly need to change

Object file format

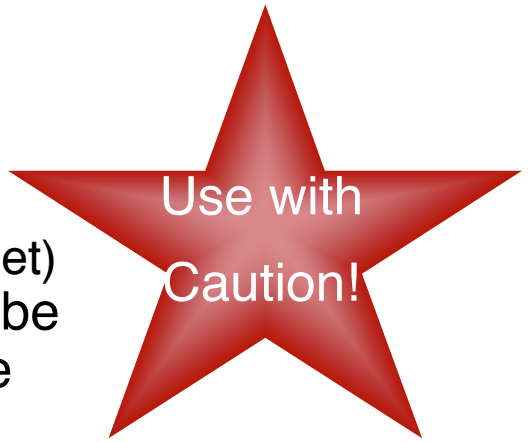
- ELF/DWARF industry standards plus our extensions
  - ELF - Executable and Linkable Format, Itanium® architecture object code, images, etc.
  - DWARF - Debugging and traceback information (embedded in ELF).
- All OpenVMS provided tools “know” about these changes
- Most user applications are not affected
- User written code that “knows” the object file format may have to change
- Specifications are available on the WEB

Reading EXE and OBJ files

- Use ANALYZE/IMAGE vs. parsing the EXE file.
- We might provide a callable interface into SHOW/SET image.

ANALYZE/IMAGE	DCL Symbol that is set	Sample Value
/SELECT=ARCHITECTURE	ANALYZE$ARCHITECTURE	OpenVMS IA64
/SELECT=NAME	ANALYZE$NAME	"DECC$COMPILER"
/SELECT=IDENTIFICATION=IMAGE	ANALYZE$IDENTIFICATION	"C T7.1-003"
/SELECT=IDENTIFICATION=LINKER	ANALYZE$LINKER_IDENTIFICATION	"Linker I02-08"
/SELECT=LINK_TIME	ANALYZE$LINK_TIME	"6/29/2004 4:26:35 PM"
/SELECT=FILE_TYPE	ANALYZE$FILE_TYPE	Image
/SELECT=IMAGE_TYPE	ANALYZE$IMAGE_TYPE	Executable

Debugging using XDELTA



- The system must be booted with XDELTA.
- From the EFI shell
  - SHELL> SET VMS_FLAGS “0,3” (bit 1 should be set)
- The SYSGEN parameter BREAKPOINTS must be set to allow breaking in user,super or exec mode

- Add breakpoints to your code
  - Macro

    ia64_break #break$c_dbg_inibrk
  - C

    __break(BREAK$C_DBG_INIBRK);

```
I64> r ast
Brk 0 at 00010030 on CPU 0
00010030!            break.m     080003   (New IPL = 0)   (New mode = USER)
00010031!            add         r12 = 3FFC, r12 ;P
```

Have fun……you might want to boot with VAXCLUSTER set to 0 to prevent a clustered system from crashing with CLUEXIT

Alignment faults

- Once the port of the application has been completed, you might want to look at alignment faults
  - Alignment faults are expensive on Alpha but are 100 times more expensive on IA64
  - The DEBUG SET MODULE/ALL command used to take 90 seconds. After fixing some alignment faults, it now takes 2 seconds.
  - DCL procedures take approx. 10% less time to execute after fixing alignment faults in DCL.
  - You may detect alignment faults using FLT extension in SDA or using SET BREAK/ALIGN option in the debugger
  - Some alignment faults are easy to fix, some are very hard and some are close to impossible.

Infrastructure changes in OpenVMS V8.2

- We made changes to some system level data structures in OpenVMS V8.2 (Alpha and I64)
- Benefits
  - Laying the foundation for scalability and performance improvements in future releases of OpenVMS
- Impact to applications
  - **Non-privileged applications are not affected**
  - Applications that access the modified data structures in non-standard ways may need to be modified
    - Examples: hard-coded data structure sizes and assumptions about the relative locations of fields within a data structure

Infrastructure changes in OpenVMS V8.2

- Impact to applications (continued)
  - Some privileged applications (such as device drivers) will need to be recompiled and relinked
    - Privileged applications in this case are images linked against the system using the /SYSEXE qualifier and reference the changed data structures or related structures and routines
    - Attempting to execute or load such an image that has not been rebuilt will result in an error during image activation of SYSVERDIF – “System Version Mismatch”.

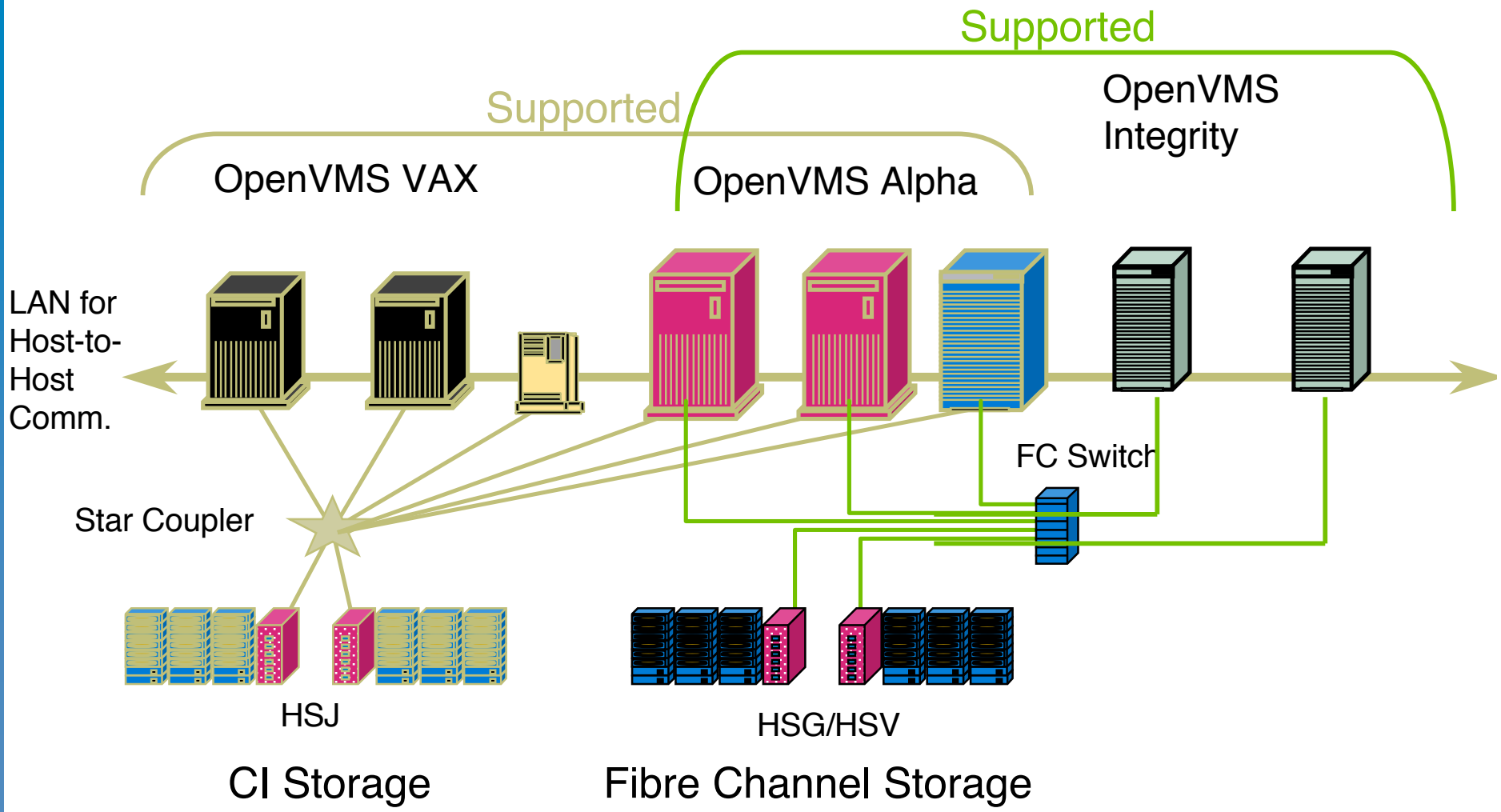
Binary Translator

- Can translate Alpha OpenVMS binary images and libraries linked under all OpenVMS versions from 6.2 to current version
- Can translate a VESTed image that was translated by DECmigrate from a VAX binary image
- Will not translate applications written BASIC, Pascal, PL/1, or Ada
- Restrictions:
  - Alpha binary code
  - Only user-mode apps
  - No privileged instruction
  - No self-modifying code
  - No sys. Memory space reference
  - No user-written system services
- Can mix native and translated images

Upgrading OpenVMS Alpha & Integrity Environments

- System Software Upgrade Paths to V8.2
- Alpha Direct Upgrade Paths:
  - V7.3-2 to V8.2
  - V7.3-1 to V8.2
- Integrity Direct Upgrade Paths:
  - Fresh install required
- Cluster Upgrade Paths – Alpha & Integrity
  - Cluster rolling upgrades are supported from V7.3-2
  - Warranted pairs are V8.2/V8.2 and V8.2/V7.3-2

Continuing evolution of OpenVMS Clusters



NOTE: Support for VAX and Integrity mixed environment is not supported in a production environment.

What we've learned

- Building applications on the latest version of OpenVMS Alpha, with the latest versions of the compilers is CRUCIAL
  - "Port to Alpha first"
  - Most of the problems we encountered were due to updates to the OS and compilers, not due to Itanium
  - Tighter compiler standards
  - Changes in the CRTL
  - Changes to DECthreads - PTHREAD_USE_D4
- When customers say, "I'm building on the latest version on Alpha", they really mean, "The last time we built this was in 1997"
  - Is this you????

What we learned

- FORTRAN 77 is not available on OpenVMS I64
  - Customers need to migrate from F77 to F90, preferably on Alpha prior to starting the port
- MACRO-32 (VAX MACRO)
  - Read the Macro-32 Porting Guide first
  - Pay attention to CALL LINKAGES
- Alpha Assembly (MACRO-64) code MUST be rewritten
- Strongly suggest that customers use supported, documented interfaces. Tricks will work but it will be difficult to port them

What we learned

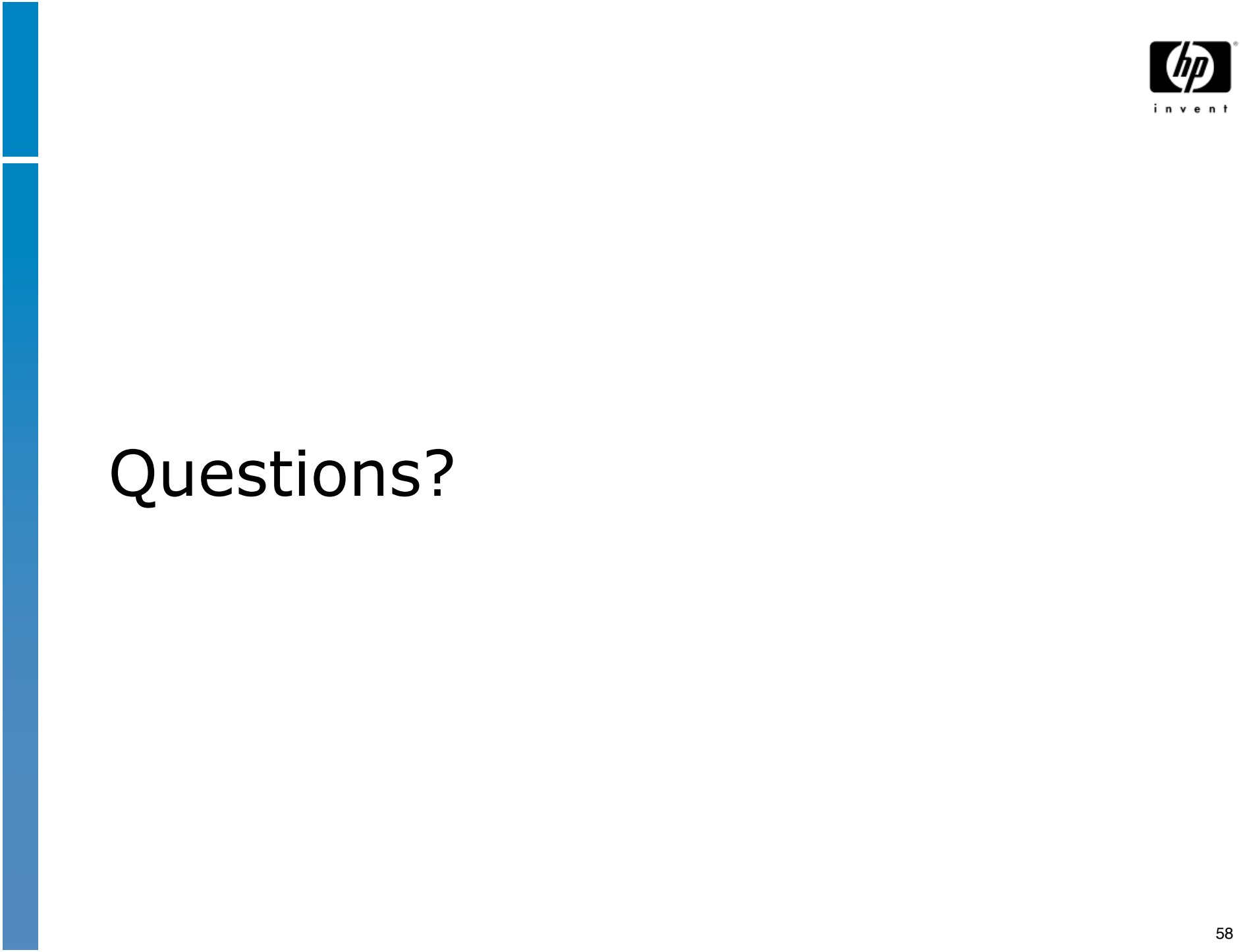
- Generate .MAP and .LIS files on Alpha prior to starting the port
  - Useful to find APIs, match PSECT attributes, identify modules needed to build the application
- Don't assume if an application runs on Alpha that it is correct
  - Some partners uncovered day-one bugs
- Increase quotas in SYSUAF on IPF to do compiles and links
  - BYTLM, FILLM, WSDEF, WSQUO, WSEXTENT, PGFLQUOTA
  - Multiply by 5 then adjust

What we learned

- The rx2600 takes PCI-X cards, not PCI cards
- Porting is really easy; many partners were successful
- Never underestimate the power of porting tools
- Old development environments
  - Haven't kept up with changes to compilers, RTLs, pthreads, etc…
- HP/Intel Developer Forum
  - 5 events in 2004, 75 participants, 51 solutions ported

For further Information about OpenVMS on Integrity Servers

- General OpenVMS on Integrity Servers
  http://h71000.www7.hp.com/openvms/integrity/index.html
- Layered product rollout schedules
  http://h71000.www7.hp.com/openvms/os/swroll/index.html
- Layered products plans (products that either will not be ported or are under review)
  http://h71000.www7.hp.com/openvms/integrity/openvms_plans.html
- OpenVMS Partner plans
  http://h71000.www7.hp.com/openvms/integrity/partners.html
- OpenVMS on Integrity Servers Total Cost of Ownership white paper
  http://h71000.www7.hp.com/openvms/whitepapers/alinean_tco.pdf


Questions?


invent