Porting OpenVMS Applications to the Itanium® Processor Family – Lessons from Real Life

Mandar Chitale
Office of OpenVMS Customer Programs

Europe 2009 Technical Update Days

Agenda

- General considerations
- Lessons
- Case Studies
- Summary
- Q&A

Agenda

- General considerations
- Lessons
- Case Studies
- Summary
- Q&A

Migration Benefits

- 30% drop in power and cooling costs
- 40% lower license cost on OpenVMS
- Reduction in server racks: from 3 to 1
- 50% higher performance with zero downtime

Some of Our Goals in Porting OpenVMS

- To Provide
  - An operating system environment
  - Development tools
  - Documentation
- Make porting as easy as possible
- Use our experiences in porting the OS
- Note: 99% of this talk is about the small percentage of exceptions

Alpha => I64 changes you might care about

- Standards and Formats
  - Object Language/Image Format (ELF/DWARF)
- Hardware/Architecture differences
  - Atomic Instructions (Load/store conditional)
  - No ASMs to specify Alpha instructions
- Both
  - Register conventions
  - Calling Standard
- Mostly care only for architecture-specific code
- In many cases we have given architecture-neutral alternatives

General Development Notes

- Use the latest versions of the compilers before porting to OpenVMS IA64
- Object file and image file sizes are larger on OpenVMS IA64 than on OpenVMS Alpha
- Pay attention to floating point format
  - Integrity supports IEEE only in hardware
  - Alpha supports IEEE and VAX Float in hardware
  - http://h71028.www7.hp.com/ERC/downloads/i64-floating-pt-wp.pdf
- Alignment faults are more costly on IA64 than on Alpha
- Runtime behavior may be different on IA64 if you're relying on "undefined" results
  - For example: COBOL divide by zero
- Refer respective product's Release Notes
  - list of fixes, problems and restrictions

Agenda

- General considerations
- Lessons
- Case Studies
- Summary
- Q&A

Lesson 0 – It is easy

YARTOO SOFTWARE PTY LTD	Compiled approximately 500K lines of code, and had the core applications running in around 6 hours
Flinders Medical Center	Porting our 1.5 million lines of code to OpenVMS on Integrity required no code changes at all.
ASX	500,000 lines of Pascal code, only changing 5 lines of code

Lesson 1 – Latest Versions

- Build your application on Alpha with the latest compilers first
  - Fortran 77 => Fortran 90
  - Ada 83 => Ada 95
    - Binary translator will not translate Ada (83 or 95)
  - Recode PL/I
    - (Note: It can be binary-translated)
  - Integrity C++ Compiler is different than C++ on Alpha.
    - Watch for mixed pointer sizes
  - Binary Translator
  - 3rd Party Products

OpenVMS on Integrity Servers Compilers

- C
  - Itanium® architecture implementation of the OpenVMS Alpha C compiler
- C++
  - Based on the same User interface as HP C++
  - This is not a port of C++ on Alpha but it will be able to compile most of the same source code as HP C++
  - Beware mixed 64- and 32-bit addresses!
- COBOL, BASIC, PASCAL, BLISS
  - Itanium architecture implementations of the OpenVMS Alpha compilers

OpenVMS on Integrity Servers Compilers

- FORTRAN
  - Itanium® architecture implementation of the OpenVMS Alpha Fortran 90 compiler
- Java
  - Itanium architecture implementation of J2SE
- IMACRO
  - Compiles ported VAX Macro-32 code for Itanium architecture
  - Itanium architecture equivalent of AMACRO
- ADA
  - GNAT Pro 6.2-2 for OpenVMS on HP Integrity Servers from AdaCore (Ada-95)
  - The HP Ada-83 compiler is not available on OpenVMS I64

Compiler Migration at a glance

Alpha		Porting	Integrity
Compiler	**Version**	Action	Version
C	**V6.5**	Ported	V7.3
C++	**V6.5**	New from Intel	V7.3
Fortran 77		Not Ported	
Fortran 90	**V7.5**	Ported	V8.2
Cobol	**V2.8**	Ported	V2.9
Basic	**V1.5**	Ported	V1.7
Pascal	**V5.9**	Ported	V6.1
Java	**V1.4.2**	Implemented	V1.5
ADA 83		Not Ported	
ADA 95		New from ACT	V6.2-2
AMacro		IMacro created	
BLISS	**V1.01**	Ported	V1.12
Macro64		Not Ported	
IAS	**N/A**	Available	v7.0U (7.00.4160)
Dibol		Ported by Synergex	
Acucorp Cobol		Ported by Acucorp	
PL/I		Not Ported	

Lesson 2 – Use Standard way

- Watch out for architecture-specific code
  - Do you really need it?
  - Processor/Compiler tech reduce need for assembly?
  - C builtins replace ASMs, work on both architectures (e.g. __CMP_SWAP_LONG not ASM("LDL_L") etc.
  - Does your code "trick the compiler"
    - E.g. Specify R26 in linkage to get return address in Bliss (use builtin)
    - E.g. Use AP as a general register in Macro32 (use R12)
  - Is there a more standard way? (Read the documentation for builtins)

Example or a "more standard way"

- Some applications open and access information in the image (EXE) or OBJ file. Since the file layout has changed on I64, code that works on Alpha will not work on I64.
- Use `ANALYZE/IMAGE` vs. parsing the EXE file.
- BTW - Symbol table files (.STB) can not be placed in object libraries any more

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

Lesson 3 – Plan the config

- Plan your final Cluster and Hardware configuration
  - VAX and Integrity only supported together for migration
  - Pay attention to MSCP-served disks, for example
  - Any specific Hardware being used
- Read the documentation
  - Porting Guides
  - Compiler and OS Release notes
  - Layered Products
  - Calling Standard

Lesson 4 – Stay current

- Stay Current
  Make time to update to most recently released
  - Operating system
  - Compilers
  - Layered products
- Take the time to read the documentation
  - Release notes (base operating system and compilers)
  - Porting Guide
  - Calling Standard
  - For drivers, user-defined system services and other privileged code, read *"HP OpenVMS Guide to Upgrading Privileged-Code Applications"*

Lesson 5 - Know Your Code

- There are not many coding changes required
  - Nearly all are uncommon
  - But you can waste a lot of time if you do not know your code well enough to determine if it has some of these problems

New Calling standard

- New Calling Standard
  - Available at http://h71000.www7.hp.com/openvms/whitepapers/index.html
  - Also in the doc set
  - Intel® calling standard with OpenVMS modifications
    - Register numbers you're familiar with will change
  - All OpenVMS tools "know" about these changes
  - Most user applications are not affected
  - User written code "knowing" about the Alpha calling standard may have to change

Floating point data

- Floating point data types
  - Itanium® architecture supports IEEE float only
  - 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
  - The HP supplied Runtime Libraries have been modified to add IEEE interfaces where needed
  - White Paper with technical details about the differences between VAX Float and IEEE Float is available at http://h71000.www7.hp.com/openvms/whitepapers/index.html

Architecture specific code

Source Code that May Need to Change

- Architecture Specific code
  - All Alpha assembler code must be rewritten
- 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

Conditionalized code

- Conditionalized code
  - Build command files
    - `$ if .not. Alpha  ! Assumes VAX`
  - Application source code
    - `#ifndef (alpha)  // Assumes VAX`
    - C asm code
  - More often, the Alpha variant works on I64
- Be consistent, use a single method to determine the hardware architecture
- Don't default to an architecture, be specific
    - $ if .not. Alpha ! Assumes VAX
      The above worked fine until 30-Jun-2003 when OpenVMS I64 V8.0 was released.
    - #ifdef __ia64

Lesson 6 – Performance considerations

- Pay attention to unaligned data
- Not only slow, but not scalable
- If you can't fix the mis-alignment, tell the compiler!

Performance Considerations – Alignment Faults

- Alignment faults can be up to100 times more expensive on IA64
- Only affects data accessed through a pointer or a parameter
  - No faults on local, stack based variables
- Detect alignment faults using:
  - FLT extension in SDA
  - SET BREAK/UNALIGN option in the debugger
  - SYS$EXAMPLES:SET_ALIGN_REPORT.C
  - $ MONITOR ALIGN (I64 only)
  - PCA SET UNALIGNED (C, COBOL, FORTRAN, BASIC, PASCAL)
  - System Services
    - $GET_SYS_ALIGN_FAULT_DATA
    - $INIT_SYS_ALIGN_FAULT_REPORT
    - $PERM_DIS_ALIGN_FAULT_REPORT
    - $PERM_REPORT_ALIGN_FAULT
    - $START_ALIGN_FAULT_REPORT
    - $STOP_ALIGN_FAULT_REPORT
    - $STOP_SYS_ALIGN_FAULT_REPORT

Alignment Faults –Compiler support

Compiler support

- Generates fetch/store instructions to avoid Alignment faults
  - Inform compiler on pointer pointing to unaligned data
  - __unaligned (C)
  - /assume=[no]aligned_objects (C)
  - .set_registers unaligned=<Rx> (Macro)
  - align(x) (Bliss32/Bliss64)
  - aligned(x) (Pascal)

Alignment Faults – Compiler support

Compiler support

- C, C++, Pascal and Fortran automatically insert padding to naturally align structures (can optionally be disabled)
  - `/nomember_align (C&C++)`
  - `/align=VAX (Pascal)`
  - `/align=PACKED (Fortran)`
- COBOL does not automatically pad structures
  - This can optionally be enabled but use it carefully because this will change the data layout
- Compilers can insert code to avoid faults for unaligned data.
  - Small performance degradation, but much better than taking an alignment fault
  - Just be sure the compiler knows using switches mentioned before

Alignment Faults – SDA Extension

- FLT extension in SDA

```
$ ANALY/SYS
SDA> FLT   ! LISTS VALID COMMANDS
SDA> FLT LOAD
FLT$DEBUG load status = 00000001
SDA> FLT START TRACE
Tracing started...
SDA> ! wait sufficient time to collect meaningful data
SDA> FLT STOP TRACE
SDA> FLT SHOW TRACE [/SUMMARY]
SDA> FLT UNLOAD
FLT$DEBUG unload status = 00000001
```

How to fix alignment problems?

- Pad structures to make them aligned if possible
- If not possible, much better to have unaligned data that the compiler knows about
- Example fix:
  - #pragma __nomember_alignment
- Externs/Pointers: Why are they misaligned?
- OpenVMS Technical Journal article on alignment: http://h71000.www7.hp.com/openvms/journal/v9/index.html

Lesson 8 – Exceptions are costly

- Consider reducing frequent use of exceptions
- For SETJMP/LONGJMP use __FAST_SETJMP if possible
  - Disadvantage: No SS$_RESIGNAL calls to handlers

Performance Considerations – Exception Handling

- Exceptions incur some overhead Alpha
- Upto 20 times more expensive on Integrity Servers
- Detect exception handling using:
  - EXC extension in SDA
  - Examine your code
    - look for TRY/CATCH blocks
    - exception handlers
    - POSIX signals
- If you use setjmp/longjmp you can significantly improve performance by using the __FAST_SETJMP or __UNIX_SETJMP macros
  - (Note: Handlers not called)

Exception Handling is Slow

- Itanium calling standard expects them to be infrequent
- Trades off slow execution of exception for fast setup
- If you signal on infrequent errors, not a problem
- If you signal as a normal part of execution, maybe a problem.
- OS Example: Search list logicals: Frequent file-not-found as normal part of processing!

Finding Exceptions

- Debug: SET BREAK/EXCEPTION
- SDA (may get you more than you want!)

Exception Handling – SDA Extension

- EXC extension in SDA

```
$ ANALY/SYS
SDA> EXC  ! LISTS VALID COMMANDS
SDA>EXC LOAD
EXC$DEBUG load status = 00000001
SDA>EXC START TRACE
Tracing Started…
SDA> ! wait sufficient time to collect meaningful data
SDA>EXC STOP TRACE
Tracing Stopped…
SDA> SET OUTPUT/NOHEAD trace.lis ! dump output to a file
SDA> EXC SHOW TRACE
SDA> EXC UNLOAD
EXC$DEBUG unload status = 00000001
SDA> EXIT
```

Exception Handling – SDA Extension

- EXC is really for debugging OS exception handling!
- Search the trace file for interesting information
- Determine the number of exceptions during the trace period
  ```
  $ search trace.lis “begin pcb” /noout/stat
  ```
- See where the handler is being called
  ```
  $ search trace.lis “About to call handler”
  ```
- See interesting application PC values
  ```
  $ search trace.lis “pc: 00000000.00”
  ```
- Lots of other useful information in the trace listing

Agenda

- General considerations
- Lessons
- Case Studies
- Summary
- Q&A

Real Life Experiences

- HP/Intel Developer Forum
  - 15 events in last several years, 250+ participants, 170+ solutions ported during 2.5 day workshops
- Large office supply company
  - 11GB save set; Basic; worked with no change
  - Performance seemed poorer until they started using multiple data disks on the test system

Real Life Experiences

- Government Regulatory Office
  - No code changes required for payroll system (Cobol, C, Macro32)
  - Built application the first day; ran tests the second day
- Large Bank
  - 4-5Million lines of VAX Basic, plus Macro32 and DCL
  - Third party products (Oracle CDD, CA Job management, IBM MQseries)
  - Built with no code changes
  - Had some informational Macro32 messages
  - Performance issues - alignment faults were worked on

Agenda

- General considerations
- Lessons
- Case Studies
- Summary
- Q&A

10 Commandments

1. Do a complete inventory of all Layered products (HP and 3rd party software
   - Ensure you know the status of each of these on OpenVMS I64 before you go too far in your port.
2. Make sure your application builds cleanly with Latest compilers and runs on latest version of Alpha
3. Check for hardware architecture in source code and DCL command procedures
4. Automate regression tests as much as possible
   - Clearly documented manual regression tests where necessary
5. Document your build procedure / process
6. Read the Porting Guide and various Release Notes (Really do it!)
7. Reduce / Recode / eliminate any Alpha Macro (Macro64 code) and PL/I . Update any Fortran 77 code to Fortran 90.
8. Where possible, use IEEE floating point
9. Compare results between Alpha and Integrity systems. Look at Alignment faults and exception handling
10. **Sit back and just… Re-compile, Re-Link, and run :-)**

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/index.htm
- Transition modules
  http://h71000.www7.hp.com/openvms/integrity/transition/modules.html

Questions?

Thank You