OpenVMS Performance Update

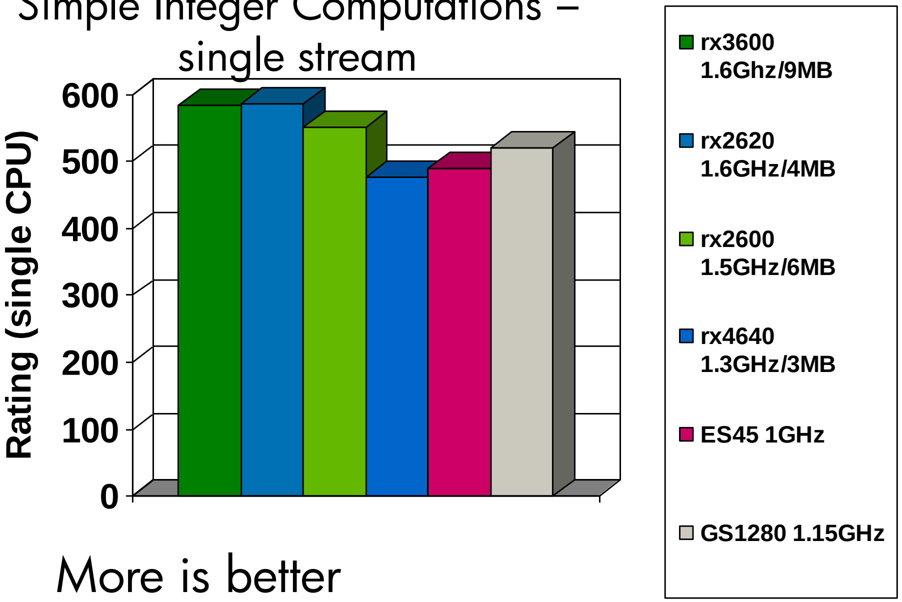
Gregory Jordan
Hewlett-Packard

Agenda

- System Performance Tests
  - Low Level Metrics
  - Application Tests
- Performance Differences Between Alpha and Integrity
- Recent Performance Work in OpenVMS
- Summary

CPU Performance Comparisons

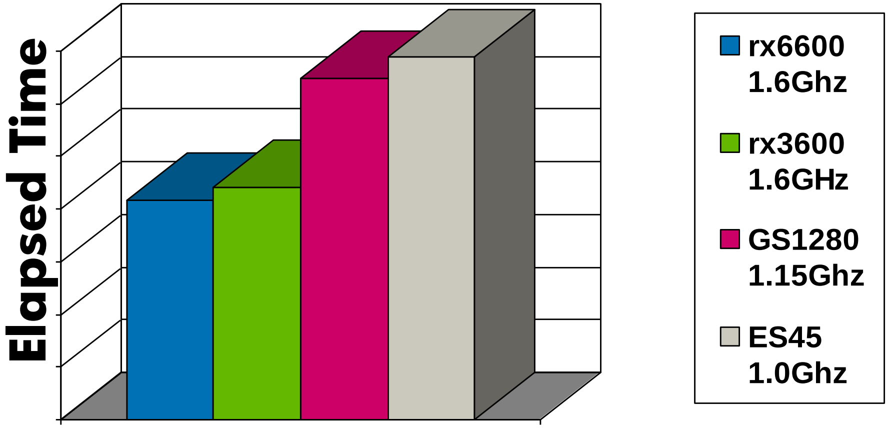
Simple Integer Computations – single stream



More is better

(Small number of computations in test do not take full advantage of EPIC)

Floating Point Computations – single stream



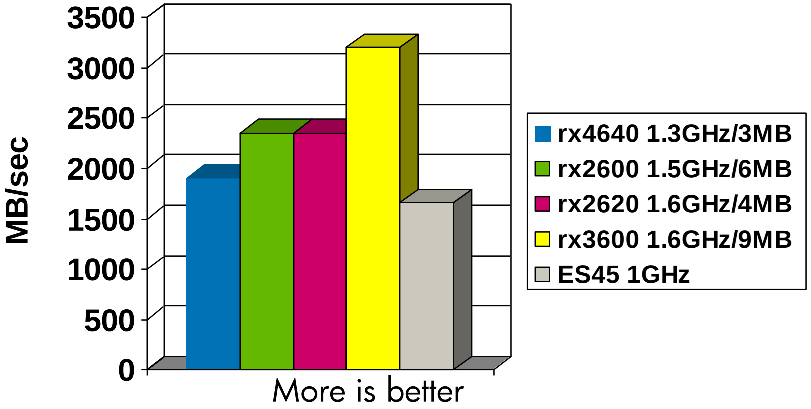
Less is better

The Itanium processors are fast.

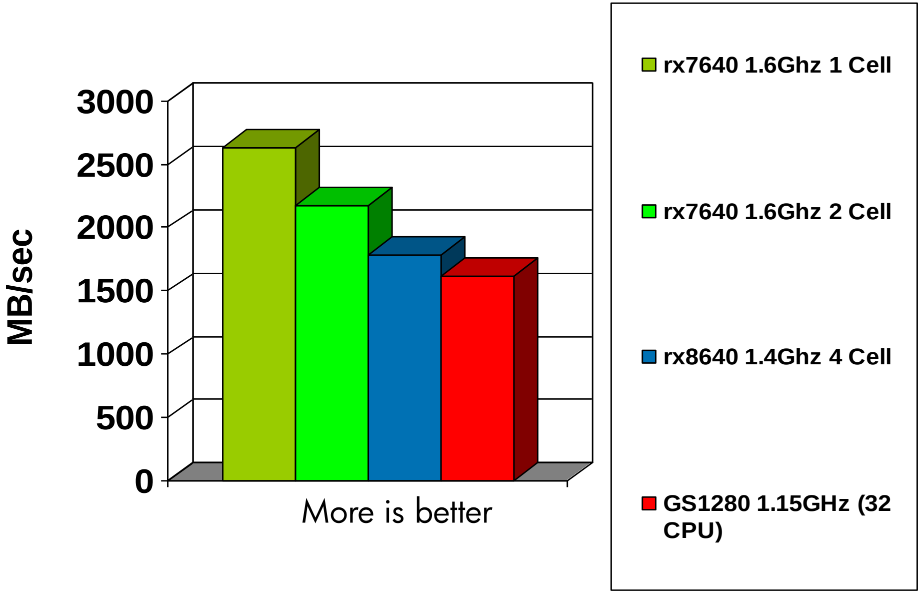
- Faster cores
- 128 general purpose and 128 floating point registers
- Large caches compared to Alpha EV7

Memory Bandwidth

Memory Bandwidth (small servers)
Computed via memory test program – single stream



Memory Bandwidth (large servers)
Computed via memory test program – single stream



The latest Integrity Servers have very good memory bandwidth

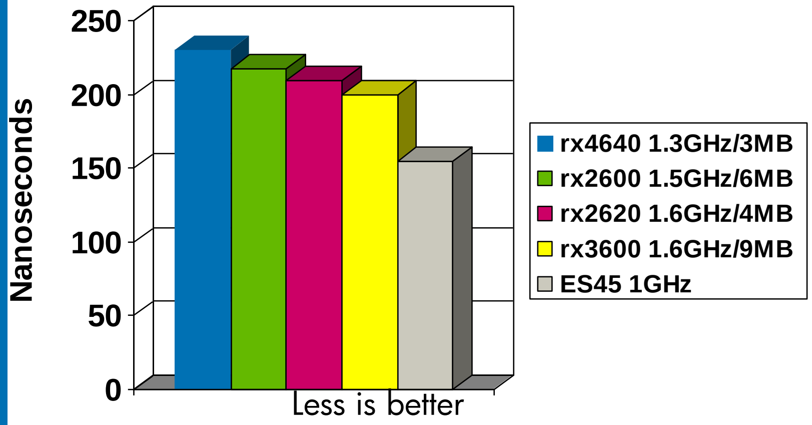
- Applications which move memory around or heavily use caches or RAMdisks should perform well

Interleaved Memory

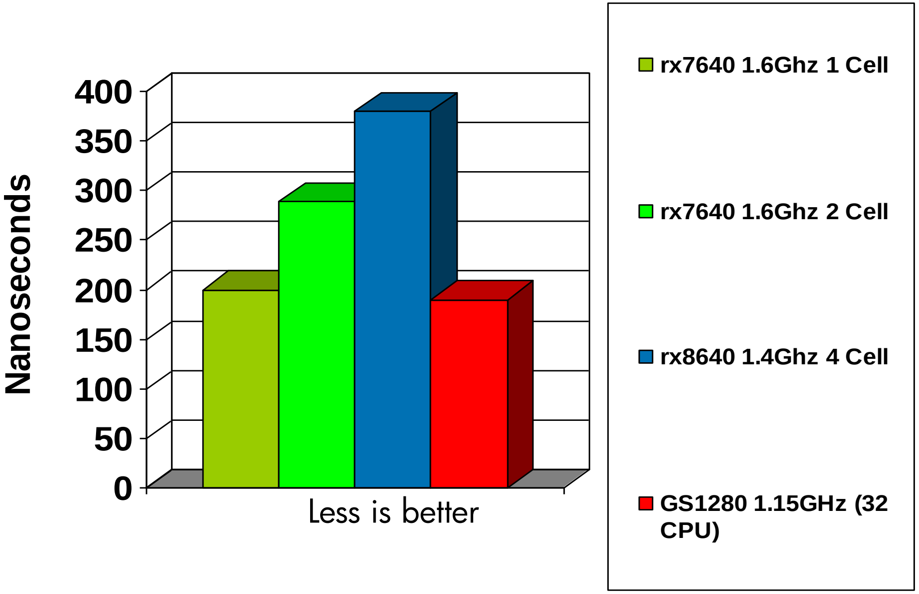
- OpenVMS supports Interleaved Memory on the cell based Integrity Servers
  - Each subsequent cache line comes from the next cell
  - The Interleaved memory results in consistent performance
- For best performance:
  - Systems should have the same amount of physical memory per cell
  - The number of cells should be a power of 2 (2 or 4 cells)

Memory Latency

Memory Latency (small servers)
Computed via memory test program – single stream



Memory Latency (large servers)
Computed via memory test program – single stream



Memory latency is slower on Integrity when compared to Alpha.

Caches

- Integrity cores have large on chip caches
  - 18MB or 24MB per processor
    - The cache is split between the two cores so each core has a dedicated 9MB or 12MB of L3 cache
      - Load time is 14 cycles – about 9 nanoseconds
    - The Alpha EV7 processor has 1.75MB of L3 Cache
  - A reference to physical memory brings in a cache line
    - Cache line size is 128 bytes on Integrity vs. 64 bytes on Alpha
    - The larger cache line size can also result in reduced references to physical memory on Integrity
  - Larger cache can reduce references to physical memory – especially for applications that share large amounts of read only data

IO Performance

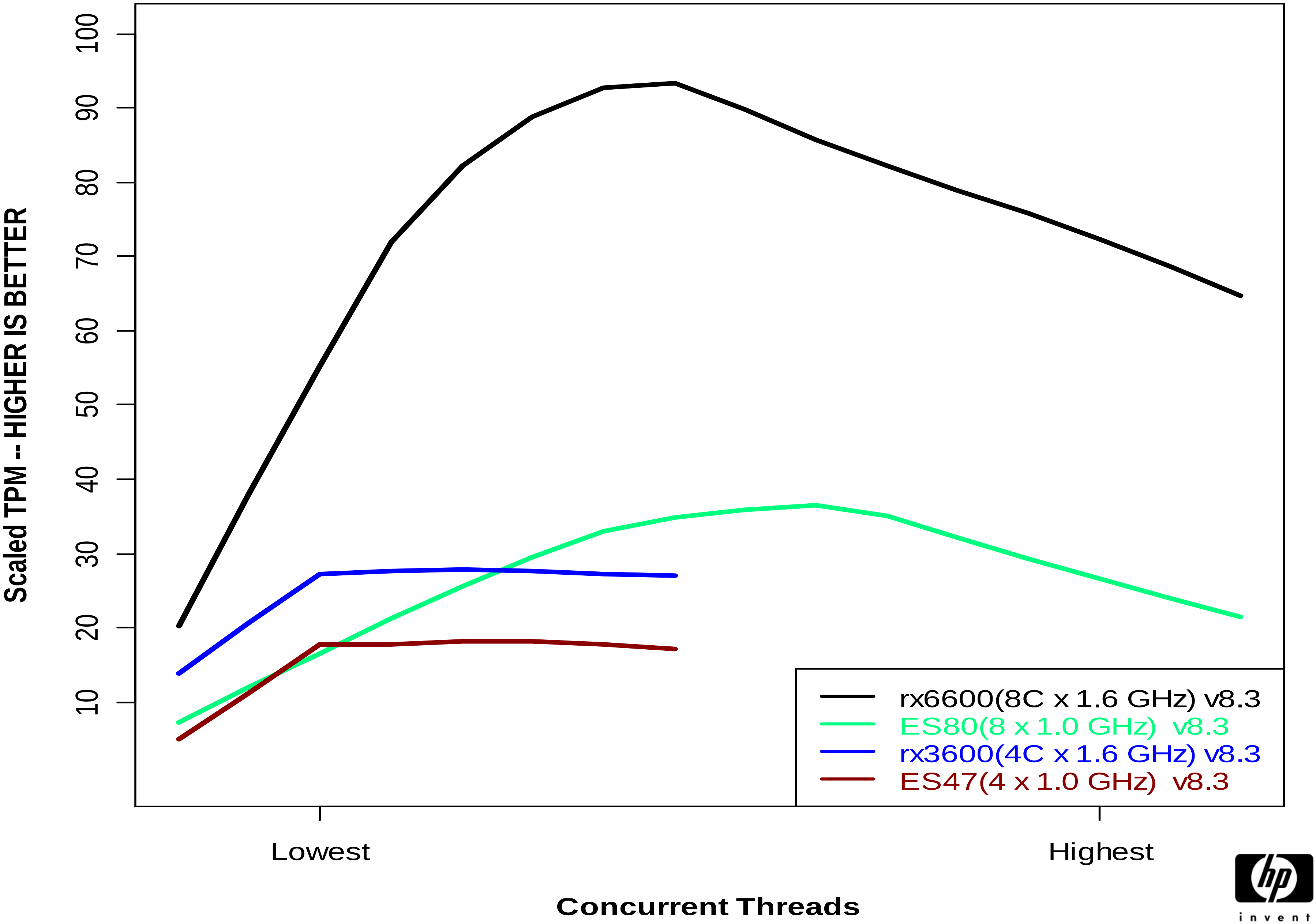
- Both Alpha and Integrity can easily saturate IO adapters
- The amount of CPU time required per IO tends to be smaller on Integrity (fibre channel and lan)
- Integrity can benefit from the better memory bandwidth

Bounded Application Comparisons

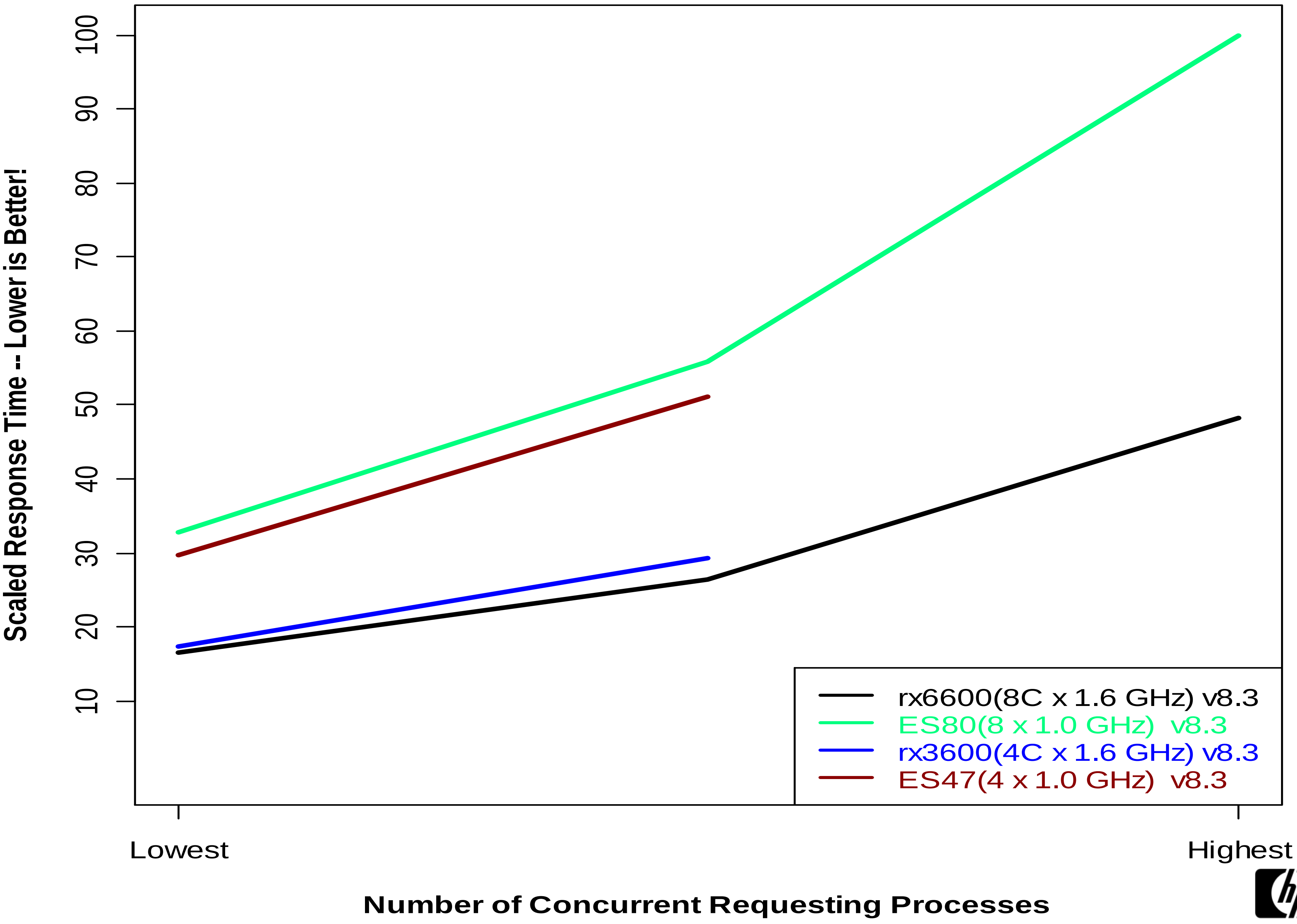
- Java
- Secure WebServer
- MySQL

ES47	rx3600
ES80	rx6600

Intensive Java Workload

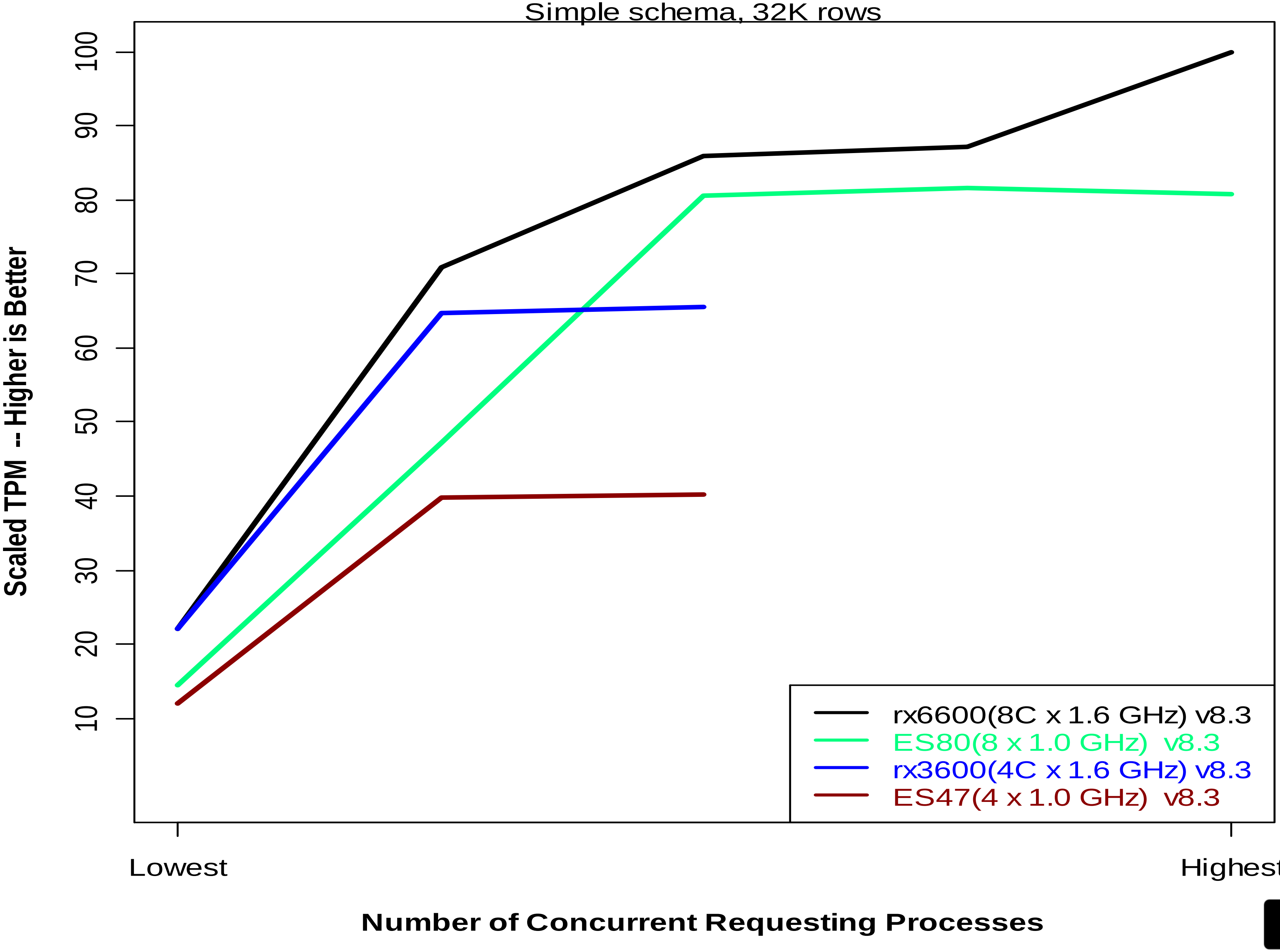


Intensive Local SWS Workload

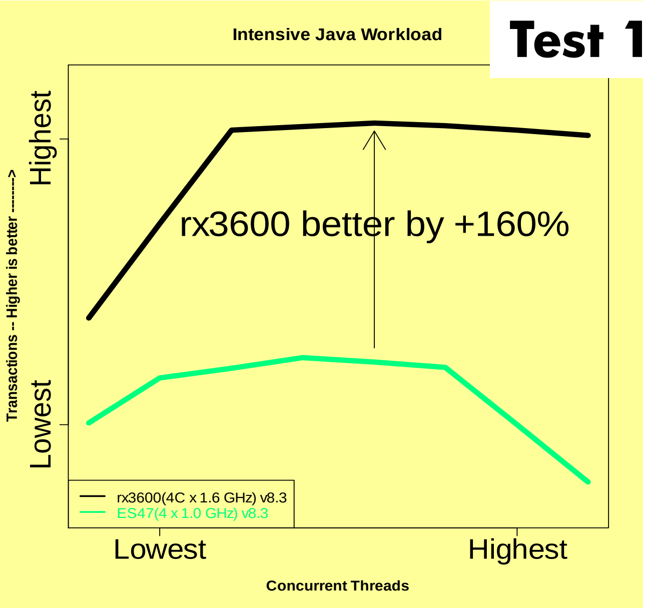


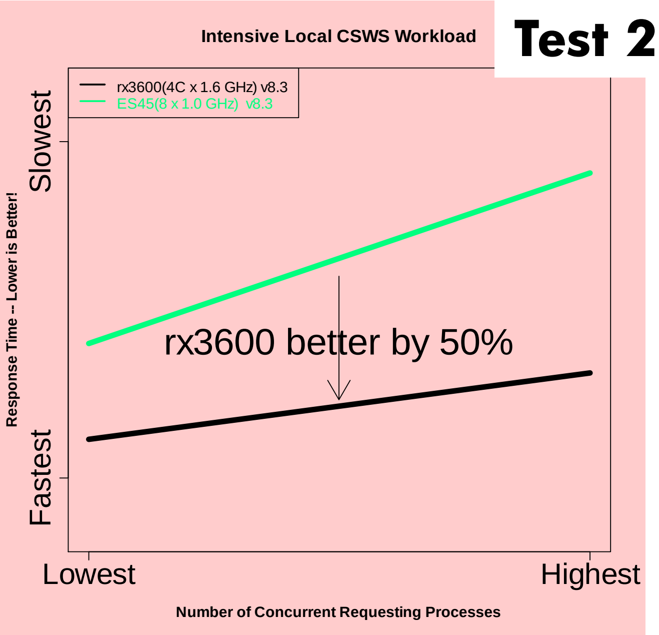
Intensive Local MySQL 4.1.14 Workload

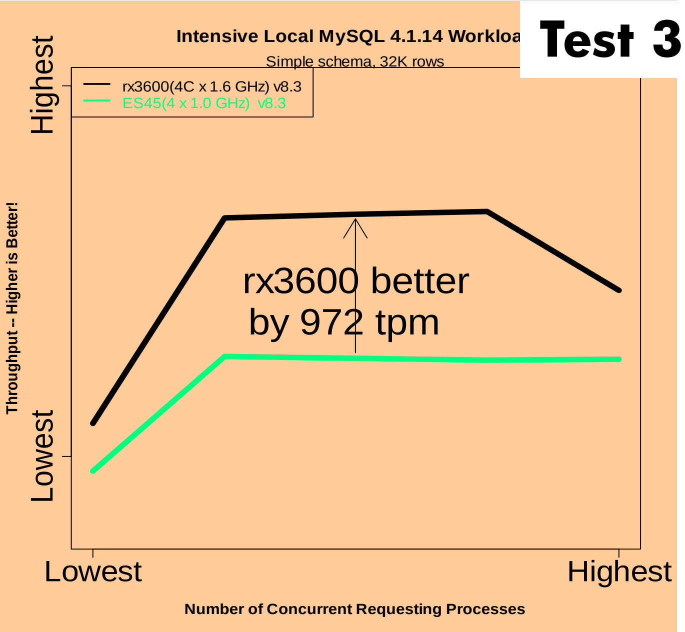
Simple schema, 32K rows



Alpha and Integrity Testing







Comparison was between:

🟩 Alpha

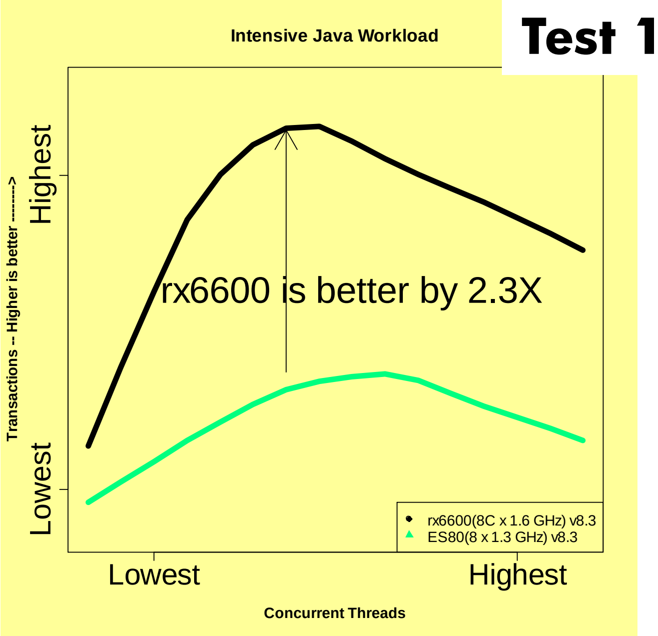
⬛ Integrity

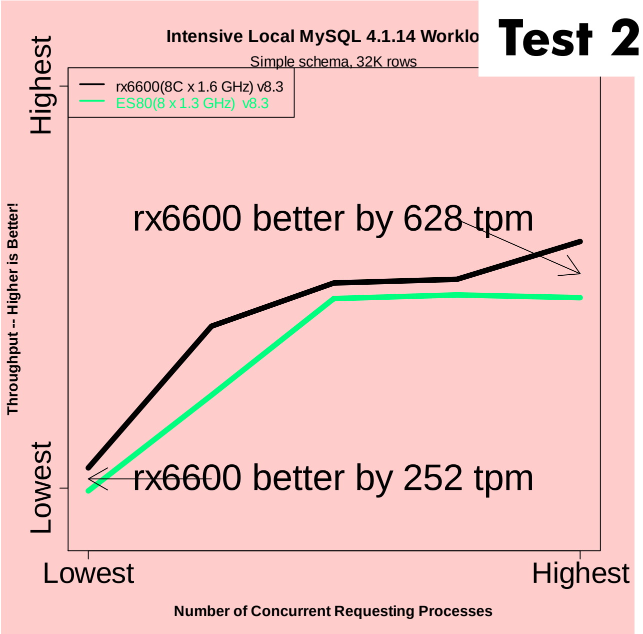
ES47 (4CPU 1.0 Ghz, OpenVMS 8.3) and

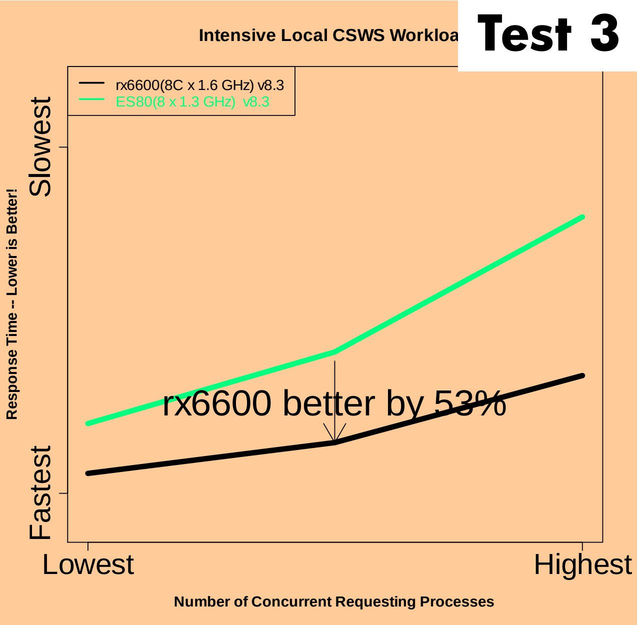
rx3600 (4C 1.6Ghz, OpenVMS 8.3)

1. First test was using an intense Java workload
2. Second test used concurrent processes
3. Third test was simulating a MySQL workload

Alpha and Integrity Testing


Test 1


Test 2


Test 3

Comparison was between: 🟩 Alpha
■ Integrity

ES80 (8CPU 1.3Ghz, OpenVMS 8.3) and

rx6600 (8C 1.6Ghz, OpenVMS 8.3)

1. First test was using an intense Java workload
2. Second test used concurrent processes
3. Third test was simulating a MySQL workload

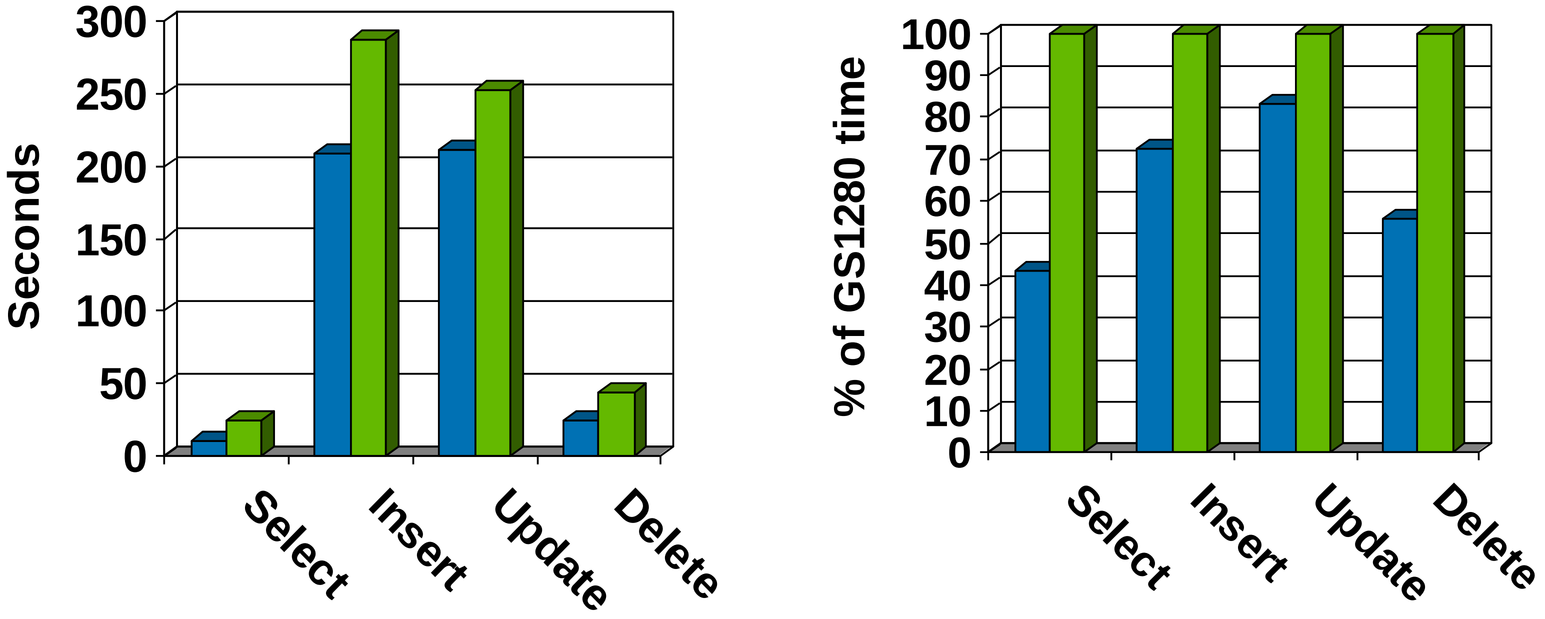
Oracle 10gR2 Comparison

OpenVMS V8.3

🟦 rx8640 (8C 1.6Ghz)

🟩 GS1280 (8CPU 1.15Ghz)

Less is better

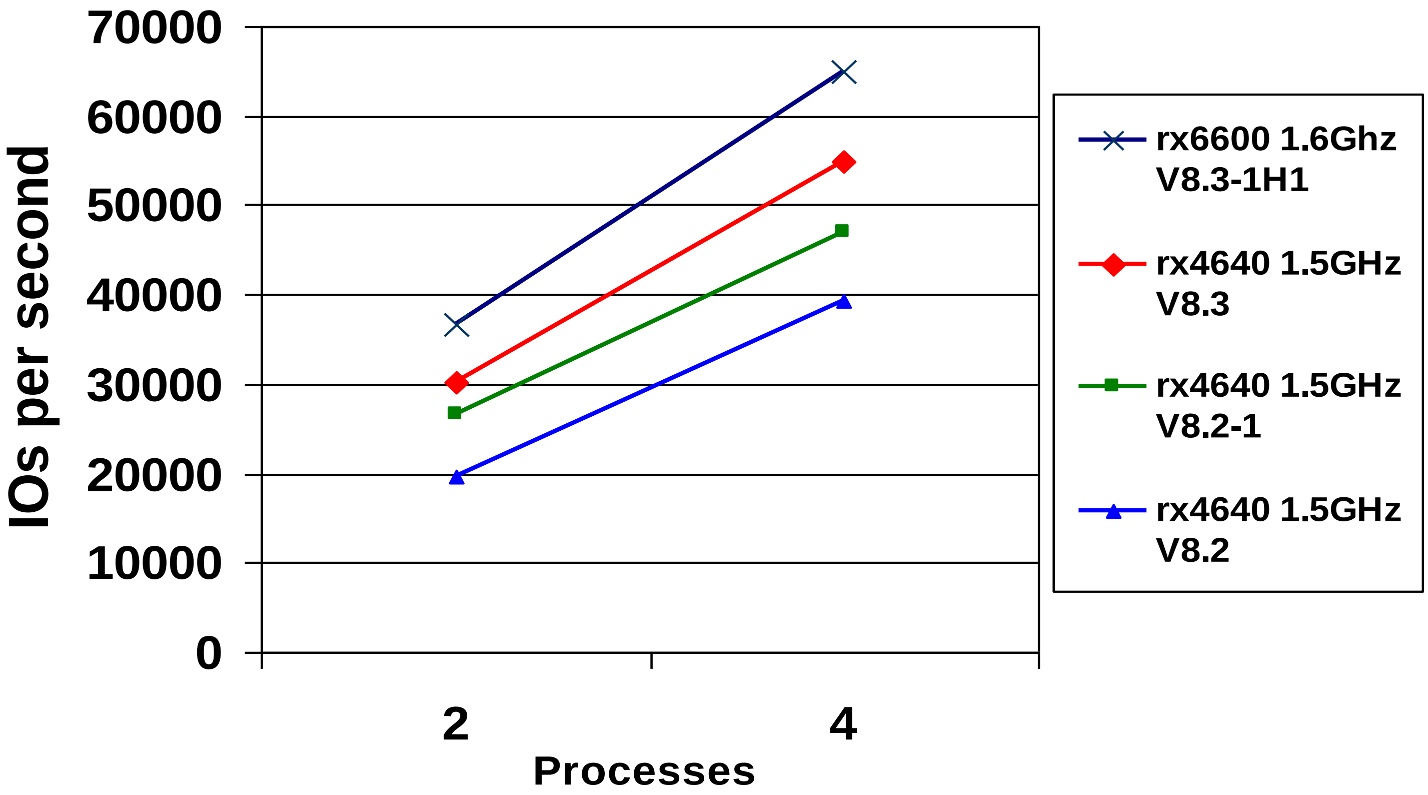


The testing was a sequence of 100,000 iterative SQL statements run multiple times. The first graph is the total elapsed time, the second shows the same data with the GS1280 normalized to 100.

If Performance is Important - Stay Current

- V8.2
  - IPF, Fast UCB create/delete, MONITOR, TCPIP, large lock value blocks
- V8.2-1
  - Scaling, alignment fault reductions, $SETSTK_64, Unwind data binary search
- V8.3
  - AST delivery, Scheduling, $SETSTK/$SETSTK_64, Faster Deadlock Detection, Unit Number Increases, PEDRIVER Data Compression, RMS Global Buffers in P2 Space, alignment fault reductions
- V8.3-1H1
  - Reduce IOLOCK8 usage by Fibre Channel port driver, reduction in memory management contention, faster TB Invalidates on IPF
- Some performance work does get back ported…

RMS1 (Ramdisk) OpenVMS Improvements by version



More is better

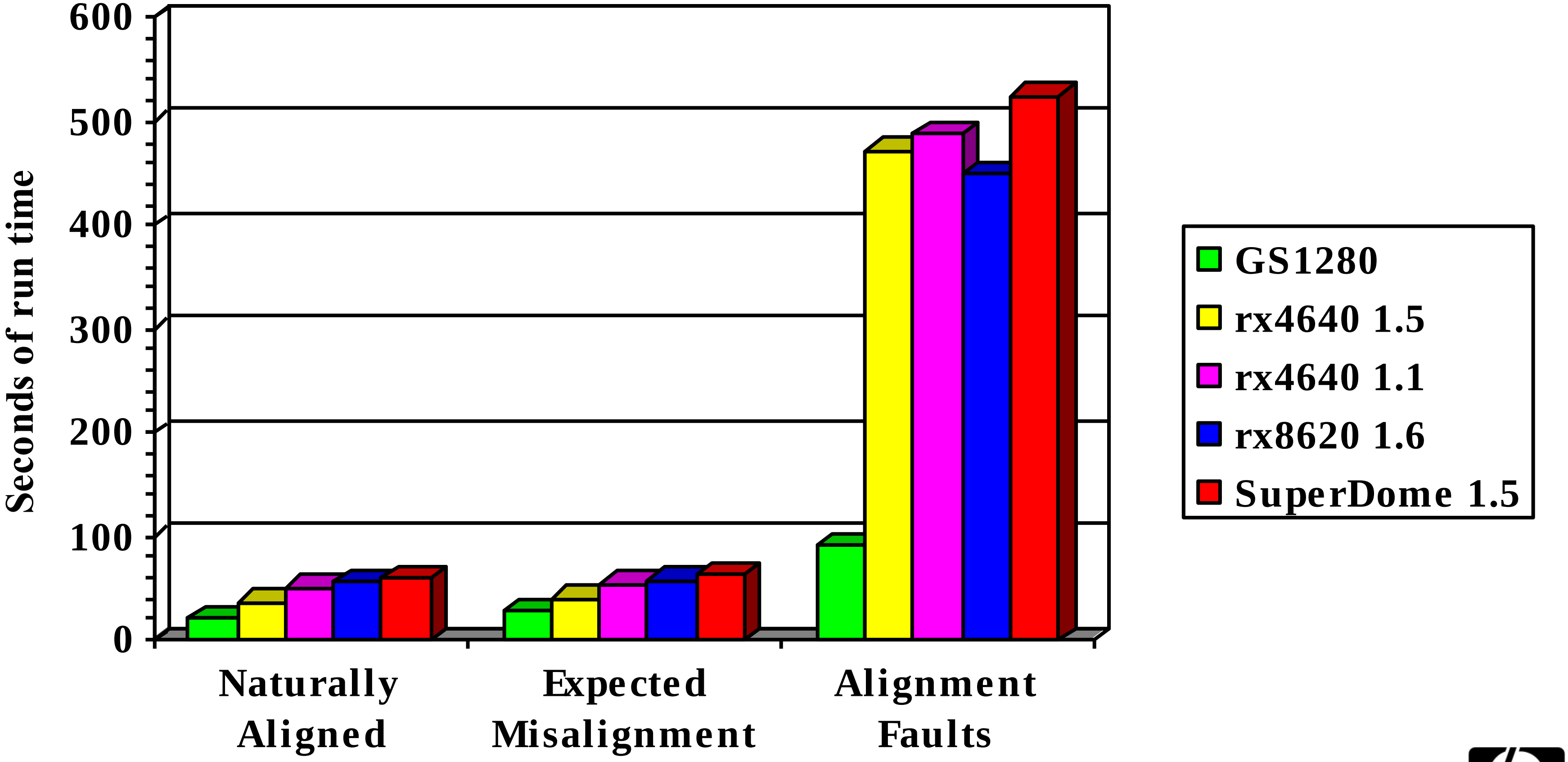
When Integrity is Slower than Alpha…

- After an application is ported to Integrity – if performance is disappointing, there are typically 4 reasons
  - Alignment Faults
  - Exception Handling
  - Usage of _setjmp/_longjmp
  - Locking code into working set

Alignment Faults

- Rates can be seen with MONITOR ALIGN (V8.3) on Integrity systems
  - 100,000 alignment faults per second is a problem
    - Fixing these will result in very noticeable performance gains
  - 10,000 alignment faults per second is potentially a problem
    - On a small system, fixing these would only provide minor performance improvements
    - On a large busy system, this is 10,000 too many

Alignment Faults – Avoid them



Exception Handling

- Usage of lib$signal() and sys$unwind is much more expensive on Integrity
  - Finding condition handlers is harder
  - Unwinding the stack is also a very compute intensive operation
  - Heavy usage of signaling and unwinding will result in performance issues on Integrity
  - In some cases, usage of these mechanisms will occur in various run time libraries
- There is work in progress to improve the performance of the calling standard routines
  - Significant improvements are expected in a future release

Exception Handling continued

- In some cases, application modifications can be made to avoid high frequency usage of lib$signal and sys$unwind
  - Several changes were made within OpenVMS to avoid calling lib$signal
  - Sometimes, a status can be returned to the caller as opposed to signaling
  - In other cases, major application changes would be necessary to avoid signaling an error
    - If there are show stopper issues for your application – we want to know

setjmp/longjmp

- usage of _setjmp and _longjmp is really just another example of using SYS$UNWIND
  - The system uses the calling standard routines to unwind the stack
- A Fast version _setjmp/_longjmp is available in C code compiled with /DEFINE=__FAST_SETJMP
  - There is a behavioral change with fast setjmp/longjmp
    - Established condition handlers will not be called
    - In most cases this is not an issue, but due to this behavioral change, the fast version can not be made the default

Recent/Current Performance Work

- Image Activation Testing
  - TB Invalidation Improvements
  - Change in XFC TB Invalidates
- Dedicated Lock Manager Improvements
- IOPERFORM improvements

Image Activation Testing

- A customer put together an image activation test
  - The test activates 150 sharable images numerous times using lib$find_image_symbol
  - The customer indicated the rx8640 was slower than the GS1280
  - The test was provided to us so we could look in detail at what was occurring
    - We reproduced similar results – the test took 6 seconds on the rx8640 vs. 5 seconds on GS1280
  - Analysis has resulted in a number of performance enhancements and some tuning suggestions
    - Some enhancements are very specific to Integrity systems, others apply to both Integrity and Alpha

Image Activation Tuning Suggestions

- One observation for the test was that there was heavy page faulting
  - there was a significant amount of demand zero page faults
  - there was also a significant amount of free list and modified list page faults
    - Due to in large number of processor registers on IPF, dispatching exceptions (such as a page fault) is slower on IPF
  - To avoid the page faults from the free and modified lists, two changes were made:
    - The processes working set default was raised
    - The system parameter WSINC was raised
  - The above changes avoided almost all of the free list and modified list faults
  - A potential performance project also being investigated is to process multiple demand zero pages during a demand zero page fault

Low Level Image Activation Analysis

- Spinlock usage was compared between Alpha and Integrity
  - Several areas stood out in this comparison
    - INVALIDATE spinlock hold time
      - GS1280 - 6 micro seconds, rx6600 - 48 microseconds
    - XFC spinlock hold time when unmapping pages
      - GS1280 - 40 microseconds, rx6600 - 110 microseconds
    - MMG hold time for paging IO operations
      - GS1280 - 6 microseconds, rx6600 - 24 microseconds
  - All of the above were fairly frequent operations 1,000s to 25,000 times per second during the image activation test

TB Invalidates

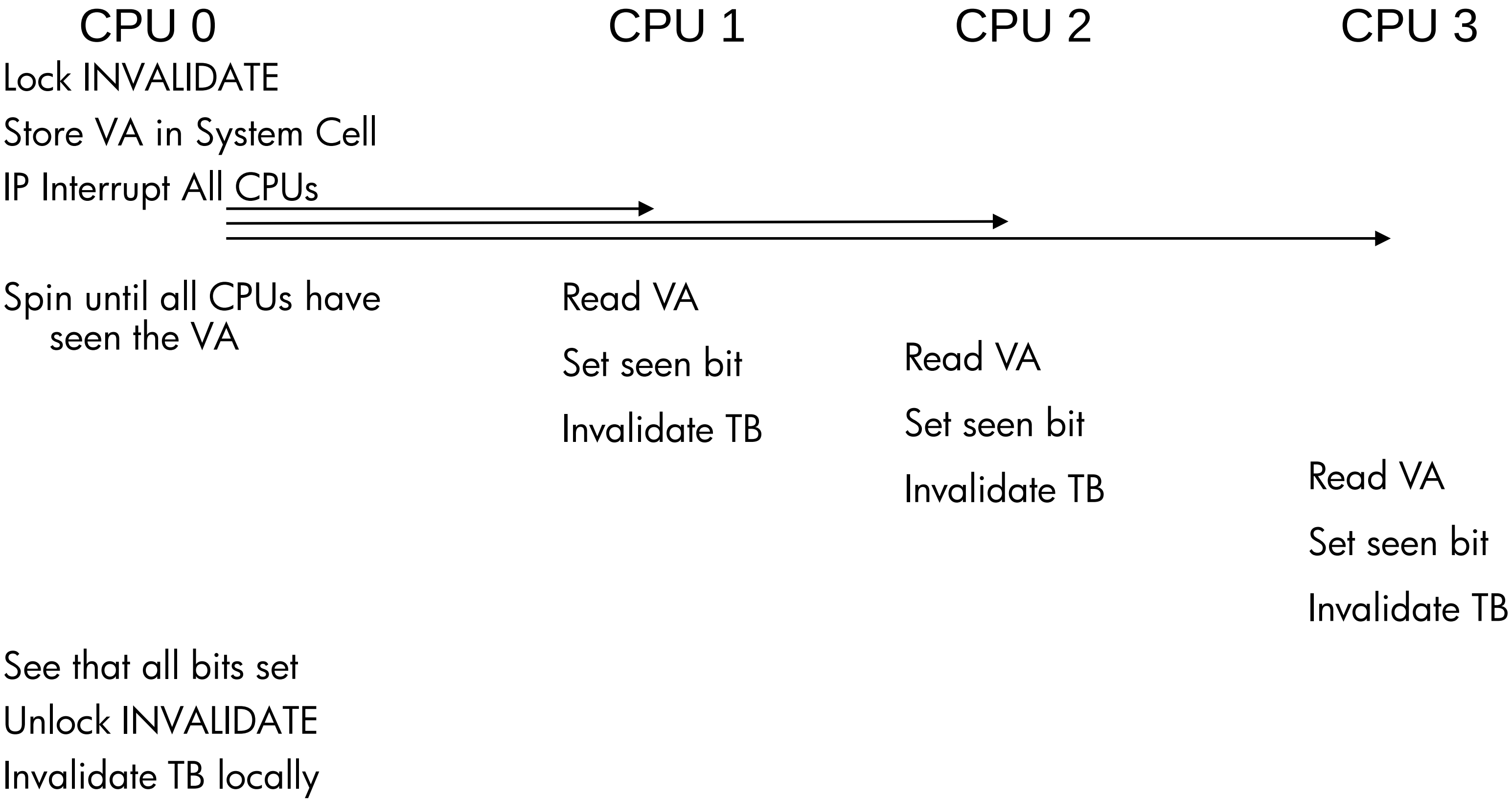
- CPUs maintain an on chip cache of Page Table Entries (PTEs) in Translation Buffer (TB) entries
  - The TB entries avoid the need for a CPU to access the page tables to find the PTE which contains the page state, protection, and PFN
  - There are a limited number of TB entries per core

- When changing a PTE, it is necessary to invalidate TB entries on the processor
  - Not doing so can result in a reference to a virtual address using a stale PTE
  - Depending on the VA mapped by the PTE, it may be necessary to invalidate TB entries on all cores on the system or on a subset of cores on the system

TB Invalidate Across all CPUs

- Both Alpha and Integrity have instructions to invalidate TB entries on the current CPU for a specific virtual address
- The current mechanism to invalidate a TB entry on all CPUs is to provide the virtual address to the other CPUs and get them to execute the TB invalidate instruction
- The CPU initiating the above operation holds the INVALIDATE spinlock, sends an interrupt to all CPUs and waits until all other CPUs have indicated they have the VA to invalidate
  - The Integrity cores were slower to respond to the inter-processor interrupts (especially if the CPUs were idle and in a state to reduce power usage)

Invalidating a TB Entry



CPU 0

Lock INVALIDATE

Store VA in System Cell

IP Interrupt All CPUs

Spin until all CPUs have
seen the VA

See that all bits set

Unlock INVALIDATE

Invalidate TB locally

CPU 1

Read VA

Set seen bit

Invalidate TB

CPU 2

Read VA

Set seen bit

Invalidate TB

CPU 3

Read VA

Set seen bit

Invalidate TB

Integrity Global TB Invalidate

- Integrity has an instruction that will invalidate a TB entry across all cores (ptc.g)
- Usage of the above does not require sending an interrupt to all cores
  - Communication of the invalidate occurs at a much lower level within the system
  - Cores in a low power state do not need to exit this state
- The OpenVMS TB invalidate routines were updated to use ptc.g for Integrity
  - What was taking 24-48 micro seconds on an rx6600 could now be accomplished in under 1 microsecond.
  - Data from larger systems such as a 32 core rx8640 brought the TB invalidate time down from 100 microseconds to 5 micro seconds

Why didn’t we use the ptc.g instruction in the first place?

XFC Unmapping Pages

- Analysis in to why the XFC spinlock was held so long showed that within it's routine to unmap pages – XFC may need to issue TB invalidates for some number of pages
  - With the old TB invalidate mechanism, these operations were costly on Integrity and thus the very long hold times
- Looking at this routine though, it was determined that it wasn't necessary to hold the XFC spinlock while doing the TB invalidate operations
  - This reduced the average hold time of the XFC spinlock and results in improved scaling
  - The average hold time of the XFC spinlock when mapping and unmapping pages was reduced by 35%

Image Activation Test Results

- With all of these changes – the image activation test that was taking over 6 seconds on an rx6600 now runs in about 3.4 seconds
- Only the working set tuning and XFC changes would impact Alpha performance
  - The working set tuning had a negligible impact
  - The XFC test has not yet been tested, but would also have no impact on a single stream test

More on Dirty Memory References

- Earlier in the year, an application test was conducted on a large superdome system
- This was a scaling test. At one point, the number of cores was doubled with the expectation of obtaining almost double the throughput
  - Only a 64% increase in throughput was seen
- PC analysis revealed a large percentage of time was spent updating various statistics
  - a number of these statistics were incremented at very high rates
  - With many cores involved, almost every statistic increment would result in a dirty memory reference
- The code was modified to stop recording statistics
  - With statistics turned off, the application obtained a 270% performance gain
- Maintaining statistics on a per CPU basis is a method to avoid the dirty reads

IOPERFORM

- A feature within OpenVMS allows third party products to obtain IO Start and End information
  - This can be used to provide IO rates and IO response time information per device
  - This capability was part of the very first OpenVMS releases on the VAX platform (authored in November 1977 by a well known engineer)
- The buffers used to save the response time data are completely unaligned…
  - There is a 13 byte header and then 32-48 byte records in the buffers
  - If IOPERFORM is in use on a system with heavy IO activity – the alignment fault rate can be quite high when VMS writes these buffers

IOPERFORM (cont)

- Third party products have knowledge of the data layout
  - It is thus not possible to align the data
  - The OpenVMS routine that records the data has been taught that the buffers are unaligned and now generates safe code
- IOPERFORM also needed to wake the data collection process when there was a full buffer
  - On systems with high IO rates, we found IOPERFORM attempting to wake the data collection process over 20,000 per second
    - A wake was attempted after every IO completion was recorded if there existed a full buffer
    - Many IO completions were occurring prior to the collection process waking up and processing the buffers
  - The routine has been taught to wake the collection process no more than 100 times per second.

Summary

- The current Integrity systems perform better than existing Alpha systems in most cases
  - often by substantial amounts and with:
    - lower acquisition costs
    - reduced floor and rack space requirements
    - reduced cooling requirements
- Significant performance improvements continue to be made to OpenVMS
    - Some improvements are Integrity specific, but others apply to Alpha
- If you have performance issues or questions, send mail to:
    - OpenVMS_Perf@hp.com