BRUDEN · OSSG
On-Shore Systems Group

HP OpenVMS: Integrity Server Performance Update

October 2006

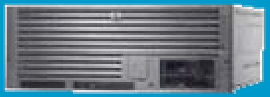
Bruce Ellis
President, BRUDEN-OSSG
Based on presentations by Greg Jordan and Burns Fisher, HP OpenVMS Engineering, Hewlett-Packard
Bruce.Ellis@BRUDEN.com

HP Integrity Servers: OpenVMS Madison9 support

New rx2620
Office Friendly
Conversion kit

rx1620
1u
2P/2C
OpenVMS V8.2,
V8.2-1, V8.3

rx2620
2u,
2P/2C
OpenVMS V8.2
V8.2-1, V8.3



rx4640
4u
4P/4C or
with MX2 8P/8C
OpenVMS V8.2
V8.2-1, V8.3

rx7620
9u
2 cell (8P/8C)
OpenVMS V8.2-1,
V8.3

rx8620
17u
4 cell (16P/16C)
OpenVMS V8.2-1,
V8.3

Superdome
Max 4 cell
(16P/16C) partition
OpenVMS V8.2-1
V8.3

OpenVMS supports the sx1000 chip set with only MAD9 processors on the rx7620, rx8620 and Superdome.

HP Integrity Servers: OpenVMS V8.3 Dual-core support

New Dual-core Processor Integrity Systems

New Entry Level System H1 2007

rx3600
4u
2P/4C
OpenVMS V8.3+

rx6600
7u
4P/8C
OpenVMS V8.3+

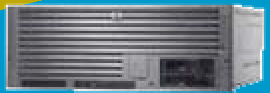
rx7640
9u
2 cell
(8P/16C)
OpenVMS V8.3

rx8640
17u
4 cell
(16P/32C)
OpenVMS V8.3

Superdome
Max 4 cell
(16P/32C)
partition
OpenVMS V8.3

Dual-core Upgrades

System board upgrade
rx2620 2u, 2P/4C
OpenVMS V8.3

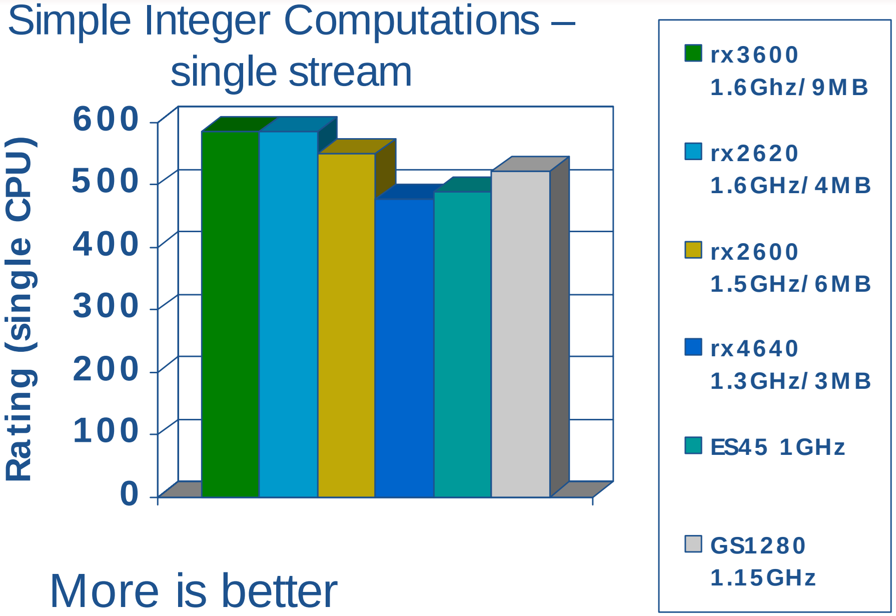


Processor upgrade
rx4640 4u, 4P/8C
OpenVMS V8.3

OpenVMS supports the sx2000 chip with only Montecito processors rx7640, rx8640 and Superdome.

CPU Performance Comparisons

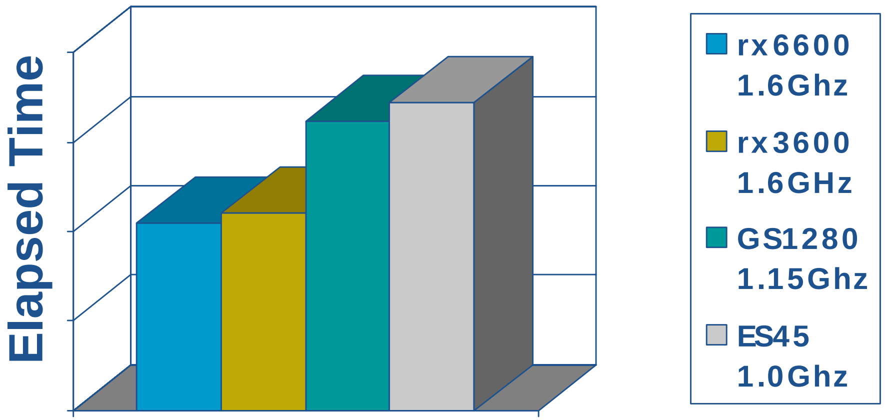
Simple Integer Computations – single stream



More is better

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

Floating Point Computations – single stream



Less is better

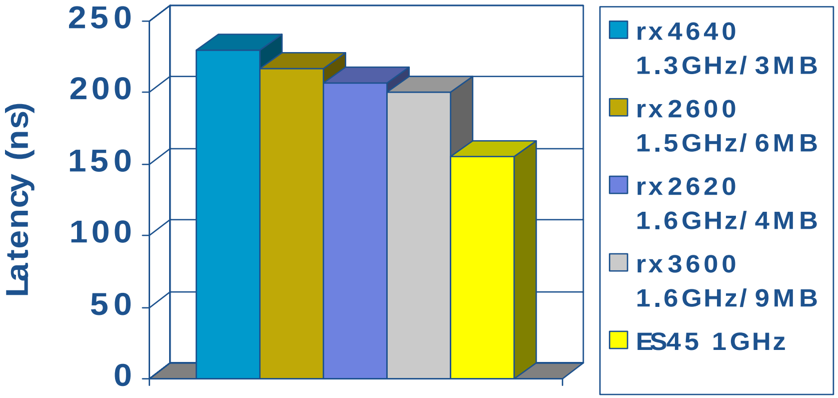
The Itanium processors are fast.

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

Various SPEC benchmarks also show the Itanium processors outperforming Alpha processors

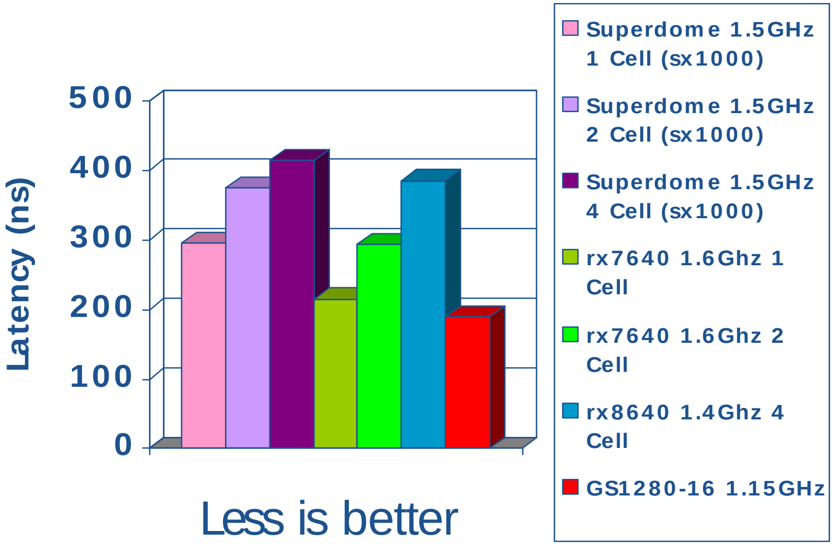
Memory Latency

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



Less is better

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



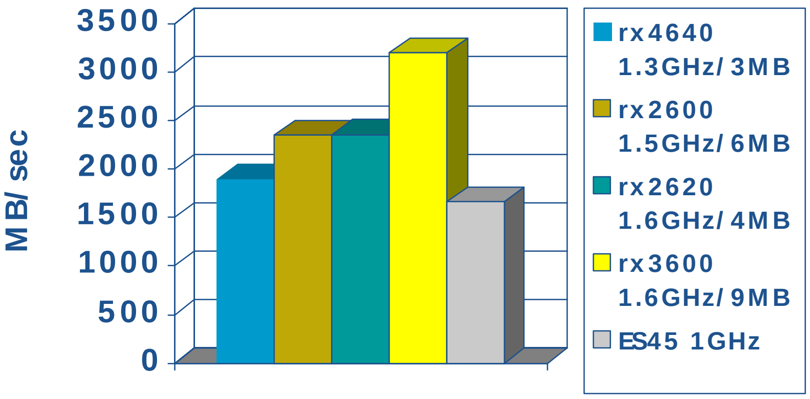
Less is better

Alpha Servers have very good memory latency

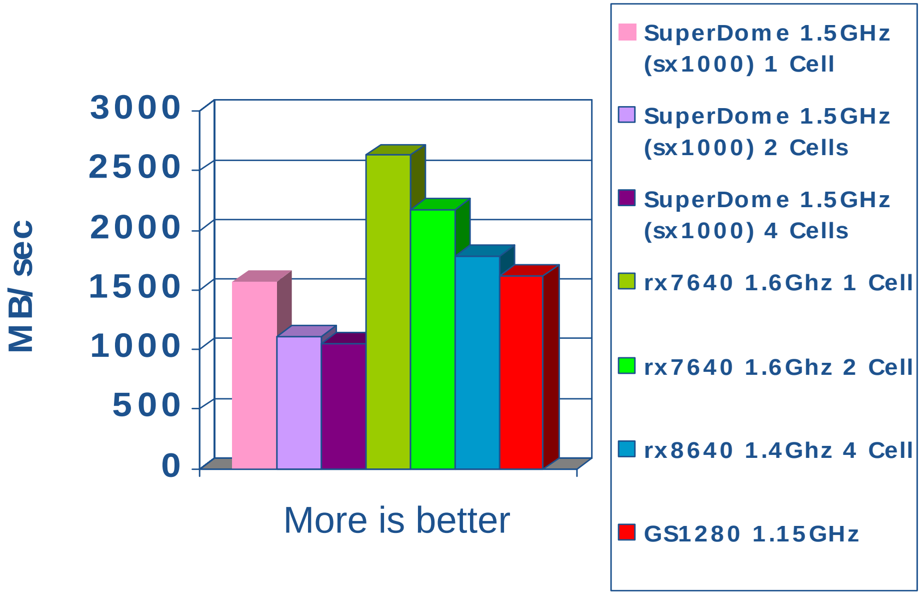
- Applications that read small amounts of data from many different memory locations should perform well

Memory Bandwidth

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



More is better



More is better

The small Integrity Servers have very good memory bandwidth

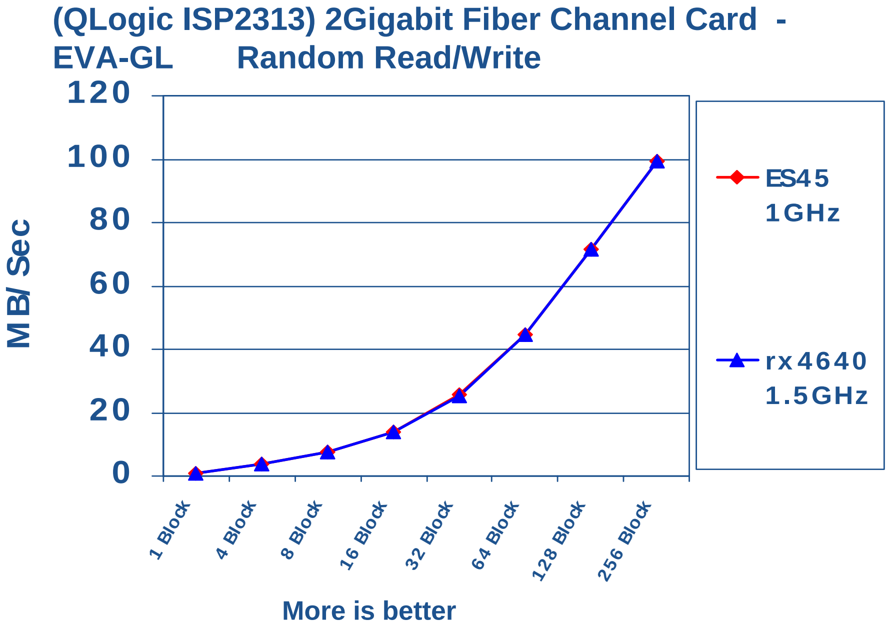
- Applications which move memory around or heavily use caches or RAMdisks should perform well

Interleaving Across Cells

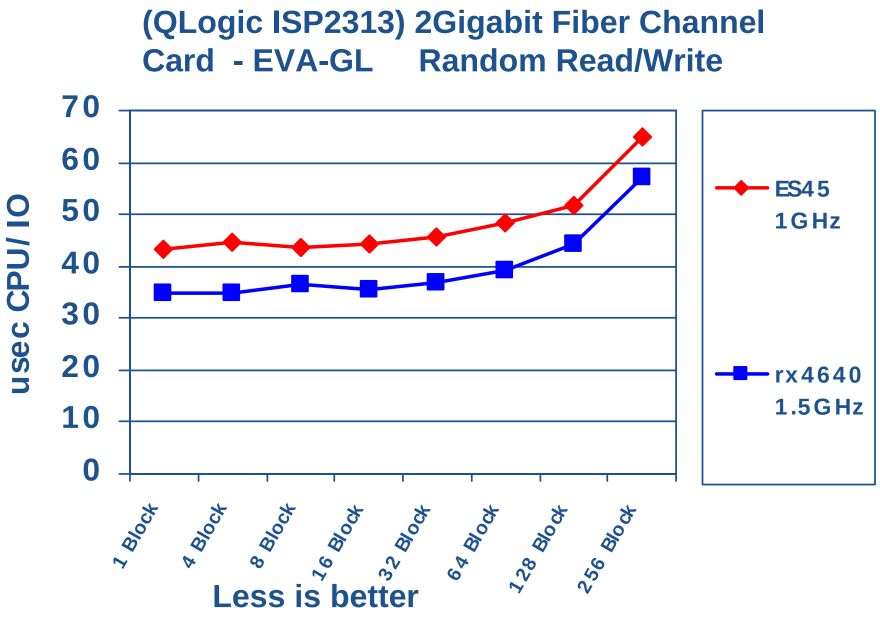
- On cell based platforms, memory can be configured as:
    - cell local
    - interleaved (the only OpenVMS supported option)
    - a combination of interleaved and cell local
- For the best interleaved performance on cell based platforms
    - The physical memory per cell should be identical
    - The number of cells should be a power of 2 (1, 2 or 4)

IO Performance

IO MB/Sec – single process



CPU per IO – single process

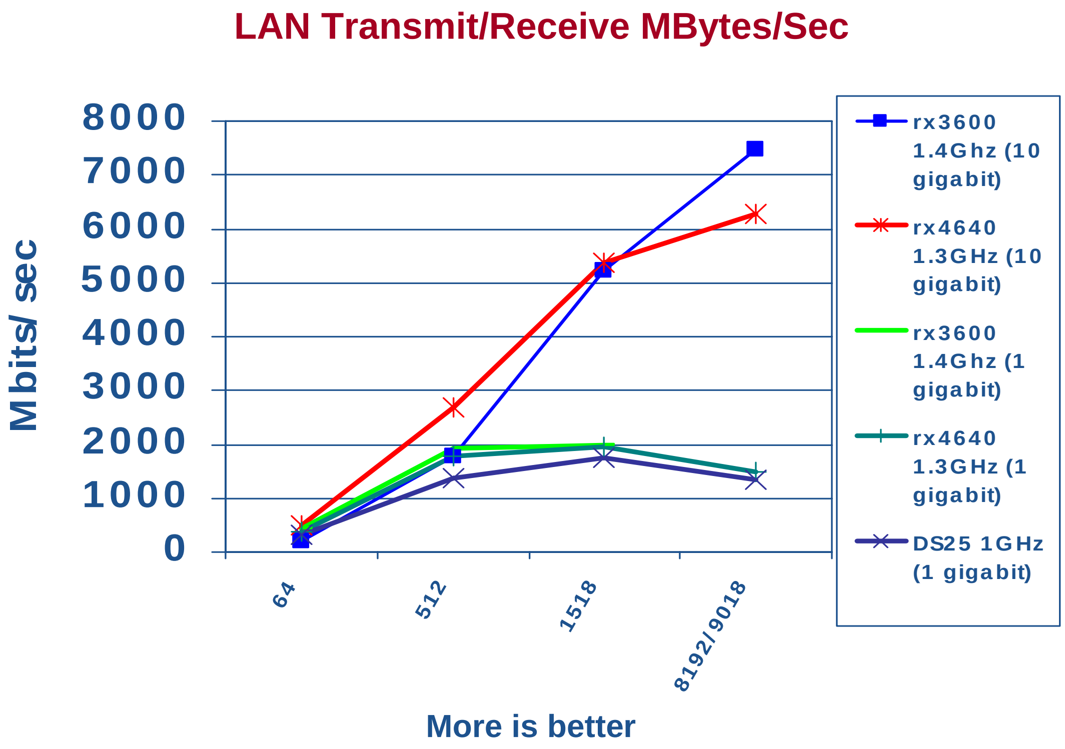


IO appears comparable between Alpha and Integrity Servers

- Both Integrity and Alpha can drive IO adapters at comparable levels

CPU cost per IO is better on Integrity servers

IO Performance



The bandwidth capabilities of 10GB LAN (by the end of 2006) and PCI-Express will only be available on Integrity

IO Performance

IO appears comparable between Alpha and Integrity Servers

- Both Integrity and Alpha can drive IO adapters at comparable levels
- CPU cost per IO is better on Integrity servers
- The bandwidth capabilities of 10GB LAN (by the end of 2006) and PCI-Express will only be available on Integrity

IO Performance and Caching

- Cached IO performance should benefit from running on Integrity Servers
  - – Better CPU Speeds
  - – Better Memory Bandwidth
    - Particularly, on smaller servers
    - On larger servers, spinlock overhead (MP Synch) may cause some degradation
  - – XFC, RAMdisk, etc. may benefit

Montecito Hyperthreads

OpenVMS V8.3 provides support for Hyperthreads on Montecito based Integrity Servers.

- Processor
  - A chip or package. 2 cores and 4 hyperthreads per Processor/Package.
- Core
  - An entity within a processor that physically executes programs.
- Hyperthread
  - An entity within a core that logically executes programs.

Montecito Hyperthreads

- CPU
  - An OpenVMS abstraction for an entity that executes programs.
- Thread of execution
  - A software concept of what a CPU executes.

Hyperthreading vs. Dual Core

Both are features of Montecito Itanium chips and are abstracted as CPUs on OpenVMS.

- Dual Core
  - Two CPUs on the same chip.
  - Separate
    - *Cache (Processor cache is divided between cores, i.e. processors with 18/24MB cache yield 9/12MB per core)*
    - *Processing units*
    - *State*
  - Shared
    - *Bus interface*
  - Simultaneous execution

Hyperthreading vs. Dual Core

- – Hyperthreads
  - A set of execution state in a core.
    - – User registers
    - – Control Registers
    - – Instruction Pointer (IP)
    - – Etc.
  - Shares execution resources with other threads.
  - Only one active thread of execution (hyperthread) at a given time.
  - Threads switch based on:
    - – Block (cache miss)
    - – Timer

Hyperthreading vs. Dual Core

- Hyperthreads
  - *O/S has no knowledge or control of hyperthread switches.*
  - Each hyperthread appears as a CPU to OpenVMS.
  - CPUs that share the same core are called "Cothread CPUs".

Enabling Hyperthreads

From OpenVMS

```
$ hthread=="$sys$test:hthreads"
$ hthread show
Hyperthread information for this system:
  These processors are capable of hyperthreading
  Hyperthreads are currently enabled.
  Hyperthreads will be enabled after the next reboot
$
$ hthread on
Hyperthread information for this system:
  These processors are capable of hyperthreading
  Hyperthreads are currently disabled.
  Hyperthreads will be enabled after the next reboot
$
```

From EFI Shell

```
Shell> cpuconfig threads [on/off]
```

Must reinitialize after change

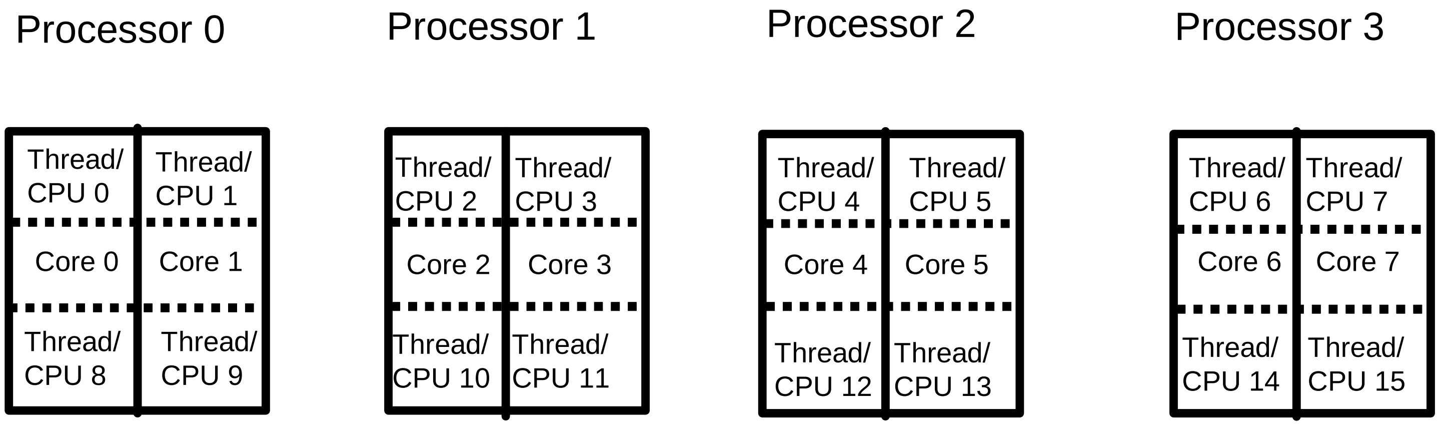
Viewing Hyperthreads

```
$ show cpu/br 0,32
System: SKD00, HP rx8640  (1.60GHz/9.0MB)
CPU 0    State: RUN                CPUDB: 82054000     Handle: 0000A5F0
         Owner: 000004C8         Current: 000004C8     Partition 0 (SKD00)
         Cothd:         32
       Process: SYSTEM               PID: 00000417
CPU 32   State: RUN                CPUDB: 824FEB00     Handle: 0000C7F0
         Owner: 000004C8         Current: 000004C8     Partition 0 (SKD00)
         Cothd:         0
$
$ show cpu/br 7,39
System: SKD00, HP rx8640  (1.60GHz/9.0MB)
CPU 7    State: RUN                CPUDB: 824CEC80     Handle: 0000AD60
         Owner: 000004C8         Current: 000004C8     Partition 0 (SKD00)
         Cothd:         39
CPU 39   State: RUN                CPUDB: 8250C580     Handle: 0000CF60
         Owner: 000004C8         Current: 000004C8     Partition 0 (SKD00)
         Cothd:         7
$
```

General View of HyperThreads

With HyperThreads, a "CPU" is something on which OpenVMS can schedule execution.

- The HyperThread defines a state, a set of registers, etc.
- Only one HyperThread can be using a core at a time.
- When two threads of execution have been scheduled on "CPUs"/HyperThreads in the same core, they can trade the use of the core, in a fashion similar to an on core context switch.



Processor 0

Thread/ CPU 0 | Thread/ CPU 1
Core 0 | Core 1
Thread/ CPU 8 | Thread/ CPU 9

Processor 1

Thread/ CPU 2 | Thread/ CPU 3
Core 2 | Core 3
Thread/ CPU 10 | Thread/ CPU 11

Processor 2

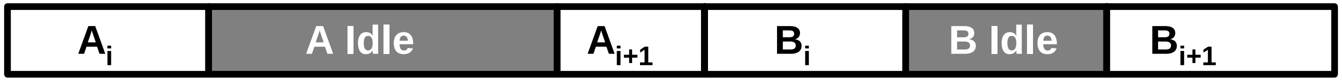
Thread/ CPU 4 | Thread/ CPU 5
Core 4 | Core 5
Thread/ CPU 12 | Thread/ CPU 13

Processor 3

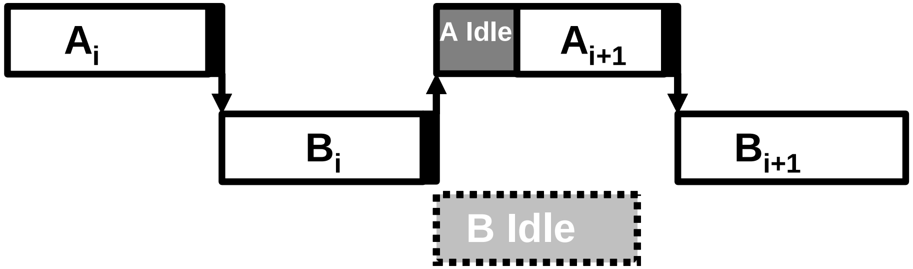
Thread/ CPU 6 | Thread/ CPU 7
Core 6 | Core 7
Thread/ CPU 14 | Thread/ CPU 15

Hyperthreading with Stalls vs Hyperthreading with No Stalls

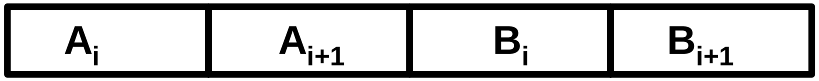
Serial Execution with Stalls (no Hyperthreading)



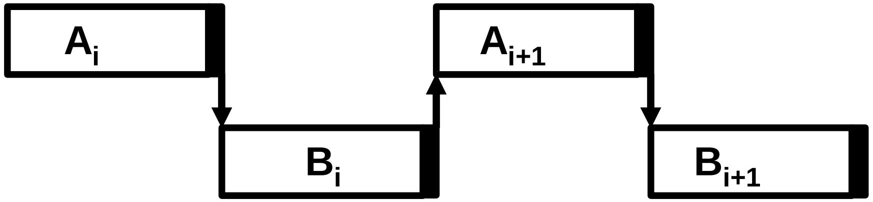
Hyperthreading with Stalls



Serial Execution with No Stalls (no Hyperthreading)

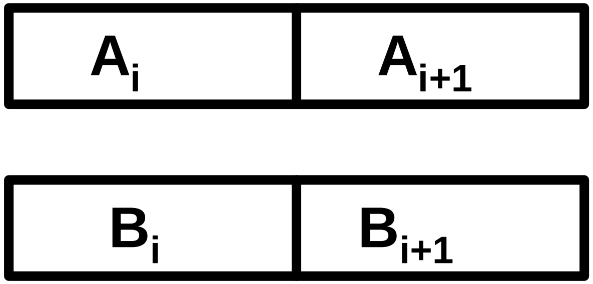


Hyperthreading with No Stalls

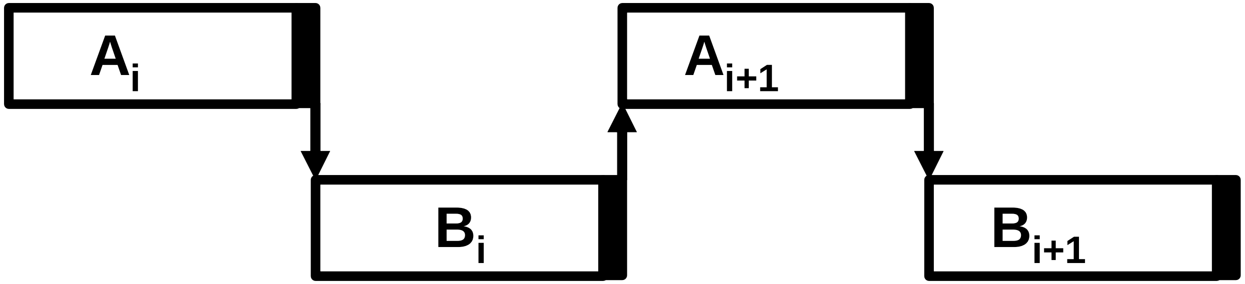


Two Cores vs Hyperthreading (NoStalls)

Serial Execution with No Stalls on Two Cores



Hyperthreading with No Stalls



Sample Run on Single Core

```
Session 1
$ set process/aff/perm/set=31
$ r tc_cf
cycle interval = 0.00000000250000 seconds
cycles 10282355676
time to execute == 25.70588919000000 seconds
Wall clock time: 49143ms
System CPU time: 25620ms
Session 2
$ set process/aff/perm/set=31
$ r tc_cf
cycle interval = 0.00000000250000 seconds
cycles 10353364312
time to execute == 25.88341078000000 seconds
Wall clock time: 49225ms
System CPU time: 25900ms
$
```

Sample Run on Single Core with CoThreads

```
Session 1
$ set process/aff/perm/set=31
$ r tc_cf
cycle interval = 0.00000000250000 seconds
cycles 14092713701
time to execute == 35.2317842525000 seconds
Wall clock time: 35322ms
System CPU time: 18640ms
$
Session 2
$ set process/aff/perm/set=63
$ run tc_cf
cycle interval = 0.00000000250000 seconds
cycles 14346088453
time to execute == 35.8652211325000 seconds
Wall clock time: 35950ms
System CPU time: 19260ms
$
```

Sample Run on Two Cores

Session 1 (Still affinitized to CPU 31)

```
$
$ r tc_cf
cycle interval = 0.00000000250000 seconds
cycles 9572697013
time to execute == 23.93174253250000 seconds
Wall clock time: 23997ms
System CPU time: 23920ms
$
```

Session 2 (Non-Co-Thread CPU)

```
$ set process/aff/perm/set=32/clear=63
$ stop/cpu 63
%SMP-I-CPUTRN, CPU #63 was removed from the active set.
$
$ r tc_cf
cycle interval = 0.00000000250000 seconds
cycles 9726287048
time to execute == 24.31571762000000 seconds
Wall clock time: 24398ms
System CPU time: 24290ms
$
```

Sample CoThread Tests

	alone	OFF	ON	Improvement
4640 (1.44GHz / 9MB)	17.9	54	51.7	4%
8640 (1.6GHz / 9MB)	21.5	63	54.8	13%
Sanddune (1.6GHz / 12MB)	23.9	67.6	52.2	23%

For the threads ON and OFF tests there were 10 copies
of the program run on 4 cores.

CoThread Considerations

- General Considerations
  - No benefit without more COM/CUR processes than cores.
  - Applications with poor locality (Many Dcache misses) will benefit
  - Applications with good locality may run slightly slower
  - Systems with higher memory latency will generally benefit more than systems with better (lower) latency
  - Tuning is critical to minimize system overhead (2% untuned vs. 13% tuned improvements on rx3600)

Areas that are Slower on Integrity

- There are a few areas where the equivalent operations on Integrity systems are slower
  - Alignment Faults
  - Exception Handling
  - Locking code in the working set (more on this coming up)
  - VAX Floating Point Data Types (due to conversions)
  - Various PAL calls (INSQUE,REMQUE)
  - Did you compile /NOOPTIMIZE?

Integrity Images are Larger

- Integrity Images are typically 3 times as large
  - Can result in more pagefaults and IO
  - Require larger GH regions if images are installed resident
  - Requires more disk space for listings and object files
  - Larger quotas may be necessary

V8.3 Performance & Scaling Enhancements

- – RMS Global Buffers in P2 Space
- – “File not found” errors perform much faster on Integrity
- – Reduced alignment faults in the OS and numerous components
- – Installed Resident Images now have code in S2 space
- – Support for shared address data for installed images

V8.3 Performance & Scaling Enhancements

- – Improvements in the code to Probe access to virtual address
- – Improvements for PEDRIVER Block Transfers
- – Reduced the time to write an Integrity crash dump
- – Lock Manager improvements for computing group grant mode (available in remedial kits too!)
- – Eliminate usage of the SCHED spinlock for PFW and PFC upcalls affects POSIX threads apps

V8.3 Performance Work

- Alignment Faults
  - On going reductions in the operating system and associated products
- Lock Manager
  - Improved Group Grant Mode computation (ECO kits back to V7.3-2)
- PEDRIVER Block Transfer Improvements
  - Changed a TB Invalidate ALL to a TB invalidate single which occurs for all block transfers
- RMS Globlal Buffers are now in 64-bit address space
  - The maximum number of global buffers per file has increased from 32k to over 2 billion
  - See the New Features manual for new commands and mixed version support

V8.3 Performance Work (Continued)

- XQP File Not Found Processing (V8.2-1 tima)
  - The design of the XQP was such that all errors were signaled
  - As noted previously, exception handling on IA64 is very slow compared to Alpha
  - Signaling a file not found error is a very common operation for the XQP especially
    - This was a big issue for applications such as web servers, etc…
  - The XQP was taught to return this status as opposed to signal the status
- Resident Image Code Region
  - The resident image code region is now created in 64-bit space
- AST Queuing and Delivery
  - The main-line code path has been streamline

V8.3 Performance Work (Continued)

- Poolzone Memory Purges
  - Reclaiming memory from pool zones (used by the XFC and the lock manager) could be slow (ECO kits back to V7.3-2)

- MONITOR ALIGN (Integrity)
  - A new MONITOR class called Align is available with V8.3
  - This will display the rate of alignment faults on the system
  - Alignment fault rates will also be broken down by mode

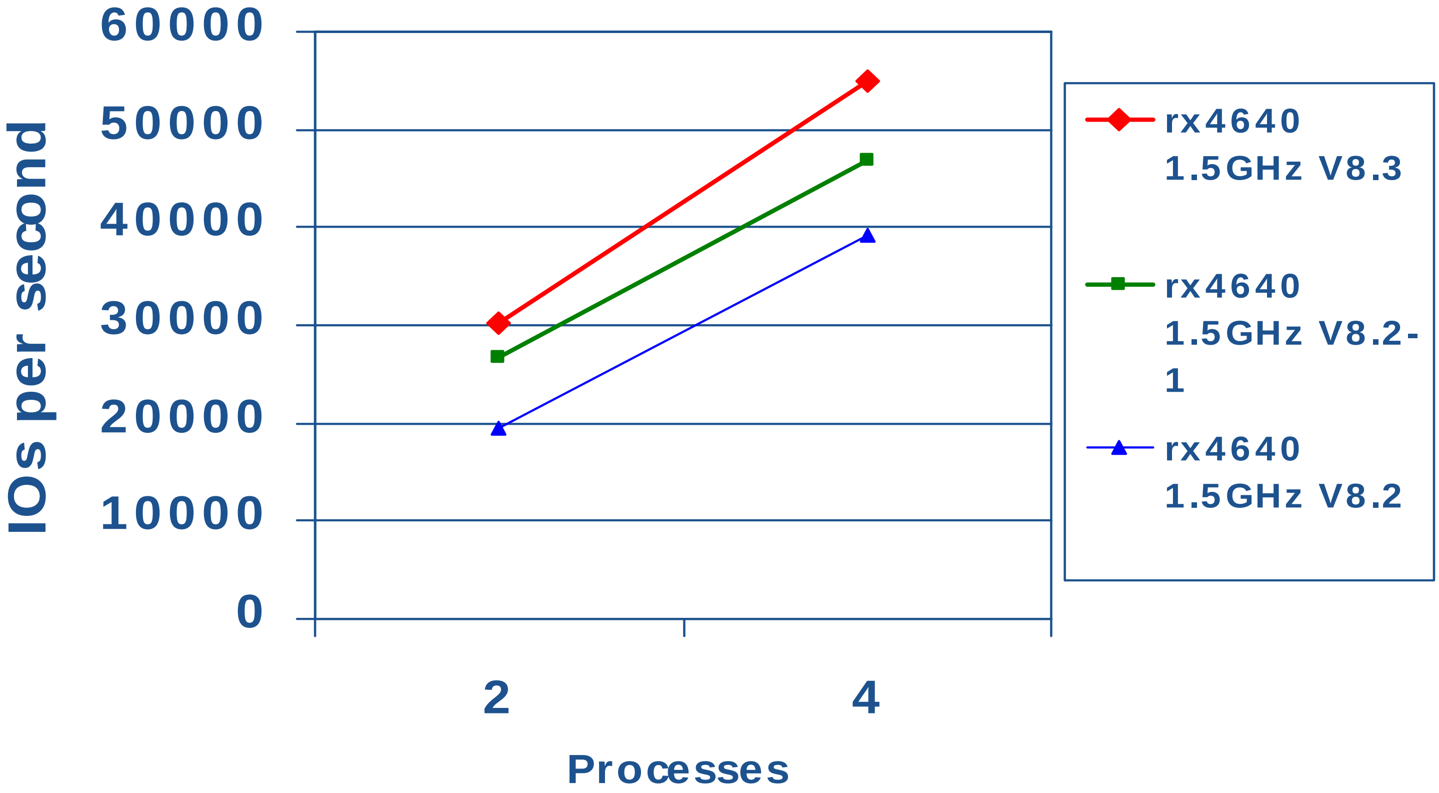
Performance Change - $LKWSET[_64]

- In OpenVMS V8.2, the behavior of $LKWSET[_64] was changed
  - $LKWSET would lock the entire image if the virtual address range fell within an image
    - The above change is necessary on Integrity platforms since it is difficult to determine all the parts of the image that must be locked when locking code sections into the working set
    - The above $LKWSET change was also made for Alpha in V8.2
  - The $LKWSET change can have a large impact on performance for code paths that frequently lock and unlock parts of an image
  - Several customers were impacted by this after migrating from OpenVMS V7.3-2 to V8.2
  - The Alpha behavior has reverted back to the prior V7.3-2 behavior via ECO kits

$LKWSET Recommendations

- For existing Alpha code – no changes should be necessary since behavior is returning to that of V7.3-2
  - For high frequency usage of $LKWSET on Integrity, there are several options:
    - Under an initialization flag or routine – lock the necessary parts of the image once
      - The above will result in the entire image getting locked on Integrity and only the necessary parts of the image for Alpha for the life of the image
    - Under an initialization flag or routine – call lib$lock_image once
    - The above will lock the entire image for both Integrity and Alpha. The lib$lock_image routine is only available as of V8.2
  - For code that resides in a very large image
    - If possible move the code that needs to be locked into a small sharable image – doing so will greatly reduce the amount of pages that need to be locked into the working set

RMS1 (Ramdisk) OpenVMS Improvements by version

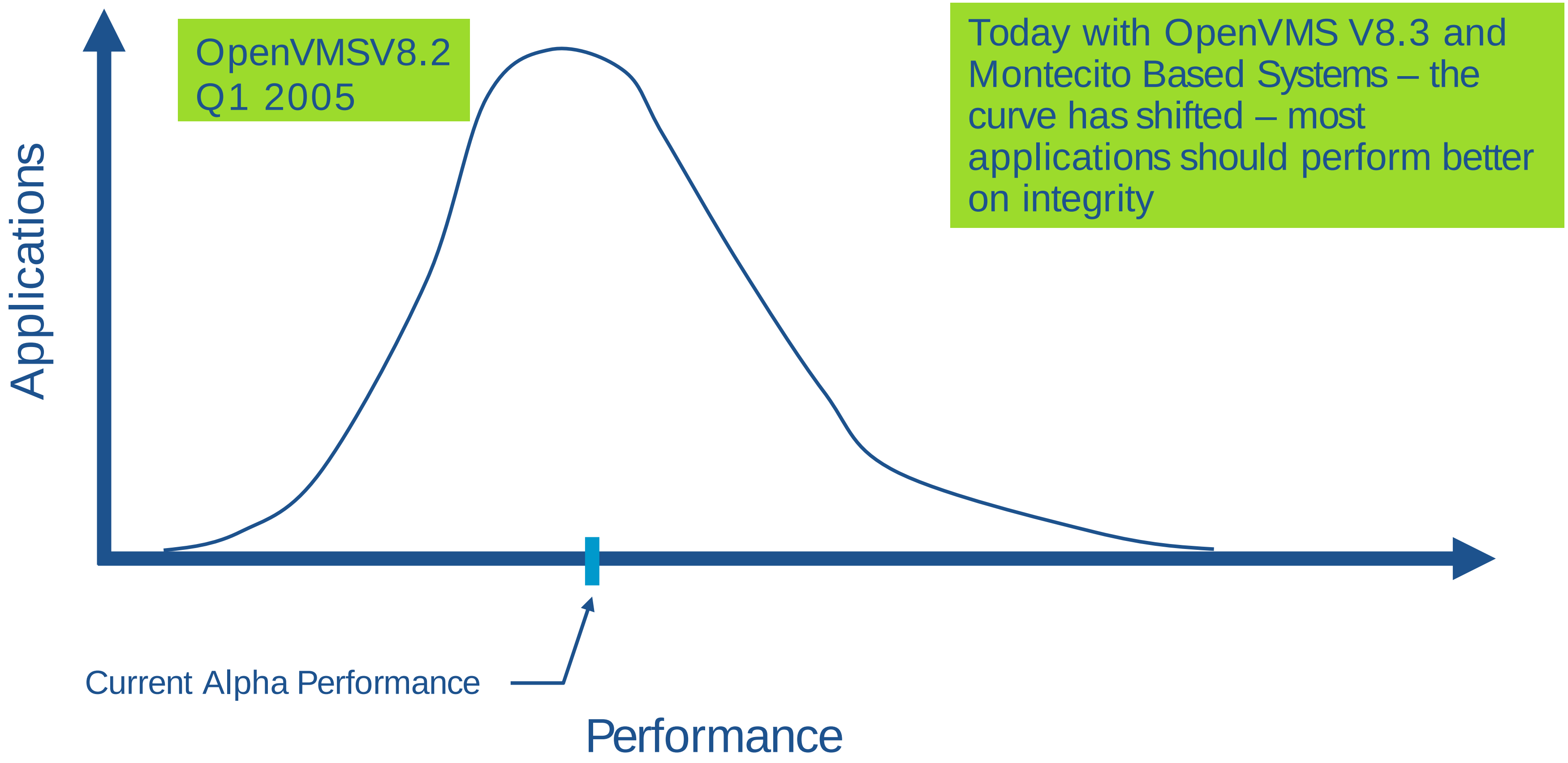


More is better

Performance

- The OpenVMS operating system performs well on Integrity servers with just a few caveats
  - alignment faults – apps as well as OS need to eliminate them
  - exception handling – we are working on significant improvements
- Each OpenVMS release shows performance improvements over the previous – 8.0, 8.1, 8.2, 8.2-1, 8.3……
- Integrity Servers have substantially lower price/performance
- Most applications should show better performance on Integrity servers
- If you port an application and are disappointed in performance, OpenVMS Engineering wants to know!
  - Please contact: OpenVMS_Perf@hp.com

The Performance Curve



OpenVMSV8.2
Q1 2005

Today with OpenVMS V8.3 and Montecito Based Systems – the curve has shifted – most applications should perform better on integrity

Questions?

BRUDEN-OSSG thanks you for attending this session.

See us at www.BRUDENOSSG.com for:

- *Performance analysis*
  - *(Performance Results Or No Expense)*
- *Porting assistance*
- *Special OPS (OpenVMS Programming Services)*