How to Get the Most Performance from Sun* JVM* on Intel® Multi-Core Servers

Kingsum Chow, Intel

Senior Staff Engineer

Dave Dagastine, Sun

Senior Staff Engineer

SSGS003

Intel Developer FORUM

Agenda

- Intel, Sun and Java – really?
- Hardware Features on Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

Intel, Sun and Java – Really?

Absolutely!

- Modern processors require carefully matched code to perform at their best.

- Tuning of the Java Virtual Machine to the processor provides:
  - Best possible performance on the platform
  - Robustness and quality across many applications

- Today's presentation covers what we've done together
  - and how you can take advantage of our work

Motivation

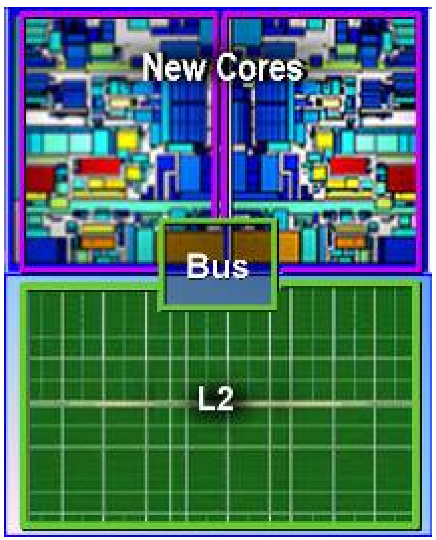
- Performance of your Java application depends on:
  - Your code
  - **Your choice of JVM and JVM parameters**
  - Your choice of operating system (Solaris, Linux, Windows, etc)
  - Your choice of hardware

Agenda

- Intel, Sun and Java – really?
- Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

New Intel® Core™ Microarchitecture

Delivering a Performance and Power Advantage for IT



FASTER

Intel® Wide Dynamic Execution Engine

4 instructions per cycle

EFFICIENT

Intel® Intelligent Power Capability

Ultra fine grained power control

SMARTER

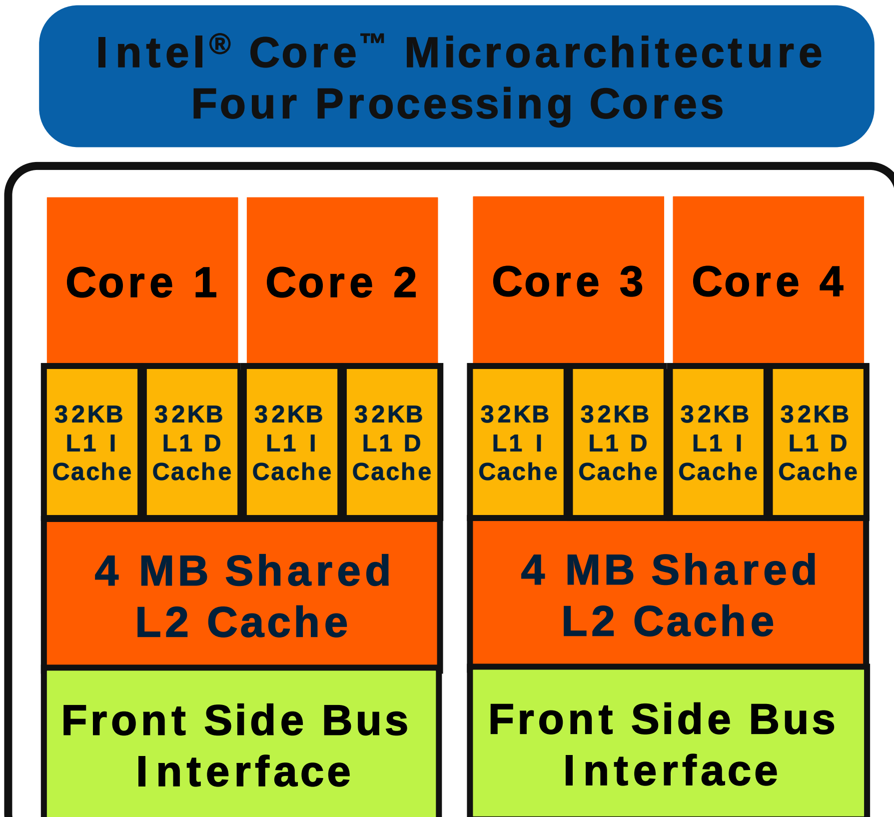
Intel® Advanced Smart Cache
Intel® Smart Memory Access

2x cache size, shared
Advanced pre-fetching

[1] compared to previous generation Dual-Core Intel® Xeon® Processor based servers

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

Intel's® Quad-Core™ Implementation



Energy Efficient Performance

Breakthrough Performance

Large L2 caches

High bandwidth

Quick access to data

IT Investment protection

Socket compatible
to 45nm quad-core

Up to 1333MHz bus

Latest Intel processors = great performance potential

Looking into the JVM…

JDK

Processor Features

JIT	Instruction Selection Instruction Ordering Dynamic Recompilation Prefetch
GC	Cache Management Memory Layout Page Management
Classlibs	Threading Performance Tuning
Interpreter	Instruction Selection

Agenda

- Intel, Sun and Java – really?
- Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

Optimization for Intel® Xeon® processors

- Plentiful Hardware Threads
  - Increase Throughput
    - Java (and JVMs) are inherently multi-threaded
    - Threading in Java is Easy!
    - java.util.concurrent
    - Parallel Garbage Collection
  - Improve Determinism
    - Concurrent Garbage Collection
    - RTSJ – Real-time System for Java
  - Both
    - Concurrent / Parallel Garbage Collection
    - Concurrent / Parallel Dynamic Compilation
    - Concurrent / Parallel Classloading

Optimization for Intel® Xeon® processors

- All those hardware threads pound memory, so must optimize memory system use
- Overcome latency (time to fetch data from memory) and bandwidth (amount of data transferred between memory and processor in a given time) limitations
- Processor / memory affinity
  - Always run a given software thread on the same hardware thread
  - Keeps data “close” to processor (caches warm)
  - OS does its best (with your help: binding, processor sets)

Optimization for Intel® Xeon® processors

- Number of simultaneously active software threads should be >= number of hardware threads
  - But may be less due to memory system limitations
  - Plan to use all the hardware threads
  - Include non-Java threads in the count: concurrent GC, native threads
- Minimize writes to shared data
  - Processor must acquire data ownership, which usually means a write to plus a read from long-latency memory
  - Synchronization requires write to shared lock word
  - Reads of shared data are ok

Optimization for Intel® Xeon® processors

- How important are memory system optimizations for your hardware?
- Optimization effectiveness depends on latency / bandwidth ratios in the memory hierarchy
  - Shared cache(s), local and remote memory
- Likely effectiveness, most to least
  - Intel Core2 L1, L2, memory

JVM Optimizations: Affinity

- Thread-Local Allocation Buffers (TLABs)
  - Java threads allocate objects in thread-private memory
  - Otherwise app serializes on shared heap access
- Parallel Thread-Local Allocation Buffers (PLABs)
  - GC threads copy live objects to thread-private memory

JVM Optimizations: Affinity / Bandwidth

- Copying garbage collectors can scatter objects around memory that were originally allocated next to each other in TLABs
- Objects allocated together are usually accessed together, so scattering causes extra memory traffic
- Object copying order can be
  - Breadth-first: copy all children, then all children's children
  - Depth-first: copy first child, then first-child's first child, ..., then second child, ...
  - Some combination of the two
- Which is better is app-dependent, but for most applications depth-first is better

JVM Optimizations: Latency (1)

- Allocation prefetch
  - Prefetch instructions can acquire cache line ownership for a processor in time for later writes
  - Allocate space in cache for the acquired line
  - When allocating objects linearly in TLABs, prefetch a platform-dependent distance ahead of address of the object being allocated
  - Subsequent allocations should find line already cached
  - Sometimes it's a good idea to prefetch multiple cache lines ahead

JVM Optimizations: Latency (2)

- Processors cache virtual-to-physical address translations in Translation Lookaside Buffers (TLBs)
- TLB size is limited, typically 8 to 64 entries
- TLB miss is expensive
  - Requires walking page table in memory
- Intel Xeon processors support large pages
  - 2 to 4 mb rather than 4 to 8 kb
  - Can map memory with many fewer TLB entries
- JVM can map Java heap and generated code cache with large pages
- Far fewer TLB misses

JVM Optimizations: Bandwidth

- Object field reordering
  - Group frequently accessed fields together so they end up in minimum number of cache lines
  - Often with object header
  - Experience shows that scalar fields should be grouped together separately from object reference fields
- Vectorization
  - Load, operate on and store multiple array elements at once with single machine instructions
  - E.g., use 8- or 16-byte loads and stores to access 4 or 8 char array elements at a time
  - Compiler-generated or tailored assembly code: e.g., System.arraycopy
  - Detect and parallelize execution in JVM
    - detect sequential data access
    - transform sequential execution code into code that takes advantage of SIMD architecture parallelism

JVM Optimizations: Latency/Bandwidth

- 64-bit JVMs enable heaps larger than 4 gb, but are ~20% slower than 32-bit JVMs
- Essentially all of the difference is due to extra memory system pressure caused by moving 64-bit pointers around
- Solution: use 32-bit offsets from a Java heap base address instead of 64-bit pointers
- If objects are highly aligned, can use > 4 gb heaps
  - If objects are 8-byte aligned, a 32-bit object offset can represent a 35-bit byte offset => 32 gb Java heap
- On Intel® Xeon® platforms, resulting 64-bit JVM can be faster than 32-bit equivalent!

JVM Optimizations: Instruction Selection

- Tune JIT code generation to match architecture:
- Take advantage of new architectural features
  - Example: Use efficient SSE instructions to move data
- Improve decode/allocation efficiency
  - Avoid length-changing prefixes to improve decode efficiency
  - Branch target alignment
- Eliminate inherent stalls in generated code
- Tune register allocation to reduce memory traffic:
  - Better register allocation can reduce stack operations
  - Use additional registers afforded by SSE or Intel64®

Sun JVM + Intel arch. = further increase performance

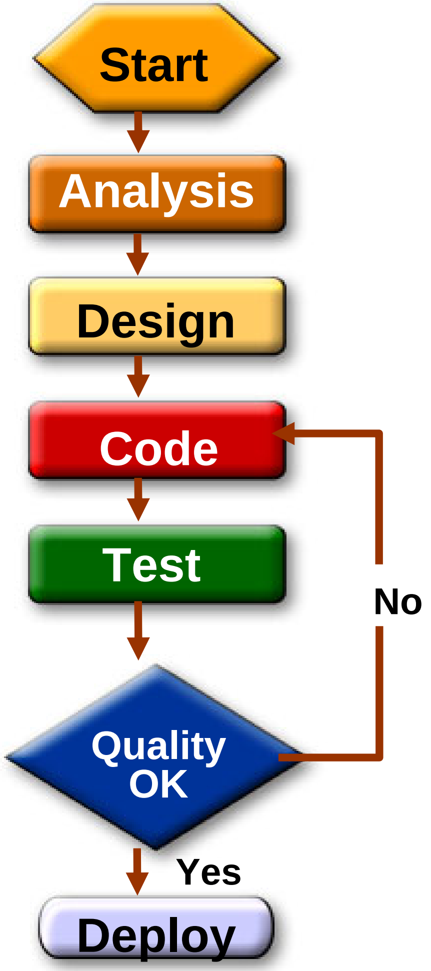
Agenda

- Intel, Sun and Java – really?
- Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

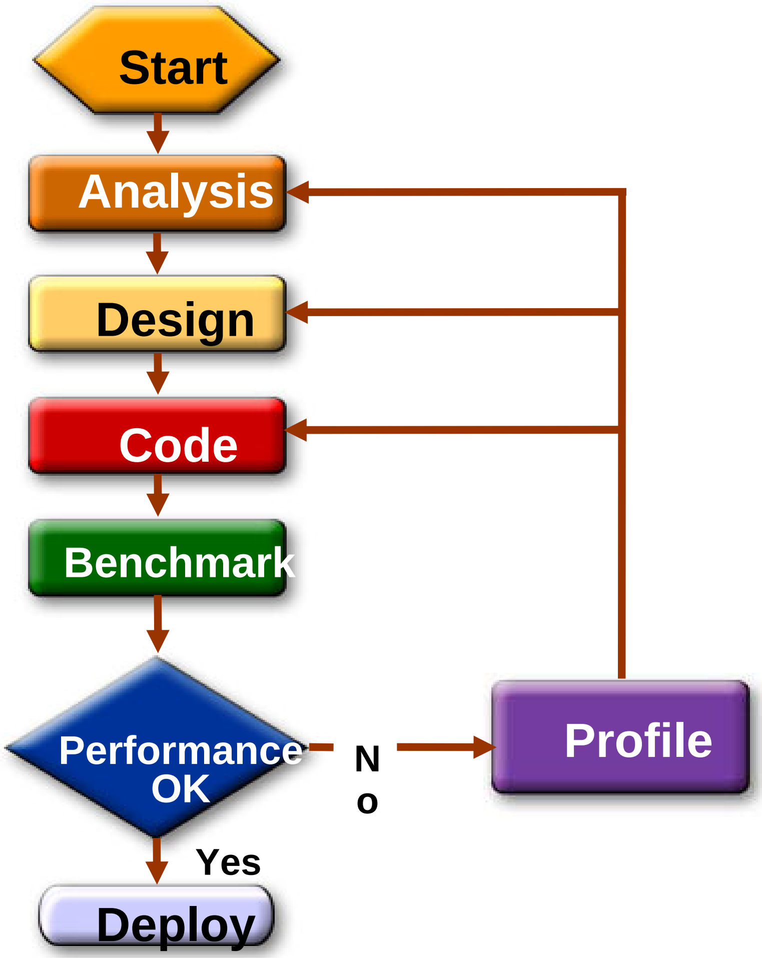
What's Tuning?

- The process of making an app run well on a particular platform
- Use the JVM to help
- Trust the JVM (but verify)
  - Don't warp your source code to compensate for perceived JVM problems
  - JVMs constantly improve
  - They optimize for the common case
  - Warped source code eventually becomes a performance liability
- http://java.sun.com/performance/reference/whitepapers/tuning.html

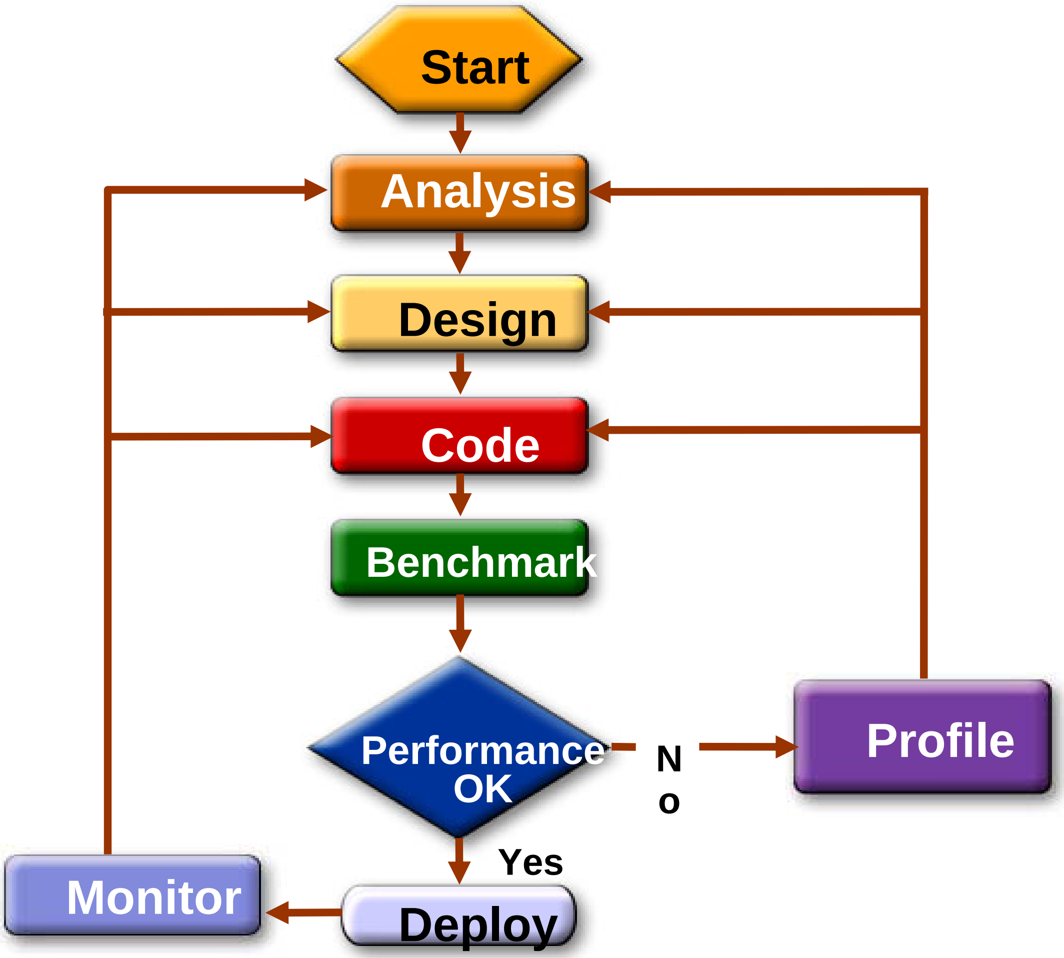
Typical Development Process



Application Performance Process



Application Performance Process



Common Bottlenecks

- Three General Categories
  - Excessive Allocation
    - Increased pressure on GC and memory systems
  - Synchronization
    - Serialization in your application will limit scalability
  - Untuned Java heap configuration, including collector selection

Reduce Object Allocation Rate

- Steps to Identify Hot Allocation Sites
  - Profile
    - Sun Studio Performance Analyzers
    - VTune
    - Netbeans Profiler
    - HPROF
  - Identify alternate strategies
    - Thread-local variables?
  - If unable to reduce allocation rate, then Tune

JVM Tuning for High Allocation Rates

- Steps to Tune the JVM
  - Observe GC behavior using VisualGC or jconsole
  - Tune Java heap and generation sizes
    - Increase overall heap and young generation sizes
    - Large heaps need a parallel collector
  - Tune TLABs
    - Not necessary with Sun's HotSpot JVM
  - Tune allocation prefetch
    - Again, not necessary with Sun's HotSpot JVM

Identify Synchronization Bottlenecks

- Steps to Identify Hot Locks
- Profile
  - Sun Studio Performance Analyzers
  - VTune
  - Netbeans Profiler
  - HPROF
- OS CPU statistics
  - High mutex spin count
  - High context switch rates
  - Unable to utilize 100% of CPU
- Identify alternate strategies
  - Maintain thread affinity
  - Use java.util.concurrent

Basic Java Heap Tuning

- First Steps to Tuning GC
- Observe GC behavior
  - -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps
  - jvmstat, VisualGC, jconsole
- Identify proper heap size
  - -Xms -Xmx
- Identify proper young generation size
  - -Xmn -XX:NewSize= -XX:MaxNewSize=

JVM Throughput Tuning

- Tuning parameters and why we use them
- -XX:+UseParallelOldGC
  - Minimize garbage collection impact on throughput
  - Does not target low pause times (use CMS for that)
- -XX:ParallelGCThreads=<n>
  - Large-scale deployment running multiple JVMs
  - By default, = #hardware threads
  - total GC threads on the system should not exceed 20: you might have expected 32
  - Experiment needed, mileage will vary

JVM Throughput Tuning

- Tuning parameters and why we use them
- -XX:+UseBiasedLocking(5-10%)
  - On by default in Java SE 6
  - Bias synchronized object to the thread that created it
  - If the synchronized block is never accessed by another thread, uses cmp+branch, not atomics, to lock/unlock
  - +3%; CAS is cheap
- -XX:+AggressiveOpts (+5-10%)
  - New wrapper flag for performance optimizations.
  - Features will be enabled by default in upcoming releases.
  - Code quality optimizations, not GC

JVM Low Pause Time Tuning

- Key Parameters for CMS
- -XX:NewRatio=N -Xmn -XX:[Max]NewSize
- -XX:SurvivorRatio=
- -XX:MaxTenuringThreshold=
  - Smaller young generation can put more pressure on CMS old generation
  - Larger young generation can increase young generation pause times
  - Experiment

JVM Low Pause Time Tuning

- Key Parameters for CMS
- -XX:ParallelCMSThreads=<n>
  - Dynamically set based on ParallelGCThreads
- -XX:ParallelGCThreads=<n>
  - Default: number of hardware threads (ncpus)
  - Try ncpus = ncpus <= 8 ? ncpus : ncpus * 5 / 8;
- -XX:CMSInitiatingOccupancyFraction=<n>
  - Old gen occupancy at which CMS starts collecting
    - Larger values improve throughput and Full GC risk
    - Lower values reduce throughput and Full GC risk

Using Large Pages on Solaris

- Enabled by default: it just works
- Default page size 4k on x64
- X86 supports 4mb pages
- X64 supports 2mb pages

Using Large Pages on Windows

- Use the local security settings console to "lock pages in memory" for the user running the application
- -XX:+UseLargePages
- For more detailed information:
  - http://java.sun.com/docs/hotspot/VMOptions.html#largepages

Using Large Pages on Linux

- Create huge page folder
  - mkdir /mnt/hugepages
- Mount the huge page file system
  - mount -t hugetlbfs nodev /mnt/hugepages
- Set permissions for read and write on the folder for the user/users that will use huge pages. By default only root will have access after mounting. In this example, all users will be allowed
  - chmod 755 /mnt/hugepages
  - chmod 777 /mnt/hugepages
- Specify how many pages you want to allocate as large pages:
  - echo 1500 > /proc/sys/vm/nr_hugepages
- Verify result:
  - cat /proc/meminfo | grep -E "(HugePage|Hugepage|Mem)"
- -XX:+UseLargePages
- For more detailed information:
  - http://java.sun.com/javase/technologies/hotspot/largememory.jsp

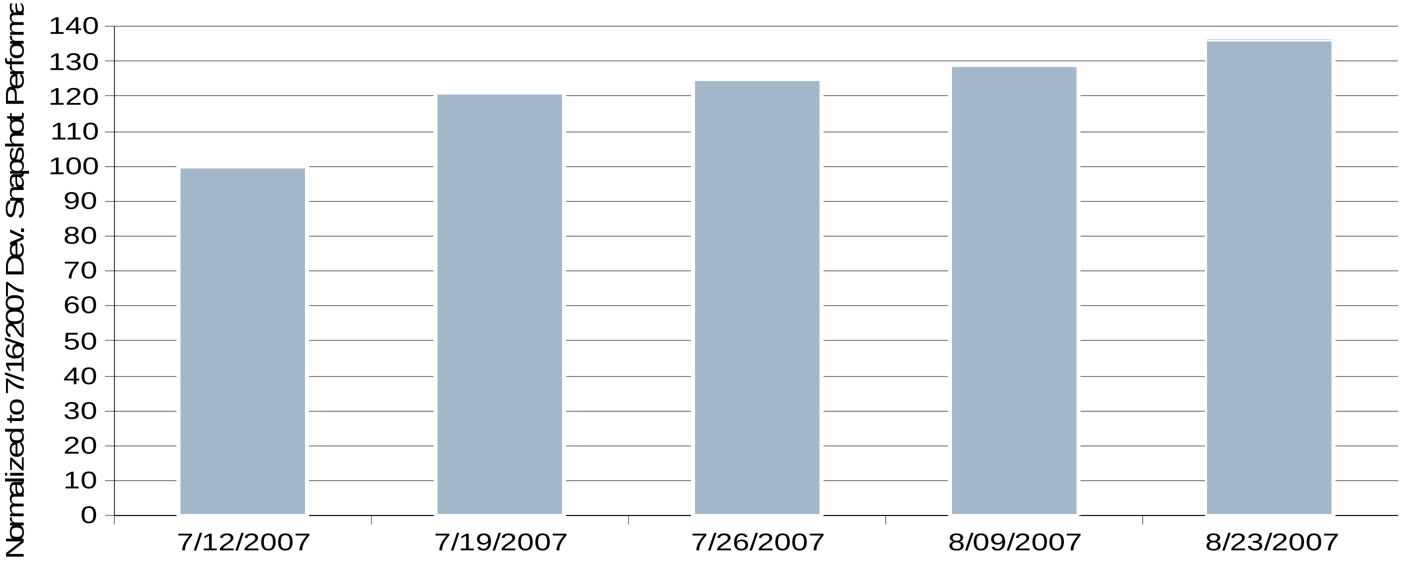
Tuning = squeeze the last bit of performance

Agenda

- Intel, Sun and Java – really?
- Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

Case Study 1 – SPECjbb2005

- Sun and Intel engineers improved performance by 20% in just 3 months (announced in JavaOne).
- Performance continued to get better after JavaOne (Chart).



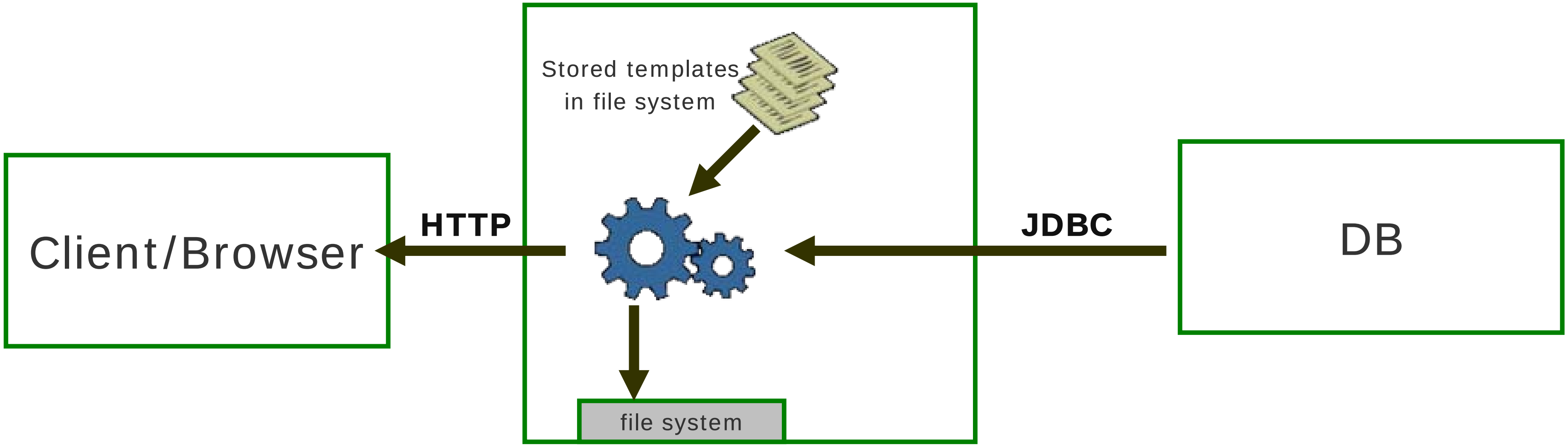
Source: Sun Microsystems, Inc.
SPECjbb2005 run on 2 x 2.66Ghz Intel Xeon processors X5355, 8GB RAM
SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation. For the latest SPECjbb2005 benchmark results, visit http://www.spec.org/osg/jbb2005.

Case Study 2 – SPECjAppServer2004

- SPECjAppServer2004 (Java Application Server) is a multi-tier benchmark for measuring the performance of Java 2 Enterprise Edition (J2EE) technology-based application servers.
  - Ref: www.spec.org
- Tuning multiple tiers (both SUT and non SUT boxes)
  - OS & network tuning
  - Java & non-Java components on the app server
- Additional tunings required on OS
- http://www.spec.org/jAppServer2004/results/res2007q3/jAppServer2004-20070619-00068.html
- Performance gain: ~1.3x (Best performance with 64-bit JVM)

Case Study 3 – Customer Workload

- A pure J2EE application
- Running on top of a J2EE Container
- Basically a XSLT transformation engine
- ~2.5x performance gain using the same default JVM parameters (response time reduction)



Case Study 4 – Customer Workload

- Data is gathered on Clovertown (2.66 GHz, 16GB memory), Linux32 RH AS4.0, and
- **defaultjdom-parsing:** jaxp_default (dom parsing): ~1.3x performance boost
- **defaultjdom:** jaxp_default (dom parsing + tree traversal): ~1.3x performance boost

Case Studies Summary

- Latest Sun JVM + Latest Intel® Xeon® processors

1.3x ~ 2.5x with Sun JDK6 for Intel Xeon processors

Agenda

- Intel, Sun and Java – really?
- Hardware Features on Intel® Xeon® Processor-based servers
- Optimizing Java SE for Multi-Core Systems
- Performance Tuning
- Case Studies
- Summary

Summary

- Latest Intel processors = great performance potential
- Sun JVM + Intel architecture = further increase performance
  - Using a VM targeted for Intel architecture may increase your application performance without changing a line of code
- Tuning = squeeze the last bit of performance
- 1.1x ~ 2.5x with Sun JDK6 for Intel® Xeon® processors

Sun JVM + Intel = High Performance for You

Additional sources of information on this topic:

- Other Sessions / Chalk Talks / Labs -
  - SSGS011– Event-Based Techniques for Software Tuning and Performance Analysis (Speaker: David A. Levinthal, Senior Software Engineer, Intel Corporation)
- More web based info:
  - http://blogs.sun.com/dagastine/
  - http://www.spec.org/

This Session presentation (PDF) is available from www.intel.com/idf. Some sessions will also provide Audio-enabled presentations after the event.

Call to Action!

- Run your Java applications with Sun JDK optimized for Intel® Xeon® processors
  - Performance gain out of the box, as Sun JDK is optimized for Intel® Xeon® processors
  - JVM and OS tunings to squeeze the last drop of performance

Risk Factors

This presentation contains forward-looking statements. All statements made that are not historical facts are subject to a number of risks and uncertainties, and actual results may differ materially. Please refer to our most recent Earnings Release and our most recent Form 10-Q or 10-K filing available on our website for more information on the risk factors that could cause actual results to differ.

Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel Inside, Xeon, Core, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
-

intel®

Leap ahead™