Maximizing Enterprise Java™ Performance on Multicore Platforms

Kingsum Chow, Intel Corporation
David Dagastine, Sun Microsystems and
Uma Srinivasan, Intel Corporation

TS-7392
Get the most performance on multicore systems for your Java™ environment applications
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
World Record SPECjbb2005* Performance

➤ Best multi-instance performance on all servers up to 4 chips
  • 464,355 SPECjbb2005 bops\(^1\)
➤ Best single-instance performance on all servers up to 16 chips.
  • 389,208 SPECjbb2005 bops\(^2\)

\(^*\)SPEC and SPECjbb2005 are registered trademarks of Standard Performance Evaluation Corporation.

\(^1\,2\)Sun Fire X4450 results submitted to SPEC. Other results as of 05/6/08 on www.spec.org. Sun Fire X4450 (4 chips, 16 cores total, Intel Xeon X7350 Quad Core 2.933GHz processors, 1066 MHz system bus, 8MB L2 Cache/chip Sun JDK 6u6-p)
Sun-Intel Collaboration

Performance Gains on Intel® Xeon® and Intel® Itanium® processors

Performance Gain on Intel Xeon processors

Performance Gain on Intel Itanium processors

> We did that with
- Multicore Processors
- Joint Java Engineering Effort – transparent to Java developers
- Performance Analysis & Tools – demo & case study

> We are going to share the secrets and the audience can apply what they learn immediately
Java Platform Performance Trends

SPECjbb2005: Intel Xeon DP

Sun and Intel collaboration started

Software performance improvements on the same hardware!
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- **Multicore Processors**
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
Quad-Core Intel® Xeon® processor

Technology Rich

- Enhanced Intel Core™ Microarchitecture
- Larger Caches
- New SSE4 Instructions
- 45 nm High-k Process Technology

Compelling IT Benefits

- Greater Performance at Given Frequency and Higher Frequencies
- Greater Energy Efficiency

Harpertown: 2nd Generation Intel Quad-Core
Quad-Core Intel® Itanium® processor

Technology Rich

More cores, higher frequency, enhanced multi-threading
Total 30 MB on die caches
Higher bandwidth memory & interconnect
Acceleration of virtualized environments

Compelling IT Benefits

Increased energy efficiency & performance-per-watt with advanced power/thermal design features
Leading edge reliability, availability & serviceability (RAS) features

Tukwila: World's first 2-billion transistor microprocessor
**Intel® QuickPath Interconnect**

**Intel® QuickPath Interconnect drives a leap forward in Platform Technology**

**Scalable solution**
- Much higher link bandwidth than Front Side Bus (FSB)
  - Headroom for higher transfer rates
- Vastly greater MP system bandwidth with multiple, independent memory controllers and Intel QuickPath Interconnect links
  - Scales efficiently with number of processors
- Many system topologies with more than four processors supported
- Common interface for Intel® Itanium® and IA-32 processor-based systems

**Improved system robustness**
- Additional levels of error recovery and logging for mission critical systems

---

**Highly Configurable System Interconnect**
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools and Demo
- Case Study and Demo
- Summary
Optimization Themes for Multicore Systems

➤ Leverage Plentiful Hardware Threads
  • Java environment and JVM™ machines are inherently multi-threaded
    • Use java.util.concurrent
  • Increase Throughput
    • 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 Themes for Multicore Systems

- 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)
  - Use Virtualization to group threads to sockets
Optimization for Multicore Systems

- Number of simultaneously active software threads should be $\geq$ number of hardware threads
  - But may be less due to memory system limitations
  - Plan to use all the hardware threads
  - More SW Threads if blocking on Network, Disk I/O
  - 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
  - If writes cannot be avoided, affinitize sharing threads to same socket
  - Reads of shared data are ok
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 Thread-Local Allocation Buffers (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
Latency (2)

- Processors cache virtual-to-physical address translations in Translation Lookaside Buffers (TLBs)
- TLB size is limited, typically 8 to 512 entries
- TLB miss is expensive
  - Requires walking page table in memory
- Modern 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
Affinity

- **Thread-Local Allocation Buffers (TLABs)**
  - Java environment 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

- **NUMA-aware allocators**
  - Chip- and / or board-local allocation regions (TLABs and PLABS)
  - Associate Java environment and GC threads with a region
  - Collect all regions when one becomes full
  - Depends on OS affinity mechanisms: e.g., Solaris™ operating system lgroups
  - Directly applicable to many systems including Xeon and Itanium NUMA systems
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.
Depth-First Performance Improvement

SPECjbb2005: Intel Xeon DP and MP

Source: Sun Microsystems
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
New Technology: Compressed Pointers

- 64-bit JVM machines enable heaps larger than 4 gb, but are ~20% slower than 32-bit JVM machines
- Difference due to extra memory system pressure caused by moving 64-bit pointers around
- Solution: use 32-bit offsets from the Java environment 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 environment heap
- On some platforms (x64), resulting 64-bit JVM machine can be faster than 32-bit equivalent!
New Technology: Escape Analysis

Definition: An object *escapes* the thread that allocated it if some other thread can ever see it.

If an object doesn't escape, we can abuse it:
- Object explosion: allocate object's fields in different places
- Scalar replacement: store scalar fields in registers
- Thread stack allocation: store fields in stack frame
- Eliminate synchronization
- Eliminate initial object zero'ing
- Eliminate GC read / write barriers

Enabled with `-XX:+DoEscapeAnalysis` in JDK™ version 6
New Technology: Tiered Compilation

- Improve startup, time-to-optimized and ultimate performance
- Single JVM machine contains interpreter and both client and server compilers
  - Right now, separate client and server JVM machines
- Client compiler generates profiled code
  - Feeds server compiler
  - Narrows profiling focus
- Server compiler guided by more accurate profile data
  - Can afford longer compile time, heavier-weight optimizations
New Technology: Garbage First

- Replaces CMS (Concurrent Mark Sweep)
- Concurrent and parallel, consistent low pause, high throughput, built-in ergonomics
  - Can specify a pause time goal: e.g., 50ms for GC every 500ms
  - (Almost) no pause time outliers

- Regionalized heap
  - Maintain estimated cost to collect each region
  - Prefer to collect regions containing lots of garbage
  - Enable segregated heap: never collect regions that don't change
  - “Young gen” is a set of regions that are always collected
Java SE Platform Libraries: New Code Optimizations

- **HashMap**
  - Faster implementation for primitive key type

- **TreeMap**
  - Faster in order traversal, addition and deletion

- **BigDecimal**

- **ReferenceQueue**

- **XML**
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
JVM Software Technology

*Code Generation & Processor Specific Optimizations*

- Optimizations for Xeon processors
- Optimizations for Itanium processors
JVM Software Technology Optimized for Xeon processors

> Tune JIT code generation to match architecture:
  - Take advantage of new architectural features
  - Example: Use efficient SSE instructions to move data

> Improve decode and resource allocation efficiency
  - Avoid length-changing prefixes to improve decode efficiency
  - Code generation to leverage macro and micro fusion benefits
    - Example: don't separate compare and branch instructions so they can be fused.
  - Branch target alignment
JVM Software Technology Optimized for Xeon processors

- Eliminate inherent stalls in generated code
  - Example: Remove Partial Register and Flag Stalls
- Tune register allocation to reduce memory traffic:
  - Better register allocation can reduce stack operations
  - Use additional registers afforded by SSE or Intel64®
- Allocation prefetch
Performance Improvements on Xeon Processors
*Impact of key features implemented*

Performance Improvements on Xeon Processors

- HashMap
- TreeMap
- BigDecimal
- Autobox Elision
- JIT/Eliminate LCP
- Improve decode/resource allocation
- Eliminate Inherent Stalls
- Tune Register Allocation
- Allocation Prefetch

% improvements in SPECjbb2005

- **Upper Bound**
- **Lower Bound**
JVM Software Technology Optimized for Itanium processors

- Basic Code Generation and Optimization Framework
  - Profiling Support: triggering method compilation, inlining, devirtualization, loop trip counts, basic block/branch frequencies
  - Mapping high level IR nodes (compiler Intermediate Representation) to machine specific encodings
  - Register allocation and assignment

- Performance impact of adding this basic support is very high
JVM Software Technology Optimized for Itanium processors

Code Generation

• Use of IP relative calls instead of indirect calls
  • avoid indirect branch misprediction penalties
  • use of long calls provides significant reach
  • better code sequence (lower number of instructions generated)

\[
\text{[MLX]} \quad \text{nop.m} \\
\text{movl} \ r29 = \text{branch target } ; ; \\
\text{[MIB]} \quad \text{nop.m} \\
\text{mov} \ br7 = r29 \\
\text{br} . \text{call} \ br0 = \ br7
\]

• Take advantage of relaxed memory ordering constraints
  • use load acquire/store release instructions instead of memory fences
  • maps well to concurrency and volatile specifications
  • allows better ordering and scheduling of instructions within single thread
  • provides better performance across concurrent threads
JVM Software Technology Optimized for Itanium processors

Register Allocation

- Using stacked registers and adjusting inlining thresholds
  - dynamic allocation of registers helps minimize spills
  - enables aggressive inlining of methods
- Using variable versus fixed size register frames
  - decreases Register Stack Engine activity in the form of reduced loads/stores to the backing store
  - provides flexibility in calling conventions between managed methods
JVM Software Technology Optimized for Itanium processors

Instruction scheduling to increase IPC (Instructions Per Cycle)

- Enhanced bundling & template generation
  - eliminate unnecessary cycle breaks
  - better ordering of instructions within a small window
  - relies on accurate modeling of underlying machine
- Scheduling across basic blocks
  - enables better packing of instructions
  - execution frequency based traces help enlarge the scope
  - can deal with instructions that have side effects
  - heuristics can leverage machine model support
JVM Software Technology Optimized for Itanium processors

Performance Improvements on Itanium Processors

- Inlining Enhancements
- Register Allocation Enhancements
- Instruction Scheduling Enhancements
- Use IP Relative Calls
- Misc. Opts Enabling (OSR, SuperWord)

% improvements in SPECjbb2005

- Upper Bound
- Lower Bound
JVM Software Technology

Future Codegen & Processor Specific Optimizations

> Nehalem optimizations
  - special instructions selection eg: for string operations

> Tukwila optimizations
  - power management sensitive code generation

> Poulson optimizations
  - take advantage of multi threading and instruction level enhancements
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
Data Driven Java Platform Performance Analysis

- Performance problems of an enterprise Java environment application may come from anywhere
  - Hardware
  - Network
  - OS
  - JVM machine
  - Java libraries
  - Java application environment

- Do not assume your code is the culprit – Collect Data

- Performance tools are available to collect data at multiple levels

- We will skip over system performance tools (e.g. Perfmon on Windows, sar/vmstat on Linux)
Java Platform Performance Tuning Methodology
Two Java Platform Performance Tools

➢ Sun Studio 12 Performance Analyzer
  • Deep profile observability into Java code, JVM, and OS
  • Full featured GUI and command line utilities
  • Simple Java command line changes to enable profiling
    • http://developers.sun.com/sunstudio/

➢ Intel® VTune™ Performance Analyzer
  • Makes application performance tuning easier with a graphical user interface and no recompiles required.
  • Compiler and language independent and it works with Java platform.
  • Offers an extensive set of tuning events for all the latest Intel® processors.
    • http://www3.intel.com/cd/software/products/asmo-na/eng/vtune
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Examples of Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
Demo
Agenda

- Performance Gains from the Joint Sun-Intel Effort
- Examples of Multicore Processors
- Processor Independent Optimizations
- Code Generation and Processor Specific Optimizations
- Java Platform Performance Tools
- Case Study and Demo
- Summary
Summary

- Intel delivers wonderful multi-core platforms based on Xeon and Itanium processors.
- Use the latest Sun JDK to enjoy all that power from the latest hardware improvements.
- Latest Sun JVM software + Latest Intel Processors = Performance Boost
  - 60% on SPECjbb2005
  - 20% on SPECjvm2008
- Many tools can help you to get more performance too.
For More Information

Technical Session

BOF's
- BOF-5268. Java™ Platform, Standard Edition: Your Performance Questions and Answers. (Tue 8:30pm)
- BOF-5218. Meet the Java™ Platform, Standard Edition (Java SE Platform) Virtual Machine Engineering Team (Wed 6:30pm)

Intel Booth
- Xeon demo: JVM software performance improvement shown on VisualVM
- Itanium demo: GlassFish™ project/Webportal monitored by VisualVM

Dave's Weblog
Acknowledgments

- Eric Delano
- Jim Fister
- Shirish Aundhe
- Yong-Fong Lee
- Eric Kaczmarek
- Kumar Shiv
THANK YOU

Kingsum Chow, Intel Corporation
David Dagastine, Sun Microsystems and
Uma Srinivasan, Intel Corporation

TS-7392