DEV5091 Up and Away: JDK Optimizations Beyond JDK 9

Vladimir Kozlov, Consulting Member of Technical Staff, Oracle vladimir.kozlov@oracle.com

Vivek Deshpande, Software Engineer, Intel Corporation vivek.r.deshpande@intel.com

October 22, 2018

Legal Disclaimer & Optimization Notice

The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
Notice revision #20110804

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction

Intel and Oracle have been collaborating to contribute many performance enhancements and features for JDK 10 and JDK 11, leveraging the faster JDK release cadence, which helps introduce new features in JDK every six months.

This session discusses some of the important contributions in the recent JDK releases, such as vectorization improvements, runtime and just-in-time compiler performance optimizations, new JVM features for getting the best out of underlying hardware, and new JDK APIs for faster I/O. Along with implementation details, it covers performance benefits of this work for various workloads and use cases.

OpenJDK Community - working together to improve Java

We are very proud to have very active and dedicated OpenJDK Developers Community which support Java for many years.

As I remember Intel was first company we start collaboration to optimize Java performance back in 2007. In 2014 Intel joined OpenJDK community.

We worked with AMD for few years when amd64 architecture was introduced.

SAP joined Community when they ported HotSpot for ia64.
SAP and IBM contributed Linux/PowerPC and AIX/PowerPC ports in JDK 8 and Linux/s390 in JDK 9.

RedHat contributed AArch64 port in JDK 9. In following releases they and other companies are working hard on optimizations for this port.

Google become very active this year in OpenJDK.

And it is only small part of OpenJDK Community - it has a lot more people and companies.

Features refresh: JDK 9

JDK 9 was released just one year ago on September 21, 2017. It was long time in making - we started work on JDK 9 in 2013.

The main feature in JDK 9 was Java Modules (Project Jigsaw), JEP-261.

Other important features in JDK 9:

- Variable Handles, JEP-193
- JShell Read-Eval-Print Loop, JEP-222
- Compact Strings, JEP-254
- Collection Factories, JEP-269
- Linux/AArch64 Port, JEP-237
- Linux/s390x Port, JEP-294
- Ahead-of-Time Compilation, JEP-295

Features refresh: JDK 9 (cont)

Total number of Enhancements integrated into JDK 9 Hotspot JVM was around 1450.

Next major features were added to Hotspot JVM in JDK 9:

- Segmented Code Cache (for Tiered JIT compilation), JEP-197
- Java-Level JVM Compiler Interface (JVMCI), JEP-243
- Make G1 the Default Garbage Collector, JEP-248

X86 code generation main optimizations added to Hotspot JVM:

- Add support for AVX512, JDK-8076276
- AVX512 equipped inflate, has_negatives & compress intrinsics, JDK-8154974
- Use AVX512 instructions for Arrays.equals() intrinsic, JDK-8145717
- Use AVX512 instructions for string compare, JDK-8144771

Features refresh: JDK 9 (cont)

X86 code generation optimizations added to Hotspot JVM (cont):

- Enables SHA Extensions on x86, JDK-8150767
- Update for x86 SHA256 using AVX2, JDK-8154495
- Support vector conditional move (CMovVD) on Intel AVX cpu, JDK-8139340
- Intrinsify fused mac (FMA) operations on x86, JDK-8154122
- CRC32C implementations for x86/x64 targets, JDK-8134553
- Use Intel's LIBM implementation for Math functions: sin, cos, tan, log, log10, pow, exp.

JDK 9 could be considered as Major release based on time we spent and number of implemented features but it was only short-term-support release - it reached its *end of life* (and *end* of public support) in March 2018 in accordance with new Java release cadence.

Features refresh: JDK 10

JDK 10 was released on March 21, 2018 - just 7 months ago. We started work on it in January 2017 (separate repository was created).

New features added in JDK 10:

- Local-Variable Type Inference, JEP-286
- Garbage Collector Interface, JEP-304
- Parallel Full GC for G1, JEP-307
- Application Class-Data Sharing, JEP-310
- Thread-Local Handshakes, JEP-312
- Heap Allocation on Alternative Memory Devices, JEP-316
- Experimental Java-Based JIT Compiler (Graal), JEP-317

Features refresh: JDK 10 (cont)

Total number of Enhancements integrated into JDK 10 Hotspot JVM was around 400.

General code generation optimizations in Hotspot JVM:

- Intrinsify Math.multiplyHigh(long, long), JDK-8187684
- Unrolling more when SLP auto-vectorization failed, JDK-8187601
- C2: loop strip mining, JDK-8186027
- C2: allow vectorization of HeapByteBuffer.putInt loops, JDK-8182475
- Analyze subword in the loop to set maximum vector size, JDK-8175096

Features refresh: JDK 10 (cont)

X86 optimizations added to Hotspot JVM:

- Support cmov vectorization for float, JDK-8192846
- Support vectorization of Math.sqrt() on floats, JDK-8190800
- FMA Vectorization on x86, JDK-8181616

JDK 10 was only short-term-support release - it reached its *end of life* (and *end* of public support) in September 2018.

Features refresh: JDK 11

JDK 11 was released on September 25, 2018. We started work on it in December 2017 (first changes were pushed).

New main features added in JDK 11:

- Nest-Based Access Control, JEP-181
- Dynamic Class-File Constants, JEP-309
- Epsilon: A No-Op Garbage Collector, JEP-318
- ZGC: A Scalable Low-Latency Garbage Collector (Experimental), JEP-333
- Local-Variable Syntax for Lambda Parameters, JEP-323
- Unicode 10, JEP-327
- Flight Recorder, JEP-328

Features refresh: JDK 11 (cont)

New features added in JDK 11 (cont):

- Launch Single-File Source-Code Programs, JEP-330
- Low-Overhead Heap Profiling, JEP-331

Total number of Enhancements integrated into JDK 11 Hotspot JVM was around 500.

General code generation optimizations in Hotspot JVM:

- Lazy allocation of compiler threads, JDK-8198756
- C2 should leverage profiling for lookupswitch/tableswitch, JDK-8200303

Features refresh: JDK 11 (cont)

X86 optimizations added to Hotspot JVM:

- Optimize (masked) byte memory comparisons on x86, JDK-8203628
- AES CBC decryption algorithm using AVX512 instructions, JDK-8205398
- Use XMM/YMM for objects initialization, JDK-8201193
- Add support for vpclmulqdq for CRC32, JDK-8200067
- Add support for vector vpopcnt, JDK-8199421

JDK 11 if the first long-term-support (LTS) release - its *end* of **Premier** support is September 2023 (with **Extended** support until 2026).

JDK 10 and 11 x86 optimizations in details

1. Improvements in vectorization
2. Improve intrinsics performance by using AVX512
3. FMA vectorization
4. Base64 intrinsics

Performance data in following slides was produced on next platform:

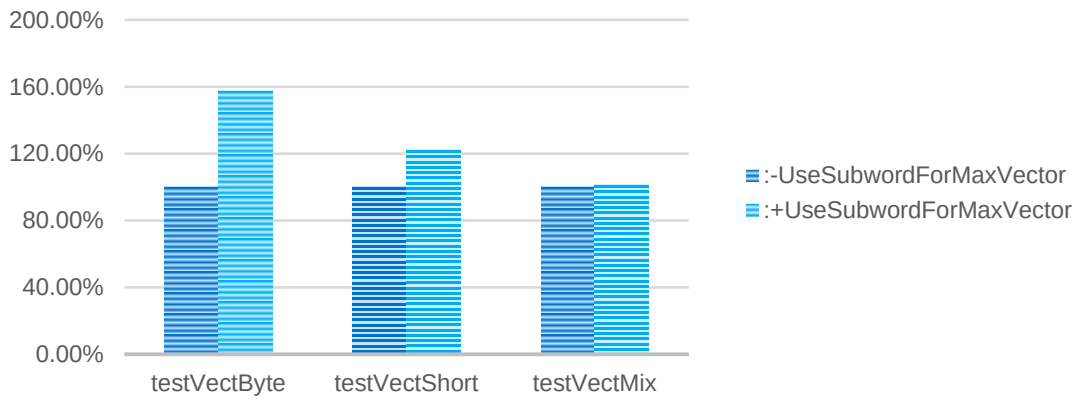
- Intel® Xeon® Platinum 8180 @ 2.5GHz, 384 GB RAM
- Software:
  - OS: Red Hat Enterprise Linux Server release 7.4
  - Openjdk version "11" 2018-09-25

    OpenJDK Runtime Environment 18.9 (build 11+28)

Optimization for small element vectors

Use Full Vector width for byte, short operations

```
@Benchmark
public void testVectByte() {
    for (int i = 0; i < size; i++) {
        data[i] = (byte)(data2[i] + data3[i]);
    }
}
```



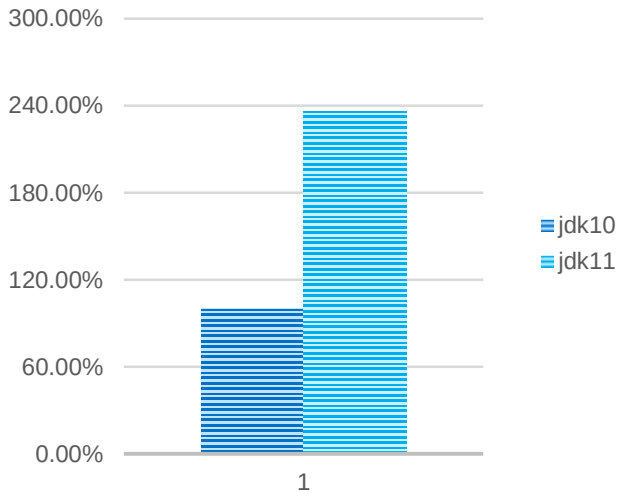
Before:
vmovdqu 0x10(%r8,%rbx,1),%xmm1
vpaddb 0x10(%rcx,%rbx,1),%xmm1,%xmm1
vmovdqu %xmm1,0x10(%rbp,%rbx,1)
vmovdqu 0x20(%r8,%r10,1),%xmm1
vpaddb 0x20(%rcx,%r10,1),%xmm1,%xmm1
vmovdqu %xmm1,0x20(%rbp,%r10,1)

After:
vmovdqu32 0x10(%r8,%rbx,1),%zmm1{%k1}{z}
vpaddb 0x10(%rcx,%rbx,1),%zmm1,%zmm1
vmovdqu32 %zmm1,0x10(%rbp,%rbx,1){%k1}
vmovdqu32 0x50(%r8,%r10,1),%zmm1{%k1}{z}
vpaddb 0x50(%rcx,%r10,1),%zmm1,%zmm1
vmovdqu32 %zmm1,0x50(%rbp,%r10,1){%k1}

Array Compare Optimization

New improvement in vectorizedMismatch() intrinsic

https://bugs.openjdk.java.net/browse/JDK-8205194

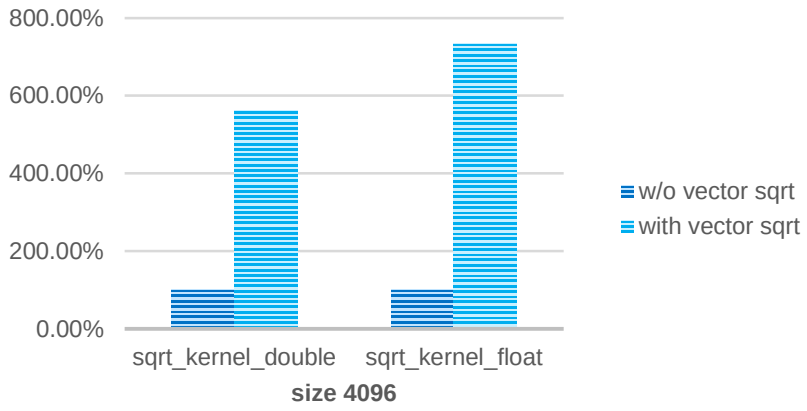


Source: Intel, as of July 9, 2018. The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations Results based on Intel measurements and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter. Configuration: See slide 4.

Sqrt Vectorization

```
@Benchmark
public void sqrt_kernel_float() {
   for (int i = 0; i < size; i++) {
      out[i] =   (float)Math.sqrt(left[i]);
   }
}
```

```
:
vsqrtps 0x50(%r11,%r8,4),%zmm0{%k1}{z}
vmovdqu32 %zmm0,0x50(%rax,%r8,4){%k1}
vsqrtps 0x90(%r11,%r8,4),%zmm0{%k1}{z}
vmovdqu32 %zmm0,0x90(%rax,%r8,4){%k1}
:
```



Source: Intel, as of July 9, 2018. The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations Results based on Intel measurements and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter. Configuration: See slide 4.

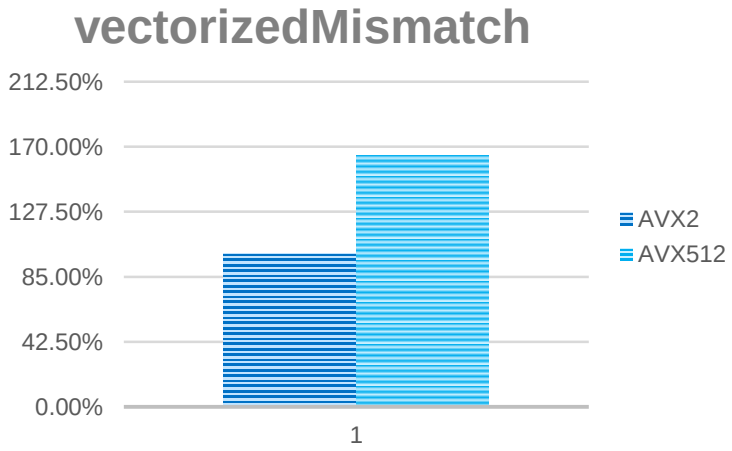
Improve intrinsics performance by using AVX512

Lexicographic array compare - ArraysSupport.vectorizedMismatch()

String Compress - StringUTF16.compress()

String Inflate - StringLatin1.inflate()

StringCoding.hasNegatives()



Source: Intel, as of July 9, 2018. The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations Results based on Intel measurements and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter. Configuration: See slide 4.

FMA (fused multiply–add)

Java semantics doesn’t allow use of FMA instructions by combining float/double multiply and add operation combination.

FMA API was added in JDK9 as part of java.lang.Math and StrictMath

The intrinsic for the API uses the FMA instruction available on the processor

FMA Vectorization support added in JDK10

Useful in HPC and Machine Learning

FMA

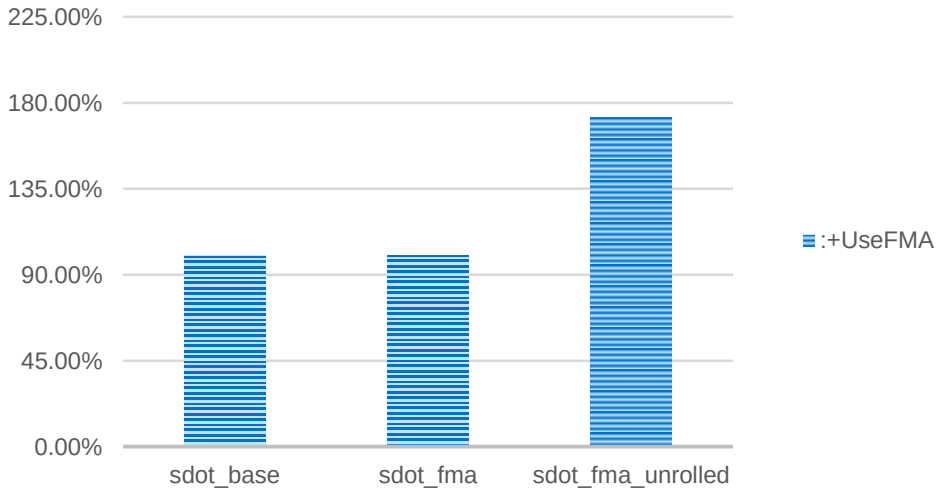
```
@Benchmark
public float sdot_base() {
  float sum = 0f;
  for (int i = 0; i < size; ++i) {
    sum = a[i]* b[i] + sum;
  }
  return sum;
}

@Benchmark
public float sdot_fma() {
  float sum = 0f;
  for (int i = 0; i < size; ++i) {
    sum = Math.fma(a[i], b[i], sum);
  }
  return sum;
}
```

```
:
:
vmovss 0x10(%rcx,%rdi,4),%xmm2
vmovss 0x10(%r9,%rdi,4),%xmm1
vmovss 0x14(%rcx,%rdi,4),%xmm4
vfmadd231ss %xmm1,%xmm2,%xmm0
vmovss 0x14(%r9,%rdi,4),%xmm1
vmovss 0x18(%rcx,%rdi,4),%xmm3
vfmadd231ss %xmm1,%xmm4,%xmm0
:
:
```

FMA Vectorization

```
@Benchmark
public float sdot_fma_unrolled() {
    float sum = 0f;
    int s_size = size/div;  //div = 16
    for (int i = 0; i < s_size; i++)
        s0[i] = Math.fma(a[0*s_size+i],  b[0*s_size+i], s0[i]);
    for (int i = 0; i < s_size; i++)
        s0[i] = Math.fma(a[1*s_size+i],  b[1*s_size+i], s0[i]);
    :
    :
    for (int i = 0; i < s_size; i++)
        s0[i] = Math.fma(a[15*s_size+i],  b[15*s_size+i], s0[i]);
    for (int i = 0; i < s_size; i++)
        sum += s0[i];
    return sum;
}
```

```
:
vmovdqu32 0x50(%rcx,%r8,4),%zmm0{%k1}{z}
vmovdqu32 0x50(%r11,%r8,4),%zmm5{%k1}{z}
vfmadd231ps 0x50(%rbp,%r8,4),%zmm5,%zmm0{%k1}{z}
vmovdqu32 %zmm0,0x50(%rcx,%r8,4){%k1}
vmovdqu32 0x90(%rcx,%r8,4),%zmm0{%k1}{z}
vmovdqu32 0x90(%r11,%r8,4),%zmm5{%k1}{z}
vfmadd231ps 0x90(%rbp,%r8,4),%zmm5,%zmm0{%k1}{z}
vmovdqu32 %zmm0,0x90(%rcx,%r8,4){%k1}
:
```

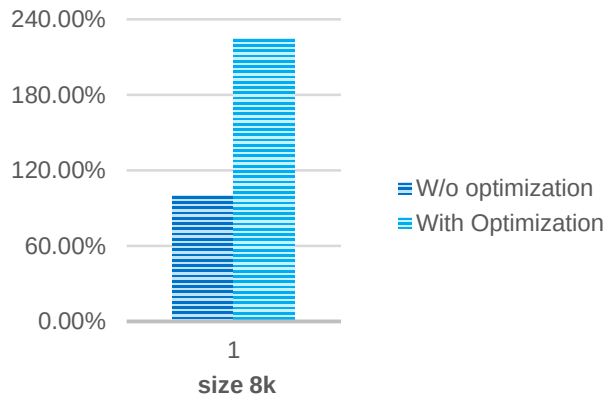


Source: Intel, as of July 9, 2018. The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations Results based on Intel measurements and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter. Configuration: See slide 4.

Base64 intrinsic

- Used to represent binary data in an ASCII string format by translating into base64 representation (set of 64 characters)
- Commonly used to encode binary data that is stored and transferred over media that deals with textual information (for example, http headers)
- Converts 3 byte input (3 * 8 bits) into four 6-bit base64 encoded output (each 6 bit output corresponds to a Base64 encoded digit)
- AVX512 Based Intrinsic

```
Encoder encoder = Base64.getEncoder();
:
:
for (long i = 0; i< 10000000; i++) {
  len+= encoder.encode(srcArr, resArr);
}
```



Source: Intel, as of July 9, 2018. The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations Results based on Intel measurements and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter. Configuration: See slide 4.

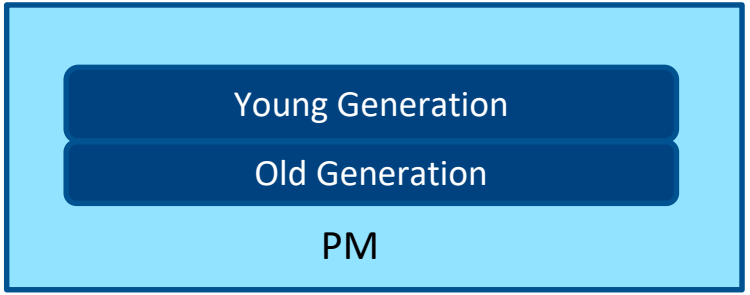
Java on Intel® Optane™ DC Persistent Memory(PM)

Whole Java Heap on PM

Partial Java Heap on PM

Application directed Allocation on PM

Whole Java heap on PM



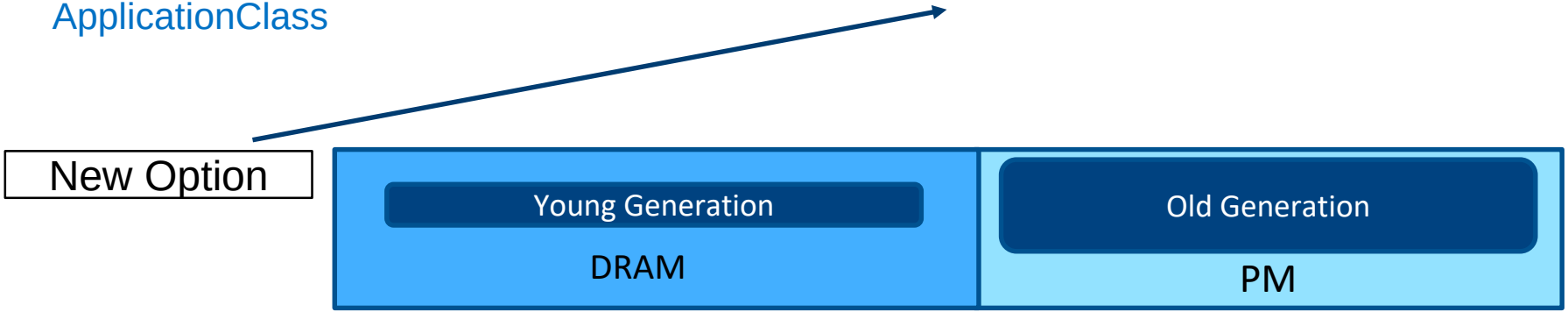
- Java Enhancement Proposal

  https://bugs.openjdk.java.net/browse/JDK-8171181
- Places whole heap on PM, the code and meta-data in DRAM
- Available from JDK 10 released in March 2018
- Cloud/Multi-jvm usage for background and non latency sensitive JVM processes with low write memory bandwidth requirement
- *Usage: java -Xmx320g -Xms320g -XX: AllocateHeapAt=/mnt/pmem12 ApplicationClass*

New Option

Partial Java Heap on PM

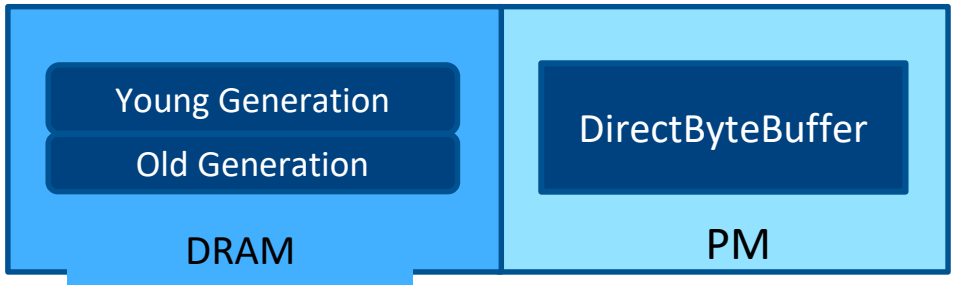
- https://bugs.openjdk.java.net/browse/JDK-8202286
- Prototype: http://cr.openjdk.java.net/~kkharbas/8204908/webrev.05
- Places old gen on PM and young gen on DRAM
- Potential usage for Java caching workloads like HBASE and NoSQL
  - ➢ The cache for these are long lived and get promoted to old gen
  - ➢ Could provide large capacity for cache and increased hit rate
- Usage: *java -Xmx640g –Xms640g –Xmn40g* -XX:AllocateOldGenAt=/mnt/pmem12 ApplicationClass



Application directed allocation on PM

Application directed allocation of Java objects on PM for volatile use

- Off Java Heap Allocation
  - ➢ Using existing Java DirectByteBuffer mechanism and File Channel Map facility
    - Serialization/De-serialization
    - DirectByteBuffer size limitations



DirectByteBuffer on PM: Example Code

```
import java.nio.channels.FileChannel;
import java.nio.channels.FileChannel.MapMode;
import java.nio.ByteBuffer;
import java.nio.ByteBuffer;
import java.io.RandomAccessFile;

class offheap
{
   public static void main(String[] args) throws Throwable
   {
      // The filesystem /mnt/pmem12 is mapped on PM
      // Open a file myfile on PM for read/write
      RandomAccessFile pmemfile = new RandomAccessFile("/mnt/pmem12/myfile", "rw");
      // Get the corresponding file channel
      FileChannel pmemchannel = pmemfile.getChannel();

      // Map a direct byte buffer in read/write mode
      ByteBuffer pmembb = pmemchannel.map(MapMode.READ_WRITE, 0, (1 << 31)-1);
```

DirectByteBuffer on PM: Example continued

```
// The direct byte buffer is now allocated on AEP
        for (int j = 0; j < 15; j++) {
          // Perform writes
          for (int i = 0; i < (1<< 31)-1; i++) {
             pmembb.put(i, (byte)((i+j)&0xff));
          }
          // Perform reads and verify
          for (int i = 0; i < (1<< 31)-1; i++) {
             if (pmembb.get(i) != (byte)((i+j)&0xff)) {
                 System.out.println("Erorr at index:" + i);
             }
          }
        }

        // Truncate the file to 0 size
        pmemchannel.truncate(0);
        // Close the file channel
        pmemchannel.close();
        // Close the file
        pmemfile.close();
    }
}
```

DirectByteBuffer using JNI

- DirectByteBuffer (DBB) can be allocated on PM using JNI
- The memkind_alloc function/mmap can be used for allocation on PM in JNI
  - memkind_alloc() available as part of memkind library

    https://github.com/memkind/memkind
  - Header file: memkind.h
- JNI has a mechanism called NewDirectByteBuffer which can form a DBB with the given underlying pre-allocated memory region

https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/functions.html#NewDirectByteBuffer

Vector API

Express vector algorithms with explicit vectors in Java.

```
for (int i = 0; i < a.length; i++) {
  c[i] = a[i] + b[i];
}
```

```
for (int i = 0; i + spec.length() < a.length; i += spec.length()) {
  FloatVector<S> vec1 = spec.fromArray(a, i);
  FloatVector<S> vec2 = spec.fromArray(b, i);
  vec1.add(vec2).intoArray(c, i);
}
```

Vector API being developed as part of Project Panama:

- http://openjdk.java.net/projects/panama/ (vectorIntrinsics branch)

JEP 338: Vector API (Incubator)

- http://openjdk.java.net/jeps/338
- More details on Vector API at CodeOne
- Vector API for Java [DEV5081]
  - Tuesday, Oct 23, 1:30 p.m. - 2:15 p.m. | Moscone West - Room 2005

DIRECT I/O

All I/O buffered by kernel in DRAM (filesystem cache)

Direct I/O is a system-wide feature that supports direct reads/writes from/to storage device to/from user memory space bypassing system page cache.

Significant Reduction in Kernel time, Increase in User time and Disk Bandwidth

Direct I/O support available from JDK 10

- APIs are designed for easy use and minimal changes to applications
- https://bugs.openjdk.java.net/browse/JDK-8189192

RDMA

Java supports socket-based networking - OS Kernel sockets

Enable Remote Direct Memory Access(RDMA) capable network adapters to transfer data directly to/from application memory

Gives high-throughput, low-latency networking

Work-in-progress

- JEP 337: RDMA Network Sockets
- Java Bug System: https://bugs.openjdk.java.net/browse/JDK-8195160

Accelarating I/O

More details on O_DIRECT and RDMA at CodeOne

- Programming Modern Storage Devices with JDK10 [DEV5058]
  - Wednesday, Oct 24, 1:30 p.m. - 2:15 p.m. | Moscone West - Room 2011

Future Features: JDK 12

JDK 12 will be released on March 19, 2019. We started work on it in June 2018 (first changes were pushed).

New main features already “targeted” for JDK 12:

- One AArch64 Port, Not Two, JEP-340
- Default CDS Archives, JEP-341

Total number of Enhancements integrated into JDK 12 Hotspot JVM is around 210.

General code generation optimizations in Hotspot JVM:

- C2 should optimize trichotomy calculations, JDK-8210215
- Optimize integer divisible by power-of-2 check, JDK-8210152
- A lot of code refactoring for new ZGC and Shenandoah GC

Future Features: JDK 12 (cont)

X86 optimizations added to Hotspot JVM:

- Update avx512 implementation, JDK-8210764
- Default mask register for avx512 instructions, JDK-8211251

JDK 12 is only short-term-support release - it will reach its *end of life* (and *end* of public support) in September 2019.

Future Features

New main “Candidate” features we and OpenJDK Community are working on:

- Vector API (Incubator), JEP-338
- Limit Speculative Execution, JEP-342
- RDMA Network Sockets, JEP-337
- JVM Constants API, JEP-334
- Intrinsics for the LDC and INVOKEDYNAMIC Instructions, 303
- Pattern Matching, JEP-305
- Generics over Primitive Types, JEP-218
- Shenandoah: An Ultra-Low-Pause-Time Garbage Collector, JEP-189

Future Features (cont.)

Active OpenJDK Projects:

‘Skara’ - move SCM from Mercurial to Git:

https://github.com/Project-Skara/jdk/

‘Loom’ - implementation of lightweight user-mode threads (fibers)

‘Panama’ - interconnecting JVM and native code (improved JNI)

‘Value Types’ JEP-169 (part of ‘Valhalla’ project) - codes like a class, works like an int

‘Metropolise’ - “Java-on-Java”. Using Java based Graal compiler and SVM static compilation technology to replace HotSpot C2 compiler, and possibly other components of HotSpot.

Call For Action

Upgrade to latest JDK with newer platforms to get maximum benefits

Try new features and Contribute to Uses Cases

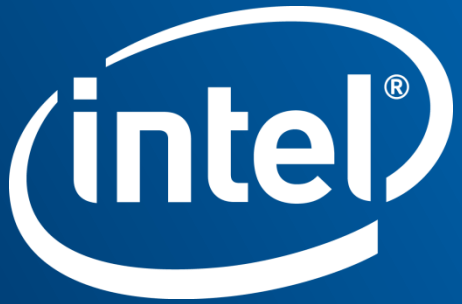
Contact us with the feedback.

Vladimir Kozlov, vladimir.kozlov@oracle.com

Vivek Deshpande, @vivdesh, vivek.r.deshpande@intel.com


Java™
ORACLE®


ORACLE®


intel®