Compact Strings: A Memory-Efficient Internal Representation for Strings

Charlie Hunt
Sandhya Viswanathan

Speakers

- Charlie Hunt, Oracle

  charlie.hunt@oracle.com
- Sandhya Viswanathan, Intel

  sandhya.viswanathan@intel.com
- You

  We encourage questions and discussions

Legal Disclaimers

- 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.
  A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.
- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
- The products described in this document 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.
- Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
- Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
- Intel, the Intel logo, Intel Xeon, and Xeon logos are trademarks of Intel Corporation in the U.S. and/or other countries.
- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to: Learn About Intel® Processor Numbers http://www.intel.com/products/processor_number

*Other names and brands may be claimed as the property of others.

Legal Disclaimers - Continued

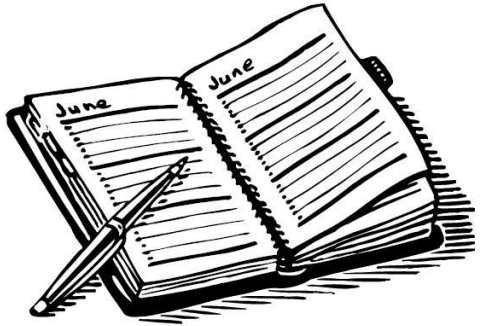
- Some results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
- Software and workloads used in performance tests may have been optimized for performance 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.
- Intel does not control or audit the design or implementation of third party benchmarks or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmarks are reported and confirm whether the referenced benchmarks are accurate and reflect performance of systems available for purchase.
- Relative performance is calculated by assigning a baseline value of 1.0 to one benchmark result, and then dividing the actual benchmark result for the baseline platform into each of the specific benchmark results of each of the other platforms, and assigning them a relative performance number that correlates with the performance improvements reported.
- SPEC, SPECint, SPECfp, SPECrate, SPECpower, SPECjbb, SPECompG, SPEC MPI, and SPECjEnterprise* are trademarks of the Standard Performance Evaluation Corporation. See http://www.spec.org for more information.
- TPC Benchmark, TPC-C, TPC-H, and TPC-E are trademarks of the Transaction Processing Council. See http://www.tpc.org for more information.
- Intel® Advanced Vector Extensions (Intel® AVX)* are designed to achieve higher throughput to certain integer and floating point operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you should consult your system manufacturer for more information.
- Intel® Advanced Vector Extensions refers to Intel® AVX, Intel® AVX2 or Intel® AVX-512. For more information on Intel® Turbo Boost Technology 2.0, visit http://www.intel.com/go/turbo

Safe Harbor Statement (Oracle)

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.

Agenda

- Introduction
- Motivation
- Design
- Performance



JEP 254: Compact Strings

- More space-efficient internal representation for Strings

- Author: Brent Christian
- Owner: Xueming Shen
- Reviewed by: Aleksey Shipilev, Brian Goetz, Charlie Hunt
- Endorsed by: Brian Goetz
- Release: 9
- Issue: 8054307

http://openjdk.java.net/jeps/254
https://bugs.openjdk.java.net/browse/JDK-8054307

Terminology

- Project name: String Density
- Feature name: Compact Strings

String Density Team

- Oracle
  - Charlie Hunt
  - Aleksey Shipilev
  - Sherman Shen
  - Brent Christian
  - Roger Riggs
  - Tobias Hartmann
  - Vladimir Kozlov
  - Guy Delemarter
- Intel
  - Sandhya Viswanathan
  - Vivek Deshpande

Project Goals

Requirements:

- Java supports Unicode (UTF-16) characters which uses 2 byte characters
- Improve the space efficiency of the String and related classes
- Preserve full compatibility for all related Java and native interfaces
- Maintain throughput performance in almost all scenarios
- Replacement for JDK 6's Compressed Strings
- Platforms: X86/X64, SPARC*, ARM* 32/64
- OS: Linux*, Solaris*, Windows* and Mac OS X*

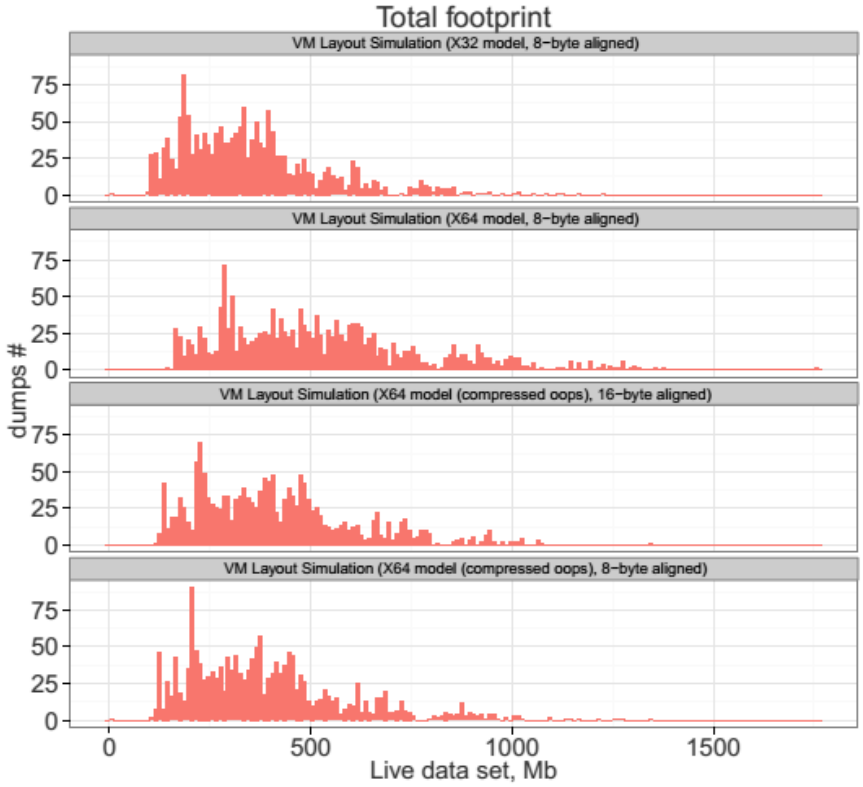
*Other names and brands may be claimed as the property of others.

Motivation

Experimental Setup

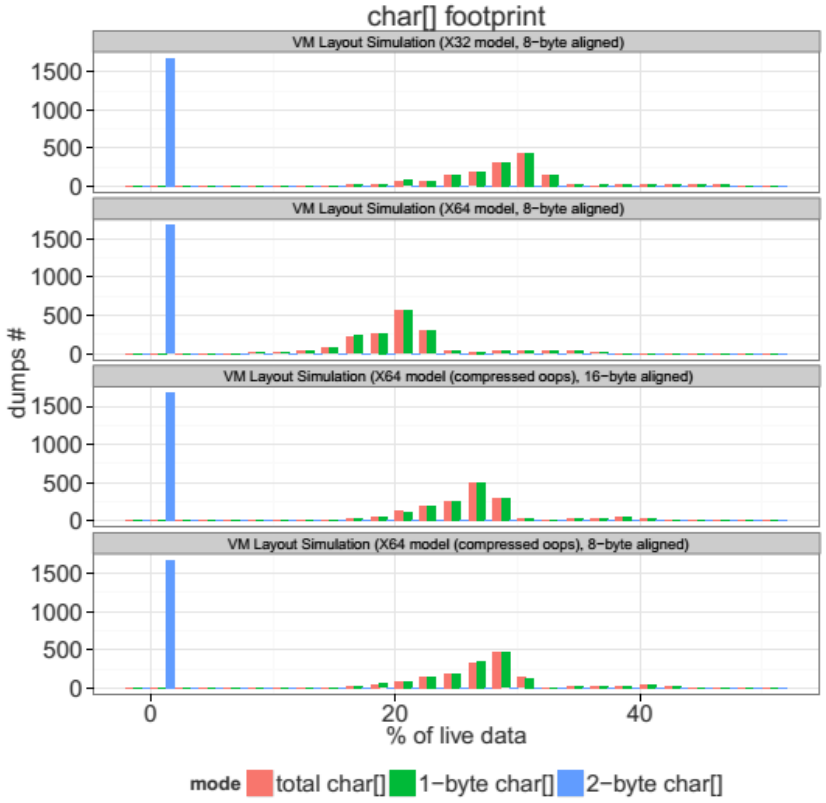
- Collection of 950+ heap dumps from
  - Oracle Fusion Middleware
  - Oracle Fusion Applications
  - Other Oracle Java Applications
- Java Object Layout Tools
- Linux X86_64 running on i7-4790K (1 socket, 4 cores, with HT)
- The following JVM modes emulated:
  - 32 bit data model
  - 64 bit data model, compressed reference disabled
  - 64 bit data model, compressed reference enabled
  - 64 bit data model, compressed reference enabled, 16 byte object alignment

Total Memory Footprint



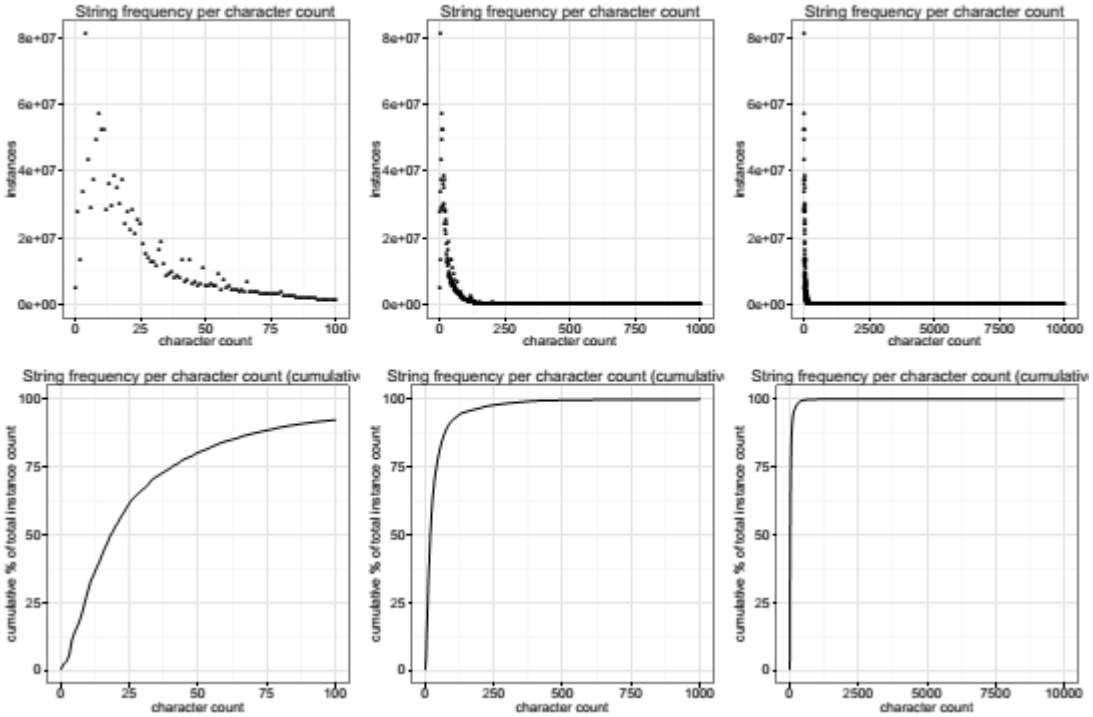
- 950+ Heap Dumps processed with Java Object Layout Tool
- Live Data Set size distribution shown
- Similar distribution in all models
- X64 dumps without compressed reference have larger footprints (2nd graph)

char[] Footprint



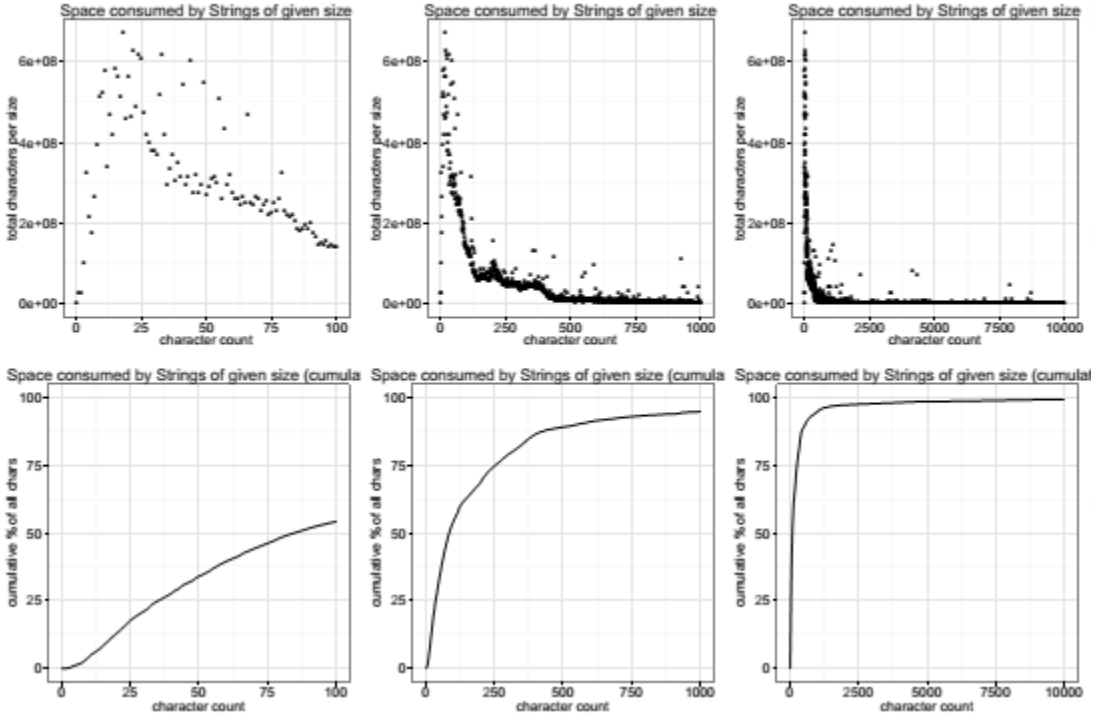
- char[] footprint as a % of LDS
- Consumes 10% to 45% of LDS
- X64 mode without compressed reference has lower % due to larger object header size
- Mostly 1 byte char[]

String Size Distribution (i)



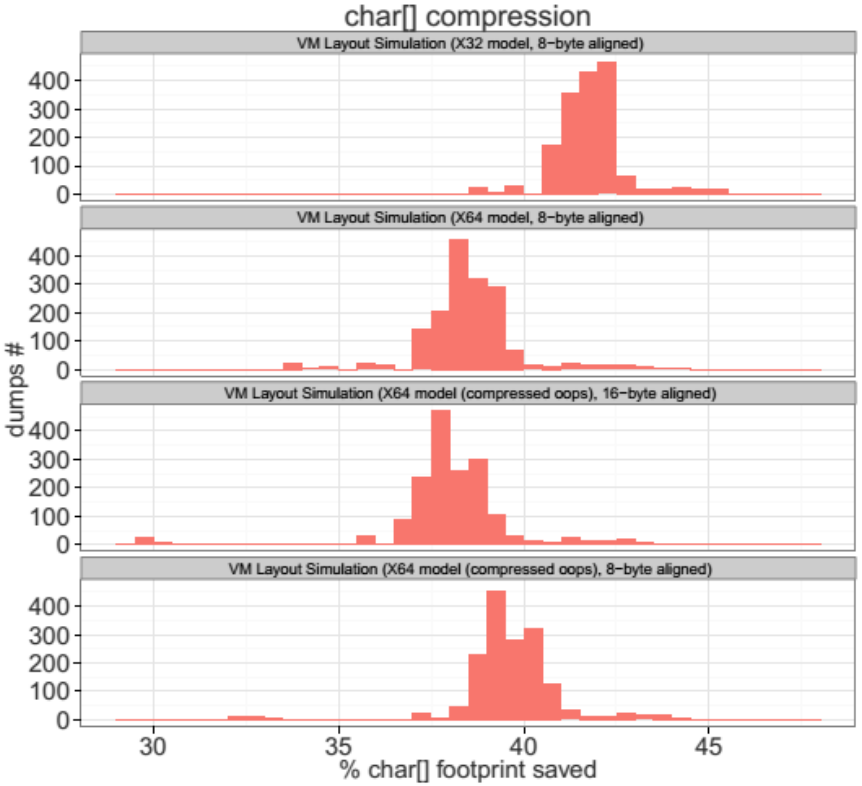
- Strings of a particular size (String instance count)
- Majority of Strings are small
- > 75% of strings are of size smaller than 35 characters

String Size Distribution (ii)



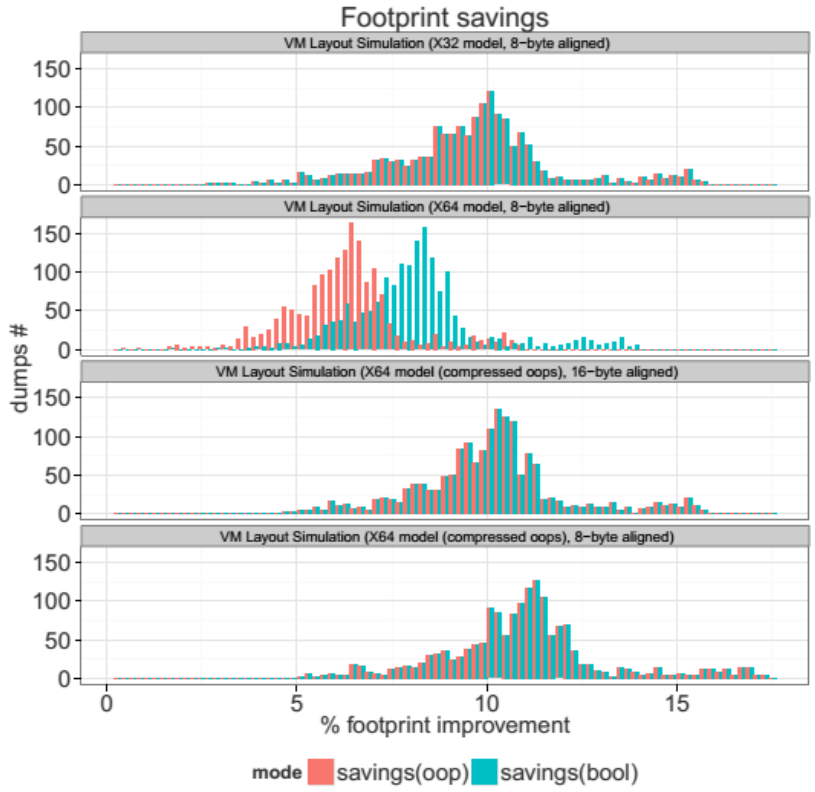
- Total character count for strings of particular size
- Skew the footprint towards the tail
- 75% of all the chars are residing in strings of size < 250

char[] Compression



- Projected char[] footprint improvements
- 35% to 40% reduction
- Less than 50% theoretical max due to dominance of small strings

Total Footprint Saving



- 5% - 15% reduction in total footprint due to:
  - Compressed underlying storage for characters in the string
  - Increased String object size

Design

New String Class Design

- Preserves full compatibility for all related Java and native interfaces
- Changes the internal representation of String class
- Characters of String encoded as either UTF-16 or ISO-8859-1/Latin-1
  - Stripped off the leading zero byte of a two byte UTF-16 character
- Uses byte array instead of char array to store characters
  - 1 byte per char for ISO-8859-1/Latin-1
  - 2 bytes per char for UTF-16
- An encoding byte field to indicate which encoding is used
  - Ability to extend to support additional character encoding(s)
- Reduces memory footprint
- Maintains throughput performance

String Encoding

- String with **any** leading byte in incoming chars as non 0
  - Cannot be compressed in our scheme
  - Stored as 2 byte chars using UTF-16 encoding
- Strings with leading byte in **all** the incoming chars as 0
  - Candidates for compression
  - Leading 0 bytes are stripped off and only the trailing byte is stored
  - This maps to the Latin-1 encoding
- Why not UTF-8?
  - UTF-8 supports variable width characters
  - Many String API implementations use random access into sequence of chars
  - Good encoding for transmission but not for performant String operations

String Class Old versus New

- Old String Class (JDK 8)

```
{
   private final char value[];
   private int hash;
   ...
}
```

- New String Class (JDK 9)

```
{
   private final byte[] value;
   private final byte coder;
   private int hash;
   ...
}
```

JDK 6 Compressed Strings

- JDK 6 String Class

```
{
    private final char value[];
    private final int offset;
    private final int count;
    private int hash;
    …
}
```

- JDK 6 Compressed String Class

```
{
    private final object value;
    private final int offset;
    private final int count;
    private int hash;
    …
}
```

Where value would point to char[] or byte[] based on String contents

JDK 6 vs JEP 254

- JDK 6 Compressed Strings
  - Two underlying implementations of String Class
  - Two sets of libraries: difficult to maintain
  - instanceof check, overhead for chars as byte[] or char[]
  - Limited support in the JRE
    - Resulted in frequent string inflation to UTF-16
- JEP 254
  - One underlying implementation of String Class using byte[]
  - Add an encoding (byte) field
  - Expanded support for compressed strings in JRE

String Class Layout

```
***** 32-bit VM: ************************************************************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0     8        (object header)                N/A
      8     4 char[] String.value                   N/A
     12     4    int String.hash                    N/A
Instance size: 16 bytes (estimated, the sample instance is not available)
Space losses: 0 bytes internal + 0 bytes external = 0 bytes total

***** 64-bit VM: ************************************************************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0    16        (object header)                N/A
     16     8 char[] String.value                   N/A
     24     4    int String.hash                    N/A
     28     4        (loss due to the next object alignment)
Instance size: 32 bytes (estimated, the sample instance is not available)
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```

String Class Layout Contd

```
***** 64-bit VM, compressed references enabled: ***************************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0    12        (object header)                N/A
     12     4 char[] String.value                   N/A
     16     4    int String.hash                    N/A
     20     4        (loss due to the next object alignment)
Instance size: 24 bytes (estimated, the sample instance is not available)
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total

***** 64-bit VM, compressed references enabled, 16-byte align: ************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0    12        (object header)                N/A
     12     4 char[] String.value                   N/A
     16     4    int String.hash                    N/A
     20    12        (loss due to the next object alignment)
Instance size: 32 bytes (estimated, the sample instance is not available)
Space losses: 0 bytes internal + 12 bytes external = 12 bytes total
```

- In 32 bit VM addition of encoding field inflates String object size
- No increase in String object size for 64 bit VM

Performance

JMH Micro-Benchmarks

- JMH based extensive throughput Micro-Benchmarks for
  - String, StringBuilder, StringBuffer APIs
  - String encoding and decoding operations
- Performance:
  http://cr.openjdk.java.net/~thartmann/compact_strings/microbenchmark/
- String methods are highly optimized using SIMD instructions where possible
- With CompactString ability to do twice the operation per iteration, e.g.:

Benchmark	BASE ns/op	SD ns/op	BASE/SD
compareto.CompareToBench.cmp1_cmp1	201.05	124.61	1.61
concat.ConcatCharBench.test_char1_cmp1	1072.21	581.13	1.85
encoding.From.ascii (ISO-8859-1)	2649.99	585.63	4.53
encoding.To.ascii (ISO-8859-1)	897.48	584.10	1.54
equals.EqualsBench.cmp1_cmp1	195.77	95.20	2.06
indexof.IndexOfChar.base1_img1__img1	906.86	752.44	1.21
indexof.IndexOfString.base1_img1__img1	1164.63	588.95	1.98

BASE: Base repo without Compact Strings

SD: With Compact Strings

Platform: Intel® Xeon ™ CPU E5-2697v3 @ 2.60GHz, 64 GB RAM with Linux 64-bit.

SPECjbb2005*

	BASE	SD	SD/BASE
Peak Throughput			1.05
Avg secs between GC (s)	3.898	4.937	1.27
Avg GC length (s)	0.018	0.019	1.06
Time Spent in GC (%)	0.459	0.384	0.84
Avg Resident Memory (K)	96139	75528	0.79

With CompactStrings

- Avg Resident Memory reduced by 21%
- Throughput increased by 5%

Comparing optimized JEP-254 String Density branch versus un-optimized JDK 9 Sandbox running on Intel® Core™ i7-4770 CPU @ 3.40GHz, 32 GB RAM with Linux 64-bit. JAVAOPTIONS="-server -showversion -Xmx12g -Xms12g -Xmn10g -XX:-UseAdaptiveSizePolicy -XX:MaxTenuringThreshold=15 -XX:InitialTenuringThreshold=15 -XX:+UseParallelOldGC -XX:ParallelGCThreads=4 -XX:+UseLargePages -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintFlagsFinal"

*SPECjbb2005 is trademark of the Standard Performance Evaluation Corporation.
See http://www.spec.org for more information.

SPECjbb2005 Heap Dump

	#instances		#bytes	
class	BASE	SD	BASE	SD
[C	1027729	1427	74592544	398096
java.lang.String	1069949	1069607	25678776	25670568
[Ljava.lang.String;	80486	80486	4510512	4510512
[B	1037	1027175	509536	49374184
...	...	...	...	...
Total	2677131	2676155	124671480	99312552

- Total # instances similar between the baseline & String Density.
- Total # bytes allocated is 21% less with String Density

*BASE: Base repo without Compact Strings

*SD: With Compact Strings

SPECjbb2015*

	BASE	SD	SD/BASE
max-jOPS			1.03
critical-jOPS			1.11
Avg secs between GC (s)	16.20	16.61	1.03
Avg GC length (s)	0.29	0.27	0.96
Time Spent in GC (%)	1.74	1.63	0.93
Avg Resident Memory (K)	1959521	1823907	0.93

With CompactStrings

- Avg Resident Memory reduced by 7%
- MultiJVM critical-jOPS increased by 11%

Comparing optimized JEP-254 String Density branch versus un-optimized JDK 9 Sandbox running on Intel® Core™ i7-4770 CPU @ 3.40GHz, 32 GB RAM with Linux 64-bit.

JAVAOPTIONS="-server -XX:+AlwaysPreTouch -XX:+UseParallelOldGC -XX:-UseAdaptiveSizePolicy -XX:MaxTenuringThreshold=15 -XX:-UseBiasedLocking -XX:+AggressiveOpts -XX:LargePageSizeInBytes=2m -XX:SurvivorRatio=28 -XX:TargetSurvivorRatio=95 -Xms19g -Xmx19g -Xmn17g -XX:+UseLargePages -XX:ParallelGCThreads=4 -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintTenuringDistribution"

*SPECjbb2015 is trademark of the Standard Performance Evaluation Corporation.

See http://www.spec.org for more information.

Call for Action

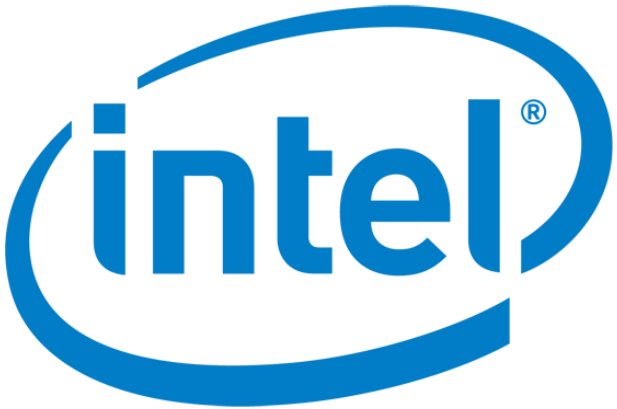
- Intrigued or Interested?
  Try CompactStrings feature on your workload and give us feedback

- Implementation:
  Repository: http://hg.openjdk.java.net/jdk9/sandbox/
  Branch: JDK-8054307-branch

- Steps:

```
$ hg clone http://hg.openjdk.java.net/jdk9/sandbox/
$ cd sandbox
$ sh ./get_source.sh
$ sh ./common/bin/hgforest.sh up -r JDK-8054307-branch
$ make configure
$ make images
```

- The option to enable/disable CompactStrings: -XX:+CompactStrings/-XX:-CompactStrings


intel