



[Home](#)
[Tutorials](#)
[Join Us](#)
[About](#)
[Resources](#)
[Job Board](#)
[Examples](#)
[Whitepapers](#)
[Academy](#)



Java Code Geeks

JAVA 2 JAVA DEVELOPERS RESOURCE CENTER




The world's best monitoring for Java, Node.js, and Mobile applications





[Free Monitoring >](#)

[Java](#)
[Android](#)
[JVM Languages](#)
[Software Development](#)
[Agile](#)
[DevOps](#)
[Communications](#)
[Career](#)
[Misc](#)
[Meta JCG](#)

You are here: [Home](#) [Core Java](#) Which memory is faster Heap or ByteBuffer or Direct ?




About Ashkrit Sharma
 Pragmatic software developer who loves practice that makes software development fun and likes to develop high performance & low latency system.

Which memory is faster Heap or ByteBuffer or Direct ?

by Ashkrit Sharma on August 17th, 2013 | Filed in: Core Java Tags: Direct Memory, Garbage Collection, Heap memory



Java is becoming new C/C++ , it is extensively used in developing High Performance System. Good for millions of Java developer like me!

In this blog i will share my experiment with different types of memory allocation that can be done in java and what type of benefit you get with that.

Memory Allocation In Java

What type of support Java provide for memory allocation:

- Heap Memory

I don't have to explain this, all java application starts with this. All object allocated using "new" keyword goes under Heap Memory

- Non Direct ByteBuffer

It is wrapper over byte array, just flavor of Heap Memory.
 ByteBuffer.allocate() can be used to create this type of object, very useful if you want to deal in terms of bytes not Object.

- Direct ByteBuffer

This is the real stuff that java added since JDK 1.4.
 Description of [Direct ByteBuffer](#) based on Java Doc

"A direct byte buffer may be created by invoking the allocateDirect factory method of this class. The buffers returned by this method typically have somewhat higher allocation and deallocation costs than non-direct buffers. The contents of direct buffers may reside outside of the normal garbage-collected heap, and so their impact upon the memory footprint of an application might not be obvious. It is therefore recommended that direct buffers be allocated primarily for large, long-lived buffers that are subject to the underlying system's native I/O operations. In general it is best to allocate direct buffers only when they yield a measureable gain in program performance."

Important thing to note about Direct Buffer is

- It is Outside of JVM
- Free from Garbage Collector reach.

These are very important things if you care about performance.



MemoryMapped file are also flavor of Direct byte buffer, i shared some of my finding with that in below blogs:

- [arraylist-using-memory-mapped-file](#)
- [power-of-java-memorymapped-file](#)

Off Heap or Direct Memory

This is almost same as Direct ByteBuffer but with little different, it can be allocated by unsafe.allocateMemory, as it is direct memory so it creates no GC overhead. Such type of memory must be manually released.

In theory Java programmer are not allowed to do such allocation and i think reason could be

Newsletter


69761 insiders are already updates and complimentary

Join them now to gain to the latest news in the Java insights about Android, Scala related technologies.

Email address:

[Sign up](#)

Join Us



With unique authentication the time around the clock enco

If you have a blog with unique content then you should check our partners program. You can for Java Code Geeks and f

Recent Jobs

[Software Developer](#)
 Reston, VA

- It is complex to manipulate such type of memory because you are only dealing with bytes not object
- C/C++ community will not like it

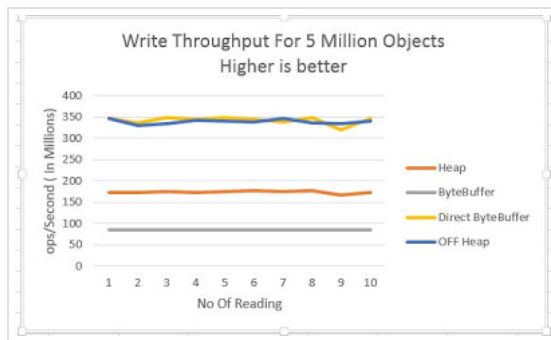
Lets take deep dive into memory allocation

For memory allocation test i will use 13 byte of message & it is broken down into

- int – 4 byte
- long – 8 byte
- byte – 1 byte

I will only test write/read performance, i am not testing memory consumption/allocation speed.

Write Performance



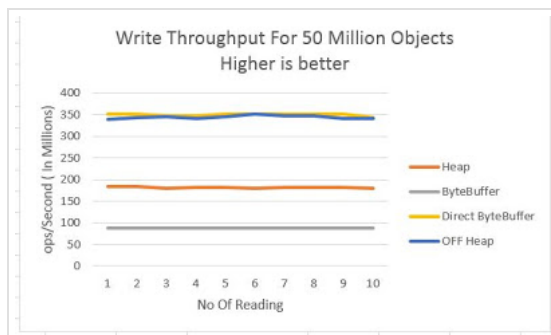
X Axis – No Of Reading

Y Axis – Op/Second in Millions

5 Million 13 bytes object are written using 4 types of allocation.

1. Direct ByteBuffer & Off Heap are best in this case, throughput is close to
2. 350 Million/Sec
3. Normal ByteBuffer is very slow, TP is just **85 Million/Sec**
4. Direct/Off Heap is around 1.5X times faster than heap

I did same test with 50 Million object to check how does it scale, below is graph for same.



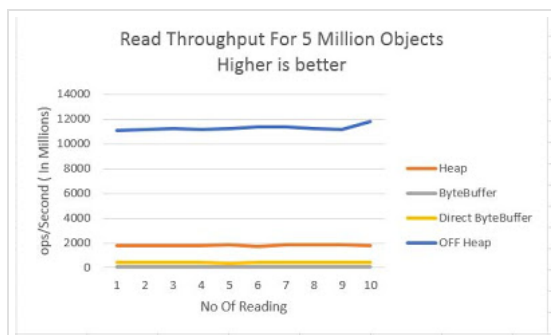
X Axis – No Of Reading

Y Axis – Op/Second in Millions

Numbers are almost same as 5 Million.

Read Performance

Lets look at read performance



X Axis – No Of Reading

Y Axis – Op/Second in Millions

This number is interesting, OFF heap is blazing fast throughput for **12,000 Millions/Sec**. Only close one is HEAP read which is around

Java Software Engineer
Mill Valley, CA

Junior to Mid-level Java Developer
Fairfax, VA

Software Developer – Project
Buffalo, NY

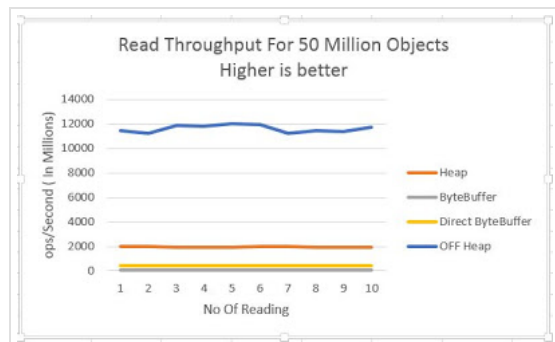
Java Developer
Atlanta, GA

[View All](#)

6X times slower than OFF Heap.

Look at Direct ByteBuffer , it is tanked at just **400 Million/Sec**, not sure why it is so.

Lets have look at number for 50 Million Object



X Axis – No Of Reading

Y Axis – Op/Second in Millions

Not much different.

Conclusion

Off heap via Unsafe is blazing fast with **330/11200 Million/Sec**.

Performance for all other types of allocation is either good for read or write, none of the allocation is good for both.

Special note about ByteBuffer, it is pathetic , i am sure you will not use this after seeing such number. DirectByteBuffer sucks in read speed, i am not sure why it is so slow.

So if memory read/write is becoming bottle neck in your system then definitely Off-heap is the way to go, remember it is highway, so drive with care.

Code is available @[git hub](#)

Reference: Which memory is faster Heap or ByteBuffer or Direct ? from our JCG partner Ashkrit Sharma at the [Are you ready](#) blog.

Do you want to know how to develop your skillset to become a **Java Rockstar**?

Subscribe to our newsletter to start Rocking right now!

To get you started we give you two of our best selling eBooks for **FREE!**

JPA Mini Book

Learn how to leverage the power of JPA in order to create robust and flexible Java applications. With this Mini Book, you will get introduced to JPA and smoothly transition to more advanced concepts.

JVM Troubleshooting Guide

The Java virtual machine is really the foundation of any Java EE platform. Learn how to master it with this advanced guide!

Email address:



16 Responses to "Which memory is faster Heap or ByteBuffer or Direct ?"

**Ashley**

February 8th, 2014 at 3:31 am

Ashkrit,

Thanks for this analysis.

ByteBuffer.allocateDirect(); also uses Unsafe by using DirectByteBuffer, so how is Unsafe.allocateMemory() different?

The results you have shown are really fascinating in favor of Off-Heap using Unsafe. Some of the questions to you here:

1. you never used setMemory() function. why? is it because Unsafe.putXXX() anyways were going to overwrite any pre-existing values?
2. when you perform operations inside the off-heap class, like identifyIndex(), etc, where are they getting executed? on-heap/off-heap? I ask this because, using Unsafe.putXXX() you use off-heap memory but all other operations like getDeclaredFields() of a particular class to get its data, and other operations like getting value of a particular field of an object before setting it off-heap, where does it operate? If on heap, then aren't we just copying data from JVM heap to non-heap?

Pls. let me know the above. Thanks

[Reply](#)**Ashkrit**

February 8th, 2014 at 4:55 am

1 – Regarding your question of why i didn't use setMemory.

This function is used to set some initial value to the memory that is allocated , if you don't do it then you will see some garbage, just like c/c++.

i didn't do it because it was ok for my test to ignore that step, but in real word we must set value to 0.

2 – Regarding your second question.

All the functions of Unsafe operate in JVM space, so there is cost of transfer because all function are native, but native function of Unsafe are special they are intrinsics, intrinsics functions doesn't have overhead like plain native/JNI call.

Intrinsic function are optimized , so you don't see big overhead of byte conversion for direct operation.

I think java Intrinsic deserve blog post.

[Reply](#)**Ashley**

February 8th, 2014 at 7:26 am

Ashkrit,

Thanks for your reply.

I understood the answer to the first question. I asked it because, I thought setMemory(offset, size, 0) is a standard practice in Unsafe to make sure the entire block is set to ZERO. Was suspecting that you didn't do it because it could cause some additional time. Do you think additional time Vs data corruption? what would you risk?

For the 2nd questions, I understand that Unsafe method calls are using intrinsic functions of java which are native like, Unsafe.allocate(), Unsafe.putXXX(), Unsafe.getXXX(). But, my question is different. I want to know that a function like identifyIndex() -> which is not Intrinsic to Unsafe -> where is it executed? heap OR off-heap?

Now since you said, that intrinsic functions are different that JNI calls and don't have overhead of JNI, how is it different from a c function using JNI in terms of execution and not efficiency of time/overhead?

Do you think intrinsic functions of Unsafe class can be utilized for doing something like getting handle to low-level registers for say, getting to a particular core of the processor? For such a thing, would you still need JNI? It would be great if you can use Unsafe like intrinsic function for such jobs.

Can I not just call some C code from Unsafe?

Thanks Ashkrit for this blog. Very informative.

[Reply](#)**Ashkrit**

February 8th, 2014 at 9:36 am

For 1st –

It is more data corruption than time. If data structure using off heap can manage the pointer in such a way that it never allow access to corrupted data or invalid/slate data then we can get rid of resetting.

2nd –

identifyIndex is executed off heap.

JVM does lot of smart thing for intrinsic, like it will do method inline, which i think doesn't not happen with native method, in many case intrinsic will try to use feature of underlying platform.

for eg – Integer.bitCount() is intrinsic, if you look in source code, it has java impl to find bit count but since it is intrinsic it will use POPCNT machine instruction to find bit count, which is very fast.

I am not sure if you have looked into

<http://hg.openjdk.java.net/jdk8/awt/hotspot/file/d61761bf3050/src/share/vm/classfile/vmSymbols.hpp>

This has list of all such function.

Unsafe is gate to get into C world, right now not every thing is exposed via unsafe.

It will be great if you can find more info about processor, like which core thread is running or pinning thread to specific core etc or access to fetch-add (i.e alternate to CAS).

There are talks to remove unsafe from hotspot, but this will setback for may high performance application, unless java decide to give such API as main API.

You might be interested in Unsafe survey.

<http://www.infoq.com/news/2014/02/Unsafe-Survey>

[Reply](#)



Ashley

February 8th, 2014 at 8:24 pm

Ashkritt,

Thanks again for this information and explanation. My question of off-heap usage is more related to code execution that necessarily uses heap like

TestMemoryAllocator.java should be using java heap while as you said identifyIndex() should still be off-heap because its related to Unsafe's positioning.

So, if I am designing a system for high speeds, what should I presume to be off-heap? Would it be all the code restricted to the usage Unsafe? If that's the case, then what's with the communication between objects on heap and data off-heap? Intrinsic functions are very useful. thanks for the link and I took the survey. I don't think Unsafe is going away from HotSpot because the buzz is that Oracle is trying to make in-roads into low-latency finance field and is getting a bit behind IBM JVM and a lot behind Azul. So, they are trying to come up with low-jitter Unsafe API that is actually Safe to use. One of their already implemented stuff is Native Byte Buffer but it sucks in reading. Any particular reason, you don't use ByteBuffer API functions in your tests?

I am very interested in low-level programming and trying to take that direction so I don't have to go into the oceanic world of programming with C for low-latency apps. Java is so much easier to maintain and faster to code as well and well-object oriented. I live in the US close to NYC and there are a lot of folks here who already use java for low-latency, however, concepts matter. As long as there is Unsafe and advanced bitwise operations not going away from JAVA, I anticipate a lot of companies changing to Java from C/C++.



Ashley

February 8th, 2014 at 3:36 am

Forgot to mention that DirectByteBuffer implementation and off-heap implementation looks identical. However, their read times are different most likely because while writing its both underlying implementations are using Unsafe but while reading, DirectByteBuffer is bringing data to heap. Any data brought from native memory to JVM will need to be converted into a byte array (its allocation on heap as an object takes time) that's why it seems to be taking so much more time, even more than the direct heap implementation. This is a thought, but I haven't verified it through tests.

[Reply](#)



Ashkritt

February 8th, 2014 at 4:59 am

With numbers it looked like DirectByteBuffer brings all data in Heap, but I didn't spend time to confirm that. I have to also do some test to confirm this fact.

[Reply](#)**Ashley**

February 8th, 2014 at 7:28 am

Ok sure some test data will be great but reading DirectByteBuffer to JVM heap is done via byte[] and that's when it takes a lot of time.

[Reply](#)**maciej**

March 7th, 2014 at 4:42 pm

I don't follow this explanation – the Off-heap implementation is calling:

```
@Override
```

```
public byte getByte() {
    return unsafe.getByte(pos + byte_offset);
}
```

the DirectMemoryBuffer implementation is calling:

```
public byte More ...get() {
    return ((unsafe.getByte(ix(nextGetIndex()))));
}
```

Where is the "bringing memory to heap space" happening in DMB vs off heap objects?

[Reply](#)**Ashley**

February 8th, 2014 at 4:08 am

Ashkrit,

One more question on why haven't you used ByteBuffer class functions?

[Reply](#)**Ashkrit**

February 9th, 2014 at 3:46 am

Reply option is disabled on the original thread, so using this one

I use below approach to solve the problem of building high speed application

Stay away from class like object model because it hurts performance due to memory indirection

– Keep data in memory using simple array like structure, best is column like approach which gives excellent performance because data is laid out linearly and you get benefit of hardware CPU cache, Prefetcher etc.

In this approach you can get reasonable performance but GC comes in picture and you have to deal with it.

– If you want to Keep GC out then I start looking at unsafe for direct memory allocation.

It is possible to do better on what type of API you provide to access/manipulate off heap data, internally I think has to be streams of byte but to outside world nice Object view.

Once you have some objects off heap then the issue of how do on-heap refer off heap, I have only used pointer (i.e reference by index or section of memory) based approach to achieve such thing.

Some other things that comes to mind is creating in-memory index/dictionary of off heap object, so that you don't deal with some numbers and you can still get them in reasonably time.

In-Memory index is being heavily used in Mongo DB, which is using Memory Mapped file to keep all the data, but it keeps index in memory and it contains reference to byte indexes.

I didn't get your question about ByteBuffer class, which function are you taking about?

Lot of interesting stuff happens in US!

I am based on Singapore and have very few options of where people are really using Java for high performance application.

I think application performance is decided by data structure, algo & design you make not by just language.

So there is a lot of myth around Java is slow and hopefully that will go.

[Reply](#)

**Ashley**

February 10th, 2014 at 8:59 pm

Ashkrit,

Thanks for your reply again. I certainly appreciate it.

ByteBuffer functions like position(), limit(), put()/get(), etc. which are defined in the API. I think you haven't used them because it would be done on JAVA heap, esp. for writing. So, you got similar to Unsafe results for writing. However, I think because reads still bring the data to JAVA heap, the results are poor. I just wanted to check with you, if it's true.

I am going to read upon your memory-mapped article files later. Thanks for great postings.

Yes, there is a lot of great opportunity here, but I believe, London is ahead on low-latency related development than NYC.

I agree that nice class view is seen by the outside world, however, internally, it's using fast Off-Heap execution.

One of the areas that I am a bit nervous about is thread safety with off-heap data manipulation. How do you port large data structures off-heap? and make them threadsafe.

Thanks again for your article.

[Reply](#)**Maciej**

March 11th, 2014 at 8:12 pm

I have one further question based on the article presented here:

<http://mentablog.solveirajr.com/2012/11/which-one-is-faster-java-heap-or-native-memory/>

It seems to me you are using the same technique to access Off-heap memory, and yet you experience much better results, clearly unaffected by the JNI calls. Could you explain why that might be the case?

[Reply](#)**Ashkrit**

March 12th, 2014 at 10:09 am

My this article was as part of curiosity from the article you mention above, i have commented on it with my result before writing this blog!

Difference is that i am allocating one big array for all the objects and the blog that you mention allocate memory per object

Allocating one big array has lot of nice benefit

- Predictable memory access
- Most of the data will be prefetched

This type of allocation will be more cpu cache friendly.

[Reply](#)**Dennis**

July 14th, 2014 at 1:04 pm

It seems that your implementation of HeapAllocatedObject is not apple-to-apple comparison, as you added extra layer HeapValue and caused additional index-lookup and memory fragmentation.

As the Object construct and GC are only happen in HeapAllocatedObject, but not others. it supposed to be slowest.

[Reply](#)**Ashkrit**

July 15th, 2014 at 6:20 pm

One of the reason to write this blog is to share the overhead you have with plain java object.

Heapvalue will have all the overhead that is associated with any object due to layout used by java, all the heap allocation will have GC overhead also and direct memory is free from it and that is one of the big reason of write speed you get with direct memory.

ByteBuffer shows interesting result, it has worst write performance although it is just backed by bytearray and most compact way to store data on heap.

Access by index is not adding any significant overhead.

[Reply](#)

Leave a Reply

Name (Required)

Mail (will not be published) (Required)

Website

× 2 = sixteen

☐

Notify me of followup comments via e-mail. You can also [subscribe](#) without commenting.

☒

Sign me up for the newsletter!

Submit Comment

Knowledge Base	Partners	Hall Of Fame	About Java Code Geeks
Academy	Mkyong	“Android Full Application Tutorial” series	JCGs (Java Code Geeks) is an independent online community creating the ultimate Java to Java developers resource. We have a technical architect, technical team lead (senior developer), and junior developers alike. JCGs serve the Java, SOA, Agile communities with daily news written by domain experts, reviews, announcements, code snippets and open source projects.
Examples	The Code Geeks Network <div>Java Code Geeks.NET Code GeeksWeb Code Geeks</div>	GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial	
Resources		Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons	
Tutorials		Android Google Maps Tutorial	
Whitepapers		Android Location Based Services Application – GPS location	
		11 Online Learning websites that you should check out	
		Java Best Practices – Vector vs ArrayList vs HashSet	
		Android JSON Parsing with Gson Tutorial	
		Android Quick Preferences Tutorial	
		Difference between Comparator and Comparable in Java	