International Journal of Scientific Engineering and Technology ISSN:2277-1581
Volume No.5 Issue No.8, pp: 400-405 01 August 2016

Study of Compilation and Heap Tuning Parameters across Virtual Machines for the Application Performance Improvements

Vijayanand Kumar[1], Ankit Babbar[2], Vikas Kala[3], Mudit Khare[4], Amitesh Anand[5]
Department of Computer Sc & Engineering
[3] IIIT, Gwalior, India
[2] Thapar University, Patiala, India
[1] Delhi Technological University, Delhi, India
[4] Bundelkhand Institute of Engineering & Technology, Jhansi, India
[5] Calcutta Institute of Engineering and Management, Kolkata, India
1vijayanand033@gmail.com , 2ankitbabbar89@gmail.com, 3vikaskala189@gmail.com, 4khare.mudit@gmail.com ,5amitesh.unique@gmail.com

Abstract : Virtual Machines are the programs for supporting platform independence, network mobility and scalability for different underlying machine architectures such as x86 and ARM etc for JAVA based framework. VM main functions are Compilation of bytecode to the native code, Memory Management and Garbage Collector Management etc. Java architecture is a stack based architecture where as Dalvik and ART possesses register based architecture. All mentioned Virtual Machines are run on battery driven devices such as laptop and mobile. So the goal of VM must be to improve performance by consuming less battery power and faster execution of programs. It is also important to maximize the throughput of the CPU cycles. CPU cycles are one more major factor to consider which impact the battery life. Many enhancements have been observed in VM evolution from JAVA VM to ART VM.JAVA Code compilation has been taken care by interpreter and just in time compilation technique which convert per java class to native code where as dalvik bytecode per application forms Dex executable and further conversion of Dex to native by using JIT compilation in runtime. ART uses new Ahead of Time compilation technique which creates executable and linkable format. ART compilation done during boot or upgrade time and overcome the runtime compilation.
So far ART Virtual Machine has been considered as best VM in terms of performance and better user experience for android system under command of linux operating system. Tradeoffs took place for performance over higher installation time for application in case of ART (Android Runtime) [14, 15, 16, 17].
Virtual Machines also evolved and experimented with different memory management techniques for fast allocation and aim to achieve parallelism and concurrency. Allocation technique used for JAVA and Dalvik VM is through dlmalloc library whereas rosalloc technique used for Android Runtime to boost performance in allocation [2, 5].Garbage collection runs as daemon in each VM for reclaiming the memory and causes application performance to go low. Several GC techniques have been experimented on VM's among which Concurrent Mark and Sweep and Generational are most famous one. Goal of GC must be lesser pause time, GC time and to overcome UI frame drops [1, 13]. This paper presents different compilation and heap tuning parameters for different virtual machine. Experimentation results discuss about the trade-off with tuning.

Keywords : Virtual Machine, Android, Dalvik, Android RunTime, Garbage Collection.

NOMENCLATURE:

GC	Garbage Collector
VM	Virtual Machine
UI	User Interface
ART	Android Runtime
Dex	Dalvik Executable
OAT	Optimized art
ODEX	Optimized Dalvik Executable
JIT	Just in Time
AOT	Ahead of Time
CGC	Concurrent GC
AGC	GC for Alloc
CSB	Concurrent Start Bytes
S.L.	Soft Limit
TU	Target Utilization
HSS	Heap Start Size
HMF	Heap Max Free

1. Introduction

Java, Dalvik and ART offer several tuning parameters to optimize the performance in terms of compilation and heap. These tuning varies with the device configuration such as RAM size and density pixel used [2]. For different mobile devices tuning parameter varies and scope for optimum tuning exists.

(a) Compilation Tuning:-

Interpreter, JIT, AOT and mix of compilation techniques with profiling of methods and bytecodes are available for different virtual machines [14-17].
For Java Applications, compilation tuning parameters relies on profiling the hot methods and taken care by the interpreter and JIT compiler itself [18, 19]. Bytecode optimization using _quick instruction improves the performance by using caching technique [8, 9]. Similar case is with the dalvik which uses JIT framework for compilation in runtime. Dalvik executable with optimization (dexopt) takes place before converting bytecode to

the native code. Only tuning associated with it is the number of methods or code needs the compiler during profiling at runtime. For dalvik and android runtime, bytecode representations are register based which is more approximated to machine architecture (x86 and ARM) and hence faster execution of bytecode takes place [3]. Compilation tuning for ART has scope to improve performance [21]. Dex2oat is the tool which converts dex bytecode to native code and there are associated tuning parameters including profiling for methods [22, 23].

ART Compilation Tuning:-

dalvik.vm.profiler [0/1]
dalvik.vm.dex2oat-Xms : [64m]
dalvik.vm.dex2oat-Xmx : [256m]
dalvik.vm.image-dex2oat-Xms:[64m]
dalvik.vm.image-dex2oat-Xmx:[64m]
dalvik.vm.dex2oat - filter
dalvik.vm.image -dex2oat - filter

Filter Options:

- everything - compiles almost everything, excluding class initializers and some rare methods that are too large to be represented by the compiler's internal representation.
- speed - compiles most methods and maximizes runtime performance, which is the default option.
- balanced - attempts to get the best performance return on compilation investment.
- space - compiles a limited number of methods, prioritizing storage space.
- interpret-only - skips all compilation and relies on the interpreter to run code.
- verify-none - special option that skips verification and compilation, should be used only for trusted system code.

Threshold Limits on the methods:-

- kDefaultHugeMethodThreshold
- kDefaultLargeMethodThreshold
- kDefaultSmallMethodThreshold
- kDefaultTinyMethodThreshold
- kDefaultNumDexMethodsThreshold
- kDefaultTopKProfileThreshold

ART uses AOT compilation and above are important parameters to control the compilation and performance. Most of the code can be compiled using "everything" filter option with more memory footprint with trade-off with runtime performance improvements.

Heap Tuning:-

Performance impact for the application can be observed with the heap tuning parameters. This tuning mainly impacts the GC work functionality. Running the GC daemon for memory reclaim for the applications is bounded by the memory tuning parameters.

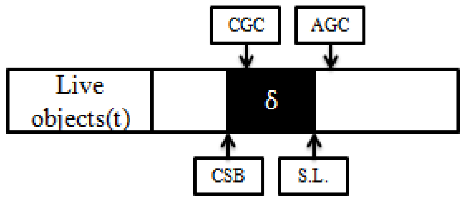

Figure 1: Concurrent GC and Allocation GC on heap

Running concurrent GC depends when concurrent start byte gets hit and allocation GC is being triggered when soft limit footprint crossover while object allocation [1]. Figure 1 is the theory behind dalvik and ART garbage collection with heap configuration for each application. One parent zygote process forks several children as application with the copy of same heap tuning configurations. Heap tuning changes system wide and overall performance impact can be observed. For the Java serial and parallel GC runs automatically when the soft limit threshold reached on allocation for each application. Figure 1 also represents the heap with live objects present and adjusted CSB and S.L corresponding to it. Concurrent start bytes and soft limit is likely to change on each allocation [1] and shows the piston behaviour which moves to and fro on allocation and collection.

For Java VM three heap parameters are essential for the tuning [25].

Parameters	**Description**	**JVM Option(Kaffe-VM)**	**Default**
Initial Heap Size	VM requests this amount of memory from operating system and add it to heap it manages	-ms	5MB
Maximum Heap Size	VM never requests more than this amount of memory	-mx	64MB
Heap Increment Size	Memory will be allocated when VM decides to postpone GC and increase heap by this amount of memory.	-as	1MB

Table 1: JVM Heap Tuning Parameters

Parameters	**Limits**
dalvik.vm.heapstartsize	5MB-16MB
dalvik.vm.heapgrowthlimit	64MB -96MB
dalvik.vm.heapsize	96MB -128MB
dalvik.vm.heaptargetutilization	0.75-.90(%)
dalvik.vm.heapminfree	512KB-2MB
dalvik.vm.heapmaxfree	8MB-16MB

Table 2: Android Dalvik/ART VM Heap Tuning Parameters

Table 1 and 2 describes some of the tuning parameters associated to the heap. Garbage collector runs as GC daemon for concurrent GC and normal collection while allocation occurs for other types of GC.
Section II explains about the relative work based on experiments conducted by hardware manufacturer. Section III describes the proposed methodology of tuning for optimum performance. Section IV deals with simulation environment and results and the conclusion has been made in Section V.

2. Related Work

Java performance for different virtual machine depends on benchmarking tool such as dacapo, jprofiler and jRockit etc. In the case of finding the best VM, machine consistency must be maintain and tuning must be set as default or described by the VM vendor. Different machines with different architecture or memory footprint have different tunings. These tuning are variables and left to the vendor for further optimization which uses VM. For the Dalvik and ART case combination of compilation tuning and heap tuning can give optimum performance in terms of better battery life and CPU usage. Techniques such as compilation of mixed AOTC and JITC [14-17] can optimize performance. Garbage collections such as including the generational collection [1] also help improving performance. ART has such configuration which uses AOTC and JITC framework with generational collection. ART also achieve parallelism by using new memory management techniques. Evolution of virtual machine is resulting in better performance by changing the VM framework. VM Vendors provide options for tuning and VM are in the generic form which later on configured by device vendor which uses the VM.

dalvik.vm.parameters	**Xhdpi 1GB**	**Xhdpi 2GB**	**Mdpi 1GB**
heapstartsize	8MB	16MB	8MB
heapgrowthlimit	128MB	200MB	64MB
heapsize	174MB	348MB	174MB
heaptargetutilization	0.75	0.75	0.75
heapminfree	512KB	512KB	512KB
heapmaxfree	8MB	8MB	8MB

Table3: Experimental Tuning by Vendor (Intel)

Table 3 is the experimental result [2] from the Intel which varies for different devices which are variable in density pixel and RAM footprint. Claim here is that these tuning are optimum with such specification mentioned in table 3.This conclude that there is scope for further tuning for different upcoming specifications with current tuning parameters. Further modification scope is present to find optimum tuning with current specification. Proposed work and experimentation results are discussed with respect to finding optimum tuning parameters for specific device configuration.

3. Proposed Work

Programming language such as C,C++,JAVA or supported framework using these language such as C based platform for mobile development or Java based android development compilation optimization is done based on the coding construct defined. In other word making code to compile and converting code to the native or machine understandable also affected by coding done for the application. Best coding guidelines supports optimum compilation and fast execution [26-28]. Virtual machine interpretations for compilation is either stack based [C, C++, Java] or register based [Android Dalvik or ART]. For the Android system google provide tuning [29] for low ram and enabling or disabling the JIT compilation. Vendors such as google a provide specification and support list of features tuned for the device configuration [2, 3].

Tuning Area for Performance	**Parameters**
Density Pixels	Ram Requirements
Garbage Collection	Pause Time GC Time GC Count Type of GC run on heap space(ART)
OS support	32 bit or 64 bit support
Kernel Memory Optimization	Swap space size Swappiness
Low Memory Killer OOM Adjustment Low Ram Flag	RAM based configuration

Table 4: Device Tuning Parameters

Apart from compilation and heap tuning, table 4 describes other tuning for the device for performance optimization [2].Till now mentioned parameters are configurable with respect to device configuration and scope for further tuning for specific device. For finding optimum tuning tools such as benchmarking for JAVA, Android specific virtual machines, GC benchmarking for performance are available [2]. For android case google code base can be tested for different applications with change in tuning parameters with specific device configuration. Tools provided by google for android such as sys trace analysis, heap dump profiler and cpu analysis tools in built with eclipse can support the analysis for the performance for particular application as well as for all the applications. Logs are enabled for virtual machine to analyze the garbage collector which can help finding GC time, GC Pause time and GC Counts. Compilation logs can be enabling to analyze the initialization of the class, time taken for verification of the class and loading of the class. Tuning does have system wide impacts. Automation, Manual and monkey tests are also useful in analyzing system wide impacts. Experimentation performed uses all the methodology for analyzing system performance. Implementation has been to parse the GC logs for the system.

GCType	Count	Pause Time	Total Time
Explicit concurrent mark sweep	1008	2533.795	108293.5
Background concurrent mark sweep	5	7.78	497.617
Explicit marksweep + semispace	146	6324.298	6410.48
Background sticky concurrent mark sweep	271	2829.847	25699.85
Background partial concurrent mark sweep	184	1185.853	25307.44
Explicit	1154	8858.093	114704
Background	460	4023.48	51504.91
Total(Explicit+BG)	1614	12881.573	166208.9

Table5: ART GC Types versus Time

GCType	Freed Obj Count	Freed Obj Size
Explicit concurrent mark sweep	11459443	782.400MB
Background concurrent mark sweep	3930	165KB
Explicit marksweep + semispace	2100211	126.0MB
Background sticky concurrent mark sweep	4918594	254.232MB
Background partial concurrent mark sweep	3290614	172.0MB
Explicit	13559654	909.400MB
Background	8213138	427.232MB
Total(Explicit+BG)	21772792	1336.632MB

Table6: ART GC Types versus Objects

GCType	Freed L Obj Count	Freed L Obj Size
Explicit concurrent mark sweep	1704	243.0MB
Background concurrent mark sweep	0	0B
Explicit marksweep + semispace	33	3.0MB
Background sticky concurrent mark sweep	2473	80.0MB
Background partial concurrent mark sweep	2316	339.0MB
Explicit	1737	246.0MB
Background	4789	419.0MB
Total(Explicit+BG)	6526	666.0MB

Table7: ART GC Types versus Large Objects

Table 5, 6 and 7 are the parsed log for android runtime garbage collection for analysis. Comparison reports can be prepared based on log analysis for particular test suite on android. Table 5, 6 and 7 prepared after tuning the heap parameter for comparison analysis with the default heap tuning.

4. Simulation Results

Experimentation has been done on Java virtual machine (Kaffe-VM), Dalvik VM and ART VM.

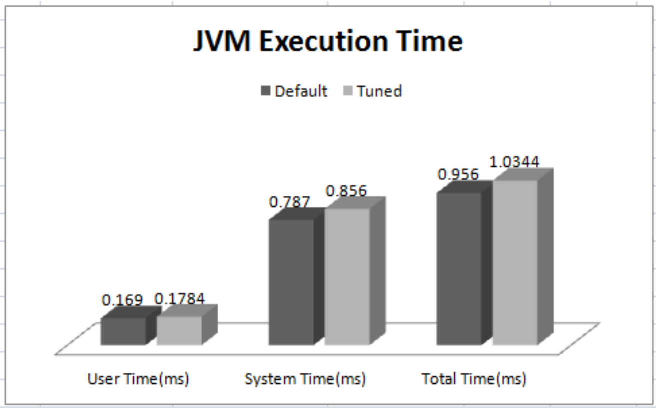

Figure 2: Execution Time Kaffe-VM

Configuration	Default: Initial Heap Size=5M, Maxheap Size=64M, Increment Size=1M Tuned: Initial Heap Size=8M, Maxheap Size=12M, Increment Size=3M
Inference	Tuning the parameters, the performance of the app is similar but consumes less memory resources, as max heap memory has been reduced; but as a trade-off initial & incremental size of heap is increased.

Table 8: Java VM Heap Tuning

Figure2 and table 8 describe the results and impact for specific application with average 12M ram usage.

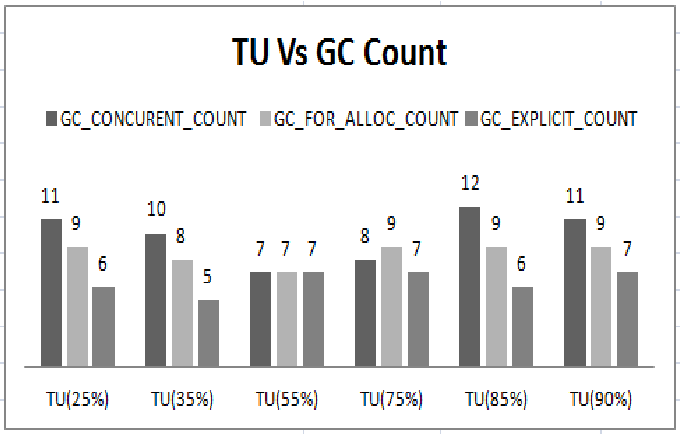

Figure 3: Dalvik VM heap tuning for target utilization

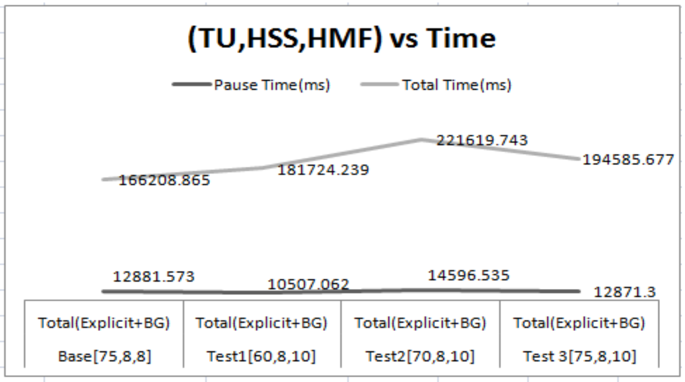

Figure 4: ART VM heap tuning versus time

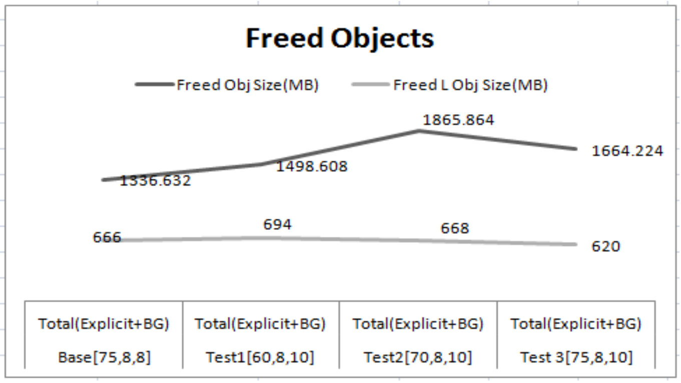

Figure 5: ART VM heap tuning versus Objects

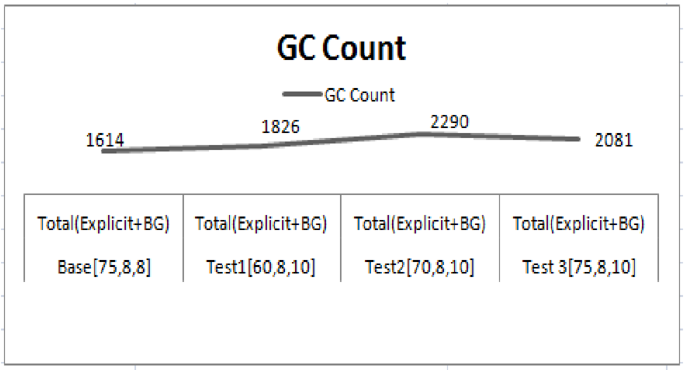

Figure 6: ART VM heap tuning versus GC counts

Figure 3, 4, 5 and 6 demonstrate the heap tuning results for specific test suit for 1GB RAM configuration.

Figure	Inferences
3	Heap target utilization varied and compared to default (75%). GC Count has been reduced.
4	Optimum GC time for [78,8,8] (166 sec) and optimum paused time at [60,8,10] (105 sec)
5	Less freed object for [78,8,8] , highest at [70,8,10]
6	Less GC count at [75,8,8]

Table 9: Android VM Analysis

Table 9 suggests that heap tuning with target utilization of 75%, heap start size with 10MB and heap max free with 8MB provides optimum GC performance for ART with configuration for Xhdpi with 1GB RAM.

For runtime improvements setting dex2oat filter in device.mk with product property configuration for 1GB RAM(Xhdpi) native application ROM footprint increased by 4MB and overall more than >15MB in size with native and downloaded applications i.e. code has been optimised with below configuration and provide >200ms average application entry time improvement(140 application's oat were present in the device)

```
PRODUCT_PROPERTY_OVERRIDES += \
    dalvik.vm.dex2oat-filter=everything \
    dalvik.vm.image-dex2oat-filter=everything
```

5. Conclusion

Goal was to find optimum tuning for different devices with different configuration so that performance boost can be realized. Introduction and related work section provide the study related to tuning parameters which is configurable for different devices for different configurations. There is vast scope for improvement through tuning parameters provided by different virtual machine vendors. These tuning will change with the change in configuration for future devices. Compilation and Garbage collector for android systems are major area for improvements. Memory management such as ROSALLOC over DLMALLOC techniques has been used for fast allocation in case of android runtime. ART provided pre-compilation technique which provide runtime performance boost over higher install time and increased ROM footprint where as DALVIK relies on interpreter and JIT compilation techniques with caching the hot methods of code for performance boost.

References

i.Ahmed Hussein, Mathias Payer et. al. Impact of GC Design on Power and Performance for Android.
ii. https://nebelwelt.net/publications/files/15SYSTOR.pdf
iii.Android Memory Tuning for Android 5.0 and 5.1. https://01.org/android-ia/user-guides/android-memory-tuning-android-5.0-and-5.1
iv.http://source.android.com/compatibility/android-cdd.pdf . Google CDD for Android 5.1
v.http://developer.android.com/tools/debugging/debugging-memory.html. Investigating Your RAM Usage
vi.http://dubroy.com/memory_management_for_android_apps.pdf and associated video at
vii. http://dubroy.com/blog/google-io-memory-management-for-android-apps/
viii.https://source.android.com/devices/tech/ram/low-ram.html Google's discussion of low-ram configuration.
ix.http://source.android.com/devices/tech/dalvik/configure.html Google's discussion on configuring ART.
x.Inside the Java Virtual Machine by Bill Venners.
xi.https://conference.hitb.org/hitbsecconf2014ams/materials/D1T2-State-of-the-Art-Exploring-the-New-Android-KitKat-Runtime.pdf.
xii.https://anatomyofandroid.com/.
xiii.http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/
xiv.http://NewAndroidBook.com/

xv. *A Generational Mostly Concurrent Garbage Collection. Tony Printezis, David Detlefs.*

xvi. *On Real-Time Performance of Ahead-of-Time Compiled Java. Anders Nilsson and Sven*

a. *Gestegard Robertz*

xvii. *Design and Optimization of a Java Ahead-of-Time Compiler for Embedded Systems, Dong-Heon Jung, Soo-Mook Moon, Sung-Hwan Bae.*

xviii. *A Method-Based Ahead-of-Time Compiler for Android Applications. Chih-Sheng Wang, Guillermo A. Perez et. al.*

xix. *A Selective Ahead-Of-Time Compiler on Android Device. Yeong-Kyu Lim, Sharfudheen Parambil,See-Hyung Lee et.al.*

xx. *http://www.javaworld.com/article/2078635/enterprise-middleware/jvm-performance-optimization-part-2-compilers.html*

xxi. *http://www.ibm.com/support/knowledgecenter/SSAW57_7.0.0/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tunejvm_v61.html*

xxii. *http://www.ibm.com/support/knowledgecenter/SSAW57_7.0.0/com.ibm.websphere.nd.doc/info/ae/ae/tprf_hotspot_jvm.html*

xxiii. *https://source.android.com/devices/tech/dalvik/configure.html*

xxiv. *http://newandroidbook.com/files/Andevcon-ART.pdf*

xxv. *https://conference.hitb.org/hitbsecconf2014ams/materials/D1T2-State-of-the-Art-Exploring-the-New-Android-KitKat-Runtime.pdf*

xxvi. *http://www.kaffe.org/*

xxvii. *http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html*

xxviii. *https://software.intel.com/en-us/articles/how-to-optimize-java-code-in-android-marshmallow*

xxix. *http://www.vogella.com/tutorials/AndroidApplicationOptimization/article.html*

xxx. *http://developer.android.com/training/articles/perf-tips.html*

xxxi. *https://source.android.com/devices/tech/config/low-ram.html*