International Journal of Allied Practice, Research and Review
Website: www.ijaprr.com (ISSN 2350-1294)

The Hotspot Java Virtual Machine: Memory and Architecture

Prof. Tejinder Singh
Assistant Professor, Deputy Dean IRP of Baba Farid College, Bathinda (Punjab)
E-Mail: tejinder31.singh@gmail.com

Abstract - In this paper is providing information about JVM Architecture and memory etc. The new compiler named GALADRIEL starts from Java class filesproduced from the initial Java specification and processes the system information in order to exploit the concurrency implicit in each method. This paper describes our experience in porting Compaq's Fast VM from the Alpha processor architecture to the Intel x86 processor architecture. Although the JVM is a great target for the Java TM programming language, it is not necessarily a good platform for other languages. Java for its portable byte codes and extensive libraries, but prefer a different language, especially for scripting.
Keyword: Java; JVM; Memory; Architecture.

I. Introduction

The Java™ programming language is a usual purpose, synchronized, objects oriented Programming language. The JVM is the foundation of the Java platform. There is the element of the technology accountable for its hardware- and operating system independence, and its tiny size of the compiled code, and its facility to defend users from mischievous programs. The first prototype implementation of the JVM (Java Virtual Machine), completed by Sun Microsystems, Inc., matched the Java virtual machine coaching usual in software presented by a handheld device that be similar to a existing Personal Digital Assistant (PDA). Oracle's current enactments emulate the Java virtual machine on handheld, PC and Super computers devices, but the Java virtual machine is not accept any specific execution technology, host hardware, or host operating system. [1].

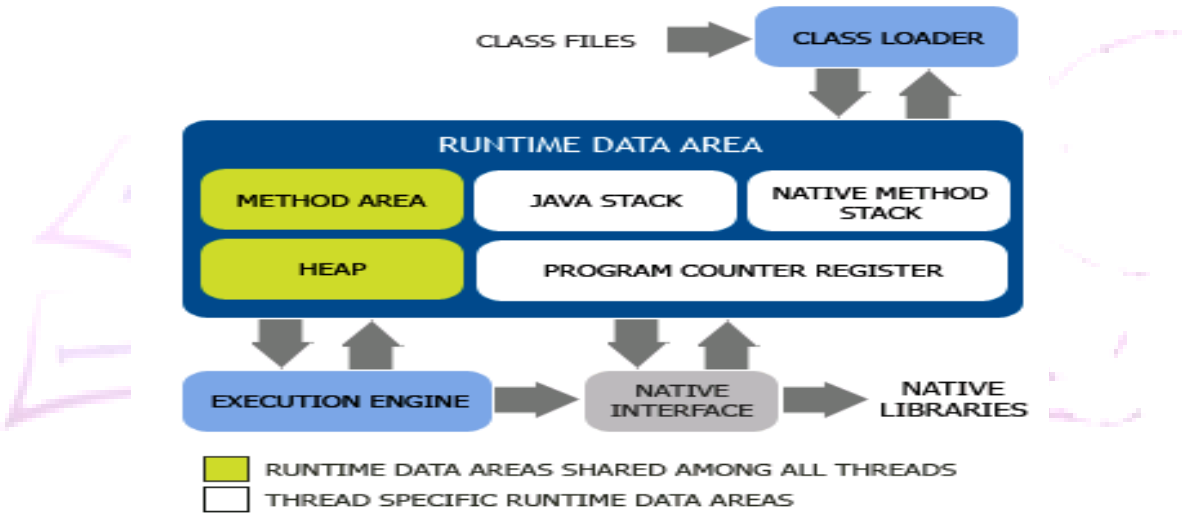
II. The Java Technology

Each Java class of a given application is compiled to a virtual machine designated by **JVM** (**J**ava **V**irtual **M**achine) [3]. The compilation results of each Java class reside in a file named Java *class file* which contains a constant pool (with symbolic references, constant strings, and information about the class) and the *byte codes* for each method, among other information [3]. The **JVM** has about 200 instructions that can be distributed over 30 functionality type groups and each instruction has an 8-bits opcode. The **JVM** is stack-oriented, with an operand's stack and local variables for each method. The operations get values from the operand stack and store the result in it. The operand stack is also used to pass arguments to methods and to

receive the return result from a called method. There are instructions to transfer word contents from/to the local variables to/from the operand stack, instructions related to the manipulation and creation of objects, arrays, and the information encapsulated within it, control-flow instruct.

III. JVM Architecture

A Java virtual machine (JVM) is a virtual machine that can execute Java byte code. It is the code execution component of the Java software platform. Sun Microsystems has stated that there are over 5.5 billion JVM-enabled devices. In order to write and execute a software program you need the following.
1.Editor – To type your program into, a notepad could be used for this.
2.Compiler – To convert your high language program into native machine code.
3.Linker – To combine different program files reference in your main program together.
4.Loader – The loader can be copy files on your hard disk device like Flash Drive, CD into RAM for run. The filling is mechanically done when *your* run your code.
5Execution – Genuine run the program code, if you need is cleared that by your Operating System and CPU.



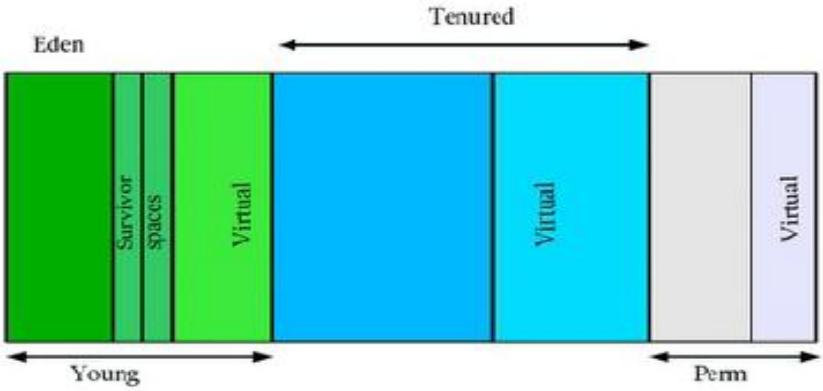
1.1 JVM Architecture

Why is Java slow?

- **Dynamic Linking** = Unlike C, linking is done at run-time, every time the program is run in Java.
- **Run-time Interpreter** = the conversion of byte code into native machine code is done at run-time in Java which furthers slows down the speed.

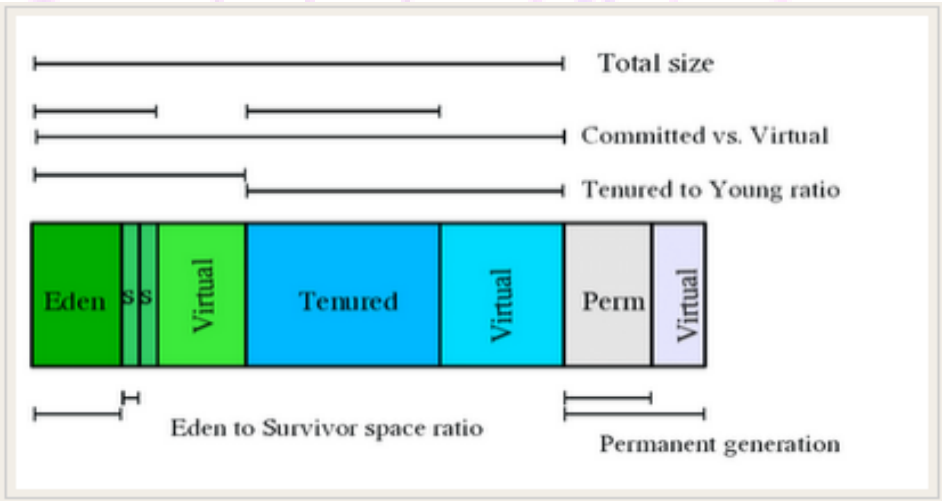
IV. JVM Memory Architecture

JVM one of important module of Java architecture and portion of the JRE (Java Runtime Environment). There are provided the annoyed stage functionality to java. It's a software procedure that transforms the complier into Java byte code into machine code. Not a machine code just a byte code is an intermediate language in the middle of Java source and the parent system. Greatest program design language like C and Pascal exchanges the basis code into computer language such as machine code for unique detailed style of instrument as the machine language differ from organization to organization. Mostly compiler produces code for a particular system but Java compiler converts a source to machine it's the structure can be used for a virtual machine. JVM offers security to java.



1.2 JVM Memory 1

- **Eden Space (heap):** The group beginning which memory is initially allotted? For most objects.
- **Survivor Space (heap):** The group Containing objects that have continued. The garbage collection of the Eden space.
- **Ensured Generation (heap):** The pool comprising objects that have existed for some time in the survivor space.
- **Permanent Generation (non-heap):** The pool containing all the reflective data of the virtual machine itself, such as class and method objects. Together Java VMs in order practice class data distribution, present age group is separated interested in read-only and read-write areas.
- **Code Cache (non-heap):** The Hotspot Java VM also contains a code cache, containing memory that is used for compilation and storage of native code.



1.3 JVM Memory 2

The heap that one is separated into three areas: **Tenured** (also called Old), **Young** (also called New), and **Permanent** (Perm). The main motive last this is to type garbage collection more well-organized. Permanent should never require collection (although you can configure it), Old should be rarely collected, and Young will be collected a lot.

V. Heap Metrics

So how is the total memory distributed onto the different areas? The JVM offers us to keep track of a lot of information on memory via JMX. For example, there is a good command line utility which shows you the usage of all your spaces.

Syntax: jstat -gcutil <ProcessID>
D:\Tejinder>jstat -gc 0

```
C:\Java>jstat -gcutil 0
  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT
  0.00  83.38  45.63   0.00   2.94      1    0.002     0    0.000    0.002
```

1.4 JVM GC

VI. JVM Pros and Cons

A virtual machine is a layer of abstraction that gives a program one simplified interface for interacting with a variety of physical computers and their operating systems.

- **Security:**Java virtual machine (VM) is that the virtual machine attempts to verify all programming before it is executed on behalf of various movement and straight thoughtful faults inside a Java code is restricted to the virtual machine's sandbox.
- **Cross Platform**: A tremendous advantage of the Java VM is that it allows a program to be written and compiled only once, which then can be run on a wide variety of systems and operating systems without modification. Many cell phones and embedded devices include a Java VM.
- **Speed :** Since programming must be translated from generic "byte code" to the machine code for the target system as it is being run, it is unbearable for Java to complete as rapidly as languages that can converts into machine code directly for the goal to systems.
- **Platform Specific Features:** Java VM must be execute continuously an extensive variability of systems; structures definite to a single OS are frequently not applied into Java programs. In accumulation, the "appearance and texture" of Java applications can regularly be moderately altered than the evasion styles of native applications within an operating system.

VII. Future Work

There are 800,000 lines of C/C++ code in Sun's JDK 1.6. We expect Robusta should be able to sandbox most of JDK's system libraries, as we have demonstrated for zip and libec. However, it is possible that not all native libraries for system classes are suitable for Robusta because of restrictions related to functionality or performance. Some system native libraries may need direct accesses to the JVM state. For instance, a security manager accesses JVM's method-call stack directly. Some system classes' native libraries may cross the boundary between the Java and native worlds so often that putting them into a sandbox would have a significant performance penalty for the JVM; in these cases. On the other hand, it does not prevent Exploits of vulnerabilities using code snippets already in the code region (e.g., return-to-libc attacks or return-oriented programming). Control-Flow Integrity can foil a large number of attacks that are based on illegal control transfers.

VIII. Conclusion

I have described the techniques used in order to provide an HW performance acceleration of Java byte codes. The developed GALADRIEL compiler allows an Efficient front-end to HLS systems starting from a system Java specification. Using the Java VM as a propagation vector for distributing dynamic languages out to a large audience is a tempting goal. We've examined two difficulties with targeting dynamic

languages to the current Java VM: the overhead of boxing small data structures, and the mismatch between the VM's Java-tuned instruction set and the requirements of dynamic languages. It appears that this extension should have no performance impact on current Java programs. Native code has always been the security dark corner of Java security.

IX. References

[1] "The Java™ Virtual Machine SpecificationJava SE 7 Edition", Tim Lindholm Frank Yellin, GiladBracha, Alex Buckley, 2012-07-27, JSR-000924 Java™ Virtual Machine Specification ("Specification") Version: 7, 2011 Oracle America, Inc. and/or its affiliates.

[2] Tim Lind Holm and Frank Yellin. The Java Virtual Machine Specification.Addison-Wesley, Reading, Massachusetts, 1996.

[3] http://www.javatutorialhub.com/java-virtual-machine-jvm.html

[4] http://middleware7.blogspot.in/2012/08/Jvm-memory-architecture.html

[5] "Robusta: Taming the Native Beast of the JVM", Joseph Siefers, Gang Tan, Greg Morrisett, *CCS'10,* October 4–8, 2010, Chicago, Illinois, USA.

[6] "Towards an Automatic Path from Java Bytecodes to Hardware through High-Level", João M P Cardoso, Horácio C Neto,In Proceedings of the 5th IEEE International Conference on Electronics, Circuits and Systems, Lisbon, Portugal, September 7-10, 1998.

[7] "The Java™ Virtual Machine Specification Java SE 7 Edition", Tim Lindholm, Frank Yellin, GiladBracha
Alex Buckley, 2012-07- 27, Oracle America, Inc. and/or its affiliates. All rights reserved. 500 Oracle Parkway M/S 5op7, California 94065, U.S.A.

[8] "Design and Performance Analysis of a Distributed Java Virtual Machine", MinhaiSurdeanu and Dan Moldovan,