VU Islamabad Campus (Help Point)

CS508-Mordren Programming Languages
Assignment Solution File

Question No.1

Sol:

My choice for this particular scenario is FORTRAN. Reason is as following!!!
It's installed on any large scale parallel machine.
Fortran has strict aliasing semantics compared to C++ and has been aggressively tuned for numerical performance for decades. Algorithms that use the CPU to work with arrays of data often have the potential to benefit from a FORTRAN implementation.
It has a simple data abstraction model.
Another factor in your decision is the amount of sample code and the quantity of reference implementations available. Fortran's strong history means that there is a wealth of numerical code available for download and even with a trip to the library. As always you will need to sift through it to find the good stuff.

Question No.2

Sol:
Memory Management in Java
JVM has mainly two area
Heap and stack
Heap:
Heap area splite into two one is called young generation and one is old.
Young generation is the place where all the new objects are created. When young generation is filled, garbage collection is performed. This garbage collection is called Minor GC. Young Generation is divided into three parts – Eden Memory and two Survivor Memory spaces.
Important Points about Young Generation Spaces:
Most of the newly created objects are located in the Eden memory space.
When Eden space is filled with objects, Minor GC is performed and all the survivor objects are moved to one of the survivor spaces.
Minor GC also checks the survivor objects and move them to the other survivor space. So at a time, one of the survivor space is always empty.
Objects that are survived after many cycles of GC, are moved to the Old generation memory space. Usually it's done by setting a threshold for the age of the young generation objects before they become eligible to promote to Old generation.
Memory Management in Java – Old Generation
Old Generation memory contains the objects that are long lived and survived after many rounds of Minor GC. Usually garbage collection is performed in Old Generation memory when it's full. Old Generation Garbage Collection is called Major GC and usually takes longer time.
Memory Management in Java – Java Garbage Collection
Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space to be allocated to objects created in the future processing. One of the best feature of java programming language is the **automatic garbage collection**, unlike other programming languages such as C where memory allocation and deal location is a manual process.
Garbage Collector is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program. All these

unreferenced objects are deleted and space is reclaimed for allocation to other objects.
One of the basic way of garbage collection involves three steps:
Marking: This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
Normal Deletion: Garbage Collector removes the unused objects and reclaim the free space to be allocated to other objects.
Deletion with Compacting: For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.
There are two problems with simple mark and delete approach.
First one is that it's not efficient because most of the newly created objects will become unused
Secondly objects that are in-use for multiple garbage collection cycle are most likely to be in-use for future cycles too.
The above shortcomings with the simple approach are the reason that **Java Garbage Collection is Generational** and we have **Young Generation** and **Old Generation** spaces in the heap memory. I have already explained above how objects are scanned and moved from one generational space to another based on the Minor GC and Major GC.
Memory Management in Java – Java Garbage Collection Types
There are five types of garbage collection types that we can use in our applications. We just need to use JVM switch to enable the garbage collection strategy for the application. Let's look at each of them one by one.
Serial GC (-XX:+UseSerialGC): Serial GC uses the simple mark-sweep-compact approach for young and old generations garbage collection i.e Minor and Major GC.
Serial GC is useful in client-machines such as our simple stand alone applications and machines with smaller CPU. It is good for small applications with low memory footprint.
Parallel GC (-XX:+UseParallelGC): Parallel GC is same as Serial GC except that is spawns N threads for young generation garbage collection where N is the number of CPU cores in the system. We can control the number of threads using -XX:ParallelGCThreads=n JVM option.
Parallel Garbage Collector is also called throughput collector because it uses multiple CPUs to speed up the GC performance. Parallel GC uses single thread for Old Generation garbage collection.
Parallel Old GC (-XX:+UseParallelOldGC): This is same as Parallel GC except that it uses multiple threads for both Young Generation and Old Generation garbage collection.
Concurrent Mark Sweep (CMS) Collector (-XX:+UseConcMarkSweepGC): CMS Collector is also referred as concurrent low pause collector. It does the garbage collection for Old generation. CMS collector tries to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads.
CMS collector on young generation uses the same algorithm as that of the parallel collector. This garbage collector is suitable for responsive applications where we can't afford longer pause times. We can limit the number of threads in CMS collector using -XX:ParallelCMSThreads=n JVM option.
G1 Garbage Collector (-XX:+UseG1GC): The Garbage First or G1 garbage collector is available from Java 7 and it's long term goal is to replace the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector.
Garbage First Collector doesn't work like other collectors and there is no concept of Young and Old generation space. It divides the heap space into multiple equal-sized heap regions. When a garbage collection is invoked, it first collects the region with lesser live data, hence "Garbage First". You can find more details about it at Garbage-First Collector Oracle Documentation.