Garbage Collection
Project Summary

Contributors: Julia Chapple, Dallas Treder, Richard Thai

Last Updated: 15 May 2013

- Project Statuses
  - Week of 4/23
  - Week of 4/30
  - Week of 5/7
- Planning
- Testing
  - Introduction
  - Logging
  - Debugging
  - Using the Test Harness

Project Statuses

Week of 4/23

We have explored the Mark Sweep code and developed a somewhat rudimentary understanding of its features and associations. We have created our own version of the mark sweep collector (using these instructions: http://jikesrvm.org/Adding+a+New+GC) and enabled logging (located here: results/buildit/localhost-[date]/tests/local/BaseBaseMyGC/gctest/ReferenceTest.default-output.txt
example: results/buildit/localhost-2013-04-21-Sun-17-40-16/tests/local/BaseBaseMarkSweepTuned/gctest/ReferenceTest.default-output.txt)
to figure out program flow. This will allow us to determine how our new code is being used and whether our tuning adjustments are being used.
Concerning the instructions to make a new GC, there is one issue: in step 1, the directory is MMTk/src/org/mmtk/plan/marksweep (src level was excluded if you're doing it by hand).
Viewing the log file, the following occurred:

1. MSTunedMutator.alloc and MSTunedMutator.postAlloc methods were called repeatedly
2. MSTuned.getPagesUsed method was called once
3. MSTunedCollector.global called
4. Steps 1 and 2 occurred repeatedly with the amount of alloc/postAlloc method calls varying.
5. MSTuned.willNeverMove method is called, and the object is in the space. This occurs twice in a row.
6. Steps 1 and 2 occurred repeatedly with the amount of alloc/postAlloc method calls varying.
7. Steps 4 and 5 repeat a couple of times.
8. MarkSweepspace.TraceObject is called.
9. Steps 1 through 8 repeat.

Week of 4/30

We've been reaching out to the community in order to better understand how garbage collectors are implemented in Jikes RVM.
A specific question that we raised concerns how bitmap marking works in the garbage collector. We've noticed within the properties files a specific attribute that seems to control bitmap marking (config.mmtk.headerMarkBit=true) in the BaseBaseMarkSweepTuned.properties file.
The response we received was that this flag switches between using mark bits in the header of an object and mark bits in a 'side' dense bitmap (he default is to use the header). We were also referred to a white paper that better explains the mark bit mechanism (section 4 in particular): http://cs.anu.edu.au/~Robin.Garner/pf-ismm-2007.pdf.
We were also referred to a reference that explained memory allocation in jikes (http://jikesrvm.org/Memory+Allocation+in+JikesRVM), though this was not helpful.
We rephrased the question to the mailing list so that we could map our understanding of an existing algorithm (mark sweep) to the implementation in jikes. One response to this question

mentioned that garbage collectors in Jikes RVM work in phases. There are phases for mutator and collector. This meant that there were no explicit calls to invoke mark and sweep phases.
For mark sweep collector, marking is done by tracing the heap. Tracing starts from the roots (see STACK_ROOTS and ROOTS phases in SimpleCollector.java) and finishes with a transitive closure (see CLOSURE phase in MSCollector.java). Sweeping is done by walking through the heap and sweep objects that are not marked (see msSpace.release method of RELEASE phase in MS.java).
We'll be looking at processPhaseStack method in Phase.java in order to better understand how the phases are executed.
We also asked the community how to get the MMtk test harness working, as we could not get it built. They responded, and so we are attempting to use that to understand the code flow.
We are also trying to explore the code with the gdb debugger. Getting this working will allow us to watch the execution of of code and see the various calls. We are currently having problems setting breakpoints correctly. To set a breakpoint the associated part of memory must be loaded. Furthermore, one must examine the memory map to find the address in memory. We contacted the mailing list about this and received some valuable feedback. They indicated some places where breakpoints can be set that allow access to more parts of the code. We now can set breakpoints in some parts of the code but other sections are currently inaccessiblei. It should just be a matter of finding memory accessible addresses to break at and using those breakpoints to access areas we are interested in.

Week of 5/7

Despite our efforts we have made no further progress on setting meaningful breakpoints with gdb. The jikes community indicated that debugging with gdb is not very valuable and generally "tedious." We've suspended our work with gdb to focus on exploring the code.
We now have a better understanding of the structure of the Mark Sweep collector. Particularly we have improved of understanding of the different phases and where they are executed. We also now have a better understanding of the general structure of Jikes and how different collectors are controlled on the top level thanks to a response by Robin Garner (link here: http://docs.codehaus.org/display/RVM/Anatomy+of+a+Garbage+Collector).
We started exploring how to implement lazy sweeping. To this end we investigated the control of the sweep step and the connection between the mutator and the collector. While we were doing this we found that lazy sweeping was already implemented and only needs to be enabled.

Planning

The first half of our endeavor involved learning about some fundamental garbage collection concepts, different styles of garbage collectors, and issues to consider with each style of collection. In addition, we also took a closer look at the Jikes RVM (downloading, using, testing). Following the resources located on the Jikes RVM webpage, we had followed some basic tutorials that walked us through creating a new garbage collector, the architecture, and the memory management interface. After having familiarized ourselves with garbage collection and the Jikes RVM, we set out to plan what we would try to accomplish for the project.

Originally we had planned to implement a mark-sweep collector and, time allowing, optimize on the implementation with a couple of optimizations discussed in our textbook: bitmap marking and lazy sweeping. If we had additional time, we were considering addressing fragmentation in the heap with the mark-compact or applying the existing mark sweep to a generational garbage collector.

What you'll notice from the first project status (week of 4/23), is that creating an implementation of a mark sweep collector was relatively straightforward; we would copy existing Jikes RVM implementation to alter. This was the recommended method for creating new implementations of collectors, and it would allow us to overlook a lot of details that were specific to the virtual machine and not the collector.

Having created a collector, we needed to better understand the architecture of a collector in order to implement the planned optimizations: bitmap marking and lazy sweeping. To do this, we set out to trace what classes and methods were enacted when the garbage collector was called. We embedded logging messages throughout the collector-specific classes which would allow us an idea of the order methods were called as well as their frequency. Running the garbage collector with these log messages gave us a log file that was unhelpful towards understanding the inner workings of the mark sweep collector. A big reason for this was that the garbage collector was called with each instantiation of the virtual machine, which would insert a lot of calls that were not specific to the application we had wanted to test the garbage collector with.

In addition to this, we explored other avenues of trying to better our understanding of the garbage collector. We would explore manually stepping through the collector with gdb as well as use existing options in the testing harness to allow us to probe the garbage collector. We better explore these two methods in the next section.

Our exploration of the garbage collector eventually lead to the discovery that bitmap marking (week of 4/30) and lazy sweeping (week of 5/7) had already existed within the Jikes RVM implementation. Bitmap marking can be found in the MarkSweepSpace class--with a variable called HEADER_MARK_BITS which acts as a flag to enable bitmap marking. Likewise, an implementation for lazy sweeping can be found in the MarkSweepSpace class--with a variable called LAZY_SWEEP which also acts as a flag to enable this optimization.

Having discovered these features, we decided to make better use of our efforts by documenting our experiences and understanding of the Jikes RVM for future courses. Having started on ground zero, we compiled a list of necessary dependencies, packages, and possible issues that can crop up while utilizing the Jikes RVM--a nontrivial effort. In addition to our issues, we also pose solutions that ought to work in the same system (Ubuntu 12.10 32-bit).

Testing

Introduction
We used testing as both a method of understanding the garbage collector and verifying its functionality. We employed multiple approaches to testing the collector. Each of them helped us somewhat, but had major drawbacks that meant that we tried other methods that we hoped would suit our purposes better.

Logging
First, we wrote to a log at the beginning of every relevant method. We used the method Log.write from the org.mmtk.utility.Log package. The log helped us understand the the different calls made by the collector and see some variables of interest. Furthermore, when we ran the collector in the command line the log printed. Hence if we included prints in our mutator we could see how the collector and the mutator interleaved. This approach was limited because the log was very difficult to understand. It had many entries and no data that would allow us to understand the call stack.

Debugging
To further our understanding we tried to use gdb to debug the collector and thus trace its execution. This approach was stymied by the complexity of Jikes. While it is possible to debug Jikes with gdb, setting breakpoints is extremely difficult. Since it is a separate VM, before you can set a breakpoint, the memory of the collector must be loaded into memory. Thus gdb must break after the collector is loaded and before it starts executing. It is difficult to determine where this happens. Although we were able to set some breakpoints in Jikes we could never get it to set them within the collector.

Using the Test Harness
We also tried to use the built-in test harness in MMTk to test the garbage collectors. While there was a bug in the version of Jikes that prevented the harness from being built, it was easily corrected and fixed in the next version. Once we were able to run tests on the different garbage collectors, we attempted to use its tracing functionality to be able to determine the flow of the garbage collector. However, the test harness' trace options left much to be desired; they either only reported in the most general situations (e.g. printed a single line whenever the garbage collector was called), reported the same information that we could already get from our logging approach, or simply was not printing anything even though it should have been.