New Parallel Full GC

The current implementation of ParallelGC's Full GC is not suitable for Lilliput 2. In particular, it does not work with Compact Identity Hashcode. The reason for that is the compressor-style algorithm that is used in this Parallel Full GC. This algorithm precomputes forwarding addresses of objects in bulk (e.g. 64-word-blocks) and then only requires a few additional computations to get the true forwarding address of each object. The problem with this approach is that it is too rigid and does not allow for expanding of moving objects, which is necessary for Compact Identity Hashcode. Here I want to outline a parallel mark-compact algorithm that does not have this restriction.

The basic idea is to use an algorithm that is similar to the one that's used by G1 and Shenandoah. We assume that the first phase of the full GC identifies all reachable objects by marking them in a marking bitmap, as is already implemented by the current implementation. Unless G1 and Shenandoah, the Parallel GC does not divide the heap into regions. This is a problem, because division into regions is the important feature that facilitates the parallelism in G1 and Shenandoah. However, the current full GC implementation provides machinery to partition the heap spaces into regions and deal with objects that overlap region boundaries.

Forwarding Phase

The forwarding phase is the 2nd phase of the full GC, and starts after marking completed. This phase scans the heap and computes the forwarding addresses for all reachable objects.

The work of calculating the forwarding addresses is carried by multiple GC workers, therefore we need a way to divide up the work between the GC threads. This is achieved as follows:

The heap is divided into N equal-sized regions, and we start out with a list of regions. This list comprises the whole heap.

Each worker iterates this list. By doing so, the worker claims regions, i.e. it atomically claims ownership of the whole region. This means, that from there on, it is this worker's responsibility to handle all work related to computing the forwarding addresses, and later to do the copying of all objects in that region. A special case are objects that overlap region boundaries: in this case, the overlapping object is 'owned' by the worker that owns the region that the object overlaps into.