Brief Announcement: ParMarkSplit: A Parallel Mark-Split Garbage Collector Based on a Lock-Free Skip-List

Nhan Nguyen[1], Philippas Tsigas[1], and Håkan Sundell[2]

[1] Chalmers University of Technology, Sweden
{nhann,tsigas}@chalmers.se
[2] University of Borås, Sweden
Hakan.Sundell@hb.se

Abstract. This brief announcement provides a high level overview of a parallel mark-split garbage collector. Our parallel design introduces and makes use of an efficient concurrency control mechanism based on a lock-free skip-list design for handling the list of free memory intervals. We have implemented the parallel mark-split garbage collector in OpenJDK HotSpot as a parallel and concurrent garbage collector for the old generation. We experimentally evaluate the collector and compare it with the default concurrent mark-sweep garbage collector in OpenJDK HotSpot, using the DaCapo benchmarks.

1 Motivation

Garbage collection (GC) is an important component of many modern programming languages and runtime systems. As parallelism has become a core issue in the design and implementation of software systems, garbage collection algorithms have been parallelized and evaluated for their potential a range of scenarios. However, none of them can outperform the other in all use cases and researchers are still trying to improve different aspects of garbage collection.

Mark-split is a new GC technique introduced by Sagonas and Wilhelmsson [1] that combines advantages of mark-sweep and copying algorithms. Mark-split evolves from mark-sweep but removes the sweep phase. Instead, it creates the list of free memory while marking by using a special operation called *split*. Mark-split does not move objects, uses little extra space and has time complexity proportional to the size of the live data set. These advantages help it outperform mark-sweep in certain scenarios in sequential environment [1]. Whether it can maintain the advantages in a parallel environment remains an open question.

As mark-split repeatedly searches for and splits memory spaces, a high performance concurrent data structure to store the spaces is essential to the parallel design of mark-split. Lock-free data structures offer scalability and high throughput, guarantee progress, immune to deadlocks and livelocks. Several lock-free implementations of data structures have been introduced in the literature [2] [3], and included in Intels Threading Building Blocks Framework, the PEPPHER framework [4], the Java concurrency package, and the Microsoft .NET Framework. Skip-list is a search data structure which provides expected logarithmic time search without the need to rebalance like balanced trees. The skip-list algorithm by Sundell and Tsigas [5] [6] is the first lock-free skip-list introduced in the literature. It is an efficient and practical lock-free implementation that is suitable for both fully concurrent (large multi-processor) systems as well as pre-emptive (multi-process) systems. We opt to extend it to store free memory spaces in our parallel mark-split.

2 Our Results

We extend the lock-free skip-list so that it is capable to handle the free memory intervals for mark-split. It is because the basic operations supported by the original lock-free skip-list, e.g *search*, *insert*, *remove*, are not strong enough to satisfy the functionality requirement of mark-split in concurrent environment. Our extension including a sophisticated concurrency control allows the skip-list to execute more complex operations such as *split* in mark-split algorithm.

Using the extended skip-list, we implement a parallel version of mark-split, namely ParMarkSplit, as a garbage collector in the OpenJDK HotSpot virtual machine. The collector performs marking and splitting in parallel to take advantage of multi-core architectures. In addition, a lazy-splitting mechanism is designed to improve the performance of the parallel mark-split.

The ParMarkSplit was evaluated and compared against a naive parallelized mark-split and the Concurrent Mark-Sweep collector bundled with the HotSpot. The former was a parallel mark-split implementation using a balanced search tree based on coarse-grained locking. The experiments were done on two contemporary multiprocessor systems, one has 12 Intel Nehalem cores with Hyper-Threading and the other has 48 AMD Bulldozer cores. A detailed version of our results will appear in a subsequent version of this brief announcement.

References

1. Sagonas, K., Wilhelmsson, J.: Mark and split. In: Proceedings of the 5th International Symposium on Memory Management, ISMM 2006, pp. 29–39. ACM (2006)
2. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann (2008)
3. Cederman, D., Gidenstam, A., Ha, P., Sundell, H., Papatriantafilou, M., Tsigas, P.: Lock-free concurrent data structures. In: Pllana, S., Xhafa, F. (eds.) Programming Multi-Core and Many-Core Computing Systems. Wiley-Blackwell (2014)
4. Benkner, S., Pllana, S., Larsson Traff, J., Tsigas, P., Dolinsky, U., Augonnet, C., Bachmayer, B., Kessler, C., Moloney, D., Osipov, V.: PEPPHER: Efficient and Productive Usage of Hybrid Computing Systems. IEEE Micro (99), 1 (2011)
5. Sundell, H., Tsigas, P.: Fast and lock-free concurrent priority queues for multi-thread systems. J. Parallel Distrib. Comput. 65(5), 609–627 (2005)
6. Sundell, H., Tsigas, P.: Scalable and lock-free concurrent dictionaries. In: Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 1438–1445. ACM (2004)