Project Documentation: Short-term Memory for Self-collecting Mutators: Benchmarks

Andreas Haas, 0421949
Andreas Schönegger, 0420929

Adviser: Christoph Kirsch

University of Salzburg
Department of Computer Science

Embedded Software Engineering, WS 2009/10

February 7, 2010

Part I

Introduction

Andreas Haas developed a memory management system, self-collecting mutators. The aim of this project was to develop benchmarks to compare self-collecting mutators with other memory management systems. The project documentation is structured as follows:

1. First we discuss the systems which we compared with self-collecting mutators
2. Then we describe the metrices we want to use
3. We present the programs we used as benchmarks
4. At last we show the results of the benchmarks

More specific details about the system can be found in the paper “Short-term Memory for Self-collecting Mutators" by Haas, Kirsch, Payer, Schönegger and Sokolova.

Part II

Systems

Self-collecting mutator were implemented in the Jikes research virtual machine. There already exist many implementations of garbage collectors in Jikes. We decided to compare self-collecting mutators with the mark-sweep garbage collector [5] and a two-generation copying collector where the higher generation is handled by an Immix collector [2]. We chose the generational garbage collector because it is the default memory management system used in Jikes, and we decided for the mark-weep garbage collector because it is fast and it was the former default system. We disables all optimizations of Jikes to avoid distortion of the results.

Part III

Metrices

We decided for three metrics to compare the performance of the systems:

- Total runtime
  - We measured the system time at the beginning and at the end of the benchmark. The total runtime is then determined by the difference of these two time values.
- Latency
  - The major part of all of our benchmarks is regular loop. To determine the latency we measure the execution time of each single loop iteration.
- Memory Consumption
  - We log the amout of free memory at the end of every loop iteration.

Part IV

Benchmarks

benchmark	LOC	added LOCs
Monte Carlo	1450	10
JLayer mp3-to-wav converter	8247	1

Table 1: The lines of code of the benchmarks and the effort of adepting them for self-collecting mutators

We chose two benchmarks: the Monte Carlo program of the Grande Java Benchmark Suite [4] and the mp3-to-wav converter JLayer[1]. The code of both benchmarks was easy to adapt for self-collecting mutators. Metrices of the benchmarks are shown in Table 1.

The Monte Carlo benchmark contained a reachable memory leak which was automatically resolved by self-collecting mutators. This was an advantage for our system. However we also executed a benchmark configuration without the memory leak.

In the Monte Carlo benchmark a result object is generated in every loop iteration in the main loop. These result objects are collected in a result set. The performance of self-collecting mutators increases when the result objects are preallocated already before the main loop. We therefore modified the Monte Carlo benchmark to use preallocated result objects.

We also measured the time-space trade-off of self-collecting mutators which is controlled by the tick-frequency and by the number of refreshing (Both is described in the paper). In order to measure this trade-off we used the Monte Carlo benchmark without preallocated result objects.

We executed both benchmarks, the Monte Carlo- and the mp3-benchmark, in parallel to measure the behavior of self-collecting mutators executing multi-threaded applications. We also executed four instances of Monte Carlo at the same time.

We tried to adept programs of the dacapo benchmark suite [1] and the CDX benchmark [3] for self-collecting mutators. However, all programs of the dacapo benchmark suite use reflection which does not work with self-collecting mutators. The results of the CDX benchmark were not feasible for our metrices, because they do not use the memory management system enough.

[1]http://www.javazoom.net/javalayer/javalayer.html

Part V

Results

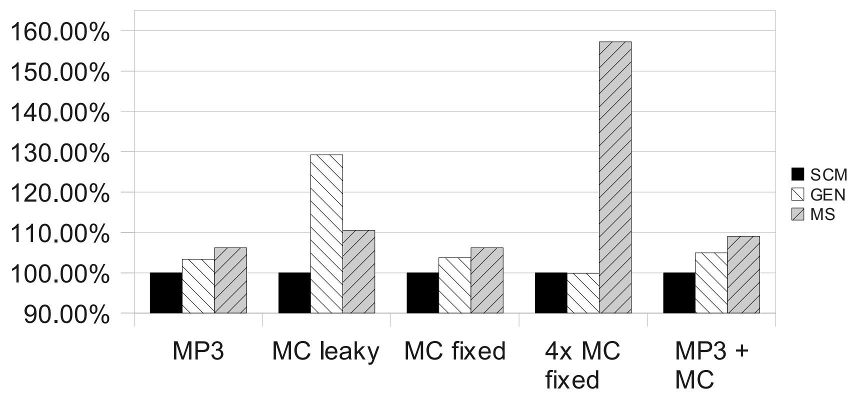

Figure 1: Total runtime of the benchmarks in percent of the runtime of the benchmark using self-collecting mutators

Figure 1, 2, 3, 4 and 5 show the results of the benchmarks. A discussion of the benchmark results can be found in the paper.

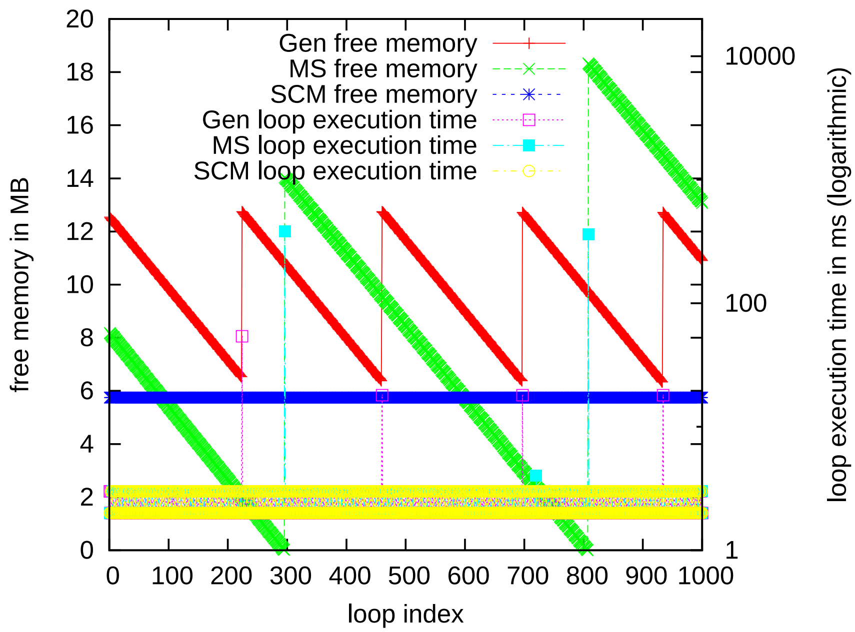

Figure 2: Free memory and loop execution time of the Monte Carlo benchmark without memory leak

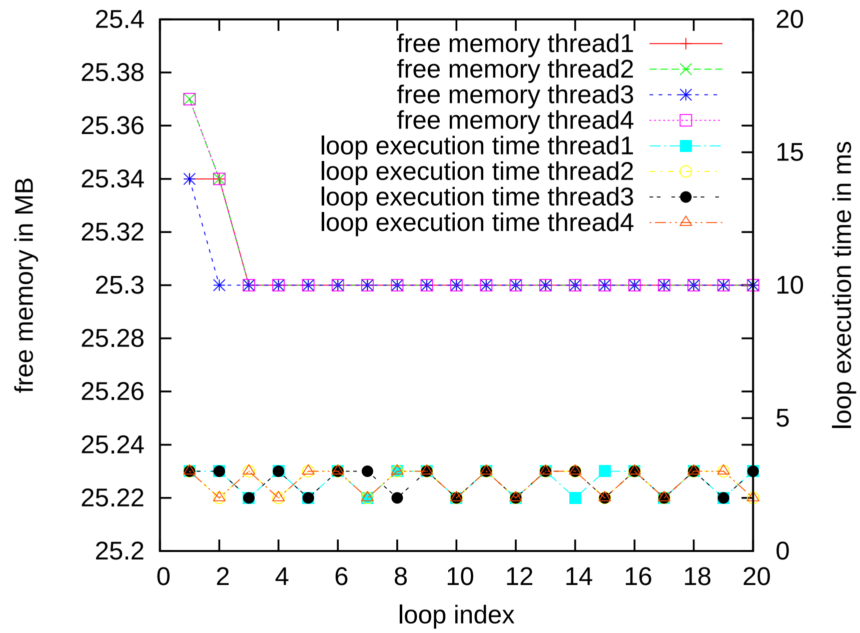

Figure 3: Free memory and loop execution time of the parallel Monte Carlo

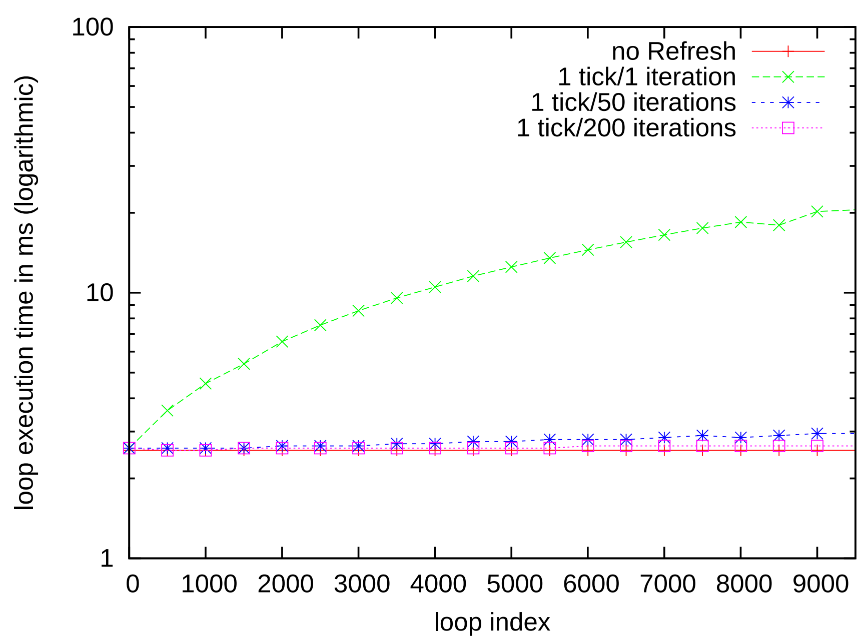

Figure 4: Loop execution time of Monte Carlo with different tick-frequencies

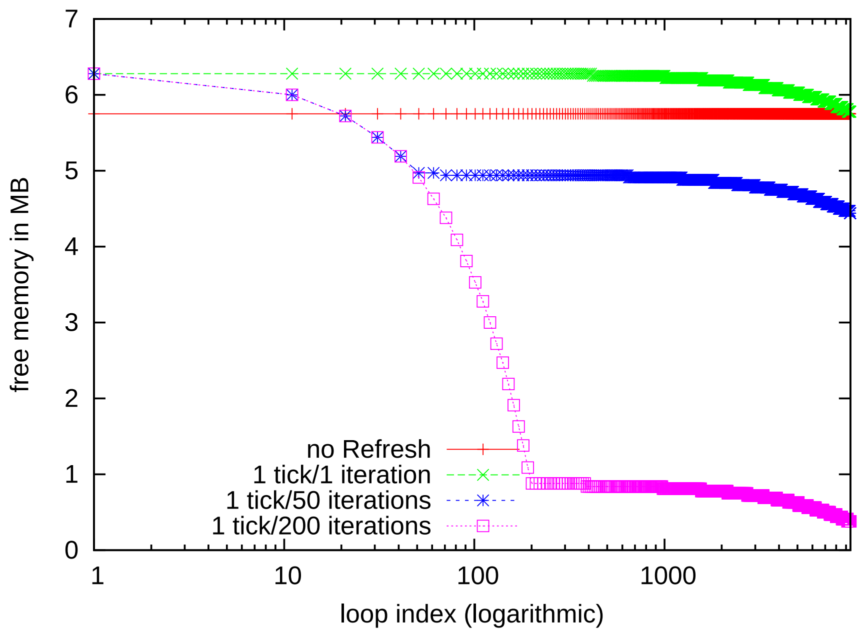

Figure 5: Free memory of Monte Carlo with different tick-frequencies

Bibliography

[1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The dacapo benchmarks: java benchmarking development and analysis. In *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 169–190, New York, NY, USA, 2006. ACM.

[2] Stephen M. Blackburn and Kathryn S. McKinley. Immix: a mark-region garbage collector with space efficiency, fast collection, and mutator performance. In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, pages 22–32, New York, NY, USA, 2008. ACM.

[3] Tomas Kalibera, Jeff Hagelberg, Filip Pizlo, Ales Plsek, Ben Titzer, and Jan Vitek. Cdx: a family of real-time java benchmarks. In *JTRES '09: Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems*, pages 41–50, New York, NY, USA, 2009. ACM.

[4] J. A. Mathew, P. D. Coddington, and K. A. Hawick. Analysis and development of java grande benchmarks. In *JAVA '99: Proceedings of the ACM 1999 conference on Java Grande*, pages 72–80, New York, NY, USA, 1999. ACM.

[5] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Commun. ACM*, 3(4):184–195, 1960.