**INTRODUCTION & MOTIVATION**

Java is one of the most used programming language these days. To execute Java programs we need a Java Virtual Machine (JVM). IBM provides a JVM called “J9 VM”.

Faster CPUs will not be the answer to all performance questions in the future, the industry moved to parallel computing. Introducing problems like deadlocks and race conditions, data coherency and consistency, and the general overhead for parallel programming and execution.

**PROBLEM**

Memory usage and management is one of the overheads. If we run two JVM based programs in parallel we have to run two JVMs. But two JVMs could possibly share a lot of data to reduce this overhead. Some of the used objects might be read only accessible and have the same content for multiple JVM instances, therefore they would be easy targets to start sharing.

This project will research some of these potential benefits by reducing the memory footprint, and sharing classes and objects between JVMs. In this sub-project we will analyze the amount of memory used and how much of that section is shareable.

Strings in Java are immutable [4] and therefore our first target object. We will measure the usage of Strings in different benchmarks.

**BENCHMARKS & TOOLS**

- **Eclipse Memory Analyzer (MAT) [3]**: “Java heap analyzer that helps you find memory leaks and reduce memory consumption”
- **SPECjbb2005 [6]**: “evaluating the performance of server side Java […] emulating a three-tier client/server system”
- **SPECjvm2008 [8]**: “focus on JRE executed a single application […] low file I/O, no network I/O”
- **daCapo [1]**: Mix of real life (e.g. headless eclipse) and area-focused applications (e.g. xml and crypto)
- **SPECjbb2013 [7]**: “Measure performance based on the latest Java application features […] a usage model based on a world-wide supermarket company IT-infrastructure”

**PRELIMINARY RESULTS**

![chart]

- **Axis:** X time in second or minute
- **Y** number of objects in thousands and Used memory in megabyte
- **Showing Char-Arrays, Strings and the memory (retained heap) used**
- **Mean:** value over the runtime
- **Makers:** Ratio to the total number of objects and used Memory
- **Retained heap:** sum of shallow sizes of all objects which would be removed by GC when X is garbage collected [2]

**REFERENCES**