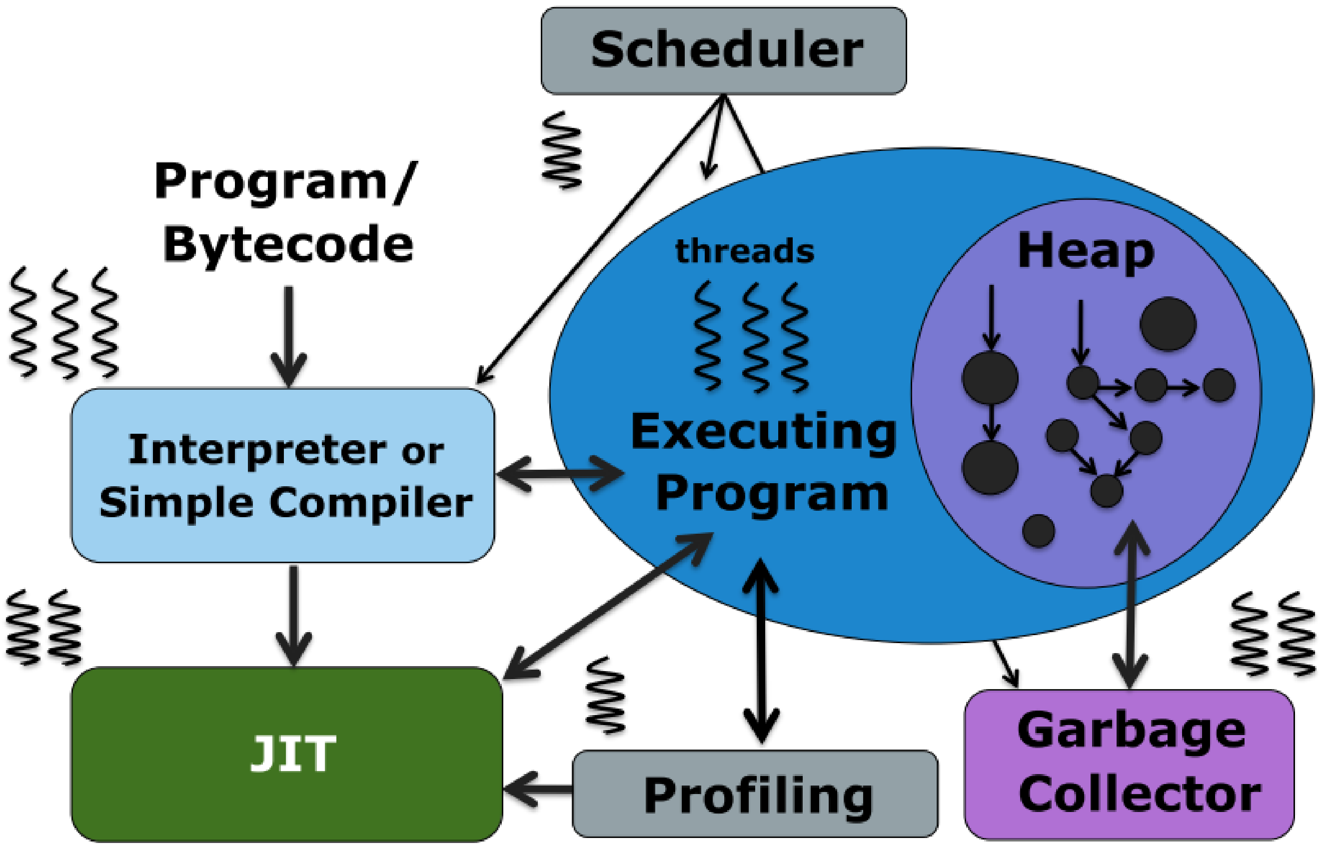
Optimising Multicore JVMs

Khaled Alnowaiser

Outline

- **JVM structure and overhead analysis**
- **Multithreaded JVM services**
- **JVM on multicore**
- **An observational study**
- **Potential JVM optimisations**

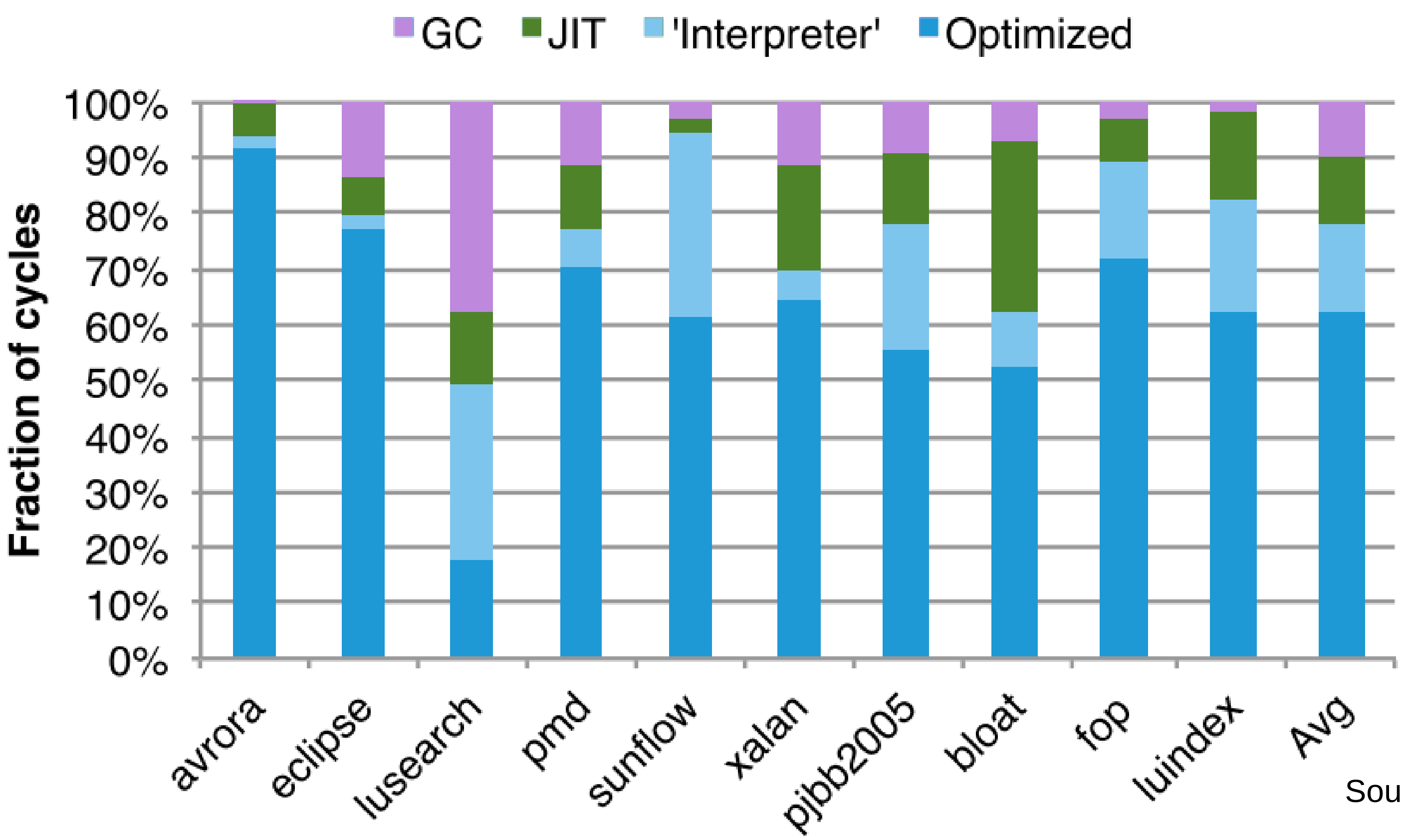
Basic JVM Services



Source: [1]

Overhead of JVM services

- **On average 40% of Application execution time is devoted to JVM services**



Source: [1]

Multithreaded JVM Services

- **Most of modern production JVMs implement multithreaded garbage collection and JIT compilation services.**

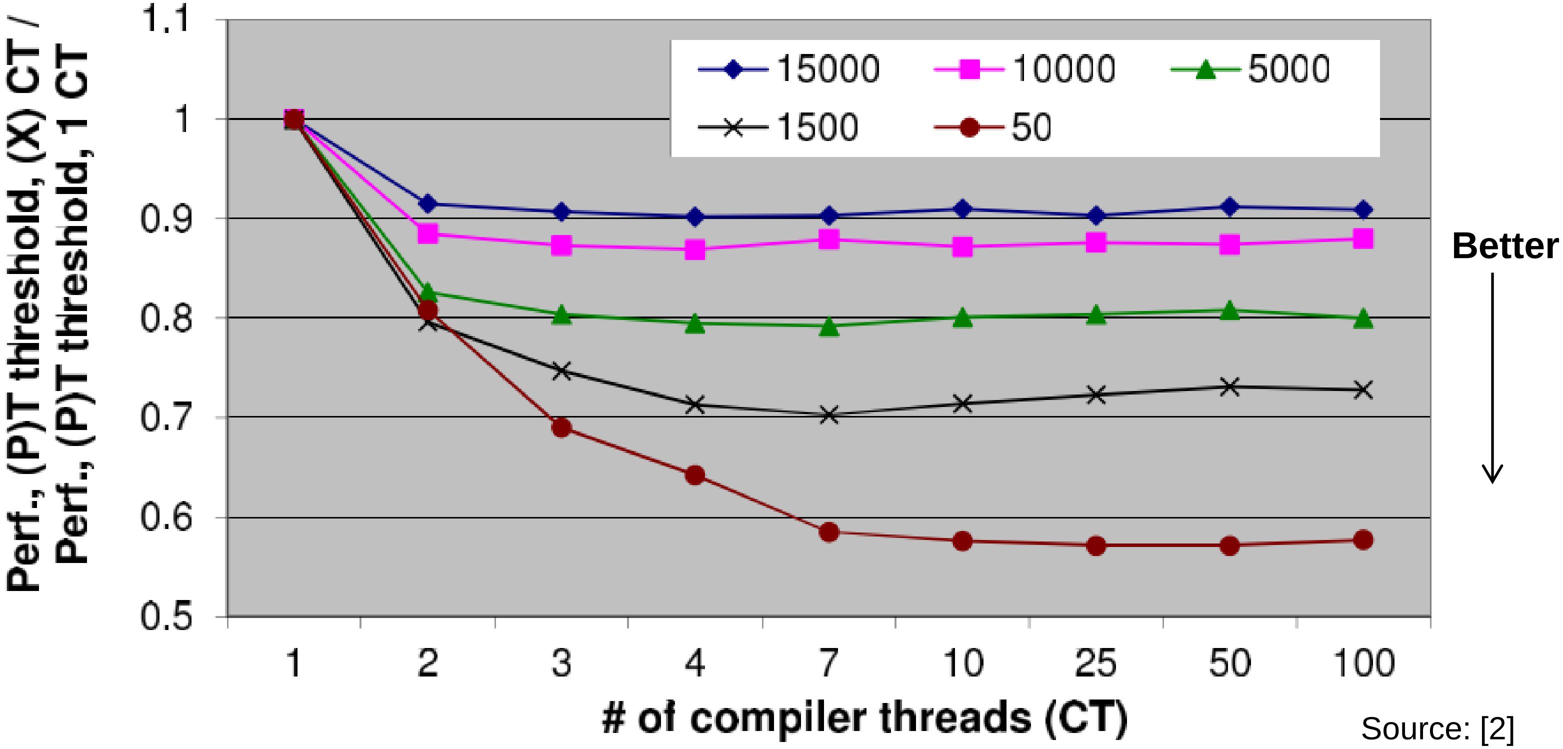
- **How the Hotspot JVM sets the number of a JVM service?**

```
Threads = 8 + 5/8 * processors
```

- **Benefits of employing multiple threads depend on several factors e.g. the amount of work and the number of threads.**

JIT Performance

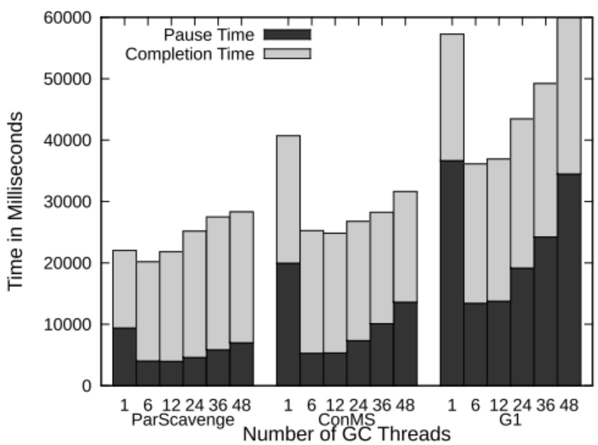
- **Increasing the number of JIT opt. compiler threads increases the application performance as long as there is work to do.**



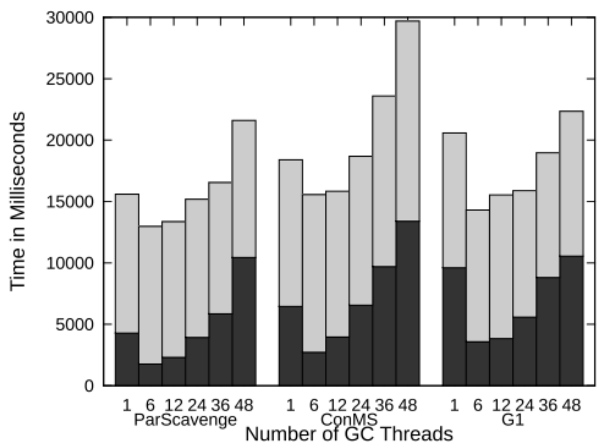
Source: [2]

GC Performance

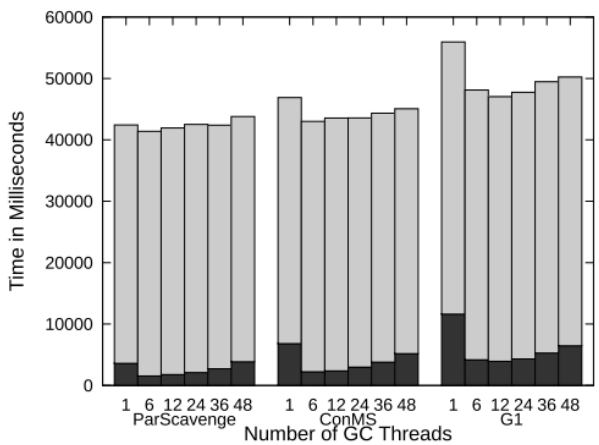
- **GC performance with multiple threads seems to have issues**



(a) Sunflow



(b) Lusearch



(c) Tomcat

Source: [3]

JVM on Multicore

- **Issues of running JVM on multicore systems**

– How many threads yield optimum performance?

– How could we distribute JVM services across multi-socket multicore systems?

JVM on Multicore

- **Summary of Sartor [4] empirical study:**

  1. On a single socket, App threads = GC threads = no. of cores, on multiple sockets, fewer collector threads is better.

  2. Offloading JVM services to another socket costs 20% performance degradation.

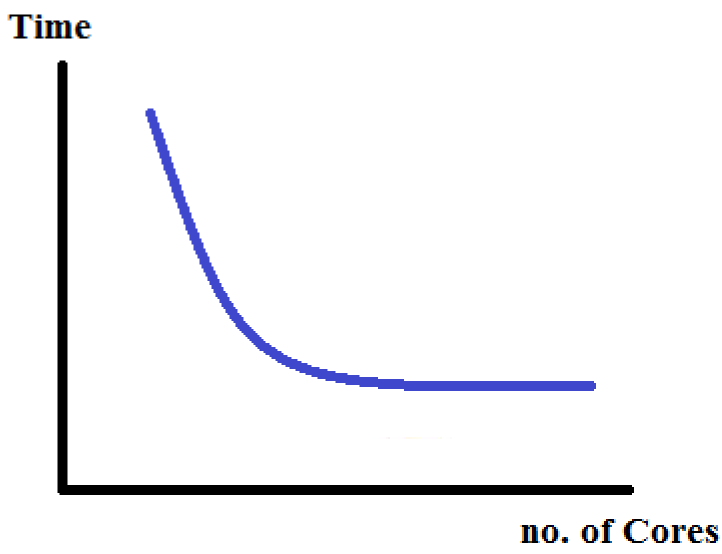
  3. Scaling down the frequency of JVM threads has less impact than application threads.

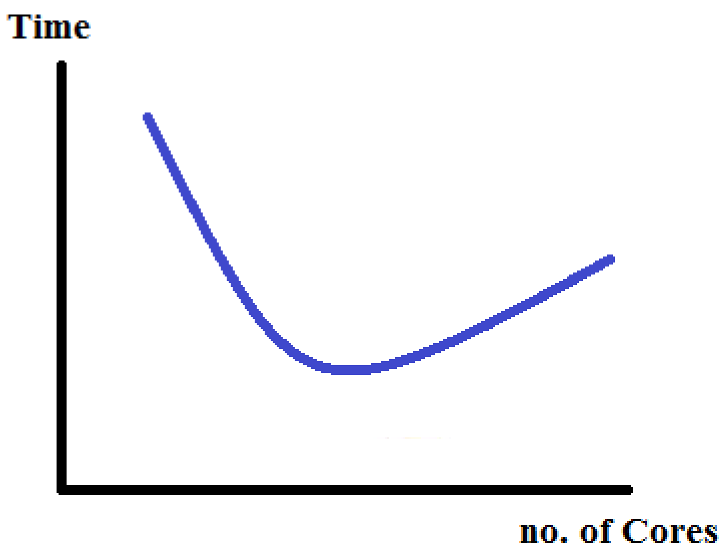
JVM Services Scalability

- **Adding more threads creates two problems:**
  1. Inefficiency

     Threads saturate beyond certain number of cores.
  2. Performance degradation

     There is a problem !





An Observational Study

- **Objective:**

  To study the Parallel GC behaviour with different number of GC threads.

- ➢ Reproduce and confirm results from other studies.

- ➢ Analyze multicore architecture overhead on parallel GC performance

Experimental Setup

- **Platform:**
  - Linux machine with 2 x 6-core Intel Xeon processors, HT enabled,
  - 12MB shared L3 cache.
- **Experimental Method:**
  - OpenJDK Hotspot JVM.
  - Dacapo-9.12 Benchmark programs.
  - The experiment is repeated 5 times and the mean is reported.
  - HW measurements is done with PAPI v5.
  - Heap Size is 3 x minimum size

Performance Profiling Tools

- **LIKWID**

  A set of tools to support developing high performance multithreaded applications

- **PAPI**

  An API for accessing hardware performance counters.

Performance Metrics

- **Architectural metrics:**

  - L3 cache misses
  - Total instructions
  - Total Cycles

- **GC Parallel processing time**

Results (Young Generation)

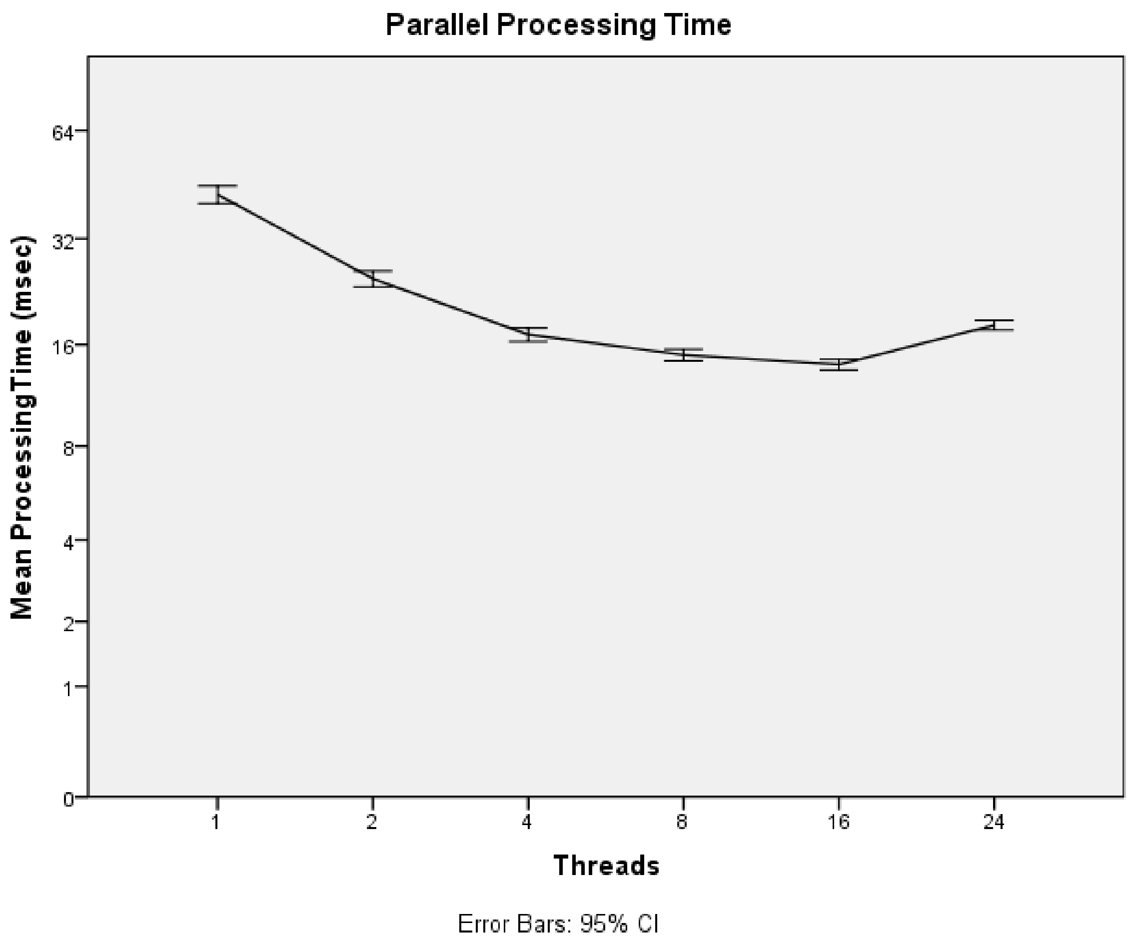


Figure1: Parallel GC time of minor collection. as the number of GC threads increases

Results (Young Generation)

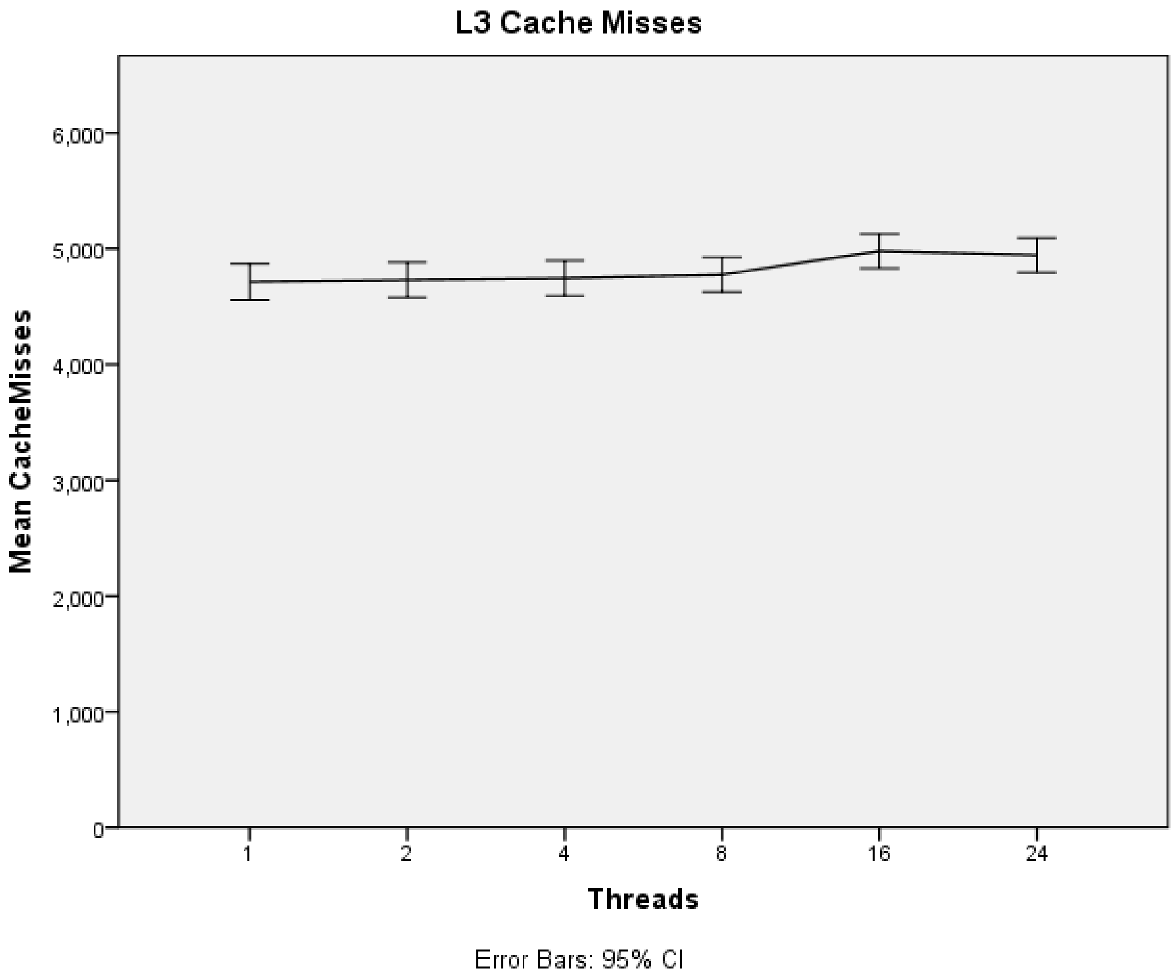


Figure2: L3 Cache misses per the number of GC threads

Results (Young Generation)

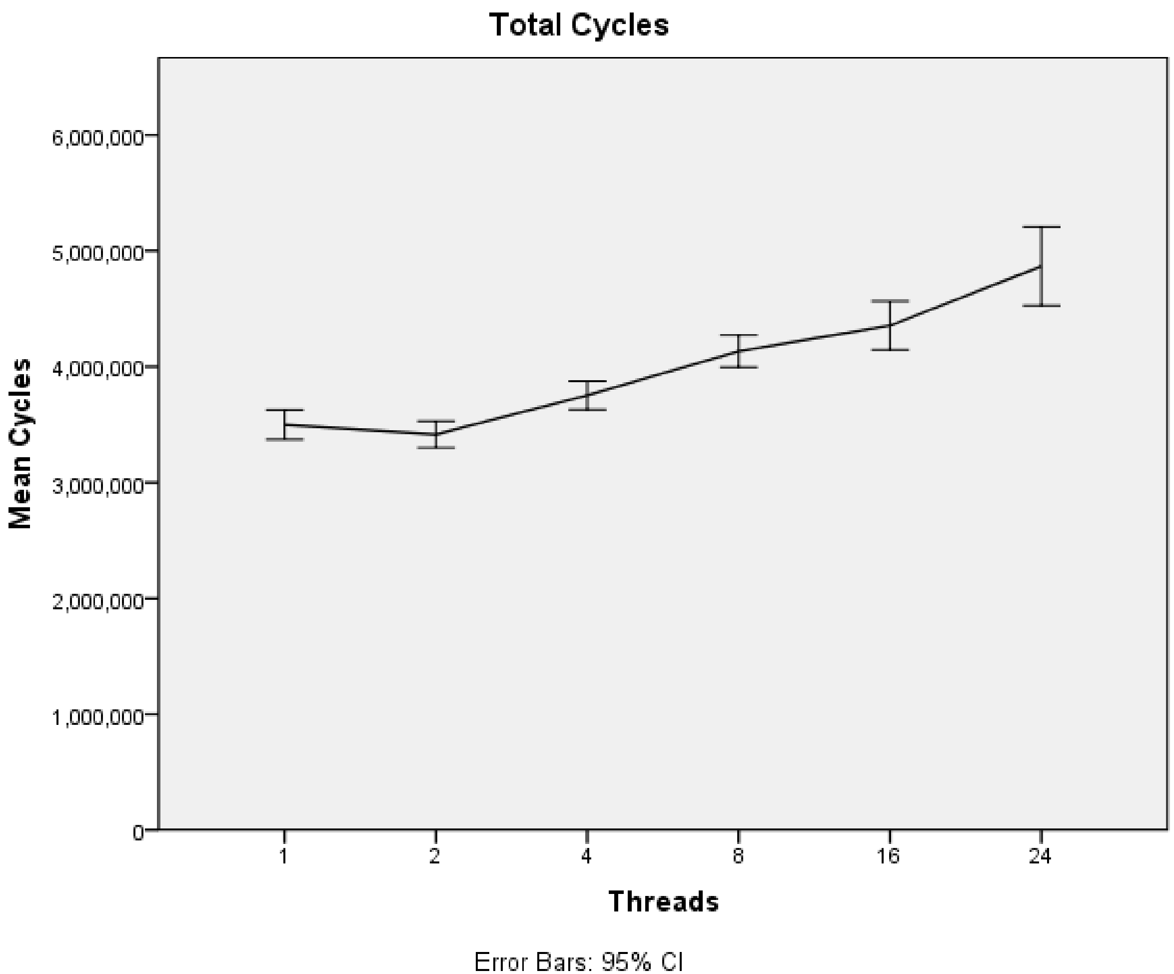


Figure3: Total Cycles consumed during the parallel part as number of threads increases

Results (Young Generation)

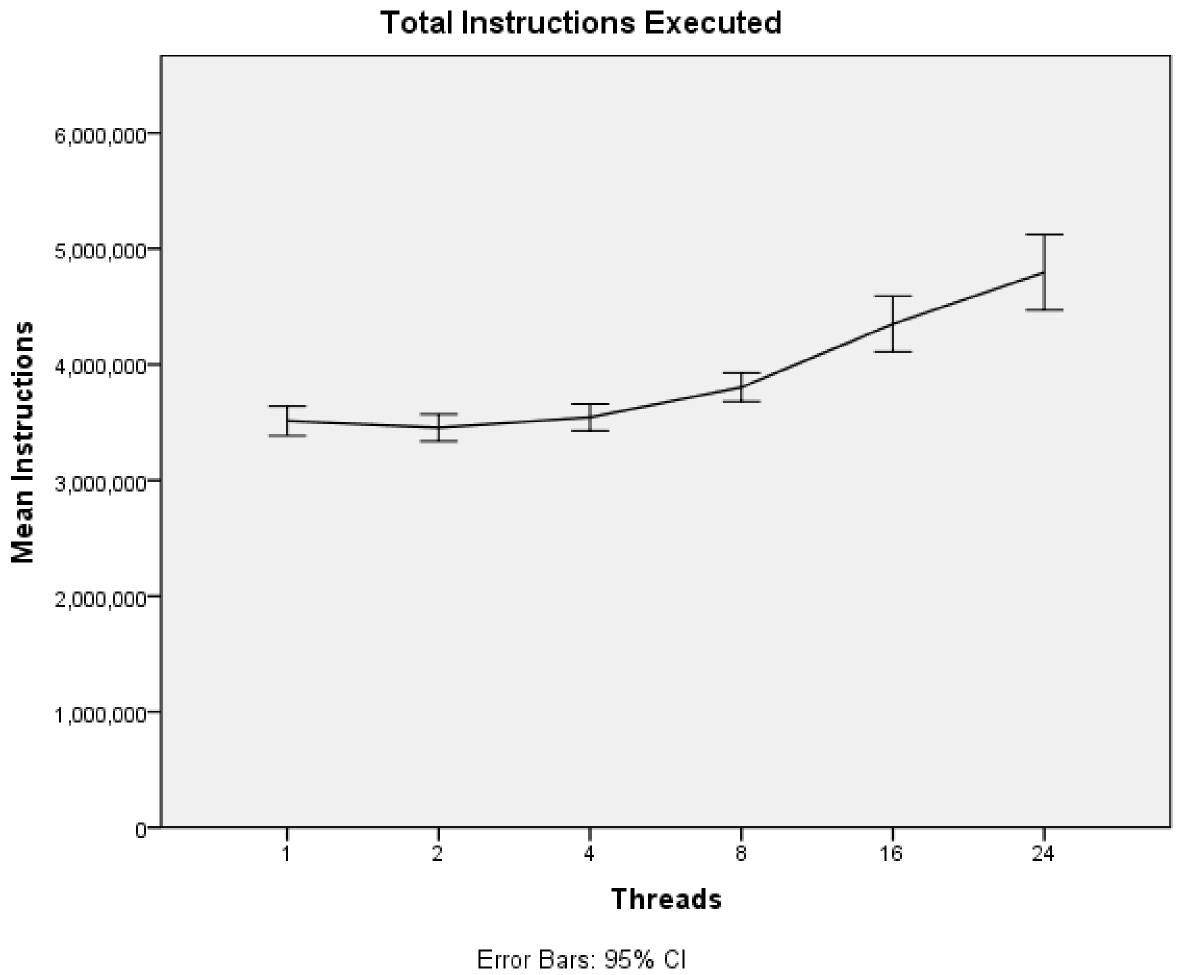


Figure4: Total Instructions executed per the number of threads

Results (Old Generation)

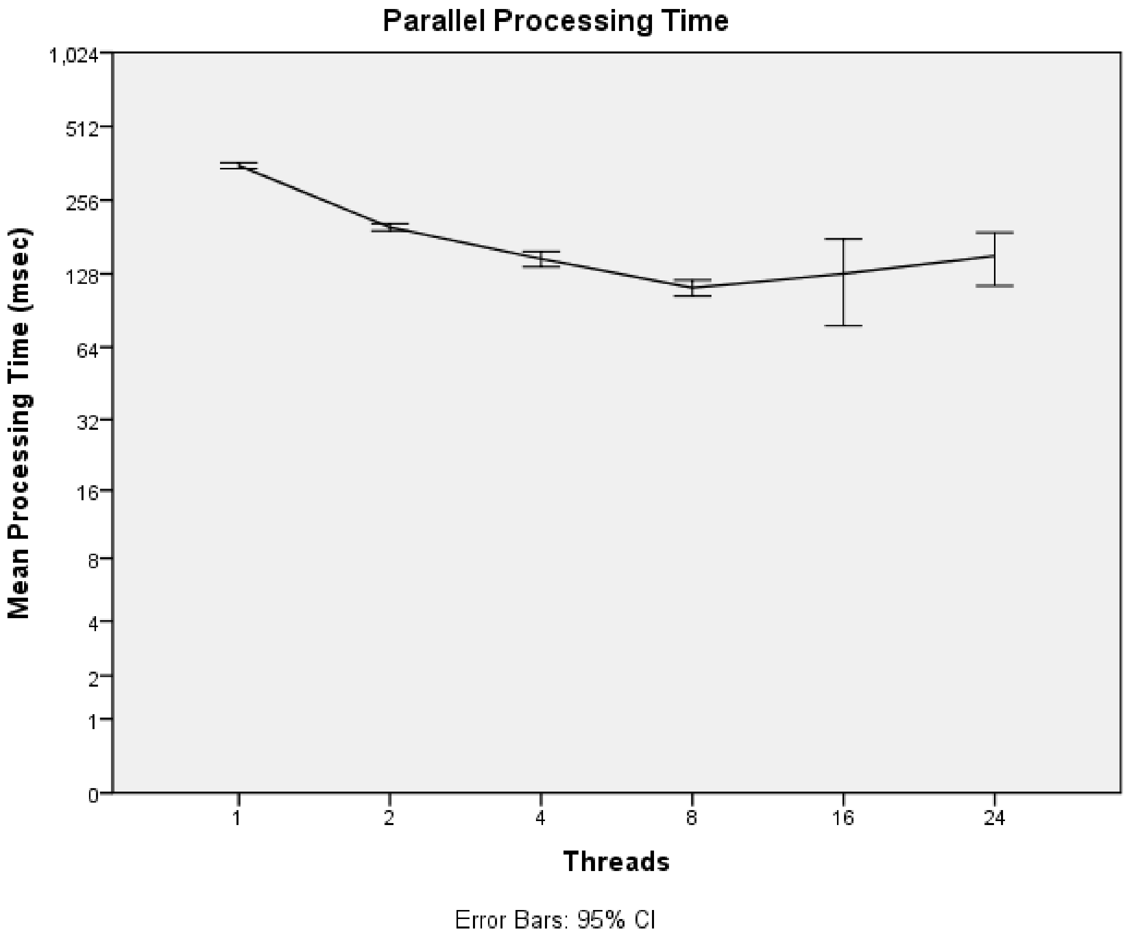


Figure5: Parallel GC time of major collection. as the number of GC threads increases

Results (Old Generation)

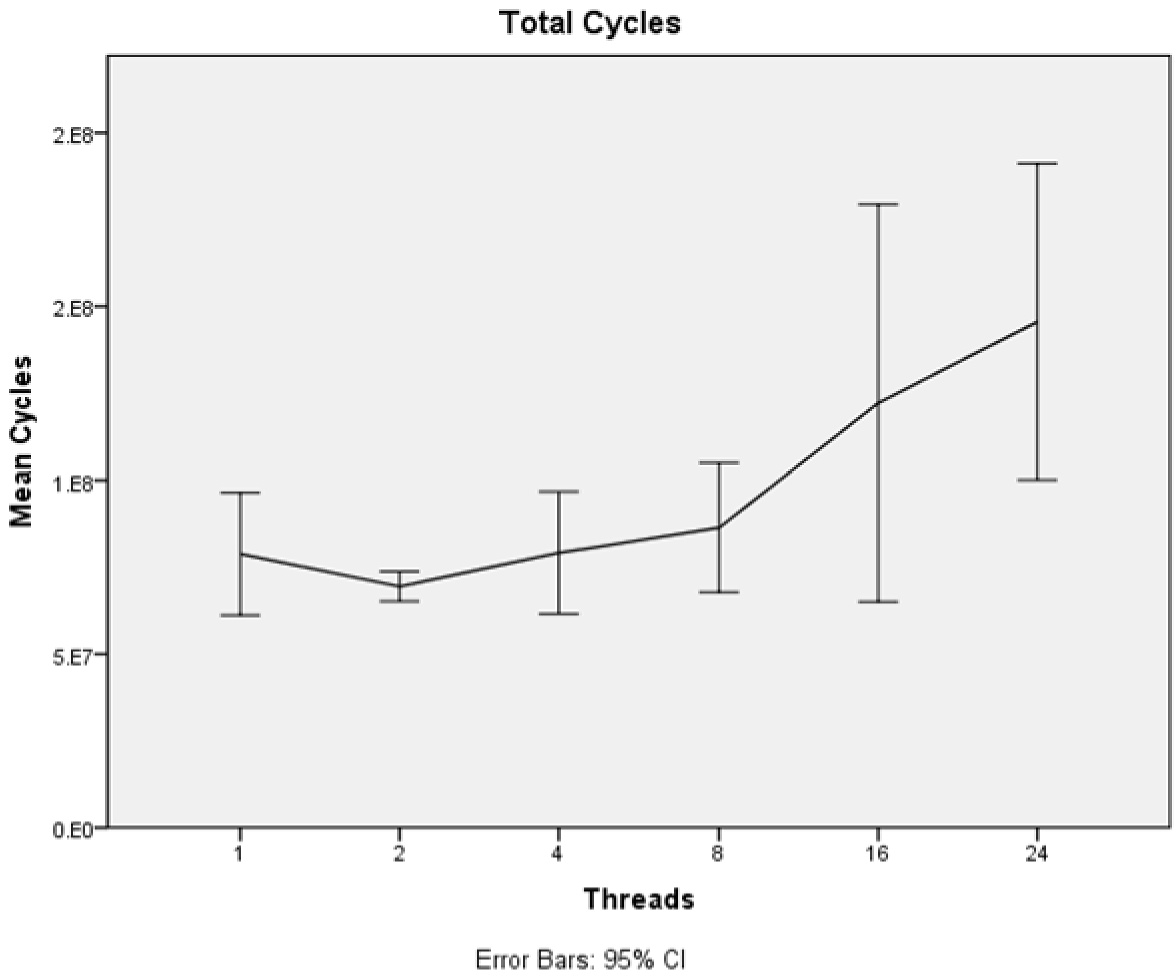


Figure6: Total Cycles consumed during the parallel part as number of threads increases

Results (Old Generation)

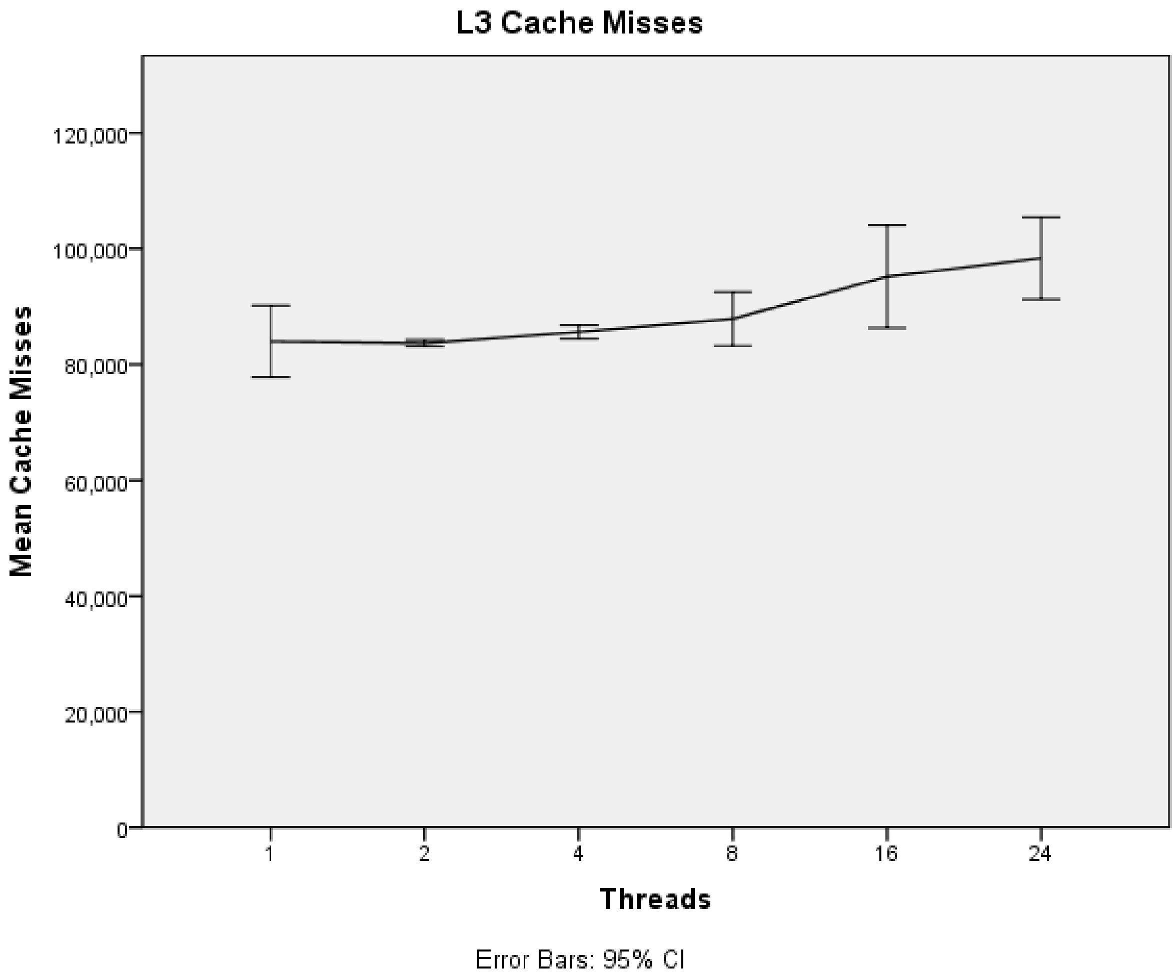


Figure7: L3 Cache misses per the number of threads

Results (Old Generation)

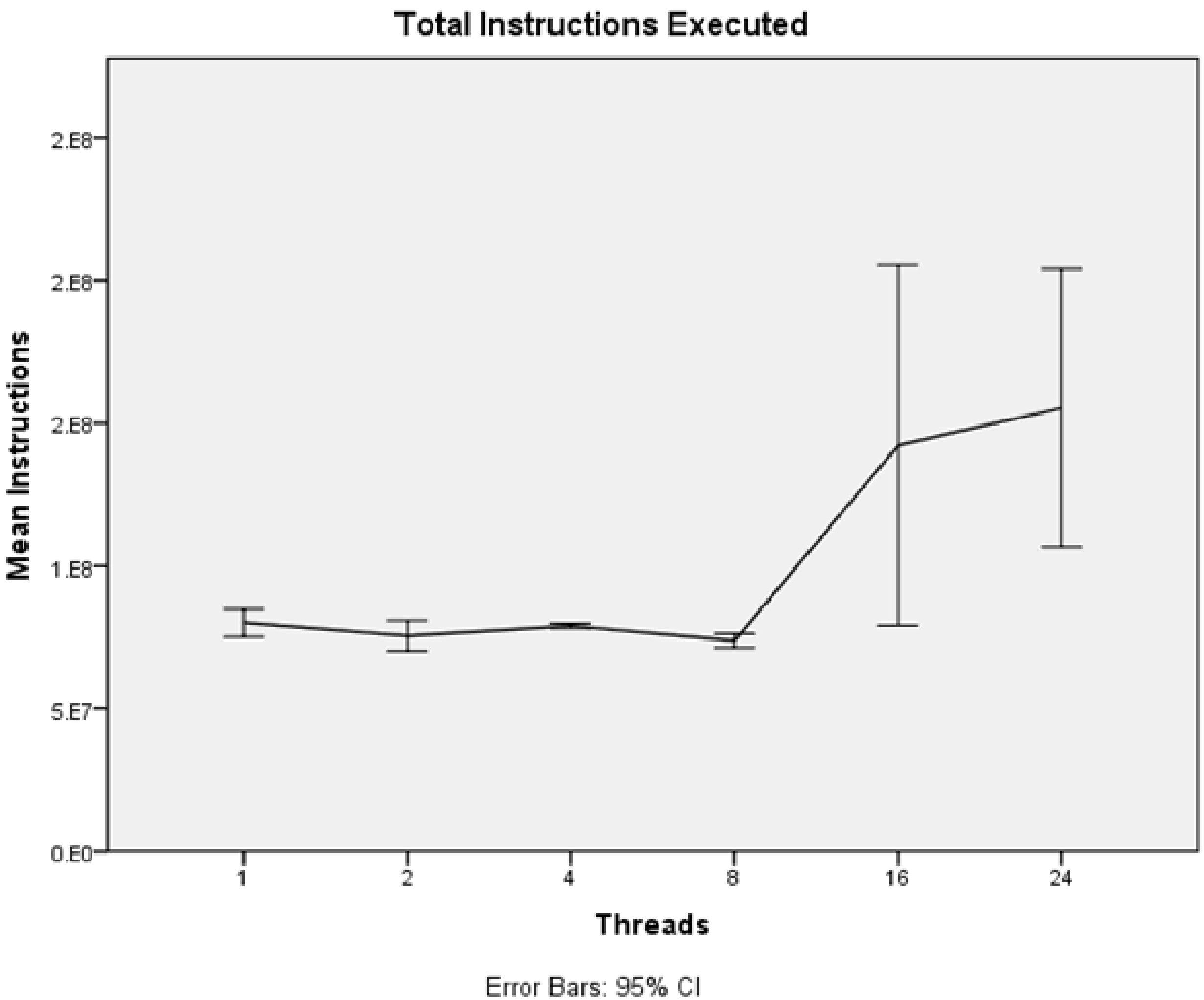


Figure8: Total Instructions executed per the number of threads

Potential Optimisations

1. **A scalability model for predicting the optimal number of threads of a JVM service.**

2. **Utilising the remaining threads as helper threads e.g. Cache prefetching**

3. **Thread management in the case of multiple JVM instances**

An adaptive policy for efficient multithreaded JVM services

References

- [1] Ting Cao et. al., The yin and yang of power and performance for asymmetric hardware and managed software, Proceedings of the 39th Annual International Symposium on Computer Architecture, p.225-236, June 09-13, 2012
- [2] Prasad A. Kulkarni., JIT compilation policy for modern machines. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11)
- [3] Lokesh Gidra, et. al. Assessing the scalability of garbage collectors on many cores. In *Proceedings of the 6th Workshop on Programming Languages and Operating Systems* (PLOS '11).
- [4] Jennifer B. Sartor et. al., Exploring multi-threaded Java application performance on multicore hardware. Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '12)