

Author: Jon Barnett
Date: 2 August 2003

1 Parallel thread performance: A visual aid

In the course of our study on parallel thread performance, there have been nearly as many questions about the comparison of the results as there have been about the results themselves. We have not sought to make direct comparisons between Java Virtual Machine (JVM) execution speeds. Our study has concentrated on the response characteristics. With this in mind, we provide the visual framework we used for interpreting the results.

1.1 Boundaries of operation

We have a simple experimental model. We execute the same task on multiple CPUs using the Java threading service. This has a reliance on the underlying operating system implementation for multi-threading and multi-processor execution. Going back to first principles, there are two limiting performance cases for parallel thread execution.

In the ideal case, given sufficient CPUs to execute each task independently, the time to execute n tasks on n CPUs should take the same time as executing the same task on one CPU.

At the other performance limit, the time to execute n tasks on the one CPU, in a serial fashion determines a boundary where additional CPUs executing tasks in parallel provide no value. We define this boundary to be a parity result to single CPU operation.

These boundaries, when provided on the same as the result graphs give a visual indication of the scale and tendency of the performance results in a JVM internally referential manner.

This can be further refined, using the ratio between the boundaries for any particular response as a measure of the tendency between ideal and parity. However, we do not provide this and leave it as an exercise for the interested reader to develop.

1.2 Test environment

The test environment consists of an 8-way Intel multiprocessor system, running RedHat 9 with an NPTL implementation. Nothing else is running on the system except the standard Linux support processes. The CPU clock speed on the system is reported as 2.0 GHz. The bus clock speed is reported as 100.0 MHz. For the remote lookup and remote invocation tests, JBoss 3.2.0 runs within an IBM Java 2 SDK 1.4.1 build cxia32141-20030522 Java Virtual Machine (JVM) with `LD_ASSUME_KERNEL=2.2.5`. This JVM configuration provides a known performance characteristic without non-linear effects under normal conditions. This ensures that the client response is not excessively affected by server-side operations.

1.2.1 Java Virtual Machine environments

The following JVM client environments are tested:

- Sun Java 2 JDK 1.4.1 03 b02 in server mode
- Sun Java 2 JDK 1.4.2 b28 in server mode with standard GC settings
- IBM Java 2 SDK 1.4.1 build cxia32141-20030522 with `LD_ASSUME_KERNEL=2.2.5`
- Sun Java 2 JDK 1.4.2 b28 in server mode with `-XX:+UseNewParGC`
- Sun Java 2 JDK 1.4.2 b28 in server mode with `-XX:+UseNewParGC` and `-XX:+UseConcMarkSweepGC`

The `LD_ASSUME_KERNEL=2.2.5` setting instructs the IBM SDK to employ the standard Linux threading model rather than use NPTL.

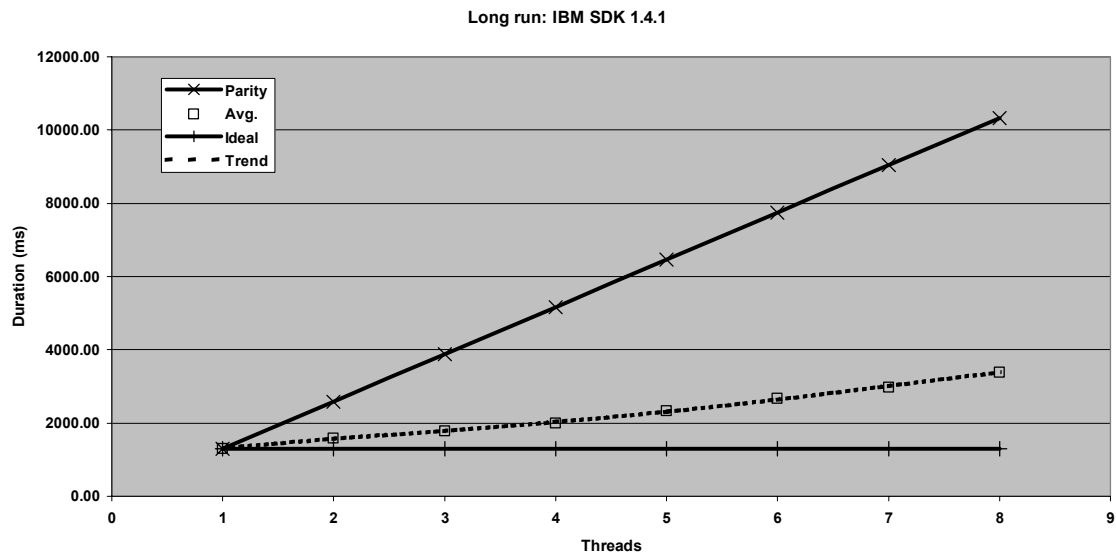
1.2.2 Result sources

The results used in this technical note have been taken from in [Java thread performance](#) and [Parallel thread performance: A JBoss client example](#).

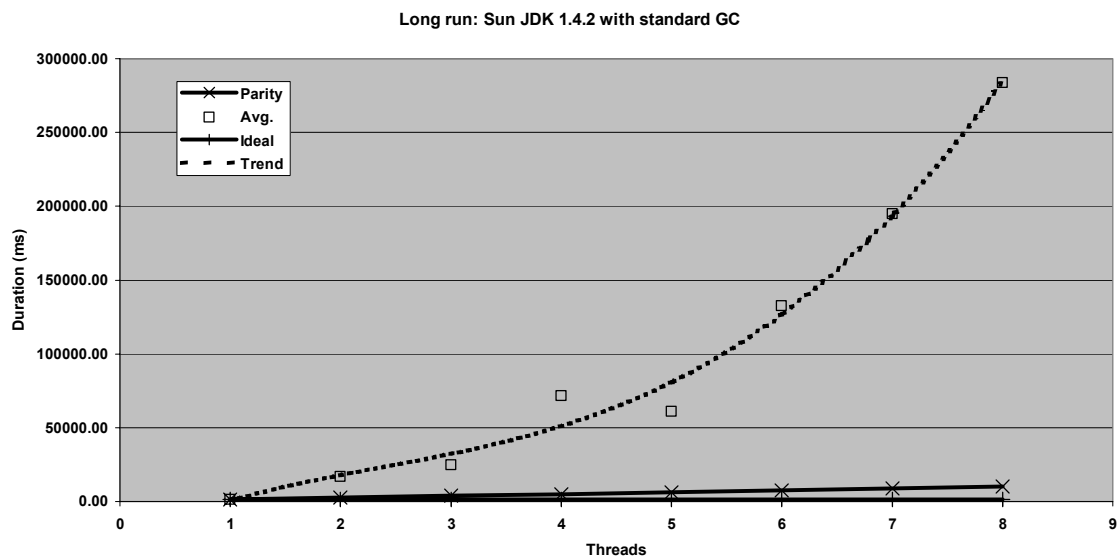
1.3 Long duration tests

These are the results obtained from tests where the 10000000 HashMap put operations were performed per thread.

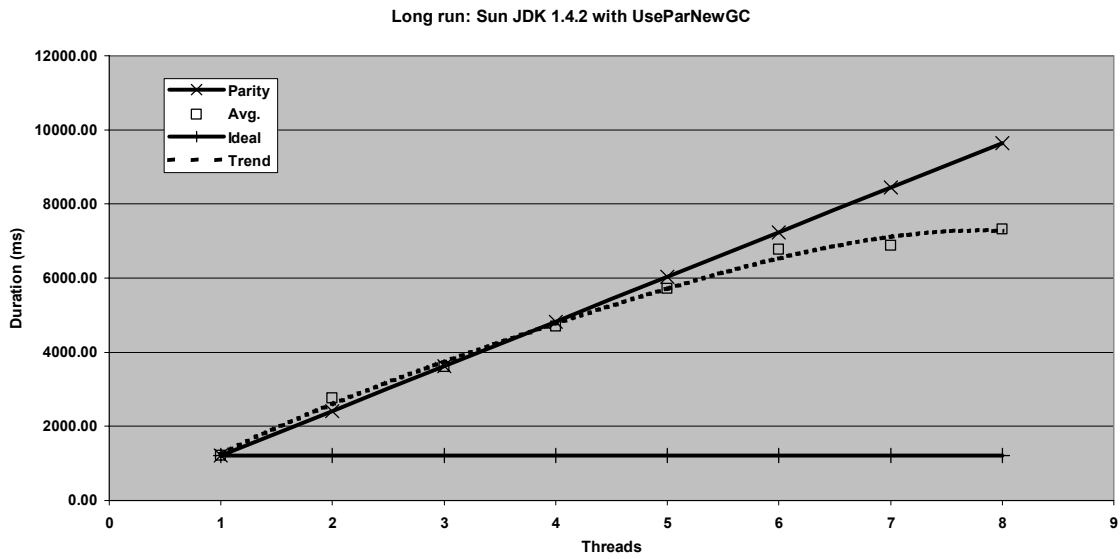
1.3.1 IBM SDK 1.4.1



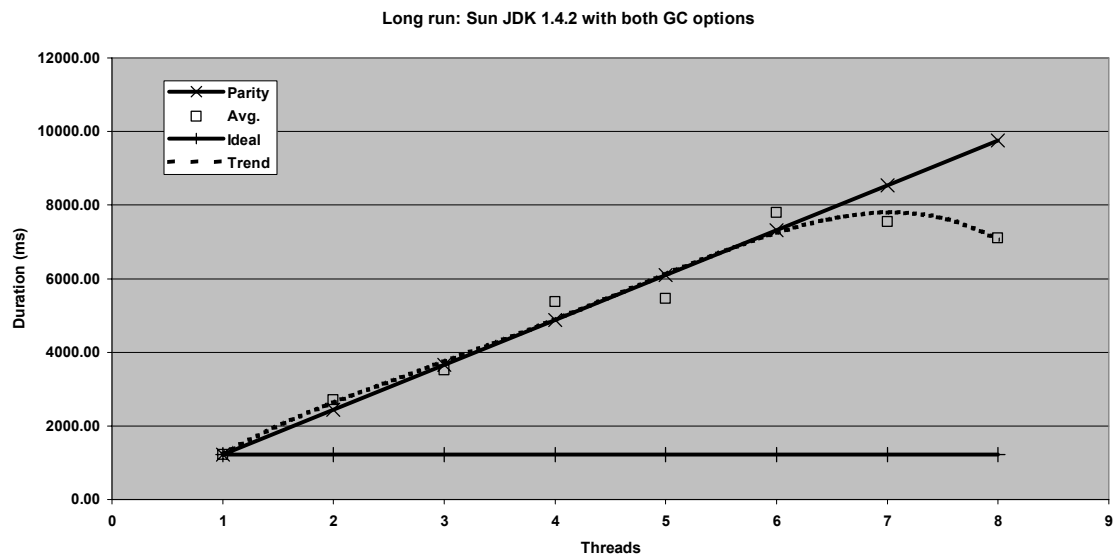
1.3.2 Sun JDK 1.4.2 with standard GC settings



1.3.3 Sun JDK 1.4.2 with UseParNewGC



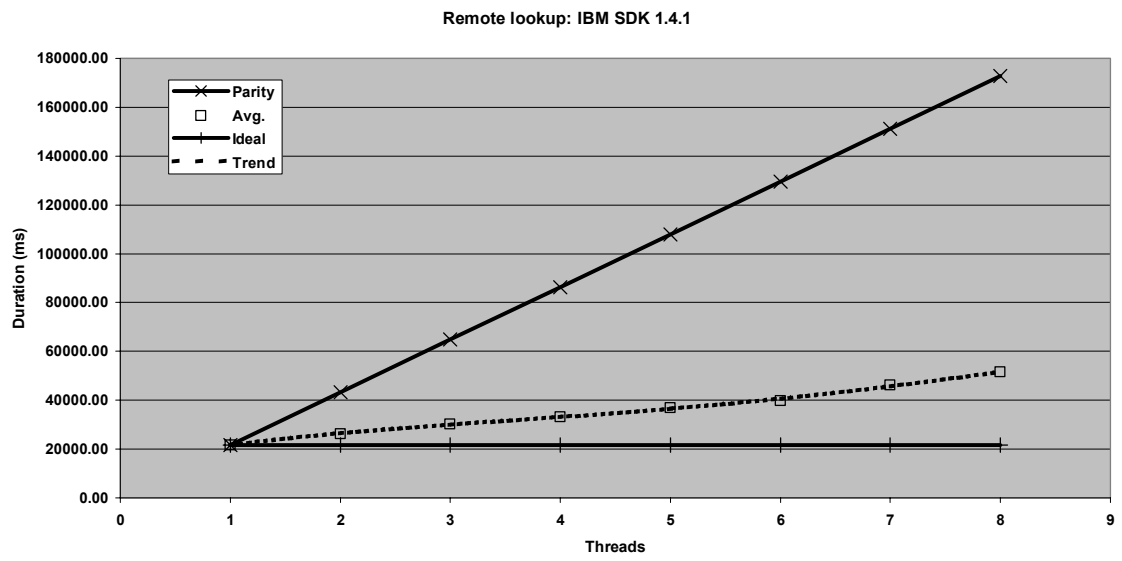
1.3.4 Sun JDK 1.4.2 with both new GC options



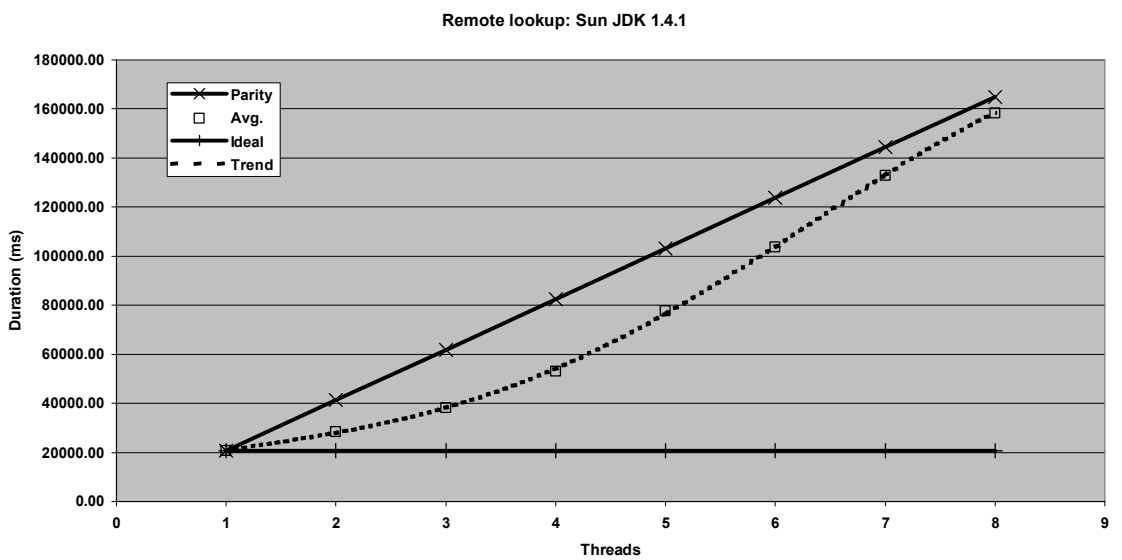
1.4 Name lookup test

This test creates client threads that continually request a name lookup from the JBoss 3.2.0 server. The duration for each thread is measured over 10000 requests.

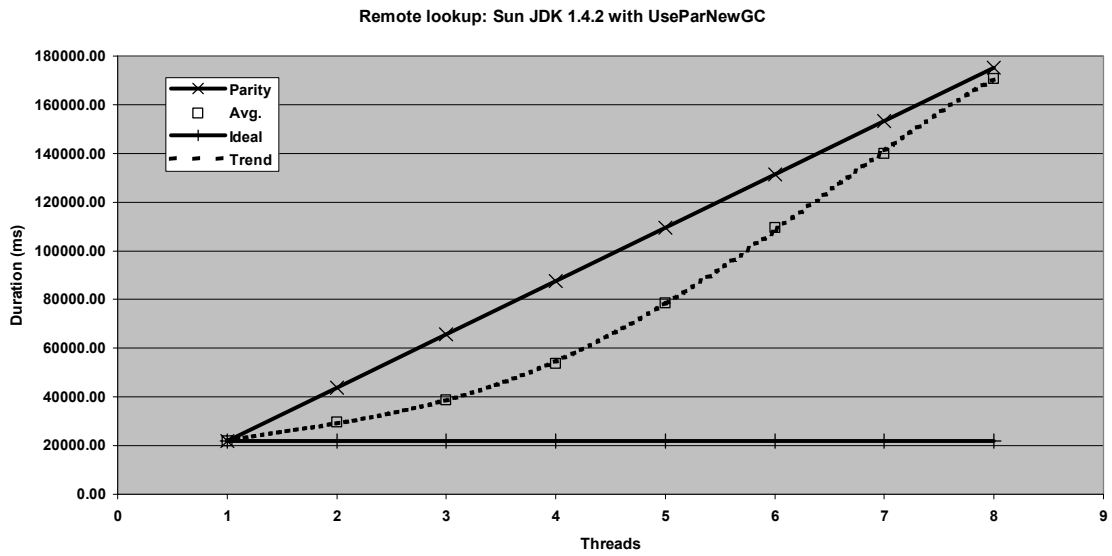
1.4.1 IBM SDK 1.4.1



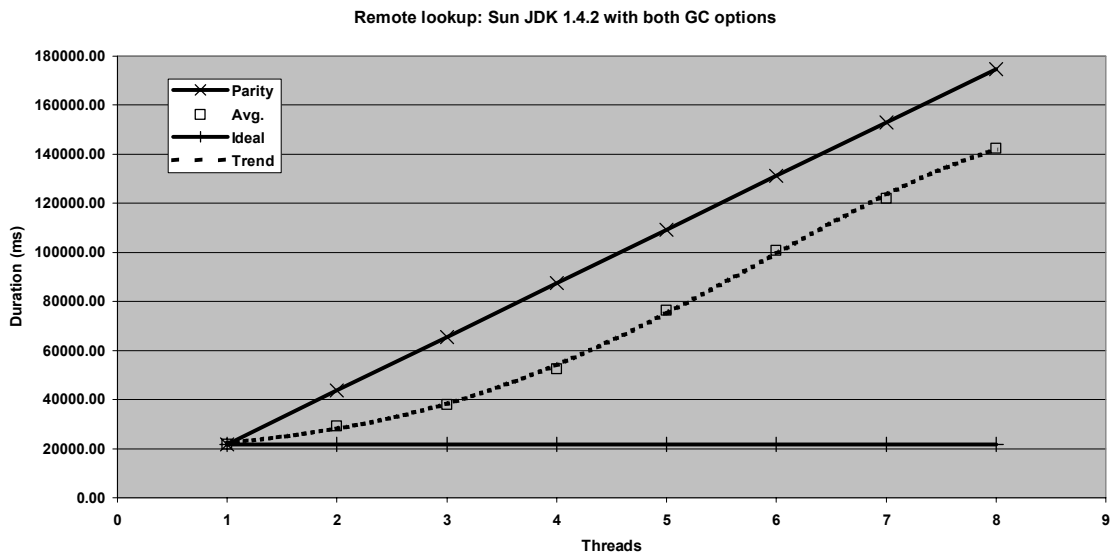
1.4.2 Sun JDK 1.4.1 with standard GC settings



1.4.3 Sun JDK 1.4.2 with UseParNewGC



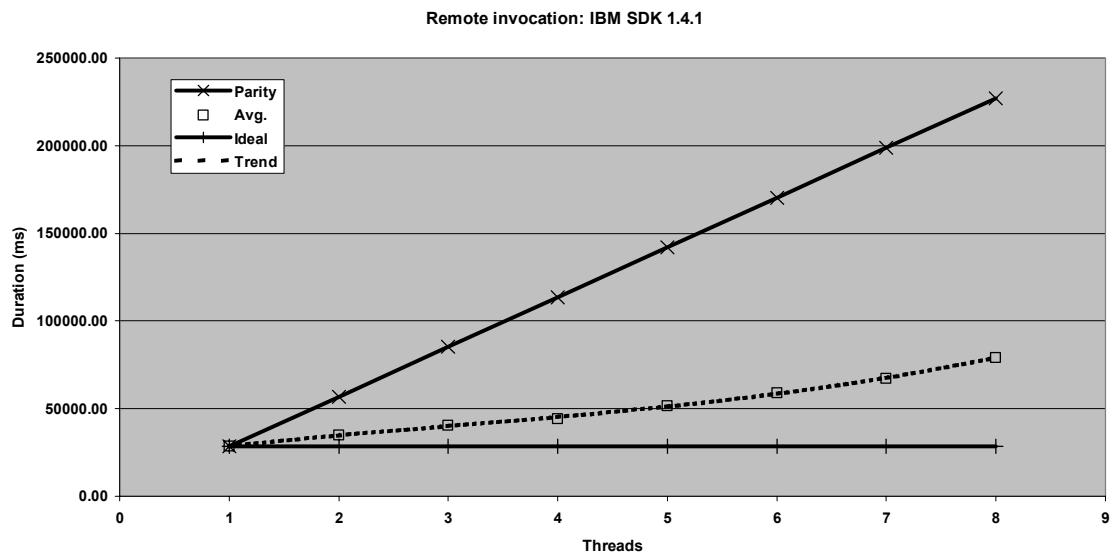
1.4.4 Sun JDK 1.4.2 with both new GC options



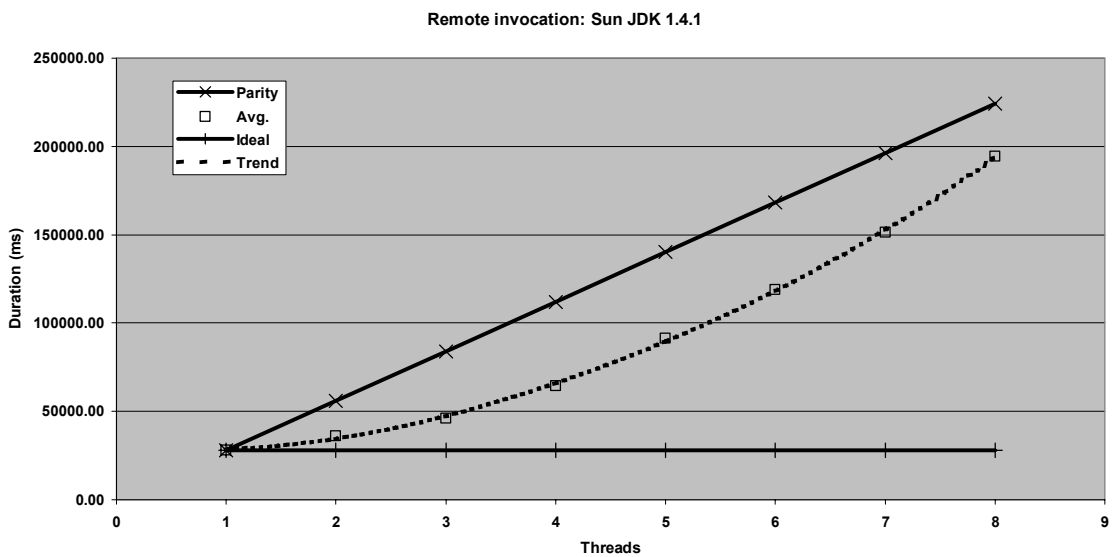
1.5 Remote invocation test

This test creates client threads that continually invoke a method from a stateless session bean on the JBoss 3.2.0 server. The duration for each thread is measured over 10000 requests. The naming server lookup is excluded from the measurement.

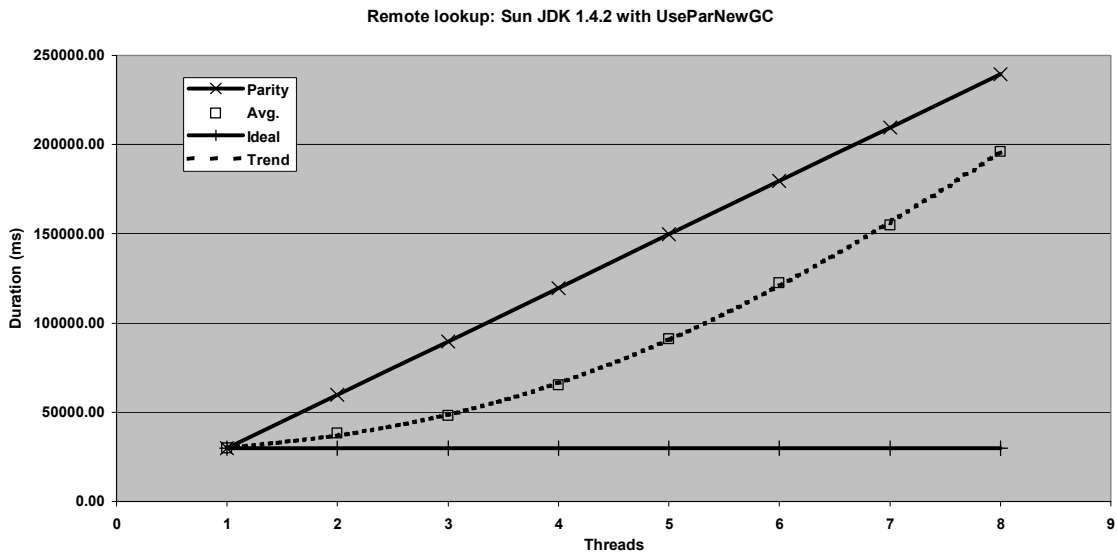
1.5.1 IBM SDK 1.4.1



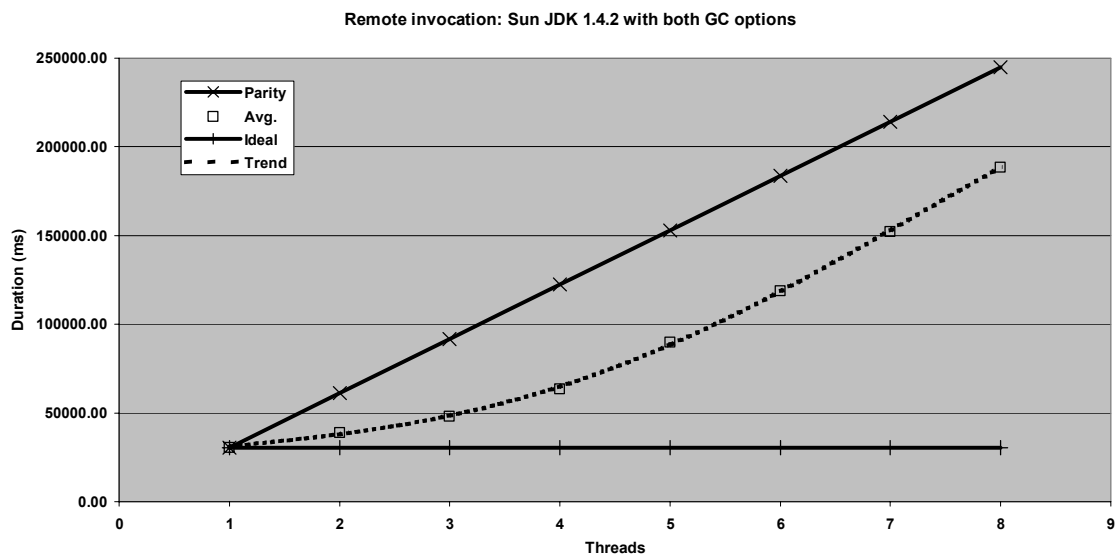
1.5.2 Sun JDK 1.4.1 with standard GC settings



1.5.3 Sun JDK 1.4.2 with UseParNewGC



1.5.4 Sun JDK 1.4.2 with both new GC options



1.6 Discussion

These limits of performance allow rapid visual interpretation of parallel thread performance. The observer can see the tendency of the response toward parity or toward ideal operation. The context of performance can also be isolated from the execution speed of tests.

Within the confines of the tests performed, it is possible to see the tendency of the Sun JDK performances towards parity or to operate at parity whereas the IBM SDK tends toward the ideal. It also shows the effect of changing JVM parameters relative to the performance boundaries. This has a significant impact in one case.

We offer no wider interpretation of the results as the test cases involve free-running parallel execution without synchronization interactions and other phenomena, and may therefore not be representative of performance in a production context.

The results do indicate that care should be taken to assess performance and the impact of adding parallel CPU power. Furthermore, attention should be paid to the tuning of the JVM for optimal Java server performance.