***Time Analysis***

The source code was ran on Linux Lab. The run time data provided below is from there:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Name of test file** | **Pages Parsed** | **Consumer threads** | **# of Processors** | **Average Execution Time** |
| 0 (Sequential) | few.xml | 5000 | 0 | 6 | 14340.67 milliseconds |
| v1 | few.xml | 5000 | 1 | 6 | 12447 milliseconds |
| v2 | few.xml | 5000 | 5 | 6 | 13859.67 milliseconds |
| v3 | few.xml | 5000 | 5 | 6 | 9523 milliseconds |
| v4 | few.xml | 5000 | 5 | 6 | 4703.67 milliseconds |

Sequential (Given): (14340ms + 14347ms + 14335ms)/3 = 14340.67ms

Version 1: (12443ms + 12457ms + 12441ms)/3 = 12447ms

Version 2: (13875ms + 13819ms + 13885ms)/3 = 13859.67ms

Version 3: (9572ms + 9531ms + 9466ms)/3 = 9523ms

Version 4: (4667ms + 4751ms + 4693ms)/3 = 4703.67ms

**Analysis:**

In the results, version 1 is faster than the given version provided by the professor which is sequential and version 2. Version 1 is faster than version 2 because version 2 has overhead involved in creating all of the consumer threads as well as switching between the threads and spending time waiting for access to the HashMap. Version 1 is faster than the given version because in version1 consumers can start processing pages as they come onto the queue, as opposed to having to wait for every page to be done before they can be processed like in the provided version.

Version 3 is faster than version 2 because in version 2, only one thread can access the HashMap at a time because the entire HashMap needs to be locked. In version 3, only the buckets of the ConcurrentHashMap need to be locked, so threads will not be blocked when they are writing to the ConcurrentHashMap so long as they are writing to different buckets.

Version 4 is the faster than Version 3 because threads are not being blocked on individual buckets of the shared map as much as they are in Version 3. Instead of having to lock each token’s bucket every single time it counts it, it only locks each token’s bucket once at the end, to add the token’s total count to the bucket. Because there isn’t as much blocking in this version, that makes it faster than Version 2 as well. It is also faster than Version 1 and the given version since it is making good use of multiple threads as opposed to only using one producer or consumer thread or just one thread overall.