Analyzing the Impact of Threading on Hashing Performance Across Varying File Sizes

This report investigates the effect of multi-threading on the computational efficiency of hashing large files, with file sizes ranging from 256 MB to 4 GB. Utilizing a controlled experimental setup on a high-performance computing system equipped with 48 CPUs, the study measures the time taken and speedup achieved by incrementally increasing the number of threads from 1 to 256. The results indicate that while threading significantly reduces computation time initially, the gains diminish with higher thread counts, particularly for larger files.

Experimental Setup

Hardware Specifications

- **CPU**: Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
- **Core Configuration**: 24 cores across 2 sockets with hyper-threading enabled, allowing up to 48 threads.
- **Thread Counts**: Tests conducted with 1, 4, 16, 64, 128, and 256 threads.
- **Data Collection**: Execution time and hash values were logged for each run, with three repetitions per configuration to ensure statistical relevance.

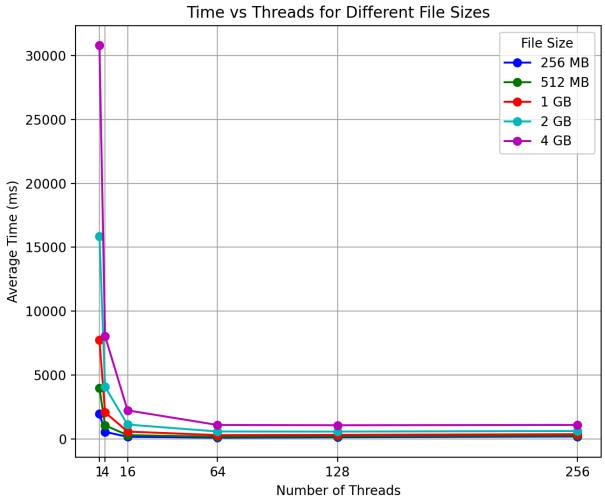
Procedure:

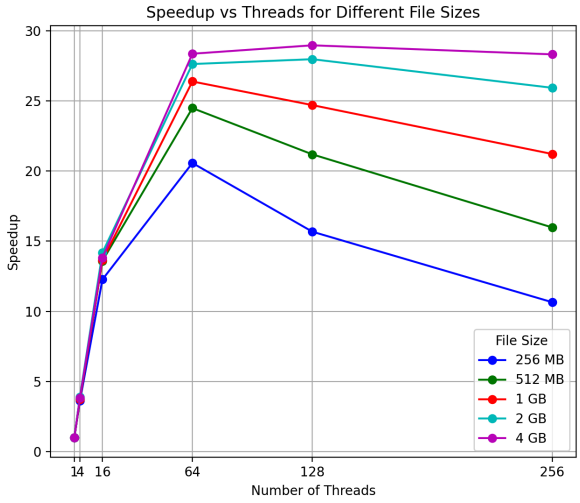
- Each file was processed using the Jenkins one-at-a-time hash algorithm, with the data loaded into memory using `mmap` to eliminate I/O overhead during computation.

- Performance metrics such as time to complete hashing and system load were monitored using system tools and custom logging mechanisms.

Results

The results from the experiments are presented below in two key formats: time taken and speedup graphs for varying thread counts across different file sizes.





Time Analysis:

- **256 MB to 4 GB files**: The graphs reveal a reduction in computation time as threads increase from 1 to 64, with the most significant decrease observed in larger files.
- **Thread saturation**: Beyond 64 threads, the reduction in time either plateaued or slightly increased. This was most evident in the smallest file size of 256 MB, where thread overhead likely negated some of the computational benefits.

Speedup Analysis:

- **General trend**: There was a clear increase in speedup as the number of threads increased from 1 to 64, demonstrating the benefits of parallel processing.
- **Diminishing returns**: For all file sizes, speedup gains diminished and sometimes decreased as threads increased beyond the optimal count (around

64), especially for smaller files.

Interpretation of Results:

- **Efficiency of threading**: The initial increase in speedup with more threads validates the effectiveness of parallel processing in hashing tasks. However, the leveling off and decline at higher thread counts point to limitations due to thread management overhead.
- **Impact of file size**: Larger files showed the greatest benefit when increasing the thread count. As you can see with the steep drop in time from 1 thread to 4 threads for the 4GB file. Conversely, smaller file sizes exhibit minimal improvements and reach a plateau in benefits more quickly, suggesting that the overhead associated with managing multiple threads over smaller data sets can substantially offset performance gains.
- **Optimal thread count**: The results suggest an optimal thread count close to 64 for the given hardware setup, aligning with the physical core count of the system.