Final Project: Parallel Web Scraping System

Wenlue Zhong

May 24, 2024

1 Program Description and Problem Statement

1.1 Overview

The objective of this project is to develop a parallel web scraping system to scrape product data from an e-commerce demo site (`https://www.scrapingcourse.com/ecommerce`). The system is designed to demonstrate the benefits of parallelization by implementing three versions of the scraper:

1. Sequential Implementation
2. Parallel Implementation using Channels
3. Work-Stealing Implementation

The website contains 12 pages of items, with 16 items per page, resulting in a total of 188 products to be scraped.

1.2 Problem Statement

The primary goal is to compare the performance of the sequential and parallel implementations and evaluate the improvements in speed and efficiency brought about by parallelization.

2 Implementation Details

2.1 Sequential Implementation

The sequential implementation processes each URL one at a time. It fetches the HTML content of each page, parses the product data, and downloads the images sequentially. This version is run using the `--seq` flag.

2.2 Parallel Implementation using Channels

The parallel implementation using channels employs goroutines and channels to distribute tasks among multiple workers. Each worker fetches and processes URLs concurrently. Image downloading is also done in parallel within each worker. This version is run using the `--chan` flag.

2.3 Work-Stealing Implementation

The work-stealing implementation uses a dequeue-based task queue. Workers pick tasks from their own queues and can steal tasks from other workers' queues when idle. This ensures better load balancing and reduces idle time. This version is run using the `--workstealing` flag.

2.4 Usage Statement

The program can be executed with the following commands:

- `python benchmark/benchmark.py`: Runs the benchmark script to reproduce the results. Each implementation (sequential, parallel channel-based, and work-stealing) is run for 10 times, and the average execution time is calculated for performance analysis.

For examination of specific implementations:

- `./scraper --seq`: Runs the sequential implementation.
- `./scraper --chan --threads <numThreads>`: Runs the parallel implementation using channels with the specified number of threads.
- `./scraper --workstealing --threads <numThreads>`: Runs the work-stealing implementation with the specified number of threads.

3 Challenges Faced

3.1 Load Balancing

Ensuring that all workers are utilized efficiently was a challenge, particularly in the channel-based parallel implementation. The work-stealing approach mitigated this by allowing idle workers to steal tasks from others.

3.2 Synchronization

Managing synchronization without using the `sync` package in the channel-based implementation required careful design to avoid race conditions and ensure that all tasks were completed.

3.3 Error Handling

Handling errors in a parallel environment, such as network failures or parsing errors, required robust error handling mechanisms to ensure that the system remained stable and did not miss any tasks.

4 Performance Analysis

4.1 Speedup Graphs

The speedup graph illustrates the performance improvements of the parallel implementations relative to the sequential implementation. The experiments were conducted with thread counts of 2, 4, 6, 8, and 12.

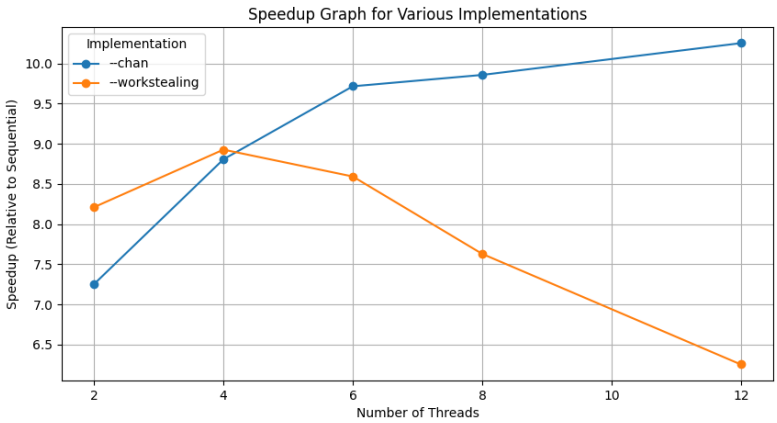

Figure 1: Speedup graph for various implementations for Parallel vs. Sequential

4.2 Observations

- The channel-based implementation showed significant speedup up to 10 threads, beyond which performance gains plateaued.
- The work-stealing implementation exhibited a more consistent speedup but did not scale as well with higher thread counts compared to the channel-based implementation.

4.3 Task Queue with Work-Stealing Performance

The usage of a task queue with work-stealing did improve performance, especially in terms of load balancing. By allowing workers to steal tasks from others when idle, the system ensured that all available processing power was utilized

efficiently. This reduced the time workers spent waiting for tasks, which is particularly beneficial when tasks have varying execution times.

However, the work-stealing implementation did not achieve the same level of speedup as the channel-based implementation with higher thread counts. The primary reasons for this include:

- **Stealing Overhead**: The overhead associated with task stealing, such as the time taken to check other workers' queues and the synchronization required to safely steal tasks, can limit the speedup.
- **Task Granularity**: The granularity of tasks affects the efficiency of work-stealing. If tasks are too small, the overhead of stealing can outweigh the benefits. If tasks are too large, some workers may remain idle while others are still processing.
- **Communication Overhead**: In work-stealing, workers need to communicate more frequently to check and steal tasks, which can introduce additional delays.

5 System Specifications

5.1 Hardware Details

- **Model Name**: MacBook Pro
- **Model Identifier**: Mac14,10
- **Model Number**: Z174000E7LL/A
- **Chip**: Apple M2 Pro
- **Total Number of Cores**: 12 (8 performance and 4 efficiency)
- **Memory**: 16 GB
- **System Firmware Version**: 10151.61.4
- **OS Loader Version**: 10151.61.4
- **Serial Number (system)**: G2KM6VWJLW
- **Hardware UUID**: 535338E8-6EC6-5FA2-84A2-58E5694D9147
- **Provisioning UDID**: 00006020-000629821A6B401E
- **Activation Lock Status**: Disabled

6 Hotspots and Bottlenecks

6.1 Hotspots

The primary hotspots in the sequential implementation were:

- Fetching HTML content
- Parsing product data
- Downloading images

6.2 Bottlenecks

The main bottleneck was the sequential processing of tasks, which limited the overall speed and efficiency. In the parallel versions, synchronization overhead and task distribution were minor bottlenecks.

7 Limitations and Conclusions

7.1 Speedup Limitations

The speedup was limited by:

- Communication and synchronization overhead in the channel-based implementation.
- Stealing overhead and task granularity in the work-stealing implementation.

7.2 Conclusions

The work-stealing implementation provided better load balancing and more consistent performance across varying thread counts. However, the channel-based implementation achieved higher speedup with fewer threads. Both parallel implementations significantly outperformed the sequential implementation, demonstrating the benefits of parallelization for web scraping tasks.

8 Data and Code Submission

All files, including the code, dataset, benchmark script, and results, are organized and included in the submission. The benchmark results and speedup graph can be found in the `benchmark` directory.