Evan Magee
Professor Jang
CmpSc 472 - 001
11/3/2025

Project Description

This project aims to explore the concepts of parallelism and synchronization in operating systems using two MapReduce-inspired implementations: Parallel Sorting and Maximum Value Search. These tasks were implemented twice using a multithreaded approach and a multiprocessing approach. This was done with a goal to compare their performance and memory usages on varying levels of parallelism, testing with 1, 2, 4, and 8 workers. The main goal of this project is to evaluate how threads and processes handle computation and memory differently when using them for parallel operations. To measure, execution time and memory usage are analyzed across the different worker counts. These results can then highlight the different advantages and disadvantages of the approaches, helping to provide insights into how synchronization, worker count, and memory sharing all impact performance in parallel systems.

How to Use

This project runs using different C files depending on the desired implementation. It also must be run on a Linux based system since functions like fork() are not natively supported by OS's like windows. Within the github there is a link to a Google Colab which allows for the running of each file. Each file does require itself to be run for the C file to be written, then underneath that section is a gcc command which will compile and run the file. Running the files will automatically produce an array for testing with them all being seeded at 42. This means it will allow for easier comparison between each of the files' performance. Thread and process information will be output on top of performance and memory info underneath the running code sections.

Structure of Code

The project composes of four main C files, each consisting of a different implementation style and task:

- ***parallel_sort_multithreading.c***
    - Multithreaded parallel sorting implementation
- ***parallel_sort_multitprocessing.c***
    - Multiprocessing parallel sorting implementation
- ***max_value_multithreading.c***
    - Multithreaded maximum value aggregation implementation
- ***max_value_multiprocessing.c***
    - Multiprocessing-based maximum value aggregation implementation

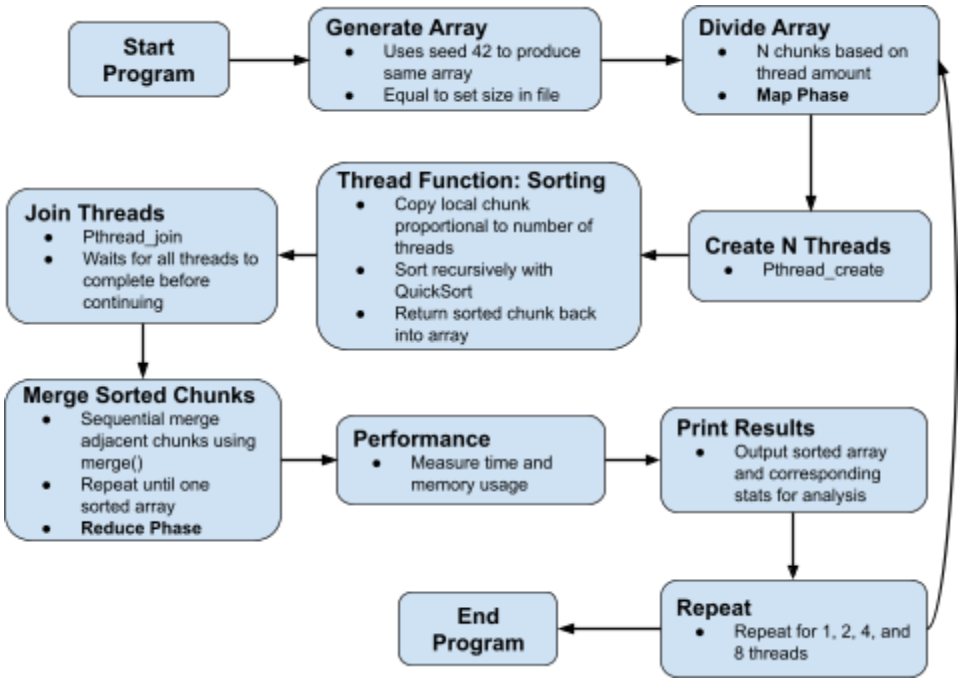
Each of these files consists of sorted sections of code which consist of methods like sorting, getting memory usage, and finding the max value.The sections differ per file but all contain similar main methods and the same memory usage measuring methods.

- ***Shared Helpers***
    - long get_memory_usage()
        - This function opens and reads the "/proc/self/status" and scans for the “VmRSS” value which is the current memory usage in kb. It then returns this value as a long and closes the file.
- ***parallel_sort_multithreading.c***
    - void quickSort(int *array, int low, int high)
        - This function implements a recursive QuickSort algorithm type which is used by workers to sort their respective chunks independently
    - void* chunk_sorting(void* arg)
        - This function helps determine a workers/threads array segment, copies that segment into a local temporary array, calls quickSort and writes sorted local arrays back into the main array.
    - void merge(int *array, int low, int mid, int high)
        - Sequentially merges the chunks sorted by the threads back into a large fully sorted array
- ***parallel_sort_multitprocessing.c***
    - void quickSort(int *array, int low, int high)
        - This function implements a recursive QuickSort algorithm type which is used by workers to sort their respective chunks independently
    - void merge(int *array, int low, int mid, int high)
        - Sequentially merges the chunks sorted by the threads back into a large fully sorted array
- ***max_value_multithreading.c***
    - void* find_local_max(void* arg)
        - This function takes an input thread id, finds its designated array segment, and finds its local maximum value. This maximum value is then compared to the global maximum while under mutex protection to ensure safe updating.
- ***max_value_multiprocessing.c***
    - void find_local_max(int id)
        - This function takes an input process id, finds its designated array segment, and finds its local maximum value. This maximum value is then compared to the global maximum while using a shared

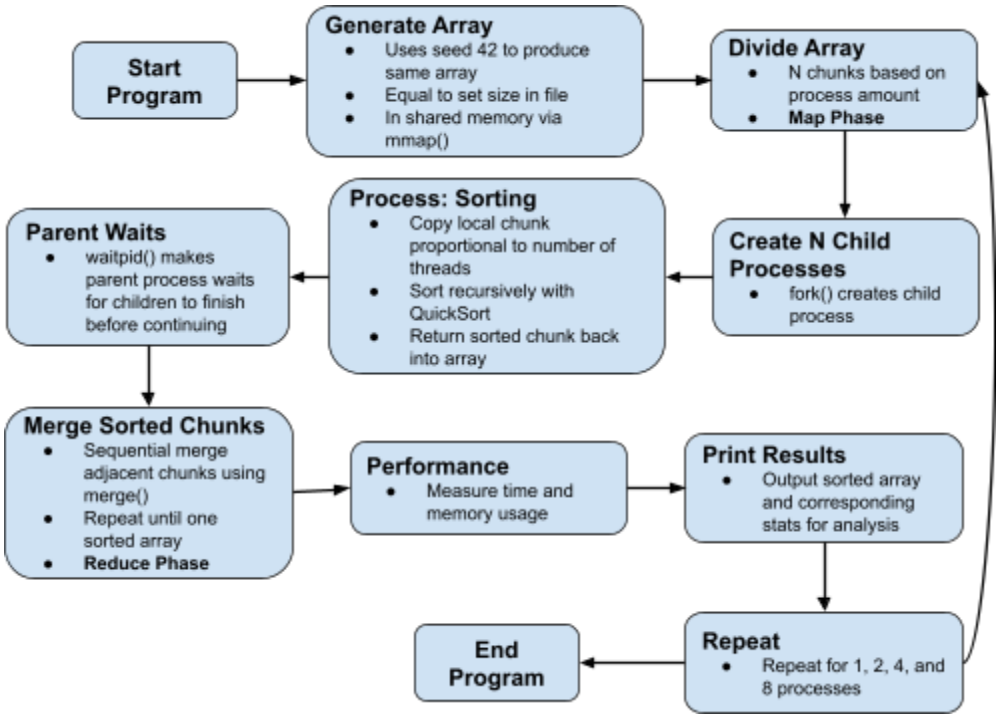
semaphore to safely update. It also records memory usage before exiting

Flowcharts

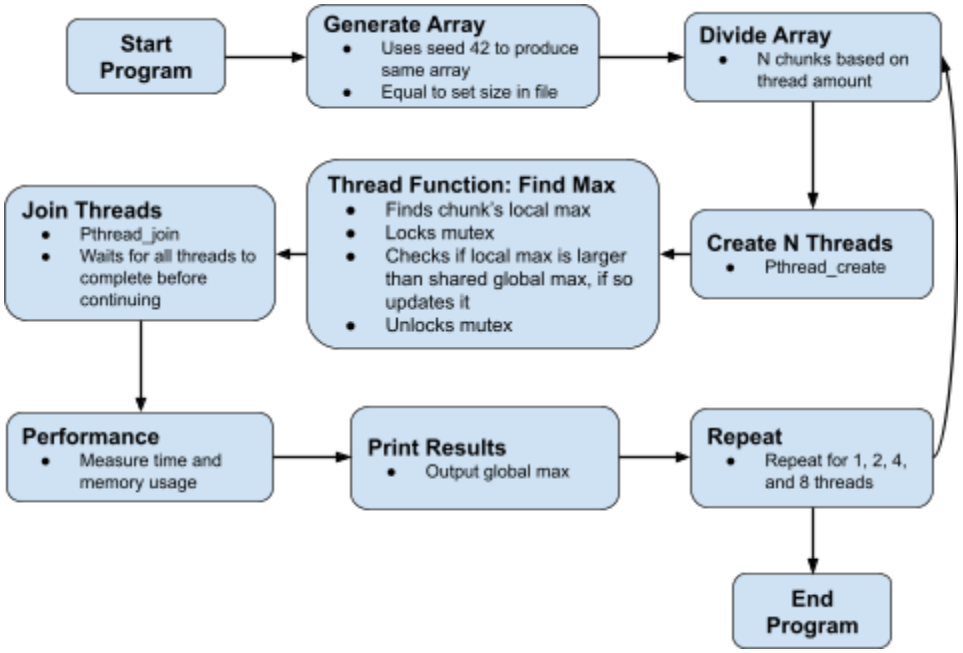
- Multithreading Parallel Sort



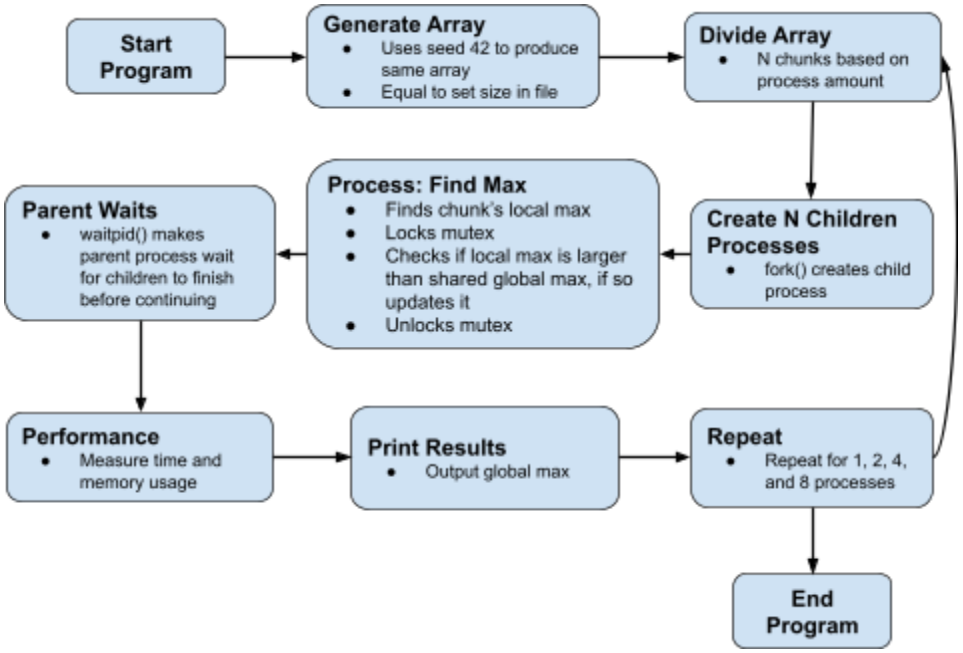
- Multiprocessing Parallel Sort



- Multithreading Max-Value Aggregation



- Multiprocessing Max-Value Aggregation



Support of MapReduce Framework

Each of these programs follow the structure of the MapReduce framework, containing both a mapping and reducing section. The mapping section for each takes the input array and correctly divides this array into chunks equal to the number of workers (threads or processes). These chunks are all equal sizes and are worked on by their respective workers. After all chunks have been worked on by their worker, they are merged back together into a single output which represents the reduce section. Shared memory and synchronization also enable these sections to communicate with each other, similar to MapReduce.

Implementation

Libraries

- <stdio.h>
    - This library provides standard input/output functions including printf() and perror().
- <stdlib.h>
    - This library provides functions for memory management and random values.
- <string.h>
    - This library provides string manipulation and comparison functions and is mainly used to help collect memory usage from the “/proc/self/status” file.
- <unistd.h>
    - This library provides functions for POSIX operating system API. This includes functions like fork() to create child processes and getpid() for printing the process IDs.
- <sys/types.h>
    - This library provides the pid_t data type which is used to represent a process ID on Unix systems.
- <sys/wait.h>
    - This library provides functions for waiting on child processes to finish.
- <sys/mman.h>
    - This library provides memory management functions for POSIX systems. This includes mmap() which is used for creating shared memory regions.
- <semaphore.h>
    - This library provides POSIX semaphores for synchronization. This is used to ensure one process or thread can update a shared global variable at a time.
- <time.h>

    - This library provides functions for measuring time and timestamps which are used to measure execution time.
- <pthread.h>
    - This library provides functions for thread creation, joining, and mutual exclusion.

Process Management

Workers are spawned using the function fork() which creates a child process to do a specific task. These processes are assigned a chunk of data which is an equal-sized piece of the main array. Each process then computes their assigned task of either sorting or finding the maximum in its assigned chunk. waitpid() is used to ensure that all processes complete before it moves on to the next step of the program. After each process is finished, _exit(0) is called to prevent duplicate code execution and cleanly disband the child process.

IPC Mechanism

Inter-Process Communication between the works is handled through the usage of shared memory. This shared memory is created using the mmap() which creates a shared region accessible by all the processes or threads. This approach was chosen because shared memory is the fastest IPC method and avoids the overhead methods like message passing. It was also easier to implement for these programs.

- Shared Variables
    - array
        - Dataset to be processed
    - global_max
        - Integer used as shared output for max-value aggregation
    - mutex
        - POSIX semaphore used to synchronize updates to global_max

Threading

Implementation of the multithreaded programs was done using threads manually created using pthreads or POSIX API. The threads were created using the pthread_create() command and joined using pthread_join(). These threads are then given their own chunk of the array to process and will wait for all other threads to finish before moving to the next tasks. The pthread mutex locks are also used to protect shared data like global_max to prevent race conditions from occurring. These threads also share the same memory space which allows for a lower memory overhead than multiprocessing and for faster context switching.

Synchronization

The synchronization strategies used were different depending on if the program is multithreading or multiprocessing. For multithreading, pthread_mutex_lock() and pthread_mutex_unlock() are used while for multiprocessing, a semaphore is used with the sem_wait(mutex) and sem_post(mutex) commands. These help to prevent race conditions when a thread or process is trying to update shared data like global_max for example. It ensures that only one thread or process can modify shared data at a time, thus maintaining data consistency.

Performance Evaluation Implementation

Performance is measured using 2 main metrics, execution time and memory usage. Execution time is calculated by getting the clock time before and after the multithreading or multiprocessing to explore the performance of those pieces specifically. Execution time is best used for comparing the number of workers and how that speeds up or slows down the program. Memory usage focuses more on viewing the different overheads of multiprocessing and multithreading to determine which is better for what tasks. Memory usage is retrieved from scanning the /proc/self/status file. Together, these measurements provide insight into the trade-offs between multiprocessing and multithreading.

Performance Evaluation

Multi-Threading Parallel Sorting (Input Size 32)

```
------------------------------------------------------------------
 - 1 THREAD:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 0: Sorting 0 to 31

         - All threads finished -

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.000553 sec

    - Memory Before: 1584 KB
    - Memory After: 2092 KB

    - Memory Delta: 508 KB
```

```
------------------------------------------------------------------
 - 2 THREADS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 0: Sorting 0 to 15
        Thread 1: Sorting 16 to 31

         - All threads finished -

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.000178 sec

    - Memory Before: 2092 KB
    - Memory After: 2100 KB

    - Memory Delta: 8 KB
```

```
------------------------------------------------------------------
 - 4 THREADS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 0: Sorting 0 to 7
        Thread 1: Sorting 8 to 15
        Thread 2: Sorting 16 to 23
        Thread 3: Sorting 24 to 31

         - All threads finished -

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.000387 sec

    - Memory Before: 2100 KB
    - Memory After: 2120 KB

    - Memory Delta: 20 KB
```

```
------------------------------------------------------------------
 - 8 THREADS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 0: Sorting 0 to 3
        Thread 1: Sorting 4 to 7
        Thread 2: Sorting 8 to 11
        Thread 3: Sorting 12 to 15
        Thread 4: Sorting 16 to 19
        Thread 7: Sorting 28 to 31
        Thread 5: Sorting 20 to 23
        Thread 6: Sorting 24 to 27

         - All threads finished -

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.000539 sec

    - Memory Before: 2120 KB
    - Memory After: 2116 KB

    - Memory Delta: -4 KB
```

```
 ---------------------------------------

Performance Summary:
Threads    Time (s)     Mem Delta (KB)
1          0.000553     508
2          0.000178     8
4          0.000387     20
8          0.000539     -4
```

Multi-Processing Parallel Sorting (Input Size 32)

```
------------------------------------------------------------------
- 1 PROCESS:
   - Before sorting (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Sorting:
       Process 0 (PID=113805): sorting 0 to 31

        - All processess finished -
       Total memory used by children: 748 KB

   - After sorting (first 20 elements):
       4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

   - Execution Time: 0.000445 sec
```

```
------------------------------------------------------------------
 - 2 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=113806): sorting 0 to 15
        Process 1 (PID=113807): sorting 16 to 31

         - All processess finished -
        Total memory used by children: 1496 KB

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.001046 sec
```

```
------------------------------------------------------------------
 - 4 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=113808): sorting 0 to 7
        Process 1 (PID=113809): sorting 8 to 15
        Process 2 (PID=113810): sorting 16 to 23
        Process 3 (PID=113811): sorting 24 to 31

         - All processess finished -
        Total memory used by children: 2992 KB

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.000837 sec
```

```
------------------------------------------------------------------
 - 8 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=113812): sorting 0 to 3
        Process 3 (PID=113815): sorting 12 to 15
        Process 4 (PID=113816): sorting 16 to 19
        Process 6 (PID=113818): sorting 24 to 27
        Process 1 (PID=113813): sorting 4 to 7
        Process 2 (PID=113814): sorting 8 to 11
        Process 5 (PID=113817): sorting 20 to 23
        Process 7 (PID=113819): sorting 28 to 31

         - All processess finished -
        Total memory used by children: 5984 KB

    - After sorting (first 20 elements):
        4 4 7 9 12 17 21 27 33 33 35 40 40 41 41 43 43 48 52 58

    - Execution Time: 0.001528 sec
```

```
----------------------------------------

Performance Summary:
Processes  Time (s)     Mem Usage (KB)
1          0.000445     748
2          0.001046     1496
4          0.000837     2992
8          0.001528     5984
```

Multi-Threading Max-Value Aggregation (Input Size 32)

```
----------------------------------------------------------------------
- 1 THREAD:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Thread 0: Finding local max in 0 to 31

        - All threads finished -

   - Global Max: 98

   - Execution Time: 0.000416 sec

   - Memory Before: 1460 KB
   - Memory After: 2036 KB
   - Memory Delta: 576 KB
```

```
----------------------------------------------------------------------
- 2 THREADS:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Thread 0: Finding local max in 0 to 15
       Thread 1: Finding local max in 16 to 31

        - All threads finished -

   - Global Max: 98

   - Execution Time: 0.000122 sec

   - Memory Before: 2036 KB
   - Memory After: 2044 KB
   - Memory Delta: 8 KB
```

```
----------------------------------------------------------------------
- 4 THREADS:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Thread 0: Finding local max in 0 to 7
       Thread 1: Finding local max in 8 to 15
       Thread 2: Finding local max in 16 to 23
       Thread 3: Finding local max in 24 to 31

        - All threads finished -

   - Global Max: 98

   - Execution Time: 0.000186 sec

   - Memory Before: 2044 KB
   - Memory After: 2064 KB
   - Memory Delta: 20 KB
```

```
------------------------------------------------------------------
 - 8 THREADS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Thread 0: Finding local max in 0 to 3
        Thread 1: Finding local max in 4 to 7
        Thread 3: Finding local max in 12 to 15
        Thread 4: Finding local max in 16 to 19
        Thread 2: Finding local max in 8 to 11
        Thread 5: Finding local max in 20 to 23
        Thread 6: Finding local max in 24 to 27
        Thread 7: Finding local max in 28 to 31

         - All threads finished -

    - Global Max: 98

    - Execution Time: 0.000392 sec

    - Memory Before: 2064 KB
    - Memory After: 2064 KB
    - Memory Delta: 0 KB
```

```
 ----------------------------------------

Performance Summary:
Thread     Time (s)     Mem Delta (KB)
1          0.000416     576
2          0.000122     8
4          0.000186     20
8          0.000392     0
```

Multi-Processing Max-Value Aggregation (Input Size 32)

```
------------------------------------------------------------------
 - 1 PROCESS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Process 0 (PID=119583): sorting 0 to 31

         - All processess finished -

    - Global Max: 98

    - Total memory used by children: 812 KB

    - Execution Time: 0.001146 sec

    - Memory Before: 1416 KB
    - Memory After: 1564 KB
    - Memory Delta: 148 KB
```

```
- 2 PROCESSES:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Process 0 (PID=119584): sorting 0 to 15
       Process 1 (PID=119585): sorting 16 to 31

        - All processess finished -

   - Global Max: 98

   - Total memory used by children: 1624 KB

   - Execution Time: 0.001850 sec

   - Memory Before: 1628 KB
   - Memory After: 1628 KB
   - Memory Delta: 0 KB
```

```
- 4 PROCESSES:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Process 0 (PID=119586): sorting 0 to 7
       Process 1 (PID=119587): sorting 8 to 15
       Process 2 (PID=119588): sorting 16 to 23
       Process 3 (PID=119589): sorting 24 to 31

        - All processess finished -

   - Global Max: 98

   - Total memory used by children: 3248 KB

   - Execution Time: 0.002286 sec

   - Memory Before: 1628 KB
   - Memory After: 1628 KB
   - Memory Delta: 0 KB
```

```
- 8 PROCESSES:
   - Array (first 20 elements):
       66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Finding Global Max:
       Process 0 (PID=119590): sorting 0 to 3
       Process 2 (PID=119592): sorting 8 to 11
       Process 1 (PID=119591): sorting 4 to 7
       Process 5 (PID=119595): sorting 20 to 23
       Process 6 (PID=119596): sorting 24 to 27
       Process 7 (PID=119597): sorting 28 to 31
       Process 3 (PID=119593): sorting 12 to 15
       Process 4 (PID=119594): sorting 16 to 19

        - All processess finished -

   - Global Max: 98

   - Total memory used by children: 6496 KB

   - Execution Time: 0.004239 sec

   - Memory Before: 1628 KB
   - Memory After: 1628 KB
   - Memory Delta: 0 KB
```

```
---------------------------------------

Performance Summary:
Processes  Time (s)     Mem Usage (KB)
1          0.001146     812
2          0.001850     1624
4          0.002286     3248
8          0.004239     6496
```

For an array input size of 32, all implementations showed correct results. This means all implementations returned either an array that was fully sorted or the accurate maximum value of the input array. This confirmed their correctness and that the mapping and reducing phase works for all.

Multi-Threading Parallel Sorting (Input Size 131,072)

```
----------------------------------------------------------------
 - 1 THREAD:
   - Before sorting (first 20 elements):
      66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Sorting:
      Thread 0: Sorting 0 to 131071

       - All threads finished -

   - After sorting (first 20 elements):
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   - Execution Time: 0.293143 sec

   - Memory Before: 1996 KB
   - Memory After: 2564 KB

   - Memory Delta: 568 KB
```

```
----------------------------------------------------------------
 - 2 THREADS:
   - Before sorting (first 20 elements):
      66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

   - Sorting:
      Thread 0: Sorting 0 to 65535
      Thread 1: Sorting 65536 to 131071

       - All threads finished -

   - After sorting (first 20 elements):
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   - Execution Time: 0.126942 sec

   - Memory Before: 2564 KB
   - Memory After: 3616 KB

   - Memory Delta: 1052 KB
```

```
------------------------------------------------------------------------
 - 4 THREADS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 0: Sorting 0 to 32767
        Thread 1: Sorting 32768 to 65535
        Thread 2: Sorting 65536 to 98303
        Thread 3: Sorting 98304 to 131071

         - All threads finished -

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.073616 sec

    - Memory Before: 3616 KB
    - Memory After: 3932 KB

    - Memory Delta: 316 KB
```

```
------------------------------------------------------------------------
 - 8 THREADS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Thread 3: Sorting 49152 to 65535
        Thread 0: Sorting 0 to 16383
        Thread 1: Sorting 16384 to 32767
        Thread 2: Sorting 32768 to 49151
        Thread 4: Sorting 65536 to 81919
        Thread 5: Sorting 81920 to 98303
        Thread 6: Sorting 98304 to 114687
        Thread 7: Sorting 114688 to 131071

         - All threads finished -

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.048156 sec

    - Memory Before: 3932 KB
    - Memory After: 4188 KB

    - Memory Delta: 256 KB
```

```
---------------------------------------

Performance Summary:
Threads     Time (s)     Mem Delta (KB)
1           0.293143     568
2           0.126942     1052
4           0.073616     316
8           0.048156     256
```

Multi-Processing Parallel Sorting (Input Size 131,072)

```
------------------------------------------------------------------
 - 1 PROCESS:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=139684): sorting 0 to 131071

         - All processess finished -

    - Total memory used by children: 1340 KB

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.258321 sec
```

```
------------------------------------------------------------------
 - 2 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=139685): sorting 0 to 65535
        Process 1 (PID=139686): sorting 65536 to 131071

         - All processess finished -

    - Total memory used by children: 2148 KB

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.092386 sec
```

```
------------------------------------------------------------------
 - 4 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=139687): sorting 0 to 32767
        Process 1 (PID=139688): sorting 32768 to 65535
        Process 2 (PID=139689): sorting 65536 to 98303
        Process 3 (PID=139690): sorting 98304 to 131071

         - All processess finished -

    - Total memory used by children: 4272 KB

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.051038 sec
```

```
------------------------------------------------------------------
 - 8 PROCESSES:
    - Before sorting (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Sorting:
        Process 0 (PID=139691): sorting 0 to 16383
        Process 1 (PID=139692): sorting 16384 to 32767
        Process 2 (PID=139693): sorting 32768 to 49151
        Process 5 (PID=139696): sorting 81920 to 98303
        Process 3 (PID=139694): sorting 49152 to 65535
        Process 6 (PID=139697): sorting 98304 to 114687
        Process 7 (PID=139698): sorting 114688 to 131071
        Process 4 (PID=139695): sorting 65536 to 81919

         - All processess finished -

    - Total memory used by children: 7988 KB

    - After sorting (first 20 elements):
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

    - Execution Time: 0.032461 sec
```

```
--------------------------------------

Performance Summary:
Processes  Time (s)     Mem Usage (KB)
1          0.258321     1340
2          0.092386     2148
4          0.051038     4272
8          0.032461     7988
```

Multi-Threading Max-Value Aggregation (Input Size 131,072)

```
------------------------------------------------------------------
 - 1 THREAD:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Thread 0: Finding local max in 0 to 131071

         - All threads finished -

    - Global Max: 99

    - Execution Time: 0.000757 sec

    - Memory Before: 1996 KB
    - Memory After: 2544 KB
    - Memory Delta: 548 KB
```

```
------------------------------------------------------------------
 - 2 THREADS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Thread 0: Finding local max in 0 to 65535
        Thread 1: Finding local max in 65536 to 131071

         - All threads finished -

    - Global Max: 99

    - Execution Time: 0.000436 sec

    - Memory Before: 2544 KB
    - Memory After: 2556 KB
    - Memory Delta: 12 KB
```

```
------------------------------------------------------------------
 - 4 THREADS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Thread 0: Finding local max in 0 to 32767
        Thread 1: Finding local max in 32768 to 65535
        Thread 3: Finding local max in 98304 to 131071
        Thread 2: Finding local max in 65536 to 98303

         - All threads finished -

    - Global Max: 99

    - Execution Time: 0.000444 sec

    - Memory Before: 2556 KB
    - Memory After: 2576 KB
    - Memory Delta: 20 KB
```

```
------------------------------------------------------------------
 - 8 THREADS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Thread 0: Finding local max in 0 to 16383
        Thread 3: Finding local max in 49152 to 65535
        Thread 1: Finding local max in 16384 to 32767
        Thread 5: Finding local max in 81920 to 98303
        Thread 2: Finding local max in 32768 to 49151
        Thread 7: Finding local max in 114688 to 131071
        Thread 6: Finding local max in 98304 to 114687
        Thread 4: Finding local max in 65536 to 81919

         - All threads finished -

    - Global Max: 99

    - Execution Time: 0.000748 sec

    - Memory Before: 2576 KB
    - Memory After: 2580 KB
    - Memory Delta: 4 KB
```

```
----------------------------------------

Performance Summary:
Thread     Time (s)     Mem Delta (KB)
1          0.000757     548
2          0.000436     12
4          0.000444     20
8          0.000748     4
```

Multi-Processing Max-Value Aggregation (Input Size 131,072)

```
----------------------------------------------------------------------
 - 1 PROCESS:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Process 0 (PID=140858): sorting 0 to 131071

         - All processess finished -

    - Global Max: 99

    - Total memory used by children: 1324 KB

    - Execution Time: 0.000882 sec

    - Memory Before: 1972 KB
    - Memory After: 2100 KB
    - Memory Delta: 128 KB
```

```
----------------------------------------------------------------------
 - 2 PROCESSES:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Process 0 (PID=140859): sorting 0 to 65535
        Process 1 (PID=140860): sorting 65536 to 131071

         - All processess finished -

    - Global Max: 99

    - Total memory used by children: 2200 KB

    - Execution Time: 0.000754 sec

    - Memory Before: 2164 KB
    - Memory After: 2164 KB
    - Memory Delta: 0 KB
```

```
------------------------------------------------------------------
 - 4 PROCESSES:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Process 0 (PID=140861): sorting 0 to 32767
        Process 1 (PID=140862): sorting 32768 to 65535
        Process 2 (PID=140863): sorting 65536 to 98303
        Process 3 (PID=140864): sorting 98304 to 131071

          - All processess finished -

    - Global Max: 99

    - Total memory used by children: 3952 KB

    - Execution Time: 0.001177 sec

    - Memory Before: 2164 KB
    - Memory After: 2164 KB
    - Memory Delta: 0 KB
```

```
------------------------------------------------------------------
 - 8 PROCESSES:
    - Array (first 20 elements):
        66 40 81 41 12 58 21 40 35 43 74 43 17 4 96 62 92 48 98 59

    - Finding Global Max:
        Process 0 (PID=140865): sorting 0 to 16383
        Process 1 (PID=140866): sorting 16384 to 32767
        Process 3 (PID=140868): sorting 49152 to 65535
        Process 2 (PID=140867): sorting 32768 to 49151
        Process 4 (PID=140869): sorting 65536 to 81919
        Process 5 (PID=140870): sorting 81920 to 98303
        Process 7 (PID=140872): sorting 114688 to 131071
        Process 6 (PID=140871): sorting 98304 to 114687

          - All processess finished -

    - Global Max: 99

    - Total memory used by children: 7452 KB

    - Execution Time: 0.001779 sec

    - Memory Before: 2164 KB
    - Memory After: 2164 KB
    - Memory Delta: 0 KB
```

```
 ---------------------------------------

 Performance Summary:
 Processes  Time (s)     Mem Usage (KB)
 1          0.000882     1324
 2          0.000754     2200
 4          0.001177     3952
 8          0.001779     7452
```

Execution Time (Input Size = 137,072)

Processes/ Threads	MT Parallel Time (s)	MP Parallel Time (s)	MT Max-Val Time (s)	MP Max-Val Time (s)
1	**0.293143**	**0.255534**	**0.000953**	**0.000882**
2	**0.126942**	**0.093328**	**0.000475**	**0.000754**
4	**0.073616**	**0.051145**	**0.000601**	**0.001177**
8	**0.048156**	**0.032543**	**0.000843**	**0.001779**

Memory Usage (Input Size = 137,072)

Processes/ Threads	MT Parallel Mem (kb)	MP Parallel Mem (kb)	MT Max-Val Mem (kb)	MP Max-Val Mem (kb)
1	**568**	**1592**	**540**	**1324**
2	**1052**	**2648**	**12**	**2200**
4	**316**	**5024**	**20**	**3952**
8	**256**	**9444**	**0**	**7452**

From the execution results, a lot of conclusions can be made about the performance of the different programs. Both multithreading and multiprocessing implementations showed clear performance improvements as the number of processes or threads were increased. This resulted in execution time decreasing for all from 1 to 4 workers due to the effectiveness of the parallelism. However, the performance gain from adding new workers beyond 4 resulted in diminished results. This could be due to things like synchronization or issues with implementations.

- Parallel Sorting
    - The multiprocessing implementation showed to be slightly faster. This could be due to its parallelism.
    - The multithreaded implementation showed to perform more efficiently using less memory due to threads sharing the same memory space.
- Max-Value Aggregation
    - Both implementations performed due to having less computation than sorting.
    - Performance for the multithreading implementation performed better than multiprocessing as more workers were added. This was due to the

semaphore synchronization causing more overhead at higher worker numbers.

Problems

There were a couple problems that were faced in this project especially around measuring more performance data. Measuring memory for multithreading especially in the max-value functions, it would at times return confusing values or even negative values at points. I found that this may be due to a number of factors especially around how little memory is needed for finding the max. Memory reading was also difficult for the multiprocessing programs as originally it was only reading the memory usage of the parent function and completely ignoring that of the child processes.

Conclusion

This project successfully demonstrates the trade-offs between multithreading and multiprocessing approaches when using them in parallel computing. While multiprocessing was shown to achieve faster raw speed, it incurred higher memory usage and synchronization costs. Multithreading, on the other hand, provides more efficient memory memory usage and faster execution times especially around 1-4 workers. This makes it much more suitable for lightweight parallel tasks. So the ideal model depends solely on the task it's being used for, allocated resources, and the complexity of the synchronization.