Multithreaded Prime Number Detection Using POSIX Threads

2024-2025 Spring CSE440 Parallel Programming Midterm Project

Buse Çoban
20200808070

Akdeniz University
Computer Engineering

1.Problem & Solution Strategy

1.1. Problem Strategy

We need to count all the primes in the range [1,500000000]Performing a trial-division primality test on each number up to its square root becomes extremely costly when done on a single thread.

#define MAX 500000000UL

printf("Threads: %d, Time taken: %.6f seconds\n", nthreads, elapsed);

The program measures wall-clock time for different thread counts (nthreads). Empirical results show that speedup saturates around 24 threads—adding more threads yields negligible further gains—indicating limits imposed by synchronization overhead, cache contention, and thread-management costs.

1.2. Solution Strategy

1.2.1.Range Partitioning

```
typedef struct {
    uint64_t start;   /* inclusive start of range */
    uint64_t end;     /* exclusive end of range */
    uint64_t count;   /* number of primes found */
} range_t;
```

1.2.2.Threaded Trial Division

```
static int is_prime(uint64_t n) {
    if (n < 2)                  return 0;  /* 0 and 1 are not prime */
    if (n == 2)                 return 1;  /* 2 is prime */
    if (n % 2 == 0)             return 0;  /* even numbers >2 are not prime */

    /* only test odd divisors up to sqrt(n) */
    uint64_t limit = (uint64_t) sqrt((double) n);
    for (uint64_t i = 3; i <= limit; i += 2) {
        if (n % i == 0)         return 0;  /* divisible => not prime */
    }
    return 1;  /* no divisor found => prime */
}
```

1.2.3.Minimizing Synchronization

Threads accumulate their own rng->count without locks. Only after pthread_join do we sum up all seg[i].count into a global total—thus keeping the critical section overhead effectively zero.

1.2.4.Time Measurement

```c
/* Start the timer */
struct timeval t0, t1;
gettimeofday(&t0, NULL);
```

```c
/* Stop the timer */
gettimeofday(&t1, NULL);
double elapsed = (t1.tv_sec - t0.tv_sec)
                + (t1.tv_usec - t0.tv_usec) / 1e6;
```

1.3.Correctness

The prime numbers found by each thread were recorded with a debug mode added to the program

```c
#ifdef DEBUG
#  define DBG(fmt, ...) \
      do { fprintf(stderr, fmt, ##__VA_ARGS__); } while (0)
#else
#  define DBG(fmt, ...) \
      do { } while (0)
#endif
```

```c
/* Wait for all threads to finish and accumulate results */
uint64_t total_primes = 0;
for (int i = 0; i < nthreads; ++i) {
    pthread_join(tid[i], NULL);
    /* bu correctness için snra kaldır */
    DBG("Thread %2d found %10lu primes\n", i, seg[i].count);
    total_primes += seg[i].count;
}
```

2.Hardware Configuration

My macOS Hardware Configuration Summary:

- **CPU Model:Apple M3 Pro (12-core: 6 performance + 6 efficiency)**
- **CPU Frequency:Performance cores: up to 4.05 GHz,Efficiency cores: up to 2.75 GHz**
- **L1 cache size:Performance cores: 192 KiB (I) + 128 KiB (D) per core,Efficiency cores: 128 KiB (I) + 64 KiB (D) per core**
- **L2 cache size:Performance cluster: 16 MiB shared,Efficiency cluster: 4 MiB shared**
- **Last-Level (System) Cache (LLC/SLC):** ~64 MiB
- **RAM Type: LPDDR5-6400**
- **RAM Capacity: 18 GB unified**
- **RAM Frequency: 6 400 MT/s (effective)**

Commands:

- (base) busecoban@Buses-MacBook-Pro ~ % sysctl -n machdep.cpu.brand_string
  Apple M3 Pro

- (base) busecoban@Buses-MacBook-Pro ~ % sysctl -a | grep cache
  hw.perflevel0.l1icachesize: 196608
  hw.perflevel0.l1dcachesize: 131072
  hw.perflevel0.l2cachesize: 16777216
  hw.perflevel1.l1icachesize: 131072
  hw.perflevel1.l1dcachesize: 65536
  hw.perflevel1.l2cachesize: 4194304

- (base) busecoban@Buses-MacBook-Pro ~ % sysctl -n hw.memsize | awk '{printf "%.2f GB\n", $1/1024/1024/1024}’
  18.00 GB

- (base) busecoban@Buses-MacBook-Pro ~ % system_profiler

SPMemoryDataType | grep -E "Type:|Speed:|Size:"

Type: LPDDR5

- system_profiler SPHardwareDataType

Hardware:

Hardware Overview:

Model Name: MacBook Pro

Model Identifier: Mac15,6

Model Number: MRX73TU/A

Chip: Apple M3 Pro

Total Number of Cores: 12 (6 performance and 6 efficiency)

Memory: 18 GB

System Firmware Version: 11881.81.4

OS Loader Version: 11881.81.4

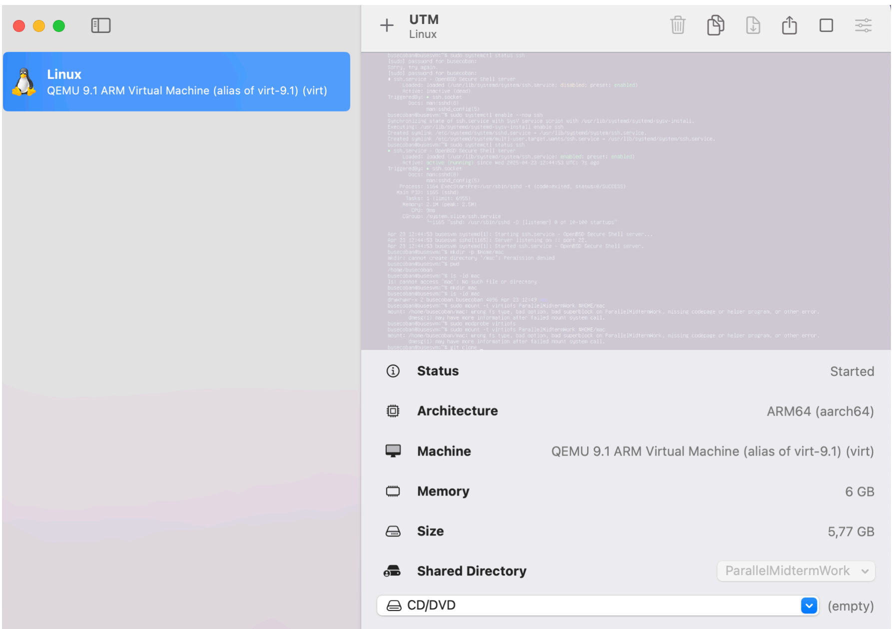
Serial Number (system): TMQ27H9XCK

Hardware UUID: 86778FB2-2F5D-5BEB-B9FE-DBB951C10983

Provisioning UDID: 00006030-001831492E50001C

Activation Lock Status: Enabled

I installed Ubuntu 24.04 LTS in a UTM/QEMU VM on macOS, allocated 8 vCPUs, 6 GB RAM, 40 GB disk



VM (Ubuntu 24.04 LTS) Hardware Configuration Summary:

Note: The VM presents "virtual" CPUs backed by the host's M3 Pro cores. Cache hierarchies and memory characteristics map to the host but are time-shared among vCPUs.

- **vCPU Count & Model: 8 vCPUs (QEMU Virtual CPU mapped to Apple M3 Pro)**
- **vCPU Frequency: Up to host core speeds (Perf 4.05 GHz / Eff 2.75 GHz)**
- **L1 Cache: Same as host (192 KiB+128 KiB per P-core; 128 KiB+64 KiB per E-core)**
- **L2 Cache: Same as host (16 MiB per P-cluster; 4 MiB per E-cluster)**
- **Last-Level Cache: ~64 MiB SLC (shared among all cores)**
- **RAM Type: LPDDR5-6400 (host-provided)**
- **RAM Capacity: 6 GB allocated to the VM**
- **RAM Frequency: 6 400 MT/s (effective)**

Commands:

```
busecoban@busesvm:~$ sudo dmidecode -t memory | grep -E 'Type:|Size:|Speed:'
[sudo] password for busecoban:
        Error Correction Type: Multi-bit ECC
        Size: 6 GB
        Type: RAM
        Speed: Unknown
        Configured Memory Speed: Unknown
```

```
busecoban@busesvm:~$ lsb_release -a
uname -mrs
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.2 LTS
Release:        24.04
Codename:       noble
Linux 6.8.0-58-generic aarch64
```

```
busecoban@busesvm:~$ lsblk -o NAME,SIZE,TYPE,MOUNTPOINT
NAME                       SIZE TYPE MOUNTPOIN
sr0                       1024M rom
vda                         40G disk
├─vda1                       1G part /boot/efi
├─vda2                       2G part /boot
└─vda3                    36.9G part
  └─ubuntu--vg-ubuntu--lv 18.5G lvm  /
```

```
busecoban@busesvm:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           5.8Gi       1.6Gi       2.3Gi       5.2Mi       2.0Gi       4.1Gi
Swap:          3.9Gi          0B       3.9Gi
```

```
busecoban@busesvm:~$ lscpu | grep -E 'Model name|Socket|Core|Thread'
Model name:                          -
Thread(s) per core:                  1
Core(s) per socket:                  8
Socket(s):                           1
```

I then connected to this VM from my host via VSCode Remote – SSH extension to edit, compile, test and visualize the code.

```
busecoban@busesvm:~$ sudo systemctl status ssh
[sudo] password for busecoban:
Sorry, try again.
[sudo] password for busecoban:
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: enabled)
     Active: inactive (dead)
TriggeredBy: ● ssh.socket
       Docs: man:sshd(8)
             man:sshd_config(5)
busecoban@busesvm:~$ sudo systemctl enable --now ssh
Synchronizing state of ssh.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable ssh
Created symlink /etc/systemd/system/sshd.service → /usr/lib/systemd/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants/ssh.service → /usr/lib/systemd/system/ssh.service.
busecoban@busesvm:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-04-23 12:44:53 UTC; 7s ago
TriggeredBy: ● ssh.socket
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 1164 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 1165 (sshd)
      Tasks: 1 (limit: 6955)
     Memory: 2.1M (peak: 2.5M)
        CPU: 9ms
     CGroup: /system.slice/ssh.service
             └─1165 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

3.Interaction with ChatGPT

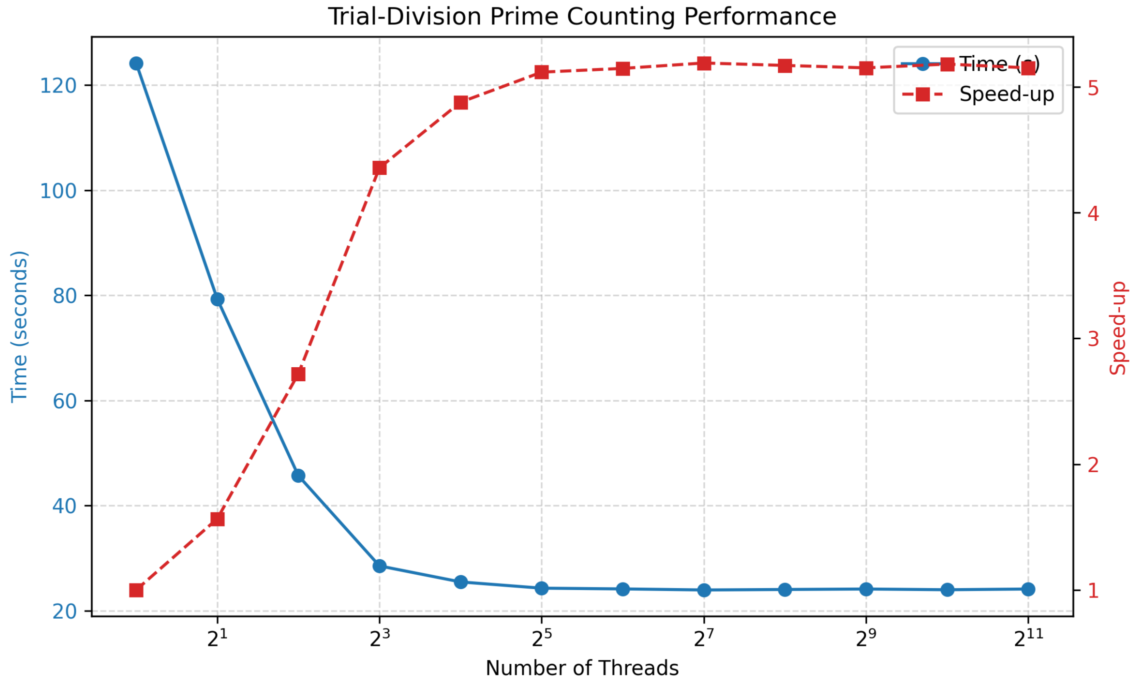
https://chatgpt.com/share/680921b5-7860-8010-9b3d-e7fe725922df

4.Data Visulazations

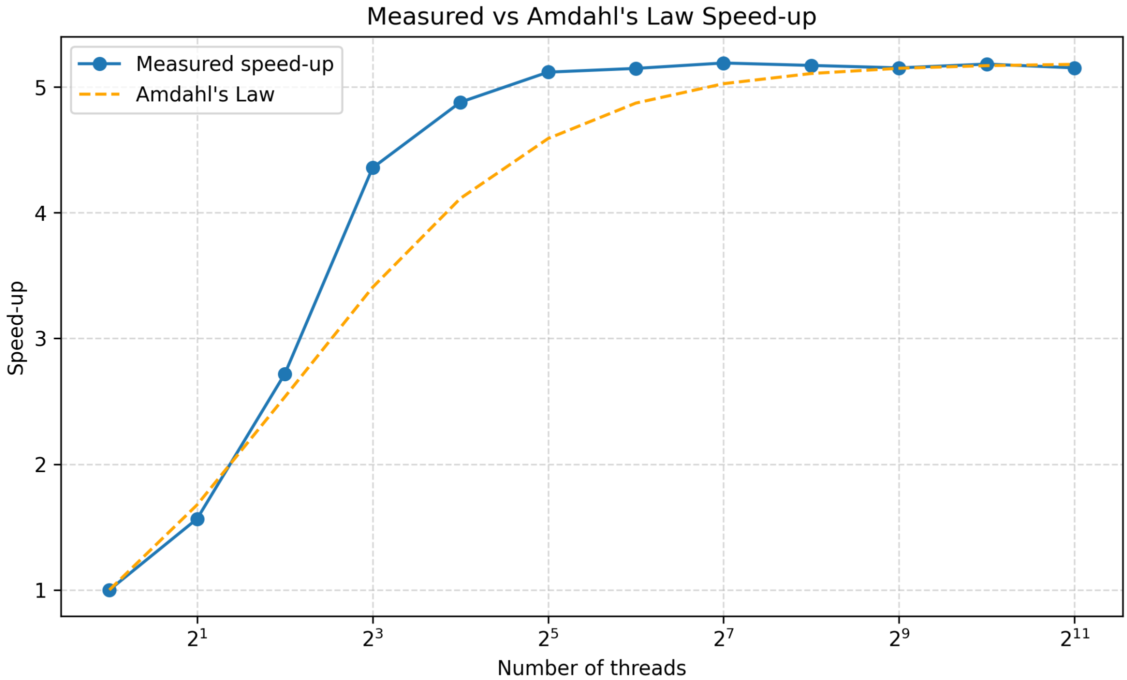
For plotting the performance results I used Python/Matplotlib script.Full visualization codes are in my GitHub repository: https://github.com/busecoban/prime-pthreads

4.1.Trial-Division Timing Results

Threads	Time (s)
1	124.19
2	79.31
4	45.68
8	28.51
16	25.46
32	24.27
64	24.13
128	23.93
256	24.02
512	24.11
1024	23.97
2048	24.11



The chart shows how the trial-division prime counting routine scales with thread count. As threads increase from 1 to 8, runtime falls sharply from ~122 s to ~28 s. Beyond 8 threads (the number of vCPUs), further increases yield only marginal gains, leveling off around ~24 s.



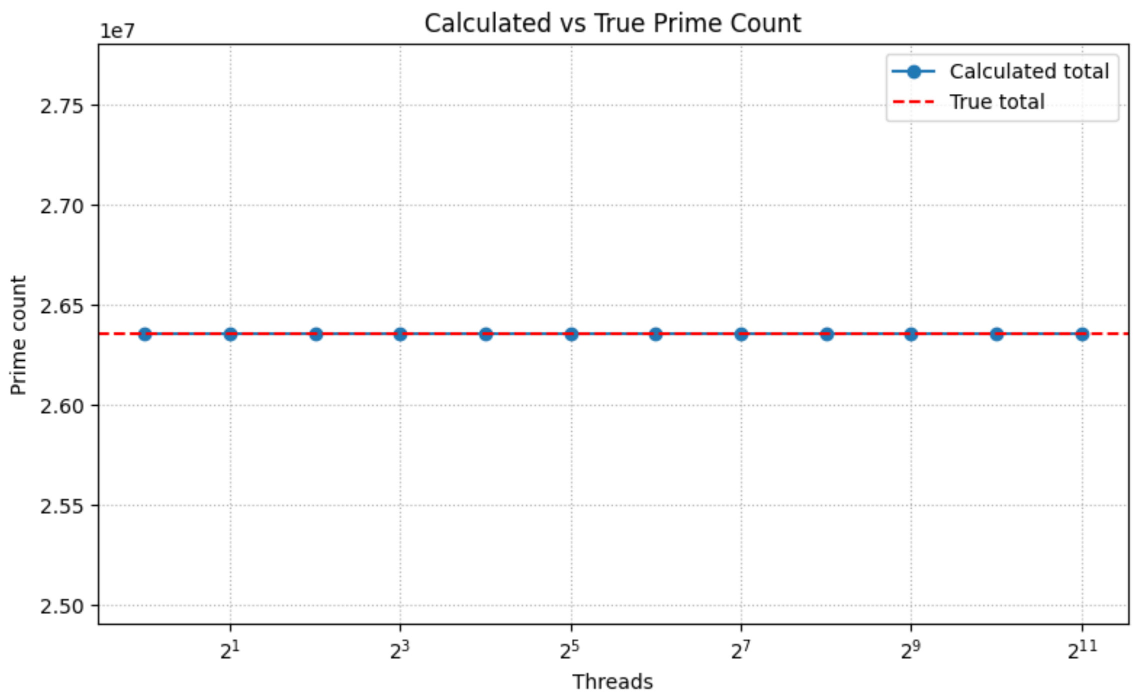
In the overlaid plot, the dashed line shows the theoretical speed-up predicted by Amdahl's Law (using a parallel fraction $p \approx 0.88$), while the solid markers plot the actual speed-up measured on the 8-vCPU VM. Up to 8 threads, the data adheres closely to the theory, reflecting near-linear scaling as each thread maps to a separate core. Beyond 8 threads, however, the measured curve flattens at around 5.1×–5.2×, diverging from the rising theoretical line. This plateau highlights the effects of physical core saturation, excess context-switch and thread-management overhead, and cache/memory contention—all of which combine with the code's inherently serial regions to limit further gains.

$$S(n) = \frac{1}{(1-p) + p/n}$$

4.2. Correctness of Results

The prime numbers found by each thread were recorded with a debug mode added to the program as mentioned in the solution design. The numbers of all threads were collected with the Python script and compared with the real value (26,355,867). Since the calculated sum in all configurations was exactly equal to the reference value, the correctness of the program was proven.

Threads	Time	Computed	True	Error
1	123.380000	26355867	26355867	0
2	80.480000	26355867	26355867	0
4	46.180000	26355867	26355867	0
8	27.970000	26355867	26355867	0
16	25.650000	26355867	26355867	0
32	24.420000	26355867	26355867	0
64	24.540000	26355867	26355867	0
128	24.400000	26355867	26355867	0
256	24.500000	26355867	26355867	0
512	24.420000	26355867	26355867	0
1024	24.780000	26355867	26355867	0
2048	24.290000	26355867	26355867	0



5.Discussion

Threads	Time (s)	Speed-up	Efficiency (%)
1	124.19	1.00	100.0
2	79.31	1.57	78.3
4	45.68	2.72	68.0
8	28.51	4.36	54.5
16	25.46	4.88	30.5
32	24.27	5.12	16.0
64	24.13	5.15	8.0
128	23.93	5.19	4.1
256	24.02	5.17	2.0
512	24.11	5.15	1.0
1024	23.97	5.18	0.5
2048	24.11	5.15	0.3

1. **Physical Core Limit (8 vCPUs):**

Scaling is near-linear only up to the number of available virtual CPUs. Beyond 8 threads, additional threads contend for the same cores, adding only context-switch overhead without meaningful speed-up.

2. **CPU-Bound Nature of Trial Division:**

Every candidate n invokes a $\sqrt{n}$ division loop, making the algorithm heavily compute-bound. While parallelism distributes these loops across cores, each division sequence itself remains strictly serial.

3. **Amdahl's Law & Serial Regions:**

Fixed serial tasks—thread creation/joining, timing measurements, and result aggregation—occupy a constant portion of total runtime. As a result, theoretical speed-up is capped well below the ideal 8×.

$$S(n) = \frac{1}{(1-p) + p/n}$$

4. **Cache & Memory Bandwidth Constraints:**

Concurrent threads repeatedly read shared data (e.g. the small-primes list), leading to L3 cache contention and memory-bandwidth thrashing. As thread count increases, this memory traffic approaches hardware limits.

5. **Thread-Management Overhead:**

Launching and scheduling large numbers of threads incurs OS overhead for stack allocation and context switching. When work chunks are small, this overhead can dominate compute time, eroding any incremental gains.

Appendix

Chat History: https://chatgpt.com/share/680921b5-7860-8010-9b3d-e7fe725922df

Github Repository: https://github.com/busecoban/prime-pthreads