CI Pathway: Parallel Computing

Assignment - 2

UCLA, Statistics
Hochan Son
Summer 2025
June 26, 2025

1 Introduction

This analysis evaluates parallel computing performance exerise for prime number . The study compares four implementation approaches across varying thread/process counts to assess scalability and efficiency characteristics.

2 Hardware Environment

2.1 System Specifications

The experiments were conducted on the NCSA Delta HPC cluster, which is a high-performance computing environment designed for parallel processing tasks. The specifications of the system are as follows:

Table 1: NCSA Delta Compute Environment

Component	**Specification**
Compute Platform	NCSA Delta HPC Cluster
Login Node	dt-login04.delta.ncsa.illinois.edu
Compute Node	cn094.delta.ncsa.illinois.edu
Operating System	Linux 4.18.0-477.95.1.el8_8.x86_64
Distribution	Red Hat Enterprise Linux 8
Architecture	x86_64
Node Interconnect	HPE Slingshot

2.2 Processor Architecture

Table 2: AMD EPYC 7763 Processor Specifications

Parameter	Value
CPU Model	AMD EPYC 7763 64-Core Processor
Architecture	AMD Zen 3 (Milan)
Physical Cores	64 per socket
Hardware Threads	128 (2-way SMT)
Base Clock	2.45 GHz
Boost Clock	Up to 3.5 GHz
Manufacturing Process	7nm TSMC
Socket Type	SP3

3 Exercises For This Module.

3.1 OpenMP Exercise

1. You will find this code in `projectsbecsurbanic` as either `prime_serial.c or prime_serial.f`.

   Your job is to accelerate this code using OpenMP. You should see a dramatic speedup if you use our OpenMP directives effectively.

   If you are not careful, you could introduce a race condition and have inconsistent results. If you use the same caution we used in the examples above, you will avoid this.

   To reiterate what you did in the previous module.

   (a) Compile with something like

   ```
   nvc -mp prime_serial.c
   or
   nvfortran -mp prime_serial.f
   ```

   (b) Grab multiple cores on a compute node with something like

   ```
   srun --account=becs-delta-cpu --partition=cpu-interactive \
   --nodes=1 --cpus-per-task=32 --pty bash
   ```

   (c) Set the number of threads you wish to run with using something like

   ```
   export OMP_NUM_THREADS=16
   ```

Submit your code and your timings for at least 1, 4, 8, 16 and 32 threads.

4 Solution:

Conclusion

5 Appendix.code

Here's some of our code (Note the use of VerbatimInput from package `fancyvrb`):

5.1 Code A: prime_serial.c

```
# include <stdlib.h>
# include <stdio.h>

int main ( int argc, char *argv[] ){

  int n = 500000;
  int not_primes=0;
  int i,j;

  for ( i = 2; i <= n; i++ ){
    for ( j = 2; j < i; j++ ){
      if ( i % j == 0 ){
        not_primes++;
        break;
      }
    }
  }

  printf("Primes: %d\n", n - not_primes);

}
```

5.2 Code B: prime_parallel.c

```
# include <stdlib.h>
# include <stdio.h>
#include <omp.h>

#define MAX_THREADS 32

int main ( int argc, char *argv[] ){

  int n = 500000;
  int not_primes=0;
  int i,j;

  // Set number of threads at runtime
  int num_threads = MAX_THREADS;
  if (omp_get_max_threads() < MAX_THREADS) {
      num_threads = omp_get_max_threads();
  }
  omp_set_num_threads(num_threads);
  printf("Running with %d OpenMP threads\n", num_threads);

  // parallel running
```

```
  #pragma omp parallel for private(i,j)
  for ( i = 2; i <= n; i++ ){
    for ( j = 2; j < i; j++ ){
      if ( i % j == 0 ){
        not_primes++;
        break;
      }
    }
  }

  printf("Primes: %d\n", n - not_primes);

}
```

6 Appendix.results

6.1 result.txt

```
=== Basic System Info ===
Date: Tue Jun 17 20:05:14 CDT 2025
System: Linux cn094.delta.ncsa.illinois.edu 4.18.0-477.95.1.el8_8.x86_64 #1
    SMP Fri Apr 11 09:50:48 EDT 2025 x86_64 x86_64 x86_64 GNU/Linux
CPU Info: AMD EPYC 7763 64-Core Processor
==========================
!!!!STARTING SERIAL PROCESS TEST!!!!
=== Test with 1 threads ===
Start time: 2025-06-17 20:05:14.216333549
Primes: 41539
End time: 2025-06-17 20:05:32.876436728
Total wall clock time: 18.656 seconds
----------------------------------------

=== Test with 4 threads ===
Start time: 2025-06-17 20:05:33.884495585
Primes: 41539
End time: 2025-06-17 20:05:52.546097926
Total wall clock time: 18.658 seconds
----------------------------------------

=== Test with 8 threads ===
Start time: 2025-06-17 20:05:53.555359569
Primes: 41539
End time: 2025-06-17 20:06:12.088907489
Total wall clock time: 18.530 seconds
----------------------------------------

=== Test with 16 threads ===
Start time: 2025-06-17 20:06:13.097441928
Primes: 41539
End time: 2025-06-17 20:06:31.879970680
Total wall clock time: 18.774 seconds
----------------------------------------
```

```
=== Test with 32 threads ===
Start time: 2025-06-17 20:06:32.887758068
Primes: 41539
End time: 2025-06-17 20:06:51.432803133
Total wall clock time: 18.541 seconds
----------------------------------------

!!!!STARTING PARALLEL PROCESS TEST!!!!
=== Test with 1 threads ===
Start time: 2025-06-17 20:06:52.443311765
Running with 1 OpenMP threads
Primes: 41539
End time: 2025-06-17 20:07:10.986187461
Total wall clock time: 18.540 seconds
----------------------------------------

=== Test with 4 threads ===
Start time: 2025-06-17 20:07:11.994337910
Running with 4 OpenMP threads
Primes: 41623
End time: 2025-06-17 20:07:19.916946608
Total wall clock time: 7.919 seconds
----------------------------------------

=== Test with 8 threads ===
Start time: 2025-06-17 20:07:20.929190870
Running with 8 OpenMP threads
Primes: 41777
End time: 2025-06-17 20:07:25.112032913
Total wall clock time: 4.180 seconds
----------------------------------------

=== Test with 16 threads ===
Start time: 2025-06-17 20:07:26.120652486
Running with 16 OpenMP threads
Primes: 41929
End time: 2025-06-17 20:07:28.352685709
Total wall clock time: 2.229 seconds
----------------------------------------

=== Test with 32 threads ===
Start time: 2025-06-17 20:07:29.360748758
Running with 32 OpenMP threads
Primes: 42290
End time: 2025-06-17 20:07:30.481227374
Total wall clock time: 1.117 seconds
----------------------------------------
```