Week 8 Lab 3 Presentation

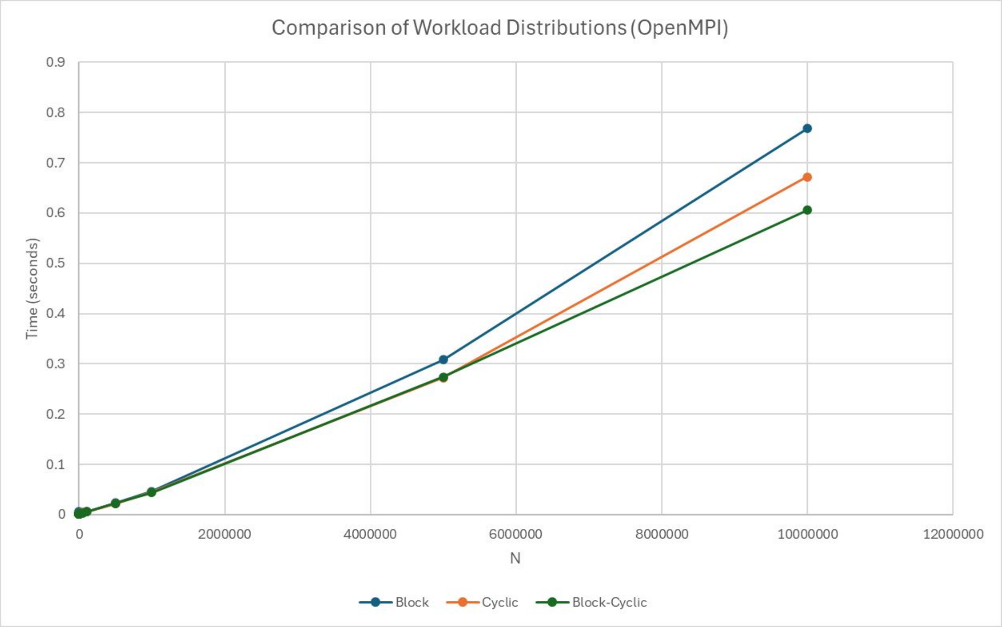
Covers: MPI II

Core Task – Prime Numbers with Open MPI

Task 1 - Workload Distributions

<table>
<tr><th>Method 1: Block</th><th>Method 2: Cyclic</th><th>Method 3: Block-Cyclic</th></tr>
<tr>
<td>Divide the range into equal, contiguous chunks, one per process.
Low overhead, output order preserved
Better Communication efficiency because fewer, larger messages over many small ones (lower latency cost).
Unequal workload distribution</td>
<td>Assign every p-th number to each process, where p = total processes. (balanced, but needs merging)
Better load balance, still not perfect
Higher merging/sorting overhead</td>
<td>In block-cyclic distribution, the data is divided into small, fixed-size blocks. These blocks are then assigned to processes in a round-robin fashion. This combines the load-balancing advantages of cyclic distribution with the communication efficiency of block distribution. (best overall)</td>
</tr>
</table>

Task 1 - Workload Distributions



n	Block	Cyclic	Block-Cyclic
10	0.006005	0.002181	0.002048
50	0.001854	0.002011	0.002064
100	0.001964	0.001923	0.001889
500	0.002087	0.001973	0.001998
1000	0.001986	0.002076	0.002116
5000	0.002043	0.002083	0.002074
10000	0.002332	0.002301	0.002302
50000	0.003974	0.003818	0.003715
100000	0.005576	0.005505	0.005831
500000	0.023474	0.021614	0.022245
1000000	0.046129	0.044979	0.043745
5000000	0.308528	0.272608	0.273717
10000000	0.768246	0.671559	0.605399

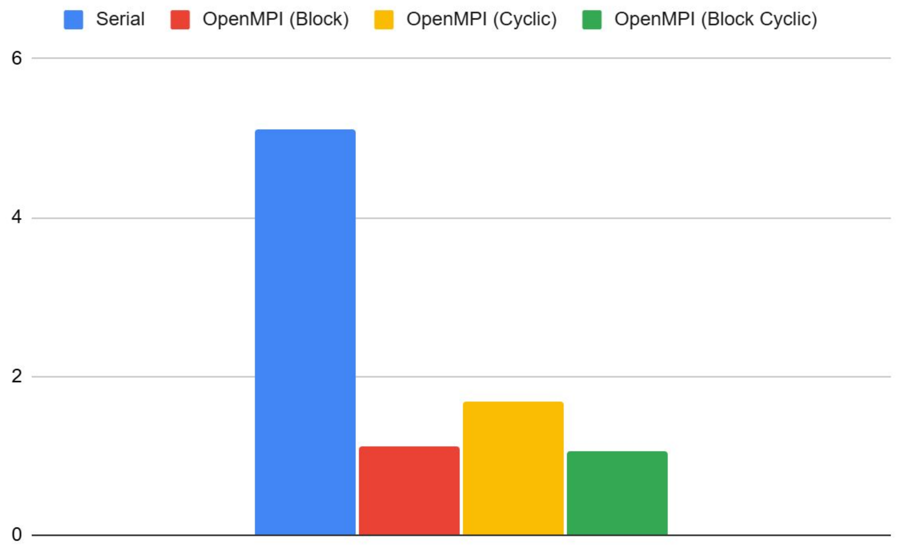
"How would the workload distribution affect the speed up?"

Each prime check costs O(√i). As n grows, √i increases very slowly compared to n. This means the workload imbalance in Block distribution is minor, so Cyclic's balancing advantage is negligible. Block-Cyclic combines the best of both: it balances workloads better than Block but avoids the high communication cost of Cyclic. Since imbalance is already minor (√i grows slowly), Block-Cyclic often gives the best overall speed-up.

Task 1 - Speedup

Comparing actual speed-up of our serial and parallel implementation of prime number searching, OpenMPI Block Cyclic seems to come up on top, with a computed speedup of:
5.10985 / 1.058977 ≈ 4.83×

Note: Ran on system with **i7-1255U with 10 Core(s).** OpenMPI ran with **4 processors.**

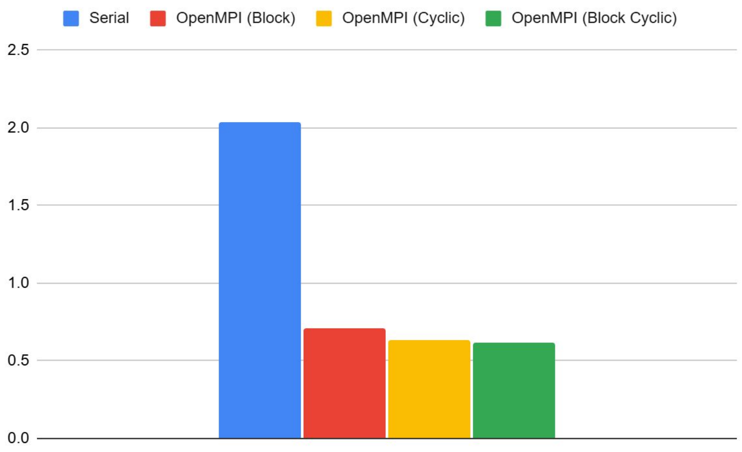


n	Serial	OpenMPI (Block)	OpenMPI (Cyclic)	OpenMPI (Block Cyclic)
10,000,000	5.10985	1.121663	1.684763	1.058977

Task 1 - Speedup (different machines)

Comparing actual speed-up of our serial and parallel implementation of prime number searching, OpenMPI Block Cyclic seems to come up on top, with a computed speedup of:
2.040729 / 0.61659 ≈ 3.30×

Note: Ran on system with **ryzen 5700x3d (8 cores)**. OpenMPI ran with **4 processors.**

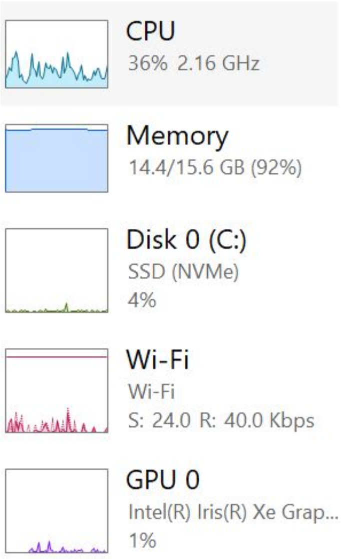


n	Serial	OpenMPI (Block)	OpenMPI (Cyclic)	OpenMPI (Block Cyclic)
10,000,000	2.040729	0.705448	0.636719	0.61659

Task 1 - Speedup (different machines)

Due to hardware differences there is a discrepancy between the runtimes:

- CPU speed and core count,
- Cache/memory bandwidth,
- Network/VM overhead



Ran on ryzen 5700x3d (8 cores)

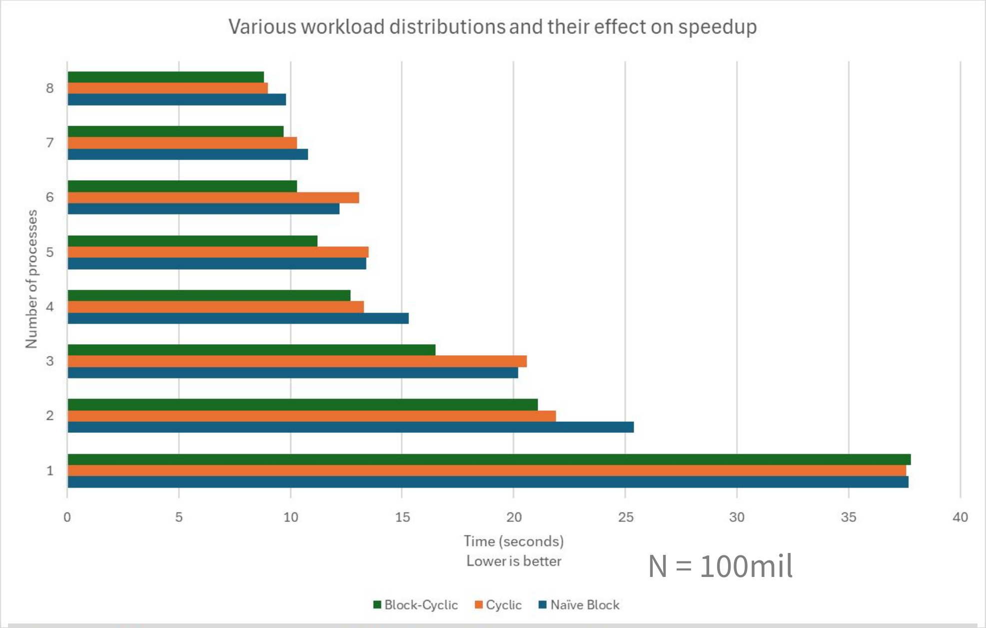
n	Serial	OpenMPI (Block)	OpenMPI (Cyclic)	OpenMPI (Block Cyclic)
10,000,000	2.040729	0.705448	0.636719	0.61659

Ran on i7-1255U with 10 Core(s)

n	Serial	OpenMPI (Block)	OpenMPI (Cyclic)	OpenMPI (Block Cyclic)
10,000,000	5.10985	1.121663	1.684763	1.058977

Great example of how memory bandwidth could demonstrate dramatically different results with i7-1255u even when it has more cores, consuming 92% of memory

Task 1 - Adding Processors Speedup



Amdahl's Law in Distributed Parallel Processing

- This results in Amdahl's Law for Distributed Parallel Processing:

$$S = \frac{1}{s_{comp} + s_{fabric} + \frac{p}{N}} \quad \text{where } s_{comp} + s_{fabric} + p = 1$$

"Will increasing the number of MPI processes always yield higher speed-ups?"

Yes, increasing MPI processes generally yields higher speedups, however there will be diminishing returns the more processes we use due to **Amdahl's Law** and **communication overhead**. There reaches a certain point where the speedup would cap due to serial code that cannot be **parallelized**. Adding more processors hits a limit: communication costs (MPI gathers, bandwidth, sync) grow faster than compute savings

Task 1 - Theoretical Performance

Fully serialised runtime = P time = 2.030952 seconds

Assume fixed workload of all primes <= 10,000,000.

Runtime to send + receive 1 byte of data: 0.000000436 seconds

- Assume to be time of sending **overhead**

Runtime for sending all primes <= 1,000,000: in 0.000949 seconds

- Assume to be total time of sending **all messages**

Loops in serialised portion:

- Sending to number to each node: N * 0.000000436 seconds
- Gathering all results: 0.000949 + [0.000000436 * (N-1)]
- **s_{fabric} time = 0.000949 + [0.000000436 * (2N-1)]**

Let N = 1:

- S_{comp} time = 0
- s_{fabric} time = 0.000949436
- P time = 2.030952

Task 1 - Theoretical Performance

Let N = 1:

- S_{comp} time = 0
- s_{fabric} time = 0.000949436
- P time = 2.030952

s = 0.000949436 / (2.030952 + 0.000949436) =0.00046726479

P = 1 - (s_{comp} + s_{fabric}) = 0.99953273521

Using Amdahl's Law because we are assuming a fixed workload while increasing processors.

S = 1 / 0.00046726479 + (0.99953273521/N)

Max Speedup = 1 / 0.00046726479 = 2140.11417381

N	S
2	1.999
3	2.997
4	3.994
5	4.990

Task 1 - Theoretical vs Actual Speedup

Compare your actual speed against the theoretical speed up. How does the actual speed up compare against the theoretical speed up?

Comparison of Speed-up for Block-Cyclic (most efficient method):

	1	n=2	n=3	n=4	n=5	n=6
Theoretical Speed up	1	1.98019802	2.941176471	3.883495146	4.807692308	5.714285714
Actual Speed up	1.596802297	2.419146184	2.444444444	2.694757472	3.06557377	3.280701754

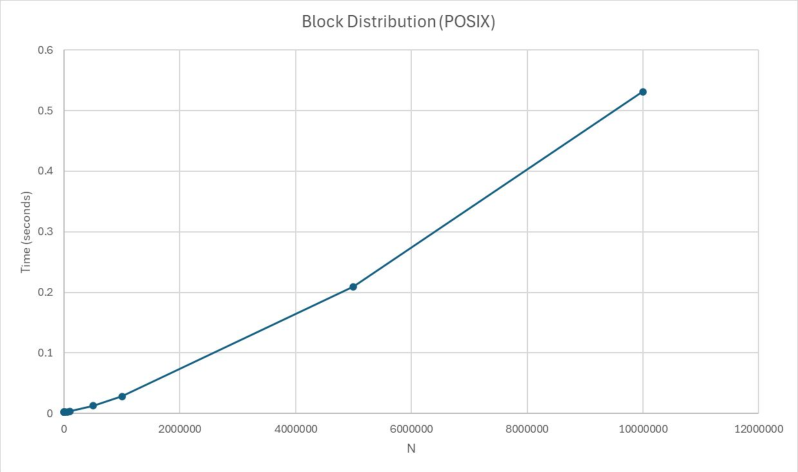
Amdahl's Law in Distributed Parallel Processing

- This results in Amdahl's Law for Distributed Parallel Processing:

$$S = \frac{1}{s_{comp} + s_{fabric} + \frac{p}{N}} \quad \text{where } s_{comp} + s_{fabric} + p = 1$$

Task 1 - MPI vs Posix Threads

n	Block	Cyclic	Block-Cyclic
10	0.006005	0.002181	0.002048
50	0.001854	0.002011	0.002064
100	0.001964	0.001923	0.001889
500	0.002087	0.001973	0.001998
1000	0.001986	0.002076	0.002116
5000	0.002043	0.002083	0.002074
10000	0.002332	0.002301	0.002302
50000	0.003974	0.003818	0.003715
100000	0.005576	0.005505	0.005831
500000	0.023474	0.021614	0.022245
1000000	0.046129	0.044979	0.043745
5000000	0.308528	0.272608	0.273717
10000000	0.768246	0.671559	0.605399

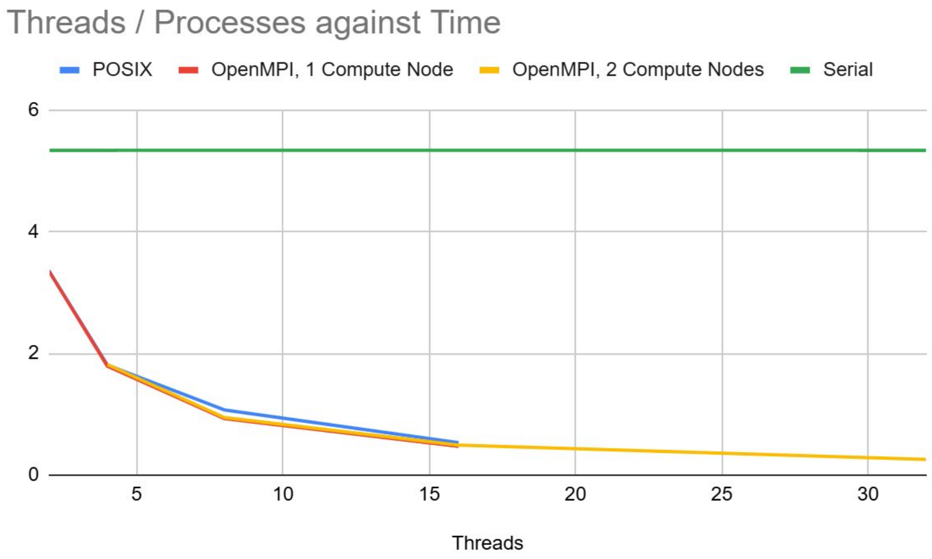


n	Block
10	0.001987
50	0.002061
100	0.002178
500	0.002035
1000	0.002073
5000	0.001944
10000	0.002114
50000	0.002565
100000	0.003506
500000	0.012642
1000000	0.028387
5000000	0.20938
10000000	0.531238

Posix performs **marginally better than MPI overall**. MPI is for distributed memory systems therefore, processes may be spread across different nodes connected by a network, so any communication between them involves sending messages over this network, which incurs latency and bandwidth costs.

Task 2: Cluster

Task 2 - CAAS



Threads	POSIX	OpenMPI, 1 Cor	OpenMPI, 2 Cor	Serial
2	3.362311	3.353395		5.338069
4	1.809572	1.791285	1.816337	5.338069
8	1.071372	0.929238	0.946251	5.338069
16	0.533827	0.473408	0.494263	5.338069
32			0.257218	5.338069

All tested against N = 10m

Task 2 - CAAS

Serial version had much worse performance run through CAAS than locally

POSIX

Speedup = 6.3x

OpenMPI 1 Compute Node

Speedup = 7.08x

OpenMPI 2 Compute Nodes

Speedup = 7.06x

Both MPI versions had larger speedups

MPI version had faster run times with same amount of processes compared to locally